

# Oblivious Gradient Clock Synchronization

Thomas Locher and Roger Wattenhofer

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

**Abstract.** We study the *gradient clock synchronization* (GCS) problem, in which the worst-case clock skew between neighboring nodes has to be minimized. In particular, we consider *oblivious* clock synchronization algorithms which base their decision on how to adapt the clock solely on the most accurate timing information received from each neighbor. For several intuitive clock synchronization algorithms, which attempt to minimize the skew at all times, we show that the clock skew between neighboring nodes can be significantly larger than the proven lower bound of  $\Omega(\frac{\log D}{\log \log D})$ , where  $D$  denotes the diameter of the network. All of these natural algorithms belong to the class of oblivious clock synchronization algorithms. Additionally, we present an oblivious algorithm with a worst-case skew of  $O(d + \sqrt{D})$  between any two nodes at distance  $d$ .

**Key words:** Distributed algorithms, synchronization protocols, asynchronous computation.

## 1 Introduction

Due to the rapidly growing popularity of distributed systems, such as the Internet or wireless networks, and the sizable amount of applications running on those systems, the classical problem of synchronizing distributed clocks has further gained in importance in the last few years. The objective of a distributed clock synchronization algorithm is to ensure that all participating nodes in the system acquire a common notion of time. In a distributed system, nodes can accomplish this goal by perpetually sending messages containing information about the current clock value to the neighboring nodes.

Nodes are equipped with a hardware clock with bounded drift. According to the current hardware clock value and the messages received from all neighboring nodes, a logical clock value is computed. The skew between the logical clocks is to be minimized. Previous work has focused primarily on minimizing the clock skew between any two nodes in the system, while inducing a moderate message overhead. Hence, the goal of past work was to ensure that clocks are well synchronized *globally*. The resilience of these algorithms in the presence of node and network failures is another aspect of distributed clock synchronization that has been studied extensively.

For several distributed applications, such as TDMA, it is mandatory that the clocks between any node and all nodes in its vicinity do not deviate considerably

from each other. This so called *gradient* property for clock synchronization was introduced in [2]. The gradient property requires that nodes that are close by have to be closely synchronized, whereas the skew between clocks of faraway nodes is allowed to be larger.

The main question is what bound on the skew between nearby nodes can be achieved by any clock synchronization algorithm. It can be shown that the skew between two nodes at distance  $d$  cannot be synchronized better than  $\Omega(d)$  by using simple indistinguishability type arguments. Surprisingly, the skew between two neighboring nodes, i.e. nodes at distance 1, cannot be guaranteed to be constant. The lower bound on the worst-case clock skew proven in [2] is  $\Omega(\frac{\log D}{\log \log D})$ , where  $D$  is the diameter of the network.

This lower bound holds even if all nodes have full knowledge of the complete message history. However, for practical algorithms it is reasonable to assume that nodes cannot store the entire history of messages ever received. As time progresses, nodes will be forced to delete outdated information. We study clock synchronization in a restricted model in which each node is only allowed to store the largest clock value ever received from each neighbor. It is natural to restrict the stored information to these values because any algorithm attempting to minimize the skew at all times will set the clock in accordance with these clock values only, due to the lack of information about the message delays and the progress each node might have made in the meantime. Since these algorithms are unaware of the communication process and determine the local clock strictly by considering the largest clock values received from all neighboring nodes, we call these algorithms *oblivious*. Studying oblivious algorithms has a long tradition in distributed computing and computer science in general. Oblivious algorithms are examined for various reasons, one being that they can give valuable insights into problems that are hard to tackle in the general case. A more practical reason is that they are normally easier to realize in hardware. Another motivation to explore oblivious algorithms is that several oblivious algorithms perform well in their respective domains, for example routing or sorting algorithms. It is therefore worthwhile to determine the effect of obliviousness on clock synchronization.

Several fundamentally different strategies can be employed in order to minimize the skew at all times. As nodes must generally strive to catch up with the faster nodes, nodes can minimize the skew to the fastest node<sup>1</sup> by setting the clock to the largest clock value. A different approach is to minimize the skew to all neighbors at all times. A third method that is worth investigating is minimizing the skew to the slowest node. If every node waits under certain circumstances for the slower nodes to catch up, the skew might be kept within reasonable bounds. We will study algorithms devoted to each of these objectives and show that all of them fail to provide a low bound on the worst-case skew between neighbors.

However, these observations enable us to devise an oblivious algorithm with a worst-case skew of  $O(d + \sqrt{D})$  between any two nodes at distance  $d$ , if the nodes

---

<sup>1</sup> The nodes with the currently largest and smallest logical clock values are called the fastest and the slowest node, respectively.

are aware of the diameter  $D$  of the network and adjust their clock synchronization mechanism accordingly. To the best of our knowledge, it is the first gradient clock synchronization algorithm guaranteeing a worst-case skew of  $o(D)$  between neighboring nodes.

After briefly summarizing related work on clock synchronization in Section 2, we formally specify the model used in this paper in Section 3. Subsequently, we propose gradient clock synchronization algorithms which minimize the skew to the neighboring nodes according to the aforementioned strategies and analyze their behavior in specific executions. This is the subject of Section 4. In Section 5, the  $O(d + \sqrt{D})$ -GCS algorithm is presented and analyzed.

## 2 Related Work

The fundamental problem of clock synchronization has been studied extensively and many theoretical results have been published. Srikanth and Toueg [7] presented a clock synchronization algorithm which minimizes the maximum skew between any pair of nodes, given the hardware clock drift. In every possible execution, their algorithm ensures that the skew between any two nodes is at most  $\Theta(D)$ , which is asymptotically optimal. However, their algorithm also incurs a skew of  $\Theta(D)$  between neighboring nodes in the worst case.

