16 Synchronous Counting

Chapter Contents

Dependency list.

1. Synchronous model and synchronous restart (logicdesign and synchronizers chapters)
2. Byzantine faults (Chapter 9)
3. Self-stabilization (Chapter 12)
4. Consensus (Chapter 14

Learning Goals

- Why studying synchronous counting is useful, and why focusing on fully connected networks is of interest.
- The relation between synchronous counting and consensus.
- A simple consensus-based algorithm with stabilization time $\Theta(f)$ for synchronous counting.
- How to efficiently generate larger counters from smaller ones.
- The relation between synchronous counting and self-stabilizing synchronous restart with permanent faults.
- A simple counting-based algorithm with stabilization and response time $\Theta(f)$ for restart with permanent faults.

- The current state-of-the-art in synchronous counting and self-stabilizing fault-tolerant synchronous restart.

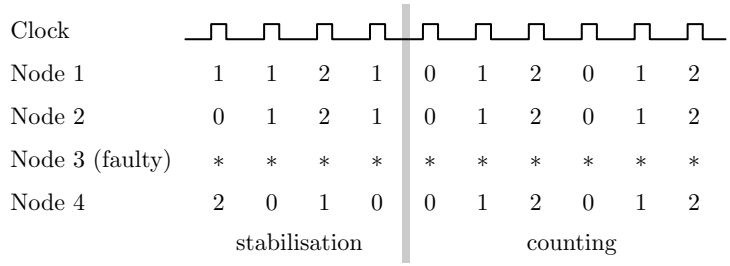

Figure 16.1
Example for the desired output of a 3-counting algorithm. After the stabilization period, the correct nodes count in synchrony; "*" indicates that the faulty node might produce arbitrary output and (inconsistently) communicate arbitrary states.

16.1 Overview

In Chaper 6, we used the synchronous abstraction to generate a shared program counter in a trivial way. In this chapter, we consider the same task in the presence of faults in the SMP model. We know from Chapter 13 that SMP can be simulated in fully connected n-node networks in a self-stabilizing manner, despite $f < \frac{n}{3}$ Byzantine faults. Matching this setting, also in this chapter we assume a fully connected network with $f < \frac{n}{3}$ Byzantine faults.

Recall that the solution in Chapter 6 was to simply maintain and increment the counter locally. Thus, the solution remains trivial even with Byzantine faults – provided that all correct nodes agree on the counter values due to correct initialization. Similarly, a self-stabilizing solution without permanent faults is easy: all we need to do is to have everyone copy the counter value of a pre-determined leader.

This situation changes drastically when asking for both self-stabilization *and* resilience to $f < \frac{n}{3}$ Byzantine faults at the same time. This gives rise to the *synchronous counting* problem, cf. Figure 16.1, which we study in this chapter.

Definition 16.1. *Denote by $V_g \subseteq V$ the set of correct nodes and fix an integer $C \geq 2$. In* synchronous C-counting, *in each synchronous round $r \in \mathbb{N}_{>0}$, each node $v \in V_g$ outputs a value $c_v(r) \in [C]$. We consider an execution* correct from round $R \in \mathbb{N}_{>0}$, *if for all rounds $r \geq R$ the following two properties hold.*

1. ***Agreement:*** *For all $v, w \in V_g$, we have that $c_v(r) = c_w(r)$.*
2. ***Validity:*** *For all $v \in V_g$, we have that $c_v(r+1) = c_v(r) + 1 \bmod C$.*

Thus, a self-stabilizing solution with stabilization time S guarantees that there is a round $R \leq S$ such that validity and agreement hold in all rounds $r \geq R$, regardless of the initial state of nodes in V_g and the behavior of nodes in $V \setminus V_g$.

Before solving the problem, we discuss its motivation in Section 16.2. We discuss how counters are useful for coordination in some tasks. We then proceed to showing a fundamental relation to consensus, which has been introduced in Chapter 14: we can utilize counting to solve consenus and vice versa.

Theorem 16.7. *Suppose that $\mathcal{A}$ solves C-counting on G with up to f Byzantine faults and stabilization time S. Then, for any set X with $|X| \leq C$, there is a consensus algorithm with inputs from X on G that tolerates up to f Byzantine faults and has round complexity S. The only communication the consensus algorithm performs is due to simulating an instance of $\mathcal{A}$ for S rounds.*

Note that Theorem 16.7 implies that lower bounds and impossibility results for consensus extend to counting.

Corollary 16.2. *C-counting with f Byzantine faults cannot be solved if the graph has $3 \leq n \leq 3f$ nodes or node connectivity smaller than $2f + 1$. Moreover, C-counting has stabilization time at least $f + 1$.*

Proof. Assuming for contradiction that a C-counting algorithm meeting one of these criteria exist, Theorem 16.7 yields a corresponding consensus algorithm. The guarantees asserted by Theorem 16.7 then contradict Theorem 14.5, Theorem 14.29, or Theorem 14.35, respectively. □

The general reduction of counting to consensus that we provide is not as efficient.

Theorem 16.8. *Suppose that $\mathcal{A}$ solves consensus on G with inputs from X, where $|X| \geq C$, for up to f Byzantine faults and with round complexity R. Then, we can solve C-counting on a fully connected n-node network with up to f faults and stabilization time $9R + 14$. The counting algorithm simulates R consensus instances concurrently and sends only the corresponding messages.*

It is important to note the mismatch between the two theorems. While Theorem 16.7 has an extremely low overhead, Theorem 16.8 runs R consensus instances concurrently. As we know from Theorem 14.35 that $R \geq f + 1$, the overhead is at least linear in f. On the other hand, for a fully connected network Corollary 14.3 shows that the overhead can also be kept within factor $O(f + \log C)$.

Corollary 16.3. *C-counting with f Byzantine faults can be solved in a fully connected network of $n > 3f$ nodes with $O(f + \log C)$-bit broadcast messages and stabilization time $O(f + \log C)$.*

Proof. By applying Theorem 16.8 to the consensus algorithm provided by Corollary 14.3. □

While this overhead might be tolerable in some cases, it is worth investigating whether more efficient solutions are possible. We remark that recursive solutions can achieve smaller asymptotic overheads than Theorem 16.8. However, the resulting algorithms are more involved, also require larger messages, and do not communicate by broadcast only. In this chapter, we will instead present a simpler and more efficient algorithm that generates large(r) counters out of smaller ones.

