Distributed Transaction Processing in Untrusted Environments

Mohammad Javad Amiri
Stony Brook University
Stony Brook, USA
amiri@cs.stonybrook.edu

Divyakant Agrawal
University of California Santa Barbara
Santa Barbara, USA
agrawal@cs.ucsb.edu

Amr El Abbadi
University of California Santa Barbara
Santa Barbara, USA
amr@cs.ucsb.edu

Boon Thau Loo
University of Pennsylvania
Philadelphia, USA
boonloo@seas.upenn.edu

ABSTRACT

Byzantine Fault-Tolerant (BFT) protocols have recently been extensively used by distributed and decentralized data management systems with non-trustworthy infrastructures to establish consensus on the order of transactions. BFT protocols cover a broad spectrum of design dimensions from infrastructure settings, such as the communication topology, to more technical features, such as commitment strategy and even fundamental social choice properties like order-fairness. The proliferation of different protocols has made it difficult to navigate the BFT landscape, let alone determine the protocol that best meets application needs. In this tutorial, we discuss BFT protocols that are used in modern large-scale data management systems, present a design space consisting of a set of design dimensions and explore several design choices that capture the trade-offs between different design space dimensions. The presented design space and its design choices will help developers analyze BFT protocols, understand how different protocols are related to each other, and find the protocol that best fits their needs.

CCS CONCEPTS

• **Information systems** → **Distributed database transactions**; • **Computer systems organization** → **Fault-tolerant network topologies**; • **Networks** → *Network protocol design.*

KEYWORDS

Distributed Transactions, Consensus, Byzantine Failure, Partial Synchrony, BFT protocols

ACM Reference Format:
Mohammad Javad Amiri, Divyakant Agrawal, Amr El Abbadi, and Boon Thau Loo. 2024. Distributed Transaction Processing in Untrusted Environments. In *Companion of the 2024 International Conference on Management of Data (SIGMOD-Companion '24), June 9–15, 2024, Santiago, AA, Chile.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3626246.3654684

1 INTRODUCTION

Distributed data management systems [37, 46, 50, 71, 79, 111, 145] rely on crash fault-tolerant protocols, e.g., Paxos [126] and Raft [153], to provide robustness and high availability and establish consensus on the order of transactions. However, today's large-scale distributed data management systems need to deal with untrustworthy environments where multiple mutually distrustful entities communicate with each other, and maintain data on untrusted infrastructure. By relying on Byzantine fault-tolerant (BFT) protocols, distributed databases have enabled a large class of applications ranging from contact tracing [156], crowdworking [22], supply chain assurance [24, 177], and federated learning [157].

Fault tolerance in large-scale systems is often achieved by replicating the data on multiple servers. The critical challenge is to execute all client transactions in the same order on all replicas. Formally, this approach is referred to as State Machine Replication (SMR) [125, 165] and BFT protocols are used to ensure that all non-faulty replicas execute all transactions in the same order despite f Byzantine (adversarial) servers. The ability to tolerate arbitrary failures makes BFT protocols a key component in various distributed data management systems with non-trustworthy infrastructures, e.g., permissioned blockchains [1–3, 18, 21, 23, 24, 27, 38, 54, 67, 97–99, 105, 124, 158, 164, 167, 184, 186], permissionless blockchains [51, 117, 119, 132, 190], distributed file systems [8, 61, 69], locking service [70], firewalls [44, 92, 93, 163, 173, 188], certificate authority systems [193], SCADA systems [35, 116, 152, 192], key-value datastores [43, 83, 96, 108, 163], and key management [137].

BFT SMR protocols differ along several dimensions, such as the number of replicas, processing strategy (i.e., optimistic, pessimistic, or robust), and the number of communication phases. While a large number of BFT protocols have been proposed [16, 25, 59, 101, 112, 120, 134, 189], there is no one-size-fits-all solution [185]. The performance trade-offs offered by BFT protocols vary significantly based on client workloads, network configurations, and application needs. Dependencies and trade-offs among different design dimensions of BFT protocols lead to several design choices. For example, protocols that reduce message complexity by increasing communication phases exhibit better throughput but worse latency (e.g., unsuitable for geo-replicated databases). In addition, adversarial behaviors in the system also affect the best-performing protocol choice. The lack of a clear "winner" among BFT protocols makes it difficult for application developers to choose one. It is, therefore, critical to

SIGMOD-Companion '24, June 9–15, 2024, Santiago, AA, Chile

ACM ISBN 979-8-4007-0422-2/24/06
https://doi.org/10.1145/3626246.3654684

study and analyze the various BFT protocols' design dimensions and their trade-offs in a unified manner.

Inspired by our Bedrock platform [26], this tutorial presents a unified framework to analyze partially synchronous SMR BFT protocols. We envision that this tutorial will provide an in-depth understanding of existing BFT protocols, highlight the trade-offs among dimensions, and will enable data management application designers to find the protocol that best fits their needs.

Our goal is to present to the database community an in-depth understanding of state-of-the-art solutions to design efficient BFT consensus protocols for large-scale fault-tolerant data management systems. We start with a design space to characterize BFT protocols based on different dimensions that capture the environmental settings, protocol structure, QoS features, and performance optimizations. Within the design space, we then discuss a set of design choices demonstrating trade-offs between different dimensions.

2 TUTORIAL OUTLINE

A BFT protocol runs on a network consisting of a set of nodes that may exhibit arbitrary, potentially malicious, behavior. BFT protocols use the State Machine Replication (SMR) algorithm [125, 165] where the system provides fault tolerance by replicating a service whose state is mirrored across different deterministic replicas. At a high level, the goal of a BFT SMR protocol is to assign each client transaction an order in the global service history and execute it in that order across all replicas[170]. In a BFT SMR protocol, all non-faulty replicas execute the same transactions in the same order (*safety*) and all correct transactions are eventually executed (*liveness*). In an asynchronous system, where replicas can fail, no consensus solutions guarantee both safety and liveness (FLP result) [89]. As a result, asynchronous consensus protocols rely on techniques such as randomization [41, 57, 91, 159], failure detectors [65, 135], hybridization/wormholes [72, 151] and partial synchrony [84, 85] to circumvent the FLP impossibility.

In this tutorial, we focus on the partial synchrony model as it is used in most practical BFT protocols [59, 101, 120, 189]. In the partial synchrony model, there exists an unknown global stabilization time (GST), after which all messages between correct replicas are received within some known bound Δ. BFT protocols follow several standard assumptions. First, while there is no upper bound on the number of faulty clients, the maximum number of concurrent malicious replicas is assumed to be f. Second, replicas are connected via an unreliable network that might drop, corrupt, or delay messages. Third, the network uses point-to-point bi-directional communication channels to connect replicas. Fourth, the failure of replicas is independent of each other, where a single fault does not lead to the failure of multiple replicas. This can be achieved by either diversifying replica implementation (e.g., n-version programming) [34, 90] or placing replicas at different geographic locations (e.g., datacenters) [42, 86, 172, 180]. Finally, a strong adversary can coordinate malicious replicas and delay communication. However, the adversary cannot subvert cryptographic assumptions.

2.1 Basics

BFT protocols structure. In a BFT protocol, as presented in Figure 1, clients communicate with a set of replicas that maintain a copy of the application state (i.e., database). A replica's lifecycle consists of ordering, execution, view-change, checkpointing, and recovery stages. The goal of **ordering** is to establish agreement on a unique order among requests executing on the application state. In leader-based consensus protocols, a designated *leader* replica proposes the order to all backup replicas and, to ensure fault tolerance, needs to get agreement from a subset of the replicas, referred to as a *quorum*. In the **execution** stage, requests are executed (i.e., applied to the replicated state machine). The **view-change** stage replaces the current leader due to failures. **Checkpointing** is used to garbage-collect data and enable trailing replicas to catch up, and finally, the **recovery** stage recovers replicas from faults.

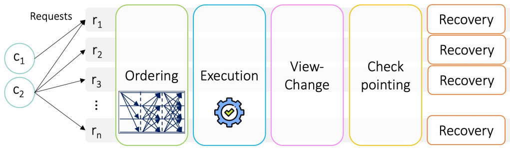

Figure 1: Different stages of replicas in a BFT protocol

The PBFT Protocol. To better illustrate the design space of BFT protocols, we give an overview of the PBFT protocol [59, 61] as a driving example. PBFT, as shown in Figure 2, is a leader-based protocol that operates in a succession of configurations called *views* [87, 88]. Each view is coordinated by a *stable* leader (primary), and the protocol *pessimistically* processes requests. In PBFT, the number of replicas, n, is at least $3f+1$ and the ordering stage consists of pre-prepare, prepare, and commit phases. The pre-prepare phase assigns an order to the request, the prepare phase guarantees the uniqueness of the assigned order, and the commit phase guarantees that the next leader can safely assign the order.

During a normal (no failure) case execution of PBFT, clients send their signed request messages (including the transaction to be executed) to the leader. In the pre-prepare phase, the leader assigns a sequence number to the request to determine the execution order of the request and multicasts a pre-prepare message to all *backups*. Upon receiving a valid pre-prepare message from the leader, each backup replica multicasts a prepare message to all replicas and waits for prepare messages from $2f$ different replicas (including the replica itself) that match the pre-prepare message. The goal of the prepare phase is to guarantee safety within the view, i.e., $2f$ replicas received matching pre-prepare messages from the leader replica and agree with the order of the request. Each replica then multicasts a commit message to all replicas. Once a replica receives $2f+1$ valid commit messages from different replicas, including itself, that match the pre-prepare message, it commits the request. The goal of the commit phase is to ensure safety across views, i.e., the request has been replicated on a majority of non-faulty replicas and can be recovered after (leader) failures. The second and third phases of PBFT follow the *clique* topology, i.e., have $O(n^2)$ message complexity. If the replica has executed all requests with lower sequence numbers, it executes the request and sends a reply to the client. The client waits for f+1 matching results from different replicas.

