Ten Myths About Blockchain Consensus

David Hyland and João Sousa and Gauthier Voron and Alysson Bessani and Vincent Gramoli

Abstract Blockchain and distributed ledger technologies have become key to develop secure decentralized applications. Their potential applications span various industries and their use-cases have already demonstrated the potential of such applications. Interestingly, these technologies rely on a large body of complex research topics like the Byzantine consensus problem. Although such problem was defined four decades ago, its subtle ramifications are largely misunderstood by many blockchain developers, let alone application programmers who build upon these blockchains. These misconceptions are dramatic as they prevent these applications from working efficiently and they make them vulnerable to attacks. This chapter debunks the major myths about blockchain consensus. First, it lists ten myths about blockchain consensus that are appearing frequently in the blockchain community. Second, it presents clarifications that debunk these myths to help build safer and more efficient blockchain and distributed ledger systems. These misconceptions are illustrated with the evaluation of three distributed ledgers, Hyperledger Fabric, Redbelly Blockchain and R3 Corda, as well as three important consensus algorithms, BFT-SMaRt, Democratic BFT and HotStuff.

David Hyland
University of Sydney, Australia, e-mail: dhyl2836@uni.sydney.edu.au

João Sousa
Faculdade de Ciências, Universidade de Lisboa, Portugal e-mail: jcsousa@fc.ul.pt

Gauthier Voron
University of Sydney, Australia, e-mail: gauthier.voron@sydney.edu.au

Alysson Bessani
Faculdade de Ciências, Universidade de Lisboa, Portugal e-mail: anbessani@fc.ul.pt

Vincent Gramoli
University of Sydney and EPFL, Switzerland e-mail: vincent.gramoli@sydney.edu.au

1 Introduction

Over the last decade, blockchain experienced an important momentum leading to a plethora of new consensus protocol proposals (e.g., Ripple's consensus algorithm [55], Democratic BFT (DBFT) [23], Aura [41], Casper [15], HotStuff [66], IBFT [46], Tendermint [43]). A consensus protocol is a key element of the blockchain system as it helps a distributed set of machines agree on a unique block at each given index of a chain. In contrast with the problem of consensus that has been known by the distributed computing community for the past four decades [52], a significant part of these proposals is often unclear. Most of these new consensus protocols are described in white papers, wikis and online documentations, rather than in more traditional academic publications and it is unclear whether they satisfy the application requirements [18].

As a result, there are several misconceptions about what is blockchain consensus, the guarantees it should offer and how it could be made secure and fast. For example, efforts to standardize blockchain technologies have revealed ambiguous terminologies [36] and, when consensus protocols are stated informally, one can easily find scenarios in which these protocols fail [6, 18, 32, 61]. As blockchains are becoming important components to guarantee the security of critical applications, these myths can have devastating effects, whose latest example is probably the vulnerability of some of the mostly deployed blockchain software (e.g., `parity` and `geth`) [30], used to handle high-value digital assets. Other examples include the recent efforts devoted to develop quantum resilient cryptographic software on top of blockchain systems that are already vulnerable to the misbehavior of a single participant [19]. While cryptography and fault tolerance address problems that may seem orthogonal, a blockchain cannot provide the security level required by such applications without both cryptography and fault tolerance (cf. Section 7).

As blockchains are now being offered as a service by most cloud providers and have become the cornerstone of various applications, it is crucial to clarify some of the misconceptions that will have, sooner or later, dramatic consequences. Some of these misconceptions could be attributed to the lack of knowledge of the distributed computing and database literatures. First, there are various failure models in which an algorithm can solve consensus. In fact, a consensus algorithm typically allows n nodes (or processes) reach an agreement despite f of them failing. These failures are generally classified in two types: crash, where a node simply stops, and Byzantine, where a node behaves arbitrarily, misbehaving either accidentally or with malicious intent. This is why we distinguish crash fault-tolerant (CFT) from Byzantine (or arbitrary) fault-tolerant (BFT) protocols (cf. Sections 6 and 8). Second, the three properties of the classic consensus problem [44] are often misinterpreted: (i) validity requires that the value decided by a non-faulty (or correct) node has to be valid, (ii) agreement requires that two non-faulty nodes cannot decide differently, and (iii) termination requires that eventually the non-faulty nodes decide. Also, the factors affecting the performance of a blockchain are often misunderstood (Sections 4 and 5) as they typically embed, in addition to the consensus protocol, other components like cryptography (Section 11). This may distract engineers from other

Section	Myth	Clarification	At risks
3.	Po* solves consensus	Po* selects blocks without guaranteeing uniqueness.	users
4.	Consensus bottlenecks blockchain in LANs	BFT-SMaRt runs faster outside Hyperledger.	designers
5.	Consensus bottlenecks distributed ledgers in WANs	DBFT and BFT-SMaRt scales better outside Corda.	designers
6.	CFT consensus tolerate Byzantine faults	BFT quorums are larger than CFT ones.	users
7.	Signatures and hashes make blockchain secure	They do not cope with disagreement & double spending.	users
8.	A CFT blockchain with a BFT consensus becomes BFT	The whole communication pattern needs to be changed.	users + designers
9.	BFT consensus needs linear message complexity	Some quadratic complexity algorithms perform better.	designers
10.	Reconfiguring consensus participants is easy	It is difficult to make it non-disuptive.	designers
11.	Blockchain performance is not limited by cryptography	CPU demand can surpass the network demand of consensus.	designers
12.	Blockchain needs to solve the classic consensus problem	Deciding $\Omega(n)$ proposals help scale.	designers

Table 1: A summary of common Blockchain consensus myths.

challenging problems (Sections 9 and 10). One of the most common misinterpretations is illustrated by the large family [49] of so-called "proof-of-$*$ consensus (*sic*)" that cannot ensure agreement upon a unique block and may lead the blockchain to counter-intuitively fork into multiple branches (cf. Section 3).

The aim of this chapter is to bust ten myths about blockchain consensus, summarized in Table 1. These myths correspond typically to misinformation that professionals and students commonly learn by reading posts and blogs online before attending a blockchain course. We proceed by listing each myth and explaining why it is incorrect by offering a counter example, sometimes using a blockchain (Hyperledger Fabric, Redbelly Blockchain or R3 Corda) or a consensus algorithm (BFT-SMaRt, DBFT or HotStuff). Our aim is not to survey existing approaches for blockchain consensus, as there is an extensive literature on the subject (e.g., [18, 49]), neither to provide a formal treatment of these myths but rather to state them in a pedagogical language to reach the blockchain community at large. As some of these clarifications already helped improving the scalability of blockchains (Section 12), we hope that they will help build secure and efficient blockchain applications in the future.

2 Background on Consensus and Proof-of-$*$

Before listing each myth, we present various interpretation of the term "consensus" in the blockchain context. A blockchain is easily understood as a chain of blocks abstraction where new blocks get regularly appended, however, the system that

replicates this abstraction on multiple machines or nodes of the network does not always maintain the sequence structure of the chained blocks.

