Towards Optimal Synchronous Counting

Christoph Lenzen
clenzen@mpi-inf.mpg.de
Max Planck Institute for Informatics

Joel Rybicki
joel.rybicki@aalto.fi
Max Planck Institute for Informatics
Helsinki Institute for Information Technology HIIT, Dept. of Computer Science, Aalto University

Jukka Suomela
jukka.suomela@aalto.fi
Helsinki Institute for Information Technology HIIT, Dept. of Computer Science, Aalto University

ABSTRACT

Consider a complete communication network of n nodes, in which the nodes receive a common clock pulse. We study the synchronous c-counting problem: given any starting state and up to f faulty nodes with arbitrary behaviour, the task is to eventually have all correct nodes count modulo c in agreement. Thus, we are considering algorithms that are self-stabilising despite Byzantine failures. In this work, we give new algorithms for the synchronous counting problem that (1) are deterministic, (2) have linear stabilisation time in f, (3) use a small number of states, and (4) achieve almost-optimal resilience. Prior algorithms either resort to randomisation, use a large number of states, or have poor resilience. In particular, we achieve an *exponential* improvement in the state complexity of deterministic algorithms, while still achieving linear stabilisation time and almost-linear resilience.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems

Keywords

synchronous counting; self-stabilisation; Byzantine fault-tolerance

1. INTRODUCTION

In this work, we design space-efficient, self-stabilising, Byzantine fault-tolerant algorithms for 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. The task is to synchronise the nodes so that all non-faulty nodes will count rounds modulo c in agreement. For example, here is a possible execution for $n = 4$ nodes, $f = 1$ faulty node, and counting modulo $c = 3$; the execution stabilises after $t = 5$ rounds:

	Stabilisation					Counting						
Node 1:	2	2	0	2	0	0	1	2	0	1	2	. . .
Node 2:	0	2	0	1	0	0	1	2	0	1	2	. . .
Node 3:	*faulty node, arbitrary behaviour*											. . .
Node 4:	0	0	2	0	2	0	1	2	0	1	2	. . .

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

Counting modulo $c = 2$ is closely related to *binary consensus*: given a synchronous counting algorithm one can design a binary consensus algorithm and vice versa [3, 4, 5]—in particular, many lower bounds on binary consensus apply here as well [6, 9, 10]. However, the existing implementations of counting from consensus incur a factor-f overhead in space and message size, rendering them very costly even in small systems.

1.1 Prior Work

It is fairly easy to design space-efficient *randomised* algorithms for synchronous counting [3, 7, 8]: as a simple example, the nodes can just pick random states until a clear majority of them has the same state, after which they start to follow the majority. However, this approach yields algorithms with exponential stabilisation time. Given a shared coin, it is possible to obtain fast randomised algorithms by defaulting to the value of the shared coin whenever no clear majority is observed [1]; unfortunately, existing shared coins are highly inefficient in terms of space and communication.

In comparison, it is even more challenging to come up with space-efficient *deterministic* algorithms for synchronous counting [3, 4, 5], and it remains open to what extent randomisation helps in designing space-efficient algorithms that stabilise quickly. Fast-stabilising deterministic algorithms build on a connection to Byzantine consensus, but require a large number of states per node [3, 4].

For small values of the parameters (e.g., $n = 4$ and $f = 1$) the synchronous counting problem is amenable to *algorithm*

PODC'15, July 21–23, 2015, Donostia-San Sebastián, Spain.

ACM 978-1-4503-3617-8 /15/07 ...$15.00.
DOI: http://dx.doi.org/10.1145/2767386.2767423.

resilience	stabilisation time	state bits	deterministic	references
$f < n/3$ (*)	$O(1)$	$n^{O(1)}$ (*)	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]
$f = 1$, $n \geq 6$	6	1	yes	[3]
$f = 1$, $n \geq 6$	3	2	yes	[3]
$f = n^{1-o(1)}$	$O(f)$	$O(\log^2 f)$	yes	**this work**

Table 1: Summary of synchronous 2-counting algorithms. "(*)" indicates that details vary, but all known shared coins with large resilience require large states and messages. For randomised algorithms, we list the expected stabilisation time. In [3], further trade-offs between n, the stabilisation time, and the number of state bits are given (for $f = 1$).

synthesis: it is possible to use computers to automatically design both space-efficient and time-efficient deterministic algorithms for synchronous counting. For example, there is a computer-designed algorithm that solves the case of $n \geq 4$ and $f = 1$ with only 3 states per node, and another algorithm that solves the case of $n \geq 6$ and $f = 1$ with only 2 states per node; both of these are optimal [3, 5]. Unfortunately, this approach does not scale, as the space of possible algorithms for given parameters n and f grows rapidly with n.

In summary, currently no algorithms for synchronous counting are known that simultaneously scale well in terms of resilience f, stabilisation time t, and the state that needs to be preserved between rounds. Here, it is worth noting that existing deterministic solutions with small state in essence run $\Omega(f)$ concurrent instances of consensus, which implies that the respective overhead extends to message size and the amount of local computations as well.

1.2 Contributions

Our main contribution is a recursive construction that shows how to "amplify" the resilience of a synchronous counting algorithm. Given a synchronous counter for some values of n and f, we will show how to design synchronous counters for larger values of n and f, with a very small increase in time and state complexity. This has two direct applications:

1. From a practical perspective, we can apply existing computer-designed algorithms (e.g. $n = 4$ and $f = 1$) as a building block in order to design efficient deterministic algorithms for a moderate number of nodes (e.g., $n = 36$ and $f = 7$).
2. From a theoretical perspective, we can now design deterministic algorithms for synchronous counting for any n and for $f = n^{1-o(1)}$ faulty nodes, with a stabilisation time of $O(f)$, and with only $O(\log^2 f / \log\log f)$ bits of state per node.

The state complexity is an *exponential* improvement over prior work, and the stabilisation time is asymptotically optimal for deterministic algorithms [9]. A summary of the related work and our contributions is given in Table 1.

In our deterministic algorithms, each node only needs to store a few number of bits between consecutive rounds, and thus, a node can afford to broadcast its entire state to all other nodes in each round. However, the small number of state bits bears the promise that the communication load can be reduced further. To substantiate the conjecture that finding algorithms with small state complexity may lead to highly communication-efficient solutions, we proceed to consider a slightly stronger synchronous *pulling model.* In this model, a node may send a request to another node and receive a response in a single round, based on the state of the responding node at the beginning of the round. The cost for the exchange is then attributed to the pulling node; in a circuit, this translates to each node being assigned an energy budget that it uses to "pay" for the communication it triggers. In this model, it is straightforward to combine our recursive construction with random sampling to obtain the following results:

1. We can achieve the properties of the deterministic algorithm with each node pulling $\operatorname{polylog} n$ messages in each round. The price is that the resulting algorithm retains a probability of $n^{-\operatorname{polylog} n}$ to fail in each round even after stabilisation.
2. If the failing nodes are chosen independently of the algorithm, we can fix the random choices. This results in a pseudo-random algorithm which stabilises with a probability of $1 - n^{-\operatorname{polylog} n}$ and in this case keeps counting correctly.

1.3 Structure

This paper is organised as follows. In Section 2 we formally define the model of computing and the synchronous counting problem. Section 3 gives the main technical result—a construction for creating a synchronous counting algorithm of larger resilience from several copies of an algorithm with smaller resilience. Section 4 uses this construction to derive deterministic synchronous counting algorithms with linear stabilisation time and polylogarithmic state complexity. Finally, in Section 5, we discuss the pulling model and how randomised sampling can be used to reduce the total number of communicated bits.

2. PRELIMINARIES

We consider a fully-connected distributed message-passing system consisting of n processors, also called nodes, with unique identifiers from the set $[n] = \{0, 1, \ldots, n-1\}$. The computation proceeds in synchronous communication rounds, where in each round each processor: (1) *broadcasts* its local state to all processors, (2) *receives* a vector of messages (that is, states) from all other processors, and (3) *updates* its local state according to the received messages. The initial state of every node is arbitrary. Moreover, up to f nodes may be *Byzantine*, i.e., exhibit arbitrary behaviour, including to send *different* messages to every node. Thus, different nodes may receive different vectors depending on what the Byzantine nodes do.