Concretely, assuming that we have a C-counting algorithm for $C \geq 3(f+1) + 2\left\lceil \frac{\log C'}{B} \right\rceil$, we can derive a C'-counting algorithms requiring only B-bit broadcasts on top of the communication of the C-counting algorithm.

Theorem 16.9. *Suppose that on G and with up to f Byzantine faults, $\mathcal{A}$ solves C-counting with stabilization time S and $\mathcal{B}$ solves $[C']$-valued consensus with round complexity R. If $C \geq R$, we can solve C'-counting on the same network with up to f faults and stabilization time $S + 2R + (C \bmod R)$. The C'-counting algorithm concurrently sends messages for $\mathcal{A}$ and $\mathcal{B}$ (one instance each), but performs no additional computation.*

Theorem 16.9 is, essentially, a straightforward application of consensus with inputs $X = [C']$. Plugging in the multi-valued consensus algorithm from Chapter 14, we get a way of generating large counters from smaller ones with small overhead.

Corollary 16.4. *Suppose that $\mathcal{A}$ solves C-counting with stabilization time S in a fully connected n-node network with $f < \frac{n}{3}$ Byzantine faults. Let $B \in \mathbb{N}_{>0}$, assume that $C \geq 2\left\lceil \frac{\log C'}{B} \right\rceil + 3(f+1)$, and denote by M the size of messages sent by $\mathcal{A}$. Then we can solve C'-counting on the same network with up to f faults, stabilization time $S + O\left(f + \left\lceil \frac{\log C'}{B} \right\rceil\right)$, and messages of size $B + M$. If $\mathcal{A}$ communicates by broadcast, so does the new algorithm.*

Proof. We use Theorem 14.22 for $X = [C']$ with Algorithm 18 (whose properties are stated in Theorem 14.18) and plug the resulting $[C']$-valued consensus algorithm into Theorem 16.9. □

This is easy enough, but leaves us with the question of where to get our first chicken (or egg). As we discuss in Section 16.4, it is reasonable to assume that the ability to simulate synchronous execution actually enables us to solve C-counting for reasonably small C as a by-product. Note that $B = 1$ requires even for $f = 1$ that $C = 6 + 2\lceil \log C' \rceil \geq 14$ to achieve that $C' > C$; $C = 16$

is needed to achieve that $C' \geq 2C = 32$. However, with $B = 2$, $C = 16$ is sufficient to obtain a 7-bit clock, i.e., $C' = 128$, for $f = 2$.

E16.1 What if longer clocks are needed? Can we avoid to significantly increase C or B in order to obtain a clock that runs for, say, a decade at GHz speed?

E16.2 Think about whether such clocks are useful in light of the fault model.

Next, we apply counting to solve a fault-tolerant variant of the synchronous restart problem. In contrast to Chapter 6, we cannot rely on a single node to initiate the restart, as this could result in handing control of the system to a faulty node. The definition is slightly cumbersome, which is due to the fact that we need to reign in the power of faulty nodes to influence when a restart occurs, yet need to leave enough flexibility to allow for a solution.

Definition 16.5 (Self-stabilizing Simultaneous Restart with Byzantine Faults)**.** *In each round $r \in \mathbb{N}_{>0}$, each node $v \in V_g$ receives a signal $\mathrm{go}_v(r) \in \{0,1\}$ indicating whether a restart should occur, and outputs $\mathrm{rst}_v(r) \in \{0,1\}$ indicating whether it locally restarts. From the viewpoint of the* simultaneous restart problem with (up to) f Byzantine faults, *an execution is correct with* response time T *from round R on, if it satisfies the following three properties for all rounds $r \geq R$.*

- ***Agreement:*** $\mathrm{rst}_v(r) = \mathrm{rst}_w(r)$ *for all $v, w \in V_g$.*
- ***Safety:*** *If $\mathrm{rst}_v(r) = 1$ for $v \in V_g$, (i) there is $r_{\mathrm{go}} \in \{r - T, \dots, r\}$ and $w \in V_g$ such that $\mathrm{go}_w(r_{\mathrm{go}}) = 1$ and (ii) $\mathrm{rst}_v(r') = 0$ for all $r' \in \{r_{\mathrm{go}}, \dots, r - 1\}$.*
- ***Liveness:*** *If $\mathrm{go}_v(r) = 1$ for at least $f + 1$ nodes $v \in V_g$, then $\mathrm{rst}_v(r') = 1$ for some $r' \in \{r, \dots, r + T\}$ and all $v \in V_g$.*

Algorithm $\mathcal{A}$ is a simultaneous restart algorithm resilient to f Byzantine faults with stabilization time S and response time T, *if all of its executions with at most f Byzantine faults are correct with response time T from round S on, regardless of the state of (correct) nodes on initialization.*

We remark that there is some flexibility regarding how the task is phrased. There is no strong reason to prefer this particular variant, but the presented techniques can be easily adjusted to similar formulations. We discuss this in more detail in Section 16.5.

Again, the synchronous restart problem with faults is intimately connected to consensus. First, we show that any synchronous restart algorithm can be used to solve binary consensus, essentially without any overhead.

Theorem 16.10. *Suppose that $\mathcal{A}$ is a simultaneous restart algorithm resilient to f Byzantine faults with response time T. Then there is a binary consensus*

algorithm resilient to f Byzantine faults with round complexity T. Compared to $\mathcal{A}$, the consensus algorithm requires no additional communication and only negligible additional computation.

The fact that the statement of the theorem does not refer to the stabilization time of $\mathcal{A}$ is no coincidence: The statement also applies to non-stabilizing solutions. This also holds for the proof, as it would work also under the condition that the initial states of (correct) nodes is under the control of the algorithm.

Concerning the reverse direction, we provide an efficient reduction to a counting and a binary consensus algorithm.

