Simplex Consensus: A Simple and Fast Consensus Protocol

Benjamin Y Chan*
Cornell University
byc@cs.cornell.edu

Rafael Pass†
Tel-Aviv University and Cornell Tech
rafaelp@tau.ac.il

June 1, 2023

Abstract

We present a theoretical framework for analyzing the efficiency of consensus protocols, and apply it to analyze the optimistic and pessimistic confirmation times of state-of-the-art partially-synchronous protocols in the so-called "rotating leader/random leader" model of consensus (recently popularized in the blockchain setting).

We next present a new and simple consensus protocol in the partially synchronous setting, tolerating $f < n/3$ byzantine faults; in our eyes, this protocol is essentially as simple to describe as the simplest known protocols, but it also enjoys an even simpler security proof, while matching and, even improving, the efficiency of the state-of-the-art (according to our theoretical framework).

As with the state-of-the-art protocols, our protocol assumes a (bare) PKI, a digital signature scheme, collision-resistant hash functions, and a random leader election oracle, which may be instantiated with a random oracle (or a CRS).

*Supported in part by NSF CNS-2128519, NSF Award RI-1703846 and AFOSR Award FA9550-18-1-0267
†Supported in part by NSF Award RI-1703846, AFOSR Award FA9550-18-1-0267, a JP Morgan Faculty Award and an award from the Algorand Foundation

1 Introduction

Distributed consensus algorithms [LSP82] allow large numbers of machines to agree on a single ground truth, even when some machines malfunction. Born out of research towards fault-tolerant aircraft control [WLG$^+$78] in the 1970s, consensus algorithms have since then touched every corner of the Internet, and are used by the Internet's most important services to replicate data at scale (e.g. Google's Chubby lock service [Bur06], Apache Zookeeper [Apa], and many more). Today, new varieties of consensus algorithms power blockchains such as Bitcoin [Nak08] and Ethereum [B$^+$14], where users propose transactions which are then batched into agreed-upon blocks, and where—unlike in classical consensus algorithms—the set of servers is not known ahead of time, and instead miners can join the system at any time (in other words, the system is "permissionless"). Subsequently, blockchains have found new applications in cryptocurrency and also in bringing liquidity to otherwise illiquid markets (such as the market for art [AJK$^+$22]).

At their core, consensus algorithms allow people to work together and collaborate without needing to trust each other. In some sense, they (and their evolution as multi-party computation algorithms [Yao82]) are the epitome of enabling collaboration without needing trust. In a society that heavily depends on trust-mitigation protocols (such as markets, democracy, a justice system, etc) to keep people safe, boost productivity, and to provide a high standard of living, continued innovation on trustless algorithms is of fundamental importance.

Relevance of the permissioned setting. In this paper, we return our focus to *classical permissioned consensus protocols*, where the set of participants is known ahead of time. This setting has by now been studied for four decades [LSP82], but importantly, many modern techniques for realizing scalable, energy efficient *permissionless* blockchains (see e.g. Ethereum [BG17], Algorand [CGMV18]) rely on classical permissioned consensus as a building block. In particular, blockchain applications bring forth new desiderata for such consensus protocols, and require them to be faster and more robust than ever before, which we will soon make more precise.

Our focus here is on the *partially-synchronous* model of computation [DLS88], which is specified by a worst-case bound on message delivery Δ, but where the actual message delivery time may be much smaller $\delta < \Delta$. The protocol is aware of Δ, but is not aware of δ; additionally, even the worst-case bound may not always hold, but is only required to hold after GST (the global stabilization time); liveness is only required to hold after GST, and consistency holds always[1]. As we are in the partially-synchronous model, we assume the byzantine attacker can control at most 1/3 of the players (which is optimal [DLS88]). The partial-synchronous model is well-suited for settings that require security even if the network is partitioned (e.g. due to a Denial-of-Service attack), or if message delivery is unreliable (e.g. on today's Internet). Note that the partially-synchronous approach underlies many of the most successful consensus protocols deployed today, including Paxos [LAM98] and PBFT [CL$^+$99].

Additionally, we adopt the *multicast model* of communication. In the multicast model, players communicate by multicasting a message to the entire network, as opposed in the point-to-point model, where players send each message to a single recipient (whose identity must usually be known). The multicast model is the model of choice for protocols built for the large-scale peer-to-peer setting (see Bitcoin, Ethereum, and Algorand as examples).

[1] As usual, the model can also be extended a more general model where liveness hold during "periods of synchrony"; for simplicity, we ignore this distinction.

Consensus with rotating/random leaders. In a blockchain protocol or a state-machine replication system, clients send the system a series of transactions (txs) over time and the protocol participants must collectively decide the order in which the transactions are executed, outputting a LOG of transactions. Suppose that there are two conflicting pending transactions txs and txs′ (for instance, comprising a double spend), and that the protocol must decide which of them to include in the final log of transactions. Typically, protocols (in the partially synchronous setting) decide which of the transactions to include by either flipping an imperfect global coin [CR93, CKS00] to directly choose one of the transactions, or by *electing a leader process* to decide on behalf of all of the participants. Nearly all modern protocols deployed in practice fall into the second category, including blockchains, where the leader is equivalent to the block proposer (see Paxos [Lam01], PBFT [CL$^+$99], Bitcoin [Nak08], Ethereum [B$^+$14], Algorand [GHM$^+$17]). The reason is one of *scalability*: a block proposer can sequence many transactions in a row (as in PBFT), and more importantly also aggregate transactions together into blocks to improve throughput (as in blockchains). In contrast, flipping a global coin for each pair of transactions is expensive and is largely restricted to solving binary consensus, unless the coin is itself used to elect a leader.

Consequently, block proposers (or leaders) have disproportionate power when deciding the order of transactions within blocks, or across blocks if it is proposing many blocks in a sequence. In a system such as Ethereum, where block proposers can profit from ordering transactions at will, such a disproportionate balance of power affects the security and fairness of the underlying protocol (e.g. via "miner extractable value" [DGK$^+$20]). Mitigating the power of block proposers to reorder transactions within a block is an active area of research (e.g. see [HCPS19, KZGJ20]). Over multiple blocks, it is prudent to ensure that each new block is proposed by a different, "fresh" block proposer[2], if only to ensure that no process is 'in power' for too long.

A slow leader can additionally cause blocks to be confirmed slowly and drastically reduce throughput. In a 'stable leader' protocol such as PBFT, where the leader is never changed unless it detectably misbehaves, this can be problematic. Even if there is only a *single* slow process, if that process is the leader, then every block proposal may take much longer than the true network speed δ. Thus one more reason to rotate leaders frequently is to ensure that a single slow leader cannot slow down the entire operation for too long (e.g. see [AAC$^+$05]).

In this paper, we focus on leader-based consensus protocols in the permissioned setting, specifically those protocols that rotate its leaders, or randomly choose their leaders for each block. This seems essential for both fairness and performance in the common case where proposers can be corrupt.

Efficiency measures for consensus with rotating leaders. There has been a recent explosion in new leader-based consensus protocols. To compare these various approaches, and to understand whether they are "optimal" in some well-defined way, we need to specify some *efficiency measures*. In this paper, we consider the following notions of efficiency:

- *Optimistic confirmation time:* Suppose that all block proposers are honest. When a transaction is provided to the protocol (after GST), how long does it take for the transaction to be finalized? The optimistic confirmation time is (informally) the sum of the "proposal confirmation time" and the "optimistic block time" (both rather imprecise, but popular, notions):

[2] See the related notion of chain quality [GKL15], which (informally) requires that any sufficiently long chain contain a large fraction of blocks that are mined by honest parties.

  - *Proposal confirmation time:* Suppose an honest proposer is elected and proposes a block; how long does it take for that block to be finalized?
  - *Optimistic block time:* Suppose all block proposers are honest. How long does a pending transaction need to wait to be included in the next honest block proposal?

  If the optimistic confirmation time depends only on the true message delivery time δ and not on the known upper bound Δ, we say that the protocol is *optimistically responsive* [PS18]. In practice, Δ is usually set conservatively s.t. δ is much smaller than Δ, in which case a protocol that runs at the speed of δ (instead of the parameter Δ) is much preferable.

- *Pessimistic confirmation time:* Suppose that some number f of eligible proposers are corrupt. When a transaction is provided to the protocol (after GST), how long does it take for the transaction to be finalized? This is an important metric in practice, where there are almost certainly leaders who are offline, or even downright malicious. Uncooperative leaders may choose to exclude a transaction from its block proposals, whereas slow leaders may not propose it at all. There are two notions here:

  - *Worst-case confirmation time.* In the very worst-case, how long must we wait for a transaction to be finalized? The bound must hold for any transaction arrival time t.
  - *(Expected) pessimistic confirmation time.* In expectation, how long does it take for a transaction to be finalized?

    Essentially all protocols in the literature consider a "view-based liveness" notion that is tailored for rotating leader based protocols.[3] Following the literature, we will also consider this notion of expected "view-based" liveness: Suppose that the protocol proceeds in incrementing "views" or "iterations" $v := 1, 2, \ldots$ where each view v is assigned a single leader L_v. Fix some view number v ahead of time, and suppose that a transaction is provided to every honest player at the time they enter view v. How long (in expectation) does it take for the transaction to be finalized, once every honest process has entered view v?

  We focus mainly on the expected pessimistic confirmation time. For our protocol, as well as many other protocols in the literature, we can turn an expected liveness bound into a worst-case liveness bound by losing an additive term of $\omega(\log \lambda) \cdot \Theta(\Delta)$. Note that, if a protocol requires multiple honest leaders in a row to confirm transactions, the worst-case bound will be worse.

The State of the Art. Before describing our contributions, we summarize the current start of the art; we focus only on leader-based protocols that rotate their leaders or elect random leaders. (The interested reader will find further comparisons of related works in Section 4, building on the survey below.) Throughout, denote n the number of participants, and f the number of faults; we consider static corruptions only.

[3] We may also consider an *expected worst-case confirmation time*, where, for any transaction arrival time t after GST, both fixed ahead of time, a transaction must be finalized soon after t in expectation over the coins of the execution. Such a bound may be useful to capture real-world liveness, but is typically difficult to analyze in a setting where the adversary can control the scheduling of messages.

