HAL
open science

Efficient counting with optimal resilience

Christoph Lenzen, Joel Rybicki

To cite this version:

Christoph Lenzen, Joel Rybicki. Efficient counting with optimal resilience. DISC 2015, Toshimitsu Masuzawa; Koichi Wada, Oct 2015, Tokyo, Japan. 10.1007/978-3-662-48653-5_2 . hal-01199803

HAL Id: hal-01199803

https://hal.science/hal-01199803v1

Submitted on 16 Sep 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Efficient counting with optimal resilience

Christoph Lenzen[1] and Joel Rybicki[1,2]

[1] Max Planck Institute for Informatics
[2] Helsinki Institute for Information Technology HIIT,
Department of Computer Science, Aalto University

Abstract. In the synchronous c-counting problem, we are given a synchronous system of n nodes, where up to f of the nodes may be Byzantine, that is, have arbitrary faulty behaviour. The task is to have all of the correct nodes count modulo c in unison in a self-stabilising manner: regardless of the initial state of the system and the faulty nodes' behavior, eventually rounds are consistently labelled by a counter modulo c at all correct nodes.
We provide a deterministic solution with resilience $f < n/3$ that stabilises in $O(f)$ rounds and every correct node broadcasts $O(\log^2 f)$ bits per round. We build and improve on a recent result offering stabilisation time $O(f)$ and communication complexity $O(\log^2 f/\log\log f)$ but with sub-optimal resilience $f = n^{1-o(1)}$ (PODC 2015). Our new algorithm has optimal resilience, asymptotically optimal stabilisation time, and low communication complexity.
Finally, we modify the algorithm to guarantee that after stabilisation very little communication occurs. In particular, for optimal resilience and polynomial counter size $c = n^{O(1)}$, the algorithm broadcasts only $O(1)$ bits per node every $\Theta(n)$ rounds without affecting the other properties of the algorithm; communication-wise this is asymptotically optimal.

1 Introduction

In this work, we seek to minimize the amount of communication required for fast self-stabilising, Byzantine fault-tolerant solutions to the *synchronous counting problem*. We are given a complete communication network on n nodes with arbitrary initial states. There are up to f faulty nodes that may behave in an arbitrary manner. The task is to synchronise the correct nodes so that they will count rounds modulo c in agreement. For example, the following is a possible execution for $n = 4$ nodes, $f = 1$ faulty node, and counting modulo $c = 4$; the execution stabilises after $T = 4$ rounds:

	Stabilisation				Counting			
Node 1	3	1	1	3	2	3	0	1
Node 2 (faulty)	*	*	*	*	*	*	*	*
Node 3	0	2	0	0	2	3	0	1
Node 4	2	0	2	2	2	3	0	1

In the severe fault-model considered in this work, synchronous counting is an important service for establishing the classic synchronous abstraction: even if a common clock signal is available, local counters may become inconsistent due to transient faults; these in turn induce arbitrary states, which is addressed by the self-stabilisation paradigm. Many, if not most, synchronous algorithms require synchronous round counters to operate correctly.

Synchronous counting is a coordination primitive that can be used e.g. in large integrated circuits to synchronise subsystems to easily implement *mutual exclusion* and *time division multiple access* in a fault-tolerant manner. Note that in this context, it is natural to assume that a synchronous clock signal is available, but the clocking system usually does not provide explicit round numbers. Solving synchronous counting thus yields highly dependable round counters for subcircuits.

If we neglect communication, counting and consensus are essentially equivalent [4,5,3]. In particular, many lower bounds on (binary) consensus directly apply to the counting problem [6,13,9]. However, the known generic reduction of counting to consensus incurs a factor-f overhead in space and message size. In recent work [12], we presented an approach that reduces the number of bits nodes broadcast in each round to $O(\log^2 f/\log\log f + \log c)$ at the expense of reduced resilience of $f = n^{1-o(1)}$. In this paper, we improve on the technique to achieve optimal resilience with $O(\log^2 f + \log c)$ bits broadcast by each node per round.

1.1 Contributions

In this work, we take the following approach. In order to devise *communication-efficient* algorithms, we first design *space-efficient* algorithms, that is, algorithms in which each node stores only a few bits between consecutive rounds. This comes with additional advantages:

- Local computations will (typically) be simple.
- Communication becomes simple, as one can afford to broadcast the entire state.
- This reduces the complexity of implementations.
- In turn, it becomes easier to use reliable components for an implementation, increasing the overall reliability of the system.

The key challenge that needs to be overcome in constructing space-efficient (and fast) solutions to counting appears to be a chicken-and-egg problem: given that the correct nodes agree on a counter, they can jointly run a (single) instance of synchronous consensus; given that they can run consensus, they can agree on a counter. In [12], this obstacle is navigated by making the statement more precise: given that the correct nodes agree on a counter *for a while*, they can run consensus. This is used to facilitate agreement on the output counter, in a way which maintains agreement even if the unreliable counters used for stabilisation fail later on.

The task of constructing counters that "work" only once in a while is easier; in particular, it does not require to solve consensus in the process. The drawback of the recursive solution in [12] is that, in order to be time-efficient, it sacrifices resilience. Our main contribution is to provide an improved construction that preserves optimal resilience.

Theorem 1. *For any integers $c, n > 1$ and $f < n/3$, there exists an f-resilient synchronous c-counter that runs on n nodes, stabilises in $O(f)$ rounds, and requires $O(\log^2 f + \log c)$ bits to encode the state of a node.*

The main hurdle that needs to be taken in order to arrive at this result when building on the techniques of [12] is the following. In both approaches, the nodes are partitioned into blocks, each of which runs a counter of smaller resilience; the construction proceeds inductively on increasing values of f, so such a counter exists by the induction hypothesis. In [12], it is assumed that a majority of these blocks contains sufficiently few faulty nodes for the counter to be operational, causing the relative resilience to deteriorate with each level of recursion in the construction. To achieve optimal resilience, we must drop this assumption, in turn necessitating novel ideas on how to establish a joint counter that is once in a while counting correctly at *all* non-faulty nodes. We show how to obtain such a counter based on simple local consistency checks, timeouts, and threshold voting.

