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 [21]. 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 [39]. Blockchain systems are based on consensus protocols to achieve chain consistency. With tremendous works focused on consensus layer to increase the blockchain system efficiency and transaction rate, broadcast frequency is growing.

Unfortunately, experiments and studies [6, 52, 32] already show that the network layer became the bottleneck of further growing of transaction rate. For example, EOS report [52] 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 [6] 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 [15]. 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 require two main functions from the underlying P2P network from : peer discovery and message dissemination. For peer discovery, DHT-based protocols such as Chord [49], Pastry [45], Tapestry [54], and Kademlia [35] 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 sys-

tems. As a side effect, Gossip generates 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 implement 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 organized as follows. Section §2 introduces the background of this problem and related works. Section §3 presents the design overview of HGFRR including topology and protocols. Section §4 gives a proof for time and message complexity and analyzes the robustness of HGFRR on node failures. Section §5 describes the implementation details. Section §6 presents and discusses the evaluation results. Section §7 concludes the paper.

2 Background and Related Works

2.1 Consensus Protocols in Blockchains

Despite the popularity of cryptocurrencies, general and all-purpose blockchain systems that accommodate various applications, such as Ethereum [51] have been proposed. However, although Ethereum is a Turing-complete system [51], 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 [28], Proof-of-Luck [36], and Proof-of-Membership [29]. Hyperledger Sawtooth [22] utilizes Intel Software Guard eXtentions (SGX, introduced below) to trust nodes and proposes Proof-of-Elapsed Time believed to be highly efficient. Additionally, EOS [14] based on DPoS, NEO [20] based on DBFT, Conflux [32], Omniledger [30], and Hyperledger Fabric [6] are all examples of new blockchain systems which are claimed to achieve more than 10k transaction rate [4].

Consensus	Tolerated Percentage	Examples
PoW	$< 25\%$	Bitcoin
PoS	$< 51\%$	PeerCoin
PBFT	$< 33.3\%$	HyperLedger Fabric
DPOS	$< 51\%$	Bitshares, EOS
Ripple	$< 20\%$	Ripple
Tendermint	$< 33.3\%$	Tendermint

Table 1: Consensus Algorithms and Their Tolerated Percentages

2.2 Peer-to-Peer Overlay Networks

There have been tremendous efforts and many technical innovations in the Internet broadcasting in the past three decades [33]. 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.

The most important feature of the p2p architecture is that individuals in their network are equal in role and function. Although each individual may handle different requests, the actual resources provided may differ after specific quantification, but they can all simultaneously provide and consume resources. If the resources in the entire network, including but not limited to computing power, storage space, network bandwidth, etc., are regarded as a total amount, the resource distribution in the p2p network is dispersed among individuals (maybe not necessarily evenly distributed). Therefore, the p2p network architecture is naturally decentralized and distributed.

2.2.1 Data Look-up

Not every individual communicates with other peers in the network. This is actually a very important feature of the p2p network: an individual only needs to connect with a small part of nodes in the network. How many neighbors and how to connect vary from one to another. Basically, P2P networks are divided into unstructured and structured networks [42]. Unstructured P2P networks are simple and easy to deploy, and the individuals in the local area of the network can be arbitrarily distributed. When dealing with a large number of new individuals joining the network and the old individuals leaving the network (churn), unstructured P2P networks are very stable [50]. The disadvantage is that the efficiency of finding data in the network is too low. Because there is no foreseeable information, it is often necessary to send query requests throughout the network (at least most individuals) or use flooding, which will occupy a large part of the network resources and greatly slow down other businesses in the same network.

The individual distribution of structured P2P networks has been carefully designed, and the main purpose is to improve the efficiency of querying data and reduce the resource consumption caused by query data. The basic means to improve query efficiency is to index data [44]. The most common implementation of structured p2p networks is distributed hash table (DHT) [16], which assigns a key to each data (value) to form $(key, value)$ pairs. Hashing can be used to uniquely identify a particular object from a group of similar objects by assigning each object a hash value. 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. In this way, when looking for a certain item of data, the search area can be continuously reduced according to the key, thereby greatly reducing resource consumption. Although structured P2P network is efficient in data look-up, the robustness is a concern. Since each individual needs to maintain a large number of neighboring individuals, when the churn events in which a large number of new and old individuals frequently join and leave occur in the network, the performance of the entire network will be greatly deteriorated. Part of the resources are consumed when updating the neighbor list (including the update of the neighbor list, and the stored list is updated between each other), and the keys of many peers also need to be redefined. Most used DHTs are Chord [49], Pastry [45], and Tapestry [54].

