Byzantine Agreement with Unknown Participants and Failures

Pankaj Khanchandani
ETH Zurich
Zurich, Switzerland
kpankaj@ethz.ch

Roger Wattenhofer
ETH Zurich
Zurich, Switzerland
wattenhofer@ethz.ch

Abstract—A set of mutually distrusting participants that want to agree on a common opinion must solve an instance of a Byzantine agreement problem. These problems have been extensively studied in the literature. However, most of the existing solutions assume that the participants are aware of n — the total number of participants in the system — and f — an upper bound on the number of Byzantine participants. In this paper, we show that most of the fundamental agreement problems can be solved without affecting resiliency even if the participants do not know the values of (possibly changing) n and f. Specifically, we consider a synchronous system where the participants have unique but not necessarily consecutive identifiers, and give Byzantine agreement algorithms for reliable broadcast, approximate agreement, rotor-coordinator, early terminating consensus and total ordering in static and dynamic systems, all with the optimal resiliency of $n > 3f$. Moreover, we show that some synchrony is necessary as an agreement with probabilistic termination is impossible in a semi-synchronous or asynchronous system if the participants are unaware of n and f.

I. Introduction

Many modern networks have to be always available and it may not be possible to know the size of the network or the number of failures in advance, since they may change over time. Consider, for example, a database cluster that requires frequent node scaling because of changing load, or a wireless sensor network that experiences a changing number of faulty or disconnected nodes over time. Nakamoto's blockchain [26] is a prominent example where the network is permissionless, i.e., the network is open to *any* number of nodes. So, the number of participants and consequently, the number of failures also change over time. Agreement is a fundamental distributed computing primitive for fault-tolerant networks, however, much of the existing literature assumes that the size n of the network and/or the upper bound f on the number of failures is known to every node [4], [3], [10], [29], [25].

In this paper, we consider fault-prone systems where the nodes do not know the number of nodes n and the maximum number of Byzantine nodes f and study fundamental agreement problems for such systems, in particular:

- Reliable broadcast — ensures that a message is either accepted by every correct node or no correct node [27];
- Rotor-coordinator — selects $f + 1$ leaders for the correct nodes;
- Consensus — every correct node has a binary input and the correct nodes output a common binary value that is an input of some correct node [21];
- Approximate agreement — each correct node has a real number input and has to output a real number that is *strictly* within the correct input values [13];
- Total ordering — each correct node maintains a total order on the system events while participants may enter and leave subject to $n > 3f$.

Since a correct node does not know n and f and a Byzantine node may not announce itself to everyone, there might be more Byzantine nodes in the system than what a correct node thinks. Thus, it may not be possible to achieve a resiliency of $n > 3f$, which can be achieved when the nodes know n and f. When f is known and the identifiers are consecutive, it is easy to agree on a set of $f + 1$ nodes, and consequently ensure the presence of a single correct leader node in the set. We show, however, that these problems can be solved without affecting resiliency even when n and f are not known. Specifically, we give algorithms for solving the above problems in synchronous systems with the resiliency of $n > 3f$, which is optimal for approximate agreement, reliable broadcast and consensus. We also show that the synchronous assumption is necessary as otherwise the problem is impossible and there is non-zero probability of terminating with a disagreement.

An advantage of designing algorithms without the knowledge of n and f is their application to networks where the set of participants change over time. We illustrate this by extending some of our algorithms to solve Byzantine agreement in dynamic networks. In case of dynamic networks, single shot problems where a node acts on one input and terminates with one output are not very useful. So, we consider an agreement task where nodes are required to totally order the events in a system and design an algorithm for that task.

II. Related Work

If the nodes do not known n and f, then the synchronous assumption is necessary. Otherwise, if the network is asynchronous and the message delays are unbounded, agreement is impossible even with probabilistic termination, as we show later. There is a line of work that deals with this problem using oracles or failure detectors [9], [20], [2]. The idea is that a failure detector supplies information about the number of

participants. But, these works also assume that every node knows f. In [28], the authors consider an asynchronous dynamic system with a failure detector where n and f are unknown, but the failure detector assumed eventually removes the Byzantine nodes.

Several ways of improving the robustness of synchronous systems with Byzantine failures have also been explored. For example, Gallet et al. [11] examine a system that can allocate the same identifier to multiple nodes. In [5], [18], [8], the authors examine a synchronous system with mobile Byzantine faults — those which hop from one node to another across rounds. In [22], [23], the authors consider self-stabilizing agreement problems in the presence of Byzantine faults, i.e., the correct nodes have to recover from arbitrary initial state even when the other Byzantine nodes maliciously prevent the correct nodes from recovering.

The Byzantine agreement problems have a long history since the work by Lamport et al. [21]. They gave a $f + 1$ round algorithm with exponential in n message complexity for $n > 3f$. They also showed that the resilience of $n > 3f$ is optimal. Berman et al. [6] later improved the message complexity to polynomial in n, while keeping optimal resilience, and increasing the number of rounds by a small constant. Garay et al. [19] further improved the round complexity to exactly $f+1$, while retaining optimal resilience and polynomial message complexity. The algorithm by Berman et al. [6] is well known as the *king* algorithm and is still commonly used [22], [12], [1]. The approximate agreement algorithm was introduced by Dolev et al. [13] and is a useful primitive in designing distributed algorithms [24], [14].It also requires $n > 3f$ and has optimal resiliency [17]. Srikanth et al. [27] introduced the reliable broadcast abstraction and its use in dealing with Byzantine failures for $n > 3f$. As they remark, this resiliency is optimal as the reliable broadcast abstraction can be used to solve consensus. The algorithms for approximate agreement, reliable broadcast and consensus in this paper generalize the ones from Dolev et al. [13], Srikanth et al. [27] and Berman et al. [6] respectively.

A rotor-coordinator, as also used in [6], is an approach to deal with at most f Byzantine faults by rotating through $f + 1$ coordinator nodes, thus ensuring that one coordinator would be correct. The rotor-coordinator can be easily implemented by rotating through $f + 1$ nodes when f is known and the identifiers are consecutive. However, it is one of the main bottleneck when n and f are unknown and the identifiers are also non-consecutive.

III. SIGNIFICANCE OF THIS WORK

It is not so difficult to observe that if all the correct nodes broadcast in a round, then each correct node v receives less than $n_v/3$ messages from the Byzantine nodes — where n_v is the number of messages received by the node v — irrespective of whether the Byzantine nodes broadcast or not. This observation helps in removing dependency on n and f from the classic known algorithms. However, this observation is not sufficient by itself. Many of the classic algorithms run for fixed $f+1$ rounds,

selecting a different leader in each round. This is a non-trivial problem in our setting, since f is not a common knowledge and also the identifiers are not consecutive. Algorithm 2 for rotor-coordinator essentially solves this problem.

The classical models studied in the literature do not allow the Byzantine nodes to lie about the number of participants in the network since it is assumed to be known by every node. Our system model allows the Byzantine nodes to send conflicting information so that the correct nodes never have a consistent information about the number of participants. Therefore, the algorithms designed are robust against a wider range of malicious behavior. This is especially useful for large dynamic systems where it may not be possible to consistently initialize every node with the value of n and f.

On the other hand, participants are assumed to have access to consistent clocks after initialization, since the computation is assumed to proceed in rounds. So, some consistent initialization (synchronization) is still needed. This is somewhat necessary, since we also show that Byzantine consensus cannot be solved with probabilistic termination if the system is semi-synchronous or asynchronous and the participants do not know n and f. This implies that it is impossible to build blockchain systems for solving agreement problems in asynchronous networks when n and f are not known.

