Tight Bounds for Clock Synchronization

[Extended Abstract] *

Christoph Lenzen
Computer Engineering and Networks Laboratory (TIK)
ETH Zurich, 8092 Zurich, Switzerland
lenzen@tik.ee.ethz.ch

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

Roger Wattenhofer
Computer Engineering and Networks Laboratory (TIK)
ETH Zurich, 8092 Zurich, Switzerland
wattenhofer@tik.ee.ethz.ch

ABSTRACT

We present a novel clock synchronization algorithm and prove tight upper and lower bounds on the worst-case clock skew that may occur between any two participants in any given distributed system. More importantly, the worst-case clock skew between neighboring nodes is (asymptotically) at most a factor of two larger than the best possible bound. While previous results solely focused on the dependency of the skew bounds on the network diameter, we prove that our techniques are optimal also with respect to the maximum clock drift, the uncertainty in message delays, and the imposed bounds on the clock rates. The presented results all hold in a general model where both the clock drifts and the message delays may vary arbitrarily within pre-specified bounds.

Furthermore, our algorithm exhibits a number of other highly desirable properties. First, the algorithm ensures that the clock values remain in an affine linear envelope of real time. A better bound on the accuracy with respect to real time cannot be achieved in the absence of an external timer. Second, the algorithm minimizes the number and size of messages that need to be exchanged in a given time period. Moreover, only a small number of bits must be stored locally for each neighbor. Finally, our algorithm can easily be adapted for a variety of other prominent synchronization models.

Categories and Subject Descriptors

C.2.4 [**Computer Systems Organization**]: Computer-Communication Networks—*Distributed Systems*; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Computations on Discrete Structures*

General Terms

Algorithms, Theory

1. INTRODUCTION

There is a wide range of tasks in distributed systems requiring its participants to maintain a common notion of time, which necessitates the use of a synchronization algorithm. In distributed systems, the participants synchronize by perpetually sending messages containing information about their current state and by applying a clock synchronization algorithm to update their clocks.

We model a distributed system as a graph $G = (V, E)$, where the nodes in V denote the participants in the system and each edge $\{v, w\} \in E$ represents a bidirectional communication link between v and w. Each node is equipped with a hardware clock with a bounded but variable drift. A *logical clock value* is computed according to the local hardware clock value and the messages received from neighboring nodes. Since it is reasonable to expect that events occuring at different real times also happen at different logical times, we demand that nodes increase the value of their logical clocks at least at a certain minimum rate. The goal is to minimize the skew between the logical clocks. The main difficulty lies in the fact that the nodes know neither the potentially variable *hardware clock rates* nor the *message delays*, which can also vary arbitrarily. Moreover, there is no external clock that could inform the nodes about the real time once in a while.

Naturally, one objective is to minimize the skew between any two nodes in the graph, regardless of the distance in G between them. We call the maximum worst-case skew between any two nodes in the graph the *global skew*. Apart from minimizing the global skew, it is essential for several distributed applications that the clock skew between neighboring nodes is as small as possible. One could even think of applications where the global skew is not of great concern, but any node only needs to be well synchronized with nodes in its vicinity. This is the case if occurrences of events are only of local importance and do not bear any (immediate) significance for nodes that are not close-by.[1] The so-called *gradient property*, which has been introduced in [4], captures this optimization criterion. It requires that the clock skew between any two nodes v, w is bounded by a monotonically increasing function of their distance $d(v, w)$. Thus, neigh-

*All proofs are omitted and can be found in [10, 11].

[1]A prominent example is TDMA in wireless networks where nodes depend on locally well synchronized time slots.

PODC'09, August 10–12, 2009, Calgary, Alberta, Canada.

boring nodes should always be well synchronized, whereas the logical clock values of distant nodes are allowed to deviate more. We will refer to the maximum worst-case clock skew between neighboring nodes as the *local skew*.

Ideally, an algorithm guarantees good bounds on both the global and the local skew. It has been shown that the smallest possible global skew that any algorithm can achieve is $D/2$ [2], where D denotes the diameter of the graph.[2] As far as the local skew is concerned, it has been proven in the surprising work by Fan and Lynch [4] that a skew of $\Omega(\log D/\log\log D)$ between neighboring nodes cannot be prevented. While it is fairly easy to come up with an algorithm guaranteeing a bound of $\Theta(D)$ on the global skew, finding an algorithm with a strong gradient property is more challenging. At the time when Fan and Lynch presented their lower bound, no algorithm guaranteeing a local skew sublinear in the diameter was known. In the meantime, the upper bound on the local skew has been improved to $\mathcal{O}(\sqrt{D})$ [13] and subsequently to $\mathcal{O}(\log D)$ [9]. However, both the upper and the lower bounds so far neglected the influence of other parameters such as the maximum clock drift rate, leaving room for improvements not visible when considering only the network diameter.

The objective of this work is to provide tight bounds on the degree of synchronization that can be achieved, taking many parameters such as the delay uncertainty and the maximum clock drift rate into account. In other words, we show how the bounds on the worst-case skews depend on all these parameters. Another aspect of clock synchronization, which so far has not received much attention, is that a practical clock synchronization algorithm must ensure that the rates of progress of all logical clock values are always within specific bounds, i.e., the clock values are not allowed to change substantially in a short time.[3] However, bounding the clock rates inhibits the ability of an algorithm to react to clock skews, which have to be kept as small as possible. Apart from bounding the logical clock rates in order to ensure that the clock values do not change abruptly, it may further be desirable to keep the logical clock values as close to real time as possible, i.e., an algorithm should guarantee the best possible real-time approximation in the absence of an external timer.

We propose a simple algorithm that bounds the minimum and maximum progress of the logical clocks and ensures that the logical clock values always remain within an affine linear envelope of real time. We prove matching bounds on both the global and the local skew of this algorithm. Given the discrete nature of clocked computational devices, we allow the algorithm to act only at certain *clock pulses*, which are periodically triggered by the system clock. What is more, we show that the message frequency can be kept quite low without increasing the worst-case clock differences significantly, which implies that techniques such as piggybacking can be employed. This is a viable option especially considering that we only require a few bits to be sent in each message, which can be included in (or appended to) any message sent by another application. To round off our analysis, we discuss how to adapt our algorithm to other synchronization models, such as external synchronization where a source of real time is available to some of the network participants, or (non-Byzantine) failures. These results imply that the techniques in this work offer asymptotically optimal solutions to the synchronization problem with respect to several optimization criteria for a wide range of models.

[2]For ease of presentation, we normalize the uncertainty in message delays to one in this introduction and the related work section as all bounds depend linearly on it.

[3]For instance, if velocities are to be determined with the help of local clocks, clock jumps would severely deteriorate the accuracy of the measurements.

2. RELATED WORK

There is a large body of work on the fundamental problem of synchronizing clocks in distributed systems. Most work mainly focuses on bounding the skew that may occur between any two clocks and also the communication costs that are required in order to guarantee a certain degree of synchronization (see, e.g., [14, 19, 21, 23]). It has been shown that a skew of $D/2$ cannot be avoided on any graph G of diameter D [2]. We will prove a stronger lower bound of roughly D for clock synchronization algorithms that strive to keep all clock values within a linear envelope of real time. The clock synchronization algorithm by Srikanth and Toueg [23] achieves a bound of $\mathcal{O}(D)$ on the skew of any two clocks at all times and is thus asymptotically optimal. The authors further show that the accuracy of their algorithm with respect to real time is also optimal as all clocks are always within a linear envelope of real time. However, their algorithm incurs a skew of $\Theta(D)$ between neighboring nodes in the worst case.

In their seminal work [4] that introduced the problem of synchronizing clocks of close nodes as accurately as possible, Fan and Lynch showed that no algorithm can avoid a clock skew of $\Omega(\log_b D)$ between neighboring nodes, where $b \in \mathcal{O}(\log D)$. The only imposed constraint is that nodes are required to increase their clock values at a given minimum progress rate, which is quite natural as otherwise events that occur at different (real) times may happen at the same logical time if an algorithm could simply halt the clocks. Subsequently, it has been shown that this bound also holds if all messages arrive instantaneously, but an adversary can determine when synchronization messages may be sent [16]. However, Fan and Lynch treated the maximum hardware clock drift ε as a constant; taking it into account as a parameter reveals that $b \in \mathcal{O}((\log D)/\varepsilon)$. We improve their result to $b \in \Theta(1/\varepsilon)$, i.e., any algorithm may experience a local skew of $\Omega(\log_{1/\varepsilon} D)$. Furthermore, we show that if clock rates bounded by constants, b depends linearly on the difference of the maximum and the minimum rate. In particular, if logical clocks must guarantee an optimal drift of $\mathcal{O}(\varepsilon)$, the bound on the local skew becomes $\Theta(\log D)$.

