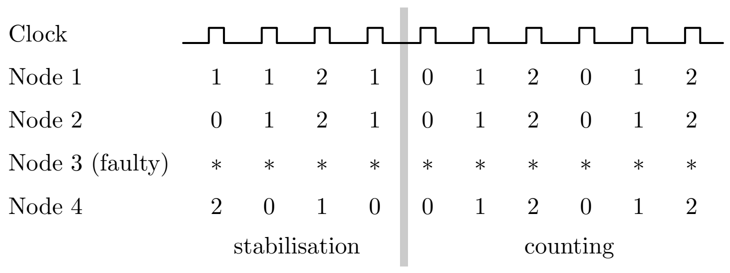
Lecture 11

Synchronous Counting

Before getting to self-stabilizing pulse synchronization in the next lecture, we consider the related task of *synchronous counting.* In synchronous counting, the goal is to establish a self-stabilizing joint counter (modulo some $2 \leq C \in \mathbb{N}$), despite $f < n/3$ Byzantine faults. This means the good traces are those in which for each round r, it holds for all $v, w \in V_g$ that $c(v,r) = c(w,r)$ and $c(v,r+1) = c(v,r) + 1 \bmod C$.



Clock										
Node 1	1	1	2	1	0	1	2	0	1	2
Node 2	0	1	2	1	0	1	2	0	1	2
Node 3 (faulty)	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$
Node 4	2	0	1	0	0	1	2	0	1	2
	stabilisation				counting					

Despite being instructive for the approach we'll take to pulse synchronization in the next lecture, this is in itself a very useful subroutine. Once the synchronous abstraction is established by a pulse synchronization algorithm, it makes sense to ask for a common numbering of the pulses, allowing for implicit coordination. For instance, this way the nodes can call a subroutine every C rounds without further communication overhead.

11.1 Synchronous Counting vs. Consensus

The first observation is that counting is no easier than (synchronous) consensus.

Lemma 11.1. *A synchronous C-counting algorithm with stabilization time S implies a synchronous C-valued consensus algorithm terminating in S rounds, which satisfies the same bounds on message and bit complexity as the original counting algorithm.*

Proof. Once stabilized, the counting algorithm guarantees a good trace, i.e., the correct nodes will jointly count modulo C. For each $c \in [C]$, denote by $\vec{x}(c)$ the state vector of the correct nodes in some round $r \geq S$ in which the count is

c. Our consensus algorithm now operates as follows. Each $v \in V_g$ runs a local instance of the counting algorithm for S rounds, where given input $c \in [C]$ it initializes its state to $x_v(c)$. At the end of round S, it outputs $c(v,S) - S \bmod C$.

The claims about running time and communication complexity are trivially satisfied, so it remains to show agreement and validity. Agreement is immediate from the fact that the counting algorithm stabilized no later than round S, i.e., $c(v,S) = c(w,S)$ for all $v, w \in V_g$. Concerning validity, observe that the initialization ensures that if each $v \in V_g$ has input $c \in [C]$, then the initial state of the counting algorithm is $\vec{x}(c)$. As this is a system state *after* stabilization, regardless of the behavior of faulty nodes, the correct nodes must increment their counters by exactly 1 modulo C in the following rounds. Thus, in round S, it holds that $c(v,S) = c + S \bmod C$, and each $v \in V_g$ outputs $c + S - S \bmod C = c$. □

The other direction is not as straightforward. However, it is not hard to come up with a reduction that translates running time to stabilization time if we neglect communication.

Lemma 11.2. *Any synchronous C-valued consensus algorithm terminating in R rounds implies a synchrounous C-counting algorithm with stabilization time $\mathcal{O}(R)$.*

Proof. Given the consensus algorithm, we solve C-counting as follows. In each synchronous round, we start a new consensus instance that will generate an output value $c(v, r+R)$ at each node $v \in V_g$ exactly R rounds later (which will double as node v's counter value); if the consensus instance terminates earlier at v, it will simply store the output value until it is needed. Note that, while we have no guarantees about the outputs in the first R rounds (as initial states are arbitrary), in all rounds $r \geq R$ all correct nodes will output the same value $c(r) = c(v,r)$ (by the agreement property of consensus). Hence, if we define the input value $f(v,r)$ of node $v \in V_g$ as a function of the most recent $\mathcal{O}(R)$ output values at node v, after $2R$ rounds all nodes will start using identical inputs $f(r) = f(v,r)$ and, by validity of the consensus algorithm, reproduce these inputs as output R rounds later (cf. Figure 11.1). In light of these considerations, it is sufficient to determine an input function f from the previous $\mathcal{O}(R)$ outputs to values $[C]$ so that counting starts within $\mathcal{O}(R)$ rounds, assuming that the output of the consensus algorithm in round $r+R$ equals the input determined at the end of round r.

We define the following input function, where all values are taken modulo C:

$$\text{input}(r) := \begin{cases} c+R & \text{if} \quad (o(r-R+1),\dots,o(r)) = (c-R+1,\dots,c) \\ x+R & \text{if} \quad \begin{array}{l}(o(r-2R+1-x),\dots,o(r)) = (0,\dots,0,1,\dots,x) \\ \text{for some } x \in [R]\end{array} \\ x & \text{if} \quad \begin{array}{l}(o(r-R+1-x),\dots,o(r)) = (0,\dots,0) \\ \text{for maximal } x \in [R]\end{array} \\ 0 & \text{else.} \end{cases}$$

In the setting discussed above, it is straightforward to verify the following properties of input:

- Always exactly one of the rules applies, i.e., input is well-defined.

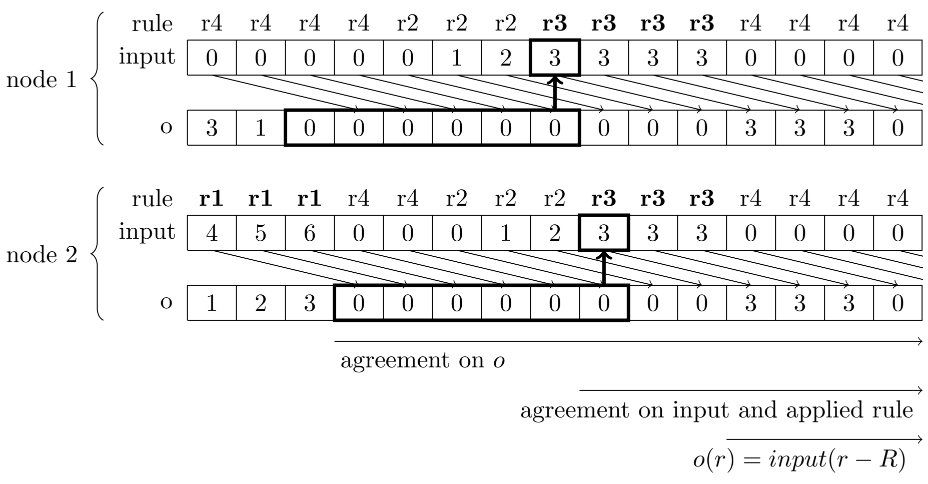

node 1
rule r4 r4 r4 r4 r2 r2 r2 r3 r3 r3 r3 r4 r4 r4 r4
input 0 0 0 0 0 1 2 3 3 3 3 0 0 0 0
o 3 1 0 0 0 0 0 0 0 0 0 3 3 3 0
node 2
rule r1 r1 r1 r4 r4 r2 r2 r2 r3 r3 r3 r4 r4 r4 r4
input 4 5 6 0 0 0 1 2 3 3 3 0 0 0 0
o 1 2 3 0 0 0 0 0 0 0 0 3 3 3 0
agreement on o
agreement on input and applied rule
$o(r) = input(r - R)$

Figure 11.1: Part of an execution of two nodes running the C-counting algorithm given in the proof of Lemma 11.2, 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) then agreement on the input and the applied rule is reached, and (iii) another R rounds later the agreed upon outputs are the agreed upon inputs shifted by 3 rounds.

