A Theory of Clock Synchronization

Extended Abstract

Boaz Patt-Shamir* Sergio Rajsbaum†

Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02139

Abstract

We consider the problem of clock synchronization with uncertain message delays and bounded clock drifts. To analyze this classical problem we introduce a characterization theorem for the tightest achievable estimate of the readings of a remote clock in any given execution of the system. Using this theorem, we obtain the first optimal on-line distributed algorithms for clock synchronization. The algorithms are optimal for all executions, rather than only worst cases. The general algorithm for systems with drifting clocks has high space overhead, which is unavoidable, as we show. For systems with drift-free clocks (i.e., clocks that run at the rate of real time), we present a remarkably simple and efficient algorithm. The discussion focuses on the variant where one of the clocks shows real time, but we present results also for the case where real time is not available from within the system. Our approach encompasses various models, such as point-to-point or broadcast channels, perfect or faulty communication links, and it has fault-detection capablities.

*Supported by Air Force Contract AFOSR F49620-92-J-0125, ONR contract N00014-91-J-1046, NSF contracts CCR-9114440 and 9225124, DARPA contracts N00014-92-J-1799 and N00014-92-J-4033, and a special grant from IBM. E-mail: `boaz@theory.lcs.mit.edu`.

†E-mail: `rajsbaum@theory.lcs.mit.edu`. On leave from Institute of Mathematics, U.N.A.M. Partly supported by DGAPA Projects.

1 Introduction

Clock synchronization is one of the most basic problems in distributed computing. Roughly speaking, the goal is to ensure that physically dispersed processors will acquire a common notion of time, using local physical clocks (whose rates may vary), and message exchange over a communication network (with uncertain transmission delays). In this work, we consider the following simple formulation of the problem: *Obtain, at all times, the smallest interval* $[a, b]$ *such that the current reading of a designated clock is in* $[a, b]$.[1]

Generally speaking, the task of clock synchronization has two main variants. In the *external synchronization* problem, we are given a *source clock* that shows real time, and the goal of all processors is to estimate that time as tightly as possible. The external clock synchronization task is especially useful in loosely coupled networks: for instance, the NTP protocol is used for external synchronization of the Internet [11]. In the *internal synchronization* problem, real time is not available from within the system, and the goal is to minimize the maximum difference between the readings of clocks, subject to some liveness condition.

The basic difficulty is that synchronization tends to deteriorate over time and space. When the local clocks are not *drift-free* (i.e., they do not run at the rate of real time), the synchronization loosens unless timing information is refreshed periodically. And when communicating information to distant processors across non-deterministic links, there is some inherent added timing uncertainty, due to unpredictable message delays. However, usually there exist some *a priori* bounds on these uncertainties that can be used for synchronization.

In addition to message delays and clock rates, many other parameters of the system can be consid-

[1] In this paper, numbers range over $\mathbf{R} \cup \{-\infty, \infty\}$ unless explicitly indicated otherwise.

ered: inaccurate source of real-time, point-to-point or broadcast channels, link reliability (are links lossy; order-preserving; duplicating), and processor reliability (may processors crash; exhibit Byzantine behavior). The large number of models is justified by the wide spectrum of applications.

Considerable work has been dedicated to clock synchronization (see surveys [14, 13] and more recent works, e.g. [1, 5, 2, 12] and references therein). Some algorithms were proposed for systems where processors and links are reliable, but both delays and clock drifts are uncertain (e.g. [7, 10]). Although the algorithms are simple to state, their analyses tend to be complicated. Moreover, even for this case, no nontrivial lower bounds were known.

The case of systems whose clocks are drift-free is better understood [8, 6, 2]. In [8, 6], the worst possible behaviors of the system (within its specifications) are analyzed, and optimal protocols for the worst case are proposed. The protocols send one message per link, because in the worst case no additional information is gained by sending more messages. Recently, Attiya et al. [2] proposed not viewing the system as an adversary; the problem they consider is how to get the tightest synchronization for any given execution (which often is much better than the worst possible). They analyze this setting using the viewpoint of an external observer, that has instantaneous access to all the information in the system. This approach leads to tight synchronization bounds per execution, but leaves unanswered the question of tight *on-line* bounds. In other words, no bounds were known for the achievable synchronization in terms of locally observable behavior and system specifications.

In this work, we take the execution-based approach further. We present a characterization of the best achievable on-line synchronization between any two events in a given execution. This characterization enables us to derive a series of results as follows. We start with the first matching upper and lower bounds for on-line external synchronization. These bounds hold in the general case, where clocks are allowed to drift. The algorithm used to prove the upper bound has unbounded space overhead. However, we prove that in a suitably defined variant of the comparison branching program model [3], the space complexity of any optimal algorithm for drifting clocks cannot be bounded by a function of the network size. The situation for systems with drift-free clocks is much better: for this case, we give an extremely simple and efficient protocol for external synchronization. For the internal synchronization problem, our algorithms (using a trivial reduction) give tightness which is at most twice the optimal. Additionally, we derive a lower bound for internal synchronization that generalizes the known bound [6, 2] to the case of drifting clocks. Finally, we discuss a few extensions of the basic model, including a way to incorporate additional *a priori* knowledge about the message traffic pattern; a simple way to detect faults; and how to deal with inaccurate source clocks. We remark that our results apply to most of the system variants mentioned above, excluding the models that allow corruption of message contents or Byzantine failures of the processors.

The theory we introduce in this paper relies on a novel concept which we call a *synchronization graph.* The synchronization graph of a given execution is a weighted directed graph, where each point corresponds to a single event, and the weights of the arcs express somehow uncertainty bounds between their incident points. Intuitively, the synchronization graph represents the topology of timing uncertainty in the given execution, and the basic idea is to reduce computations of uncertainty intervals to distance computations in the synchronization graph. We suspect that synchronization graphs will prove useful in the analysis of other timing-based protocols.

The rest of this paper is organized as follows. In Section 2 we informally describe the system model. In Section 3 we derive our main characterization result. In Section 4 we analyze the external synchronization problem. In Section 5 we give a general lower bound for internal synchronization. We conclude in Section 6 with a few extensions of the basic theory.

