Contents

1 Synchronizing Clocks .......................... 1
  1.1 The Clock Synchronization Problem ........ 1
  1.2 The Max Algorithm .............................. 3
  1.3 Lower Bound on the Global Skew .......... 4
  1.4 Refining the Max Algorithm ............... 8
  1.5 Afterthought: Stronger Lower Bound ..... 10

2 Gradient Clock Synchronization ............. 13
  2.1 Formalizing the Problem ..................... 13
  2.2 Averaging Protocols .......................... 14
  2.3 GCS Algorithm ................................... 16
  2.4 Analysis of the GCS Algorithm .......... 19

3 Lower Bound on the Local Skew .............. 27
  3.1 Lower Bound with Bounded Clock Rates .. 27
  3.2 Lower Bound with Arbitrary Clock Rates .. 32

4 Fault-Tolerant Clock Synchronization ........ 37
  4.1 The Pulse Synchronization Problem ....... 38
  4.2 A Variant of the Srikanth-Toueg Algorithm ... 38
  4.3 Impossibility of Synchronization for 3f ≥ n ... 42

5 Synchronizing by Approximate Agreement .... 47
  5.1 Approximate Agreement ......................... 47
  5.2 A Variant of the Lynch-Welch Algorithm ... 51

A Notation and Preliminaries .................. 57
  A.1 Numbers and Sets ............................... 57
  A.2 Graphs ........................................... 57
  A.3 Asymptotic Notation .......................... 58
  A.4 Bounding the Growth of a Maximum of Differentiable Functions 60
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 $H_v : \mathbb{R}_0^+ \rightarrow \mathbb{R}_0^+$, whose rate of increase is between $1$ and $\vartheta > 1$:

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

where $t \in \mathbb{R}_0^+$ denotes “perfect” Newtonian real time (which is not known to the nodes). For simplicity, we assume that hardware clocks are differentiable and denote the derivative by $h_v$. Thus, the above inequalities are equivalent to $h_v(t) \in [1, \vartheta]$ at all times $t$. However, all claims can be shown solely based on the above requirement.

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 $L_v : \mathbb{R}_0^+ \to \mathbb{R}_0^+$, 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

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

where

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

is the global skew at time $t$.

For simplicity, this notation does not reflect the dependence on the execution. The goal is to bound $G$ 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 \( G \). 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 \( G \) 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 \( t \).

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( L_v(0) := H_v(0) )</td>
</tr>
<tr>
<td>2</td>
<td>at all times, increase ( L_v ) at the rate of ( H_v )</td>
</tr>
<tr>
<td>3</td>
<td>if received ( \langle L \rangle ) at time ( t ) and ( L &gt; L_v(t) ) then</td>
</tr>
<tr>
<td>4</td>
<td>( L_v(t) := L )</td>
</tr>
<tr>
<td>5</td>
<td>if ( L_v(t) = kT ) for some ( k \in \mathbb{N} ) then</td>
</tr>
<tr>
<td>6</td>
<td>send ( \langle L_v(t) \rangle ) to all neighbors</td>
</tr>
</tbody>
</table>

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

\[
G(t) \leq \vartheta dD + (\vartheta - 1)T
\]

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 \( \vartheta \), 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

$$L_w(t) \geq L_w(t' + dD) + t - (t' + dD)$$

$$\geq L_v(t') + t - (t' + dD)$$

$$\geq L_v(t - dD - T) + t' - (t - dD - T) + t - (t' + dD) = L + T.$$  

As $w$ is arbitrary, it follows that

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

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

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

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

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

**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 $G$ 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 $E_0$ and $E_1$ are indistinguishable at node $v \in V$ until local time $H$, if $H_v^{(E_0)}(0) = H_v^{(E_1)}(0)$ (where the superscripts indicate the execution) and, for $i \in \{0, 1\}$, for each message $\nu$
receives at local time $H'$ \leq H in $E_i$ from some neighbor $w \in V$, it receives an identical message from $w$ at local time $H'$ in $E_{i-1}$. 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 $\alpha$-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 $E_v$ and $E_1$ that are indistinguishable such that

1. $H^{(E_1)}_x(t) = t$ for all $x \in V$ and $t$,

2. $H^{(E_v)}_v(t) = H^{(E_1)}_v(t) + uD - \varepsilon$ for all $t \geq t_0 := \frac{uD - \varepsilon}{\varepsilon}$ for some $\rho \in (1, \vartheta]$,

3. $H^{(E_v)}_w(t) = t$ for some $w \in V$ and all $t$.

**Proof.** In both executions and for all $x \in V$, we set $H_x(0) := 0$. 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 $d(x) := d(x, w) - d(x, v)$. Execution $E_1$ 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 $d - (\frac{1}{2} - \frac{d(x) - d(w)}{4})u$.

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 $E_v$, 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 \( \rho \in (1, \theta) \) later). This implies that

\[
H_{y}^{(x)}(t_0) = \rho t_0 = t_0 + (\rho - 1)t_0 = t_0 + uD - \varepsilon = H_{y}^{(x)}(t_0) + uD - \varepsilon \quad \text{and} \quad H_{u}^{(x)}(t_0) = t_0.
\]

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 \( E_v \) arrives at the same local time at \( y \) as it does in \( E_1 \). 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

\[
H_{y}^{(x)}(t_0) - H_{y}^{(x)}(t_0) = \frac{d(y) - d(x)}{2D} \cdot (\rho - 1)t_0 = \frac{d(y) - d(x)}{2D} \cdot \left( u - \frac{\varepsilon}{D} \right).
\]

In execution \( E_1 \), a message sent from \( x \) to \( y \) at local time \( H_{y}^{(x)}(t) = t \) is received at local time \( H_{y}^{(x)}(t) = H_{x}^{(x)}(t) + d - \left( \frac{1}{2} - \frac{d(x) - d(y)}{4} \right)u \). If a message is sent at time \( t \) in \( E_v \), we have that

\[
H_{y}^{(x)}(t + d) \geq H_{y}^{(x)}(t) + d
\]

\[
= H_{x}^{(x)}(t) + d - \left( \frac{1}{2} - \frac{d(x) - d(y)}{4} \right)u + \frac{2 + d(x) - d(y)}{4} \cdot u - \frac{(d(x) - d(y))\varepsilon}{2D}
\]

where the last inequality uses that \( d(x) - d(y) \geq -2 \) and assumes that \( \varepsilon < uD \), i.e., \( \varepsilon \) is sufficiently small. On the other hand, as clock rates in \( E_v \) are at most \( \rho \),

\[
H_{y}^{(x)}(t + d - u)
\]

\[
\leq H_{y}^{(x)}(t) + \rho d - u
\]

\[
= H_{x}^{(x)}(t) + \rho d - u + \frac{d(x) - d(y)}{2} \cdot \left( u - \frac{\varepsilon}{D} \right)
\]

\[
= H_{x}^{(x)}(t) + \rho d - \left( \frac{1}{2} - \frac{d(x) - d(y)}{4} \right)u + \frac{d(x) - d(y) - 2}{4} - \frac{(d(x) - d(y))\varepsilon}{2D}.
\]

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

\[
(\rho - 1)d + \frac{d(x) - d(y) - 2}{4} \cdot u - \frac{(d(x) - d(y))\varepsilon}{2D} < 0.
\]
1.4. REFINING THE MAX ALGORITHM

We are still free to choose \( \rho \) from \((1, \vartheta]\). We set \( \rho := \min\{1 + \varepsilon/(2dD), \vartheta\} \), 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 \( \varepsilon \) (and thus also \( \rho - 1 \)) sufficiently close to 0 ensures that the inequality holds.

**Theorem 1.6.** If an algorithm satisfies the amortized \( \alpha \)-progress condition for some \( \alpha \in \mathbb{R}^+ \), then \( G \geq \frac{\alpha uD}{2} \), 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 \( E_v, E_1 \) and nodes \( v, w \in V \) such that

- \( H_v^{(E_1)}(t) = H_v^{(E_2)}(t) = H_w^{(E_1)}(t) = t \) for all \( t \in \mathbb{R}^+ \) and
- there is a time \( t_0 \) such that \( H_v^{(E_1)}(t) = t + uD - \varepsilon \) for all \( t \geq t_0 \).

Because the algorithm satisfies the amortized \( \alpha \)-progress condition, we have that \( L_v^{(E_1)}(t) \geq at - C \) for all \( t \) and some \( C \in \mathbb{R}^+ \). We claim that there is some \( t \geq t_0 \) satisfying that

\[
L_v^{(E_1)}(t + uD - \varepsilon) - L_v^{(E_1)}(t) \geq \alpha(uD - 2\varepsilon).
\]

(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

\[
L_v^{(E_1)}(t) \leq L_v^{(E_1)}(t_0) + \rho(t - t_0) = \alpha(t - t_0) - (\alpha - \rho)(t - t_0) + L_v^{(E_1)}(t_0).
\]

Choosing \( t > \frac{L_v^{(E_1)}(t_0) + C}{\alpha - \rho} \), we get that \( L_v^{(E_1)}(t) < at - C \), violating the \( \alpha \)-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 \( H_w^{(E_1)}(t) = H_v^{(E_2)}(t) \), by indistinguishability of \( E_v \) and \( E_1 \) we have that \( L_v^{(E_1)}(t) = L_w^{(E_1)}(t) \). As \( H_v^{(E_1)}(t + uD - \varepsilon) = at + uD - \varepsilon = H_v^{(E_1)}(t) \), we have that \( L_v^{(E_1)}(t) = L_v^{(E_1)}(t + uD - \varepsilon) \). Hence,

\[
L_v^{(E_1)}(t + uD - \varepsilon) - L_v^{(E_1)}(t + uD - \varepsilon) \\
geq L_v^{(E_1)}(t) + \alpha(uD - 2\varepsilon) - L_v^{(E_1)}(t + uD - \varepsilon) \\
= L_v^{(E_1)}(t) - L_v^{(E_1)}(t) + \alpha(uD - 2\varepsilon).
\]

We conclude in at least one of the two executions, the logical clock difference between \( v \) and \( w \) reaches at least \( \frac{\alpha uD}{2} - \varepsilon \). As \( \varepsilon > 0 \) can be chosen arbitrarily small, it follows that \( G \geq \frac{\alpha uD}{2} \), 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 \( \max_{w \in V \setminus \{v\}} \{L_w(t)\} \).

Proof. If any node \( v \in V \) sends message \( \langle L_v(t) \rangle \) at time \( t \), it is not received before \( t + d - u \), 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 \( \vartheta \), 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 arbitrary, 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
\[
L_{v_i}(t_i) \geq L_v(t - (d + T)D) + t_i - (t - (d + T)D) = L + i(d + T).
\]
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

\[
L_{v_i}(t_i) \geq L_{v_i}(t_r) + t_i - t_r \\
\geq L_{v_i}(t_s) + d - u + t_i - t_r \\
\geq L_{v_i}(t_s) + t_i - t_s - u \\
\geq L_{v_i}(t_{i-1}) + t_i - t_{i-1} - u \\
= L_{v_i}(t_{i-1}) + d + T - u \\
\geq L + i(d + T - u),
\]

where the last step uses the induction hypothesis. This completes the induction.

Inserting \( i = D \) yields that \( L_u(t) \geq L_{v_D}(t_D) = L + (d + T - u)D \), as claimed, completing the proof.

**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 \( t \in \mathbb{R}_+^+ \). 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

\[
L_v(t) - L_w(t) \leq L_v(0) - L_w(0) + (\vartheta - 1)t \leq H + (\vartheta - 1)(d + T)D.
\]

**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 \( \mathcal{G} \in \max\{H, uD\} + O((\vartheta - 1)dD) \). 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 \( O(uD + (\vartheta - 1)d) \).
- 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 $G \geq uD$, 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 $H_{E^1_v}(t) = t$ for all $x \in V$. The strong envelope condition thus entails that $L_{E^1_v}(t) = H_{E^1_v}(t) = t$ for all $x$ and $t$. As $E_v$ is indistinguishable from $E_1$, it follows that also $L_{E_v}(t) = H_{E_v}(t)$ for all $x$ and $t$. In particular, there is some $w \in V$ such that

$$L_{E_v}(t_0) - L_{E_w}(t_0) = uD - \varepsilon .$$

As this holds for arbitrarily small $\varepsilon > 0$, we conclude that indeed $G \geq uD$, 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 \((1 - \frac{1}{n}) u\) 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 \(O(u)\) 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


Lecture 2

Gradient Clock Synchronization

In the previous lesson, we proved essentially matching upper and lower bounds on the worst-case global skew for the clock synchronization problem. We saw that during an execution of the Max algorithm (Algorithm 1.2), all logical clocks in all executions eventually agree up to an additive term of $O(uD)$ (ignoring other parameters). The lower bound we proved in Section 1.3 shows that global skew of $\Omega(uD)$ is unavoidable for any algorithm in which clocks run at an amortized constant rate, at least in the worst case. In our lower bound construction, the two nodes $v$ and $w$ that achieved the maximal skew were distance $D$ apart. However, the lower bound did not preclude neighboring nodes from remaining closely synchronized throughout an execution. In fact, this is straightforward if one is willing to slow down clocks arbitrarily (or simply stop them), even if the amortized rate is constant.

Today, we look into what happens if one requires that clocks progress at a constant rate at all times. In many applications, it is sufficient that neighboring clocks are closely synchronized, while nodes that are further apart are only weakly synchronized. To model this situation, we introduce the gradient clock synchronization (GCS) problem. Intuitively, this means that we want to ensure a small skew between neighbors despite maintaining “proper” clocks. That is, we minimize the local skew under the requirement that logical clocks always run at least at rate 1.

2.1 Formalizing the Problem

Let $G = (V, E)$ be a network. As in the previous lecture, each node $v \in V$ has a hardware clock $H_v : \mathbb{R}^+_0 \rightarrow \mathbb{R}^+_0$ that satisfies for all $t, t' \in \mathbb{R}^+_0$ with $t' < t$

$$t - t' \leq H_v(t) - H_v(t') \leq \vartheta(t - t').$$

Again, we denote by $h_v(t)$ the rate of $H_v(t)$ at time $t$, i.e., $1 \leq h(t) \leq \vartheta$ for all $t \in \mathbb{R}^+_0$. Recall that each node $v$ computes a logical clock $L_v : \mathbb{R}^+_0 \rightarrow \mathbb{R}^+_0$ from its hardware clock and messages received from neighbors. During an execution $E$, for each edge $e = \{v, w\} \in E$, we define the local skew of $e$ at time $t$ to be
\( L_e(t) = |L_v(t) - L_w(t)| \). The gradient skew at time \( t \) in the network, denoted \( \mathcal{L}(t) \), is the largest local skew across any edge: \( \mathcal{L}(t) = \max_{e \in E} L_e(t) \). Finally, the gradient skew over an execution \( \mathcal{E} \) is defined to be

\[
\mathcal{L} = \sup_{t \in \mathbb{R}_+^0} \{ \mathcal{L}(t) \}.
\]

The goal of the gradient clock synchronization problem is to minimize \( \mathcal{L} \) for any possible execution \( \mathcal{E} \).

Attention: In order to simplify our presentation of the gradient clock synchronization problem, we abstract away from the individual messages and message delays from the previous chapter. Instead, we assume that throughout an execution, each node \( v \) maintains an estimate of its neighbors’ logical clocks. Specifically, for each neighbor \( w \in N_v \), \( v \) maintains a variable \( \tilde{L}_v^w(t) \). The parameter \( \delta \) represents the error in the estimates: for all \( \{v, w\} \in E \) and \( t \in \mathbb{R}_+^0 \), we have

\[
L_w(t) \geq \tilde{L}_v^w(t) > L_w(t) - \delta.
\]  \( (2.1) \)

When the node \( v \) is clear from context, we will omit the superscript \( v \), and simply write \( \tilde{L}_w \).

In order to obtain the estimates \( \tilde{L}_w^v(t) \), each node \( w \) periodically broadcasts its logical clock value to its neighbors. Each neighbor \( v \) then computes \( \tilde{L}_w^v(t) \) using the known bounds on message delays, and increases \( \tilde{L}_w^v \) at rate \( h_v/\vartheta \) between messages from \( w \). Thus, an upper bound on the error parameter \( \delta \) can be computed as a function of \( u \) (the uncertainty in message delay), \( \vartheta \) (the maximum clock drift), \( T \) (the frequency of broadcasts), and \( \mu \) (a parameter determining how fast logical clocks may run, see below); you do this in the exercises.

To focus on the key ideas, we make another simplifying abstraction: Instead of analyzing the global skew, we assume that it is taken care of and plug in \( \mathcal{G} \) as a parametrized upper bound. You will address this issue as an exercise, too.

### 2.2 Averaging Protocols

In this section, we consider a natural strategy for achieving gradient clock synchronization: trying to bring the own logical clock to the average value between the neighbors whose clocks are furthest ahead and behind, respectively. Specifically, each node can be in either fast mode or slow mode. If a node \( v \) detects that its clock is behind the average of its neighbors, it will run in fast mode, and increase its logical clock at a rate faster than its hardware clock by a factor of \( 1 + \mu \), where \( \mu \) is some appropriately chosen constant. On the other hand, if \( v \)’s clock is at least the average of its neighbors, it will run in slow mode, increasing its logical clock only as quickly as its hardware clock. Note that this strategy results in logical clocks that behave like “real” clocks of drift \( \vartheta' = \vartheta(1 + \mu) - 1 \).

If \( \mu \in O(\vartheta) \), these clocks are roughly as good as the original hardware clocks.

The idea of switching between fast and slow modes gives a well-defined protocol if neighboring clock values are known precisely,\(^1\) however ambiguity

---

\(^1\)There is one issue of pathological behavior in which nodes could switch infinitely quickly between fast and slow modes. This can be avoided by introducing a small threshold \( \delta \) so that a node only changes, say, from slow to fast mode if it detects that its clock is \( \delta \) time units behind the average.
arises in the presence of uncertainty.

We consider two natural ways of dealing with the uncertainty. Set \( L_{N_v}^{\max}(t) := \max_{w \in N_v} \{ \hat{L}_w \} \) and \( L_{N_v}^{\min}(t) := \min_{w \in N_v} \{ \hat{L}_w \} \).