In the view change stage, upon detecting the failure of the leader of view v using timeouts, replicas exchange view-change messages including requests that have been received by the replicas. After receiving $2f+1$ view-change messages, the designated leader of view

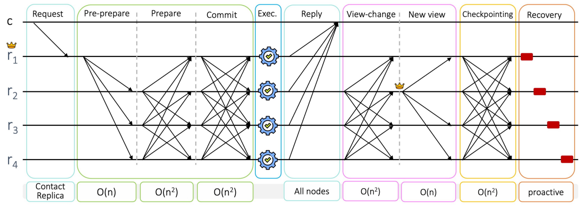

Figure 2: Different stages of PBFT protocol

$v + 1$ proposes a new view message, including the list of requests that should be processed in the new view.

In PBFT, replicas periodically generate checkpoint messages and send them to all replicas. If a replica receives $2f + 1$ matching checkpoint messages, the checkpoint is stable. PBFT includes a *proactive* recovery mechanism that periodically rejuvenates replicas one by one. PBFT uses either signatures [59] or MACs [61] for authentication. Using MACs, replicas need to send view-change-ack messages to the leader after receiving view-change messages. Since new view messages are not signed, these view-change-ack messages enable replicas to verify the authenticity of new view messages.

2.2 Design Space

Each BFT protocol can be analyzed along several dimensions. These dimensions (and values associated with each dimension) collectively help to define the overall design space of BFT protocols. The dimensions are categorized into four main families: *protocol structure* and *environmental settings* that present the core dimensions of BFT protocols, two optional *QoS features* including order-fairness and load balancing that a BFT protocol might support, and a set of *performance optimizations*, such as request pipelining, parallel execution, and trusted hardware, for tuning BFT protocols. In this tutorial, we focus on the first three families. In the rest of this section, we describe these families of dimensions in greater detail.

2.2.1 Protocol Structure.

P 1. **Commitment strategy.** BFT protocols process transactions in either an optimistic, pessimistic, or robust manner. *Optimistic* BFT protocols make optimistic assumptions on failures, synchrony, or data contention and might execute requests without necessarily establishing consensus. An optimistic BFT protocol might make a subset of the following assumptions:

- a_1. The leader is non-faulty, assigns a correct order to requests and sends it to all backups, e.g., Zyzzyva [120],
- a_2. The backups are non-faulty and *actively* and *honestly* participate in the protocol, e.g., CheapBFT [112],
- a_3. All non-leaf replicas in a tree topology are non-faulty, e.g., Kauri[149],
- a_4. The workload is conflict-free and concurrent requests update disjoint sets of data objects, e.g., Q/U [4],
- a_5. The clients are honest, e.g., Quorum [31], and
- a_6. The network is synchronous (in a time window), and messages are not lost or delayed, e.g., Tendermint [52].

Optimistic protocols are either *speculative* or *non-speculative*. In non-speculative protocols, e.g., CheapBFT [112] and SBFT [101], replicas execute a transaction only if the optimistic assumption holds. Speculative protocols, e.g., Zyzzyva [120] and PoE [103], on the other hand, optimistically execute transactions. If the assumption is not fulfilled, replicas might have to rollback the executed transactions. Optimistic BFT protocols improve performance in fault-free situations. If the assumption does not hold, the replicas, e.g., SBFT [101], or clients, e.g., Zyzzyva [120], detect the failure and use a fallback protocol. *Pessimistic* BFT protocols, on the other hand, do not make any optimistic assumptions about failures, synchrony, or data contention. In pessimistic BFT protocols, replicas communicate to agree on the order of requests. Finally, *robust* protocols, e.g., Prime [16], Aardvark [70], R-Aliph [31], Spinning [179] and RBFT [32], go one step further and consider scenarios where the system is under attack by a very strong adversary.

P 2. **Number of commitment phases.** The number of commitment (ordering) phases or *good-case latency* [7] of a BFT SMR protocol is the number of phases needed for all non-faulty replicas to commit when the leader is non-faulty, and the network is synchronous. We consider the number of commitment phases from the first time a replica (typically the leader) receives a request to the first time any participant (i.e., leader, backups, client) learns the commitment of the request, e.g., PBFT executes in 3 phases.

P 3. **View-change.** BFT protocols follow either the *stable leader* or the *rotating leader* mechanism to replace the current leader. The stable leader mechanism [59, 101, 120, 140] replaces the leader when the leader is suspected to be faulty by other replicas. In the rotating leader mechanism [13, 54, 62–64, 70, 95, 107, 118, 124, 179, 180, 189], the leader is replaced periodically, e.g., after a single attempt, insufficient performance, or an epoch (multiple requests).

Using the stable leader mechanism, the view-change stage becomes more complex. However, the routine is only executed when the leader is suspected to be faulty. On the other hand, the rotating leader mechanism requires ensuring view synchronization frequently (whenever the leader is rotated). Rotating the leader has several benefits, such as balancing load across replicas [39, 40, 179], improving resilience against slow replicas [70], and minimizing communication delays between clients and the leader [86, 139, 180].

P 4. **Checkpointing.** Checkpointing is used to first, garbage-collect data of completed consensus instances to save space and second, restore in-dark replicas (due to network unreliability or leader maliciousness) to ensure all non-faulty replicas are up-to-date [59, 80, 103]. Checkpointing is typically initiated after a fixed window in a decentralized manner without relying on a leader [59].

P 5. **Recovery.** When there are more than f failures, BFT protocols, apart from some exceptions [68, 130], completely fail and do not give any guarantees on their behavior [80]. BFT protocols perform recovery using *reactive* or *proactive* mechanisms (or a combination [173]). Reactive recovery mechanisms detect faulty replica behavior [106] and recover the replica by applying software rejuvenation techniques [76, 109] where the replica reboots, reestablishes its connection with other replicas and clients, and updates its state. On the other hand, proactive recovery mechanisms recover replicas in periodic time intervals. Proactive mechanisms do not require any fault detection techniques; however, they might unnecessarily recover non-faulty replicas [80]. During recovery, a replica is unavailable. A BFT protocol can rely on $3f + 2k + 1$ replicas to improve resilience and availability during recovery where k is the maximum number of servers that rejuvenate concurrently [173].

P 6. **Types of clients.** BFT protocols might have three types of clients: requester, proposer, and repairer. *Requester* clients perform a basic functionality and communicate with replicas by sending requests and receiving replies. A requester client may need to verify the results by waiting for a number of matching replies, e.g., f+1 in PBFT [59], $2f$+1 in PoE [103] and PBFT (for read-only requests) [61], or $3f$+1 is Zyzzyva [120]. Using trusted components, e.g., Troxy [129], or threshold signatures, e.g., SBFT [101], the client does not even need to wait for and verify multiple results from replicas. Clients might also play the *proposer* role by proposing a sequence number (acting as the leader) for its request [4, 100, 136, 138]. *Repairer* clients, on the other hand, detect the failure of replicas, e.g., Zyzzyva [120], and even change the protocol configuration, e.g., Scrooge[166], Abstract [31], and Q/U[4].

2.2.2 *Environmental Settings.*

E 1. **Number of replicas.** The first dimension concerns selecting BFT protocols based on the number of replicas used in a deployment. In the presence of f malicious failures, BFT protocols require at least $3f$+1 replicas to guarantee safety [47, 48, 74, 85, 127]. Using trusted hardware, the malicious behavior of replicas is restricted and safety can be guaranteed using $2f$+1 replicas [68, 73, 75, 161, 180, 180, 181]. Similarly, leveraging new hardware capabilities or using message-and-memory models the required number of replicas can be reduced to $2f + 1$ [9–11]. On the other hand, the number of communication phases can be reduced by increasing the number of replicas to $5f+1$ [140] (its proven lower bound, $5f - 1$ [7, 123]) or $7f + 1$ [171]. A BFT protocol might also optimistically assume the existence of a set of $2f + 1$ active non-faulty replicas, which participate in every quorum to establish consensus (and f passive replicas, which are informed about the decisions and become active if any active replica fails) [81, 112]. Using both trusted hardware and active/passive replication, the quorum size is further reduced to $f + 1$ during failure-free situations [81, 82, 112].

E 2. **Communication topology.** BFT protocols follow different communication topologies, including: (1) the star topology where communication is strictly from a designated replica, e.g., the leader, to all other replicas and vice-versa, resulting in linear message complexity [120, 189], (2) the clique topology where all (or a subset of) replicas communicate directly with each other (quadratic message complexity) [59], (3) the tree topology where the replicas are organized in a tree with the leader placed at the root, and at each phase, a replica communicates with either its child replicas or its parent replica (logarithmic message complexity) [117, 118, 149], or (4) the chain topology where replicas construct a pipeline and each replica communicates with its neighbor replicas [31].

E 3. **Authentication.** Participants authenticate their messages to enable other replicas to verify a message's origin. BFT protocols either use signatures, e.g., RSA [162], or authenticators [59], i.e., MACs [178]. Constant-sized threshold signatures [57, 168] have also been used to reduce the size of a set (quorum) of signatures. A protocol might even use different techniques (i.e., signatures, MACs) in different stages to authenticate messages sent by clients and sent by replicas in the ordering or view-change stage.

E 4. **Responsiveness, synchronization, and timers.** A BFT protocol is *responsive* if its normal case commit latency depends only on the actual network delay needed for replicas to process and exchange messages rather than any (usually much larger) predefined upper bound on message transmission delay [30, 154, 155, 169]. Responsiveness might be sacrificed in different ways. First, rotating the leader, the new leader might need to wait for a predefined time before initiating the next request to ensure that it receives the decided value from all non-faulty but slow replicas, e.g., Tendermint [124] and Casper [55]. Second, assuming all replicas are non-faulty, replicas (or clients) need to wait for a predefined time to receive messages of all replicas, e.g., SBFT [101] and Zyzzyva [120].