Instead a more precise description of this abstraction is a directed acyclic graph or—to put it simply—a tree structure whose nodes are blocks and whose edges are directed upwards in the tree, from children blocks to their parent block [34]. Each of these edges is implemented as a hash of the content of the parent block stored as a field of the child block. A parent block has multiple children in the tree as soon as the blockchain forks, which means that two nodes disagree about the block that should be inserted at the next available index of the chain. The consensus abstraction is employed to guarantee that there is no such disagreement among all nodes about the unique block to be appended next.

The consensus problem was defined more than half a century ago [52] and requires to guarantee that if replicas propose their block, then no two replicas should decide differently (Agreement), the block that is decided is one of the proposed block (Validity), and the non-faulty replicas should eventually decide (Termination). While this definition presents some inherent limitations for scalability that will be discussed in Section 11, it paved the way for research on consensus algorithms in small settings and influenced newer definitions of the consensus problem considering cryptography [17] and scalability [23, 24].

Interestingly, the term "consensus" has been extensively used on blog posts and websites about blockchain to denote proof-of-$*$ methods of selecting a subset of blocks proposed to a particular index of the blockchain. As surveyed in [49], these proof-of-$*$ (Po*) methods include proof-of-work, proof-of-reputation, proof-of-lock, proof-of-activity, proof-of-stake, proof-of-burn, proof-of-authority, proof-of-location, each choosing a different local attribute in order to decide whether a block is selectable to a particular index. These attributes include the computational power of the node, stake owned by the node in the considered blockchain system, its activity, location, etc. The diversity of existing proof-of-$*$ makes it difficult to list them all here and is not part of the scope of this paper.

3 Myth #1: Proof-of-$*$ solves consensus

A proof-of-work [28] is a mechanism to limit the power of the adversary that can control the Byzantine participants. By requesting each participant to solve a moderately complex crypto-puzzle and produce a proof of this effort, called proof-of-work, before producing a block [48], the ability for the adversary to overwhelm the system with new valid blocks becomes limited by its computational power. Many alternatives to this proof-of-work mechanism have been proposed, like proof-of-stake that limits the power of the adversary to its stake, hence leading to the large family of proof-of-$*$ mechanisms. For a list of more than a dozen of these proof-of-$*$ proposals, we refer the interested reader to a survey on the topic [49].

As it is widely believed that proof-of-∗ protocols solve consensus,[1] many protocols have been called "proof-of-∗ consensus (sic)". As an example, Coin Telegraph explains how proof-of-work can be seen as the "original consensus algorithm in a blockchain network" by using a "complicated mathematical puzzle and a possibility to easily prove the solution".[2] It may seem as if "proof-of-work consensus" was a metonymy aimed at being pedagogical and referring to proof-of-work as the component of a more complex consensus procedure. However, the frequent forks [65] in proof-of-work blockchains indicate the possibility of disagreements, which contradict the scientific notion of consensus [52]. This metonymy, albeit simple, hides more complex intricacies. Such simplifications put the users at risk, because, strictly speaking, consensus prevents double spending that could be induced by forks whereas proof-of-∗ actually tolerates forks and may lead to double spending.

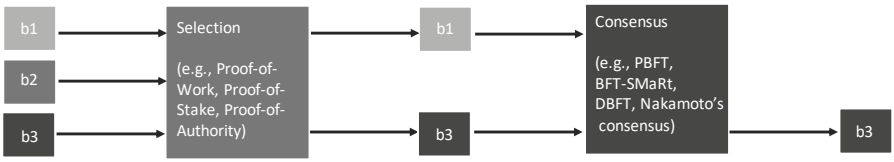


Fig. 1: Proof-of-∗ mechanisms do not solve the consensus problem. Instead, they select a subset, say $\{b1, b3\}$, of proposed blocks, say $\{b1, b2, b3\}$, as legitimate based on some attribute (computational power, coins owned, etc.) of the nodes that generated them but a separate consensus algorithm is necessary to guarantee that the block, like $b3$, decided for a given index of the chain is unique.

In fact, the proof-of-∗ mechanism cannot solve consensus but is instead a mechanism to enforce some of the consensus prerequisites so that a proper consensus algorithm can be executed. Typically, a proof-of-∗ mechanism selects a small subset of nodes to participate in the consensus by proposing a block for the same index of the chain as depicted in Figure 1, where $\{b1, b3\}$ are proposed. As this mechanism cannot guarantee that a unique block is proposed, it is insufficient to decide a single block and does not solve the consensus problem. Recently, researchers have tried to clarify the terminology to address this confusion [36] but it remains well spread on the Internet.

Example: Bitcoin

Bitcoin [48] uses proof-of-work to limit the number of nodes that can create a new block for a given index of the chain. Once these blocks are created, Bitcoin solves

[1] Wikipedia makes use of the term "proof-of-work consensus (*sic*)" at `https://en.wikipedia.org/wiki/Proof_of_work`.

[2] `https://cointelegraph.com/explained/proof-of-work-explained`.

consensus by deciding among multiple candidate blocks (that all contain a valid proof-of-work) the unique block that is part of the longest branch. This is the reason why Bitcoin [48] uses an extra mechanism, Nakamoto's consensus, to try to solve the problem: it selects the blocks of the longest branch by assuming that the delay to create and propagate every block to all miners in the network is upper-bounded such that every miner can safely choose the candidate block at index i once they have waited long enough [33]. The proof-of-work mechanism helps limit the block creation speed by requiring that a node first solves a computationally hard crypto-puzzle before propagating a new block. This proof-of-work mechanism alone does not ensure the uniqueness of the block at each index—this is precisely why a separate consensus protocol based on identifying the block of the longest branch is required in Bitcoin.

4 Myth #2: Consensus is the bottleneck of blockchains in LANs

Distributed consensus performance (expressed for example in decisions per unit of time) generally degrades with the number of participants. This led to the folklore belief that the performance of consensus is the key limiting factor of the performance of the blockchain in normal executions as was described by NEC.[3] Typically, BFT consensus algorithms involve a quadratic number of messages [12, 20, 23] that is believed to rapidly consume the limited bandwidth resource. For example, the classic PBFT consensus algorithm [20] employs a distinguished participant, the leader, proposing a value and having participants exchanging sufficiently many prevotes to make sure that no other value will be accepted, then having participants exchanging sufficiently many votes to make sure the value is accepted by other participants before deciding this value.

For this reason, it has long been thought that a blockchain performance is limited by the performance of its consensus, and recent solutions, like HotStuff [66], attempted to boost blockchain performance by reducing the message complexity of the consensus algorithm using threshold signatures (as discussed in Section 9). HotStuff follows the classic leader-based pattern but reduces the quadratic message complexity, induced by the participants sending prevotes and votes to each other, to the linear complexity, induced by the participants sending their prevotes and votes only to the leader that retransmits them.

