Performance Evaluation 153 (2022) 102244

Contents lists available at ScienceDirect

Performance Evaluation

journal homepage: www.elsevier.com/locate/peva

Efficient and DoS-resistant Consensus for Permissioned Blockchains☆

Xusheng Chen[a], Shixiong Zhao[a], Ji Qi[a], Jianyu Jiang[a], Haoze Song[a], Cheng Wang[a,b], Tsz On Li[a], T-H. Hubert Chan[a], Fengwei Zhang[c], Xiapu Luo[d], Sen Wang[e], Gong Zhang[e], Heming Cui[a,*]

[a] The University of Hong Kong, Hong Kong
[b] Huawei Cloud, Huawei Technologies, China
[c] Southern University of Science and Technology (SUSTech), China
[d] The Hong Kong Polytechnic University, Hong Kong
[e] Theory Lab, 2012 Labs, Huawei Technologies, China

ARTICLE INFO

Article history:
Available online 18 October 2021

Keywords:
Consensus
Byzantine fault tolerance
Permissioned blockchain
DoS atack
Trusted execution environment

ABSTRACT

Existing permissioned blockchain systems designate a fixed and explicit group of committee nodes to run a consensus protocol that confirms the same sequence of blocks among all nodes. Unfortunately, when such a permissioned blockchain runs on a large scale on the Internet, these explicit committee nodes can be easily turned down by denial-of-service (DoS) or network partition attacks. Although recent studies proposed scalable BFT protocols that run on a larger number of committee nodes, these protocols' efficiency drops dramatically when only a small number of nodes are attacked.

In this paper, we propose a novel protocol named EGES that leverages hardware trusted execution environments (e.g., Intel SGX) to develop a new abstraction called "stealth committee", which effectively hides a committee into a large pool of fake committee nodes. EGES selects a different stealth committee for each block and confirms the same blocks among all nodes with overwhelming probability. Our evaluation shows that EGES is the first efficient permissioned blockchain's consensus protocol, which simultaneously satisfies two important metrics: (1) EGES can tolerate tough DoS and network partition attacks; and (2) EGES achieves comparable throughput and latency as existing fastest permissioned blockchains' consensus protocols. EGES's source code is available on http://github.com/hku-systems/eges.

1. Introduction

A blockchain is a distributed ledger recording transactions maintained by nodes running on a peer-to-peer (P2P) network. These nodes run a consensus protocol to ensure consistency: nodes *confirm* (i.e., agree on committing [1–3])

☆ We thank our shepherd, Zhenhua Liu, and all anonymous reviewers for their valuable comments. This project was funded by Huawei Innovation Research Program Flagship 2018, Huawei Theory Lab Flagship 2021, HKU-SCF R&D funding scheme, HK RGC GRF (17202318, 17207117, 17201220), HK RGC ECS (27200916), Research Grants Council of the Hong Kong Special Administrative Region, China (No. PolyU15222320), and a Croucher Innovation Award.

* Corresponding author.
E-mail addresses: xschen@cs.hku.hk (X. Chen), sxzhao@cs.hku.hk (S. Zhao), jqi@cs.hku.hk (J. Qi), jyjiang@cs.hku.hk (J. Jiang), hzsong@cs.hku.hk (H. Song), cwang2@cs.hku.hk (C. Wang), toli2@cs.hku.hk (T.O. Li), huber@cs.hku.hk (T.-H.H. Chan), zhangfw@sustech.edu.cn (F. Zhang), csxluo@comp.polyu.edu.hk (X. Luo), wangsen31@huawei.com (S. Wang), nicholas.zhang@huawei.com (G. Zhang), heming@cs.hku.hk (H. Cui).

https://doi.org/10.1016/j.peva.2021.102244
0166-5316/© 2021 Published by Elsevier B.V.

the same sequence of blocks (i.e., no forks). Each block contains the hash of its previous block, forming an immutable hash chain. A blockchain can be permissioned or permissionless. A typical permissionless blockchain does not manage membership for nodes and is usually equipped with a cryptocurrency mechanism (e.g., Bitcoin [4]) to incentivize nodes to follow the blockchain's protocol [5,6].

In contrast, a permissioned blockchain runs on a set of authenticated member nodes and can leverage the mature Byzantine Fault-Tolerant (BFT) protocols [7–9] to achieve better efficiency (i.e., throughput and latency). This paper focuses on permissioned blockchains because their decoupling from cryptocurrencies has facilitated the deployment of many general data-sharing applications, including a UK medical chain [10], IBM supply chains [11], and the Libra payment system [3].

For performance and regulation reasons (e.g., meeting the honesty threshold of BFT protocols [12]), a permissioned blockchain (e.g., Hyperledger Fabric [2]) typically runs its consensus protocol on a static and explicit committee. This static committee approach is already robust for a permissioned blockchain among a small scale of enterprises [11].

Unfortunately, as permissioned blockchains are deployed on large scales on the Internet, this static committee approach is vulnerable [1,13–15] to Denial-of-Service and network partition attacks targeting committee nodes. We discuss these two types of attacks together because any single node cannot distinguish them (Section 2), and we call them *targeted DoS attacks* altogether. Libra [3] also identifies DoS attacks as a significant threat but provides only partial mitigation (Section 2.2). Indeed, great progress has been made in designing scalable BFT protocols (e.g., SBFT [8]) running on a larger group of committee nodes and tolerating more nodes being attacked. However, these protocols designate a small number of committee nodes to finish critical tasks (e.g., combing ACKs), making these protocols' efficiency drop dramatically if these nodes are under DoS attacks (Section 2.2). With recent DoS attacks lasting for days [16,17], tolerating such attacks is crucial, yet challenging, for applications deployed on permissioned blockchains.

To address such vulnerabilities of static committees, a promising direction is to adopt the *dynamic committee* merit from permissionless blockchain systems [1,18]. These systems select a different committee for each block (mainly for fairness) and confirm a consistent sequence of blocks with a tailored consensus protocol [1,19]. As the committee member is rotated, the dynamic committee merit brings the potential to achieve liveness even if a committee is under targeted DoS attacks.

Simply applying the dynamic committee approach, however, is not enough for a permissioned blockchain to be resistant to targeted DoS attacks. To achieve DoS-resistance, it is crucial that the committee selection is unpredictable: the identities of nodes in a committee must be unpredictable to the attacker before the committee tries to achieve consensus on a block. Otherwise, the attacker can adaptively attack the ready-to-be committee and cause the system stuck. For instance, ByzCoin [20] lets the proof-of-work winners of recent blocks be the committee, but these explicit nodes are easily targeted by a DoS attacker, causing ByzCoin to lose liveness permanently [21].

To the best of our knowledge, Algorand is the only work that can tolerate targeted DoS attacks. Algorand adopts the dynamic committee approach and lets each node independently determine its committee membership based on the confirmed part of the blockchain. However, as Algorand is designed for permissionless blockchains, it confirms a block with up to 15 rounds and minute-level latency (discussed in Section 2.2), making it unsuitable for general data-sharing applications on permissioned blockchains (e.g., Libra [3]).

This paper aims to explore the new design point of building a permissioned blockchain's consensus protocol that adopts the unpredictable dynamic committee merit to defend against targeted DoS or targeted partition attacks, and at the same time, achieves comparable efficiency as existing BFT protocols (e.g., SBFT [8] has second-level latency). To achieve this goal, a main obstacle is to ensure that any selected committee meets the honesty requirement for byzantine problems: for consistency, each committee must have at most one-third of nodes being malicious [12]. Permissionless blockchains meet this requirement by selecting committees based on nodes' wealth in the cryptocurrency (i.e., proof-of-stake), but cryptocurrencies are usually unavailable in permissioned blockchains. Consequently, to meet such a requirement, one has to unrealistically assume that almost all member nodes ($>$90%) are honest (see Section 2).

Fortunately, the recent pervasive usage of hardware Trusted Execution Environments (TEEs) such as Intel SGX [22] in blockchain systems (e.g., Microsoft CCF [23], REM [24], Ekiden [25], Intel PoET [26]) shows that the code integrity feature of TEEs can surmount this obstacle. For instance, a recent implementation [27] of MinBFT leverages TEEs to ensure that a node cannot send different messages to different nodes and is incorporated into Hyperledger [2]. However, these systems still run their consensus protocols on a group of fixed and explicit committee nodes, making them susceptible to targeted DoS attacks.

We present EGES,[1] the first efficient consensus protocol that can tackle targeted DoS or targeted partition attacks for a permissioned blockchain. EGES adopts the dynamic committee merit to select a different committee for confirming each block. To defend against DoS or partition attacks targeting the committees, we leverage the integrity and confidentiality features of TEEs to present a new abstraction called *stealth committee*.

EGES's stealth committee has two new features. First, EGES selects a stealth committee in TEE: the selection progress has no communication among committee nodes, and the selection result cannot be predicted from outside TEE. This ensures that a committee node stays stealth (cannot be targeted by the attacker) before sending out its protocol messages. Second,

[1] EGES stands for Efficient, GEneral, and Scalable consensus.

Table 1
Comparison of EGES to baseline protocols. DoS resistance is analyzed in Section 2; evaluation setup is in Section 7. EGES is the only protocol that is both DoS-resistant and is among the fastest consensus protocols for permissioned blockchains.

Protocol	DoS and partition resistance	With TEEs?	Number of nodes	Tput (txn/s)	Confirm latency (s)
EGES	High	Yes	300	3226	0.91
			10 K	2654	1.13
Algorand	High	No	10 K	~727	~22
PoET	Medium[a]	Yes	100	149	45.2
Ethereum	Medium[a]	No	100	178	82.3
SBFT	Low	No	62	1523	1.13
MinBFT	Low	Yes	64	2478	0.80
BFT-SMaRt	Low	No	10	4512	0.67
Tendermint	Low	No	64	2462	1.31
HotStuff	Low	No	64	2686	2.63
HoneyBadger	Low	No	32	1078	9.39

[a]PoET and Ethereum cannot ensure consistency on network partition attacks [13].

when nodes in a committee are trying to confirm a block, EGES hides them into a large pool of fake committee nodes that behave identically as the real ones observed from outside TEE, so that an attacker cannot identify the real committees.

However, even equipped with TEEs and stealth committee, it is still challenging to efficiently ensure both consistency (i.e., no two member nodes confirm conflicting blocks) and reasonable liveness (i.e., allow non-empty blocks to get confirmed) in the asynchronous Internet due to the FLP impossibility [28]. Specifically, suppose a committee node x for the n^{th} block fails to receive the $(n-1)^{th}$ block after a timeout, x cannot distinguish whether it is because the committee for the $(n-1)^{th}$ block failed to confirm the $(n-1)^{th}$ block, or because x itself does not receive the confirmed $(n-1)^{th}$ block due to network problems. As the committee nodes for the $(n-1)^{th}$ block may be under DoS attacks and be unreachable, x must have a mechanism to distinguish these two scenarios in order to maintain both consistency and reasonable liveness in EGES.

EGES tackles this challenge by leveraging simple probability theory. EGES's committee for each block contains one proposer and n_A (e.g., 300) acceptors, randomly and uniformly selected from all nodes. The proposer broadcasts its block proposal to all nodes by P2P broadcasts and seeks quorum ACKS from the acceptors. EGES models the randomly selected acceptors as a sampling of the *delivery rate* of the proposal in the P2P overlay network [1]. In the previous example, EGES confirms the proposal for the $(n-1)^{th}$ block only if the proposal is delivered to a large portion of member nodes; if multiple rounds ($D = 4$ by default) of the sampling show that very few nodes have received that proposal for the $(n-1)^{th}$ block, nodes in EGES consistently confirm the $(n-1)^{th}$ block as an empty block (with an overwhelming probability).

In sum, EGES efficiently enforces consistency and can defend against targeted DoS or partition attacks. Specifically, EGES defends against such attacks by (1) letting committee nodes stay stealth *before* they start achieving consensus for a block, (2) using fake committee nodes to conceal real committee nodes *while* they are achieving consensus for a block, and (3) switching to a different committee and consistently confirming a block even if the attacker luckily guesses most real committee nodes for this block.

In essence, EGES's stealth committee is a moving target defense approach [29,30] that unpredictably replaces the committee to make a DoS attacker cannot launch effective targeted attacks. EGES is efficient because confirming a block in a gracious run (e.g., the proposer can reach most acceptors) only involves two P2P broadcasts and a UDP one-way delay (Section 4). We provide a rigorous analysis of EGES's DoS-resistance and proof of EGES's consistency guarantee in Section 5.

We implemented EGES using the codebase from Ethereum [31] and compared EGES with nine consensus protocols for blockchain systems, including five state-of-the-art efficient BFT protocols for permissioned blockchains (BFT-SMaRt [7], SBFT [8], HoneyBadger [32], and HotStuff [9]), two TEE-powered consensus protocols for permissioned blockchains (Intel-PoET [26] and MinBFT [33]), the default consensus protocol in our codebase (Ethereum-PoW [31]), and two permissionless blockchains' protocols that run on dynamic committees (Algorand [1] and Tendermint [18]). We ran EGES on both our cluster and AWS. The extensive evaluation results (Table 1) show that:

- EGES is robust. Among all consensus protocols for permissioned blockchains, EGES is the only protocol that can defend against targeted DoS and network partition attacks, by both a theoretical analysis (Section 5) and evaluation (Section 7.2).
- EGES is efficient. EGES confirms a block with 3000 transactions in less than two seconds in typical geo-distributed settings, comparable to evaluated consensus protocols that cannot tolerate targeted DoS attacks.
- EGES's throughput and latency are scalable to the number of nodes. When running 10k nodes, EGES showed 2.3X higher throughput and 16.8X lower latency than Algorand with 10k nodes.

Compared to existing BFT protocols [7–9,32] and TEE-powered consensus protocols [26,33] for permissioned blockchains, EGES is the only protocol that can tolerate targeted DoS attacks, and EGES's efficiency is comparable to the fastest of these protocols. Compared to Algorand, the only known DoS-resistant consensus protocol for permissionless blockchains, EGES has much higher throughput and lower latency.

Our contribution is three-fold. First, EGES leverages TEEs to explore the new design point of tackling DoS attacks while enforcing both consistency and reasonable liveness (including efficiency) for a permissioned blockchain in the asynchronous Internet. Second, we designed the new stealth committee abstraction and implemented EGES's consensus protocol. Our third contribution includes an implementation of the EGES prototype and the extensive experiments of EGES and existing blockchain consensus protocols on diverse adversarial network conditions, including targeted DoS attacks, ubiquitous DoS attacks, and network partitions. Our paper reveals that, in addition to safety and performance, DoS resistance is also an essential evaluation metric for practical Internet-scale blockchain applications (e.g., e-voting [34] and payment [3]). For instance, SGX-ToR [35], a blockchain client anonymity service, is shown [36] to be susceptible to DoS attacks targeting its directory service; deploying SGX-ToR on EGES can make SGX-ToR DoS-resistant (Section 7.5).

In the rest of the paper, Section 2 introduces EGES's background and motivation; Section 3 gives an overview of EGES; Section 4 introduces EGES's consensus protocol; Section 5 analyzes the safety and liveness of EGES; Section 6 covers our implementation; Section 7 shows our evaluation, and Section 8 concludes.

2. Background and related work

We discuss targeted DoS and network partition attacks together because these two attacks cannot be effectively distinguished in an asynchronous network. When a node cannot reach a remote node, the node cannot determine whether it is because the remote node is under DoS attacks or because the network is partitioned. Therefore, EGES maintains consistency by handling both cases together.