IV. MODEL

The system consists of n nodes, out of which at most f are faulty nodes. The faulty nodes can behave in anyway whatsoever, also known as *Byzantine* behavior. We call the non-Byzantine nodes *correct*. The nodes have unique identifiers, which are not necessarily consecutive. Each node knows its identifier only at initialization apart from a possible input and does not know any global information like n or f. The system is *synchronous* and the computation proceeds in *rounds*. In each round, every node receives the messages that were sent to it in the previous round, does some local computations, and then sends again messages to the other nodes to be consumed in the following round. A correct node can broadcast a message to all the nodes or send a message to a specific node that sent a message to the node before. The identifier of a node is included in the message it sends so the receiver of the message can decipher its sender. Thus, a Byzantine node cannot forge its identifier when communicating directly. However, it can help other Byzantine nodes to do so indirectly by claiming to have received messages from other, possible non-existent, nodes. Byzantine nodes can send duplicate messages across rounds but duplicate messages from the same node in a round are simply discarded.

Note that the only way for a correct node to know about the existence of another node is to receive a message from that node. A Byzantine node may get itself known to only a subset of nodes, however, it can behave as if it already knows all the nodes without having received messages from them. In the rest of the paper, we will sometimes refer to the above model as the *id-only* model for brevity. We give the following algorithms in the *id-only* model for $n > 3f$: reliable broadcast in Section V,

rotor-coordinator in Section VI, consensus in Section VII, and approximate agreement in Section VIII. In Section IX, we show that to solve agreement with probabilistic termination, when n and f are unknown, synchronous assumption is necessary. In Section X, we give a parallel version of the consensus algorithm, where several consensus algorithms can be run in parallel, however, the nodes do not initally agree on the instances to start. In Section XI, we build on the parallel consensus to give algorithms for achieving approximate agreement and total ordering of events in a dynamic network. In Section XII, we discuss the results and some further interesting questions.

V. Reliable Broadcast

Reliable broadcast [27] is an abstraction to deal with the messages sent by the Byzantine nodes. The idea is to enforce that a Byzantine node cannot send contradictory information to different nodes. It can still send around false information but the abstraction ensures that the same false information is seen by all the correct nodes. Concretely, let s be a designated node that may or may not be correct and (m, s) be a message broadcast by s. The message (m, s) is *reliably broadcast* when the following three properties are satisfied.

1) Correctness: If s is correct, then each correct node accepts (m, s).
2) Unforgeability: If a correct node accepts a message (m, s) and s is a correct node, then the message (m, s) was broadcast or sent to all the nodes by the node s.
3) Relay: If a correct node accepts the message (m, s) in a round r, then each correct node accepts the message (m, s) by the round $r + 1$.

Algorithm 1 gives an algorithm for a node v to reliably broadcast a message (m, s) sent by a node s in the first round. Note that in Line 10, the value n_v is *not* the number of messages received in the round r but the number of nodes that sent at least one message to v until the current round r. Also, the algorithm does not terminate as the idea is to use this mechanism as a subroutine in another algorithm that implements its own termination mechanism, as we will see for consensus, where few additional messages per round are used to detect termination. In the following lemmas, we show that the algorithm satisfies the three properties of the reliable broadcast. We will again assume that $n > 3f$.

Lemma 1. *If $n > 3f$, then Algorithm 1 satisfies the correctness property of the reliable broadcast.*

Proof. If the node s is correct, it sends the message (m, s) to all the nodes during the initial broadcast (Line 2). Every good node receives the message and broadcasts $echo(m, s)$ in the next round (Line 7). Let g be the number of good nodes. Then, in the third round, every correct node receives g $echo(m, s)$ messages. Moreover, the value of $n_v \leq n$ in the third round as n is the maximum number of nodes that can send a message to v. As $n > 3f$, we have $g > 2f$ or $3g > 2(f + g) = 2n$. Thus, we have $g > 2n/3 \geq 2n_v/3$. Therefore, every correct node accepts the message in the third round (Line 15). □

Algorithm 1 Reliable broadcast algorithm for a node v to broadcast a message (m, s) sent by a node s in the first round. Each iteration of the loop is a single round.

1: **if** v = s **then**   ▷ Round 1
2:      Broadcast (m, s)
3: **else**
4:      Broadcast $present$
5: **end if**
6: **if** Received (m, s) from s **then**   ▷ Round 2
7:      Broadcast $echo(m, s)$
8: **end if**
9: **for** $r \leftarrow 1$ to ∞ **do**   ▷ Rounds 3 to ∞
10:      Let n_v be the number of nodes that sent at least one message to v until the round r.
11:      **if** Received at least $n_v/3$ $echo(m, s)$ messages and not accepted (m, s) already **then**
12:           Broadcast $echo(m, s)$
13:      **end if**
14:      **if** Received at least $2n_v/3$ $echo(m, s)$ messages and not accepted (m, s) already **then**
15:           Accept (m, s)
16:      **end if**
17: **end for**

Lemma 2. *If $n > 3f$ and a correct node v receives at least $n_v/3$ copies of a message m from distinct nodes in a round r, then at least one of those messages was sent by a correct node.*

Proof. Let f_v'' be the number of faulty nodes that sent m to v in the round r. Since every correct node transmits a message in the first round (Lines 2 and 4), we have $n_v \geq g$, where g is the number of good nodes. So, we can write $n_v = g + f_v'$, where f_v' is the number of faulty nodes that sent at least one message to v until the round r. Using $f_v'' \leq f_v'$ and $n_v = g + f_v'$, the number of correct nodes G that sent a message to v in the round r are at least $n_v/3 - f_v'' \geq (g - 2f_v')/3$. As $g > 2f$, we have $G > 2(f - f_v')/3$ or at least one as $f \geq f_v'$. So, at least one correct node sent the message m to v in the round r. □

Lemma 3. *If $n > 3f$, then Algorithm 1 satisfies the unforgeability property of the reliable broadcast.*

Proof. We need to show that if a correct node accepts a message (m, s) and s is a correct node, then the message (m, s) was broadcast by s. If a message (m, s) was accepted by a correct node v in a round r_a, then v received at least $2n_v/3$ $echo(m, s)$ messages in the round r_a. Thus, the number of correct nodes from which v received the $echo(m, s)$ messages in round r_a are at least $2n_v/3 - f_v'' \geq n_v/3 - f_v''$, where f_v'' is the number of messages received by v from the faulty nodes in the round r_a. Using Lemma 2, at least one of the $echo(m, s)$ messages received by v in the round r_a was sent by a correct node.

Let r_f be the first round when an $echo(m, s)$ message was sent by a correct node u. Thus, in the round r_f, the node u

either received at least $n_u/3$ $echo(m, s)$ messages or received the message (m, s) from s (Lines 12 or 7). If u received at least $n_u/3$ $echo(m, s)$ messages, then using Lemma 2, there is at least one correct node that sent an $echo(m, s)$ message in the previous round. Since r_f is the first round when a correct node sends an $echo(m, s)$ message, the node u must have received the message (m, s) from s in the round r_f. Thus, node s indeed sent the message (m, s). As s is correct, the message (m, s) was broadcast to all the nodes in the first round. □

Lemma 4. *If $n > 3f$ and a correct node v receives at least $2n_v/3$ copies of a message m in a round r, then every correct node u receives at least $n_u/3$ copies of m in the round r.*

Proof. As v receives at least $2n_v/3$ messages, at least $2n_v/3 - f_v''$ of them were sent by the correct nodes, where f_v'' is the number of messages received by v from the faulty nodes in the round r. Let f_v' be the number of faulty nodes from which v received at least one message until the round r. Then, we have $2n_v/3 - f_v'' = 2(g + f_v')/3 - f_v''$, where g is the number of good nodes. As $f_v'' \leq f_v'$ and $f_v' \leq f$ by definition, we have $2(g + f_v')/3 - f_v'' \geq (2g - f)/3$.

