Optimal Gradient Clock Synchronization in Dynamic Networks

Fabian Kuhn
Faculty of Informatics
University of Lugano
fabian.kuhn@usi.ch

Christoph Lenzen
Computer Engineering and Networks Laboratory
ETH Zurich
lenzen@tik.ee.ethz.ch

Thomas Locher
IBM Zurich Research Laboratory
thl@zurich.ibm.com

Rotem Oshman
Computer Science and Artificial Intelligence Lab
MIT
rotem@csail.mit.edu

ABSTRACT

We study the problem of clock synchronization in highly dynamic networks, where communication links can appear or disappear at any time. The nodes in the network are equipped with hardware clocks, but the rate of the hardware clocks can vary within specific bounds, and the estimates that nodes can obtain about the clock values of other nodes is inherently inaccurate. Our goal in this setting is to output a logical clock at each node, such that the logical clocks of any two nodes are not too far apart, and nodes that remain close to each other in the network for a long time are better synchronized than far-apart nodes. This property is called *gradient clock synchronization.*

Gradient clock synchronization has been widely studied in the static setting, where the network topology does not change. We show that the asymptotically optimal bounds obtained for the static case also apply to our highly dynamic setting: if two nodes remain at distance d from each other for sufficiently long, it is possible to synchronize their clocks to within $\mathcal{O}(d \log(D/d))$, where D is the diameter of the network. This is known to be optimal for static networks, and since a static network is a special case of a dynamic network, it is optimal for dynamic networks as well. Furthermore, we show that our algorithm has optimal *stabilization time*: when a path of length d appears between two nodes, the time required until the skew between the two nodes is reduced to $\mathcal{O}(d \log(D/d))$ is $\mathcal{O}(D)$, which we prove is optimal.

1. INTRODUCTION

A core algorithmic problem in distributed computing is to establish coordination among the participants of a distributed system, which is often achieved through a common notion of time. Typically, every node in a network has its own local hardware clock, which can be used for this purpose; however, hardware clocks of different nodes run at slightly different rates, and the rates can change over time. This *clock drift* causes clocks to drift out of synch, requiring periodic communication to restore synchronization. However, communication is typically subject to delay, and although an upper bound on the delay may be known, specific message delays are unpredictable. Consequently, estimates for the current local time at other nodes are inherently inaccurate.

A distributed clock synchronization algorithm computes *logical clocks* at every node, and the goal is to synchronize these clocks as tightly as possible. Traditionally, distributed clock synchronization algorithms focuse on minimizing the *clock skew* between the logical clocks of any two nodes in the network. The clock skew between two clocks is simply the difference between the two clock values. The maximum clock skew that may occur in the worst case between any two nodes at any time is called the *global skew* of a clock synchronization algorithm. A well-known result states that no algorithm can guarantee a global skew better than $\Omega(D)$, where D denotes the diameter of the network [1]. However, 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, if a time division multiple access (TDMA) protocol is used to coordinate access to a shared communication medium in a wireless sensor network, it suffices to synchronize the clocks of nodes that interfere with each other when transmitting. The problem of providing better guarantees on the synchronization quality between nodes that are closer is called *gradient clock synchronization.* The problem was introduced in a seminal paper by Fan and Lynch [5], where the authors show that a clock skew of $\Omega(\log D / \log\log D)$ cannot be prevented between immediate neighbors in the network. The largest possible clock skew that may occur between the logical clocks of any two adjacent nodes at any time is called the *local skew* of a clock synchronization algorithm. For static networks, it has been proven that the best possible local skew that an algorithm can achieve is bounded by $\Theta(\log D)$ [10, 11].

While tight bounds have been shown for the static model, the dynamic case has not been as well understood. A dynamic network arises in many natural contexts: for example, when nodes are mobile, or when communication links are unreliable and may fail and recover. The dynamic network model we consider in this paper is general: it allows communication links to appear and disappear arbitrarily, subject only to a global connectivity constraint (which is required to synchronize all nodes to each other). Hence the model is suitable for modeling various types of dynamic networks which remain connected over time.

In a dynamic network the distances between nodes change over time, as communication links appear and disappear. Consequently, we divide the synchronization guarantee into two parts: a *global skew guarantee* bounds the skew between any two nodes in the network at any time; this bound applies always, and does not depend on changes to the network topology. The second part, a *dynamic gradient skew guarantee*, bounds the skew between two nodes as a function of the distance between them and how long they remain at that distance.

In [7], three of the authors showed that a clock synchronization algorithm cannot react immediately to the formation of new links, and that a certain *stabilization time* is required before the clocks of newly-adjacent nodes can be brought into synch. The stabilization time is inversely related to the synchronization guarantee: the tighter the synchronization required in stable state, the longer the time to reach that state. Intuitively, this is because when strict synchronization guarantees are imposed, the algorithm cannot change clock values quickly without violating the guarantee, and hence it takes longer to react. The algorithm given in [7] achieved the optimal trade-off between skew bound and stabilization time; however, the local skew bound it achieved was $\mathcal{O}(\sqrt{n})$, which is far from optimal.

In this paper we describe two algorithms which achieve the same asymptotically optimal skew bounds as in the static model: if two nodes remain at distance d for sufficiently long, the skew between them is reduced to $\mathcal{O}(d\log(D/d))$, where D is the dynamic diameter of the network (corresponding roughly to the time it takes for information to reach from one end of the network to the other). The two algorithms differ in the time required to reach this guarantee: their stabilization time is $\mathcal{O}(D\log D)$ and $\mathcal{O}(D)$, respectively. The first algorithm, which stabilizes in $\mathcal{O}(D\log D)$ time, is much simpler to describe, and this extended abstract will focus on it. The second algorithm, which stabilizes in $\mathcal{O}(D)$, is described in full in the accompanying technical report[1]. Finally, we improve the trade-off lower bound from [7] to show that a stabilization time of $\Omega(D)$ is necessary for an algorithm with the $\mathcal{O}(d\log(D/d))$-gradient skew property. This shows that our second algorithm is optimal in its stabilization time as well as its skew guarantee.

2. RELATED WORK

The fundamental problem of synchronizing clocks in distributed systems has been studied extensively and many results have been published for various models over the course of the last approximately 30 years (see, e.g, [15, 17, 18, 19]). Until recently, the main focus has been on bounding the clock skew that may occur between any two nodes in the network, and a tight bound of $\Theta(D)$ has been proven [1, 3, 11, 19].

The problem of synchronizing clocks of nodes that are close-by as accurately as possible has been introduced by Fan and Lynch [5]. In their work, the authors show that a clock skew of $\Omega(\log D/\log\log D)$ between neighboring nodes cannot be avoided if the clock values must increase at a constant minimum progress rate. Subsequently, this result has been improved to $\Omega(\log D)$ [11]. If we take the minimum clock rate α, the maximum clock rate β, and the maximum clock drift rate ρ into account, the more general statement of the lower bound is that a clock skew of $\Omega(\log_b D)$, where $b := \min\{1/\rho, (\beta-\alpha)/(\alpha\rho)\}$ cannot be avoided. The first algorithm guaranteeing a sublinear bound on the worst-case clock skew between neighbors achieves a bound of $\mathcal{O}(\sqrt{\rho D})$ [12, 13]. Recently, this result has been improved to $\mathcal{O}(\log D)$ [10] (the base of the logarithm is a constant) and subsequently to $\mathcal{O}(\log_b D)$ [11]. Thus, tight bounds have been achieved for static networks in which neither nodes nor edges fail.

The problem of synchronizing clocks in the presence of faults has also received considerable attention (see, e.g., [2, 6, 9, 14, 16]). Some of the proposed algorithms are able to handle not only simple node or edges failures but also Byzantine behavior, which is outside the scope of this paper. However, while these algorithms can tolerate a broader range of failures, their network model is not fully dynamic as their results rely on the assumption that a large part of the network remains non-faulty and stable at all times. For the fully dynamic setting, it has been shown that there is an inherent trade-off between the clock skew $\mathcal{S}$ guaranteed between neighboring nodes that have been connected for a long time and the time it takes to guarantee a small clock skew over newly added edges. In particular, the time it takes to reduce the clock skew over new edges to $\mathcal{O}(\mathcal{S})$ is $\Omega(n/\mathcal{S})$, where n denotes the number of nodes in the network [7]. In the same work, it is shown that for $\mathcal{S} \in \Omega(\sqrt{\rho n})$, there is an algorithm that reduces the clock skew between any two nodes to $\mathcal{O}(\mathcal{S})$ in $\Theta(n/\mathcal{S})$ time. In this paper, we show that $\mathcal{S}$ can be reduced to $\Omega(\log_b n)$, i.e., the same optimal bound as for static networks can be achieved.

3. MODEL AND DEFINITIONS

Clock synchronization. In the clock synchronization problem, each node u is equipped with a continuous *hardware clock* $H_u : \mathbb{R}_0^+ \to \mathbb{R}_0^+$, which is initialized to $H_u(0) := 0$. The hardware clocks do not necessarily progress at the rate of real time; they are subject to a clock drift bounded by ρ. At all times t we assume that $\frac{d}{dt}H_u(t) \in [1-\rho, 1+\rho]$ for all nodes u.

