Noname manuscript No.
(will be inserted by the editor)

Rui Fan · Nancy Lynch

Gradient Clock Synchronization

the date of receipt and acceptance should be inserted later

Abstract We introduce the distributed *gradient clock synchronization* problem. As in traditional distributed clock synchronization, we consider a network of nodes equipped with hardware clocks with bounded drift. Nodes compute logical clock values based on their hardware clocks and message exchanges, and the goal is to synchronize the nodes' logical clocks as closely as possible, while satisfying certain validity conditions. The new feature of gradient clock synchronization (*GCS* for short) is to require that the skew between any two nodes' logical clocks be bounded by a nondecreasing function of the *uncertainty in message delay* (call this the *distance*) between the two nodes, and other network parameters. That is, we require nearby nodes to be closely synchronized, and allow faraway nodes to be more loosely synchronized. We contrast GCS with traditional clock synchronization, and discuss several practical motivations for GCS, mostly arising in sensor and ad-hoc networks. Our main result is that the worst case clock skew between two nodes at distance d or less from each other is $\Omega(d + \frac{\log D}{\log \log D})$, where D is the diameter[1] of the network. This means that clock synchronization is not a *local* property, in the sense that the clock skew between two nodes depends not only on the distance between the nodes, but also on the size of the network. Our lower bound implies, for example, that the TDMA protocol with a fixed slot granularity will fail as the network grows, even if the maximum degree of each node stays constant.

Keywords clock synchronization, lower bounds, indistinguishability, ad-hoc networks

1 Introduction

Consider the classical distributed clock synchronization problem. A set of nodes communicate over a reliable network with bounded message delay. Each node is equipped with a *hardware clock* with *bounded drift*, that is, a timer running at roughly, but possibly not exactly, the rate of real time. Each node continuously computes *logical clock* values based on its hardware clock, and on messages exchanged with other nodes. The goal is to synchronize the nodes' logical clocks as closely as possible. To rule out trivial algorithms, the logical clocks must satisfy some validity conditions, for example, that they remain close to real time. This problem has been the subject of extensive research. Previous work in the area has focused on minimizing the clock skew between nodes and minimizing the amount of communication used by the synchronization algorithm [12,2], on tolerating various types of failures of the nodes and the network [5,12], and on proving lower bound results about clock skew and communication costs [7,3,10, 9]. In this paper, we introduce a new property for clock synchronization algorithms (*CSA* for short), the *gradient property*. Define the *distance* between two nodes to be the uncertainty in message delay between the nodes. Informally, the gradient property requires that the skew between two nodes forms a gradient

R. Fan
MIT CSAIL, 32 Vassar St., Cambridge, MA 02139 E-mail: rfan@theory.csail.mit.edu

N. Lynch
MIT CSAIL, 32 Vassar St., Cambridge, MA 02139 E-mail: lynch@theory.csail.mit.edu

[1] That is, the maximum message delay uncertainty between any pair of nodes.

with respect to the distance between the nodes. That is, nearby nodes should be closely synchronized, while faraway nodes may be more loosely synchronized.

We first contrast gradient clock synchronization with traditional clock synchronization. Let D be the diameter of the network. Then there exists a well-known lower bound result [7] saying that, for any CSA, the worst case clock skew between some pair of nodes in the network is $\Omega(D)$[2]. Most CSAs (e.g., [12]) achieve a worst case skew of $O(D)$. However, these CSAs allow $O(D)$ skew between *any* two nodes. In particular, to our knowledge, in all existing CSAs, there exist executions in which a pair of nodes at $O(1)$ distance from each other have $O(D)$ skew. Thus, current CSAs do not satisfy the gradient property, because nearby nodes are not always well synchronized.

We now discuss some motivation for studying the gradient property. In many highly decentralized networks, such as sensor and ad-hoc networks, applications are *local* in nature. That is, only nearby nodes in the network need to cooperate to perform some task, and nodes that are far away interact much less frequently. Hence, only nearby nodes need to have highly synchronized clocks. As nodes get farther apart, they can tolerate greater clock skew. Thus, for these applications, the maximum acceptable clock skew between two nodes forms a gradient in their distance.

As an example, consider first the *data fusion* [11] problem in a sensor network. A group of distributed sensors collect data, then try to aggregate their data at one node to perform some signal processing on it. In order to conserve energy, the sensors form a communication tree. Starting from the leaves, each sensor sends its data to its parent sensor. When a parent sensor has received data from all its children, it "fuses" the data, that is, constructs a summary representation of the data, and sends the summary to its own parent. Since sensors typically measure real-world phenomena, times are associated with the sensor measurements. When fusing data, the children of a parent node must synchronize their clocks, so that the times of their readings are consistent and a fused reading will make sense. Hence, nearby nodes, which may be children of the same parent, need to have well synchronized clocks. But faraway nodes, which are not children of the same parent, can tolerate more poorly synchronized clocks.

Next, consider the *target tracking* problem in a sensor network. Suppose two sensor nodes want to measure the speed of an object. Each node records the time when the object crosses within its vicinity. Then the nodes exchange their time readings, and compute t, the difference in their readings. The amount of error in t is related to the clock skew between the nodes. The object's velocity is computed as $v = \frac{d}{t}$, where d is the known Euclidean distance between the nodes. Suppose the nodes do not need to compute v exactly, but only to an accuracy of 1%. Since $v = \frac{d}{t}$, then the larger the Euclidean distance is between the nodes, the more error is acceptable in t, while still computing v to 1% accuracy. Thus, the acceptable clock skew of the nodes forms a gradient[3].

What kind of gradient can be achieved by a clock synchronization algorithm? When the network consists of two nodes at distance d from each other, the tightest possible worst-case synchronization between the nodes is $O(d)$. If there are more nodes, arranged in an arbitrary topology, is there a synchronization algorithm that ensures that the clock skew between all pairs of nodes is linear in their distance at all times? We show that no such algorithm exists. Our main result, stated in Theorem 1 of Section 5, is that if hardware clocks can drift, then given any clock synchronization algorithm and any $D \geq 1$, there exists a network of diameter D, such that for any $d \in [1, D]$, there exists some execution of the CSA in which two nodes at most distance d apart in the network have $\Omega(d + \frac{\log D}{\log \log D})$ clock skew. An implication of this result is that an application, such as TDMA [6], that requires a fixed maximum skew between nearby nodes, cannot scale beyond networks of a certain diameter. We conjecture that the lower bound is nearly tight for small d, and that there exist CSAs which ensure that $O(1)$ distance nodes always have $O(\log D)$ clock skew.

The rest of this paper is organized as follows. Section 2 describes previous work on clock synchronization and its relation to our work. Section 3 defines our model for clock synchronization, and Section 4 formally defines the gradient clock synchronization problem. We state our main lower bound and give an overview of its proof in Section 5. We prove two lemmas in Sections 6 and 7, then prove the GCS lower bound in Section 8. Finally, we conclude in Section 9 with some remarks and open problems.

---

[2] The result in [7] assumes that hardware clocks do not drift, and nodes start with arbitrary logical clock values. In this paper, we assume that hardware clocks can drift, and nodes start with the same logical clock value. However, it is possible to adapt the techniques of [7] to show an $\Omega(D)$ lower bound on clock skew in our model.

[3] Note that here we are assuming the Euclidean distance between two nodes corresponds to the uncertainty in their message delay. This is the case if, for example, there are multiple network hops between the nodes, with the number of hops proportional to the Euclidean distance between the nodes.

2 Relation to Previous Work

To our knowledge, this work is the first theoretical study of gradient clock synchronization and lower bounds for the problem. Many other lower bounds have been proven for clock synchronization. The two most important parameters in these lower bounds are the uncertainty in message delay, and the rate of clock drift[4].

Lundelius and Lynch [7] proved that in a complete network of n nodes where the distance between each pair of nodes is d, nodes cannot synchronize their clocks to closer than $d(1 - \frac{1}{n})$. Halpern *et al* [3] and Biaz and Welch [1] extended the previous result to more general graphs, and gave algorithms which match or nearly match the lower bounds. These papers all assume nodes have perfect (non-drifting) clocks. Patt-Shamir and Rajsbaum [10] proved lower bounds on clock skew in terms of a *synchronization graph*, which is a graph over the events of an execution, with edge weights related to the times of the events. While their results are somewhat similar in spirit to our distance-based lower bound, the problem they considered is one in which several nodes try to output a signal as closely together in *real time* as possible. The clock skew in this context is the real time difference between when different nodes output the signal. In contrast, clock skew in our context is defined as the difference in the nodes' logical clock values at any real time. Thus, our work is not directly comparable to [10]. Ostrovsky and Patt-Shamir [9] also proved lower bounds using synchronization graphs. However, their work deals with *external synchronization*, in which nodes try to synchronize to a common source of time. Our works deals with gradient synchronization. Accurate external synchronization does not in general imply accurate gradient synchronization. This is because even an algorithm which ensures all nodes are synchronized to within $O(D)$ of real time (this is the tightest synchronization achievable) may not provide good gradient clock synchronization, because two $O(1)$-distance nodes may have $O(D)$ clock skew.