Last but not least, we show how to reduce the number of bits broadcast after stabilisation to $\log c / \log \kappa + O(1)$ per node and κ rounds for an essentially unconstrained choice of κ, at the expense of additively increasing the stabilisation time by $O(\kappa)$. In particular, for the special case of optimal resilience and polynomial counter size, we obtain the following result.

Corollary 1. *For any $n > 1$ and $c = n^{O(1)}$ that is an integer multiple of n, there exists a synchronous c-counter that runs on n nodes, has optimal resilience $f = \lfloor (n-1)/3 \rfloor$, stabilises in $O(n)$ rounds, requires $O(\log^2 n)$ bits to encode the state of a node, and for which after stabilisation correct nodes broadcast (asymptotically optimal) $O(1)$ bits every $\Theta(n)$ rounds.*

1.2 Prior work

In terms of lower bounds, several impossibility results for consensus directly yield bounds for the counting problem as well [3]: counting cannot be solved in the presence of at least $n/3$ Byzantine failures [13] and any deterministic algorithm needs to run for at least f rounds [9] and communicate $\Omega(nf)$ bits to stabilise [6].

In contrast, there exist several algorithms to the synchronous counting problem, albeit these solutions exhibit different trade-offs in terms of resilience, stabilisation time, space and/or communication complexity, or whether a source of random bits is required. For a brief summary, see Table 1.

Designing space-efficient *randomised* algorithms for synchronous counting is fairly straightforward [7,8,3]: for example, the nodes can simply choose random states until a clear majority of nodes has the same state, after which they start to follow the majority. Likewise, given a shared coin, one can quickly reach

resilience	stabilisation time	state bits	deterministic	ref.
$f < n/3$ (*)	$O(1)$	$n^{O(1)}$ (*)	no	[1]
$f < n/3$	$O(f)$	$O(f \log f)$	yes	[4]
$f < n/3$	$2^{2(n-f)}$	2	no	[7,8]
$f < n/3$	$\min\{2^{2f+2}+1, 2^{O(f^2/n)}\}$	1	no	[3]
$f = 1$, $n \geq 4$	7	2	yes	[3]
$f = n^{1-o(1)}$	$O(f)$	$O(\log^2 f / \log\log f)$	yes	[12]
$f < n/3$	$O(f)$	$O(\log^2 f)$	yes	**here**

Table 1. Summary of counting algorithms for the case $c = 2$. For randomised algorithms, we list the expected stabilisation time. The solution from [10] relies on a shared coin. "(*)" indicates that details vary, but all known shared coins with large resilience require large states and messages.

agreement by defaulting to the coin whenever no clear majority is observed [1]; alas, existing shared coins are highly inefficient in terms of communication. Designing quickly stabilising algorithms that are both communication- and space-efficient has turned out to be a challenging task [4,5,3], and it remains open to what extent randomisation can help in designing such algorithms.

In the case of deterministic algorithms, algorithm synthesis has been used for computer-aided design of optimal algorithms with resilience $f = 1$, but the approach does not scale due to the extremely fast-growing space of possible algorithms [3]. In general, many fast-stabilising algorithms build on a connection between Byzantine consensus and synchronous counting, but require a large number of states per node [4] due to, e.g., running a large number of consensus instances in parallel. In [12], the approach outlined earlier was leveraged to ensure that each node participates in only $O(\log f / \log\log f)$ instances of consensus, resulting in small state and communication complexity, but reducing resilience to $f = n^{1-o(1)}$.

As a side note, the recursive construction presented in this work bears similarity to the recursive variant of the phase king algorithm [2], for which the goal of the recursion was also to control the communication complexity (reducing it from $\Theta(n^3)$ to $\Theta(n^2)$ for optimal resilience). In retrospect, the structural similarity is striking; one may think of our algorithm as a generalization of the approach to the case where there is no initial agreement on round numbers. The initial lack of consistent round labels is what causes a roughly factor n larger communication complexity in our case, which then can be removed after stabilisation leveraging consistent counters.

1.3 Structure of the article

In the next section, we provide formal descriptions of the model and the problem, and introduce some notation. In Section 3, we prove the main technical result on optimal resilience boosting and infer Theorem 1. In Section 4, we describe how to

reduce the amount of bits communicated after stabilisation. Finally, in Section 5, we discuss how randomisation can help in further reducing the communication complexity and conclude the paper.

2 Preliminaries

In this section, we define the model of computation and the counting problem.

Model of computation. We consider a fully-connected synchronous message-passing network. That is, our distributed system consists of a network of n nodes, where each node is a state machine and has communication links to all other nodes in the network. All nodes have a unique identifier from the set $[n] = \{0, 1, \ldots, n-1\}$. The computation proceeds in synchronous communication rounds. In each round, all processors perform the following in a lock-step fashion: (1) *broadcast* their current state to all nodes, (2) *receive* messages from all nodes, and (3) *update* their local state. We assume that the initial state of each node is arbitrary and there are up to f Byzantine nodes. A Byzantine node may have arbitrary behaviour, that is, it can deviate from the protocol in any manner. In particular, the Byzantine nodes can collude together in an adversarial manner and a single Byzantine node can send *different* messages to different correct nodes.

Algorithms and executions. Formally, we define an algorithm as a tuple $\mathbf{A} = \langle X, g, p \rangle$, where X is the set of all states any node can have, $g\colon [n] \times X^n \to X$ is the *state transition function*, and $p\colon [n] \times X \to [c]$ is the *output function*. That is, at each round when node v receives a vector $\mathbf{x} = \langle x_0, \ldots, x_{n-1} \rangle$ of messages, node v updates it state to $g(v, \mathbf{x})$ and outputs $p(v, x_v)$. As we consider c-counting algorithms, the set of output values is the set set $[c]$ of counter values. Note that the tuples passed to g are ordered according to the node identifiers, i.e., nodes can identify the sender of a message (this is frequently referred to as source authentication).

