Lecture 1

Synchronizing Clocks

In this lecture series, we consider fault-tolerant clock generation and distribution from a theoretical perspective. We 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 many of the ideas and concepts can be used in the design of highly reliable and scalable hardware solutions. The first lecture introduces the basic clock synchronization task and provides rudimentary algorithms and analysis. More refined questions will prompt more refined answers later in the course. Nonetheless, the initial lecture offers a taste 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 **??**), 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 v is equipped with a *hardware clock*, denoted H_v. The goal of clock synchronization is for each node to compute a *logical clock*, denoted L_v, such that all logical clocks remain as closely synchronized as possible. The challenge of clock synchronization arises as a result of uncertainty in the system. Specifically, the basic model has two forms of uncertainty: uncertainty in the hardware clock rate, called *clock drift*, and uncertainty in the transit times of messages between nodes.

For every execution, we assume that there is an *objective* "true" Newtonian time taking values in $\mathbb{R}_0^+$. We will typically denote objective times by the variables t and t'. Objective time allows us to define (and reason about) the global state of the system at any given instant, but the objective time is never known to any processor.

For each node $v \in V$, we model v's hardware clock as a strictly increasing function $H_v \colon \mathbb{R}_0^+ \to \mathbb{R}_0^+$. We assume that H_v increases at a rate between 1 and $\vartheta > 1$:

$$\forall v \in V, t, t' \in \mathbb{R}_0^+, t \geq t' \colon t - t' \leq H_v(t) - H_v(t') \leq \vartheta(t - t')\,, \qquad (1.1)$$

where $t, t' \in \mathbb{R}_0^+$ denote objective times. For simplicity, we assume that hardware clocks are differentiable and denote the derivative by h_v.[1] We call $h_v(t)$ the (instantaneous) *rate* of v at objective time t. Observe that Equation (1.1) implies that $h_v(t) \in [1, \vartheta]$ at all times t. The parameter ϑ—an upper bound on the rates of all hardware clocks—is known to the algorithm designer, however processes have no way of learning the values of $h_v(t)$ directly. Thus any possible (differentiable) hardware clock values H_v satisfying (1.1) are admissible, and a good clock synchronization algorithm should maintain synchronization for all possible H_v without knowledge of the rates h_v beyond what is implied by (1.1). We 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 it will take for v's message to be delivered to w. A message sent at objective 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*. The delay d subsumes delays due to local computations, etc. That is, in our model, at the time t' when the message is received, all updates to the state of the receiving node take effect immediately, and messages it sends in response may also be sent immediately.

An *event* consists of (1) a node sending or receiving a message, or (2) a node's hardware clock reaching some prescribed value (possibly determined in response to a previous event). Every event e seen by a node v has both an associated objective time t_e when the event occurs, and an associated hardware time $H_v(t_e)$ when v witnesses the event. The *state* of a node at objective time t consists of the entire history of events witnessed by v up to time t along with the associated hardware times at which v witnessed the evenets, as well as the current hardware time $H_v(t)$. Informally, an *algorithm* specifies when and how a node responds to each event it sees, given its current state when the event occurs. We assume that an algorithm produces (and hence witnesses) a finite number of events in every bounded interval of time, but we do not make any other assumptions about nodes' local computations.

An *execution* of an algorithm on a system specifies hardware clock functions H_v as above for each $v \in V$, and assigns to each event e an objective time t_e at which the event occurs. In particular, a message sent by v at objective time t must be received at time $t' \in [t + d - u, t + d]$. Since an algorithm only produces finitely many events in any bounded interval of time, there is an increasing sequence of times $t_1 < t_2 < t_3 < \cdots$ at which some event(s) occur (at any node). Further, the state of the system at these times is defined inductively: given the execution and states of processors at time t_i for $i \geq 1$, one can determine the time t_{i+1} at which the next event(s) occur, as well as the state of the system at this time.

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

[1]All of the claims we make can be derived from (1.1) without the assumption of differentiability, but this assumption simplifies our analysis.

of the node (including $H_v(t)$). The goal is to minimize the *global skew*

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

over all exicutions $\mathcal{E}$, where

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

is the *global skew at time t*. The suprema and maxima above are also taken over all possible executions. 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 that we will encounter. 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 forms of communication other than point-to-point message passing described above. However, many algorithms can be adapted to other modes of communication relatively small conceptual changes.
- Clocks may not be perfectly synchronized at objective 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).

1.2 The Max Algorithm

Let's start with our first algorithm. It is 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.

