15 Self-stabilizing Pulse Synchronization

Chapter Contents

Learning Goals

CL: todo

15.1 Overview

In the previous chapter, we introduced and studied consensus in SMP (see Definition 14.1). In this chapter, we will utilize this tool to achieve self-stabilizing pulse synchronization (in TMP with the additional assumption that nodes have known labels $1, \ldots, n$) in fully connected networks. The resulting algorithms exhibit skew $\Theta(d)$, but using the techniques from Chapter 13, this can be improved to $\Theta(u + (\vartheta - 1)d)$.

On the highest level, our plan is to repeatedly simulate consensus instances, which after stabilization will always output 1, triggering a synchronized pulse.

E15.1 Recall the pulse synchronization algorithm from Chapter 9, whose state machine is given in Figure 9.5. If all correct nodes decide to start simulation of a run of a consensus instance within a time interval of length at most τ, how can this algorithm be used to ensure that the simulation is successful? How far apart are the times at which correct nodes complete the simulation of the consensus routine?

E15.2 Assuming that $\max_{v,w \in V_g}\{H_v(0) - H_w(0)\} \leq \tau/\vartheta$, provide a (not self-stabilizing) pulse synchronization algorithm based on the above simulation.

After transient faults, an instance might output 0, letting nodes transition into a recovery state. Being in the recovery state is a proof that some misstep occurred, allowing nodes the freedom to diverge from the behavior they would exhibit in regular operation. They use this freedom to try to convince other nodes that recovery is needed, too. If successful, all nodes transition to the recovery state, from which then a consistent "reboot" is performed.

A closer look suggests that this strategy might run into chicken-and-egg problems: How do nodes reach consensus on *when* to start simulating a consensus instance? Given that they are, initially, in arbitrary states, only some of them might execute the consensus subroutine, breaking its guarantees and, as a result, preventing to establish the required synchrony for starting the next instance consistently. Similarly, we cannot expect to guarantee that all correct nodes transition to the recovery state without first solving consensus. If f or fewer correct nodes observed an inconsistent state, the remaining at least $n-2f$ correct nodes could be fooled by the faulty nodes into believing that the system works correctly; from their perspective, the resulting execution would be indistinguishable from the one in which indeed all correct nodes are synchronized, while the at most f nodes in the recovery state are faulty.

Our key ingredient to overcoming these challenges will be to allow for an "intermediate" option. This intermediate option is that not all correct nodes participate, but in this case use input 0. The non-participating nodes will be considered to support output 0 by virtue of their very non-participation; while they will not explicitly generate a local output of 0, we can let them switch to the recovery state based on the lack of a consensus instance they participate in. This behavior of a consensus algorithm is formalized in the following definition.

Definition 15.1 (Silent binary consensus)**.** *We call a binary consensus protocol* silent*, if in each execution in which all correct nodes have input* 0*, correct nodes send no messages.*

Thus, if no correct node has input 1, non-participation of correct nodes is indistinguishable (to others) from them having input 0 and, by validity of the consensus protocol, participating correct nodes output 0.

In SMP, silent consensus can be achieved without significant overhead.

Theorem 15.8. *Assume that we are given an SMP consensus protocol for a fully connected network. Then there is an SMP silent binary consensus protocol of the same resilience and message size, whose round complexity is larger by* 2 *extra rounds.*

Corollary 15.2. *For any $n > 3f$, there is an SMP silent binary consensus protocol running for $R(f) = 3(f+1) + 2 \in O(f)$ rounds, in which correct nodes communicate exclusively by* 1*-bit broadcasts.*

Proof. We apply Theorem 15.8 to Algorithm 18, whose properties are established in Theorem 14.18. □

From here on, we will use R as a shorthand to refer to the round complexity of the SMP silent binary consensus algorithm we employ.

In order to leverage this result in TMP, we need to repeatedly simulate an SMP protocol. Moreover, we will need that this simulation is self-stabilizing, in the sense that if the calls to the subroutine satisfy certain timing requirements, then the simulation wrapper will recommence operating according to specification from any initial state. We capture the behavior we require in the following definition.

Definition 15.3 (Recurrent TMP simulation of SMP silent binary consensus)**.** *An algorithm is a* recurrent TMP simulation of SMP silent binary consensus algorithm $\mathcal{A}$ *if and only if it meets the following specification. Nodes $v \in V_g$ may generate* run events *"RUN 0" or "RUN 1", which locally initiate simulation of a consensus instance with input* 0 *or* 1*, respectively. Run events satisfy the following constraints:*

- *All run events corresponding to a simulated instance fall into a time interval of length at most τ.*

- *During each such interval, each $v \in V_g$ generates at most one run event. If it does, it is a* participating *node in this instance.*
- *If any node $v \in V_g$ does not participate in an instance, each participating node generates a* RUN *0 event for the instance.*
- *No node $v \in V_g$ generates another run event until each participating node locally generated a unique corresponding* output event*, i.e., either "*OUTPUT *0" or "*OUTPUT *1."*

The output events simulate the behavior of $\mathcal{A}$, by which we mean the following:

- ***Agreement:*** *All output events corresponding to an instance are of the same type, i.e., either all are* OUTPUT *0 events or all are* OUTPUT *1 events.*
- ***Validity:*** *If all run events of an instance are* RUN *0 events, the corresponding output events are* OUTPUT *0 events. If all run events are* RUN *1 events, the corresponding output events are* OUTPUT *1 events.*
- ***Termination:*** *The output events of an instance occur within $O(\tau + Rd)$ time of their run events, where R is the round complexity of $\mathcal{A}$.*
- ***Timeliness:*** *If the output of an instance is* 1*, all corresponding output events occur within a time interval of length $\lambda = O(d)$. Regardless of the output, no output events occurs earlier than τ time after the first run event of the instance.*

The additional condition of timeliness is imposed to ensure a required timing relation between the relevant outputs. In the synchronous model, such timing is trivial to achieve by simply deciding on which round the output is "used," but more care is needed in the context of our simulation.

Due to the careful choice of constraints on run events, it is straightforward not only to devise an algorithm meeting the requirements of Definition 15.3, but to also make it self-stabilizing.

E15.3 Recall the algorithm you devised for Section 15.1. Can you modify it such that is satisfies Definition 15.3. Hint: Apart from wrapping the previous routine into a loop, you will have to account for the case that not all correct nodes participate. Tackle this by locally aborting the simulation if it cannot progress fast enough for any reason.

E15.4 What, if anything, needs to be done to make your routine self-stabilizing?

Theorem 15.9. *Assume that we are given an SMP silent binary consensus protocol $\mathcal{A}$ for a fully connected network that terminates after exactly R rounds at all correct nodes. Fix any $\tau \in \mathbb{R}_{>0}$. Set $H_0 = \vartheta(\tau + d)$ and choose T_1, T_2, and T_3 in accordance with Theorem 9.18 for $T = (2\vartheta^2 + 3\vartheta)d$. Then Algorithm 20* *with $\mathcal{A}$ is a self-stabilizing recurrent TMP simulation of $\mathcal{A}$, i.e., meets the requirements of Definition 15.3. It has stabilization time $O(\tau + Rd)$.*

To meet the input constraints, we employ the recovery mechanism mentioned earlier. That is, by default consensus is initiated regularly by all correct nodes in synchrony, which then results in output 1 (due to everyone using input 1 and validity), which triggers the next synchronized pulse. If this scheme is (visibly) disrupted, nodes use input 0 instead, essentially voting for transition to recovery. In the extreme case that a node does not observe enough pulses to even participate in an instance, the silent consensus algorithm implicitly interprets this as voting for a transition to the recovery state. This achieves the goal of either forcing a synchronized pulse by all correct nodes—resulting in stabilization due to established timing relations—or eventually getting all correct nodes into the recovery state, in which they refuse to support generating further pulses.

Naturally, *eventually* nodes in recovery must decide to start producing pulses again. Unfortunately, the above scheme relies on nodes in the recovery state to try to *stop* generation of (inconstent) pulses. In order to get out, they would all have to decide to change strategy with sufficient synchrony. Nonetheless, all this effort went not to waste. If we end up with all correct nodes in the recovery state, this amounts to all of them having *proof* that something is amiss. We exploit this by relying on a subroutine that only *every now and then* produces a synchronized pulse.

Definition 15.4 (Resynchronization algorithm)**.** *B is an f-resilient* resynchronization algorithm *with skew ρ and separation window Ψ that stabilizes in time $S(B)$, if the following holds. Regardless of initial states, there exists a time $t \leq S(B)$ such that every correct node $v \in V_g$ locally generates a* resynchronization pulse *at time from $[t, t + \rho)$ and no resynchronization pulse during time $[t - \Psi, t)$. We call such a resynchronization pulse* good.

A good pulse can be used to jump-start the pulse generation in case all correct nodes end up in the recovery state. Subsequently, pulses will be generated as expected. Hence, correct nodes will not transition to recovery again, and therefore safely ignore any future (possibly inconsistent) resynchronization pulses.

Theorem 15.16. *If* (15.1)–(15.12) *hold and the network is fully connected, the pulse synchronization algorithm given by the state machines in Figures 15.1 and 15.2 stabilizes on a good resynchronization pulse. It has skew $2d$, $P_{\min} = 3\lambda + 3d$, and $P_{\max} = O(T_4 + T_5 + \tau + Rd)$.*

The remaining piece to the puzzle is to provide a resynchronization algorithm. A very simple solution is given by having each correct node consider each of $f + 1$ nodes the leader for some (local) time in a round-robin fashion. By chosing for each subsequent node the speed at which switches between leaders by a constant factor larger, we ensure that, eventually, each of the $f+1$ candidates will be considered the leader for some time by all correct nodes. Thus, a correct leader can generate a resynchronization pulse.

Theorem 15.18. *For any Ψ and $f < n$, there is an f-resilient deterministic resynchronization algorithm with skew u and separation window Ψ that stabilizes in $O((\Psi + d)(\vartheta(f + 2))^n)$ time.*

Corollary 15.5. *For any fully connected network of $n > 3f$ nodes, there is a self-stabilizing pulse synchronization algorithm tolerating f faults that stabilizes in time $O(fd(\vartheta(f + 2))^n)$. It has skew $2d$, $P_{\min} = 3\lambda + 3d$, and $P_{\max} = O(Rd)$.*

Proof. Follows from Corollary 15.2 and Theorems 15.9, 15.16, and 15.18 after determining suitable constraints so that $\Psi = O(Rd)$, which is done in Lemma 15.17. □

For a very small number of nodes, the above approach might yield an acceptable stabilization time. Yet, even for tolerating $f = 2$ faults with the smallest corresponding number of nodes, $n = 7$, $(f + 1)^n > 2000$.

A much faster and still simple resynchronization algorithm can be obtained using randomization. The idea is to have nodes spontenously declare themselves leader. If this is unpredictable (due to randomization), faulty nodes need to constantly interfere to prevent a successful resynchronization pulse. However, by limiting how frequently nodes may generate such pulses and ignoring them if they violate this constraint, eventually a good pulse emerges with a large probability. This would result in stabilization time $\Theta(n^2\Psi)$ for a suitable choice of parameters.