Although there exist many influential systems [37,38] addressing DDoS attacks on specific nodes, EGES is complementary to them because EGES handles diverse attack scenarios (e.g., an attacker controls nodes' P2P modules to cause network partition [13,15], see Section 3.2) from the consensus layer.

We assume that the attacker has an attack budget B (e.g., $B = 300$): the attacker can mount DoS attacks targeting B nodes at a time. We will formally define the threat model in Section 3.2.

2.1. Trusted execution environments and Intel SGX

EGES explores the design space that leverages the data confidentiality feature of hardware Trusted Execution Environments (TEEs) to make the committee members' identities stealth, and EGES leverages the code integrity feature of TEEs to ensure the faithful execution of EGES's protocol. TEEs provide both confidentiality and integrity to designated memory regions via hardware protection; the memory regions (containing code and data) are isolated from unauthenticated accesses, even from higher privileged system software. TEEs are widely available on most modern commercial platforms, including Intel SGX [22], ARM TrustZone [39], AMD SEV [40], and RISC-V Multizone [41].

In this paper, we present EGES's protocol with Intel SGX because SGX is one of the most widely accepted and studied [22] TEE implementations, but EGES's protocol is general for any TEE product with code integrity and data confidentiality guarantees. Intel SGX is a hardware feature on commodity Intel CPUs. SGX allows applications to crate TEEs called *enclaves*, where data and code execution cannot be seen or tampered with from outside. Code outside enclaves can enter an enclave by ECalls, and SGX uses remote attestations [22] to prove that a particular piece of code is running in an enclave on a genuine SGX-enabled CPU. SGX provides a trustworthy random source (`sgx_read_rand`), which calls the hardware pseudo-random generator through the RDRAND CPU instruction seeded by on-chip entropy sources [22]. Previous studies show that this random source complies with security and cryptographic standards and cannot be seen or tampered with from outside enclaves [42,43].

Recent work leverages SGX to improve diverse aspects of blockchain systems. Intel's PoET [26] replaces the PoW puzzles with a trusted timer in SGX; EGES is more efficient than PoET (Section 7.1). REM [24] uses SGX to replace the useless PoW puzzles with useful computation (e.g., big data), orthogonal to EGES. Microsoft CCF [23] is a permissioned blockchain platform using SGX to achieve transaction privacy, but it does not include a DoS-resistance approach. Scifer [44] uses SGX to maintain reliable node identities on the blockchain, which is adopted in EGES (Section 6.1).

Ekiden [25] and ShadowEth [45] offload the execution of smart contracts to SGX-powered nodes to avoid redundant execution and to preserve privacy; TEEChain [46] uses SGX to build an efficient and secure off-chain payment channel; Town Crier [47] uses SGX to build a trustworthy data source for smart contracts; Tesseract [48] uses SGX to build a cross-chain coin exchange framework; Obscuro [49] uses SGX to improve bitcoin's privacy; these systems do not focus on consensus protocols and are orthogonal to EGES.

2.2. Consensus for permissioned blockchains

We briefly introduce recent notable consensus protocols for permissioned blockchains, which are also EGES's evaluation baselines. Overall, *all* these protocols run on a static committee. To ensure liveness under a DoS attacker with an attack budget of B, these protocols must scale to $3 \times B + 1$ nodes (for BFT protocols) or $2 \times B + 1$ nodes (for SGX-powered protocols). However, to our best knowledge, no existing protocol can achieve such scalability.

BFT-SMaRt [7] is an optimized implementation of PBFT [50]. As each node broadcasts consensus messages to all other nodes, BFT-SMaRt has $O(n^2)$ message complexity to the number of committee nodes, resulting in poor scalability. Its paper [7] only evaluated up to 10 nodes. SBFT [8] is a scalable BFT protocol that uses a new type of committee nodes called collectors. A node sends its consensus messages to only c (usually $c < 8$) explicit collectors who will then broadcast a combined message using the threshold signature. SBFT's fast path can commit a block if fewer than c nodes fail; however, SBFT's performance drops dramatically if an attacker targets the c collectors (Section 7.4).

HotStuff [9] is a BFT protocol optimized for frequent leader changes, and Libra [3] leverages Hotstuff to tolerate targeted DoS attacks *on leaders*. However, since Hotstuff reports a near-linear increment of latency with an increasing number of nodes, it only evaluated up to 128 nodes, where an attacker can DoS attack or partition one-third of all nodes rather than finding the leader. HoneyBadger [32] uses randomization to remove the partial synchrony assumption of PBFT. However, both its paper and our evaluation show that HoneyBadger achieves high latency due to many rounds of broadcasts in its asynchronous byzantine agreements.

MinBFT [33] is an SGX-powered BFT protocol with the same fault model as EGES. MinBFT reduces the number of rounds in PBFT and can tolerate more faulty node failures, but MinBFT still has $O(n^2)$ message complexities, so its performance is not scalable to the number of nodes.

2.3. Consensus for permissionless blockchains

Existing permissionless blockchains can be divided into two categories based on how they confirm blocks. The first category confirms block with variants of the longest-chain rule (i.e., Nakamoto consensus [4]), including BitCoin [4], Ethereum [31], BitCoin-NG [51], Snow-White [52], Ouroboros [53], Paros [54], Genesis [55], and GHOST [56]. Specifically, each node asynchronously selects the longest chain it received and confirms a block when there are k blocks succeeding it. However, waiting for k more blocks leads to a long confirm latency, and previous work [57] shows that this k must be large enough to ensure consistency. Moreover, the longest-chain rule cannot ensure consistency under partition attacks [1,14,58]. Intuitively, during a network partition, each partition will independently grow a chain; if these chains diverge for more than k blocks, nodes in different partitions will confirm conflicting blocks.

The second category of permissionless blockchains confirms blocks using the committee-based BFT approach, which can confirm a block as soon as the BFT consensus is achieved. This category includes Algorand [1], ByzCoin [20], Tendermint [18], and PeerCensus [59]. These systems select distinct (dynamic) committees for different blocks based on the content (e.g., nodes' wealth) on the blockchain for fairness and for handling nodes joining or leaving. Similar to EGES, these systems run a tailored consensus protocol (BA* in Algorand [1], Tendermint [18], Tenderbake [60], and Tenderand [19]) on dynamic committees to confirm blocks.

However, these protocols cannot be ported to a permissioned blockchain because of the tight coupling with cryptocurrency. For instance, although Tendermint [61] and Tenderbake [60] are described as stand-alone BFT protocols, they assume that in any committee, fewer than one-third of nodes are malicious. In a permissioned blockchain without cryptocurrency, if we want to ensure that any randomly selected committee (say 100 nodes) from a large number of (say 10k) nodes meets this requirement with overwhelming probability ($> 1–10^{-10}$), we need to assume over 91% of all nodes being honest (by the hypergeometric distribution), which is an overly-strong assumption for a practical large-scale blockchain system (e.g., a global payment system [3]) on the Internet.

Moreover, these systems (except Algorand) cannot ensure liveness under targeted DoS attacks because they select committees in a predictable way so that all nodes can verify the identities of committees. For instance, ByzCoin [20] lets the proof-of-work winners of recent blocks be the committee. However, these nodes with explicit identities are easily targeted by a DoS attacker, and ByzCoin may lose liveness permanently [21] if more than one-third of these nodes are attacked. Algorand defends against targeted DoS attacks by letting each node use verifiable random functions to determine its committee membership. We provide a detailed comparison showing why EGES is more efficient than Algorand in Section 4.3.

3. Overview

3.1. Architecture

EGES is a consensus protocol for a permissioned blockchain running on M member nodes (*nodes* for short) connected with an asynchronous network. EGES adopts the hybrid fault model used in existing SGX-powered consensus protocols [33,62,63], where each node has a trusted module (i.e., the SGX enclave) that will only fail by crashing, and all other components can behave arbitrarily.

Fig. 1 shows the architecture of an EGES node. Each node is equipped with an attested SGX enclave running only EGES's consensus protocol. The blockchain application layer, the transaction generation and query libraries (e.g., web3.js [64]), and the blockchain storage module are outside the enclave because they are already cryptographically protected. The P2P network and operating system are outside the enclave and untrusted.

For each block index n, EGES determines one committee among all nodes (uniqueness proved in Section 5.1), and a node's *committee selection module* (Section 4.2) knows whether the node is a committee member only within its enclave.

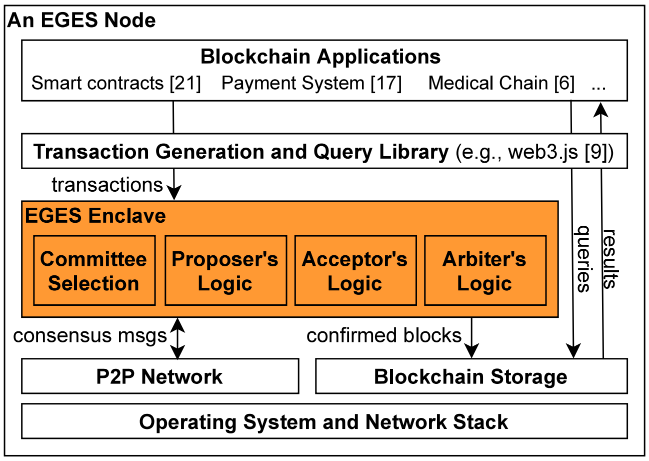


Fig. 1. An EGES node's architecture. EGES's consensus protocol has four components running in the enclave. Block proposals are included in consensus messages.

Each committee has one *proposer* (denoted as P_n), a group of *acceptors* (denoted as $\mathbb{A}_n$) with the count of n_A, and a group of *arbiters* with the count of n_α. A committee node's enclave activates corresponding *committee logic modules* for this nth block according to its committee roles if the node is a committee member, and the committee logic modules will generate or respond to consensus messages following EGES's consensus protocol (Section 4.3).

The proposer P_n takes a batch of transactions from outside the enclave, generates a unique block proposal (denoted as $proposal_n$) within its enclave, and tries to finalize it by collecting quorum ACKs from the acceptors (Section 4.3). EGES ensures that P_n's identity is unknown to any node's modules outside the node's enclave (even if the node is controlled by an attacker) before P_n broadcasts its $proposal_n$, while all acceptors $\mathbb{A}_n$'s identities are unknown to the outside throughout the whole consensus process (Section 4.3). To handle DoS attacks targeting P_n, EGES uses the arbiters to help P_n finalize $\texttt{proposal}_n$ if P_n is under attack, but the arbiters do *not* generate new block proposals (Section 4.4).

SGX is essential for EGES due to three main reasons. First, EGES uses SGX to regulate the behaviors of randomly selected committee nodes (Section 4.3); otherwise, the blockchain may fork if committee nodes equivocate (i.e., sending conflicting messages to different nodes). In EGES, each node has a private key that is *only visible* within the node's SGX enclave, and the corresponding public key works as the node account saved in all other nodes' SGX enclaves (Section 4.1). A valid consensus message must carry a valid signature, proving that the message is generated in the sender node's enclave with code integrity. By doing so, a node cannot equivocate or forge protocol messages (e.g., a proposer sending out a `finalize` message without receiving quorum ACKs).

Second, EGES leverages SGX to make its committee's identities stealth: only a node's enclave knows whether it is a committee member for the current block (Section 4.2). This not only enables EGES to maintain practical liveness under DoS attacks, but more importantly, makes EGES's consistency model resistant to targeted attacks. Specifically, EGES leverages probability theory to model randomly selected acceptors as a uniform sampling of the delivery rate of a block proposal in the P2P network (same as Algorand [1], see Section 4.3). If acceptors' identities are public, an attacker can selectively transfer or drop packets towards them, breaking EGES's safety.

Third, the usage of SGX enables EGES to select a stealth committee with a known count. As illustrated later in Section 4.3, this helps EGES to be more efficient than Algorand.

EGES has the following design goals:

- **Safety (consistency)**. EGES ensures safety in an asynchronous network. Formally, if a node confirms a block b as the nth block on the blockchain, the probability that another node confirms $b' \neq b$ as the nth block is overwhelmingly low ($< 10^{-10}$).
- **DoS-resistance (liveness)**. In addition to safety, EGES can make progress (i.e., allow non-empty blocks to be confirmed) with the assumptions about DoS attackers' capability as described below.

Let us reconsider the subtle challenge we mentioned in Section 1 with a concrete example in Fig. 2. This challenge is unique in EGES because EGES uses different committees for different blocks to resist DoS attacks targeting the committee. When a node cannot receive a block after a timeout, for liveness, the node cannot wait forever, but for safety, the node must figure out whether this block may have been confirmed by some nodes. Existing consensus protocols on static committees use a view change protocol that queries how many nodes have sent out ACKs and leverages quorum intersections [50,65,66] to address this problem. However, in EGES, this method is not viable because EGES must ensure liveness even if most nodes in $\mathbb{A}_n$ are DoS attacked after sending out their ACKs.

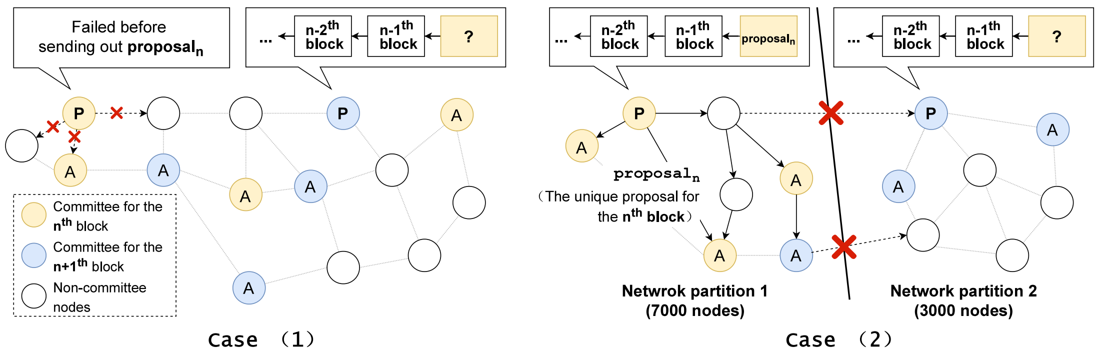

Failed before sending out proposal$_n$
... n-2th block n-1th block ?
Committee for the nth block
Committee for the n+1th block
Non-committee nodes
Case (1)
... n-2th block n-1th block proposal$_n$
proposal$_n$
(The unique proposal for the nth block)
Netwrok partition 1 (7000 nodes)
Network partition 2 (3000 nodes)
Case (2)

Fig. 2. A key challenge of EGES is to determine whether a block (the nth block in this example) has ever been confirmed when a node cannot receive the block after a timeout. 'P' means the proposer; 'A' means an acceptor; arbiters are omitted in this figure for brevity.

Fig. 2 shows two subtle cases illustrating the challenge. Note that these two cases are deliberately simple for a clear exposition of EGES's idea, and EGES can ensure safety in all scenarios in the asynchronous network (Section 5.1). In case (1), the proposer for the nth block (P_n) failed before broadcasting its $proposal_n$. Therefore, $proposal_n$ is not confirmed on any node, and all nodes can safely confirm an empty block at the nth position. In case (2), nodes are divided into two partitions right after the $n - 1^{th}$ block is confirmed. P_n and most nodes of $\mathbb{A}_n$ are in partition 1, so nodes in partition 1 can confirm the $porposal_n$ successfully. In this case, although nodes in partition 2 cannot receive $porposal_n$, they should *not* confirm an empty block. From any single node point of view, these two cases cannot be effectively distinguished.

