Synchronous Byzantine Agreement with Expected $O(1)$ Rounds, Expected $O(n^2)$ Communication, and Optimal Resilience

Ittai Abraham[1], Srinivas Devadas[2], Danny Dolev[3], Kartik Nayak[1], and Ling Ren[1]

[1] VMware Research – {iabraham,nkartik,lingren}@vmware.com
[2] MIT – devadas@mit.edu
[3] Hebrew University of Jerusalem – danny.dolev@mail.huji.ac.il

Abstract. We present new protocols for Byzantine agreement in the synchronous and authenticated setting, tolerating the optimal number of f faults among $n = 2f + 1$ parties. Our protocols achieve an expected $O(1)$ round complexity and an expected $O(n^2)$ communication complexity. The exact round complexity in expectation is 10 for a static adversary and 16 for a strongly rushing adaptive adversary. For comparison, previous protocols in the same setting require expected 29 rounds and expected $\Omega(n^4)$ communication. We also give a lower bound showing that expected $\Omega(n^2)$ communication is necessary against a strongly rushing adaptive adversary.

1 Introduction

Byzantine consensus [26] is a fundamental problem in distributed computing and cryptography. It has been used to build fault tolerant distributed systems [2, 7, 24, 38, 40], secure multi-party computation [4, 19], and more recently cryptocurrencies [1, 23, 29, 34, 35]. Byzantine consensus considers the problem of reaching agreement among a group of n parties, among which up to f can have Byzantine faults and deviate from the protocol arbitrarily. There exist different formulations for the Byzantine consensus problem.[4] Two classic formulations are Byzantine broadcast and Byzantine agreement [26, 36]. In Byzantine broadcast, there is a designated *sender* who tries to broadcast a value; In Byzantine agreement, every party holds an initial input value. Both problems require all honest parties to eventually commit on the same value. To rule out trivial solutions, both problems have additional validity requirements.

Byzantine agreement and Byzantine broadcast have been studied under various combinations of assumptions, most notably timing assumptions – synchrony, asynchrony or partial synchrony, and setup assumptions – cryptography and public-key infrastructure (PKI). It is now well understood that these assumptions drastically affect the fault tolerance bounds. In particular, Byzantine broadcast and Byzantine agreement both require $f < n/3$ under partial synchrony or asynchrony. But under synchrony with digital signatures and PKI, Byzantine agreement can be solved with $f < n/2$ while Byzantine broadcast can be solved with $f < n - 1$.

In this paper, we consider the synchronous and authenticated (i.e., assuming digital signatures and PKI) setting. The efficiency metrics we consider are (1) round complexity, i.e., the number of rounds of communication before the protocol terminates, and (2) communication complexity, i.e., the amount of information exchanged between parties during the protocol. For convenience, we measure communication complexity using the number of signatures exchanged between parties. Assuming each signature has λ bits, multiplying our communication complexity by λ yields the asymptotic communication complexity in bits.

In the synchronous and authenticated setting, Dolev and Strong gave a deterministic Byzantine broadcast protocol for $f < n - 1$ [11]. Their protocol achieves $f + 1$ round complexity and $O(n^2 f)$ communication complexity. The $f+1$ round complexity matches the lower bound for deterministic protocols [11,14]. To further improve round complexity, randomized protocols have been introduced [3, 13, 37] and culminated in the work of Katz and Koo [21], which solves Byzantine broadcast for $f < n/2$ using expected $O(n^3)$ communication and expected $O(1)$ rounds, and to be precise, expected 29 rounds. Recently, Micali and Vaikuntanathan gave a randomized protocol that solves Byzantine broadcast for $f < n/2$ with $1 - 2^{-\kappa}$ probability using $O(\kappa)$

[4] We use the word "consensus" as a collective term for these variants; other papers have different conventions.

rounds and $O(n^3\kappa)$ communication [31]. Their protocol does not support early termination, so all parties always have to run for $O(\kappa)$ rounds and incur the round/communication complexity above. Furthermore, all of the above protocols solve Byzantine broadcast. To solve Byzantine agreement, the standard technique is to run n instances of Byzantine broadcast in parallel and output the most frequently broadcast value [36]. This adds an additional $\Theta(n)$ factor to communication complexity.

Thus, to the best of our knowledge, existing Byzantine agreement protocols in the synchronous and authenticated setting with optimal resilience all require $\Omega(n^4)$ communication complexity. For round complexity, the best-known protocol (Katz-Koo [21]) requires expected 29 rounds. We improve communication complexity to $O(n^2)$ and round complexity to 16. To the best of our knowledge, our protocol is the first to simultaneously achieve (i) $O(1)$ rounds, (ii) expected $O(n^2)$ communication, and (iii) optimal fault tolerance of $n = 2f + 1$ in the synchronous and authenticated setting. In fact, we do not know of any existing protocol that achieves both (ii) and (iii).

Our protocols use threshold signatures [6, 39] to reduce communication complexity and a random leader election subroutine to reduce round complexity. The random leader election subroutine can be constructed using verifiable random functions (VRF) [29,30], which can be instantiated from RSA [30] or bilinear maps [16]. Alternatively, we can use threshold coin-tossing schemes [6, 28]. Both approaches require a single round and $O(n^2)$ communication. We achieve the following result.

Theorem 1. *Assuming verifiable random functions, synchronous authenticated Byzantine agreement can be solved for $f < n/2$ with*

- *expected 10 rounds and expected $O(n^2)$ communication against a static adversary,*
- *expected 16 rounds and expected $O(n^2)$ communication against a strongly rushing adaptive adversary.*

It is worth noting that our protocols work even in the presence of a very powerful adversary, which we call a *strongly rushing adaptive adversary*. The adversary can adaptively decide which f parties to corrupt and when to corrupt them. And by "strongly rushing", we mean that if the adversary decides to corrupt a party h after observing messages sent from h to any other party in round r, it can remove h's round-r messages from the network before they reach other honest parties. In comparison, a standard rushing adversary can decide its own round-r messages after learning honest parties' round-r messages, but if it corrupts h in round r, it cannot "take back" or alter h's round-r messages to other parties. The Dolev-Strong, Katz-Koo, and Micali-Vaikuntanathan protocols all work against such a strongly rushing adaptive adversary.

The $O(1)$ expected round complexity is clearly asymptotically optimal. A natural question is whether or not the quadratic communication complexity can be further improved. Dolev and Reischuk [10] were the first to investigate this question. They showed that $\Omega(f^2)$ messages are necessary for *deterministic* protocols. When $f = \Theta(n)$, it translates to a $\Omega(n^2)$ lower bound. King-Saia [22] and Chan-Pass-Shi [8] circumvent this lower bound and achieve sub-quadratic communication by allowing imperfect security, i.e., by introducing failure probability. In this work, building on Dolev and Reischuk [10], we show that even after allowing a constant failure probability, $\Omega(n^2)$ expected messages are necessary against a *strongly rushing* adaptive adversary. Specifically, we show the following lower bound:

Theorem 2. *If a protocol solves Byzantine broadcast with $\frac{1}{2} + \epsilon$ probability against a strongly rushing adaptive adversary, then in expectation, honest parties need to send at least $(\epsilon f/2)^2$ messages.*

Note that even though the lower bound is stated for Byzantine broadcast, it applies to Byzantine agreement as well, because Byzantine broadcast can be achieved by having the sender broadcast its value and then running one instance of Byzantine agreement. Not surprisingly, the sub-quadratic solutions by King-Saia [22] and Chan-Pass-Shi [8] only work against a standard rushing adaptive adversary but not a strongly rushing adaptive one.

We remark that, besides failure probability and security against a strongly rushing adversary, both King-Saia and Chan-Pass-Shi protocols also sacrifice optimal resilience (cf. Section 1.1 and Table 1). An interesting open question is whether Byzantine broadcast/agreement can be solved with sub-quadratic *expected* communication complexity while preserving *optimal resilience*. We leave it to future work.

