12 Self-Stabilization and Recovery

Chapter Contents

Learning Goals

CL: todo

12.1 Overview

In Chapters 9 and 10 we studied how to achieve synchronization despite Bzyantine, i.e., worst-case failures of a minority of the nodes in a system. Our solutions required full connectivity, which is impractical for all but small systems. To make do with fewer links, in Chapter 11 we relaxed the "where" of faults, in the sense that we dropped the assumption of a worst-case *distribution* of faults. We saw that if faults are (close to) uniformly distributed across the system, fairly high probabilities of individual node failure can be supported with very small node degrees.

In this chapter, we introduce a related notions corresponding to the "when" of faults. Rather than assuming that a node becomes forever, what happens if faults are *transient* in nature? Possibilities for such faults are endless: radiation-induced spurious pulses on communication links or bit flips in memory, a temporary drop or outage of supply voltage, electro-magnetic interference from a close-by source, delay bound violations due to (local) overheating, etc. In none of these examples, there is physical damage to the hardware, meaning that it will commence to function correctly again as soon as the transient event is over.

So far, our models did not consider that nodes violating the behavior prescribed by our algorithms could ever recover. This has dramatic consequences in a setting where transient faults are more common that permanent ones.

E12.1 Suppose that every second, a correct node becomes faulty with (independent) probability $p = 10^{-8}$. What is the mean time between failures (MTBF), i.e., the expected time until the node fails? (Hint: For $|q| < 1$, $\sum_{i=1}^{\infty} iq^{i-1} = (1-q)^{-2}$.)

E12.2 Now assume that we add redundancy to make the system more reliable. Using $n = 10$ nodes and an algorithm that can sustain up to any $f = 3$ nodes being faulty, what is the MTBF of the algorithm on this system if correct nodes still become faulty with independent probability $p = 10^{-8}$ each second? What about $n = 100$ and $f = 33$, or $n = 1000$ and $f = 333$?

E12.3 Are the assumptions that the failure rate remains constant and nodes fail independently when adding redundancy realistic? If not, how does this affect the MTBF?

But how should we characterize what makes a fault "transient?" Also here, we follow a similar strategy as in earlier chapters. Rather than relying on an empiric approach—deriving from data on deployed systems, experiments, or simulations which faults and resulting behavior to expect—we aim for the broadest behavioral characterization we can handle; this reduces the burden of determining and verifying fault models, while simultaneously maximizing

the possible scenarios we cover. Given that we rely on nodes executing the algorithms we design to prove properties of the system, a natural minimal requirement is that for a node to be considered correct again after undergoing a transient fault, it must faithfully execute the specified algorithm. Obviously, what precisely this means depends on the system model we employ. To be concrete, consider the TMP model, which will continue to be our model of choice.

Definition 12.1 (Correct Nodes in TMP)**.** *Node $v \in V$ is correct during $[t, t'] \subseteq \mathbb{R}_{\geq 0}$ if and only if*

- *at each time $t_r \in [t, t']$, v receives a message from $w \in V$ only if the same message was sent by w during $[t_r - d, t_r)$,*
- *if $w \in V$ sends a message to v at time $t_s \in [t, t']$, there is a unique time $t_r \in (t_s, t_s + d]$ when the message is received by v,*
- *at each time $t'' \in [t, t']$, $1 \leq \frac{dH_v}{dt}(t'') \leq \vartheta$, and*
- *the state machine of v performs state transitions and sends messages according to the TMP model and the specified algorithm.*

Note that this definition associates each link (w, v) with its endpoint v, in the sense that v is considered faulty whenever (w, v) does not behave correctly. Other mappings of links to nodes are perfectly fine, as is to treat the links as separate entities, distinguishing between correct and faulty links. We make this choice for convenience, and because especially in low-degree networks, the results would remain very similar.

More importantly, the above definition again encapsulates the idea that no assumptions are made on the behavior of faulty nodes. That is, we continue to consider Byzantine faults, and an algorithm needs to overcome *any* possible behavior of faulty nodes. In other words, correct nodes receive arbitrary messages on links originating at faulty nodes.

This worst-case assumption is also significant for what we face when a transient fault is over. The challenge is that, while the now again correct node faithfully executes the algorithm, its state machine is in an arbitrary (read: the worst possible choice of) state. As we made no assumptions on what happens to the node during its transient failure, its state-holding memory could have transitioned to any configuration. In particular, the node might have lost any and all information on the state of the remaining system, instead storing completely misleading incorrect data. Moreover, the node has no way of reliably being *aware* that it just suffered a transient fault, as any pertinent diagnostic data might have been lost or compromised, too. This requires highly circumspect design of algorithms, in that one needs to make sure that in *all* cases, such

inconsistent state information is eventually removed and the node starts to operate as intended also in relation to the state of the system as a whole.

As the first result of this chapter, we show that with only small modifications, Algorithm 11 can be augmented such that nodes resynchronize after suffering a transient fault.

Theorem 12.13. *Suppose that* $11 - 10\vartheta^2 > 0$*, fix any*

$$T \geq \frac{2\vartheta^2(2\vartheta - 1)u + \vartheta^3(4\vartheta - 3)d}{19 - 18\vartheta^2} \in O(d), \tag{12.3}$$

and set

$$\mathcal{S} \coloneqq \frac{2(2\vartheta - 1)(u + (\vartheta - 1)d) + 2(\vartheta - 1)T}{\vartheta(9 - 8\vartheta^2)} \in O(u + (\vartheta - 1)T). \tag{12.4}$$

Then there is $T_0 \in O(T)$ *such that the following holds. If for each time* $t \in \mathbb{R}_{\geq 0}$*, the set* $V_g(t) \subseteq V$ *of all nodes that are correct during* $[\max\{t - T_0, 0\}, t]$ *has size at least* $n - f$ *and* $\max_{v \in V_g(T_0)}\{H_v(0)\} \leq \mathcal{S}$*, then Algorithm 11 lets these nodes generate pulses with skew at most* $\mathcal{S}$*,* $P_{\min} \geq (T - (\vartheta + 1)\mathcal{S})/\vartheta$*, and* $P_{\max} \leq T + 3\mathcal{S}$*. More precisely, we can inductively label for all* $t \in \mathbb{R}_{\geq 0}$ *and all pulses by nodes* $v \in V_g(t)$ *by round numbers* $k \in \mathbb{N}_{>0}$ *such that (i) pulses with the same round number are at most* $\mathcal{S}$ *time apart, (ii) pulses with consecutive round numbers are at least* $P_{\min}$ *time apart, and (iii) pulses with consecutive round numbers are at most* $P_{\max}$ *time apart.*

We emphasize that this constitutes a huge improvement in overall robustness of the system in face of *independent* transient faults.

E12.4 Suppose that every second, each correct node becomes faulty with independent probability $p = 10^{-8}$. However, so long as no more than $n - f$ nodes are neither faulty nor in the process of recovery, it will recover, i.e., resynchronize with the correct synchronized nodes, within another second. Using $n = 10$ nodes and an algorithm that can sustain up to any $f = 3$ nodes being faulty or unsynchronized, what is the MTBF? What about $n = 100$ and $f = 33$, or $n = 1000$ and $f = 333$?

E12.5 What if faults are correlated? In the extreme case of the only mode of failure being all nodes failing together, is there any gain in the result of Theorem 12.13?

Take careful note of the requirement that faults are independent. If a too large fraction of the nodes transiently fails concurrently, the system will still fail as a whole. In this case, no guarantee of recovery is given.

Avoiding any assumption on independence of faults, Edger Dijkstra introduced an even stronger notion of recovery, which is called *self-stabilization*. Here, the goal is for the system to recover from arbitrary transient faults, including all nodes jointly failing. Thus, once the transient faults are over, the system

is expected to recommence its intended operation. The meaning of "transient" is determined by specifying the space of executions that are considered feasible, i.e., satisfying the model assumptions; the meaning of "intended operation" is determined by specifying a subset of executions that are considered to fulfil the requirements of the task at hand.

Definition 12.2 (Self-stabilization)**.** *Denote by $\Upsilon_{\mathcal{A}}$ the set of feasible executions of an algorithm $\mathcal{A}$ in a given system, which is closed under taking suffixes (i.e., removing an arbitrarily long initial part of an execution $\mathcal{E} \in \Upsilon_{\mathcal{A}}$ must result in an execution $\mathcal{E}' \in \Upsilon_{\mathcal{A}}$). Denote by $\Upsilon_{\mathcal{A},T} \subseteq \Upsilon_{\mathcal{A}}$ the subset of executions satisfying the specification of some task T that $\mathcal{A}$ is supposed to solve. Then $\mathcal{A}$ is* self-stabilizing *if and only if every execution $\mathcal{E} \in \Upsilon_{\mathcal{A}}$ has a suffix $\mathcal{E}' \in \Upsilon_{\mathcal{A},T}$.*

If there is a length measure ℓ for execution prefixes and for each $\mathcal{E} \in \Upsilon_{\mathcal{A}}$ there is $\mathcal{E}' \in \Upsilon_{\mathcal{A},T}$ such that $\ell(\mathcal{E} \setminus \mathcal{E}') \leq L$, then $\mathcal{A}$ has stabilization time L.

To make this highly abstract definition more palpable, let us illustrate the concept by an example. As the set of feasible executions, we choose fault-free TMP executions. In other words, we consider an execution feasible if and only if it behaves according to the model from the previous sections. A suffix of such an execution is obtained by simply restricting the execution to times $t' \geq t$ for some time t.

Definition 12.3 (Suffixes of TMP Executions)**.** *Denote by $\Upsilon_{\mathcal{A}}(0)$ the set of feasible executions of TMP Algorithm $\mathcal{A}$ according to Section 1.2. For $t \in \mathbb{R}_{>0}$, the set of feasible executions $\Upsilon_{\mathcal{A}}(t)$ of $\mathcal{A}$ is obtained by restricting executions $\mathcal{E} \in \Upsilon_{\mathcal{A}}(0)$ to times $t' \geq t$, i.e., discarding all events at times $t' < t$ from $\mathcal{E}$ and restricting hardware clocks and all other functions to times $t' \geq t$. We define $\Upsilon_{\mathcal{A}} := \bigcup_{t \in \mathbb{R}_{\geq 0}} \Upsilon_{\mathcal{A}}(t)$. Execution $\mathcal{E}' \in \Upsilon_{\mathcal{A}}(t')$ is a suffix of $\mathcal{E} \in \Upsilon_{\mathcal{A}}(t)$ if and only if $\mathcal{E}'$ can be obtained from $\mathcal{E}$ in the manner described above, and in this case the length of the prefix is $\ell(\mathcal{E} \setminus \mathcal{E}') := t' - t$.*

Recall that for clock synchronization algorithm $\mathcal{A}$ outputting logical clocks $L_v \colon \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$, we defined

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

$$\mathcal{L}(t) := \max_{\substack{v,w \in V_g \\ \wedge (v,w) \in E}} \{|L_v(t) - L_w(t)|\}$$

and set $\mathcal{G}$ and $\mathcal{L}$ to the supremum of these values across all times $t \in \mathbb{R}_{\geq 0}$. We define $\mathcal{G}$ and $\mathcal{L}$ for executions in $\Upsilon_{\mathcal{A}}(t)$ for $t > 0$ analogously, by setting

$$\mathcal{G} := \sup_{t' \geq t} \{\mathcal{G}(t)\}$$
$$\mathcal{L} := \sup_{t' \geq t} \{\mathcal{L}(t)\}.$$

With these definitions in place, we can now set our task to be as in Chapter 8, i.e., ensuring logical clock rates between 1 and $(1+\mu)\frac{dH_v}{dt}$, a global skew of $O(uD)$, and a local skew of $O(u \log_{\mu/(\vartheta-1)} D)$. For this particular example, it turns out that the algorithm presented in the chapter already is self-stabilizing.

