Optimal Clock Synchronization under Different Delay Assumptions

Hagit Attiya[*] Amir Herzberg[†] Sergio Rajsbaum[‡]

April 7, 1994

Abstract

The problem of achieving optimal clock synchronization in a communication network with arbitrary topology and perfect clocks (that do not drift) is studied. Clock synchronization algorithms are presented for a large family of delay assumptions. Our algorithms are modular and consist of three major components. The first component holds for any type of delay assumptions; the second component holds for a large, natural family of local delay assumptions; the third component has to be tailored for each specific delay assumption.

Optimal clock synchronization algorithms are derived for several types of delay assumptions by appropriately tuning the third component. The delay assumptions include lower and upper delay bounds, no bounds at all, and bounds on the difference of the delay in opposite directions. In addition, our model handles systems where some processors are connected by broadcast networks in which every message arrives to all processors at approximately the same time. A composition theorem allows combinations of different assumptions for different links or even for the same link; such mixtures are common in practice.

Our results achieve the best possible precision in each execution. This notion of optimality is stronger than the more common notion of worst case optimality. The new notion of optimality applies to systems where the worst case behavior of any clock synchronization algorithm is inherently unbounded.

Keywords: distributed systems, real-time systems, clock synchronization, message passing systems, networks, optimization, message delay assumptions, precision.

[*] Department of Computer Science, Technion, Haifa 32000, Israel. Supported by grant No. 92-0233 from the United States-Israel Binational Science Foundation (BSF), Jerusalem, Israel, Technion V.P.R.—Argentinian Research Fund and the fund for the promotion of research in the Technion. Email: hagit@cs.technion.ac.il.

[†] IBM T. J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598, USA. Partially supported by DGAPA Projects, National Autonomous University of Mexico (UNAM). Email: amir@yktvmh.bitnet or amir@watson.ibm.com.

[‡] MIT Laboratory for Computer Science, 545 Technology Square, Cambridge, MA 02139, USA. On leave from Instituto de Matemáticas, UNAM, México. Partially supported by DGAPA Projects, UNAM. Email: rajsbaum@theory.lcs.mit.edu.

1 Introduction

In most large-scale distributed systems, processors communicate by message transmission, and do not have access to a central clock. Nonetheless it is useful, and sometimes even necessary, for the processors to obtain some common notion of time. The technique used to attain this notion of time is known as *clock synchronization*. Synchronized clocks are useful for various applications such as control of real-time processes, transaction processing in database systems, and communication protocols. Recently, several software protocols that support clock synchronization in communication networks have been proposed [1, 6, 12, 13, 14]; system designers have been advocating the use of synchronized clocks [9].

The quality of synchronization is measured by its *precision*, i.e., how close together it brings the clocks at different processors.[1] The precision influences the correctness and the efficiency of applications using the synchronized clocks.

The best precision that can be achieved is determined by the timing uncertainty that is inherent in the system. There are two main sources of timing uncertainty in a distributed system. First, local clocks at different processors are independent: they do not start together and may run at different speeds. Second, messages sent between processors incur uncertain delays.

A relatively simple case is when local clocks are accurate, i.e., run at the same speed, and there are upper and lower bounds for the delay on each link. Clock synchronization algorithms for under this assumption, whose precision is optimal in the worst case, are described in [3, 10]. Subsequent work concentrated on clocks that may drift and on fault-tolerance (e.g., [2, 6, 18, 19], see survey in [17]). To achieve high precision, these algorithms require the existence of tight lower and upper bounds on message delay.

However, in real systems it is often the uncertainty of message delay, rather than clock drift, that causes most of the difficulty in synchronizing clocks [12, 6]. Almost every processor in a distributed system has access to a high-quality, very accurate hardware clock; it is not far from reality to assume that local clocks are accurate and have no drift.[2] On the other hand, often there do not exist tight upper and lower bounds on message delay, while there is other relevant information about the delays. For example, in some systems, a bound on the difference between delays in opposite directions is known. This motivated us to revisit the case in which local clocks run at the same speed and have no drift, thus focusing on the impact of message delay uncertainty on clock synchronization.

Our main contribution is a methodology for designing optimal clock synchronization algorithms under a variety of assumptions on message delay uncertainty. The strongest results are

[1] This does not ensure that clocks are close to real time. It is easy to adapt our results to reach this goal if a perfect real time clock is available. Synchronization to real time is often useful, and is achieved by practical protocols which usually deal also with multiple, imperfect real time clocks, e.g., the Internet NTP [12].

[2] To deal with the small drift which does exist, the clock synchronization mechanism is invoked periodically; e.g. Kopetz and Ochsenreiter [6].

obtained for a natural family of local delay assumptions; this family includes all the assumptions studied in previous theoretical work. A delay assumption is *local* if it is specified for a pair of processors, for example, processors connected by a link. We show that in this case, a clock synchronization algorithm can be obtained by considering each pair separately. This simplifies the analysis substantially, and allows different pairs of processors to satisfy different assumptions. Furthermore, we prove a composition theorem which allows us to combine several local delay assumptions. For example, it is possible that for some pair of processors there will be upper and lower bounds on the message delay in each direction, as well as a bound on the difference between message delays in opposite directions.

The basic difficulty of clock synchronization stems from the existence of two different system executions in which all processors have the same views. The tightness of the achievable synchronization depends on how "far away" in real time can such executions be. We capture this notion quantitatively as the *maximal shift* between processors in a given execution. Using this notion, we partition the design of a clock synchronization algorithm into three stages:

1. Optimal clock synchronization by computing corrections to the local clocks from the maximal shifts. This stage is valid for any kind of message delay assumptions.
2. Computation of maximal shifts from *maximal local shifts*, which depend only on the views of pairs of processors. This stage is valid for message delay assumptions which are local.
3. Computation of the maximal local shifts from the local views. This depends on the specific message delay assumptions.

Our methodology yields optimal clock synchronization algorithms for a variety of delay assumptions, by adapting the third stage above. In particular, we show how to compute maximal local shifts for the following message delay assumptions:

1. upper and lower bounds on delays are known;
2. only lower bounds on the delays are known;
3. no bounds are known;
4. bound on the difference on the delay in opposite directions; and
5. there is a bound on the difference in the times when different processors receive a multicast message.

Most previous formal work on deterministic clock synchronization addressed only the first assumption. However, many practical systems are better modeled by the other assumptions. The second assumption follows an observation of [1] that in many actual links, there is some minimal delay (e.g., due to the actual transmission rate and processing time). The fourth assumption follows experimental results (cf. [12]), showing that message delays in opposite directions of a bi-directional link are usually very close. The fifth assumption is useful for broadcast networks, such as these used in many local area networks; this is the assumption used in [4, 16].

Our composition theorem implies that our algorithms apply to systems where the same pair of processors satisfies several different delay assumptions. Such mixtures are quite common in practical, heterogeneous systems. For example, there are systems in which several local area (broadcast) networks are connected by bridges or (long distance) links.

Our work extends the results of Halpern, Megiddo and Munshi [3]. Halpern et al. use linear programming techniques which do not illuminate the inherent difficulties of synchronizing clocks. We believe that our work gives a more precise understanding of the problem, explicitly showing what are the requirements of each step and thereby facilitating adaptation to other delay assumptions. Their results are a special case of the general methodology developed here, in which exactly one message is sent on each link, and upper and lower bounds on delays are known. In fact, the algorithm we obtain for this specific setting is essentially the one in [3].

Previous definitions of optimal clock synchronization were based on the worst (largest) difference between clocks of two processors in any execution. For some of the assumptions that we study in this paper, e.g., when no upper bounds on the delays are known, this worst case is inherently unbounded. Moreover, as already stated in [3], we would like to award algorithms that exploit favorable conditions, and achieve precision that is as good as can be in each specific instance. We give a precise definition of optimality for each specific execution, and show that it is achieved by our algorithms (and hence also by the algorithm of [3]).

When trying to crystallize these ideas, it turned out that the decision of which messages to send should be separated from the method for adjusting the clocks based on the local message histories. Our framework shows how to optimally adjust the clocks, given any set of local message histories. The decision of which messages to send, to whom, when, etc., can therefore take other considerations into account, e.g., message traffic optimization, and is left outside of the scope of this paper.

The rest of this paper is organized as follows. In Section 2, the model and the clock synchronization problem are defined. Section 3 presents the general clock synchronization algorithm, and proves that it achieves optimal precision; the algorithm is independent of the message delay assumptions. In Section 4, it is shown how to compute the inputs needed for the general clock synchronization algorithm, when some local information about the views of the processors is given; the computation is valid for local systems. In Section 5, it is shown how to compute the required information on the views for several specific delay assumptions. Conclusions and open questions appear in Section 6.

2 Definitions

2.1 Defining Optimal Precision

We would like a clock synchronization algorithm to obtain the best possible precision, that is, to bring the logical clocks as close to each other as possible. However, it is not obvious how to compare the precision achieved by different algorithms, and how to define optimality.

An elegant solution is to evaluate a clock synchronization algorithm by the worst (largest) precision achieved in any of its executions. This worst case interpretation follows the tradition of worst case complexity analysis of algorithms.

This definition has two drawbacks. First, like any definition that concentrates on the worst case, it does not award algorithms that behave well in other cases. An algorithm that is optimal under this definition can be very inefficient in executions where the delays are favorable. Second, worst case analysis is meaningful only if the worst case precision is bounded. However, in many important cases, the worst case precision can be easily shown to be unbounded, e.g., when there are no upper bounds on message delay.

We believe a more refined notion of optimality is called for. Intuitively, an optimal algorithm is one whose precision, in every execution, is not larger than the precision of any other algorithm in an execution where the message delivery system "acts the same."

