This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2021.3104465, IEEE Transactions on Dependable and Secure Computing

Social Characteristic-based Propagation-efficient PBFT Protocol to Broadcast in Unstructured Overlay Networks

Xiaoqin Feng, Jianfeng Ma, *Member, IEEE,* Yinbin Miao, Ximeng Liu, *Senior Member, IEEE* and Kim-Kwang Raymond Choo, *Senior Member, IEEE*

Abstract—Blockchain allows for secure management of a shared ledger by agreement protocols, where transactions are validated over network without central authorities. Although the agreement protocol has been thoroughly conducted of propagation and consensus researches, mobility of nodes in unstructured overlay networks has not received much attention. Besides, current dynamic propagation schemes waste travel hops and are low of delivery ratio. In this paper, we propose a social characteristic-based propagation-efficient protocol NefSBFT to agree on system state plus consensus mechanisms in public blockchains. We devise a propagation technique (travel hops of at least $\frac{1}{3}$ savings, delivery ratio above 0.93, etc.) for message multicasting when exploiting real nodes' social characteristics of intermittent connectivity and frequent partitions. This propagation technique is executed in the improved FastBFT to achieve transaction ordering and block verification, thus, no controllable mobility is required during the whole system's execution. NefSBFT achieves fast propagation, small message complexity and few resource consumption of travel hops and running nodes for complete protocol execution. We analyze NefSBFT's security against DDOS attack of non-primary failure. The experiments show the performance tradeoff under different parameters, compare the propagation efficiency with Erlay and Flooding, and clarify NefSBFT's impact on the whole system performance through comparison.

Index Terms—Block verification, efficiency, message propagation, node mobility, transaction ordering.

1 Introduction

The blockchain system is primarily composed of an agreement protocol [1], [2] for broadcasting exchanges [3] between participants of Peer-to-Peer (P2P) network. Although the consensus mechanisms are the core of state synchronization among the network members, efficient and reliable message propagation is intrinsically connected with the security properties of such consensus mechanisms (e.g., Delegated Proof-of-Stake [4], Proof-of-Work [5], [6], Proof-of-Stake and Directed Acyclic Graph [7]). However, some propagations in blockchain agreement protocols consider the controllable mobility of participating nodes. In particular, P2P network nodes in the public blockchain (e.g., Thunder Core, BOSCore (Business Operating System), Ethereum, blockchain-based internet of things [8], and Blockchain Vehicle Information System [9]) go through the mobility [10] a lot. Besides, the extant propagation techniques require numerous travel hops to diffuse the messages or are low of delivery ratio which means the success rate in message transmission.

Researchers have designed various consensus mechanisms [11], [12], [13], [14], [15], [16], [17] which use different propagation techniques. For example, the Delegated Proof-of-Stake (DPoS) [12], Proof-of-Stake (PoS) [13], and Ouroboros [14] consensus mechanisms take the direct transmission among a few delegates, while the propagation among other network nodes remains unquestioned. Bitcoin [16] and Bitcoin-NG [17] adopt the original flooding technique in Bitcoin to diffuse messages, thereby consuming plenty of travel hops and having a low delivery ratio. Other researchers are investigating integration of BFT-based (Byzantine Fault-Tolerant) protocols [18], [19], [20], [21], [22] with existing consensus mechanisms. For example, the replication architecture ReBFT [19] succeeds the propagation in BFT, which wastes travel hops. FastBFT [21] typically relies on tree topology of participating nodes to reduce the communication overhead. Unfortunately, this is impractical in the unstructured overlay networks. Specifically, the P2P network in the public blockchain is a typical unstructured overlay network where nodes experience mobility caused by their social characteristics in terms of intermittent connectivity and frequent partitions. It is also worth noticing that the consensus mechanisms and BFT-based protocols barely focus on the promotion of propagation techniques.

We focus on the strategies [23], [24], [25], [26] which are dedicated to the propagation methods compatible with unstructured overlay networks [27], [28]. These strategies do not rely on the mobility of the nodes. For example, the P2P propagation protocol Kadcast [23] achieves efficient broadcast in terms of the propagation delay, but relies heavily on

- *X. Feng and Y. Miao (Corresponding author) are with the School of Cyber Engineering, Xidian University, Xi'an 710071, China; Guangxi Key Laboratory of Trusted Software, Guilin University of Electronic Technology, Guilin 541004, China. Email: fengxiaoqin@stu.xidian.edu.cn; ybmiao@xidian.edu.cn.*
- *J. Ma is with the School of Cyber Engineering, and the Shaanxi Key Laboratory of Network and System Security, Xidian University, Xi'an 710071, China. Email: jfma@mail.xidian.edu.cn.*
- *X. Liu is with the Key Laboratory of Information Security of Network Systems, College of Computer and Big Data, Fuzhou University, Fuzhou 350108, China. Email: snbnix@gmail.com.*
- *K.-K. R. Choo is with the Department of Information Systems and Cyber Security, The University of Texas at San Antonio, San Antonio, TX 78249 USA. Email: raymond.choo@fulbrightmail.org.*

numerous travel hops. Another transaction dissemination protocol Erlay [25] reduces the bandwidth consumption by 40% of the Bitcoin's transaction relay protocol [29]. Unfortunately, these propagation methods result in a bottleneck, leading to low delivery ratio and weak consistency which are critical to the reliability of message propagation.

In order to address the challenges of node mobility and propagation efficiency while providing messages with high consistency, we present **N**efSBFT, a social characteristic-based propagation-efficient Practical Byzantine Fault Tolerance (PBFT) protocol, to broadcast in unstructured overlay networks. NefSBFT is designed to securely and efficiently agree on the order of transactions and achieve consensus on the produced block (generated through a consensus mechanism). The core of NefSBFT is a message propagation technique that exploits the social characteristics of public blockchain nodes in terms of intermittent connectivity and frequent partitions. Thus, NefSBFT is built essentially compatible with unstructured overlay networks and needs no control of node mobility. This propagation technique is adopted in the improved FastBFT paradigm by combining lightweight secret-sharing and a trusted counter service maintained in the local Trusted Execution Environment (TEE) of each participating node. To this end, NefSBFT achieves efficient and reliable message synchronization as the premise of the consensus mechanism. In summary, this paper makes the following contributions.

- **Social characteristic-based propagation PogaSoC.** By adopting the early decision-based multicasting strategy [30], we design a blockchain-applicable propagation technique. It considers the equations which are most close to intrinsic social characteristics of blockchain nodes to update the delivery predictability in [30]. The new hop selecting rule is designed to enhance the delivery ratio.
- **Propagation-efficient consistency protocol.** By improving the propagation way and interrelated secret-sharing and secret collection (i.e., through the tree) of FastBFT and extending its applications to the transaction ordering, we devise the NefSBFT consistency protocol. It novelly achieves the dynamic message propagation on the message's whole life course when using the PBFT-integrated consensus protocols. Thus, NefSBFT fits perfectly for the unstructured overlay network of public blockchain.
- **Reliable and efficient broadcast.** High delivery ratio guaranteed in the propagation technique and consistency agreement of messages fasten the reliability of messages synchronization. A few resource is consumed because of the message complexity reduction and partial execution guaranteed by TEE's function in NefSBFT and the travel hops saving achieved by the propagation technique.

2 Related Work

Our overall goal is to create a broadcast protocol, where the network nodes synchronize their state of transactions and blocks. This protocol suffices to efficiently form a totally-ordered transaction log and globally-consistent block append with no assumption of controllable node mobility. The core of our design is the propagation technique. Effective message propagation techniques provide liveness and reliability guarantees, allowing a distributed system [31] to share information in spite of network limitations [32]. A vast body of works have studied such techniques, making various assumptions, applying to different scenarios, and offering different performance tradeoffs. We demonstrate the most closely connected efforts.

2.1 Reliable Propagation Techniques

While some schemes [26], [33] improve the propagation performance of Bitcoin transactions, the greedy approach [26] provides faster transaction spreading and lighter bifurcation. An ITM (Influence Time Minimization) problem was driven to select the seed nodes as the sources of propagation mode. The minimum time solved in ITM mitigates Bitcoin's bifurcation. However, none of the practical delivery ratio is mentioned in this greedy approach.

Many subsequent schemes [34], [35], [36], [37] offer fast propagation in blockchain systems by using *enlightened connectivity* or *financial incentive* that provides strong reliability. For example, BlockP2P-E [34] accelerates the propagation rate and retains transmission reliability. It applies a hierarchical topological structure according to node attribute classification to ensure robust connectivity. However, it clearly depends on the assumption of the controllable node mobility. Similarly is the tree routing [38] propagation. Decker et al. [36] initiated a propagation protocol by advocating financial incentive mechanisms. It provides system reliability and bandwidth alleviation even if this comes at the sacrifice of message overhead and resource consumption.

However, although these reliable propagation techniques gracefully provide efficient and reliable propagation, they still rely on mobility assumptions about the network nodes. Our work takes this issue further, guaranteeing the compatibility with unstructured overlay networks, at the same time, supporting high message consistency.

2.2 Unstructured Propagation

Deterministic mobility is impossible for most nodes of unstructured overlay networks. While the vast of consensus protocols barely focus on this issue with making mobility assumptions, some propagation techniques individually provide full consideration. Indeed we know propagation in unstructured overlay network for a variety of performance tradeoffs such as Kadcast, Erlay, and more [23], [25], [39]. Kadcast incorporated the FEC (Forward Error Correction) to increase reliability, while it consumes heavy hops and lacks consistency guarantee. Erlay incurs a few bandwidth costs even though the network connectivity increases. Although the Bitcoin network is hardened with high reliability, it still faces the problem of numerous travel hops.

Our work is most close to FastBFT [21], a PBFT series consistency protocol, and an early decision-based multicasting strategy [30] (cf. 3.3). FastBFT uses a counter service from each local TEE to decrease the number of active participating nodes from $2f + 1$ to $f + 1$. Besides, a secret-sharing framework is used to reduce the committing complexity of PBFT to a polynomial level. The multicasting strategy [30] is naturally for ad hoc network scenarios. By using the dynamically updated delivery predictability, it reduces the travel hops to multicast information, thus becomes a good candidate for unstructured overlay networks.

The key invention is a novel protocol form integrating PBFT series consistency protocol with information propagation technique that provides complete message propagation and consistency. This protocol uses the social characteristics of public blockchain nodes on the maintenance of delivery predictability to provide mobility resilience. We also make better applicability in the public blockchain P2P networks by using the most related equations to the social characteristics to update the delivery predictability. In particular, we sidestep the low delivery ratio in [30] by using a new hop selecting rule, which saves the taken travel hops at the same time. Finally, our promoted propagation technique is applied in several phases of FastBFT for transaction ordering and block validation since the transaction diffusion is out of consideration in FastBFT, besides, its block propagation closely relies on tree topology.

3 Preliminaries

In this section, we describe the transaction and block verification in Blockchain P2P network, prototype of PBFT [40] protocol, multicasting strategy that are considered when we design our propagation technique, and the adopted TEE.

3.1 Message Verification in P2P

The P2P network protocol allows full nodes $\mathcal{N}_f$ to collaboratively maintain exchanges of transactions $\mathcal{T}$ and blocks $\mathcal{B}$. In particular, the exchanges happen among network nodes as shown in Fig. 1. Full nodes download recent transactions and blocks for verification and relay only the valid one to other nodes. The validation task is undertaken by full nodes themselves, so there is no need to trust a third-party. As the nodes can send any kind of transaction, block-producing nodes $\mathcal{N}_p$, whose honesty is guaranteed through economic jeopardy, must validate the transactions before adding them to the transaction pool $\mathcal{P}oo$. To verify a transaction, each node checks the complete content. The validation of a block can be divided into two parts: 1) the validation of metadata in the block header $\mathcal{B}_h$ including the witness, which determines if $\mathcal{N}_p$ is qualified to create the new block; and 2) the validation that each transaction in the block body $\mathcal{B}_b$ is valid. In fact, having all nodes maintain an identical copy of the ledger is essential for the validation.

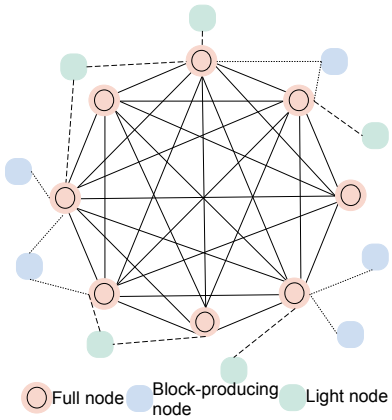


Fig. 1: Network topology of blockchain nodes.

3.2 Practical Byzantine Fault Tolerance

PBFT protocol is a way to maintain the order of transactions in a distributed network of block strings, despite the threats [41] to that order. It achieves the optimal $\frac{1}{3}$ fault tolerance. Nodes $\mathcal{N}_p, \mathcal{N}_f \in \mathcal{R}(\mathcal{R} = \{0, \cdots, \mathcal{R}-1\}, |\mathcal{R}| = 3f+1)$ in a PBFT-based distributed system are sequentially ordered with one being the primary replica $\mathcal{R}_p$ and the others referred as backup replicas $\mathcal{R}_b$s. $\mathcal{R}_b$ forwards with every *view* $\mathcal{V}$ which satisfies $\mathcal{R}_p = \mathcal{V} \, mod \, |\mathcal{R}|$. Note that any eligible nodes in the system can become the primary replica, typically in the case of $\mathcal{R}_p$ failure. The goal of PBFT is that all honest nodes help in reaching consistency regarding the order of transactions using the majority rule. On a high level, PBFT consists of five phases in the normal-case operation (cf. Fig. 2) and are described as follows:

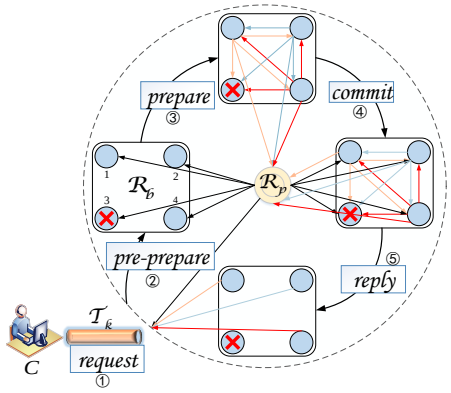


Fig. 2: Normal-case operation of PBFT.

- *The request phase.* The client $\mathcal{C}$ sends a $\mathcal{T}_k$ to $\mathcal{R}_p$.
- *The pre-prepare phase.* $\mathcal{R}_p$ produces a *pre-prepare* proposal $\left\langle \langle pre-prepare, \mathcal{V} \rangle_{\sigma_p}, \mathcal{T}_k \right\rangle$ and multicasts it to all $\mathcal{R}_b$s.
- *The prepare phase.* $\left\langle \langle pre-prepare, \mathcal{V} \rangle_{\sigma_p}, \mathcal{T}_k \right\rangle$ is verified by $\mathcal{R}_b$ upon being received. If the verification succeeds, $\mathcal{R}_b$ will multicast the *prepare* message $\langle prepare, \mathcal{V}, j \rangle_{\sigma_j}$ to all other replicas. This is the first round of voting.
- *The commit phase.* Upon receiving $\langle prepare, \mathcal{V}, j \rangle_{\sigma_j}$ from at least $\frac{2}{3}$ backups, $\mathcal{R}_b$ will now multicast a *commit* message $\langle commit, \mathcal{V}, j \rangle_{\sigma_j}$. This is another round of voting.
- *The reply phase.* $\mathcal{R}_p$ and $\mathcal{R}_b$ verify $\mathcal{T}_k$ and then send back a *reply* to $\mathcal{C}$. $\mathcal{T}_k$ is candidate successfully when $\mathcal{C}$ receives $f+1$ consistent replies from different replicas.

However, PBFT has been providing service when all replicas perform the complete agreement process. Besides, it demands a high-level communication among the replicas. The message complexity exponentially increases due to the number of multicasting messages in each phase is multiplied by each replica in $\mathcal{R}$. FastBFT has solved the two issues since only $f+1$ active replicas are required for complete agreement execution, besides, the secret-sharing technology reduces the message complexity to the polynomial level.

3.3 Multicasting Strategy

In computer networking, multicasting [42] is group communication where data transmission is addressed to a group of

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2021.3104465, IEEE Transactions on Dependable and Secure Computing

destination computers simultaneously. The early decision-based multicasting strategy [30] relies on contact history information (described by the delivery predictability) and a propagation rule to choose the travel hops. Specifically, we demonstrate it by an example as follows (cf. Fig. 3):

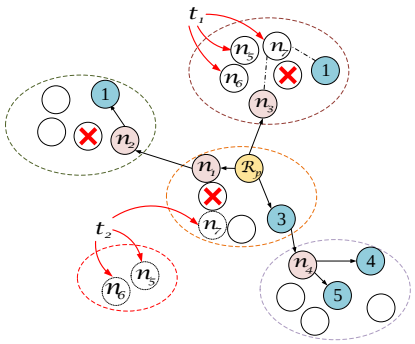


Fig. 3: Routing using decision-based multicasting.