To address this challenge, EGES introduces a new consensus protocol based on probability theory (Section 4.3). Specifically, if $\texttt{proposal}_n$ is confirmed on some nodes, $\texttt{proposal}_n$ should have been delivered to a large enough portion of nodes in the P2P network because (1) confirming $\texttt{proposal}_n$ needs quorum ACKs from nodes in $\mathbb{A}_n$, and (2) $\mathbb{A}_n$ is uniformly selected from all nodes. Therefore, if we repetitively sample many nodes from all nodes, and no node has received $\texttt{proposal}_n$, we can predicate that only a small portion of (or no) nodes have received $\texttt{proposal}_n$, and thus the probability of $\texttt{proposal}_n$ having been confirmed is overwhelmingly low. To be DoS-resistant, these multiple rounds of checking must be initiated by different nodes, so EGES lets the proposers for subsequent blocks (i.e., P_{n+1}, P_{n+2}, etc.) do such samplings while seeking ACKs for their own proposals (Section 4.3).

3.2. Threat model

SGX's threat model. EGES has the same threat model for SGX as typical SGX-based systems [27,67–70]. We trust the hardware and firmware of Intel SGX, which ensures that code and data in an enclave cannot be seen or tampered with from outside. We trust that the remote attestation service can identify genuine SGX devices from fake ones (e.g., emulated with QEMU). Side-channel and access pattern attacks on SGX are out of the scope of this paper. Moreover, the adversary cannot break standard cryptographic primitives, including public-key based signatures and collision-resistant hash functions.
Transaction model. Same as most existing blockchain systems [1,24,26,71], EGES assumes that (1) each transaction has a verifiable client signature, and (2) the execution and validation of any transaction are deterministic and can be performed by any node independently.
Communication model. EGES maintains safety in an asynchronous network, where network packets can be dropped, delayed, or reordered arbitrarily. Nodes may be nonresponsive, due to going offline or targeted DoS attacks (e.g., botnet DDoS attacks [16]) by a DoS attacker. When a node cannot reach a remote node, the node cannot determine whether the remote node is under DoS attack (or is offline) or the network packets are delayed. Nodes are equipped with loosely synchronous clocks (e.g., by running NTP), but EGES does not rely on the correctness of these clocks for safety.

To achieve liveness, same as existing protocols [7,9,50,72–74]. EGES has the partial synchrony [75] assumption: there is an unknown global stabilization time (GST), after which messages between two nodes not under DoS attacks can be delivered within a known time-bound.

Nodes are connected with a P2P overlay network, same as existing large-scale blockchain systems [1,4]. Each node has a P2P module connecting to a random set of other nodes and relays messages using the gossip protocol [76–79].

A node's P2P module is outside SGX and can be controlled by the attackers [15]: the attacker can partition some nodes from other nodes [13,14]) or selectively pass consensus messages to nodes' SGX enclaves. However, such manipulations are already included in EGES's asynchronous network assumption. For safety, EGES leverages the sampling merit to estimate the delivery rate of a specific block proposal and derives overwhelming probability, regardless of how nodes are connected. For liveness, EGES can tolerate the adversary controlling the P2P modules of a number of nodes with the restriction of the adversary's attack budget described below.
Assumptions on the capability of DoS attackers. EGES has three assumptions on the capability of a DoS attacker, same as Algorand [1] and existing move target defense (MTD) systems [29,30]. First, the adversary has a targeted attack budget

B: the adversary cannot *constantly* cause more than B targeted nodes in EGES to be nonresponsive. B can be a constant number (e.g., 300) or a fraction (e.g., 10%) of the total number of nodes, but B should be bounded by the ubiquitous attack threshold in the second assumption below. Note that this budget B is adaptive: the adversary can attack different nodes at different times, but the number of attacked nodes at a time cannot be constantly larger than B. By "constantly", we mean that the adversary can sometimes cause more than B targeted nodes to be non-responsive (e.g., because most randomly selected committee nodes are luckily controlled by the adversary) and cause an EGES block cannot be confirmed. However, EGES can still achieve liveness by letting subsequent committees consistently confirm this block (Section 4.3).

Second, the attacker can conduct ubiquitous DoS attacks (without targeting specific nodes) or partition a number of nodes from other nodes (e.g., by manipulating nodes' P2P modules [13,14]). However, the P2P overlay network should have a large enough portion (e.g., 65%) of nodes connected. We provide a quantitative analysis of how EGES can preserve liveness under such attacks in Section 5.3.

Third, the adversary cannot constantly succeed in mounting an attack targeting a node within the time window for the node to send out an EGES protocol message. Specifically, EGES protocol messages are larger than the network maximum packet size and are fragmented into multiple packets; an EGES committee node's identity is unknown to the adversary before sending out the first packet, and we assume that the adversary cannot mount targeted DoS attacks until the node sends out all packets belonging to this message (at most hundreds of kB and can be sent within one second).

EGES already assumes a strong attacker for practical distributed systems on the Internet. Although modern botnet attacks may call up thousands of devices to conduct an attack [80–82], the attacker has to use tens or hundreds of devices to successfully make a single targeted node unresponsive [80,83,84]. Moreover, existing single-node DoS defense techniques [81,82,85], although cannot completely defend DoS attacks targeting this node, can further elevate the number of botnet devices required to make this node unresponsive. Therefore, setting the budget to a few hundred (e.g., 300 in EGES's evaluation) is sufficient for defending most botnet DoS attacks targeting EGES committees.

As pointed out by Algorand [1], a more powerful adversary than our model usually controls the internet service provider and can prevent all EGES nodes from communicating at all: no practical system can ensure liveness under such a strong adversary, and such attacks can be easily detected. We will provide a rigorous analysis of EGES's DoS resistance in Section 5.2.

DoS resistance of EGES. EGES has three important features to achieve DoS resistance:

- EGES randomly selects a distinct committee for each block. The selection is done inside the SGX enclaves of a previous committee, and the selection result is encrypted on the confirmed common prefix of the blockchain. By doing so, a committee node can determine its committee membership without interactions with other nodes, making it *stay stealth* before trying to achieve consensus on its block.
- When a committee is achieving consensus for a given block, EGES uses fake committee nodes to conceal the real ones by sending encrypted dummy messages. Since whether a node is a real committee node is only known within the node's SGX enclave, and the encrypted dummy messages are of the same format as real ones, a DoS attacker cannot distinguish the real committee nodes from the fake ones. Therefore, the attacker must have an unrealistic large attack budget to attack all the real and fake committees; otherwise, he has to randomly guess who are the real ones.
- Even if the attacker luckily guesses the real ones (he may eventually succeed if trying persistently), EGES can ensure safety with overwhelming probability. Specifically, even if a committee cannot confirm its own block, committees for subsequent blocks can help to consistently confirm this block (Section 4.3). This feature is in contrast to most existing consensus protocols (i.e., all except Algorand [1]), where the system must wait statically until a quorum of nodes becomes reachable.

4. EGES consensus protocol

4.1. Protocol preliminaries

Protocol parameters. EGES's consensus protocol has three parameters, n_A (default 300), τ (default 59%), and D (default 4), where n_A is the number of acceptors, τ is the quorum ratio, and D is the finalization depth for an empty block. We will show how to select these parameters in Section 5.3 and how these parameters affect EGES's performance in Section 7.3.

Block structure. EGES adds one data field to the block structure of common blockchain systems [4,31]: the encrypted committee identities for a future block (Section 4.2). EGES is oblivious to how transactions are stored or executed (see Fig. 3).

Invariant 1 (*See Section 5.1 for Proof*). *For any block index n, at most one unique block proposal* ($\texttt{proposal}_n$) *is generated; a node can only confirm* $\texttt{proposal}_n$ *or a default empty block* ($\texttt{empty}_n$) *as its nth block.*

Block status. Each block in a node's `chain` has three states: *undecided*, *finalized*, and *confirmed*. An undecided nth block can only be $\texttt{empty}_n$. A node appends $\texttt{empty}_n$ to its `chain` when the node triggers a timeout waiting for the `finalize` message for the nth block; the block is in the undecided state because the node cannot determine (for now) whether it should confirm $\texttt{proposal}_n$ or $\texttt{empty}_n$.

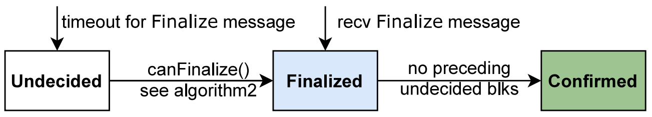


Fig. 3. EGES's block status diagram.

A finalized n^{th} block can be either $\texttt{proposal}_n$ or $\texttt{empty}_n$. EGES ensures the following invariant:

Invariant 2. *If a node's n^{th} block is finalized as $\texttt{proposal}_n$, no other nodes will finalize the n^{th} block as $\texttt{empty}_n$, and vice versa.*

There are two rules for appending a finalized n^{th} block: (1) a node appends finalized $\texttt{proposal}_n$ if it receives the `finalize` message for $\texttt{proposal}_n$, and (2) a node changes the $\texttt{empty}_n$ from the undecided state to finalized if the node can predicate that no node has finalized $\texttt{proposal}_n$ (Section 4.3).

A node confirms its finalized n^{th} block if all blocks with indices smaller than n in its `chain` are finalized. Note that although EGES may finalize blocks out of order, EGES confirms blocks sequentially, same as typical blockchains [2,31].

Each node's local states. Each EGES node maintains three major local states: a local blockchain (the `chain`), a proposal cache (the `cache`), and a set of `learntProposals`, The `cache` is maintained in the node's EGES enclave. When the node receives $\texttt{proposal}_n$, it puts the proposal into the `cache`, in case the committees of future blocks query the delivery rate of $\texttt{proposal}_n$.

The `chain` on each node is divided into two parts: the confirmed part and the unconfirmed part. We use `MC` to represent the maximum confirmed index and $\mathbb{U}$ to represent the indices of undecided blocks in `chain`. The confirmed part of `chain` (i.e., indices$\leq$`MC`) is cryptographically-chained by hash values and can be saved out of the enclave and get executed, while unconfirmed parts are saved in the EGES enclave. The `learntProposals` is the set of known proposals for undecided blocks on this node and is saved in EGES enclave.

Membership and key management. Each node i has a key pair $\langle pk_i, sk_i \rangle$, with the public key pk_i as its account, and its secret key sk_i is *only visible within its enclave*: even this node's administrator cannot see the plain-text of its secret key. We use the notations from PBFT [50]: we denote a message m_1 sent to node i encrypted by i's public key pk_i as $\{m_1\}_{pk_i}$; we denote a message m_2 generated by node i's enclave and signed by sk_i as $\langle m_2 \rangle_{\sigma_i}$. For efficiency, EGES signs on message digests.

To ease understanding, we describe EGES protocol on a fixed membership, where all nodes' accounts (public keys) are loaded to nodes' EGES enclaves a priori, and all nodes' EGES enclaves are attested. We show how EGES supports dynamic membership and attestations in Section 6.1.

4.2. Selecting a stealth committee

For each block, EGES selects a committee, including **one proposer**, $\boldsymbol{n_A}$ acceptors, and $\boldsymbol{n_\alpha}$ arbiters, in an unpredictable way without communication among nodes.

The committee members for the n^{th} block is selected in the EGES enclave of P_{n-lb}, and these committee nodes' identities are encrypted in the $(n-lb)^{th}$ block. lb (look-back) is a system parameter and needs to be large enough (e.g., the number of blocks confirmed in days) to ensure that even when the network condition is poor, and new blocks cannot be confirmed in time, EGES can still derive committees for future blocks. We assume that the first lb committees' identities are encrypted in the genesis (0^{th}) block by a trusted party, or that the blockchain is bootstrapped in a controlled domain for at least lb blocks. Note that the value of lb does not affect EGES's safety.

Occasionally, a node may be selected as the committee for a future block and then leave the system, which EGES already tolerates as a failed node. If the $(n-lb)^{th}$ block happens to be an empty block, EGES uses the committee identities encrypted in the $(n-2lb)^{th}$ block (and identities in the $(n-3lb)^{th}$ block if the $(n-2lb)^{th}$ block is also empty, recursively). Although this proposer's identity is already explicit when confirming the $(n-lb)^{th}$ block and may be targeted, EGES can tolerate it as a failed proposer and uses subsequent committees to confirm $\texttt{empty}_n$.

$P_{(n-lb)}$ selects the committee for the n^{th} block with two steps, which are done in $P_{(n-lb)}$'s EGES enclave to ensure both integrity (i.e., an attacker cannot control the selection) and confidentiality (i.e., an attacker cannot know the selection result). In the first step, $P_{(n-lb)}$ randomly selects the committee members from all member nodes following the uniform distribution. Recall that the member list is loaded in the EGES enclave on each node (Section 4.1), so $P_{(n-lb)}$ simply selects $n_A + 1$ nodes from the list using the SGX's trustworthy pseudo-random number generator as the random source, which has been shown to be cryptographically-secure and cannot be seen or tampered with from outside enclave (Section 2.1).

In the second step, for each selected committee node, $P_{(n-lb)}$ generates one certificate, which is the cipher-text of the concatenation of a predefined byte string and a random nonce (for making the cipher-text unpredictable), encrypted with that committee node's public key. Then, $P_{(n-lb)}$ includes these $(n_A + n_\alpha + 1)$ certificates in the $(n-lb)^{th}$ block's proposal. The first certificate is for the proposer, and the other n_A certificates are for acceptors. When a node confirms this block, it

Propose	
n	The index of the proposal
blk	The content of the block to be proposed
$\mathtt{MC}_p$	The MC value of the proposer
$\mathbb{U}_p$	The $\mathbb{U}$ list of the proposer
$\mathtt{proposal}_{u_m}$	$u_m = \max(\mathbb{U}_p)$. The proposer tries to finalize this proposal together with its proposed n^{th} block
ACK	
n	The index of corresponding proposal
sender	The sender's public key (account)
notifications	Proposals to notify the proposer
isReal	For identifying real ACK from cover messages
nonce	Random padding to make cipher text unpredictable
Finalize	
n	The index of the block finalized
u_m	The index of the block that is finalized together
learnt	Proposals learnt from acceptors notifications

Table 2: EGES's messages' fields. Blue fields are used only in the checking mode (`nil` in the normal mode).