Table 1: **Comparison with closely related works.** Here, κ is a security parameter and ϵ is a positive constant. We assume all protocols have access to ideal digital signatures. Dolev-Strong, Katz-Koo, and Micali-Vaikuntanathan solve Byzantine broadcast; if transformed into Byzantine agreement protocols with standard techniques, their communication complexity increase by a factor of n.

Protocol	Exp. Comm. Complexity	Exp. Round Complexity	Resilience	Failure Probability	Adversarial Model
Dolev-Strong [11]	$O(n^2 f)$	$f+1$	$f < n-1$	0	strongly rushing adaptive
Katz-Koo [21]	$O(n^3)$	29	$f < n/2$	0	strongly rushing adaptive
Micali-Vaikuntanathan [31]	$O(n^3\kappa)$	$O(\kappa)$	$f < n/2$	$2^{-\kappa}$	strongly rushing adaptive
King-Saia [22]	$n^{1.5} \cdot \mathsf{poly}\log(n)$	$\mathsf{poly}\log(n)$	$f < (\frac{1}{3} - \epsilon)n$	$1/\mathsf{poly}(n)$	rushing adaptive
Chan-Pass-Shi [8]	$n \cdot \mathsf{poly}\log(\kappa)$	$O(1)$	$f < (\frac{1}{2} - \epsilon)n$	$\mathsf{negl}(\kappa)$	rushing adaptive
This work (upper bound)	$O(n^2)$	16	$f < n/2$	0	strongly rushing adaptive
This work (lower bound)	$> (\epsilon f/2)^2$	N/A	any f	$\frac{1}{2} - \epsilon$	strongly rushing adaptive

1.1 Related Work

Byzantine agreement and broadcast were first introduced by Lamport, Shostak and Pease [26, 36]. They presented protocols and fault tolerance bounds for two settings (both synchronous). Without cryptographic assumptions (the unauthenticated setting), Byzantine broadcast and agreement can be solved if and only if $f < n/3$. Assuming digital signatures (the authenticated setting), Byzantine broadcast can be solved if $f < n-1$ and Byzantine agreement can be solved if and only if $f < n/2$. The protocols from [26, 36] had exponential message complexities. Fully polynomial protocols were later shown for both the authenticated ($f < n/2$) [11] and the unauthenticated ($f < n/3$) [18] settings. Both protocols require $f+1$ rounds of communication, which matches the lower bound on round complexity for deterministic protocols [11, 14]. To circumvent the $f+1$ round lower bound, a line of work explored the use of randomization [3, 37] which eventually led to expected constant-round protocols for both the authenticated ($f < n/2$) [21] and the unauthenticated ($f < n/3$) [13] settings. In the asynchronous setting, the FLP impossibility [15] rules out any deterministic solution. Some works use randomization [3, 5, 6] or partial synchrony [12] to circumvent the impossibility.

We have already compared with closely related works by Dolev-Strong, Katz-Koo, and Micali-Vaikuntanathan. Table 1 summarizes them and compares with our results. For completeness, we mention that the numbers in Table 1 have considered adding threshold signatures to those works. For Dolev-Strong [11], there is no obvious way to utilize threshold signatures because each round uses a different threshold: a valid message in round k must be signed by k replicas. Katz-Koo [21] and Micali-Vaikuntanathan [31] both invoke n instances of gradecasts in parallel. Threshold signatures can improve one gradecast instance to $O(n^2)$, so n parallel gradecast instances result in $O(n^3)$ communication complexity.

The King-Saia protocol [22] solves synchronous Byzantine agreement for $f < (\frac{1}{3} - \epsilon)n$ with $1/\mathsf{poly}n$ failure probability and uses $n^{1.5} \cdot \mathsf{poly}\log(n)$ communication. Recently, Chan, Pass and Shi [8] have shown a protocol that solves *binary* Byzantine agreement for $f < (\frac{1}{2} - \epsilon)n$ with $\mathsf{negl}(\kappa)$ failure probability in expected $O(1)$ rounds and $n \cdot \mathsf{poly}\log(\kappa)$ communication [8]. Intuitively, both protocols save communication by down-sampling the population into a smaller committee and running a Byzantine agreement protocol within the committee. Their techniques are orthogonal and complementary to our work since they still need a protocol for the committee. Interestingly, the protocol run by the committee in Chan-Pass-Shi is inspired by the protocol described in this work.

1.2 Technical Overview

We first describe our core protocol, which ensures agreement (safety)[5] and termination as required by Byzantine broadcast/agreement, but provides a weak notion of validity. Specifically, it achieves

- **Termination:** all honest parties eventually commit,
- **Agreement/safety:** all honest parties commit on the same value, and
- **Validity:** if all honest parties start with certificates for the same value v, and no Byzantine party starts with a certificate for a contradictory value, then all honest parties commit on v.

In Section 5 we will describe how to obtain these certificates to solve Byzantine broadcast or Byzantine agreement.

The core protocol runs in iterations. In each iteration, a unique leader is elected. Each new leader picks up the state left by previous leaders and proposes a value in its iteration. Parties then cast votes on the leader's value v. Ideally, if the leader is honest, all honest parties commit v upon receiving $f+1$ votes for v at the end of that iteration. A Byzantine leader can easily waste its iteration by not proposing. But it can also perform the following more subtle attacks: (1) send contradicting proposals to different honest parties, or (2) send a proposal to some but not all honest parties. We must ensure these Byzantine behaviors do not violate safety.

The need for equivocation checks. To ensure safety in the first attack, parties engage in an all-to-all round of communication to forward the leader's proposal to each other for an equivocation check. If a party detects leader equivocation, i.e., sees two conflicting signed proposals from the leader, it does not commit even if it receives $f+1$ votes.

The need for a notify round. Using the second attack, a Byzantine leader can make some, but not all, honest parties commit on a value v. If the other honest parties do not know that v has been committed, they may commit $v' \neq v$ in a subsequent iteration. Therefore, whenever an honest party h commits on a value v, h needs to *notify* all other honest parties of its commit. h can do this by broadcasting the $f+1$ votes it received. When another party h' receives such a notification, it "accepts" the value v. If a party has accepted v and receives a proposal $v' \neq v$ in a later iteration, it will not vote for v' unless it is shown a proof that voting for v' is safe. The details can be found in Section 3.

Safety, termination, and validity. Safety is preserved because when an honest party commits, (1) no other party can commit a different value in the same iteration (due to equivocation checks), and (2) no other value can gather enough votes in subsequent iterations (due to notify by the honest party). Validity follows from a similar argument: if all honest parties start the protocol with the same certified (i.e., accepted) value v and Byzantine parties do not have a different certified value, only v can gather enough votes. Termination is achieved when some honest party h receives $f+1$ notify messages. At this point, h sends these $f+1$ notifications to all other parties and terminates. The $f+1$ notifications h sends will ensure termination of all other parties in the next round. If an honest leader emerges, all parties terminate in its iteration.

Round complexity and communication complexity. Since there are $f+1$ honest out of $2f+1$ parties, by electing a random leader in every iteration, the protocol terminates in 2 iterations in expectation. Depending on the adversarial model, each iteration ranges from 4 to 7 rounds. The communication bottleneck happens at rounds that require all-to-all exchanges. In these rounds, each message is either a single signature or $f+1$ signatures on the same message. The latter can be reduced to $O(1)$ using threshold signatures. Thus, the protocol runs in expected $O(1)$ rounds and uses expected $O(n^2)$ communication.