There has also been a lot of (more practical) work on clock synchronization for specific computing environments. For example, the clock synchronization problem in wireless sensor networks has been extensively studied [3, 6, 15, 20]. Synchronizing clocks is also an important issue for other forms of distributed systems such as the Internet [17] or systems-on-a-chip [5]. However, apart from processor design, where one seeks to control signal delays by means of placement and wiring (see, e.g., [7] and references therein), synchronizing close-by devices particularly well has scarcely been considered as an optimization criterion. In fact, to the best of our knowledge, the only attempt has been made in the context of sensor networks [22].

The first algorithm guaranteeing a sublinear bound on the clock skew between neighboring nodes achieved a bound of

$\mathcal{O}\left(\varepsilon\sqrt{D}\right)$ [12, 13], a result that has recently been generalized to dynamic networks [8]. For static networks, the upper bound has been improved to $\mathcal{O}(\log D)$ [9]. However, the algorithm that guarantees a logarithmic bound has several disadvantages: Apart from being quite complicated, the algorithm has the undesirable property that the clock values can "jump" by $\Theta(\log D)$ in a single time step, and thus the clock values may not change smoothly. Moreover, both the message frequency and the size of the messages are fairly large, which prohibits techniques such as piggybacking and which may imply that the algorithm is not useful in practice. What is more, the base of the logarithm can hardly be increased if ε becomes small. Since typically $\varepsilon \ll 1$, a notable gap to the lower bound remains. The algorithm presented in this work does not have these shortcomings.

3. MODEL

We model a distributed system as a connected, undirected graph $G = (V, E)$ of diameter D, where nodes represent computational devices and egdes represent bidirectional communication links. Each node v can communicate with all neighboring nodes by exchanging messages. The set of v's neighbors is denoted by $\mathcal{N}_v := \{w \in V \,|\, \{v, w\} \in E\}$. We assume that, for any two nodes $u, w \in \mathcal{N}_v$, node v can distinguish u from w, e.g., by means of a port numbering or node identifiers, and also that all communication is reliable, i.e., messages are never lost. However, communication takes some time, and this *delay* may vary. In general, a message delay may consist of two parts, a fixed known delay and an additional variable delay. Since any fixed fraction of the total delay can be added to a received clock value, we define it to be zero (we will discuss the impact of this simplification in Section 7.3). Thus, the time that passes from the moment a message is sent until the recipient can act upon it may be any value in the range $[0, \mathcal{T}]$, where $\mathcal{T}$ is the *delay uncertainty*. While the bound $\mathcal{T}$ is unknown to the algorithm, we assume that the nodes know an upper bound $\hat{\mathcal{T}} \in \mathcal{O}(\mathcal{T})$ on $\mathcal{T}$.

Each node v is equipped with a *hardware clock* H_v which starts running at the time t_v when v is initialized. The first node starts its clock at real time $t = 0$. An *initialization message* is then flooded through the network in order to start the clocks at the other nodes. We denote the value of the hardware clock at real time t by $H_v(t)$, i.e., $H_v : \mathbb{R}_0^+ \to \mathbb{R}_0^+$ is a monotonically increasing function. The value of the hardware clock of v is 0 until time t_v and $H_v(t) := \int_{t_v}^{t} h_v(\tau)\, d\tau$ afterwards, where $h_v(\tau)$ is the *hardware clock rate* of v at time τ. The clock rates may vary over time, but we assume that there is a constant $0 < \varepsilon < 1$ such that the following condition holds.

$$\forall v \in V\ \forall t \geq t_v : 1 - \varepsilon \leq h_v(t) \leq 1 + \varepsilon.$$

While the exact value of ε is unknown, we assume that the nodes know an upper bound $\hat{\varepsilon}$ that is strictly smaller than one, i.e., hardware clocks guarantee a strictly positive minimum progress rate.

Since a computational device typically synchronizes its operations internally based on a *clock pulse*, we assume that a node $v \in V$ performs computations and sends and receives messages only at such clock pulses. Each node v has a clock pulse at time t if $H_v(t)$ is an integer value.[4] We call such a time a *tick event at node* v or simply a *tick* at v. Note that this definition implies that

$$\mathcal{T} \geq \frac{1}{1-\varepsilon}, \tag{1}$$

as v might have to wait arbitrarily close to $\frac{1}{1-\varepsilon}$ time until it can process a message that arrived right after the last tick event.

Additionally, each node v has a *logical clock* L_v, which is also a function $L_v : \mathbb{R}_0^+ \to \mathbb{R}_0^+$ whose value until time t_v is 0 as well. It is desirable to keep all logical clock values within an (affine) linear envelope of real time. Therefore, we require that any algorithm satisfies the following condition, which takes the different initialization times t_v into account.

$$\forall v \in V\ \forall \text{ticks } t \text{ at } v :\ (1-\varepsilon)(t-t_v) \leq L_v(t) \leq (1+\varepsilon)t. \tag{2}$$

Moreover, we demand that the logical clocks behave normally in the sense that the logical clock values may not change dramatically in a short time. Formally, there are constants $0 < \alpha \leq 1 - \varepsilon$ and $1 + \varepsilon \leq \beta$ such that

$$\begin{aligned} &\forall v \in V\ \forall \text{ticks } t < t' \text{ at } v : \\ &\alpha(t' - t) \leq L_v(t') - L_v(t) \leq \beta(t' - t). \end{aligned} \tag{3}$$

The increased (or lowered) clock rates of the logical clocks allow the nodes to correct differences between the logical clock values in the network. The difference between the values of logical clocks is called *clock skew*. Ideally, the logical clocks behave just like the hardware clocks even in the presence of clock skews, albeit with a slightly worse clock drift, i.e., $\alpha \in 1 - \mathcal{O}(\varepsilon)$ and $\beta \in 1 + \mathcal{O}(\varepsilon)$. Note that Condition (3) implies that clocks are not allowed to run backwards and thus the algorithm can only manipulate the logical clock value by increasing it.

Since the algorithm modifies the value of $L_v(\cdot)$ at discrete points in time, we have to specify the meaning of $L_v(t)$ at times where the clock value changes. If L_v is increased at time t, we define $L_v(t)$ to be the value *after* the algorithm changed it.

A clock synchronization algorithm $\mathcal{A}$ executed at node v specifies how the logical clock $L_v(t)$ of node v is adapted based on its hardware clock and the information received from its neighbors up to time t in such a way that Conditions (2) and (3) are satisfied. If an algorithm may only act at certain clock pulses, the algorithm is called a *discrete* clock synchronization algorithm. By contrast, an algorithm that is capable of changing the logical clock value at any time is referred to as a *continuous* clock synchronization algorithm. While such an algorithm is slightly more powerful because it does not have to wait for clock pulses, its behavior is more restricted as Conditions (2) and (3) are naturally expected to hold at all times. In this paper, we mainly consider the discrete clock synchronization problem. Not surprisingly, basically the same results can be obtained for continuous clock synchronization (cf. [12]).

Given a clock synchronization algorithm $\mathcal{A}$ and a (connected) graph G, an *execution* $\mathcal{E}$ specifies the delays of all messages and also the hardware clock rates of all nodes at each point in time when $\mathcal{A}$ is executed on G. Thus, the information contained in an execution completely determines the state of the network at any time for a run of $\mathcal{A}$ on G. The *global* and the *local skew* are formally defined as follows:

[4] Of course, only tick events at times $t \geq t_v$ when v has been initialized are considered.