2.2.2 Message Dissemination

Gossip based protocols are developed for providing high reliability and scalability of message delivery [23]. Gossip protocols are highly used for reducing control message overhead [19]. 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.

In addition, tree-based and data-driven broadcast/multicast algorithms in video streaming or file sharing does not fit in blockchain system context [33]. Tree-based approaches like SplitStream [8] and CoopNet [40] is efficient but need to be maintained. It works poorly when dealing with node failures. For DHT, the cost of correcting the routing table of each node is also high. In addition to Gossip, other data-driven approaches like ChainSaw [41], Bullet [31], and CoolStream [53] have reduced the redundancy of Gossip a lot. However, pull-based broadcast and single-source broadcast/multicast do not fit into the context of blockchain systems.

2.3 P2P Networks in Blockchain Systems

2.3.1 Requirements from Blockchain Systems

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 [12], Peer-Assisted [7], LiveBT [34], and Give-To-Get [37]; (ii) audio/video conferencing applications deal with small scale point-to-point connected networks which requires low latency, e.g. Skype [5]; (iii) peer-to-peer file sharing makes efficient indexing and searching possible, e.g. Napster [47], Gnutella [43], and KaZaA [18]; (iv) video streaming applications enables single-source broadcasting efficient, e.g. SplitStream [8], Bullet [31], and ChainSaw [41]. (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.

2.3.2 State-of-Art P2P Networks in Blockchain Systems

Blockchain systems are either based random unstructured network or DHT-based structured network. Ethereum is implemented based on Kademlia [35], 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[48] or the Gnutella[43], Kademlia uses DHTs to look up files in the network. 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.

Most existing blockchain systems use some form of gossiping to disseminate transactions, blocks, and membership information. By utilizing gossip, participants will eventually receive all transactions and blocks with high probability. For example, participants in Bitcoin [39] gossip with neighboring peers about recent transactions, blocks, and advertise membership of other participants. Hyperledger Fabric [2] is a platform for deploying and operating permissioned blockchains. In Fabric, participants gossip about blocks, transactions, and membership information. Fabric divides gossiping into two modes: pull and push, where participants request state from other peers during pulling, and sends their state while pushing. Algorand [17] uses a similar gossip approach as Bitcoin [39], where participants select a small subset of peers to gossip with.

2.4 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) [46]. TEE is a secure, integrity-protected processing environment, consisting memory and storage capabilities [3].

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 [11]. 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 [10] 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 [25]. However, there is no related work on using Intel SGX to improve asynchronous P2P network performance, which is the main focus of this project.

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 [38].
- 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.

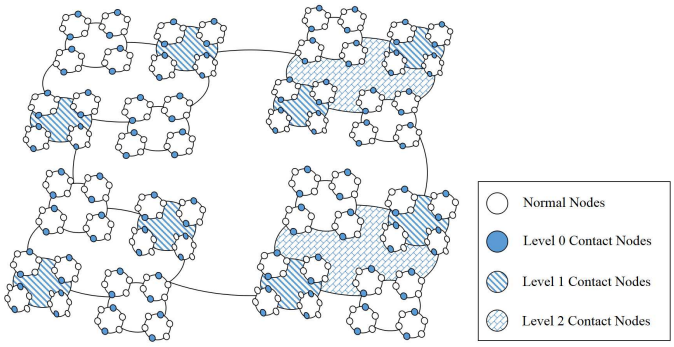


Figure 1: Illustration of a 3-level HGFRR Network

- 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.
- 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.

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.

The network topology of HGFRR is basically a fractal-ring structure, where lower level rings reside on higher level rings (See Figure 1) 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.

