Lecture 1

Synchronizing Clocks

In this lecture series, we are going to approach fault-tolerant clock generation and distribution from a theoretical angle. This means we will formalize parametrized problems and prove impossibilities, lower bounds, and upper bounds for them. However, make no mistake: these tasks are derived from real-world challenges, and a lot of the ideas and concepts can be used in the design of highly reliable and scalable hardware solutions. The first lecture focuses on the basic task at hand, without bells and whistles. Asking more refined questions will prompt more refined answers later in the course; nonetheless, the initial lecture offers a good impression of the general approach and flair of the course.

1.1 The Clock Synchronization Problem

We describe a distributed system by a simple, connected graph $G = (V, E)$ (see Appendix A), where V is the set of $n := |V|$ nodes (our computational entities, e.g., computers in a network) and nodes v and w can directly communicate if and only if there is an edge $\{v, w\} \in E$. Each node is equipped with a *local* or *hardware clock.* We model this clock as a strictly increasing function $H_v \colon \mathbb{R}_0^+ \to \mathbb{R}_0^+$, whose rate of increase is between 1 and $\vartheta > 1$:

$$\forall v \in V, t, t' \in \mathbb{R}_0^+, t \geq t' \colon t - t' \leq H_v(t) - H_v(t') \leq \vartheta(t - t')\,,$$

where $t \in \mathbb{R}_0^+$ denotes "perfect" Newtonian *real time* (which is not known to the nodes). Note that even if the hardware clocks of nodes v and w would be initially perfectly synchronized (i.e., $H_v(0) = H_w(0)$), over time they could drift apart at a rate of up to $\vartheta - 1$. Accordingly, we refer to $\vartheta - 1$ as the *maximum drift,* or, in short, *drift.* In order to establish or maintain synchronization, nodes need to communicate with each other. To this end, on any edge $\{v, w\}$, v can send messages to w (and vice versa). However, it is not known how long such a message is under way. A message sent at time t is received at a time $t' \in (t + d - u, t + d)$, where d is the (maximum) *delay* and u is the (delay) *uncertainty.* We subsume possible delays due to computations in d, i.e., at the time t' when the message is received in our abstract model, all updates to the state of the receiving node take effect and any message it sends in immediate response is sent. Nodes may also send messages later, at a time t'' specified by some hardware clock value $H > H_v(t')$; the messages are then sent at the time

t'' when $H_v(t'') = H$, unless reception of a message at an earlier time makes v "change its mind."

An *execution* of an algorithm on a system is given by specifying clock functions H_v as above to each $v \in V$ and assigning to each message a reception time $t' \in (t + d - u, t + d)$, where t is the time it was sent. Note that by performing this inductively over increasing reception times enables to always determine from the execution up to the current time what the state of each node is and which messages are in transit, i.e., choosing clock functions and delays fully determines an execution.

The *clock synchronization problem* requires each node $v \in V$ to compute a *logical clock* $L_v \colon \mathbb{R}_0^+ \to \mathbb{R}_0^+$, where $L_v(t)$ is determined from the current state of the node (computed when receiving the most recent message, or the initial state if no message has been received yet) and $H_v(t)$. The goal is to minimize, for any possible execution $\mathcal{E}$, the *global skew*

$$\mathcal{G} \coloneqq \sup_{t \in \mathbb{R}_0^+} \{\mathcal{G}(t)\},$$

where

$$\mathcal{G}(t) \coloneqq \max_{v,w \in V} \{|L_v(t) - L_w(t)|\} = \max_{v \in V}\{L_v(t)\} - \min_{v \in V}\{L_v(t)\}$$

is the *global skew at time t.*

For simplicity, this notation does not reflect the dependence on the execution. The goal is to bound $\mathcal{G}$ for all possible executions, yet frequently we will argue about specific executions. We will make the dependence explicit only when reasoning about different executions concurrently.

Remarks:

- For practical purposes, clocks are discrete and bounded (i.e., wrap around to 0 after reaching a maximum value), and nodes may not be able to read them (perform computations, send messages, etc.) at arbitrary times. We hide these issues in our abstraction, as they can be handled easily, by adjusting d and u to account for them and making minor adjustments to algorithms.