We present an improved scheme which adds a voting step: In order for a resynchronization pulse to be locally accepted, $n - f$ nodes must claim to have observed it. Thus, $n - 2f$ correct nodes must have done so, meaning that faulty nodes use up their "budget of distraction" much faster. This leads to stabilization time $\Theta(n\Psi)$ with a large probability.

Theorem 15.21. *If the assumptions in Definition 15.19 hold and $3f < n$, Algorithm 22 with timeouts given by (15.16)–(15.18) is an f-resilient resynchronization algorithm with skew $2d$ that stabilizes in time $O((\Psi + d)n)$ with probability $1 - 2^{-\Omega(n)}$.*

Corollary 15.6. *For any fully connected system of $n > 3f$ nodes to which Definition 15.19 applies, there is a randomized self-stabilizing pulse synchronization algorithm tolerating f faults that stabilizes in time $O(fnd)$ with probability $1 - 2^{-\Omega(n)}$. It has skew $2d$, $P_{\min} = 3\lambda + 3d$, and $P_{\max} = O(Rd)$.*

Proof. Follows from Corollary 15.2 and Theorems 15.9, 15.16, and 15.21 after determining suitable constraints so that $\Psi = O(Rd)$, which is done in Lemma 15.17. □

Throughout this chapter, we assume a fully connected network of $n > 3f$ nodes, in which nodes have known labels $1, \ldots, n$.

E15.5 Derive variants of Corollaries 15.5 and 15.6 in which all occurences of n are replaced by $O(f)$.

E15.6 For Corollary 15.6, in case $f \ll n$ this results in a much smaller probability of stabilization in the stated time. Does this imply a trade-off between the two variants of the corollary?

Remark 15.7.

It is possible to reduce the stabilization time of the second approach by avoiding consensus in favor of using similar ideas with the pulse generation scheme from Figure 9.5. The resulting algorithm is also simpler. However, we opted to not analyze two very similar algorithms in this chapter.

In terms of the asymptotic behavior, the best known results are obtained by using the approach we presented here recursively. This results in a deterministic algorithm with stabilization time $O(nd)$ and randomized algorithms stabilizing in $(\log n)^{O(1)}d$ time with probability $1 - 1/n^c$ for a constant c that can be chosen freely. These algorithms are also communication-efficient in that only $(\log n)^{O(1)}$ bits need to be broadcast by correct nodes in $\Theta(d)$ time.

However, since all of these algorithms require full connectivity, they scale poorly. However, these asymptotic bounds matter little for small numbers of nodes.

15.2 Silent Consensus

In order to obtain silent binary consensus protocols, we take an arbitrary (non-silent) consensus protocol R-round $\mathcal{A}$ and add an initial voting procedure, in which sending no message implicitly supports the default output of 0. Nodes with input 1 broadcast an (empty or 1-bit) message. If no correct node has input 1, then no node will receive any kind of message from more than f distinct nodes. Thus, they can safely opt for output 0 without ever running the protocol. This requires some care, however, since some nodes might receive more than

f messages, while others do not. To address this, we add a second round of voting and let the nodes behave as follows based on the number of distinct senders (also counting themselves) from which the receive messages:

1. $\leq f$ messages in first vote: decide on output 0 and terminate;
2. $> f$, but $< n - f$ messages in first vote: send no message in second vote;
3. $\geq n - f$ messages in first vote: broadcast message in second vote;
4. $\leq f$ messages in second vote: execute $\mathcal{A}$ with input 0, but output 0 regardless of the result;
5. $> f$, but $< n - f$ messages in second vote: execute $\mathcal{A}$ with input 0 and output the result; and
6. $\geq n - f$ messages in second vote: execute $\mathcal{A}$ with input 1 and output the result.

We remark that in the event that not all nodes execute $\mathcal{A}$, the execution of $\mathcal{A}$ is inconsistent with the model assumptions. Thus, $\mathcal{A}$ might fail to specify state updates or messages to send, or it might fail to terminate within R rounds at a correct node. If any of this applies, the respective node immediately aborts the local execution of $\mathcal{A}$ and terminates with output 0.

Theorem 15.8. *Assume that we are given an SMP consensus protocol for a fully connected network. Then there is an SMP silent binary consensus protocol of the same resilience and message size, whose round complexity is larger by* 2 *extra rounds.*

Proof. We claim that the protocol described above meets all the requirements. Note first that by Theorem 14.29, the resilience of $\mathcal{A}$ cannot exceed $\lceil n/3 \rceil - 1$, so it holds that $n > 3f$. Since correct nodes abort the local execution of $\mathcal{A}$ after at most R rounds, the new protocol runs for at most $R + 2$ rounds. Clearly, $\mathcal{A}$ will also have to use *some* kind of messages, so the initial two rounds of voting do not increase the message size.

With these things out of the way, it remains to show silence, validity, and agreement of the new protocol. To this end, we consider the following cases.

Case 1: No correct node has input 1. Then all correct nodes output 0 and terminate right away.

Case 2: Some correct node terminates after the first vote. Then no correct node received more than $2f$ messages in the first vote. Hence, no correct node sends a message in the second vote. Thus, all correct nodes running (or aborting) $\mathcal{A}$ ignore the result and output 0.

Case 3: No correct node terminates after the first vote, but some node ignores the result of executing $\mathcal{A}$. Then no correct node received more than $2f < n - f$

messages in the second vote. Since all correct nodes execute $\mathcal{A}$ with input 0, by validity of $\mathcal{A}$ all correct nodes output 0.

Case 4: No correct node terminates after the first vote or ignores the result of executing $\mathcal{A}$. Then, by agreement of $\mathcal{A}$, all correct nodes output the same value.

Case 5: All correct nodes have input 1. Then all correct nodes execute $\mathcal{A}$ with input 1 and output the result. By validity of $\mathcal{A}$, they hence all output 1.

Case 1 shows silence. Cases 1 and 5 prove validity. Cases 2, 3, and 4 are exhaustive and hence establish agreement. □

Knowing that the requirement of silence comes at no additional asymptotic cost, we can move on to simulating silent binary consensus in TMP. Here, we can leverage any non-stabilizing pulse synchronization algorithm, exploiting that we only need to generate R pulses. Therefore, wrapping the non-stabilizing algorithm in a loop that resets it after completion and running a thread performing simple consistency checks, we readily obtain the desired simulation. For simplicity, we choose to employ the algorithm from Chapter 9 in our construction.

We remark that, at first glance, the possibility that some nodes might not participate in the simulation might seem like an obstacle. However, recall that in this case the assumption is that *no* correct node participates with input 1, so no messages are sent by correct nodes and all participating nodes immediately terminate. In the event that a correct node wants to execute $\mathcal{A}$, all correct nodes participate and can therefore take part in running the pulse synchronization algorithm driving the simulation (even if they immediately terminate in the simulated synchronous consensus protocol).

Theorem 15.9. *Assume that we are given an SMP silent binary consensus protocol $\mathcal{A}$ for a fully connected network that terminates after exactly R rounds at all correct nodes. Fix any $\tau \in \mathbb{R}_{>0}$. Set $H_0 = \vartheta(\tau + d)$ and choose T_1, T_2, and T_3 in accordance with Theorem 9.18 for $T = (2\vartheta^2 + 3\vartheta)d$. Then Algorithm 20 with $\mathcal{A}$ is a self-stabilizing recurrent TMP simulation of $\mathcal{A}$, i.e., meets the requirements of Definition 15.3. It has stabilization time $O(\tau + Rd)$.*

Proof. First, observe that in absence of run events, each node will set r to 0 within $O(\tau + Rd)$ due to the local time window of $[c, c + O(\tau + Rd)]$. By Definition 15.3, each run event a node $v \in V_g$ is followed by an output event before the next run event at the node. Therefore, after discarding at most one initial output event (occuring within $O(\tau + Rd)$ time) for each $v \in V_g$, we can associate each output event with the respective preceding run event at the node. Noting that in the absence of run events, any $v \in V_g$ with $r = 1$ will set $r = 0$ and generate an output event within $O(\tau + Rd)$ time, we have a one-to-one mapping

Algorithm 20 TMP simulation of silent binary consensus at node $v \in V_g$. Memory is allocated for exactly one instance of SIMULATE. Thus, calling SIMULATE will effectively terminate any running instance by clearing any associated state.

```
if run event RUN B occurs then
    $r \leftarrow 1$    ▹ indicates that an instance is (supposed to be) running
    $c \leftarrow$ getH()    ▹ store local time of call
    call SIMULATE($b$)
end if
if (getH() $\geq c + 2\vartheta^2\tau + 11\vartheta^4(R+1)d$ or getH() $< c$)) and $r = 1$ then
    $r \leftarrow 0$    ▹ terminate current instance
    generate OUTPUT 0 event
end if
procedure SIMULATE($b$)
    initialize local instance of state machine from Figure 9.5 to RESET
    run instance with getH() replaced by getH() $- c$    ▹ "initialize" $H_v$ to 0
    initialize local instance of $\mathcal{A}$ with input $b$
    for each generated pulse do
        $h \leftarrow$ getH()
        update state of $\mathcal{A}$ based on stored messages $(m^{\leftarrow}(w))_{w \in V}$
        if state of $\mathcal{A}$ indicates termination with output $b$ and $r = 1$ then
            generate OUTPUT B event
            $r \leftarrow 0$    ▹ terminate current instance
        end if
        for each $w \in V \setminus \{v\}$ do
            compute the message $m^{\rightarrow}(w)$ that $v$ sends to $w$ next in $\mathcal{A}$
            $m^{\leftarrow}(w) \leftarrow \perp$ // indicating no message received from $w$ (yet)
        end for
        wait until getH() $= h + 2\vartheta d$
        for each $w \in V \setminus \{v\}$ do
            send $\langle \mathcal{A}, m^{\rightarrow}(w) \rangle$ to $w$
        end for
        wait until getH() $= h + (2\vartheta^2 + 3\vartheta)d$
        for each $w \in V \setminus \{v\}$ do
            if received $\langle \mathcal{A}, m \rangle$ from $w$ during current loop iteration then
                $m^{\leftarrow}(w) \leftarrow m$    ▹ can be arbitrary if received more than one
            end if
        end for
    end for
end procedure
```

between input and output events. In particular, termination is satisfied, and for each "instance" of $\mathcal{A}$ (defined by the run events and resulting calls to SIMULATE) occuring within a time interval of length τ) we have well-defined output events at each participating $v \in V_g$.

