Lecture 9

Self-Stabilization

So far we have considered permanently damaged (Byzantine) nodes. What if faults are transient? There are plenty of causes for such transient faults: radiation, power fluctuations, etc. One way of dealing with transient faults is to just consider the nodes undergoing faults becoming Byzantine, but this may be too pessimistic: after the transient faults cease, they can recover a correct state and be good citizens again. Also, we may be able to recover from $n/3$ or more nodes undergoing transient faults. In fact, we want that the system recovers even if *all* nodes fail and $f < n/3$ of them *remain* faulty!

But what does "recover" from transient faults mean? We need to capture this in a way enabling us to prove (or disprove) this property for a given algorithm.

Definition 9.1 (Self-Stabilization)**.** *Given a system, denote by S its state space, i.e., the possible values that transient memory of nodes, message buffers, and any other state-holding device in the system can hold. An* execution trace *is a path in S obeying the restrictions the system model imposes on how the state may evolve over time. A* good trace *is one satisfying desired properties (depending on the task at hand). An algorithm is self-stabilizing, if it guarantees that any trace has a good suffix, i.e., any trace satisfies that there is a time such that its subtrace starting at this time is good. If this time until this subtrace starts is bounded, the* stabilization time *is a (possibly parametrized) worst-case upper bound on this time difference.*

Example 9.2 (Approximate Agreement (Flawed))**.** *Consider a synchronous system in which the nodes perform approximate agreement (Definition 5.2) using some algorithm $\mathcal{A}$. The state of the system is described by the subset of correct nodes $V_g \subseteq V$, their current values $r_v \in \mathbb{R}$, and whether they terminated, i.e., $S = \bigcup_{g=n-f}^{n} (\{0,1\} \times \mathbb{R})^g$ (plus any additional state the algorithm may maintain). A trace is an execution of the algorithm starting from any round i and state, which is completely defined by the messages faulty nodes send in rounds $i, i+1, \ldots$ A good trace is one in which all correct nodes eventually have terminated with values within ε of each other.*

Unfortunately, no matter what $\mathcal{A}$ we choose, it will not be self-stabilizing: We choose as initial state one where all correct nodes are terminated, but their values are not within ε of each other. No correct node will change state anymore, so there is no good subtrace.

Example 9.3 (Approximate Agreement (Fixed))**.** *Consider a synchronous system in which the nodes keep executing approximate agreement steps (i.e., perform Algorithm 5.1) in each round. The state of the system is fully described by the subset of correct nodes $V_g \subseteq V$ and their current values $r_v \in \mathbb{R}$, i.e., $S = \bigcup_{g=n-f}^{n} \mathbb{R}^g$. A trace is an execution of the algorithm starting from any round i and state, which is completely defined by the messages faulty nodes send in rounds $i, i+1, \ldots$ We decide that a good trace satisfies that the diameter of all state vectors is smaller than ε and, in each round, values are within the range spanned by the correct nodes' values in the previous round.*

From Lemmas 5.4 and 5.5, we get that this holds for any round $j \geq i + \log(\|\vec{r}_i\|/\varepsilon)$, i.e., the algorithm is self-stabilizing. Note, however, that there is no bound on the stabilization time, unless we restrict the maximum initial distance between values, i.e., we bound $\|\vec{r}\|_1$.

Example 9.4 (GCS)**.** *Consider the task of gradient clock synchronization. The state space at time t is given by the nodes' hardware and logical clock values, their estimates of neighbors' clocks, the content of messages that are in transit and their sending times $t - d < t_s \leq t$, and any other state a node may hold according to the algorithm (none in case of our GCS algorithm — at least in the abstract model we considered!). A trace starting at time t is given by an* arbitrary *such state (even if it can't be reached in an execution faithful to the model!), from which we run the system in accordance with the model. A good trace starting at time t' is a trace satisfying a bound $\mathcal{L}$ on the local skew at all times $t'' \geq t'$. A self-stabilizing algorithm now guarantees that for any trace starting at time t, there is some time $t' \geq t$ so that the subtrace starting at time t' satisfies the local skew bound.*

In this setting, the stabilization time is the maximum difference $t' - t$ over all traces (possibly parametrized by, e.g., the number of nodes n, etc.).

Remarks:

- As the examples illustrate, the definition is quite flexible and can be applied to discrete and continuous systems, as well as those with and without permanent faults.
- "Time" is not clearly defined, as it depends on the system what this means. For example, in synchronous systems, time progresses discre(e)tely in rounds, while in GCS we have a continuous reference time.
- Even in the fixed approximate agreement example, the stabilization time is unbounded (i.e., ∞): the bound from Lemma 5.5 is tight in the worst case, and transient faults could bring the stored values arbitrarily far apart. We will show a stabilization time of $\mathcal{O}(\mathcal{G}/\mu)$ for the GCS algorithm. This is good in case there is a self-stabilizing mechanism ensuring a small global skew without interfering with the GCS algorithm (after stabilizing itself). If not, this is no better than the situation for approximate agreement: transient faults may bring the logical clocks arbitrarily far apart.
- It is important to carefully contemplate what "recovering correct operation" after transient faults actually means. This strongly affects whether a solution is possible and how efficient it can be. For instance, the approximate agreement example begs the question whether after transient

faults, the (now arbitrarily corrupted) values nodes store hold any relevant information.