Paxos, PBFT, and our protocol. Abstractly, this core protocol resembles the synod algorithm in Paxos [25] but is adapted to the synchronous and Byzantine setting. The main idea of the synod algorithm is to ensure *quorum* intersection [25] at one *honest* party. The core idea of Paxos is to form a quorum of size $f+1$ before committing a value. With $n = 2f+1$, two quorums always intersect at one party, which is honest in Paxos. This honest party in the intersection will force a future leader to respect the committed value. In order to tolerate f Byzantine faults, PBFT [7] uses quorums of size $2f+1$ out of $n = 3f+1$, so that two quorums

[5] We will refer to the agreement requirement as "safety" for the rest of the paper.

intersect at $f+1$ parties, among which one is guaranteed to be honest. Similar to PBFT, we also need to ensure quorum intersection at $f+1$ parties. But this requires new techniques with $n = 2f+1$ parties in total. On the one hand, an intersection of size $f+1$ seems to require quorums of size $1.5f+1$. (An subsequent work called Thunderella [35] uses quorums of size $1.5f+1$ to improve the optimistic case.) On the other hand, a quorum size larger than $f+1$ (the number of honest parties) seems to require participation from Byzantine parties and thus loses liveness. As described in the core protocol, our synchronous *notify* round forms a *post-commit quorum* of size $2f+1$, which intersects with any *pre-commit quorum* of size $f+1$ at $f+1$ parties. This satisfies the requirement of one honest party in the intersection. Moreover, since parties in the post-commit quorum only receive messages, liveness is not affected.

Achieving Byzantine broadcast and Byzantine agreement. The core protocol already ensures safety and termination, so we only need some technique to boost its weaker validity to what Byzantine broadcast/agreement require. Our protocol achieves this using a single round of all-to-all communication before invoking the protocol. This allows us to avoid the standard transformation of composing n parallel Byzantine broadcasts to achieve Byzantine agreement. As a result, our Byzantine agreement protocol has the same asymptotic round/communication complexity as the core protocol.

A case for synchronous protocols. Synchronous protocols are often considered an unrealistic assumption in practice. This may be because (i) an adversary can attack the network to violate the synchrony assumption, (ii) the protocol can not progress at the actual speed of the network, or (iii) synchronous protocols require parties to run in lock-step rounds. In Section 7, we explain that the first two concerns are shared by Bitcoin. The success of Bitcoin shows that a protocol with a conservative message delay bound, though slow, can still be safe and useful in practice. For the last concern, we give a clock synchronization protocol to bootstrap lock-step synchrony from bounded message delay.

2 Model

We assume synchrony. If an honest party i sends a message to another honest party j at the beginning of a round, the message is guaranteed to reach by the end of that round. We describe the protocol assuming lock-step execution, i.e., parties enter and exit each round simultaneously. Later in Section 7, we will present a clock synchronization protocol to bootstrap lock-step execution from bounded message delay.

We assume digital signatures and trusted setup. In the trusted setup phase, a trusted dealer generates public/private key pairs for digital signatures and other cryptographic primitives for each party, and certifies each party's public keys. We use $\langle x \rangle_i$ to denote a message x signed by party i, i.e., $\langle x \rangle_i = (x, \sigma)$ where σ is a signature of message x produced by party i using its private signing key. For efficiency, it is customary to sign the hash digest of a message. A message can be signed by multiple parties (or the same party) in layers, i.e., $\langle \langle x \rangle_i \rangle_j = \langle x, \sigma_i \rangle_j = (x, \sigma_i, \sigma_j)$ where σ_i is a signature of x and σ_j is a signature of $x \mid\mid \sigma_i$ ($\mid\mid$ denotes concatenation). When the context is clear, we omit the signer and simply write $\langle x \rangle$ or $\langle \langle x \rangle \rangle$.

We require a random leader election subroutine. As mentioned, this subroutine can be instantiated using random verifiable functions [29] or threshold coin-tossing [6]. It may also be left to higher level protocols. For example, a cryptocurrency may elect leaders based on proof of work.

We assume a strongly rushing adaptive adversary. After the trusted setup phase, the adversary can adaptively decide which f parties to corrupt and when to corrupt each of them as the protocol executes. Note, however, that the adversary is not *mobile*: it cannot un-corrupt a Byzantine party to restore its corruption budget. The adversary is also strongly rushing. In each round, the adversary observes any party i's message to any other party j. If the adversary decides to corrupt i at this point, it controls which other honest parties (if any) i sends messages to and what messages i sends them *in that round*.

3 A Synchronous Byzantine Synod Protocol

Our core protocol is a synchronous Byzantine synod protocol with $n = 2f+1$ parties. The goal of the core synod protocol is to guarantee that all honest parties eventually commit (termination) on the same value

(agreement). In addition, it achieves the following notion of validity: if (1) all honest parties start with the same value and have a *certificate* for this value, and (2) the adversary does not start with a certificate for a contradictory value, then all honest parties commit on this value. In Section 5, we show how to obtain these certificates using a single pre-round to achieve Byzantine broadcast and Byzantine agreement.

For ease of exposition, we will temporarily assume a static adversary in Section 3.1 while presenting the core protocol. A static adversary has to decide which parties to corrupt after the trusted setup phase and before the protocol starts. We will also temporarily assume a trusted random beacon that selects a random leader L_k for each iteration k and informs all parties of its selection.

Each iteration consists of 4 rounds. The first three rounds are conceptually similar to Paxos and PBFT: (1) the leader learns the states of the system, (2) the leader proposes a value, and (3) parties vote on the value. If a party receives $f+1$ votes for the same value and does not detect leader equivocation, it commits on that value. We then add another round: (4) if a party commits, it notifies all other parties about the commit; upon receiving a notification, other parties *accept* the committed value and will vouch for that value to future leaders. We now describe the protocol in detail.

3.1 Detailed Protocol

When a leader proposes a value v in iteration k, we say the proposal has rank k and write them as a tuple (v, k). The first iteration has $k = 1$. Each party i internally maintains states $\mathsf{accepted}_i = (v_i, k_i, \mathcal{C}_i)$ across iterations to record its *accepted* proposal. Initially, each party i initializes $\mathsf{accepted}_i := (\perp, 0, \perp)$. If party i later accepts (v, k), it sets $\mathsf{accepted}_i := (v, k, \mathcal{C})$ such that $\mathcal{C}$ *certifies* that v is legally accepted in iteration k. $\mathcal{C}$ consists of $f+1$ commit requests for proposal (v, k) (see the protocol for details). We also say $\mathcal{C}$ certifies, or is a certificate for, (v, k). Proposals are ranked by the iteration number in which they are made. Namely, (v, k) is ranked higher than, lower than, or equal to (v', k') if $k > k'$, $k < k'$ and $k = k'$, respectively. Certificates are ranked by the proposals they certify. When we say a party "broadcasts" a message, we mean it sends the message to all parties including itself.

Round 1 (status) Each party i sends a $\langle k, \mathsf{status}, v_i, k_i, \mathcal{C}_i \rangle_i$ message to L_k to report its current accepted value. We henceforth write L_k as L for simplicity.
At the end of this round, if party i reports the highest certificate to L (i could be L itself), L sets $\mathsf{accepted}_L = (v_L, k_L, \mathcal{C}_L) := (v_i, k_i, \mathcal{C}_i)$. If no party reports a certificate, L chooses v_L freely and sets $k_L := 0$ and $\mathcal{C}_L := \perp$.

Round 2 ($\mathsf{propose}$) L broadcasts a signed proposal $\langle \langle k, \mathsf{propose}, v_L \rangle_L, k_L, \mathcal{C}_L \rangle_L$.
At the end of this round, party i sets $v_{L \to i} := v_L$ if the certificate it receives in the above leader proposal is no lower than what i reported to the leader, i.e., if $k_L \geq k_i$. Otherwise (leader is faulty), it sets $v_{L \to i} := \perp$.