Roughly speaking, the literature has focused on optimizing the *optimistic confirmation times*, but this has come at the expense of sacrificing the *pessimistic confirmation time*. In more details, let's first examine optimistic confirmation time:

- *Proposal confirmation-time favoring.* The seminal paper of PBFT [CL+99] achieves a proposal confirmation time of 3δ for honest block proposals, but has stable leaders. Instead, consider an optimistically responsive "base version" of the Algorand agreement protocol [CGMV18], denote Algorand*[4], which is similar to PBFT but allows a different leader for each block. Algorand* achieves a proposal confirmation time of 3δ, and an optimistic block time of 3δ. [ANRX21] later showed that a 3δ proposal confirmation time is optimal when $f \geq (n+2)/5$.[5]

- *Block-time favoring.* Following Hotstuff [YMR+19], a new class of pipe-lined protocols were subsequently designed to improve the optimistic block time of rotating-leader protocols from 3δ to 2δ. These protocols pipe-line proposals, similar to Nakamoto style consensus [Nak08], and achieve an optimistic block time of 2δ. However, these protocols require a worse proposal confirmation time: proposals take 4δ time (Pala [CPS18]), or 5δ time (Jolteon [GKKS+22]/Pipelined Fast-Hotstuff [JNFG20]) or 7δ time (Hotstuff [YMR+19]) to be confirmed, and additionally require 2, 3, and 4 honest leaders in a row respectively to confirm each block. In essence, these protocols improve the blocktime to just 2δ, but they sacrifice the proposal confirmation time. Moreover, the pessimistic confirmation time blows up significantly. Note that Hotstuff and its variants are generally designed for the point-to-point messaging model, where, to reduce communication, players often send their votes to a single recipient who aggregates them; if players instead multicast their votes, the confirmation time improves slightly (but only shaves off δ, or one less honest leader in a row).

- *Simultaneously proposal confirmation-time and block-time friendly:* The recent ICC protocol [CDH+22] simultaneously achieves the state-of-the-art 3δ proposal confirmation time and 2δ optimistic block time. However, this protocol again requires (essentially) 2 honest leaders in a row to confirm a block if a faulty leader was previously elected. Requiring multiple honest leaders in a row to finalize a block severely impacts the pessimistic confirmation time, which we will explore next.

Pessimistic liveness. Few protocols explicitly analyze their expected or worst-case confirmation time under pessimistic conditions, despite it being an important performance desiderata in practice—after all, it is natural to assume that a fraction of the participants will be offline if not outright malicious. To explicitly compare the different approaches, we focus on the setting where each "iteration" of the protocol is associated with a randomly selected leader (essentially all the protocols in the literature, for this setting, assume or instantiate such a sequence of leaders). Throughout, assume that $f = \lfloor n/3 \rfloor$.

The current state-of-the-art is achieved by Algorand* (as defined above), which achieves $4\delta + 2\Delta$ expected view-based liveness. Protocols that require multiple honest leaders in a row to confirm

[4]While the version of Algorand agreement in [CGMV18] is not optimistically responsive, it can be easily made so if every period has a unique leader known ahead of time (provided by a leader election oracle in lieu of using a VRF). Then, players can simply 'soft-vote' immediately on seeing a proposal from the leader, and 'cert-vote' immediately on seeing $2n/3$ soft-votes, much like in PBFT.

[5]We do not consider optimizations of the variety made in Parametrized FaB Paxos [MA06] and SBFT [GAG+19], where if $f < (n+2)/5$, a proposal confirmation time of 2δ is possible without affecting worst-case fault tolerance. We are particularly interested in the case when leaders are honest, but 1/3 of voters are not.

	Proposal Conf. Time	Optimistic Block Time	Pessimistic Liveness ($f = \lceil n/3 \rceil - 1$)
Simplex	$\boldsymbol{3\delta}$	$\boldsymbol{2\delta}$	$\boldsymbol{3.5\delta + 1.5\Delta}$
Algorand* [CGMV18]	$\boldsymbol{3\delta}$	3δ	$4\delta + 2\Delta$
ICC [CDH+22]	$\boldsymbol{3\delta}$	$\boldsymbol{2\delta}$	$5.5\delta + 1.5\Delta$
PaLa [CPS18]	4δ	$\boldsymbol{2\delta}$	$6.25\delta + 9.25\Delta$
Pipeline Fast-Hotstuff [JNFG20] Jolteon [GKKS+22]	5δ	$\boldsymbol{2\delta}$	$10.87\delta + 9.5\Delta$
Chained Hotstuff (v6) [YMR+19]	7δ	$\boldsymbol{2\delta}$	$19.31\delta + 12.18\Delta$
P-Sync Bullshark [SGSKK22b]	6δ	3δ	$6\delta + 9\Delta$
Streamlet [CS20a]	6Δ	2Δ	35.56Δ

*Base protocol without sortition.

Table 1: Latency of Popular Consensus Protocols (Random Leaders)

transactions generally achieve degraded expected liveness: ICC [CDH+22], PaLa [CPS18], Pipelined Fast-Hotstuff/Jolteon [JNFG20, GKKS+22], and Chained Hotstuff (v6) [YMR+19] have expected liveness of $5.5\delta + 1.5\Delta$, $6.25\delta + 9.25\Delta$, $10.87\delta + 9.5\Delta$, and $19.31\delta + 12.18\Delta$ respectively.[6] Generally speaking, the more honest leaders required in a row to finalize a block, the worse the expected pessimistic liveness.

These protocols also get an even worse *worst-case confirmation time*: a protocol that requires k honest leaders in a row would need a longer sequence of random leaders to guarantee that k honest leaders in a row are elected, compared to if it only required 1 honest leader for confirmation.[7] Protocols with random leaders generally require just a sequence of at least $\omega(\log \lambda)$ leaders to guarantee one honest leader.

So, summarizing the state of the art, there is a trade-off between achieving good optimistic liveness bounds, and and achieving pessimistic liveness bounds.

[6] Here, Chained Hotstuff is analyzed using a timeout-based pacemaker from LibraBFT [BCC+19] and a timeout of 3Δ, since they don't instantiate their own pacemaker. PaLa is analyzed with a less conservative timeout of 1min=5Δ, and 1sec=2Δ than the ones presented in their paper.

[7] An alternative is to have a single leader be in power for k iterations in a row, before switching to the next leader; this sacrifices fairness, since we would like to rotate leaders more frequently, and the latency is still not ideal, since a faulty leader can delay the execution for k iterations.

Our contributions. In this paper, we present a simple consensus protocol, named Simplex, that matches the best optimistic liveness bounds, while at the same time matching and even improving the best known pessimistic liveness bounds (both in expectation and in the worst-case). Most importantly, our protocol is simple to describe and has an, in our eyes, a very simple proof of correctness—including an easy liveness proof. In contrast, most protocols in the literature have somewhat informal or difficult liveness arguments or do not analyze their own theoretical latency at all. (In terms of simplicity, while the protocol itself is arguably more complicated to describe than Streamlet [CS20a], the proof of correctness is significantly easier, which may make it more understandable.)

In more details, assuming a "bare" PKI, collision-resistant hash functions, and a random leader election oracle (which can be instantiated using a Random Oracle), Simplex implements partially-synchronous consensus (tolerating the standard $f < n/3$ static byzantine faults), while achieving

- an optimal proposal confirmation time of 3δ,
- an optimistic block time of 2δ,
- an expected pessimistic confirmation time of $3.5\delta + 1.5\Delta$,
- a worst-case pessimistic confirmation time of $4\delta + \omega(\log \lambda) \cdot (3\Delta + \delta)$,
- and using $O(n)$ multicast complexity.

As observed by [PS17], the Random Oracle can be replaced with a PRF (which follows from collision-resistant hash functions [HILL99]) and a CRS chosen after the public keys.

We also note that using now standard techniques, we can bring down the communication complexity to $\mathsf{polylog}(\lambda)$ (in the multi-cast model) and linear (in the point-to-point model) through subsampling the committee of voters [CM19, CPS18]. This gives a scalable protocol that may be suitable for deployment in practice on large-scale peer-to-peer networks. We do not innovate on this subsampling and therefore just focus on the "base protocol" (which determines the concrete efficiency also of the protocol with subsampling).

Comparison of techniques. Our contributions build on the techniques of many prior works. In contrast with the pipelined protocols of [CS20a, YMR$^+$19, CPS18, JNFG20, GKKS$^+$22], which generally fall into a streamlined "propose/vote" paradigm where processes only send one vote type, our protocol follows [CL$^+$99] and requires processes to multicast a second "finalize" message to finalize blocks (in parallel with the next block proposal). We posit that the resulting protocol is actually simpler despite the second voting step.[8] Moreover, having a second voting step makes for a faster protocol in the optimistic case, matching the proposal confirmation time of PBFT. Another technique that we use is that of a "dummy block": if, during some iteration, a process detects no progress (i.e. due to a faulty leader or network), it will timeout and vote for the dummy block. On seeing $2n/3$ votes for the dummy block, a player may move to the next iteration and try again with a new leader. It is similar to the notion of a timeout certificate in [CPS18, GKKS$^+$22, YMR$^+$19]; the difference is that, in our protocol, a process must either vote for the dummy block of height h, or vote "finalize" for a block of height h (whereas no such stipulation was made in prior work). Then, if we see $2n/3$ finalize messages, we can be sure that no process saw $2n/3$ votes for the dummy block at that height, and it is safe to finalize the block. The result is a simpler consistency and liveness proof; moreover, a block proposer never needs to wait extra time before proposing (unlike in [CPS18, BKM18]) even after seeing a timeout block.

[8] To recover a streamlined protocol, it is possible to "piggyback" the finalize message onto the first vote message of the next block; this would only make liveness a little slower, and consistency would still hold.

Comparison with asynchronous protocols. Consensus protocols have also been designed for the *asynchronous setting* [Rab83, CR93, CKS00, MXC$^+$16, AMS19, etc], where protocols should make progress even before GST, as long as messages are eventually delivered. Asynchronous protocols generally have worse latency than their partial-synchronous counterparts after GST. Here, we only mention the elegant DAGRider protocol [KKKNS21], which requires 4 rounds of reliable broadcast ($\approx 12\delta$ time) in the optimistic case, and Bullshark [SGSKK22a], which adds a synchronous fast-path but still requires 2 rounds of reliable broadcast (or 6δ time) optimistically, as opposed to the optimal 3δ. Moreover, essentially every asynchronous protocol requires a common coin, which is most practically implemented using threshold signatures, following [CKS00]. Threshold signatures require a private setup or trusted dealer, which aside from being hard to implement, makes it difficult to subsample the protocol for scalability.