Accordingly, it is sufficient to show that for each instance whose first run event occurs after the last discarded output event that agreement, validity, and timeliness hold. Observe that the constraints on run events imposed by Definition 15.3 guarantee that for each instance, all its output events occur before any new run events, i.e., the next instance is started at *any* correct node. Thus, non-participating nodes are not executing SIMULATE during the instance, i.e., at any time in the interval spanned by the run and output events of the instance. Denote by $V_p \subseteq V_g$ the set of participating nodes. By Definition 15.3, each $v \in V_p$ has a run event at some time t_v, where $\max_{v\in V_g}\{t_v\} - \min_{v\in V_g}\{t_v\} \leq \tau$. Since $H_0 > \vartheta(\tau + d)$, no $v \in V_p$ can generate a pulse before time $\min_{v\in V_g}\{t_v\} + \tau + d \geq \max_{v\in V_g}\{t_v\}$. Hence, the second part of the timeliness condition is satisfied and no participating node will receive any $\langle \mathcal{A}, m\rangle$ messages that do not "belong" to this instance from correct nodes after generating its first pulse.

We distinguish two cases, the first being that not all correct nodes participate. In this case, Definition 15.3 requires that all run events of the instance are RUN 0 events, i.e., all participating nodes use input 0 in their local instance of $\mathcal{A}$. Because $\mathcal{A}$ is silent, this entails that correct nodes will not send any $\langle \mathcal{A}, m\rangle$ messages without receiving any such message from another correct node first. However, as all calls to SIMULATE at participating nodes use input 0, no correct node will send a first such a message belonging to the instance. Since there are also no stray messages not belonging to the instance are received, we conclude that no $\langle \mathcal{A}, m\rangle$ messages are sent by correct nodes that are received after the first pulse at any participating nodes, but before the instance is terminated at all participating nodes. We conclude that all participating nodes generate an OUTPUT 0 event, establishing validity and agreement; timeliness is satisfied, as only the second, already established part of the condition applies in case no OUTPUT 1 events are generated.

The second case is that all correct nodes participate, i.e., $V_p = V_g$. Consider the time $t_0 := \max_{v\in V_g}\{t_v\} + d$. By this time, all correct nodes have set their local copy of the state machine from Figure 9.5 to RESET at least d time ago, i.e., no PROPOSE messages from earlier instances are still in transit. At time t_0, at $v \in V_g$ a call to getH() $- c$ will return the value $H_v(t_0) - c_v(t_0) =$

$H_v(t_0) - c_v(t_v) = H_v(t_0) - H_v(t_v)$ satisfying that

$$\begin{aligned} 0 &\leq H_v(t_0) - H_v(t_v) && \tfrac{dH_v}{dt} > 0 \\ &\leq \vartheta(t_0 - t_v) && \tfrac{dH_v}{dt} \leq \vartheta \\ &\leq \vartheta(\tau + d) && \text{defition of } t_0 \\ &\leq H_0. && \text{choice of } H_0 \end{aligned}$$

Since the instances of Figure 9.5 run by the nodes use getH() $- c$ as their hardware clock function, this establishes Equation (9.13) with time 0 replaced by time t_0. As we chose T_1, T_2, and T_3 in accordance with Theorem 9.18 for $T = (2\vartheta^2 + 3\vartheta)d$, we may hence "shift" time such that $t_0 = 0$ and apply the theorem to conclude that the correct nodes will produce pulses with skew $2d$, $P_{\min} = T$, and $P_{\max} = \vartheta T + (2\vartheta + 4)d$, where the first pulse of each correct node is generated by time $t_0 + H_0 + (\vartheta - 1)T + (2\vartheta + 2)d$; as no new run events occur until all correct nodes generated their output events, this continues at least until then.

By induction on the pulse index, which is equal to the round index of the simulated execution of $\mathcal{A}$, we now claim that synchronous execution of $\mathcal{A}$ is correctly simulated with inputs matching the run events of the instance (until a new run event occurs at a correct node), simulation of round $i \in \mathbb{N}_{>0}$ is complete by time $t_0 + H_0 + (\vartheta - 1)T + (2\vartheta + 2)d + iP_{\max}$. To see this, first observe that $P_{\min} = T = (2\vartheta^2 + 3\vartheta d)$ implies that each iteration of the for loop in SIMULATE completes before the next pulse, which occurs after at most $P_{\max}$ time. Because the skew is $2d$, no correct node sends a message for round/pulse r before all correct nodes have started the r-th loop iteration. Similarly, a message sent by node $v \in V_g$ in the r-th loop iteration to $w \in V_g$ is received no later than $(2\vartheta + 1)d$ time after v generated the pulse, and hence more than $(2\vartheta + 3)d$ time after w started its r-th loop iteration; thus, it is received no more than $(2\vartheta^2 + 3\vartheta)d$ local time after the start of the loop iteration and taken into account by w when performing its state update in the next loop iteration.

The induction is anchored by each node initializing $\mathcal{A}$ locally according to its run event and generating its first pulse by time $t_0 + H_0 + (\vartheta - 1)T + (3 + 2(\vartheta - 1))d + iP_{\max}$. For the induction step, the above observations imply that all messages sent in the previous loop iteration by correct nodes have been received and stored; hence, state update and computed outgoing messages are consistent with synchronous execution of $\mathcal{A}$ and the induction step succeeds. In particular, R rounds of the simulation are complete by time

$$t_0 + H_0 + (\vartheta - 1)T + (2\vartheta + 2)d + RP_{\max} < \min_{v \in V_g}\{t_v\} + 2\vartheta\tau + 11\vartheta^3(R + 1)d.$$

It follows that each node generates its local output event for the instance because it determines that $\mathcal{A}$ terminated, or due to $2\vartheta^2\tau + 11\vartheta^4(R+1)d$ having passed since SIMULATE has been called (i.e., the second if-block of Algorithm 20 being executed). Hence, correct nodes generate output events satisfying agreement and validity due to the respective properties of $\mathcal{A}$. By assumption, this happens after exactly R simulated rounds, i.e., between consecutive pulses. Since pulses occur at most $P_{\max} = O(d)$ time apart, we already have the second part of the timeliness condition, the output events also satisfy timeliness. □

15.3 Stabilization based on a resynchronization algorithm

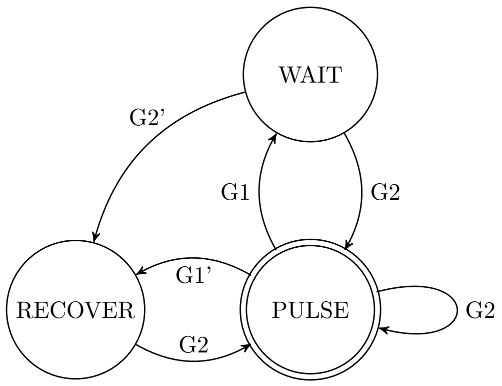
As discussed in the overview section, our approach to obtaining a self-stabilizing pulse synchronization is

- generate pulses based on repeatedly running consensus,
- let nodes transition to state RECOVER if certain consistency checks fail or the output of a consensus instance is 0, and
- "jump-start" the iterative process again using a resynchronization pulse if all nodes end up in state RECOVER.

More concretely, take a look at the state machine depicted in Figure 15.1. Each node runs a local copy of this state machine. In normal operation, nodes will alternate between the states PULSE (locally generating a pulse) and WAIT. The transition from PULSE to WAIT requires that $n - f$ nodes (are being observed) to transition to PULSE within $T_4 = O(d)$ local time around their own transition to PULSE, i.e., things looking like all correct nodes might have made generated a pulse with skew $O(d)$. The transition from WAIT to PULSE requires that an OUTPUT 1 event is generated, locally terminating an instance of a (simulated silent binary) consensus instance. The node will transition to RECOVER instead if

- too few nodes transition to PULSE in synchrony when a pulse is generated,
- a consensus instance generates an OUTPUT 0 event while in state WAIT, or
- $T_{\text{wait}} = \Omega(Rd)$ local time expired since transitioning to WAIT, i.e., in normal operation a pulse should have been generated by now.

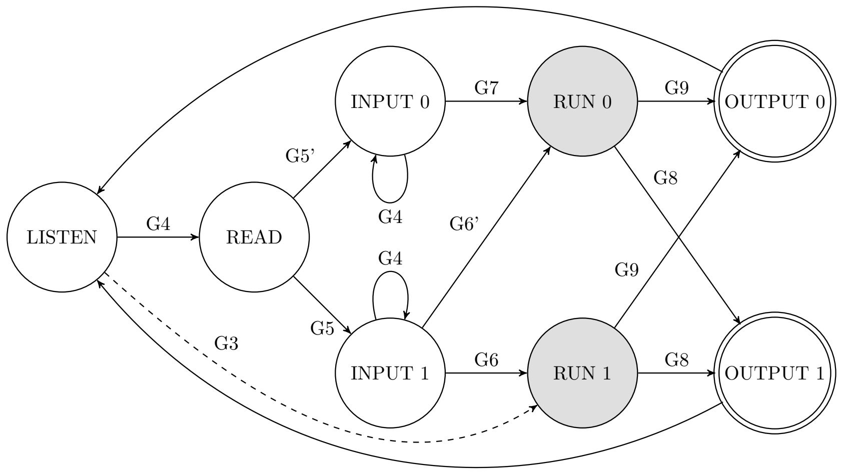
The sole way out of RECOVER is also generation of an OUTPUT 1 event, which triggers a pulse regardless of the current state of the main state machine. The difference between WAIT and RECOVER lies in how the states relate to *initiating* consensus instances, which is the sole purpose of the auxilliary state machine depicted in Figure 15.2.



Guard	Condition
G1	$\langle T_4 \rangle$ expires and received $\geq n - f$ PULSE messages within time T_4 before T_4 expired
G1'	$\langle T_4 \rangle$ expires and $\neg$G1
G2	auxiliary machine signals OUTPUT 1
G2'	$\langle T_{wait} \rangle$ expires or auxiliary machine signals OUTPUT 0

Figure 15.1
The main state machine. When a node transitions to state PULSE it generates a local pulse event and send a PULSE message to all nodes. When the node transitions to state WAIT it broadcasts a WAIT message to all nodes. Guard G1 employs a sliding window memory buffer, which stores any PULSE messages that have arrived within time T_4 (as measured by the local clock). When a correct node transitions to PULSE it resets a local T_4 timeout. Once this expires, either Guard G1 or Guard G1' become satisfied. Similarly, the timer T_{wait} is reset when node transitions to WAIT. Once it expires, Guard G2' is satisfied and node transitions from WAIT to RECOVER. The node can transitions to the PULSE state when Guard G2 is satisfied, which requires an OUTPUT 1 signal from the auxiliary state machine given in Figure 15.2.