It turns out that there are other scalability limitations in blockchain systems, especially in Local Area Networks (LANs)—one example is the verification of transaction signatures [11, 63]. Traditional blockchain would require every node to verify the signatures of every transaction. Although there are some blockchains that employ verification sharding [24] for each signature to be verified by only $f + 1$ nodes, in general, even this optimization is not sufficient to make the verification

[3] https://www.nec.com/en/global/insights/article/2020022520/index.html.

finish significantly earlier. We present the cost of verification sharding in Myth #11. Additionally, previous works [67] point the execution of smart contracts as a significant scalability limitation, specifically the frequent disk operations they cause as well as their computational cost.

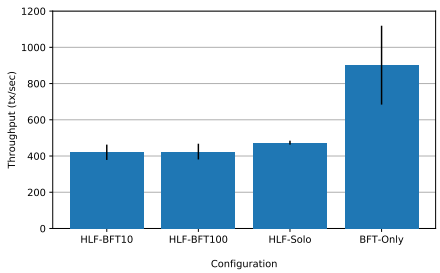


Fig. 2: In a LAN, Hyperledger Fabric running with an experimental BFT orderer based on BFT-SMaRt is slightly slower than without consensus (HLF-Solo) but significantly slower than the BFT orderer alone (BFT-Only). This indicates that the bottleneck of blockchains is not necessarily their consensus component.

Example: Hyperledger Fabric

In order to test our hypothesis on the existence of other performance limitations, we selected the Hyperledger Fabric blockchain [3] that aims at being modular and offering privacy. Even though Hyperledger Fabric v1.x is crash fault tolerant and does not aim at tolerating malicious behaviors, there exists a prototype version [57] that makes use of a BFT orderer (cf. Section 8 for more details) based on BFT-SMaRt. BFT-SMaRt [12] is a modular BFT consensus protocol in which a stable leader sends a proposal to all nodes that confirm its consistency in two all-to-all communication steps, similarly to PBFT [20]. The goal of the test is to observe whether Hyperledger Fabric could offer similar performance as its consensus or orderer component alone, when run in the same environmental settings.

We ran the Hyperledger Fabric blockchain system and a BFT-SMaRt baseline on four machines, each with Ubuntu Linux 14.04.1 LTS, hosted in Dell PowerEdge R410 servers. Each machine has 32 GB of memory and two quad-core 2.27 GHz Intel Xeon E5520 processor with hyper-threading. These machines communicate through an isolated gigabit Ethernet LAN. The results were obtained with Hyperledger Caliper [16] and are averaged over 5 runs with error bars showing the standard deviation. Figure 2 depicts throughput for HLF-BFT for blocks of 10 (HLF-BFT10) and 100 (HLF-BFT100) transactions of payload 4 KiB and envelopes of 900 bytes, HLF-Solo for blocks of 100 transactions (Caliper benchmarks could not complete with blocks of 10) and a BFT-Only benchmark with blocks of 10 transactions. We can

clearly observe that the Byzantine fault-tolerant orderer based on BFT-SMaRt [57] performs 2× better than Hyperledger Fabric with the same version of this orderer. Furthermore, the BFT-SMaRt replication library alone, without the required Fabric machinery, can order 5× more 4 KiB transactions per second than what was obtained in the orderer [12].

5 Myth #3: Consensus is the bottleneck of distributed ledgers in WANs

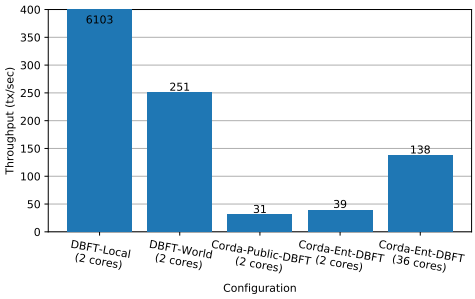
Section 4 showed that consensus is not necessarily the bottleneck of a blockchain in a LAN. This result could be attributed to the blockchain prototype in use, and it is interesting to explore whether the same observation would apply to a distributed ledger like Corda [14] that aims at enabling businesses to transact in privacy using smart contracts without necessarily chaining blocks. Additionally, Section 4 did not contradict the belief that the consensus protocol could be the bottleneck of a distributed ledger deployed on the Internet. A LAN typically offers bandwidth resources that are not comparable to the resources one could obtain in a Wide Area Network (WAN) subject to congestions, long distances, etc.

Here we push the study one step further by selecting a recent BFT consensus protocol designed especially for distributed ledgers, DBFT [23]; two versions of Corda, its public version and its enterprise edition; and running experiments across four regions, in Australia, Brazil, Europe and USA. The goal of these new sets of experiments is to pinpoint what overhead comes from the consensus itself and what overhead comes from other parts of the distributed ledger. If consensus was truly the bottleneck of the distributed ledger, the performance of the distributed ledger would be comparable to the performance of its bottleneck when running in isolation.

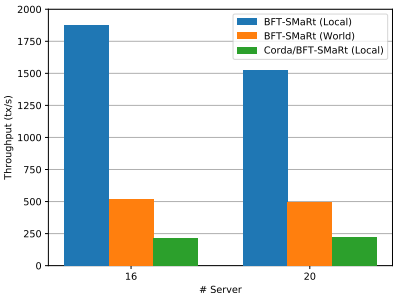
Example: Corda

Corda is crash fault tolerant as it uses the Raft consensus algorithm and cannot tolerate Byzantine failures. In fact, some of us tried deploying a BFT prototype of Corda running on BFT-SMaRt in the past, however, we were not able to make it run and the Corda developers recommended that we use the crash fault tolerant version instead [38]. We thus modified Corda to use BFT-SMaRt and DBFT as consensus algorithms. DBFT is a formally-verified consensus protocol [23, 61] that differs from the previously-mentioned leader-based protocols as each of its n participants reliably broadcasts its proposal to other nodes and spawn n binary consensus instances that define a bitmask in order to output a decision. We measure both the throughput of the Corda distributed ledger, its public and enterprise version Corda 4.0-SNAPSHOT, as well as the throughput of the BFT-SMaRt and DBFT consensus algorithms, as the average observed over 4 runs on Amazon Web Services (AWS) EC2 instances. As

in Corda, each transaction has a serialized size of 8 KiB bytes (that is much larger than the original size), we use the same transaction size for running the consensus algorithms alone.



(a) Corda w/ DBFT vs. DBFT alone



(b) Corda w/ BFT-SMaRt vs. BFT-SMaRt alone

Fig. 3: The bottleneck of distributed ledger is not necessarily the consensus protocol. (a) When running with the DBFT consensus, the public and the entreprise versions of the Corda distributed ledger are significantly slower than DBFT alone. (b) When running with the BFT-SMaRt consensus, the enterprise version of Corda runs slower than BFT-SMaRt alone.

