Lecture 4

Fault-Tolerant Clock Synchronization

In the previous lectures, we assumed that all processors faithfully execute a prescribed algorithm without faults. This assumption is not realistic in large-scale systems, and it is problematic in high reliability systems as well. After all, if the system clock fails, there may be no further computations at all!

In general, it is difficult to predict what kind of faults may occur, so 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. Arbitrary adversarial behavior may be overly pessimistic in the sense that "real" faults might have a hard time producing such behavior. However, if our algorithms *can* handle the worst-case scenarios, we do not have to study what kind of faults may actually happen and verify the resulting fault model(s) for each system we build.

Definition 4.1 (Byzantine Faults)**.** *A* Byzantine *faulty node is a node that may behave arbitrarily. That is, such a node need not follow any algorithm prescribed by the system designer. An algorithm is* robust *to f Byzantine faults it its performance guarantees hold for any exuction in which there are at most f Byzantine faulty nodes.*

Given a graph $G = (V, E)$ we denote the set of non-faulty nodes by V_g. The set of faulty nodes is (initially) unknown to the other nodes. Thus, any algorithm robust to f Byzantine faults must work correctly regardless of which nodes are faulty.

Allowing Byzantine faults forces some limitations on the performance guarantees we can achieve. 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, sychronization is impossible. The intuition is simple: Split the neighborhood of some node v in two sets A and B and consider two executions, $\mathcal{E}_A$ and $\mathcal{E}_B$, such that A is faulty in $\mathcal{E}_A$ and B is faulty in $\mathcal{E}_B$. Given that A is faulty in $\mathcal{E}_A$, B and v need to stay synchronized in $\mathcal{E}_A$, regardless of what the nodes in A do. However, the same applies to $\mathcal{E}_B$ 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.

Later, we will show that the number f of faulty nodes must satisfy $3f < n$ or no sychronization is possible (without further computational assumptions) Motivated by the considerations above, we confine ourselves to G being a complete graph: Each node is connected to each other node, so that each pair of nodes can communicate directly.

4.1 The Pulse Synchronization Problem

We 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 generate synchronized *pulses* whose frequency is bounded from above and below.

Definition 4.2 (Pulse Synchronization)**.** *For each $i \in \mathbb{N}$, every (non-faulty) node $v \in V_g$ generates pulse i exactly once. Let $p_{v,i}$ denote the time when v generates pulse i. We require that there are $\mathcal{S}, P_{\min}, P_{\max} \in \mathbb{R}^+$ satisfying*

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

Remarks:

- The pulse synchronization problem is closely related to clock synchronization, as one can interpret the pulses as the "ticks" of a common clock.
- Ideally, $\mathcal{S}$ 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 $\mathcal{S} \geq u/2$ in the worst case.
- Since $D = 1$, pulse synchronization can be easily achieved in the fault-free setting. For instance, the Max Algorithm would achieve skew $u + (\vartheta - 1)(d + T)$, and pulses could be triggered every $\Theta(\mathcal{G})$ local time.
- The difficulty of pulse synchronization lies in preventing the faulty nodes from dividing the correctly functioning nodes into unsynchronized subsets.

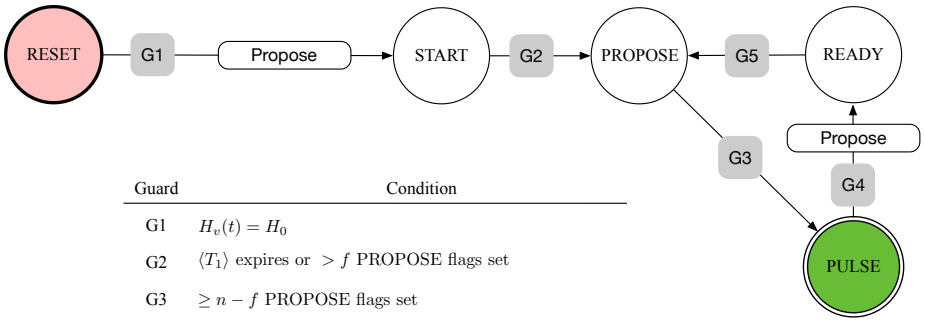
4.2 A Variant of the Srikanth-Toueg Algorithm

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

- Nodes will communicate by broadcast only (i.e., sending the same information to all nodes, including themselves). Note that faulty nodes do not need to obey this rule!