BFT protocols need to guarantee that all non-faulty replicas will eventually be synchronized to the same view with a non-faulty leader, thus enabling the leader to collect the decided values in previous views and making progress in the new view [49, 146, 147]. This is needed because a quorum of $2f + 1$ replicas might include f Byzantine replicas and the remaining f "slow" non-faulty replicas might stay behind (i.e., in-dark) and not even advance views at all. *View synchronization* can be achieved by integrating the functionality with the core consensus protocol, e.g., PBFT [59], or assigning a distinct synchronizer component, e.g., Pacemaker in HotStuff [189], and hardware clocks [5].

Depending on the environment, network characteristics, and processing strategy, a BFT protocol uses a subset of the following timers to ensure responsiveness and synchronization.

τ_1. Waiting for reply messages, e.g., Zyzzyva [120],
τ_2. Triggering (consecutive) view-change, e.g., PBFT [59],
τ_3. Detecting backup failures, e.g., SBFT [101],
τ_4. Quorum construction in an ordering phase, e.g., prevote and precommit timeouts in Tendermint [52],
τ_5. View synchronization, e.g., Tendermint [52],
τ_6. Finishing a (preordering) round, e.g., Themis [113],
τ_7. Performance check (heartbeat), e.g., Aardvark [70], and
τ_8. Atomic recovery (watchdog timer) to periodically hand control to a recovery monitor [60], e.g., PBFT [61].

2.2.3 *Quality of Service.*

Q 1. **Order-fairness.** Order-fairness deals with preventing adversarial manipulation of request ordering [36, 58, 113, 114, 121, 122, 191]. Order-fairness is defined as: "if a large number of replicas receives a request t_1 before another request t_2, then t_1 should be ordered before t_2" [114]. Order-fairness has been partially addressed using different techniques: (1) monitoring the leader to ensure it does not initiate two new requests from the same client before initiating an old request of another client, e.g., Aardvark [70], (2) adding a preordering phase, e.g., Prime [16], where replicas order the received requests locally and share their orderings with each other, (3) encrypting requests and revealing the contents only once their ordering is fixed [29, 56, 141, 174], (4) reputation-based systems [29, 78, 119, 128] to detect unfair censorship of specific client requests, and (5) providing opportunities for every replica to propose and commit its requests using fair election [6, 29, 95, 115, 128, 154, 187].

Q 2. **Load balancing.** The performance of fault-tolerant protocols is usually limited by the computing and bandwidth capacity of the leader [12, 14, 45, 66, 143, 144, 149, 183]. The leader coordinates the consensus protocol and multicasts/collects messages to all other replicas in different protocol phases. Load balancing is defined as distributing the load among the replicas of the system to balance the number of messages any single replica has to process.

Load balancing can be partially achieved using the rotating leader mechanism, multi-layer, or multi-leader protocols. Using leader rotation, one replica (leader) is still highly loaded in each consensus instance. In multi-layer protocols [17, 105, 131, 148, 150], the load is distributed between the leaders of different clusters. However, the system still suffers from load imbalance between the leader and backups in each cluster. In multi-leader protocols [15, 28, 33, 104, 175, 182], all replicas can initiate consensus to partially order requests in parallel. However, slow replicas still affect the global ordering of requests.

2.3 Design Choices Landscape

Given a set of specified dimension values in Section 2.2, each protocol represents a point in the design space. In this section, using PBFT and our design dimensions as a baseline, we illustrate a series of design choices that expose different trade-offs BFT protocols need to make. Each design choice acts as a one-to-one function that maps each valid input point (i.e., a protocol) to another valid output point in the design space.

Design Choice 1. ***(Linearization).*** This function explores a trade-off between communication topology and communication phases. The function takes a quadratic phase, e.g., prepare or commit in PBFT, and splits it into two linear phases: one phase from all replicas to a collector (typically the leader) and one phase from the collector to all replicas, e.g., SBFT [101], HotStuff [189] and HotStuff-2 [134]. The output protocol requires (threshold) signatures for authentication. The collector collects a quorum of (typically $n - f$) *signatures* from replicas and broadcasts its message, including the signatures, as a certificate of having received the required signatures. Using threshold signatures [56, 57, 160, 168] the collector message size becomes constant. Some BFT protocols [94, 110, 176] use linear communication during the ordering phase but follow the quadratic view-change routine of PBFT.

Design Choice 2. ***(Phase reduction through redundancy).*** This function explores a trade-off between the number of ordering phases and the number of replicas. The function transforms a protocol with $3f + 1$ replicas and 3 ordering phases (i.e., one linear, two quadratic), e.g., PBFT, to a fast protocol with $5f + 1$ replicas and 2 ordering phases (one linear, one quadratic), e.g., FaB [140]. In the second phase of the protocol, matching messages from a quorum of $4f + 1$ replicas are required. Recently, $5f - 1$ has been proven as the lower bound for two-step Byzantine consensus [7, 123]. The intuition behind the $5f - 1$ lower bound is that in an authenticated model, when replicas detect leader equivocation and initiate view-change, they do not include view-change messages coming from the malicious leader, reducing the maximum number of faulty messages to $f - 1$ [7, 123].

Design Choice 3. ***(Leader rotation).*** This function replaces the stable leader with the rotating leader mechanism, e.g., HotStuff [189], where the rotation happens after each request or epoch or due to low performance (as discussed in P **3**). This function eliminates the view-change stage and adds a quadratic phase or two linear phases (the linearization function) to the ordering stage to ensure that the new leader is aware of the correct state of the system.

Design Choice 4. ***(Non-responsive leader rotation).*** This function replaces the stable leader mechanism with the rotating leader mechanism *without* adding a new ordering phase (in contrast to design choice **3**) while sacrificing responsiveness. The new leader assumes that the network is synchronous (after GST) and waits for a predefined known upper bound Δ (Timer τ_5) before initiating the next request. This is needed to ensure that the new leader is aware of the highest assigned order to the requests, e.g., Tendermint [53, 124] and Casper [55]. As an optimization, if the new leader is aware of the highest assigned order (the leader was part of the quorum), it can initiate the next request right after receiving $2f + 1$ votes (without necessarily waiting for Δ [134]).

Design Choice 5. ***(Optimistic replica reduction).*** This function reduces the number of involved replicas in consensus from $3f+1$ to $2f+1$ while optimistically assuming all $2f+1$ replicas are non-faulty (assumption P **1**, a_2). In each phase of a BFT protocol, matching messages from a quorum of $2f + 1$ replicas is needed. If a quorum of $2f + 1$ non-faulty replicas is identified, they can order (and execute) requests without the participation of the remaining f replicas. Those f replicas remain passive and are needed if any of the active replicas become faulty [81, 112]. Note that n is still $3f + 1$.

Design Choice 6. ***(Optimistic phase reduction).*** Given a *linear* BFT protocol, this function optimistically eliminates two linear phases (i.e., the equivalence of a single quadratic prepare pahse) assuming all replicas are non-faulty, e.g., SBFT [101]. The leader (collector) waits for signed messages from all $3f + 1$ replicas in the second phase of ordering, combines signatures and sends a signed message to all replicas. Upon receiving the signed message from the leader, each replica ensures that all non-faulty replicas have received the request and agreed with the order. As a result, the third phase of communication can be omitted and replicas can directly commit the request. If the leader has not received $3f + 1$ messages after a predefined time (timer τ_3), the protocol fallbacks to its slow path and runs the third phase of ordering.

Design Choice 7. ***(Speculative phase reduction).*** This function, similar to the previous one, optimistically eliminates two linear phases of the ordering stage, assuming that non-faulty replicas can construct a quorum of responses, e.g., PoE [103]. The main difference is that the leader waits for signed messages from only $2f + 1$ replicas in the second phase of ordering and sends a signed message to all replicas. Upon receiving a message signed by $2f + 1$ replicas from the leader, each replica speculatively executes the transaction, optimistically assuming that either (1) all $2f + 1$ signatures are from non-faulty replicas or (2) at least $f + 1$ non-faulty replicas received the signed message from the leader. If (1) does not hold, other replicas receive and execute transactions during the view-change. However, if (2) does not hold, the replica might have to rollback the executed transaction.

Design Choice 8. ***(Speculative execution).*** This function eliminates the prepare and commit phases while optimistically assuming that all replicas are non-faulty (assumptions P **1**, a_1 and a_2), e.g., Zyzzyva [120]. Replicas speculatively execute transactions upon receiving them from the leader. If the client does not receive $3f + 1$ matching replies after a predefined time (timer τ_1) or it receives conflicting messages, the (repairer) client detects the failure and communicates with replicas to receive $2f + 1$ commit messages.

Design Choice 9. ***(Optimistic conflict-free).*** If requests of different clients are conflict-free (assumption P **1**, a_4), there is no need for a total order among all transactions. This function eliminates all ordering phases while optimistically assuming that requests are conflict-free and all replicas are non-faulty. The client becomes the *proposer* and sends its request to all (or a quorum of) replicas where replicas execute the requests without any communication [4, 77].

Design Choice 10. ***(Resilience).*** This function increases the number of replicas by $2f$, enabling the protocol to tolerate f more failure with the same safety guarantees. In particular, optimistic BFT protocols that assume all $3f + 1$ replicas are non-faulty (quorum size is also $3f + 1$) tolerate zero failures. By increasing the number of replicas to $5f + 1$ replicas, such BFT protocols can provide the same safety guarantees with quorums of size $4f + 1$ while tolerating f failures, e.g., Zyzzyva5 [120], Q/U [4]. Similarly, a protocol with the network size of $5f + 1$ can tolerate f more faulty replicas by increasing the network size to $7f + 1$ [171].