Theorem 12.18. *Suppose that*

- *Algorithm 13 with $T = d$ is used to compute clock estimates,*
- $2(\vartheta - 1) \leq \mu = O(u/d)$*, and*
- $\kappa = \delta$*, where δ is as in Lemma 8.31.*

Then Algorithm 14 is a self-stabilizing solution to the task of guaranteeing that

- $\frac{dH_v}{dt}(t) \leq \frac{dL_v}{dt}(t) \leq (1+\mu)\frac{dH_v}{dt}(t)$ *for all nodes v and times t,*
- $\mathcal{G} = O(uD)$*, and*
- $\mathcal{L} = O(u \log_\sigma D)$*, where $\sigma = \mu/(\vartheta - 1)$.*

These properties are established within $O(\mathcal{G}(0)/\mu)$ time.

This seemingly powerful result is rendered ineffective by two important, related drawbacks. First, the algorithm does not guarantee a (finite) stabilization time. This can easily be seen from the fact that Algorithm 14 has no mechanism for adjusting logical clocks at rates other than $\frac{dH_v}{dt}$ and $(1+\mu)\frac{dH_v}{dt}$. However, a transient fault could change the logical clock value of node v arbitrarily, as a mutable variable like a logical clock value must be stored in *some* kind of (volatile) memory. Thus, a transient fault could result in an arbitrarily large global skew, which subsequently can only be reduced at a rate of μ.

The reader might now bring up the point that the global skew cannot, in fact, become arbitrarily large: the abstraction of representing hardware and logical clocks by functions $\mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ is hiding that any physical implementation will have finite memory and, accordingly, will operate with bounded clocks that eventually wrap around to 0. Unfortunately, this only causes further trouble. The (formally) self-stabilizing Algorithm 6, which computes estimates of the difference in logical clock values to those of neighbors, is stated under the assumption of unbounded logical clocks whose values are explicitly communicated. While under fault-free operation it is not too hard to implement this

abstraction efficiently with small messages, a self-stabilizing solution is much more involved.

E12.6 Assume that logical clocks have rates between 1 and $1 + O(u/d)$ and $(\vartheta - 1)d \ll u$, and are initialized to 0. Describe a simple way to estimate logical clock differences to neighbors with error $O(u)$ that requires messages of $O(\mathcal{L}/u)$ every $\Theta(d)$ time and locally stores only $O(\mathcal{L}/u)$ bits. (You may assume that the logical clock increments discretely in steps of u, without worrying about the logic implementing this or the hardware clock.)

E12.7 Can your scheme also be made to work if there is some initial, but bounded skew between logical clocks? What if there is no bound on the initial skew?

Remark 12.4. *This issue highlights the gap between the TMP model and the pseudocode. The pseudocode uses instructions that cannot be directly implemented in the model, but rather need subroutines realizing them. These in turn might make use of assumptions on initialization or invariants maintained by the algorithm. However, when considering self-stabilization, any such guarantee breaks down. If we pretend that one can implement the pseudocode without providing details, we are implicitly adding model assumptions, making it all the harder (and sometimes impossible) to implement the augmented model.*

This is a major pitfall in designing self-stabilizing algorithms: properties that are trivial to ensure under correct operation might be inadvertently taken for granted by the algorithm designer. Depending on the point of view, this either results in a more restrictive model that cannot be implemented (or not efficiently so), or in pseudocode that does not actually describe an algorithm in the stated model. Similarly, pushing assumptions about initialization into the model might defeat the purpose of self-stabilization: When assuming something about the volatile state of nodes at the beginning of feasible executions (e.g. bounded skew), we implicitly make any "transient" fault affecting the memory holding this state into a permanent one from the perspective of the model; so long as the skew bound is not re-established, the execution starting at that point in time is not considered feasible, conveniently removing the burden (and the benefits) of ensuring recovery from the "self-stabilizing" algorithm.

To overcome the above obstacle in our example, we introduce a recovery mechanism that bounds $\mathcal{G}$ in a self-stabilizing manner, but does not interfere with logical clocks whenever it is small enough. Using clocks that are bounded, but do not wrap around to 0 too often (e.g., with a period of at least $5\mathcal{G}$), such a bound allows us to implement Algorithm 6. By combining both algorithms, we (i) get a self-stabilizing implementation of Algorithm 6 and (ii) ensure that the global skew is reduced to within a constant factor of the target value quickly.

From there, Algorithm 5 can take over to maintain a small global skew (thus ensuring that the recovery mechanism does not meddle) and converge to a small local skew in time $O(\mathcal{G}/\mu)$.

Theorem 12.24. *Suppose that*

- *Algorithm 6 with $T = d$ is used to compute clock estimates,*
- $2(\vartheta - 1) \leq \mu = O(u/d)$,
- $\kappa = \delta$, *where δ is as in Lemma 8.31, and*
- *Algorithm 15 is run with a sufficiently large value for $\mathcal{G} = O(uD)$.*

With this modification, Algorithm 14 is a self-stabilizing solution to the task of guaranteeing that

- $\frac{dH_v}{dt}(t) \leq \frac{dL_v}{dt}(t) \leq (1+\mu)\frac{dH_v}{dt}(t)$ *for all nodes v and times t,*
- $\mathcal{G} = O(uD)$, *and*
- $\mathcal{L} = O(u \log_\sigma D)$, *where $\sigma = \mu/(\vartheta - 1)$.*

It has a stabilization time of $O(dD)$.

12.2 A Variant of the Lynch-Welch Algorithm

Algorithm 12 modifies Algorithm 11 to allow for recovery of nodes after transient faults. We made the following changes:

1. All waiting statements check whether the local time until which too wait is in the past or too far in the future, and in this case stop the waiting instruction.
2. The waiting periods before sending the message for the current round and for receiving other nodes' messages allow for twice the skew. This way, resynchronizing nodes need not immediately meet the skew bound to start executing rounds correctly again.
3. A resynchronizing node might not receive all $n - f$ messages from correct nodes during the expected window. To avoid that this can cause faulty nodes to "lure" them away from the crowd, missing values are replaced by the median of received values. From the perspective of approximate agreement, this is safe in the sense that this guarantees that the output lies in the range spanned by correct nodes. (The convergence argument does not apply, however, which is the reason for the additional slack in the initial waiting periods.)
4. If too few (i.e., less than $n - f$) messages are received, the node switches into a recovery mode, in which it waits until it observes a pulse from the crowd. As this pulse must be tightly synchronized, this is implemented by a "sliding window"—the node waits until it observes $n - f$ messages

Algorithm 12 Lynch-Welch pulse synchronization algorithm with recovery mechanism, code for node $v \in V_g$.

```
1:  wait until getH() ≥ S                          // H_w(0) ∈ [0, S) for correct nodes w
2:  for all rounds r ∈ ℕ do
3:      generate r-th pulse
4:      h ← getH()
5:      wait until getH() ≥ h + 2ϑS or getH() < h  // all nodes are in round r
6:      broadcast empty message to all nodes (including self)
7:      wait until getH() ≥ h + 2(ϑ^2 + ϑ)S + ϑd or getH() < h + 2ϑS   // denote this time by τ_{v,r}
                                                   // correct nodes' messages should have arrived
8:      if received messages from n − f distinct nodes during current loop iteration then
9:          h' ← median of {h_x | h_x local reception time of latest message from some x ∈ V} (as multiset, i.e., values may repeat)
10:         for each node w ∈ V do
11:             if received message from w during current loop iteration then
12:                 let h_w be local time at latest received message from w
13:                 Δ(w) ← h_w − h − d + u − 2S
14:             else
15:                 Δ(w) ← h' − h − d + u − 2S // replace missing ones by median
16:             end if
17:         end for
18:         U ← {Δ(w) | w ∈ V} (as multiset, i.e., values may repeat)
19:         Δ ← (U^(f+1) + U^(n−f)) /2
20:         wait until getH() ≥ h + Δ + T or getH() < h + Δ − 3S
21:     else
22:         wait for n − f messages from distinct nodes within ϑ^2 S + ϑu local time
23:         let h' be the local reception time of the (f + 1)-th such message
24:         wait until getH() ≥ h' − d + u − 2S + T or getH() < h' − (ϑ^2 S + ϑu)
25:     end if
26: end for
```

from distinct sources within the narrow local time window guaranteed by the skew bound $\mathcal{S}$, the delay uncertainty u, and the drift bound of ϑ for the hardware clocks. Then it jumps onto the moving train, re-initializes the loop with sufficiently accurate timing.

E12.8 How would you implement the waiting statement in Line 22?

E12.9 Revisit your implementation of Line 22. Did you make sure that transient faults cannot result in a state that causes the node to deadlock or fail to implement the waiting statement correctly? If not, what is the stabilization time of your subroutine?

Before proving the recovery property, we first adapt the definition of correct execution of round $k \in \mathbb{N}_{>0}$. Note that for nodes undergoing transient faults, the k-th local pulse might not correspond to the k-th "global" pulse, assuming that we can define the latter term in a meaningful way. However, we will make assumptions enabling this, so we use according indexation in the definition.

(i) There is a set V_k of $n - f$ correct nodes that start a loop iteration within $\mathcal{S}$ time and stay correct until completing the iteration.

(ii) Messages sent by these nodes are received at all such nodes after starting the loop iteration and before they evaluate the if-condition, i.e., during $[p_{v,k}, \tau_{v,k}]$.

(iii) These nodes execute the if-block.

(iv) T is large enough to accomodate the adjustments of the next pulse times. This means to ensure that $H_v(\tau_{v,k}) \leq h_v(\tau_{v,k}) + \Delta_v(\tau_{v,k}) + T$, i.e., that $H_v(\tau_{v,k}) \leq H_v(p_{v,k}) + \Delta + T$, where Δ is the value determined in Line 18 of the k-th iteration of the loop.

Note that the fact that these nodes stay correct until completing the loop iteration means they generate another pulse at this point. By tying iterations together using these new pulses as reference—because $n - f$ nodes are required for each iteration, there is an overlap in consecutive sets—we will inductively define rounds $k \in \mathbb{N}_{>0}$, even if individual nodes might not be able to keep track of k due to transient faults.

With this in mind, we can define that *round k is executed correctly by set V_k* if these properties are satisfied in round k.

We also preemptively set

$$\delta := u + (\vartheta - 1)d + (2\vartheta^2 + \vartheta - 3)\mathcal{S}$$

for this section. This value is slightly larger than the δ used in Chapter 10, accounting for the larger waiting times used for sending and receiving messages of Algorithm 12.

We start by repeating the analysis from Chapter 10 for the modified algorithm, showing that its synchronization properties are maintained. The following are adjusted versions of Lemma 10.7, Lemma 10.8, and Theorem 10.9.

Lemma 12.5. *Suppose that* $T \geq (2\vartheta^2 + 2\vartheta + 1)\mathcal{S} + \vartheta d$ *and*

$$\mathcal{S} \geq \frac{2(2\vartheta - 1)\delta + 2(\vartheta - 1)T}{2 - \vartheta}.$$

Moreover, assume V_{r-1} *executed round* $r - 1$ *correctly and that* $V_r \subseteq V$ *with* $|V_r| \geq n - f$ *starts a loop iteration during* $[t_r, t_r + \mathcal{S}]$*, where for some* $v \in V_{r-1} \cap V_r \neq \emptyset$ *the new loop iteration is the one right after that of round* $r - 1$*. If* V_r *is correct during* $[t_r, t_r + T + \mathcal{S} + \delta]$*, then*

(i) round r *is executed correctly by* V_r*,*
(ii) $(T - \mathcal{S})/\vartheta \leq \min_{v \in V_g}\{p_{v,r+1}\} - \min_{v \in V_g}\{p_{v,r}\} \leq T + \mathcal{S} + \delta$*, and*
(iii) the nodes in V_r *start the next loop iteration within* $\mathcal{S}$ *time.*

Proof sketch. Consider the message $v \in V_r$ sends after entering round r. If it is sent at time t_s, it is received at time $t_r \in [t_s + d - u, t_s + d]$. If a message sent earlier by v arrives later, then the fact that v is correct during this iteration implies that this message is also received during $[t_s + d - u, t_s + d]$. Since Algorithm 12 makes use only of the latest such received message in Line 12, for the purpose of this proof we may treat this case simply as if the message v sent in round r arrives at this time instead.