2.1 Algorithms and Executions

A deterministic algorithm in this model is a tuple $\mathbf{A} = (X, g, h)$, where X is the set of all possible states for a node, $g\colon [n] \times X^n \to X$ is the state transition function, and $h\colon [n] \times X \to [c]$ maps the internal state of a node to an output value. That is, when node $i \in [n]$ receives a vector $\mathbf{x} \in X^n$ of messages, it will update its internal state to $g(i, \mathbf{x}) = s$ and output $h(i, s) \in [c]$. The *state complexity* $S(\mathbf{A})$ of an algorithm $\mathbf{A}$ is the total number of bits required to store the state of a node between consecutive rounds. That is, $S(\mathbf{A}) = \lceil \log |X| \rceil$.

For any given set $\mathcal{F} \subseteq [n]$ of faulty nodes, we define a projection $\pi_{\mathcal{F}}$ as follows: for any received message vector $\mathbf{x} \in X^n$, let $\pi_{\mathcal{F}}(\mathbf{x})$ be a vector $\mathbf{e}$ such that $e_i = *$ if $i \in \mathcal{F}$ and $e_i = x_i$ otherwise. We call $\pi_{\mathcal{F}}(X^n) = \{\pi_{\mathcal{F}}(\mathbf{x}) : \mathbf{x} \in X^n\}$ the set of *configurations*. That is, a configuration consists only of the state of all non-faulty nodes.

We say that a configuration $\mathbf{d}$ is *reachable* from configuration $\mathbf{e}$ if for every non-faulty node $i \notin \mathcal{F}$ there exists some $\mathbf{x} \in X^n$ satisfying $\pi_{\mathcal{F}}(\mathbf{x}) = \mathbf{d}$ and $g(i, \mathbf{x}) = d_i$. This means that when the system is in configuration $\mathbf{e}$, the Byzantine nodes can send node i such messages that i chooses to switch to state d_i. Given the set of faulty nodes, an *execution* of an algorithm $\mathbf{A}$ is an infinite sequence of configurations $\xi = \langle \mathbf{e}_0, \mathbf{e}_1, \mathbf{e}_2, \dots \rangle$ such that $\mathbf{e}_{t+1}$ is reachable from $\mathbf{e}_t$.

2.2 Synchronous Counters

We say that an execution ξ of algorithm $\mathbf{A} = (X, g, h)$ *stabilises* in time t if there is some $r_0 \geq 0$ so that for every non-faulty node $i \in [n]$ the output satisfies

$$h(i, x_{t+r,i}) = r - r_0 \bmod c \text{ for all } r \geq 0.$$

That is, within t steps all non-faulty nodes agree on a common output and start incrementing their counters modulo c each round. Moreover, we say that $\mathbf{A}$ is a *synchronous c-counter* with *resilience* f if there exists a t such that for every $\mathcal{F} \subseteq [n]$, $|\mathcal{F}| \leq f$, all executions of $\mathbf{A}$ stabilise in t rounds. We say that the *stabilisation time* $T(\mathbf{A})$ of algorithm $\mathbf{A}$ is the minimal such t. In what follows, we use $\mathcal{A}(n, f, c)$ to denote the family of synchronous c-counters of resilience f that run on n nodes.

3. BOOSTING RESILIENCE

In this section, we show that given a family of synchronous c-counters for a small number of nodes n and resilience f, we can construct a new family of C-counters of a larger resilience without increasing the number of nodes in the network or the stabilisation time by too much. More precisely, we prove the following main result.

THEOREM 1. *Given $n, f \in \mathbb{N}$, pick $N, F, C \in \mathbb{N}$, where*

- *the number of nodes is $N = kn$ for some number of blocks $3 \leq k \in \mathbb{N}$,*
- *the resilience is $F < (f+1)m$, where $m = \lceil k/2 \rceil$, and*
- *the counter size is $C > 1$.*

Choose any $c \in \mathbb{N}$ that is a multiple of $3(F+2)(2m)^k$. Then for any $\mathbf{A} \in \mathcal{A}(n, f, c)$ there exists a $\mathbf{B} \in \mathcal{A}(N, F, C)$ with

$$T(\mathbf{B}) \leq T(\mathbf{A}) + 3(F+2)(2m)^k,$$
$$S(\mathbf{B}) = S(\mathbf{A}) + \lceil \log(C+1) \rceil + 1.$$

3.1 High-level Idea of the Construction

Given a suitable counter $\mathbf{A} \in \mathcal{A}(n, f, c)$, we construct a larger network with $N = kn$ nodes and divide the nodes in k blocks of n nodes. Each block runs a copy of $\mathbf{A}$ that is used to (1) determine a “leader block” and (2) output a consistent round counter *within* a block.

Due to the bound on F, only a minority of the blocks (fewer than m) contains more than f faulty nodes. Thus, the counters of a majority of the blocks stabilise within $T(\mathbf{A})$ rounds. By dividing the counters within each block, we let them “point” to one of m possible leader blocks for a fairly large number of consecutive rounds. Block i switches through leaders by a factor of $2m$ faster than block $i+1$. This ensures that, eventually, all stabilised counters point to the same leading block for $3(F+2)$ rounds.

Using a majority vote on the pointers to leader blocks, we can make sure that all correct nodes—also those in blocks with more than f faulty nodes—will recognise the same block as the leader for sufficiently long. In fact, eventually this will happen for each of the m blocks that may become leaders, one of which must have fewer than f faulty nodes. In particular, its counter will be stabilised and can be “read” by all other nodes based on taking the majority value within that block. Hence, we can guarantee that at some point all nodes agree on a common counter value for at least $3(F+2)$ rounds. This value is used to control an execution of the well-known *phase king protocol* [2], which solves consensus in $\Theta(F)$ rounds in systems of $N > 3F$ nodes. As the constraint $F < (f+1)m$ also ensures that $F < N/3$, we can use the phase king protocol to let all nodes agree on the current value of the C-counter that is to be computed. It is straightforward to guarantee that, once this agreement is achieved, it will not be lost again and the counter is incremented by one modulo C in each round.

3.2 Setup

Our goal is to construct a synchronous C-counter that runs in a network of $N = kn$ nodes and tolerates $F < (f+1)m$ faults. We divide the nodes into k blocks, each of size n. We identify each node $v \in [kn]$ with a tuple $(i, j) \in [k] \times [n]$. Thus, node $v = (i, j)$ is the jth node of block i. Blocks that contain more than f faulty nodes are said to be *faulty*. Otherwise, a block is *non-faulty*.

We will have each block i run a synchronous counter $\mathbf{A}_i$ constructed as follows. Define $\tau = 3(F+2)$ and $c_i = \tau(2m)^{i+1}$ for $i \in [k]$. Let $\mathbf{A} = (X, g, h) \in \mathcal{A}(n, f, c)$ be the given synchronous c-counter, where $c = \alpha\tau(2m)^k$ for some integer α. Now for any $i \in [k]$, we obtain a synchronous c_i-counter $\mathbf{A}_i = (X, g, h_i)$ by defining the output function as $h_i(x) = h(x) \bmod c_i$. That is, $\mathbf{A}_i$ outputs the output of $\mathbf{A}$ modulo c_i. It follows that $T(\mathbf{A}_i) = T(\mathbf{A})$ and $S(\mathbf{A}_i) = S(\mathbf{A})$. Thus, if block i is non-faulty, then $\mathbf{A}_i$ will stabilise in $T(\mathbf{A})$ rounds.