- A cheap quartz oscillator has a drift of $\vartheta - 1 \approx 10^{-5}$, which will be more than accurate enough for running all the algorithms we'll get to see. In some cases, however, one might only want to use basic digital ring oscillators (an odd number of inverters arranged in a cycle), for which $\vartheta - 1 \approx 10\%$ is not unusual.

- There are other forms of communication than point-to-point message passing. Changing the mode of communication has, in most cases, little influence on a conceptual level, though.

- Another issue is that clocks may not be perfectly synchronized at time 0. After all, we want to run a synchronization algorithm to make clocks agree, so assuming that this is already true from the start would create a chicken-and-egg problem. But if we assume that initial clock values are arbitrary, we cannot bound $\mathcal{G}$. Instead, we assume that, for some $F \in \mathbb{R}^+$, it holds that $H_v(0) \in [0, F]$ for all $v \in V$. We then can bound $\mathcal{G}$ in terms of F (and, of course, other parameters).

- In order to perform induction over message sending and/or reception times, we need the additional assumption that nodes send only finitely many messages in finite time. As physics ensure that is the case (and any reasonable algorithm should not attempt otherwise), we implicitly make this assumption throughout the course.

1.2 The Max Algorithm

Let's start with our first algorithm. It's straightforward: Nodes initialize their logical clocks to their initial hardware clock value, increase it at the rate of the hardware clock, and set it to the largest value they can be sure that some other node has reached. To make the latter useful, each node broadcasts its clock value (i.e., sends it to all neighbors) whenever it reaches an integer multiple of some parameter T. See Algorithm 1.1 for the pseudocode.

Algorithm 1.1: Basic Max Algorithm. Parameter $T \in \mathbb{R}^+$ controls the message frequency. The code lists the actions of node v at time t.

```
1 L_v(0) := H_v(0)
2 at all times, increase L_v at the rate of H_v
3 if received ⟨L⟩ at time t and L > L_v(t) then
4 |   L_v(t) := max{L_v(t), L}
5 if L_v(t) = kT for some k ∈ ℕ then
6 |   send ⟨L_v(t)⟩ to all neighbors
```

Lemma 1.1. *In a system executing Algorithm 1.1, it holds that*

$$\mathcal{G}(t) \leq \vartheta dD + (\vartheta - 1)T$$

for all $t \geq dD + T$, where D is the diameter of G.

Proof. Set $L := \max_{v \in V}\{L_v(t - dD - T)\}$. No node ever sets its logical clock to a value that has not been reached by another node before. Together with the fact that hardware clocks increase at rate at most ϑ, this implies that

$$\max_{v \in V}\{L_v(t)\} \leq \max_{v \in V}\{L_v(t - dD - T)\} + \vartheta(dD + T) = L + \vartheta(dD + T)\,.$$

Let v be a node such that $L_v(t - dD - T) = \max_{w \in V}\{L_w(t - dD - T)\}$. As the logical clock of v increases at least at rate 1, the minimum rate of its hardware clock, and is never set back to a smaller value, we have that $L_v(t') = kT$ for some $k \in \mathbb{N}$ and $t' \in [t, t + T)$. At time t', v sends $\langle kT \rangle = \langle L_v(t') \rangle$ to all neighbors. These will receive it before time $t' + d$ and, if they have not reached clock value kT and sent a message $\langle kT \rangle$ yet, do so now. By induction, every node within D hops of v will receive a message $\langle kT \rangle$ by time $t' + dD$. As we assume G to be connected, these are all nodes.

Consider any node $w \in V$. As w sets L_w to value kT when receiving a message $\langle kT \rangle$ (unless it is already larger), we have that

$$\begin{aligned} L_w(t) &\geq L_w(t' + dD) + t - (t' + dD) \\ &\geq L_v(t') + t - (t' + dD) \\ &\geq L_v(t - dD - T) + t' - (t - dD - T) + t - (t' + dD) = L + T\,. \end{aligned}$$

As w is arbitary, it follows that

$$\mathcal{G}(t) = \max_{v \in V}\{L_v(t)\} - \min_{w \in V}\{L_w(t)\} \leq \vartheta dD + (\vartheta - 1)T\,. \qquad \square$$

