Distributed Computing (2019) 32:339–360
https://doi.org/10.1007/s00446-018-0342-6

Near-optimal self-stabilising counting and firing squads

Christoph Lenzen[1] · **Joel Rybicki**[2]

Received: 17 January 2017 / Accepted: 7 September 2018 / Published online: 12 September 2018
© The Author(s) 2018

Abstract
Consider a fully-connected synchronous distributed system consisting of n nodes, where up to f nodes may be faulty and every node starts in an arbitrary initial state. In the *synchronous C-counting* problem, all nodes need to eventually agree on a counter that is increased by one modulo C in each round for given $C > 1$. In the *self-stabilising firing squad* problem, the task is to eventually guarantee that all non-faulty nodes have simultaneous responses to external inputs: if a subset of the correct nodes receive an external "go" signal as input, then all correct nodes should agree on a round (in the not-too-distant future) in which to jointly output a "fire" signal. Moreover, no node should generate a "fire" signal without some correct node having previously received a "go" signal as input. We present a framework reducing both tasks to binary consensus at very small cost. For example, we obtain a deterministic algorithm for self-stabilising Byzantine firing squads with optimal resilience $f < n/3$, asymptotically optimal stabilisation and response time $O(f)$, and message size $O(\log f)$. As our framework does not restrict the type of consensus routines used, we also obtain efficient randomised solutions.

Keywords Byzantine faults · Self-stabilisation · Clock synchronisation · Synchronous counting · Firing squads

1 Introduction

The design of distributed systems faces several unique issues related to redundancy and fault-tolerance, timing and synchrony, and the efficient use of communication as a resource [28]. In this work, we give near-optimal solutions to two fundamental distributed synchronisation and coordination tasks: the synchronous counting and the firing squad problems. For both tasks, we devise fast self-stabilising algorithms [16] that are not only communication-efficient, but also tolerate the optimal number of permanently faulty nodes. That is, our algorithms efficiently recover from transient failures that may arbitrarily corrupt the state of the distributed system *and* permanently damage a large number of the nodes.

✉ Joel Rybicki
joel.rybicki@ist.ac.at

Christoph Lenzen
clenzen@mpi-inf.mpg.de

[1] Department of Algorithms and Complexity, Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

[2] Institute of Science and Technology Austria (IST Austria), 3400 Klosterneuburg, Austria

1.1 Synchronous counting and firing squads

We assume a synchronous message-passing model of distributed computation. The distributed system consists of a fully-connected network of n nodes, where up to f of the nodes may be faulty and the initial state of the system is arbitrary. There exist various ways to model permanent faults, including e.g.:

- crashes (the faulty node stops sending information),
- omissions (some or all of the messages sent by faulty nodes can be lost), and
- Byzantine faults (the faulty node exhibits arbitrary misbehaviour).

Here we focus on Byzantine faults, as this is the strongest model of permanently faulty behaviour.

Note that even though the communication proceeds in a synchronous fashion, the nodes may have different notions of current time due to the arbitrary initial states. However, many typical distributed protocols assume that the system has either been properly initialised or that the nodes should collectively agree on the rounds in which to perform certain actions. Thus, we are essentially faced with the task of having to agree on a common time in a manner that is both

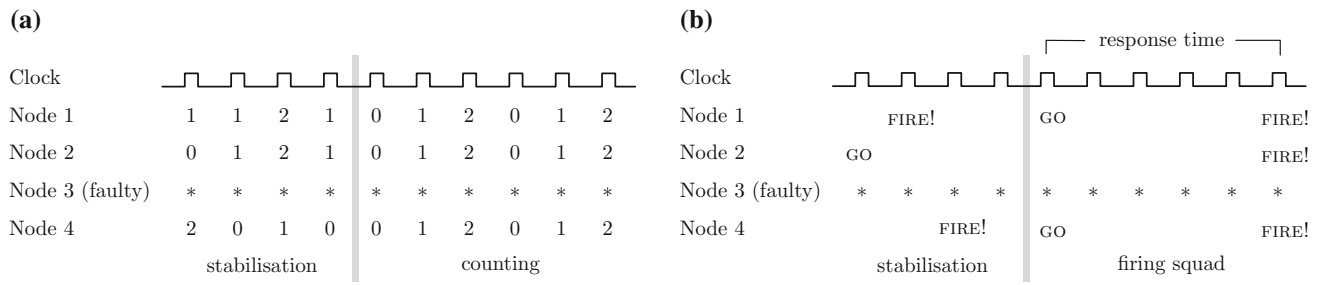

Fig. 1 **a** An execution of a synchronous counting protocol. **b** An execution of a firing squad protocol

self-stabilising and tolerates permanently faulty behaviour from some of the nodes. To address this issue, we study the synchronous counting and firing squad problems, which are among the most fundamental challenges in fault-tolerant distributed systems.

In the *synchronous counting* problem, all the nodes receive well-separated synchronous clock pulses that designate the start of a new round. The received clock pulses are anonymous, and hence, all correct nodes should eventually stabilise and agree on a round counter that increases consistently by one modulo C. The problem is also known as *digital clock synchronisation*, as all non-faulty nodes essentially have to agree on a shared logical clock. A stabilising execution of such a protocol for $n = 4$, $f = 1$, and $C = 3$ is given in Fig. 1a.

In the *self-stabilising firing squad problem*, the task is to have all correct nodes eventually stabilise and respond to an external input simultaneously. That is, once stabilised, when a sufficiently large (depending on the type of permanent faults) subset of the correct nodes receive an external "go" signal, then all correct nodes should eventually generate a local "fire" event on the same round. The time taken to react to the "go" signal is called the response time. Note that before stabilisation the nodes may generate spurious firing signals, but after stabilisation no correct node should generate a "fire" event without some correct node having previously received a "go" signal as input. An execution of such a protocol with $n = 4$, $f = 1$, and response time $R = 5$ is illustrated in Fig. 1b.

A firing squad protocol can be used, for example, to agree in a self-stabilising manner on when to initiate a new instance of a non-self-stabilising distributed protocol, as response to internal or external "go" inputs.

1.2 Connections to fault-tolerant consensus

Reaching agreement is perhaps the most intrinsic problem in fault-tolerant distributed computing. It is known that both the synchronous counting [11] and the self-stabilising firing squad problem [14] are closely connected to the well-studied consensus problem [24,30], where each node is given an input bit and the task is to agree on a common output bit such that if every non-faulty node received the same value as input, then this value must also be the output value. Indeed, the connection is obvious on an intuitive level, as in each task the goal is to agree on a common decision (that is, the output bit, clock value, or whether to generate a firing event).

However, the key difference between the problems lies in self-stabilisation. Typically, the consensus problem is considered in a non-self-stabilising setting with only permanent faults (e.g. $f < n/3$ nodes with arbitrary behaviour), whereas synchronous counting copes with both transient and permanent faults. In fact, it is easy to see that synchronous counting is trivial in a non-self-stabilising setting: if all nodes are initialised with the same clock value, then they can simply locally increment their counters each round without any communication. Furthermore, in a properly initialised system, one can reduce the firing squad problem to repeatedly calling a consensus routine [5].

Interestingly, imposing the requirement of self-stabilisation—convergence to correct system behavior from arbitrary initial states—reverses the roles. Solving either the synchronous counting or firing squad problem in a self-stabilising manner also yields a solution to binary consensus, but the converse is not true. In fact, in order to internally or externally trigger a consistent execution of a consensus protocol (or any other non-self-stabilising protocol, for that matter), nodes typically rely on some agreement on time (by using e.g. a self-stabilising synchronous counting to get a clock or firing squad algorithm to know when the execution should start).

In light of this, the self-stabilising variants of both problems are important generalisations of consensus. While considerable research has been dedicated to both tasks [2,11–14,18,27], our understanding is significantly less developed than for the extensively studied consensus problem. Moreover, it is worth noting that all existing algorithms utilise consensus subroutines [13,27] or shared coins [2], the latter of which essentially solves consensus as well. Given that both tasks are at least as hard as consensus [11], this seems to be a natural approach. However, it raises the question of how much of an overhead must be incurred by such a reduction. In

this paper, we subsume and improve upon previous results by providing a generic reduction of synchronous counting and self-stabilising firing squad to binary consensus that incurs very small overheads.

1.3 Contributions

We develop a framework for efficiently transforming *non-self-stabilising* consensus algorithms into *self-stabilising* algorithms for synchronous counting and firing squad problems. In particular, the resulting self-stabilising algorithms have the same resilience as the original consensus algorithms, that is, the resulting algorithms tolerate the same number and type of permanent faults as the original consensus algorithm (e.g. crash, omission, or Byzantine faults).

The construction we give incurs a small overhead compared to time and bit complexities of the consensus routines: the stabilisation time and message size are, up to constant factors, given as the sum of the cost of the consensus routine for f faults and recursively applying our scheme to $f' < f/2$ faults. Finally, our construction can be used in conjunction with both deterministic and randomised consensus algorithms. Consequently, we also obtain algorithms for probabilistic variants of the synchronous counting and firing squad problems.

Our novel framework enables us to address several open problems related to self-stabilising firing squads and synchronous counting. We now give a brief summary of the open problems we solve and the new results obtained using our framework.

1.3.1 Self-stabilising firing squads

In the case of self-stabilising firing squads, Dolev et al. [14] posed the following two open problems:

1. Are there solutions that tolerate either omission or Byzantine (i.e., arbitrary) faults?
2. Are there algorithms using $o(n)$-bit messages only?

We answer both questions in the affirmative by giving algorithms that achieve both properties *simultaneously*. Concretely, our framework implies a deterministic solution for the self-stabilising Byzantine firing squad problem that

- tolerates the optimal number of $f < n/3$ Byzantine faulty nodes,
- uses messages of $O(\log f)$ bits, and
- is guaranteed to stabilise and respond to inputs in linear-in-f communication rounds.

Thus, compared to prior state-of-the-art solutions [14], our algorithm tolerates a much stronger form of faulty behaviour and uses exponentially smaller messages, yet retains asymptotically optimal worst-case stabilisation and response time.

1.3.2 Synchronous counting

We attain novel algorithms for synchronous counting, that is, self-stabilising Byzantine fault-tolerant *digital clock synchronisation* [2,18,22]. Our new algorithms resolve questions left open by our own prior work [27], namely, whether there exist

1. deterministic linear-time algorithms with optimal resilience and message size $o(\log^2 f)$, or
2. randomised sublinear-time algorithms with small bit complexity.

Again, we answer both questions positively using our framework developed in this paper. For the first question, we give linear-time deterministic algorithms that have message size of $O(\log f)$ bits. For the second question, we show that our framework can utilise efficient randomised consensus algorithms to obtain probabilistic variants of the synchronous counting and firing squad problems. For example, using the result of King and Saia [23] we get algorithms that stabilise in polylog n expected rounds and use messages of size polylog n bits, assuming private communication links and an adaptive Byzantine adversary corrupting $f < n/(3+\varepsilon)$ nodes for an arbitrarily small constant $\varepsilon > 0$.

1.4 Related work

In this section, we overview prior work on the synchronous counting and firing squad problems. By now it has been established that both problems [11,14] are closely connected to the well-studied (non-stabilising) consensus [24,30]. As there exists a vast body of literature on synchronous consensus, we refer the interested reader to, e.g., the survey by Raynal [31]. We note that self-stabilising variants of consensus have been studied [1,8,9,17] but in different models of computation and/or for different types of failures than what we consider in this work.

1.4.1 Synchronous counting and digital clock synchronisation

In the past two decades, there has been increased interest in combining self-stabilisation with Byzantine fault-tolerance. One reason is that algorithms in this fault model are very attractive in terms of designing highly-resilient hardware [11]. A substantial amount of work on synchronous counting has been carried out [2,12,13,18,22,27], comprising both positive and negative results.

In terms of lower bounds, many impossibility results for consensus [10,15,21,30] also directly apply to synchronous counting, as synchronous counting solves binary consensus [11,12]. In particular, no algorithm can tolerate more than $f < n/3$ Byzantine faulty nodes [30] (unless cryptographic assumptions are made) and any deterministic algorithm needs at least $f + 1$ rounds to stabilise [21].

In a seminal work, Dolev and Welch [18] showed that synchronous counting can be solved in a self-stabilising manner in the presence of (the optimal number of) $f < n/3$ Byzantine faults using randomisation; see also [16, Ch. 6]. While this algorithm can be implemented using only constant-size messages, the expected stabilisation time is exponential. Later, Ben-Or et al. [2] showed that it is possible to obtain optimally-resilient solutions that stabilise in expected constant time. However, their algorithm relies on shared coins, which are costly to implement and assume private communication channels.

In addition to the lower bound results, there also exist deterministic algorithms for the synchronous counting problem [12,13,22,27]. Many of these algorithms utilise consensus routines [13,22,27], but obtaining fast and communication-efficient solutions with optimal resilience has been a challenge. For example, Dolev and Hoch [13] apply a pipelining technique, where $\Omega(f)$ consensus instances are run in parallel. While this approach attains optimal resilience and linear stabilisation time in f, the large number of parallel consensus instances necessitates large messages.

