A Recursive Early-Stopping Phase King Protocol

Christoph Lenzen
CISPA Helmholtz Center for Information Security
Germany
lenzen@cispa.de

Sahar Sheikholeslami
Ferdowsi University of Mashhad
Iran
seslami2@gmail.com

ABSTRACT

Early-stopping consensus protocols guarantee termination within a number of rounds that depends only on the actual number f of faulty nodes in a run, not the maximum number of faults that can be tolerated. We consider early-stopping deterministic synchronous consensus with Byzantine faults in a fully connected message passing system of n nodes. Many such protocols are known, but so far none combine early-stopping in $O(f+1)$ rounds with optimal resilience and a bit complexity of $o(n^2(f+1))$.

We provide two solutions to the above problem. The first is fairly simple and almost matches the above goals, but has worst-case message and bit complexities of $\Theta(n^2 \log(f+2))$. The second reduces the bit complexity further to $O(n^2)$ by calling itself recursively at most twice on $\Theta(n)$-sized subsets. The result is the first protocol that is simultenously optimally resilient, asymptotically optimally early-stopping, and asymptotically bit- and message-optimal.

CCS CONCEPTS

• **Theory of computation → Distributed algorithms**.

KEYWORDS

consensus, Byzantine faults, bit complexity

ACM Reference Format:
Christoph Lenzen and Sahar Sheikholeslami. 2022. A Recursive Early-Stopping Phase King Protocol. In *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing (PODC '22), July 25–29, 2022, Salerno, Italy*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3519270.3538425

1 INTRODUCTION & RELATED WORK

Consensus, the task of reaching agreement among the participants of a distributed system despite faults, is arguably the most fundamental fault-tolerance primitive. As a result, over the course of half a century researchers generated over half a million pages[1] of research articles related to this topic; see [20] for general treatments of the topic from a theory perspective. In this work, we study the bit complexity of asymptotically optimally *early-stopping* consensus, in which the time until termination is required to be a function of the actual number of faults in the execution, as opposed to the maximum number of faults than can be tolerated.

In more detail, we consider the following setting:

- binary consensus, i.e., inputs and outputs from $\{0,1\}$ (multi-valued protocols can be derived without asymptotic overhead [19]);
- Byzantine (i.e., worst-case) faults;
- message passing with authenticated channels, i.e., the receiver of a message is aware of the sender's identity;
- synchronous communication, i.e., coordinated send-receive-compute rounds;
- optimal resilience, i.e., tolerance of $t = \lceil n/3 \rceil - 1$ faults in an n-node system [22] (smaller t can be handled by running the protocol on $3t+1$ nodes and broadcasting the output to all nodes);
- full connectivity (general networks can simulate this if and only if they are $(2t+1)$-connected [20]);
- known node identifiers, w.l.o.g. from 1 to n;
- deterministic algorithms (i.e., no restrictions on the adversary apart from the limit on faults); and
- asymptotically optimal early-stopping, i.e., if $0 \leq f \leq t$, then correct nodes terminate within $O(f+1)$ rounds.

Our goal is to determine the message and bit complexity of this task,[2] i.e., the total number of messages and bits that need to be sent by an algorithm solving consensus in this setting.

Not only are these quality metrics of general interest, but prior work motivates this specific setup:

(1) The Phase King protocol [1] has an early-stopping variant [3]. It matches the above constraints, but has message and bit complexity $\Theta(nt + t^2 f)$.
(2) The Phase King protocol has also a recursive variant [2], which has asymptotically optimal message and bit complexity of $\Theta(nt)$ [14], but is not early-stopping.
(3) It has been shown that optimal early-stopping necessitates $\Omega(nt + t^2 f)$ messages [13], i.e., in general optimal early-stopping is incompatible with asymptotically optimal message and bit complexity.

In other words, we set out to answer the question

> "Does a relaxed early-stopping requirement enable us to achieve bit complexity $O(nt)$?"

Our Contribution

We present two novel algorithms for the task. The first falls out of a straightforward combination of known techniques. The reason why

[1] This conservative lower bound was generated by multiplying the more than 50 000 hits Google Scholar generated for "consensus algorithm" (with quotes) by 10.

[2] In the following, we adopt the convention to state all bounds assuming that protocols are executed on $3t+1$ nodes. Once the protocol is complete, these nodes broadcast their output to any non-participating nodes, which decide on the value supported by (at least) $t+1$ participating nodes. Since $\Omega(nt)$ messages must be sent [14], this does not affect asymptotic bounds.

PODC '22, July 25–29, 2022, Salerno, Italy

ACM ISBN 978-1-4503-9262-4/22/07.
https://doi.org/10.1145/3519270.3538425

the classic Phase King protocol [1] has bit and message complexity of $\Theta(nt + t^3)$ is that it establishes agreement over $t + 1$ phases by executing a subroutine with a dedicated king that guarantees validity and establishes agreement if the king is correct. Each time this requires a constant number of broadcasts per node, causing $\Theta(t^3)$ messages. To avoid this, the recursive Phase King protocol [2] calls itself twice on roughly half of the nodes, replacing the king by running consensus on each subset. Since the cost of each iteration is quadratic, the two recursive calls in sum cause fewer bits to be sent then the calling routine, resulting in bit complexity $O(nt)$.

Applying the same idea, our first solution calls the recursive Phase King protocol as subroutine replacing the king. However, to ensure that the algorithm is early-stopping, we start from a constant-sized set and increase the number of nodes in each iteration by a factor of 2. Analogously to the early-stopping Phase King algorithm, after each call the algorithm tries to terminate, which succeeds as soon as a subset had fewer than one third of its nodes being faulty. Consequently, the bit complexity of this approach is also dominated by the broadcasts of the top-level routine. Because with f faults $\Omega(\log f)$ subroutine calls might fail due to having too many faulty nodes, this results in a worst-case bit complexity of $\Theta(nt \log(f+2))$.

Theorem 1.1. *Algorithm 4 solves binary consensus with up to $t := \lceil n/3 \rceil - 1$ Byzantine faults in $O(f + 1)$ rounds, where $0 \leq f \leq t$ is the number of actual faults. Its bit complexity is $O(nt \log(f + 2))$.*

To remove this $\Theta(\log(f + 2))$ overhead, our second approach performs at most two recursive calls on sets of size $\Theta(t)$, just like the recursive Phase King algorithm. Unfortunately, this disallows to allocate a predetermined number of rounds to spend on such a recursive call: in a run with $O(1)$ faults only $O(1)$ rounds can be spent, while the recursive call must tolerate $\Omega(t)$ faults and hence potentially run longer. We deal with this by having nodes in the calling routine, which is guaranteed to have sufficiently many correct nodes, vote on when to proceed. This approach is borrowed from a classic pulse synchronization algorithm [25], in which $n - f$ votes are required to generate a pulse, while $f+1$ votes are sufficient to make nodes vote for the next pulse themselves. Thus, as soon as any correct node generates a pulse, $n - 2f > f$ nodes must already have voted in favor, causing all remaining correct nodes to add their votes and reach the threshold to generate a pulse. In our synchronous setting, this ensures that all correct nodes proceed to continue from the subroutine call within one round of each other. A simple simulation strategy allows us to nonetheless pretend that our protocol runs in perfect synchrony, provided we maintain the invariant that nodes are within one round of each other: we append a bit distingishuing between even and odd round messages and have each node idle for one round between communication steps to collect messages that arrive "late" by one round. Finally, we start our protocol by a single "standard" Phase King iteration with a single king, to avoid running for $\Theta(\log n)$ rounds even when there are no faults. This results in a protocol which is optimally resilient and asymptotically optimal in its early-stopping guarantee and communication complexity.

