SoK: X-assisted BFT Consensus Protocols

Gang Wang[1,2(✉)] and Mark Nixon[1]

[1] Emerson Automation Solutions, Austin, USA
[2] University of Connecticut, Storrs, USA
gang.wang.dr@gmail.com

Abstract. Blockchain, as an enabler of the current Internet infrastructure, has introduced a plethora of unique features, revolutionizing distributed systems and propelling us into a new era. Its core principles of decentralization, immutability, and transparency have enticed numerous applications to embrace the blockchain design philosophy and tailor diverse replicated solutions. At the heart of the blockchain lies the consensus protocols, which play a pivotal role in achieving distributed replication systems. The distributed system community has invested significant efforts in comprehensively studying the technical components of consensus to enable agreement among a group of nodes. Nonetheless, the presence of various faults and trust issues poses challenges in designing resilient systems for practical applications. To address this, Byzantine fault-tolerant (BFT) state machine replication (SMR) emerges as an ideal candidate capable of tolerating arbitrary faulty behaviors. Despite its promise, the inherent complexity and rapid evolution of BFT consensus protocols hinder their practical adaptation to different application domains. Remarkably, there exists a wealth of exceptional Byzantine-based replicated solutions and innovative ideas that have notably improved performance, availability, and resource efficiency. This paper aims to conduct a systematic and comprehensive study of X-assisted BFT consensus protocols, with a specific focus on the blockchain era. For instance, numerous studies have explored the utilization of trusted components and cryptographic primitives to assist in tolerating Byzantine nodes and reducing the number of communication rounds. We delve into the essentials of BFT consensus protocols for blockchains in Byzantine settings. We then decompose the state-of-the-art solutions to gain a comprehensive BFT consensus in detail. For each X-assisted protocol, we conduct an in-depth discussion of its essential architectural building blocks and the key techniques employed. We aim that this paper can provide system researchers and developers with a concrete view of the current design landscape and facilitate their quest for practical solutions to specific problems.

1 Introduction

The consensus protocol serves as the *core* of the blockchain, providing essential agreement services that significantly impact the performance and scalability of

Q. Wang et al. (Eds.): ICBC 2023, LNCS 14206, pp. 54–71, 2023.
https://doi.org/10.1007/978-3-031-44920-8_4

the entire system. In the absence of trusted intermediaries, participants in a blockchain network may act arbitrarily and deviate from the established consensus procedures, creating what can be described as a Byzantine environment. While blockchain can leverage various technologies for consensus, state replication, and transaction broadcasting, uncertainties in network connectivity can lead to node crashes or subversion by adversaries. To address these challenges, proof-based protocols have been developed for blockchain, such as Proof-of-Work (PoW) in Bitcoin [1]. However, these protocols often lack energy efficiency and may lead to power shortages. Fortunately, Byzantine fault-tolerant (BFT) state machine replication (SMR) offers promising opportunities to design consensus protocols that can tolerate arbitrary faults [2]. The underlying BFT SMR replicates the state of each replica in the system, rendering it capable of withstanding diverse faults and making it suitable for practical and critical applications. However, designing a functioning BFT system remains a challenging task, primarily due to its inherent complexity.

In general, a consensus protocol must satisfy three fundamental requirements [3]: (a) *Non-triviality:* If a correct entity outputs a value v, then some entity proposed v. (b) *Safety:* If a correct entity outputs a value v, then all correct entities output the same value v. (c) *Liveness:* If all correct entities initiated the protocol, then, eventually, all correct entities output some value. However, Fisher, Lynch, and Paterson (FLP) [4] demonstrated the *FLP impossibility*, proving that a deterministic agreement protocol in an asynchronous network cannot guarantee liveness if one entity may crash, even when links are assumed to be reliable. In an asynchronous system, it is impossible to distinguish between a crashed node and a correct one. Therefore, deciding the full network's state and deducing an agreed-upon output from it is deemed impossible. Nevertheless, several extensions have been developed to circumvent the FLP result and achieve asynchronous consensus. These extensions include randomization, timing assumptions, failure detectors, and strong primitives [5]. Over the course of two decades, BFT algorithms have evolved into a diverse array of protocols and applications. However, this progress has been primarily designed for closed groups based on specific application scenarios.

BFT consensus protocols form the crux of blockchain technology, determining its applicability to practical real-world scenarios. The literature encompasses numerous works discussing different aspects of Byzantine-related protocols, ranging from theoretical foundations to practical prototype deployments. While the application of BFT protocols to blockchain holds promise, it also faces significant design challenges when considering the specific requirements of the blockchain environment. In the literature, some works have explored the integration of BFT consensus protocols into the blockchain ecosystem, such as the work [6]. This paper focuses on X-assisted BFT protocols, aiming to provide a comprehensive survey of existing X-assisted Byzantine-related protocols and in-depth discussions on their implementations. Our primary goal is to offer a concrete view of the state-of-the-art literature in the domain of Byzantine-related consensus, thereby aiding researchers and system designers in finding solutions tailored to their spe-

cific needs. For each surveyed paper, we endeavor to provide detailed information and identify potential issues when applying these protocols to blockchain scenarios. Notably, there is ample literature discussing BFT consensus protocols in general forms or from architectural and theoretical perspectives.

The rest of this paper is organized as follows. Section 2 presents well-known X-assisted BFT consensus protocols. In Sect. 3, we provide discussions and explore future directions in this domain. Finally, Sect. 4 concludes the paper (Fig. 1).

2 X-assisted BFT Protocols

X-assisted BFT consensus is primarily employed to bolster robustness or enhance scalability and efficiency. Here, the term 'X' can refer to software primitives (e.g., crypto-primitives) or hardware components (e.g., trusted hardware). The core idea behind X-assisted BFT consensus revolves around ensuring the authenticity of communicated messages. For example, certain protocols, such as SBFT [7], utilize threshold signature schemes to ensure sufficient replicas can collaboratively process requests. Similarly, protocols like Steroids [8] may leverage trusted execution environments (e.g., Intel SGX) as trusted hardware to verify message authenticity. Moreover, approaches incorporating both cryptographic primitives and trusted hardware can work in tandem to improve efficiency. For instance, FastBFT [9] integrates Trusted Execution Environments (TEEs) with a lightweight secret-sharing scheme, enabling efficient message aggregation and achieving scalable Byzantine consensus. This section provides an in-depth discussion of works on X-assisted BFT consensus protocols.

