Optimal Clock Synchronization under Different Delay Assumptions

(Preliminary Version)

Hagit Attiya* Amir Herzberg† Sergio Rajsbaum‡

Abstract: The problem of achieving optimal clock synchronization in a communication network with arbitrary topology and perfect clocks (that do not drift) is studied. A novel modular presentation of the problem is described which allows to deal with different assumptions for the delay of messages.

We present a definition of clock synchronization under arbitrary delay assumptions, and present an optimal clock synchronization algorithm for general systems. We then show that in local systems (where delays on each link are independent of the other links) the inputs for the clock synchronization algorithm can be computed from the *maximum local shifts* for each pair of processors sharing a link. The maximum local shift for two processors depends only on their views. This allows our theory to deal with systems where different links adhere to different assumptions, or the same link satisfies several sets of assumptions; such mixtures are quite likely in practice. In particular, we show how to compute the maximum local shifts from the views, and hence provide optimal algorithms for systems where some links may have upper and/or lower bounds on the delay, some may have a bound on the difference between the delay in both directions, some may have both kinds of bounds and some may have no bounds. Previous results dealt only with the case where upper and lower bounds were known for all links.

We introduce a new notion of optimality, that requires an algorithm to achieve the best possible precision *on each instance*; this notion is stronger than the previously used notion of worst case optimality. In contrast to the worst case approach, the new notion handles models where the worst-case behavior of any clock synchronization algorithm is inherently unbounded.

*Department of Computer Science, Technion. Partially supported by Technion V.P.R. funds from the Argentinian Research Fund, and by the fund for the promotion of research in the Technion. Email: hagit@cs.technion.ac.il.

†IBM T. J. Watson Research Center. Partly supported by DGAPA Projects, National Autonomous University of Mexico (UNAM). Email: amir@yktvmh.bitnet or amir@watson.ibm.com.

‡Instituto de Matemáticas, U.N.A.M., D.F. 04510, Mexico. Partly supported by DGAPA Projects, National Autonomous University of Mexico (UNAM). Email: rajsbaum@redvax1.dgsca.unam.mx.

12th ACM Symposium on Principles on Distributed Computing, Ithaca NY

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 (e.g. in factories and space vehicles), 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]; designers of practical systems have been advocating the use of synchronized clocks [9].

We measure the quality of synchronization by its *precision*, i.e., how close together it brings the clocks at different processors. This does not ensure that clocks are close to the real time, although it is easy to adapt our results to obtain 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.)

The precision can influence correctness and efficiency of applications that use the synchronized clocks. It is known that the precision which can be

achieved depends on 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 this model, whose precision is optimal in the worst case, are described in [10, 3]. Subsequent work (e.g. [2, 6, 16, 17], see survey in [15]), concentrated on clocks that may drift and on fault-tolerance. To achieve high precision, these works require the existence of fairly tight lower and upper bounds on message delay.

In real systems, however, it is the uncertainty of message delay, and not the 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.[1] On the other hand, it is very hard to come up with tight upper and lower bounds on message delay. For example, in some systems, only a bound on the difference between delays in opposite directions is known.

In this paper, we assume that local clocks run at the same speed and have no drift. Our main contribution is a methodology for designing optimal clock synchronization algorithms under a variety of assumptions on message delay uncertainty. We partition the design of a clock synchronization algorithm into four parts:

First, we show how to achieve optimal clock synchronization given information about the execution which captures the maximal possible "shifts" of one processor with respect to another. (This reduction holds for any kind of delay assumptions.) Second, we show how to compute maximal shifts from maximal "local shifts," which can be derived from a local computation. Third, we prove a decomposition theorem which allows to combine algorithms for multiple delay assumptions, even for the same link. (The second and the third steps apply under the natural assumption that delays on each link are independent of the operation of other links.) Fourth, we show how to compute the maximal local shifts from the local views for a number of specific delay models.

[1] Of course, this is a simplification. However, Kopetz and Ochsenreiter show that this simplification is reasonable in practice [6]. To deal with the small drift which does exist, the clock synchronization mechanism is invoked periodically, as explained in [6].

Our methodology yields optimal clock synchronization algorithms for a large variety of delay models. In particular, we show how to compute maximal local shifts for the following models:

1. upper and lower bounds on delays are known;
2. only lower bounds on the delays are known;
3. no bounds are known; and
4. only a bound on the difference of the round trip delays is known.

Previous formal work on deterministic clock synchronization addressed only the first model.[2] However, many practical systems are better modeled by the second and the fourth models. The second model 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 model follows experimental results (cf. [12]), showing that usually the delay in two directions of a bidirectional link is roughly the same.

Our decomposition theorem imply that our algorithms apply to systems where the same link satisfies several different delay assumptions. Such mixtures are quite common in practical wide-area, heterogeneous systems.