From here on, the proof is analogous to that of Lemma 10.7, with the following modifications:

- all statements are about nodes in V_r,
- we note that the waiting statements before the if-block complete because the first criterion holds with equality,
- we replace the respective local waiting times in the proof by the modified ones of Lines 5 and 7 (i.e., account for the additional factors of 2),
- we note that from showing that all messages from nodes in V_r are received in a timely fashion and $|V_r| \geq n - f$, it follows that nodes in V_r execute the if-statement,
- we use Lemma 12.6 to show that the estimates computed in Line 13 have indeed error at most δ, and
- we conclude from the determined bounds on Δ that the waiting statement in Line 20 completes due to the first condition being met with equality. □

Lemma 12.6. *Suppose that the prerequisites of Lemma 12.5 hold and* $v \in V_r$ *receives the message from* $w \in V_r$ *in the loop iteration started at time* t *at time* t_r*. Then setting*

$$\Delta(w) := H_v(t_r) - H_v(p_{v,r}) - d + u - 2\mathcal{S}$$

as in Line 13 produces an estimate with error at most $\delta = u + (\vartheta - 1)d + (2\vartheta^2 + \vartheta - 3)\mathcal{S}$.

Proof. Analogous to the one of Lemma 10.8, accounting for the additional factors of 2 in Lines 5 and 7. □

Theorem 12.7. *Suppose that there is a set* $V_g \subseteq V$ *of size at least* $n - f$ *that is correct at all times and that* $11 - 10\vartheta^3 > 0$. *For any choice of*

$$T \geq \frac{2\vartheta^2(2\vartheta - 1)u + \vartheta^3(4\vartheta - 3)d}{19 - 18\vartheta^2} \in \mathcal{O}(d), \tag{12.1}$$

set

$$\mathcal{S} := \frac{2(2\vartheta - 1)(u + (\vartheta - 1)d) + 2(\vartheta - 1)T}{\vartheta(9 - 8\vartheta^2)} \in \mathcal{O}(u + (\vartheta - 1)T). \tag{12.2}$$

If $\max_{v \in V}\{H_v(0)\} \leq \mathcal{S}$, *then Algorithm 11 solves pulse synchronization with skew at most* $\mathcal{S}$, $P_{\min} \geq (T - (\vartheta + 1)\mathcal{S})/\vartheta$, *and* $P_{\max} \leq T + 3\mathcal{S}$.

Proof sketch. The proof is analogous to the one of Theorem 10.9, but with $\delta = u + (\vartheta - 1)d + (2\vartheta^2 + \vartheta - 3)\mathcal{S}$ as defined earlier, $\mathcal{S}$ and T as in the prerequisites, and using Lemma 12.5 for the induction step. □

We now turn to showing that correct nodes can resynchronize, so long as $n - f$ nodes keep producing synchronized pulses. To keep the notation tractable, we employ the trick of assuming that there is a majority of correct nodes running in synchrony at all times, and that recovering nodes do not fail again. We will later circumvent this limitation by arguing that (i) future faults do not affect the present state and (ii) due to the "forgetful" nature of the algorithm, we can remove execution prefixes with additional faults (up to a certain point) without affecting the behavior of the nodes. This then allows us to "pretend" that the preconditions used in our statements apply.

As a first helper lemma, we observe that the messages corresponding to a pulse of such a synchronized majority are received in close temporal proximity.

Lemma 12.8. *Suppose that the prerequisites of Theorem 12.7 hold. Then all messages from nodes in* V_g *sent in round* $k \in \mathbb{N}_{>0}$ *are received within a time interval of length* $\vartheta\mathcal{S} + u$.

Proof. Let $v, w \in V_g$ send their messages at times t_s and t'_s, respectively. Denote by t_r and t'_r the respective times when the messages are received, we

bound

delays in $[d-u, d]$

$$\begin{aligned} |t_r - t_r'| &\le |t_s - t_s'| + u \\ &= |H_v^{-1}(H_v(p_{v,k}) + \vartheta\mathcal{S}) - H_w^{-1}(H_w(p_{w,k}) + \vartheta\mathcal{S})| + u \\ &\le |p_{v,k} - p_{w,k}| + (\vartheta - 1)\mathcal{S} + u \\ &\le \vartheta\mathcal{S} + u. \end{aligned}$$

$1 \le \frac{dH}{dt} \le \vartheta$

Theorem 12.7

□

Next, we show that no matter which state a node is in when it becomes correct, it will start a new loop iteration within $3T$, effectively clearing its state (except for the time when the loop iteration starts). The previous lemma here serves to show that a node will leave the recovery mode (the else-block of the loop) in a timely fashion if it ends up there.

Lemma 12.9. *Suppose that the prerequisites of Theorem 12.7 hold and let $v \in V \setminus V_g$ be correct at times $t \ge t_v \in \mathbb{R}_{\ge 0}$. Then v starts a new iteration of the loop of Algorithm 12 by time $t_v + 3T$.*

Proof. If v currently is not executing the loop, recall that at time t_v, getH() returns $H_v(t_v) \in \mathbb{R}_{\ge 0}$. Hence,

$\frac{dH_v}{dt} \ge 1$

$$H_v(t_v + \mathcal{S}) \ge H_v(t_v) + \mathcal{S} \ge \mathcal{S},$$

implying that the condition to start a loop iteration is satisfied by time $t_v + \mathcal{S}$. Accordingly, assume that v is executing the loop at time t_v.

Observe that if v is executing parts of the loop before the if-else branch, it will complete this at some time $t_v' \le t_v + 2(\vartheta^2 + \vartheta)\mathcal{S} + \vartheta d$, regardless of the value $h(t_v)$ it stores in h at time t_v. Hence, it remains to show that the loop iteration completes in a timely fashion after the branch.

As $|V_g| \ge n - f$, by Theorem 12.7 the nodes in V_g solve pulse synchronization with skew $\mathcal{S}$, $P_{\min} = (T - (\vartheta + 1)\mathcal{S})/\vartheta$ and $P_{\max} = T + 3\mathcal{S}$. Let r be maximal such that there is some node $w \in V_g$ satisfying $p_{w,r-1} < t_v$ (if no pulses have been sent yet, set $r := 1$). Hence, we have for all $w \in V_g$ that $p_{w,k} \in [t_v, t_v + P_{\max} \le t_v + T + 3\mathcal{S}]$. Denote by $t_s(w,k)$ the time when w sends its message for pulse k to v and by $t_r(w,k)$ the time when it is received by v.

We distinguish two cases. If v executes the if statement of the branch, it waits for at most $T + 3\mathcal{S}$ local time before completing the loop iteration, meaning that it completes the loop iteration by time

(12.3),(12.4)

$$H_v^{-1}(H_v(t') + T + 3\mathcal{S}) \le t_v' + T + 3\mathcal{S} \le t_v + T + (2\vartheta^2 + 2\vartheta + 3)\mathcal{S} + \vartheta d < t_v + 2T,$$

as claimed.

If v executes the else statement of the branch, this implies that is some $w \in V_g$ such that $t_r(w,k) > t_v'$; otherwise, v would have stored reception times for

$n - f$ distinct nodes during the current loop iteration by time t'_v and hence executed the if statement. By Lemma 12.8, for all $x, y \in V_g$ we have that $|t_r(x,k) - t_r(y,k)| \leq \vartheta\mathcal{S} + u$. Therefore, there is some $t_0 > t'_v$ such that v receives messages from $n - f$ distinct nodes during $[t_0 - \vartheta\mathcal{S} - u, t_0]$ and

$$\begin{aligned}
t_0 &\leq \max_{w \in V_g} \{t_r(w,k)\} \\
&\leq \max_{w \in V_g} \{t_s(w,k)\} + d && \text{max. delay} \\
&= \max_{w \in V_g} \{H_w^{-1}(H_w(p_{w,k} + 2\vartheta\mathcal{S})\} + d \\
&\leq \max_{w \in V_g} \{p_{w,k}\} + 2\vartheta\mathcal{S} + d && \tfrac{dH_w}{dt} \geq 1 \\
&\leq t_v + P_{\max} + 2\vartheta\mathcal{S} + d && \text{choice of } k \\
&= t_v + T + (2\vartheta + 3)\mathcal{S} + d
\end{aligned}$$

Because $\vartheta\mathcal{S} + u$ time corresponds to at most $\vartheta^2\mathcal{S} + \vartheta u$ local time, v thus passes the first waiting statement in the else branch no later than time $t_0 + \vartheta\mathcal{S} + u \leq t_v + T + (3\vartheta + 3)\mathcal{S} + d + u$. Finally, the second waiting statement is complete within at most $T - d + u - 2\mathcal{S}$ additional time, i.e., no later than time

$$t_v + 2T + (3\vartheta + 1)\mathcal{S} + 2u < t_v + 3T. \qquad \square \quad (12.3),(12.4)$$

A second helper lemma shows that all messages received from synchronized nodes while waiting for messages before the if-branch correspond to the same pulse.

Lemma 12.10. *Suppose that the prerequisites of Theorem 12.7 hold and fix any $t_0 \in \mathbb{R}_{\geq 0}$. Then there is some time $t_1 \in [t_0, t_0 + 2(\vartheta^2 + \vartheta)\mathcal{S} + \vartheta d]$ such that all messages received from nodes $v \in V_g$ during this interval are*

- *received during $[t_1, t_1 + \vartheta\mathcal{S} + u]$ and*
- *all such messages belong to the same pulse, i.e., have been sent during the k-th loop iteration of the sender for some $k \in \mathbb{N}_{>0}$.*

Proof. If no messages from nodes in V_g are received during $[t_0, t_0 + (\vartheta^2 + ?\vartheta)\mathcal{S} + \vartheta d]$, the claim vacuously holds. Hence, assume that such a message is received, and let $t_r \in [t_0, t_0 + (\vartheta^2 + ?\vartheta)\mathcal{S} + \vartheta d]$ be the minimal such time. Denote by t_s the time it was sent and by $v \in V_g$ its sender. Consider another message sent by $w \in V_g$ at time t'_s, which is received at time t'_r.

First, we show that both messages must belong to the same pulse k. Assuming towards contradiction that this is not the case, denote by k and k', $k \neq k'$, the

pulse numbers of the messages by v and w, respectively. We get that

delays in $[d-u, d]$

$$
\begin{aligned}
|t_r - t_r'| &\geq |t_s - t_s'| + u \\
&= |H_v^{-1}(H_v(p_{v,k}) + \vartheta\mathcal{S}) - H_w^{-1}(H_w(p_{w,k'}) + \vartheta\mathcal{S})| - u \\
&\geq |p_{v,k} - p_{w,k'}| - (\vartheta - 1)\mathcal{S} - u && 1 \leq \tfrac{dH}{dt} \leq \vartheta \\
&\geq P_{\min} - (\vartheta - 1)\mathcal{S} - u && \text{Theorem 12.7} \\
&= \frac{T - (\vartheta^2 + 1)\mathcal{S}}{\vartheta} - u && \text{Theorem 12.7} \\
&\geq 2(\vartheta^2 + \vartheta)\mathcal{S} + \vartheta d, && \text{(12.3),(12.4)}
\end{aligned}
$$

contradicting the assumption that $t_r, t_r' \in [t_0, t_0 + 2(\vartheta^2+\vartheta)\mathcal{S} + \vartheta d]$. Therefore, both messages belong to the same pulse $k \in \mathbb{N}_{>0}$ and by Lemma 12.10 it follows that $|t_r - t_r'| \leq \vartheta\mathcal{S} + u$. □

Using this statement, we can now establish that once a "fresh" loop iteration is started, a recovering node will establish sufficient synchrony to "catch" all messages corresponding to a subsequent pulse.

