Research and Application of BFT Algorithms Based on the Hybrid Fault Model

Qichao Zhang*†, Zhuyun Qi*†, Xiaoyou Liu‡, Tao Sun‡, Kai Lei*†
*Shenzhen Key Lab for Information Centric Networking & Blockchain Technology (ICNLAB)
†School of Electronics and Computer Engineering, Peking University, Shenzhen, China
‡The Network Information Center, University Town of Shenzhen, Shenzhen, China
Email: qczhang@pku.edu.cn, qizy@pkusz.edu.cn, {liuxy, suntao}@utsz.edu.cn, *:leik@pkusz.edu.cn

***Abstract*—The recent explosion of interest in blockchain led to a plethora of researches on consensus algorithms. Compared with bitcoin-derived consensus mechanisms, Byzantine Fault-Tolerance (BFT) algorithms are more applicable for consortium blockchain. However, these algorithms work lies in the foundation that $3f+1$ nodes are required to tolerate only f faults, which results in high network traffic and cost. The hybrid fault model with Trusted Components (TC) assistance is proposed to reduce the minimum to $2f+1$ and improve overall performance. In this paper, we firstly introduce the basic principles of BFT algorithms based on the hybrid fault model. Then we analyze different TC abstractions and implementation scenarios, and express our own choices. The general framework for BFT algorithms design choices and the applications of this model in blockchain are also discussed. Therefore, this paper aims to provide guidance for reasonable design of efficient BFT algorithms and application of TC assistance mechanism.**

***Index Terms*—Byzantine Fault-Tolerance, hybrid fault model, trusted components, blockchain, consensus**

I. Introduction

The blockchain is a decentralized network system, and is the disruptive technology supporting a suite of cryptocurrency like Bitcoin [1]. It enables multiple parties without mutual trust to maintain shared ledgers, and extends smart contracts to automatic business mode. This innovative distributed computing paradigm has gained more attention from industries.

In the distributed environment, nodes may crash or violate the rules for personal benefits. For example, Bitcoin users may collude to eclipse honest nodes [2], resulting in the possible occurrence of double spending attacks. These unpredictable arbitrary faults are called Byzantine faults, and the consensus algorithm in blockchain is designed to handle such faults and strictly guarantee the ledger's consistency. However, Bitcoin-derived consensus algorithms have many defects in performance and scalability, while Byzantine Fault-Tolerance (BFT) algorithms are more applicable in many real scenarios due to its feature of voting mechanism, but they still have some restrictions.

The big restriction is the well-known theorem that BFT system requires a minimum of $3f+1$ replicas to tolerate only f Byzantine faults [3]. The multi-stage broadcast of the BFT algorithms greatly increases network traffic and degrades performance. In addition, the system needs to make full use of diversity to avoid the same software bugs or internal errors [4], which increases hardware/software and management cost.

To reduce cost, improve performance as well as promote the application of the Byzantine system, researchers have conducted extensive research. The *hybrid fault model* is an improved design of the generic Byzantine fault model. It divides the system into two parts, one is *Trusted Component* (TC) with benign faults (i.e., crash failure) occur only, while the other can behave in a Byzantine way. Since the use of TC limits the ability of Byzantine replicas to perform deception, the system can tolerate $\lfloor \frac{N-1}{2} \rfloor$ Byzantine replicas, which greatly reduces system cost and network traffic during algorithm execution. Futhermore, the latest general-purpose processors have provided Trusted Execution Environment (TEE) to ensure security execution of critical application code. The development of related technologies such as Intel Software Guard Extensions (SGX) [5] and ARM TrustZone [6] has greatly promoted the research and application of BFT algorithms based on the hybrid fault model.

Although there have been a body of research on hybrid fault models [7]–[15], at present, these studies are mainly focused on how to design their own algorithms, rather than analyzing design principles and different implementation choices. This paper systematically analyzes and discusses the development and application direction of the BFT algorithms based on the hybrid fault model, which provides guidance for designing and selecting suitable mechanisms and implementation architecture.

The rest of this paper is organized as follows. Section II and section III briefly describes background and system model; section IV discusses the classification and comparison of TC abstractions; section V analyzes different TC implementation scenarios; section VI discusses general framework for BFT algorithm design choices; section VII shows the applications in blockchain; and section VIII concludes this paper.

II. Background

A. Byzantine General Problem

The Byzantine General Problem (BGP) [16] can be described simply as the following story: there are a few of geographically separated forces in an empire, among which some generals are traitors. These traitorous generals can convey arbitrary decisions to interfere with the rest of generals. The critical problem they face is that how to reach an agreement on the attack of enemy army. In essence, BGP is a typical

978-1-5386-4870-4/18/$31.00 ©2018 IEEE

consensus problem under the general Byzantine model and is also a crucial issue that must be addressed in blockchain systems. The system which tolerates Byzantine faults is called the Byzantine system, in which all loyal generals need to follow a certain set of protocols to make consistent decisions.