For any set of $\mathcal{F} \subseteq [n]$ of faulty nodes, we define a projection $\pi_{\mathcal{F}}$ that maps any state vector $\mathbf{x} \in X^n$ to a *configuration* $\pi_F(\mathbf{x}) = \mathbf{e}$, where $e_v = *$ if $v \in \mathcal{F}$ and $e_v = x_v$ otherwise. That is, the values given by Byzantine nodes are ignored and a configuration consists of only the states of correct nodes. A configuration $\mathbf{d}$ is *reachable* from configuration $\mathbf{e}$ if for every correct node $v \notin \mathcal{F}$ there exists some $\mathbf{x} \in X^n$ satisfying $\pi_{\mathcal{F}}(\mathbf{x}) = \mathbf{e}$ and $g(v, \mathbf{x}) = d_v$. Essentially, this means that when the system is in configuration $\mathbf{e}$, the Byzantine nodes can send node v messages so that it decides to switch to state d_v. An *execution* of an algorithm $\mathbf{A}$ is an infinite sequence of configurations $\xi = \langle \mathbf{e}_0, \mathbf{e}_1 \ldots, \rangle$ where configuration $\mathbf{e}_{r+1}$ is reachable from configuration $\mathbf{e}_r$.

Synchronous counters. We say that an execution $\xi = \langle \mathbf{e}_0, \mathbf{e}_1 \ldots, \rangle$ of algorithm $\mathbf{A}$ stabilises in time T if there is some $x \in [c]$ such that for every correct node $v \notin \mathcal{F}$ it holds that

$$p(v, e_{T+r,v}) = r - x \bmod c \text{ for all } r \geq 0,$$

where $e_{T+r,v}$ is the state of node v on round $T + r$.

An algorithm $\mathbf{A}$ is said to be a *synchronous c-counter* with *resilience* f that stabilises in time T, if for every $\mathcal{F} \subseteq [n]$, $|\mathcal{F}| \leq f$, all executions of algorithm $\mathbf{A}$ stabilise within T rounds. In this case, we say that the *stabilisation time* $T(\mathbf{A})$ of $\mathbf{A}$ is the minimal such T that all executions of $\mathbf{A}$ stabilise in T rounds. The *state complexity* of $\mathbf{A}$ is $S(\mathbf{A}) = \lceil \log |X| \rceil$, that is, the number of bits required to encode the state of a node between subsequent rounds. For brevity, we will often refer to $\mathcal{A}(n, f, c)$ as the family of synchronous c-counters over n nodes with resilience f. For example, $\mathbf{A} \in \mathcal{A}(4, 1, 2)$ denotes a synchronous 2-counter over 4 nodes tolerating one failure.

3 Optimal resilience boosting

In this section, we show how to use existing synchronous counters to construct new counters in larger networks with higher resilience. The construction is similar in spirit to the one given in [12], but somewhat simpler and allows for optimal resilience boosting. We first state the boosting theorem together with a general overview of the approach, then provide our novel construction, and subsequently discuss how to stabilise the output counters using the unreliable "helper" counters. Finally, we prove the main result.

3.1 The road map

The high-level idea of the resilience boosting method is as follows. We first start with counters that have a low resilience and use these to construct a new "weaker" counter that has a higher resilience but only needs to behave correctly *once in a while* for sufficiently long. Once such a weak counter exists, it can be used to provide consistent round numbers for long enough to execute a *single* instance of a high-resilience consensus protocol. This can be used to reach agreement on the output counter. Once we can boost resilience in the above manner, we can recursively apply this approach to get the desired resilience.

We now focus on a single recursion step of the resilience boosting. As in [12], the basic idea is to use multiple counters that run in parallel to perform a leader election process that is guaranteed to consider each of the counters as leader eventually. Eventually, a stabilised and correctly behaving counter is elected as a leader for some time and can be used to clock the consensus protocol.

The approach in [12] is inefficient in the sense that using many parallel counters scales poorly in terms of how fast the process operates, which in turn results in large stabilisation times. On the other hand, using only a small number of parallel counters yields poor resilience. Here, we introduce an approach that can—and in fact, must—operate with two counters only, resulting in optimal resilience and fast stabilisation. The key idea is that by running only two counters in parallel, we can utilise all the nodes for filtering out "bad counter values" for both counters and have the nodes carefully choose which counter to follow (and for how long).

In each application of the resilience boosting, each of the two counters is run by roughly half of the nodes. For $f = 0$, these counters are trivial: all nodes simply reproduce a local counter of a designated leader node. For $f > 0$, we assume that reliable counters for all $f' < f$ already exist, and combine an f_0-resilient and an f_1-resilient counter with $f_0, f_1 < f$ so that $f_0 + f_1 + 1 = f$. This implies that, no matter which nodes are faulty, one of the two counters will eventually stabilise.

Our first goal is to construct a τ-counter that counts correctly only once in a while; τ will roughly be the running time of the consensus protocol we will execute later on. In order to do this, we take two counting algorithms $\mathbf{A}_i$, $i \in \{0, 1\}$ with different counter ranges. We will have these two counters alternatively point to a "leader counter" for $\tau = \Theta(f)$ rounds, simply by dividing the counters by τ, rounding down, and taking the result modulo 2. However, to ensure that each $\mathbf{A}_i$ is eventually considered the leader for τ rounds by *both* counters, we let the pointer generated by $\mathbf{A}_1$ switch between leaders by factor 2 slower than the one of $\mathbf{A}_0$.