Lemma 12.11. *Suppose that the prerequisites of Theorem 12.7 hold and let $v \in V \setminus V_g$ be correct at times $t \geq p_v \in \mathbb{R}_{\geq 0}$, where it starts a new iteration of the loop at time p_v. Then there is $k \in \mathbb{N}_{>0}$ such that v will receive all messages sent by nodes in V_g in their k-th iteration of the loop after starting its next loop iteration, but before evaluating the if-condition.*

Proof. First consider the case that in its loop iteration starting at time p_v, node v executes the if-block. Thus, it receives at least $n - f > 2f$ messages from distinct senders during $[p_v, \tau_v]$, where $\tau_v := H_v^{-1}(p_v + 2(\vartheta^2 + \vartheta)\mathcal{S} + \vartheta d)$ is the time when v evaluates the if-condition. Consider the ordered set of more than $2f$ local reception times, up to f which may correspond to faulty nodes. Observe that, no matter what the latter f times are, the median of the set lies within the interval spanned by the reception times from correct nodes. Therefore, replacing any missing reception times by the median reception time results in at least $n - f$ values from that range.

Recall that by Lemma 12.10, all messages from correct nodes belong to the same pulse, say pulse $k - 1 \in \mathbb{N}_{>0}$. Denoting by $t_{r,w}$ the reception time of the

message sent by node $w \in V_g$ for pulse $k-1$, we conclude that

$$\begin{aligned}
&\min_{w \in V_g} \{H_v(t_{r,w})\} - h_v(\tau_v) - d + u - \mathcal{S} \\
&\leq U_v^{(f+1)}(\tau_v) \\
&\leq U_v^{(n-f)}(\tau_v) \\
&\leq \max_{w \in V_g} \{H_v(t_{r,w})\} - h_v(\tau_v) - d + u - \mathcal{S}.
\end{aligned}$$

In particular,

$$\begin{aligned}
\frac{U_v^{(f+1)}(\tau_v) + U_v^{(n-f)}(\tau_v)}{2} &= \Delta_v(\tau_v) \\
&\leq H_v(\tau_v) - h_v(\tau_v) - d + u - \mathcal{S} \\
&< H_v(\tau_v) - h_v(\tau_v) + 3\mathcal{S},
\end{aligned}$$

implying that the loop iteration ends once the local time reaches or exceeds $h_v(\tau_v) + \Delta_v(\tau_v) + T$. Since the above also shows that

$$\begin{aligned}
h_v(\tau_v) + \Delta_v(\tau_v) + T &\geq H_v(p_v) - d + u - \mathcal{S} + T \\
&> H_v(p_v) + 2(\vartheta^2 + \vartheta)\mathcal{S} + \vartheta d \qquad \text{(12.3),(12.4)} \\
&= H_v(\tau_v),
\end{aligned}$$

the loop iteration ends exactly at time $p'_v := H_v^{-1}(h_v(\tau_v) + \Delta_v(\tau_v) + T)$.

Now consider a node $x \in V_g$. Since the preconditions of the lemma apply also to node x and pulse time $p_{x,k-1}$, we can repeat our reasoning to show that $p_{x,k} = H_x^{-1}(h_x(\tau_{x,k-1}) + \Delta_x(\tau_{x,k-1}) + T)$ with analogous bounds on $\Delta_x(\tau_{x,k-1})$. Denote by $t'_{r,w}$ the reception time at node x corresponding to $t_{r,w}$. By Lemma 12.10, all reception times for pulse messages from nodes in V_g lie

within a time interval of length $\vartheta\mathcal{S} + u$, yielding that

$$\begin{aligned}
p'_v - p_{w,k} &= H_v^{-1}(h_v(\tau_v) + \Delta_v(\tau_v) + T) - H_x^{-1}(h_x(\tau_{x,k-1}) + \Delta_x(\tau_{x,k-1}) + T) \\
&\le H_v^{-1}\left(\max_{w\in V_g}\{H_v(t_{r,w})\} - d + u - \mathcal{S} + T\right) \\
&\quad - H_x^{-1}\left(\min_{w\in V_g}\{H_x(t'_{r,w})\} - d + u - \mathcal{S} + T\right) \\
&\le H_v^{-1}\left(\max_{w\in V_g}\{H_v(t_{r,w})\}\right) - H_x^{-1}\left(\min_{w\in V_g}\{H_x(t'_{r,w})\}\right) \\
&\quad + (\vartheta - 1)|-d + u - \mathcal{S} + T| && 1 \le \tfrac{dH}{dt} \le \vartheta \\
&= \max_{w\in V_g}\{t_{r,w}\} - \min_{w\in V_g}\{t'_{r,w}\} + (\vartheta - 1)|T - d + u - \mathcal{S}| && \tfrac{dH}{dt} \ge 0 \\
&\le \vartheta\mathcal{S} + u + (\vartheta - 1)|T - d + u - \mathcal{S}| && \text{Lemma 12.10} \\
&\le \vartheta\mathcal{S} + u + (\vartheta - 1)T && d \ge u,\ \mathcal{S} > 0 \\
&< 2\mathcal{S}. && (12.4)
\end{aligned}$$

Symetrically and analogously, the same bound can be derived for $p_{w,k} - p'_v$.

In order words, v takes part in the k-th pulse of V_g, albeit with skew (smaller than) $2\mathcal{S}$. Observing that the waiting times in Line 5 and **??** match those of Algorithm 11 for skew $\mathcal{S}' = 2\mathcal{S}$, analogously to Lemmas 12.5 and 10.7 it follows that v will receive the messages for pulse k from nodes in V_g within the stated time bounds. □

Once this coarse, but sufficiently precise synchronization is established, the node will execute subsequent loop iterations correctly, in the sense that it receives all messages corresponding to the respective pulse in a timely fashion. This in turn implies that tight synchronization is (re-)established.

Corollary 12.12. *Suppose that the prerequisites of Theorem 12.7 hold and let $v \in V \setminus V_g$ be correct at times $t \ge p_v \in \mathbb{R}_{\ge 0}$, where it starts a new iteration of the loop at time p_v. Then there is $k \in \mathbb{N}_{>0}$ such that the following holds. Denoting by $p_{v,k}$ its second next pulse, and enumerate subsequent pulses accordingly (i.e., $p_{v,k+1}, p_{v,k+2}, \ldots$), it holds for all $k' \ge k$ that*

$$\max_{w\in V_g}\{|p_{v,k'} - p_{w,k'}|\} \le \mathcal{S}.$$

Proof. By Lemma 12.11, in its next loop iteration v will receive all messages from nodes in V_g for some round $k-1 \in \mathbb{N}_{>0}$ before evaluating the if-condition. Thus, analogously to nodes in V_g, we can perform the induction step as in Lemma 12.5 (where Lemma 12.6 does not require v to meet a skew bound of

$\mathcal{S}$). Accordingly, v will generate its next pulse, and by induction all subsequent ones, with skew $\mathcal{S}$ relative to the nodes of V_g. □

We are now ready to prove Theorem 12.13. Essentially, the assumptions of the theorem provide enough time for recovering nodes to (re-)synchronize before too many previously correct nodes may fail. We leverage this to "stitch together" the pulses generated by (currently) correct and synchronized nodes, which in turn provide the time reference for recovering nodes to synchronize to.

Theorem 12.13. *Suppose that* $11 - 10\vartheta^2 > 0$*, fix any*

$$T \geq \frac{2\vartheta^2(2\vartheta - 1)u + \vartheta^3(4\vartheta - 3)d}{19 - 18\vartheta^2} \in \mathcal{O}(d), \tag{12.3}$$

and set

$$\mathcal{S} \coloneqq \frac{2(2\vartheta - 1)(u + (\vartheta - 1)d) + 2(\vartheta - 1)T}{\vartheta(9 - 8\vartheta^2)} \in \mathcal{O}(u + (\vartheta - 1)T). \tag{12.4}$$

Then there is $T_0 \in O(T)$ *such that the following holds. If for each time* $t \in \mathbb{R}_{\geq 0}$*, the set* $V_g(t) \subseteq V$ *of all nodes that are correct during* $[\max\{t - T_0, 0\}, t]$ *has size at least* $n - f$ *and* $\max_{v \in V_g(T_0)}\{H_v(0)\} \leq \mathcal{S}$*, then Algorithm 11 lets these nodes generate pulses with skew at most* $\mathcal{S}$*,* $P_{\min} \geq (T - (\vartheta + 1)\mathcal{S})/\vartheta$*, and* $P_{\max} \leq T + 3\mathcal{S}$*. More precisely, we can inductively label for all* $t \in \mathbb{R}_{\geq 0}$ *and all pulses by nodes* $v \in V_g(t)$ *by round numbers* $k \in \mathbb{N}_{>0}$ *such that (i) pulses with the same round number are at most* $\mathcal{S}$ *time apart, (ii) pulses with consecutive round numbers are at least* $P_{\min}$ *time apart, and (iii) pulses with consecutive round numbers are at most* $P_{\max}$ *time apart.*

Proof. Choose $T_0 = CT$ for a sufficiently large constant C. The proof is by induction over the time up to which we have performed the assignment of round labels. As additional invariant for the induction, we add that for each time $t \geq T_0$, there is some set of at least $n - f$ nodes that have been generating (suitably labelled) pulses during $[t - T_0/2, t]$. To anchor the induction at time T_0, note that $V_g(T_0)$ is correct until time T_0, has size at least $n - f$, and the hardware clocks of nodes in $V_g(T_0)$ are initialized with values between 0 and $\mathcal{S}$. Observe that we can extend the execution prefix until time T_0 in a way such that the nodes in $V_g(T_0)$ stay correct forever without making any changes before time T_0. Thus, we can apply Theorem 12.7 to show that the nodes in $V_g(T_0)$ generate pulses satisfying the requirements until time T_0 also in the original execution, as there is no difference between the two executions before time T_0.

For the induction step, assume that we have shown all claims until some time t, i.e., all pulses generated at times $t' \leq t$ by nodes in $V_g(t')$ have been

assigned labels satisfying the requirements. We will show that can extend this assignment consistenly until time $t + T$. Consider any $t' \in (t, t + T]$. By the induction hypothesis, the nodes in $V_g(t' - T_0/2)$ have generated pulses satisfying the constraints during $[t' - T_0, t' - T/2]$. Let $k \in \mathbb{N}_{>0}$ be the minimal round label such that all pulses with label k of nodes in $V_g(t' - T_0)$ fall into this interval. Thus, the first such pulse is generated at a time $p_{v,k} \leq t' - T_0 + P_{\max}$.

We now argue that we can apply Theorem 12.7 to $V_g(t' - T_0/2)$, even though it is not properly initialized for this set at this time. To this end, we construct an execution that results in the exact same behavior at all nodes, but with shifted times to match the requirements of the theorem. We "cut off" the prefix of the execution before $p_{v,k}$, also dropping all sending events of messages sent earlier. Then we shift time, i.e., map time $t \in [p_{v,k}, \infty)$ to time $t - p_{v,k}$, shifting all clock functions and event times with it (e.g., the new hardware clock value at time t is the old one at time $t + p_{v,k}$, etc.). Finally, observe that since all nodes in $V_g(t' - T_0/2)$ now generate a pulse by the (new) time $\mathcal{S}$, we can change their hardware clock functions such that (i) the clocks have values between 0 and $\mathcal{S}$ at time 0, (ii) they have value $\mathcal{S}$ when generating the first pulse of the node, (iii) they have derivative 1 until the first pulse of the node, and (iv) their derivative is identical afterwards. Since the algorithm only uses the hardware clocks to measure time differences relative to the start of a loop iteration, this change does not affect the bevavior of nodes.

