Blockchain Peer-to-peer Network: Performance and Security

Phuc D. Thai, Minh Doan, Wei Liu, Tianming Liu, Sheng Li, Hong-sheng Zhou, and Thang N. Dinh[†]

Abstract The security and performance of blockchain systems such as Bitcoin critically rely on the P2P network. This paper aims to investigate the blockchain P2P networks. We explore the topologies, peer discovery, and data forwarding and examine the security and performance of the P2P network. Further, we formulate an optimization problem to study the theoretical limit of the performance and provide a solution to achieve optimal performance in a blockchain P2P network.

1 Introduction

In a blockchain system, all nodes share and agree on a common ledger, consisting of transactions, data, timestamps, organized into blocks. Every block is linked to a previous one, via a cryptographic hash link, forming a chain (hence, the name).

[†] This work is supported in part by NSF under the grant CNS-2140411

Phuc D. Thai
Virginia Commonwealth University, USA, e-mail: thaipd@vcu.edu

Minh Doan
Harmony.one, e-mail: minh@harmony.one

Wei Liu
Mayo Clinic, USA, e-mail: Liu.Wei@mayo.edu

Tianming Liu
University of Georgia, USA, e-mail: tliu@uga.edu

Sheng Li
University of Georgia, USA, e-mail: sheng.li@uga.edu

Hong-sheng Zhou
Virginia Commonwealth University, USA, e-mail: hszhou@vcu.edu

Thang N. Dinh
Virginia Commonwealth University, USA, e-mail: tndinh@vcu.edu

Blockchain has been positioned as a disruptive force to decentralize economy and society structures. It removes the need for trust in a single party as well as single point-of-failures in existing centralized systems, making those systems more resilient against censorship [1]. Major applications of blockchain include cryptocurrencies such as Bitcoin [2] and Ethereum [3], decentralized finance [4], decentralized AI [5], the Internet-of-Things [6, 7] and Digitial Health [8, 9].

A P2P network is required to propagate data (blocks, transactions, or other control messages) in a blockchain system. To agree on a common ledger, nodes in the blockchain system are required broadcast (or multicast) data (blocks, transactions, or other control messages) via the P2P network. Recent studies show that the network can become the bottleneck and root-cause for the scalability [10, 11] and security [12] of a blockchain system. Thus, it is crucial to study the security and performance of the blockchain P2P networks.

As the performance of a blockchain system is affected by the performance of the P2P network. We aim to investigate the theoretical limit of the performance of the P2P network. In [13], Kumar and Ross study the theoretical limit on the throughput of a P2P with a single source (data is originally generated at the source node). The study shows that the throughput is bounded by the bandwidth of the source node and the average bandwidth of all nodes. A propagation scheme is proposed to achieve the optimal throughput. However, in a blockchain system, any node can be the source and broadcast data to other nodes. Thus, we cannot directly apply the result in this study to a blockchain P2P network.

Our contributions are summarized as follows

- We provide an overview of some important aspects of blockchain P2P networks. We present the schemes to form the network topologies (including peer discovery and connection rule and forward data. Further, we study the security and performance of the blockchain P2P networks.
- We formulate an optimization problem to study the theoretical limit of the performance (throughput and latency). We also provide a solution to achieve optimal performance in a blockchain P2P network.

Organization. A summary of our paper is given is Fig. 1. In Section 2, we provide an overview of some importance aspects of blockchain networks. Next, we discuss the P2P network topologies and data forwarding scheme in Section 3. Section 4 provides some attacks on the P2P network. In Section 5, we study the performance (throughput and latency) of blockchain systems. Finally, in Section 6, we formalize an optimization problem for the performance and present a solution to achieve optimal performance.

1.1 Related Work

We present the related works on P2P network optimization before the appearance of the Bitcoin [2]. The problem of designing an optimal network for data distribution,

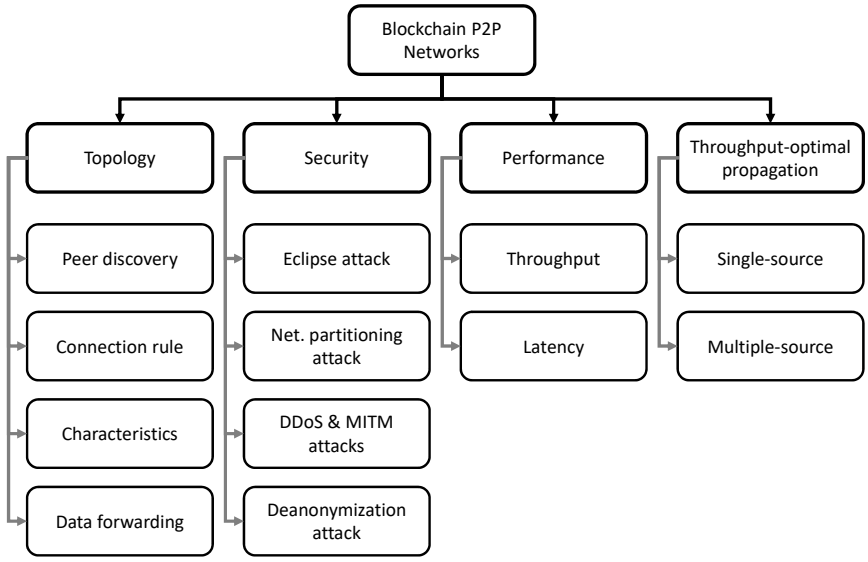


Fig. 1: Paper organization.