Theorem 16.11. *Suppose that $\mathcal{A}$ is a binary consensus algorithm of round complexity T that is resilient to f Byzantine faults, and that $\mathcal{B}$ is a C-counting algorithm for $C \geq T$ with stabilization time S that is resilient to f Byzantine faults. Then there is a simultaneous restart algorithm resilient to f Byzantine faults with stabilization time $S+2C$ and response time $2C$. Apart from running an instance of $\mathcal{B}$ and (simple) local computations, the algorithm has correct nodes send messages and perform computations for at most one instance of $\mathcal{A}$ in each round.*

Recall that we know from Theorem 16.8 that counting can be reduced to consensus, and from Theorem 14.22 that consensus can be reduced to binary consensus. Thus, from Theorem 16.11 we can conclude that self-stabilizing synchronous restart can be reduced to binary consensus as well.

Corollary 16.6. *Suppose that $\mathcal{A}$ is a binary consensus algorithm of round complexity T that is resilient to f Byzantine faults. Then there is a simultaneous restart algorithm resilient to f Byzantine faults with stabilization and response times of $O(T)$. The restart algorithm runs $O(T)$ concurrent instances of $\mathcal{A}$, but otherwise performs no communication and few additional local computations.*

In contrast to counting, the reduction to consensus is efficient both in terms of communication and computation. However, it is worth noting that this depends on the specific formulation of the simultaneous restart problem. For instance, one could require that a restart is performed in round r if and only if there where sufficiently many go signals in round $r - T$ for a fixed T. This is equivalent to solving one binary consensus for *each* round. The formulation we chose circumvents this by allowing slack in terms of the number of rounds between go signals and letting only go signals *after* the most recent restart trigger the next restart.

16.2 Problem Statement and Motivation

Synchronous counting is the task of maintaining a common round counter in a self-stabilizing and fault-tolerant manner. This means two things: (i) nodes should agree on the counter value and (ii) the counter value should increase by one in each synchronous round. This is captured by the agreement and validity conditions in the specification of the task.

Definition 16.1. *Denote by $V_g \subseteq V$ the set of correct nodes and fix an integer $C \geq 2$. In* synchronous C-counting, *in each synchronous round $r \in \mathbb{N}_{>0}$, each node $v \in V_g$ outputs a value $c_v(r) \in [C]$. We consider an execution* correct from round $R \in \mathbb{N}_{>0}$, *if for all rounds $r \geq R$ the following two properties hold.*

1. ***Agreement:*** *For all $v, w \in V_g$, we have that $c_v(r) = c_w(r)$.*
2. ***Validity:*** *For all $v \in V_g$, we have that $c_v(r+1) = c_v(r) + 1 \bmod C$.*

As pointed out earlier, what makes synchronous counting difficult is the need to overccome both transient and permanent faults, i.e., self-stabilization and Byzantine faults, respectively. The requirement to recover from transient faults necessitates to adjust node's own counter when it appears to be out-of-sync with others, as it cannot always be trusted. The requirement to tolerate (ongoing) Byzantine faults means that no single other node's clock can be used as common reference to ensure agreement.

The main purpose of solving synchronous counting is to schedule execution of algorithms in a reliable manner. This can take two forms. One are regular "maintenance" tasks, e.g., making sure that different parts of a chip that perform some task redundantly actually agree on the state of the computation, or regularly taking some measurement. Any subroutine that should be executed in regular intervals qualifies. If we want to execute a task that requires (up to) R rounds of computation every $T \geq R$ rounds, we can do so using a C-counting algorithm for any C that is a multiple of T: node $v \in V_g$ simply checks in each round r whether $c_v(r) \bmod T = 0$ and locally initializes the algorithm executing the task if this is the case. Note that one might choose T substantially larger than R, in order to save energy or allow for other task competing for the same computational resources to be executed. For instance, accesses to a communication channel or routing schemes may rely on a pattern repeating in time; in such a case, synchronous counting is key for chip-wide communication to be re-established after transient faults.

The second use case is for scheduling tasks that are triggered by external or internal events. An archetypical example is the synchronous restart task; depending on whether nodes locally receive a signal indicating the need for a restart, they need to globally agree on a time when to execute it synchronously.

Definition 16.5 (Self-stabilizing Simultaneous Restart with Byzantine Faults)**.** *In each round $r \in \mathbb{N}_{>0}$, each node $v \in V_g$ receives a signal $\mathrm{go}_v(r) \in \{0,1\}$ indicating whether a restart should occur, and outputs $\mathrm{rst}_v(r) \in \{0,1\}$ indicating whether it locally restarts. From the viewpoint of the* simultaneous restart problem with (up to) f Byzantine faults, *an execution is correct with* response time T *from round R on, if it satisfies the following three properties for all rounds $r \geq R$.*

- ***Agreement:*** $\mathrm{rst}_v(r) = \mathrm{rst}_w(r)$ *for all $v, w \in V_g$.*
- ***Safety:*** *If $\mathrm{rst}_v(r) = 1$ for $v \in V_g$, (i) there is $r_{\mathrm{go}} \in \{r - T, \dots, r\}$ and $w \in V_g$ such that $\mathrm{go}_w(r_{\mathrm{go}}) = 1$ and (ii) $\mathrm{rst}_v(r') = 0$ for all $r' \in \{r_{\mathrm{go}}, \dots, r-1\}$.*
- ***Liveness:*** *If $\mathrm{go}_v(r) = 1$ for at least $f+1$ nodes $v \in V_g$, then $\mathrm{rst}_v(r') = 1$ for some $r' \in \{r, \dots, r+T\}$ and all $v \in V_g$.*

Algorithm $\mathcal{A}$ is a simultaneous restart algorithm resilient to f Byzantine faults with stabilization time S and response time T, *if all of its executions with at most f Byzantine faults are correct with response time T from round S on, regardless of the state of (correct) nodes on initialization.*

E16.3 Adjust Definition 16.5 for non-stabilizing algorithms and argue that Theorem 16.10 also applies to this task.

E16.4 Show that the non-stabilizing version of the task can be reduced to binary consensus, where a round complexity of T translates to a response time of $O(T)$.

