Early-Stabilizing Counting

Christoph Lenzen
Aalto University
Helsinki, Finland
Reykjavik University
Reykjavik, Iceland
christophlen@ru.is

Julian Loss
Ruhr University Bochum
Bochum, Germany
julian.loss@rub.de

Abstract

Synchronous *Counting* is the task of reaching agreement on a common round counter in a synchronous system of n nodes with up to t Byzantine faults in a *self-stabilizing* manner. That is, after transient faults may have arbitrarily corrupted the system state and ceased, the at least $n - t$ non-faulty nodes need to (re-)establish that (i) their local outputs are identical and (ii) increase by 1 modulo C in each round. An overhead-free reduction from consensus shows that all known lower bounds and impossibilities for consensus carry over to the counting problem. In the other direction, prior work has established that a consensus algorithm $\mathcal{A}$ can be turned into a counting algorithm at small overhead relative to the running time and bit complexity of $\mathcal{A}$, without losing resilience.

Taking inspiration from early-stopping consensus protocols, in this work we introduce the concept of *early stabilization.* That is, if there are $0 \leq f \leq t$ (persistent) faults in an execution, the algorithm should stabilize in a number of rounds that depends on f only. Likewise, we seek to achieve an amortized bit complexity that is *adaptive* in the number of actual faults f. By developing a number of modular building blocks suitable to these goals, we develop a C-counting algorithm that stabilizes within asymptotically optimal $O(f + 1)$ rounds, has message size $O(\log^2 n + \log C)$, and has amortized bit complexity $O(n(f \log C + \log^2 n))$.

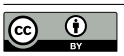
CCS Concepts

• **Theory of computation** → **Distributed algorithms**; *Distributed computing models*; • **Computer systems organization** → *Real-time systems*; • **Hardware** → Very large scale integration design.

Keywords

Byzantine fault-tolerance, self-stabilization, digital clock synchronization, early-stopping

ACM Reference Format:
Christoph Lenzen and Julian Loss. 2026. Early-Stabilizing Counting. In *ACM Symposium on Principles of Distributed Computing (PODC '26), July 06–10, 2026, Egham, United Kingdom.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3796701.3815925



PODC '26, Egham, United Kingdom

ACM ISBN 979-8-4007-2512-8/2026/07
https://doi.org/10.1145/3796701.3815925

1 Introduction & Related Work

Synchronous *Counting*, a.k.a. Byzantine digital clock synchronization, asks the nodes of a synchronous system to agree on a common round counter despite arbitrary initial states and interference from Byzantine faulty parties. This task naturally arises once its big brother, Byzantine fault-tolerant pulse synchronization [11, 18], has been solved: *this* problem establishes synchronized, self-stabilizing regular pulses in a system where nodes are equipped with local clocks of bounded relative drift and exchange messages whose communication delay varies within known bounds. These pulses can serve as a clock signal establishing the synchronous abstraction despite both bounded Byzantine and and unbounded transient faults, but the resulting rounds are "anonymous." This gets in the way of basic operations like initiating a subroutine that should be executed regularly, but not *every* round, or responding coherently to external signals that are not necessarily observed by all nodes in the same round. The latter can be achieved by solving the closely related firing squad problem, cf. [17].

Thus, at first glance, counting is a fundamental challenge with practical utility. However, the latter demands a more nuanced perspective. While applying pulse synchronization naively results in unlabeled rounds, one can use them as a "heartbeat" to stabilize a faster, more accurate pulse synchronization algorithm that would not stabilize on its own, but labels pulses modulo C [14]. This comes at the expense of slowing down the heartbeat by a factor of $\Theta(C)$, which negatively impacts the stabilization time of the overall protocol stack. Ben-Or et al. [3] achieve a much smaller constant expected time overhead, by leveraging a self-stabilizing stream of weak shared coins. However, this comes at the expense of large communication overhead and additional assumptions.[1] Alternatively, one can build large clocks from smaller ones [15], turning the multiplicative overhead in stabilization time of [14] into an additive one. This work gives a deterministic algorithm that runs in $O(t)$ rounds and tolerates $t < n/3$ Byzantine faults, alongside a randomized solution that achieves stabilization within $O(t + \log C)$ rounds with probability $1 - 2^{-t+\log C}$. These bounds are optimal up to small factors in the worst case [5, 13, 20] due to a folklore reduction from consensus to counting. A remaining niche case is when the goal is to stabilize within $S \in \log^{O(1)} n$ rounds. Here, combining [17] and [18] achieves this property with probability $1 - 2^{-\log^{O(1)} n}$, at the expense of higher communication complexity (depending on the consensus algorithm plugged into their framework). This can be viewed a generalization of the aforementioned earlier work

[1] The classic protocol by Feldman and Micali would require $\Omega(n^3\kappa)$ bits *per heartbeat*, where κ is the security parameter, and is secure against a static, computationally bounded adversary. One cannot use more efficient protocols that rely on a trusted setup, as transient faults could reveal any pre-shared secret information.

by Ben-Or et al. [3] that also solves the problem of generating the heartbeats, without needing to rely on a shared coin primitive.

So, should we cast aside the counting problem or at least reduce it to the status of a theoretical curiosity? We argue that this would be premature, as the above characterization limits its view to assuming a worst-case number of faults. The round lower bound for consensus breaks down when considering executions with $f \ll t$ faults [9], or more precisely, requires a more fine-grained argument that yields a round complexity of $\min\{f+2, t+1\}$ [10]. Regarding bit complexity, current solutions all send $\Omega(nt)$ bits *per round,* factor t beyond what the strongest known lower bound for consensus implies for counting [1, 8]. Accordingly, in this work we explore the question how much can be gained by optimizing stabilization time and bit complexity as function of f rather than t.

Our Contributions. To the best of our knowledge, this work is the first to carry over the concept of early-stopping consensus algorithms, which terminate within $O(f+1)$ rounds, to the realm of self-stabilization. We coin algorithms in which the stabilization time depends only on f, i.e., $S = S(f)$, *early-stabilizing*. Likewise, we are unaware of prior attempts to reduce the communication cost of fault-tolerant self-stabilizing algorithms in executions with few faults. On a high level, we perceive three key benefits in doing so.

(1) Reducing communication complexity simply saves resources. While it is true that larger bandwidth still needs to be available *if* there is a large number of faults, these might not simultaneously affect all parts of the system or network. Hence, if communication resources are shared with other subroutines or subsystems, it can even be possible that overall bandwidth requirements are reduced for achieving the same level of resilience.
(2) Reduced stabilization time can clearly improve system availability. However, in the context of self-stabilization, we emphasize a compounding impact that goes way beyond shorter recovery time after a burst of transient faults overwhelms the system. Quicker recovery affects also how quickly nodes undergoing transient faults reattain a state consistent with the system when the fault threshold t is not reached. This entails that they contribute again to the quorum of $n-t$ synchronized nodes required to count despite Byzantine faults. For example, if $t = \lfloor (n-1)/3 \rfloor$ and in each round each node suffers a transient fault with independent probability $9/n$ (and there are no other faults), straightforward calculations and concentration bounds show the following.
    (i) Within t rounds, with probability $1 - 2^{-\Omega(n)}$, more than t distinct nodes suffer a transient fault. Hence, a system in which $t+1$ or more rounds are required for individual nodes to recover from transients would be virtually guaranteed to fail globally within t rounds.[2]
    (ii) Within $t/3$ rounds, the probability that more than t nodes fail *or* that during each subinterval of $S(0) \in O(1)$ rounds some transient fault occurs is $2^{-\Omega(n)}$. Thus, an algorithm that stabilizes in $S(0)$ rounds when there are no faults is guaranteed to maintain count for $2^{\Omega(n)}$ rounds with probability $1 - 2^{-\Omega(n)}$.

    Put simply, early stabilization fundamentally improves the ability of the system to contain transient faults.
(3) In [18], the authors mention that their work on pulse synchronization builds on earlier results on counting, where key techniques were developed in the "sandbox" provided by the more structured synchronous model. We hope the techniques we develop here for counting can be utilized in a similar way.

Concretely, we craft modular tools that ultimately achieve the following main result; fix optimal $t := \lfloor (n-1)/3 \rfloor$ from hereon.