While there has been a lot of research on bounds for the skew and the communication costs [3, 5, 6] and also on the capability of clock synchronization algorithms to cope with both node and network failures [7], the gradient property has not been studied until the remarkable work by Fan and Lynch [2].

An algorithm  $\mathcal{A}$  is said to be an  $f$ -GCS algorithm, if for all nodes  $i$  and  $j$  the clock skew between node  $i$  and  $j$  is at most  $f(d_{i,j})$  at all times, where  $d_{i,j}$  denotes the *distance* between node  $i$  and  $j$ .

Fan and Lynch prove that there is an execution after which the skew between two neighboring nodes is at least  $\Omega(\frac{\log D}{\log \log D})$ , independent of the chosen algorithm. They consider a linear network of  $n$  nodes, thus  $D = n - 1$ . Their proof relies on the fact that a node cannot increase its clock too quickly, i.e. by more than  $O(f(1))$  in  $O(1)$  time, otherwise it would violate the gradient property in a different execution that is indistinguishable from the original execution. The skew between all neighbors among  $k$  nodes can be increased by  $O(1)$  in  $O(k)$  time, which can be shown using again an indistinguishability type argument. In their execution, they build up a constant skew  $c_1$  in time  $O(n)$  between all  $n$  nodes. In the next step, the execution continues to run for  $O(\frac{n}{f(1)})$  time, which means that the average skew during this time can only be reduced by a constant  $c_2$  between each pair of neighboring nodes. The parameters can be chosen such that  $c_1 > c_2$  and thus the skew is still larger than before this round. During the same time span, the skew between neighbors of a set of  $O(\frac{n}{f(1)})$  nodes can again be increased by a constant. This procedure can be repeated recursively  $\log_{f(1)} n$  times and since  $n = D - 1$  and  $f(1) \in \Omega(\log_{f(1)} n)$ , it follows that  $f(1) \in \Omega(\frac{\log D}{\log \log D})$ . Meier and Thiele [4] showed that this bound also holds for a

different model in which the delay of each message is 0, but the communication frequency is bounded.

An open problem for gradient clock synchronization is whether this lower bound is *tight* or whether an algorithm cannot even reach this bound asymptotically. We will show that many natural clock synchronization algorithms do not even come close to this bound. In the context of clock synchronization for wireless networks, Fan et al. show that when nodes occasionally receive a message informing them of the correct real time using a GPS service, the skew between any two nodes can be bounded by a small constant  $\epsilon > 0$  “some of the time” [1]. Thus, this algorithm only satisfies a weakened version of the gradient property. We show that there is a general gradient clock synchronization algorithm for which it holds that the skew between nodes at distance  $d$  is bounded by  $O(d + \sqrt{D})$  at all times. This is the first non-trivial upper bound for gradient clock synchronization.

### 3 Model

We consider an arbitrary graph  $G = (V, E)$ ,  $V = \{1, \dots, n\}$ , where  $|V| = n$  and  $E \subseteq V \times V$ . Any node  $i$  can communicate with any node  $j$  to which node  $i$  is directly connected, i.e.  $\{i, j\} \in E$ . These nodes are referred to as the neighboring nodes or neighbors of node  $i$ . Let  $\mathcal{N}_i$  denote the set of all neighboring nodes of node  $i$ . The communication between neighboring nodes is reliable, but messages can have variable delays in the range  $[0, 1]$ . The *distance* between nodes  $i$  and  $j$  is defined as the length of the shortest path between those two nodes. The *diameter*  $D$  of  $G$  is the maximum distance between any two nodes.

Each node  $i \in V$  is equipped with a *hardware clock*  $H_i(\cdot)$  whose value at time  $t$  is  $H_i(t) := \int_0^t h_i(\tau) d\tau$ , where  $h_i(\tau)$  is the *hardware clock rate* of node  $i$  at time  $\tau$ . For all nodes  $i \in V$  and all times  $t$ , it holds that  $h_i(t) \in [\mathcal{L}, \mathcal{U}]$ , where  $0 < \mathcal{L} < \mathcal{U}$ . The degree of synchronization that can be achieved is related to the maximum message delay. It is therefore reasonable to assume that a fast processor can increase its clock by at least 1, if it takes up to 1 time before a message arrives, therefore we assume  $\mathcal{U} \geq 1$ .

In addition to the hardware clock, each node  $i$  further has a *logical clock*  $L_i(\cdot)$ . As long as no new messages arrive, the logical clock value increases at the rate of the hardware clock. This implies that all nodes steadily make progress. A clock synchronization algorithm  $\mathcal{A} : L \times \Psi \rightarrow L$  specifies how the logical clock  $L_i(t)$  of node  $i$  at time  $t$  is adapted according to the current value of the logical clock and the message history  $\Psi_i(t)$  of node  $i$  at time  $t$ . The algorithms are *reactive* in that they perform this update on the logical clock whenever a message from a neighboring node arrives. Let  $\tilde{L}_j(t)$  denote the maximum logical clock value ever received from neighboring node  $j$ .<sup>2</sup> For every algorithm  $\mathcal{A}$  it

<sup>2</sup> Note that this might not be the *last* message received from node  $j$ , as the communication network does not necessarily satisfy the FIFO condition.

must hold that

$$\forall i \forall t : L_i(t) \leq \mathcal{A}(L_i(t), \Psi_i(t)) \leq \max_{j \in \mathcal{N}_i} \tilde{L}_j(t).$$

As the logical clock is not allowed to run backwards, the algorithm can either increase the logical clock or leave it at the current value. Moreover, the algorithm can set the logical clock at most to the maximum logical clock value it has ever received. If a node  $i$  set its logical clock to a value exceeding any value it has ever received from a neighbor, a neighboring node  $j$  could potentially increase its logical clock value even more, based on the new clock value of node  $i$  etc. resulting in a large clock skew between some nodes. We further assume that the adaptation of the logical clock through  $\mathcal{A}$  requires 0 time.