- At time t_1, $\mathcal{R}_p$ encounters n_1, n_3, and the receiver $\mathcal{R}_b^3$. Based on the delivery predictability values of n_1, n_3 and $\mathcal{R}_b^3$ to all receivers, $\mathcal{R}_p$ finds that n_1 has higher predictability to $\mathcal{R}_b^1$ than itself, which is similar to n_3 and $\mathcal{R}_b^3$. Thus, $\mathcal{R}_p$ duplicates the multicasting packet into three: the first one including receiver $\mathcal{R}_b^1$ is sent to n_1; the second one including $\mathcal{R}_b^2$ is sent to n_3; and the last one including $\mathcal{R}_b^4$ and $\mathcal{R}_b^5$ is sent to $\mathcal{R}_b^3$.
- At time t_2, n_1 sends the duplicated packet to n_2, and $\mathcal{R}_b^3$ sends the duplicated packet to n_4. Go forward one by one, $\mathcal{R}_b^1$, $\mathcal{R}_b^4$ and $\mathcal{R}_b^5$ will receive the message packets finally. For n_5, n_6, and n_7, they have moved at this time. However, only n_3 has the packet with the receiver $\mathcal{R}_b^1$. Thus, $\mathcal{R}_b^1$ may not receive the message.

The decision-based multicasting can offer a primary use of message propagation in the unstructured overlay networks of public blockchains. However, multicasting messages are duplicated at every branching node of downstream neighbors. Besides, note that an early decision is made to split the receivers' list, thus, the messages may not reach some receivers.

3.4 Trusted Execution Environments

TEE [43], inside the main processor of a device, provides security features such as isolated code execution, the integrity of applications executed with TEE, along with confidentiality of their assets. It has exclusive access to certain hardware resources and can ignore threats from the rest of the devices. TEE has been widely used in current blockchain systems [44], [45], [46]. In general terms, we use it in NefSBFT to enable the following services:

- It is intended to provide high-level security for trusted applications (TAs) by using both hardware and software to protect data and code. We adopt the trusted counter service and secret-sharing as TAs maintained in TEE.
- TAs running in a TEE have access to the device's main processor, peripherals and memory, whereas hardware isolation protects these components from the user's installed applications. Thus, TEE offers a secure environment for the counter service and secret-sharing.
- Software and cryptographic isolation inside the TEE protect different TAs. Therefore, a unique feature of NefSBFT is that TAs of the counter service and secret-sharing pass with their operations independently.
- TEE is a trustworthy remote management infrastructure. Operations are monitored through secure channels or cryptographically secured tokens, which offer the secure distribution of secret shares. Besides, remote verifiers can ascertain the device's behavior via remote attestation. Since each participating node is deployed of TEE, each local TEE is the verifier. Thus, TEE cannot be Byzantine.
- Also, TEE only accepts code for execution that has been appropriately authorized and checked by other authorized codes. Thus, $\mathcal{R}_p$ can ask its local TEE $\mathcal{T}E_p$ for the secret share identifying of each replica by attesting $\mathcal{T}E_b$.

In conclusion, we deploy the TEE at each participating node of NefSBFT to maintain the distributed blockchain structure. Each node and the TEE execute NefSBFT as a whole, rather than the separate obligations. Note that TEE is discriminated from the server which is not secure and requires trust, thus, does not act as a centralized third-party to each node. Furthermore, all the network nodes are in charge of the crucial distributed storage, maintaining, and accounting of blockchain. In a word, TEE in NefSBFT has no influence on the blockchain decentralization.

4 NefSBFT with Efficient Propagation

In this section, we provide the system model, NefSBFT overview, and the detailed description of our NefSBFT as a social characteristic-based propagation-efficient PBFT protocol to broadcast in unstructured overlay networks. TABLE 1 demonstrates the notations used in this paper.

4.1 System Model

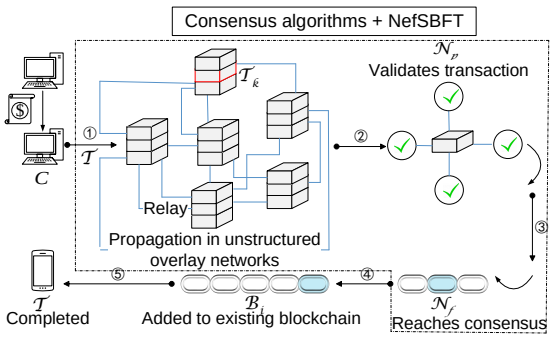


Fig. 4: The system model of public blockchains.

This section includes the main component of public blockchain system model used in our work. As shown in Fig. 4, main system entities are represented as clients, block-producing nodes, and full nodes. The key procedures of one complete system running are explained as follows:

- Step①: *Transactions propagation in the unstructured overlay network.* The transaction travels in hops to be diffused overall the network. We take the Flooding propagation of Bitcoin as an example where nodes diffuse each message to all the encountered neighbors. Considering the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2021.3104465, IEEE Transactions on Dependable and Secure Computing

TABLE 1: Notations and definitions

Notations	Descriptions
f	Tolerable faults in PBFT series protocol
$\mathcal{N}$	Nodes; $\mathcal{N}_f$ full nodes, $\mathcal{N}_p$ block-producing nodes
$\mathcal{T}$	Transactions in each epoch; $\mathcal{T} = \{\mathcal{T}_1, \cdots, \mathcal{T}_K\}$
$\mathcal{B}$	Blocks; $\mathcal{B}_i$ in i-th epoch, $\mathcal{B}_h$ header, $\mathcal{B}_b$ body
$\mathcal{P}oo$	Transaction pool
$\mathcal{R}$	Replicas; $\mathcal{R}_p$ primary, $\mathcal{R}_b$ backup
$\mathcal{V}$	View in PBFT series protocol
$\mathcal{C}$	The client
σ	signatures; σ_p of $\mathcal{TE}_p$, σ_j of $\mathcal{R}_b^j$
$\mathcal{TE}$	Local TEE; $\mathcal{TE}_p$ of $\mathcal{R}_p$, $\mathcal{TE}_b$ of $\mathcal{R}_b$
$\mathcal{D}$	Message receivers; $\mathcal{D} = \{\mathcal{R}_b^1, \cdots, \mathcal{R}_b^s\}$
$\mathcal{G}_{PLY}^{pa}$	Polynomial growth of network diffusion
$\mathcal{G}_{PLY}^{cx}$	Polynomial growth of message complexity
$\mathcal{F}$	Function; $\mathcal{F}_{VSS}$ secret-sharing, $\mathcal{F}_{TC}$ counter service
$\mathcal{R}_b$	Backup replicas; $\mathcal{R}_b^a$ active, $\mathcal{R}_b^p$ passive
$\mathcal{S}e$	Secrets; $\mathcal{S}e^j$ secret share of $\mathcal{R}_b^{a(j)}$; h_e hash of $\mathcal{S}e$
$\mathcal{N}et_{ust}^{\Delta}$	Unstructured overlay network
$\mathcal{P}d$	Delivery predictability, $\mathcal{P}d(i,j)$ of i to j
$\mathcal{M}tx$	Matrix of $\mathcal{P}d$, bea_{ij} element, bea_{ij} of $\mathcal{N}_i$ to $\mathcal{N}_j$
$\mathcal{C}n$	Counters; $\mathcal{C}n^T$ for transactions; $\mathcal{C}n^B$ for blocks
$\mathcal{C}n_{la}$	The latest counter value in $\mathcal{R}_p$; $\mathcal{C}n_j$ of $\mathcal{R}_b^{a(j)}$
ϵ, α, τ	Encountering, aging, and transitivity coefficients
$l\mu$	Time units, n total number of blockchain nodes
$\mathcal{H}o_{cu}$	Current hop node, $neighbor$ neighbors of $\mathcal{H}o_{cu}$
$\mathcal{T}h$	Delivery threshold; $\mathcal{WT}$ Wait timer
$\mathcal{T}_k \| \mathcal{B}_i$	$\mathcal{T}_k$ or $\mathcal{B}_i$ message; $(\mathcal{R}_b^a \&\& \mathcal{R}_b^p) \|\|$ message to $\mathcal{R}_b^a$, $\mathcal{R}_b^p$
$\mathcal{K}v_j$	View key of $\mathcal{R}_b^{a(j)}$
ε_e^j	Ciphertext of $\mathcal{S}e^j$ encrypted by $\mathcal{K}v_j$
$\mathcal{E}k_j$	Ciphertext of $\mathcal{K}v_j$ encrypted by $\mathcal{TE}_b^{a(j)}$'s public key
ep_i	i-th epoch; sl slot of transaction, eb epochB of block
Th_{ac}	Threshold of accepted non-primary failures

message during the transaction-advertising portion, with each iteration multiple sets (calculated by the network size) of two nodes are transmitted the message. Thus, the network diffusion grows with $2^n \sim 2^{2n}$. In fact, there is a possibility that two nodes may announce the same transaction simultaneously to each other.

- Step②: *Transactions to form a data block $\mathcal{B}_i$.* Based on the consensus mechanism, $\mathcal{N}_p$ proceeds to take a set of transactions $\mathcal{T} = \{T_1, \cdots, T_K\}$ from its $\mathcal{P}oo$ and packages them into a data structure known as the block.
- Step③: *$\mathcal{B}_i$ validation and added to the existing blockchain.* We consider $\mathcal{B}_i$ to be propagated and validated among all $\mathcal{N}_f$s through the PBFT-based consistency protocol. Thinking each message during the block-advertising portion, the network diffusion grows with the same $2^n \sim 2^{2n}$ by using Flooding. Within each verification round, a set of n_{3f+1} nodes are sent the values (*e.g.*, $\langle commit, \mathcal{V}, j \rangle_{\sigma_j}$) by the others, thus, it draws $2(n_{3f+1} - \frac{1}{2})^2 - \frac{1}{2}$ message complexity in PBFT. However, only $2(n_f + 1)$ message complexity is driven in FastBFT.
- Step④: *New blocks added behind $\mathcal{B}_i$.* Each participant adds new blocks on the majority chain with $\mathcal{B}_i$, fastens the network's immutable and auditable blockchain.
- Step⑤: *The transactions completed.* New confirmation appears with each new created block. Finally, the transactions are confirmed and the client obtains the payment.

The giant network diffusion of 2^n and high message complexity of $2(n_f - \frac{1}{2})^2 - \frac{1}{2}$ are crucial concerns in public blockchains. In our NefSBFT, the promotions focus on the Step①-Step③ that are collectively achieved by a consensus mechanism plus two executions of NefSBFT separately on transactions ordering and block verification. The propagation technique of NefSBFT is properly designed to distribute messages by capturing the social characteristics of public blockchain nodes. Thus, for each message diffused to several destinations $\mathcal{D} = \{\mathcal{R}_b^1, \cdots, \mathcal{R}_b^s\}$, it decreases the exponential diffusion growth to a polynomial growth $\mathcal{G}_{PLY}^{pa}$ (cf. 5.3.2). NefSBFT adopts the proposed propagation technique for message propagation in the FastBFT paradigm, thus, leading to a clear consistency agreement of both the transactions and block. As the same with FastBFT, it is secure and efficient to use the TEE at each participating node. In one way, it supports verifiable secret-sharing function $\mathcal{F}_{VSS}$ to decrease the parabolic factor of message complexity to the polynomial growth $\mathcal{G}_{PLY}^{cx}$ (cf. 5.3.2), in another way, it maintains the trusted counter function $\mathcal{F}_{TC}$ to reduce the active participating nodes for complete execution to $f + 1$.

4.2 NefSBFT Overview

We provide a high-level description of NefSBFT. The protocol is similar to FastBFT but differs considerably in the stressed message propagation and applicable unstructured overlay networks of public blockchains.

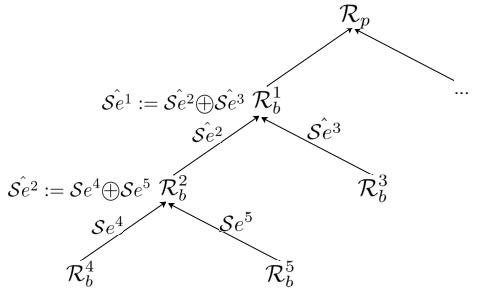


Fig. 5: Propagation tree for *Commit1* and *Reply1* of FastBFT.

While FastBFT has various superiority in terms of performance, it relies on a communicating tree to gather the shared secret. As shown in Fig. 5, the replica $\mathcal{R}_b^2$ has to firstly collect and verify the secret shares $\mathcal{S}e^4$ of $\mathcal{R}_b^4$ and $\mathcal{S}e^5$ of $\mathcal{R}_b^5$ to compose its secret share $\hat{\mathcal{S}e}^2$. The progressive operations are the same for $\hat{\mathcal{S}e}^3$, $\hat{\mathcal{S}e}^1$, and the final secret $\mathcal{S}e$. In fact, a tree of active replicas is maintained by the primary replica according to the relationships of child and father nodes. That is, there are determined connectivity and partitions. Unfortunately, all replicas $\mathcal{R}_b$s are uncontrollable of mobility, thus, it cannot be guaranteed that the child node (e.g., $\mathcal{R}_b^4$) can always directly propagate messages to its father node (e.g.,$\mathcal{R}_b^2$). In a word, FastBFT cannot be precisely used in the unstructured overlay networks. Besides, FastBFT is used as the consistency protocol rather than the message propagation. We adopt FastBFT in our design mainly for consistency during the message propagation, while the key is the promotion of the message propagation technique. It considers the social characteristics of public blockchain nodes and requires no control of node mobility. To assist the explicit and secure executions of NefSBFT protocol, we make the following practical assumptions.

Assumption 1. The blockchain nodes' social relationships are denoted by different partitions (e.g., the companies,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2021.3104465, IEEE Transactions on Dependable and Secure Computing

schools, and hospitals). Connectivity is constructed inner the partition. Thus, we assume that blockchain nodes within the same partition are cooperative (e.g., hospital departments) and familiar (families and friends). Note that this is appropriate since the public blockchain has to be applied in practical scenarios such as cloud computing, the internet of things, etc.

It worth noticing that the social characteristics of nodes in terms of intermittent connectivity and frequent partitions are self-maintained through the delivery predictability, which is rational in practice. We choose the social characteristic in NefSBFT for two reasons. 1) *For the protocol designing.* In view of the network attributes, node mobility is determined by the intermittent connectivity and frequent partitions when concentrating on message propagation, while the intermittent connectivity and frequent partitions play as two main social characteristics of public blockchain nodes. 2) *For the practice.* In distributed systems like Bitcoin, message sharing happens to drive the system. The impetus of message sharing is either the financial incentives or nodes' connection (cooperation and close relationships). Thus, social characteristics in terms of intermittent connectivity and frequent partitions which are determined by social relationships of public blockchain nodes are the key and most reliable.

Assumption 2. The protocol only assumes the acknowledgment to $\mathcal{N}_p$ and $\mathcal{N}_f$, and access to $\mathcal{F}_{TC}$ and $\mathcal{F}_{VSS}$.

The protocol proceeds in consecutive *consensus epochs*. Every epoch is one execution round of consensus algorithm in terms of $\mathcal{T}$'s validation, plus two separate executions of NefSBFT in terms of $\mathcal{T}_k$'s ordering and $\mathcal{B}_i$'s verification. Importantly, execution is divided in such a way that all $\mathcal{N}_p$s know when a new epoch starts, it derives the transaction sequence by NefSBFT and takes them for validation by the consensus mechanism. Then, it sends the produced block to $\mathcal{N}_f$ for consistency agreement via NefSBFT. Without loss of generality, the current epoch index is ep_i.

Delivery predictability. In epoch ep_i, participants execute a delivery predictability procedure to structure the blockchain propagation. NefSBFT defines the delivery predictability via the intermittent connectivity and frequent partitions of public blockchain nodes in network $\mathcal{N}et^{\Delta}_{ust}$, that is, how likely does the node $\mathcal{N}_1$ relay messages to the node $\mathcal{N}_2$. The delivery predictabilities of each node to the others are recorded into a matrix $\mathcal{M}tx$, where each element bea_{ij} is the delivery predictability of node $\mathcal{N}_i$ to node $\mathcal{N}_j$. Abstractly in practice, bea_{ij} is shaped based on the encountering, the conditions while two nodes do not encounter each other during some time units, and the transitivity from another intermediate node. Informally, each bea_{ij} is updated by $\mathcal{N}_i$ itself through several formulas specified in Section 4.3.

TEE assistance. Each local-maintained TEE mainly provides the unique, monotonic, and sequential counter service on any new message propagation. Its functions are as below.

- Each monotonic counter proposes one new request of the transaction or block in order for their agreement execution through NefSBFT.
- Later, $\mathcal{R}_p$'s local TEE bounds the one-plus counter value to the primarily committed block (in block verification) by all active $\mathcal{R}_b$s.
- At the end of each NefSBFT, the counter value sequentially adds one.

In order to achieve our NefSBFT in two different scenarios including the transaction ordering and block verification, the function $\mathcal{F}_{TC}$ is defined using different TAs that are separately mandated to process the transactions and blocks. In particular, we denote the counters separately as $\mathcal{C}n^T$ and $\mathcal{C}n^B$. Furthermore, the remote attestation by other TEEs can ascertain whether the counter values are the same currently. In conclusion, only $f+1$ active $\mathcal{R}^a_b$s are required in NefSBFT with the secure counter service. Fig. 6 demonstrates the TEE's functions during the transaction ordering and block verification, where $\mathcal{S}e$ denotes the generated secret.

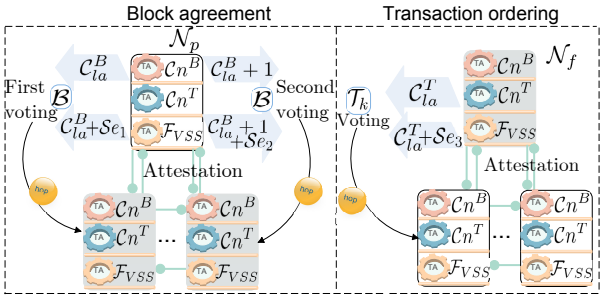


Fig. 6: The TEE function details.