2 The Protocol

We describe the Simplex consensus protocol in the framework of blockchains, which provides an elegant framework for reasoning about consensus on a sequence of values.

To start, the protocol assumes a (bare) public-key infrastructure (PKI), and a digital signature scheme. Additionally, let $H : \{0,1\}^* \to \{0,1\}^*$ be a publicly known collision-resistant hash function.

- **Bare PKI setup.** Before the protocol execution, a trusted party generates a $(\mathsf{pk}_i, \mathsf{sk}_i)$ keypair for each process $i \in [n]$ using the key generation algorithm for the digital signature scheme, and for each $i \in [n]$ sends $(\mathsf{pk}_i, \mathsf{sk}_i)$ to process i. Each process replies with a pk'_i, where honest processes reply with the same $\mathsf{pk}'_i = \mathsf{pk}_i$. The trusted party then sends $\{\mathsf{pk}'_i\}_{i\in[n]}$ to all parties.[9]

- **Notation for digital signatures.** For any message $m \in \{0,1\}^*$ and a process $p \in [n]$, we denote by $\langle m \rangle_p$ a tuple of the form (m, σ), where σ is a valid signature for m under process p's public key.

The protocol uses the following data structures.

- **Blocks, block heights, and the genesis block.** A *block* b is a tuple $(h, \mathsf{parent}, \mathsf{txs})$, where $h \in \mathbb{N}$ is referred to as the *height* of the block, parent is a string that (typically) is meant to be the hash of a "parent" blockchain, and txs is an arbitrary sequence of strings, corresponding to transactions contained in the block.

  Define the *genesis block* to be the special tuple $b_0 := (0, \emptyset, \emptyset)$.

- **Dummy blocks.** The special *dummy block* of height h is the tuple $\bot_h := (h, \bot, \bot)$. This is an empty block that will be inserted into the blockchain at heights where no agreement is reached. A dummy block does not point to a specific parent.

- **Blockchains.** A *blockchain* of *height* h is a sequence of blocks $(b_0, b_1, \ldots, b_h)$ such that b_0 is the genesis block, and for each $i \in [h]$, either $b_i = \bot_i$ or $b_i = (i, H(b_0, \ldots, b_{i-1}), \mathsf{txs})$ for some txs.

[9] Note that this is referred to as a Bare PKI [BCNP04] since malicious parties may pick their own, potentially malformed, public keys.

- **Notarized blocks and blockchains.** A *notarization* for a block b —which may be the dummy block— is a set of signed messages of the form $\langle \mathsf{vote}, h, b \rangle_p$ from $\geq 2n/3$ unique processes $p \in [n]$, where h is the height of the block b. A *notarized block* is a block augmented with a notarization for that block.

  A *notarized blockchain* is a tuple $(b_0, \ldots, b_h, S)$, where $b_0, \ldots, b_h$ is a blockchain, and S is a set of notarizations, one for each block $b_1, \ldots, b_h$. We stress that a notarized blockchain may contain notarized dummy blocks.

- **Finalized blocks and blockchains.** A *finalization* for a height h is a set of signed messages of the form $\langle \mathsf{finalize}, h \rangle_p$ from $\geq 2n/3$ unique processes $p \in [n]$. We say that a block of height h is *finalized* if it is notarized and accompanied by a finalization for h.

  A *finalized blockchain* is either just the genesis block b_0, or a notarized blockchain accompanied by a finalization for the last block.

- **Linearizing a blockchain.** Given a blockchain $b_0, b_1, \ldots, b_h$, denote $\mathsf{linearize}(b_0, b_1, \ldots, b_h)$ to be the (most natural) operation that takes the sequences of transactions from each individual block, in order, and outputs the concatenation, to form a total ordering of all transactions in the blockchain.

2.1 The Protocol Description and Main Theorem

We are now ready to describe the protocol. The protocol runs in sequential iterations $h = 1, 2, 3, \ldots$ where each process starts in iteration $h = 1$. Note that each process may advance through iterations at a different speed, and at any given time, two processes may be in two different iterations, due to network delay (since we are in the partially synchronous setting). As local state, each process $p \in [n]$ keeps track of which iteration h it is currently in, and also stores all of the notarized blocks and messages that it has seen thus far.

Additionally, we assume that each iteration h has a pre-determined block proposer or leader $\mathsf{L}_h \in [n]$ that is randomly chosen ahead of time; this is referred to as a *random leader election oracle* and can be implemented using a random oracle: namely, $\mathsf{L}_h := H^*(h) \mod n$, where $H^*(\cdot)$ is some public hash function modeled as a random oracle.

Player p on entering iteration h does the following:

1. **Leader proposal:** If $p = \mathsf{L}_h$, p multicasts a single proposal of the form

$$\langle \mathsf{propose}, h, b_0, \ldots, b_{h-1}, b_h, S \rangle_p.$$

   Here, $b_0, \ldots, b_h$ is p's choice of a blockchain of height h, where $(b_0, \ldots, b_{h-1}, S)$ is a notarized parent blockchain, and $b_h \neq \bot_h$. The new block b_h should contain every pending transaction that p has seen that is not already in the parent chain.

2. **Dummy blocks:** Each process p starts a new timer T_h, set to fire locally after 3Δ time.[10] If T_h fires, vote for the dummy block by multicasting $\langle \mathsf{vote}, h, \bot_h \rangle_p$.

[10] In addition, we could optimistically fire the timer when the leader "equivocates"; we omit the rule for brevity.

3. **Notarizing block proposals:** On seeing the *first* proposal of the form

$$\langle \mathsf{propose}, h, b_0, \ldots, b_h, S\rangle_{\mathsf{L}_h}$$

check that $b_h \neq \bot_h$, that $b_0, \ldots, b_h$ is a valid blockchain, and that $(b_0, \ldots, b_{h-1}, S)$ is a notarized blockchain. If all checks pass, multicast $\langle \mathsf{vote}, h, b_h\rangle_p$.

4. **Next iteration and finalize votes:** On seeing a notarized blockchain of height h, enter iteration $h+1$. At the same time, p multicasts its view of the notarized blockchain to everyone else.

   At this point in time, if the timer T_h did not fire yet: cancel T_h (so it never fires) and multicast $\langle \mathsf{finalize}, h\rangle_p$.

5. **Finalized Outputs:** Whenever p sees a finalized blockchain $b_0, \ldots, b_{h'}$, output the contents $\mathsf{LOG} \leftarrow \mathsf{linearize}(b_0, \ldots, b_{h'})$.

Let us informally describe the intuition behind the protocol. In each iteration, the processes collectively try to get the leader's block (proposal) notarized; if this fails due to a faulty leader or a faulty network, then the timer will fire. The timer thus upper bounds the amount of time each leader has to get its block notarized; when it fires, processes now have the option of voting for the dummy block and (on seeing a notarization for this dummy block) moving to next height to try again with the next leader. Eventually, we will hit a good leader and have good network conditions and an iteration h will complete without the timer T_h firing for any honest process. Consistency, on the other hand, will follow from a straight-forward use of the standard "quorum intersection lemma".

Summarizing, we get the following theorem.

Theorem 2.1 (Partially-synchronous Consensus)**.** *Assuming collision-resistant hash functions, digital signatures, a PRF, a bare PKI, and a CRS, there is a partially-synchronous blockchain protocol for $f < n/3$ static corruptions that has $O(n)$ multicast complexity, optimistic confirmation time of 5δ, worst-case confirmation time of $4\delta + \omega(\log\lambda)\cdot(3\Delta+\delta)$, and expected view-based liveness of $3.5\delta + 1.5\Delta$.*

Note that the existence of digital signatures and PRFs follows from the existence of collison-resistant hash functions (see [Rom90] and [HILL99] respectively), but we include it here to emphasize what cryptographic building blocks we rely on. Also, as noted in [PS17], we can replace the random oracle with a common reference string (CRS) that is chosen after the adversary has registered its public keys, and instead use $\mathsf{L}_h = \mathsf{PRF}_{\mathsf{crs}}(h) \mod n$ (and note that existence of PRFs follow from the existence of collision-resistant hash functions [HILL99]).

2.2 Proof outline

Let us provide a proof outline which conveys the essence of the whole proof:

Consistency. The consistency proof is straightforward. Let Alice and Bob be two honest players. We want to show that if Alice finalizes a chain $b_0, \ldots, b_h$, and Bob finalizes a longer chain $b_0, \ldots, b'_h, \ldots, b'_{h'}$, then Alice's chain is a prefix of Bob's, namely that $b_0 \ldots, b_h = b_0, \ldots, b'_h$. It suffices to show that Bob's block at height h is identical to Alice's, namely that $b'_h = b_h$; by the collision resistant hash function, the parent chains must be the same.

First, since Alice saw a finalization for height h, then the dummy block $\perp_h$ cannot be notarized in Bob's view, by the standard "quorum intersection" lemma. This is because an honest player only votes for either $\langle \mathsf{finalize}, h\rangle$, or $\langle \mathsf{vote}, h, \perp_h\rangle$, and never both. Likewise, since Alice saw that b_h is notarized, then a competing non-dummy block $b'_h \neq b_h$ also cannot be notarized in Bob's view, again by the standard "quorum intersection" lemma. This follows because an honest player only votes for a single (non-dummy) block proposal per iteration. Immediately it must be that $b'_h = b_h$, as required.

Liveness. Perhaps more interestingly, the liveness proof is also simple—this is in contrast to all previous protocols in the partially-synchronous setting that we are aware of, each of which require a subtle/complex analysis. We first claim that any honest leader can drive the protocol forward after GST. To see this, let us first observe that honest players need to be "synchronized" after GST:

> *"Synchronization after GST" (Sync): If an honest player enters an iteration h by time $t > \mathsf{GST}$, then every honest player enters iteration h by time $t + \delta$.*

This follows because when an honest player enters iteration h, it forwards a notarized blockchain of height $h - 1$ to everyone else (and thus they will enter iteration h once they receive it, unless they had already entered iteration h before).

