A Geography-Based P2P Overlay Network for Fast and Robust Blockchain Systems

Haoran Qiu[1,2], Tao Ji[1,3], Shixiong Zhao[1], Xusheng Chen[1], Ji Qi[1], Heming Cui[1], and Sen Wang[4]

[1]University of Hong Kong, [2]University of Illinois, Urbana-Champaign, [3]University of Texas, Austin, [4]Huawei Technologies Co. Ltd, China

Abstract—Numerous blockchain systems with various consensus protocols have emerged to achieve high transaction rates (2∼10K tps). However, their underlying P2P network primitives constrain further improvements due to two problems (i) high message redundancy and (ii) long broadcast convergence time. The first problem is caused by the excessive robustness of the dominant broadcast approach Gossip. All state-of-the-art blockchain systems only tolerate 20-50% node failure while Gossip can withstand up to 90%. The reason for (ii) is that existing broadcast topologies ignore geographical distances among nodes and incur paths with unnecessarily high latency. We present FRING, a geography-based P2P overlay network for fast and robust broadcast in blockchain systems. FRING has three main features: sufficient robustness, low message redundancy, and fast convergence. To reduce convergence time, FRING forms the network topology by considering geographical proximity. A novel broadcast algorithm based on FRING topology is proposed to lower message redundancy while maintaining sufficient robustness. One major challenge is to eliminate the risk of topology inference by traffic pattern analysis. FRING leverages Intel SGX to guarantee nodes' behavior integrity and incorporates pattern obfuscation to prevent traffic pattern analysis. The evaluation shows that FRING improved the throughput of EOS by 220% and Hyperledger Fabric by 210%.

Index Terms—Distributed Networks, Network Topology, Reliability, Network-level Security and Protection

1 Introduction

After the popularity of Bitcoin [1], numerous block-chain systems [2], [3], [4], [5] appeared to support both cryptocurrency and general applications (e.g. smart contracts [2]). Conceptually, a blockchain system is divided into three layers. From top to bottom, the first layer is the *application layer*, which can be cryptocurrencies or smart contracts. The second layer is the *consensus layer*, where nodes agree on a chain of blocks via consensus protocols (e.g. Proof-of-Work [1]). The third layer is the *P2P overlay network layer*. It handles broadcasting data packets (e.g. consensus messages, newly generated blocks or transactions) and discovering peers.

To measure the blockchain system performance, a crucial metric is the transaction rate, i.e., the number of transactions confirmed by the system per second [6]. Benefiting from the evolution of the consensus layer [4], [5], [7], [8], the transaction rate of blockchain systems has increased significantly. Some public [5] and private [9] blockchain systems have claimed to achieve 2∼10K tps transaction rates.

Unfortunately, recent studies [9], [14], [15] show that the underlying P2P network layer has become the bottleneck for transaction rate. Blockchain systems form random P2P networks and predominantly use Gossip [13] as the broadcast mechanism for robustness. For each hop in the broadcast process, a node pushes the message to a randomly chosen subset of its neighbors or pulls from the message sender. There are two notorious problems associated with this network solution.

The first problem is that Gossip [13] generates excessive redundant messages because Gossip is designed for extreme robustness (can tolerate up to 90% node failure [16]). Such redundant messages lead to traffic congestion under high transaction rates. Although several works reduce the message redundancy of Gossip, they are either not scalable [17] or incurring huge performance overhead [18]. ByzCoin [12] leverages tree-based broadcast, which is efficient in the number of messages generated ($O(N)$) but criticized for low robustness (only tolerating leaf node failure) [10]. This problem is exacerbated as the network or the number of broadcast events scales up. Old messages on the flight may accumulate and gradually cause cascading traffic congestion.

We argue that Gossip is overly robust for blockchain systems. All state-of-the-art blockchain systems only tolerate 20–50% node failure (Table 2) while Gossip can tolerate nearly 90%. For instance, the robustness of Gossip is meaningless when a 60% node failure rate already crashes the consensus protocol.

The second problem is that random network topology causes unnecessarily long broadcast convergence time (i.e. time used for a message to cover all nodes in the network). Since the peer discovery process and peer selection on broadcast are essentially random (§2.2, §2.3), the broadcast graph formation ignores *per-hop* latency between peers. The consequence is that high-latency hops are often incurred when broadcasting a message, which leads to frequent jumping between two components that are far away from each other. In the end, a broadcast process might take an excessively long time to converge in the network.

We present FRING, a geography-based P2P overlay network for blockchain systems with a high transaction rate. It addresses the above two problems with three features:

(i) Sufficient robustness: a broadcast operation can tolerate at least the same portion of node failure as the blockchain consensus protocol;
(ii) Low message redundancy: the messages generated in each broadcast is efficient ($O(N)$); and
(iii) Fast convergence: the convergence time is minimized so that the message accumulation is reduced effectively;

Conceptually, the network topology of FRING is a self-maintained fractal ring structure, where lower level rings (typically formed by inner-region nodes) reside on higher level rings (typically formed by cross-region nodes) in a recursive way. To form this topology, FRING groups all nodes based on their geographical proximity (*per-hop* latency) at the bottom level. Each group forms a fully-connected ring. FRING selects multiple nodes from each ring to serve as *representatives* and they form a new ring in the higher level recursively. Finally,

TABLE 1: Comparison among different broadcast approaches.

Broadcast Approaches	Msg Redundancy	Convergence Time	Robustness
Tree-based [10], [11], [12]	$O(N)$, optimal	Medium (non geo-based, deterministic)	Low, tolerate only leaf node failure
Gossip-based [2], [13]	$O(NlogN)$	Slow (non geo-based, probabilistic)	Extremely high, tolerate up to 90% node failure
FRING	$O(N)$, optimal	Fast (geo-based, deterministic)	Sufficient for all blockchain systems (Tab. 2), tolerate over 50% node failure

there is only one global ring remaining at the top level and all nodes are connected through *representatives*.

To broadcast a message, FRING introduces a new two-phase broadcast mechanism. In phase one, a node first sends the message to a random subset of its *representatives* in the ring. Recursively, the *representative* then sends the message to a random subset of the *representatives* at the upper level. Each representative who sends the message to the upper level will initiate an in-ring broadcast, other representatives will stop the broadcast-up process. In the second phase, after the message reaches one of the *representatives* at the top level, the message is broadcast downwards. Each *representative* will receive the message from its peers in the ring at the upper level. It then initiates a distributed k-ary broadcast method in the ring at the next lower level. It is inspired by the distributed k-ary search method [19] which provides $O(N)$ message efficiency.

Although adopting geographical proximity reduces convergence time, *representative* nodes are more vulnerable to targeted attacks. The reason is that attackers can eclipse-attack a node by setting up malicious nodes near to it. To mitigate the problem, we run FRING program in Intel SGX [20], which guarantees code execution integrity and provides privacy protection.

One major challenge is that attackers can infer the structure of FRING by observing I/O patterns at each node so that the *representatives* are at high risk of being attacked. We will show in §8 that the pattern difference between normal nodes and *representatives* is negligible with pattern obfuscation.

We implemented FRING as a library in C++. We evaluated EOS with FRING (i.e., EOS-FRING) and HLF with FRING (i.e., HLF-FRING) on Amazon AWS, and compared them with EOS and HLF. We also evaluated FRING against Gossip in terms of time and message efficiency, as well as fault tolerance. Evaluation shows that:

- The throughput of EOS-FRING and HLF-FRING are greater than EOS-Gossip and HLF-Gossip by up to 2.2× and 2.1× respectively (see Fig. 1). Specifically, the advantage of FRING becomes bigger when transaction generation rate grows.
- FRING converges faster. Our broadcast algorithm on FRING network is six to ten times much faster than Gossip on random networks in blockchain systems under reasonable convergence rate requirements.
- FRING is network-efficient. FRING reduces 52%∼77% redundant messages of Gossip when the total number of nodes in the network scales from 2000 up to 8000.
- FRING has sufficient robustness. FRING can tolerate over 50% node failure, higher than the portion of nodes that all blockchain system consensus protocols can tolerate. In addition, traffic analysis shows that the traffic pattern difference between normal nodes and *representatives* is negligible.

Our main contribution is FRING, the first geography-based P2P overlay network that achieves fast and robust blockchain broadcast. On top of the FRING network, we also propose a novel broadcast algorithm for blockchain system context. Table 1 shows a qualitative comparison between our approach of broadcast and the conventional ones. FRING is orthogonal to the consensus layer and can be adopted in general blockchain systems by replacing their network layers. FRING has the potential to enable the development of blockchain consensus protocols for even higher transaction rates. An implementation of FRING and evaluation scripts are open-sourced at https://github.com/tsc19/fring.

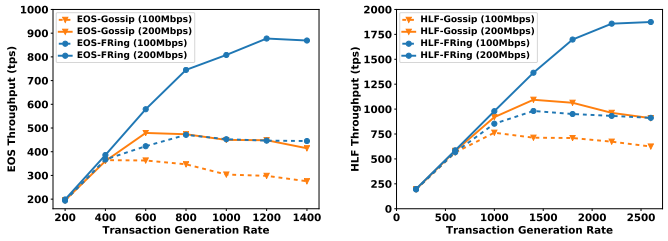


Fig. 1: End-to-End performance comparison between EOS, HLF and EOS-FRING, HLF-FRING. The total number of nodes in the network is 200. Two corresponding network settings are 100 Mbps and 200 Mbps for each node.

The rest of this paper is organized as follows. §2 introduces the background of the problem we focus on and related works. §3 presents the motivation, design guidance, and an overview of FRING design. §4, §5, and §6 elaborate the remaining details of FRING design. §7 gives theoretical proof for time and message complexity of broadcast and analyzes the robustness of FRING with respect to node failures. §8.1 describes the implementation details. §8 presents and discusses the evaluation results. §10 concludes the paper.

2 Background and Related Work

2.1 Blockchain Systems

Conceptually, as shown in Figure 2, the components of a blockchain system can be categorized into three layers on top of the OS layer. The highest layer is the application layer which can be cryptocurrency applications or smart contracts. In the middle layer, all nodes in the blockchain systems agree on a consistent chain of blocks through consensus protocols. The model of transactions and blocks are also defined here. Below is the P2P overlay network layer which has two major functions: peer discovery, maintaining the table of peered nodes to communicate with; and information dissemination, broadcasting the events such as consensus messages, membership updates, new transactions or newly generated blocks.

The earliest blockchain systems such as Bitcoin and Ethereum use the Proof-of-Work (PoW) consensus protocols to support the valuation of the cryptocurrency in a socioeconomic sense [1], [2], which results in poor efficiency and low transaction rate. Different techniques and blockchain systems [21], [22], [23] have been proposed to increase the scalability and transaction rate (throughput), including sharding, off-chain transactions, consensus protocols, and so on. To facilitate general-purpose applications in a more efficient manner, other PoX protocols have been presented in different scenarios such as Proof-of-Stake [7], Proof-of-Luck [8], and Proof-of-Membership [12]. Non-PoX-based protocols are also proposed such as DPoS [5] and DBFT [24]. Hyperledger Sawtooth [4] utilizes Intel Software Guard eXtentions (SGX, introduced in 2.4) to provide integrity guarantee to nodes and proposes Proof-of-Elapsed Time (PoET), which is believed to be highly efficient. Additionally, EOS [5] based on DPoS, NEO [24] based on DBFT, Conflux [15], Omniledger [25], and Hyperledger Fabric (HLF) [9] all claim to achieve 2K to 10K tps transaction rate [26], [27], [28] (300X to 1500X higher than Ethereum [2]). We further

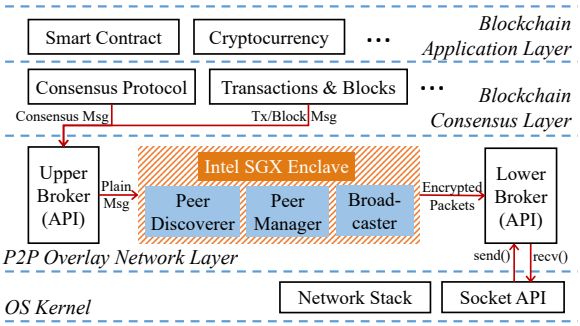


Fig. 2: A layered conceptual model of blockchain systems on top of the host OS. In FRING, shaded modules are executed in the Intel SGX enclave, i.e. peer discoverer module, peer manager module, broadcaster module.

improve the transaction throughput of blockchain systems from the overlay network layer (as shown in §8.3).