**Aggressive strategy:** each \( v \) computes an upper bound on the average between \( L_{N_v}^{\max} \) and \( L_{N_v}^{\min} \), and determines whether to run in fast or slow mode based on this upper bound.

**Conservative strategy:** each \( v \) computes a lower bound on the average between \( L_{N_v}^{\max} \) and \( L_{N_v}^{\min} \) and determines the mode accordingly.

We will see that, in fact, both strategies yield terrible results, but for opposite reasons. In Section 2.3, we will derive an algorithm that strikes an appropriate balance between both strategies, with impressive results!

**Aggressive Averaging**

Here we analyze the aggressive averaging protocol described above. Specifically, each node \( v \in V \) computes an upper bound on the average of its neighbors’ logical clock values:

\[
\hat{L}_v^{\text{up}}(t) = \frac{\max_{w \in N_v} \{ \hat{L}_w \} + \min_{w \in N_v} \{ \hat{L}_w \}}{2} + \delta \geq \frac{L_{N_v}^{\max} + L_{N_v}^{\min}}{2}.
\]

The algorithm then increases the logical clock of \( v \) at a rate of \( h_v(t) \) if \( L_v(t) > \hat{L}_v^{\text{up}}(t) \), and a rate of \((1 + \mu)h_v(t)\) otherwise. We show that the algorithm performs poorly for any choice of \( \mu \geq 0 \).

**Claim 2.1.** Consider the aggressive averaging protocol on a path network of diameter \( D \), i.e., \( V = \{v_i \mid i \in [D + 1]\} \) and \( E = \{v_i, v_{i+1}\} \mid i \in [D] \). Then there exists an execution \( E \) such that the gradient skew satisfies \( \mathcal{L} \in \Omega(\delta D) \).

**Proof Sketch.** Throughout the execution, we will assume that all clock estimates are correct: for all \( v \in V \) and \( w \in N_v \), we have \( \hat{L}_w^{\text{up}}(t) = L_w(t) \). This means for all \( i \in [D] \setminus \{0\} \) that \( \hat{L}_v^{\text{up}}(t) = (L_{v_{i-1}}(t) + L_{v_{i+1}}(t))/2 + \delta \), whereas \( \hat{L}_v^{\text{up}}(t) = L_{v_i}(t) + \delta \) and \( \hat{L}_v^{\text{up}} = L_{v_{D-1}}(t) + \delta \). Initially, the hardware clock rate of node \( v_i \) is \( 1 + \frac{i(\delta - 1)}{\delta D} \). Thus, even though all nodes immediately “see” that skew is building up, they all set their clock rates to fast mode in order to catch up in case they underestimate their neighbors’ clock values.

Now let’s see what happens to the logical clocks in this execution. While nodes are running fast, skew keeps building up, but the property that \( L_{v_i}(t) = (L_{v_{i+1}}(t) - L_{v_{i-1}}(t)) \) is maintained at nodes \( i \in [D] \setminus \{0\} \). In this state, \( v_0 \) — despite running fast — has no way of catching up to \( v_1 \). However, at time \( t_0 := \frac{\delta D}{(1 + \mu)(\delta - 1)} \), we would have that \( L_{v_D}(\tau_0) = L_{v_{D-1}}(\tau_0) + \delta = L_{v_D}(\tau_0) \) and \( v_D \) would stop running fast. We set \( t_0 := \tau_0 - \varepsilon \) for some arbitrarily small \( \varepsilon > 0 \) and set \( h_{v_D}(t) := h_{v_{D-1}}(t) \) for all \( t \geq t_0 \). Thus, all nodes would remain in fast mode until the time \( t_1 := t_0 + \frac{\delta D}{(1 + \mu)(\delta - 1)} \) when we had \( L_{v_{D-1}}(\tau_1) = L_{v_{D-1}}^{\text{up}}(\tau_1) \).

We set \( t_1 := \tau_1 - \varepsilon \) and proceed with this construction inductively. Note that, with every hop, the local skew increases by \((\text{almost}) \) \( 2\delta \), as this is the additional skew that \( L_v \) must build up to \( L_{v_{D-1}} \) when \( L_{v_{D+1}} = L_{v_i} \) in order to increase \( \hat{L}_v^{\text{up}} - L_{v_i} \) by \( \delta \), i.e., for \( v_i \) to stop running fast. As \( \varepsilon \) is arbitrarily small, we build up a local skew that is arbitrarily close to \((2D - 1)\delta \). \( \square \)
Remarks:

- The algorithm is also bad in that the above execution results in a global skew of $\Omega(\delta D^2)$.
- This could be fixed fairly easily, but without further changes still a large local skew could build up.
- The above argument can be generalized to arbitrary graphs, by taking two nodes $v, w \in V$ in distance $D$ and using the function $d(x) = d(x, v) - d(x, w)$, just as in Lemma 1.5.

Conservative Averaging

Let’s be more careful. Now each node $v \in V$ computes a lower bound on the average of its neighbors’ logical clock values:

$$\tilde{L}_{\text{up}}^v(t) = \frac{\max_{w \in N_v} \{ \tilde{L}_w \} + \min_{w \in N_v} \{ \tilde{L}_w \}}{2} \leq \frac{L_{\text{max}}^N + L_{\text{min}}^N}{2}.$$

The algorithm then increases the logical clock of $v$ at a rate of $h_v(t)$ if $L_v(t) > \tilde{L}_{\text{up}}^v(t)$, and a rate of $(1 + \mu) h_v(t)$ otherwise. Again, the algorithm fails to achieve a small local skew.

Claim 2.2. Consider the conservative averaging protocol on a path network of diameter $D$. Then there exists an execution $E$ such that the gradient skew satisfies $L \in \Omega(\delta D)$.

Proof Sketch. We do the same as for the aggressive strategy, except that now for each $v \in V$, $w \in N_w$, and time $t$, we rule that $\tilde{L}_w(t) = L_w(t) - \delta + \varepsilon$ for some arbitrarily small $\varepsilon > 0$. Thus, all nodes are initially in slow mode. We inductively change hardware clock speeds just before nodes would switch to fast mode, building up the exact same skew between logical clocks as in the previous execution. The only difference is that now it does not depend on $\mu$ how long this takes!

Remarks:

- It seems as if we just can’t do things right. Both the aggressive and the conservative strategy do not result in a proper response to the gobal distribution of clock values.
- Surprisingly, mixing the two strategies works! We study this during the remainder of the lecture.

2.3 GCS Algorithm

The high-level strategy of the algorithm is as follows. As above, at each time each node can be either in slow mode or fast mode. In slow mode, a node $v$ will increase its logical clock at rate $h_v(t)$. In fast mode, $v$ will increase its logical clock at rate $(1 + \mu) h_v(t)$. The parameter $\mu$ will be chosen large enough for nodes whose logical clocks are behind to be able to catch up to
other nodes. The conditions for a node to switch from slow to fast or vice versa are simple, but perhaps unintuitive. In what follows, we first describe “ideal” conditions to switch between modes. In the ideal behavior, each node knows exactly the logical clock values of its neighbors. Since the actual algorithm only has access to estimates of neighboring clocks, we then describe fast and slow triggers for switching between modes that can be implemented in our model for GCS. We conclude the section by proving that the triggers do indeed implement the conditions.

**Fast and Slow Conditions**

**Definition 2.3 (FC: Fast Mode Condition).** We say that a node \( v \in V \) satisfies the fast mode condition (FC) at time \( t \in \mathbb{R}^+ \) if there exists \( s \in \mathbb{N} \) such that:

- **FC1**: \( \exists x \in N_v : L_x(t) - L_v(t) \geq 2s\delta \);
- **FC2**: \( \forall y \in N_v : L_v(t) - L_y(t) \leq 2s\delta \).

Informally, FC1 says that \( v \) has a neighbor \( x \) whose logical clock is significantly ahead of \( L_v(t) \), while FC2 stipulates that none of \( v \)'s neighbors' clocks is too far behind \( L_v(t) \). In particular, if FC is satisfied with \( x \in N_v \) satisfying FC1, then the local skew across \( \{v, x\} \) is at least \( 2s\delta \), where \( L_x \) is at least \( 2s\delta \) time units ahead of \( L_v \). Since none of \( v \)'s neighbors are running more than \( 2s\delta \) units behind \( L_v \), \( v \) can decrease the maximum skew with its neighbors by increasing its logical clock.

The slow mode condition below is dual to FC. It essentially gives conditions under which \( v \) could decrease the maximum skew in its neighborhood by decreasing its logical clock.

**Definition 2.4 (SC: Slow Mode Condition).** We say that a node \( v \in V \) satisfies the slow mode condition (or SC) at time \( t \in \mathbb{R}^+ \) if there exists \( s \in \mathbb{N} \) such that:

- **SC1**: \( \exists x \in N_v : L_v(t) - L_x(t) \geq (2s - 1)\delta \);
- **SC2**: \( \forall y \in N_v : L_y(t) - L_v(t) \leq (2s - 1)\delta \).

Subtracting an additional \( \delta \) in SC1 and SC2 ensures that conditions FC and SC are mutually exclusive. Together, the conditions mean that, if in doubt, the algorithm alternates between aggressively seeking to reduce skew towards neighbors that are ahead (FC) and conservatively avoiding to build up additional skew to neighbors that are behind (SC), depending on the currently observed average skew.

**Fast and Slow Triggers**

While the fast and slow mode conditions described in the previous section are well-defined (and mutually exclusive), uncertainty on neighbors' clock values prevents an algorithm from checking the conditions directly. Here we define corresponding triggers that our computational model does allow us to check.

The separation of \( \delta \) between the conditions is just enough for this purpose. As we assumed that clock values are never overestimated, but may be underestimated by \( \delta \), the fast mode trigger needs to shift its thresholds by \( \delta \).
Definition 2.5 (FT: Fast Mode Trigger). We say that \( v \in V \) satisfies the fast mode trigger (FT) at time \( t \in \mathbb{R}_0^+ \) if there exists an integer \( s \in \mathbb{N} \) such that:

\[
\text{FT 1: } \exists x \in N_v: \tilde{L}_x(t) - L_v(t) > (2s - 1)\delta;
\]

\[
\text{FT 2: } \forall y \in N_v: L_v(t) - \tilde{L}_y(t) < (2s + 1)\delta.
\]

Definition 2.6 (ST: Slow Mode Trigger). We say that a node \( v \in V \) satisfies the slow mode trigger (or ST) at time \( t \in \mathbb{R}_0^+ \) if there exists \( s \in \mathbb{N} \) such that:

\[
\text{ST 1: } \exists x \in N_v: L_v(t) - \tilde{L}_x(t) \geq (2s - 1)\delta;
\]

\[
\text{ST 2: } \forall y \in N_v: \tilde{L}_y(t) - L_v(t) \leq (2s - 1)\delta.
\]

Before we formally describe the GCS algorithm, we give two preliminary results about the fast and slow mode triggers. The first result claims that FT and ST cannot simultaneously be satisfied by the same node. The second shows that FT and ST implement FC and SC, respectively. That is, if the fast (resp. slow) mode condition is satisfied, then the fast (resp. slow) mode trigger is also satisfied.

Lemma 2.7. No node \( v \in V \) can simultaneously satisfy FT and ST.

Proof. Suppose \( v \) satisfies FT, i.e., there is \( s \in \mathbb{N} \) so that there is some \( x \in N_v \) such that \( \tilde{L}_x(t) - L_v(t) > (2s - 1)\delta \) and for all \( y \in N_v \) we have \( L_v(t) - \tilde{L}_y(t) < (2s + 1)\delta \). Consider \( s' > s \), then for all \( y \in N_v \) we have that

\[
L_v(t) - \tilde{L}_x(t) < (2s + 1)\delta \leq (2s' - 1)\delta,
\]

so ST 1 is not satisfied for \( s' \). If \( s' \leq s \), then there is some \( x \in N_v \) so that

\[
\tilde{L}_x(t) - L_v(t) > (2s - 1)\delta \geq (2s' - 1)\delta,
\]

so ST 2 is not satisfied for \( s' \). Hence, ST is not satisfied.

Lemma 2.8. Suppose \( v \in V \) satisfies FC (resp. SC) at time \( t \). Then \( v \) satisfies FT (resp. SC) at time \( t \).

Proof. Suppose FC holds (at time \( t \)). Then, by (2.1), there is some \( s \in \mathbb{N} \) such that

\[
\exists x \in N_v: \tilde{L}_x(t) - L_v(t) > L_x(t) - \delta - L_v(t) \geq (2s - 1)\delta
\]

and

\[
\forall y \in N_v: L_v(t) - \tilde{L}_y(t) < L_y(t) - L_v(t) + \delta \leq (2s + 1)\delta,
\]

i.e., FT holds. Similarly, if SC holds, (2.1) yields that

\[
\exists x \in N_v: L_v(t) - \tilde{L}_x(t) \geq L_x(t) - L_v(t) \geq (2s - 1)\delta
\]

and

\[
\forall y \in N_v: \tilde{L}_y(t) - L_x(t) \leq L_y(t) - L_v(t) \leq (2s - 1)\delta
\]

for some \( s \in \mathbb{N} \), establishing ST.

We now describe the GCS algorithm. Each node \( v \) initializes its logical clock to its hardware clock value. It continuously checks if the fast (resp. slow) mode trigger is satisfied. If so, it increases its logical clock at a rate of \( (1 + \mu)h_v(t) \) (resp. \( h_v(t) \)). Pseudocode is presented in Algorithm 2.1. The algorithm itself is simple, but the analysis of the algorithm (presented in the following section) is rather delicate.
2.4. ANALYSIS OF THE GCS ALGORITHM

Algorithm 2.1: GCS algorithm

1. $L_v(0) := H_v(0)$
2. $r := 1$
3. at all times $t$ do the following
4. if $FT$ then
5. \[ r := 1 + \mu \]
   // $v$ is in fast mode
6. if $ST$ then
7. \[ r := 1 \]
   // $v$ is in slow mode
8. increase $L_v$ at rate $r h_v(t)$

Remarks:

- In fact, when neither $FT$ nor $ST$ hold, the logical clock may run at any speed from the range $[h_v(t), (1 + \mu) h_v(t)]$.
- In order for the algorithm to be implementable, $\delta$ should leave some wiggle space. We expressed this by having (2.1) include a strict inequality, but if the inequality can become arbitrarily tight, the algorithm may have to switch between slow and fast mode arbitrarily fast.
- For technical reasons, we will assume that logical clocks are differentiable. Thus, $l_v := \frac{d}{dt} L_v$ exists and is between 1 and $\vartheta (1 + \mu)$ at all times. It is possible to prove the guarantees of the algorithm without this assumption, but all this does is making the math harder.
- Even with this assumption, we still need Lemma A.1. This is not a mathematics lecture, but as we couldn’t find any suitable reference, the lemma and a proof is given in the appendix.

2.4 Analysis of the GCS Algorithm

We now show that the GCS algorithm (Algorithm 2.1) indeed achieves a small local skew, which is expressed by the following theorem.

Theorem 2.9. For every network $G$ and every execution $E$ in which $H_v(0) - H_w(0) \leq \delta$ for all edges $\{v, w\} \in E$, the GCS algorithm achieves a gradient skew of $L \leq 2 \delta \lceil \log_{\sigma} G/\delta \rceil$, where $\sigma := \mu / (\vartheta - 1)$.

In order to prove Theorem 2.9, we analyze the average skew over paths in $G$ of various lengths. For long paths of $\Omega(D)$ hops, we will simply exploit that $G$ bounds the skew between any pair of nodes. For successively shorter paths, we inductively show that the average skew between endpoints cannot increase too quickly: reducing the length of a path by factor $\sigma$ can only increase the skew between endpoints by an additive constant term. Thus, paths of constant length (in particular edges) can only have a skew that is logarithmic in the network diameter.

Leading Nodes

We start by showing that skew cannot build up too quickly. This is captured by the following functions.
we define

\[ \Psi^a(t) = \max_{w \in V}\{L_w(t) - L_v(t) - (2s-1)\delta d(v, w)\}, \]

where \(d(v, w)\) denotes the distance between \(v\) and \(w\) in \(G\). Moreover, set

\[ \Psi^b(t) = \max_{w \in V}\{\Psi^a_w(t)\}. \]

Finally, we say that \(w \in V\) is a leading node if there is some \(v \in V\) so that

\[ \Psi^* = \Psi^b(t) = L_w(t) - L_v(t) - (2s-1)\delta d(v, w) > 0. \]

We will show that \(\Psi^*(t) \leq G/\sigma^*\) for each \(s \in \mathbb{N}\) and all times \(t\). For \(s = \lceil \log_\sigma G/\delta \rceil\), this yields that

\[ L_v(t) - L_w(t) - (2s-1)\delta \leq G/\sigma^* \leq \delta \quad \Rightarrow \quad L_v(t) - L_w(t) \leq 2\delta \lceil \log_\sigma G/\delta \rceil. \]

The definition of \(\Psi^*_w\) is closely related to the slow mode condition \(\text{SC}\). It makes sure that leading nodes are always in slow mode.

**Lemma 2.11 (Leading Lemma).** Suppose \(w \in V\) is a leading node at time \(t\). Then \(w\) satisfies \(\text{SC}\) and \(\text{ST}\).

**Proof.** As \(w\) is a leading node at time \(t\), there are \(s \in \mathbb{N}\) and \(v \in V\) so that

\[ \Psi^*_w(t) = L_w(t) - L_v(t) - (2s-1)\delta d(v, w) > 0. \]

In particular, \(L_w(t) > L_v(t)\), so \(w \neq v\). For any \(y \in V\), we have that

\[ L_w(t) - L_v(t) - (2s-1)\delta d(v, w) = \Psi^*_w(t) \geq L_y(t) - L_v(t) - (2s-1)\delta d(y, w). \]

Rearranging this yields

\[ L_w(t) - L_y(t) \geq (2s-1)\delta (d(v, w) - d(y, w)). \]

In particular, for any \(y \in N_v\), \(d(v, w) \geq d(y, w) - 1\) and hence

\[ L_y(t) - L_w(t) \leq (2s-1)\delta, \]

i.e., \(\text{SC}2\) holds at \(w\). Now consider \(x \in N_v\) so that \(d(x, w) = d(v, w) - 1\); as \(v \neq w\), such a node exists. We get that

\[ L_w(t) - L_y(t) \geq (2s-1)\delta, \]

showing \(\text{SC}1\). By Lemma 2.8, \(w\) then also satisfies \(\text{ST}\) at time \(t\).

