Gradient Clock Synchronization in Dynamic Networks

Fabian Kuhn
Computer Science and
Artificial Intelligence Lab
MIT
Cambridge, MA 02139, USA
fkuhn@csail.mit.edu

Thomas Locher
Computer Engineering and
Networks Laboratory
ETH Zurich
8092 Zurich, Switzerland
lochert@tik.ee.ethz.ch

Rotem Oshman
Computer Science and
Artificial Intelligence Lab
MIT
Cambridge, MA 02139, USA
rotem@csail.mit.edu

ABSTRACT

Over the last years, large-scale decentralized computer networks such as peer-to-peer and mobile ad hoc networks have become increasingly prevalent. The topologies of many of these networks are often highly dynamic. This is especially true for ad hoc networks formed by mobile wireless devices.

In this paper, we study the fundamental problem of clock synchronization in dynamic networks. We show that there is an inherent trade-off between the skew $\mathcal{S}$ guaranteed along sufficiently old links and the time needed to guarantee a small skew along new links. For any sufficiently large initial skew on a new link, there are executions in which the time required to reduce the skew on the link to $O(S)$ is at least $\Omega(n/\mathcal{S})$.

We show that this bound is tight for moderately small values of $\mathcal{S}$. Assuming a fixed set of n nodes and an arbitrary pattern of edge insertions and removals, a weak dynamic connectivity requirement suffices to prove the following results. We present an algorithm that always maintains a skew of $O(n)$ between any two nodes in the network. For a parameter $\mathcal{S} = \Omega(\sqrt{\rho n})$, where ρ is the maximum hardware clock drift, it is further guaranteed that if a communication link between two nodes u, v persists in the network for $\Theta(n/\mathcal{S})$ time, the clock skew between u and v is reduced to no more than $O(\mathcal{S})$.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*computations on discrete structures*;
G.2.2 [**Discrete Mathematics**]: Graph Theory—*graph algorithms*;
G.2.2 [**Discrete Mathematics**]: Graph Theory—*network problems*

General Terms

Algorithms, Theory

Keywords

clock synchronization, distributed algorithms, dynamic networks

1. INTRODUCTION

Establishing coordination between participants is at the core of many algorithmic challenges in distributed computing. A fundamental coordination task, and a basic prerequisite for many applications, is achieving a common notion of time. Typically every node in the network has access to a local hardware clock, but the hardware clocks of different nodes run at slightly different rates, and the rates can change over time. In addition, although a bound on the message delays in the network may be known, specific message delays are unpredictable. As a consequence it is generally not possible for any node in the network to get an accurate estimate of the clock values of neighboring nodes.

Operating under these uncertainties, a distributed clock synchronization algorithm computes logical clocks at every node, with the goal of synchronizing these clocks as tightly as possible. Traditionally, distributed clock synchronization algorithms tried to minimize the maximal difference between any two logical clocks in the network. We call this quantity the *global skew* of a clock synchronization algorithm. It is well-known that no algorithm can guarantee a global skew better than $\Omega(D)$, where D is the diameter of the network [3].

In many cases it is more important to tightly synchronize the logical clocks of nearby nodes in the network than it is to minimize the global skew. For example, to run a time division multiple access (TDMA) protocol for coordinating access to the shared communication medium in a wireless network, one only needs to synchronize the clocks of nodes that interfere with each other when transmitting. The problem of achieving synchronization that depends on the distance between the two nodes is called *gradient clock synchronization*. It was introduced in a seminal paper by Fan and Lynch, where it is also shown that surprisingly, a clock skew of $\Omega(\log D/\log\log D)$ cannot be prevented even between immediate neighbors in the network [8]. The maximal difference between the two logical clocks of adjacent nodes in the network is called the *local skew* of a clock synchronization algorithm; for static networks, Lenzen et. al. have recently proven an asymptotically tight bound of $\Theta(\log D)$ for the best possible local skew an algorithm can achieve [12, 13]. For other related work on clock synchronization, see Section 6.

Most existing work on clock synchronization considers static networks. However, many modern networks are inherently dynamic. Typically formed by autonomous agents without central control, nodes can join and leave the network in an arbitrary pattern. In ad hoc networks where often the devices are even assumed to be mobile, the resulting network topology can be highly dynamic even if the set of participating nodes remains stable. Coordination in dynamic networks is challenging, and due to the increasing significance of such networks, it is also particularly important.

In this paper we study the gradient clock synchronization problem in dynamic networks. Because the distance between nodes in the network can change over time, the problem becomes significantly harder in a dynamic setting. Consequently, unlike the static case, the requirements we make on the skew between the logical clocks of different nodes can also change over time. Every new edge that is formed induces a new and stronger constraint on the skew between its endpoints; the algorithm must adapt by reducing the skew on the edge until the new constraint is satisfied. Hence, we distinguish between two requirements: a *stable local skew* bound applies, conceptually, to edges that exist for a long time. This is analogous to the local skew guaranteed by gradient clock synchronization algorithms for static networks. In practice, we impose a weaker *dynamic local skew* bound on all the edges, including new ones. The dynamic local skew bound is a function of how long the edge has existed: the bound starts out weak and grows stronger with time, until in the limit it converges to the stable local skew bound.

The following intuitive example shows that in general, the clock skew on a new edge cannot be reduced too quickly without violating the stable local skew bound on edges that were formed a long time before. Let u and v be two nodes at distance k from each other. As no algorithm can prevent a skew of $\Omega(k)$ between nodes at distance k, a newly formed edge between nodes u and v can carry $\Omega(k)$ local skew. To reduce the skew on the new edge, whichever node is behind must increase its logical clock by a large amount. However, a sudden increase in u or v's clocks will create a large skew along the edges of the old path that connects them. Specifically, if the algorithm guarantees a stable local skew of $\mathcal{S}$, neither u nor v can instantaneously increase their logical clocks to more than $\mathcal{S}$ ahead of their next neighbor along the old path. In turn, when this neighbor realizes it must increase its clock, it cannot increase it to more than $\mathcal{S}$ ahead of *its* next neighbor, and so on. It takes $\Omega(k/\mathcal{S})$ time until the skew can be reduced, as information about the new edge can take time to propagate through the path.

Somewhat surprisingly, the example above is not the worst one possible: adjusting the local skew on a newly formed edge can require even more than $\Omega(k/\mathcal{S})$ time, where k is the previous distance between the endpoints of the new edge. We show that (almost) independent of the initial skew on a new edge, the time required to reduce the initial skew to $\mathcal{S}$ is at least $\Omega(n/\mathcal{S})$ where n is the number of nodes in the system. This is shown in Section 3.

In Section 4 we show that this lower bound is asymptotically tight for moderately small values of $\mathcal{S}$ by extending a simple gradient clock synchronization algorithm described in [14] to the dynamic case. In a static setting, the algorithm of [14] guarantees a local skew of $O(\sqrt{\rho D})$ where ρ is the maximum hardware clock drift. When modeling a dynamic network, we assume that the set of nodes remains fixed, but edges can appear and disappear in a completely arbitrary pattern. If a weak connectivity requirement is satisfied, the algorithm guarantees a global skew of $O(n)$ at all times. Further, for a parameter $\mathcal{S} \geq \sqrt{\rho n}$ and a sufficiently large constant λ, the algorithm guarantees a local skew of at most $\mathcal{S}$ on all edges that are present for at least $\lambda \cdot n/\mathcal{S}$ time. It will be interesting to see whether techniques used in the recent strong static gradient clock synchronization algorithms in [12, 13] can be adapted to the dynamic setting, in order to obtain similar results for smaller values of $\mathcal{S}$. A first step in this direction was recently made in [10], where we extended the algorithm from [13] to handle links with different bounds on message delay [6].

The full version of this paper can be found on the authors' webpages, and includes the full proofs of Theorem 3.2 and the claims in Section 5.

2. PRELIMINARIES

2.1 Notation

Given an undirected static graph $G = (V, E)$, we denote by $\mathcal{P}$ the set of all (undirected) paths in G. For convenience in notation we regard each path $P \in \mathcal{P}$ as a set of edges $P \subseteq E$. We use $\mathcal{P}(u, v)$ to denote all paths between two nodes $u, v \in V$. The distance between two nodes u and v is defined by

$$\mathrm{dist}(u, v) := \min_{P \in \mathcal{P}(u,v)} |P|.$$

The definitions above are used only in the context of a static graph. (We use static graphs in the proof of the lower bound in Section 3). In this work we are often concerned with dynamic graphs, which do not have a static set of edges. We use $V^{(2)} := \{\{u, v\} \mid u, v \in V\}$ to denote the set of all *potential* edges over a static set V of nodes.