Current blockchain systems cannot serve for higher transaction rates, mainly because their network layer constrains improvements. EOS technical report [14] claims that poor network latency caused by high bandwidth utilization can undermine its transaction throughput (50 ms network latency leads to a 75% transaction rate drop). HLF [9] also blames the current broadcast solutions for being the cap of improving its transaction throughput. As more and more blockchain systems are deployed in commodity servers and geo-located datacenters connected together, blockchain-powered ledger can be used in broader cases (application layer in Figure 2). With the transaction rate of state-of-the-art blockchain systems being 2-10k txns/s, we envision that it will keep increasing (e.g., [29] proposes a blockchain system that already achieves 40k txns/s).

2.2 P2P Overlay Networks

Many of the existing P2P overlay networks are tailor-made for various applications. We categorize the mainstream P2P applications into four types: i) On-demand streaming (e.g.. BASS [30], Peer-Assisted [31], LiveBT [32] and Give-To-Get [33]); ii) high-throughput multicast (e.g.. SplitStream [34], Bullet [35] and ChainSaw [36]); iii) Audio/video conferencing (e.g.. Skype [37]); and iv) P2P file sharing (e.g.. Napster [38], Gnutella [39] and KaZaA [40]).

However, none of them are suitable for blockchain systems. i) is designed for single-destination persistent data streaming, while the blockchain system network demands broadcast. ii) and iii) are optimized for high-bandwidth persistent multi-cast, whereas the blockchain system network mostly broadcasts relatively small blocks of data with high concurrency (numerous broadcast operations simultaneously in progress) and needs to be bandwidth-friendly. iv) is leveraged for distributed file indexing and searching, while the retrieval is mostly point-to-point without the involvement of the P2P network, which makes it irrelevant to our problem.

There is no P2P overlay network designed for blockchain systems with high transaction rates. By looking into the documentation and the code of open-source blockchain system implementations, we discover that the network topology of most blockchain systems is randomly generated, resulting in the network topology being a random graph. This approach is preferred for simplicity. Bitcoin nodes, for instance, retrieve the information of active nodes in the network from DNS seeds or the designated nodes [41], and randomly peer up with a subset of nodes. Perigee [42] proposes a scoring algorithm and adaptively decides which neighbors to connect to purely based on the network measurements between a node and its neighbor nodes in the network topology graph.

Ethereum, on the contrary, uses a Kademlia-like distributed hash table (DHT) to store the peer information [2], [43]. Each node ID is a public key randomly generated from the elliptic curve secp256k1. However, the ID to geographical locality mapping is still random. The node table stores up to 16 nodes of distances $\{2^i, 2^i+1, ..., 2^{i+1}-1\}$, $\forall i \in \{0, 1, ..., 255\}$, where the distance is determined by the bitwise XOR of two node IDs. Compared to random neighbor selection, the schema used in Kademlia network is deterministic. Such a schema makes the node table in each node predictable so that properties can be formally proved [44]. Function mapping has high efficiency on peer discovery. Kademlia contacts only $O(log(N))$ nodes during the search out of a total of N nodes in the system.

Other DHTs include Chord [45], Pastry [46] and Tapestry [47]. These DHTs are also promising choices for the node table. Nonetheless, it is worth mentioning that simply adopting such a schema does not make the broadcast performance superior to the randomly generated network topologies, since the ID distance has nothing to do with the geographical locality, not to mention the latency between two nodes, which has led us to design a novel network that takes the latency into account.

2.3 Dissemination Algorithms

Apart from the network topology, the algorithm that broadcasts the data over the network is also critical for a blockchain system. We find that all the blockchain systems that we looked into use Gossip [48] as the fundamental broadcast algorithm. Gossip-based algorithms are developed for high reliability and scalability of information delivery [49]. They are widely used for reducing control message overhead [50] and are scalable because they do not require as much synchronization as traditional reliable multi-cast algorithms. With Gossip, nodes push the messages to a randomly chosen subset of known peers if there are new messages (push-version Gossip) or pull from the message sender (pull-version Gossip). The dynamics of the gossip-like behavior and redundant messages lead to high fault tolerance. Such algorithms usually do not require error recovery mechanisms, hence possess a large advantage in simplicity, often incurring only moderate overhead compared to optimal deterministic algorithms.

The drawback of the Gossip algorithms, though, lies in their inherent redundant messages [13], which may lead to traffic congestion as broadcast frequency grows. There are several improvements made on Gossip. Directional Gossip [17] uses a Gossip server to construct a spanning tree but it is not scalable. Intelligent Gossip [18] selects directional children based on intelligent heuristics to build a tree. However, it does not work if nodes do not have an overview of the whole network. Other improvements include the adoption of time to live (TTL) [51], and the use of unique message identifier (UMID) to reduce redundancy [13], but they hardly mitigate the essential redundancy that exists in the algorithm.

This problem is exacerbated as the network or the number of broadcast events scales up. In such cases, data packet transportation is slowed down and the blockchain system throughput is reduced. Worse, to cover as many nodes as possible, Gossip retransmits messages for more times [13]. Therefore, to reach the entire network, the number of message retransmissions must be set to a value equal to or higher than the network diameter [10]. With broadcast frequency increasing, old messages on the flight may accumulate and gradually cause cascading traffic congestion. If the network fails to achieve super majority agreement on validations, consequently the consensus process is hard to produce consistent proposals. In this case, the servers repeat the consensus process [52].

Although such redundancy is arguably necessary to guarantee the robustness [13], the level of fault-tolerance is much

TABLE 2: Consensus algorithms, blockchain system examples and their tolerated percentages

Consensus Protocols	Max # of Failures	Examples
Proof-of-Work	$N/2-1$	Ethereum [2]
Proof-of-Stake	$N/2-1$	PeerCoin [53]
Practical BFT	$N/3-1$	HyperLedger Fabric [9]
Distributed PoS	$N/2-1$	Bitshares [54], EOS [5]
Ripple	$N/5-1$	Ripple [52]
Tendermint	$N/3-1$	Tendermint [55]

higher than what is required by consensus protocol. Table 2 lists the maximum number of failed nodes that can be tolerated for popular consensus protocols. Typically, a consensus protocol cannot bear more than half of the nodes being failed, while some Gossip-based algorithms such as HyParView can stand as much as 90% node failure [13]. The actual implemented version of Gossip in different blockchain systems is hard to extract (as mentioned in §8.2) and based on our observation on the Kademlia-based Gossip used in Ethereum, HLF and EOS, the actual percentage highly depends on the topology of the network. Since in Gossip nodes select their peers to broadcast randomly, the percentage of the node failures that can be tolerated ranges from 50% to around 90%. Therefore, it is possible to exchange the surplus margin for better efficiency.

Some blockchain systems and P2P applications use Tree-based broadcast algorithms, which are well-known for their low redundancy. For instance, there are tree-based broadcast/-multicast algorithms leveraged by video streaming and file sharing applications [56]. Tree-based broadcast approaches like SplitStream [34] and CoopNet [57] are efficient but the structure needs to be maintained. ByzCoin [12] also creates communication trees for broadcast. However, this approach introduces overhead since each node will create a communication tree covering all nodes when a new block is created. Despite low message complexity and broadcast efficiency, tree-based broadcast algorithms are notorious for their vulnerability of node failure and high overhead of tree maintenance [10]. Therefore, ByzCoin uses Gossip instead of the tree-based broadcast when the liveness of nodes is low.

Data-driven broadcast approaches in ChainSaw [36], Bullet [35], and CoolStream [58] have also significantly reduced the redundancy compared to Gossip. However, the model of single-source persistent broadcast/multicast streaming model does not fit in the context of blockchain systems, where small data blocks are broadcast and multiple sources should be able to broadcast simultaneously. Therefore, it turns out that no existing broadcast algorithm can be regarded as a perfect solution for current or future blockchain systems, especially with high transaction rates and global availability.

2.4 Intel SGX

Intel SGX is a set of extensions to the Intel architecture that aims to provide integrity and confidentiality guarantees to security-sensitive computation performed on a computer where all the privileged software (kernel, hypervisor, etc.) is potentially malicious [20]. Intel SGX provides two kinds of attestations, local and remote, to enable the verification such that a particular piece of code is running in an enclave of a genuine SGX-enabled CPU [20], [59], [60]. Enclaves are isolated memory regions of code and data. One part of physical memory (RAM) is reserved for enclaves. During a remote attestation, the challenger establishes a secure communication channel with the help of key-exchange protocols (Diffie-Hellman Key Exchange [61]). In addition to the attestation, Intel SGX provides a trustworthy source of random numbers via its `sgx_read_rand` API [62] which calls the hardware-based pseudo-random generator (PRNG) through RDRAND on Intel CPUs [20]. Previous studies show that this random number generator is safe and cannot be altered from outside the enclave [63], [64], [65].

In general, Intel SGX is a powerful tool that can be leveraged to establish trust among the nodes in a distributed system. It has been leveraged by some consensus protocols to eliminate the overhead of byzantine fault-tolerance [66], [67]. Later, we will also illustrate how FRING uses such techniques to mitigate potential attacks. Apart from Intel SGX, other commodity hardwares also provide similar protection or trusted execution environments. For instance, AMD has its Secure Virtual Machine (SVM) architecture [68] and memory region encryption technology [69]; ARM has the TrustZone technique [70].

3 Design Motivation

By studying the blockchain systems and their underlying P2P networks, we summarized three key requirements from blockchain systems with high transaction rates (they are also three features of FRING, see §1). These three requirements motivate us to come up with the following four guidances of FRING design. In each subsection, we first elaborate the design motivation in each particular aspect of three requirements and then discuss how it serves as a guide for the design of FRING.

3.1 Fast Convergence

There are two factors that could affect the convergence time of broadcasting a message in a P2P network, the first is the number of rounds needed to complete the broadcast process, and the other is the communication delay which is the time consumed in one hop of transmission. Gossip-based broadcast approaches are criticized for their probabilistic broadcast and "long tail" [71] (as mentioned in Section §1). The randomization in selecting the neighbor nodes increases the likelihood of a node remaining isolated for several broadcast rounds. Besides, the ignorance of geographical location leads to longer tail latencies. A tree-based broadcast approach achieves $O(logN)$ rounds in a deterministic way. However, the robustness of tree-based broadcast approaches needs to be improved. All internal nodes in the broadcast tree are fragile in the sense that once an internal node fails then all its descendants would not get the broadcast message.