Definition 3.1 (Global Skew). *Given the connected graph $G = (V, E)$ and the clock synchronization algorithm $\mathcal{A}$, the* global skew *is defined as the value*

$$\sup_{\mathcal{E}, v \in V, w \in V, t} \{L_v(t) - L_w(t)\},$$

where $\mathcal{E}$ is any execution of $\mathcal{A}$ on G.

Definition 3.2 (Local Skew). *Given the connected graph $G = (V, E)$ and the clock synchronization algorithm $\mathcal{A}$, the* local skew *is defined as the value*

$$\sup_{\mathcal{E}, v \in V, w \in \mathcal{N}_v, t} \{L_v(t) - L_w(t)\},$$

where $\mathcal{E}$ is any execution of $\mathcal{A}$ on G.

Naturally, the goal of an algorithm $\mathcal{A}$ is to ensure the best possible bounds on both the global and the local skew on any graph G.

4. ALGORITHM

In this section, we introduce the synchronization algorithm $\mathcal{A}^{\text{opt}}$. Due to lack of space, we confine ourselves to briefly describing the core aspects. For a more detailed discussion the reader is referred to [10].

In order to synchronize the logical clocks, any node v must perpetually send synchronization messages informing the neighboring nodes about its current clock value L_v. Node v itself adapts its clock value according to the information received from its neighbors. However, the information about the neighboring clock values is not sufficient to guarantee an optimal bound on the global skew, because the neighboring nodes might have similar clock values while the skew to nodes at greater distances may be large. This problem can be solved by including an estimate of the maximum clock value in the network in each message. Hence, whenever a node v sends a message, it is of the form $\langle L_v, L_v + \Lambda_v^{\max} \rangle$, where $\Lambda_v^{\max} \geq 0$ is the estimated clock skew between v's clock value and the currently largest clock value. Similarly, we define for any $w \in \mathcal{N}_v$ that Λ_v^w is the estimated difference between v's and w's clock value from v's perspective. This variable is updated whenever v receives a new estimate L_v^w of the current clock value of w.

At each tick, any node $v \in V$ receives a set $\mathcal{M}$ of such messages. The set $\mathcal{M}$ may consist of any number of messages from each neighbor as messages with different delays may be processed at the same tick. Therefore, we define the set $\mathcal{L}_w := \{L_w \,|\, \langle L_w, L_w + \Lambda_w^{\max} \rangle \in \mathcal{M}\} \cup \{0\}$ of all clock values received from w since the last tick event for each neighbor $w \in \mathcal{N}_v$.[5] Analogously, let $\mathcal{L}_{\max} := \{L_u + \Lambda_u^{\max} \,|\, \langle L_u, L_u + \Lambda_u^{\max} \rangle \in \mathcal{M}\} \cup \{0\}$ denote the set of estimates of the maximum clock value received from all $u \in \mathcal{N}_v$.

The algorithm takes three parameters, H_0, μ, and κ. The first parameter $H_0 \geq 1$ determines the *message frequency*: As we will see, each node v sends a message to all neighbors at the latest after its hardware clock H_v has advanced by H_0. In order to determine whether the next multiple of H_0 is reached, each node v stores a variable $\tilde{H}_v$, which holds v's estimate of the largest multiple of H_0 that any hardware clock has reached yet. If there are no clock skews, each node v increases its clock value L_v by exactly 1 at each tick, but v may increase L_v by more if its clock is behind.

[5]Zero is merely added to ensure that $\mathcal{L}_w$ is not empty.

The parameter $\mu > 0$ bounds the progress rate in that the algorithm demands that the logical clock value is increased by at most $1 + \mu$ at each tick. Given the bound ε on the clock drift and the fact that the algorithm increases by at least 1 and at most $1 + \mu$ at each tick event, the amortized logical clock rates are bounded by $\alpha = 1 - \varepsilon$ and $\beta = (1 + \varepsilon)(1 + \mu)$. It may be desirable to keep the parameter μ as small as possible.[6] The same is true for the parameter κ, which can be considered a "base unit" that the algorithm uses to measure clock skew. It is beneficial to minimize this parameter because the local skew depends linearly on it. However, it is necessary to choose $\kappa \in \Omega(\mathcal{T})$. The precise conditions that have to be met are as follows:

$$\mu \geq \frac{24\varepsilon}{1 - \varepsilon} \tag{4}$$

$$\text{and} \quad \kappa \geq 2\left((1+\varepsilon)(1+\mu)\left(\mathcal{T} + \frac{1}{1-\varepsilon}\right) + 4 + \mu + (2\varepsilon + \mu)H_0\right). \tag{5}$$

Thus, for a minimal μ, the precision of the clocks reduces by slightly more than one order of magnitude while clock skews are corrected. Naturally, μ can also be set to a larger value, which results in a smaller local skew. This effect is limited by Condition (5), which states that we must choose $\mu \in \mathcal{O}(1)$ as otherwise the necessary increase of κ outweighs the logarithmic gain from a larger value of μ. Throughout this paper, we will assume that Inequalities (4) and (5) are satisfied, and Theorems 5.1 and 5.4 make use of this assumption.

Since all variables and parameters have been introduced, we can now proceed to describe the algorithm itself. As mentioned before, some node starts the execution of the algorithm by flooding an initialization message through the entire network. We simply define that the first received synchronization message is considered the initialization message. If a node receives a synchronization message for the first time, it executes Algorithm 1.

Algorithm 1 Initialization($\mathcal{M}$)

```
1: L_v := 0;  H̃_v := ⌊max L_max⌋;  Λ_v^max := max L_max
2: for w ∈ N_v do
3:     L_v^w := max L_w
4:     Λ_v^w := L_v − L_v^w
5: Send ⟨L_v, L_v + Λ_v^max⟩ to all u ∈ N_v
6: send := false
```

An initialized node performs three subroutines in the given order at each tick event:

1. **UpdateVariables:** Adapt the local variables according to the information received since the last tick event.
2. **SetClock:** Decide by how much the clock value is increased, set the logical clock and adapt the affected local variables.
3. **SendMessage:** Send a message if necessary.

Since the first and third routine (listed in Algorithm 2 and Algorithm 4) just adapt local variables according to the

[6]Note that if $\mu \in \mathcal{O}(\varepsilon)$, the maximum clock rate is bounded by $1 + \mathcal{O}(\varepsilon)$.

Algorithm 2 UpdateVariables($\mathcal{M}$)

1: $L_{\max} := \max \mathcal{L}_{\max}$
2: **if** $L_{\max} \geq L_v + 1 + \Lambda_v^{\max}$ **then**
3: $\quad \tilde{H}_v := \lfloor L_{\max} \rfloor$
4: $\quad \Lambda_v^{\max} := L_{\max} - (L_v + 1)$
5: $\quad send := true$
6: **for** $w \in \mathcal{N}_v$ where $\max \mathcal{L}_w > L_v^w$ **do**
7: $\quad L_v^w := \max \mathcal{L}_w$
8: $\quad \Lambda_v^w := L_v + 1 - L_v^w$
9: $\Lambda_v^{\downarrow} := \max_{u \in \mathcal{N}_v} \{\Lambda_v^u\}$
10: $\Lambda_v^{\uparrow} := \max_{u \in \mathcal{N}_v} \{-\Lambda_v^u\}$

newly received values and decide whether to send a message, respectively, we restrict our attention to the core of $\mathcal{A}^{\mathrm{opt}}$, the procedure **SetClock**. The goal of the subroutine is to determine if L_v has to be increased by a certain value R_v in addition to the normal increase of 1. The steps of this subroutine are summarized in Algorithm 3.