Figure 3a compares the throughput of DBFT and Corda with DBFT on 4 c5.large instances, each with 1 hyperthreaded Intel Xeon 8124M core. We also add the throughput of Corda enterprise with DBFT on 4 c5.9xlarge instances, each with 18 hyperthreaded Intel Xeon 8124M cores. We execute DBFT in two setups. In the "DBFT-Local" setup, the 4 instances are in a single datacenter. In the "DBFT-World" setup, each instance is in a separate datacenter located in Ohio, Frankfurt, Sydney and São Paulo. We execute Corda in a local setup only as we did not manage to deploy it on a geo-distributed setup. The result indicates that with comparable setups the throughput of DBFT is 156× higher than the throughput we could obtain from any version of Corda running DBFT with the same number of cores. Moreover, DBFT executed on a georeplicated setup on a 2-core machine still has a throughput 1.8× higher than Corda on a local setup with a 36-core machine, and 6.4× higher than Corda on a local setup with a 2-core machine. We were not able to locate precisely the cause of the Corda performance drop, however, we observe that Corda enterprise performance increases with the number of cores. This seems to indicate that the throughput of Corda is limited by computation.

On Figure 3b we compare the throughput of BFT-SMaRt and Corda enterprise with BFT-SMaRt on 16 and 20 c5.4xlarge instances, each with 8 hyperthreaded Intel Xeon Platinum 8275CL cores. Similarly to DBFT, deploy BFT-SMaRt in two setups. In the "Local" setup, all instances are in the same datacenter. In the "World" setup, instances are spread among 4 datacenters in Ohio, Frankfurt, Sydney and Sao Paulo.

We execute Corda in a local setup only as we did not manage to deploy it on a geo-distributed setup. We observe that, on a local setup, the throughput of BFT-SMaRt is consistently more than 6× larger than the throughput of Corda. Moreover, even when executed in a geo-distributed setup, the throughput of BFT-SMaRt is still more than 2× larger than Corda executing in a single datacenter. Since the only difference between BFT-SMaRt and Corda is the set of operations outside the consensus, this performance gap indicates that these additional operations are the bottleneck of the Corda distributed ledger.

6 Myth #4: CFT consensus algorithms are safe under Byzantine faults

There is a misconception that CFT protocols are enough for implementing a safe blockchain consensus. In particular, the possibility that Byzantine nodes can affect the safety of these protocols is often overlooked: two recent consensus algorithms, called Clique and Aura [41], have been integrated in two of the mostly deployed blockchain software, called `geth` and `parity`, in order to cope with $\lceil n/2 \rceil - 1$ Byzantine failures, a problem that is known to be unsolvable [13]. As a result, the Attack of the Clones led to double-spending in two testnets running these software [30], hence demonstrating, in practice, this safety violation. More generally, consider a system with n nodes in which up to f can fail. In a practical system, where there are no strict time bounds for communication, a node can wait for responses from at most $n - f$ nodes (a quorum) since up to f nodes can be faulty and never answer. In such systems, quorums must intersect in at least one correct process to ensure actions observed from a quorum (i.e., a subset of $n - f$ nodes) are observable from any quorum. In the crash failure model, all $n - f$ responding nodes are correct, and thus $n > 2f$ (*the number of nodes in any two quorums must be greater than the total number of nodes in the system*, i.e., $(n - f) + (n - f) > n$) is enough to ensure intersection. In the Byzantine model, there might be faulty nodes in an accessed quorum sending wrong values, therefore $n > 3f$ (*the number of nodes in any two quorums must be greater than the total number of nodes in the system plus the maximum number of "liars"*, i.e., $(n - f) + (n - f) > n + f$) and quorums intersecting in at least $f + 1$ nodes.

Example: Raft

In Figure 4, one can see that two quorums of three nodes intersecting at one node could be sufficient to ensure safety if nodes can only crash but would violate safety if the intersecting node misbehaves by responding that no new transaction *tx* has been written. Raft [51] is a CFT protocol currently employed in popular blockchains such as Corda [14] and Hyperledger Fabric [3]. In Raft, a leader coordinates the replicated

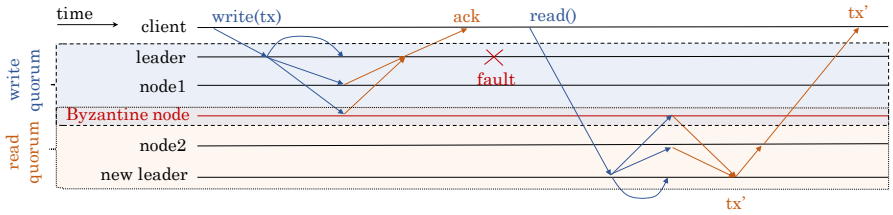


Fig. 4: A possible execution of a replicated write-then-read with a quorum size of $\lfloor n/2 \rfloor + 1$. We assume the quorum leaders are correct for simplicity. The execution would be correct in a CFT model but is incorrect in a BFT model. The intersection of BFT quorums should be larger than the intersection of CFT quorums for the consensus protocol to guarantee safety. If the intersection contains only Byzantine nodes, then the response obtained after reading from a quorum can violate safety, which can be exploited to double-spend in cryptocurrency applications.

execution by writing the next transaction to be processed onto a write quorum and then informs the client that its transaction is committed as depicted in the distributed execution of Figure 4 where time increases from left to right. If, after this, the leader fails before committing this transaction in all servers and a new leader takes over, this new leader will read the pending transactions to ensure the same safety requirement of the algorithm: *that if some server committed this transaction, all servers will commit it in the same position on the transaction log*. If this new leader accesses a read quorum that intersects with the write quorum in a single server (as in the figure), and if this server is Byzantine and misbehaves, then it can lie about what happened (ignoring or replacing the transaction) and make the new leader commit a corrupted state, hence *violating safety*. This is precisely why, in contrast with Raft, BFT systems typically use quorums whose intersections contains at least $f+1$ nodes.

7 Myth #5: A blockchain with signatures and hashes is secure

An important misconception is that cryptography could solve the same problem as fault tolerance as illustrated by quantum-resilient efforts devoted on top of fundamentally insecure blockchains [19]. In particular, public-key cryptosystems are typically used in blockchains to ensure authentication: if a transaction is not signed by the owner of the account from which it tries to withdraw, then it is rejected—meaning that the signature is incorrect. Combined with hashes these signatures guarantee tamper-evidence in that a non-authorized change will be easily detected as the hash of the signed transaction will not correspond to the observed signatures. These guarantees are not sufficient for security to ensure for example that a client is observing the right blockchain version. A malicious node could convince a client who acts as a merchant that a transaction buying its product is committed while it is not. To do

so, the malicious node presents a fake version of the blockchain where it appends correctly signed transactions that it created without ever sending them to the actual blockchain.

This misconception is illustrated by some recent efforts devoted to develop quantum-resilient cryptographic software [19] on top of blockchain systems because these blockchains cannot tolerate the misbehavior of a single participant. In fact, quantum resilience aims at coping with the misbehavior of a quantum computer node to avoid asset theft, yet the misbehavior of a single node participating in the underlying CFT blockchain is sufficient to produce this theft, hence defeating the purpose of the quantum resilience.