Secret-sharing. The secret-sharing is adopted in NefSBFT to reduce the committing complexity. The secret-sharing $\mathcal{F}_{VSS}$ works through several aspects. (1) One secret is opened in a round of transaction ordering and two secrets are opened in block verification by connecting with the counter-bound message. Note that any two secrets can not be the same. Besides, each secret will be split into secret shares for all active replicas $\mathcal{R}^a_b$s, respectively. $\mathcal{R}^p_b$ stands for the passive replica as shown in Eq. 1.

$$\begin{aligned} \mathcal{R}^a_b &= \{\mathcal{R}^{a(1)}_b, \cdots, \mathcal{R}^{a(s)}_b\}; \\ \mathcal{R}^p_b &= \{\mathcal{R}^{p(1)}_b, \cdots, \mathcal{R}^{p(s)}_b\}; \\ a(s) + p(s) &= s = 3f+1, a(s) = f+1. \end{aligned} \tag{1}$$

(2) The secret is secure of generation, distribution, verification according to the secure storage, code execution, channels, and attestation of maintained TEEs. (3) The secret share of each $\mathcal{R}^a_b$ is signed by $\mathcal{R}^a_b$'s view key generated by $\mathcal{R}_p$. (4) Each $\mathcal{R}^a_b$ honestly prepares for the *prepare* message from $\mathcal{R}_p$ to acquire its secret share from $\mathcal{T}E_b$. Then, it locally verifies the *prepare* message using associated secret share, $\mathcal{C}n_j$, and $\mathcal{V}_j$. (5) The secret share of $\mathcal{R}^a_b$ is revealed to $\mathcal{R}_p$ during committing. In particular, the second secret share of $\mathcal{R}^a_b$ is revealed in *Reply1* phase in block verification. The secret opening in NefSBFT ensures the message reduction during committing. (6) For the secret shares of malicious or crashed $\mathcal{R}^a_b$, $\mathcal{R}_p$ uses the verifiability of the secret by secret shares and related cryptographic hash. Thereby, authentication can be proposed to $\mathcal{T}E_p$ by attesting $\mathcal{T}E_b$. Otherwise, the secret is aggregated by $\mathcal{R}_p$ through $f+1$ secrete shares of $\mathcal{R}^a_p$s, then $\mathcal{C}$ and all $\mathcal{R}^p_b$s only receive one reply message.

4.3 PogaSoC Design

We start with some notations. A block is a piece of transactions acquired through the continuous execution of NefS-

BFT. The transactions ordering spends the time *slots* and block verification spends time *epochB*. An epoch contains K *slots*, the block-producing time by a consensus algorithm, and one *epochB*. We use $\epsilon \in (0, 1]$ to indicate the encountering coefficient that works to update bea_{ij} whenever $\mathcal{N}_j$ is encountered. We define a time unit by μ. If $l\mu$ time units have elapsed since $\mathcal{N}_j$ was encountered by $\mathcal{N}_i$, α is denoted as the aging coefficient to update bea_{ij}. Besides, we denote by τ the transitivity coefficient to reflect the transitive impact of the intermediate node. The transitivity has a great influence on bea_{ij} when the intermediate node $\mathcal{N}_k$ (the neighbor of $\mathcal{N}_i$) encounters $\mathcal{N}_j$ a lot. More concrete impacts of ϵ, μ, α, and τ are given next. It is worth noticing that the transitivity of $\mathcal{N}_k$ to $\mathcal{N}_j$ leaks the network topology to its neighbor $\mathcal{N}_i$, thus we limit the use of transitive property inner the same partition within three nodes adjacent to each other based on **Assumption 1**. Furthermore, the nodes across different partitions do not provide their transitivity. However, one ensuing limitation is that the delivery predictability can not fully reflect the social characteristics of nodes in order to protect nodes' privacy, which slightly reduces PogaSoC's performance.

Abstracting social characteristics. A PBFT-based protocol comprises of a sequence of transmission operations. In PBFT-based public blockchains of $\mathcal{N}et^{\Delta}_{ust}$, the social characteristics of participants include their intermittent connectivity and frequent partitions, which results in the possible losing or establishment of communications when considering the message transmission. Thus, a consistency protocol needs a mechanism to evaluate the probability when a node in $\mathcal{N}et^{\Delta}_{ust}$ delivers messages to other nodes, since connectivity and partitions from one to the other nodes may vary in the middle execution of such a PBFT protocol. Fortunately, such a mechanism is implicitly described by the delivery predictability $\mathcal{P}d$ in PogaSoC. This $\mathcal{P}d$ can be made explicit by measuring the common encountering conditions of each $\mathcal{N}_j$, non-encountering duration while $\mathcal{N}_j$ was not delivered of any message, and the transitive property of the intermediate node from $\mathcal{N}_i$ to $\mathcal{N}_j$.

Specifically, at the core of the PogaSoC is the delivery predictability simulated by $\mathcal{M}tx$ that allows each node to maintain the delivery predictabilities $bea_{i\cdot}$ from itself to the destined nodes. Three different equations are especially for public blockchain nodes to update the delivery predictability, which corresponds to the above three conditions.

- The condition (ENCOUNTER, $\mathcal{N}_j$, transaction/block) refers that a message is reach $\mathcal{N}_j$ from $\mathcal{N}_i$, that is, $\mathcal{N}_j$ is directly encountered. It results in $\mathcal{N}_i$ updating bea_{ij} and storing it in its local delivery predictability column in $\mathcal{M}tx$ according to Eq. 2:

$$Pd(i,j)_E = Pd(i,j)_{old} + (1 - Pd(i,j)_{old}) \cdot \epsilon;$$
$$\mathcal{M}tx = \begin{pmatrix} bea_{11} & \cdots & bea_{1n} \\ \vdots & \vdots & \vdots \\ \cdots & Pd(i,j)_E & \cdots \\ \vdots & \vdots & \vdots \\ bea_{n1} & \cdots & bea_{nn} \end{pmatrix} \quad (2)$$

  Note that $Pd(i,j)_{old}$ denotes the old value of delivery predictability. Since ϵ is adopted in the range $(0, 1]$ and Eq. 2 is designed for the encountering circumstance, bea_{ij} is updated to a larger value each time $\mathcal{N}_j$ is encountered.
- The condition (NON-ENCOUNTER, $\mathcal{N}_j$) refers that any message has not been transmitted to $\mathcal{N}_j$ from $\mathcal{N}_i$ for several time units $l\mu$, that is, $\mathcal{N}_j$ has not been encountered. It results $\mathcal{N}_i$ aging bea_{ij} with time and storing it in its local delivery predictability column in $\mathcal{M}tx$ according to Eq. 3:

$$Pd(i,j)_N = Pd(i,j)_{old} \cdot \alpha^{l\mu};$$
$$\mathcal{M}tx = \begin{pmatrix} bea_{11} & \cdots & bea_{1n} \\ \vdots & \vdots & \vdots \\ \cdots & Pd(i,j)_N & \cdots \\ \vdots & \vdots & \vdots \\ bea_{n1} & \cdots & bea_{nn} \end{pmatrix} \quad (3)$$

  As we argue that the delivery predictability must age for time given that $\mathcal{N}_j$ has not been transmitted to any message by $\mathcal{N}_i$. Note that $\mathcal{N}_j$ is not a good forwarder in this case. Thus, α is adopted in the range $[0, 1)$ and Eq. 3 is designed as a monotonously decreasing exponential function, so that bea_{ij} is updated to a smaller value.
- The condition (TRANSITIVITY, $\mathcal{N}_i$, $\mathcal{N}_k$, $\mathcal{N}_j$, transaction/block) refers that the intermediate $\mathcal{N}_k$ of $\mathcal{N}_i$ encounters $\mathcal{N}_j$ a lot. That is, although $\mathcal{N}_j$ is not encountered by $\mathcal{N}_i$, while $\mathcal{N}_j$ is frequently encountered by the neighbor $\mathcal{N}_k$ of $\mathcal{N}_i$. Thereby, $\mathcal{N}_j$ plays a good forwarder. $\mathcal{N}_i$ then invokes Eq. 4 to update bea_{ij}. Note that $\mathcal{N}_i$, $\mathcal{N}_k$ and $\mathcal{N}_j$ belong to the same partition.

$$Pd(i,j)_T = Pd(i,j)_{old} + (1 - Pd(i,j)_{old}) \cdot Pd(i,k) \cdot Pd(k,j) \cdot \tau;$$
$$\mathcal{M}tx = \begin{pmatrix} bea_{11} & \cdots & bea_{1n} \\ \vdots & \vdots & \vdots \\ \cdots & Pd(i,j)_T & \cdots \\ \vdots & \vdots & \vdots \\ bea_{n1} & \cdots & bea_{nn} \end{pmatrix} \quad (4)$$

  The transitivity property means that the delivery predictability of the transitive node $\mathcal{N}_k$ has an impact on that of $\mathcal{N}_i$ to $\mathcal{N}_j$, and this impact can be determined by τ-factored Eq. 4. Also, since τ is adopted in the range $(0, 1]$ and Eq. 4 is designed for the transitive circumstance, bea_{ij} is updated to a larger value.

Each node maintains a row in $n * n$ matrix $\mathcal{M}tx$, which locally records the delivery predictabilities of the node to the others. Note that the diagonal entry bea_{ii} is meaningless.

Choosing the next travel hop. Using the outputs in $\mathcal{M}tx$ and a propagation rule to decide the next hop for each message, PogaSoC not only allows for a high delivery ratio but also consumes a few travel hops for efficient delivery. At a high level, PogaSoC allows to propagate messages from the message owners to all $\mathcal{D}$ and is therefore appropriate to be used in the phases of *Secret-sharing*, *Prepare*, *Commit2*, and *Reply2/Reply* of the NefSBFT protocol. The core propagation rule during this process is to choose a node as the next hop if it has the higher delivery predictability to one receiver in $\mathcal{D}$ than a defined delivery threshold $\mathcal{T}h$ before a wait timer $\mathcal{WT}$ expires. Unfortunately, to be the current hop $\mathcal{H}o_{cu}$ to $\mathcal{R}_b \in \mathcal{D}$ reveals a probabilistic network

topology ($Pd(\mathcal{H}o_{cu}, \mathcal{R}_b) > \mathcal{T}h$) of $\mathcal{H}o_{cu}$ to $\mathcal{R}_b$. This leaks information especially when $\mathcal{H}o_{cu}$ is across the different partitions. Hence, we design the PoSoC to provide a balance between the node's privacy and PogaSoC's performance. For the performance-conscious propagation, PogaSoC sets the $\mathcal{T}h$ with the value at best performance, while it sets $\mathcal{T}h$ with a lower value for the privacy-conscious propagation. Note that the ensuing limitation is that we cannot just select the performance-optimistic parameter $\mathcal{T}h$ to protect nodes privacy, while the worst case of PogaSoC is the Flooding. The detailed PogaSoC propagation is explained as follows.

$\mathcal{C}$ or $\mathcal{N}_p$ propagates the transaction or block to $\mathcal{R}_p \in \mathcal{N}_f$. After verifying the message through Val_exch($\mathcal{T}, \mathcal{B}$) in **Algorithm.** 1, the propagation proceeds in one of the following three modes depend on the results of $Pd(neighbor, \mathcal{D}) > \mathcal{T}h$ from the encountered nodes.

Algorithm 1 Val_exch($\mathcal{T}, \mathcal{B}$)

Input: $\mathcal{T} = \{\mathcal{T}_k\} = \{(format, inputs, outputs, signatures)\}$
$= \{(\mathcal{T}\mathcal{F}o, \mathcal{T}\mathcal{I}p, \mathcal{T}\mathcal{O}p, \mathcal{T}\int g)\}$
Output: $\mathcal{B} = \{\mathcal{B}_h = (metadata), \mathcal{B}_b = (\mathcal{T})\}$
while $\mathcal{N}_f$ has complete network state **do**
  if $\mathcal{N}_f$ receives $\mathcal{T}_k \in \mathcal{T}$ **then**
    for $\mathcal{T}_k$ and depending on network parameters **do**
      Step1: $\mathcal{N}_f$ checks $\mathcal{T}\mathcal{F}o$, $\mathcal{T}\mathcal{I}p$,$\mathcal{T}\mathcal{O}p$ and $\mathcal{T}\int g$, etc.;
      Step2: $\mathcal{N}_f$ determines that $\mathcal{T}_k$ is valid;
      Step3: $\mathcal{T}_k$ is transcribed in a secured record in $\mathcal{P}oo$;
    end
    $\mathcal{N}_f$ broadcasts $\mathcal{T}_k$ to $\mathcal{N}_p$;
    $\mathcal{N}_p$ has consistent transaction sequence of $\mathcal{T}$;
  end
  $\mathcal{N}_p$ produces a new block $\mathcal{B}_i$ with $\mathcal{T}$;
  while $\mathcal{N}_p$ broadcasts $\mathcal{B}_i$ to $\mathcal{N}_f$ **do**
    for $\mathcal{B}_i$ and depending on network parameters **do**
      Step1: $\mathcal{N}_f$ validates metadata in $\mathcal{B}_h$;
      Step2: $\mathcal{N}_f$ validates $\mathcal{T}$ in $\mathcal{B}_b$;
      if $\mathcal{B}_i$ is compliant to the agreed protocol **then**
        $\mathcal{N}_f$ adds $\mathcal{B}_i$ to local copy of blockchain;
      end
      else if the ledgers are not identical **then**
        $\mathcal{N}_f$ disagrees over the facts written;
      end
    end
  end
  A network-wide consensus on the valid system state is reached;
  $\mathcal{N}_p$ commence new race to solve a block of next transaction set.
end

(1) In the optimistic mode, there is at least one qualified node to each $\mathcal{R}_b \in \mathcal{D}$. $\mathcal{R}_p$ separately interacts with the nodes that have the largest $Pd(i, \mathcal{R}_b)$ to propagate the message.
(2) In the semi-optimal mode, i.e., there exist the qualified nodes to part of receivers, the propagation is same as (1) in view of these receivers. In addition, $\mathcal{R}_p$ holds the message headed with the left destinations until the qualified node to a destination is encountered. Otherwise, $\mathcal{R}_p$ multicasts the headed message to each corresponding hop which has the largest $Pd(i, \mathcal{R}_b)$ before $\mathcal{WT}$ expires. Therefore, the message will arrive at $\mathcal{D}$ prior to this wait timer $\mathcal{WT}$.
(3) In the worst mode, there are none of the qualified nodes. $\mathcal{R}_p$ will cache the message until a triggering by qualified nodes or expiring $\mathcal{WT}$ to multicast this message.

Receiving this message will enable the current hop node $\mathcal{H}o_{cu}$ to proceed the multicasting. As shown in Eq. 5, the node $\mathcal{H}o_{cu}$ indeed is qualified since $Pd(\mathcal{H}o_{cu}, \mathcal{R}_b^1) = max\{Pd(i, \mathcal{R}_b^1)(i \in neighbor)\} > \mathcal{T}h$ and $Pd(\mathcal{H}o_{cu}, \mathcal{R}_b^s) = max\{Pd(i, \mathcal{R}_b^s)(i \in neighbor)\} > \mathcal{T}h$. Recall that $Pd(neighbor, \mathcal{R}_b^1)$ ($neighbor$ of $\mathcal{H}o_{cu}$) is checked to select the next hop to $\mathcal{R}_b^1$.

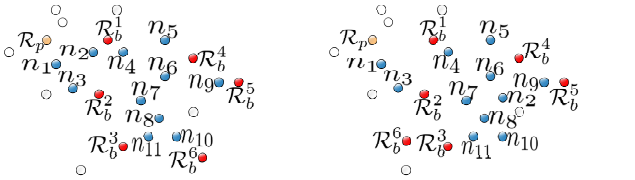
$$
\begin{array}{l}
 \\
\mathcal{N}_1 \\
\mathcal{N}_2 = \mathcal{R}_p \\
\vdots \\
\mathcal{N}_i = \boxed{\mathcal{H}o_{cu}} \\
\mathcal{N}_{i+1}(bea_{i+1}, \cdot) \\
\vdots \\
\mathcal{N}_n
\end{array}
\begin{array}{c}
\begin{array}{cccccc}
\mathcal{N}_1 & \mathcal{R}_p & \mathcal{R}_b^1 & \cdots & \mathcal{R}_b^s & \mathcal{N}_n
\end{array} \\
\begin{pmatrix}
- & 0.09 & 0.12 & \cdots & 0.4 & 0.05 \\
0.12 & - & 0.07 & \cdots & 0.22 & 0.35 \\
\vdots & \cdots & \cdots & \cdots & \cdots & \vdots \\
0.22 & 0.08 & 0.33 & \cdots & 0.29 & 0.26 \\
0.04 & 0.33 & 0.16 & \cdots & 0.15 & 0.13 \\
\vdots & \cdots & \cdots & \cdots & \cdots & \vdots \\
0.1 & 0.26 & 0.05 & \cdots & 0.19 & -
\end{pmatrix}
\end{array}
\tag{5}
$$

We notice the consequence of PogaSoC is that the node's network topology to its direct neighbor inner a partition can be driven through the transitive property only by its direct neighbors. This leads to the de-anonymization attack by the internal malicious node. We avoid the transitive property in the privacy-preserving partition (the ensuing limitation is degraded presentation of full social characteristics) and make the following assumption.

Assumption 3. We assume the malicious node is eager to de-anonymization attack on other partitions because of the social relationships with its partition's members.