Our work can be viewed as an extension of the work by Halpern, Megiddo and Munshi [3], sharing many ideas and techniques. 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 each of their techniques, explicitly showing what are the requirements of each step and thereby facilitating adaptations to other delay assumptions. Given the framework developed here, their results become a special case where 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]. Our additions in order to deal with other and more general models are simple and independent. It is our belief that this will lead to the design of optimal clock synchronization algorithms for other network models.

Previous definitions of optimal clock synchronization were based on the worst (largest) difference between clocks of two processors in any execution. In some of the models that we address in this paper,

[2] Some previous works, e.g., [4, 14], also consider a model in which each processor has a clock that runs at the rate of real time. However, they assume that the network has a broadcast primitive. We believe our results can be used to deal with this model too.

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 application of the protocol. Although their algorithms achieve this stronger notion of optimality (as our results show), Halpern, Meggiddo and Munshi do not make explicit what is the precise definition of optimality under favorable conditions. When trying to crystallize these ideas, it turned out we had to separate the decision of which messages to send, 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.

2 Formal Definitions

In this section we define the model and the problem; our definitions follow [10, 3].

2.1 Model of Computation

We consider a finite directed graph $G = (V, E)$ where the nodes $V = \{p_1, \ldots, p_n\}$ represent *processors* and the edges E represent directed communication links. With each processor $p \in V$ we associate a (local) clock. The clock cannot be modified by the processor. Processors do not have access to the 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 have no drift, i.e., that they run at the same rate as real time, but they are not necessarily synchronized with each other.

Roughly, the following *events* which can occur at processor p: *Message receive events*– processor p receives message m from processor q; *Message send events*—processor p sends message m to processor q; *Timer set events*—processor p sets a timer to go off when its clock reads T; *Timer events*—a timer that was set for time T on p's clock goes off; and *Start events*—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 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 $t - S_\pi$; hence, 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 and clocks, one for each processor p in V, such that there is a one-to-one and onto 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*, denoted d(m), of any message m received in an execution to be the real time of receipt minus the real time of sending. 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 or lose messages; the system can reorder messages. A system $(G, \mathcal{A})$ is a graph $G = (V, E)$ and a set of executions $\mathcal{A}$, called *admissible executions.*

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. Two histories of processor p are *equivalent*, if the view of p is the same in both histories. Two executions α and α' are *equivalent*, denoted $\alpha \equiv \alpha'$, if for each processor p, the component histories for p in α and α' are equivalent.

2.2 The Clock Synchronization Problem

The goal of a clock synchronization algorithm is to bring the clocks of processors to be as close together as possible, while keeping the clocks' values with the progress of real time. Intuitively, the processors maintain 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, for each processor, a correction such that, for any two processors, the values of the local clocks (at the same real time) plus the respective corrections, are not far away. More precisely, we define:

Definition 2.1 *Let ε be a function from $\mathcal{A}$ to $\Re$. An algorithm* solves the clock synchronization problem on a system $(G, \mathcal{A})$ with $\varepsilon()$ precision *if for every admissible execution $\alpha \in \mathcal{A}$, it computes* corrections $\text{offset}_{\alpha,p}$ *for every processor p, such that for every pair of processors p and q,*

$$|(S_{\alpha,p} - \text{offset}_{\alpha,p}) - (S_{\alpha,q} - \text{offset}_{\alpha,q})| \leq \varepsilon(\alpha).$$

To understand the intuition behind this definition, recall that at any real time t, the clock value of p is $t - S_{\alpha,p}$. Given corrections $\text{offset}_{\alpha,p}$, the corrected local time of p is $t - S_{\alpha,p} + \text{offset}_{\alpha,p}$. Therefore, $|(S_{\alpha,p} - \text{offset}_{\alpha,p}) - (S_{\alpha,q} - \text{offset}_{\alpha,q})|$ is the difference between the corrected local times of p and q.

Our definition differs from the ones in [3, 10] since we do not require a fixed bound on the precision in all executions. Rather, we allow the precision to be a function of the execution. This is motivated by our definition of optimality, to which we dedicate the next section.

3 Defining Optimal Precision

Clearly, we would like a clock synchronization algorithm to obtain the best possible precision, that is, to bring the logical clocks to be as close together 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 admissible executions. This worst case interpretation follows the tradition of worst case complexity analysis of algorithms.

This definition suffers from 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. This is probably the reason that no clock synchronization algorithm was suggested for this very important class of systems.[3]

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

Formalizing this concept, however, is not simple. The major difficulty is finding a satisfying definition for executions where the message delivery system acts the same. The problem is that some properties of the execution are determined by the message deliv-

[3] Two previous works [14, 4] assume there is no upper bound on the delays. However, they assume the existence of a time-bounded broadcast primitive. In fact, they claim that clock synchronization is impossible in this model without some multicast primitive since there is no apriori bound on the precision achieved [4, page 589]. While this claim is true, we show it can be sidesteped by providing a bound on the precision achieved in each execution rather than an overall bound.

ery system and some by the algorithm. The algorithm controls the execution, e.g., by deciding when to send messages.[4] 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 executions of one algorithm with executions of another algorithm which are unfairly lucky, where message delays are favorable for the latter algorithm; a definition is too weak if executions with the same message delivery policy are not compared. We avoid 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.