Example: Raft

To guarantee that a client can truly identify whether its transaction is committed, it must make sure that $f + 1$ nodes say so. If not, it risks to be a victim of a double spending where the coins used in the supposedly committed transaction of the fake version are actually reused in a truly committed transaction of the blockchain. Notice that the use of signatures cannot help in the Raft scenario described in Myth #6. If the write operation of the leader is signed in Figure 4, the Byzantine node on the intersection cannot create a different proposal (with tx') for the leader, but it can still hide it, stating it has not received any transaction at all. In this way, the transaction, despite being confirmed to the client, will be "undone" from the system history, opening up the possibility of double spending attacks that may be devastating for financial applications. More generally, there exist various attacks against blockchains that rely on delaying messages (e.g., selfish mining [31], eclipse attacks [39], BGP hijacking [5] and more generally man-in-the-middle attack [29]) that an attacker can exploit to double spend without violating authentication or forging private keys.

8 Myth #6: Adding a BFT consensus to a CFT blockchain makes it BFT

Modularity is important for enabling project collaboration at the heart of open source communities. Due to this, several blockchain platforms have been built with modular or replaceable components [3, 8]. Hyperledger Fabric separates the orderer service in charge of ordering transactions from the endorser service in charge of validating transactions to offer modularity [3]. In particular, Fabric offers several interchangeable orderers: a centralized service, a CFT one, and an experimental BFT one [57], hence illustrating this modularity.

A common mistake stems from extrapolating the guarantees offered by a module of a blockchain system to the whole blockchain system, for example when using

Corda in applications where a consortium of competitors do not trust each other,[4] or when using a BFT orderer within Hyperledger Fabric, which tolerates only crashes. To illustrate this issue, let us consider a distributed ledger that tolerates only crash failures, like Corda or Hyperledger Fabric—this type of distributed ledgers can thus only run in a trustworthy environment as its guarantees would be violated if a participant misbehaves. A developer who want to make the blockchain service more robust may identify the consensus module as a module that is limited by its crash fault tolerance and be tempted to replace it by a more robust module offering Byzantine fault tolerance. If the blockchain system is modular enough, then the developer expects the BFT consensus module to have the same interface as the CFT module. However, typical BFT services do not have the same interface as CFT services, precisely because the acknowledgement of a Byzantine server does not convey the same guarantee as the acknowledgement of a correct server. Figure 5 depicts the problem that might happen when the interface remains unchanged while the model changes from CFT to BFT: the client might be fooled by a flawed acknowledgement coming from a Byzantine front-end while it could only receive correct servers in a CFT model.

Fig. 5: A client contacting a single Byzantine front-end server makes the BFT consensus useless. This is why, replacing one component of the software architecture is rarely sufficient to make a blockchain originally tolerant to crash failures, a blockchain that also tolerates Byzantine failures.

Example: Hyperledger Fabric and BFT-SMaRt

Hyperledger Fabric [3] defines a modular architecture for supporting permissioned blockchains, which is used in production since version 1.0.[5] An important component of this architecture is the orderer service, a group of nodes responsible for ordering and packing transactions into blocks. At the time of writing, Fabric supports only CFT orderers: one based on Apache Kafka and another based on CoreOS' Raft implementation. In both cases, a single front-end/server receives the transaction to be ordered, and produces and signs the block (with ordered transactions) in the

[4] https://www.corda.net/samples/

[5] https://www.fintechfutures.com/2017/03/ibm-cloud-launches-enterprise-ready\-blockchain-for-hyperledger-fabric/.

end. This design still tolerates crashes because there are redundant front-ends in the system, and in case of failures a backup front-end takes over the role. However, in this model, even if we replace Raft by PBFT [20] the front-end of the ordering service will still be receiving and signing the block. An experimental BFT orderer available for Fabric [57] changes this design to make the server create a block and sign it, however, it impacts modularity. In the end, the final generated block is signed by at least $f+1$ servers. Yet, this change requires additional changes in other modules, say to verify these $f+1$ signatures. This shows that replacing a module was insufficient.

9 Myth #7: BFT consensus needs a linear message complexity

It is often the case that bad consensus performance at large scale is attributed to the all-to-all communication pattern of state-of-the-art consensus protocols [22, 64, 66]. In fact n servers sending messages to all other servers necessarily lead to a quadratic messages complexity [12, 20, 21]. As a result, lots of efforts have been devoted to replacing all-to-all message exchanges by one-to-all exchanges [1, 7, 8, 42, 66]. The consensus protocol at the heart of Facebook's blockchain [66] is a typical case which uses threshold signatures to replace all-to-all by one-to-all communications. This result in a linear message complexity for the steady state case. It appears that improving this message complexity does not necessarily lead to better performance. Several factors may limit the throughput of a replicated state machine before the message complexity becomes an issue. Some of these factors are the computational cost of cryptography (cf. Myth #11), the latency of writing new blocks to stable storage [11], the unique value decided per consensus instance (cf. Myth #12), and the use of a slow leader [9, 4].

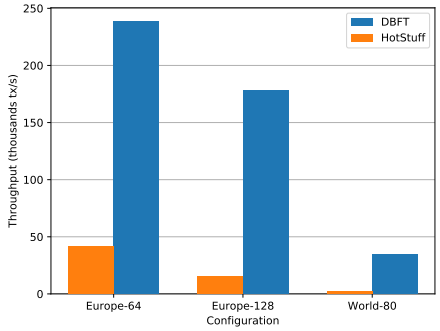


Fig. 6: Linear message complexity does not always bring good performance. The throughput of DBFT, which has a quadratic message complexity, is always higher than HotStuff, which has a linear message complexity in the steady state. This is because DBFT balances the network load on more links.

Example: HotStuff vs. Democratic BFT (DBFT)

In Figure 6, we compare the throughput of two systems, HotStuff [66] and DBFT [24], when the number of server increases on two different setups. On the "Europe" setup, a total of 64 or 128 servers run in four different AWS datacenters in Ireland, London, Paris and Frankfurt. On the "World" setup, a total of 80 servers run in four different AWS datacenters in Ohio, Frankfurt, Sydney and São Paulo. Each server is an AWS EC2 c5.xlarge instance with 2 hyperthreaded Intel Xeon 8124M cores for a total of 4 hardware threads. HotStuff has a linear message complexity while DBFT has a quadratic message complexity. The DBFT replicated state machine does not use any specific optimization and is written in Java. HotStuff replicated state machine uses pipelining to reduce communication delays and is written in C++. We use transactions of 400 bytes, which is a typical Bitcoin transaction size. We observe that DBFT has a throughput up to 11× larger than HotStuff in the "Europe" setup and 16× larger in the "World" setup despite the fact that HotStuff has a better message complexity. We explain this difference by the fact that the HotStuff consensus is leader based and uses expensive threshold signatures while DBFT consensus is leaderless and its most expensive operation is a cryptographic hash. While we do not deny the impact of message complexity, we stress that other bottlenecks limit a consensus performance before message complexity becomes an issue.