Therefore, FRING uses a tree-based broadcast mechanism but with multiple internal nodes connected with a sub-tree. In this way, the number of rounds to broadcast a message in the network would still be $O(logN)$ deterministically. The robustness of such a network topology will be discussed in §3.3. §3.2 will discuss how FRING could be designed to deal with the issue that every node in the network should be able to initiate the broadcast process instead of only the root node.

To reduce the communication delay, the construction of a P2P network should also consider geographical proximity. When a node requests to join the network, FRING will recursively introduce the node to the *representative* node which has the lowest network latency with it. In the end, the node will join a group where it has a node table containing all neighbors which are geographically proximate to it. Therefore, by considering geographical locality and using structured broadcast, a broadcast operation could achieve fast convergence.

3.2 Low Message Redundancy

Gossip algorithms do not fit in the context of blockchain systems where the transaction generation rate is high enough to cause traffic congestion in the network. Tree-based broadcast approaches can reduce message redundancy since pushing messages from the root to all leaves in the tree results in $O(N)$ message complexity. However, as mentioned in §3.1, tree-based broadcast approaches have two main problems: (i) the robustness is low; and (ii) there will be N trees if any node can be the source node, otherwise, it is time-consuming to construct broadcast-tree before each broadcast operation.

Problem (i) can be addressed together with providing sufficient robustness (§3.3). Problem (ii) can be solved by a novel two-phase broadcast mechanism (§5.1, §5.2) in FRING so that neither does the network need to store N broadcast trees, nor does it need to construct a broadcast-tree before each broadcast operation.

3.3 Sufficient Robustness

A tree-based broadcast approach has low robustness and all internal nodes in a tree are quite important. In addition, the higher level a node is at, the more important role it plays in a broadcast process. Consequently, it would be easier to be the attack target by hackers outside the network. Fault-tolerant replication of such nodes can improve the robustness and a blockchain system needs sufficient robustness in a way that a message should reach enough number of nodes.

FRING replaces a single node at the connection point to a sub-tree with a ring of nodes (illustrated in §4.1). To be more specific: all nodes in FRING that reside in the bottom level are grouped into rings according to geographical locality (§3.1). There will be *representative* nodes elected from each ring. All *representative* nodes elected are seen as normal nodes in the upper level. Such an election process is done recursively to form a tree-like recursive ring structured network topology.

In this fractal-ring "tree", instead of having one node connected with a sub-tree, there is a ring of *representative* nodes connected with the sub-trees having the *representative* nodes as the root nodes. In a broadcast process, the same message will be pushed to all *representative* nodes of a sub-tree. Only one alive message will be broadcast further in the sub-tree. By using this approach, the robustness is improved and sufficiency can be adjusted by setting the parameter of how many *representative* nodes should be elected from a ring, in order to satisfy blockchain systems' needs.

3.4 Handling Targeted Attacks

Grouping nodes that are geographically proximate with each other together in a ring may cause eclipse attacks [72]. In other words, if a victim node joins a group that only contains bad nodes, an eclipse attack could be conducted easily. In addition, a structured P2P network is different from an unstructured one in the sense that structured overlays are not secure; even a small fraction of malicious nodes can prevent correct message delivery throughout the overlay [73]. Especially when a malicious node plays an important role in the network, its misbehavior may cause the whole FRING network to malfunction. For routing algorithms that use network proximity information to improve routing efficiency, attackers may fake proximity to increase the fraction of bad routing table entries.

Intel SGX can be leveraged to avoid eclipse attacks by protecting the confidentiality of routing information and to make sure the integrity of node behavior (§6). Running code in Intel SGX enclaves can avoid node misbehavior so that secure message forwarding can be guaranteed though packets could still be dropped (which is addressed in §6.2). Apart from using Intel SGX to improve security, FRING periodically elects new *representative* nodes and equalizes all nodes in the structured network to remove the existence of super peers. All nodes have equal opportunities to be rotated into *representative* nodes at different levels in the network.

4 Network Design

In this section, we present the structure-wise design details of FRING which include the topology of the network (§4.1), how the structure is formed (§4.2), how node failure is detected (§4.3), the maintenance of FRING network (§4.4), and *representative* node election (§4.5).

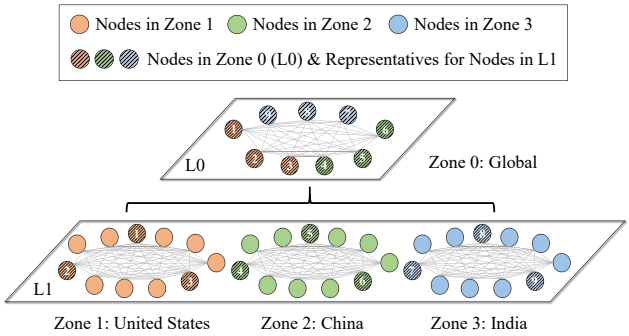


Fig. 3: Recursive step of FRING network structure construction. Three representatives are selected from each ring (zone) in level 1 to be normal nodes in level 0. Recursively, three representatives in Zone 0 will be selected.

Before diving into the details of FRING design, we define several key terminologies here:

- **Normal Node/Node**: one instance of servers or VMs in the FRING network;
- ***Representative* Node**: the node in a ring who is in charge of membership updates, connecting to the ring at the upper level, initiating *representative* node election, and broadcasting downwards messages;
- **Ring**: a group of nodes connected in a ring-like structure, which have the list of nodes in the same ring as their routing table;
- **Predecessor & Successor**: the node ahead and behind a node (clock-wise) in a ring, e.g., the predecessor and successor of node 1 are node 9 and node 2, respectively (as in Fig. 3);
- **Level**: all nodes reside at the bottom level, *representative* nodes for level n are elected to be normal nodes in level $n-1$ (the network topology is recursively formed in this way until the top level).

4.1 Topology

Conceptually, the network topology of FRING is a fractal-ring structure, where lower level rings reside on higher level rings in a recursive way. At the top level resides the largest ring where several sub-rings reside on. Each nested ring is formed all the way to the bottom level at which resides the smallest rings. To form such a structure, all nodes in FRING network reside at the bottom level, grouped in rings according to their geographical locality (§3.1). In each ring, multiple nodes are elected to be the *representative* nodes for this ring. These *representative* nodes in level n become the normal nodes in level $n-1$. Recursively, all levels of nodes are connected through *representative* nodes to form this fractal-ring structured topology.

Fig. 3 illustrates how rings at two consecutive levels (level 0 and level 1) are connected and the relationship between them. At level 1, there are three rings and the nodes colored in shadow (which are also numbered from 1 to 9) are the *representative* nodes elected for each ring at this level. They are also the normal nodes at level 0. Three sub-rings at level 1 form a ring at level 0 with the connection points being the *representative* nodes for rings at level 1. Recursively, multiple nodes will be elected to be the *representative* nodes at level n and they are going to be the normal nodes at level $n-1$. The recursion will stop when reaching level 0 which is also the top level.

4.2 Node Join

When a new node wants to join the network, it will first send ping-messages to each of the *representative* nodes of the largest ring at the top level maintained by a seed node network [74]. The new node will pick the *representative* node with the shortest response time to send a join-message. The *representative* node

will then decide which sub-ring it should add this new node to, by sending the *representative* information of *representative* nodes of each sub-ring back to the new node. Recursively, the new node will then choose the sub-ring which is optimal in terms of response time. And the *representative* node of the sub-ring will then introduce the new node to the sub-sub-ring. In the end, the *representative* node of a ring at the bottom level will then add the new node to the ring.

If the number of nodes on the ring that the new node joins exceeds the threshold `MAX_THRESHOLD`, then this ring will be split up into two rings evenly. The transformation will be further elaborated in §4.4. After a new node joins the network, the *representative* node of this ring will broadcast within the ring this node's information. Other peers in this ring will then tell their node information by sending welcome-messages to the new node. At the end of the bootstrap process for the new node, a routing table will be formed to contain all its peers in the ring.

4.3 Node Leave

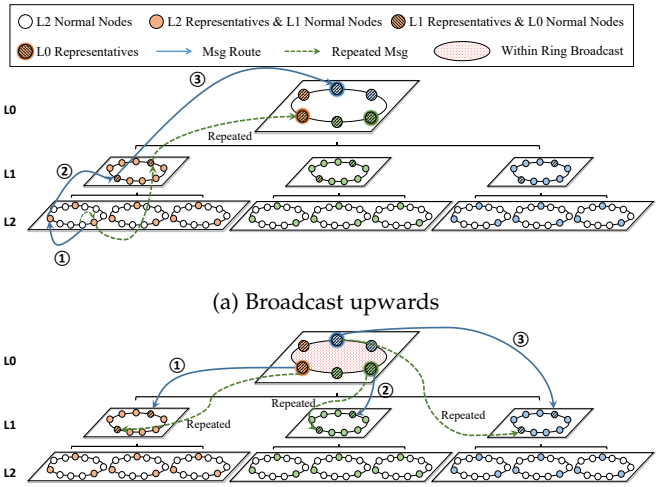
A node in the network may intentionally leave the network or go through network failure or machine failure. We treat node failure the same as node leave and use a heartbeat-style detection mechanism to handle this. Since all nodes exist at the bottom level and all nodes in the upper levels above the bottom level are only roles, detection is only needed at the bottom level. Each node on any ring in the bottom level will send heartbeat messages periodically to its successor and predecessor. Once a node does not receive any heartbeat message for a predefined timeout value, the node will double-check the aliveness of the heartbeat sender with its direct neighbor.

For example, there are nodes $D \rightarrow B \rightarrow A \rightarrow C \rightarrow E$ which are a subset of the node ring in Zone 1 on L1 in Fig. 3. if node A's predecessor B does not respond, A will check with the predecessor of B (similarly if A's successor C does not respond, A will check with the successor of C). If they agree that this node left the network (either intentionally or accidentally), the information will be disseminated in the ring (broadcast within-ring described in §5.3) and this node will be officially removed from the network. If the missing node is the *representative* node, then the next generation of *representative* nodes will be elected. If the number of nodes on the ring is smaller than the lower limit `MIN_THRESHOLD`, two rings will be merged into a larger ring. Merge steps will be elaborated in §4.4.

4.4 Structure Maintenance

FRING structure maintenance consists of two parts: ring split and ring merge. Each ring has two thresholds limiting the number of nodes in that ring, i.e., `MAX_THRESHOLD` and `MIN_THRESHOLD`. After a node-join, if the number of nodes exceeds the `MAX_THRESHOLD`, then a ring split process will be performed. Similarly, after a node-leave, if the number of nodes is smaller than the `MIN_THRESHOLD`, a ring merge process will be performed. An appropriate number of nodes in a ring will lead to an optimal performance in terms of the time used to broadcast a message. Having too many and too few nodes in a ring would have nearly linear time complexity in extreme cases (i.e., all nodes in one ring or one node in one ring). In our experiments (i.e., §8), we choose `MIN_THRESHOLD` to be 10 and `MAX_THRESHOLD` to be 20 to maintain the number of within-ring hops to be under 3 (based on the broadcast algorithm described later in §5.2) which yields the optimal performance. Both two processes are initiated by one of the *representative* nodes of the ring.