The auxilliary state machine, which run in parallel to the main state machine, shoulders the lion's share of the stabilization work. It is carefully engineered to ensure that, within $O(Rd)$ time, the run events it triggers via transitions to states RUN 0 and RUN 1, respectively, meet the specification given in Definition 15.3. These run events drive the recurrent simulation of a silent consensus protocol in accordance with the definition. Hence, once the run events satisfy the requirements, we are guaranteed that whenever *any* correct node generates an OUTPUT 1 event, *all* correct nodes do so within $O(d)$ time. Such an event will cause all correct nodes to transition to PULSE and subsequently WAIT in tight synchronization. In the auxilliary state machine, this lets all correct nodes, which after the pulse immediately transitioned to state LISTEN, to transition via



Guard	Condition
G3	generating resynchronization pulse
G4	$\geq f+1$ WAIT messages within time T_{listen}
G5	$\geq f-1$ WAIT messages within time T_{listen}
G5'	$\langle T_{listen} \rangle$ expires
G6	$\langle T_5 \rangle$ expires while not in RECOVER
G6'	$\langle T_5 \rangle$ expires while in RECOVER
G7	$\langle T_5 \rangle$ expires
G8	**A** outputs '1'
G9	**A** outputs '0' or G4 is satisfied

Figure 15.2
The auxiliary state machine. The auxiliary state machine is responsible for generating the input events for the consensus simulation routine. The gray states correspond to simulation of the consensus routine. If the node transitions to RUN 0, it generates an INPUT 0 event. If the node transitions to RUN 1, it generates an RUN 1 event. When the consensus simulation produces an output event, the node transitions to either OUTPUT 0 or OUTPUT 1 (sending the respective output signal to the main state machine) and immediately to state LISTEN. The timeouts T_{listen} and T_5 are reset when a node transitions to the respective states that use a guard referring to them. Both INPUT 0 and INPUT 1 have a self-loop that is activated if Guard G4 is satisfied. This means that if Guard G4 is satisfied while in these states, the timer T_5 is reset.

READ and INPUT 1 to RUN 1 within a time window of length $\tau = O(Rd)$. Given suitable constraints on timeouts, we can repeat this argument inductively to show that any such event implies stabilization.

For simplicitly, we collect all inequalities that are required here and assume throughout this section that all of them hold without stating this explicitly in lemmas. For a sufficiently large constant $C > 0$, the following constraints then are sufficient for our purposes:

$$T_4 \geq \vartheta(\lambda + d) \tag{15.1}$$

$$T_5 > (\vartheta - 1)T_4 + \vartheta(T_{\text{listen}} + 2d) \tag{15.2}$$

$$T_{\text{listen}} \geq (\vartheta - 1)T_4 + \vartheta(\lambda + 2u) \tag{15.3}$$

$$T_{\text{wait}} \geq \vartheta(\lambda + T_4 + T_5 + T_{\text{listen}} + 2d) \tag{15.4}$$

$$\tau > \left(1 - \frac{1}{\vartheta}\right) T_5 + 2T_4 + 3T_{\text{listen}} + \lambda + 3d \tag{15.5}$$

$$T_5 \geq \frac{\vartheta(4T_4 + 3T_{\text{listen}} + CRd)}{3 - 2\vartheta} \tag{15.6}$$

$$\tau = O(Rd) \tag{15.7}$$

$$T_4 + T_{\text{listen}} = O(d) \tag{15.8}$$

$$T_{\text{listen}} \geq \vartheta(2T_4 + 2d) \tag{15.9}$$

$$T_{\text{wait}} = O(T_5) \tag{15.10}$$

$$\tau \geq \rho \tag{15.11}$$

$$\Psi \geq CT_5 \tag{15.12}$$

With this prerequisites in place, we can now formalize the above statement.

Lemma 15.10. *Assume that during $[t_0, t_1]$, Guard G4 is not satisfied by any $v \in V_g$ that is in state* RUN *0 or* RUN *1. Moreover, assume that the run events generated by nodes $v \in V_g$ transitioning to* RUN *0 or* RUN *1 satisfy the conditions of Definition 15.3 during $[t_0, t_1]$ and are used as input to a recurrent TMP simulation of SMP silent binary consensus. If there is $t \in [t_0 + O(Rd), t_1 - \lambda]$ when some $v \in V_g$ triggers an* OUTPUT *1 event, denote by $t' \in [t - \lambda, t]$ the first output event of the same consensus instance at a correct node. Then the state machines given in Figures 15.1 and 15.2 solve pulse synchronization with skew λ, $P_{\min} = 3\lambda + 3d$, and $P_{\max} = O(T_4 + T_5 + \tau + Rd)$ from time t' on.*

Proof. Note that the condition that Guard G4 is not satisfied by any $v \in V_g$ in RUN 0 or RUN 1 implies that correct nodes transition to OUTPUT 0 or OUTPUT 1 if and only if the simulation algorithm generates a respective event. Given that the transitions to RUN 0 and RUN 1 satisfy the conditions of Definition 15.3, consensus instances (and which output events are associated with them) are well-defined. By timeliness, t' is hence well-defined and each $v \in V_g$ generates a (unique) OUTPUT 1 event at a time $p_{v,1} \in [t', t'+\lambda]$. In the main state machine, this forces a transition to PULSE from any state (including PULSE, as indicated by the loop), i.e., generation of a pulse.

Therefore, it remains to show that the nodes $v \in V_g$ will continue to generate pulses with skew λ satisfying the period bounds $P_{\min}$ and $P_{\max}$. We show this by induction, where the induction hypothesis not only states that the first $i \in \mathbb{N}_{>0}$ pulses have been generated correctly at times $p_{v,i}$, $v \in V_g$, but also that the run events satisfy the conditions of Definition 15.3 during $[t_0, \max_{v\in V_g}\{p_{v,i}\}]$.

The above considerations and the prerequisites of the lemma anchor the induction at $i = 1$. For the step from $i \in \mathbb{N}_{>0}$ to $i + 1$, observe first that each $v \in V_g$ will receive $n - f$ PULSE messages from distinct senders (counting themselves as well) during

$$[\min_{w\in V_g}\{p_{w,i}\} + d - u, \max_{w\in V_g}\{p_{w,i}\} + d] \subset [p_{v,i} - \lambda + d - u, p_{v,i} + \lambda + d].$$

Since this time window is of length at most $\lambda + u$ and $T_4 > \vartheta(\lambda + d)$ by (15.1), it follows that all correct nodes transition to state WAIT during

$$\left[\min_{w\in V_g}\{p_{w,i}\} + \frac{T_4}{\vartheta} + d - u, \max_{w\in V_g}\{p_{w,i}\} + T_4 + d\right],$$

with the corresponding messages being received during

$$\left[\min_{w\in V_g}\{p_{w,i}\} + \frac{T_4}{\vartheta} + 2(d - u), \max_{w\in V_g}\{p_{w,i}\} + T_4 + 2d\right].$$

By (15.3) and the fact that the i-th pulse has skew at most λ, this lies within an interval of T_{listen} local time at each correct node. Because each $v \in V_g$ transitions to state PULSE at time $p_{v,i}$, by timeliness it must have been in state RUN 0 or RUN 1 for at least τ time. Hence, no WAIT message from this node is received during $[\max_{w\in V_g}\{p_{w,i}\} - \tau + d, \min_{w\in V_g}\{p_{w,i}\} + \frac{T_4}{\vartheta} + 2(d-u)]$, i.e., for more than $\tau - \lambda - u$ time before the above time interval begins. By (15.5), we have that

$$\tau - \lambda - u > T_{\text{listen}}.$$

Together, this entails that each $v \in V_g$ transitions to READ and then INPUT 1 at some time from $[\min_{w\in V_g}\{p_{w,i}\} + T_4/\vartheta + 2(d-u), \max_{w\in V_g}\{p_{w,i}\} + T_4 + 2d]$.

Next, note that no correct node can transition to WAIT again without producing another pulse first and $T_4 > \vartheta\lambda$ expiring again at correct nodes. Thus, we get that no WAIT messages are received from correct nodes during $(\max_{w\in V_g}\{p_{w,i}\} + T_4 + 2d, \min_{w\in V_g}\{p_{w,i+1}\} + \lambda]$ (where so far $\min_{w\in V_g}\{p_{w,i+1}\} = \infty$ has not been ruled). Consequently, no correct node satisfies Guard G4 at any time during $(\max_{w\in V_g}\{p_{w,i}\}+T_4+T_{\text{listen}}+2d, \min_{w\in V_g}\{p_{w,i+1}\}+\lambda]$. In particular, no correct node in state INPUT 1 will reset its timeout T_5 or leave state RUN 1 without generating an output event.

Denote by t the earliest time when a correct node $v \in V_g$ leaves state INPUT 1 (after $p_{v,i}$, i.e., during the current iteration). As v transitioned to INPUT 1 no earlier than time $\min_{w\in V_g}\{p_{w,i}\}+T_4/\vartheta+2(d-u)$, we have that

$$\begin{aligned} t &\geq \min_{w\in V_g}\{p_{w,i}\}+\frac{T_4+T_5}{\vartheta}+2(d-u) && (15.13)\\ &> \max_{w\in V_g}\{p_{w,i}\}+T_4+T_{\text{listen}}+2d\ . && (15.2) \end{aligned}$$

Define $\bar{t}$ as the infimum of all times $t' \geq t$ when one of the following occurs at some $v \in V_g$:

(i) v leaves RUN 0 or RUN 1 due to satisfying Guard G4.
(ii) v generates its second run event after time $p_{v,i}$.
(iii) v transitions to RUN 0.
(iv) v generates its first output event after time $p_{v,i}$.

If (i) applies at time $\bar{t}$, we have that $\min_{w\in V_g}\{p_{w,i+1}\} \geq \bar{t} \geq t$, contradicting that we established that (i) cannot occur during $(\max_{w\in V_g}\{p_{w,i}\}+T_4+2d, \min_{w\in V_g}\{p_{w,i+1}\}+\lambda] \supset [t, \min_{w\in V_g}\{p_{w,i+1}\}]$. If (ii) applies at time $\bar{t}$, v must have transitioned to and left one of the states RUN 0 or RUN 1 during $(p_{v,i}, \bar{t})$. However, this by defintion of t this would imply that (i) or (iii) occured at v during $(p_{v,i}, \bar{t})$, contradicting the definition of $\bar{t}$. If (iii) applies at time $\bar{t}$, v must be in state RECOVER and T_5 expires at v. Since Guard G4 does not hold at v during $(\max_{w\in V_g}\{p_{w,i}\}+T_4+T_{\text{listen}}+2d, \min_{w\in V_g}\{p_{w,i+1}\}+\lambda] \supset (\max_{w\in V_g}\{p_{w,i}\}+T_4+T_{\text{listen}}+2d, \bar{t}]$, we can bound

$$\begin{aligned} \bar{t} &\leq \max_{w\in V_g}\{p_{w,i}\}+T_4+T_5+T_{\text{listen}}+2d && T_5 \text{ expires at } \bar{t}\\ &\leq p_{v,i}+\lambda+T_4+T_5+T_{\text{listen}}+2d && \text{skew } \lambda\\ &\leq p_{v,i}+\frac{T_{\text{wait}}}{\vartheta}. && (15.4) \end{aligned}$$

Because v transitioned to WAIT after time $p_{v,i}$, this entails that v generates an OUTPUT 0 event at time $\bar{t}$, i.e., (iv) applies as well at time $\bar{t}$.