This definition of the restart problem is more involved than the vanilla variety due to taking into account faults. The safety and liveness properties strike a balance between reacting to external events and avoiding that faulty nodes can produce a false response. Here the assumption is that it is safe to execute a task, if at least one correct node believes this to be required, while one must execute the task when $f+1$ correct nodes believe this (which is sufficient to prove to all correct nodes that it is safe to perform the task).

Essentially, in Section 16.5, we show how to solve this task based on a "maintenance" task as described above. By regularly executing consensus on whether there were sufficiently many go signals, the system can respond in (as we show) a number of rounds which is optimal up to constants. As this solution is based on a (self-stabilizing) solution to synchronous counting, it simply inherits the self-stabilization property.

Recall that the synchronous restart task is not limited to restarting the system. It can be used to trigger any task based on external inputs to the nodes. Hence, Definition 16.5 can be seen as providing a fault-tolerant and self-stabilizing

command interface to the system running the synchronous restart routine. Note also that, if desired, it is straightforward to make the entity providing the commands redundant and provide the go signals over independent channels, enabling extension of the strong fault-tolerance properties to the compound system.

With all of this in mind, we can see that synchronous counting is a core primitive for designing highly robust general-purpose systems. The intimate connection to consensus shown in Section 16.4 reaffirms the statement from Chapter 14 that consensus is one (if not the) most fundamental fault-tolerance primitive.

We conclude this section with some remarks on C. For the above purposes, one should expect that values of C between 100 to 10000 should suffice for most basic tasks, even if they should be executed not too often to reduce energy consumption. Hence, 7-bit ($C = 128$) or 16-bit ($C = 65536$) counters should be sufficient for most cases. However, one might object that in a system running at, say, 3 GHz, even a 16-bit counter would overflow about every 20 microseconds. At first glance, this is hardly enough to use the clock for coordination with the outside world – what if some task should be executed only once per second, or once per day? While it is tempting to conclude that one should invoke Theorem 16.9 to generate sufficiently large clocks – adding just one bit would do the trick – one should keep in mind that the synchronous counting problem does *not* ensure any fixed relation between the internal counters and the external world. Even if such a relation is ensured at initialization, transient faults may cause a complete loss of this information within the system. Therefore, if a task requires a notion of time that is shared with some external system, we need to solve clock synchronization in the compound system in a self-stabilizing way. The utility of synchronous counting here is limited to supporting this, e.g. by making it possible to regularly execute consensus or other subroutines to perform this synchronization.

16.3 Relation to Consensus

Theorem 16.7. *Suppose that $\mathcal{A}$ solves C-counting on G with up to f Byzantine faults and stabilization time S. Then, for any set X with $|X| \leq C$, there is a consensus algorithm with inputs from X on G that tolerates up to f Byzantine faults and has round complexity S. The only communication the consensus algorithm performs is due to simulating an instance of $\mathcal{A}$ for S rounds.*

Proof. W.l.o.g., we assume that $X = [|X|] \subseteq [C]$; otherwise, simply fix a bijection between X and $[|X|]$ and apply it as appropriate.

Algorithm 23 Consensus algorithm based on counting algorithm at node $v \in V_g$. Recall that $x_v \in X$ is the input value of $v \in V_g$ for the consensus instance.

```
1: execute S rounds of A, where the initial state is given by state_v(x_v)
2: o := c(S) − S mod C, where c is the output function of A
3: if o ∈ [|X|] then
4:     return o
5: else
6:     return 0
7: end if
```

In order to construct our consensus algorithm, we first fix an arbitrary fault-free execution $\mathcal{E}$ of $\mathcal{A}$. To this end, we pick arbitrary initial states for each $v \in V$, which determines $\mathcal{E}$, as all nodes follow the algorithm. We simulate the first $S+C-1$ rounds of $\mathcal{E}$. Because $\mathcal{A}$ has stabilization time S, the values $c_v(r)$ nodes output in rounds $r \in \{S, \dots, S+C-1\}$ satisfy agreement and validity. As these are exactly C rounds, it follows that for each $c \in [C]$, there is exactly one round $r_c \in \{S, \dots, S+C-1\}$ such that $c_v(r_c) = c$ for all $v \in V$.

Denote for $c \in [C]$ by $\text{state}_v(c)$ the state of v at the end of round r_c of $\mathcal{E}$. Our proof rests on the following key claim. If we initialize each node $v \in V_g$ to $\text{state}_v(c)$ and $|V \setminus V_g| \leq f$, the resulting execution $\mathcal{E}'$ will satisfy validity right from the start – regardless of the behavior of faulty nodes. To see that this is true, consider the execution $\mathcal{E}''$ that is identical to $\mathcal{E}$ until round r_c (i.e., correct nodes have the same initial state as in $\mathcal{E}$ and faulty nodes send the same messages as in $\mathcal{E}$), and in rounds $r > r_c$ has faulty nodes send the same messages as in round $r - r_c$ of $\mathcal{E}'$. By induction, the state of each node $v \in V_g$ in round r of $\mathcal{E}'$ is identical to the state it has in round $r_c + r$ of $\mathcal{E}''$. In particular, v outputs the same value in rounds r of $\mathcal{E}'$ and round $r_c + r$ or $\mathcal{E}''$. Because $r_c \geq S$ and $|V \setminus V_g| \leq f$, $\mathcal{E}''$ satisfies validity in rounds $r \geq r_c$, implying that $\mathcal{E}'$ satisfies validity in all rounds $r \in \mathbb{N}$. This proves the claim.

We will exploit that the claim implies that each $v \in V_g$ outputs $c + S \bmod C$ in round S of $\mathcal{E}'$, i.e., initializing each $v \in V_g$ to $\text{state}_v(c)$ guarantees that each $v \in V_g$ satisfies $c_v(S) = c + S \bmod C$. This gives rise to the consensus algorithm shown in Algorithm 23.

