# HGFRR: Hidden Geographical Fractal Random Ring

Anonymous Authors

## Abstract

Blockchain systems are used to record transactions among parties without a central authority. Since the very first instance of a blockchain systems, Bitcoin, tremendous amount of blockchain systems with various consensus protocols are designed and implemented to achieve fast transaction rate. With the transaction rate increasing, experiments and studies show that network layer become the bottleneck on the way to further improve blockchain system efficiency. However, current blockchain systems are based on unstructured, structured, or hybrid Peer-to-Peer networks. Gossiping messages on such networks creates repeated messages and leads to traffic congestion when transaction rates grows. In addition, lack of attention paid to geographical locality and security in the network layer also limits the improvement of the P2P network underlying blockchain systems.

In this paper, we present **H**idden **G**eographical **F**ractal **R**andom **R**ing (HGFRR). It constructs and maintains a recursive DHT-ring like structure according to geographical proximity of nodes. The broadcast operation on such a network which contains $N$ nodes achieves $O(N)$ in terms of message complexity, and $O(logN)$ in terms of time complexity. Security issues are also addressed to protect the anonymity of nodes. Evaluation shows that HGFRR outperforms typical P2P network in blockchain systems in both time complexity and messages complexity. Consequently, the throughput of the blockchain system can be further improved around 1.4X to 2.1X. Source code of the implementation of HGFRR (in C++) will be available on GitHub once appropriate.

## 1 Introduction

A blockchain is essentially a distributed ledger that permanently records the transactions among parties [18]. The transactions recorded are verifiable and resistant to modification without the need of a centralized third party. The decentralization nature and proved immutability have led to the emergence of cryptocurrencies which leverages the blockchain system as their cornerstone, the most salient example being Bitcoin [31]. Despite the popularity of cryptocurrencies, general and all-purpose blockchain systems that accommodate various applications, such as Ethereum [39] have been proposed. However, although Ethereum is a Turing-complete system [39], it is in essence designed for the cryptocurrency based on it. Hence the Proof-of-Work (PoW) consensus has been utilized used to support the valuation of the cryptocurrency in the socioeconomic sense, which results in poor efficiency. To facilitate general-purpose applications in a more efficient manner, other consensus protocols have been proposed to improve the performance of the blockchain systems in different scenarios such as Proof-of-Stake [22], Proof-of-Luck [28], and Proof-of-Membership [23]. Hyperledger Sawtooth [19] utilizes Intel Software Guard eXtentions (SGX, introduced below) to trust nodes and proposes Proof-of-Elapsed Time believed to be highly efficient. Additionally, EOS [12] based on DPoS, NEO [17] based on DBFT, and Hyperledger Fabric [5] are all examples of new blockchain systems which are claimed to achieve more than 10k transaction rate [3].

With tremendous works focused on increasing the blockchain system efficiency and transaction rate, broadcast frequency is growing. Unfortunately, experiments and studies [5, 40] already show that the network layer became the bottleneck of further growing of transaction rate. For example, EOS report [40] shows that EOS is sensitive to network latency. Higher network latency caused by other bandwidth-intensive applications in the same network or high client transaction input rate can lead to significant drop of throughput. Hyperledger Fabric [5] also shows that its throughput is capped by the low efficiency of the P2P network. However, current blockchain systems are based on Distributed Hash Table (DHT) structure and message broadcast in a Gossip manner [13]. It is efficient in terms of node discovery and data look up. However, broadcasting messages suffers from traffic congestion and inefficient convergence problems when transaction rate gets higher. Our key insight is that the Gossip algorithm used to broadcast a message does not fit in the demand for P2P networks from blockchain systems. Although many improvements has been made on Gossip algorithm such as adding unique message ID, using Time-to-Live field to control flooding, using pull-version sending mechanism to reduce repeated messages, our evaluations show that they are not efficient, and still suffer from traffic congestion problem.

