The Distributed Computing Column

BY

Stefan Schmid

Aalborg University
Selma Lagerlöfs Vej 300, DK-9220 Aalborg, Denmark

This issue of the distributed computing column includes two articles:

1. Ittai Abraham and Dahlia Malkhi present an interesting perspective on Blockchain consensus protocols, through the lens of distributed computing, and discuss the relationship to Byzantine Fault Tolerant (BFT) protocols.

2. Carlos Baquero, Paulo Sérgio Almeida, Alcino Cunha, and Carla Ferreira present a survey of the mathematical structures and compositional properties of state-based replicated data types that support the implementation of eventually consistent, geo-replicated data management solutions.

Enjoy!

The Blockchain Consensus Layer and BFT

Ittai Abraham Dahlia Malkhi
Vmware Research Group
VMware
{iabraham, dmalkhi}@vmware.com

Abstract

In this paper, we analyze Blockchain consensus protocols in the lens of the foundations of distributed computing. Our goal is to present analogies and connections between Blockchain protocols and Byzantine fault tolerant (BFT) protocols. We also discuss opportunities to consider hybrid solutions.

Keywords: Blockchain, Byzantine Agreement, BFT

1 Introduction

In the early 2000's, a group of activists advocating the wide-spread use of cryptography and privacy-enhancing technologies were engaging over the *cypherpunks* mailing-list in an effort to create an anonymous, monitor-free digital cash. Step by step, they jointly built the ingredients that eventually lead to the emergence of Bitcoin in 2009. In recent years more crypto-currency variants have emerged. As of late 2017, the market cap associated with Bitcoin is over $70 billion and the total crypto-currency market cap is about twice that.

The steps leading to the construction of Bitcoin harness deep ideas and methods from published academic works. Its incredible market cap reflects the public trust in the robustness and soundness of the technology, without any company or institution backing it.

At the core of Bitcoin is a method for reaching agreement on a shared chain of blocks where each block contains a sequence of transactions. This core is called the Blockchain. In many ways the Blockchain is the most intriguing and innovative aspect of Bitcoin. In this paper, we study Blockchain through the lens of the theory of distributed computing.

To put this exploration in context, let's take a quick perspective on the full Bitcoin approach, and explain what challenge Blockchain addresses. Bitcoin relies heavily on the idea of *computational puzzles* as proof-of-work (PoW). The idea of using computational puzzles first appeared in the pioneering work of Dwork and Naor in [17]. Similar approaches were taken to fight email spam by forcing senders to work by Hashcash [8]. Aspnes et al. [7] suggested to use computational puzzles for preventing Sybil attacks in a Byzantine setting.

The cypherpunks were interested in a much more ambitious use-case of crypto-puzzles such as Hashcash, the use of computation power as means for minting crypto-currency. They recognized that PoW provided scarcity and uniqueness, two necessary ingredients for creating value. However, the main challenge was to provide users with a decentralized, anonymity-preserving solution to protect against double-spending. At first, this seems like a daunting problem: How can you track spendings and preserve privacy at the same time?

A couple of ideas that eventually led to the Blockchain solution were posted to the cypherpunks mailing list. Dai proposed b-money [12], a crypto-currency system that already uses crypto-puzzles for minting digital currency, in which participants themselves track all digital account balances. This was not a very practical approach because it relied on a timely multicast channel in a peer-to-peer setting, and rather than preventing double spending, involved rather cumbersome remediation mechanisms. Szabo enhanced this idea with the Bit Gold crypto-currency [38]. The Bit Gold system uses Haber and Stornetta's hash-chaining [22] to create a secure ordering of transfers, a transaction ledger. The position of a transaction in the ledger determines the latest owner of a digital asset. The question was who is in charge of maintaining the ledger. Bit Gold used Byzantine quorums [31] to maintain the ledger replicated among all users. Since there was no way to validate membership in Bit Gold, quorums were not enough to provide consistency guarantees.

The challenge of preventing double-spending in a permissionless setting thus remained unsolved until the introduction of the Bitcoin Blockchain protocol. The last piece of a full crypto-currency solution was finally addressed.

Blockchain uses PoW as a way to obtain several goals at once. It is a way to mint currency but more importantly it is a key ingredient in the Bitcoin Blockchain protocol that reaches agreement and prevents double spending. Newly minted blocks are spread among the miners over a peer-to-peer network, and each miner keeps the longest chain as the winning chain, even if it means overturning earlier segments of the chain. Since winning the longest chain means solving cryptographic puzzles in each block, overturning a long tail segment is hard. For this reason, blocks "buried" deep in a miner's chain are typically considered committed with high probability.

1.1 A Layered view

The Blochain technology is more than a consensus engine. Zooming out a bit, we advocate a *layered view*, which decomposes Blockchain into different components that each deals with separate concerns.

Layer-1: State-machine replication (SMR). The bottom layer is focused on an infrastructure for storing an immutable sequence of transaction-blocks among distrustful parties.

From a foundational standpoint, this layer builds a chain of *consensus* blocks, a problem that has received tremendous attention in the distributed systems arena. As evidenced from the above brief history, Bitcoin incorporates several deep ideas that have been around for quite a while, some more than two decades. Yet the consensus engine, which achieves agreement among distrusting parties in a scalable settings with unknown participants, seems very different than the classical methods for Byzantine fault tolerance (BFT).

This writeup is dedicated to an exploration of the analogous replication problem model with Byzantine fault tolerance in the distributed computing literature and relating BFT to Blockchain.

Layer-2: Smart contracts. The middle layer takes a simple shared data structure and exposes rich, high level, business relevant abstractions. This allows users and applications to use advanced cryptography tools to generate *smart contracts* [37]. Contracts allow to programatically facilitate, verify, or enforce the negotiation or performance of a business transactions.

They are embedded as Blockchain SMR commands, hence all participants in share the responsibility of executing them, their logic is immutable, and their audit-trace is open. A future writeup will overview the foundations of this layer.

Layer-3: Services. Finally, the top layer, *applications*, is where the customer value is created. There is great excitement about the possibilities that shared provenance of asset and identity tracking brings. Not surprisingly, many of them are in the financial arena, but not only, and it is yet to be seen what are the "killer-apps" for Blockchains.

1.2 Roadmap

In this paper, we analyze Blockchain consensus protocols in the lens of the foundations of distributed computing. In §2, we present the algorithmic foundation of Nakamoto Consensus (NC), explaining how it solves (with high probability) the state-machine replication (SMR) problem. In §3, we attempt to relate NC to the classical literature on Byzantine fault tolerant SMR (BFT). We continue in §4 to overview several approaches for bringing the two paradigms together. We conclude in §5.