B. State Machine Replication

Depending on whether the requests must be performed in certain order, BFT techniques can usually be divided into State Machine Replication (SMR) algorithm and quorum Byzantine algorithm [17]. Byzantine SMR algorithm, which ensures all non-faulty replicas to execute requests in the same order by multiple rounds of mutual negotiation, is a common method for dealing with Byzantine faults. The BFT SMR algorithm must guarantee the correctness of the following two attributes [18]:

Safety: all non-faulty replicas execute the requests in the same order (*i.e.*, consensus);

Liveness: clients eventually receive replies to their requests.

The SMR algorithm usually includes three sub-protocols: agreement protocol, checkpoint protocol, and view-change protocol. The agreement protocol generally adopts the primary-backup mode to guarantee that all the requests are committed and executed by correct replicas in a determined order. The checkpoint protocol is used to periodically clear log information and synchronize status. Besides, the goal of the view-change protocol is to replace the current ineffective primary and to ensure requests that have been executed by non-faulty replicas cannot be maliciously eliminated.

C. Equivocation

The core threat to safety and liveness in BFT algorithms is the capacity of ***equivocation***, *i.e.*, the possibility of a Byzantine server sending inconsistent messages to different non-faulty servers or clients. For example, a Byzantine server may attempt to send two different signed messages to different subsets of servers, which impedes the agreement on request order among all correct replicas. Assume that the number of all the replicas and Byzantine replicas in the system is N and f respectively, f non-faulty replicas may be affected by equivocation and fail, ultimately N-f replicas must contain a majority of correct replicas, *i.e.*, $N > 3f$.

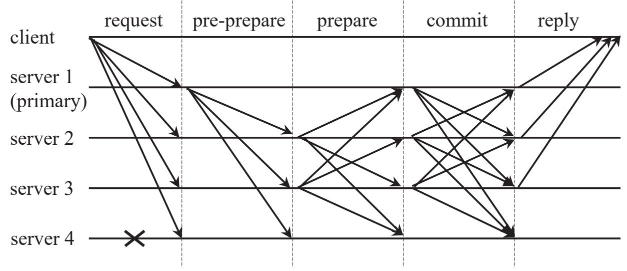

Fig. 1. PBFT normal operation.

Practical Byzantine Fault Tolerance algorithm (PBFT) [19] is a classical SMR algorithm, it contains at least $3f + 1$ total replicas and 5 phases (two full broadcast phases) in the normal case operation. As shown in Fig.1, server 4 is a faulty node, in order to ensure safety and liveness, PBFT specifies that servers must receive at least $2f + 1$ matching legitimate messages (including themselves) during the prepare and commit phases to update the corresponding state, while clients must receive matching legitimate replies from $f + 1$ different servers to determine the execution result of the request.

III. System Model

We assume that a Byzantine system has a group of clients $C = \{C_0, C_1 ...\}$ and N servers $S = \{S_1, S_2 ... S_N\}$ that provide BFT services. In the hybrid fault model, the system needs to satisfy the following conditions:

- Byzantine faults can be arbitrary, but the potential software or internal errors should be independent to prevent possible common-mode attacks;
- The number of Byzantine servers in the system is limited, and it is stipulated that no more than f;
- In an asynchronous environment, data packet discarding and duplication may occur, and the use of retransmission mechanisms ensures that data packets are eventually delivered. However, in order to ensure the liveness of Byzantine system, the system adopts a partial synchronization model to circumvent the FLP impossibility (*i.e.*, there is no consensus solution that can completely tolerate one or more crash failures in a fully asynchronous system) result [20], that is, there exists an unknown upper bound T_{delay} for communication and processing delays of servers.
- Cryptographic techniques are used to ensure the data integrity, such as digital signatures or Message Authentication Codes (MAC), and Byzantine nodes cannot break the standard assumptions that the hash functions are collision-resistant and the signature cannot be forged.
- The Byzantine system contains a secure TC that cannot be invaded and held by the adversary, even local administrators cannot tamper with. Thus, TC provides a TEE to protect internal code secure execution.

By leveraging secure TC, Byzantine servers cannot equivocate or their equivocation can be easily detected by other servers and clients. In fact, the hybrid fault model is more like a trade-off between the Byzantine and benign fault models. The limitation of Byzantine servers' equivocation ability reduces the number of additional f non-faulty replicas required in the generic Byzantine model, raising the system's fault tolerance up to $\lfloor \frac{N-1}{2} \rfloor$, and a more detailed proof can be referred to [21].