The objective of a clock synchronization algorithm (CSA) is to output a *logical clock* $L_u : \mathbb{R}_0^+ \to \mathbb{R}_0^+$ (also initialized to $L_u(0) := 0$), such that at all times, the logical clock values of different nodes are close to each other. Logical clocks must also have a bounded drift: there must exist constants $\alpha, \beta > 0$ such that $\frac{d}{dt}L_u(t) \in [\alpha, \beta]$ for all times t and for all nodes u.

The estimate graph. In [8] two of the authors introduced an abstraction called *the estimate layer*, which simplifies reasoning about CSAs. Synchronization typically involves periodic exchanges of clock values between nodes, either through messages or by other means (e.g., RBS [4]). The estimate layer encapsulates all means by which nodes can estimate the clock values of other nodes, and eliminates the need to reason explicitly about delay bounds and other parameters of the system.

[1]http://disco.ethz.ch/publications/dynsynch.pdf

The estimate layer provides an *estimate graph*, where each edge $\{u,v\}$ represents the fact that node u has some means of estimating v's current clock value and vice-versa. The edges of the estimate graph are not necessarily direct communication links between nodes. Node u is provided with a *local estimate* $\tilde{L}_u^v$ of L_v, whose accuracy is guaranteed by the estimate layer:

$$\forall t\ \forall u \in V, v \in N_u(t) : |L_v(t) - \tilde{L}_u^v(t)| \leq \epsilon_{\{u,v\}}, \qquad (1)$$

where $\epsilon_{\{u,v\}}$ is called the *uncertainty*, or the *weight*, of the edge $\{u,v\}$. Each edge is also associated with a propagation delay $\mathcal{T}_{\{u,v\}}$ that bounds the time needed to send a message from u to v and vice versa, such that $\epsilon_{\{u,v\}} \geq (\beta - \alpha)\mathcal{T}_{\{u,v\}}$. This last assumption is reasonable because nodes typically need to communicate (although perhaps not directly) to estimate each others' clock values, and over the time required for communication, their logical clocks continue to drift apart. In the sequel, we refer to estimate edges of the sort described above as simply *edges*; similarly, when we say "the graph" we mean the estimate graph. We do not reason explicitly about the communication graph, as the salient aspects of communication are encapsulated by the estimate layer.

Dynamic networks. We consider dynamic networks over a fixed set of nodes V, where we denote $n := |V|$. Edge insertions and removals are modeled as discrete events controlled by a worst-case adversary. In keeping with the abstract representation from [8], we say that there is an *estimate edge* $\{u,v\}$ between two nodes $u, v \in V$ at time $t \geq 0$ iff u and v have a means of obtaining clock value estimates about each other at time t. As explained above, this does not necessarily mean that there is a direct communication link between u and v at time t.

We assume that nodes do not necessarily detect link formations and failures immediately, or even at the same time as the other endpoint of the link. For each edge $\{u,v\}$, we assume that there is a parameter $\tau_{\{u,v\}}$ such that both nodes u and v find out about the appearance or disappearance of edge $\{u,v\}$ within $\tau_{\{u,v\}}$ time units of the event itself. Hence, although we are interested in undirected networks, we model the network as a directed graph, where edge (u,v) corresponds to the fact that u thinks v is its neighbor. The appearance and disappearance of edges induces a dynamic graph $G = (V, E)$, where $E : \mathbb{R}_0^+ \to 2^{V \times V}$ maps non-negative times $t > 0$ to a set of directed estimate edges that exist at time t. The graph is subject to the following constraints, which approximate symmetry up to the delay ($\tau_{\{u,v\}}$) in finding out about link changes: (a) if for all $t' \in [t, t + 2\tau_{\{u,v\}}]$ we have $(u,v) \in E(t')$, then $(v,u) \in E(t + \tau_{\{u,v\}})$; (b) if for all $t' \in [t, t + 2\tau_{\{u,v\}}]$ we have $(u,v) \notin E(t')$, then $(v,u) \notin E(t + \tau_{\{u,v\}})$. Throughout each execution, every node u maintains a dynamic set of neighbors $N_u : \mathbb{R}_0^+ \to 2^V$, where $N_u(t)$ contains all nodes v such that $(u,v) \in E(t)$. In the sequel, we frequently refer to *undirected* edges $\{u,v\}$. When we write $\{u,v\} \in E(t)$ we mean that both $(u,v) \in E(t)$ and $(v,u) \in E(t)$.

To simplify the presentation, in Section 6 we assume that nodes find out about link changes immediately: that is, for all $u, v \in V$ we have $(u,v) \in E(t)$ iff $(v,u) \in E(t)$. This assumption is not made in the accompanying tech report.

We say that edge $\{u,v\}$ *exists throughout* an interval $[t_1, t_2]$ if for all $t \in [t_1, t_2]$ we have $\{u,v\} \in E(t)$. By extension, a path p is said to exist throughout $[t_1, t_2]$ if all its edges exist throughout the interval.

DEFINITION 1 (WEIGHTED PATHS). *Let $G = (V, E)$ be a dynamic graph with edge weights ϵ_e, $e \in E(t)$. A* path *$p = (u_0, \ldots, u_k)$ of length $k \geq 0$ in G at time t is a tuple of nodes such that for all $i \in \{1, \ldots, k\}$ it holds that $\{u_{i-1}, u_i\} \in E(t)$. The* weight *of p is $\epsilon_p := \sum_{i=1}^{k} \epsilon_{\{u_{i-1}, u_i\}}$. For simplicity, we will assume that the minimum edge weight is normalized to 1 throughout this extended abstract.*

We frequently refer to *the skew on a path* $p = (u_0, \ldots, u_k)$ at time t, by which we mean $|L_{u_0}(t) - L_{u_k}(t)|$.

Dynamic diameter. A fundamental lower bound from [1] shows that the performance of a CSA in a static network depends on the diameter of the network. In dynamic networks there is no immediate equivalent to a diameter. Informally, the diameter corresponds to the length of time it takes (at most) for information to spread from one end of the network to the other. To formalize this idea we adopt the following definitions.

DEFINITION 2 (FLOODING). *A* flood *originating at node u is a process initiated when node u sends a* Flood *message to all its neighbors. Each node that receives the message for the first time forwards it immediately to all its neighbors. We say that the flood is* complete *when all nodes have received a* Flood *message.*

DEFINITION 3 (DYNAMIC DIAMETER). *We say that dynamic graph G has a* dynamic diameter *of D (or simply "diameter" for short) if a flood originating at any node in the graph at any time in the execution always completes in at most D time units.*

Clock skew. To measure the quality of a CSA we consider two kinds of requirements: a *global skew constraint* which gives a bound on the difference between any two logical clock values in the system, and a *dynamic gradient skew constraint* which becomes stronger the closer two nodes u, v are to each other and the *longer* u, v stay close to each other. In particular, for nodes that remain neighbors for a long time, the dynamic gradient skew constraint requires a much smaller skew than the global skew constraint.

DEFINITION 4 (GLOBAL SKEW). *A CSA guarantees a* global skew *of $\bar{\mathcal{G}}$ if at all times t, for any two nodes $u, v \in V$, it holds that $L_u(t) - L_v(t) \leq \bar{\mathcal{G}}$.*

In contrast, the dynamic gradient skew constraint does depend on the dynamic graph: the older the shortest path between u and v, the better-synchronized u and v are required to be.

DEFINITION 5 (DYNAMIC GRADIENT SKEW). *Given a function $\mathcal{S} : \mathbb{R}_0^+ \times \mathbb{R}_0^+ \to \mathbb{R}_0^+$ that is non-decreasing in the first parameter (distance) and non-increasing in the second (time), we say that CSA $\mathcal{A}$ guarantees a* dynamic gradient skew *of $\mathcal{S}$ if for all time intervals $[t_1, t_2]$ and each path $p = (u_0, \ldots, u_k)$ that exists throughout the interval $[t_1, t_2]$, we have that*

$$L_{u_0}(t_2) - L_{u_k}(t_2) \leq \mathcal{S}(\epsilon_p, t_2 - t_1).$$

If the limit $\bar{\mathcal{S}}(d) := \lim_{\Delta t \to \infty} \mathcal{S}(d, \Delta t)$ is defined for all d, then we say that $\mathcal{A}$ guarantees a stable gradient skew *of $\bar{\mathcal{S}}$, where $\bar{\mathcal{S}} : \mathbb{R}_0^+ \to \mathbb{R}_0^+$. In this case we also say that $\mathcal{A}$ is* $\bar{\mathcal{S}}$-stabilizing.

If a CSA $\mathcal{A}$ guarantees a dynamic gradient skew of $\mathcal{S}$, then we call $\mathcal{A}$ an "$\mathcal{S}$-dynamic gradient CSA". The literature on gradient clock synchronization (e.g., [5, 7, 11, 13]) is typically concerned with the *local skew* of a CSA, which bounds the skew on any single edge. The local skew can be considered equivalent to the stable gradient skew $\bar{\mathcal{S}}(1)$, provided that all edges are of uniform weight 1.