Therefore, we have that $\bar{t}$ is the first time when some $v \in V_g$ generates its first output event after time $p_{v,i}$. Since (i) and (ii) are excluded during $[t, \bar{t}]$, we can conclude that the requirements of Definition 15.3 on run events hold until at least time $\min t+\tau, \bar{t}$. This, in turn, entails that the output events satisfy the constraints imposed by Definition 15.3 as well. In particular, timeliness of the recurrent consensus simulation ensures that no output event is generated before time $t+\tau$, i.e., $\bar{t} \geq t+\tau$. Given that

$$\begin{aligned}
t+\tau &\geq \min_{w\in V_g}\{p_{w,i}\} + \frac{T_4+T_5}{\vartheta} + 2(d-u) + \tau \\
\text{skew } \lambda \qquad &\geq \max_{w\in V_g}\{p_{w,i}\} + \frac{T_4+T_5}{\vartheta} + 2(d-u) + \tau - \lambda \\
\text{(15.5)} \qquad &> \max_{w\in V_g}\{p_{w,i}\} + T_4 + T_5 + T_{\text{listen}} + 2d,
\end{aligned}$$

this yields that at all $v \in V_g$ timeout T_5 expires before time $\bar{t}$. Thus, by (iii) all correct nodes transition to RUN 1 during time $[t, t+\tau]$.

Note that no correct node can generate another run event before time $\bar{t}+T_4/\vartheta > \bar{t}+\lambda$. Hence, we conclude that the conditions on run events from Definition 15.3 hold at least until time $\bar{t}+\lambda$. Using that some correct node generates an output event at time $\bar{t}$, that all correct nodes generated a INPUT 1 event during $[t, t+\tau]$, and the conditions on output events imposed by Definition 15.3, we conclude that each $v \in V_g$ generates a (unique) OUTPUT 1 event at a time $p_{v,i+1} \in [\bar{t}, \bar{t}+\lambda]$.

To complete the induction step and the proof, it remains to establish the period bounds. We have that

$$\begin{aligned}
\text{dfn. of } \bar{t} \qquad \min_{v\in V_g}\{p_{v,i+1}\} - \max_{v\in V_g}\{p_{v,i}\} &= \bar{t} - \max_{v\in V_g}\{p_{v,i}\} \\
\text{skew } \lambda \qquad &\geq \bar{t} - \min_{v\in V_g}\{p_{v,i}\} - \lambda \\
\text{lower bound on } \bar{t} \qquad &\geq t - \min_{v\in V_g}\{p_{v,i}\} + \tau - \lambda \\
\text{(15.13)} \qquad &\geq \frac{T_4+T_5}{\vartheta} + 2(d-u) + \tau - \lambda \\
\text{(15.1), (15.2), (15.5)} \qquad &> 2T_{\text{listen}} + \lambda + 3d \\
\text{(15.3)} \qquad &> 3\lambda + 3d
\end{aligned}$$

and

$$\begin{aligned}
&\max_{v\in V_g}\{p_{v,i+1}\} - \min_{v\in V_g}\{p_{v,i}\} \\
\text{dfn. of } \bar{t}\text{, skew } \lambda \qquad &\leq \bar{t} - \min_{v\in V_g}\{p_{v,i}\} + \lambda \\
\text{termination of simulation} \qquad &= t - \min_{v\in V_g}\{p_{v,i}\} + \lambda + O(\tau + Rd) \\
T_5 \text{ expires} \qquad &= \max_{w\in V_g}\{p_{w,i}\} - \min_{v\in V_g}\{p_{v,i}\} + T_4 + T_5 + T_{\text{listen}} + 2d + \lambda + O(\tau + Rd) \\
\text{skew } \lambda \qquad &= T_4 + T_5 + T_{\text{listen}} + O(\tau + \lambda + Rd) \\
\text{(15.1), (15.2), (15.5)} \qquad &= O(T_4 + T_5 + \tau + Rd)
\end{aligned}$$

□

Hence, our task is to establish the preconditions of Lemma 15.10 for some sufficiently small time t'. This means to show that the auxilliary state machine will run as intended for some sufficiently large time window, in the sense that the generated run events satisfy Definition 15.3 and Guard GG4 is not interfering with the control flow by kicking correct nodes out of RUN 0 or RUN 1. The observation towards this goal is that Guard G1 and Guard G1' represent a barrier that enforces spacing between (groups of) PULSE-WAIT transitions.

Lemma 15.11. *Suppose that $v \in V_g$ transitions to WAIT at time $t > 2T_4 + d$ and Guard G3 does not hold at correct nodes during $[t-2T_4-d, t+T_5/\vartheta-2T_4-d]$. Then no correct nodes transition to WAIT during $(t+2T_4+d, t-2T_4-d+T_5/\vartheta)$.*

Proof. Since $t > 2T_4$, in order to satisfy Guard G1 at time t, v must have actually received $n-f$ PULSE messages from distinct at of after time d. This must actually been sent at or after time 0. Thus, at least $n-2f > f$ correct nodes transitioned to state PULSE during $[t-2T_4-d, t)$.

Denote by $A \subseteq V_g$ the respective set of nodes. Any correct node satisfying Guard G1 at a time $t' > t+2T_4+d$ cannot do so based on messages nodes $w \in V_g$ sent before time t, since these are received by time $t+d$ and hence "forgotten" before time t' for the purpose of Guard G1. However, $|V \setminus A| < n-f$, so at least one node $w \in A$ must transition to PULSE again for any correct node to transition to WAIT at a time $t' > t+2T_4+d$. Since w transitioned to PULSE at or after time $t-2T_4-d$, it must have generated an OUTPUT 1 event at this time and transitioned to LISTEN. Because the prerequisites of the lemma rule out that Guard G3 holds at w during the relevant time interval, w generating another OUTPUT 1 event entails that w must satisfy Guard G7 after transitioning to INPUT 0 or either Guard G6 or Guard G6' after transitioning to INPUT 1. Either way, this involves a timeout of T_5 being reset and expiring, so this takes at least until time $t-2T_4-d+T_5/\vartheta$. □

Using this lemma, we can derive a very similar statement for correct nodes satisfying Guard G4.

Lemma 15.12. *Suppose that $v \in V_g$ satisfies Guard G4 at time $t > T_{listen} + 2T_4 + 2d$ and Guard G3 does not hold at correct nodes during $[t-T_{listen}-2T_4-2d, t+T_5/\vartheta+T_{listen}-2T_4]$. Then no correct nodes satisfy Guard G4 during $[t+T_{listen}+2T_4+2d, t-2T_4-2d+T_5/\vartheta)$.*

Proof. Node v must have received at least one WAIT message from a correct node no earlier than time $t-T_{\text{listen}}$, which was sent at a time $t_s \in [t-T_{\text{listen}}-d, t)$. Applying Lemma 15.11 to t_s, we get that no WAIT messages are sent by correct nodes during $(t_s+2T_4+d, t_s-2T_4-d+T_5/\vartheta)$. Hence, no such messages are received from correct nodes during $(t_s+2T_4+2d, t_s-2T_4-d+T_5/\vartheta) \subset [t+2T_4+2d, t-T_{\text{listen}}-2T_4-2d+T_5/\vartheta)$. It follows that Guard G4 cannot be satisfied at any correct node during $[t+T_{\text{listen}}+2T_4+2d, t-T_{\text{listen}}-2T_4-2d+T_5/\vartheta)$. □

Next, we leverage Lemma 15.12 to show that the requirements imposed by Definition 15.3 hold. First, we establish that the (global) state of the consensus simulation will be "cleared." This is implicitly expressed by having a time window of length $O(Rd)$ during which no run events are generated, as this entails that the simulation stabilizes gets any "residual" output events "out of its system."

Lemma 15.13. *Suppose that Guard G3 does not hold at correct nodes during* $[\underline{t}, \overline{t}]$*, where* $\underline{t} \geq T_{listen}+2T_4+2d$*. Moreover, transitions to states* RUN *0 and* RUN *1 are used as run events to a self-stabilizing recurrent TMP simulation of SMP binary consensus with stabilization time* $O(Rd)$*. If* $v \in V_g$ *transitions to* RUN *1 or* RUN *0 at a time* $t \in (\underline{t}+T_5+3T_{listen}+3T_4+2d, \overline{t}-(T_{listen}-2T_4)-T_5/\vartheta]$*, then correct nodes generate neither input nor output events during* $[t_0-CRd/2, t_0)$*, where*

$$t_0 := t + \left(1 - \frac{1}{\vartheta}\right) T_5 + 2T_{listen} + 2T_4 + CRd.$$

Proof. We claim that Guard G4 held at v at some time from $[\underline{t}, t]$. Assume towards a contradiction that this is not the case. Otherwise, v would have left state READ (if it was in that state at time $\underline{t}$) by time $\underline{t}+T_{\text{listen}}$ and then transitioned to either RUN 0 or RUN 1 by time $\underline{t}+T_{\text{listen}}+T_5$. This would imply that v transitions to and leave again state LISTEN before time t, as otherwise it couldn't generate a run event at time t. However, since Guard G3 does not hold during $[\underline{t}, \overline{t}] \ni t$, this requires Guard G4 to hold at v at some time from $[\underline{t}, t]$, contradicting our assumption that this is not the case. Hence, indeed Guard G4 held at v at some time from $[\underline{t}, t]$.

We set $t_v := \sup_{t' \leq t}\{\text{Guard G4 holds at v at time } t'\}$. Since T_5 keeps being reset (i.e., does not start running) at v whenever Guard G4 holds, Guard G6, Guard G6', and Guard G7 imply that v cannot transition from INPUT 0 or INPUT 1 to RUN 0 or RUN 1 (possibly again) before time $t_v + T_5/\vartheta$. Because Guard G3 cannot be satisfied at time t, this entails that

$$t \geq t_v + \frac{T_5}{\vartheta}. \tag{15.14}$$

On the other hand, v must be in one of the states READ, INPUT 0, or RUN 1 at time t_v, implying as above that

$$t \leq t_v + T_{\text{listen}} + T_5 \tag{15.15}$$

due to timeouts expiring.