Srikanth and Toueg [12] gave an optimal clock synchronization algorithm, where optimal means that the skew of a node's logical clock from real time is as small as possible, given the hardware clock drift of the node. Their algorithm ensures that any pair of nodes have $O(D)$ clock skew, where D is the diameter of the network. However, it does not guarantee a gradient in the clock skew, because even nodes that are $O(1)$ distance apart can have $O(D)$ skew. We now explain how this can happen. Because Srikanth and Toueg's algorithm is complicated, we consider a much simpler algorithm, which nevertheless illustrates the main reason why [12] violates the gradient property. Intuitively, the reason is that a node's logical clock value is allowed to suddenly "jump" to a much higher value, without coordinating with neighboring nodes. The simple algorithm works as follows. Nodes periodically broadcast their logical clock values, and any node receiving a value sets its logical clock value to be the larger of its own clock value and the received value. Now, consider an execution consisting of three nodes x, y and z, arranged in a line topology. Let the distance between x and y be X, for some constant $X \gg 1$, let the distance between y and z be 1, and let the distance between x and z be $X + 1$. By making the message delay X between x and y and 1 between y and z, and by making x's hardware clock rate higher than y's, which is in turn higher than z's, we can create an execution in which x's clock is X higher than y's clock, which in turn is 1 higher than z's clock. Now, we extend this execution by changing all future message delays between x and y to be 0, but keeping the delay between y and z at 1. Then, when y receives a message from x, y will realize its clock is X lower than x's clock, and so y will increase its clock by X. However, because the message delay between y and z is still 1, z receives x's message one second later than y does. Thus, there is a one second interval during which y has increased its clock by X, but z has not increased its clock. During this one second interval, y's clock is $X + 1$ higher than z's clock, even though y and z have distance 1. Thus, this execution violates the gradient property.

Elson *et al.* [2] studied time synchronization in a sensor network. Their algorithm, RBS, relies on physical properties of the radio broadcast medium to reduce message delay uncertainty to almost 0. RBS works as follows: A beacon node initially broadcasts a signal to its neighboring nodes. Each neighbor records its logical clock value when it receives the signal. Then, in a second phase, each neighbor broadcasts its recorded clock value, and computes its clock skew with respect to another node as the difference between that node's recorded value and its own recorded value. Since the initial signal is broadcast by radio, it takes about the same amount of time to reach all the neighboring nodes, and so the uncertainty in message delay of the signal is close to 0. Our lower bound result still applies in the RBS setting, but it gives a rather small bound because the diameter of the network is small. However, in principle, as the network expands, our lower bound becomes more relevant.

[4] The clock drift rate is defined as a constant $0 \leq \rho < 1$, such that at all times, the rate of increase of each node's hardware clock lies within the interval $[1-\rho, 1+\rho]$. Our lower bound only holds when clock drift is positive. Thus, for the remainder of this paper, we will assume that $\rho > 0$.

Recently, Meier and Thiele [8] extended our work and showed a lower bound on gradient clock synchronization in a different communication model. While our communication model has nonzero uncertainty for message delays and allows nodes to communicate arbitrarily, Meier and Thiele's model has *zero* message delay uncertainty, but, roughly speaking, only allows nodes to communicate once every R time, where $R > 0$ is some parameter[5]. This model is intended to capture certain characteristics of radio networks. Using techniques based on ours, [8] shows that for any CSA, there exist neighboring nodes that have $\Omega(R\frac{\log n}{\log\log n})$ skew, where n is the number of nodes in the network. The number of nodes n in [8] is analogous to the diameter D in this paper, so the lower bound in [8]'s model and our model are similar.

3 Model

In this section, we describe our system model, and formally define *clock synchronization algorithms* (*CSA*s). CSAs are algorithms run by a *network* of nodes equipped with *hardware clocks* with bounded drift. We first define these two terms.

We model nodes as timed automata [4]. Given a node i, a *hardware clock* for i is a read-only variable which can only be read by i. We define the value of i's hardware clock in terms of its rate of change. Specifically, we denote i's *hardware clock rate* at real time t of an execution α by $h_i^\alpha(t)$. We define i's *hardware clock value* at time t in α to be $H_i^\alpha(t) = \int_0^t h_i^\alpha(r)\mathrm{d}r$. We assume that the hardware clock of any node has *bounded drift*. That is, we assume that there exists a constant ρ called the *drift rate*, with $0 \leq \rho < 1$, such that for any execution α, and for any node i, the following holds.

Assumption 1 $\forall t : 1 - \rho \leq h_i^\alpha(t) \leq 1 + \rho$.

For our lower bound result to hold, we need the drift rate to be positive. Thus, for the remainder of this paper, we assume that $0 < \rho < 1$.

A *network* consists of a set of nodes, some pairs of which are connected by reliable message channels with bounded delay. Formally, a network $\mathcal{N} = (N, E, \delta)$, where N is a set of nodes, $E \subseteq N \times N$ is a set of ordered pairs representing nodes which are connected by a communication channel, and $\delta : E \to \mathbb{R}^{\geq 0}$ is a mapping representing the *uncertainty* in message delay on each channel in E ($\mathbb{R}^{\geq 0}$ is the set of nonnegative reals). Specifically, if $(i, j) \in E$, then every message sent from i to j takes at least 0 time, and at most $\delta(i, j)$ time, to arrive at j. The time taken by each message from i to j may vary during an execution. For convenience, we abbreviate $\delta(i, j)$ by $d_{i,j}$, and call this the *distance* from i to j. We denote the *diameter* of $\mathcal{N}$ by $D(\mathcal{N}) = \max_{(i,j)\in E} d_{i,j}$. We do not require that distances be symmetric. That is, we do not require $d_{i,j} = d_{j,i}$. Since our results are stated in terms of the diameter of the network, we need to define a unit distance. Thus, for any network, we assume that $\min_{(i,j)\in E} d_{i,j} = 1$.

A *clock synchronization algorithm* (*CSA*) is any algorithm run by a network of nodes equipped with hardware clocks with bounded drift, in which each node continuously computes a real value called its *logical clock value*. More precisely, an algorithm $\mathcal{A}$ is a CSA if for all networks $\mathcal{N}$, for all executions α in which the nodes of $\mathcal{N}$ run algorithm $\mathcal{A}$, and at any time t during α, each node i of $\mathcal{N}$ uses its hardware clock values up to time t, and the set of messages it received from other nodes of $\mathcal{N}$ up to time t, to compute its logical clock value at time t of α, denoted by $L_i^\alpha(t)$. When the nodes of $\mathcal{N}$ run algorithm $\mathcal{A}$, we call this an *execution of $\mathcal{A}$ in $\mathcal{N}$*.

We now describe several assumptions we make about any CSA $\mathcal{A}$. Let $\mathcal{N}$ be an arbitrary network. First, we assume that the logical clock values of all nodes start at 0. That is, we assume that for all nodes i of $\mathcal{N}$ and for all executions α of $\mathcal{A}$ in $\mathcal{N}$, we have $L_i^\alpha(0) = 0$. Next, we assume that all nodes i know the an upper bound for ρ, which is strictly less than 1 (recall that $\rho < 1$). Lastly, we assume that all the nodes know $\mathcal{N}$. That is, we assume that in any execution of $\mathcal{A}$ in $\mathcal{N}$, every node of $\mathcal{N}$ knows the topology and message delay uncertainties of the network it is executing in. The reason for making these assumptions is to simplify our presentation, and also to satisfy certain properties we will define in Section 4. Notice that making these assumptions about $\mathcal{A}$ does not weaken our lower bound, since a CSA that knows an upper bound for ρ, knows $\mathcal{N}$, and has logical clock values starting at 0, can easily simulate another CSA without such knowledge, and in which logical clock values do not start at 0.

Lastly, we define some notation, and describe an *indistinguishability* principle that will be used in proving our lower bounds. Given a finite execution α of an algorithm, we say the *duration* of α, written as $\ell(\alpha)$, is the real time duration of α. Let $\alpha(t)$ denote the state of α at real time t, before any events occur at t. If π is an event in α, we let $T_\alpha(\pi)$ denote the real time when π occurs.

[5] [8] uses the variable d instead of R. We use R since d has a different meaning in our paper.

Because a node computes its logical clock values based only on readings from its hardware clock and on messages it has received, it cannot distinguish between two executions in which the same actions occur at the same readings of its hardware clock, in the same order. More precisely, let α and β be two executions, and suppose that the same set of actions occur in α and β, in the same order. For each action π at node i in α, let π' denote the corresponding action in β. Then, if for every such π we have $H_i^\alpha(T_\alpha(\pi)) = H_i^\beta(T_\beta(\pi'))$, i behaves the same in α and β.

4 Gradient Clock Synchronization

In this section, we formally define the gradient clock synchronization problem. We first define two properties which a clock synchronization algorithm may satisfy. For the remainder of this section, fix $\mathcal{A}$ to be an arbitrary CSA.

Property 1 (Validity) *Let $\mathcal{N} = (N, E, \delta)$ be any network. Then for all executions α of $\mathcal{A}$ in $\mathcal{N}$, we have $\forall i \in N \forall t \forall r > 0 : \frac{r}{2} \leq L_i^\alpha(t+r) - L_i^\alpha(t)$.*

This property says that in any execution of $\mathcal{A}$ in any network, the rate of increase of each node's logical clock is at least $\frac{1}{2}$, *at all times*. Note that the value $\frac{1}{2}$ was chosen for simplicity, and can be replaced by an arbitrary positive constant. Many existing CSAs satisfy this validity property (e.g., [12, 2]). A primary motivation for defining this property is that many applications that use synchronized clocks require the clocks to never increase too slowly. For example, if a sensor node is using its logical clock to timestamp readings of physical events, then the clock should increase at roughly the rate of real time. We note, however, that not all CSAs satisfy Property 1, e.g., [13] and [5] do not. Our lower bound does not apply to such algorithms. Also note that because we assume that $\mathcal{A}$ knows an upper bound for ρ, it is possible for $\mathcal{A}$ to satisfy Property 1 by, for example, always increasing its logical clock at a rate of at least $\frac{1}{2(1-\hat{\rho})}$ times its hardware clock, where $\hat{\rho}$ is the upper bound on ρ.