- If the outputs counted modulo C for $2R$ consecutive rounds, they will do so forever (by induction, using the first rule); cf. Figure 11.2.
- If this does not happen within $\mathcal{O}(R)$ rounds, there will be R consecutive rounds where input 0 will be used (by the third and the last rule), cf. Figure 11.2.
- Once R consecutive rounds with input 0 occurred, inputs $1, \ldots, 2R$ will be used in the following $2R$ rounds (by the second and third rule).
- Finally, the algorithm will commence counting correctly (by the first rule).

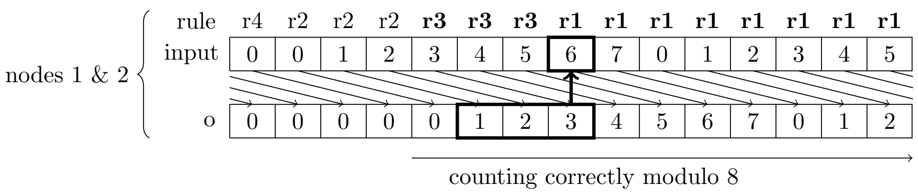

nodes 1 & 2
rule r4 r2 r2 r2 r3 r3 r3 r1 r1 r1 r1 r1 r1 r1 r1
input 0 0 1 2 3 4 5 6 7 0 1 2 3 4 5
o 0 0 0 0 0 1 2 3 4 5 6 7 0 1 2
counting correctly modulo 8

Figure 11.2: Extension of the execution shown in Figure 11.1. 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.

Overall, if each node i computes its input from its local view of the previous outputs using input, the algorithm will start counting correctly within $S \in \mathcal{O}(R)$ rounds. $\square$

Remarks:

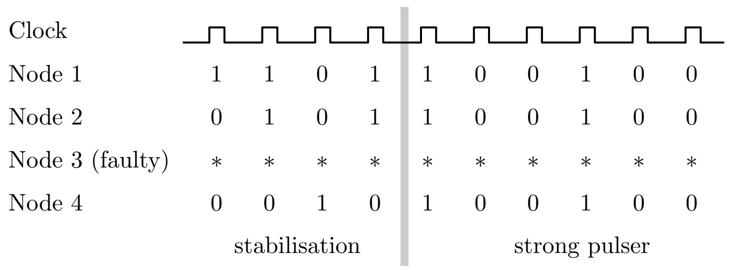
- The second reduction shows that the time complexities of both tasks are, up to a constant factor, identical.
- However, reduction from counting to consensus is inefficient in terms of communication and computation, as there are always R consensus instances running concurrently.
- Resolving this issue will be more challenging, as we can't simply circumvent the issue that the correct nodes don't agree on round numbers any more when using consensus as a subroutine without starting a new instance each round.

11.2 Pulsers

As useful tools, we introduce two tasks that are closely related to counting, but not exactly the same. The first one is, essentially, just slightly rephrasing the counting task.

Definition 11.3 (Strong pulser)**.** *An algorithm P is an f-resilient strong Ψ-pulser that stabilizes in $S(P)$ rounds if it satisfies the following conditions in the presence of at most f faulty nodes. Each node $v \in V_g$ produces an output bit $p(v, r) \in \{0, 1\}$ on each round $r \in \mathbb{N}$. We say that v generates a pulse in round r if $p(v, r) = 1$ holds. We require that there is a round $r_0 \leq S(P)$ such that:*

1. *For each $v \in V_g$ and round $r = r_0 + k\Psi$, where $k \in \mathbb{N}_0$, it holds that $p(v, r) = 1$.*
2. *For each $v \in V_g$ and round $r \geq r_0$ satisfying $r \neq r_0 + k\Psi$ for $k \in \mathbb{N}_0$, we have $p(v, r) = 0$.*



Node 1	1	1	0	1	1	0	0	1	0	0
Node 2	0	1	0	1	1	0	0	1	0	0
Node 3 (faulty)	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$
Node 4	0	0	1	0	1	0	0	1	0	0

Figure 11.3: An example execution of a strong 3-pulser on $n = 4$ nodes with $f = 1$ faulty node.

Lemma 11.4. *Let $C \in \mathbb{N}$ and $\Psi \in \mathbb{N}$. If C divides Ψ, then a strong Ψ-pulser that stabilizes in S rounds implies a synchronous C-counter that stabilizes in at most S rounds. If Ψ divides C, then a synchronous C-counter that stabilizes in S rounds implies a strong Ψ-pulser that stabilizes in at most $S + \Psi - 1$ rounds.*

Proof. For the first claim, set $c(v, r) = 0$ in any round r for which $p(v, r) = 1$ and $c(v, r) = c(v, r - 1) + 1 \bmod C$ in all other rounds. For the second claim, set $p(v, r) = 1$ in all rounds r in which $c(v, r) \bmod \Psi = 0$ and $p(v, r) = 0$ in all other rounds. □

Remarks:

- Another way of interpreting this relation is to view a strong Ψ-pulser as a different encoding of the output of a Ψ-counter: since the system is synchronous, it suffices to communicate when the counter overflows to value 0 and otherwise count locally. This saves bandwidth when communicating the state of the counter.
- The additive overhead of Ψ will not matter to us, as we will recursively construct strong pulsers, deriving a counter only in the very end.

A weak Φ-pulser is similar to a strong pulser, but does not guarantee a fixed frequency of pulses. However, it guarantees to *eventually* generate a pulse followed by $\Phi - 1$ rounds of silence.

Definition 11.5 (Weak pulsers)**.** *An algorithm W is an f-resilient weak Φ-pulser that stabilizes in $S(W)$ rounds if the following holds. In each round $r \in \mathbb{N}$, each node $v \in V_g$ produces an output $a(v, r)$. Moreover, there exists a round $r_0 \leq S(W)$ such that*

1. *for all $v, w \in V_g$ and all rounds $r \geq r_0$, $a(v, r) = a(w, r)$,*
2. *$a(v, r_0) = 1$ for all $v \in V_g$, and*
3. *$a(v, r) = 0$ for all $v \in V_g$ and $r \in \{r_0 + 1, \ldots, r_0 + \Phi - 1\}$.*

We say that on round r_0 a good *pulse is generated by W.*

Figure 11.4 illustrates a weak 4-pulser.

Remarks:

- While the definition formally only asks for one good pulse, the fact that the algorithm guarantees this property for any starting state implies that there is a good pulse at least every $S(W)$ rounds.
- Weak pulsers are (surprise!) easier to construct than strong pulsers. Yet, they are good enough to *eventually* get a consensus instance to be executed correctly, using the good pulse as starting shot for the execution of the consensus algorithm. This we can use to stabilize a strong pulser.

Constructing Strong Pulsers from Weak Pulsers

For constructing a strong Ψ-pulser, we assume that we have the following f-resilient algorithms available:

- an $R(C)$-round Ψ-valued consensus algorithm C and
- a weak Φ-pulser W for some $\Phi \geq R(C)$.