Blockchain systems' requirements are different from other Peer-to-Peer applications: (i) on-demand streaming allows users to look up data in the P2P network and download stream data from the source, e.g. BitTorrent-based streaming systems like BASS [10], Peer-Assisted [6], LiveBT [26], and Give-To-Get [29]; (ii) audio/video conferencing applications deal with small scale point-to-point connected networks which requires low latency, e.g. Skype [4]; (iii) peer-to-peer file sharing makes efficient indexing and searching possible, e.g. Napster [36], Gnutella [33], and KaZaA [15]; (iv) video streaming applications enables single-source broadcasting efficient, e.g. SplitStream [7], Bullet [24], and ChainSaw [32]. (i) and (ii) are not relevant to the context of blockchain systems since nodes in a blockchain system network should be in a large scale and broadcasting a message is an active operation instead of searching and downloading data. (iii) and (iv) are more similar to blockchain systems' use case. However, P2P file sharing is not real-time and the broadcast model in a blockchain system is not indexing and searching. In video streaming, time is stringent and the network size can be large-scale. However, it is a data or bandwidth-intensive communication which means control messages in a broadcast operation are relatively small compared to the data to transmit.

Through our study, we summarized two main functions required for the underlying P2P network from blockchain systems: peer discovery and message dissemination. For peer discovery, DHT-based protocols such as Kademlia, Chord, Pastry, Tapestry, CAN are used to achieve efficiency. For message dissemination, Gossip algorithms are used due to its robustness and simplicity. Under 50% failure, Gossip can send twice amount of messages to cover the remaining nodes. However, the robustness of Gossip exceeds too much of the requirement from the consensus protocols in most blockchain systems. As a side effect, Gossip generate redundant messages in the network which lead to traffic congestion. To tackle the problem, our key idea is that broadcasting using the DHT-based structure in a hidden and secure way can improve the broadcast efficiency in terms of both time complexity and message complexity.

We implemented our idea in HGFRR, which is a **H**idden **G**eographical **F**ractal **R**andom **R**ing structured P2P network. HGFRR contains multi-level fractal random rings. Each ring at the lower level will have a couple of contact nodes which is randomly selected to represent the ring in the upper level ring. Unlike DHT-based protocols which indexes the whole network as a ring, HGFRR recursively constructs rings based on the proximity of peers and the number of peers in a ring. The broadcast on HGFRR is recursively performed on each ring. The in-ring broadcast uses the k-ary distributed spanning tree formed within the ring. Both the proof and evaluation show that the broadcast operation in HGFRR is more efficient than the P2P network in Ethereum, in terms of time complexity and message complexity. The message complexity of our network with $N$ nodes is $O(N)$ and time complexity of message broadcast is logarithm, which are both better than extant work.

The paper makes the following contributions. First, the paper identifies requirements for the P2P networks from blockchain systems and pointed out the over-robustness and thus the inefficiency of current message dissemination algorithm. Second, it presents and proves a new network protocol HGFRR that improves message dissemination efficiency and thus the overall throughput of the blockchain system. Third, HGFRR is the first P2P network in blockchain systems that addresses geographical locality and security problems. Fourth, HGFRR is implemented in C++ which is portable and runnable across-platform and intensively evaluated.

The remaining of this paper is structured as follows: the background, the design overview, proof of analysis, implementation, and evaluation, related works. [TODO]

## 2 Background and Motivation

### 2.1 Peer-to-Peer Overlay Networks

There have been tremendous efforts and many technical innovations in the Internet broadcasting in the past three decades [25]. Internet protocol (IP) multicast represented an earlier attempt to tackle this problem but failed largely due to concerns regarding scalability, deployment, and support for higher level functionality. In contrast, Peer-to-peer based broadcast has been shown to be cost effective and easy to deploy. This new paradigm brings a number of unique advantages such as scalability, resilience, and effectiveness in coping with dynamics and heterogeneity. With Network Function Virtualization (NFV), upper-layer applications are allowed to control the lower-layer functionalities of the network such as routing. There are tremendous P2P networks which we divide them into two groups: one group focusing on node discovery and the other group focusing on message dissemination.