In order to achieve better communication and state complexity, the use of computational algorithm design and synthesis techniques have also been investigated [4,12]. While this line of research has produced novel optimal and computer-verified algorithms, so far the techniques have not scaled beyond $f = 1$ faulty node due to the inherent combinatorial explosion in the search space of potential algorithms.

Recently, we gave recursive constructions for synchronous counting that achieve linear stabilisation time using only polylogarithmic message size and state bits per node [27]. However, our previous constructions relied on specific (deterministic) consensus routines and their properties in a relatively ad hoc manner. In contrast, our new framework presented here lends itself to any (possibly randomised) synchronous consensus routine and improves the best known upper bound on the message size to $O(\log f)$ bits. Currently, it is unknown whether it is possible to deterministically achieve message size of $o(\log f)$ bits.

1.4.2 Firing squads

In the original formulation of the firing squad synchronisation problem, the system consists of a path of n finite state machines (whose number of states is independent of n) and the goal is to have all machines switch to the same "fire" state simultaneously after one node receives a "go" signal. This formulation of the problem has been attributed to John Myhill and Edward Moore and has subsequently been studied in various settings; see e.g. [29] for a survey of early work related to the problem.

In the distributed computing community, the firing squad problem has been studied in fully-conneted networks in the presence of faulty nodes. Similarly to synchronous counting, the firing squad problem is closely connected to Byzantine agreement and simultaneous consensus [5–7,14,19]. Both Burns and Lynch [5] and Coan et al. [6] studied the firing squad problem in the context of Byzantine failures. Burns and Lynch [5] considered both permissive and strict variants of the problem (i.e., whether faulty nodes can trigger a firing event or not) and showed that both can be solved using Byzantine consensus algorithms with only a relatively small additional overhead in the number of communication rounds and total number of bits communicated. On the other hand, Coan et al. [6] gave *authenticated* firing squad algorithms for various Byzantine fault models. Coan and Dwork [7] gave worst-case time lower bounds of $f + 1$ rounds for deterministic and randomised algorithms solving the firing squad problem in the crash fault model.

However, neither the solutions of Burns and Lynch [5] or Coan et al. [6] are self-stabilising or use small messages. Almost two decades later, Dolev et al. [14] gave the first self-stabilising algorithm for the firing squad problem. In particular, their solution has optimal stabilisation time and response time depending on the fault pattern. However, their algorithm tolerates only crash faults and uses messages of size $\Theta(n \log n)$ bits. In this work, we obtain Byzantine fault-tolerant algorithms that use messages of size $O(\log n)$ bits. However, the time complexity is only (asymptotically) optimal in the worst case.

1.5 Outline of the paper

The paper is structured as follows. In the first part of the paper, we consider the deterministic setting under Byzantine faults. In the second part, we discuss how to extend our results into the randomised setting, where sublinear time algorithms are possible.

We start with Sect. 2, where we give formal definitions related to the model of computation, synchronous counting, and firing squads in the Byzantine setting. In the sections following this, we show our main result in a top-down fashion as illustrated in Fig. 2. We introduce a series of new problems and give reductions between them:

– Section 3 shows how to obtain synchronous counting and firing squad algorithms that rely on binary consensus routines and *strong pulsers*,

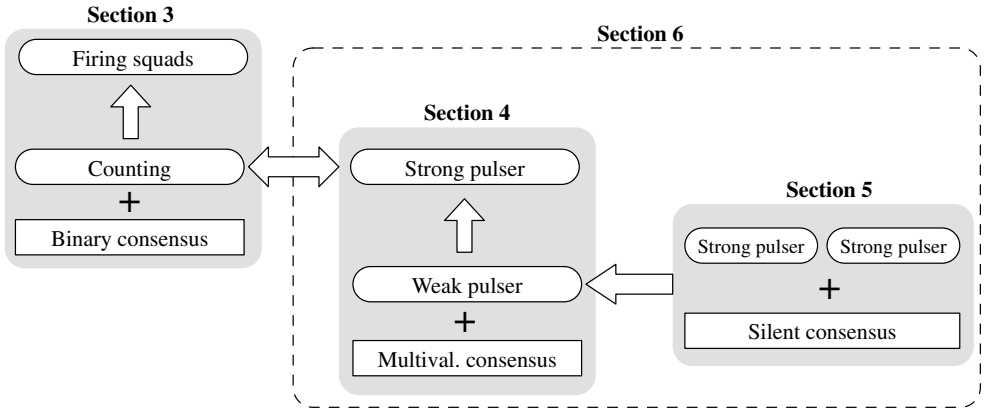

Fig. 2 High-level overview of our construction and the structure of the paper. Rounded boxes denote algorithms that are both self-stabilising and Byzantine fault-tolerant, whereas rectangular boxes denote non-stabilising Byzantine fault-tolerant consensus routines

- Section 4 devises strong pulsers with the help of *weak pulsers* and multivalued consensus,
- Section 5 constructs weak pulsers using *silent consensus* and less resilient strong pulsers.

Section 6 combines the results of Sects. 4 and 5 to obtain a recursive construction for strong pulsers used by the algorithms given in Sect. 3. Finally, we extend our results to work with randomised consensus routines in Sect. 7.

2 Preliminaries

In this section, we first fix some basic notation, then describe the model of computation, and finally give formal definitions of the synchronous counting, self-stabilising firing squad, and consensus problems.

2.1 Notation

We use $\mathbb{N} = \{1, 2, \ldots\}$ to denote the set of positive integers and $\mathbb{N}_0 = \{0\} \cup \mathbb{N}$ to denote the set of all non-negative integers. For any $k \in \mathbb{N}$, we write $[k] = \{0, \ldots, k-1\}$ to be the set of the first k non-negative integers.

2.2 Model of computation

We consider a fully-connected synchronous network on node set V consisting of $n = |V|$ processors. We assume there exists a subset of $F \subseteq V$ faulty nodes that is initially unknown to the correct nodes, where the upper bound f on the size $|F| \leq f$ is known to the nodes. We say that nodes in $V \setminus F$ are correct and nodes in F are faulty.

All correct nodes in the system will follow a given algorithm $\mathbf{A}$ that is the same for all the nodes in the system. The execution proceeds in synchronous rounds, where in each round $t \in \mathbb{N}$ the nodes take the following actions in lockstep:

1. perform local computations,
2. send messages to other nodes, and
3. receive messages from other nodes.

We assume that nodes have unique identifiers from $\{1, \ldots, n\}$ and can identify the sender of incoming messages.

We say that an algorithm $\mathbf{A}$ has message size $M(\mathbf{A})$ if no correct node sends more than $M(\mathbf{A})$ bits to any other node during a single round.

The local computations of a node v determine which messages v sends to other nodes and what is the new state of the node v, where the new state is a function of the node's previous local state and received messages. As we are interested in self-stabilising algorithms, the initial state of a node is arbitrary; this is equivalent to assuming that transient faults have arbitrarily corrupted the state of each node, but the transient faults have ceased by the beginning of the first round.

As mentioned above, we assume the presence of (possibly permanent) *Byzantine* faults. A Byzantine faulty node $v \in F$ may deviate from the algorithm arbitrarily, i.e., send arbitrary messages in each round. In particular, a Byzantine faulty node can send different messages to each correct node in the system, even if the algorithm specifies otherwise. Since we consider deterministic algorithms, the meaning of "arbitrary" in this context is that the algorithm must succeed for any possible choice of behavior of the faulty nodes. We assume a bound $f < n/3$ on the number $|F|$ of Byzantine faulty nodes, as otherwise none of the problems we consider can be solved due to the impossibility of consensus under $f \geq n/3$ Byzantine faults [30].

2.3 Synchronous counting

In the *synchronous C-counting* problem, the task is to have each node $v \in V$ output a counter value $c(v,t) \in [C]$ on each round $t \in \mathbb{N}$ in a consistent manner. We say that an execution of a synchronous counting algorithm *stabilises in round* t if and only if all $t \leq t' \in \mathbb{N}$ and $v, w \in V \setminus F$ satisfy

SC1 **Agreement:** $c(v,t') = c(w,t')$ and
SC2 **Consistency:** $c(v,t'+1) = c(v,t') + 1 \bmod C$.

We say that $\mathbf{A}$ is an f-resilient C-counting algorithm that stabilises in time t if all executions with at most f faulty nodes stabilise by round t. The stabilisation time $T(\mathbf{A})$ of $\mathbf{A}$ is the maximum such t over all executions.

2.4 Self-stabilising firing squad

In the self-stabilising Byzantine firing squad problem, in each round $t \in \mathbb{N}$, each node $v \in V$ has an input channel $\mathrm{GO}(v,t) \in \{0,1\}$. If $\mathrm{GO}(v,t) = 1$, then we say that v receives a GO input signal in round t. Moreover, the algorithm determines an output $\mathrm{FIRE}(v,t) \in \{0,1\}$ at each node $v \in V$ in each round $t \in \mathbb{N}$. We say that an execution of an algorithm *stabilises in round* $t \in \mathbb{N}$ if the following three properties hold:

FS1 **Agreement:** $\mathrm{FIRE}(v,t') = \mathrm{FIRE}(w,t')$ for all $v, w \in V \setminus F$ and $t \leq t' \in \mathbb{N}$.
FS2 **Safety:** If $\mathrm{FIRE}(v,t_F) = 1$ for $v \in V \setminus F$ and $t \leq t_F \in \mathbb{N}$, then there is $t_F \geq t_G \in \mathbb{N}$ s.t.
  (i) $\mathrm{GO}(w,t_G) = 1$ for some $w \in V \setminus F$ and
  (ii) $\mathrm{FIRE}(v,t') = 0$ for all $t' \in \{t_G+1, \dots, t_F-1\}$.
FS3 **Liveness:** If $\mathrm{GO}(v,t_G) = 1$ for at least $f+1$ correct nodes $v \in V \setminus F$ and $t \leq t_G \in \mathbb{N}$, then $\mathrm{FIRE}(v,t_F) = 1$ for all nodes $v \in V \setminus F$ and some $t_G < t_F \in \mathbb{N}$.

Note that the liveness condition requires $f+1$ correct nodes to receive an external GO input, as otherwise it would be impossible to guarantee that a correct node has received a GO input when firing; this corresponds to the definition of a strict Byzantine firing squad [5]. We say that an execution stabilised by round t has *response time* of at most R from round t onwards if the following conditions are satisfied:

(i) if at least $f+1$ correct nodes $v \in V \setminus F$ satisfy $\mathrm{GO}(v,t_G) = 1$ in some round $t_G \geq t$, then all correct nodes $u \in V \setminus F$ satisfy $\mathrm{FIRE}(v,t_F) = 1$ in some round $t_G < t_F \leq t_G + R$, and
(ii) if there is a round $t_F \geq t$ such that $\mathrm{FIRE}(v,t_F) = 1$ for some correct $v \in V \setminus F$, then there is a round t_G with $t_F > t_G \geq t_F - R$ and some correct node $u \in V \setminus F$ with $\mathrm{GO}(u,t_G) = 1$.

Finally, we say that an algorithm $\mathbf{F}$ is an f-resilient firing squad algorithm with stabilisation time $T(\mathbf{F})$ and response time $R(\mathbf{F})$ if in any execution of the system with at most f faulty nodes there is a round $t \leq T(\mathbf{F})$ by which the algorithm stabilised and has response time at most $R(\mathbf{F})$ from round t onwards.

We remark that under Byzantine faults, previous non-stabilising algorithms [5] have considered the case where the input signals (from different nodes) do not need to be received on the same round, but they can be spread out over several rounds. In the self-stabilising setting, we can easily cover the case where $f+1$ input signals are received within a time window of Δ rounds as follows: instead of relying on the input GO signals as-is, we can use an auxiliary variable $\mathrm{GO}'(v,t)$ as input to our algorithms, where $\mathrm{GO}'(v,t) = 1$ iff there is a round $t' \in \{t-\Delta+1, \dots, t\}$ with $\mathrm{GO}(v,t') = 1$.

2.5 Consensus

Let us conclude this section by defininining the *multivalued consensus problem*. Unlike the synchronous counting and self-stabilising firing squad problems, the standard definition of consensus does *not* require self-stabilisation: we assume that all nodes start from a fixed starting state and the algorithm terminates in finitely many communication rounds.

In the multivalued consensus problem for $L > 1$ values, each node $v \in V$ receives an input value $x(v) \in [L]$ and the task is to have all correct nodes output the same value $k \in [L]$. We say that an algorithm $\mathbf{C}$ is an f-resilient $T(\mathbf{C})$-round consensus algorithm if the following conditions hold when there are at most f faulty nodes:

C1 **Termination:** Each $v \in V \setminus F$ decides on an output $y(v) \in [L]$ by the end of round $T(\mathbf{C})$.
C2 **Agreement:** For all $v, w \in V \setminus F$, it holds that $y(v) = y(w)$.
C3 **Validity:** If there exists $k \in [L]$ such that for all $v \in V \setminus F$ it holds that $x(v) = k$, then each $v \in V \setminus F$ outputs the value $y(v) = k$.

We remark that one may ask for stronger validity conditions, but for our purposes this condition is sufficient. The *binary consensus* problem is the special case $L = 2$ of the above multivalued consensus problem. In the case of binary consensus, the stated validity condition is equivalent to requiring that if $v \in V \setminus F$ outputs $y(v) = k \in \{0,1\}$, then some $w \in V \setminus F$ has input value $x(w) = k$.