We will further discuss the *stabilization time* of a CSA, which we define as follows.

Definition 6 (Stabilization Time). *Let $\mathcal{A}$ be a dynamic gradient CSA with a dynamic gradient skew of $\mathcal{S}$ and a stable gradient skew of $\bar{\mathcal{S}}$. The* stabilization time *of $\mathcal{A}$ is defined as*

$$\mathcal{T}_S := \inf\left\{\Delta t \mid \forall d \forall \Delta t' \geq \Delta t : \mathcal{S}(d, \Delta t') \leq 2\bar{\mathcal{S}}\right\}.$$

The dynamic and stable gradient skew and the stabilization time are parameterized by D, the diameter of the network, and potentially other parameters such as the bound on the clock drift ρ or the minimum edge weight. Usually we omit these depedencies to simplify the notation. Note that the choice of 2 as the constant in the definition above is arbitrary; we are interested in the asymptotic behavior of the clock skew as $D \to \infty$.

4. OVERVIEW OF THE OPTIMAL CSA IN STATIC NETWORKS

The algorithms in the current paper are based on the algorithm of [8, 11], which achieves the optimal gradient property for static networks. In [8, 11], each node of the network can be in one of two modes: in *slow mode*, the logical clock is increased at the rate of the node's hardware clock; in *fast mode*, the logical clock progresses at a faster rate. The rate of the logical clock in fast mode is $(1 + \mu)$ times the rate of the hardware clock, where $\mu > 0$ is a parameter of the algorithm. This ensures that logical clock rates are always bounded as required: they are between $1-\rho$ and $(1+\rho)(1+\mu)$ at all times.

The heart of the algorithm is the logic that controls which mode a node is in at any given time. This is specified in the form of two conditions: the *fast condition* (**FC**) tells nodes when to enter fast mode, and the *slow condition* (**SC**) tells nodes when to enter slow mode. To determine the appropriate mode, each node examines its estimate for the logical clock values of its neighbors. Both **FC** and **SC** are conditions on these estimates; informally, the fast condition **FC** checks if the node is too far behind its neighbors, in which case it enters fast mode; and the slow condition **SC** checks if the node is too far *ahead* of its neighbors, in which case it goes into slow mode.

The two conditions **FC** and **SC** are mutually exclusive, and furthermore they are strictly separated and defined as closed regions. This is necessary to ensure that (from a control-theoretic point of view) the algorithm can be implemented. If neither condition is satisfied, a node is free to choose non-deterministically between fast mode and slow mode (correctness is guaranteed for either choice).

The algorithm in [8] allows the use of edges e with different uncertainties ϵ_e. For every edge e, the algorithm uses a parameter κ_e which directly corresponds to ϵ_e and, roughly speaking, determines how much clock skew the nodes are willing to tolerate on the edge e. We call κ_e the *weight* of edge e. An edge $\{u, v\}$ with a large weight $\kappa_{\{u,v\}}$ corresponds to a neighbor v for which node u cannot obtain reliable estimates. Accordingly, node u treats its estimates of node v's clock value as less significant than estimates along edges with a smaller weight. One may think of each edge as a "leash" that pulls the clocks of its endpoint together. Edges with smaller weights correspond to shorter leashes, which require closer synchronization than edges with larger weights. For example, if $\kappa_{\{u,v_1\}} = 2\kappa_{\{u,v_2\}}$, and a skew of Δ is required on edge $\{u, v_1\}$ to cause u to enter fast mode or slow mode, then a skew of about 2Δ is required on $\{u, v_2\}$ to cause u to enter the same mode.

5. ALGORITHMS

In this section, we adapt the algorithm of [8, 11] for the dynamic model, and obtain an algorithm with an asymptotically optimal global skew of $\mathcal{O}(D)$ and stable local skew of $\mathcal{O}(\log_{\mu/\rho} D)$, where μ is the parameter governing logical clock rate in fast mode. We present two variants of the algorithm, based on the same technique. Both variants achieve optimal skew bounds: for any two nodes that remain at distance d from each other for sufficiently long, the algorithms guarantee a skew of at most $\mathcal{O}(d \log(D/d))$. The first algorithm we present is simpler and more elegant, but it does not achieve the optimal trade-off between skew and stabilization time; its stabilization time is $\mathcal{O}((D \log D)/\mu)$, which is off by an $\mathcal{O}(\log D)$ factor from the optimum. The second variant is more complex, and it achieves an optimal stabilization time of $\mathcal{O}(D\ \mu)$. To simplify the presentation, we focus here on the first algorithm, which we describe and analyze in detail. The second algorithm is presented in full in the accompanying technical report. We include here a brief overview.

5.1 A Dynamic-Weight Algorithm

The first of the two algorithms we present, $\mathcal{A}^{\mathrm{DW}}$, is a dynamic-weight algorithm. Instead of a fixed value κ_e for an edge e, algorithm $\mathcal{A}^{\mathrm{DW}}$ maintains a *dynamic weight* κ_e that starts with a large value for each newly formed edge e and is gradually decreased. The initial edge weight is chosen large enough such that an edge has no effect on its endpoints' modes immediately after insertion. Then, κ_e is decreased exponentially until it reaches its final value, which corresponds to the uncertainty ϵ_e of e (as in the static case). Hence, edges are initially treated as *unimportant*, and they continuously gain in importance while present.

Parameters and constants. Algorithm $\mathcal{A}^{\mathrm{DW}}$ involves several parameters which can be tuned to reduce the skew at the cost of longer stabilization time, or vice-versa. In addition we define several constants to simplify the presentation. The constants and parameters are described below.

$\bar{\mathcal{G}}$: an upper bound on the global skew of the algorithm. We assume that all nodes have access to this bound. It is sufficient for all nodes to know the number n of nodes in the network, and use n as a conservative estimate for the diameter D of the graph in the global skew bound established in Section 6.1.

μ: a parameter that determines logical clock rate in fast mode. To allow nodes to catch up when they are behind, μ must be sufficiently large; we require

$$\mu \geq \frac{16\rho}{1-\rho}. \tag{2}$$

On the other hand, nodes should not increase their clocks too quickly, or synchronization will be lost; we also require $\mu\epsilon_{\{u,v\}} \leq 12\bar{\mathcal{G}}$ for any possible edge $\{u,v\}$.

$\boldsymbol{\kappa^{\infty}_{\{u,v\}}}$: the "stable" weight of edge $\{u,v\}$. This is the weight eventually assigned to edge $\{u,v\}$ if it exists long enough in the system, and it corresponds to the uncertainty $\epsilon_{\{u,v\}}$. We require

$$\kappa^{\infty}_{\{u,v\}} > \frac{2}{\lambda} \cdot \left(1 + \frac{\mu}{6}\right) \cdot \epsilon_{\{u,v\}}. \tag{3}$$

$\boldsymbol{\lambda}$: a constant, chosen as $0 < \lambda < 1/4$, which determines the slack in the fast and slow conditions (see Definitions 8,9). The choice of λ does not affect the asymptotic behavior of the algorithm.

$\boldsymbol{\eta}$: a constant that controls the rate at which $\kappa_{\{u,v\}}$ is increased. It is defined as

$$\eta := \frac{\lambda(1-\rho)}{6} \cdot \mu. \tag{4}$$

Next we describe the components that make up the algorithm.

Dynamic weights. When edge $\{u,v\}$ appears at time t_0, nodes u, v establish and maintain a dynamic weight $\kappa_{\{u,v\}}(t)$ for the edge. Both u and v maintain a local estimate of the weight; we use $\kappa^v_u(t)$ to denote u's estimate and $\kappa^u_v(t)$ to denote v's estimate of $\kappa_{\{u,v\}}(t)$. The "true" weight induced by the two estimates is defined by $\kappa_{\{u,v\}}(t) := \min\{\kappa^v_u(t), \kappa^u_v(t)\}$.

Suppose that $u < v$ [2]. The node with the smaller identifier, in this case u, is responsible for setting the initial weight and decreasing it over time, while the other node, in this case v, periodically synchronizes its estimate to u's value. Node u (the "master node") changes $\kappa^v_u(t)$ over time according to the following dynamics, decreasing it exponentially from an initial value of $\bar{\mathcal{G}}$ to the final value of $\kappa^{\infty}_{\{u,v\}}$.

$$\begin{aligned} \kappa^v_u(t_0) &:= \kappa_{\{u,v\}}(t_0) = \bar{\mathcal{G}}, \\ \frac{d}{dt}\kappa^v_u(t) &:= \begin{cases} -\frac{\eta}{\bar{\mathcal{G}}} \cdot \frac{\frac{d}{dt}H_u(t)}{(1+\rho)} \cdot \kappa^v_u(t) & \kappa^v_u(t) > \kappa^{\infty}_{\{u,v\}}, \\ 0 & \kappa^v_u(t) = \kappa^{\infty}_{\{u,v\}}. \end{cases} \end{aligned} \tag{5}$$

Using the estimate layer, node v (the "slave node") conservatively synchronizes its estimate κ^u_v to u's value. We omit the technical details here; the following lemma characterizes the accuracy of v's estimates.