10 Myth #8: Reconfiguring consensus participants is easy

Consortium blockchains rely on a globally known set of participants. This set of participants does not change while the consensus decides a new block. However, it is necessary or even desirable to change this set regularly between the consensus executions (e.g., to prevent bribery attacks). This operation, called reconfiguration or membership change, seems to be a built-in feature of some blockchains. For example, Tendermint design mentions a validator set changes [58] but unfortunately this requires to shutdown the blockchain. One of the challenge is purely related to software design as it is generally hard for a complex system to handle components hotplug [40, 47]. A more subtle issue is the possibility of blockchain forks when reconfiguration is employed, since participants already removed from the group can be compromised and cope with the ledger [11]. More generally, the challenge is for the system client to know where to send requests as the participant set changes over time. This last challenge is especially hard since a malicious frontend could always present fake participant set in a scenario similar to Myth #8.

Examples: Hyperledger Fabric and Tendermint

Hyperledger Fabric aims at supporting membership changes without compromising the network, however, it still *"requires that the peer or orderer process is restarted"*.[6] Tendermint mentions validator set changes [58], however, this requires an external application that handles those reconfigurations.[7] BFT replication systems like BFT-SMaRt also require a trusted external application to manage such reconfigurations [12]. New implementations to reconfigure the Byzantine consensus participants rely on consensus protocol themselves and need a trusted domain name system to catch up with the latest configuration [62], and a forgetting protocol to ensure removed participants are unable to validate blocks from an invalid fork [11]. These new implementations make it possible for alive participants (i.e., participants being notified of configuration changes when they happen) to keep track of the system configuration. However, participants joining the system for the first time or catching up after being offline still have to rely on a permissioned trusted infrastructure that cannot be reconfigured, hence falling back on the initial challenge.

11 Myth #9: Blockchain performance is not limited by the cryptography

Blockchain is believed to be bottlenecked by the large bandwidth consumed by its Byzantine consensus component. This is the reason why substantial efforts were devoted to reduce communication complexity at the price of increasing CPU overhead through the use of threshold signatures [66] (cf. Section 9). There is, however, a CPU bottleneck induced by the cryptographic verification of transaction signatures inherent to blockchain systems [24, 60] that is often neglected. In particular, this CPU-intensive verification is necessary to ensure that a Byzantine node does not withdraw an amount of assets from the account of someone else. Each transaction has thus an associated signature that needs to be cryptographically verified. This verification limits the speed at which the blockchain can commit transactions.

Example: Redbelly Blockchain

Redbelly Blockchain [24, 59] relies on verification sharding to minimize the number of verifications needed per transaction and to maximize its performance. The idea is to have each transaction being verified by between $f + 1$ and $2f + 1$ nodes instead

[6] `https://hyperledger-fabric.readthedocs.io/en/latest/msp.html`.

[7] See Section 7.1, second paragraph, of `https://atrium.lib.uoguelph.ca/xmlui/bitstream/handle/10214/9769/Buchman_Ethan_201606_MAsc.pdf?sequence=7&isAllowed=y`

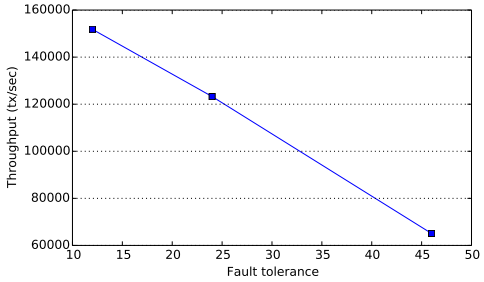


Fig. 7: The performance of Redbelly Blockchain decreases with the number of verifications of transaction signatures that is linear in the fault tolerance.

of all nodes in order to minimize CPU consumption. Figure 7 depicts the throughput (expressed in 400-byte transactions per second) of Redbelly when increasing the number of failures f on 140 c4.8xlarge AWS VMs geo-distributed on four continents (see details in [24]). The performance decreases linearly with the system' fault tolerance f precisely because the number of verifications per node increases, illustrating the negative impact of verifications on the throughput of Redbelly Blockchain. Note that similar observations have been made in the context of Hyperledger Fabric [60] where the caching of verified endorsement certificates led to significant performance improvements.

12 Myth #10: Blockchain needs to solve the classic consensus problem

A blockchain system needs a consensus protocol to totally order a series of blocks in order to guarantee that the blockchain it implements remains a chain. If disagreement happens, then the blockchain forks, making the system vulnerable to double spending. As participants can be incentivized to steal assets from others, most systems resort to traditional Byzantine consensus protocols [52], which ensures (1) agreement in that correct nodes cannot decide different values, (2) termination in that a correct node eventually decides, and (3) validity in that the decided value is one of the proposals. This is what is done in most blockchain systems that rely on a BFT consensus: Tendermint consensus [43] uses a variant of DSL [27], Facebook's Libra [8] uses HotStuff [66], and an orderer of Hyperledger Fabric uses BFT-SMaRt [12].

These protocols share the same goal of guaranteeing that only one proposal among all the proposals of the system get decided. This means that if each server proposes $O(1)$ transactions to the consensus service, then the number of transactions decided by the consensus is $O(1)$, regardless of the number of servers participating in the protocol. This happens because they ensure this classic notion of consensus validity where the value decided is one of the proposed values. There are variants of the

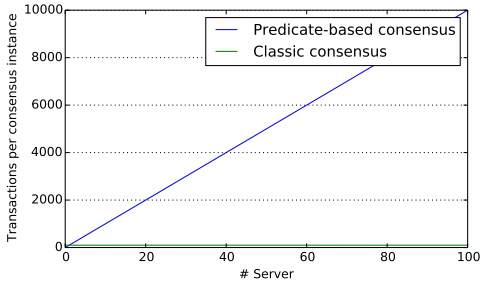


Fig. 8: A protocol solving the classic consensus definition would commit less transactions than one solving predicate-based validity consensus.

consensus problem definition for the Byzantine model that replace this property. For instance, in interactive consistency [44], which is well suited for synchronous system but cannot be solved in a partially synchronous environment, the proposed values of all correct nodes are obtained in the end. Another definition is vector consensus [26, 50] that requires servers to decide the same vector containing at least $f + 1$ values proposed by different servers, where f is the maximum number of Byzantine failures. Unfortunately, in blockchain systems one cannot guarantee that $f + 1$ values are valid and thus, we would need to select the union of the valid transactions in each vector position. In particular, proposers (or miners) typically group transaction requests from client and cannot guarantee that all the received transactions are valid. An alternative to this problem is thus to relax the validity further to only accept values as long as they are deemed valid by the application (e.g., are correctly signed [12, 17]), but without any restrictions on their numbers [23, 24].

Example: validity predicate-based Byzantine consensus problem

With this new consensus definition called validity predicate-based consensus [23], one can thus have distinct servers collecting transaction requests from the clients and then propose a batch of these transactions to a common consensus instance. If each of n server propose distjoint batches of $\Omega(1)$ transactions, and provided that the transactions are correctly signed, then the number of transactions that can be decided at the end of the consensus is $\Omega(n)$. This approach is currently implemented in the pipeplined Byzantine state machine replication, called Dispel [63], and in Redbelly Blockchain [24, 59] in the form of a Set Byzantine Consensus solution. As depicted in Figure 8, the number of committed transactions per consensus instance is linear in the number of servers participating and the throughput of the blockchain system grows with the number of servers (until exhaustion of some resources). This Byzantine consensus variant is especially designed for the geo-distributed environment of blockchains.