Later, we utilise the fact that multivalued consensus can be reduced to binary consensus with only a small overhead in time. In [25], it is shown how to do this with 1-bit messages

and an additive overhead of $O(\log L)$ rounds, preserving resilience.

Theorem 1 ([25]) *Let $L > 1$. Given an f-resilient binary consensus algorithm $\mathbf{C}$, we can solve L-value consensus in $O(\log L) + T(\mathbf{C})$ rounds using $M(\mathbf{C})$-bit messages while tolerating f faults.*

3 Synchronous counting and firing squads

In this section, we give a firing squad algorithm with asymptotically optimal stabilisation and response times. The algorithm relies on two auxiliary routines: a so-called strong pulser and a consensus algorithm. We start with a discussion on strong pulsers.

3.1 Strong pulsers and counting

Our approach to the firing squad problem is to solve it by repeated consensus, which in turn is controlled by a joint round counter. To minimise message size, however, we will not communicate counter values directly. Instead we make use of what we call a *strong pulser*.

Definition 1 (*Strong pulser*) An algorithm $\mathbf{P}$ is an f-resilient strong Ψ-pulser that stabilises in $T(\mathbf{P})$ rounds if it satisfies the following conditions in the presence of at most f faulty nodes. Each node $v \in V$ produces an output bit $p(v,t) \in \{0,1\}$ on each round $t \in \mathbb{N}$. We say that v generates a pulse in round t if $p(v,t) = 1$ holds. We require that there is a round $t_0 \leq T(\mathbf{P})$ such that:

S1. For any $v \in V \setminus F$ and round $t = t_0 + k\Psi$, where $k \in \mathbb{N}_0$, it holds that $p(v,t) = 1$.
S2. For any $v \in V \setminus F$ and round $t \geq t_0$ satisfying $t \neq t_0 + k\Psi$ for $k \in \mathbb{N}_0$, we have $p(v,t) = 0$.

Put otherwise, a strong Ψ-pulser consistently generates pulses at all non-faulty nodes exactly every Ψ rounds. Figure 3 illustrates an execution of a strong pulser with $\Psi = 3$. It is straightforward to see that strong pulsers and synchronous counting are almost equivalent.

Lemma 1 *Let $C \in \mathbb{N}$ and $\Psi \in \mathbb{N}$. If C divides Ψ, then a strong Ψ-pulser that stabilises in T rounds can be used to implement a synchronous C-counter that stabilises in at most T rounds. If Ψ divides C, then a synchronous C-counter that stabilises in T rounds can be used to implement a strong Ψ-pulser that stabilises in at most $T + \Psi - 1$ rounds.*

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

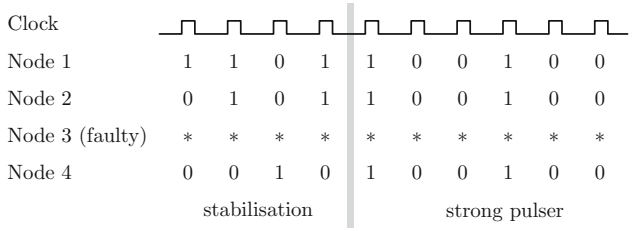

Fig. 3 An example execution of a strong 3-pulser on $n = 4$ nodes with $f = 1$ faulty node

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.

3.2 Firing squads via pulsers and consensus

We now show how an f-resilient strong pulser and f-resilient binary consensus algorithm can be used to devise an f-resilient firing squad algorithm. As a strong pulser can be used to control repeated execution of a non-self-stabilising algorithm, it enables us to run consensus on whether a firing event should be triggered or not repeatedly. As the firing squad problem is at least as hard as consensus, this maintains asymptotically optimal round complexity.

Recall that for the *Byzantine* firing squad problem, we are interested in a liveness condition in which a firing event needs to be generated if at least $f + 1$ non-faulty nodes $v \in V \setminus F$ recently saw $\mathrm{GO}(v,t) = 1$ in some round t. To this end, we have each node continuously inform all other nodes about its GO values (i.e. their received input signals). Whenever node $v \in V$ sees $f + 1$ nodes $w \in V$ claim $\mathrm{GO}(w,t) = 1$, it will memorise this and use input $x(v) = 1$ for the next consensus instance. Otherwise, it will use the input value $x(v) = 0$; this ensures that at least one non-faulty node w had $\mathrm{GO}(w,t) = 1$ recently in case v uses input $x(v) = 1$. The validity condition of the (arbitrary) $T(\mathbf{C})$-round consensus routine $\mathbf{C}$ thus ensures both liveness and safety for the resulting firing squad algorithm. Apart from $\mathbf{C}$, the algorithm concurrently runs a strong Ψ-pulser $\mathbf{P}$ for some $\Psi > T(\mathbf{C})$.

3.2.1 The firing squad algorithm

Given a strong Ψ-pulser algorithm $\mathbf{P}$ and a binary consensus algorithm $\mathbf{C}$, each node v stores the following variables on every round t:

- $p(v,t) \in \{0,1\}$, the output variable of $\mathbf{P}$,
- $x(v,t) \in \{0,1\}$ and $y(v,t) \in \{0,1\}$, the input and output variables of $\mathbf{C}$, and
- $m(v,t) \in \{0,1\}$, an auxiliary variable used to memorise whether sufficiently many GO signals were received to warrant a firing event.

In the following algorithm, on each round $t \in \mathbb{N}$ any (correct) node $v \in V$ will broadcast the value $\mathrm{GO}(v,t)$ and receive the values $\mathrm{GO}(v,w,t-1)$ sent by every $w \in V$ in the previous round. The algorithm consists of each node v executing the following operations[1] in each round $t \in \mathbb{N}$:

1. Broadcast $\mathrm{GO}(v,t)$.
2. If v received $\mathrm{GO}(v,w,t-1) = 1$ from at least $f+1$ nodes $w \in V$, then set $x(v,t) = 1$ and $m(v,t) = 1$. Otherwise, set $x(v,t) = x(v,t-1)$ and $m(v,t) = m(v,t-1)$.
3. If $p(v,t) = 1$, start executing a new instance of $\mathbf{C}$ using the value $x(v,t)$ as input and set $m(v,t) = 0$ while aborting any previously running instance. More specifically, this entails the following:
   - Maintain a local round counter r, which is initialised to 1 on round t and increased by 1 after each round.
   - Maintain the local state variables related to the consensus routine $\mathbf{C}$.
   - On each round, execute round r of algorithm $\mathbf{C}$; if the state variables indicate that $\mathbf{C}$ terminated at v, then do nothing.
   - On the round when r would attain the value $T(\mathbf{C})+1$, stop the simulation (indicating this, e.g., by setting $r(v) = \perp$) and locally output the value of $y(v)$ computed by the simulation of $\mathbf{C}$.
4. If $\mathbf{C}$ outputs $y(v,t) = 1$ on round t, then output $\mathrm{FIRE}(v,t) = 1$ and set $x(v,t) = 0$.
   Otherwise, set $\mathrm{FIRE}(v,t) = 0$.
5. If $\mathbf{C}$ outputs $y(v,t) = 0$ on round t and $m(v,t) = 0$, then set $x(v,t) = 0$.

We now show that the above algorithm satisfies the properties required from a self-stabilising firing squad.

Theorem 2 *Suppose that there exists an f-resilient strong Ψ-pulser $\mathbf{P}$ and a consensus algorithm $\mathbf{C}$, where $\Psi > T(\mathbf{C})$. Then there exists an f-resilient firing squad algorithm $\mathbf{F}$ that*

- *stabilises in time $T(\mathbf{F}) \le T(\mathbf{P}) + \Psi$,*
- *has response time $R(\mathbf{F}) \le \Psi + T(\mathbf{C})$, and*
- *uses messages of size $M(\mathbf{F}) \le M(\mathbf{P}) + M(\mathbf{C}) + 1$ bits.*

Proof Let $\mathbf{F}$ be the algorithm described above. We now argue that the algorithm satisfies the three properties given in Sect. 2.4: (FS1) agreement, (FS2) safety, and (FS3) liveness. We will show that the algorithm has a response time bounded by $R = T(\mathbf{C}) + \Psi$.

(FS1) Denote by $t_0 \le T(\mathbf{P})$ the round in which the execution of the strong Ψ-pulser $\mathbf{P}$ has stabilised and generated a pulse. That is, for rounds $t \ge t_0$ we have that $p(v,t) = 1$ is equivalent to $t = t_0 + k\Psi$ for some $k \in \mathbb{N}_0$. This implies that the algorithm will correctly simulate instances of the consensus routine $\mathbf{C}$ and locally output its decision on rounds $r_k = t_0 + T(\mathbf{C}) + k\Psi < t_{k+1}$ for $k \in \mathbb{N}_0$. The agreement property of the firing squad thus follows from the agreement property of consensus for all rounds $t \ge t_0$, as $\mathrm{FIRE}(v,t) = 1$ if and only if $t = r_k$ and the simulation of $\mathbf{C}$ output the value $y(v,t) = 1$ in Step 4.

(FS2) Concerning safety, suppose $v \in V \setminus F$ outputs $\mathrm{FIRE}(v,t_F) = 1$ in round $t_F \ge t_1 + T(\mathbf{C})$. By the above discussion and the validity property of consensus, this implies that there was some node $w \in V \setminus F$ that started a (successfully and completely simulated) instance of $\mathbf{C}$ with input $x(w,t_k) = 1$ in round $t_k = t_F - T(\mathbf{C}) = t_0 + k\Psi$ and that $t_F = r_k$ for some $k \in \mathbb{N}$. Assume for contradiction that there are no $u \in V \setminus F$ and $t_G \in \{t_{k-1}, \dots, t_k - 1\}$ satisfying $\mathrm{GO}(u,t_G) = 1$. Then, w does not set $x(w,t')$ or $m(w,t')$ to 1 in rounds $t' \in \{t_{k-1}+1, \dots, t_k\}$ in Step 2. However, in round $t_{k-1} = t_F - T(\mathbf{C}) - \Psi = t_0 + (k-1)\Psi$ node w set $m(w,t_{k-1}) = 0$ (by Step 3) and thus w sets $x(w,r_{k-1}) = 0$ later in round r_{k-1} (by Steps 4 and 5), the round in which the previous instance of $\mathbf{C}$ locally output some value. This contradicts the fact that $x(w,t_k) = 1$ is set in round t_k. Hence, there must be $u \in V \setminus F$ and $t_G \in \{t_{k-1}, \dots, t_k - 1\}$ such that $\mathrm{GO}(u,t_G) = 1$.

Recall that the above claimed existence of $u \in V \setminus F$ and t_G such that $\mathrm{GO}(u,t_G) = 1$ is necessary for the safety condition to hold, but not sufficient. It is also required that $\mathrm{FIRE}(v,t') = 0$ for all $t' \in \{t_G + 1, \dots, t_F - 1\}$. To show this, observe that the time t_G shown to exist by the above reasoning does not satisfy this additional constraint if and only if some instance of $\mathbf{C}$ locally outputs $y(v,t') = 1$ at node v in such a round t'. The only possible such round t' is r_{k-1}, as $t' \ge t_G + 1 > t_{k-1} > r_{k-2}$. However, in this case, each $w \in V \setminus F$ sets $x(w,r_{k-1}) = 0$ in round r_{k-1} regardless of $m(w,r_{k-1})$ in Step 4, and we can conclude that some $w \in V \setminus F$ must set $x(w,t'') = 1$ in some round $t'' \in \{r_{k-1}+1, \dots, t_k\}$. As above, it follows that there is a round $t_G \in \{t_{k-1}, \dots, t_k - 1\}$ and a node $u \in V \setminus F$ such that $\mathrm{GO}(u,t_G) = 1$. Overall, we see that the safety condition for a firing squad algorithm with response time

[1] For better readability, we allow for statements about what a node communicates appearing anywhere in the description. Note, however, that sending operations happen after local computation, i.e., only information sent in the previous rounds is available for computations.

$$t_F - t_G \leq r_k - t_{k-1} = t_k + T(\mathbf{C}) - t_{k-1} = \Psi + T(\mathbf{C}) = R$$

is satisfied in rounds $t_F \geq r_1 = t_1 + T(\mathbf{C})$.

(FS3) It remains to argue that the algorithm satisfies the liveness property with response time bounded by R. Suppose that at least $f+1$ nodes $v \in V \setminus F$ satisfy $\mathrm{GO}(v, t_G) = 1$ in some round $t_G \geq t_0 - 1$. Then, in round $t_G + 1 \geq t_0$ all nodes $v \in V \setminus F$ set $x(v, t_G + 1) = 1$ and $m(v, t_G + 1) = 1$ according to Step 2. Assume for contradiction that $\mathrm{FIRE}(v, t) = 0$ for all $t \in \{t_G + 1, \ldots, t_G + R\}$. Denote by $t_G + 1 \leq t_k \leq t_G + \Psi$ the unique round such that $t_k = t_0 + k\Psi$ for some $k \in \mathbb{N}_0$. The instance of $\mathbf{C}$ started in this round will satisfy that all correct nodes $v \in V \setminus F$ have input $x(v, t_k) = 1$: by our assumption towards contradiction, no node can locally output $y(v, t') = 1$ during rounds $t' \in \{t_G + 1, \ldots, t_G + R\}$; thus, no node can set $x(v, \cdot)$ to 0 without setting $m(v, \cdot)$ to 0 first (by Step 3 and Step 5), which in turn entails that at time t_k an instance of $\mathbf{C}$ with value of $x(v, t_k) = 1$ is started before this happens. By the properties of $\mathbf{C}$, it follows that each $v \in V \setminus F$ locally outputs 1 in round $r_k = t_k + T(\mathbf{C}) \leq t_G + \Psi + T(\mathbf{C}) \leq t_G + R$, contradicting our previous assumption. We conclude that our algorithm satisfies the liveness property with response time $R = \Psi + T(\mathbf{C})$ for rounds $t_G \geq t_0 - 1$.