This can readily be translated into a bound on the growth of \(\Psi^*_w\) whenever it is positive.

**Lemma 2.12 (Wait-up Lemma).** Suppose \(w \in V\) satisfies \(\Psi^*_w(t) > 0\) for all \(t \in (t_0, t_1]\). Then

\[ \Psi^*_w(t_1) \leq \Psi^*_w(t_0) - (L_w(t_1) - L_w(t_0)) + \vartheta(t_1 - t_0). \]
2.4. ANALYSIS OF THE GCS ALGORITHM

Proof. Fix \( w \in V \), \( s \in \mathbb{N} \) and \((t_0,t_1] \) as in the hypothesis of the lemma. For \( v \in V \) and \( t \in (t_0,t_1] \), define the function \( f_v(t) = L_v(t) - (2s - 1)\delta d(v,w) \). Observe that

\[
\max_{v \in V} \{ f_v(t) \} - L_w(t) = \Psi_w^s(t) .
\]

Moreover, for any \( v \) satisfying \( f_v(t) = L_w(t) + \Psi_w^s(t) \), we have that \( L_v(t) - L_w(t) - (2s - 1)\delta d(v,w) = \Psi_w^s(t) > 0 \). Thus, Lemma 2.11 shows that \( v \) is in slow mode at time \( t \). As (we assume that) logical clocks are differentiable, so is \( f_v \), and it follows that \( \frac{d}{dt} f_v(t) \leq \vartheta \) for any \( v \in V \) and time \( t \in (t_0,t_1] \) satisfying \( f_v(t) = \max_{x \in V} \{ f_x(t) \} \). By Lemma A.1, it follows that \( \max_{v \in V} \{ f_v(t) \} \) grows at most at rate \( \vartheta \):

\[
\max_{v \in V} \{ f_v(t_1) \} \leq \max_{v \in V} \{ f_v(t_0) \} + \vartheta (t_1 - t_0) .
\]

We conclude that

\[
\Psi_w^s(t_1) - \Psi_w^s(t_0) = \max_{v \in V} \{ f_v(t_1) \} - L_w(t_1) - (\max_{v \in V} \{ f_v(t_0) \} - L_w(t_0)) \\
\leq -(L_w(t_1) - L_w(t_0)) + \vartheta (t_1 - t_0) ,
\]

which can be rearranged into the claim of the lemma. \( \Box \)

Trailing Nodes

As \( L_w(t_1) - L_w(t_0) \geq t_1 - t_0 \) at all times, Lemma 2.15 shows that \( \Psi^s \) cannot grow faster than at rate \( \vartheta - 1 \) when it is positive. This buys us some time, but we need to show that \( w \) will make sufficient progress before \( \Psi^s \) grows larger than the desired bound. The approach to showing this is very similar to the one for Lemma 2.12, where now we need to exploit the fast mode condition \( \text{FC} \).

Definition 2.13 (Trailing Nodes). We say that \( w \in V \) is a trailing node at time \( t \), if there is some \( s \in \mathbb{N} \) and a node \( v \) such that

\[
L_v(t) - L_w(t) - 2s\delta d(v,w) = \max_{x \in V} \{ L_v(t) - L_x(t) - 2s\delta d(v,x) \} > 0 .
\]

Lemma 2.14 (Trailing Lemma). Suppose \( w \in V \) is a trailing node at time \( t \). Then \( w \) satisfies \( \text{FC} \) and \( \text{FT} \).

Proof. Let \( s \) and \( v \) be such that

\[
L_v(t) - L_w(t) - 2s\delta d(v,w) = \max_{x \in V} \{ L_v(t) - L_x(t) - 2s\delta d(v,x) \} > 0 .
\]

In particular, \( L_v(t) > L_w(t) \), implying that \( v \neq w \). For \( y \in V \), we have that

\[
L_v(t) - L_w(t) - 2s\delta d(v,w) \geq L_v(t) - L_y(t) - 2s\delta d(v,y)
\]

and thus for all neighbors \( y \in N_w \) that

\[
L_y(t) - L_w(t) + 2s\delta (d(v,y) - d(v,w)) \geq 0 .
\]

It follows that

\[
\forall y \in N_v : L_w(t) - L_y(t) \leq 2s\delta ,
\]

i.e., \( \text{FC} \) \( 2 \) holds. As \( v \neq w \), there is some node \( x \in N_v \) with \( d(v,x) = d(v,w) - 1 \). We obtain that

\[
\exists x \in N_v : L_y(t) - L_w(t) \geq 2s\delta ,
\]

showing \( \text{FC} \) \( 1 \). By Lemma 2.8, \( w \) thus also satisfies \( \text{FT} \) at time \( t \). \( \Box \)
Using this, we can show that if \( \Psi^s_w(t_0) > 0 \), \( w \) will eventually catch up. How long this takes can be expressed in terms of \( \Psi^{s-1}(t_0) \), or, if \( s = 1 \), \( \mathcal{G} \).

**Lemma 2.15** (Catch-up Lemma). Let \( s \in \mathbb{N} \) and \( t_0, t_1 \) be times. If \( s = 1 \), suppose that \( t_1 \geq t_0 + \mathcal{G}/\mu \); otherwise, suppose that \( t_1 \geq t_0 + \Psi^{s-1}(t_0)/\mu \). Then, for any \( w \in V \),

\[
L_w(t_1) - L_w(t_0) \geq t_1 - t_0 + \Psi^s_w(t_0).
\]

**Proof.** Choose \( v \in V \) such that

\[
\Psi^s_w(t_0) = L_v(t_0) - L_w(t_0) - (2s - 1)\delta d(v, w) > 0.
\]

Define \( f_x(t) := L_v(t_0) + (t - t_0) - L_x(t) - (2s - 2)\delta d(v, x) \) for \( x \in V \) and observe that \( \Psi^s_w(t_0) \leq f_w(t_0) \). Hence, if \( \max_{x \in V} \{ f_x(t) \} \leq 0 \) for some \( t \in [t_0, t_1] \), then

\[
L_w(t_1) - L_w(t) - (t_1 - t) \geq 0 \geq f_w(t)
\]

\[
= L_w(t_0) + (t - t_0) - L_w(t) - (2s - 2)\delta d(v, x) = f_w(t_0) + (t - t_0) - (L_w(t) - L_w(t_0))
\]

\[
\geq \Psi^s_w(t_0) + (t - t_0) - (L_w(t) - L_w(t_0)),
\]

which can be rearranged into the claim of the lemma.

To show this, consider any time \( t \in [t_0, t_1] \) when \( \max_{x \in V} \{ f_x(t) \} > 0 \) and let \( y \in V \) be any node such that \( \max_{x \in V} \{ f_x(t) \} = f_y(t) \). Then \( y \) is trailing, as

\[
\max_{x \in V} \{ L_v(t) - L_x(t) - (2s - 2)\delta d(v, x) \} = L_v(t) - L_v(t_0) - (t - t_0) + \max_{x \in V} \{ f_x(t) \}
\]

\[
= L_v(t) - L_v(t_0) - (t - t_0) + f_y(t)
\]

\[
= L_v(t) - L_y(t) - (t - t_0) + (2s - 2)\delta d(v, y)
\]

and

\[
L_v(t) - L_v(t_0) - (t - t_0) + \max_{x \in V} \{ f_x(t) \} > L_v(t) - L_v(t_0) - (t - t_0) \geq 0.
\]

Thus, by Lemma 2.14 \( y \) is in fast mode. As logical clocks are (assumed to be) differentiable, we get that \( \frac{d}{dt} f_y(t) = 1 - l_y(t) \leq -\mu \).

Now assume for contradiction that \( \max_{x \in V} \{ f_x(t) \} > 0 \) for all \( t \in [t_0, t_1] \). Then, applying Lemma A.1 again, we conclude that

\[
\max_{x \in V} \{ f_x(t_0) \} > -(\max_{x \in V} \{ f_x(t_1) \} - \max_{x \in V} \{ f_x(t_0) \}) \geq \mu(t_1 - t_0).
\]

If \( s = 1 \), \( \mu(t_1 - t_0) \geq \mathcal{G} \), contradicting the fact that

\[
f_x(t_0) = L_v(t_0) - L_x(t_0) \leq \mathcal{G}
\]

for all \( x \in V \). If \( s > 1 \), then \( \mu(t_1 - t_0) \geq \Psi^{s-1}(t_0) \). However, we have that

\[
f_x(t_0) \leq L_v(t_0) - L_x(t_0) - (2s - 3)\delta d(v, x) \leq \Psi^{s-1}(t_0)
\]

for all \( x \in V \). As this is a contradiction as well, the claim of the lemma follows.
Putting Things Together

**Theorem 2.16.** Assume that $H_v(0) - H_w(0) \leq \delta$ for all $\{v, w\} \in E$. Then, for all $s \in \mathbb{N}$, Algorithm 2.1 guarantees $\Psi^*(t) \leq \mathcal{G}/\sigma^s$, where $\sigma = \mu/(1 - \vartheta)$.

**Proof.** Suppose for contradiction that the statement of the theorem is false. Let $s \in \mathbb{N}$ be minimal such that there is a time $t_1$ for which $\Psi^*(t_1) = \mathcal{G}/\sigma^s + \varepsilon$ for some $\varepsilon > 0$. Thus, there is some $w \in V$ such that

$$\Psi^w_s(t_1) = \Psi^*(t_1) = \frac{\mathcal{G}}{\sigma^s} + \varepsilon.$$

Set $t_0 := \max\{t - \mathcal{G}/(\mu\sigma^{s-1}), 0\}$. Consider the time $t' \in [t_0, t_1]$ that is minimal with the property that $\Psi^w_s(t) > 0$ for all $t \in (t', t_1]$ (by continuity of $\Psi^w_s$ such a time exists). Thus, we can apply Lemma 2.12 to this interval, yielding that

$$\Psi^w_s(t_1) \leq \Psi^w_s(t') + \vartheta(t_1 - t') - (L_w(t_1) - L_w(t')) \leq \Psi^w_s(t') + (\vartheta - 1)(t_1 - t').$$

If $\Psi^w_s(t')$ cannot be 0, as otherwise

$$\Psi^w_s(t_1) \leq (\vartheta - 1)(t_1 - t') \leq \frac{\vartheta - 1}{\mu} \cdot \mathcal{G}/\sigma^{s-1} = \frac{\mathcal{G}}{\sigma^s},$$

contradicting $\Psi^w_s(t_1) = \mathcal{G}/\sigma^s + \varepsilon$.

On the other hand, if $\Psi^w_s(t') > 0$, we must have $t' = t_0$ from the definition of $t'$, and $t_0 \neq 0$ because

$$\max_{v, w \in V} \{L_v(0) - L_w(0) - (2s - 1)\delta d(v, w)\} = \max_{v, w \in V} \{H_v(0) - H_w(0) - (2s - 1)\delta d(v, w)\} \leq \max_{v, w \in V} \{H_v(0) - H_w(0) - \delta d(v, w)\} \leq 0,$$

as $H_v(0) - H_w(0) \leq \delta$ for all neighbors $v, w$ by assumption. Hence, $t' = t_0 = t_1 - \mathcal{G}/(\mu\sigma^{s-1})$. If $s > 1$, the minimality of $s$ yields that $\Psi^s(t_0) \leq \mathcal{G}/\sigma^{s-1}$. We apply Lemma 2.15 to level $s$, node $w$, and time $t' = t_0$, yielding that

$$\Psi^w_s(t_1) \leq \Psi^w_s(t_0) + \vartheta(t_1 - t_0) - (L_w(t_1) - L_w(t_0)) \leq (\vartheta - 1)(t_1 - t_0) \leq \frac{\mathcal{G}}{\sigma^s},$$

again contradicting $\Psi^w_s(t_1) = \mathcal{G}/\sigma^s + \varepsilon$. Reaching a contradiction in all cases, we conclude that the statement of the theorem must indeed hold.

Our main result, Theorem 2.9, is now immediate.

**Proof of Theorem 2.9.** We apply Theorem 2.16 and consider $s := \lceil \log_\sigma(\mathcal{G}/\delta) \rceil$. For any $\{v, w\} \in E$ and any time $t$, we thus have that

$$L_v(t) - L_w(t) - (2s - 1)\delta = L_v(t) - L_w(t) - (2s - 1)\delta d(v, w) \leq \Psi^s(t) \leq \frac{\mathcal{G}}{\sigma^s} \leq \delta.$$

Rearranging this and exchanging the roles of $v$ and $w$, we obtain

$$L(t) := \max_{\{v, w\} \in E} \{|L_v(t) - L_w(t)|\} \leq 2s\delta = 2\delta\lceil \log_\sigma(\mathcal{G}/\delta) \rceil.$$

□
What to Take Home

- A very simple algorithm achieves a surprisingly good local skew, even if clocks must advance at all times.

- The base of the logarithm in the bound is typically large. A cheap quartz oscillator guarantees $\vartheta - 1 \leq 10^{-5}$, while typically $u/d \geq 10^{-2}$. With a base of roughly $10^3$, the logarithmic term usually remains quite small.

- The algorithmic idea is surprisingly versatile. It works if $\delta$ is different for each link, and with some modifications (to algorithm and analysis), adversarial changes in the graph can be handled.

Bibliographic Notes

Gradient clock synchronization was introduced by Fan and Lynch [FL06], who show a lower bound of $\Omega(\log(uD)/\log\log(uD))$ on the local skew. Some researchers found this result rather counter-intuitive, and it triggered a line of research seeking to resolve the question what precisely can be achieved. The first non-trivial upper bound was provided by Locher and Wattenhofer [LW06]. Their blocking algorithm bounds the local skew by $O(\sqrt{\delta D})$. The first logarithmic bound on the local skew was given in [LLW08] and soon after improved to the algorithm presented here [LLW10]. However, the elegant way of phrasing it in terms of the fast and slow modes and conditions is due to Kuhn and Oshman [KO09].

The algorithmic idea underlying the presented solution turns out to be surprisingly robust and versatile. Essentially the same algorithm works for different uncertainties on the edges [KO09]. With a suitable method of carefully incorporating newly appearing edges, it can handle dynamic graphs [KLO10] (this problem is introduced in [KLO11]), in the sense that edges that were continuously present for sufficiently long satisfy the respective guarantee on the skew between their endpoints. Recently, the approach has been independently discovered (twice!) for solving load balancing tasks that arise in certain packet routing problems [DLNO17, PR17].

Bibliography


Lecture 3

Lower Bound on the Local Skew

In Chapter 1, we proved tight upper and lower bounds of $\Theta(D)$ for the global skew of any clock synchronization algorithm. However, the algorithms achieving optimal global skew had the undesirable feature that the maximal global skew could be attained between any pair of nodes in the network—even adjacent nodes. In Chapter 2, we developed a more refined algorithm that further controlled the gradient skew—the maximum skew between any pair of adjacent nodes. Specifically, the gradient clock synchronization (GCS) algorithm of Chapter 2 achieved a local skew of $O(\delta \log D)$.

In this chapter, we address the question of whether the $O(\delta \log D)$ skew upper bound for GCS can be improved. Since gradient clock synchronization is a local property (in the sense that the definition of gradient skew only references logical clocks of neighboring nodes), one may expect that a distributed algorithm may be able to achieve $O(\delta)$ local skew. However, we will show that this is impossible: any GCS algorithm must incur local skew of $\Omega(u \log D)$ for some executions. Thus, the GSC algorithm of Chapter 2 is asymptotically optimal.

3.1 Lower Bound with Bounded Clock Rates

In this section, we first prove a lower bound assuming that each logical clock increases at a rate of at most $(1 + \mu)h_v > 1$. That is, for all $v \in V$ and $t, t' \in \mathbb{R}_+^*$ with $t < t'$, we assume $L_v(t') - L_v(t) \leq (1 + \mu)(H_v(t') - H_v(t))$. We use the model of Chapter 1. Moreover, all logical clocks have a minimum rate of 1: for all $v \in V$ and $t, t' \in \mathbb{R}_+^*$ with $t < t'$, we have $L_v(t') - L_v(t) \geq t' - t$. Under these assumptions, we will prove the following theorem.

**Theorem 3.1.** Any algorithm for the gradient clock synchronization problem with logical clock rates between 1 and $(1 + \mu)h_v$ incurs a worst-case gradient skew of $L \geq (u/4 - (\vartheta - 1)d) \log_{[\sigma]} D$, where $\sigma := \mu/((\vartheta - 1))$.\(^1\)

\(^1\)Note that this assumption does not allow for algorithms that increase their clocks discontinuously. For example, the argument does not apply to the max algorithm presented in Chapter 1.
To gain some intuition, assume that \((\vartheta - 1)d < u\), so we can neglect this term. In order to prove Theorem 3.1, we first show that the adversary can build up a hardware clock skew of \(\Omega(uk)\) between any pair of nodes in distance \(k\) in \(O(uk/(\vartheta - 1))\) time, in an indistinguishable way. Specifically, for \(v\) and \(w\) in distance \(k\), we get that \(H^{(E_1)}_v(t) - H^{(E_1)}_w(t) \in \Omega(uk)\) for some time \(t\), while \(H^{(E_1)}_v(t) = H^{(E_1)}_w(t)\). By the minimum progress condition, this implies that the logical clock of \(v\) differs by at least \(\Omega(uk)\) between the two executions. This is, in fact, a straightforward generalization of Lemma 1.5. The key difference is that we sacrifice a factor of 2 in the amount of skew we sneak in, so we can choose the pair of nodes between which we build the skew after examining what happens in \(E_1\), i.e., \(E_1\) provides no information regarding where the skew will “appear.”