Formalizing this idea is not so simple. The major difficulty is finding a satisfying definition for executions where the message delivery system acts the same. The problem is that the properties of the execution are determined by the interaction between the message delivery system and the algorithm. The algorithm controls the execution by deciding when to send messages, while the message delivery system controls the execution by determining their delay.[3] It is difficult to isolate the effect on the execution determined by the message delivery system. Such isolation is necessary in order to compare executions of a given algorithm to executions of other algorithms where the message delivery system is equally adversarial. A definition is too strong if it compares an execution of one algorithm with an execution of another algorithm in which message delays are unfairly favorable. Conversely, a definition is too weak if executions with the same message delivery policy are not compared. We sidestep this problem by noticing that the construction of a clock synchronization algorithm has two aspects. First, the design of the interactive part where the processors send messages. Second, calculating corrections using the views of the processors that were obtained during the interactive part. In this paper, we do not address the first aspect. We assume that we have a set of views, one for each processor, and we ask how to compute optimal corrections for this set of views.

[3] This is not merely a formal issue: from a practical point of view, if an algorithm sends too many messages in a short period of time, the network becomes congested and delays are long and highly variant.

2.2 Model of Computation

Here we formalize the behavior of the interactive part of a clock synchronization algorithm, which is a distributed algorithm running on a network. The distributed algorithm decides when to send messages, while the network decides when to deliver the messages. The interplay between the distributed algorithm and the network generates a set of executions. The result of this execution is the input to the clock synchronization function.

We consider a set $P = \{p_1, \ldots, p_n\}$ of *processors*. With each processor $p \in P$ we associate a (local) clock. The clock cannot be modified by the processor. Processors do not have access to real time; each processor obtains its only information about time from its clock and from messages sent by other processors. The clock is represented by a local time component, which is a real number. In the sequel, the term *clock time* refers to the local time component of the processor, while the term *real time* refers to the absolute time as measured by an outside observer. In this work we assume that clocks do no drift, i.e., that they run at the same rate as real time, but they are not necessarily synchronized with each other.

Below we list the *events* which can occur at processor p, together with an informal explanation:

Message receive events– receive(p, m, q), for all messages m and processors q: processor p receives message m from processor q.

Message send events– send(p, m, q), for all messages m and processors q: processor p sends message m to processor q.

Timer set events– timer-set(p, T), for all clock times T: processor p sets a timer to go off when its clock reads T.

Timer events– timer(p, T), for all clock times T: a timer that was set for time T on p's clock goes off.

Start events– start$(p, 0)$: p starts executing the algorithm, with the initial value of its clock being 0.

The message receive, timer and start events are *interrupt events.*

Each processor is modeled as an automaton with a (possibly infinite) set of states, including an initial state, and a transition function. Each interrupt event causes an application of the transition function. The transition function is a function from states, clock times, and interrupt events to states, sets of message-send events, and sets of timer-set events (for subsequent clock times). That is, the transition function takes as input the current state, clock time, and interrupt event (which is the receipt of a message from another processor or a timer going off), and produces a new state, a set of messages to be sent, and a set of timers to be set for the future.

A *step* of p is a tuple (s, T, i, s', M, TS), where s and s' are states, T is a clock time, i is an interrupt event, M is a set of message-send events, TS is a set of timer-set events, and s', M, and TS are the result of p's transition function acting on s, T, and i. A *history* π of a processor p is a mapping associating to each number from $\Re$ (real time) a finite sequence (possibly empty) of steps such that:

1. For each real time t, there is only a finite number of times $t' < t$ such that the corresponding sequence of steps is nonempty (thus the concatenation of all the sequences in real-time order is a sequence);
2. The interrupt event in the first step of the history is a start event, and the old state in the first step is p's initial state; let S_π be the real time of the start event;
3. There are no other start events and the old state of each subsequent step is the new state of the previous step;
4. For each real time t, the clock time component T of each step in the corresponding sequence is equal to $t - S_\pi$ (thus, the clock time of the start event of p is 0);
5. For each real time t, in the corresponding sequence there is at most one timer event and it is ordered after all other events; and
6. A timer is received by p at clock time T if and only if p has previously set a timer for T.

An *execution* is a set of histories, one for each processor p in P, such that there is a one-to-one correspondence between the messages received by q from p and the messages sent by p to q, for any processors p and q. (To simplify our discussion, we assume that messages are unique, so this correspondence is uniquely defined.) We use the message correspondence to define the *delay* of a message m received in execution α, denoted $\mathrm{d}_\alpha(m)$, to be the real time of receipt minus the real time of sending. When α is clear from the context, we simply write $\mathrm{d}(m)$.

Let $S_{\alpha,p} = S_\pi$ where π is p's history in α; that is, $S_{\alpha,p}$ is the real time of the start event of processor p in α.

Note that the message delivery system is not explicitly modeled. The requirements from an execution state that messages are delivered without duplication, and that the system does not generate messages; the system can reorder or lose messages. A system $(P, \mathcal{A})$ is a set of processors P and a set of executions $\mathcal{A}$, called *admissible executions.* For example, $\mathcal{A}$ may allow communication only between specific pairs of processors connected by a link.

The cornerstone of our definitions and proofs is the notion of equivalent executions. Informally, two executions are equivalent if they are indistinguishable to the processors; only an outside observer who has access to the real time can tell them apart.

To formalize this notion, define the *view* of processor p in history π to be the concatenation of the sequences of steps in π, in real-time order. (Note that the view includes the clock times.)

The real times of occurrence are not represented in the view. Let α be an execution, and let π be p's history in α. The view of p in α is the view of p in π and is denoted $\alpha|p$. Two executions α and α' are *equivalent*, denoted $\alpha \equiv \alpha'$, if for every processor $p \in P$, $\alpha|p = \alpha'|p$.

2.3 The Clock Synchronization Problem

The goal of a clock synchronization algorithm is to bring the clocks of the processors to be as close to each other as possible, while keeping the clocks' values with the progress of real time. Intuitively, each processor maintains a *logical clock*, which "corrects" the value of the local clock. Since the logical clock is required not to drift from the progress of real time, it is straightforward to see that the logical clock must be the local clock plus some correction factor. Thus, the goal of a clock synchronization algorithm is to compute a correction for each processor, such that for any two processors, the values of the local clocks (at the same real time) plus the respective corrections are close.

Specifically, a *clock synchronization algorithm* is a function from a set of n views to a vector of n real numbers, called *corrections*. Given a clock synchronization algorithm f and an execution α, we abuse notation and denote by $f(\alpha)$ the vector obtained by applying f to the n views in α; we denote by $f(\alpha, p)$ the component of $f(\alpha)$ that corresponds to p. Since a clock synchronization algorithm depends only on the views, we have:

Claim 2.1 *If $\alpha \equiv \alpha'$ then $f(\alpha) = f(\alpha')$.*

Recall that at any real time t, the clock value of p is $t - S_{\alpha,p}$. Given a clock synchronization function f, the corrected local time of p in α is $t - S_{\alpha,p} + f(\alpha, p)$. Therefore, $|(S_{\alpha,p} - f(\alpha,p)) - (S_{\alpha,q} - f(\alpha, q))|$ is the difference between the corrected local times of p and q in α.

To capture the precision achieved by some vector of corrections $\vec{x} = \langle x_1, \ldots, x_n \rangle$ denote $\rho(\alpha, \vec{x}) = \max_{p,q \in P} |(S_{\alpha,p} - x_p) - (S_{\alpha,q} - x_q)|$. That is, $\rho(\alpha, \vec{x})$ is the largest discrepancy between two clocks of different processors after they are corrected.

Because the computation of the corrections does not distinguish between equivalent executions, we measure the precision for a specific execution α, by considering the worst discrepancy achieved for all the executions equivalent to α. Let $\mathcal{A}$ be the set of admissible executions. Formally, for any execution $\alpha \in \mathcal{A}$, the inherent precision achieved by a vector of corrections $\vec{x}$ is

$$\bar{\rho}(\alpha, \vec{x}) = \sup\{\rho(\alpha', \vec{x}) : \alpha' \equiv \alpha \text{ and } \alpha' \in \mathcal{A}\} .$$

Definition 2.1 *A clock synchronization algorithm f* computes optimal corrections *if for every admissible execution α and every vector of corrections $\vec{x}$, $\bar{\rho}(\alpha, f(\alpha)) \leq \bar{\rho}(\alpha, \vec{x})$.*

We call $\bar{\rho}(\alpha, f(\alpha))$ the *precision* of f on α, and use the shorthand $\bar{\rho}(\alpha, f)$.

3 A General Clock Synchronization Algorithm

As mentioned before, the basic difficulty of computing corrections is the fact that there may be two admissible executions α and α' in which all processors have the same views. Clearly, the tightness of the achievable synchronization depends on how "far away" in real time can α' be from α. We formally quantify this idea by defining the maximal shift between processors in a given execution. We show that if estimates of the maximal shifts are available, then there exists a function that computes optimal corrections. This is done by showing a lower bound for the precision which depends only on the maximal shifts. Then we show that this bound is tight by presenting a method for computing corrections that achieves this value as its precision. In subsequent sections we show how to estimate the maximal shifts for specific systems.

3.1 Maximal Shifts

Consider two equivalent executions α and α'. It follows that for any $p \in P$, the sequence of steps in α' is equal to the sequence of steps in α, except that p executes its steps at different real times. Since the clocks have no drift, it follows that the difference between the real times of occurrence of a step in α and the corresponding step in α' is *fixed*, independently of the step. This implies that α' can be obtained by "shifting" the steps of the processors in α.

In the rest of this section, we formalize this notion of *shifting* and study its properties. This technique was originally introduced by Lundelius and Lynch [10] to prove lower bounds on the precision achieved by clock synchronization algorithms in complete graphs.