Final Remarks

Related work. Although we are not aware of any work focusing on debunking blockchain myths, there are several research papers presenting evidence contradicting previous claims about blockchain consensus. Armknecht et al. [6] show that some assumptions listed in the Ripple consensus protocol white paper are not sufficient to ensure its safety. Cachin and Vukolić [18] explain that proof-of-work blockchains do not offer finality. Amoussou-Guenou et al. give counter-examples where Tendermint core does not solve consensus [2]. Saltini and Hyland-Wood [54] explain that the consensus at the heart of the Quorum blockchain does not terminate. Tholoniat and Gramoli [61] explain that the consensus at the heart of HoneyBadgerBFT does not terminate and argue in favor of formal verification. We ourselves rely on the correctness of consensus protocols to debunk these myths, but we have chosen the BFT-SMaRt library that has been publicly available for more than a decade [12] and Democratic BFT [23] that has recently been formally verified using parameterized model checking [10].

Prior to this work, various efforts were dedicated to evaluating the performance of distributed ledgers. BFT-Bench [37] predates blockchains and evaluates multiple BFT consensus algorithms. Blockbench [25] as well as [53] compare the performance of proof-of-work blockchains against private blockchains. Han et al. measures the number of participants at which the performance of blockchain drops [38]. Some evaluate proof-of-authority blockchains [45] that rely on a set of n authoritative validators among which at most f can be Byzantine to run the consensus. An evaluation focuses on comparing the performance of Byzantine Fault Tolerant blockchains [56] whereas the recent DIABLO benchmark allows to measure blockchain performance under realistic workloads [35].

Conclusion. We presented ten myths on blockchain consensus that represent misunderstandings about the performance and trust model of existing applications. Since consensus is a key element of blockchain and distributed ledger systems and is instrumental in the design of fault-tolerant replicated state machines, these misconceptions can lead to dramatic consequences, including asset losses in financial applications. In fact, several blockchains have been built to offer crash fault tolerant consensus, however, adapting them to work in an adversarial context is not trivial. It is thus crucial to debunk these myths to strengthen the security of future blockchain-based applications to be put in production. We argue that it is possible for application designers to rethink blockchains with built-in Byzantine fault tolerance to offer reasonable performance after identifying the right bottlenecks.

References

1. Abd-El-Malek, M., Ganger, G.R., Goodson, G.R., Reiter, M.K., Wylie, J.J.: Fault-scalable Byzantine fault-tolerant services. In: SOSP, pp. 59–74 (2005)

2. Amoussou-Guenou, Y., Pozzo, A.D., Potop-Butucaru, M., Tucci Piergiovanni, S.: Correctness of tendermint-core blockchains. In: OPODIS, vol. 125, pp. 16:1–16:16 (2018)
3. Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., De Caro, A., Enyeart, D., Ferris, C., Laventman, G., Manevich, Y., Muralidharan, S., Murthy, C., Nguyen, B., Sethi, M., Singh, G., Smith, K., Sorniotti, A., Stathakopoulou, C., Vukolić, M., Cocco, S., Yellick, J.: Hyperledger Fabric: A distributed operating system for permissioned blockchains. In: EuroSys (2018)
4. Antoniadis, K., Desjardins, A., Gramoli, V., Guerraoui, R., Zablotchi, I.: Leaderless consensus. In: ICDCS (2021)
5. Apostolaki, M., Zohar, A., Vanbever, L.: Hijacking bitcoin: Routing attacks on cryptocurrencies. In: S&P, pp. 375–392 (2017)
6. Armknecht, F., Karame, G.O., Mandal, A., Youssef, F., Zenner, E.: Ripple: Overview and outlook. In: Trust and Trustworthy Computing, pp. 163–180 (2015)
7. Aublin, P.L., Guerraoui, R., Knežević, N., Quéma, V., Vukolić, M.: The next 700 BFT protocols. ACM Trans. Comput. Syst. **32**(4), 12:1–12:45 (2015)
8. Bano, S., Baudet, M., Ching, A., Chursin, A., Danezis, G., Garillot, F., Li, Z., Malkhi, D., Naor, O., Perelman, D., Sonnino, A.: State machine replication in the Libra blockchain (2019). `https://developers.libra.org/docs/assets/papers/libra-consensus-state-machine-replication-in-the-libra-blockchain.pdf` (accessed 10/2019)
9. Berger, C., Reiser, H.P., Sousa, J., Bessani, A.: AWARE: Adaptive Wide-Area Replication for Fast and Resilient Byzantine Consensus. IEEE Transactions on Dependable and Secure Computing pp. 1–16 (2020). Early access
10. Bertrand, N., Gramoli, V., Konnov, I., Lazic, M., Tholoniat, P., Widder, J.: Compositional verification of byzantine consensus. Tech. Rep. 03158911, HAL (2021)
11. Bessani, A., Alchieri, E., Sousa, J.a., Oliveira, A., Pedone, F.: From Byzantine Replication to Blockchain: Consensus is only the Beginning. In: DSN (2020)
12. Bessani, A., Sousa, J., Alchieri, E.E.P.: State machine replication for the masses with BFT-SMART. In: DSN, pp. 355–362 (2014)
13. Bracha, G.: An asynchronous [(n - 1)/3]-resilient consensus protocol. In: PODC, pp. 154–162 (1984)
14. Brown, R.G., Carlyle, J., Grigg, I., Hearn, M.: Corda: An introduction. R3 CEV, August (2016)
15. Buterin, V., Griffith, V.: Casper the friendly finality gadget. Tech. Rep. 1710.09437v4, arXiv (2019)
16. Hyperledger caliper benchmarks (2020). `https://www.hyperledger.org/use/caliper`
17. Cachin, C., Kursawe, K., Petzold, F., Shoup, V.: Secure and efficient asynchronous broadcast protocols. In: CRYPTO, pp. 524–541 (2001)
18. Cachin, C., Vukolić, M.: Blockchain consensus protocols in the wild (keynote talk). In: DISC, pp. 1:1–1:16 (2017)
19. Campbell, R.: Transitioning to a Hyperledger Fabric quantum-resistant classical hybrid public key infrastructure. The Journal of The British Blockchain Association (2019)
20. Castro, M., Liskov, B.: Practical Byzantine fault tolerance and proactive recovery. ACM Trans. Comput. Syst. **20**(4), 398–461 (2002)
21. Clement, A., Wong, E., Alvisi, L., Dahlin, M., Marchetti, M.: Making Byzantine fault tolerant systems tolerate Byzantine faults. In: NSDI, pp. 153–168 (2009)
22. Coelho, P.R., Junior, T.C., Bessani, A., Dotti, F.L., Pedone, F.: Byzantine fault-tolerant atomic multicast. In: DSN, pp. 39–50 (2018)
23. Crain, T., Gramoli, V., Larrea, M., Raynal, M.: DBFT: Efficient leaderless Byzantine consensus and its applications to blockchains. In: NCA, pp. 1–8 (2018)
24. Crain, T., Natoli, C., Gramoli, V.: Red belly: A secure, fair and scalable open blockchain. In: S&P (2021)
25. Dinh, T.T.A., Wang, J., Chen, G., Liu, R., Ooi, B.C., Tan, K.L.: Blockbench: A framework for analyzing private blockchains. In: SIGMOD, pp. 1085–1100 (2017)
26. Doudou, A., Schiper, A.: Muteness detectors for consensus with byzantine processes. In: PODC, p. 315 (1998)