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 ← queryNode(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:         contactNode ← recvMsg.node
24:         contactNodeList.insert(contactNode)
```

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 2 Maintenance

```
1:  // This function will be called every TIME_INTERVAL
2:  function DETECT_NODE_LEFT
3:      if liveness_check_predecessor() = false then
4:          msg ← constructNodeLeaveMSG()
5:          this.inRingBroadcast(msg)
6:      if liveness_check_successor() = false then
7:          msg ← constructNodeLeaveMSG()
8:          this.inRingBroadcast(msg)
9:
10: function LIVENESS_CHECK_PREDECESSOR
11:     ret = sendHeartBeatMSG(this.predecessor)
12:     if ret.type = TIMEOUT then
13:         status ← confirmWithPrePredecessor()
14:         return status
15:
16: function LIVENESS_CHECK_SUCCESSOR
17:     ret = sendHeartBeatMSG(this.successor)
18:     if ret.type = TIMEOUT then
19:         status ← confirmWithSucSuccessor()
20:         return status
```

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 [13] 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.

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.

3.4 Security Consideration

Talk about the usage of Intel SGX[TODO].

To hide the existence of HGFRR contact nodes from outsiders, there are two mechanisms. First, fake messages are used to make the behavior of a contact node the same with that of a normal node. For a contact node, it will send messages of the same size to both one of the contact nodes and some of the past contact nodes in the upper level ring. For a normal node, it will send messages of the same size to both one of the contact nodes and some of its peers in the same level ring. When broadcasting messages withing the ring, the random parameter k will hide the relative orders of each node. Apart from broadcasting to its children in the constructed k-ary distributed spanning tree, each node will randomly choose a node to send fake message. All fake messages are of the same size of the real messages. Hackers may watch the packets sending out from a node and sending to the node for a long time. However, during one contact node service term, no difference can be discovered from the data collected. Second, contact nodes of a ring will be elected for every service term. One contact node of the last service term will generate random IDs by using Intel SGX's `sgx_read_rand ()` function [11], and then broadcast the nomination result withing the ring. Due to both limited contact node service term period and the same behavior during

Algorithm 3 Broadcast