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

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

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

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 $\langle L \rangle$ at time $t$ and $L > L_v(t)$ then
4     $L_v(t) := L$
5 if $L_v(t) = kT$ for some $k \in \mathbb{N}$ then
6     send $\langle L_v(t) \rangle$ to all neighbors
```

Proof. For any time t, let $L_{\max}(t) = \max_{w \in V}\{L_w(t)\}$ be the maximum logical clock value in the system at time t. Observe that any node v satisfying $L_v(t) = L_{\max}(t)$ cannot satisfy the condition in Line 3. Therefore, $L_{\max}(t)$ increases at a rate at most ϑ (the maximum rate of any hardware clock), so that

$$L_{\max}(t') \leq L_{\max}(t) + \vartheta \cdot (t' - t) \quad \text{for all } t' > t. \tag{1.2}$$

Fix a time $t' \geq dD + T$, and let v be the node with the maximum logical clock value at time $s := t' - dD - T$. That is, $L_v(s) = L_{\max}(s)$. Applying (1.2), we find that

$$L_{\max}(t') \leq L_{\max}(s) + \vartheta \cdot (t' - s) = L_{\max}(s) + \vartheta \cdot (dD - T). \tag{1.3}$$

To finish the proof, it suffices to show that at time t', all nodes $w \in V$ satisfy $L_w(t') \geq L_{\max}(s)$. To this end observe that v's logical clock increases at a rate of at least 1, so there exists a time $s' \in [s, s+T]$ such that $L_v(s') = kT \geq L_v(s)$ for some integer $k \in \mathbb{N}$.[2] At time s', v sends the message $\langle kT \rangle$ to all of its neighbors in accordance with line 6. This message is received by all of v's neighbors by time $s' + d$, hence by time $s' + d$, all of v's neighbors' logical clocks are at least $L_{\max}(s') \geq L_{\max}(s)$. Continuing in this way, a straightforward induction argument shows that for all $\ell \in \mathbb{N}$ and all nodes w within distance ℓ from v will satisfy $L_w(s' + \ell \cdot d) \geq L_{\max}(s)$. In particular, taking $\ell = D$ (the network diameter), we find that for all $w \in V$

$$L_{\max}(s) \leq L_{\max}(s') \leq L_w(s' + D \cdot d) \leq L_w(t')$$

which gives the desired result. $\square$

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 $\mathcal{G}(t) \leq \vartheta dD + (\vartheta - 1)T$ by Lemma 1.1. If $t < dD + T$, then for any $v, w \in V$ we have that

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

[2]It could be the case that L_v reaches kT because v received a message from some other node v' that overtook v as the fastest node in the network. However, our argument only relies on the fact that L_v reaches an integer multiple $kT \geq L_v(s)$ at some time in the interval $[s, s + T]$.

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 some nodes $v, w \in V$ and $\rho \in (1, \vartheta)$ such that $(\rho - 1)d < u/2$, and set $t_0 := d(v,w)(u/(2(\rho-1)) - d)$. For any algorithm, there are indistinguishable executions $\mathcal{E}_1$ and $\mathcal{E}_v$ satisfying that*

- *$H_x^{(\mathcal{E}_1)}(t) = t$ for all $x \in V$ and t,*
- *$H_v^{(\mathcal{E}_v)}(t) = H_v^{(\mathcal{E}_1)}(t) + d(v,w)(u/2 - (\rho-1)d)$ for all $t \geq t_0$,*
- *$H_w^{(\mathcal{E}_v)}(t) = t$ for all t, and*
- *$\mathcal{E}_1$ does not depend on the choice of v and w.*

Proof. In both executions and for all $x \in V$, we set $H_x(0) := 0$. Execution $\mathcal{E}_1$ is given by running the algorithm with all hardware clock rates being 1 at all times and the message delay from x to y being $d - u/2$.

Set

$$d(x) := \begin{cases} -d(v,w) & \text{if } d(x,w) - d(x,v) < -d(v,w) \\ d(v,w) & \text{if } d(x,w) - d(x,v) > d(v,w) \\ d(x,w) - d(x,v) & \text{else.} \end{cases}$$

Note that $|d(x) - d(y)| \leq 2$ for any $\{x,y\} \in E$. Moreover, $d(v) = d(v,w)$ and $d(w) = -d(v,w)$. In $\mathcal{E}_v$, we set the hardware clock rate of node $x \in V$ to $1 + (\rho-1)(d(x) + d(v,w))/(2d(v,w))$ at all times $t \leq t_0$ and to 1 at all times $t > t_0$. This implies that

$$H_v^{(\mathcal{E}_v)}(t_0) = \rho t_0 = H_v^{(\mathcal{E}_1)}(t_0) + d(v,w)\left(\frac{u}{2} - (\rho-1)d\right) \quad \text{and}$$
$$H_w^{(\mathcal{E}_v)}(t_0) = t_0 = H_w^{(\mathcal{E}_1)}(t_0)\,.$$

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 $y \in N_x$ in $\mathcal{E}_v$ arrives at the same local time at y as it does in $\mathcal{E}_1$. By induction over the arrival and sending times of messages, then indeed all nodes also send identical messages at identical local times in both executions, i.e., the executions are indistinguishable. However, it remains to prove that this results in all message delays being in the range $(d-u, d)$.