Now, assume the honest leader for height h proposes a block at time $t > \mathsf{GST}$. We shall argue that the block it proposes will be finalized by time $t + 3\delta$. This follows from the following observations.

1. *Every honest player must enter iteration $h + 1$ by time $t + 2\delta$.* This is because either (a) *every* honest player votes for the leader's block by time $t + \delta$ (thus they all see a notarized blockchain for h by time $t + 2\delta$), or (b) *some* honest player i did *not* vote for the leader's block. In the latter case, player i must have already been in iteration $h + 1$ (or higher) when it saw the block proposal by time $t + \delta$ (since otherwise it would have voted as by Sync, it must be in iteration at least h by time $t + \delta$) and thus would have forwarded a notarized blockchain of height h by $t + \delta$. So again, every honest player will enter iteration $h + 1$ by time $t + 2\delta$.

2. *No timer of an honest player can fire until after time $t + 2\delta$.* This is because no honest player could have entered iteration h before time $t - \delta$ by Sync. Consequently, the earliest any (honest) timer can fire is after time $t - \delta + 3\Delta > t + 2\delta$.

Combining Observation 1 and 2, we have that every honest player must vote $\langle \mathsf{finalize}, h\rangle$ when it enters iteration $h + 1$ (because none of their timers have fired by the time they enter iteration $h + 1$). Consequently, by time $t + 3\delta$, every honest player sees that h is finalized. Since no honest player voted for the dummy block (since by Observation 2, their timers did not fire), it must be the leader's block that is finalized, which concludes the proof for the case that the leader is honest.

Let us next consider the case that the leader for iteration h is malicious. We shall argue that in this case, the iteration will be "skipped" (and all honest processes move on to iteration $h + 1$) after at most $3\Delta + \delta$ time: Suppose that every honest player is in iteration h by time $t > \mathsf{GST}$. Either every honest player fires its timer (by time $t + 3\Delta$), or some honest player does not fire its timer. In the first case, they all vote for the dummy block and see a notarized dummy block by time $t + 3\Delta + \delta$, thus entering iteration $h + 1$. In the second case, that player must have entered iteration $h+1$ before its timer could fire, i.e. before $t+3\Delta$; but as before, then every honest player will follow it into iteration $h + 1$ by time $t + 3\Delta + \delta$.

3 Formal Analysis

3.1 Preliminaries

We analyze the Simplex protocol in the framework of blockchains. Throughout, when we say that a message is "in honest view", we mean that it is in the view of some honest process (but perhaps not all honest processes).

The Permissioned Setting. We consider static byzantine corruptions. Denote n the number of players, $f < n/3$ of which are set to be "corrupted". The corrupted players are chosen ahead of time, before the setup phase. The remaining players are "honest".

Protocol Execution. In our setting the adversary has power over the network and can choose when to deliver messages sent by honest players. However, in good network conditions, message delivery should occur "quickly". This requires that we formalize a notion of time, in addition to specifying the execution of a distributed protocol.[11] In the spirit of the UC framework [Can01], consider n player machines, an adversary machine $\mathcal{A}$, and an environment $\mathcal{Z}$, where each is modeled as an instance of a non-uniform probabilistic polynomial time (nuPPT) interactive Turing Machine (ITM) that takes as input the security parameter 1^λ. The adversary $\mathcal{A}$ controls the f (statically corrupted) faulty players. Each player additionally has access to an authenticated multicast channel $\mathcal{F}_{\mathsf{mult}}$ over which $\mathcal{A}$ has scheduling power. We use the notation $\mathsf{EXEC}_\Pi(1^\lambda, \mathcal{Z}, \mathcal{A})$ to denote a random execution of the protocol Π with $\mathcal{A}$ and $\mathcal{Z}$, over the coins of the setup, the players, $\mathcal{A}$, and $\mathcal{Z}$.

To model time, we consider environments $\mathcal{Z}$ that send special "tick" messages to the n player machines. Fix any execution of a protocol with $\mathcal{Z}$ and $\mathcal{A}$ on some 1^λ. Recall that an execution is a sequence of activations of instances of machines. Given an execution, we say that it proceeds in timesteps $t = 0, 1, 2, \ldots$ where timestep t starts at the first point (in the execution) where the environment $\mathcal{Z}$ has sent any player machine exactly t "tick" messages. An event happens before/at/after time t if it occurs before/during/after timestep t in the execution. The execution ends when $\mathcal{Z}$ halts.

Network Model and Clocks. We consider adversaries that are partially synchronous [DLS88]. An environment/attacker tuple $(\mathcal{Z}, \mathcal{A})$ is "δ-bounded partially synchronous" if, for every protocol Π, for every security parameter λ, there exists a time $\mathsf{GST} \in \mathbb{N}$ s.t. in every execution of Π with $\mathcal{Z}$ and $\mathcal{A}$ on 1^λ:

- *Synchronized clocks.* Denote t^* the time at which $\mathcal{Z}$ halts. It must be that $\mathcal{Z}$ sends every honest player exactly one "tick" message at each time $t \in [t^*]$.

[11] Doing this composably has been visited in-depth in works such as [KLP05, KMTZ13, BMM14, KZZ16, CHMV17, BDD+21]. However, here we ignore the question of composability and use an simple stand-alone model for convenience.

- *Message delivery guarantee after GST.* For every time $t \in \mathbb{N}$, if some honest player sends a message by time t, the message is delivered by time $\max(\mathsf{GST}, t + \delta)$. Note that GST is not publicly known.

Definition of a blockchain protocol. Let $T : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, and $t \in \mathbb{N}$. A protocol Π, parametrized by Δ, is said to compute a blockchain with (expected) $T(\delta, \Delta)$-*liveness* if for every environment $\mathcal{Z}$ and attacker $\mathcal{A}$, in executions of Π in $\mathcal{Z}$ with $\mathcal{A}$ on 1^λ, the following properties hold with all but negligible probability in λ:

- *Consistency.* If two honest players ever output sequences of transactions LOG and LOG' respectively, either $\mathsf{LOG} \preceq \mathsf{LOG}'$ or $\mathsf{LOG}' \preceq \mathsf{LOG}$, where "$\preceq$" means "is a prefix of or is equal to".

- $T(\delta, \Delta)$*-Liveness.* Fix any time $t \in \mathbb{N}$. Suppose that $(\mathcal{A}, \mathcal{Z})$ is additionally δ-bounded partially synchronous for some $\delta < \Delta$ and $\mathsf{GST} < t$. Suppose that $\mathcal{Z}$ never halts early and that it always delivers some input txs to every honest player by time t. Then txs is in the output of every honest player by time $t + T(\delta, \Delta)$.

We say that a protocol has $T(\delta, \Delta)$ *optimistic confirmation time* if it is $T(\delta, \Delta)$-live when $f = 0$. Similarly, a protocol has $T(\delta, \Delta)$ *worst-case confirmation time* if it is $T(\delta, \Delta)$-live for any $f < n/3$. Now, suppose that a protocol proceeds in incrementing views (or iterations) $v = 1, 2, \ldots$, implemented by a local counter on each process. We say that a protocol has *expected* $T(\delta, \Delta)$ *view-based liveness* (w.r.t. the counter v) if:

- *Expected* $T(\delta, \Delta)$ *View-Based Liveness.* Fix any view $v \in \mathbb{N}$. Suppose that $(\mathcal{A}, \mathcal{Z})$ is additionally δ-bounded partially synchronous for some $\delta < \Delta$ and some GST. Suppose that $\mathcal{Z}$ never halts early and that it always delivers some input txs to every honest player before they enter view v. Suppose that every honest player enters view v by time t. If $t > \mathsf{GST}$, then txs is in the output of every honest player by time $t + T(\delta, \Delta)$, in expectation (over the coins of the execution).

Optimistic Responsiveness. (Variant of definition from [PS18].) We say that a blockchain protocol is *optimistically responsive* if it is has $T(\delta)$-time liveness for some function $T(\cdot)$ that is not a function of Δ, conditioned on all processes being honest. In other words, when all processes are honest, the liveness depends only on δ, not Δ.

3.2 Consistency of Simplex

In this section, we present a consistency proof for the protocol. We start with a simple fact about digital signatures:

Lemma 3.1. *With overwhelming probability in λ, for any honest process i, no honest process will see a valid signature of the form $\langle m \rangle_i$ in honest view unless process i previously signed m.*

Proof. This is by a direct reduction to the unforgeability of the signature scheme. □

We next state the standard *quorum-intersection lemma*, and for completeness provide its proof. (This is a very standard technique.)

Lemma 3.2. *Let $h \in \mathbb{N}$. Let b_h and b'_h be two distinct blocks s.t. neither are equal to the dummy block $\bot_h$. It cannot be that both b_h and b'_h are both notarized in honest view (except with negligible probability in λ).*

Proof. Consider a random execution and let b_h and b'_h be any two blocks of height h in the execution transcript, s.t. $b_h \neq b'_h$ and moreover $b_h \neq \bot_h$ and $b'_h \neq \bot_h$. We call a tuple (i, m) "good" if $m \in \{(\mathsf{vote}, h, b_h), (\mathsf{vote}, h, b'_h)\}$ and there exists a valid signature $\langle m \rangle_i$ in the view of some honest player. By the construction of the protocol, each honest process signs at most one of (vote, h, b_h) and (vote, h, b'_h). On the other hand, each corrupted player can sign both messages. Applying Lemma 3.1, then there are at most $(n-f)+f \cdot 2 = n+f < 4n/3$ good tuples with all but negligible probability in the security parameter. Now assume for the sake of contradiction that there are both notarizations for b_h and b'_h in honest view. Then there are $\geq 2n/3$ signatures for $\langle \mathsf{vote}, h, b_h \rangle$ and likewise $\geq 2n/3$ signatures for $\langle \mathsf{vote}, h, b'_h \rangle$ in honest view; thus there are $\geq 4n/3$ good tuples in honest view, which is a contradiction. ☐

We can also apply the exact same quorum-intersection technique to $\mathsf{finalize}$ and $\bot$ messages; for completeness, we write out the proof (again).