Obviously, employing this approach naively is not good enough: since $f > \max\{f_1, f_2\}$, it may happen that either $\mathbf{A}_0$ or $\mathbf{A}_1$ never stabilises. However, we are satisfied if nodes *behave* as if following an operational counter for τ rounds. To this end, we apply for each node v executing $\mathbf{A}_i$ the trivial consistency check whether the local output variable of $\mathbf{A}_i$ increases by 1 in each round. If not, it will switch to using $\mathbf{A}_{1-i}$ as reference for a sufficient number, in this case $\Theta(\tau)$, of rounds to ensure that both v and the nodes executing $\mathbf{A}_{1-i}$ will consider $\mathbf{A}_{1-i}$ as the leader for sufficiently long.

This almost cuts it—except that two nodes $w \neq v$ executing $\mathbf{A}_i$ may have a different opinion on the output variable for $\mathbf{A}_i$, as there are more than f_i faulty nodes executing $\mathbf{A}_i$. This final hurdle is passed by enlisting the help of *all* nodes for a majority vote on what the current output of $\mathbf{A}_i$ actually is. Essentially, here we use threshold voting, which in each round r at each node yields either a globally unique counter value $c_i(r)$ for $\mathbf{A}_i$ or $\perp$, indicating that $\mathbf{A}_i$ is not operating correctly. This entails that, eventually,

- There are unique values $c_i(r)$ that increase by 1 in each round and are considered to be the current counter value of $\mathbf{A}_i$ by all nodes executing $\mathbf{A}_i$ that are not currently relying on the counter of $\mathbf{A}_{1-i}$.
- If a node executing $\mathbf{A}_i$ defaults to the counter of $\mathbf{A}_{1-i}$, there are fewer than f_{1-i} faulty nodes executing $\mathbf{A}_{1-i}$.
- Hence, all correct nodes consider $\mathbf{A}_i$ with fewer than f_i faults for τ rounds as the leader.

We leverage this last property to execute the *phase king algorithm* [2] in the same way as in [12] to stabilise the output counters.

We remark that the stabilisation time on each level is the maximum of that for the used counters plus $O(f)$; by choosing $f_1 \approx f_2 \approx f/2$, we can thus ensure an overall stabilisation time of $O(f)$, irrespectively of the number of recursion levels. Formally, we prove the following theorem:

Theorem 2. *Let $c, n > 1$ and $f < n/3$. Define $n_0 = \lfloor n/2 \rfloor$, $n_1 = \lceil n/2 \rceil$, $f_0 = \lfloor (f-1)/2 \rfloor$, $f_1 = \lceil (f-1)/2 \rceil$, and $\tau = 3(f+2)$. If for $i \in \{0,1\}$ there exist synchronous counters $\mathbf{A}_i \in \mathcal{A}(n_i, f_i, c_i)$ such that $c_i = 3^i \cdot 2\tau$, then there exists a synchronous c-counter $\mathbf{B} \in (n, f, c)$ such that*

- $T(\mathbf{B}) = \max\{T(\mathbf{A}_0), T(\mathbf{A}_1)\} + O(f)$, *and*
- $S(\mathbf{B}) = \max\{S(\mathbf{A}_0), S(\mathbf{A}_1)\} + O(\log f + \log c)$.

We fix the notation of this theorem for the remainder of this section, as it is dedicated to its proof. Moreover, for notational convenience we abbreviate $T = \max\{T(\mathbf{A}_0), T(\mathbf{A}_1)\}$ and $S = \max\{S(\mathbf{A}_0), S(\mathbf{A}_1)\}$.

3.2 Agreeing on a common counter (once in a while)

In this part, we construct a counter that will eventually count consistently at all nodes for τ rounds. The τ-counter then will be used as a common clock for executing the phase king algorithm.

First, we partition $V = V_0 \cup V_1$ such that $V_0 \cap V_1 = \emptyset$, $|V_0| = n_0$ and $|V_1| = n_1$. We often refer to the set V_i as *block* i. For both $i \in \{0,1\}$, the nodes in set V_i execute the algorithm $\mathbf{A}_i$. In case block i has more than f_i faults, we call the block i faulty. Otherwise, we say that block i is correct. By construction, at least one of the blocks is correct. Hence, there is a correct block i for which $\mathbf{A}_i$ stabilises within T rounds, i.e., nodes in block i output a consistent c_i-counter in rounds $r \geq T$.

Lemma 1. *For some $i \in \{0,1\}$, block i is correct.*

Proof. By choice of f_i, we have $f = f_0 + f_1 + 1$. Hence, at least one of the sets V_i will contain at most f_i faults.

Next, we apply the typical threshold voting mechanism employed by most Byzantine tolerant algorithms in order to filter out differing views of counter values that are believed to be consistent. This is achieved by broadcasting candidate counter values and applying a threshold of $n - f$ as a consistency check, which guarantees that only one candidate value (besides the fallback value $\perp$ indicating an inconsistency) can remain. This is applied for each block concurrently, and all nodes participate in the process, so we can be certain that fewer than one third of the voters are faulty.

In addition to passing this voting step, we require that the counters also have behaved consistently over a sufficient number of rounds; this is verified by the obvious mechanism of testing whether the counter increases by 1 each round and counting the number of rounds since the last inconsistency was detected.

In the following, nodes frequently examine a set of values, one broadcast by each node, and determine majority values. Note that Byzantine nodes may send different values to different nodes, that is it may happen that correct nodes output different values from such a vote. We refer to a *strong majority* as at least $n - f$ nodes supporting the same value, which is then called the *majority*

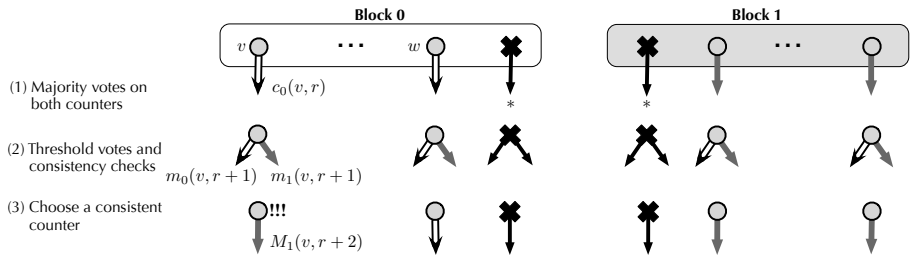

Fig. 1. Forming an opinion. The white block depicts nodes in the set V_0 running the c_0-counter, and the gray block the set V_1 running the c_1-counter. The white and gray filled arrows indicate the messages output by the white or gray block, respectively. The crosses denote Byzantine nodes with arbitrary output. In the above scenario, the white block is faulty and node v observes that the c_0-counter behaves inconsistently, hence it chooses to use the majority output of block 1; node w in the same block still relies on the c_0-counter, as it appears consistent from the perspective of node w.

value. If a node does not see a strong majority, it outputs the symbol $\perp$ instead. Clearly, this procedure is well-defined for $f < n/2$.