We now define the gradient property. Let $\mathfrak{N}$ represent the class of all networks, and let $f : \mathbb{R}^{\geq 0} \times \mathfrak{N} \to \mathbb{R}^{\geq 0}$ be any function. We say $\mathcal{A}$ satisfies the *f-gradient* property if for any network $\mathcal{N}$ and any execution of $\mathcal{A}$ in $\mathcal{N}$, the difference in logical clock values between any two nodes i and j is at most $f(d, \mathcal{N})$ at all times, for any d *greater than or equal to* $d_{i,j}$, the distance between i and j in $\mathcal{N}$. That is, given any two nodes which are at most distance d apart, their clock skew can always be bounded by a function f of d and some properties of the network the nodes reside in. Formally, we have

Property 2 (f-Gradient) *Let $\mathcal{N} = (N, E, \delta)$ be any network. Then for all executions α of $\mathcal{A}$ in $\mathcal{N}$, we have*

$$\forall i, j \in N \ \forall t \ \forall d \geq d_{i,j} : |L_i^\alpha(t) - L_j^\alpha(t)| \leq f(d, \mathcal{N})$$

Finally, we define an f-gradient clock synchronization (f-GCS) algorithm.

Definition 1 Let $f : \mathbb{R}^{\geq 0} \times \mathfrak{N} \to \mathbb{R}^{\geq 0}$ be an arbitrary function. We say a clock synchronization algorithm $\mathcal{A}$ is an *f-gradient clock synchronization algorithm* if $\mathcal{A}$ satisfies the Validity and f-Gradient properties.

5 Overview of Lower Bound on GCS

In this section, we state our main theorem, and give an overview of its proof.

Theorem 1 *Let $\mathcal{A}$ be an arbitrary f-GCS. Then we have $f(d, \mathcal{N}) = \Omega(d + \frac{\log D(\mathcal{N})}{\log \log D(\mathcal{N})})$. In particular, for any D, there exists a network $\mathcal{N}$ of diameter D, such that for any $d \in [1, D]$, there exists an execution α of $\mathcal{A}$ in $\mathcal{N}$, such that some two nodes at most distance d apart in $\mathcal{N}$ have $\Omega(d + \frac{\log D}{\log \log D})$ clock skew at some time in α.*

To prove Theorem 1, it suffices to show that for any f-GCS algorithm $\mathcal{A}$ and any $D \geq 2$, there exists a particular network of diameter $D-1$ such that Theorem 1 holds. Such a network is the line network. For the remainder of this paper, fix $\mathcal{A}$ to be an arbitrary f-GCS, let $D \geq 2$ be an arbitrary integer, and let $\mathcal{N}$ be a network with nodes $1, \ldots, D$, such that $d_{i,j} = |i - j|$, for $1 \leq i, j \leq D$. Note that the diameter of $\mathcal{N}$ is $D-1$. All executions we describe in the remainder of this paper will be executions of $\mathcal{A}$ in $\mathcal{N}$. Thus, for the remainder of this paper, we will usually refrain from explicitly mentioning $\mathcal{A}$ or $\mathcal{N}$. All executions, distances, etc., will implicitly be with respect to $\mathcal{A}$ and $\mathcal{N}$. Furthermore, instead of writing

$f(\cdot, \mathcal{N})$, we will typically simply write $f(\cdot)$. These definitions allow us to state Theorem 1', which implies Theorem 1, but which is somewhat simpler.

Theorem 1' *For any $d \in [1, D-1]$, there exists an execution α of $\mathcal{A}$, such that some two nodes i, j, with $d_{i,j} \leq d$, have $\Omega(d + \frac{\log D}{\log \log D})$ clock skew at some time in α.*

To prove Theorem 1', we show that the following hold:

1. For any $d \in [1, D-1]$, there exist two nodes that are distance d apart, such that the two nodes have $\Omega(d)$ clock skew in some execution of $\mathcal{A}$. This implies that $f(d) = \Omega(d)$ for algorithm $\mathcal{A}$.
2. There exist two nodes i and j that are distance 1 apart, such that the two nodes have $\Omega(\frac{\log D}{\log \log D})$ clock skew in some execution of $\mathcal{A}$. This implies that $f(1) = \Omega(\frac{\log D}{\log \log D})$, for algorithm $\mathcal{A}$. Also, since Property 2 requires that the skew between i and j be at most $f(d)$, for all $d \geq d_{i,j} = 1$, then $f(d) = \Omega(\frac{\log D}{\log \log D})$, for all $d \in [1, D-1]$, for algorithm $\mathcal{A}$.

By adding the two lower bounds, we obtain that $f(d) = \Omega(d + \frac{\log D}{\log \log D})$ for $\mathcal{A}$. A formal proof of Theorem 1' is given at the end of Section 8.

The executions demonstrating the lower bounds are created by adversarially controlling the hardware clock rates and message delays of the nodes, and by using indistinguishability type arguments.

We first show that $f(d) = \Omega(d)$, for any $d \in [1, D-1]$. This result is folklore, and it follows from the type of indistinguishability argument used, for example, by Lundelius and Lynch in [7]. We only sketch the proof. Let i and j be two nodes which are distance d apart, for some $d \geq 1$. Then i and j cannot distinguish between the following two executions:

1. Nodes i and j have equal clock values. Message delay from i to j is 0, and message delay from j to i is d.
2. Node i's clock value is d less than node j's clock value. Message delay from i to j is d, and message delay from j to i is 0.

Using this idea, we can show that, by choosing message delays and hardware clock rates appropriately, we can create two indistinguishable executions in which a pair of nodes that are distance d apart have d greater skew in one execution than in the other. Thus, in at least one of the executions, the nodes must have at least $\frac{d}{2}$ skew.

Next, we describe how to show that $f(1) = \Omega(\frac{\log D}{\log \log D})$. The basic idea is that we can create a lot of skew in the network without the algorithm "knowing" about it. Later in the execution, we let the algorithm find out about the skew, but show that the algorithm cannot remove the skew fast enough. We need two lemmas to prove the lower bound. The first lemma, which we call the *Add Skew* lemma, states that given two arbitrary nodes, and given an execution α of $\mathcal{A}$ such that a suffix of α satisfies certain bounds on the hardware clock rates and message delays of the nodes, we can find another execution β such that the two given nodes have greater skew at the end of β than at the end of α. The second lemma, which we call the *Bounded Increase* lemma, states that in any execution of $\mathcal{A}$ satisfying some bounds on the hardware clock rates and message delays, no node can increase its logical clock too quickly. The Bounded Increase lemma implies that in any execution of $\mathcal{A}$, the clock skew between two nodes cannot decrease too quickly. Using these lemmas, we prove the lower bound on $f(1)$ by creating an execution in which we repeatedly apply the Add Skew lemma to increase clock skew between some nodes, while limiting the rate at which the skew decreases between those nodes via the Bounded Increase lemma. We show that we can increase the skew faster than the skew decreases for long enough time so that some pair of nodes i and $i+1$ have $\Omega(\frac{\log D}{\log \log D})$ skew between them by the end of the execution.

In the following two sections, we prove the Add Skew and Bounded Increase lemmas. We prove the lower bound on $f(1)$ and formally prove Theorem 1 in Section 8.

6 Add Skew Lemma

In this section, we formally state and prove the Add Skew lemma. We will sometimes talk about the message delay between a pair of nodes *during* a time interval of an execution. By this, we mean the delay of a message sent between the nodes, in either direction, that is *received* during that time interval of the execution. This statement does not talk about the delay of messages that are sent, but not received in the interval.

Lemma 1 (Add Skew lemma) *Let i, j be two nodes with $1 \leq i < j \leq D$. Let $\tau = \frac{1}{\rho}$, $\gamma = 1 + \frac{\rho}{4+\rho}$, $S \geq 0$, $T = S + \tau(j-i)$, and $T' = S + \frac{\tau}{\gamma}(j-i)$. Let α be an execution of $\mathcal{A}$ of duration T, and suppose the following hold:*

1. *The message delay between any two nodes k_1 and k_2 during the time interval $[S, T]$ in α is $\frac{|k_1-k_2|}{2}$.*
2. *Every node has hardware clock rate 1 during the time interval $[S, T]$ in α. That is, $\forall i \forall t \in [S, T]$: $h_i^\alpha(t) = 1$.*

Then there exists an execution β of $\mathcal{A}$ such that the following are true:

1. *$L_i^\beta(T') - L_j^\beta(T') \geq L_i^\alpha(T) - L_j^\alpha(T) + \frac{j-i}{12}$.*
2. *The message delay between any two nodes k_1 and k_2 during the time interval $[0, S]$ is the same in α and β. The message delay between k_1 and k_2 is within $[\frac{|k_1-k_2|}{4}, \frac{3|k_1-k_2|}{4}]$ during the time interval $(S, T']$ in β.*

This lemma says that given two arbitrary nodes $i < j$, and given any execution α of $\mathcal{A}$ satisfying certain bounds on message delays and hardware clock rates during the time interval $[S, T]$, we can find an execution β such that nodes i and j have $\frac{j-i}{12}$ greater clock skew at real time T' in β than they do at real time T in α. That is, β "adds skew" between nodes i and j, as compared to α. Furthermore, all message delays during the time interval $[0, S]$ are the same in α and β, and they fall within certain bounds during the interval $(S, T']$ in β.