Theorem 1.2. *Algorithm 8 solves binary consensus with up to $t := \lceil n/3 \rceil - 1$ Byzantine faults in $O(f + 1)$ rounds, where $0 \leq f \leq t$ is the number of actual faults. Its bit complexity is $O(nt)$.*

Further Related Work

To limit the scope of this literature review, we do not discuss randomized protocols. These require varying restrictions on the adversarial model and, due to achieving much smaller running times in the presence of many faults, usually are not studied with respect to their early-stopping properties. However, one should take of note the possibility to interleave randomized and deterministic early-stopping algorithms for the sake of achieving both small running times with a high probability against many faults, while guaranteeing early termination in case of few faults.

In [22], it was shown that consensus can be solved in $t+1$ rounds, if at most $t < n/3$ nodes can be Byzantine faulty. Under cryptographic hardness assumptions, the resilience can be increased to (trivially optimal) $t < n/2$ [15]. In general networks, in addition to the above constraint t must be smaller than the node connectivity. Under crash faults or cryptographic hardness assumptions, this is also sufficient, since trivially or using signatures full connectivity can be simulated. With Byzantine faults and without cryptography, it is necessary and sufficient for t to be smaller than half the node connectivity (for Byzantine faults) [12]. These results are not concerned with the running time. For crash faults, [9] provides an algorithm that solves consensus under f crash faults in $O(f + D_f)$ rounds, where D_f is the worst-case diameter of the network obtained by removing up to f nodes.

In [23], consensus studied in weaker fault models. Toueg and Chandra [6] introduced the concept of early-stopping algorithms, in the context of such weaker fault models. They solve reliable broadcast using $O(fn)$ messages in $f + 2$ rounds under crash failures and in $2f + 3$ rounds under omission failures, the latter meaning that faulty nodes might drop messages instead of sending or receiving them. In [18], an lower bound of $f + 2$ on the round complexity of uniform agreement for every uniform consensus algorithm tolerating $f \leq t - 1$ faults is given; uniform consensus requires that faulty nodes do not decide on a different value than correct ones, while "plain" consensus requires agreement only from correct nodes. Continuating this work, in [21], time-optimal protocols with running time $\min\{f + 2, t + 1\}$ and message complexity $O(n^2 f)$ are presented for uniform consensus under crash and omission faults. An early-stopping protocol for k-set agreement, where the agreement condition is relaxed to permitting k different outputs at correct nodes (and validity requires them to be inputs of some nodes), is presented in [24]. This protocol tolerates $t < n/2$ omission faults, i.e., nodes that might drop messages. In this protocol, not only correct nodes, but all nodes that do neither crash nor drop messages they should receive terminate by round $\min(\lfloor f/k \rfloor + 2, \lfloor t/k \rfloor + 1)$. Finally, [5] refined notion of time complexity further by presenting an *unbeatable* predicate for early decision, in that any predicate that decides earlier at some node in some execution, it must decide later in some other execution.

Concerning communication-efficient protocols, Coan and Welch [10] show that (under all fault models) consensus can be solved in $(1 + \varepsilon)t$ rounds communicating $O(t^{2/\varepsilon} + nt)$ bits, for $\varepsilon > 0$. In [11], they reduce the bit complexity asymptotically optimal $O(nt)$ bits complexity using message size t^ε and $t + o(t)$ rounds. Note that this improves over the running time of the Phase King protocol [1] at the expense of non-constant message size. In [17], an algorithm

solving consensus with crash failures using asymptotically optimal $O(n)$ messages and $O(t^{1+\varepsilon})$ rounds is given, for any constant $0 < \varepsilon < 1$. An early-stopping algorithm for crash faults with message complexity $O(n + fn^{\varepsilon})$ and running time $O((f+1)8^{1/\varepsilon})$ is also presented in this paper. The message size in these protocols is $O(n)$. In [7], the running time overhead is reduced to a constant with $O(n \log^2 n)$ messages and message size $O(n \log n)$, for $n - t \in \Omega(n)$. In [8], they extend their techniques to obtain an early-stopping algorithm solving consensus in $O(f+1)$ rounds using $O(n \log^5 n)$.

Organization of this Article

We start by revisiting the early-stopping Phase King algorithm in Section 2, where we break it down into smaller subtasks and their solutions. In Section 3, we rearrange these, replacing the eponymous King's Broadcast by a call to the recursive Phase King algorithm. Finally, in Section 4, we develop our recursive early-stopping algorithm. Here, we also show how to efficiently simulate perfect synchrony when nodes might have off-by-one round counters, such that using a voting barrier (rather than consensus) is sufficient to deal with unknown termination time of recursive calls.

2 EARLY-STOPPING PHASE KING, BROKEN DOWN & REASSEMBLED

In this section we revisit the Early-Stopping Phase King algorithm [3] and break it down into subroutines for later reuse. While we slightly deviate from the original protocol, all of these building blocks and their analysis are well known; we merely restate these results in our own terms. We define subtasks the subroutines solve, from which we can readily infer the correctness of the overall algorithm. In later sections, we will provide novel algorithms by rearranging the subroutines and providing new implementations of the *electors' broadcast* subtask.

First, let us recall the input-output specification of binary consensus, where $V_g \subseteq V$ denotes the set of correct nodes.

Definition 2.1 (Binary consensus). Each node $i \in V$ receives an input $b_i \in \{0,1\}$ and computes an output $o_i \in \{0,1\}$. The outputs of correct nodes satisfy the following constraints.

validity: If $b_i = b \in \{0,1\}$ for all $i \in V_g$, then $o_i = b$ for all $i \in V_g$.

agreement: For all $i, j \in V_g$ it holds that $o_i = o_j$.

We remark that traditionally termination is listed as a third requirement. Throughout this work, we adopt the convention that all routines terminate at a node when it computes it output.

In pseudocode, we use a number of further conventions that simplify the code and can easily be realized by simple checks and/or calculations.

(1) In each round, each node accepts from each sender only a single message of the form the algorithm specifies for this round and discards any other message(s).
(2) When nodes broadcast a message, they also receive the message themselves and process them like other senders' messages.
(3) If $i \in V_g$ terminates with output $b \in \{0,1\}$ and communicates this to other nodes, they will assume that i sends b in all future rounds.
(4) Statements of the form "if received at least x times $b \in \{0,1\}$ do $y(b)$" are executed for an arbitrary $b \in \{0,1\}$ if the condition is satisfied for both 0 and 1 (except for Algorithm 8, where the behavior is explicitly specified).
(5) If an algorithm or subroutine is called on node set V', then $V' = \{1, \dots, |V'|\}$. Since node identifiers are known and which nodes participate in a recursive call is pre-determined, each node can easily perform the necessary mapping of identifiers in our algorithms.
(6) If a (sub)routine terminates at a node executing it, the node also terminates all subroutines that have been called by the terminating routine.
(7) Any guarantees hold only if strictly fewer than one third of the nodes executing a routine are faulty. Otherwise, no guarantee on the behavior is given, except that correct nodes stick to sending only correctly formatted messages and only to the node set on which this instance of the algorithm runs when specified.

We describe the early-stopping Phase King protocol, given in Algorithm 1, in terms of three subroutines.

Algorithm 1: Early-stopping Phase King algorithm at node $i \in V_g$, where $b_i \in \{0,1\}$ is the node's input.

```
1 op_i := b_i
  // loop terminates by reaching correct king j
2 for j = 1 to t + 1 do
      // check whether changing opinion is valid
3     (op_i, strong_i) := validator(op_i)
      // adopt king's value if valid
4     op_i := electorsBC(op_i, strong_i, {j})
      // check for termination
5     (op_i, term_i) := termcheck(op_i)
6     if term_i = 1 then
          // termination broadcast
7         broadcast op_i
          // terminate
8         return op_i
```

Weak Validator

The first subroutine,[3] *weak validator,* aims to test for pre-existing agreement. It is allowed to err on the side of caution, in that some nodes might end up believing that the inputs agree even if that is not the case. However, it guarantees that correct nodes can do so only for one $b \in \{0,1\}$, which then all nodes adopt as output value.