As $t_0 \leq T(\mathbf{P})$, it follows that the algorithm satisfies (FS1) agreement after round t_0, (FS2) safety after round t_1, and (FS3) liveness after round $t_0 - 1$. Since $t_1 = t_0 + \Psi \leq T(\mathbf{P}) + \Psi$, it follows that the algorithm is a firing squad with response time at most $R = \Psi + T(\mathbf{C})$ that stabilises in $\max\{T(\mathbf{P}), T(\mathbf{P}) + \Psi, T(\mathbf{P}) - 1\} = T(\mathbf{P}) + \Psi$ rounds. The bound on the message size follows from the fact that the algorithm $\mathbf{F}$ only broadcasts 1 bit in Step 1 in addition to the messages related to $\mathbf{P}$ and $\mathbf{C}$. □

4 From weak pulsers to strong pulsers

In Sect. 3, we established that it suffices to construct suitable strong pulsers to solve the synchronous counting and firing squad problems. We will now reduce the construction of strong pulsers to constructing *weak pulsers*.

4.1 Weak pulsers

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. Formally, we define weak pulsers as follows.

Definition 2 (*Weak pulsers*) An algorithm $\mathbf{W}$ is an f-resilient weak Φ-pulser that stabilises in $T(\mathbf{W})$ rounds if the following holds. In each round $t \in \mathbb{N}$, each node $v \in V$ produces an output $a(v, t)$. Moreover, there exists a round $t_0 \leq T(\mathbf{W})$ such that

W1 for all $v, w \in V \setminus F$ and all rounds $t \geq t_0$, $a(v, t) = a(w, t)$,
W2 $a(v, t_0) = 1$ for all $v \in V \setminus F$, and
W3 $a(v, t) = 0$ for all $v \in V \setminus F$ and $t \in \{t_0 + 1, \ldots, t_0 + \Phi - 1\}$.

We say that on round t_0 a *good* pulse is generated by $\mathbf{W}$.

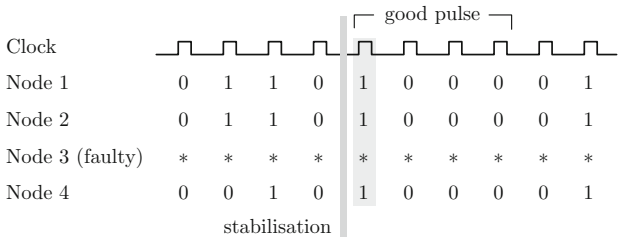

Fig. 4 An example execution of a weak 4-pulser on $n = 4$ nodes with $f = 1$ faulty node. Eventually, a good pulse is generated (highlighted column). A good pulse is followed by *at least* three rounds in which no correct node generates a pulse. In contrast, the pulse two rounds earlier is not good, as it is followed by only one round of silence. Unlike strong pulsers, weak pulsers allow for behaviour where the correct nodes do not generate any pulses for some time after a good pulse (last two columns)

Figure 4 illustrates a weak 3-pulser. Note that 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 once every $T(\mathbf{W})$ rounds.

4.2 Constructing strong pulsers from weak pulsers

Recall that a strong pulser can be obtained by having nodes locally count down the rounds until the next pulse, provided we have a way of ensuring that the local counters eventually agree. This can be achieved by using a weak pulser to control a suitable consensus routine, where again we always have only a single instance running at any time. While some instances will be aborted before they can complete, this will not affect the counters, as we only adjust them when the consensus routine completes. On the other hand, the weak pulser guarantees that within $T(\mathbf{W})$ rounds, there will be a pulse followed by $\Phi - 1$ rounds of silence, enabling to complete a run of any consensus routine $\mathbf{C}$ satisfying $T(\mathbf{C}) \leq \Phi$. Thus, for constructing a strong Ψ-pulser, we assume that we have the following f-resilient algorithms available:

– a $T(\mathbf{C})$-round Ψ-value consensus algorithm $\mathbf{C}$ and
– a weak Φ-pulser $\mathbf{W}$ for $\Phi \geq T(\mathbf{C})$.

Given the above two algorithms, we show how to construct an f-resilient strong Ψ-pulser for any $\Psi > 1$. The pulser will stabilise in time $T(\mathbf{W}) + T(\mathbf{C}) + \Psi$ and the message size of the strong pulser will be bounded by $M(\mathbf{W}) + M(\mathbf{C})$.

As mentioned earlier, the idea is to have nodes simply count locally between pulses and use the weak pulser to execute a single instance of the consensus algorithm $\mathbf{C}$. Eventually, a good pulse will trigger an instance consistently and establish agreement among the local counters. Leveraging validity, we can ensure that the counters will never be affected by the consensus routine running in the background again.

4.2.1 Variables

Besides the variables of the weak pulser $\mathbf{W}$ and (a single copy of) $\mathbf{C}$, our construction of a strong Ψ-pulser uses the following local variables:

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

4.2.2 Strong pulser algorithm

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

1. If $c(v,t) = 0$, then set $b(v,t) = 1$ and otherwise $b(v,t) = 0$.
2. Set $c'(v,t) = c(v,t)$.
3. If $d(v,t) \neq \bot$, then
   (a) Execute the instructions of $\mathbf{C}$ for round $d(v,t)$.
   (b) If $d(v,t) \neq T(\mathbf{C})$, set $d(v,t+1) = d(v,t) + 1$.
   (c) If $d(v,t) = T(\mathbf{C})$, then
      i Set $c'(v,t) = y(v,t) + T(\mathbf{C}) \bmod \Psi$, where $y(v,t)$ is the output value of $\mathbf{C}$.
      ii Set $d(v,t+1) = \bot$.
4. Update $c(v,t+1) = c'(v,t) + 1 \bmod \Psi$.
5. If $a(v,t) = 1$, then
   (a) Start a new instance of $\mathbf{C}$ using $c'(v,t)$ as input (resetting all state variables of $\mathbf{C}$).
   (b) Set $d(v,t+1) = 1$.

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 $c'(v,t)$ to hold the counter value, which is overwritten by the output of $\mathbf{C}$ (increased by $T(\mathbf{C}) \bmod \Psi$) 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 $\mathbf{C}$ 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 $\mathbf{W}$ and the consensus algorithm $\mathbf{C}$. Thus, there is no additional overhead in communication and the message size is bounded by $M(\mathbf{W}) + M(\mathbf{C})$. Hence, it remains to show that the local counters $c(v,t)$ implement a strong Ψ-counter.

Theorem 3 *The variables $c(v,t)$ in the above algorithm implement a synchronous Ψ-counter that stabilises in $T(\mathbf{W}) + T(\mathbf{C}) + 1$ rounds and uses messages of at most $M(\mathbf{W}) + M(\mathbf{C})$ bits.*

Proof Suppose that round $t_0 \leq T(\mathbf{W})$ is as in Definition 2, that is, $a(v,t) = a(w,t)$ for all $t \geq t_0$, and a good pulse is generated in round t_0. Thus, all correct nodes participate in simulating an instance of $\mathbf{C}$ during rounds $t_0 + 1, \dots, t_0 + T(\mathbf{C})$, since no pulse is generated during rounds $t_0 + 1, \dots, t_0 + T(\mathbf{C}) - 1$, 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 $c'(v, t_0 + T(\mathbf{C})) = c'(w, t_0 + T(\mathbf{C}))$ for all $v, w \in V \setminus F$ after Step 3ci. By Steps 2 and 4, the same will hold for both $c(\cdot, t')$ and $c'(\cdot, t')$, $t' > t_0 + T(\mathbf{C})$, provided that we can show that in rounds $t' > t$, Step 3ci never sets $c'(v,t)$ to a value different than $c(v,t)$ for any $v \in V \setminus F$; as this also implies that $c(v, t'+1) = c(v,t') + 1 \bmod \Psi$ for all $v \in V \setminus F$ and $t' > t_0 + T(\mathbf{C})$, this will complete the proof.

Accordingly, consider any execution of Step 3ci in a round $t' > t_0 + T(\mathbf{C})$. The instance of $\mathbf{C}$ terminating in this round was started in round $t' - T(\mathbf{C}) > t_0$. However, in this round the weak pulser must have generated a pulse, yielding that, in fact, $t' - T(\mathbf{C}) \geq t_0 + T(\mathbf{C})$. Assuming for contradiction that t' is the earliest round in which the claim is violated, we thus have that $c'(v, t' - T(\mathbf{C})) = c'(w, t' - T(\mathbf{C}))$ for all $v, w \in V \setminus F$, i.e., all correct nodes used the same input value c for the instance. By the validity property of $\mathbf{C}$, this implies that $v \in V \setminus F$ outputs $y(v,t') = c$ in round t' and sets $c'(v,t') = c + T(\mathbf{C}) \bmod \Psi$. However, since t' is the earliest round of violation, we already have that $c'(v,t') = c(v,t') = c + T(\mathbf{C}) \bmod \Psi$ after the second step, contradicting the assumption and showing that the execution stabilised in round $t_0 + T(\mathbf{C}) + 1 \leq T(\mathbf{W}) + T(\mathbf{C}) + 1$. □

Together with Lemma 1, we get the following corollary.

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

- *stabilises in time $T(\mathbf{P}) \leq T(\mathbf{C}) + T(\mathbf{W}) + \Psi$, and*
- *uses messages of size at most $M(\mathbf{P}) \leq M(\mathbf{C}) + M(\mathbf{W})$ bits.*

5 Constructing weak pulsers from less resilient strong pulsers

Having seen that we can construct strong pulsers from weak pulsers using a consensus algorithm, the only piece missing in our framework is the existence of efficient weak pulsers. Indeed, having a pair of an f-resilient weak pulser and a consensus routine, we immediately obtain a corresponding firing squad algorithm.

In this section, we 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.

Our approach bears similarity to our constructions from earlier work [27], but attains better bit complexity and can be used with an arbitrary consensus routine. At a high level, we take the following approach as also illustrated in Fig. 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 stabilises.
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 stabilise, 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 all 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.
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.

5.1 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 $\mathbf{P}_i$ be an f_i-resilient strong Ψ_i-pulser. That is, $\mathbf{P}_i$ generates a pulse every Ψ_i rounds once stabilised, granted that V_i contains at most f_i faulty nodes. Nodes in block i execute the algorithm $\mathbf{P}_i$. 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 $a_i(v, t) \in \{0, 1\}$ indicate the output bit of $\mathbf{P}_i$ 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 $\mathbf{P}_i$ never stabilises. 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 $\mathbf{P}_i$ is *spurious*. We proceed by showing how to filter out such spurious pulses if they occur too often.

5.1.1 Filtering rules

We define five variables with the following semantics:

- $m_i(v, t + 1)$ indicates whether at least $n_i - f_i$ nodes $u \in V_i$ sent $a_i(u, t) = 1$,
- $M_i(v, t+1)$ indicates whether at least $n-f$ nodes $u \in V$ sent $m_i(u, t) = 1$,
- $\ell_i(v, t)$ indicates when was the last time block i triggered a (possibly spurious) pulse,
- $w_i(v, t)$ is a *cooldown counter* that indicates how many rounds any firing events coming from block i are ignored, and
- $b_i(v, t)$ 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:

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

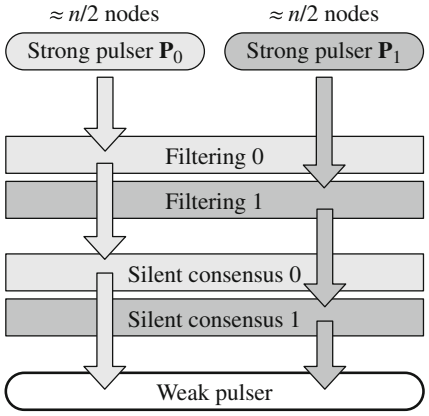

Fig. 5 Overview of the weak pulser construction. Light and dark grey boxes correspond to steps of block 0 and 1, respectively. The small rounded boxes denote the pulser algorithms $\mathbf{P}_i$ that are run (in parallel) on two disjoint sets of roughly $n/2$ nodes, whereas the wide rectangular boxes denote to the filtering steps in which all of the n nodes are employed. The arrows indicate the flow of information for each block

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

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

where $y = \min\{\Psi_i, \ell_i(v,t)+1\}$. In words, the counter is reset on round $t+1$ if v has proof that at least one correct node u had $m_i(u,t) = 1$, that is, some u observed $\mathbf{P}_i$ generating a (possibly spurious) pulse.