Algorithm 1: the proposer for the n^{th} block

```
1  $n_A$ ← the number of acceptors
2  blk ← the content of the $n^{th}$ block to propose
3  function normalPropose():
4      bcast ⟨Propose, blk, MC, nil, nil⟩$_{\sigma_{me}}$
5      upon receiving ⟨ACK, n, $pk_i$, nil⟩$_{\sigma_i}$
6          ACKs.insert($pk_i$)
7          if $ACKs.count \geq \tau \times n_A$ : bcast ⟨Finalize, n, nil, nil⟩$_{\sigma_{me}}$
8  function checkPropose():
9      $u_m$ ← max($\mathbb{U}$)
10     if $cache[u_m]$ != nil || $learntProposals[u_m]$ != nil :
11         $proposal_{u_m}$ ← the proposal for index $u_m$
12     else: $proposal_{u_m}$ ← nil
13     bcast ⟨Propose, blk, MC, $\mathbb{U}$, $proposal_{u_m}$⟩$_{\sigma_{me}}$
14     learnt = []
15     upon receiving ⟨ACK, n, $pk_i$, notifications⟩$_{\sigma_i}$
16         ACKs.insert($pk_i$)
17         learnt.insert(notifications)
18         if $ACKs.count \geq \tau \times n_A$ :
19             bcast(Finalize, n, $proposal_{u_m}$, learnt)
```

Algorithm 2: all nodes' action

```
/* All message senders' memberships and signatures are verified, ommited
   in all algorithms for brevity                                          */
1  upon receiving msg = ⟨Propose, n, blk, $MC_p$ ...⟩$_{\sigma_i}$
2      cache[n] = msg
3      if $MC < MC_p$ : ask peers for missing blocks
4      if $hash(sk_i, n) > threshold$ : Reply fake (cover) ACKs message
         /* same format as true ACKs, with isReal = false.              */
5  upon receiving ⟨Finalize, n, $u_m$, learnt⟩$_{\sigma_i}$
6      if $u_m \neq nil$ :
7          chain[$u_m$].status ← finalized
8          $\mathbb{U} = \mathbb{U} \setminus u_m$
9      if $|\mathbb{U}| == 0$ : Confirm chain up to index n (MC ← n)
10     else:
11         learntProposals.insert(learnt)
12         for $u$ in $\mathbb{U}$ in descending order :
13             if canFinalize($u$) :
14                 chain[u].status = finalized
15             else: break // Empty blocks are finalized in descending
                  order.
16 upon timeout waiting for next block
17     chain.append (empty, status = undecided)
18 function canFinalize($u$):
19     count = 0
20     for $i = u + 1; i \leq chain.len; ++i$ :
21         if $chain[i]$ != $empty$ :
22             count++
23             if $count \geq D$ : return true
24         else: return false
```

Algorithm 3: an acceptor for the n^{th} block

```
1  upon receiving ⟨Propose, blk, $MC_p$, $\mathbb{U}_p$, $proposal_{u_m}$⟩$_{\sigma_i}$
2      r = rand()
3      if $MC_p < MC$ : notify proposer to catch up (omitted)
4      if $|\mathbb{U}_p| == 0$ :
5          reply ⟨{ACK, n, $pk_{me}$, nil, true, r}$_{pk_i}$⟩$_{\sigma_{me}}$
           // Only the leader($i$) can decrypt this message
6      else:
7          notifications = []
8          for $u$ in $\mathbb{U}_p$ :
9              if $cache[u]$ != nil : notifications.append(cache[u])
10         reply ⟨{ACK, n, notifications, $true$, $r$}$_{pk_i}$⟩$_{\sigma_{me}}$
```

Fig. 4. EGES's consensus protocol.

tries to decrypt one of these certificates using its own secret key in its enclave; if the node can get the predefined string, it predicates that it is a committee node for the n^{th} block.

Despite using asymmetric cryptography, this mechanism is efficient in EGES because both encryption and decryption are done asynchronously off the consensus' critical path. For encryption, since $P_{(n-lb)}$'s enclave knows it is the proposer for the $(n - lb)^{th}$ block after confirming the $(n - 2lb)^{th}$ block, $P_{(n-lb)}$ starts selecting the committees and encrypting the certificates as soon as confirming the $(n-2lb)^{th}$ block. Similarly, the decryption is also off the critical path as the decryption result is used lb blocks later.

In addition, EGES also selects a group of *fake acceptors* for each block index n. Since EGES does not need to control the exact identities or number of the fake acceptors, EGES lets each node independently determine whether it is a fake acceptor within its enclave (Algorithm 2 Line 4), and EGES only control the expected number of fake acceptors by controlling the probability for each node to become a fake acceptor. We use the n_F to represent the number of fake acceptors, and n_F is a binomial random variable.

EGES's committee selection mechanism is unpredictable and non-interactive because: (1) the random source cannot be seen or tampered with from outside the enclave of P_{n-lb}, and the certificates can only be verified within a selected committee node's enclave; and (2) the selection process is solely done within the EGES enclave of P_{n-lb}. These two features ensure the committee nodes' identities are not exposed during the selection, so the committee nodes cannot be targeted before sending out protocol messages for the n^{th} block.

Discussions. EGES selects only one proposer for each block to achieve good efficiency: EGES only needs to achieve a binary consensus on whether to confirm the unique proposal by this proposer or a default empty block. For acceptors, an alternative design is to let each node *independently* determine whether it is an acceptor for the current block with a probability, and EGES only controls the *expected* total count. However, this alternative design will lead to a much larger quorum ratio (i.e., τ) to ensure safety and thus worse liveness (quantitative analysis in Section 7.3) (see Fig. 4).

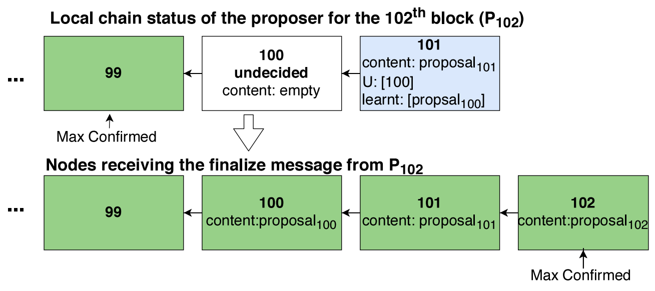

Local chain status of the proposer for the 102th block (P102)
...
99
100 undecided content: empty
101 content: proposal101 U: [100] learnt: [propsal100]
Max Confirmed
Nodes receiving the finalize message from P102
99
100 content:proposal100
101 content: proposal101
102 content:proposal102
Max Confirmed

Fig. 5. An example where p_{102} helps finalizing $\texttt{proposal}_{100}$ while proposing its (102th) block.

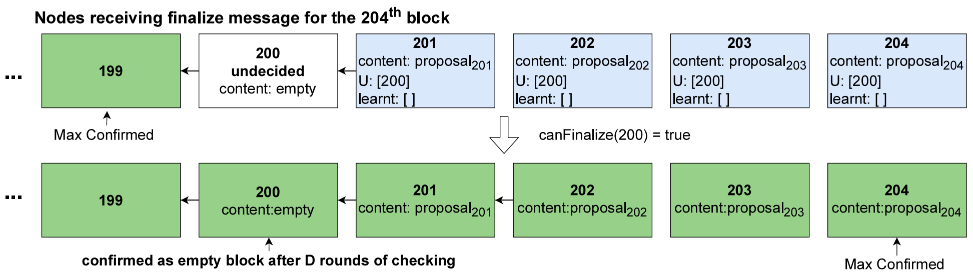

Nodes receiving finalize message for the 204th block
...
199
200 undecided content: empty
201 content: proposal201 U: [200] learnt: [ ]
202 content: proposal202 U: [200] learnt: [ ]
203 content: proposal203 U: [200] learnt: [ ]
204 content: proposal204 U: [200] learnt: [ ]
Max Confirmed
canFinalize(200) = true
199
200 content:empty
201 content: proposal201
202 content:proposal202
203 content:proposal203
204 content:proposal204
confirmed as empty block after D rounds of checking
Max Confirmed

Fig. 6. An example for confirming an empty block (200th) after $D = 4$ succeeding blocks containing 200 in $\mathbb{U}$.

4.3. *Confirming a block*

As shown in Fig. 4, a proposer P_n has two operation modes: *normal* mode and *checking* mode. P_n is in the normal mode if all blocks in its `chain` before n are confirmed (i.e., $\mathbb{U} = \emptyset$), and P_n tries to confirm $\texttt{proposal}_n$ quickly. Otherwise, P_n is in the checking mode: while proposing $\texttt{proposal}_n$, it also checks the status of the undecided blocks in its `chain`.

Normal mode. Algorithm 1 Line 3∼7 shows how a normal mode P_n tries to confirm $\texttt{proposal}_n$ in a gracious run. First, P_n broadcasts a `propose` request through the P2P network carrying $\texttt{proposal}_n$ and its `MC` (Section 4.1). The `MC` value helps nodes align confirmed parts of their `chain`: if a node's `MC` is smaller than the proposer's, the node asks for the missing confirmed blocks from its peers (Algorithm 2 Line 3). Upon receiving this `propose` request, an acceptor replies an `ACK` using UDP directly to P_n (Algorithm 3 Line 5). Second, P_n waits for quorum ($\tau \times n_A$) `ACK`s from $\mathbb{A}_n$. P_n does not know which nodes are acceptors, but EGES's ensures that a non-acceptor cannot send valid `ACK`s (Section 4.1). Third, P_n broadcasts a `finalize` message; on receiving this message, a node finalizes $\texttt{proposal}_n$.

Checking mode. P_n is in the checking mode if it has undecided blocks (i.e., $\mathbb{U}$ is non-empty), and its workflow is shown in Algorithm 1 Line 8∼19. P_n checks the status of its undecided blocks and tries to finalize them (if possible) by adding additional fields to the `propose` message.

Each $u \in \mathbb{U}$ in P_n's `chain` is categorized into one of the two types: (1) if P_n has learnt the unique $\texttt{proposal}_u$, either from the `propose` messages or from P_u from the notifications of other nodes, we call u a "known undecided" block; (2) otherwise we call u "unknown undecided". For each unknown undecided block u, P_n tries to learn $\texttt{proposal}_u$ from $\mathbb{A}_n$. If P_n learns the proposal, P_n carries it in the `finalize` message in order to let subsequent proposers finalize it. Otherwise, P_n carries a message stating that most acceptors in $\mathbb{A}_n$ never received $\texttt{proposal}_n$, and a node receiving this message finalize $\texttt{empty}_n$ if the node received such messages from more than D consecutive proposers (Algorithm 2 Line 18).

For known undecided blocks, P_n helps to finalize *only* $\texttt{proposal}_{u_m}$ where $u_m = max(\mathbb{U})$ and leaves other blocks for subsequent proposers. In other words, undecided blocks must be finalized in descending order. This is because, without this restriction, when a node finalizes $\texttt{empty}_n$, it only ensures proposers for blocks with index $\leq n + D$ has not finalized $\texttt{proposal}_n$; adding this restriction helps to ensure that proposers for blocks with index $> n + D$ cannot finalize $\texttt{proposal}_n$.

To help understanding, we give three concrete examples showing (1) how a proposer helps to finalize an undecided proposal, (2) how a proposer finalizes an undecided block as empty, and (3) why a checking mode proposer can only finalize $\texttt{proposal}_{u_m}$ where $u_m = max(\mathbb{U})$.

Fig. 5 gives an example illustrating how a proposer P_{102} helps to finalize an undecided $\texttt{proposal}_{100}$. Suppose P_{100} failed just before broadcasting its `finalize` message. Therefore, the 100th block's state is undecided among all nodes. Then, P_{101} learns $\texttt{proposal}_{100}$ and carries it in its `finalize` message, and P_{102} learns it. Then P_{102} finalizes $\texttt{proposal}_{100}$ together with $\texttt{proposal}_{102}$. Moreover, since all blocks before 102 are finalized, the chain is confirmed up to 102.

Fig. 6 gives an example showing how an undecided block is finalized as empty. Suppose P_{200} failed before broadcasting its proposal, and $D = 4$. When P_{201}∼P_{204} asks whether their acceptors ($\mathbb{A}_{201}$∼$\mathbb{A}_{204}$) receive $\texttt{proposal}_{200}$, they get no

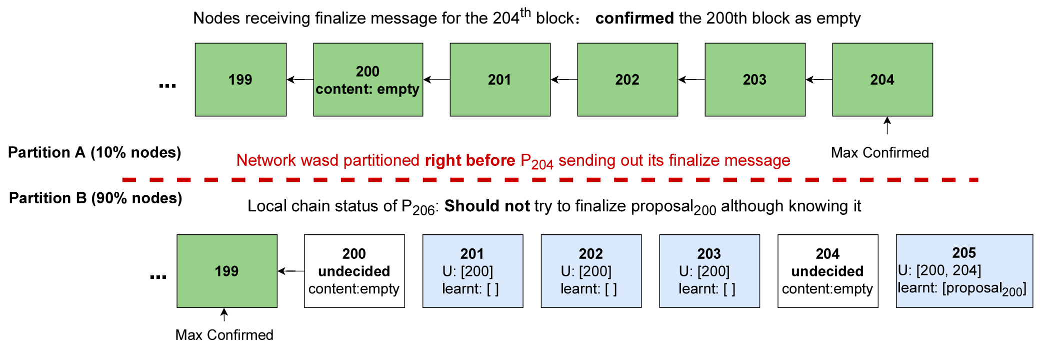

Nodes receiving finalize message for the 204th block: confirmed the 200th block as empty
199
200 content: empty
201
202
203
204
Max Confirmed
Partition A (10% nodes)
Network wasd partitioned right before P204 sending out its finalize message
Partition B (90% nodes)
Local chain status of P206: Should not try to finalize proposal200 although knowing it
199
200 undecided content:empty
201 U: [200] learnt: [ ]
202 U: [200] learnt: [ ]
203 U: [200] learnt: [ ]
204 undecided content:empty
205 U: [200, 204] learnt: [proposal200]
Max Confirmed

Fig. 7. An example showing why a checking mode proposer can finalize only $\texttt{proposal}_i$ where $i = max(\mathbb{U})$. $D = 4$ in this example.

positive answers. Therefore, these four blocks are all finalized, carrying a message stating that four samplings have been conducted on the delivery rate of $\texttt{proposal}_{200}$, but *no replied node has received* $\texttt{proposal}_{200}$. This indicates that the probability that $\texttt{proposal}_{200}$ is finalized at some nodes is overwhelmingly low. Therefore, a node can *independently* finalize $\texttt{empty}_{200}$, after which the `chain` is confirmed to 204.

Fig. 7 shows why it is essential that a checking mode proposer can only finalize $\texttt{proposal}_{u_m}$ where $u_m = max(\mathbb{U})$. Suppose we remove this restriction, and we consider the following scenario. (1) P_{200} fails after broadcasting $\texttt{proposal}_{200}$, and only very few nodes received it: none of $P_{201}{\sim}P_{204}$ learns $\texttt{proposal}_{200}$. (2) The network is divided into two partitions A&B just before P_{204} broadcasting its `finalize` message; P_{204} is in partition A, so nodes in partition A confirm $\texttt{proposal}_{204}$ and confirm $\texttt{empty}_{200}$ (3) P_{205} and P_{206} are in partition B, so they timeout waiting for the `finalize` message for $\texttt{proposal}_{204}$ and mark the 204th block as undecided. (4) P_{205} learns $\texttt{proposal}_{200}$ from one node from $\mathbb{A}_{205}$ (who happens to be in the very few nodes) and carries $\texttt{proposal}_{200}$ with its `finalize` message, which is learnt by P_{206}. (5) P_{206} finalizes $\texttt{proposal}_{200}$ and causes inconsistency: nodes in partition A confirm $\texttt{empty}_{200}$, while nodes in partition B confirm $\texttt{proposal}_{200}$.

By restricting that a checking mode proposer can only finalize $\texttt{proposal}_{u_m}$ where $u_m = max(\mathbb{U})$, such inconsistency will not happen. This is because nodes in partition B must first finalize the $\texttt{empty}_{204}$ before finalizing $\texttt{proposal}_{200}$. However, since $\texttt{proposal}_{204}$ has already been delivered to a large portion of nodes, this inconsistency cannot happen.

Discussions. EGES does not let the acceptors validate transactions enclosed in $\texttt{proposal}_n$ because EGES may finalize blocks out of order. Specifically, if $\texttt{proposal}_n$ contains a money transfer transaction, and an acceptor has undecided blocks preceding n, the acceptor cannot predicate whether the source account has enough balance without a complete history. Therefore, EGES validates transactions when they are confirmed at each node because EGES always confirms blocks in order. An invalid transaction is deterministically (Section 3.2) discarded by all nodes, without impairing EGES's safety.

For the same reason, EGES does not force P_n to generate its $\texttt{proposal}_n$ within its SGX enclave because P_n cannot forge a validly signed transaction to affect EGES's safety. Overall, EGES achieve consensus on $\texttt{proposal}_n$ opaquely.