Theorem 1.1. *There is a C-counting algorithm with stabilization time $O(f+1)$, message size $O(\log^2 n + \log C)$, and bit complexity of $O(n(f \log C + \log^2 n))$ amortized over n rounds.*

Compared to the state of the art [15, 17], this reduces stabilization time from being larger than t to asymptotically optimal $O(f+1)$ *and* bit complexity from $\Omega(n^2)$ to (amortized) $\tilde{O}(n(f+1))$ per round.

Paper Organization. In this conference version, we confine ourselves to an overview of the main ideas and showcasing the spirit of the techniques at hand of a weaker result achieving $O(f + \log n)$ stabilization time without improving on the communication complexity. The full version [16] builds on this approach to prove our main result. In Section 2, we discuss the system model, formalize the counting problem, and introduce some tasks that are highly useful in modularizing our solutions. Next, in Section 3 we provide an overview of the key challenges in obtaining our results and sketch the ideas underlying our algorithms and proofs. To prepare for the technical exposition, in Section 4 we introduce a number of conventions we use in pseudocode to hide important, but distracting book-keeping operations and other details.[3] Section 5 discusses how to achieve stabilization time $O(f + \log n)$. We conclude with a discussion of open questions in Section 6.

2 Model and Preliminaries

Communication Model. We consider a fully connected synchronous system with node set $V := [n]$, where we use the shorthand $[k] := \{0, \dots, k-1\}$. That is, algorithms proceed in *rounds,* in which nodes send and receive messages and perform computations in lock-step. When a node v sends a message m to another node w at the onset of a round $r \in \mathbb{N}$, w receives and processes m by the end of round r, knowing that m was sent by v. The *round complexity* of a (non-stabilizing) protocol is the maximum number of rounds until all nodes have decided on their output and terminated, i.e., cease to send messages.

Fault Model. In this work, we seek so-called *self-stabilizing* solutions, i.e., the system recovers correct operation after a period of unbounded transient faults. We assume that by the start of the first round (that we analyze), the most recent such period is over. In addition, we require that the system can sustain any number of Byzantine *faulty* nodes $f \leq t := \lfloor (n-1)/3 \rfloor$, where recovery should succeed despite these ongoing faults. Accordingly, the non-Byzantine nodes, which we refer to as *correct,* may initially have

[2] For this informal line of reasoning, we assume that a transient fault will invalidate a node's state and it will take the full stabilization time to recover a consistent state.

[3] Readers familiar with self-stabilization should not be surprised by any of them, and a conceptual understanding does not require examining them closely. However, they imbue the pseudocode with a sufficiently rigorous meaning to enable proofs.

an arbitrary state, but the model assumptions on the communication network and the number of correct nodes hold again at the beginning of round 1.[4] The goal is for the system as a whole to recover correct operation again within a bounded number of rounds S. This bound is referred to as *stabilization time.* Note that self-stabilizing algorithms can never terminate; the stabilization time is their closest equivalent to round complexity [2].

We consider deterministic algorithms, i.e., the system must stabilize for any initial states of correct nodes and behavior of the faulty nodes. This is equivalent to assuming a powerful adversary that controls initial states and has perfect knowledge of the system, as the initial states and messages sent by faulty nodes fully determine an execution. In contrast, the algorithm is not aware of f, and must achieve its guarantees based on the knowledge that $f \leq t$ only. This is crucial, as in this work we are interested in a stabilization time that depends on f, not t or n: we set out to achieve $S \in O(f+1)$. In analogy to the concept of early-stopping algorithms [9], we refer to this as *early stabilization.*

As a secondary optimization criterion, we seek to minimize the amortized bit complexity of our algorithms. That is, if honest parties send a total of $B(r)$ bits in round r, there should be a value R such that $(\sum_{r=1}^{R} B(r))/R \leq B$. In this case, the algorithm has *bit complexity of B amortized over R rounds.* We remark that it is common to ignore any communication by faulty nodes altogether, as they could send anything. However, our algorithms use short messages of polylogarithmic length, so that correct nodes can safely ignore longer messages without processing them, i.e., no backdoor through which an attacker could overtax correct nodes' computational capacity is introduced.

Finally, our self-stabilizing algorithms will list the variables that are *maintained* between rounds, where we use subscript v to indicate the node. Note that subroutines may come with their own variables, but we will use their outputs only. For analysis purposes, we will refer to the state of a variable X_v of node v *at the end* of round $r \in \mathbb{N}$ by $X_{v,r}$, where $X_{v,0}$ refers to the (arbitrary) initial state of X_v. We do the same for output variables of self-stabilizing subroutines.

Definitions. The core task we consider in this work is digital clock synchronization or synchronous *counting*.

Definition 2.1 (C-Counting). Suppose that each node maintains $C_v \in [C]$, where $2 \leq C \in \mathbb{N}$. We say that these variables *C-count* iff they satisfy for all correct nodes v and w and $r \in \mathbb{N}$ that

- **Agreement**: $C_{v,r} = C_{w,r}$.
- **Validity**: $C_{v,r+1} = C_{v,r} + 1 \bmod C$.

We say that the variables C_v *C-count from round $r \in \mathbb{N}$* iff the above conditions hold in rounds $r' \geq r$.

We formalize a task that is a core step in the construction used in [17], which we will need to implement in a different way. Intuitively, filtering distributes a C-counter that is (recursively) generated by a subset T of the nodes to all, while limiting the inconsistency in views it the subset contains too many faulty nodes to correctly generate and distribute its shared count. More concretely, the requirement is that over any X consecutive rounds, there is only a single count value that increases by one modulo C in each round that may be output by correct nodes; the only other feasible output is $\perp$, a special symbol indicating (possibly transient) faults.

Definition 2.2 ((C,X,T)-Clock Filtering). Suppose that $v \in V$ maintains a variable $F_v \in [C] \cup \{\perp\}$, where $2 \leq C \in \mathbb{N}$. Moreover, there is a designated *clock set* $T \subset V$ that maintains variables $C_v \in [C]$. The variables F_v *(C,X,T)-filter* C_v from round $r \in \mathbb{N}$ iff the following properties are true:

- **Validity**: If fewer than $|T|/2$ nodes in T are faulty and the variables C_v C-count, then the variables F_v C-count from round r.
- **X-round Crusader Agreement**: If for $r' \geq r$ and correct v it holds that $F_{v,r'} = c \bmod C$, then for all correct w and rounds $\hat{r} \in \{r', \dots r'+X\}$, it holds that $F_{w,\hat{r}} \in \{F_{v,\hat{r}} + \hat{r} - r' \bmod C, \perp\}$.

For the sake of notational convenience, we adopt the following conventions with respect to the above definitions. First, if C, X, or T are clear from context, we may omit them. Second, when we say that an algorithm solves one of these problems with stabilization time S, we mean that it maintains variables that count or filter from round S, respectively.

Lemma 2.3 (implicit in [17, 19]). *(C,X,T)-Clock Filtering can be solved with stabilization time $S = X + 2$, where each node sends an $O(\log |C|)$-sized message to each other node in each round.*

Proof. We claim that the following algorithm solves the task.

Algorithm 1: (C,X,T)-filtering algorithm implicit in [17, 19].

Input: $C_v \in [C]$ iff $v \in T$
Variables: $m_v, M_v \in [C] \cup \{\perp\}$, $X_v \in [X+1]$
Output: $F_v \in [C] \cup \{\perp\}$

```
send (C_v, m_v) to all nodes
if received (C, ·) from the majority of nodes w ∈ T then
    m_v := C else m_v := ⊥
if received (·, m) from at least n − t nodes w ∈ V then
    if m = M_v + 1 mod C then X_v := max{X_v − 1, 0} else
        X_v := X
    M_v := m
else X_v := X
if X_v = 0 then F_v := M_v else F_v := ⊥
```

The bound on message size can be readily verified from this description.

Validity: If fewer than $|T|/2$ nodes in T are faulty and the variables C_v C-count, the variables m_v and M_v count from rounds 1 and 2, respectively. Then all counters X_v count down to 0 by the end of round $X+2$, so that the outputs count from round $X+2$.