Formally, given a history π of processor p and a real number s, a new history $\pi' = \mathit{shift}(\pi, s)$ is defined by $\pi'(t) = \pi(t + s)$ for all t. That is, all tuples are shifted earlier in π' by s if s is positive, and later by $-s$ if s is negative. Clearly, the views do not change with shifting. Furthermore:

Lemma 3.1 (Lundelius and Lynch [10]) *Let π be a history of processor p and let s be a real number. Then $\pi' = \mathit{shift}(\pi, s)$ is a history of p and $S_{\pi'} = S_\pi - s$.*

Let α and α' be two equivalent executions such that each processor $p \in P$ is shifted in α' w.r.t. α by s_p; the *vector of shifts* of α' w.r.t. α is the vector $S = \langle s_1, \ldots, s_n \rangle$. That is, execution α' was obtained by replacing p's history in α, π, with $\mathit{shift}(\pi, s_p)$, for each $p \in P$, and by retaining the same correspondence between message send and receive events. (Technically, the correspondence is redefined so that a pairing in α that involves the event for p at time t, in α' involves the event for p at time $t - s_p$.) We denote α' by $\mathit{shift}(\alpha, S)$. Clearly:

Claim 3.2 *Let $\alpha' = \mathit{shift}(\alpha, \langle s_1, \ldots, s_n \rangle)$. For every message m received by processor p from q in α, $\mathrm{d}_{\alpha'}(m) = \mathrm{d}_\alpha(m) + (s_q - s_p)$.*

Note that if $\alpha \equiv \alpha'$ then there exists a vector of shifts S such that $\alpha' = \mathit{shift}(\alpha, S)$.

We now formalize the notion of how "far away" can a processor be shifted w.r.t. another processor. Fix a system $(P, \mathcal{A})$, and let $\alpha \in \mathcal{A}$. We say that s is an *admissible shift of q w.r.t. p in α*, if there exists a vector of shifts $S = \langle s_1, \ldots, s_n \rangle$ with $s_q - s_p = s$, such that $\alpha' = \mathit{shift}(\alpha, S)$ is in $\mathcal{A}$. Define

$$\mathrm{ms}_\alpha(p, q) = \sup\{s : s \text{ is an admissible shift of } q \text{ w.r.t. } p \text{ in } \alpha\}.$$

This is the *maximal shift* of q w.r.t. p in α; that is, how far away can q be shifted from p while retaining the admissibility of the execution. Since 0 is obviously an admissible shift of q w.r.t. p in α, it follows that $\mathrm{ms}_\alpha(p, q) \geq 0$.

Claim 3.3 *Let $\alpha \in \mathcal{A}$ and let $\alpha' \equiv \alpha$. If $\alpha' \in \mathcal{A}$, then $S_{\alpha',p} - S_{\alpha',q} \leq S_{\alpha,p} - S_{\alpha,q} + \mathrm{ms}_\alpha(p, q)$, for any two processors p and q.*

Proof: Since $\alpha' \equiv \alpha$ it follows that $\alpha' = \mathit{shift}(\alpha, S)$ for some vector of shifts $S = \langle s_1, \ldots, s_n \rangle$. Fix a pair of processors p and q. Since $\alpha' \in \mathcal{A}$ it follows that $s_q - s_p \leq \mathrm{ms}_\alpha(p, q)$. The claim follows since $S_{\alpha',p} = S_{\alpha,p} - s_p$ and $S_{\alpha',q} = S_{\alpha,q} - s_q$. ⊠

3.2 The Lower Bound

Fix a system $(P, \mathcal{A})$, an admissible execution α, and a clock synchronization algorithm f. The following lemma relates the maximal shift and the attainable precision.

Lemma 3.4 *For any pair of processors p and q, $\bar{\rho}(\alpha, f) \geq S_{\alpha,p} - f(\alpha, p) - S_{\alpha,q} + f(\alpha, q) + \mathrm{ms}_\alpha(p, q)$.*

Proof: Let s be an arbitrary admissible shift of q w.r.t. p in α. Let $\alpha' \in \mathcal{A}$ be an execution such that $\alpha' \equiv \alpha$ and q is shifted w.r.t. p by s. Then,

$$\begin{aligned}
\bar{\rho}(\alpha, f) \geq \rho(\alpha', f(\alpha')) &\geq S_{\alpha',p} - f(\alpha', p) - S_{\alpha',q} + f(\alpha', q) \\
&= S_{\alpha,p} - f(\alpha', p) - S_{\alpha,q} + f(\alpha', q) + s \\
&= S_{\alpha,p} - f(\alpha, p) - S_{\alpha,q} + f(\alpha, q) + s \qquad \text{(by Claim 2.1)}\ .
\end{aligned}$$

Since s was chosen arbitrarily,

$$\bar{\rho}(\alpha, f) \geq S_{\alpha,p} - f(\alpha, p) - S_{\alpha,q} + f(\alpha, q) + \mathrm{ms}_\alpha(p, q).$$

⊠

The expression defined next will turn out to be a lower bound on the precision that can be achieved in α. Let θ be a cyclic sequence of processors, that is, $\theta = p_0, p_1, \ldots, p_{k-1}, p_k$, where $p_k = p_0$; processors p_i and p_{i+1} are not necessarily adjacent in the graph. Denote $|\theta| = k$ and $\mathrm{ms}_\alpha(\theta) = \sum_{i=0}^{k-1} \mathrm{ms}_\alpha(p_i, p_{i+1})$. Let $A_\alpha(\theta) = \mathrm{ms}_\alpha(\theta)/|\theta|$, and define

$$A_\alpha^{\max} = \max\{A_\alpha(\theta) : \theta \text{ is a cyclic sequence of processors}\}.$$

Theorem 3.5 *For any clock synchronization algorithm f, $\bar{\rho}(\alpha, f) \geq A_\alpha^{\max}$.*

Proof: Let $\theta = p_0, \ldots, p_k$ be an arbitrary cyclic sequence of processors (where $p_k = p_0$). By Lemma 3.4,

$$\bar{\rho}(\alpha, f) \geq S_{\alpha,p_i} - f(\alpha, p_i) - S_{\alpha,p_{i+1}} + f(\alpha, p_{i+1}) + \mathrm{ms}_\alpha(p_i, p_{i+1}) \ ,$$

for every i, $0 \leq i \leq k-1$. Summing over all the consecutive processors in θ, we have

$$k \cdot \bar{\rho}(\alpha, f) \geq \sum_{i=0}^{k-1} \left[S_{\alpha,p_i} - f(\alpha, p_i) - S_{\alpha,p_{i+1}} + f(\alpha, p_{i+1}) + \mathrm{ms}_\alpha(p_i, p_{i+1})\right] .$$

Clearly,

$$\sum_{i=0}^{k-1} \left[S_{\alpha,p_i} - f(\alpha, p_i) - S_{\alpha,p_{i+1}} + f(\alpha, p_{i+1})\right] = 0 \ ,$$

and hence,

$$\bar{\rho}(\alpha, f) \geq \frac{1}{k} \sum_{i=0}^{k-1} \mathrm{ms}_\alpha(p_i, p_{i+1}) = A_\alpha(\theta) \ ,$$

as needed. ∎

3.3 The Upper Bound

We now show the converse direction, i.e., that there exists a clock synchronization algorithm f with $\bar{\rho}(\alpha, f) = A_\alpha^{\max}$, for every α, provided certain estimates can be computed from the views. By Theorem 3.5, no other clock synchronization algorithm can achieve better precision. Hence our clock synchronization algorithm computes optimal corrections, in the sense of Definition 2.1.

Clearly, if the values of $\mathrm{ms}_\alpha(p, q)$ are known then it is possible to calculate $A_\alpha^{\max}$. As we shall see, computing $A_\alpha^{\max}$ is the crux of computing optimal corrections. However, since the views do not include the actual message delays, it is not clear what is the set of equivalent executions; hence, in general, it is impossible to compute the values of $\mathrm{ms}_\alpha(p, q)$ from the views. Below we show that it suffices to have only estimates on $\mathrm{ms}_\alpha(p, q)$. In the next sections, we show how to obtain these estimates for specific systems.

Define the *estimated maximal global shift* to be $\tilde{\text{ms}}_\alpha(p,q) = \text{ms}_\alpha(p,q) + S_{\alpha,p} - S_{\alpha,q}$. The next lemma is the key to replacing ms_α with the estimates $\tilde{\text{ms}}_\alpha$ in the calculation of $A_\alpha^{\max}$. The lemma shows that the maximum average cycle weight with respect to the actual maximal shifts is equal to the maximum average cycle weight with respect to the estimates. Specifically, for any cyclic sequence of processors $\theta = p_0, \ldots p_k$ (where $p_k = p_0$), let $\tilde{\text{ms}}_\alpha(\theta) = \sum_{i=0}^{k-1} \tilde{\text{ms}}_\alpha(p_i, p_{i+1})$. Also, let $\tilde{A}_\alpha(\theta) = \tilde{\text{ms}}_\alpha(\theta)/|\theta|$, and define

$$\tilde{A}_\alpha^{\max} = \max\{\tilde{A}_\alpha(\theta) : \theta \text{ is a cyclic sequence of processors}\}.$$

Lemma 3.6 $A_\alpha^{\max} = \tilde{A}_\alpha^{\max}$.