#### 2.1.1 Node Discovery

**Distributed Hash Table (DHT)** Hashing can be used to uniquely identify a particular object from a group of similar objects by assigning each object a hash value. A DHT [14] is a class of decentralized distributed system that has $(key, value)$ pairs and any participating node can efficiently retrieve the value associated with a given key. This is similar to the working of a hash table which forms a data structure for storing (hash-value, object) pairs. Nodes in the system are responsible for managing the mapping from keys to values in a way that minimizing the disruption caused to the participants. This mechanism allows DHTs to scale to a large number of nodes, afford constant arrivals, and tolerate intermittent node failures.

A DHT normally have three main features: (i) Autonomy and decentralization: the nodes collectively form the system without any central coordination; (ii) Fault tolerance: the system should be reliable (in some sense) even with nodes continuously joining, leaving, and failing; (iii) Scalability: the system should function efficiently even with thousands or millions of nodes.

**Chord** [38] is an algorithm for P2P DHT. It indicates how keys are assigned to nodes, and how a node can look up the value for a given key by first locating the node responsible for that key. Chord queries a key from a client (usually a node) to find the successor($k$). If the key can not be found locally, the query is passed to a nodes successor, which leads to a $O(N)$ query time where $N$ is the number of nodes in the ring. The implementation of a faster search method which requires each node to keep a so called "finger table" can avoid the linear search above. The finger table contains up to $m$ entries, where $m$ is the number of entries in the hash key. With this kind of a finger table, the number of nodes that must be contacted in an N-node network to find a successor becomes $O(logN)$.

**Pastry** [34], is another overlay and routing network for the implementation of a DHT. Pastry uses consistent hashing as a hash algorithm. The key value obtained by hashing is one-dimensional (in fact, 128-bit integer space is used). Pastry does not specify which hash algorithm should be used. In the Pastry protocol, each node has a 128-bit identity (Node Id). In order to ensure the uniqueness of the Node ID, the network identifier (such as the IP address) of the node is generally obtained by hashing. Each node in Pastry has a routing table, a neighboring node set and a leaf node set, which together form the node's state table. Each routing step is closer to the target node than the previous step, so the process is convergent. If the routing table is not empty, at least one prefix matching digit can be added to each route. Therefore, when the routing table is always valid, the number of routing steps is at most $O(logBN)$ where $B$ is the system parameter.

Pastrys hash table has a circular key-space, just like Chords hash table. Node IDs are used to represent position in the circular key-space. These are chosen randomly so as to ensure that adjacent node IDs represent geographically diverse peers. Pastry's routing takes advantage of the proven maximum mask matching algorithm, so it can be implemented with many off-the-shelf software algorithms and hardware frameworks for good efficiency. Compared to Chord, Pastry introduces the concept of a set of leaf nodes and neighbor nodes. When the application layer can obtain the node information of the two sets in time and accurately, the speed of the route search can be greatly accelerated, and the network transmission overhead caused by the route can be reduced; but how to do this ideally in the dynamically changing P2P network does have some difficulty.

### 2.1.2 Message Dissemination

Gossip based protocols are developed for providing high reliability and scalability of message delivery [20]. Gossip protocols are highly used for reducing control message overhead [16]. Gossip protocols are scalable because they do not require as much synchronization as traditional reliable multicast protocols. In gossip-based protocols, each node contacts one or a few nodes in each round usually chosen at random, and exchanges information with these nodes. The dynamics of information spread algorithm behavior stems from the work in epidemiology, and leads to high fault tolerance. Gossip-based protocols usually do not require error recovery mechanisms, and thus enjoy a large advantage in simplicity, while often incurring only moderate overhead compared to optimal deterministic protocols.

Unfortunately, gossip algorithms suffer from repeated messages which may lead to traffic congestion when broadcast frequency grows. There are several improvements made on gossip algorithm. Directional gossip uses a gossip server to construct spanning tree but it is not scalable. Intelligent select node selects directional children to build a tree. Some other improvements add TTL, use UID to reduce redundancy but still has overhead.

Tree-based: multicast/multi-tree [not every node can be the source]

## 2.2 P2P Networks in Blockchain Systems

Ethereum is implemented based on Kademlia [27], which is also a distributed hash table for decentralized P2P networks. Kademlia uses UDP for communication among peers and specifies the structure of the network and the exchange of information through node lookups. Similar to Pastry, each node is identified by a Node ID. Kademlia has many ideal features that previous DTHs could not provide at the same time. By incorporating broadcast configuration information into the loop-up messages, it minimizes the configuration messages that nodes must send in order to understand each other. Nodes have enough knowledge and flexibility to route queries through low latency paths. Kademlia uses concurrent asynchronous queries to avoid timeouts caused by node failures. Nodes record each other's existence against certain basic denial of service attacks.

