Trusted Hardware-Assisted Leaderless Byzantine Fault Tolerance Consensus

Zhao, Liangrong ; Decouchant, Jérémie; Liu, Joseph; Lu, Qinghua ; Yu, Jiangshan

DOI
10.1109/TDSC.2024.3357521

Publication date
2024

Document Version
Final published version

Published in
IEEE Transactions on Dependable and Secure Computing

Trusted Hardware-Assisted Leaderless Byzantine Fault Tolerance Consensus

Liangrong Zhao, Jérémie Decouchant, Joseph K. Liu, *Senior Member, IEEE*, Qinghua Lu, *Senior Member, IEEE*, and Jiangshan Yu

***Abstract*—Byzantine Fault Tolerance (BFT) Consensus protocols with trusted hardware assistance have been extensively explored for their improved resilience to tolerate more faulty processes. Nonetheless, the potential of trust hardware has been scarcely investigated in leaderless BFT protocols. RedBelly is assumed to be the first blockchain network whose consensus is based on a truly leaderless BFT algorithm. This paper proposes a trusted hardware-assisted leaderless BFT consensus protocol by offering a hybrid solution for the set BFT problem defined in the RedBelly blockchain. Drawing on previous studies, we present two crucial trusted services: the counter and the collector. Based on these two services, we introduce two primitives to formulate our leaderless BFT protocol: a hybrid verified broadcast (VRB) protocol and a hybrid binary agreement. The hybrid VRB protocol enhances the hybrid reliable broadcast protocol by integrating a verification function. This addition ensures that a broadcast message is verified not only for authentication but also for the correctness of its content. Our hybrid BFT consensus is integrated with these broadcast protocols to deliver binary decisions on all proposals. We prove the correctness of the proposed hybrid protocol and demonstrate its enhanced performance in comparison to the prior trusted BFT protocol.**

***Index Terms*—Reliable broadcast, byzantine fault tolerance, trusted execution environment, trusted services.**

I. Introduction

Byzantine fault tolerant (BFT) consensus algorithms have been widely explored in blockchain networks for their built-in finality and high throughput capacities [1], [2], [3]. Nonetheless, most BFT protocols have been challenged by their high communication complexity and limited scalability [4]. To tackle these issues, an increasing number of BFT protocols have been proposed aiming for improvements in communication complexity and scalability [5]. The Practical Byzantine Fault Tolerance (PBFT) protocol is considered the first BFT algorithm for practical use [6]. PBFT forms the basis of numerous permissioned consensus protocols for blockchain networks, including Tendermint [7], Streamlet [8], HotStuff [9], and Honeybadger [10]. Most of these protocols operate under a leader-based system, executing in a series of views, with each view being led by a designated process as a leader responsible for coordinating all consensus decisions. However, a faulty leader can effectively halt leader-based BFT protocols, requiring a view-change mechanism to replace the defective leader and ensure continued progress. Current techniques to implement view changes tend to be redundant and bug-prone [11], [12].

To circumvent the single-point failure caused by the fault of the leader, BFT protocols featuring a leaderless design have emerged [13], [14], [15]. The leaderless design has been extensively applied in asynchronous BFT protocols to achieve an asynchronous common subset (ACS) [16]. Ben-Or et al. set a standard for ACS by proposing a practical common subset agreement protocol that includes both a broadcast instance and an agreement instance for every party in the network [17]. This ”broadcast+agreement” paradigm is also applied in RedBelly, which is assumed to be the first blockchain network based on a genuine leaderless BFT algorithm [14]. RedBelly's goal is to solve the set BFT consensus problem, which is a variant of the BFT consensus defined by RedBelly, to reach an agreement on the ordered common subset. Each process is supposed to broadcast a proposal before reaching a consensus on every proposal through binary Democratic BFT (DBFT) [18].

RedBelly presents verified reliable broadcast (VRB), a variation of the Byzantine reliable broadcast (BRB) that incorporates verification within message dissemination to effectively filter out any invalid transactions in a proposal batch during broadcasting. However, this advancement still preserves the message complexity inherent to the legacy double-echo BRB protocol. It requires 2 rounds of all-to-all communication, leading to a complex and extensive message exchange pattern with N concurrent instances [19]. The potentially high system load is still an open challenge in designing leaderless BFT protocols [20].

While leaderless designs offer solutions to the leader's single-point failure, approaches employing trusted hardware to enhance resilience have also been explored. The *hybrid* BFT protocols with trusted hardware assistance aim to improve resilience by restricting Byzantine processes and to reduce communication complexity through the simplification of the protocol [21], [22], [23], [24], [25]. The use of trusted hardware in BFT protocols is to circumvent the impossibility of an asynchronous or partially synchronous consensus protocol to tolerate f Byzantine faults

Manuscript received 6 April 2023; revised 2 January 2024; accepted 6 January 2024. Date of publication 22 February 2024; date of current version 13 November 2024. This work was supported by Australian Research Council (ARC) under Grant DE210100019. *(Corresponding author: Jiangshan Yu.)*

Liangrong Zhao and Joseph K. Liu are with the Faculty of Information Technology, Monash University, Clayton, VIC 3800, Australia (e-mail: liangrong.zhao@monash.edu; joseph.liu@monash.edu).

Jérémie Decouchant is with the Delft University of Technology, 2628, CD Delft, The Netherlands (e-mail: j.decouchant@tudelft.nl).

Qinghua Lu is with the Data61, CSIRO, Canberra, ACT 2601, Australia (e-mail: qinghua.Lu@data61.csiro.au).

Jiangshan Yu is with the University of Sydney, Camperdown, NSW 2050, Australia (e-mail: jiangshan.yu@monash.edu).

Digital Object Identifier 10.1109/TDSC.2024.3357521

with less than $3f+1$ total processes [26]. The trusted hardware provides solutions to eliminate equivocation to prevent processes from sending conflicting messages to different recipients. With the non-equivocation mechanism and digital signatures, a crash-fault protocol can be compiled into a Byzantine-fault protocol with the same amount of faulty processes in the asynchronous communication model [27], [28]. Efforts have also been made to construct a trusted BRB primitive with $N \geq 2f+1$ [24], [25], [29]. While hybrid solutions have been extensively explored in the context of traditional leader-based BFT protocols, aiming to mitigate the adverse effects of an arbitrary leader, they have been less investigated in the context of leaderless protocols.

Contributions: This paper proposes a trusted hardware-assisted leaderless BFT consensus algorithm, enabling the execution of parallel broadcast and consensus instances. We highlight our main contributions as follows.

- *We provide a trusted leaderless BFT consensus algorithm by providing a hybrid solution for the set BFT consensus problem.* Essentially, the set BFT problem is for each process to concurrently propose a proposal and then decide on a commonly agreed subset of all proposed proposals. Our approach incorporates two main primitives: a hybrid verified reliable broadcast (VRB) protocol and a hybrid binary consensus.
- *We propose a hybrid verified reliable broadcast protocol based on our implementation of a hybrid BRB protocol.* The VRB protocol is a verified variant of the reliable broadcast protocol with a verification function, ensuring each broadcast message is checked for both authentication and the correctness of its content. Our hybrid VRB leverages hardware assistance while still positioning the verification function outside the TEE, ensuring the trusted hardware remains lightweight.
- *We introduce a hybrid BFT consensus to facilitate the parallel binary consensus invocations in the leaderless design framework.* Our hybrid BFT consensus is structured following the broadcast protocols, providing binary decisions on all proposals to determine their inclusion in the final set.
- We provide security analysis and complexity evaluation to demonstrate the optimal resilience and improved message complexity of the proposed leaderless consensus protocol. We show that our hybrid approach attains optimal resilience with an overall message complexity of $O(2n^2+2n^3)$ for $O(n)$ parallel proposals, and $O(2n+2n^2)$ for each proposal. Additionally, we conduct a comparative simulation evaluation against the prominent open-source trusted protocols MinBFT.

II. Related Works and Background

A. Related Works

1) Byzantine Reliable Broadcast: The original Bracha's double-echo BRB protocol incurs a total communication cost of $O(n^2|M|)$ where $|M|$ is the size of the broadcast message. Multiple following works have endeavoured to reduce this communication cost. The original BRB protocol does not impose computational bounds on adversaries. The following works predominantly utilise cryptographic primitives to simplify the message exchange pattern or diminish the size of the messages being exchanged. An improved version of BRB with a communication cost of $O(n|M| + \kappa n^2 logn)$ assumes a computational bound limiting the adversary's capabilities to break a collision-resistant hash function. κ is the hash function's output size [30]. The improvement is achieved by reducing the rounds of full-size message propagation as the message size is typically much larger than the total number of processes n.