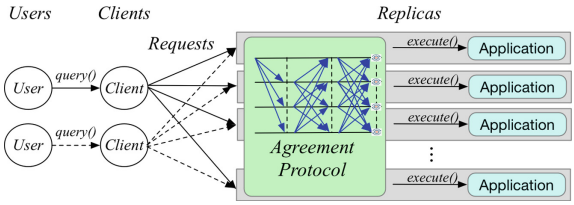

Fig. 1. Abstract of BFT replication system. Users send requests to replicas via client interfaces (with a well-defined client library). Replicas together run an agreement protocol to obtain an order on clients' requests, and then each replica executes them in its stateful application [2].

2.1 X-assisted BFT in Details

A2M. A2M, short for *Attested Append-Only Memory*, was proposed by Chun et al. in 2007 [10] to eliminate equivocation, a common source of Byzantine headaches. A2M serves as a trusted system facility that is small, easy to implement, and verifiable formally. It provides a programming abstraction of a trusted

log, leading to protocol designs immune to equivocation. Equivocation refers to the ability of a faulty replica to lie in different ways to different clients or servers. The A2M protocol can be an add-on component to existing Byzantine fault-tolerant replicated state machines (e.g., PBFT, Q/U. HQ), enabling A2M-enabled protocols. In replicated state machines, the target safety guarantee is typically *linearizability*, which ensures that client requests appear to be processed in a single, totally ordered, serial schedule consistent with the order in which clients submitted their requests and received responses. A2M achieves linearizability through a small trusted log abstraction as its primitive. One key insight behind A2M is its provision of a mechanism (trusted log) that prevents participants from equivocating, thereby improving the fault-tolerance of Byzantine protocols to f out of $2f+1$. Once an action is recorded in the log, it cannot be overwritten, as A2M does not provide a modification interface.

The overall design of A2M is based on a classic client-server system, where clients request *authenticated* operations, and the server responds to these requests. A2M's network model operates in the partially synchronous model, where a finite upper bound exists for message delivery. A2M considers two fault models: the *faulty application model*, where the node's owner is well-intentioned but unaware of the node's compromised software, and the *faulty operator model*, where the node exhibits Byzantine behavior due to malicious instructions from its owner. For each fault model, A2M has a different trusted computing base. In the first model, the service owner establishes the trusted computing base, while in the second model, the owners cannot be trusted, and a third party is responsible for setting up the trusted computing base. An A2M implementation within the trusted computing base allows a protocol to assume that a seemingly correct host can provide only a single response to each distinct protocol request. Therefore, informally, A2M can be thought of as equipping a host with a set of trusted, undeniable, ordered logs. An A2M log provides methods for *appending* values, *looking up* values within the log or obtaining the *end* of the log, as well as *truncating* and *advancing* the log suffix stored in memory. Importantly, there are no methods to replace values that have already been assigned, as A2M employs cryptography to enforce its properties and attest the log's contents to other machines. By incorporating A2M into its trusted computing base, reliable service can mitigate the effects of Byzantine faults in its untrusted components by relying on small fallback information about individual operations or histories of operations that cannot be tampered with.

TrInc. TrInc, short for *Trust Incrementer*, is a small trusted component designed to address equivocation in large-scale distributed systems, proposed by Levin et al. in 2009 [11]. TrInc is motivated by the assumption that individual components in the system are completely untrusted, necessitating the use of trusted technologies to ensure trustworthiness and eliminate equivocation. For instance, A2M uses trusted logs for this purpose. However, trusted log modules often require substantial storage space and can be challenging to implement and deploy in large distributed systems. The primary security goal of TrInc is

to remove participants' ability to equivocate. It achieves this through the use of a non-decreasing trusted counter and a key, enabling it to provide a new primitive: unique, once-in-a-lifetime attestations. With this primitive, TrInc can support a broader range of protocols, including not only client-server systems but also peer-to-peer systems. One advantage of TrInc is its smaller size and simpler semantics, making it easier to deploy. It can be implemented on off-the-shelf available trusted hardware, and its core functional elements are included in a Trusted Platform Module (TPM) [12], commonly found in many modern devices. This suggests that such a component could become widely available. Additionally, TrInc utilizes a shared symmetric session key among all participants in protocol instances, significantly reducing cryptographic overhead.

One common approach to address equivocation is by using heavy-communication protocols designed to handle a threshold number of faulty participants, as exemplified by PBFT. However, TrInc aims to minimize both the communication overhead and the required number of non-faulty participants. By leveraging trusted hardware, TrInc can eliminate the ability of a malicious participant to equivocate without necessitating communication among other participants. For TrInc to be practical in distributed systems, the trusted component must be small, allowing for feasible manufacturing and deployment. It is difficult and costly to create tamper-resistant large components, making a small form factor essential. The "trinket" serves as such a trusted piece of hardware within the TrInc system. The trinket's API relies solely on its internal state, distinguishing it from typical TPMs that need to access the state of host devices (e.g., computers). Instead, the trinket requires only an untrusted channel through which it can receive input and produce output.

MinBFT. Both MinBFT and MinZyzzyva are trust-assisted BFT protocols, designed to tolerate f faulty replicas with only $2f + 1$ replicas, and were proposed by Veronese in 2011 [13]. While MinBFT is based on PBFT, MinZyzzyva is based on Zyzzyva, both being asynchronous algorithms. For the purpose of this discussion, we will focus on MinBFT, the PBFT version, to explore its technical advantages. MinBFT significantly improves efficiency compared to previous algorithms in three key metrics: the number of replicas, the simplicity of trusted services, and the number of communication steps. The main source of efficiency in MinBFT lies in the use of a simple trusted component. More precisely, the trusted services assisting in reducing the number of replicas are designed to be straightforward, facilitating verified implementations and even feasibility using commercial trusted hardware. Moreover, algorithms based on hardware tend to be simpler, approaching the level of crash fault-tolerant replication algorithms.