Due to lack of space, most proofs are omitted.

2 Overview of the Model

In this section we give a high-level description of the system structure for the clock synchronization problem. The underlying formal model is based on the *timed I/O Automata* model of Lynch and Vaandrager [9]. The system is modeled as a collection of interacting state machines with input and output actions. The model associates real time, denoted *real_time*, with every state and event. Real time is not directly observable by the processors. The first step of each processor occurs at arbitrary real time, so that the real time of the start of the execution cannot be used for synchronization.

Intuitively, the clock synchronization problem is to provide the user at a processor with a *logical clock*, such that the local state of the processor encodes information of the type "the logical time at state s is $T \pm \varepsilon$." Below, we outline our model for this problem, whose motivation is to facilitate comparison be-

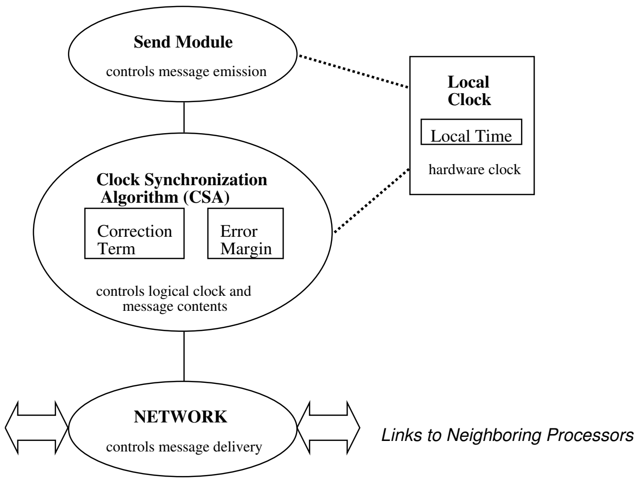

Figure 1: *The modules and interfaces at one processor of a clock synchronization system.*

tween different algorithms for clock synchronization (see also [2]). The ultimate goal is that algorithms would perform optimally in each execution; the basic difficulty is that different algorithms may never share the same execution. For instance, some trivial algorithm that does not send any message could be considered optimal, in some degenerate sense. Our aim is to compare algorithms on "equal grounds" insofar as the available information is concerned. The idea in the following model is to take the message traffic patterns (including generating messages) out of the control of the synchronization algorithm, and thus limit the problem of synchronization to make the best use of these patterns.

More specifically, the system consists of a communication network, where each processor consists of the three distinct modules as follows (see Figure 1).

The *local clock* module represents the hardware clock at the processor. It encodes, at all states s of processor v, a real number called *local time*, denoted $loc\_time_v(s)$.[2] We are mainly interested in *bounded-drift* and *drift-free* clocks. A drift-free clock is assumed to run at the rate of real time, but does not necessarily shows real time. A bounded-drift clock is assumed to have known lower and upper bounds on the rate of its progress relative to real time, denoted $\underline{\varrho}_v$ and $\overline{\varrho}_v$, respectively. We shall assume w.l.o.g. that $\underline{\varrho} \leq 1 \leq \overline{\varrho}$. (e.g., for a drift-free clock, $\underline{\varrho} = \overline{\varrho} = 1$). Our model does not allow adjustment of local clocks, but only of the logical clocks. This assumption is made for notational convenience only.

The *network* module connects processors to their

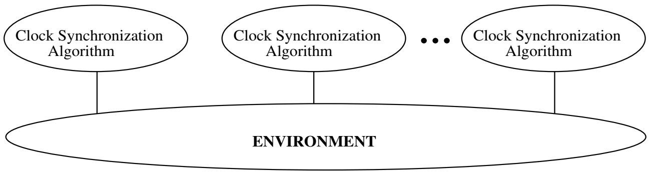

Figure 2: *The conceptual arrangement of CSA modules at a clock synchronization system.*

neighbors. The network may lose, duplicate, and deliver messages arbitrarily out of order.[3] For simplicity, we assume that all sent messages are unique.

We assume, however, that any message received was indeed sent. We assume that the real time delay of each message m is within some known bounds, $0 \leq L(m) \leq H(m) \leq \infty$. These bounds may vary from link to link and from message to message. For the optimality results, we shall assume that the bounds are tight, in the sense that all delays in $[L(m), H(m)]$ are possible for a message m.

The *send module* at a processor determines when to send the messages and to which neighbors. The send module isolates the message generation part, so that the synchronization achieved can be evaluated with respect to a given message pattern. For the most part of this paper, we assume that the send module is completely unstructured, that is, a message may be generated at any time.

The *Clock Synchronization Algorithm (CSA)* uses the readings of the local clock, the messages received, and the specification of the system to compute a *correction term* Δ and an *error margin*, ε. The logical time is (T, ε), where $T = loc\_time + \Delta$. (The semantics of the logical time depends on the specific variant of the problem; see Sections 4 and 5.) We stress that CSAs do not initiate the sending of any message. Informally, think of the CSA as a filter that "sticks" a few extra bytes to each outgoing message, and "strips" the corresponding bytes of incoming messages, without changing the timing of the messages. In other words, a CSA may use the existing message pattern to obtain timing information and exchange data with remote CSA modules, but it may not "meddle" with the ongoing traffic otherwise.

Our next step is to view the aggregate of the network, send modules and local clocks of the whole system as a single *environment* automaton (see Figure 2). First, define the *real-time specifications* of a system to consist of the bounds on clock rates and message delays. An environment automaton governs the local clocks and the message traffic pattern, subject

[2] In this work we often omit subscripts when they are clear from the context.

[3] It turns out that assuming lossless channels may have a significant impact on the analysis. See Section 6.

to the given real-time specifications. Given an execution of the environment, its *pattern* is the sequence of all send and receive events, with their *loc_time* and *real_time* mappings. Given a pattern, its *view* is obtained by omitting the real time mapping (but retaining local times). An event of a pattern or a view is called a *point*.