We interpret the value of the counter of (non-faulty, stabilised) block i as a tuple $(r, y) \in [\tau] \times [(2m)^{i+1}]$, where r is incremented by one modulo τ each round, and y is incremented by one whenever r “overflows” to 0. We refer to the counter value that node (i, j) currently has according to $\mathbf{A}_i$

as $(r[i,j], y[i,j])$. Note that this value can be directly inferred from its state, so by broadcasting its state (i,j) implicitly announces its current counter value to all other nodes. We stress that there is no guarantee whatsoever on how these variables behave in faulty blocks, even for non-faulty nodes in faulty blocks—the only guarantee is that non-faulty nodes in non-faulty blocks will count correctly after round $T(\mathbf{A})$.

We define the short-hand

$$b[i,j] = \left\lfloor \frac{y[i,j]}{(2m)^i} \right\rfloor \bmod m.$$

The value $b[i,j]$ indicates the block that the nodes in block i currently consider to be the "leader block" by interpreting the counter given by $\mathbf{A}_i$ appropriately.

Let $b_q[i,j]$ and $r_q[i,j]$ denote the values of $b[i,j]$ and $r[i,j]$ in round q. We will now show that after stabilisation, a non-faulty block i will within c_i rounds point to every block $\beta \in [m]$ for at least τ consecutive rounds. For notational convenience, we let $c_{-1} = \tau$.

Lemma 1. *Let $i \in [k]$ be a non-faulty block and $t \geq T(\mathbf{A})$. For any $\beta \in [m]$, there exists some $t \leq w \leq t + c_i - c_{i-1}$ such that if (i,j) is non-faulty, then $b_q[i,j] = \beta$ for $w \leq q < w + c_{i-1}$.*

Proof. By round t, the counter $\mathbf{A}_i$ has stabilised and all non-faulty nodes (i,j) agree. First observe that since $\mathbf{A}_i$ is a c_i-counter, $b[i,j]$ cycles through the set $[m]$ twice in c_i rounds. Moreover, once $b[i,j]$ changes its value, it will keep the new value for $c_{i-1} = \tau(2m)^i$ consecutive rounds.

More formally, let (i,j) be a non-faulty node and $u \geq t$ be the minimal u such that $r_u[i,j] = 0$ and $y_u[i,j] \in \{0, c_i/(2\tau)\}$. In particular, at round u we have $b_u[i,j] = 0$. Moreover, $u \leq t + c_i/2$. Since $b[i,j]$ retains the same value for c_{i-1} consecutive rounds, we get that at round $w = u + (\beta - 1)c_{i-1}$ the value $b[i,j]$ changes to β. Now we have that $b_{w'}[i,j] = \beta$ for $w \leq w' < w + c_{i-1}$. A simple check confirms that $w + c_{i-1} \leq t + c_i$. □

Using this lemma, we can show that after stabilisation, all non-faulty nodes in non-faulty blocks will within c_k rounds point to each $\beta \in [m]$ simultaneously for at least τ rounds.

Lemma 2. *Let $t \geq T(\mathbf{A})$ and $\beta \in [m]$. There exists some $t \leq u \leq t + c_k - \tau$ such that if (i,j) is a non-faulty node in a non-faulty block, then $b_q[i,j] = \beta$ for $u \leq q < u + \tau$.*

Proof. We prove the statement for the special case that all blocks are non-faulty. As there is no interaction between the algorithm $\mathbf{A}_i$ of different blocks, the lemma then follows by simply excluding all nodes in faulty blocks. We prove the following claim using induction; see Figure 1 for illustration.

Claim: For any round $t \geq T(\mathbf{A})$, block $\beta \in [m]$ and $h \in [k]$, there exists a round $t \leq u(h) \leq t + c_h - \tau$ such that for all non-faulty nodes (i,j) where $i \leq h$ we have $b_q[i,j] = \beta$ for $u(h) \leq q < u(h) + \tau$.

In particular, the lemma follows from the case $h = k$ of the above claim. For the base case of the induction, observe that case $h = 0$ follows from Lemma 1 as $c_{-1} = \tau$. For the inductive step, suppose the claim holds for some $h \in [k-1]$ and consider the non-faulty block $h+1$. By Lemma 1 there exists $t \leq w \leq t + c_{h+1} - c_h$ such that $b_q[h+1,j] = \beta$ for all $w \leq q < w + c_h$ where $(h+1,j)$ is a non-faulty node. Applying the induction hypothesis to w we get $u(h)$. Setting $u(h{+}1) = u(h)$ yields that $u(h{+}1) \leq w{+}c_h{-}\tau \leq t{+}c_{h+1}{-}\tau$. This proves the claim and the lemma follows. □

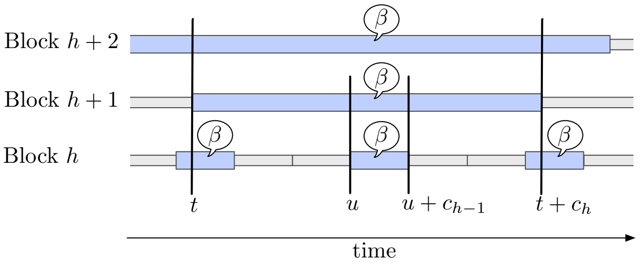

Figure 1: Counters in non-faulty blocks will eventually coincide. The picture illustrates the output of $b[\cdot]$ for three blocks $i \in \{h, h+1, h+2\}$ running $\tau(2m)^{i+1}$-counters with base $2m = 6$. For every $\beta \in [m]$, we can find an interval (highlighted segments) where all non-faulty blocks point to the same value β for sufficiently long, even though the counters may cycle at different time points.

3.3 Voting Blocks

Next, we define the voting scheme for all the blocks. Define the majority operation for every message vector $\mathbf{x} \in X^{kn}$ as follows:

$$\operatorname{majority} \mathbf{x} = \begin{cases} a & \text{if } a \text{ appears in } \mathbf{x} \text{ more than } kn/2 \text{ times,} \\ * & \text{otherwise,} \end{cases}$$

where the symbol $*$ indicates that the function may evaluate to an arbitrary value, including different values at different non-faulty nodes. We use the following short-hands as local variables:

$$\begin{aligned} b^i &= \operatorname{majority}\{b[i,j] \mid j \in [n]\}, \\ B &= \operatorname{majority}\{b^i \mid i \in [k]\}, \\ R &= \operatorname{majority}\{r[B,j] \mid j \in [n]\}. \end{aligned}$$

Note that these functions can be locally computed from the received state vectors by checking for a majority and defaulting to, e.g., 0, when no such majority is found: by definition, the majority function may return an arbitrary value if $kn/2$ or fewer correct nodes "vote" for the same value; as non-faulty nodes broadcast the same state to all nodes, there can only be one such majority value.

In words, b^i denotes the block which the nodes in block i support as a leader; different correct nodes may "observe" different values of b^i only if i is a faulty block or $\mathbf{A}_i$ has not yet stabilised. As $F < (f+1)m = (f+1)\lceil k/2 \rceil$, majority of the blocks are non-faulty. Hence, if all non-faulty blocks support the same leader block $\beta \in [m]$, then B evaluates to β at all correct nodes. By Lemma 2, this is bound to happen eventually. Finally, R denotes the round counter of block B, which is "read" correctly by all non-faulty nodes if B is non-faulty.

Analogously to before, let $i' \in [k]$ and denote by $B_q[i,j]$ and $R_q[i,j]$, respectively, the values to which the above functions evaluate in round q at node (i,j). Then we can conclude from Lemma 2 that eventually all non-faulty nodes agree on R for τ rounds.

Lemma 3. *There is a round $t \leq T(\mathbf{A}) + c_k - \tau$ such that:*

(a) $R_q[i,j] = R_q[i',j']$ for any $t \leq q < t + \tau$ and non-faulty nodes (i,j) and (i',j').

(b) $R_{q+1}[i,j] = R_q[i,j] + 1 \bmod \tau$ *for any* $t \le q < t + \tau - 1$ *and non-faulty node* (i,j).

PROOF. As $F < (f+1)m$, there is a non-faulty block $\beta \in [m]$. By applying Lemma 2 to round $T(\mathbf{A})$, there is a round $t \le T(\mathbf{A}) + c_k - \tau$ such that $b_q[i,j] = \beta$ for each $t \le q < t+\tau$ and non-faulty node (i,j) in a non-faulty block i. Therefore, $b_q^i = \beta$ for all non-faulty blocks i. As $F < (f+1)m$, the number of faulty blocks is at most $m-1 = \lceil k/2 \rceil - 1 < k/2$, and thus, the majority vote yields $B = \beta$.