LEMMA 7. *For all $t \geq 0$ and nodes $u < v$, we have*

$$0 \leq \kappa^u_v(t) - \kappa^v_u(t) \leq \frac{\mu\epsilon_{\{u,v\}}}{6\bar{\mathcal{G}}} \cdot \kappa_{\{u,v\}}(t).$$

In particular, we always have $\kappa_{\{u,v\}}(t) = \kappa^v_u(t)$, that is, the master node's estimate is the true weight.

Max estimates. As in [7, 11, 12, 13], each node maintains a local estimate M_u of the maximum logical clock value in the network, and makes sure never to exceed it. As this is a standard technique we omit the implementation details, and note only that max estimates satisfy the following constraint: if the dynamic graph has a diameter of D, then for all $t \geq 0$ and for all nodes u we have

$$M_u(t) \leq \max_{v \in V}\{L_v(t)\}, \tag{6}$$

$$\forall t \geq 2D: \ M_u(t) > \max_{v \in V}\{L_v(t-2D)\}, \tag{7}$$

$$M_u(t) \geq L_u(t), \tag{8}$$

$$M_u(t) \geq \max_{v \in N_u(t)}\left\{\tilde{L}^v_u(t) - \epsilon_{\{u,v\}}\right\}, \tag{9}$$

That is, the max estimate of any node is never more than the true maximum, and it represents the true maximum from $2D$ time units ago, where D is the time required to complete a flood in the dynamic graph. In addition, (8) asserts that nodes cannot set their logical clock ahead of their max estimate, and (9) asserts that the max esimate always reflects the logical clock values of immediate neighbors. (The factor 2 arises from the fact that nodes do not constantly flood the network with max estimates. It can be reduced to $(1+\iota)$, where ι is an arbitrarily small constant, by starting floods sufficiently often.)

The fast and slow conditions. If node u finds that it is too far behind, it goes into fast mode and uses a fast rate of $(1+\mu)\frac{d}{dt}H_u(t)$ to increase its logical clock. The following rule is used to determine when to go into fast mode. Informally, it states that some neighbor is far ahead, and no neighbor is too far behind.

DEFINITION 8 (FAST CONDITION **FC**). *At time t, node $u \in V$ satisfies the* fast condition*, denoted* **FC***, if there is some integer $s \geq 1$ for which the following conditions are satisfied:*

$$\exists w \in N_u(t) : \tilde{L}^w_u(t) - L_u(t) \geq (s - \lambda)\kappa^w_u(t)$$
$$\forall v \in N_u(t) : L_u(t) - \tilde{L}^v_u(t) \leq (s + \lambda)\kappa^v_u(t).$$

Conversely, if a node is far behind some neighbor, and no other neighbor is too far ahead of it, it enters slow mode and uses the slow rate. The rule for entering slow mode is as follows.

DEFINITION 9 (SLOW CONDITION **SC**). *At time t, node $u \in V$ satisfies the* slow condition*, denoted* **SC***, if there is some integer $s \geq 0$ for which the following conditions are satisfied:*

$$\exists w \in N_u(t) : L_u(t) - \tilde{L}^w_u(t) \geq \left(s + \frac{1}{2} - \lambda\right)\kappa^w_u(t)$$
$$\forall v \in N_u(t) : \tilde{L}^v_u(t) - L_u(t) \leq \left(s + \frac{1}{2} + \lambda\right)\kappa^v_u(t).$$

Since a node cannot be in fast mode and in slow mode at the same time, **SC** and **FC** are required to be mutually exclusive, otherwise the algorithm would be impossible to implement.

LEMMA 10. *No node can satisfy* **FC** *and* **SC** *at the same time.*

This essentially follows from the fact that $\lambda < 1/4$, implying that $s + 1/2 - \lambda > s + \lambda$ and $s + 1/2 + \lambda < (s+1) - \lambda$ for all s. A formal proof of this statement is given in [8] and an analogous result for our second algorithm is shown in the technical report.

[2]We assume unique node identifiers, but other symmetry-breaking mechanisms may be substituted.

With the above definitions in place, $\mathcal{A}^{\mathrm{DW}}$ is given by the following rules, executed at each node u at all times t:

– If **FC** holds, or if **SC** does not hold and $L_u(t) < M_u(t)$, then node u must be in fast mode, setting $\frac{d}{dt}L_u = (1 + \mu)\frac{d}{dt}H_u$.

– If **SC** holds, or if **FC** does not hold and $L_u(t) = M_u(t)$, then node u must be in slow mode, setting $\frac{d}{dt}L_u = \frac{d}{dt}H_u$.

If neither of the conditions above holds, the node is free to choose its mode nondeterministically.

5.2 Overview of Algorithm $\mathcal{A}^{\mathrm{OPT}}$

The second variant of the algorithm uses exactly the same definition of fast mode and slow mode. In particular, it also guarantees that all logical clock rates are always in the range $[1 - \rho, (1 + \mu)(1 + \rho)]$. The main difference is that the algorithm uses a different mechanism to ensure that new edges cannot immediately determine the current mode. Instead of a dynamic weight κ_e, each node u uses several *neighborhood sets* $N_u^1, N_u^2, \ldots$ (in addition to N_u), for which it holds at all times that $N_u \supseteq N_u^1 \supseteq N_u^2$ etc. Both **FC** and **SC** are changed in that they now apply when there is some integer s for which the **FC**/**SC** preconditions restricted to $N_u^s(t)$ are satisfied. For example, the first condition of **FC** changes to "when there is a neighbor $w \in N_u^s(t)$ such that $\tilde{L}_u^w(t) - L_u(t) \geq (s - \lambda)\kappa_u^w(t)$." If the neighborhood sets are chosen and updated appropriately, these modified conditions also guarantee that new edges cannot cause a violation of the desired skew bounds.

The crucial question is when and how the neighborhood sets are updated. We will now briefly outline this procedure. The algorithm operates in loosely synchronized rounds, i.e., a round always begins when the logical clock reaches a certain value (which may occur at different times at different nodes). The edges that are incorporated in any round are all the new edges that are present at the beginning of the round, i.e., if new neighbors appear during the course of a round, they are ignored until the next round starts. In each round, the sets $N_u^1, N_u^2, \ldots$, are updated exactly once and in this order at times $T_1 < T_2 < \ldots$ etc. Updating a set N_u^s simply means setting it to N_u^{s-1}. The set N_u^1 is updated using the neighborhood N_u of u at time T_1. The update times occur in intervals that ensure that the targeted skew bounds are never violated. In the technical report, we prove that these times can be defined in such a way that the duration of a round is $\mathcal{O}(D/\mu)$ and that all new edges introduced at the beginning of a round have stabilized by its end.

6. ANALYSIS

In this section we sketch the analysis of $\mathcal{A}^{\mathrm{DW}}$, the simpler of the two algorithms, and bound its worst-case global and dynamic gradient skew. To simplify further, we assume here that when an edge appears or disappears, both endpoints find out about this event immediately. Hence, in this section, the following four statements are equivalent: I. $(u, v) \in E(t)$, II. $(v, u) \in E(t)$, III. $u \in N_v(t)$, and IV. $v \in N_u(t)$. Consequently, we refer only to undirected edges $\{u, v\}$ throughout the section. In the accompanying technical report, algorithm $\mathcal{A}^{\mathrm{OPT}}$ is analyzed without this assumption.

6.1 The Global Skew

Like its predecessors in [8, 11], algorithm $\mathcal{A}^{\mathrm{DW}}$ achieves an asymptotically-optimal global skew of $\mathcal{O}(D)$, where D is now defined as the dynamic diameter of the graph (see Section 3).

THEOREM 11. *Algorithm $\mathcal{A}^{\mathrm{DW}}$ achieves a global skew of $2D(1 + \rho) \in \mathcal{O}(D)$ in networks of diameter D.*

PROOF. The proof is similar to the ones in [7, 11], and depends solely on the behavior of the nodes with the largest and smallest clock values in the network. We argue that these nodes act to reduce the global skew: the node with the largest clock is always in slow mode, and if the skew is large, then the node with the smallest clock is in fast mode, trying to catch up.

First, consider the node u with the largest clock in the network: $L_u(t) = \max_{v \in V}\{L_v(t)\}$. From (6) and (8) we have $M_u(t) = L_u(t)$, that is, node u *knows* that it has the largest clock. In addition, by (3) and (9) we have $M_u(t) \geq \max_{v \in N_u(t)}\{L_v(t) - \epsilon_{\{u,v\}}\}$ and for any neighbor v, $(1 - \lambda)\kappa_u^v(t) \geq (1 - \lambda)\kappa_{\{u,v\}}^{\infty} > \epsilon_{\{u,v\}}$. Hence, u's estimates of its neighbors' clock values cannot be large enough for **FC** to hold. Together with the fact that $L_u(t) = M_u(t)$, this forces u to be in slow mode.

For this reason, the maximum clock value in the network increases at most at rate $1 + \rho$, the maximum hardware clock rate. From (7) it follows that for any node v we have $M_v(t) > \max_{w \in V}\{L_w(t)\} - 2D(1 + \rho)$; that is, the max estimates of all nodes are "not too far off" the true maximum.