We reset the cooldown counter w_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
- 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

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

where $y = \max\{w_i(v,t)-1, 0\}$ 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 $b_i(v,t)$ indicates whether node v accepted a pulse from block i on round t. The variable is set using the rule

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

5.2 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 $\mathbf{P}_i$ run by a correct block i will stabilise. This yields the following lemma.

Lemma 2 *For some $i \in \{0,1\}$, the strong pulser algorithm $\mathbf{P}_i$ stabilises by round $T(\mathbf{P}_i)$.*

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 3 *If block i is correct, then there exists a round $r_0 \leq T(\mathbf{P}_i)+2$ such that for each $v \in V \setminus F$, $M_i(v,t) = 1$ if and only if $t = r_0 + k\Psi_i$ for $k \in \mathbb{N}_0$.*

Proof If block i is correct, then the algorithm $\mathbf{P}_i$ stabilises by round $T(\mathbf{P}_i)$. Hence, there is some $t_0 \leq T(\mathbf{P})$ so that the output variable $a_i(\cdot)$ of $\mathbf{P}_i$ satisfies

$$a_i(v,t) = 1 \text{ if and only if } t = t_0 + k\Psi_i \text{ for } k \in \mathbb{N}_0$$

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

If $\mathbf{P}_i$ generates a pulse on round $t \geq t_0$, then at least $n_i - f_i$ correct nodes $u \in V_i \setminus F$ have $a_i(u,t) = 1$. Therefore, for

all $v \in V \setminus F$ we have $m_i(v, t+1) = 1$, and consequently, $M_i(v, t+2) = 1$. Since block i is correct, there are at most f_i faulty nodes in the set V_i. Observe that by Lemma 1 strong pulsers solve synchronous counting, which in turn is as hard as consensus [11]. This implies that we must have $f_i < n_i/3$, as $\mathbf{P}_i$ is a strong f_i-resilient pulser for n_i nodes. Therefore, if $\mathbf{P}_i$ does not generate a pulse on round $t \geq t_0$, then at most $f_i < n_i - f_i$ faulty nodes u may claim $a_i(u, t) = 1$. This yields that $m_i(v, t+1) = M_i(v, t+2) = 0$ for all $v \in V \setminus F$. □

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

Lemma 4 *If block i is correct, then there exists a round $t_0 \leq T(\mathbf{P}_i) + 2C$ such that for each $v \in V \setminus F$, $b_i(v, t) = 1$ for any $t \geq t_0$ if and only if $t = t_0 + k\Psi_i$ for $k \in \mathbb{N}_0$.*

Proof Lemma 3 implies that there exists $r_0 \leq T(\mathbf{P}_i) + 2$ such that both $M_i(v, t) = 1$ and $\ell_i(v, t) = 0$ hold for $t \geq r_0$ if and only if $t = r_0 + k\Psi_i$ for $k \in \mathbb{N}_0$. It follows that $w_i(v, t+1) = \max\{w_i(v, t) - 1, 0\}$ for all such t and hence $w_i(v, t') = 0$ for all $t' \geq r_0 + C + 2$. The claim now follows from the definition of $b_i(v, t')$, the choice of r_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 5 *Let $v, v' \in V \setminus F$ and $t > 2$. Suppose that $b_i(v, t) = 1$ and $t' > t$ is minimal such that $b_i(v', t') = 1$. Then $t' = t + \Psi_i$ or $t' > t + C$.*

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

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

5.3 Introducing silent consensus

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 no correct nodes. In order to achieve this, we employ *silent consensus.*

Definition 3 (*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. We show that silent consensus protocols can be obtained from non-silent ones using a simple transformation.

Theorem 4 *Any consensus protocol $\mathbf{C}$ can be transformed into a silent binary consensus protocol $\mathbf{C}'$ with $T(\mathbf{C}') = T(\mathbf{C}) + 2$ and the same resilience and message size.*

Proof The new protocol $\mathbf{C}'$ can be seen as a "wrapper" protocol that manipulates the inputs and then lets each node decide whether it participates in an instance of the original protocol. The output of the original protocol, $\mathbf{C}$, will be taken into account only by correct nodes that participate throughout the protocol, as specified below.

In the first round of the new protocol, $\mathbf{C}'$, each participating node broadcasts its input if it is 1 and otherwise sends nothing. If a node receives fewer than $n - f$ times the value 1, it sets its input to 0. In the second round, the same pattern is applied.

Subsequently, $\mathbf{C}$ is executed by all nodes that received at least $f + 1$ messages in the first round. If during the execution of $\mathbf{C}$ a node

(i) cannot process the messages received in a given round in accordance with $\mathbf{C}$ (this may happen e.g. when not all of the correct nodes participate in the instance, which is not covered by the model assumptions of $\mathbf{C}$),
(ii) would have to send more bits than it would have according to the known bound $M(\mathbf{C})$, or
(iii) would violate the running time bound of $\mathbf{C}$,

then the node (locally) aborts the execution of $\mathbf{C}$. Finally, a node outputs 0 in the new protocol if it did not participate in the execution of $\mathbf{C}$, aborted it, or received f or fewer messages in the second round, and it outputs the result according to the run of $\mathbf{C}$ otherwise.

We first show that the new protocol, $\mathbf{C}'$, is a consensus protocol with the same resilience as $\mathbf{C}$ and the claimed bounds on communication complexity and running time. We distinguish two cases. First, suppose that all correct nodes participate in the execution of $\mathbf{C}$ at the beginning of the third round. As all nodes participate, the bounds on resilience, communication

complexity, and running time that apply to $\mathbf{C}$ hold in this execution, and no node will quit executing the protocol before termination. To establish agreement and validity, again we distinguish two cases. If all nodes output the outcome of the execution of $\mathbf{C}$, these properties follow right away since $\mathbf{C}$ satisfies them; here we use that although the initial two rounds might affect the inputs of nodes, a node will change its input to 0 only if there is at least one correct node with input 0. On the other hand, if some node outputs 0 because it received f or fewer messages in the second round of $\mathbf{C}'$, no node received more than $2f < n-f$ messages in the second round. Consequently, all nodes executed $\mathbf{C}$ with input 0 and computed output 0 by the agreement property of $\mathbf{C}$, implying agreement and validity of the new protocol.

The second case is that some correct node does not participate in the execution of $\mathbf{C}$. Thus, it received at most f messages in the first round of $\mathbf{C}'$, implying that no node received more than $2f < n-f$ messages in this round. Consequently, correct nodes set their input to 0 and will not transmit in the second round. While some nodes may execute $\mathbf{C}$, all correct nodes will output 0 no matter how $\mathbf{C}$ behaves. Since nodes abort the execution of $\mathbf{C}$ if the bounds on communication or time complexity are about to be violated, the claimed bounds for the new protocol hold.

It remains to show that the new protocol is silent. Clearly, if all correct nodes have input 0, they will not transmit in the first two rounds. In particular, they will not receive more than f messages in the first round and not participate in the execution of $\mathbf{C}$. Hence correct nodes do not send messages at all, as claimed. □

For example, plugging in the phase king protocol [3], we get the following corollary.

Corollary 2 *For any $f < n/3$, there exists a deterministic f-resilient silent binary consensus protocol $\mathbf{C}$ with $T(\mathbf{C}) \in \Theta(f)$ and $M(\mathbf{C}) \in O(1)$.*

5.4 Using silent consensus to prune spurious pulses

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 $b_i(v,t) \in \{0,1\}$ indicates whether v observed a (filtered) pulse of the strong pulser $\mathbf{P}_i$ in round t. Moreover, assume that $\mathbf{C}$ is a silent consensus protocol running in $T(\mathbf{C})$ rounds. We use two copies $\mathbf{C}_i$, where $i \in \{0,1\}$, of the consensus routine $\mathbf{C}$. We require that $\Psi_i \geq T(\mathbf{C})$, which guarantees by Lemma 5 that (after stabilisation) every instance of $\mathbf{C}$ has sufficient time to complete. Adding one more level of voting to clean up the inputs, we arrive at the following routine.

5.4.1 The pruning algorithm

Besides the local variables of $\mathbf{C}_i$, the algorithm will use the following variables for each $v \in V$ and round $t \in \mathbb{N}$:

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

Now each node v executes the following on round t:

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

   Otherwise, set $B_i(v,t+1) = 0$.
5. If $r_i(v,t) \notin \{T(\mathbf{C}), \bot\}$, then
   (a) execute round $r_i(v,t)$ of $\mathbf{C}_i$, and
   (b) set $r_i(v,t+1) = r_i(v,t) + 1$.

5.4.2 Analysis

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

We say that $v \in V \setminus F$ *executes the round* $r \in \{1,\dots,T(\mathbf{C})\}$ of $\mathbf{C}_i$ in round t iff $r_i(v,t) = r$. By Lemma 5, in rounds $t > T(\mathbf{C}) + 2$, there is always at most one instance of $\mathbf{C}_i$ being executed, and if so, consistently.

Corollary 3 *Suppose that $v \in V \setminus F$ executes round 1 of $\mathbf{C}_i$ in some round $t > T(\mathbf{C}) + 2$. Then there is a subset $U \subseteq V \setminus F$ such that each $u \in U$ executes round $r \in \{1,\dots,T(\mathbf{C})\}$ of $\mathbf{C}_i$ in round $t + r - 1$ and no $u \in V \setminus (F \cup U)$ executes any round of $\mathbf{C}_i$ in round $t + r - 1$.*

Exploiting silence of $\mathbf{C}_i$ and the choice of inputs, we can ensure that the case $U \neq V \setminus F$ causes no trouble.

Lemma 6 *Let $t > T(\mathbf{C})+2$ and U be as in Corollary 3. Then $U = V \setminus F$ or each $u \in U$ has input 0 for the respective instance of $\mathbf{C}_i$.*

Proof Suppose that $u \in U$ starts an instance with input 1 in round $t' \in \{t - T(\mathbf{C}) - 1, \dots, t\}$. Then $b_i(w, t'-1) = 1$ for at least $n - 2f$ nodes $w \in V \setminus F$, since u received $b_i(u, w, t'-1) = 1$ from $n - f$ nodes $w \in V$. Thus, each $v \in V \setminus F$ received $b_i(v, w, t'-1) = 1$ from at least $n-2f$ nodes w and sets $r_i(v, t') = 1$, i.e., $U = V \setminus F$. The lemma now follows from Corollary 3. □

Recall that if all nodes executing $\mathbf{C}_i$ have input 0, non-participating correct nodes behave exactly as if they executed $\mathbf{C}_i$ as well, i.e., they send no messages. Hence, if $U \neq V \setminus F$, all nodes executing the algorithm will compute output 0. Therefore, Corollary 3, Lemma 5, and Lemma 6 imply the following corollary.

Corollary 4 *In rounds $t > T(\mathbf{C}) + 2$ it holds that $B_i(v, t) = B_i(w, t)$ for all $v, w \in V \setminus F$ and $i \in \{0, 1\}$. Furthermore, if $B_i(v, t) = 1$ for $v \in V \setminus F$ and $t > T(\mathbf{C}) + 2$, then the minimal $t' > t$ so that $B_i(v, t') = 1$ (if it exists) satisfies either $t' = t + \Psi_i$ or $t' > t + C = t + \max\{\Psi_0, \Psi_1\} + \Phi + 2$.*

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

Lemma 7 *If block i is correct, there is a round $t_0 \leq T(\mathbf{P}_i) + 2C + T(\mathbf{C}) + 1$ so that for any $t \geq t_0$, $B_i(v, t) = 1$ if and only if $t = t_0 + k\Psi_i$ for some $k \in \mathbb{N}_0$.*

Proof Lemma 4 states the same for the variables $b_i(v, t)$ and a round $t_0' \leq T(\mathbf{P}_i)+2C$. If $b_i(v, t) = 1$ for all $v \in V \setminus F$ and some round t, all correct nodes start executing an instance of $\mathbf{C}_i$ with input 1 in round $t+1$. As, by Corollary 3, this instance executes correctly and, by validity of $\mathbf{C}_i$, outputs 1 in round $t + T(\mathbf{C})$, all correct nodes satisfy $B_i(v, t + T(\mathbf{C}) + 1) = 1$. Similarly, $B_i(v, t + T(\mathbf{C}) + 1) = 0$ for such v and any $t \geq t_0'$ with $b_i(v, t) = 0$. □

5.5 Obtaining the weak pulser

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

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

As we have eliminated the possibility that $B_i(v, t) \neq B_i(w, t)$ for $v, w \in V \setminus F$ and $t > T(\mathbf{C}) + 2$, Property W1 holds. Since there is at least one correct block i by Lemma 2, Lemma 7 shows that there will be good pulses (satisfying Properties W2 and W3) regularly, unless block $1 - i$ interferes by generating pulses violating Property W3 (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 4), 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 $O(\Phi)$ rounds after the B_i stabilised.