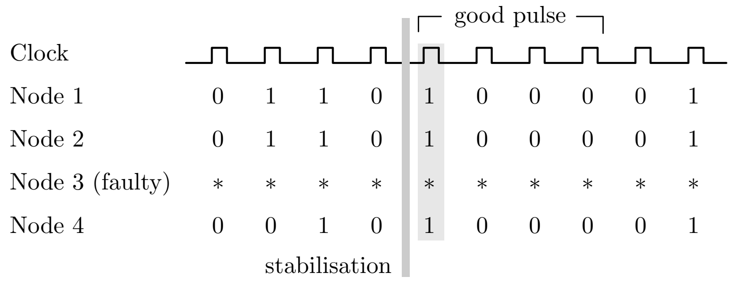

good pulse
Clock
Node 1 0 1 1 0 1 0 0 0 0 1
Node 2 0 1 1 0 1 0 0 0 0 1
Node 3 (faulty) * * * * * * * * * *
Node 4 0 0 1 0 1 0 0 0 0 1
stabilisation

Figure 11.4: An example execution of a weak 4-pulser on $n = 4$ nodes with $f = 1$ faulty node. Eventually, a good pulse is generated, which is highlighted. A good pulse is followed by three rounds in which no correct node generates a pulse. In contrast, the pulse two rounds earlier is not good, as it is followed by only one round of silence.

Variables. Beside the variables of the weak pulser W and (a single copy of) C, our construction of a strong Ψ-pulser uses the following local variables:

- $a(v, r) \in \{0, 1\}$ is the output variable of the weak Φ-pulser W,
- $b(v, r) \in \{0, 1\}$ is the output variable of the strong Ψ-pulser we are constructing,
- $c(v, r) \in [\Psi]$ is the local counter keeping track on when the next pulse occurs, and
- $d(v, r) \in \{1, \ldots, R(C)\} \cup \{\perp\}$ keeps track of how many rounds an instance of C has been executed since the last pulse from the weak pulser W. The value $\perp$ denotes that the consensus routine has stopped.

Strong pulser algorithm. The algorithm is as follows. Each node v executes the weak Φ-pulser algorithm W in addition to the following instructions on each round $r \in \mathbb{N}$:

1. If $c(v, r) = 0$, then set $b(v, r) = 1$ and otherwise $b(v, r) = 0$.
2. Set $c'(v, r) = c(v, r)$.
3. If $d(v, r) \neq \perp$, then
   (a) Execute the instructions of C for round $d(v, r)$.
   (b) If $d(v, r) \neq R(C)$, set $d(v, r + 1) = d(v, r) + 1$.
   (c) If $d(v, r) = R(C)$, then
      i. Set $c'(v, r) = y(v, r) + R(C) \bmod \Psi$, where $y(v, r)$ is the output value of C.
      ii. Set $d(v, r + 1) = \perp$.
4. Update $c(v, r + 1) = c'(v, r) + 1 \bmod \Psi$.

5. If $a(v,r) = 1$, then
   (a) Start a new instance of C using $c'(v,r)$ as input (resetting all state variables of C).
   (b) Set $d(v,r+1) = 1$.

In the above algorithm, the first step simply translates the counter value to the output of the strong pulser. We then use a temporary variable $c'(v,r)$ to hold the counter value, which is overwritten by the output of C (increased by $R(C) \bmod \Psi$) if it completes a run in this round. In either case, the counter value needs to be increased by 1 mod Ψ for the next round. The remaining code does the bookkeeping for an ongoing run of C and starting a new run if the weak pulser generates a pulse.

Observe that in the above algorithm, each node only sends messages related to the weak pulser W and the consensus algorithm C. Thus, there is no additional overhead in communication and the message size is bounded by $M(W) + M(C)$, where $M(\cdot)$ denotes the maximum message size of an algorithm. Hence, it remains to show that the local counters $c(v,r)$ implement a strong Ψ-counter.

Theorem 11.6. *The variables $c(v,r)$ in the above algorithm implement a synchronous Ψ-counter that stabilizes in $S(W)+R(C)+1$ rounds and uses messages of at most $M(W)+M(C)$ bits.*

Proof. Suppose round $r_0 \leq S(W)$ is as in Definition 11.5, that is, $a(v,r) = a(w,r)$ for all $r \geq r_0$, and a good pulse is generated in round r_0. Thus, all correct nodes participate in simulating an instance of C during rounds $r_0+1, \dots, r_0+R(C)$, since no pulse is generated during rounds $r_0+1, \dots, r_0+R(C)-1$, and thus, also no new instance is started in the last step of the code during these rounds.

By the agreement property of the consensus routine, it follows that $c'(v, r_0+R(C)) = c'(w, r_0+R(C))$ for all $v, w \in V_g$ after Step 3ci. By Steps 2 and 4, the same will hold for both $c(\cdot, r')$ and $c'(\cdot, r')$, $r' > r_0+R(C)$, provided that we can show that in rounds $r' > r$, Step 3ci never sets $c'(v,r)$ to a value different than $c(v,r)$ for any $v \in V_g$; as this also implies that $c(v, r'+1) = c(v,r') + 1 \bmod \Psi$ for all $v \in V_g$ and $r' > r_0 + R(C)$, this will complete the proof.

Accordingly, consider any execution of Step 3ci in a round $r' > r_0 + R(C)$. The instance of C terminating in this round was started in round $r' - R(C) > t_0$. However, in this round the weak pulser must have generated a pulse, yielding that, in fact, $r' - R(C) \geq r_0 + R(C)$. Assuming for contradiction that r' is the earliest round in which the claim is violated, we thus have that $c'(v, r'-R(C)) = c'(w, r'-R(C))$ for all $v, w \in V_g$, i.e., all correct nodes used the same input value c for the instance. By the validity property of C, this implies that $v \in V_g$ outputs $y(v,r') = c$ in round r' and sets $c'(v,r') = c + R(C) \bmod \Psi$. However, since r' is the earliest round of violation, we already have that $c'(v,r') = c(v,r') = c+R(C) \bmod \Psi$ after the second step, contradicting the assumption and showing that the execution stabilized in round $r_0 + R(C) + 1 \leq S(W) + R(C) + 1$. □

Together with Lemma 11.4, we get the following corollary.

Corollary 11.7. *Let $\Psi > 1$. Suppose that there exists an f-resilient Ψ-value consensus routine C and a weak Φ-pulser W, where $\Phi \geq R(C)$. Then there exists an f-resilient strong Ψ-pulser P that*

- *stabilizes in time $S(P) \leq R(C) + S(W) + \Psi$, and*
- *uses messages of size at most $M(P) \leq M(C) + M(W)$ bits.*

Remarks:

- This straightforward construction reduces our task to designing weak pulsers.
- Even though we "havn't done much," constructing weak pulsers is significantly easier.
- This is an example where the hardest part is to come up with the right question, or rather problem to solve. By giving rise to the questions "can we obtain strong pulsers from weak ones?" and "can we construct weak pulsers?" the notion of weak pulsers breaks the question "can we construct strong pulsers" into (as it turns out) more managable tasks.

11.3 Weak from (less Resilient) Strong Pulsers

Having seen that we can construct strong pulsers from weak pulsers using a consensus algorithm, the missing piece is the existence of efficient weak pulsers. We now devise a recursive construction of a weak pulser from strong pulsers of smaller resilience. Given that a 0-resilient pulser is trivial and that we can obtain strong pulsers from weak ones without losing resilience, this is sufficient for constructing strong pulsers of optimal resilience from consensus algorithms of optimal resilience.