Round 3 (commit) If $v_{L \to i} \neq \perp$, then party i forwards the proposal $\langle k, \mathsf{propose}, v_{L \to i} \rangle_L$ to all other parties and broadcasts a $\langle k, \mathsf{commit}, v_{L \to i} \rangle_i$ request.
At the end of this round, if party i is forwarded a properly signed proposal $\langle k, \mathsf{propose}, v' \rangle_L$ in which $v' \neq v_{L \to i}$, it does not commit in this iteration (leader has equivocated). Else, if party i receives $f+1$ $\langle k, \mathsf{commit}, v \rangle_j$ requests in all of which $v = v_{L \to i}$, it commits on v and sets its internal state $\mathcal{C}_i$ to be these $f+1$ commit requests concatenated. In other words, party i commits if and only if it receives $f+1$ matching commit requests and does not detect leader equivocation.

Round 4 (notify) If party i has committed on v at the end of the previous round, it sends a notification $\langle \langle \mathsf{notify}, v \rangle_i, \mathcal{C}_i \rangle_i$ to every other party.
At the end of this round, if party i receives a $\langle \langle \mathsf{notify}, v \rangle_j, \mathcal{C} \rangle_j$ message, it accepts v by setting $\mathsf{accepted}_i = (v_i, k_i, \mathcal{C}_i) := (v, k, \mathcal{C})$. If party i receives multiple valid notify messages with different values (how this can happen is explained at the end of Section 3.2), it can accept an arbitrary one. Lastly, party i increments the iteration counter k and enters the next iteration.

Early and non-simultaneous termination. At any point during the protocol, if a party gathers *notification headers* (excluding certificates) $\langle \mathsf{notify}, v \rangle$ from $f+1$ distinct parties, it sends these $f+1$ notification headers to all other parties and terminates. This ensures that when the first honest party terminates, all other honest parties receive $f+1$ notification headers and terminate in the next round.

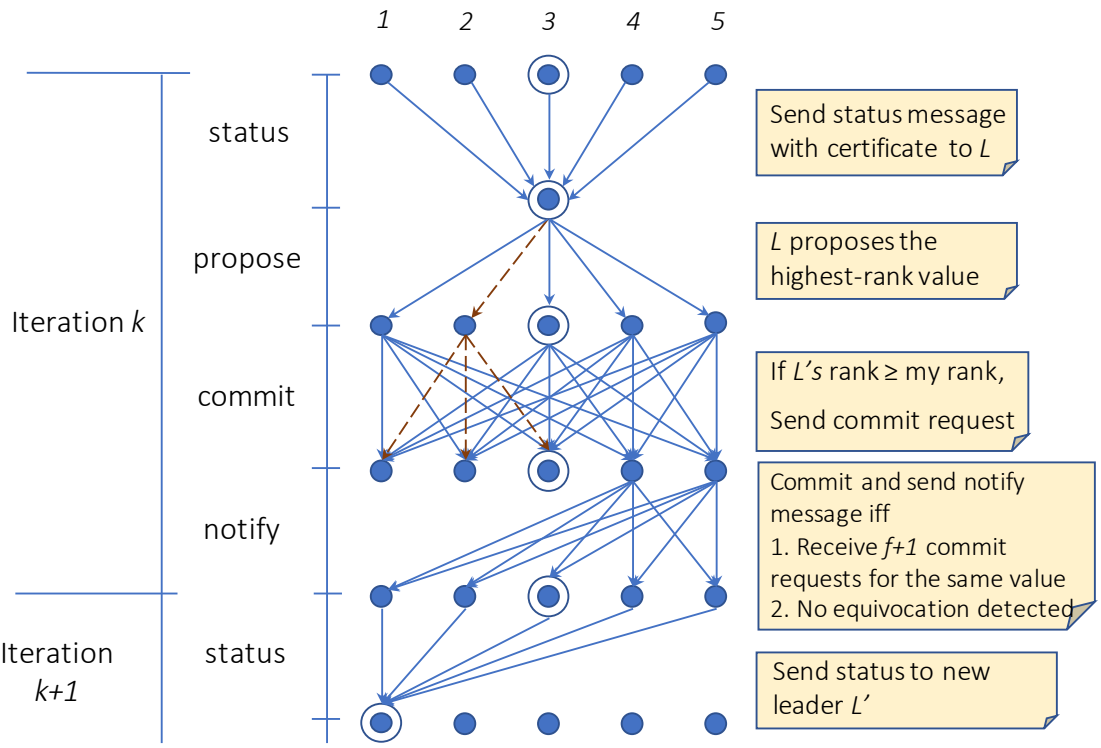

Fig. 1: An example iteration of the core protocol. In this example, $f = 2$, $n = 2f + 1 = 5$, parties 3 and 4 are Byzantine. **1.** (status) Each party sends its current states to $L = 3$. **2.** (propose) No party has committed or accepted any value, so L can propose any value of its choice. L equivocates and sends one proposal to party 4 (shown by dashed red arrow) and a different proposal to honest parties. **3.** (commit) Honest parties forward L's proposal and send commit requests to all parties. Party 4 only sends to parties $\{3, 4, 5\}$. Parties 1 and 2 receive $f + 1$ commit requests for the blue value and do not detect equivocation, so they commit. Party 5 detects leader equivocation and does not commit despite also receiving $f + 1$ commit requests for the blue value. **4.** (notify) Parties 1 and 2 notify all other parties. On receiving a valid notification, party 5 accepts the blue value. **5.** (status) The parties send status messages to the new leader $L' = 1$ for iteration $k + 1$.

3.2 Safety, Termination, and Validity

In this section, we prove that the core protocol in Section 3.1 provides safety, termination and a weak notion of validity.

Safety. We first give some intuition to aid understanding. The scenario to consider for safety is when an honest party h commits on a value v^* in iteration k^*. We first show that Byzantine parties cannot commit or accept a value other than v^* in iteration k^*. Thus, all other honest parties accept v^* at the end of iteration k^* upon receiving notify from the honest party h. Thus, a value other than v^* cannot gather enough votes in iteration $k^* + 1$, and hence cannot be committed or accepted in iteration $k^* + 1$, and hence cannot gather enough votes in iteration $k^* + 2$, and so on. Safety then holds by induction.

We now formalize the above intuition by proving the following lemma about certificates: once an honest party commits, all certificates in that iteration and future iterations can only certify its committed value.

Lemma 1. *Suppose party h is the first honest party to commit and it commits on v^* in iteration k^*. If a certificate $\mathcal{C}$ for (v, k^*) exists, then $v = v^*$.*

Proof. $\mathcal{C}$ must consist of $f+1$ commit requests for v. At least one of these comes from an honest party (call it h_1). Thus, h_1 must have received a proposal for v from the leader, and must have forwarded the proposal to all other parties. If $v \neq v^*$, h would have detected leader equivocation, and would not have committed on v^* in this iteration. So we have $v = v^*$.

Lemma 2. *If at the start of iteration k, (1) every honest party i has a certificate for (v, k_i), and (2) all conflicting certificates are lower ranked, i.e, any certificate for (v', k') where $v \neq v'$ must have $k' < k_i$ for all honest i, then the above two conditions will hold at the end of iteration k.*

Proof. Suppose for contradiction that some party (honest or Byzantine) acquires a *higher* certificate than what it had previously for $v' \neq v$. Then it must receive from one honest party (call it h) a $\langle k, \mathsf{commit}, v' \rangle_h$ request in iteration k. Note that h has a certificate for (v, k_h) at the start of iteration k. In order for h to send a commit request for v', the leader L_k must show a certificate for (v', k') such that $k' \geq k_h$, which contradicts condition (2).

A simple induction shows that the above two conditions, if true at the start of an iteration, will hold true forever.

Theorem 3 (Safety). *If two honest parties commit on v and v' respectively, then $v = v'$.*

Proof. Suppose party h is the first honest party to commit, and it commits on v^* in iteration k^*. After the notify round of iteration k^*, every honest party receives a certificate for (v^*, k^*) and accepts v^*. Furthermore, due to Lemma 1, there cannot be a certificate for (v, k^*) in iteration k^* for $v \neq v^*$. Thus, the two conditions in Lemma 2 hold at the end of iteration k^*. So no certificate for a value other than v^* can be formed from this point on. In order for an honest party to commit on v, there must be a certificate for (v, k) where $k \geq k^*$. Therefore, $v = v^*$. Similarly, $v' = v^*$, and we have $v = v'$.