After a ring split process is initiated, the peer list of this ring is split randomly into two halves, each of which forms a new ring. *Representative* node election will be initiated by the first node on the new peer list. One of the *representative* nodes from the last term will multicast the election result notification message to the *representative* nodes in the upper level ring. After a ring merge process is initiated, the *representative* node of this ring will find another ring randomly to merge with through the common ring at the upper level. Two peer lists are merged together and a *representative* node election will be initiated by one of the *representative* nodes. One of the *representative* nodes from the last term will multi-cast the election result notification message to the *representative* nodes in the upper level ring. In addition, after the ring split, the rings in the upper level rings may also need to be split into two rings. Similarly, after the ring merge, the new ring may need to be split into two rings. The chain processes will be done recursively.



(a) Broadcast upwards

(b) Broadcast downwards to the second top layer

Fig. 4: Illustration of a broadcast upwards process (a) and a broadcast downwards process (b) in a three-layer FRING network.

4.5 Representative Node Election

Representative node election will be done first when forming a new ring, and periodically afterward. Each generation of *representative* nodes have their term of service. At the end of the term, one of the *representative* nodes will generate random IDs from all member node IDs by using Intel SGX API `sgx_read_rand()` (§6). This election result will be dispersed within the ring, and multi-casted to the *representative* nodes of the upper level ring and the lower level rings.

5 Broadcast Algorithm

In this section, the broadcast algorithm is stated which contains three parts: broadcast upwards (§5.1), broadcast downwards (§5.2), and broadcast within-ring (§5.3).

5.1 Broadcast Upwards

Broadcast is the process of disseminating a message from any node to the whole network. A broadcast process contains two phases: broadcast upwards (this section) and broadcast downwards (§5.2). When a node wants to send a message to the whole network, it will first send the broadcast-message to one of the *representative* nodes of the ring it resides on. The *representative* node will then send broadcast-message to one of the *representative* nodes of the upper level ring. Recursively, the broadcast-message will be received by one of the *representative* nodes of the largest ring. Then the *representative* node will start to broadcast within-ring (§5.3) and broadcast downwards (§5.2). To further optimize performance, whenever a *representative* node in the upwards path receives the message, it will also broadcast the message in the ring and recursively in the sub-rings in the same way as broadcast downwards.

Fig. 4(a) illustrates a broadcast upwards process in a three-layer FRING network. A node at the bottom level (level 2) wants to broadcast a message to the whole network. It will first send the message to the *representative* node in the same ring (step ①). Then the *representative* node at level 2 will send the message to multiple *representative* nodes at level 1 (step ②). Each representative who sends the message to the upper level will initiate an in-ring broadcast (§5.3), other representatives will stop the broadcast-up process. Finally, the *representative* node at the top level will receive the message (step ③) and the broadcast upwards process ends.

The distributed k-ary broadcast method inspired by a search method [19] is used to disseminate a message in a ring. Details will be presented in §5.3. Based on this method, the time complexity of a whole broadcast operation will be $O(logN)$ and message complexity will be $O(N)$, which are currently the best among related works.

5.2 Broadcast Downwards

The broadcast downwards process happens when an internal node in the FRING tree-like structure receives a message to broadcast downwards. This process is done in a recursive way in all rings at all levels (excepts the bottom level). Any normal nodes at level n are the *representative* nodes for a sub-ring at level $n+1$. To broadcast downwards, the node at level n will initiate a within-ring broadcast in the sub-ring at level $n+1$ (§5.3) after it receives the message to broadcast. Recursively till the bottom level, all nodes in the fractal ring will receive this message. Fig. 4(b) illustrates a broadcast downwards process happening in the ring at level 0. After the message is dispersed within the ring, three L0 representative nodes (highlighted with bold borders) initiate broadcast-down process to their sub-rings at level 1 respectively.

5.3 Broadcast Within-Ring

In-ring broadcast is based on the distributed k-ary broadcast method. It is inspired by the distributed k-ary search method proposed by El-Ansary S. [19] etc.. The basic idea is that the broadcast starter will first generate a random number k by using Intel SGX API `sgx_read_rand()`, and then a k-ary broadcast spanning tree can be constructed in a distributed manner within the ring. The broadcast will then be triggered from the root node and messages are pushed down to every node in the tree. The parameter k is randomly chosen by the root node (within-ring broadcast initiator) via API `sgx_read_rand()`. The reason we choose to randomize the parameter k is that the network should be hidden from the attacker. If it keeps using the same parameter k, the routing pattern will be known easily by watching the network activities for a long time.

The spanning trees are formed by using the broadcaster (which is numbered 0) as the root node. Node 0 will connect to node $0+k^0$, $0+k^1$, $0+k^2$, and so on. Similarly, node 1 will connect to $1+k^0$, $1+k^1$, $1+k^2$, and so on. The pattern is: node i will connect to $i+k^0$, $i+k^1$, $i+k^2$, and so on. The overall time complexity of this method will be $O(logN)$, where N is the number of nodes in the ring. Fig. 5 shows that a ring of ten nodes in (a) forms a 2-ary spanning tree in (b) ($k=2$).

6 Attack Handling

In this section, we will first present the threat model we considered in the design of FRING. We then argue how Intel SGX is used and why it is necessary in FRING to ensure security under the threat model. Besides, we introduce two mechanisms to achieve pattern obfuscation in FRING network. The effectiveness is evaluated in §8.7.

6.1 Threat Model

We adopt a common threat model of using SGX in the network layer [75]. SGX assumes that an adversary can compromise any

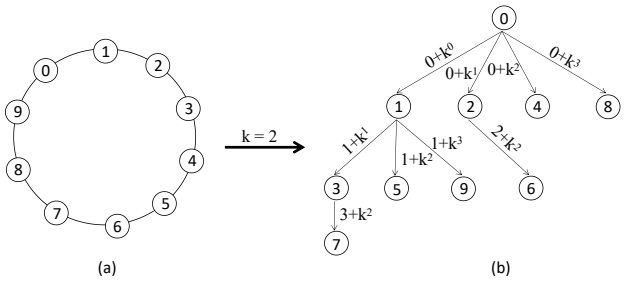


Fig. 5: Broadcast within-ring using distributed k-ary broadcast method. A spanning tree is constructed in a distributed way and the process starts from the broadcast initiator which is node 0 (the root node).

software components including the operating system, hypervisor, and firmware. Also, hardware components (e.g., memory and I/O devices) can be inspected by an attacker except for the CPU package itself. However, denial-of-service [76] is out of scope of this paper; a malicious privilege software could easily crash or halt the system. Thus in FRING's network, we assume that an attacker can obtain full control of the software and hardware (except the CPU package) of any node.

6.2 Use of Intel SGX

FRING uses Intel SGX attestations to build a trust base among blockchain nodes. The routing information is well saved by Intel SGX enclaves. The behavior of each node after receiving a message or when sending a message is protected by the enclaved execution. In addition, the parameter k used in within-ring broadcast and the IDs of the *representative* nodes are randomly generated by using the `sgx_read_rand` API [77].

Although grouping nodes that are geographically proximate with each other may lead to eclipse attacks, relying on Intel SGX, FRING could avoid eclipse attacks by removing misbehavior. In designing our protocols, we first perform a setup phase where each peer connects to every other peer in the network and then performs a series of steps. Every enclave first uses remote attestation to verify the correctness of the protocol executing on other peers. Next, they generate public/private key pairs inside the enclaves and exchange the public keys with each other. Then all the messages transmitted between any two enclaves can be signed to ensure integrity and authenticity. Moreover, the internal states of the program are also protected using enclaved execution. Outside the enclave, the hackers do not know the content of the intercepted messages. Therefore, the integrity of all messages including input/output/intermediate states is guaranteed. In this case, it is clear that an adversary cannot forge valid messages to bias the victims.

A malicious hacker can also intercept consensus messages, transactions or blocks directed to a victim. However, since he cannot identify heartbeat messages from other messages, he needs to intercept all messages. Missing heartbeat messages will kick the hacker out of the network. Therefore, an eclipsed attack can be avoided. Sybil attacks [78] could be potentially conducted by controlling all nodes from a city where an attacker subverts the blockchain system by creating a large number of pseudonymous identities and uses them to gain a large influence. However, there has been a line of work [78], [79], [80], [81] preventing such attacks which can be complementary to our work. Besides, taking control of the whole city is not an easy task; even if the attack succeeded, he can only subvert a ring in the city-level while at the upper level (state-level) the attacker only subverted one normal node. Taking control of the whole state or upper levels is much harder, attributing to the geographical locality-aware fractal ring structure of FRING. Since the nodes are structured in terms of location, an attacker

needs to control all nodes in a state or even in a country to compromise the whole system.

6.3 Pattern Obfuscation

To hide the existence of FRING *representative* nodes from outsiders, there are two mechanisms leveraged in FRING. First, fake messages are used to make the behavior of a *representative* node the same as that of a normal node. For a *representative* node, it will send messages of the same size to both one of the *representative* nodes and some of the past *representative* nodes in the upper level ring. For a normal node, it will send messages of the same size to both one of the *representative* nodes and some of its peers in the same level ring. When broadcasting messages within the ring, the random parameter k will hide the relative orders of each node. Apart from broadcasting to its children in the constructed distributed k-ary spanning tree, each node will randomly choose a node to send fake messages. As a result, the number of fake messages is linear to the number of nodes. All fake messages are of the same size as the real messages. Hackers may watch the packets being sent out from a node and being sent to the node for a long time. However, during one *representative* node service term, no difference can be discovered from the data collected.

Second, *representative* nodes of a ring will be elected for every service term. One *representative* node of the last service term will generate random IDs by using Intel SGX's `sgx_read_rand()` function [20], and then broadcast the nomination result within the ring and to the ring at the upper level. Due to both the limited *representative* node service term period and the same behavior during one service term, an outsider cannot differentiate *representative* nodes from normal nodes. Our evaluation shows that FRING's network topology and structure are well hidden from the outsiders. Packet analysis is done by using WireShark and the evaluation result is discussed in section §8.7.

7 Proof and Analysis

In order to show that FRING effectively reduces the message complexity and achieves fast broadcast convergence, we present the proof of the message complexity (§7.1) and the time complexity (§7.2) of broadcasting a message in FRING. Then we analyze the robustness (§7.3) of FRING to prove its sufficiency for blockchain systems.

7.1 Broadcast Message Complexity

Since FRING is a structured network and its neighbor selection is also deterministic, the number of hops needed is formally provable. Consider a network of size N. All nodes in the network are geographically distributed evenly. In FRING, every r nodes that are geographically near to each other will be gathered into a ring at the bottom level. If there are c *representative* nodes in each ring at the bottom level, then the number of nodes elected to be the normal nodes at the second level will be:

$$L(1) = \frac{cN}{r} \tag{1}$$

Then according to the protocol of FRING, the number of nodes elected to be the normal nodes at the first level will be the number of rings at the second level multiplied with parameter c, which is:

$$L(2) = \frac{c^2 N}{r^2} \tag{2}$$

Recursively, the number of nodes $L(h)$ and the number of rings $R(h)$ at level h will be:

$$L(h) = \frac{c^h N}{r^h}, R(h) = \frac{c^h N}{r^{h+1}} \tag{3}$$

Let there be T nodes in the top level (which are DNS seeds for a networked system), let $C = c/r$ be the *representative* node/normal node ratio at each ring, then the number of levels will be:

$$l = log_C \frac{T}{N} \tag{4}$$

To broadcast a message from a node in any ring at the bottom level, the message will first be disseminated upwards until it reached the ring at the top level. The number of messages used to reach any *representative* nodes at the top level is:

$$M_1(N) = 1 + l = 1 + log_C \frac{T}{N} \tag{5}$$

The number of messages used to broadcast from the top level rings recursively to all nodes in each ring at each level is:

$$M_2(N) = \sum_{i=0}^{l} N \frac{c}{r}^i = N \frac{1 - C^{l-1}}{1 - C} = \frac{CN - T}{C(1 - C)} \tag{6}$$

Hence the message complexity of a broadcast operation is $O(N)$, which is better than the current message complexity of both push Gossip ($O(NlogN)$) and push-pull Gossip ($O(NloglogN)$) [82].