At a high level, we take the following approach (see Figure 11.5):

1. Partition the network into two parts, each running a strong pulser (with small resilience). Our construction guarantees that at least one of the strong pulsers stabilizes.
2. Filtering of pulses generated by the strong pulsers:
    a) Nodes consider the observed pulses generated by the strong pulsers as *potential* pulses.
    b) Since one of the strong pulsers may not stabilize, it may generate *spurious* pulses, that is, pulses that only a subset of the correct nodes observe.
    c) We limit the *frequency* of the spurious pulses using a filtering mechanism based on threshold voting.
3. We force any spurious pulse to be observed by either all or none of correct nodes by employing a *silent consensus* routine. In silent consensus, no message is sent (by correct nodes) if all correct nodes have input 0. Thus, if all nodes actually participating in an instance have input 0, non-participating nodes behave *as if they participated* with input 0. This avoids the chicken-and-egg problem of having to solve consensus on participation in the consensus routine. We make sure that if any node uses input 1, i.e., the consensus routine may output 1, all nodes participate. Thus, when a pulse is generated, all correct nodes agree on this.

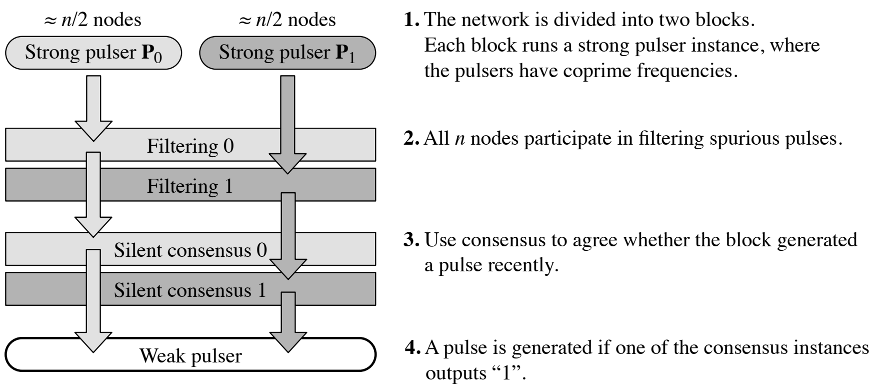

≈ n/2 nodes
≈ n/2 nodes
Strong pulser P0
Strong pulser P1
Filtering 0
Filtering 1
Silent consensus 0
Silent consensus 1
Weak pulser
1. The network is divided into two blocks. Each block runs a strong pulser instance, where the pulsers have coprime frequencies.
2. All n nodes participate in filtering spurious pulses.
3. Use consensus to agree whether the block generated a pulse recently.
4. A pulse is generated if one of the consensus instances outputs "1".

Figure 11.5: Overview of the weak pulser construction. Light and dark grey boxes correspond to steps of block 0 and 1, respectively. The small rounded boxes denote the pulser algorithms P_i that are run (in parallel) on two disjoint sets of roughly $n/2$ nodes, whereas the wide rectangular boxes denote to the filtering steps in which all of the n nodes are employed. The arrows indicate the flow of information for each block.

4. If a potential pulse generated by one of the pulsers both passes the filtering step and the consensus instance outputs "1", then a weak pulse is generated.

The Filtering Construction

Our goal is to construct a weak Φ-pulser (for sufficiently large Φ) with resilience f. We partition the set of n nodes into two disjoint sets V_0 and V_1 with n_0 and n_1 nodes, respectively. Thus, we have $n = n_0 + n_1$. For $i \in \{0,1\}$, let P_i be an f_i-resilient strong Ψ_i-pulser. That is, P_i generates a pulse every Ψ_i rounds once stabilized, granted that V_i contains at most f_i faulty nodes. Nodes in block i execute the algorithm P_i. Our construction tolerates $f = f_0 + f_1 + 1$ faulty nodes. Since we consider Byzantine faults, we require the additional constraint that $f < n/3$.

Let $a_i(v,r) \in \{0,1\}$ indicate the output bit of P_i for a node $v \in V_i$. Note that we might have a block $i \in \{0,1\}$ that contains more than f_i faulty nodes. Thus, it is possible that the algorithm P_i never stabilizes. In particular, we might have the situation that some of the nodes in block i produce a pulse, but others do not. We say that a pulse generated by such a P_i is *spurious.* We proceed by showing how to filter out such spurious pulses if they occur too often.

Filtering rules. We define five variables with the following semantics:

- $m_i(v,r{+}1)$ indicates whether at least $n_i - f_i$ nodes $u \in V_i$ sent $a_i(u,r) = 1$,
- $M_i(v,r{+}1)$ indicates whether at least $n - f$ nodes $u \in V$ sent $m_i(u,r) = 1$,

- $\ell_i(v, r)$ indicates when was the last time block i triggered a (possibly spurious) pulse,
- $x_i(v, r)$ is a *cooldown counter* that indicates how many rounds any firing events coming from block i are ignored, and
- $b_i(v, r)$ indicates whether node v accepts a firing event from block i.

The first two of the above variables are set according to the following rules:

- $m_i(v, r+1) = 1$ if and only if $|\{w \in V_i : a_i(v, w, r) = 1|\} \geq n_i - f_i$,
- $M_i(v, r+1) = 1$ if and only if $|\{w \in V : m_i(v, w, r) = 1\} \geq n - f$,

where $a_i(v, w, r)$ and $m_i(v, w, r)$ denote the values for $a(\cdot)$ and $m(\cdot)$ node v received from w at the end of round r, respectively. Furthermore, we update the $\ell(\cdot, \cdot)$ variables using the rule

$$\ell_i(v, r+1) = \begin{cases} 0 & \text{if } |\{w \in V : m_i(w, r) = 1\}| \geq f + 1, \\ y & \text{otherwise,} \end{cases}$$

where $y = \min\{\Psi_i, \ell_i(v, r) + 1\}$ (and, of course, each node $v \in V_g$ performs the update according to the count it perceives). In words, the counter is reset on round $r + 1$ if v has proof that at least one correct node w had $m_i(w, r) = 1$, that is, some $w \in V_g$ observed P_i generating a (possibly spurious) pulse.

We reset the cooldown counter x_i whenever suspicious activity occurs. The idea is that it is reset to its maximum value C by node v in the following two cases:

- some other correct node $u \neq v$ observed block i generating a pulse, but the node v did not, or
- block i generated a pulse, but this happened either too soon or too late.

To capture this behaviour, the cooldown counter is set with the rule

$$x_i(v, r+1) = \begin{cases} C & \text{if } M_i(v, r+1) = 0 \text{ and } \ell_i(v, r+1) = 0, \\ C & \text{if } M_i(v, r+1) = 1 \text{ and } \ell_i(v, r) \neq \Psi_i - 1, \\ y & \text{otherwise,} \end{cases}$$

where $y = \max\{x_i(v, r) - 1, 0\}$ and $C = \max\{\Psi_0, \Psi_1\} + \Phi + 2$. Finally, a node v accepts a pulse generated by block i if the node's cooldown counter is zero and it saw at least $n - f$ nodes supporting the pulse. The variable $b_i(v, r)$ indicates whether node v accepted a pulse from block i on round r. The variable is set using the rule

$$b_i(v, t) = \begin{cases} 1 & \text{if } x_i(v, r) = 0 \text{ and } M_i(v, r) = 1, \\ 0 & \text{otherwise.} \end{cases}$$

Analysis of the Filtering Construction

We now analyse when the nodes accept firing events generated by the blocks. We say that a block i is correct if it contains at most f_i faulty nodes. Note that since there are at most $f = f_0 + f_1 + 1$ faulty nodes, at least one block $i \in \{0, 1\}$ will be correct. Thus, eventually the algorithm P_i run by a correct block i will stabilize. This yields the following lemma.