TABLE I
Comparison of BRB Schemes

Protocols	N	Communication rounds	Message complexity
Bracha [19]	$3f+1$	3	$O(n+2n^2)$
Cachin-Tessaro [30]	$3f+1$	4	$O(n^2 \log n)$
Patra [36]	$3f+1$	11	$O(n+n^4 \log n)$
Das [33]	$3f+1$	4	$O(n+\kappa n^2)$
EFBRB [32]	$3f+1$	9	$O(n+n^2 \log n)$
Correia [29]	$2f+1$	2	$O(n+n^2)$
Aguilera [24], [25]	$2f+1$	3	$O(n+2n^2)$
Hybrid BRB/VRB	$2f+1$	2	$O(n+n^2)$

An erasure coding scheme-assisted message exchange is also explored where each element of a codeword can be verified as proof of correctness for the transmitted messages. [31]. Limitations of the state-of-the-art BRB protocols include unbalanced communication cost for broadcaster and recipients [32], inefficiency in the computation for the models relying on mathematical coding algorithms [16], [33], and practicality in a partially connected network [34].

While $3f+1$ is the minimum number of processes for Bracha's BRB to operate with the presence of f faulty processes, it has been shown that a reduction from $3f+1$ to $2f+1$ is attainable when equipped with a non-equivocation mechanism [29], [35]. Correia et al. propose a prototype for a reliable broadcast protocol designed to facilitate the transformation of crash consensus algorithms into Byzantine consensus algorithms. Their approach includes a hybrid BRB protocol with $2f+1$ processes, each having access to a generalized trusted service *wormhole*. This wormhole service allows each process to generate signatures with identifiers, thereby preventing equivocation [29].

In a separate development, Remote Direct Memory Access (RDMA) with SWMR registers is used to implement a reliable broadcast protocol primitive. This approach ensures processes cannot equivocate by cross-referencing the data preserved in shared memory [24]. A subsequent work further improves this approach by providing a consistent broadcast variant for the fast consensus path with fewer amounts of signatures required. Notably, both of the two shared memory-based reliable broadcast protocols rely on an "Init-Echo-Ready" structure with 2 all-to-all communication rounds [25]. A comparison of related works on reliable broadcast protocols can be found in Table I.

2) Hybrid BFT Protocols With Hardware Assistance: Correia et al. introduce the Trusted Timely Computing Base (TTCB), a tamper-proof distributed oracle. It delivers authentication and ordering services by assigning order numbers to messages, enabling the implementation of an intrusion-tolerant atomic multicast service with $2f+1$ processes [37]. The Attested Append-Only Memory (A2M) adopts an alternative approach to prevent equivocation by providing a set of trusted, undeniable, ordered logs. A message has to be logged in the A2M for a sequenced attestation to be generated before transmission and to be verified upon receipt. PBFT-EA leverages A2M to enhance Correia's trusted service by transitioning it from a distributed computing base to a tamperproof component at processes' local devices [38]. TrInc is a lightweight trusted incrementer designed to improve upon the trusted logs of A2M. TrInc mitigates equivocation with great versatility in large-scale distributed systems as TrInc's core functional elements are available in many modern PCs [39]. Hybster later introduces a customized abstraction of TrInc, termed TrInx, which incorporates the ability to generate diverse certificate types for parallelization [40]. Different from TrInc, where counter values are predefined during certificate creation, MinBFT utilizes a simple trusted service named the "Unique Sequential Identifier Generator" (USIG) to assign monotonic, and sequential identifiers to messages at each certification. USIG ensures non-equivocation and reduces three communication rounds down to two during regular operations [23]. CheapBFT employs a unique FPGA-based trusted subsystem to prevent equivocation by issuing unique message certificates for protocol messages. Specially, CheapBFT provides optimistic operation with a lower number $f+1$ of active processes while f other processes can choose to be passive without hindering the normal operations [41]. FastBFT shares similarities with CheapBFT in terms of the monotonic counter setup as well as optimistic operations. Nonetheless, FastBFT additionally incorporates a hardware-based secret-sharing mechanism to reduce message complexity [42]. Damysus is a hybrid streamlined BFT protocol that improves resilience and communication overhead of streamlined HotStuff protocol with augmenting trusted counters with additional secure storage as the authors demonstrate a simple monotonic counter is inadequate to improve resilience for streamline protocols [21].

Some protocols take a different path to adopt the message-and-memory (M&M) model to ensure non-equivocation. The M&M model is a generalization of both message-passing and shared memory. RDMA is a hardware technology that permits memory access to other hosts in the network without involving their CPUs. It provides communication primitives for processes in M&M models to directly write or read remote shared memory, facilitating cross-checking to prevent non-equivocation [25]. Aguilera et al. provide an algorithm with RDMA-based BRB we discussed in the previous subsection. The algorithm considers both crash and Byzantine fault tolerance and it can tolerate a minority of Byzantine faults in shared-memory [24]. Frugal Byzantine Computing advances the communication complexity bounds off the fast path by presenting a consistent broadcast primitive with cheaper signature cost compared to the reliable broadcast [25]. uBFT relies on RDMA-based trusted disaggregated memory, which is encapsulated within a consistent broadcast primitive where the partial order of the messages from a process is ordered. This broadcast primitive in uBFT improves upon Frugal Byzantine Computing's version by offering a signatureless fast-path operation [43].

A prevalent trend in trusted protocols is to assign unique identifiers to protocol messages, ensuring that conflicting messages are either prevented or detectable. The foundation of trusted services in distributed systems has undergone transitions: from a tamperproof computing base to append-only logs, and further to lightweight monotonic counters and message-and-memory (M&M) models [21], [24]. In this work, the minimal trusted services required for our design align with Damysus' configuration, including a monotonic counter and a trusted log.

3) Leaderless BFT Models: Most BFT algorithms involve a leader that helps processes converge towards a decision in a fast manner, which makes the correctness of the leader crucial for most of the BFT models. When the leader is Byzantine, a leader-changing procedure is required as most of the algorithms are unable to proceed in the absence of a valid leader [44]. The time-consuming leader-changing mechanisms inspire recent efforts devoted to minimising the role of a leader, and one of the ideas is to entirely eliminate the role of a leader [45]. Recent advancements in Byzantine atomic broadcast protocols involve directed acyclic graph (DAG) based approaches, which achieve consensus by utilizing a shared DAG data structure among the honest processes via reliable broadcast, enabling the processes to agree on the order of transactions in the DAG [46], [47], [48]. The role of a leader in DAG-based BFT has been weakened due to the parallelizable nature of DAG to deprecate the need for complex view-change mechanisms [46], [47], [49]. Fewer efforts have been made on the true leaderless BFT [14], [15], [50], and Dester is the only leaderless hybrid BFT protocol with trusted components [13]. However, the underlying leaderless consensus protocol Dester is built on is challenged with safety and consistency violations [51].

B. The Set Byzantine Consensus Problem and RedBelly's Solution

The Set Byzantine consensus (SBC) problem is formalized by the RedBelly blockchain for correct processes to reach an agreement for parallel consensus instances [14]. In a network where all processes are supposed to propose a proposal, SBC allows all correct processes to eventually agree on a common set merged from correct proposals.

SBC is significant for leaderless BFT models as it enables a scheme for the execution of multiple BFT instances in parallel with transactions originating from different processes. RedBelly's solution incorporates a variant of double-echo reliable broadcast, a binary BFT algorithm for parallel processing, and a reconciliation protocol to merge the outputs from parallel BFT instances into a final block. The main focus of this work is to improve the broadcast protocol and binary consensus algorithm by leveraging trusted hardware.

1) Verified Reliable Broadcast: Verified reliable broadcast lets all processes in the set BFT exchange their proposals with each other and get their proposals verified at the same time prior to joining the binary consensus instances. The VRB implementation is derived from Bracha's double-echo reliable broadcast protocol, augmented with an additional verification function during the second round of all-to-all communication [19]. In contrast to the Bracha protocol, RedBelly's VRB mandates that a proposal (which is equivalent to a message in BRB) must undergo verification against blockchain rules before being endorsed in a valid message along with verification results. As a result, a proposal in the VRB is not only checked for sender authentication but is also examined for the correctness of the proposal's content. Since a proposal in the set BFT is a set of transactions, the attached verification result *verif* for a proposal is set as a list where elements indicate the indices of invalid transactions.