Define a *correction function* to be a function from a set of n views to a vector of n real numbers, called *corrections*. Given a correction function 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.

To capture the precision achieved by some set of corrections $\vec{x} = \langle x_1, \ldots, x_n \rangle$ denote $\rho(\alpha, \vec{x}) = \max_{p,q} |(S_{\alpha,p} - x_p) - (S_{\alpha,q} - x_q)|$. That is, what is the largest discrepancy between two clocks of different processors, after the corrections are applied to them and $\text{offset}_{\alpha,p}$ takes the value x_p.

Since a correction function depends only on the views, we have:

Claim 3.1 *If $\alpha \equiv \alpha'$ then $f(\alpha) = f(\alpha')$.*

Because the computation of the corrections does not distinguish between equivalent executions, we measure its performance on a specific execution α, by considering the worst precision it achieved on all the executions equivalent to α. Formally, define for any execution $\alpha \in \mathcal{A}$, $\bar{\rho}_\alpha(\vec{x}) = \sup\{\rho(\alpha', \vec{x}) : \alpha' \equiv \alpha \text{ and } \alpha' \in \mathcal{A}\}$. A vector of corrections $\vec{x}$ is *optimal* for an execution α if for any vector of corrections $\vec{x}'$, $\bar{\rho}_\alpha(\vec{x}) \leq \bar{\rho}_\alpha(\vec{x}')$.

[4] 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.

A correction function f computes *optimal corrections* if, for every admissible execution α, $f(\alpha)$ is an optimal vector of corrections for α. Namely, a function 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 sometimes write $\bar{\rho}_\alpha(f)$ instead of $\bar{\rho}_\alpha(f(\alpha))$.

4 A General Clock Synchronization Algorithm

The basic difficulty of computing corrections is the fact that there may be two admissible executions α and α' that are indistinguishable, i.e. 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 start this section by formally quantifying this idea, by defining the maximal admissible shift between processors in a given execution. We show that if some information on the maximal admissible shifts is 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 admissible shifts. 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 obtain the information on the maximal admissible shifts needed by the algorithm for several specific systems.

4.1 Shifting

Consider two equivalent executions α and α'. It follows that for any $p \in V$, the sequence of steps in α' is equal to the sequence of steps in α, only that p executes the steps at different real times. Since the clocks have no drift, it follows that the difference in the real time of occurrence between 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:

Lemma 4.1 (Lundelius and Lynch) *Let π be a history of processor p and let s be a real number. Then shift(π, s) is a history of p and $S_{\alpha',p} = S_{\alpha,p} - s$.*

Let α and α' be two equivalent executions such that each processor $p \in V$ 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, denoted π, with *shift*(π, s_p), for each $p \in V$, and by retaining the same correspondence between sends and receives of messages. (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 $t - s_p$.) Namely, all tuples for processor p are shifted by s_p. We denote α' by *shift*(α, S). It is fairly simple to see that if $\alpha \equiv \alpha'$ then there exists a vector of shifts S such that $\alpha' =$*shift*(α, S).

4.2 Maximal Admissible Shifts

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

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

Intuitively, $\mathrm{ms}_\alpha(p, q)$ represents the *maximal admissible shift* of q w.r.t. p in α; that is, how far away can q be shifted from p. Since 0 is obviously an admissible shift of q w.r.t. p in α, it follows that $\mathrm{ms}_\alpha(p, q) \geq 0$. Clearly, we have:

Claim 4.2 *Let $S = \langle s_1, \ldots, s_n \rangle$ be a vector of shifts and $\alpha' =$shift(α, S) where $\alpha \in \mathcal{A}$. If $\alpha' \in \mathcal{A}$ then $s_q - s_p \leq \mathrm{ms}_\alpha(p, q)$, for any two processors p and q. This implies that $S_{\alpha',p} - S_{\alpha',q} \leq S_{\alpha,p} - S_{\alpha,q} + \mathrm{ms}_\alpha(p, q)$.*

4.3 The Lower Bound

Fix a system $(G, \mathcal{A})$, an admissible execution α of a clock synchronization algorithm, and a correction function f. The following lemma relates admissible shifts and attainable precision in clock synchronization algorithms (its proof appears in the full version).