Theorem 1.2. *Set* $H := \max_{v \in V}\{H_v(0)\} - \min_{v \in V}\{H_v(0)\}$. *Then Algorithm 1.1 achieves*

$$\mathcal{G} \leq \max\{H, dD\} + (\vartheta - 1)(dD + T)\,.$$

Proof. Consider $t \in \mathbb{R}_0^+$. If $t \geq dD+T$, then $\mathcal{G}(t) \leq dD+(\vartheta-1)T$ by Lemma 1.8. If $t < dD + T$, then for any $v, w \in V$ we have that

$$L_v(t) - L_w(t) \leq L_v(0) - L_w(0) + (\vartheta - 1)t \leq H + (\vartheta - 1)(dD + T)\,. \qquad \square$$

Remarks:

- H reflects the skew on initialization. Getting H small may or may not be relevant to applications, but it yields little understanding of the overall problem; hence we neglect this issue here.
- Making H part of the bound means that we do not bound $\mathcal{G}$ for all executions, as the model allows for executions with arbitrarily large initial clock offsets $H_v(0) - H_w(0)$. An unconditional bound will require to ensure that H is small — but of course this "unconditional" bound then still relies on the assumptions of the model.
- Is this algorithm good? May it even be optimal in some sense?

1.3 Lower Bound on the Global Skew

To argue that we performed well, we need to show that we could not have done (much) better (in the worst case). We will use the *shifting technique,* which enables to "hide" skew from the nodes. That is, we construct two executions which look completely identical from the perspective of all nodes, but different hardware clock values are reached at different times. No matter how the algorithm assigns logical clock values, in one of the executions the skew must be large — provided that nodes *do* increase their clocks. First, we need to state what it means that two executions are *indistinguishable* at a node.

Definition 1.3 (Indistinguishable Executions)**.** *Executions* $\mathcal{E}_0$ *and* $\mathcal{E}_1$ *are* indistinguishable at node $v \in V$ until local time H, *if* $H_v^{(\mathcal{E}_0)}(0) = H_v^{(\mathcal{E}_1)}(0)$ *(where the superscripts indicate the execution) and, for* $i \in \{0, 1\}$*, for each message* v *receives at local time* $H' \leq H$ *in* $\mathcal{E}_i$ *from some neighbor* $w \in V$*, it receives an identical message from* w *at local time* H' *in* $\mathcal{E}_{1-i}$*. If we drop the "until local time* H*," this means that the statement holds for all* H*, and if we drop the "at node* v*," the statement holds for all nodes.*

Remarks:

- If two executions are indistinguishable until local time H at $v \in V$, it sends the same messages in both executions and computes the same logical clock values — in terms of its *local* time — until local time H. This holds because our algorithms are deterministic and all actions nodes take are determined by their local perception of time and which messages they received (and when).
- As long as we can ensure that the receiver of each message receives it at the same *local* time in two executions without violating the constraint that messages are under way between $d-u$ and d *real* time in both executions, we can inductively maintain indistinguishability: as long as this condition is never violated, each node will send the same messages in both executions at the same hardware times.

Before showing that we cannot avoid a certain global skew, we need to add a requirement, namely that clocks actually behave like clocks and make progress. Note that, without such a constraint, setting $L_v(t) = 0$ at all $v \in V$ and times t is a "perfect" solution for the clock synchronization problem.

Definition 1.4 (Amortized Minimum Progress)**.** *For $\alpha \in \mathbb{R}^+$, an algorithm satisfies the* amortized α-progress condition, *if there is some $C \in \mathbb{R}_0^+$ such that $\min_{v\in V}\{L_v(t)\} \geq \alpha t - C$ for all $t \in \mathbb{R}_0^+$ and all executions.*

We now prove that we cannot only "hide hardware clock skew," but also keep nodes from figuring out that they might be able to advance their logical clocks slower than their hardware clocks in such executions.