2.2 Network Model

We model a dynamic network over a static set V of nodes using Timed I/O Automata (TIOA) [9]. Each node in the network is modelled as a TIOA, and the environment is also modelled as a TIOA. The dynamic behavior of the network is modelled using events of the form $\mathsf{add}(\{u, v\})$ and $\mathsf{remove}(\{u, v\})$ for $u, v \in V$, which correspond to the formation and failure (respectively) of a link between u and v. It is assumed that no edge is both added and removed at the same time.

The history of link formations and failures in a particular execution α, together with an initial set of edges E_0^α, induces a *dynamic graph* $G = (V, E^\alpha)$, where $E^\alpha : \mathbb{R}^+ \to V^{(2)}$ is a function that maps a time $t \geq 0$ to the set of edges (links) that exist in α at time t. We define $E^\alpha(t)$ to be the set of edges that are added no later than time t, and not removed between the last time they are added and time t (inclusive). This includes edges that appear in E_0^α and are not removed by time t. We say that an edge e *exists throughout the interval* $[t_1, t_2]$ in α if $e \in E^\alpha(t_1)$ and e is not removed at any time during the interval $[t_1, t_2]$.

A *static execution* is one in which no edges are added or removed. Formally, α is a static execution if for all $t_1, t_2 \in \mathbb{R}^+$ we have $E^\alpha(t_1) = E^\alpha(t_2)$.

We consider a very general model, in which edges can be inserted or removed arbitrarily, subject only to the following connectivity constraint.

DEFINITION 2.1 (T-INTERVAL CONNECTIVITY). *We say that a dynamic graph $G = (V, E^\alpha)$ is T-*interval connected *if for all $t \geq 0$, the static subgraph $G_{[t,t+T]} = (V, E^\alpha|_{[t,t+T]})$ is connected, where $E^\alpha|_{[t,t+T]}$ is the set of all edges that exist throughout the interval $[t, t + T]$.*

In the sequel we omit the superscript α when it is clear from the context.

We assume that nodes do not necessarily find out immediately about edge insertions and removals[1]. Instead, we assume that there is a parameter $\mathcal{D}$, such that if an edge appears or disappears at time t in an execution, and the change is not reversed by time $t + \mathcal{D}$, the endpoints of the edge find out no later than time $t + \mathcal{D}$. Transient link formations or failures, which do not persist for $\mathcal{D}$ time, may or may not be detected by the nodes affected. We model the discovery by node u of a link formation or failure $X \in \{\mathsf{add}(\{u, v\}), \mathsf{remove}(\{u, v\}) \mid v \in V\}$ by an event $\mathsf{discover}(X)$

[1]Otherwise reference-broadcast-style synchronization would be possible using these events [6].

that occurs at node u. (A $\mathsf{discover}(X)$ event is always preceded by event X itself.)

We also assume reliable FIFO message delivery, with message delays bounded by $\mathcal{T}$. This is modelled using events of the form $\mathsf{send}(u, v, m)$ and $\mathsf{receive}(u, v, m)$ that occur at node u. If node u sends a message to node v at time t, the environment guarantees the following. If edge $\{u, v\}$ exists throughout the interval $[t, t + \mathcal{T}]$, then node v is guaranteed to receive the message no later than time $t + \mathcal{T}$. If edge $\{u, v\}$ exists at time t but is removed at some point in the interval $[t, t+\mathcal{T}]$, there are two possible outcomes: either the message is delivered before the edge is removed, or the message is not delivered and node u discovers the edge removal no later than time $t + \mathcal{D}$. Finally, if edge $\{u, v\}$ does not exist at time t, the message is not delivered, and node u discovers that the edge does not exist no later than time $t + \mathcal{D}$. These definitions correspond to an abstract version of MAC layer acknowledgements.

In the sequel we assume that $\mathcal{D} > \mathcal{T}$, that is, nodes do not necessarily find out about changes to the network within $\mathcal{T}$ time units. This is a reasonable assumption because even if nodes transmit very frequently, as much as $\mathcal{T}$ time may pass without any message being received on a link, and during this time link formations and failures cannot be detected.

2.3 The Clock Synchronization Problem

In the clock synchronization problem, each node $u \in V$ has access to a continuous *hardware clock* $H_u(t)$, which may progress at a different rate than real time. The hardware clocks suffer from *bounded drift* ρ: although they progress at a variable rate, their rate is always between $1 - \rho$ and $1 + \rho$ the rate of real time, so that for any node u and times $t_1 < t_2$ we have

$$(1 - \rho)(t_2 - t_1) \leq H_u(t_2) - H_u(t_1) \leq (1 + \rho)(t_2 - t_1).$$

For simplicity we assume that at the beginning of any execution the hardware clock values are all 0.

The goal of a dynamic clock synchronization algorithm (DCSA) is to output a *logical clock* $L_u(t)$ such that the logical clocks of different nodes are close to each other. In particular we consider two requirements. First, a *global skew constraint* bounds the difference between the logical clocks of any two nodes in the network at all times in the execution. Second, a *dynamic local skew constraint* bounds the skew between two neighbors as a function of how long the link between them has existed. These requirements are formally defined as follows.

DEFINITION 2.2 (GLOBAL SKEW). *A DCSA guarantees a* global skew of $\bar{\mathcal{G}}(n)$ *if in any execution of the algorithm in a network of n nodes, for any two nodes u, v and time $t \geq 0$ we have*

$$L_u(t) - L_v(t) \leq \bar{\mathcal{G}}(n).$$

DEFINITION 2.3 (SKEW FUNCTION). *A function $s : \mathbb{N}\times\mathbb{R}^+\times \mathbb{R}^+ \to \mathbb{R}^+$ (where $\mathbb{R}^+$ are the nonnegative reals) is said to be a* skew function *if the following conditions hold.*

1. *The function $s(n, I, t)$ is non-decreasing in I and non-increasing in t; and*
2. *For all $n \in \mathbb{N}$, $I \in \mathbb{R}^+$, the limit $\lim_{t\to\infty} s(n, I, t)$ is defined and finite; and*
3. *For all $I_1, I_2 \in \mathbb{R}^+$ we have*

$$\lim_{t\to\infty} s(n, I_1, t) = \lim_{t\to\infty} s(n, I_2, t).$$

DEFINITION 2.4 (DYNAMIC LOCAL SKEW). *A DCSA guarantees a* dynamic local skew *of $s : \mathbb{N} \times \mathbb{R}^+ \times \mathbb{R}^+ \to \mathbb{R}^+$, where s is a skew function, if in every execution of the algorithm in a network over n nodes, for any edge $e = \{u, v\}$ and times $t_1 \leq t_2$, if edge e exists throughout the interval $[t_1, t_2]$ in the execution, then we have*

$$|L_u(t_2) - L_v(t_2)| \leq s(n, |L_u(t_1) - L_v(t_1)|, t_2 - t_1).$$

DEFINITION 2.5 (STABILIZING DCSA). *A DCSA $\mathcal{A}$ is said to be* stabilizing *if there is a skew function s such that $\mathcal{A}$ guarantees a dynamic local skew of s. In this case we say that $\mathcal{A}$ guarantees a* stable local skew *of $\bar{s}(n) = \lim_{t\to\infty} s(n, I, t)$ for some $I \in \mathbb{R}^+$.*

Finally, logical clocks have to be strictly increasing and are thus not allowed to temporarily stop. In particular, we require the rate of each logical clock to be at least half the rate of real time; that is, for any node u and times $t_1 \leq t_2$ we require

$$L_u(t_2) - L_u(t_1) \geq \frac{1}{2}(t_2 - t_1).$$

3. LOWER BOUND

We begin our analysis of dynamic clock synchronization algorithms with a lower bound on the time needed to adjust the local skew on a newly formed edge. Specifically, we show that for every sufficiently large initial skew I, the time needed to reduce the skew by a factor of $\Theta(n/\bar{\mathcal{G}}(n))$ is $\Omega(n/\bar{s}(n))$. Thus, there is an inherent tradeoff between the stable skew guaranteed by the algorithm and the time the algorithm requires to reduce the skew on new edges.

