Streamlined Blockchains: A Simple and Elegant Approach

(A Tutorial and Survey)

Elaine Shi
Cornell University runting@gmail.com

September 16, 2019

Abstract

A blockchain protocol (also called state machine replication) allows a set of nodes to agree on an ever-growing, linearly ordered log of transactions. The classical consensus literature suggests two approaches for constructing a blockchain protocol: 1) through composition of single-shot consensus instances often called Byzantine Agreement; and 2) through direct construction of a blockchain where there is no clear-cut boundary between single-shot consensus instances. While conceptually simple, the former approach precludes cross-instance optimizations in a practical implementation. This perhaps explains why the latter approach has gained more traction in practice: specifically, well-known protocols such as Paxos and PBFT all follow the direct-construction approach.

In this tutorial, we present a new paradigm called "streamlined blockchains" for directly constructing blockchain protocols. This paradigm enables a new family of protocols that are extremely simple and natural: every epoch, a proposer proposes a block extending from a notarized parent chain, and nodes vote if the proposal's parent chain is not too old. Whenever a block gains enough votes, it becomes *notarized*. Whenever a node observes a notarized chain with several blocks of consecutive epochs at the end, then the entire chain chopping off a few blocks at the end is *final*.

By varying the parameters highlighted in blue, we illustrate two variants for the partially synchronous and synchronous settings respectively. We present very simple proofs of consistency and liveness. We hope that this tutorial provides a compelling argument why this new family of protocols should be used in lieu of classical candidates (e.g., PBFT, Paxos, and their variants), both in practical implementation and for pedagogical purposes.

1 Introduction

In a blockchain protocol, a set of nodes seek to reach agreement on an ever-growing, linearly ordered log. It is helpful to think of this log as an ordered chain of blocks where each block may contain application-specific payload as well as metadata pertaining to the consensus protocol, and hence the name *blockchain*.

In this tutorial, we consider how to construct a blockchain protocol in a "permissioned" setting, assuming the existence of a public-key infrastructure and that the public key of every consensus node is common knowledge. This is the also the classical setting under which consensus has been studied for more than three decades. Classically, this problem was often called "State Machine Replication" [12, 13, 15] or "Byzantine Fault Tolerance" [3, 9]. In this work, we also refer to it as "consensus" for short.

Such permissioned blockchains can serve as the cornerstone not only for a private, consortium blockchain, but also for building open-access "proof-of-stake" systems. In a proof-of-stake setting,

a set of nodes (called a committee) are elected based on their stake in the system to vote in the consensus protocol. The election is typically repeated over time, using the blockchain protocol itself to agree on the next committee (and assuming the existence of an initial committee that is common knowledge).

The goal of this tutorial is to illustrate a new paradigm called "streamlined blockchains" that enables extremely simple and natural blockchain constructions. This new paradigm emerged as a result of the community's joint push at building better consensus protocols in the past few years, motivated by large-scale cryptocurrency applications. Elements of this idea were developed and improved in a sequence of works, including Casper-FFT [16], Dfinity [8], Hotstuff [1], Pili [5] and Pala [4], but understanding of this line of work still appears somewhat "scattered".

In this tutorial, we hope to describe the simplest possible embodiments of this idea, with concise and clean proofs that are suitable for pedagogy. We hope that this tutorial helps to illustrate the most compelling advantage of this new paradigm, i.e., its conceptual simplicity, making the resulting protocols desirable for practical implementation. We also contrast this new paradigm with classical blockchain constructions represented by Paxos [9], PBFT [3], and their variants. We hope that this will shed light on how the community's push in the past few years has enabled a leap: we now have practical blockchain constructions that are significantly simpler and fundamentally better than classical approaches.

1.1 Problem Statement

Slightly informally, we would like to construct a blockchain protocol satisfying two properties for all but a negligible fraction of executions:

- *Consistency*: if two blockchains chain and chain' are ever considered *final* by two honest nodes, it must be that $\mathsf{chain} \preceq \mathsf{chain}'$ or $\mathsf{chain}' \preceq \mathsf{chain}$ where $\preceq$ means "is a prefix of or equal to".
- *Liveness*: if an honest node receives a transaction, the transaction will appear in every honest node's finalized blockchain in a bounded amount of time.

In a cryptocurrency application, all transactions contained in a *final* chain are considered confirmed and the merchant may ship the product. If all nodes are honest and always correctly follow the prescribed protocol, then designing a blockchain protocol is trivial. We consider a setting where a subset of the nodes can be corrupt; corrupt nodes are controlled by a single adversary and they can deviate from the protocol arbitrarily — such a fault model is commonly referred to as Byzantine Faults in the classical distributed consensus literature.