X-round Crusader Agreement: If in round $r \geq X+2$ correct v outputs $C_{v,r} \in [C]$, it follows that it received $M_{v,r'} = m_{w,r'-1} = C_{v,r} + r' - r \bmod C$ from at least $n-t$ nodes $w \in V$ each in rounds $r' \in \{r-X, r-X-1, \dots, r\}$. At least $n - 2t > t$ of these are correct, implying that no w receives $n-t$ times another value in round r'. Hence, $M_{w,r'} \in \{M_{v,r'}, \perp\}$ for each correct w and such r', implying that also $F_{w,r'} \in \{M_{v,r'}, \perp\} = \{C_{v,r} + r' - r \bmod C, \perp\}$, as required. □

[4] In particular, the code of the algorithm must be protected against alterations by transient faults, e.g. by being stored in more resilient non-volatile memory.

A well-known building block of consensus algorithms is graded agreement.

Definition 2.4 (Graded Agreement). The *Graded Agreement* problem is specified as follows. Each node v has input $x_v \in \mathcal{V}$ and computes an output $(y_v, g_v) \in \mathcal{V} \times \{0,1\}$ with the following guarantees:

- **Validity**: If there is $x \in \mathcal{V}$ so that $x_v = x$ for all correct v, then $(y_v, g_v) = (x, 1)$ for all correct v.
- **Graded Agreement**: If $g_w = 1$ for some correct w, then $y_v = y_w$ for all correct v.

Lemma 2.5 (implicit in [4], see also [12]). *Graded agreement can be solved in 2 rounds, with each node sending an $O(\log|\mathcal{V}|)$-sized message to each other node in each round.*

The full version [16] provides a proof of this well-known result.

We will make use of variants of the central building block of the Phase King algorithm from [4], which can be generalized to solve the following weaker form of multi-valued consensus.

Definition 2.6 (King Consensus). In the *King Consensus* problem, each node v has input $(x_v, \ell_v) \in \mathcal{V} \times (V \dot\cup \{\perp\})$ and computes output $y_v \in \mathcal{V} \dot\cup \{\perp\}$ with the following guarantees:

- **Validity**: If there is $x \in \mathcal{V}$ so that $x_v = x$ for all correct v, then $y_v \in \{x, \perp\}$ for all correct v.
- **Default**: If $\ell_v = \perp$ for all correct v, then $y_v = \perp$ for all correct v.
- **King Agreement**: If there is correct $\ell \in V$ so that $\ell_v = \ell$ for all correct v, then $y_v = y_\ell \neq \perp$ for all correct v.

King consensus relaxes classic (multi-valued) consensus in two ways. If all nodes' inputs agree on the same correct leader ℓ, intuitively it behaves like consensus: all outputs agree, and if $x_v = x$ for all nodes v, then this output is going to be x. However, we relax the validity condition in that if $x_v = x$ for all nodes v, then it is sufficient to output x or $\perp$. This provides leeway to save on communication when nodes think that there is no need to execute consensus at all, represented by $\ell_v = \perp$. The second relaxation is that agreement is only guaranteed if all nodes' inputs agree on some correct leader $\ell = \ell_v$, allowing for an $O(1)$-round solution that still meets the requirements if the nodes do not agree on the filtered clock values.

Lemma 2.7 (implicit in [4]). *King Consensus can be solved in 3 rounds, where each node sends an $O(\log|\mathcal{V}|)$-sized message to each other node in each round.*

Proof. We claim that the following algorithm solves the task.

Algorithm 2: King consensus algorithm implicit in [4].

Input: $(x_v, \ell_v) \in \mathcal{V} \times (V \dot\cup \{\perp\})$
Output: $y_v \in \mathcal{V} \dot\cup \{\perp\}$

1 run the protocol of Lemma 2.5 with input x_v; denote output by (z_v, g_v) `// two rounds`
2 **if** $\ell_v = \perp$ **then** $y_v := \perp$
3 **else if** $g_v = 0$ *and received* z_{ℓ_v} *from* ℓ_v **then** $y_v := z_{\ell_v}$
4 **else** $y_v := z_v$

The running time and bound on message size can be readily verified from this description.

Validity: If there is $x \in \mathcal{V}$ so that $x_v = x$ for all correct v, by validity of graded agreement all correct nodes output $(z_v, g_v) = (x, 1)$ from their call to graded agreement. Hence, they all output x or $\perp$, as required.

Default: Immediately follows from the output instruction.

King agreement: If there is correct $\ell \in V$ so that $\ell_v = \ell$ for all correct v, distinguish two cases. If some correct node w outputs $(z_w, 1)$ from the call to graded agreement, the graded agreement property ensures that each correct v outputs (z_w, g_v) from this call for some $g_v \in \{0,1\}$. In particular, $\ell = \ell_\ell$ sends $z_\ell = z_w$ in the third round, so that each correct node outputs $y_v = z_\ell = z_w$. The other case is that no correct node outputs $(z_w, 1)$ for some z_w from the call to graded agreement. In this case all nodes output z_ℓ. □

3 Technical Overview

In this section, we present the high-level ideas underlying our results. The focus is on the main obstacles and solutions. We follow the same order as in the subsequent sections, so this outline sketches the progression of the technical exposition in the remainder of the paper.

The big picture. Essentially, solving counting boils down to solving consensus on the current values of the output counters or clocks: if they all agree, they should not be changed except for being incremented deterministically by one modulo C each round; otherwise, *any* agreed-on value is fine. This corresponds to the validity and agreement properties of consensus, but runs into the issue that there is no (guaranteed) agreement as to *when* to run consensus. Comparatively simple solutions can be constructed by initializing a new instance to consensus *every* round. However, this is inherently inefficient in terms of communication, as it incurs the communication cost of a complete run of a consensus protocol per round.

Other approaches rely on randomization to get "lucky" and achieve sufficient coordination, as e.g. done in [15]. However, any such approach has the structural properties subjecting it to the same trade-off between the probability of stabilization and the time to achieve it as the one between round complexity and probability of success for consensus [5].

The third type of method that has been used in the literature is to partition the node set into V_0 and V_1, solve the task recursively on each of these sets, and then use the obtained counters to coordinate when consensus instances are initialized, cf. [17]. Using a balanced partition limits the recursion depth, number of instances each node participates in, and incurred overheads to $O(\log n)$. We follow this strategy as the only one being compatible with our aims. While we have to swap out every component used in [17] to achieve our main result, it is instructive to start from their solution and evolve it step by step.

Conveniently, it is trivial to solve counting when $n = 1$, so the challenge is to (efficiently) solve counting on $n = 2^k$ nodes given a working algorithm for $n/2 = 2^{k-1}$ nodes. If in neither of the two sets V_b a third of the nodes would be faulty, both sets would stabilize to produce counting outputs. The nodes of V_b would send their current output value to all nodes, which would adopt the majority

value as (perceived) "clock" C_b for each b. We could use these clocks to initialize a consensus instance that terminates after R rounds every $2R + 1$ rounds, where the time to run the instance as well as the input given to it are given by the value taken by C_b. We could then set the current output clock value to the output of any terminating instance plus R mod C; otherwise, the output is simply incremented by 1 mod C. While the clocks and hence the instances might be arbitrarily aligned relative to each other, one of them is going to terminate while no "companion" instance controlled by the other clock is running. From then on, all clock values agree, so that future consensus instances do not affect the count, as they always end up outputting the already agreed-on input, which is then incremented by exactly R mod C.

The obstacle this strawman approach runs into is that there is no guarantee that C_b counts correctly for both $b \in \{0, 1\}$, or even that all correct nodes perceive the same majority value sent by nodes in V_b. Abstractly speaking, we have two supernodes V_0 and V_1 each supplying a clock, but one of them may be Byzantine, sending arbitrary values. In essence, the authors of [17] address this by preventing equivocation via applying crusader broadcast (a.k.a. crusader agreement, see [7]). This means to ensure that there is a unique (clock) value $c \in [C]$ such that each correct party either accepts c as the value sent by V_b or outputs $\perp$ (indicating that V_b is Byzantine or has not stabilized yet). However, "equivocation" here needs to be interpreted in a broader sense: if what V_b sends is not counting modulo C, this is also showing that something is still amiss, and recipients will output $\perp$ whenever the count was off within the last X rounds. This is captured by the X-round crusader agreement property of the clock filtering task.