The successful implementation of trusted services in MinBFT is based on the usage of USIG (Unique Sequential Identifier Generator). USIG provides an interface with operations to increment the counter and verify the correct authentication of other counter values (incremented by other replicas). Each server has a local USIG service responsible for assigning unique, monotonic, and sequential identifiers to messages. Even if a server is compromised, USIG guarantees these

properties, making it essential to implement the service in a tamper-proof module or a trusted component. Fortunately, the trusted component can be implemented even on commercially available trusted hardware, such as the *trusted platform module* [14].

In MinBFT, one main role of the leader is to assign a unique sequence number to each request, and this number is the counter value returned by the USIG service, ensuring the uniqueness, monotonicity, and sequentiality of identifiers. These sequence numbers remain sequential as long as the leader does not change but may change during a view change. To ensure fault-tolerance and the possibility of resending messages, servers keep a message log that stores sent messages. MinBFT employs a garbage collection mechanism based on checkpoints, similar to PBFT, to discard unnecessary messages from the log. Besides, the implementation of MinBFT and MinZyzzyva provides several levels of isolation for a trusted component used to enhance BFT algorithms. They have also implemented multiple versions of the USIG service, each using different cryptographic mechanisms. These implementations are isolated in both separate virtual machines and trusted hardware.

CheapBFT. CheapBFT is a resource-efficient BFT system based on a trusted subsystem designed to prevent equivocation, proposed by Kapitza in 2012 [15]. CheapBFT can tolerate the failure of *all but one* of the replicas that are active during normal case operation. In general, it runs a composite agreement protocol and utilizes passive replication to save resources. At a high-level perspective, the agreement protocol of CheapBFT consists of three sub-protocols: the normal case protocol *CheapTiny*, the transition protocol *CheapSwitch*, and the fall-back protocol *MinBFT*. Essentially, CheapBFT relies on an FPGA-based trusted subsystem known as *CASH* to prevent equivocation and ensure the system's integrity and correctness during the consensus process.

CASH stands for *Counter Assignment Service in Hardware*, and it is designed to assist CheapBFT with message authentication and verification. To prevent equivocation, each replica in CheapBFT must be equipped with a trusted CASH subsystem. Each CASH subsystem is initialized with a secret key and uniquely identified by a subsystem ID, corresponding to the replicas that host the subsystem. The primary function of CASH is to provide a trusted counter service, achieved by issuing message certificates for protocol messages. These certificates contain the identity *id* of the subsystem, the assigned counter value, and a MAC generated using the secret key. CASH employs symmetric-key cryptographic operations for message authentication and verification. In its basic version, CASH offers functions to create (via *createMC*) and verify (via *checkMC*) message certificates, tailored for single counter cases. For more complex scenarios with distinct counter instances and several concurrent protocols, the full version of CASH supports multiple counters, each specified by a different *counter name*. To ensure practicality, CASH is designed with two primary goals: a minimal trusted computing base and high performance. Keeping the code size of CASH small reduces the probability of program errors that could be exploited by poten-

tial attacks. Additionally, CASH ensures a high throughput during interactions involving authenticated messages to meet the system's performance requirements. Importantly, the trusted CASH subsystem is crash-fault tolerant, and its key remains secret even in the presence of Byzantine replicas (Figs. 2 and 3).

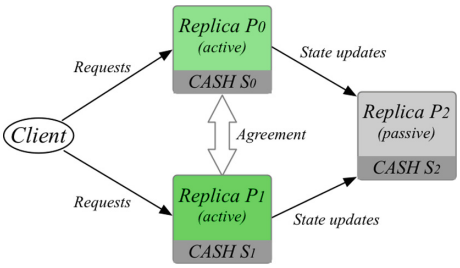

Fig. 2. CheapBFT with two active replicas and a passive replica ($f = 1$) for normal-case operation [15].

Hybster. Hybster is a hybrid BFT protocol proposed by Behl et al. in 2017 [8], which leverages a trusted subsystem for message authentication to prevent equivocation. It demonstrates the ability to tolerate up to f Byzantine faults with only $2f+1$ replicas, thanks to the assistance of Intel SGX [16]. In modern multi-core systems, new parallelization schemes have emerged, enabling traditional BFT protocols to achieve unparalleled performance levels. Some state-of-the-art general-purpose processors offer a trusted execution environment, safeguarding software components even against the malicious behavior of an untrusted operating system. Hybster, being a highly parallelizable and formally specified hybrid SMR protocol, takes advantage of this trend. In hybrid fault models, prior SMR systems usually necessitate sequential processing of consensus instances to agree on the execution order of commands or all incoming messages. Hybster, on the other hand, explores the potential of parallelism. It abstractly presents a parallelizable structure (shown in Fig. 4), wherein multiple instances can be executed simultaneously on some physical replicas. This feature contributes to an accelerated throughput of the system. The central concept that ensures undetected

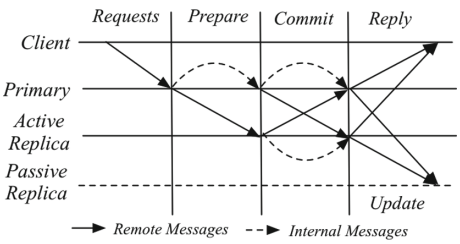

Fig. 3. CheapTiny protocol messages exchanged between a client, two active replicas, and a passive replica ($f = 1$) [15].

equivocation in Hybster involves cryptographically binding sensitive outgoing messages to a unique monotonically increasing timestamp, accomplished through the trusted subsystem. This approach enhances security while capitalizing on the benefits of parallel processing.