In general, we can construct a blockchain protocol in two ways: 1) through composition of single-shot consensus instances; and 2) direct construction of a blockchain protocol where there is no clearly defined boundary between consensus instances. From a historical perspective, the study of distributed consensus in fact originated from the study of one-shot consensus protocols, often called Byzantine Agreement [10]. While composing single-shot instances is a conceptually clean approach towards building a blockchain, cross-instance performance optimizations are often challenging. This is arguably why later approaches such as Paxos and PBFT and their variants — also coinciding with most deployed systems — adopt the direct-construction approach. In this tutorial we will also focus on the direct-construction approach.

1.2 Classical Blockchain Protocols: A Bi-Modal Approach

Most approaches in the classical consensus literature adopt a bi-model approach. We illustrate the idea assuming that fewer than $n/3$ nodes are corrupt where n denotes the total number of nodes[1].

1.2.1 Normal Mode: A Natural Voting Protocol

The normal mode is simple and natural and works by super-majority voting. We shall explain the idea semi-formally, since this is the nice part of the protocol we would like to preserve in our new paradigm. Recall that every block is part of a blockchain and henceforth its index within the blockchain is called its position. We assume that every block encodes its own position.

Imagine that a designated proposer proposes blocks, and nodes vote on the proposed blocks by signing the block's hash. Whenever a block gains votes from at least $2n/3$ distinct nodes, it becomes final. If in a blockchain every block is final, then the chain is considered final too.

An important invariant is that *an honest node never votes for two distinct blocks at the same position* (even if the proposer is corrupt and proposes multiple blocks at the same position). This enforces consistency at every position, i.e., at each position, there cannot be two different blocks both gaining at least $2n/3$ votes. The proof of consistency is extremely simple: suppose that two different blocks at the same position both gain at least $2n/3$ votes. It must be that a set of at least $2n/3$ distinct nodes denoted S_1 have voted for one, and a set of at least $2n/3$ distinct nodes denoted S_2 have voted for the other. Obviously $|S_1 \cap S_2| \geq 2n/3 + 2n/3 - n = n/3$. Since fewer than $n/3$ nodes are corrupt, it must be that an honest node lives in the intersection $S_1 \cap S_2$ and has voted for both blocks at the same position — but this is ruled out by the aforementioned invariant.

Such a normal-mode protocol is extremely simple and natural, and it gives consistency as long as fewer than $n/3$ nodes are corrupt; and moreover, consistency does not rely on the proposer being honest. However, if the proposer is corrupt, e.g., if it stops making proposals or makes different proposals to different nodes, then liveness can be stalled and the blockchain can stop growing. We note also that here, consistency is guaranteed without having to make any network timing assumptions such as synchrony assumptions.

1.2.2 Recovery Mode: Ensuring Liveness

Given the aforementioned normal-mode protocol, the only remaining problem is how to achieve liveness when the proposer is corrupt. We informally explain how classical protocols deal with this problem without going into details, since this is the complicated part of classical approaches that we would ideally like to get rid of.

Most classical protocols such as Paxos, PBFT and their variants solve this problem by falling back to a recovery mode (often called "view change") whenever liveness is stalled. Typically the view change implements a mechanism to rotate to the next proposer such that progress can be resumed. Thus a view can be regarded as a phase of the protocol in which a specific node acts as the proposer. Without going into details, and perhaps unsurprisingly, from a technical standpoint the view change protocol must be a full-fledged consensus protocol offering both consistency and liveness (*c.f.* the normal mode guarantees only consistency assuming fewer than $n/3$ corrupt).

At an intuitive level, this perhaps explains why in most classical consensus approaches, the view change is often much more complicated to understand and subtle to implement correctly than the normal mode. In fact, the need for such a recovery mode often imposes more requirements

[1] Our exposition in spirit illustrates the ideas behind most classical blockchain constructions although our exposition is not necessarily faithful to any particular protocol. In fact, we give a simplified exposition of the technical ideas to maximally aid understanding.

on the normal mode too — and this is why most actual instantiations of this bi-modal idea such as Paxos and PBFT introduce more iterations of voting in the normal mode (unlike our earlier description that has only one iteration of voting). Very roughly, the additional iterations of voting in the normal mode give amplified properties that the recovery mode can make use of.

1.3 Streamlined Blockchains: A New Paradigm

Classical approaches are somewhat undesirable because most of the time we expect that the protocol should operate in the normal mode (since faults should not happen very often); however, the conceptual complications and the heavy-lifting in implementation stem from the complicated recovery path. Ideally, we would like to achieve the following holy grail:

> Can we have a blockchain protocol that is (almost) *as simple as the normal mode*?