IV. Abstractions of Trusted Component

TC is a core component to implement the hybrid fault model, and its implementation logic has different abstractions. Depending on the functions and internal state, we can distinguish three main categories of abstractions: the early TC abstraction, trusted logs, and trusted increment-only counters.

This section will introduce these abstractions in detail, and conclude with a comparative analysis.

A. *Early Trusted Component Abstraction*

The early TC abstraction contains the whole logic responsible for request atomic multicasts, and the TC is usually unconstructed by security hardware. Correia et al. [7], [8] utilized the hybrid fault model, assuming that the system contains a privileged distributed component called *wormhole*. They designed a specic wormhole called Trusted Timely Computing Base (TTCB) with three important features: (1) it is assumed to be secure and cannot be affected by malicious faults, *i.e.*, it can only fail by crashing; (2) the communication between the TTCBs is real-time, which circumvents the FLP impossibility result; (3) it provides a few basic services, including local authentication service and trusted multicast ordering service. Since the TTCB is secure, these critical services are reliable and cannot be affected by malicious faults.

Reiser et al. [9] proposed an efficient proactive recovery approach based on virtualization, using a hybrid fault model to reduce the required replicas. In the adopted VM-FIT architecture, a small and trusted *Domain NV* is separated from *Domain 0* to provide atomic broadcast and proactive recovery services. These services executed in Domain NV are assumed not be affected by malicious faults. SPARE [10] is a similar approach, which also utilizes virtualization technology and the hybrid fault model to improve system resilience and reduce proactive recovery overhead. Setting f replicas to be passive, SPARE requires only $f+1$ active replicas in the fault-free case. Moreover, its TC abstraction is the same as that of VM-FIT, which contains all the logic for ordering requests.

B. *Trusted Logs*

Attested Append-only Memory (A2M) [11] provides an abstraction of trusted logs. Fig.2 illustrates the structure of an A2M. Each node in Byzantine systems is required to be equipped with a TC (*i.e.*, A2M), and this TC contains a series of trusted, undeniable, ordered logs. Each log has its own identifier q_i, and different trusted logs are responsible for attesting the specific category of messages generated during the algorithm. Only the latest log entries are stored in A2M from the lowest slot (in the position L) to the highest slot (in the position H), consisting of a sequence of values X_{L+j} (usually represented by the hash of the message) and cumulative digests of all log entries up to itself.

A2M provides a rich set of interfaces to support removing the ability of performing equivocation. A node can add log entries to trusted logs by executing *append* operation or *advance* operation, and get $LOOKUP$ attestations for certain entries by executing $lookup(q, n, z)$ operation. The $LOOKUP$ attestation contains the state of the log entry in the nth slot of the log q, and only the attestation with $ASSIGNED$ state can prove that this certain entry is indeed appended to this log. Since the trusted logs can only be appended and each log entry is bound to a sequence algorithm instance number, nodes can confirm that their received messages are the same as

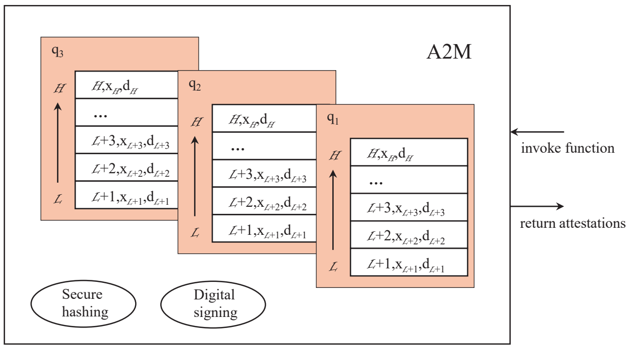

Fig. 2. Structure of an A2M.

long as they receive valid message with matching $LOOKUP$ attestation.

In [11], the author describes two algorithms based on A2M logs. A2M-PBFT-E only uses one local trusted log to protect reply phase of PBFT. Clients have to receive at least $2f+1$ matching execution results and attestations to accept this result. Therefore, Byzantine nodes cannot equivocate about different results to clients, which achieves safety without exceeding $2\lfloor\frac{N-1}{3}\rfloor$ faulty replicas. However, this algorithm cannot ensure liveness when the number of faulty replicas exceeds $\lfloor\frac{N-1}{3}\rfloor$. The other algorithm A2M-PBFT-EA uses five local trusted logs to process different type of algorithm messages including *preprepare/prepare*, *commit*, *checkpoint*, *viewchange*, and *newview* messages. Since all the messages are required to be verified by checking their matching A2M attestations, the equivocation about any type of messages is prevented, and finally, the system resilience is increased to $\lfloor\frac{N-1}{2}\rfloor$.