Lemma 8 *In the construction described in the previous two subsections, choose $\Psi_0 = 2\Phi$ and $\Psi_1 = 3\Phi$ for any $\Phi \geq T(\mathbf{C})$. Then $B(v, t)$ is the output variable of a weak Φ-pulser with stabilisation time $\max\{T(\mathbf{P}_0), T(\mathbf{P}_1)\} + O(\Phi)$.*

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

$$\begin{aligned} t &\in \max\{T(\mathbf{P}_0), T(\mathbf{P}_1)\} + T(\mathbf{C}) + O(\Phi) \\ &\subseteq \max\{T(\mathbf{P}_0), T(\mathbf{P}_1)\} + O(\Phi) \end{aligned}$$

satisfying the following four properties. For either block $i \in \{0, 1\}$, we have by Corollary 4 that

1. $B_i(v, t') = B_i(w, t')$ and $B(v, t') = B(w, t')$ for any $v, w \in V \setminus F$ and $t' \geq t$.

Moreover, for a correct block i and for all $v \in V \setminus F$ we have from Lemma 7 that

2. $B_i(v, t) = B_i(v, t + \Psi_i) = 1$,
3. $B_i(v, t') = 0$ for all $t' \in \{t + 1, \dots, t + \Phi - 1\} \cup \{t + \Psi_i + 1, \dots, t + \Psi_i + \Phi - 1\}$,

and for a (possibly faulty) block $1 - i$ we have from Corollary 4 that

4. if $B_{1-i}(v, t') = 1$ for some $v \in V \setminus F$ and $t' \in \{t + 1, \dots, t + \Psi_i + \Phi - 1\}$, then $B_{1-i}(u, t'') = 0$ for all $u \in V \setminus F$ and $t'' \in \{t' + 1, \dots, t' + C\}$ that do not satisfy $t'' = t' + k\Psi_{1-i}$ 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 2. By the first property, it suffices to show that a good pulse occurs in round t or in round $t + \Psi_i$. From the second property, we get for all $v \in V \setminus F$ that $B(v, t) = 1$ and $B(v, t + \Psi_i) = 1$. If the pulse in round t is good, the claim holds. Hence, assume that there is a round $t' \in \{t + 1, \dots, t + \Psi_i - 1\}$ in which another pulse occurs, that is, $B(v, t') = 1$ for some $v \in V \setminus F$. This entails that $B_{1-i}(v, t') = 1$ by the third property. We claim that in this case the pulse in round $t + \Psi_i$ is good. To show this, we exploit the fourth property. Recall that $C > \Psi_i + \Phi$, i.e., $t' + C > t + \Psi_i + \Phi$. We distinguish two cases:

– In the case $i = 0$, we have that $t' + \Psi_{1-i} = t' + 3\Phi = t' + \Psi_0 + \Psi > t + \Psi_0 + \Phi$, that is, the pulse in round $t + \Psi_0 = t + \Psi_i$ is good.
– In the case $i = 1$, we have that $t' + \Psi_{1-i} = t' + 2\Phi < t + 3\Phi = t + \Psi_1$ and $t' + 2\Psi_{1-i} = t' + 4\Phi = t' + \Psi_1 + \Phi > t + \Psi_1 + \Phi$, that is, the pulse in round $t + \Psi_1 = t + \Psi_i$ is good.

In either case, a good pulse occurs by round

$$t + \max\{\Psi_0, \Psi_1\} \in \max\{T(\mathbf{P}_0), T(\mathbf{P}_1)\} + O(\Phi).$$

□

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

Theorem 5 *Let $n = n_0 + n_1$ and $f = f_0 + f_1 + 1$, where $n > 3f$. Suppose that $\mathbf{C}$ is an f-resilient consensus algorithm on n nodes and let $\Phi \geq T(\mathbf{C}) + 2$. 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 $\mathbf{W}$ on n nodes that satisfies*

– $T(\mathbf{W}) \in \max\{T(\mathbf{P}_0), T(\mathbf{P}_1)\} + O(\Phi)$,
– $M(\mathbf{W}) \in \max\{M(\mathbf{P}_0), M(\mathbf{P}_1)\} + O(M(\mathbf{C}))$.

Proof By Theorem 4, we can transform $\mathbf{C}$ into a silent consensus protocol $\mathbf{C}'$, at the cost of increasing its round complexity by 2. Using $\mathbf{C}'$ in the construction, Lemma 8 shows that we obtain a weak Φ-pulser with the stated stabilisation time, which by construction tolerates f faults. Concerning the message size, note that we run $\mathbf{P}_0$ and $\mathbf{P}_1$ on disjoint node sets. Apart from sending $\max\{M(\mathbf{P}_0), M(\mathbf{P}_1)\}$ bits per round for its respective strong pulser, each node may send $M(\mathbf{C})$ bits each to each other node for the two copies $\mathbf{C}_i$ of $\mathbf{C}$ it runs in parallel, plus a constant number of additional bits for the filtering construction including its outputs $b_i(\cdot, \cdot)$. □

6 Main results

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 6 *Suppose that we are given a family of f-resilient deterministic consensus algorithms $\mathbf{C}(f)$ running on any number $n > 3f$ of nodes in $T(\mathbf{C}(f))$ rounds using $M(\mathbf{C}(f))$-bit messages, where both $T(\mathbf{C}(f))$ and $M(\mathbf{C}(f))$ 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 $\mathbf{P}$ on n nodes that stabilises in time*

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

and uses messages of size at most

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

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

Proof We show by induction on k that f-resilient strong Ψ-pulsers $\mathbf{P}(f, \Psi)$ on $n > 3f$ nodes with the stated complexity exist for any $f < 2^k$, with the addition that the (bounds on) stabilisation 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 stabilises 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 $\mathbf{P}(f', \Psi')$ denote a strong Ψ'-pulser that exists by the induction hypothesis for $f' < 2^k$.

Choose $\Phi \in O(\log \Psi) + T(\mathbf{C}(f))$ in accordance with Theorem 1 for $L = \Psi$; without loss of generality we may assume that the $O(\log \Psi)$ term is at least 2, that is, $\Phi \geq 2 + T(\mathbf{C}(f))$. We apply Theorem 5 to $\mathbf{C}(f)$ and $\mathbf{P}_i = \mathbf{P}(f_i, \Psi_i)$, where $\Psi_0 = 2\Phi$ and $\Psi_1 = 3\Phi$, to obtain a weak Φ-pulser $\mathbf{W}$ with resilience f on n nodes and stabilisation time of

$$T(\mathbf{W}) \in \max\{T(\mathbf{P}_0), T(\mathbf{P}_1)\} + O(\Phi),$$

and message size of

$$M(\mathbf{W}) \in \max\{M(\mathbf{P}_0), M(\mathbf{P}_1)\} + O(M(\mathbf{C}(f))).$$

Next, we apply Theorem 1 to $\mathbf{C}(f)$ to obtain an f-resilient Ψ-value consensus protocol $\mathbf{C}'$ that uses messages of size $M(\mathbf{C}(f))$ bits and runs in $T(\mathbf{C}') \leq \Phi$ rounds. We feed the weak pulser $\mathbf{W}$ and the multivalued consensus protocol $\mathbf{C}'$ into Corollary 1 to obtain an f-resilient strong Ψ-pulser $\mathbf{P}$ that stabilises in

$$\begin{aligned} T(\mathbf{P}) &\leq T(\mathbf{C}') + T(\mathbf{W}) + \Psi \leq T(\mathbf{W}) + \Psi + \Phi \\ &\in \max\{T(\mathbf{P}_0), T(\mathbf{P}_1)\} + \Psi + O(\Phi) \end{aligned}$$

rounds and has message size bounded by

$$M(\mathbf{P}) \leq M(\mathbf{W}) + M(\mathbf{C}(f))$$

$$\in \max\{M(\mathbf{P}_0), M(\mathbf{P}_1)\} + O(M(\mathbf{C}(f))).$$

Applying the bounds given by the induction hypothesis to $\mathbf{P}_0$ and $\mathbf{P}_1$, the definitions of Φ, Ψ_0 and Ψ_1, and the fact that both $T(\mathbf{C}(f))$ and $M(\mathbf{C}(f))$ are non-decreasing in f, we get that the stabilisation time satisfies

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

and message size is bounded by

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

Because we bounded complexities using $\max_i\{T(\mathbf{P}_i)\}$, $\max\{M(\mathbf{P}_i)\}$, $T(\mathbf{C}(f))$ and $M(\mathbf{C}(f))$, all of which are non-decreasing in f by assumption, we also maintain that the new bounds on stabilisation 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 [3], which has optimal resilience, running time $O(f)$, and constant message size, we can extract a strong pulser that is optimally resilient, has asymptotically optimal stabilisation time, and message size $O(\log f)$.

Corollary 5 *For any $\Psi, f \in \mathbb{N}$ and $n > 3f$, an f-resilient strong Ψ-pulser on n nodes with stabilisation time $(1 + o(1))\Psi + O(f)$ and message size $O(\log f)$ exists.*

We obtain efficient solutions to the firing squad and synchronous counting problems.

Corollary 6 *For any $f \in \mathbb{N}$ and $n > 3f$, an f-resilient firing squad on n nodes with stabilisation and response times of $O(f)$ and message size $O(\log f)$ exists.*

Proof We use Corollary 5 with $\Psi \in O(f)$ being the running time of the phase king protocol [3], followed by applying Theorem 2 to the obtained pulser and the phase king protocol. □

Corollary 7 *For any $C, f \in \mathbb{N}$ and $n > 3f$, an f-resilient C-counter on n nodes with stabilisation time $O(f + \log C)$ and message size $O(\log f)$ exists.*

Proof In the last step of the construction of Theorem 6, we do not use Corollary 1 to extract a strong pulser, but directly obtain a counter using Theorem 3. 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 $O(\log \Psi)$ additive overhead in time of the multi-value consensus routine, the claim follows. □

We remark that one can strengthen the bound on the stabilisation time to $O(f + (\log \Psi)/B)$ using messages of size B, by using larger messages in the reduction given by Theorem 1 [25]. However, this affects the asymptotic stabilisation time only if Ψ is super-exponential in f.

7 Randomised sublinear-time algorithms

So far, we have confined our discussion to the deterministic setting. However, it is possible to adapt our framework to also utilise *randomised* consensus routines, which can break the linear-in-f time bound for consensus [20] and attain better bit complexities than deterministic algorithms [23]. Indeed, Ben-Or et al. [2] have shown how to obtain randomised counting algorithms that stabilise in $O(1)$ expected time given a shared coin. However, implementations of shared coins are expensive in terms of communication.

As an example, we use our framework to obtain communication-efficient *randomised* pulsers that stabilise in polylogarithmic time with polylogarithmic message size. Here, the stabilisation time has probabilistic guarantees, but once stabilisation is successful, then the correct behaviour deterministically persists.

7.1 Randomised consensus algorithms

Randomised consensus algorithms can give probabilistic guarantees on any of the termination, agreement, or validity conditions. Las Vegas algorithms have randomised running time (i.e. probabilistic guarantee on the termination condition) while having always (deterministically) correct output. In contrast, Monte Carlo algorithms have probabilistic guarantees on the correctness (i.e. validity and/or agreement) with a deterministic running time.

For our constructions, we need Monte Carlo algorithms that have a deterministic guarantee on the running time *and*

the validity constraint, while having only a probabilistic guarantee on reaching agreement.

Definition 4 (*Probabilistic consensus*) Let $L > 1$ and $p \in (0, 1)$. Suppose that each node $v \in V$ receives an input value $x(v) \in [L]$. We say that an algorithm $\mathbf{C}$ is an f-resilient probabilistic consensus algorithm with agreement probability $p \in [0, 1]$ if the following conditions hold when there are at most f faulty nodes:

PC1 **Termination:** Each $v \in V \setminus F$ decides on an output $y(v) \in [L]$ by the end of round $T(\mathbf{C})$.

PC2 **Agreement:** For any given input $x(\cdot)$, it holds that $y(v) = y(w)$ for all $v, w \in V \setminus F$ with probability at least p.

PC3 **Validity:** If there exists $x \in [L]$ such that for all $v \in V \setminus F$ it holds that $x(v) = x$, then each $v \in V \setminus F$ outputs the value $y(v) = x$.

Note that the validity condition PC3 ensures that agreement is deterministic if the inputs of correct nodes already agree.

For the sake of completeness, we show how to obtain a probabilistic consensus algorithm that satisfies deterministically conditions PC1 and PC3 from an algorithm that deterministically only satisfies PC1. This is done by devising a wrapper for $\mathbf{C}$, which is essentially a variant of a phase king algorithm [3], where the consensus algorithm $\mathbf{C}$ is used as a tie breaker instead of relying on a node that acts as a"king".