As mentioned before, we focus on the class of oblivious clock synchronization algorithms. Nodes only store the reduced history  $\tilde{\Psi}_i(t) := \{\tilde{L}_j(t)\}^{j \in \mathcal{N}_i}$ , i.e. the largest clock values received from each neighboring node is stored. Whenever a message is received, these values are updated and, subsequently, the logical clock is computed according to a function on  $\tilde{\Psi}_i(t)$ . Naturally, the old logical clock value also has to be considered, since clocks can only make progress. In case the computed value does not exceed the old logical clock value, the logical clock simply remains unchanged, otherwise the logical clock is updated and the new logical clock value is broadcast immediately to all neighbors.

An *execution* is a tuple  $\mathcal{E} = (\mathcal{M}, \mathcal{R})$ , where  $\mathcal{M} : \mathcal{T} \times V \times V \rightarrow [0, 1]$  defines the message delays and the integrable function  $\mathcal{R} : \mathcal{T} \times V \rightarrow [\mathcal{L}, \mathcal{U}]$  determines the hardware clock rates of each node. Hence  $\mathcal{M}(t, i, j)$  specifies how long it takes for the message sent by node  $i$  at *real time*  $t$  to arrive at  $j$  and  $h_i(t) := \mathcal{R}(t, i)$ .

For any gradient clock synchronization algorithm, the goal is to ensure a small logical clock skew between neighboring nodes, i.e. a gradient clock synchronization algorithm strives to minimize  $\max_{i, j \in \mathcal{N}_i, t} |L_i(t) - L_j(t)|$  over all possible executions  $\mathcal{E}$  for every graph  $G$ . In the following section, we present natural clock synchronization algorithms and bounds on the induced worst-case skew.

## 4 Algorithms and Bounds

Throughout this section, we consider the graph  $G^{list}$  consisting of a linear list of  $n$  nodes, i.e.  $|V| = n$  and  $E^{list} = \{\{1, 2\}, \{2, 3\}, \dots, \{n-1, n\}\}$ . This simple graph is suitable to show that there are executions for all presented algorithms leading to a large skew between neighboring nodes.

Initially, all logical clock values are 0. We assume that, at real time 0, node  $n$  sends a *start message* to its neighbor and starts its logical clock. Every node that receives a start message for the first time also starts its clock and broadcasts the start message. For the purpose of synchronization, each node regularly informs all neighbors about its current logical clock value. In particular, we assume that every node transmits a message containing its ID and its logical clock value to all neighboring nodes when its logical clock reaches an integer value or when the

logical clock is updated due to a received message. If a node has not yet received a message from a certain neighbor, it assumes that this neighbor's logical clock is still at 0.

Note that this initialization process is used for the sake of simplicity. The same bounds also hold asymptotically for other start-up procedures. By proving that the proposed algorithms incur a large skew among neighboring nodes in  $G^{list}$  using the stated initialization process, we can conclude that they are poor gradient clock synchronization algorithms in general.

#### 4.1 Minimizing the Skew to the Fastest Neighbor

A straightforward algorithm, denoted  $\mathcal{A}^{max}$ , always sets the logical clock to the largest clock value ever received, if this value exceeds the current logical clock value. More formally, the logical clock value of node  $i$  is set to the value  $\max(L_i(t), \max_{j \in \mathcal{N}_i} \tilde{L}_j(t))$ , if it receives a message at time  $t$ . Thus, the skew to the fastest node is minimized by simply adopting the maximum clock value. It has been pointed out in [2] that  $\mathcal{A}^{max}$  potentially incurs a large skew between neighboring nodes, due to the fact that a skew of  $\Theta(n)$  between node 1 and  $n$  cannot be avoided and a *fast message*, i.e. a message that is transmitted with 0 delay, which is forwarded along the chain causes a skew of  $\Theta(n)$  between two neighboring nodes. We will now briefly dwell on this simple algorithm in order to introduce our notation. The following execution  $\mathcal{E} = (\mathcal{M}, \mathcal{R})$  induces a skew of  $\Theta(n) = \Theta(D)$  between the nodes 1 and 2:

$$\mathcal{M}(t, i, j) := \begin{cases} 0 & \text{if } t \geq n - 1, j \neq 1 \\ 1 & \text{else} \end{cases}$$

and  $\forall t \forall i : \mathcal{R}(t, i) := 1 - \epsilon_i$ , where  $\epsilon_n = 0$  and  $\epsilon_i > 0$  for all  $i \in \{1, \dots, n - 1\}$ .<sup>3</sup> It holds that  $L_j(n - 1) = n - 1$  for all  $j \in \{2, \dots, n\}$ , as the logical clock of node  $n$  has reached  $n - 1$  and this value is forwarded along the chain with a delay of 0. Since node 1 receives the start message at time  $n - 1$ , its logical clock is still at 0 and thus the skew between node 1 and 2 is  $\Theta(n)$ . Note that this effect does not occur due to the fact that node 1 has merely received its start message. If there was no fast message, the logical clock of node  $i$  at time  $t \in \mathbb{N}$ , where  $t > n - 1$ , would be  $L_i(t) = t - (n - i)$ . Setting the message delay to 0, except between nodes 1 and 2, at this point in time would still incur a skew of  $\Theta(n)$  between nodes 1 and 2.