We will refer to this procedure as a *majority vote*, and slightly abuse notation by saying "majority vote" when, precisely, we should talk of "the output of the majority vote at node v". Since we require that $f < n/3$, the following standard argument shows that for each vote, there is a unique value such that each node either outputs this value or $\perp$.

Lemma 2. *If $v, w \in V \setminus \mathcal{F}$ both observe a strong majority, they output the same majority value.*

Proof. Fix any set A of $n - f$ correct nodes. As correct nodes broadcast the same value to each node, v and w observing strong majorities for different values would require that for each value A contains $n - 2f$ supporting it. However, this is impossible since $2(n - 2f) = n - f + (n - 3f) > n - f = |A|$.

We now put this principle to use. We introduce the following local variables for each node $v \in V$, block $i \in \{0, 1\}$, and round r:

- $m_i(v, r)$ stores the most frequent counter value in block i in round r, which is determined from the broadcasted output variables of $\mathbf{A}_i$ with ties broken arbitrarily,
- $M_i(v, r)$ stores the majority vote on $m_i(v, r - 1)$,
- $w_i(v, r)$ is a cooldown counter which is reset to $2c_1$ whenever the node perceives "the" counter of block i behaving inconsistently, that is, $M_i(v, r) \neq M_i(v, r-1)+1 \bmod c_i$. Note that this test will automatically fail if either value is $\perp$. Otherwise, if the counter behaves consistently, $w_i(v, r) = \max\{w_i(v, r - 1) - 1, 0\}$.

Figure 1 illustrates how the values of the m_i and M_i are determined. Clearly, these variables can be updated based on the local values from the previous round

and the states broadcasted at the beginning of the current round. This requires nodes to store $O(\log c_i) = O(\log f)$ bits.

Furthermore, we define the following derived variables for each $v \in V$, $i \in \{0,1\}$, and round r:

- $d_i(v,r) = M_i(v,r)$ if $w_i(v,r) = 0$, otherwise $d_i(v,r) = \bot$,
- $\ell_i(v,r) = \lfloor d_i(v,r)/(3^i\tau) \rfloor$ if $d_i(v,r) \neq \bot$, otherwise $\ell_i(v,r) = \bot$,
- for $v \in V_i$, $\ell(v,r) = \ell_i(v,r)$ if $\ell_i(v,r) \neq \bot$, otherwise $\ell(v,r) = \ell_{1-i}(v,r)$, and
- $d(v,r) = d_{\ell(v,r)}(v,r) \bmod \tau$ if $\ell(v,r) \neq \bot$, otherwise $d(v,r) = 0$.

These can be computed locally, without storing or communicating additional values. The variable $\ell(v,r)$ indicates the block that node v currently considers leader.

We now verify that $\ell_i(v,r)$ has the desired properties. To this end, we analyse $d_i(v,r)$. We start with a lemma showing that eventually a correct block's counter will be consistently observed by all correct nodes.

Lemma 3. *Suppose block $i \in \{0,1\}$ is correct. Then for all $v, w \in V \setminus \mathcal{F}$, and rounds $r \geq R = T + O(f)$ it holds that $d_i(v,r) = d_i(w,r)$ and $d_i(v,r) = d_i(v,r-1) + 1 \bmod c_i$.*

Proof. Within $T(\mathbf{A}_i)$ rounds, $\mathbf{A}_i$ stabilises. Moreover, any Byzantine tolerant counter must satisfy that $f_i < n_i/3$, implying that $m_i(v,r+1) = m_i(v,r) + 1 \bmod c_i$ for all $r \geq T(\mathbf{A}_i)$. Consequently, $M_i(v,r+1) = M_i(v,r) + 1 \bmod c_i$ for all $r \geq T(\mathbf{A}_i) + 1$. Therefore, $w_i(v,r)$ cannot be reset in rounds $r \geq T(\mathbf{A}_i) + 2$, yielding that $w_i(v,r) = 0$ for all $r \geq T(\mathbf{A}_i) + 2 + 2c_1 = T + O(f)$. The claim follows from the definition of variable $d_i(v,r)$.

The following lemma states that if a correct node v does not detect an error in a block's counter, then this means that any other correct node considering the block's counter correct *in any of the last $2c_1$ rounds* computed a counter value for that block consistent with the one of v.

Lemma 4. *Suppose for $i \in \{0,1\}$, $v \in V \setminus \mathcal{F}$, and $r \geq 2c_1 = O(f)$ it holds that $d_i(v,r) \neq \bot$. Then for each $w \in V \setminus \mathcal{F}$ and each $r' \in \{r - 2c_1 + 1, \ldots, r\}$ either $d_i(w,r') = d_i(v,r) - (r - r') \bmod c_i$ or $d_i(w,r') = \bot$.*

Proof. Suppose $d_i(w,r') \neq \bot$. Thus, $d_i(w,r') = M_i(w,r') \neq \bot$. By Lemma 2, either $M_i(v,r') = \bot$ or $M_i(v,r') = M_i(w,r')$. However, $M_i(v,r') = \bot$ would imply that $w_i(v,r') = 2c_1$ and thus

$$w_i(v,r) \geq w_i(v,r') + r - r' = 2c_1 + r - r' > 0,$$

contradicting the assumption that $d_i(v,r) \neq \bot$. Thus, $M_i(v,r') = M_i(w,r') = d_i(w,r')$. More generally, we get from $r - r' < 2c_1$ and $w_i(v,r) = 0$ that $w_i(v,r'') \neq 2c_1$ for all $r'' \in \{r', \ldots, r\}$. Therefore, we have that $M_i(v,r''+1) = M_i(v,r'') + 1 \bmod c$ for all $r'' \in \{r', \ldots, r-1\}$, implying

$$d_i(v,r) = M_i(v,r) = M_i(v,r') + r - r' = d_i(w,r') + r - r',$$

proving the claim of the lemma.