While searching for $n$ nodes in a system, Kademlia only contacts $O(log(n))$ nodes, which is very efficient. Unlike first or second generation P2P file sharing networks such as Napster[37] or the Gnutella[33], Kademlia uses DHTs to look up files in the network. A DHT, as discussed above, stores resource locations throughout the network, and a major criterion for these protocols is to locate the desired nodes quickly. Many of the advantages stem from the use of novel XOR metrics to define the distance between two points in the primary key space. XOR is symmetric, which allows Kademlia participants to receive query requests from the exact same node distribution contained in their routing tables. Without this feature, systems like Chord cannot learn useful routing information from the queries they receive. Worse, asymmetry can make routing tables less flexible. For example, in Chord, each finger table must store the exact nodes before an interval. In fact, any node within the interval and those nodes before the same interval may be physically far apart. In contrast, Kademlia can send queries to any node within an interval, which allows it to select the optimal route based on the delay, and even asynchronously query several equally suitable nodes in parallel.

### 2.3 Trusted Execution Environment (TEE)

Trusted computing has been defined to help systems to achieve secure computation, privacy and data protection. Originally, the Trusted Platform Module (TMP) allows a system to provide evidence of its integrity in a separate hardware module. In recent years, a new approach to address trusted computing has emerged, which allow the execution of arbitrary code within a confined environment that provides tamper-resistant execution to its applications - trusted execution environment (TEE) [35]. TEE is a secure, integrity-protected processing environment, consisting memory and storage capabilities [2].

Intel SGX is one popular instance of TEE which 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 potentialy malicious [9]. Intel SGX provides two kinds of attestations (local and remote) to prove that particular piece of code is running in a genuine SGX-enabled CPU [8] and also provides a trustworthy source of random number [1]. Currently there is one related work which uses Intel SGX to provide reliable broadcast for P2P network [21]. However, there is no related work on using Intel SGX to improve asynchronous P2P network performance, which is the main focus of this project.

### 2.4 Design Motivation

Talk about the design motivation, kind of combining the two.

## 3 HGFRR Design

In this section, we first introduce the design principles of HGFRR (Section §3.1). Then we present the topology of the network (Section §3.2), how the structure is formed (Section §3.2.1) and maintained (Section §3.2.2), and the broadcast algorithm (Section §3.3). Security issues are also addressed in the design of HGFRR (Section §3.4).

### 3.1 Design Principles

The design of HGFRR as a Peer-to-Peer network layer under a blockchain system follows four principles:

- Fair: An unfair P2P network may ascend free-riders, frustrate majority of users and consequently lead to instability of the network [30].
- Self-organizing: No central server should be responsible for organizing the structure of the network. In other words, the decentralization nature of the P2P network should not be affected.
- Anonymous: Each node in the network and the network topology should be hidden from the outsider so that target attack can be avoided to some extent.
- Efficient on broadcast: The converge time of each message to broadcast should be as small as possible. Therefore, the number of concurrent messages on flight will be reduced.
- Robust in the dynamic environment: The network is unstable due to the frequent disconnection or node-join. The structure of the network should be easy to maintain in a distributed manner.
- Scalable to large number of nodes: The P2P network should be able to scale up to tremendous number of nodes as a public block chain may grow up to millions of nodes.

### 3.2 Topology

Before presenting the protocol, several key concepts should be defined clearly:

- Node: One instance of a server/virtual machine in the network;
- Ring: A group of nodes connected in a ring-like structure.
- Contact Node: the node on the ring who is in charge of adding new nodes, contacting the nodes in the upper level of the network, and broadcasting the message.