Lemma 4.3 *Let p and q be arbitrary processors. If s is an admissible shift of q w.r.t. p in α, then $\bar{\rho}_\alpha(f) \geq S_{\alpha,p} - f(\alpha, p) - S_{\alpha,q} + f(\alpha, q) + s$.*

We now define an expression which we later show 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

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

Theorem 4.4 *For any correction function f, $\bar{\rho}_\alpha(f) \geq A_\alpha^{\max}$.*

Proof: Let $\theta = p_0, \ldots, p_k$ be an arbitrary cyclic sequence of processors. For every i, $0 \leq i \leq k-1$, let s be an arbitrary admissible shift of p_{i+1} w.r.t. p_i in α. Lemma 4.3 implies that

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

Since this holds for every admissible shift, the definition of $\mathrm{ms}_\alpha(p_i, p_{i+1})$ implies that

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

Summing over all the consecutive processors in θ, we have

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

Clearly,

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

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. ■

Observe that $A_\alpha^{\max}$ depends only on the values of $\mathrm{ms}_\alpha(p, q)$, for every pair of processors p and q; the topology of the graph is not explicitly used.

4.4 The Upper Bound

We now show the converse direction, i.e. that there exists a correction function f with $\bar{\rho}_\alpha(f) = A_\alpha^{\max}$, for every α, provided certain estimates can be computed from the views. By Theorem 4.4, no other correction function can achieve better results and hence the function we present is optimal, in a very strong sense.

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 such estimates for specific systems.

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

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

In the full version we prove:

Lemma 4.5 $A_\alpha^{\max} = \tilde{A}_\alpha^{\max}$.

Thus, we have the following function SHIFTS, for computing corrections, given inputs $\tilde{\mathrm{ms}}_\alpha(p,q) = \mathrm{ms}_\alpha(p,q) + S_{\alpha,p} - S_{\alpha,q}$, 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 $p \in V$, is $\mathrm{dist}_w(r,p)$—the distance in the (complete) graph relative to the weights $w(p,q) = A_\alpha^{\max} - \tilde{\mathrm{ms}}_\alpha(p,q)$.

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 4.5, this is equivalent to computing $A_\alpha^{\max}$. By definition, $A_\alpha^{\max} \geq A_\alpha = \mathrm{ms}_\alpha(\theta)/|\theta|$, for any cycle θ. Therefore, there are no negative weight cycles in the graph respective to the weights $w(p,q) = (A_\alpha^{\max} - \tilde{\mathrm{ms}}_\alpha(p,q))$, since

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

Thus, the distances can be computed in Step 2.

Theorem 4.6 *The function* SHIFTS *computes optimal corrections, with precision given by* $\bar{\rho}_\alpha = A_\alpha^{\max}$.

Proof: Denote by $f(\alpha)$ the vector of corrections computed by SHIFTS when given $\tilde{\mathrm{ms}}_\alpha(p,q)$. We will show that $\bar{\rho}_\alpha(f) \leq A_\alpha^{\max}$; it follows from Theorem 4.4, that these are optimal corrections and that $\bar{\rho}_\alpha = A_\alpha^{\max}$.

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 $\alpha' \equiv \alpha$. That is, we need to show that for every $p, q \in V$,

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

By Claim 4.2,

$$S_{\alpha',p} - S_{\alpha',q} - f(\alpha', p) + f(\alpha', q) \leq S_{\alpha,p} - S_{\alpha,q} + \mathrm{ms}_\alpha(p,q) - f(\alpha', p) + f(\alpha', q) .$$

However, since $\alpha \equiv \alpha'$, it follows from Claim 3.1 that $f(\alpha) = f(\alpha')$, and hence the right hand side is equal to

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

Thus, it suffices to prove that

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

This amounts to proving that for any two processors p and q,

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

By the definition of estimated maximal admissible shifts, $\tilde{\mathrm{ms}}_\alpha(p,q) = \mathrm{ms}_\alpha(p,q) + S_{\alpha,p} - S_{\alpha,q}$, that is, $S_{\alpha,p} - S_{\alpha,q} = \tilde{\mathrm{ms}}_\alpha(p,q) - \mathrm{ms}_\alpha(p,q)$, hence it suffices to prove that

$$A_\alpha^{\max} - \mathrm{ms}_\alpha(p,q) \geq \tilde{\mathrm{ms}}_\alpha(p,q) - \mathrm{ms}_\alpha(p,q) + f(\alpha,q) - f(\alpha,p) \quad (1)$$

By the definition of the function SHIFTS,

$$f(\alpha,q) = \mathrm{dist}_w(r,q) \text{ and } f(\alpha,p) = \mathrm{dist}_w(r,p) ,$$

relative to the weights $w(p,q) = (A_\alpha^{\max} - \tilde{\mathrm{ms}}_\alpha(p,q))$. By the triangle inequality,

$$\mathrm{dist}_w(r,p) + w(p,q) \geq \mathrm{dist}_w(r,q) ,$$

which implies

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

To prove Equation (1) note that

$$\begin{aligned} &\tilde{\mathrm{ms}}_\alpha(p,q) - \mathrm{ms}_\alpha(p,q) + f(\alpha,q) - f(\alpha,p) \\ &\quad \leq \tilde{\mathrm{ms}}_\alpha(p,q) - \mathrm{ms}_\alpha(p,q) + A_\alpha^{\max} - \tilde{\mathrm{ms}}_\alpha(p,q) \\ &\quad = A_\alpha^{\max} - \mathrm{ms}_\alpha(p,q) \end{aligned}$$

■

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 admissible shifts. Given such estimates, the clock synchronization problem can be solved by computing the function SHIFTS.

