Chapter 23

Clock Synchronization

> "A man with a clock knows what time it is – a man with two is never sure." (Segal's Law)

23.1 Time & Clocks

Definition 23.1 (Second)**.** *A **second** is the time that passes during 9,192,631,770 oscillation cycles of a caesium-133 atom.*

Remarks:

- This definition is a bit simplified. The official definition is given by the *Bureau International des Poids et Mesures*.
- Historically, a second was defined as one in 86,400 parts of a day, dividing the day into 24 hours, 60 minutes and 60 seconds.
- Since the duration of a day depends on the unsteady rotation cycle of the Earth, the novel oscillation-based definition has been adopted. Leap seconds are used to keep time synchronized to Earth's rotation.

Definition 23.2 (Wall-Clock Time)**.** *The **wall-clock time** t^* is the true time (a perfectly accurate clock would show).*

Definition 23.3 (Clock)**.** *A **clock** is a device that tracks and indicates time.*

Remarks:

- A clock's time t is a function of the wall-clock time t^*, i.e., $t = f(t^*)$. Ideally, $t = t^*$, but in reality there are often errors.

Definition 23.4 (Clock Skew)**.** *The **clock skew** or clock error is the difference between two clocks. In practice the clock skew is often modeled as* $t = (1+\delta)t^* + \xi(t^*)$.

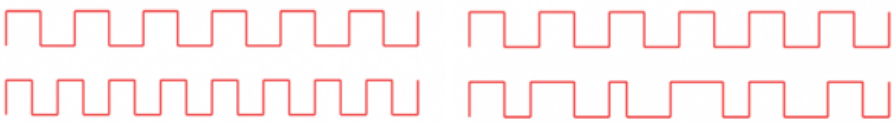

Figure 23.8: Drift (left) and Jitter (right). On top is a square wave, the wall-clock time t^*.

Remarks:

- The importance of accurate timekeeping and clock synchronization is reflected in the following statement by physicist Steven Jefferts: "We've learned that every time we build a better clock, somebody comes up with a use for it that you couldn't have foreseen."

Definition 23.5 (Drift)**.** *The* ***drift*** δ *is the predictable clock error.*

Remarks:

- Drift is relatively constant over time, but may change with supply voltage, temperature and age of an oscillator.
- Stable clock sources, which offer a low drift, are generally preferred, but also more expensive, larger and more power hungry, which is why many consumer products feature inaccurate clocks.

Definition 23.6 (Parts Per Million)**.** *Clock drift is indicated in* ***parts per million (ppm)****. One ppm corresponds to a time error growth of one microsecond per second.*

Remarks:

- In PCs, the so-called *real-time clock* normally is a crystal oscillator with a maximum drift between 5 and 100 ppm.
- Applications in signal processing, for instance GPS, need more accurate clocks. Common drift values are 0.5 to 2 ppm.

Definition 23.7 (Jitter)**.** *The* ***jitter*** ξ *is the unpredictable, random noise of the clock error.*

Remarks:

- In other words, jitter is the irregularity of the clock. Unlike drift, jitter can vary fast.
- Jitter captures all the errors that are not explained by drift. Figure 23.8 visualizes the concepts.

23.2 Clock Synchronization Algorithm

How can the nodes of a distributed system remain synchronized? We first need to specify the model more clearly.

Remarks:

- The system is modeled as an arbitrary connected graph $G = (V, E)$, where each node v is equipped with a clock C_v. A node v can only communicate directly with its *neighboring nodes*. Every node w such that $\{v, w\} \in E$ is called a *neighbor* of node v.
- The longest shortest path between any two nodes is called the *diameter* of the network, denoted by D.
- The *clock rates*, i.e., the rate at which clock values change over time, is always in the range $[1 - \varepsilon, 1 + \varepsilon]$ for some constant $\varepsilon \ll 1$.
- Messages exchanged over any edge $e \in E$ is subject to a *message delay*, which is always in the range $[0, T]$ for some upper bound T unknown to the nodes.
- What about networks where message delays are fairly predictable? In that case, the range $[0, T]$ can be considered the unpredictable component of the message delay.