Algorithm 3 SetClock()

1: $R_v := \sup \left\{ R \in \mathbb{R} \;\middle|\; \left\lfloor \frac{\Lambda_v^{\uparrow} - R}{\kappa} \right\rfloor \geq \left\lfloor \frac{\Lambda_v^{\downarrow} + R}{\kappa} \right\rfloor \right\}$
2: $R_v := \max \left\{ \min \left\{ \max \left\{ \kappa - \Lambda_v^{\downarrow}, R_v \right\}, \mu, \Lambda_v^{\max} \right\}, 0 \right\}$
3: $L_v := L_v + 1 + R_v$
4: $\Lambda_v^{\max} := \Lambda_v^{\max} - R_v$
5: **for** $w \in \mathcal{N}_v$ **do**
6: $\quad \Lambda_v^w := \Lambda_v^w + R_v$

In Line 1, the node computes which value R_v should take according to the locally observed clock skew. Roughly speaking, the goal of Algorithm 3 is to ensure that the rounded clock skews to the neighbor whose clock is assumed to be behind the most ($\Lambda_v^{\downarrow}$) and the neighbor with the largest estimated clock value ($\Lambda_v^{\uparrow}$) are the same integer multiple of κ. The variable R_v attains the largest value that satisfies this constraint. More precisely, if $\Lambda_v^{\uparrow} \leq s\kappa$ and $\Lambda_v^{\downarrow} \geq s\kappa$ for some $s \in \mathbb{N}$, v does not raise its clock value, i.e., $R_v = 0$. If there is no integer s that satisfies this condition, $R_v > 0$ becomes exactly the increase of the clock value that causes the condition to hold. Line 1 of Algorithm 3 is a concise formulation of this rule.

In Line 2, the final value of R_v is determined. Any node is allowed to raise its clock at least κ above the smallest estimate of its neighbors' clock value. Therefore, R_v is simply set to the maximum of $\kappa - \Lambda_v^{\downarrow}$ and the value R_v computed in Line 1. In other words, algorithm $\mathcal{A}^{\mathrm{opt}}$ tolerates a clock skew of κ, which ensures that the node with the smallest clock value in the network is able to raise its clock value even if it may have received delayed messages from its neighbors containing smaller clock values than its own. However, v is not allowed to raise its clock by more than μ or to set its clock to a larger clock value than the (estimate of) the largest clock value in the network. This condition prevents nodes from violating Condition (2) and gives slow nodes the possibility to catch up as the fast nodes can only increase their clock values at low rates. The outer maximum simply ensures that $R_v \geq 0$. After computing R_v, L_v is set to the new value and the estimated clock differences are adapted accordingly in Lines 4-6. Note that $\Lambda_v^{\max}$ and the estimates Λ_v^w for all $w \in \mathcal{N}_v$ are only changed by R_v (and not by $1 + R_v$), which basically means that v assumes that all other nodes also have a tick event at the same time and increase their clock values by 1. This assumption ensures that the accuracy of these estimates deteriotes slowly.

Algorithm 4 SendMessage()

1: **if** $L_v + \Lambda_v^{\max} \geq \tilde{H}_v + H_0$ **then**
2: $\quad send := true$
3: $\quad \tilde{H}_v := \tilde{H}_v + H_0$
4: **if** $send$ **then**
5: $\quad$ Send $\langle L_v, L_v + \Lambda_v^{\max} \rangle$ to all $u \in \mathcal{N}_v$
6: $\quad send := false$

5. SKEW BOUNDS

We proceed by presenting the upper bounds on the global and local skew that Algorithm $\mathcal{A}^{\mathrm{opt}}$ guarantees together with the matching lower bounds.

5.1 Global Skew

First, we state the bound on the global skew when executing $\mathcal{A}^{\mathrm{opt}}$ on any graph G.

THEOREM 5.1. *The global skew of Algorithm $\mathcal{A}^{\mathrm{opt}}$ is bounded by*

$$\mathcal{G} := (1+\varepsilon)D\mathcal{T} + \frac{2\varepsilon}{1-\varepsilon}\partial H + \frac{6}{1-\varepsilon}. \tag{6}$$

The corresponding lower bound slightly depends on how accurate the estimates $\hat{\varepsilon}$ and $\hat{\mathcal{T}}$ on the clock drift and the delay uncertainty are.

THEOREM 5.2. *Assume that a clock synchronization algorithm $\mathcal{A}$ is equipped with initial parameters $\hat{\varepsilon} \in (0,1)$, and $\hat{\mathcal{T}} \in \mathbb{R}^+$ such that $c_1\hat{\mathcal{T}} \leq \mathcal{T} \leq \hat{\mathcal{T}}$ and $c_2\hat{\varepsilon} \leq \varepsilon \leq \hat{\varepsilon}$ for certain values $c_1, c_2 \in (0,1]$. Define $\varrho := \min\{\varepsilon, (1-c_2\hat{\varepsilon})/c_1 - 1\} \in [-\varepsilon, \varepsilon]$. If algorithm $\mathcal{A}$ is bound to satisfy Condition (2), it cannot avoid a global skew of at least*

$$(1+\varrho)D\mathcal{T}$$

on any graph G of diameter D.

We can conclude from this theorem that the estimates of $\mathcal{T}$ and ε must be extremely accurate in order to guarantee a better bound than $(1+\varepsilon)D\mathcal{T}$. However, even if the exact values are known, a global skew of $(1-\varepsilon)D\mathcal{T}$ cannot be prevented subject to the condition that the logical clock values must be within a linear envelope of real time.

COROLLARY 5.3. *Without knowledge of a lower bound on ε, no clock synchronization algorithm can avoid a global skew of $D\mathcal{T}$. Moreover, no algorithm without knowledge of bounds on $\mathcal{T}$ stronger than $\mathcal{T} \in \left[(1-\varepsilon)\hat{\mathcal{T}}/(1+\varepsilon), \hat{\mathcal{T}}\right]$ can achieve a better bound on the global skew than $(1+\varepsilon)D\mathcal{T}$.*

This corollary and Theorem 5.1 imply that $\mathcal{A}^{\mathrm{opt}}$ is essentially optimal as far as the global skew is concerned. As mentioned earlier, it can be shown that a global skew of $D\mathcal{T}/2$ cannot be prevented even if the restriction that the algorithm must satisfy Condition (2) is dropped [2]. Thus, the bound on the global skew of $\mathcal{A}^{\mathrm{opt}}$ is roughly a factor of two worse than the bound on the global skew of any algorithm whose behavior is not constrained by any additional restrictions.

5.2 Local Skew

The upper bound states that the local skew of $\mathcal{A}^{\text{opt}}$ grows logarithmically with the diameter D of the graph.

THEOREM 5.4. *In any execution of Algorithm $\mathcal{A}^{\text{opt}}$, the skew between any two nodes $v, w \in V$ at distance $d(v,w)$ is bounded by*

$$\mathcal{O}\left(\kappa d(v,w) \log_{\mu/\varepsilon} \frac{\mathcal{T}D}{\kappa d(v,w)}\right).$$

In particular, the local skew of Algorithm $\mathcal{A}^{\text{opt}}$ is upper bounded by

$$\kappa\left(\left\lceil \log_\sigma \frac{2\mathcal{G}}{\kappa}\right\rceil + \frac{1}{2}\right),$$

where $\sigma := \left\lfloor \frac{\mu(1-\varepsilon)}{8\varepsilon}\right\rfloor - 1$ and $\mathcal{G}$ is the bound on the global skew from Theorem 5.1.

Observe that the base of the logarithm σ is in $\Theta(\mu/\varepsilon)$. Thus, choosing $\kappa \in \Theta((1+\mu)\mathcal{T} + \mu H_0)$ results in a local skew of

$$\mathcal{O}\left(((1+\mu)\,\mathcal{T} + \mu H_0)\log_{\mu/\varepsilon} D\right).$$

Recall that we can choose $\kappa \in \Theta(\mathcal{T})$ because we know upper bounds $\hat{\mathcal{T}} \in \mathcal{O}(\mathcal{T})$ and $\hat{\varepsilon} < 1$ on $\mathcal{T}$ and ε, respectively. If $\mu \in \Theta(\varepsilon)$ and $H_0 \in \mathcal{O}(\mathcal{T}/\mu) = \mathcal{O}(\mathcal{T}/\varepsilon)$, Theorem 5.4 states that the local skew is upper bounded by $\mathcal{O}(\mathcal{T}\log D)$. Note that choosing $\mu \in \Theta(\varepsilon)$ entails that the maximum logical clock rate β is upper bounded by $1+\mathcal{O}(\varepsilon)$. If the logical clock rate is allowed to be larger than the hardware clock rate by a constant factor, i.e., $\mu \in \Theta(1)$, and we choose $H_0 \in \mathcal{O}(\mathcal{T})$, the bound on the local skew reduces to $\mathcal{O}\left(\mathcal{T}\log_{1/\varepsilon} D\right)$.