2 Nakamoto Consensus through the lens of the theory of distributed computing

Distributed protocols often have some desired properties they wish to obtain. For example, Byzantine agreement protocols aim to solve the "consensus" problem and it is often said that Blockchains solve the "double spending" problem. It is crucial to understand under what conditions do these protocols obtain their desired properties. In distributed computing theory we typically start by defining an adversary model and then prove that the desired properties hold in any execution against such an adversary.

2.1 Adversary Model

In the traditional distributed computing literature, the most common adversary assumption is the *Threshold Adversary Model*. In this model there is typically an assumption of a threshold gap between two parameters:

1. The total number of parties, often denoted by n.

2. The total number of parties the adversary can control, often denoted by f.

The typical *threshold* chosen is either of a minority $n > 2f$ or a third $n > 3f$. This model is sometimes also called the "permissioned model" to indicate that it is not the case that anyone can join the group of n parties, but rather one needs to have a "permission to join". Indeed in this model we tyically assume there is a fixed set n participants.

One of the major model modifications that Bitcoin considers is the alternative *Computational Threshold Adversary (CTA)* model. In this model, instead of bounding the total number of parties the adversary controls relative to the total number of parties, the model bounds the

total amount of computational power the adversary has relative to the total available computational power of all parties. To push the analogy we can formally define:

1. The total amount of computational power, denoted by N_C.

2. The total amount of computational power the adversary can control, denoted by F_C.

The assumption in Bitcoin is that the adversary controls a minority of computational power. We can denote this by a *threshold* of $N_c > 2F_C$. In some cases it helps to assume the adversary controls some ϵ fraction less than a majority, $N_C > 2(1 + \epsilon)F_C$.

We note that the formal definition of "computational power" needs a concrete materialization. For example, in Bitcoin, this assumption builds on mechanisms that harnesses the difficulty of inverting SHA1.

This model is often also called the "permissionless model" to indicate that there is no explicit membership protocol, anyone who can solve cryptographic puzzles can join the system.

A new model modification is being debated and considered lately. The idea is to bound the adversary by allowing him a minority *stake* in some abstract finite resource. Indeed, instead of bounding the relative number of parties or the relative computational power of the adversary, one can imagine bounding some other limited resource. In a crypto-currency use case it is very natural to use the crypto-currency itself as the limited resource.

This leads to the *Stake Threshold Adversary (STA)* model. In this model, there is some finite abstract resource we will call R. Again we can define:

1. The total amount of resource R, denoted by N_R.

2. The total amount of resource R that the adversary can control, denoted by F_R.

For example, the underlining assumption in Ethereum's Proof-of-Stake approach can be modeled as an adversary that controls a third of the resource R (in their case of the Ethereum crypto-currency). We can simply denote this as a *threshold* assumption $N_R > 3F_R$.

In many ways this third model is a generalization of the previous two. The classic threshold model is just a one-party one-vote resource and the computational threshold model just using computation as the limited resource.

We note that in all three models there is a possibility to dynamically modify the parameters as long as the threshold remains the same. These dynamic changes often bring more challenges for protocol solutions.

An additional power of the Proof-of-Stake approach is that it allows for punishing parties (by "slashing" their stake) if they can be publicly detected as malicious. We briefly discuss this next.

2.2 Game theoretic model

Instead of designing protocols against a malicious adversary, an alternative approach is to design protocols that form a threshold coalition resilient equilibrium [3, 4]. In both models the potential deviating parties are limited by a threshold. The main difference is that in the malicious model the deviating parties can act completely arbitrarily while in the game theoretic model we often assume the deviating coalitions acts rationally while attempting to maximize their utility.

In the crypto-currency scenario it is often natural to define the utility simply as a function of the crypto-currency itself. The main mechanism used is the fear of punishment. This leads to the idea of each member putting some deposit (stake) in such a way that if a deviating member is publicly verified as malicious then its deposit can be deleted or reduced. This approach appears in Wei Dai's b-money [12] and in recent Casper suggestions [9].

In the rest of this section, we analyze Bitcoin in the Computational Threshold Adversary model, and leave the game theoretic model for a future survey.

2.3 The Computational Threshold Adversary model

We now discuss the Computational Threshold Adversary model in more detail. A key element of the CTA model is the notion of Proof-of-work (PoW): The adversary cannot prove more work than its threshold share of computation power.

Synchrony. We note that bounding the computational power is often meaningless as PoW in a fully asynchronous system. If network delay can be unbounded then the adversary can boost its computational power just by making the non-faulty parties incur much higher delays. It is therefore commonly accepted that the CTA model incorporates in it some degree of synchrony assumptions [20, 13, 35, 6]. A more detailed analysis of network delays appears in [34]. This is in contrast to the traditional threshold adversary model where both asynchrony and synchrony models can be considered.

PCNELE. Bitcoin uses PoW to implement a leader election "oracle" with several interesting properties:

- (Independence) Each party is elected independently (so multiple parties may be elected in the same round).
- (Fairness) The probability of electing each party is proportional to its relative computational power.
- (Pre-Commit Non-Equivocation Leadership announcement) In each round, each party commits to an action and the oracle probabilistically elects parties and announces them and their action.

We name an oracle with these properties a Fair, Pre-Commit, Non-Equivocation, Leader Elections (PCNELE) Oracle. The first PCNELE property, Independence, implies that in some rounds it may elect multiple leaders. This is typically tolerated in distributed protocols, as long as sufficiently often just one leader is elected.

The second one, Fairness, is important for many distributed protocols, e.g., for load balancing or contention resolution. It is crucial for Bitcoin's setting because winning an Oracle election has economical value.

The last, Pre-Commit Non-Equivocation, captures two unique functionalities. In a Byzantine fault model, just electing a leader is not enough because we want the leader to be able to add just a *single* block (and not be able to send different blocks to different parties). Moreover we would like the potential leader to *commit* to its single block content *before* the election

winner(s) are announced. In particular this means the leader cannot equivocate by sending two different messages to different parties.

Note that PCNELE is much stronger than the classic Ω Oracle that is sufficient for consensus in an asynchronous environment. This oracle is a key building block of the Nakamoto Consensus protocol. Getting the full protocol requires just one more idea: each new block will contain a reference to a previous block and the protocol advices to connect the new block so it forms the longest chain. Before we get there, we discuss a concrete PCNELE implementation via crypto-puzzles.