This does not yet fully overcome the above obstacle, as now some correct nodes may participate in a consensus instance triggered by C_b and others do not. While it is possible to require that any already established agreement is not broken this way, the "bad" instance controlled by C_b might be initialized before one controlled by C_{1-b} terminates. Hence, some nodes might change their output clocks as a result, breaking the agreement that has just been reached with the help of the instance controlled by C_{1-b}. To address this, the final tweak is to have the two clocks C_b initialize instances at slightly different rates, e.g. every $2R$ and $3R$ rounds, respectively. Then, the above scenario implies that the next instance controlled by C_{1-b} runs without the same kind of interference and achieves stabilization of the output clock values.

In summary, the strategy from [17] works as follows.

(1) Partition $[n] = V_0 \dot\cup V_1$.
(2) Recursively let V_b count modulo $C \in \Theta(n)$.
(3) Use these outputs as inputs to $(C, \Theta(n), V_b)$-filtering.
(4) Use the outputs of filtering to initialize consensus instances (where even if some nodes do not participate, validity holds).
(5) Use any outputs of such a consensus instance to adjust the output clock; otherwise increment by one modulo C.

We stress that naively replacing the consensus routine with an early-stopping one is insufficient for our purposes: we do not know at which frequency to initialize fresh instances to (i) achieve stabilization time $O(f + 1)$ yet (ii) avoid that instances remain well-separated.

Reducing stabilization time. In light of the above, to make the stabilization time of a scheme like this $O(f + 1)$, we need to use (i) $(C, O(1), V_b)$-filtering and (ii) a consensus subroutine that runs for $O(1)$ rounds only. Alas, achieving consensus deterministically takes at least $t + 1$ rounds, so we need to relax the task somehow. Hence, the first crucial adaptation is to replace consensus by king consensus with varying "kings" or *leaders* ℓ, where F_b tells us for each instance which leader to use. Note that king consensus relaxes the agreement property of consensus to hold only when the leader is correct. Fortunately, we only need the (king) agreement property *once,* to establish agreement on the count; afterwards, validity ensures that the output clocks are never modified again by a call to king consensus and hence count correctly, regardless of whether future leaders are correct. Each call to king consensus requires only $O(1)$ rounds, achieving (ii). At the same time, the output clocks stabilize at the latest when a correct leader is reached, i.e., within $O(f + 1)$ rounds—counting from the round in which the clock filtering subroutine stabilized.

As already indicated, to achieve fast stabilization for clock filtering, we need to choose $X = O(1)$, i.e., "forgive" an incorrect count within $O(1)$ rounds. This allows the adversary to make the value of F_b "jump" throughout the course of the stabilization process if V_b contains at least $|V_b|/3$ faulty nodes. To address this, we choose X as a large enough constant such that the additional "collisions" that the adversary can induce between instances controlled by F_b and F_{b-1} are limited to a small fraction, say $1/4$, of the instances. While this may prevent *some* correct leaders from enforcing stabilization, among $2f$ instances there will be at least f correct leaders, so that at least $2f - f - 2f/4 > 0$ succeed in stabilizing the output clock values.

Note that this recursion scheme succeeds based on the *first* clock C_b that stabilizes. Accordingly, we can determine a stabilization time bound based on the recurrence $S(f, n) \leq O(f) + S(\lfloor f/2 \rfloor, \lceil n/2 \rceil)$, leading to $S(f, n) \leq O\left(\sum_{i=0}^{\lceil \log n \rceil} \max\{f/2^i, 1\}\right) = O(f + \log n)$. One could remove the additive $O(\log n)$ by alternating between partitioning evenly and splitting off a single node in the recursion, as then the summation would end once $f/2^i < 1$. We postpone this, as such an imbalanced split poses its own challenges when also trying to control bit complexity.

Reducing amortized bit complexity. In order to achieve an amortized bit complexity of $\tilde{O}(n(f + 1))$ per round in our framework, we need to reduce the bit complexities of both the filtering procedure and king consensus. The first observation is that, so long as we have $|V_0| \approx |V_1| \approx n/2$, it is sufficient to do so only when V_b contains fewer than $|V_b|/3$ faults for *each* $b \in \{0, 1\}$, as otherwise $\tilde{O}(n(f + 1)) = \tilde{O}(n^2)$. Similarly, we do not need to worry about the communication cost during the stabilization phase, as the construction uses small messages. During $\tilde{O}(f + 1)$ rounds, correct nodes send $\tilde{O}(n^2(f + 1))$ bits, contributing $\tilde{O}(n(f + 1))$ to the average over n rounds, which is our target.

This is excellent news, as it allows us to work with the assumption that the recursively constructed counters C_b both already count and all correct nodes agree on their values. Thus, king consensus instances are initialized with all correct nodes in agreement on when they start and which party is the leader. In particular, there are only $O(f)$ instances with faulty leaders within n rounds, i.e.,

such instances also contribute $\tilde{O}((f+1)n)$ bits amortized over n rounds.

Accordingly, we only need to control the communication cost of king consensus instances with agreed-on correct leaders and those without a leader.[5] To do so, let us revisit the implementation of king consensus derived from the work introducing the Phase King algorithm [4] for the special case that all nodes agree on a leader ℓ, i.e., $\ell_v = \ell \neq \perp$ for all $v \in [n]$.

(1) Perform graded agreement on the input values x_v, resulting in outputs (y_v, g_v).
(2) The leader ℓ sends y_ℓ to all nodes.
(3) If $g_v = 1$, v outputs y_v. Otherwise, v outputs y_ℓ.

The reader can readily verify that the guarantees of graded agreement ensure the validity and king agreement properties of king consensus.[6] The leader's transmission is uncritical from the perspective of communication cost, but graded agreement is costly, requiring $\Omega(n^2)$ bits.

To reduce the bit complexity of king consensus in this special case, we first have the king determine whether there is actual *need* to establish agreement, by having each node send their input to the king. If the system has stabilized, the king will receive the same input from $n - t$ nodes and conclude that an expensive graded consensus is not necessary.

However, the king might reach this conclusion also prior to stabilization, with up to t correct nodes having a differing minority input value. To these nodes, the king points out that their input value differs from the majority. While the king cannot be trusted blindly, this is justification for these nodes to ask all others for their input and for them to respond: we already pointed out that we can bear a cost of $\tilde{O}(n^2)$ bits if the king is faulty, and faulty nodes querying correct nodes for their input can cause no more than $\tilde{O}(nf)$ bits of communication. When a querying node receives at least $n - 2t \geq t + 1$ values different from its own, it can safely adopt this value, as this is proof that its own input is not shared by all correct nodes. Thus, if the king concludes that no graded consensus is necessary, all correct nodes will adopt the majority value.

On the other hand, if the king receives fewer than $n - t$ times the same value, this is proof that stabilization has not yet occured. In this case it triggers graded agreement. Note that this step empowers a faulty king to invoke graded agreement with only a subset of the correct nodes taking part. We resolve this issue by defining a weaker version of graded agreement. Here, formally all nodes take part, but the graded agreement property only holds if all correct nodes have an additional input bit s_v equal to 1. A correct king uses the latter to create the behavior of a "regular" graded agreement when needed; the validity property remains unchanged, preventing a faulty king from disrupting existing agreement. This relaxation allows an implementation in which node v stays silent, i.e., sends no messages, if $s_v = 0$, achieving the desired reduction in communication cost if all correct nodes share the same input to the king consensus instance.

To achieve filtering at amortized cost of $\tilde{O}(n(f+1))$ bits, recall that we may assume that $f < |V_b|/3$ and that the counting instance on V_b has already stabilized. Hence, all but f nodes in $|V_b|$ are correct and agree on C_b. Node v maintains a "guess" of the current value of C_b and memorizes what it believes this value to be at all other nodes. If the guess has been counting for X rounds and no inconsistency was proven, it will output it, otherwise it will output $\perp$. The algorithm now carefully implements several checks and queries to other nodes' current guesses or C_b values (for those in V_b) to ensure that (i) if the guess of v is incorrect, it will correct it within $O(f+1)$ rounds and (ii) after stabilization, only $\tilde{O}(n(f+1))$ bits are sent on average.

The principal ideas to achieve this are the following.