Using the above concepts, we summarize the fundamental assumption of our model as follows.

Lemma 2.1 *If all message delays and all local clocks in a pattern α respect the real-time specification of an environment A, then α is a pattern of A.*

For on-line analysis we define the following notion. The *local view of a pattern α at a point p* is the view containing the set of all points of α that can influence p.[4] The output of a CSA (like any other distributed algorithm) is required to be a function of the local view of the pattern at the point of output.

We shall say that a CSA is *optimal* for a given environment automaton A if, at any point of any pattern of A, the logical time satisfies the correctness requirement (of the specific problem), and the error margin is the smallest of all correct CSA algorithms for A at that point.

To capture the efficiency of an algorithm, we use the notion of *communication overhead* to denote the maximal amount of extra information added by a CSA to the ongoing message traffic. We shall consider also the *space overhead* of the CSA, defined in Section 4.2.

Remark. As mentioned above, we distinguish between the send module and the CSA module for the purpose of comparing different algorithms. This approach can be used in other cases to compare the quality of different protocols on "equal grounds." We also believe that this kind of "hitch-hiking" philosophy may be of practical interest for tasks where there is always something to be done (e.g., topology-maintenance protocols). The assumption that data can be appended and extracted from a message without affecting its timing serves as a convenient abstraction of reality, that can be justified if the computation and communication overhead involved are negligible. It is interesting to note that in many networks, the message delivery system is already appending "headers" to messages to facilitate delivery.

[4]Formally (cf. [7]), we say that a point q can influence another point q' if either q occurs before q' at the same processor, or if q is a send point and q' is a receive point of the corresponding message, or if there is a point q'' such that q can influence q'' and q'' can influence q'.

3 The Basic Theorem

In this section we analyze the model outlined in Section 2. Throughout this section we shall consider an arbitrary view β of a fixed environment, and study the patterns with view β that satisfy some given real time specifications. The main result of this section is Theorem 3.6, that characterizes the tightest achievable synchronization in terms of the given view and real-time specifications.

We start with some notations. Let p and q be arbitrary points of pattern α with the given view β. We define the *actual* and *virtual* delay of p relative to q as follows.

$$\begin{aligned} act\_del_\alpha(p,q) &= real\_time_\alpha(p) - real\_time_\alpha(q) \\ virt\_del(p,q) &= loc\_time(p) - loc\_time(q) \end{aligned}$$

Notice that given a view, only virtual delays are defined.

Next, we model the view β as a *view graph* $\Gamma_\beta = (V,E)$. To do that, we call a pair of points *adjacent*, if either one is a send event and the other is the receive event of the corresponding message, or if the two points represent two consecutive events at the same processor. Now, V is defined to be the set point of β, and in E there are two antisymmetrical directed arcs joining each pair of adjacent points of β. We use the terms "points" and "arcs" for view graphs to avoid confusion with "processors" and "links" of the communication network.

Our next step is to model the real time specifications of the environment automaton. These specifications are essentially bounds on the difference between the real time of occurrence of two adjacent points of a view. We represent them formally with the following type of functions.

Definition 3.1 (Bounds Mapping)
A bounds mapping *for β is a function B that maps every pair p,q of adjacent points in β to a number $B(p,q) > -\infty$. A pattern α with view β is said to* satisfy B *if $act\_del_\alpha(p,q) \le B(p,q)$ for every pair of adjacent points p,q.*

Notice that Definition 3.1 implies that if α satisfies B then for any two adjacent points p,q we have

$$act\_del(p,q) \in [-B(q,p), B(p,q)] \ .$$

This implies that for α to satisfy B, it is necessary that $B(p,q) + B(q,p) \ge 0$.

We remark that bounds mappings model timing assumptions in a general way: drift bounds of a clock may change over time, and delivery time bounds may vary from message to message (even on the same link). Moreover, bounds mappings model all timing assumptions uniformly, in the sense that there is no substantial difference between the bounds on communication delays and the bounds on drifting clocks.

We say that a bounds mapping is *standard* for an environment A if it is derived from the real-time specification of A. Formally, given a view β of A, the standard bounds mapping B_A is defined as follows. For a message m with send point p, receive point q, and delay in the range $[L(m), H(m)]$, we have $B_A(q,p) = H(m)$ and $B_A(p,q) = -L(m)$. For any two adjacent points p and q at a processor whose local clock has drift bounds $\underline{\varrho} \leq \overline{\varrho}$, and assuming that p occurs before q, we have $B_A(q,p) = (virt\_del(q,p))/\underline{\varrho}$ and $B_A(p,q) = (virt\_del(p,q))/\overline{\varrho}$. Note that standard bounds mappings have the important property of being stated in terms of quantities available to the processors: $L(m), H(m), \overline{\varrho}, \underline{\varrho}$ and virtual delays.

Our next step is to define the concept of *relative offset*, that ties together actual and virtual delays. Relative offsets are the central quantities of interest in our analysis.

Definition 3.2 (Offset) *Given a pattern α, the* absolute offset *of a point p in α is $\delta_\alpha(p) = real\_time_\alpha(p) - loc\_time(p)$; for any two points p, q in α the* offset of p relative to q *is $\delta_\alpha(p,q) = \delta_\alpha(p) - \delta_\alpha(q)$.*

We state two immediate properties of the relative offsets.

Lemma 3.1 *For any two points p, q of a given pattern, $\delta(p,q) = -\delta(q,p)$.*

Lemma 3.2 *For any three points p, q, r of a given pattern, $\delta(p,q) = \delta(p,r) + \delta(r,q)$.*

We are now ready to define our main tool in analyzing the view of a given pattern.

Definition 3.3 (Synchronization Graph) *The* synchronization graph *$\Gamma_{\beta B} = (V, E, w)$ of a view β and a bounds mapping B is defined by the graph $\Gamma_\beta = (V, E)$ with a weight function w, where for each $(p,q) \in E$, $w(p,q) = B(p,q) - virt\_del(p,q)$.*

We stress that in the definition above, the bounds mapping is not necessarily standard. The following lemma states the basic property of the arc weights.