Before the fast message is sent, the clock value of node  $i$  at time  $n - 1$  is  $i - 1$ . If each node allowed a slack of 1 between the clock of the fastest node and its own, the fast message would not alter any clock values. By setting the logical clock to  $\max(L_i(t), \max_{j \in \mathcal{N}_i} \tilde{L}_j(t) - \gamma)$  for a particular  $\gamma > 0$ , it seems that the effect a fast message has in  $\mathcal{A}^{max}$  can be avoided. Unfortunately, this is not the case. Let  $\mathcal{R}(t, n) := \mathcal{U}$  and  $\mathcal{R}(t, i) := \mathcal{L}$  for all  $i \neq n$ . The message delays are

<sup>3</sup> Node  $n$  is the fastest node and therefore sets the pace for the other nodes. The clock rates can be viewed as *relative* rates compared to the clock rate of the fastest node.

$\mathcal{M}(t, i, j) := 0$  for all  $i, j \neq 1$  and  $t \geq \vartheta$  for a specific time  $\vartheta$ , and  $\mathcal{M}(t, i, j) := 1$  otherwise. In this scenario, it holds that  $\lim_{t \rightarrow \infty} L_i(t) - L_i(t-1) = \mathcal{U}$ , as all nodes are paced by the fastest node. If node  $i$  receives the value  $x$  from node  $i+1$ ,  $i$  sets its logical clock to  $x - \gamma$ . In this time, node  $i+1$  has increased its clock by  $\mathcal{U}$ , therefore  $\lim_{t \rightarrow \infty} L_{i+1}(t) - L_i(t) = \mathcal{U} + \gamma$  for all  $i \in \{1, \dots, n-1\}$ . Assume that at time  $\vartheta$ , this stabilization has occurred and that  $L_n(\vartheta) = x$ . The message delay at this time is reduced to 0 and thus node  $n-1$  can increase its clock to  $x - \gamma$ . The skew between  $n-1$  and  $n-2$  is then  $2\mathcal{U} + \gamma$ , therefore node  $n-2$  can increase its clock by  $2\mathcal{U}$ . In general, node  $n-i$  will increase its logical clock by  $i\mathcal{U}$  and thus the skew between node 1 and 2 at time  $\vartheta$  is  $(n-2)\mathcal{U} \in \Theta(n)$ . Hence, this variation of  $\mathcal{A}^{max}$  does not reduce the worst-case skew between neighboring nodes asymptotically.

As it is not an effective strategy to strictly minimize the skew to the fastest node, we will analyze the effect of taking the values  $\tilde{L}_j(t)$  from all neighbors  $j$  into account.

## 4.2 Minimizing the Skew to All Neighbors

We will now consider the algorithm that sets the logical clock to the *average value* of all the neighbors' clock values in an attempt to minimize the clock skew to all neighbors at all times, i.e. node  $i$  sets its logical clock to the value

$$L_i(t) := \max(L_i(t), \frac{1}{|\mathcal{N}_i|} \sum_{j \in \mathcal{N}_i} \tilde{L}_j(t))$$

upon receiving a message from a neighbor at time  $t$ . We call this algorithm  $\mathcal{A}^{avg}$ .

In a very simple execution, the skew in  $G^{list}$  can become very large. The execution  $\mathcal{E}^{nice} = (\mathcal{M}^{nice}, \mathcal{R}^{nice})$  is defined as follows.  $\forall t \forall i \forall j : \mathcal{M}^{nice}(t, i, j) := 1$  and  $\forall t \forall i : \mathcal{R}^{nice}(t, i) = 1 - \epsilon_i$ , where again  $\epsilon_n = 0$  and  $\epsilon_i > 0$  for all  $i \in \{1, \dots, n-1\}$ . Since the hardware clock rates never change, the message delays are the same at any point in time and identical between any two neighboring nodes, one might assume that the skew between neighbors cannot become exceedingly large. Surprisingly, this is not true, as we will prove now.

**Lemma 1.** *Let  $\mathcal{A}^{avg}$  be the clock synchronization algorithm in use. When executing  $\mathcal{E}^{nice}$ , it holds that  $\forall t \forall i \in \{2, \dots, n\} : L_i(t) - L_{i-1}(t) \leq 2i - 3$ , independent of the choices of  $\epsilon_i > 0$ .*

PROOF. First, we define  $\Delta L_i(t) := L_i(t) - L_i(t-1)$ . It holds that  $\forall i \forall t : \Delta L_i(t) \leq 1$ , as the average speed is upper bounded by the maximum hardware clock rate, which is 1 in this particular execution. It immediately follows that  $L_i(t+k) \leq L_i(t) + k$ .

We have that  $L_1(t) = L_2(t-1)$ , as node 2 is the only neighbor of node 1. If node 1 is informed about a higher value, it can increase its logical clock immediately to this value. Since  $\Delta L_2(t) \leq 1$  for all  $t$ , it follows that  $L_2(t) - L_1(t) \leq 1$  for all  $t$ . Assume that it holds for all  $t$  and all  $j \leq i$  that  $L_j(t) -$

$L_{j-1}(t) \leq 2j - 3$ . We will now prove a bound on the skew between node  $i$  and  $i + 1$ . For  $t = 0$ , it is trivially true that  $L_{i+1}(t) - L_i(t) \leq 2(i + 1) - 3$ . Assume that it holds for all  $t' \leq t$ . For  $t + 1$ , we have that

$$\begin{aligned} L_i(t + 1) &\geq \frac{L_{i+1}(t) + L_{i-1}(t)}{2} \\ &\geq \frac{L_{i+1}(t) + L_i(t) - (2i - 3)}{2} \\ &\geq \frac{L_{i+1}(t) + L_i(t + 1) - 1 - (2i - 3)}{2} \\ &\geq L_{i+1}(t + 1) - (2(i + 1) - 3). \end{aligned}$$

The first inequality holds because the logical clock value is always at least the average value of its neighbors. The second inequality follows by induction and the third and fourth inequalities hold because  $\Delta L_i(t) \leq 1$ .  $\square$

Lemma 1 shows that the skew between any two nodes is bounded, when executing  $\mathcal{E}^{nice}$ . In order to prove that the skew can in fact become large, we need another lemma.