The drawback is that EGES may waste resources on achieving consensus on invalid transactions. However, as all invalid transactions are recorded on the blockchain, if an entity keeps injecting invalid transactions, it can be easily detected and penalized in a permissioned deployment.

Comparison with Algorand. EGES and Algorand both select a distinct committee for each block in an unpredictable way, and both use the delivery rate of a block proposal to confirm a block. Algorand leverages its built-in cryptocurrency to incentivize committee nodes to follow its protocol (i.e., proof of stake). However, even if one runs Algorand within SGX in a permissioned blockchain, there are still two major design differences making EGES more efficient than Algorand.

First, Algorand uses verifiable random functions (VRF) to determine committees, so it can only control the expected count of proposers for each block without an exact number (1∼70 in their experiment [1]). This design makes Algorand's consensus protocol not responsive [3,9]: informally, a responsive protocol lets nodes wait for a number of messages rather than a large amount of time in each protocol step, which ensures a good performance when the network is in good condition. For each block, Algorand selects 1∼70 proposers, and each proposer broadcasts a block proposal with a distinct priority level. Then, Algorand selects one of these proposals by letting nodes vote for the received proposal with the highest priority. Since the total number of proposers is unknown, each node must wait for a conservatively long time (e.g., 10s) before voting to ensure it has received most proposals. In contrast, EGES selects one proposer for each block, without the necessity for the selection progress, and EGES's protocol is responsive (Section 4).

Second, EGES adopts an optimistic design while Algorand adopts a pessimistic design. Specifically, Algorand uses a heavy step for both confirming a non-empty block and confirming an empty block. In contrast, EGES optimistically makes its gracious runs (i.e., confirming $\texttt{proposal}_n$) fast and shifts the burden of maintaining consistency to the rare failure cases (i.e., confirming $\texttt{empty}_n$).

4.4. Handling DoS attacks targeting proposers

In EGES, a proposer stays stealth before proposing its block, but its identity becomes explicit after broadcasting its proposal. If the proposer is DoS attacked at this time, this block cannot be finalized in time, impairing EGES's liveness.

To address this problem, we propose a new role of nodes called `arbiter`. An arbiter for a block index n does *not* generate new block proposals but only helps the proposer to finalize its proposal. For each block index n, the count of arbiters n_α is large than the attack budget B, and these arbiters do the same tasks to tolerate DoS attacks targeting them.

On receiving $\texttt{proposal}_n$, an arbiter for the n^{th} block broadcasts an `arbit` request following the same protocol as the proposer (Algorithm 1), and an acceptor responds to an `arbit` message with the same logic responding to a `propose` message.

Discussions. With the arbiters' help, a proposer's critical task is only to *send out* its `propose` request, and the arbiters help to finalize it. However, responding to both proposer and multiple arbiters makes acceptors targets of DoS attacks. Therefore, EGES lets normal nodes also randomly send fake (dummy) ACKs to cover the real acceptors (Algorithm 2 Line 4). Since real or fake ACKs are all encrypted with the receiver's public key, only the receiver's EGES enclave can decrypt them in the enclave and distinguish the real ones, so an attacker cannot know who are the real acceptors.

5. Security analysis

5.1. Safety with overwhelming probability

EGES ensures safety with overwhelming probability (i.e., $> 1 - 10^{-10}$).

Theorem 1 (*Safety*). *If an* EGES *node confirms a block b as the ith block on the blockchain, the probability that another* EGES *node confirms $b' \neq b$ as the ith block is $< 10^{-10}$.*

We prove the safety guarantee of EGES by induction: suppose EGES guarantees safety from the 0^{th} block to the $n-1^{th}$ block (hypothesis 1), and we prove that there is only one unique block that can be confirmed as the nth block among nodes in the blockchain. The base case is trivial because all nodes start from the same 0^{th} block.

Lemma 1. *if two nodes have the same maximum confirmed block in their `chain` (i.e., $\texttt{MC} = n-1$ due to hypothesis 1), then during consensus for the $(n+i)^{th}$ block where $i \geq 0$, as long as `MC` is not changed, these two nodes see the same member list.*

Proof. Proving this lemma is trivial if EGES works on a fixed member list, and we will show in Section 6.1 that EGES's protocol for dynamic memberships also ensures this lemma.

Lemma 2. *(Invariant 1 in Section 4.1): at most one proposal can be generated for the n^{th} block.*

Proof. This lemma is proved by two steps. First, as the proposer for the n^{th} block is encrypted in the $(n-lb)^{th}$ block, and the $(n-lb)^{th}$ block is the same among nodes because of hypothesis 1. Therefore, there is only one proposer (may have failed) for the n^{th} block. Second, this proposer generates at most one proposal, and non-proposer nodes cannot generate valid proposals for the n^{th} block because EGES's consensus module runs in SGX.

Proof of the induction step. In EGES, each block has only two choices (Lemma 2), and a confirmed block must be first finalized (Section 4.1). Therefore, it is sufficient to prove the following proposition 1: the probability that one node finalizes $\texttt{empty}_n$ (event A), and another node finalizes $\texttt{proposal}_n$ (event B) is overwhelmingly small.

For event A, suppose node X finalizes $\texttt{empty}_n$. We use f_{m_X} to denote the maximum finalized block index on node X. Consider blocks with indices in $[n+1, f_{m_X}]$. Since EGES finalizes *empty* blocks in descending order (Algorithm 2 Line 12~15), there are no undecided blocks in $[n+1, f_{m_X}]$, and we can have another level of induction by supposing blocks finalized as empty in $(n+1, f_{m_X}]$ are finalized consistently (name it hypothesis 2). For event B, $\texttt{proposal}_n$ can be finalized either by P_n (call it event B1) or by subsequent proposers that have learnt this proposal (event B2).

First, we prove that the probability that event A and event B1 happen together is overwhelmingly low. Suppose that a portion p of all M EGES nodes received and cached the $\texttt{proposal}_n$, and we calculate the probability for event B1. We use `Re` to denote the number of acceptors for the n^{th} block that REceived $\texttt{proposal}_n$. Since $\texttt{proposal}_n$ is broadcasted in EGES's P2P network and the stealth acceptors are selected uniformly, `Re` follows hypergeometric distribution $\texttt{Re} \sim H(M, n_A, p \times M)$. Thus, the probability that P_n finalizes $\texttt{proposal}_n$ is

$$Prob(B1) = Prob(\texttt{Re} > \tau \times n_A)$$

We then calculate the probability of event A. Event A infers that after the n^{th} block, there are at least D non-empty blocks that are finalized and carrying n in the undecided list. This means each proposer of these D blocks received $(\tau \times n_A)$ ACKs from their acceptor group, and none of these acceptors sending those ACKs has received $\texttt{proposal}_n$. For each of the D blocks, the number of acceptors `NR` not receiving $\texttt{proposal}_n$ follows hypergeometric distribution $\texttt{NR} \sim H(M, n_A, (1-p) \times M)$.

Therefore, the probability of event A is

$$Prob(A) = (Prob(\texttt{NR} > \tau \times n_A))^D$$

The calculation shows that the probability of event A and event B1 happening together $Prob(A) \times Prob(B1)$ is overwhelmingly low for any delivery rate p by setting τ and n_A (Section 7.3). For instance, our evaluation chose τ as 59%, D as 4, n_A as 300, M as 10K, and the probability of EGES enforcing safety is $1 - 10^{-9}$. In real deployments, M may change due to membership changes; however, when M is much bigger (e.g., 20X) than n_A, this probability is not sensitive to M because hypergeometric distribution is approximate to binomial.

For the second step, we prove that event A and event B2 cannot happen together. For event B2, we suppose that proposer P_i, where $i > n$, learns and finalizes $\texttt{proposal}_n$. We discuss by comparing i and f_{m_x} and derive contradictions. If $i \le f_{m_x}$, hypothesis 2 infers that X did not finalize $\texttt{empty}_i$, so $\texttt{proposal}_n$ is finalized together with $\texttt{proposal}_i$ at node X, contradicting to event A. Else if $i > f_{m_x}$, since a proposer can only finalize the maximum index in its local $\mathbb{U}$ list (Algorithm 1 Line 9), for node P_i we can predicate that blocks with index in $[n+1, i)$ are finalized. Due to hypothesis 2, blocks within $[n+1, f_{m_x}]$ are finalized the same as node X, and therefore P_i should also finalize the n^{th} block as empty, causing contradiction.

Putting the two steps together, we proved proposition 1 and thus proved the induction step that the n^{th} block must be confirmed consistently among nodes with overwhelmingly high probability. Therefore, EGES ensures safety with overwhelmingly high probability.

5.2. Liveness under targeted DoS attacks

EGES can defend against DoS attacks and partition attacks targeting committee nodes under the threat model in Section 3.2. Since from a single node's point of view, it cannot distinguish whether a remote node is under DoS attack or is partitioned, EGES handles these two attacks altogether.

EGES has three types of committee nodes for each block, a proposer, n_A acceptors, and n_α arbiters, randomly selected from all nodes in the system. Because of EGES's stealth committee abstraction (Section 4.2), the identity of each committee node is unknown to attackers outside SGX enclaves before the node sending out its first protocol message.

Lemma 3. *For any block index n, a DoS attacker cannot distinguish whether an* EGES *node is a real acceptor or a fake acceptor for this nth block by observing from outside* EGES *enclaves.*

Proof. This lemma is proved by two steps. First, the identities of real EGES acceptors for the nth block (i.e., $\mathbb{A}_n$) is unknown to the attacker before each acceptor takes its role and sends out its first protocol message. This is because $\mathbb{A}_n$ are randomly selected by P_{n-lb}, inside its enclave for integrity and confidentiality; these identities are secretly transferred to each real acceptor's enclave (Section 4.2), so that only each acceptor's enclave knows whether the node is in $\mathbb{A}_n$.

Second, an acceptor sends ACK messages to both the proposer and arbiters, and its identity becomes explicit after sending out its first ACK message, so EGES uses fake acceptors to conceal the real ones. If observed from outside enclaves, the fake acceptors behave identically as real ones so an attacker cannot differentiate the real acceptors and attack them. EGES achieves this with three design points. (1) fake acceptors are randomly selected from all nodes for each block so that an attacker cannot determine the fake acceptors by monitoring network packets (Section 4.2). (2) Real and fake acceptors' EGES enclaves respond to protocol messages (propose and arbit) in the same way if observed outside the enclaves (Section 4.3), so an attacker cannot distinguish real or fake acceptors by watching their behaviors. (3) All messages from real or fake acceptors have the same format, encrypted with the receiver's public key (Section 4.3) and can only be decrypted in the receiver's enclave, so an attacker cannot differentiate real acceptors from fake ones by watching the packet content.

Lemma 4. *For each block index n, with the attack model defined in Section 3.2, a DoS attacker cannot make* EGES *fail to achieve consensus on* $\texttt{proposal}_n$ *by targeting the proposer* P_n.

Proof. This lemma is proved by two steps. First, same as the first step in the proof for Lemma 3, P_n's identity is unknown to the attacker before sending out $\texttt{proposal}_n$. Second, the identity of P_n becomes explicit after broadcasting its $\texttt{proposal}_n$ and may be targeted attacked when it is waiting for quorum ACKs. However, since EGES has many (i.e., $n_\alpha > B$) arbiters that can help to finalize the $\texttt{proposal}_n$, attacking P_n will not affect EGES's liveness. Note that targeting the arbiters is not viable either as the total number of arbiters n_α is set to be larger than the attack budget B; as long as one arbiter is not under targeted DoS attack, it can finalize the current n^{th} block.

Theorem 2 (*Targeted DoS Resistance*)**.** *For any block index n, the probability for an DoS attacker with budget B (defined in Section 3.2) to succeed in preventing* EGES *from achieving consensus on* $\texttt{proposal}_n$ *by targeting* EGES*'s committee is* $P(X > (1-\tau) \times n_A)$*, where* $X \sim Hypergeom(n_A + n_F, n_A, B)$*, and* n_F *is the number of fake acceptors defined in Section 4.2.*

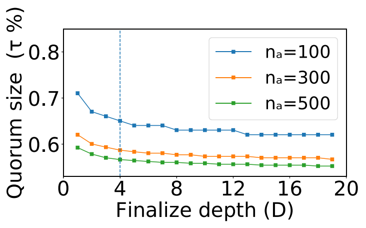


Fig. 8. Parameter selection for τ and D on 10K nodes for different n_A values.

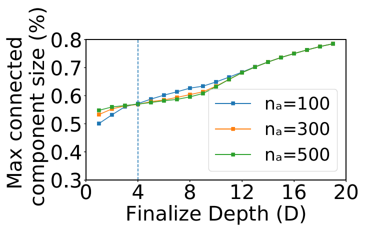


Fig. 9. Connected component size required to ensure liveness with different D values.

Proof. Due to Lemma 4, a smart attacker will target EGES's acceptors instead of wasting its budget on proposers or arbiters in the committee. However, due to Lemma 3, the attacker can only randomly select B targets from the $n_A + n_F$ real and fake acceptors. To successfully mount targeted DoS attacks, the attacker must guess at least $(1-\tau) \times n_A$ real acceptors, and the probability is calculated as in the theorem.

Theorem 2 shows that the probability of a successful targeted DoS attack can be limited very small by controlling the ratio of fake and real acceptors. For instance, if EGES has $n_A = 300$, $\tau = 59\%$ and (expected) 600 fake acceptors; if the attacker's attack budget $B = 300$, the probability that the attacker can luckily attack more than $(1-\tau) \times n_A$ real acceptors for one block is 0.3%. Moreover, even if the attacker is so lucky that it successfully guessed more than $(1-\tau) \times n_A$ acceptors for some block (say n^{th}), EGES can still consistently determine whether to confirm $\texttt{proposal}_n$ or $\texttt{empty}_n$ using the committees for subsequent blocks (Section 4.3); in other words, the attacker must constantly be so lucky to make EGES lose liveness. Overall, EGES has a strong resistance to targeted DoS resistance. We will also quantify EGES's ability to tolerate ubiquitous (non-targeted) DoS or network partition attacks in Section 5.3.

DoS attacks targeting network links. Note that EGES's targeted attack model (Section 3.2) handles only DoS or partition attacks targeting specific EGES nodes. A more powerful attacker may also target EGES's major communication links. From the protocol aspect, EGES mitigates such vulnerabilities in the Network layer (in the OSI model [86]) by using different committees for different blocks: EGES's protocol traffic is spread among the whole P2P network rather than centralized among a few dedicated nodes. However, when EGES is deployed, EGES's communication messages may be aggregated in the Link or Physical layer. For instance, if a large number of EGES nodes are hosted in the same data center (DC), the links connecting this DC and the Internet may be susceptible to attacks. Fortunately, such attacks are usually easier to detect by the administrator of a permissioned blockchain. Moreover, such attacks are not adaptive, and as long as a great majority of nodes are connected, EGES can achieve practical liveness. Section 7.3 shows the relation between EGES's liveness and the maximum connected component size in the P2P network. Nevertheless, EGES cannot ensure liveness under arbitrary partitions, and previous work shows that it is theoretically impossible to guarantee both consistency and liveness under partitions [28,87], so EGES choose to ensure consistency.

5.3. Parameter selection

EGES has two correlated parameters τ and D. Fig. 8 shows the relation between τ and D to ensure EGES's safety. With a smaller τ, a proposer P_n can finalize $\texttt{proposal}_n$ after collecting fewer ACKs from acceptors, so subsequent proposers need more rounds of checking (larger D) when trying to finalize the $\texttt{empty}_n$ (Section 4.3).