Amazingly, it turns out that this is in fact possible! All the protocols we describe in this tutorial, for different settings, can be obtained by making small tweaks to the aforementioned normal-path voting protocol. Through these tweaks we now offer not just consistency but also liveness and thus there is no need for a separate recovery mode! Specifically, the entire protocol always follows a unified propose-vote paradigm as described below:

- Every epoch, a proposer proposes a block extending from a parent chain. Every block encodes its own epoch.
- Nodes vote on the proposed block if they have seen the parent chain's notarization and if the parent chain is not too old (where "old" means that the block contains a small epoch number, and we will specify the concrete parameter in the later sections).
- Whenever a block gains sufficiently many votes, it becomes *notarized*.
- Notarized does not mean *final*. Finality is determined as follows: if all blocks in a blockchain are notarized and the chain ends at several blocks of consecutive epochs, then the entire chain chopping off the trailing few blocks are considered final.

We show how to use this simple paradigm to obtain protocols under various network assumptions, by modifying the parameters highlighted in blue, and by slightly varying a couple other details such as how epochs are determined for different settings.

So what became of the view change? As mentioned earlier, in classical approaches the view change was necessary to attain liveness under a corrupt proposer. So technically, how can we achieve both consistency and liveness without the view change? In the streamlined blockchain paradigm described in this tutorial, basically every epoch embeds a proposer-rotation opportunity, and thus an implicit view change mechanism is already inherently baked in the protocol everywhere. This is arguably the coolest feature of this new paradigm: we show that the traditionally complicated view change can be embedded into an extremely simple paradigm by small tweaks to the normal-path voting protocol.

For this reason, another advantage of our streamlined blockchain protocols is that they readily support two distinct proposer-rotation policies: the *democracy-favoring* policy where one wishes to rotate proposer every block; and the classical *stability-favoring* policy (adopted by classical approaches such as Paxos and PBFT) where we stick to the same proposer until it starts to misbehave. In new cryptocurrency applications, the democracy-favoring policy may be more desired

due to better decentralization; however, a stability-favoring policy is likely more friendly towards performance optimizations.

Throughout this paper, we use the democracy-favoring policy for exposition. Some recent works [4, 5] have shown how minor tweaks to the protocol can support a stability-favoring policy[2].

2 A Blockchain Tolerating $< 1/3$ Corruptions

Recall that we consider a network consisting of n nodes numbered $0, 1, \ldots, n-1$ respectively. We assume that there is a public-key infrastructure such that all nodes' public keys are common knowledge. In this section we shall assume that fewer than $n/3$ nodes are corrupt. In our protocol, whenever a node *multicasts* a message to everyone, it means it sends this message to every node.

Delay parameter Δ. The protocol is parametrized with a parameter Δ which captures our a-priori guess of the maximum message delay. We will prove that *consistency holds regardless even if our guess of Δ is wrong and network delays are arbitrary.* However, liveness only holds during "periods of synchrony", i.e., periods in which honest messages are delivered in at most Δ rounds.

Remark 1. Although we assume that time progresses in discrete *rounds* in this tutorial, all the results still hold if the round is infinitesimally small, i.e., if time is continuous. We assume that all nodes have local clocks that increment per round. Since clock offsets can be absorbed by the network delay, our consistency proof holds even if clock offsets between nodes are arbitrarily large. However, unsynchronized clocks may stall liveness by preventing a period of synchrony from happening.

2.1 Valid Blockchain and Freshness

Our protocol will progress in epochs where each epoch contains 2Δ rounds, i.e., long enough for honest nodes to make a round trip during a period of synchrony.

Valid blockchain. A blockchain, often denoted chain, is an ordered sequence of blocks. Each block $\mathsf{chain}[\ell]$ where $\ell \geq 0$ is of the format $(e, \mathsf{TXs}, h_{-1})$, where e encodes the epoch number, TXs is application-specific payload (e.g., a set of transactions to confirm), and h_{-1} is the parent block's hash. In a valid blockchain chain, the 0-th block must be a special genesis block of the form $(0, \bot, \bot)$.

When we define a chain's length denoted $|\mathsf{chain}|$, it does not count the genesis block. This way, the chain's length is the same as the index of the last block. Henceforth for $\ell \geq 0$, we use the notation $\mathsf{chain}[:\ell]$ to denote the prefix of the blockchain up to the ℓ-th block. and $\mathsf{chain}[:-\ell]$ is an alias for $\mathsf{chain}[:m-\ell]$ where $m := |\mathsf{chain}|$ denotes the length of the blockchain. Similarly, $\mathsf{chain}[-\ell]$ is an alias for $\mathsf{chain}[m-\ell]$.