5 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 admissible 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 between a specific pair of processors (along edges interconnecting them) do not depend on the delays of messages sent between 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 for several systems the local estimates given the views.

The same computation applies to all local systems. Therefore, 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 cases solved in the next section, the calculation of local estimates 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 towards the end of this section.

5.1 Local Systems

Informally, a system is local if its behavior can be expressed as the intersection of a set of locally admissible histories, e.g., one for each pair of processors sharing an edge.

In more detail, let $\mathcal{A}_{p,q}$ be a set of (unordered) pairs of histories, one for processor p and one for processor q. Let α be an execution, and let $\alpha|_{p,q}$ be the (unordered) pair consisting of the history of p in α and the history of q in α. We say that α is *locally admissible* on $\{p,q\}$ w.r.t. $\mathcal{A}_{p,q}$ if $\alpha|_{p,q}$ is in $\mathcal{A}_{p,q}$. When $\mathcal{A}_{p,q}$ is obvious from the context, we say simply that α is *locally admissible* on $\{p,q\}$.

We say that s is a *locally admissible shift of* q w.r.t. p in α, if there exists a vector of shifts $S = \langle s_1, \ldots, s_n \rangle$, with $s_q - s_p = s$, such that $\alpha' = \mathit{shift}(\alpha, S)$ is locally admissible on $\{p,q\}$. If S is a vector of shifts such that $\alpha' = \mathit{shift}(\alpha, S)$ is locally admissible on $\{p,q\}$, then for every $S' = \langle s'_1, \ldots, s'_n \rangle$ such that $s'_p = s_p$ and $s'_q = s_q$, $\mathit{shift}(\alpha, \langle s'_1, \ldots, s'_n \rangle)$ is also locally admissible on $\{p,q\}$. This follows since $\mathcal{A}_{p,q}$ involves only the histories of p and q are involved.

A set of pairs of histories, $\mathcal{A}_{p,q}$ is *closed under constant shifts* provided that if $\{\pi_p, \pi_q\}$ is in $\mathcal{A}_{p,q}$ then $\{\mathit{shift}(\pi_p, c), \mathit{shift}(\pi_q, c)\}$ is in $\mathcal{A}_{p,q}$, for any constant c.

Claim 5.1 *If $\mathcal{A}_{p,q}$ is closed under constant shifts, then s is a locally admissible shift of q w.r.t. p in α if and only if for every vector $S = \langle s_1, \ldots, s_n \rangle$, with $s_q - s_p = s$, $\mathit{shift}(\alpha, S)$ is locally admissible on $\{p,q\}$.*

A set of admissible executions $\mathcal{A}$ is *local* if there are sets $\mathcal{A}_{p,q}$, closed under constant shifts, such that an execution α is in $\mathcal{A}$ if and only if $\alpha|_{p,q}$ is in $\mathcal{A}_{p,q}$, for every set $\mathcal{A}_{p,q}$. Namely, execution α is locally admissible on $\{p,q\}$, for every pair of processors p and q. We say that $\mathcal{A}$ is local w.r.t. $\{\mathcal{A}_{p,q}\}$.

Define the *maximal local shift of q w.r.t. p in α* under $\mathcal{A}$ 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, q need not be preserved. Hence, $\text{ms}_\alpha(p,q) \leq \text{mls}_\alpha(p,q)$. We say that $\text{mls}_\alpha(p,q)$ is a *local shift*, while $\text{ms}_\alpha(p,q)$ is a *global shift*. Note that $\text{mls}_\alpha(p,q) \geq 0$ and that $\text{mls}_\alpha(p,q)$ may differ from $\text{mls}_\alpha(q,p)$. However, if $\mathit{shift}(\alpha, \langle s_1, \ldots, s_n \rangle)$ is locally admissible on $\{p,q\}$ then $s_q - s_p$ is an admissible local shift of q w.r.t. p and $s_p - s_q$ is a admissible local 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, and 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 *For every two processors p, q, and every $\alpha \in \mathcal{A}$, if x, y s.t. $y < x$ 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.*

5.2 From Local Shifts to Global Shifts

Our goal is to compute global estimates $\tilde{\text{ms}}_\alpha(p,q)$ from local estimates $\tilde{\text{mls}}_\alpha(p,q)$. In this section, as a first step in this direction, we show how to obtain maximal global shifts $\text{ms}_\alpha(p,q)$ from maximal local shifts $\text{mls}_\alpha(p,q)$. 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. In the full version we prove the following lemma.

Lemma 5.2 *Let $S = \langle s_1, \ldots, s_n \rangle$ be a vector of shifts and let $\alpha \in \mathcal{A}$. Then S is an admissible vector of shifts for α, 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.*

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

Lemma 5.3 *For any pair of processors p and q, $dist_{w'}(p,q) = \text{ms}_\alpha(p,q)$.*

Proof: We show only that $\text{dist}_{w'}(p,q) \leq \text{ms}_\alpha(p,q)$; the other inequality is easier and appears in the full version.