Lemma 11.8. *For some $i \in \{0, 1\}$, the strong pulser algorithm P_i stabilizes by round $S(P_i)$.*

We proceed by establishing some bounds on when (possibly spurious) pulses generated by block i are accepted. We start with the case of having a correct block i.

Lemma 11.9. *If block i is correct, then there exists a round $t_0 \leq S(P_i) + 2$ such that for each $v \in V_g$, $M_i(v, r) = 1$ if and only if $r = t_0 + k\Psi_i$ for $k \in \mathbb{N}_0$.*

Proof. If block i is correct, then the algorithm P_i stabilizes by round $S(P_i)$. Hence, there is some $r_0 \leq S(P)$ so that the output variable $a_i(\cdot)$ of P_i satisfies

$$a_i(v, r) = 1 \text{ if and only if } r = r_0 + k\Psi_i \text{ for } k \in \mathbb{N}_0$$

holds for all $r \geq r_0$. We will now argue that $t_0 = r_0 + 2$ satisfies the claim of the lemma.

If P_i generates a pulse on round $r \geq r_0$, then at least $n_i - f_i$ correct nodes $u \in V_i \cap V_g$ have $a_i(u, r) = 1$. Therefore, for all $v \in V_g$ we have $m_i(v, r+1) = 1$, and consequently, $M_i(v, r + 2) = 1$. Since block i is correct, there are at most f_i faulty nodes in the set V_i. Observe that by Lemma 11.4 strong pulsers solve synchronous counting, which in turn is as hard as consensus (by Lemma 11.1). This implies that we must have $f_i < n_i/3$, as P_i is a strong f_i-resilient pulser for n_i nodes. Therefore, if P_i does not generate a pulse on round $r \geq r_0$, then at most $f_i < n_i - f_i$ faulty nodes w may claim $a_i(w, t) = 1$. This yields that $m_i(v, t + 1) = M_i(v, t + 2) = 0$ for all $v \in V_g$. □

We can now establish that a correct node accepts a pulse generated by a correct block i exactly every Ψ_i rounds.

Lemma 11.10. *If block i is correct, then there exists a round $t_1 \leq S(P_i) + 2C$ such that for each $v \in V_g$, $b_i(v, r) = 1$ for any $r \geq r_0$ if and only if $r = t_1 + k\Psi_i$ for $k \in \mathbb{N}_0$.*

Proof. Lemma 11.9 implies that there exists $t_0 \leq S(P_i) + 2$ such that both $M_i(v, t) = 1$ and $\ell_i(v, r) = 0$ hold for $r \geq t_0$ if and only if $r = t_0 + k\Psi_i$ for $k \in \mathbb{N}_0$. It follows that $x_i(v, r + 1) = \max\{x_i(v, r) - 1, 0\}$ for all such r and hence $x_i(v, r') = 0$ for all $r' \geq t_0 + C + 2$. The claim now follows from the definition of $b_i(v, r')$, the choice of t_0, and the fact that $\Psi_i \leq C - 2$. □

It remains to deal with the faulty block. If we have Byzantine nodes, then a block i with more than f_i faulty nodes may attempt to generate spurious pulses. However, the filtering mechanism prevents the spurious pulses from occuring too frequently.

Lemma 11.11. *Let $v, v' \in V_g$ and $t > 2$. Suppose that $b_i(v, r) = 1$ and $r' > r$ is minimal such that $b_i(v', r') = 1$. Then $r' = r + \Psi_i$ or $r' > r + C$.*

Proof. Suppose that $b_i(v,r) = 1$ for some correct node $v \in V_g$ and $r > 2$. Since $b_i(v,r) = 1$, $x_i(v,r) = 0$ and $M_i(v,r) = 1$. Because $M_i(v,r) = 1$, there must be at least $n - 2f > f$ correct nodes w such that $m_i(w, r-1) = 1$. Hence, $\ell_i(w,t) = 0$ for every node $w \in V_g$.

Recall that $r' > r$ is minimal so that $b_i(v', r') = 1$. Again, $x_i(v', r') = 0$ and $M_i(v', r') = 1$. Moreover, since $\ell_i(v', r) = 0$, we must have $\ell_i(v', t) < \Psi_i - 1$ for all $r \leq t < r + \Psi_i - 1$. This implies that $r' \geq r + \Psi_i$, as $x_i(v', r') = 0$ and $M_i(v', r') = 1$ necessitate that $\ell_i(v', r'-1) = \Psi_i - 1$. In the event that $r' \neq r + \Psi_i$, the cooldown counter must have been reset at least once, i.e., $x_i(v', t) = C$ holds for some $r < t \leq r' - C$, implying that $r' > r + C$. □

Remarks:

- The bottomline: The filtering mechanism does not interfere with the output of correct blocks, but it restricts the possible confusion arising from faulty blocks to either sticking to a fixed frequency or being eliminated completely for long enoug (i.e., C rounds).

Using Silent Consensus to Prune Spurious Pulses

The above filtering mechanism prevents spurious pulses from occurring too often: if some node accepts a pulse from block i, then no node accepts a pulse from this block for at least Ψ_i rounds. We now strengthen the construction to enforce that any (possibly spurious) pulse generated by block i will be accepted by either all or none of the correct nodes. In order to achieve this, we employ *silent consensus.*

Definition 11.12 (Silent consensus)**.** *We call a consensus protocol* silent*, if in each execution in which all correct nodes have input* 0*, correct nodes send no messages.*

The idea is that this enables to have consistent executions even if not all correct nodes actually take part in an execution, provided we can ensure that in this case all participating correct nodes use input 0: the non-participating nodes send no messages either, which is the exact same behavior participating nodes would exhibit.

Theorem 11.13. *Any consensus protocol C can be transformed into a silent binary consensus protocol C' with $R(C') = R(C) + 2$ and the same resilience and message size.*

Proof. Exercise. □

For example, plugging in the Phase King protocol, we get the following corollary.

Corollary 11.14. *For any $f < n/3$, there exists a deterministic f-resilient silent binary consensus protocol C with $R(C) \in \Theta(f)$ and $M(C) \in \mathcal{O}(1)$.*

As the filtering construction bounds the frequency at which spurious pulses may occur from above, we can make sure that at each time, only one consensus instance can be executed for each block. However, we need to further preprocess the inputs, in order to make sure that (i) all correct nodes participate in an instance or (ii) no participating correct node has input 1; here, output 1 means agreement on a pulse being triggered, while output 0 results in no action.

Recall that $b_i(v,r) \in \{0,1\}$ indicates whether v observed a (filtered) pulse of the strong pulser P_i in round r. Moreover, assume that C is a silent consensus protocol running in $R(C)$ rounds. We use two copies C_i, where $i \in \{0,1\}$, of the consensus routine C. We require that $\Psi_i \geq R(C)$, which guarantees by Lemm 11.11 that (after stabilization) every instance of C has sufficient time to complete. Adding one more level of voting to clean up the inputs, we arrive at the following routine.

The pruning algorithm. Besides the local variables of C_i, the algorithm will use the following variables for each $v \in V_g$ and round $r \in \mathbb{N}$:

- $y_i(v,r) \in \{0,1\}$ denotes the output value of consensus routine C_i,
- $t_i(v,r) \in \{1,\dots,R(C)\} \cup \{\bot\}$ is a (local) round counter for controlling C_i, and
- $B_i(v,r) \in \{0,1\}$ is the output of block i.