Again, observe that there are executions identical to the one we created up to time $T_0/2 - (p_{v,k} - (t' - T_0)) \geq T_0/2 - P_{\max}$ (formerly time $t' - T_0/2$) in which $V_g(t' - T_0/2)$ is correct at all times. Thus, our manipulations result in an execution for which Theorem 12.7 shows that the nodes in $V_g(t' - T_0/2)$ solve pulse synchronization with the stated parameters until time $T_0/2 - (p_{v,k} - (t' - T_0)) \geq T_0/2 - P_{\max} > T_0/2 - 2T$. Now consider a node $w \in V_g(t')$. Since in the original execution, it is correct during $[t' - T_0, t']$, in the new execution it is correct during $[0, T_0/2 - 2T]$. Because it could also be correct in the future, we can apply Lemma 12.9 to show that by time $3T$ (which, as C is sufficiently large, is smaller than $T_0/2 - 2T$), it will have started a new loop iteration. By Corollary 12.12, it will start generating pulses synchronized with $V_g(t' - T_0/2)$ two loop iterations later, which by Lemma 12.10 happens no later than time $9T < T_0/2 - 2T$.

Translating this back to the original execution, we conclude that w generates pulses synchronized with $V_g(t' - T_0/2)$ during $[t' - T_0 + 11T, t' - T_0/2]$. Thus, we can consistenly assign pulses of w during this period labels $k \in \mathbb{N}_{>0}$ satisfying the bounds. Because $P_{\max} < 2T$ and T_0 is sufficiently large, this applies to a non-zero number of pulses. We now inductively repeat this procedure

with later pulses, by gradually using larger times, e.g., $t' - T_0/2 + iP_{\min}$ for $i = 1, 2, \ldots$. Note that because $|V_g(t'')| \geq n - f$ for all $t'' \in \mathbb{R}_{\geq 0}$, any two such sets intersect, guaranteeing that the pulses satisfy not only the skew, but also the period bounds. This process allows us to assign consistent pulse numbers for pulses of w until time t', which also establishes that all of $V_g(t')$ generates suitably labelled pulses during $[t' - T_0/2, t']$. This completes the induction step and hence the proof. □

12.3 Self-stabilizing gradient clock synchronization

It turns out that Algorithm 5 has inherent self-stabilization properties, which originate in its goal to keep the local skew small. Since it continually "spreads out" large local skews over larger distances, this means that it also (re-)establishes a small local skew and, similarly, a small global skew, if it is ever lost. However, proving this requires to revisit the algorithm, to check that incorrect state does not break the mechanism minimizing local skews. Subsequently, we need to prove that this indeed achieves the desired stabilization from a global perspective.

Algorithm 13 Pseudocode for $v \in V$ keeping track of the logical clock of its neighbor w.

```
if v just woke up, i.e., t = 0 then
    h ← getH()
    ℓ̃_w ← getL()                          ▷ default to own clock value
end if
if getH() = kT for some k ∈ ℕ then
    send ⟨getL()⟩ to w
end if
if received ⟨ℓ⟩ from w then
    ℓ̃_w ← ℓ + d − u                       ▷ take message delay into account
    h ← getH()
end if
procedure getL(w)                                  ▷ returns L̃_v^w(t)
    return ℓ̃_w + (getH() − h)/ϑ            ▷ own clock might be faster than w's
end procedure
```

Before performing these tasks, let us quickly recap how the algorithm operates.

- The algorithm operates on an (arbitrary) undirected connected network $G = (V, E)$ of (unknown) diameter D.

- Each node $v \in V$ maintains for each incident edge $\{v, w\} \in E$ an estimate $\tilde{L}_w^v(t)$ of the logical clock $L_w(t)$ of its neighbor w of error δ, i.e.,

$$L_w(t) \geq \tilde{L}_w(t) > L_w(t) - \delta.$$

  This is implemented by Algorithm 13, which achieves $\delta = (\vartheta(1+\mu) - 1/\vartheta)(T+u) + \vartheta(u+\mu d)$ so long as logical clock rates satisfy the constraints imposed by the gradient clock synchronization algorithm, cf. Lemma 8.31.
- The main algorithm, whose code is shown in Algorithm 14, then lets each node continuously evaluate the *slow mode trigger* **ST**, which is satisfied at $v \in V$ and time t if and only if there is $s \in \mathbb{N}_{>0}$ such that:

ST-1 $\exists \{v, x\} \in E \colon L_v(t) - \tilde{L}_x(t) \geq (2s-1)\kappa$,

ST-2 $\forall \{v, y\} \in E \colon \tilde{L}_y(t) - L_v(t) \leq (2s-1)\kappa$.

  For κ we can choose any value of at least δ, so we assume $\kappa = \delta$ in the following. If the trigger holds, the node increases its logical clock at rate

Algorithm 14 GCS algorithm

```
if v just woke up, i.e., t = 0 then
    ℓ ← getH()
    h ← getH()
    if ST then
        r ← 1                                   ▷ v is in slow mode
    else
        r ← 1 + μ                               ▷ v is in fast mode
    end if
end if
if ST stops to hold then
    ℓ ← getL()                                  ▷ always keep track of clock progress
    h ← getH()
    r ← 1 + μ                                   ▷ v is in fast mode
end if
if ST starts to hold then
    ℓ ← getL()                                  ▷ always keep track of clock progress
    h ← getH()
    r ← 1                                       ▷ v is in slow mode
end if
procedure getL()                                ▷ returns L_v(t)
    return ℓ + r(getH() − h)                    ▷ logical clock increases at rate r dH_v/dt
end procedure
```

$\frac{dH_v}{dt}$ at time t, and otherwise at rate $(1+\mu)\frac{dH_v}{dt}$. This is implemented by the procedure getL() together with memorizing logical and hardware clock values whenever **ST** starts or stops to hold.

12.3.1 Algorithm 14 is self-stabilizing

Algorithm 14 assumes that there are no faults. If any node fails, this entails that no guarantees on the behavior of the system are implied from its analysis. Our goal now is to establish such guarantees in face of transient faults. However, we do not need to (nor will we be able to) show that skew is bounded *while* faults are ongoing. Hence, our analysis commences only after transient faults cease, and we only will make statements about times before the next transient fault. By allowing for d additional time to pass "before" we start our analysis, we can also remove all messages from the system that have not been sent according to the instructions of the algorithm. Therefore, w.l.o.g., we may assume that our execution starts at time 0 without any faults, *but* the state machines of nodes are in arbitrary states. In particular, any instructions performed upon initialization, like the first instruction block of Algorithm 14, are irrelevant and could even be discarded from the code altogether.

For the algorithm at hand, given by Algorithm 13 and Algorithm 14, the state of the algorithm at node $v \in V$ consists of (i) the state of the local variables (h and $\tilde{\ell}_w$ in Algorithm 13 and ℓ, h, and r in Algorithm 14) and (ii) the state of the hardware clock. Naturally, an implementation needs to deal with further details, such as how the hardware clocks are accessed and how **ST** is evaluated, which is likely to introduce further state; we assume here that all these operations are implemented in a self-stabilizing manner, too, and our analysis begins only after these subroutines stabilized.

E12.10 The "continuous" evaluation of **ST** is likely to be implemented by some kind of loop that keeps checking the condition, and any implementation will incur some computational delay. In our model, we can easily hide this issue by making this delay part of d. However, the evaluation still needs to be self-stabilizing. Conceive a simple implementation, assuming that Algorithm 13 is already operating correctly, and check whether it is self-stabilizing.

Before arguing about convergence, we need to make sure that the code actually operates as intended after transient faults. In most cases, this requires adjustments avoiding deadlocks, etc. By sheer coincidence, this particular implementation happens to be self-stabilizing. However, lacking even elementary sanity checks for variables, it is instructive to take note of the rather unexpected behavior that could surface in the aftermath of transient faults. Recall that $L_v(t)$

is defined as the return value of the getL() procedure when (hypothetically) being called at time t.

E12.11 Under normal circumstances, Algorithm 14 ensures logical clock rates of either $\frac{dH_v}{dt}$ or $(1+\mu)\frac{dH_v}{dt}$ at all times. Is this still the case when state variables can be set to arbitrary values?

E12.12 Specify the type of each state variable, i.e., which values they may attain. Assuming implicit type checks (resulting in resetting the variable to a valid default value in case of violation), show that the logical clock satisfies the above invariant.

Observation 12.14. *Assuming suitable typing of variables, Algorithm 14 satisfies that* $\frac{dH_v}{dt} \leq \frac{dL_v}{dt} \leq (1+\mu)\frac{dH_v}{dt}$.

In the following, we assume that this invariant holds. With the logical clocks behaving as expected, the estimate algorithm turns out to be self-stabilizing by design; since it regularly updates the estimates, all its state information is transient and therefore any incorrect state is quickly removed from the system.

Corollary 12.15 (of Lemma 8.31)**.** *Suppose that Algorithm 13 is executed with* $T = \Theta(d)$ *and that* $2(\vartheta - 1) \leq \mu = O(u/d)$. *Then at times* $t \geq t_0 \in \Theta(d)$, *it computes estimates satisfying* $\delta = O(u)$.

Proof. The proof of Lemma 8.31 applies verbatim, showing that

Lemma 8.31
$$\delta = \left(\vartheta(1+\mu) - \frac{1}{\vartheta}\right)(T+u) + \vartheta(u+\mu d)$$
$T = \Theta(d)$
$$= O\left(\vartheta(1+\mu) - \frac{1}{\vartheta}\right)(d+u) + \vartheta(u+\mu d)$$
$u \leq d$, $\mu d = O(u)$
$$= O\left(\vartheta - \frac{1}{\vartheta}\right)d + \vartheta u$$
$1 - 1/\vartheta < \vartheta - 1$
$$= O((\vartheta - 1)d + \vartheta u)$$
$\vartheta - 1 = O(u/d) \subseteq O(1)$
$$= O(u).$$
□

Thus, this part of the algorithm is self-stabilizing with stabilization time $O(d)$. Since Algorithm 14 continuously checks whether **ST** holds or not, it will immediately (read: with whatever delay the implementation of this check incurs) start responding to skews as it is designed to. Accordingly, from here on, it remains to show that the potentially large clock skews resulting from arbitrary changes to the logical clock values are reduced.

Our first step is to note that two key lemmas from the analysis of the algorithm readily apply. These are concerned with the potential functions that lie at the heart of the analysis (cf. Definition 8.19): For each $v \in V$, $s \in \mathbb{N}_{>0}$, and

$t \in \mathbb{R}_{\geq 0}$,

$$\Psi_w^s(t) = \max_{v \in V} \{L_v(t) - L_w(t) - (2s-1)\kappa \operatorname{dist}(v,w)\},$$

where $\operatorname{dist}(v,w)$ denotes the distance between v and w in G. The following two lemmas apply to any system state, so long as the algorithm has clock estimates of error $\delta \leq \kappa$ and behaves as intended. In light of the above reasoning and Corollary 12.15, they can be applied after an initial grace period of $\Theta(d)$. For simplicity, in the following we will not explicitly add this to all statements and pretend that it holds starting from time 0; however, we will have to bear this in mind when arguing about stabilization time later.

The first of the two lemmas shows that the potentials cannot increase quickly.

Lemma 8.21 (Wait-up Lemma)**.** *Suppose $w \in V$ satisfies $\Psi_w^s(t) > 0$ for all $t \in (t_0, t_1]$. Then*

$$\Psi_w^s(t_1) \leq \Psi_w^s(t_0) - (L_w(t_1) - L_w(t_0)) + \vartheta(t_1 - t_0).$$

The second lemma shows that nodes that are behind on level $s \in \mathbb{N}_{>0}$ "catch up" with a delay that is proportional to the current value of the potential from the previous level $s-1$ (or the global skew, in case $s = 1$).

Lemma 8.25 (Catch-up Lemma)**.** *Let $s \in \mathbb{N}_{>0}$ and t_0, t_1 be times. Suppose that*

$$t_1 \geq \begin{cases} t_0 + \frac{\mathcal{G}(t_0)}{\mu} & \text{if } s = 1 \\ t_0 + \frac{\Psi^{s-1}(t_0)}{\mu} & \text{otherwise.} \end{cases}$$

Then, for any $w \in V$,

$$L_w(t_1) - L_w(t_0) \geq t_1 - t_0 + \Psi_w^s(t_0).$$

Together, these lemmas imply not only that large skews cannot build up, but also that they decrease at an amortized rate of roughly $\mu - (\vartheta - 1)$. Assuming that $\mu \geq 2(\vartheta - 1)$, we can use this to show that the initial global skew is reduced to an asymptotically optimal amount at amortized rate $\Omega(\mu)$.