Using $n > 3f$ or $g > 2f$, we have $(2g - f)/3 = (g + (g - f))/3 > (g + f)/3$. Thus, at least $(g + f)/3$ correct nodes broadcast the message m and every correct node receives at least $(g + f)/3$ copies of m in the round r. For a correct node u, we have $(g + f)/3 \geq (g + f_u)/3 = n_u/3$, where f_u is the number of faulty nodes from which u has received at least one message until the round r. □

Lemma 5. *If $n > 3f$, then Algorithm 1 satisfies the relay property of the reliable broadcast.*

Proof. Let r be the first round in which a correct node v accepts the message (m, s). Then, we show that every correct node accepts the message (m, s) by the round $r + 1$.

As v accepts the message (m, s) in round r, it received at least $2n_v/3$ $echo(m, s)$ messages. Using Lemma 4, each correct node u receives at least $n_u/3$ $echo(m, s)$ messages in the round r. So, every correct node broadcasts $echo(m, s)$ message in the round r (Line 12) and each one of them receives g $echo(m, s)$ messages in the round $r + 1$. As $g > 2f$, we have $3g > 2(f + g) = 2n$. Thus, we have $g > 2n/3 \geq 2n_u/3$ for every correct node u. Therefore, every correct node accepts the message (m, s) in the round $r + 1$. □

Using Lemma 1, Lemma 3 and Lemma 5, all the properties of the reliable broadcast are satisfied and we have the following theorem.

Theorem 1. *If $n > 3f$, then Algorithm 1 satisfies the properties of the reliable broadcast in the* id-only *model.*

VI. ROTOR-COORDINATOR

The purpose of Rotor-Coordinator is to have a common coordinator node in each round, where the coordinator node is trusted by everyone in that round. After $f + 1$ different coordinators are selected, everyone is sure that at least one of those $f + 1$ selected coordinators was correct, since there are at most f faulty nodes. Algorithm 2 gives the algorithm for selecting a set of $f + 1$ different coordinators, each one in a separate round.

Algorithm 2 Rotor-Coordinator algorithm for a node v. The sets C_v and S_v are used by v to store process identifiers. The set C_v is ordered by the process identifiers in increasing order. We use $|C_v|$ for the size of C_v and $C_v[i]$ for its ith member, where $i \geq 0$. The set B_v holds messages before they are broadcast by v at the end of a round. Note that each iteration of the loop is a single round.

1: $C_v \leftarrow \phi$ ▷ Set of *candidate* coordinators
2: $S_v \leftarrow \phi$ ▷ Set of *selected* coordinators
3: Broadcast *init*   ▷ Round 1
4: Broadcast $echo(p)$ if received *init* from p ▷ Round 2
5: **for** $r \leftarrow 0 \rightarrow \infty$ **do**   ▷ Rounds 3 up to termination
6:      $B_v \leftarrow \phi$
7:      Let n_v be the number of nodes that sent at least one message to v until the round r.
8:      **if** Received at least $n_v/3$ $echo(p)$ and $p \notin C_v$
9:      **then**
10:          $B_v \leftarrow B_v \cup \{echo(p)\}$
11:      **end if**
12:      **if** Received at least $2n_v/3$ $echo(p)$ and $p \notin C_v$
13:      **then**
14:          $C_v \leftarrow C_v \cup \{p\}$
15:      **end if**
16:      $p \leftarrow C_v[r \bmod |C_v|]$ ▷ Select the next coordinator as p.
17:      Let p' be the coordinator selected in the previous round.
18:      **if** Received $opinion(x)$ from p' **then**
19:          Accept x as the coordinator's opinion
20:      **end if**
21:      **if** $p \in S_v$ **then**
22:          **break**
23:      **end if**
24:      $S_v \leftarrow S_v \cup \{p\}$
25:      **if** $v = p$ **then**   ▷ Check if v itself is the coordinator.
26:          Let o_v be v's current opinion.
27:          $B_v \leftarrow B_v \cup \{opinion(o_v)\}$ ▷ Broadcast v's opinion at the end of the round.
28:      **end if**
29:      Broadcast B_v if its non-empty
30: **end for**

The idea is that every correct node broadcasts its willingness to become a coordinator initially, when the faulty nodes may or may not participate (Line 3). Every correct node v keeps a set of candidate coordinators C_v, which it updates in a reliable broadcast fashion (Lines 10 and 14). In each round, a correct node v selects the coordinator with the next larger identifier, say p, from the set C_v and adds it to the set of selected coordinators S_v (Lines 16 and 24). The node v accepts the opinion from p in the next round as the coordinator's opinion (Line 19) and broadcasts its own opinion as the coordinator's

opinion in case v was selected as the coordinator from the set C_v (Line 27). The node v terminates when it reselects the same node as the coordinator (Line 22). The hope is that by the time a correct node terminates, it has already witnessed a round in which every correct node accepts the opinion of a common and a correct coordinator. We start by observing that if a correct node adds p to its set of candidate coordinator C_v, then another correct node u adds p to its set C_u as well by the next round.

Lemma 6. *If a correct node v adds p to the set C_v in a round r, then any correct node $u \neq v$ adds p to the set C_u by the round $r + 1$.*

Proof. The set B_v is emptied at the beginning of every round r and is broadcast at the end of the round r. Thus, the algorithm for adding a process identifier p to C_v is same as that of accepting a message (m, s) in Algorithm 1 if $(m, s) = p$. So, the lemma follows using Lemma 5 for the relay property of the reliable broadcast. □

We call a round a *good round* if the same node p was selected as a coordinator by every correct node and the node p is correct. In the following, we show that every correct node witnesses a good round before it terminates, if $n > 3f$. We will call a round as a *silent* round if the set C_v remains unchanged for every correct node v, i.e, no correct node executes Line 14 in that round. A *non-silent* round is a round that is not silent. We observe that in a silent round, the value of C_v is identical for every correct node v. If they were not, then there is a silent round between a correct node v adding an identifier p to its set C_v and another correct node $u \neq v$ adding p to its set C_u. This contradicts *Lemma 6*. The assumption $n > 3f$ is used for reliable broadcast and also to ensure a good round. With $n > 4f$, a good round is easily ensured, but $n > 3f$ suffices with careful observation as follows.

Lemma 7. *If $n > 3f$, then every correct node witnesses at least one good round until it terminates.*

Proof. Assume for contradiction that a node v terminates in the round with $r = r_t$ without witnessing a good round. Consider a round with $r = r_c \leq r_t$. Let $F_v \subseteq C_v$ and $G_v \subseteq C_v$, respectively, be the set of faulty node identifiers and the set of good or correct node identifiers in C_v when the coordinator node is selected in the round r_c (Line 16). Thus, we have $|C_v| = |F_v| + |G_v|$.

Using Lemma 1, all the correct node identifiers are added to C_v, even before the first coordinator node is selected. So, we have $|G_v| = n - f$ and $|C_v| = |F_v| + n - f$. Using $n > 3f$, we get $|C_v| > |F_v| + 2f$. Say that there is *no* correct node u that added a faulty identifier to its set C_u in the round with $r = 0$. Then, every correct node selects a common coordinator from the set G_v and v witnesses a good round before termination, a contradiction. Thus, there is a correct node u that adds a faulty identifier to its set C_u in the round with $r = 0$. For every non-silent round afterwards, at least one faulty node identifier is added to the set C_u of some correct node u. Using

Lemma 6, if a faulty node identifier p is added to C_u, every correct node $w \neq u$ adds p to C_w by the next round. Thus, we have $2f \geq n_{ns}$, where n_{ns} is the number of non-silent rounds prior to the round r_c and starting from the round $r = 0$. Therefore, we have $|C_v| > |F_v| + n_{ns}$.