For a blockchain chain to be valid, all the blocks must have strictly increasing epoch numbers, and moreover for every $\ell \geq 0$, the block $\mathsf{chain}[\ell].h_{-1}$ must be equal to $H(\mathsf{chain}[:\ell-1])$. In our protocol, all protocol messages containing ill-formed blockchains are immediately discarded.

[2] Author's note: even if the syntactical changes to the protocol are minor, it is important that they be done correctly since some additional subtleties arise in the liveness proof for a stability-favoring policy. See the recent works [4, 5] for more explanation.

Freshness. For a blockchain chain, the larger $\mathsf{chain}[-1].e$ is the fresher chain is. Formally, we say that chain is fresher than chain' if $\mathsf{chain}[-1].e > \mathsf{chain}'[-1].e$. For a blockchain chain, $\mathsf{chain}[-1].e$ is also said to be the blockchain chain's epoch number.

2.2 Protocol

Now, imagine that the protocol proceeds in *epochs* numbered $1, 2, \ldots$. Each epoch is 2Δ rounds, i.e., the maximum round-trip delay during a period of synchrony. In each epoch $e \in \{1, 2, \ldots, \}$, we use a hash function H^* (i.e., a random oracle) to select a random node $(H^*(e) \mod n)$ to be the designated proposer — note that here we are using a democracy-favoring proposer-rotation policy as an example.

The protocol proceeds as follows where we assume that a node always signs every message it wants to send, and that every valid message must be tagged with the purported sender; further, nodes discard messages with invalid signatures. The notation "_" denotes a wildcard field.

Notarization: A valid vote for chain from node i is a valid signature from node i on $H(\mathsf{chain})$ where H is a global hash function chosen at random from a collision-resistant hash family upfront. A collection of at least $2n/3$ votes from distinct nodes on some chain is said to be a notarization for chain.

For each epoch $e = 1, 2, \ldots$**:**

- **Propose:** At the beginning of the epoch, node $(H^*(e) \mod n)$ proposes a new block $\mathsf{B} := (e, \mathsf{TXs}, h_{-1})$ extending the freshest notarized chain in its view denoted chain. Here TXs denotes a set of outstanding transactions to confirm and $h_{-1} = H(\mathsf{chain})$. The proposal, containing $\mathsf{chain}||\mathsf{B}$ and a notarization for chain, is signed and multicast to everyone — here chain is referred to as the parent chain of B.
- **Vote:** Every node performs the following: when the *first* valid proposal of the form $\mathsf{chain}||(e, \_, \_)$ is received from node $(H^*(e) \mod n)$ with a valid notarization on chain, vote on the proposal iff chain is at least as fresh as the freshest notarized chain the node has observed *at the beginning of the previous* epoch or if the current epoch $e = 1$.

  To vote on $\mathsf{chain}||\mathsf{B}$, simply multicast a signature on the value $H(\mathsf{chain}||\mathsf{B})$ to everyone.

Finalization: At any time, if a notarized chain has been observed ending at three consecutive epochs, then $\mathsf{chain}[:-2]$ is considered final.

Remark 2 (Block and chain as aliases of each other)**.** Suppose that there are no hash collisions, then due to the structure of the blockchain where every block must refer to its parent's hash, in fact a block $\mathsf{chain}[\ell]$ and the chain $\mathsf{chain}[:\ell]$ can be used interchangeably, since the block $\mathsf{chain}[\ell]$ uniquely defines the entire prefix $\mathsf{chain}[:\ell]$. Therefore, henceforth whenever convenient, we use "a vote or a notarization for $\mathsf{chain}[\ell]$" and "a vote or notarization for $\mathsf{chain}[:\ell]$" interchangeably.

Remark 3 (Practical considerations)**.** The above protocol is described in a way that maximizes conceptual simplicity. In practice, a couple of obvious optimizations can be made. First, the hash H can be computed incrementally by hashing the parent block's hash and the current block's contents. Second, the proposer need not include the entire parent chain in the proposal, it suffices to include the hash $h_{-1} = H(\mathsf{chain})$. When a proposal is received, if the recipient has not received a parent chain consistent with h_{-1}, it buffers this proposal until it has received a consistent parent chain.