Implementing PCNELE using crypto-puzzles. In the CTA model, we can implement the FPCNELE oracle using cryptographic puzzles. Such a puzzle needs to have three properties:

- (Pre-commit) Solving the puzzle requires to commit to a specific *block*.
- (Public verifiability) The solver of the puzzle can generate a *solution certificate* of his correct solution and his committed block. This certificate can be efficiently publicly verified.
- (Fairness) the probability of solving the puzzle (generating a solution certificate) at any given round is proportional to the computational power of the party.

Note that this definition inherently assumes a round based model (or a partially synchronous model).

We now detail a concrete protocol based on a very simple cryptographic puzzle (a simplification of the Bitcoin cryptographic puzzle). We use two parameters: P which is the base of the puzzle and H which is the hardness parameter. For party with a public key i and a desired action o to solve the puzzle it must find a number *nonce* such that

$$SHA1(P||i||o||nonce) < H$$

Here we use SHA1 as an example of a secure hash function. Pre-commit is obtained because the action o is embedded into the puzzle and it is computationally hard to find a collision with another action o'. Public verifiability is obtained, because once the party discovers an adequate nonce, then it can publish it and anyone can verify solution. Fairness is obtained under the assumption that the cryptographic hash function is essentially a random function. So the only solution strategy is use brute force to find the nonce.

The parameter H needs to be tuned relative to the total amount of computational power. A large value of H will cause many parties to solve the puzzle in the same round, causing contention. A small value of H may cause the expected time it takes to elect a leader to be too long. Note that the election process essentially induces a Poisson process on elected parties whose parameter depends on H.

2.4 The Nakamoto Consensus protocol - Longest Fork Wins

At its core, Nakamoto Consensus (NC) is a protocol that implements a replication of a Blockchain by using two elegant and powerful ideas: PoW and a longest fork win (LFW) strategy.

The LFW strategy works as follows. When pre-committing a block content, a party must also pre-commit on a reference to a previous block. The LFW rule is to always choose to reference the longest chain.

We begin by providing an abstract solution that assumes access to an abstract FPCNELE Oracle of the previous section. We then define what problem this solution solves. Finally we detail a concrete example that follows the Bitcoin protocol.

An abstract NC protocol: Implementing Blockchain replication We now focus on implementing Blockchain replication assuming access to a FPCNELE Oracle.

The abstract NC protocol has just two principles:

- (New Block via FPCNELE Oracle): the Oracles allows each elected leader to announce a single pre-committed new block.
- (Longest Fork Wins): non-faulty leaders will connect their pre-committed block to the leaf block which forms the longest path from the genesis block.

Concretely, in each round, parties pre-commit to the FPCNELE Oracle which block they want to add and to which existing block they want to add it to. The Oracle then announces the set of parties that are elected (possibly none, possibly more than one) and the blocks of the elected parties are announced.

Now that we defined the abstract NC protocol, let's formally define what problem it solves.

Blockchain replication The core goal of state-machine replication (SMR) is to form a growing log of commands. A new command is appended to the tail of the log by a consensus decision, and does not modify anything before the tail. We will return to the classical SMR problem in the next section. Here, we view an even more abstract graph theoretic model.

We can view a log of commands as a dynamically growing *directed path*. A new command is added by adding a new node that points to the previous last node of the path. To define a *Blockchain replication protocol* we need to deal with potential forks: instances where there is (some) uncertainty about which command it decides on. For example, there may be several competing blocks that all point to the same parent block. To model this we need to extend the dynamically growing directed path abstraction to a more general *dynamic direct acyclic graph* model. We will call this graph G and call the root node g of G the "genesis" node. Given any existing leaf y we can extend y with a new node x by announcing the edge $y \leftarrow x$. We call this operation "announcing a new block".

We note that while G can be an arbitrary DAG, the goal of a Blockchain protocol is to force G to be as similar as possible to a directed path.

Blockchain Replication Properties. The goal of the Nakamoto Consensus protocol is to implement a *Blockchain replication protocol* in the computational threshold adversary model.

To this end we define four desired properties:

- (Uniqueness) there is a unique deterministic function $L(G)$ to extract a single path $P = g, a_1, a_2, a_3, \ldots, a_k$ from G. Typically $L(G)$ is the longest path in G from the genesis (with some tie-breaker).

- (Liveness) the path $L(G)$ is constantly growing (this often depends on the synchrony assumptions, for example: that a new block is added in expectation every 10 minutes).
- (Safety) If $P = g, a_1, a_2, a_3, \ldots, a_k = L(G)$ is a path of length k for G and G grows to become G' then for any $a_i \in P$ the probability that $a_i \notin L(G')$ is exponentially decreasing proportional to $k - i$. This is the famous "burying" property: the deeper a block is buried in the path P the harder it is to revoke it. In bitcoin, a block is typically considered "committed" if its buried behind a chain if 6 newer blocks (= one hour's worth of computational puzzle solving).
- (Fairness) the proportion of nodes in P that belong to non-faulty parties is at least proportional to their relative computational power.

We note that uniqueness is typically obtained by defining $L(G)$ as the longest path in G. If there are several paths of maximal length then we can deterministically choose one of them. Using a randomized tie breaker is also an option and has certain advantages (improved fairness and resilience to selfish mining) and disadvantages (potentially slower conversion).

The liveness and fairness properties are natural: we want the unique chain to grow and we want the proportion of blocks to represent the proportion of computational power of the parties.

The safety property is somewhat subtle. In order to make sure that a block is committed and will not be removed with high probability one needs to wait until the block is "buried" deep enough in the path P.

As we will later discuss, a solution for Blockchain replication can be trivially used to solve state-machine replication: simply define the state-machine as the operations in $P = L(G)$ after removing the k most recent nodes in P. Note that this implementation provides safety only with high probability.

Analyzing abstract NC If non-faulty parties follow the LFW principle, then it can be shown that indeed the properties of Blockchain replication are held.

While Liveness and Fairness seem to follow almost directly from the FPCNELE Oracle properties, again Safety is more subtle. Roughly speaking, the reason that a block a_i that is "buried" k blocks deep will not be replaced is that the probability that the adversary manages to fork the chain from a_{i-1} and create an alternate longer chain of length $k + 1$ is exponentially small as a function of k. The exponent of this property is a function of how far the adversary is from controlling half of the computational resources.