Now consider any $w \in V_g$. As

$$\begin{aligned}
&[t_v - T_{\text{listen}} - 2T_4 - 2d, t_v + T_5/\vartheta + T_{\text{listen}} - 2T_4] \\
&\subset [t - T_5 - 3T_{\text{listen}} - 3T_4 - 2d, t + T_{\text{listen}} - 2T_4] && \text{(15.14), (15.15)} \\
&\subset [\underline{t}, \bar{t}], && \text{prereq. on } t
\end{aligned}$$

by Lemma 15.12 Guard G4 does not hold at w during $[t_v + T_{\text{listen}} + 2T_4 + 2d, t_v - 2T_4 - 2d + T_5/\vartheta)$. Because

- states READ, INPUT 0, and RUN 1 are vacated within $T_{\text{listen}} + T_5$ time if Guard G4 does not hold,
- transitioning from LISTEN via these states to RUN 0 or RUN 1 takes at least T_5/ϑ time, and
- Guard G3 is not satisfied at correct nodes during

$$\begin{aligned}
&\left[t_v + T_5 + 2T_{\text{listen}} + 2T_4 + 2d, t_v - 2T_4 - 2d + \frac{2T_5}{\vartheta}\right] \\
&\subset \left[t, t + \frac{T_5}{\vartheta}\right] && \text{(15.14), (15.15)} \\
&\subset [\underline{t}, \bar{t}], && \text{prereq. on } t
\end{aligned}$$

it follows that no run events are generated by correct nodes during $[t_v + T_5 + 2T_{\text{listen}} + 2T_4 + 2d, t_v - 2T_4 - 2d + 2T_5/\vartheta)$. Because the simulation of binary consensus stabilizes in time $O(Rd)$, it follows that during

$$\begin{aligned}
&\left[t_v + T_5 + 2T_{\text{listen}} + 2T_4 + O(Rd), t_v - 2T_4 - 2d + \frac{2T_5}{\vartheta}\right) \\
&\supset \left[t + \frac{(\vartheta - 1)T_5}{\vartheta} + 2T_{\text{listen}} + 2T_4 + O(Rd), t - T_{\text{listen}} - 2T_4 - 2d + \frac{(2 - \vartheta)T_5}{\vartheta}\right) && \text{(15.14), (15.15)} \\
&\supset \left[t + \frac{(\vartheta - 1)T_5}{\vartheta} + 2(T_{\text{listen}} + T_4 + \frac{CRd}{2}, t + \frac{(\vartheta - 1)T_5}{\vartheta} + 2T_{\text{listen}} + 2T_4 + CRd\right) && \text{(15.6), } C \text{ large}
\end{aligned}$$

no output events occur at correct nodes. □

Lemma 15.14. *Suppose the preconditions of Lemma 15.13 are satisfied and let t_0 be as in the statement of the lemma. Let $t \in [t_0, \bar{t}]$ be infimal with the property that a correct node generates a run event at time t (i.e., if no such event occurs, $t = \bar{t}$) and let $p \geq t_0$ be infimal with the property that a correct node generates an* OUTPUT *1 event (i.e., if no such event occurs, $p = \infty$). Then $t \leq p$, the input events that occur during $[t_0, \max\{p + T_5/\vartheta, \bar{t}\})$ satisfy the conditions imposed by Definition 15.3, and no correct node transitions to state* OUTPUT *0 while satisfying Guard G4.*

Proof. Define t_1 as the supremum of times larger or equal to $t_0 - CRd/2$ such that the run events that are generated during $[t_0, t_1]$ satisfy the conditions imposed by Definition 15.3 and no node transitions to OUTPUT 0 while satisfying Guard G4. As C is a sufficiently large constant, $CRd/2$ exceeds both the stabilization time of the consensus simulation routine and, by Equation (15.7), the time limit of $O(\tau + Rd) = O(Rd)$ imposed by Definition 15.3 on the time difference between input and output events. Note that any output event at a time from $[t_0, t_1)$ can be matched to a corresponding run event: there are no “stray” output events, since after stabilization each output event can be matched to a run event that occured at most $O(Rd)$ time in the past, while Lemma 15.13 excludes such events during $[t_0 - CRd/2, t_1)$. In particular, any output event must be preceded by a run event that occurs at or after time t_0, establishing that $t \leq p$.

Our goal now is to show that $t_1 > \max\{p + T_5/\vartheta, \bar{t}\}$, as this will prove the remaining claims of the lemma. Recall that a node generating a run event needs to first transition to LISTEN (i.e., generate an output event), then satisfy Guard G4, and finally have a timeout of T_5 expire in order to generate another input event. Given that $T_4 + T_{\text{listen}} < CRd/2$ by (15.8) and the fact that C is sufficiently large, any correct node transitioning to WAIT at a time $t' \geq t_0 - d - T_{\text{listen}}$ must do so after generating an OUTPUT 1 event later than time $t_0 - CRd/2$; otherwise, T_4 would have been expired earlier than time $t' - T_{\text{listen}}$, i.e., Guard G1 could not hold at time t'. Together with the fact that no output events are generated during $[t_0 - CRd/2, t_0)$, this entails that no WAIT message from a correct node can be received during $[t_0 - T_{\text{listen}}, p + T_4/\vartheta)$—no “fresh” OUTPUT 1 event is generated before time p and a timeout of T_4 must expire before an WAIT message is sent by a correct node. Given that Lemma 15.13 ensures that Guard G4 does not hold at correct nodes during $[t_0 - CRd/2, t_0)$, it follows that Guard G4 does not hold at such nodes during $[t_0 - CRd/2, p + T_4/\vartheta)$.

Recall that correct nodes that generate a run event need to generate an output event (in order to return to state LISTEN), which they need to leave before generating another input event. The above implies that this cannot take place before time $\min\{p + T_4/\vartheta, \bar{t}\} \geq \min\{p, \bar{t}\}$. If such a node leaves before time $\bar{t}$, this cannot be due to Guard G3 holding, so it must be due to a transition to state READ. However, this entails that a timeout of T_5 must expire before another input event occurs at the respective node. This establishes that during $[t_0, \max\{p + T_5/\vartheta, \bar{t}\})$ indeed (i) Guard G4 is not satisfied at a node in state RUN 0 or RUN 1 and (ii) no correct node generates two run events.

We distinguish two cases. Assume first that there is a RUN 1 event during $[t_0, \max\{p + T_5/\vartheta, \bar{t}\})$. As $t - T_5 - T_{\text{listen}} \geq \underline{t}$, any correct node generating such an event must have transitioned from LISTEN via READ and INPUT 1 to RUN 1. In this case, denote by $t' \geq t - T_5 - T_{\text{listen}}$ the time of the earliest corresponding transition from LISTEN to READ. At time t', the respective node satisfied Guard G5. At least $n-2f \geq f+1$ of the corresponding received WAIT messages were sent by correct nodes. Noting that $T_{\text{listen}} < T_5/\vartheta - 4T_4 - 2d$ by (15.6) and $t' > \underline{t} \geq 2T_4 + d$, Lemma 15.11 implies that the corresponding transitions to WAIT happened within a time window of length at most $2T_4 + d$. Hence, each $w \in V_g$ received $f+1$ WAIT messages from distinct senders within at most $\vartheta(2T_4 + 2d)$ local time at a time from $[t' - T_{\text{listen}} - d, t' + d]$. By Equation (15.9), each $w \in V_g$ thus satisfied Guard G4 at such a time. A node satisfying Guard G4 either transitions to or is in one of the states READ, INPUT 0, or INPUT 1. By Lemma 15.12, each such node stops satisfying Guard G4 no later than time $t' + T_{\text{listen}} + 2T_4 + 2d$. By expiring timeouts, each correct node hence transitions to either RUN 0 or RUN 1 during

$$\begin{aligned} &\left[\max\left\{t' - T_{\text{listen}} - d + \frac{T_5}{\vartheta}, t\right\}, t' + 2T_{\text{listen}} + 2T_4 + 2d + T_5\right] && \text{dfn. of } t \\ \subseteq\ &[t, t+\tau] && (15.5) \end{aligned}$$

As we already established that each node generates at most one run event during $[t_0, \max\{p + T_5/\vartheta, \bar{t}\})$, this shows that the requirements of Definition 15.3 hold during the relevant period.

Finally, consider the the case that there is no RUN 1 event during $[t_0, \max\{p + T_5/\vartheta, \bar{t}\})$. In this case, it remains to prove only that all RUN 0 events during $[t_0, \max\{p + T_5/\vartheta, \bar{t}\})$ indeed occur during $[t, t+\tau]$. If no node generates a run event at time t, the claim trivially holds, so assume some correct node does. Since at time $t - T_{\text{listen}} - T_5 \geq \underline{t}$, any correct node generating a run event during the relevant time interval must have satisfied Guard G4 no earlier than time $t - T_{\text{listen}} - T_5$. Denote by $t' \geq t - T_{\text{listen}} - T_5$ the earliest such time. By Lemma 15.11, no correct node satisfies Guard G4 during $(t' + T_{\text{listen}} + 2T_4 + 2d, t' - 2T_4 - 2d + T_5/\vartheta]$. However, since the at least $n - 2f > f$ correct nodes generating the respective messages cannot transition to WAIT again until time $p + T_4/\vartheta$, in fact no correct node can satisfy Guard G4 $(t' + T_{\text{listen}} + 2T_4 + 2d, \min\{p + T_4/\vartheta, \bar{t}\}]$. Therefore, all correct nodes that generate a run event during $[t, \min\{p + T_4/\vartheta, \bar{t}\})$ do so during

$$\begin{aligned} &\left[\max\left\{t' - T_{\text{listen}} - d + \frac{T_5}{\vartheta}, t\right\}, t' + 2T_{\text{listen}} + 2T_4 + 2d + T_5\right] && \text{dfn. of } t \\ \subseteq\ &[t, t+\tau]\ . && (15.5) \end{aligned}$$

Finally, observe that now we have established that the conditions on run events in Definition 15.3 have been established for times $[t_0 - CRd/2, \min\{p + T_4/\vartheta, \bar{t}\}]$. Thus, since the simulation of consensus stabilized, we can leverage the guarantees on output events for this time period. As we are in the case that no INPUT 1 event is generated, no OUTPUT 1 event is generated. It follows that $p \geq \bar{t}$, so we an infer that $t_1 > \max\{p + T_5/\vartheta, \bar{t}\}$ as desired. □

Corollary 15.15. *Suppose the preconditions of Lemma 15.13 are satisfied and let t_0 be as in the statement of the lemma. Then all input events that occur during $[t_0, \bar{t}\})$ satisfy the conditions imposed by Definition 15.3, and no correct node transitions to state* OUTPUT *0 while satisfying Guard G4.*

Proof. Follows from inductive use of Lemma 15.14. □

Theorem 15.16. *If* (15.1)–(15.12) *hold and the network is fully connected, the pulse synchronization algorithm given by the state machines in Figures 15.1 and 15.2 stabilizes on a good resynchronization pulse. It has skew $2d$, $P_{\min} = 3\lambda + 3d$, and $P_{\max} = O(T_4 + T_5 + \tau + Rd)$.*

Proof. Checking the inequalities, we can see that all involved parameters (except possibly for Ψ) are bounded by $O(T_5)$. Thus, as C is sufficiently large, we can apply all the above lemmas (constantly often) assuming that $\bar{t} - \underline{t}$ is large enough. We distinguish two cases. First, suppose that Lemma 15.13 is applicable (i.e., there is a sufficiently late run event). We apply Corollary 15.15, implying that if any OUTPUT 1 event is generated after time t_0, then Lemma 15.10 implies stabilization. As Ψ is large enough, in this case stabilization is complete before the good synchronization pulse occurs.

The other two cases, i.e., no sufficiently late run event or no subsequent OUTPUT 1 event occur, entail that all correct nodes transition to RECOVER at most $T_1 + T_{\text{wait}}$ time after the latest OUTPUT 1 event. Since $T_1 + T_{\text{wait}} = O(T_5)$, this means that by the time the good resynchronization pulse occurs, all correct nodes have transitioned to state RECOVER, which they cannot leave before another OUTPUT 1 event occurs or Guard G3 holds. By Definition 15.4 and (15.11), the latter happens within a time interval of length $\rho \leq \tau$. Hence, the (by then stabilized) simulation of consensus guarantees that all correct nodes generated synchronized OUTPUT 1 events. Thus, also in this case Lemma 15.10 implies stabilization.