Lemma 3.3 *If α satisfies B then for every pair $(p,q) \in E$, $\delta_\alpha(p,q) \leq w(p,q)$, where w is the weight function in $\Gamma_{\beta B}$.*

Our strategy is to reduce the problem of computing bounds on the readings of remote clocks to distance computations in $\Gamma_{\beta B}$. Define the *synchronization distance $d(p,q)$* from p to q to be the length of the shortest directed path in $\Gamma_{\beta B}$ from p to q. We claim that synchronization distances are well defined in "true" synchronization graphs.

Lemma 3.4 *If α satisfies B and has view β, then the sum of weights of any directed cycle in $\Gamma_{\beta B}$ is non-negative.*

The following key lemma characterizes the set of patterns that have a given view and satisfy a given bounds mapping, in terms of synchronization distances.

Lemma 3.5 *Let α be a pattern with view β, and let B be a bounds mapping for β. Then α satisfies B if and only if for any two points p, q in $\Gamma_{\beta B}$ we have $\delta_\alpha(p,q) \leq d(p,q)$.*

We now arrive at the main result of this section. Loosely speaking, the following theorem says that the synchronization distance bounds of the previous lemma can indeed be met by the offsets of some pattern with identical synchronization graph.

Theorem 3.6 *Let $\Gamma_{\beta B} = (V, E, w)$ be a synchronization graph obtained from some pattern with view β satisfying bounds mapping B. Then for any given point $p_0 \in V$ there exist patterns α_0 and α_1 with view β, such that both α_0 and α_1 satisfy B, and such that the following hold.*[5]

1. *In α_0, for all $q \in V$, $\delta_{\alpha_0}(q, p_0) = d(q, p_0)$.*
2. *In α_1, for all $q \in V$, $\delta_{\alpha_1}(q, p_0) = -d(p_0, q)$.*

Proof: To prove the theorem, we first take care of infinite distances by constructing a related graph in which all distances are finite. Then we define the patterns α_0 and α_1, and show that they satisfy the required properties.

We start by noting that the distances in $\Gamma_{\beta B}$ are well defined, by Lemma 3.4. Now, let $N > 0$ be an arbitrary number; choose M that is sufficiently large so as to satisfy

$$M \;>\; N + \sum_{\substack{(p,q)\in E \\ 0<w(p,q)<\infty}} w(p,q) \;-\; \sum_{\substack{(p,q)\in E \\ -\infty<w(p,q)<0}} w(p,q)\ .$$

Using M, we augment $\Gamma_{\beta B}$ with extra arcs as follows. For each pair of points p, q such that $d(p,q) = \infty$, we

[5] The interpretation of $\delta_{\alpha_0}(q,p_0) = \infty$ is that for any given N, there exists α_0^N such that $\delta_{\alpha_0^N}(q,p_0) > N$; similarly where $\delta_{\alpha_1}(q,p_0) = -\infty$.

add an *artificial arc* (p,q) with weight M. Call the resulting augmented graph Γ^*, and denote its distance function by d^*. In the following claim we show that Γ^* has the intended finite distances for the given N.

Claim A. *For all $p,q \in V$, if $d(p,q) < \infty$, then $d^*(p,q) = d(p,q)$, and if $d(p,q) = \infty$, then $N < d^*(p,q) < \infty$.*

Proof of Claim A: We start (for future reference) with an inequality that follows directly from the choice of M. Let X and Y denote arbitrary subsets of the arcs of $\Gamma_{\beta B}$ with finite weights. Then

$$M + \sum_{(p,q)\in X} w(p,q) > \max\left\{ N \, , \sum_{(p,q)\in Y} w(p,q) \right\} \qquad (1)$$

Next, we argue that the augmented graph Γ^* has no negative weight cycles. Suppose, for contradiction, that there exists some negative weight cycle in Γ^*. Then one of arcs of the cycle, say (p,q), must be an artificial arc, and there must be a simple directed path Z in Γ^* from q to p with total weight w_Z such that $M + w_Z < 0$. Let $\underline{w}_Z$ be the sum of *negative* weight arcs of Z. Clearly, $\underline{w}_Z \le w_Z$. Also, by Eq. (1), we have that the sum of M and the weights of any subset of arcs of $\Gamma_{\beta B}$ is at least N. Since all artificial arcs have positive weight, we know that $\underline{w}_Z$ is the sum of weights of arcs from $\Gamma_{\beta B}$. Therefore we have that $M + w_Z \ge M + \underline{w}_Z > N > 0$, a contradiction.

To show that the finite distances in $\Gamma_{\beta B}$ remain invariant in Γ^*, we first note that since $\Gamma_{\beta B}$ is a subgraph of Γ^*, it must be the case for all $p,q \in V$ that $d^*(p,q) \le d(p,q)$. Suppose for contradiction that for some $p,q \in V$ we have $d^*(p,q) < d(p,q)$. Since, as we showed above, Γ^* has no negative-weight cycles, we may assume that there exists a simple path in Γ^* with length $d^*(p,q)$. Clearly, one of its arcs is artificial. However, by Eq. (1), this means that the total weight of that path is larger than the total weight of any finite-weight simple path in $\Gamma_{\beta B}$, a contradiction.