After defining an abstract NC protocol (that uses an Oracle) and defining what problem it solves we now describe a very simply concrete NC protocol.

2.5 Nakamoto Consensus - A concrete protocol

A concrete Nakamoto consensus protocol can be written in a single line:

- Given a party i that wants to commit a block with transactions o, let $P = a_1, \ldots, a_k$ be the longest path in G then party i attempts to solve the puzzle that will allow it to announce the $a_k \leftarrow b$ where b is a new block that contains i public key, and o.

  Concretely, find the nonce such that

$$SHA1(a_k||i||o||nonce) < H$$

The simple, single line protocol is a simplified version that captures the central idea behind the Bitcoin's Blockchain consensus protocol. At its core it uses Proof-of-Work to implement a Fair Leader Election Oracle that provides non-equivocation and pre-commitment. Non-faulty leaders are expected to always use the Longest Fork Wins rule and extend the longest path. This in turn implies that the non-faulty parties manage to implement a Blockchain replication protocol. Which in turn can be used to implement a state-machine replication protocol (where safety is guaranteed with high probability). A rigorous and detailed analysis of this approach appears in [20, 34].

3 Relating NC to BFT

When an NC leader announces a block $a_k \leftarrow b$, it implicitly announces not just the new block b, but also any block in $\{a_1, ..., a_k\}$ missing from G. By the Longest Fork Wins principle, this leader "proposal" wins if the path $a_1, ..., a_k, b$ is $L(G)$, i.e., the longest path in G. As mentioned above, this is the NC solution to the classical state-machine replication (SMR) problem of reaching agreement on a *growing log of commands*.

Relating this to the classical SMR literature, there is a common theme where proposals are prioritized by leader *ranks*: A proposal by a *highest ranking leader* is accepted and forces the leader's log-prefix. In the BFT literature, replication consistency is maintained by two principles:

- (Non-equivocation) leaders are prevented from *equivocating*, so that there is only one possible proposal per leader per rank
- (Proposal-safety) a (higher-ranking) proposal may extend, but not modify, any lower-ranking committed log prefix.

NC and BFT differ in how they accomplish these two key principles, and consequently, in BFT having instant finality and NC not having it.

We now put aside NC and discuss the foundations of replicated services. We will get back to relating the two approaches later.

SMR Problem Model. We start with a brief overview of state-machine replication (SMR) and related terminology. The State-Machine-Replication (SMR) approach [28, 36] is well known paradigm for building a fault-tolerant service, that linearizes all updates as if they occur one after another [23]. Driving an SMR service is a sequence of consensus decisions on a *growing log* of state-machine operations. The parties in the protocol are called *replicas*, and each state-machine *replica* executes the log of operations deterministically, arriving at a consistent replicated state. Here we adopt the traditional Threshold Adversary model (see §2). In this model, a threshold of the replica set may suffer certain failures; below, we discuss both benign failures and arbitrary corruptions (Byzantine failures). For simplicity, we neglect clients and the details by which requests arrive at replicas, and simply assume replicas have their own inputs.

Focusing on the consensus core, the key challenge is to repeatedly reach agreement decisions among the replicas on extending the log with a new command:

- (Agreement) There is agreement on a sequence of decisions among all correct replicas.
- (Validity) Committed decisions form a monotonically growing log of commands. Each command has been proposed by a replica. (In particular, in the Byzantine failure case, each command must carry some replica's signature.)
- (Liveness) If a correct replica proposes a command, then eventually some decision is committed.

3.1 Asynchronous benign framework with $N = 2F + 1$

We start with the benign-failure model, which allows us to focus on Safety; we get back to dealing with the threat of leader equivocation later, in the Byzantine-failure model.

We present a framework that borrows from the asynchronous consensus solution introduced by Dwork, Lynch and Stockmeyer in [16], and widely employed for SMR, e.g., in Viewstamped Replication [32], Paxos [29], Raft [33], and others. The DLS framework has the desirable property that Agreement is kept under all scenarios, including asynchrony; liveness depends on successfully electing a correct leader with timely communication channels to replicas.

A key concept of the framework is an explicit ranking among proposals. In DLS, ranks are termed *phases*, in Paxos, they are called *ballots*, and in VR, *views*. Here we use *views*. Replicas all start with an initial view, and progress from one view to the next. Commands are accepted in the highest view to have started. In some views, a decision will be reached to extend the log-prefix by one slot; other views will expire without a decision. Liveness relies on having a constant fraction of the views with a good and timely leader.

Decision values are monotonically increasing: Once a certain prefix becomes committed in a view, higher views can only extend it, but no slot in the committed log-prefix may ever be reverted. Therefore, once a certain prefix is committed, it becomes ready for execution by the replicas.

Algorithm 1 provides a breakdown of the framework into five abstract components: VIEW, a scheme for prioritizing proposals; WEDGE, a mechanism for starting new views; SAFE, a mechanism for collecting information about lower views and picking a safe value to propose based on responses; ACCEPT, a mechanism for making an accepted leader proposal durable; and DECIDE, a predicate for committing a decision.

Briefly, in each view there is a single designated *leader* (in fact, we typically identify the leader with the view-number). The other replicas are called *acceptors*, they accept at most one proposed value in the view. A replica moves to a higher view if a local timer expires.

A leader in a view ensures Validity by collecting information from a quorum that intersects every commit quorum in lower views in at least one correct acceptor. The leader then picks a safe log-prefix to propose, and extends it by one slot. Specifically, in a benign settings, the algorithm works for $N = 2F + 1$ replicas and quorums of size $N - F$. The leader collects STATUS messages from a quorum of $N - F$ acceptors, thus intersecting the quorums of all previous views. Among all the accepted prefixes reported to it, the leader picks the one accepted at the highest view, if any, or an empty prefix if none. It extends the log-prefix with its own input command, and proposes it to replicas.

Pipelining. Practical SMR solutions, such as VR/Paxos/Raft, let the leader of the current view drive repeated extensions to the growing log. This enables an optimized pipeline of proposals: A stable leader avoids waiting for the current view to expire for each command, and does not need to collect status messages. In this tutorial, for pedagogical reasons we will ignore this optimization (for a discussion of this optimization, see [11]).

Correctness In a nutshell, this protocol guarantees Agreement because a leader proposes only safe proposals: A value is safe to propose in a view if no lower view can ever decide a conflicting value.

As for liveness, the protocol makes a decision as soon as a good view is activated, meaning that the view-leader has timely communication with a majority quorum, none of which times out and moves to a higher view.