Proof. The basic idea is as follows. We create an execution β in which we speed up the hardware clocks of some nodes. We adjust the message delays to and from these nodes appropriately, so that execution β looks indistinguishable from α to all the nodes. In β, the nodes with sped up hardware clocks will also have sped up logical clocks, which allows β to add skew between nodes i and j.

We now define β. The actions of β are a (not necessarily proper) subset of the actions of α. That is, an action π occurs in β only if π occurs in α. Some actions occur at different *real times* in β than they do in α. Some nodes have faster hardware clocks in β than in α. The duration of β is T', whereas the duration of α is T. First, we define T_k, for $1 \leq k \leq D$, as follows

$$T_k = \begin{cases} S & \text{if } 1 \leq k \leq i \\ S + \frac{\tau}{\gamma}(k-i) & \text{if } i < k < j \\ T' & \text{if } j \leq k \leq D \end{cases}$$

Now, for $1 \leq k \leq D$, define the hardware clock rate of node k in β by

$$h_k^\beta(t) = \begin{cases} 1 \text{ if } t \in [0, T_k] \\ \gamma \text{ if } t \in (T_k, T'] \end{cases}$$

The hardware clock rates of the nodes in β are shown in Figure 1.

For each action π which occurs in α, let $\kappa(\pi)$ be the node at which π occurs. Recall that $T_\alpha(\pi)$ is the time at which π occurs in α. Let $R(\pi) = \frac{1}{\gamma}(T_\alpha(\pi) - T_{\kappa(\pi)})$. Define the time when π occurs in β by

$$T_\beta(\pi) = \begin{cases} T_\alpha(\pi) & \text{if } T_\alpha(\pi) \in [0, T_{\kappa(\pi)}] \\ T_{\kappa(\pi)} + R(\pi) & \text{if } T_\alpha(\pi) \in (T_{\kappa(\pi)}, T'] \end{cases}$$

Intuitively, in execution β, we have simply sped up the hardware clock rate of node k to γ, starting at time T_k, for $k = 1, \ldots, D$. Then, we changed the real time at which events in β (which are a subset of the events in α) occur, to ensure that β is still a proper execution of $\mathcal{A}$. Formally, we claim that β satisfies the conclusions of the lemma. This is proven by the following four claims.

Claim 1 *Executions α and β are indistinguishable to all the nodes.*

Proof. Clearly, all actions occur in the same order in α and β.

We now show that each node has the same hardware clock value in α and β when any action occurs. If this holds, then α and β are indistinguishable to all the nodes. Consider any action π occurring at an arbitrary node k in α. For brevity, let $t_0 = T_\alpha(\pi)$. Suppose first that $t_0 \in [0, T_k]$. Then by definition, we have $T_\beta(\pi) = t_0$. Now, we have $h_k^\alpha(t) = h_k^\beta(t) = 1$ for all $t \in [0, t_0]$, so $H_k^\alpha(t_0) = H_k^\beta(t_0)$.

Next, suppose that $t_0 \in (T_k, T]$. Then by definition, $T_\beta(\pi) = T_k + \frac{1}{\gamma}(t_0 - T_k)$. Now, we have $h_k^\beta(t) = 1$ for $t \in [0, T_k]$, and $h_k^\beta(t) = \gamma$ for $t \in (T_k, t_0]$. Thus, $H_k^\beta(T_k + \frac{1}{\gamma}(t_0 - T_k)) = H_k^\alpha(t_0)$. □

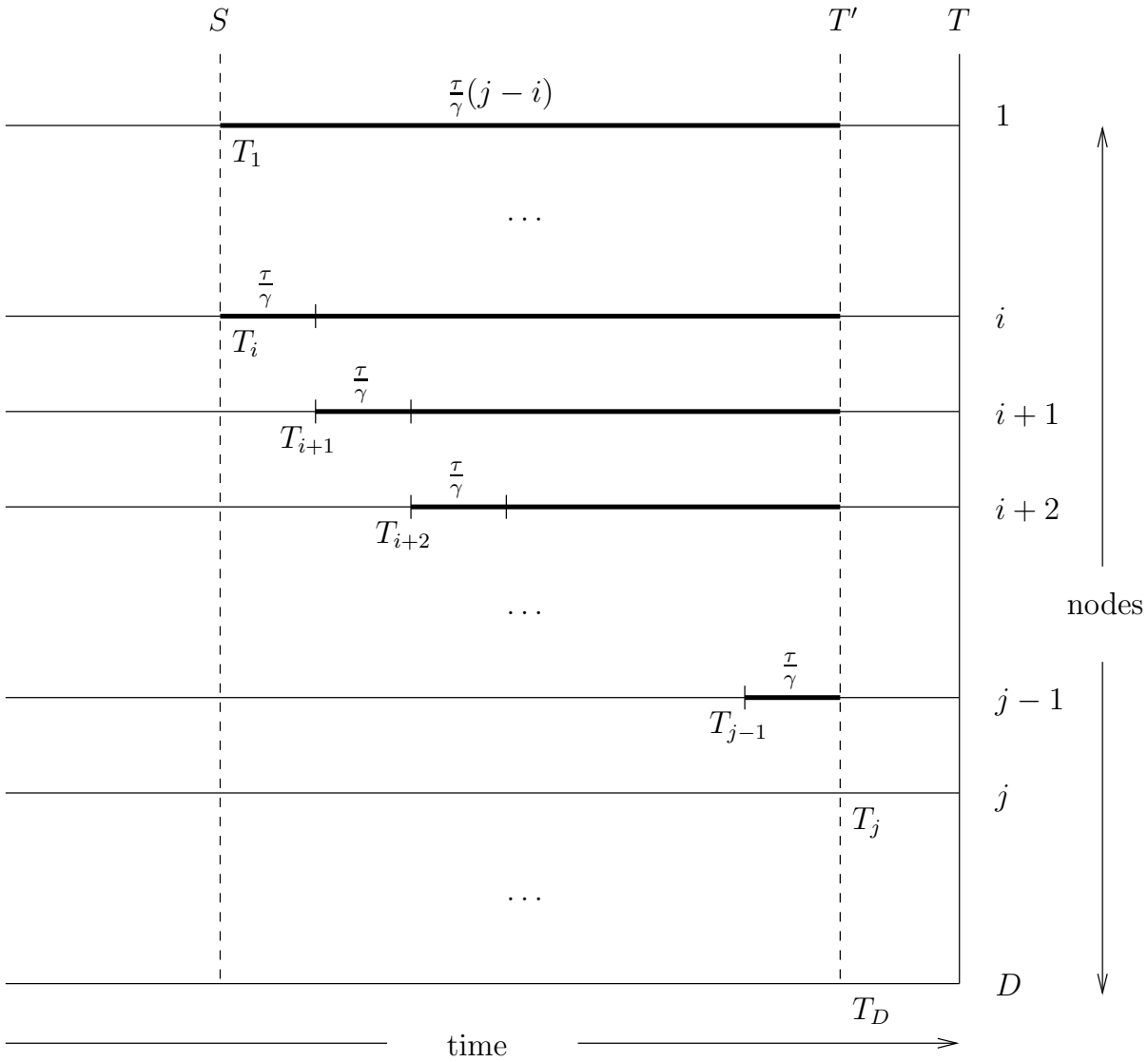

Fig. 1 The hardware clock rates of nodes $1, \ldots, D$ in execution β. Thick lines represents the time interval during which a node has hardware clock rate γ. Node k runs at rate γ for $\frac{\tau}{\gamma}$ time longer than node $k+1$, for $k = i, \ldots, j-1$.

Claim 2 *The hardware clock rate of every node in β is within the correct bounds.*

Proof. The hardware clock rate of any node during the time interval $[0, S]$ is the same in α and β. The minimum hardware clock rate of any node during time interval $(S, T']$ in β is 1, and the maximum clock rate is $\gamma = 1 + \frac{\rho}{4+\rho} < 1 + \rho$. Thus, the claim follows. □

Claim 3 *The message delay is the same in α and β during the time interval $[0, S]$. During the interval $(S, T']$ in β, the message delay between any pair of nodes k_1 and k_2 is within $[\frac{|k_1-k_2|}{4}, \frac{3|k_1-k_2|}{4}]$.*

Proof. Executions α and β are identical up to time S, so the message delay during $[0, S]$ is the same in α and β.

Next, consider a send action π_1 in α, whose corresponding receive action π_2 occurred during $(S, T]$ in α. Let $s_\alpha = T_\alpha(\pi_1), t_\alpha = T_\alpha(\pi_2), s_\beta = T_\beta(\pi_1)$, and $t_\beta = T_\beta(\pi_2)$. We consider two cases. Either the message was sent from a lower indexed node to a higher indexed node, or vice versa. We will show that in the first case, the message delay is not much longer in β than it is in α, while in the second case, the message delay is not much shorter.