- The first example was bad because we asked for nodes to terminate. As we show below, self-stabilizing algorithms must never terminate, simply because a transient fault then could result in wrong output.
- The algorithm's code and the model assumptions are untouchable to transient faults. In the former case, that's obviously necessary: If transient faults can corrupt the algorithm itself, the algorithm designer has no chance to ensure recovery. It is thus advisable to hardwire the algorithm and/or store code in non-volatile memory. The model assumptions should be examined carefully, however. One will actually have to implement all this reliably, or the system might end up experiencing "transient" faults in perpetuity!
- For instance, this is relevant to the synchronous model. If the synchronous abstraction is implemented using an unreliable clocking method, a single "transient" fault may permanently disrupt the clocking scheme. Yes, the synchronous self-stabilizing algorithm will recover right after the clocking scheme — but the clocking scheme will never do so.

Lemma 9.5. *A self-stabilizing algorithm can never terminate, unless for each node there is a single output that is always correct.*

Proof. Suppose there are two possible conflicting outputs for a node v. More precisely, there is a terminal state (of the system as a whole) in which some possible differing terminal state of v is incorrect. We simply set the system to this state, but v to the incorrect one. As all nodes are in a terminal state, no further changes of node states is possible, implying that this combination of terminal states is preserved forever. Thus, the trace has no good suffix, showing that the algorithm is not self-stabilizing. □

Corollary 9.6. *Suppose $\sigma = \mu/(\vartheta-1) \in 1+\Omega(1)$. Then Algorithm 3.1 stabilizes in $\mathcal{O}(\mathcal{G}/\mu)$ time.*

Proof. W.l.o.g., consider traces that start at time 0. We claim that, for all $s \in \mathbb{N}$ and times $t \geq T_s := \sum_{s'=1}^{s} \mathcal{G}/(\mu\sigma^{s-1})$,

$$\Psi^s(t) \leq \frac{\mathcal{G}}{\sigma^s}\,.$$

The statement of the corollary then follows as in Theorem 3.7 for any time

$$t \geq \sup_{s\in\mathbb{N}}\{T_s\} = \sum_{s'=1}^{\infty} \frac{\mathcal{G}}{\mu\sigma^{s'-1}} = \frac{\sigma\mathcal{G}}{\mu(\sigma-1)} \in \mathcal{O}\left(\frac{\mathcal{G}}{\mu}\right),$$

where the last step uses the assumption that $\sigma \in 1+\Omega(1)$.

The claim is shown as for Theorem 3.14, with the modification that the time of violation now must be at least T_s, where s is the minimal level on which the bound is violated. This is only relevant in a single step of the proof, when invoking Lemma 3.13: here, the proof exploits that $\Psi^{s-1}(t_0) \leq t_1 - t_0 = \mathcal{G}/(\mu(\sigma^{s-1}))$. As $T_s - T_{s-1} = \mathcal{G}/(\mu(\sigma^{s-1}))$, $t_1 \geq T_s$ implies that $t_0 \geq T_{s-1}$, i.e., this condition is satisfied. □

Remarks:

- You might think "this was almost too easy." The response to this has two parts. The first is that the algorithmic approach just happens to be that the algorithm continuously struggles on each level to distribute the skew in a way keeping Ψ^s small. The property of being self-stabilizing then emerges naturally. The second is that the model is hiding a lot of things. In order to make the algorithm self-stabilizing, one needs self-stabilizing solutions for maintaining a small global skew, computing estimates, and providing all the other convenient abstractions the model assumes.

- Fortunately, the algorithm really *is* doing a great job (which is rather coincidental, given that it was not designed with the goal of self-stabilization in mind). You'll show in the exercises that the necessary adaptions are minimal.

9.1 Making Lynch-Welch Self-Stabilizing

Why the Lynch-Welch algorithm again? Well, it achieves asymptotically optimal skew, tolerates the maximum possible number of $\lceil n/3 \rceil - 1$ Byzantine faults, and it's simple to implement. As we showed in the previous lectures, we can even handle metastability, which is a concern if we perform the iterations so quickly that it matters. Combining all of this with self-stabilization would result in an extremely robust algorithm!

Alas, we won't get self-stabilization "for free" as with the GCS algorithm. The Lynch-Welch algorithm relies on some initial degree of synchronization to maintain the abstraction of rounds it uses. It is simulating synchronous execution, but self-stabilization requires that we can deal with a complete (initial) lack of synchrony! It turns out that this is an incredibly hard problem, and we will only take a first step today. This step is reducing the task to finding an (efficient) self-stabilizing solution to pulse synchronization with a much weaker bound on the skew.

Good traces are easily defined: There should be a time t from which on the algorithm behaves just like expected, i.e., as if it was initialized at this time and thus exhibits the skew and period bounds from Theorem 5.10.