Algorithm 1 Benign SMR solution skeleton with $N = 2F + 1$.

VIEW Initially, each replica starts in view 0. A replica starts view $v + 1$ if view v expires with no progress, or if it receives any message from a view higher than its current view. The leader for view v is the replica whose ID modulo N equals the view number. We denote it by $L(v)$.[1]

WEDGE An acceptor Q, upon starting view v, sends $L(v)$ a message (STATUS, v, Q, H), where H is the proposal and view number in a maximal view for which Q ever received a proposal, or $\perp$ if none received. After moving to view v, an acceptor rejects messages from views lower than v.

SAFE $L(v)$ waits for $N - F$ status responses (STATUS, v, Q, H_Q). It picks a safe value S to propose. S is the status value whose view is highest among the STATUS responses collected, if any, or an empty log.

ACCEPT $L(v)$ extends S with its own input value, and sends the extended prefix $S*$ to all replicas a message (PROPOSE, v, $S*$).

An acceptor Q in view v, upon receiving a (PROPOSE, v, $S*$) message from $L(v)$, records v, $S*$ as the highest proposal it accepted, and sends (ACK, v, Q) to the leader.

DECIDE When $N-F$ acceptors of any view v accepted a proposal S it becomes the committed decision. The leader, or anyone learning this decision, broadcasts a (DECIDE, v, $S*$) notification.

3.2 Asynchronous Byzantine framework with $N = 5F + 1$

The above solution works in a model where the adversary can only cause crash failures. For the Byzantine model where the adversary can perform arbitrary actions on the faulty parties we need to make a few adjustments. First, all messages are signed, to prevent spoofing and to enable forwarding messages on behalf of others as proofs of safety and validity in various steps.

Second, the quorums of the SAFE mechanism, which are used for collecting information about lower views and picking a safe value to propose based on responses, need to be adjusted to guarantee intersection in sufficiently many non-faulty replicas [31]. Third, the ACCEPT component requires a mechanism that prevents a leader from equivocating and sending conflicting proposals to different parties.

The first solution we illustrate addresses these challenges simply by increasing quorum intersection to $3F + 1$. Consequently, this solution requires $N = 5F + 1$, and does not have optimal resilience. Nevertheless, it is a simple adaptation of Algorithm 1, and has the benefit of incurring linear communication complexity. Similar $5F + 1$ schemes appeared in [27, 1]. Algorithm 2 below highlights the modified parts.

Algorithm 2 Byzantine SMR solution skeleton with $N = 5F + 1$.

VIEW Initially, each replica starts in view 0. A replica starts view $v + 1$ if view v expires with no progress, or if it receives $F + 1$ messages (directly or indirectly) from a view higher than its current view. The leader for view v is the replica whose ID modulo N equals the view number. We denote it by $L(v)$.

WEDGE An acceptor Q, upon starting view v, sends $L(v)$ a message (STATUS, v, Q, H), where H is the proposal and view number in a maximal view for which Q ever received a proposal, or $\perp$ if none received. After moving to view v, an acceptor rejects messages from views lower than v.

SAFE $L(v)$ waits for $N - F$ status responses (STATUS, v, Q, H_Q). It picks a safe value S to propose. S is the proposal whose view is highest such that $2F + 1$ replicas accepted it in the view, if any, or $\perp$ if none received.

ACCEPT $L(v)$ extends S with its own input value, and sends the extended prefix $S*$ to all replicas a message (PROPOSE, v, $S*$). It attaches to its proposal a proof of safety, e.g., the collection of STATUS messages.

An acceptor Q in view v, upon receiving a (PROPOSE, v, $S*$) message from $L(v)$ for the first time, verifies the proposal validity and then records v, $S*$ as the highest proposal it accepted, and sends (ACK, v, Q) to the leader.

DECIDE When $N-F$ acceptors of any view v accepted a proposal S it becomes the committed decision. The leader, or anyone learning this decision, broadcasts a (DECIDE, v, S) notification.

A brief note about correctness of this Byzantine protocol. It guarantees agreement because each two quorums intersect in $3F + 1$ replicas, and $2F + 1$ of them are correct. Therefore, if a

decision is made in a view, a higher view intersects it in $2F + 1$ responses. Furthermore, in a system of $N = 5F + 1$, no other value can appear $3F + 1$ times. Similar to the benign case, the protocol is live if a correct leader has timely links with a quorum.

3.3 Synchronous Byzantine framework with $N = 3F + 1$

We can adapt Algorithm 2 to a synchronous settings, such that the resulting synchronous algorithm requires only $N = 3F + 1$ replicas, instead of $N = 5F + 1$. We do not repeat the algorithm, and instead we just highlight the adaptations.

In a synchronous setting, a party can collect messages from all non-faulty replicas by waiting the appropriate number of maximal transmission durations.

The leader sends a signed PROPOSE. A replica accepts the first PROPOSE message it receives and "echoes" it–adding its own signature–to all other replicas in a PROPOSE message.

After it accepts a proposal, a replica waits for echoes from all other replicas. A replica DECIDES if it hears $N - F$ PROPOSE messages with the same value and does not hear any PROPOSE message with a different value.

For a decision to be safe, the SAFE component requires a new leader to choose the value whose view is highest in which $F + 1$ replicas accepted an identical value, if any, or $\perp$ if none received.

Briefly, this algorithm maintains Agreement because a decision is reached in a round in which all correct replicas accept some proposal and no correct replica accepts a different proposal. Because correct replicas always respond in time, a leader of a higher round picking among STATUS messages the highest view in which $F + 1$ replicas accepted an identical value is guaranteed to choose this value as the only safe possibility.

3.4 Byzantine frameworks with optimal resilience

The two Byzantine protocols above do not have optimal resilience. To obtain a protocol for $N = 3F + 1$ for the asynchronous model, and likewise, $N = 2F + 1$ for the synchronous model, the trick is to replace the leader's broadcast with a protocol that provides two guarantees. One, the leader cannot equivocate. And two, every non-faulty replica that echoes a leader proposal can prove that it is unique. Obtaining these two properties typically requires replacing the trivial leader broadcast with a two-round protocol [14].