The main idea in the proof of the lower bound is to show that because of the local skew guarantee, even nodes that are distant from a new edge may prevent the skew on it from being reduced. To do this we choose two nodes that are far from the new edge and create a large skew between them, while using large message delays to prevent them from hearing about the new edge.

The skew is created using shifting (see, e.g., [16]). A standard shifting argument shows that because of the uncertainty regarding message delays, nodes cannot tell the difference between an execution in which the skew is large, and an execution in which it is not. Consequently the nodes cannot avoid having a large skew between them. In the resulting execution, the message delays on some links are zero, and in the standard construction it is not possible to control which links these will be.

In our proof we require large message delays along certain specific links. A straightforward modification of the argument from [1] and [8] allows us to create large skews while maintaining a predefined pattern of message delays.

More formally, given a static network $G = (V, E)$, a *delay pattern* for G is a pair $M = (E^{\mathrm{C}}, P)$, where $E^{\mathrm{C}} \subseteq E$ is a set of *constrained links* and $P : E^{\mathrm{C}} \to [0, \mathcal{T}]$ assigns a message delay to each constrained link. We define the M*-flexible distance* between two nodes $u, v \in V$ by

$$\mathrm{dist}_M(u, v) = \min_{P\in\mathcal{P}(u,v)} \left|P \setminus E^{\mathrm{C}}\right|.$$

LEMMA 3.1 (MASKING LEMMA). *Let $G = (V, E)$ be a static network, and let $M = (E^{\mathrm{C}}, P)$ be a delay pattern for G. For any time $t > \mathcal{T} \cdot \mathrm{dist}_M(u, v)(1 + 1/\rho)$, there is a static execution α in which*

$$|L_u(t) - L_v(t)| \geq \frac{1}{4}\mathcal{T}\,\mathrm{dist}_M(u, v),$$

and furthermore, in α the delays on every link $e \in E^{\mathrm{C}}$ are in the range $[P(e)/(1 + \rho), P(e)]$. □

The formal statement of the lower bound is as follows.

THEOREM 3.2. *Let $\mathcal{A}$ be a stabilizing DCSA that guarantees a global skew of $\bar{\mathcal{G}}(n)$ and a dynamic local skew of s with a stable local skew of $\bar{s}(n) = o(n)$. Then there exist constants $\lambda, \zeta \geq 0$ such that for all sufficiently large n and I we have*

$$s(n, I, \lambda \cdot \frac{n}{\bar{s}(n)}) \geq \zeta \frac{n}{\bar{\mathcal{G}}(n)} \cdot I.$$

PROOF SKETCH. Consider a network comprising two parallel chains, A and B, joined at both ends w_0, w_n (see Fig. 1(a)). The length of each chain is $n/2$. The two chains exist throughout the construction; new edges are eventually added along the B-chain, but no edges are ever removed.

We wait until the algorithm has stabilized, choosing a sufficiently large time T_s such that $s(n, 0, T_s) \leq \xi \cdot \bar{s}(n)$ where $\xi \in (1, 2]$ is a constant. Then we select two sufficiently large times $T_1, T_2 \geq T_s$, such that $T_2 - T_1 = \lambda(n/\bar{s}(n))$ for some constant λ. Our goal is to add new edges at time T_1, each with a skew of at most I (see Fig. 1(b)), and cause at least one new edge to still have a skew of $\Omega(I \cdot n/\bar{\mathcal{G}}(n))$ at time T_2. This last part is achieved by (a) adding only $O(\bar{\mathcal{G}}(n)/I)$ new edges at time T_1, and (b) creating a skew of $\Omega(n)$ between w_0 and w_n at time T_2. The *average* skew on the new edges must then be $\Omega(I \cdot n/\bar{\mathcal{G}}(n))$, which implies that at least one new edge has a skew of $\Omega(I \cdot n/\bar{\mathcal{G}}(n))$.

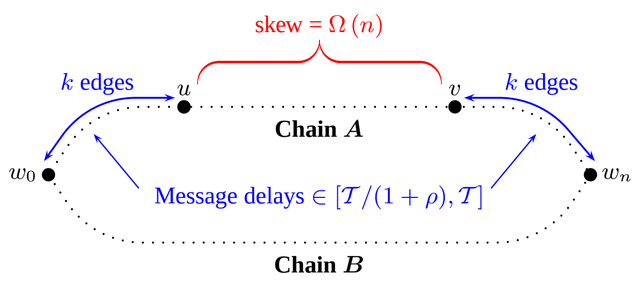
First we show how to create $\Omega(n)$ skew between w_0 and w_n at time T_2. Note that, because of the new edges, the distance between w_0 and w_n at time T_2 is reduced to $O(\bar{\mathcal{G}}(n)/I)$. Standard shifting arguments create a skew proportional to the distance, which is not enough in our case, and hence we use a more roundabout way. Informally, we want to show that w_0 and w_n cannot react quickly enough to the new edges, or they would violate the local skew guarantee along the A-chain.

We choose two nodes u, v on the A-chain such that $\mathrm{dist}(w_0, u) = \mathrm{dist}(w_n, v) = k$, where $k = \Theta(n/\bar{s}(n))$, and $\mathrm{dist}(u, v) = \Omega(n)$. Nodes u and v are "shielded" from events on the B-chain by large message delays (see Fig. 1(a)). We first consider an execution α in which the network is static and no new edges are added at time T_1. Using Lemma 3.1, we create a skew of $\Omega(n)$ between u and v at time T_2 in α, while keeping delays of at least $\mathcal{T}/(1+\rho)$ on all links between w_0 and u and between w_n and v.

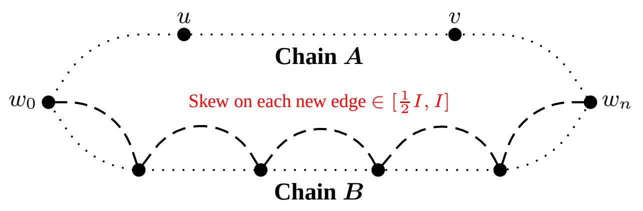
Nodes u, v act as a barrier between w_0 and w_n: the local skew guarantee implies that the clocks of w_0 and w_n cannot be more than $k \cdot \xi\bar{s}(n) = \Theta(n)$ removed from the clocks of u and v respectively. Hence, whenever the skew between u and v is $\Omega(n)$, the skew between w_0 and w_n is also $\Omega(n)$. (See Fig. 1(d), and note that the figure depicts the best-case scenario for the algorithm; it could be, for example, that the skew between w_0 and w_n is actually greater than the skew between u and v).

Finally, we create a new execution β, which is identical to α until time T_1. At time T_1 we add new edges as shown in Fig. 1(b). Recall that at time T_1 the skew on each edge of the B-chain is at most $\xi \cdot \bar{s}(n)$. Thus, we can find a set of edges as shown in Fig. 1(b), such that (a) each edge carries a skew in the range $[I - \xi \cdot \bar{s}(n), I]$, and (b) the skews sum up to at most the skew between w_0 and w_n, which is bounded by $\bar{\mathcal{G}}(n)$. When $I \geq 2\xi \cdot \bar{s}(n)$, the number of edges required is at most $2\bar{\mathcal{G}}(n)/I$.