Since $T(\mathbf{A}) \le t \le q < t + \tau$, we have that block β has stabilised by round t and therefore, $r_q[\beta, j] = r_q[\beta, j']$ for all non-faulty nodes (β, j), (β, j') in the non-faulty block β. Moreover, as block β is non-faulty, it contains at most f faulty nodes. In particular, it must be that $f < n/3 < n/2$ as otherwise counting cannot be solved and $\mathbf{A} \in \mathcal{A}(n, f, c)$. Hence, a majority vote yields $R = r_q[\beta, j]$, where (β, j) is any non-faulty node in block β proving claim (a).

To show (b), observe that since block β is non-faulty and has stabilised by round t we have that $R_q[i,j] = r_q[\beta, j']$. Moreover, non-faulty nodes in block β increment $r[\beta, j]$ by one modulo τ in the considered interval. □

3.4 Executing the Phase King

We have now built a voting scheme that allows the nodes to eventually agree on common counter for τ rounds. Roughly speaking, what remains is to use this common counter to control a non-self-stabilising F-resilient C-counting algorithm.

We require that this algorithm guarantees two properties. First, all non-faulty nodes reach agreement and start counting correctly within τ rounds provided that the underlying round counter is consistent. Second, if all non-faulty nodes agree on the output, then the agreement persists *regardless* of the round counter's value. It turns out that a straightforward adaptation of the classic phase king protocol [2] does the job.

From now on, we refer to nodes by their indices $v \in [N]$. The phase king protocol (like any consensus protocol) requires that $F < N/3$. It is easy to verify that this follows from the preconditions of Theorem 1.

Denote by $a[v] \in [C] \cup \{\infty\}$ the output register of the algorithm, where ∞ is used as a "reset state". There is also an auxiliary register $d[v] \in \{0, 1\}$. Define the following short-hand for the increment operation modulo C:

$$\text{increment}\, a[v] = \begin{cases} a[v] \leftarrow a[v] + 1 \bmod C & \text{if } a[v] \ne \infty, \\ \text{no action} & \text{if } a[v] = \infty. \end{cases}$$

For $\ell \in [F+2]$, we define the instruction sets listed in Table 2. First, we show that if these instructions are executed in the right order by all non-faulty nodes for a non-faulty leader $\ell \in [F+2]$, then agreement on a counter value is established.

LEMMA 4. *Suppose that for some non-faulty node* $\ell \in [F+2]$ *and a round* q*, all non-faulty nodes execute instruction sets* $I_{3\ell}$*,* $I_{3\ell+1}$*, and* $I_{3\ell+2}$ *in rounds* $q-2$*,* $q-1$*, and* q*, respectively. Then* $a_{q+1}[v] = a_{q+1}[u] \ne \infty$ *for any two non-faulty nodes* $u, v \in [N]$*. Moreover,* $d_{q+1}[v] = 1$ *at each non-faulty node.*

PROOF. This is essentially the correctness proof for the phase king algorithm. Without loss of generality, we can assume that the number of faulty nodes is exactly F. By assumption, we have $F < N/3$ and hence $2(N - 2F) > N - F$. It follows that it is not possible that two non-faulty nodes $v, u \in [N]$ satisfy both $a_{q-1}[v], a_{q-1}[u] \in [C]$ and $a_{q-1}[v] \ne a_{q-1}[u]$: this would imply that there are at least $N - 2F$ non-faulty nodes w that had $a_{q-2}[w] + 1 \bmod C = a_{q-1}[v]$ and the same number of non-faulty nodes w' with $a_{q-2}[w'] + 1 \bmod C = a_{q-1}[u]$; however, there are only $N - F < 2(N - 2F)$ non-faulty nodes. Therefore, there is some $x \in [C]$ so that $a_{q-1}[v] \in \{x, \infty\}$ for all non-faulty nodes v. Checking $I_{3\ell+2}$ and exploiting that $2(N-F) > N - F$ once more, we see that this also implies $a_q[v] \in \{x + 1 \bmod C, \infty\}$ for any non-faulty node v.

We need to consider two cases. In the first case, all non-faulty nodes execute the first instruction of $I_{3\ell+2}$ in round q. Then $a_{q+1}[v] = \min\{C, a_q[\ell]\} + 1 \bmod C$ for any non-faulty node v. In the second case, there is some node v not executing the first instruction of $I_{3\ell+2}$. Hence, $d_q[v] = 1$, implying that v computed $z_{a_{q-1}[v]} \ge N - F$ in round $q-1$. Consequently, at least $N - 2F > F$ non-faulty nodes u satisfy $a_{q-1}[u] = a_{q-1}[v]$. We infer that $a_q[v] = x'$ for all non-faulty nodes v: the third instruction of $I_{3\ell+1}$ must evaluate to $x' \in [C]$ at all non-faulty nodes. Clearly, this implies that $a_{q+1}[v] = a_{q+1}[u] \ne \infty$ for non-faulty nodes v, u, regardless of whether they execute the first instruction of $I_{3\ell+2}$ or not. Trivially, $d_{q+1}[v] = 1$ at each non-faulty node v due to the second instruction of $I_{3\ell+2}$. □

Next, we argue that once agreement is established, it persists—it does not matter any more which instruction sets are executed.

LEMMA 5. *Assume that* $a_q[v] = x \in [C]$ *and* $d_q[v] = 1$ *for all non-faulty nodes* v *in some round* q*. Then* $a_{q+1}[v] = x + 1 \bmod c$ *and* $d_{q+1}[v] = 1$ *for all non-faulty nodes* v*.*

PROOF. Each node will observe at least $N - F$ nodes with counter value x, and hence at most F nodes with some value $y \ne x$. For non-faulty node v, consider all possible instruction sets it may execute.

First, consider the case where instruction set $I_{3\ell}$ is executed. In this case, v increments x, resulting in $a_{q+1}[v] = x + 1 \bmod C$ and $d_{q+1}[v] = 1$. Second, executing $I_{3\ell+1}$, node v evaluates $z_x \ge N - F$ and $z_y \le F$ for all $y \ne x$. Hence it sets $d_{q+1}[v] = 1$ and $a_{q+1}[v] = x + 1 \bmod C$. Finally, when executing $I_{3\ell+2}$, node v skips the first instruction and sets $d_{q+1}[v] = 1$ and $a_{q+1}[v] = x + 1 \bmod C$. □

3.5 Proof of Theorem 1

We can now prove the main result. As shown in Lemma 3 we have constructed a τ-counter that will remain consistent at least τ rounds. This is a sufficiently long time for the nodes to execute the phase king protocol in synchrony. This protocol will stabilise the C-counter for the network of N nodes. More precisely, each node (i,j) runs the following algorithm:

1. Update the state of algorithm $\mathbf{A}_i$.
2. Compute the counter value R.
3. Update state according to instruction set I_R of the phase king protocol.

By Lemma 3, there is a round $t \le T(\mathbf{A}) + c_k - \tau$ so that the variables $R_q[i,j]$ meet the requirements of τ-counting for rounds $t \le q < t + \tau$. For each round q, all non-faulty nodes execute the same set of instructions. In particular, as $\tau = 3(F+2)$, no matter from which value the τ-counting starts, for at least $F + 1$ values $\ell \in [F+2]$ the instruction sets $I_{3\ell}$,

Set	Instructions
$I_{3\ell}$:	1. If fewer than $N-F$ nodes sent $a[v]$, set $a[v] \leftarrow \infty$. 2. increment $a[v]$.
$I_{3\ell+1}$:	1. Let $z_j = \lvert\{u \in [N] : a[u] = j\}\rvert$ be the number of j values received. 2. If $z_{a[v]} \geq N-F$, set $d[v] \leftarrow 1$. Otherwise, set $d[v] \leftarrow 0$. 3. Set $a[v] \leftarrow \min\{j : z_j > F\}$. 4. increment $a[v]$.
$I_{3\ell+2}$:	1. If $a[v] = \infty$ or $d[v] = 0$, then set $a[v] \leftarrow \min\{C, a[\ell]\}$. 2. Set $d[v] \leftarrow 1$ and increment $a[v]$.