(1) Node v queries nodes $w \in V_b$ for the C_b value on a round-robin basis, one per round ($\tilde{O}(n)$ bits). If w responds with a value different from v's current guess, v queries all nodes, causing an amortized cost of $\tilde{O}(nf)$ bits.
(2) If v believes that a majority of nodes in V_b has a different value than its current guess, it adjusts its guess, sends its new guess to all nodes, and queries all nodes (for verification purposes). This happens at most twice after stabilization of C_b, as after a query to all nodes v updated all stale information. Hence, the amortized cost is $\tilde{O}(n)$ bits. Together with the first rule, this guarantees that all nodes adopt the correct guess within $f + O(1)$ rounds.
(3) If a node has a guess that is different from the locally memorized guess of another node, it queries that node and updates its memory based on the response. Once all correct nodes adopted the correct guess and communicated about this, this costs only $\tilde{O}(nf)$ bits per round, since there are no queries between correct nodes anymore.

This last rule enforces that correct nodes with different guesses notice the discrepancy. If a node believes that more than $n/3$, i.e., more than t, guesses deviate from its own, it can conclude that not all correct nodes agree on C_b and will output $\perp$ at least for the next X rounds. This guarantees that there is only one non-$\perp$ value output by correct nodes within X rounds, i.e., the X-round crusader agreement property is satisfied. On the other hand, if V_b contains fewer than $|V_b|/3$ faulty parties, all correct parties adopt the majority value in V_b and will synchronize their views of each other's guesses. Afterwards, the will not perceive any inconsistencies and start outputting the majority value within an additional $X \in O(1)$ rounds, showing the validity property of clock filtering.

Achieving asymptotically optimal stabilization time. As mentioned earlier, removing the additive $O(\log n)$ from the stabilization time is challenging when seeking to simultaneously keep the amortized bit complexity small. Our approach is to use a recursion template where one branch has only a single node ℓ, which can easily generate and distribute a count (if correct), while the other branch generates its clock C by recursing on the *entire* node set. By alternating between the two templates we ensure stabilization in $O(1)$ rounds once the current node set contains no faults, yet cut the number of faults in the branch with fewest faults in half with

[5] As validity must hold even if nodes do not agree on who the leader is or even whether an instance is running, we formalize this as an instance being initiated in *every* round, but nodes using input $\perp$ to indicate that they have no leader for this instance. This permits correct nodes to "help" in achieving the properties if necessary even when they believe that no instance should be started.

[6] Outputs of $\perp$ and the default property handle the case that some or all nodes have input $\ell_v = \perp$, which we ruled out for the special case we consider here.

every other level of recursion, maintaining the geometric decay in stabilization time on the "fastest" branch.

The primary obstacle to keeping a small communication footprint with this approach is that if ℓ is faulty, we cannot prevent it from manipulating the clock it generates to make itself leader every $X+1$ rounds. However, we need that $X \in O(1)$ to ensure fast stabilization if $f = 0$. Hence, our previous strategy of amortizing the cost incurred by instances with faulty leaders is not going to work here. In fact, we can give up on filtering altogether and accept that ℓ is the leader in every instance, letting it choose when to initiate king consensus on its own. If correct, ℓ will choose a time that prevents an overlap with an instance controlled by C, and we let nodes ignore ℓ if it attempts to initiate an instance that would collide with one controlled by C.

To avoid paying too much for the king consensus instances lead by ℓ, we relax the king agreement property to what we term *weak king agreement:* only if there are no faults whatsoever, ℓ needs to be able to get correct nodes to adopt the same value if their inputs differ. This relaxation to the *weak king consensus* task is irrelevant for our stabilization mechanism, as (weak) king agreement is only needed to stabilize the output clocks, and for this recursive pattern, the single-node branch of ℓ needs to ensure stabilization only if there are no faults.

On the other hand, this provides us with just enough leeway to ensure that weak king consensus can be implemented at bit complexity $\tilde{O}(n(f+1))$ even if ℓ is faulty. Our strategy here is to not rely on ℓ for "justifying" communication by correct nodes. Instead, we employ the following strategy, where we describe the special case that no node ignores the leader.

(1) Nodes compare their current clock value to their neighbors in a constant-degree expander. If stabilization has been achieved, i.e., all correct nodes have the same clock value, no more than $O(f)$ nodes, those with a faulty neighbor in the expander, can observe a discrepancy. On the other hand, if $f = 0$ and $k > 0$ nodes have a clock value different from the majority value, at least εk nodes, where the constant ε depends on the expander, observe a discrepancy.
(2) All nodes observing a discrepancy alert all nodes of this. If v receives k_v such alert messages, it queries each node $w \in \{v, \dots, v + \lceil 2k_v/\varepsilon \rceil \bmod n\}$ for its clock value; these nodes respond with their clock values. If the system has already stabilized, all of this incurs only $\tilde{O}(nf)$ bits of communication: by the previous point, then $k \in O(f)$ at each node v, so there are $O(nf)$ notification, query, and response messages in total.
(3) All nodes for which half of their queries resulted in responses with clock values different from their own now query *all* nodes for their clock values, which respond. Again, by the choice of the communication pattern in the previous step, if all correct nodes agree, this does not cause more than $O(f)$ nodes to perform such queries, resulting in $O(nf)$ messages in total. On the other hand, if $f = 0$, each node that does not have the majority clock value will be among the querying nodes in this step and learn that at least half of the nodes (claim to) have a different clock value. In particular, more than t nodes have a different clock value, proving to them that it is safe, i.e., will not violate validity, to adopt a clock value proposed by ℓ.
(4) This is, finally, the point where ℓ steps in and proposes the majority value it perceives to all nodes. If $f = 0$, this *is* the actual majority value. All nodes that learned in the previous step that it is safe to change value adopt the proposal by ℓ, which in case of $f = 0$ establishes agreement, i.e., weak king agreement is satisfied. All other nodes simply output their input value, guaranteeing validity.

4 Conventions for Pseudocode

We will employ subroutines that are not self-stabilizing, but run only for a fixed number of rounds $R \in \mathbb{N}$. We refer to such a subroutine as an *R-round algorithm.* Note that it is trivial to obtain an R-round algorithm from any algorithm of round complexity at most R, by instructing nodes to wait until round R before terminating and producing their output. There is also a fixed bound on the number of subroutine calls that are initialized per round. Each such instance has its own dedicated memory for each of its rounds, where we tacitly assume that there is a known bound on the amount of memory the subroutine requires. This approach can be seen as a implementing "self-stabilizing pipeline" for R-round algorithms: within R rounds, all inconsistent state from a period of transient faults is erased. Note, however, that we cannot enforce a globally consistent initialization without a common round counter, which is the key challenge of this work.

Moreover, we will run self-stabilizing algorithms as subroutines on a subset of the nodes as part of our recursive solutions. Again, there is a fixed number of such routines. We emphasize this, because it requires care when seeking to reduce the amortized bit complexity to $o(n^2)$: this requires that in many rounds, most pairs of nodes do not exchange any messages. To make this possible while freely combining subroutines, the main routine gathers all messages to be sent to a given destination and concatenates them as a list of pairs consisting of a bit string identifying a subroutine and the corresponding message. If no message is sent by a given subroutine in a given round, we can safely omit the respective pair from the list. Whenever a subroutine sends a message, we will assume that it has size $\Omega(\log n)$ for analysis purposes (even if it is smaller). Since in our recursive algorithms each node participates in $O(\log n)$ different subroutines, this approach bears no asymptotic cost on communication bounds, but greatly simplifies the description of our routines.

With this convention, we can leave the association of subroutine messages to the compiler. In fact, we will allow a constant number of send instructions to be executed by the same (sub)routine in a single round, tacitly assuming that they are treated in the same way. Moreover, we will occasionally send "special" messages that have a unique name, e.g. "req," but no content. Again, we leave it to the compiler to choose an encoding such that these cannot be misinterpreted as, e.g., a clock value being sent. For convenience, we also assume that the compiler will choose an encoding that is not too wasteful, i.e., we can assume that if k different messages can possibly be sent by a (sub)routine, the encoding usese $O(\log k)$ bits. Faulty nodes may of course deviate from these rules. However, the compiler will instruct nodes to drop any received bit string that

is not encoding a valid message. Note that it may still happen that due to an inconsistent (global) state or more than t faults,[7] code might produce out-of-spec results or not be able to execute at all. To address this, we make the following assumptions:

- The compiler will insert checks for all state variables to make sure that the stored value encodes a valid value. If this is not the case, it resets the variable to an arbitrary in-spec default before executing the code.
- R-round algorithms that are used as subroutines are given by a state machine. This state machine takes the round number, input (for round 1) or state at the end of the previous round (for rounds $2, \dots, R$) to compute the messages it sends in that round, and with the additional input given by messages received from other nodes computes the state at the end of the round (for rounds $1, \dots, R-1$) or output (for round R). For all our non-stabilizing subroutines, it will be obvious from the description that they can be realized this way with bounded memory. The number of required rounds R will also be readily visible.
- Pseudocode for self-stabilizing (sub)routines executes successfully for any valid state and received messages. Again, this will be obvious from the description of our self-stabilizing (sub)routines. A more subtle point in this regard is that the code must also execute *in a single round,* as we cannot rely on the node knowing "which round it is," i.e., any notion of round progression must be explicitly captured by local variables that can be affected by transient faults. To address this, we require that once a variable has been changed by an instruction that is conditioned on message reception, it cannot be used in any subsequent send instruction. Equivalently, when executing the code for a self-stabilizing routine in round r, the content of each message sent is a function of the local variables' states $X_{v,r-1}$ at the end of round $r-1$.
- Bounds on message size are maintained even if $f > t$. This is the only property we use in subroutine calls on node sets with too many faults, and it will be clear from our descriptions of algorithms as well.