Moreover, until the round r_c, node v has neither witnessed a good round, nor it has selected the same node again as a coordinator by our assumption. So, in all the silent rounds prior to the round r_c, a unique faulty node was selected as a coordinator by v. Therefore, if n_s is the number of silent rounds prior to the round r_c, then $|F_v| \geq n_s$ since v selects a node as a coordinator only after adding it to the set C_v. So, we have $|C_v| > n_s + n_{ns}$.

Since r starts from 0, we have $n_s + n_{ns} = r_c$. So, we have $|C_v| > r_c$ and $r_c \bmod |C_v| = r_c$. Since the above inequality is true for every round $r_c \leq r_t$, a node that was already selected as a coordinator, is in the set $\{C_v[r \bmod |C_v|] : r < r_c\}$. Therefore, for selecting the same identifier as a coordinator again, it must be that $r > |C_v| > r_c$, a contradiction. □

Theorem 2. *If $n > 3f$, then every correct node terminates in $O(n)$ rounds and there is a round in which every correct node accepts the opinion of a common and a correct coordinator node.*

Proof. As a node terminates as soon as it selects the same node as a coordinator and there are n nodes in total, the node terminates in at most n rounds. Using Lemma 7, the node also witnesses a good round before termination and accepts the corresponding opinion in the next round (Line 19). □

VII. CONSENSUS

In this section, we give an $O(f)$ round consensus algorithm in the *id-only* model, where f is the number of faulty byzantine nodes in the system. Algorithm 3 gives an algorithm based on [7]. Every correct node v has an input x_v, which is a real number. Again, every correct node has to output a common correct value. If the inputs are all same, then the output must be that value. We consider real number inputs here, unlike binary inputs in Section VII, since we use it later for ordering events in a system, which can be non-binary.

In the following, we prove the correctness of Algorithm 3. We refer to an iteration of the loop as a *phase*.

Lemma 8. *If $x_v = x$ for every correct node at the start of the phase, all the nodes terminate with the output x at the end of the phase.*

Proof. Every correct node broadcasts $input(x)$ at the start of the phase. So, every correct node v receives g $input(x)$ messages. As $n > 3f$, we have $g > 2f$. Thus, we have $g + 2g > 2(f + g)$ or $g > 2n/3 \geq 2n_v/3$. So, all the correct nodes broadcast $prefer(x)$ (Line 6). Every correct node v receives $g \geq 2n_v/3$ $prefer(x)$ messages, keeps their opinion to x (Line 9), and broadcasts $strongprefer(x)$ (Line 12). Again, each correct node v receives $g \geq 2n_v/3$ $strongprefer(x)$ messages and terminates with the output x (Line 19). □

Algorithm 3 An $O(f)$ round consensus algorithm in the *id-only* model. To initialize the rotor-coordinator in Line 1, run the first two lines of the Algorithm 2. To initialize n_v in Line 2, collect the identifiers from which a message has been received, and count them. Later, a node only accepts messages from a node if it counted towards n_v during the initialization and discards the messages from the other nodes. If a node u receives a message from another node v during initialization but not later inside the loop, then u assumes that v sent the same message as sent by u in the previous round.

1: Initialize rotor-coordinator   ▷ Rounds 1 and 2
2: Initialize n_v
3: **while** $true$ **do**
4:      Broadcast $input(x_v)$ ▷ Next Round
5:      **if** Received at least $2n_v/3$ $input(x_v)$ **then**   ▷ Next Round
6:         Broadcast $prefer(x_v)$
7:      **end if**
8:      **if** Received at least $n_v/3$ $prefer(x)$ **then**   ▷ Next Round
9:         $x_v = x$
10:     **end if**
11:     **if** Received at least $2n_v/3$ $prefer(x)$ **then**
12:        Broadcast $strongprefer(x)$
13:     **end if**
14:     Execute a round of rotor-coordinator ▷ Next Round using x_v as v's current opinion. Let c be the value accepted as the coordinator's opinion.
15:     **if** Received less than $n_v/3$ $strongprefer(x)$ **then**   ▷ Next Round
16:        $x_v = c$
17:     **end if**
18:     **if** Received at least $2n_v/3$ $strongprefer(x)$ **then**
19:        Terminate and output x
20:     **end if**
21: **end while**

Lemma 9. *If a correct node u receives $2n_u/3$ copies of a message m and a correct node v receives $2n_v/3$ copies of a message m' in the same round, then at least one correct node sent both m and m' in the previous round.*

Proof. The number of messages G sent by the good nodes is at least $2n_u/3 - f_u + 2n_v/3 - f_v$, where f_u is the number of m messages sent to u by the faulty nodes, and f_v is the number of m' messages sent to v by the faulty nodes. As $n_u = g + f_u$ and $n_v = g + f_v$, we have $G > 4g/3 - (f_u + f_v)/3$. We have $g > f_u + f_v$ since $f_u \leq f$, $f_v \leq f$ and $g > 2f$. Thus, we have $G > g$ and at least one correct node sent both m and m' in the previous round. $\square$

Lemma 10. *If a correct node terminates in a phase, then all other correct nodes have the same opinion at the end of the phase.*

Proof. Say, a correct node v terminates with the output x. Then, it received at least $2n_v/3$ $strongprefer(x)$ messages. So, all the correct nodes received at least $n_v/3$ $strongprefer(x)$ messages using Lemma 4 and none of them switches to the coordinator's opinion (Line 22). Moreover, at least one correct node u sent a $strongprefer(x)$ message using Lemma 2. The node u received $2n_u/3$ $prefer(x)$ messages. Using Lemma 4, at least $n_u/3$ of those messages were sent by the correct nodes and so each node received at least $n_u/3$ $prefer(x)$ messages. It is not possible that a correct node also received $prefer(x')$, where $x \neq x'$. Indeed, if it was so, then using Lemma 2 there is a correct node s that sent $prefer(x)$ and a correct node t that sent $prefer(x')$. Thus, node s received $2n_s/3$ $input(x)$ messages and node t received $2n_t/3$ $input(x')$ messages. Using Lemma 9, a correct node sent both $input(x)$ and $input(x')$ messages in the same round, a contradiction. Thus, every correct node changed their opinion to x (Line 9), which remains unchanged until the end of the phase. $\square$

Lemma 11. *If the coordinator is correct and none of the correct nodes have terminated, then all the correct nodes have the same opinion by the end of the phase.*

Proof. Consider the first phase when the coordinator is correct. Either every correct node v receives less than $n_v/3$ $strongprefer(x)$ messages, in which case all the correct nodes have the same opinion by the end of the phase, and we are done. Otherwise, there is a correct node u that received at least $n_u/3$ $strongprefer(x)$ messages. Using Lemma 2, there is at least one correct node w that sent a $strongprefer(x)$ message. Thus, node w received at least $2n_w/3$ $prefer(x)$ messages. Using Lemma 4, every correct node v received at least $n_v/3$ $prefer(x)$ messages. As before, it is impossible that a correct node v also receives $n_v/3$ $prefer(x')$ messages, where $x \neq x'$. So, every correct node, including the coordinator, changes its opinion to x (Line 9). Since the coordinator is correct, it sends the same opinion x to all the nodes. Thus, even if some correct nodes decide to change their opinion to the coordinator's opinion, all the correct nodes still have the same opinion at the end of the phase. $\square$

We can now combine the previous lemmas into the following theorem.

Theorem 3. *Algorithm 3 solves consensus in $O(f)$ rounds in the id-only model.*

Proof. If the correct nodes have the same input x, then they output x using Lemma 8. Otherwise, one of the following happens within $O(f)$ rounds: either a correct node terminates with an output x or a correct coordinator gets picked. In either case, the correct nodes have the same opinion at the end of the phase, and terminate with the same output by the next round using Lemma 8. $\square$

VIII. Approximate Agreement

In the approximate agreement problem [13], each correct node takes a real number input and outputs a real number. Let

i_{min} and i_{max}, respectively, be the minimum and the maximum value that is an input of a correct node. Similarly, let o_{min} and o_{max}, respectively, be the minimum and maximum value that is output by a correct node. The values output by the correct nodes must satisfy the following conditions.