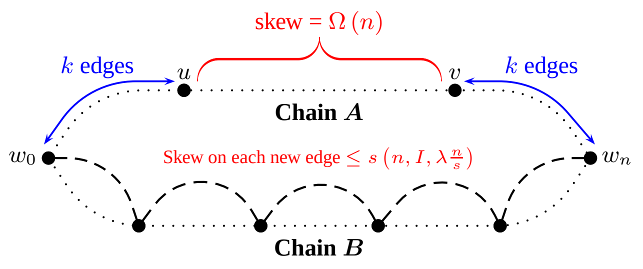
By time T_2 in β, the skew on each new edge must be reduced to at most $s(n, I, T_2 - T_1) = s(n, I, \lambda(n/\bar{s}(n)))$, and consequently the total skew between nodes w_0 and w_n cannot exceed $(2\bar{\mathcal{G}}(n)/I) \cdot s(n, I, \lambda(n/\bar{s}(n)))$ (see Fig. 1(c)). However, in addition to this upper bound on the skew, we can also show that the skew between w_0 and w_n at time T_2 in β is at least $\Omega(n)$: nodes u and v cannot distinguish between α and β until time T_2, since they are shielded from the B-chain by $k = \Theta(n/\bar{s}(n))$ edges with large message delays. At time T_2 in β, nodes u, v have the same skew of $\Omega(n)$ that they have in α, and as argued above, this implies that w_0 and w_n also have $\Omega(n)$ skew between them. Combining the upper and lower bounds on the skew between w_0 and w_n, we see that $s(n, I, \lambda(n/\bar{s}(n)))$ must be at least $\Omega(I \cdot (n/\bar{\mathcal{G}}(n)))$. This concludes the proof. $\square$



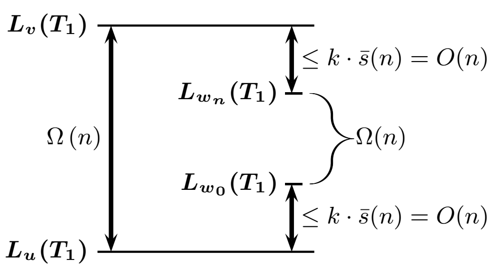
(a) Execution α at time T_2.



(b) Execution β at time T_1 (new edges shown as dashed lines)



(c) Execution β at time T_2



(d) The logical clocks of w_0, u, v, w_n at time T_2 in executions α and β (assuming $L_u(T_1) \leq L_v(T_1)$)

Figure 1: Illustrations for the proof of Theorem 3.2

Theorem 3.2 makes no assumptions on the global skew $\bar{\mathcal{G}}(n)$. However, most static clock synchronization algorithms in the literature guarantee a global skew of $O(D)$ in networks of diameter D. Moreover, all gradient clock synchronization algorithms of which we are aware *rely* on having a global skew of $O(D)$ in order to prove their gradient property [12, 13, 14].

In dynamic graphs the diameter is undefined, and the natural extention is to require a global skew of $\bar{\mathcal{G}}(n) = O(n)$. This is achieved by the algorithm presented in Section 4, and here, too, the global skew bound is used to prove the local skew guarantee. It therefore seems most interesting to consider algorithms that provide a global skew guarantee of $O(n)$. For such algorithms, Theorem 3.2 shows that it takes $\Theta(n/\bar{s}(n))$ time to reduce the initial skew on a new edge by a *constant* fraction.

COROLLARY 3.3. *Let $\mathcal{A}$ be a stabilizing DCSA that guarantees a global skew of $\bar{\mathcal{G}}(n) = O(n)$ and a dynamic local skew of s with a stable local skew of $\bar{s}(n) = o(n)$. Then there exist constants $\lambda, \zeta \geq 0$ such that for all sufficiently large n and I we have*

$$s(n, I, \lambda \cdot \frac{n}{\bar{s}(n)}) \geq \zeta \cdot I.$$

4. A DYNAMIC CLOCK SYNCHRONIZATION ALGORITHM

Next we present a simple DCSA that achieves the tradeoff demonstrated in the previous section.

The algorithm is based on nodes sending each other periodic updates containing their own logical clock value and their estimate for the maximal logical clock in the network. Updates are sent to all neighbors every ΔH subjective time units; that is, if node u sends an update to all its neighbors at real time t, the next time it will send an update is real time t' such that $H_u(t') = H_u(t) + \Delta H$. Since the hardware clock of u progresses at a rate of at least $1 - \rho$, updates are sent at least once every $\frac{\Delta H}{1-\rho}$ real time units.
Define

$$\Delta\mathcal{T} := \mathcal{T} + \frac{\Delta H}{1-\rho}, \qquad \Delta\mathcal{T}' := (1+\rho)\,\Delta\mathcal{T}.$$

Since every node sends messages to all its neighbors at least once every $\frac{\Delta H}{1-\rho}$ time units, and messages take at most $\mathcal{T}$ time units to arrive, $\Delta\mathcal{T}$ is the longest amount of real time that may pass between the receipt of two messages along an edge, provided the edge exists throughout the interval. Since nodes do not have access to real time, they use $\Delta\mathcal{T}'$ to conservatively estimate the time they have to wait between receiving two messages from a neighbor. If $\Delta\mathcal{T}'$ subjective time has passed and a message was not received, the link to that neighbor must have failed.

The algorithm we present here is event-based: nodes react to messages they receive, and to $\mathsf{discover}(X)$ events, where $X \in \{\mathsf{add}(\{u,v\}), \mathsf{remove}(\{u,v\}) \mid v \in V\}$. In addition, each node can schedule delayed events by invoking $\mathsf{set\_timer}(\Delta t,$ timer-ID). If $\mathsf{set\_timer}(\Delta t,$ timer-ID) is called by u at real time t, then at real time t' such that $H_u(t') = H_u(t) + \Delta t$, an alarm(timer-ID) event is triggered at node u. A delayed event can be cancelled by calling cancel(timer-ID).

The algorithm uses two types of timers: the tick timer is set to go off every subjective ΔH time, and a $\mathsf{lost}(v)$ timer is set to go off $\Delta\mathcal{T}'$ subjective time units after a message from v is received.

Throughout the run of the algorithm each node u maintains two sets Γ_u, Υ_u of nodes, with $\Gamma_u \subseteq \Upsilon_u$. The nodes in Υ_u can be thought of as "tentative neighbors" of u: a node v is added to Υ_u when a $\mathsf{discover}(\mathsf{add}(\{u,v\}))$ event occurs at u, and removed when a $\mathsf{discover}(\mathsf{remove}(\{u,v\}))$ event occurs. The criterion for membership in Γ_u is more restrictive: the set Γ_u contains those nodes of Υ_u that u has heard from at most $\Delta\mathcal{T}'$ subjective time units ago. If $\Delta\mathcal{T}'$ subjective time units pass and u does not receive a message from v, it removes v from Γ_u (but not from Υ_u). The nodes in Γ_u are the only ones used to determine node u's logical clock value, since they are the ones for which u has an accurate estimate. However, u sends (or tries to send) periodic updates to all nodes in Υ_u.

In addition to Γ_u and Υ_u, node u maintains the following local variables.

L_u	Node u's logical clock.
$L_u^{\max}$	Node u's estimate for the maximum logical clock in the network.
L_u^v for $v \in \Gamma_u$	Node u's estimate for the current value of node v's logical clock.
C_u^v for $v \in \Gamma_u$	The value of node u's hardware clock when v was last added to Γ_u.

The local variables are modified upon processing the various events as shown in Algorithm 2. Between events, the variables L_u, $L_u^{\max}$ and L_u^v for all $v \in \Gamma_u$ are increased at the rate of u's hardware clock.

Node u uses a non-increasing function $B : \mathbb{R}^+ \to \mathbb{R}^+$ to determine how much perceived skew it is willing to tolerate on each edge $\{u,v\}$ where $v \in \Gamma_u$. The parameter to the function is $(H_u - C_u^v)$, the subjective amount of time that has passed since u discovered the edge. The amount of perceived skew nodes are willing to tolerate starts out very large, and decreases linearly until it reaches a "target skew" of B_0 (a parameter). Specifically, the function B is given by

$$B(\Delta t) := \max\left\{B_0, 5\bar{\mathcal{G}}(n) + (1+\rho)\tau + B_0 - \frac{B_0}{(1+\rho)\tau}\Delta t\right\},$$

where

$$\tau := \frac{1+\rho}{1-\rho}\Delta\mathcal{T} + \mathcal{T} + \mathcal{D},$$

and where $\bar{\mathcal{G}}(n)$ is the bound on the global skew derived in Theorem 5.5 (Section 5.1). Informally, the purpose of B is to have nodes wait $\Theta(\bar{\mathcal{G}}(n)/B_0)$ time units before they start adjusting the skew on new edges (this is formally stated in Section 5.2, Lemma 5.6). Since B_0 is approximately the local skew of the algorithm and we will show that $\bar{\mathcal{G}}(n) = O(n)$, this waiting period matches the lower bound of Section 3. For correctness we require

$$B_0 \geq 2(1+\rho)\tau. \tag{1}$$

The logical clock of each node is adjusted after every event. In each adjustment, node u increases L_u to the largest value that it can, subject to the following constraints: (1) L_u is never decreased, (2) L_u cannot exceed $L_u^{\max}$, and (3) for all $v \in \Gamma_u$, the perceived skew on edge $\{u,v\}$ cannot exceed the value of B for that edge; that is, for all $v \in \Gamma_u$ we require $L_u - L_u^v \leq B(H_u - C_u^v)$. If the constraints cannot be met (e.g., if u has a neighbor that is very far behind), node u cannot make a discrete increase to its logical clock. However, the logical clock continues to increase at the rate of u's hardware clock. The update rule is given by