Asynchronous Byzantine framework with $N = 3F + 1$: In the asynchronous model, the leader protocol works in two-phases as follows. In the first phase, the leader sends a signed proposal tagged PRE-PROPOSE. Replicas echo–adding their own signature–the first leader's PRE-PROPOSE message they receive either directly from the leader, or in a PRE-PROPOSE message they receive from other replicas. In the second phase, upon receiving the same value in either $2F + 1$ PRE-PROPOSE messages, or $F + 1$ PROPOSE messages, replicas echo with a signed PROPOSE message and accept the proposal.

Upon accepting a PROPOSE value from the 2-phase broadcast protocol, replicas proceed to process the message as a leader proposal. Specifically, a decision is reached in a view (as usual) upon receiving $N - F$ PROPOSE messages for a value.

For a decision to be safe, the SAFE component requires a new leader to choose the value whose view is highest, such that either (i) $F + 1$ replicas sent a PROPOSE message with the

value, if any, or (ii) $2F + 1$ replicas sent a PRE-PROPOSE for it, if any, or (iii) $F + 1$ replicas sent a PRE-PROPOSE for it, if any. It chooses $\perp$ if non such value is received.

For further details of folding this leader broadcast protocol into the SMR full protocol, we refer to the PBFT protocol of Castro and Liskov [10]. Note that, the communication complexity here is quadratic, and matches the communication lower bound of Dolev, Reischuk and Strong [15].

Synchronous Byzantine framework with $N = 2F + 1$: In synchronous settings, we can implement the leader's broadcast protocol using two all-to-all exchanges as follows.

In the first phase, the leader broadcasts a PRE-PROPOSE value. Replicas echo to all other replicas–adding their own signature–the first leader PRE-PROPOSE message they receive.

In the second phase, a replica accepts a value if all PRE-PROPOSE echoes (of which there are at least $F + 1$) match. Note that, due to synchrony, if two correct replicas echo conflicting PRE-PROPOSE values, no correct replica will accept any value.

Upon accepting a value, a replica broadcasts to all other replicas a PROPOSE messages that carries the value it accepted, along with a proof of its validity, e.g., $F+1$ signed PRE-PROPOSE echoes.

It should be obvious from the above discussion that if any value is proposed by a correct replica, then it is unique. Furthermore, in that case, there may be no valid PROPOSE with a conflicting value. Therefore, a decision can be reached in a view upon receiving $F + 1$ PROPOSE messages carrying an identical value.

For this decision to be safe, upon starting a new view a replica broadcasts a PRE-STATUS message containing the last value it accepted, if any, along with a proof of its safety. It waits for all other PRE-STATUS messages and keeps among the valid ones the one whose view is highest. The SAFE component requires a new leader to choose the value whose view is highest which a *single* replica proposed, if any, or $\perp$ if either conflicting ones received or none received. The leader attaches to a PRE-PROPOSE messages a proof of safety, e.g., $F + 1$ STATUS messages.

Briefly, this algorithm maintains Agreement because if any correct replica sends PROPOSE, no conflicting valid PROPOSE exists. The leader cannot hide the proposed value because all correct replicas will receive a PRE-STATUS message about it.

This synchronous $N = 2F + 1$ paradigm appears in [2]. A slightly different leader-based solution paradigm called XFT, which guarantees safety against Byzantine failures in synchronous settings, is given by Liu et al. in [30]. A framework for several of these trade-offs is provided in [5].

3.5 Back to Nakamoto Consensus

We now get back to the world of *parties* that solve computation puzzles instead of replicas, and a Computational Threshold Adversary instead of a Threshold Adversary.

At first sight, Nakamoto Consensus (NC) seems very different than traditional SMR protocols. Instead of an explicit safe value selection scheme it just uses Longest Fork Wins to prioritize the currently longest "proposed" chain. Instead of using explicit quorum to guarantee that decision persists, NC uses synchrony assumptions and cryptographic puzzles to progressively make decisions harder to revoke.

Nevertheless in this section we attempt to view NC in the SMR framework we have just detailed. This exercise highlights many of the similarities. Algorithm 3 illustrates how NC (almost) implements the necessary ingredients we characterized above.

In NC, the notion of a view number (and implicit proposal rank) translates into chain-length; the longer the chain, the higher the rank (with ties broken by some deterministic rule). Like BFT, initially, each party starts in view 0 and moves from one view to the next. A leader of view v is a party solving a puzzle for a chain of length $v-1$, which it announces in a PROPOSE message sent to everyone. The Computational Threshold adversary captures the guarantee of having a constant fraction of the views with a good and timely leader. Indeed, in NC, if an adversary takes control of more than 50% of the computation power in the network, it can theoretically prevent progress: By forking blocks buried at arbitrary level, it can prevent any block from ever becoming finalized.

Similar to BFT solutions, the NC PROPOSE message is self-validating and causes other parties to move to view v when they receive this announcement, and subsequently reject messages from lower views. Different from BFT, there is no explicit collection of STATUS responses, nor explicit proof of safety of the proposal. Nevertheless, NC obtains Agreement (with high probability), and we now explain the relation to STATUS messages and quorums.

The key observation is that a party acceptance is implicit by the collaborative work represented in a puzzle solution. More specifically, recall that we model a Fair PCNELE Oracle model in §2 as choosing only one (or a few) blocks to extend G out of all the parties attempting to. Since all parties wish their proposal to win longest path of G, a block announcement $a_{v-1} \leftarrow b$ that extends the current longest path $a_1, ..., a_{v-1}$ in G to length v represents the collective "acceptance" of this path by many parties presenting proposed blocks to the Oracle. Indeed, if a party tried to introduce a path that replaces the last ϵ blocks of $L(G)$, it would need to win the oracle's choice ϵ times. This occurs with probability that decays exponentially with ϵ. Thus, the PCNELE selection policy incentivizes parties to accept G and extend $L(G)$, rather than replacing any part of it.

In summary, a PoW $a_{v-1} \leftarrow b$, although not a strict proof of acceptance by a quorum of the parties, is a statistical tallying of their acceptances of $a_1, ..., a_{v-1}$. Hence, while Validity is not strictly guaranteed, the probability of replacing a block buried ϵ levels deep is exponentially diminishing with ϵ. As a corollary, Agreement on blocks buried k level deep in $L(G)$ holds with high probability.

Algorithm 3 NC in the view of the Threshold Adversary framework.

VIEW Initially, each party starts in view 0. By the "longest-chain-wins" principle, upon receiving an announcement that extends $L(G)$ to length v, a party moves to view v if it is currently in a lower view.