```
1: function BROADCAST(data)
2:     level ← 0
3:     contactNode ← selectFromContactNodes(level)
4:     message ← wrapMessage(data)
5:     sendTo(contactNode, message)
6:
7: function BROADCAST_UP(currentLevel, data)
8:     level ← currentLevel+1
9:     contactNode ← selectFromContactNodes(level)
10:    message ← wrapMessage(data)
11:    sendTo(contactNode, message)
12:
13: function BROADCAST_WITHIN_RING(currentLevel,
    message, sentIDs, k)
14:    endID ← this.getNodeTableSize(currentLevel)
15:    i ← 0
16:    nodeOrder ← message.nodeOrder
17:    while nodeOrder + pow(k, i) < endID do
18:        currentID = nodeOrder + pow(k, i)
19:        if pow(k, i) <= nodeOrder then
20:            i++
21:            continue
22:        else
23:            targetNodeID = NodeID + pow(k, i)
24:            if targetNodeID > endID then
25:                targetNodeID -= endID + 1
26:            args=(targetNodeID, currentLevel)
27:            receiver ← (this.getPeer(args))
28:            i++
29:            sendTo(receiver, message)
```

one service term, outsider cannot differentiate contact nodes from normal nodes. HGFRR '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 [TODO].

4 Proof and Analysis

4.1 Broadcast Message Complexity

Consider a network of size N. All nodes in the network are geographically distributed evenly. In HGFRR , every r nodes that are geographically near to each other will be gathered into a ring at the bottom level. If there are c contact 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:

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

Then according to the protocol of HGFRR , the number of nodes elected to be the normal nodes at the next level will be the number of rings at the second level multiply with parameter c, which is:

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

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

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

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

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

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 contact nodes at the top level is:

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

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

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

$$= \frac{CN - T}{C(1 - C)}$$

Hence the message complexity of a broadcast operation is $O(N)$, which is better than current message complexity of both push Gossip ($O(NlogN)$) and push-pull Gossip ($O(NloglogN)$) [24].

4.2 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 transmit 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:

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

After the message reaches any of the contact 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:

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

$$= \sum_{i=0}^{l} C(i) \log_k C^i N$$

Hence the time complexity of a broadcast operation is $O(logN)$, which is also the complexity of the number of rounds in broadcast. Although the time complexity of Gossip algorithm is also $O(logN)$, in this case, if each node only connect to those nodes who have the smallest proximity in geographical locality, the total time complexity will be $O(N)$ [26, 27].

4.3 Robustness

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 system and link failure cause problems to the message dissemination. Since the dissemination of membership information and transactions require to reach all nodes, even the consensus protocol requires the message to reach at least a half (PoW, PoS, DPoS, Ripple) or two thirds (BFT, PBFT, Tendermint, Algorand BA*) [55], the P2P network under a blockchain system should try the 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 HGFRR , 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 cannot be reached is:

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

Thus the reliability of HGFRR will be:

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

In the context of blockchain systems, a 7-level HGFRR 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%. In the evaluation, we set the fault rate of all nodes to be from 10% to 70%, the reliability of HGFRR can reach the same level with Gossip. 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 need responses from a half or two thirds of all nodes to be honest, we could trade fault-tolerance off to gain efficiency.

5 Implementation Details