Algorithm 23.9 Clock synchronization algorithm (code for node v)

```
1: Increase clock C_v at local clock rate
2: upon clock value C_v reaches next integer value:
3:    Send C_v to all neighboring nodes
4: end upon
5: upon receiving clock value C_w from node w:
6:    if C_w > C_v then
7:       C_v := C_w
8:       Send C_v to all neighboring nodes
9:    end if
10: end upon
```

Remarks:

- Algorithm 23.9 is quite simple: Periodically inform all neighboring nodes about the current clock value, for example, when it reaches the next integer value, where an integer increment represents the passing of some specific amount of time. Whenever a node notices that it is lagging behind, which must be the case when it receives a larger clock value, it sets its clock value to the received value.

Theorem 23.10. *Algorithm 23.9 guarantees that the clock skew between any two nodes at any time is at most* $(1 + \varepsilon)DT + \frac{2\varepsilon}{1-\varepsilon}$.

Proof. Assume that the largest clock skew is reached when some node has the clock value C_{max} at some (real) time t^*. Let node v have the smallest clock value C_v at that time, maximizing the clock skew. Let $C'_{max} < C_{max}$ denote the last clock value that was propagated from one of the (possibly multiple) nodes with the largest clock value and reached node v at time $t^* - \delta$ for some $\delta \geq 0$. Since the message delay is at most T, we have that the time interval between C'_{max} and C_{max} is upper bounded by $DT + \delta$.

Note that it further holds that

$$\delta \leq \frac{1}{1-\varepsilon} \tag{23.1}$$

because at most this time passes before a node broadcasts the next clock value, i.e., $\delta > \frac{1}{1-\varepsilon}$ would imply that v must also receive $C'_{max} + 1$, a contradiction to the assumption that C'_{max} is the last received value.

Since v sets its clock value to any received value that is larger than its own clock value, we have that

$$C_v \geq C'_{max} + (1-\varepsilon)\delta. \tag{23.2}$$

The largest clock value always increases at most at rate $1+\varepsilon$ during the time interval between C'_{max} and C_{max}, which is upper bounded by $DT + \delta$, so we also have that

$$C_{max} \leq C'_{max} + (1+\varepsilon)(DT + \delta). \tag{23.3}$$

Combining these inequalities, we get

$$C_{max} - C_v \overset{(23.2),(23.3)}{\leq} (1+\varepsilon)DT + 2\varepsilon\delta \overset{(23.1)}{\leq} (1+\varepsilon)DT + \frac{2\varepsilon}{1-\varepsilon}.$$

□

23.3 Global Skew

Definition 23.11 (Global Skew)**.** *For any network of nodes running a specific clock synchronization algorithm, the* ***global skew*** *is the maximum clock skew between any two nodes.*

Remarks:

- As shown in the previous section, we can bound the global skew to $\approx DT$. Can we do better?
- There is a well-known result that a global skew of $DT/2$ cannot be prevented. However, we can prove a stronger lower bound on the global skew if we add the following condition.

Definition 23.12 (Linear real-time envelope)**.** *A clock synchronization algorithm, starting at time $t = 0$ is said to satisfy the* ***linear real-time envelope*** *condition if the clock values of all nodes are in the range $[(1-\varepsilon)t, (1+\varepsilon)t]$ at all times t.*

Remarks:

- In order to prove lower bounds on clock skews, we examine clock values in indistinguishable *executions*.

Definition 23.13 (Execution)**.** *An **execution** $\mathcal{E}$ of a clock synchronization algorithm specifies the hardware rate of every node and the message delay of every message sent during the execution.*

Theorem 23.14. *Any clock synchronization algorithm that satisfies the linear real-time envelope condition must have a global skew of at least* $(1-2\varepsilon)DT$.