We show that for any $\gamma > 1$, $\frac{\text{dist}_{w'}(p,q)}{\gamma}$ is a (globally) admissible shift of q w.r.t. p in α. The lemma follows since if we assume, by way of contradiction, that for some pair of processors p and q, $\text{dist}_{w'}(p,q) > \text{ms}_\alpha(p,q)$, it follows that there exists some $\gamma > 1$ such that $\text{dist}_{w'}(p,q) = \gamma^2 \cdot \text{ms}_\alpha(p,q)$. But we have shown that $\frac{\text{dist}_{w'}(p,q)}{\gamma} = \gamma \cdot \text{ms}_\alpha(p,q)$ is a (globally) admissible shift of q w.r.t. p in α. Since $\gamma \cdot \text{ms}_\alpha(p,q) > \text{ms}_\alpha(p,q)$, this contradicts the definition of $\text{ms}_\alpha(p,q)$.

To prove that $\frac{\text{dist}_{w'}(p,q)}{\gamma}$ is a (globally) admissible shift of q w.r.t. p in α, note that for any two processors k, j, by the triangle inequality

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

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

$$\text{dist}_{w'}(p,j) - \text{dist}_{w'}(p,k) \leq \text{mls}_\alpha(k,j) \ . \qquad (2)$$

For every processor i, define $s_i = \frac{\text{dist}_{w'}(p,i)}{\gamma}$. Since $\gamma > 1$, by dividing Equation (2) by γ and substituting s_j and s_k, we get

$$s_j - s_k \leq \frac{\text{mls}_\alpha(k,j)}{\gamma} < \text{mls}_\alpha(k,j) \qquad (3)$$

$$\text{and} \quad s_k - s_j \leq \frac{\text{mls}_\alpha(j,k)}{\gamma} < \text{mls}_\alpha(j,k) \ ,$$

which implies

$$-\text{mls}_\alpha(j,k) < -\frac{\text{mls}_\alpha(j,k)}{\gamma} \leq s_j - s_k \qquad (4)$$

By Assumption 1, $\frac{\text{mls}_\alpha(k,j)}{\gamma}$ is a locally admissible shift of j w.r.t. k and $\frac{\text{mls}_\alpha(j,k)}{\gamma}$ is a locally admissible shift of k w.r.t. j. Since the system is local, Assumption 1 and the definition of mls imply that $-\frac{\text{mls}_\alpha(j,k)}{\gamma}$ is a locally admissible shift of j w.r.t. k. Thus, Assumption 1 and Equations (3) and (4) imply that $s_j - s_k$ is a locally admissible shift of j w.r.t. k in α. Lemma 5.2 implies that $\langle s_1, \ldots, s_n \rangle$ is a (globally) admissible shift vector of α. In particular, since $s_p = 0$, it follows that $s_q = \frac{\text{dist}_{w'}(p,q)}{\gamma}$ is a (globally) admissible shift of q w.r.t. p in α. ■

Since $\text{dist}_{w'}(p,q)$ depends only on $\text{mls}_\alpha(p,q)$, we get:

Theorem 5.4 *For any admissible execution α and any two processors p and q, $\text{ms}_\alpha(p,q)$ can be computed from $\text{mls}_\alpha(p,q)$.*

5.3 Using Estimates for Local Shifts

Now, the issue is how to compute the values $\tilde{\text{ms}}_\alpha(p,q)$ needed as inputs for 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 $\tilde{\text{mls}}_\alpha(p,q) = \text{mls}_\alpha(p,q) + S_{\alpha,p} - S_{\alpha,q}$, for every pair of processors p and q.

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

Theorem 5.5 *The function* GLOBAL ESTIMATES *computes $\tilde{\text{ms}}_\alpha(p,q)$, for every pair of processors p and q.*

Proof: Observe that the weight of any cycle w.r.t. the weights $\tilde{\text{mls}}_\alpha$ 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 with weights $\tilde{\text{mls}}_\alpha$. Also, the weight of any path from p to q w.r.t. weights $\tilde{\text{mls}}_\alpha$ 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 5.4. ■

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 $\tilde{\text{mls}}_\alpha$. This follows immediately from Theorem 5.5 above, together with Theorem 4.6.

5.4 A Decomposition Theorem

In the full paper, we prove a decomposition theorem that allows us to find optimal corrections for systems with multiple local restrictions on the delays, by combining the solution for each restriction.