- Messages are very short: The only messages exchanged carry the information that a node transitioned to state PROPOSE, so broadcasting a single bit suffices.
- Each node v stores a *flag* for every node $w \in V$ (including v itself), indicating whether v received such a message from w in the current "round." On some state transitions, v will reset all of its flags to 0, indicating that it has not yet received any propose messages in the current round.
- 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 **true**. The only exception is the starting state RESET, from which nodes transition to 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:



Guard	Condition
G1	$H_v(t) = H_0$
G2	$\langle T_1 \rangle$ expires or $> f$ PROPOSE flags set
G3	$\geq n - f$ PROPOSE flags set
G4	$\langle T_2 \rangle$ expires
G5	$\langle T_3 \rangle$ expires or $> f$ PROPOSE flags set

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 PROPOSE. The notation $\langle T \rangle$ evaluates to **true** when T time units have passed on the local clock since the transition to the current state. The boxes labeled PROPOSE indicate that a node clears its PROPOSE memory flags when transitioning from RESET to START or from PULSE to 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 RESET, which they leave at the time t when $H_v(t) = H_0$. Whenever a node transitions to state PULSE, it generates a pulse. The constraints imposed on the timeouts are listed in Inequalities (4.1)–(4.4).

- At the beginning of an iteration, all nodes transition to state READY (or, initially, START) within a bounded time span. This resets the flags.
- Nodes wait in this state until they are sure that all correct nodes reached it. Then, when a local timeout expires, they transition to PROPOSE.
- When it looks like all correct nodes (may) have arrived there, they transition to PULSE. As the faulty nodes may never send a message, this means to wait for $n - f$ nodes having announced to be in PROPOSE.
- However, faulty nodes may also sent PROPOSE messages, meaning that the threshold is reached despite some nodes still waiting in READY for their timeouts to expire. To "pull" such stragglers along, nodes will also transition to PROPOSE if more than f of their memory flags are set. This is proof that at least one correct node transitioned to 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 PULSE sufficiently long to ensure that no PROPOSE messages are in transit any more before transitioning to READY and starting the next iteration.

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

$$H_0 > \max_{v \in V_g} \{H_v(0)\} \tag{4.1}$$

$$\frac{T_1}{\vartheta} \geq H_0 \tag{4.2}$$

$$\frac{T_2}{\vartheta} \geq 3d \tag{4.3}$$

$$\frac{T_3}{\vartheta} \geq \left(1 - \frac{1}{\vartheta}\right) T_2 + 2d \tag{4.4}$$

Lemma 4.3. *Suppose $3f < n$, $\Delta \geq 0$, and the above constraints are satisfied. Moreover, assume that each $v \in V_g$ transitions to* START *(*READY*) at a time $t_v \in [t - \Delta, t]$, no such node transitions to* PROPOSE *during $(t - \Delta - d, t_v)$, and $T_1 \geq \vartheta\Delta$ ($T_3 \geq \vartheta\Delta$). Then there is a time $t' \in (t - \Delta + T_1/\vartheta, t + T_1 - d)$ ($t' \in (t - \Delta + T_3/\vartheta, t + T_3 - d)$) such that each $v \in V_g$ transitions to* PULSE *during $[t', t' + 2d)$.*

Proof. We perform the proof for the case of START and T_1; the other case is analogous. Let t_p denote the smallest time larger than $t - \Delta - d$ when some $v \in V_g$ transitions to PROPOSE (such a time exists, as T_1 will expire if a node does not transition to PROPOSE before this happens). By assumption and the definition of t_p, no $v \in V_g$ transitions to 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 READY at time $t_v \geq t - \Delta$, this implies that the node(s) from V_g that transition to PROPOSE at time t_p do so

because T_1 expired. As hardware clocks run at most at rate ϑ 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 READY and will not reset its memory flags again without transitioning to PULSE first. Therefore, each $v \in V_g$ will transition to PULSE: Each $v \in V_g$ transitions to 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 PULSE.

It remains to show that all correct nodes transition to PULSE within $2d$ time. Let t' be the minimum time after t_p when some $v \in V_g$ transitions to 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 **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 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 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 PULSE during $[t', t' + 2d)$. □

Theorem 4.4. *Suppose $3f < n$ and the constraints of Equation (1.1–4) are satisfied. Then the algorithm given in Figure 4.1 solves the pulse synchronization problem with $\mathcal{S} = 2d$, $P_{\min} = (T_2 + T_3)/\vartheta - 2d$ and $P_{\max} = T_2 + T_3 + 3d$.*