We can use this inductively as follows. Assuming that we know how to build up a skew of \(\alpha uk\) between nodes in distance \(k\) (initially, \(k \approx D\) and \(\alpha = 0\)), we run a given GCS algorithm for \(O(uk/(\vartheta - 1))\) time with all hardware clock rates being 1 (that’s the case in \(E_1\)), where \(k' \in \Theta(k/\sigma)\) (with constants chosen suitably). As logical clock rates are between 1 and \(1 + \mu\) in \(E_1\), the skew between the original nodes is still \(\alpha uk - \Omega(\mu uk/(\vartheta - 1)) = (\alpha - \Omega(1))uk\). Thus, there must be two nodes in distance \(k'\) with skew at least \((\alpha - \Omega(1))uk'\). If \(v\) is the node with the larger clock value, we now consider \(E_v\), in which the skew is by \(\Omega(uk')\) larger. For the right choice of \(k'\), we end up with a path of length \(k'\) that has skew \((\alpha + \Omega(1))uk'\). We can repeat this up to \(\Theta(\log_\sigma D)\) many times, yielding the desired lower bound.

**Lemma 3.2.** Assume that \((\vartheta - 1)d < u/2\) and set \(t_0 := d(v, w)(u/(2(\vartheta - 1)) - d)\). For any algorithm, there is an execution \(E_v\) such that for any \(v, w \in V\), there is an indistinguishable execution \(E_v\) satisfying that

\[
\begin{align*}
H^{(E_v)}_x(t) &= t \text{ for all } x \in V \text{ and } t, \\
H^{(E_v)}_v(t) &= H^{(E_v)}_w(t) + d(v, w)(u/2 - (\vartheta - 1)d) \text{ for all } t \geq t_0, \text{ and} \\
H^{(E_v)}_w(t) &= t \text{ for all } t.
\end{align*}
\]

**Proof.** The proof is very similar to the one of Lemma 1.5. In both executions and for all \(x \in V\), we set \(H_x(0) := 0\). Execution \(E_v\) 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 \(E_v\), we set the hardware clock rate of node \(x \in V\) to \(1 + (\vartheta - 1)(d(x) + d(v, w))/(2d(v, w))\) at all times \(t \leq t_0\) and to 1 at all times \(t > t_0\). This implies that

\[
H^{(E_v)}_v(t_0) = \vartheta t_0 = H^{(E_v)}_w(t_0) + d(v, w)\left(\frac{u}{2} - (\vartheta - 1)d\right) \quad \text{and} \\
H^{(E_v)}_w(t_0) = t_0.
\]
3.1. LOWER BOUND WITH BOUNDED CLOCK RATES

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 $E_i$ arrives at the same local time at $y$ as it does in $E_1$. 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 are indistinguishable. 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 $t_1$ 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

$$H_x^{(E_1)}(t_0) - H_y^{(E_1)}(t_0) = \frac{d(y) - d(x)}{2d(v,w)} \cdot (\vartheta - 1)t_0 = \frac{d(y) - d(x)}{2} \left( \frac{u}{2} - (\vartheta - 1)d \right).$$

In execution $E_1$, a message sent from $x$ to $y$ at local time $H_x^{(E_1)}(t) = t$ is received at local time $H_y^{(E_1)}(t) = H_x^{(E_1)}(t) + d - u/2$. If a message is sent at time $t$ in $E_i$, we have that

$$H_y^{(E_i)}(t + d) \geq H_y^{(E_i)}(t) + d = H_x^{(E_i)}(t) + d + \frac{d(x) - d(y)}{2} \left( \frac{u}{2} - (\vartheta - 1)d \right) > H_x^{(E_i)}(t) + d - \frac{u}{2},$$

where the last inequality uses that $d(x) - d(y) \geq -2$ and that $u/2 > (\vartheta - 1)d$ by assumption. On the other hand,

$$H_y^{(E_i)}(t + d - u) < H_y^{(E_i)}(t) + \vartheta d - u = H_x^{(E_i)}(t) + \vartheta d - u + \frac{d(x) - d(y)}{2} \left( \frac{u}{2} - (\vartheta - 1)d \right) \leq H_x^{(E_i)}(t) + d - \frac{u}{2},$$

where the final inequality holds with equality if $d(x) - d(y) = 2$ and thus also for $d(x) - d(y) < 2$, as $u/2 > (\vartheta - 1)d$.

Proof of Theorem 3.1. Note that the claim is vacuous if $(\vartheta - 1)d \geq u/4$, so we can assume the opposite in the following. Let $b := \lceil 2\vartheta \rceil$ and $i_{\max} := \lceil \log_b D \rceil$. By induction over $i \in [i_{\max} + 1]$, we show that we can build up a skew of $(i+2)(u/4-(\vartheta-1)d)d(v,w)$ between nodes $v, w \in V$ in distance $d(v,w) = b^{i_{\max} - 1}$ at a time $t_i$ in execution $E^{(i)}$, such that after time $t_i$ all hardware clock rates are 1 and all sent messages have delays of $d - u/2$.

We anchor the induction at $i = 0$ by applying Lemma 3.2, choosing $t_0$ as in the lemma. We pick two nodes $v, w \in V$ in distance $b^{i_{\max}} \leq D$ of each other such that $L_x^{(E_1)}(t_0) \geq L_w^{(E_1)}(t_0)$. Now consider $E_i$ for this choice of $v, w \in V$, which satisfies that $H_v^{(E_i)}(t_0) = H_w^{(E_i)}(t_0) + (u/2 - (\vartheta - 1)d)d(v,w)$ and $H_w^{(E_i)}(t_0) = H_w^{(E_1)}(t_0)$. By indistinguishability of the two executions and the
minimum logical clock rate of 1, we get that

\[ L_v^{(c)}(t_0) - L_w^{(c)}(t_0) = L_v^{(c)}(t_0 + \left( \frac{u}{2} - (\vartheta - 1)d \right) d(v, w) - L_w^{(c)}(t_0) \]

\[ \geq L_v^{(c)}(t_0) + \left( \frac{u}{2} - (\vartheta - 1)d \right) d(v, w) - L_w^{(c)}(t_0) \]

\[ \geq \left( \frac{u}{2} - (\vartheta - 1)d \right) d(v, w). \]

We obtain \( \mathcal{E}(0) \) by changing all hardware clock rates in \( \mathcal{E}_v \) to 1 at time \( t_0 \) and all message delays of messages sent at or after time \( t_0 \) to \( d - u/2 \). As this does not affect the logical clock values at time \( t_0 - \mathcal{E}(0) \) is indistinguishable from \( \mathcal{E}_v \) at \( x \in V \) until local time \( H_x^{(c)}(t_0) \) — this shows the claim for \( i = 0 \).

For the induction step from \( i \) to \( i + 1 \), let \( v, w \in V, \mathcal{E}^{(i)} \), and \( t_i \) be given by the induction hypothesis, i.e.,

\[ L_v^{(c)}(t_i) - L_w^{(c)}(t_i) \geq (i + 2) \left( \frac{u}{4} - (\vartheta - 1)d \right) d(v, w), \]

and from time \( t_i \) on all hardware clock rates are 1 and sent messages have delay \( d - u/2 \). Note that the latter conditions mean that \( \mathcal{E}^{(i)} \) behaves exactly like \( \mathcal{E}_v \) from Lemma 3.2 from time \( t_i \) on, except that some messages sent at times \( t < t_i \) may arrive during \([t_i, t_i + d] \). Hence, if we apply the same modifications to \( \mathcal{E}^{(i)} \) as to \( \mathcal{E}_v \), but starting from time \( t_i + d \) instead of time 0, we can, for any \( v', w' \in V \), construct an execution \( \mathcal{E}_{v'} \) indistinguishable from \( \mathcal{E}^{(i)} \), where

- \( H_x^{(c)}(t) = H_x^{(c)}(t_i) + t - t_i \) for all \( x \in V \) and \( t \geq t_i \),
- \( H_{v'}^{(c)}(t) = H_{v'}^{(c)}(t_i) + d(v', w')(u/2 - (\vartheta - 1)d) \) for all times \( t \geq t_i + d \) and \( (u/(2(\vartheta - 1)) - d)d(v', w'), \) and
- \( H_{w'}^{(c)}(t) = H_{w'}^{(c)}(t_i) + t - t_i \) for all \( t \geq t_i \).

Consider the logical clock values of \( v \) and \( w \) in \( \mathcal{E}^{(i)} \) at time

\[ t_{i+1} := t_i + d + \left( \frac{u}{2(\vartheta - 1)} - d \right) \frac{d(v, w)}{b}. \]

Recall that \( t_v(t) \geq h_v(t) \geq 1 \) and \( l_w(t) \leq (1 + \mu)h_w(t) \) at all times \( t \). As \( h_w^{(c)}(t_i) = 1 \) at times \( t \geq t_i \), we get that

\[ L_v^{(c)}(t_{i+1}) - L_w^{(c)}(t_{i+1}) \geq L_v^{(c)}(t_i) - L_w^{(c)}(t_i) - \mu(t_{i+1} - t_i). \]

(3.1)

Recall that \( d(v, w) \approx b^{max} \) and that \( b = [2\sigma] \). We split up a shortest path from \( v \) to \( w \) in \( b \) subpaths of length \( b^{max}-(i+1) \). By the pigeon hole principle, at least one of these paths must exhibit at least a \( 1/b \) fraction of the skew between \( v \) and \( w \), i.e., there are \( v', w' \in V \) with \( d(v', w') = b^{max}-(i+1) = d(v, w)/b \) so
that

\[ L^{(\xi)}_{w'}(t_{i+1}) - L^{(\xi)}_{w'}(t_{i+1}) \]

\[ \geq \frac{L^{(\xi)}_{w'}(t_i) - L^{(\xi)}_{w'}(t_i) - \mu(t_{i+1} - t_i)}{b} \]

by (3.1) we have:

\[ \geq \frac{L^{(\xi)}_{w'}(t_i) - L^{(\xi)}_{w'}(t_i) - \mu(d + (u/(2(\vartheta - 1)) - d)d(v', w'))}{b} \]

\[ = \frac{L^{(\xi)}_{w'}(t_i) - L^{(\xi)}_{w'}(t_i) - \mu(\vartheta - 4)\cdot d(v', w')}{b} \]

\[ = \frac{L^{(\xi)}_{w'}(t_i) - L^{(\xi)}_{w'}(t_i) - \frac{u}{4} \cdot d(v', w')}{b} \]

\[ = \frac{(i + 2)(u/4 - (\vartheta - 1)d)d(v, w) - \frac{u}{4} \cdot d(v', w')}{b} \]

In other words, as the average skew on a shortest path from \( v \) to \( w \) did not decrease by more than \( u/4 \), there must be some subpath of length \( d(v, w)/b \) with at least the same average skew. Now we sneak in additional skew by advancing the (hardware and thus also logical) clock of \( v' \) using the indistinguishable execution \( \mathcal{E}_{v'} \):

\[ L^{(\xi)}_{v'}(t_{i+1}) - L^{(\xi)}_{v'}(t_{i+1}) \]

\[ = L^{(\xi)}_{v'}(t_{i+1}) + \left( \frac{u}{2} - (\vartheta - 1)d \right)d(v', w') - L^{(\xi)}_{v'}(t_{i+1}) \]

\[ \geq L^{(\xi)}_{v'}(t_{i+1}) + \left( \frac{u}{2} - (\vartheta - 1)d \right)d(v', w') - L^{(\xi)}_{v'}(t_{i+1}) \]

\[ \geq (i + 3) \left( \frac{u}{4} - (\vartheta - 1)d \right)d(v', w') \]

This completes the induction. Plugging in \( i = i_{\max} \) and noting that \( \log b = \log[2\sigma] \leq 1 + \log[\sigma] \), we get an execution in which two nodes at distance \( b^i = 1 \) exhibit a skew of at least

\[ (i_{\max} + 2) \left( \frac{u}{4} - (\vartheta - 1)d \right) \geq \left( \frac{u}{4} - (\vartheta - 1)d \right) (1 + \log_b D) \]

\[ \geq \left( \frac{u}{4} - (\vartheta - 1)d \right) \log[\sigma] D. \]

**Remarks:**

- It is somewhat “bad form” to adapt Lemma 3.2 on the fly, as we did in the proof. However, the alternative of carefully defining partial executions, how to stitch them together, and proving indistinguishability results in this setting would mean to crack a nut with a sledgehammer.
By making the base of the logarithm larger (i.e., making paths shorter more quickly), we can reduce the “loss” of skew in each step. Thus, we get a skew of \((u/2 - (\vartheta - 1)d - \varepsilon)\) per iteration, at the cost of reducing the number of iterations by a factor of \(\log \sigma / (\log \sigma - \log \varepsilon^{-1})\).

We can gain another factor of two by introducing skew more carefully. If we construct \(E_1\) so that messages “in direction of \(w\)” have delay (roughly) \(d - u\) and messages “in direction of \(v\)” have delay \(d\), we can hide \(u\) skew per hop, just like in Lemma 1.5. We favored the simpler construction to avoid additional bookkeeping.

Overall, if \((\vartheta - 1)d \ll u, \sigma \gg 1, \text{ and } \log \sigma D \gg 1\), we can show a lower bound of \((u - \varepsilon) \log \sigma D\) for some small \(\varepsilon > 0\).

Assuming a similar bunch of reasonable things and that \(T \in \mathcal{O}(d)\) (i.e., message frequency is not the bottleneck in determining estimates), the asymptotically optimal choice of \(\mu\) we computed in the exercises yields a skew of roughly \(2u \log \sigma D\) for our GCS algorithm. Thus, this lower bound shows that the algorithm is optimal up to a factor of roughly 2, provided \(\sigma \gg 1\) and \((\vartheta - 1)d \ll u\). Dropping that \(\sigma \gg 1\), we still get optimality up to a constant factor.

So what of the case that \((\vartheta - 1)d\) is comparable to \(u\) or even larger? Recall that we have shown how to generate a better “logical hardware clock” in this case by bouncing messages back and forth between nodes. Using this idea (with some modifications and the occasional atrocity), one could, up to an additive \(\mathcal{O}((\vartheta - 1)d)\), eliminate the dependence of the upper bound on \((\vartheta - 1)d\).

As for a lower bound construction we can always pretend that clock drifts are actually smaller, e.g., \(\vartheta' := \min\{\vartheta, 1 + u/(4d)\}\), the lower bound is asymptotically optimal in all cases...

...except for unbounded clock rates, which we will deal with next.

### 3.2 Lower Bound with Arbitrary Clock Rates

It can be shown that clock rates \(l_v(t) \in \omega(1)\) do not help. That is, if \((\vartheta - 1)d < u/4\), we have that \(L \in \Omega(u \log_{l_v/(\vartheta - 1)} D)\). However, the only (currently known) proof for this is tedious, to the point where it conveys little insight regarding what’s going on. Hence, we will settle for a (much) simpler argument by Fan and Lynch showing a slightly weaker lower bound, followed by some intuition as to why the stronger result is true as well.

We need a technical lemma stating that, provided that we leave some slack in terms of clock drifts and message delays, we can introduce \(\Omega(u)\) hardware clock skew between any pair of neighbors in an indistinguishable manner. As this follows from repetition of previous arguments, we skip the proof.

**Lemma 3.3.** Let \(E\) be any execution in which clock rates are at most \(1 + (\vartheta - 1)/2\) and message delays are in the range \((d - 3u/4, d - u/4)\). Then, for any \(\langle v, w \rangle \in E\) and sufficiently large times \(t\), there is an indistinguishable execution \(E_v\) such that \(L^E_v(t) = L^E_v(t + u/4)\) and \(L^E_w(t) = L^E_w(t)\).
3.2. LOWER BOUND WITH ARBITRARY CLOCK RATES

Proof Sketch. The general idea is to use the remaining slack of \( u/2 \) to hide the additional skew, and the slack in the clock rates to introduce it. We can do this as slowly as needed, just as in the proof of Lemma 1.5. Again, we can choose the clock rates according to the function \( d(x) \) defined in Lemma 3.2; as \( v \) and \( w \) are neighbors here, it can only take on values of \(-1, 0, \) or \(1\).

This is all we need to generalize our lower bound to arbitrarily large logical clock rates.

Theorem 3.4. Assume that \( \vartheta \leq 2 \). Any algorithm for the gradient clock synchronization problem with logical clock rates of at least \( 1 \) incurs a worst-case gradient skew of

\[
\mathcal{L} \in \Omega \left( \frac{u}{4} - (\vartheta - 1)d \right) \log_{(\log D) / (\vartheta - 1)} D.
\]

Proof. Set \( u' := u/2, d' := d - u/4 \), and \( \vartheta' := 1 + (\vartheta - 1)/2 \). We perform the exact same construction as in Theorem 3.1, with three modifications. First, \( u, d, \) and \( \vartheta \) are replaced by \( u', d', \) and \( \vartheta' \). Second, before starting the construction, we wait for sufficiently long so that Lemma 3.3 is applicable to all times when we actually “work,” i.e., we let the algorithm run for the required time with hardware clock rates of 1 and message delays of \( d' - u'/2 \). Third, we assume that \( \mu = \log_{1/(\vartheta - 1)} D \) in the construction; if ever we attempt to use this (assumed) bound on the clock rates in an inequality and it does not hold, the construction fails.

Now two things can happen. The first is that the construction succeeds. Note that we may assume that \( u'/4 > (\vartheta' - 1)d' \), as otherwise \( u/4 < (\vartheta - 1)d \), i.e., nothing is to show. Thus, the construction shows a lower bound of

\[
\left( \frac{u'}{4} - (\vartheta' - 1)d' \right) \log_{\log(\sigma)} D > \left( \frac{u}{8} - \frac{(\vartheta - 1)d}{2} \right) \log_{\mu/(\vartheta - 1)} D
\]

\[
\in \Omega \left( \frac{u}{4} - (\vartheta - 1)d \right) \log_{\mu/(\vartheta - 1)} D.
\]

As

\[
\log_{\mu/(\vartheta - 1)} D = \frac{\log D}{\log \mu - \log(\vartheta - 1)}
\]

\[
= \frac{\log D}{\log D - \log(\vartheta - 1)} - \log(\vartheta - 1)
\]

\[
\in \Omega \left( \log \log D - \log(\vartheta - 1) \right)
\]

\[
= \Omega \left( \log_{(\log D) / (\vartheta - 1)} D \right),
\]

the claim follows in this case.