Table 2: Phase king instruction sets for node $v \in [N]$.

$I_{3\ell+1}$, and $I_{3\ell+2}$, in this order, will be jointly executed by all non-faulty nodes at some point during rounds $t, \ldots, t+\tau-1$.

As there are only F faulty nodes, there are at least two non-faulty nodes $\ell \in [F+2]$. Thus, the prerequisites of Lemma 4 are satisfied in some round $q \leq t+\tau \leq T(\mathbf{A}) + c_k$. By an inductive application of Lemma 5, we conclude that the variables $a_q[v]$ are valid outputs for C-counting, and therefore, we have indeed constructed an algorithm $\mathbf{B} \in \mathcal{A}(N, F, C)$.

The bound on q yields that $T(\mathbf{B}) \leq T(\mathbf{A}) + c_k = T(\mathbf{A}) + 3(F+2)(2m)^k$. Concerning the state complexity, observe that each non-faulty node (i,j) needs the to store the state for executing (1) the algorithm $\mathbf{A}_i$, which needs $S(\mathbf{A}_i) = S(\mathbf{A})$ bits of state, and (2) the phase king protocol, which needs $\lceil \log(C+1) \rceil$ bits to store $a_q[v] \in [C] \cup \{\infty\}$ and one additional bit to store $d_q[v]$. $\square$

4. THE RECURSIVE CONSTRUCTION

In this section, we show how to use Theorem 1 recursively to construct synchronous c-counters with a near-optimal resilience, linear stabilisation time, and a small number of states (see Figure 2 for an illustration). First, we show how to satisfy the preconditions of Theorem 1 in order to start the recursion. Then we demonstrate the principle by choosing a fixed value of k throughout the construction; this achieves a resilience of $\Omega(n^{1-\varepsilon})$ for any constant $\varepsilon > 0$. However, as the number of nodes in the initial applications of Theorem 1 is small, better results are possible by starting out with large values of k and decreasing them later. This yields an algorithm with a resilience of $n^{1-o(1)}$ and $O(f)$ stabilisation time using $O(\log^2 f / \log\log f + \log c)$ state bits.

4.1 The Base Case

To apply Theorem 1, we need counters of resilience $f > 0$. For example, one can use the space-efficient 1-resilient counters from [3] as base of the construction. Alternatively, we can use as a starting point trivial counters for $n = 1$ and $f = 0$. Then we can apply the same construction as in Theorem 1 with the parameters $n = 1$, $f = 0$, $k = N$, and $F < N/3$. The same proof goes through in this case and yields the following corollary. Note that here the resilience is optimal but the algorithm is inefficient with respect to stabilisation time and state complexity.

Corollary 1. *For any $c > 1$, there exists a synchronous c-counter with optimal resilience $f < n/3$ that stabilises in $f^{O(f)}$ rounds and uses $O(f \log f + \log c)$ bits of state.*

Proof. For any $f > 0$, we can construct a f-resilient counter for $3f+1$ nodes. We use the trivial 0-resilient counter for one node as the base case for Theorem 1 and set $k = 3f+1$, that is, each block consists of a single node. Theorem 1 yields an f-resilient algorithm that stabilises in $(3f+2)(2m)^k = f^{O(f)}$ rounds, where the $O(f)$ term is $(3+o(1))f$. $\square$

4.2 Using a Fixed Number of Blocks

For the sake of simplicity, we will first discuss the recursive construction for a fixed value of k here. Improved resilience can be achieved by varying k depending on the level of recursion which we show afterwards.

Theorem 2. *Let $1 > \varepsilon > 0$ and $2 \leq c \in \mathbb{N}$. There exists a synchronous c-counting algorithm with a resilience of $f = \Omega(n^{1-\varepsilon})$ that stabilises in $O(2^{2^{1/\varepsilon}} f)$ rounds and uses $O(2^{1/\varepsilon} \log f + \log^2 f + \log c)$ bits of state per node.*

Proof. Fix $0 < \varepsilon < 1$ and let $k = 2h \geq 4$ be minimal such that $\varepsilon \geq 1/\log h$. Assume $f = 2^j$ for some $j \geq h$ and let $L = \log f / \log h$; w.l.o.g., assume that $L \in \mathbb{N}$. We will analyse how many nodes are required to get to a resilience of $\Omega(n^{1-\varepsilon})$ by applying Theorem 1 for $L \leq \varepsilon \log f$ iterations.

For all $i \geq 0$, let $f_i = h^i$ and $n_i = 4k^i$. At iteration $i+1$ we use Theorem 1 to construct algorithms in $\mathcal{A}(n_{i+1}, f_{i+1}, c)$ (for any c) using n_i and f_i as the input parameters. Since $f_{i+1} = f_i h < (f_i + 1)\lceil k/2 \rceil$ and $n_{i+1} = kn_i$, the conditions of Theorem 1 are satisfied. To start the recursion, we will use an algorithm with parameters $f_0 = 1$ and $n_0 = 4$. By Corollary 1, such an algorithm with a stabilisation time of $T_0 = O(1)$ and a state complexity of $S_0 = O(\log c)$ exists.

Every iteration increases the resilience by a factor of at least $h = k/2$. After L iterations, we tolerate at least $f = f_L = h^L$ failures using $n = n_L = 4k^L$ nodes. This gives

$$\frac{n}{f} = 4\left(\frac{k}{k/2}\right)^L = 4 \cdot 2^L \leq 8f^\varepsilon < 8n^\varepsilon.$$

and it follows that the resilience is $\Omega(n^{1-\varepsilon})$.

It remains to analyse the stabilisation time and state complexity of the resulting algorithm; both follow from Theorem 1. The stabilisation time of layer $i+1$ is

$$T_{i+1} \leq T_i + 3(f_{i+1} + 2)h^k,$$

where $h = \lceil k/2 \rceil$. From the definition of f_i, we get the bound $\sum_{i=0}^{L} f_i = O(f_L) = O(f)$. Hence, overall stabilisation takes

$$T = T_L \leq O\left(h^k \sum_{i=0}^{L} f_i\right) = O(h^k f)$$

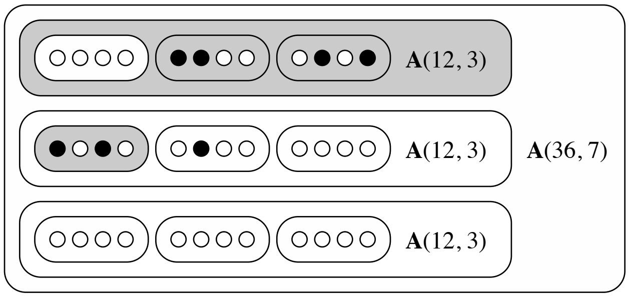

Figure 2: Recursive application of our construction using $k = 3$ blocks. The small circles represent the nodes. Each group of four nodes runs a 1-resilient counter $\mathbf{A}(4, 1)$. On top of this, each larger group of 12 nodes runs a 3-resilient counter $\mathbf{A}(12, 3)$. At the top-most layer, the nodes run a 7-resilient counter $\mathbf{A}(36, 7)$. Faulty nodes are black and faulty blocks are gray.

rounds. From Theorem 1, we get that the state complexity of layer $i+1$ is at most

$$S_i + \lceil \log(3(f_{i+2}+2)h^k + 1) \rceil + 1 = S_i + O(k \log h + \log f).$$

As $L \le \varepsilon \log f$, the total number of bits is then bounded by

$$\sum_{i=0}^{L} S_i + O(\log c) = O(\varepsilon k \log h \log f + \log^2 f + \log c).$$

Recall that $h = k/2$ is minimal such that $\varepsilon \ge 1/\log h$. Thus $k = O(2^{1/\varepsilon})$ and $\log h = O(1/\varepsilon)$, yielding the claimed bounds on time and state complexity. □

Choosing a constant ε, we arrive at the following corollary.