The above properties allow us to prove a key lemma: within $T + O(f)$ rounds, there will be τ consecutive rounds during which the variable $\ell_i(v, r)$ points to the same correct block for all correct nodes.

Lemma 5. *Let R be as in Lemma 3. There is a round $r \leq R + O(f) = T + O(f)$ and a correct block i so that for all $v \in V \setminus \mathcal{F}$ and $r' \in \{r, \ldots, r + \tau - 1\}$ it holds that $\ell(v, r') = i$.*

Proof. By Lemma 1, there exists a correct block i. Thus by Lemma 3, variable $d_i(v, r)$ counts correctly during rounds $r \geq R$. If there is no round $r \in \{R, \ldots, R + c_i - 1\}$ such that some $v \in V \setminus \mathcal{F}$ has $\ell_{1-i}(v, r) \neq \bot$, then $\ell(v, r) = \ell_i(v, r)$ for all such v and r and the claim of the lemma holds true by the definition of $\ell_i(v, r)$ and the fact that $d_i(v, r)$ counts correctly and consistently.

Hence, assume that $r_0 \in \{R, \ldots, R + c_i - 1\}$ is minimal with the property that there is some $v \in V \setminus \mathcal{F}$ so that $\ell_{1-i}(v, r_0) \neq \bot$. Therefore, $d_{1-i}(v, r_0) \neq \bot$ and, by Lemma 4, this implies for all $w \in V \setminus \mathcal{F}$ and all $r \in \{r_0, \ldots, r_0 + 2c_1 - 1\}$ that either $d_{1-i}(w, r) = \bot$ or $d_{1-i}(w, r) = d_{1-i}(v, r_0) + r - r_0$. In other words, there is a "virtual counter" that equals $d_{1-i}(v, r_0)$ in round r_0 so that during $\{r_0, \ldots, r_0 + 2c_1 - 1\}$ correct nodes' d_{1-i} variable either equals this counter or $\bot$.

Consequently, it remains to show that both ℓ_i and the variable ℓ_{1-i} derived from this virtual counter equal i for τ consecutive rounds during the interval $\{r_0, \ldots, r_0 + 2c_1 - 1\}$, as then $\ell(v, r) = i$ for $v \in V \setminus \mathcal{F}$ and such a round r. Clearly, the c_1-counter consecutively counts from 0 to $c_1 - 1$ at least once during rounds $\{r_0, \ldots, r_0 + 2c_1 - 1\}$. Recalling that $c_1 = 6\tau$, we see that $\ell_1(v, r) = i$ for all $v \in V \setminus \mathcal{F}$ with $\ell_1(v, r) \neq \bot$ for 3τ consecutive rounds during $\{r_0, \ldots, r_0 + 2c_1 - 1\}$. As $c_0 = 2\tau$, we have that $\ell_0(v, r) = i$ for all $v \in V \setminus \mathcal{F}$ with $\ell_0(v, r) \neq \bot$ for τ consecutive rounds during this subinterval. As argued earlier, $\ell_0(v, r) \neq \bot$ or $\ell_1(v, r) \neq \bot$ and hence $\ell(v, r) = i$ for each such node and round. Because $r_0 + 2c_1 - 1 < R + 3c_1 = T + O(f)$, this completes the proof.

Using the above lemma, we get a counter where all nodes eventually count correctly and consistently modulo τ for at least τ rounds.

Corollary 2. *There is a round $r = T + O(f)$ so that (1) for all $v, w \in V \setminus \mathcal{F}$ it holds that $d(v, r) = d(w, r)$ and (2) for all $v \in V \setminus \mathcal{F}$ and $r' \in \{r + 1, \ldots, r + \tau - 1\}$ we have $d(v, r') = d(v, r' - 1) + 1 \bmod \tau$.*

Proof. By Lemma 5, there is a round $r = T + O(f)$ and a correct block i such that for all $v \in V \setminus \mathcal{F}$ we have $\ell(v, r') = i$ for all $r' \in \{r, \ldots, r + \tau - 1\}$. Moreover, r is sufficiently large to apply Lemma 3 to $d_i(v, r') = d(v, r')$ for $r' \in \{r + 1, \ldots, r + \tau - 1\}$, yielding the claim.

3.3 Reaching consensus

For every node $v \in V$, let $a(v, r)$ denote the output variable of the synchronous c-counting algorithm $\mathbf{B}$ we are constructing. Similarly as in a prior work [12], we now apply the phase king consensus algorithm [2] to get all nodes in the network agree on the output value of the c-counter. The phase king algorithm has the following properties:

- the algorithm tolerates $f < n/3$ Byzantine failures,
- the running time of the algorithm is $O(f)$ rounds and it uses $O(\log c)$ bits of state,
- if node v is correct, then agreement is reached if all correct nodes execute rounds $3v$, $3v+1$, and $3v+2$ consecutively,
- once agreement is reached, then agreement persists even when nodes execute *different* rounds.