Design Choice 11. ***(Authentication).*** This function replaces MACs with signatures for a given stage. Signatures are typically more costly than MACs. However, in contrast to MACs, signatures provide non-repudiation and are not vulnerable to MAC-based attacks from malicious clients. If a protocol follows the star communication topology where a replica needs to include a quorum of signatures as a proof of its messages, e.g., HotStuff [189], k signatures can be replaced with a threshold signature. In such protocols, MACs cannot be used since MACs do not provide non-repudiation.

Design Choice 12. ***(Robust).*** This function makes a pessimistic protocol robust by adding a preordering stage to the protocol, e.g., Prime [16]. In the preordering stage and, upon receiving a request, each replica locally orders and broadcasts the request to all other replicas. All replicas then acknowledge the receipt of the request in an all-to-all communication phase and add the request to their local request vector. Replicas periodically share their vectors with each other. The robust function provides (partial) fairness as well. Robustness has also been addressed in other ways, e.g., using the leader rotation and a blacklisting mechanism in Spinning [179] or isolating the incoming traffic of different replicas, and checking the performance of the leader in Aardvark [70].

Design Choice 13. ***(Fair).*** This function transforms an unfair protocol, e.g., PBFT, into a fair protocol by adding a preordering phase to the protocol. In the preordering phase, clients send requests to all replicas, and once a round ends (timer τ_6), each replica sends a batch of requests in the received order to the leader. The leader then initiates consensus on the requests following the order of requests in the received batches. Depending on the order-fairness parameter γ ($0.5<\gamma\leq 1$) that defines the fraction of replicas receiving the requests in that specific order, at least $4f + 1$ replicas ($n>\frac{4f}{2\gamma-1}$) replicas are needed to provide order-fairness [113, 114] [1].

Design Choice 14. ***(Tree-based LoadBalancer).*** This function explores a trade-off between the communication topology and load balancing where load balancing is supported by organizing replicas in a tree topology, with the leader at the root, e.g., Kauri [149]. This function splits a linear communication phase into h phases where h is the tree's height and each replica uniformly communicates with its child/parent replicas in the tree. The protocol optimistically assumes all non-leaf replicas are non-faulty (assumption P **1**, a_3). Otherwise, the tree is reconfigured (i.e., view change).

3 TUTORIAL INFORMATION

This is a **three hours** tutorial targeting researchers, designers, and practitioners interested in consensus protocols and their applications in distributed transaction processing systems. The **target audience** with a basic background in distributed systems should benefit the most from this tutorial. For the general audience and newcomers, the tutorial explains the design space of consensus protocols in large-scale data management systems.

This tutorial differs from previous tutorials on the same topic in database conferences. The tutorial presented by Amiri et al. at ICDE 2020 [19] was mainly on a small subset of design dimensions, e.g., synchrony mode, failure model, and participant types. This tutorial focuses on partial synchrony protocols with the Byzantine failure model and explores many dimensions. This tutorial is also different from the tutorial presented by Gupta et al. [102] at VLDB 2020 where the focus of that tutorial was on designing consensus protocols for permissioned blockchains and the blockchain tutorials [20, 133, 142] presented in the DB community.

4 BIOGRAPHICAL SKETCHES

Mohammad Javad Amiri is an Assistant Professor in the Department of Computer Science at Stony Brook University. Before joining Stony Brook, he was a postdoctoral researcher in the Computer and Information Science department at the University of Pennsylvania. He received his Ph.D. in Computer Science at the University of California, Santa Barbara.

Divyakant Agrawal is a Distinguished Professor of Computer Science at the University of California at Santa Barbara. Over the course of his career, he has published more than 400 research articles and has mentored approximately 50 PhD students. He served as a journal editor for several database journals and is currently serving as the Chair of ACM Special Interest Group on Management of Data (SIGMOD). He is a Fellow of the ACM, the IEEE, and the AAAS.

Amr El Abbadi is a Distinguished Professor of Computer Science at the University of California, Santa Barbara. Prof. El Abbadi is an ACM Fellow, AAAS Fellow, and IEEE Fellow. He has served as a journal editor for several database journals and has been Program Chair for multiple databases and distributed systems conferences. He has published over 400 articles in databases and distributed systems and has supervised over 40 Ph.D. students.

Boon Thau Loo is an RCA Professor at the Computer and Information Science department with a secondary appointment in Electrical and Systems Engineering. He has graduated 16 Ph.D. students including five winners of dissertation awards. He received his Ph.D. degree in Computer Science from the University of California at Berkeley in 2006. He was awarded the 2006 David J. Sakrison Memorial Prize for the most outstanding dissertation research in the Department of EECS at the University of California-Berkeley, and the 2007 ACM SIGMOD Dissertation Award.

ACKNOWLEDGMENTS

This work is funded by NSF grant CNS-2104882.

---

[1] With $3f$+1 replicas, as shown in [113], order-fairness requires a synchronized clock [191] or does not provide censorship resistance [121].

REFERENCES

[1] [n. d.]. Chain. http://chain.com.

[2] [n. d.]. Corda. https://github.com/corda/corda.

[3] [n. d.]. Hyperledger Iroha. https://github.com/hyperledger/iroha.

[4] Michael Abd-El-Malek, Gregory R Ganger, Garth R Goodson, Michael K Reiter, and Jay J Wylie. 2005. Fault-scalable Byzantine fault-tolerant services. *Operating Systems Review (OSR)* 39, 5 (2005), 59–74.

[5] Ittai Abraham, Srinivas Devadas, Danny Dolev, Kartik Nayak, and Ling Ren. 2019. Synchronous Byzantine Agreement with Expected O(1) Rounds, Expected $O(n^2)$ Communication, and Optimal Resilience. In *Int. Conf. on Financial Cryptography and Data Security*. Springer, 320–334.

[6] Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Alexander Spiegelman. 2017. Solida: A Blockchain Protocol Based on Reconfigurable Byzantine Consensus. In *Int. Conf. on Principles of Distributed Systems (OPODIS)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[7] Ittai Abraham, Kartik Nayak, Ling Ren, and Zhuolun Xiang. 2021. Good-case Latency of Byzantine Broadcast: a Complete Categorization. In *Symposium on Principles of Distributed Computing (PODC)*. ACM, 331–341.

[8] Atul Adya, William J Bolosky, Miguel Castro, Gerald Cermak, Ronnie Chaiken, John R Douceur, Jon Howell, Jacob R Lorch, Marvin Theimer, and Roger P Wattenhofer. 2002. *FARSITE*: Federated, Available, and Reliable Storage for an Incompletely Trusted Environment. In *Symposium on Operating Systems Design and Implementation (OSDI)*. USENIX Association.

[9] Marcos K Aguilera, Naama Ben-David, Irina Calciu, Rachid Guerraoui, Erez Petrank, and Sam Toueg. 2018. Passing messages while sharing memory. In *Symposium on Principles of Distributed Computing (PODC)*. ACM, 51–60.

[10] Marcos K Aguilera, Naama Ben-David, Rachid Guerraoui, Virendra Marathe, and Igor Zablotchi. 2019. The impact of RDMA on agreement. In *Symposium on Principles of Distributed Computing (PODC)*. 409–418.

[11] Marcos K Aguilera, Naama Ben-David, Rachid Guerraoui, Dalia Papuc, Athanasios Xygkis, and Igor Zablotchi. 2021. Frugal Byzantine Computing. In *Int. Symposium on Distributed Computing*.

[12] Ailidani Ailijiang, Aleksey Charapko, and Murat Demirbas. 2019. Dissecting the performance of strongly-consistent replication protocols. In *SIGMOD Int. Conf. on Management of Data*. ACM, 1696–1710.

[13] Amitanand S Aiyer, Lorenzo Alvisi, Allen Clement, Mike Dahlin, Jean-Philippe Martin, and Carl Porth. 2005. BAR fault tolerance for cooperative services. In *Symposium on Operating Systems Principles (SOSP)*. ACM, 45–58.

[14] Nicolas Alhaddad, Sourav Das, Sisi Duan, Ling Ren, Mayank Varia, Zhuolun Xiang, and Haibin Zhang. 2022. Balanced byzantine reliable broadcast with near-optimal communication and improved computation. In *Symposium on Principles of Distributed Computing (PODC)*. ACM, 399–417.

[15] Salem Alqahtani and Murat Demirbas. 2021. BigBFT: A Multileader Byzantine Fault Tolerance Protocol for High Throughput. In *Int. Performance Computing and Communications Conf. (IPCCC)*. IEEE, 1–10.

[16] Yair Amir, Brian Coan, Jonathan Kirsch, and John Lane. 2011. Prime: Byzantine replication under attack. *Transactions on Dependable and Secure Computing* 8, 4 (2011), 564–577.

[17] Yair Amir, Claudiu Danilov, Danny Dolev, Jonathan Kirsch, John Lane, Cristina Nita-Rotaru, Josh Olsen, and David Zage. 2008. Steward: Scaling Byzantine fault-tolerant replication to wide area networks. *IEEE Transactions on Dependable and Secure Computing* 7, 1 (2008), 80–93.

[18] Mohammad Javad Amiri, Divyakant Agrawal, and Amr El Abbadi. 2019. CAPER: a cross-application permissioned blockchain. *Proc. of the VLDB Endowment* 12, 11 (2019), 1385–1398.

[19] Mohammad Javad Amiri, Divyakant Agrawal, and Amr El Abbadi. 2020. Modern Large-Scale Data Management Systems after 40 Years of Consensus. In *Int. Conf. on Data Engineering (ICDE)*. IEEE, 1794–1797.