9.2 First Attempt: Reset on Heartbeats

In the following, we assume that we already have a self-stabilizing pulse synchronization algorithm with skew σ_h in place. Thus, there is some time t when it stabilized from which on it generates pulses $h_{v,i}$, $v \in V_g$, $i \in \mathbb{N}$, satisfying that $\max_{i \in \mathbb{N}, v, w \in V_g} \{|h_{v,i} - h_{w,i}|\} \leq \sigma_h$. Moreover, we have lower and upper bounds on the time between pulses. We will refer to these pulses as *heartbeats,* or simply beats. They are supposed to be fairly slow in comparison to the pulses of the (modified) Lynch-Welch algorithm; from now on, when we talk of pulses, these will be those of the Lynch-Welch algorithm only.

There's a single hurdle keeping the LW algorithm from being self-stabilizing: the need for a (known) bound on the initial deviation between the nodes' local times. The heartbeats provide exactly that — they are at most σ_h apart from

each other. So we could simply reset the LW algorithm on every heartbeat, setting $\mathcal{S} := \sigma_h$ for the initialization of the algorithm. That's going to work splendidly, as we won't even have to change the analysis — until the next beat comes along and messes things up. As the beats are not as well-synchronized as the LW pulses (otherwise we wouldn't go through this trouble), the reset will destroy the better synchronization guarantee again. Even worse, it may interrupt the LW algorithm generating a pulse!

Remarks:

- Actually, one needs to be slightly more careful when resetting, in that any messages sent by a node just before it is reset by its beat should not be confused with his "round 1"-message following the reset. This is easily addressed by offsetting the first round by ϑd local time compared to $h_{v,i}$, or by using the last message received during the time window in which receivers listen for messages from other nodes.

- It's important not to overlook such "details" when designing self-stabilizing algorithms. Another example is to make sure that variables that are stored in a way admitting infeasible values need to be regularly tested for having a valid value and reset to some default if they don't.

- It's also important to not lose sight of the big picture due to such details, though. A good way of designing self-stabilizing algorithms is to eliminate obstacles one by one, starting with establishing very basic properties and increasing the amount of "control" one has over the system state step by step. The more restricted the state beomes, the easier it typically becomes to reason about it and establish more complicated constraints.

- One could see what we're doing now as doing this process in reverse: We want to solve self-stabilizing pulse synchronization with asymptotically optimal skew, and reduce this task to solving self-stabilizing pulse synchronization with (fairly) large skew.

9.3 Second Attempt: Adding Feedback

The naive solution does not work, because heartbeats may arrive at inconvenient times. However, the "first" beat (in our analysis) establishes a timing relation between the LW instance and the instance of the self-stabilizing pulse synchronization algorithm generating the beats. If we add the additional requirement that the pulse synchronization algorithm accepts some external input that can shift the time when the next beat occurs (within certain bounds), we could align them with the pulses generated by the LW instance.

More specifically, after a (suitably chosen) fixed number of LW pulses, nodes will issue a NEXT signal to the part of them running the algorithm generating the heartbeats. Thus, the beat generation mechanism needs only be "responsive" to the NEXT signal within a specific time window in relation to the previous beat. Under some mild conditions on ϑ, this will turn out to be a fairly harmless constraint. We use this to trigger the next beat, aligned up to $\mathcal{O}(\sigma_h + \mathcal{S})$ time with when the nodes issue the NEXT signals (where $\mathcal{S}$ is the skew of the LW algorithm). This can be kept within a single round of the

LW algorithm (without affecting more than constants), as both $\sigma_h \in \mathcal{O}(d)$ and $\mathcal{S} \in \mathcal{O}(d)$, and the round duration of the LW algorithm $T \in \Omega(d)$ anyway.

Is this good enough? Not yet, as resets may cause large skew every time—unless, in addition, we require that well-synchronized NEXT signals result in an equally well-synchronized heartbeat. Instead of adding even more constraints on the self-stabilizing algorithm (not knowing whether they can be satisfied), we use a different solution.

9.4 Third Attempt: Reset on Unexpected Heartbeats Only

The final adjustment is to *not* perform a reset when a beat arrives on schedule, i.e., within a time window of size $\mathcal{O}(\sigma_h + \mathcal{S})$ around the point when it would occur in a world of perfect synchrony. The size of this window is chosen such that once the heartbeat generation has stabilized, after the first "proper" heartbeat it never happens again that a node is reset. Yet, the reset mechanism still guarantees that a heartbeat will enforce synchronization up to a skew of $\mathcal{O}(\sigma_h + \mathcal{S})$: either a node is not reset (defining an $\mathcal{O}(\sigma_h + \mathcal{S})$-sized window of possible local times) or it is (forcing the local time into the window).

It remains to formalize this approach and prove it correct. W.l.o.g., we assume in the following that the heartbeats stabilized by time 0, and start to reason from there. (Note, however, that this means that arbitrary messages may be in transit at time 0!). Let us first specify our expecations on the feedback mechanism.