COROLLARY 2. *For any constant $1 > \varepsilon > 0$ and any $2 \le c \in \mathbb{N}$, there exists a synchronous c-counter with resilience $f = \Omega(n^{1-\varepsilon})$ that stabilises in $O(f)$ rounds and uses $O(\log^2 n + \log c)$ bits of state.*

4.3 Varying the Number of Blocks

Obviously, the factor $2^{2^{1/\varepsilon}}$ makes the previous construction impractical unless $1/\varepsilon$ is small. However, it turns out that we can still achieve good resilience without a doubly-exponential blow-up in the stabilisation time by carefully varying the number of blocks at each level.

THEOREM 3. *For any $c > 1$, there exist synchronous c-counters with a resilience of $f = n^{1-o(1)}$ that stabilises in $O(f)$ rounds and uses $O(\log^2 f / \log\log f + \log c)$ state bits per node.*

Proving this theorem boils down to choosing k in each iteration as large as possible without violating the bound on the stabilisation time. We again rely on Theorem 1, but instead of using a fixed number of blocks at each iteration, we divide the construction into P *phases*. During each phase, we use a different number of blocks and iterations of Theorem 1. The goal is to have the running time of the last phase dominate the running time of earlier phases.

PROOF. We set each phase $1 \le p \le P$ to use $k_p = 4 \cdot 2^{P-p} > 3$ blocks per layer and then iterate Theorem 1 exactly $R_p = 2k_p$ times. During phase p, we use iteration $1 \le i+1 \le R_p$ to get an algorithm from Theorem 1 that tolerates

$$f_{i+1,p} = \frac{f_{i,p} k_p}{2} < (f_{i,p} + 1) \left\lceil \frac{k_p}{2} \right\rceil$$

failures, where $f_{0,p} = f_{R_{p-1},p-1}$ and $f_{R_0,0} = f_{0,0} = 1$. Thus, for any p and i, the values k_p and $f_{i,p}$ satisfy the conditions of Theorem 1. Again, to start the recursion we may use any algorithm tolerating a single fault among 4 nodes giving $f_{0,0} = 1$.

Now, every phase p increases the resilience by a factor of

$$d_p = \left(\frac{f_{i+1,p}}{f_{i,p}} \right)^{R_p} = \left(\frac{k_p}{2} \right)^{R_p}.$$

As there are in total P phases, this means that the total resilience and number of nodes n are given by

$$f = f_P = \prod_{p=1}^{P} d_p = \prod_{p=1}^{P} \left(\frac{k_p}{2} \right)^{R_p} \text{ and } n = n_P = 4 \prod_{p=1}^{P} k_p^{R_p}.$$

In order to get resilience of $f \ge n^{1-\varepsilon}$, where $\varepsilon = o(1)$, we want to ensure that

$$\frac{n}{f} = 4 \prod_{p=1}^{P} 2^{R_p} = 4 \prod_{j=0}^{P-1} 2^{2 \cdot 2^j} = 2^{2(2^P - 1)} \le f^{\varepsilon},$$

which is equivalent to

$$2(2^P - 1) = 4 \cdot 2^{P-1} - 2 = k_1 - 2 \le \varepsilon \log f.$$

Hence, it is feasible to choose $\varepsilon = k_1 / \log f$. Observe that $\sum_{p=1}^{P} k_p = 4(2^P - 1) < 2k_1$. It follows that

$$\log f = \log \prod_{p=1}^{P} (k_p/2)^{R_p} = 2 \sum_{p=1}^{P} k_p \log(k_p/2) = \Theta(k_1 \log k_1),$$

implying that $k_1 = \Theta(\log f / \log\log f)$. We conclude that $\varepsilon = \Theta(1/\log\log f)$. Note that it is always possible to increase n by letting additional nodes simply adopt the majority counter value among those that execute the "actual" algorithm. Hence, for any sufficiently large n, we can construct an algorithm with resilience $f \ge n^{1-\varepsilon} = n^{1-o(1)}$. □

Let us now analyse the stabilisation time and state complexity of the construction.

LEMMA 6. *The algorithm stabilises in $O(f)$ rounds.*

PROOF. To analyse the stabilisation time, we first bound the stabilisation time of each phase p separately. By Theorem 1, iteration $i+1$ of phase p has stabilisation time

$$T_{i+1,p} = T_{i,p} + O\Big(f_{i+1,p} \cdot k_p^{k_p} \Big).$$

Analogously to the proof of Theorem 2, we again get a geometric series and the total stabilisation time of phase p is bounded by

$$\sum_{i=1}^{R_p} T_{i+1,p} = O(T_{R_p,p}) = O\Big(f_p \cdot k_p^{k_p} \Big).$$

Therefore, the total stabilisation time is

$$T = \sum_{p=1}^{P} T_p = O\left(\sum_{p=1}^{P} f_p \cdot k_p^{k_p} \right).$$

For $1 \leq p \leq P-4$, using the shorthand $\lambda = k_{p+1}$, we get

$$\frac{f_{p+1}}{f_p} \cdot \frac{\lambda^\lambda}{k_p^{k_p}} = d_{p+1} \frac{\lambda^\lambda}{(2\lambda)^{2\lambda}} = \frac{\lambda^{2\lambda}}{2^{2\lambda}} \cdot \frac{\lambda^\lambda}{(2\lambda)^{2\lambda}} = \left(\frac{\lambda}{16}\right)^\lambda \geq 2.$$

Thus, we get a geometric series $\sum_{p=1}^{P-3} T_p \leq 2T_{P-2} = O(f)$, and since also $T_{P-2} + T_{P-1} + T_P = O(f)$, the stabilisation time of all phases is bounded by $O(f)$. □

Lemma 7. *Every node uses at most $O(\log^2 f / \log\log f + \log c)$ bits to encode its state.*

Proof. By Theorem 1, the number of state bits increases each iteration by $\Theta(\log C_i)$, where C_i is the counter size needed for iteration i. During phase p, the counter size is $O(f \cdot k_p^{k_p})$ in each iteration. There are $R_p = 2k_p$ iterations phase p, and thus the number of bits we need is bounded by

$$S_p = O(k_p (\log f + k_p \log k_p)) = O\big(k_p \log f + k_p^2 \log k_p\big)\,.$$

From earlier computations, we know that $\sum_{i=1}^{P} k_p = O(k_1)$ and $k_1 = O(\varepsilon \log f)$. Thus, we use

$$\begin{aligned} S = \sum_{p=1}^{P} S_p &= O\left(\sum_{p=1}^{P} (k_p \log f + k_p^2 \log k_p)\right) \\ &= O(\varepsilon \log^2 f + \varepsilon^2 \log^2 f (\log \varepsilon + \log\log f)) \\ &= O\left(\frac{\log^2 f}{\log\log f}\right) \end{aligned}$$

bits in total, as $\varepsilon = \Theta(1/\log\log f)$. Storing the output of the resulting c-counter introduces additional $O(\log c)$ bits. □

5. SAVING ON COMMUNICATION USING RANDOMISATION

So far we have considered the model where each node broadcasts its entire state every round. In the case of the algorithm given in Theorem 3, every node will send $S = O(\log^2 f / \log\log f + \log c)$ bits in each round. As there are $\Theta(n^2)$ communication links, the total number of communicated bits in each round is $\Theta(n^2 S)$. In this section, we consider a randomised variant of the algorithm that achieves better message and bit complexities in a slightly different communication model.

5.1 The Pulling Model

Throughout this section we consider the following model, where in every synchronous round: (1) each processor contacts a subset of other nodes by *pulling* their state, (2) each contacted node *responds* by sending their state to the pulling nodes, and (3) all processors update their local state according to the received messages. As before, faulty nodes may respond with arbitrary states that can be different for different pulling nodes. We define the (per-node) message and bit complexities of the algorithm as the maximum number of messages and bits, respectively, pulled by a non-faulty node in any round.