Definition 2.2 (Weak validator). In validator(b_i), each node $i \in V$ receives an input $b_i \in \{0,1\}$ and computes output $(o_i, \text{strong}_i) \in \{0,1\}^2$. Correct nodes' outputs satisfy the following constraints.

validity: If $b_i = b \in \{0,1\}$ for all $i \in V_g$, then $o_i = b$ and $\text{strong}_i = 1$ for all $i \in V_g$.

[3] We stress that the subroutines in this section are not novel building blocks. For example, both the weak validator and the termination check are instances of *Graded Broadcast* [16]. However, our goal is to give intuition on the conceptional building blocks of our algorithms, prompting us to use context-specific terms.

weak agreement: If $\text{strong}_i = 1$ for some $i \in V_g$, then $o_j = o_i$ for all $j \in V_g$.

A weak validator is implemented by a 2-round algorithm involving two rounds of broadcast, see Algorithm 2.

Algorithm 2: Weak validator algorithm at $i \in V_g$, where b_i is the input value of i.

```
1  o_i := b_i
2  broadcast b_i                                  // first broadcast
3  if received at least n − t times b ∈ {0,1} then
4  |  broadcast b                                 // second broadcast
5  strong_i := 0
6  if received at least t + 1 times b ∈ {0,1} then
7  |  o_i := b
8  if received at least n − t times b ∈ {0,1} then
9  |  strong_i := 1
10 return (o_i, strong_i)
```

Lemma 2.3. *Algorithm 2 satisfies the validity condition of Definition 2.2.*

Proof. Suppose that $b_i = b \in \{0,1\}$ for all $i \in V_g$. Hence, each $i \in |V_g|$ broadcasts b in the first round and all nodes receive at least $n-t$ times b and at most t times $1-b$. Because $n > 3t$, $|V_g| \geq n-t > t$. Therefore, each $i \in |V_g|$ again broadcasts b in the second round and receives at least $n-t$ times b and at most t times $1-b$. We conclude that all $i \in V_g$ output $(1, b)$, as required. □

Lemma 2.4. *Algorithm 2 satisfies the weak agreement condition of Definition 2.2.*

Proof. Suppose that $i \in V_g$ outputs $(1, b)$ for $b \in \{0,1\}$. Thus, i received $n-t$ times b in the second round. Because $n > 3t$, at least $n - 2t > t$ nodes from V_g must have sent b. Therefore, each $j \in V_g$ sets $o_j := b$, provided that j does not receive $1-b$ from more than t nodes in the second broadcast.

We show this by establishing that no correct node sends $1-b$ in the second broadcast. To see this, observe that each of the correct nodes sending b in the second broadcast received b from at least $n-t$ nodes in the first round. If f nodes are faulty, then at least $n-t-f$ of these are from V_g. Since correct nodes broadcast only one value, no node receives more than $f + |V_g| - (n-t-f) = f+t \leq 2t < n-t$ times $1-b$ in the first round. Accordingly, no correct node sends $1-b$ in the second round, as claimed. □

Electors' Broadcast

The second subroutine we refer to as electors' broadcast. In the early-stopping Phase King protocol, it is implemented by having a "phase king," but we later replace the single king by larger groups of nodes, the electors.

Definition 2.5 (Electors' broadcast). In $\text{electorsBC}(b_i, \text{strong}_i, K)$, each node $i \in V$ receives input $(b_i, \text{strong}_i) \in \{0,1\}^2$ and computes output $o_i \in \{0,1\}$. Moreover, there is a set $K \subseteq V$ of nodes that are designated *electors*. The outputs of correct nodes satisfy the following constraints.

validity: If $b_i = b \in \{0,1\}$ and $\text{strong}_i = 1$ for all $i \in V_g$, then $o_i = b$ and $\text{strong}_i = 1$ for all $i \in V_g$.
electors' agreement: If $|V_g \cap K| > \lfloor 2|K|/3 \rfloor$ and the inputs satisfy weak agreement, then $o_i = o_j$ for all $i, j \in V_g$.

A simple implementation of electors' broadcast with $K = \{j\}$ for $j \in V_g$ is given by the eponymous "king's broadcast."

Algorithm 3: King's broadcast with king j at node $i \in V_g$, where (b_i, strong_i) is the input of i.

```
1 o_i := b_i
2 if i = j then
3 |  broadcast b_i                          // king's broadcast
4 if strong = 0 and received b ∈ {0,1} from j then
5 |  o_i := b
6 return o_i
```

Lemma 2.6. *Algorithm 3 implements electors' broadcast for $K = \{j\}$.*

Proof. Validity is immediate from the fact that each node returns b_i when $\text{strong}_i = 1$. Regarding electors' agreement, observe that $|V_g \cap K| > \lfloor 2|K|/3 \rfloor$ simply means that $j \in V_g$. Thus, each correct node with strong = 0 adopts value b_j. Recalling that weak agreement of the inputs entails that if any $i \in V_g$ has $\text{strong}_i = 1$, then $b_j = b_i$, electors' agreement follows. □

Termination Check

The third and final building block of the early-stopping Phase King protocol is the one that makes it early-stopping. It implements a termination check that, if passed at some correct node, ensures that agreement is established at all nodes that did not terminate yet. This results in termination of the remaining nodes in the next iteration of the overall algorithm.

Definition 2.7 (Termination check). In a call to $\text{termcheck}(b_i)$, each node $i \in V$ receives input $b_i \in \{0,1\}$ and computes output $(o_i, \text{term}_i) \in \{0,1\}^2$. The outputs of correct nodes satisfy the following constraints.

validity: If $b_i = b \in \{0,1\}$ for all $i \in V_g$, then $o_i = b$ and $\text{term}_i = 1$ for all $i \in V_g$.
termination agreement: If $\text{term}_i = 1$ for some $i \in V_g$, then $o_j = o_i$ for all $j \in V_g$.

In the original formulation of the early-stopping Phase King protocol, this step is merged with the King's broadcast, but for modularity and convenience of presentation we seperate this task here. Our implementation, which we will be able to reuse for our novel algorithms, is essentially a synchronous version of the termination test in the first randomized asynchronous consensus routine due to Bracha [4] – but as mentioned earlier, it can also be seen as an instance of Graded Broadcast [16].

More specifically, note that Definition 2.2 and Definition 2.7 are essentially identical. The only difference that strong_i has been relabeled as term_i, as we seek to decide on termination rather than whether we will ignore the input from the electors. We conclude that Algorithm 2 also implements Definition 2.7.

Corollary 2.8. *Algorithm 2 satisfies the validity and termination agreement conditions of Definition 2.7.*

Analysis of the Early-Stopping Phase King Protocol

Analyzing Algorithm 1 now is straightforward.

Theorem 2.9. *Suppose that the subroutine calls in Algorithm 1 are implemented by Algorithms 2 and 3. Then the resulting algorithm solves binary consensus in the presence of up to $t := \lceil n/3 \rceil - 1$ Byzantine faults. It runs for at most $6(f+1)$ rounds and has correct nodes in total send at most $6n^2(f+1)$ single-bit messages.*

Proof. By Lemmas 2.3, 2.4 and 2.6 and Corollary 2.8, the subroutines satisfy the specifications given in Definitions 2.2, 2.5 and 2.7. Hence, if $\mathrm{op}_i = b \in \{0,1\}$ for all $i \in V_g$ at the beginning of a loop iteration, the validity conditions of the subtasks ensure that no correct node ever changes $\mathrm{op}_i = b$ and that they terminate with output b in this loop iteration. In particular, this proves the validity condition of binary consensus. Note also that, due to the convention that terminated nodes are assumed to resend their output on each broadcast, this applies even when a subset of the nodes in V_g already terminated with output b in an earlier loop iteration.