The τ and D values also affect EGES's ability to achieve liveness (confirm non-empty blocks) on network partitions (or ubiquitous DoS attacks). We quantify this ability to the "minimum largest connected component size" (cc%) required in the P2P graph, provided that nodes in a connected component can reach each other before a timeout. A smaller cc means that EGES is more robust to partition and ubiquitous DoS attacks. From a mathematical aspect, as long as the probability of finalizing a proposal is non-zero, the probability p_D that D consecutive proposals are successfully finalized is always larger than zero, inferring that *eventually* EGES can achieve liveness. However, we conservatively calculate the required cc to make the p_D larger than 5% for practical liveness, as shown in Fig. 9. In our evaluation, we chose τ as 59%, D as 4, n_A as 300, which ensures both safety and achieves good liveness on partition attacks (Section 7.2).

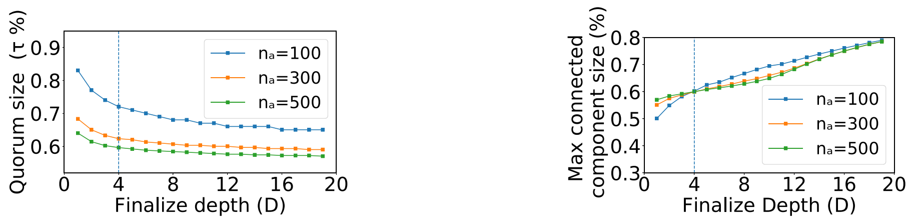


Fig. 10. Parameter selection and liveness requirements if EGES lets each node to independently decide its committee membership.

Fig. 10 shows the parameter selections if EGES does not use its stealth committee mechanism, but lets each node *independently* determine whether it is an acceptor with the probability of M/n_A, with n_A being the *expected* number of acceptors for each block. If EGES makes such a design choice, the Re (Section 5.1) becomes a binomial distribution with the probability of $p \times M/n_A$, and other distribution changes similarly. As shown in Fig. 10, EGES would need a larger quorum size $\tau \times n_A$ and achieve worse liveness on network partition.

6. Implementation

We selected the Golang implementation of Ethereum (i.e., geth) as our codebase because geth is heavily tested on the Internet. We leveraged the P2P libraries from geth and rewrote the functions for generating, verifying, and handling new blocks. Since SGX only provides SDKs in C/C++, we used CGo to invoke ECalls. We modified 2073 lines of Golang code and implemented the consensus protocol for 1943 lines of C code. For asymmetric key-based encryption, we used ECC-256 from the API provided by the SGX SDK. For timeouts, we used the trusted timer API *sgx_get_trusted_time* provided by the SGX platform

Each EGES node has three modules: a consensus module running the EGES consensus protocol and storing nodes' member list (Section 4), a P2P module connecting to a random set of peers and relaying messages using the Gossip [76] protocol, and a blockchain core module storing confirmed parts of `chain` and client transactions. Only the consensus module runs in the node's SGX enclave.

In EGES, a node may finalize a block before knowing its preceding blocks. Therefore, when an EGES proposer proposes a block or a node finalizes a block, it lefts the block's field of "hash of the previous block" empty, and EGES's enclave computes this field when confirming the block. In essence, EGES achieves consensus on a totally ordered sequence of transactions, same as Hyperledger Fabric [2], and encapsulates these batches into a hash-chain of blocks while confirming them.

6.1. *Dynamic membership and attestation*

To support dynamic memberships, EGES leverages the idea from SCIFER [44] to record the joining of new nodes as transactions on the blockchain. This mechanism ensures Lemma 1 because all updates to the member list are only determined by confirmed blocks.

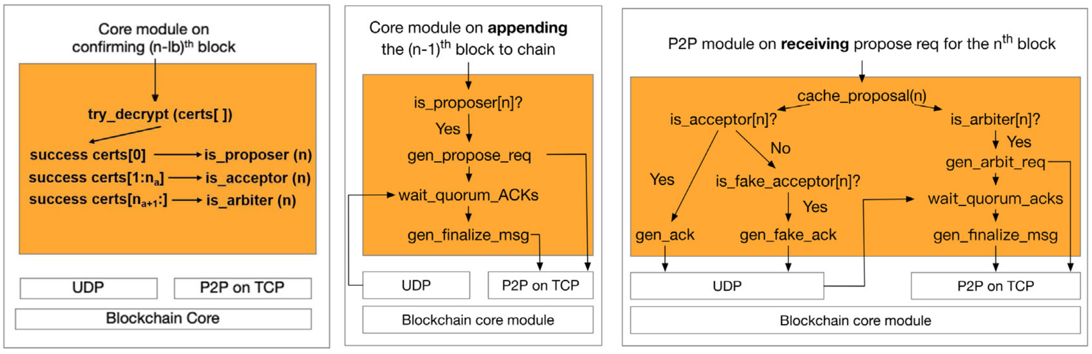
When a node i wants to join the system, it needs to find a member node j to do attention (Section 2) through an out-of-band peer discovery service. We assume that node i knows the genesis (0^{th}) block, so node i can inductively verify the blockchain, without relying on whether peer j is malicious.

A node i joins EGES with three steps. First, i launches its EGES enclave, which generates its account (pk_i, sk_i) and creates the hardware monotonic counter c for defending forking attacks (Section 6.2). The node's account is securely saved to permanent storage using SGX's seal mechanism [22] for recoveries from machine failures (e.g., power off). Then, i sends a *join* request to j. Second, j does a standard SGX remote attestation [22], which succeeds with a signed quote Q_i from Intel's attestation service, and i's enclave transfers its public key pk_i and counter value c to j's enclave through the secure communication channel between two enclaves created during attestation. Third, node j's enclave creates a signed registration transaction including pk_i, c, $addr_i$, Q_i and i's ip address. Node i joins EGES when the transaction is included in a confirmed block.

6.2. *Enclave interactions*

Fig. 11 shows the implementation of EGES enclave. An EGES enclave holds three data structures shared among ECalls: `cache`, `is_proposer`, and `is_acceptor`, each as a hash map. As explained in Section 4.1, the `cache` saves received block proposals. In our implementation, the `cache` only keeps the hash values of block headers instead of whole blocks to save enclave memory. `is_proposer` and `is_acceptor` are hash maps with block indices as keys and boolean as values, saving whether this node will be in the committee for a future block.

In Fig.11a, when a node's blockchain core module confirms the $(n - lb)^{th}$ block, it asynchronously invokes an ECall letting the enclave check whether it will be the proposer/acceptor for the n^{th} block (Section 4.2). In Fig.11b, when a



(a) On confirming a block. (b) On appending a block. (c) On receiving a `propose` request

Fig. 11. EGES's enclave interactions (ECalls and OCalls). The enclave module is shaded in orange.

Table 2
EGES's evaluation parameters.

Config	# Nodes	Acceptor group size (n_A)	D	τ	LB	*timeout*	SGX mode
Cluster	300	100	4	65%	5000	2 s	Hardware mode
AWS Cloud	Up to 10 K	300	4	59%	10 000	3 s	Simulation mode

node's core module appends the $(n-1)^{th}$ block, it invokes an ECall with a batch of transactions, and the enclave will follow the protocol in Algorithm 1 if it is the proposer for the n^{th} block. In Fig.11c, when a node's P2P module received a `propose` request, it invokes an ECall passing this request to the enclave, and the enclave will generate an ACK that will be sent through UDP using an OCall if it is an acceptor (Algorithm 3) or fake acceptor (Algorithm 2 Line 4). If it is an arbiter for the n^{th} block, it will also start working as an arbiter (Section 4.4).

Enclave forking attacks. In the P2P scenario, one challenge is enclave forking attacks [88]. EGES must permit a node to reuse its sealed account (Section 6.1) in case the node restarts its machine. However, a malicious node can create multiple copies of EGES enclaves with the same account *pk*, directs different messages to them, and lets them generate conflicting messages (e.g., block proposals). Existing defending techniques [88,89] work in the client–server manner, where clients attest and communicate to only a single server. These techniques are not suitable for P2P settings because they will need every two EGES nodes to connect and attest each other.

EGES defends such attacks using SGX's platform counter [22], which is monotonic among all enclaves on the same machine. When a node launches its EGES enclave, the enclave increments and read this counter value *c* and enclose this *c* to its registration transaction: the node's membership is bound to the enclave with counter value *c* but not the account *pk*. When the enclave sees a registration with the same account but a higher counter value, it quits automatically.

7. Evaluation

Evaluation setup. Our evaluation was done on both our own cluster with 30 machines and the AWS cloud, with parameters shown in Table 2. In our cluster, each machine has 40 Gbps NIC, 2.60 GHz Intel E3-1280 V6 CPU with SGX, 64 GB memory, and 1TB SSD. On AWS, we started up to 100 c5.18xlarge instances (VMs) running in the same city, each of which has 72 cores, 128 GB memory, and up to 25 Gbps NIC. We ran up to 100 EGES nodes on each VM (10k nodes in total), with each EGES node running in a docker container.

To evaluate EGES and baseline protocols in a geo-replicated setting, while running EGES on both our cluster and AWS, we emulated the world scale Internet by using the Linux traffic control (TC) to limit the RTT between every two nodes to a random value between 150 ms and 300 ms. These settings are comparable to Algorand's setting on AWS. As AWS does not provide SGX hardware, we ran EGES in the SGX simulation mode on AWS and in the SGX hardware mode on our cluster; we show that EGES's performance in simulation mode is roughly the same as hardware mode because EGES's performance is bound to network latency in WAN (Section 7.1). The scalability (Fig. 12) and robustness (Fig. 13) experiments were done on AWS, and the rest were in our cluster.

We evaluated EGES with nine consensus protocols for blockchain systems, including five state-of-the-art efficient BFT protocols for permissioned blockchains (BFT-SMaRt [7], SBFT [8], HoneyBadger [32], and HotStuff [9]), two SGX-powered consensus protocols for permissioned blockchains (Intel-PoET [26] and MinBFT [33]), the default consensus protocol in our codebase (Ethereum-PoW [31]), and two permissionless blockchains' protocol that runs on dynamic committees (Algorand [1] and Tendermint [18]). A detailed description of these protocols is in Section 2.2.

Since Algorand's open-source code is under development, and we were unable to deploy its latest release [90] to the same scale as EGES and Algorand's paper (i.e., 10k nodes), we took Algorand's performance from Figure 5 in its paper [1].

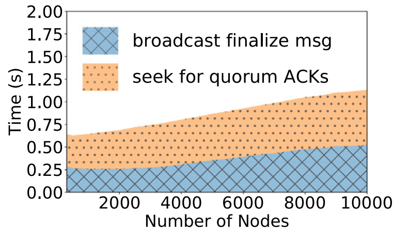


Fig. 12. Scalability to the number of nodes on the Internet.

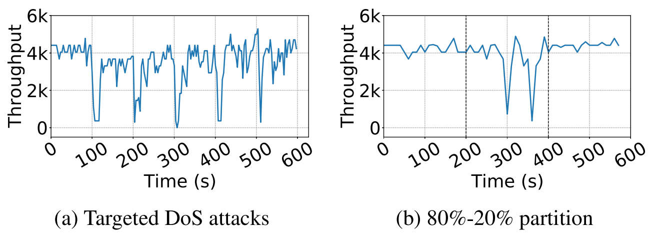


Fig. 13. EGES's throughput on DoS and network partition attacks. There were 1000 nodes on AWS at 0 s.

To make the comparison fair, we make EGES's network setting more rigorous than Algorand's: Algorand divided nodes into multiple cities where intra-city packets have negligible latency, while EGES lets the RTT among any two nodes be at least 150 ms.

For all evaluated protocols, we measured their performance when each of them reached peak throughput. For an apple-to-apple comparison of latency, we adopted Algorand's method to measure transactions' server-side confirmation time: from the time a transaction is first proposed by a committee node to the time the transaction is confirmed at this node, excluding the time for clients' transaction submissions. We measured the server-side instead of the client-side latency because this method precludes the disturbance of client behaviors as these protocols run on different blockchain frameworks. For instance, in Ethereum, PoET (running on Hyperledger Sawtooth [26]), and EGES, a client submits a transaction to a random node, and the transaction is disseminated via P2P networks; in BFT-SMaRt, a client submits transactions to a fixed node (i.e., the leader); in Algorand, a consensus node directly packs a block with a fixed amount of data (e.g., 1MB) instead of using separate transactions.

We set EGES's transaction size to 250 bytes, a typical transaction size for general data-sharing applications [61,91]. Since Algorand reported throughput on block size, we convert it to txn/s by assuming the same size of transactions as EGES's. The transaction sizes for the other eight baseline protocols are either equal to or smaller than that of EGES. Our evaluation focuses on these questions:

Section 7.1: Is EGES efficient and scalable?
Section 7.2: What is EGES's performance under DoS attacks?
Section 7.3: How sensitive is EGES to its parameters?
Section 7.4: How do EGES performance and fault tolerance compare with notable BFT protocols?
Section 7.5: What are the limitations and future work of EGES?

7.1. Efficiency and scalability

Table 1 shows the performance comparison of EGES and eight baseline protocols. As Alogrand's paper [1] evaluated at least 2K nodes, we postpone the comparison between EGES and Algorand to when we evaluated EGES's scalability.

Overall, in the geo-replicated setting, EGES achieved comparable performance to MinBFT, Tendermint, HotStuff, and SBFT. We ran BFT-SMaRt in its default setting (ten nodes), and it showed higher throughput and lower latency than EGES. BFT-SMaRt is more suitable for small-scale permissioned blockchains where a few companies run nodes in a controlled environment, so it lets nodes send messages to each other directly. In contrast, EGES is designed for tolerating targeted DoS attacks on committee nodes, so it has two P2P broadcasts to confirm a block. Section 7.4 shows that EGES's scalability and fault tolerance are better than BFT-SMaRt.

SBFT and HotStuff had a lower throughput and a higher latency than EGES. They rely on designated nodes to collect the consensus messages that were originally all-to-all broadcasted and to distribute a combined message to all nodes. Although this approach improves scalability, it also incurs two more RTTs, limiting their performance in a geo-distributed deployment. Moreover, an attacker targeting these designated nodes will cause a dramatic performance drop to the system, which is evaluated in Section 7.4.

Table 3
Proposer's micro-events for finalizing a block.

blk size	txns/blk	# ECalls	CPU usage	Network usage
750 kB	3000	97	12.4%	15.53 Mbps

HoneyBadger showed a lower throughput and a higher latency than EGES because HoneyBadger uses multiple rounds of broadcasts for a single block, which incurred a long latency in a geo-distributed setting. EGES showed orders of magnitude better performance than PoET and Ethereum, two PoW protocols. Their performance is limited by the time for solving PoW puzzles (or sleep time) and the number of blocks to wait for before confirming a block (Section 2.3). Our evaluation result for PoET is similar to a recent study [92].

Breakdown and micro-events. To understand EGES's latency, we recorded the time taken for the two steps of EGES's protocol (Section 4.3): seeking for quorum ACKs took 576 ms; broadcasting `finalize` messages took 329 ms. The first step took a longer time because EGES broadcasts the proposed block in its P2P network in this step. This P2P broadcast time is essential in any blockchain system because new blocks need to be broadcasted to all nodes.

SGX's overhead. Table 3 shows the micro-events of EGES. The ECall column shows the number of times that EGES's proposer node entered its SGX enclaves on finalizing a block. Since each ECall only takes around 3us [22], and EGES's proposer only did 97 ECalls on average for each block, running in SGX hardware mode and simulation mode makes little difference for EGES's performance.