Two theorems show that these bounds are asymptotically optimal. The first one reveals that permitting (amortized) clock rates of $\beta \in \omega(1)$ is of no use when trying to achieve small local skews.

THEOREM 5.5. *No continuous clock synchronization algorithm can achieve a better bound on the local skew than*

$$\Omega\left(\alpha\mathcal{T}\left(1+\log_{1/\varepsilon} D\right)\right)$$

on any graph of diameter D.

Thus, for $\mu \in \Theta(1)$ and $\kappa \in \Theta(\mathcal{T})$, $\mathcal{A}^{\text{opt}}$ is asymptotically optimal also with respect to the local skew. The second lower bound on the local skew states that the dependency of the base σ on μ is asymptotically optimal as well.

THEOREM 5.6. *For any discrete clock synchronization algorithm and any graph of diameter D the local skew is lower bounded by*

$$\Omega\left(\alpha\mathcal{T}\left(1+\log_{(\beta-\alpha)/(\alpha\varepsilon)} D\right)\right).$$

If we demand that the logical clocks run roughly at the same rates as the hardware clocks, e.g., $\alpha \in 1-\mathcal{O}(\varepsilon)$ and $\beta \in 1+\mathcal{O}(\varepsilon)$, we get that $b \in \mathcal{O}(1)$ and thus a lower bound of $\Omega(\mathcal{T}\log D)$, which matches the upper bound of algorithm $\mathcal{A}^{\text{opt}}$ when $\mu \in \Theta(\varepsilon)$ and $H_0 \in \mathcal{O}(\mathcal{T}/\varepsilon)$. Similarly, if we allow a logical clock rate that is a constant times larger than real time, i.e., $\beta \in \Theta(1)$, the lower bound reduces to $\Omega(\mathcal{T}\log_{1/\varepsilon} D)$. Algorithm $\mathcal{A}^{\text{opt}}$ guarantees an upper bound on the local skew of $\mathcal{O}(\mathcal{T}\log_{1/\varepsilon} D)$ when choosing $\mu \in \Theta(1)$ and $H_0 \in \mathcal{O}(\mathcal{T})$. More generally, we get the following result.

COROLLARY 5.7. *If $\kappa \in \mathcal{O}(\mathcal{T})$, Algorithm $\mathcal{A}^{\text{opt}}$ achieves an asymptotically optimal local skew of $\Theta\left(\mathcal{T}\log_{\mu/\varepsilon} D\right)$. Furthermore, if $D, \mathcal{T} \to \infty$ and $\varepsilon \to 0$, the approximation ratio of $\mathcal{A}^{\text{opt}}$ tends to $2\hat{\mathcal{T}}/\mathcal{T}$.*

What is more, it is evident from the proof of Theorem 5.6 that the dependency of the maximum skew between two nodes $v, w \in V$ on their distance $d(v,w)$ given by Theorem 5.4 is asymptotically optimal, i.e., $\mathcal{A}^{\text{opt}}$ features an asymptotically optimal gradient property as defined by Fan and Lynch [4].

COROLLARY 5.8. *If $\beta - \alpha \in \mathcal{O}(1)$, the best worst-case guarantee on the clock skew between nodes at distance d that any algorithm can achieve is*

$$\Theta\left(\alpha\mathcal{T}d\left(1+\log_{(\beta-\alpha)/(\alpha\varepsilon)}(D/d)\right)\right).$$

We note that the previous results hold irrespective of α since $\mathcal{A}^{\text{opt}}$ can exploit smaller values of α by simply reducing the speed of the logical clocks by an appropriate factor whenever neighbors are behind too much.

Another important point that can be deduced from the proof of Theorem 5.6 is that clock skews are built up during time periods of considerable lengths. This has two profound consequences. First, for any constant $c < 1$, in some execution the average clock skew on some path, and thus also the maximum clock skew among all neighboring nodes, is $\Omega((\alpha\mathcal{T}\log_{(\beta-\alpha)/(\alpha\varepsilon)} D)/c)$ for $\Theta\left(D^{1-c}\,\mathcal{T}\right)$ time, i.e., the phenomenon of large local skews is not short-lived. Second, this means that the same asymptotic bounds hold if nodes are allowed to reduce their clock rates arbitrarily, even to or below zero, as long as the *average* clock rate in an interval of length $\Theta\left(D^{1-c}\,\mathcal{T}\right)$, for some $c < 1$, is at least α. This observation implies that it is not a severe limitation that the progress rate of all clocks is always at least α. Note that if the clocks are allowed to stand still for $\Theta(D\mathcal{T})$ time, a simple synchronizer [1], which trivially guarantees a bound of $\Theta(\alpha\mathcal{T})$ on the local skew, can be used instead of a clock synchronization protocol.

6. COMPLEXITY

In this section, we discuss the cost of Algorithm $\mathcal{A}^{\text{opt}}$ with regard to several measures. In particular, we analyze how many messages need to be exchanged and also the (maximum) size of these messages. Moreover, we upper bound the number of bits that each node needs to store locally.

6.1 Message Complexity

An essential optimization criterion is the frequency of communication required to sustain a given quality of synchronization. If resource consumption due to communication is critical, the average of this value over time, i.e., the *amortized message frequency*, is highly relevant. When executing $\mathcal{A}^{\text{opt}}$, nodes send at most one message for each integer multiple of H_0 that $L_v + \Lambda_v^{\max}$ passes [10], implying that $\mathcal{A}^{\text{opt}}$ exhibits an amortized message frequency of $\Theta(1/H_0)$ at each node. The bound from Theorem 5.4 and Inequality (5) suggest to choose $H_0 \in \Theta\left(\hat{\mathcal{T}}/\mu\right)$, which for a minimal $\mu \in \Theta(\hat{\varepsilon})$ entails that the amortized message frequency is only $\Theta\left(\hat{\varepsilon}/\hat{\mathcal{T}}\right)$.

In a short time period, however, a node v might receive $\Theta(\mathcal{G}/H_0)$ messages containing new estimates on the maximum clock value, each larger by H_0 than the previous one, which cause v to send as many messages. Thus, the algorithm in the presented form does not guarantee a non-trivial lower bound on the message frequency. The message frequency could be bounded by adding another term in the order of $\Theta(\mu H_0)$ to κ and forcing nodes to wait at least until the progress of their hardware clocks is H_0 since they last sent a message. The price of this modification is that the bound on the global skew increases by $\Theta(\varepsilon D H_0)$ as the time it takes to propagate information through the whole network increases by $\mathcal{O}(DH_0)$ while nodes increase the estimate $L_v + \Lambda_v^{\max}$ of the maximum logical clock value locally at their hardware clock rate. This is, up to constant factors, the best possible trade-off, since in the $\Theta(DH_0)$ time a pair of nodes at distance D may have to act without updates about each other's state a skew of $\Theta(\varepsilon D H_0)$ can be built up by manipulating the hardware clock rates.

6.2 Bit Complexity

Another important property is the *bit complexity*, i.e., the maximum number of bits that must be sent in a message. Since the same update information is sent to all neighbors at the same send event, we define that the bit complexity in our model is simply the maximum size of this message.

It is evident from the analysis of $\mathcal{A}^{\mathrm{opt}}$ that κ must be sufficiently large to account for any inaccuracies of the nodes' estimates of their neighbors' clock values. This implies that κ needs to be in $\Omega(\mathcal{T})$. However, this also means that—as long as we are not interested in minimizing the constant in the bound on the local skew—we may broadcast rounded clock values. Moreover the parameter $\mu \in \mathcal{O}(1)$ can be encoded as a constant times $1/n$, for a number $n \in \mathbb{N}$. The other parameters, κ and H_0, can then be chosen as integers since both are in $\Omega(1)$ anyway. We will make use of these basic observations in the following.