Lemma 1.5. *Fix an arbitrary algorithm and any node $v \in V$. For arbitrarily small $\varepsilon > 0$, there are executions $\mathcal{E}_v$ and $\mathcal{E}_1$ that are indistinguishable such that*

- $H_x^{(\mathcal{E}_1)}(t) = t$ *for all $x \in V$ and t,*
- $H_v^{(\mathcal{E}_v)}(t) = H_v^{(\mathcal{E}_1)}(t) + uD - \varepsilon$ *for all $t \geq t_0 \coloneqq \frac{uD-\varepsilon}{\rho-1}$ for some $\rho \in (1, \vartheta]$,*
- $H_w^{(\mathcal{E}_v)}(t) = t$ *for some $w \in V$ and all t.*

Proof. In both executions and for all $x \in V$, we set $H_x(0) \coloneqq 0$. Denote by $d(x, y)$ the distance (i.e., hop count of a shortest path) between nodes x and y, and fix some node $w \in V$ with $d(v, w) = D$. Abbreviate $d(x) \coloneqq d(x, w)-d(x, v)$. Execution $\mathcal{E}_1$ is given by running the algorithm will all hardware clock rates being 1 at all times and the message delay from x to y being $d-(\frac{1}{2}-\frac{d(x)-d(y)}{4})u$.

Observe that $d(x) \in [-D, D]$, where $d(v) = D$ and $d(w) = -D$, and that $d(\cdot)$ differs by at most 2 between neighbors. In $\mathcal{E}_v$, we set the hardware clock rate of node $x \in V$ to $1 + \frac{(\rho-1)(d(x)+D)}{2D}$ at all times $t \leq t_0$ and 1 at all times $t > t_0$ (we will specify $\rho \in (1, \vartheta)$ later). This implies that

$$H_v^{(\mathcal{E}_v)}(t_0) = \rho t_0 = t_0 + (\rho - 1)t_0 = t_0 + uD - \varepsilon = H_v^{(\mathcal{E}_1)}(t_0) + uD - \varepsilon \quad \text{and}$$
$$H_w^{(\mathcal{E}_v)}(t_0) = t_0\,.$$

As clock rates are 1 from time t_0 on, this means that the hardware clocks satisfy all stated constraints.

It remains to specify message delays and show that the two executions are indistinguishable. We achieve this by simply ruling that a message sent from some $x \in V$ to a neighbor $y \in V$ in $\mathcal{E}_v$ arrives at the same local time at y as it does in $\mathcal{E}_1$. By induction over the arrival sending times of messages, then indeed all nodes also send identical messages at identical local times in both executions, i.e., the executions remain indistinguishable at all nodes and times. However, it remains to prove that this results in all message delays being in the range $(d-u, d)$.

To see this, recall that for any $\{x, y\} \in E$, we have that $|d(x) - d(y)| \leq 2$. As clock rates are 1 after time t_0 and constant before, and all hardware clocks are 0 at time 0, the maximum difference between any two local times between neighbors is attained at time t_0. We compute

$$H_x^{(\mathcal{E}_v)}(t_0) - H_y^{(\mathcal{E}_v)}(t_0) = \frac{d(x) - d(y)}{2D} \cdot (\rho - 1)t_0 = \frac{d(x) - d(y)}{2} \cdot \left(u - \frac{\varepsilon}{D}\right).$$

In execution $\mathcal{E}_1$, a message sent from x to y at local time $H_x^{(\mathcal{E}_1)}(t) = t$ is received at local time $H_y^{(\mathcal{E}_1)}(t) = H_x^{(\mathcal{E}_1)}(t) + d - (\frac{1}{2} - \frac{d(x)-d(y)}{4})u$. If a message is sent at time t in $\mathcal{E}_v$, we have that

$$\begin{aligned}
&H_y^{(\mathcal{E}_v)}(t+d) \\
&\geq H_y^{(\mathcal{E}_v)}(t) + d \\
&= H_x^{(\mathcal{E}_v)}(t) + d + \frac{d(x) - d(y)}{2} \cdot \left(u - \frac{\varepsilon}{D}\right) \\
&= H_x^{(\mathcal{E}_1)}(t) + d - \left(\frac{1}{2} - \frac{d(x) - d(y)}{4}\right) u + \frac{2 + d(x) - d(y)}{4} \cdot u - \frac{(d(x) - d(y))\varepsilon}{2D} \\
&> H_x^{(\mathcal{E}_1)}(t) + d - \left(\frac{1}{2} - \frac{d(x) - d(y)}{4}\right) u
\end{aligned}$$