Proof. Let $v_0, v_1 \ldots, v_D$ be a path connecting nodes v_0 and v_D at distance D. We restrict our attention to nodes on this path. We prove this statement using three indistinguishable executions.

- Execution $\mathcal{E}_1$: All clock rates are always $1-\varepsilon$. For all $i \in \{1, \ldots, D\}$, all message delays from v_i to v_{i-1} are T, and the message delays from v_{i-1} to v_i are 0.

- Execution $\mathcal{E}_2$: All clock rates are always $1+\varepsilon$. For all $i \in \{1, \ldots, D\}$, all message delays from v_i to v_{i-1} are $\frac{1-\varepsilon}{1+\varepsilon}T$, and the message delays from v_{i-1} to v_i are 0.

- Execution $\mathcal{E}_3$: For all $i \in \{0, \ldots, D\}$, the clock rate of node v_i is $1+\varepsilon-\frac{2\varepsilon}{D}i$ until time $t_0 := \frac{DT}{2\varepsilon}$. After that, all clock rates are $1-\varepsilon$. All messages delays are adjusted so that the execution is indistinguishable from execution $\mathcal{E}_1$ and $\mathcal{E}_2$.

Since all clocks in execution $\mathcal{E}_2$ run faster by a factor of $\frac{1+\varepsilon}{1-\varepsilon}$ compared to execution $\mathcal{E}_1$, messages delays must be lower by a factor of $\frac{1-\varepsilon}{1+\varepsilon}$ for messages to arrive at the same local times. Since this is the case, the executions are indistinguishable. Note that nodes may not run their clocks more slowly (or decrease their clock value if this is allowed), as otherwise they would violate the linear real-time envelope condition in execution $\mathcal{E}_1$ because all clock rates are always $1-\varepsilon$. Similarly, the nodes may not increase their clock rates or clock values as otherwise they might violate the same condition if the nodes are in execution $\mathcal{E}_2$.

Execution $\mathcal{E}_3$ is indistinguishable by definition but it remains to show that it is a *valid* execution, i.e., all message delays are in the range $[0, T]$. All clock rates are the same as in execution $\mathcal{E}_1$, so we only need to consider messages that are sent and received before t_0. Since the difference in the clock rates between neighboring nodes is $\frac{2\varepsilon}{D}$, the maximum skew that is built up until time t_0 is $\frac{DT}{2\varepsilon} \cdot \frac{2\varepsilon}{D} = T$. For all $i \in \{1, \ldots, D\}$, v_{i-1}'s clock runs faster than v_i's clock, so the delays of messages from v_i to v_{i-1} may decrease by at most T, which is fine because it is T in execution $\mathcal{E}_1$. Similarly, the message delays from any node v_{i-1} to v_i may increase by at most T, which is fine because they are 0 in execution $\mathcal{E}_1$ (and $\mathcal{E}_2$). We conclude that the executions are all indistinguishable.

Without loss of generality, assume that $C_{v_0} \geq C_{v_D}$ at time t_0 in execution $\mathcal{E}_1$. Node v_0 reaches C_{v_0} already at time $t_o' := \frac{1-\varepsilon}{1+\varepsilon}t_0$ in execution $\mathcal{E}_3$, i.e., $\frac{1-\varepsilon}{1+\varepsilon}t_0 - t_0 = \frac{2\varepsilon}{1+\varepsilon}t_0 = \frac{DT}{1+\varepsilon}$ sooner than node v_D reaches clock value C_{v_D}. Since

v_D's clock value increases at a rate of $1-\varepsilon$, v_D's clock value at time t_0' is $C'_{v_D} \coloneqq C_{v_D} - (1-\varepsilon)\frac{DT}{1+\varepsilon}$. Hence, it follows that the clock skew at time t_0' is

$$C_{v_0} - C'_{v_D} = C_{v_0} - \left(C_{v_D} - (1-\varepsilon)\frac{DT}{1+\varepsilon}\right) \geq (1-2\varepsilon)DT.$$

□

Remarks:

- Note that Algorithm 23.9 satisfies Definition 23.12. We conclude that its global skew is nearly optimal.