[TODO: add figure - network topology]

The network topology of HGFRR is basically a fractal-ring structure, where lower level rings reside on higher level rings (See Figure [TODO]) in a recursive way. At the top level resides the largest ring where several sub-ring resides on while at the bottom level resides the smallest rings. The figure shows a network of 3 levels. Level 2 contains the largest ring. On the largest ring, there are three sub-rings of 2 levels. Level 1 contains the second largest rings and level 0 contains the smallest rings. The nodes in red are contact nodes of each ring. They are elected to be normal nodes of the upper level ring.

### 3.2.1 Structure Formation

When a new node wants to join the network, it will first send ping-messages to each of the contact nodes of the largest ring (at the top level). The new node will pick the contact node with the shorted response time to send a join-message. The contact node will then decide which sub-ring it should add this new node to, by sending the node information of contact 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 contact node of the sub-ring will then introduce the new node to the sub-sub-ring. In the end, the contact node of a ring at the bottom level will then add the new node to the ring. If the number of node on the ring that the new node join to exceeds the threshold, then this ring will transform to a two-level ring (See Figure TODO), i.e. several groups of nodes on the large ring will form several rings. The transformation will be further elaborated in Section §3.2.2. After a new node joins the network, the contact node of this ring will broadcast within ring this node's information. The member nodes of this ring will then tell their information by sending welcome-messages to the new node.

The upper level rings are formed by contact node election. When a ring is first formed, the first member node of the ring will be the contact node of this ring. Each generation of contact nodes have their term of service. At the end of the term, one of the contact nodes will generate random IDs from all member node IDs. This election result will be dispersed within ring, and multi-casted to the contact nodes of the upper level ring and the lower level rings.

### 3.2.2 Structure Maintenance

Each node on the bottom ring will send heart-beat messages to its successor and predecessor to check the aliveness of them. Once a node are not responding to the heart-beat message after the timeout value, the node will double check the liveness of this node with its, e.g. if A's predecessor B does not respond, A will check with the predecessor of B. If they agree that this node left the network (intentionally or accidentally), the information will be disseminated to the ring and this node will be officially removed from the network. If the missing node is the contact node, then the next generation of contact nodes will be elected. If the number of nodes on the ring is smaller than the lower limit, a transformation from right to left in Figure [TODO] will be performed.

**Algorithm 1** Bootstrap

```
 1: procedure NODE JOIN
 2:     response ← queryDNSSeeds()
 3:     contactNodeList ← response.nodes
 4:     level ← response.topLevel
 5: loop:
 6:     if level = 0 then
 7:         break
 8:     else
 9:         optimalMetric ← 0
10:         for contactNode : contactNodeList do
11:             metric ← testProximity(contactNode)
12:             if metric > optimalMetric then
13:                 optimalMetric ← metric
14:                 targetNode ← contactNode
15:         response ← queryContactNode(contactNode, level)
16:         contactNodeList ← response.nodes
17:         level ← level-1
18:         goto loop.
19:     /∗contact node of level 0 ring broadcast node-join
20:      * message withing the ring */
21:     this.nodeTable.insert(recvMsg.node)
22:     if recvMsg.node.contactNode = true then
23:         this.contactNodeList.insert(recvMsg.node)
```

## 3.3 Broadcast

**Broadcast** is the process of disseminating a message from any node to the whole network. When a node wants to send a message to the whole network, it will first send broadcast-message to one of the contact nodes of the ring it resides on. Then the message will be routed to two directions: one direction is downwards, i.e. the contact node will broadcast the message in the ring and recursively in the sub-rings; the other direction is upwards, i.e. the contact node will send broadcast-message to one of the contact nodes of the upper level ring. Recursively, the broadcast-message will be received by one of the contact nodes of the largest ring. Then the contact node will broadcast recursively in the sub-rings. Till the bottom level, each node in the fractal ring will receive this message.

The k-ary distributed spanning tree method [11] is used to broadcast message in a ring. Details will be presented in the subsection. Based on this method, the time complexity of a broadcast operation will be $O(logN)$ and message complexity will be $(O(N))$, which are currently the best among related works.