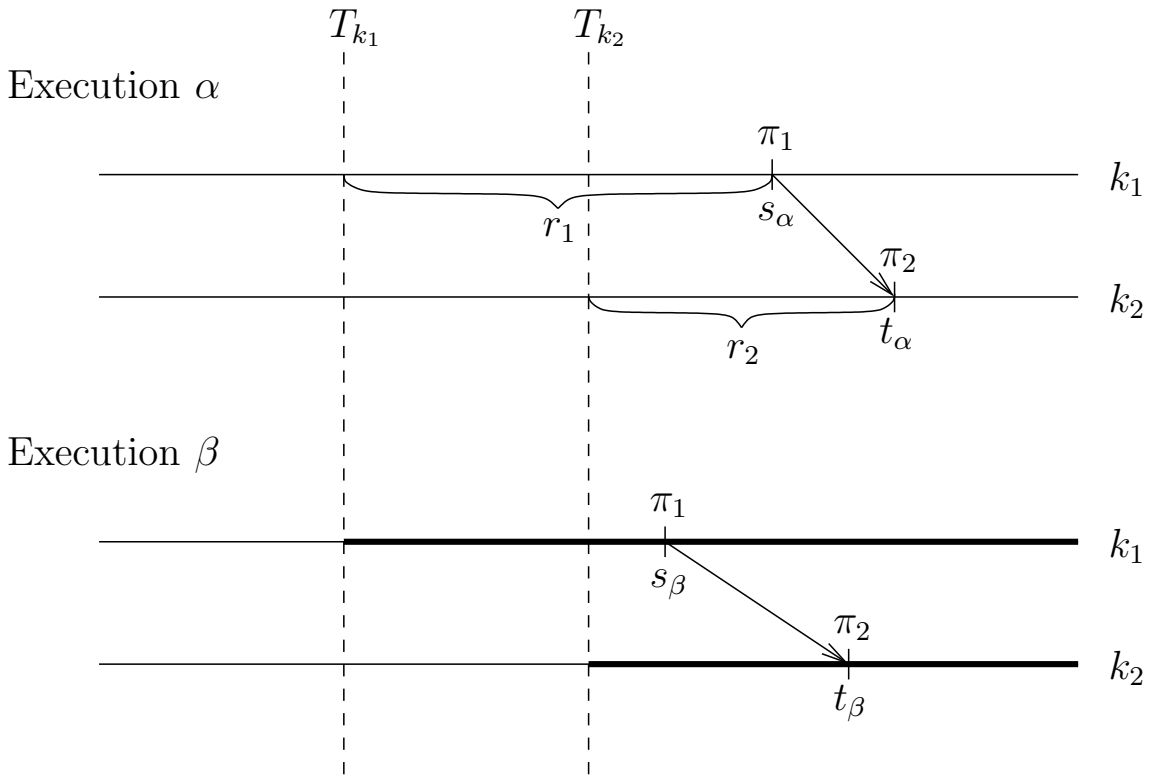

Fig. 2 Node k_1 sends a message to node $k_2 > k_1$. The delay of the message is $\frac{k_2-k_1}{2}$ in execution α, and is within $[\frac{k_2-k_1}{2}, \frac{3(k_2-k_1)}{2}]$ in execution β. Note that the hardware clocks of nodes k_1 and k_2 are running at rate γ during the time interval represented by the thick lines.

In the first case, let k_1 be the sending node, and $k_2 > k_1$ be the receiving node. By the first assumption of the Add Skew lemma, we have $t_\alpha - s_\alpha = \frac{k_2-k_1}{2}$. Define $r_1 = \max(s_\alpha - T_{k_1}, 0), r_2 = \max(t_\alpha - T_{k_2}, 0)$. Please see Figure 2 for an illustration. We claim that $s_\beta = s_\alpha - r_1(1 - \frac{1}{\gamma})$. Indeed, if $r_1 = 0$, then $s_\alpha \leq T_{k_1}$, so by the definition of $T_\beta(\cdot)$, we have $s_\beta = s_\alpha = s_\alpha - r_1(1 - \frac{1}{\gamma})$. If $r_1 > 0$, then we have $s_\beta = T_{k_1} + \frac{1}{\gamma}(s_\alpha - T_{k_1}) = s_\alpha + (T_{k_1} - s_\alpha) - \frac{1}{\gamma}(T_{k_1} - s_\alpha) = s_\alpha - r_1(1 - \frac{1}{\gamma})$. Similarly, we have $t_\beta = t_\alpha - r_2(1 - \frac{1}{\gamma})$. Subtracting, we get $t_\beta - s_\beta = t_\alpha - s_\alpha + (r_1 - r_2)(1 - \frac{1}{\gamma})$.

We now bound $r_1 - r_2$. Suppose first that $r_1 = 0$. Then $s_\alpha \leq T_{k_1}$. We can check that $\frac{\tau}{\gamma} = \frac{4+\rho}{4\rho+2\rho^2} \geq \frac{1}{2}$ for $\rho \in (0, 1)$. So, $t_\alpha = s_\alpha + \frac{k_2-k_1}{2} \leq T_{k_2} = T_{k_1} + \frac{\tau}{\gamma}(k_2 - k_1)$. Thus, we have $r_2 = \max(t_\alpha - T_{k_2}, 0) = 0$, and $r_1 - r_2 = 0$. Next, suppose $r_1 > 0$. Then

$$\begin{aligned} r_1 - r_2 &= s_\alpha - T_{k_1} - \max(t_\alpha - T_{k_2}, 0) \\ &\leq s_\alpha - T_{k_1} - (t_\alpha - T_{k_2}) \\ &= T_{k_2} - T_{k_1} + s_\alpha - t_\alpha \\ &= \frac{\tau}{\gamma}(k_2 - k_1) - \frac{k_2 - k_1}{2} \\ &\leq \frac{\tau}{\gamma}(k_2 - k_1) \end{aligned}$$

Thus, we have

$$\begin{aligned} t_\beta - s_\beta &= t_\alpha - s_\alpha + (r_1 - r_2)(1 - \frac{1}{\gamma}) \\ &\leq t_\alpha - s_\alpha + (r_1 - r_2)(\gamma - 1) \\ &\leq \frac{k_2 - k_1}{2} + \frac{\tau}{\gamma}(\gamma - 1)(k_2 - k_1) \end{aligned}$$

$$= (k_2 - k_1)\left(\frac{1}{2} + \frac{1}{4\tau + 2}\right)$$
$$\leq 3(k_2 - k_1)/4$$

where the first inequality follows because $1 - \frac{1}{\gamma} \leq \gamma - 1$ for all γ, the second equality follows by simplification, and the last inequality follows because $\tau > 1$. Thus, a message from k_1 to k_2 has delay at most $\frac{3(k_2-k_1)}{4}$. Next, we show the delay is at least $\frac{k_2-k_1}{2}$. We have $t_\alpha - s_\alpha = \frac{k_2-k_1}{2}$, and $T_{k_2} - T_{k_1} = \frac{\tau}{\gamma}(k_2 - k_1) \geq \frac{1}{2}(k_2 - k_1)$, and so $r_1 = \max(s_\alpha - T_{k_1}, 0) \geq r_2 = \max(t_\alpha - T_{k_2}, 0)$. Also, we have $\frac{1}{\gamma} \leq 1$. Thus, $t_\beta - s_\beta = t_\alpha - s_\alpha + (r_1 - r_2)(1 - \frac{1}{\gamma}) \geq t_\alpha - s_\alpha = \frac{k_2-k_1}{2}$. Thus, we have shown that all message delays from a smaller indexed node to a larger indexed node are within the bounds required by the lemma.

Next, we consider the case when a node k_2 sends to a node $k_1 < k_2$. Define $r_1 = \max(t_\alpha - T_{k_1}, 0), r_2 = \max(s_\alpha - T_{k_2}, 0)$. As above, we have $s_\beta = s_\alpha - r_2(1 - \frac{1}{\gamma}), t_\beta = t_\alpha - r_1(1 - \frac{1}{\gamma})$, and so $t_\beta - s_\beta = t_\alpha - s_\alpha + (r_2 - r_1)(1 - \frac{1}{\gamma})$. Also as above, we can show that $r_2 - r_1 \geq -(k_2 - k_1)(\frac{\tau}{\gamma} + \frac{1}{2})$. Plugging this into the expression for $t_\beta - s_\beta$ and simplifying, we get that $t_\beta - s_\beta \geq \frac{k_2-k_1}{4}$. Also as above, we have $r_2 \leq r_1$, and so we have $t_\beta - s_\beta \leq t_\alpha - s_\alpha$. Thus, all messages sent from a larger indexed node to a smaller indexed node have delay between $\frac{k_2-k_1}{4}$ and $\frac{k_2-k_1}{2}$. Together with the previous paragraph, this shows that all messages received in β during $(S, T']$ have delays within the required bounds. □

Combining Claims 1, 2 and 3, we get that β is an execution of $\mathcal{A}$. Finally, we show that β increases the skew between nodes i and j.

Claim 4 $L_i^\beta(T') - L_j^\beta(T') \geq L_i^\alpha(T) - L_j^\alpha(T) + \frac{j-i}{12}$.

Proof. From the definition of $H_i^\beta(\cdot)$, we have that $H_i^\beta(T') = H_i^\alpha(T)$, and so because α and β are indistinguishable to i, we have

$$L_i^\beta(T') = L_i^\alpha(T) \tag{1}$$

Also, we have $H_j^\beta(T') = H_j^\alpha(T')$, so that $L_j^\beta(T') = L_j^\alpha(T')$. Now, from the validity requirement in Section 4, we have that $L_j^\alpha(T) - L_j^\alpha(T') \geq \frac{1}{2}(T - T')$. Thus, we get

$$L_j^\beta(T') \leq L_j^\alpha(T) - \frac{1}{2}(T - T') \tag{2}$$

Subtracting equation (2) from equation (1), we get

$$L_i^\beta(T') - L_j^\beta(T') \geq L_i^\alpha(T) - L_j^\alpha(T) + \frac{1}{2}(T - T') \tag{3}$$

We compute

$$T - T' = (S + \tau(j - i)) - (S + \frac{\tau}{\gamma}(j - i))$$
$$= \tau(1 - \frac{1}{\gamma})(j - i)$$
$$\geq \frac{1}{6}(j - i)$$

where the last inequality follows because $\rho < 1$. Plugging this into equation (3), the claim follows. □

7 Bounded Increase Lemma

In this section, we formally state and prove the Bounded Increase lemma.