2.3 Consistency Proof

We now present a very simple consistency proof. Recall that the adversary controls strictly fewer than $n/3$ nodes. Throughout, we assume that the signature and hash schemes are ideal, i.e., the adversary cannot forge honest nodes' signatures or find hash collisions. Technically we are removing from our consideration the negligible fraction of bad executions in which an honest node's signature is forged or hash collisions are found — all the lemmas and theorems below hold for all but the negligible fraction of such bad executions.

We say that some string is *in honest view* iff some honest node observes it at some point during the execution. The following simply lemma is in fact already proven in Section 1, but we restate it for completeness.

A simple fact is the following: if there is a notarization for chain in honest view, there must be a notarization $\mathsf{chain}[:-1]$ in honest view since if not, no honest node would have voted for chain and chain cannot gain notarization in honest view. Applying this argument inductively, if there is a notarization of chain in honest view then there must be a notarization of every prefix of chain in honest view.

Lemma 2.1 (Uniqueness per epoch)**.** *There cannot be two different blocks of epoch e both notarized in honest view.*

Proof. Suppose that two different blocks B_1 and B_2 of epoch e both gained notarization in honest view. Let S_1 be the set of at least $2n/3$ nodes who have signed B_1 and let S_2 be the set of at least $2n/3$ nodes who have signed B_2. It must be that $|S_1 \cap S_2| \geq 2n/3 + 2n/3 - n = n/3$. This means that at least one honest node is in $S_1 \cap S_2$, and this honest node must have signed both B_1 and B_2 in epoch e. By our protocol definition, every honest node signs only one epoch-e block in each epoch e. Thus we have reached a contradiction. □

Theorem 2.2 (Consistency)**.** *Suppose that* chain *and* chain' *are notarized chains in honest view both ending at three consecutive epochs, it must be that* $\mathsf{chain}[:-2] \preceq \mathsf{chain}'[:-2]$ *or* $\mathsf{chain}'[:-2] \preceq \mathsf{chain}[:-2]$.

Proof. Suppose that chain ends with three blocks of epochs $e-2$, $e-1$, and e, and chain' ends with three blocks of epochs $e'-2$, $e'-1$, and e'. Without loss of generality, assume that $e' \geq e$. For the sake of reaching a contradiction, suppose that $\mathsf{chain}[:-2]$ and $\mathsf{chain}'[:-2]$ are not prefixes of each other. Due to Lemma 2.1, chain' cannot have a block at epochs $e-2$, $e-1$, or e; since otherwise $\mathsf{chain}'[:-2]$ must contain the prefix $\mathsf{chain}[:-2]$ which ends at a block of epoch $e-2$. Therefore, there is some block in chain' with an epoch number greater than e. Let $e'' > e$ be the smallest epoch number greater than e in chain', and let $\mathsf{chain}'[\ell]$ be the block in chain' with epoch number e''. It must be that $\mathsf{chain}'[\ell-1]$ has epoch smaller than $e-2$.

Since (every prefix of) chain gained notarization in honest view, it must be that at least $2n/3$ distinct nodes have signed the block $\mathsf{chain}[-1]$ of epoch $e-1$, meaning that more than $n/3$ honest nodes have signed this block. Moreover, honest nodes can only sign this block in epoch $e-1$. This means that more than $n/3$ honest nodes have observed a notarization for $\mathsf{chain}[:-2]$ of epoch $e-2$ in epoch $e-1$, i.e., before the beginning of epoch e — let S denote this set of more than $n/3$ honest nodes. The set S therefore will not vote for $\mathsf{chain}'[\ell]$ in epoch $e'' > e$ which extends from a parent chain of epoch less than $e-2$; and thus $\mathsf{chain}'[\ell]$ cannot have gained notarization in honest view. □

2.4 Liveness Proof

Message delivery assumption during periods of synchrony. As mentioned earlier, a period of synchrony is a period with good network conditions such that all messages sent by honest nodes are delivered to the recipients within at most Δ rounds.

Without loss of generality, we shall assume that every honest node always echos (i.e., multicasts) every fresh message as soon as it is observed. Thus, during a period of synchrony, the following holds:

> *If an honest node has observed a message* m *in round* t*, then all honest nodes must have observed* m *by the beginning of round* $t + \Delta$ *if not earlier.*

Liveness proof. Suppose a period of synchrony eventually takes place. We now prove liveness during such a period of synchrony. Specifically, we prove that during a period of synchrony, honest nodes' finalized blockchains will grow whenever there are 3 consecutive epochs with honest proposers (note that under random proposer election, this takes $O(1)$ number of epochs in expectation).

To see this, it suffices to show that every honest node will vote on the proposal of an honest proposer — since an honest proposer makes a proposal at the beginning of the epoch e, as long as every honest node votes on it, the honest votes will have been received by all honest nodes by the beginning of epoch $e + 1$; and thus epoch $(e + 1)$'s proposer, if honest, will propose to extend a notarized chain ending at epoch $e + 1$. We now prove this.