**Lemma 2.**  $\forall i \in \{1, \dots, n\} : \lim_{t \rightarrow \infty} \Delta L_i(t) = 1$ .

PROOF. Assume that  $\Delta L_{n-1}(t)$  does not converge to 1. In this case, either there is an  $\epsilon > 0$  such that for all  $t$  it holds that  $\Delta L_{n-1}(t) \leq 1 - \epsilon$  or  $\Delta L_{n-1}(t) = 1$  only for some  $t$ . By definition of  $\mathcal{E}^{nice}$ ,  $\Delta L_n(t)$  is always 1. If there is such an  $\epsilon > 0$ , this would imply that  $\lim_{t \rightarrow \infty} L_n(t) - L_{n-1}(t) = \infty$ , which is a contradiction to Lemma 1. If for some  $t$  we have  $\Delta L_{n-1}(t) = 1$ , but the value never converges to 1, there is an unbounded number of times  $t'$  where  $\Delta L_{n-1}(t') < 1$ , which also implies that  $\lim_{t \rightarrow \infty} L_n(t) - L_{n-1}(t) = \infty$ , again a contradiction. Hence,  $\lim_{t \rightarrow \infty} \Delta L_{n-1}(t) = 1$  and, applying the same argument to the other nodes, it follows inductively that  $\lim_{t \rightarrow \infty} \Delta L_i(t) = 1$  for all nodes  $i \in \{1, \dots, n\}$ .  $\square$

We are now in the position to prove the following theorem.

**Theorem 1.** *Let  $\mathcal{A}^{avg}$  be the clock synchronization algorithm in use. When executing  $\mathcal{E}^{nice}$ , the largest skew between neighbors in  $G^{list}$  is  $2n - 3 \in \Theta(n)$ .*

PROOF. In particular, we show that  $\lim_{t \rightarrow \infty} L_i(t) - L_{i-1}(t) = 2i - 3$  for all  $i \in \{2, \dots, n\}$ . Since  $L_1(t) = L_2(t - 1)$ , it holds that  $\lim_{t \rightarrow \infty} L_2(t) - L_1(t) = \lim_{t \rightarrow \infty} \Delta L_1(t + 1) = 1$ , according to Lemma 2. We assume now that  $\lim_{t \rightarrow \infty} L_j(t) - L_{j-1}(t) = 2j - 3$  for all  $j \leq i$ . Lemma 4.2 states that  $\lim_{t \rightarrow \infty} L_{i+1}(t) - L_i(t) = \mathcal{Q}$  for a constant  $\mathcal{Q}$  which is upper bounded by  $2(i + 1) - 3$ , due to Lemma 1. If  $\mathcal{Q} < 2(i + 1) - 3$ , we get that

$$\begin{aligned} \lim_{t \rightarrow \infty} L_i(t) &= \lim_{t \rightarrow \infty} \frac{L_{i-1}(t - 1) + L_{i+1}(t - 1)}{2} \\ &= \lim_{t \rightarrow \infty} \frac{2L_i(t - 1) - (2i - 3) + \mathcal{Q}}{2} \end{aligned}$$

and thus  $\lim_{t \rightarrow \infty} \Delta L_i(t) < 1$ , a contradiction to Lemma 2.  $\square$

Note that the skew between node 1 and  $n$  is  $\Theta(n^2)$ , which is worse than the tight upper bound of  $\Theta(n)$  skew between any two nodes for  $\mathcal{A}^{max}$ . This execution shows that the neighboring node with the fastest clock must have a weight larger than the weight of all other neighbors together. To see this, consider a  $k$ -ary tree where the root is the fastest node. The skew between neighboring nodes will also be large when executing  $\mathcal{E}^{nice}$  if all  $k$  children together have a weight that is at least the weight of the parent node, as this is equivalent to performing this execution on the linear list where the weight of the higher indexed node, i.e. the faster node, is not larger than the weight of the lower indexed node. We proved that in this case the skew is  $\Theta(D)$  between neighboring nodes.

### 4.3 Minimizing the Skew to the Slowest Neighbor

In this section, we present a different approach which actively tries to bound the skew between neighbors. As proven in [2], a constant bound between neighboring nodes cannot be maintained. However, introducing a constant bound might still result in a significantly improved worst-case behavior.

The algorithm  $\mathcal{A}^{bound}$  increases the logical clock proactively to the maximum value of all neighbors as long as the clock skew between its own logical clock and the clock of any of its neighbors does not exceed a predefined constant bound  $\mathcal{B}$ . Once the skew between node  $i$  and a neighbor  $j$  is at least  $\mathcal{B}$  according to the state information about node  $j$ , i.e.  $L_i(t) - \tilde{L}_j(t) \geq \mathcal{B}$ , node  $i$  does not increase its logical clock due to an external message again until node  $j$  has caught up.<sup>4</sup>  $\mathcal{A}^{bound}$  is immune to both the execution  $\mathcal{E}^{nice}$  and also the execution that incurred a large clock skew when  $\mathcal{A}^{max}$  is used. Nevertheless,  $\mathcal{A}^{bound}$  is not better asymptotically in the worst case. The idea behind the adversarial schedule for this specific algorithm is the following. Using fast messages, a chain of nodes within the graph is constructed such that the skew between all neighboring nodes in this chain is  $\mathcal{B}$ , creating a chain of dependency. Consequently, each node has to wait for his slower neighbor to catch up, resulting in long waiting times before the logical clocks can again be increased.