In order to bound the bit complexity, we cannot send the unbounded clock values. Instead, nodes simply communicate the $\mathcal{O}((1+\mu)H_0)$ progress their clocks made since they last sent a message, which requires $\mathcal{O}(\log H_0)$ bits. However, since we have that $\kappa \in \Omega(\mu H_0)$, we might as well discretize the sent values in steps of μH_0, reducing the number of bits to $\mathcal{O}(\log(1/\mu))$.[7]

Advantageously, the estimate $L_v + \Lambda_v^{\max}$ is a multiple of H_0, but unfortunately it may increase by $\Theta(\mathcal{G})$ in a single message, which would necessitate $\Omega(\log(\mathcal{T}D/H_0))$ bits. In order to reduce the number of bits that are required to encode $L_v + \Lambda_v^{\max}$, we may limit the maximum increase of this value that a node *informs* its neighbors about in a single message to $\lceil (1+\hat{\varepsilon})(1+\mu)/(1-\hat{\varepsilon}) \rceil H_0 \in \mathcal{O}(H_0)$, which can be encoded using $\mathcal{O}(1)$ bits. If the actual value is larger, v stores the difference and informs its neighbors about the remaining increase in its subsequent messages. The intuition behind this is that the maximum estimate of the maximum clock value throughout the network does not increase faster than at rate $1+\varepsilon$, therefore sending an update of at least $(1+\varepsilon)H_0/(1-\varepsilon)$ every $H_0/(1-\varepsilon)$ time is sufficient not to fall behind. We conclude that Algorithm $\mathcal{A}^{\mathrm{opt}}$ can be implemented with a bit complexity of $\mathcal{O}(\log 1/\mu) \subseteq \mathcal{O}(\log 1/\hat{\varepsilon})$.

Moreover, if we enforce that nodes wait between sending events for H_0 local time, we can further reduce this number. Since now we implicitly know of the progress of the hardware clock since the last message, we can encode the differences between the logical clock values more efficiently. To this end, the progress of the logical clock is described relative to the progress H_0 of the hardware clock, which means that a difference of at most μH_0 has to be conveyed. Since we may round the progress to multiples of μH_0, this requires merely $\mathcal{O}(1)$ bits. From this and the previous observation that only $\mathcal{O}(1)$ bits are needed to update the estimate of the maximum clock value, we deduce that this variant of $\mathcal{A}^{\mathrm{opt}}$ offers a constant bit complexity.

6.3 Space Complexity

The *space complexity* of an algorithm is the maximum amount of memory that it requires to run. Since the logical clock value L_v grows indefinitely, we disregard it in our analysis of the space complexity of $\mathcal{A}^{\mathrm{opt}}$. Furthermore, we will consider the amount of memory that the implementations of $\mathcal{A}^{\mathrm{opt}}$ discussed in the previous section require.[8]

Each node v must store the estimated clock skew Λ_v^w to each node $w \in \mathcal{N}_v$ and the estimated difference $\Lambda_v^{\max}$ to the maximum clock value. The value of Λ_v^w is bounded by $\mathcal{O}(\kappa \log_{\mu/\varepsilon} D)$ for all neighbors. Assuming that we choose $H_0 \in \Theta(\mathcal{T}/\mu)$ and $\kappa \in \Theta(\mathcal{T})$, rounding clock values to multiples of μH_0 implies that the memory requirement for these values is bounded by $\mathcal{O}(\Delta \log\log_{\mu/\varepsilon} D)$, where Δ denotes the maximum node degree. As $\Lambda_v^{\max}$ is bounded by $\mathcal{G} \in \mathcal{O}(\mathcal{T}D)$ and $L_v + \Lambda_v^{\max}$ is a multiple of H_0, it can be encoded using $\mathcal{O}(\log(\mathcal{T}D/H_0)) = \mathcal{O}(\log(\mu D))$ bits.

Furthermore, in order to properly adapt Λ_v^w, each node must store the amount of local time that elapsed since the last message from $w \in \mathcal{N}_v$ was received because in the absence of better information v assumes that the neighbors' clocks run at the same rate as its own hardware clock. This requires $\mathcal{O}(\log(H_0))$ memory for each $w \in \mathcal{N}_v$. However, such a high accuracy is not needed. Instead, a local counter of size $\Theta(\log(\mu H_0)) = \Theta(\log \mathcal{T})$ can be used to generate events every $\Theta(\mathcal{T})$ local time and reduce the resolution to $\Theta(\mu H_0)$, and the necessary memory requirement to $\mathcal{O}(1/\mu)$ bits per neighbor. Thus, these values require $\mathcal{O}(\Delta \log \mu + \log \mathcal{T})$ bits in total.

If the proposed technique to update the estimates of the maximum clock value only by constant multiples of H_0 is employed (in order to keep the bit complexity low), we have to keep track of the updates that have already occurred. Since nodes send identical messages to all neighbors, a node must store only another multiple of H_0 bounded by $\mathcal{O}(\mathcal{G})$, which requires $\mathcal{O}(\log(\mu D))$ bits. However, nodes may receive different estimates from different neighbors. If messages are sent every $\Theta(H_0)$ time, these values may drift apart at an (amortized) rate of 2ε for $\mathcal{O}(\mathcal{G})$ time, until the (local) estimates become exact, i.e., reach the current maximum estimate. Therefore, if these values are encoded relative to each other, $\mathcal{O}(\Delta \log(\varepsilon \mu D))$ additional bits are needed. The same is true if messages are also triggered upon receiving messages, but nodes control the rate at which they forward estimates by means of their hardware clocks. Setting the maximal amortized update rate to a constant factor times the hardware clock rate (i.e., at least $(1+\hat{\varepsilon})(1+\mu)/(1-\hat{\varepsilon}) \in \mathcal{O}(1)$) ensures that estimates are forwarded fast enough, but

[7] We assume that $\mu H_0 \in \Omega(1)$. Since $\kappa \in \Omega(1)$, there is no gain when sending messages more frequently.

[8] Note that by putting more information into the messages, the space complexity can be reduced.

locally drift apart at a rate of at most $\mathcal{O}(\varepsilon)$.

Finally, observe that $\tilde{H}_0$ does not need to be stored since it can be computed from the other variables. Hence, adding all terms up, we see that $\mathcal{A}^{\text{opt}}$ needs to allocate at most $\mathcal{O}(\log \mathcal{T} + \log(\mu D) + \Delta(\log(1/\mu) + \log(\varepsilon\mu D) + \log\log_{\mu/\varepsilon} D))$ bits of memory.[9]

7. DIFFERENT MODELS

In this section, we analyze the details of the model assumptions and point out to what extend the results carry over to other prominent models of clock synchronization.

7.1 Estimates of $\mathcal{T}$ and ε

Our model assumes that upper bounds $\hat{\mathcal{T}} \in \mathcal{O}(\mathcal{T})$ and $\hat{\varepsilon} < 1$ on $\mathcal{T}$ and ε are known. These assumptions can be justified as follows.

Assuming that $\mathcal{T}$ is completely unknown to the algorithm is no restriction. In this case, nodes acknowledge every message, and perpetually measure the corresponding round trip times by means of their hardware clocks. Multiplying these values by $1/(1-\hat{\varepsilon})$ then yields an estimate of the round trip times, which are in $\mathcal{O}(\mathcal{T})$ and which upper bound the delays of the messages. Nodes keep track of the largest estimate they either measured themselves or received in a message. If a larger (estimated) round trip time is detected, it is flooded through the network and κ (and possibly H_0) is adjusted accordingly. Note that it is not a problem if the nodes underestimate $\mathcal{T}$ because, until the time when larger delays actually occur, the skew bounds hold with respect to the smaller delays and thus the smaller κ. In order to keep the number of messages low, one could initially use an estimate of $\Theta(1)$ and double it in every step, reducing the number of updates to at most $\mathcal{O}(\log \mathcal{T})$.

As far as the assumption that ε is bounded by $\hat{\varepsilon} < 1$ is concerned, we point out that an ε arbitrarily close to one means that we do not have clocks in the truest sense of the word.[10] In particular, $\varepsilon = 0$ would allow the hardware clocks to stand still, a case in which nodes are not able to react to clock skews at all. In such a setting it would be reasonable to drop the constraint that the progress rates must be bounded at all times in favor of sudden clock jumps (i.e., $\mu = \infty$ and therefore still "large enough" to guarantee Inequality (4)). However, we do not cover this rather extreme scenario in our discussion.

