Gradient Clock Synchronization*

Rui Fan †   Nancy Lynch †
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
{rfan, lynch}@theory.csail.mit.edu

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. 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 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.

Categories and Subject Descriptors: F.2.0 [Theory of Computation]: analysis of algorithms and problem complexity—*general*
General Terms: Theory

†This work was supported by the following contracts: DARPA Award number: F33615-01-C-1896, NSF-Texas Engineering Experiment Station grant 64961-CS, NSF CCR-0121277, Air Force Aerospace Research-OSR Contract: #FA9550-04-1-0121.

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

Keywords: clock synchronization, lower bounds, local algorithms, 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 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 [9, 2], on tolerating various types of failures of the nodes and the network [9], and on proving lower bound results about clock skew and communication costs [5, 7, 6]. 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 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 [5] saying that, for any CSA, the worst case clock skew between some pair of nodes in the network is $\Omega(D)$. Most CSAs (e.g., [9]) 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* [8] 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[2].

What kind of gradient can be achieved by a clock synchronization algorithm? When the network contains 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 is that given any clock synchronization algorithm, there exists an execution in which two nodes that are distance d apart, for an arbitrary d, have $\Omega(d + \frac{\log D}{\log \log D})$ clock skew, where D is the diameter of the network. An implication of this result is that an application, such as TDMA [4], 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, and that there exist CSAs that achieve $O(d + \log D)$ skew between nodes that are d distance apart.

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 give an overview of the proof of the lower bound in section 5. We prove two lemmas in sections 6 and 7, then prove the main theorem on GCS in section 8. Finally, we conclude in section 9 with some remarks and open problems.

[2]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[3].

Lundelius-Welch and Lynch [5] 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 $d(1 - \frac{1}{n})$. Biaz and Welch [1] extended the previous result to show that the best synchronization in a network of diameter D is $\frac{D}{2}$. Both papers assume nodes have perfect (non-drifting) clocks. Patt-Shamir and Rajsbaum [7] 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-shot" synchronization, where 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 [7]. Ostrovsky and Patt-Shamir [6] also proved lower bounds using synchronization graphs. However, their work deals with *external synchronization*, in which nodes try to synchronize to a common source time. Our works deals with gradient synchronization. Accurate external synchronization does not in general imply accurate gradient synchronization.

Srikanth and Toueg [9] gave an optimal clock synchronization algorithm, where optimal means that the maximum skew between any pair of nodes is as small as possible given the hardware clock drift. 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 simplified version of their algorithm based on the same idea, which illustrates how their algorithm violates the gradient property. The algorithm works as follows. Nodes periodically broadcast their clock values, and any node receiving a value sets its 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 D, for some constant D, let the distance between y and z be 1, and let the distance between x and z be $D + 1$. By making the message delay D 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 exe-

[3]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 $[1 - \rho, 1 + \rho]$.

cution in which x's clock is D higher than y's clock, which in turn is 1 higher than z's clock. Now, extend this execution by changing the message delay between x and y to be 0, but keeping the delay between y and z 1. Then, when y receives a message from x, y will realize its clock is D lower than x's clock, and so y will increase its clock by D. 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 D, but z has not increased its clock. During this one second interval, y's clock is $D+1$ 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 all other nodes. Each node records its clock value when it receives the signal. Then, in a second phase, each node 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 broadcast is done by radio, it takes about the same amount of time to reach all nodes, so the uncertainty in message delay 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.

3. MODEL

We consider an arbitrary, fixed set of nodes, with each node modeled as a timed automaton [3]. We assume that all the nodes start executing at the same time. We assume all nodes can communicate reliably with each other. We let $d_{i,j}$ be the uncertainty in the message delay between nodes i and j. In particular, a message sent from i to j takes at least 0 time, and at most $d_{i,j}$ time, to arrive at j. We call $d_{i,j}$ the *distance* between i and j, and we denote the *diameter* of the network by $D = \max_{i,j} d_{i,j}$. Since our results are stated in terms of the diameter of the network, we need to define a unit distance. Thus, we assume that $\min_{i,j} d_{i,j} = 1$.

Given an execution α of the system, 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.

Each node i is equipped with a hardware clock. We define the value of the 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 all the hardware clocks have bounded drift. That is, we assume that there exists a constant ρ, where $0 \le \rho < 1$, such that for any execution α the following holds.

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

Each node uses its hardware clock and the messages it receives from other nodes to compute its *logical clock*. Denote the logical clock value of node i at time t in an execution α by $L_i^\alpha(t)$. Because i computes $L_i^\cdot(\cdot)$ based only on readings from $H_i^\cdot(\cdot)$ and on messages i has received, i cannot distinguish between two executions in which the same actions occur at the same readings of H_i, 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. We will use this indistinguishability principle in proving our lower bound.