Lemma 3.3. *If there is a finalization for height h in honest view, $\bot_h$ cannot be notarized in honest view (except with negligible probability in λ).*

Proof. We call a tuple (i, m) "good" if $m \in \{(\mathsf{finalize}, h), (\mathsf{vote}, h, \bot_h)\}$ and there exists a valid signature $\langle m \rangle_i$ in the view of some honest player. By the construction of the protocol, each honest process signs at most one of $\langle \mathsf{finalize}, h \rangle$ or $\langle \mathsf{vote}, h, \bot_h \rangle$, whereas each corrupted player can sign either message. Applying Lemma 3.1, there are thus at most $(n-f)+f \cdot 2 = n+f < 4n/3$ good tuples (with all but negligible probability in λ). But now assume for the sake of contradiction that b_h is finalized and $\bot_h$ is also notarized. Since b_h is finalized, there are $\geq 2n/3$ signatures for $\langle \mathsf{finalize}, h \rangle$ in honest view, and likewise since $\bot_h$ is notarized, there are $\geq 2n/3$ signatures for $\langle \mathsf{vote}, h, \bot_h \rangle$ in honest view; thus there are $\geq 4n/3$ good tuples, which is a contradiction. ☐

The main theorem immediately follows:

Theorem 3.1 (Consistency)**.** *Suppose that two sequences of transactions, denote LOG and LOG', are both output output in honest view. Then either $\mathsf{LOG} \preceq \mathsf{LOG}'$ or $\mathsf{LOG}' \preceq \mathsf{LOG}$ (with overwhelming probability in λ).*

Proof. Immediately, there must be two blockchains denoted $b_0, b_1, \ldots, b_h$ and $b_0, b'_1, \ldots, b'_{h'}$, such that both are finalized in honest view, where $\mathsf{LOG} \leftarrow \mathsf{linearize}(b_0, b_1, \ldots, b_h)$ and $\mathsf{LOG}' \leftarrow \mathsf{linearize}(b_0, b'_1, \ldots, b'_{h'})$. Without loss of generality, we assume that $h \leq h'$.

It suffices to show that $b_h = b'_h$ and moreover that $b_h \neq \bot_h$. In plainer English, the two chains should contain the same block at height h, and moreover this block is not the dummy block. Then by collision-resistant property of the hash function $H(\cdot)$, the parent chains are the same $b_0, b_1, \ldots, b_{h-1} = b_0, b'_1, \ldots, b'_{h-1}$, and thus $\mathsf{LOG} \preceq \mathsf{LOG}'$.

To prove that $b_h = b'_h$, first observe that both $b'_{h'}$ and b_h are finalized in honest view, and thus both $b'_{h'}$ and b_h are notarized in honest view. Because $b'_{h'}$ is notarized in honest view, then some honest process must have voted for it in iteration h' which implies that b'_h is also notarized in honest view. By Lemma 3.3, observing that b_h is finalized in honest view, it must be that $b_h \neq \bot_h$, and likewise $b'_h \neq \bot_h$ (except with negligible probability in λ). Finally, we apply Lemma 3.2, which says that since b_h and b'_h are both notarized and not the dummy block, it must be that $b_h = b'_h$ (except with negligible probability), concluding the proof. ☐

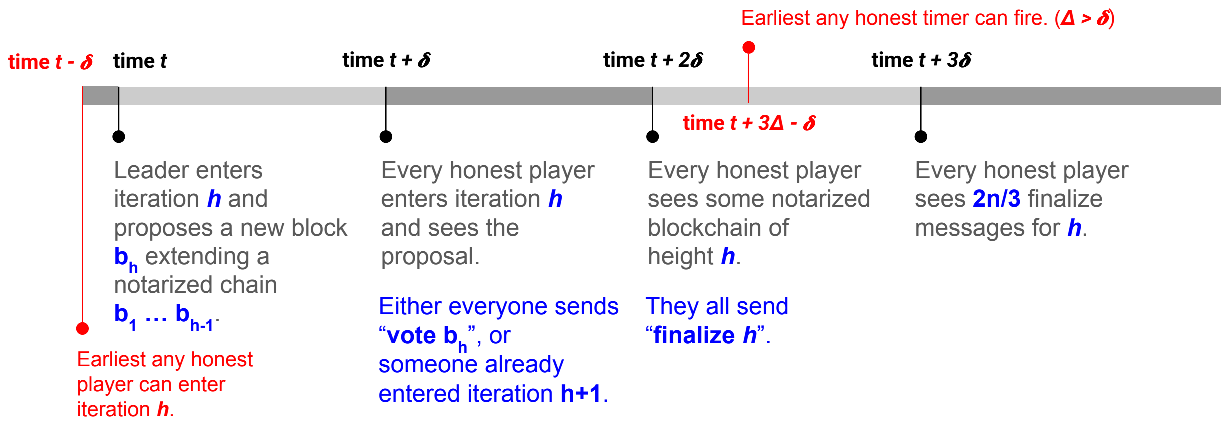

Figure 1: Timeline of events when the leader of iteration h is honest ($t > \mathsf{GST}$).

3.3 Liveness of Simplex

In this section, we analyze the liveness of the protocol. Throughout, suppose that $(\mathcal{A}, \mathcal{Z})$ is additionally δ-bounded partially synchronous, for some $\delta < \Delta$. Recall that the protocol is parametrized by Δ and not δ. When we say that "an honest process has entered iteration h by time t", we mean that the process previously entered iteration h at some time $t' \leq t$; when time t comes around, the process may well be in a larger iteration $h' > h$.

Lemma 3.4 (Synchronized Iterations)**.** *If some honest process has entered iteration h by time t, then every honest process has entered iteration h by time* $\max(\mathsf{GST}, t + \delta)$.

Proof. By the assumption that some honest process p has entered iteration h by time t, we know that process p must have seen a notarized blockchain of height $h - 1$ at or before time t. By the protocol design, p will multicast their view of this notarized blockchain immediately before entering iteration h. Subsequently, every honest process will have seen a notarized blockchain of height $h-1$, and thus also a notarized blockchain for every height $h' \leq h-1$, by time $\max(\mathsf{GST}, t+\delta)$. Thus, by time $\max(\mathsf{GST}, t+\delta)$, every honest process that is not yet in an iteration $\geq h$ will have incremented its iteration number until it is in iteration h. □

We show that the proposal confirmation time is 3δ, and that the optimistic block time is 2δ.

Lemma 3.5 (The Effect of Honest Leaders)**.** *Let h be any iteration with an honest leader L_h. Suppose that L_h entered iteration h by some time $t > \mathsf{GST}$. Then, with all but negligible probability, every honest process will have entered iteration $h+1$ by time $t+2\delta$. Moreover, every honest process will see a finalized block at height h, proposed by L_h, by time $t + 3\delta$.*

We break the proof down into two subclaims.

Subclaim 3.1. *Every honest player will see a notarized blockchain of height h, and thus enter iteration $h + 1$, by time $t + 2\delta$ (except with negligible probability).*

Proof. Recall that L_h enters iteration h by time t. Thus, L_h must multicast a proposal for a new non-dummy block b_h by time t. Thus, by time $t+\delta$ (observing that $t > \mathsf{GST}$), every honest process must have seen a valid proposal from the leader for b_h. There are now two cases:

- *Case 1.* Every honest process p casts a vote $\langle \mathsf{vote}, h, b_h \rangle_p$ by time $t + \delta$. Subsequently every honest process will see a notarization for b_h and thus a notarized blockchain of height h by time $t + 2\delta$, if not earlier, and enter iteration $h + 1$, as required.

- *Case 2.* There is some honest process p that did not multicast a vote $\langle \mathsf{vote}, h, b_h \rangle_p$ by time $t + \delta$. However, by Lemma 3.4, every honest process should have entered iteration h by time $t+\delta$, so the only way this could happen is if p entered iteration $h+1$ before time $t+\delta$. Then, every honest process will have entered iteration $h+1$ (and thus seen a notarized blockchain of height h) by time $t + 2\delta$, again by Lemma 3.4. (This case may have occured if, for instance, p saw a notarization for L_h's proposed block without seeing the proposal itself.) □

Subclaim 3.2. *Every honest player p will multicast $\langle \mathsf{finalize}, h \rangle_p$ by time $t + 2\delta$, and thus see a finalized block of height h by time $t + 3\delta$ (except with negligible probability).*

Proof. By Subclaim 3.1, every honest player sees a notarized blockchain of height h (thus finishing iteration h) by time $t + 2\delta$. We will show below that no honest player's timer for iteration h can fire before time $\leq t+2\delta$. Then, when each honest player p finishes iteration h, they must multicast a $\langle \mathsf{finalize}, h \rangle_p$ message, as their timer cannot have fired yet, showing the claim.

Let $t' \leq t$ be the time at which the first honest process enters iteration h. By Lemma 3.4, all honest processes—including the leader L_h—will have entered iteration h by $\max(\mathsf{GST}, t' + \delta)$, implying that $t \leq t' + \delta$ (since t is strictly greater than GST). The earliest an honest timer can fire is at or after $t' + 3\Delta > t' + 3\delta \geq t + 2\delta$ time (noting that $\Delta > \delta$), as desired. □

Finishing the Proof of Lemma 3.5. It remains to show that this finalized block is proposed by the leader. Recall that by Subclaim 3.2, every honest player p will have seen a finalized block b_h of height h by time $t + 3\delta$. Applying Lemma 3.3, we know that $b_h \neq \perp_h$. Thus, b_h must be proposed by L_h, because for it to be notarized, some honest player must have voted for it. The lemma follows. □

Theorem 3.2 (Optimistic Confirmation Time)**.** *Simplex has an optimistic confirmation time of 5δ.*

Proof. Suppose that there is a set of transactions txs in the view of every honest player by time t, where $t > \mathsf{GST}$, where txs is not yet in the output of any honest player. Let h be the highest iteration that any honest player is in at time t. There are two cases. If L_h has not entered iteration h yet by time t, then by Lemma 3.4, it will enter iteration h by time $t+\delta$, at which point it proposes a blockchain that contains txs; applying Lemma 3.5 then completes the proof. In the second case, by time t, L_h has already started iteration h, and by Lemma 3.5 every honest process will be in iteration $h + 1$ by time $t + 2\delta$, and see a finalized block from L_{h+1} by $t + 5\delta$. □

Now, we are ready to reason about worst-case confirmation time.