Next, we establish that termination occurs within $6(f+1)$ rounds. To see this, observe that there is some $j \in \{1, \dots, f+1\}$ such that $j \in V_g$ and consider the j-th loop iteration. Since the outputs of the call to validator(op_i) satisfy weak agreement, by electors' agreement the call to electorsBC$(\mathrm{op}_i, \mathrm{strong}_i, j)$ will result in all correct nodes agreeing on some output $b \in \{0,1\}$. Hence, by validity of the call to termcheck(op_i), all correct nodes terminate by the end of this loop iteration. Because each loop iteration has 6 rounds—two times two for Algorithm 2, one for Algorithm 3, and one for broadcasting the termination value—the running time bound follows. The bound on the number of messages sent by correct nodes trivially follows, and the fact that they are single-bit messages is immediate from the pseudocode listings.

It remains to show the agreement condition of binary consensus. To this end, suppose that $i \in V_g$ is one of the first terminating nodes, with output o_i. Thus, the output of its preceding call to termcheck(op_i) returned $(o_i, 1)$. By termination agreement, this entails that each node $j \in V_g$ had output (o_i, term_j) for some $\mathrm{term}_j \in \{0,1\}$. Thus, j either terminates immediately with output $\mathrm{op}_j = o_i$ after broadcasting its output or it participates in another loop iteration which all correct nodes $k \in V_g$ start with $\mathrm{op}_k = o_i$. As discussed earlier, this results in all of these nodes returning o_i in this loop iteration, proving agreement. □

3 EARLY-STOPPING PHASE KING WITH DOUBLING ELECTOR SET SIZE

In this section, we present a simple approach to reducing the bit complexity of the early-stopping Phase King protocol by calling an efficient consensus routine on an initially small elector set.

The pseudocode is given in Algorithm 4 and deviates from Algorithm 1 in that it calls Electors' Broadcast on disjoint sets of exponentially growing size. Thus, consensus is guaranteed after $O(\log f)$ iterations, while constant-factor growth of an initially constant-sized set ensures the early-stopping property.

Algorithm 4: Early-Stopping Phase King with Doubling Elector Set Size at node $i \in V_g$, where $b_i \in \{0,1\}$ is the node's input. Differences to Algorithm 1 are marked in blue. In our implementation subroutine calls are realized by Algorithms 2 and 5, respectively.

```
1  op_i := b_i
   // minimum index of participants in Electors'
      Broadcast for current loop iteration
2  ind := 1
   // loop will not be completed, terminating once
      fewer than 1/3 of electors are faulty
3  for j = 0 to ⌈log(n/6)⌉ do
       // check whether changing opinion is valid
4      (op_i, strong_i) := validator(op_i)
       // adopt Electors' value if valid
5      op_i :=
         electorsBC(op_i, strong_i, {ind, ..., max{ind + 3 · 2^j, n}})
6      ind := ind + 3 · 2^j + 1
       // check for termination
7      (op_i, term_i) := termcheck(op_i)
8      if term_i = 1 then
9          broadcast op_i        // termination broadcast
10         return op_i                          // terminate
```

Lemma 3.1. *There is some $j \in \{0, \dots, \lceil \log(\max\{f/2, 1\}) \rceil$ such that fewer than one third of the nodes on which Electors' broadcast is called in iteration j of Algorithm 4 are faulty.*

Proof. Note that the claim is trivially satisfied if $f \leq 1$, since for $j = 0$ Electors' Broadcast is called on 4 nodes. Hence, assume that $f \geq 2$, implying that $\lceil \log(\max\{f/2, 1\}) \rceil = \lceil \log(f/2) \rceil$.

If $\sum_{j=0}^{\lceil \log(f/2) \rceil} 3 \cdot 2^j + 1 \geq n$, the sets on which Electors' Broadcast is called in iterations 0 to $\lceil \log(f/2) \rceil$ form a partition of the entire node set. Because $f < n/3$, then there must be one of the considered iterations satisfying the claim.

On the other hand, if $\sum_{j=0}^{\lceil \log(f/2) \rceil} 3 \cdot 2^j + 1 < n$, assume for contradiction that the claim is false. Thus, in each iteration at least $2^j + 1$ nodes on which Electors' Broadcast is called are faulty. However, this is a contradiction, because the total number of faults is then at least

$$\sum_{j=0}^{\lceil \log(f/2) \rceil} 2^j + 1 = 2^{\lceil \log(f/2) \rceil + 1} - 1 + (\lceil \log(f/2) \rceil + 1) > f. \quad \square$$

Corollary 3.2. *If the implementations of the subroutine calls in Algorithm 4 meet the specifications given in Definitions 2.2, 2.5 and 2.7, Algorithm 4 solves binary consensus and terminates at the latest in iteration $\lceil \log(\max\{f/2, 1\}) \rceil$ of its main loop.*

Proof. Analogous to Theorem 2.9, using Lemma 3.1 to show that the algorithm terminates within $\lceil \log(\max\{f/2, 1\}) \rceil$ iterations. □

What remains is to come up with an efficient implementation of Electors' Broadcast for $|K| \neq 1$ and to bound the resulting time and bit complexity of the algorithm.

Efficient Electors' Broadcast when $|K| > 1$

This task has already been solved in prior work for the purpose of obtaining an efficient consensus protocol that is not early-stopping. Since Algorithm 4 starts with $|K| = 4$ and increases K by a constant factor in each step, the Electors' Broadcast implementation itself needs not be early-stopping. In [2], Electors' Broadcast is solved by letting the electors run consensus to achieve a common view, and then act as if they were a single king. For completeness, we state the resulting Electors' Broadcast routine in Algorithm 5.

Algorithm 5: Electors' broadcast with elector set K at node $i \in V_g$, where (b_i, strong_i) is the input of i. The algorithm assumes that a consensus protocol for the nodes of K is given and has a known running time R, so that all correct nodes expect the electors' broadcast in round $R + 1$ and terminate at the end of round $R + 1$.

```
1 o_i := b_i
2 if i ∈ K then
3 |   participate in consensus on node set K with input b_i
4 |   broadcast the local output c_i of the consensus run
  |       // electors' broadcast
5 if strong = 0 then
  |   // break ties arbitrarily
6 |   set o_i to majority value received from nodes in K
7 return o_i
```

In [2], this approach is used to solve consensus recursively, i.e., the consensus routine called is identical to the top-level protocol executed on roughly half of the nodes. This ensures that the top-level routine requires only a constant number of broadcasts from each node, while the two recursive calls each have both half as many senders and receivers. This reduces the bit complexity from cubic to quadratic, resulting in the following theorem.

Theorem 3.3 ([2]). *The recursive Phase King protocol solves binary consensus in the presence of up to $t \leq \lceil n/3 \rceil - 1$ Byzantine faults. Regardless of the number of faults, it terminates in $O(t)$ rounds and the total number of bits communicated by nodes in V_g is $O(tn)$.*

We can now plug this algorithm for $t = \lceil |K|/3 \rceil - 1$ into Algorithm 5 to obtain an efficient implementation of Electors' Broadcast.

Lemma 3.4. *Algorithm 5 with the consensus routine given by Theorem 3.3 implements Electors' Broadcast with running time $O(|K|)$. Only correct nodes in K send messages, with in total $O(|K|n)$ bits.*

Proof. To see that the validity condition of Electors' Broadcast holds, observe that all $i \in V_g$ with $\text{strong}_i = 1$ maintain their opinion and return b_i.

Regarding electors' agreement, assume that $|V_g \cap K| > \lfloor 2|K|/3 \rfloor$. Thus, by Theorem 3.3 the consensus call succeeds, i.e., agreement and validity hold. Denote by $b \in \{0, 1\}$ the (agreed-on) return value of nodes in $V_g \cap K$, i.e., $c_i = b$ for all $i \in V_g \cap K$.