Proof: Consider any cyclic sequence of processors $\theta = p_0, \ldots, p_k$ (where $p_k = p_0$), and sum the estimates around the cycle:

$$\sum_{i=0}^{k-1} \tilde{\text{ms}}_\alpha(p_i, p_{i+1}) = \sum_{i=0}^{k-1} [\text{ms}_\alpha(p_i, p_{i+1}) + S_{\alpha,p_i} - S_{\alpha,p_{i+1}}] \ .$$

However, the values for S_{α,p_i} cancel each other, and we get $\sum_{i=0}^{k-1} \text{ms}_\alpha(p_i, p_{i+1})$, which is $\text{ms}_\alpha(\theta)$. Since this holds for every cyclic sequence of processors, it holds also for any critical sequence θ where A_α is maximized, i.e., where $A_\alpha(\theta) = A_\alpha^{\max}$. ■

Thus, we have the following function SHIFTS for computing corrections given inputs $\tilde{\text{ms}}_\alpha(p,q) = \text{ms}_\alpha(p,q) + S_{\alpha,p} - S_{\alpha,q}$, for every pair of processors p and q.

1. Compute $A_\alpha^{\max}$ (by computing $\tilde{A}_\alpha^{\max}$).
2. Select an arbitrary root processor r. The correction for each processor $p \in P$ is $\text{dist}_w(r,p)$—the distance in the (complete) graph relative to the weights $w(p,q) = A_\alpha^{\max} - \tilde{\text{ms}}_\alpha(p,q)$.

The value of $\tilde{A}_\alpha^{\max}$ can be computed in Step 1 by using an algorithm of Karp [5], that runs in $O(n^3)$ time. By Lemma 3.6, this is equivalent to computing $A_\alpha^{\max}$. By definition, $A_\alpha^{\max} \geq A_\alpha(\theta) = \text{ms}_\alpha(\theta)/|\theta|$, for any cycle θ. Therefore,

$$\sum_{\langle p,q\rangle \in \theta} (A_\alpha^{\max} - \tilde{\text{ms}}_\alpha(p,q)) = |\theta| A_\alpha^{\max} - \tilde{\text{ms}}_\alpha(\theta) \geq 0.$$

This implies that there are no negative weight cycles in the complete graph with the weights $w(p,q) = A_\alpha^{\max} - \tilde{\text{ms}}_\alpha(p,q)$. Thus, the distances can be computed in Step 2.

Theorem 3.7 *The function* SHIFTS *computes optimal corrections with precision $A_\alpha^{\max}$ in each execution α.*

Proof: Denote by $f(\alpha)$ the vector of corrections computed by SHIFTS when given $\tilde{\text{ms}}_\alpha(p,q)$. We will show that $\bar{\rho}(\alpha,f) \leq A_\alpha^{\max}$; it follows from Theorem 3.5 that these are optimal corrections.

To prove that $\bar{\rho}(\alpha,f) \leq A_\alpha^{\max}$ we need to show that $\rho(\alpha', f(\alpha')) \leq A_\alpha^{\max}$ for any admissible execution $\alpha' \equiv \alpha$. It suffices to show that for every pair of processors p and q,

$$S_{\alpha',p} - f(\alpha',p) - S_{\alpha',q} + f(\alpha',q) \leq A_\alpha^{\max} .$$

Fix some pair of processors p and q. By the definition of the function SHIFTS, $f(\alpha,q) = \text{dist}_w(r,q)$ and $f(\alpha,p) = \text{dist}_w(r,p)$, relative to the weights $w(p,q) = A_\alpha^{\max} - \tilde{\text{ms}}_\alpha(p,q)$. Thus

$$f(\alpha,q) - f(\alpha,p) = \text{dist}_w(r,q) - \text{dist}_w(r,p) \leq w(p,q) = A_\alpha^{\max} - \tilde{\text{ms}}_\alpha(p,q) .$$

Adding $\tilde{\text{ms}}_\alpha(p,q) - \text{ms}_\alpha(p,q)$ to both sides, we get

$$\tilde{\text{ms}}_\alpha(p,q) - \text{ms}_\alpha(p,q) + f(\alpha,q) - f(\alpha,p) \leq A_\alpha^{\max} - \text{ms}_\alpha(p,q) .$$

By the definition of estimated maximal shifts, $\tilde{\text{ms}}_\alpha(p,q) - \text{ms}_\alpha(p,q) = S_{\alpha,p} - S_{\alpha,q}$. Hence

$$S_{\alpha,p} - S_{\alpha,q} + f(\alpha,q) - f(\alpha,p) + \text{ms}_\alpha(p,q) \leq A_\alpha^{\max} .$$

Since $\alpha \equiv \alpha'$, Claim 2.1 implies that $f(\alpha) = f(\alpha')$, and thus

$$S_{\alpha,p} - S_{\alpha,q} + f(\alpha',q) - f(\alpha',p) + \text{ms}_\alpha(p,q) \leq A_\alpha^{\max} .$$

By Claim 3.3, $S_{\alpha',p} - S_{\alpha',q} \leq S_{\alpha,p} - S_{\alpha,q} + \text{ms}_\alpha(p,q)$, and thus,

$$S_{\alpha',p} - f(\alpha',p) - S_{\alpha',q} + f(\alpha',q) \leq A_\alpha^{\max} ,$$

as needed. ▨

So far, we have reduced the problem of designing an optimal clock synchronization algorithm to the problem of finding the estimates $\tilde{\text{ms}}_\alpha$ of maximal shifts. Given such estimates, the clock synchronization problem can be solved by computing the function SHIFTS. The next two sections show how to calculate $\tilde{\text{ms}}_\alpha$.

4 Calculating Estimates in Local Systems

In the previous section, we presented a function for computing optimal corrections, which relies on estimates $\tilde{\text{ms}}(p,q)$ of the maximal shifts for each pair of processors p and q. We next show how to compute these estimates in the natural class of local systems. Intuitively, in local systems the delays of messages sent to a pair of processors, e.g., along edges connecting them, do not depend on the delays of messages sent to other processors.

For local systems, estimates $\tilde{\text{ms}}(p,q)$ can be computed in two steps. In the first step, local (pairwise) estimates $\tilde{\text{mls}}(p,q)$ are computed. In the second step, the desired global estimates $\tilde{\text{ms}}(p,q)$ are produced by combining the local estimates. In this section we deal only with the second step, i.e., we show how to compute global estimates from local estimates. In the next section, we compute the local estimates based on the views for several specific systems.

In order to design a clock synchronization algorithm for a specific local system, only the calculation of local estimates needs to be modified. As illustrated by the particular systems discussed in the next section, this calculation handles each pair of processors separately. This significantly simplifies reasoning, and allows us to deal with combinations of several assumptions on the same or on different edges, as we show at the end of this section.

4.1 Local Systems

Informally, a system is local if its admissible executions can be expressed as the intersection of sets of executions, each set restricting only the views of a specific pair of processors.

Definition 4.1 *A set of executions $\mathcal{A}_{p,q}$ is* local to p and q *provided that for every execution $\alpha \in \mathcal{A}_{p,q}$, if there exists an execution α' such that $\alpha'|q = \alpha|q$ and $\alpha'|p = \alpha|p$, then $\alpha' \in \mathcal{A}_{p,q}$.*

Note that this implies that $\mathcal{A}_{p,q}$ allows arbitrary shifts as long as both p and q are shifted by the same amount. That is:

Claim 4.1 *Assume $\mathcal{A}_{p,q}$ is local to p and q. Let $\alpha \in \mathcal{A}_{p,q}$ and let $S = \langle s_1, \ldots, s_n \rangle$ be a vector of shifts such that $s_p = s_q$. Then shift$(\alpha, S) \in \mathcal{A}_{p,q}$.*

Let $\alpha \in \mathcal{A}_{p,q}$. We say that s is a *locally admissible shift of q* w.r.t. p in α if there exists a vector of shifts $S = \langle s_1, \ldots, s_n \rangle$, such that $s_q - s_p = s$ and *shift*$(\alpha, S) \in \mathcal{A}_{p,q}$. We have:

Claim 4.2 *Assume $\mathcal{A}_{p,q}$ is local to p and q, and consider an execution $\alpha \in \mathcal{A}_{p,q}$. A value s is a locally admissible shift of q w.r.t. p in α if and only if for every vector of shifts $S = \langle s_1, \ldots, s_n \rangle$ with $s_q - s_p = s$, shift$(\alpha, S) \in \mathcal{A}_{p,q}$.*

Define the *maximal local shift of q w.r.t. p in α* to be

$$\text{mls}_\alpha(p,q) = \sup\{s : s \text{ is a locally admissible shift of } q \text{ w.r.t. } p \text{ in } \alpha\}.$$

Intuitively, $\text{mls}_\alpha(p,q)$ is the maximal possible shift of q w.r.t. p in α, when the admissibility of processors other than p and q need not be preserved.

We usually leave the sets $\mathcal{A}_{p,q}$ unspecified, when they are clear from the context. Later, when we want to specify different assumptions on the same pair of processers, we consider more than one local set of executions for this pair. In this case, to distinguish between different local sets for processors p and q we will use $\mathcal{A}_{p,q}$, $\mathcal{A}'_{p,q}$, $\mathcal{A}''_{p,q}$, etc.

Definition 4.2 *A set of admissible executions $\mathcal{A}$ is* local *if there exist local sets $\mathcal{A}_{p,q}$, such that $\mathcal{A} = \bigcap_{p,q \in P} \mathcal{A}_{p,q}$, and for every p and q, $\mathcal{A}_{p,q}$ is local to p and q.*

Claim 4.2 implies:

Claim 4.3 *Assume $\mathcal{A}$ is local. Let $S = \langle s_1, \ldots, s_n \rangle$ be a vector of shifts and let $\alpha \in \mathcal{A}$. Then shift$(\alpha, S) \in \mathcal{A}$ if and only if $s_q - s_p$ is a locally admissible shift of q w.r.t. p in α, for every pair of processors p and q.*

Notice that $\mathrm{mls}_\alpha(p,q) \geq \mathrm{ms}_\alpha(p,q)$ and $\mathrm{mls}_\alpha(p,q) \geq 0$. We say that $\mathrm{mls}_\alpha(p,q)$ is a *local shift*, while $\mathrm{ms}_\alpha(p,q)$ is a *global shift*.