4. GRADIENT CLOCK SYNCHRONIZATION

In this section, we formally define the gradient clock synchronization property. Before proceeding, we first describe a *validity* condition which we require any CSA to satisfy. Let $\mathcal{A}$ be a CSA, and consider any execution α of $\mathcal{A}$. Then we require the following.

Requirement 1 (Validity). $\forall i \forall t \forall r > 0 : \frac{r}{2} \le L_i^\alpha(t+r) - L_i^\alpha(t)$.

This requirement says that the rate of increase of each node's logical clock must be 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. Almost all existing CSAs satisfy this validity requirement (e.g., [5, 9, 2]). One motivation for the requirement is to rule out some trivial CSAs that circumvent our lower bound, for example, an algorithm that initializes every node's logical clock to 0, and never increases any node's logical clock. Another motivation is that many applications using synchronized clocks require that the clocks do not 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.

Now, we define the gradient property. Let $f : \mathbb{R}^+ \to \mathbb{R}^+$ be any nondecreasing function. We say a CSA $\mathcal{A}$ satisfies the *f-gradient clock synchronization* (*f-GCS* for short) property if, given any set of nodes and distances between the nodes, $\mathcal{A}$ ensures that the difference in logical clock values between any two nodes i and j is at most $f(d_{i,j})$ at all times. More precisely, $\mathcal{A}$ satisfies the following.

Requirement 2 (Gradient Property). *Let n be any positive integer, and let $\mathcal{N}$ be any network with nodes $1, \ldots, n$, and distances between the nodes $\{d_{i,j} \mid 1 \le i, j \le n\}$. In every execution α of $\mathcal{A}$ in $\mathcal{N}$, we have*

$$\forall t \forall i, j : |L_i^\alpha(t) - L_j^\alpha(t)| \le f(d_{i,j})$$

5. OVERVIEW OF LOWER BOUND ON GCS

In this section, we give an overview of our proof that $f(d) = \Omega(d + \frac{\log D}{\log\log D})$, for any f-GCS algorithm. Let $\mathcal{A}$ be an arbitrary f-GCS algorithm. To prove the lower bound, we show the following:

1. For every real number $d \ge 1$, there exists a network containing two nodes at distance d from each other, such that the two nodes have $\Omega(d)$ clock skew in some execution of $\mathcal{A}$ in the network. This implies $f(d) = \Omega(d)$. The reason why we only consider $d \ge 1$ is because we defined 1 as the minimum distance between any two nodes, for each network.

2. For every integer $D \ge 1$, there exists a network of diameter D and an execution of $\mathcal{A}$ in the network, such

that two nodes at distance 1 from each other in the network have $\Omega(\frac{\log D}{\log\log D})$ clock skew in the execution. This implies $f(1) = \Omega(\frac{\log D}{\log\log D})$.

The executions 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 $f(d) = \Omega(d)$. This result is folklore, and it follows from the type of indistinguishability argument used, for example, by Lundelius-Welch and Lynch in [5]. 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.

To show that $f(1) = \Omega(\frac{\log D}{\log\log D})$, let D be an arbitrary positive integer, and let $\mathcal{N}$ be a network with nodes $1, \ldots, D$, such that $d_{i,j} = |i - j|$, for $1 \leq i, j \leq D$. 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 $f(1)$ 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 6.1 (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 subset of the actions of α. That is, an action π occurs in β only if π occurs 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. Define the time when π occurs in β by

$$T_\beta(\pi) = \begin{cases} T_\alpha(\pi) & \text{if } T_\alpha(\pi) \in [0, T_{\kappa(\pi)}] \\ T_{\kappa(\pi)} + \frac{1}{\gamma}(T_\alpha(\pi) - T_{\kappa(\pi)}) & \text{if } T_\alpha(\pi) \in (T_{\kappa(\pi)}, T'] \end{cases}$$

We claim β satisfies the requirements of the lemma. This is proven by the following three claims.