Procedure `AdjustClock`

```
$L_u \leftarrow \max\left\{L_u, \min\left\{L_u^{\max}, \min_{v\in\Gamma_u}\{L_u^v + B(H_u - C_u^v)\}\right\}\right\}$
```

Algorithm 2: Responses to events that occur at node u

```
when discover(add({$u, v$})) occurs at $u$
    send($u, v, \langle L_u, L_u^{\max}\rangle$)
    $\Upsilon_u \leftarrow \Upsilon_u \cup \{v\}$
    AdjustClock()
end
when discover(remove({$u, v$})) occurs at $u$
    $\Gamma_u \leftarrow \Gamma_u \setminus \{v\}$
    $\Upsilon_u \leftarrow \Upsilon_u \setminus \{v\}$
    AdjustClock()
end
when alarm(lost($v$)) occurs at $u$
    $\Gamma_u \leftarrow \Gamma_u \setminus \{v\}$
    AdjustClock()
end
when receive($u, v, \langle L_v, L_v^{\max}\rangle$) occurs at $u$
    cancel(lost($v$))
    if $v \notin \Gamma_u$ then
        $\Gamma_u \leftarrow \Gamma_u \cup \{v\}$
        $C_u^v \leftarrow H_u$
    $L_u^v \leftarrow L_v$
    $L_u^{\max} \leftarrow \max\{L_u^{\max}, L_v^{\max}\}$
    AdjustClock()
    set_timer($\Delta\mathcal{T}'$, lost($v$))
end
when alarm(tick) occurs at $u$
    forall $v \in \Upsilon_u$ do
        send($u, v, \langle L_u, L_u^{\max}\rangle$)
    AdjustClock()
    set_timer($\Delta H$, tick)
end
```

For simplicity, we assume that all nodes know (upper bounds on) the maximum hardware clock drift ρ, the propagation delay $\mathcal{T}$, as well as the bound $\mathcal{D}$ on the time between topology changes and the nodes discovering these changes. Depending on how edge insertions and deletions are discovered, $\mathcal{D}$ typically is a function of ρ, $\mathcal{T}$, as well as the parameter ΔH. Throughout the remainder of the paper, we assume that $\mathcal{D} > \max\{\mathcal{T}, \Delta H/(1-\rho)\}$. We also assume that all nodes know n, the number of nodes participating in the system. With these assumptions, each node u knows enough to compute the value of B_u^v for every $v \in \Gamma_u$. In particular, all nodes can compute the bound $\bar{\mathcal{G}}(n)$ on the global skew. Note that the same asymptotic results can be achieved if all nodes know n up to a constant factor. This would allow to generalize the setting and also adapt to nodes joining and leaving the system as long as n only changes at a constant rate.

5. ANALYSIS OF THE ALGORITHM

To analyze the algorithm it is important to understand what conditions prevent nodes from making discrete changes to their logical clocks. The following definitions and lemmas characterize these conditions and describe basic properties of the algorithm.
Let

$$B_u^v(t) := B(H_u(t) - C_u^v(t))$$

be the amount of perceived skew node u is willing to tolerate on the edge $\{u, v\}$ at real time t.

DEFINITION 5.1 (BLOCKED NODES). *We say that a node u is* blocked by node v *at time t if $L_u^{\max}(t) > L_u(t)$ and $v \in \Gamma_u(t)$ and $L_u(t) - L_u^v(t) > B_u^v(t)$. In this case we also say that node v blocks node u at time t and that node u is blocked at time t.*

It is easy to see that being blocked prevents node u from increasing its logical clock in procedure `AdjustClock()`. The next lemma shows that being blocked is the *only* reason that can prevent a node from increasing its logical clock to its max estimate.

LEMMA 5.1. *If $L_u^{\max}(t) > L_u(t)$, then node u is blocked at time t.* □

Each node u decides whether or not to increase its clock based on its estimates of its neighbors' clocks, aiming to keep the skew on each edge $\{u, v\}$ no greater than B_u^v. Since the estimate L_u^v may be larger than the real value of v's clock, node u may overshoot the mark. The following lemmas relate node u's estimates to the real clock values of its neighbors, and show that u's perceived skew is not too far off the mark when it decides to make a jump.

LEMMA 5.2 (ESTIMATE QUALITY). *For all $v \in \Gamma_u(t)$ we have $L_u^v(t) \geq L_v(t - \tau)$.* □

LEMMA 5.3 (REAL VS. DESIRED SKEW). *If node u makes a discrete change to its logical clock at time t, then immediately following the jump, for all $v \in \Gamma_u(t)$ we have $L_u(t) - L_v(t) \leq B_u^v(t) + 2\rho \cdot \tau$.* □

5.1 Global Skew

The basic strategy to bound the global skew of our dynamic clock synchronization algorithm is the same as the one used in a static network (see [14]). We first show that for any two nodes u and v, the estimates $L_u^{\max}(t)$ and $L_v^{\max}(t)$ of the maximum clock value in the system are not too far apart. Second, we show that if the global skew exceeds a certain value at time t, the node v with the smallest logical clock value $L_v(t)$ cannot be blocked at time t. By Lemma 5.1, we then have $L_v(t) = L_v^{\max}(t)$, and thus the bound on the maximal difference between two estimates $L_u^{\max}(t)$ and $L_v^{\max}(t)$ also yields a bound on the global skew. We define

$$L^{\max}(t) := \max_{v \in V} L_v^{\max}(t).$$

PROPERTY 1 (RATE OF $L^{\max}$). *The value of $L^{\max}$ increases at a rate of at most $1 + \rho$. That is, for all $t_2 \geq t_1 \geq 0$ we have*

$$L^{\max}(t_2) - L^{\max}(t_1) \leq (1 + \rho)(t_2 - t_1)$$

The error in the estimates $L_u^{\max}(t)$ can be bounded by applying an interval connectivity assumption (Definition 2.1). This is stated by the following lemma.

LEMMA 5.4 (MAX PROPAGATION LEMMA). *If the dynamic graph G is $(\mathcal{T} + \mathcal{D})$-interval connected, then for all $t \geq 0$ and all $u \in V$ it holds that*

$$L^{\max}(t) - L_u^{\max}(t) \leq ((1 + \rho) \cdot \mathcal{T} + 2\rho \cdot \mathcal{D}) \cdot (n - 1).$$

PROOF. All hardware clocks and max estimates are initialized to 0 at time 0, and hence $L^{\max}(0) - L_u^{\max}(0) = 0$. The max clock $L^{\max}$ increases at a rate of no more than $1 + \rho$, and the max estimate $L_u^{\max}(t)$ of any node u increases at a rate of at least $1 - \rho$. Consequently, the difference $L^{\max}(t) - L_u^{\max}(t)$ grows at a rate of no more than $(1+\rho) - (1-\rho) = 2\rho$, and because $\rho < 1$, the claim holds at least until time

$$t = \frac{(1 + \rho)\mathcal{T} + 2\rho \cdot \mathcal{D}}{2\rho} \cdot (n - 1) > (\mathcal{T} + \mathcal{D}) \cdot (n - 1).$$

Thus, it is sufficient to consider times t such that $t > (\mathcal{T} + \mathcal{D}) \cdot (n-1)$.

For $i \in \{1, \ldots, n\}$, define

$$t_i := t - (n-i)(\mathcal{T} + \mathcal{D}),$$

and

$$V_i := \{v \in V \mid L_v^{\max}(t_i) \geq L^{\max}(t_1) + (i-1)(1-\rho)\mathcal{D}\}.$$

Intuitively, V_i is the set of nodes that have a good max estimate at time t_i. We prove by induction on i that for all $i \in \{1, \ldots, n\}$ we have $|V_i| \geq i$.