The motivation for this model is that it permits to attribute the energy cost for a message to the pulling node. In a circuit, this means that the pulling node provides the energy for the signal transitions of the communication link: logically, the link is part of the pulling node's circuitry, whereas the "sender" merely spends the energy for writing its state into the register from which all its outgoing links read.

Our goal will be to keep the number of pulls by non-faulty nodes small at all times. This way a small energy budget per round per node suffices in correct operation. By limiting the energy supply of each node, we can also effectively limit the energy consumption of the Byzantine nodes.

5.2 The High-Level Idea

To keep the number of pulls, and thus number of messages sent, small, we modify the construction of Theorem 1 to use random sampling where useful. Essentially, the idea is to show that *with high probability* a small set of sampled messages accurately represents the current state of the system and the randomised algorithm will behave as the deterministic one. There are two steps where the nodes rely on information broadcast by all the nodes: the majority voting scheme over the blocks and our variant of the phase king algorithm. Both can be shown to work with high probability by using concentration bound arguments.

More specifically, for any constant $\kappa \geq 1$ we can bound the probability of failure by $\eta^{-\kappa}$ by sampling $M = \Theta(\log \eta)$ messages; here η denotes the total number of nodes in the system. The idea is to use a union bound over all levels of recursion, nodes, and considered rounds, to show that the sampling succeeds with high probability in all cases. For the randomised variant of Theorem 1, we will require the following additional constraint: when constructing a counter on $N = kn$ nodes, the total number of failures is bounded by $F < \frac{N}{3+\gamma}$, where $\gamma > 0$ is some constant. Since the resilience of the recursive construction is suboptimal anyway, this constraint is always going to be satisfied. This allows us to construct *probabilistic synchronous c-counters* in the sense that the counter stabilises in time T if for all rounds $t \geq T$ all non-faulty nodes count correctly with probability $1 - \eta^{-\kappa}$.

5.3 Sampling Communication Channels

There are two steps in the construction of Theorem 1 where we rely on deterministic broadcasting: the majority sampling for electing a leader block and the execution of the phase king protocol. We start with the latter.

Randomised Phase King. Instead of checking whether at least $N - F$ of all messages have the same value, we check whether at least a fraction of $2/3$ of the sampled messages have the same value. Similarly, when checking for at least $F + 1$ values, we check whether a fraction $1/3$ of the sampled messages have this value.

Lemma 8. *Let $x \in [C] \cup \{\infty\}$ and suppose a node samples M values from the other nodes. Then there exists $M_0(\eta, \kappa, \gamma) = \Theta(\log \eta)$ so that $M \geq M_0$ implies the following with high probability.*

(a) If all non-faulty nodes agree on value x, then x is seen at least $2/3 \cdot M$ times.

(b) If the majority of non-faulty nodes have value x, then more than $1/3 \cdot M$ sampled values will be x.

(c) If at least $2/3 \cdot M$ sampled values have value x, then x is a majority value.

Proof. Define $\delta = 1 - \frac{2}{3} \cdot \frac{3+\gamma}{2+\gamma}$ and let the random variable X denote the number of x values sampled from *non-faulty* nodes.

(a) If all non-faulty nodes agree on value x, then

$$\mathbf{E}[X] = \left(1 - \frac{F}{N}\right) M > \frac{2+\gamma}{3+\gamma} M.$$

As δ satisfies $(1-\delta)\,\mathbf{E}[X] > 2/3 \cdot M$, Chernoff's bound yields

$$\Pr\left[X < \frac{2}{3}M\right] \leq \Pr[X < (1-\delta)\,\mathbf{E}[X]] \leq \exp\left(-\frac{\delta^2}{2}\,\mathbf{E}[X]\right) \leq \exp\left(-\delta^2 \frac{2+\gamma}{2(3+\gamma)} M_0\right).$$

For sufficiently large $M_0(N,\kappa,\gamma) = \Theta(\log N)$ this probability is bounded by $N^{-\kappa}$.

(b) If a majority of non-faulty nodes have value x, then $\mathbf{E}[X] \geq \frac{1}{2} \cdot \frac{2+\gamma}{3+\gamma} M$. As above, by picking the right constants and using concentration bounds, we get that

$$\begin{aligned}\Pr\left[X \leq \frac{1}{3}M\right] &\leq \Pr[X < (1-\delta)\,\mathbf{E}[X]] \\ &\leq \exp\left(-\frac{\delta^2}{2}\,\mathbf{E}[X]\right) \\ &\leq \exp\left(-\delta^2 \frac{2+\gamma}{4(3+\gamma)} M_0\right) \leq N^{-\kappa}.\end{aligned}$$

(c) Suppose the majority of non-faulty nodes have values different from x. Defining $\bar{X}$ as the random variable counting the number of samples with values different from x and arguing as for (b), we see that

$$\Pr\left[X \geq \frac{2}{3}M\right] = \Pr\left[\bar{X} < \frac{1}{3}M\right] \leq N^{-\kappa},$$

where again we assume that $M_0(N,\kappa,\gamma) = \Theta(\log N)$ is sufficiently large. Thus, $X \geq 2/3 \cdot M$ implies with high probability that the majority of non-faulty nodes have value x. □

As a corollary, we get that when using the sampling scheme, the execution of the phase king essentially behaves as in the deterministic broadcast case.

Corollary 3. *When executing the randomised variant of the phase king protocol from Section 3 for $\eta^{O(1)}$ rounds, the statements of Lemma 4 and Lemma 5 hold with high probability.*

Proof. The algorithm uses two thresholds, $N-F$ and $F+1$. As discussed, these are replaced by $2/3 \cdot M$ and $1/3 \cdot M$ when taking M samples. Using the statements of Lemma 8, we can argue analogously to the proofs of Lemma 4 and Lemma 5; we apply the union bound over all rounds and samples taken by non-faulty nodes ($N-F < \eta$ per round), i.e., over $\eta^{O(1)}$ events. □

Randomised majority voting. It remains to handle the case of majority voting in the construction of Theorem 1. Consider some level of the recursive construction, in which we want to construct a counter of $N = kn$ nodes out of k n-node counters. If $N \ll \log\eta/\log\log\eta$, we can perform the step in the recursive construction using the deterministic algorithm, that is, pulling from all kn nodes. Otherwise, similar to the above sampling scheme for randomised phase king, each node will from each block uniformly sample $M \geq M_0(\eta,\kappa,\gamma) = \Theta(\log\eta)$ states. Again by applying concentration bounds, we can show that with high probability, the non-faulty nodes sample a majority of non-faulty nodes from non-faulty blocks. Thus, we can get a probabilistic version of Lemma 3.

Analogously to Section 3, we define the following local variables at node (i,j) in round q:

$$\begin{aligned} b_q^{i'}[i,j] &= \text{majority}\{b_q[i',j'] : (i',j') \\ &\qquad \text{sampled by } (i,j) \text{ in round } q\}, \\ B_q[i,j] &= \text{majority}\{b_q^{i'}[i,j] \mid i' \in [k]\}, \\ R_q[i,j] &= \text{majority}\{r[B_q[i,j],j'] : (i',j') \\ &\qquad \text{sampled by } (i,j) \text{ in round } q\}. \end{aligned}$$

Here we sample with repetition and the above sets are multisets; this means all samples from a block are independent and we can readily apply Chernoff's bound.