The execution  $\mathcal{E}^{fast} = (\mathcal{M}^{fast}, \mathcal{R}^{fast})$ , given the constant bound  $\mathcal{B}$ , is defined as follows. We set

$$\mathcal{M}^{fast}(t, i, j) := \begin{cases} 0 & \text{if } t = n - 1, j \neq 1 \\ 1 & \text{else.} \end{cases}$$

Let  $\hat{i} := \lfloor \frac{n-1}{\mathcal{B}} \rfloor + 1$ . The hardware clock rates are  $\forall t \forall i \neq \hat{i} + 1 : \mathcal{R}^{fast}(t, i) := 1 - \epsilon_i$ , where again  $\epsilon_n = 0$  and  $\epsilon_i > 0$  for all  $i \in \{1, \dots, n-1\}$ .  $\forall t < n - 1 : \mathcal{R}^{fast}(t, \hat{i} + 1) := 1 - \epsilon_{\hat{i} + 1}$ ,  $\forall t \geq n - 1 : \mathcal{R}^{fast}(t, \hat{i} + 1) := 1$ . Thus, the hardware clock of node  $\hat{i} + 1$  is sped up at time  $n - 1$ .

Note that the delay of messages is 0 at time  $n - 1$ , unless they are sent to node 1. As local computation requires 0 time in this asynchronous computation model, some nodes can potentially communicate an unbounded number of times, while other nodes wait for the arrival of some particular messages. As far as clock

<sup>4</sup> Note that node  $i$  still makes progress at the rate of its hardware clock.

synchronization is concerned, this entails that the clock values of nodes whose communication links have a delay of 0 at a specific time  $t$  will *stabilize* according to the clock synchronization algorithm in use. Such a stabilization always occurs independent of the clock synchronization algorithm, since logical clocks can only make progress, but the logical clock values cannot exceed the maximum clock value. This characteristic of asynchronous communication is exploited in the execution  $\mathcal{E}^{fast}$ .

**Lemma 3.** *Let  $\vartheta := n + \lfloor \frac{n-1}{\mathcal{B}} \rfloor - \kappa$ ,  $0 < \kappa < 1$ . When executing  $\mathcal{E}^{fast}$ , parameterized by  $\mathcal{B}$ , the skew between nodes  $\hat{i} := \lfloor \frac{n-1}{\mathcal{B}} \rfloor + 1$  and  $\hat{i} + 1$  at time  $\vartheta$  is at least  $(\lfloor \frac{n-1}{\mathcal{B}} \rfloor + 1 - \kappa) \epsilon_{\hat{i}}$ .*

PROOF. For any  $\mu < 1$  and  $i \geq 2$ , we have  $L_i(n-2+\mu) = i-2 + \mu\epsilon_i$ . Node 1 does not start its logical clock before time  $n-1$ . At time  $n-1$ , all communication between the nodes  $i \in \{2, \dots, n\}$  requires 0 time. Note that we do not need to specify which messages are handled first. The outcome of this stabilization process solely depends on the clock synchronization algorithm. In this case, after the clocks have stabilized, it holds that

$$L_i(n-1) := \begin{cases} (i-1)\mathcal{B} & i \in \{1, \dots, \hat{i}\} \\ n-1 & \text{else} \end{cases}$$

as node 1 can increase its logical clock to  $\mathcal{B}$  and consequently, node 2 can raise its clock to  $2\mathcal{B}$  etc. At this point, the clock rate of node  $\hat{i} + 1$  is set to 1, which means that  $L_{\hat{i}+1}(\tau) = \tau$  for all  $\tau \geq n-1$ . Node 1 receives the message that the logical clock of node 2 is already at  $\mathcal{B}$  at time  $n$  and subsequently increases its own clock to this value. In general, node  $j$  has to wait until time  $n+j-1$  before it can increase its logical clock by  $\mathcal{B}$ . Accordingly, node  $\hat{i}$  has to wait until time  $n + \lfloor \frac{n-1}{\mathcal{B}} \rfloor > \vartheta$  before it can increase its logical clock by  $\mathcal{B}$ .

Since  $L_{\hat{i}}(n-1) \leq n-1$ , we have that  $L_{\hat{i}}(\vartheta) \leq n-1 + (1-\epsilon_{\hat{i}})(1-\kappa + \lfloor \frac{n-1}{\mathcal{B}} \rfloor)$ . Hence  $|L_{\hat{i}+1}(\vartheta) - L_{\hat{i}}(\vartheta)| \geq \epsilon_{\hat{i}} (\lfloor \frac{n-1}{\mathcal{B}} \rfloor + 1 - \kappa)$ .  $\square$

The following theorem is immediate from Lemma 3.

**Theorem 2.** *Let  $\mathcal{A}^{bound}$  be the clock synchronization algorithm in use. When executing  $\mathcal{E}^{fast}$ , the skew between neighbors in  $G^{list}$  can be at least  $n\frac{\epsilon_i}{\mathcal{B}} - (\frac{\epsilon_i}{\mathcal{B}} + 1) \in \Theta(n)$ .*

$\square$

It is a strong assumption that some nodes can communicate an unbounded number of times while other nodes are not making any progress. If only a constant number of communication rounds were possible, the skew would be constant in this execution. However, the result is quite counterintuitive, as one might assume that the more and the faster nodes can communicate, the better clocks can be synchronized in general.