Lemma 2 (Bounded Increase lemma) *Let α be an execution of $\mathcal{A}$ of duration $T \geq \tau$, and let i be any node. Suppose that the following hold:*

1. *Every node has hardware clock rate within $[1, 1 + \frac{\rho}{2}]$ at all times in α.*
2. *The message delay between i and any node j is within $[\frac{|i-j|}{4}, \frac{3|i-j|}{4}]$ at all times in α.*

Then, for any $t \geq \tau = \frac{1}{\rho}$, we have $L_i^\alpha(t+1) - L_i^\alpha(t) \leq 16f(1)$.

This lemma says that in any execution of $\mathcal{A}$ satisfying some conditions about the hardware clock rates and message delays, no node can increase its logical clock too quickly.

Proof. The idea is the following. Assume that i increases its logical clock very quickly. Then we create another execution β in which we speed up i's hardware clock. We make β indistinguishable from α to all the nodes by adjusting message delays appropriately. Because node i has a faster hardware clock in β, it also has a faster logical clock. But because i increases its logical clock so quickly, we can show that in β, i has a large clock skew compared to a nearby node, which violates the gradient property. Thus, i cannot increase its logical clock too quickly.

Let j be a node such that $d_{i,j} = 1$. Suppose for contradiction that there exists a $t \geq \tau$ such that $L_i^\alpha(t+1) - L_i^\alpha(t) > 16f(1)$. Then there exists $t_0 \in [t, t + \frac{7}{8}]$ such that $L_i^\alpha(t_0 + \frac{1}{8}) - L_i^\alpha(t_0) > 2f(1)$. Define an execution β as follows. β contains the exact same actions as α. Node i's hardware clock rate in β is defined by

$$h_i^\beta(t) = \begin{cases} h_i^\alpha(t) + \frac{\rho}{4} \text{ if } t \in [t_0 - \tau, t_0] \\ h_i^\alpha(t) \qquad \text{otherwise} \end{cases}$$

The hardware clock rates of all nodes other than i are the same in α and β.

Now, we define the real times when actions in β occur. If π is an action of α at a node different from i, then π occurs at the same real time in α and β. If π is an action of α at i, then suppose π occurs when i's hardware clock value is H in α. In β, we define π to occur at the real time t such that $H_i^\beta(t) = H$. Note that with this implicit definition of β, we have *a priori* that α and β are indistinguishable to all the nodes, since all the nodes see the same actions at the same values on their hardware clocks in α and β. Now, we show that β is an execution of $\mathcal{A}$.

First, we show that the hardware clock rates of all nodes in β are within the correct bounds. This is clearly true, since the minimum hardware clock rate of any node in β is at least the minimum rate in α, and the maximum rate of any node in β is $1 + \frac{\rho}{2} + \frac{\rho}{4} \leq 1 + \rho$.

Next, we show the message delays are within the correct bounds. We first prove

Claim 5 $\forall t \leq T - \frac{1}{4} : H_i^\beta(t) \leq H_i^\alpha(t + \frac{1}{4})$.

Proof. Let $s_0 = t_0 - \tau$. Since $h_i^\beta(t) = h_i^\alpha(t)$ for all $t \leq s_0$, the claim holds for all $t \leq s_0$. Now, suppose $t \in (s_0, t_0]$. Then

$$\begin{aligned} H_i^\beta(t) &= H_i^\alpha(t) + (t - s_0)\frac{\rho}{4} \\ &\leq H_i^\alpha(t) + \tau\frac{\rho}{4} \\ &= H_i^\alpha(t) + \frac{1}{4} \end{aligned}$$

Since $h_i^\alpha(r) \geq 1$ for all r, we have $H_i^\alpha(t + \frac{1}{4}) \geq H_i^\alpha(t) + \frac{1}{4} \geq H_i^\beta(t)$, and so the claim holds for all $t \in (s_0, t_0]$. Lastly, for $t \in (t_0, T - \frac{1}{4}]$, we have $h_i^\beta(t) = h_i^\alpha(t)$, so by the same reasoning as above, the claim also holds. □

The above claim shows that any action π at node i which occurs at real time t in α occurs no earlier than $t - \frac{1}{4}$ in β. Thus, since i's message delay with any node j is within $[\frac{|i-j|}{4}, \frac{3|i-j|}{4}]$ in α, the delay of the same message is between $[\frac{|i-j|}{4} - \frac{1}{4}, \frac{3|i-j|}{4} + \frac{1}{4}] \subseteq [0, |i-j|]$ in β. Thus, all the hardware clock rates and message delays in β are within the correct bounds, so β is an execution of $\mathcal{A}$.

We need one more claim.

Claim 6 $H_i^\beta(t_0) \geq H_i^\alpha(t_0 + \frac{1}{8})$.

Proof. From Claim 5, we have $H_i^\beta(t_0) \leq H_i^\alpha(t_0) + \frac{1}{4}$. Also, $H_i^\alpha(t_0 + \frac{1}{8}) \leq H_i^\alpha(t_0) + (1 + \frac{\rho}{2})\frac{1}{8} \leq H_i^\alpha(t_0) + \frac{1}{4}$, where the first inequality follows because all hardware clock rates in α are at most $1 + \frac{\rho}{2}$. Thus, $H_i^\beta(t_0) \geq H_i^\alpha(t_0 + \frac{1}{8})$. □

Now, we prove the lemma. We have

$$\begin{aligned} L_i^\beta(t_0) &\geq L_i^\alpha(t_0 + \frac{1}{8}) \\ &> L_i^\alpha(t_0) + 2f(1) \\ &\geq L_j^\alpha(t_0) + f(1) \\ &= L_j^\beta(t_0) + f(1) \end{aligned}$$

The first inequality follows because α and β are indistinguishable, and by Claim 6, we have $H_i^\beta(t_0) \geq H_i^\alpha(t_0 + \frac{1}{8})$, so that $L_i^\beta(t_0) \geq L_i^\alpha(t_0 + \frac{1}{8})$. The second inequality follows because of our choice of t_0 at the beginning of the proof of the lemma. The third inequality follows because $\mathcal{A}$ satisfies the f-GCS property, so that it must ensure that nodes i and j have logical clock skew which is at most $f(d_{i,j}) = f(1)$ at all times. That is, we must have $L_j^\alpha(t_0) - L_i^\alpha(t_0) \leq f(1)$. The final equality follows because node j has the same hardware clock rate in α and β, so $L_j^\beta(r) = L_j^\alpha(r)$, for all r. However, the above inequalities are a contradiction, because they imply $L_i^\beta(t_0) - L_j^\beta(t_0) > f(1)$, which violates the gradient property. Thus, there does not exist a $t \geq \tau$ such that $L_i^\alpha(t+1) - L_i^\alpha(t) > 16f(1)$. □

8 The Main Theorem

In this section, we prove the lower bound on $f(1)$ and formally prove Theorem 1. Recall that $\ell(\alpha)$ is the duration of an execution α. The following theorem states that there exists an execution of $\mathcal{A}$, at the end of which, a pair of nodes that are at distance 1 from each other have logical clock skew that is $\Omega(\frac{\log D}{\log \log D})$.

Theorem 2 *There exists an execution α of $\mathcal{A}$, and nodes i, j with $d_{i,j} = 1$, such that $|L_i^\alpha(\ell(\alpha)) - L_j^\alpha(\ell(\alpha))|$ $= \Omega(\frac{\log D}{\log \log D})$. Therefore, $f(1) = \Omega(\frac{\log D}{\log \log D})$.*

Proof. The idea is to create an execution in which we repeatedly apply the Add Skew lemma to increase the clock skew between some nodes, while limiting how quickly the skew between those nodes can decrease via the Bounded Increase lemma. We show the skew increases faster than it decreases for long enough time so that two nodes which are distance 1 apart end up with $\Omega(\frac{\log D}{\log \log D})$ clock skew.

We first define some constants we need. Let $n_0 = D - 1$, and $n_k = \frac{n_{k-1}}{384\tau f(1)}$ for $k \geq 1$. To avoid dealing with roundoff errors, we assume that $384\tau f(1)$ is an integer, and that $D-1$ is a power of $384\tau f(1)$. Note that these assumptions are valid because we can easily show that $f(1) \geq \frac{1}{2}$, as in [7]. These assumptions do not affect the asymptotics of the theorem, but they simplify the proof.

We will create a series of executions $\alpha_0, \alpha_1, \ldots$, and also define nodes $i_0, i_1, \ldots$ and $j_0, j_1, \ldots$. Before describing the construction of α_k, i_k and j_k, we first list some properties which we ensure will hold about α_k, i_k and j_k, for all $k = O(\frac{\log D}{\log \log D})$.

Property 1

1. $j_k - i_k = n_k$.
2. $\Delta_k \equiv L_{i_k}^{\alpha_k}(\ell(\alpha_k)) - L_{j_k}^{\alpha_k}(\ell(\alpha_k)) \geq \frac{k}{24} n_k$. That is, the logical clock skew between nodes i_k and j_k at the end of α_k is at least $\frac{k}{24} n_k$.
3. The message delay between any two nodes i and j is $\frac{|i-j|}{2}$ during the time interval $[\ell(\alpha_k) - \tau n_k, \ell(\alpha_k)]$ in α_k. The hardware clock rate of every node during this interval in α_k is 1.
4. The hardware clock rate of any node at any time in α_k is within $[1, 1 + \frac{\rho}{2}]$.
5. The message delay between any two nodes i and j is within $[\frac{|i-j|}{4}, \frac{3|i-j|}{4}]$ at all times in α_k.