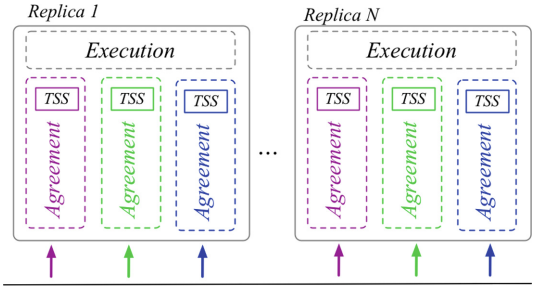

Fig. 4. Hybster: A parallelizable Hybrid [8].

In more detail, Hybster is designed around a two-phase ordering process, utilizing multiple instances of a TrInc-based trusted subsystem realized using Intel SGX to prevent equivocation. While there are other trusted schemes like A2M-PBFT and MinBFT, Hybster distinguishes itself with three key features: relaxation, formal specification, and parallelizability. Hybster relies on a trusted subsystem abstraction, known as TrInX, which is similar but not identical to TrInc [11]. It is implemented in Java and employs a consensus-oriented parallelization scheme, optimized to fully utilize multi-core CPUs. As a result, Hybster achieves high performance, and its scalability improves as the number of NIC and CPU cores increases.

FastBFT. FastBFT is a fast and scalable BFT protocol with the help of trusted hardware, proposed by Liu et al. in 2018 [9]. Essentially, FastBFT utilizes a message aggregation technique that combines a hardware-based trusted execution environment (TEE) with a lightweight secret-sharing scheme. From a high-level perspective, FastBFT also combines several other optimizations, such as optimistic execution, tree topology, and failure detection, to achieve low latency and high throughput even for large-scale networks. By using message aggregation, it can reduce the message complexity from $O(n^2)$ to $O(n)$, and the message aggregation in FastBFT does not require any public-key operations (e.g., multi-signatures), which can further reduce the computation/communication overhead. With a tree topology design in arranging nodes, FastBFT can balance computation and communication load, so that inter-server communication and message aggregation take place along the edges of the tree. Due to the optimistic design, FastBFT only requires a subset of nodes to actively run the protocol. Additionally, FastBFT utilizes a simple failure detection mechanism to handle non-primary faults efficiently.

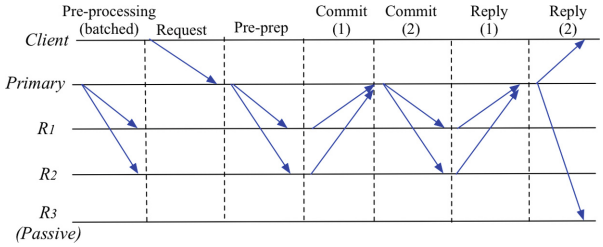

Fig. 5. Message patterns of FastBFT [9].

In general, there are two main categories for improving BFT performance when replicas rarely fail: speculative and optimistic mechanisms. The speculative mechanism typically involves not running any explicit agreement protocol (e.g., Zyzzyva). On the other hand, the optimistic mechanism only requires a subset of replicas to run the agreement protocol, while other replicas passively update their states and become actively involved only if the agreement protocol fails. The FastBFT protocol adopts an optimistic mechanism by incorporating a Trusted Execution Environment (TEE) environment. This allows replicas to remotely verify (e.g., via remote attestation) the behaviors of other replicas, with TEEs being capable of crashing but not acting in a Byzantine manner. FastBFT guarantees safety in asynchronous networks but requires weak synchrony for liveness, and each replica holds a hardware-based TEE that maintains a monotonic counter and rollback-resistant memory. The TEEs can verify one another using remote attestation and establish secure communication channels among replicas. Figure 5 illustrates the message communication pattern of the FastBFT consensus protocol. Essentially, the consensus protocol of FastBFT consists of four phases: pre-processing, request, prepare, and commit. The commit phase further includes two sub-phases that are used to update the state of replicas, similar to the execution phase in traditional BFTs. Besides, the overall FastBFT protocol also includes failure detection and view-change processes.

SACZyzzyva. SACZyzzyva, short for Single-Active Counter Zyzzyva, is a protocol designed to provide resilience to slow replicas and requiring only $3f + 1$ replicas, with just one replica needing an active monotonic counter at any given time. It was proposed by Gunn et al. in 2019 [17]. Speculative BFT protocols, such as Zyzzyva and Zyzzyva5, offer highly efficient speculative execution paths when there are no faults or delays. However, these protocols come with trade-offs. For instance, Zyzzyva requires $3f + 1$ replicas to tolerate f faults, but *even a single slow replica* can force Zyzzyva to fall back to a more expensive non-speculative operation. Similarly, while Zyzzyva5 does not necessitate a non-speculative fallback, it does require $5f + 1$ replicas to tolerate f faulty replicas. In realistic communication networks, like the Internet, which are only partially synchronous, the presence of just a single *slow* but not faulty replica can trigger non-speculative execution for each protocol run of Zyzzyva, undermining the

efficiency promised by the speculative approach. SACZyzzyva addresses these drawbacks by requiring only a single replica, the primary, to have an active monotonic counter. This eliminates the need for a non-speculative fallback and enables tolerance for a subset of replicas being slow while still requiring only $3f + 1$ replicas. SACZyzzyva leverages the trusted hardware of some replicas (not all replicas) to assist in its process, following a practical setting where only some devices have the necessary hardware support. Furthermore, other BFT protocols can also adopt the single active counter approach of SACZyzzyva to reduce latency without the requirement of equipping all replicas with hardware-supported monotonic counters.

In more detail, SACZyzzyva operates under a weak-synchrony model, which allows for the analysis of liveness during a period of synchrony that will eventually occur. Additionally, SACZyzzyva assumes that *some*, but not all, replicas are equipped with a trusted component, specifically a trusted monotonic counter. The fundamental principle behind SACZyzzyva is to utilize a trusted monotonic counter in the primary replica. This counter binds a sequence of consecutive counter values to incoming requests, effectively ordering the requests without the need for communication between replicas, either directly or via the client. The primary achieves this by signing a tuple comprising the cryptographic hash of the request and a fresh counter value, resulting in a single active counter. As a result, SACZyzzyva only requires that $f + 1$ replicas have a trusted component, ensuring that there will always be at least one correct replica that can serve as the primary.