By definition, $V_1 = \{v \in V \mid L_v^{\max}(t_1) \geq L^{\max}(t_1)\}$. There exists some node v such that $L_v^{\max}(t_1) = L^{\max}(t_1)$, and consequently we get $|V_1| \geq 1$. For the induction step, suppose that $|V_{i-1}| \geq i-1$. By definition, for all $v \in V_{i-1}$ we have

$$L_v^{\max}(t_{i-1}) \geq L^{\max}(t_1) + (i-2)(1-\rho)\mathcal{D}. \tag{2}$$

The max estimate of each node increases at least at the rate of its hardware clock. Consequently, from (2), for all $v \in V_{i-1}$,

$$\begin{aligned} L_v^{\max}(t_i) &\geq L_v^{\max}(t_{i-1}) + (t_i - t_{i-1})(1-\rho) \\ &\geq L^{\max}(t_1) + (i-2)(1-\rho)\mathcal{D} + (t_i - t_{i-1})(1-\rho) \\ &\geq L^{\max}(t_1) + (i-1)(1-\rho)\mathcal{D}, \end{aligned}$$

and hence $V_{i-1} \subseteq V_i$.

If $V \setminus V_{i-1} = \emptyset$, then $|V_i| \geq |V_{i-1}| = n$ and we are done. Otherwise, by $(\mathcal{T}+\mathcal{D})$-interval connectivity of G, there exists an edge $e = \{v, w\}$, where $v \in V_{i-1}$ and $w \in V \setminus V_{i-1}$, such that e exists throughout the interval $[t_{i-1}, t_i]$. There are times $t_{\text{snd}} \geq t_{i-1}$ and $t_{\text{rcv}} \leq t_i$ such that node v sends node w a message containing $L_v^{\max}(t_{\text{snd}})$ at time t_{snd}, and node w receives the message at time t_{rcv} and updates its max estimate. Thus we have

$$\begin{aligned} L_w^{\max}(t_i) &\geq L_w^{\max}(t_{\text{rcv}}) + (1-\rho)(t_i - t_{\text{rcv}}) \\ &\geq L_v^{\max}(t_{\text{snd}}) + (1-\rho)(t_i - t_{\text{rcv}}) \\ &\geq L_v^{\max}(t_{i-1}) + (1-\rho)(t_i - t_{\text{rcv}} + t_{\text{snd}} - t_{i-1}) \\ &\geq L_v^{\max}(t_{i-1}) + (1-\rho)(t_i - t_{i-1} - \mathcal{T}) \\ &= L_v^{\max}(t_{i-1}) + (1-\rho)\mathcal{D} \qquad \text{(From (2))} \\ &\geq L^{\max}(t_1) + (i-1)(1-\rho)\mathcal{D}. \end{aligned}$$

It follows that $w \in V_i$. Since $w \notin V_{i-1}$ and $V_{i-1} \cup \{w\} \subseteq V_i$ we have $|V_i| \geq |V_{i-1}| + 1 \geq i$. This concludes the induction.

The claim we proved implies that $V_n = V$; that is, for all $v \in V$, at time $t_n = t$ we have

$$L_v^{\max}(t) \geq L^{\max}(t_1) + (n-1)(1-\rho)\mathcal{D}. \tag{3}$$

From Property 1,

$$\begin{aligned} L^{\max}(t) &\leq L^{\max}(t_1) + (1+\rho)(t - t_1) \\ &= L^{\max}(t_1) + (1+\rho)(n-1)(\mathcal{T} + \mathcal{D}), \end{aligned} \tag{4}$$

and combining (3) and (4) yields

$$L^{\max}(t) - L_v^{\max}(t) \leq (n-1)\left((1+\rho)\mathcal{T} + 2\rho \cdot \mathcal{D}\right).$$

□

Using the approach sketched above, Lemma 5.4 allows us to prove the following theorem, which bounds the global skew of our algorithm.

THEOREM 5.5 (GLOBAL SKEW). *The algorithm guarantees a global skew of*

$$\bar{\mathcal{G}}(n) := ((1+\rho) \cdot \mathcal{T} + 2\rho \cdot \mathcal{D}) \cdot (n-1).$$

PROOF. We show the stronger statement that at all times t,

$$\forall v \in V \quad : \quad L^{\max}(t) - L_v(t) \leq \bar{\mathcal{G}}(n),$$

and the claim then follows from the definition of $L^{\max}$ and because for all $u \in V$ and times $t \geq 0$ we have $L_u^{\max}(t) \geq L_u(t)$.

For the sake of contradiction, assume that this is not the case. Then there is some time t, node $v \in V$ and $\varepsilon > 0$ such that

$$L^{\max}(t) - L_v(t) \geq \bar{\mathcal{G}}(n) + \varepsilon. \tag{5}$$

Let $\bar{t}$ be the infimum of times when (5) holds for some node v. By Lemma 5.4, we have $L^{\max}(\bar{t}) - L_v^{\max}(\bar{t}) \leq \bar{\mathcal{G}}(n)$, and thus $L_v(\bar{t}) < L_v^{\max}(\bar{t})$. Hence, Lemma 5.1 shows that node v is blocked at time $\bar{t}$. By Definition 5.1, there is a node $u \in \Gamma_v(\bar{t})$ such that $L_v(\bar{t}) - L_v^u(\bar{t}) > B_v^u(\bar{t}) \geq B_0$. From Lemma 5.2, it therefore holds that $L_u(\bar{t} - \tau) < L_v(\bar{t}) - B_0$, and by Property 1 we have $L^{\max}(\bar{t} - \tau) \geq L^{\max}(\bar{t}) - (1+\rho)\tau$. We therefore obtain

$$L^{\max}(\bar{t} - \tau) - L_u(\bar{t} - \tau) > L^{\max}(\bar{t}) - L_v(\bar{t}) - (1+\rho)\tau + B_0.$$

Because we assume that $B_0 \geq (1+\rho)\tau$, this is a contradiction to the assumption that $\bar{t}$ is the infimum of times when (5) is satisfied for the first time for some node v. □

5.2 Local Skew

The local skew guarantee of the algorithm hinges on the fact that the constraint imposed by a newly formed edge is so weak that *no* edge can violate it: for a long time after edge $\{u, v\}$ is detected, the value of B_u^v stays greater than the global skew $\bar{\mathcal{G}}(n)$. Since no edge carries a skew that is greater than $\bar{\mathcal{G}}(n)$, the requirement is trivially satisfied. In fact, only after $\Omega\left(\bar{\mathcal{G}}(n)/B_0\right)$ time can a node be blocked by a new neighbor, and this is formalized by the following lemma.

LEMMA 5.6. *If node v blocks node u at time t, then $v \in \Gamma_u(t')$ for all $t' \in [t - W, t]$, where W (standing for "wait") is given by*

$$W := \left(4\frac{\bar{\mathcal{G}}(n)}{B_0} + 1\right)\tau.$$ □

Informally, the interval $[t - W, t]$ corresponds to the time required according to Theorem 3.2 for information about the new edge to spread throughout the network.

Recall that node u communicates frequently with nodes in Γ_u, so intuitively, any node in Γ_u should have fairly up-to-date information about node u. In particular, Lemma 5.6 implies that if v blocks u, then v has had accurate information about u for a long time prior to t. This observation will allow us to argue that when v blocks u, it is "aware" for a long time that it lags far behind; thus, node v must itself be blocked, or else it would have caught up with node v. Note, however, that the neighbor relation in our algorithm is not symmetric: it is possible that $u \notin \Gamma_v(t)$ even when v blocks u at time t. This can happen if the edge $\{u, v\}$ was recently removed, and the removal was discovered only by v. Therefore, instead of arguing about $L_v^u(t)$, which is undefined if $u \notin \Gamma_v(t)$, we show that $L_v^{\max}(t)$ reflects a recent value of $L_u^{\max}$, as u was *recently* in Γ_v. Since $L_u^{\max} \geq L_u$, this is sufficient for node v to see that it has fallen behind.

LEMMA 5.7 ("EDGE REVERSAL"). *If node v blocks node u at time t, then for all $t' \in [t - W + \Delta\mathcal{T}, t - \mathcal{T}]$ we have $L_v^{\max}(t') \geq L_u^{\max}(t' - \tau)$.* □

We are now ready to prove the local skew guarantee.