1) The value output by each correct node is within the *input range* $[i_{min}, i_{max}]$.
2) The output range $[o_{min}, o_{max}]$ is strictly smaller than the input range, i.e., $(o_{max} - o_{min}) < (i_{max} - i_{min})$ if $i_{max} \neq i_{min}$.

Algorithm 4 Approximate Agreement algorithm for a node v. The input value of the node is i_v.

1: Broadcast i_v to all the nodes (including self).
2: Let R_v be the set of received values and $n_v = |R_v|$.
3: Discard $\lfloor n_v/3 \rfloor$ smallest and $\lfloor n_v/3 \rfloor$ largest values from the set R_v to obtain the set S_v.
4: Output $o_v = (\min S_v + \max S_v)/2$, where $\min S_v$ and $\max S_v$ are the minimum and maximum value of the set S_v respectively.

Algorithm 4 solves the problem. The following lemma shows that the algorithm satisfies the first property of the approximate agreement, i.e., the output range lies within the input range.

Lemma 12. *If $n > 3f$, then $o_v \in [i_{min}, i_{max}]$ for every correct node v.*

Proof. Let g be the number of correct nodes. Then, the node v receives at least g values from the correct nodes after the first round. Let f_v be the number of values received by v from the Byzantine nodes. Therefore, we have $f_v \leq f$ as f is the number of faulty nodes and v receives at most one value from each faulty node in a round. As $n = f + g$ and $f_v \leq f$, we can rewrite $n > 3f$ as $g + f > 2f_v + f$. Thus, we have $(g + f_v)/3 > f_v$ or $\lfloor (g + f_v)/3 \rfloor \geq f_v$ as f_v is an integer. As $n_v = g + f_v$, we have $\lfloor n_v/3 \rfloor \geq f_v$.

As there are at most f_v faulty values in the set R_v and $\lfloor n_v/3 \rfloor \geq f_v$, the minimum value $\min S_v$ left after discarding $\lfloor n_v/3 \rfloor$ smallest values from R_v satisfies $\min S_v \geq i_{min}$, where i_{min} is the minimum value received from a correct node. Using a similar argument, the maximum value $\max S_v$ satisfies $\max S_v \leq i_{max}$. Therefore, the output o_v, which is the average of $\min S_v$ and $\max S_v$, satisfies $o_v \in [i_{min}, i_{max}]$. □

Let i_{med} be the median of the input values at the correct nodes. In the following lemma, we show that the value i_{med} is never discarded by a correct node while computing the set S_v.

Lemma 13. *If $n > 3f$, then the value $i_{med} \in S_v$ for every correct node v.*

Proof. Let g be the number of correct nodes. Using $n > 3f$ and $n = g + f$, we get $f < g/2$. Using $n_v = g + f_v$ and $f_v \leq f$, we get $\lfloor n_v/3 \rfloor \leq n_v/3 = (g + f_v)/3 \leq (g + f)/3$.

As $f < g/2$, we get $\lfloor n_v/3 \rfloor < g/2$. Therefore, even if all the smallest $\lfloor n_v/3 \rfloor$ discarded values are from the good nodes,

then also strictly less than half of the smallest good values are discarded to obtain the set S_v. Similarly, strictly less than half of the largest good values are discarded to obtain the set S_v. Thus, we have $i_{med} \in S_v$. □

Combining the previous two lemmas, we can state the following theorem.

Theorem 4. *If $n > 3f$, then Algorithm 4 achieves approximate agreement in the* id-only *model.*

Proof. Using Lemma 12, the output range lies within the input range and the first property of the approximate agreement is satisfied.

Using Lemma 13, we have $i_{med} \in S_v$. Thus, we have $\max S_v \geq i_{med}$ and that $\min S_v \leq i_{med}$. Moreover, using Lemma 12, we also get that $\min S_v \geq i_{min}$ and $\max S_v \leq i_{max}$. Therefore, we have that the average $o_v = (\min S_v + \max S_v)/2$ lies within the range $[(i_{min} + i_{med})/2, (i_{med} + i_{max})/2]$. So, the size of output range $(o_{max} - o_{min}) = (i_{max} - i_{min})/2 < (i_{max} - i_{min})$ if $i_{max} \neq i_{min}$. □

IX. SYNCHRONY IS NECESSARY

In our work, we have assumed that the system is synchronous. Intuitively, this is a necessary assumption as a node does not know n and f and hence, the number of messages to wait for before deciding. So, it might end up deciding before receiving a message that was delayed for long, as such the decision might be incorrect. The following lemma proves this for consensus.

Lemma 14. *In an asynchronous system where the number of nodes n and an upper bound f on the number of failures is not known to the nodes, consensus is impossible, even with probabilistic termination.*

Proof. Assume a system $\mathcal{S}$ in which all the nodes are correct. We partition the set of the nodes into sets A and B. A node v has input 1 if $v \in A$; input 0 if $v \in B$. The messages between A and B are arbitrarily delayed. To a node $v \in A$, this is indistinguishable from a system $\mathcal{A}$ where the nodes in B are absent, as v only knows its id initially in both $\mathcal{S}$ and $\mathcal{A}$. Similarly, system $\mathcal{S}$ is indistinguishable to a node $v \in B$ from a system $\mathcal{B}$ where the nodes A are absent. The nodes A decide 0 in the system $\mathcal{A}$ with a non-zero probability, since they only hear from the nodes with the input 0. Similarly, the nodes B decide 1 in the system $\mathcal{B}$ with a non-zero probability. So, the nodes in the system $\mathcal{S}$ decide on different values with a non-zero probability. □

Similar problems can happen in a *semi-synchronous* system [15], where the message delays have a fixed upper bound Δ, but its value is unknown to the nodes. However, the previous argument does not work since we cannot arbitrarily delay the messages due the existence of the fixed upper bound Δ. Instead, we start with the partitions $\mathcal{A}$ and $\mathcal{B}$ and inductively build an invalidating execution for a union of them.

Lemma 15. *In a semi-synchronous system, where the message delays have a fixed upper bound Δ and the nodes do not know*

the value of Δ, n and f, consensus is impossible, even with probabilistic termination.

Proof. Consider a system $\mathcal{A}$ where all the nodes have input 1 and the message delays are at most Δ_a. Each node $v \in \mathcal{A}$ decides 1 with non-zero probability. Let E_a be such an execution in $\mathcal{A}$ of duration T_a. Similarly, consider another system $\mathcal{B}$ where all the nodes have input 0 and the message delays are at most Δ_b. Let E_b be an execution in $\mathcal{B}$ where all the nodes decide 0 in duration T_b. We consider another system $\mathcal{S}$ consisting of $|\mathcal{A}| + |\mathcal{B}|$ nodes, and set the maximum message delay $\Delta_s > \max(\Delta_a, T_a, \Delta_b, T_b)$. We partition the set $\mathcal{S}$ into a set A of $|\mathcal{A}|$ nodes and a set B of $|\mathcal{B}|$ nodes. The nodes in A have input 1 where as the nodes in B have input 0. We also assume some bijective mapping between the sets A and $\mathcal{A}$ and between the sets B and $\mathcal{B}$. We use a' denote the counterpart of a in this bijective map.