Lemma 9 *Let $f < n/3$ and $\mathbf{C}$ be an f-resilient consensus algorithm that deterministically terminates in $T(\mathbf{C})$ rounds while satisfying validity and agreement conditions with probability at least p. Then there is a probabilistic consensus algorithm $\mathbf{C}'$ that deterministically terminates in $T(\mathbf{C}) + O(1)$ rounds, deterministically satisfies validity, and satisfies agreement with probability at least p. Moreover, $M(\mathbf{C}') \in O(1) + M(\mathbf{C})$.*

Proof Let $x(v) \in [L]$ be the input value of node v. Node v runs the following simulation wrapper for $\mathbf{C}$:

1. Broadcast the input value $x(v)$ to all nodes.
2. If $x(v, w) = x(v)$ for at least $n - f$ nodes $w \in V$, then set $z(v) = x(v)$. Otherwise, set $z(v) = \infty$. Broadcast the value of $z(v)$ to all nodes.
3. Define $Z_a(v) = \{w \in V : z(v, w) = a\}$. Let $z'(v) = \min\{a : |Z_a(v)| \geq f + 1\} \cup \{\infty\}$. If $z'(v) = z(v) \neq \infty$ and $|Z_{z(v)}(v)| \geq n - f$, set $b(v) = 1$. Otherwise, set $b(v) = 0$.
4. Simulate $\mathbf{C}$ for $T(\mathbf{C})$ rounds using the input value

$$x'(v) = \begin{cases} z'(v) & \text{if } z'(v) \neq \infty, \\ 0 & \text{otherwise.} \end{cases}$$

5. Once $\mathbf{C}$ terminates with output $y'(v)$, node v decides on the value

$$y(v) = \begin{cases} z'(v) & \text{if } b(v) = 1, \\ y'(v) & \text{otherwise.} \end{cases}$$

Steps 1–3 take 2 communication rounds and Step 4 takes $T(\mathbf{C})$ communication rounds. The bound on the message size follows from the fact that the first three steps use constant-size messages and the simulation of $\mathbf{C}$ uses messages of size at most $M(\mathbf{C})$.

Condition PC1 is trivially satisfied, as the algorithm terminates in $T(\mathbf{C}) + 2$ rounds. For validity (Condition PC3), observe that if $x(v) = x(w)$ holds for all correct nodes $v, w \in V \setminus F$, then $z(v) = z'(v) = x(v)$ holds. Thus, in Step 5 each correct v ignores the output of $\mathbf{C}$ and sets $y(v) = x(v)$. This implies that validity is deterministically satisfied.

It remains to show that agreement is satisfied with probability p. There are two cases. First, if $b(v) = 0$ for all $v \in V \setminus F$, then in Step 5 each v uses the output of the simulated algorithm $\mathbf{C}$, which by assumption reaches agreement with probability at least p. For the second case, suppose that for some correct $v \in V \setminus F$ we have $b(v) = 1$. This means that v will still participate in the simulation of $\mathbf{C}$, but will instead use the value $y(v) = z'(v) \neq \infty$ as output in Step 5.

Note that if $b(v) = 1$ for some correct node $v \in V \setminus F$, we have $z'(v) = z'(w)$ for all $v, w \in V \setminus F$. To see why, let $a = z'(v)$ and without loss of generality assume that $n = 3f + 1$ and there are exactly f faulty nodes. Since $b(v) = 1$, we have that at least $n-2f$ correct nodes $w \in V \setminus F$ sent the value $z'(v) = z(w) = a$ to v. In particular, these nodes must have received the value from $n-2f > f$ correct nodes during Step 2. Now suppose that some node $u \in V \setminus F$ has $z'(u) = a' \neq a$. Then there must have been $n - 2f > f$ correct nodes that sent the value a' to u in Step 2. Thus, there are at least $n - 2f$ correct nodes with $z(\cdot) = a$ and at least $n - 2f$ correct nodes with $z(\cdot) = a'$. This implies that there are in total $2(n - 2f) + f > 3f + 2$ nodes, which is a contradiction.

Since all correct nodes $w \in V \setminus F$ have $z'(w) = a$, every node uses the same input for $\mathbf{C}$ and the execution of $\mathbf{C}$ satisfies agreement and validity with probability at least p. Thus, in either case condition PC2 is satisfied. □

7.2 Using probabilistic consensus

We now extend the deterministic framework described earlier to work with probabilistic consensus routines. We say that an execution of probabilistic consensus routine $\mathbf{C}$ is *successful* if the output variables of all correct nodes agree.

Remark 1 Suppose that $\mathbf{C}$ is a probabilistic consensus routine with agreement probability p. Let $X(\mathbf{C})$ be the random

variable denoting the number of trials until a sequence of $T(\mathbf{C})$-round executions of $\mathbf{C}$ yield a successful execution. Then $X(\mathbf{C})$ is upper bounded by a geometric distribution, i.e., it holds that the expectated value of $X(\mathbf{C})$ satisfies $\mathbf{E}[X(\mathbf{C})] \leq 1/p$, regardless of the choice of inputs in each trial.

First, we show the randomised variant of Theorem 3. We can use the same strong pulser algorithm as in Sect. 4.2 with a slightly modified analysis. From now on, for weak and strong pulsers, respectively, denote by $E(\mathbf{W})$ and $E(\mathbf{P})$ (upper bounds on) the *expected* stabilisation time that hold in any execution. Note that $T(\mathbf{C})$ is not a random variable, as the consensus algorithms we consider have deterministic running time.

Theorem 7 (Probabilistic variant of Theorem 3) *Given a probabilistic consensus routine with agreement probability p and a weak pulser $\mathbf{W}$ that stabilises in expected $R(\mathbf{W})$ rounds, the strong pulser algorithm of Sect. 4.2 implements a synchronous Ψ-counter that stabilises in at most $E(\mathbf{W})/p + T(\mathbf{C}) + 1$ rounds in expectation and uses messages of at most $M(\mathbf{W}) + M(\mathbf{C})$ bits.*

Proof We need to adapt the analysis of Theorem 3 to take into account the probabilistic guarantee on stabilisation. Let the random variable t_i be the round on which the ith good pulse of $\mathbf{W}$ is generated. As before, all correct nodes simulate $\mathbf{C}$ during the rounds $t_i + 1, \ldots, t_i + T(\mathbf{C}) + 1$. If the simulation of $\mathbf{C}$ is successful, then the original analysis in Theorem 3 holds as is. However, since $\mathbf{C}$ is a probabilistic consensus routine, the simulation of $\mathbf{C}$ starting in round $t_i + 1$ may fail to satisfy agreement with probability at most $1 - p$.

By Remark 1, we get that the expected number of attempts until a successful execution of $\mathbf{C}$ is bounded by $1/p$; counting only attempts where $\mathbf{C}$ is simulated correctly, this bound holds independently of the inputs to $\mathbf{C}$. Note that once a simulation attempt of $\mathbf{C}$ results in a successful execution, then all subsequent attempts will also be successful, as validity will be trivially satisfied.

Accordingly the strong pulser algorithm stabilises by round $t_X + T(\mathbf{C}) + 1$, where $X = X(\mathbf{C})$ is the random variable counting the number of good pulses of the weak pulser and t_X denotes the round when the respective good pulse occurs, which is also a random variable. By the above considerations, we can bound the expected stabilisation time of the strong pulser by

$$\begin{aligned}
\mathbf{E}[t_X + T(\mathbf{C}) + 1] &= T(\mathbf{C}) + 1 + \mathbf{E}[t_X] \\
&= T(\mathbf{C}) + 1 + \sum_{x=1}^{\infty} \Pr[X = x] \cdot \sum_{t=1}^{\infty} \Pr[t_x = t \mid X = x] \cdot t_x \\
&\leq T(\mathbf{C}) + 1 + \sum_{x=1}^{\infty} \Pr[X = x] \cdot x \cdot E(\mathbf{W}) \\
&= T(\mathbf{C}) + 1 + \mathbf{E}[X] \cdot E(\mathbf{W}) \\
&\leq E(\mathbf{W})/p + T(\mathbf{C}) + 1\,,
\end{aligned}$$

where in the first and third step we used linearity of expectation. As before, the message size is bounded by $M(\mathbf{W}) + M(\mathbf{C})$. □

Now Lemma 1 together with Theorem 7 implies the randomised variant of Corollary 1.

Corollary 8 *Let $\Psi > 1$. Suppose that there exists an f-resilient Ψ-value probabilistic consensus routine $\mathbf{C}$ with agreement probability p and a weak (possibly randomised) Φ-pulser $\mathbf{W}$, where $\Phi \geq T(\mathbf{C})$. Then there exists a randomised f-resilient strong Ψ-pulser $\mathbf{P}$ that*

- *stabilises in expected $E(\mathbf{P}) \leq T(\mathbf{C}) + E(\mathbf{W})/p + \Psi$ rounds, and*
- *uses messages of size at most $M(\mathbf{P}) \leq M(\mathbf{C}) + M(\mathbf{W})$ bits.*

Next we need to analyse the construction of weak pulsers when utilising probabilistic consensus routines. Note that both weak and strong pulsers behave deterministically after stabilisation and only the stabilisation time is a random variable. Thus, Theorem 5 is straightforward to adapt to the probabilistic setting. The key changes are in the analysis involving the use of the consensus routines in Sects. 5.4 and 5.5.

Theorem 8 (Probabilistic variant of Theorem 5) *Let $n = n_0 + n_1$ and $f = f_0 + f_1 + 1$, where $n > 3f$. Suppose that $\mathbf{C}$ is an f-resilient probablistic consensus algorithm with agreement probability $p \geq 1/2$ on n nodes and let $\Phi \geq T(\mathbf{C}) + 2$. 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 $\mathbf{W}$ on n nodes that satisfies*

- $E(\mathbf{W}) \in \max\{E(\mathbf{P}_0), E(\mathbf{P}_1)\} + O(\Phi)$,
- $M(\mathbf{W}) \in \max\{M(\mathbf{P}_0), M(\mathbf{P}_1)\} + O(M(\mathbf{C}))$.

Proof First, note that we can also transform a probabilistic consensus protocol $\mathbf{C}$ into a silent probabilistic consensus protocol as before using Theorem 4. Since the strong pulsers $\mathbf{P}_i$ behave deterministically after stabilisation, Lemma 8 can be applied as is with the difference that the $E(\mathbf{P}_i)$ for $i \in \{0, 1\}$ denote the (upper bounds on the) expected stabilisation times under the assumption that block i is correct. Moreover, the bound on the message size follows from the same arguments as in Theorem 5. □

It remains to check that the recursive construction of Theorem 6 also works in the probabilistic setting: once lower levels stabilise, they behave deterministically, and thus, we

can simply repeat the analysis as before (using Theorem 8 instead of Theorem 5) and apply linearity of expectation in the inductive step.

Theorem 9 *Suppose that we are given a family of f-resilient probabilistic consensus algorithms $\mathbf{C}(f)$ with agreement probability $p \geq 1/2$ that run on any number $n > 3f$ of nodes in $T(\mathbf{C}(f))$ rounds using $M(\mathbf{C}(f))$-bit messages, where both $T(\mathbf{C}(f))$ and $M(\mathbf{C}(f))$ 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 $\mathbf{P}$ on n nodes that stabilises in expected time*

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

and uses messages of size at most

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

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

Proof We proceed by induction as in the original proof of Theorem 6. That is, we show that f-resilient randomised strong Ψ-pulsers $\mathbf{P}(f, \Psi)$ on $n > 3f$ nodes exist for any $f < 2^k$ by induction on k. As the analysis on the message size is the same as in Theorem 6, we refrain from repeating it here and focus on the stabilisation time.

For the base case $k = 0$, we can recall that 0-resilient strong pulsers trivially exist. For the inductive step, we assume that $2^k \leq f < 2^{k+1}$ for $k \in \mathbb{N}_0$ and the claim holds for all $0 \leq f' < 2^k$. As before, we can pick $f_i < 2^k$ and $n_i > 3f_i$ to satisfy $n = n_0 + n_1$ and $n > 3f$. Let $\mathbf{P}(f', \Psi')$ denote a strong randomised Ψ'-pulser that exists by the induction hypothesis for $f' < 2^k$.

Choose $\Phi \in O(\log \Psi) + T(\mathbf{C}(f))$ in accordance with Theorem 1 for $L = \Psi$ and apply Theorem 8 to $\mathbf{C}(f)$ and $\mathbf{P}_i = \mathbf{P}(f_i, \Psi_i)$, where $\Psi_0 = 2\Phi$ and $\Psi_1 = 3\Phi$. This yields a randomised weak Φ-pulser $\mathbf{W}$ with resilience f on n nodes and an expected stabilisation time bounded by

$$E(\mathbf{W}) \in \max\{E(\mathbf{P}_0), E(\mathbf{P}_1)\} + O(\Phi).$$

Applying Theorem 1 to $\mathbf{C}(f)$ gives an f-resilient Ψ-value consensus protocol $\mathbf{C}'$ that uses messages of size $M(\mathbf{C}(f))$ bits and runs deterministically in $T(\mathbf{C}') \leq \Phi$ rounds (but may fail with probability p). Together with the randomised weak pulser $\mathbf{W}$ and the multivalued consensus protocol $\mathbf{C}'$, Corollary 8 gives a randomised f-resilient strong Ψ-pulser $\mathbf{P}$ that stabilises in

$$\begin{aligned}
E(\mathbf{P}) &\leq T(\mathbf{C}') + E(\mathbf{W})/p + \Psi \\
&\leq E(\mathbf{W})/p + \Psi + \Phi \\
&\in (\max\{E(\mathbf{P}_0), E(\mathbf{P}_1)\} + O(\Phi))/p + \Psi + \Phi \\
&\subseteq 2\left(E(\mathbf{P}_0) + E(\mathbf{P}_1)\right) + \Psi + O(\Phi)
\end{aligned}$$

expected rounds, where the last step uses that $p \geq 1/2$.