Scalability. To evaluate EGES's scalability, we ran 100–10000 nodes on AWS and evaluated its confirm latency with the same block size as in the cluster evaluation. Fig. 12 shows the result. The latency is divided into two parts. The figure shows that the seeking for quorum ACKs phase of EGES (Section 4.3) is the dominant factor because it broadcasts the proposed 750 kB block on the P2P network. Fortunately, a P2P broadcast latency is proportional to approximately the *log* of the number of nodes [93], indicating EGES's reasonable scalability. The increase rate was slightly greater than the log scale because 100 nodes were run in one VM with CPU and NIC contentions. EGES's latency on AWS was slightly faster than on our cluster, because AWS CPUs are faster.

Compared to Algorand's performance in Table 1, EGES showed 2.3X higher throughput and 16.8X faster latency than Algorand. This is due to two reasons. First, Algorand's VRF-based method selects multiple proposers for each block, and Algorand uses a reduction step to select one proposal by these proposers. Moreover, as the VRF-based approach cannot control the exact number of proposers, nodes must wait for a conservatively long time in the reduction step (Section 4.3). In contrast, EGES's stealth committee abstraction selects one proposer for each block, without the need for such a reduction step. Second, EGES's consensus protocol has only two rounds in gracious runs to confirm a block (Section 4.3).

7.2. Performance on DoS attacks

To evaluate EGES's robustness under DoS attacks, we ran EGES with 1000 nodes on AWS with $n_A = n_\alpha = 100$, and conducted targeted DoS attacks that are compliant with our attack model (Section 3.2): we assumed the attacker's budget $B = 10\%$ of total nodes, and we set the expected count for fake acceptors and arbiters to be 200. Each time we targeted the current proposer and 99 arbiters or real/fake acceptors (because we cannot distinguish real acceptors). For each attack, we blocked all communication from the attacked nodes for 20 s.

We deem such attacks to be powerful enough, as no existing protocols for permissioned blockchain can maintain liveness under such powerful attacks. As shown in Section 7.3, existing consensus protocols, which ran on static committee nodes, lost liveness *until* the DoS attack ended. In contrast, each time after we attacked 100 nodes, EGES's throughput had a temporary drop and recovered *before* the DoS attack ended, which shows that EGES can ensure practical liveness under such powerful attacks.

After the first attack, the line started to go up after 11.3 s, much slower than the other attacks (about 3.1 s). We inspected the log and found that the slow recovery was because the proposer for the next block happened to be attacked, and EGES waited until D more blocks to confirm that block as empty. After the second attack, the line took about 7.2 s to go up. This is because most real acceptors happened to be attacked together with the proposer, which makes the arbiters failed to finalize the block for the proposer (Section 4.4). For the other three attacks, the arbiters successfully helped corresponding proposers to finalize their blocks quickly.

To evaluate EGES performance on network partitions, we manually divided the network into two partitions at 200 s and reconnected them at 400 s, with one partition containing 80% nodes and the other containing 20% nodes. Fig. 13b shows the throughput measured in the large partition. Overall, the large partition maintained liveness during the partition. The small partition did not succeed in confirming any block during the partition and caught up after the network reconnected, preserving safety. There are two obvious throughput drops in the figure, which are caused by the pre-designated proposers being in the small partition, and EGES confirmed empty blocks for them. Note that EGES may temporarily lose liveness in catastrophic partitions (e.g., 50–50 or 40-30-30 partitions) but can preserve safety. Section 7.3 shows a quantitative analysis of how EGES can preserve liveness under network partitions.

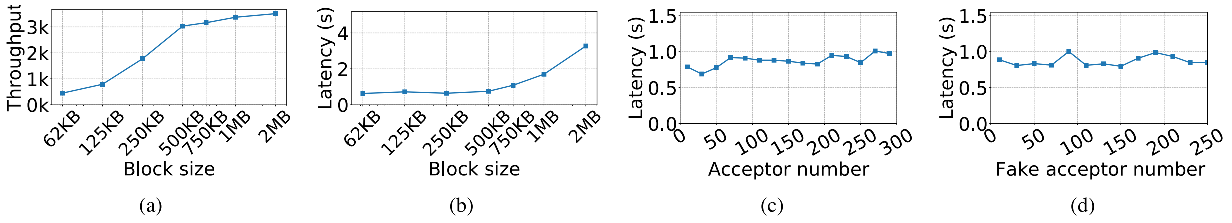


Fig. 14. Sensitivity on block size, acceptor numbers, and expected fake acceptor numbers (cluster setting).

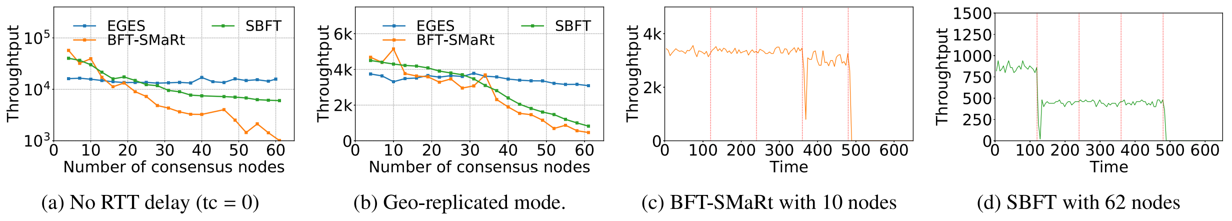


Fig. 15. Comparing EGES, SBFT, and BFT-SMaRt.

7.3. Sensitivity

Figs. 14a and 14b show EGES's performance sensitivity on block size. When the block size was larger, EGES's throughput did increase, but its block confirm latency also increased. In our evaluation, we set EGES's block size to be 750 kB, which is a near-optimal setting for both throughput and latency.

EGES's throughput and confirm latency depend on three important protocol parameters, the number of acceptors, the number of fake acceptors, and block size. Figs. 14c and 14d show the sensitivity results. EGES's performance turns out to be insensitive to the first two parameters because the latency is dominated by the time for broadcasting the new block on the P2P network.

7.4. Comparison to BFT-SMaRt and SBFT

Since BFT-SMaRt with 10 (committee) nodes was faster than EGES with 100 acceptors, we evaluated both of them on a different number of nodes because more such nodes can tolerate more faults and DoS attacks. Fig. 15a shows the results using the same setting for both systems (e.g., in our cluster, TC disabled, and the same number of transactions in each batch). Overall, EGES throughput was stable because the number of acceptors affects little on the latency in the seeking for quorum ACKs phase. BFT-SMaRt's throughput drops dramatically because its protocol involves a quadratic number of messages on the number of ordering nodes.

Fig. 15a also shows SBFT's performance. In the non-geo-replicated mode, when the number of nodes increased from 4 to 62, SBFT's throughput dropped from 38.2K to 6.9K transactions/s. This is because SBFT's collectors (Section 2.2) need to collect more messages and to verify their signatures, so the time spent in collectors increased from 2.5 ms to 13.1 ms.

Fig. 15b shows the performance comparison of EGES, BFT-SMaRt, and SBFT in the geo-replicated setting. EGES's throughput was at least 3.4X larger than both systems on 62 nodes. BFT-SMaRt's performance trend was similar to the no-delay setting because of its PBFT all-to-all broadcasted messages. SBFT's throughput also dropped dramatically because some nodes became stragglers for the collectors due to the varied RTT. Since SBFT's fast path can only tolerate a small number of straggler nodes (usually two (Section 2.2)), we observed that 87% of the consensus rounds in SBFT have reverted to the slow path (PBFT).

More importantly, EGES can safely switch its acceptor groups across blocks, and it tolerated various failure scenarios, including DoS attacks (Fig. 13a). For comparison, we evaluated the performance of BFT-SMaRt and SBFT on node failures (i.e., DoS attacks targeting consensus nodes). Fig. 15c shows the result of BFT-SMaRt with its default 10-node setting. We randomly killed one node on each vertical line. The third time we killed its leader coincidentally, so there was a noticeable performance drop. BFT-SMaRt's throughput dropped to zero after we killed the fourth node. For SBFT (Fig. 15d), we started with 62 nodes and killed 7 nodes every time. Since SBFT's fast path can only tolerate two crashed or straggler nodes, its throughput dropped significantly (reverted to PBFT) after the first kill.

Overall, EGES is complementary to BFT-SMaRt and SBFT: BFT-SMaRt is the fastest in a small scale; SBFT has better scalability, but its high performance requires a synchronous network (stated in their paper). EGES achieved reasonable efficiency and DoS resiliency in a geo-replicated setting.

7.5. Discussions

EGES has two limitations. First, EGES requires each node to have an SGX device. We deem this requirement reasonable because SGX is available on commodity hardware, and both academia and industry are actively improving the security of SGX. Recent permissionless [24] and permissioned blockchains [23,26,33] also use SGX. Second, EGES targets Internet-scale permissioned blockchain systems (e.g., a global payment system [3]), while for small-scale deployments (e.g., supply chain among a few small companies), existing consensus protocols (e.g., BFT-SMaRt) are more suitable.

One device may want to join multiple instances of EGES for different applications. As shown in Section 7.1, EGES is network-bounded instead of CPU-bounded. Therefore, the co-running multiple instances of EGES will not affect each others' performance if the bandwidth of this device is not the bottleneck. More importantly, thanks to EGES's adoption of TEEs on each member node, multiple applications can co-run on the same EGES instance. Specifically, a major concern for multiple applications to share the same blockchain stems from data access isolation [94,95]. Fortunately, in EGES, since each EGES node is equipped with an attested EGES enclave, EGES can leverage existing work (e.g., Ekiden [25]) to provide data privacy among these co-running applications.

Our paper reveals that, in addition to safety and high performance, DoS resistance is also an essential evaluation metric for a practical Internet-scale blockchain application, including e-voting [96], decentralized auction [97], and payment systems [3]. Moreover, the attested SGX enclave on each EGES node brings the potential to port existing centralized SGX-powered applications [35,67,68,98] onto EGES and to make them DoS-resistant. For instance, ToR [99] is a popular anonymous network and is widely used for providing client anonymity to blockchain systems [100,101]; SGX-ToR [35] greatly improves the security and privacy of ToR by leveraging SGX. However, SGX-ToR relies on a few directory servers for maintaining the list of attested nodes (relays), which has been shown [36] to be susceptible to DoS attacks. By deploying SGX-ToR's directory service as a blockchain application on EGES, SGX-ToR can be made DoS-resistant.

8. Conclusion

We have presented EGES, the first efficient permissioned blockchain consensus protocol that can tolerate targeted DoS and partition attacks. EGES achieves comparable performance to the existing fastest permissioned blockchain's consensus protocols while achieving much stronger robustness. Our evaluation reveals that, in addition to safety and performance, DoS resistance should also be an essential evaluation metric for a blockchain system deployed on the Internet. EGES's source code is available on github.com/hku-systems/eges.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, N. Zeldovich, Algorand: Scaling Byzantine Agreements for Cryptocurrencies, Cryptology ePrint Archive, Report 2017/454, 2017, Accessed: 2017-06-29.
[2] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, et al., Hyperledger fabric: a distributed operating system for permissioned blockchains, in: Proceedings of the Thirteenth EuroSys Conference (EuroSys 2018), ACM, 2018, p. 30.
[3] M. Baudet, A. Ching, A. Chursin, G. Danezis, F. Garillot, Z. Li, D. Malkhi, O. Naor, D. Perelman, A. Sonnino, State machine replication in the libra blockchain, 2019.
[4] S. Nakamoto, Bitcoin: A peer-to-peer electronic cash system, 2008, https://bitcoin.org/bitcoin.pdf (Accessed: 2015-07-01).
[5] C. Hou, M. Zhou, Y. Ji, P. Daian, F. Tramer, G. Fanti, A. Juels, Squirrl: Automating attack analysis on blockchain incentive mechanisms with deep reinforcement learning, 2019, arXiv preprint arXiv:1912.01798.
[6] D. Harz, L. Gudgeon, A. Gervais, W.J. Knottenbelt, Balance: Dynamic adjustment of cryptocurrency deposits, in: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, 2019, pp. 1485–1502.
[7] J. Sousa, A. Bessani, M. Vukolić, A byzantine fault-tolerant ordering service for the hyperledger fabric blockchain platform, in: IEEE/IFIP International Conference on Dependable System and Network (DSN 2018), 2018.
[8] G.G. Gueta, I. Abraham, S. Grossman, D. Malkhi, B. Pinkas, M.K. Reiter, D.-A. Seredinschi, O. Tamir, A. Tomescu, Sbft: a scalable decentralized trust infrastructure for blockchains, in: IEEE/IFIP International Conference on Dependable System and Network (DSN 2019), 2019.
[9] M. Yin, D. Malkhi, M.K. Reiter, G.G. Gueta, I. Abraham, Hotstuff: bft consensus with linearity and responsiveness, in: Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, ACM, 2019, pp. 347–356.
[10] Medical chain, 2017, http://www.medicalchain.org/.
[11] Blockchain for supply chain, 2017, https://www.ibm.com/blockchain/industries/supply-chain.
[12] L. Lamport, R. Shostak, M. Pease, The byzantine generals problem, 4, (3) ACM, 1982, pp. 382–401,
[13] C. Natoli, V. Gramoli, The balance attack against proof-of-work blockchains: The R3 testbed as an example, 2016.
[14] E. Heilman, A. Kendler, A. Zohar, S. Goldberg, Eclipse attacks on bitcoin's peer-to-peer network, in: 24th USENIX Security Symposium (USENIX Security 15), 2015, pp. 129–144.
[15] S. Tochner, A. Zohar, S. Schmid, Route hijacking and dos in off-chain networks, in: Proceedings of the 2nd ACM Conference on Advances in Financial Technologies, 2020, pp. 228–240.
[16] O. Kupreev, E. Badovskaya, A. Gutnikov, Ddos attacks in q2 2019, 2019.
[17] A. Rayome, Major ddos attack lasts 297 hours, as botnets bombard businesses, 2018.