More formally, we have the following lemma:

Lemma 6 (adapted from [12]). *Let $v \in [f+2]$ be a correct node and $r \geq 0$.*
- *If all correct nodes execute the instructions $3v+k$ of the phase king algorithm during round $r+k$ for all $k \in \{0,1,2\}$, then for any $r' > r+2$, we have $a(u,r') = a(w,r')$ and $a(u,r'+1) = a(u,r') + 1 \bmod c$ for all $u,w \in V \setminus \mathcal{F}$.*
- *If $a(u,r') = a(w,r')$ for all $u,w \in V \setminus \mathcal{F}$, then $a(u,r'+1) = a(w,r'+1) = a(w,r') + 1 \bmod c$ no matter which (even if different) instructions nodes u and w execute on round r'.*

3.4 Proofs of Theorems 1 and 2

We are now ready to prove our main results of this section.

Theorem 2. *Let $c,n > 1$ and $f < n/3$. Define $n_0 = \lfloor n/2 \rfloor$, $n_1 = \lceil n/2 \rceil$, $f_0 = \lfloor (f-1)/2 \rfloor$, $f_1 = \lceil (f-1)/2 \rceil$, and $\tau = 3(f+2)$. If for $i \in \{0,1\}$ there exist synchronous counters $\mathbf{A}_i \in \mathcal{A}(n_i, f_i, c_i)$ such that $c_i = 3^i \cdot 2\tau$, then there exists a synchronous c-counter $\mathbf{B} \in (n,f,c)$ such that*
- *$T(\mathbf{B}) = \max\{T(\mathbf{A}_0), T(\mathbf{A}_1)\} + O(f)$, and*
- *$S(\mathbf{B}) = \max\{S(\mathbf{A}_0), S(\mathbf{A}_1)\} + O(\log f + \log c)$.*

Proof. First, we apply the construction underlying Corollary 2. Then we have every node $v \in V$ in each round r execute the instructions for round $d(v,r)$ of the phase king algorithm discussed in the previous paragraph. It remains to show that this yields a correct algorithm $\mathbf{B}$ with stabilisation time $T(\mathbf{B}) = T + O(f)$ and space complexity $S(\mathbf{B}) = S + O(\log f + \log c)$, where $T = \max\{T(\mathbf{A}_i)\}$ and $S = \max\{S(\mathbf{A}_i)\}$.

By Corollary 2, there exists a round $r = T + O(f)$ so that the variables $d(v,r)$ behave as a consistent τ-counter during rounds $\{r, \ldots, r+\tau-1\}$ for all $v \in V \setminus \mathcal{F}$. As there are at most f faulty nodes, there exist at least two correct nodes $v \in [f+2]$. Since $\tau = 3(f+2)$, for at least one correct node $v \in [f+2] \setminus \mathcal{F}$, there is a round $r \leq r_v \leq r + \tau - 3$ such that $d(w, r_v + k) = 3v + k$ for all $w \in V \setminus \mathcal{F}$ and $k \in \{0,1,2\}$. By Lemma 6, it follows that the output variables $a(w,r')$ count correctly and consistently for all $r' \geq r_v + 3$ and $w \in V \setminus \mathcal{F}$. Thus, the algorithm stabilises in $r_v + 3 \leq r + \tau = r + O(f) = T + O(f)$ rounds.

The bound for the space complexity follows from the facts that, at each node, we need (a) at most S bits to store the state of $\mathbf{A}_i$, (b) $O(\log \tau) = O(\log f)$ bits to store the auxilary variables underlying Corollary 2, (c) $O(\log \tau) = O(\log f)$ bits for the helper variables underlying Lemma 6 [12], and (d) $\lceil \log c \rceil$ bits to store the output variable $a(v,r)$.

Theorem 1. *For any integers $c, n > 1$ and $f < n/3$, there exists an f-resilient synchronous c-counter that runs on n nodes, stabilises in $O(f)$ rounds, and requires $O(\log^2 f + \log c)$ bits to encode the state of a node.*

Proof. We show the claim by induction on f. The induction hypothesis is that for all $f > f' \geq 0$, $c > 1$, and $n > 3f'$, we can construct $\mathbf{B} \in \mathcal{A}(f', n, c)$ with

$$T(\mathbf{B}) = 1 + \alpha f' \sum_{k=0}^{\lceil \log f' \rceil} (1/2)^k \qquad \text{and} \qquad S(\mathbf{B}) = \beta(\log^2 f' + \log c),$$

where α and β are sufficienlty large constants and for $f' = 0$ the sum is empty, that is, $T(\mathbf{B}) = 1$. As $\sum_{k=0}^{\infty}(1/2)^k = 2$, this will prove the theorem. Note that for $f \geq 0$ it is sufficient to show the claim for $n(f) = 3f + 1$, as we can easily generalise to any $n > n(f)$ by running $\mathbf{B}$ on the first $n(f)$ nodes and letting the remaining nodes follow the majority counter value among the $n(f)$ nodes executing the algorithm; this increases the stabilisation time by one round and induces no memory overhead.

For the base case, observe that a 0-tolerant c-counter of $n(0) = 1$ node is trivially given by the node having a local counter. It stabilises in 0 rounds and requires $\lceil \log c \rceil$ state bits. As pointed out above, this implies a 0-tolerant c-counter for any n with stabilisation time 1 and $\lceil \log c \rceil$ bits of state.

For the inductive step to f, we apply Theorem 2. For $i \in \{0, 1\}$, we have that $f_i \leq f/2$, $n_i > 3f_i$, and $c_i = O(f)$. This implies by the induction hypothesis that there are $\mathbf{A}_i(n_i, f_i, c_i)$ with

$$T(\mathbf{A}_i) = 1 + \frac{\alpha f}{2} \sum_{k=0}^{\lceil \log f/2 \rceil} \left(\frac{1}{2}\right)^k + O(f) = 1 + \alpha f \sum_{k=0}^{\lceil \log f \rceil} \left(\frac{1}{2}\right)^k,$$

where in the last step we use that α is sufficiently large, and

$$S(\mathbf{B}) = \beta \left(\log^2 \frac{f}{2} + \log \frac{f}{2} \right) + O(\log f + \log c) = \beta \left(\log^2 f + \log c \right),$$

where we exploit that β is sufficiently large. Hence, the induction step succeeds.