C. *Trusted Increment-only Counters*

The core problem for SMR is how to make all the correct replicas get the same order for requests. Therefore, the trusted service must provide some functions to limit the ability of Byzantine nodes to declare that different operations are ith valid operation. Obviously, monotonic counters can be employed to address this problem. By assigning unique sequence numbers to each operation, it is ensured that all the valid messages with the same sequence number are identical, namely that a Byzantine node cannot send two different messages with the same count value. Since the assigned sequence numbers must be unforgeable, this abstraction also contains an encryption primitive responding for creating signature certificates that associate the sequence number to the hash of the message. Therefore, TC needs to provide a certificate-creating interface, while the verification of certificates can be executed out of TC if TC uses public keys rather than symmetric keys. As a result, Byzantine nodes' equivocation ability is greatly limited and system resilience is also improved.

This simple abstraction becomes widely used in recent years. MinBFT [12] is designed based on PBFT and has fewer algorithm communication phases and lower minimum of required replicas. Its TC provides a local service called *Unique Sequential Identifier Generator* (USIG) which serves only two functions for creating and verifying certificates. Futhermore, the author also extended MinBFT by leveraging speculative mechanism and proposed a new algorithm MinZyzzyva with only three algorithm phases. ReBFT [13] is a resource-efficient algorithm based on MinBFT, which further reduces resource consumption in the absence of faults without losing the ability to ensure safety and liveness in the presence of faults. In addition, Levin et al. [14] proposed a similar TC called *TrInc* to combat equivocation in large, distributed systems. TrInc provides some extra interfaces, including creating and deleting counters, importing and using symmetric keys, etc. It supports multiple independent counters to provide richer functions, and the author also presented how to build A2M using TrInc with lower complexity.

D. Analysis of Diverse TC Abstractions

As illustrated in Tab.1, we analyze the differences of the above three abstractions from the aspects of complexity, versatility, and the approaches to deal with equivocation.

TABLE I
COMPARISON OF THE THREE TC ABSTRACTIONS

	Complexity	***Versatility***	***Deal with equivocation***
The early TC abstraction	High	Low	Prevention
Trusted logs	Medium	High	Prevention
Trusted increment-only counters	Low	Medium/High	Detection

Complexity. The security of TC is more easily to be proved and guaranteed if the logic inside TC is simple enough. Since a lot of security hardware that provides trusted computing service is limited by its own storage capacity, a low-complexity abstraction can also contribute to the development and deployment of TC. The early TC abstraction contains all the logic responsible for atomic broadcasts, so its complexity is high and the security can be easily compromised. The abstraction of trusted logs is less complex and more secure than the early TC abstraction. However, these active logs are part of the A2M's state, which might restrict the trusted order service due to the decreasing available storage capacity. Although A2M provides *truncate* function to discard obsolete log entries, opportunities to truncate the log may be limited by the algorithm. Compared to A2M, the abstraction of trusted increment-only counters has lower complexity and less coupling between TC and distributed algorithms, so it is easier to design and deploy.

Versatility. Versatility depends on the functionality and extensibility provided by the TC abstractions. The early TC abstraction is more biased towards customization, such as the execution of crucial code written in the kernel, so its versatility is poor. A2M provides more functions to support high versatility, while the counter-based abstraction provides fewer interfaces than A2M, but it can utilize multiple counters to improve versatility and support a wider range of applications.

Approaches. Approaches to deal with equivocation can be divided into prevention and detection. Preventing equivocation can further restrict malicious behavior and simplify algorithm design. Since the early TC abstraction contains the whole logic for ordering requests and TC is assumed to be secure, equivocation is prevented directly. Similarly, the specific ith log slot in A2M can only store a log entry with the same sequence number which is also equal to the algorithm instance number i, so Byzantine nodes cannot produce two different messages M_1 and M_2 for a certain algorithm instance with matching valid attestations. Therefore, A2M can also prevent equivocation. However, in the counter-based abstraction, a Byzantine node may still send two different messages for algorithm instance i with valid certificates, but the messages' assigned counter value is different. Therefore, this abstraction does not directly prevent equivocation but can detect equivocation by comparing the assigned counter value or processing the received message in the order of counter values. Ultimately, all the above approaches eliminate Byzantine nodes' ability to equivocate without being convicted by others.

Our choices. An important principle goal of TC abstractions is to provide an interface for a widely applied expression, while restricting as much as possible to a small Trusted Computing Base (TCB). Considering the comprehensive advantages of the counter-based abstraction in complexity and versatility, we believe this abstraction is most promising, and a series of recent works [12]–[15] also confirm the trend. In addition, we can use the counter values instead of the dedicated message sequence number to determine the message order, and use a two-phase ordering protocol to reduce the total algorithm phases. Although more considerations are needed to ensure that the historical state can be safely passed to the next view during view change phrase, the above improved approach can effectively enhance the overall performance.