7.2 Broadcast Time Complexity

Considering that the cost to transmit a data packet between two nodes in any two different continents is far larger than the cost to transmit between two nodes in the same city, let the cost of transmitting data packets in different rings at level h be $C(h)$, which is a mapping from levels to time cost constants. To broadcast a message from a node in any ring at the bottom level, the going-up path of the message to broadcast takes at most:

$$T_1(N) = \sum_{i=1}^{l} C(i) \tag{7}$$

After the message reaches any of the *representative* nodes at the top level, the message starts to broadcast downwards recursively in each ring (in parallel) from the top level to the bottom level. The time it takes to touch each individual node at the bottom level is:

$$T_2(N) = \sum_{i=0}^{l} C(i) \log_k \frac{L(i)}{R(i)} R(i) = \sum_{i=0}^{l} C(i) \log_k C^i N \tag{8}$$

Hence the time complexity of a broadcast operation is $O(logN)$, which is also the complexity of the number of rounds in the broadcast process. Although the time complexity of the Gossip algorithm is also $O(logN)$, more hops with high latency are incurred as the geographical locality is not considered and long tail latency issues can occur (as mentioned in §3.1). Besides, in the worst case, if each node only connects to those nodes that have the smallest proximity in the geographical locality, the total time complexity will be $O(N)$ [83], [84].

7.3 Fault Tolerance for Node Failure

In a blockchain system, individual machines are often under the control of a large number of heterogeneous users who may join or leave the network at any time. The dynamic of large-scale distributed systems and link failures cause problems to the message dissemination. Since the dissemination of membership information, transactions or blocks requires to reach all nodes, even the consensus protocol requires the message to reach at least half (PoW, PoS, DPoS, Ripple) or two-thirds (BFT, PBFT, Tendermint, Algorand BA*) [85], the P2P network under a blockchain system should try its best to reach as many nodes as possible. Under dynamic node joining/leaving and link failure, the network should be robust enough to cover as many as possible the remaining working nodes.

To analyze the robustness of FRING, we define reliability metric to be the ratio of covered nodes and remaining nodes. Let the probability of node failure be p, therefore, the number of nodes that cannot be reached is:

$$F(N) = \sum_{i=1}^{l} \sum_{j=i+1}^{l} (1-p)^j p^c R(i) r^i \tag{9}$$

As defined in §8.4, c is the number of representative nodes in a ring, r is the number of nodes in a ring, and $R(i)$ is the number of rings at level i.

Thus the reliability of FRING will be:

$$Reliability = \frac{N - F(N)}{(1-p)N} \quad (10)$$

In the context of geographically distributed blockchain systems, a 7-level FRING will reach 1/2 of all nodes on broadcasting a message if the node failure rate is less than 13.8%, and will reach 2/3 of all nodes if the failure rate is less than 13.3%. The number of levels is chosen to draw an analogy between the network topology and the geography notions (i.e., global, countries, states, cities, districts, etc.) because FRING is a geography-based overlay network taking inter-node latency into consideration. In the evaluation, we set the fault rate of all nodes to be from 10% to 70%, the reliability of FRING can reach the same level with Gossip when the fault rate is below 30%. And we found that in blockchain system context, Gossip is overly strong in terms of robustness. As the consensus protocol of a blockchain system only needs responses from half or two-thirds of all nodes to be honest, we could trade fault-tolerance off to gain efficiency.

8 EVALUATION

This section demonstrates the evaluation results of the implementation of FRING. Our evaluation aims to answer the following six key questions:

- §8.3 How effective can FRING improve the end-to-end throughput of blockchain systems?
- §8.4 How effective can FRING reduce the message complexity to complete a broadcast operation?
- §8.5 Can FRING improve the convergence time compared with state-of-the-art P2P networks?
- §8.4 Can FRING scale with respect to the number of nodes in the network?
- §8.6 Can FRING provide sufficient fault tolerance to blockchain systems?
- §8.7 Can FRING protect each node well so that *representative* nodes can not be easily differentiated from normal nodes?

Our evaluation is done on the AWS cloud [86]. We started 30 c4.4xlarge instances (VMs) running in the same region, each of which has 16 cores and 30 GB memory. Our FRING topology setting is 5∼7 layers (i.e., global, countries, states, cities, districts, etc.), which is enough for 2000 to 8000 nodes in the evaluation. The nodes are evenly distributed to each VM and the number of cities considered is different to achieve a 2000 to 8000 range of nodes in the FRING topology. For example, the total number of cities included is 40 and 60 when the number of nodes is 4000 and 6000, respectively. We refer readers to our open-source repository for more evaluation setup details. To simulate the real traffic, we simulate the network latency by delaying each received packet according to the distance of the sender and receiver. Since all VMs are located at the same datacenter region in AWS and we simulate the network latency between two FRING nodes on any VMs according to their distance on FRING's topology (because the distance is not real distance) by monitoring the distance-latency mapping across global AWS datacenters. Because AWS does not provide SGX hardware, we ran FRING in the SGX simulation mode [77]. The Intel SGX SDK provides simulation libraries to run application enclaves in simulation mode (Intel SGX hardware is not required). Although simulation mode does not require the Intel SGX support in the CPU, the applications developed in the simulation environment can be directly ported to the SGX hardware environment. We use typical Ethereum packets (measured from Etherscan [87]) as the data to broadcast: the size of one block is randomly chosen between 15 KB and 19KB, and the size of a transaction or consensus message is randomly chosen between 190 bytes and 210 bytes (in §8.3, §8.4, §8.5).

8.1 Implementation Details

We have implemented FRING in C++ with the only external dependence being `Protobuf` [88]. `Protobuf` is Google's language-neutral, platform-neutral, extensible mechanism for serializing structured data. It is used by FRING to serialize packet data. The FRING implementation contains roughly 3800 lines of code, not including tests, configuration on docker, comments, and blank lines.

A FRING application running on a server (node) mainly consists of four modules (Fig. 2), a `PeerDiscoverer` module, a `PeerManager` module, a `Broadcaster` module, and a socket API module. A `PeerDiscoverer` is responsible for bootstrapping the node. After the bootstrap, the `PeerManager` service and the `Broadcaster` service will be created and initiated for the node. The `PeerManager` stores peer information at each level of FRING and maintains the network structure by periodically sending heartbeat messages within the ring at the bottom level. The `Broadcaster` deals with incoming messages and is responsible for broadcasting messages. All modules are executed in SGX enclaves except the socket API module, which will send and receive data packets to and from the OS kernel network subsystem (Fig. 2).

The API of FRING simply contains two entries: (i) JoinNetwork: given IP address and listening port number, a node will join the network and start to work; and (ii) Broadcast: given the message to broadcast, a broadcast process is initiated. We open-source the prototype of FRING at https://github.com/tsc19/fring.

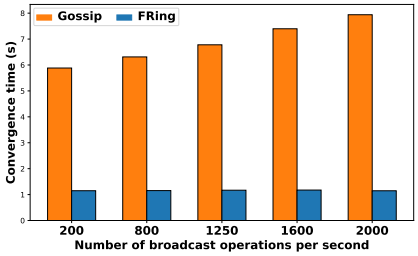
8.2 Comparison Baseline

As mentioned in §2, many blockchain systems are based on randomly generated P2P networks, which is difficult to compare with since the graph of the network can significantly vary in different instances. We choose a Kademlia-based P2P network and the Gossip broadcast algorithm as our comparison baseline, as it is used by the popular blockchain Ethereum. The baseline approaches are deployed and evaluated in AWS as well.

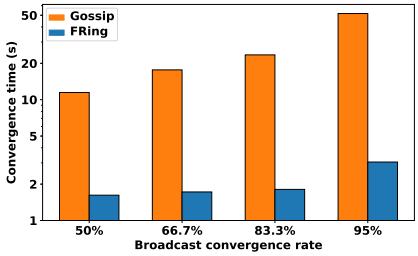
However, Ethereum's code is coupled with complex functions such as serialization and encryption that incur unclear overhead, making it potentially unfair to directly compare with FRING. Also, no isolated Gossip network layer can be used to compare FRING with. Thus, we implement our own version of Kademlia-based Gossip based on Ethereum's codebase. It is easier to be integrated with blockchain systems such as EOS and HLF. To validate our implementation, we test it with Ethereum's workload (with block size, message size, and transaction rate sampled from Etherscan [87] measurements between Jan.–July 2019) and then compare the behaviors and performances.

We run Ethereum with 200 nodes for six hours in our cluster. In our cluster, each machine has Linux 3.13.0, 40Gbps NIC, 2.60GHz Intel E3-1280 V6 CPU with SGX, 64GB memory, and 1TB SSD. On each node, we record the timestamps and node IDs of all broadcast operations as well as the timestamps of all sent and received messages. We then bootstrap a 200-node Kademlia-based network and let each node perform Gossip broadcast using the same broadcast timestamps recorded in the Ethereum case. In addition, we recorded the same information as in Ethereum for all nodes.

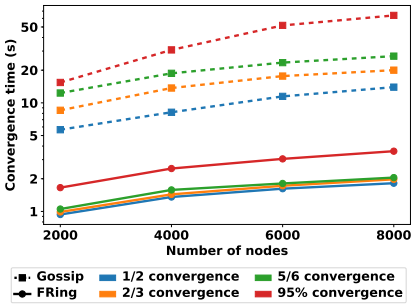
We count the total number of messages generated in the network in every ten minutes as a function of elapsed time. Statistics show that two implementations generate approximately the same amount of messages with the same input. The difference is smaller than 3%. We also discover that the average message complexity and average time complexity for broadcasting a



(a) Convergence time comparison with respect to the broadcast rate for 6K nodes and 2/3 convergence rate.



(b) Convergence time comparison with respect to the convergence rate for 6K nodes and 200 Tps Tx generation rate.



(c) Scalability analysis on the number of nodes with 200 tps Tx generation rate.

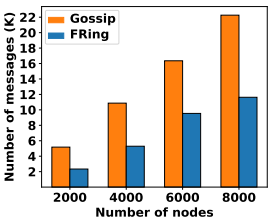
Fig. 6: Comparison of convergence time between Gossip and FRING.

message are almost the same (4% variance). More rigorously, we conduct the one-tailed Fisher's statistical hypothesis testing with the null hypothesis being that the two overlay networks have the same behavior in terms of the time-indexed message complexity, which refers to the number of messages generated per ten minutes and covers both message complexity and the time complexity. We observed a p-value higher than 0.0782 that is higher than 0.05 (with significant level alpha 5%), indicating strong evidence for the null hypothesis. Thus, we are certain that it is reasonable to use the self-implemented Kademlia-based Gossip instead of Ethereum's code to compare with the implementation of FRING. Therefore in the following sections, we refer to Gossip as our implementation of Kademlia-based Gossip, which is our comparison baseline.