TBFT. TBFT is a TEE-based BFT protocol inspired by the structure of CFT (Crash Fault Tolerant) protocols, aiming to provide simplicity and ease of understanding. It was proposed by Zhang et al. in 2021 [18]. Unlike most existing TEE-based BFT protocols, which often involve complex operations to address security challenges introduced by TEE, TBFT takes inspiration from CFT protocols to create a straightforward and comprehensible design. In practical scenarios, many TEE-based protocols assume adversaries similar to CFT, leading to the elimination of Byzantine failures and focusing on crashed failures. The authors identified key differences between TEE-based BFT and CFT protocols and proposed four principles to bridge the gap between them. Building on these principles, TBFT introduces several improvements to enhance both performance and security. These enhancements include pipeline mechanisms, a TEE-assisted secret sharing scheme, and a trusted leader election process, all contributing to improved performance and scalability. By adopting the advantages of CFT, such as a high resilient fault rate, TBFT offers a TEE-based BFT solution with a clear and concise structure. This design approach makes TBFT more accessible for understanding and implementation compared to traditional TEE-based BFT protocols, which often tend to be more complex and challenging to grasp.

2.2 Last but Not Least

In more detail, most protocols assume that TEE may crash but will never provide malicious execution results, which makes TEE-based BFT more similar to CFT rather than BFT. However, even with the existence of TEE, a Byzantine host can still terminate TEE at any moment, schedule TEE arbitrarily, or drop, reply, delay, reorder, or modify the I/O messages of TEE. This simply can be stated that the host in TEE-based BFT may be Byzantine. Thus, it is necessary to bridge the gap between TEE-based BFT and CFT. To bridge the gap, the authors proposed four principles: *one protocol*, *one vote*, *restricted commit*, and *restricted log generation*. For one proposal, the leaders need to call a function, e.g., *create counter* for trusted monotonic counters based TEE, to assign a (c, v) (c is a monotonic counter, v is the current view) for each proposal while other replicas will keep track of the leader's (c, v).

In addition to the above-mentioned representative X-assisted BFT protocols, there are other notable works that utilize trusted hardware. Yandamuri et al. [19] proposed a scheme that utilizes small trusted hardware without increasing communication complexity, assuming the adversary controls a fraction of the network that is less than one-half. This scheme builds upon a version of HotStuff to preserve linear communication complexity while leveraging trusted hardware to tolerate a minority of corruptions. Wang et al. [20] introduced ENGRAFT, a secure enclave-guarded Raft implementation designed to achieve consensus on a cluster of $2f + 1$ replicas, with up to f replicas exhibiting Byzantine behavior (while operating within well-behaved enclaves). This solution provides an abstraction of confidential consensus, enabling privacy-preserving State Machine Replication (SMR) and facilitating the integration of a production-quality Raft implementation (BRaft). Aguilera et al. [21] proposed uBFT, a consensus protocol designed to achieve microsecond-scale latency in data centers using only $2f+1$ replicas to tolerate up to f Byzantine failures. uBFT relies on a small, non-tailored trusted computing base and leverages disaggregated memory, ensuring a practical and bounded memory consumption. The protocol is built upon an abstraction named Consistent Tail Broadcast, which prevents equivocation while efficiently managing memory. By incorporating RDMA-based disaggregated memory, uBFT achieves an impressive end-to-end latency as low as 10 microseconds. Feng et al. [22] introduced a secure and trusted BFT (S2BFT) consensus, employing trusted committees. This protocol generates anonymous numbers using TEE for each server node and selects committees through a pseudo-random algorithm.

DAMYSUS is a streamlined protocol based on basic HotStuff, enhanced by the utilization of two fundamental trusted services: *Checker* and *Accumulator* [23]. The Checker service ensures that nodes cannot vote for conflicting blocks or misrepresent the blocks they have previously voted for, while the Accumulator service guarantees that leaders can only propose blocks consistent with past votes. The protocol requires $2f+1$ replicas to tolerate up to f Byzantine failures and is capable of terminating within 2 communication phases. *SplitBFT* leverages TEE-based compartmentalization technology to enhance the safety and confidentiality guarantees of BFT systems, bolstering the trust in code-based

deployments of permissioned blockchains [24]. Unlike traditional assumptions, *SplitBFT* acknowledges that code protected by trusted expectations may still fail. To address this, they propose to split and isolate the core logic of BFT protocols into multiple compartments. This approach improves resilience and confidentiality while simplifying the implementation of diversity. *InterTrust* is an interoperable cross-blockchain communication architecture designed to facilitate interoperability and trustworthiness among diverse blockchain systems [25]. At its core, the architecture relies on a TEE-assisted BFT consensus protocol, enabling seamless interoperability within an autonomous system. *InterTrust* incorporates two groundbreaking techniques: a threshold signature scheme and trusted hardware. The threshold signature scheme ensures consistency and verifiability in the target blockchain systems, while the trusted hardware guarantees trusted services across distinct blockchain systems. The combination of these techniques results in an efficient cross-chain communication protocol, fostering atomic swaps and facilitating interoperable operations between different blockchain systems.

3 Discussion and Future Directions

This section presents some discussion on applying BFT protocols to blockchains and explores potential future directions.

3.1 Choices on Paxos Vs. BFT

Paxos is a well-known consensus protocol that achieves agreement under crash failures [26]. Initially proposed as a solution to the FLP impossibility, Paxos can forgo progress during temporary asynchrony. However, when the system returns to synchrony, Paxos resumes its operation and ensures system consistency.