23.4 Local Skew

Definition 23.15 (Local Skew)**.** *For any network of nodes running a specific clock synchronization algorithm, the* ***local skew*** *is the maximum clock skew between neighboring nodes.*

Remarks:

- We showed in the previous section that the global skew can be bounded by $\Theta(DT)$, incurring an average clock skew of $\Theta(T)$ on a path of length D.
- One might reasonably assume that a local skew bound of $\Theta(T)$ is possible. As we will prove now, it is not: The best possible synchronization between neighboring nodes depends on the *diameter of the network*!
- Let's start with the local skew of Algorithm 23.9.

Lemma 23.16. *Algorithm 23.9 has a local skew of* $\Theta(DT)$.

Proof. We saw that a global skew of $\Theta(DT)$ can be built up on a path $v_0, v_1, \ldots, v_D$ of length D. If the message delay is now reduced to zero to all nodes except v_D, then all nodes will immediately increase their clock values to the largest clock value. Since this includes also the neighbors of node v_D, the clock skew between v_D and its neighbors is $\Theta(DT)$. □

Remarks:

- Naturally, as soon as v_D receives a message from its neighbors, the clock skew will disappear. However, the clock skew may not be short-lived: The clock skew can be $s \in O(DT)$ for $\Theta(DT/s)$ time!
- This may seem like a weakness of this particular algorithm but other "reasonable" algorithm also have a bad local skew. For example, an algorithm that averages between all neighbors is even worse! It has a global skew of $\Theta(D^2T)$ and a local skew of $\Theta(DT)$.
- We will now show that no algorithm can guarantee a local skew of $\Theta(T)$.

- As before, the hardware clock rate is in the range $[1-\varepsilon, 1+\varepsilon]$ for every node v at all times. Let $h_v(t)$ denote the hardware clock rate of node v at time t. Instead of changing clock values instantaneously, any node v may adapt the clock rate at which its clock C_v advances. Concretely, we introduce the constants α and β such that for any node v and time t it may increase its clock value at a rate of at least α and at most β.

- We will again use indistinguishable executions in our proof. Let $t_{\mathcal{E}}$ denote the time when execution $\mathcal{E}$ ends, and let $C_v^{\mathcal{E}}(t)$ be the clock value at node v in execution $\mathcal{E}$ at time t during the execution.

- Let $d(v,w)$ be the distance between v and w. If $d(v,w) > 1$, then v and w are not directly connected.

Lemma 23.17. *Given any clock synchronization algorithm and any two nodes v and w, if all clock rates are 1 and all message delays are $T/2$ in an execution $\mathcal{E}$ of duration $\frac{1+\varepsilon/2}{\varepsilon}d(v,w)T$, then there is an execution $\bar{\mathcal{E}}$, starting at the same time as $\mathcal{E}$ and with a duration of $\frac{1}{\varepsilon}d(v,w)T$ such that $C_v^{\mathcal{E}}(t_{\mathcal{E}}) = C_v^{\bar{\mathcal{E}}}(t_{\bar{\mathcal{E}}})$ and $C_w^{\mathcal{E}}(t_{\bar{\mathcal{E}}}) = C_w^{\bar{\mathcal{E}}}(t_{\bar{\mathcal{E}}})$.*

Proof. Execution $\bar{\mathcal{E}}$ is defined as follows. The hardware clock rate of any node u is

$$h_u(t) = \begin{cases} 1 + \frac{\varepsilon}{2}\left(1 - \frac{d(v,u)}{d(v,w)}\right) & \text{if } d(v,u) \leq d(v,w) \\ 1 & \text{else} \end{cases}$$

Message delays are adjusted in such a way that, at each node, each send and receive event in execution $\bar{\mathcal{E}}$ happens at the same hardware clock time as in execution $\mathcal{E}$. Since all events happen at the same local times, the executions are indistinguishable!