By construction, this algorithm has round complexity S and outputs values from $X = [|X|]$. As $\mathcal{A}$ has stabilization time S, by agreement of $\mathcal{A}$ we have that $o_v(S) = o_v(S)$ for all $v, w \in V_g$, implying agreement of the consensus algorithm. Regarding validity, suppose that there is $x \in X$ so that $x_v = x$ for all $v \in V_g$. Then by the above claim, $o_v(S) = x + S \bmod C$ for all $v \in V_g$, resulting

in $o_v(S) - S \bmod C = x$. Accordingly, indeed each $v \in V_g$ then outputs x, proving validity. □

We remark that the computational overhead of this transformation is small. In addition to the computations of $\mathcal{A}$, nodes need to (i) translate the input into an initial state for $\mathcal{A}$, (ii) count until S, and (iii) compute the output from $c(S)$. If C is large, (i) might require a large lookup table, namely if there is no compact representation of suitable initial states for $\mathcal{A}$. However, for the important case of $X = \{0, 1\}$, all steps become very efficient: (i) requires only two different initial states, (ii) needs an S-counter, with $\mathcal{A}$ likely making use of similar counters, and (iii) can be performed by testing whether $c(S) = S + 1 \bmod C$, returning 1 if this is true and 0 otherwise. This makes it very likely that overhead in terms of computations and memory compared to just running $\mathcal{A}$ is small.

Theorem 16.8. *Suppose that $\mathcal{A}$ solves consensus on G with inputs from X, where $|X| \geq C$, for up to f Byzantine faults and with round complexity R. Then, we can solve C-counting on a fully connected n-node network with up to f faults and stabilization time $9R + 14$. The counting algorithm simulates R consensus instances concurrently and sends only the corresponding messages.*

Proof. W.l.o.g., we assume that $X = [C]$; otherwise we can enumerate the elements of X and interpret each as its index modulo C in this enumeration. Moreover, we assume that $R \bmod C \neq 0$ and show the claim of the theorem with stabilization time $9R+5$. The case $R \bmod C = 0$ is then covered by noting that any consensus algorithm with round complexity R is also a consensus algorithm with round complexity $R + 1$, and $R + 1 \bmod C \neq 0$ if $R \bmod C = 0$.

We instantiate $\mathcal{A}$ once per round, using the generated outputs with a delay of exactly R rounds to produce a stream of one output value $c_v(r)$ at each $v \in V_g$, where $r \in \mathbb{N}_{>0}$. Because initial states are arbitrary, we have no guarantee on the generated output values in rounds $1, \dots, R - 1$, nor that the respective instance of $\mathcal{A}$ actually *does* halt and output any value. However, using simple consistency checks and outputting a default value if needed, we can make sure that each correct node generates an output from $[C]$ in each round and terminate execution of an instance of $\mathcal{A}$ locally if it runs for too long. By agreement of $\mathcal{A}$, this ensures that in each round $r \geq R$, all correct nodes generate the same output value from $[C]$, which we refer to as $c(r)$. By having each $v \in V_g$ choose its input $x_v(r)$ to the instance of $\mathcal{A}$ started in round r as function of the preceding $2R + 2$ output values it perceived (or whatever is stored in the respective memory locations), after at most $3R+2$ rounds, all correct nodes use the same input value in each round. Thus, validitiy of $\mathcal{A}$ ensures that in rounds $r \geq 4R + 2$, $c(r)$ equals the agreed-upon input in round $r - R$, cf. Figure 16.2.

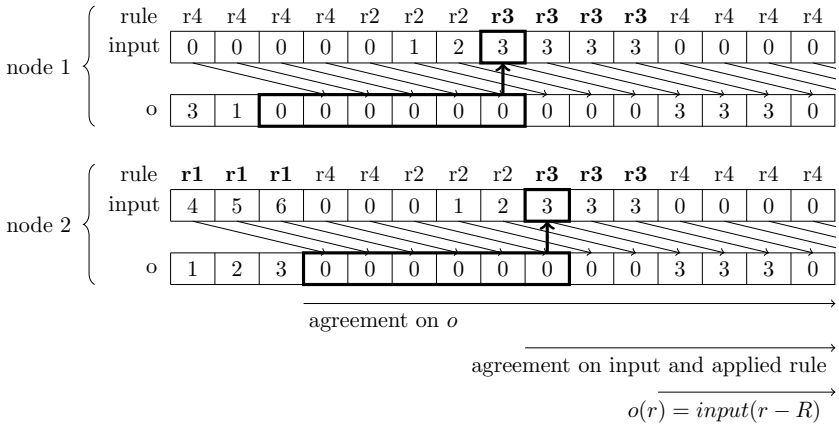

Figure 16.2

CL: Warning: Figure is out of date! Part of an execution of Algorithm 23 at two nodes, for $C = 8$ and $R = 3$. The execution progresses from left to right, each box representing a round. On top of the input field the applied rule (1 to 4) to compute the input is displayed. Displayed are the initial phases of stabilization: (i) after R rounds agreement on the output is guaranteed by consensus, (ii) after $2R + 2$ more rounds agreement on the applied rule and hence the input is reached, and (iii) another R rounds later the agreed-upon outputs are the agreed-upon inputs shifted by $R = 3$ rounds.

Denote by $\text{input}\colon [C]^{2R+2} \to [C]$ the input function we use, i.e., $x_v(r) := \text{input}(c_v(r-2R-2), c_v(r-2R-1), \ldots, c_v(r-1))$. In the following, all our arguments will refer to rounds $r \geq R$ exclusively, so we can simplify notation by using $c(r)$ in lieu of $c_v(r)$.