In all cases, the skew and period bounds follow from Lemma 15.10. □

Lemma 15.17. *Suppose that $\vartheta < 3/2$ and $\rho = O(Rd)$. Then we can assign timeouts satisfying* (15.1)–(15.12) *such that $T_4 + T_5 + \tau + Rd = O(Rd)$ and $\Psi = O(Rd)$.*

E15.7 Prove the lemma. Hint: Proceed by fixing timeouts for which the right hand side of their inequalities have already been determined. (15.7), (15.8), and (15.4) then happen to be satisfied (unless you have used too generous slack in your assigments).

15.4 Providing the feedback mechanism for Algorithm 16

E15.8 How big needs B_1 to be for the approach from Chapter 13 to work? Observe that after stabilization, the state machines in Figure 15.2 can easily guarantee that no consensus instance runs earlier than B_1 time after a pulse was (locally) generated.

E15.9 Suppose that we modify the state machine in Figure 15.2 by delaying output events by at most $T_{\text{delay}} = \Theta(Rd)$ local time. When a NEXT signal is locally generated, the output event is triggered and the transition to LISTEN occurs. Does the algorithm still work correctly?

E15.10 To ensure that OUTPUT 1 events of the same instance still occur within $2d$ time of each other, modify the algorithm further by incorporating a vote-pull step in the vein of Figure 9.5 when delaying output events. Argue that for the resulting algorithm Theorem 15.16 holds again.

E15.11 By choosing M large enough, (13.8) can be easily satisfied. Argue that the mechanism proposed above ensures that also B_3 can be made large enough (provided that ϑ is not too big).

15.5 Deterministic resynchronization using exponential clocks

Our first resynchronization algorithm is very simple and resource-efficient, but costly in terms of stabilization time. The idea is to have nodes consider $f + 1$ *potential* leaders in a round-robin fashion. By scaling the frequency at which this cycling through leaders happens exponentially with the node label, we can ensure that every combination occurs eventually. In particular, all nodes will agree on a correct leader eventually for at least $\vartheta(2\Psi + 3d)$ time. By having the $f + 1$ potential leaders broadcast a "resync" message every $\vartheta(\Psi + d)$ local time and correct nodes locally trigger a resynchronization pulse when receiving such a message from a node it currently considers to be the leader, we obtain a resynchronization algorithm with separation window Ψ and skew u.

Algorithm 21 Deterministic resynchronization algorithm based on local clocks at node $v \in V_g$. The algorithm assumes that $V = \{1, \dots, n\}$.

```
1: if if $v \in \{1, \dots, f+1\}$ and getH() mod $\vartheta(\Psi + d) = 0$ then
2:     broadcast ⟨resync⟩
3: end if
4: if received ⟨resync⟩ from $w \in \{1, \dots, f+1\}$ and getH()/$(2\vartheta^2(\Psi + d)(f + 1)^v)$ mod $(f+1) \in [w, w+1)$ then
5:     generate resynchronization pulse
6: end if
```

Theorem 15.18. *For any Ψ and $f < n$, there is an f-resilient deterministic resynchronization algorithm with skew u and separation window Ψ that stabilizes in $O((\Psi + d)(\vartheta(f+2))^n)$ time.*

Proof. We claim that Algorithm 21 meets the requirements. As there are at most f faults, there is a correct node $w \in \{1, \dots, f+1\}$. To show the claim, we first prove by induction on i that within any time interval $[t, t + \vartheta^{i+1}(2\Psi + 3d)(f+2)^i]$, there is subinterval of length at least $\vartheta(2\Psi + 3d)$ during which all $v \in V_g \cap \{1, \dots, i\}$ agree on w being the leader, i.e., they would generate a pulse when receiving a ⟨resync⟩ message from w. This is trivially the case for $i = 0$, since then $V_g \cap \{1, \dots, i\} = V_g \cap \emptyset = \emptyset$. Hence, assume that we have shown this for $V_g \cap \{1, \dots, i-1\}$ for $i \in \mathbb{N}_{>0}$ and consider node i. If i is faulty, nothing is to show, so assume that $i \in V_g$ and consider the time interval $[t, t + \vartheta^{i+1}(2\Psi + 3d)(f+2)^i]$ for an arbitrary $t \geq 0$. Since $\frac{dH_v}{dt} \geq 1$, at least $\vartheta^{i+1}(2\Psi + 3d)(f+2)^i$ local time passes at i. In particular, there is a subinterval of $\vartheta^{i+1}(2\Psi + 3d)(f+2)^{i-1}$ local time during which getH()$/(\vartheta^{i+1}(2\Psi + 3d)(f+1)^i) \in [w, w+1)$. Since $\frac{dH_v}{dt} \leq \vartheta$, this entails the same for a subinterval of $\vartheta^i(2\Psi + 3d)(f+2)^{i-1}$ time lying within $[t, t + \vartheta^{i+1}(2\Psi + 3d)(f+2)^i]$. By applying the induction hypothesis to this subinterval, the induction step succeeds.

To complete the proof, we apply the statement of the induction for $i = n$ for $t = 0$, showing that there is some time t' satisfying that during $[t', t' + \vartheta(2\Psi + 3d)] \subset [0, \vartheta^{n+1}(2\Psi + 3d)(f+2)^n]$, all nodes agree on w being the leader. Because more than $\Psi + d$ local time passes at w during $[t' + \vartheta(\Psi + d), t' + \vartheta(2\Psi + 2d)]$, there is a time $t'' \in [t' + \vartheta(\Psi + d), t' + \vartheta(2\Psi + 2d)]$ when w broadcasts a ⟨resync⟩ message. This message is received by all nodes during time

$$\begin{aligned} &[t'' + d - u, t'' + d] \\ \text{delay bounds} \quad &\subset [t' + \vartheta(\Psi + d), t' + \vartheta(2\Psi + 3d)]. \end{aligned}$$

Thus, each $v \in V_g$ will generate a resync pulse within $[t'' + d - u, t'' + d]$. Moreover, no $v \in V_g$ generates a resync pulse during $[t' + d, t' + \vartheta(2\Psi + 3d)] \supset [t'' - \Psi, t'' + d]$ due messages received from nodes other than w. Finally, as $\frac{dH_w}{(}dt) \leq \vartheta$, w sent no ⟨resync⟩ during $[t'' - (\Psi + d), t'')$, so none was received during $[t'' - \Psi, t'']$. We conclude that a good resynchronization pulse of skew u and separation window Ψ occurs by time

$$\begin{aligned} t'' + d &\leq t' + \vartheta(2\Psi + 3d) \\ &\leq \vartheta^{n+1}(2\Psi + 3d)(f + 2)^n \\ &= O((\Psi + d)(\vartheta(f + 2))^n). \end{aligned}$$ □

E15.12 The above theorem holds for any $n > f$, despite the fact that we face Byzantine faults. Should we be worried that this contradicts any of the previous impossibility results?

15.6 Resynchronization using randomized timeouts for resynchronization messages

The exponential overhead of the deterministic solution comes from the need to get everyone to listen to the same leader. We can circumvent this by accepting resynchronization messages from all nodes, but only at a certain frequency, while randomizing sending times to prevent faulty nodes from interfering too much with a large probability.

Definition 15.19 (Uniformly random timeouts)**.** *A* randomized timeout *at $v \in V_g$ operates like a standard timeout that is reset to a uniformly random value from $[T_{\min}, T_{\max}]$ for some parameters $T_{\min} \leq T_{\max}$. That is, if the timout is associated with state s, it behaves as if upon each transition to s it was set to a fresh value T drawn independently and uniformly from $[T_{\min}, T_{\max}]$. However, we assume that faulty nodes cannot estimate when the timeout expires at v any better than by the (local) time passed: If the timeout was reset at time t_s and has not yet expired at time $t \geq t_s$ with $H_v(t) \in [H_v(t_s) + T_{\min}, H_v(t_s) + T_{\max}]$, then the probability density of the timeout expiring at time t' with $H_v(t') \in [H_v(t) + T_{\min}, H_v(t_s) + T_{\max}]$ equals $\frac{dH_v}{dt}(t') \cdot \frac{1}{H_v(t_s) + T_{\max} - H_v(t)}$. The behavior of faulty nodes at times t when the timeout is not expired depends only on this probability (although it can be affected by the speed of the hardware clock of v, which is known to faulty nodes in advance).*

This definition limits the power of faulty nodes in that they cannot "predict the future," at least when it comes to the random decisions of correct nodes. It is a common assumption for randomized algorithms, as many of these al-

Algorithm 22 Resynchronization algorithm based on randomized timeouts at $v \in V_g$. Nodes also "send messages to themselves."

```
 1: while true do
 2:     reset uniformly random timeout with duration from $[T_{\min}, T_{\max}]$
 3:     send ⟨propose⟩ to all nodes once timeout expires
 4: end while
 5: if received ⟨propose⟩ from $w \in V$ and timeout $T_w$ is expired then
 6:     send ⟨resync⟩ to all nodes
 7:     reset $T_w$
 8: end if
 9: if received ⟨resync⟩ from $n - f$ senders within $2\vartheta d$ local time then
10:     generate resynchronization pulse
11: end if
```

gorithms exploit that one can simply defer a decision for long enough such that the respective information becomes insufficient for a (hypothetical or real) adversary controlling the faulty nodes to prevent success of the algorithm when it is revealed. Since we are not focusing on attacks that are orchestrated by an evil mastermind, this might seem overly cautious at first. However, it is a good idea cleanly specify the assumption made and avoid a setup in which the random (or possibly pseudo-random) decisions taken by correct nodes can easily influence the behavior of the faulty nodes; even if this appears unlikely, it might cause correlated behavior breaking the algorithm.

We will show that a very simple algorithm given in Algorithm 22 meets the requirements of Theorem 15.21, where

$$\forall w \in V: \; T_w = \vartheta C(n-f)(\Psi + 3\vartheta d) \tag{15.16}$$

$$T_{\min} = \vartheta (T_w + d) \tag{15.17}$$

$$T_{\max} = 3T_{\min} + 2\vartheta C(n-f)(\Psi + 3\vartheta d)) \tag{15.18}$$

for a sufficiently large constant C.

We first show that the randomizated timeouts help to spread out the times when correct nodes attempt to initiate a good resynchronization pulse, in a way faulty nodes cannot predict.

Lemma 15.20. *Fix any interval $[t, t + C(n-f)(\Psi + 3\vartheta d)]$ with $t \geq T_{\max} + T_{\min}$ and divide it uniformly into $C(n-f)$ subintervals of length $\Psi + 3\vartheta d$. Then, with probability $1 - 2^{\Omega(n)}$ the following holds. There is a set of at least $C(n-f)/10$ non-adjacent subintervals satisfying that during the preceding subinterval, no correct node had its random timeout expire. Consider such a subinterval* *starting at time t_i. Conditioning on the execution up to time t_i, the probability that a randomized timeout at a correct node expires during $[t_i, t_i + \Psi + 3\vartheta d]$ is at least* $1/(20\lceil\vartheta^2\rceil C) = \Omega(1/C)$.