If an honest node i rejects a proposal from an honest proposer j, it must be that the proposed block extends from a parent chain that is less fresh than the freshest notarized chain (denoted chain^*) observed at the beginning of the *previous* epoch. However, if i has observed chain^* at the beginning of the previous epoch, then due to the message delivery assumption during a period of synchrony, by the beginning of this epoch, node j must have observed it and thus j cannot have proposed to extend from a less fresh parent chain.

Remark 4. Alternatively, we can modify the proposer rotation policy for the same node to serve as a proposer for three consecutive epochs. In this case, progress will be made whenever an honest proposer makes proposals for 3 consecutive epochs.

3 A Synchronous Blockchain Tolerating Minority Corruptions

In the previous section, we presented a streamlined blockchain protocol whose consistency guarantee holds with arbitrary network delays, but whose liveness guarantee may hold only during periods of synchrony — such protocols are said to be secure in a "partially synchronous" network [6]. Due to a well-known lower bound by Dwork et al. [6], no partially synchronous protocol can tolerate $n/3$ or more corruptions, and therefore the protocol in the previous section is in fact optimal in resilience.

In this section we illustrate another streamlined blockchain protocol that tolerates up to minority corruptions. To achieve this, however, we must make a synchrony assumption even for the consistency proof. Recall that earlier in Section 2.4 we made the following synchrony assumption for proving liveness:

> *If an honest node has observed a message* m *in round* t*, then all honest nodes must have observed* m *by the beginning of round* $t + \Delta$ *if not earlier.*

In this section, we shall make this assumption for proving both consistency and liveness.

Remark 5. The consensus problem would be trivial if all honest nodes must observe every message m in the same round. In fact, in the synchronous setting, the crux of the consensus problem is essentially to overcome the Δ difference in the timing at which honest nodes observe the same message m.

3.1 Protocol

The protocol is almost identical as the one in Section 2 except for two modifications: 1) the parameters for forming a notarization and for finalizations are chosen differently; and 2) the finalization rule makes an additional check for conflicting proposals. The protocol is described below and the difference from the earlier protocol in Section 2 is highlighted in blue.

Notarization: A valid vote for chain from node i is a valid signature from node i on $H(\mathsf{chain})$ where H is a hash function chosen at random from a collision-resistant hash family. A collection of at least $n/2$ votes from distinct nodes on some chain is said to be a notarization for chain.

For each epoch $e = 1, 2, \ldots$**:**

- **Propose:** At the beginning of the epoch, node $(H^*(e) \mod n)$ proposes a new block $\mathsf{B} := (e, \mathsf{TXs}, h_{-1})$ extending the freshest notarized chain in its view denoted chain. Here TXs denotes a set of outstanding transactions to confirm and $h_{-1} = H(\mathsf{chain})$. The proposal, containing $\mathsf{chain}||\mathsf{B}$ and a notarization for chain, is signed and multicast to everyone — here chain is referred to as the parent chain of B.
- **Vote:** Every node performs the following: when the *first* valid proposal of the form $\mathsf{chain}||(e, \_, \_)$ is received from node $(H^*(e) \mod n)$ with a valid notarization on chain, vote on the proposal iff chain is at least as fresh as the freshest notarized chain the node has observed *at the beginning of the previous* epoch or if the current epoch $e = 1$.

  To vote on $\mathsf{chain}||\mathsf{B}$, simply multicast a signature on the value $H(\mathsf{chain}||\mathsf{B})$ to everyone.

Finalization: At any time, if a notarized chain has been observed ending at 6 blocks with consecutive epoch numbers, and moreover for each these 6 epoch numbers, no conflicting proposal (from an eligible proposer) for a different block has been seen, then the prefix $\mathsf{chain}[:-5]$ is final.

3.2 Consistency Proof

In comparison with Section 2.3, under minority corruption, the "uniqueness per epoch" lemma (Lemma 2.1) no longer holds. Consistency now crucially relies on the new finalization rule which additionally checks for conflicting proposals. We thus present a different but nonetheless simple consistency proof. Henceforth we use the notation $\mathsf{chain}\langle e\rangle$ to denote the block at epoch e in chain, and we use $\mathsf{chain}\langle : e\rangle$ to denote the prefix of chain up to and including the block of epoch e.