[18] E. Buchman, Tendermint: Byzantine fault tolerance in the age of blockchains, 2016, http://atrium.lib.uoguelph.ca/xmlui/bitstream/handle/10214/9769/Buchman_Ethan_201606_MAsc.pdf (Accessed: 2017-02-06).
[19] S. Pyo Kim, Tenderand: Randomized leader election in tendermint, 2020.
[20] E.K. Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, B. Ford, Enhancing bitcoin security and performance with strong consistency via collective signing, in: 25th USENIX Security Symposium (USENIX Security 16), USENIX Association, Austin, TX, 2016.
[21] P. Jovanovic, Byzcoin: Securely scaling blockchains, 2016, http://hackingdistributed.com/2016/08/04/byzcoin/ (Accessed: 2019-08-01).
[22] V. Costan, S. Devadas, Intel sgx explained, IACR Cryptol. ePrint Arch. 2016 (2016) 86.
[23] CCF: A Framework for Building Confidential Verifiable Replicated Services, Tech. Rep. MSR-TR-2019-16, Microsoft, 2019.
[24] F. Zhang, I. Eyal, R. Escriva, A. Juels, R. van Renesse, Rem: Resource-efficient mining for blockchains, 2017, http://eprint.iacr.org/2017/179.
[25] R. Cheng, F. Zhang, J. Kos, W. He, N. Hynes, N. Johnson, A. Juels, A. Miller, D. Song, Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contract execution, 2018, arXiv preprint arXiv:1804.05141.
[26] https://www.hyperledger.org/projects/sawtooth, 0000.
[27] hyperledger-labs/minbft, https://github.com/hyperledger-labs/minbft, 0000.
[28] M.J. Fischer, N.A. Lynch, M.S. Paterson, Impossibility of distributed consensus with one faulty process, 32, (2) ACM, 1985, pp. 374–382,
[29] J.-H. Cho, D.P. Sharma, H. Alavizadeh, S. Yoon, N. Ben-Asher, T.J. Moore, D.S. Kim, H. Lim, F.F. Nelson, Toward proactive, adaptive defense: A survey on moving target defense, IEEE Commun. Surv. Tutor. 22 (1) (2020) 709–745.
[30] S. Venkatesan, M. Albanese, K. Amin, S. Jajodia, M. Wright, A moving target defense approach to mitigate ddos attacks against proxy-based architectures, in: 2016 IEEE Conference on Communications and Network Security (CNS), IEEE, 2016, pp. 198–206.
[31] V. Buterin, Ethereum: A next-generation smart contract and decentralized application platform, 2014, https://github.com/ethereum/wiki/wiki/White-Paper (Accessed: 2016-08-22).
[32] A. Miller, Y. Xia, K. Croman, E. Shi, D. Song, The honey badger of bft protocols, 2016, https://eprint.iacr.org/2016/199.pdf.
[33] G.S. Veronese, M. Correia, A.N. Bessani, L.C. Lung, P. Verissimo, Efficient byzantine fault-tolerance, 62, (1) IEEE, 2013, pp. 16–30,
[34] R. Riemann, S. Grumbach, Distributed protocols at the rescue for trustworthy online voting, 2017, arXiv:1705.04480.
[35] S.M. Kim, J. Han, J. Ha, T. Kim, D. Han, Enhancing security and privacy of tor's ecosystem by using trusted execution environments., in: NSDI, 2017, pp. 145–161.
[36] T. Shen, J. Jiang, Y. Jiang, X. Chen, J. Qi, S. Zhao, F. Zhang, X. Luo, H. Cui, Daenet: Making strong anonymity scale in a fully decentralized network, IEEE Trans. Dependable Secure Comput. (2021).
[37] G. Mendonça, G.H. Santos, E.d.S. e Silva, R.M. Leão, D.S. Menasché, D. Towsley, An extremely lightweight approach for ddos detection at home gateways, in: 2019 IEEE International Conference on Big Data (Big Data), IEEE, 2019, pp. 5012–5021.
[38] T. Wong, K. Law, J.C. Lui, M.H. Wong, An efficient distributed algorithm to identify and traceback ddos traffic, Comput. J. 49 (4) (2006) 418–442.
[39] ARM, Security technology building a secure system using trustZone technology (white paper), 0000.
[40] AMD, Amd Secure Encrypted Virtualization (sev) development, http://developer.amd.com/amd-secure-memory-encryption-sme-amd-secure-encrypted-virtualization-sev/, 0000.
[41] H.F. Security, Multizone: The first trusted execution environment for RISC-V, 2018, https://hex-five.com/.
[42] J. Aumasson, L. Merino, Sgx secure enclaves in practice–security and crypto review, Black Hat (2016).
[43] M. Hamburg, P. Kocher, M.E. Marson, Analysis of intel's ivy bridge digital random number generator, 2012, Online: http://www.cryptography.com/public/pdf/Intel_TRNG_Report_20120312.Pdf.
[44] M. Ahmed, K. Kostiainen, Identity aging: Efficient blockchain consensus, 2018, arXiv preprint arXiv:1804.07391.
[45] R. Yuan, Y.-B. Xia, H.-B. Chen, B.-Y. Zang, J. Xie, Shadoweth: Private smart contract on public blockchain, J. Comput. Sci. Tech. 33 (3) (2018) 542–556.
[46] J. Lind, O. Naor, I. Eyal, F. Kelbert, E.G. Sirer, P. Pietzuch, Teechain: A secure payment network with asynchronous blockchain access, in: Proceedings of the 27th ACM Symposium on Operating Systems Principles, in: SOSP '19, ACM, New York, NY, USA, 2019, pp. 63–79, http://dx.doi.org/10.1145/3341301.3359627.
[47] F. Zhang, E. Cecchetti, K. Croman, A. Juels, E. Shi, Town crier: An authenticated data feed for smart contracts, in: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, ACM, 2016, pp. 270–282.
[48] I. Bentov, Y. Ji, F. Zhang, Y. Li, X. Zhao, L. Breidenbach, P. Daian, A. Juels, Tesseract: Real-Time Cryptocurrency Exchange using Trusted Hardware, 2017, Accessed:2017-12-04.
[49] M. Tran, L. Luu, M.S. Kang, I. Bentov, P. Saxena, Obscuro: A Bitcoin Mixer using Trusted Execution Environments, Cryptology ePrint Archive, Report 2017/974, 2017, Accessed:2017-10-06.
[50] M. Castro, B. Liskov, Practical byzantine fault tolerance, in: Proceedings of the Third Symposium on Operating Systems Design and Implementation (OSDI '99), 1999.
[51] I. Eyal, A.E. Gencer, E.G. Sirer, R. van Renesse, Bitcoin-ng: A scalable blockchain protocol, in: 13th USENIX Security Symposium on Networked Systems Design and Implementation (NSDI'16), USENIX Association, 2016.
[52] I. Bentov, R. Pass, E. Shi, Snow white: Provably secure proofs of stake, 2016, https://eprint.iacr.org/2016/919.pdf (Accessed: 2016-11-08).
[53] A. Kiayias, A. Russell, B. David, R. Oliynykov, Ouroboros: A provably secure proof-of-stake blockchain protocol, 2016, https://pdfs.semanticscholar.org/1c14/549f7ba7d6a000d79a7d12255eb11113e6fa.pdf (Accessed: 2017-02-20).
[54] B. David, P. Gaži, A. Kiayias, A. Russell, Ouroboros Praos: An Adaptively-Secure, Semi-Synchronous Proof-of-Stake Protocol, Cryptology ePrint Archive, Report 2017/573, 2017, Accessed: 2017-06-29.
[55] C. Badertscher, P. Gaži, A. Kiayias, A. Russell, V. Zikas, Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability, in: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, ACM, 2018, pp. 913–930.
[56] Y. Sompolinsky, A. Zohar, Accelerating bitcoin's transaction processing. Fast money grows on trees, not chains, IACR Cryptol. ePrint Arch. (2013) 881, URL http://eprint.iacr.org/2013/881.pdf.
[57] A. Gervais, G.O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, S. Capkun, On the security and performance of proof of work blockchains, 2016, https://eprint.iacr.org/2016/555.pdf (Accessed: 2016-08-10).
[58] M. Apostolaki, A. Zohar, L. Vanbever, Hijacking bitcoin: Routing attacks on cryptocurrencies, in: 2017 IEEE Symposium on Security and Privacy (SP), IEEE, 2017, pp. 375–392.
[59] C. Decker, J. Seidel, R. Wattenhofer, Bitcoin meets strong consistency, in: Proceedings of the 17th International Conference on Distributed Computing and Networking, ACM, 2016, p. 13.
[60] L. Aştefanoaei, P. Chambart, A. Del Pozzo, E. Tate, S. Tucci, E. Zălinescu, Tenderbake–classical bft style consensus for public blockchains, 2020, arXiv preprint arXiv:2001.11965.
[61] Y. Amoussou-Guenou, A. Del Pozzo, M. Potop-Butucaru, S. Tucci-Piergiovanni, Correctness of tendermint-core blockchains, in: 22nd International Conference on Principles of Distributed Systems (OPODIS 2018), Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.
[62] B.-G. Chun, P. Maniatis, S. Shenker, J. Kubiatowicz, Attested append-only memory: Making adversaries stick to their word, in: ACM SIGOPS Operating Systems Review, Vol. 41, ACM, 2007, pp. 189–204.

[63] Q. Zhang, Z. Qi, X. Liu, T. Sun, K. Lei, Research and application of bft algorithms based on the hybrid fault model, in: 2018 1st IEEE International Conference on Hot Information-Centric Networking (HotICN), IEEE, 2018, pp. 114–120.
[64] Web3.js - ethereum javascript api, 2021, https://web3js.readthedocs.io/.
[65] D. Ongaro, J. Ousterhout, In search of an understandable consensus algorithm, in: Proceedings of the USENIX Annual Technical Conference (ATC '14), 2014.
[66] D. Mazieres, Paxos Made Practical, Tech. Rep., Technical Report, 2007, http://www.scs.stanford.edu/dm/home/papers.
[67] F. Shaon, M. Kantarcioglu, Z. Lin, L. Khan, Sgx-bigmatrix: A practical encrypted data analytic framework with trusted processors, in: Proceedings of the 17th ACM Conference on Computer and Communications Security (CCS '10), 2017.
[68] C. Priebe, K. Vaswani, M. Costa, Enclavedb: A secure database using sgx, in: Proceedings of the 2018 IEEE Symposium on Security and Privacy, IEEE, 2018.
[69] J. Lind, C. Priebe, D. Muthukumaran, D. O'Keeffe, P.-L. Aublin, F. Kelbert, T. Reiher, D. Goltzsche, D. Eyers, R. Kapitza, et al., Glamdring: Automatic application partitioning for Intel sgx, in: 2017 USENIX Annual Technical Conference (USENIX ATC 17), Santa Clara, CA, 2017.
[70] J. Jiang, X. Chen, T. Li, C. Wang, T. Shen, S. Zhao, H. Cui, C.-L. Wang, F. Zhang, Uranus: Simple, efficient sgx programming and its applications, in: Proceedings of the 15th ACM Asia Conference on Computer and Communications Security, 2020, pp. 826–840.
[71] S. Tikhomirov, Ethereum: state of knowledge and research perspectives, 2017, Accessed:2018-01-05.
[72] L. Lamport, Paxos made simple, http://research.microsoft.com/en-us/um/people/lamport/pubs/paxos-simple.pdf, 0000.
[73] X. Chen, H. Song, J. Jiang, C. Ruan, C. Li, S. Wang, G. Zhang, R. Cheng, H. Cui, Achieving low tail-latency and high scalability for serializable transactions in edge computing, in: Proceedings of the Sixteenth European Conference on Computer Systems, 2021, pp. 210–227.
[74] J. Qi, X. Chen, Y. Jiang, J. Jiang, T. Shen, S. Zhao, S. Wang, G. Zhang, L. Chen, M.H. Au, H. Cui, Bidl: A high-throughput, low-latency permissioned blockchain framework for datacenter networks, in: The 28th ACM Symposium on Operating Systems Principles, 2021.
[75] C. Dwork, N. Lynch, L. Stockmeyer, Consensus in the presence of partial synchrony, 35, (2) ACM, 1988, pp. 288–323,
[76] A.-M. Kermarrec, M. Van Steen, Gossiping in distributed systems, Oper. Syst. Rev. 41 (5) (2007) 2–7.
[77] A.-M. Kermarrec, L. Massoulié, A.J. Ganesh, Probabilistic reliable dissemination in large-scale systems, IEEE Trans. Parallel Distrib. Syst. 14 (3) (2003) 248–258.
[78] I. Gupta, A.-M. Kermarrec, A.J. Ganesh, Efficient and adaptive epidemic-style protocols for reliable and scalable multicast, IEEE Trans. Parallel Distrib. Syst. 17 (7) (2006) 593–605.
[79] D. Shah, et al., Foundations and trends® in networking, Found. Trends® Netw. 3 (1) (2009) 1–125.
[80] A. Wang, W. Chang, S. Chen, A. Mohaisen, Delving into internet ddos attacks by botnets: characterization and analysis, IEEE/ACM Trans. Netw. 26 (6) (2018) 2843–2855.
[81] B. Al-Duwairi, W. Al-Kahla, M.A. AlRefai, Y. Abdelqader, A. Rawash, R. Fahmawi, Siem-based detection and mitigation of iot-botnet ddos attacks, Int. J. Electr. Comput. Eng. (2088-8708) 10 (2) (2020).
[82] Y. Jia, F. Zhong, A. Alrawais, B. Gong, X. Cheng, Flowguard: An intelligent edge defense mechanism against iot ddos attacks, IEEE Internet Things J. 7 (10) (2020) 9552–9562.
[83] W. Chang, A. Mohaisen, A. Wang, S. Chen, Understanding adversarial strategies from bot recruitment to scheduling, in: International Conference on Security and Privacy in Communication Systems, Springer, 2017, pp. 397–417.
[84] A. Wang, W. Chang, S. Chen, A. Mohaisen, A data-driven study of ddos attacks and their dynamics, IEEE Trans. Dependable Secure Comput. 17 (3) (2018) 648–661.
[85] T.A. Tuan, H.V. Long, L.H. Son, R. Kumar, I. Priyadarshini, N.T.K. Son, Performance evaluation of botnet ddos attack detection using machine learning, Evol. Intell. 13 (2) (2020) 283–294.
[86] OSI model - Wikipedia, https://en.wikipedia.org/wiki/OSI_model, 0000.
[87] S. Gilbert, N. Lynch, Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services, 33, (2) ACM, 2002, pp. 51–59,
[88] S. Matetic, M. Ahmed, K. Kostiainen, A. Dhar, D. Sommer, A. Gervais, A. Juels, S. Capkun, Rote: Rollback protection for trusted execution, IACR Cryptol. ePrint Arch. 2017 (2017) 48.
[89] J. Gu, Z. Hua, Y. Xia, H. Chen, B. Zang, H. Guan, J. Li, Secure live migration of sgx enclaves on untrusted cloud, in: 2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), IEEE, 2017, pp. 225–236.
[90] Algorand/go-algorand, https://github.com/algorand/go-algorand/releases/tag/v2.0.14-beta, 0000.
[91] Cryptocurrency statistics, 2019.
[92] Deloitte, Blockchain performance report, 2018, https://www2.deloitte.com/content/dam/Deloitte/ie/Documents/Technology/IE_C_blockchain_performance_report.pdf.
[93] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, H. Balakrishnan, Chord: A scalable peer-to-peer lookup service for internet applications, SIGCOMM Comput. Commun. Rev. 31 (4) (2001) 149–160.
[94] I. Weber, Q. Lu, A.B. Tran, A. Deshmukh, M. Gorski, M. Strazds, A platform architecture for multi-tenant blockchain-based systems, in: 2019 IEEE International Conference on Software Architecture (ICSA), IEEE, 2019, pp. 101–110.
[95] A. Baliga, I. Subhod, P. Kamat, S. Chatterjee, Performance evaluation of the quorum blockchain platform, 2018, arXiv preprint arXiv:1809.03421.
[96] P. Tarasov, H. Tewari, Internet Voting Using Zcash, 2017, Accessed: 2017-06-29.
[97] E.-O. Blass, F. Kerschbaum, Strain: A Secure Auction for Blockchains, Cryptology ePrint Archive, Report 2017/1044, 2017, Accessed:2017-11-06.
[98] T. Kim, J. Park, J. Woo, S. Jeon, J. Huh, Shieldstore: Shielded in-memory key-value storage with sgx, in: Proceedings of the Fourteenth EuroSys Conference 2019, ACM, 2019, p. 14.
[99] R. Dingledine, N. Mathewson, P. Syverson, Tor: The Second-Generation Onion Router, Tech. Rep., Naval Research Lab Washington DC, 2004.
[100] R. Henry, A. Herzberg, A. Kate, Blockchain access privacy: Challenges and directions, IEEE Secur. Priv. 16 (4) (2018) 38–45.
[101] Q. Feng, D. He, S. Zeadally, M.K. Khan, N. Kumar, A survey on privacy protection in blockchain system, J. Netw. Comput. Appl. 126 (2019) 45–58.