However, it remains to show that $\bar{\mathcal{E}}$ is actually a *valid* execution. The hardware clock rates are in the range $[1, 1+\varepsilon/2] \subset [1-\varepsilon, 1+\varepsilon]$, which is fine. What about the message delays? The hardware clocks of neighboring nodes drift apart at a rate of at most $\frac{\varepsilon}{2d(v,w)}$. Since the duration of $\bar{\mathcal{E}}$ is $\frac{1}{\varepsilon}d(v,w)T$, events are "spread apart" by at most $\frac{\varepsilon}{2d(v,w)}\frac{1}{\varepsilon}d(v,w)T = T/2$. Since messages delays are always $T/2$ in $\mathcal{E}$, messages delays are in the range $[0,T]$ in $\bar{\mathcal{E}}$.

Node v's clock runs $1+\frac{\varepsilon}{2}$ times faster in $\bar{\mathcal{E}}$ but $\bar{\mathcal{E}}$ is $1+\frac{\varepsilon}{2}$ times shorter than $\mathcal{E}$. Since $\mathcal{E}$ and $\bar{\mathcal{E}}$ are indistinguishable, it follows that $C_v^{\mathcal{E}}(t_{\mathcal{E}}) = C_v^{\bar{\mathcal{E}}}(t_{\bar{\mathcal{E}}})$. Node w's clock runs at a clock rate of 1 in both executions. Since the executions are indistinguishable, it follows that $C_w^{\mathcal{E}}(t_{\bar{\mathcal{E}}}) = C_w^{\bar{\mathcal{E}}}(t_{\bar{\mathcal{E}}})$. □

Remarks:

- It is important to read the clock conditions carefully: $C_v^{\mathcal{E}}(t_{\mathcal{E}}) = C_v^{\bar{\mathcal{E}}}(t_{\bar{\mathcal{E}}})$ means that the value of node v's clock at the end of execution $\mathcal{E}$ and at the end of execution $\bar{\mathcal{E}}$ are the same, even though $\bar{\mathcal{E}}$ is shorter! On the other hand, $C_w^{\mathcal{E}}(t_{\bar{\mathcal{E}}}) = C_w^{\bar{\mathcal{E}}}(t_{\bar{\mathcal{E}}})$ means that w has the same time at time $t_{\bar{\mathcal{E}}}$ in both $\mathcal{E}$ and $\bar{\mathcal{E}}$.

- So, the clock skew between v and w at time $t_{\bar{\mathcal{E}}}$ is greater in execution $\bar{\mathcal{E}}$ than in execution $\mathcal{E}$.

Definition 23.18 (Extended execution)**.** *Given an execution $\mathcal{E}$ ending at time $t_{\mathcal{E}}$, we can define hardware clock rates and message delays in the interval $[t_{\mathcal{E}}, t_{\mathcal{E}'}]$. This extension is called an* ***extended execution*** *$\mathcal{E}'$ with duration $t_{\mathcal{E}'} - t_{\mathcal{E}}$. Execution $\mathcal{E}'$ inherits the state of all nodes and all messages sent in $\mathcal{E}$ that did not reach their destination until time $t_{\mathcal{E}}$.*

Theorem 23.19. *Let $b := \left\lceil \frac{4(\beta-\alpha)(1+\varepsilon/2)}{\alpha\varepsilon} \right\rceil$. No clock synchronization algorithm can prevent a local skew of*

$$\frac{\lfloor \log_b D \rfloor + 2}{4} \alpha T \in \Omega(T \log D)$$

on any graph G of diameter D.