We distinguish two cases, the first being that there is no $i \in V_g$ with $\text{strong}_i = 1$. In this case, each $i \in V_g$ sets $o_i := b$, since strictly more than one half of the nodes in K are correct and broadcast b after termination of the consensus run.

The other case is that for some $i \in V_g$, $\text{strong}_i = 1$. Then, by weak agreement of the inputs, for all $j \in V_g$, $b_j = b_i$. Since b_j is the input nodes in K use in the call to consensus, validity of this call implies that $b = b_i$. Hence, a majority of the nodes in K broadcast b after completing execution of the consensus protocol. We conclude that any $j \in V_g$ satisfying $\text{strong}_j = 0$ sets $o_j = b = b_i$, while those with $\text{strong}_j = 1$ stick with $o_j = b_j = b_i$.

To show the bit complexity, observe that only correct nodes that are in K communicate. They send $|V_g \cap K|(n-1) < |K|n$ bits in the final broadcast and, by Theorem 3.3, $O(|K|^2) = O(|K|n)$ bits in the consensus call. Finally, the time complexity equals the time complexity of the call to consensus plus one round for the broadcast, which by Theorem 3.3 is $O(|K|)$. □

From the above results, Theorem 1.1 readily follows.

Theorem 1.1. *Algorithm 4 solves binary consensus with up to $t := \lceil n/3 \rceil - 1$ Byzantine faults in $O(f + 1)$ rounds, where $0 \leq f \leq t$ is the number of actual faults. Its bit complexity is $O(nt \log(f + 2))$.*

Proof. Corollary 3.2 shows correctness. To show the running time and bit complexity bounds, denote by $j_{\max}$ the last iteration of the main loop. By Lemma 3.1, $j_{\max} \leq \lceil \log(\max\{f/2, 1\}) \rceil$. Observe that each iteration of the main loop takes a constant number of rounds and broadcasts, plus the time and communication cost for executing Electors' Broadcast with $|K| = \Theta(2^j)$. By Lemma 3.4, the running time is hence bounded by

$$\Theta\left(\sum_{j=0}^{j_{\max}} 2^j + 1\right) = \Theta(f + 1)$$

and the bit complexity by

$$\Theta\left(\sum_{j=0}^{j_{\max}} (2^j + t)n\right) = \Theta(nt \log(f + 2)).$$ □

4 RECURSIVE EARLY-STOPPING PHASE KING

In this section, we present our early-stopping recursive Phase King protocol, which has optimal resilience and asymptotically optimal round and bit complexities. In the previous section, we used recursion under the hood to achieve a better bit complexity, by using the (non-early-stopping) recursive Phase King protocol from [2]. Now, we eliminate the resulting $\Theta(\log f)$ overhead in bit complexity by recursively calling the early-stopping algorithm itself. In this algorithm, nodes first try to achieve consensus using a single king. If they do not succeed, they use about half of the nodes as electors. If this fails again, the remaining nodes become electors and agreement is guaranteed.

Handling Off-by-One Round Counters

In our algorithm, we face the difficulty that subroutines might stop early in an a priori unknown round. In order to continue execution, nodes need to coordinate *when* to do so. Unfortunately, we cannot rely on consensus to solve this task: the subset of nodes executing the subroutine call might contain too many faulty nodes to be able to do so, and running consensus on all nodes of the instance of the

Algorithm 6: Simulating synchronous algorithm $\mathcal{A}$, code at node v. Correct nodes might not start execution in the same round, but in two consecutive rounds. $\perp$ is used as a special symbol representing that no message has been received from a node during the preceding round (or the algorithm has just been started). For convenience, we assume that the state of s_v maintains all information needed to determine which messages to send. The simulated sequence here is send-receive-compute, where computations "before" the first round are performed before the while loop. However, the computations for simulated round $r \in \mathbb{N}_{>0}$ are deferred to the end of round $2r+1$, since all round-r messages must be collected first.

```
1  r_v := 1                                              // count rounds
2  for w ∈ V do
3  |  m_{w,0} := m_{w,1} := ⊥
4  set s_v to the initial state of v in A based on v's input
5  while A has not terminated do
6  |  if r_v mod 2 = 0 then
7  |  |  for w ∈ V do
8  |  |  |  denote by m the message v sends to w in round
   |  |  |    r_v/2 of A when in state s_v
9  |  |  |  send ((r_v mod 4)/2, m) to w
10 |  if received (b, m) from w ∈ V then
11 |  |  m_{w,b} := m
12 |  if r_v mod 2 = 1 and r_v ≠ 1 then
13 |  |  update s_v, i.e., perform computations of A at the
   |  |    end of round (r_v − 1)/2, assuming that messages
   |  |    m_{w,((r_v−1) mod 4)/2), w ∈ V, have been received
14 |  |  for w ∈ V do
15 |  |  |  m_{w,((r_v−1) mod 4)/2)} := ⊥        // clear memory
16 |  if s_v indicates termination with output o_v then
17 |  |  return o_v
18 |  r_v := r_v + 1
```

algorithm initiating the subroutine call would defeat the purpose of the recursion.

We compromise by ensuring that all nodes recommence execution within one round of each other. However, this requires us to handle this "desynchronization," in that despite it we need a way of ensuring consistent execution of the code. Rather than manually adapting the algorithm, we describe a generic compiler that takes a synchronous algorithm $\mathcal{A}$ and transforms it into a modified algorithm that simulates $\mathcal{A}$ at the expense of a factor-2 slowdown and one additional bit per message.

The simulation spends two rounds per simulated round to ensure that all messages from round $r \in \mathbb{N}_{>0}$ are received before proceeding to the next round. Each messages is prefixed by the simulated round it belongs to modulo 2, which enables receivers to correctly attribute messages to simulated rounds. The pseudocode is given in Algorithm 6. The following theorem formalizes these statements.

Theorem 4.1. *Assume that Algorithm 6 is started at each $v \in V_g$ in two consecutive rounds. Then Algorithm 6 simulates $\mathcal{A}$, i.e., each $v \in V_g$ produces the same output as it would in some execution of $\mathcal{A}$ with the same set of correct nodes V_g with the same inputs. At each node, it terminates after $2r+1$ rounds if it terminates after $r \in \mathbb{N}$ rounds in this execution of $\mathcal{A}$. Correct nodes send the same messages as in this execution, but with an additional bit as prefix.*

Proof. Denote by local round r at node v the loop iteration during which $r_v = r$. Moreover, denote by superspript $s_v^r \in \mathbb{N}_{>0}$ the value of s_v at the end of local round r at node v and by $m_{w,b}^r$ the value of $m_{w,b}$ at node v in local round r before executing Line 15, i.e., clearing memory after state update. We claim that for each $r \in \mathbb{N}$ such that $v \in V_g$ has not terminated, in its local instance of the algorithm

(i) s_v^{2r+1} equals the state of $\mathcal{A}$ after round r (where $r = 0$ means at initialization);
(ii) if r is odd, $m_{w,1}^{2r+1}$ equals the message received from w in round r;
(iii) if $r \neq 0$ is even, $m_{w,0}^{2r+1}$ equals the message received from w in round r; and
(iv) v terminates within $2r+1$ local rounds with output o_v if and only if does so in $\mathcal{A}$ in round r

in some[4] synchronous execution of $\mathcal{A}$ with the same set of correct nodes and inputs. Note that these statements imply the claims of the theorem, so proving the claim will complete the proof of the theorem.

We show the claim by induction on r. For the base case of $r = 0$, observe that s_v^1 equals the state of v at initialization of $\mathcal{A}$ and v terminates in local round 1 with output o_v if and only if $\mathcal{A}$ does so based on its state at initialization. Since (ii) and (iii) are vacuously true for $r = 0$, this covers the base case.