Termination. We now show that an honest leader will guarantee all honest parties terminate by the end of that iteration.

Theorem 4 (Termination). *If the leader L_k in iteration k is honest, then every honest party terminates one round after iteration k (or earlier).*

Proof. The honest leader L_k will send a proposal to all parties. It will propose a value reported by the highest certificate it collects in the status round. This certificate will be no lower than any certificate held by honest parties. Additionally, the unforgeability of digital signatures prevents Byzantine parties from falsely accusing L of equivocating. Therefore, all honest parties will send commit requests for v, receive $f+1$ commit requests for v, commits on v, send notification headers for v, receive $f+1$ notification headers for v (this is the end of iteration k), and terminate in the next round. (It is possible that they receive $f+1$ notification headers and terminate at any earlier time.)

Validity. We now discuss the validity achieved by our core protocol. In the theorem, we assume the existence of initial certificates for $(v, 0)$ that are input to our core protocol. These initial certificates will be provided by higher-level protocols that invoke the core protocol (c.f. Section 5).

Theorem 5 (Validity). *All honest parties will commit on v if (1) every honest party starts with an initial certificate $\mathcal{C}$ certifying v, and (2) no Byzantine party has a certificate $\mathcal{C}'$ certifying $v' \neq v$.*

Proof. The proof is straightforward from Lemma 2 and Theorem 4. The input constraints satisfy the two conditions for Lemma 2 with each $k_i = 0$. Due to Lemma 2, for all subsequent iterations, only v can have certificates and thus, only v can be committed. By Theorem 4, when an honest leader emerges, all honest parties will commit on v.

Finally, we mention an interesting scenario that does not have to be explicitly addressed in the proofs. Before any honest party commits, Byzantine parties may obtain certificates for multiple values in the same iteration. In particular, the Byzantine leader proposes two values v and v' to all the f Byzantine parties. (An example with more than two values is similar.) Byzantine parties then exchange f commit requests for both values among them. Additionally, the Byzantine leader proposes v and v' to different honest parties. Now with one more commit request for each value from honest parties, Byzantine parties can obtain certificates for both v and v', and can make honest parties accept different values by showing them different certificates (notify messages). However, this will not lead to a safety violation because no honest party would have committed in this iteration: the leader has equivocated to honest parties, so all honest parties will detect equivocation from forwarded proposals and thus refuse to commit. This scenario showcases the necessity of both the synchrony assumption and the use of digital signatures for our protocol. Lacking either one, equivocation cannot be reliably detected and any protocol will be subject to the $f < n/3$ bound. For completeness, we note that the above scenario will not lead to a violation of the termination property, either. At the end of the iteration, honest parties may accept either value. But in the next iteration, they can still vote for either value despite having accepted the other, since the two values have the same rank.

4 Random Leader Election using Verifiable Random Function

4.1 Background on Verifiable Random Function (VRF)

A verifiable random function F is a pseudorandom function that provides a proof of its correct computation [30]. Given a secret key sk, one can evaluate F on any input x and obtain a pseudorandom output $y = F_{sk}(x)$ together with a proof π. From π and the corresponding public key pk, anyone can verify that y is correctly computed from x and sk, in which case $\mathrm{Ver}_{pk}(x, y, \pi) = 1$. Additionally, a VRF needs to satisfy *uniqueness*: there do not exist $x, y_1, y_2, \pi_1, \pi_2$ such that $y_1 \neq y_2$ but $\mathrm{Ver}_{pk}(x, y_2, \pi_1) = \mathrm{Ver}_{pk}(x, y_2, \pi_2) = 1$. Hence, a VRF is also called a unique signature. Efficient constructions for VRFs have been described in [9, 16].

4.2 Against a Static Adversary

At a high level, random leader election using VRF works as follows in our protocol. In each iteration, each party i evaluates the VRF on the iteration number k. The VRF output $y_i = F_{sk_i}(k)$ serves as a *secondary rank* of party i's proposal. Namely, proposals now have the form (v, k, y) where v is the proposed value, k is the iteration number as before, and y is the candidate leader's VRF output. Naturally, commit messages, certificates, and notify messages now additionally include the proposer's VRF output y as well. Proposals and certificates are ranked by k first and then by y. Every party should act as a leader in the status round and the propose round of each iteration to collect states and make a proposal. The corresponding VRF proofs are then sent in a new elect round. The party with the largest VRF output is the real leader of that iteration.

The above idea works against a static adversary after the necessary modifications described in this subsection. Achieving expected constant rounds against an adaptive adversary requires additional techniques, which we present in Section 4.3.

In the static case, the elect round happens concurrently with the propose round. In the commit round, each party forwards the highest ranked proposal. A party sends a commit request for the highest ranked proposal it receives in each iteration (unless it has seen a certificate for an even higher proposal in previous iterations). At the end of the commit round, a party commits a proposal (v, k, y) in iteration k, if (1) it receives $f + 1$ matching commit requests (could be virtual) for the proposal, (2) it does not receive a higher ranked or equally ranked proposal. Receiving a higher ranked proposal implies that the certified proposal is not from real leader of the iteration; receiving an equally ranked proposal implies proposer equivocation. At the end of the notify round, party i accepts the highest proposal with a certificate.

If an honest party has the highest rank in an iteration, every honest party will receive its VRF rank and will consider it the leader. Thus, termination holds without any change. However, if a Byzantine party has the highest rank, it can selectively withhold its VRF rank from other parties. This may cause honest parties

to have different views on the identity of the leader. We now show such a misbehavior does not affect safety or validity.

Lemma 3 (cf. Lemma 1). *Suppose replica h is the first honest replica to commit and it commits on (v^*, k^*, y^*) in iteration k^*. If a certificate $\mathcal{C}$ for (v, k^*, y) exists, then either $y < y^*$ or $v = v^*$.*

Proof. Similar to the proof of Lemma 1, for such a cert $\mathcal{C}$ to exist, a honest replica h_1 must have sent a commit request for (v, k^*, y). Thus, h_1 must have received a proposal for (v, k^*, y) and must have forwarded it to all other replicas. If (v, k^*, y) is higher ranked than (v^*, k^*, y^*), or if they are equally ranked but $v \neq v^*$, then h would not have committed. Therefore, either $y < y^*$ or $v = v^*$.

The statements and proofs of Lemma 2, Theorem 3 and Theorem 5 remain unchanged, once we augment proposal ranks with VRF outputs. We do not repeat them.

4.3 Against an Adaptive and Strongly Rushing Adversary

The protocol presented so far does not achieve expected constant rounds against an adaptive adversary. The adversary learns who the leader L_k is after the elect round in iteration k. It can then immediately corrupt L_k and prevent it from sending any proposal. This way, the adversary forces the protocol to run for f iterations.

A first modification towards adaptive security is to move the elect round after the propose round and before the commit round. The hope is that, by the time L_k is corrupted, all honest parties have already received their proposal. However, this idea alone is not sufficient. At the end of the elect round, after learning the identity of L_k, the adversary corrupts L_k, signs an equivocating proposal using L_k's secret key and forwards it to all honest parties. Honest parties will detect equivocation from L_k and will not commit in this iteration. We are again forced to run the protocol for f iterations.

To this end, we need to add a step for each party to "prepare" its proposal before the leader is revealed. Afterwards, only "prepared" proposals are considered in equivocation checking. The prepare step should guarantee that, if a party h is honest throughout the prepare process but becomes corrupted afterwards, an adversary cannot construct a "prepared" equivocating proposal on h's behalf. We achieve the prepare step in two rounds as follows.