Theorem 5.6 *Consider two sets of admissible executions, $\mathcal{A}'$ which is local w.r.t. $\mathcal{A}'_{p,q}$ and $\mathcal{A}''$ which is local w.r.t. $\mathcal{A}''_{p,q}$. Let $\mathcal{A}_{p,q}$ be the intersection of $\mathcal{A}'_{p,q}$ and $\mathcal{A}''_{p,q}$, and let $\mathcal{A}$ be the corresponding set of admissible executions. For any execution $\alpha \in \mathcal{A}$, let $\mathrm{mls}_\alpha(p,q)$, $\mathrm{mls}'_\alpha(p,q)$ and $\mathrm{mls}''_\alpha(p,q)$ be the maximal local shift of q w.r.t. p in α under, respectively, $\mathcal{A}$, $\mathcal{A}'$ or $\mathcal{A}''$. Then $\mathrm{mls}_\alpha(p,q) = \min\{\mathrm{mls}'_\alpha(p,q), \mathrm{mls}''_\alpha(p,q)\}$.*

6 Clock Synchronization for Specific Delay Assumptions

We now show how to compute optimal corrections based on the views of the processors, in systems which satisfy two types of timing assumptions; both assumptions are reasonable in realistic systems. The first assumption is lower and upper bounds on the delays (including infinite bounds). The second assumption is a bound on the difference between the delay in two directions of a link. In both cases, the system is local and optimal corrections are computed using the results of the previous sections. By Theorem 5.5, all we have to show is how to compute the estimates of the maximal local shifts $\tilde{\mathrm{mls}}_\alpha(p,q)$. This calculation is based on estimates for the delays (defined below), which can be easily computed from the views of the processors during the execution.

The *estimated delay* $\tilde{\mathrm{d}}(m)$ of 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 in flavor 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. (The proof appears in the full version.)

Lemma 6.1 *Given the views v_p, v_q of processors p and q during execution α, it is possible to compute the estimated delay $\tilde{\mathrm{d}}(m)$ of any message m sent from p to q.*

6.1 Bounds on the Delay

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

Consider sets $\mathcal{A}_{p,q}[lb, ub]$ of pairs of histories for each edge (p,q), defined as follows. We associate a pair of numbers $ub(p,q)$ and $lb(p,q)$ such that $0 \leq lb(p,q) \leq ub(p,q) \leq \infty$. For every execution α, the pair of histories $\alpha|_{p,q}$ (of p and q) is in $\mathcal{A}_{p,q}[lb, ub]$ if the delay of every message sent from p to q is in the range $[lb(p,q), ub(p,q)]$ and the delay of every message from q to p is in the range $[lb(q,p), ub(q,p)]$. Clearly, the local sets $\mathcal{A}_{p,q}$ are closed under constant shifts.

Define a set of admissible executions $\mathcal{A}[lb, ub]$ which includes all executions α such that, for each edge (p,q), the pair of histories $\alpha|_{p,q}$ is in $\mathcal{A}_{p,q}[lb, ub]$. Obviously this set of executions is local. Observe that an execution α is admissible if and only if for each edge (p,q), the delay of every message sent in α from p to q is in the range $[lb(p,q), ub(p,q)]$.

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

Lemma 6.2 *Let α be an admissible execution of system $(G, \mathcal{A}[lb, ub])$. Then $\mathrm{mls}_\alpha(p,q) = \min\{(ub(q,p) - \mathrm{d}^{\max}_\alpha(q,p)), (\mathrm{d}^{\min}_\alpha(p,q) - lb(p,q))\}$.*

Lemma 6.2 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 6.3 *Let α be an admissible execution of $(G, \mathcal{A}[lb, ub])$. Then $\tilde{\mathrm{mls}}_\alpha(p,q) = \min\{(ub(q,p) - \tilde{\mathrm{d}}^{\max}_\alpha(q,p)), (\tilde{\mathrm{d}}^{\min}_\alpha(p,q) - lb(p,q))\}$.*

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

Corollary 6.4 *Let α be an admissible execution of a local system $(G, \mathcal{A})$. Then $\mathrm{mls}_\alpha(p,q) \leq \mathrm{d}^{\min}_\alpha(p,q)$ and $\tilde{\mathrm{mls}}_\alpha(p,q) \leq \tilde{\mathrm{d}}^{\min}_\alpha(p,q)$.*

6.2 Links with Bounds on the Round Trip Delay Bias

In many bidirectional 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 on the other 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 other direction. We now show how to calculate maximal local shifts in systems where there is a bound on the difference between the delay in one direction and delay in the other direction of a link.

For the purpose of this paper, we simplify the assumption and require that the difference between the delay of any two messages in opposite directions is bounded. It is possible to generalize our results to the more realistic model in which this assumption holds only for messages that were sent "around the same time."

In more detail, we associate a positive number $b(p,q) = b(q,p)$, with each edge (p,q). Denote the delay of message m by $\mathrm{d}(m)$. A pair of histories for p and q is in $\mathcal{A}_{p,q}[b]$ if for every message m_p received by p from q, and every message m_q received by q from p, $|\mathrm{d}(m_p) - \mathrm{d}(m_q)| \leq b(p,q)$. We also restrict $\mathcal{A}_{p,q}[b]$ to non-negative delays, i.e., for every message m, $d(m) \geq 0$. This obviously defines a local set of admissible executions $\mathcal{A}[b]$, where $\alpha \in \mathcal{A}[b]$ if and only if $\alpha \in \mathcal{A}_{p,q}[b]$ for each edge (p,q). The next lemma shows that, as for the case of lower and upper bounds, $\mathrm{mls}_\alpha(p,q)$ depends only on the maximal and minimal delays between p and q.