Note that $\mathrm{mls}_\alpha(p,q)$ may differ from $\mathrm{mls}_\alpha(q,p)$. However, if *shift*$(\alpha, \langle s_1, \ldots, s_n \rangle)$ is in $\mathcal{A}_{p,q}$, then $s_q - s_p$ is an locally admissible shift of q w.r.t. p and $s_p - s_q$ is a locally admissible shift of p w.r.t. q. Thus, if s is a locally admissible shift of q w.r.t. p in α, then $-s$ is a locally admissible shift of p w.r.t. q in α.

Throughout the rest of the section, we assume that $\mathcal{A}$ is local with respect to some local sets of executions. Hence, the locally admissible shifts are defined. Furthermore, we assume that the locally admissible shifts have the following property which holds in most natural applications.

Assumption 1 *Let x and y be two numbers such that $x < y$. For every two processors p, q, and every $\alpha \in \mathcal{A}$, if x and y are locally admissible shifts of q w.r.t. p in α, then every value $z \in [y, x]$ is a locally admissible shift of q w.r.t. p in α.*

4.2 From Local Shifts to Global Shifts

Our goal is to compute global estimates $\tilde{\mathrm{ms}}_\alpha(p,q)$ from local estimates $\tilde{\mathrm{mls}}_\alpha(p,q)$. In this section, as a first step in this direction, we show how to obtain maximal global shifts $\mathrm{ms}_\alpha(p,q)$ from maximal local shifts $\mathrm{mls}_\alpha(p,q)$. This also shows how to derive a lower bound on the precision of clock synchronization from a lower bound on the precision of each edge independently.

Let α be an admissible execution. Denote by $\mathrm{dist}_{w'}(p,q)$ the distance from p to q in the graph G relative to the weights $w'(p,q) = \mathrm{mls}_\alpha(p,q)$.

Lemma 4.4 *For any pair of processors p and q, $dist_{w'}(p,q) \leq \mathrm{ms}_\alpha(p,q)$.*

Proof: Assume, by way of contradiction, that for some pair of processors p and q, $\mathrm{dist}_{w'}(p,q) > \mathrm{ms}_\alpha(p,q)$. Thus there exists some value s, $\mathrm{ms}_\alpha(p,q) < s < \mathrm{dist}_{w'}(p,q)$ (we assume that all the distances $\mathrm{dist}_{w'}$ are finite; it is not difficult to generalize the result for the case of infinite distances). We show that s is a (globally) admissible shift of q w.r.t. p in α, which contradicts the definition of $\mathrm{ms}_\alpha(p,q)$.

Since $s > \mathrm{ms}_\alpha(p,q)$, it follows that $s > 0$. Thus we can write $s = c \cdot \mathrm{dist}_{w'}(p,q)$, for some real number c, $0 < c < 1$. Fix some pair of processors j and k. By Assumption 1, $c \cdot \mathrm{mls}_\alpha(k,j)$ is a locally admissible shift of j w.r.t. k in α. In addition, $c \cdot \mathrm{mls}_\alpha(j,k)$ is a locally admissible shift of k w.r.t. j in α, and thus $-c \cdot \mathrm{mls}_\alpha(j,k)$ is a locally admissible shift of j w.r.t. k in α.

For every processor i define $s_i = c \cdot \mathrm{dist}_{w'}(p,i)$; note that $s_p = 0$ and $s_q = s$. We now show that $s_j - s_k$ is a locally admissible shift of j w.r.t. k in α. By the triangle inequality

$$\mathrm{dist}_{w'}(p,j) \leq \mathrm{dist}_{w'}(p,k) + w'(k,j) \ ,$$

and since $w'(k,j) = \mathrm{mls}_\alpha(k,j)$, we have (by changing sides)

$$\mathrm{dist}_{w'}(p,j) - \mathrm{dist}_{w'}(p,k) \leq \mathrm{mls}_\alpha(k,j) \ .$$

Since $c > 0$, by multiplying by c and substituting s_j and s_k, we get

$$s_j - s_k \leq c \cdot \mathrm{mls}_\alpha(k,j) \ .$$

By similar reasoning,

$$s_k - s_j \leq c \cdot \mathrm{mls}_\alpha(j,k) \ .$$

This implies

$$-c \cdot \mathrm{mls}_\alpha(j,k) \leq s_j - s_k \leq c \cdot \mathrm{mls}_\alpha(k,j) \ .$$

Since $-c \cdot \mathrm{mls}_\alpha(j,k)$ and $c \cdot \mathrm{mls}_\alpha(k,j)$ are locally admissible shifts of j w.r.t. k in α, Assumption 1 implies that $s_j - s_k$ is a locally admissible shift of j w.r.t. k in α.

Since this holds for any j and k, Claim 4.3 implies that *shift*$(\alpha, \langle s_1, \ldots, s_n \rangle)$ is in $\mathcal{A}$. Therefore, $s = s_q - s_p$ is a (globally) admissible shift of q w.r.t. p in α. Since $s > \mathrm{ms}_\alpha(p,q)$, this contradicts the definition of $\mathrm{ms}_\alpha(p,q)$. ◙

We now show that $\mathrm{ms}_\alpha(p,q) = \mathrm{dist}_{w'}(p,q)$, by proving the converse inequality.

Lemma 4.5 *For any two processors p and q, $\mathrm{ms}_\alpha(p,q) \leq dist_{w'}(p,q)$,*

Proof: Let s be an arbitrary admissible shift of q w.r.t. p in α. Then there exists a vector of shifts $S = \langle s_1, \ldots, s_n \rangle$ with $s = s_q - s_p$, such that *shift*$(\alpha, S) \in \mathcal{A}$. Consider a shortest path $p_0, \ldots, p_k$ from $p = p_0$ to $q = p_k$ with respect to the weights w'. Summing over the path we get

$$\sum_{i=1}^{k} (s_{p_i} - s_{p_{i-1}}) = s_{p_k} - s_{p_0} = s_q - s_p = s \ . \tag{1}$$

Since *shift*$(\alpha, S) \in \mathcal{A}$, Claim 4.3 implies that $s_{p_i} - s_{p_{i-1}}$ is a locally admissible shift of p_i w.r.t. p_{i-1} in α. Hence,

$$s_{p_i} - s_{p_{i-1}} \leq \mathrm{mls}_\alpha(p_{i-1}, p_i) = w'(p_{i-1}, p_i) \ , \tag{2}$$

for each $i = 1, \ldots, k$. By combining Equations (1) and (2) we get:

$$s = \sum_{i=1}^{k} (s_{p_i} - s_{p_{i-1}}) \leq \sum_{i=1}^{k} w'(p_{i-1}, p_i) = \mathrm{dist}_{w'}(p, q) \ .$$

Therefore, $s \leq \mathrm{dist}_{w'}(p, q)$. Since this holds for any admissible shift, it follows that $\mathrm{ms}_\alpha(p, q) \leq \mathrm{dist}_{w'}(p, q)$. ■

Theorem 4.6 *For any admissible execution α and any two processors p and q, $\mathrm{ms}_\alpha(p, q)$ can be computed from $\mathrm{mls}_\alpha(p, q)$.*

Proof: For any admissible execution α and any two processors p and q, Lemma 4.4 implies that $\mathrm{ms}_\alpha(p, q) \geq \mathrm{dist}_{w'}(p, q)$, where $w'(p, q) = \mathrm{mls}_\alpha(p, q)$. Lemma 4.5 implies that $\mathrm{ms}_\alpha(p, q) = \mathrm{dist}_{w'}(p, q)$. The claim follows since $\mathrm{dist}_{w'}(p, q)$ depends only on $\mathrm{mls}_\alpha(p, q)$. ■

4.3 Using Estimates for Local Shifts

Now, the issue is how to compute the values $\tilde{\mathrm{ms}}_\alpha(p, q)$ needed as inputs to the function SHIFTS. We assume that the function is provided with estimates of the local shifts. Under this assumption the computation can be accomplished by the following function GLOBAL ESTIMATES, with inputs $\tilde{\mathrm{mls}}_\alpha(p, q) = \mathrm{mls}_\alpha(p, q) + S_{\alpha,p} - S_{\alpha,q}$, for every pair of processors p and q.

1. Compute $\tilde{\mathrm{ms}}_\alpha(p, q)$ by a shortest path computation in G with weights $\tilde{\mathrm{mls}}_\alpha(p, q)$.

Theorem 4.7 *The function* GLOBAL ESTIMATES *computes $\tilde{\mathrm{ms}}_\alpha(p, q)$, for every pair of processors p and q.*

Proof: Observe that the weight of any cycle w.r.t. the weights $\tilde{\mathrm{mls}}_\alpha$ is equal to the weight of the cycle w.r.t. the weights mls_α because the S components cancel. It follows that there are no negative weight cycles in G w.r.t. the weights $\tilde{\mathrm{mls}}_\alpha$. Also, the weight of any path from p to q w.r.t. weights $\tilde{\mathrm{mls}}_\alpha$ is equal to the weight of the path w.r.t. mls_α plus $S_{\alpha,p} - S_{\alpha,q}$. The claim follows from Theorem 4.6. ■

By composing functions GLOBAL ESTIMATES and SHIFTS, we can compute the optimal corrections and their precision given only the estimates to the maximal local shifts $\tilde{\mathrm{mls}}_\alpha$. This follows immediately from Theorem 4.7 above, together with Theorem 3.7.