V. Implementation Scenarios of TC

The TCB of a computer system is the set of all hardware, firmware, and/or software components, the combination of which is responsible for enforcing a computer security policy [28]. In addition to the abstractions, the implementation scenarios of TC are also different, resulting in different TCB size and complexity. Fig.3 illustrates six diverse implementation scenarios. In this section, we analyze their differences from the following three levels, *i.e.*, application level, operating system (OS) level, and hardware level, finally we discuss our choices.

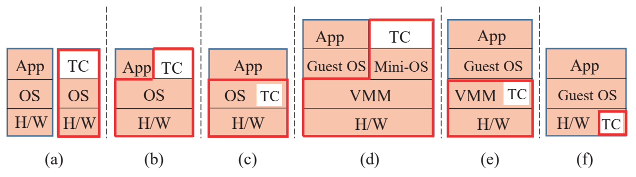

Fig. 3. The TC implementation scenarios: (a) trusted service, (b) trusted software isolation, (c) hardened OS kernel, (d) trusted VM, (e) trusted VMM, and (f) security hardware.

Application level. TC can be implemented as a software abstraction, and the implementation scenarios could be a trusted service offered by trusted providers (Fig.3(a)) or a software-isolated module (Fig.3(b)). The former implementation approach is similar to the notarization approaches [22], and the trusted providers can be the third parties such as trusted organizations or enterprises. However, the big drawback of this approach lies in its time-consuming remote communication, resulting in poor system performance. The latter implementation approach uses software isolation technique to ensure security. For instance, it can take advantage of programming language type and memory safety for isolation. Since the software-isolated processes in TC are run in the same address space, this approach has better processing capacity than trusted services.

OS level. Fig.3(c) illustrates that the protected crucial code can be executed in a hardened OS kernel, and this approach has been applied in [7]. The virtual layer is a simple method to achieve fault isolation. As shown in Fig.3(d), TC is separated from the main application by deploying the TC on a verifiable mini-OS on top of a virtual machine monitor (VMM). The TC's execution results can be trusted if the VMM and the micro-OS is guaranteed to be secure and reliable. VM-FIT and SPARE both adopt this implementation approach. In [12], MinBFT is implemented through many different approaches, and its evaluation results show that the achieved high performance of approach (d) is close to approach (b).

Futhermore, the VMM can also be considered as a microkernel. Fig.3(e) illustrates another implementation scenario with a smaller trusted footprint. Since the TC implementation is put in the VMM, potential guest OS or application errors above the VMM cannot affect the TC's operation. Xen hypervisor can be utilized to easily implement the virtualization-based approaches in (d) and (e). To ensure that the TC's state is kept during replicas reboot, the state should be promptly stored in the non-volatile, high-speed storage such as flash memory.

Hardware level. Fig.3(f) describes the implementation with security hardware. This implementation approach is widely applied in recent years. [14] implements a TC called *Trinc* on a smartcard which is connected to a PC via a USB card reader. [13], [23] uses the FPGA-based CASH subsystem that preprograms necessary functions on the FPGA. The Trusted Platform Module (TPM) chips developed by the Trusted Computing Group [24] are commonly used to protect the authenticity and confidentiality of data and code. Some versions of TPM directly support usable monotonic counters, so it is suitable for implementation of the trusted monotonic counter based abstraction [12], [25]. Futhermore, some innovative trusted computing technologies are rapidly developed and widely applied, including SGX and Trustzone. SGX protects the code and data in enclaves from being tampered with or monitored by other applications and high-level system software such as OS and VMM. Compared with Trustzone, SGX achieves a smaller TCB that only includes the processor and the enclave, so its security level is higher.

Our choices. The TC implementation scenarios have different trusted footprints. Red boxes in Fig.3 delineate the TCB. Obviously, TC with smaller TCB is considered to be more secure and difficult to be attacked. The security hardware prevents TC from being affected by external attacks and server operators malicious operations, which ensures the TC security without physical attacks. It achieves smaller trusted footprint, so we think the design of TC from hardware level is more dependable. However, TC is required to be accessed several times in each algorithm instance, which greatly increases the latency and may become a potential bottleneck. Some optimized solutions are considered in practical application, such as using symmetric encryption instead of asymmetrical encryption, using fast persistent memory to store state and possible certificates. Moreover, numerous researches on SGX technology have been carried out to further improve its security [26] and reduce the internal overhead. We believe that SGX is promising and can be widely applied in the future.

VI. General Framework for BFT Algorithm Design Choices

The TC assistance mechanism has strong applicability and flexibility, and can be combined with other mechanisms to design more efficient BFT algorithms based on the hybrid fault model. Fig.4 illustrates different BFT algorithm design choices. In this section, we analyze their differences and features, and compare their performance and resilience.