On the other hand, if the construction fails, there is an index \( i < i_{\text{max}} \) for which (3.1) does not hold — this is the only place where we make use of the fact that logical clocks do not run faster than rate \( \mu \). Thus,

\[
L_{i_{\text{max}}}^{(\varepsilon)}(t_{i+1}) - L_{i_{\text{max}}}^{(\varepsilon)}(t_i) > \mu(t_{i+1} - t_i)
\]
for some \( i < i_{\text{max}} \). Recall that in the construction, \( d(v, w) = b^{i_{\text{max}} - i} \geq b \) and

\[
t_{i+1} - t_i = d + \left( \frac{u}{2(\vartheta - 1)} - d \right) \frac{d(v, w)}{b} > \frac{u}{2(\vartheta - 1)} - d > \frac{u}{4(\vartheta - 1)} \geq \frac{u}{4}.
\]

Hence, there must be a time \( t \geq t_i \) so that

\[
L_{w}(E_{w})(t + \frac{u}{4}) - L_{w}(E_{\upsilon})(t) > \frac{\mu u}{4}.
\]

Let \( x \in N_w \) be arbitrary. By Lemma 3.3, we can construct an execution \( E_{w} \) so that

\[
L_{w}(E_{w})(t) = L_{w}(E_{\upsilon})(t + \frac{u}{4}) > L_{w}(E_{\upsilon})(t) + \frac{\mu u}{4}
\]

and \( L_{x}(E_{w})(t) = L_{x}(E_{\upsilon})(t) \). Thus, in at least one of the executions, the local skew exceeds

\[
\frac{\mu u}{8} = \frac{u}{8} \log \frac{1}{(\vartheta - 1)} D.
\]

We conclude this chapter with the promised intuition regarding the influence of \( D \) on the base of the logarithm. Consider a path of length \( k \) with a skew of exactly \( \alpha \) per hop, for a total of \( \alpha k \) between its endpoints. Now suppose that an algorithm cleverly uses a large logical clock rate, perfectly reducing the skew at the same rate between any pair of neighbors. Consider the point in time when the skew has been reduced to, say, \( \alpha - \frac{u}{8} \) per hop. The node in the middle of the path has increased its logical clock at half the rate of the endpoint that’s catching up — and the nodes in between have been even faster! Denoting this rate by \( r \), slipping in hardware clock skew at rate \( \vartheta - 1 \) means adding logical clock skew at rate at least \( \frac{r}{2} \). So, even if it takes factor \( r \) less time to reduce the skew to, say \( \alpha - \frac{u}{8} \) per hop than it would for \( \mu = 1 \), it also takes factor \( \frac{r}{2} \) less time to build up additional skew. We would end up with the same result!

Remarks:

- Unfortunately, molding this idea into a proof is challenging, and the result is not pretty.
- The \( D \) in the base of the logarithm is of little importance unless clocks are of poor quality. A standard quartz oscillator guarantees that \( \vartheta - 1 \leq 10^{-5} \). Even a gigantic diameter of \( 10^5 \) would not affect the bound by more than a factor 2 for such clocks!
- The assumption that \( \vartheta \leq 2 \) in Theorem 3.4 is an artifact of the proof. However, hardware clocks that are this inaccurate hardly deserve the name “clock,” so this corner case is not of interest.
- Overall, the GCS algorithm from the previous lecture appears to be optimal or very close to optimal for essentially all choices of parameters.
- Don’t fall into the trap of forgetting that relaxing the model enables better solutions! For instance, if it is not important that clocks make progress at all times (or most of the time), constant local skew can be achieved (buzzword: \( \alpha \)-synchronizer)!
Bibliographic Notes

There is not much to add to the notes for the previous lecture. The seminal paper by Fan and Lynch [FL06] introducing the problem provided Theorem 3.4. Meier and Thiele show that essentially the same lower bound arises from bounded communication rates, without uncertainty (i.e., $u = 0$)?]. Theorem 3.1 follows [LLW10], which also tightens the lower bound for unbounded clock rates by removing the $D$ from the base of the logarithm. In the dynamic setting, one can show bounds on how quickly an edge can be incorporated into the subgraph of edges that satisfy the skew bounds, and asymptotic optimality can be achieved simultaneously with other guarantees [KLO11, KLOO10].

Bibliography


Lecture 4

Fault-Tolerant Clock Synchronization

In the previous lectures, we assumed that the world is a happy place without any kind of faults. This is not a realistic assumption in large-scale systems, and it is an issue in high reliability systems as well. After all, if the system clock fails, there may be no further computations at all!

As, in general, it is difficult to predict what kind of faults may happen, again we assume a worst-case model: Failing nodes may behave in any conceivable manner, including collusion, predicting the future, sending conflicting information to different nodes, or even pretending to be correct nodes (for a while). In other words, the system should still function no matter what kind of faults may occur. This may be overly pessimistic in the sense that “real” faults might have a very hard time to produce such behavior. However, if we can handle all of these possibilities, we’re on the safe side in that we do not have to study what kind of faults may actually happen and verify the resulting fault model(s) for each and every system we build.

Definition 4.1 (Byzantine Faults). A Byzantine faulty node may behave arbitrarily, i.e., it does not follow any algorithm described by the system designer. The set of faulty nodes is (initially) unknown to the other nodes. In other words, the algorithm must be designed in such a way that it works correctly regardless of which nodes are faulty. “Working correctly” here means that all requirements and guarantees on clocks, skew, etc. need only be satisfied by the set $V_0$ of nodes that are not faulty.

Unsurprisingly, such a strong fault model results in limitations on what can be achieved. For instance, if more than half of the nodes in the system are faulty, there is no way to achieve any kind of synchronization. In fact, even if half of the neighbors of some node are faulty, this is impossible. The intuition is simple: Split the neighborhood of some node $v$ in two sets $A$ and $B$ and consider two executions, $E_A$ and $E_B$, such that $A$ is faulty in $E_A$ and $B$ is faulty in $E_B$. Given that $A$ is faulty in $E_A$, $B$ and $v$ need to stay synchronized in $E_A$, regardless of what the nodes in $A$ do. However, the same applies to $E_B$ with the roles of $A$ and $B$ reversed. However, $A$ and $B$ can have different opinions on the time, and $v$ has no way of figuring out which set to trust.
In fact, it turns out that the number \( f \) of faulty nodes must satisfy \( 3f < n \) or no solution is possible (without cryptographic assumptions); we show this later. Motivated by the above considerations, we also confine ourselves to \( G \) being a complete graph: each node is connected to each other node, i.e., each pair of nodes can communicate directly.

### 4.1 The Pulse Synchronization Problem

Let’s study a simpler version of the clock synchronization problem, which we call pulse synchronization. Instead of outputting a logical clock at all times, nodes merely need to jointly generate roughly synchronized pulses whose frequency is bounded from above and below.

**Definition 4.2 (Pulse Synchronization).** Each (non-faulty) node is to generate each pulse \( i \in \mathbb{N} \) exactly once. Denoting by \( p_{v,i} \) the time when node \( v \) generates pulse \( i \), we require that there are \( S, P_{\min}, P_{\max} \in \mathbb{R}^+ \) so that

- \( \max_{i \in \mathbb{N}, v, w \in V} \{|p_{v,i} - p_{w,i}|\} \leq S \) (skew)
- \( \min_{i \in \mathbb{N}} \{\min_{v \in V} \{p_{v,i+1}\} - \max_{v \in V} \{p_{v,i}\}\} \geq P_{\min} \) (minimum period)
- \( \max_{i \in \mathbb{N}} \{\max_{v \in V} \{p_{v,i+1}\} - \min_{v \in V} \{p_{v,i}\}\} \leq P_{\max} \) (maximum period)

**Remarks:**

- The idea is to interpret the pulses as the “ticks” of a common clock.
- Ideally, \( S \) is as small as possible, while \( P_{\min} \) and \( P_{\max} \) are as close to each other as possible and can be scaled freely.
- Due to the lower bound from Lecture 1, we have that \( S \geq u/2 \).
- Clearly, we cannot expect better than \( P_{\max} \geq \vartheta P_{\min} \), i.e., matching the quality of the hardware clocks. Also, \( P_{\max} - P_{\min} \geq S \).
- Because \( D = 1 \), the problem would be trivial without faults. For instance, the Max Algorithm would achieve skew \( u + (\vartheta - 1)(d + T) \), and pulses could be triggered every \( \Theta(G) \) local time.
- The difficulty lies in preventing the faulty nodes from dividing the correctly functioning nodes into unsynchronized subsets.

### 4.2 A Variant of the Srikanth-Toueg Algorithm

One of our design goals here is to keep the algorithm extremely simple. To this end, we decide that

- Nodes will communicate by broadcast (i.e., sending the same information to all other nodes, for simplicity including themselves) only. Note that faulty nodes do not need to stick to this rule!
- Messages are going to be very short. In fact, there is only a single type of message, carrying the information that a node transitioned to state \( \text{PROPOSE} \).
4.2. A VARIANT OF THE SRIKANTH-TOUEG ALGORITHM

- Nodes will store, for each node, whether they received such a message. On some state transitions, they will reset these memory flags to 0 (i.e., no message received yet).

- Not accounting for the memory flags, each node runs a state machine with a constant number of states.

- Transitions in this state machine are triggered by expressions involving (i) the own state, (ii) thresholds for the number of memory flags that are 1, and (iii) timeouts. A timeout means that a node waits for a certain amount of local time after entering a state before considering a timeout expired, i.e., evaluating the respective expression to \( \text{true} \). The only exception is the starting state \( \text{reset} \), from which nodes transition to \( \text{start} \) when the local clock reaches \( H_0 \), where we assume that \( \max_{v \in V_G} \{ H_v(0) \} < H_0 \).

The algorithm, from the perspective of a node, is depicted in Figure 4.1. The idea is to repeat the following cycle:

- At the beginning of an iteration, all nodes transition to state \( \text{READY} \) (or, initially, \( \text{START} \)) within a bounded time span. This resets the flags.

![State machine of a node in the pulse synchronisation algorithm.](image)

<table>
<thead>
<tr>
<th>Guard</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>G1</td>
<td>( H_v(t) = H_0 )</td>
</tr>
<tr>
<td>G2</td>
<td>( (T_1) ) expires or ( &gt; f ) PROPOSE flags set</td>
</tr>
<tr>
<td>G3</td>
<td>( \geq n - f ) PROPOSE flags set</td>
</tr>
<tr>
<td>G4</td>
<td>( (T_2) ) expires</td>
</tr>
<tr>
<td>G5</td>
<td>( (T_3) ) expires or ( &gt; f ) PROPOSE flags set</td>
</tr>
</tbody>
</table>

Figure 4.1: State machine of a node in the pulse synchronisation algorithm. State transitions occur when the condition of the guard in the respective edge is satisfied (gray boxes). All transition guards involve checking whether a local timer expires or a node has received PROPOSE messages from sufficiently many different nodes. The only communication is that a node broadcasts to all nodes (including itself) when it transitions to \( \text{PROPOSE} \). The notation \( \langle T \rangle \) evaluates to \( \text{true} \) when \( T \) time units have passed on the local clock since the transition to the current state. The boxes labeled \( \text{PROPOSE} \) indicates that a node clears its PROPOSE memory flags when transitioning from \( \text{RESET} \) to \( \text{START} \) and \( \text{PULSE} \) to \( \text{READY} \). That is, the node forgets who it has “seen” in PROPOSE at some point in the previous iteration. All nodes initialize their state machine to state \( \text{RESET} \), which they leave at the time \( t \) when \( H_v(t) = H_0 \). Whenever a node transitions to state \( \text{PULSE} \), it generates a pulse. The constraints imposed on the timeouts are listed in Inequalities (4.1)–(4.4).
• Nodes wait in this state until they are sure that all correct nodes reached it. Then, when a local timeout expires, they transition to \textit{propose}.

• When it looks like all correct nodes (may) have arrived there, they transition to \textit{pulse}. As the faulty nodes may never send a message, this means to wait for \(n - f\) nodes having announced to be in \textit{propose}.

• However, faulty nodes may also send \textit{propose} messages, meaning that the threshold is reached despite some nodes still waiting in \textit{ready} for their timeouts to expire. To “pull” such stragglers along, nodes will also transition to \textit{propose} if more than \(f\) of their memory flags are set. This is proof that at least one correct node transitioned to \textit{propose} due to its timeout expiring, so no “early” transitions are caused by this rule.

• Thus, if any node hits the \(n - f\) threshold, no more than \(d\) time later each node will hit the \(f + 1\) threshold. Another \(d\) time later all nodes hit the \(n - f\) threshold, i.e., the algorithm has skew \(2d\).

• The nodes wait in \textit{pulse} sufficiently long to ensure that no \textit{propose} messages are in transit any more before transitioning to \textit{ready} and starting the next iteration.

For this reasoning to work out, a number of timing constraints need to be satisfied:

\[
H_0 > \max_{v \in V_g} \{H_v(0)\} \quad (4.1)
\]

\[
\frac{T_1}{\vartheta} \geq H_0 \quad (4.2)
\]

\[
\frac{T_2}{\vartheta} \geq 3d \quad (4.3)
\]

\[
\frac{T_3}{\vartheta} \geq \left(1 - \frac{1}{\vartheta}\right) T_2 + 2d \quad (4.4)
\]

\textbf{Lemma 4.3.} Suppose \(3f < n\) and the above constraints are satisfied. Moreover, assume that each \(v \in V_g\) transitions to \textit{start} (\textit{ready}) at a time \(t_v \in [t - \Delta, t]\), no such node transitions to \textit{propose} during \((t - \Delta - d, t_v)\), and \(T_1 \geq \vartheta \Delta \) \((T_3 \geq \vartheta \Delta)\). Then there is a time \(t' \in (t - \Delta + \frac{T_1}{\vartheta}, t + T_1 - d)\) \((t' \in (t - \Delta + \frac{T_3}{\vartheta}, t + T_3 - d))\) such that each \(v \in V_g\) transitions to \textit{pulse} during \([t', t' + 2d]\).

\textbf{Proof.} We perform the proof for the case of \textit{start} and \(T_1\); the other case is analogous. Denote by \(t_p\) the smallest time larger than \(t - \Delta - d\) when some \(v \in V_g\) transitions to \textit{propose} (such a time exists, as \(T_1\) will expire if a node does not transition to \textit{propose} before this happens). By assumption and the definition of \(t_p\), no \(v \in V_g\) transitions to \textit{propose} during \((t - \Delta - d, t_p)\), implying that no node receives a message from any such node during \([t - \Delta, t_p]\). As \(v \in V_g\) clears its memory flags when transitioning to \textit{ready} at time \(t_v \geq t - \Delta\), this implies that the node(s) from \(V_g\) that transition to \textit{propose} at time \(t_p\) do so because \(T_1\) expired. As hardware clocks run at most at rate \(\vartheta\) and for each \(v \in V_g\) it holds that \(t_v \geq t - \Delta\), it follows that

\[
t_p \geq t - \Delta + \frac{T_1}{\vartheta} \geq t.
\]
Thus, at time \( t_p \geq t \), each \( v \in V_g \) has reached state \textit{READY} and will not reset its memory flags again without transitioning to \textit{PULSE} first.

From this observation we can infer that each \( v \in V_g \) will transition to \textit{PULSE}: Each \( v \in V_g \) transitions to \textit{PROPOSE} during \([t_p, t+T_1]\), as it does so at the latest at time \( t_v + T_1 \leq t + T_1 \) due to \( T_1 \) expiring. Thus, by time \( t + T_1 + d \) each \( v \in V_g \) received the respective messages and, as \( |V_g| \geq n - f \), transitioned to \textit{PULSE}.

It remains to show that all correct nodes transition to \textit{PULSE} within \( 2d \) time. Let \( t' \) be the minimum time after \( t_p \) when some \( v \in V_g \) transitions to \textit{PULSE}. If \( t' \geq t + T_1 - d \), the claim is immediate from the above observations. Otherwise, note that out of the \( n - f \) of \( v \)'s flags that are \textit{true}, at least \( n - 2f > f \) correspond to nodes in \( V_g \). The messages causing them to be set have been sent at or after time \( t_p \), as we already established that any flags that were raised earlier have been cleared before time \( t \leq t_p \). Their senders have broadcasted their transition to \textit{PROPOSE} to all nodes, so any \( w \in V_g \) has more than \( f \) flags raised by time \( t' + d \), where \( d \) accounts for the potentially different travelling times of the respective messages. Hence, each \( w \in V_g \) transitions to \textit{PROPOSE} before time \( t' + d \), the respective messages are received before time \( t' + 2d \), and, as \( |V_g| \geq n - f \), each \( w \in V_g \) transitions to \textit{PULSE} during \([t', t' + 2d]\).

\[ \text{Theorem 4.4.} \text{ Suppose that } 3f < n \text{ and the above constraints are satisfied. Then the algorithm given in Figure 4.1 solves the pulse synchronization problem with } S = 2d, P_{\text{min}} = (T_2 + T_3)/\vartheta - 2d \text{ and } P_{\text{max}} = T_2 + T_3 + 3d. \]

\[ \text{Proof.} \text{ We prove the claim by induction on the pulse number. For each pulse, we invoke Lemma 4.3. The first time, we use that all nodes start with hardware clock values in the range } [0, H_0) \text{ by (4.1). As hardware clocks run at least at rate 1, thus all nodes transition to state START by time } H_0. \text{ By (4.2), the lemma can be applied with } t = \Delta = H_0, \text{ yielding times } p_{v,1}, v \in V_g, \text{ satisfying the claimed skew bound of } 2d. \]

For the induction step from \( i \) to \( i + 1 \), (4.3) yields that \( v \in V_g \) transitions to \textit{READY} no earlier than time

\[ p_{v,i} + \frac{T_2}{\vartheta} \geq \max_{w \in V_g} \{p_{w,i}\} + \frac{T_2}{\vartheta} - 2d \geq \max_{w \in V_g} \{p_{w,i}\} + d \]

and no later than time

\[ p_{v,i} + T_2 \leq \max_{w \in V_g} \{p_{w,i}\} + T_2. \]

Thus, by (4.4) we can apply Lemma 4.3 with \( t = \max_{w \in V_g} \{p_{w,i}\} + T_2 \) and \( \Delta = (1 - 1/\vartheta)T_2 + 2d \), yielding pulse times \( p_{v,i+1}, v \in V_g \), satisfying the stated skew bound.