[20] Mohammad Javad Amiri, Divyakant Agrawal, and Amr El Abbadi. 2021. Permissioned Blockchains: Properties, Techniques and Applications. In *SIGMOD Int. Conf. on Management of Data*. ACM, 2813–2820.

[21] Mohammad Javad Amiri, Divyakant Agrawal, and Amr El Abbadi. 2021. SharPer: Sharding Permissioned Blockchains Over Network Clusters. In *SIGMOD Int. Conf. on Management of Data*. ACM, 76–88.

[22] Mohammad Javad Amiri, Joris Duguépéroux, Tristan Allard, Divyakant Agrawal, and Amr El Abbadi. 2021. SEPAR: Towards Regulating Future of Work Multi-Platform Crowdworking Environments with Privacy Guarantees. In *Proceedings of The Web Conf. (WWW)*. 1891–1903.

[23] Mohammad Javad Amiri, Ziliang Lai, Liana Patel, Boon Thau Loo, Eric Lo, and Wenchao Zhou. 2023. Saguaro: An Edge Computing-Enabled Hierarchical Permissioned Blockchain. In *Int. Conf. on Data Engineering (ICDE)*. IEEE, 259–272.

[24] Mohammad Javad Amiri, Boon Thau Loo, Divyakant Agrawal, and Amr El Abbadi. 2022. Qanaat: A Scalable Multi-Enterprise Permissioned Blockchain System with Confidentiality Guarantees. *Proc. of the VLDB Endowment* 15, 11 (2022), 2839–2852.

[25] Mohammad Javad Amiri, Sujaya Maiyya, Divyakant Agrawal, and Amr El Abbadi. 2020. SeeMoRe: A fault-tolerant protocol for hybrid cloud environments. In *Int. Conf. on Data Engineering (ICDE)*. IEEE, 1345–1356.

[26] Mohammad Javad Amiri, Chenyuan Wu, Divyakant Agrawal, Amr El Abbadi, Boon Thau Loo, and Mohammad Sadoghi. 2024. The Bedrock of Byzantine Fault Tolerance: A Unified Platform for BFT Protocols Analysis, Implementation, and Experimentation. In *Symposium on Networked Systems Design and Implementation (NSDI)*. USENIX Association.

[27] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, and Yacov Manevich. 2018. Hyperledger Fabric: a distributed operating system for permissioned blockchains. In *European Conf. on Computer Systems (EuroSys)*. ACM, 30:1–30:15.

[28] Balaji Arun, Sebastiano Peluso, and Binoy Ravindran. 2019. ezbft: Decentralizing byzantine fault-tolerant state machine replication. In *Int. Conf. on Distributed Computing Systems (ICDCS)*. IEEE, 565–577.

[29] Avi Asayag, Gad Cohen, Ido Grayevsky, Maya Leshkowitz, Ori Rottenstreich, Ronen Tamari, and David Yakira. 2018. A fair consensus protocol for transaction ordering. In *Int. Conf. on Network Protocols (ICNP)*. IEEE, 55–65.

[30] Hagit Attiya, Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. 1994. Bounds on the time to reach agreement in the presence of timing uncertainty. *Journal of the ACM (JACM)* 41, 1 (1994), 122–152.

[31] Pierre-Louis Aublin, Rachid Guerraoui, Nikola Knežević, Vivien Quéma, and Marko Vukolić. 2015. The next 700 BFT protocols. *Transactions on Computer Systems (TOCS)* 32, 4 (2015), 12.

[32] Pierre-Louis Aublin, Sonia Ben Mokhtar, and Vivien Quéma. 2013. Rbft: Redundant byzantine fault tolerance. In *Int. Conf. on Distributed Computing Systems (ICDCS)*. IEEE, 297–306.

[33] Zeta Avarikioti, Lioba Heimbach, Roland Schmid, Laurent Vanbever, Roger Wattenhofer, and Patrick Wintermeyer. 2023. FnF-BFT: A BFT protocol with provable performance under attack. In *Int. Colloquium on Structural Information and Communication Complexity (SIROCCO)*. Springer, 165–198.

[34] Algirdas Avizienis. 1985. The N-version approach to fault-tolerant software. *IEEE Transactions on Software Engineering* 12 (1985), 1491–1501.

[35] Amy Babay, John Schultz, Thomas Tantillo, Samuel Beckley, Eamon Jordan, Kevin Ruddell, Kevin Jordan, and Yair Amir. 2019. Deploying intrusion-tolerant SCADA for the power grid. In *Int. Conf. on Dependable Systems and Networks (DSN)*. IEEE, 328–335.

[36] Leemon Baird. 2016. The swirlds hashgraph consensus algorithm: Fair, fast, byzantine fault tolerance. *Swirlds Tech Reports SWIRLDS-TR-2016-01, Tech. Rep* (2016).

[37] Jason Baker, Chris Bond, James C Corbett, JJ Furman, Andrey Khorlin, James Larson, Jean-Michel Leon, Yawei Li, Alexander Lloyd, and Vadim Yushprakh. 2011. Megastore: Providing scalable, highly available storage for interactive services. In *Conf. on Innovative Data Systems Research (CIDR)*.

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

[39] Johannes Behl, Tobias Distler, and Rüdiger Kapitza. 2015. Consensus-oriented parallelization: How to earn your first million. In *Annual Middleware Conf. (Middleware)*. 173–184.

[40] Johannes Behl, Tobias Distler, and Rüdiger Kapitza. 2017. Hybrids on steroids: SGX-based high performance BFT. In *European Conf. on Computer Systems (EuroSys)*. 222–237.

[41] Michael Ben-Or. 1983. Another Advantage of Free Choice: Completely Asynchronous Agreement Protocols (Extended Abstract).. In *Symposium on Principles of Distributed Computing (PODC)*. ACM, 27–30.

[42] Christian Berger, Hans P Reiser, João Sousa, and Alysson Bessani. 2019. Resilient wide-area Byzantine consensus using adaptive weighted replication. In *Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 183–18309.

[43] Alysson Bessani, Miguel Correia, Bruno Quaresma, Fernando André, and Paulo Sousa. 2013. DepSky: dependable and secure storage in a cloud-of-clouds. *Transactions on Storage (TOS)* 9, 4 (2013), 12.

[44] Alysson Neves Bessani, Paulo Sousa, Miguel Correia, Nuno Ferreira Neves, and Paulo Verissimo. 2008. The CRUTIAL way of critical infrastructure protection. *IEEE Security & Privacy* 6, 6 (2008), 44–51.

[45] Martin Biely, Zarko Milosevic, Nuno Santos, and Andre Schiper. 2012. S-paxos: Offloading the leader for high throughput state machine replication. In *Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 111–120.

[46] Kenneth P Birman, Thomas A Joseph, Thomas Raeuchle, and Amr El Abbadi. 1985. Implementing fault-tolerant distributed objects. *Trans. on Software Engineering* 6 (1985), 502–508.

[47] Gabriel Bracha. 1984. An asynchronous [(n-1)/3]-resilient consensus protocol. In *Symposium on Principles of Distributed Computing (PODC)*. ACM, 154–162.

[48] Gabriel Bracha and Sam Toueg. 1985. Asynchronous consensus and broadcast protocols. *Journal of the ACM (JACM)* 32, 4 (1985), 824–840.

[49] Manuel Bravo, Gregory Chockler, and Alexey Gotsman. 2020. Making Byzantine consensus live. In *Int. Symposium on Distributed Computing (DISC)*. Schloss

Dagstuhl-Leibniz-Zentrum für Informatik.

[50] Nathan Bronson, Zach Amsden, George Cabrera, Prasad Chakka, Peter Dimov, Hui Ding, Jack Ferris, Anthony Giardullo, Sachin Kulkarni, and Harry Li. 2013. TAO: Facebook's Distributed Data Store for the Social Graph. In *Annual Technical Conf. (ATC)*. USENIX Association, 49–60.

[51] Richard Gendal Brown, James Carlyle, Ian Grigg, and Mike Hearn. 2016. Corda: an introduction. *R3 CEV, August* 1, 15 (2016), 14.

[52] Ethan Buchman. 2016. *Tendermint: Byzantine fault tolerance in the age of blockchains*. Ph. D. Dissertation.

[53] Ethan Buchman, Jae Kwon, and Zarko Milosevic. 2018. The latest gossip on BFT consensus. *arXiv preprint arXiv:1807.04938* (2018).

[54] Yehonatan Buchnik and Roy Friedman. 2020. FireLedger: a high throughput blockchain consensus protocol. *Proceedings of the VLDB Endowment* 13, 9 (2020), 1525–1539.

[55] Vitalik Buterin and Virgil Griffith. 2017. Casper the friendly finality gadget. *arXiv preprint arXiv:1710.09437* (2017).

[56] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. 2001. Secure and efficient asynchronous broadcast protocols. In *Annual Int. Cryptology Conf.* Springer, 524–541.

[57] Christian Cachin, Klaus Kursawe, and Victor Shoup. 2005. Random oracles in Constantinople: Practical asynchronous Byzantine agreement using cryptography. *Journal of Cryptology* 18, 3 (2005), 219–246.

[58] Christian Cachin, Jovana Mićić, and Nathalie Steinhauer. 2022. Quick Order Fairness. In *Int. Conf. on Financial Cryptography and Data Security (FC)*. Springer, 1–18.

[59] Miguel Castro and Barbara Liskov. 1999. Practical Byzantine fault tolerance. In *Symposium on Operating Systems Design and Implementation (OSDI)*. USENIX Association, 173–186.