Finally, let $p,q \in V$ be such that $d(p,q) = \infty$. Clearly $d^*(p,q) < \infty$ by virtue of the artificial arc (p,q). To see that $d^*(p,q) > N$, consider any simple path from p to q. As before, this path contains at least one artificial arc, and therefore its total weight is at least M plus all negative weights of Γ. Using Eq. (1), we get that the total weight of the path is greater than N. ∎

We now define the patterns α_0 and α_1 explicitly. Since their view is given, the events and their local times are already fixed; we complete the construction by specifying the real time mappings of the patterns. Let $q \in V$. We set

$$\mathit{real\_time}_{\alpha_0}(q) \quad = \quad \mathit{loc\_time}(q) + d^*(q,p_0)$$

$$\mathit{real\_time}_{\alpha_1}(q) \quad = \quad \mathit{loc\_time}(q) - d^*(p_0,q)$$

By the construction, for all $q \in V$ we have

$$\begin{aligned} \delta_{\alpha_0}(q) &= \mathit{real\_time}_{\alpha_0}(q) - \mathit{loc\_time}(q) \\ &= (d^*(q,p_0) + \mathit{loc\_time}(q)) - \mathit{loc\_time}(q) \\ &= d^*(q,p_0) \ . \end{aligned} \qquad (2)$$

Since $d^*(p_0,p_0) = 0$, we conclude that $\delta_{\alpha_0}(q,p_0) = d^*(q,p_0)$. Similarly, we obtain that $\delta_{\alpha_1}(p_0,q) = -d^*(p_0,q)$. Therefore, by Claim A, α_0 and α_1 satisfy conditions (1) and (2) of the theorem. The following claim completes the proof.

Claim B. *The patterns α_0 and α_1 defined above satisfy the bounds mapping B.*

Proof of Claim B: By Lemma 3.5, it is sufficient to prove that for all $p,q \in V$, $\delta_{\alpha_0}(p,q) \le d(p,q)$. So let p and q be arbitrary points in the synchronization graph. In what follows, we consider Γ^*, in which all points are connected by finite-length paths with p_0. Since $d^*(p,q) \le d(p,q)$, it is sufficient to prove that $\delta_{\alpha_0}(p,q) \le d^*(p,q)$.

Let R be any shortest path from p to q. Consider the path obtained by following the arcs of R from p to q, and then the arcs of a shortest path from q to p_0. This path leads from p to p_0, and hence $d^*(p,q) + d^*(q,p_0) \ge d^*(p,p_0)$. It follows from Eq. (2) and the definition of relative offsets that

$$\begin{aligned} d^*(p,q) &\ge d^*(p,p_0) - d^*(q,p_0) \\ &= \delta_{\alpha_0}(p) - \delta_{\alpha_0}(q) \\ &= \delta_{\alpha_0}(p,q) \ . \end{aligned}$$

I.e., for all $p,q \in V$, $\delta_{\alpha_0}(p,q) \le d^*(p,q)$, and therefore, by Lemma 3.5, we conclude that α_0 satisfies the given bounds mapping B, as desired.

The proof for α_1 is analogous. We consider a shortest path R connecting two arbitrary points p and q. To show that its weight $d^*(p,q)$ is at least $\delta(p,q)$, we look at the path consisting of a shortest path P from p_0 to p, followed by R. As before, we have that $d^*(p_0,p) + d(p,q) \ge d^*(p_0,q)$, and hence we get

$$\begin{aligned} d^*(p,q) &\ge d^*(p_0,q) - d^*(p_0,p) \\ &= -\delta_{\alpha_1}(q) + \delta_{\alpha_1}(p) \\ &= \delta_{\alpha_1}(p,q) \ . \end{aligned}$$

Therefore, $\delta_{\alpha_1}(p,q) \le d^*(p,q)$ for all points $p,q \in V$, and applying Lemma 3.5 shows that α_1 satisfies B, as desired. ∎

This completes the proof of Theorem 3.6. ∎

Theorem 3.6 essentially says that if the only knowledge we have is a view and a bounds mapping, then *the tightest bounds one can hope to get on the offset between two points are exactly the synchronization distances between these points*, in the sense that the extreme values of that range are indeed attained by indistinguishable executions of the system. Note that by Lemma 3.5, breaking the distance bounds is possible only by patterns that do not satisfy B.

Theorem 3.6, when combined with Lemma 2.1, can be immediately used to derive lower bounds on the various synchronization problems. But also, it suggests a way to compute the bounds, which can be translated into optimal synchronization algorithms.

4 External Synchronization

In this section we apply the theory of Section 3 to the problem of external clock synchronization. After defining the problem, we proceed in Section 4.1 to give matching lower and upper bounds for the general case, where clocks are allowed to drift within known bounds. Then we show in Section 4.2 that the (appropriately defined) space overhead of any optimal general protocol cannot be bounded. On the other hand, we give in Section 4.3 a very simple and efficient algorithm for the case where all clocks are drift-free.

The external clock synchronization model is as follows. There exists a distinguished processor s, called the *source*, whose local clock shows exactly the real time, i.e., for any state x we have $loc\_time_s(x) = real\_time(x)$. The correctness requirement of a CSA at any processor is that the logical time (T, ε) at every state x satisfies $real\_time(x) \in [T-\varepsilon, T+\varepsilon]$. We call these CSAs *external*. We recall that a CSA is optimal for a given environment automaton A, if its error margin at any point of any pattern of A is the smallest of all CSAs at that point.

As a preliminary step, we state a general property of drift-free clocks (such as the source clock).

Lemma 4.1 *Suppose that the clock of processor v is drift-free. Then in all synchronization graphs defined using the standard bounds mapping, the distance between any two points that occur in v is 0.*

The meaning of Lemma 4.1 is that all points associated with a processor with perfect-rate local clock may be "collapsed" into a single "super-point" for synchronization distance purposes. Since the source clock s is drift-free by definition, we are justified in calling the aggregate of points associated with the source the *source point*, denoted p_s. For the source point, we have the following property.

Lemma 4.2 *Given a pattern of external synchronization system, for any point q we have $\delta(q) = \delta(q, p_s)$.*

4.1 The General Case

We are now ready to state bounds on the tightness of external synchronization. Let α be an arbitrary pattern of the environment, and consider any point q of α. Let Γ be the synchronization graph defined by the standard bounds mapping, and the view of α at q. The following theorem can be proved using Theorem 3.6 and Lemma 2.1.

Theorem 4.3 *For any external CSA, at any point q, $\varepsilon \geq \frac{1}{2}(d(p_s, q) + d(q, p_s))$.*

The lower bound of Theorem 4.3 can be matched by an on-line algorithm that simply maintains, at each point q, the complete synchronization graph of the view at q. This idea leads to the following theorem.

