HGFRR: Hidden Geographical Fractal Random Ring

Anonymous Authors

Abstract

This is the abstract.

1 Introduction

A blockchain is essentially a distributed ledger that permanently records the transactions among parties [14]. The transactions recorded are verifiable and resistant to modification. This feature of security has led to the emergence of cryptocurrencies which leverages the blockchain system as their cornerstone, the most salient example being Bitcoin [25]. Despite the popularity of cryptocurrencies, general and all-purpose blockchain systems that accommodate various applications, such as Ethereum [34] have been proposed. However, although Ethereum is a Turing-complete system [34], 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 system in different scenarios such as Proof-of-Stake [18] and Proof-of-Luck [23]. Hyperledger Sawtooth [15] utilizes Intel Software Guard eXtentions (SGX, introduced below) to trust nodes and proposes Proof-of-Elapsed Time believed to be highly efficient, yet scalibility might not be a critical performance in this consensus. GEEC is a new public blockchain protocol which leverages the strong confidentiality and integrity of the hardware [7]. Its throughput is comparable to Visa's and is 1.7X to 88.5X higher than Ethereum [4], EOS [10], and Intel-PoET [27].

With the increasing of the blockchain system efficiency and transaction rate, broadcast frequency is growing. However, current blockchain systems are based on Distributed Hash Table (DHT) structure. It is efficient in terms of node discovery and data look up, however, broadcasting messages on top of this kind of network structure suffers from traf-fic congestion and inefficient convergence problems when transaction rate gets higher. Our key insights 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 [9], Peer-Assisted [5], LiveBT [21], and Give-To-Get [24]; (ii) audio/video conferencing applications deal with small scale point-to-point connected networks which requires low latency, e.g. Skype [3]; (iii) peer-to-peer file sharing makes efficient indexing and searching possible, e.g. Napster [31], Gnutella [28], and KaZaA [12]; (iv) video streaming applications enables single-source broadcasting efficient, e.g. SplitStream [6], Bullet [19], and ChainSaw [26]. (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 requirements for the underlying P2P network from blockchain systems: peer discovery and message dissemination. There are related works focused on improving peer discovery and message dissemination in a separated way. For example, DHT-based

protocols such as Kademlia, Chord, Pastry, Tapestry, CAN focus on peer discovery, while Gossip-based protocols and tree-based protocols focus on message dissemination. There is very few works addressing both issues at the same time. To fill to gap, our key insight is that broadcasting using the DHT-based structure 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 requirements for the P2P networks from blockchain systems are summarized. Second, a new network protocol that addresses both peer discovery and message dissemination. 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 cross-platform.

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 [20]. 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 [11] 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 [33] 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 [29], 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 [16]. Gossip protocols are highly used for reducing control message overhead [13]. 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 [22], 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[32] or the Gnutella[28], 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) [30]. 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 [8]. 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 [7] 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 [17]. 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

This is the design.

3.1 Design Principles

Talk about the basis of the design.

3.2 Topology

Talk about the p2p network structure. [TODO: add figure]

The network topology is basically a recursive ring-fractal structure. At the top level resides the largest ring where several sub-ring resides on. The figure shows a network of 3 levels. Level 1 is the largest ring. On the largest ring, there are three sub-rings of 2 levels. Level 2 is the second largest ring and level 3 is the smallest ring. The nodes in red are contact nodes.

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.

3.2.1 Structure Formation

Talk about the p2p network formation. boostrap.

Node-Join is the process of a new node joining the network. When a new node wants to join the network, it will send message to the contact nodes of the largest ring. The contact node will judge which sub-ring this new node should be added to, based on some metrics related to locality. Recursively, 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 in the lowest level will then add the new node to the ring.

If the number of node on a ring exceeds a threshold, then this ring will transform to a two-level ring (See Figure 4.2), i.e. several groups of nodes on the large ring will form several rings.

3.2.2 Structure Maintenance

Talk about the p2p network maintenance.

Node-Leave is the process of the network reacting to node-leave. Each node on the 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, the node will double check this with the neighbor of the dead node. 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 node(s) 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 4.2 will be performed.

3.3 Broadcast

Talk about broadcast mechanism.

Broadcast is the process of disseminating a message from one node to the whole network. When a node wants to send a message to the whole network, it will first contact the contact node 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 find the contact node of the largest ring by recursively contacting the contact node in the upper level. Once the contact node in the largest ring is found, it will broadcast recursively in the rings and sub-rings.

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

3.3.1 Broadcast Mechanism

We devised a k-ary distributed spanning tree method to facilitate the in-ring broadcast in the protocol. The basic idea is to generate a random number k, form a k-ary spanning tree, and broadcast 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 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.

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.

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.

9 Acknowledgments

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

10 ATC Template For Reference

```
int wrap_fact(ClientData clientData,
              Tcl_Interp *interp,
              int argc, char *argv[]) {
    int result;
    int arg0;
    if (argc != 2) {
        interp->result = "wrong # args";
        return TCL_ERROR;
    }
    arg0 = atoi(argv[1]);
```

Figure 1: Wonderful Flowchart

```
    result = fact(arg0);
    sprintf(interp->result,"%d",result);
    return TCL_OK;
}
```

Now we're going to cite somebody. Watch for the cite tag. Here it comes [33]. The tilde character (˜) in the source means a non-breaking space. This way, your reference will always be attached to the word that preceded it, instead of going to the next line.

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

11   This Section has SubSections

11.1   First SubSection

Here's a typical figure reference. The figure is centered at the top of the column. It's scaled. It's explicitly placed. You'll have to tweak the numbers to get what you want.

This text came after the figure, so we'll casually refer to Figure 1 as we go on our merry way.

11.2   New Subsection

It can get tricky typesetting Tcl and C code in LaTeX because they share a lot of mystical feelings about certain magic characters. You will have to do a lot of escaping to typeset curly braces and percent signs, for example, like this: "The `%module` directive sets the name of the initialization function. This is optional, but is recommended if building a Tcl 7.5 module. Everything inside the `%{`, `%}` block is copied directly into the output. allowing the inclusion of header files and additional C code."

Sometimes you want to really call attention to a piece of text. You can center it in the column like this:

```
_1008e614_Vector_p
```

and people will really notice it.

The noindent at the start of this paragraph makes it clear that it's a continuation of the preceding text, not a new para in its own right.

Now this is an ingenious way to get a forced space. `Real *` and `double *` are equivalent.

Now here is another way to call attention to a line of code, but instead of centering it, we noindent and bold it.

```
size_t :   fread ptr size nobj stream
```

And here we have made an indented para like a definition tag (dt) in HTML. You don't need a surrounding list macro pair.

> `fread` reads from `stream` into the array `ptr` at most `nobj` objects of size `size`. `fread` returns the number of objects read.

This concludes the definitions tag.

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

[4] BUTERIN, V., ET AL. A next-generation smart contract and decentralized application platform. *white paper* (2014).

[5] 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.

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

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

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

[9] 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.

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

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

[12] 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.

[13] 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.

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

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

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

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

[26] 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.

[27] PRISCO, G. Intel develops sawtooth lakedistributed ledger technology for the hyperledger project. *Bitcoin Magazine* (2016).

[28] 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.

[29] 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.

[30] 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).

[31] 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.

[32] 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.

[33] 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.

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

Notes

[1] Remember to use endnotes, not footnotes!