We have implemented HGFRR in C++. An HGFRR application running on a server mainly includes three components, a `NodeTable`, a `PeerManager`, and a `Discoverer`. The `Discoverer` is responsible for bootstrapping node. The `NodeTable` stores peer information at each level and maintains the network structure. The `PeerManager` deal with incoming messages and is responsible for broadcasting messages.

The source code of HGFRR will be available at `github.com/hku-systems/hgfrr`.

6 Evaluation

This is the evaluation.

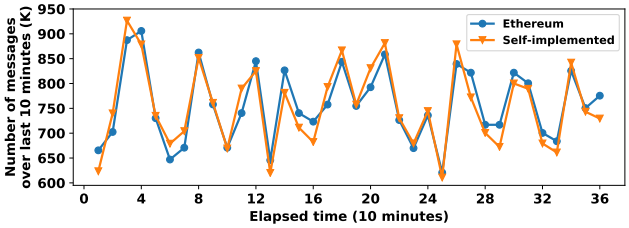
how we evaluated HGFRR .

6.1 Evaluation Setup

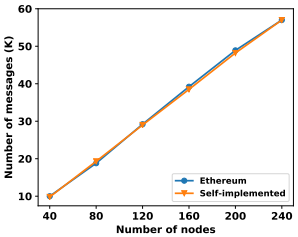
talk about how we set up the evaluation

6.2 Ease of Use

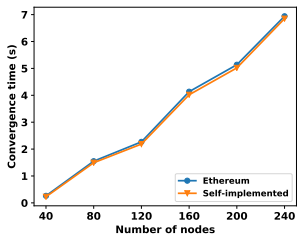
use kad+gossip to substitute eth



(a) Broadcast behavior.



(b) Message complexity.

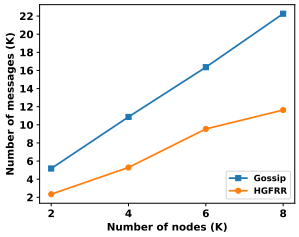


(c) Time complexity.

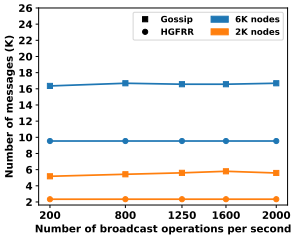
Figure 2: The comparison between Ethereum and self-implemented Kademlia-based Gossip.

6.3 Performance Improvements

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



(a) Message complexity with respect to number of nodes.



(b) Message complexity with respect to rate of broadcast operations.

Figure 3: Comparison of message complexity between Kademlia-based Gossip and HGFRR .

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

Table 2: Send-Packet Analysis of Node in HGFRR

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

Table 3: Receive-Packet Analysis of Node in HGFRR

6.3.1 Broadcast

6.4 Fault tolerance

6.5 End-to-end comparison

6.6 Security

We uses WireShark [9] to watch the packet sending out from a node and heading to the node. We counts three types of packets: around 17 kb, around 200 bytes, and less than 150 bytes, since they represents 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 contact node, contact nodes of a ring cannot be differentiated from the normal nodes. By watching and recording for a long time, we found that the overall packet statistics show that all nodes have same percentages of sent and received three types of messages (See Table 2 and Table 3). Therefore, due to the short service term of contact nodes and the similar percentages of all types of messages, it is hard for an outsider of the system to differentiate contact nodes from normal nodes.

7 Conclusion

This paper identifies the problem of using Gossip in the P2P network layer of a blockchain system, which to some extent already became the bottleneck of further improving the transaction rate. Though tremendous work has been proposed in consensus layer to improve transaction rate, few work addressed the problem from the P2P network layer. HGFRR makes use of the recursive structure to broadcast messages so that messages redundancy could be reduced to the lowest. HGFRR also take the geographical locality of each node into consideration to further improve the efficiency of a broadcast operation. HGFRR is robust even in a dynamic network environment and is hidden from outsiders. Although HGFRR is not as robust as Gossip, it is evaluated and analyzed that the robustness of HGFRR is sufficient in blockchain system context. By trading robustness, HGFRR improves the throughput of blockchain systems by increasing broadcast efficiency. In the future, the transaction rate of the blockchain systems can be improved further in a low bandwidth-consumption or crowded network.

References

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

[2] Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., De Caro, A., Enyeart, D., Ferris, C., Laventman, G., Manevich, Y., et al. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the Thirteenth EuroSys Conference* (2018), ACM, p. 30.

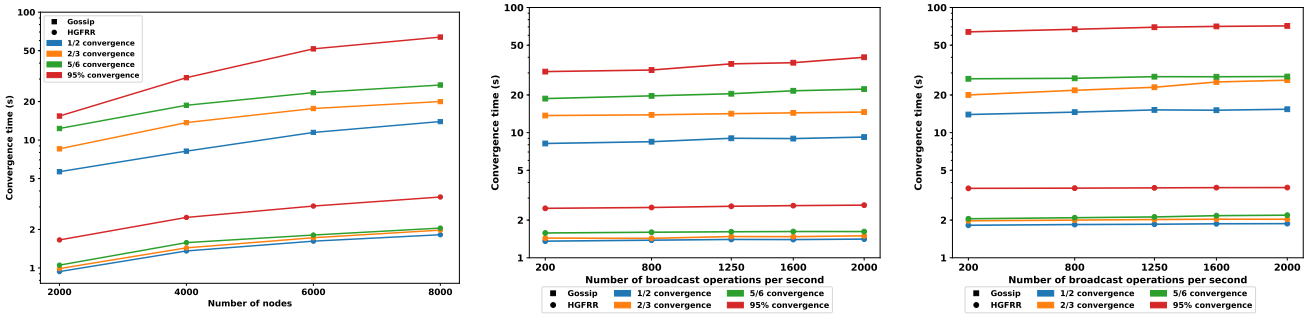
[3] Asokan, N., Ekberg, J.-E., Kostiainen, K., Rajan, A., Rozas, C., Sadeghi, A.-R., Schulz, S., and Wachsmann, C. Mobile trusted computing. *Proceedings of the IEEE 102*, 8 (2014), 1189–1206.

[4] 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.

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

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

[7] Carlsson, N., and Eager, D. L. Peer-assisted on-demand streaming of stored media using bittorrent-like



(a) Convergence time with respect to number of nodes.

(b) Convergence time with respect to broadcast rate under 2K nodes.

(c) Convergence time with respect to broadcast rate under 6K nodes.

Figure 4: Comparison of convergence time between Kademlia-based Gossip and HGFRR .

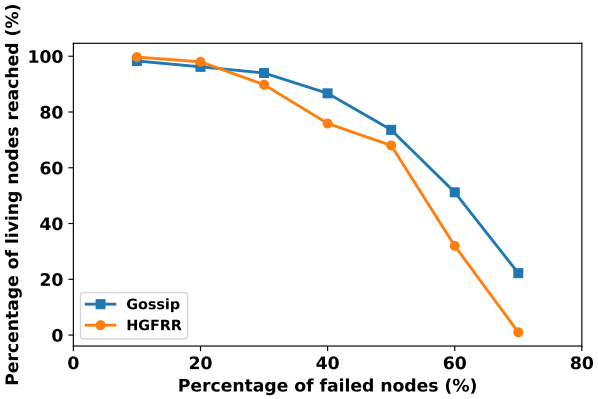
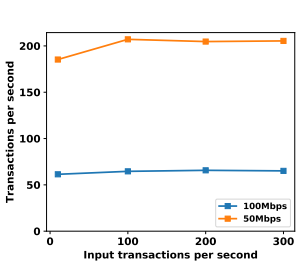
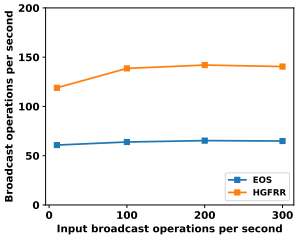


Figure 5: Fault tolerance of HGFRR and Gossip.



(a) End-to-end performance of EOS transactions.