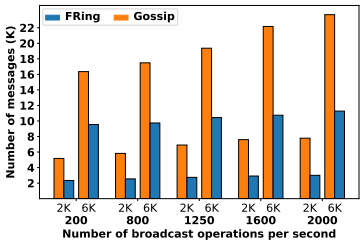
8.3 End-to-end Throughput Comparison

We leverage EOS [5] and HLF [3] as two representative examples and evaluate the performance improvement that FRING can bring to the blockchain system as a whole. EOS is a proof-of-stake-based public blockchain system and HLF is a private blockchain system using Raft to achieve consensus. Two network settings (100Mbps and 200Mbps) are used at each node. In Fig. 1, it shows that by using FRING as the underlying P2P network, the throughputs of EOS and HLF are increased by up to 2.2× and 2.1× respectively. Specifically, when the transaction generation rate grows or the network is more crowded, the advantage of FRING becomes bigger. The throughputs of EOS-FRing and HLF-Fring at 100 Mbps are even nearly the same as the throughputs of EOS-Gossip and HLF-Gossip at 200 Mbps. Low bandwidth settings and numerous client input transactions limit the throughput of EOS and HLF. However, we owe the benefits to FRING's fast convergence and low message redundancy (they are evaluated in §8.4 and §8.5 respectively). FRING reduces traffic congestion and thus improves the end-to-end throughput.

There is a fundamental trade-off of performance and robustness in distributed systems [89], [90] which we found that also exists in blockchain systems. Blockchain systems aim at achieving an acceptable trade-off between the conflicting goals of broadcast convergence performance versus robustness to uncertainty or faults. We found that the lower requirement to fault tolerance in blockchain systems yields a richer design space. The state-of-the-art P2P network and its broadcast algorithm Gossip provide extremely high robustness and can tolerate up to 90% node failure, at the cost of higher message complexity and slower convergence especially under congestion. By lowering the robustness to node failures (sufficient fault tolerance in blockchain systems), FRING generates lower message redundancy and provides faster convergence and thus higher end-to-end throughput.



(a) Message complexity with respect to the number of nodes under 200 operations per second rate.



(b) Message complexity with respect to broadcast operation rate from 200 to 2000 operations per second rate.

Fig. 7: Comparison of message complexity between Gossip and FRING in terms of scalability on both the number of nodes and broadcast operation rate.

8.4 Message Complexity

In this section, we show that the average number of messages generated in the whole FRING network to broadcast a message increases linearly. FRING and the baseline overlay network are evaluated in isolation and the workload is exactly the configuration described in §8 evaluation setup. As shown in Fig. 7a, with the number of nodes in the network increasing, the total number of messages generated in the whole FRING network increases linearly. Also, the difference between FRING and Gossip is enlarging as the number of nodes increases.

Fig. 7b shows that when the number of nodes is fixed, the average number of messages FRING generates to broadcast a message is around half of the Gossip generates. With the broadcast operation rate increasing, Gossip generates more messages to deal with packet loss due to traffic congestion. This inherent redundancy in turn leads to cascading accumulated on-flight messages. However, the fast convergence of FRING makes traffic congestion harder to happen. Consequently, the advantage of FRING is increasing with the broadcast operation rate growing.

8.5 Time Complexity (Latency)

In this section, we show that although the time complexities of FRING and Gossip are both logarithm, yet the average time used by FRING to broadcast a message is still far lower than Gossip. The reason is that Gossip does not take geographical locality into consideration. In addition, under different conditions of convergence (1/2, 2/3, 5/6, 95%), it takes Gossip significantly more time to converge. Since Gossip is non-deterministic, it has a long "tail" to cover all nodes in the network.

Fig. 6a and Fig. 6b show that the average convergence time of FRING is six to ten times shorter than Gossip when the number of nodes is 6k. Fig. 6c shows that both Gossip and FRING are scalable with respect to the number of nodes in the network. However, FRING has a lower average convergence

TABLE 3: Broadcast hop analysis for Gossip and FRING.

Hop Type	FRING	Gossip
0∼40 ms (Within District)	75.49% (4194)	29.50% (3026)
40∼80 ms (Between District)	19.94% (1108)	26.10% (2677)
80∼120 ms (Between City)	4.250% (236)	18.07% (1853)
120∼160 ms (Between State)	0.289% (16)	15.30% (1569)
160∼200 ms (Between Country)	0.054% (3)	11.03% (1131)
Total # of Hops	5557	10256

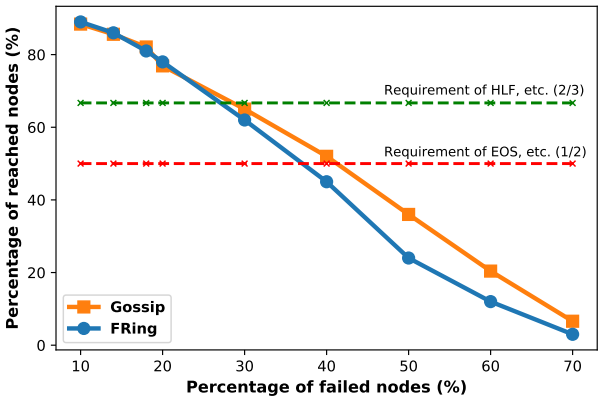


Fig. 8: Fault tolerance comparison between FRING and Gossip. FRING is sufficient for all blockchain systems.

time to broadcast a message compared with Gossip. In addition, the more nodes there are in the network and the higher the cover ratio is, the larger the advantage is. To conclude, FRING has a lower convergence time than Gossip.

To understand why FRING reduces convergence time so much, we counted the number of each type of hops (Table 3) for broadcasting a message on both FRING and Gossip (with 4000 nodes and 200 tps). In FRING, most hops are those with low latency. The higher the latency of the hop is, the less likely it is chosen to be one of the broadcast paths. However in Gossip, since geographical locality is not considered, more hops with high latency are incurred.

8.6 Fault Tolerance

We evaluated the fault-tolerance of Gossip and FRING in the AWS cloud as well. To simulate the node failure, we send a signal to a randomly selected subset of nodes so that they do not respond to broadcast requests. Then we perform broadcast in the network and collect the logs for statistics.

Figure 8 shows that FRING has sufficient fault tolerance as the underlying P2P network of all blockchain systems except Ripple [52] whose consensus protocol requires at least 4/5 node reachability. The tolerance of node failure of well-known blockchain consensus protocols are listed in Table 2. The most common requirements are to reach at least 1/2 (e.g. Ethereum, PeerCoin, EOS) and 2/3 (e.g. HLF, Tendermint) living nodes, demonstrated in Fig. 8 as the red line and green line respectively. Above such requirements, Fig. 8 shows that FRING can reach nearly the same percentage of nodes as Gossip. Specifically, the intersections of the curves of FRING and Gossip with the red line are close to each other, which means that the percentage of failed nodes that FRING can tolerate to satisfy the requirement is not much lower than Gossip.

Below those requirements, although the FRING deteriorates faster than Gossip, maintaining the fault tolerance of broadcast is already meaningless since the consensus protocol that we are supporting has failed already. An interesting point is that, when the node failure rate is more than 70%, the reachability of both Gossip and FRING is nearly the same. It's because the network graph is already broken into several sub-graphs. Low connectivity leads to poor performance for both broadcast approaches. However, a 20% node failure is already an extreme case for a normal functioning application. Previous work [91] shows that 15% high packet loss rate already significantly downgrades the quality of most Internet applications. Since node failures potentially cause the loss of a large number of small packets, the impact of 20 % node failure is considered even worse than the same percentage of packet loss in previous work [92]. Even under tough node failure scenarios (more than 20% node failures), FRING still maintained a similar high node reachability rate as Gossip, which is sufficient for both EOS and HLF's consensus protocols.

TABLE 4: Receive-Packet Analysis of Node in FRING

Node Type	$\sim 17KB$	$\sim 200B$	$< 150B$
Normal node in one term	33.10%	58.60%	5.90%
Representative node in one term	34.00%	61.20%	4.50%
Node at all time	33.70%	60.90%	4.60%

TABLE 5: Send-Packet Analysis of Node in FRING

Node Type	$\sim 17KB$	$\sim 200B$	$< 150B$
Normal node in one term	35.50%	58.60%	5.60%
Representative node in one term	34.40%	59.20%	4.70%
Node at all time	34.60%	59.10%	5.10%

8.7 Security

We use WireShark [93], [94] to watch the packet being sent out from a node and received by the node. We count three types of packets: around 17 KB, around 200 bytes, and less than 150 bytes, since they represent three types of messages correspondingly, i.e., the block, the transaction, and other messages including control messages or membership messages. We found that during one service term of a *representative* node, *representative* nodes of a ring cannot be differentiated from the normal nodes.

By watching and recording for a long time, the overall packet analysis statistics show that all nodes have the same percentages of sent and received three types of messages (See Table 5 and Table 4). Therefore, due to the short service term of *representative* nodes and the similar percentages of all types of messages, their traffic patterns are indistinguishable (E.g. in Table 4, the packets sent from a normal node in one term, a representative node in one term, and any node at all time are 33.1%, 34.0%, and 33.7% respectively.). Thus it is hard for an outsider of the system to differentiate *representative* nodes from normal nodes.

9 Discussion

9.1 Threats to Validity

Threats to validity for the proposed approach and experiments are discussed in this section which include internal validity, external validity, and construct validity [95]. Threats to the internal validity concern about the internal factors of our research experiment design. As the code base for any blockchain system is highly coupled and monolithic (e.g., the overlay networking module/layer is associated with serialization and encryption that incur unclear overhead), we implemented our own version of Gossip network (see §8.2) and use it to compare with FRING. Although it is verified that our version of Gossip network is statistically the same as the one implemented as part of Ethereum in terms of both message and time complexity, it is not theoretically proved. Other than that, we make sure that the only difference in the experiments is caused by the substitution of the overlay network to fairly compare FRING with the other baseline approaches. External validity is concerned with the generalization of the results. In our experiments, we evaluated the end-to-end performance of two representative blockchain systems, EOS and HLF, without implementing and running all the other blockchain systems. We admit that the replicated study is needed for generalization. However, we believe FRING

is a general solution for all blockchain systems since they are all based on the layered structure as mentioned in Fig. 2. On the other hand, to generalize the evaluation results (§8.4, §8.5) using Ethereum's typical workload, we characterize the same experiments by sweeping over the block size up to 1 MB (block size limit) and found similar results. However, Ethereum's workload could be random and bursty, which is a potential threats to generalization. Construct validity is the other main threat related to the software engineering experimentation work which is the relationship between theory and observations [95]. In the context of this research, all performance-wise measurements such as latency and throughput are measured based on the output from the running FRING program (where we inserted the code to print the timestamps of each event). We cannot ignore that there could be time drift in asynchronous distributed systems. The total message measurements are counted from the message sender side to include lost messages as well.

9.2 Limitations

Maintainability. Although FRING is a self-managed overlay network, network structure management is still required and additional messages are needed compared to the Gossip network where neighbor selection and node selection for relaying the message can be simple and random. Besides, the pattern obfuscation and representative node rotation techniques proposed to improve the robustness of FRING also introduce more management overhead. However, both theoretical analysis and empirical experiments show that the overall message overhead of FRING is still less than Gossip. Hence we argue that the benefits of FRING's structured topology overbear the compromised maintainability problem compared to Gossip.

Deployment Feasibility. FRING can be adopted in general blockchain systems by replacing their network layers but a joining node is required 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 public blockchains [96] and permissioned blockchains [4], [68], [97], [98], also use SGX. Apart from Intel SGX, AMD has its Secure Virtual Machine (SVM) architecture [68] and memory region encryption technology [69]; ARM has the TrustZone technique [70], which all provide similar protection. In addition, FRING targets large-scale, geo-distributed blockchain systems like a global payment system [99]), while for small-scale deployments in a single data center, existing overlay networks could be equally suitable without the maintainability overhead.