where the last inequality uses that $d(x) - d(y) \geq -2$ and assumes that $\varepsilon < uD$, i.e., ε is sufficiently small. On the other hand, as clock rates in $\mathcal{E}_v$ are at most ρ,

$$\begin{aligned}
&H_y^{(\mathcal{E}_v)}(t+d-u) \\
&\leq H_y^{(\mathcal{E}_v)}(t) + \rho d - u \\
&= H_x^{(\mathcal{E}_v)}(t) + \rho d - u + \frac{d(x) - d(y)}{2} \cdot \left(u - \frac{\varepsilon}{D}\right) \\
&= H_x^{(\mathcal{E}_1)}(t) + \rho d - \left(\frac{1}{2} - \frac{d(x) - d(y)}{4}\right) u + \frac{d(x) - d(y) - 2}{4} u - \frac{(d(x) - d(y))\varepsilon}{2D}.
\end{aligned}$$

We want to bound this term by $H_x^{(\mathcal{E}_1)}(t) + d - \left(\frac{1}{2} - \frac{d(x)-d(y)}{4}\right) u$, which is equivalent to requiring that

$$(\rho - 1)d + \frac{d(x) - d(y) - 2}{4} \cdot u - \frac{(d(x) - d(y))\varepsilon}{2D} < 0\,.$$

We are still free to choose ρ from $(1, \vartheta]$. We set $\rho := 1 + \frac{\varepsilon}{2dD}$, implying that the left hand side is smaller than 0 if $d(x) - d(y) = 2$. The other case is that $d(x) - d(y) \leq 1$, and choosing ε (and thus also $\rho - 1$) sufficiently close to 0 ensures that the inequality holds. □

Theorem 1.6. *If an algorithm satisfies the amortized α-progress condition for some $\alpha \in \mathbb{R}^+$, then $\mathcal{G} \geq \frac{\alpha u D}{2}$, even if we are guaranteed that $H_v(0) = 0$ for all $v \in V$.*

Proof. From Lemma 1.5, for arbitrarily small $\varepsilon > 0$ we have two indistinguishable executions $\mathcal{E}_v$, $\mathcal{E}_1$ and nodes $v, w \in V$ such that

- $H_v^{(\mathcal{E}_1)}(t) = H_w^{(\mathcal{E}_1)}(t) = H_w^{(\mathcal{E}_v)}(t) = t$ for all $t \in \mathbb{R}_0^+$ and
- there is a time t_0 such that $H_v^{(\mathcal{E}_v)}(t) = t + uD - \varepsilon$ for all $t \geq t_0$.

Because the algorithm satisfies the amortized α-progress condition, we have that $L_v^{(\mathcal{E}_1)}(t) \geq \alpha t - C$ for all t and some $C \in \mathbb{R}_0^+$. We claim that there is some $t \geq t_0$ satisfying that

$$L_w^{(\mathcal{E}_1)}(t + uD - \varepsilon) - L_w^{(\mathcal{E}_1)}(t) \geq \alpha(uD - 2\varepsilon)\,. \tag{1.1}$$

Assuming for contradiction that this is false, set $\rho := \frac{\alpha(uD-2\varepsilon)}{uD-\varepsilon} < \alpha$ and consider times $t := t_0 + k(uD - \varepsilon)$ for $k \in \mathbb{N}$. We get that

$$L_w^{(\mathcal{E}_1)}(t) \leq L_w^{(\mathcal{E}_1)}(t_0) + \rho(t - t_0) = \alpha(t - t_0) - (\alpha - \rho)(t - t_0) + L_w^{(\mathcal{E}_1)}(t_0)\,.$$

Choosing $t > \frac{L_w^{(\mathcal{E}_1)}(t_0)+C}{\alpha-\rho}$, we get that $L_w^{(\mathcal{E}_1)}(t) < \alpha t - C$, violating the α-progress condition. Thus, we reach a contradiction, i.e., the claim must hold true.