CLAIM 6.2. *Executions α and β are indistinguishable to all the nodes.*

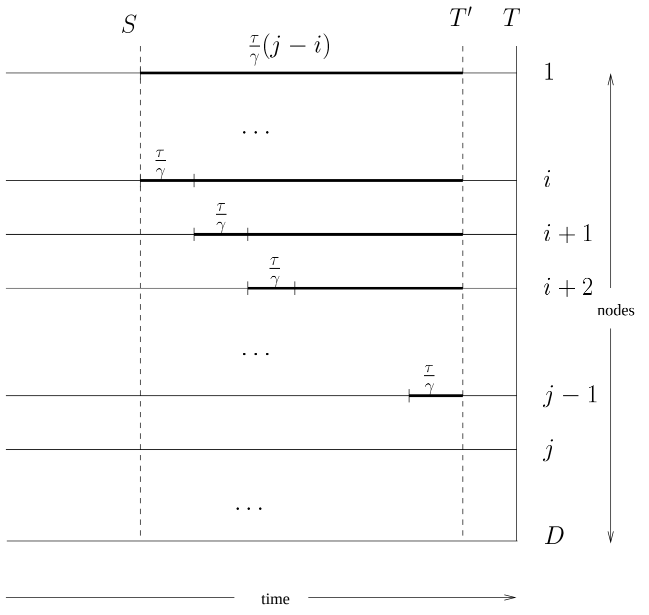

Figure 1: The hardware clock rates of nodes $1, \ldots, D$ in execution β. Thick lines represents the time interval during which a node has hardware clock rate γ. Node k runs at rate γ for $\frac{\tau}{\gamma}$ time longer than node $k+1$, for $k = i, \ldots, j-1$.

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, $H_k^\beta(T_k + \frac{1}{\gamma}(t_0 - T_k)) = H_k^\alpha(t_0)$. □

Claim 6.3. *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 6.4. *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.

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 $r_1 = \max(s_\alpha - T_{k_1}, 0), r_2 = \max(t_\alpha - T_{k_2}, 0)$. We claim that $s_\beta = s_\alpha - r_1(1 - \frac{1}{\gamma})$. Indeed, if $r_1 = 0$, then $s_\alpha \le T_{k_1}$, so by the definition of $T_\beta(\cdot)$, we have $s_\beta = s_\alpha = s_\alpha - r_1(\gamma - 1)$. If $r_1 > 0$, then we have $s_\beta = T_{k_1} + \frac{1}{\gamma}(s_\alpha - T_{k_1}) = s_\alpha + T_{k_1} - s_\alpha - \frac{1}{\gamma}(T_{k_1} - s_\alpha) = s_\alpha - r(1 - \frac{1}{\gamma})$. 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 $s_\alpha \le T_{k_1}$. So $t_\alpha = s_\alpha + \frac{k_2-k_1}{2} \le T_{k_2} = T_{k_1} + \frac{\tau}{\gamma}(k_2 - k_1)$, since $\frac{\tau}{\gamma} \le \frac{1}{2}$. Thus, $r_2 = 0$, and $r_1 - r_2 = 0$. Next, suppose $r_1 > 0$. Then

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

Thus, we have

$$\begin{aligned}
t_\beta - s_\beta &= t_\alpha - s_\alpha + (r_2 - r_1)(1 - \frac{1}{\gamma}) \\
&\le t_\alpha - s_\alpha + (r_2 - r_1)(\gamma - 1) \\
&\le \frac{k_2 - k_1}{2} + \frac{\tau}{\gamma}(\gamma - 1)(k_2 - k_1) \\
&= (k_2 - k_1)\left(\frac{1}{2} + \frac{\tau}{4\tau + 2}\right) \\
&\le 3(k_2 - k_1)/4
\end{aligned}$$

where the first inequality follows because $1 - \frac{1}{\gamma} \le \gamma - 1$ for all γ, and the second equality follows by simplification. Also, since $r_2 \ge r_2$ and $1 \ge \frac{1}{\gamma}$, we have $t_\alpha - s_\alpha \le t_\beta - s_\beta$. Thus, all messages sent from k_1 to k_2 in β have delay between $\frac{k_2-k_1}{2}$ and $\frac{3(k_2-k_1)}{4}$. So, all message delays from a smaller indexed node to a larger indexed node are within the required bounds.

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 $r_2 - r_1 \ge -(k_2 - k_1)(\frac{\tau}{\gamma} + \frac{1}{2})$. Plugging this into the expression for $t_\beta - s_\beta$ and simplifying, we get that $t_\beta - s_\beta \ge \frac{k_2-k_1}{4}$. Since $r_2 \le r_1$, we also have $t_\beta - s_\beta \le t_\alpha - s_\alpha$. 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 6.2, 6.3 and 6.4, we get that β is an execution of $\mathcal{A}$. Finally, we show that β increases the skew between nodes i and j.

Claim 6.5. $L_i^\beta(T') - L_j^\beta(T') \ge L_i^\alpha(T) - L_j^\alpha(T) + \frac{j-i}{12}$.

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

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

Subtracting equation 2 from equation 1, we get

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

We compute

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

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 7.1 (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 $t_0 \in [t, t+1]$ 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

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

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. We define π to occur at the real time t_0 such that $H_i^\beta(t_0) = H$. 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 7.2. $\forall t \leq T - \frac{1}{4} : H_i^\beta(t) \leq H_i^\alpha(t + \frac{1}{4})$.