2) Binary BFT Consensus: As mentioned above, all the proposals delivered via verified reliable broadcast are free from undetected invalid transactions. Before joining reconciliation to filter all the across-proposal contradictory transactions, all verified proposals are still required to go through parallel binary BFT instances and only proposals with positive consensus outcomes are allowed to be merged into the final block. Reaching a consensus is crucial to ensure that all correct processes have the same set of proposals for reconciliation. Otherwise, failure to reach a consensus can result in different or even conflicting outcomes for correct processes.

RedBelly leverages the DBFT binary consensus protocol [18] because it is optimal in resilience and time and has been verified using model checking techniques [52]. Unlike most of the BFT consensus algorithms, DBFT has a leaderless design and only requires a weak coordinator to enable rapid consensus among correct processes.

3) Reconciliation Protocol: The reconciliation protocol in RedBelly examines all valid transactions in parallel binary BFT instances and establishes a total order for those transactions. This protocol serves a dual purpose: it resolves conflicts between parallel BFT instances and establishes a total order for blockchain operations. However, if transactions from the same source are confined to the same binary BFT instance, the latter task may be unnecessary [53].

III. Preliminaries

A. System Model

1) Process Model: We consider a network $\Pi = \{p_1, p_2, \ldots, p_N\}$ of N processes with identifiers known to all in the network. The non-ideal nature of a process's operations in real practice makes it not practical to assume that all processes operate correctly all the time. Up to f processes in a network of size N can be faulty. Faulty processes might stay silent, or send arbitrary or conflicting messages. We assume that processes are authenticated, i.e., they can rely on a digital signature scheme to authenticate each other's messages. Processes are equipped with trusted components (cf. Section IV). The fault threshold f is known to all processes. Our proposed hybrid BRB has a threshold of $f < N/2$. A process is regarded as faulty if it crashes indefinitely or deliberately deviates from the protocol. In this paper, the words *arbitrary*, *faulty* and *Byzantine* are synonyms with the same meaning.

2) Communication Link: Processes are connected in a distributed manner where each process is able to communicate with any other process via a direct connection. The communication link between each pair of processes is a bidirectional perfect point-to-point link, which enforces that no message from an honest sender is lost, duplicated or indefinitely delayed [54]. The network is partially synchronous, whereby messages can be eventually delivered despite the delays in communication links being constrained by an indeterminate upper bound.

3) Signature: In our work, we assume that the cryptographic primitives, including the hash function and signature scheme, are secure and reliable thus message validity is contingent upon proper signature authentication. Each process is equipped with a unique public/private key pair that is integrated into its trusted hardware and consequently, restricting message signing and verification within the trusted hardware. We assume a signature contains the identity of the signing process, and as such, a process' trusted hardware must verify a received message's signature before any further handling.

B. Problem Formulations

We aim to design a trusted hardware-assisted leaderless BFT consensus protocol. Our approach is to provide a trusted solution to the set BFT problem by designing a hybrid broadcast protocol and a hybrid binary consensus protocol. A protocol is a set BFT protocol if the following properties hold:

- *Termination:* Every correct node eventually decides a set of proposals.
- *Agreement:* No two correct processes decide different sets of proposals.
- *Validity:* A decided set of transactions is a valid non-conflicting subset of the union of the proposed sets.
- *Nontriviality:* If all nodes are correct and propose an identical valid non-conflicting set of transactions, then this set is the decided set.

The properties for the broadcast and consensus protocols, along with their respective proofs, are presented in Section VII.

IV. Trusted Services

A simple trusted configuration is required for each process to implement the hybrid models, where all components are subject to Byzantine failures except the ones in the TEE. A trusted service consists of preconfigured functions that remain secure in the presence of Byzantine faults. Previous work has proven the simplest trusted design with merely a counter is insufficient to enhance resilience or performance for streamlined BFT protocols [21]. Our approach relies on two main trusted services: a monotonic *counter* that assigns unique immutable labels to messages, and a *collector* that verifies the validity of the collected messages.

The two trusted services are responsible for executing critical functions and producing verifiable outcomes encapsulated in messages. These messages can be signed and authenticated using private/public keys stored within the TEE. Each process keeps its own private key secret while the public key is publicly accessible and linked to the process' unique identifier. A diagram outlining

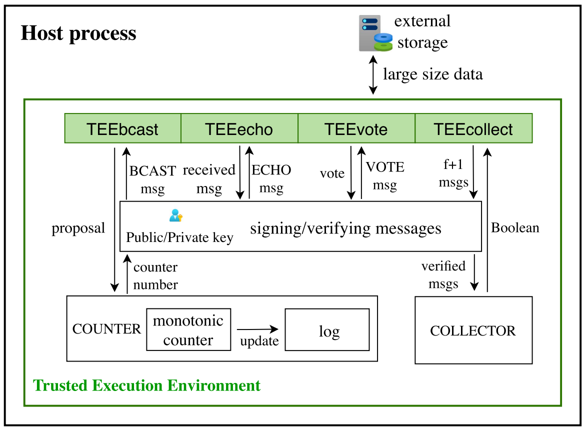

Fig. 1. Operational structure of TEE in this work.

the comprehensive operational structure of TEE in this work is provided in Fig. 1.

A. Counter

The trusted counter is used to assign unique identifiers for proposals that are broadcast by each process. To prevent any equivocation, a monotonic counter is employed to certify proposals in signed messages. As a result, any conflicting proposals generated from equivocation are rejected by honest processes since they lack valid identifiers.

In each execution, a process interacts with its counter to allocate a unique counter number to the proposal it aims to broadcast. The monotonic counter increments every time a broadcast message is generated, which ensures that all allocated numbers are distinct and unchangeable. The highest counter numbers that have been received from all processes are recorded in a trusted secure log, which guarantees that messages from the same broadcaster with counter numbers smaller than the recorded one will not be handled.

It is possible for messages to be delivered in a different order than they were sent. Therefore, if a process receives a new message from the same broadcaster with a higher counter number than the one it is currently working on, it will prioritize the message with the highest counter number and abandon the old one. Additionally, each process's secure log maintains the current status (ongoing/finalized) of the latest proposal it broadcasts. The process can only broadcast a new proposal after the previous one has been finalized in the consensus instance.

A random process $process_i$ records all processes' highest counter numbers as well as the status of its own proposal. As long as $process_i$'s proposal status remains *ongoing*, it cannot broadcast a proposal as the trusted counter is unable to assign a new counter number for the new proposal unless the status of the current proposal is updated to *finalised*.

The following interfaces are provided for a process to access the counter in its TEE:

- TEEbcast(*proposal*) allows a process to generate a message to broadcast a *proposal*. It takes the *proposal* the process intends to broadcast and returns a BCAST message with a counter number included.
- TEEecho(*message*) allows a process to rebroadcast a message after receiving a proposal. It takes the received BCAST *message* and returns a ECHO message.

Apart from interfaces, internal functions are also provided to facilitate the interfaces. Following functions can only be called inside TEE:

- tsign(*message*) signs a *message* in TEE. It takes the *message* and returns the signature.
- tcheck(*message*) checks whether the *message* has a valid counter number attached. If the counter number in *message* has not been received before, a Boolean value TRUE is returned, otherwise FALSE is returned.
- tupdate(*proposal*) updates the record of a *proposal* in the secure log. It will be called once the proposal has been received and has a higher counter number than its predecessors. Additionally, the broadcaster will call this function when its proposal has been finalized in the consensus instance.

B. Collector

The trusted collector serves the purpose of counting the input messages from the external storage. Typically, a process retrieves messages from external storage and carries out the verification process within the TEE before accumulating verified messages in the collector. The collector returns a positive Boolean value if a minimum threshold amount ($f+1$ in this work) of messages have been collected. In particular, the collector can ensure the required minimum amount of messages have to be received for further handling and malicious processes are unable to proceed unless the received messages have been verified by the trusted collector. The collector is to ensure that faulty processes are unable to falsify the type or quantity of messages that have been received. To ensure that malicious behaviours can be easily detected, messages' validity has been checked against predetermined standards and the collector only collects the legitimate ones.

The collector necessitates an additional interface in conjunction with the aforementioned interfaces and functions. This new interface is defined as follows:

- TEEcollect(*messages*, *value*) is designed to collect *messages* from a quorum of $f+1$ distinct processes, all of which should contain the same *value*. If the input *messages* has been successfully verified and collected, TRUE is returned. It is worth mentioning the collector only checks and collects *messages* without storing them inside TEE.