A party becomes a leader $L(v)$ for view v when solves the puzzle that allows it to announce into G a block $a_{v-1} \leftarrow b$, such that b forms a chain $a_1 \leftarrow ... \leftarrow a_{v-1} \leftarrow b$ of length v. Note that this leader is not necessarily unique, so there may be multiple leaders for view v, and multiple proposals with the same view.

WEDGE Upon starting a new view, a party chooses to work on the puzzle at the end of $L(G)$. By the "longest-chain-wins" principle, after moving to view v, a party rejects messages from views lower than v.

SAFE Successfully solving the puzzle at the end of $L(G)$ implicitly and statistically represents acceptance of $L(G)$ by a large fraction of parties.

ACCEPT A leader $L(v)$ announces to all parties a block $a_{v-1} \leftarrow b$ that extends $L(G)$ to length v. A party chooses to work on the puzzle at the end of $L(G)$, implicitly conveying acceptance.

DECIDE A party in view v may decide on the block at position $v - \epsilon$ in $L(G)$, the longest chain in G. With probability proportional to ϵ, this block will not be overturned by any future insertions to G.

4 Combining NC with BFT

In this section we discuss emerging approaches to combine NC with BFT. Such hybrid solutions address the deficiencies in either arena, and bring the combined benefits of both worlds.

We begin by underscoring challenges of either paradigm.

4.1 Blockchain replication challenges

While Blockchain replication can be used to implement state-machine replication, this implementation has several limitations.

Lack of finality Blockchain replication suffers from a long-term deficiency where there is always the risk that a block will be un-committed. In fact the formal safety seems to require infinite executions and to assume that the relative minority computational bound of the adversary will remain true till infinity. In practice this raises many concerns: what if after some safe period an adversary acquires more than half of the relative computational power and reverts all the transactions?

Commit Latency Blockchain replication suffers from a short-term deficiency. Even if we assume infinite execution and are willing to assume that a block that is buried by an hour's worth of cryptographic puzzles is completely safe, this still means we need to wait an hour! In fact one of the biggest advantages of BFT protocols is that they provide "instant finality". This is a property that the NC does not achieve.

Selfish Mining This is an attack on the fairness property first suggested by Eyal and Sirer [19]. This attack is related to the incentives behind publishing PoW puzzles immediately when they are solved.

4.2 BFT State-machine replication challenges

State-machine replication with Byzantine fault tolerance brings many advantages. Unlike NC, it does not suffer from lack of finality. On the contrary, once a block is committed it can never be revoked. Moreover, modern BFT SMR systems have very low latency and high throughput. In particular these systems are often tuned to provide high performance in the common, failure-free case. Even in a WAN deployment with tens of servers these systems typically manage to commit hundreds (if not thousands) of operations per second in good network conditions and failure free executions.

The permissionless model of NC allows for a much larger set of replicas. It is commonly believed that there are at least several thousands of miners implementing the BitCoin NC replication protocol.

While BFT SMR protocols seem to work well for few tens of servers, scaling these solutions to hundreds or even thousands of replicas raises new challenges.

4.3 Bridging the two paradigms

We proceed to mention a few approaches to combine both the NC and BFT techniques to overcome the above challenges.

The Ethereum Casper [9] approach harnesses BFT to deal with the lack of finality in NC. Casper sets up a trusted committee of validators, who may be selected by placing a security deposit. The committee post-validates NC blocks (buried to a certain depth) via a BFT protocol. Only after a path is validated by the committee it is considered committed. In this approach, the NC chain can be viewed as a client sending a stream of requests to a permissioned BFT engine. Note that due to the slow cadence of requests (e.g., an Ethereum average block time is 14 seconds), the BFT protocol can work in essentially synchronous mode.

Another approach uses NC to elect a rotating committee, and then the committee is responsible for deciding on blocks to add to the chain. The first step in this approach was BitCoin-NG [18], they suggest that after a leader is elected it continues to perform micro-transactions. ByzCoin [26] took this idea a step further, and defined a committee of size C simply as the last C elements in the NC chain. ByzCoin suggests that this committee then proceeds to use PBFT to drive micro-transactions. One challenge is that committee members themselves may not be finalized. To remedy this, a similar approach is taken by Hybrid Consensus [35], where the committee C is chosen as the members that are buried $2C$ to $C + 1$ in the NC chain. In addition to finality of decisions, the benefit of this approach is that the BFT engine provides faster turnaround of (finalized) decisions.

A third approach introduced in Solidus [6] builds the Blockchain itself by consensus and does not use LFW at all. It uses PoW as a way to both mint new currency and rotate a committee. A newly minted block does **not** win by LFW, instead it needs to go through consensus approval by the existing committee in order to be added to the Blockchain. Once a block is added to the chain, it also rotates the committee (similar to Byzcoin and HC), i.e., the new element gets admitted to the committee, and the oldest element retired. Like the above approaches, Solidus provides finality and low latency. In addition, in Solidus the details of *reconfiguring* the committee members are weaved into the Blockchain protocol and are based on traditional SMR-BFT reconfiguration approaches.

Scaling Byzantine agreement through a randomized sample of a sub-committee is, in itself, a known idea (see, e.g., [25]). A vulnerability of all committee-based approaches is that an adversary can mount a targeted attack on the (smaller) set of committee members. Cryptographic techniques were introduced in [24] to hide the committee selection against such a *rushing adversary*. More recently, Algorand [21] uses Proof-of-Stake coupled with verifiable random functions (VRF) to secure a randomized selection of committee members in permissionless setting. Algorand further mitigates vulnerability by having each committee determine only one block, and converging in one communication step by each member.

5 Concluding Remarks

In this tutorial, we discussed the foundational aspect of Blockchain as a Byzantine fault-tolerant state-machine-replication engine.

We first provided a foundational breakdown of NC as a randomized protocol in the synchronous settings. We introduced the abstractions of a randomized oracle, that selects a small number of requests to extend a directed graph. We then expressed NC as a simple combination

of two principles, request to extend the longest chain in the graph, and accept the longest chain as winning.

We then introduced a general framework for classical BFT solutions that preserves two key safety properties: Validity and Agreement. It builds around a view by view solution that explicitly collect information from a quorum of parties, and determines a proper way to the log from one view to the next. We used the framework to relate NC to classical BFT solutions. The NC oracle, by admitting only a small selection of requested block additions, implicitly performs approximate quorum collection.

An exciting emerging direction is to bring the two worlds together. We provided a glimpse into several ideas for "hybrid" solutions.