Now let $t \geq t_0$ be such that (1.1) holds. As $H_w^{(\mathcal{E}_1)}(t) = H_w^{(\mathcal{E}_v)}(t)$, by indistinguishability of $\mathcal{E}_1$ and $\mathcal{E}_v$ we have that $L_w^{(\mathcal{E}_1)}(t) = L_w^{(\mathcal{E}_v)}(t)$. As $H_v^{(\mathcal{E}_1)}(t+uD-\varepsilon) = t + uD - \varepsilon = H_v^{(\mathcal{E}_v)}(t)$, we have that $L_v^{(\mathcal{E}_v)}(t) = L_v^{(\mathcal{E}_1)}(t + uD - \varepsilon)$. Hence,

$$\begin{aligned}
&L_w^{(\mathcal{E}_1)}(t + uD - \varepsilon) - L_v^{(\mathcal{E}_1)}(t + uD - \varepsilon) \\
&\geq L_w^{(\mathcal{E}_1)}(t) + \alpha(uD - 2\varepsilon) - L_v^{(\mathcal{E}_1)}(t + uD - \varepsilon) \\
&= L_w^{(\mathcal{E}_v)}(t) - L_v^{(\mathcal{E}_v)}(t) + \alpha(uD - 2\varepsilon)\,.
\end{aligned}$$

We conclude in at least one of the two executions, the logical clock difference between v and w reaches at least $\frac{\alpha u D}{2} - \varepsilon$. As $\varepsilon > 0$ can be chosen arbitrarily small, it follows that $\mathcal{G} \geq \frac{\alpha u D}{2}$, as claimed. □

Remarks:

- The good news: We have a lower bound on the skew that is linear in D. The bad news: typically $u \ll d$, so we might be able to do much better.
- When propagating information, we haven't factored in yet that we *know* that messages are under way for at least $d - u$ time. Let's exploit this!

1.4 Refining the Max Algorithm

Lemma 1.7. *In a system executing Algorithm 1.2, no $v \in V$ ever sets L_v to a value larger than* $\max_{w \in V \setminus \{v\}}\{L_w(t)\}$.

Algorithm 1.2: Refined Max Algorithm.

```
1 L_v(0) := H_v(0)
2 at all times, increase L_v at the rate of H_v
3 if received ⟨L⟩ at time t and L > L_v(t) then
4 |   L_v(t) := max{L_v(t), L + d − u}
5 if H_v(t) = kT for some k ∈ ℕ then
6 |   send ⟨L_v(t)⟩ to all neighbors
```

Proof. If any node $v \in V$ sends message $\langle L_v(t) \rangle$ at time t, it is not received before time $t + d - u$, for which it holds that

$$\max_{w \in V} \{L_w(t + d - u)\} \geq L_v(t + d - u) \geq L_v(t) + d - u \,,$$

as all nodes, in particular v, increase their logical clocks at least at rate 1, the minimum rate of increase of their hardware clocks. □

Lemma 1.8. *In a system executing Algorithm 1.2, it holds that*

$$\mathcal{G}(t) \leq ((\vartheta - 1)(d + T) + u)D$$

for all $t \geq (d + T)D$, where D is the diameter of G.

Proof. Set $L := \max_{v \in V} \{L_v(t - (d + T)D)\}$. By Lemma 1.7 and the fact that hardware clocks increase at rate at most ϑ, we have that

$$\max_{v \in V} \{L_v(t)\} \leq \max_{v \in V} \{L_v(t - (d + T)D)\} + \vartheta(d + T)D = L + \vartheta(d + T)D \,.$$

Consider any node $w \in V$. We claim that $L_w(t) \geq L + (d + T - u)D$, which implies

$$\max_{v \in V} \{L_v(t)\} - L_w(t) \leq L + \vartheta(d+T)D - (L + (d+T-u)D) = ((\vartheta - 1)(d+T) + u)D \,;$$

as w is arbitary, this yields the statement of the lemma.

It remains to show the claim. Let $v \in V$ be such that $L_v(t - (d+T)D) = L$. Denote by $(v_{D-h} = v, v_{D-h+1}, \ldots, v_D = w)$, where $h \leq D$, a shortest v-w-path. Define $t_i := t - (D - i)(d + T)$. We prove by induction over $i \in \{D - h, D - h + 1, \ldots, D\}$ that

$$L_{v_i}(t_i) \geq L + i(d + T - u) \,,$$

where the base case $i = D - h$ is readily verified by noting that

$$L_v(t_i) \geq L_v(t - (d + T)D) + t_i - (t - (d + T)D) = L + i(d + T) \,.$$

For the induction step from $i - 1 \in \{D - h, \ldots, D - 1\}$ to i, observe that v_{i-1} sends a message to v_i at some time $t_s \in (t_{i-1}, t_{i-1} + T]$, as its hardware clock increases by at least T in this time interval. This message is received by v_i at some time $t_r \in (t_s, t_s + d) \subseteq (t_{i-1}, t_{i-1} + d + T)$. Note that $t_{i-1} < t_s < t_r < t_i$.