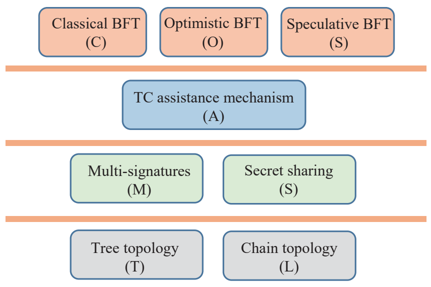

Fig. 4. BFT Algorithm Design Choices.

BFT paradigms. BFT paradigms can be divided into three categories: classical BFT (*e.g.*, PBFT), optimized BFT (*e.g.*, CheapBFT [23]) and speculative BFT (*e.g.*, Zyzzyva [27]). The classic BFT can tolerate non-primary faults without requiring additional operations, so it has high resilience. But it requires all replicas to participate in the agreement stage, which leads to poor performance. The speculative BFT allows replicas to speculatively execute the request and send the result to the client. It has good performance in the fault-free case due to the absence of mutual broadcast and its lower phrase number. However, it simply relies on the client's feedbacks and rollback mechanisms to ensure safety and liveness, which is unpractical in real scenarios since the client can be malicious. The optimized BFT provides a trade-off between performance and resilience, and its core idea is to improve performance in the fault-free case. It distinguishes replicas into active and passive replicas, and passive replicas only participate in the agreement stage when any fault occurs.

TC assistance. We have previously discussed the abstractions and implementation scenarios of TC. The adoption of TC (generally implemented by security hardware) removes the ability of equivocation and reduces the minimum of replicas, so the system performance can be effectively improved. Also, TC assistance mechanism (A) can be flexibly combined with different BFT paradigms. For example, MinBFT is the combination of C and A, REBFT is the combination of O and A, and MinZyzzyva is the combination of S and A.

Message aggregation. Normally, the message complexity of BFT algorithms is $O(N^2)$ since each replica needs to broadcast a prepare or commit message to all (active) replicas. By using message aggregation techniques, such as multi-signatures (M) and secret sharing (S), message complexity can be reduced to $O(N)$. Multi-signatures requires larger message sizes and longer processing time, so its performance is limited. As for secret sharing, it brings new security risk since a Byzantine primary can equivocate by using the same secret for different requests. To address this problem, it also needs to use TC assistance mechanism to generate and split secrets, deliver and release shares securely [28].

Communication topology. We can use tree topology (T) or chain topology (L) with fault detection to further enhance communication efficiency. Also, we can combine the above mechanisms and topologies to design various algorithms, such as *OML* (*e.g.*, BChain [29]), *CMT* (*e.g.*, ByzCoin [30]) and *OAST* (*e.g.*, FastBFT [28]). These algorithms greatly improve the performance compared to PBFT, but the resilience of some algorithms may be compromised. For example, BChain needs to initiate re-chaining process when non-primary faults occur.

The rational design of BFT algorithms requires comprehensive consideration of performance, scalability, implementation costs, and resilience. Although it is difficult to achieve the best in all aspects, we can make some necessary trade-offs for diverse application scenarios and make full use of the flexibility of these design choices.

VII. Applications in Blockchain

Blockchain is an innovative distributed ledger technology that establishes trust via running algorithms by all participants to reach consensus on same order of transactions and serve a tamper-resistant digital ledger. Because of its attractive features, blockchain is widely applicable to many business applications where there is a requirement of trust among multiple parties. In this section, we will discuss the applications of TC assistance mechanism and the hybrid fault model in building trusted blockchain applications and designing efficient consensus algorithms.

A. Building Trusted Applications

Trusted computing is a broad term that refers to technologies for resolving computer security problems through hardware enhancements and associated software modifications. Recently, blockchain and trusted computing are getting increasingly connected, and their combination is commonly utilized to build trusted applications. TM-Coin [31] is a trustworthy management of TCB measurements for IoT devices, which enables veriers to securely perform remote attestation of sensed data received from the devices by leveraging TrustZone and blockchain. Town Crier [32] is an authenticated data feed system for smart contracts, serves datagrams with a high degree of trustworthiness by combining a smart-contract front end in Ethereum [33] and an SGX-based trusted hardware back end.

B. Designing Efficient Consensus Algorithms

The application and promotion of blockchain systems are still limited by the high consumption and low performance of the underlying consensus algorithms. For example, Bitcoin uses *Proof-of-Work* (PoW) mechanism, which consumes a large amount of global resources but can only process nearly 7 transactions per second (TPS) [34]. Other cryptocurrencies like PPCoin [35] adopts *Proof-of-Stake* (PoS) mechanism to limit the hashing power of each node, but their TPS still cannot satisfy the requirement of many real-time applications.