Recall that $b(v,r)$ indicates whether v (locally) accepts a firing event, and that $b(v,w,r)$ indicates the value v receives from w if w sends this value in round r. Each node v executes the following on round r:

1. Broadcast the value $b_i(v,r)$.
2. If $b_i(v,w,r-1) = 1$ for at least $n-2f$ nodes $w \in V$, then reset $t_i(v,r) = 1$.
3. If $t_i(v,r) = 1$, then
   (a) start a new instance of C_i, that is, re-initialise the variables of C_i correctly,
   (b) use input 1 if $b_i(v,w,r-1) = 1$ for at least $n-f$ nodes $w \in V$ and 0 otherwise.
4. If $t_i(v,r) = R(C)$, then
   (a) execute round $R(C)$ of C_i,
   (b) set $t_i(v,r+1) = \bot$,
   (c) set $B_i(v,r+1) = y_i(v,r)$, where $y_i(v,r) \in \{0,1\}$ is the output variable of C_i.

   Otherwise, set $B_i(v,r+1) = 0$.
5. If $t_i(v,r) \notin \{R(C),\bot\}$, then
   (a) execute round $t_i(v,r)$ of C_i, and
   (b) set $t_i(v,r+1) = t_i(v,r) + 1$.

Analysis. Besides the communication used for computing the values $b_i(\cdot)$, the above algorithm uses messages of size $M(C)+1$, as $M(C)$ bits are used when executing C_i and one bit is used to communicate the value of $b_i(v,r)$.

We say that $v \in V_g$ *executes the round* $t \in \{1,\ldots,T(C)\}$ of C_i in round r iff $t_i(v,r)=t$. By Lemm 11.11, in rounds $t > R(C)+2$, there is always at most one instance of C_i being executed, and if so, consistently.

Corollary 11.15. *Suppose that $v \in V_g$ executes round 1 of C_i in some round $r > T(C)+2$. Then there is a subset $U \subseteq V_g$ such that each $w \in U$ executes round $t \in \{1,\ldots,R(C)\}$ of C_i in round $r+t-1$ and no $u \in V_g \setminus U$ executes any round of C_i in round $r+t-1$.*

Exploiting silence of C_i and the choice of inputs, we can ensure that the case $U \neq V_g$ causes no trouble.

Lemma 11.16. *Let $r > T(C)+2$ and U be as in Corollary 11.15. Then $U = V_g$ or each $w \in U$ has input 0 for the respective instance of C_i.*

Proof. Suppose that $v \in U$ starts an instance with input 1 in round $r' \in \{r-T(C)-1,\ldots,r\}$. Then $b_i(w,r'-1)=1$ for at least $n-2f$ nodes $w \in V_g$, since v received $b_i(v,w,r'-1)=1$ from $n-f$ nodes $w \in V$. Thus, each $v' \in V_g$ received $b_i(v',w,r'-1)=1$ from at least $n-2f$ nodes w and sets $r_i(v',r')=1$, i.e., $U=V_g$. The lemma now follows from Corollary 11.15. □

Recall that if all nodes executing C_i have input 0, non-participating correct nodes behave exactly as if they executed C_i as well, i.e., they send no messages. Hence, if $U \neq V_g$, all nodes executing the algorithm will compute output 0. Therefore, Corollary 11.15, Lemm 11.11, and Lemm 11.16 imply the following corollary.

Corollary 11.17. *In rounds $r > T(C)+2$ it holds that $B_i(v,r)=B_i(w,t)$ for all $v,w \in V_g$ and $i \in \{0,1\}$. Furthermore, if $B_i(v,r)=1$ for $v \in V_g$ and $r > T(C)+2$, then the minimal $r' > r$ so that $B_i(v,r')=1$ (if it exists) satisfies either $r' = r+\Psi_i$ or $r' > r + C = t + \max\{\Psi_0,\Psi_1\}+\Phi+2$.*

Finally, we observe that our approach does not filter out pulses from correct blocks.

Lemma 11.18. *If block i is correct, there is a round $t_2 \leq S(P_i)+2C+R(C)+1$ so that for any $r \geq t_2$, $B_i(v,r)=1$ if and only if $r = t_2 + k\Psi_i$ for some $k \in \mathbb{N}_0$.*

Proof. Lemm 11.10 states the same for the variables $b_i(v,r)$ and a round $t_1 \leq S(P_i)+2C$. If $b_i(v,r)=1$ for all $v \in V_g$ and some round r, all correct nodes start executing an instance of C_i with input 1 in round $r+1$. As, by Corollary 11.15, this instance executes correctly and, by validity of C_i, outputs 1 in round $r+R(C)$, all correct nodes satisfy $B_i(v,r+R(C)+1)=1$. Similarly, $B_i(v,r+R(C)+1)=0$ for such v and any $r \geq t_1$ with $b_i(v,r)=0$. □

Remarks:

- The bottomline: we used consensus to enforce consistency of the outputs of correct nodes.
- In order to resolve the issue that not always all correct nodes will know to participate, we used silent consensus. If *anyone* is set on using input 1 (everything seems to be fine), all correct nodes participate. Otherwise, the participating nodes have input 0, and because the non-participating nodes do not send messages, the fact that the consensus routine is silent means that the run behaves just as if everyone participated with input 0.
- Note how this is similar to how we made the Phase King algorithm work: either the Phase King figures out that someone is stuck with value $b \in \{0, 1\}$ and broadcasts b, or no correct node is stuck with a fixed value, so it doesn't matter which value the king broadcasts.

Obtaining the Weak Pulser

Finally, we define the output variable of our weak pulser as

$$B(v, r) = \max\{B_0(v, r), B_1(v, r)\}.$$

As we have eliminated the possibility that $B_i(v, r) \neq B_i(w, r)$ for $v, w \in V_g$ and $r > R(C) + 2$, the first requirement of Definition 11.5. Since there is at least one correct block i by Lemma 11.8, Lemma 11.18 shows that there will be good pulses (satisfying the second and third requirement) regularly, unless block $1-i$ interferes by generating pulses violating the third requirement (i.e., in too short order after a pulse generated by block i). Here the filtering mechanism comes to the rescue: as we made sure that pulses are either generated at the chosen frequency Ψ_i or a long period of C rounds of generating no pulse is enforced (Corollary 11.17), it is sufficient to choose Ψ_0 and Ψ_1 as coprime multiples of Φ.

Accordingly, we pick $\Psi_0 = 2\Phi$ and $\Psi_1 = 3\Phi$ and observe that this results in a good pulse within $\mathcal{O}(\Phi)$ rounds after the B_i stabilized.

Lemma 11.19. *In the construction described in the previous two subsections, choose $\Psi_0 = 2\Phi$ and $\Psi_1 = 3\Phi$ for any $\Phi \geq R(C)$. Then $B(v, r)$ is the output variable of a weak Φ-pulser with stabilization time* $\max\{S(P_0), S(P_1)\} + \mathcal{O}(\Phi)$.

Proof. We have that $C = \max\{\Psi_0, \Psi_1\} + \Phi + 2 \in \mathcal{O}(\Phi)$. By the above observations, there is a round

$$\begin{aligned} r &\in \max\{S(P_0), S(P_1)\} + R(C) + \mathcal{O}(\Phi) \\ &\subseteq \max\{S(P_0), S(P_1)\} + \mathcal{O}(\Phi) \end{aligned}$$

satisfying the following four properties. For either block $i \in \{0, 1\}$, we have by Corollary 11.17 that

1. $B_i(v, r') = B_i(w, r')$ and $B(v, r') = B(w, r')$ for any $v, w \in V_g$ and $r' \geq r$.