It remains to show that \( \min_{v \in V_g} \{p_{v,i+1}\} - \max_{v \in V_g} \{p_{v,i}\} \geq (T_2 + T_3)/\vartheta - 2d \) and \( \max_{v \in V_g} \{p_{v,i+1}\} - \min_{v \in V_g} \{p_{v,i}\} \leq T_2 + T_3 + 3d. \) By Lemma 4.3,

\[ p_{v,i+1} \in \left( t - \Delta + \frac{2T_2}{\vartheta}, t + T_3 + d \right) \]

\[ = \left( \max_{w \in V_g} \{p_{w,i}\} + \frac{T_2 + T_3}{\vartheta} - 2d, \max_{w \in V_g} \{p_{w,i}\} + T_2 + T_3 + d \right). \]

Thus, the first bound is satisfied. The second follows as well, as we have already shown that \( \max_{w \in V_g} \{p_{w,i}\} \leq \min_{w \in V_g} \{p_{w,i}\} + 2d \). \[ \square \]
Remarks:

- The skew bound of $2d$ can be improved to $d + u$ by a more careful analysis; you’ll show this as an exercise.
- By making $T_2 + T_3$ large, the ratio $P_{\text{max}}/P_{\text{min}}$ can be brought arbitrarily close to $\vartheta$.
- On the other hand, we can go for the minimal choice $T_2 = 3\vartheta d$ and $T_3 = (3\vartheta^2 - \vartheta)d$, yielding $P_{\text{min}} = 3\vartheta d$ and $P_{\text{max}} = (3\vartheta^2 + 2\vartheta + 2)d$.

4.3 Impossibility of Synchronization for $3f \geq n$

If $3f \geq n$, the faulty nodes can force correct nodes to lose synchronization in some executions. We will use indistinguishability again, but this time there will always be some correct nodes who can see a difference. The issue is that they cannot prove to the other correct nodes that it’s not them who are faulty.

We partition the node set into three sets $A, B, C \subset V$ so that $|A|, |B|, |C| \leq f$. We will construct a sequence of executions showing that either synchronization is lost in some execution (i.e., any finite skew bound $S$ is violated) or the algorithm cannot guarantee bounds on the period. In each execution, one of the sets consists entirely of faulty nodes. In each of the other sets, the hardware clocks of all nodes will be identical. The same holds for the faulty set, but the

<table>
<thead>
<tr>
<th>$\mathcal{E}_0$</th>
<th>$H_A(t)$</th>
<th>$H_B(t)$</th>
<th>$H_C(t)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\rho t$</td>
<td>$\rho^2 t$</td>
<td>$\leftarrow$ arbitrary $t \rightarrow$</td>
<td></td>
</tr>
<tr>
<td>$\mathcal{E}_1$</td>
<td>$\rho^2 t$</td>
<td>$\leftarrow \rho^3 t$</td>
<td>$t \rightarrow \rho t$</td>
</tr>
<tr>
<td>$\mathcal{E}_2$</td>
<td>$\leftarrow \rho^4 t$</td>
<td>$t \rightarrow \rho t$</td>
<td>$\rho^2 t$</td>
</tr>
<tr>
<td>$\mathcal{E}_3$</td>
<td>$\rho t$</td>
<td>$\rho^2 t$</td>
<td>$\leftarrow \rho^4 t$</td>
</tr>
<tr>
<td>$\mathcal{E}_4$</td>
<td>$\rho^2 t$</td>
<td>$\leftarrow \rho^3 t$</td>
<td>$t \rightarrow \rho t$</td>
</tr>
<tr>
<td>$\mathcal{E}_5$</td>
<td>$\leftarrow \rho^4 t$</td>
<td>$t \rightarrow \rho t$</td>
<td>$\rho^2 t$</td>
</tr>
<tr>
<td>$\mathcal{E}_6$</td>
<td>$\rho t$</td>
<td>$\rho^2 t$</td>
<td>$\leftarrow \rho^3 t$</td>
</tr>
</tbody>
</table>

Table 4.1: Hardware clock speeds in the different executions for the different sets. The red entries indicate faulty sets, simulating a clock speed of $\rho^3 t$ to the set “to the left” and $t$ to the set “to the right.” For $k \in \mathbb{N}_0$, execution pairs ($\mathcal{E}_{3k}, \mathcal{E}_{3k+1}$) are indistinguishable to nodes in $A$, pairs ($\mathcal{E}_{3k+1}, \mathcal{E}_{3k+2}$) are indistinguishable to nodes in $C$, and pairs ($\mathcal{E}_{3k+2}, \mathcal{E}_{3k+3}$) are indistinguishable to nodes in $B$. That is, in $\mathcal{E}_i$ faulty nodes mimic the behavior they have in $\mathcal{E}_{i-1}$ to the set left of them, and that from $\mathcal{E}_{i+1}$ to the set to the right.
nodes there play both sides differently: to one set, they make their clocks appear to be very slow, to the other they make them appear fast. All clock rates (actual or simulated) will lie between 1 and \( \rho^3 \), where \( \rho > 1 \) is small enough so that \( \rho^3 \leq \vartheta \) and \( d \leq \rho^3(d - u) \); this way, message delays can be chosen such that messages arrive at the same local times without violating message delay bounds.

Note that for each pair of consecutive executions, the executions are indistinguishable to the set that is correct in both of them and a factor of \( \rho > 1 \) lies between the speeds of hardware clocks. This means that the pulses are generated at a by factor \( \rho \) higher speed. However, as the skew bounds are to be satisfied, this means that also the set of correct nodes that knows that something is different will have to generate pulses faster. This means that in execution \( \mathcal{E}_i \), pulses are generated at an amortized rate of \( \rho P_{\text{max}} / P_{\text{min}} \). For \( i > \log \rho P_{\text{max}} / P_{\text{min}} \), this is a contradiction.

Lemma 4.4. Suppose \( 3f \geq n \). Then, for any algorithm \( A \), there exists \( \rho > 1 \) and a sequence of executions \( \mathcal{E}_i, i \in \mathbb{N}_0 \), with the properties stated in Table 4.1.

Proof. Choose \( \rho := \min \left\{ \vartheta, \frac{d}{d - u} \right\}^{1/3} \). We construct the entire sequence concurrently, where we advance real time in execution \( \mathcal{E}_i \) at speed \( \rho^{-i} \). All correct nodes run \( A \), which specifies the local times at which these nodes send messages as well as their content. We maintain the invariant that the constructed parts of the executions satisfy the stated properties. In particular, this defines the hardware clocks of correct nodes at all times. Any message a node \( v \) (faulty or not) sends at time \( t \) to some node \( w \) is received at local time \( H_w(t) + d \). By the choice of \( \rho \), this means that all hardware clock rates (of correct nodes) and message delays are within the required bounds, i.e., all constructed executions are feasible.

We need to specify the messages sent by faulty nodes in a way that achieves the desired indistinguishability. To this end, consider the set of faulty nodes in execution \( \mathcal{E}_i, i \in \mathbb{N}_0 \). If in execution \( \mathcal{E}_{i+1} \) such a node \( v \) sends a message to some \( w \) in the “right” set (i.e., \( B \) is right of \( A \), \( C \) of \( B \), and \( A \) of \( C \)) at time \( t = H_v^{(\mathcal{E}_i)}(t)/\rho \), it sends the same message in \( \mathcal{E}_i \) at time \( t \). Thus, it is received at local time

\[
H_w^{(\mathcal{E}_i)}(t) + d = \rho t + d = H_w^{(\mathcal{E}_{i+1})}(t) + d.
\]

Similarly, consider the set of faulty nodes in execution \( \mathcal{E}_i, i \in \mathbb{N} \). If in execution \( \mathcal{E}_{i-1} \) a node \( v \) from this set sends a message to some \( w \) in the “left” set (i.e., \( A \) is left of \( B \), \( B \) of \( C \), and \( C \) or \( A \)) at time \( t = H_v^{(\mathcal{E}_{i-1})}(t)/\rho^2 \), it sends the same message in \( \mathcal{E}_i \) at time \( t/\rho^3 \). Thus, it is received at local time

\[
H_w^{(\mathcal{E}_i)} \left( \frac{t}{\rho^3} \right) + d = \frac{t}{\rho} + d = H_w^{(\mathcal{E}_{i-1})}(t) + d.
\]

Together, this implies that for \( k \in \mathbb{N}_0 \), execution pairs \( \mathcal{E}_{3k}, \mathcal{E}_{3k+1} \) are indistinguishable to nodes in \( A \), pairs \( \mathcal{E}_{3k+1}, \mathcal{E}_{3k+2} \) are indistinguishable to nodes in \( C \), and pairs \( \mathcal{E}_{3k+2}, \mathcal{E}_{3k+3} \) are indistinguishable to nodes in \( B \), as claimed. Note that it does not matter which messages are sent from the nodes in \( C \) to nodes in \( B \) in execution \( \mathcal{E}_0 \); for example, we can rule that they send no messages to nodes in \( B \) at all.
It might seem as if the proof were complete. However, each execution is defined in terms of others, so it is not entirely clear that the above assignment is possible. This is where we use the aforementioned approach of “constructing execution $E_i$ at speed $\rho^{-i}$.” Think of each faulty node as simulating two virtual nodes, one for messages sent “to the left,” which has local time $\rho^jt$ at time $t$, and one for messages sent “to the right,” which has local time $t$ at time $t$. This way, there is a one-to-one correspondence between the virtual nodes of a faulty node $v$ in execution $E_i$ and the corresponding nodes in executions $E_{i-1}$ and $E_{i+1}$, respectively (up to the case $i = 0$, where the “left” virtual nodes do not send messages). If a faulty node $v$ needs to send a message in execution $E_i$, the respective virtual node sends the message at the same local time as $v$ sends the message in execution $E_{i-1}$ (left) or $E_{i+1}$ (right). In terms of real time, there is exactly a factor of $\rho$: if $v$ is faulty in $E_i$ and wants to determine the behavior of its virtual node corresponding to $E_{i-1}$ up to time $t$, it needs to simulate $E_{i-1}$ up to time $\rho t$; similarly, when doing the same for its virtual node corresponding to $E_{i+1}$, it needs to simulate $E_{i+1}$ up to time $t/\rho$. Thus, when simulating all executions concurrently, where $E_i$ progresses at rate $\rho^{-i}$, at all times the behavior of faulty nodes according to the above scheme can be determined. This completes the proof. 

\[ \square \]

**Theorem 4.6.** Pulse synchronization is impossible if $3f \geq n$.

**Proof.** Assume for contradiction that there is an algorithm solving pulse synchronization. We apply Lemma 4.5, yielding a sequence of executions $E_i$ with the properties stated in Table 4.1. We will show that pulses are generated arbitrarily fast, contradicting the minimum period requirement. We show this by induction on $i$, where the induction hypothesis is that there is some $v \in V_g$ satisfying that
\[
P^{(E_i)}_{v, j} - P^{(E_i)}_{v, 1} \leq (j - 1)\rho^{-i}P_{\max} + 2iS
\]
for all $j \in \mathbb{N}_0$, where $\rho > 1$ is given by Lemma 4.5. This is trivial for the base case $i = 0$ by the maximum period requirement.

For the induction step from $i$ to $i + 1$, let $v \in V_g$ be a node with $P^{(E_i)}_{v, j} - P^{(E_i)}_{v, 1} \leq (j - 1)\rho^{-i}P_{\max} + 2iS$ for all $j \in \mathbb{N}_0$. Let $w \in V_g$ be a node that is correct in both $E_i$ and $E_{i+1}$. By the skew bound,
\[
P^{(E_{i+1})}_{w, j} - P^{(E_{i+1})}_{w, 1} \leq P^{(E_i)}_{v, j} - P^{(E_i)}_{v, 1} + 2S \leq (j - 1)\rho^{-i}P_{\max} + 2(i + 1)S
\]
for all $j \in \mathbb{N}_0$. By Lemma 4.5, $w$ cannot distinguish between $E_i$ and $E_{i+1}$. Because $H^{(E_{i+1})}_{w}(t/\rho) = \rho t = H^{(E_{i+1})}_{w}(t)$, we conclude that $P^{(E_{i+1})}_{w, j} = \rho^{-1}P^{(E_i)}_{w, j}$ for all $j \in \mathbb{N}_0$. Hence,
\[
P^{(E_{i+1})}_{w, j} - P^{(E_{i+1})}_{w, 1} \leq \rho^{-1} \left( P^{(E_i)}_{w, j} - P^{(E_i)}_{w, 1} \right) \leq (j - 1)\rho^{-1}P_{\max} + 2(i + 1)S
\]
for all $j \in \mathbb{N}_0$, completing the induction step.

Now choose $i \in \mathbb{N}$ large enough so that $\rho^{-i}P_{\max} < P_{\min}$ and let $v \in V_g$ be a node to which the claim applies in $E_i$. Choosing $j - 1 > 2iS(P_{\min} - \rho^{-i}P_{\max})$, it follows that
\[
P^{(E_i)}_{v, j} - P^{(E_i)}_{v, 1} \leq (j - 1)\rho^{-i}P_{\max} + 2iS < (j - 1)P_{\min}.
\]
Hence, the minimum period bound is violated, as there must be some index $j' \in \{1, \ldots, j - 1\}$ for which $P^{(E_i)}_{v, j'} - P^{(E_i)}_{v, 1} < P_{\min}$. 

\[ \square \]
Bibliographic Notes

The algorithm presented in this lecture is a variant of the Srikanth-Toueg algorithm [ST87]. An actual implementation in hardware [FS12] (of another variant) was performed in the DARTS project. In a form close to the one presented here, it was first given in [DFL+15], a survey on fault-tolerant clocking methods for hardware. In all of these cases, the main difference to the original is getting rid of communicating the “tick” number explicitly. The impossibility of achieving synchronization if \( f \geq n/3 \) was first shown in [DHS86]. Conceptually, the underlying argument is related to the impossibility of consensus in synchronous systems with \( f \geq n/3 \) Byzantine faults [PSL80].

Concerning the skew bound, we know that \( u/2 \) skew cannot be avoided from the first lecture. Moreover, \((1 - 1/\vartheta)d/2\) skew cannot be avoided either, as it takes \( d \) time to communicate. Note that the upper bound of \( 2d \) shown here only holds on the real time between corresponding ticks; if we derive continuous logical clocks, we get at least an additional \( \Omega((\vartheta - 1)d) \) contribution to the skew from the hardware clock drift in between ticks, so there is no contradiction. We’ll push the skew down to a matching \( \mathcal{O}(u + (\vartheta - 1)d) \) in the next lecture.

Bibliography


Lecture 5

Synchronizing by Approximate Agreement

In the previous lecture, we’ve seen how to achieve a skew of $O(d)$ in a system of $n$ fully connected nodes with $f < n/3$ Byzantine faults. We’ve also seen that we can’t do any better in terms of the number of faults that can be tolerated. So let’s ask our usual question: Is this skew bound (asymptotically) optimal or can we do better? Already in a fault-free system, we know that we can’t beat $\Omega(u + (\vartheta - 1)d)$. But can this bound be attained in the presence of faults?

5.1 Approximate Agreement

The answer is provided by leveraging techniques for the task of approximate agreement. For this problem, we assume the (convenient) abstraction of a synchronously operating system.

**Definition 5.1 (Synchronous Execution).** A synchronous execution proceeds in synchronous rounds. At the start of the execution, each node receives an input (whose type depends on the task at hand). In each round,

1. nodes perform local computations,
2. send messages to their neighbors in the network graph,
3. receive the messages of their neighbors, and (optionally)
4. may compute an output value and terminate (i.e., stop executing the other steps in future rounds).

Note that a synchronous execution of a deterministic algorithm is fully determined by the input values and the (arbitrary) messages sent by faulty nodes. The key performance measures are the round complexity — the number of rounds until all nodes terminated — and the maximum size of messages (sent by correct nodes).

This model provides a very clean abstraction for describing the tool we would like to use.
Definition 5.2 (Approximate Agreement). Each node \( v \in V \) is given an input value \( r_v \in \mathbb{R} \). Given a constant \( \varepsilon > 0 \), the task is to generate output values \( o_v \in \mathbb{R} \) so that

**agreement:** \( \max_{v, w \in V_g} \{o_v - o_w\} \leq \varepsilon \),

**validity:** \( \forall v \in V_g: \min_{w \in V_g} \{r_w\} \leq o_w \leq \max_{w \in V_g} \{r_w\} \), and

**termination:** each \( v \in V_g \) determines is output \( o_v \) and terminates within a finite number of rounds.

Remarks:

- The synchronous model is a highly useful abstraction in distributed computing. With known upper bounds \( L \) on local skew, \( \lambda \) on logical clock rates, and \( d \) on message delays, it is straightforward to simulate. Assuming that \( \max_{v \in V_g} \{L_v(0)\} = L \), nodes send their messages for round \( r \in \mathbb{N} \) at the time \( t \) when \( L_v(t) = L + (r - 1)\lambda(d + L) \). Thus, all messages for round \( r \) are received before the ones for round \( r + 1 \) need to be sent.

- If the round number is not to be sent along with the message or for some other reason it’s important that messages for round \( r + 1 \) must not arrive anywhere before round \( r \) is complete at all nodes, one may add an additional \( \lambda S \) at the beginning of the round before messages are sent. We will use this in our algorithm!

- Recall that \( d \) accounts for local computations, not only the time messages are in transit. Thus, involved calculations affect the time the simulation takes via \( d \)!

- Lower bounds on the progress of logical clocks are needed for guaranteeing progress. The better the lower bound, the earlier the simulation completes (i.e., all nodes terminate).

- Without faults, synchronizers provide elegant solutions that work even if \( d \) is unknown. However, synchronizers wait for proof that all other nodes finished their current round before proceeding. Even a single crash fault (a node not sending any messages any more) would halt the entire system!

- Once we solved approximate agreement in this abstract model, we will employ it to agree on when the nodes should generate clock pulses, i.e., solve the pulse synchronization problem with it.

- The simulation of the synchronous algorithm and maintaining a small skew will go hand in hand!