TABLE II
COMPARISON OF FIVE ALGORITHMS

	PoW	***PoS***	***PoET***	***PBFT***	***Hybrid fault model based BFT***
Resilience	50%	50%	$\Theta(\frac{loglogN}{logN})$ [37]	33%	50%
TPS	<100	<1000	<2000	<2000	<10000
Scalability	strong	strong	strong	weak	weak

To design more energy-efficient consensus algorithms, TC assistance mechanism can be leveraged. For instance, *Proof-of-Elapsed-Time* (PoET) algorithm [36] utilizes SGX to implement a leader-election lottery system. Instead of using computational effort to solve cryptographic puzzles, PoET uses a TEE to generate random wait times. Compared with PoW and PoS, PoET is more efficient and fair, and it achieves the goal of “one CPU one vote”. Tab.2 shows the comparison of the five consensus algorithms.

Hybrid fault model based BFT algorithms are suitable for application in consortium blockchain. Many platforms [38] use PBFT to ensure consistency, but it can only tolerant 33% Byzantine faults. Since the design choices can be diversified, we can design more efficient and resilient algorithms by referring to the general framework in Section VI. Performance and scalability are significant indicators for blockchain consensus algorithms, and blockchain systems can leverage multi-core CPUs and trusted hardware to boost these two metrics. For instance, Hybster [15] is a new hybrid SMR algorithm that is highly parallelizable and specified formally. The consensus algorithms based on the hybrid fault model can effectively improve system resilience from 33% to 50% and reduce network traffic. As for new trusted computing technologies, the Windows SDK [39] and programming manual [40] for SGX have been released. With the development of these technologies, the application of TC assistance mechanism and the hybrid fault model in blockchain has more possibilities.

VIII. Conclusion

In this paper, we analyze the design choices and applications of TC assistance mechanism and BFT algorithms based on the hybrid fault model. We classify and compare different TC abstractions and implementation scenarios, and propose a flexible design framework. By leveraging TC assistance mechanism, we can guarantee off-chain secure data processing and build reliable blockchain systems. Also, blockchain consensus algorithms based on this model can achieve 50% Byzantine fault tolerance and higher transaction throughput compared with PBFT. However, there are still many open issues, including the security and performance optimization of TC, the design of efficient blockchain consensus algorithms and scalable architectures. Our future work will focus on the better application of these mechanisms in blockchain, and designing more efficient systems under future network architectures, such like Named Data Networking [41].

Acknowledgment

This work has been financially supported by Shenzhen Key Fundamental Research Projects (No. JCYJ20170412150946024, JCYJ20170412151008290).

References

[1] Bitcoin. https://bitcoin.org/en/.

[2] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, "Eclipse attacks on bitcoins peer-to-peer network," USENIX Security Symp. pp. 129-144, 2015.

[3] G. Bracha and S. Toueg, "Asynchronous consensus and broadcast protocols," J. of the ACM, vol. 32, pp. 824-840, 1985.

[4] M. Garcia, A. Bessani, I. Gashi, N. Neves, and R. Obelheiro, "OS Diversity for Intrusion Tolerance: Myth or Reality," Proc. Int'l Conf. Dependable Systems and Networks, 2011.

[5] F. McKeen, et al. "Innovative instructions and software model for isolated execution " Proc. 2nd HASP Wksp. 2013.

[6] ARM. Security technology building a secure system using TrustZone technology (white paper). ARM Limited, 2009.

[7] M. Correia, N. F. Neves, and P. Verissimo, "How to tolerate half less one Byzantine nodes in practical distributed systems," Proc. 23rd Symp. SRDS, pp. 174-183, 2004.

[8] M. Correia, N. F. Neves, L. C. Lung, and P. Verissimo, "WormIT - A wormhole-based intrusion-tolerant group communication system," J. of Systems and Software, vol. 80, pp. 178-197, 2007.

[9] H. P. Reiser and R. Kapitza, "Hypervisor-based efficient proactive recovery," Proc. 26th Symp. SRDS, pp. 83-92, 2007.

[10] T. Distler, R. Kapitza, I. Popov, H. P. Reiser, and W. Schroder, "SPARE: Replicas on hold," Proc. 18th NDSS, pp. 407-420, 2011.

[11] B.-G. Chun, P. Maniatis, S. Shenker, and J. Kubiatowicz, "Attested append-only memory: Making adversaries stick to their word," Proc. 21st Symp. SOSP, pp. 189-204, 2007.

[12] G. S. Veronese, M. Correia, A. N. Bessani, L. C. Lung, and P. Verissimo, "Efficient byzantine fault-tolerance," IEEE Trans. Comput., vol. 62, pp. 16-30, 2013.

[13] T. Distler, C. Cachin, and R. Kapitza, "Resource-efficient Byzantine fault tolerance," IEEE Trans. Comput., vol. 65, pp. 2807-2819, 2016.