Lemma 3.6 (The Effect of Faulty Leaders)**.** *Suppose every honest process has entered iteration h by time t, for some $t >$ GST. Then every honest process will have entered iteration $h + 1$ by time $t + 3\Delta + \delta$.*

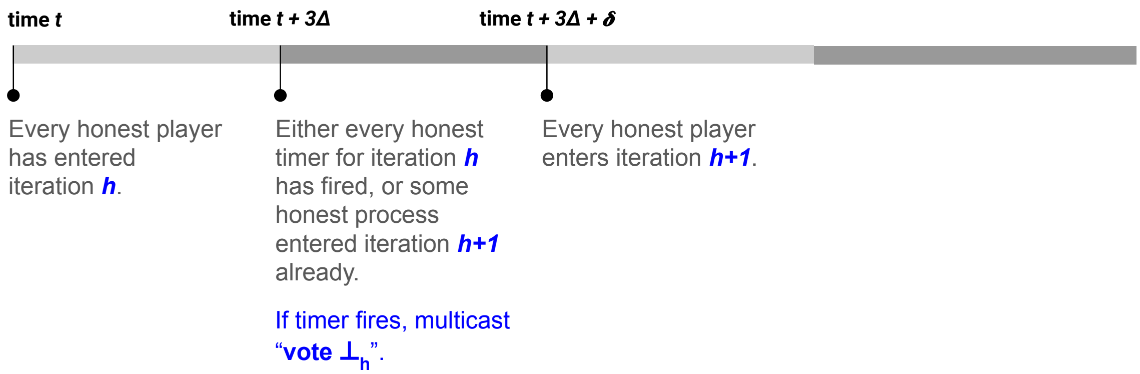

Figure 2: Timeline of events when the leader of iteration h is faulty ($t > \mathsf{GST}$).

Proof. There are two cases. First, suppose that for every honest process, its timer in iteration h fires; then every honest process p will cast a vote $\langle \mathsf{vote}, h, \perp_h \rangle_p$ at some time $\leq t + 3\Delta$, and subsequently this vote will be in the view of every honest process by time $\max(\mathsf{GST}, t + 3\Delta + \delta) = t + 3\Delta + \delta$. These votes comprise a notarization for $\perp_h$ and thus every honest process will see a notarized blockchain of height h by time $t + 3\Delta + \delta$ (if not earlier) and subsequently enter iteration $h + 1$ as required. The second case is if an iteration h timer does not fire for some honest process p. Then it must be that p entered iteration $h + 1$ at a time before its timer could fire, i.e. before time $t + 3\Delta$, and applying Lemma 3.4 yields the claim. ☐

Theorem 3.3 (Worst-Case Confirmation Time)**.** *Simplex has worst-case confirmation time of* $(4\delta + \omega(\log \lambda) \cdot (3\Delta + \delta))$.

Proof. Suppose that there is a set of transactions txs in the view of every honest player by time t, where $t > \mathsf{GST}$, where txs is not yet in the output of any honest player. Let h be the highest iteration that any honest player is in at time t. By Lemma 3.4 every every honest process must have entered iteration h by time $t + \delta$. Now, suppose that at least one iteration $i \in \{h + 1, \ldots, h + k\}$ has an honest leader L_i, for some choice of $k \in \mathbb{N}$. Then, applying Lemmas 3.5 and 3.6, every honest process will see a finalized block containing txs by time $t + 4\delta + k(3\Delta + \delta)$.

It remains to analyze the probability that, in a random execution, there is a sequence of k iterations in a row $h, h + 1, \ldots, h + k - 1$ s.t. for every $i \in [k]$, L_{h+i-1} is corrupt. First, observe that the attacker (and the environment) is PPT, and so there is a polynomial function $m(\cdot)$ s.t. any execution of the protocol on a security parameter 1^λ must contain at most $m(\lambda)$ number of iterations. Fix any $\lambda \in \mathbb{N}$. Recall that, for all $i \in [m(\lambda)]$, L_i is selected using a random oracle, and is thus corrupt with independent probability $f/n \leq 1/3$. (Recall that we instantiated the leader election oracle to be either $\mathsf{L}_i := H^*(i) \mod n$ or $\mathsf{L}_i := H^*(\sigma_i) \mod n$, where H^* is a random oracle and σ_i is a unique threshold signature on i.)

To help, we analyze the probability that in a sequence of $m(\lambda)$ unbiased coin flips, there is a consecutive sequence of at least k tails. There are at most $(m(\lambda) - k + 1) \cdot 2^{m(\lambda)-k}$ possible sequences with at least k consecutive tails, out of $2^{m(\lambda)}$ total; thus the probability is less than $\frac{(m(\lambda)-k+1)}{2^k}$. Immediately, the probability there are k corrupt leaders in a row is $< \frac{(m(\lambda)-k+1)}{2^k}$, since the probability a leader is corrupt is less than the probability an unbiased coin is tails. Observing that $\frac{(m(\lambda)-k+1)}{2^k}$ is a negligible function in λ when $k = \omega(\log \lambda)$, the theorem follows. □

For the sake of comparison, we also compute the expected view-based liveness. Recall that it says that, if every honest process sees some transaction txs before they enter iteration h, once every honest process enters iteration h, then txs will soon be confirmed:

Theorem 3.4 (Expected View-Based Liveness)**.** *Simplex has expected* $3.5\delta + 1.5\Delta$ *view-based liveness.*

Proof. Fix any iteration $h \in \mathbb{N}$. Suppose that there is a set of transactions txs in the view of every honest player before they enter iteration h, and moreover suppose that every honest player entered iteration h by some time $t > \mathsf{GST}$. Recall that for each iteration i, we defined the leader to be $\mathsf{L}_i := H^*(i) \mod n$, where H^* is a random oracle (chosen independently of GST and h). Immediately, for each $i \in \mathbb{N}$, L_i must be an honest player with independent probability $(n-f)/n \geq 2/3$. Denote X the number s.t. L_{h+X} is honest but, when $X > 0$, L_i is faulty $\forall i$ where $h \leq i < h + X$. Here X is a random variable, and immediately $\mathbb{E}[X] \leq 3/2 - 1 = 1/2$. Observe that, importantly, L_{h+X} will propose a blockchain that contains txs.

It remains to upper bound the time at which some honest process enters iteration $h + X$. By Lemma 3.6, every honest process will have entered iteration $h+X$ by time $t+X\cdot(3\Delta+\delta)$. Applying Lemma 3.5, we conclude that every honest process will see a finalized block proposed by L_{h+X} by time $t + 3\delta + (X) \cdot (3\Delta + \delta)$. Moreover, this block contains txs if not already in a previous block. Taking the expectation of the time elapsed since t, the theorem statement follows. □

3.4 Communication Complexity

Each iteration of the Simplex protocol (without subsampling) requires $O(n)$ multicasts. Note that we do not make any additional "relay rule assumptions", unlike other protocols in the multicast model (e.g. Streamlet [CS20a], Algorand Agreement [CGMV18], PaLa [CPS18]).

Lemma 3.7. *In each iteration h, each honest process will multicast at most* 4 *messages.*

Proof. For each iteration $h \in \mathbb{N}$, an honest process p will multicast at most one $\mathsf{propose}$ message, at most one vote message for a non-$\perp$ block, at most one of $\langle \mathsf{vote}, h, \perp_h \rangle_p$ and $\langle \mathsf{finalize}, h \rangle_p$, and will relay their view of a notarized blockchain of height h at most once. □

Each message is at most size $O(\lambda^\varepsilon \cdot n)$ (since a notarization contains up to n signatures of $O(\lambda^\varepsilon)$ size each, where $0 < \varepsilon \leq 1$), so in each iteration, at most $O(\lambda^\varepsilon \cdot n^2)$ bits total are multicast (in the system as a whole, summing over all players). We note that this matches the multicast bit complexity of the base Algorand Agreement protocol [CGMV18]. By using now standard techniques, we can bring down the multicast bit complexity to $O(\lambda^\varepsilon \cdot \mathsf{polylog}(\lambda))$ through subsampling the committee of voters [CM19]. Assuming the sub-exponential security of OWFs, then the length of a signature can be made to be $O(\mathsf{polylog}(\lambda))$, yielding a total multicast bit complexity of $O(\mathsf{polylog}(\lambda))$. We do not innovate on these methods, and focus on the base protocol.

Multicast can be implemented in various different ways on an underlying communication network. In a point-to-point network, each multicast involves sending the message to every other player; thus, our protocol—without subsampling—has $O(\lambda^\varepsilon \cdot n^3)$ total bit complexity in a point-to-point setting (counting every bit that was sent in the whole system). However, in modern peer-to-peer networks or gossip networks, peers do not need to directly communicate with every other peer. In the same vein as e.g. [BCP15, CCG+15] or the implementation of [GHM+17], it is possible to simulate all-to-all communication if each peer talks to $O(\mathsf{polylog}(n))$ other peers (if chosen appropriately). Then, total communication can be reduced to $O(\lambda^{\varepsilon} \cdot n^2) \cdot \mathsf{polylog}(n)$ bits (assuming that players do not relay the same message twice when gossiping messages). This may be reduced further by subsampling (again, following the techniques in Algorand).

We note that many nice works, starting with Hotstuff [YMR+19], propose base protocols that achieve $O(\lambda^{\varepsilon} \cdot n)$ bit complexity in a point-to-point setting, when all players are honest. Pessimistically, the bit complexity is $O(\lambda^{\varepsilon} \cdot n^2)$. Hotstuff and many follow-up works use threshold signatures to compress the size of notarizations, which requires a private setup, making it unclear how to do subsampling. Indeed, our protocol with subsampling (without gossip) matches the asymptotic communication complexity of their base protocols, achieving $O(\lambda^{\varepsilon} \cdot \mathsf{polylog}(\lambda) \cdot n)$ total bit complexity, or $O(\mathsf{polylog}(\lambda) \cdot n)$ assuming sub-exponentially secure OWFs. Moreover, this bound holds both pessimistically and optimistically for our protocol, as opposed to just the optimistic case in Hotstuff.

4 Related Work

The roots of consensus research in the permissioned and partially-synchronous setting dates back to the seminal paper of [DLS88]. Subsequently, the classical approaches in Paxos [LAM98] and PBFT [CL+99] became mainstream and were further studied in [KAD+07, GKQV10, MA06, OO14]. These protocols typically adopt an expensive or complex "view-change" phase for switching out a faulty leader for a new leader, and are built mainly for the stable leader setting, where a single leader can propose many blocks in a row without ceding its leadership.