For the step from $r-1 \in \mathbb{N}$ to r, observe that node $w \in V_g$ sends the message indicated by s_w^{2r-1} in its local round $2r$, which could possibly be no message if $\mathcal{A}$ specifies so or w has already terminated. If a message is sent, it is received by v in local round $2r-1$, $2r$, or $2r+1$ due to the prerequisite of the theorem that all nodes start execution within one round of each other. By the induction hypothesis, s_w^{2r-1} corresponds to the state of w after round $r-1$ in some execution of $\mathcal{A}$, so w sends the message consistent with the execution of $\mathcal{A}$ we constructed so far. This applies also if w sends no message, since this is then either indicated by s_w^{2r-1} or w has terminated, which by point (iv) of the induction hypothesis happens if and only if w terminated within r rounds in the constructed execution of $\mathcal{A}$.

If r is odd and w sends a message, it is prefixed by $b = 1$ and v stores it in $m_{w,1}$. Because v resets this variable only when its round counter equals 3 modulo 4, such a reset cannot occur in local rounds $2r-1$ or $2r$. Since $m_{w,1}^{2r+1}$ denotes the value of the variable before the reset, statement (ii) for index r follows if w sends a message. Analogously, if r is even, this message is prefixed by $b = 0$ and stored in $m_{w,0}$, which is not reset in local rounds $2r-1$ or $2r$ because v resets $m_{w,0}$ only when its round counter equals 1 modulo 4. This shows statement (iii) for index r in case w sends a message.

On the other hand, if r is odd and w sends no message to v when $r_w = 2r$, then it sends no message to v that is prefixed with

[4] An execution is fully specified by the inputs of correct nodes and faulty nodes' messages. All statements are meant to refer to the same execution, which is constructed inductively depending on the (arbitrary) messages sent by faulty nodes.

1 in rounds when $r_w \in \{2r-3, \dots, 2r+3\}$ either, because it only sends such messages in local rounds that are equal to 2 modulo 4. Accordingly, v receives no message from w prefixed by 1 from w during rounds with $r_v \in \{2r-2, \dots, 2r+2\}$. Given that v resets $m_{v,1}$ in the round satisfying $r_v = 2r-3$ (after processing messages with prefix 1 received from w), (ii) follows also in this case for index r. Statement (iii) for index r for the case that w sends no message to v in the round when $r_w = 2r$ is shown analogously.

By the induction hypothesis, s_v^{2r-1} equals the state of v after $r-1$ rounds of the execution of $\mathcal{A}$ constructed so far. As we already established (ii) and (iii) for r, the messages v assumes to have received from correct nodes $w \in V_g$ when updating s_v in round $2r+1$ are consistent with this execution of $\mathcal{A}$ as well. Since faulty nodes may send arbitrary messages (which determine round r of the constructed execution), (i) follows for index r. Finally, (iv) is now immediate from (i) for index r. This completes the induction and hence the proof of the theorem. □

Electors' Broadcast for Early-Stopping Recursive Phase King

Our second ingredient is an efficient implementation of Electors' Broadcast for arbitrary K that stops early if the utilized consensus algorithm stops early. It needs to function correctly under the relaxed condition that all correct nodes start execution within one round of each other, imposing the same constraint on the consensus algorithm it calls. In turn, it is only required to have correct nodes terminate within one round of each other. As we will show later, these conditions are sufficient to enable us to use our consensus protocol recursively to implement the consensus routine we use in the solution to Electors' Broadcast given in Algorithm 7.

To achieve the required property that all nodes terminate within one round of each other, we introduce what we call a *voting barrier*, in which the correct nodes vote to proceed when they are ready. In order to overcome the obstacle that the called consensus routine might fail arbitrarily when a third or more of the participating nodes are faulty, this voting process involves all nodes and allows for "convincing" other nodes if there is proof that a correct node voted for proceeding with output $b \in \{0,1\}$. Concretely, nodes collect $n-t$ votes for value $b \in \{0,1\}$ before proceeding with this value, while voting for output b not only if K indicates this to be the output of the consensus call, but also if there are already $t+1$ votes for this value. Thus, if any correct node passes the barrier in round r due to seeing $n-t$ votes, the remaining correct nodes see $n-2t > t$ votes and vote to proceed in round $r+1$. Conceptually, this approach is borrowed from the classic fault-tolerant clock synchronization protocol by Srikanth and Toueg [25].

Lemma 4.2. *Suppose that correct nodes start executing Algorithm 7 within one round of each other and denote for $f < k/3$ by $T(k,f)$ a worst-case running time bound for the consensus algorithm on k nodes with f faults and by $B(k)$ its bit complexity. Then*

- *Algorithm 7 satisfies the validity and electors' agreement conditions of Definition 2.5;*
- *correct nodes terminate within one round of each other;*
- *correct nodes terminate within $T(|K|, \lceil |K|/3 \rceil - 1) + O(1)$ rounds;*

Algorithm 7: Electors' broadcast with elector set K at node $i \in V_g$, where (b_i, strong_i) is the input of i. The algorithm assumes that a consensus protocol of resilience $\lceil |K|/3 \rceil - 1$ on node set K is given, whose worst-case running time is bounded by a known value $T(|K|, \lceil |K|/3 \rceil - 1)$.

```
1  op_i := b_i
2  if i ∈ K then
3      in separate thread:
4      execute the given consensus protocol on node set K
         with input b_i
5      if the consensus run terminated after 0 rounds then
           /* wait for other nodes to begin execution
              of Algorithm 7                          */
6          wait for one round
       // electors' broadcast
7      broadcast elect(c_i), where c_i is the local output of the
         consensus run
8  for max{T(|K|, ⌈|K|/3⌉ − 1) + 3, 4} rounds do
       // execute only once for each b
9      if for b ∈ {0,1} received elect(b) from at least ⌈|K|/3⌉
         nodes in K or elect(b) from at least t + 1 nodes in V
         during loop then
10         broadcast vote(b)
11     if for b ∈ {0,1} received vote(b) from at least n − t nodes
         in V during loop then
12         if strong = 0 then
               // if true for both b, anything is fine
13             op_i := b
14         return (op_i)
15 return (op_i)                    // consensus call timed out
```

- *if $|K \cap V_g| > \lfloor 2|K|/3 \rfloor$, then correct nodes terminate within $T(|K|, |K \setminus V_g|) + O(1)$ rounds; and*
- *correct nodes send in total $B(k) + O(n^2)$ bits.*[5]

Proof. Nodes with $\text{strong}_i = 0$ maintain their opinion and return b_i, showing the validity condition of Definition 2.5. For electors' agreement, observe that the precondition that $|K \cap V_g| > \lfloor 2|K|/3 \rfloor$ ensures that the consensus call satisfies agreement and validity, and also terminates within $T(|K|, |K \setminus V_g|) \leq T(|K|, \lceil |K|/3 \rceil - 1)$ rounds at all nodes in $K \cap V_g$. Denote by $b \in \{0,1\}$ the output of this call. Observe that correct nodes sends no elect$(1-b)$ messages. Hence, correct nodes send no vote$(1-b)$ messages. Thus, no correct node adopts opinion $1-b$, and each correct node i outputs b_i or b.

We distinguish two cases. If there is a node $i \in V_g$ with $\text{strong}_i = 1$, then $b_j = b_i$ for all $j \in V_g$ by weak agreement of the inputs. Hence, by validity of the consensus routine, $b = b_i$ and electors' agreement follows.