[14] D. Levin, J. R. Douceur, J. R. Lorch, and T. Moscibroda, "TrInc: Small trusted hardware for large distributed systems," Proc. 6th Symp. NSDI, 2009.

[15] J. Behl, T. Distler, and R. Kapitza, "Hybrids on steroids: Sgx-based high performance bft," Proc. 12th EuroSys, 2017.

[16] L. Lamport, R. Shostak, and M. Pease, "The Byzantine generals problem," ACM Trans. on Programming Languages and Systems, vol. 4, pp. 382-401, 1982.

[17] M. Abd-El-Malek, G. Ganger, G. Goodson, M. Reiter, and J. Wylie, "Fault-scalable byzantine fault-tolerant services," In Proc. of SOSP, 2005.

[18] L. Lamport, "The part-time parliament," ACM Trans. Comput. Syst., 1998.

[19] M. Castro, B. Liskov, and M. Pease, "Practical Byzantine fault tolerance and proactive recovery," ACM Trans. Comput. Syst,, vol. 20, pp. 398-461, 2002.

[20] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," J. of the ACM, vol. 32, pp. 374-382, 1985.

[21] A. Clement, F. Junqueira, A. Kate, R. Rodrigues, "On the (limited) power of non-equivocation," In Proc. of the 2012 ACM Symp. Principles of distributed computing, pp. 301-308, 2012.

[22] A. R. Yumerefendi and J. S. Chase, "Strong accountability for network storage," In Proc. of USENIX FAST, 2007.

[23] R. Kapitza, et al. "CheapBFT: resource-efficient byzantine fault tolerance," In Proc. of the 7th EuroSys, pp. 295-308, 2012.

[24] Trusted Computing Group (TCG). http://www.trustedcomputinggroup.org/.

[25] G. S. Veronese, M. Correia, A. N. Bessani, and L. C. Lung, "EBAWA: Efficient Byzantine agreement for wide-area networks," IEEE 12th Int'l. Symp. HASE, pp. 10-19, 2010.

[26] S. Matetic, et al. "ROTE: Rollback Protection for Trusted Execution," IACR Cryptology ePrint Archive, 2017.

[27] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong, "Zyzzyva: Speculative Byzantine Fault Tolerance," Proc. 21st Symp. Operating Systems Principles, 2007.

[28] J. Liu, W. Li, G. O. Karame, and N. Asokan, "Scalable Byzantine Consensus via Hardware-assisted Secret Sharing," arXiv preprint arXiv:1612.04997, 2016.

[29] S. Duan, H. Meling, S. Peisert, and H. Zhang, "BChain: Byzantine replication with high throughput and embedded reconfiguration," In Int'l. Conf. on Principles of Distributed Systems, pp. 91-106, 2014.

[30] E. K. Kogias, et al. "Enhancing Bitcoin security and performance with strong consistency via collective signing," In 25th USENIX Security Symposium, Aug. 2016.

[31] J. Park and K. Kim, "TM-Coin: Trustworthy management of TCB measurements in IoT," IEEE Int'l. Conf. PerCom Wksp. pp. 654-659, 2017.

[32] F. Zhang, E. Cecchetti, K. Croman, A. Juels, and E. Shi, "Town crier: an authenticated data feed for smart contracts," In Proc. of the 2016 ACM SIGSAC Conf. Comput. Commun. Security, pp. 270-282, 2016.

[33] Ethereum. https://www.ethereum.org/.

[34] A. Gervais, et al. "On the security and performance of proof of work blockchains," In Proc. of the 2016 ACM SIGSAC Conf. Comput. Commun. Security, pp. 3-16, 2016.

[35] Peercoin whitepaper. https://peercoin.net/whitepaper/.

[36] Hyperledger Sawtooth. https://www.hyperledger.org/projects/sawtooth/.

[37] L. Chen, et al. "On Security Analysis of Proof-of-Elapsed-Time (PoET)," In Int'l. Symp. on Stabilization, Safety, and Security of Distributed Systems, pp. 282-297, 2017.

[38] Hyperledger Fabric. https://www.hyperledger.org/projects/fabric/.

[39] Intel: Intel(R) Software Guard Extensions SDK. https://software.intel.com/en-us/sgx-sdk/.

[40] Intel: Intel(R) Software Guard Extensions Programming Reference, Revision 2. https://software.intel.com/sites/default/files/managed/48/88/329298-002.pdf

[41] K. Lei, S. Zhong, F. Zhu, K. Xu and H. Zhang, "A NDN IoT Content Distribution Model with Network Coding Enhanced Forwarding Strategy for 5G," IEEE Trans. Industrial Informatics, vol. 14, pp. 2725-2735, 2017.