Round P1 ($\mathsf{prepare}_1$) Each party i broadcasts its proposal $\langle v_i, k \rangle_i$.
Round P2 ($\mathsf{prepare}_2$) If party j receives a proposal $\langle v_i, k \rangle_i$ from party i in the previous round, party j signs the proposal and sends $\langle v_i, k \rangle_j$ back to party i.

We say a proposal (v_i, k) is prepared if it carries $f + 1$ signatures from distinct parties. Each honest party will be able to prepare its proposal. If party i is honest in the two prepare rounds and becomes corrupted only afterwards, preparing a conflicting proposal on party i's behalf requires forging at least one honest party's signature, which a computationally bounded adversary cannot do.

The core protocol against an adaptive and strongly rushing adversary now has 7 rounds: status, $\mathsf{prepare}_1$, $\mathsf{prepare}_2$, propose, elect, commit, and notify. Proofs for safety and validity remain unchanged from the static case. Proof of termination and round complexity analysis also hold once we observe that (1) there is a $> 1/2$ chance that each leader L_k is honest up to the point at which it is revealed, and (2) if L_k is still honest by the end of the propose round of iteration k, all honest parties will consider its proposal valid and terminate one round after iteration k.

4.4 Round Complexity and Communication Complexity

The first honest leader will ensure termination. The random leader election subroutine ensures a $(f+1)/(2f+1) > 1/2$ probability that each leader is honest, so the core protocol terminates in expected 2 iterations, plus one extra round to forward $f + 1$ notify. Thus, if an iteration requires r rounds, our core protocol requires $2r + 1$ rounds to terminate in expectation. If the adversary is adaptive and strongly rushing, each iteration requires $r = 7$ rounds. If the adversary is adaptive and normal rushing, the elect round can happen in parallel

to propose, and each iteration has $r = 6$ rounds. If the adversary is static (rushing or otherwise), we do not need the two prepare rounds, and the elect round can happen in parallel to either status or propose, giving $r = 4$ rounds per iteration.

Next, we analyze the communication complexity. We will show that each of the 7 rounds consumes $O(n^2)$ communication. So the core protocol requires expected $O(n^2)$ communication (whether the adversary is adaptive or static, rushing or not). First of all, note that although a certificate consists of $f + 1$ signatures, its size can be reduced to a single signature using threshold signatures [6, 20, 27, 39]. In the status round, every party is reporting its currently accepted certificate to every other party (every party can potentially be the leader since the leader identity has not been revealed). In $\mathsf{prepare}_1$, every party sends a signed proposal, which is $O(1)$ in size, to every other party. In $\mathsf{prepare}_2$, every party sends back a doubly signed proposal, which is $O(1)$ in size, to every other party. In the propose round, every party sends a proposal, which carries a certificate, to every other party. The leader election subroutines from prior work require $O(n^2)$ communication. In the commit round, every party sends an $O(1)$-sized commit message to every other party. In the notify round, every party sends a notify message, which carries a certificate, to every other party. Lastly, before termination, every party sends $f+1$ notification headers $\langle \mathsf{notify}, v \rangle$, which can be reduced to a single threshold signature, to every other party.

5 Byzantine Broadcast and Agreement

In this section, we describe how to use the core protocol to solve synchronous authenticated Byzantine broadcast and agreement for the $f < n/2$ case. For both problems, we design a "pre-round" to let honest parties obtain initial certificates and then invoke the core protocol.

Byzantine broadcast. In Byzantine broadcast, a designated *sender* tries to broadcast a value to n parties. A solution needs to satisfy three requirements:

(termination) all honest parties eventually commit,
(agreement) all honest parties commit on the same value, and
(validity) if the sender is honest, then all honest parties commit on the value it broadcasts.

Let L_s be the designated sender. In the pre-round, L_s broadcasts a signed value $\langle v_s \rangle_{L_s}$ to every party. Such a signed value by the sender is an initial certificate certifying $(v_s, 0)$. We then invoke the core protocol. Safety and termination are satisfied due to Theorems 3 and 4. If the designated sender is honest, each honest party has a certificate for $(v_s, 0)$ and no conflicting initial certificate can exist, satisfying the condition for Theorem 5. Thus, validity is satisfied.

Byzantine agreement. In Byzantine agreement, every party holds an initial input value. A solution needs to satisfy the same termination and agreement requirements as in Byzantine broadcast. There exist a few different validity notions. We adopt a common one known as strong unanimity [12]:

(validity) if all honest parties hold the same input value v, then they all commit on v.

In the pre-round, every party i broadcasts its value $\langle v_i \rangle_i$. $f + 1$ signatures from distinct parties for the same value v form an initial certificate for $(v, 0)$. We then invoke the core protocol. Safety and termination are satisfied due to Theorems 3 and 4. If all honest parties have the same input value, then they will have an initial certificate for v and no conflicting initial certificate can exist, satisfying the condition for Theorem 5. Thus, validity is satisfied.

The efficiency of the protocols is straightforward given the analysis of the core protocol. Both protocols require one more round than the core protocol and the same $O(n^2)$ communication complexity as the core protocol.

6 Lower Bound on Expected Communication Against a Strongly Rushing Adaptive Adversary

Our proof strategy builds on and extends the classic Dolev and Reischuk [10, Theorem 2] lower bound, which shows that in every deterministic Byzantine broadcast protocol honest parties need to send at least $\Omega(f^2)$ messages. We show that in every randomized protocol solving Byzantine broadcast with ϵ error probability against a strongly rushing adaptive adversary, honest parties need to send at least $(\epsilon f/2)^2$ messages.

The Dolev-Reischuk lower bound. We first explain the Dolev-Reischuk proof at a high level. Observe that for a deterministic protocol, an execution is completely determined by the input (of the designated sender) and the adversary's strategy. Consider the following adversarial strategy. The adversary corrupts an arbitrary set B of $f/2$ parties that does not include the designated sender. Let A denote the set of remaining parties. All parties in B behave like honest parties, except that (i) they ignore the first $f/2$ messages sent to them, and (ii) they do not send messages to each other. Call the above adversarial strategy Strat. Suppose the honest designated sender sends 0 under Strat. For validity to hold, then all honest parties commit on 0.

If at most $(f/2)^2$ messages are sent to B in the above execution, then there exists a party $p \in B$ such that p receives at most $f/2$ messages. Let $S(p)$ denote the set of parties that send messages to p. Clearly, $|S(p)| \leq f/2$. We define strategy Strat' identically as Strat except that: (i) do not corrupt p, (ii) corrupt all parties in $S(p)$ (possibly including the designated sender) and prevent them from sending any messages to p (but let $S(p)$ behave honestly to other parties). Since $|S(p)| \leq f/2$, Strat' corrupts at most f parties.

Observe that honest parties in $A \backslash S(p)$ receive identical messages from all other parties under Strat and Strat', so they cannot distinguish the two executions. Thus, $A \backslash S(p)$ would still commit on 0 under Strat' However, p does not receive any message and has to commit on an arbitrary value. If this value is 1, safety is violated. If p commits on 0 when receiving no messages, we can let the sender send $v = 1$ under Strat and derive a safety violation under Strat' following a symmetric argument.

Our lower bound. We now extend the above proof to randomized protocols. In a randomized protocol, there are two sources of randomness that need to be considered carefully. On the one hand, honest parties can use randomization to their advantage. On the other hand, an adaptive adversary can also leverage randomness. Indeed our lower bound uses a randomized adversarial strategy. In addition, our lower bound crucially relies on the adversary being *strongly rushing* – the adversary can observe that a message is sent by an honest party h to any other party in a given round r, decide to adaptively corrupt h, and then remove messages sent by h in round r. We prove the following theorem:

Theorem 6 (Theorem 2, restated). *If a protocol solves Byzantine broadcast with $\frac{1}{2} + \epsilon$ probability against a strongly rushing adaptive adversary, then in expectation, honest parties collectively need to send at least $(\epsilon f/2)^2$ messages.*

Proof. We first define adversarial strategy Strat (same as Dolev-Reischuk):

1. Corrupt an arbitrary set B of $f/2$ parties. Let A denote the remaining parties.
2. Each party in B behaves like an honest party except that, it ignores the first $f/2$ messages sent to it, and it does not send messages to other parties in B.