**Algorithm 2** Maintenance

```
// This function will be called every TIME_INTERVAL
function DETECT_NODE_LEFT
    if liveness_check_predecessor() = false then
        msg ← constructNodeLeaveMSG()
        this.inRingBroadcast(msg)
    if liveness_check_successor() = false then
        msg ← constructNodeLeaveMSG()
        this.inRingBroadcast(msg)

function LIVENESS_CHECK_PREDECESSOR
    ret = sendHeartBeatMSG(this.predecessor)
    if ret.type = TIMEOUT then
        status ← confirmWithPrePredecessor()
        return status

function LIVENESS_CHECK_SUCCESSOR
    ret = sendHeartBeatMSG(this.successor)
    if ret.type = TIMEOUT then
        status ← confirmWithSucSuccessor()
        return status
```

### 3.3.1 Broadcast Within-Ring Mechanism

In-ring broadcast is based on the k-ary distributed spanning tree method. The basic idea is that the broadcast starter will first generate a random number $k$, and then a k-ary spanning tree can be formed in a distributed manner. Broadcast will then be triggered from the root to every node in the tree. 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 tree are formed by using the broadcaster (which is numbered 0) as the root. 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.

[TODO] For example, Figure 4.3(a) shows a ring of 10 nodes numbered from 0 to 9. As the k-ary spanning tree algorithm states, when $k = 2$: node 0 will connect to node 1, 2, 4, 8; node 1 will connect to node 3, 5, 9; node 2 will connect to node 6; node 3 will connect to node 7. After a spanning tree is formed (see Figure 4.3(b)), the message will be disseminated from node 0 down to every node in the spanning tree.

## 3.4 Security Consideration

Talk about the usage of Intel SGX, fake messages, contact node election.

**Algorithm 3** Broadcast

```
function BROADCAST(data)
    level ← 0
    contactNode ← selectFromContactNodes(level)
    message ← wrapMessage(data)
    sendTo(contactNode, message)

function BROADCAST_UP(currentLevel, data)
    level ← currentLevel+1
    contactNode ← selectFromContactNodes(level)
    message ← wrapMessage(data)
    sendTo(contactNode, message)

function BROADCAST_WITHIN_RING(currentLevel, message, sentIDs, k)
    endID ← this.getNodeTableSize(currentLevel)
    i ← 0
    nodeOrder ← message.nodeOrder
    while nodeOrder + pow(k, i) < endID do
        currentID = nodeOrder + pow(k, i)
        if pow(k, i) <= nodeOrder then
            i++
            continue
        else
            targetNodeID = NodeID + pow(k, i)
            if targetNodeID > endID then
                targetNodeID -= endID + 1
            args=(targetNodeID, currentLevel)
            receiver ← (this.getPeer(args))
            i++
            sendTo(receiver, message)
```

## 4 Proof and Analysis

This is the proof and analysis.

### 4.1 Proof of Broadcast Performance

give a proof of time complexity and message comlexity of broadcast, node join and contact node election. Compare to current works.

### 4.2 Security and Robustness Analysis

analyze of fault tolerance, anonymity.

## 5 Implementation Details

This is the implementation details.

How we implement the system. and the architecture of the codebase.

## 6 Evaluation

This is the evaluation.

how we evaluated hgfrr.

### 6.1 Evaluation Setup

talk about how we set up the evaluation

### 6.2 Ease of Use

use kad+gossip to substitute eth

### 6.3 Performance Improvements

talk about the performance improvement in terms of convergence time, message complexity.

#### 6.3.1 Broadcast

broadcast performance

#### 6.3.2 Robustness and Scalability

fault-tolerance and scalability of broadcast

#### 6.3.3 Contact Node Election

performance of contact node election

#### 6.3.4 Node Join and Leave

performance of node join and leave, which may lead to structure change

### 6.4 Security Evaluation

wireshark analysis to show the anonymity of contact nodes

## 7 Related Work

This is the related work.

**P2P Network in Ethereum** Talk about kademlia+gossip in ethereum.
**Broadcast in DHT** Talk about a work working on broadcast in dht.
more to be added and compared.

## 8 Conclusion

This is the conclusion.

Give a conclusion on background, motivation, overview of hgfrr, feature and analysis result. future work.

## Acknowledgments

A polite author always includes acknowledgments. Thank everyone, especially those who funded the work.