More recently, the rise of blockchain applications (in particular, Proof-of-Stake systems) motivated a new line of work towards building fairer and simpler consensus protocols, where each leader generally only gets to propose a single block. In particular, Casper FFG [BG17], Hotstuff (v1) [AGM18], Tendermint [BKM18], and PaLa [CPS18] introduced a new "streamlined" approach where consensus is reached on many pipelined blocks at once, largely avoiding the complexity of a dedicated view-change subroutine. All four protocols proceed in incrementing epochs $e = 1, 2, \ldots$ and have a similar voting rule, where voters vote only for proposals extending the "freshest block" they've seen, that is, the one with the highest epoch number. As a consequence of the voting rule, the protocols maintain some intricacy when arguing liveness (i.e. the proposer may have to wait an extra 2Δ before proposing a new block, if the previous proposer crashed).[12] (Note that Simplex does not have this intricacy.) Somewhat separately, but also building on the streamlined paradigm, Streamlet [CS20a] achieves a simple protocol description but has worse optimistic and pessimistic liveness.

PBFT [CL+99] remains the fastest protocol when leaders are honest, requiring only 3δ timesteps to confirm transactions; [ANRX21] showed that this is optimal in the partially synchronous setting for $f \leq n/3$. (We mention works such as Parametrized FaB Paxos [MA06] and SBFT [GAG+19], which achieve even faster optimistic confirmation when $f \leq n/5$, but this requires that the fraction of faulty voters also be small, in addition to honest leaders.) However, we are more interested in the setting with rotating/random leaders, due to issues of fairness. (Random leaders also give

[12] Sometimes, this is referred to as the 'hidden lock problem'.

	Proposal Conf. Time	Optimistic Block Time	Expected View-Based Liveness ($\gamma := \frac{n}{n-f}$)	Expected Comm. Complexity
Simplex	$\mathbf{3\delta}$	$\mathbf{2\delta}$	$\mathbf{3\delta + (\gamma - 1) \cdot (3\Delta + \delta)}$	$O(n)$ multicasts†
Algorand* [CGMV18]	$\mathbf{3\delta}$	3δ	$3\delta + (\gamma - 1) \cdot (4\Delta + 2\delta)$	$O(n)$ multicasts
ICC [CDH+22]	$\mathbf{3\delta}$	$\mathbf{2\delta}$	$3\delta + (\gamma - 1) \cdot (\gamma \cdot (2\Delta + 2\delta) + 2\delta)$	$O(n)$ multicasts
PaLa [CPS18]	4δ	$\mathbf{2\delta}$	$4\delta + (\gamma^2 - 1) \cdot (5\Delta + \delta) + (\gamma - 1) \cdot 2\delta + \gamma \cdot 2\Delta$	$O(n)$ multicasts
Pipelined Fast-Hotstuff^ [JNFG20] Jolteon^ [GKKS+22]	5δ	$\mathbf{2\delta}$	$5\delta + (\gamma^3 - 1) \cdot (4\Delta + \delta) + (\gamma^2 + \gamma - 2) \cdot 2\delta$	$O(n^2)$ messages
Chained Hotstuff (v6)^ [YMR+19]	7δ	$\mathbf{2\delta}$	$7\delta + (\gamma^4 - 1) \cdot (3\Delta + \delta) + (\gamma^3 + \gamma^2 + \gamma - 3) \cdot 2\delta$	$O(n^2)$ messages
Streamlet [CS20a]	6Δ	2Δ	$6\Delta + (\gamma^5 + \gamma^4 + \gamma^3 + \gamma^2 + \gamma - 5) \cdot 2\Delta$	$O(n)$ multicasts
P-Sync Bullshark‡ [SGSKK22b]	6δ	3δ	$6\delta + (\gamma - 1) \cdot 18\Delta$	$O(n^2)$ messages
Hotstuff (v1)^ [AGM18, Abr22] Casper FFG^ [BG17] Chained Tendermint^ [BKM18, Abr22]	5δ	$\mathbf{2\delta}$	$5\delta + (\gamma^3 - 1) \cdot (5\Delta + \delta) + \gamma^2 \cdot 2\Delta$	$O(n^2)$ messages

^With random leaders. *Base protocol without sortition, with optimistic responsiveness. †Following the techniques of Algorand [GHM+17], by using subsampling, many (if not all) of the protocols here built for the multicast model can be adapted to use only $\mathsf{polylog}(\lambda)$ multicasts per block. ‡Assuming that it takes 3δ to generate a vertex (optimistically), and where the timeout is set to 9Δ according to Section C.2 in [SGSKK22a].

Table 2: Extended Comparison of Popular Consensus Protocols (Random Leaders)

better worst-case confirmation time.) Consequently, many works have adapted PBFT to a setting with rotating leaders; we mention Algorand [CGMV18], Fast-Hotstuff [JNFG20], and Jolteon [GKKS+22]. In particular, Pipelined Fast-Hotstuff and Jolteon propose streamlined versions of PBFT-like protocols that also seek to reduce the complexity of the view-change, again by pipelining proposals; both are slower than Algorand. Algorand and Fast-Hotstuff (without pipelining) achieve 3δ proposal confirmation time, but worse 3δ block time.

A major concern when designing modern consensus protocols is that of communication complexity. A low communication complexity is essential for a scalable protocol. In the multicast model, Algorand [CM19] showed an elegant way to subsample the committee of voters to achieve

$\mathsf{polylog}(\lambda)$ multicasts (or $O(n)$ messages in the point-to-point model). We remark that their techniques also apply to our protocol. (If we additionally trade off optimistic responsiveness, Algorand [CM19] can also achieve *adaptive security*.) In the point-to-point messaging model, a nice line of works, starting with Hotstuff [YMR$^+$19], seeked to reduce communication complexity by funneling all messages through the leader (a technique from [Rei94]) and by using threshold signatures to reduce the size of certificates (a technique from [CKS00]). Hotstuff, Pipelined Fast-Hotstuff [JNFG20] and Jolteon [GKKS$^+$22] all achieve $O(n)$ messages per block optimistically, and $O(n^2)$ messages in the case of a faulty leader (or better when amortized over many round-robin leaders, depending on the implementation of the 'pacemaker'). It remains unclear how realistic it is to funnel communication through a single leader on a peer-to-peer network. We note that their optimizations to message size using threshold signatures or aggregate signatures also apply to other protocols; of course, threshold signatures require a much stronger private setup (e.g. see [CKS00]) that we wish to avoid. Apart from requiring additional complexity to instantiate the private setup, it is also not clear how to do subsampling for protocols that require threshold signatures, as an example.

Our protocol has a similar 'notarize/finalize' voting procedure to that of [CDH$^+$22] (Dfinity ICC) and an old variant of Streamlet ([CS20b], Appendix A), but uses different techniques for proposals and for switching leaders. The ICC protocol [CDH$^+$22] is also quite nice but does not use timeout/dummy blocks, and incurs some complexity in how they rank leaders and allow multiple leaders in each round (they additionally assume threshold signatures and a trusted private setup, which we avoid). Consequently, if the first leader in a round is corrupt, they need essentially two honest leaders in a row to confirm a subsequent block. It is worth exploring whether their techniques for block dissemination can be used to improve Simplex. Independently, we also note structural similarities between Simplex and the Graded Binding Crusader Agreement protocol (using a weak common coin) in [ABDY22], albeit that protocol is for the asynchronous setting.

In a somewhat distinct line of work, [SGSKK22a] propose a partially-synchronous version of Bullshark built on top of what they call a DAG. These protocols are well-suited for the asynchronous setting, but when adapted for partial-synchrony, much care is needed when integrating timeouts into the DAG construction, to ensure that a slow honest leader is not left behind before it can propose a block; moreover, it trades-off latency.

We summarize the comparisons in Table 3.4 (for the random leader setting). When computing the expected view-based liveness, we define a parameter $\gamma := n/(n-f)$ that corresponds to the inverse fraction of eligible leaders who are honest.[13] Table 1 presents concrete values for expected liveness when $f = \lceil n/3 \rceil - 1$. Importantly, we remark that the landscape of consensus protocols is rich and ever-changing; this survey may not be comprehensive. In particular, while we compare only theoretical works here, much has been done to improve the performance of consensus protocols in practice by the systems community.

5 Other Remarks

In practice, one may in fact choose to run Simplex on three disjoint sets of players: a set of proposers, a set of validators (who validates blocks and votes for valid blocks only), and a set of "progress detectors" (who sends finalize votes or $\perp$ votes). Members of the validating set do not

[13] Another reason we use a different variable is to make clear that it is possible to elect leaders from an entirely different set of processes than the main set of protocol processes; indeed, our protocol supports a setting where most leaders are corrupt (i.e. $\gamma > 3/2$) even when the number of faulty voters is required to be small ($f < n/3$).

send ⊥ votes. Importantly, no state needs to be shared across the three roles, making it easy to distribute the work necessary to run the protocol, and also to incentivize each role in a different way. For example, the block itself only needs to be delivered to the block validators (and clients).

Bibliography

[AAC+05] Amitanand S Aiyer, Lorenzo Alvisi, Allen Clement, Mike Dahlin, Jean-Philippe Martin, and Carl Porth. Bar fault tolerance for cooperative services. In *Proceedings of the twentieth ACM symposium on Operating systems principles*, pages 45–58, 2005.

[ABDY22] Ittai Abraham, Naama Ben-David, and Sravya Yandamuri. Efficient and adaptively secure asynchronous binary agreement via binding crusader agreement. *Cryptology ePrint Archive*, 2022.

[Abr22] Ittai Abraham. Two round hotstuff. `https://decentralizedthoughts.github.io/2022-11-24-two-round-HS/`, 2022. Accessed: 2022-12-30.

[AGM18] Ittai Abraham, Guy Gueta, and Dahlia Malkhi. Hot-stuff the linear, optimal-resilience, one-message bft devil. *CoRR, abs/1803.05069*, 2018.

[AJK+22] Sarah Allen, Ari Juels, Mukti Khaire, Tyler Kell, and Siddhant Shrivastava. Nfts for art and collectables: Primer and outlook. 2022.

[AMS19] Ittai Abraham, Dahlia Malkhi, and Alexander Spiegelman. Asymptotically optimal validated asynchronous byzantine agreement. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 337–346, 2019.