Now consider a node u with the *smallest* clock in the network, $L_u(t) = \min_{v \in V}\{L_v(t)\}$, and suppose that $L_u(t) = \max_{v \in V}\{L_v(t)\} - 2D(1 + \rho)$, in other words, that there is a large gap between the smallest and largest clocks. Because $M_v(t) < \max_{w \in V}\{L_w(t)\} - 2D(1 + \rho)$, we immediately obtain $L_u(t) < M_u(t)$, so node u knows it is behind. In addition, since node u has the smallest clock, for each $v \in N_u(t)$ we have $L_v(t) \geq L_u(t)$, and hence $\tilde{L}_u^v(t) \geq L_u(t) - \epsilon_{\{u,v\}} > L_u(t) - (1/2 - \lambda)\kappa_u^v(t)$, which means that **SC** does not hold (all of u's neighbors are too far ahead). Together with the fact that $L_u(t) < M_u(t)$, this forces node u to be in fast mode.

We have shown that whenever there is a large enough skew such that $\min_{v \in V}\{L_v(t)\} = \max_{v \in V}\{L_v(t)\} - 2D(1+\rho)$, all nodes with the smallest logical clock value, $\min_{v \in V}\{L_v(t)\}$, will be in fast mode, and all nodes with the largest logical clock value will be in slow mode. A node in fast mode increases its logical clock at a rate of at least $(1 + \mu)(1 - \rho)$, and a node in slow mode increases its logical clock at a rate of at most $(1 + \rho)$. By (2), $(1 + \mu)(1 - \rho) > 1 + \rho$, so the nodes that are the most behind cannot fall behind any further. The continuity of logical clocks thus ensures that the global skew never exceeds $2D(1 + \rho)$. $\square$

As explained in Section 5.1, we assume that each node maintains an upper bound $\bar{\mathcal{G}}$ on the global skew of the network. This can be done dynamically, by running an estimation protocol alongside the clock synchronization algorithm, or the bound $\bar{\mathcal{G}}$ can be computed statically based on known properties of the network. For simplicity, we use a single global skew estimate $\bar{\mathcal{G}}$ throughout the paper. All algorithms and proofs can however be adapted to a scenario where each node maintains an individual and possibly dynamic upper bound on the global skew.

6.2 Analysis of the Gradient Skew

In this section we prove the gradient skew property of Algorithm $\mathcal{A}^{\text{DW}}$:

THEOREM 12. *$\mathcal{A}^{\text{DW}}$ has a stable gradient skew of $\bar{\mathcal{S}}(d) \in \mathcal{O}(d\log(D/d))$, with a stabilization time of $\mathcal{O}(D\log D)$.*

We start by analyzing the properties of the dynamic weights $\kappa_{\{u,v\}}(t)$. For a path $p = (u_0, \ldots, u_k)$ that exists at time t, we define $\kappa_p(t) := \sum_{i=1}^k \kappa_{\{u_{i-1},u_i\}}(t)$ to be the total weight of the path at time t. In addition, for any path p that exists throughout an interval $[t_0, t_1]$, let $\Delta\kappa_p(t_0,t_1) := \kappa_p(t_1) - \kappa_p(t_0)$ be the change in the path's weight from time t_0 to time t_1. The following technical lemma asserts that over sufficiently short intervals $[t_0, t_1]$, the change in a path's weight is bounded as a function of its weight at time t_1.

LEMMA 13. *For any path p, integer $s \geq 1$, and times $t_1 \geq t_0$ such that $t_1 - t_0 \leq \frac{\bar{\mathcal{G}}/2^{s-1}}{(1-\rho)\mu}$, we have*

$$0 \geq \Delta\kappa_p(t_0,t_1) \geq -\kappa_p(t_1)\cdot\frac{\lambda/3}{s+1/2}.$$

PROOF. First, observe that $\kappa_p(t)$ is positive and non-increasing (along paths that do not disappear). From (5), we have

$$\begin{aligned}\Delta\kappa_p(t_0,t_1) &\geq -\kappa_p(t_0)\cdot\frac{\eta}{\bar{\mathcal{G}}}(t_1-t_0) = -\kappa_p(t_0)\cdot\frac{\lambda}{3\cdot 2^s}\\ &\geq -\kappa_p(t_0)\cdot\frac{\lambda/3}{s+1}.\end{aligned}$$

This implies that $\kappa_p(t_1)/\kappa_p(t_0) \geq 1 - \frac{\lambda/3}{s+1} \geq \frac{s-1/2}{s+1}$, and thus the claim follows. $\square$

The following definition captures the formal requirement on the skew along paths of different weight.

DEFINITION 14 (LEGAL STATE). *We say that the network is in a* legal state *at time t if and only if for all integers $s \geq 1$ and all paths $p = (v_0,\ldots,v_k)$ with $\kappa_p(t) \geq C_s := \bar{\mathcal{G}}/2^{s-1}$ that exist at time t it holds that*

$$L_{v_k}(t) - L_{v_0}(t) \leq s\cdot\kappa_p(t).$$

We will show that the legal state condition is an invariant maintained throughout any execution of the algorithm, which implies Theorem 12.

In the analysis we work with two notions of "weighted skew", capturing how much a node v_0 is ahead or behind any other node, respectively. Both notions of weighted skew are essentially the difference between the clocks of the nodes at the endpoints of a path, normalized by the total weight of the path. However, we use different constants for each, corresponding to the constants used in **FC** and **SC**.

DEFINITION 15. *Given an integer $s \geq 1$, a time t, and a path $p = (v_0,\ldots,v_k)$, we define*

$$\Xi_p^s(t) := L_{v_0}(t) - L_{v_k}(t) - s\cdot\kappa_p(t), \quad \textit{and}$$
$$\Xi_{v_0}^s(t) := \max_{p=(v_0,\ldots)} \Xi_p^s(t).$$

DEFINITION 16. *Given an integer $s \geq 1$, a time t, and a path $p = (v_0,\ldots,v_k)$, we define*

$$\Psi_p^s(t) := L_{v_k}(t) - L_{v_0}(t) - \left(s+\frac{1}{2}\right)\cdot\kappa_p(t), \quad \textit{and}$$
$$\Psi_{v_0}^s(t) := \max_{p=(v_0,\ldots)} \Psi_p^s(t).$$

Given a path $p = (v_0,\ldots,v_k)$, we use $p^{-1} = (v_k,\ldots,v_0)$ to denote the inverted path. Note that

$$\Psi_p^s(t) = \Xi_{p^{-1}}^s(t) - \frac{\kappa_p(t)}{2}. \tag{10}$$

In particular, if node v_0 is "far behind" node v_k, as reflected by a large value of $\Psi_p^s(t)$, then node v_k is "far ahead" of node v_0, which is reflected by a large value of $\Xi_{p^{-1}}^s(t)$.

We use the following abbreviation for the increment of a logical clock value:

$$I_v(t,t') := L_v(t') - L_v(t). \tag{11}$$

The fast and slow conditions are "subjective" conditions that tell a node when to enter fast or slow mode. The following two lemmas are an "objective" statement which characterizes, from the point of view of an external observer who can see the entire state of the system, when a specific node must be in fast or slow mode.

LEMMA 17. *If there is an integer $s \geq 1$ such that the following two conditions are satisfied, a node u is in fast mode:*

$$\exists w \in N_u(t) : L_w(t) - L_u(t) \geq \left(s - \frac{\lambda}{2}\right)\kappa_{\{u,w\}}(t)$$
$$\forall v \in N_u(t) : L_u(t) - L_v(t) \leq \left(s + \frac{\lambda}{2}\right)\kappa_{\{u,v\}}(t).$$

LEMMA 18. *If there is an integer $s \geq 0$ such that the following two conditions are satisfied, a node u is in slow mode:*

$$\exists w \in N_u(t) : L_u(t) - L_w(t) \geq \left(s + \frac{1-\lambda}{2}\right)\kappa_{\{u,w\}}(t)$$
$$\forall v \in N_u(t) : L_v(t) - L_u(t) \leq \left(s + \frac{1+\lambda}{2}\right)\kappa_{\{u,v\}}(t).$$

The proofs are technical and they are omitted here.

The next two lemmas prove important properties regarding the functions $\Xi_v^s(t)$ and $\Psi_v^s(t)$.