Experiment Limitations. Large-scale experiments are done in AWS datacenters at one region by simulating the per-hop network latency variations based on the abstract distance between every two nodes. Realistic network simulation is known to be a hard problem and we leave the actual deployment of a FRING network to the future work.

10 Conclusion

This paper presented FRING, the first geography-based P2P overlay network that achieves fast and robust broadcast for blockchain systems. By trading off excessive robustness, FRING improves the throughput of blockchain systems by increasing broadcast message efficiency and convergence time. Evaluation and analysis show that FRING is efficient, sufficiently robust, and secure. Overall, FRING increases the overall throughput of EOS and HLF by up to 2.2× and 2.1× respectively. An implementation of FRING is open-sourced at https://github.com/tsc19/fring.

Acknowledgments

We thank all reviewers for their helpful comments. This work is supported in part by an HKU-SCF FinTech Academy R&D Funding Scheme in 2021, HK RGC GRF (17202318, 17207117), HK ITF (GHP/169/20SZ), the Pujiang National Lab (Heming Cui is a courtesy researcher in this lab), and the HKU and ISCAS Joint Lab for Intelligent System Software.

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized Business Review*, p. 9, 2008.
[2] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, pp. 1–32, 2014.
[3] 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*. ACM, 2018, p. 30.
[4] Intel, "Introduction to hyperledger sawtooth, v1.1.2," https://sawtooth.hyperledger.org/docs/core/releases/latest/introduction.html.
[5] EOSIO, "Eos.io technical white paper version no.2." [Online]. Available: https://github.com/EOSIO/Documentation/blob/master/TechnicalWhitePaper.md
[6] A. Baliga, "Understanding blockchain consensus models," 2017.
[7] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *Annual International Cryptology Conference*. Springer, 2017, pp. 357–388.
[8] M. Milutinovic, W. He, H. Wu *et al.*, "Proof of luck: An efficient blockchain consensus protocol," in *Proceedings of the 1st Workshop on System Software for Trusted Execution*. ACM, 2016, p. 2.
[9] C. Cachin, "Architecture of the hyperledger blockchain fabric," in *Workshop on Distributed Cryptocurrencies and Consensus Ledgers*, vol. 310, 2016.
[10] J. Leitao, J. Pereira, and L. Rodrigues, "Epidemic broadcast trees," in *2007 26th IEEE International Symposium on Reliable Distributed Systems (SRDS 2007)*. IEEE, 2007, pp. 301–310.
[11] A. Jüttner and Á. Magi, "Tree based broadcast in ad hoc networks," *Mobile Networks and Applications*, vol. 10, no. 5, pp. 753–762, 2005.
[12] E. K. Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford, "Enhancing bitcoin security and performance with strong consistency via collective signing," in *25th USENIX Security Symposium (USENIX Security 16)*, 2016, pp. 279–296.
[13] J. Leitao, J. Pereira, and L. Rodrigues, "Gossip-based broadcast," in *Handbook of Peer-to-Peer Networking*. Springer, 2010, pp. 831–860.
[14] B. Xu, D. Luthra, Z. Cole, and N. Blakely, "Eos: An architectural, performance, and economic analysis," *Bitmex*, 2018.
[15] C. Li, P. Li, W. Xu, F. Long, and A. C.-c. Yao, "Scaling nakamoto consensus to thousands of transactions per second," *arXiv preprint arXiv:1805.03870*, 2018.
[16] J. Leitao and L. Pereira, "Hyparview: A membership protocol for reliable gossip-based broadcast," in *Dependable Systems and Networks, 2007. DSN'07. 37th Annual IEEE/IFIP International Conference on*. IEEE, 2007, pp. 419–429.
[17] M.-J. Lin and K. Marzullo, "Directional gossip: Gossip in a wide area network," in *European Dependable Computing Conference*. Springer, 1999, pp. 364–379.
[18] A. Montresor, "Intelligent gossip," in *Intelligent Distributed Computing, Systems and Applications*. Springer, 2008, pp. 3–10.
[19] S. El-Ansary, L. O. Alima, P. Brand, and S. Haridi, "Efficient broadcast in structured P2P networks," in *International workshop on Peer-to-Peer systems*. Springer, 2003, pp. 304–314.
[20] V. Costan and S. Devadas, "Intel SGX explained." *IACR Cryptology ePrint Archive*, vol. 2016, no. 086, pp. 1–118, 2016.
[21] D. Yang, C. Long, H. Xu, and S. Peng, "A review on scalability of blockchain," in *Proceedings of the 2020 The 2nd International Conference on Blockchain Technology*, 2020, pp. 1–6.
[22] Q. Zhou, H. Huang, Z. Zheng, and J. Bian, "Solutions to scalability of blockchain: A survey," *IEEE Access*, vol. 8, pp. 440–455, 2020.
[23] J. Xie, F. R. Yu, T. Huang, R. Xie, J. Liu, and Y. Liu, "A survey on the scalability of blockchain systems," *IEEE Network*, vol. 33, no. 5, pp. 166–173, 2019.
[24] L. Hoxha, "Hashgraph the future of decentralized technology and the end of blockchain," *European Journal of Formal Sciences and Engineering*, vol. 1, no. 2, pp. 29–32, 2018.
[25] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford, "Omniledger: A secure, scale-out, decentralized ledger via sharding," in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 583–598.

[26] L. Bach, B. Mihaljevic, and M. Zagar, "Comparative analysis of blockchain consensus algorithms," in *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. IEEE, 2018, pp. 1545–1550.

[27] P. Thakkar, S. Nathan, and B. Viswanathan, "Performance benchmarking and optimizing hyperledger fabric blockchain platform," in *2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE, 2018, pp. 264–276.

[28] S. Bano, A. Sonnino, M. Al-Bassam, S. Azouvi, P. McCorry, S. Meiklejohn, and G. Danezis, "Consensus in the age of blockchains," *arXiv preprint arXiv:1711.03936*, 2017.

[29] J. Qi, X. Chen, Y. Jiang, J. Jiang, T. Shen, S. Zhao, S. Wang, G. Zhang, L. Chen, M. H. Au *et al.*, "Bidl: A high-throughput, low-latency permissioned blockchain framework for datacenter networks," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles (SOSP 2021)*, 2021, pp. 18–34.

[30] C. Dana, D. Li, D. Harrison, and C.-N. Chuah, "Bass: Bittorrent assisted streaming system for video-on-demand." in *MMSP*, vol. 5, 2005, pp. 1–4.

[31] N. Carlsson and D. L. Eager, "Peer-assisted on-demand streaming of stored media using bittorrent-like protocols," in *International Conference on Research in Networking*. Springer, 2007, pp. 570–581.

[32] J. Lv, X. Cheng, Q. Jiang, J. Ye, T. Zhang, S. Lin, and L. Wang, "Livebt: Providing video-on-demand streaming service over bittorrent systems," in *Parallel and Distributed Computing, Applications and Technologies, 2007. PDCAT'07. Eighth International Conference on*. IEEE, 2007, pp. 501–508.

[33] J. J.-D. Mol, J. A. Pouwelse, M. Meulpolder, D. H. Epema, and H. J. Sips, "Give-to-get: free-riding resilient video-on-demand in P2P systems," in *Multimedia Computing and Networking 2008*, vol. 6818. International Society for Optics and Photonics, 2008, p. 681804.

[34] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi *et al.*, "Splitstream: high-bandwidth multicast in cooperative environments," in *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5. ACM, 2003, pp. 298–313.

[35] D. Kostić, A. Rodriguez, J. Albrecht, and A. Vahdat, "Bullet: High bandwidth data dissemination using an overlay mesh," in *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5. ACM, 2003, pp. 282–297.

[36] V. Pai, K. Kumar *et al.*, "Chainsaw: Eliminating trees from overlay multicast," in *International Workshop on Peer-to-Peer Systems*. Springer, 2005, pp. 127–140.

[37] S. A. Baset and H. Schulzrinne, "An analysis of the skype peer-to-peer internet telephony protocol," *arXiv preprint cs/0412017*, 2004.

[38] S. Saroiu, K. P. Gummadi, and S. D. Gribble, "Measuring and analyzing the characteristics of napster and gnutella hosts," *Multimedia systems*, vol. 9, no. 2, pp. 170–184, 2003.

[39] M. Ripeanu, "Peer-to-peer architecture case study: Gnutella network," in *Peer-to-Peer Computing, 2001. Proceedings. First International Conference on*. IEEE, 2001, pp. 99–100.

[40] N. S. Good and A. Krekelberg, "Usability and privacy: a study of kazaa P2P file-sharing," in *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 2003, pp. 137–144.

[41] Bitcoin.org, "Bitcoin developer guide." [Online]. Available: https://bitcoin.org/en/developer-guide

[42] Y. Mao, S. Deb, S. B. Venkatakrishnan, S. Kannan, and K. Srinivasan, "Perigee: Efficient peer-to-peer network design for blockchains," in *Proceedings of the 39th Symposium on Principles of Distributed Computing*, 2020, pp. 428–437.

[43] "Node discovery protocol v4," https://github.com/ethereum/devp2p/blob/master/discv4.md.

[44] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the xor metric," in *International Workshop on Peer-to-Peer Systems*. Springer, 2002, pp. 53–65.

[45] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," *ACM SIGCOMM Computer Communication Review*, vol. 31, no. 4, pp. 149–160, 2001.

[46] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing*. Springer, 2001, pp. 329–350.

[47] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz, "Tapestry: A resilient global-scale overlay for service deployment," *IEEE Journal on selected areas in communications*, vol. 22, no. 1, pp. 41–53, 2004.

[48] P. T. Eugster, R. Guerraoui, A.-M. Kermarrec, and L. Massoulié, "Epidemic information dissemination in distributed systems," *Computer*, vol. 37, no. 5, pp. 60–67, 2004.

[49] M. H. Islam, S. Waheed, and I. Zubair, "An efficient gossip based overlay network for peer-to-peer networks," in *Ubiquitous and Future Networks, 2009. ICUFN 2009. First International Conference on*. IEEE, 2009, pp. 62–67.

[50] I. Gupta, K. P. Birman, and R. van Renesse, "Fighting fire with fire: using randomized gossip to combat stochastic scalability limits," *Quality and Reliability Engineering International*, vol. 18, no. 3, pp. 165–184, 2002.

[51] H. Zhuge and X. Li, "RSM-based gossip on P2P network," in *International Conference on Algorithms and Architectures for Parallel Processing*. Springer, 2007, pp. 1–12.

[52] "Consensus of xrp ledger (ripple)." [Online]. Available: https://developers.ripple.com/consensus.html

[53] [Online]. Available: https://github.com/peercoin/peercoin

[54] F. Schuh and D. Larimer, "BitShares 2.0: General overview," *BitShares*, 2015. [Online]. Available: http://docs.bitshares.org/_downloads/bitshares-general.pdf

[55] J. Kwon, "Tendermint: Consensus without mining," *v. 0.6*, 2014.

[56] J. Liu, S. G. Rao, B. Li, and H. Zhang, "Opportunities and challenges of peer-to-peer internet video broadcast," *Proceedings of the IEEE*, vol. 96, no. 1, pp. 11–24, 2008.