Lemma 9. *Suppose $x \in [k]$ is a non-faulty block, $M_0 = \Theta(\log\eta)$ is sufficiently large, and all non-faulty blocks count correctly in round q. If for all non-faulty blocks i and non-faulty nodes (i,j) it holds that $b_q[i,j] = x$, then with high probability*

1. *$b_q^{i'}[i,j] = x$ for all non-faulty blocks i',*
2. *$B_q[i,j] = x$, and*
3. *$R_q[i,j] = r_q[x,j']$ for an arbitrary non-faulty node (B,j') in block x.*

Proof. Consider a non-faulty block (recall that a block is non-faulty if it has at most f faulty nodes). Let X denote the number of states of non-faulty nodes sampled from this block by (i,j) in round q. As $f < n/3$, we have that $\mathbf{E}[X] \geq M(n-f)/n > 2/3 \cdot M$. Applying Chernoff's bound for $\delta = 1/4$ and choosing sufficiently large $M_0(\eta,\kappa) = \Theta(\log\eta)$, we obtain that

$$\begin{aligned}\Pr[X \leq M/2] &\leq \Pr[X \leq (1-\delta)\,\mathbf{E}[X]] \\ &\leq \exp\left(-\frac{\delta^2}{2}\,\mathbf{E}[X]\right) \leq \eta^{-\kappa}.\end{aligned}$$

Applying the union bound to all nodes and all blocks, it follows that, with high probability, non-faulty nodes always sample a majority of non-faulty nodes from non-faulty blocks. The first statement follows, immediately yielding the second as a majority of the blocks is non-faulty. The third statement now holds because we assume that non-faulty blocks count correctly and x is non-faulty. □

5.4 Randomised Resilience Boosting

Define $\mathcal{P}(n,f,c,\eta,\kappa)$ as the family of probabilistic synchronous c-counters on n nodes with resilience f, where probabilistic means that an algorithm $\mathbf{P} \in \mathcal{P}(n,f,c,\eta,\kappa)$ of stabilisation time $T(\mathbf{P})$ merely guarantees that it counts correctly with probability $1-\eta^{-\kappa}$ in rounds $t \geq T(\mathbf{P})$. Thus with high probability, eventually all non-faulty nodes agree on a common counter for sufficiently many rounds. Together with Corollary 3, we get a randomised variant of Theorem 1.

Theorem 4. *Given $n,f,\eta \in \mathbb{N}$, pick $N,F,C \in \mathbb{N}$ and $\kappa > 0$, where*

- *the number of nodes $N = kn \leq \eta$ for some number of blocks $3 \leq k \in \mathbb{N}$,*
- *the resilience $F < (f+1)m$, where $m = \lceil k/2 \rceil$,*
- *$C > 1$ is the new counter size, and*
- *κ is a constant.*

Choose any $c \in \mathbb{N}$ that is an integer multiple of $3(F+2)(2m)^k$. Then for any $\mathbf{A} \in \mathcal{A}(n,f,c)$, there exists $\mathbf{P} \in \mathcal{P}(N,F,C,\eta,\kappa)$ with the following properties.

1. *$T(\mathbf{P}) = T(\mathbf{A}) + 3(F+2)(2m)^k$, and*
2. *$S(\mathbf{P}) = S(\mathbf{A}) + \lceil \log(C+1) \rceil + 1$.*
3. *Each node pulls $O(k \log \eta)$ messages in each round.*

We can choose to replace $\mathbf{A} \in \mathcal{A}(n,f,c)$ by $\mathbf{Q} \in \mathcal{P}(n,f,c,\eta,\kappa)$ when applying this theorem, arguing that with high probability it *behaves* like a corresponding algorithm $\mathbf{A} \in \mathcal{A}(n,f,c)$ for polynomially many rounds. Applying the recursive construction from Section 4 and the union bound, this yields Corollary 4. By always choosing $k = O(\log \eta)$, each node pulls $O(\log^2 \eta)$ messages from other nodes for each layer.

Corollary 4. *For any $c > 1$, there exist probabilistic synchronous c-counters with a resilience of $f = n^{1-o(1)}$ that stabilise in $O(f)$ rounds, use $O(\log^2 f / \log\log f + \log c)$ state bits per node, and in which the number of messages pulled by a non-faulty node is $O(\log \eta (\log f / \log\log f)^2)$ per round.*

We note that it is also possible to boost the probability of success, and thus the period of stability, by simply increasing the sample size. For instance, sampling polylog η messages yields an error probability of $\eta^{-\operatorname{polylog} \eta}$ in each round, whereas in the extreme case, by "sampling" all nodes the algorithm reduces to the deterministic case.

5.5 Oblivious Adversary

Finally, we remark that under an *oblivious adversary*, that is, an adversary that picks the set of faulty nodes independently of the randomness used by the non-faulty nodes, we get *pseudo-random* synchronous counters satisfying the following: (1) the execution stabilises with high probability and (2) if the execution stabilises, then all non-faulty nodes will deterministically count correctly. Put otherwise, we can fix the random bits used by the nodes to sample the communication links *once*, and with high probability we sample sufficiently many communication links to non-faulty nodes for the algorithm to (deterministically) stabilise. This gives us the following result.

Corollary 5. *For any $c > 1$, there exist pseudo-random synchronous c-counters with a resilience of $f = n^{1-o(1)}$ against an oblivious fault pattern that stabilise in $O(f)$ rounds with high probability, use $O(\log^2 f / \log\log f + \log c)$ state bits per node, and in which the number of messages pulled by a non-faulty node is $O(\log \eta (\log f / \log\log f)^2)$ per round.*

6. CONCLUSIONS

In this work, we showed that there exist (1) *deterministic* algorithms for synchronous counting that have (2) linear stabilisation time, (3) need to store a very small number of state bits between consecutive rounds while still achieving (4) almost-optimal resilience—something no prior algorithms have been able to do. In addition, we discussed how to reduce the total number of communicated bits in the network, while still achieving (2)–(4) by considering probabilistic and pseudo-random synchronous counters.

We conclude by highlighting a few open problems:

1. Are there randomised or deterministic algorithms with optimal resilience of $f < n/3$ that use polylog f state bits and stabilise in $O(f)$ rounds (with small constant factors)?
2. Are there deterministic algorithms that use substantially fewer than $\log^2 f$ state bits?
3. Are there communication-efficient and space-efficient algorithms with high resilience that stabilise quickly in the usual synchronous model?

Acknowledgements

We thank Tuomo Lempiäinen and the anonymous reviewers for helpful comments.

7. REFERENCES

[1] Michael Ben-Or, Danny Dolev, and Ezra N. Hoch. Fast self-stabilizing Byzantine tolerant digital clock synchronization. In *Proc. 27th Annual ACM Symposium on Principles of Distributed Computing (PODC 2008)*, pages 385–394. ACM Press, 2008. doi:10.1145/1400751.1400802.

[2] Piotr Berman, Juan A. Garay, and Kenneth J. Perry. Towards optimal distributed consensus. In *Proc. 30th Annual Symposium on Foundations of Computer Science (FOCS 1989)*, pages 410–415. IEEE, 1989. doi:10.1109/SFCS.1989.63511.

[3] Danny Dolev, Keijo Heljanko, Matti Järvisalo, Janne H. Korhonen, Christoph Lenzen, Joel Rybicki, Jukka Suomela, and Siert Wieringa. Synchronous counting and computational algorithm design, 2015. arXiv:1304.5719v2.

[4] Danny Dolev and Ezra N. Hoch. On self-stabilizing synchronous actions despite Byzantine attacks. In *Proc. 21st International Symposium on Distributed Computing (DISC 2007)*, volume 4731 of *Lecture Notes in Computer Science*, pages 193–207. Springer, 2007. doi:10.1007/978-3-540-75142-7_17.

[5] Danny Dolev, Janne H. Korhonen, Christoph Lenzen, Joel Rybicki, and Jukka Suomela. Synchronous counting and computational algorithm design. In *Proc. 15th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS 2013)*, volume 8255 of *Lecture Notes in Computer Science*, pages 237–250. Springer, 2013. doi:10.1007/978-3-319-03089-0_17. arXiv:1304.5719v1.

[6] Danny Dolev and Rüdiger Reischuk. Bounds on information exchange for Byzantine agreement. *Journal of the ACM*, 32(1):191–204, 1985. doi:10.1145/2455.214112.

[7] Shlomi Dolev. *Self-Stabilization.* The MIT Press, Cambridge, MA, 2000.

[8] Shlomi Dolev and Jennifer L. Welch. 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] Michael J. Fischer and Nancy A. Lynch. 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] Marshall C. Pease, Robert E. Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, 1980. doi:10.1145/322186.322188.