Intuitively, condition 2 of Property 1 states that nodes i_k and j_k have clock skew proportional to k times their distance at the end of execution α_k. Condition 3 is used to ensure we can apply the Add Skew Lemma to α_k. Conditions 4 and 5 are to ensure that we can apply the Bounded Increase Lemma to α_k.

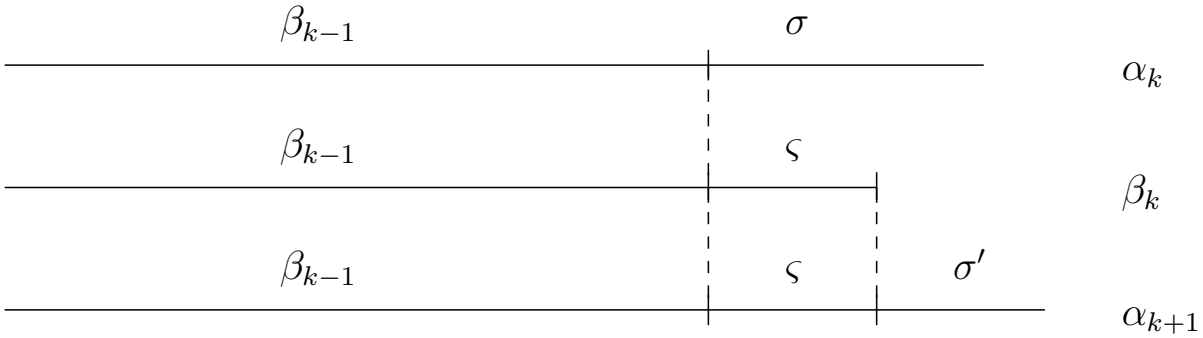

Fig. 3 α_k is the concatenated execution $\beta_{k-1}\sigma$, and satisfies the preconditions of the Add Skew lemma. Applying the Add Skew lemma to α_k produces execution $\beta_k = \beta_{k-1}\varsigma$. Execution α_{k+1} is constructed from β_k by concatenating suffix σ', in which we set the hardware clock rates and message delays so that σ' satisfies the preconditions of the Bounded Increase lemma. This implies that the skew between nodes i_{k+1} and j_{k+1} is at least $\frac{k+1}{24}(j_{k+1} - i_{k+1})$ at the end of α_{k+1}. Lastly, α_{k+1} satisfies the preconditions of the Add Skew lemma, so we can repeat this procedure.

The plan for the construction of α_k, i_k and j_k is as follows. Assume that we have constructed executions $\alpha_0, \ldots, \alpha_k$, for some k, and α_k satisfies Property 1. Then we show that α_k satisfies the preconditions of the Add Skew lemma, so that we can apply the lemma to α_k to obtain an execution β_k which increases the skew between some pair of nodes i_k and j_k. We then extend β_k to a longer execution α_{k+1}. During the extension, we carefully control the hardware clock rates and message delays, so that α_{k+1} satisfies the preconditions of the Bounded Increase lemma. This allows us to conclude that the skew between i_k and j_k did not decrease too much during the extension. In particular, the Add Skew lemma increased the skew twice as much as the skew decreased during the extension. We then show that α_{k+1} satisfies Property 1, which allows us to repeat the procedure. The construction is illustrated in Figure 3.

We now describe the construction. Let α_0 be any execution of $\mathcal{A}$ of duration $\tau(D-1)$, such that

1. The hardware clock rate of any node at any time in α_0 is 1.
2. The message delay between any two nodes i and j is $\frac{|i-j|}{2}$ during all of α_0.

Let $i_0 = 1$, $j_0 = D$. Assume without loss of generality that $L_{i_0}^{\alpha_0}(\ell(\alpha_0)) \geq L_{j_0}^{\alpha_0}(\ell(\alpha_0))$, i.e., that node 1's logical clock value is at least as large as node D's, at the end of α_0. If this is not true, we can simply renumber the nodes in the opposite order. Clearly, α_0 satisfies all the conditions in Property 1.

Next, we describe how to construct execution α_{k+1}, given execution α_k, for $k \geq 0$. We first claim

Claim 7 *α_k satisfies the preconditions of the Add Skew lemma.*

Proof. Instantiate the node "i" in the Add Skew lemma by i_k, and instantiate "j" by j_k. By induction, α_k satisfies Property 1. By conditions 1 and 3 of the property, during the time interval $[\ell(\alpha_k) - \tau n_k, \ell(\alpha_k)] = [\ell(\alpha_k) - \tau(j_k - i_k), \ell(\alpha_k)]$ of α_k, all hardware clock rates are 1, and the message delay between any nodes i and j is $\frac{|i-j|}{2}$. Thus, α_k satisfies the preconditions of the Add Skew lemma. □

By applying the Add Skew lemma to α_k, we obtain an execution β_k such that the following holds:

$$L_{i_k}^{\beta_k}(\ell(\beta_k)) - L_{j_k}^{\beta_k}(\ell(\beta_k)) \geq \Delta_k + \frac{j_k - i_k}{12} \tag{4}$$

$$\geq \frac{k}{24} n_k + \frac{1}{12} n_k \tag{5}$$

$$= \frac{k+2}{24} n_k \tag{6}$$

Now, we extend β_k to an execution which is $n_{k+1}\tau$ (real time) longer. That is, we take the execution β_k, then let algorithm $\mathcal{A}$ run for $n_{k+1}\tau$ time, starting from the last state in β_k. During this extension, we set the hardware clock rates of all nodes to be 1, and set the message delay between any two nodes i and j to be $\frac{|i-j|}{2}$. Also, for any message between i and j which was sent but not received during β_k, we set the delay of that message to be $\frac{|i-j|}{2}$. We call the extended execution α_{k+1}. Our goal in the next four claims is to show that α_{k+1} satisfies the conditions in Property 1, which will allow us to repeat the procedure in the preceding two paragraphs to produce executions α_{k+2}, α_{k+3}, ...

Claim 8 *α_{k+1} satisfies the preconditions of the Bounded Increase lemma.*

Proof. Execution α_{k+1} is of the form $\beta_k\sigma$, where σ is an execution of length $n_{k+1}\tau$. We first show that the β_k portion of α_{k+1} satisfies the preconditions of the Bounded Increase lemma.

To prove that the hardware clock rate of any node is within $[1, 1+\frac{\rho}{2}]$ during β_k, note that, by looking at the construction of β_k in the proof of the Add Skew lemma, we have that β_k and α_k are identical up to time $t_0 = \ell(\alpha_k) - \tau n_k$. So, since α_k satisfies the fourth condition in Property 1, the hardware clock rate of any node is within $[1, 1+\frac{\rho}{2}]$ up to time t_0 in β_k. During the time interval $(t_0, \ell(\beta_k)]$, the hardware clock rate of any node in β_k is within $[1, \gamma] \subseteq [1, 1+\frac{\rho}{2}]$, because the Add Skew lemma sets a node's hardware clock rate to at most γ. Thus, the hardware clock rate of any node is within $[1, 1+\frac{\rho}{2}]$ during the β_k portion of α_k.

To prove that the message delay between any two nodes i and j is within $[\frac{|i-j|}{4}, \frac{3|i-j|}{4}]$ during β_k, we again use the fact that α_k and β_k are identical up t_0. Then, since α_k satisfies the fifth condition in Property 1, we get that the message delay between i and j is within $[\frac{|i-j|}{4}, \frac{3|i-j|}{4}]$ in β_k up to time t_0. In the interval $(t_0, \ell(\beta_k)]$, we have, by the second conclusion of the Add Skew lemma, that the message delay is also within $[\frac{|i-j|}{4}, \frac{3|i-j|}{4}]$.

Lastly, we show that the extension portion of α_{k+1} satisfies the preconditions of the Bounded Increase lemma. But this is clear, because during the extension, we defined the hardware clock rate of all nodes to be 1, and all the message delays to be $\frac{|i-j|}{2}$. Thus, α_{k+1} satisfies the preconditions of the Bounded Increase lemma. □

Claim 9 $L_{i_k}^{\alpha_{k+1}}(\ell(\alpha_{k+1})) - L_{j_k}^{\alpha_{k+1}}(\ell(\alpha_{k+1})) \geq \frac{k+1}{24} n_k$.