Our goal is to choose the input function such that $c(r)$ starts to properly count modulo C within $9R + 5$ rounds, i.e., node $v \in V_g$ using $c_v(r)$ as the local output of the C-counting algorithm in round r satisfies validity. As we already observed that $c_v(r) = c_w(r) = c(r)$ for $r \geq R$, this will complete the proof. We specify input as follows, where the function values are taken modulo R:

$$\text{input}(c(r-3R-1), \ldots, c(r-1)) := \begin{cases} c + R - 1 & \text{if } (c(r-R-2), \ldots, c(r-1)) = (c-R-1, \ldots, c-1) & \text{rule } (a) \\ x + R - 1 & \text{if } (c(r-R-2-x), \ldots, c(r-1)) = (0, \ldots, 0, 1, \ldots, x) \text{ for } x \in \{1, \ldots, R\} & \text{rule } (b) \\ x & \text{if } (c(r-R-2-x), \ldots, c(r-1)) = (0, \ldots, 0) \text{ for maximal } x \in [R+1] & \text{rule } (c) \\ 0 & \text{else.} & \text{rule } (d) \end{cases}$$

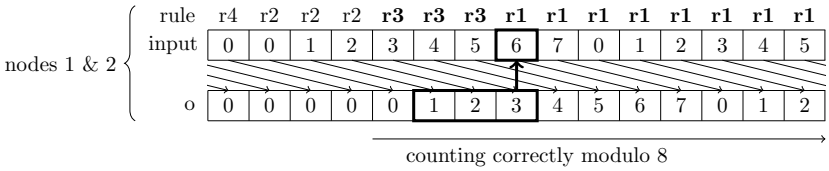

Figure 16.3
CL: Warning: Figure is out of date! Extension of the execution shown in Figure 16.2. Nodes have already agreed upon inputs and outputs so that the latter just reproduce the inputs from R rounds ago. The rules now make sure that the nodes start counting modulo 8 in synchrony, always executing rule 1.

See Figure 16.3 for an example of how these rules play out. Observe first that the input function is well-defined, i.e., exactly one of the rules (a) to (d) applies for any string of $2R+2$ numbers from $[C]$.

E16.5 Verify that rules (a) to (c) are mutually exclusive.

It remains to prove that proper counting commences at the latest by round $8R+5$. We make a case distinction.

Case 1: $(c(r'), c(r'+1)) \neq (0,0)$ and $c(r'+1) \neq c(r')+1 \bmod C$ for some $r' \geq R$. Then rules (a) to (c) do not apply in rounds $r \in \{r'+2, \dots, r'+R+2\}$, as $r-R-2 \leq r'$. Thus, all correct nodes use input 0 in these rounds, implying by validity of $\mathcal{A}$ that $c(r) = 0$ for all $r \in \{r'+R+1, \dots, r'+2R+1\}$. Denote by r_0 the infimum of all rounds larger than $r'+R+2$ such that input 0 is used in rounds $r'+2, \dots, r_0$. Note that $c(r) = 0$ for all $r \in \{r'+R+1, \dots, r_0\}$, so eventually rule (c) applies for $x \neq 0$, showing that $r_0 \leq r'+2R+3$. By validity of $\mathcal{A}$, we get that $c(r) = 0$ for all $r \in \{r'+R+1, r_0+R-1\}$. As rule (c) applies for $x = 0$ in round r_0 and x is maximal, it follows that for each $x \in [R]$, rule (c) applies for x in round r_0+x. We keep repeating the same argument, showing that (i) the previous statement holds also for $x = R$, (ii) for $x \in \{1, \dots, R\}$, rule (b) applies in round r_0+R+x, and (iii) rule (a) applies in all rounds $r \geq r_0+2R+1$. Thus, we have shown that $c(r)$ properly counts at the latest starting from round $r_0+R-1 \leq r'+3R+2$.

Case 2: $c(r'+1) = c(r')+1 \bmod C$ for some $r' \geq R$.

Case 2a: $c(r+1) = c(r)+1 \bmod C$ for all $r \in \{r', \dots, r'+2R+1\}$. By induction, it holds that in all rounds $r \geq r'+R+2$, rule (a) applies and $c(r+R-1) = c(r)+R-1$. Thus, $c(r)$ counts correctly from round r' on.

Case 2b: $c(r+1) \neq c(r)+1 \bmod C$ for some $r \in \{r'+1, \dots, r'+2R+1\}$. Choose $r_{\neq}$ as the minimal such r. As also $c(r_{\neq}-1) \neq c(r_{\neq})$, rules (a), (b),

and (except for $x = 0$) (c) do not apply in rounds $r \in \{r_{\neq}+2, \dots, r_{\neq}+R+2\}$. Hence we can proceed analogously to Case 1 with $r_{\neq}$ playing the role of r', showing that proper counting starts by round $r_{\neq} + 3R + 2 \leq r' + 5R + 3$.

Case 3**:** $(c(R), c(R+1)) = (0, 0)$. Let $r_{\neq}$ be the infimal round larger than R such that $c(r_{\neq}) \neq 0$. We claim that $r_{\neq} \leq 4R + 2$. To see this, observe that $(c(R), \dots, c(4R+1)) = (0, \dots, 0)$ in particular implies that rule (c) applies in round $3R + 2$ for some $x \neq 0$. Note that this rule does not depend on output values before round R, regardless of the value of x that applies. Accordingly, all nodes used the same input $x \bmod C$ in this round, and validity of $\mathcal{A}$ implies that $c(4R+1) = x \bmod C$. If $x \bmod C \neq 0$, this shows the claim. Otherwise, $x \neq R$, as we assumed that $R \bmod C \neq 0$. Thus, in round $3R + 3$, rule (c) applies for $x + 1$, again by validity of $\mathcal{A}$ implying that $c(4R + 2) = x + 1 \neq 0 \bmod C$. This proves the claim.

Case 3*a***:** $c(r_{\neq}) \neq 1$. Thus, Case 1 applies for round $r_{\neq}$, yielding that $c(r)$ counts at the latest starting from round $r_{\neq} + 3R + 2 \leq 7R + 4$.

Case 3*b***:** $c(r_{\neq}) = 1$. Thus, Case 2 applies for round $r_{\neq}$, yielding that $c(r)$ counts at the latest starting from round $r_{\neq} + 5R + 3 \leq 9R + 5$.