We explain an example of PogaSoC to multicast a transaction message $\mathcal{T}_k$ to the destinations $\mathcal{D} = \{\mathcal{R}_b^1, \cdots, \mathcal{R}_b^6\}$.

Exam PogaSoC $\mathcal{T}_k$: For clarity we conclude with an example of the PogaSoC execution as shown in Fig. 7. Assume that $\mathcal{R}_p$ is mandated to multicast $\mathcal{T}_k$ to $\mathcal{R}_b^1, \cdots, \mathcal{R}_b^6$. The following steps proceed then.



t_1

bea_{ij}	$\mathcal{R}_b^1$	$\mathcal{R}_b^2$	$\mathcal{R}_b^3$	$\mathcal{R}_b^4$	$\mathcal{R}_b^5$	$\mathcal{R}_b^6$
n_1	0.22	0.24	0.17	0.11	0.08	0.11
n_2	0.31	0.20	0.14	0.16	0.07	0.09

t_2

bea_{ij}	$\mathcal{R}_b^1$	$\mathcal{R}_b^2$	$\mathcal{R}_b^3$	$\mathcal{R}_b^4$	$\mathcal{R}_b^5$	$\mathcal{R}_b^6$
n_1	0.22	0.24	0.17	0.11	0.08	0.16
n_2	0.17	0.19	0.20	0.25	0.27	0.12

Fig. 7: PogaSoC propagation example.

Step1: When propagating $\mathcal{T}_k$, $\mathcal{R}_p$ encounters n_1 and n_2. $\mathcal{R}_p$ goes to the optimistic, semi-optimal, or worst mode by checking the results $bea_{ij}(i = 1, 2; j = 1, \cdots, 6) > \mathcal{T}h = 0.15$. Note that $bea_{21} > bea_{11} > 0.15$, $bea_{12} > bea_{22} > 0.15$, $bea_{13} > 0.15$, and $bea_{24} > 0.15$. Hence, the command of semi-optimal mode is operational. $\mathcal{R}_p$ copies the message as a $(\mathcal{R}_b^2 \&\& \mathcal{R}_b^3)||\mathcal{T}_k$ to n_1 and a $(\mathcal{R}_b^1 \&\& \mathcal{R}_b^4)||\mathcal{T}_k$ to n_2. These messages proceed to be multicasted only if they are verified correct, otherwise, they are not cached into $\mathcal{P}oo_{n_1}$ and $\mathcal{P}oo_{n_2}$. The message multicasting to $\mathcal{R}_b^5$ and $\mathcal{R}_b^6$ will be performed until a command of the worst mode is triggered.

Step2: The multicasting processes of n_1 and n_2 are analogous. At time t_1, n_1 encounters n_3 where $(bea_{32} \& bea_{33}) > 0.15$. Recall that the optimistic mode is met, n_1 next sends $(\mathcal{R}_b^2 \&\& \mathcal{R}_b^3)||\mathcal{T}_k$ to n_3. Similarly, n_2 sends $\mathcal{R}_b^1||\mathcal{T}_k$ to $\mathcal{R}_b^1$ and

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2021.3104465, IEEE Transactions on Dependable and Secure Computing

$\mathcal{R}_b^4||\mathcal{T}_k$ to n_4. Since $\mathcal{R}_b^1$ receives $\mathcal{T}_k$ before $\mathcal{WT}$ expires, it replies to $\mathcal{R}_p$ with the committing message.

Step3: At time $t_2(t_2 < \mathcal{WT})$, there is $(bea_{16}\&bea_{25}) > 0.15$ since the nodes' mobility caused by intermittent connectivity and frequent partitions results in the variation of delivery predictability. Then, $\mathcal{R}_p$ proceeds to send a $\mathcal{R}_b^6||\mathcal{T}_k$ to n_1, and a $\mathcal{R}_b^5||\mathcal{T}_k$ to n_2. The next commands are the same with $\mathcal{H}o_{cu}$ multicasting $\mathcal{T}_k$ to $\mathcal{R}_b^2 \sim \mathcal{R}_b^6$. Thus, $\mathcal{T}_k$ will be finally propagated to all $\mathcal{D}$ before $\mathcal{WT}$ expires. $\mathcal{R}_p$ will remove the multicasting message until it has received at least $f+1$ replies.

At this point, the PogaSoC technique is designed completely.

4.4 Formal Protocol Operation

The NefSBFT protocol is explained in detail in this section. For completeness, the description includes a series of phases (cf. Fig. 8.) in terms of *Secret-sharing*, *Prepare*, *Commit1*, *Commit2*, *Reply1*, and *Reply2*/*Reply*.

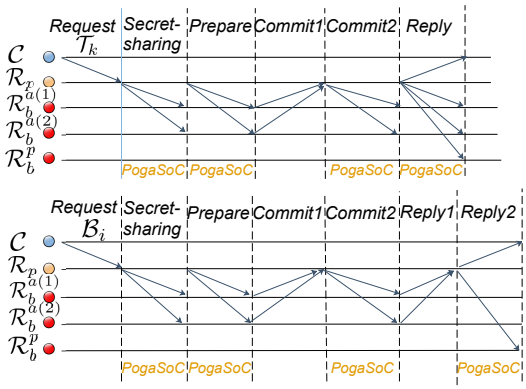


Fig. 8: Main phases of NefSBFT protocol.

Transaction/block verification (*The Request phase*). We denote the $\mathcal{T}_k$ or $\mathcal{B}_i$ message in NefSBFT with $(\mathcal{R}_b^a\&\&\mathcal{R}_b^p)||\mathcal{T}_k\|\mathcal{B}_i$. To multicast $(\mathcal{R}_b^a\&\&\mathcal{R}_b^p)||\mathcal{T}_k\|\mathcal{B}_i$, a full node is connected to be $\mathcal{R}_p$ in our protocol, in addition, $\mathcal{R}_p$ sets its view as $\mathcal{V} = \mathcal{V}+1$. A crash of the detection and computing on $\mathcal{T}_k$ is performed to verify that $(\mathcal{TF}_o, \mathcal{TI}_p, \mathcal{TO}_p, \mathcal{TS}_g) = 1$. To capture the high-level security, $\mathcal{T}_k$ will be verified at each of the receivers.

Secret-sharing (*The Secret-sharing phase*). The secret-sharing with any new request $\mathcal{R}_b^a||\mathcal{T}_k\|\mathcal{B}_i$ is straightforward. Upon receiving $\mathcal{R}_b^a||\mathcal{T}_k\|\mathcal{B}_i$, $\mathcal{R}_p$ first generates the secret $\mathcal{S}e$ ($\mathcal{S}e_1$ only for $\mathcal{T}_k$, $\mathcal{S}e_1$ and $\mathcal{S}e_2$ for $\mathcal{B}_i$) as shown in Eq. 6 by invoking $\mathcal{TE}_p$. Note that r is the randomly generated coefficient. $\mathcal{S}e$ is computed of hash as $h_{e1} \leftarrow H(\mathcal{S}e_1, (\mathcal{C}n_{la}, \mathcal{V}))$ and $h_{e2} \leftarrow H(\mathcal{S}e_2, (\mathcal{C}n_{la}+1, \mathcal{V}))$.

$$\begin{aligned} f_{e1}(x) &= \mathcal{S}e_1 + r_{11}x^1 + \cdots + r_{f1}x^f; \\ f_{e2}(x) &= \mathcal{S}e_2 + r_{12}x^1 + \cdots + r_{f2}x^f. \end{aligned} \tag{6}$$

$\mathcal{TE}_p$ randomly splits $\mathcal{S}e$ into $f+1$ secret shares and sends them to the active $\mathcal{R}_b^a$s by channels to $\mathcal{TE}_b^a$s, respectively. Specifically, it calculates $\mathcal{S}e^1 \oplus \cdots \oplus \mathcal{S}e^{f+1} \leftarrow \mathcal{S}e$ and $h_e^j \leftarrow H(\mathcal{S}e^j)$, where $\mathcal{S}e^j = f_e(x_j)$. $\mathcal{TE}_p$ generates a view key $\mathcal{K}v_j \leftarrow \{0,1\}^k$ for each $\mathcal{R}_b^{a(j)}$. Thus, Each secret share $\mathcal{S}e^j$ is encrypted using authenticated encryption as $\varepsilon_e^j \leftarrow E(\mathcal{K}v_j, \langle (x_j, f_e(x_j)), (\mathcal{C}n, \mathcal{V}), h_e^j, h_e\rangle)(\mathcal{C}n = \mathcal{C}n_{la}\, or\, \mathcal{C}n_{la}+1)$. The related $\mathcal{C}n$ of $\mathcal{TE}_p$ is signed into $\langle h_e, (\mathcal{C}n, \mathcal{V})\rangle_{\sigma_p} \leftarrow Sign(\langle h_e, (\mathcal{C}n, \mathcal{V})\rangle)$ by the $\mathcal{TE}_p$'s signing key, which in essence binds $\mathcal{S}e$ to the $\mathcal{C}n$-fixed $\mathcal{R}_b^a||\mathcal{T}_k\|\mathcal{B}_i$ (see the *Prepare*). In conclusion, the messages of the secret shares are

$$\begin{aligned} &\{\langle h_{e1}, (\mathcal{C}n_{la}, \mathcal{V})\rangle_{\sigma_p}, \varepsilon_{e1}^j\}_{\mathcal{C}n_{la}}; \\ &\{\langle h_{e2}, (\mathcal{C}n_{la}+1, \mathcal{V})\rangle_{\sigma_p}, \varepsilon_{e2}^j\}_{\mathcal{C}n_{la}+1}. \end{aligned} \tag{7}$$

The secret shares are transmitted to $\mathcal{TE}_b^{a(j)}$ of each $\mathcal{R}_b^{a(j)}$ via the proposed PogaSoC propagation technique, in the form of $\mathcal{R}_b^{a(j)}||\{\langle h_{e1}, (\mathcal{C}n_{la}, \mathcal{V})\rangle_{\sigma_p}, \varepsilon_{e1}^j\}_{\mathcal{C}n_{la}}||\{\langle h_{e2}, (\mathcal{C}n_{la}+1, \mathcal{V})\rangle_{\sigma_p}, \varepsilon_{e2}^j\}_{\mathcal{C}n_{la}+1}$.

Prepare for the request (*The Prepare phase*). The signed message $((\mathcal{R}_b^a\&\&\mathcal{R}_b^p)||\mathcal{T}_k\|\mathcal{B}_i)_\sigma$ of C or $\mathcal{N}_p$ will now be transmitted to $\mathcal{R}_b^a$ with PogaSoC in the form of the *Prepare* message. Concretely, $M = (\mathcal{R}_b^a||\mathcal{T}_k\|\mathcal{B}_i)_\sigma$ has to be bound to $(\mathcal{C}n_{la}, \mathcal{V})$ by the signing of $\mathcal{TE}_p$, i.e., $\langle H(M), (\mathcal{C}n_{la}, \mathcal{V})\rangle_{\sigma_p} \leftarrow Sign(\langle H(M), (\mathcal{C}n_{la}, \mathcal{V})\rangle)$. Then, $\mathcal{R}_p$ includes all components of *Prepare* message as $\langle Prepare, M, \langle H(M), (\mathcal{C}n_{la}, \mathcal{V})\rangle_{\sigma_p}\rangle$ to be multicasted to $\mathcal{R}_b^a$. At this point, M is prepared.

Commit to the first round of voting (*The Commit phase*). After receiving $\langle Prepare, M, \langle H(M), (\mathcal{C}n_{la}, \mathcal{V})\rangle_{\sigma_p}\rangle$, each $\mathcal{R}_b^a$ confirms the first round of voting by committing to $\mathcal{R}_p$ in the form of secret share revealing. First, the bound secret share in $\mathcal{TE}_b^{a(j)}$ will be sent to $\mathcal{R}_b^{a(j)}$ if $\mathcal{R}_b^{a(j)}$ provides the formal $\langle Prepare, M, \langle H(M), (\mathcal{C}n_{la}, \mathcal{V})\rangle_{\sigma_p}\rangle$. Having the input $\langle H(M), (\mathcal{C}n_{la}, \mathcal{V})\rangle_{\sigma_p}$, ε_{e1}^j and $\langle h_{e1}, (\mathcal{C}n_{la}, \mathcal{V})\rangle_{\sigma_p}$, $\mathcal{R}_b^{a(j)}$ then initiates the verification process (cf. **Algorithm.** 2).

Algorithm 2 *Veri_mess*$(\langle H(M), (\mathcal{C}n, \mathcal{V})\rangle_{\sigma_p}, \varepsilon_{e1}^j)$

Input: $\langle H(M), (\mathcal{C}n', \mathcal{V}')\rangle_{\sigma_p}, \varepsilon_{e1}^j, \langle h''_{e1}, (\mathcal{C}n'', \mathcal{V}'')\rangle_{\sigma_p}$
Output: M
while holding ε_{e1}^j **do**
  if Vrfy$(\langle H(M), (\mathcal{C}n', \mathcal{V}')\rangle_{\sigma_p} = 0)$ **then**
    return *"invalid signature"*;
  end
  else if Vrfy$(\langle h''_{e1}, (\mathcal{C}n'', \mathcal{V}'')\rangle_{\sigma_p} = 0)$ **then**
    return *"invalid signature"*;
  end
  else if $\langle \mathcal{S}e_1^j, (\mathcal{C}n''', \mathcal{V}'''), h_{e1}^j, h'_{e1}\rangle \leftarrow D(\varepsilon_{e1}^j)$ fails **then**
    return *"invalid encryption"*;
  end
  else if $(\mathcal{C}n', \mathcal{V}') \neq (\mathcal{C}n'', \mathcal{V}'') \neq (\mathcal{C}n''', \mathcal{V}''')$ **then**
    return *"invalid counter value"*;
  end
  else if $\mathcal{C}n' \neq \mathcal{C}n_{la}$ **then**
    return *"invalid counter value"*;
  end
  else if $h'_{e1} \neq h''_{e1}$ **then**
    return *"invalid hash"*;
  end
  else
    return *"valid M"*.
  end
end

Specifically, $\mathcal{R}_b^{a(j)}$ can verify all (1) validity of σ_p, (2) effectiveness of $E(\mathcal{K}v_j, \langle\rangle)$, (3) reliability of $(\mathcal{C}n_{la}, \mathcal{V})$ inside $\langle H(M), (\mathcal{C}n', \mathcal{V}')\rangle_{\sigma_p}, \varepsilon_{e1}^j$ and $\langle h_{e1}, (\mathcal{C}n'', \mathcal{V}'')\rangle_{\sigma_p}$, and (4) correctness of h_{e1}. At this point, M has been verified. $\mathcal{R}_b^{a(j)}$ directly recoveries $\langle \mathcal{S}e_1^j, h_{e1}^j, h_{e1}\rangle$ through the decryption by $\mathcal{K}v_j$. Note that $\mathcal{K}v_j$ is encrypted as $\mathcal{E}k_j \leftarrow Enc(\mathcal{K}v_j)$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2021.3104465, IEEE Transactions on Dependable and Secure Computing

by $\mathcal{TE}_p$ using $\mathcal{TE}_b^{a(j)}$'s public key. Then, we capture the correctness of $h_{e1}^j \leftarrow H(\mathcal{S}e_1^j)$. $\mathcal{R}_b^{a(j)}$ will initiate the committing if the verification succeeds. Each $\mathcal{R}_b^{a(j)}$ reveals its $\mathcal{S}e_1^j$ by sending the message $\langle \mathcal{S}e_1^j, h_{e1}^j \rangle_{\sigma_j}$ to $\mathcal{R}_p$.

$(\mathcal{R}_b^a||\mathcal{T}_k||\mathcal{B}_i)_\sigma$ is successfully committed by $\mathcal{R}_b^{a(j)}$ once $\mathcal{R}_p$ gives a *Commit* reply aiming at $\langle \mathcal{S}e_1^j, h_{e1}^j \rangle_{\sigma_j}$. In a detail, in order to recover $\mathcal{S}e_1$, $\mathcal{R}_p$ uses at least $f+1$ secret shares $\mathcal{S}e_1^1, \cdots, \mathcal{S}e_1^{f+1}$ as the inputs of Eq. 8,

$$\mathcal{S}e_1 = \Sigma_{j=1}^{f+1} f_{e1}(x_j) \Pi_{i \neq j} \frac{x_i}{x_i - x_j}. \tag{8}$$

$\mathcal{S}e_1$ is proven of correctness by $h_{e1} \leftarrow H(\mathcal{S}e_1)$. Note that $\mathcal{R}_p$ asks $\mathcal{TE}_p$ for the detection of malicious $\mathcal{R}_b^{a(j)}$ by offering $\langle Prepare, M, \langle H(M), (\mathcal{C}n_{la}, \mathcal{V}) \rangle_{\sigma_p} \rangle$, h_{e1}, and $\mathcal{S}e_1^j$ when $H(\mathcal{S}e_1) \neq h_{e1}$. Such a malicious/crashed $\mathcal{R}_b^{a(j)}$ which submitted a false secret share or failed to provide the secret share can be captured by $\mathcal{TE}_p$ through attesting $\mathcal{TE}_b^{a(j)}$, and will then be substituted of a $\mathcal{R}_b^p$. Otherwise, $\mathcal{R}_p$ technically verifies $\mathcal{B}_i$ in a further step in terms of $metadata$ in $\mathcal{B}_h$ and $\mathcal{T}$ in $\mathcal{B}_b$ for only the message $M_1 = ((\mathcal{R}_b^a \&\& \mathcal{R}_b^p)||\mathcal{B}_i)_{\sigma_{N_p}}$. The verification result will be added as the *proof* for $\mathcal{B}_i$. To conclude this first round of committing (at the same time starting the second round of voting for M_1), $\mathcal{R}_p$ multicasts a $\langle Commit, \mathcal{S}e_1, proof, \langle H(M_1||proof), (\mathcal{C}n_{la}+1, \mathcal{V}) \rangle_{\sigma_p} \rangle$ or $\langle Commit, \mathcal{S}e_1, \langle H(M_2), (\mathcal{C}n_{la}, \mathcal{V}) \rangle_{\sigma_p} \rangle (M_2 = ((\mathcal{R}_b^a \&\& \mathcal{R}_b^a)||\mathcal{T}_k)_{\sigma_c}$ to all $\mathcal{R}_b^a$s according to PogaSoC.