This theorem shows that the resulting skew can be large even though a constant bound has been specified. It turns out that the constant bound potentially results in waiting times that incur a skew of much more than the specified bound. In general, for any bound  $\mathcal{B}$ , it holds that  $\exists t, i, j \in \mathcal{N}_i : |L_i(t) - L_j(t)| = \mathcal{B}$ . If

the delay is 0 between a set of nodes in a particular execution, this results in a chain of nodes with clock values  $x, x + \mathcal{B}, x + 2\mathcal{B}, \dots, O(D)$ . If the length of this chain is  $\lambda$ , then the worst-case skew between two neighbors can be at least  $\Omega(\lambda)$ , because all nodes in the chain are constrained to wait for the slower node in the chain. The length of this chain can be  $\Theta(\frac{D}{\mathcal{B}})$  when the entire skew of  $\Theta(D)$  is allocated. Hence it follows that the skew between neighboring nodes can be at least  $\Omega(\frac{D}{\mathcal{B}})$ . According to the bounds maintained by neighboring nodes, a node might adapt its bound in order to adjust to this situation. Using a smaller bound than the current bound of one of the neighbors does not help, as the chain becomes even longer in the worst case, resulting in a larger worst-case skew. A node might allow a skew of  $\delta\mathcal{B}$  for any  $\delta > 1$ , if the maximum skew between any of its neighbors and one of this node's neighbors has already reached  $\mathcal{B}$ . In an execution such as  $\mathcal{E}^{fast}$ , the length of the chain is at most  $\Theta(\log_\delta \frac{D}{\mathcal{B}})$ . However, the maximum skew between neighbors is then  $\Omega(\delta^{\log_\delta \frac{D}{\mathcal{B}}}) = \Omega(\frac{D}{\mathcal{B}})$ , thus adapting the bounds does not improve the worst-case behavior either. Consequently, when minimizing the skew to the slowest node while increasing the clock whenever possible, the worst-case skew between neighboring nodes is always at least  $\Omega(\frac{D}{\mathcal{B}} + \mathcal{B}) = \Omega(\sqrt{D})$ .

## 5 A $O(d + \sqrt{D})$ -GCS Algorithm

The idea is that the knowledge of the diameter  $D$  can be exploited by setting the bound to  $O(\sqrt{D})$ . If the algorithm can ensure that the skew between any two nodes is always at most  $O(D)$ , the skew between neighbors will be at most  $O(\sqrt{D})$ , because nodes do not allow a larger skew than  $O(\sqrt{D})$  and the waiting time until they can raise their logical clocks again considerably is also bounded by  $O(\sqrt{D})$ . The algorithm which achieves this goal is described in greater detail in the following section.

### 5.1 Description of the Algorithm

As in the previous algorithms, the algorithm presented here, denoted by  $\mathcal{A}^{root}$ , also mandates the forwarding of the clock value to all neighboring nodes when the logical clock reaches an integer value. Apart from the diameter  $D$ , the algorithm must also know an upper bound on the hardware clock rate. Usually, the hardware clock rates will only differ slightly, therefore one could simply set  $\mathcal{U}$  to a realistic upper bound on the maximum clock rate, if the true bound is unknown. Algorithm 1 depicts the steps taken upon receipt of a message from a neighbor.

The information about the corresponding neighbor is updated if the newly arrived message indicates progress. Subsequently, the logical clock is increased if the slowest neighbor is not more than  $\mathcal{U}\sqrt{D} + 1$  behind. The logical clock is raised at most to the maximum logical clock value of all neighbors. Any message that does not cause a change of the logical clock is simply dropped.

```

if  $\tau > \tilde{L}_j(t)$  then
   $\tilde{L}_j(t) := \tau$ 
end if
if  $\max_{j \in \mathcal{N}_i}(\tilde{L}_j(t)) > L_i(t)$  and  $\min_{j \in \mathcal{N}_i}(\tilde{L}_j(t)) + \mathcal{U}\sqrt{D+1} > L_i(t)$  then
   $L_i(t) := \min(\max_{j \in \mathcal{N}_i}(\tilde{L}_j(t)), \min_{j \in \mathcal{N}_i}(\tilde{L}_j(t)) + \mathcal{U}\sqrt{D+1})$ 
  send  $\langle i, L_i(t) \rangle$  to all  $j \in \mathcal{N}_i$ 
end if

```

**Algorithm 1:** Node  $i$  calls this procedure when a message  $\langle j, \tau \rangle$  from node  $j$  with time stamp  $\tau$  is received at time  $t$ .

## 5.2 Analysis of the Algorithm

First, we prove that the worst-case skew between any two nodes is at most  $O(D)$ , which is asymptotically optimal. This *global* property is further used to derive the gradient property of  $\mathcal{A}^{root}$ . Note that both properties hold independent of the underlying network structure, thus the algorithm effectively bounds the skew between neighboring nodes in arbitrary graphs.

**Theorem 3 (Global Property).** *Let  $\mathcal{A}^{root}$  be the clock synchronization algorithm in use. For all executions and for any graph, it holds that  $\forall i, j, t : |L_i(t) - L_j(t)| < \mathcal{U}D + 1 \in O(D)$ .*

PROOF. The crucial observation is that for the slowest node  $\mathcal{A}^{root}$  is identical to  $\mathcal{A}^{max}$ . Recall that the slowest node is the node with the currently lowest clock value, and the node with the largest clock value is denoted the fastest node. After at most  $D$  time, the slowest node starts its clock. The progress of the fastest node is at most  $\mathcal{U}D$  during this time, resulting in a skew not larger than  $\mathcal{U}D$ . Before the next message reaches the slowest node, the fastest node can increase its logical clock by less than 1, resulting in a skew of less than  $\mathcal{U}D + 1$ . Once this message reaches the slowest node, the skew drops back to at most  $\mathcal{U}D$ . The slowest node can increase its logical clock at least at the same speed of the fastest node, thus the skew cannot grow any further. By reducing the messages delays, the slowest node can even catch up, as it can increase its clock earlier. If the messages are sped up such that the skew between the slowest node and any of its neighbors reaches  $\mathcal{U}\sqrt{D+1}$ , the slowest node can instantaneously raise its logical clock by  $\mathcal{U}\sqrt{D+1} > \mathcal{U}$ , hence the skew again decreases in this case.  $\square$

Using this bound on the *global* skew, we can limit the waiting time for any node and thereby guarantee that the skew between neighbors is always at most  $O(\sqrt{D})$ .