Theorem 4.4 *There exists an external CSA such that at every point q, $\varepsilon = \frac{1}{2}(d(p_s, q) + d(q, p_s))$.*

4.2 Space Lower Bound

The first problem in formalizing a space lower bound is that our model allows real numbers, because a real number can be used to encode unbounded amount of information. Our strategy to get around this difficulty is to use a generalization of the *comparison branching program* model [3], thus obtaining a bound on the number of "control bits" required to run the program. Very briefly, in our model a program is specified by a set of input variables, and a directed acyclic labeled graph with a single source (that corresponds to the initial state), a single sink (corresponding to the halting state), where all nodes have a bounded number of outgoing edges. The nodes represent control configurations of the program (excluding the input), labeled by some expressions of the input variables. An execution proceeds by evaluating these expressions, and selecting the next configuration according to their outcome (e.g., in [3], nodes specify pairs of input variables, and edges are labeled by "$<$", "$=$", or "$>$"). The *space requirement* of a branching program is the logarithm (to base 2) of the number of nodes in its graph: this is the least number of bits necessary to distinguish between different configurations of the program.

The lower bound argument relies on the bounded out-degree of nodes, and on a restriction on the way output is represented. Specifically, define a *special linear combination* for a set $X = \{x_1, \ldots, x_k\}$ to be an expression of the form $c_0 + \sum_{i=1}^{k} c_i x_i$, where $c_i \in [-1, 1]$ for all $1 \le i \le k$ and $c_0 \in \mathbf{R}$. A *special linear form* for X is the sequence of coefficients of some special linear combination for X. In our model, the output of an execution is specified by special linear forms associated with edges entering the sink node (i.e., last steps of executions). We argue that this limitation is reasonable, since optimal logical time can be expressed this way (synchronization distances are special linear combinations of local times and bounds).

To prove a lower bound, we focus on the way a processor computes its logical time when a new message arrives. Specifically, after a message is received, the processor executes some program (in our computational model). We stress that the program is a function of the current view (i.e., it may have all the knowledge of the view built into its structure). The input variables are the values that arrive in the message, and the output consists of a value for the current logical time. We have the following key lemma.

Lemma 4.5 *For any integer $M > 0$ there exist M patterns $\alpha_1, \ldots, \alpha_M$, and a receive point p at a processor v, common to all the patterns, such that the local view of all patterns at the point preceding p is identical, and such that the optimal logical time for each pattern after p must be produced by a distinct linear form.*

The following theorem is a consequence of Lemma 4.5, the model definition and Theorem 4.4.

Theorem 4.6 *The space overhead of any optimal protocol for external synchronization cannot be bounded by a function of the network size.*

4.3 The Case of Drift-Free Clocks

We now restrict attention to the case where all clocks in the system are drift-free. The property that all points in the synchronization graph associated with a drift-free clock can be "collapsed" into a single point (see Lemma 4.1) leads us to a simple algorithm for on-line distributed computation of synchronization distances. The overhead of the algorithm is just a few timestamps. Once having the distances computed, producing the logical time is a trivial matter.

Before we describe the algorithm, we introduce another piece of notation as follows. Given a synchronization graph with arc set E, define a set Q^{uv}, for each pair u, v of neighboring processors to be the set of all numbers $w(p, q)$, where p occurs at u, q occurs at v, and $(p, q) \in E$. We now describe the algorithm.

The *state* of at processor v consists, for each neighbor u, of estimates $\tilde{\mathtt{w}}_v(v, u)$ and $\tilde{\mathtt{w}}_v(u, v)$ of $\min Q^{uv}$ and $\min Q^{vu}$, respectively.[6] In addition, v maintains estimates $\tilde{\mathtt{d}}_v(v, s)$ and $\tilde{\mathtt{d}}_v(s, v)$ of its synchronization distances to and from the source point (i.e., estimates of $d(p_v, p_s)$ and $d(p_s, p_v)$, where p_v and p_s denote the "super-points" associated with v and s, respectively).

The *output* variables are the correction term Δ_v (to be added to the local time to produce the logical time), and the error margin ε_v.

All local variables take values from *signed* timestamp differences. All $\tilde{\mathtt{w}}$ variables are initially ∞. For non-source processors, $\tilde{\mathtt{d}}$ variables are initially ∞, Δ is initially undefined, and ε is initially ∞. At the source s, we have the $\tilde{\mathtt{d}}$ variables initialized to 0, $\Delta_s = 0$, and $\varepsilon_s = 0$.

We now describe the contents of the messages appended to on-going traffic by the CSA. Suppose that a message is sent at point q by a processor u to a processor v. This message carries five fields as follows. The first field is a time stamp that specifies $loc\_time(q)$. The other fields contain some of the state of u at point q. More specifically, u sends to v its current estimates of the minimal weights of arcs between u and v, (i.e., the current values of $\tilde{\mathtt{w}}_u(v, u)$ and $\tilde{\mathtt{w}}_u(u, v)$), and its current estimates of its synchronization distances relative to the source (i.e., $\tilde{\mathtt{d}}_u(s, u)$ and $\tilde{\mathtt{d}}_u(u, s)$).