We construct an execution E_s from E_a and E_b as follows. If a node $a \in \mathcal{A}$ sends a message to a node $b \in \mathcal{A}$, then $a' \in \mathcal{S}$ sends the same message to b'. The message sent in $\mathcal{S}$ has the same delay as the message sent in $\mathcal{A}$. If a node $a \in \mathcal{A}$ *broadcasts* a message to all the nodes $\mathcal{A}$, then $a' \in \mathcal{S}$ broadcasts the same message to all the nodes $\mathcal{S}$. The delays for the messages broadcast are assigned as follows. The message delay in $\mathcal{S}$ for the messages broadcast to the nodes $A \subset \mathcal{S}$ is same as the delay of those messages in $\mathcal{A}$. The message delay in $\mathcal{S}$ for the messages broadcast to the nodes $B \subset \mathcal{S}$ are Δ_s. Similarly, we assign message actions and delays to the nodes $B \subset \mathcal{S}$. Inductively, a node $a \in A \subset \mathcal{S}$ makes the same decisions as a node $a' \in \mathcal{A}$, since both of them do not know the value of n and f, and node a makes the (same) decision before it even hears from a node in B. Similarly, a node $b \in B \subset \mathcal{S}$ makes the same decisions as a node $b' \in \mathcal{B}$. Therefore, there is an execution E_s in $\mathcal{S}$ so that $a \in \mathcal{S}$ decides 1 and $b \in \mathcal{S}$ decides 0, a disagreement. $\square$

The above argument essentially means that an agreement protocol designed to work without the knowledge of n and f (such as the Bitcoin blockchain [26]), either must assume synchronous execution for guaranteed agreement or sacrifice agreement with some probability.

X. PARALLEL CONSENSUS

In the consensus problem, each correct node had only one opinion and had to output a single opinion in agreement with other nodes. Later, when a correct node can submit multiple opinions, we need to agree on every opinion submitted by a correct node. Therefore, we consider the *parallel consensus* problem: Every correct node v has a set of k_v input pairs (id_v^i, x_v^i) for $1 \le i \le k_v$, where x_v^i is an opinion and id_v is the identifier of the input pair. Each correct node outputs a set of pairs subject to the following conditions.

1) Validity: If (id, x) is an input pair of every correct node and $x \ne \perp$, then all the correct nodes must output the pair (id, x).
2) Agreement: If a correct node v outputs a pair (id_v, x_v), then all other correct nodes must output (id_v, x_v) as well.

3) Termination: Every correct node outputs a set of pairs in finite number of rounds.

Note that the rules allow a pair (id_v, x_v) as an input of a correct node v, but not all the correct nodes, and be absent from the output of every correct node.

First, we describe the $EarlyConsensus(id)$ algorithm, where every correct node v has at most one input pair (id, x_v), i.e., all nodes may not be aware of the identifier id. The pseudocode is given in Algorithm 5. To help a node v distinguish if another node u is aware of id or has no preference or no strong preference of an opinion, we use id:*nopreference* and id:*nostrongpreference* messages.

Next, we describe the $ParallelConsensus$ algorithm using the previous one: The node v starts the $EarlyConsensus(id_v)$ algorithm for every (id_v, x_v) pair input at v. If the node v first hears id:*input*, id:*prefer*, id:*strongprefer* respectively during the second, third, and fifth round of the first phase and no input pair corresponding to id was present at v, then also the node v starts the $EarlyConsensus(id)$ algorithm from that round.

Theorem 5. *The $ParallelConsensus$ algorithm satisfies the parallel consensus properties.*

Proof. Consider a pair (id, x_v) that is input at a correct node v, where $x_v \ne \perp$. In the first round of the phase, the node v broadcasts id:*input*(x_v). So, every correct node hears an id:*input* message in the second round, and fills the missing opinions from the correct nodes with a id:*input*$(\perp)$. In the subsequent rounds, if a correct node u does not receive enough messages to send a id:*prefer* or a id:*strongprefer* message, then it respectively sends a id:*nopreference* and id:*nostrongpreference* message. So, the node v does not fill in a message for u. Therefore, the execution of $EarlyConsensus(id_v)$ is identical to an execution of Algorithm 3, where the input of a correct node v is x_v if (id_v, x_v) is an actual input and $\perp$ if such a pair is absent. Using Theorem 3, every correct node v outputs a pair (id, o_v) in $O(f)$ rounds, so that it is in agreement with other correct nodes, and is same as the input (id, x_v) if it is present at all the correct nodes. Discarding the output pairs of the form $(id, \perp)$ does not affect the agreement and validity properties required by parallel consensus (Line 25).

Now, consider that no correct node has an input pair with the identifier id. If we show that no correct node outputs a pair with the identifier id, then we are done. Let r be the first round when a correct node v receives an id message. If r is the second phase, or the fourth round (rotor-coordinator) of the first phase, then v simply discards it. Otherwise, the round r can be the second (Line 7), third (Lines 12 and 15) or the fifth one (Lines 21 and 24) of the first phase. First, consider that r is the second round of the first phase and a correct node v first received the id:*input* message during round r. Since no other correct node u had an id pair as input, node v fills a default id:*input*$(\perp)$ for every correct node $u \ne v$ and decides to broadcast id:*prefer*$(\perp)$. Similarly, any other correct node $w \ne v$ that first received the id:*input* message in the round r broadcasts id:*prefer*$(\perp)$. In the next round, every correct node receives an id:*prefer*$(\perp)$ message. If a

Algorithm 5 *EarlyConsensus*(*id*) algorithm at node v: The node has at most one input pair (id, x_v). The rotor-coordinator and n_v are initialized as in Algorithm 3. Later, a node only accepts messages from a node if it counted towards n_v during the initialization and discards the messages from the other nodes. The types $M = \{id\text{:}input, id\text{:}prefer, id\text{:}strongprefer\}$ of received messages are counted as follows. If a message of type $m \in M$ is received for the *first* time during the second phase, then it is discarded (considered as not received). If a message of type $m \in M$ is received for the *first* time during the first phase, then the message $m(\bot)$ is substituted for every node u that counted towards n_v during initialization but did not send a type m message. If a node v has received a type $m \in M$ message already during the first phase and a node u that counted towards n_v does not send a type $m' \in M$ message in a subsequent round, then for every such node u, the node v substitutes the message of type m' that it sent most recently.

1: Initialize rotor-coordinator   ▷ Rounds 1 and 2
2: Initialize n_v
3: **while** *true* **do**
4:     **if** Input pair (id, x_v) present and $x_v \neq \bot$ **then**
5:        Broadcast $id\text{:}input(x_v)$ ▷ Next Round
6:     **end if**
7:     **if** Received at least $2n_v/3$ $id\text{:}input(x_v)$ **then** ▷ Next Round
8:        Broadcast $id\text{:}prefer(x_v)$
9:     **else**
10:        Broadcast $id\text{:}nopreference$
11:     **end if**
12:     **if** Received at least $n_v/3$ $id\text{:}prefer(x)$ **then** ▷ Next Round
13:        $id\text{:}x_v = x$
14:     **end if**
15:     **if** Received at least $2n_v/3$ $id\text{:}prefer(x)$ **then**
16:        Broadcast $id\text{:}strongprefer(x)$
17:     **else**
18:        Broadcast $id\text{:}nostrongpreference$
19:     **end if**
20:     Execute a round of rotor-coordinator ▷ Next Round using x_v as v's current opinion. Let c be the value accepted as the coordinator's opinion.
21:     **if** Received less than $n_v/3$ $id\text{:}strongprefer(x)$ **then** ▷ Next Round
22:        $id\text{:}x_v = c$
23:     **end if**
24:     **if** Received at least $2n_v/3$ $id\text{:}strongprefer(x)$ **then**
25:        Terminate and output (id, x) if $x \neq \bot$
26:     **end if**
27: **end while**

correct node heard $id\text{:}prefer$ message for the first time, then it will fill a default $id\text{:}prefer(\bot)$ for every node u that did not send a message to it. If a correct node p already heard an id message, then we know that it sent $id\text{:}prefer(\bot)$ in the previous round and will fill the same for missing opinions. Thus, every correct node p receives at least $2n_p/3$ $id\text{:}prefer(\bot)$ messages, sets $id\text{:}x_p = \bot$ and broadcasts $id\text{:}strongprefer(\bot)$. So, every correct node p receives at least $2n_p/3$ $id\text{:}strongprefer(\bot)$ in the next round, terminates but does not output an id pair since $\bot$ is the associated opinion.