Definition 9.7 (Feedback Mechanism)**.** *Nodes $v \in V_g$ generate* beats *at times $h_{v,i} \in \mathbb{R}$, $i \in \mathbb{N}$, such that for parameters $0 < B_1 < B_2 < B_3 \in \mathbb{R}$ the following properties hold for all $i \in \mathbb{N}$.*

1. *For all $v, w \in V_g$, we have that $|h_{v,i} - h_{w,i}| \leq \sigma_h$.*
2. *If no $v \in V_g$ triggers its NEXT signal during $[\min_{w \in V_g}\{h_{w,i}\} + B_1, t]$ for some $t < \min_{w \in C}\{h_{w,i}\} + B_3$, then $\min_{w \in V_g}\{h_{w,i+1}\} > t$.*
3. *If all $v \in V_g$ trigger their NEXT signals during $[\min_{w \in V_g}\{h_{w,i}\} + B_2, t]$ for some $t \leq \min_{w \in V_g}\{h_{w,i}\} + B_3$, then $\max_{w \in V_g}\{h_{w,i+1}\} \leq t + \sigma_h$.*

B_1, B_2, and B_3 cannot be chosen arbitrarily for our approach to work. We will determine sufficient constraints from the analysis.

In order to describe the algorithm, we assume that each node is running an instance of Algorithm 5, the beat generation algorithm, and some additional high-level control we give now. The high-level control may (re-)initialize the instance of Algorithm 5, which is described in the subroutine `reset`(τ) it may call. It has a few parameters:

M: The pulses of Algorithm 5 are counted modulo M. Every M pulses, a heartbeat is expected.

R^-: If a beat arrives at time t and the pulse number is 0 mod M, it should take at least R^- local time before the node generates the next pulse. Instead of trying to compute upfront when Algorithm 5 would generate a pulse,

we simply "catch" the event and perform a reset if the pulse would be generated too early.

R^+: This is the matching upper bound, i.e., under the same conditions, it should take at most R^+ local time before the node generates the next pulse.

$\mathcal{S}(r)$: This denotes the skew bound guaranteed by Algorithm 5 for pulse $1 < r \in \mathbb{N}$, provided the algorithm is initialized with skew $\mathcal{S}(1)$, i.e., in the code of Algorithm 5, $\mathcal{S}$ is replaced by $\mathcal{S}(1)$. Algorithm 6 needs to make use of $\mathcal{S}(1)$ and $\mathcal{S}(M)$ only.

Algorithm 9.1: Interface algorithm, actions for node $v \in V_g$ in response to a local event at time t. Runs in parallel to local instances of the beat generation algorithm and Algorithm 5.

```
1  // algorithm maintains local variable i ∈ [M]
2  if v generates a pulse at time t then
3  |  i := i + 1 mod M;
4  |  if i = 0 then
5  |  |  wait until local time H_v(t) + ϑS(M);
6  |  |  trigger NEXT signal;
7  if v generates a beat at time t then
8  |  if i ≠ 0 then
9  |  |  // beats should align with every M^th pulse, hence reset
10 |  |  reset(R^+);
11 |  else if Algorithm 5 would require v to generate a pulse before local
   |   time H_v(t) + R^- then
12 |  |  // reset at pulse time t' to avoid early pulse or message
13 |  |  reset(R^+ − (H_v(t') − H_v(t))), where t' is the current time;
14 |  else if next pulse is not generated by local time H_v(t) + R^+ then
15 |  |  // reset to avoid late pulse and start listening for other nodes'
   |  |   pulses on time
16 |  |  reset(0);
17 Function reset(τ)
18 |  halt local instance of Algorithm 5;
19 |  wait for τ local time;
20 |  i := 0;
21 |  L_v(t') := S(1), where t' is the current time;
22 |  generate pulse and restart loop of Algorithm 5 (in round r = 1);
```

Figure 9.1 illustrates how the control algorithm ensures stabilization. In words, Algorithm 6 triggers the NEXT signal $\vartheta\mathcal{S}(M)$ local time after generating a beat (i.e., at the earliest time when certainly all nodes have generated the beat), and checks whether pulse 1 modulo M occurs between R^- and R^+ local time after the beat (which necessitates that the beat occurs after pulse 0 modulo M). If this is not the case, the algorithm generates a pulse and restarts the loop of Algorithm 5 exactly R^+ local time after the beat was generated. Moverover, it ensures that no other pulse is generated between the beat and then.