4 Less communication after stabilisation

We now sketch how to reduce the number of bits broadcast by a node after stabilisation; see [11] for the complete construction. The techniques we use are very similar to the ones we used for deriving Theorem 1. Essentially, we devise a "silencing wrapper" for algorithms given by Theorem 1. Let $\mathbf{A}$ be such a counting algorithm. The high-level idea and the key ingredients are the following:

- The goal is that nodes eventually become *happy*: they assume stabilisation has occured and check for counter consistency only every κ rounds (as self-stabilising algorithms always need to verify their output).

- Happy nodes do not execute the underlying algorithm **A** to avoid the involved communication. This necessitates a fall-back stabilisation mechanism covering the case that a subset of the correct nodes is happy, but does not detect a problem.
- Using a cooldown counter with similar effects as shown in Lemma 4, we enforce that all happy nodes output consistent counters.
- We override the phase king instruction of **A** if at least $n - 2f \geq f + 1$ nodes (claim to be) happy and propose a counter value x. Instead nodes adjust their counter output accordingly to match x. If there is no strong majority of happy nodes a supporting counter value, either all nodes become unhappy or all correct nodes reach agreement and start counting correctly.
- If all correct nodes are unhappy, they execute **A** "as is" reaching agreement eventually.
- The agreed-upon counters are used to make all nodes concurrently switch their state to being happy (once the cooldown counters have expired), in a way that does not interfere with the above stabilisation process.

The final observation is that happy nodes can communicate their counter values very efficiently in a manner that self-stabilises within κ rounds. As their counter increases by 1 modulo c in every round (or they become unhappy), they can use κ rounds to encode a counter value; the recipient simply counts locally in the meantime.

5 Discussion

We presented a deterministic counting algorithm that has low state and communication complexity, optimal resilience, and asymptotically optimal stabilisation time. In addition, we gave a variant of the algorithm that communicates extremely little once stabilisation is achieved. In [12], we consider the so-called pulling model, in which nodes *request* messages from others instead of broadcasting a message to everyone, and use randomisation to reduce the amount of bits communicated (in contrast to broadcasting) by each correct node to $\log^{O(1)} n$ per round. We remark that this approach can also applied to the solution given in this work.

From our point of view, the most thrilling open question is whether similar ideas can be applied to randomised consensus routines in order to achieve sublinear stabilisation time with high resilience and small communication overhead. Another point of note is that this general type of recursion, which we essentially extended from its use for synchronous consensus [2] (where the clock is implicitly given by the synchronous start), might also prove useful for deriving improved pulse synchronisation [4] algorithms. Interestingly, no reduction from consensus to pulse synchronisation is known, so there is hope for efficient deterministic algorithms that stabilise in sublinear time.

Acknowledgements. We thank anonymous reviewers for helpful feedback and Jukka Suomela for discussions and comments.

References

1. Ben-Or, M., Dolev, D., Hoch, E.N.: Fast self-stabilizing Byzantine tolerant digital clock synchronization. In: Proc. 27th Annual ACM Symposium on Principles of Distributed Computing (PODC 2008). pp. 385–394. ACM Press (2008), doi:10.1145/1400751.1400802.
2. Berman, P., Garay, J.A., Perry, K.J.: Towards optimal distributed consensus. In: Proc. 30th Annual Symposium on Foundations of Computer Science (FOCS 1989). pp. 410–415. IEEE (1989), doi:10.1109/SFCS.1989.63511.
3. Dolev, D., Heljanko, K., Järvisalo, M., Korhonen, J.H., Lenzen, C., Rybicki, J., Suomela, J., Wieringa, S.: Synchronous counting and computational algorithm design (2015), arXiv:1304.5719v2.
4. Dolev, D., Hoch, E.N.: On self-stabilizing synchronous actions despite Byzantine attacks. In: Proc. 21st International Symposium on Distributed Computing (DISC 2007). Lecture Notes in Computer Science, vol. 4731, pp. 193–207. Springer (2007), doi:10.1007/978-3-540-75142-7_17.
5. Dolev, D., Korhonen, J.H., Lenzen, C., Rybicki, J., Suomela, J.: Synchronous counting and computational algorithm design. In: Proc. 15th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS 2013). Lecture Notes in Computer Science, vol. 8255, pp. 237–250. Springer (2013), doi:10.1007/978-3-319-03089-0_17.
6. Dolev, D., Reischuk, R.: Bounds on information exchange for Byzantine agreement. Journal of the ACM 32(1), 191–204 (1985), doi:10.1145/2455.214112.
7. Dolev, S.: Self-Stabilization. The MIT Press, Cambridge, MA (2000)
8. Dolev, S., Welch, J.L.: Self-stabilizing clock synchronization in the presence of Byzantine faults. Journal of the ACM 51(5), 780–799 (2004), doi:10.1145/1017460.1017463.
9. Fischer, M.J., Lynch, N.A.: A lower bound for the time to assure interactive consistency. Information Processing Letters 14(4), 183–186 (1982), doi:10.1016/0020-0190(82)90033-3.
10. Hoch, E., Dolev, D., Daliot, A.: Self-stabilizing Byzantine digital clock synchronization. In: Proc. 8th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS 2006). vol. 4280, pp. 350–362 (2006)
11. Lenzen, C., Rybicki, J.: Efficient counting with optimal resilience (2015), arXiv:1508.02535.
12. Lenzen, C., Rybicki, J., Suomela, J.: Towards optimal synchronous counting. In: Proc. 34th Annual ACM Symposium on Principles of Distributed Computing (PODC 2015). pp. 441–450. ACM Press (2015), doi:10.1145/2767386.2767423.
13. Pease, M.C., Shostak, R.E., Lamport, L.: Reaching agreement in the presence of faults. Journal of the ACM 27(2), 228–234 (1980), doi:10.1145/322186.322188.