**Theorem 4 (Gradient Property).** *Let  $\mathcal{A}^{root}$  be the clock synchronization algorithm in use. For all executions and for any graph, it holds that  $\forall i, \forall j \in \mathcal{N}_i, t : |L_i(t) - L_j(t)| < 2\mathcal{U}\sqrt{D+1} \in O(\sqrt{D})$ .*

PROOF. It is evident that the skew can only be larger than  $\mathcal{U}\sqrt{D+1}$  when nodes are forced to wait for other nodes to increase their logical clocks. Let the skew between node  $i$  and  $j$  be  $\mathcal{U}\sqrt{D+1}$  at time  $t$ . Without loss of generality,

let  $L_i(t) = L_j(t) + \mathcal{U}\sqrt{D+1}$ . If there is a node  $k \in \mathcal{N}_j$  such that  $L_j(t) = L_k(t) + \mathcal{U}\sqrt{D+1}$ , node  $j$  has to wait for node  $k$  to increase its logical clock. Node  $k$  can again have a neighbor whose logical clock is  $\mathcal{U}\sqrt{D+1}$  behind etc. If this chain of dependent nodes has length  $\lambda$ , it takes at most  $\lambda$  time steps until node  $j$  can increase its logical clock by  $\mathcal{U}\sqrt{D+1}$ . The length  $\lambda$  is upper bounded by the maximum skew between any two nodes divided by  $\mathcal{U}\sqrt{D+1}$ . Hence, using Theorem 3,  $\lambda \leq \frac{\mathcal{U}D+1}{\mathcal{U}\sqrt{D+1}} \leq \sqrt{D+1}$ , because  $\mathcal{U} \geq 1$ . Node  $i$  cannot increase its logical clock by more than  $\mathcal{U}\sqrt{D+1}$  during this time, as the maximum hardware clock rate is  $\mathcal{U}$ , and node  $j$  increases its logical clock by at least  $\mathcal{U}\sqrt{D+1}$ , thus nodes can always catch up.

Before node  $j$  can raise its logical clock after  $\sqrt{D+1}$  time,  $i$  can increase its logical clock by less than  $\mathcal{U}\sqrt{D+1}$ , thus, at all times, the skew between any two neighbors is less than  $2\mathcal{U}\sqrt{D+1}$ .  $\square$

## 6 Conclusion

We have shown that aiming at achieving a minimal skew at all times naturally translates to oblivious algorithms, due to the fact that nodes do not have any information about the message delays and the hardware clock rates. Focusing on the fastest nodes potentially incurs a large skew between neighbors, but the fastest node must nevertheless have a large weight, as proven in our analyses. By assigning a large weight to the fastest node, the clocks will converge quickly to a large value, even if some neighboring nodes do not make any significant progress during the same time span.

However, there is an oblivious clock synchronization algorithm with a worst-case skew of  $O(d + \sqrt{D})$  between any two nodes at distance  $d$ , which answers the question whether there is an GCS algorithm with a skew of  $o(D)$  between neighboring nodes. This algorithm further guarantees a skew of  $\Theta(D)$  between any two nodes, which is globally asymptotically optimal.

A challenging open problem is whether the bound of  $\Theta(\sqrt{D})$  skew between neighbors is asymptotically optimal for oblivious algorithms. Additionally, it is also worth investigating how much more knowledge, e.g. the times when messages arrived or a larger message history in general, is required in order to substantially improve the worst-case skew. Another important aspect of clock synchronization is the number of messages required in order to effectively bound the skew between nodes. Analyzing the message complexity of gradient clock synchronization algorithms is another demanding problem which has not been studied so far.

## 7 Acknowledgements

We would like to thank Regina O'Dell and Thomas Moscibroda for many fruitful discussions and all the anonymous reviewers for their constructive comments and suggestions which helped to improve this paper.

## References

1. R. Fan, I. Chakraborty, and N. Lynch. Clock Synchronization for Wireless Networks. In *Proc. 8th International Conference on Principles of Distributed Systems (OPODIS)*, pages 400–414, 2004.
2. R. Fan and N. Lynch. Gradient Clock Synchronization. In *Proc. 23rd Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 320–327, New York, NY, USA, 2004. ACM Press.
3. J. Lundelius and N. Lynch. An Upper and Lower Bound for Clock Synchronization. *Information and Control*, 62(2/3):190–204, 1984.
4. 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)*, 2005.
5. R. Ostrovsky and B. Patt-Shamir. Optimal and Efficient Clock Synchronization under Drifting Clocks. In *Proceedings 18th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 3–12, 1999.
6. B. Patt-Shamir and S. Rajsbaum. A Theory of Clock Synchronization. In *Proceedings 26th Annual ACM Symposium on Theory of Computing (STOC)*, pages 810–819, 1994.
7. T. K. Srikanth and S. Toueg. Optimal Clock Synchronization. *J. ACM*, 34(3):626–645, 1987.