Lemma 3.1 (No contiguous skipping)**.** *Suppose that a notarized chain with two consecutive epoch numbers e and $e+1$ appear in honest view. Then, no notarized chain in honest view whose ending epoch at least e can skip all of the epochs $e, e+1, e+2, e+3$ (i.e., one of these epochs must be contained in the notarized chain).*

Proof. Let chain be the notarized chain in honest view with two consecutive epochs e and $e+1$. It must be that at least one honest node i has voted for $\mathsf{chain}\langle : e+1\rangle$ during epoch $e+1$, and thus i has observed a notarization for $\mathsf{chain}\langle : e\rangle$ in epoch $e+1$. Therefore all honest nodes must have

observed a notarization for $\mathsf{chain}\langle : e\rangle$ in epoch $e+2$, i.e., *by the beginning of* epoch $e+3$. Thus in any epoch $e' > e+3$ no honest node will vote to extend a parent chain whose epoch is smaller than e.

Suppose chain' is a notarized chain in honest view whose ending epoch is at least $e+4$ and moreover chain' does not contain the epochs $e, e+1, e+2, e+3$. Let e' be the smallest epoch in chain' that is greater than $e+3$. It must be that at least one honest node voted on $\mathsf{chain}'\langle : e'\rangle$ during epoch e' but this is impossible because $\mathsf{chain}'\langle : e'\rangle$'s parent has epoch smaller than e. □

Theorem 3.2 (Consistency)**.** *Suppose that an honest node i triggered the finalization rule on* chain *and an honest node j triggered the finalization rule on* chain'*, then it must be that either* $\mathsf{chain}[:-5] \preceq \mathsf{chain}'[:-5]$ *or* $\mathsf{chain}'[:-5] \preceq \mathsf{chain}[:-5]$.

Note that i and j can be the same or different node in the above theorem.

Proof. Suppose that chain ends at 6 consecutive epochs $e-5, e-4, \ldots, e$ and chain' ends at 6 consecutive epochs $e'-5, e'-4, \ldots, e'$. Without loss of generality, assume that $e' \geq e$.

Since chain contains two consecutive epochs $e-5$ and $e-4$, due to Lemma 3.1, chain' cannot skip all of epochs $e-5, e-4, e-3, e-2$. Therefore there must be a block in chain' at epoch $\widetilde{e} \in \{e-5, e-4, e-3, e-2\}$. Thus, at least one honest node must have voted for $\mathsf{chain}'\langle\widetilde{e}\rangle$ in epoch $\widetilde{e}$, and this honest node must have observed a proposal for $\mathsf{chain}'\langle\widetilde{e}\rangle$ from an eligible proposer in epoch $\widetilde{e}$. This means that all honest nodes must have observed a proposal for $\mathsf{chain}'\langle\widetilde{e}\rangle$ from an eligible proposer by the beginning of $\widetilde{e}+2 \leq e$.

Notice that a notarization for chain cannot appear in honest view before epoch e since honest nodes will only vote for chain in epoch e. Thus the finalization rule for chain must be triggered after epoch e starts, but by this time all honest nodes have observed a proposal for $\mathsf{chain}'\langle : \widetilde{e}\rangle$. Therefore it must be that $\mathsf{chain}'\langle : \widetilde{e}\rangle = \mathsf{chain}\langle : \widetilde{e}\rangle$ since otherwise the finalization rule cannot trigger on chain due to seeing a conflicting proposal for $\widetilde{e}$. □

3.3 Liveness Proof

We can show that honest nodes' finalized chains must grow whenever there are 6 consecutive epochs all with honest proposers. The proof follows almost identically as in Section 2.4, where we can prove that an honest proposer's proposal never gets rejected by honest recipients. The liveness claim therefore follows by observing that an honest proposer does not propose two blocks of the same epoch.

4 Additional Improvements and References

Optimistic responsiveness. The protocols described earlier are preconfigured with an anticipated delay parameter Δ, and a new block can only be confirmed per $\Theta(\Delta)$ rounds (also called an epoch earlier). In practice, if and whenever the actual network delay δ is much smaller than Δ, it would be desirable to confirm transactions as fast as the network makes progress, i.e., the confirmation time should be dependent only on the actual delay δ and not on the a-priori upper bound Δ. Protocols that achieve this property are said to be *optimistically responsive* [14].

In Pala [4] and Pili [5], the authors show that with very minor tweaks to protocols described in this tutorial, one can achieve optimistic responsiveness in the partial synchronous and synchronous settings respectively. Later versions of the Hotstuff [1] paper and subsequently Sync Hotstuff [2] also achieved optimistic responsiveness.