Suppose for contradiction that a protocol solves Byzantine broadcast with less than ϵ probability of error using less than $(\epsilon f/2)^2$ expected messages. For a protocol to have an expected message complexity of $(\epsilon f/2)^2$, honest parties collectively need to send fewer than that many messages in expectation *regardless of* the adversary's strategy. Let z be a random variable denoting the number of messages sent by honest parties to B. We have $E[z] < (\epsilon f/2)^2$. Let X be the event that $z < \epsilon(f/2)^2$. By Markov's inequality, $\Pr[z \geq \frac{1}{\epsilon}E[z]] \leq \epsilon$. Thus, $Pr[z < \epsilon(f/2)^2] \geq \Pr[z < \frac{1}{\epsilon}E[z]] > 1 - \epsilon$.

Let Y be the event that among the first $\epsilon(f/2)^2$ messages, a party p picked uniformly at random by the adversary receives at most $f/2$ messages. Observe that among the first $\epsilon(f/2)^2 = \epsilon|B|(f/2)$ messages, there

exist at most $\epsilon|B|$ parties that receive more than $f/2$ of those. Since p has been picked uniformly at random from B, $\Pr[Y] \geq 1 - \epsilon$.

$$\begin{aligned}\Pr[X \cap Y] &= \Pr[X] + \Pr[Y] - \Pr[X \cup Y] \\ &\geq (1-\epsilon) + (1-\epsilon) - 1 \\ &= 1 - 2\epsilon\end{aligned}$$

Define adversarial strategy Strat' identically to Strat except:

1. The adversary picks a party $p \in B$ uniformly at random. The adversary does not corrupt p.
2. Whenever some party $s \in A$ attempts to send a message to p in a round, if the adversary has not used up its corruption quota f, it immediately corrupts s and removes the message sent by s to p in that round. Once corrupted, s does not send p any messages but otherwise behaves correctly.

Observe that $X \cap Y$ denotes the event where the total number of messages sent by honest parties to B is less than $\epsilon(f/2)^2$ and among those p has received at most $f/2$ messages. Let $S(p)$ be the set of parties that attempt to send p messages (some or all of these attempts are blocked). Thus, we have shown that, $\Pr[S(p) \leq f/2] \geq 1 - 2\epsilon$. This is the probability that the random party p picked by the adversary under Strat receives $\leq f/2$ messages, which also means p receives no message at all under Strat'. In this case, honest parties in $A \backslash S(p)$ receive identical messages under the two adversarial strategies while p needs to commit on a value without receiving any message under Strat'.

Without loss of generality, suppose that when receiving no message p commits on 1 with at least $1/2$ probability (otherwise, p commits on 0 with at least $1/2$ probability that the proof follows from a symmetric argument). Let the designated sender sends 0 under Strat. Conditioned on $X \cap Y$, validity holds under Strat if and only if safety is violated under Strat'. Thus, with at least $\frac{1}{2}(1 - 2\epsilon) = \frac{1}{2} - \epsilon$ probability, either validity is violated under Strat or safety is violated under Strat'.

7 Clock Synchronization and the Case for Synchrony

Synchrony in distributed computing has long been considered an unrealistic assumption in practice. Almost all practical consensus protocols and systems assume partial synchrony or asynchrony [7, 25, 32]. However, Bitcoin's rise to fame provides an argument that synchrony is not just convenient theoretical simplification but it can play a role in practice. In this subsection, we revisit the standard synchrony assumption and the timing assumption implicitly in Bitcoin. We explain how the two relate to and differ from each other, and re-evaluate the (im-)practicality of synchrony.

The synchrony assumption states that there is an upper bound Δ, known to all parties, on the time for messages to be delivered. [6] If such a Δ bound does not exist, the system is said to be asynchronous. It is well-known that under asynchrony, no deterministic protocol can solve consensus even against a single fail-stop fault [15]. This impossibility result motivated the introduction of partial synchrony [12]. One formulation of partial synchrony states that a Δ bound does exist but is unknown. To articulate how partial synchrony differs from synchrony, it is helpful to consider a game between *a protocol designer* and an adversarial environment [12]. Under synchrony, the environment specifies Δ first and the protocol designer needs to come up with a protocol that is correct under this particular Δ. Under partial synchrony, the protocol designer must supply a protocol first and the environment picks Δ after that; the protocol should remain correct no matter what Δ is picked.

We can think of three main concerns as to why synchrony is impractical.

1. It is easy to attack the network and violate any Δ bound, so a synchronous protocol is unsafe.

[6] In theory, the synchrony assumption has another crucial component that says there is an upper bound Φ, known to all parties, on the rate at which one party's clock runs faster than another party. But a large clock drift seems to be less of a concern in practice, so we assume a known Φ bound exists and omit discussing it here.

2. A synchronous protocol must proceed at the speed of a conservatively chosen Δ bound, so it is very slow and hence impractical.
3. A synchronous protocol requires all parties to run in synchronized rounds; it is impractical because achieving this lock-step synchrony is too hard.

Next, we study what timing assumption is implicit in Bitcoin's Nakamoto consensus protocol. First, it is not hard to see that Nakamoto consensus will fail under asynchrony. Imagine that whenever an honest node in Nakamoto consensus finds a proof-of-work solution, its messages are arbitrarily delayed due to asynchrony. Then, it is as if the adversary controls 100% of the mining power and can create arbitrarily long forks. Nakamoto consensus exhibits some unique and interesting properties under partial synchrony as analyzed by Pass, Seeman, and shelat [33][7], building on Garay, Kiayias, and Leonardos [17]. Nakamoto consensus does not have to know an exact message delay bound Δ. However, its fault tolerance is a function of Δ and a difficulty parameter p. For a given p, its fault tolerance gradually deteriorates as Δ increases, and eventually drops to 0 for very large Δ. Thus, the analysis of Pass, Seeman, and shelat [33] shows that Nakamoto consensus cannot handle partial synchrony, either.

The gradual deterioration behavior does make the timing assumption of Nakamoto consensus slightly weaker than standard synchrony. Typically, if the Δ bound is violated, a standard synchronous consensus protocol will lose safety even without the presence of any fault. Whether we can design synchronous protocols with gradual deterioration without using proof-of-work is an interesting open question.

However, it is unclear to what extent Bitcoin's success is attributed to this gradual deterioration property. In fact, Bitcoin shares the first two limitations of standard synchrony. Its designer guessed a conservative bound, 10 minutes. Progress can only be made roughly every 10 minutes; if most messages take longer than 10 minutes to deliver, Bitcoin will be plagued by frequent forks and fail to achieve consensus.[8] We think Bitcoin's almost 10-year operation provides strong evidence that a synchronous protocol will be safe if a very conservative Δ is picked, and that slow progress at a Δ speed can still provide great utility to users.

This leaves us with the third limitation — synchronous protocols will likely struggle in practice due to the difficulty of lock-step execution. Note that lock-step execution is crucial in our protocol – if party i enters a round much earlier than party j, then i may end up finishing the round too soon without waiting for j's message to arrive. This could prevent i from detecting leader equivocation and result in a safety violation. To this end, we provide a simple clock synchronization protocol to bootstrap lock-step execution from standard synchrony. We hope our clock synchronization protocol together with the above discussion can justify the use of synchronous protocols in some scenarios.

Clock Synchronization. Our clock synchronization will be executed at known time intervals. We call each interval a "day".

Round 0 (sync) When party i's clock reaches the beginning of day X, it sends a $\langle \mathsf{sync}, X \rangle_i$ message to all parties including itself.