7.2 Minimum clock rate α

As pointed out in Section 5, the lower bound on the local skew does not depend on clocks running always at least at speed α. Requiring an average rate of α merely for intervals of a certain minimum length does not change the asymptotic bounds, unless clocks are allowed to stand still (or even run backwards) for almost $D\mathcal{T}$ time. Moreover, the lower bound on the global skew of roughly $D\mathcal{T}$ trivially implies a bound of $\mathcal{T}$ on the local skew, i.e., even if we do not require any minimum progress rate, any algorithm guaranteeing Condition (3) will suffer a local skew of $\mathcal{T}$ in the worst case. Considering that typically $\mathcal{T}\log_{1/\varepsilon} D \in \mathcal{O}(\mathcal{T})$ and $\mathcal{A}^{\text{opt}}$ attains this bound on the local skew, there is little gain in choosing $\alpha < 1 - \mathcal{O}(\varepsilon)$.

[9] This notation is somewhat sloppy. To be formally correct, each summand has to be replaced by the maximum of the term itself and 1.

[10] A cheap quartz oscillator exhibits a relative drift of less than 10^{-4} and even the clock drift of a ring oscillator under varying temperature and support voltages is not considerably larger than 0.2.

7.3 Lower Bounded Delays

Throughout this paper, we assumed that delays are always in the range $[0, \mathcal{T}]$. In many distributed systems, it is more adequate to assume that all delays lie in a range $[\mathcal{T}_1, \mathcal{T}_2]$, where $\mathcal{T}_2 - \mathcal{T}_1 \ll \mathcal{T}_1$. It is evident from the proofs of the lower bounds that they still hold with $\mathcal{T}$ replaced by $\mathcal{T}_2 - \mathcal{T}_1$ in this situation. Similarly, the algorithm can be modified for this scenario by adding $\mathcal{T}_1$ to all values received. However, triggering messages when $L_v + \Lambda_v^{\max}$ reaches a multiple of H_0 in order to bound the (amortized) message frequency does not work any more. This can either be solved by simply sending messages every H_0 local time (as discussed in Section 6), or by enforcing one logical clock to be the fastest by slowing down all other clocks slightly and performing "external" synchronization where the distinguished node serves as the reference (cf. Section 7.4).

Another effect, however, might be of more concern: The skew bounds will degrade because the algorithm needs more time to react to clock skews. The global skew will increase by $\mathcal{O}(\varepsilon D\mathcal{T}_1)$, which is basically asymptotically optimal due to the fact that distant nodes may not receive messages from each other for $\Omega(D\mathcal{T}_1)$ time because of the slow information transport. As far as the local skew is concerned, $\mu \geq (\mathcal{T}_2 - \mathcal{T}_1)/\mathcal{T}_2$ will only improve the base σ of the logarithm further at the expense of increasing κ linearly with σ. In this case, the result of the (properly adapted) Theorem 5.4 is no longer matched by the lower bounds. However, if $\mathcal{T}_2 \leq \sqrt{\varepsilon}(\mathcal{T}_2 - \mathcal{T}_1)$, for instance, choosing $\mu \in \Theta(\sqrt{\varepsilon})$ and κ appropriately will still result in an asymptotically optimal local skew of $\mathcal{O}(\kappa \log_{1/\varepsilon} D)$.

7.4 External Clock Synchronization

There is a lot of work on *external* clock synchronization algorithms [18, 19, 21], where a source of real time is available and the objective is to synchronize all clocks to this source. Thus, there is a single node v_0 for which logical clock time, hardware clock time, and real time are identical. In order to allow for the fact that a distant node v may not be informed about the real time more accurately than $\mathcal{T}d(v, v_0)$, Condition (2) has to be changed to $t - d(v, v_0)\mathcal{T} - \tau \leq L_v(t) \leq t$, where $\tau \in \mathbb{R}^+$ addresses the issue that nodes should only send a finite number of messages in constant time.

$\mathcal{A}^{\text{opt}}$ can easily be adjusted to handle this modified constraint. The node v_0 has to propagate its clock value through the network periodically, at least every $\Theta(\tau/\hat{\varepsilon})$ time. The nodes behave the same way as in $\mathcal{A}^{\text{opt}}$, except that they reduce $\Lambda_v^{\max}$ by $1 - 1/(1+\hat{\varepsilon})$ at each tick if $\Lambda_v^{\max}$ is positive and increase L_v only by $1/(1+\hat{\varepsilon})$ whenever $\Lambda_v^{\max} = 0$. This technique ensures that the logical clock rates are upper bounded by 1 whenever the largest clock value in the system is attained, implying that $L_v(t) \leq t$ at all times. On the other hand, nodes still raise their clocks quickly when large estimates are received. Apparently, the global skew is bounded by $\mathcal{T}D + \mathcal{O}(\tau)$, and the worst-case clock skew between some node v and v_0 is linearly bounded in the distance between the two nodes. The main difference is that the minimum progress rate is now only bounded by $1 - \mathcal{O}(\hat{\varepsilon})$, which can easily be accounted for when determining μ and κ. Thus, the algorithm still guarantees roughly the same

skew bounds without significantly increasing μ or κ. The amortized message frequency is $\Theta\left(\tau/\hat{\varepsilon} + 1/H_0\right)$.

7.5 Hardware Clock Envelope Condition

A similar technique is applicable if Condition (2) is replaced by

$$\forall v \in V\ \forall \text{ticks } t \text{ at } v : \min_{w \in V}\{H_w(t)\} \leq L_v(t) \leq \max_{w \in V}\{H_w(t)\},$$

i.e., the time envelope condition is sharpened to the requirement that all logical clock values must always be at least the smallest and at most the largest hardware clock value in the network. In this case, a node $v \in V$ must reduce its clock rate when $L_v(t) > H_v(t)$, while still responding to clock skews. This is accomplished by increasing $L_v + \Lambda_v^{\max}$ at the reduced (amortized) rate $(1 - \hat{\varepsilon})h_v/(1 + \hat{\varepsilon})$ whenever it exceeds H_v and again slowing down L_v to avoid that $\Lambda_v^{\max}$ becomes negative whenever $\Lambda_v^{\max} = 0$. Thus, nodes will never have a larger logical clock rate than $1 - \varepsilon$ if $L_v(t) = L_v^{\max}(t) = \max_{w \in V}\{H_w(t)\} > H_v(t)$. As they also increase their logical clocks at the normal rate when $L_v(t) = H_v(t)$, the requested constraint is satisfied. Clock rates change merely by a factor of $1 - \mathcal{O}(\hat{\varepsilon})$, therefore the bounds on κ and μ, and the impact of H_0 remain basically the same as in Section 7.4.

7.6 Network Dynamics

The distributed system may be dynamic in the sense that new nodes may join and leave the system. Therefore, a clock synchronization algorithm must also be able to cope with node arrivals and departures. While the latter is trivial if the nodes notify the system before leaving, this task becomes more complicated if nodes (or links) crash.

Apparently, a joining node, or a newly established link, may in the worst case lead to skews in the order of $\mathcal{G}$ between nodes, which cannot be reduced faster than in time $\Omega\left(\mathcal{G}/\beta\right)$. However, this bound can be achieved by separating the problem of node integration from the running algorithm: The skew ΔL on a newly created edge can be distributed over the network without considering it in the execution of $\mathcal{A}^{\text{opt}}$, i.e., the nodes do not increase their clocks due to this clock skew. Thus, the skew will eventually be propagated to nodes with small clock skews to all neighbors, which act as "sinks", ensuring that the separately handled skew will be removed from the system after $\mathcal{O}\left(D\mathcal{T} + \Delta L/\beta\right)$ time. This can be achieved by using the simple algorithm that always increases its clock value as much as possible provided that no neighbor is more than κ behind [12, 13]. Obviously, this technique will not affect the asymptotic bounds. However, since β might be small, waiting for up to $\Omega\left(\mathcal{G}/\beta\right)$ time to integrate nodes may not be desirable. The bounds on the clock rates and the tolerated skew could be relaxed in an adaptive manner when nodes arrive in order to speed up the integration process. We dispense with the analysis of such an adaptive node integration scheme.