On the other hand, if $\text{strong}_i = 0$ for all $i \in V_g$, note that termination of the consensus call within $T(|K|, \lceil |K|/3 \rceil - 1)$ rounds implies that each node in $K \cap V_g$ broadcasts elect(b) no later than its local round $\max\{T(|K|, \lceil |K|/3 \rceil - 1) + 1, 2\}$, where the 2 accounts for the

[5] Note that the bit complexity will depend on the recursion level, since we need to distinguish between messages from different threads. However, the effect on the overall bit complexity is asymptotically negligible.

special case that $T(|K|, \lceil |K|/3 \rceil - 1) = 0$ and the node waits for one round. Since we make sure that no such message is broadcast in the first round, by assumption each correct node will receive this message while executing Algorithm 7, no later than local round $\max\{T(|K|, \lceil |K|/3 \rceil - 1) + 2, 3\}$. It follows that all correct nodes broadcast elect(b) by the next round, and consequently terminate with output b if they did not already do so.

To see that correct nodes terminate within one round of each other, suppose that $i \in V_g$ terminates in round $r \in \mathbb{N}$. If termination at i is due to the for loop having completed, by the assumption that all correct nodes start within one round of each other, each node $j \in V_g$ terminates by the end of round $r + 1$. If termination is due to i having received vote(b) messages from $n - t$ nodes during execution of the for loop, observe that at least $n - 2t > t$ of them were sent by correct nodes. Since correct nodes cannot satisfy any of the preconditions for broadcasting vote(b) in the first round of executing the loop, these messages were sent when all correct nodes were already executing Algorithm 7 (and thus in particular the loop). Hence, any correct node satisfies the precondition for sending vote(b) by the end of round r and broadcasts vote(b) in round $r + 1$. It follows that each correct node received vote(b) messages from all at least $n - t$ correct nodes while executing the loop and terminates by the end of round $r + 1$.

Next, termination within $T(|K|, \lceil |K|/3 \rceil - 1) + O(1)$ trivially follows from the maximum number of rounds spent in the for loop. If $|K \cap V_g| > \lfloor 2|K|/3 \rfloor$, correct nodes in K terminate the consensus instance and, by agreement, broadcast elect(b) for the same $b \in \{0, 1\}$ within $T(|K|, |K \setminus V_g|) + O(1)$ rounds. Hence, all correct nodes will sent vote(b) at most one round later (unless some node already terminated; this case is covered by the preceding analysis) and terminate one round after that; again, we use that correct nodes send no vote messages in the first round of the loop, so correct nodes do not "miss" any such messages due to not having started Algorithm 7 yet.

Finally, the bound on message complexity follows from observing that, apart from messages sent by the consensus run, each correct node broadcasts at most twice. □

Main Algorithm

With the above pieces in place, we are ready to state and analyze our recursive early-stopping protocol. The pseudo-code is given in Algorithm 8. The algorithm follows the same pattern as the previous ones, but implements Electors' Broadcast by Algorithm 7 with recursive application of itself, which it calls at most twice on roughly half of the nodes. In order to cope with correct nodes potentially terminating in different rounds, the compiler given in Algorithm 6 is applied to the entire code with the exception of subroutine calls to Algorithm 7.

We now turn to analyzing the algorithm. First, we observe that at last one of the calls to Electors' Broadcast must succeed, i.e., there are not too many faulty nodes in K.

Lemma 4.3. *If $1 \notin V_g$, then $|V_j| > 3|V_j \setminus V_g|$ for some $j \in \{0, 1\}$.*

Proof. Observe that the sets $\{1\}$, V_0, and V_1 form a partition of V. Since $n > 3t$, thus at least one of the sets must contain fewer than one third faulty nodes. □

Algorithm 8: Recursive early-stopping Phase King algorithm at node $i \in V_g$, where $b_i \in \{0, 1\}$ is the node's input. The algorithm assumes that all correct nodes start execution within one round of each other. All code but the two calls to electorsBC($\text{op}_i, \text{strong}_i, V_j$) for $j \in \{0, 1\}$ is passed through the compiler given by Algorithm 6. These two calls are implemented by Algorithm 7 recursively using Algorithm 8 as consensus routine. Note that this can cause nodes to send messages from multiple parallel threads in the same round, one for each level of recursion. By labeling the messages of recursion level ℓ with their level index using a variable length code, recipients can correctly attribute messages to threads at an additive overhead of $O(\log(\ell + 1))$ bits for each level-ℓ message. As before, the calls to validator(op_i) and termcheck(op_i) are implemented by Algorithm 2, and Algorithm 3 implements the call to electorsBC($\text{op}_i, \text{strong}_i, \{1\}$).

```
1  op_i := b_i
   // check whether changing opinion is valid
2  (op_i, strong_i) := validator(op_i)
   // electors' broadcast implemented in Algorithm 3
3  op_i := electorsBC(op_i, strong_i, {1})
   // check for termination
4  (op_i, term_i) := termcheck(op_i)
5  if term_i = 1 then
       // termination broadcast
6      broadcast op_i
7      return op_i                                  // terminate
8  V_0 := {2, ..., ⌈n/2⌉}
9  V_1 := {⌈n/2⌉ + 1, ..., n}
10 for j ∈ {0, 1} do
11     (op_i, strong_i) := validator(op_i)
       /* electors' broadcast implemented in
          Algorithm 7                                  */
12     op_i := electorsBC(op_i, strong_i, V_j)
       // check for termination
13     (op_i, term_i) := termcheck(op_i)
14     if term_i = 1 then
           // termination broadcast
15         broadcast op_i
16         return op_i                              // terminate
```

Correctness readily follows from the already established results.

Corollary 4.4. *If all correct nodes start executing Algorithm 8 within one round of each other, Algorithm 8 solves binary consensus.*

Proof. The proof is by induction on n. For the base case of $n = 1$, $f = t = 0$ and hence $1 \in V_g = \{1\}$. Correctness hence follows analogously to the proof of Theorem 2.9 based on the correctness of the synchronous subroutines and Theorem 4.1. For the step from n to $n + 1$, from the induction hypothesis and Lemma 4.2 we get the correctness of the calls to for electorsBC($\text{op}_i, \text{strong}_i, V_j$) for $j \in \{0, 1\}$.

Correctness of the remaining subroutine calls follows from the prerequisites of the corollary and Theorem 4.1. We can hence reason analogously to the proof of Theorem 2.9, where Lemma 4.3 shows that at the latest the final call to Electors' Broadcast guarantees that $|K \cap V_g| > \lfloor 2|K|/3 \rfloor$. □

It remains to analyze the time and message complexity of the algorithm. To this end, denote by $T(n, f)$ the worst-case running time and by $B(n, \ell)$ the worst-case bit complexity of the algorithm when executed in recursion depth $\ell \in \mathbb{N}$ ($\ell = 1$ denotes the initial call). Here, we take into account ℓ, since we need to add $O(\log \ell)$ bits to each message in order for nodes to distinguish between messages from different recursion levels that are sent concurrently.

Lemma 4.5. *When implemented as stated in Algorithm 8 with Algorithms 2 and 3 providing the synchronous routines for the calls to* $\mathrm{op}_i :=$ electorsBC$(\mathrm{op}_i, \mathrm{strong}_i, \{1\})$, validator, *and* termcheck, *respectively, Algorithm 8 terminates within* $O(f + 1)$ *rounds and lets correct nodes send* $O(n^2)$ *bits in total.*

Proof. We prove the claim by induction on $k \in \mathbb{N}$, where the claim is for $n < 2^{k+1}$ it holds that $T(n, f) \leq C(f + 1)$ and $B(n, \ell) \leq Cnt \log(\ell + 1)$ for a sufficiently large constant C. For the base case of $k = 0$, $n = 1$, all subroutines take $O(1)$ rounds, and no messages are sent.