Lemma 12.16. *If $\mu \geq 2(\vartheta - 1)$, there is $T_0 = O(\mathcal{G}(0)/\mu)$ such that for all $t \geq T_0$ it holds that $\mathcal{G}(t) = O(\kappa D)$. If $\mathcal{G}(0) = O(\kappa D)$, then $T_0 = 0$.*

Proof. Set $\sigma := \mu/(\vartheta - 1) \geq 2$, $t^{(0)} := 0$, and $\mathcal{G}^{(0)} := \mathcal{G}(0)$. We claim that for $i \in \mathbb{N}_{>0}$, at time $t^{(i)} := t^{(i-1)} + \mathcal{G}^{(i-1)}/\mu$ it holds that

$$\mathcal{G}(t^{(i)}) \leq \mathcal{G}^{(i)} := \kappa D + \frac{\mathcal{G}^{(i-1)}}{\sigma}, \tag{12.5}$$

which we prove by induction on i.

For $i = 0$, the claim holds by induction, so we only need to perform the step from $i - 1 \in \mathbb{N}$ to i. Assume towards a contradiction that $\mathcal{G}(t^{(i)}) > \mathcal{G}^{(i)}$. Thus, there are $v, w \in V$ such that

$$L_v(t^{(i)}) - L_w(t^{(i)}) > \mathcal{G}^{(i)}, \tag{12.6}$$

implying that

$$\begin{aligned}
\Psi^1_w(t^{(i)}) &\geq L_v(t^{(i)}) - L_w(t^{(i)}) - \kappa \operatorname{dist}(v, w) \\
&\qquad \text{(12.6)} \\
&> \mathcal{G}^{(i)} - \kappa \operatorname{dist}(v, w) \\
&\qquad G \text{ has diam. } D \\
&\geq \mathcal{G}^{(i)} - \kappa D \\
&\qquad \text{(12.5)}. \\
&= \frac{\mathcal{G}^{(i-1)}}{\sigma}
\end{aligned}$$

We apply Lemmas 8.21 and 8.25 with $t_0 = t^{(i-1)}$ and $t_1 = t^{(i)}$, yielding the contradiction

$$\begin{aligned}
\Psi^1_w(t^{(i)}) &\leq \Psi^1_w(t^{(i-1)}) - (L_w(t^{(i)}) - L_w(t^{(i-1)})) + \vartheta(t^{(i)} - t^{(i-1)}) && \text{Lemma 8.21} \\
&\leq (\vartheta - 1)(t^{(i)} - t^{(i-1)}) && \text{Lemma 8.25} \\
&= \frac{(\vartheta - 1)\mathcal{G}^{(i-1)}}{\mu} && \text{def. of } t^{(i)} \\
&= \frac{\mathcal{G}^{(i-1)}}{\sigma}. && \text{def. of } \sigma
\end{aligned}$$

Therefore, the step succeeds, completing the induction.

Moreover, for all $i \in \mathbb{N}$ and $t \in [t^{(i-1)}, t^{(i)}]$, we claim that

$$\mathcal{G}(t) \leq \left(1 + \frac{1}{\sigma}\right) \mathcal{G}^{(i-1)},$$

which due to $\sigma \geq 2$ is bounded by $3\mathcal{G}^{(i-1)}/2$. To see this, again assume towards contradiction that the claim is violated, for some time $t \in [t^{(i-1)}, t^{(i)}]$. Analogously to above, we get that there is $w \in V$ such that

$$\begin{aligned}
\Psi^1_w(t) &> \left(1 + \frac{1}{\sigma}\right) \mathcal{G}^{(i-1)} - \kappa D \\
&\qquad \text{def. of } \sigma \text{ and } \mathcal{G}^{(i-1)} \\
&\geq \mathcal{G}^{(i-1)} - \kappa D + (\vartheta - 1)(t - t^{(i-1)}).
\end{aligned}$$

However,

$$\begin{aligned}
\Psi^1_w(t) &\le \Psi^1_w(t^{(i-1)}) - (L_w(t) - L_w(t^{(i-1)})) + \vartheta(t^{(i)} - t^{(i-1)}) && \text{Lemma 8.21}\\
&\le \Psi^1_w(t^{(i-1)}) + (\vartheta - 1)(t - t^{(i-1)}) && \tfrac{dL}{dt} \ge \tfrac{dH}{dt} \ge 1\\
&\le \max_{x,y \in V} \left\{ L_x(t^{(i-1)}) - L_y(t^{(i-1)}) - \kappa \operatorname{dist}(x,y) \right\} + (\vartheta - 1)(t - t^{(i-1)}) && \text{def. of } \Psi^1_w(t^{(i-1)})\\
&\le \mathcal{G}(t^{(i-1)}) - \kappa D + (\vartheta - 1)(t - t^{(i-1)}) && G \text{ has diam. } D\\
&\le \mathcal{G}^{(i-1)} - \kappa D + (\vartheta - 1)(t - t^{(i-1)}). && (12.5)
\end{aligned}$$

In summary, the claim of the lemma follows if we can show that there is some $i \in \mathbb{N}_{>0}$ such that $t^{(i)} = O(\mathcal{G}(0)/\mu)$ (or 0, if $\mathcal{G}(0) = O(\kappa D)$) and $\mathcal{G}^{(i)} = O(\kappa D)$. If $\mathcal{G}(0) < 2\kappa D$, we can pick $i = 0$, so assume that $\mathcal{G}(0) \ge 2\kappa D$. We fix $i_0 := \lceil \log_\sigma \mathcal{G}^{(0)}/(\kappa D) \rceil$. By induction on i, we get that

$$\begin{aligned}
\mathcal{G}^{(i_0)} &= \frac{\mathcal{G}^{(0)}}{\sigma^{i_0}} + \sum_{i=0}^{i_0-1} \frac{\kappa D}{\sigma^i}\\
&\le \kappa D + \sum_{i=0}^{\infty} \frac{\kappa D}{\sigma^i} && \text{def. of } i_0\\
&\le 3\kappa D. && \sigma \ge 2
\end{aligned}$$

Similarly,

$$
\begin{aligned}
\text{adding 0, } t^{(0)} = 0 \qquad t^{(i_0)} &= \sum_{i=1}^{i_0} t^{(i)} - t^{(i-1)} \\
\text{def. of } t^{(i)} \qquad &= \sum_{i=1}^{i_0} \frac{\mathcal{G}^{(i-1)}}{\mu} \\
\text{def. of } \mathcal{G}^{(i-1)} \qquad &= \frac{\mathcal{G}^{(0)}}{\mu} + \sum_{i=2}^{i_0} \left(\frac{\kappa D}{\mu} + \frac{\mathcal{G}^{(i-2)}}{\mu\sigma} \right) \\
&< \frac{\mathcal{G}^{(0)}}{\mu} + \frac{(i_0 - 1)\kappa D}{\mu} + \frac{1}{\sigma} \sum_{i=1}^{i_0} \frac{\mathcal{G}^{(i-1)}}{\mu} \\
\sigma \geq 2 \qquad &\leq \frac{\mathcal{G}^{(0)}}{\mu} + \frac{(i_0 - 1)\kappa D}{\mu} + \frac{t^{(i_0)}}{2} \\
\text{def. of } i_0 \qquad &\leq \frac{\mathcal{G}^{(0)}}{\mu} + \frac{\kappa D \log_\sigma \mathcal{G}^{(0)}/(\kappa D)}{\mu} + \frac{t^{(i_0)}}{2} \\
\log(y/x) < y/x \text{ for } y \geq x \qquad &< \frac{2\mathcal{G}^{(0)}}{\mu} + \frac{t^{(i_0)}}{2} \\
\text{def. of } \mathcal{G}^{(0)} \qquad &\leq \frac{2\mathcal{G}(0)}{\mu} + \frac{t^{(i_0)}}{2},
\end{aligned}
$$

which can be rearranged to show that $t^{(i_0)} \leq 4\mathcal{G}(0)$. □

This bounds the global skew of the algorithm, and the bound on the local skew follows as in Chapter 8.

Lemma 12.17. *There is $T_0 = O((\mathcal{G}(0) + \kappa D)/\mu)$ such that for all $s \in \mathbb{N}_{>0}$ and $t \geq T_0$, Algorithm 5 guarantees for each $w \in V$ that $\Psi_w^s(t) = O(\kappa D/\sigma^s)$, where $\sigma = \mu/(\vartheta - 1)$.*

Proof. By Lemma 12.16, there is $T_0' = O(\mathcal{G}(0)/\mu)$ so that $\mathcal{G}(t) = O(\kappa D)$ at times $t \geq T_0'$. The lemma is now shown analogously to Lemma 8.27 for times $t \geq T_0 := T_0' + \mathcal{G}(T_0')/\mu = T_0' + O(\kappa D/\mu) = O((\mathcal{G}(0) + \kappa D)/\mu)$, which circumvents the need for the prerequisite that $H_v(0) - H_w(0) \leq \kappa$ for all $\{v, w\} \in E$ of Lemma 8.27. □

Putting these results together, we see that Algorithm 14 indeed achieves small global and local skews in a self-stabilizing manner.

Theorem 12.18. *Suppose that*

- *Algorithm 13 with $T = d$ is used to compute clock estimates,*
- *$2(\vartheta - 1) \leq \mu = O(u/d)$, and*

- $\kappa = \delta$, *where* δ *is as in Lemma 8.31.*

Then Algorithm 14 is a self-stabilizing solution to the task of guaranteeing that

- $\frac{dH_v}{dt}(t) \leq \frac{dL_v}{dt}(t) \leq (1+\mu)\frac{dH_v}{dt}(t)$ *for all nodes* v *and times* t,
- $\mathcal{G} = O(uD)$, *and*
- $\mathcal{L} = O(u \log_\sigma D)$, *where* $\sigma = \mu/(\vartheta - 1)$.

These properties are established within $O(\mathcal{G}(0)/\mu)$ *time.*

Proof. By Corollary 12.15, after $\Theta(d)$ time estimates of neighbors' clock values with error $\delta = O(u)$ are available to the nodes. From this time on, we can make use of Lemmas 8.21 and 8.25, and by Lemma 12.16 within $O(\mathcal{G}(0)/\mu)$ additional time the global skew has stabilized to $\mathcal{G} = O(\kappa D) = O(\delta D) = O(uD)$.

By Lemma 12.17, after at most $O(\kappa D/\mu)$ more time, we have for $s := \lceil \log_\sigma(\mathcal{G}/\kappa) \rceil$ and any $\{v, w\} \in E$ that

$$\begin{aligned} L_v(t) - L_w(t) - (2s-1)\kappa &= L_v(t) - L_w(t) - (2s-1)\kappa \operatorname{dist}(v,w) && \{v,w\} \in E \\ &\leq \Psi^s(t) && \text{def. of } \Psi^s_w \\ &\leq \frac{\mathcal{G}}{\sigma^s} && \text{Lemma 12.17} \\ &\leq \kappa. && s \geq \log_\sigma(\mathcal{G}/\kappa) \end{aligned}$$

By exchanging the roles of v and w, we analogously obtain that $L_w(t) - L_v(t) - (2s-1)\kappa \leq \kappa$. Rearranging these inequalities, we conclude

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

for sufficiently large times t, i.e.,

$$\mathcal{L}(t) = O\left(\kappa \log_\sigma \frac{\mathcal{G}}{\kappa}\right) = O(\kappa \log_\sigma D) = O(u \log_\sigma D).$$

Since Algorithm 14 satisfies the bounds on clock rates at all times, this completes the proof. □

Carefully note that Theorem 12.18 does *not* offer any bound on the stabilization time. This means that we do not get any actual guarantee, regardless of how far in the past the most recent transient fault lies! In addition, as we mentioned earlier, a large global skew renders implementation of Algorithm 13 impractical. Hence, a different mechanism for (re-)establishing small global skew is needed.