Lemma 6.5 *Let α be an admissible execution of $(G, \mathcal{A}[b])$. Then*

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

Proof: Consider the following two local sets of admissible executions. The first set $\mathcal{A}'$ contains every execution α such that the delay of each message in α is non-negative. The second set $\mathcal{A}''$ is like $\mathcal{A}[b]$ except that the delays are allowed to be negative. Clearly $\mathcal{A}[b]$ is the intersection of $\mathcal{A}'$ and $\mathcal{A}''$, and thus, Theorem 5.6 implies that $\mathrm{mls}_\alpha(p,q) = \min\{\mathrm{mls}'_\alpha(p,q), \mathrm{mls}''_\alpha(p,q)\}$. By Lemma 6.2, $\mathrm{mls}'_\alpha(p,q) = \mathrm{d}_\alpha^{\min}(p,q)$, and thus we only have to prove that $\mathrm{mls}''_\alpha(p,q) = \frac{b(p,q)+\mathrm{d}_\alpha^{\min}(p,q)-\mathrm{d}_\alpha^{\max}(q,p)}{2}$.

Let π_p denote p's history in α and π_q denote q's history in α. For any $s_q \geq 0$ let $\pi_q^s = \mathit{shift}(\pi_q, s_q)$. In (π_p, π_q^s), the delay $d_{(\pi_p,\pi_q^s)}(m_p)$ of any message m_p from q to p is s_q more than the delay $d_\alpha(m_p)$ of the same message in α; similarly, the delay of any message m_q from p to q, $d_{(\pi_p,\pi_q^s)}(m_q)$ is s_q less than the delay, $d_\alpha(m_q)$ of the same message in α. It follows that

$$\begin{aligned} d_{(\pi_p,\pi_q^s)}(m_q) - d_{(\pi_p,\pi_q^s)}(m_p) &= \\ & d_\alpha(m_q) - d_\alpha(m_p) - 2s_q\ , \\ d_{(\pi_p,\pi_q^s)}(m_p) - d_{(\pi_p,\pi_q^s)}(m_q) &= \\ & d_\alpha(m_p) - d_\alpha(m_q) + 2s_q\ . \end{aligned}$$

Therefore,

$$\begin{aligned} d_\alpha(m_q) - d_\alpha(m_p) - 2s_q &\leq b(p,q) \\ \text{and}\quad d_\alpha(m_p) - d_\alpha(m_q) + 2s_q &\leq b(p,q) \end{aligned}$$

if and only if the round trip delay bias in (π_p, π_q^s) of m_p and m_q is at most $b(p,q)$. Since α is admissible and $s_q \geq 0$, the first inequality trivially holds. Hence, both inequalities hold for every $s_q \geq 0$ such that

$$s_q \leq \frac{b(p,q) + d_\alpha(m_q) - d_\alpha(m_p)}{2}.$$

The inequalities hold for every pair of messages simultaneously, for every $s_q \geq 0$ such that

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

Namely, this is the set of locally admissible shifts of q w.r.t. p in α relative to $\mathcal{A}''$. We conclude that $\mathrm{mls}''_\alpha(p,q) = \frac{b(p,q)+\mathrm{d}_\alpha^{\min}(p,q)-\mathrm{d}_\alpha^{\max}(q,p)}{2}$. ∎

Corollary 6.6 *Let α be an admissible execution of $(G, \mathcal{A}[b])$. Then*

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

7 Discussion and Open Questions

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 for the following models: upper and lower bounds on delays are known; only lower bounds on the delays are known; no bounds are known; and only a bound on the difference of the round trip delays is known. Moreover, the results apply to cases where different links satisfy different assumptions. Our results extend and simplify results of Halpern, Megiddo and Munshi [3]. Our results are based on a new notion of optimality in any specific instance.

In this paper, we only address the issue of computing optimal corrections, given (somehow) 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 $\tilde{mls}_\alpha(p,q)$ and $\tilde{mls}_\alpha(q,p)$ 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.

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

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

Acknowledgments: We thank Joe Halpern 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." *CACM*, 21:7 (July 1978), pp. 558–565.

[8] L. Lamport and P. Melliar-Smith, "Synchronizing clocks in the presence of faults," *JACM*, 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 *Proc. 10th ACM Symp. on Principles of Distributed Computing*, 1991, pp. 1–9.

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

[14] 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.

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

[16] T. Srikanth and S. Toueg, "Optimal Clock Synchronization," *JACM*, 34:3 (July 1987), pp. 626–645.

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