[57] V. N. Padmanabhan, H. J. Wang, P. A. Chou, and K. Sripanidkulchai, "Distributing streaming media content using cooperative networking," in *Proceedings of the 12th international workshop on Network and operating systems support for digital audio and video*. ACM, 2002, pp. 177–186.

[58] X. Zhang, J. Liu, B. Li, and Y.-S. Yum, "Coolstreaming/donet: A data-driven overlay network for peer-to-peer live media streaming," in *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, vol. 3. IEEE, 2005, pp. 2102–2111.

[59] I. Anati, S. Gueron, S. Johnson, and V. Scarlata, "Innovative technology for cpu based attestation and sealing," in *Proceedings of the 2nd international workshop on hardware and architectural support for security and privacy*, vol. 13. ACM New York, NY, USA, 2013.

[60] F. McKeen, I. Alexandrovich, I. Anati, D. Caspi, S. Johnson, R. Leslie-Hurd, and C. Rozas, "Intel® software guard extensions (Intel® SGX) support for dynamic memory management inside an enclave," in *Proceedings of the Hardware and Architectural Support for Security and Privacy 2016*. ACM, 2016, p. 10.

[61] E. Bresson, O. Chevassut, D. Pointcheval, and J.-J. Quisquater, "Provably authenticated group diffie-hellman key exchange," in *Proceedings of the 8th ACM conference on Computer and Communications Security*. ACM, 2001, pp. 255–264.

[62] "Software guard extensions reference." [Online]. Available: https://kib.kiev.ua/x86docs/Intel/SGX/329298-001.pdf

[63] J. Aumasson and L. Merino, "SGX secure enclaves in practice–security and crypto review," *Black Hat*, 2016.

[64] M. Hamburg, P. Kocher, and M. E. Marson, "Analysis of intel's ivy bridge digital random number generator," *Cryptography Research, Inc.*, 2012. [Online]. Available: http://www.cryptography.com/public/pdf/Intel_TRNG_Report_20120312.pdf

[65] M. H. Mofrad and A. Lee, "Leveraging Intel SGX to create a nondisclosure cryptographic library," *arXiv preprint arXiv:1705.04706*, 2017.

[66] L. Chen, L. Xu, N. Shah, Z. Gao, Y. Lu, and W. Shi, "On security analysis of proof-of-elapsed-time (poet)," in *International Symposium on Stabilization, Safety, and Security of Distributed Systems*. Springer, 2017, pp. 282–297.

[67] X. Chen, S. Zhao, C. Wang, S. Zhang, and H. Cui, "GEEC: Scalable, Efficient, and Consistent Consensus for Blockchains," *ArXiv e-prints*, Aug. 2018.

[68] X. Chen, S. Zhao, J. Qi, J. Jiang, H. Song, C. Wang, T. O. Li, T. H. Chan *et al.*, "Efficient and DoS-resistant consensus for permissioned blockchains," *Performance Evaluation*, p. 102244, 2021.

[69] S. Mofrad, F. Zhang, S. Lu, and W. Shi, "A comparison study of Intel SGX and AMD memory encryption technology," in *Proceedings of the 7th International Workshop on Hardware and Architectural Support for Security and Privacy*, 2018, pp. 1–8.

[70] M. A. Mukhtar, M. K. Bhatti, and G. Gogniat, "Architectures for security: A comparative analysis of hardware security features in Intel SGX and ARM TrustZone," in *2019 2nd International Conference on Communication, Computing and Digital systems (C-CODE)*. IEEE, 2019, pp. 299–304.

[71] N. Berendea, H. Mercier, E. Onica, and E. Riviere, "Fair and efficient gossip in hyperledger fabric," in *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2020, pp. 190–200.

[72] A. Singh *et al.*, "Eclipse attacks on overlay networks: Threats and defenses," in *In IEEE INFOCOM*. Citeseer, 2006.

[73] M. Castro, P. Druschel *et al.*, "Secure routing for structured peer-to-peer overlay networks," *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 299–314, 2002.

[74] J. A. D. Donet, C. Pérez-Sola, and J. Herrera-Joancomartí, "The bitcoin P2P network," in *International Conference on Financial Cryptography and Data Security*. Springer, 2014, pp. 87–102.

[75] S. Kim, Y. Shin, J. Ha, T. Kim, and D. Han, "A first step towards leveraging commodity trusted execution environments for network applications," in *Proceedings of the 14th ACM Workshop on Hot Topics in Networks*. ACM, 2015, p. 7.

[76] X. Chen, S. Zhao, J. Qi, J. Jiang, H. Song, C. Wang, T. On Li, T.-H. Hubert Chan, F. Zhang, X. Luo, S. Wang, G. Zhang, and H. Cui, "Efficient and dos-resistant consensus for permissioned blockchains," *SIGMETRICS Performance Evaluation Review*, vol. 49, no. 3, p. 61–62, mar 2022.

[77] "Intel software guard extensions SDK for Linux OS." [Online]. Available: https://download.01.org/intel-sgx/linux-2.2/docs/Intel_SGX_Developer_Reference_Linux_2.2_Open_Source.pdf

[78] H. Rowaihy, W. Enck, P. McDaniel, and T. La Porta, "Limiting Sybil attacks in structured P2P networks," in *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*. IEEE, 2007, pp. 2596–2600.

[79] B. N. Levine, C. Shields, and N. B. Margolin, "A survey of solutions to the Sybil attack," *University of Massachusetts Amherst, Amherst, MA*, vol. 7, p. 224, 2006.

[80] H. Yu, M. Kaminsky, P. B. Gibbons, and A. Flaxman, "SybilGuard: defending against Sybil attacks via social networks," in *Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, 2006, pp. 267–278.

[81] R. John *et al.*, "A survey of techniques to prevent Sybil attacks," in *International Conference on Soft-Computing and Networks Security (ICSNS)*. IEEE, 2015, pp. 1–6.

[82] M. Jelasity, "Gossip," in *Self-organising Software*. Springer, 2011, pp. 139–162.

[83] S. Kashyap, S. Deb, K. Naidu, R. Rastogi, and A. Srinivasan, "Efficient gossip-based aggregate computation," in *Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM, 2006, pp. 308–317.

[84] S. Kaune, T. Lauinger, A. Kovacevic, and K. Pussep, "Embracing the peer next door: Proximity in kademlia," in *Peer-to-Peer Computing, 2008. P2P'08. Eighth International Conference on*. IEEE, 2008, pp. 343–350.

[85] Z. Zheng, S. Xie, H.-N. Dai, and H. Wang, "Blockchain challenges and opportunities: A survey," *Work Pap.–2016*, 2016.

[86] "Aws instance type list." [Online]. Available: https://aws.amazon.com/cn/ec2/instance-types/

[87] "Etherscan." [Online]. Available: https://etherscan.io/

[88] "Protobuf github repo." [Online]. Available: https://github.com/protocolbuffers/protobuf

[89] W. Vogels, "Eventually consistent: Building reliable distributed systems at a worldwide scale demands trade-offs? between consistency and availability." *Queue*, vol. 6, no. 6, pp. 14–19, 2008.

[90] K. Yang, Y. Wu, J. Huang, X. Wang, and S. Verdú, "Distributed robust optimization for communication networks," in *IEEE INFOCOM 2008-The 27th Conference on Computer Communications*. IEEE, 2008, pp. 1157–1165.

[91] "Definition of packet loss in wikipedia." [Online]. Available: https://en.wikipedia.org/wiki/Packet_loss

[92] P. Gill, N. Jain, and N. Nagappan, "Understanding network failures in data centers: measurement, analysis, and implications," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 350–361, 2011.

[93] L. Chappell and G. Combs, *Wireshark network analysis: the official Wireshark certified network analyst study guide*. Protocol Analysis Institute, Chappell University, 2010.

[94] C. Sanders, *Practical packet analysis: Using Wireshark to solve real-world network problems*. No Starch Press, 2017.

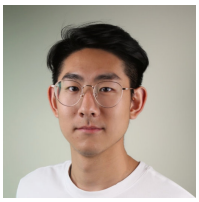
[95] R. Feldt and A. Magazinius, "Validity threats in empirical software engineering research: An initial survey." in *Seke*, 2010, pp. 374–379.

[96] F. Zhang, I. Eyal, R. Escriva, A. Juels, and R. Van Renesse, "REM: Resource-efficient mining for blockchains," in *26th USENIX Security Symposium (USENIX Security 2017)*, 2017, pp. 1427–1444.

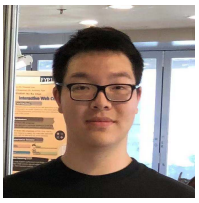
[97] M. Russinovich, E. Ashton, C. Avanessians, M. Castro, A. Chamayou, S. Clebsch, M. Costa, C. Fournet, M. Kerner, S. Krishna *et al.*, "CCF: A framework for building confidential verifiable replicated services," *Technical report, Microsoft Research and Microsoft Azure*, 2019.

[98] G. S. Veronese, M. Correia, A. N. Bessani, L. C. Lung, and P. Verissimo, "Efficient byzantine fault-tolerance," *IEEE Transactions on Computers*, vol. 62, no. 1, pp. 16–30, 2011.

[99] M. Baudet, A. Ching, A. Chursin, G. Danezis, F. Garillot, Z. Li, D. Malkhi, O. Naor, D. Perelman, and A. Sonnino, "State machine replication in the Libra blockchain," *The Libra Assn., Tech. Rep*, 2019.



Haoran Qiu is a Ph.D. candidate in Computer Science at the University of Illinois at Urbana-Champaign (2019–now). He is under the supervision of Prof. Ravishankar K. Iyer. His research interests include distributed systems and cloud datacenter reliability. He received his Bachelor of Engineering degree in Computer Science from the University of Hong Kong.

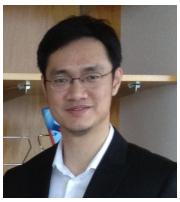


Tao Ji is currently pursuing Ph.D. degree in Computer Science at the University of Texas Austin. He is under the supervision of Prof. Aditya Akella. He received the M.S. in Computer Science in University of Wisconsin-Madison in 2021 and B.Eng. in Computer Science at the University of Hong Kong in 2019. His is broadly interested networked systems.

Shixiong Zhao received his Bachelor degree in HKU and his Master degree in HKUST. He is currently a Ph.D. student in Computer Science at HKU (2017–now). He is under the supervision of Prof. Heming Cui. His research interests include distributed systems for high performance computing, distributed systems, and system security.

Xusheng Chen received his Ph.D. degree and Bachelor degree in HKU. He is currently a researcher at Huawei Cloud. His research interests include distributed consensus protocols, distributed systems, and system security.

Ji Qi received the B.S. degree from Beijing Institute of Technology and the M.S. degree from Tsinghua University. He is currently a Ph.D. student in Computer Science at HKU under the supervision of Prof. Heming Cui. His research interests include domain-specific modeling, distributed system, and cloud computing. He is a student member of the IEEE.



Heming Cui is an associate professor in computer science of HKU. His research interests include operating systems, programming languages, distributed systems, and cloud computing, with a particular focus on building software infrastructures and tools to improve reliability and security of real-world software. He is a member of the IEEE.

Sen Wang received the B.S. degree from the University of Science and Technology of China (USTC), Hefei, China, in 2005, the M.S. degree from the Chinese Academy of Sciences (CAS), Beijing, China, in 2008, and the Ph.D. degree from Tsinghua University, Beijing, China, in 2014, all in computer science. His research interests include information-centric networking, Security and Privacy in Artificial Intelligence, Distributed Systems and so on.