Proof. Let $D' := b^{\lfloor \log_b D \rfloor} \leq D$. For any $k \in \mathbb{N}$, $0 \leq k \leq \log_b D'$, we claim that there are two nodes v_k and w_k at distance $d(v_k, w_k) = D'/b^k$ such that

$$C_{v_k}^{\bar{\mathcal{E}}_k}(t_{\bar{\mathcal{E}}_k}) - C_{w_k}^{\bar{\mathcal{E}}_k}(t_{\bar{\mathcal{E}}_k}) \geq \frac{k+2}{4} \alpha d(v_k, w_k) T \tag{23.4}$$

at the end of some execution $\bar{\mathcal{E}}_k$. Note that when setting $k := \log_b D'$, which means that $d(v_k, w_k) = 1$, we get the claimed bound on the local skew of $\frac{\lfloor \log_b D \rfloor + 2}{4} \alpha T$. So, it remains to prove the claim!

Consider any two nodes v_0 and w_0 at distance $d(v_0, w_0) = D'$. Execution $\mathcal{E}_0$ has a duration of $\frac{1+\varepsilon/2}{\varepsilon} D'T$. All message delays are $T/2$ and all hardware clock rates are 1. Without loss of generality, we can assume that $C_{v_0}^{\mathcal{E}_0}(t_{\mathcal{E}_0}) \geq C_{w_0}^{\mathcal{E}_0}(t_{\mathcal{E}_0})$. According to Lemma 23.17, there is an execution $\bar{\mathcal{E}}_0$ of duration $\frac{1}{\varepsilon} D'T$ that makes v_0's clock run faster but the executions are indistinguishable until time $t_{\bar{\mathcal{E}}_0}$. Since $C_v^{\bar{\mathcal{E}}_0}(t_{\bar{\mathcal{E}}_0}) = C_v^{\mathcal{E}_0}(t_{\mathcal{E}_0}) \geq C_{w_0}^{\mathcal{E}_0}(t_{\mathcal{E}_0})$ and w_0's clock must increase by at least $\alpha(t_{\mathcal{E}_0} - t_{\bar{\mathcal{E}}_0}) = \alpha D'T/2$ between $t_{\bar{\mathcal{E}}_0}$ and $t_{\mathcal{E}_0}$, the clock skew at time $t_{\bar{\mathcal{E}}_0}$ must be at least $\alpha D'T/2 = \frac{0+2}{4} \alpha d(v_0, w_0) T$.

For the induction step, we assume that the claim is true for k, i.e., Inequality 23.4 holds for some nodes v_k and w_k at time $t_{\bar{\mathcal{E}}_k}$. We will now extend the execution for a duration of $\frac{1+\varepsilon/2}{\varepsilon} \frac{d(v_k, w_k)}{b} T$. For messages that did not arrive before $t_{\bar{\mathcal{E}}_k}$, we simply define that they all arrive immediately at the start of the extended execution. As before, the hardware clock rate of all nodes is 1 and message delays are $T/2$ for messages sent after $t_{\bar{\mathcal{E}}_k}$. During the extended execution, v_k increases its clock at least at rate α, whereas w_k increases its clock at most at rate β for the entire duration of the extended execution. Thus, the clock skew is reduced by at most $(\beta - \alpha) \frac{1+\varepsilon/2}{\varepsilon} \frac{d(v_k, w_k)}{b} T \leq \frac{\alpha}{4} d(v_k, w_k) T$, i.e., the clock skew at the end of the extended execution, at time $t_{\mathcal{E}_{k+1}}$, is at least $\frac{k+1}{4} \alpha d(v_k, w_k) T$.

Applying the pidgeonhole principle, there must be nodes v_{k+1} and w_{k+1} at distance $d(v_{k+1}, w_{k+1}) = d(v_k, w_k)/b$ for which it holds that the clock skew at time $t_{\mathcal{E}_{k+1}}$ is at least $\frac{k+1}{4} \alpha d(v_{k+1}, w_{k+1}) T$.