## ATC Template For Reference

More fascinating text. Features[1] galore, plethora of promises.

## References

[1] Software Guard Extensions Programming Reference. http://kib.kiev.ua/x86docs/SDMs/329298-001.pdf.

[2] Asokan, N., Ekberg, J.-E., Kostiainen, K., Rajan, A., Rozas, C., Sadeghi, A.-R., Schulz, S., and Wachsmann, C. Mobile trusted computing. *Proceedings of the IEEE 102*, 8 (2014), 1189–1206.

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

[4] BASET, S. A., AND SCHULZRINNE, H. An analysis of the skype peer-to-peer internet telephony protocol. *arXiv preprint cs/0412017* (2004).

[5] CACHIN, C. Architecture of the hyperledger blockchain fabric. In *Workshop on Distributed Cryptocurrencies and Consensus Ledgers* (2016), vol. 310.

[6] CARLSSON, N., AND EAGER, D. L. Peer-assisted on-demand streaming of stored media using bittorrent-like protocols. In *International Conference on Research in Networking* (2007), Springer, pp. 570–581.

[7] CASTRO, M., DRUSCHEL, P., KERMARREC, A.-M., NANDI, A., ROWSTRON, A., AND SINGH, A. Splitstream: high-bandwidth multicast in cooperative environments. In *ACM SIGOPS Operating Systems Review* (2003), vol. 37, ACM, pp. 298–313.

[8] CHEN, X., ZHAO, S., WANG, C., ZHANG, S., AND CUI, H. GEEC: Scalable, Efficient, and Consistent Consensus for Blockchains. *ArXiv e-prints* (Aug. 2018).

[9] COSTAN, V., AND DEVADAS, S. Intel sgx explained. *IACR Cryptology ePrint Archive 2016*, 086 (2016), 1–118.

[10] DANA, C., LI, D., HARRISON, D., AND CHUAH, C.-N. Bass: Bittorrent assisted streaming system for video-on-demand. In *MMSP* (2005), vol. 5, pp. 1–4.

[11] EL-ANSARY, S., ALIMA, L. O., BRAND, P., AND HARIDI, S. Efficient broadcast in structured p2p networks. In *International workshop on Peer-to-Peer systems* (2003), Springer, pp. 304–314.

[12] EOSIO. Eos.io technical white paper v2. https://github.com/EOSIO/Documentation/blob/master /TechnicalWhitePaper.md.

[13] EUGSTER, P. T., GUERRAOUI, R., KERMARREC, A.-M., AND MASSOULIÉ, L. Epidemic information dissemination in distributed systems. *Computer 37*, 5 (2004), 60–67.

[14] GALUBA, W., AND GIRDZIJAUSKAS, S. Distributed hash table. In *Encyclopedia of Database Systems*. Springer, 2009, pp. 903–904.

[15] GOOD, N. S., AND KREKELBERG, A. Usability and privacy: a study of kazaa p2p file-sharing. In *Proceedings of the SIGCHI conference on Human factors in computing systems* (2003), ACM, pp. 137–144.

[16] GUPTA, I., BIRMAN, K. P., AND VAN RENESSE, R. Fighting fire with fire: using randomized gossip to combat stochastic scalability limits. *Quality and Reliability Engineering International 18*, 3 (2002), 165–184.

[17] HOXHA, L. Hashgraph the future of decentralized technology and the end of blockchain. *European Journal of Formal Sciences and Engineering 1*, 2 (2018), 29–32.

[18] IANSITI, M., AND LAKHANI, K. R. The truth about blockchain.

[19] INTEL, C. Introduction to sawtooth, v1.1.2. https://sawtooth.hyperledger.org/docs/core/releases /latest/introduction.html.

[20] ISLAM, M. H., WAHEED, S., AND ZUBAIR, I. An efficient gossip based overlay network for peer-to-peer networks. In *Ubiquitous and Future Networks, 2009. ICUFN 2009. First International Conference on* (2009), IEEE, pp. 62–67.

[21] JIA, Y., TOPLE, S., MOATAZ, T., GONG, D., SAXENA, P., AND LIANG, Z. Robust synchronous p2p primitives using sgx enclaves. *IACR Cryptology ePrint Archive 2017* (2017), 180.