Round 1 (new-day) The first time a party j receives $f+1$ $\langle \mathsf{sync}, X \rangle$ messages from distinct parties (either as $f+1$ separate sync messages or within a single new-day message), it

- sets its clock to the beginning of day X, and
- sends all other parties a new-day message, which is the concatenation of $f+1$ $\langle \mathsf{sync}, X \rangle$ messages from distinct parties.

The above protocol refreshes honest parties' clock difference to at most the message delay bound Δ at the beginning of each day. The first honest party to start a new day will broadcast a new-day message, which makes all other honest parties start the new day within Δ time. Obtaining a new-day message also means at

[7] The paper called their model "asynchronous" but assumed a Δ message delay bound, putting it under the partially synchronous model in conventional terminology.

[8] Note that for both Nakamoto and conventional synchronous protocols, if a few parties violate the Δ bound, we can classify them as being faulty (even if they have no ill intentions) and the protocol still functions correctly; if most parties violate the bounds, however, the protocol will fail.

least one honest party has sent a valid sync message, ensuring that roughly one day has indeed passed since the previous day. We can then set the duration of each round to $2\Delta + \phi$ where ϕ is the maximum clock drift between two honest parties in a "day".

The clock synchronization protocol bootstrap lock-step synchrony from the message delay bound Δ and a clock drift bound. Each sync message is triggered by a party's own local clock, independent of when day X would start for other parties.

Acknowledgments

We would like to acknowledge Dahlia Malkhi for many discussions in the early version of the paper and the lower bound.

References

1. Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Alexander Spiegelman. Solida: A cryptocurrency based on reconfigurable byzantine consensus. OPODIS, 2017.
2. Atul Adya, William J. Bolosky, Miguel Castro, Gerald Cermak, Ronnie Chaiken, John R. Douceur, Jon Howell, Jacob R. Lorch, Marvin Theimer, and Roger P. Wattenhofer. FARSITE: Federated, available, and reliable storage for an incompletely trusted environment. *ACM SIGOPS Operating Systems Review*, 36(SI):1–14, 2002.
3. Michael Ben-Or. Another advantage of free choice (extended abstract): Completely asynchronous agreement protocols. In *Proceedings of the second annual ACM symposium on Principles of distributed computing*, pages 27–30. ACM, 1983.
4. Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 1–10. ACM, 1988.
5. Gabriel Bracha. Asynchronous byzantine agreement protocols. *Information and Computation*, 75(2):130–143, 1987.
6. Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in Constantinople: Practical asynchronous byzantine agreement using cryptography. *Journal of Cryptology*, 18(3):219–246, 2005.
7. Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.
8. TH Chan, Rafael Pass, and Elaine Shi. Communication-efficient byzantine agreement without erasures. *arXiv preprint arXiv:1805.03391*, 2018.
9. Yevgeniy Dodis and Aleksandr Yampolskiy. A verifiable random function with short proofs and keys. In *International Workshop on Public Key Cryptography*, pages 416–431. Springer, 2005.
10. Danny Dolev and Rüdiger Reischuk. Bounds on information exchange for byzantine agreement. *Journal of the ACM (JACM)*, 32(1):191–204, 1985.
11. Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983.
12. Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323, 1988.
13. Pesech Feldman and Silvio Micali. An optimal probabilistic protocol for synchronous byzantine agreement. *SIAM Journal on Computing*, 26(4):873–933, 1997.
14. Michael J. Fischer and Nancy A. Lynch. A lower bound for the time to assure interactive consistency. *Information processing letters*, 14(4):183–186, 1982.
15. Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, 1985.
16. Matthew Franklin and Haibin Zhang. Unique ring signatures: A practical construction. In *International Conference on Financial Cryptography and Data Security*, pages 162–170. Springer, 2013.
17. Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The Bitcoin backbone protocol: Analysis and applications. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 281–310. Springer, 2015.
18. Juan A. Garay and Yoram Moses. Fully polynomial byzantine agreement for $n > 3f$ processors in $f + 1$ rounds. *SIAM Journal on Computing*, 27(1):247–290, 1998.

19. Shafi Goldwasser, Silvio Micali, and Avi Wigderson. How to play any mental game, or a completeness theorem for protocols with an honest majority. In *Proc. of the 19th Annual ACM STOC*, volume 87, pages 218–229, 1987.
20. Guy Golan Gueta, Ittai Abraham, Shelly Grossman, Dahlia Malkhi, Benny Pinkas, Michael K Reiter, Dragos-Adrian Seredinschi, Orr Tamir, and Alin Tomescu. SBFT: a scalable decentralized trust infrastructure for blockchains. *arXiv preprint 1804.01626*, 2018.
21. Jonathan Katz and Chiu-Yuen Koo. On expected constant-round protocols for Byzantine agreement. In *Annual International Cryptology Conference*, volume 4117, pages 445–462. Springer, 2006.
22. Valerie King and Jared Saia. Breaking the $O(n^2)$ bit barrier: scalable byzantine agreement with an adaptive adversary. *Journal of the ACM (JACM)*, 58(4):18, 2011.
23. Eleftherios Kokoris Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. Enhancing Bitcoin security and performance with strong consistency via collective signing. In *25th USENIX Security Symposium*, pages 279–296. USENIX Association, 2016.
24. John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geels, Ramakrishan Gummadi, Sean Rhea, Hakim Weatherspoon, Westley Weimer, and Ben Zhao. Oceanstore: An architecture for global-scale persistent storage. *ACM Sigplan Notices*, 35(11):190–201, 2000.
25. Leslie Lamport. The part-time parliament. *ACM Transactions on Computer Systems*, 16(2):133–169, 1998.
26. Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982.
27. Benoît Libert, Marc Joye, and Moti Yung. Born and raised distributively: Fully distributed non-interactive adaptively-secure threshold signatures with short shares. *Theoretical Computer Science*, 645:1–24, 2016.
28. Julian Loss and Tal Moran. Combining asynchronous and synchronous byzantine agreement: The best of both worlds. Cryptology ePrint Archive, Report 2018/235, 2018.
29. Silvio Micali. Algorand: The efficient and democratic ledger. arXiv:1607.01341, 2016.
30. Silvio Micali, Michael Rabin, and Salil Vadhan. Verifiable random functions. In *Foundations of Computer Science, 1999. 40th Annual Symposium on*, pages 120–130. IEEE, 1999.
31. Silvio Micali and Vinod Vaikuntanathan. Optimal and player-replaceable consensus with an honest majority, 2017.
32. Brian M. Oki and Barbara H. Liskov. Viewstamped replication: A new primary copy method to support highly-available distributed systems. In *Proceedings of the seventh annual ACM Symposium on Principles of distributed computing*, pages 8–17. ACM, 1988.
33. Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 643–673. Springer, 2017.
34. Rafael Pass and Elaine Shi. Feasibilities and infeasibilities for achieving responsiveness in permissionless consensus. In *International Symposium on Distributed Computing*. Springer, 2017.
35. Rafael Pass and Elaine Shi. Thunderella: Blockchains with optimistic instant confirmation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 3–33. Springer, 2018.
36. Marshall Pease, Robert Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *Journal of the ACM (JACM)*, 27(2):228–234, 1980.
37. Michael O. Rabin. Randomized byzantine generals. In *Proceedings of the 24th Annual Symposium on Foundations of Computer Science*, pages 403–409. IEEE, 1983.
38. Michael K. Reiter, Matthew K. Franklin, John B. Lacy, and Rebecca N. Wright. The Ω key management service. In *Proceedings of the 3rd ACM conference on Computer and communications security*, pages 38–47. ACM, 1996.
39. Victor Shoup. Practical threshold signatures. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 207–220. Springer, 2000.
40. Lidong Zhou, Fred Schneider, and Robbert van Renesse. COCA: A secure distributed online certification authority. *ACM Transactions on Computer Systems*, 20(4):329–368, 2002.