Classic Paxos (or more generally, CFT) and BFT consensus protocols explicitly model machine faults only and can be combined with orthogonal network fault models, such as the synchronous and asynchronous models. Consequently, the scope can be broadly classified into four categories [27]: synchronous CFT [28] [29], asynchronous CFT [29] [26], synchronous BFT [30] [31], and asynchronous BFT [32] [33]. Depending on the specific requirements of different blockchain applications, system designers can choose the appropriate consensus protocols from the above categories. Additionally, there exist some hybrid fault models, such as XFT [27], Byzantine Paxos [34], and heterogeneous Paxos [35], which aim to handle both CFT and BFT fault scenarios.

3.2 Hybrid Fault Models

The Byzantine fault model inherently poses difficulties in the development of consensus protocols. Typically, a BFT system may assume a powerful adversary or harsh network conditions, or even a combination of both, which introduces complexity and overhead in designing a well-replicated system [2]. As a result,

some designers have observed that it may not be worthwhile to design Byzantine replicated systems for certain secure and reliable applications, such as use cases in data centers [36] [37]. Some recent works have transitioned to hybrid fault models [38] with weaker guarantees, where Byzantine replicas only account for a small portion of all faulty replicas, allowing for more practical implementations. There are several literature works that focus on these hybrid fault models, such as UpRight [39], VFT [37], and XFT [27].

Trust plays a crucial role in ensuring the effectiveness of replicated systems under hybrid fault models. In essence, a trusted system is equipped with a small trusted computing base [40], which enables the identification of incorrectness. While a malicious replica may have the ability to operate on untrusted components, it lacks the capability to control trusted components. With the advancements in modern processors, implementing trust components in dedicated hardware modules, such as Trusted Platform Module (TPM) [41] [13], Intel's SGX [42], and ARM's TrustZone [43], to provide trusted execution environments has become more favorable. Additionally, there are software-based solutions to establish trusted components, such as via the proxy [44], a multicast ordering service [45] [46], or a virtualization layer [47] [48] [49]. In general, trusted components ensure that replicas can recover even if they become compromised [32]. They also prevent a faulty leader from successfully equivocating. As a result, whether in the form of hardware or software, trusted components offer a level of trustworthiness under hybrid fault models, helping replicated systems reach consensus with fewer required replicas. This approach proves to be more practical in certain application scenarios, such as data centers and permissioned blockchain systems.

3.3 Liveness in Consensus

A BFT consensus protocol typically achieves progress through a sequence of *views*, with each view having a designated leader responsible for driving the entire consensus process. Liveness is one of the two fundamental properties that consensus aims to achieve, along with safety. Liveness ensures that a transaction sent to all honest validators will eventually be executed. Theoretically, consensus protocols can achieve liveness by assuming an unknown *Global Stabilization Time (GST)*. After some GST period, the network may enter a period of synchrony, characterized by bounded but unknown constant message delay. However, despite claims of providing liveness guarantees, most existing works fail to offer a concrete value (e.g., latency) for this bound, making it challenging to make informed decisions.

In the literature, some works propose approaches to address liveness issues under diverse network conditions. For instance, Abraham et al. [50] introduce the concept of clock synchronization [51,52] to achieve "view synchronization," wherein each correct replica can access hardware clocks with reliable and bounded time drift. The HotStuff protocol [53] incorporates a component named *PaceMaker* to achieve view synchronization and advance progress. However, it does not provide a detailed specification of how this functionality is achieved.

Bravo et al. [54] present a similar view synchronization scheme, which provides a wrapper for BFT consensus procedures' functionality but offers formal specifications only under partial synchrony. While significant progress has been made in addressing liveness issues in BFT protocols, there is still a lack of practical live Byzantine consensus protocols that can effectively operate under fully asynchronous environments, such as the Internet. As a result, achieving a safe and live BFT consensus protocol remains a challenging task.

3.4 Scalability

Scaling protocols for BFT consensus typically prioritize either reducing the number of nodes required to tolerate f Byzantine faults [10] or minimizing the protocol's communication complexity to accommodate larger network sizes [55].

Reducing the Number of Nodes. To tolerate f Byzantine nodes that can *equivocate* in a *quorum* system like PBFT, quorums must be intersected by at least $f+1$ nodes [56]. Consequently, if a BFT protocol requires $n = 3f+1$ nodes, its quorum size is at least $2f+1$. A smaller n implies lower communication costs incurred in tolerating the same number of faults. Additionally, for the same number of nodes n, the network can tolerate more faulty nodes.

Reducing Communication Complexity. Despite reducing the network size, PBFT still exhibits a communication complexity of $O(n^2)$. Byzcoin [55] proposed an optimization using the collective signing protocol (CoSi) [57], wherein the leader aggregates other nodes' messages into a single authenticated message. This approach allows each node to forward its messages to the leader and verify the aggregate message, effectively reducing the communication complexity to $O(n)$ by avoiding broadcasting. Additionally, some works [58] explore leveraging trusted execution environments (TEEs) such as Intel SGX [59] to scale distributed consensus, like the topic presented in this paper. TEEs provide protected memory and isolated execution, ensuring that regular operating systems or applications cannot control or observe the data stored or processed inside them [60]. Although trusted hardware can only crash and not act in a Byzantine manner, introducing it into consensus nodes is costly and requires specific knowledge for protocol implementation. Moreover, the security in this category can be enhanced by using cryptographic primitives, such as threshold signatures [61] [62].

Furthermore, several other intriguing research topics are emerging, such as testing technologies to evaluate the efficiency of both BFT protocols and blockchains, and schemes aimed at preventing malicious replicas' collaboration or centralization. The journey ahead for both BFT consensus protocols and blockchains remains extensive and filled with opportunities for exploration and advancement.

4 Conclusion

In recent years, research on BFT consensus has experienced a dramatic surge, partially attributed to the emergence of blockchain technology. This paper

presents a Systematization of Knowledge (SoK) for existing efforts on X-assisted BFT consensus protocols. We meticulously studied the selected BFT protocols and strived to provide a comprehensive review with detailed analysis. This paper serves as a valuable starting point for exploring consensus in the realms of both X-assisted BFT and blockchain. Additionally, we present several potential research directions that can contribute to advancing reliable and robust BFT consensus within the blockchain ecosystem.