Synchronous and yet partition tolerant. The synchronous, honest-majority protocol described in Section 3 makes a strong network synchrony assumption for its consistency proof. Specifically, every honest node's messages must be delivered within Δ delay. In other words, if an honest node ever temporarily drops offline and violates the Δ bound, it is treated as corrupt by the model and the consensus protocol is no longer required to provide consistency and liveness guarantees for this node. In practice, typically no one can deliver 100% uptime — since blockchains are long running, every node may become offline at some point, and thus at the end time, the classical synchronous model will treat everyone as corrupt! This means that protocols proven secure in the classical synchronous model do not necessarily offer strong enough robustness for practical deployment. A symptom of this is that almost all known *synchronous* consensus protocols appear *under-specified* and *unimplementable*: typically these protocols do not fully specify what a node should do if it receives messages out of sync, e.g., after coming back online after a short outage (and it is dangerous to leave this decision entirely to an ordinary implementer).

Recently, Guo, Pass, and Shi [7] propose a new model that allows one to capture a notion of "best-possible partition tolerance" while making mild network timing assumptions. Specifically, in their model, a secure consensus protocol must provide both consistency and liveness to all honest nodes, even those who might have suffered from temporary outages but have come back online, as long as at any point of time, there exists a set of honest and online nodes that are majority in size. Moreover, this honest and online set may even churn rapidly over time. Given Guo et al.'s model, a recent work called Pili showed how to achieve this notion of best-possible partition tolerance through very minor tweaks to the protocol described in Section 3 (and at the same time offering optimistic responsiveness too).

Reference implementation. We refer the reader to an open-source implementation of Pala (`https://github.com/thundercore/pala`). This implementation adopted a doubly-streamlined, and optimistically responsive variant of the protocol described in Section 2. We briefly explain the "doubly-streamlined" idea: in the protocol in Section 2, a node must have received the parent chain's notarization to vote on the next block — this can lead to pipeline stalls in settings with long delay and large bandwidth. Double streamlining is a generalization of the protocol in Section 2 such that nodes can propose and vote on a block as long as its k-th ancestor's notarization has been received; however, for finalization, one has to chop off roughly k blocks too.

Acknowledgments

We gratefully acknowledge Kartik Nayak, Ling Ren, Robbert van Renesse, and Steven Galbraith for helpful feedback on improving the writeup. The author would like to thank T-H. Hubert Chan and Rafael Pass for inspiring discussions.

References

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

[2] Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Maofan Yin. Sync hotstuff: Simple and practical synchronous state machine replication. Cryptology ePrint Archive, Report 2019/270, 2019. `https://eprint.iacr.org/2019/270`.

[3] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *OSDI*, 1999.

[4] T-H. Hubert Chan, Rafael Pass, and Elaine Shi. Pala: A simple partially synchronous blockchain. Manuscript, 2018. https://eprint.iacr.org/2018/981.

[5] T-H. Hubert Chan, Rafael Pass, and Elaine Shi. Pili: A simple, fast, and robust family of blockchain protocols. Cryptology ePrint Archive, Report 2018/980, 2018. https://eprint.iacr.org/2018/980.

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

[7] Yue Guo, Rafael Pass, and Elaine Shi. Synchronous, with a chance of partition tolerance. Cryptology ePrint Archive, 2018.

[8] Timo Hanke, Mahnush Movahedi, and Dominic Williams. Dfinity technology overview series: Consensus system. https://dfinity.org/tech.

[9] Leslie Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, May 1998.

[10] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, July 1982.

[11] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[12] Rafael Pass and Elaine Shi. Hybrid consensus: Efficient consensus in the permissionless model. In *DISC*, 2017.

[13] Rafael Pass and Elaine Shi. Rethinking large-scale consensus. In *CSF*, 2017.

[14] Rafael Pass and Elaine Shi. Thunderella: Blockchains with optimistic instant confirmation. In *Eurocrypt*, 2018.

[15] Fred B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Comput. Surv.*, 22(4):299–319, December 1990.

[16] Virgil Griffith Vitalik Buterin. Casper the friendly finality gadget. https://arxiv.org/abs/1710.09437.

A Notations

Variable	Meaning
chain	blockchain
$\mathsf{chain}[:\ell]$	prefix of chain upto and including the ℓ-th block
$\mathsf{chain}[:-\ell]$	prefix of chain after removing the trailing ℓ blocks
Δ	maximum network delay between honest nodes (during a period of synchrony)
e	epoch number, i.e., a collection of 2Δ rounds
n	number of nodes
h_{-1}	parent hash encoded in a block
TXs	application-specific payload in a block, e.g., a set of transactions to confirm
H	collision-resistant hash function for hashing blockchains
H^*	a random oracle for proposer election