If necessary, v_i will increase its clock at time t_r, ensuring that

$$\begin{aligned} L_{v_i}(t_i) &\geq L_{v_i}(t_r) + t_i - t_r \\ &\geq L_{v_{i-1}}(t_s) + d - u + t_i - t_r \\ &\geq L_{v_{i-1}}(t_s) + t_i - t_s - u \\ &\geq L_{v_{i-1}}(t_{i-1}) + t_i - t_{i-1} - u \\ &= L_{v_{i-1}}(t_{i-1}) + d + T - u \\ &\geq L + i(d + T - u)\,, \end{aligned}$$

where the last step uses the induction hypothesis. This completes the induction. Inserting $i = D$ yields that $L_w(t) \geq L_{v_D}(t_D) = L + (d + T - u)D$, as claimed, completing the proof. □

Theorem 1.9. *Set* $H \coloneqq \max_{v \in V}\{H_v(0)\} - \min_{v \in V}\{H_v(0)\}$. *Then Algorithm 1.2 achieves*

$$\mathcal{G} \leq \max\{H, uD\} + (\vartheta - 1)(d + T)D\,.$$

Proof. Consider $t \in \mathbb{R}_0^+$. If $t \geq (d+T)D$, then $\mathcal{G}(t) \leq uD + (\vartheta - 1)(d+T)D$ by Lemma 1.8. If $t < (d + T)D$, then for any $v, w \in V$ we have that

$$L_v(t) - L_w(t) \leq L_v(0) - L_w(0) + (\vartheta - 1)t \leq H + (\vartheta - 1)(d + T)D\,. \qquad \square$$

Remarks:

- Note the change from using logical clock values to hardware clock values to decide when to send a message. The reason is that increasing received clock values to account for minimum delay pays off only if the increase is also forwarded in messages. However, sending a message every time the clock is set to a larger value might cause a lot of messages, as now different values than kT for some $k \in \mathbb{N}$ might be sent. The compromise presented here keeps the number of messages in check, but pays for it by exchanging the $(\vartheta - 1)T$ term in skew for $(\vartheta - 1)TD$.
- Choosing $T \in \Theta(d)$ means that nodes need to send messages roughly every d time, but in return $\mathcal{G} \in \max\{H, uD\} + \mathcal{O}((\vartheta - 1)dD)$. Reducing T further yields diminishing returns.
- Typically, $u \ll d$, but also $\vartheta - 1 \ll 1$. However, if $u \ll (\vartheta - 1)d$, one might consider to build a better clock by bouncing messages back and forth between pairs of nodes. Hence, this setting makes only sense if communication is expensive or unreliable, and in many cases one can expect uD to be the dominant term.
- In the exercises, you will show how to achieve a skew of $\mathcal{O}(uD + (\vartheta - 1)d)$.
- So we can say that the algorithm achieves asymptotically optimal global skew (in our model). The lower bound holds in the worst case, but we have shown that it applies to *any* graph. So, for deterministic guarantees, changing the network topology has no effect beyond influencing the diameter.
- We neglected important aspects like local skew and fault-tolerance, which will keep us busy during the remainder of the course.

1.5 Afterthought: Stronger Lower Bound

Both of our algorithms are actually much more restrained in terms of clock progress than just satisfying an amortized lower bound of 1 on the rates.

Definition 1.10 (Strong Envelope Condition)**.** *An algorithm satisfies the* strong envelope condition, *if at all times and for all nodes $v \in V$, it holds that* $\min_{w\in V}\{H_w(t)\} \leq L_v(t) \leq \max_{w\in V}\{H_w(t)\}$.