27. Dwork, C., Lynch, N., Stockmeyer, L.: Consensus in the presence of partial synchrony. J. ACM **35**(2), 288–323 (1988)
28. Dwork, C., Naor, M.: Pricing via processing or combatting junk mail. In: CRYPTO, vol. 740 (1993)
29. Ekparinya, P., Gramoli, V., Jourjon, G.: Impact of man-in-the-middle attacks on Ethereum. In: SRDS, pp. 11–20 (2018)
30. Ekparinya, P., Gramoli, V., Jourjon, G.: The Attack of the Clones against Proof-of-Authority. In: Proceedings of the Network and Distributed Systems Security Symposium (NDSS'20) (2020)
31. Eyal, I., Sirer, E.G.: Majority is not enough: Bitcoin mining is vulnerable. In: International conference on financial cryptography and data security, pp. 436–454. Springer (2014)
32. Gramoli, V.: On the danger of private blockchains. In: DCCL (2016)
33. Gramoli, V.: From blockchain consensus back to byzantine consensus. Future Generation of Computer Systems (2017)
34. Gramoli, V.: Blockchain scalability and its foundations in distributed systems (2021). Coursera MOOC
35. Gramoli, V., Guerraoui, R., Lebedev, A., Natoli, C., Voron, G.: Diablo: A benchmark suite for blockchains. In: EuroSys (2023)
36. Gramoli, V., Staples, M.: Blockchain standard: Can we reach consensus? IEEE Communications Standards Magazine (2018)
37. Gupta, D., Perronne, L., Bouchenak, S.: BFT-Bench: A framework to evaluate BFT protocols. In: ACM/SPEC ICPE, pp. 109–112 (2016)
38. Han, R., Shapiro, G., Gramoli, V., Xu, X.: On the performance of distributed ledgers for internet of things. Internet of Things **10** (2020)
39. Heilman, E., Kendler, A., Zohar, A., Goldberg, S.: Eclipse attacks on bitcoin's peer-to-peer network. In: USENIX Security, pp. 129–144 (2015)
40. Kemme, B., Bartoli, A., Babaoglu, O.: Online reconfiguration in replicated databases based on group communication. In: DSN, pp. 117–126. IEEE (2001)
41. Khahulin, P., Barinov, I., Baranov, V.: POA network white paper (2018). `https://github.com/poanetwork/wiki/wiki/POA-Network-Whitepaper`
42. Kotla, R., Alvisi, L., Dahlin, M., Clement, A., Wong, E.: Zyzzyva: Speculative Byzantine fault tolerance. In: SOSP, pp. 45–58 (2007)
43. Kwon, J.: Tendermint: Consensus without mining (2014)
44. Lamport, L., Shostak, R., Pease, M.: The Byzantine generals problem. ACM Trans. Program. Lang. Syst. **4**(3), 382–401 (1982)
45. Leal, F., Chis, A.E., González-Vélez, H.: Performance evaluation of private ethereum networks. SN Computer Science **1**(285) (2020)
46. Moniz, H.: The Istanbul BFT consensus algorithm. Tech. Rep. 2002.03613, arXiv (2020)
47. Mwaikambo, Z., Raj, A., Russell, R., Schopp, J., Vaddagiri, S.: Linux kernel hotplug CPU support. In: Linux Symposium, vol. 2 (2004)
48. Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system (2008)
49. Natoli, C., Yu, J., Gramoli, V., Esteves-Verissimo, P.: Deconstructing blockchains: A comprehensive survey on consensus, membership and structure. Tech. Rep. 1908.08316, arXiv (2019)
50. Neves, N.F., Correia, M., Verissimo, P.: Solving vector consensus with a wormhole. IEEE Trans. Parallel Distrib. Syst. **16**(12), 1120–1131 (2005)
51. Ongaro, D., Ousterhout, J.K.: In search of an understandable consensus algorithm. In: USENIX Annual Technical Conference, pp. 305–319 (2014)
52. Pease, M.C., Shostak, R.E., Lamport, L.: Reaching agreement in the presence of faults. J. ACM **27**(2), 228–234 (1980)
53. Pongnumkul, S., Siripanpornchana, C., Thajchayapong, S.: Performance analysis of private blockchain platforms in varying workloads. In: ICCCN, pp. 1–6 (2017)
54. Saltini, R., Hyland-Wood, D.: Correctness analysis of IBFT. Tech. Rep. 1901.07160, arXiv (2019)

55. Schwartz, D., Youngs, N., Britto, A.: The Ripple consensus protocol (2014). https://ripple.com/files/ripple_consensus_whitepaper.pdf
56. Shapiro, G., Natoli, C., Gramoli, V.: The performance of Byzantine fault tolerant blockchains. In: NCA (2020)
57. Sousa, J.a., Bessani, A., Vukolić, M.: A Byzantine fault-tolerant ordering service for the hyperledger fabric blockchain platform. In: DSN (2018)
58. Tendermint 0.10.2 (2017). White paper - https://jepsen.io/analyses/tendermint-0-10-2.pdf
59. Tennakoon, D., Hua, Y., Gramoli, V.: Smart Redbelly Blockchain: Reducing congestion for Web3. In: IPDPS (2023)
60. Thakkar, P., Nathan N, S., Vishwanathan, B.: Performance benchmarking and optimizing Hyperledger Fabric blockchain platform. In: MASCOTS (2018)
61. Tholoniat, P., Gramoli, V.: Formal verification of blockchain byzantine fault tolerance. In: FRIDA (2019)
62. Vizier, G., Gramoli, V.: ComChain: A blockchain with Byzantine fault tolerant reconfiguration. Concurrency and Computation, Practice and Experience **32**(12) (2019)
63. Voron, G., Gramoli, V.: DISPEL: Byzantine SMR with distributed pipelining. Tech. Rep. 1912.10367, arXiv (2019)
64. Vukolic, M.: The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication. In: IFIP WG 11.4 International Workshop on Open Problems in Network Security, pp. 112–125 (2015)
65. Weber, I., Gramoli, V., Ponomarev, A., Staples, M., Holz, R., Tran, A.B., Rimba, P.: On availability for blockchain-based systems. In: SRDS, pp. 64–73 (2017)
66. Yin, M., Malkhi, D., Reiter, M.K., Gueta, G.G., Abraham, I.: HotStuff: BFT consensus with linearity and responsiveness. In: PODC, pp. 347–356 (2019)
67. Zheng, P., Zheng, Z., Luo, X., Chen, X., Liu, X.: A detailed and real-time performance monitoring framework for blockchain systems. In: ICSE-SEIP, pp. 134–143 (2018)