(b) Broadcast performace of HGFRR and the network of EOS.

Figure 6: Comparison betweem EOS and HGFRR .

protocols. In *International Conference on Research in Networking* (2007), Springer, pp. 570–581.

[8] 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.

[9] CHAPPELL, L., AND COMBS, G. *Wireshark network analysis: the official Wireshark certified network analyst study guide*. Protocol Analysis Institute, Chappell University, 2010.

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

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

[12] 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.

[13] 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.

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

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

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

[17] GILAD, Y., HEMO, R., MICALI, S., VLACHOS, G., AND ZELDOVICH, N. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles* (2017), ACM, pp. 51–68.

[18] 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.

[19] 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.

[20] 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.

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

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

[23] 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.

[24] JELASITY, M. Gossip. In *Self-organising software*. Springer, 2011, pp. 139–162.

[25] 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.

[26] KASHYAP, S., DEB, S., NAIDU, K., RASTOGI, R., AND SRINIVASAN, A. Efficient gossip-based aggregate computation. In *Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems* (2006), ACM, pp. 308–317.

[27] KAUNE, S., LAUINGER, T., KOVACEVIC, A., AND PUSSEP, K. Embracing the peer next door: Proximity in kademlia. In *Peer-to-Peer Computing, 2008. P2P'08. Eighth International Conference on* (2008), IEEE, pp. 343–350.

[28] 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.

[29] 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.

[30] KOKORIS-KOGIAS, E., JOVANOVIC, P., GASSER, L., GAILLY, N., SYTA, E., AND FORD, B. Omniledger: A secure, scale-out, decentralized ledger via sharding. In *2018 IEEE Symposium on Security and Privacy (SP)* (2018), IEEE, pp. 583–598.

[31] 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.

[32] LI, C., LI, P., XU, W., LONG, F., AND YAO, A. C.-C. Scaling nakamoto consensus to thousands of transactions per second. *arXiv preprint arXiv:1805.03870* (2018).

[33] 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.

[34] 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.

[35] 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.

[36] 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.

[37] 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.

[38] 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.

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

[40] PADMANABHAN, V. N., WANG, H. J., CHOU, P. A., AND SRIPANIDKULCHAI, K. 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* (2002), ACM, pp. 177–186.

[41] 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.

[42] QIU, T., CHEN, G., YE, M., CHAN, E., AND ZHAO, B. Y. Towards location-aware topology in both unstructured and structured p2p systems. In *Parallel Processing, 2007. ICPP 2007. International Conference on* (2007), IEEE, pp. 30–30.

[43] 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.

[44] RISSON, J., AND MOORS, T. Survey of research towards robust peer-to-peer networks: Search methods. *Computer networks 50*, 17 (2006), 3485–3521.

[45] 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.

[46] 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).

[47] 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.

[48] 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.

[49] 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.

[50] STUTZBACH, D., AND REJAIE, R. Understanding churn in peer-to-peer networks. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement* (2006), ACM, pp. 189–202.

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

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

[53] ZHANG, X., LIU, J., LI, B., AND YUM, Y.-S. 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* (2005), vol. 3, IEEE, pp. 2102–2111.

[54] ZHAO, B. Y., HUANG, L., STRIBLING, J., RHEA, S. C., JOSEPH, A. D., AND KUBIATOWICZ, J. D. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on selected areas in communications 22*, 1 (2004), 41–53.

[55] ZHENG, Z., XIE, S., DAI, H.-N., AND WANG, H. Blockchain challenges and opportunities: A survey. *Work Pap.–2016* (2016).