Finally, we briefly discuss how the algorithm can be adapted in order to cope with node or edge failures. Since the algorithm cannot distinguish between node and edge failures, it suffices to consider edge failures. We assume for the sake of simplicity that a failed edge does not reappear. The proposed strategy relies on the observation that the algorithm can dynamically determine an estimate $\hat{\mathcal{T}} \in \mathcal{O}(\mathcal{T})$ of the maximum delay. Every node v stores the times when it last received a message from each of its neighbors. It is possible that a message from a neighbor $w \in \mathcal{N}_v$ is "overdue" according to v's hardware clock, i.e., more than $(1 + \hat{\varepsilon})(\hat{\mathcal{T}} + H_0/(1 - \hat{\varepsilon}))$ time has passed since v received a message from w. This situation can occur for one of two reasons, either w crashed or the delay is larger than estimated. In both cases, v (temporarily) removes w from its set of neighbors and adjusts its logical clock rate if necessary. If v receives the next message from w at a later point in time, w is added to the set $\mathcal{N}_v$ again, $\hat{\mathcal{T}}$ and κ are updated, and the logical clock rate is reevaluated. Subsequently, the new estimate of the maximum delay is flooded through the network. It is possible that the exclusion of w mistakenly causes v to increase its logical clock rate because w was the only neighbor that prevented v from setting its logical clock rate multiplier to $1 + \mu$. However, if v considers the edge $\{v, w\}$ to have failed for ΔT time, it increases its logical clock by at most $\Delta L = (1 + \varepsilon)(1 + \mu)\Delta T - (1 - \varepsilon)\Delta T = 2\varepsilon\Delta T + (1 + \varepsilon)\mu\Delta T$ more than the minimum progress in this time interval, i.e., the clock skew to any neighbor grows by at most ΔL. The bound on the local skew relies on the fact that the average clock skew of a path of a given length can only be a specific multiple of κ. Consider any path containing v with a certain average clock skew $\Lambda \geq \kappa$. According to Condition (5), the new estimate of $\mathcal{T}$ causes κ to increase by $\Delta\kappa > 2(1+\mu)\Delta T > (2\varepsilon+(1+\varepsilon)\mu)\Delta T = \Delta L$, which implies that $(\Lambda + \Delta L)/(\kappa + \Delta\kappa) < \Lambda/\kappa$.

Thus, the average clock skew is at most the same multiple of κ with respect to the new value of κ. The same argument holds if v no longer increases its logical clock at the rate $1 + \mu$ because the node u that maximizes $\Lambda_v^{\uparrow}$ does not respond within the expected amount of time. We conclude that temporarily excluding nodes from the set of neighbors because they do not respond in time does not have an impact on the clock skew bounds with respect to the increased κ, and thus $\mathcal{A}^{\text{opt}}$ can also be used in an environment where edges (or nodes) fail. Apparently, this simple strategy cannot cope with temporary failures, since a long-term malfunction of a communication link would cause the nodes to set $\hat{\mathcal{T}}$ to an exceedingly large value. This problem can be tackled if the nodes have some means to detect that they, or the links between them, must have been inoperative so that their recurrence does not cause the nodes to increase their estimate of the maximum delay.

8. CONCLUSION

We studied the well-known clock synchronization problem for arbitrary underlying network topologies. As our main result, we presented the synchronization algorithm $\mathcal{A}^{\text{opt}}$ and proved matching upper and lower bounds on the worst-case clock skew between neighboring nodes and between arbitrary nodes in the network. Remarkably, these results hold in a very general model where clock drifts and delays may vary arbitrarily within unknown bounds. Surprisingly, this can also be achieved if the logical clock rates must always be in the range $[1-\mathcal{O}(\varepsilon), 1+\mathcal{O}(\varepsilon)]$ and we require that $\Omega(\mathcal{T}/\varepsilon)$ time elapses between message transmissions at each node, where ε and $\mathcal{T}$ denote the maximum clock drift rate and the maximum delay, respectively. Moreover, the bound on the global skew is essentially optimal and, if upper bounds on both the clock drift and the maximum delay are known fairly accurately, the asymptotic approximation ratio of the proposed algorithm with regard to the local skew is 2. The algorithm $\mathcal{A}^{\text{opt}}$ can further be adjusted in order to be appli-

cable to various other models and constraints, which demonstrates the generality and flexibility of our techniques.

Our results may be relevant for practical applications for the following reasons. We showed that although the local skew must grow logarithmically in the diameter D of the network, the base of the logarithm can be bounded by $\Theta(1/\varepsilon)$. Thus, if we have that $D \in \mathcal{O}(1/\varepsilon^c)$ for a constant c, a worst-case clock skew of $\mathcal{O}(\mathcal{T})$ betweeen neighboring nodes can be guaranteed. Since typical clock drifts are in the order of 10^{-5} and the diameters of most current networks are not larger than roughly 20 to 30, the clock skew between neighboring nodes can be bounded by $\mathcal{O}(\mathcal{T})$ in most real-world systems. Furthermore, it is clearly desirable that the clocks run smoothly and progress at reasonable rates at all times. It is possible to achieve such a strong bound even if we impose tight restrictions on the rates of the clocks, i.e., the clocks are never allowed to change abruptly in order to correct the observed clock skews. In particular, rates of $1 \pm \mathcal{O}(\sqrt{\varepsilon})$ are sufficient to guarantee an optimal global skew and a roughly 4-competitive local skew even is $\mathcal{T}$ is initially unknown. Given that $\mathcal{A}^{\text{opt}}$ is further computationally efficient and that it requires a low message frequency, we believe that pratical protocols can be designed that guarantee not only small global skews at low communication costs, but also small local skews and smoothly progressing logical clocks. We hope that some of the ideas introduced in this work might lead to advances in the field of applied synchronization protocols.

Acknowledgments

We would like to thank Fabian Kuhn for valuable comments.

9. REFERENCES

[1] B. Awerbuch. Complexity of Network Synchronization. *Journal of the ACM*, 32(4):804–823, 1985.

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

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

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

[5] M. Fuegger, U. Schmid, G. Fuchs, and G. Kempf. Fault-Tolerant Distributed Clock Generation in VLSI Systems-on-Chip. In *Proceedings of the Sixth European Dependable Computing Conference (EDCC-6)*, pages 87–96, 2006.

[6] S. Ganeriwal, R. Kumar, and M. B. Srivastava. Timing-Sync Protocol for Sensor Networks. In *Proc. 1st international Conference on Embedded Networked Sensor Systems*, pages 138–149, 2003.

[7] B. Korte, D. Rautenbach, and J. Vygen. BonnTools: Mathematical Innovation for Layout and Timing Closure of Systems on a Chip. In *Proceedings of the IEEE 95*, pages 555–572, 2007.

[8] 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)*, 2009.

[9] 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.

[10] C. Lenzen, T. Locher, and R. Wattenhofer. Optimal Clock Synchronization with Bounded Rates. Technical Report 301, ETH Zurich, 2009. ftp.tik.ee.ethz.ch/pub/publications/TIK-Report-301.pdf.

[11] C. Lenzen, T. Locher, and R. Wattenhofer. Tight Lower Bounds for Clock Synchronization. Technical Report 299, ETH Zurich, 2009. ftp.tik.ee.ethz.ch/pub/publications/TIK-Report-299.pdf.

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

[15] M. Maróti, B. K. G. Simon, and Á. Lédeczi. The Flooding Time Synchronization Protocol. In *Proc. 2nd International Conference on Embedded Networked Sensor Systems*, pages 39–49, 2004.

[16] L. Meier and L. Thiele. Brief Announcement: Gradient Clock Synchronization in Sensor Networks. In *Proc. 24th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, page 238, 2005.

[17] D. Mills. Internet Time Synchronization: the Network Time Protocol. *IEEE Transactions on Communications*, 39:1482–1493, 1991.

[18] Y. Moses and B. Bloom. Knowledge, Timed Precedence and Clocks. In *Proc. 13th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 294–303, 1994.

[19] 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.

[20] S. PalChaudhuri, A. K. Saha, and D. B. Johnson. Adaptive Clock Synchronization in Sensor Networks. In *Proc. 3rd International Symposium on Information Processing in Sensor Networks*, pages 340–348, 2004.

[21] 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.

[22] P. Sommer and R. Wattenhofer. Gradient Clock Synchronization in Wireless Sensor Networks. In *Proc. 8th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN), San Francisco, USA*, April 2009.

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