Moreover, for a correct block i and for all $v \in V_g$ we have from Lemma 11.18 that

2. $B_i(v, r) = B_i(v, r + \Psi_i) = 1$,

3. $B_i(v, r') = 0$ for all $r' \in \{r+1, \dots, r+\Phi-1\} \cup \{r+\Psi_i+1, \dots, r+\Psi_i+\Phi-1\}$,

and for a (possibly faulty) block $1 - i$ we have from Corollary 11.17 that

4. if $B_{1-i}(v, r') = 1$ for some $v \in V_g$ and $r' \in \{r + 1, \dots, r + \Psi_i + \Phi - 1\}$, then $B_{1-i}(w, r'') = 0$ for all $w \in V_g$ and $r'' \in \{r' + 1, \dots, r' + C\}$ that do not satisfy $r'' = r' + k\Psi_{1-i}$ for some $k \in \mathbb{N}_0$.

Now it remains to argue that a good pulse is generated. Suppose that i is a correct block given by Lemma 11.8. By the first property, it suffices to show that a good pulse occurs in round r or in round $r + \Psi_i$. From the second property, we get for all $v \in V_g$ that $B(v, r) = 1$ and $B(v, r + \Psi_i) = 1$. If the pulse in round r is good, the claim holds. Hence, assume that there is a round $r' \in \{r + 1, \dots, r + \Psi_i - 1\}$ in which another pulse occurs, that is, $B(v, r') = 1$ for some $v \in V_g$. This entails that $B_{1-i}(v, r') = 1$ by the third property. We claim that in this case the pulse in round $r + \Psi_i$ is good. To show this, we exploit the fourth property. Recall that $C > \Psi_i + \Phi$, i.e., $r' + C > r + \Psi_i + \Phi$. We distinguish two cases:

- In the case $i = 0$, we have that $r' + \Psi_{1-i} = r' + 3\Phi = r' + \Psi_0 + \Psi > r + \Psi_0 + \Phi$, that is, the pulse in round $r + \Psi_0 = r + \Psi_i$ is good.
- In the case $i = 1$, we have that $r' + \Psi_{1-i} = r' + 2\Phi < r + 3\Phi = r + \Psi_1$ and $r' + 2\Psi_{1-i} = r' + 4\Phi = r' + \Psi_1 + \Phi > r + \Psi_1 + \Phi$, that is, the pulse in round $r + \Psi_1 = r + \Psi_i$ is good.

In either case, a good pulse occurs by round

$$r + \max\{\Psi_0, \Psi_1\} \in \max\{S(P_0), S(P_1)\} + \mathcal{O}(\Phi). \qquad \square$$

From the above lemma and the constructions discussed in this section, we get the following theorem.

Theorem 11.20. *Let $n = n_0 + n_1$ and $f = f_0 + f_1 + 1$, where $n > 3f$. Suppose that C is an f-resilient consensus algorithm on n nodes and let $\Phi \geq R(C)+2$. If there exist f_i-resilient strong Ψ_i-pulser algorithms on n_i nodes, where $\Psi_0 = 2\Phi$ and $\Psi_1 = 3\Phi$, then there exists an f-resilient weak Φ-pulser W on n nodes that satisfies*

- $S(W) \in \max\{S(P_0), S(P_1)\} + \mathcal{O}(\Phi)$,
- $M(W) \in \max\{M(P_0), M(P_1)\} + \mathcal{O}(M(C))$.

Proof. By Theorem 11.13, we can transform C into a silent consensus protocol C', at the cost of increasing its round complexity by 2. Using C' in the construction, Lemma 11.19 shows that we obtain a weak Φ-pulser with the stated stabilization time, which by construction tolerates f faults. Concerning the message size, note that we run P_0 and P_1 on disjoint node sets. Apart from sending at most $\max\{M(P_0), M(P_1)\}$ bits per round for its respective strong pulser, each node may send up to $M(C) \geq 1$ bits each to each other node for the two copies C_i of C it runs in parallel, plus a constant number of additional bits for the filtering construction including its outputs $b_i(\cdot, \cdot)$. $\square$

Remarks:

- The work is done, we merely need to chain the constructions for weak and strong pulsers recursively now.

11.4 Plugging it Together

Finally, in this section we put the developed machinery to use. As our main result, we show how to recursively construct strong pulsers out of consensus algorithms.

Theorem 11.21. *Suppose that we are given a family of f-resilient consensus algorithms $C(f)$ running on any number $n > 3f$ of nodes in $R(C(f))$ rounds using $M(C(f))$-bit messages, where both $R(C(f))$ and $M(C(f))$ are non-decreasing in f. Then, for any $\Psi \in \mathbb{N}$, $f \in \mathbb{N}_0$, and $n > 3f$, there exists a strong Ψ-pulser P on n nodes that stabilizes in time*

$$S(P) \in (1 + o(1))\Psi + \mathcal{O}\left(\sum_{j=0}^{\lceil \log f \rceil} R(C(2^j))\right)$$

and uses messages of size at most

$$M(P) \in \mathcal{O}\left(1 + \sum_{j=0}^{\lceil \log f \rceil} M(C(2^j))\right)$$

bits, where the sums are empty for $f = 0$.

Proof. We show by induction on k that f-resilient strong Ψ-pulsers $P(f, \Psi)$ on $n > 3f$ nodes with the stated complexity exist for any $f < 2^k$, with the addition that the (bounds on) stabilization time and message size of our pulsers are non-decreasing in f. We anchor the induction at $k = 0$, i.e., $f = 0$, for which, trivially, a 0-resilient strong Ψ-pulser with $n \in \mathbb{N}$ nodes is given by one node generating pulses locally and informing the other nodes when to do so. This requires 1-bit messages and stabilizes in $\Psi + 1$ rounds.

Now assume that $2^k \leq f < 2^{k+1}$ for $k \in \mathbb{N}_0$ and the claim holds for all $0 \leq f' < 2^k$. Since $2 \cdot (2^k - 1) + 1 = 2^{k+1} - 1$, there are $f_0, f_1 < 2^k$ such that $f = f_0 + f_1 + 1$. Moreover, as $n > 3f > 3f_0 + 3f_1$, we can pick $n_i > 3f_i$ for both $i \in \{0, 1\}$ satisfying $n = n_0 + n_1$. Let $P(f', \Psi')$ denote a strong Ψ'-pulser that exists by the induction hypothesis for $f' < 2^k$.