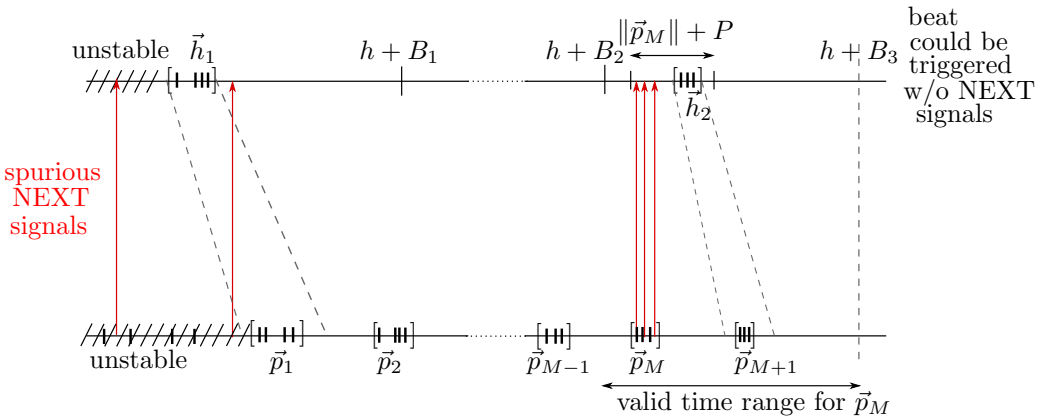

Figure 9.1: Interaction of the beat generation and Algorithm 5 in the stabilization process, controlled by Algorithm 6. Beat $\vec{h}_1$ forces pulse $\vec{p}_1$ to be roughly synchronized. The approximate agreement steps then result in tightly synchronized pulses. By the time the nodes trigger beat $\vec{h}_2$ by providing NEXT signals based on $\vec{p}_M$, synchronization is tight enough to guarantee that the beat results in no resets.

This properly "initializes" Algorithm 5 with skew $\mathcal{S}(1) := R^+ + \sigma_h - R^-/\vartheta$, which then ensures that the skew has been reduced to $\mathcal{S}(M)$ by the time the next beat is due. By choosing all parameters right, we ensure that the M^{th} pulse (after stabilization) falls in the time window provided by Definition 9.7 for making use of the NEXT signals, which then trigger the next beat such that no $v \in V_g$ performs another reset. From there, inductive reasoning shows that no $v \in V_g$ ever performs a reset again (so as long as there are no more transient faults), and the analyis of Algorithm 5 from Chapter 5 yields a bound on the skew achieved. Figure 9.2 illustrates how the nodes locally check whether they should perform a reset or not.

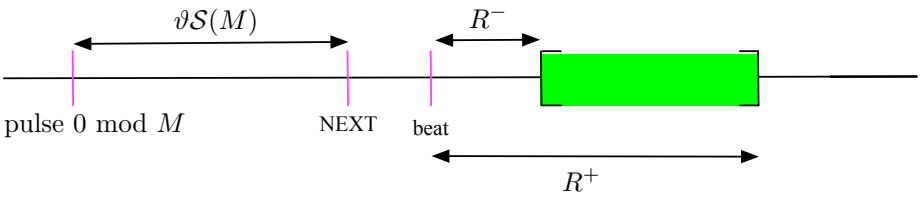

Figure 9.2: After M pulses a node v waits for $\mathcal{S}(M)$ local time and then generates the NEXT signal. After stabilization, the next heartbeat occurs shortly after. If the next pulse (which is going to be generated by the Lynch-Welch algorithm), with number $i = 1$, is not generated at least R^- and at most R^+ local time after the heartbeat (i.e., within the green box), the node resets the Lynch-Welch algorithm, restarting its loop R^+ local time after the beat.

9.5 Analysis

In the following, we assume that in Algorithm 5, $\mathcal{S}$ is replaced by $\mathcal{S}(1)$ in the code, estimates are computed according to Lemma 5.9 (yielding $\delta = u + (\vartheta - 1)d + 2(\vartheta^2 - \vartheta)\mathcal{S}(1)$), and $T \coloneqq \vartheta((\vartheta^2 + \vartheta + 1)\mathcal{S}(1) + \vartheta d)$ (in accordance with Lemma 5.8); as we require that $\mathcal{S}(1) \geq \mathcal{S}(r)$ for all $r \in \mathbb{N}$ (which is implied by (9.8)), this means that T is large enough for all rounds. For the outlined approach to work in addition the following constraints need to be satisfied.

$$\mathcal{S}(1) \geq 2\left(\delta + \left(1 - \frac{1}{\vartheta}\right)T\right) \tag{9.1}$$

$$\frac{R^-}{\vartheta} \geq \sigma_h + \vartheta\mathcal{S}(1) + d \tag{9.2}$$

$$\frac{B_2}{\vartheta} > \sigma_h + R^+ + T + 2\mathcal{S}(1) \tag{9.3}$$

$$B_1 > \sigma_h + R^+ \tag{9.4}$$

$$B_3 > R^+ + (M-1)(T + \mathcal{S}(1)) + (\vartheta + 1)\mathcal{S}(M) + \sigma_h \tag{9.5}$$

$$B_2 \leq \frac{R^-}{\vartheta} + (M-1)\left(\frac{T}{\vartheta} - \mathcal{S}(1)\right) + \mathcal{S}(M) \tag{9.6}$$

$$\frac{R^+}{\vartheta} \geq (\vartheta + 1)\mathcal{S}(M) + \sigma_h \tag{9.7}$$

$$2(\mathcal{S}(1) - \mathcal{S}(M)) \geq \sigma_h \tag{9.8}$$

We will worry about satisfying all of these constraints later. For now, we assume that they hold; what follows is conditional on this assumption.

We first establish that the first beat guarantees to "initialize" the synchronization algorithm such that it will run correctly from this point on (neglecting for the moment the possible intervention by further beats). We use this do define the "first" pulse times $p_{v,1}$, $v \in V_g$, as well; we enumerate consecutive pulses accordingly.