Moreover, we will frequently use statements of the form "if received k times value x." In such statements we tacitly assume that the respective messages are sent by distinct senders. In particular, if a faulty node sends more than one message in a given round, the compiler ensures that a correct receiver will process only the first one that arrives. Moreover, for notational convenience, nodes may send messages "to themselves;" in particular, a statement like "send x to all nodes" executed by v will trigger a subsequent "if received x from w" condition at v with $w = v$ at node v.

5 Early-Stabilizing Counting

In this section, we develop the main approach for achieving stabilization time of $O(f+1)$, leaving the issue of small amortized bit complexity to the full version [16]. Observe that for a single node, C-counting is trivial to solve with $S = 1$: the solitary node increments a counter modulo C in each round and outputs its value. We will now solve C-counting recursively. To this end, we establish a general template for recursive C-counting, whose pseudocode is given in Algorithm 3.

As outlined in Section 3, we partition the node set into V_0 and V_1, execute a counting algorithm on each of them, and pass the outputs through filtering subroutines. We use *their* output to initiate instances of king consensus, cycling through leaders, but at slightly different frequencies determined by constants k_b, where $b \in \{0,1\}$.

In the following, for $b \in \{0,1\}$ denote by S_C^b the stabilization time of the instance of $(k_b n)$-counting on node set V_b. Similarly, S_F^b is the stabilization time of the instance of $(k_b n, X, V_b)$-filtering. First, let us note that if there are not too many faults within V_b, then the variables F_v^b stabilize and begin to count.

Lemma 5.1. *Suppose that for $b \in \{0,1\}$, it holds that $f < |V_b|/3$. Then the variables F_v^b $(k_b n)$-count from round $r \geq S_C^b + S_F^b$.*

Proof. Within S_C^b rounds, the variables C_v^b begin to count. By validity of clock filtering, the variables F_v^b thus count from round $S_C^b + S_F^b$. □

On the other hand, even if there are too many faults in V_b, the filtering subroutine ensures that there is a "virtual clock" $\hat{F}_r^b \in [k_b n]$

Algorithm 3: A recursion template for C-counting on node set V. We provide the code $v \in V$ executes in each round. The positive integers $X, R, k_0, k_1 \in O(1)$ will be fixed later. The template is parametrized a partition $V = V_0 \dot\cup V_1$, solutions to counting on these sets, to filtering with them as clock sets, and to R-round king consensus on node set V.

Variables: for $b \in \{0,1\}$, outputs C_v^b and F_v^b of counting on V_b and filtering with clock set V_b
Output: $C_v \in [C]$
1 **foreach** $b \in \{0,1\}$ **do**
2 **if** $F_v^b \bmod k_b n = k_b w$ **then** $\ell_v^b := w$ **else** $\ell_v^b := \bot$
3 initialize king consensus on universe $[C] \cup \{\bot\}$ with input $(C_v + R \bmod C, \ell_v^b)$
4 **for** $r \in \{R, \dots, 1\}$ **do**
5 execute the code of round r of king consensus on the state stored for round r and b
6 **if** *$r = R$ and the output of the instance is $c \in [C]$* **then**
7 $C_v := c$
8 **else**
9 store the new state in the memory block allocated for round $r+1$ and b
10 **if** $v \in V_b$ **then**
11 execute the code of an instance of $(k_b n, X, V_b)$-filtering with input C_v^b and output F_v^b
12 execute the code of an instance of $(k_b n)$-counting on node set V_b with output C_v^b
13 **else**
14 execute the code of an instance of $(k_b n, X, V_b)$-filtering with output F_v^b
15 $C_v := C_v + 1 \bmod C$

[7] While we assume that $f \leq t$ during our analysis, note that subroutine calls on $n' < n$ nodes may have to contend with $f' > n'/3$ faults.

that counts for during each X consecutive rounds, such that correct node v either outputs $F^b_{v,r} = \hat{F}^b_r$ or $F^b_{v,r} = \bot$ in round r.

Lemma 5.2. *For $b \in \{0,1\}$ and rounds $r \geq S^b_F$, there is a* virtual clock $\hat{F}^b_r \in [k_b n]$ *such that (i) for all correct nodes v, it holds that $F^b_{v,r} \in \{\hat{F}^b_r, \bot\}$ and (ii) unless $r - r_0 = 0 \bmod X$, $\hat{F}^b_r = \hat{F}^b_{r-1} + 1 \bmod k_b n$.*

Proof. To define $\hat{F}^b$, consider in each interval $I_i := [r_0 + iX, r_0 + i(X+1) - 1]$ the minimal round $r \in I_i$ such that $F^b_{v,r} \neq \bot$ for some correct v and set $\hat{F}^b_{r'} := F^b_{v,r} + r - r'$ for all $r' \in I_i$ (if no such r exists, use an arbitrary value instead of $F^b_{v,r}$). By the X-round crusader agreement property of clock filtering, this results in a well-defined $\hat{F}^b$ meeting the requirements. □

The main pillar of the proof of stabilization of Algorithm 3 is showing that eventually, there will be a "good" instance of king consensus, in the sense that all nodes agree on a correct leader and there is no "competing" instance that is initialized during its execution.

Definition 5.3 (Unimpeded Instances). For $b \in \{0,1\}$, we say that the b-th instance of king consensus initialized in round r *runs with leader* $\ell \in V$ if (i) the variables F^b_v count from round r and (ii) each correct node v inputs (x_v, ℓ) for some $x_v \in [C]$. Furthermore, such an instance is *unimpeded* if in addition (i) ℓ is correct and (ii) correct nodes v initialize all other instances of king consensus in rounds $r' \in \{r, r+1, \dots, r+R\}$ with $\ell_v = \bot$.

Before proving that such an instance will occur in a timely fashion, let us formalize that an unimpeded instance translates to stabilization.

Lemma 5.4. *Assume that in round $r \in \mathbb{N}$ an unimpeded instance of king consensus is initialized. Then the variables C_v count from round $r + R$.*

Proof. By the king agreement property of R-round king consensus, there is some $y \in [C]$ such that all correct nodes v output $y_v = y$ from the unimpeded instance. Observe that, granted that no king consensus instance outputs a value different from $y + r' - r \bmod C$ or $\bot$ in a round $r' \geq r$ at some correct v, correct nodes will set $C_{v,r} := y$ and, by induction, the variables C_v will count from round r.

To show this, recall that the definition of an unimpeded instance requires that correct nodes v have $\ell_v = \bot$ in all other king consensus instances initialized in rounds $r' \in \{r, r+1, \dots, r+R\}$, i.e., from the initiliazation of the unimpeded instance until it outputs in round $r + R$. By the default property of king consensus, this entails that all other instances initiated in these rounds output $\bot$ at correct nodes.

Accordingly, it remains to consider instances that are initiated in rounds $r' > r$. Assume for contradiction that there is a minimal round $r' > r$ such that an instance initiated in round r' outputs $y_v \notin \{y + r' - r + R \bmod C, \bot\}$ at some correct v (which it does in round $r' + R$). As this instance is initialized after round r, but is the first instance that outputs $y_v \notin \{y + r' - r + R \bmod C, \bot\}$, correct nodes used inputs $x_v \in \{y + r' - r + R \bmod C, \bot\}$ to the instance. By the validity property of king consenus, they must output $y_v \in \{y + r' - r + R \bmod C, \bot\}$, reaching a contradiction and concluding the proof. □

In order to show that suitable parameter choices result in quick emergence of an unimpeded instance, we first state a straightforward, but technical helper lemma for said choices.