Proof. We have $\ell(\alpha_{k+1}) - \ell(\beta_k) = n_{k+1}\tau = \frac{n_k}{384f(1)}$. By Claim 8, α_{k+1} satisfies all the preconditions of the Bounded Increase lemma, so by the Bounded Increase lemma, we have

$$\begin{aligned} L_{j_k}^{\alpha_{k+1}}(\ell(\alpha_{k+1})) &\leq L_{j_k}^{\alpha_{k+1}}(\ell(\beta_k)) + (\ell(\alpha_{k+1}) - \\ &\quad \ell(\beta_k))16f(1) \\ &= L_{j_k}^{\alpha_{k+1}}(\ell(\beta_k)) + \frac{n_k}{24} \end{aligned}$$

Let $\Gamma = L_{i_k}^{\alpha_{k+1}}(\ell(\alpha_{k+1})) - L_{j_k}^{\alpha_{k+1}}(\ell(\alpha_{k+1}))$. Then

$$\begin{aligned} \Gamma &\geq L_{i_k}^{\alpha_{k+1}}(\ell(\beta_k)) - L_{j_k}^{\alpha_{k+1}}(\ell(\alpha_{k+1})) \\ &\geq L_{i_k}^{\alpha_{k+1}}(\ell(\beta_k)) - L_{j_k}^{\alpha_{k+1}}(\ell(\beta_k)) - \frac{n_k}{24} \\ &= \frac{k+2}{24} n_k - \frac{1}{24} n_k \\ &= \frac{k+1}{24} n_k \end{aligned}$$

The first inequality follows because $\ell(\alpha_{k+1}) > \ell(\beta_k)$, and $L_{i_k}^{\alpha_{k+1}}(\cdot)$ is monotonically increasing. The second inequality follows by the first set of inequalities in the claim. The first equality follows from equations (4) to (6). □

Claim 10 *There exist nodes i_{k+1} and $j_{k+1} = i_{k+1} + n_{k+1}$ such that $L_{i_{k+1}}^{\alpha_{k+1}}(\ell(\alpha_{k+1})) - L_{j_{k+1}}^{\alpha_{k+1}}(\ell(\alpha_{k+1})) = \Delta_{k+1} \geq \frac{k+1}{24} n_{k+1}$.*

Proof. By Claim 9, we have $L_{i_k}^{\alpha_{k+1}}(\ell(\alpha_{k+1})) - L_{j_k}^{\alpha_{k+1}}(\ell(\alpha_{k+1})) \geq \frac{k+1}{24} n_k$. Since the nodes are arranged on a line, then by a pigeon-hole type argument, there must be two nodes that are distance n_{k+1} apart whose skew is at least a $\frac{n_{k+1}}{n_k}$ fraction of the skew between nodes i_k and j_k. That is, there exist two nodes $i_{k+1}, j_{k+1} \in [i_k, j_k]$ with $j_{k+1} = i_{k+1} + n_{k+1}$, such that $L_{i_{k+1}}^{\alpha_{k+1}}(\ell(\alpha_{k+1})) - L_{j_{k+1}}^{\alpha_{k+1}}(\ell(\alpha_{k+1})) = \Delta_{k+1} \geq \frac{k+1}{24} n_{k+1}$. □

Claim 11 *Execution α_{k+1} satisfies all the conditions of Property 1.*

Proof. Claim 10 shows that α_{k+1} satisfies the first condition of Property 1. Conditions 2 and 3 are satisfied, because we constructed α_{k+1} as an extension of β_k such that conditions 2 and 3 hold within the last $n_{k+1}\tau$ portion of α_{k+1}. Conditions 4 and 5 hold by Claim 8. □

Claim 12 *For any k with $n_k \geq 1$, there exists a node i such that $L_i^{\alpha_k}(\ell(\alpha_k)) - L_{i+1}^{\alpha_k}(\ell(\alpha_k)) \geq \frac{k}{24}$.*

Proof. By condition 2 of Property 1, we have that $L_{i_k}^{\alpha_k}(\ell(\alpha_k)) - L_{j_k}^{\alpha_k}(\ell(\alpha_k)) \geq \frac{k}{24} n_k$. So, by a pigeonhole type argument, there must exist an i with $i_k \leq i < j_k$ such that $L_i^{\alpha_k}(\ell(\alpha_k)) - L_{i+1}^{\alpha_k}(\ell(\alpha_k)) \geq \frac{k}{24}$. □

Claim 11 shows that we can construct execution α_{k+1} from α_k, as long as $n_{k+1} \geq 1$. By the definition of n_k, we have that $n_k = \frac{D-1}{(384\tau f(1))^k}$. Therefore, we can construct α_k for all k up to $k = \log_{384\tau f(1)}(D-1) = \Omega(\log_{f(1)} D)$. By Claim 12, for every α_k, there exists a node i with $L_i^{\alpha_k}(\ell(\alpha_k)) - L_{i+1}^{\alpha_k}(\ell(\alpha_k)) \geq \frac{k}{24}$. Since $\mathcal{A}$ satisfies the f-GCS property, we must have

$$\begin{aligned} f(1) &\geq L_i^{\alpha_k}(\ell(\alpha_k)) - L_{i+1}^{\alpha_k}(\ell(\alpha_k)) \\ &\geq \frac{k}{24}, \forall k = \Omega(\log_{f(1)} D) \end{aligned}$$

Thus, solving $f(1) = \Omega(\log_{f(1)} D)$ for $f(1)$, we get that $f(1) = \Omega(\frac{\log D}{\log\log D})$, which proves Theorem 2. □

Finally, we use Theorem 2 to prove Theorem 1', which was stated in Section 5. Recall also that Theorem 1' implies the more general Theorem 1.

Proof of Theorem 1'. Let $d \in [1, D-1]$ be arbitrary. As shown in Section 5, there exists an execution of $\mathcal{A}$, call it σ_1, in which two nodes which are distance d apart have $\Omega(d)$ clock skew. This shows that $f(d) = \Omega(d)$.

By Theorem 2, there exists an execution of $\mathcal{A}$, call it σ_2, such that two nodes which are distance 1 apart have $\Omega(\frac{\log D}{\log\log D})$ clock skew. Since $d \geq 1 = d_{i,j}$, then by Property 2 of Section 5, we also have $f(d) = \Omega(\frac{\log D}{\log\log D})$.

To show that there is one particular execution which achieves $\Omega(d + \frac{\log D}{\log\log D})$ between some two nodes at most distance d apart, we simply choose σ_1 as such an execution if $d = \Omega(\frac{\log D(\mathcal{N})}{\log\log D(\mathcal{N})})$, and choose σ_2 if $d = O(\frac{\log D(\mathcal{N})}{\log\log D(\mathcal{N})})$. Thus, the theorem is proved. □

9 Conclusions and Future Work

We have introduced the gradient clock synchronization problem. We have shown the problem's usefulness in the context of sensor and ad-hoc networks, and have also noted that many current clock synchronization algorithms do not solve the problem. We proved that for any f-GCS algorithm, $f(d, \mathcal{N}) = \Omega(d + \frac{\log D(\mathcal{N})}{\log\log D(\mathcal{N})})$. We also discussed some implications of this result.

Though our results show that nearby nodes may have large skew, we did not carefully analyze the *amount of time* for which the large skew may persist. However, it may be seen from the construction given in Section 8 that the $\Omega(\frac{\log D}{\log\log D})$ clock skew between neighboring nodes only lasts $\Theta(1)$ amount of time. Furthermore, it took $\Theta(D)$ amount of time to construct the high skew situation, suggesting that large skew between neighbors is a relatively rare occurrence. Algorithms which rely on gradient clock synchronization, such as TDMA, may exploit this fact. For example, a TDMA algorithm may adjust the granularity of the nodes' broadcast slots depending on the amount of clock skew between nodes. Nevertheless, performing such on-the-fly adjustments and disseminating the new broadcast schedule throughout the network may prove challenging.

The main open problem for GCS is whether there exists any f-GCS algorithm with $f(d, \mathcal{N}) = O(d) + o(D(\mathcal{N}))$. We conjecture the answer is yes, and that there exists a CSA in which nodes at $O(1)$ distance apart have $O(\log D(\mathcal{N}))$ clock skew.

The gradient property emphasizes the local nature of distributed computation, especially in emerging platforms such as mobile networks. We believe a very interesting research direction is the discovery of new distributed algorithms which are more local in nature, or discovering impossibility results against such algorithms.

Acknowledgments. We thank the anonymous referees for their many helpful corrections, comments and suggestions.

References

1. Saâd Biaz and Jennifer L. Welch. Closed form bounds for clock synchronization under simple uncertainty assumptions. *Information Processing Letters*, 80(3):151–157, 2001.
2. Jeremy Elson, Lewis Girod, and Deborah Estrin. Fine-grained network time synchronization using reference broadcasts. *SIGOPS Operating Systems Review*, 36(SI):147–163, 2002.
3. Joseph Y. Halpern, Nimrod Megiddo, and Ashfaq A. Munshi. Optimal precision in the presence of uncertaint. *Journal of Complexity*, 1(2):170–196, 1985.
4. Dilsun K. Kaynar, Nancy Lynch, Roberto Segala, and Frits Vaandrager. Timed I/O automata: A mathematical framework for modeling and analyzing real-time systems. In *Proceedings of the 24th IEEE International Real-Time Systems Symposium*, 2003.
5. Leslie Lamport and P. Michael Melliar-Smith. Synchronizing clocks in the presence of faults. *Journal of the ACM*, 32(1):52–78, 1985.
6. Errol Lloyd. Broadcast scheduling for tdma in wireless multihop networks. *Handbook of wireless networks and mobile computing*, pages 347–370, 2002.
7. Jennifer Lundelius and Nancy Lynch. An upper and lower bound for clock synchronization. *Information and Control*, 62:190–204, 1984.
8. Lennart Meier and Lothar Thiele. Gradient clock synchronization in sensor networks. Technical report, Computer Engineering and Networks Laboratory, Swiss Federal Institute of Technology Zurich, 2005.
9. Rafail Ostrovsky and Boaz Patt-Shamir. Optimal and efficient clock synchronization under drifting clocks. In *Proceedings of the eighteenth annual ACM symposium on Principles of distributed computing*, pages 3–12. ACM Press, 1999.
10. Boaz Patt-Shamir and Sergio Rajsbaum. A theory of clock synchronization. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 810–819. ACM Press, 1994.
11. Hairong Qi, Xiaoling Wang, S. Sitharama Iyengar, and Krishnendu Chakrabarty. Multisensor data fusion in distributed sensor networks using mobile agents. In *Proceedings of the International Conference on Information Fusion*, pages 11–16, 2001.
12. T. K. Srikanth and Sam Toueg. Optimal clock synchronization. *Journal of the ACM*, 34(3):626–645, 1987.
13. Jennifer Lundelius Welch and Nancy Lynch. A new fault-tolerant algorithm for clock synchronization. *Information and Computation*, 77(1):1–36, 1988.