To see this, fix a time t and set $\lambda := \max\{t/t_0, 1\}$. We compute

$$H_x^{(\mathcal{E}_v)}(t) - H_y^{(\mathcal{E}_v)}(t) = \frac{d(y) - d(x)}{2d(v,w)} \cdot (\rho-1)\lambda t_0 = \lambda \cdot \frac{d(y)-d(x)}{2}\left(\frac{u}{2} - (\rho-1)d\right).$$

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 $H_y^{(\mathcal{E}_1)}(t + d - u/2) = H_x^{(\mathcal{E}_1)}(t) + d - u/2$. Thus, showing that

$H_y^{(\mathcal{E}_v)}(t+d-u) < H_x^{(\mathcal{E}_v)}(t) + d - u/2 < H_x^{(\mathcal{E}_v)}(t) + d$ will complete the proof. Recall that ρ is such that $u/2 - (\rho-1)d > 0$. We have that

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

where the second to last inequality uses that $d(x) - d(y) \geq -2$ and $0 \leq \lambda \leq 1$. On the other hand,

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

where the second inequality uses that $d(x) - d(y) \leq 2$ and $0 \leq \lambda \leq 1$. □

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. Fix $v, w \in V$ such that $d(v,w) = D$ and set $\rho \in (1, \vartheta)$ such that $(\rho-1)d < u/2$. In the following, we abbreviate $\varepsilon := (\rho-1)d$; note that we can choose $\varepsilon > 0$ arbitrarily small by picking ρ accordingly. We apply Lemma 1.5 twice, where the second time we reverse the roles of v and w. As $\mathcal{E}_1$ does not depend on the choice of v and w and indistinguishability of executions is transitive, we get two indistinguishable executions $\mathcal{E}_v$ and $\mathcal{E}_w$ such that there is a time t_0 satisfying for all $t \geq t_0$ that

- $H_v^{(\mathcal{E}_w)}(t) = H_w^{(\mathcal{E}_v)}(t) = t$ and
- $H_v^{(\mathcal{E}_v)}(t) = H_w^{(\mathcal{E}_w)}(t) = t + (u/2 - \varepsilon)D$.

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

$$L_v^{(\mathcal{E}_w)}(t+(u/2-\varepsilon)D) - L_v^{(\mathcal{E}_w)}(t) + L_w^{(\mathcal{E}_v)}(t+(u/2-\varepsilon)D) - L_w^{(\mathcal{E}_v)}(t) \geq \alpha(u-3\varepsilon)D\,. \tag{1.4}$$

Assuming for contradiction that this is false, set $0 < 2\alpha' := \frac{\alpha(u-3\varepsilon)D}{(u/2-\varepsilon)D} < 2\alpha$ and consider times $t_k := t_0 + k(u/2-\varepsilon)D$ for $k \in \mathbb{N}$. By induction over k, we get that

$$\begin{aligned} L_v^{(\mathcal{E}_w)}(t_k) + L_w^{(\mathcal{E}_v)}(t_k) &\leq L_v^{(\mathcal{E}_w)}(t_0) + L_w^{(\mathcal{E}_v)}(t_0) + 2\alpha'(t_k - t_0) \\ &\leq 2\alpha t_k - 2(\alpha - \alpha')t_k + L_v^{(\mathcal{E}_w)}(t_0) + L_w^{(\mathcal{E}_v)}(t_0)\,. \end{aligned}$$

Choosing k large enough so that $t_k > (L_v^{(\mathcal{E}_w)}(t_0) + L_w^{(\mathcal{E}_v)}(t_0) + 2C)/(2(\alpha - \alpha'))$, we get that

$$L_v^{(\mathcal{E}_w)}(t_k) + L_w^{(\mathcal{E}_v)}(t_k) < 2(\alpha t_k - C)\,.$$

Therefore, $L_v^{(\mathcal{E}_w)}(t_k) < \alpha t_k - C$ or $L_w^{(\mathcal{E}_v)}(t_k) < \alpha t_k - C$, violating the α-progress condition in at least one of the executions. This is a contradiction, i.e., the claim must hold true.

Now let $t \geq t_0$ be such that (1.4) holds. As $H_v^{(\mathcal{E}_w)}(t + (u/2 - \varepsilon)D) = t + (u/2 - \varepsilon)D = H_v^{(\mathcal{E}_v)}(t)$, by indistinguishability of $\mathcal{E}_v$ and $\mathcal{E}_w$ we have that $L_v^{(\mathcal{E}_v)}(t) = L_v^{(\mathcal{E}_w)}(t+(u/2-\varepsilon)D)$. Symetrically, $L_w^{(\mathcal{E}_w)}(t) = L_w^{(\mathcal{E}_v)}(t+(u/2-\varepsilon)D)$. Hence,

$$\begin{aligned}
&|L_v^{(\mathcal{E}_v)}(t) - L_w^{(\mathcal{E}_v)}(t)| + |L_v^{(\mathcal{E}_w)}(t) - L_w^{(\mathcal{E}_w)}(t)| \\
\geq\ & L_v^{(\mathcal{E}_v)}(t) - L_w^{(\mathcal{E}_v)}(t) + L_w^{(\mathcal{E}_w)}(t) - L_v^{(\mathcal{E}_w)}(t) \\
=\ & L_v^{(\mathcal{E}_w)}(t + (u/2 - \varepsilon)D) - L_w^{(\mathcal{E}_v)}(t) + L_w^{(\mathcal{E}_v)}(t + (u/2 - \varepsilon)D) - L_v^{(\mathcal{E}_w)}(t) \\
=\ & L_v^{(\mathcal{E}_w)}(t + (u/2 - \varepsilon)D) - L_v^{(\mathcal{E}_w)}(t) + L_w^{(\mathcal{E}_v)}(t + (u/2 - \varepsilon)D) - L_w^{(\mathcal{E}_v)}(t) \\
\geq\ & \alpha(u - 3\varepsilon)D\,.
\end{aligned}$$

We conclude that in at least one of the two executions, the logical clock difference between v and w reaches at least $(\alpha(u - 3\varepsilon)D)/2$. 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

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 $\langle L \rangle$ at time $t$ and $L + d - u > L_v(t)$ then
4 |  $L_v(t) := L + d - u$
5 if $H_v(t) = kT$ for some $k \in \mathbb{N}$ then
6 |  send $\langle L_v(t) \rangle$ to all neighbors
```