Now, consider that r is the third round of the first phase and a correct node v first hears an $id\text{:}prefer$ message in the round r. The node v fills a default $id\text{:}prefer(\bot)$ opinion for every correct node u, sets $id\text{:}x_v = \bot$ and broadcasts $id\text{:}strongprefer(\bot)$. In the next round, every correct node hears an $id\text{:}strongprefer(\bot)$ message. If a correct node w hears $id\text{:}strongprefer(\bot)$ for the first time, it fills the missing messages with the default $id\text{:}strongprefer(\bot)$ message. If not, the node w fills the missing opinion with what it sent previously, which is again $id\text{:}strongprefer(\bot)$. Thus, every correct node w receives at least $2n_w/3$ $id\text{:}strongprefer(\bot)$ messages and does not output any id pair.

Lastly, consider that r is the fifth round of the first phase and a correct node v first hears an $id\text{:}strongprefer$ message in the round r. No correct node received an id before the round r by assumption, so no correct node sends an id message before round r. So, the node v fills the default $id\text{:}strongprefer(\bot)$ message for every correct node u. Consequently, the node v receives $2n_v/3$ $id\text{:}strongprefer(\bot)$ messages and does not output an id pair.   □

XI. APPLICATION TO DYNAMIC NETWORKS

Building upon the previous algorithms, we can easily devise agreement algorithms for the dynamic networks where the participants enter and leave the system under the influence of an adversary. In the Appendix, we give the algorithms for approximate agreement and total ordering in dynamic networks. The dynamic version of approximate agreement is essentially the same as the static one as the required properties continue to hold in the dynamic case as well.

The total ordering algorithm makes the correct nodes output a sequence or chain of values such that they satisfy the following properties: (1) If a and b are the current chains of any two correct nodes, then either a is a prefix of b or b is a prefix of a. (2) The chain length at every correct node increases over time provided that the a correct node submits a value in every round. The idea is to use an instance of parallel consensus in every round by tagging these messages with the starting round number of the instance. The full algorithm and the proof is in the Appendix.

XII. DISCUSSION

In this paper, we investigated distributed systems where the participants are neither aware of the size n nor the safe estimate f of Byzantine failures. We examined fundamental distributed computing problems such as, approximate agreement, reliable broadcast, rotor-coordinator and consensus; concluding that all of them can be solved with the optimal resiliency of $n > 3f$. Each of these algorithms illustrated a different method

of computing. It is interesting to note that "replacing" f by $n_v/3$ works in these algorithms although $n_v/3$ is an incorrect upper bound on the number of failures. An algorithm using a combination of some of the discussed primitives could be "compiled" to work without the knowledge of n and f keeping resiliency unaffected. We evaluated resiliency in this work but other metrics such as message complexity, round complexity, etc. do not change much either. For example, the message complexity of reliable broadcast is unaffected compared to the original algorithm, the convergence rate of the approximate agreement algorithm remains unchanged and the $O(f)$ round complexity of consensus algorithm is optimal [16].

Removing knowledge of n and f from the participants has other benefits too. For example, we show in Section XI that the design of agreement algorithms for dynamic networks becomes much easier and the nodes do not need to agree on the number of participants in the network. It also opens up ways to achieve agreement in networks without using information from every node. For example, consider a set of nodes that are in approximate agreement with each other already and a new node joins. Then, the new node can execute Algorithm 4 only with a subset of nodes to get closer to the value of most of the nodes. Self-stabilizing algorithms may not need to restore the value of n and f.

It is unclear if the resiliency of rotor-coordinator is optimal, a question left for further work. Also, one could look if these techniques could benefit semi-synchronous or asynchronous dynamic systems where the rate of change of n is controlled, since without having any knowledge about n or f guaranteed agreement is impossible in such systems.

XIII. Acknowledgements

References

[1] Yehuda Afek, James Aspnes, Edo Cohen, and Danny Vainstein. Brief Announcement: Object Oriented Consensus. In *Symposium on Principles of Distributed Computing (PODC), Washington, D.C.*, July 2017.

[2] Eduardo A. P. Alchieri, Alysson Neves Bessani, Joni da Silva Fraga, and Fabíola Greve. Byzantine Consensus with Unknown Participants. In *International Conference On Principles Of Distributed Systems (OPODIS), Luxor, Egypt*, December 2008.

[3] James Aspnes. Notes on Theory of Distributed Systems. Chapter 10, February 2018.

[4] Hagit Attiya and Jennifer Welch. *Distributed Computing: Fundamentals, Simulations, and Advanced Topics*, chapter 5. John Wiley & Sons, 2004.

[5] Nazreen Banu, Samia Souissi, Taisuke Izumi, and Koichi Wada. An Improved Byzantine Agreement Algorithm for Synchronous Systems with Mobile Faults. *International Journal of Computer Applications*, 2012.

[6] Piotr Berman, Juan A. Garay, and Kenneth J. Perry. Towards Optimal Distributed Consensus. In *30th Annual Symposium on Foundations of Computer Science (FOCS), Research Triangle Park, NC*, October 1989.

[7] Piotr Berman, Juan A. Garay, and Kenneth J. Perry. Optimal Early Stopping in Distributed Consensus. In *6th International Workshop on Distributed Algorithms (WDAG), Haifa, Israel*, November 1992.

[8] Silvia Bonomi, Antonella Del Pozzo, Maria Potop-Butucaru, and Sébastien Tixeuil. Approximate Agreement under Mobile Byzantine Faults. In *36th International Conference on Distributed Computing Systems (ICDCS), Nara, Japan*, June 2016.

[9] David Cavin, Yoav Sasson, and André Schiper. Consensus with Unknown Participants or Fundamental Self-Organization. In *International Conference on Ad-Hoc Networks and Wireless (ADHOC-NOW), Vancouver, BC, Canada*, July 2004.

[10] Bernadette Charron-Bost, Matthias Függer, and Thomas Nowak. Approximate Consensus in Highly Dynamic Networks: The Role of Averaging Algorithms. In *42nd International Colloquium on Automata, Languages, and Programming (ICALP), Kyoto, Japan*, July 2015.

[11] Carole Delporte-Gallet, Hugues Fauconnier, Rachid Guerraoui, Anne-Marie Kermarrec, Eric Ruppert, and Hung Tran-The. Byzantine Agreement with Homonyms. In *30th Annual Symposium on Principles of Distributed Computing (PODC), San Jose, California*, June 2011.

[12] Danny Dolev, Keijo Heljanko, Matti Järvisalo, Janne H. Korhonen, Christoph Lenzen, Joel Rybicki, Jukka Suomela, and Siert Wieringa. Synchronous Counting and Computational Algorithm Design. *Journal of Computer and System Sciences*, 2016.

[13] Danny Dolev, Nancy A. Lynch, Shlomit S. Pinter, Eugene W. Stark, and William E. Weihl. Reaching Approximate Agreement in the Presence of Faults. *Journal of the ACM (JACM)*, 1986.

[14] Shlomi Dolev and Jennifer L. Welch. Self-Stabilizing Clock Synchronization in the Presence of Byzantine Faults. *Journal of the ACM (JACM)*, 2004.

[15] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 1988.

[16] Michael J. Fischer and Nancy A. Lynch. A Lower Bound for the Time to Assure Interactive Consistency. *Information Processing Letters*, 1982.