Solving Approximate Agreement

Definition 5.3 (Diameters of Vectors). Denote by \( \bar{r} \) the \( |V_g| \)-dimensional vector of correct nodes’ inputs, i.e., \( (\bar{r})_v = r_v \) for \( v \in V_g \). Denote by \( r^{(k)} \), \( k \in \{1, \ldots, |V_g|\} \), the \( k^{th} \) entry when ordering the entries of \( \bar{r} \) ascendingly.
Algorithm 5.1: Approximate agreement step at node \( v \in V_g \) (with synchronous message exchange).

1 // node \( v \) is given input value \( r_v \);
2 broadcast \( r_v \) to all nodes (including self);
3 receive \( \hat{r}_{wv} \) from each node \( w \) (\( \hat{r}_{wv} := r_v \) if no message with correct type of content from \( w \) received);
4 \( S_v \leftarrow \{ \hat{r}_{wv} | w \in V \} \);
5 \( o_v := \frac{S_v^{(f+1)} + S_v^{(n-f)}}{2} \);
6 return \( o_v \);

The diameter \( \|\vec{r}\| \) of \( \vec{r} \) is the difference between the maximum and minimum components of \( \vec{r} \). Formally,

\[
\|\vec{r}\| := r^{(|V_g|)} - r^{(1)} = \max_{r \in V_g} \{r_v\} - \min_{v \in V_g} \{r_v\}.
\]

We will use the same notation for other values, e.g. \( \vec{o}, o^{(k)}, \|\vec{o}\| \), etc.

For simplicity, we assume that \( |V_g| = n - f \) in the following; all statements can be adapted by replacing \( n - f \) with \( |V_g| \) where appropriate. As usual, we require that \( 3f < n \).

Intuitively, Algorithm 5.1 discards the smallest and largest \( f \) values each to ensure that values from faulty nodes cannot cause outputs to lie outside the range spanned by the correct nodes’ values. Afterwards, \( o_v \) is determined as the midpoint of the interval spanned by the remaining values. Since \( f < n/3 \), i.e., \( n - f \geq 2f + 1 \), the median of correct nodes’ values is part of all intervals computed by correct nodes. From this, it is easy to see that \( \|\vec{o}\| \leq \|\vec{r}\|/2 \). We now prove these properties.

**Lemma 5.4.**

\[ \forall v \in V_g: r^{(1)} \leq o_v \leq r^{(n-f)} . \]

**Proof.** As there are at most \( f \) faulty nodes, for \( v \in V_g \) we have that

\[ S_v^{(f+1)} \geq \min_{w \in V_g} \{\hat{r}_{wv}\} = r^{(1)} . \]

Analogously, \( S_v^{(n-f)} \leq r^{(n-f)} \). We conclude that

\[ r^{(1)} \leq S_v^{(f+1)} \leq \frac{S_v^{(f+1)} + S_v^{(n-f)}}{2} = o_v \leq S_v^{(n-f)} \leq r^{(n-f)} . \]

**Lemma 5.5.** \( \|\vec{o}\| \leq \|\vec{r}\|/2 \).

**Proof.** Since \( f < n/3 \), we have that \( n - f \geq 2f + 1 \). Hence, for all \( v \in V_g \),

\[ r^{(1)} \leq S_v^{(f+1)} \leq r^{(f+1)} \leq S_v^{(2f+1)} \leq S_v^{(n-f)} \leq r^{(n-f)} . \]
For any $v, w \in V_g$, it follows that
\[
\omega_v - \omega_w = \frac{S_v^{(f+1)} - S_w^{(f+1)} + S_v^{(n-f)} - S_w^{(n-f)}}{2} \\
\leq \frac{r^{(f+1)} - r^{(1)} + r^{(n-f)} - r^{(f+1)}}{2} = \frac{r^{(n-f)} - r^{(1)}}{2} = \frac{\|\vec{r}\|}{2}.
\]
As $v, w \in V_g$ were arbitrary, this yields $\|\vec{a}\| \leq \|\vec{r}\|/2$. \hfill \Box

Applying this approach inductively yields a straightforward algorithm provided an upper bound $R \geq r(|V_g|) - r^{(1)}$ is known.

**Theorem 5.6 (Approximate Agreement).** Applying Algorithm 5.1 iteratively (using the output of one step as input to the next) for $\lceil \log(R/\varepsilon) \rceil$ steps solves approximate agreement.

**Proof.** Applying Lemma 5.5 inductively shows agreement. Applying Lemma 5.4 inductively shows validity. By construction, all nodes terminate after $\lceil \log(R/\varepsilon) \rceil$ synchronous rounds. \hfill \Box

**Modifications for the Pulse Synchronization Problem**

In our setting, we will not be able to guarantee exact communication of clock values. Accordingly, we slightly modify the communication model. More specifically, at certain times, nodes will need estimates of each other’s logical clock values. Node $v$ will use its estimate of $w$’s clock value as approximation of the “input” $r_w$ of $w \in V$. Thus, instead of receiving $\hat{r}_{wv} = r_w$ from $w \in V$, $v$ will receive
\[
r_w - \delta < \hat{r}_{wv} \leq r_w.
\]
As shifting the values $\hat{r}_{wv}$ in Algorithm 5.1 by less than $\delta$ will affect the outputs by less than $\delta$, we obtain the following corollary to Lemmas 5.4 and 5.5. See Figure 5.1 for a visualization.

**Corollary 5.7.** With the above modification to the communication model, Algorithm 5.1 guarantees
\[
(i) \forall v \in V_g: r^{(1)} - \delta < \omega_v \leq r^{(n-f)} \quad \text{and} \\
(ii) \|\vec{\omega}\| \leq \|\vec{r}\|/2 + \delta.
\]

**Remarks:**

- Now all we need to do is to gather estimates, use Algorithm 5.1 to determine adjustments to the logical clocks, and iterate.
- Trivia: When I suggested to Danny Dolev that one could make use of approximate agreement as the basis for a clock synchronization algorithm, he told me that this was precisely the motivation for introducing the problem and pointed me towards the paper implementing this approach. He and his co-authors were merely about three decades and a brilliant abstraction ahead of me!
5.2 A Variant of the Lynch-Welch Algorithm

The algorithm is now constructed as follows. Assuming some bound \( H \geq \max_{v \in V_g} \{H_v(0)\} \) on the skew at initialization, nodes generate their first pulse at local time \( H \). This marks the (local) start of the first round. Then they wait until they can be sure that all nodes have generated their pulse. At the respective hardware time, they transmit an empty message—no content is needed, as the local time when the message is sent is hardwired into the algorithm. Then

![Figure 5.1: An execution of Algorithm 5.1 at nodes \( v \) and \( w \) of a system consisting of \( n = 4 \) nodes. There is a single faulty node and its values are indicated in red. Note that the ranges spanned by the values received from non-faulty nodes are almost identical; the difference originates in the perturbations of up to \( \delta \).](image)

**Algorithm 5.2:** Lynch-Welch pulse synchronization algorithm, code for node \( v \in V_g \). \( S \) denotes a (to-be-determined) upper bound on \( \|p_v\| \) for each \( r \in \mathbb{N} \) and \( T \) is the nominal round duration.

```plaintext
1 // \( H_v(0) \in [0, S) \) for all \( w \in V \\
2 \text{set } L_v(0) := H_v(0) \\
3 \text{increase } L_v \text{ at rate } h_v \text{ at all times} \\
4 \text{generate pulse } 1 \text{ at the time } p_{v,1} \text{ with } L_v(p_{v,1}) = S; \\
5 \text{foreach round } r \in \mathbb{N} \text{ do} \\
6 \quad \text{wait until local time } (r - 1)T + (\vartheta + 1)S; // all nodes are in round \( r \) \\
7 \quad \text{broadcast empty message to all nodes (including self)}; \\
8 \quad \text{wait until time } \tau_{v,r} \text{ when } L_v(\tau_{v,r}) = (r - 1)T + (\vartheta^2 + \vartheta + 1)S + \vartheta d; \\
9 \quad // \text{correct nodes’ messages arrived} \\
10 \quad \text{for each node } w \in V \text{ do} \\
11 \quad \quad \text{// abbreviate } p_r := \max_{w \in V_g} \{p_{w,r}\} \text{ (unknown to the node!)} \\
12 \quad \quad \text{compute } \Delta_w \equiv (L_v(p_r) - L_w(p_r)) - \delta, L_v(p_r) - L_w(p_r)] \\
13 \quad \quad S_v \leftarrow \{\Delta_w \mid w \in V\} \text{ (as multiset, i.e., values may repeat)} \\
14 \quad \quad L_v(\tau_{v,r}) \leftarrow L_v(\tau_{v,r}) + \left(S_v^{f+1} + S_v^{n-f}\right)/2 \\
15 \quad \quad \text{generate pulse } r + 1 \text{ at the time } p_{v,r+1} \text{ with } L_v(p_{v,r+1}) = S + rT
```
nodes wait until the local time when all such messages from correct nodes are
certainly received and compute their estimates of the relative clock differences
to other nodes. Finally, they apply Algorithm 5.1 to compute an adjustment to
the (local) starting time of the next round. This ensures bounded skew for the
next pulse and thus also the starting times of the next round. From there, the
process is iterated.

Algorithm 5.1 is phrased in a parametrized fashion suitable for the analysis.
This means that we assume a skew bound of $S$ to hold on initialization, an
error bound $δ$ on the logical clock estimates nodes compute of each other, and
a nominal round duration of $T$. We then determine valid choices for these
parameters from the analysis, where we need to determine $δ$ depending on how
the estimates are computed.

“Rounds” of the algorithm simulate the synchronous operation assumed in
the approximate agreement problem, where each iteration of the loop simulates
one synchronous round. For this to work as intended, two requirements need to
be met in each round:

(i) Messages sent by correct nodes are received at all correct nodes after
starting the round and before they compute their clock adjustment, i.e.,
during $[p_{v,r}, τ_{v,r}]$.

(ii) $T$ is large enough to ensure that the clock adjustment makes no logical
clock “jump” past $L_v (p_{v,r+1}) = S + rT$, skipping a pulse.

If these properties are satisfied in round $r$, we will say that round $r$ is executed
correctly. We will show that this holds for all $r ∈ N$ inductively, where the
induction hypothesis is that $∥p_r∥ ≤ S$; this simultaneously shows that the algo-
rithm has a small skew! For $r = 1$, this is immediate from our assumption on
the initial hardware clock values.

**Lemma 5.8.** Suppose that $T/∂ ≥ (∂^2 + 1)S + ∂d$ and $S ≥ (δ + (1−1/∂)T)$. Moreover, assume that for $r ∈ N$ it holds that all prior rounds have been executed
correctly, and that $∥p_r∥ ≤ S$. Then

(i) round $r$ is executed correctly,

(ii) $∥p_{r+1}∥ ≤ S$, and

(iii) $T/∂ + S ≤ p_{r+1} - p_r ≤ T + δ$.

**Proof.** By assumption, no messages sent by correct nodes in rounds $r' < r$ are
received in round $r$. Consider the message $v ∈ V$ sends after entering round
$r$. It is sent no earlier than time $p_{v,r} + S ≥ \max_{w∈V_g} \{p_{w,r}\}$, as $∥p_r∥ ≤ S$ by
assumption. It is received before time

$$p_{v,r} + ∂S + d ≤ \min_{w∈V_g} \{p_{w,r}\} + (∂ + 1)S + d.$$  

As $τ_{w,r} ≥ p_{w,r} + (∂ + 1)S + d$ for all $w ∈ V_g$, this shows part (i) of correct
execution of round $r$. 


As hardware clock rates are at least 1, this shows that our reasoning, pretend that the clock adjustments from round the upper bound of the third claim of the lemma holds.

It follows that no node can reach logical clock value \( r_T + S \) earlier than time \( \min_{w \in V_g} (p_{v,w} + T)/\vartheta \). In particular, this is bounded from below by \( p_r + T/\vartheta - S \), showing the lower bound of the third claim of the lemma.

On the other hand, for all \( v \in V_g \), we have that

\[
\tau_{v,r} \leq p_{v,r} + (\vartheta^2 + \vartheta)S + \vartheta d \leq \min_{w \in V_g} \{ p_{w,r} \} + (\vartheta^2 + \vartheta + 1)S + \vartheta d,
\]

where the second step uses that \( \| p_{v,r} \| \leq S \). As \( T/\vartheta \geq (\vartheta^2 + \vartheta + 1)S + \vartheta d \), this shows that round \( r \) is executed correctly. In particular, the times \( p_{v,r+1} \), \( v \in V_g \), are well-defined.

By statement (i) of Corollary 5.7, we have that, at time \( \tau_{v,r} \), \( v \in V_g \) cannot set its logical clock to a smaller value than

\[
L_v(\tau_{v,r}) \geq L_v(p_r) + \int_{p_r}^{\tau_{v,r}} h_v(t) \, dt + \min_{w \in V_g} \{ L_w(p_r) \} - L_v(p_r) - \delta
\]

where

\[
\geq \int_{p_r}^{\tau_{v,r}} h_v(t) \, dt + \min_{w \in V_g} \{ L_w(p_r) \} - \delta
\]

As hardware clock rates are at least 1, this shows that \( p_{r+1} \leq p_r + T + \delta \), i.e., the upper bound of the third claim of the lemma holds.

It remains to show the second claim, i.e., the bound on the skew. To simplify our reasoning, pretend that the clock adjustments from round \( r \) would take place at time \( p_r \). Denote by \( L'_v(p_r), v \in V_g \), the respective modified logical clocks, which increase at the rate of the hardware clocks during round \( r \) and satisfy \( L'_v(t) = L_v(t) \) at times \( t \geq \tau_{v,r} \). By the above bound, we thus have that

\[
L'_v(p_r) = L_v(\tau_{v,r}) - \int_{p_r}^{\tau_{v,r}} h_v(t) \, dt \geq (r - 1)T + S - \delta.
\]

Next, note that \( \| \hat{L}(p_r) \| \leq \vartheta \| p_r \| \leq \vartheta S \) by assumption. By statement (ii) of Corollary 5.7, this implies that \( \| \hat{L}(p_r) \| \leq \vartheta S/2 + \delta \). Now let \( v, w \in V_g \) maximize \( p_{r+1,v} - p_{r+1,w} \). We have that \( p_{r+1,v} - p_r \leq rT + S - L'_v(p_r) \) and \( p_{r+1,w} - p_r \geq (rT + S - L'_w(p_r))/\vartheta \) due to the bounds on the hardware clock.
rates. Hence,

\[ p_{r+1,v} - p_{r+1,w} \leq L'_w(p_r') - L'_v(p_r') + \left( 1 - \frac{1}{\vartheta} \right) (rT + S - L'_w(p_r')) \]
\[ \leq \|\tilde{E}'(p_r')\| + \left( 1 - \frac{1}{\vartheta} \right) (rT - (L'_v(p_r') + \|\tilde{E}'(p_r')\|)) \]
\[ \leq \|\tilde{E}'(p_r')\| + \left( 1 - \frac{1}{\vartheta} \right) (T + \vartheta - \|\tilde{E}'(p_r')\|) \]
\[ \leq \frac{\vartheta S}{2} + \delta + \left( 1 - \frac{1}{\vartheta} \right) \left( T - \frac{\vartheta S}{2} \right) \]
\[ = \frac{S}{2} + \delta + \left( 1 - \frac{1}{\vartheta} \right) T. \]

This being bounded by \( S \) is equivalent to \( S \geq 2(\delta + (1 - 1/\vartheta)T) \).

Before we can prove our main theorem, we need to get a hold on \( \delta \). This is a straightforward calculation.

**Lemma 5.9.** Suppose round \( r \) is executed correctly and \( v \in V_g \) receives the message from \( w \in V_g \) for this round at time \( t \). Then setting

\[ \Delta^w_v := L_v(t) - (r - 1)T - (\vartheta^2 + 1)S - \vartheta d \]

yields an estimate satisfying \( \delta \leq u + (\vartheta - 1)d + 2(\vartheta^2 - \vartheta)S \).

**Proof.** Denote by \( t \) the time when \( v \) receives the message from \( w \) and by \( t_s \) the time when it was sent. We have that

\[ L_v(t) - L_w(t_s) \in (L_v(t_s) - L_w(t_s)) + d - u, L_v(t_s) - L_w(t_s) + \vartheta d. \]

Moreover,

\[ |L_v(t_s) - L_w(t_s) - (L_v(t_s) - L_w(p_r))| \leq (\vartheta - 1)(t_s - p_r) \leq (\vartheta^2 - \vartheta)S. \]

We conclude that

\[ L_v(t) - L_w(t_s) \in (L_v(p_r) - L_w(p_r)) + d - u - (\vartheta^2 - \vartheta)S, L_v(p_r) - L_w(p_r) + \vartheta d + (\vartheta^2 - \vartheta)S. \]

As \( L_w(t_s) = (r - 1)T + (\vartheta + 1)S \) by the design of the algorithm, the claim of the lemma follows.