[ANRX21] Ittai Abraham, Kartik Nayak, Ling Ren, and Zhuolun Xiang. Good-case latency of byzantine broadcast: A complete categorization. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, pages 331–341, 2021.

[Apa] Apache Software Foundation. ZooKeeper internals. `https://zookeeper.apache.org/doc/r3.4.13/zookeeperInternals.html`. Accessed: 2023-02-24.

[B+14] Vitalik Buterin et al. A next-generation smart contract and decentralized application platform. *white paper*, 3(37):2–1, 2014.

[BCC+19] Mathieu Baudet, Avery Ching, Andrey Chursin, George Danezis, François Garillot, Zekun Li, Dahlia Malkhi, Oded Naor, Dmitri Perelman, and Alberto Sonnino. State machine replication in the libra blockchain. *The Libra Assn., Tech. Rep*, 7, 2019.

[BCNP04] Boaz Barak, Ran Canetti, Jesper Buus Nielsen, and Rafael Pass. Universally composable protocols with relaxed set-up assumptions. In *45th Annual IEEE Symposium on Foundations of Computer Science*, pages 186–195. IEEE, 2004.

[BCP15] Elette Boyle, Kai-Min Chung, and Rafael Pass. Large-scale secure computation: Multi-party computation for (parallel) ram programs. In *Advances in Cryptology–CRYPTO 2015: 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*, pages 742–762. Springer, 2015.

[BDD+21] Carsten Baum, Bernardo David, Rafael Dowsley, Jesper Buus Nielsen, and Sabine Oechsner. Tardis: a foundation of time-lock puzzles in uc. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 429–459. Springer, 2021.

[BG17] Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget. *arXiv preprint arXiv:1710.09437*, 2017.

[BKM18] Ethan Buchman, Jae Kwon, and Zarko Milosevic. The latest gossip on bft consensus. *arXiv preprint arXiv:1807.04938*, 2018.

[BMM14] Michael Backes, Praveen Manoharan, and Esfandiar Mohammadi. Tuc: Time-sensitive and modular analysis of anonymous communication. In *2014 IEEE 27th Computer Security Foundations Symposium*, pages 383–397. IEEE, 2014.

[Bur06] Mike Burrows. The chubby lock service for loosely-coupled distributed systems. In *Proceedings of the 7th symposium on Operating systems design and implementation*, pages 335–350, 2006.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pages 136–145. IEEE, 2001.

[CCG+15] Nishanth Chandran, Wutichai Chongchitmate, Juan A Garay, Shafi Goldwasser, Rafail Ostrovsky, and Vassilis Zikas. The hidden graph model: Communication locality and optimal resiliency with adaptive faults. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science*, pages 153–162, 2015.

[CDH+22] Jan Camenisch, Manu Drijvers, Timo Hanke, Yvonne-Anne Pignolet, Victor Shoup, and Dominic Williams. Internet computer consensus. In *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing*, pages 81–91, 2022.

[CGMV18] Jing Chen, Sergey Gorbunov, Silvio Micali, and Georgios Vlachos. Algorand agreement: Super fast and partition resilient byzantine agreement. *Cryptology ePrint Archive*, 2018.

[CHMV17] Ran Canetti, Kyle Hogan, Aanchal Malhotra, and Mayank Varia. A universally composable treatment of network time. In *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*, pages 360–375. IEEE, 2017.

[CKS00] Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in constantinopole: practical asynchronous byzantine agreement using cryptography. In *Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing*, pages 123–132, 2000.

[CL+99] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OsDI*, volume 99, pages 173–186, 1999.

[CM19] Jing Chen and Silvio Micali. Algorand: A secure and efficient distributed ledger. *Theoretical Computer Science*, 777:155–183, 2019.

[CPS18] TH Hubert Chan, Rafael Pass, and Elaine Shi. Pala: A simple partially synchronous blockchain. *Cryptology ePrint Archive*, 2018.

[CR93] Ran Canetti and Tal Rabin. Fast asynchronous byzantine agreement with optimal resilience. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 42–51, 1993.

[CS20a] Benjamin Y Chan and Elaine Shi. Streamlet: Textbook streamlined blockchains. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, pages 1–11, 2020.

[CS20b] Benjamin Y Chan and Elaine Shi. Streamlet: Textbook streamlined blockchains (earlier version). Cryptology ePrint Archive, Paper 2020/088, 2020. `https://eprint.iacr.org/archive/2020/088/20200204:124247`.

[DGK+20] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 910–927. IEEE, 2020.

[DLS88] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM (JACM)*, 35(2):288–323, 1988.

[GAG+19] Guy Golan Gueta, Ittai Abraham, Shelly Grossman, Dahlia Malkhi, Benny Pinkas, Michael Reiter, Dragos-Adrian Seredinschi, Orr Tamir, and Alin Tomescu. Sbft: a scalable and decentralized trust infrastructure. In *2019 49th Annual IEEE/IFIP international conference on dependable systems and networks (DSN)*, pages 568–580. IEEE, 2019.

[GHM+17] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th symposium on operating systems principles*, pages 51–68, 2017.

[GKKS+22] Rati Gelashvili, Lefteris Kokoris-Kogias, Alberto Sonnino, Alexander Spiegelman, and Zhuolun Xiang. Jolteon and ditto: Network-adaptive efficient consensus with asynchronous fallback. In *Financial Cryptography and Data Security: 26th International Conference, FC 2022, Grenada, May 2–6, 2022, Revised Selected Papers*, pages 296–315. Springer, 2022.

[GKL15] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Advances in Cryptology-EUROCRYPT 2015: 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, pages 281–310. Springer, 2015.

[GKQV10] Rachid Guerraoui, Nikola Knežević, Vivien Quéma, and Marko Vukolić. The next 700 bft protocols. In *Proceedings of the 5th European Conference on Computer Systems*, EuroSys '10, pages 363–376, New York, NY, USA, 2010. ACM.

[HCPS19] T-H Hubert Chan, Rafael Pass, and Elaine Shi. Consensus through herding. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 720–749. Springer, 2019.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.

[JNFG20] Mohammad M Jalalzai, Jianyu Niu, Chen Feng, and Fangyu Gai. Fast-hotstuff: A fast and resilient hotstuff protocol. *arXiv preprint arXiv:2010.11454*, 2020.

[KAD+07] Ramakrishna Kotla, Lorenzo Alvisi, Michael Dahlin, Allen Clement, and Edmund L. Wong. Zyzzyva: speculative byzantine fault tolerance. In *Proceedings of the 21st ACM Symposium on Operating Systems Principles 2007, SOSP 2007, Stevenson, Washington, USA, October 14-17, 2007*, pages 45–58, 2007.

[KKKNS21] Idit Keidar, Eleftherios Kokoris-Kogias, Oded Naor, and Alexander Spiegelman. All you need is dag. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, pages 165–175, 2021.

[KLP05] Yael Tauman Kalai, Yehuda Lindell, and Manoj Prabhakaran. Concurrent general composition of secure protocols in the timing model. In *Proceedings of the thirty-seventh annual ACM Symposium on Theory of Computing*, pages 644–653, 2005.

[KMTZ13] Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. Universally composable synchronous computation. In *Theory of Cryptography Conference*, pages 477–498. Springer, 2013.

[KZGJ20] Mahimna Kelkar, Fan Zhang, Steven Goldfeder, and Ari Juels. Order-fairness for byzantine consensus. In *Advances in Cryptology–CRYPTO 2020: 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17–21, 2020, Proceedings, Part III 40*, pages 451–480. Springer, 2020.

[KZZ16] Aggelos Kiayias, Hong-Sheng Zhou, and Vassilis Zikas. Fair and robust multi-party computation using a global transaction ledger. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 705–734. Springer, 2016.

[LAM98] LESLIE LAMPORT. The part-time parliament. *ACM Transactions on Computer Systems*, 16(2):133–169, 1998.

[Lam01] Leslie Lamport. Paxos made simple. *ACM SIGACT News (Distributed Computing Column) 32, 4 (Whole Number 121, December 2001)*, pages 51–58, 2001.

[LSP82] LESLIE LAMPORT, ROBERT SHOSTAK, and MARSHALL PEASE. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982.

[MA06] Jean-Philippe Martin and Lorenzo Alvisi. Fast byzantine consensus. *IEEE Trans. Dependable Secur. Comput.*, 3(3), 2006.

[MXC+16] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of bft protocols. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 31–42, 2016.

[Nak08] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Decentralized business review*, page 21260, 2008.

[OO14] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*, USENIX ATC'14, page 305–320, USA, 2014. USENIX Association.

[PS17] Rafael Pass and Elaine Shi. The sleepy model of consensus. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 380–409. Springer, 2017.

[PS18] 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.

[Rab83] Michael O Rabin. Randomized byzantine generals. In *24th annual symposium on foundations of computer science (sfcs 1983)*, pages 403–409. IEEE, 1983.

[Rei94] Michael K Reiter. Secure agreement protocols: Reliable and atomic group multicast in rampart. In *Proceedings of the 2nd ACM Conference on Computer and Communications Security*, pages 68–80, 1994.

[Rom90] John Rompel. One-way functions are necessary and sufficient for secure signatures. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 387–394, 1990.

[SGSKK22a] Alexander Spiegelman, Neil Giridharan, Alberto Sonnino, and Lefteris Kokoris-Kogias. Bullshark: Dag bft protocols made practical. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 2705–2718, 2022.

[SGSKK22b] Alexander Spiegelman, Neil Giridharan, Alberto Sonnino, and Lefteris Kokoris-Kogias. Bullshark: The partially synchronous version. *arXiv preprint arXiv:2209.05633*, 2022.

[WLG+78] John H Wensley, Leslie Lamport, Jack Goldberg, Milton W Green, Karl N Levitt, Po Mo Melliar-Smith, Robert E Shostak, and Charles B Weinstock. Sift: Design and analysis of a fault-tolerant computer for aircraft control. *Proceedings of the IEEE*, 66(10):1240–1255, 1978.

[Yao82] Andrew C Yao. Protocols for secure computations. In *23rd annual symposium on foundations of computer science (sfcs 1982)*, pages 160–164. IEEE, 1982.

[YMR+19] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. Hotstuff: Bft consensus with linearity and responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 347–356, 2019.