References

1. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. Tech. Rep, Manubot (2008)
2. Distler, T.: Byzantine fault-tolerant state-machine replication from a systems perspective. ACM Comput. Surv. (CSUR) **54**(1), 1–38 (2021)
3. Maric, O., Sprenger, C., Basin, D.: Consensus refined. In: 2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, pp. 391–402. IEEE (2015)
4. Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of distributed consensus with one fault process. YALE UNIV NEW HAVEN CT DEPT OF COMPUTER SCIENCE, Technical report (1982)
5. Aspnes, J.: Randomized protocols for asynchronous consensus. Distrib. Comput. **16**(2–3), 165–175 (2003)
6. Wang, G.: Sok: understanding BFT consensus in the age of blockchains. Cryptology ePrint Archive (2021)
7. Gueta, G.G., et al.: Sbft: a scalable and decentralized trust infrastructure. In: 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp. 568–580. IEEE (2019)
8. Behl, J., Distler, T., Kapitza, R.: Hybrids on steroids: Sgx-based high performance bft. In: Proceedings of the Twelfth European Conference on Computer Systems, pp. 222–237 (2017)
9. Liu, J., Li, W., Karame, G.O., Asokan, N.: Scalable byzantine consensus via hardware-assisted secret sharing. IEEE Trans. Comput. **68**(1), 139–151 (2018)
10. Chun, B.-G., Maniatis, P., Shenker, S., Kubiatowicz, J.: Attested append-only memory: making adversaries stick to their word. ACM SIGOPS Operating Syst. Rev. **41**(6), 189–204 (2007)
11. Levin, D., Douceur, J.R., Lorch, J.R., Moscibroda, T.: Trinc: small trusted hardware for large distributed systems. In: NSDI, vol. 9, pp. 1–14 (2009)
12. Kinney, S.L.: Trusted platform module basics: using TPM in embedded systems. Elsevier (2006)
13. Veronese, G.S., Correia, M., Bessani, A.N., Lung, L.C., Verissimo, P.: Efficient byzantine fault-tolerance. IEEE Trans. Comput. **62**(1), 16–30 (2011)
14. Ryan, M.: Introduction to the tpm 1.2. DRAFT of March, vol. 24 (2009)
15. Kapitza, R., et al.: Cheapbft: resource-efficient byzantine fault tolerance. In: Proceedings of the 7th ACM European Conference on Computer Systems, pp. 295–308 (2012)
16. McKeen, F., et al.: Innovative instructions and software model for isolated execution. Hasp@ isca, vol. 10, no. 1 (2013)

17. Gunn, L.J., Liu, J., Vavala, B., Asokan, N.: Making speculative BFT resilient with trusted monotonic counters. In: 2019 38th Symposium on Reliable Distributed Systems (SRDS), pp. 133–13 309. IEEE (2019)
18. Zhang, J., et al.: Tbft: understandable and efficient byzantine fault tolerance using trusted execution environment. arXiv preprint arXiv:2102.01970 (2021)
19. Yandamuri, S., Abraham, I., Nayak, K., Reiter, M.K.: Communication-efficient bft using small trusted hardware to tolerate minority corruption. In: 26th International Conference on Principles of Distributed Systems (OPODIS 2022). Schloss Dagstuhl-Leibniz-Zentrum für Informatik (2023)
20. Wang, W., Deng, S., Niu, J., Reiter, M.K., Zhang, Y.: Engraft: enclave-guarded raft on byzantine faulty nodes. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, pp. 2841–2855 (2022)
21. Aguilera, M.K., Ben-David, N., Guerraoui, R., Murat, A., Xygkis, A., Zablotchi, I.: UBFT: microsecond-scale BFT using disaggregated memory. In: Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, vol. 2, pp. 862–877 (2023)
22. Feng, L., Ding, Y., Tan, Y., Fu, X., Wang, K., sheng Chang, J.: Trusted-committee-based secure and scalable BFT consensus for consortium blockchain. In: 2022 18th International Conference on Mobility, Sensing and Networking (MSN), pp. 363–370. IEEE (2022)
23. Decouchant, J., Kozhaya, D., Rahli, V., Yu, J.: Damysus: streamlined BFT consensus leveraging trusted components. In: Proceedings of the Seventeenth European Conference on Computer Systems, pp. 1–16 (2022)
24. Messadi, I., Becker, M.H., Bleeke, K., Jehl, L., Mokhtar, S.B., Kapitza, R.: Splitbft: improving byzantine fault tolerance safety using trusted compartments. In: Proceedings of the 23rd ACM/IFIP International Middleware Conference, pp. 56–68 (2022)
25. Wang, G., Nixon, M.: Intertrust: towards an efficient blockchain interoperability architecture with trusted services. In: 2021 IEEE International Conference on Blockchain (Blockchain), pp. 150–159. IEEE (2021)
26. Lamport, L.: The part-time parliament. ACM Trans. Comput. Syst. **16**(2), 133–169 (1998)
27. Liu, S., Viotti, P., Cachin, C., Quéma, V., Vukolić, M.: XFT: practical fault tolerance beyond crashes. In: 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), pp. 485–500 (2016)
28. Cristian, F., Aghili, H., Strong, R., Dolev, D.: Atomic broadcast: From simple message diffusion to byzantine agreement. Inf. Comput. **118**(1), 158–179 (1995)
29. Schneider, F.B.: Implementing fault-tolerant services using the state machine approach: a tutorial. ACM Comput. Surv. (CSUR) **22**(4), 299–319 (1990)
30. Lamport, L., Shostak, R., Pease, M.: The byzantine generals problem. ACM Trans. Program. Lang. Syst. **4**(3), 382–401 (1982)
31. Berman, P., Garay, J.A., Perry, K.J., et al.: Towards optimal distributed consensus. In: FOCS, vol. 89. Citeseer, pp. 410–415 (1989)
32. Castro, M., Liskov, B.: Practical byzantine fault tolerance and proactive recovery. ACM Transactions on Computer Systems (TOCS) **20**(4), 398–461 (2002)
33. Guerraoui, R., Knežević, N., Quéma, V., Vukolić, M.: The next 700 bft protocols. In: Proceedings of the 5th European Conference on Computer Systems, pp. 363–376 (2010)
34. Lamport, L.: Byzantizing paxos by refinement. In: International Symposium on Distributed Computing. Springer, pp. 211–224 (2011)