Corollary 1.11. *For any algorithm satisfying the strong envelope condition, it holds that $\mathcal{G} \geq uD$, even if we are guaranteed that $H_v(0) = 0$ for all $v \in V$.*

Proof. Apply Lemma 1.5 for some $v \in V$ and $\varepsilon > 0$. We have that $H_x^{\mathcal{E}_1}(t) = t$ for all $x \in V$. The strong envelope condition thus entails that $L_x^{\mathcal{E}_1}(t) = H_x^{\mathcal{E}_1}(t) = t$ for all x and t. As $\mathcal{E}_v$ is indistinguishable from $\mathcal{E}_1$, it follows that also $L_x^{\mathcal{E}_v}(t) = H_x^{\mathcal{E}_v}(t)$ for all x and t. In particular, there is some $w \in V$ such that

$$L_v^{\mathcal{E}_v}(t_0) - L_w^{\mathcal{E}_w}(t_0) = uD - \varepsilon\,.$$

As this holds for arbitrarily small $\varepsilon > 0$, we conclude that indeed $\mathcal{G} \geq uD$, as claimed. □

Remarks:

- Thus, in some sense the term uD in the skew bound is optimal.
- If one merely requires the weaker bound $t \leq L_v(t) \leq \max_{v\in V}\{H_v(0)\}+\vartheta t$, then a lower bound of $\frac{uD}{\vartheta}$ can be shown.
- Playing with such progress conditions is usually of limited relevance, as one cannot gain more than a factor of 2 — unless one is willing to simply slow down everything.

What to Take Home

- The shifting technique is an important source of lower bounds. We will see it again.
- If all that we're concerned with is the global skew and we have no faults, things are easy.
- There are other communication models, giving slightly different results. However, in a sense, our model satisfies the minimal requirements to be different from an asynchronous system (in which nodes have no meaningful sense of time): They can measure time with some accuracy, and messages cannot be delayed arbitrarily.
- The linear lower bound on the skew is highly resilient to model variations. If delays are distributed randomly and independently, a probabilistic analysis yields skews proportional to roughly $\sqrt{D}$, though (for most of the time). This is outside the scope of this lecture series.

Bibliographic Notes

The shifting technique was introduced by Lundelius and Lynch, who show that even if the system is fully connected, there are no faults, and there is no drift (i.e., $\vartheta = 1$), better synchronization than $\left(1 - \frac{1}{n}\right) u$ cannot be achieved [LL84]. Biaz and Lundelius Welch generalized the lower bound to arbitrary networks [BW01]. Note that Jennifer Lundelius and Jennifer Lundelius Welch are the same person — and the double name "Lundelius Welch" will be frequently cited as Welch (as "Lundelius" will be treated as a middle name, both by typesetting systems and people who don't know otherwise). I will stick to "Welch" as well, but for a different reason: "the Lynch-Lundelius-Welch algorithm" is a mouthful, and "the Lynch-Welch algorithm" rolls off the tongue much better (I hope that I'll be forgiven if she ever finds out!).

As far as I know, the max algorithm has been mentioned first in writing by Locher and Wattenhofer [LW06] — but not because it is such a good synchronization algorithm, but rather due its terrible performance when it comes to the skew between neighboring nodes (see excersise). Being an extremely straightforward solution, it is likely to appear earlier and in other places and should be considered folklore. In contrast to the earlier works mentioned above (and many more), [LW06] uses a model in which clocks drift, just like in this lecture. At least for this line of work, this goes back to a work by Fan and Lynch on *gradient clock synchronization,* [FL06] which shows that it is *not* possible to distribute the global skew of $\Omega(uD)$ "nicely" so that the skew between adjacent nodes is $\mathcal{O}(u)$ at all times; the possibility to "introduce skew on the fly" is essential for this observation. More on this in the next two lectures!

Bibliography

[BW01] Saâd Biaz and Jennifer Lundelius Welch. Closed Form Bounds for Clock Synchronization under Simple Uncertainty Assumptions. *Information Processing Letters*, 80:151–157, 2001.

[FL06] Rui Fan and Nancy Lynch. Gradient Clock Synchronization. *Distributed Computing*, 18(4):255–266, 2006.

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

[LW06] Thomas Locher and Roger Wattenhofer. Oblivious Gradient Clock Synchronization. In *Proc. 20th Symposium on Distributed Computing (DISC)*, pages 520–533, 2006.