THEOREM 5.8. *For any two nodes u, v and time t such that $v \in \Gamma_u(t)$,*

$$L_u(t) - L_v(t) \le B_u^v(t-W) + 2\rho W$$
$$= B_u^v(t-W) + 2\rho\tau\left(4\frac{\bar{\mathcal{G}}(n)}{B_0} + 1\right).$$

PROOF SKETCH. Suppose by way of contradiction that at time t there are two nodes $u, v \in V$ such that $v \in \Gamma_u$ but

$$L_u(t) - L_v(t) > B_u^v(t-W) + 2\rho W.$$

There are two parts to the proof. First, we show that since the skew between u and v is very large, u has been blocked for a long time, and its logical clock has not increased by much. More formally, since B_u^v is non-increasing, for all $t' \in [t-W, t]$ we have

$$B_u^v(t') \le B_u^v(t-W). \tag{6}$$

From Lemma 5.3 and Lemma 5.6, at any time $t' \in [t-W, t]$ node u's logical clock cannot jump to a value that exceeds $L_v(t') + B_u^v(t') + 2\rho\tau \le L_v(t') + B_u^v(t-W) + 2\rho\tau$. Thus, whenever the skew between u and v exceeds $B_u^v(t-W) + 2\rho\tau$, node u's logical clock increases at the rate of u's hardware clock, which is at most $1+\rho$. In addition, node v's logical clock always increases at a rate of at least $1-\rho$. Together we have that whenever the skew between u and v exceeds $B_u^v(t-W) + 2\rho\tau$ it increases at a rate of at most $(1+\rho) - (1-\rho) = 2\rho$.

At time t the skew between u and v exceeds $B_u^v(t-W) + 2\rho W$. By the argument above, the difference $2\rho W - 2\rho\tau$ was built up at a rate of at most 2ρ. It follows that throughout the interval $[t - W + \tau, t]$, node u's logical clock increases at the rate of its hardware clock, and for all $t' \in [t - W + \tau, t]$ we have

$$L_u(t) - L_u(t') \le (1+\rho)(t - t'). \tag{7}$$

In the second part of the proof we argue that node v would not have fallen so far behind node u unless it was itself blocked until very recently by some other node u_2, which is far behind v. And why does u_2 lag behind v? Why, it must *also* have been blocked recently. In this way we construct a chain $u_0, u_1, \ldots$ of nodes, where $u_0 = u$, $u_1 = v$, and each node u_i is blocked by u_{i+1} at time $t_i := t - i\tau$. We are able to extend the chain up to length $\ell + 1$, where

$$\ell := \lfloor \frac{W-\tau}{2\tau} \rfloor. \tag{8}$$

Since u_{i+1} blocks u_i at time t_i, by definition we have $L_{u_i}(t_i) - L_{u_i}^{u_{i+1}}(t_i) > B_{u_i}^{u_{i+1}}(t_i) \ge B_0$, and Lemma 5.2 allows us to translate this into

$$L_{u_i}(t_i) - L_{u_{i+1}}(t_{i+1}) > B_0.$$

Summing the inequalities for all $0 \le i \le \ell + 1$ we get

$$L_u(t) - L_{u_{\ell+1}}(t_{\ell+1}) > (\ell+1)B_0. \tag{9}$$

Note that by definition of ℓ we have $(\ell+1)B_0 > \bar{\mathcal{G}}(n)$; however, this is not quite a contradiction to the global skew, because (9) relates the clocks of u and $u_{\ell+1}$ *at different times*. Now the first part of the proof comes into play: we use (7) to obtain

$$L_u(t) - L_u(t_{\ell+1}) \le (1+\rho)(t - t_{\ell+1}) = (1+\rho)(\ell+1)\tau, \tag{10}$$

which we combine with (9), yielding

$$L_u(t_{\ell+1}) - L_{u_{\ell+1}}(t_{\ell+1}) > (\ell+1)(B_0 - (1+\rho)\tau)$$
$$\overset{(1)}{\ge} \frac{1}{2}(\ell+1)B_0$$
$$\ge \frac{1}{2}\left(\frac{W-\tau}{2\tau}\right) \cdot B_0$$
$$= \bar{\mathcal{G}}(n).$$

This is the contradiction we sought.

Next we describe in more detail how the chain $u_0, \ldots, u_{\ell+1}$ is constructed inductively. At each step we maintain the following three properties:

(1) If $i \ne 0$ then $L_u(t) - L_{u_i}(t_i) > i \cdot B_0$,

(2) For all $t' \in [t_{2\ell - i + 1}, t_i]$ we have $L_{u_i}^{\max}(t') \ge L_u(t' - i\tau)$, and

(3) If $i \le \ell$ then node u_i is blocked at time t_i.

The first property is the one we are really interested in, and the other two are necessary for the induction to go through. Specifically, property (2) is used to show that u_i is blocked: by showing that u_i is "aware" of a recent value of u's logical clock we can show that $L_{u_i}^{\max}(t_i) > L_{u_i}(t_i)$, and then we apply Lemma 5.1 to show that u_i is blocked. This fact is then used to extend the chain further.

The base is $u_0 = u$, which clearly satisfies all three properties. For the induction step, suppose that u_i satisfies the properties, and let u_{i+1} be a node that blocks u_i at time t_i. By definition,

$$L_{u_i}(t_i) - L_{u_i}^{u_{i+1}}(t_i) > B_{u_i}^{u_{i+1}}(t_i) \ge B_0,$$

and using Lemma 5.2 we obtain

$$L_{u_i}(t_i) - L_{u_{i+1}}(t_{i+1}) > B_0.$$

If $i = 0$ this is exactly property (1), and if $i > 0$ we can use the induction hypothesis to obtain property (1).

To show that u_{i+1} satisfies property (2), let $t' \in [t_{2\ell - i}, t_{i+1}]$. Simple math shows that $t' \in [t_i - W + \Delta\mathcal{T}, t_i - \mathcal{T}]$, so we can use Lemma 5.6 to obtain

$$L_{u_{i+1}}^{\max}(t') \ge L_{u_i}^{\max}(t' - \tau).$$

Now we use the induction hypothesis, applied to node u_i at time $t' - \tau$, to get

$$L_{u_{i+1}}^{\max}(t') \ge L_u(t' - \tau - i\tau) = L_u(t' - (i+1)\tau),$$

as desired.

Finally we use properties (1) and (2) to show that node u_{i+1} is blocked at time t_{i+1} (property (3)) if $i \le \ell$. By Lemma 5.1, it is sufficient to establish that $L_{u_{i+1}}^{\max}(t_{i+1}) > L_{u_{i+1}}(t_{i+1})$. Property (1) shows that

$$L_{u_{i+1}}(t_{i+1}) < L_u(t) - (i+1)B_0,$$

and property (2) shows that

$$L_{u_{i+1}}^{\max}(t_{i+1}) \ge L_u(t_{i+1} - (i+1)\tau) = L_u(t_{2i+2}).$$

To relate the two bounds we use (7), which gives us

$$L_u(t_{2i+2}) \ge L_u(t) - 2(1+\rho)(i+1)B_0$$
$$\overset{(1)}{\ge} L_u(t) - (i+1)B_0.$$

This concludes the induction. □

Theorem 5.8 describes the local skew guarantee from a point of view that is subjective to node u: the statement of the theorem assumes that $v \in \Gamma_u$, and the value of B_u^v depends on the local variables C_u^v and H_u. In particular, H_u may progress at a rate slower than real time, and v may not be in Γ_u when the edge $\{u, v\}$ is formed (although it is added quickly). The following corollary states the "objective" local skew guarantee of the algorithm.

COROLLARY 5.9. *The algorithm guarantees a dynamic local skew of*

$$\begin{aligned} s(n, I, \Delta t) &= B\left((1-\rho)(\Delta t - W - \tau)\right) + 2\rho W \\ &= \Theta\left(\max\left\{B_0, n - \frac{B_0}{1+\rho} \cdot \Delta t\right\} + \frac{\rho n}{B_0}\right), \end{aligned}$$

regardless of the initial skew I on the edge.