35. Sheff, I., Wang, X., van Renesse, R., Myers, A.C.: Heterogeneous paxos. In: 24th International Conference on Principles of Distributed Systems (OPODIS 2020). Schloss Dagstuhl-Leibniz-Zentrum für Informatik (2021)
36. Kuznetsov, P., Rodrigues, R.: Bftw3: why? when? where? workshop on the theory and practice of byzantine fault tolerance. ACM SIGACT News **40**(4), 82–86 (2010)
37. Porto, D., et al.: Visigoth fault tolerance. In: Proceedings of the Tenth European Conference on Computer Systems, pp. 1–14 (2015)
38. Thambidurai, P., Park, Y.-K.: Interactive consistency with multiple failure modes. In: Proceedings Seventh Symposium on Reliable Distributed Systems. IEEE Computer Society, pp. 93–94 (1988)
39. Clement, A.: Upright cluster services. In: Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles, pp. 277–290 (2009)
40. Rushby, J.M.: Design and verification of secure systems. ACM SIGOPS Operat. Syst. Rev. **15**(5), 12–21 (1981)
41. Veronese, G.S., Correia, M., Bessani, A.N., Lung, L.C.: Ebawa: efficient byzantine agreement for wide-area networks. In: IEEE 12th International Symposium on High Assurance Systems Engineering. IEEE 2010, pp. 10–19 (2010)
42. Anati, I., Gueron, S., Johnson, S., Scarlata, V.: Innovative technology for CPU based attestation and sealing. In: Proceedings of the 2nd International Workshop on Hardware and Architectural Support for security and privacy, vol. 13, p. 7. ACM, New York (2013)
43. A. ARM: Security technology-building a secure system using trustzone technology. ARM Technical White Paper (2009)
44. Rüsch, S., Bleeke, K., Kapitza, R.: Bloxy: providing transparent and generic bft-based ordering services for blockchains. In: 2019 38th Symposium on Reliable Distributed Systems (SRDS), pp. 305–30 509. IEEE (2019)
45. Correia, M., Neves, N.F., Lung, L.C., Veríssimo, P.: Worm-it-a wormhole-based intrusion-tolerant group communication system. J. Syst. Softw. **80**(2), 178–197 (2007)
46. Correia, M., Veronese, G.S., Neves, N.F., Verissimo, P.: Byzantine consensus in asynchronous message-passing systems: a survey. Int. J. Critical Comput.-Based Syst. **2**(2), 141–161 (2011)
47. Distler, T., Popov, I., Schröder-Preikschat, W., Reiser, H.P., Kapitza, R.: Spare: replicas on hold. In: NDSS (2011)
48. Garcia, M., Bessani, A., Neves, N.: Lazarus: automatic management of diversity in bft systems. In: Proceedings of the 20th International Middleware Conference, pp. 241–254 (2019)
49. Reiser, H.P., Kapitza, R.: Hypervisor-based efficient proactive recovery. In: 26th IEEE International Symposium on Reliable Distributed Systems (SRDS 2007). IEEE 2007, pp. 83–92 (2007)
50. Abraham, I., Devadas, S., Dolev, D., Nayak, K., Ren, L.: Synchronous byzantine agreement with expected o (1) rounds, expected $o(^n2)$ communication, and optimal resilience. In: International Conference on Financial Cryptography and Data Security, pp. 320–334. Springer (2019)
51. Dolev, D., Halpern, J.Y., Simons, B., Strong, R.: Dynamic fault-tolerant clock synchronization. J. ACM (JACM) **42**(1), 143–185 (1995)
52. Simons, B.: An overview of clock synchronization. In: Fault-Tolerant Distributed Computing, pp. 84–96 (1990)
53. Yin, M., Malkhi, D., Reiter, M.K., Gueta, G.G., Abraham, I.: Hotstuff: Bft consensus with linearity and responsiveness. In: Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, pp. 347–356 (2019)

54. Bravo, M., Chockler, G., Gotsman, A.: Making byzantine consensus live. In: 34th International Symposium on Distributed Computing (DISC 2020). Schloss Dagstuhl-Leibniz-Zentrum für Informatik (2020)
55. Kogias, E.K., Jovanovic, P., Gailly, N., Khoffi, I., Gasser, L., Ford, B.: Enhancing bitcoin security and performance with strong consistency via collective signing. In: 25th usenix security symposium (usenix security 16), pp. 279–296 (2016)
56. Malkhi, D., Reiter, M.: Byzantine quorum systems. Distrib. Comput. **11**(4), 203–213 (1998)
57. Syta, E., et al.: Keeping authorities "honest or bust" with decentralized witness cosigning. In: IEEE Symposium on Security and Privacy (SP). IEEE 2016, pp. 526–545 (2016)
58. Dang, H., Dinh, A., Chang, E.-C., Ooi, B.C.: Chain of trust: can trusted hardware help scaling blockchains? arXiv preprint arXiv:1804.00399 (2018)
59. Costan, V., Devadas, S.: Intel sgx explained. IACR Cryptology ePrint Archive **2016**(086), 1–118 (2016)
60. Ekberg, J.-E., Kostiainen, K., Asokan, N.: The untapped potential of trusted execution environments on mobile devices. IEEE Secur. Privacy **12**(4), 29–37 (2014)
61. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the weil pairing. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 514–532. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45682-1_30
62. Stathakopoulous, C., Cachin, C.: Threshold signatures for blockchain systems. Swiss Federal Institute of Technology (2017)