Further, M_2 in $\langle Commit, \mathcal{S}e_1, \langle H(M_2), (\mathcal{C}n_{la}, \mathcal{V}) \rangle_{\sigma_p}$ is checked of $h_{e1} \leftarrow H(\mathcal{S}e_1)$ in each $\mathcal{R}_b^{a(j)}$. This means that $\mathcal{T}_k$ is finally committed and confirmed by at least other f $\mathcal{R}_b^a$s at the same time and can be added into $\mathcal{P}oo_b^{a(j)}$ of each $\mathcal{R}_b^{a(j)}$. While for $M_1 = (\mathcal{R}_b^a||\mathcal{B}_i)_{\sigma_{N_p}}$, $\mathcal{R}_b^{a(j)}$ proceeds the second round of voting to prove the statement *proof*.

Commit to the second round of voting (*The Reply phase*). To catch an agreement on the $proof$, we follow the steps of $\mathcal{S}e_2$ revealing (as the $\mathcal{S}e_1$ revealing in *Commit1*). $\mathcal{R}_b^{a(j)}$ captures the effectiveness of $\langle H(M_1||proof), (\mathcal{C}n_{la}+1, \mathcal{V}) \rangle_{\sigma_p}$ once receiving the message $\langle Commit, \mathcal{S}e_1, proof, \langle H(M_1||proof), (\mathcal{C}n_{la} + 1, \mathcal{V}) \rangle_{\sigma_p} \rangle$. In a sense, $\mathcal{R}_b^{a(j)}$ performs $h_{e1} \leftarrow H(\mathcal{S}e_1)$ and checks the *proof* against $metadata$ and $\mathcal{T}$. We refer to the *View change* in common PBFT protocols if there is $(H(\mathcal{S}e_1) = h_{e1}) \cap (proof) = 0$. Otherwise, the similar operations as in *Commit1* are executed to open $\mathcal{S}e_2$. In a detail, $\mathcal{R}_b^{a(j)}$ interacts with its $\mathcal{TE}_b^{a(j)}$ to obtain $\mathcal{S}e_2^{a(j)}$ by offering $\langle Commit, \mathcal{S}e_1, proof, \langle H(M_1||proof), (\mathcal{C}n_{la}+1, \mathcal{V}) \rangle_{\sigma_p} \rangle$. Also, $\mathcal{R}_b^{a(j)}$ executes the verification based on **Algorithm.** 2 and reveals $\langle \mathcal{S}e_2^j, h_{e2}^j \rangle_{\sigma_j}$ to $\mathcal{R}_p$ if the verification is valid. At this point, the proved $(\mathcal{R}_b^a||\mathcal{B}_i)_{\sigma_{N_p}}$ is committed by at least $f+1$ active replicas when $\mathcal{R}_p$ calculates $h_{e2} \leftarrow H(\mathcal{S}e_2)$.

Informally, as the last step, $\mathcal{R}_p$ proposes a *Reply* message $\langle Reply, M_1, proof, \mathcal{S}e_1, \mathcal{S}e_2, \langle h_{e1}, (\mathcal{C}n_{la}, \mathcal{V}) \rangle_{\sigma_p}, \langle h_{e2}, (\mathcal{C}n_{la}+1, \mathcal{V}) \rangle_{\sigma_p}, \langle H(M_1), (\mathcal{C}n_{la}, \mathcal{V}) \rangle_{\sigma_p}, \langle H(M_1||proof), (\mathcal{C}n_{la} + 1, \mathcal{V}) \rangle_{\sigma_p} \rangle$ to send to $\mathcal{C}$ as well as each passive replica $\mathcal{R}_b^{p(j)}$ by PogaSoC. Similar to $M_2 = ((\mathcal{R}_b^a \&\& \mathcal{R}_b^p \&\& \mathcal{C})||\mathcal{T}_k)_{\sigma_c}$, the $\langle Reply, M_2, \mathcal{S}e_1, \langle h_{e1}, (\mathcal{C}n_{la}, \mathcal{V}) \rangle_{\sigma_p}, \langle H(M_2), (\mathcal{C}n_{la}, \mathcal{V}) \rangle_{\sigma_p} \rangle$ is sent to $\mathcal{C}$, $\mathcal{R}_b^a$, and $\mathcal{R}_b^p$. As argued in NefSBFT, the passive $\mathcal{R}_b^p$ does not execute the agreement execution. $\mathcal{R}_b^p$ then only performs the receiving and verification of this message as well as updates $(\mathcal{C}n_{la}, \mathcal{V})$. The counter value is updated by invoking the $\mathcal{TE}_b^p$.*Coun_update*$(\mathcal{S}e_1, \langle h_{e1}, (\mathcal{C}n_{la}, \mathcal{V}) \rangle_{\sigma_p})$ and the $\mathcal{TE}_b^p$.*Coun_update*$(\mathcal{S}e_2, \langle h_{e2}, (\mathcal{C}n_{la}+1, \mathcal{V}) \rangle_{\sigma_p})$ (for block verification only) according to **Algorithm.** 3.

Algorithm 3 *Coun_update*$(\mathcal{S}e_1, \langle h_{e1}, (\mathcal{C}n_{la}, \mathcal{V}) \rangle_{\sigma_p})$

Input: *Reply* message
Output: $\mathcal{C}n_{la}$
while holding $\langle H(M), (\mathcal{C}n_{la}, \mathcal{V}) \rangle_{\sigma_p}$ **do**
  if Vrfy$(\langle h_{e1}, (\mathcal{C}n, \mathcal{V}) \rangle_{\sigma_p} = 0)$ **then**
    return *"invalid signature"*;
  end
  else if $\mathcal{C}n \neq \mathcal{C}n_{la}$ **then**
    return *"invalid counter"*;
  end
  else if $H(\mathcal{S}e_1) \neq h_{e1}$ **then**
    return *"invalid secret"*;
  end
  else
    return $"\mathcal{C}n_{la} = \mathcal{C}n_{la} + 1"$.
  end
end

Validity of this *Reply* message is performed in the following steps by $\mathcal{C}$ to confirm the processing results.

- Given $\langle h_e, (\mathcal{C}n, \mathcal{V}) \rangle_{\sigma_p}$, $\mathcal{C}$ checks the validity of this signature to determine whether $\mathcal{S}e$ is bound to $\mathcal{C}n$-denoted message. $\mathcal{TE}$ in NefSBFT maintains a monotonic and unique $\mathcal{C}n$ to remark the message. Each secret can be bound to the $\mathcal{C}n$-denoted message, but can not be bound to another message to ensure $\langle h_e, (\mathcal{C}n, \mathcal{V}) \rangle_{\sigma_p} = 1$.
- $\mathcal{C}$ performs the validity verification on $\langle H(M), (\mathcal{C}n, \mathcal{V}) \rangle_{\sigma_p}$ such that each $\mathcal{C}n$ uniquely defines one request.
- The correctness of $\mathcal{S}e_1$, that is $H(\mathcal{S}e_1) = h_{e1}$, plays as the committing over the message agreement, with the counter $\mathcal{C}n_{la}$ as well as among at least $f+1$ active replicas.
- As is $\mathcal{S}e_1$, $\mathcal{C}$ checks $\mathcal{S}e_2$ when it receives the *Reply* message of $\mathcal{B}_i$. $H(\mathcal{S}e_2) = h_{e2}$ further ensures that at least $f+1$ active replicas yield consensus on $\mathcal{B}_i$ with the $proof$. Thus, $\mathcal{B}_i$ is valid through the NefSBFT protocol.

For each destined $\mathcal{R}_b$ of the $\mathcal{T}_k||\mathcal{B}_i$ message, the ordered transactions and agreed block state have been synchronized locally at this point through our NefSBFT protocol. Then, replicas can start the next slot depending on their current view $(\mathcal{C}n_{la}^T, \mathcal{V})$, or they will start the next epochB based on $(\mathcal{C}n_{la}^B, \mathcal{V})$. Since more serious attacks may be launched by non-primary replicas, we explain the practical solution and analysis in the next Section.

5 Security and Performance Analysis

5.1 NefSBFT Security Properties

Standard securities of PBFT protocols relate to the primary and the backup replicas, which are synoptically defined as the security properties of counter reliability, secret uniqueness and DDOS guarding in NefSBFT. Recall that the security of malicious/crashed $\mathcal{R}_b^a$s except for the DDOS attack is critically considered in the scheme designing.

Counter Reliability (CR), with parameters $\beta \in [1, +\infty], s, e \in \mathbb{N}^+$. Consider the $\mathcal{C}n$ maintained in $\mathcal{TE}_p$ at onset of the slot sl or epochB eb. Let sl_1, sl_2 or eb_1, eb_2 be two previous slots or epochBs for which $sl_1 + s \leq sl_2 \leq sl$ or $eb_1 + e \leq eb_2 \leq eb$, so sl_1 or eb_1 is at least s slots or e epochBs ahead of sl_2 or eb_2. Then for $|\mathcal{T}[sl_1 : sl_2]| \geq \beta s$, $|\mathcal{B}[eb_1 :$

$eb_2]| \geq \beta e$, there is $\mathcal{C}n > \mathcal{C}n' \in [sl_1 : sl_2] \cup [eb_1 : eb_2]$, and TEEs of all honest replicas have the same $\mathcal{C}n$ of sl or eb.

Secret Uniqueness (SU), with parameter $u \in \mathbb{N}^+$. The secrets $\mathcal{S}e_{2u}, \mathcal{S}e_{2u+2}$ generated by two $\mathcal{TE}_p$s at onset of the slots $sl_u < sl_{u+1}$ or epochBs $eb_u < eb_{u+1}$ are such that $\mathcal{S}e_{2u} \neq \mathcal{S}e_{2u+1} \neq \mathcal{S}e_{2u+2} \neq \mathcal{S}e_{2u+3}$. Besides, $\mathcal{R}_p$ can change nothing about the original $\mathcal{S}e_{2u}, \cdots, \mathcal{S}e_{2u+3}$.

DDOS Guarding (DG), with parameters $\gamma, \theta \in (0, 1]$ and $Th_{ac} \in \mathbb{N}^+$. Assume that $Th_{ac} \in \mathbb{N}^+$ is the threshold of accepted non-primary ($\mathcal{R}_b$) failures. At each sl or eb, $\mathcal{R}_p$'s guarding on DDOS attack is at least $1-\gamma$, and the available substitution $\mathcal{R}_b^p$'s guarding on DDOS attack is at least $1-\theta$. We call γ and θ the influence of DDOS attack.

Note that typically these security properties for blockchain PBFT protocols are formulated so that they grant the above-described security guarantees to all *honest* replicas $\mathcal{R}_p$ and R_bs. In our more fine-grained designing, a natural choice is to analyze these properties for the primary $\mathcal{R}_p$, active $\mathcal{R}_b^a$s and passive $\mathcal{R}_b^p$s all.

5.2 Security Proofs of NefSBFT

Our first goal is to establish that the useful properties of counter reliability, secret uniqueness, and DDOS guarding are achieved by NefSBFT. Namely, we start by assuming that all participating nodes of NefSBFT are deployed of $\mathcal{TE}$ in the network $\mathcal{N}et_{ust}^{\Delta}$. We refer to this setting as the *setting with $\mathcal{TE}$-deployed $\mathcal{N}et_{ust}^{\Delta}$*. The rest of this section is dedicated to the proofs.

Theorem 1. Consider the execution of NefSBFT with adversary $\mathcal{A}_{bp}$ of $\mathcal{R}_p$ and $\mathcal{R}_b$s and in the *setting with $\mathcal{TE}$-deployed $\mathcal{N}et_{ust}^{\Delta}$*. NefSBFT achieves the guarantee of counter reliability as follows:

- **CR.** The counter reliability that our NefSBFT applies with parameter s, e and $v_1 \geq 1, v_2 \geq 1$ is at least

$$\begin{aligned} \mathcal{C}n_{la}^T(s) &= \beta s \cdot \mathcal{C}n_{la-s}^T + v_1; \\ \mathcal{C}n_{la}^B(e) &= \beta e \cdot \mathcal{C}n_{la-e}^B + v_2. \end{aligned} \tag{9}$$

Besides, at least all $2f+1$ honest replicas agree on $\mathcal{C}n_{la}$.

Proof: We show that when designing each $\mathcal{TE}$ with the TA of counter service, the overall maintenance of this TA is secure. There requires the current value determination of $\mathcal{C}n_{la}$ and its management including storage and attestation. To see this, consider the function $\mathcal{C}n_{la}(se) = \beta se \cdot \mathcal{C}n_{la-s} + v, (se = s\,or\,e)$ as the determination rule to denote the monotonically growing $\mathcal{C}n$ when s or e rounds of NefSBFT have flowed ahead of current sl or eb with $\mathcal{C}n_{la}$.

For $\mathcal{TE}_p$, it compares the new request in NefSBFT to determine the current $\mathcal{C}n_{la}$. Concretely, in this case $\mathcal{TE}_p$ always prefers the $\mathcal{C}n_{la}$ using $\mathcal{C}n_{la}(1) = \beta \cdot 1 \cdot \mathcal{C}n_{la-1} + v$. Regarding $\mathcal{C}n_{la-1} \prec\prec \mathcal{C}n_{la}$, where $\prec\prec$ denotes that $\mathcal{C}n_{la-1}$ is a counter ahead of $\mathcal{C}n_{la}$, $\mathcal{TE}_p$ will yield $\mathcal{C}n_{la} = \beta \cdot \mathcal{C}n_{la-1} + v \geq \mathcal{C}n_{la-1} + 1$ to define the new request. We argue that $\mathcal{TE}_p$ favors each new request (the new request at the start of a different round of NefSBFT or the new processed request after *Commit1*) of the monotonic and unique $\mathcal{C}n$ since the malicious $\mathcal{R}_p$ will be substituted through *View change*. That is, $\mathcal{TE}_p$ will determine the $\mathcal{C}n_{la}$ and $\mathcal{C}n_{la}+1$ at each sl or eb throughout the whole replicas' network.

Recall that the metric of TEE, for current $\mathcal{C}n_{la}$ at $\mathcal{TE}_p$, any $\mathcal{TE}_b$ maintains the acknowledgement by remote attestation for this value. For the attested $\mathcal{C}n_{la}$, there are the events. (1) Each $\mathcal{TE}_b$ of $\mathcal{R}_b$ further validates $\mathcal{C}n_{la}$ as follows by comparing with its local $\mathcal{C}n$:

$$\mathcal{C}n_{la}(s)^{-1} = \mathcal{C}n_{la-s} = \frac{\mathcal{C}n_{la} - v}{\beta se}. \tag{10}$$

$\mathcal{TE}_b^j$ locally affirms $\mathcal{C}n_{la}$ if $\mathcal{C}n_{la-s} = \mathcal{C}n_j$. (2) $\mathcal{TE}_b^j$ updates $\mathcal{C}n_j$ once it receives new request from $\mathcal{R}_p$. (3) if $\mathcal{R}_b^{a(j)}$ is honest and wants to synchronize the system state, it honestly performs the left operations and updates $\mathcal{C}n_{a(j)}$. Otherwise, it deliberately prevents the new request from being successfully processed. While the TEE-deployed $\mathcal{R}_b^{a(j)}$ can not be Byzantine by being substituted of the $\mathcal{R}_b^p$, the request can be finished and $\mathcal{C}n_{la}$ can be reliably updated among at least all $2f+1$ honest replicas.

□

Theorem 2. Consider the execution of NefSBFT with adversary $\mathcal{A}_p$ of $\mathcal{R}_p$ and in the *setting with $\mathcal{TE}$-deployed $\mathcal{N}et_{ust}^{\Delta}$*. NefSBFT achieves the guarantee of secret uniqueness as follows:

- **SU.** The secret uniqueness that our NefSBFT applies with parameter u is as follows:

$$\begin{aligned} &\mathcal{C}n_{la-2}||M_1||\mathcal{S}e_{2u} \neq \mathcal{C}n_{la-1}||\hat{M}_1||\mathcal{S}e_{2u+1} \\ &\neq \mathcal{C}n_{la}||M_2||\mathcal{S}e_{2u+2} \neq \mathcal{C}n_{la+1}||\hat{M}_2||\mathcal{S}e_{2u+3}. \end{aligned} \tag{11}$$

$\mathcal{C}n_{la}||M$ in $\mathcal{C}n_{la}||M||\mathcal{S}e_{2u+2}$ denotes the $\mathcal{C}n_{la}$-bound message M. $\mathcal{R}_p$ also does not know $\mathcal{S}e$s before *Commit* and *Reply*, later, $\mathcal{S}e$s can not be changed either.