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 $t \in (s_0, t_0]$. 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 for those t. □

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

Proof. From claim 7.2, we have $H_i^\beta(t_0) = H_i^\alpha(t_0) + \frac{1}{4}$. 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 7.3, 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 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

We can now prove the lower bound on $f(1)$. 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 which is $\Omega(\frac{\log D}{\log\log D})$.

THEOREM 8.1. *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 $384\tau f(1)$. 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 will ensure 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 $\frac{k}{24}$.*

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.*

The plan for our construction 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}, being careful to control the hardware clock rates and message delays during the extension. We show that α_{k+1} satisfies the preconditions of the Bounded Increase lemma, which 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 α_{k+1} satisfies property 1, which allows us to repeat the procedure.

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_k.

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. We first claim

CLAIM 8.2. *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:

$$\begin{aligned} L_{i_k}^{\beta_k}(\ell(\beta_k)) - L_{j_k}^{\beta_k}(\ell(\beta_k)) &\geq \Delta_k + \frac{j_k - i_k}{12} && (4)\\ &\geq \frac{k}{24} n_k + \frac{1}{12} n_k && (5)\\ &= \frac{k+2}{24} n_k && (6)\end{aligned}$$

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}.

CLAIM 8.3. *α_{k+1} satisfies the preconditions of the Bounded Increase lemma.*

Proof. 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 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 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 by our definition of the hardware clock rates and message delays during the extension. Thus, α_{k+1} satisfies the preconditions of the Bounded Increase lemma. □

CLAIM 8.4. $L_{i_k}^{\alpha_{k+1}}(\ell(\alpha_{k+1})) - L_{j_k}^{\alpha_{k+1}}(\ell(\alpha_{k+1})) \geq \frac{k+1}{24}n_k$.

Proof. We have $\ell(\alpha_{k+1}) - \ell(\beta_k) = n_{k+1}\tau = \frac{n_k}{384f(1)}$. By claim 8.3, α_{k+1} satisfies all the preconditions of the Bounded Increase lemma, so by the Bounded Increase lemma, we have

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

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 second inequality follows because α_{k+1} and β_k are the same up to time $\ell(\beta_k)$. The first equality follows from equations 4 to 6. □

CLAIM 8.5. *There exists a node 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 8.4, we have $L_{i_k}^{\alpha_{k+1}}(\ell(\alpha_{k+1})) - L_{j_k}^{\alpha_{k+1}}(\ell(\alpha_{k+1})) \geq \frac{k+1}{24}n_k$. Thus, by a pigeon-hole type argument, we have that there exists $i_{k+1} \geq i_k$ and $j_{k+1} = i_{k+1} + n_{k+1}$ such that $L_{i_{k+1}}^{\alpha_{k+1}}(\ell(\alpha_{k+1})) - L_{j_{k+1}}^{\alpha_{k+1}}(\ell(\alpha_{k+1})) = \Delta_{k+1} \geq \frac{k+1}{24}n_{k+1}$. □

CLAIM 8.6. *Execution α_{k+1} satisfies all the conditions of property 1.*

Proof. Claim 8.5 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 8.3. □

CLAIM 8.7. *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 pigeon-hole 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 8.6 shows 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 8.7, 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 $f(1) = \Omega(\frac{\log D}{\log\log D})$.

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 shown that current clock synchronization algorithms do not solve the problem. We proved that for any f-GCS algorithm, $f(d) = \Omega(d + \frac{\log D}{\log\log D})$. Lastly, we discussed some implications of this result.

The main open problem for GCS is whether there exists any f-GCS algorithm with $f(d) = o(D)$. We believe the answer is yes, and that there exist an f-GCS algorithm with $f(d) = O(d + \log D)$. We are currently analyzing one such candidate algorithm. Another open question is whether our lower bound is tight. We believe that the correct lower bound is $f(d) = \Omega(d + \log D)$.

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.

10. REFERENCES

[1] Saâd Biaz and Jennifer L. Welch. Closed form bounds for clock synchronization under simple uncertainty assumptions. *Inf. Process. Lett.*, 80(3):151–157, 2001.

[2] J. Elson, L. Girod, and D. Estrin. Fine-grained network time synchronization using reference broadcasts, 2002.

[3] 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.

[4] Errol L. Lloyd. Broadcast scheduling for tdma in wireless multihop networks. pages 347–370, 2002.

[5] J. Lundelius-Welch and N. A. Lynch. An upper and lower bound for clock synchronization. *Information and Control*, 62:190–204, 1984.

[6] 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.

[7] 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.

[8] H. Qi, X. Wang, S. S. Iyengar, and K. Chakrabarty. Multisensor data fusion in distributed sensor networks using mobile agents. In *Proceedings of the International Conference on Information Fusion*, pages 11–16, 2001.

[9] T. K. Srikanth and Sam Toueg. Optimal clock synchronization. *J. ACM*, 34(3):626–645, 1987.