Proof. 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 $[0, H_0)$ by (4.1). As hardware clocks run at least at rate 1, thus all nodes transition to state START by time H_0. By (4.2), the lemma can be applied with $t = \Delta = H_0$, yielding times $p_{v,1}$, $v \in V_g$, 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 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,

$$\begin{aligned} p_{v,i+1} &\in \left(t - \Delta + \frac{T_3}{\vartheta}, t + T_3 + d\right) \\ &= \left(\max_{w \in V_g}\{p_{w,i}\} + \frac{T_2 + T_3}{\vartheta} - 2d, \max_{w \in V_g}\{p_{w,i}\} + T_2 + T_3 + d\right). \end{aligned}$$

Thus, the first bound is satisfied. The second follows as well, as we have already shown that $\max_{w \in V_g}\{p_{w,i}\} \leq \min_{w \in V_g}\{p_{w,i}\} + 2d$. □

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 $P_{\max}/P_{\min}$ can be brought arbitrarily close to ϑ.
- 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 $P_{\min} = 3\vartheta d$ and $P_{\max} = (3\vartheta^2 + 2\vartheta + 2)d$.

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 $\mathcal{S}$ is violated) or the algorithm cannot guarantee bounds on the period. In each execution, one of the sets, say A, consists entirely of faulty nodes. All of the nodes in the (correct) set B will have identical hardward clocks, as will the nodes in C. The faulty nodes in A attempt to fool the correct nodes in B and C as follows: to one set, say B, faulty nodes send messages to each $vinB$ that lead v to believe that v's clock is fast. Similarly, nodes in A try to convince each $w \in C$ that w's clock is slow. All clock rates (actual or simulated) will lie between 1 and ρ^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.

For each pair of consecutive executions, the executions are indistinguishable to the set that is correct in both executions *and* there is a factor of $\rho > 1$ between the speeds of hardware clocks. This means that the pulses are generated at a by factor ρ higher speed. However, as the skew bounds are to be satisfied, the set of correct nodes that *know* that something is different will have to generate pulses faster. Thus, in execution $\mathcal{E}_i$, pulses are generated at an amortized rate of (at least) $\rho^i P_{\min}$. For $i > \log_\rho P_{\max}/P_{\min}$, we arrive at a contradiction.

Lemma 4.5. *Suppose $n \leq 3f$. Then, for any algorithm $\mathcal{A}$, 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 ρ^{-i}. All correct nodes run $\mathcal{A}$, 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 ρ, 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 $t = H_v(t)/\rho$, it sends the same message in $\mathcal{E}_i$ at time ρt. Thus, it is received at local time

$$H_w^{(\mathcal{E}_i)}(\rho t) + d = \rho^2 t + d = H_w^{\mathcal{E}_{i+1}}(t) + d\,.$$

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, it sends the same message in $\mathcal{E}_i$ at

	$H_A(t)$	$H_B(t)$	$H_C(t)$
$\mathcal{E}_0$	ρt	$\rho^2 t$	$\leftarrow$ arbitrary $t \rightarrow$
$\mathcal{E}_1$	$\rho^2 t$	$\leftarrow \rho^3 t$ $t \rightarrow$	ρt
$\mathcal{E}_2$	$\leftarrow \rho^3 t$ $\rightarrow t$	ρt	$\rho^2 t$
$\mathcal{E}_3$	ρt	$\rho^2 t$	$\leftarrow \rho^3 t$ $t \rightarrow$
$\mathcal{E}_4$	$\rho^2 t$	$\leftarrow \rho^3 t$ $t \rightarrow$	ρt
$\mathcal{E}_5$	$\leftarrow \rho^3 t$ $\rightarrow t$	ρt	$\rho^2 t$
$\mathcal{E}_6$	ρt	$\rho^2 t$	$\leftarrow \rho^3 t$ $t \rightarrow$
$\dots$	$\dots$	$\dots$	$\dots$

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.

time t/ρ. Thus, it is received at local time

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

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 $\mathcal{E}_i$ at speed ρ^{-i}." Think of each faulty node as simulating two virtual nodes, one for messages sent "to the left," which has local time $\rho^3 t$ 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 $\mathcal{E}_i$ and the corresponding nodes in executions $\mathcal{E}_{i-1}$ and $\mathcal{E}_{i+1}$, 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 $\mathcal{E}_i$, the respective virtual node sends the message at the same local time as v sends the message in execution $\mathcal{E}_{i-1}$ (left) or $\mathcal{E}_{i+1}$ (right). In terms of real time, there is exactly a factor of ρ: if v is faulty in $\mathcal{E}_i$ and wants to determine the behavior of its virtual node corresponding to $\mathcal{E}_{i-1}$ up to time t, it needs to simulate $\mathcal{E}_{i-1}$ up to time ρt; similarly, when doing the same for its virtual node corresponding to $\mathcal{E}_{i+1}$, it needs to simulate $\mathcal{E}_{i+1}$ up to time t/ρ. Thus, when simulating all executions concurrently, where $\mathcal{E}_i$ progresses at rate ρ^{-i}, at all times the behavior of faulty nodes according to the above scheme can be determined. This completes the proof. □