[17] Michael J. Fischer, Nancy A. Lynch, and Michael Merritt. Easy Impossibility Proofs for Distributed Consensus Problems. In *4th Annual ACM Symposium on Principles of Distributed Computing (PODC), Minaki, Ontario, Canada*, August 1985.

[18] Juan A. Garay. Reaching (and Maintaining) Agreement in the Presence of Mobile Faults. In *International Workshop on Distributed Algorithms (WDAG), Terschelling, Netherlands*, September 1994.

[19] Juan A. Garay and Yoram Moses. Fully Polynomial Byzantine Agreement for $n > 3t$ Processors in $t + 1$ Rounds. *SIAM Journal on Computing (SICOMP)*, 1998.

[20] Fabiola Greve and Sebastien Tixeuil. Knowledge Connectivity vs. Synchrony Requirements for Fault-Tolerant Agreement in Unknown Networks. In *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Edinburgh, UK*, June 2007.

[21] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 1982.

[22] Christoph Lenzen and Joel Rybicki. Efficient Counting with Optimal Resilience. In *29th International Symposium on Distributed Computing (DISC), Tokyo, Japan*, October 2015.

[23] Christoph Lenzen and Joel Rybicki. Self-Stabilising Byzantine Clock Synchronisation is Almost as Easy as Consensus. In *International Symposium on Distributed Computing (DISC), Vienna, Autria*, October 2017.

[24] Jennifer Lundelius and Nancy Lynch. A New Fault-Tolerant Algorithm for Clock Synchronization. In *3rd Annual ACM Symposium on Principles of Distributed Computing (PODC), Vancouver, British Columbia, Canada*, August 1984.

[25] Hammurabi Mendes, Maurice Herlihy, Nitin Vaidya, and Vijay K. Garg. Multidimensional Agreement in Byzantine Systems. *Distributed Computing*, 2015.

[26] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. 2008.

[27] T. K. Srikanth and Sam Toueg. Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. *Distributed Computing*, 1987.

[28] Erfan Taheri and Mohammad Izadi. Byzantine Consensus for Unknown Dynamic Networks. *The Journal of Supercomputing*, 2015.

[29] Lewis Tseng and Nitin H. Vaidya. Fault-Tolerant Consensus in Directed Graphs. In *Symposium on Principles of Distributed Computing (PODC), Donostia-San Sebastián, Spain*, July 2015.

In this section, we see how the protocols that we developed can be applied to networks, where the participants enter or leave the system, subject to the constraint that $n > 3f$. First, we look into the approximate agreement problem. We use Algorithm 4 in the dynamic setting as well. It is easy to observe that the Lemmas 12 and 13 apply even if the participants enter and leave the system in every round subject to $n > 3f$. So, the range of correct values still gets halved in every round, with respect to the previous round. However, new nodes entering the system might also increase the range of values at the correct nodes. So, whether the range decreases or increases over time depends on the actual inputs of nodes entering or leaving the system.

Next, we consider the problem of total ordering of events in a dynamic system. We can run the parallel consensus algorithm in every round to agree on the events occurred during that round. We just need to make sure that the set of identifiers used for every parallel consensus instance remains consistent. To do that, we have to specify some more details about the model. The adversary can decide the number of nodes that can join the network *before* every round starts, subject to the constraint that $n > 3f$ remains true when the round starts. Once a node joins the network, it can broadcast to all the nodes that have joined but not left already. A node leaves the network by announcing so to all the participants. A correct node decides itself when to leave. The adversary decides when a faulty node leaves the network. Algorithm 6 lists the pseudocode.

In the following, we show that the nodes agree on the sequences that they output in Line 30. Let T_v^r be the sequence output by a correct node v at the end of round r (Line 30). Our goal is that T_v^r satisfies the following two agreement properties.

1) Chain-prefix: For any pair of correct nodes u, v, either T_u^r is a prefix of T_v^r or T_v^r is a prefix of T_u^r.
2) Chain-growth: For every correct node v, events are appended to T_v^r over time, if a correct node submits an event in every round.

Theorem 6. *Algorithm 6 outputs a chain of events that satisfy the chain-prefix and chain-growth properties.*

Proof. Initially, the node v_0 stores the correct round number 0. By assumption, we have $n > 3f$ in every round. Then, by induction on rounds, selecting the round number based on the majority of received ack messages always returns the correct round number for every correct node. Therefore, every correct node that starts a parallel consensus instance in a round r, tags it with the same identifier r. Each of these instances are then correct using Theorem 5.

Consider a round r' that is final with respect to v. Since each phase of Algorithm 5 is five rounds and the initialization is two rounds, the parallel consensus instance r' terminates by $r' + 5f'_r + 2$ rounds using Theorem 5, where f'_r is the number of faulty nodes in the round r'. Let g'_r be the number of good nodes in the round r' and n'_r be the total number of nodes in the round r'. Since we have $n'_r > 3f'_r$ by assumption,

Algorithm 6 Algorithm at a node v to order events in a dynamic network. Initially, there is one node v_0 in the network, whose r is initialized to 0 and S is initialized to v_0. Since there could be multiple parallel consensus instances running at the same time, we identify them by the round in which they start by appending the round number to the messages. Also, running a parallel consensus instance with respect to S means recording the value of S at the start of the instance, and only accepting the messages from the node identifiers in S, discarding the rest.

```
 1: if v wants to participate then
 2:     Broadcast present                        ▷ Next Round
 3:     Let A_v be the multiset of (ack, t) messages received
        by v in the next round, where t ≥ 0
 4:     Initialize r = r_0 + 1, where (ack, r_0)
        is the majority in A_v
 5:     Initialize S to the identifiers which sent a message in
        A_v
 6: end if
 7: while true do
 8:     r ← r + 1
 9:     I_v^r ← {}
10:     if Received present from u then
11:         S ← S ∪ {u}
12:         Send (ack, r) to u.                   ▷ Next Round
13:     end if
14:     if v wants to stop participating then
15:         Broadcast absent                      ▷ Next Round
16:         Wait and participate in outstanding parallel consensus
            instances until termination
17:     end if
18:     if Received absent from u then
19:         S ← S\{u}
20:     end if
21:     if v witnesses an event m ≠ ⊥ then
22:         Broadcast (m, r)                      ▷ Next Round
23:     end if
24:     if Received (m, r − 1) from u then
25:         I_v^r ← I_v^r ∪ {(u, m)}
26:     end if
27:     Start a parallel consensus instance r      ▷ Next Round
        with the input pairs I_v^r with respect to
        the set S
28:     A round r' < r is final if r − r' > 5|S_v^{r'}|/2 + 2
29:     Let R be the largest round such that all
        the rounds at most R are final
30:     Order the outputs of the consensus
        instances with identifiers at most R
        in the order of increasing identifiers,
        breaking ties arbitrarily
31: end while
```

we have $|S_v^{r'}| \geq g'_r > 2f'_r$. Since r' is final, the current round $r > r' + 5|S_v^{r'}|/2 + 2 > r' + 5f'_r + 2$. So, the parallel consensus instance r' has terminated by the previous round

and no further output from consensus instance r' is produced. Moreover, using Theorem 5, any other correct node $u \neq v$ has also accepted the same output pairs corresponding to the consensus instance id. Also, node u has not accepted any other output pairs corresponding to the consensus instance id, which would contradict the agreement property of parallel consensus. Let R_u and R_v respectively be the value of R computed in Line 29 by the nodes u and v. Then, rounds up to $R_{min} = \min\{R_u, R_v\}$ are final for both the nodes u and v. Thus, the outputs of the consensus instances up to R_{min} is the common prefix of T_u^r and T_v^r, which is the common-prefix property.

Since the parallel consensus instance r' terminates in $O(f_r')$ rounds, the earliest non-final round eventually becomes final and the chain-growth property is satisfied as well. $\qquad\square$