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

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. $\square$

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 $(v_{D-h} = v, v_{D-h+1}, \dots, v_D = w)$, where $h \leq D$, a shortest v-w-path. Define $t_i := t - (D - i)(d + T)$. We prove by induction over $i \in \{D - h, D - h + 1, \dots, D\}$ 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

$$L_v(t_{D-h}) \geq L_v(t - (d + T)D) + t_{D-h} - (t - (d + T)D) = L + (D - h)(d + T)\,.$$

For the induction step from $i - 1 \in \{D - h, \dots, D - 1\}$ 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. $\square$

Theorem 1.9. *Set* $H := \max_{v \in V}\{H_v(0)\} - \min_{v \in V}\{H_v(0)\}$. *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 fulfills the* strong envelope condition, *if at all times and for all nodes $v \in V$, it holds that $\min_{w \in V}\{H_w(t)\} \leq L_v(t) \leq \max_{w \in V}\{H_w(t)\}$.*

Theorem 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 sketch. It is possible to adapt Lemma 1.5 such that the execution $\mathcal{E}_1$ is not using delays of roughly $u/2$ between any pair of nodes, but delays are roughly $d - u$ when messages are sent “in direction of w” and d when they are sent "in direction of v." This is very similar to the use of $d(x)$ in $\mathcal{E}_v$, but we use the uncertainty "the other way round." This implies that the hardware clock difference at v between $\mathcal{E}_1$ and $\mathcal{E}_v$ can be increased to about uD (as opposed to only $uD/2$) before we run out of slack in the delays. However, in $\mathcal{E}_1$ still $H_x(t) = t$ for all $x \in V$ and times t, so nodes must maintain that $H_x(t) = L_x(t)$ in $\mathcal{E}_1$ to satisfy the strong envelope condition. Because $\mathcal{E}_v$ is indistinguishable from $\mathcal{E}_1$, the is true in $\mathcal{E}_v$. In particular,

$$L_v^{\mathcal{E}_v}(t_0) - L_w^{\mathcal{E}_v}(t_0) \approx L_v^{\mathcal{E}_1}(t_0 + uD) - L_w^{\mathcal{E}_1}(t_0) = uD\,. \qquad \square$$

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.