We now specify the way messages received alter the state of the algorithm. Suppose that at point p processor v receives a message that was sent at point q, and contains

$$\left\langle loc\_time(q), \tilde{\mathtt{w}}_u(v, u), \tilde{\mathtt{w}}_u(u, v), \tilde{\mathtt{d}}_u(s, u), \tilde{\mathtt{d}}_u(u, s) \right\rangle$$

For brevity, let $\tilde{\mathtt{v}} = loc\_time(p) - loc\_time(q)$, i.e., $\tilde{\mathtt{v}}$ is the virtual delay of the arriving message. The updates described in Figure 3 are made instantaneously at point p.

$$\begin{array}{rcl}
\tilde{\mathtt{w}}_v(v, u) & \leftarrow & \min\{B(p, q) - \tilde{\mathtt{v}}\,,\, \tilde{\mathtt{w}}_u(v, u)\,,\, \tilde{\mathtt{w}}_v(v, u)\} \\
\tilde{\mathtt{w}}_v(u, v) & \leftarrow & \min\{B(q, p) + \tilde{\mathtt{v}}\,,\, \tilde{\mathtt{w}}_u(u, v)\,,\, \tilde{\mathtt{w}}_v(u, v)\} \\
\tilde{\mathtt{d}}_v(v, s) & \leftarrow & \min\left\{\tilde{\mathtt{w}}_v(v, u) + \tilde{\mathtt{d}}_u(u, s)\,,\, \tilde{\mathtt{d}}_v(v, s)\right\} \\
\tilde{\mathtt{d}}_v(s, v) & \leftarrow & \min\left\{\tilde{\mathtt{d}}_u(s, u) + \tilde{\mathtt{w}}_v(u, v)\,,\, \tilde{\mathtt{d}}_v(s, v)\right\} \\
\Delta_v & \leftarrow & \frac{1}{2}(\tilde{\mathtt{d}}_v(v, s) - \tilde{\mathtt{d}}_v(s, v)) \\
\varepsilon_v & \leftarrow & \frac{1}{2}(\tilde{\mathtt{d}}_v(v, s) + \tilde{\mathtt{d}}_v(s, v))
\end{array}$$

Figure 3: *Actions taken when the CSA at processor v receives a message from processor u. The value $\tilde{\mathtt{v}}$ denotes the virtual delay of the message.*

The following lemma, proved by induction on the

[6] Variables are subscripted by the processor in which they are located.

length of the view, essentially shows that the algorithm is correct.

Lemma 4.7 *Let p be a point that occurs at processor v. Let $\Gamma = (V, E, w)$ be the synchronization graph of the local view at p, and the standard bounds mapping. Then the following invariant holds at p: $\tilde{\mathtt{w}}(v, u) = \min Q^{vu}$, $\tilde{\mathtt{w}}(u, v) = \min Q^{uv}$, $d_\Gamma(p_s, p) = \tilde{\mathsf{d}}(s, v)$, and $d_\Gamma(p, p_s) = \tilde{\mathsf{d}}(v, s)$.*

Using Lemma 4.7, Theorem 4.4 and Lemma 2.1, we obtain the following theorem.

Theorem 4.8 *The algorithm above is an optimal external CSA.*

5 Internal Synchronization

In this section we use the Theorem 3.6 to obtain a lower bound on the tightness of internal clock synchronization, in terms of the actual execution. This lower bound is a generalization of the known bounds for the drift-free case [6, 2].

Intuitively, the goal of internal synchronization is to keep logical clocks as close as possible. To avoid liveness issues when proving a lower bound, and following [4, 6], we use a "one shot" definition of the problem. Specifically, we assume that for each node v, there is a special action called fire_v. Each processor must output this action exactly once, and the goal is to minimize the length of the real-time interval in which all the *fire* actions occur.[7] Formally, we define the *tightness* of a pattern α of an internal clock synchronization system by $\mathit{tight}(\alpha) = \max_v \{\mathit{real\_time}(\mathit{fire}_v)\} - \min_v \{\mathit{real\_time}(\mathit{fire}_v)\}$. The *tightness of a view* β, denoted $\mathit{tight}(\beta)$, is the supremum of $\mathit{tight}(\alpha)$ over all patterns α of the system with view β.

We use the following graph-theoretic concept.

Definition 5.1 *Let $G = (V, E, w)$ be a weighted directed graph. The* maximum cycle mean *of G, denoted $\mathrm{mcm}(G)$, is the maximum average weight of an edge in a directed cycle of G. Symbolically,*

$$\mathrm{mcm}(G) = \max_\theta \left\{ \frac{1}{|\theta|} \sum_{i=1}^{|\theta|} w(v_{i-1}, v_i) \right\} ,$$

where $\theta = \langle v_0, \ldots, v_{|\theta|-1} \rangle$ ranges over all directed cycles of G, and $v_{|\theta|} = v_0$.

[7] The intended meaning is that a processor "fires" when its logical clock shows a certain value; this value can be arbitrary, provided that all logical clocks show it sometime.

Our first step is to extend the notion of synchronization graph to contain the "fire" points, and then condense its information as follows.

Definition 5.2 *Given a synchronization graph $\Gamma = (V, E, w)$ of an internal clock synchronization system, the* internal synchronization graph *is a directed, weighted graph $\overline{\Gamma} = (\overline{V}, \overline{E}, \overline{w})$, where the set of points $\overline{V}$ consists of all the fire points in V; there is an arc in $\overline{E}$ between every pair of points of $\overline{V}$; and $\overline{w}(\mathit{fire}_v, \mathit{fire}_u) = d_\Gamma(\mathit{fire}_v, \mathit{fire}_u)$.*

We can now state the lower bound.

Theorem 5.1 *Let β be the view of a pattern of an internal clock synchronization system, and let $\overline{\Gamma}$ be the internal synchronization graph defined by β and the standard bounds mapping. Then $\mathit{tight}(\beta) \geq \mathrm{mcm}(\overline{\Gamma})$.*

Proof Sketch: In this sketch we prove the case of finite tightness; the extension for infinite tightness is trivial. Consider any directed cycle $\theta = p_0, p_1, \ldots, p_{|\theta|} = p_0$ in the internal synchronization graph $\overline{\Gamma} = (\overline{V}, \overline{E}, \overline{w})$. By Theorem 3.6, there exist indistinguishable patterns α_i such that $\delta_{\alpha_i}(p_{i-1}, p_i) = \overline{w}(p_{i-1}, p_i)$, for each $1 \leq i \leq k$. It follows from the definition of tightness that

$$\begin{aligned} \mathit{tight}(\beta) \;\geq\; \mathit{tight}(\alpha_i) \;&\geq \\ \mathit{real\_time}_{\alpha_i}(p_{i-1}) \;&-\; \mathit{real\_time}_{\alpha_i}(p_i) \\ = \delta_{\alpha_i}(p_{i-1}, p_i) \;&+\; \mathit{virt\_del}(p_{i-1}, p_i) \\ = \overline{w}(p_{i-1}, p_i) \;&+\; \mathit{virt\_del}(p_{i-1}, p_i) \; . \end{aligned} \qquad (3)$$