For the step from $k \in \mathbb{N}$ to $k + 1$, denote for $j \in \{0, 1\}$ by $f_j := |V_j \setminus V_g|$. Note that, analogously to the proof of Theorem 2.9, the algorithm terminates at the latest when calling termcheck after a call to electorsBC with $|V_g \cap K| > 2|K|/3$. Moreover, by Theorem 4.1, Lemma 4.2, the fact that all subroutines except for the calls to electorsBC$(\mathrm{op}_i, \mathrm{strong}_i, V_j)$ are implemented by constant-round primitives, and the assumption that C is sufficiently large, all steps but the recursive calls to Algorithm 8 take at most C rounds. We distinguish three cases.

(1) Node 1 is correct. Then the algorithm terminates without a recursive call, i.e., within $C \leq C(f + 1)$ rounds.
(2) Node 1 is faulty, but $f_0 < |V_0|/3$. Since $|V_0| \leq n/2 \leq 2^{k+1}$, we can apply the induction hypothesis to bound $T(|V_0|, f_0) \leq C(f_0 + 1)$. The algorithm terminates after the first recursive call, by round $C + C(f_0 + 1) \leq C(f + 1)$.
(3) Node 1 is faulty and $f_0 \geq |V_0|/3$. Then Lemma 4.2 shows termination of the first recursive call in $T(|V_0|, \lceil |V_0|/3 \rceil - 1) \leq T(|V_0|, f_0 - 1)$ rounds. As for $j \in \{0, 1\}$ we have that $|V_j| \leq n/2 \leq 2^{k+1}$, we can apply the induction hypothesis to bound $T(|V_0|, f_0 - 1) \leq Cf_0$ and $T(|V_1|, f_1) \leq C(f_1 + 1)$. We conclude that the algorithm terminates within $C + Cf_0 + C(f_1 + 1) = C(f + 1)$ rounds.

It remains to bound the message complexity. From Theorem 4.1, Lemma 4.2, and the fact that all subroutines but the calls to electorsBC$(\mathrm{op}_i, \mathrm{strong}_i, V_j)$ take constant time, we get that the number of broadcast operations per node outside of recursive calls is constant. Since C is sufficiently large and each such message has size $O(\log(\ell + 1))$ (due to the need to identify the level of recursion it belongs to), this sume up to in total at most $(Cn^2 \log(\ell + 1))/5$ bits. The recursive calls are executed on at most half of the nodes each. By the induction hypothesis, they hence each contribute at most

$$C \log(\ell+2) \left(\frac{n}{2}\right)^2 = Cn^2 \log(\ell+1) \cdot \frac{\log(\ell+2)}{2\log(\ell+1)} < Cn^2 \log(\ell+1) \cdot \frac{4}{5},$$

since $\log(\ell + 2)/\log(\ell + 1) < 1.6$ for $\ell \geq 1$. Hence, the bound on the number of communicated bits follows. This completes the induction. The claim of the lemma now follows, because $T(n, f) \leq C(f + 1) \in O(f + 1)$ and $B(n, 1) \leq Cn^2 \in O(n^2)$. □

Theorem 1.2 is now immediate from Corollary 4.4 and Lemma 4.5.

REFERENCES

[1] Piotr Berman, Juan A. Garay, and Kenneth J. Perry. Towards Optimal Distributed Consensus (Extended Abstract). In *Symposium on Foundations of Computer Science (FOCS)*, page 410–415, 1989.
[2] Piotr Berman, Juan A. Garay, and Kenneth J. Perry. *Bit Optimal Distributed Consensus*, page 313–321. Plenum Press, USA, 1992.
[3] Piotr Berman, Juan A. Garay, and Kenneth J. Perry. Optimal Early Stopping in Distributed Consensus. In *Distributed Algorithms*, pages 221–237, 1992.
[4] Gabriel Bracha. Asynchronous Byzantine Agreement Protocols. *Information and Computation*, 75(2):130–143, 1987.
[5] Armando Castañeda, Yoram Moses, Michel Raynal, and Matthieu Roy. Early decision and stopping in synchronous consensus: a predicate-based guided tour. In *International Conference on Networked Systems (NETYS)*, pages 206–221, 2017.
[6] Tushar Deepak Chandra and Sam Toueg. Time and Message Efficient Reliable Broadcasts. In *Workshop on Distributed Algorithms (WDAG)*, pages 289–303, 1990.
[7] Bogdan S Chlebus and Dariusz R Kowalski. Robust gossiping with an application to consensus. *Journal of Computer and System Sciences*, 72(8):1262–1281, 2006.
[8] Bogdan S Chlebus and Dariusz R Kowalski. Time and communication efficient consensus for crash failures. In *International Symposium on Distributed Computing*, pages 314–328, 2006.
[9] Bogdan S. Chlebus, Dariusz R. Kowalski, and Jan Olkowski. Fast Agreement in Networks with Byzantine Nodes. In *Symposium on Distributed Computing (DISC)*, volume 179, pages 30:1–30:18, 2020.
[10] Brian A Coan. A Communication-Efficient Canonical Form for Fault-Tolerant Distributed Protocols. In *Symposium on Principles of Distributed Computing (PODC)*, pages 63–72, 1986.
[11] Brian A. Coan and Jennifer L. Welch. Modular construction of a Byzantine agreement protocol with optimal message bit complexity. *Information and Computation*, 97(1):61–85, 1992.
[12] Danny Dolev. The Byzantine Generals Strike Again. *Journal of Algorithms*, 3(1):14–30, 1982.
[13] Danny Dolev and Christoph Lenzen. Early-Deciding Consensus is Expensive. In *Symposium on Principles of Distributed Computing (PODC)*, pages 270––279, 2013.
[14] Danny Dolev and Rüdiger Reischuk. Bounds on Information Exchange for Byzantine Agreement. *Journal of the ACM (JACM)*, 32(1):191–204, 1985.
[15] Danny Dolev and H. Raymond Strong. Authenticated Algorithms for Byzantine Agreement. *SIAM Journal on Computing (SICOMP)*, 12(4):656–666, 1983.
[16] Paul Feldman and Silvio Micali. Optimal Algorithms for Byzantine Agreement. In *Symposium on Theory of Computing (STOC)*, pages 148–161, 1988.
[17] Z. Galil, A. Mayer, and Moti Yung. Resolving message complexity of Byzantine Agreement and beyond. In *Foundations of Computer Science (FOCS)*, pages 724–733, 1995.
[18] Idit Keidar and Sergio Rajsbaum. A Simple Proof of the Uniform Consensus Synchronous Lower Bound. *Information Processing Letters (IPL)*, 85(1):47–52, 2003.
[19] Christoph Lenzen, Matthias Függer, Markus Hofstätter, and Ulrich Schmid. Efficient Construction of Global Time in SoCs Despite Arbitrary Faults. In *Euromicro Conference on Digital System Design (DSE/SEAA)*, pages 142–151, 2013.
[20] Nancy A Lynch. *Distributed algorithms*. Elsevier, 1996.
[21] Philippe Raipin Parvédy and Michel Raynal. Optimal Early Stopping Uniform Consensus in Synchronous Systems with Process Omission Failures. In *Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures*, pages 302–310, 2004.
[22] Marshall C. Pease, Robert E. Shostak, and Leslie Lamport. Reaching Agreement in the Presence of Faults. *Journal of the ACM (JACM)*, 27(2):228–234, 1980.
[23] Kenneth J Perry and Sam Toueg. Distributed Agreement in the Presence of Processor and Communication Faults. *IEEE Transactions on Software Engineering (TSE)*, (3):477–482, 1986.
[24] Philippe Raïpin Parvédy, Michel Raynal, and Corentin Travers. Strongly Terminating Early-stopping k-set Agreement in Synchronous Systems with General Omission Failures. *Theory of Computing Systems (TCS)*, 47(1):259–287, 2010.
[25] T.K. Srikanth and Sam Toueg. Optimal Clock Synchronization. In *Symposium on Principles of Distributed Computing (PODC)*, pages 71–86, 1985.