Lemma 5.5. *Suppose that a constant $0 < \varepsilon < 1$ and natural $R \in O(1)$ are given. Then there are natural $R < k_0, k_1 \in O(1)$ with the following property. For any choices of $o \in \mathbb{Z}$, $b \in \{0,1\}$, and $L_{\max} \in \mathbb{Z}_{>0}$, it holds that*

$$|\{L \in \mathbb{N}, L \leq L_{\max} \mid [k_b \cdot L, k_b \cdot L + R] \cap \{k_{1-b} \cdot L' + o \mid L' \in \mathbb{Z}\} \neq \emptyset\}| \leq \varepsilon L_{\max} + 1.$$

Proof. Let $k_0 := \lceil (R+1)/\varepsilon \rceil$ and $k_1 := k_0 + R + 1$. Let $L \in \mathbb{N}$ be minimal with the property that

$$[k_b \cdot L, k_b \cdot L + R] \cap \{k_{1-b} \cdot L' + o \mid L' \in \mathbb{Z}\} \neq \emptyset.$$

By the choices of $k_b, b \in \{0,1\}$, it follows that for $L+1, L+2, \dots, L + \lceil 1/\varepsilon \rceil - 1$ the intersection is empty. Repeating this argument inductively, the claim follows. □

We are now ready to resolve the choices of constants that guarantee an unimpeded instance $O(f+1)$ rounds after one of the recursive counting instances and both filtering subroutines have stabilized.

Lemma 5.6. *If $R \in O(1)$, there are constants X, k_0, k_1 with the following property. If for $b \in \{0,1\}$ it holds that $f < |V_b|/3$, there is an unimpeded instance within $\max\{S^b_C + S^b_F, S^{1-b}_F\} + O(f+1)$ rounds.*

Proof. By Lemma 5.1, the variables F^b_v count from round $S^b_C + S^b_F$. By Lemma 5.2, there is a virtual clock $\hat{F}^{1-b}$ such that for each $i \in \mathbb{N}$ and $r \in I_i := [S^{1-b}_F + iX + 1, S^{1-b}_F + (i+1)X - 1]$, it holds that $\hat{F}^{1-b}_r = \hat{F}^{1-b}_{r-1} + 1 \bmod k_b n$ and for each correct node v, we have that $F^{1-b}_{r,v} \in \{\hat{F}^{1-b}_r, \bot\}$ for all $r \geq S^{1-b}_F$. We apply Lemma 5.5 as follows:

- Fix k_0, k_1 as given by the lemma for $\varepsilon = 1/4$ and the given value of R.
- Fix $X = 5\max\{k_0, k_1\} + R$ and $L_{\max} := \lfloor (X - R)/k_b \rfloor - 1 > 4$.
- Fix $i \in \mathbb{N}$ such that $S^{1-b}_F + iX \geq S^b_C + S^b_F$. Let $r_b \in I_i$ be minimal such that there is $\ell_b \in V$ with $F^b_{v,r_b} = k_b \ell_b \bmod k_b n$ for some correct node v.
- Let $r_{1-b} \in I_i$ be minimal such that there is $\ell_{1-b} \in V$ with $\hat{F}^{1-b}_{r_{1-b}} = k_{1-b}\ell_{1-b} \bmod k_{1-b} n$ for some correct node v. Choose $o := r_{1-b} - r_b$.

Now consider a leader $\ell \in \{\ell_b, \ell_b + 1, \dots, \ell_b + L_{\max}\}$, where for convenience we identify values $\ell > n$ with the node v such that $v = \ell \bmod n$. Observe that $r_b + (\ell - \ell_b)k_b \geq r_b$ and

$$r_b + (\ell - \ell_b)k_b + R \leq r_b + L_{\max} k_b + R \leq r_b + X - k_b \leq S^{1-b}_F + (i+1)X - 1,$$

where the last step uses that r_b is one of the first k_b rounds of I_i. That is, we have that $J_\ell := \{r_b + (\ell - \ell_b)k_b, r_b + (\ell - \ell_b)k_b + 1, \dots, r_b + (\ell - \ell_b)k_b + R - 1\} \subset I_i$.

Recall that the variables F^b_v count from round $S^b_C + S^b_F$, which due to our choice of i is no later than the last round before I_i. Hence, the b-th instance of king consensus initialized in round $r_b + (\ell - \ell_b)k_b - 1$ runs with leader ℓ, where the substraction of 1 takes into account that ℓ^b_v is determined based on $F^b_{v,r-1}$ in round r, i.e., before F^b_v is updated by the filtering subroutine. Because $k_b > R$, each correct node v initializes its b-th instance in rounds $r \in J_\ell \setminus \{r_b + (\ell - \ell_b)k_b\}$ with $\ell_v = \bot$. We conclude that the instance is unimpeded if and

only if (i) ℓ is correct and (ii) no correct node v uses input $\ell_v \neq \bot$ on its $(1-b)$-th instance in some round $r \in J_\ell$.

Let $i_0 \in \mathbb{N}$ be minimal such that (i) $S_F^{1-b} + i_0 X + 1 \geq S_C^b + S_F^b$, (ii) the variables F_v^b do not "overflow" during I_{i_0}, i.e., $F_{v,r}^b = F_{v,r-1}^b + 1$ for all $r \in I_{i_0}$, and (iii) fewer than half of the leaders $\{\ell_b, \ell_b + 1, \dots, \ell_b + L_{\max}\}$ for the interval I_i are faulty. Since there are at most f faulty nodes and $n > 3f$ (i.e., we do not run out of "fresh" leaders before exhausting the fault budget),[8] we have that $i_0 \in \max\{(S_C^b + S_F^b - S_F^{1-b})/X, 0\} + O(f+1)$. By our choices of ε, k_0, k_1, and o, Lemma 5.5 guarantees that out of the remaining at least $L_{\max}/2$ correct leaders ℓ, all but $L_{\max}/4 + 1$ satisfy that

$$[k_b(\ell - \ell_b), k_b(\ell - \ell_b) + R] \cap \{k_{1-b} \cdot L + r_{1-b} - r_b \mid L \in \mathbb{Z}\} = \emptyset.$$

By our choice of $L_{\max}$, $L_{\max}/2 - L_{\max}/4 - 1 > 0$, i.e., some correct ℓ satisfying the above constraint exists.

We claim that the corresponding b-th instance initiliazed in round $r_b + (\ell - \ell_b)k_b$ is unimpeded. Given that ℓ is correct, it remains to show that no correct node v initilializes a $(1-b)$-th instance with $\ell_v \neq \bot$ in some round $r \in J_\ell$. Because $F_{v,r}^{1-b} \in \{\hat{F}_r^{1-b}, \bot\}$, this is only possible if $\hat{F}_r^{1-b} \bmod k_{1-b} = 0$.

Accordingly, assume for contradiction that $\hat{F}_r^{1-b} \bmod k_{1-b} = 0$ for some $r \in J_\ell$. Recall that $\hat{F}_{r'}^{1-b} = \hat{F}_r^{1-b} + r' - r \bmod k_{1-b} n$ for all $r, r' \in I_i$ and that $\hat{F}_{r_{1-b}}^{1-b} = 0 \bmod k_{1-b}$. Therefore,

$$0 = \hat{F}_r^{1-b} \bmod k_{1-b} = \hat{F}_{r_{1-b}}^{1-b} + r - r_{1-b} \bmod k_{1-b} = r - r_{1-b} \bmod k_{1-b},$$

i.e., $r - r_b \in \{k_{1-b} \cdot L + r_{1-b} - r_b \mid L \in \mathbb{Z}\}$. On the other hand, by our choice of i_0,

$$F_{v,r}^b - F_{v,r_b}^b = r - r_b \in [k_b(\ell - \ell_b), k_b(\ell - \ell_b) + R].$$

We reach the contradiction that $r - r_b$ is in the intersection of these two sets, which by Lemma 5.5 is empty. We conclude that the instance is unimpeded, as claimed. Since $i_0 \in \max\{(S_C^b + S_F^b - S_F^{1-b})/X, 0\} + O(f+1)$, this instance occurs by round $\max\{S_C^b + S_F^b, S_F^{1-b}\} + O(f+1)$. □

We arrive at the following theorem.