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

Proof. Assume for contradiction that there is an algorithm solving pulse synchronization. We apply Lemma 4.5, yielding a sequence of executions $\mathcal{E}_i$ 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 $v \in V_g^{(\mathcal{E}_i)}$ satisfying that

$$p_{v,j}^{(\mathcal{E}_i)} - p_{v,1}^{(\mathcal{E}_i)} \leq (j-1)\rho^{-i} P_{\max} + 2i\mathcal{S}$$

for all $j \in \mathbb{N}$, 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 $v \in V_g^{(\mathcal{E}_i)}$ be a node with $p_{v,j}^{(\mathcal{E}_i)} - p_{v,1}^{(\mathcal{E}_i)} \leq (j-1)\rho^{-i} P_{\max} + 2i\mathcal{S}$ for all $j \in \mathbb{N}_0$. Let $w \in V_g^{(\mathcal{E}_i)} \cap V_g^{(\mathcal{E}_{i+1})}$ be a node that is correct in both $\mathcal{E}_i$ and $\mathcal{E}_{i+1}$. By the skew bound,

$$p_{w,j}^{(\mathcal{E}_i)} - p_{w,1}^{(\mathcal{E}_i)} \leq p_{v,j}^{(\mathcal{E}_i)} - p_{v,1}^{(\mathcal{E}_i)} + 2\mathcal{S} \leq (j-1)\rho^{-i} P_{\max} + 2(i+1)\mathcal{S}$$

for all $j \in \mathbb{N}$. By Lemma 4.5, w cannot distinguish between $\mathcal{E}_i$ and $\mathcal{E}_{i+1}$. Because $H_w^{(\mathcal{E}_{i+1})}(t/\rho) = \rho t = H_w^{(\mathcal{E}_{i+1})}(t)$, we conclude that $p_{w,j}^{(\mathcal{E}_{i+1})} = \rho^{-1} p_{w,j}^{(\mathcal{E}_i)}$ for

all $j \in \mathbb{N}$. Hence,

$$p_{w,j}^{(\mathcal{E}_{i+1})} - p_{w,1}^{(\mathcal{E}_{i+1})} \leq \rho^{-1}\left(p_{w,j}^{(\mathcal{E}_i)} - p_{w,1}^{(\mathcal{E}_i)}\right) \leq (j-1)\rho^{-(i+1)}P_{\max} + 2(i+1)\mathcal{S}$$

for all $j \in \mathbb{N}$, completing the induction step.

Now choose $i \in \mathbb{N}$ large enough so that $\rho^{-i}P_{\max} < P_{\min}$ and let $v \in V_g^{(\mathcal{E}_i)}$ be a node to which the claim applies in $\mathcal{E}_i$. Choosing $j - 1 > 2i\mathcal{S}(P_{\min} - \rho^{-i}P_{\max})$, it follows that

$$p_{v,j}^{(\mathcal{E}_i)} - p_{v,1}^{(\mathcal{E}_i)} \leq (j-1)\rho^{-i}P_{\max} + 2i\mathcal{S} < (j-1)P_{\min}\,.$$

Hence, the minimum period bound is violated, as there must be some index $j' \in \{1, \dots, j-1\}$ for which $p_{v,j'+1}^{(\mathcal{E}_i)} - p_{v,j'}^{(\mathcal{E}_i)} < P_{\min}$. □

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

[DFL+15] Danny Dolev, Matthias Függer, Christoph Lenzen, Ulrich Schmid, and Andreas Steininger. Fault-tolerant Distributed Systems in Hardware. *Bulletin of the EATCS*, 116, 2015.

[DHS86] Danny Dolev, Joseph Y. Halpern, and H.Raymond Strong. On the Possibility and Impossibility of Achieving Clock Synchronization. *Journal of Computer and System Sciences*, 32(2):230–250, 1986.

[FS12] Matthias Függer and Ulrich Schmid. Reconciling fault-tolerant distributed computing and systems-on-chip. *Distributed Computing*, 24(6):323–355, 2012.

[PSL80] M. Pease, R. Shostak, and L. Lamport. Reaching Agreement in the Presence of Faults. *J. ACM*, 27(2):228–234, 1980.

[ST87] T. K. Srikanth and Sam Toueg. Optimal Clock Synchronization. *J. ACM*, 34(3):626–645, 1987.