We intend to use the Ψ-valued consensus algorithm C' on n nodes resilient to f faults that we obtain from $C(f)$ as in Task 1 of Exercise 10. In order to make use of it, we need a weak Φ-pulser, where $\Phi \in \mathcal{O}(\log \Psi) + R(C(f))$ matches the time complexity of C'. Without loss of generality, we may assume that the $\mathcal{O}(\log \Psi)$ term is at least 2, that is, $\Phi \geq 2 + R(C(f))$. We apply Theorem 11.20 to $C(f)$ and $P_i = P(f_i, \Psi_i)$, where $\Psi_0 = 2\Phi$ and $\Psi_1 = 3\Phi$, to obtain a weak Φ-pulser W on n nodes with resilience f, stabilization time of

$$S(W) \in \max\{S(P_0), S(P_1)\} + \mathcal{O}(\Phi),$$

and message size of

$$M(W) \in \max\{M(P_0), M(P_1)\} + \mathcal{O}(M(C(f)))\,.$$

Recall from Task 1 of Exercise 10 that C' uses messages of size $M(C(f))$ bits and runs in $R(C') \leq \Phi$ rounds. We feed the weak pulser W and the multivalued consensus protocol C' into Corollary 11.7 to obtain an f-resilient strong Ψ-pulser P that stabilizes in

$$\begin{aligned} S(P) &\leq R(C') + S(W) + \Psi \leq S(W) + \Psi + \Phi \\ &\in \max\{S(P_0), S(P_1)\} + \Psi + \mathcal{O}(\Phi) \end{aligned}$$

rounds and has message size bounded by

$$\begin{aligned} M(P) &\leq M(W) + M(C(f)) \\ &\in \max\{M(P_0), M(P_1)\} + \mathcal{O}(M(C(f))). \end{aligned}$$

Applying the bounds given by the induction hypothesis to P_0 and P_1, the definitions of Φ, Ψ_0 and Ψ_1, and the fact that both $R(C(f))$ and $M(C(f))$ are non-decreasing in f, we get that the stabilization time satisfies

$$\begin{aligned} S(P) &\in \max\{S(P(f_0, \Psi_0)), S(P(f_1, \Psi_1))\} + \Psi + \mathcal{O}(\Phi) \\ &\subseteq (1 + o(1)) \cdot 3\Phi + \mathcal{O}\left(\sum_{j=0}^{\lceil \log 2^k \rceil} R(C(2^j)) \right) \\ &\quad + \Psi + \mathcal{O}(\Phi) \\ &\subseteq \Psi + \mathcal{O}(\log \Psi) + \mathcal{O}\left(\sum_{j=0}^{\lceil \log 2^k \rceil} R(C(2^j)) \right) \\ &\quad + \mathcal{O}(R(C(f))) \\ &\subseteq (1 + o(1))\Psi + \mathcal{O}\left(\sum_{j=0}^{\lceil \log f \rceil} R(C(2^j)) \right), \end{aligned}$$

and message size is bounded by

$$\begin{aligned} M(P) &\in \max\{M(P(f_0, \Psi_0)), M(P(f_1, \Psi_1))\} \\ &\quad + \mathcal{O}(M(C(f))) \\ &\subseteq \mathcal{O}\left(1 + \sum_{j=0}^{\lceil \log 2^k \rceil} M(C(2^j)) \right) + \mathcal{O}(M(C(f))) \\ &\subseteq \mathcal{O}\left(1 + \sum_{j=0}^{\lceil \log f \rceil} M(C(2^j)) \right). \end{aligned}$$

Because we bounded complexities using $\max_i\{S(P_i)\}$, $\max_i\{M(P_i)\}$, $R(C(f))$ and $M(C(f))$, all of which are non-decreasing in f by assumption, we also maintain that the new bounds on stabilization time and message size are non-decreasing in f. Thus, the induction step succeeds and the proof is complete. □

Plugging in the Phase King protocol, we can extract a strong pulser that is optimally resilient, has asymptotically optimal stabilization time, and message size $\mathcal{O}(\log f)$.

Corollary 11.22. *For any $\Psi, f \in \mathbb{N}$ and $n > 3f$, an f-resilient strong Ψ-pulser on n nodes with stabilization time $(1+o(1))\Psi + \mathcal{O}(f)$ and message size $\mathcal{O}(\log f)$ exists.*

Corollary 11.23. *For any $C, f \in \mathbb{N}$ and $n > 3f$, an f-resilient C-counter on n nodes with stabilization time $\mathcal{O}(f + \log C)$ and message size $\mathcal{O}(\log f)$ exists.*

Proof. In the last step of the construction of Theorem 11.21, we do not use Corollary 11.7 to extract a strong pulser, but directly obtain a counter using Theorem 11.6. This avoids the overhead of Ψ due to waiting for the next pulse. Recalling that the $o(\Psi)$ term in the complexity comes from the $\mathcal{O}(\log \Psi)$ additive overhead in time of the multi-value consensus routine, the claim follows. □

Remarks:

- The construction may look awfully complicated, but this is not the result of a high difficulty of the proof.
- Taking into account that the idea that recursion might help is borrowed from the recursive variant of the Phase King protocol, the main challenges were coming up with the idea to break the problem up into the subtasks of constructing weak and strong pulsers, and seeing that silent consensus can be used to circumvent the need for running consensus on whether to run consensus.
- The entire construction works, without any changes, with randomized consensus routines (satisfying certain constraints, which can as easily and generically be achieved as silence). In particular, this yields solutions with stabilization time $\log^{\mathcal{O}(1)} f$.
- Needless to say that you're not expected to know the full details of the construction by heart!

Bibliographic Notes

The synchronous counting problem was dubbed by Dolev and Hoch under the name of *self-stabilizing Byzantine digital clock synchronization* [HDD06]. They provide a linear-time solution based on consensus. The construction given in Lemma 11.2 is a simplification given in a later survey [DFL$^+$15]. The term "synchronous counting" came up later, because "self-stabilizing Byzantine digital clock synchronization" just takes way too long to say (try it out 10 times).

However, (another) Dolev and Welch were the ones who originally introduced the task, in the same article in which they introduce and solve self-stabilizing pulse synchronization [DW04]. They devise an exponential-time solution for counting, which they then adapt to yield an exponential-time self-stabilizing pulse synchronization algorithm. Apart from introducing the problems, this work surprised by showing that the tasks *can* actually be solved, despite the severe fault model. It also shows how the "synchronous version" of pulse synchronization can serve as a testing ground for algorithmic ideas, without the messy details of drifting clocks and uncertain communication delays, before adapting them into solutions to pulse synchronization. This was also a main motivation of the line of work [DHJ+16, LRS15, LRS17] culminating in the recursive construction presented in this lecture [LR16]: at the time it was unknown if more efficient (in particular sub-linear time) self-stabilizing solutions to pulse synchronization could be achieved, so we decided to study the synchronous counting problem as the "closest of kin" in the synchronous model.

Bibliography

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

[DHJ+16] Danny Dolev, Keijo Heljanko, Matti Järvisalo, Janne H. Korhonen, Christoph Lenzen, Joel Rybicki, Jukka Suomela, and Siert Wieringa. Synchronous Counting and Computational Algorithm Design. *Journal of Computer and System Sciences*, 82(2):310–332, 2016.

[DW04] S. Dolev and J. L. Welch. Self-Stabilizing Clock Synchronization in the Presence of Byzantine Faults. *Journal of the ACM*, 51(5):780–799, 2004.

[HDD06] Ezra Hoch, Danny Dolev, and Ariel Daliot. Self-Stabilizing Byzantine Digital Clock Synchronization. In *Proc. 8th Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, pages 350–362, 2006.

[LR16] Christoph Lenzen and Joel Rybicki. Near-Optimal Self-stabilising Counting and Firing Squads. In Borzoo Bonakdarpour and Franck Petit, editors, *Proc. Symposium on Stabilization, Safety, and Security of Distributed Systems*, pages 263–280, 2016.

[LRS15] Christoph Lenzen, Joel Rybicki, and Jukka Suomela. Towards Optimal Synchronous Counting. In *Proc. 34th Symposium on Principles of Distributed Computing (PODC)*, pages 441–450, 2015.

[LRS17] C. Lenzen, J. Rybicki, and J. Suomela. Efficient Counting with Optimal Resilience. *SIAM Journal on Computing*, 46(4):1473–1500, 2017.