Theorem 5.7. *If $R \in O(1)$, there are constants k_0, k_1, and X such that Algorithm 3 solves C-counting with stabilization time $\max_{b\in\{0,1\}}\{S_C^b + S_F^b\} + O(f+1)$. If $f < |V_b|/3$ for some $b \in \{0,1\}$, then the stabilization time is $\max\{S_C^b + S_F^b, S_F^{1-b}\} + O(f+1)$.*

Proof. We show the second statement first. By Lemma 5.6 and the assumptions of the theorem, there is an unimpeded instance of king consensus within $\max\{S_C^b + S_F^b, S_F^{1-b}\} + O(f+1)$ rounds. By Lemma 5.4, the output variables then start to count from some round $r \in \max\{S_C^b + S_F^b, S_F^{1-b}\} + O(f+1)$.

Regarding the first statement, recall that $f < n/3$ and V_0 and V_1 partition $V = \{1, \dots, n\}$, so there must be some $b \in \{0,1\}$ so that $f < |V_b|/3$. Since $\max\{S_C^b + S_F^b, S_F^{1-b}\} \leq \max_{b\in\{0,1\}}\{S_C^b + S_F^b\}$, the claim readily follows from the second statement. □

In the following, we will tacitly assume that the constants in Algorithm 3 are chosen in accordance with Theorem 5.7 for the $R \in O(1)$ given by the choice of the king consensus algorithm employed. Applying the recursive scheme with the trivial solution on single nodes, we arrive at the following result.

Corollary 5.8. *C-counting can be solved with stabilization time $O(f + 1 + \log n)$, where each correct node sends $O(\log C + \log^2 n)$ bits to each other node in each round.*

Proof. By Lemmas 2.3 and 2.7, there are filtering and king consensus protocols that run in a constant number of rounds and use messages of size $O(\log C)$ and $O(\log |\mathcal{V}|)$, respectively. We recursively apply Algorithm 3, where we partition the node set as evenly as possible in each recursion step and the base case of $n = 1$ is trivial. In each step of the recursion, either V_0 or V_1 contains at most half of the faulty nodes. Hence, summing over the at most $\lceil \log n \rceil$ steps along the respective root-leaf path in the recursion tree, the stabilization time is bounded by $\sum_{i=0}^{\lceil \log n \rceil - 1} O(2^{-i} f + 1) = O(f + 1 + \log n)$. The message size bound follows by observing that (i) each node participates in $O(\log n)$ recursive instances, each running a constant number of concurrent instances of filtering and king consensus, and (ii) all of these instances but the top-level king consensus instances, which have message size $O(\log C)$, use messages of size $O(\log n)$. □

6 Open Questions

We conclude the paper by listing a number of open questions we consider to be of interest.

(1) Is the achieved bit complexity (nearly) optimal?
(2) Is it possible to interleave our approach with randomized techniques to get both stabilization time $O(f+1)$ deterministically and $O(1)$ in expectation?
(3) Can the proposed or similar techniques be applied to the pulse synchronization problem to achieve early stabilization also there?
(4) Cryptographic tools can enable to increase t to $\lfloor (n-1)/2 \rfloor$ and circumvent the Dolev-Reischuk bound, e.g. by means of threshold signatures [6]. Can higher resilience or lower communication cost be achieved while maintaining early stabilization?
(5) Addressing the previous question requires to reconcile the total loss of volatile state and arbitrary transient violation of the protocol assumed in the context of self-stabilization with the need to hide secret keys and signatures from the adversary. Can this be achieved under reasonable assumptions? And if yes, how should this be modeled?

Acknowledgments

This work is funded by the European Union, ERC-2023-STG, Project ID: 101116713. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them. Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972.

[8] Due to condition (ii), this applies as-is only if n is a sufficiently large constant. However, if $n = O(1)$, we may work with clocks that count modulo $mk_b n$ for a sufficiently large constant m instead.

References

[1] Ittai Abraham, T.-H. Hubert Chan, Danny Dolev, Kartik Nayak, Rafael Pass, Ling Ren, and Elaine Shi. 2019. Communication Complexity of Byzantine Agreement, Revisited. In *Symposium on Principles of Distributed Computing (PODC)*, Peter Robinson and Faith Ellen (Eds.). ACM, 317–326.

[2] B. Awerbuch and G. Varghese. 1991. Distributed Program Checking: a Paradigm for Building Self-stabilizing Distributed Protocols. In *Symposium of Foundations of Computer Science (FOCS)*. 258–267.

[3] Michael Ben-Or, Danny Dolev, and Ezra N. Hoch. 2008. Fast self-stabilizing byzantine tolerant digital clock synchronization. In *Symposium on Principles of Distributed Computing (PODC)*, Rida A. Bazzi and Boaz Patt-Shamir (Eds.). ACM, 385–394.

[4] P. Berman, J.A. Garay, and K.J. Perry. 1989. Towards Optimal Distributed Consensus. In *Symposium on Foundations of Computer Science (FOCS)*. 410–415.

[5] Benny Chor, Michael Merritt, and David B. Shmoys. 1989. Simple constant-time consensus protocols in realistic failure models. *J. ACM* 36, 3 (July 1989), 591–614.

[6] Yvo Desmedt and Yair Frankel. 1989. Threshold Cryptosystems. In *Advances in Cryptology (CRYPTO)*, Gilles Brassard (Ed.), Vol. 435. Springer, 307–315.

[7] Danny Dolev. 1982. The Byzantine Generals Strike Again. *Journal of Algorithms* 3, 1 (1982), 14–30.

[8] Danny Dolev and Rüdiger Reischuk. 1985. Bounds on Information Exchange for Byzantine Agreement. *J. ACM* 32, 1 (Jan. 1985), 191–204.

[9] Danny Dolev, Ruediger Reischuk, and H. Raymond Strong. 1982. 'Eventual' is Earlier than 'Immediate'. In *Symposium on Foundations of Computer Science (SFCS)*. 196–203.

[10] Danny Dolev, Ruediger Reischuk, and H. Raymond Strong. 1990. Early stopping in Byzantine agreement. *J. ACM* 37, 4 (1990), 720–741.

[11] Shlomi Dolev and Jennifer L. Welch. 2004. Self-stabilizing clock synchronization in the presence of Byzantine faults. *J. ACM* 51, 5 (Sept. 2004), 780–799.

[12] Pesech Feldman and Silvio Micali. 1997. An Optimal Probabilistic Protocol for Synchronous Byzantine Agreement. *SIAM J. Comput.* 26, 4 (1997), 873–933.

[13] Michael J. Fischer and Nancy A. Lynch. 1982. A Lower Bound for the Time to Assure Interactive Consistency. *Inform. Process. Lett.* 14, 4 (1982), 183–186.

[14] Pankaj Khanchandani and Christoph Lenzen. 2019. Self-Stabilizing Byzantine Clock Synchronization with Optimal Precision. *Theory Comput. Syst.* 63, 2 (2019), 261–305.

[15] Christoph Lenzen, Matthias Függer, Markus Hofstatter, and Ulrich Schmid. 2013. Efficient Construction of Global Time in SoCs Despite Arbitrary Faults. In *Euromicro Conference on Digital System Design (DSD)*. IEEE Computer Society, 142–151.

[16] Christoph Lenzen and Julian Loss. 2026. Early-Stabilizing Counting. arXiv:2605.18171 [cs.DC] https://arxiv.org/abs/2605.18171

[17] Christoph Lenzen and Joel Rybicki. 2019. Near-optimal self-stabilising counting and firing squads. *Distributed Comput.* 32, 4 (2019), 339–360.

[18] Christoph Lenzen and Joel Rybicki. 2019. Self-Stabilising Byzantine Clock Synchronisation Is Almost as Easy as Consensus. *J. ACM* 66, 5, Article 32 (Aug. 2019).

[19] Christoph Lenzen, Joel Rybicki, and Jukka Suomela. 2017. Efficient Counting with Optimal Resilience. *SIAM J. Comput.* 46, 4 (2017), 1473–1500.

[20] M. Pease, R. Shostak, and L. Lamport. 1980. Reaching Agreement in the Presence of Faults. *J. ACM* 27, 2 (April 1980), 228–234.