Finally, observe that one of the Cases 1, 2, and 3 must apply to round R, each of which imply proper counting starting at the latest in round $9R + 5$. □

16.4 Communication-efficient Large Counters from Small Counters

Theorem 16.9. *Suppose that on G and with up to f Byzantine faults, $\mathcal{A}$ solves C-counting with stabilization time S and $\mathcal{B}$ solves $[C']$-valued consensus with round complexity R. If $C \geq R$, we can solve C'-counting on the same network with up to f faults and stabilization time $S + 2R + (C \bmod R)$. The C'-counting algorithm concurrently sends messages for $\mathcal{A}$ and $\mathcal{B}$ (one instance each), but performs no additional computation.*

Proof. We run an instance of $\mathcal{A}$ and use the generated counters to repeatedly and consistently execute $\mathcal{B}$. Once the C-counters stabilized, $\mathcal{B}$ will be executed correctly, ensuring by agreement of consensus that the variables holding the local values of the C'-counter agree. To satisfy validity of the counters, we increase the respective local variable c' by 1 mod C' in each round. Using $c' + R - 1 \bmod C'$ as input to each consensus instance and setting c' to the (local) output of the instance exactly $R - 1$ rounds later, agreement of the counters when the instance is initiated and the validity of $\mathcal{B}$ imply that the counters are not changed by the output of the consensus instance.

This strategy results in Algorithm 24:

Algorithm 24 C'-counting algorithm based on R-round $[C']$-valued consensus algorithm $\mathcal{B}$ and C-counting algorithm $\mathcal{A}$, at $v \in V_g$ in round $r \in \mathbb{N}_{>0}$. Local variables are persistent. $\mathcal{A}$ runs in the background and has output variable c.

```
1:  c' ← c' + 1 mod C'
2:  if c mod R = 0 then
3:      initialize B (for f and G) with input c' + R − 1 mod C'
4:  end if
5:  execute round c + 1 mod R of B (if terminated, do nothing)
6:  if c mod R = R − 1 then
7:      denote by o the output variable of the local instance of B
8:      c' ← o
9:  end if
10: return c'
```

Because $\mathcal{A}$ has stabilization time S, the counters $c_v(r)$, $v \in V_g$ and $r \in \mathbb{N}$, satisfy agreement and validity in rounds $r \geq S$. Accordingly, in rounds $r \geq S$ all correct nodes agree on when they initialize a fresh instance of $\mathcal{B}$ and in which round of the execution of $\mathcal{B}$ they are. Hence, each such instance simulates a correctly initialized execution of $\mathcal{B}$, implying that agreement and validity apply to the output variables when they are used in round $r \geq S + R - 1$ (with respect to the inputs determined in round $r - R + 1$). In particular, if r_0 is the first such round, we get that $c'_v(r_0) = c'_w(r_0)$ for all $v, w \in V_g$. Moreover, induction on the round number shows that future executions of Line 8 do not change c'_v at any $v \in V_g$ and $c'_v(r+1) = c'_v(r) + 1 \bmod C'$ for all $v \in V_g$. Hence $c'_v(r+1) = c'_v(r) + 1 \bmod C'$ for all $v \in V_g$, i.e., the c'_v count correctly from round r_0 on.

Thus, to show that Algorithm 24 has stabilization time $S + 2R + (C \bmod R)$, it suffices to show that $r_0 \leq S + 2R + (C \bmod R)$. By validity of the counters c_v, $v \in V_g$, in rounds $r \geq S$, the largest number of consecute rounds in which $c_v(r) \bmod C \neq R - 1$ is $C \bmod R + R - 1$. Thus, $r_0 \in \{S + R - 1, \ldots, S + 2R - 1 + (C \bmod R)\}$, as claimed. □

16.5 Relation to Synchronous Restart

Theorem 16.10. *Suppose that $\mathcal{A}$ is a simultaneous restart algorithm resilient to f Byzantine faults with response time T. Then there is a binary consensus algorithm resilient to f Byzantine faults with round complexity T. Compared to $\mathcal{A}$, the consensus algorithm requires no additional communication and only negligible additional computation.*

Proof. We first simulate an execution of $\mathcal{A}$ without any faults, in which $\mathrm{go}_v(r) = 0$ for all $v \in V$ and $r \in \mathbb{N}_{>0}$, while the initial states are arbitrary. Denote by state_v the state of $v \in V$ in round S, where S is the stabilization time of $\mathcal{A}$.

Algorithm 25 Binary consensus algorithm from self-stabilizing simultaneous restart algorithm, code for node $v \in V_g$.

```
1: locally initialize an instance of A with state state_v
2: for T rounds do
3:     simulate a round of A with go_v(r) = b for each r ∈ N_{>0}
4:     if A computed go = 1 then
5:         return 1
6:     end if
7: end for
8: return 0
```

Observe first that in the simulated execution, any subset of up to f nodes could be Byzantine nodes that chose to behave like correct nodes. Therefore, for any feasible set $V_g \subseteq V$ of correct nodes, the initial state of the simulated instance of $\mathcal{A}$ in Algorithm 25 is equal to that after S rounds of an execution of $\mathcal{A}$. In particular, the simulated execution satisfies the agreement, safety, and liveness properties stated in Definition 16.5 in all rouns $r \in \mathbb{N}_{>0}$.

We conclude that Algorithm 25 is indeed a binary consensus algorithm with round complexity T (cf. Definition 14.1):

- Algorithm 25 satisfies agreement due to agreement of $\mathcal{A}$.
- Algorithm 25 satisfies validity, because (i) safety of $\mathcal{A}$ entails that $\mathrm{rst}_v(r) = 0$ for all $v \in V_g$ and $r \in \mathbb{N}_{>0}$ (and hence the output is 0) if all $v \in V_g$ have input 0 (and hence $\mathrm{go}_v = 0$) and (ii) liveness of $\mathcal{A}$ entails that for each $v \in V_g$ there is some $r \leq T$ such that $\mathrm{rst}_v(r) = 1$ (and hence the output is 1) if all $v \in V_g$ have input 1 (and hence $\mathrm{go}_v = 1$).
- Algorithm 25 unconditionally terminates within T rounds. □

Theorem 16.11. *Suppose that $\mathcal{A}$ is a binary consensus algorithm of round complexity T that is resilient to f Byzantine faults, and that $\mathcal{B}$ is a C-counting algorithm for $C \geq T$ with stabilization time S that is resilient to f Byzantine faults. Then there is a simultaneous restart algorithm resilient to f Byzantine faults with stabilization time $S+2C$ and response time $2C$. Apart from running an instance of $\mathcal{B}$ and (simple) local computations, the algorithm has correct*