LEMMA 19. *If for a node $u \in V$, an integer $s \in \mathbb{N}$ and a time t, $\Xi_u^s(t) > 0$, then $\frac{d}{dt}\Xi_u^s(t) \leq \frac{d}{dt}L_u(t) - (1-\rho)(1+\mu) + \eta$.*

PROOF SKETCH. Set $u_0 := u$ and let $p = (u_0,\ldots,u_k)$ be a path maximizing $\Xi_u^s(t)$. Since $\Xi_u^s(t)$ is positive, we know that $u_0 \neq u_k$, i.e., the path is non-empty. We have that

$$L_{u_{k-1}}(t) - L_{u_k}(t) \geq s\kappa_{\{u_{k-1},u_k\}}(t),$$

since otherwise $\Xi_{(u_0,\ldots,u_{k-1})}^s(t)$ would be larger than $\Xi_p^s(t)$; also, for all $v \in N_{u_k}(t)$

$$L_{u_k}(t) - L_v(t) \leq s\kappa_{\{u_k,v\}}(t),$$

since otherwise $\Xi_{(u_0,\ldots,u_k,v)}^s(t)$ would be larger than $\Xi_p^s(t)$. Hence, Lemma 17 states that node u_k is in fast mode, which yields that $\frac{d}{dt}L_{u_k}(t) = \frac{d}{dt}H_{u_k}(t)(1+\mu) \geq (1-\rho)(1+\mu)$. Further, by the definition of $\kappa_{\{u,v\}}(t)$ in (5), we have that $s\cdot d\kappa_p(t)/dt \geq -s\cdot\eta\kappa_p(t)/\bar{\mathcal{G}} \geq -\eta$. We conclude that

$$\begin{aligned}\frac{d}{dt}\Xi_p^s(t) &= \frac{d}{dt}L_u(t) - \frac{d}{dt}L_{u_k}(t) - s\cdot\frac{d}{dt}\kappa_p(t)\\ &\leq \frac{d}{dt}L_u(t) - (1-\rho)(1+\mu) + \eta.\end{aligned}$$

Since this holds for all paths p that maximize $\Xi_u^s(t)$, the statement follows. $\square$

LEMMA 20. *For all nodes $v \in V$, integers $s \in \mathbb{N}$, and times t, it holds that $\Xi_v^s(t) < C_s$.*

PROOF. Analogous to [8, 11]. $\square$

DEFINITION 21. *Given a node u and times $t > t_0 \geq 0$, we define* the swing time *of u relative to t_0 and t by*

$$\mathrm{sw}_u(t_0, t) := \min\left\{ t' \in [t_0, t] \;\middle|\; \begin{array}{l} L_u(t) - L_u(t') \\ \leq (1+\rho)\cdot(t - t') \end{array} \right\}. \quad (12)$$

By definition, $\mathrm{sw}_u(t_0, t)$ is the earliest time relative to t such that between $\mathrm{sw}_u(t_0, t)$ and t node u's clock increased by no more than its maximal natural rate. Hence, the rate at time $\mathrm{sw}_u(t_0, t)$ is greater than $1 + \rho$ and thus u is in fast mode at time $\mathrm{sw}_u(t_0, t)$ provided that $\mathrm{sw}_u(t_0, t) > t_0$.[3] We show that in this case, one of the following two conditions must hold at time $\mathrm{sw}_u(t_0, t)$.

(SW-$\exists$) There exist a neighbor $w \in N_u(\mathrm{sw}_u(t_0, t))$ and an integer $s \geq 0$ such that

$$L_w(\mathrm{sw}_u(t_0,t)) - L_u(\mathrm{sw}_u(t_0,t)) > \left(s + \frac{1+\lambda}{2}\right)\kappa_{\{u,v\}}(\mathrm{sw}_u(t_0,t)).$$

(SW-$\forall$) For all neighbors $w \in N_u(\mathrm{sw}_u(t_0, t))$ and integers $s \geq 0$ we have

$$L_u(\mathrm{sw}_u(t_0,t)) - L_w(\mathrm{sw}_u(t_0,t)) < \left(s + \frac{1-\lambda}{2}\right)\kappa_{\{u,v\}}(\mathrm{sw}_u(t_0,t)).$$

LEMMA 22. *For all nodes u and times t, if $\mathrm{sw}_u(t_0, t) > t_0$, then either (SW-$\exists$) or (SW-$\forall$) holds.*

The following theorem proves the gradient skew property of $\mathcal{A}^{\mathrm{DW}}$. Due to lack of space, we include only an overview of the proof. For the full proof, we refer to the technical report.

THEOREM 23. *The system is always in a legal state.*

PROOF. Assume by way of contradiction that the legal state condition is violated, and let $\bar{t}$ be the infimum of times for which it is violated. In this case there exists an integer $s \geq 1$ and a path $q = (w, \ldots, w')$ such that $\kappa_q(\bar{t}) \geq C_{s+1}$ and

$$L_w(\bar{t}) - L_{w'}(\bar{t}) = (s+1)\cdot\kappa_q(\bar{t}). \quad (13)$$

Note that by the global skew constraint, the legal state is always satisfied for paths q with $\kappa_q(t) \geq C_1$. We can therefore assume that $s \geq 1$. Let $p = (u_0, \ldots, u_k)$ be a path for which $\Psi_{p^{-1}}^s(\bar{t}) = \Psi^s(\bar{t})$. We have

$$\Psi_{p^{-1}}^s(\bar{t}) = \Psi^s(\bar{t}) \geq \Psi_{(w',\ldots,w)}^s(\bar{t}) = \Xi_{(w,\ldots,w')}^s(\bar{t}) - \frac{\kappa_q(\bar{t})}{2} \overset{(13)}{=} \frac{\kappa_q(\bar{t})}{2} \geq \frac{C_{s+1}}{2}. \quad (14)$$

From legality (which still holds at time $\bar{t}$) and (13), we obtain $\kappa_p(\bar{t}) < C_s$. Define

$$\underline{t} := \bar{t} - \frac{C_s}{(1-\rho)\mu}.$$

[3]More precisely, there is a time $t_0 < t' < \mathrm{sw}_u(t_0, t)$ for which it holds that the amortized rate of u is larger than $1+\rho$ over any time interval $[t'', \mathrm{sw}_u(t_0, t)]$ where $t'' \in (t', \mathrm{sw}_u(t_0, t))$.

Roughly speaking, our goal is to show that there is always some node that pulls node u_k ahead, acting to reduce Ξ. For this to happen we need Ξ to remain large as we go back in time, allowing us to use Lemma 19. Furthermore, in order to show an actual decrease of Ξ, we need the node that is ahead to remain in slow mode while u_k (or whichever node is behind the most) catches up. We cannot guarantee that the foremost nodes will indeed remain in slow mode, but we know that if node v enters fast mode, there is a reason: node v believes itself free to move at the fast rate, therefore either some neighbor w is far ahead (SW-$\exists$), or no neighbor is very far behind (SW-$\forall$).

In the first case (SW-$\exists$), we can switch to reasoning about the path that extends to w, and doing so only increases the weighted skew of the path (because w has a large weighted skew relative to v). We refer to such a switch as a *forward switch.* However, the second case (SW-$\forall$) is problematic: in this case we actually lose weighted skew, because we have to switch to a node that is behind v, and the weighted skew of the path decreases. We call this a *backwards switch.* In order to retain positive weighted skew we must bound the number of backwards switches. This is accomplished by only making backwards switches along the path from u_0 to u_k, so that eventually we "run out of nodes". We show that eventually we can stop making backwards switches.

I. Backwards switches, tracing p inwards towards u_k.

We define a sequence of non-increasing times $t_0 = \bar{t} \geq t_1 \geq \ldots \geq t_\ell$, where $\ell < k$. The construction is inductive, with $t_0 = \bar{t}$ and for all $1 \leq i \leq \ell$, $t_{i+1} := \mathrm{sw}_{u_i}(\underline{t}, t_i)$. The construction stops at the minimal index ℓ such that either $t_{\ell+1} = \underline{t}$ or (SW-$\exists$) is satisfied for u_ℓ at time $t_{\ell+1}$.

First, observe that since $t_{i+1} = \mathrm{sw}_{u_i}(\underline{t}, t_i)$, for all $0 \leq i \leq k-1$ and times $t \in [t_{i+1}, t_i]$ we have

$$\begin{aligned}\Xi_{u_i,\ldots,u_k}^s(t) = {} & \Xi_{u_i,\ldots,u_k}^s(t_i) - I_{u_i}(t, t_i) + I_{u_k}(t, t_i) \\ & + s\Delta\kappa_{(u_0,\ldots,u_i)}(t, t_i) \\ \overset{(12)}{\geq} {} & \Xi_{u_i,\ldots,u_k}^s(t_i) - (1+\rho)(t_i - t) + I_{u_k}(t, t_i) \\ & + s\Delta\kappa_{(u_0,\ldots,u_i)}(t, t_i). \end{aligned} \quad (15)$$

Next, we show by induction on i that for all $i \in \{0, \ldots, \ell\}$,

$$\begin{aligned}\Xi_{u_i,\ldots,u_k}^s(t_i) \geq {} & \Xi_p^s(t_0) - \frac{\kappa_{u_0,\ldots,u_i}(t_0)}{2} - (1+\rho)(t_0 - t_i) \\ & + I_{u_k}(t_i, t_0) + s\Delta\kappa_{(u_0,\ldots,u_i)}(t_i, t_0). \end{aligned} \quad (16)$$

The induction is omitted here for lack of space. Informally, this bounds the weighted skew from below as we trace p inwards towards node u_k, showing that plenty of weighted skew remains. This will be used to derive a contradiction to the fact that the system is legal at time $\bar{t}$.

Induction base. For $i = 0$ the claim is trivial, since $(u_0, \ldots, u_k) = p$.