Proof. As $t \geq T_{\max} + T_{\min}$, each $v \in V_g$ generated at least one $\langle$propose$\rangle$ message, reset its randomized timeout, and had $T_{\min}$ afterwards expire. Denote by t_i the time when the i-th subinterval begins. By Definition 15.19, the probability that the randomized timeout of $v \in V_g$ expires during the i-th subinterval is bounded from above by

$$\frac{H_v(t_i + \Psi + 3\vartheta d) - H_v(t_i)}{T_{\max} - T_{\min}} \leq \frac{\vartheta(\Psi + 3\vartheta d)}{T_{\max} - T_{\min}} \qquad \frac{dH_v}{dt} \leq \vartheta$$

$$\leq \frac{1}{2C(n-f)}. \qquad (15.18)$$

For subinterval I, denote by $I^{(0)}$ the event that no randomized counter of a correct node expires. By the above bound, we get that

$$P[I^{(0)}] \geq \left(1 - \frac{1}{2C(n-f)}\right)^{n-f} \geq \left(1 - \frac{1}{2C}\right), \qquad \text{induction, each probability} \leq 1$$

where we assume w.l.o.g. that there are exactly f faults (faulty nodes could behave as if correct). Thus, dividing the $C(n-f)$ subintervals into consecutive groups of 3 intervals and applying the above probability bound, we get that in expectation at least

$$\frac{C(n-f)}{3} \cdot \frac{2C}{2C-1} \geq \frac{C(n-f)}{4} \qquad C \text{ sufficiently large}$$

groups satisfy that the first subinterval has no randomized timeout at a correct node expire. By applying a concentration bound,[9] it follows that with probability $1 - 2^{-\Omega(n)}$, this applies to at least $C(n-f)/5$ groups. We conclude that the middle subintervals of these groups satisfy the first requirement of the lemma.

It remains to establish the second requirement for at least half of these subintervals. Denote by t_v the latest time when v reset its randomized timeout prior to reaching local time $H_v(t) - T_{\min}$. By Definition 15.19, with probability at least $1/2$ it holds that

$$H_v(t_v) \geq H_v(t) - T_{\min} - \left(T_{\min} + \frac{T_{\max} - T_{\min}}{2}\right) = H_v(t) - \frac{T_{\max} + 3T_{\min}}{2}$$

[9] This is outside the scope of the course; we will add respective technical statements to an appendix of the book later.

Because

$$\begin{aligned} H_v(t + C(n-f)(\Psi + 3\vartheta d)) - H_v(t) &\leq \vartheta C(n-f)(\Psi + 3\vartheta d) \\ \text{(15.18)} \qquad &\leq \frac{T_{\max} - 3T_{\min}}{2}, \end{aligned}$$

this implies for each $t' \in [t_i, t_i + \Psi + 3\vartheta d]$ that

$$H_v(t') \in [H_v(t_v) + T_{\min}, H_v(t) + T_{\max}].$$

Thus, by Definition 15.19, overall the probability that the randomized timeout of v expires during the i-th time interval is at least

$$\begin{aligned} \tfrac{dH_v}{dt} \geq 1 \qquad \frac{1}{2} \cdot \frac{H_v(t_i + \Psi + 3\vartheta d) - H_v(t_i)}{T_{\max} - T_{\min}} &\geq \frac{\Psi + 3\vartheta d}{2(T_{\max} - T_{\min})} \\ \text{(15.18)} \qquad &= \frac{\Psi + 3\vartheta d}{4T_{\min} + 2\vartheta C(n-f)(\Psi + 3\vartheta d))} \\ \text{(15.17), (15.16)} \qquad &= \frac{\Psi + 3\vartheta d}{6\vartheta^2 C(n-f)(\Psi + 4\vartheta d))} \\ &> \frac{1}{8\vartheta^2 C(n-f)}. \end{aligned}$$

Denote by $I^{\neq 0}$ the event that at least one randomized counter of a correct node expires during subinterval I. We get

$$\begin{aligned} P[I^{(\neq 0)}] &\geq \binom{n-f}{1} \cdot \frac{1}{8\vartheta^2 C(n-f)} \cdot \left(1 - \frac{1}{8\vartheta^2 C(n-f)}\right)^{n-f-1} \\ \text{induction, each probability} \leq 1 \qquad &\geq \frac{1}{8\vartheta^2 C}\left(1 - \frac{1}{8\vartheta^2 C}\right) \\ C \text{ sufficiently large} \qquad &\geq \frac{1}{10\vartheta^2 C}, \end{aligned}$$

where we again used that w.l.o.g. we can assume that there are exactly f faults.

Note that the event $I^{(\neq 0)}$ is not independent of which timeouts expired in previous subintervals. However, conditioning on the event that indeed $C(n-f)/5$ non-adjacent subintervals satisfy that no randomized timeout expired in the preceding subinterval can only increase the probability that a timeout expires during I. Therefore, we can infer that individually, when not conditioning on the execution up to the beginning of the subinterval, each of these subintervals has a randomized timeout at a correct node expire with probability at least $\frac{1}{10\vartheta^2 C}$.

Again, applying a concentration bound[10] we can infer that with probability at least $1-2^{-\Omega(n)}$, at least $C(n-f)/10$ of the subintervals satisfying the first part of the statement of the lemma also satisfy the second part.

Finally, we apply the union bound to see that with probability at least $1-2^{-\Omega(n)}-2^{-\Omega(n)}=1-2^{-\Omega(n)}$, all required events occur concurrently. □

Theorem 15.21. *If the assumptions in Definition 15.19 hold and $3f<n$, Algorithm 22 with timeouts given by (15.16)–(15.18) is an f-resilient resynchronization algorithm with skew $2d$ that stabilizes in time $O((\Psi+d)n)$ with probability $1-2^{-\Omega(n)}$.*

Proof. To analyze the algorithm, we consider only times $t \geq T_{\max}$, i.e., each timeout that was not expired at time 0 expired at least once. Suppose $v \in V_g$ broadcasts a ⟨propose⟩ message at a time t satisfying that during $[t-\Psi-3\vartheta d, t]$, correct nodes performed in total fewer than $n-2f$ brodcasts of ⟨resync⟩ messages. In particular, no correct node received $n-f$ ⟨resync⟩ messages during $[t-\Psi-2\vartheta d, t]$. Hence, as at least $2\vartheta d$ local time passes in the same amount of real time, no correct node generates a resynchronization pulse during $[t-\Psi, t]$.

On the other hand, because each timeout expired at least once by time t, no node received a ⟨propose⟩ message from v during

$$\left[t-\frac{T_{\min}}{\vartheta}+d, t\right) \subset [t-T_w, t) \tag{15.17}$$

Thus, T_w is expired at each $v \in V_g$, implying that v broadcasts a ⟨resync⟩ message in response, no later than time $t+d$. It follows that all correct nodes receive ⟨resync⟩ messages from at least $n-f$ distinct nodes during $[t+2(d-u), t+2d]$. We conclude that all correct nodes generate a good resynchronization pulse during $[t, t+2d]$.

It remains to show that such an event occurs within $O((\Psi+d)n)$ time with probability at least $1-2^{\Omega(n)}$. By Lemma 15.20 and (15.16), with probability $1-2^{\Omega(n)}$, during $[T_{\max}+T_{\min}, T_{\max}+T_{\min}+T_w/\vartheta]$ there are at least $C(n-f)/10$ non-adjacent subintervals of length $\Psi+3\vartheta d$ during which a correct node having its randomized timeout expire results in a good resynchronization pulse, unless faulty nodes intervene. By the lemma, this occurs with probability at least $\Omega(1/C)$ for each such subinterval, even when conditioning on the events up to

[10] Here a stronger statement is needed, since we require that even after knowing which timeouts expired during the first $i-1$ subintervals, there is still a large probability that one expires in the next subinterval in question. Again, this is out of the scope of the course.

the start of the subinterval. In order to intervene, faulty nodes need to send at least $n - 2f$ $\langle$propose$\rangle$ messages over different, previously unused links to correct nodes: each such message causes $\langle$propose$\rangle$ messages to be ignored for T_w local time, which means that no other such message is accepted for T_w/ϑ time. Condition on the event that these sub-intervals exist, with the guarantees provided by the lemma.

Recall that by Definition 15.19, faulty nodes cannot predict when a randomized timeout expires at $v \in V_g$ beyond what they can determine from (perfect) knowledge of H_v and the values of $T_{\min}$ and $T_{\max}$. Thus, faulty nodes need to decide for each of the subintervals whether they send $\langle$propose$\rangle$ messages *before* they know whether a correct node broadcasts such a message or a good resynchronization pulse occurs with probability $\Omega(1/C)$. However, since the subintervals are non-adjacent and their length is larger than $\Psi + 3\vartheta d$, the messages used to interfere with one interval cannot do so for another. Moreover, there are no more than $\max_{0 \le f' \le f}\{f'(n - f')\} \le f(n - f)$ links from faulty to correct nodes. Thus, since $f < n/3$, faulty nodes can pre-emptively avoid stabilization for at most $\lfloor f(n - f)/(n - 2f)\rfloor < n - f$ of them. Therefore, a good resynchronization pulse occurs with probability at least

$$\begin{aligned}
&& 1 - (1 - \Omega(1/C))^{(C/10-1)(n-f)} &= 1 - (1 - \Omega(1/C))^{\Omega(C(n-f))} && C \text{ large}\\
&& &= 1 - e^{-\Omega(n-f)} && (1 - \tfrac{1}{C})^C \le \tfrac{1}{e}\\
&& &= 1 - 2^{-\Omega(n)}. && f < n/3
\end{aligned}$$

We conclude that a good resynchronization pulse occurs with probability $(1 - 2^{-\Omega(n)})^2 = 1 - 2^{-\Omega(n)}$ by time

$$T_{\max} + T_{\min} + \frac{T_w}{\vartheta} = O((\Psi + d)n). \qquad (15.16)\text{–}(15.18)$$

□

Bibliography

[1] Hopkins, A. L., T. B. Smith, and J. H. Lala. 1978. Ftmp – a highly reliable fault-tolerant multiprocess for aircraft. *Proceedings of the IEEE* 66 (10): 1221–1239. doi:10.1109/PROC.1978.11113.

[2] Kopetz, H. 2003. Fault containment and error detection in the time-triggered architecture. In *The sixth international symposium on autonomous decentralized systems, 2003. isads 2003.*, 139–146. doi:10.1109/ISADS.2003.1193942.

[3] Pease, M., R. Shostak, and L. Lamport. 1980. Reaching agreement in the presence of faults. *J. ACM* 27 (2): 228–234. doi:10.1145/322186.322188. http://doi.acm.org/10.1145/322186.322188.

[4] Srikanth, T. K., and Sam Toueg. 1987. Optimal clock synchronization. *J. ACM* 34 (3): 626–645. doi:10.1145/28869.28876. https://doi.org/10.1145/28869.28876.