[22] KIAYIAS, A., RUSSELL, A., DAVID, B., AND OLIYNYKOV, R. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Annual International Cryptology Conference* (2017), Springer, pp. 357–388.

[23] KOGIAS, E. K., JOVANOVIC, P., GAILLY, N., KHOFFI, I., GASSER, L., AND FORD, B. Enhancing bitcoin security and performance with strong consistency via collective signing. In *25th USENIX Security Symposium (USENIX Security 16)* (2016), pp. 279–296.

[24] KOSTIĆ, D., RODRIGUEZ, A., ALBRECHT, J., AND VAHDAT, A. Bullet: High bandwidth data dissemination using an overlay mesh. In *ACM SIGOPS Operating Systems Review* (2003), vol. 37, ACM, pp. 282–297.

[25] LIU, J., RAO, S. G., LI, B., AND ZHANG, H. Opportunities and challenges of peer-to-peer internet video broadcast. *Proceedings of the IEEE 96*, 1 (2008), 11–24.

[26] LV, J., CHENG, X., JIANG, Q., YE, J., ZHANG, T., LIN, S., AND WANG, L. 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* (2007), IEEE, pp. 501–508.

[27] MAYMOUNKOV, P., AND MAZIERES, D. Kademlia: A peer-to-peer information system based on the xor metric. In *International Workshop on Peer-to-Peer Systems* (2002), Springer, pp. 53–65.

[28] MILUTINOVIC, M., HE, W., WU, H., AND KANWAL, M. Proof of luck: An efficient blockchain consensus protocol. In *Proceedings of the 1st Workshop on System Software for Trusted Execution* (2016), ACM, p. 2.

[29] MOL, J. J.-D., POUWELSE, J. A., MEULPOLDER, M., EPEMA, D. H., AND SIPS, H. J. Give-to-get: free-riding resilient video-on-demand in p2p systems. In *Multimedia Computing and Networking 2008* (2008), vol. 6818, International Society for Optics and Photonics, p. 681804.

[30] NAGHIZADEH, A. Improving fairness in peer-to-peer networks by separating the role of seeders in network infrastructures. *Turkish Journal of Electrical Engineering & Computer Sciences 24*, 4 (2016), 2255–2266.

[31] NAKAMOTO, S. Bitcoin: A peer-to-peer electronic cash system.

[32] PAI, V., KUMAR, K., TAMILMANI, K., SAMBAMURTHY, V., AND MOHR, A. E. Chainsaw: Eliminating trees from overlay multicast. In *International Workshop on Peer-to-Peer Systems* (2005), Springer, pp. 127–140.

[33] RIPEANU, M. Peer-to-peer architecture case study: Gnutella network. In *Peer-to-Peer Computing, 2001. Proceedings. First International Conference on* (2001), IEEE, pp. 99–100.

[34] ROWSTRON, A., AND DRUSCHEL, P. 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* (2001), Springer, pp. 329–350.

[35] SABT, M., ACHEMLAL, M., AND BOUABDALLAH, A. Trusted execution environment: what it is, and what it is not. In *14th IEEE International Conference on Trust, Security and Privacy in Computing and Communications* (2015).

[36] SAROIU, S., GUMMADI, K. P., AND GRIBBLE, S. D. Measuring and analyzing the characteristics of napster and gnutella hosts. *Multimedia systems 9*, 2 (2003), 170–184.

[37] SAROIU, S., GUMMADI, P. K., AND GRIBBLE, S. D. Measurement study of peer-to-peer file sharing systems. In *Multimedia Computing and Networking 2002* (2001), vol. 4673, International Society for Optics and Photonics, pp. 156–171.

[38] STOICA, I., MORRIS, R., KARGER, D., KAASHOEK, M. F., AND BALAKRISHNAN, H. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review 31*, 4 (2001), 149–160.

[39] WOOD, G. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper 151* (2014), 1–32.

[40] XU, B., LUTHRA, D., COLE, Z., AND BLAKELY, N. Eos: An architectural, performance, and economic analysis.

## Notes

[1]Remember to use endnotes, not footnotes!