4.4 A Composition Theorem

In many systems, several constraints are imposed on the delay of messages. For example, it is possible that there is a bound on the delay in each direction of the link as well as a bound on the difference in message delay in opposite directions. In these cases, the system is local with respect to several sets of executions, each of which is local to the same pair of processors p and q. We now show how to combine several sets of executions local to p and q into a single complex set of executions local to p and q. This allows us to deal with complex local systems by regarding each pair of processors and each assumption separately; this will be useful in the next section.

We remark that the theory developed so far already allows us to deal with local systems where different pairs of processors obey different types of constraints.

Note that the notion of an admissible shift (and the derived notion of maximal shift) is defined in the context of a specific set of admissible executions. To develop the results in this section it is convenient to state this fact explicitly by saying that a value is an admissible shift (or maximal shift) *under* $\mathcal{A}$, where $\mathcal{A}$ is some set of executions.

For some pair of processors p and q, let $\mathcal{A}'_{p,q}$ be a set of executions local to p and q, and let $\mathcal{A}''_{p,q}$ be another set of executions local to p and q. Denote $\mathcal{A}_{p,q} = \mathcal{A}'_{p,q} \cap \mathcal{A}''_{p,q}$. It is easy to see that $\mathcal{A}_{p,q}$ is local to p and q. For any execution $\alpha \in \mathcal{A}$, let $\mathrm{mls}'_\alpha(p,q)$ be the maximal local shift of q w.r.t. p in α under $\mathcal{A}'_{p,q}$. Similarly, define $\mathrm{mls}_\alpha(p,q)$ and $\mathrm{mls}''_\alpha(p,q)$. We have:

Theorem 4.8 $\mathrm{mls}_\alpha(p,q) = \min\{\mathrm{mls}'_\alpha(p,q), \mathrm{mls}''_\alpha(p,q)\}$.

Proof: The fact that $\mathrm{mls}_\alpha(p,q) \leq \mathrm{mls}'_\alpha(p,q)$ follows immediately since every execution in $\mathcal{A}_{p,q}$ is an execution in $\mathcal{A}'_{p,q}$. Thus if s is a locally admissible shift of q w.r.t. p in α under $\mathcal{A}_{p,q}$, then it is a locally admissible shift of q w.r.t. p in α under $\mathcal{A}'_{p,q}$. Similarly $\mathrm{mls}_\alpha(p,q) \leq \mathrm{mls}''_\alpha(p,q)$. Therefore, $\mathrm{mls}_\alpha(p,q) \leq \min\{\mathrm{mls}'_\alpha(p,q), \mathrm{mls}''_\alpha(p,q)\}$.

Assume for contradiction that s is a value such that $\mathrm{mls}_\alpha(p,q) < s < \min\{\mathrm{mls}'_\alpha(p,q), \mathrm{mls}''_\alpha(p,q)\}$. Since $\mathrm{mls}_\alpha(p,q) < s < \mathrm{mls}'_\alpha(p,q)$, Assumption 1 implies that s is a locally admissible shift of q w.r.t. p in α under $\mathcal{A}'_{p,q}$. Similarly, s is also a locally admissible shift of q w.r.t. p in α under $\mathcal{A}''_{p,q}$. By Claim 4.2, for every vector of shifts $S = \langle s_1, \ldots, s_n \rangle$, such that $s = s_q - s_p$, *shift*(α, S) is in both $\mathcal{A}'_{p,q}$ and $\mathcal{A}''_{p,q}$. Therefore, *shift*(α, S) is in $\mathcal{A}_{p,q}$. Thus, s is a locally admissible shift of q w.r.t. p in α under $\mathcal{A}_{p,q}$, a contradiction since $\mathrm{mls}_\alpha(p,q) < s$. Therefore $\mathrm{mls}_\alpha(p,q) = \min\{\mathrm{mls}'_\alpha(p,q), \mathrm{mls}''_\alpha(p,q)\}$. ∎

5 Clock Synchronization for Specific Delay Assumptions

We now show how to compute estimated maximal local shifts for specific sets of executions $\mathcal{A}_{p,q}$ local to p and q, given the views. By Theorem 4.7 and Theorem 4.8, this implies a clock synchronization algorithm that computes optimal corrections for any system whose set of admissible executions is the intersection of any collection of sets of these types. By Theorem 4.7, all we have to show is how to compute the estimates of the maximal local shifts $\widetilde{\mathrm{mls}}_\alpha(p,q)$. This calculation is based on estimates for the delays (defined below) which can be easily computed from the views of the processors.

The *estimated delay* $\tilde{\mathrm{d}}(m)$ of a message m sent from p to q is the actual (real time) delay plus the difference in (real time) start times of the processors; that is, $\tilde{\mathrm{d}}(m) = \mathrm{d}(m) + S_{\alpha,p} - S_{\alpha,q}$. This is similar to the definitions of estimated maximal global shifts and estimated maximal local shifts. The next lemma shows that the estimated delay can be computed from the views.

Lemma 5.1 *Given the views of processors p and q in an execution α, it is possible to compute the estimated delay $\tilde{\mathrm{d}}(m)$ of any message m sent from p to q.*

Proof: Let $t_p(m)$ denote the local (p's) clock time when p sent the message m according to p's view; similarly, $t_q(m)$ denotes the local (q's) clock time when q received the message m according to q's view. By Property 4 of histories as defined in Section 2.2, m was sent at real time $t_p(m)+S_{\alpha,p}$; similarly, m was received at real time $t_q(m)+S_{\alpha,q}$. It follows that the delay of m is $\mathrm{d}(m) = (t_q(m)+S_{\alpha,q}) - (t_p(m)+S_{\alpha,p})$. Hence, $\tilde{\mathrm{d}}(m) = \mathrm{d}(m)+S_{\alpha,p}-S_{\alpha,q} = t_q(m)-t_p(m)$. Since messages are unique, $t_p(m)$ and $t_q(m)$ can be computed from the views of p and q. ▣

5.1 Bounds on the Delay

In the systems considered in [3, 10], there is an upper and a lower bound on the transmission delay, for any edge. We extend this assumption by allowing edges without upper bounds, in which case we say that the upper bound is ∞. Thus, in particular, we have for the first time an optimal clock synchronization in a completely asynchronous network where there are no bounds on the delay.

Consider a set $\mathcal{A}_{p,q}[l,u]$ where l and u give bounds (real numbers) for each ordered pair of processors p and q, such that $0 \le l(p,q) \le u(p,q) \le \infty$. Execution α is in $\mathcal{A}_{p,q}[l,u]$ if the delay of every message sent from p to q is in the range $[l(p,q), u(p,q)]$ and the delay of every message from q to p is in the range $[l(q,p), u(q,p)]$. Clearly, $\mathcal{A}_{p,q}[l,u]$ is local to p and q.

The maximal delay of a message received by q from p in execution α is denoted $\mathrm{d}_\alpha^{\max}(p,q)$. Similarly, the minimal delay of a message received by q from p in α is denoted $\mathrm{d}_\alpha^{\min}(p,q)$. If no message was received by q from p in α then $\mathrm{d}_\alpha^{\max}(p,q) = -\infty$ and $\mathrm{d}_\alpha^{\min}(p,q) = \infty$. We first observe that in such systems, $\mathrm{mls}_\alpha(p,q)$ depends only on the maximal and minimal delays between p and q.

Lemma 5.2 *Let α be an execution of $(P, \mathcal{A}_{p,q}[l,u])$. Then*

$$\mathrm{mls}_\alpha(p,q) = \min\{(u(q,p) - \mathrm{d}_\alpha^{\max}(q,p)), (\mathrm{d}_\alpha^{\min}(p,q) - l(p,q))\}.$$

Proof: We can partition the constraints on the communication between p and q into two: The conditions on the delay of messages from p to q and the conditions on the delay of messages from q to p. This is done by expressing the set $\mathcal{A}_{p,q}[l,u]$ as the intersection of $\mathcal{A}_{<q,p>}[l,u]$, which constrains the messages from q to p, and $\mathcal{A}_{<p,q>}[l,u]$, which constrains the messages from p to q. Let $\mathrm{mls}'_\alpha(p,q)$ be the maximal local shift of q w.r.t. p in α under $\mathcal{A}_{<q,p>}[l,u]$; $\mathrm{mls}''_\alpha(p,q)$ is defined similarly under $\mathcal{A}_{<p,q>}[l,u]$.

We first show that $\mathrm{mls}'_\alpha(p,q) = u(q,p) - \mathrm{d}^{\max}_\alpha(q,p)$. It is obvious that, $\mathrm{mls}'_\alpha(p,q) \leq u(q,p) - \mathrm{d}^{\max}_\alpha(q,p)$. Assume, by way of contradiction, that $s > u(q,p) - \mathrm{d}^{\max}_\alpha(q,p)$ is a locally admissible shift of q w.r.t. p in α. This immediately implies that $\mathrm{d}^{\max}_\alpha(q,p) > -\infty$, i.e., at least one message was received by p from q.

Denote $\alpha' = \mathit{shift}(\alpha, \langle s_1, \ldots, s_n \rangle)$, where $s_q = s$ and $s_i = 0$ for all $i \neq q$. By Claim 4.2, α' is in $\mathcal{A}_{<q,p>}$. By Claim 3.2, if a message m from q to p has delay d in α, then m has delay $\mathrm{d} + s$ in α'. Thus, $\mathrm{d}^{\max}_{\alpha'}(q,p) = \mathrm{d}^{\max}_\alpha(q,p) - s$. Since $s > u(q,p) - \mathrm{d}^{\max}_\alpha(q,p)$ and $\mathrm{d}^{\max}_\alpha(q,p) > -\infty$, it follows that the maximal delay of a message from q to p in α' is strictly greater than $u(q,p)$. A contradiction.