Algorithm 26 Restart algorithm from T-round binary consensus algorithm $\mathcal{A}$ and C-counting algorithm $\mathcal{B}$ with output $c(r)$ in round $r \in \mathbb{N}_{>0}$, code for round r at node $v \in V_g$. In each round r, v receives input $\mathrm{go}_v(r)$ and outputs $\mathrm{rst}_v(r)$. It maintains variable g_v to store whether it believes a restart should be performed on the next consensus instance. The algorithm assumes that $C \geq T$.

```
1:  if go(r) = 1 then
2:      broadcast ⟨ propose ⟩
3:  end if
4:  if received ⟨ propose ⟩ from f + 1 distinct senders (including self) then
5:      g ← 1
6:  end if
7:  if c(r) = 0 then
8:      initialize A with input g (clear all prior local state of A)
9:      g ← 0
10: end if
11: if c(r) ∈ [T] then
12:     locally simulate round c(r) + 1 of A
13: end if
14: if c(r) = T − 1 and output variable of A holds 1 then
15:     rst(r) ← 1
16:     g ← 0
17: else
18:     rst(r) ← 0
19: end if
```

nodes send messages and perform computations for at most one instance of $\mathcal{A}$ in each round.

Proof. The idea is to repeatedly run consensus to agree on whether a reset needs to be performed. Running consensus can be done using the counters provided by $\mathcal{B}$, which eventually stabilize. Algorithm 26 provides the respective code.

We need to show that agreement, safety, and liveness hold for rounds $r \geq S + O(C)$. Consider rounds $r \geq S + C$. For such rounds, the counters stabilized by round $r - C$, and hence the local output variables of the consensus instance hold the outputs of a consistent execution of $\mathcal{A}$. By agreement of $\mathcal{A}$, we clearly have $\mathrm{rst}_v(r) = \mathrm{rst}_w(r)$ for each $v, w \in V_g$, i.e., agreement is satsisfied.

For safety, consider round $r \geq S + 2C$. If the counters have value different from $T - 1$, no correct $v \in V_g$ will set $\mathrm{rst}_v(r)$ to 1, so assume that this is the case. Observe that an instance of $\mathcal{A}$ has been initialized in round $r - C - (T - 1) \geq r - 2C \geq S$ and terminated before round r. On initiliazing the instance, each

correct node $v \in V_g$ set $g_v := 0$. Since at least $f + 1$ ⟨ propose ⟩ message need to be received by v in a single round to set g_v back to 1, this can only happen if some correct nodes broadcasts such a message in round $r - 2C + 1$ or later. We distinguish four cases:

1. **No** $v \in V_g$ **has** $\mathrm{go}_v(r_{\mathrm{go}}) = 1$ **for some** $r_{\mathrm{go}} \in \{r - 2C + 1, \ldots, r\}$**.** Then all correct nodes used input 0 for the instance terminating in round r. By validity, the output variables are hence 0 and $\mathrm{rst}_v(r) = 0$ for each $v \in V_g$.
2. **Some** $v \in V_g$ **has** $\mathrm{go}_v(r_{\mathrm{go}}) = 1$ **for some** $r_{\mathrm{go}} \in \{r - 2C + 1, \ldots, r\}$ **and the previous instance had output** 0**.** Then no $v \in V_g$ set $\mathrm{rst}_v(r') = 1$ for any $r' \in \{r_{\mathrm{go}}, \ldots, r - 1\}$.
3. **No** $v \in V_g$ **has** $\mathrm{go}_v(r_{\mathrm{go}}) = 1$ **for some** $r_{\mathrm{go}} \in \{r - C + 1, \ldots, r\}$ **and the previous instance had output** 1**.** Then each correct $v \in V_g$ set $g_v := 0$ in the round when the instance terminated. Note that this happens after potentially setting $g_v := 1$ in the same round. As $v \in V_g$ receives no ⟨ propose ⟩ messages from correct nodes in rounds $r - C + 1, \ldots, r$, it will not set g_v back to 1. Hence, the instance terminating in round r had all correct nodes use input 0 and by validity outputs 0. Therefore, $\mathrm{rst}_v(r) = 0$ for each $v \in V_g$.
4. **Some** $v \in V_g$ **has** $\mathrm{go}_v(r_{\mathrm{go}}) = 1$ **for some** $r_{\mathrm{go}} \in \{r - C + 1, \ldots, r\}$**.** Since the previous instance terminated in round $r - C$, we have that no $v \in V_g$ set $\mathrm{rst}_v(r') = 1$ for any $r' \in \{r_{\mathrm{go}}, \ldots, r - 1\}$.

In all cases, safety is satisfied in round r for response time $2C$.

Now consider liveness for a round $r \geq S + C$, i.e., at least $f + 1$ nodes $v \in V_g$ satisfy $\mathrm{go}_v(r) = 1$. Thus, they broadcast ⟨ propose ⟩ in round r, and each $v \in V_g$ sets $g_v(r) := 1$. We distinguish 2 cases:

1. **Some** $v \in V_g$ **sets** $g_v(r') := 0$ **in round** $r' \in \{r, \ldots, r+C-1\}$ **when setting** $\mathrm{rst}_v(r') := 1$**.** As we already established agreement, then all correct nodes $w \in V_g$ satisfy $\mathrm{rst}_w(r') = 1$.
2. **No** $v \in V_g$ **sets** $g_v(r') := 0$ **in round** $r' \in \{r, \ldots, r + C - 1\}$ **when setting** $\mathrm{rst}_v(r') := 1$**.** Thus, each correct $v \in V_g$ still satisfies $g_v(r'') = 1$ when initializing $\mathcal{A}$ in the round $r'' \in \{r, \ldots, r + C - 1\}$ when $c_v(r'') = 0$. Thus, each $v \in V_g$ uses input 1 for the instance initialized in this round. By validity of $\mathcal{A}$, its output variable is hence 1 in round $r' := r''+T-1 \leq r+2C$. We conclude that $\mathrm{rst}_v(r') = 1$.

In both cases, liveness is satisfied in round r for response time $2C$. We conclude that the algorithm has stabilization time $S + 2C$ and response time $2C$, as claimed. □

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.