12.3.2 Self-stabilizing GCS in asymptotically optimal time

Our approach is simple: We add a simple subroutine that runs in the background, detects excessive global skew, and in this case triggers a global reset. If no reset occurs during the stabilization process, the global skew must have been small to begin with, and stabilization is quick. If the global skew is too large, a reset is guaranteed to occur within $O(dD)$ time. By making sure that a reset (i) reduces the global skew to $O(uD)$ and (ii) leaving enough slack that after a reset, there can be no other reset until the GCS algorithm had enough time to stabilize, we prevent that a sequence of repeated resets interferes with stabilization.

Our approach makes use of a breadth-first-search (BFS) tree of known depth $\tilde{D} \leq D$. If G is known at design time, we can simply hardcode this structure into the algorithm, i.e., the FSMs of the nodes (and their circuit implementations) are designed to take into account, e.g., which neighbor is the parent. This way, the BFS tree is not part of the state and therefore cannot be affected by transient faults. However, this also means that the algorithm cannot handle unknown or changing network topology—and one of the nice properties one gets for free from self-stabilization is that such algorithms automatically adjust if the topology changes! We will discuss afterwards how to remove the assumption of a pre-defined BFS tree, by employing a self-stabilizing BFS tree construction as subroutine.

The birds eye view of the algorithm, whose pseudocode is given in Algorithm 15, is as follows:

- The BFS tree is used for regular broadcasts of the root, which on reception allow each node to estimate their logical clock offset to the root with an error of at most

$$(u + \vartheta(1 + \mu) - 1)d)\tilde{D} \leq (u + \vartheta(1 + \mu) - 1)d)D = O(uD),$$

  where the last step uses the assumption that $2(\vartheta - 1) \leq \mu = O(u/d)$.
- Denote by $\mathcal{G}$ the target bound on the global skew after stabilization (as D is known, because G is known, this bound can be computed; we can also use $\tilde{D}$ as a factor-2 approximation to derive an upper bound). If a node estimates a skew of more than $2\mathcal{G} + (u + \vartheta(1 + \mu) - 1)d)\tilde{D}$ to the root, it alerts the root (with the help of its ancestors in the tree).
- If the root is alerted of large skew, it broadcasts a reset message through the tree, causing nodes in distane dist from the root to set their logical clocks to $(d - u)$ dist.

Algorithm 15 Mechanism to reduce global skew to $O(\mathcal{G} + uD)$ after transient faults, for a given target global skew $\mathcal{G}$. The procedure setL(ℓ') is specific to Algorithm 14 and resets its variables ℓ and h to set the logical clock to ℓ'. The logical clocks are affected by the algorithm only if the global skew exceeds $2\mathcal{G}$ or a fault directly triggers a reset. The algorithm assumes a pre-defined BFS tree of depth $\tilde{D}$ to be given.

```
1: if root then
2:     while true do
3:         send ⟨getL()⟩ to each child
4:         wait for 2ϑdD̃ local time
5:         if received alert message from any child in this loop iteration then
6:             call setL(0)
7:             send ⟨reset⟩ to each child
8:         end if
9:         wait for ϑdD̃ local time
10:    end while
11: end if
12: if received ⟨ℓ′⟩ from parent then
13:    if |ℓ′ + d − u − getL()| ≥ 2𝒢 + (u + ϑ(1 + μ) − 1)d)D̃ then
14:        send ⟨alert⟩ to parent
15:    else
16:        send ⟨ℓ′ + d − u⟩ to each child
17:    end if
18: end if
19: if not root and received ⟨alert⟩ from any child then
20:    send ⟨alert⟩ to parent
21: end if
22: if received ⟨reset⟩ from parent then
23:    call setL((d − u) dist), where dist is the distance to the root
24:    send ⟨reset⟩ to each child
25: end if
26: procedure setL(()ℓ′)          ▷ resets variables ℓ and h of Algorithm 14
27:    ℓ ← ℓ′
28:    h ← getH()
29: end procedure
```

As our first step in formalizing the above intuition, we show that any stale information still present due to transient faults is flushed out in $O(d\tilde{D}) = O(dD)$ time.

Lemma 12.19. *If any node executing Algorithm 15 sends a related message or calls* setL(ℓ') *for any* ℓ' *at a time* $t \geq (2\vartheta+3)d\tilde{D}$*, then this is a result of a message chain initiated by a message* $\langle$getL()$\rangle$ *by the root no later than time* $t-(2\vartheta+3)d\tilde{D}$*.*

Proof. Suppose node $v \in V$ performs any such action at time $t \geq (2\vartheta+3)d\tilde{D}$. Calling setL($\ell'$) at non-root nodes is triggered by receiving a reset message. Such messages are only caused by nodes receiving them from their parent. Tracing the respective message chain back to the root, the root must have called setL(0) no earlier than time $t - d\tilde{D}$. In particular, the root must have received an alert message by time $t-(2\vartheta-1)d\tilde{D}$, since this is the earliest possible time when it could have started its respective loop iteration.

Any alert message received at or after time $t-(2\vartheta-1)d\tilde{D}$ can be backtracked to some node finding that $|\ell + d - u + \text{getL}()| \geq 2\mathcal{G} + (u + \vartheta(1+\mu) - 1)d)\tilde{D}$ on reception of a message $\langle \ell' \rangle$ from a parent, which must have happened no earlier than time $t-(2\vartheta-2)d\tilde{D}$.

Finally, any message $\langle \ell' \rangle$ can be traced back to the root sending a $\langle$getL()$\rangle$ no earlier than time $t-(2\vartheta-3)d\tilde{D}$. Since there are no other message types, this completes the proof. □

Next, we show that the message cascade triggered by the root broadcasting a $\langle$getL()$\rangle$ message results in sufficiently good estimates of clock offsets to the root.

Lemma 12.20. *If the root* r *of the BFS tree sends a* $\langle$getL()$\rangle$ *message at time* t *and* $\frac{dH_v}{dt} \leq \frac{dL_v}{dt} \leq (1+\mu)\frac{dH_v}{dt}$ *for some* $v \in V$ *during* $[t, t+d\tilde{D}]$*, then* v *receives a corresponding* $\langle \ell' - d + u \rangle$ *message at a time* $t' \in [t + \text{dist}(r,v)(d-u), t + \text{dist}(r,v)d]$*, such that*

$$|\ell' - L_v(t') - (L_r(t) - L_v(t))| \leq (\vartheta(1+\mu) - 1)d + u)\tilde{D}.$$

Proof. There is a message chain from the root to v that causes v to receive a message with value $\ell' - d + u = L_r(t) + (\text{dist}(r,v) - 1)(d-u)$ at a time $t' \in [t + \text{dist}(r,v)(d-u), t + \text{dist}(r,v)d]$. We have that

$$\begin{aligned}
L_v(t') - L_v(t) - \text{dist}(r,v)(d-u) &\geq H_v(t') - H_v(t) - \text{dist}(r,v)(d-u) && \frac{dL_v}{dt} \geq \frac{dH_v}{dt} \\
&\geq t' - t - \text{dist}(r,v)(d-u) && \frac{dH_v}{dt} \geq 1 \\
&\geq 0 && t' \geq t + \text{dist}(r,v)(d-u)
\end{aligned}$$

and

$$\begin{aligned}
&L_v(t') - L_v(t) - \operatorname{dist}(r,v)(d-u) \\
&\leq (1+\mu)(H_v(t') - H_v(t)) - \operatorname{dist}(r,v)(d-u) && \tfrac{dL}{dt} \leq \tfrac{(1+\mu)dH}{dt} \\
&\leq \vartheta(1+\mu)(t'-t) - \operatorname{dist}(r,v)(d-u) && \tfrac{dH_v}{dt} \leq \vartheta \\
&\leq \vartheta(1+\mu)\operatorname{dist}(r,v)d - \operatorname{dist}(r,v)(d-u) && t' \leq t + \operatorname{dist}(r,v)d \\
&= (\vartheta(1+\mu)-1)d + u)\operatorname{dist}(r,v) \\
&\leq (\vartheta(1+\mu)-1)d+u)\tilde{D}. && \tilde{D} \text{ is depth of tree}
\end{aligned}$$

Thus

$$\begin{aligned}
|\ell' - L_v(t') - (L_r(t) - L_v(t))| &= |L_v(t') - L_v(t) - \operatorname{dist}(r,v)(d-u)| \\
&\leq (\vartheta(1+\mu)-1)d+u)\tilde{D}. \qquad \square
\end{aligned}$$

Using the previous statements, we can show that a too large global skew will be detected successfully, resulting in the root broadcasting a reset message.

Lemma 12.21. *If*

$$\mathcal{G}(t) \geq 4\mathcal{G} + (4u + (6\vartheta+4)(\vartheta(1+\mu)-1)d)\tilde{D}$$

for a time $t \geq (2\vartheta+3)d\tilde{D}$, then the root r broadcasts a reset message during $[t - d\tilde{D}, t + (5\vartheta+2)d\tilde{D}]$.

Proof. Assume towards a contradiction that the root does not broadcast a reset message during $[t - d\tilde{D}, t + (5\vartheta+2)d\tilde{D}]$. By Lemma 12.19, any node $v \in V$ calling setL(ℓ') for some ℓ' at or after time t is caused by a message chain initiated by the root. Such a message chain would, in particular, require the root to send a reset message at most $d\tilde{D}$ time earlier. Since this by assumption is not the case, it follows that $\frac{dH_v}{(}dt) \leq \frac{dL_v}{dt} \leq (1+\mu)\frac{dH_v}{dt}$ during $[t, t+(5\vartheta+2)d\tilde{D}]$.

Since $\mathcal{G}(t) \geq 4\mathcal{G} + (4u + (6\vartheta+4)(\vartheta(1+\mu)-1)d)\tilde{D}$, there are $v, w \in V$ with this skew at time t. Hence, it must hold that the skew from v or w to the root $r \in V$ is at least half as much. Assuming w.l.o.g. that this applies to v, we have that

$$|L_r(t) - L_v(t)| \geq 2\mathcal{G} + (2u + (3\vartheta+2)(\vartheta(1+\mu)-1)d)\tilde{D}. \tag{12.7}$$

Within $3\vartheta d\tilde{D}$ local time, r will broadcast a $\langle$getL()$\rangle$ message, say at time t_B. We get that

$$\begin{aligned}
|L_r(t_B) - L_v(t_B)| &\geq |L_r(t) - L_v(t)| - ((1+\mu)\vartheta - 1)(t_B - t) && \text{rate bounds} \\
&\geq 2(\mathcal{G} + (u + (\vartheta(1+\mu)-1)d)\tilde{D}). && (12.8)\ (12.7)
\end{aligned}$$

By Lemma 12.20, v will receive a message $\langle \ell' - d + u \rangle$ at some time $t' \leq t_B + \operatorname{dist}(r,v)d \leq t + (3\vartheta+1)d\tilde{D}$ such that

$$
\begin{aligned}
&|L_v(t') - (\ell' + \operatorname{dist}(r,v)(d-u))| \\
\text{Δ-inequality} \quad &\geq |L_v(t_B) - L_r(t_B)| - |L_v(t') - (\ell' + \operatorname{dist}(r,v)(d-u)) - (L_v(t_B) - L_r(t_B))| \\
\text{Lemma 12.20} \quad &\geq |L_v(t_B) - L_r(t_B)| - (\vartheta(1+\mu) - 1)d + u)\tilde{D} \\
\text{(12.8)} \quad &\geq 2\mathcal{G} + (u + (\vartheta(1+\mu) - 1)d)\tilde{D}
\end{aligned}
$$

Thus, v sends an alert message to its parent, causing a chain of alert messages that reaches the root no later than time $t' + d\tilde{D}$. At most $2\vartheta d\tilde{D}$ time after reception, by time

$$t' + (2\vartheta+1)d\tilde{D} \leq t + (5\vartheta+2)d\tilde{D},$$

the root broadcasts a reset message. □

Once a reset message is sent, we need that it results in a sufficiently small global skew. However, this global skew will be maintained by Algorithm 14, which relies on the estimates computed by Algorithm 13. The latter in turn relies on bounded logical clock rates, which Algorithm 15 deliberately violates in order to quickly remove skew from the system. To avoid this becoming a chicken-and-egg problem, we need to also show that after correcting the global skew, Algorithm 15 does not interfere for long enough so that Algorithm 14 can take the helm. The following lemma shows both properties.