Proof: Let sl_{u+1} or eb_{u+1} be the current round of NefSBFT and $\mathcal{C}n_{la}$ be the current counter. Recall that by the proof of the counter reliability, each counter is monotonic, unique, and secure. Thus, we see that at sl_{u+1} or eb_{u+1}, the new request and handled request can be uniquely bound to $\mathcal{C}n_{la}$ and $\mathcal{C}n_{la+1}$, respectively. For each $\mathcal{S}e$, there are the following events.

(1) Once receiving $\mathcal{T}_k||\mathcal{B}_i$, $\mathcal{R}_p$ invokes $\mathcal{TE}_p$ to use the TA of secret-sharing over $\mathcal{T}_k||\mathcal{B}_i$, that is, $\mathcal{TE}_p$ securely generates $\mathcal{S}e_{2u+2}$ and $\mathcal{S}e_{2u+3}$.

(2) Then, $\mathcal{TE}_p$ bounds each secret to the new $\mathcal{T}_k||\mathcal{B}_i$ and processed $\mathcal{T}_k||\mathcal{B}_i$ by $\langle H(\mathcal{S}e_{2u+2}), (\mathcal{C}n_{la}, \mathcal{V})\rangle_{\sigma_p}$ and $\langle H(\mathcal{S}e_{2u+3}), (\mathcal{C}n_{la}+1, \mathcal{V})\rangle_{\sigma_p}$, respectively. Observed that the secrets $\mathcal{S}e_{2u+2}$ and $\mathcal{S}e_{2u+3}$ associated with the new $\mathcal{T}_k||\mathcal{B}_i$ and handled $\mathcal{T}_k||\mathcal{B}_i$ are then precisely appearing as:

$$\begin{aligned} \langle H(\mathcal{S}e_{2u+2}), (\mathcal{C}n_{la}, \mathcal{V})\rangle_{\sigma_p} &= \mathcal{C}n_{la}||\mathcal{T}_k||\mathcal{B}_i||\mathcal{S}e_{2u+2}; \\ \langle H(\mathcal{S}e_{2u+3}), (\mathcal{C}n_{la+1}, \mathcal{V})\rangle_{\sigma_p} &= \mathcal{C}n_{la+1}||\hat{\mathcal{T}_k}||\hat{\mathcal{B}_i}||\mathcal{S}e_{2u+3}. \end{aligned} \tag{12}$$

Then, based on the **Theorem 1** we have

$$\begin{aligned} &\mathcal{C}n_{la-2}||\mathcal{T}_k{}'||\mathcal{B}_i{}'||\mathcal{S}e_{2u} \neq \mathcal{C}n_{la-1}||\hat{\mathcal{T}_k}{}'||\hat{\mathcal{B}_i}{}'||\mathcal{S}e_{2u+1} \\ &\neq \mathcal{C}n_{la}||\mathcal{T}_k||\mathcal{B}_i||\mathcal{S}e_{2u+2} \neq \mathcal{C}n_{la+1}||\hat{\mathcal{T}_k}||\hat{\mathcal{B}_i}||\mathcal{S}e_{2u+3}. \end{aligned} \tag{13}$$

(3) Note that by $\langle H(\mathcal{S}e), (\mathcal{C}n, \mathcal{V})\rangle_{\sigma_p}$, $\mathcal{S}e$ is indeed unchangeable before $\mathcal{R}_p$ obtains it during *Commit* and *Reply*. Moreover, since the secret has been opened and each $\mathcal{R}_b^a$ has committed to the request and the processed request, $\mathcal{R}_p$ that

maliciously denies the truth of secret shares and constructs the false $\mathcal{S}e$ to slander $\mathcal{R}_b^a$ will be overruled by $\mathcal{T}E_p$.

□

Theorem 3. Consider the execution of NefSBFT with adversary $\mathcal{A}_b$ of $\mathcal{R}_b$. $\mathcal{A}_b$ initiates DDOS attack by one-by-one opening of wrong $\mathcal{S}e^j$ during *Commit 1* and *Reply1*. Th_{ac} is set in designing NefSBFT in order to limit the consumption from non-primary DDOS attacks. Otherwise, at most f times of failures for secret share revealing can be launched. Note that NefSBFT is designed to transit to PBFT execution when Th_{ac} is attained. Thus, the lower value is set of Th_{ac} to avoid the intense DDOS attack in the sacrifice of performance and vice versa.

For the primary replica $\mathcal{R}_p$ which verifies the secret shares and $2f$ passive replicas $\mathcal{R}_b^p$s as the substitutions for several times, NefSBFT achieves the guarantee of DDOS guarding as follows:

- **DG.** The DDOS guarding that our NefSBFT applies with parameters $\gamma, \theta, Th_{ac} \leq f$ is

$$\begin{aligned}\lim_{Th_{ac}\to 0} \mathcal{DG}_{R_p} &= \lim_{Th_{ac}\to 0} 1-\gamma \leq \lim_{Th_{ac}\to 0} 1-\frac{Th_{ac}}{|\mathcal{R}|} = 1;\\ \lim_{Th_{ac}\to 0} \mathcal{DG}_{R_b^p} &= \lim_{Th_{ac}\to 0} 1-\theta \leq \lim_{Th_{ac}\to 0} 1-\frac{Th_{ac}}{2f} = 1.\end{aligned} \tag{14}$$

  Where $\mathcal{DG}_{R_p}$ and $\mathcal{DG}_{R_b^p}$ denote the $\mathcal{R}_p$'s and $\mathcal{R}_b^p$'s guarding against DDOS attack, respectively.

Proof: The proof is straightforward. Since at most Th_{ac} active $\mathcal{R}_b^a$s initiate the DDOS attack during *Commit1* and *Reply1*, $\mathcal{R}_p$ performs Th_{ac} times of verification and substitutions of passive $\mathcal{R}_b^p$s. Note that there are $|\mathcal{R}| = 3f+1$ replicas in total and $2f$ passive $\mathcal{R}_b^p$s during NefSBFT execution, we therefore have the largest DDOS's influences separately on $\mathcal{R}_p$ and $\mathcal{R}_b^p$ as follows:

$$\gamma = \frac{Th_{ac}}{|R|}; \theta = \frac{Th_{ac}}{2f}. \tag{15}$$

Furthermore, we explain the benefits of $\mathcal{A}_b$ from DDOS attack. To initiate the DDOS attack, $\mathcal{A}_b$ has to sacrifice its network and trust resources. Assume that the consumption of network resources and trust from the honest replicas during one time of opening the wrong secret share are cs, we therefore have benefits $\mathcal{B}e$:

$$\mathcal{B}e = \gamma + \theta - Th_{ac} \cdot cs = Th_{ac}(\frac{1}{|R|} + \frac{1}{2f} - cs). \tag{16}$$

We roughly use γ and θ to denote the resource consumption when $\mathcal{R}_p$ performs verification and substitutions.

□

5.3 Performance Analysis

To complete the argument, we observe the performance of NefSBFT protocol. Specifically, considering the prominent performance advantages, we capture the analysis of average hops for any message to be broadcasted and the message complexity during *Commit1* and *Reply1*.

5.3.1 Average Travel Hops

We remark that our PogaSoC technique allows efficient multicasting regarding a few travel hops as analyzed in this section. First, we need to define some relevant quality.

Definition 1. **(Non-delivery probability.)** For the encounters $n_{e(WT)}$ that are met by a hop during $\mathcal{WT}$, let Pd_0 denote the delivery probability with respect to the condition $Pd(n_{e(WT)}, \mathcal{D}) < \mathcal{T}h$. In other words, none of $n_{e(WT)}$ are qualified to be the next hop to the receiver.

Definition 2. **(Average-delivery probability.)** At any time $t < \mathcal{WT}$, we let Pd_i be the average delivery probability of node $\mathcal{N}_i$ to the destination after the former hop.

Definition 3. **(Relayed-delivery probability.)** We say that the former hop has relayed the message M to the next hop before $\mathcal{WT}$ expires. The new delivery probability aims to the destination is defined by Pd_g.

Definition 4. **(Expired-delivery probability.)** We call Pd_w the new delivery probability towards the receiver, if M has been advertised at the expiration of $\mathcal{WT}$.

The heart of our argument for average hops taking to $\mathcal{D}$ is captured in the following theorem.

Theorem 4. In the *setting with $\mathcal{T}E$-deployed* $\mathcal{N}et_{ust}^{\Delta}$, any message packet M will be multicasted to $\mathcal{D}$ by wasting the number of travel hops as follows:

$$\mathcal{H}_{Aev} = \frac{1}{(1-Pd_0) * Pd_g + Pd_0 * Pd_w}. \tag{17}$$

Proof: We assume that each $\mathcal{N}_i$ moves at the average speed of $\mathcal{SP}_{Ave}$ in network $\mathcal{N}et_{ust}^{\Delta}$ with the density $\mathcal{DS}$ of nodes. Moreover, let $\mathcal{TR}$ be the transmission range of each node. The argument can then be inductively applied to any $\mathcal{N}_i$ and M requested later.

Consider the transmission of M to the destination $\mathcal{D}$, and let $n_{e(WT)}$ be the encountered nodes at any time $t \leq \mathcal{WT}$. Then, in view of possible encountering, we have $n_{e(WT)}$ as:

$$n_{e(WT)} = 2 * \mathcal{TR} * \mathcal{SP}_{Ave} * \mathcal{WT} * \mathcal{DS}. \tag{18}$$

Additionally, let $P(\mathcal{H}o_{Aev} > n)$ be the probability that it takes more than n hops to reach $\mathcal{D}$, then it meets:

$$P(\mathcal{H}o_{Aev} > n) = \Pi_{i=1}^{n}(1-Pd_i). \tag{19}$$

As Pd_0, Pd_g, and Pd_w have defined all other conditions of delivery predictability, we can drive:

$$Pd_i = (1-Pd_0) * Pd_g + Pd_0 * Pd_w \tag{20}$$

Depending on the independent property among each hop, then $P(\mathcal{H}o_{Aev} > n)$ can be simply derived as :

$$\begin{aligned}P(\mathcal{H}o_{Aev} > n) &= \Pi_{i=1}^{n}(1-Pd_i)\\ &= (1-Pd_1)\cdots(1-Pd_n)\\ &= (1-((1-Pd_0) * Pd_g + Pd_0 * Pd_w))^n.\end{aligned} \tag{21}$$

As invoking in Eq. 21, $P(\mathcal{H}o_{Aev} > n) = (1-P)^n (P = Pd_i)$, we argue that $\mathcal{H}o_{Aev}$ meets the geometrical distribution. This means that its expectation can be calculated as:

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2021.3104465, IEEE Transactions on Dependable and Secure Computing

$$E(\mathcal{H}o_{Aev}) = \mathcal{H}o_{Aev} = \frac{1}{Pd_i} = \frac{1}{(1-Pd_0)*Pd_g + Pd_0 * Pd_w}. \quad (22)$$

□

5.3.2 Message Complexity

The analysis is generalized in this section to fully analyze the message complexity of NefSBFT by showing that this complexity grows with replicas by $\mathcal{G}^{ex}_{PLY}$. The desired statement for the main message complexity is given in **Theorem 5**. The main message complexities of *Commit1* and *Reply1* in FastBFT are the same.

Theorem 5. Consider the protocol NefSBFT using the PogaSoC technique and secret-sharing as described in Section 4 in the *setting with $\mathcal{TE}$-deployed $\mathcal{N}et^{\Delta}_{ust}$*. If $\mathcal{R}_p$ multicasts the message M to all $R_b^{a(j)}(j = 1, \cdots, f+1)$ and $R_b^{p(j)}(j = 1, \cdots, 2f)$, and $\mathcal{H}o_{Aev}$ travel hops are taken for each message, then the message complexity consumed during each phase in NefSBFT is shown as follows:

$$\begin{aligned} &Cle_{ss} = Cle_{pre} = Cle_{cm2} = (f+1) \cdot \mathcal{H}o_{Aev}; \\ &Cle_{cm1} = Cle_{rpy1} = f+1; \\ &Cle_{rpy} = (3f+2) \cdot \mathcal{H}o_{Aev}; \\ &Cle_{rpy2} = (2f+1) \cdot \mathcal{H}o_{Aev}. \end{aligned} \quad (23)$$

Proof: We have to analyze seven different phases executed for $\mathcal{B}_i$ and $\mathcal{T}_k$ which are classified into four cases here, depending on which message in NefSBFT is handled.

(1) *$\mathcal{R}_p$ multicasts each message packet to $R_b^{a(j)}(j = 1, \cdots, f+1)$*. This condition is revoked during *Secret-sharing*, *Prepare*, and *Commit2*. This means that $\mathcal{R}_p$ separately multicasts the message packets in terms of secret shares, request, and processed request to all active replicas. Since there are $f+1$ active replicas and average $\mathcal{H}o_{Aev}$ hops are required for each message to be multicasted to one receiver, we have $Cle_{ss} = Cle_{pre} = Cle_{cm2} = (f+1) \cdot \mathcal{H}o_{Aev}$.

(2) *$R_b^{a(j)}(j = 1, \cdots, f+1)$ opens secret share to $\mathcal{R}_p$*. This condition is revoked during *Commit1* and *Reply1*. This means that $f+1$ active replicas confirm their committing by revealing the secret shares to $\mathcal{R}_p$. Then, there is $Cle_{cm1} = Cle_{rpy1} = f+1$.

(3) *$\mathcal{R}_p$ replies to $\mathcal{C}$, R_b^a and R_b^b*. This condition is revoked during the final *Reply* of $\mathcal{T}_k$. $\mathcal{R}_p$ replies with the verified $\mathcal{T}_k$ and revealed secret to confirm the result with all replicas and $\mathcal{C}$. Hence, we can derive $Cle_{rpy} = (3f+2) \cdot \mathcal{H}o_{Aev}$.

(4) *$\mathcal{R}_p$ replies to $\mathcal{C}$ and $R_b^{b(j)}(j = 1, \cdots, 2f)$*. This condition is revoked during the final *Reply2* of $\mathcal{B}_i$. An earlier agreement execution has synchronized the valid and processed $\mathcal{B}_i$ among $\mathcal{R}_p$ and R_b^a, thus, $\mathcal{R}_p$ only sends the verified $\mathcal{B}_i$ and revealed secret to $\mathcal{C}$ and passive $R_b^{b(j)}$s. Then, we can easily conclude $Cle_{rpy2} = (2f+1) \cdot \mathcal{H}o_{Aev}$.

□

6 Performance Evaluation

In this section, we evaluate the performance of NefSBFT for transactions ordering and block verification by leveraging various PogaSoC parameters and NefSBFT instantiations. Furthermore, PogaSoC is compared with the frequently-used propagation techniques Erlay and Flooding in Bitcoin. Another comparison is set among NefSBT-based blockchain systems with two simulated systems (one uses the FastBFT) focusing on NefSBFT's improvement over FastBFT and impact on the whole system performance. To this end, we construct a blockchain simulator. This allows us to apply our NefSBFT protocol into the blockchain simulator in a unified framework. Specifically, we measure and compare the average hops, delivery ratio, and propagation delays of PogaSoC, and latency and consistency level of NefSBFT.

6.1 Blockchain Simulator and Experimental Setup

TABLE 2: Blockchian simulator parameters

Network parameter	Description
Transaction size	Average size (0.512KB)
Blockchain nodes $\mathcal{N}$	block-producing, light, full nodes
Message broadcast	Transactions and blocks
IP and Port distribution	Possible node discovery
Block size	Average size (1024KB)
Consensus parameter	Description
Computing power	PoW power
Stakes holding	PoS stakes
Block producing rule	PoW/PoS
Resource parameter	Description
Storage space distribution	Average transaction catch

Blockchain simulator. In TABLE 2, we summarize parameters that we capture in our simulator. Our simulator mimics the P2P network layer by separately implementing PogaSoC, Erlay, and Flooding. The use of both propagation techniques and consensus protocols drive the whole system's execution. Then, the simulated systems with integrating consensus protocols are NefSBFT plus PoW, Flooding plus PoW (e.g., Bitcoin), NefSBFT plus PoS (e.g., PoS in Cardano [47]), and FastBFT plus PoS.

Experimental setup. Java is used as the programming language executed on a computer with Intel(R) Core(TM) i5-4590 CPU @ 3.30GHZ, 8GB RAM. We evaluate NefSBFT when setting following related parameters and methods:

- Parameters. Clients initiate concurrent requests to the blockchain network of $3000 * 3000m^2$ with $\mathcal{N} = 1000$ distributed nodes. Each node has $1Gbps$ of bandwidth and is able to transmit up to $500m$. In addition, $\epsilon = 0.75, \alpha = 0.98, \tau = 0.25$ are used to update $Pd(i,j)$. Each secret in the *Secret-sharing* phase has the size of 128 bits.
- Methods. $SHA256$ as in PoW is used for hashing, 256-bit ECDSA as in Bitcoin for signatures, and 128-bit CMAC for MACs. To simulate intermittent connectivity and frequent partitions of nodes in $\mathcal{N}et^{\Delta}_{ust}$, we assume that the nodes move with the speed of $1 \sim 5m/s$.

Our findings show that NefSBFT, to a large extent, improves the performance of the whole blockchain system in terms of latency, consistency (implied by block confirmation), and throughput. Besides, PogaSoC outperforms Erlay and Flooding in terms of travel hops, propagation delays, and delivery ratio. For instance, PogaSoC saves up to half of the taken travel hops of the well-performed Erlay. NefSBFT reduces $\frac{1}{2} \sim \frac{1}{3}$ latency of whole blockchain system. Besides, NefSBFT is superior to other methods in the consistency level of the system, which reaches 10% promotion.