Lemma 9.8. *Let $h \coloneqq \min_{v \in V_g}\{h_{v,1}\}$ and $\mathcal{S}(1) \coloneqq R^+ + \sigma_h - R^-/\vartheta$. We have that*

1. *Each $v \in V_g$ generates a pulse at a unique time $p_{v,1} \in [h + R^-/\vartheta, h + \sigma_h + R^+]$.*
2. *$\|\vec{p}(1)\| \leq \mathcal{S}(1)$.*
3. *At time $p_{v,1}$, $v \in V_g$ sets $i := 1$.*
4. *At the time $\min_{v \in V_g}\{p_{v,1}\}$, no message (of Algorithm 5) sent by node $v \in V_g$ before time $p_{v,1}$ is in transit any more.*

Proof. Assume for the moment that $\min_{v \in V_g}\{h_{v,2}\}$ is sufficiently large, i.e., no second beat will occur at any correct node for the times relevant to the proof of the lemma; we will verify this at the end of the proof.

From the pseudocode given in Algorithm 6, it is straightforward to verify that $v \in V_g$ generates a pulse at a local time from $[H_v(h_{v,1}) + R^-, H_v(h_{v,1}) + R^+]$, and does not generate a pulse at a local time from $[H_v(h_{v,1}), H_v(h_{v,1}) + R^-)$. Denote by $p_{v,1}$ the time when $v \in V_g$ generates its first pulse after time $h_{v,1}$. By

Algorithm 5 and the choice of $\mathcal{S}(1)$, no $v \in V_g$ will send a message or generate another pulse during $(p_{v,1}, p_{v,1} + \mathcal{S}(1))$, where $p_{v,1} \geq h_{v,1} + R^-/\vartheta + \mathcal{S}(1) \geq h + \sigma_h + R^+$. Because $h_{v,1} \in [h, h + \sigma_h]$ for all $v \in V_g$ by Definition 9.7, this implies that for each $v \in V_g$, $p_{v,1} \in [h + R^-/\vartheta, h + \sigma_h + R^+]$ and $p_{v,1}$ is indeed unique. The second claim is now immediate from the choice of $\mathcal{S}(1)$.

Concerning the third claim, observe that if at time $h_{v,1}$ it held that the i-variable of $v \in V_g$ was not 0, it was set to 0. Thus, when v generates its next pulse at time $p_{v,1}$, it is increased to 1. Concerning the final claim, we have established that $v \in V$ generates no pulse during $[h + \sigma_h, h + R^-/\vartheta)$; thus, it sends no message during $[h + \sigma_h + \vartheta\mathcal{S}(1), h + R^-/\vartheta)$ (cf. Algorithm 5), and Inequality (9.2) ensures that no message of $v \in V_g$ sent before time $h_{v,1}$ is in transit any more at time $p_{w,1}$ for any $w \in V_g$.

It remains to show that indeed $\min_{v \in V_g}\{h_{v,2}\}$ is sufficiently large to not interfere with the above reasoning. Clearly, this is the case if round 1 ends at all nodes before this time. Let H be the infimum[1] of times t such that some $v \in V_g$ executes `reset` at a time $t > p_{v,1}$. Clearly, $H \geq \min_{v \in V_g}\{h_{v,2}\}$. By Definition 9.7 and Inequality (9.3), we can conclude that $H \geq h + B_2 \geq h + \sigma_h + R^+ + T + 2\mathcal{S}(1)$. All parts of the statements of this lemma that refer to times smaller than H hold. As $H > h + \sigma_h + R^+$, this implies that Algorithm 5 behaves exactly as if it was initialized with skew $\mathcal{S}(1)$ at time $h + R^-/\vartheta$. We can thus apply all results from Chapter 5 (for times $t < H$) accordingly. In particular, we get the same results as in Theorem 5.10 (as Inequality (9.1) and our choice of T and δ make sure that we can apply all lemmas), yielding that

$$\max_{v \in V_g}\{p_{v,2}\} \leq \min_{v \in V_g}\{p_{v,1}\} + P_{\max} \leq h + \sigma_h + R^+ + T + 2\mathcal{S}(1) < H\,. \quad \square$$

Lemma 9.8 serves as induction anchor for the argument showing that all rounds of the algorithm are executed correctly. Let H be defined as in the previous proof. From the results in Chapter 5, we can bound $\mathcal{S}(r)$ for rounds $r \in \mathbb{N}$ that are complete before time H.

Corollary 9.9. *Suppose for $r \in \mathbb{N}$ that $\max_{v \in V_g}\{p_{v,r}\} < H$. Then*

$$\begin{aligned}\|\vec{p}_r\| &\leq \mathcal{S}(r)\\ &\coloneqq \frac{\mathcal{S}(1)}{2^{r-1}} + \left(2 - \frac{1}{2^{r-2}}\right)\left(\delta + \left(1 - \frac{1}{\vartheta}\right)T\right)\\ &< \frac{\mathcal{S}(1)}{2^{r-1}} + 2(u + (\vartheta^2 - 1)d + (\vartheta - 1)(\vartheta^2 + 3\vartheta + 1)\mathcal{S}(1))\,,\end{aligned}$$

which for sufficiently large $r \in \mathbb{N}$ is in $\mathcal{O}\left(u + (\vartheta - 1)(d + \mathcal{S}(1))\right)$. Moreover, the generated pulses satisfy $P_{\min} \geq T/\vartheta - 2\mathcal{S}(1)$ and $P_{\max} \leq T + 2\mathcal{S}(1)$.