V. Hybrid Reliable Broadcast Protocols

In this section, we introduce the first main primitive in our approach, the hybrid verified reliable broadcast. Prior to entering the consensus phase for binary decisions, proposals must be disseminated as votes in the consensus phase depend on their delivery status. It's crucial to ensure that the broadcast protocol aligns with the subsequent consensus algorithm in terms of resilience and message complexity; otherwise, the broadcast part could become the system's bottleneck. Correia et al. present a transformation methodology that leverages trusted hardware to enable crash consensus algorithms to tolerate Byzantine faults. The authors outlined a BRB that achieves a reduction from a Byzantine reliable broadcast with $3f+1$ processes to a regular reliable broadcast with $2f+1$ processes with a generalized trusted service termed *wormhole* [29]. The authors demonstrate that Byzantine reliable broadcast can adopt the simple message exchange pattern inherent to regular reliable broadcast, effectively reducing one round of all-to-all communication. Since VRB augments BRB by integrating verification, we adopt Correia's design paradigm for BRB, implementing a hybrid BRB and further extending it to a hybrid VRB protocol, without introducing extra message complexity or compromising resilience, and avoid overburdening the TEE.

A. Hybrid Byzantine Reliable Broadcast Protocol

The broadcaster first sends its proposal in a BCAST message generated in TEE to all other processes. A process delivers a proposal upon receiving the BCAST message and after checking its correctness inside TEE. Before the proposal is delivered, the process generates an ECHO message with the proposal included and sends it to all processes. Other processes that have not received the BCAST message can therefore deliver it when they receive an ECHO message.

We implement our algorithm in an event-triggered approach. The operations of a process can be triggered by a total of four triggering events:

- Broadcast: this event is triggered when the process (broadcaster) intends to broadcast a *proposal*.
- Receive: this event is triggered when the process receives a message (it is either a BCAST message or ECHO message).
- Send: this event is triggered when the process intends to send a message to another process via the link *pl*. The broadcast operation is completed by triggering the Send event to all the processes.
- Deliver: this event is triggered when the process successfully delivers a *proposal*.

When a process (broadcaster) broadcasts a proposal, the Broadcast event is triggered. Upon this event, the process will input its proposal into TEE via TEEbcast to generate a BCAST message and send it to all processes in the network by triggering the Send event. When generating a BCAST message, the proposal is wrapped with an incremented counter number along with its identifier and signature.

After a process receives a message, the Receive event is triggered and the process will attempt to echo (rebroadcast) the received message in a new ECHO message via TEEecho. When TEEecho is invoked, the received message undergoes verification before the process's secure log is updated with the received counter number and broadcaster ID. Upon successful verification and updates, TEEecho returns an ECHO message. If the signature is valid and the counter value has not been received before, an ECHO message is successfully generated, and the process will send it to all processes and deliver the proposal by triggering the Deliver event. The message to trigger the Receive event can be a BCAST message or an ECHO message, whichever is received first.

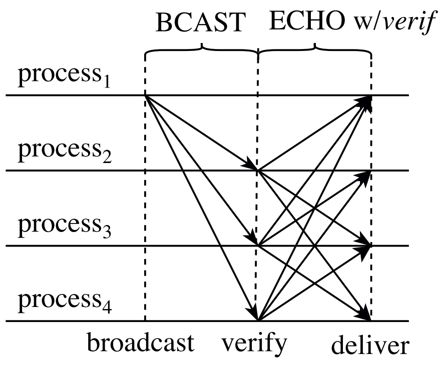

Fig. 2. Hybrid verified reliable broadcast workflow.

B. Hybrid Verified Reliable Broadcast Protocol

Compared with the hybrid Byzantine reliable broadcast protocol, the hybrid VRB requires an extra proposal-verifying function that is integrated with the broadcast protocol. In the verified BRB, the verification results are generated by a selection of processes, called *verifiers*, and the results *verif* are attached to READY messages propagated in the second round of Bracha's double-echo protocol.

To keep the trusted setup lightweight, we use a function *verify* that is executed outside of the trusted hardware to perform the verification for messages while the authentication verification still remains inside TEE. The message exchange pattern remains identical to the hybrid Byzantine reliable broadcast. Although the message exchange pattern remains the same with the hybrid BRB, a quorum is necessary for proposal delivery in the hybrid VRB. Therefore, a TEE interface TEEecho is used to collect $f+1$ valid ECHO messages for a proposal to be delivered.

In our hybrid BRB protocol, as depicted in Fig. 2, a proposal is delivered upon receiving a BCAST or ECHO message, and an ECHO message is generated and broadcast when the proposal is delivered. In hybrid VRB, ECHO messages have an extra field *verif*, which is a list that contains verification results. Processes verify a received message before sending their ECHO message. Since the verification is put outside of the TEE, the correctness of *verif* cannot be unconditionally trustworthy like the outputs from TEE's interfaces or functions. It is possible for a Byzantine process to send a correctly generated message with an incorrect *verif*. To tackle this, $f+1$ messages with the same *verif* are required for a proposal to be delivered.

A process keeps a record of all the verified ECHO messages with the same *verif* (including its own) and waits for a quorum to be collected. Note that those messages are kept in the external storage outside the process's TEE. The size of the required quorum is $f+1$ in our settings to guarantee the correctness of a proposal's *verif*.

Details of the implementations are stated in Algorithm 1. The broadcaster sends a proposal in a BCAST message by following the same procedures required for the hybrid BRB. After receiving a message, the recipient process generates an ECHO message after checking the received BCAST message's signature and counter number. When generating an ECHO message in TEE, the process needs to verify the message's proposal and attach the verification results *verif* to the message content. Each

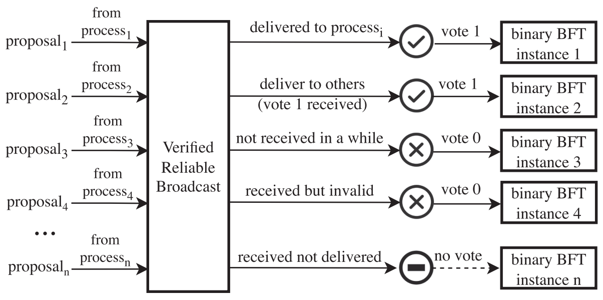

Fig. 3. Hybrid Binary BFT workflow for $process_i$.

process needs to store all the legal ECHO messages for any future feasible quorum to deliver a proposal. The storage for all the verified ECHO messages is outside TEE's secure log due to their relatively large sizes. Although ECHO messages are stored outside the trusted hardware, TEE provides an interface TEEcollect for a process to form a quorum from adequate ECHO messages. Only after a quorum is successfully collected via TEEcollect, a process can finally deliver this proposal.

VI. Hybrid Binary BFT Consensus

Since proposals delivered via VRB are all considered valid and the potential conflicts between proposals (if any) can be sorted out in the final merging after the consensus, we consider that all delivered proposals from the verified reliable broadcast can be accepted. The corresponding binary BFT instances should produce a positive decision for a delivered proposal via VRB. The challenge is to ensure all honest processes always accept the same set of proposals. Thus, the goal of the binary consensus is to provide a simple and reliable method for all correct processes to end up with the same set of proposals.

In RedBelly, the binary value broadcast protocol has been piggybacked with the verified reliable broadcast for communication optimization. Similarly, the proposed hybrid BFT algorithm is integrated with the hybrid VRB we introduced previously. All processes are supposed to send a proposal to the network via the hybrid VRB and a binary BFT instance is set for each proposal to decide whether it shall be included for further handling. The potential conflicts between proposals (if any) can be sorted out in the reconciliation after the consensus as long as honest processes have the same set of decided proposals. A diagram can be found in Fig. 3. For each binary BFT instance, processes must vote via TEEvote. We consider *N* parallel binary BFT instances representing *N* total processes that are all required to send a proposal. All processes should also participate in each instance to reach a consensus for the corresponding proposal.

- TEEvote(*bID*, *bv*) allows a process to generate a VOTE message, *bID* is the identifier for the broadcaster as well as its proposal's instance. The *bv* is the binary vote the process intends to cast.