[60] Miguel Castro and Barbara Liskov. 2000. Proactive Recovery in a Byzantine-Fault-Tolerant System. In *Symposium on Operating Systems Design and Implementation (OSDI)*. USENIX Association.

[61] Miguel Castro and Barbara Liskov. 2002. Practical Byzantine fault tolerance and proactive recovery. *Transactions on Computer Systems (TOCS)* 20, 4 (2002), 398–461.

[62] Benjamin Y Chan and Elaine Shi. 2020. Streamlet: Textbook streamlined blockchains. In *Conf. on Advances in Financial Technologies (AFT)*. ACM, 1–11.

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

[64] TH Hubert Chan, Rafael Pass, and Elaine Shi. 2018. PiLi: An Extremely Simple Synchronous Blockchain. *Cryptology ePrint Archive* (2018).

[65] Tushar Deepak Chandra and Sam Toueg. 1996. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM (JACM)* 43, 2 (1996), 225–267.

[66] Aleksey Charapko, Ailidani Ailijiang, and Murat Demirbas. 2021. PigPaxos: Devouring the communication bottlenecks in distributed consensus. In *SIGMOD Int. Conf. on Management of Data*. ACM, 235–247.

[67] JP Morgan Chase. 2016. Quorum white paper.

[68] Byung-Gon Chun, Petros Maniatis, Scott Shenker, and John Kubiatowicz. 2007. Attested append-only memory: Making adversaries stick to their word. In *Operating Systems Review (OSR)*, Vol. 41-6. ACM SIGOPS, 189–204.

[69] Allen Clement, Manos Kapritsos, Sangmin Lee, Yang Wang, Lorenzo Alvisi, Mike Dahlin, and Taylor Riche. 2009. Upright cluster services. In *Symposium on Operating Systems Principles (SOSP)*. ACM, 277–290.

[70] Allen Clement, Edmund L Wong, Lorenzo Alvisi, Michael Dahlin, and Mirco Marchetti. 2009. Making Byzantine Fault Tolerant Systems Tolerate Byzantine Faults.. In *Symposium on Networked Systems Design and Implementation (NSDI)*, Vol. 9. USENIX Association, 153–168.

[71] James C Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, Jeffrey John Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, and Peter Hochschild. 2013. Spanner: Google's globally distributed database. *Transactions on Computer Systems (TOCS)* 31, 3 (2013), 8.

[72] Miguel Correia, Nuno Ferreira Neves, Lau Cheuk Lung, and Paulo Veríssimo. 2005. Low complexity Byzantine-resilient consensus. *Distributed Computing* 17, 3 (2005), 237–249.

[73] Miguel Correia, Nuno Ferreira Neves, and Paulo Verissimo. 2004. How to tolerate half less one Byzantine nodes in practical distributed systems. In *Int. Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 174–183.

[74] Miguel Correia, Nuno Ferreira Neves, and Paulo Veríssimo. 2006. From consensus to atomic broadcast: Time-free Byzantine-resistant protocols without signatures. *Comput. J.* 49, 1 (2006), 82–96.

[75] Miguel Correia, Nuno Ferreira Neves, and Paulo Verissimo. 2013. Bft-to: Intrusion tolerance with less replicas. *Comput. J.* 56, 6 (2013), 693–715.

[76] Domenico Cotroneo, Roberto Natella, Roberto Pietrantuono, and Stefano Russo. 2014. A survey of software aging and rejuvenation studies. *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 10, 1 (2014), 1–34.

[77] James Cowling, Daniel Myers, Barbara Liskov, Rodrigo Rodrigues, and Liuba Shrira. 2006. HQ replication: A hybrid quorum protocol for Byzantine fault tolerance. In *Symposium on Operating Systems Design and Implementation (OSDI)*. USENIX Association, 177–190.

[78] Tyler Crain, Christopher Natoli, and Vincent Gramoli. 2021. Red Belly: a secure, fair and scalable open blockchain. In *Symposium on Security and Privacy (SP)*. IEEE.

[79] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. 2007. Dynamo: Amazon's highly available key-value store. *Operating Systems Review (OSR)* 41, 6 (2007), 205–220.

[80] Tobias Distler. 2021. Byzantine fault-tolerant state-machine replication from a systems perspective. *ACM Computing Surveys (CSUR)* 54, 1 (2021), 1–38.

[81] Tobias Distler, Christian Cachin, and Rüdiger Kapitza. 2016. Resource-efficient Byzantine fault tolerance. *Transactions on Computers* 65, 9 (2016), 2807–2819.

[82] Tobias Distler, Ivan Popov, Wolfgang Schröder-Preikschat, Hans P Reiser, and Rüdiger Kapitza. 2011. SPARE: Replicas on Hold. In *Network and Distributed System Security Symposium (NDSS)*.

[83] Dan Dobre, Ghassan Karame, Wenting Li, Matthias Majuntke, Neeraj Suri, and Marko Vukolić. 2013. PoWerStore: Proofs of writing for efficient and robust storage. In *Conf. on Computer and communications security (CCS)*. ACM, 285–298.

[84] Danny Dolev, Cynthia Dwork, and Larry Stockmeyer. 1987. On the minimal synchronism needed for distributed consensus. *Journal of the ACM (JACM)* 34, 1 (1987), 77–97.

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

[86] Michael Eischer and Tobias Distler. 2018. Latency-aware leader selection for geo-replicated Byzantine fault-tolerant systems. In *Int. Conf. on Dependable Systems and Networks Workshops (DSN-W)*. IEEE, 140–145.

[87] Amr El Abbadi, Dale Skeen, and Flaviu Cristian. 1985. An efficient, fault-tolerant protocol for replicated data management. In *SIGACT-SIGMOD symposium on Principles of database systems*. ACM, 215–229.

[88] Amr El Abbadi and Sam Toueg. 1986. Availability in partitioned replicated databases. In *SIGACT-SIGMOD symposium on Principles of database systems*. ACM, 240–251.

[89] Michael J Fischer, Nancy A Lynch, and Michael S Paterson. 1985. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)* 32, 2 (1985), 374–382.

[90] Stephanie Forrest, Anil Somayaji, and David H Ackley. 1997. Building diverse computer systems. In *Workshop on Hot Topics in Operating Systems*. IEEE, 67–72.

[91] Yingzi Gao, Yuan Lu, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. 2022. Efficient asynchronous byzantine agreement without private setups. In *Int. Conf. on Distributed Computing Systems (ICDCS)*. IEEE, 246–257.

[92] Miguel Garcia, Nuno Neves, and Alysson Bessani. 2013. An intrusion-tolerant firewall design for protecting SIEM systems. In *Conf. on Dependable Systems and Networks Workshop (DSN-W)*. IEEE, 1–7.

[93] Miguel Garcia, Nuno Neves, and Alysson Bessani. 2016. SieveQ: A layered bft protection system for critical services. *IEEE Transactions on Dependable and Secure Computing* 15, 3 (2016), 511–525.

[94] Rati Gelashvili, Lefteris Kokoris-Kogias, Alberto Sonnino, Alexander Spiegelman, and Zhuolun Xiang. 2022. Jolteon and ditto: Network-adaptive efficient consensus with asynchronous fallback. In *Int. Conf. on Financial Cryptography and Data Security*. Springer, 296–315.

[95] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. 2017. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Symposium on Operating Systems Principles (SOSP)*. ACM, 51–68.

[96] Garth R Goodson, Jay J Wylie, Gregory R Ganger, and Michael K Reiter. 2004. Efficient Byzantine-tolerant erasure-coded storage. In *Int. Conf. on Dependable Systems and Networks (DSN)*. IEEE, 135–144.

[97] Christian Gorenflo, Lukasz Golab, and Srinivasan Keshav. 2020. XOX Fabric: A hybrid approach to transaction execution. In *Int. Conf. on Blockchain and Cryptocurrency (ICBC)*. IEEE, 1–9.

[98] Christian Gorenflo, Stephen Lee, Lukasz Golab, and Srinivasan Keshav. 2019. Fastfabric: Scaling hyperledger fabric to 20,000 transactions per second. In *Int. Conf. on Blockchain and Cryptocurrency (ICBC)*. IEEE, 455–463.

[99] Gideon Greenspan. 2015. MultiChain private blockchain-White paper. *URl: http://www. multichain. com/download/MultiChain-White-Paper. pdf* (2015).

[100] Rachid Guerraoui, Nikola Knežević, Vivien Quéma, and Marko Vukolić. 2010. The next 700 BFT protocols. In *European conf. on Computer systems (EuroSys)*. ACM, 363–376.

[101] Guy Golan Gueta, Ittai Abraham, Shelly Grossman, Dahlia Malkhi, Benny Pinkas, Michael K Reiter, Dragos-Adrian Seredinschi, Orr Tamir, and Alin Tomescu. 2019. SBFT: a Scalable Decentralized Trust Infrastructure for Blockchains. In *Int. Conf. on Dependable Systems and Networks (DSN)*. IEEE/IFIP, 568–580.

[102] Suyash Gupta, Jelle Hellings, Sajjad Rahnama, and Mohammad Sadoghi. 2020. Building high throughput permissioned blockchain fabrics: challenges and opportunities. *Proc. of the VLDB Endowment* 13, 12 (2020), 3441–3444.

[103] Suyash Gupta, Jelle Hellings, Sajjad Rahnama, and Mohammad Sadoghi. 2021. Proof-of-execution: Reaching consensus through fault-tolerant speculation. In *Int. Conf. on Extending Database Technology (EDBT)*. 301–312.

[104] Suyash Gupta, Jelle Hellings, and Mohammad Sadoghi. 2021. Rcc: Resilient concurrent consensus for high-throughput secure transaction processing. In *Int. Conf. on Data Engineering (ICDE)*. IEEE, 1392–1403.