We can now use Lemma 23.17 again: There is an extended execution $\bar{\mathcal{E}}_{k+1}$

of duration $\frac{1}{\varepsilon}\frac{d(v_k,w_k)}{b}T = \frac{1}{\varepsilon}d(v_{k+1},w_{k+1})T$ such that

$$\begin{aligned}
C_{v_k}^{\bar{\mathcal{E}}_k}(t_{\bar{\mathcal{E}}_{k+1}}) &= C_{v_k}^{\mathcal{E}_{k+1}}(t_{\mathcal{E}_{k+1}}) \\
&\geq C_{w_k}^{\mathcal{E}_{k+1}}(t_{\mathcal{E}_{k+1}}) + \frac{k+1}{4}\alpha d(v_{k+1},w_{k+1})T \\
&\geq C_{w_k}^{\bar{\mathcal{E}}_k}(t_{\bar{\mathcal{E}}_{k+1}}) + \alpha(t_{\mathcal{E}_{k+1}} - t_{\bar{\mathcal{E}}_{k+1}}) + \frac{k+1}{4}\alpha d(v_{k+1},w_{k+1})T \\
&= C_{w_k}^{\bar{\mathcal{E}}_k}(t_{\bar{\mathcal{E}}_{k+1}}) + \frac{\alpha}{2}d(v_{k+1},w_{k+1})T + \frac{k+1}{4}\alpha d(v_{k+1},w_{k+1})T \\
&= C_{w_k}^{\bar{\mathcal{E}}_k}(t_{\bar{\mathcal{E}}_{k+1}}) + \frac{(k+1)+2}{4}\alpha d(v_{k+1},w_{k+1})T,
\end{aligned}$$

which proves the claim. □

Remarks:

- The lower bound uses a fixed upper bound β on the rate at which the clock value can be increased. Note that $b \to \infty$ if $\beta \to \infty$, that is, the local skew bound becomes $\Omega(T)$ when nodes are allowed to increase their clocks at arbitrarily high rates, including instantaneous increases by any amount. A more complex proof exists that shows that the lower bound of $\Omega(T \log D)$ holds even for an unbounded maximum clock rate.

- Note that these are worst-case bounds. In practice, clock drift and message delays may not be the worst possible. Typically the speed of hardware clocks changes at a comparatively slow pace and the message transmission times follow a benign probability distribution. If we assume this, better protocols do exist, in theory as well as in practice.

Chapter Notes

It has been known for a long time that the global clock skew is $\Theta(DT)$ [LL84, ST87]. The problem of synchronizing the clocks of nearby nodes was introduced by Fan and Lynch in [LF04]; they proved a surprising lower bound of $\Omega(T \log D / \log\log D)$ for the local skew. The first algorithm providing a non-trivial local skew of $\mathcal{O}(T\sqrt{D})$ was given in [LW06]. Later, matching upper and lower bounds of $\Theta(T \log D)$ were proven in [LLW10]. The problem has also been studied in a dynamic setting [KLO09, KLLO10] or when a fraction of nodes experience byzantine faults and the other nodes have to recover from faulty initial state (i.e., self-stabilizing) [DD06, DW04]. The self-stabilizing byzantine case has been solved with asymptotically optimal skew [KL18].

Clock synchronization is a well-studied problem in practice, for instance regarding the global clock skew in sensor networks, e.g. [EGE02, GKS03, MKSL04, PSJ04]. One more recent line of work is focussing on the problem of minimizing the local clock skew [BvRW07, SW09, LSW09, FW10, FZTS11].

This chapter was written in collaboration with Thomas Locher.

Bibliography

[BvRW07] Nicolas Burri, Pascal von Rickenbach, and Roger Wattenhofer. Dozer: Ultra-Low Power Data Gathering in Sensor Networks. In *International Conference on Information Processing in Sensor Networks (IPSN), Cambridge, Massachusetts, USA*, April 2007.