Lemma 12.22. *If the root broadcasts a reset message at a time $t \geq (2\vartheta+3)d\tilde{D}$, then $\mathcal{G}(t + d\tilde{D}) \leq (u + (\vartheta(1+\mu) - 1)d)D$ and no node calls* setL(ℓ') *for any ℓ' during $[t + d\tilde{D}, t + (2\mathcal{G} - uD)/(\vartheta(1+\mu) - 1))]$.*

Proof. By Lemma 12.19, any node calling setL$(()\ell')$ for any ℓ' at or after time t must do so because of a message chain started by the root, which in particular involves it broadcasting a reset message. Since the root sends such a message at time t, it does not send such a message during $[t - 2d\tilde{D}, t]$ due to the waiting instruction and the fact that local time advances at most at rate ϑ. Hence, the triggered messages (and the reset of the root r itself) cause each $v \in V$ to call setL$((d-u)\operatorname{dist}(v,r))$ at a time $t_v \in [t + (d-u)\operatorname{dist}(v,r), t + d\operatorname{dist}(v,r)]$. Moreover, the root sends no other messages during $[t, t + d\tilde{D}]$ due to a waiting

instruction. Therefore, for any $v, w \in V$, we get that

$$
\begin{aligned}
& L_v(t + d\tilde{D}) - L_w(t + d\tilde{D}) \\
&= L_v(t_v) + (L_v(t + d\tilde{D}) - L_v(t_v)) - (L_w(t_w) + L_w(t + d\tilde{D}) - L_w(t_w)) && \text{adding 0} \\
&\le L_v(t_v) - L_w(t_w) + (1 + \mu)(H_v(t + d\tilde{D}) - H_v(t_v)) && \tfrac{dH}{dt} \le \tfrac{dL}{dt} \le (1+\mu)\tfrac{dH}{dt} \\
&\qquad - (H_w(t + d\tilde{D}) - H_w(t_w)) \\
&\le L_v(t_v) - L_w(t_w) + \vartheta(1 + \mu)(t + d\tilde{D} - t_v) - (t + d\tilde{D} - t_w) && 1 \le \tfrac{dH}{dt} \le \vartheta \\
&\le L_v(t_v) - L_w(t_w) + \vartheta(1 + \mu)(d\tilde{D} - (d - u)\operatorname{dist}(v, r)) - (d\tilde{D} - d\operatorname{dist}(w, r)) && \text{bounds on } t_v, t_w \\
&= (\operatorname{dist}(v, r) - \operatorname{dist}(w, r))(d - u) \\
&\qquad + \vartheta(1 + \mu)(d\tilde{D} - (d - u)\operatorname{dist}(v, r)) - (d\tilde{D} - d\operatorname{dist}(w, r)) \\
&= (\vartheta(1 + \mu) - 1)d\tilde{D} + u\operatorname{dist}(w, r) \\
&\le (u + (\vartheta(1 + \mu) - 1)d)D. && \operatorname{dist}(w, r) \le \tilde{D} \le D
\end{aligned}
$$

As this holds for any $v, w \in V$, this establishes the desired bound on $\mathcal{G}(t + d\tilde{D})$.

It remains to show that no node calls setL(ℓ') for any ℓ' during $[t + d\tilde{D}, t + 2\mathcal{G}/(u + (\vartheta(1 + \mu) - 1)d)]$. By Lemma 12.19, this must be caused by message chain triggered by a $\langle$getL()$\rangle$ message of the root after time $t - 2d\tilde{D}$. Due to the waiting instructions, this entails that this message was sent after time $t + d\tilde{D}$. Let $\bar{t}$ be the infimal time larger than $t + d\tilde{D}$ when the root sends a reset message. Since no node calls setL() during $[t + d\tilde{D}, \bar{t}]$, we need to show that $\bar{t} \ge t + 2\mathcal{G}/(u + (\vartheta(1 + \mu) - 1)d)$. Moreover, note that $\frac{dH_v}{dt} \le \frac{dL_v}{dt} \le (1 + \mu)\frac{dH_v}{dt}$ for all $v \in V$ during this time interval. Therefore, we can apply Lemma 12.20 to see that if the root r sends an $\langle$getL()$\rangle$ message at any time $t_r \in [t+d\tilde{D}, t+2\mathcal{G}/(u+(\vartheta(1+\mu)-1)d)]$, any caused reception of an $\langle \ell' - d + u \rangle$ message at a time t' by node $v \in V$ satisfies that

$$|\ell' - L_v(t')| \le |L_r(t_r) - L_v(t_r)| + (u + (\vartheta(1 + \mu) - 1)d)\tilde{D}.$$

Thus, it is sufficient to bound $\mathcal{G}(t_r) < 2\mathcal{G}$ for all $t_r \in [t + d\tilde{D}, t + 2\mathcal{G}/(u + (\vartheta(1 + \mu) - 1)d)]$, as then

$$
\begin{aligned}
|\ell' - L_v(t')| &< \mathcal{G}(t_r) + (\vartheta(1 + \mu) - 1)d)\tilde{D} \\
&\le 2\mathcal{G} + (u + (\vartheta(1 + \mu) - 1)d)\tilde{D},
\end{aligned}
$$

no alert message is triggered, and accordingly no reset message is sent by the root. Since due to the rate bounds we have that

$$\mathcal{G}(t_r) \le \mathcal{G}(t + d\tilde{D}) + (\vartheta(1 + \mu) - 1)(t_r - (t + d\tilde{D})),$$

for any time $t_r < \bar{t}$, we conclude that indeed $\bar{t} \ge t + (2\mathcal{G} - uD)/(\vartheta(1+\mu)-1)$. □

As our final ingredient, we observe that if the global skew remains sufficiently small, then Algorithm 15 does not interfere with the operation of Algorithm 14, i.e., does not reset the logical clocks.

Corollary 12.23. *If for times $t_0 \geq (2\vartheta+3)d\tilde{D}$ and $t_1 \geq t_0$ we have that $\mathcal{G}(t) \leq 2\mathcal{G}$ for all $t \in [t_0, t_1]$, then no node calls* setL(ℓ') *for any ℓ' during $[t_0 + (2\vartheta+3)d\tilde{D}, t_1]$.*

Proof. Analogously to the proof of Lemma 12.22, we get that no resets are triggered due to the root broadcasting $\langle \text{getL}() \rangle$ messages during $[t_0, t_1]$. Thus, by Lemma 12.19, no resets are triggered. □

Putting the above pieces together, we arrive at a variant of Algorithm 14 that stabilizes in time $O(dD)$. Note that this stabilization time is trivially asymptotically optimal, as it might $\Omega(dD)$ for distant nodes in the network to affect each other.

Theorem 12.24. *Suppose that*

- *Algorithm 6 with $T = d$ is used to compute clock estimates,*
- $2(\vartheta - 1) \leq \mu = O(u/d)$,
- $\kappa = \delta$, *where δ is as in Lemma 8.31, and*
- *Algorithm 15 is run with a sufficiently large value for $\mathcal{G} = O(uD)$.*

With this modification, Algorithm 14 is a self-stabilizing solution to the task of guaranteeing that

- $\frac{dH_v}{dt}(t) \leq \frac{dL_v}{dt}(t) \leq (1+\mu)\frac{dH_v}{dt}(t)$ *for all nodes v and times t,*
- $\mathcal{G} = O(uD)$, *and*
- $\mathcal{L} = O(u \log_\sigma D)$, *where $\sigma = \mu/(\vartheta - 1)$.*

It has a stabilization time of $O(dD)$.

Proof. Let C be a sufficiently large constant C and set $\mathcal{G} = CuD$ for Algorithm 15.

We distinguish two cases. The first is that there is a time $t \in [(2\vartheta + 4)d\tilde{D}, CdD]$ when $\mathcal{G}(t) \geq 4\mathcal{G} + (4u + (6\vartheta+4)(\vartheta(1+\mu) - 1)d)\tilde{D}$. By Lemma 12.21, the root broadcasts a reset message at some time $t' \in [t - d\tilde{D}, t + (5\vartheta+2)d\tilde{D}]$. It follows that

Lemma 12.22

$2(\vartheta - 1) \leq \mu = O(u/d)$

$$\begin{aligned}\mathcal{G}(t' + d\tilde{D}) &\leq (u + (\vartheta(1+\mu) - 1)d)D \\ &= O(uD).\end{aligned}$$

Moreover, Lemma 12.21 guarantees that no calls to setL() occur after time $t' + d\tilde{D}$ for at least

$$\frac{2\mathcal{G} - uD}{(\vartheta(1+\mu) - 1)} = \Omega\left(\frac{(2\mathcal{G} - uD)d}{u}\right) \qquad 2(\vartheta - 1) \leq \mu = O(u/d)$$
$$= \Omega\left(\frac{\mathcal{G}d}{u}\right) \qquad \mathcal{G} > uD$$

time. Because $\mathcal{G} = CuD$ for a sufficiently large constant C, this time period is at least $C'dD$ for a sufficiently large constant C'. By Corollary 12.15, after $O(d)$ additional time, say from time $\underline{t}$ on, Algorithm 13 provides suitable estimates of neighbors' clock values to nodes, and we may apply Lemma 12.16 to show that the global skew remains bounded by $O(\kappa D) = O(\delta D) = O(uD)$—at least until some node calls setL() again.

Now let $\bar{t}$ be the infimal time larger than $\underline{t}$ that (i) the global skew exceeds $O(uD)$ or (ii) a node calls setL(). Because $\mathcal{G} = CuD$ is sufficiently large, Corollary 12.23 shows that no node calls setL() at time $\bar{t}$. However, in absence of such a reset Lemma 12.16 guarantees a global skew of $O(uD)$. Therefore, $\bar{t} = \infty$, i.e., no resets occur after time $\underline{t}$ and the global skew remains bounded by $O(uD)$.

Without calls to setL(), Algorithm 15 does not affect the execution of Algorithms 13 and 14, and Theorem 12.18 can be applied to prove that the remaining properties are satisfied, too. Because

$$\underline{t} = t + (5\vartheta + 2)d\tilde{D} + O(d) = CdD + O(dD) = O(dD),$$

the stabilization time is indeed $O(dD)$. □

Handling unknown network topology

Algorithm 15 assumes that each node knows its parent and its distance to the root in a BFS tree, and that all nodes know the depth of this tree. This will not do if the network topology is not fixed at design time. Fortunately, we can overcome this obstacle by another building block, which can be run without relying on synchronization: a self-stabilizing BFS tree construction algorithm. Our synchronization algorithm then simply uses the output variables of this algorithm to determine parents, distance to the root, and depth of the tree. While transient faults might mess with these variables, too, we can make sure that the tree construction algorithm stabilizes in $O(dD)$ time, after which the analysis of the synchronization algorithm kicks in.

E12.13 Assume that a node $r \in V$ has been pre-selected. Provide a self-stabilizing algorithm that lets each node determine (i) its parent and (ii) its distance to the root in a BFS tree rooted at r. (Hint: Let nodes broadcast their state to all neighbors

every $\Theta(d)$ time. Let nodes update their state whenever an inconsistency is observed, e.g., the current parent claims a different distance to the root than the own minus 1 or another neighbor claims to be closer to the root than the current parent.)

E12.14 Augment your algorithm such that nodes learn about the depth of the tree. (Hint: Reverse the distance game: Let leafs assign themselves value 0 and other nodes the maximum of their children plus 1. Then copy the value of the root to everyone.)

E12.15 Modify your algorithm to handle the case that no root is pre-determined, but all nodes have (hardcoded) distinct identifiers from a known range. (Hint: Let the smallest identifier win!)