Summing Eq. 3 over all i, and observing that the sum of virtual delays over a cyclic sequence vanishes, we get

$$\begin{aligned} |\theta| \cdot \mathit{tight}(\beta) &\geq \sum_{i=1}^{|\theta|} \overline{w}(p_{i-1}, p_i) + \sum_{i=1}^{|\theta|} \mathit{virt\_del}(p_{i-1}, p_i) \\ &= \sum_{i=1}^{|\theta|} \overline{w}(p_{i-1}, p_i) \end{aligned}$$

Since θ was an arbitrary cycle in $\overline{\Gamma}$, we conclude that $\mathit{tight}(\beta) \geq \mathrm{mcm}(\overline{\Gamma})$. ∎

We remark that algorithms for external synchronization can be used for internal synchronization, by letting an arbitrary processor play the role of the source; the optimality of our algorithms for the external variant implies that they guarantee tightness which is at most twice the optimum for internal synchronization.

6 Extensions

Structured Environments. The basic theory studies the case where send modules are completely unstructured (technically, the "send" action is always enabled), and where the network module may lose messages arbitrarily. Somewhat surprisingly, it turns out that one may gain timing knowledge also from the *absence* of a message receive event, in the case of reliable communication. Extending the model to this case is straightforward: the only relevant fact is the knowledge of the local time when a message is sent.

Lemma 6.1 *Suppose that a message m is known to be sent from u at point q and is guaranteed to be delivered within $H(m)$ time at v. Let p be any point at v where m has not yet been delivered. Then $\delta(p,q) \leq H(m) - virt\_del(p,q)$.*

Inaccurate Source Clocks. In the external synchronization model we assumed that a source clock shows the real time precisely. However, the synchronization graph model can be used to deal with an inaccurate source clock also. The idea is as follows. Assume the existence of some abstract clock that shows the time precisely (it may be illustrative to think of it as "Nature's clock"). This clock is represented as a source point p_s in the synchronization graph, and all other points in the graph are connected to p_s by arcs. By Lemma 4.2, the weights of theses arcs are chosen so that the offset of a point q is known to be certainly in the range $[-w(p_s,q), w(q,p_s)]$. It is easy to verify that all the algorithms presented in this paper can be extended to deal with this concept without increasing their complexity.

Fault Detection. Throughout the discussion on synchronization graphs we relied heavily on their integrity, i.e., the fact that $\delta(p,q) \in [-d(q,p), d(p,q)]$. This assumption may not hold if some component of the system malfunctions, which may cause it to either generate inconsistent timestamps, or to break the bounds mapping, or simply to have an inaccurate image of the synchronization graph. Fortunately, Theorem 3.6 guarantees a strong fault-detection property: if a fault can be detected, it will. More precisely, a faulty pattern is *detectable* if there is no pattern of the environment with the same view. Using Theorem 3.6, we derive the following result.

Lemma 6.2 *Let B be a bounds mapping, and let β be a view of a pattern α. Then α has a detectable fault if and only if $\Gamma_{\beta B}$ contains a negative cycle.*

Acknowledgment

We thank Nancy Lynch for many useful discussions.

References

[1] J. E. Abate, E. W. Butterline, R. A. Carley, P. Greendyk, A. M. Montenegro, C. D. Near, S. H. Richman, and G. P. Zampelli. AT&T new approach to the synchronization of telecommunication networks. *IEEE Communication Magazine*, pages 35–45, Apr. 1989.

[2] H. Attiya, A. Herzberg, and S. Rajsbaum. Optimal clock synchronization under different delay assumptions. In *Proceedings of the 12th Annual ACM Symposium on Principles of Distributed Computing*, pages 109–120, 1993.

[3] A. Borodin, M. Fischer, D. Kirkpatrick, N. Lynch, and M. Tompa. A time-space tradeoff for sorting on non-oblivious machines. *J. Comp. and Syst. Sci.*, 22:351–364, 1981.

[4] D. Dolev, J. Y. Halpern, and R. Strong. On the possiblity and impossibility of achieving clock synchronization. *J. Comp. and Syst. Sci.*, 32(2):230–250, 1986.

[5] J. Halpern and I. Suzuki. Clock synchronization and the power of broadcasting. In *Proc. of Allerton Conference*, pages 588–597, 1990.

[6] J. Y. Halpern, N. Megiddo, and A. A. Munshi. Optimal precision in the presence of uncertainty. *Journal of Complexity*, 1:170–196, 1985.

[7] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Comm. ACM*, 21(7):558–565, July 1978.

[8] J. Lundelius and N. Lynch. An upper and lower bound for clock synchronization. *Information and Computation*, 62(2-3):190–204, 1984.

[9] N. Lynch. Simulation techniques for proving properties of real-time systems. In *Rex Workshop '93*, Lecture Notes in Computer Science, Mook, the Netherlands, 1994. Springer-Verlag. To appear.

[10] K. Marzullo and S. Owicki. Maintaining the time in a distributed system. In *Proceedings of the 2nd Annual ACM Symposium on Principles of Distributed Computing*, pages 44–54, 1983.

[11] D. L. Mills. Internet time synchronization: the Network Time Protocol. *IEEE Trans. Comm.*, 39(10):1482–1493, Oct. 1991.

[12] Y. Ofek. Generating a fault tolerant global clock using high-speed control signals for the MetaNet architecture. *IEEE Trans. Comm.*, Dec. 1993.

[13] F. B. Schneider. Understanding protocols for byzantine clock synchronization. Research Report 87-859, Department of Computer Science, Cornell University, Aug. 1987.

[14] B. Simmons, J. L. Welch, and N. Lynch. An overview of clock synchronization. Research Report RC 6505 (63306), IBM, 1988.