[105] Suyash Gupta, Sajjad Rahnama, Jelle Hellings, and Mohammad Sadoghi. 2020. ResilientDB: Global Scale Resilient Blockchain Fabric. *Proceedings of the VLDB Endowment* 13, 6 (2020), 868–883.

[106] Andreas Haeberlen, Petr Kouznetsov, and Peter Druschel. 2006. The Case for Byzantine Fault Detection.. In *HotDep*.

[107] Timo Hanke, Mahnush Movahedi, and Dominic Williams. 2018. Dfinity technology overview series, consensus system. *arXiv preprint arXiv:1805.04548* (2018).

[108] James Hendricks, Gregory R Ganger, and Michael K Reiter. 2007. Low-overhead byzantine fault-tolerant storage. *ACM SIGOPS Operating Systems Review* 41, 6 (2007), 73–86.

[109] Yennun Huang, Chandra Kintala, Nick Kolettis, and N Dudley Fulton. 1995. Software rejuvenation: Analysis, module and applications. In *Int. Symposium on fault-tolerant computing. Digest of papers*. IEEE, 381–390.

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

[111] Robert Kallman, Hideaki Kimura, Jonathan Natkins, Andrew Pavlo, Alexander Rasin, Stanley Zdonik, Evan PC Jones, Samuel Madden, Michael Stonebraker, and Yang Zhang. 2008. H-store: a high-performance, distributed main memory transaction processing system. *Proc. of the VLDB Endowment* 1, 2 (2008), 1496–1499.

[112] Rüdiger Kapitza, Johannes Behl, Christian Cachin, Tobias Distler, Simon Kuhnle, Seyed Vahid Mohammadi, Wolfgang Schröder-Preikschat, and Klaus Stengel. 2012. CheapBFT: resource-efficient byzantine fault tolerance. In *European Conf. on Computer Systems (EuroSys)*. ACM, 295–308.

[113] Mahimna Kelkar, Soubhik Deb, Sishan Long, Ari Juels, and Sreeram Kannan. 2022. Themis: Fast, Strong Order-Fairness in Byzantine Consensus. *The Science of Blockchain Conf. (SBC)* (2022).

[114] Mahimna Kelkar, Fan Zhang, Steven Goldfeder, and Ari Juels. 2020. Order-fairness for byzantine consensus. In *Annual Int. Cryptology Conf.* Springer, 451–480.

[115] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. 2017. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Annual Int. Cryptology Conf.* Springer, 357–388.

[116] Jonathan Kirsch, Stuart Goose, Yair Amir, Dong Wei, and Paul Skare. 2013. Survivable SCADA via intrusion-tolerant replication. *IEEE Transactions on Smart Grid* 5, 1 (2013), 60–70.

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

[118] Eleftherios Kokoris-Kogias. [n. d.]. Robust and Scalable Consensus for Sharded Distributed Ledgers. ([n. d.]).

[119] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. 2018. Omniledger: A secure, scale-out, decentralized ledger via sharding. In *Symposium on Security and Privacy (SP)*. IEEE, 583–598.

[120] Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong. 2007. Zyzzyva: speculative byzantine fault tolerance. *Operating Systems Review (OSR)* 41, 6 (2007), 45–58.

[121] Klaus Kursawe. 2020. Wendy, the good little fairness widget: Achieving order fairness for blockchains. In *Conf. on Advances in Financial Technologies (AFT)*. ACM, 25–36.

[122] Klaus Kursawe. 2021. Wendy Grows Up: More Order Fairness. In *Int. Conf. on Financial Cryptography and Data Security (FC)*. Springer, 191–196.

[123] Petr Kuznetsov, Andrei Tonkikh, and Yan X Zhang. 2021. Revisiting Optimal Resilience of Fast Byzantine Consensus. In *Symposium on Principles of Distributed Computing (PODC)*. ACM, 343–353.

[124] Jae Kwon. 2014. Tendermint: Consensus without mining. (2014).

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

[126] Leslie Lamport. 2001. Paxos made simple. *ACM Sigact News* 32, 4 (2001), 18–25.

[127] Leslie Lamport, Robert Shostak, and Marshall Pease. 1982. The Byzantine generals problem. *Transactions on Programming Languages and Systems (TOPLAS)* 4, 3 (1982), 382–401.

[128] Kfir Lev-Ari, Alexander Spiegelman, Idit Keidar, and Dahlia Malkhi. 2019. FairLedger: A Fair Blockchain Protocol for Financial Institutions. In *Int. Conf. on Principles of Distributed Systems (OPODIS)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[129] Bijun Li, Nico Weichbrodt, Johannes Behl, Pierre-Louis Aublin, Tobias Distler, and Rüdiger Kapitza. 2018. Troxy: Transparent access to byzantine fault-tolerant systems. In *Int. Conf. on Dependable Systems and Networks (DSN)*. IEEE, 59–70.

[130] Jinyuan Li and David Maziéres. 2007. Beyond One-Third Faulty Replicas in Byzantine Fault Tolerant Systems.. In *Symposium on Networked Systems Design and Implementation (NSDI)*. USENIX Association.

[131] Wenyu Li, Chenglin Feng, Lei Zhang, Hao Xu, Bin Cao, and Muhammad Ali Imran. 2020. A scalable multi-layer PBFT consensus for blockchain. *Transactions on Parallel and Distributed Systems* 32, 5 (2020), 1146–1160.

[132] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. 2016. A secure sharding protocol for open blockchains. In *SIGSAC Conf. on Computer and Communications Security (CCS)*. ACM, 17–30.

[133] Sujaya Maiyya, Victor Zakhary, Mohammad Javad Amiri, Divyakant Agrawal, and Amr El Abbadi. 2019. Database and distributed computing foundations of blockchains. In *SIGMOD Int. Conf. on Management of Data*. ACM, 2036–2041.

[134] Dahlia Malkhi and Kartik Nayak. 2023. HotStuff-2: Optimal Two-Phase Responsive BFT. *Cryptology ePrint Archive* (2023).

[135] Dahlia Malkhi and Michael Reiter. 1997. Unreliable intrusion detection in distributed computations. In *Computer Security Foundations Workshop*. IEEE, 116–124.

[136] Dahlia Malkhi and Michael Reiter. 1998. Byzantine quorum systems. *Distributed computing* 11, 4 (1998), 203–213.

[137] Dahlia Malkhi and Michael K Reiter. 1998. Secure and scalable replication in Phalanx. In *Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 51–58.

[138] Dahlia Malkhi and Michael K Reiter. 1998. Survivable consensus objects. In *Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 271–279.

[139] Yanhua Mao, Flavio P Junqueira, and Keith Marzullo. 2009. Towards low latency state machine replication for uncivil wide-area networks. In *Workshop on Hot Topics in System Dependability*. Citeseer.

[140] J-P Martin and Lorenzo Alvisi. 2006. Fast byzantine consensus. *Transactions on Dependable and Secure Computing* 3, 3 (2006), 202–215.

[141] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. 2016. The honey badger of BFT protocols. In *Conf. on Computer and Communications Security (CCS)*. ACM, 31–42.

[142] C Mohan. 2018. Blockchains and databases: A new era in distributed computing. In *Int. Conf. on data engineering (ICDE)*. IEEE, 1739–1740.

[143] Iulian Moraru, David G Andersen, and Michael Kaminsky. 2012. Egalitarian paxos. In *Symposium on Operating Systems Principles (SOSP)*. ACM.

[144] Iulian Moraru, David G Andersen, and Michael Kaminsky. 2013. There is more consensus in egalitarian parliaments. In *Symposium on Operating Systems Principles (SOSP)*. ACM, 358–372.

[145] Louise E Moser, Peter M Melliar-Smith, Priya Narasimhan, Lauren A Tewksbury, and Vana Kalogeraki. 1999. The Eternal system: An architecture for enterprise applications. In *Int. Enterprise Distributed Object Computing Conf. (EDOC)*. IEEE, 214–222.

[146] Oded Naor, Mathieu Baudet, Dahlia Malkhi, and Alexander Spiegelman. 2019. Cogsworth: Byzantine view synchronization. *arXiv preprint arXiv:1909.05204* (2019).

[147] Oded Naor and Idit Keidar. 2020. Expected Linear Round Synchronization: The Missing Link for Linear Byzantine SMR. In *Int. Symposium on Distributed Computing (DISC)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

[148] Faisal Nawab and Mohammad Sadoghi. 2019. Blockplane: A global-scale byzantizing middleware. In *Int. Conf. on Data Engineering (ICDE)*. IEEE, 124–135.

[149] Ray Neiheiser, Miguel Matos, and Luís Rodrigues. 2021. Kauri: Scalable BFT Consensus with Pipelined Tree-Based Dissemination and Aggregation. In *Symposium on Operating Systems Principles (SOSP)*. ACM, 35–48.

[150] Ray Neiheiser, Daniel Presser, Luciana Rech, Manuel Bravo, Luís Rodrigues, and Miguel Correia. 2018. Fireplug: Flexible and robust n-version geo-replication of graph databases. In *Int. Conf. on Information Networking (ICOIN)*. IEEE, 110–115.

[151] Nuno Ferreira Neves, Miguel Correia, and Paulo Verissimo. 2005. Solving vector consensus with a wormhole. *IEEE Transactions on Parallel and Distributed Systems* 16, 12 (2005), 1120–1131.

[152] André Nogueira, Miguel Garcia, Alysson Bessani, and Nuno Neves. 2018. On the challenges of building a BFT SCADA. In *Int. Conf. on Dependable Systems and Networks (DSN)*. IEEE, 163–170.

[153] Diego Ongaro and John K Ousterhout. 2014. In search of an understandable consensus algorithm. In *Annual Technical Conf. (ATC)*. USENIX Association, 305–319.