Proof. We inductively apply Lemmas 5.8, and 5.5, yielding

$$\begin{aligned}\|\vec{p}_r\| &\leq \frac{\mathcal{S}(1)}{2^{r-1}} + \sum_{r'=2}^{r} \frac{1}{2^{r-r'}}\left(\delta + \left(1 - \frac{1}{\vartheta}\right)T\right)\\ &= \frac{\mathcal{S}(1)}{2^{r-1}} + \left(2 - \frac{1}{2^{r-2}}\right)\left(\delta + \left(1 - \frac{1}{\vartheta}\right)T\right)\,.\end{aligned}$$

[1] This means "smallest upper bound," which is defined also if there is no minimum. In particular, we will see that $H = \infty$, as no such time exists.

Plugging in δ and our choice of T and bounding $2-2^{-(r-2)} < 2$ yields the stated upper bound on this term. By Inequality (9.8), we have that $\mathcal{S}(1) \geq \mathcal{S}(r)$ for all $r \in \mathbb{N}$. Thus, the inductive use of the lemmas (cf. Statement (iii) of Lemma 5.8) also shows the bounds on the period. □

In other words, all we need to show is that $H = \infty$, i.e., no further resets occur after the first beat. In fact, it suffices to show this for the second beat, as this constitutes the necessary induction step. To this end, we first show that the NEXT signals occur within the "window of opportunity" provided by Definition 9.7.

Lemma 9.10. *For all $v \in V_g$, it holds that $h_{v,2} \in (p_{v,M} + \mathcal{S}(M), p_{v,M} + (\vartheta + 1)\mathcal{S}(M) + \sigma_h]$. In particular, no node calls the* `reset` *subroutine due to its second beat.*

Proof. Checking Algorithm 6 (and noting that by Lemma 9.8 we have that i is set to 1 at time $p_{v,1}$), we see that after time $p_{v,1}$, $v \in V_g$ will not locally trigger a NEXT signal before either time $p_{v,M}+\mathcal{S}(M)$ or H. Denote $p \coloneqq \min_{v\in V_g}\{p_{v,M}\}$. As Lemma 9.8 and Inequality (9.4) show that $\max_{v\in V_g}\{p_{v,1}\} \leq h + \sigma_h + R^+ \leq h+B_1$, no NEXT signal is triggered during $[h+B_1, \min\{p+\mathcal{S}(M), H\}]$. However, by Definition 9.7, in absence of any NEXT signal, $h' \coloneqq \min_{v\in V_g}\{h_{v,2}\}$ satisfies $h' \geq h+B_3$, implying that no NEXT signal is triggered during $[h+B_1, \min\{p+\mathcal{S}(M), h+B_3\}]$. By Definition 9.7, this entails that $H \geq h' \geq \min\{p+\mathcal{S}(M), h+B_3\}$, where equality can hold only if $h' = h + B_3$.

Next, we show that $h' < h + B_3$. Assuming the contrary, we have that $H \geq h' \geq h + B_3$, and get from Lemma 9.8 and Corollary 9.9 that

$$\begin{aligned}
&p + (\vartheta+1)\mathcal{S}(M) + \sigma_h \\
&\leq \min\left\{\max_{v\in V_g}\{p_{v,1}\} + (M-1)(T+\mathcal{S}(1)) + (\vartheta+1)\mathcal{S}(M) + \sigma_h, H\right\} \\
&\leq \min\left\{h + \sigma_h + R^+ + (M-1)(T+\mathcal{S}(1)) + (\vartheta+1)\mathcal{S}(M) + \sigma_h, h+B_3\right\} \\
&< h + B_3\,,
\end{aligned}$$

where the last step uses Inequality (9.5). Thus, as H is larger than this time, each $v \in V_g$ triggers its NEXT signal before time $h + B_3 - \sigma_h$, because the corollary also shows that $\max_{v\in V_g}\{p_{v,M}\} \leq p + \mathcal{S}(M)$, and nodes wait for $\vartheta\mathcal{S}(M)$ local time before triggering the signal. On the other hand, Lemma 9.8, Corollary 9.9, and Inequality (9.6) show that

$$\begin{aligned}
p + \mathcal{S}(M) &\geq \min_{v\in V_g}\{p_{v,1}\} + (M-1)\left(\frac{T}{\vartheta} - \mathcal{S}(1)\right) + \mathcal{S}(M) \\
&\geq h + \frac{R^-}{\vartheta} + (M-1)\left(\frac{T}{\vartheta} - \mathcal{S}(1)\right) + \mathcal{S}(M) \\
&\geq h + B_2\,,
\end{aligned}$$

i.e., all of these NEXT signals are triggered no earlier than time $h + B_2$. By Definition 9.7, this entails that $h' \leq p+(\vartheta+1)\mathcal{S}(M)+\sigma_h < h+B_3$, contradicting the assumption that $h' \geq h + B_3$.