Induction step. Suppose that the claim holds for i and consider some time $t \in [t_{i+1}, t_i]$. We have

$$\begin{aligned}\Xi_{u_i,\ldots,u_k}^s(t) \overset{(15)}{\geq} {} & \Xi_{u_i,\ldots,u_k}^s(t_i) - (1+\rho)(t_i - t) + I_{u_k}(t, t_i) \\ & + s\Delta\kappa_{(u_0,\ldots,u_i)}(t, t_i) \\ \overset{(16)}{\geq} {} & \Xi_p^s(t_0) - \frac{\kappa_{u_0,\ldots,u_i}(t_0)}{2} - (1+\rho)(t_0 - t) \\ & + I_{u_k}(t, t_0) + s\Delta\kappa_{(u_0,\ldots,u_i)}(t, t_0). \end{aligned} \quad (17)$$

This holds for all $t \in [t_{i+1}, t_i]$; now let us show that we do not "lose" too much weighted skew when we switch from u_i to u_{i+1} at time t_{i+1} (and shorten the path to $u_{i+1},\ldots,u_k$). By definition, $t_{i+1}=\mathrm{sw}_{u_i}(\underline{t},t_i)$. From Lemma 22, either (SW-$\forall$) or (SW-$\exists$) is satisfied for u_i at time t_{i+1}; however, in the latter case the construction halts. Therefore (SW-$\forall$) holds for u_i at time t_{i+1}. In particular, since $u_{i+1}\in N_{u_i}(t_{i+1})$,

$$\begin{aligned}L_{u_i}(t_{i+1})-L_{u_{i+1}}(t_{i+1}) &\overset{\text{(SW-}\forall)}{<} \left(s+\frac{1-\lambda}{2}\right)\kappa_{\{u_i,u_{i+1}\}}(t_{i+1})\\ &\overset{(\lambda>0)}{<} \left(s+\frac{1}{2}\right)\kappa_{\{u_i,u_{i+1}\}}(t_{i+1}).\end{aligned}\tag{18}$$

Using (18) and (17) we obtain

$$\begin{aligned}\Xi^s_{u_{i+1},\ldots,u_k}(t_{i+1}) &\geq \Xi^s_p(t_0)-\frac{\kappa_{u_0,\ldots,u_{i+1}}(t_0)}{2}+I_{u_k}(t_{i+1},t_0)\\ &\quad-(1+\rho)(t_0-t_{i+1})+s\Delta\kappa_{(u_0,\ldots,u_{i+1})}(t_{i+1},t_0).\end{aligned}$$

This completes the induction.

II. Forward switches until time $\underline{t}$. Having reached a node where we can make a forward switch for the first time, we show that we can continue to make forward switches as we go back in time until we reach $\underline{t}$. We construct a chain $v_\ell,\ldots,v_{\ell+m}$, where $v_\ell=u_\ell$, and a sequence of times $t_\ell,\ldots,t_{\ell+m}=\underline{t}$ (where t_ℓ is the time we reached in the previous part). The times t_i are defined as before, for all $\ell\leq i\leq \ell+m-1$, we define

$$t_{i+1}:=\mathrm{sw}_{v_i}(\underline{t},t_i).\tag{19}$$

The construction is inductive.

Assume that we have reached node v_i at time $t_{i+1}=\mathrm{sw}_{v_i}(\underline{t},t_i)$. To obtain v_{i+1}, we show that we can make a forward switch from v_i, that is, that (SW-$\exists$) is satisfied at v_i at time t_{i+1}. For $i=\ell$ the claim follows from the halting condition of Part I; thus, suppose that $i>\ell$ and that (SW-$\exists$) is not satisfied. Then from Lemma 22, (SW-$\forall$) is satisfied at time t_{i+1} for v_i. In particular, for $v_{i-1}\in N_{v_i}(t_{i+1})$ we then have

$$L_{v_i}(t_{i+1})-L_{v_{i-1}}(t_{i+1})<\left(s+\frac{1-\lambda}{2}\right)\kappa_{\{v_i,v_{i-1}\}}(t_{i+1}).\tag{20}$$

Since v_i was reached by a forward switch from v_{i-1} at time t_i, we also have

$$L_{v_i}(t_i)-L_{v_{i-1}}(t_i)>\left(s+\frac{1+\lambda}{2}\right)\kappa_{\{v_i,v_{i-1}\}}(t_i).\tag{21}$$

Combining (20) and (21) yields

$$\begin{aligned}I_{v_{i-1}}(t_{i+1},t_i) &< I_{v_i}(t_{i+1},t_i)-\lambda\kappa_{\{v_i,v_{i+1}\}}(t_i)\\ &\quad-\left(s+\frac{1-\lambda}{2}\right)\Delta\kappa_{\{v_i,v_{i+1}\}}(t_{i+1},t_i)\\ &\overset{\text{(Lem. 13)}}{\leq} I_{v_i}(t_{i+1},t_i).\end{aligned}\tag{22}$$

We thus obtain

$$\begin{aligned}I_{v_{i-1}}(t_{i+1},t_{i-1}) &= I_{v_{i-1}}(t_{i+1},t_i)+I_{v_{i-1}}(t_i,t_{i-1})\\ &\overset{(22)}{<} I_{v_i}(t_{i+1},t_i)+I_{v_{i-1}}(t_i,t_{i-1})\\ &\overset{(12)}{\leq} (1+\rho)(t_{i-1}-t_{i+1}).\end{aligned}$$

This is a contradiction to the choice of $t_i=\mathrm{sw}(t_{i-1})$. Thus, (SW-$\exists$) must be satisfied at node v_i at time t_{i+1}. Therefore, we can choose a node $v_{i+1}\in N_{v_i}$ for which

$$L_{v_{i+1}}(t_{i+1})-L_{v_i}(t_{i+1})>\left(s+\frac{1+\lambda}{2}\right)\kappa_{\{v_i,v_{i+1}\}}(t_{i+1}).\tag{23}$$

In the following, we show by induction on i that for all $i\in\{\ell,\ldots,\ell+m-1\}$,

$$\begin{aligned}\Xi^s_{v_i}(t_{i+1}) &\geq \Xi^s_{v_\ell}(t_{\ell+1})+((1-\rho)\mu-2\rho-\eta)(t_{\ell+1}-t_{i+1})\\ &> 0.\end{aligned}\tag{24}$$

Induction base. The base case follows from applying the bound obtained in (14) to (16), with $i=\ell$ and $t=t_{\ell+1}$. We omit the details.

Induction step. We first extend the path by adding node v_{i+1}. Let $p=(v_i,\ldots,v)$ be a path maximizing $\Xi^s_{v_i}(t_{i+1})$. Note that since p exists at time t_{i+1} and $v_{i+1}\in N_{v_i}(t_{i+1})$, the path $(v_{i+1},v_i,\ldots,v)$ exists at time t_{i+1} as well. We have

$$\begin{aligned}\Xi^s_{v_{i+1}}(t_{i+1}) &\geq \Xi^s_{(v_{i+1},v_i,\ldots,v)}(t_{i+1})\\ &= \Xi^s_{(v_i,\ldots,v)}(t_{i+1})+L_{v_{i+1}}(t_{i+1})-L_{v_i}(t_{i+1})\\ &\quad- s\kappa_{\{v_{i+1},v_i\}}(t_{i+1}) \qquad (25)\\ &\overset{(23)}{>} \Xi^s_{(v_i,\ldots,v)}(t_{i+1}) \;=\; \Xi^s_{v_i}(t_{i+1})\\ &\overset{\text{(I.H.)}}{\geq} \Xi^s_{v_\ell}(t_{\ell+1})+((1-\rho)\mu-2\rho-\eta)(t_{\ell+1}-t_{i+1})\\ &> 0. \qquad (26)\end{aligned}$$

We now show that $\Xi^s_{v_{i+1}}(t)>0$ for all times $t\in[t_{i+2},t_{i+1}]$. For the sake of contradiction, assume that there is a $t\in[t_{i+2},t_{i+1}]$ for which $\Xi^s_{v_{i+1}}(t)\leq 0$ and assume that t is the largest such time. We then have $\Xi^s_{v_{i+1}}(\tau)>0$ for all $\tau\in(t,t_{i+1}]$ and therefore

$$\begin{aligned}0 &\overset{(26)}{<} \Xi^s_{v_{i+1}}(t_{i+1})=\Xi^s_{v_{i+1}}(t)+\int_t^{t_{i+1}}\frac{d}{dt}\Xi^s_{v_{i+1}}(\tau)d\tau\\ &\overset{\text{(Lem. 19)}}{\leq} L_{v_{i+1}}(t_{i+1})-((1-\rho)(1+\mu)-\eta)(t_{i+1}-t)\\ &\qquad -L_{v_{i+1}}(t)\\ &\overset{(19)}{\leq} -((1-\rho)\mu-2\rho-\eta)(t_{i+1}-t) \;<\; 0.\end{aligned}$$