COROLLARY 5.10. *If the parameter B_0 is chosen as $B_0 \geq \lambda\sqrt{\rho n}$ for a constant $\lambda > 0$, the stable local skew of the algorithm is $O(B_0)$. Further, the time to reach this stable skew on a new edge is $O(n/B_0)$. Hence, for this choice of B_0, the trade-off achieved by the algorithm asymptotically matches the trade-off given by the lower bound in Theorem 3.2.*

6. RELATED WORK

Being a fundamental problem in distributed computing, it is not surprising that there is a rich literature on clock synchronization algorithms and lower bounds. Until recently, the work on clock synchronization focused on global synchronization, i.e., on minimizing the maximal clock difference between any two nodes in the system. Essentially all lower bounds on distributed clock synchronization use the *shifting* technique introduced in [15], which exploits uncertainty resulting from unknown message delays, the *scaling* technique from [5], which uses uncertainty that arises as a consequence of different clock rates, or a combination of the two techniques. Using the shifting technique, it is shown in [3] that even if clocks experience no drift, a clock skew of $D/2$ can not be avoided in a network of diameter D. In light of this result, the algorithm described in [20], which guarantees a global skew of $O(D)$, is asymptotically optimal. In [1], Attiya et. al. show that no two nodes can avoid a skew that is equal to the cumulative uncertainty on the minimum-uncertainty path between them. We generalize this result slightly in Section 3 (Lemma 3.1), and use it to prove our lower bound.

A number of related algorithms and lower bounds for varying models and with different properties have been described (see e.g. [1, 2, 7, 18, 19]). All algorithms described in these papers do not guarantees a skew between neighboring nodes that is better than $O(D)$. Recognizing this limitation of existing algorithms, Fan and Lynch introduce the gradient clock synchronization problem in [8], and show that on a path of length D, no clock synchronization algorithm can avoid incurring a skew of $\Omega(\log D / \log\log D)$ between adjacent nodes. This lower bound has recently been improved to $\Omega(\log D)$ in [13].

The first algorithm to guarantee a non-trivial local skew was described by Locher and Wattenhofer in [14]. The algorithm in [14] guarantees a local skew of $O(\sqrt{\rho D})$ between any two neighbors in a network of diameter D, where ρ denotes the maximal hardware clock drift. The algorithm of [14] forms the basis for the dynamic gradient clock synchronization algorithm described in this paper. For static networks, Lenzen et. al. recently improved the upper bound to $O(\log D)$ in [12, 13]. This is asymptotically optimal in light of the lower bound from [13]. All non-trivial gradient clock synchronization algorithms of which we are aware [12, 13, 14] also guarantee a global skew of $O(D)$. (Note that the lower bounds of [8, 13] make no assumptions about the global skew.)

Most closely related to the dynamic clock synchronization problem considered in this work are algorithms that cope with faulty nodes (e.g. [4, 5, 11, 17]). This line of research goes far beyond studying crash failures and message omissions, and describes algorithms that can cope with Byzantine faults, a topic that is out of the scope of the present paper. However, none of these papers consider a truly dynamic setting; in particular, the results rely on the fact that a considerable part of the network remains non-faulty and stable. Additionally, the algorithms and lower bounds described in these papers focus solely on global synchronization, and do not guarantee a local skew that is smaller than the global skew. To the best of our knowledge, the present paper is the first to look at gradient clock synchronization in dynamic networks.

7. CONCLUSION

We have established fundamental trade-offs for gradient clock synchronization algorithms in dynamic networks. First, the time to adjust the skew on a newly formed edge is inversely proportional to the skew one is willing to tolerate on well-established edges. Hence, having a stronger skew requirement in stable conditions impairs the ability to adapt to dynamic changes. Second, contrary to what one might initially think, reducing the skew on edges with a small initial skew turns out to be as hard as reducing the skew on edges with a large initial skew. The time needed in both cases is linear in the global skew bound of the algorithm and is thus at least linear in n.

It will be interesting to see whether the trade-off established by our algorithm can also be achieved for smaller stable skew bounds. In particular, it will be interesting to see whether the techniques developed in [12, 13] to guarantee a local skew of $O(\log n)$ in the static case can be adapted for the dynamic setting. Note, however, that as a consequence of the lower bound proven in Section 3, such an improved local skew bound necessarily comes at the cost of worse adaptability to topology changes.

In this paper we used a weighted-graph approach to deal with the dynamic topology: in the algorithm of Section 4, each edge carries a weight, which starts out very large when the edge first appears and decreases over time. We use the dynamic weights to gradually decrease the effective diameter of the graph, giving nodes time to adapt to the appearance of new edges. In a companion paper [10] we use a similar approach to incorporate reference broadcast synchronization in the algorithm from [13]. In that case the weight of the edge has the traditional meaning in the context of clock synchronization: it corresponds to the uncertainty along the edge. It is our hope that extending the algorithm from [13] to the weighted-graph model will serve as a first step towards a dynamic clock synchronization algorithm with $O(\log n)$ stable local skew, but this seems challenging.

An additional obvious generalization would be to incorporate node insertions and deletions in the dynamic graph model. As long as nodes join and leave at a constant rate, it might be possible to be able to adapt all the parameters used sufficiently quickly in order to still guarantee the same basic results. The details of such a protocol as well as possible limitations on how fast one can adapt to changes of the network size remain interesting open questions.

8. ACKNOWLEDGEMENTS

The authors would like to thank the anonymous reviewers for their valuable comments on the phrasing of Theorem 3.2. We also wish to thank Nancy Lynch for many fruitful discussions.

9. REFERENCES

[1] H. Attiya, D. Hay, and J. Welch. Optimal clock synchronization under energy constraints in wireless ad-hoc networks. In *Proc. of 9th Int. Conf. on Principles of Distributed Systems (OPODIS)*, pages 221–234, 2005.

[2] H. Attiya, A. Herzberg, and S. Rajsbaum. Optimal clock synchronization under different delay assumptions. *SIAM Journal on Computing*, 25(2):369–389, 1996.

[3] S. Biaz and J. Welch. Closed form bounds for clock synchronization under simple uncertainty assumptions. *Information Processing Letters*, 80(3):151–157, 2001.

[4] D. Dolev, J. Halpern, B. Simons, and R. Strong. Dynamic fault-tolerant clock synchronization. *Journal of the ACM*, 42(1):143–185, 1995.

[5] D. Dolev, J. Halpern, and H. Strong. On the possibility and impossibility of achieving clock synchronization. *Journal of Computer and System Sciences*, 32(2):230–250, 1986.

[6] J. Elson, L. Girod, and D. Estrin. Fine-grained network time synchronization using reference broadcasts. *ACM SIGOPS Operating Systems Review*, 36(SI):147–163, 2002.

[7] R. Fan, I. Chakraborty, and N. Lynch. Clock synchronization for wireless networks. In *Proc of 8th Int. Conf. on Principles of Distributed Systems (OPODIS)*, pages 400–414, 2004.

[8] R. Fan and N. Lynch. Gradient clock synchronization. *Distributed Computing*, 18(4):255–266, 2006.

[9] D. K. Kaynar, N. Lynch, R. Segala, and F. Vaandrager. *The Theory of Timed I/O Automata (Synthesis Lectures in Computer Science)*. Morgan & Claypool Publishers, 2006.

[10] F. Kuhn and R. Oshman. Gradient clock synchronization using reference broadcasts. arXiv:0905.3454v1, 2009. `http://arxiv.org/abs/0905.3454v1`.

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

[12] C. Lenzen, T. Locher, and R. Wattenhofer. Clock synchronization with bounded global and local skew. In *Prof. of 49th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 500–510, 2008.

[13] C. Lenzen, T. Locher, and R. Wattenhofer. Tight bounds for clock synchronization. In *Proc. of 28th ACM Symp. on Principles of Distributed Computing (PODC)*, 2009.

[14] T. Locher and R. Wattenhofer. Oblivious gradient clock synchronization. In *Proc. of 20th Int. Symp. on Distributed Computing (DISC)*, pages 520–533, 2006.

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

[16] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996.

[17] K. Marzullo and S. Owicki. Maintining the time in a distributed system. In *Proc. of 2nd ACM Symp. on Principles of Distributed Computing (PODC)*, pages 44–54, 1983.

[18] R. Ostrovsky and B. Patt-Shamir. Optimal and efficient clock synchronization under drifting clocks. In *Proc. of 18th ACM Symp. on Principles of Distributed Computing (PODC)*, pages 400–414, 1999.

[19] B. Patt-Shamir and S. Rajsbaum. A theory of clock synchronization. In *Proc. of 26th ACM Symp. on Theory of Computing (STOC)*, pages 810–819, 1994.

[20] T. Srikanth and S. Toueg. Optimal clock synchronization. *Journal of the ACM*, 34(3):626–645, 1987.