[DD06] Ariel Daliot and Danny Dolev. Self-Stabilizing Byzantine Pulse Synchronization. *Computing Research Repository*, 2006.

[DW04] Shlomi Dolev and Jennifer L. Welch. Self-stabilizing clock synchronization in the presence of Byzantine faults. September 2004.

[EGE02] Jeremy Elson, Lewis Girod, and Deborah Estrin. Fine-grained Network Time Synchronization Using Reference Broadcasts. *ACM SIGOPS Operating Systems Review*, 36:147–163, 2002.

[FW10] Roland Flury and Roger Wattenhofer. Slotted Programming for Sensor Networks. In *International Conference on Information Processing in Sensor Networks (IPSN), Stockholm, Sweden*, April 2010.

[FZTS11] Federico Ferrari, Marco Zimmerling, Lothar Thiele, and Olga Saukh. Efficient Network Flooding and Time Synchronization with Glossy. In *Proceedings of the 10th International Conference on Information Processing in Sensor Networks (IPSN)*, pages 73–84, 2011.

[GKS03] Saurabh Ganeriwal, Ram Kumar, and Mani B. Srivastava. Timing-sync Protocol for Sensor Networks. In *Proceedings of the 1st international conference on Embedded Networked Sensor Systems (SenSys)*, 2003.

[KL18] Pankaj Khanchandani and Christoph Lenzen. Self-Stabilizing Byzantine Clock Synchronization with Optimal Precision. January 2018.

[KLLO10] Fabian Kuhn, Christoph Lenzen, Thomas Locher, and Rotem Oshman. Optimal Gradient Clock Synchronization in Dynamic Networks. In *29th Symposium on Principles of Distributed Computing (PODC), Zurich, Switzerland*, July 2010.

[KLO09] Fabian Kuhn, Thomas Locher, and Rotem Oshman. Gradient Clock Synchronization in Dynamic Networks. In *21st ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), Calgary, Canada*, August 2009.

[LF04] Nancy Lynch and Rui Fan. Gradient Clock Synchronization. In *Proceedings of the 23rd Annual ACM Symposium on Principles of Distributed Computing (PODC)*, 2004.

[LL84] Jennifer Lundelius and Nancy Lynch. An Upper and Lower Bound for Clock Synchronization. *Information and Control*, 62:190–204, 1984.

[LLW10] Christoph Lenzen, Thomas Locher, and Roger Wattenhofer. Tight Bounds for Clock Synchronization. In *Journal of the ACM, Volume 57, Number 2*, January 2010.

[LSW09] Christoph Lenzen, Philipp Sommer, and Roger Wattenhofer. Optimal Clock Synchronization in Networks. In *7th ACM Conference on Embedded Networked Sensor Systems (SenSys), Berkeley, California, USA*, November 2009.

[LW06] Thomas Locher and Roger Wattenhofer. Oblivious Gradient Clock Synchronization. In *20th International Symposium on Distributed Computing (DISC), Stockholm, Sweden*, September 2006.

[MKSL04] Miklós Maróti, Branislav Kusy, Gyula Simon, and Ákos Lédeczi. The Flooding Time Synchronization Protocol. In *Proceedings of the 2nd international Conference on Embedded Networked Sensor Systems*, SenSys '04, 2004.

[PSJ04] Santashil PalChaudhuri, Amit Kumar Saha, and David B. Johnson. Adaptive Clock Synchronization in Sensor Networks. In *Proceedings of the 3rd International Symposium on Information Processing in Sensor Networks*, IPSN '04, 2004.

[ST87] T. K. Srikanth and S. Toueg. Optimal Clock Synchronization. *Journal of the ACM*, 34:626–645, 1987.

[SW09] Philipp Sommer and Roger Wattenhofer. Gradient Clock Synchronization in Wireless Sensor Networks. In *8th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN), San Francisco, USA*, April 2009.