Applying the bounds given by the induction hypothesis to $\mathbf{P}_0$ and $\mathbf{P}_1$, the definitions of Φ, Ψ_0 and Ψ_1, and the fact that $T(\mathbf{C}(f))$ is non-decreasing in f, we get that

$$\begin{aligned}
E(\mathbf{P}) &\in \left(E(\mathbf{P}_0) + E(\mathbf{P}_1)\right)/p + \Psi + O(\Phi) \\
&\subseteq (1+o(1)) \cdot 6\Phi + O\left(\sum_{j=0}^{\lceil \log 2^k \rceil} T\left(\mathbf{C}\left(2^j\right)\right)\right) \\
&\quad + \Psi + O(\Phi) \\
&\subseteq \Psi + O(\log \Psi) + O\left(\sum_{j=0}^{\lceil \log 2^k \rceil} T\left(\mathbf{C}\left(2^j\right)\right)\right) \\
&\quad + O(T(\mathbf{C}(f))) \\
&\subseteq (1+o(1))\Psi + O\left(\sum_{j=0}^{\lceil \log f \rceil} T\left(\mathbf{C}\left(2^j\right)\right)\right),
\end{aligned}$$

which completes the inductive step. □

7.3 Randomised pulsers, counting, and firing squads

As a concrete example, we illustrate our framework with the randomised consensus algorithm by King and Saia [23]. The algorithm assumes that (1) the number of faults is restricted to $f < n/(3+\varepsilon)$ (for arbitrarily small constant $\varepsilon > 0$) and (2) communication is via private channels, i.e., the behaviour of faulty nodes in round t is a function of all communication from correct nodes to faulty nodes in rounds $t' \leq t$.

Theorem 10 ([23]) *Let $f \in \mathbb{N}$, $\varepsilon > 0$ and $c > 1$ be constants, and $n > (3+\varepsilon)f$. Suppose that communication is via private channels. There exists a protocol $\mathbf{C}$ that with probability $p = 1 - 1/f^c$ solves consensus (i.e. satisfies agreement and validity), runs in* polylog f *rounds, and uses messages of size at most* polylog f.

We remark that the consensus algorithm from [23] actually limits the total number of bits sent by each node to $O(\sqrt{n}\,\text{polylog}\, n)$, but in our recursive framework each node broadcasts $\Omega(\log f)$ bits per round. By Lemma 9 we can convert the above algorithm into a probabilistic consensus algorithm that can be used in our framework yielding the following corollary of Theorem 9.

Corollary 9 *Let $n > (3+\varepsilon)f$, $f \in \mathbb{N}$ and $\varepsilon > 0$ be a constant and suppose communication is via private channels. Then for*

any $\Psi \in \mathbb{N}$ there exists a randomised f-resilient strong Ψ-pulser that stabilises in $(1 + o(1))\Psi +$ polylog f expected rounds and uses messages of size at most polylog f *bits.*

As a corollary, we can obtain a randomised firing squad algorithm that *with high probability* works correctly.

Corollary 10 *Let $n > (3 + \varepsilon)f$, $f \in \mathbb{N}$ and $\varepsilon > 0$ be a constant and suppose communication is via private channels. Then for a randomised f-resilient firing that stabilises in* polylog f *expected rounds and, satisfies FS1–FS3 with high probability, has response time of* polylog f *rounds, and uses messages of size at most* polylog f *bits.*

Proof We use Corollary 9 with $\Psi \in$ polylog f being the running time of the probabilistic consensus algorithm **C** given by Theorem 10. This yields obtain a probabilistic strong pulser **P**. Then we can use Theorem 2 with **P** and **C** to obtain a firing squad algorithm that satisfies agreement, safety, and liveness if an execution of **C** is successful, which happens with probability at least $p = 1 - 1/f^c$ for any sufficiently large constant c. □

Finally, as before, we can obtain synchronous counting algorithms from strong pulsers.

Corollary 11 *Let $n > (3 + \varepsilon)f$, $f \in \mathbb{N}$ and $\varepsilon > 0$ be a constant and suppose communication is via private channels. For any $C \in \mathbb{N}$, there exists a randomised f-resilient C-counter on n nodes with stabilisation time $O(\text{polylog } f + \log C)$ and message size* polylog f *bits.*

Proof The proof is similar to the deterministic version. In the last step of the construction of Theorem 9, we do not use Corollary 8 to extract a strong pulser, but directly obtain a counter using Theorem 7. 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 $O(\log \Psi)$ additive overhead in time of the multi-value consensus routine, the claim follows. □

8 Discussion

In this work we have given a framework for transforming non-self-stabilising consensus algorithms into self-stabilising synchronous counting and firing squad algorithms. In particular, our work shows that the ability to tolerate transient faults *in addition* to permanent faults does not induce a large hit on the complexity of fault-tolerant distributed coordination.

Our framework is modular in the sense that one can easily use any consensus algorithm—even randomised ones—with the framework. Moreover, one of the key features of our construction is that the resilience of the underlying consensus routine essentially dictates what kind of—and how many—permanent faults our self-stabilising counting and firing squad algorithms tolerate.

Here, we have restricted our attention to Byzantine faults, which is the most extreme form of faulty behavior, but the same ideas can be used to deal with other types of (more benign) permanently faulty behavior. For example, in the case of consensus, it is possible to tolerate any number of $f < n$ crash faults or $f < n/2$ omission faults. Our approach can also be applied to this setting with relatively minor modifications [26].

We conclude by highlighting a few open problems that still remain:

- The complexities of our algorithms depend on the maximum bound f on the number of permanent faults. Is it possible to obtain Byzantine-tolerant algorithms, where the stabilisation time depends on the *actual number* $t \leq f$ of faulty nodes?
- Can either synchronous counting or self-stabilising firing squads be deterministically solved using $o(\log f)$-bit messages under Byzantine faults or by communicating $o(f^2 \log f)$ bits overall?

Acknowledgements Open access funding provided by Institute of Science and Technology (IST Austria). We are grateful to Danny Dolev for inspiring discussions and valuable comments, especially concerning silent consensus. We thank the anonymous reviewers for their comments and helpful suggestions. Part of this work was done when JR was affiliated with the University of Helsinki and Aalto University.

Open Access This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

References

1. Angluin, D., Fischer, M.J., Jiang, H.: Stabilizing consensus in mobile networks. In: Proceedings of 2nd IEEE international conference on distributed computing in sensor systems, vol. 4026 LNCS, pp. 37–50 (2006). https://doi.org/10.1007/11776178_3
2. Ben-Or, M., Dolev, D., Hoch, E.N.: Fast self-stabilizing Byzantine tolerant digital clock synchronization. In: Proceedings of 27th annual ACM symposium on principles of distributed computing (PODC 2008), pp. 385–394. ACM Press (2008). https://doi.org/10.1145/1400751.1400802
3. Berman, P., Garay, J.A., Perry, K.J.: Towards optimal distributed consensus. In: Proceedings of 30th annual symposium on foundations of computer science (FOCS 1989), pp. 410–415. IEEE (1989). https://doi.org/10.1109/SFCS.1989.63511
4. Bloem, R., Braud-Santoni, N., Jacobs, S.: Synthesis of self-stabilising and Byzantine-resilient distributed systems. In: Proceedings of 28th international conference on computer aided

verification (CAV 2016), Part I, pp. 157–176 (2016). https://doi.org/10.1007/978-3-319-41528-4_9
5. Burns, J.E., Lynch, N.A.: The Byzantine firing squad problem. Adv. Comput. Res. **4**, 147–161 (1987)
6. Coan, B., Dolev, D., Dwork, C., Stockmeyer, L.: The distributed firing squad problem. SIAM J. Comput. **18**(5), 990–1012 (1989). https://doi.org/10.1137/0218068
7. Coan, B.A., Dwork, C.: Simultaneity is harder than agreement. Inf. Comput. **91**(2), 205–231 (1991). https://doi.org/10.1016/0890-5401(91)90067-C
8. Daliot, A., Dolev, D.: Self-stabilizing Byzantine agreement. In: Proceedings of 25th annual ACM symposium on principles of distributed computing (PODC 2006), pp. 143–152. ACM (2006). https://doi.org/10.1145/1146381.1146405
9. Doerr, B., Goldberg, L.A., Minder, L., Sauerwald, T., Scheideler, C.: Stabilizing consensus with the power of two choices. In: Proceedings of 23rd ACM symposium on parallelism in algorithms and architectures (SPAA 2011), pp. 149–158. ACM (2011). https://doi.org/10.1007/978-3-642-15763-9_50
10. Dolev, D.: The Byzantine generals strike again. J. Algorithms **3**(1), 14–30 (1982)
11. Dolev, D., Függer, M., Lenzen, C., Schmid, U., Steininger, A.: Fault-tolerant distributed systems in hardware. Bull. EATCS (116) (2015). http://bulletin.eatcs.org/index.php/beatcs/issue/view/18
12. Dolev, D., Heljanko, K., Järvisalo, M., Korhonen, J.H., Lenzen, C., Rybicki, J., Suomela, J., Wieringa, S.: Synchronous counting and computational algorithm design. J. Comput. Syst. Sci. **82**(2), 310–332 (2016). https://doi.org/10.1016/j.jcss.2015.09.002
13. Dolev, D., Hoch, E.N.: On self-stabilizing synchronous actions despite Byzantine attacks. In: Proceedings of 21st international symposium on distributed computing (DISC 2007). Lecture Notes in Computer Science, vol. 4731, pp. 193–207. Springer (2007). https://doi.org/10.1007/978-3-540-75142-7_17
14. Dolev, D., Hoch, E.N., Moses, Y.: An optimal self-stabilizing firing squad. SIAM J. Comput. **41**(2), 415–435 (2012). https://doi.org/10.1137/090776512
15. Dolev, D., Reischuk, R.: Bounds on information exchange for Byzantine agreement. J. ACM **32**(1), 191–204 (1985). https://doi.org/10.1145/2455.214112
16. Dolev, S.: Self-Stabilization. The MIT Press, Cambridge (2000)
17. Dolev, S., Kat, R.I., Schiller, E.M.: When consensus meets self-stabilization. J. Comput. Syst. Sci. **76**(8), 884–900 (2010). https://doi.org/10.1016/j.jcss.2010.05.005
18. Dolev, S., Welch, J.L.: Self-stabilizing clock synchronization in the presence of Byzantine faults. J. ACM **51**(5), 780–799 (2004). https://doi.org/10.1145/1017460.1017463
19. Dwork, C., Moses, Y.: Knowledge and common knowledge in a Byzantine environment: crash failures. Inf. Comput. **88**(2), 156–186 (1990). https://doi.org/10.1016/0890-5401(90)90014-9
20. Feldman, P., Micali, S.: An optimal probabilistic algorithm for synchronous Byzantine agreement. In: Proceedings of 16th international colloquium on automata, languages and programming (ICALP 1989). Lecture Notes in Computer Science, vol. 372, pp. 341–378. Springer (1989). https://doi.org/10.1007/BFb0035770
21. Fischer, M.J., Lynch, N.A.: A lower bound for the time to assure interactive consistency. Inf. Process. Lett. **14**(4), 183–186 (1982). https://doi.org/10.1016/0020-0190(82)90033-3
22. Hoch, E.N., Dolev, D., Daliot, A.: Self-stabilizing Byzantine digital clock synchronization. In: Proceedings of 8th international symposium on stabilization, safety, and security of distributed systems (SSS 2006). Lecture Notes in Computer Science, vol. 4280, pp. 350–362. Springer (2006). https://doi.org/10.1007/978-3-540-49823-0_25
23. King, V., Saia, J.: Breaking the $O(n^2)$ bit barrier. J. ACM **58**(4), 1–24 (2011). https://doi.org/10.1145/1989727.1989732
24. Lamport, L., Shostak, R., Pease, M.: The Byzantine generals problem. ACM Trans. Program. Lang. Syst. **4**(3), 382–401 (1982). https://doi.org/10.1145/357172.357176
25. Lenzen, C., Függer, M., Hofstätter, M., Schmid, U.: Efficient construction of global time in SoCs despite arbitrary faults. In: Proceedings of 16th euromicro conference on digital system design (DSD 2013), pp. 142–151 (2013). https://doi.org/10.1109/DSD.2013.97
26. Lenzen, C., Rybicki, J.: Near-optimal self-stabilising counting and firing squads (2017). arXiv:1508.02535
27. Lenzen, C., Rybicki, J., Suomela, J.: Efficient counting with optimal resilience. SIAM J. Comput. **64**(4), 1473–1500 (2017). https://doi.org/10.1137/16M107877X
28. Lynch, N.A.: Distrib. Algorithms. Morgan Kaufmann Publishers, San Francisco (1996)
29. Nishitani, Y., Honda, N.: The firing squad synchronization problem for graphs. Theor. Comput. Sci. **14**(1), 39–61 (1981). https://doi.org/10.1016/0304-3975(81)90004-9
30. Pease, M.C., Shostak, R.E., Lamport, L.: Reaching agreement in the presence of faults. J. ACM **27**(2), 228–234 (1980). https://doi.org/10.1145/322186.322188
31. Raynal, M.: Fault-Tolerant Agreement in Synchronous Message-Passing Systems. Morgan & Claypool, San Rafael (2010). https://doi.org/10.2200/S00294ED1V01Y201009DCT003