Guaranteed by the hybrid VRB, a delivered proposal is considered valid, and a corresponding VOTE message will be generated and broadcast to all processes. If the proposal has not been delivered to p_i yet but has already been received (waiting to

Algorithm 1: Hybrid Verified Reliable Broadcast.

```
1:  Implements:
2:      HybridVerifiedByzantineReliableBroadcast, instance $hvrb$.
3:  Properties:
4:      Validity: If a correct process p broadcasts a message m, then every correct process eventually delivers m.
5:      No duplication: No correct process delivers message m more than once.
6:      Integrity: If a correct process delivers a message m from the correct sender p, then m is previously broadcast by p.
7:      Agreement: If some correct process delivers a message m, then every correct process eventually delivers m.
8:  Uses:
9:      PerfectPointToPointLinks, instance $pl$.
10: Variables:
11:     prop: the proposal a process intends to broadcast
12:     bID: broadcaster's ID, the process broadcasting the proposal
13:     cNo: counter number attached to a proposal
14:     sID: sender's ID, the process sending this message
15:     sig: signature generated by the sender
16:     echos: collection of ECHO messages
17: upon event $\langle hvrb, \texttt{Broadcast} \mid prop\rangle$ where prop $\neq \varnothing$ do
18:     msg ← TEEbcast(prop)
19:     forall $q \in \Pi$ do
20:         trigger $\langle pl, \texttt{Send} \mid q, msg\rangle$
21: upon event $\langle pl, \texttt{Receive} \mid msg\rangle$ where msg contains (BCAST, prop, bID, cNo, sig) do
22:     //no duplicate messages can be generated from TEEecho
23:     echo ← TEEecho(msg)
24:     forall $q \in \Pi$ do
25:         trigger $\langle pl, \texttt{Send} \mid q, echo\rangle$
26:     echos ← echos ∪ echo
27:     if size(echos) ← f+1 then
28:         trigger $\langle hvrb, \texttt{Deliver} \mid prop\rangle$
29: upon event $\langle pl, \texttt{Receive} \mid msg\rangle$ where msg contains (ECHO, prop, bID, cNo, verif, sig) do
30:     if verif is valid then
31:         echos ← echos ∪ echo
32:     if TEEcollect(echos, verif) then
33:         trigger $\langle hvrb, \texttt{Deliver} \mid prop\rangle$
34:     else
35:         //in case this message hasn't been echoed before
36:         echo ← TEEecho(msg)
37:         forall $q \in \Pi$ do
38:             trigger $\langle pl, \texttt{Send} \mid q, echo\rangle$
39:         echos ← echos ∪ echo
40: //broadcaster generates BCAST message
41: interface TEEbcast(prop) where prop $\neq \varnothing$:
```

Algorithm 1: (Continued).

```
42:     cNo++
43:     sig ← tsign([BCAST, prop, bID, cNo])
44:     return [BCAST, prop, bID, cNo, sig]
45: interface TEEecho($msg$) where msg contains (prop,
    bID, cNo, sig):
46:     if tcheck($msg$) ← FALSE then
47:         break
48:         //this function is outside TEE, included for
            simplicity
49:         verif ← verify(msg.prop)
50:     if verif ← ∅ then
51:         break
52:     sig ← tsign([ECHO, prop, bID, cNo, verif])
53:     //store this echoed broadcaster's value
54:     tupdate(prop)
55:     return [ECHO, prop, bID, cNo, verif,sig]
56: interface TEEcollect(echos, verif) where echos are
    ECHO messages:
57:     if ∃ f+1 different echo in echos: echo.verif ← verif
        then
58:         return TRUE
59:     else
60:         return FALSE
```

collect f+1 VOTE messages in VRB before being delivered), p_i will not generate any VOTE message until the proposal is delivered or a VOTE message **1** from other processes is received. The very first VOTE message **1** can only be generated after a proposal has been delivered. As a result, an honest process will directly follow along to vote **1** upon receiving a valid VOTE message **1** from the others. In the binary BFT instance for a *proposal*, process p_i generates a VOTE message and broadcasts to the network if one of the following events is triggered:

- Vote **1** if the *proposal* has been received and delivered to the process p_i.
- Vote **1** if a valid VOTE message **1** has been received from another process, meaning the *proposal* has been delivered to others.
- Vote **0** if the *proposal* is verified to be invalid.
- Vote **0** if the *proposal* has not been received after *N-f* instances output 1 and an oldest-transaction-based timer has expired.

The conditions for process p_i to vote **0** are relatively limited. In general, p_i votes **0** for a proposal only if the proposal is invalid or extremely slow. p_i votes **0** if the proposal is received and verified to be invalid (no valid transaction contained). Due to the partial synchrony setting, the time bound to receive all correct processes' messages is unknown. It is essential to establish a mechanism to periodically produce a set of proposals for further handling, rather than relying on the delivery of a proposal for unpredictable time. Therefore, a process p_i will generate a VOTE message **0** for a proposal only if the three conditions are met: 1) the proposal has not been received via the VRB, 2) *N-f* other proposals have been decided, and 3) the timer has expired.

It is important to keep in mind that receiving a proposal is different from delivering it. A proposal is delivered to process p_i when p_i has successfully collected $f+1$ valid ECHO messages in hybrid VRB and the corresponding Deliver event is triggered. On the other hand, a proposal is received when p_i has received the proposal for the first time, usually from a BCAST message or a ECHO message, and starts to collect $f+1$ valid ECHO messages for this proposal to be delivered.

There might be some undecided instances whose proposals have already been received but not delivered, which means they are still waiting for a $f+1$ quorum in VRB before being delivered. The proposals in those instances can be considered to be valid as they have been verified by TEE when received. However, they cannot be directly delivered due to the quorum requirements in VRB, which guarantees the proposal has been verified by at least $f+1$ processes before joining the consensus instance. The process will not cast a vote for a proposal like this until it is delivered or a valid VOTE **1** message has been received, meaning the proposal has been successfully delivered to others. A process p_i decides in a binary BFT instance by triggering the Decide event if $f+1$ VOTE message with the same binary vote value are received and collected via TEEcollect.

Details of the implementations are stated in Algorithm 2. The hybrid binary BFT is set as an event-triggered scheme where a process' operations are triggered upon several events. There is a total of three possible triggering events in the model: Deliver, receive and *N-f* instances decide 1 **and** *timer* expires.

- Propose: propose a binary vote in a VOTE message.
- Receive: receives a VOTE message.
- Deliver: a *proposal* is delivered via VRB.
- Decide: decides the consensus outcome based on the VOTE message.
- *N-f* instances decide 1 **and** *timer* expires: *N-f* parallel instances have decided and a timer has also expired.

VII. Evaluation

A. Security Analysis

1) Hybrid Byzantine Reliable Broadcast: A secure Byzantine reliable broadcast protocol ensures that the correct processes always agree on the proposal they receive with the presence of Byzantine processes. The resilience against Byzantine processes is significantly improved in the proposed BRB protocol due to the TEE's trusted capabilities. We provide proof showing the proposed hybrid BRB has satisfied the required properties [54].

Lemma 1 (BRB-Validity): If a correct process *p* broadcasts a message *m*, then every correct process eventually delivers *m*.

Proof: We prove this by contradiction. Assume there is a correct process *q* that fails to deliver a message from a correct process *p*, *q* must fail to receive *m* either as a BCAST message from *p* or a ECHO message from all others. There are three cases to consider:

- *Case 1: m* has been sent to *q* but is not delivered.
- *Case 2: p* has never sent a BCAST message *m* to *q*.

Algorithm 2: Hybrid Binary BFT.

```
Implements:
    HybrifBinaryByzantineFaultTolerance, instance $hbbft$.
Uses:
    HybridVerifiedByzantineReliableBroadcast, instance hvrb.

Variables:
    bv: binary vote
    votes: collection of VOTE messages
upon event $\langle hvrb, \texttt{Deliver} \mid bID, prop\rangle$ do
    vote ← TEEvote(bID, prop, 1)
    trigger $\langle hbbft, \texttt{Propose} \mid vote\rangle$
upon event $\langle hbbft, \texttt{Propose} \mid vote\rangle$ do
    forall $q \in \Pi$ do
        trigger $\langle pl, \texttt{Send} \mid q, vote\rangle$
    votes ← votes ∪ vote
upon event $\langle pl, \texttt{Receive} \mid vote\rangle$ where vote is [VOTE, bID, prop, bv, sig]do
    //include the received VOTE message
    votes ← votes ∪ vote
    if vote.bv ← 1 and no vote proposed yet then
        vote ← TEEvote(1, prop, vote.bv)
        trigger $\langle hbbft, \texttt{Propose} \mid vote\rangle$
        //include its own VOTE message
        votes ← votes ∪ vote
    if TEEcollect(votes, prop) ← TRUE
        trigger $\langle hbbft, \texttt{Decide} \mid vote\rangle$
        //only if the process is the broadcaster
        tupdate(prop)
upon event ⟨N-f instances decide 1⟩
    vote ← TEEvote(bID, prop, 0)
    trigger $\langle hbbft, \texttt{Propose} \mid vote\rangle$
//generate a VOTE message
interface TEEvote(bID, prop, bv):
    sig ← tsign([VOTE, bID, prop, bv])
    return [VOTE, bID, prop, bv, sig]
interface TEEcollect(votes, prop) where votes are VOTE messages:
    if ∃ f+1 different vote in votes: vote.prop ← prop then
        return TRUE
    else
        return FALSE
```

- *Case 3:* Other processes have never sent an ECHO message *m* to *q*.