Knowing that $h' < h + B_3$, we can conclude that $\max_{v\in V_g}\{p_{v,M}\} \leq p + \mathcal{S}(M) < h' \leq H$. As we can derive the same bounds as above, we also get that

$\max_{v\in V_g}\{h_{v,2}\} \leq p + (\vartheta+1)\mathcal{S}(M) + \sigma_h = \min_{v\in V_g}\{p_{v,M}\} + (\vartheta+1)\mathcal{S}(M) + \sigma_h$, provided that no node performs a reset before triggering its NEXT signal, i.e., $H > p + (\vartheta+1)\mathcal{S}(M) + \sigma_h$. Recall that we already established that $H \geq h' > \max_{v\in V_g}\{p_{v,M}\}$, i.e., the local i variables have been set to 0 mod M again and will not change before the next pulse. Checking Algorithm 6, we see that such a reset thus would either occur R^+ local time after the (local) beat or due to the next pulse occuring before local time $h_{v,2} + R^-$. As $R^+/\vartheta \geq (\vartheta+1)\mathcal{S}(M) + \sigma_h$ by Inequality (9.7), the former cannot happen.

Observe that if the latter does not take place either, it would indeed follow that no node performs a reset on its second beat. Therefore, we conclude that $H \geq \min_{v\in V_g}\{p_{v,M+1}, p + (\vartheta+1)\mathcal{S}(M) + \sigma_h\}$ (where we slightly abuse notation in that if v *would* generate pulse $M+1$, but Algorithm 6 prevents this and performs a reset instead, we still denote this time by $p_{v,M+1}$). Finally, assume for contradiction that $H < p + (\vartheta+1)\mathcal{S}(M) + \sigma_h$. Thus, there is some $v \in V_g$ so that $H = p_{v,M+1} < p + (\vartheta+1)\mathcal{S}(M) + \sigma_h$. However, as v is the first node performing a reset, the period bound applies, i.e.,

$$\begin{aligned} p_{v,M+1} &\geq p + \frac{T}{\vartheta} - \mathcal{S}(1) \\ &= p + (\vartheta^2+\vartheta)\mathcal{S}(1) + \vartheta d \\ &> p + (\vartheta+1)\mathcal{S}(M) + 2(\mathcal{S}(1) - \mathcal{S}(M)) \\ &\geq p + (\vartheta+1)\mathcal{S}(M) + \sigma_h\,, \end{aligned}$$

where the last step uses Inequality (9.8). Thus all possible cases lead to the desired bounds on $h_{v,2}$ for all $v \in V_g$. □

We summarize today's findings in the following theorem.

Theorem 9.11. *Assume that* $3+4\vartheta-4\vartheta^2-2\vartheta^3 > 0$ *and Inequalities* (9.1)-(9.8) *hold. Set* $T \coloneqq \vartheta((\vartheta^2+\vartheta+1)\mathcal{S}(1) + \vartheta d)$*, where* $\mathcal{S}(1) \coloneqq R^+ + \sigma_h - R^-/\vartheta$*. If the beats behave as required by Definition 9.7, Algorithm 6 running in conjunction with Algorithm 5 (where estimates are computed according to Lemma 5.9) is a self-stabilizing solution to the pulse synchronization problem. Its skew is in* $\mathcal{O}\left(u + (\vartheta-1)(d + \mathcal{S}(1))\right)$ *and the generated pulses satisfy* $P_{\min} \geq T/\vartheta - 2\mathcal{S}(1)$ *and* $P_{\max} \leq T + 2\mathcal{S}(1)$*. The stabilization time (not accounting for the beats) is* $\mathcal{O}(MT)$*.*

Proof. We apply Lemma 9.10 to each beat but the first, showing that $H = \infty$. Corollary 9.9 then yields the claims. □

Bibliographic Notes

The concept of self-stabilization was introduced by Dijkstra [Dij74]. The definition here is more general, but in turn also somewhat informal — notions like "time" need to be assigned meaning according to the specific system model. There are some generic constructions for self-stabilizing algorithms. For instance, Awerbuch et al. showed that any synchronous message-passing algorithm can be modified into a self-stabilizing asynchronous message-passing algorithm that stabilizes in the same time as needed to compute the solution from scratch [APSV91]. The approach to making the Lynch-Welch algorithm self-stabilizing discussed in this lecture is taken from [KL18].

Bibliography

[APSV91] Baruch Awerbuch, Boaz Patt-Shamir, and George Varghese. Self-Stabilization By Local Checking and Correction. In *Symposium on Foundations of Computer Science (FOCS)*, 1991.

[Dij74] Edsger W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11):943–644, November 1974.

[KL18] Pankaj Khanchandani and Christoph Lenzen. Self-Stabilizing Byzantine Clock Synchronization with Optimal Precision. *Theory of Computing Systems*, 2018.