In a similar manner, we show that $\mathrm{mls}''_\alpha(p,q) = \mathrm{d}^{\min}_\alpha(p,q) - l(p,q)$. It is obvious that $\mathrm{mls}''_\alpha(p,q) \leq \mathrm{d}^{\min}_\alpha(p,q) - l(p,q)$. Assume, by way of contradiction, that $s > \mathrm{d}^{\min}_\alpha(p,q) - l(p,q)$ is a locally admissible shift of q w.r.t. p in α. This immediately implies that $\mathrm{d}^{\min}_\alpha(p,q) < \infty$, i.e., at least one message was received by q from p.

Let α' be as defined above. By Claim 4.2, α' is in $\mathcal{A}_{<p,q>}$. By Claim 3.2, if a message m from p to q has delay d in α, then m has delay $\mathrm{d} - s$ in α'. Thus, $\mathrm{d}^{\min}_{\alpha'}(p,q) = \mathrm{d}^{\min}_\alpha(p,q) - s$. Since $s > \mathrm{d}^{\min}_\alpha(p,q) - l(p,q)$ and $\mathrm{d}^{\min}_\alpha(p,q) < \infty$, it follows that the minimal delay of a message from p to q in α' is less than $l(p,q)$. A contradiction.

The claim now follows from Theorem 4.8. ◻

Lemma 5.2 gives the maximal local shifts as a function of the actual maximal and minimal delays. However, the views of the processors give only estimates of the delays, not the delays themselves. Yet, the estimates of the delays give an estimate for the maximal local shift $\tilde{\mathrm{mls}}_\alpha(p,q)$. Formally, the *estimated maximal delay* is defined as

$$\tilde{\mathrm{d}}^{\max}_\alpha(p,q) = \mathrm{d}^{\max}_\alpha(p,q) + S_{\alpha,p} - S_{\alpha,q} \ ,$$

while the *estimated minimal delay* is defined as

$$\tilde{\mathrm{d}}^{\min}_\alpha(p,q) = \mathrm{d}^{\min}_\alpha(p,q) + S_{\alpha,p} - S_{\alpha,q} \ .$$

We have:

Corollary 5.3 *Let α be an admissible execution of $(P, \mathcal{A}_{p,q}[l,u])$. Then*

$$\tilde{\mathrm{mls}}_\alpha(p,q) = \min\{(u(q,p) - \tilde{\mathrm{d}}^{\max}_\alpha(q,p)), (\tilde{\mathrm{d}}^{\min}_\alpha(p,q) - l(p,q))\}.$$

Note that $\tilde{\mathrm{d}}^{\max}$ and $\tilde{\mathrm{d}}^{\min}$ can be computed from the views of p and q, since $\tilde{\mathrm{d}}^{\min}_{\alpha}(p,q)$ is the minimum of $\tilde{\mathrm{d}}(m)$ for all messages m received by q from p in α, and $\tilde{\mathrm{d}}^{\max}_{\alpha}(p,q)$ is the maximum of $\tilde{\mathrm{d}}(m)$ for all messages m received by q from p in α.

If we make the natural assumption that all delays are non-negative, we get a general bound on mls and $\widetilde{\mathrm{mls}}$ (without any other bounds on the delay).

Corollary 5.4 *Let α be an admissible execution of a system $(P, \mathcal{A}_{p,q})$ local to p, q. Then* $\mathrm{mls}_{\alpha}(p,q) \leq \mathrm{d}^{\min}_{\alpha}(p,q)$ *and* $\widetilde{\mathrm{mls}}_{\alpha}(p,q) \leq \tilde{\mathrm{d}}^{\min}_{\alpha}(p,q)$.

5.2 Links with Bounds on the Round Trip Delay Bias

In many communication links there are no tight bounds on the transmission delays. However, whenever the traffic load on one direction of a link is high, the load in the opposite direction of the link is also high. Thus, it is possible to give a bound on the difference, or bias, between delay in one direction and delay in the opposite direction. For the purpose of illustrating our techniques, we simplify the assumption and require that the difference between the delay of any pair of messages in opposite directions is bounded. We now show how to calculate maximal local shifts for links in this case. It is possible to generalize our results to the more realistic scenario in which this assumption holds only for messages that were sent "around the same time."

Specifically, we associate a nonnegative number $b(p,q)$ with processors p, q. An execution α is in $\mathcal{A}_{p,q}[b]$ if for any message m received by p from q, and any message m' received by q from p, $|\mathrm{d}_{\alpha}(m) - \mathrm{d}_{\alpha}(m')| \leq b(p,q)$. We also restrict $\mathcal{A}_{p,q}[b]$ to nonnegative delays, i.e., for every message m, $d(m) \geq 0$. The next lemma shows that $\mathrm{mls}_{\alpha}(p,q)$ depends only on the maximal and minimal delays between p and q.

Lemma 5.5 *Let α be an admissible execution of $(P, \mathcal{A}_{p,q}[b])$. Then*

$$\mathrm{mls}_{\alpha}(p,q) = \min\{\mathrm{d}^{\min}_{\alpha}(p,q), \frac{1}{2}[b(p,q) + \mathrm{d}^{\min}_{\alpha}(p,q) - \mathrm{d}^{\max}_{\alpha}(q,p)]\}.$$

Proof: Consider the following two sets of admissible executions local to p and q. The first set $\mathcal{A}'_{p,q}$ contains every execution α such that the delay of every message in α is non-negative. The second set $\mathcal{A}''_{p,q}$ is like $\mathcal{A}_{p,q}[b]$ except that the delays are allowed to be negative.

Clearly $\mathcal{A}_{p,q}[b]$ is the intersection of $\mathcal{A}'_{p,q}$ and $\mathcal{A}''_{p,q}$. By Theorem 4.8, $\mathrm{mls}_{\alpha}(p,q) = \min\{\mathrm{mls}'_{\alpha}(p,q), \mathrm{mls}''_{\alpha}(p,q)\}$. By Lemma 5.2, $\mathrm{mls}'_{\alpha}(p,q) = \mathrm{d}^{\min}_{\alpha}(p,q)$. Therefore, it suffices to prove that $\mathrm{mls}''_{\alpha}(p,q) = \frac{1}{2}[b(p,q) + \mathrm{d}^{\min}_{\alpha}(p,q) - \mathrm{d}^{\max}_{\alpha}(q,p)]$.

Fix a value s and denote $\alpha' = \mathit{shift}(\alpha, \langle s_1, \ldots, s_n \rangle)$, where $s_q = s$ and $s_i = 0$ for all $i \neq q$. By Claim 4.2, s is a locally admissible shift of q w.r.t. p in α if and only if $\alpha' \in \mathcal{A}_{p,q}(\varepsilon)$.

By Claim 3.2, for any message m received by p from q, $d_{\alpha'}(m) = d_\alpha(m) + s$. Similarly, for any message m' received by q from p, $d_{\alpha'}(m') = d_\alpha(m') - s$. Therefore,

$$\begin{array}{rrcl} & d_{\alpha'}(m') - d_{\alpha'}(m) & = & d_\alpha(m') - d_\alpha(m) - 2s, \\ \text{and} & d_{\alpha'}(m) - d_{\alpha'}(m') & = & d_\alpha(m) - d_\alpha(m') + 2s. \end{array}$$

The round trip delay bias of an arbitrary pair of messages m and m' in α' is at most $b(p,q)$ if and only if

$$\begin{array}{rrcl} & d_\alpha(m') - d_\alpha(m) - 2s & \leq & b(p,q), \\ \text{and} & d_\alpha(m) - d_\alpha(m') + 2s & \leq & b(p,q). \end{array}$$

Since α is admissible and $s \geq 0$, the first inequality trivially holds. Hence, s is a locally admissible shift of q w.r.t. p if and only if

$$s \leq \frac{1}{2}[b(p,q) + d_\alpha(m') - d_\alpha(m)] \ .$$

Namely,

$$s \leq \frac{1}{2}[b(p,q) + \mathrm{d}_\alpha^{\min}(p,q) - \mathrm{d}_\alpha^{\max}(q,p)] \ .$$

Thus $\mathrm{mls}''_\alpha(p,q) = \frac{1}{2}[b(p,q) + \mathrm{d}_\alpha^{\min}(p,q) - \mathrm{d}_\alpha^{\max}(q,p)]$. ∎

Corollary 5.6 *Let α be an admissible execution of $(P, \mathcal{A}[b])$. Then*

$$\widetilde{\mathrm{mls}}_\alpha(p,q) = \min\{\tilde{\mathrm{d}}_\alpha^{\min}(p,q), \frac{1}{2}[b(p,q) + \tilde{\mathrm{d}}_\alpha^{\min}(p,q) - \tilde{\mathrm{d}}_\alpha^{\max}(q,p)]\}.$$

5.3 Multicast Networks

Communication in many networks is performed through broadcast media where a message is transmitted simultaneously to a subset of the processors. Multicast transmission may often have useful timing properties for clock synchronization. In this subsection we investigate a simple timing property: there exists a bound ε on the difference between the arrival times of a message at different processors.

Optimal clock synchronization algorithms for multicast networks which have a bound on the differences in delay for different processors are presented in [4, 16]. Our solution demonstrates the usefulness of the reductions of the preceding sections. To provide optimal clock synchronization under the multicast assumption we need only to find a way of defining local shifts. This, somewhat surprisingly, turns out to be an easy task. Furthermore, the broadcast model can be limited to specific subsets corresponding to subnetworks of an internet, and combined with the other assumptions using the Composition Theorem.

To define this assumption, we allow events of the form send(p, m, Q), for all messages m and sets of processors Q; this event represents a multicast of m to the processors in Q. The definition of an execution is modified so that there is a one-to-one correspondence between the messages received by p from k to messages sent by k to p or multicast by k to a set Q which includes p, for any processors p and k. Let $d(p,m)$ denote the difference between the time the message m is multicast by some processor k and the time processor p receives it; this is the *delay of the message m to p*. The *estimated delay of the message m to p* in α is $\tilde{d}(p,m) = d(p,m) + S_{\alpha,k} - S_{\alpha,p}$. As before, the estimated delay of a message from k to p can be computed from the views of p and k.