Case 1: this case contradicts the properties of the underlying links *pl*.

Case 2: this case contradicts the assumption that process *p* is correct, which guarantees *p* sends a BCAST message *m* to all other processes.

Case 3: there are two subcases to consider: (3-1) no process has sent an ECHO *m* to *q* after receiving a BCAST message *m* from *p*; (3-2) no process has received a BCAST message *m* from *p*.

In subcase 3-1, this subcase contradicts the assumption that at least half of the total processes are correct.

In subcase 3-2, this subcase contradicts the assumptions given in Case 1 and Case 2.

The proof of the lemma is concluded. □

Lemma 2 (BRB-No duplication): No correct process delivers message *m* more than once.

Proof: This is inherently met by the trusted components as they ensure each message to be attached with a unique *counter number*, which is generated by a monotonic counter inside TEE. Processes will keep a record of all received messages' counter numbers. Any redundant message with the same *counter number* will be ignored. The proof of the lemma is concluded. □

Lemma 3 (BRB-Integrity): If a correct process delivers a message *m* from the correct sender *p*, then *m* is previously broadcast by *p*.

Proof: We prove this by contradiction. Assume a correct process *q* delivers a message *m* from the correct sender *p* while *p* never broadcast *m*. There are two cases to consider:

- *Case 1: m* is delivered after the Deliver event is triggered upon *q* receiving a BCAST message.
- *Case 2: m* is delivered after the Deliver event is triggered upon *q* receiving a ECHO message.

Case 1: q receives a BCAST message with the sender ID set to *p*. BCAST message must be previously broadcast by *p* because only *p* can generate a BCAST message in its TEE with the sender ID set to *p*. It contradicts our assumption that *p* has never broadcast *m*.

Case 2: q receives a ECHO message with the sender ID set to *p*. There must be a BCAST message with the same *broadcasterID* broadcast previously. The very first ECHO message with *broadcasterID* set to *p* can only be generated in some process' TEE after receiving a BCAST message with the same *broadcasterID* set to *p*. It contradicts our assumption as demonstrated in Case 1.

The proof of the lemma is concluded. □

Lemma 4 (BRB-Agreement): If some correct process delivers a message *m*, then every correct process eventually delivers *m*.

Proof: We prove this by contradiction. Assume correct process *p* fails to deliver a message *m* from sender *s* and some other correct process *q* does. There are two cases to consider:

- *Case 1: s* is correct.
- *Case 2: s* is faulty.

Case 1: From **BRB-Integrity**, we deduce that *m* is broadcast by s. From **BRB-Validity**, we assert all process, including *p*, eventually delivers *m*, a contradiction.

Case 2: s only broadcasts *m* to part of the network and has never sent *m* to *p*. Since *q* is correct and delivers *m*, *q* rebroadcasts *m* in an ECHO message before it delivers as required by the protocol. From Case 1 of the proof of **BRB-Validity**, the *pl* link guarantees *q*'s ECHO message to be eventually delivered to *p*. In our assumption, *p* fails to deliver a message *m* after receiving *q*'s ECHO message, which contradicts with our protocol requirements and *p*, being correct, is expected to adhere to the protocol. A contradiction.

The proof of the lemma is concluded. □

2) Hybrid Verified Reliable Broadcast: The hybrid VRB protocol can be viewed as a hybrid BRB with a verification function. Here, the verification function is an external component situated outside the TEE. Given that the trusted components cannot ensure the correctness of the verification's output *verif*, a quorum of $f+1$ is required in a network of $2f+1$.

For a message *m* to be delivered to process *p*, *p* must receive at least $f+1$ ECHO messages with the same *verif*. To validate the correctness of the VRB protocol, we provide proofs for **VRB-Validity** and **VRB-Agreement** while the proofs for **VRB-No duplication** and **VRB-Integrity** align with the aforementioned BRB.

Lemma 5 (VRB-Validity): If a correct process *p* broadcasts a message *m*, then every correct process eventually delivers *m*.

Proof: We prove this by contradiction. Assume there is a correct process *q* that fails to deliver a message from a correct process *p*. *q* is unable to collect $f+1$ ECHO messages with the same *verif*. There are three cases to consider:

- *Case 1: p* has sent a BCAST message *m* to *q* but neither a BCAST nor an ECHO message is delivered to *q*.
- *Case 2: p* has never sent a BCAST message *m* to *q*.
- *Case 3:* Less than $f+1$ processes have sent an ECHO message *m* with *verif* to *q*.

Case 1: the proof for this case aligns with **BRB-Validity** Case 1 and Case 3.

Case 2: this case contradicts the assumption that process *p* is correct.

Case 3: since there are at least $f+1$ correct processes, there exists at least one correct process, say *x*, that never sends a valid ECHO message to *q*. Drawing from Case 1 and Case 2, we deduce that process *x* can eventually receive *m*. In this case, process *x* deviates from the protocol by not broadcasting a valid ECHO message after receiving *m*, which contradicts the assumption that process *x* is correct.

The proof of the lemma is concluded. □

Lemma 6 (VRB-Agreement): If some correct process delivers a message *m*, then every correct process eventually delivers *m*.

Proof: We prove this by contradiction. Assume correct process *p* fails to deliver a message *m* from sender *s* and some other correct process *q* does. There are two cases to consider:

- *Case 1: s* is correct.
- *Case 2: s* is faulty.

Case 1: From **VRB-Integrity**, we deduce that *m* is broadcast by *s*. From **VRB-Validity**, we assert all processes, including *p*, eventually deliver *m*, a contradiction.

Case 2: s only broadcasts *m* to part of the network and never sends *m* to *p*. Given that *q* is correct and delivers *m*, *q* rebroadcasts *m* in an ECHO message after receiving *m* as required by the protocol. Upon receiving *q*'s ECHO message, all correct processes will broadcast ECHO messages. Since *p* fails to deliver a *m*, it implies *p* has never received $f+1$ valid ECHO messages and there exists at least one correct process that never sends an ECHO message to *p*, which contradicts the assumptions in this case.

The proof of the lemma is concluded. □

3) Hybrid Binary BFT Consensus: Deployed with the hybrid VRB and a reconciliation algorithm, the proposed binary BFT consensus is to solve the set BFT problem with a hybrid VRB and a reconciliation algorithm. Since hybrid BRB is proven to be secure and the reconciliation algorithm remains unchanged and can barely be affected by the other two parts, we here use the properties of set BFT defined in the RedBelly blockchain to attest to the correctness of binary BFT consensus.

Lemma 7 (SBC-Termination): Every correct node eventually decides a set of proposals.

Proof: We prove this by contradiction. Assume there is a correct process *p* that fails to decide any proposal(with a positive outcome).

There are three cases to consider:

- *Case 1: p* eventually decides 0 for all the proposals.
- *Case 2: p* eventually fails to decide for any proposal.
- *Case 3: p* eventually decides 0 for some proposals while failing to decide for all other proposals.

Case 1: p must have received $f+1$ 0 votes for all the proposals, indicating that all proposals have at least $f+1$ processes voting 0 for them. Furthermore, none of these proposals is delivered to any process otherwise there can only be at most *f* processes voting 0. This contradicts the **VRB-Validity**.

Case 2: p must have neither received vote 1 nor $f+1$ votes 0 for any proposal otherwise *p* would eventually decide 1 or 0. It implies proposals from all correct processes fail to be delivered to *p*, which contradicts the **VRB-Validity**.

Case 3: Similar to Case 2, this case suggests that proposals from all correct processes fail to be delivered to *p*, which contradicts the **VRB-Validity**.

The proof of the lemma is concluded. □

Lemma 8 (SBC-Agreement): No two correct processes decide different sets of proposals.

Proof: We prove this by contradiction. Assume there are correct processes *p* and *q* that end up with different sets of proposals, which means there is at least one proposal, say *prop*, that has different consensus outcomes from *p* and *q*. For a process to output a consensus outcome, it must have received $f+1$ VOTE messages with the same vote value and at least one process has voted twice for *prop*. In this case, the conditions to vote 0 and 1 have all been triggered in this process' TEE, which is impractical in our design. A process votes 0 for *prop* if the *prop* has not been received or verified *prop* to be invalid. It contradicts the vote 1 condition as *prop* needs to be received and verified to be valid in order to be delivered. The proof of the lemma is concluded. □

Lemma 9 (SBC-Validity): A decided set of transactions is a valid non-conflicting subset of the union of the proposed sets.