As discussed in the introduction, the SMR engine of the Blockchain provides only the base layer for the Blockchain stack. Future tutorials may cover the smart contract layer, where there are fascinating connections between this service logic layer and parallel advances in the area of removing trust from centralized implementations.

References

[1] Michael Abd-El-Malek, Gregory R. Ganger, Garth R. Goodson, Michael K. Reiter, and Jay J. Wylie. Fault-scalable byzantine fault-tolerant services. *SIGOPS Oper. Syst. Rev.*, 39(5):59–74, October 2005.

[2] Ittai Abraham, Srini Devadas, Kartik Nayak, and Ling Ren. Brief announcement: Practical synchronous byzantine consensus. In *Proceedings of the Int'l Symposium on Distributed Computing*, October 2017.

[3] Ittai Abraham, Danny Dolev, Rica Gonen, and Joe Halpern. Distributed computing meets game theory: Robust mechanisms for rational secret sharing and multiparty computation. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Principles of Distributed Computing*, PODC '06, pages 53–62, New York, NY, USA, 2006. ACM.

[4] Ittai Abraham, Danny Dolev, and Joseph Y. Halpern. Lower bounds on implementing robust and resilient mediators. In Ran Canetti, editor, *Theory of Cryptography, Fifth Theory of Cryptography Conference, TCC 2008, New York, USA, March 19-21, 2008.*, volume 4948 of *Lecture Notes in Computer Science*, pages 302–319. Springer, 2008.

[5] Ittai Abraham and Dahlia Malkhi. BVP: Byzantine vertical paxos. `https://www.zurich.ibm.com/dccl/papers/abraham_dccl.pdf`, May 2016. Accessed: 2016-11-08.

[6] Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Alexander Spiegelman. Solidus: An incentive-compatible cryptocurrency based on permissionless byzantine consensus. *CoRR*, abs/1612.02916, 2016.

[7] James Aspnes, Collin Jackson, and Arvind Krishnamurthy. Exposing computationally-challenged Byzantine impostors. Technical Report YALEU/DCS/TR-1332, Yale University Department of Computer Science, July 2005.

[8] Adam Back. Hashcash - a denial of service counter-measure. Technical report, 2002.

[9] Vitalik Buterin. Casper version 1 implementation guide. Technical report, 2017.

[10] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, OSDI '99, pages 173–186, Berkeley, CA, USA, 1999. USENIX Association.

[11] Tushar D. Chandra, Robert Griesemer, and Joshua Redstone. Paxos made live: An engineering perspective. In *Proceedings of the Twenty-sixth Annual ACM Symposium on Principles of Distributed Computing*, PODC '07, pages 398–407, New York, NY, USA, 2007. ACM.

[12] Wei Dai. B-money. Technical report, 1998.

[13] Christian Decker and Roger Wattenhofer. Information Propagation in the Bitcoin Network. In *13th IEEE International Conference on Peer-to-Peer Computing (P2P), Trento, Italy*, September 2013.

[14] Danny Dolev. The byzantine generals strike again. Technical report, Stanford, CA, USA, 1981.

[15] Danny Dolev and Rüdiger Reischuk. Bounds on information exchange for byzantine agreement. *J. ACM*, 32(1):191–204, January 1985.

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

[17] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '92, pages 139–147, London, UK, UK, 1993. Springer-Verlag.

[18] Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert Van Renesse. Bitcoin-ng: A scalable blockchain protocol. In *Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation*, NSDI'16, pages 45–59, Berkeley, CA, USA, 2016. USENIX Association.

[19] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. In Nicolas Christin and Reihaneh Safavi-Naini, editors, *Financial Cryptography and Data Security - 18th International Conference, FC 2014, Christ Church, Barbados, March 3-7, 2014, Revised Selected Papers*, volume 8437 of *Lecture Notes in Computer Science*, pages 436–454. Springer, 2014.

[20] Juan Garay and Leonardos Kiayias. The bitcoin backbone protocol: Analysis and applications. In *EUROCRYPT*, pages 281–310, 2015.

[21] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *In Proceedings of the 26th ACM Symposium on Operating Systems Principles (SOSP)*, October 2017.

[22] Stuart Haber and W. Scott Stornetta. How to time-stamp a digital document. In *Proceedings of the 10th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '90, pages 437–455, London, UK, UK, 1991. Springer-Verlag.

[23] Maurice P. Herlihy and Jeannette M. Wing. Linearizability: A correctness condition for concurrent objects. *ACM Trans. Program. Lang. Syst.*, 12(3):463–492, July 1990.

[24] Valerie King and Jared Saia. Breaking the $o(n^2)$ bit barrier: Scalable byzantine agreement with an adaptive adversary. *J. ACM*, 58(4):18:1–18:24, July 2011.

[25] Valerie King, Jared Saia, Vishal Sanwalani, and Erik Vee. Scalable leader election. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*, SODA '06, pages 990–999, Philadelphia, PA, USA, 2006. Society for Industrial and Applied Mathematics.

[26] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. In *USENIX Security Symposium*, pages 279–296. USENIX Association, 2016.

[27] Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong. Zyzzyva: Speculative byzantine fault tolerance. In *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles*, SOSP '07, pages 45–58, New York, NY, USA, 2007. ACM.

[28] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, July 1978.

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

[30] Shengyun Liu, Paolo Viotti, Christian Cachin, Vivien Quéma, and Marko Vukolic. Xft: Practical fault tolerance beyond crashes. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, pages 485–500, Berkeley, CA, USA, 2016. USENIX Association.

[31] Dahlia Malkhi and Michael Reiter. Byzantine quorum systems. *Distrib. Comput.*, 11(4):203–213, October 1998.

[32] Brian M. Oki and Barbara H. Liskov. Viewstamped replication: A new primary copy method to support highly-available distributed systems. In *Proceedings of the Seventh Annual ACM Symposium on Principles of Distributed Computing*, PODC '88, pages 8–17, New York, NY, USA, 1988. ACM.

[33] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *Proc. USENIX Annual Technical Conference*, pages 305–320, 2014.

[34] Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. In *EUROCRYPT (2)*, volume 10211 of *Lecture Notes in Computer Science*, pages 643–673, 2017.

[35] Rafael Pass and Elaine Shi. Hybrid consensus: Efficient consensus in the permissionless model. *IACR Cryptology ePrint Archive*, 2016:917, 2016.

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

[37] Nick Szabo. Smart contracts: Building blocks for digital market. Technical report, 1996.

[38] Nick Szabo. Bit gold. Technical report, 2005.