[154] Rafael Pass and Elaine Shi. 2017. Hybrid Consensus: Efficient Consensus in the Permissionless Model. In *Int.Symposium on Distributed Computing (DISC)*. 6.

[155] Rafael Pass and Elaine Shi. 2018. Thunderella: Blockchains with optimistic instant confirmation. In *Annual Int. Conf. on the Theory and Applications of Cryptographic Techniques*. Springer, 3–33.

[156] Zhe Peng, Cheng Xu, Haixin Wang, Jinbin Huang, Jianliang Xu, and Xiaowen Chu. 2021. P2B-Trace: Privacy-Preserving Blockchain-based Contact Tracing to Combat Pandemics. In *SIGMOD Int. Conf. on Management of Data*. ACM, 2389–2393.

[157] Zhe Peng, Jianliang Xu, Xiaowen Chu, Shang Gao, Yuan Yao, Rong Gu, and Yuzhe Tang. 2021. Vfchain: Enabling verifiable and auditable federated learning via blockchain systems. *IEEE Transactions on Network Science and Engineering* (2021).

[158] Ji Qi, Xusheng Chen, Yunpeng Jiang, Jianyu Jiang, Tianxiang Shen, Shixiong Zhao, Sen Wang, Gong Zhang, Li Chen, Man Ho Au, et al. 2021. Bidl: A High-throughput, Low-latency Permissioned Blockchain Framework for Datacenter

Networks. In *Symposium on Operating Systems Principles (SOSP)*. ACM, 18–34.

[159] Michael O Rabin. 1983. Randomized byzantine generals. In *Symposium on Foundations of Computer Science (SFCS)*. IEEE, 403–409.

[160] HariGovind V Ramasamy and Christian Cachin. 2005. Parsimonious asynchronous byzantine-fault-tolerant atomic broadcast. In *Int. Conf. On Principles Of Distributed Systems (OPODIS)*. Springer, 88–102.

[161] Hans P Reiser and Rudiger Kapitza. 2007. Hypervisor-based efficient proactive recovery. In *Int. Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 83–92.

[162] Ronald L Rivest, Adi Shamir, and Leonard M Adleman. 2019. *A method for obtaining digital signatures and public key cryptosystems.* Routledge.

[163] Tom Roeder and Fred B Schneider. 2010. Proactive obfuscation. *ACM Transactions on Computer Systems (TOCS)* 28, 2 (2010), 1–54.

[164] Pingcheng Ruan, Dumitrel Loghin, Quang-Trung Ta, Meihui Zhang, Gang Chen, and Beng Chin Ooi. 2020. A Transactional Perspective on Execute-order-validate Blockchains. In *SIGMOD Int. Conf. on Management of Data*. ACM, 543–557.

[165] Fred B Schneider. 1990. Implementing fault-tolerant services using the state machine approach: A tutorial. *Computing Surveys (CSUR)* 22, 4 (1990), 299–319.

[166] Marco Serafini, Péter Bokor, Dan Dobre, Matthias Majuntke, and Neeraj Suri. 2010. Scrooge: Reducing the costs of fast Byzantine replication in presence of unresponsive replicas. In *Int. Conf. on Dependable Systems and Networks (DSN)*. IEEE, 353–362.

[167] Ankur Sharma, Felix Martin Schuhknecht, Divya Agrawal, and Jens Dittrich. 2019. Blurring the lines between blockchains and database systems: the case of hyperledger fabric. In *SIGMOD Int. Conf. on Management of Data*. ACM, 105–122.

[168] Victor Shoup. 2000. Practical threshold signatures. In *Int. Conf. on the Theory and Applications of Cryptographic Techniques*. Springer, 207–220.

[169] Nibesh Shrestha, Ittai Abraham, Ling Ren, and Kartik Nayak. 2020. On the optimality of optimistic responsiveness. In *Conf. on Computer and Communications Security (CCS)*. ACM, 839–857.

[170] Atul Singh, Tathagata Das, Petros Maniatis, Peter Druschel, and Timothy Roscoe. 2008. BFT Protocols Under Fire.. In *Symposium on Networked Systems Design and Implementation (NSDI)*, Vol. 8. USENIX Association, 189–204.

[171] Yee Jiun Song and Robbert van Renesse. 2008. Bosco: One-step byzantine asynchronous consensus. In *Int. Symposium on Distributed Computing (DISC)*. Springer, 438–450.

[172] João Sousa and Alysson Bessani. 2015. Separating the WHEAT from the chaff: An empirical design for geo-replicated state machines. In *Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 146–155.

[173] Paulo Sousa, Alysson Neves Bessani, Miguel Correia, Nuno Ferreira Neves, and Paulo Verissimo. 2009. Highly available intrusion-tolerant services with proactive-reactive recovery. *IEEE Transactions on Parallel and Distributed Systems* 21, 4 (2009), 452–465.

[174] Chrysoula Stathakopoulou, Signe Rüsch, Marcus Brandenburger, and Marko Vukolić. 2021. Adding Fairness to Order: Preventing Front-Running Attacks in BFT Protocols using TEEs. In *Int. Symp on Reliable Distributed Systems (SRDS)*. IEEE, 34–45.

[175] Chrysoula Stathakopoulou, David Tudor, Matej Pavlovic, and Marko Vukolić. 2022. [Solution] Mir-BFT: Scalable and Robust BFT for Decentralized Networks. *Journal of Systems Research* 2, 1 (2022).

[176] Diem Team. 2021. *DiemBFT v4: State Machine Replication in the Diem Blockchain.* Technical Report. Technical Report. Diem. https://developers. diem. com/papers/diem-consensus ....

[177] Feng Tian. 2017. A supply chain traceability system for food safety based on HACCP, blockchain & Internet of things. In *Int. Conf. on service systems and service management (ICSSSM)*. IEEE, 1–6.

[178] Gene Tsudik. 1992. Message authentication with one-way hash functions. *ACM SIGCOMM Computer Communication Review* 22, 5 (1992), 29–38.

[179] Giuliana Santos Veronese, Miguel Correia, Alysson Neves Bessani, and Lau Cheuk Lung. 2009. Spin one's wheels? Byzantine fault tolerance with a spinning primary. In *Int. Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 135–144.

[180] Giuliana Santos Veronese, Miguel Correia, Alysson Neves Bessani, and Lau Cheuk Lung. 2010. EBAWA: Efficient Byzantine agreement for wide-area networks. In *Int. Symposium on High Assurance Systems Engineering (HASE)*. IEEE, 10–19.

[181] Giuliana Santos Veronese, Miguel Correia, Alysson Neves Bessani, Lau Cheuk Lung, and Paulo Verissimo. 2013. Efficient byzantine fault-tolerance. *Transactions on Computers* 62, 1 (2013), 16–30.

[182] Gauthier Voron and Vincent Gramoli. 2019. Dispel: Byzantine SMR with distributed pipelining. *arXiv preprint arXiv:1912.10367* (2019).

[183] Michael Whittaker, Ailidani Ailijiang, Aleksey Charapko, Murat Demirbas, Neil Giridharan, Joseph M Hellerstein, Heidi Howard, Ion Stoica, and Adriana Szekeres. 2021. Scaling replicated state machines with compartmentalization. *Proceedings of the VLDB Endowment* 14, 11 (2021), 2203–2215.

[184] Chenyuan Wu, Mohammad Javad Amiri, Jared Asch, Heena Nagda, Qizhen Zhang, and Boon Thau Loo. 2022. FlexChain: An Elastic Disaggregated Blockchain. *Proc. of the VLDB Endowment* 16, 01 (2022), 23–36.

[185] Chenyuan Wu, Mohammad Javad Amiri, Haoyun Qin, Bhavana Mehta, Ryan Marcus, and Boon Thau Loo. [n. d.]. Towards Full Stack Adaptivity in Permissioned Blockchains. ([n. d.]).

[186] Chenyuan Wu, Bhavana Mehta, Mohammad Javad Amiri, Ryan Marcus, and Boon Thau Loo. 2023. AdaChain: A Learned Adaptive Blockchain. *Proc. of the VLDB Endowment* 16, 8 (2023), 2033–2046.

[187] David Yakira, Avi Asayag, Gad Cohen, Ido Grayevsky, Maya Leshkowitz, Ori Rottenstreich, and Ronen Tamari. 2021. Helix: A Fair Blockchain Consensus Protocol Resistant to Ordering Manipulation. *IEEE Transactions on Network and Service Management* 18, 2 (2021), 1584–1597.

[188] Jian Yin, Jean-Philippe Martin, Arun Venkataramani, Lorenzo Alvisi, and Mike Dahlin. 2003. Separating agreement from execution for byzantine fault tolerant services. *Operating Systems Review (OSR)* 37, 5 (2003), 253–267.

[189] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. 2019. HotStuff: BFT consensus with linearity and responsiveness. In *Symposium on Principles of Distributed Computing (PODC)*. ACM, 347–356.

[190] Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. 2018. RapidChain: Scaling blockchain via full sharding. In *SIGSAC Conf. on Computer and Communications Security*. ACM, 931–948.

[191] Yunhao Zhang, Srinath Setty, Qi Chen, Lidong Zhou, and Lorenzo Alvisi. 2020. Byzantine ordered consensus without Byzantine oligarchy. In *Symposium on Operating Systems Design and Implementation (OSDI)*. USENIX Association, 633–649.

[192] Lidong Zhou, Fred Schneider, Robbert VanRenesse, and Zygmunt Haas. 2002. Secure distributed on-line certification authority. US Patent App. 10/001,588.

[193] Lidong Zhou, Fred B Schneider, and Robbert Van Renesse. 2002. COCA: A secure distributed online certification authority. *ACM Transactions on Computer Systems (TOCS)* 20, 4 (2002), 329–368.