Proof: The validity of the decided sets can be guaranteed by the verification function as well as the $f+1$ quorum required for proposal delivery in the hybrid VRB. The cross-proposal non-conflicting feature can be achieved via the reconciliation procedure. The proof of the lemma is concluded. □

Lemma 10 (SBC-Nontriviality): If all nodes are correct and propose an identical valid non-conflicting set of transactions, then this set is the decided set.

Proof: This is to prevent a trivial algorithm that always outputs an empty set to solve the problem. Assume correct processes all propose an identical proposal *p* where all transactions are valid, and the final decided set is q, which is a subset of p. During the verification of p using VRB, at least $f+1$ correct processes will verify and accept all transactions, eventually delivering the proposal p in its entirety. The following consensus instance in $f+1$ correct processes will output 1 for p as it has been successfully delivered, which contradicts the assumption. The proof of the lemma is concluded. □

B. Performance Evaluation

1) Message Complexity. Hybrid reliable broadcast protocols: The proposed hybrid BRB requires one round of one-to-all communication for the BCAST message and one round of all-to-all communication for ECHO messages. Therefore, the maximum message complexity is $O(n+n^2)$. In comparison, the message complexity of the original Bracha's authenticated double-echo protocol is $O(n+2n^2)$ with one round of one-to-all communication (SEND) and two rounds of all-to-all communication (ECHO and READY) (cf. Table I).

Binary BFT consensus: As for message complexity, in a BFT instance where 1 is eventually decided, the worst case is when f processes have voted 0, and a valid VOTE 1 message has arrived. In this case, all the processes that have not voted 0 need to send VOTE 1 message and those who sent VOTE 0 messages before will also need to send VOTE 1 message. As a result, the message complexity is $O(n^2+n)$. In a BFT instance where 0 is eventually decided, no VOTE 1 message can ever be generated in TEE, there are at most N VOTE 0 messages, so the message complexity is $O(n)$. In general, the highest possible message complexity is $O(n^2+n)$.

2) Simulation Performance: To simulate the performance of our hybrid set BFT protocol, we deploy C++ implementations of the protocol on the Intel i5-10505 3.20 GHz processor with 16 GB of memory and run Ubuntu 18.04 LTS. Processes use ECDSA signatures with prime256v1 elliptic curves (available in OpenSSL [55]) and are connected using Salticidae [56]. For each experiment, we generate the average result of 5 runs. The latency and throughput of each network link is set to 0.5 ms and 1 Mbps, respectively. We measure the protocol's latency as the time required for all correct processes to deliver the broadcast proposals. We consider payloads of 250 bytes for each transaction, and each process broadcast proposals with a 50-millisecond interval. We evaluate the performance of our hybrid set BFT method against the trusted BFT protocol minBFT. In our comparison, we adjust both the number N of processes and the number f of Byzantine processes, maintaining $N \leq 2f+1$ for the hybrid set BFT.

We first analyze the performance of the protocols in a network that has the maximum number of Byzantine faults, which is f in a $2f+1$ configuration in our scenario. As shown in Fig. 4, the throughput of hybrid set BFT is roughly 89% greater than that of minBFT. When the batch (proposal) size exceeds 400, minBFT exhibits only a marginal increase in throughput. In contrast, a larger batch in set BFT continues to enhance the throughput, reaching around 7 k tps. Notably, minBFT's performance diminishes in larger networks as it incurs a more complex message

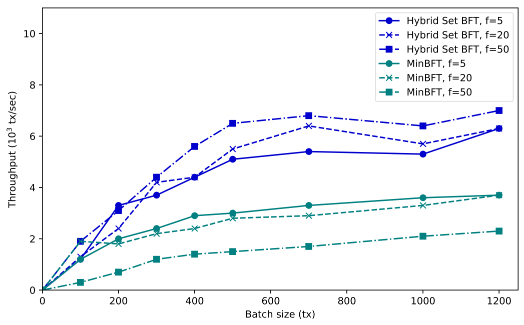

Fig. 4. Throughputs for different batch sizes.

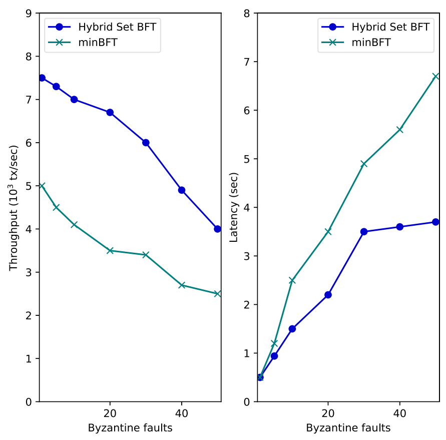

Fig. 5. Throughput and latency comparison in the network of 100 processes with varying Byzantine processes.

exchange pattern while the throughput of hybrid set BFT may increase in larger networks. This is attributed to the leaderless design of set BFT, which can make the most use of the bandwidth, allowing more proposals to be processed concurrently.

When evaluating the two protocols in a network of 100 processes with varying Byzantine faults, hybrid set BFT demonstrates (in Fig. 5) 67% higher throughput than minBFT when Byzantine processes are fewer than 10%, reaching nearly 90% higher throughput when over 30% of processes are Byzantine. In terms of latency, minBFT experiences a surge as Byzantine faults increase, while set BFT shows a more gradual growth. Set BFT maintains a steady latency of around 4 seconds when Byzantine percentage exceeds 30%, whereas minBFT approaches close to 7 seconds.

VIII. Conclusion

We design a leaderless BFT protocol, leveraging trusted hardware to provide a hybrid solution for the set consensus problem. To implement this protocol, we introduce two primitives: a hybrid verified reliable broadcast protocol and a hybrid binary consensus. The hybrid designs in our model provide all the necessary trusted interfaces and internal functions in a simple hardware configuration, which only requires a monotonic counter and a small secure log. Our analysis shows that the proposed protocol can provide optimal resilience with improved message complexity.

References

[1] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” 2008.

[2] C. Natoli, J. Yu, V. Gramoli, and P. Esteves-Verissimo, “Deconstructing blockchains: A comprehensive survey on consensus, membership and structure,” 2019, *arXiv: 1908.08316*.

[3] Y. Xiao, N. Zhang, W. Lou, and Y. T. Hou, “A survey of distributed consensus protocols for blockchain networks,” *IEEE Commun. Surveys Tuts.*, vol. 22, no. 2, pp. 1432–1465, Second Quarter 2020.

[4] M. Vukolić, “The quest for scalable blockchain fabric: Proof-of-work versus BFT replication,” in *Proc. Int. Workshop Open Problems Netw. Secur.*, Springer, 2015, pp. 112–125.

[5] S. Bano et al., “SoK: Consensus in the age of blockchains,” in *Proc. 1st ACM Conf. Adv. Financial Technol.*, 2019, pp. 183–198.

[6] M. Castro et al., “Practical Byzantine fault tolerance,” in *Proc. 3rd Symp. Operating Syst. Des. Implementation*, 1999, pp. 173–186.

[7] E. Buchman, “Tendermint: Byzantine fault tolerance in the age of blockchains,” Ph.D. dissertation, University of Guelph, 2016.

[8] B. Y. Chan and E. Shi, “Streamlet: Textbook streamlined blockchains,” in *Proc. 2nd ACM Conf. Adv. Financial Technol.*, 2020, pp. 1–11.

[9] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, “HotStuff: BFT consensus with linearity and responsiveness,” in *Proc. ACM Symp. Princ. Distrib. Comput.*, 2019, pp. 347–356.

[10] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song, “The honey badger of BFT protocols,” in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2016, pp. 31–42.

[11] M. Baudet et al., State machine replication in the Libra blockchain, 2019.

[12] L. Lamport, “Brief announcement: Leaderless Byzantine paxos,” in *Proc. Int. Symp. Distrib. Comput.*, 2011, pp. 141–142.

[13] B. Arun, S. Peluso, and B. Ravindran, “ezBFT: Decentralizing byzantine fault-tolerant state machine replication,” in *Proc. IEEE 39th Int. Conf. Distrib. Comput. Syst.*, 2019, pp. 565–577.

[14] T. Crain, C. Natoli, and V. Gramoli, “Red belly: A secure, fair and scalable open blockchain,” in *Proc. IEEE Symp. Secur. Privacy*, 2021, pp. 466–483.

[15] T. Rocket, M. Yin, K. Sekniqi, R. van Renesse, and E. G. Sirer, “Scalable and probabilistic leaderless BFT consensus through metastability,” 2019, *arXiv: 1906.08936*.