6.2 Travel Hops

With the objective to experimentally validate our design, we compare the number of hops taken by PogaSoC by varying the following factors.

$\mathcal{T}h$. Fig. 9a depicts the testified number of travel hops to propagate a message M to 50, 100, 150, and 200 receivers, respectively, where there are $\mathcal{T}h \in [0.2, 0.9]$ and $\mathcal{WT} = 2s$. Our results suggest that the number of travel hops decreases with the threshold of delivery predictability $\mathcal{T}h$ up to 0.9. In fact, a larger $\mathcal{T}h$ means that a node will not easily forward M to the next hop. For example, for 100 receivers, the number of travel hops is 124.22 for $\mathcal{T}h = 0.2$ and 95.02 for $\mathcal{T}h = 0.5$, respectively. Second, we clearly see that for the same $\mathcal{T}h$, a less number of receivers incurs less travel hops. While conforming with PogaSoC, the more the message receivers are, the more the saved hops are. For example, when $\mathcal{T}h = 0.5$, the number of travel hops is 37.76 for 50 receivers and 120.23 for 150 receivers. In the case of 50 receivers, travel hops only occupy up to $\frac{1}{30}$ of the total network nodes. Even though there are 200 receivers, the biggest proportion of travel hops is $\frac{1}{4}$.

$\mathcal{WT}$. Fig. 9b describes the variation of travel hops' number when propagating M to 50, 100, 150, and 200 receivers, respectively. Values of $\mathcal{WT}$ are varied as $0.5s \sim 4s$ and $\mathcal{T}h = 0.2$. We discover that the number of travel hops decreases with the wait timers $\mathcal{WT}$ up to $4s$ since the nodes have more time to choose the better hops to receivers with growing $\mathcal{WT}$. For example, for 150 receivers, the number of travel hops is 242.15 for $\mathcal{WT} = 1s$ and 123.65 for $\mathcal{WT} = 4s$. For the fixed $\mathcal{WT}$, we clearly see that the number of travel hops has the similar tendency and saving with varying the number of receivers as in Fig. 9a by fixing $\mathcal{T}h$. For example, when $\mathcal{WT} = 2s$, the number of travel hops is 81.23 for 50 receivers and 200.11 for 150 receivers. Besides, the biggest proportion of travel hops in the network is $\frac{1}{10}$ for 50 receivers.

Receivers' number. Fig. 9c describes the travel hops taken in instantiations of various receivers. To this end, we vary the receivers' number from 10 to 200. For each simulated NefSBFT instance, we separately test the number of travel hops when $(\mathcal{T}h = 0.2, \mathcal{WT} = 4s)$, $(\mathcal{T}h = 0.2, \mathcal{WT} = 3s)$, $(\mathcal{T}h = 0.7, \mathcal{WT} = 2s)$, and $(\mathcal{T}h = 0.5, \mathcal{WT} = 2s)$. Clearly, our results indicate that different parameters' configurations yield different number of hops. In particular, we observe that $(\mathcal{T}h = 0.2, \mathcal{WT} = 4s)$ lightly requires the hops compared to a less wait timer of $(\mathcal{T}h = 0.2, \mathcal{WT} = 3s)$ given the same $\mathcal{T}h$, and a less wait timer of $(\mathcal{T}h = 0.7, \mathcal{WT} = 2s)$ and $(\mathcal{T}h = 0.5, \mathcal{WT} = 2s)$ given the different $\mathcal{T}h$s. Thus, we know that $\mathcal{WT}$ has a higher weight than $\mathcal{T}h$ to determine the number of travel hops. For example, for 190 receivers, the number of travel hops is 115.89 for $(\mathcal{T}h = 0.2, \mathcal{WT} = 4s)$, 141.08 for $(\mathcal{T}h = 0.2, \mathcal{WT} = 3s)$, 137.07 for $(\mathcal{T}h = 0.7, \mathcal{WT} = 2s)$, and 160.38 of $(\mathcal{T}h = 0.5, \mathcal{WT} = 2s)$. In addition, our results suggest that the number of travel hops increases with the number of receivers up to 190, then it slightly grows. For all the conditions plotted, the biggest proportion of travel hops is $\frac{1}{5}$ of the whole network.

6.3 Delivery Ratio and Propagation Delay

Fig. 10 demonstrates the delivery ratio and propagation delays of PogaSoC. Specifically, we compare the delivery ratio among the respective simulated parameters $(\mathcal{T}h, \mathcal{WT})$ when varying the receivers' number. We do the comparisons between one-hop delay and the average delays among the simulated receivers when adjusting $\mathcal{T}h$ and $\mathcal{WT}$, respectively. The adopted payload size is $1MB$.

Delivery ratio. Fig. 10a shows the delivery ratio of PogaSoC with $(\mathcal{T}h = 0.3, \mathcal{WT} = 3s)$, $(\mathcal{T}h = 0.3, \mathcal{WT} = 1s)$, $(\mathcal{T}h = 0.6, \mathcal{WT} = 1s)$, and $(\mathcal{T}h = 0.6, \mathcal{WT} = 2s)$, respectively. The number of receivers is varied from 10 to 200. We observe that the delivery ratio fluctuates with the number of receivers. Besides, since the parameters are altered, the delivery ratio differs from each other. While as is expected, PogaSoC is good at delivery and the delivery ratio falls into the quite high range $0.93 \sim 0.98$. Furthermore, it shows that for the different values of $(\mathcal{T}h, \mathcal{WT})$, the delivery ratios are extremely close and high.

The delays of varying $\mathcal{T}h$. In Fig. 10b, the one-hop and the average propagation delays of PogaSoC are tested when there are separately 100 and 200 receivers. $\mathcal{T}h$ is adjusted from 0.2 to 0.9 and $\mathcal{WT} = 2s$. We discover that the one-hop delay of the same receivers has a gap with the average delays which is at most $7s$. Also, the delays are slightly affected by the number of receivers. For example, excessive one-hop delays are at most $0.1s$ and the average delays are $2.2s$ for 100 more receivers. Thus, more advantages are achieved with more receivers. In addition, delays gently increase with $\mathcal{T}h$ since growing $\mathcal{T}h$ will reduce the availability of hops. For example, for 200 receivers, the average delays for $\mathcal{T}h = 0.2$ are $6.582s$, while they are only $7.273s$ for $\mathcal{T}h = 0.9$. Besides, one round of PogaSoC takes at most $7.3s$.

The delays of varying $\mathcal{WT}$. Fig. 10c depicts the influence of $\mathcal{WT}$ on the one-hop and the average propagation delays. To this end, we vary $\mathcal{WT}$ from $0.5s$ to $4s$, besides, $\mathcal{T}h$ is set to 0.2. We note that the delays slightly increase with the growth of $\mathcal{WT}$. An increase of $\mathcal{WT}$ represents that a longer time can be waited to find the next hop, thus, more delays are caused. However, we see that the delays fluctuate slightly, and the longest of them are $5s$. Furthermore, different number of receivers yields slightly different one-hop and average delays, which have the same tendency as in Fig. 10b. For the same number of receivers, the average delays are at most $4.5s$ more than the one-hop delay. For example, for 100 receivers and $\mathcal{WT} = 0.5s$, the one-hop delay is $0.132s$ and the average delays are $2.980s$.

6.4 NefSBFT Latency

We study PogaSoC's impact on NefSBFT, and compare the NefSBFT latency of different phases and scenarios. Since NefSBFT contains the phases of *Secret-sharing*, *Prepare*, *Commit2*, *Reply*, and *Reply2* which are executed on PogaSoC, we provide the latency comparisons of *Secret-sharing*, *Commit1*, and *Reply1*. Based on the executing scenarios of NefSBFT, we test the overall latency of transaction ordering and block verification. Fig. 11 shows the resulting impact of PogaSoC on *Secret-sharing* and comparison results of latency.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2021.3104465, IEEE Transactions on Dependable and Secure Computing

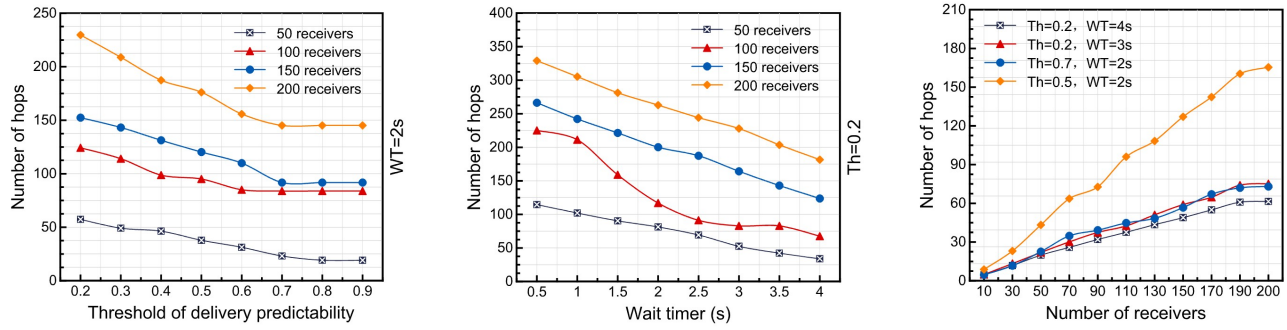


Fig. 9: The taken travel hops in PogaSoC.

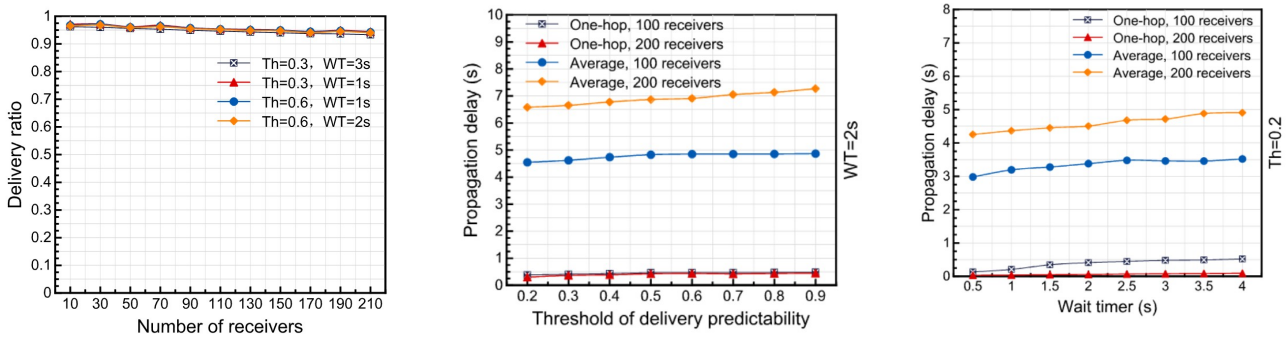


Fig. 10: Delivery ratio and propagation delays of PogaSoC.

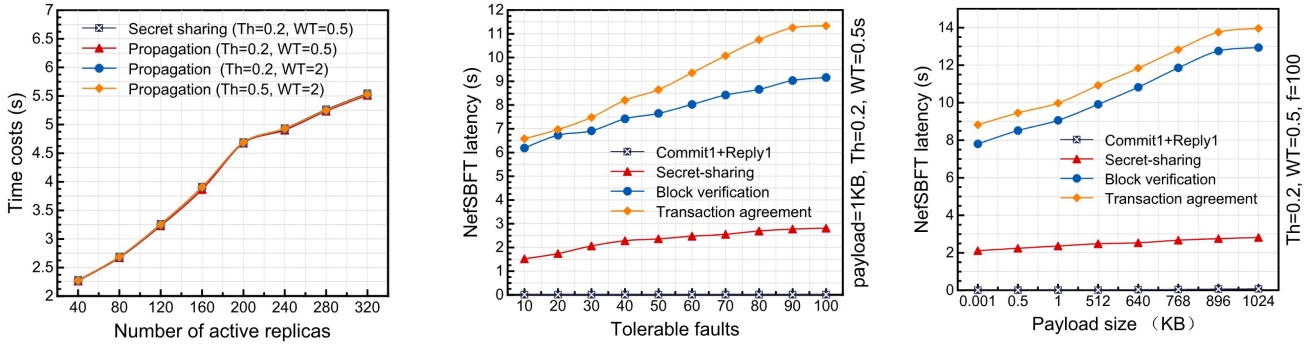


Fig. 11: Latency of NefSBFT.

PogaSoC's impact. Fig. 11a compares the time costs of secret-sharing and only secret share's propagation under different values of $(\mathcal{WT}, \mathcal{T}h)$. Payload size is $1MB$ and the number of active replicas varies from 40 to 320. The results show that it takes a little more time (up to $0.05s$) in secret-sharing than the propagation because of the secret generation. Thus, NefSBFT is efficient in *Secret-sharing*. Note that PogaSoC's greater impact on NefSBFT is in the sacrifice of time to provide a large resource saving and high delivery ratio. In addition, the time costs for secret-sharing increase with the growing number of active replicas, while the growth rate is very small. For example, time costs are $3.261s$ for 120 $\mathcal{R}_b^a$s and $5.545s$ for 320 $\mathcal{R}_b^a$s.

The latency of varying f. Fig. 11b compares the phases' and overall latency when the payload size is $1KB$ and there is $(\mathcal{WT} = 0.5s, \mathcal{T}h = 0.2)$. The NefSBFT instances are simulated by adjusting the number of tolerable faults as $f \in [10, 100]$. We discover that the latency increases with the growing f, while it presents a light growth starting from $f = 90$. Furthermore, we clearly see that the *Commit1* plus *Reply1* phases have less latency than the *Secret-sharing*, since no multicasting is involved. The transaction ordering causes more overall latency than block verification because there are $3f + 2$ receivers during *Reply*, while only $2f + 2$ during *Reply2*. Note that even the longest latency is below $12s$.

The latency of varying payload size. Fig. 11c compares the

phases' and overall latency with $f = 100$ and ($\mathcal{WT} = 0.5s, \mathcal{T}h = 0.2$). Payload size is varied from $1B$ to $1024KB$. The results indicate that, when the payload size is fixed, the latency distinction among *Secret-sharing*, *Commit1+Reply1*, transaction ordering, and block verification are the same as in Fig. 11b. When the payload size increases, the latency of each case grows. Even though there is a big change in the payload size, the latency slightly grows. For example, the largest latency is $14.604s$ for a transaction ordering of $1024KB$.

6.5 Comparison Results

We compare the performance between our design and other schemes in Fig. 12. Specifically, we analyze the performance of PogaSoC, Flooding, and Erlay in terms of the number of travel hops, propagation delays, and delivery ratio. Since Flooding and Erlay are used for transaction propagation, we execute on the transaction of $0.512KB$. Besides, the receivers' number is varied from 200 to 600.

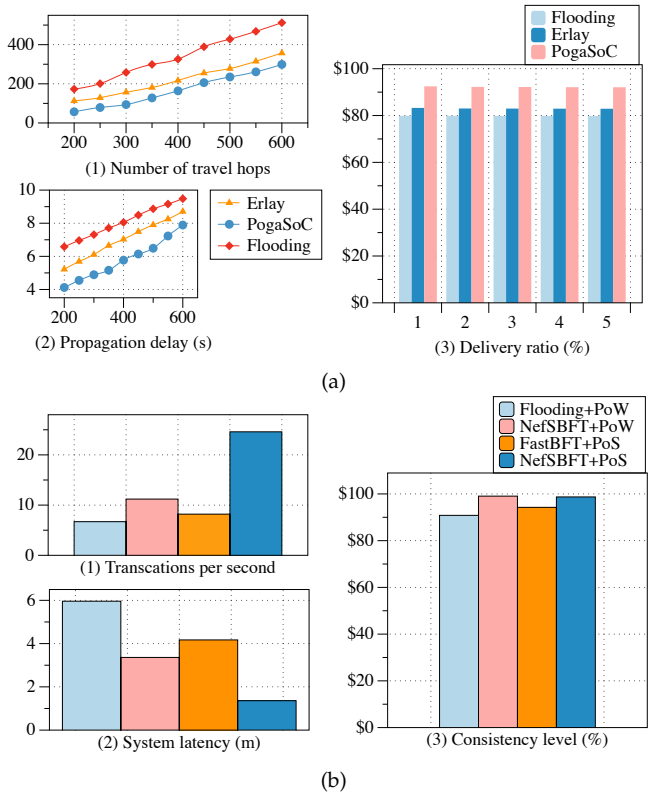


Fig. 12: Performance comparison: (a) PogaSoC vs. Erlay and Flooding, with varying receivers' number. (b) Simulated NefSBFT-based vs. Bitcoin and FastBFT-based systems.

To clarify NefSFT's improvement over FastBFT and impact on the whole system performance, we do comparisons among several simulated blockchain systems. In particular, 1) two systems (**Sym 1** and **Sym 2**) are simulated separately with Flooding plus PoW (e.g., Bitcoin system), and NefSBFT plus PoW; 2) another two systems (**Sym 3** and **Sym 4**) are simulated separately with FastBFT plus PoS, and NefSBFT plus PoS. The same consensus mechanisms are selected to capture the NefSBFT's influence. As the NefSBFT is prominent in efficiency and reliability, we execute several complete running of systems to test the average transaction throughput, system latency, and consistency level of block confirmation. Note that each block of $1MB$ has 2000 transactions and there are 600 participating nodes in NefSBFT.