**Theorem 5.10.** Assume that \( 3 + 4\vartheta - 4\vartheta^2 - 2\vartheta^3 > 0 \) and that estimates are computed according to Lemma 5.9. For any choice of

\[ T \geq \frac{6\vartheta^4(u + d)}{3 + 4\vartheta - 4\vartheta^2 - 2\vartheta^3} \in O(d), \]

set

\[ S := \frac{2(u + (\vartheta - 1)d + (1 - 1/\vartheta)T)}{1 + 4\vartheta - 4\vartheta^2} \in O\left( u + \left( 1 - \frac{1}{\vartheta} \right) T \right). \]

If \( \max_{v \in V} \{H_v(0)\} \leq S \), then Algorithm 5 solves pulse synchronization with skew \( S \), \( P_{\text{min}} \geq T/\vartheta - S \), and \( P_{\text{max}} \leq T + 2S \).
5.2. A VARIANT OF THE LYNCH-WELCH ALGORITHM

Proof. Set \( \delta := u + (\vartheta - 1)d + 2(\vartheta^2 - \vartheta)S \) in accordance with Lemma 5.9. Thus,

\[
S = 2 \left( u + (\vartheta - 1)d + 2(\vartheta^2 - \vartheta)S + \left( 1 - \frac{1}{\vartheta} \right) T \right) = 2 \left( \delta + \left( 1 - \frac{1}{\vartheta} \right) T \right).
\]

Moreover,

\[
T \geq \frac{6\vartheta^4(u + d) + 2(\vartheta^3 - 1)T}{3 + 4\vartheta - 4\vartheta^2 - 2\vartheta^3 + 2(\vartheta^3 - 1)} > \frac{6\vartheta^3(u + (\vartheta - 1)d) + 2(\vartheta^3 - 1)T}{1 + 4\vartheta - 4\vartheta^2} + \vartheta^2d,
\]

i.e.,

\[
\frac{T}{\vartheta} > (\vartheta^2 + \vartheta + 1) \cdot \frac{2(u + (\vartheta - 1)d) + 2(1 - 1/\vartheta)T}{1 + 4\vartheta - 4\vartheta^2} + \vartheta d = (\vartheta^2 + \vartheta + 1)S + \vartheta d.
\]

The claim is now shown by a straightforward induction on the pulse number, where the hypothesis includes that all previous rounds have been executed correctly. The induction is anchored at the first pulse, which satisfies the skew bounds due to the assumed bound on the hardware clock values at time 0. The induction step is performed by invoking Lemma 5.8, where Lemma 5.9 shows that \( \delta \) is indeed a bound on the quality of estimates. We obtain that \( S \) is a bound on the skew for all pulses and that \( T/\vartheta - S \leq p_{r+1} - p_r \leq T + \delta \) for each \( r \in \mathbb{N} \). This implies that \( P_{\min} \geq (T - S)/\vartheta \) and, using that \( \max_{v \in V_g} \{ p_{v,r} \} - \min_{v \in V_g} \{ p_{v,r} \} \leq S \) and \( \delta < S \), that \( P_{\max} \leq T + 2S \). \( \square \)

Remarks:

- The theorem requires that \( 3 + 4\vartheta - 4\vartheta^2 - 2\vartheta^3 > 0 \), which is the case for \( \vartheta \leq 1.09 \). As \( \vartheta \) approaches this threshold, the skew goes to \( \infty \).
- Sending \( (T, \vartheta) \to (\infty, 1) \), the ratio \( P_{\max}/P_{\min} \in (1 + o(1))\vartheta \). However, when sending \( T \to \infty \) while keeping \( \vartheta \) fixed, the ratio converges to a constant \( c \in 1 + \mathcal{O}(\vartheta - 1) \).
- If on initialization such a tight skew bound cannot be guaranteed, one can choose \( T \) accordingly larger.
- Alternatively, one can only initially use the larger \( T \) and keep reducing \( T \) alongside the decrease in (the worst-case bound on) the skew. You’ll analyze this in the exercises.
- A known bound on the initial skew is necessary for executing the algorithm. You’ll show this in the exercises as well.
- We haven’t clarified how nodes compute their estimates of faulty nodes’ clocks. What if these nodes send no or many messages during a round? The answer is simple: It doesn’t matter. As the approximate agreement algorithm works regardless of what values faulty nodes provide, choosing any default value for nodes clearly not obeying the protocol will do.
Bibliographic Notes

Approximate agreement was introduced by Dolev et al. [DLP+86], actually having the goal in mind to use it for clock synchronization. As shown by Fekete [Fek86], the rate of convergence provided by their algorithm is close to being asymptotically optimal, and it is asymptotically optimal if only one round of communication per iteration is performed. He also shows that faster convergence is possible if the (maximum) number of possible faults is smaller.

The clock synchronization protocol by Lynch and Welch [LL84] is able to exploit this, too, to achieve faster convergence and thus slightly smaller skews. The respective modification is straightforward and can also be applied to the variant presented in this lecture, which follows [KL16]. The main difference to [LL84] is, just as for the Srikanth-Toueg algorithm from the previous lecture, that no tick numbers are communicated by the algorithm. One can also adjust clock rates (as opposed to just correcting clock offsets), but this requires the additional assumption that hardware clock rates change slowly [KL16]. The Lynch-Welch algorithm has already found its way into practice: it’s the synchronization mechanism underlying industrial systems used, e.g., in cars and planes [KB03, FAS09].

Bibliography


Appendix A

Notation and Preliminaries

This appendix sums up important notation, definitions, and key lemmas that are not the main focus of the lecture.

A.1 Numbers and Sets

In this lecture, zero is not a natural number: 0 \notin \mathbb{N}; we just write \mathbb{N}_0 := \mathbb{N} \cup \{0\} whenever we need it. \mathbb{Z} denotes the integers, \mathbb{Q} the rational numbers, and \mathbb{R} the real numbers. We use \mathbb{R}^+ = \{x \in \mathbb{R} \mid x > 0\} and \mathbb{R}_0^+ = \{x \in \mathbb{R} \mid x \geq 0\}.

Rounding down \lfloor x \rfloor \in \mathbb{R} is denoted by \lfloor x \rfloor := \max\{z \in \mathbb{Z} \mid z \leq x\} and rounding up by \lceil x \rceil := \min\{z \in \mathbb{Z} \mid z \geq x\}.

For \(n \in \mathbb{N}_0\), we define \(\lfloor n \rfloor := \{0, \ldots, n - 1\}\), and for a set \(M\) and \(k \in \mathbb{N}_0\), \(\binom{M}{k} := \{N \subseteq M \mid |N| = k\}\) is the set of all subsets of \(M\) that contain exactly \(k\) elements.

A.2 Graphs

A finite set of vertices, also referred to as nodes \(V\) together with edges \(E \subseteq \binom{V}{2}\) defines a graph \(G = (V, E)\). Unless specified otherwise, \(G\) has \(n = |V|\) vertices and \(m = |E|\) edges and the graph is simple: Edges \(e = \{v, w\} \subseteq V\) are undirected, there are no loops, and there are no parallel edges.

If \(e = \{v, w\} \in E\), the vertices \(v\) and \(w\) are adjacent, and \(e\) is incident to \(v\) and \(w\), furthermore, \(e' \in E\) is adjacent to \(e\) if \(e \cap e' \neq \emptyset\). The neighborhood of \(v\) is \(N_v := \{w \in V \mid \{v, w\} \in E\}\), i.e., the set of vertices adjacent to \(v\). The degree of \(v\) is \(\delta_v := |N_v|\), the size of \(v\)’s neighborhood. We denote by \(\Delta := \max_{v \in V} \delta_v\) the maximum degree in \(G\).
APPENDIX A. NOTATION AND PRELIMINARIES

A $v_1$-$v_d$-path $p$ is a set of edges $p = \{\{v_1, v_2\}, \{v_2, v_3\}, \ldots, \{v_{d-1}, v_d\}\}$ such that $|\{e \in p \mid v \in e\}| \leq 2$ for all $v \in V$. $p$ has $|p|$ hops, and we call $p$ a cycle if it visits all of its nodes exactly twice. The diameter $D$ of the graph is the minimum integer such that for any $v, w \in V$ there is a $v$-$w$-path of at most $D$ hops (or $D = \infty$ if no such integer exists). We consider connected graphs only, i.e., graphs satisfying $D \neq \infty$.

A.2.1 Trees and Forests

A forest is a cycle-free graph, and a tree is a connected forest. Trees have $n - 1$ edges and a unique path between any pair of vertices. The tree $T = (V, E)$ is rooted if it has a designated root node $r \in V$. A leaf is a node of degree 1. A rooted tree has depth $d$ if the maximum length of a root-leaf path is $d$.

A.3 Asymptotic Notation

We require asymptotic notation to reason about the complexity of algorithms. This section is adapted from Chapter 3 of Cormen et al. [?]. Let $f, g : \mathbb{N}_0 \rightarrow \mathbb{R}$ be functions.

A.3.1 Definitions

$\mathcal{O}(g(n))$ is the set containing all functions $f$ that are bounded from above by $cg(n)$ for some constant $c > 0$ and for all sufficiently large $n$, i.e. $f(n)$ is asymptotically bounded from above by $g(n)$.

$$\mathcal{O}(g(n)) := \{f(n) \mid \exists c \in \mathbb{R}^+, n_0 \in \mathbb{N}_0:\ \forall n \geq n_0: \quad 0 \leq f(n) \leq cg(n)\}$$

The counterpart of $\mathcal{O}(g(n))$ is $\Omega(g(n))$, the set of functions asymptotically bounded from below by $g(n)$, again up to a positive scalar and for sufficiently large $n$:

$$\Omega(g(n)) := \{f(n) \mid \exists c \in \mathbb{R}^+, n_0 \in \mathbb{N}_0:\ \forall n \geq n_0: \quad 0 \leq cg(n) \leq f(n)\}$$

If $f(n)$ is bounded from below by $c_1g(n)$ and from above by $c_2g(n)$ for positive scalars $c_1$ and $c_2$ and sufficiently large $n$, it belongs to the set $\Theta(g(n))$; in this case $g(n)$ is an asymptotically tight bound for $f(n)$. It is easy to check that $\Theta(g(n))$ is the intersection of $\mathcal{O}(g(n))$ and $\Omega(g(n))$.

$$\Theta(g(n)) := \{f(n) \mid \exists c_1, c_2 \in \mathbb{R}^+, n_0 \in \mathbb{N}_0:\ \forall n \geq n_0:\ \quad 0 \leq c_1g(n) \leq f(n) \leq c_2g(n)\}$$

$$f(n) \in \Theta(g(n)) \iff f \in (\mathcal{O}(g(n)) \cap \Omega(g(n)))$$

For example, $n \in \mathcal{O}(n^2)$ but $n \not\in \Omega(n^2)$ and thus $n \not\in \Theta(n^2)$. But $3n^2 - n + 5 \in \mathcal{O}(n^2)$, $3n^2 - n + 5 \in \Omega(n^2)$, and thus $3n^2 - n + 5 \in \Theta(n^2)$ for $c_1 = 1$, $c_2 = 3$, and $n_0 = 4$.

\footnote{We write $f(n) \in \mathcal{O}(g(n))$ unlike some authors who, by abuse of notation, write $f(n) = \mathcal{O}(g(n))$. $f(n) \in \mathcal{O}(g(n))$ emphasizes that we are dealing with sets of functions.}
A.3. ASYMPTOTIC NOTATION

In order to express that an asymptotic bound is not tight, we require \( o(g(n)) \) and \( \omega(g(n)) \). \( f(n) \in o(g(n)) \) means that for any positive constant \( c \), \( f(n) \) is strictly smaller than \( cg(n) \) for sufficiently large \( n \).

\[
o(g(n)) := \{ f(n) \mid \forall c \in \mathbb{R}^+: \exists n_0 \in \mathbb{N}_0: \forall n \geq n_0: 0 \leq f(n) < cg(n) \}
\]

As an example, consider \( \frac{1}{n} \). For arbitrary \( c \in \mathbb{R}^+, \frac{1}{n} < c \) we have that for all \( n \geq \frac{1}{c} + 1 \), so \( \frac{1}{n} \in o(1) \). A similar concept exists for lower bounds that are not asymptotically tight; \( f(n) \in \omega(g(n)) \) if for any positive scalar \( c \), \( cg(n) < f(n) \) as soon as \( n \) is large enough.

\[
\omega(g(n)) := \{ f(n) \mid \forall c \in \mathbb{R}^+: \exists n_0 \in \mathbb{N}_0: \forall n \geq n_0: 0 \leq cg(n) < f(n) \}
\]

A.3.2 Properties

We list some useful properties of asymptotic notation, all taken from Chapter 3 of Cormen et al. [\cite{cormen2009}] . The statements in this subsection hold for all \( f, g, h: \mathbb{N}_0 \to \mathbb{R} \).

Transitivity

\[
\begin{align*}
f(n) &\in \mathcal{O}(g(n)) \land g(n) \in \mathcal{O}(h(n)) \Rightarrow f(n) \in \mathcal{O}(h(n)), \\
f(n) &\in \Omega(g(n)) \land g(n) \in \Omega(h(n)) \Rightarrow f(n) \in \Omega(h(n)), \\
f(n) &\in \Theta(g(n)) \land g(n) \in \Theta(h(n)) \Rightarrow f(n) \in \Theta(h(n)), \\
f(n) &\in o(g(n)) \land g(n) \in o(h(n)) \Rightarrow f(n) \in o(h(n)), \text{ and}
\end{align*}
\]

Reflexivity

\[
\begin{align*}
f(n) &\in \mathcal{O}(f(n)), \\
f(n) &\in \Omega(f(n)), \text{ and}
\end{align*}
\]

Symmetry

\[
\begin{align*}
f(n) &\in \Theta(g(n)) \iff g(n) \in \Theta(f(n)).
\end{align*}
\]

Transpose Symmetry

\[
\begin{align*}
f(n) &\in \mathcal{O}(g(n)) \iff g(n) \in \Omega(f(n)), \text{ and}
\end{align*}
\]

\[
\begin{align*}
f(n) &\in o(g(n)) \iff g(n) \in \omega(f(n)).
\end{align*}
\]
A.4 Bounding the Growth of a Maximum of Differentiable Functions

**Lemma A.1.** For $k \in \mathbb{N}$, let $\mathcal{F} = \{ f_i \mid i \in [k] \}$, where each $f_i : [t_0, t_1] \to \mathbb{R}$ is differentiable, and $[t_0, t_1] \subset \mathbb{R}$. Define $F : [t_0, t_1] \to \mathbb{R}$ by $F(t) := \max_{i \in [k]} \{ f_i(t) \}$. Suppose $\mathcal{F}$ has the property that for every $i$ and $t$, if $f_i(t) = F(t)$, then $\frac{d}{dt} f_i(t) \leq r$. Then for all $t \in [t_0, t_1]$, we have $F(t) \leq F(t_0) + r(t - t_0)$.

**Proof.** We prove the stronger claim that for all $a, b$ satisfying $t_0 \leq a < b \leq t_1$, we have

$$
\frac{F(b) - F(a)}{b - a} \leq r. \quad (A.1)
$$

To this end, suppose to the contrary that there exist $a_0 < b_0$ satisfying $F(b_0) - F(a_0) > r + \varepsilon$ for some $\varepsilon > 0$. We define a sequence of nested intervals $[a_0, b_0] \supset [a_1, b_1] \supset \cdots$ as follows. Given $[a_j, b_j]$, let $c_j = (b_j + a_j)/2$ be the midpoint of $a_j$ and $b_j$. Observe that

$$
F(b_j) - F(a_j) \geq \frac{1}{2} F(b_j) - F(c_j) + \frac{1}{2} F(c_j) - F(a_j) \geq r + \varepsilon,
$$

so that

$$
\frac{F(b_j) - F(c_j)}{b_j - c_j} \geq r + \varepsilon \quad \text{or} \quad \frac{F(c_j) - F(a_j)}{c_j - a_j} \geq r + \varepsilon.
$$

If the first inequality holds, define $a_{j+1} = c_j$, $b_{j+1} = b_j$, and otherwise define $a_{j+1} = a_j$, $b_{j+1} = c_j$. From the construction of the sequence, it is clear that for all $j$ we have

$$
\frac{F(b_j) - F(a_j)}{b_j - a_j} \geq r + \varepsilon. \quad (A.2)
$$

Observe that the sequences $\{a_j\}_{j=0}^\infty$ and $\{b_j\}_{j=0}^\infty$ are both bounded and monotonic, hence convergent. Further, since $b_j - a_j = \frac{1}{2^j} (b_0 - a_0)$, the two sequences share the same limit.

Define

$$
c := \lim_{j \to \infty} a_j = \lim_{j \to \infty} b_j,
$$

and let $f \in \mathcal{F}$ be a function satisfying $f(c) = F(c)$. By the hypothesis of the lemma, we have $f'(c) \leq r$, so that

$$
\lim_{h \to 0} \frac{f(c + h) - f(h)}{h} \leq r.
$$

Therefore, there exists some $h > 0$ such that for all $t \in [c - h, c + h]$, $t \neq c$, we have

$$
\frac{f(t) - f(c)}{t - c} \leq r + \frac{1}{2}\varepsilon.
$$

Further, from the definition of $c$, there exists $N \in \mathbb{N}$ such that for all $j \geq N$, we have $a_j, b_j \in [c - h, c + h]$. In particular this implies that for all sufficiently large $j$, we have

$$
\frac{f(c) - f(a_j)}{c - a_j} \leq r + \frac{1}{2}\varepsilon, \quad (A.3)
$$

$$
\frac{f(b_j) - f(c)}{b_j - c} \leq r + \frac{1}{2}\varepsilon. \quad (A.4)
$$
A.4. BOUNDING THE GROWTH OF A MAXIMUM OF DIFFERENTIABLE FUNCTIONS

Since \( f(a_j) \leq F(a_j) \) and \( f(c) = F(c) \), (A.3) implies that for all \( j \geq N \),

\[
\frac{F(c) - F(a_j)}{c - a_j} \leq r + \frac{1}{2^N}.
\]

However, this expression combined with (A.2) implies that for all \( j \geq N \)

\[
\frac{F(b_j) - F(c)}{b_j - c} \geq r + \varepsilon. \tag{A.5}
\]

Since \( F(c) = f(c) \), the previous expression together with (A.4) implies that for all \( j \geq N \) we have \( f(b_j) < F(b_j) \).

For each \( j \geq N \), let \( g_j \in \mathcal{F} \) be a function such that \( g_j(b_j) = F(b_j) \). Since \( \mathcal{F} \) is finite, there exists some \( g \in \mathcal{F} \) such that \( g = g_j \) for infinitely many values \( j \).

Let \( j_0 < j_1 < \cdots \) be the subsequence such that \( g = g_{j_k} \) for all \( k \in \mathbb{N} \). Then for all \( j_k \), we have \( F(b_{j_k}) = g(b_{j_k}) \). Further, since \( F \) and \( g \) are continuous, we have

\[
g(c) = \lim_{k \to \infty} g(b_{j_k}) = \lim_{k \to \infty} F(b_{j_k}) = F(c) = f(c).
\]

By (A.5), we therefore have that for all \( k \)

\[
\frac{g(b_{j_k}) - g(c)}{b_{j_k} - c} = \frac{F(b_{j_k}) - F(c)}{b_{j_k} - c} \geq r + \varepsilon.
\]

However, this final expression contradicts the assumption that \( g'(c) \leq r \). Therefore, (A.1) holds, as desired. \( \square \)