[16] M. Ben-Or, R. Canetti, and O. Goldreich, “Asynchronous secure computation,” in *Proc. 25th Annu. ACM Symp. Theory Comput.*, 1993, pp. 52–61.

[17] M. Ben-Or, B. Kelmer, and T. Rabin, “Asynchronous secure computations with optimal resilience,” in *Proc. 13th Annu. ACM Symp. Princ. Distrib. Comput.*, 1994, pp. 183–192.

[18] T. Crain, V. Gramoli, M. Larrea, and M. Raynal, “DBFT: Efficient leaderless Byzantine consensus and its application to blockchains,” in *Proc. IEEE 17th Int. Symp. Netw. Comput. Appl.*, 2018, pp. 1–8.

[19] G. Bracha, “Asynchronous Byzantine agreement protocols,” *Inf. Comput.*, vol. 75, no. 2, pp. 130–143, 1987.

[20] R. Palmieri, “Leaderless consensus: The state of the art,” in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops*, 2016, pp. 1307–1310.

[21] J. Decouchant, D. Kozhaya, V. Rahli, and J. Yu, “DAMYSUS: Streamlined BFT consensus leveraging trusted components,” in *Proc. 17th Eur. Conf. Comput. Syst.*, 2022, pp. 1–16.

[22] J. Zhang et al., “Efficient byzantine fault tolerance using trusted execution environment: Preventing equivocation is only the beginning,” 2021, *arXiv:2102.01970*.

[23] G. S. Veronese, M. Correia, A. N. Bessani, L. C. Lung, and P. Verissimo, “Efficient byzantine fault-tolerance,” *IEEE Trans. Comput.*, vol. 62, no. 1, pp. 16–30, Jan. 2013.

[24] M. K. Aguilera, N. Ben-David, R. Guerraoui, V. Marathe, and I. Zablotchi, “The impact of RDMA on agreement,” in *Proc. ACM Symp. Princ. Distrib. Comput.*, 2019, pp. 409–418.

[25] M. K. Aguilera, N. Ben-David, R. Guerraoui, D. Papuc, A. Xygkis, and I. Zablotchi, “Frugal byzantine computing,” 2021, *arXiv:2108.01330*.

[26] M. J. Fischer, N. A. Lynch, and M. Merritt, “Easy impossibility proofs for distributed consensus problems,” *Distrib. Comput.*, vol. 1, pp. 26–39, 1986.

[27] N. Ben-David, B. Y. Chan, and E. Shi, “Revisiting the power of non-equivocation in distributed protocols,” in *Proc. ACM Symp. Princ. Distrib. Comput.*, 2022, pp. 450–459.

[28] A. Clement, F. Junqueira, A. Kate, and R. Rodrigues, “On the (limited) power of non-equivocation,” in *Proc. ACM Symp. Princ. Distrib. Comput.*, 2012, pp. 301–308.

[29] M. Correia, G. S. Veronese, and L. C. Lung, “Asynchronous byzantine consensus with 2F+ 1 processes,” in *Proc. ACM Symp. Appl. Comput.*, 2010, pp. 475–480.

[30] C. Cachin and S. Tessaro, “Asynchronous verifiable information dispersal,” in *Proc. IEEE 24th Symp. Reliable Distrib. Syst.*, 2005, pp. 191–201.

[31] J. Hendricks, G. R. Ganger, and M. K. Reiter, “Verifying distributed erasure-coded data,” in *Proc. 26th Annu. ACM Symp. Princ. Distrib. Comput.*, 2007, pp. 139–146.

[32] N. Alhaddad, S. Das, and S. Duan, “Balanced Byzantine reliable broadcast with near-optimal communication and improved computation,” in *Proc. ACM Symp. Princ. Distrib. Comput.*, 2022, pp. 399–417.

[33] S. Das, Z. Xiang, and L. Ren, “Asynchronous data dissemination and its applications,” in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2021, pp. 2705–2721.

[34] S. Bonomi, J. Decouchant, G. Farina, V. Rahli, and S. Tixeuil, “Practical byzantine reliable broadcast on partially connected networks,” in *Proc. IEEE 41st Int. Conf. Distrib. Comput. Syst.*, 2021, pp. 506–516.

[35] S. Toueg, “Randomized byzantine agreements,” in *Proc. 3rd Annu. ACM Symp. Princ. Distrib. Comput.*, 1984, pp. 163–178.

[36] A. Patra, “Error-free multi-valued broadcast and byzantine agreement with optimal communication complexity,” in *Proc. Int. Conf. Princ. Distrib. Syst.*, Springer, 2011, pp. 34–49.

[37] M. Correia, N. F. Neves, and P. Verissimo, “How to tolerate half less one byzantine nodes in practical distributed systems,” in *Proc. IEEE 23rd Int. Symp. Reliable Distrib. Syst.*, 2004, pp. 174–183.

[38] B.-G. Chun, P. Maniatis, S. Shenker, and J. Kubiatowicz, “Attested append-only memory: Making adversaries stick to their word,” *ACM SIGOPS*, vol. 41, no. 6, pp. 189–204, 2007.

[39] D. Levin, J. R. Douceur, J. R. Lorch, and T. Moscibroda, “Trinc: Small trusted hardware for large distributed systems,” in *Proc. Symp. Netw. Syst. Des. Implementation*, 2009, pp. 1–14.

[40] J. Behl, T. Distler, and R. Kapitza, “Hybrids on steroids: SGX-based high performance BFT,” in *Proc. 12th Eur. Conf. Comput. Syst.*, 2017, pp. 222–237.

[41] R. Kapitza et al., “CheapBFT: Resource-efficient byzantine fault tolerance,” in *Proc. 7th ACM Eur. Conf. Comput. Syst.*, 2012, pp. 295–308.

[42] J. Liu, W. Li, G. O. Karame, and N. Asokan, “Scalable Byzantine consensus via hardware-assisted secret sharing,” *IEEE Trans. Comput.*, vol. 68, no. 1, pp. 139–151, Jan. 2019.

[43] M. K. Aguilera, N. Ben-David, R. Guerraoui, A. Murat, A. Xygkis, and I. Zablotchi, “UBFT: Microsecond-scale BFT using disaggregated memory,” in *Proc. 28th ACM Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2023, pp. 862–877.

[44] I. Abraham, G. Gueta, D. Malkhi, L. Alvisi, R. Kotla, and J.-P. Martin, “Revisiting fast practical Byzantine fault tolerance,” 2017, *arXiv: 1712.01367*.

[45] K. Antoniadis, A. Desjardins, V. Gramoli, R. Guerraoui, and I. Zablotchi, “Leaderless consensus,” in *Proc. IEEE 41st Int. Conf. Distrib. Comput. Syst.*, 2021, pp. 392–402.

[46] G. Danezis, L. Kokoris-Kogias, A. Sonnino, and A. Spiegelman, “Narwhal and tusk: A DAG-based mempool and efficient BFT consensus,” in *Proc. 17th Eur. Conf. Comput. Syst.*, 2022, pp. 34–50.

[47] A. Spiegelman, N. Giridharan, A. Sonnino, and L. Kokoris-Kogias, “Bullshark: DAG BFT protocols made practical,” in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2022, pp. 2705–2718.

[48] I. Keidar, E. Kokoris-Kogias, O. Naor, and A. Spiegelman, “All you need is DAG,” in *Proc. ACM Symp. Princ. Distrib. Comput.*, 2021, pp. 165–175.

[49] Q. Wang, J. Yu, S. Chen, and Y. Xiang, “Sok: DAG-based blockchain systems,” *ACM Comput. Surv.*, vol. 55, pp. 1–38, 2022.

[50] J. Niu and C. Feng, “Leaderless byzantine fault tolerant consensus,” 2020, *arXiv: 2012.01636*.

[51] N. Shrestha and M. Kumar, “Revisiting ezBFT: A decentralized byzantine fault tolerant protocol with speculation,” 2019, *arXiv: 1909.03990*.

[52] N. Bertrand, V. Gramoli, I. Konnov, M. Lazic, and P. Tholoniat, Compositional verification of byzantine consensus, 2021.

[53] R. Guerraoui, P. Kuznetsov, M. Monti, M. Pavlovič, and D.-A. Seredinschi, “The consensus number of a cryptocurrency,” in *Proc. ACM Symp. Princ. Distrib. Comput.*, 2019, pp. 307–316.

[54] C. Cachin, R. Guerraoui, and L. Rodrigues, *Introduction to Reliable and Secure Distributed Programming*. Berlin, Germany: Springer, 2011.

[55] Retrieved in 2022. [Online]. Available: http://openssl.org

[56] Retrieved in 2022. [Online]. Available: https://github.com/Determinant/salticidae