Fig. 12a (1) shows the taken travel hops during the transaction propagation of PogaSoC, Flooding, and Erlay, respectively. We observe that the number of travel hops increases with the growing receivers' number. While comparing with Flooding and Erlay, our PogaSoC requires the least travel hops. In Fig. 12a (2), the propagation delays are tested. We discover that PogaSoC is the fastest in transaction propagation, and the smallest time difference among PogaSoC with Flooding and Erlay is $1s$. Fig. 12a (3) depicts the delivery ratio using different propagation schemes. To this end, we list several candidates of varying receivers' number. We note that although Erlay passes transactions to more receivers than Flooding, PogaSoC is clearly the superior technique which has more than 9% delivery ratio.

Fig. 12b (1) tests the NefSBFT's impact on the TPS (Transactions Per Second) of PoW-based (i.e., **Sym 1** and **Sym 2**) and PoS-based (i.e., **Sym 3** and **Sym 4**) blockchain systems, respectively. We discover that **Sym 2** system with NefSBFT is about 5 TPS more than **Sym 1**. This is because not only NefSBFT is more efficient but also NefSBFT provides reliable transactions input to PoW. Similarly, PoS-based **Sym 3** and **Sym 4** have the same tendency since NefSBFT uses PogaSoC to enlarge the delivery ratio of messages. The complete latency is demonstrated in Fig. 12b (2). To this end, we clarify that latency relationships among **Sym 1** with **Sym 2**, and **Sym 3** with **Sym 4** are just the opposite to TPS. For example, the NefSBFT-based **Sym 4** is at least 2 minutes less of latency than **Sym 3**. We study NefSBFT's impact on the system consistency level in Fig. 12b (3). The consistency level is defined as the accumulated computing power (in PoW-based systems) or stakes holding (in PoS-based systems) of a confirmed block. The results show that **Sym 2** is higher of consistency level than **Sym 1**. In fact, for the same PoW consensus mechanism, NefSBFT greatly improves the delivery ratio and consistency of messages, which leads to a more reliable PoW. On the listed result, **Sym 1** is 9% higher of consistency level. Although in **Sym 3**, FastBFT provides well consistency of messages, it is lower of delivery ratio than that guaranteed with PogaSoC. Therefore, NefSBFT-based **Sym 4** has more reliable input to PoS consensus mechanism. Note that the comparisons among **Sym 3** and **Sym 4** explicitly show NefSBFT's improvement over FastBFT.

7 Conclusion

In this paper, we propose NefSBFT, a social characteristic-based propagation-efficient protocol to broadcast transactions and blocks. NefSBFT provides full consideration on the message propagation in the PBFT-based public blockchain and objectively applies to the unstructured overlay networks. While the core is to design the propagation technique, NefSBFT achieves strong consistency and secure agreement by using the improved FastBFT in terms of tree-depending issue. This protocol gives considerable efficiency in terms of low latency and a high delivery ratio. Besides,

it is resource-efficient by saving travel hops and decreasing the message complexity from parabolic factors to the polynomial level. Our progaSoC technique enables us to compare the number of travel hops, delivery ratio, and propagation delays, so as to show the performance impact of our NefSBFT protocol on the blockchain system in terms of overall latency and consistency level. We specify the security provision of our protocol during the transaction ordering and block verification. Namely, our design promises strong security of counter reliability, secret uniqueness, and non-primary failures. The experimental results suggest that our NefSBFT offers system latency shortening of almost 3 minutes and consistency level promotion of 9%.

ACKNOWLEDGMENTS

This work was supported by the Key Program of NSFC (No. U1405255), the Shaanxi Science & Technology Coordination & Innovation Project (No. 2016TZC-G-6-3), the Fundamental Research Funds for the Central Universities (No. SA-ZD161504), the National Natural Science Foundation of China (No. 62072361), the Fundamental Research Funds for the Central Universities (No. JB211505), Guangxi Key Laboratory of Cryptography and Information Security (No. GCIS201917), Guangxi Key Laboratory of Trusted Software (No. KX202028), CCF-Tencent Open Fund WeBank Special Funding (CCF-Webank RAGR 20200102), CCF-NSFOCUS Kunpeng Research Fund (No. CCF-NSFOCUS 20200004), Henan Key Laboratory of Network Cryptography Technology (No. LNCT2020-A06), Hong Kong Scholar Program (No. XJ2019038). The work of Kim-Kwang Raymond Choo was supported only by the Cloud Technology Endowed Professorship.

REFERENCES

[1] Y. Liu, J. Liu, Z. Zhang, and H. Yu, "A fair selection protocol for committee-based permissionless blockchains," *Comput Security.*, vol. 91, p. 101718, 2020.

[2] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, "A secure sharding protocol for open blockchains," in *proc. 2016 ACM CCS, Vienna, Austria, October 24-28*, 2016, pp. 17–30.

[3] J. Shen, T. Zhou, D. He, Y. Zhang, X. Sun, and Y. Xiang, "Block design-based key agreement for group data sharing in cloud computing," *IEEE Trans. Dependable Secur. Comput.*, vol. 16, no. 6, pp. 996–1010, 2020.

[4] J. Kang, Z. Xiong, D. Niyato, P. Wang, D. Ye, and D. I. Kim, "Incentivizing consensus propagation in proof-of-stake based consortium blockchain networks," *IEEE Wirel. Commun. Lett.*, vol. 8, no. 1, pp. 157–160, 2019.

[5] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun, "On the security and performance of proof of work blockchains," in *proc. 2016 ACM CCS, Vienna, Austria, October 24-28*, 2016, pp. 3–16.

[6] J. A. Garay, A. Kiayias, and N. Leonardos, "The bitcoin backbone protocol: Analysis and applications," in *Proc. 34th EUROCRYPT, Sofia, Bulgaria, April 26-30*, ser. Lecture Notes in Computer Science, vol. 9057, 2015, pp. 281–310.

[7] B. Cao, Z. Zhang, D. Feng, S. Zhang, and Y. Li, "Performance analysis and comparison of pow, pos and dag based blockchains," *Digit. Commun. Netw.*, vol. 6, no. 4, pp. 480–485, 2020.

[8] H. Cho, H. Wu, C. Lai, T. K. Shih, and F. Tseng, "Intelligent charging path planning for iot network over blockchain-based edge architecture," *IEEE Internet Things J.*, vol. 8, no. 4, pp. 2379–2394, 2021.

[9] S. Distefano, A. D. Giacomo, and M. Mazzara, "Trustworthiness for transportation ecosystems: The blockchain vehicle information system," *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 4, pp. 2013–2022, 2021.

[10] S. Kim, "Impacts of mobility on performance of blockchain in vanet," *IEEE Access*, vol. 7, pp. 68 646–68 655, 2019.

[11] J. Wang and H. Wang, "Monoxide: Scale out blockchains with asynchronous consensus zones," in *proc. 16th NSDI, Boston, MA, February 26-28*, 2019, pp. 95–112.

[12] D. Larimer, "Delegated proof-of-stake (dpos)," *Bitshare whitepaper.*, 2014.

[13] J. Siim, "Proof-of-stake," in *Research Seminar in Cryptography*, 2017.

[14] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *proc. 37th CRYPTO, Santa Barbara, CA, USA, August 20-24*, 2017, pp. 357–388.

[15] Y. Sompolinsky, Y. Lewenberg, and A. Zohar, "SPECTRE: A fast and scalable cryptocurrency protocol," *IACR Cryptol. ePrint Arch.*, vol. 2016, p. 1159, 2016.

[16] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2019.

[17] I. Eyal, A. E. Gencer, E. G. Sirer, and R. van Renesse, "Bitcoin-ng: A scalable blockchain protocol," in *proc. 13th NSDI, Santa Clara, CA, USA, March 16-18*, 2016, pp. 45–59.

[18] R. Kapitza, J. Behl, C. Cachin, T. Distler, S. Kuhnle, S. V. Mohammadi, W. Schröder-Preikschat, and K. Stengel, "Cheapbft: resource-efficient byzantine fault tolerance," in *proc. 7th ACM EuroSys, Bern, Switzerland, April 10-13*, 2012, pp. 295–308.

[19] T. Distler, C. Cachin, and R. Kapitza, "Resource-efficient byzantine fault tolerance," *IEEE Trans. Comput.*, vol. 65, no. 9, pp. 2807–2819, 2016.

[20] 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, 2013.

[21] 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, 2019.

[22] G. S. Veronese, M. Correia, A. N. Bessani, and L. C. Lung, "EBAWA: efficient byzantine agreement for wide-area networks," in *proc. 12th IEEE HASE, San Jose, CA, USA, November 3-4*, 2010, pp. 10–19.

[23] E. Rohrer and F. Tschorsch, "Kadcast: A structured approach to broadcast in blockchain networks," in *proc. 1st ACM AFT, Zurich, Switzerland, October 21-23*, 2019, pp. 199–213.

[24] F. Y. Lin, C. Hsiao, Y. Wen, and Y. Su, "Adaptive broadcast routing assignment algorithm for blockchain synchronization services," in *proc. 10th ICUFN, Prague, Czech Republic, July 3-6*, 2018, pp. 487–492.

[25] G. Naumenko, G. Maxwell, P. Wuille, A. Fedorova, and I. Beschastnikh, "Erlay: Efficient transaction relay for bitcoin," in *proc. 2019 ACM CCS, London, UK, November 11-15*, 2019, pp. 817–831.

[26] H. Zhang, C. Feng, and X. Wang, "A greedy-based approach of fast transaction broadcasting in bitcoin networks," in *proc. ACM TUR-C, Chengdu, China, May 17-19*, 2019, pp. 29:1–29:5.

[27] V. Šešum-Čavić, E. Kühn, and L. Fleischhacker, "Efficient search and lookup in unstructured p2p overlay networks inspired by swarm intelligence," *IEEE Trans. Emerg. Top. Comput. Intell.*, vol. 4, no. 3, pp. 351–368, 2020.

[28] F. S. Annexstein, K. A. Berman, and M. A. Jovanovic, "Broadcasting in unstructured peer-to-peer overlay networks," *Theor. Comput. Sci.*, vol. 355, no. 1, pp. 25–36, 2006.

[29] R. Nagayama, K. Shudo, and R. Banno, "Simulation of the bitcoin network considering compact block relay and internet improvements," *CoRR*, vol. abs/1912.05208, 2019.

[30] J. Wang and H. Wang, "A fuzzy decision based intelligent qos multicast routing algorithm," in *proc. 8-th CIS, Guangzhou, China, November 17-18*, 2012, pp. 150–153.

[31] M. Li, J. Weng, A. Yang, W. Lu, Y. Zhang, L. Hou, J. Liu, Y. Xiang, and R. H. Deng, "Crowdbc: A blockchain-based decentralized framework for crowdsourcing," *IEEE Trans. Parallel Distributed Syst.*, vol. 30, no. 6, pp. 1251–1266, 2019.

[32] N. Sun, J. Zhang, P. Rimba, S. Gao, L. Y. Zhang, and Y. Xiang, "Data-driven cybersecurity incident prediction: A survey," *IEEE Commun. Surv. Tutorials*, vol. 21, no. 2, pp. 1744–1772, 2019.

[33] M. Jin, X. Chen, and S. Lin, "Reducing the bandwidth of block propagation in bitcoin network with erasure coding," *IEEE Access*, vol. 7, pp. 175 606–175 613, 2019.

[34] W. Hao, J. Zeng, X. Dai, J. Xiao, Q.-S. Hua, H. Chen, K.-C. Li, and H. Jin, "Towards a trust-enhanced blockchain p2p topology for enabling fast and reliable broadcast," *IEEE Trans. Netw. Serv. Manag.*, vol. 17, no. 2, pp. 904–917, 2020.

[35] F. Y.-S. Lin, C.-H. Hsiao, Y.-F. Wen, and Y.-C. Su, "Adaptive broadcast routing assignment algorithm for blockchain synchronization services," in *Proc. ICUFN, Prague, Czech Republic, July 3-6*, 2018, pp. 487–492.

[36] C. Decker and R. Wattenhofer, "Information propagation in the bitcoin network," in *in proc. 13th P2P, Trento, Italy, September 9-11*, 2013, pp. 1–10.

[37] O. Ersoy, Z. Ren, Z. Erkin, and R. L. Lagendijk, "Transaction propagation on permissionless blockchains: Incentive and routing mechanisms," in *Proc. CVCBT, Zug, Switzerland, June 20-22*. IEEE, 2018, pp. 20–30.

[38] J. Kan, L. Zou, B. Liu, and X. Huang, "Boost blockchain broadcast propagation with tree routing," in *Proc. SmartBlock, Tokyo, Japan, December 10-12*, ser. Lecture Notes in Computer Science, vol. 11373, 2018, pp. 77–85.

[39] W. Bi, H. Yang, and M. Zheng, "An accelerated method for message propagation in blockchain networks," *CoRR*, vol. abs/1809.00455, 2018.

[40] M. Castro and B. Liskov, "Practical byzantine fault tolerance," in *proc. Third OSDI, New Orleans, Louisiana, USA, February 22-25*, 1999, pp. 173–186.

[41] L. Liu, O. Y. de Vel, Q. Han, J. Zhang, and Y. Xiang, "Detecting and preventing cyber insider threats: A survey," *IEEE Commun. Surv. Tutorials*, vol. 20, no. 2, pp. 1397–1417, 2018.

[42] S. Sekar and B. Latha, "Lightweight reliable and secure multicasting routing protocol based on cross-layer for MANET," *Concurr. Comput. Pract. Exp.*, vol. 32, no. 4, 2020.

[43] P. Jauernig, A. Sadeghi, and E. Stapf, "Trusted execution environments: Properties, applications, and challenges," *IEEE Secur. Priv.*, vol. 18, no. 2, pp. 56–60, 2020.

[44] L. P. Maddali, M. S. D. Thakur, R. Vigneswaran, M. A. Rajan, S. Kanchanapalli, and B. Das, "Veriblock: A novel blockchain framework based on verifiable computing and trusted execution environment," in *Proc. COMSNETS, Bengaluru, India, January 7-11*. IEEE, 2020, pp. 1–6.

[45] S. Andreina, J. Bohli, G. O. Karame, W. Li, and G. A. Marson, "Pots - A secure proof of tee-stake for permissionless blockchains," *IACR Cryptol. ePrint Arch.*, vol. 2018, p. 1135, 2018.

[46] S. Matetic, K. Wüst, M. Schneider, K. Kostiainen, G. Karame, and S. Capkun, "BITE: bitcoin lightweight client privacy using trusted execution," in *Proc. 28th USENIX, Santa Clara, CA, USA, August 14-16*, 2019, pp. 783–800.

[47] L. M. Bach, B. Mihaljevic, and M. Zagar, "Comparative analysis of blockchain consensus algorithms," in *Proc. 41st MIPRO, Opatija, Croatia, May 21-25*, 2018, pp. 1545–1550.

JIANFENG MA received the B.S. degree in mathematics from Shaanxi Normal University, Xi'an, China, in 1985, and the M.S. degree and Ph.D. degree in computer software and communications engineering from Xidian University, Xi'an, China, in 1988 and 1995, respectively. From 1999 to 2001, he was a Research Fellow with Nanyang Technological University of Singapore. He is currently a Professor and a Ph.D. Supervisor with the Department of Computer Science and Technology, Xidian University, Xi'an, China. He is also the Director of the Shaanxi Key Laboratory of Network and System security. His current research interests include information and network security, wireless and mobile computing systems, and computer networks.

YINBIN MIAO received the B.E. degree in telecommunication engineering from Jilin University, Changchun, China, in 2011, and Ph.D. degree in telecommunication engineering from Xidian university, Xi'an, China, in 2016. He is currently an associate professor with the School of Cyber Engineering, Xidian University. His research interests include information security and applied cryptography.

XIMENG LIU (S'13-M'16-SM'21) received the B.Sc. degree in electronic engineering from Xidian University, Xi'an, China, in 2010 and the Ph.D. degree in cryptography from Xidian University, China, in 2015. Now he is the full professor in the College of Computer and Big Data, Fuzhou University. He was a research fellow at the School of Information System, Singapore Management University, Singapore. He has published more than 250 papers on the topics of cloud security and big data security including papers in IEEE TC, IEEE TIFS, IEEE TDSC, IEEE TPDS, IEEE TKDE, IEEE TSC, IEEE IoT Journal, and so on. He awards "Minjiang Scholars" Distinguished Professor, "Qishan Scholars" in Fuzhou University, and ACM SIGSAC China Rising Star Award (2018). His research interests include cloud security, applied cryptography and big data security.

KIM-KWANG RAYMOND CHOO received the Ph.D. degree in information security from the Queensland University of Technology, Australia, in 2006. He currently holds the Cloud Technology Endowed Professorship at The University of Texas at San Antonio (UTSA). He is the founding co-editor-in-chief of ACM Distributed Ledger Technologies: Research & Practice (commencing June 2021), founding chair of IEEE Technology and Engineering Management Society's Technical Committee on Blockchain and Distributed Ledger Technologies. He is an ACM Distinguished Speaker and IEEE Computer Society Distinguished Visitor (2021-2023), and included in Web of Science's Highly Cited Researcher in the field of Cross-Field-2020. In 2015, he and his team won the Digital Forensics Research Challenge organized by Germany's University of Erlangen-Nuremberg. He is the recipient of the 2019 IEEE Technical Committee on Scalable Computing Award for Excellence in Scalable Computing (Middle Career Researcher), the 2018 UTSA College of Business Col. Jean Piccione and Lt. Col. Philip Piccione Endowed Research Award for Tenured Faculty.

XIAOQIN FENG received the B.S. degree in information and computing science from Xidian University, Xi'an, China, in 2016. She is currently studying for the Ph.D. degree in cyberspace security, Xidian University. Her research interests include blockchain, consensus mechanism, and blockchain applications.