The system is the pair $(P, \mathcal{A}(\varepsilon))$, where $\mathcal{A}(\varepsilon)$ is is the the intersection of local sets $\mathcal{A}_{p,q}(\varepsilon)$, for every unordered pair of processors p and q. An execution α is in $\mathcal{A}_{p,q}(\varepsilon)$ if for every message m multicast to both p and q, $|d(p,m) - d(q,m)| \leq \varepsilon$. That is, m reaches p at most ε after it reaches q, and vice versa.

Note that $\mathcal{A}_{p,q}(\varepsilon)$ is local to p and q. This is because any shift applied to both p and q does not change the difference $d(p,m) - d(q,m)$. The next lemma shows how to calculate $\mathrm{mls}_\alpha(p,q)$.

Lemma 5.7 *Let α be an admissible execution of $(P, \mathcal{A}_{p,q}(\varepsilon))$. Then*

$$\mathrm{mls}_\alpha(p,q) = \varepsilon + \min_m\{d(q,m) - d(p,m)\}.$$

Proof: Fix a value s and denote $\alpha' = \mathit{shift}(\alpha, \langle s_1, \ldots, s_n \rangle)$, where $s_q = s$ and $s_i = 0$ for all $i \neq q$. By Claim 4.2, s is a locally admissible shift of q w.r.t. p in α if and only if $\alpha' \in \mathcal{A}_{p,q}(\varepsilon)$.

For every message m multicast to both p and q, let $d'(q,m)$ and $d'(p,m)$ denote the delay of m in α' for processors p and q respectively. The value s is a locally admissible shift of q with respect to p in α if and only if

$$|d'(p,m) - d'(q,m)| \leq \varepsilon\ ,$$

for every message m multicast to both p and q. If q is not the sender of m then $d'(p,m) = d(p,m)$ and $d'(q,m) = d(q,m) - s$; if q is the sender of m then $d'(q,m) = d(q,m)$ and $d'(p,m) = d(p,m) + s$. In both cases, s is a locally admissible shift of q with respect to p in α if and only if

$$|d(p,m) - d(q,m) + s| \leq \varepsilon\ ,$$

for every message m multicast to both p and q.

Hence, s is a locally admissible shift of q with respect to p in α if and only if

$$|s| \leq \varepsilon + |d(p,m) - d(q,m)|\ ,$$

for every message m multicast to both p and q. Since $\mathrm{mls}_\alpha(p,q) \geq 0$, this implies that

$$\mathrm{mls}_\alpha(p,q) = \varepsilon + \min_m\{d(p,m) - d(q,m)\}\ ,$$

as needed. ∎

As before, this result applies also to the estimated delays which can be computed from the views.

Corollary 5.8 *Let α be an admissible execution of $(P, \mathcal{A}_{p,q}(\varepsilon))$. Then*

$$\tilde{\mathrm{mls}}_\alpha(p,q) = \varepsilon + \min_m\{\tilde{d}(q,m) - \tilde{d}(p,m)\} \ .$$

Proof: Fix a message m and let k be the sender of m. We have

$$\begin{aligned}
\tilde{\mathrm{mls}}_\alpha(p,q) &= \mathrm{mls}_\alpha(p,q) + S_{\alpha,p} - S_{\alpha,q} && \text{(by definition)}\\
&= \varepsilon + \min_m\{d(q,m) - d(p,m)\} + S_{\alpha,p} - S_{\alpha,q} && \text{(by Lemma 5.7)}\\
&= \varepsilon + \min_m\{(d(q,m) + S_{\alpha,k} - S_{\alpha,q}) - (d(p,m) + S_{\alpha,k} - S_{\alpha,p})\}\\
&= \varepsilon + \min_m\{\tilde{d}(q,m) - \tilde{d}(p,m)\} && \text{(by definition),}
\end{aligned}$$

as needed. □

6 Discussion

We have shown a framework for designing optimal clock synchronization algorithms under a variety of assumptions on message delay uncertainty. The general result yields optimal clock synchronization algorithms under the following assumptions: upper and lower bounds on delays are known; only lower bounds on the delays are known; no bounds are known; only a bound on the difference of the round trip delays is known; and a multicast assumption which bounds the difference in delay in reaching different processors. Moreover, the results apply to cases where different links satisfy different assumptions, or the same link satisfies several assumptions. This work extends results of Halpern, Megiddo and Munshi [3], and introduces a new notion of optimality on any specific instance.

The specific delay assumptions analyzed here are typical of realistic systems, and it seems relatively easy to perform similar analysis for additional delay assumptions. It is our belief that this will lead to the design of optimal clock synchronization algorithms for other message delay assumptions.

In this paper, we only address the issue of computing optimal corrections, given the views of the processors. An interesting open question is to compute the optimal corrections in a distributed manner. To understand the difficulty involved in the distributed implementation of this computation, consider the following straightforward approach. Each pair of neighboring processors p and q compute $\tilde{\mathrm{mls}}_\alpha(p,q)$ and $\tilde{\mathrm{mls}}_\alpha(q,p)$ using the estimated delays (which can be deduced from their views). All processors send the estimated maximum local shifts to a distinguished processor (leader). The leader computes the estimated maximum global shifts using function GLOBAL ESTIMATES, and a correction value for each processor according to function SHIFTS. Finally, the leader sends the corrections to the processors. Note, however, that the precision obtained by this centralized clock synchronization algorithm is optimal only with respect to the part of the execution that does not include the messages to and from the leader. That is, any additional communication, required for exchanging the views, is bound to change the views themselves. A solution may require the definition of optimality to be relaxed. Some extensions in this direction have already been obtained in [15]. Also in this work, some of our results have been generalized to clocks that drift.

Another important open question is to achieve optimal clock synchronization in systems where the probabilistic properties of the message delay distribution are known. This assumption is at the heart of most practical algorithms for clock synchronization [1, 12]. We believe the setting developed here allows one to address this assumption, and that this will lead to improvements in these important algorithms.

Finally, an obvious open problem is to make our results to be fault-tolerant, following the many works addressing fault-tolerant clock synchronization.

Acknowledgments: We thank Joe Halpern, Marios Mavronicolas and Boaz Patt-Shamir for helpful comments.

References

[1] F. Cristian, "Probabilistic Clock Synchronization," *Dist. Comp.*, 3 (1989), pp. 146–158.

[2] D. Dolev, J. Halpern and H. R. Strong, "On the possibility and impossibility of achieving clock synchronization," *J. Comp. and Sys. Sci.*, 32:2 (1986) pp. 230–250.

[3] J. Halpern, N. Megiddo and A. A. Munshi, "Optimal precision in the presence of uncertainty," *J. Complexity,* 1 (1985), pp. 170–196.

[4] J. Halpern and I. Suzuki, "Clock Synchronization and the Power of Broadcasting," *Proc. Allerton Conference*, 1990, pp. 588–597.

[5] R. M. Karp, "A characterization of the minimum cycle mean in a digraph," *Disc. Math.*, 23 (1978), pp. 309–311.

[6] H. Kopetz and W. Ochsenreiter, "Clock Synchronization in Distributed Real-Time Systems," *IEEE Trans. Comp.*, 36:8 (August 1987), pp. 933–939.

[7] L. Lamport, "Time, clocks and the ordering of events in distributed systems," *Communications of the ACM,* 21:7 (July 1978), pp. 558–565.

[8] L. Lamport and P. Melliar-Smith, "Synchronizing clocks in the presence of faults," *Journal of the ACM,* 32:1 (January 1985), pp. 52–78.

[9] B. Liskov, "Practical Uses of Synchronized Clocks in Distributed Systems," invited talk at the *9th ACM Symp. on Principles of Distributed Computing,* 1990. Appeared in *Distributed Computing,* 6 (1993), pp. 211–219.

[10] J. Lundelius and N. Lynch, "An Upper and Lower Bound for Clock Synchronization," *Info. and Control,* 62:2/3 (August/September 1984), pp. 190–204.

[11] K. Marzullo, *Loosely-Coupled Distributed Services: A Distributed Time Service,* Ph.D. thesis, Stanford University, 1983.

[12] D. Mills, "Network Time Protocol (Version 2) Specification and Implementation," *IEEE Trans. Comm.*, Vol. 39, No. 10 (October 1991), pp. 1482–1493.

[13] Y. Ofek, "Generating a fault tolerant clock using high-speed control signals for the MetaNet architecture," *IEEE Trans. Comm.*, 1993. To appear.

[14] Open Software Foundation, *Introduction to OSF DCE,* OSF, Cambridge, Massechusetts, December 1991.

[15] B. Patt-Shamir and S. Rajsbaum, "A Theory of Clock Synchronization," to appear in *Proc. 26th ACM Symp. on Theory of Computing,* 1994.

[16] K. Sugihara and I. Suzuki, "Nearly Optimal Clock Synchronization Under Unbounded Message Transmission Time," *Proc. 1988 International Conference on Parallel Processing III*, 1988, pp. 14–17.

[17] B. Simons, J. L. Welch and N. Lynch, *An overview of clock synchronization,* IBM Technical Report RJ 6505, October 1988.

[18] T. Srikanth and S. Toueg, "Optimal Clock Synchronization," *Journal of the ACM,* 34:3 (July 1987), pp. 626–645.

[19] J. L. Welch and N. Lynch, "A new fault-tolerant algorithm for clock synchronization," *Info. and Comp.,* 77:1 (April 1988), pp. 1–36.