We conclude that $\Xi^s_{v_{i+1}}(t)>0$ for all $t\in[t_{i+2},t_{i+1}]$, which allows us to apply Lemma 19 to lower bound $\Xi^s_{v_{i+1}}(t_{i+2})$:

$$\begin{aligned}\Xi^s_{v_{i+1}}(t_{i+2}) &\geq \Xi^s_{v_{i+1}}(t_{t+1})+((1-\rho)\mu-2\rho-\eta)(t_{i+1}-t_{i+2})\\ &\overset{(26)}{\geq} \Xi^s_{v_\ell}(t_{\ell+1})+((1-\rho)\mu-2\rho-\eta)(t_{\ell+1}-t_{i+2}).\end{aligned}$$

III. Putting everything together. Using Inequality (16) for $i=\ell$ and $t=t_{\ell+1}$, we obtain

$$\begin{aligned}\Xi^s_{u_\ell}(\bar{t}) &\geq \Xi^s_{(u_\ell,\ldots,v_k)}(\bar{t})\\ &= \Xi^s_{u_\ell}(t_{\ell+1})+I_{u_\ell}(t_{\ell+1},\bar{t})-I_{u_k}(t_{\ell+1},\bar{t})\\ &\quad- s\Delta\kappa_{(u_\ell,\ldots,u_k)}(t_{\ell+1},\bar{t}) \qquad (27)\\ &\overset{(16),(10)}{\geq} \Psi^s_{p^{-1}}(\bar{t})-(1+\rho)(\bar{t}-t_{\ell+1})+I_{u_\ell}(t_{\ell+1},\bar{t})\\ &\quad+ s\Delta\kappa_{(u_0,\ldots,u_\ell)}(\underline{t},\bar{t}). \qquad (28)\end{aligned}$$

Applying (24) for $i = \ell + m$ (note that $u_\ell = v_\ell$) yields

$$\begin{aligned}
\Xi^s_{v_{\ell+m}}(\underline{t}) &\overset{(24)}{\geq} \Xi^s_{u_\ell}(t_{\ell+1}) + ((1-\rho)\mu - 2\rho - \eta)(t_{\ell+1} - \underline{t}) \\
&\overset{\text{(Lem. 19)}}{\geq} \Xi^s_{u_\ell}(\bar{t}) - I_{u_\ell}(t_{\ell+1}, \bar{t}) + ((1-\rho)(1+\mu) - \eta)(\bar{t} - t_{\ell+1}) \\
&\quad + ((1-\rho)\mu - 2\rho - \eta)(t_{\ell+1} - \underline{t}) \\
&\overset{(28)}{\geq} \Psi^s_{p-1}(\bar{t}) + ((1-\rho)\mu - 2\rho - \eta)(\bar{t} - \underline{t}) + s\Delta\kappa_p(\underline{t}, \bar{t}) \\
&\overset{(2),\text{(Lem. 13)}}{\geq} \frac{C_{s+1}}{2} + C_s - \frac{C_{s+1}}{4} - \frac{C_{s+1}}{12} - \frac{C_{s+1}}{6} \geq C_s,
\end{aligned}$$

which contradicts Lemma 20. □

Theorem 12 follows from Theorem 23: the legal state condition implies that the stable local skew is $\mathcal{O}(d \log(D/d))$. The stabilization time of $\mathcal{O}(D \log D)$ follows from the dynamics of $\kappa_{\{u,v\}}$, which stabilizes to its final value of $\kappa^\infty_{\{u,v\}}$ after $\mathcal{O}(D \log D)$ time. (The weights are defined in terms of the global skew $\bar{\mathcal{G}}$, which Theorem 11 shows is $\mathcal{O}(D)$.)

7. LOWER BOUND

The lower bound of [7] stated, roughly speaking, that the stabilization time of any $\mathcal{S}$-dynamic gradient CSA with a stable gradient skew of $\bar{\mathcal{S}}$ cannot be better than $\Omega(D/\bar{\mathcal{S}}(1))$ in graphs of diameter D. For CSA with $\mathcal{O}(\log D)$-local skew, this bound implies that the stabilization time must be $\Omega(D/\log D)$. Algorithm $\mathcal{A}^{\mathrm{OPT}}$ has a stabilization time of $\mathcal{O}(D)$, which does not match the bound in [7]; however, by refining the analysis in the lower bound we can show that the algorithm is in fact asymptotically optimal in its stabilization time. The key to the stronger bound is reasoning about the *full* gradient property, which bounds the skew on paths of all distances, rather than just the local skew property, which only bounds the skew on single edges.

Let us call a dynamic gradient CSA *non-trivial* if it has a stable gradient skew satisfying $\bar{\mathcal{S}}(1) \in o(D)$. This essentially means that the algorithm guarantees a *local* skew (e.g., along single edges) that is better than the global skew. The stronger lower bound states the following.

THEOREM 24. *Let $\mathcal{F} = \{f_D : \mathbb{R}^+_0 \to \mathbb{R}^+_0 \mid D \in \mathbb{R}\}$ be a family of functions, and let $c_1, c_2 \in (0, 1/16)$ be constants such that for all $f_D \in \mathcal{F}$ we have $f_D(c_1 D) \leq c_2 D$. Let $\mathcal{A}$ be a non-trivial stabilizing CSA guaranteeing a dynamic gradient skew of f_D in graphs of weighted diameter D. Then the stabilization time of $\mathcal{A}$ is at least $\Omega(D)$.*

The proof is similar to the one in [7]; it appears in the technical report. Algorithms $\mathcal{A}^{\mathrm{OPT}}$, $\mathcal{A}^{\mathrm{DW}}$, which guarantee a stable gradient skew of $\bar{\mathcal{S}}(d) \in \mathcal{O}(d \log(D/d))$, satisfy the conditions of the theorem; hence, $\mathcal{A}^{\mathrm{OPT}}$ has optimal stabilization time, while $\mathcal{A}^{\mathrm{DW}}$ is within a $\log D$ factor of optimal.

8. REFERENCES

[1] S. Biaz and J. Lundelius Welch. Closed Form Bounds for Clock Synchronization Under Simple Uncertainty Assumptions. *Information Processing Letters*, 80(3):151–157, 2001.

[2] D. Dolev, J. Halpern, B. Simons, and R. Strong. Dynamic Fault-tolerant Clock Synchronization. *Journal of the ACM (JACM)*, 42(1):143–185, 1995.

[3] D. Dolev, J. Halpern, and R. Strong. On the Possibility and Impossibility of Achieving Clock Synchronization. In *Proc. 16th Annual ACM Symposium on Theory of Computing (STOC)*, pages 504–511, 1984.

[4] J. Elson, L. Girod, and D. Estrin. Fine-grained Network Time Synchronization Using Reference Broadcasts. *ACM SIGOPS Operating Systems Review*, 36:147–163, 2002.

[5] R. Fan and N. Lynch. Gradient Clock Synchronization. In *Proc. 23rd Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 320–327, 2004.

[6] J. Halpern, B. Simons, R. Strong, and D. Dolev. Fault-tolerant Clock Synchronization. In *Proc. 3rd Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 89–102, 1984.

[7] F. Kuhn, T. Locher, and R. Oshman. Gradient Clock Synchronization in Dynamic Networks. In *Proc. 21st ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 270–279, 2009.

[8] F. Kuhn and R. Oshman. Gradient Clock Synchronization Using Reference Broadcasts. In *Proc. 13th International Conference on Principles of Distributed Systems (OPODIS)*, pages 204–218, 2009.

[9] L. Lamport and P. Melliar-Smith. Synchronizing Clocks in the Presence of Faults. *Journal of the ACM (JACM)*, 32(1):52–78, 1985.

[10] C. Lenzen, T. Locher, and R. Wattenhofer. Clock Synchronization with Bounded Global and Local Skew. In *Proc. 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 500–510, 2008.

[11] C. Lenzen, T. Locher, and R. Wattenhofer. Tight Bounds for Clock Synchronization. In *Proc. 28rd Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 46–55, 2009.

[12] T. Locher. *Foundations of Aggregation and Synchronization in Distributed Systems*. PhD thesis, ETH Zurich, 2009.

[13] T. Locher and R. Wattenhofer. Oblivious Gradient Clock Synchronization. In *Proc. 20th International Symposium on Distributed Computing (DISC)*, pages 520–533, 2006.

[14] J. Lundelius and N. Lynch. A New Fault-tolerant Algorithm for Clock Synchronization. In *Proc. 3rd Annual ACM Symposium on Principles of Distributed Computing*, pages 75–88. ACM, 1984.

[15] J. Lundelius and N. Lynch. An Upper and Lower Bound for Clock Synchronization. *Information and Control*, 62(2/3):190–204, 1984.

[16] K. Marzullo and S. Owicki. Maintaining the Time in a Distributed System. In *Proc. 2nd Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 44–54, 1983.

[17] R. Ostrovsky and B. Patt-Shamir. Optimal and Efficient Clock Synchronization under Drifting Clocks. In *Proc. 18th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 400–414, 1999.

[18] B. Patt-Shamir and S. Rajsbaum. A Theory of Clock Synchronization. In *Proc. 26th Annual ACM Symposium on Theory of Computing (STOC)*, pages 810–819, 1994.

[19] T. K. Srikanth and S. Toueg. Optimal Clock Synchronization. *Journal of the ACM*, 34(3):626–645, 1987.