e.g. video streaming, from a *single source* was investigated in Kumar and Ross [13]. The paper presents a transmission schedule in which a single source node sends packets to a set of target nodes, optimizing both throughput and latency. In their model, the bandwidth bottlenecks are assumed to be in the uploading and downloading link rates instead of in the Internet core, and every node participates in the system until all nodes finish downloading packets. The paper shows a tight bound on the throughput as a function of the upload bandwidth of the source node, and the upload and download bandwidth of the target nodes. The transmission schedule in [13] can achieve both optimal throughput and latency. However, the constructed network is a complete network with $O(n^2)$ links. Our blockchain network design, studied in this paper, is different in two important aspects a) the all-to-all broadcast and b) the designed network is required to be sparse for practicality.

SplitStream [14] constructs a transmission schedule from a single source node by constructing multiple multicast trees in which an interior node in one tree is a leaf node in all other trees. This ensures that the forwarding load can be well-balanced divided into all nodes while achieving low latency and sparsity. If all nodes have the same capacity, we can achieve optimal throughput. The paper, however, does not consider heterogeneous capacities and provides no theoretical guarantees on the throughput and latency.

In [15], the single-source broadcast problem was solved by dividing nodes into clusters of size $\log(n)$, in which the total capacity of each cluster is roughly the same. Nodes in each cluster are fully connected to each other in order to obtain optimal throughput in each cluster. Clusters are then abstracted as super-nodes, and multiple

broadcast trees are then built over those super-nodes. Since the capacity of each cluster is almost the same, they can achieve within a factor $(1 - \epsilon)$ of the optimal throughput.

Many blockchain systems construct their P2P network using Distributed Hash Table (DHT). For example, Ethereum[4] uses Kademlia [16] to construct the overlay network. In DHT, an overlay network is constructed by assigning nodes with different identifiers. For any message that is assigned with a key, nodes can efficiently choose some other nodes to forward the message based on the identifiers that are stored in the DHT. Most DHTs such as Pastry [17], Chord [18], and Kademlia [16] guarantee that any message can be delivered to any nodes in $O(\log n)$ steps.

2 Overview on blockchain P2P networks

In a *blockchain* system, participants, called *miners*, follow a consensus protocol to maintain the ledger (records of transactions). Based on the consensus protocol, some miners are selected to create new blocks that consist of multiple transactions. The blocks are linked together via a cryptographic hash.

A blockchain realizes a distributed ledger, allows participants to agree on a list of transactions without the use of a central authority as a trusted intermediary. Once transactions are added to the ledger, they cannot be removed or altered. The guarantee holds despite Byzantine behavior of a fraction of the participants maintaining the blockchain.

A main application of blockchains arises in the context of cryptocurrencies, by permitting mutually distrusting participants to engage in financial operations securely. These systems guarantee that participants can issue transactions to send or receive money from each other.

A blockchain system runs on top of a P2P network. Nodes exchange information via the P2P network to sync up the state of the blockchain. As each node may download a store the current state of blockchain, it creates redundancy. However, it is necessary for decentralization, removing the single point of failure for blockchain when some nodes in the network go offline. Note that, in the blockchain network, there are some nodes that do not participate in the consensus protocol. They only take the role of forwarding data (and possibly create new transactions). Now, we summarize the important aspects of the blockchain networks.

In a permissionless blockchain system, such as Bitcoin and Ethereum, any node can join the P2P network. A node first runs a peer discovery protocol to find the addresses of the current nodes in the network. Then, the nodes follow some connections rules to establish connections to the discovered nodes. (See more details in Section 3.) When miners generate new blocks (or nodes create new transactions), the data is forwarded through a P2P network using a propagation scheme. Note that, nodes do not forward invalid data. Upon receiving a new block/ transaction, a node

[4] https://github.com/ethereum/wiki

verifies the correctness and adds to its memory (the mempool), if valid. The node only advertises and forwards the information in its mempool.

The security of a blockchain system critically depends on the correctness of the consensus protocol and the set of data each node receives. As we mentioned in the previous paragraphs, data are forwarded through the P2P network, using a propagation scheme. The blockchain P2P network should be resistant to many types of attacks such as eclipse attacks, network partitioning attacks, denial of service (DoS) attacks (see more detail in Section 4). If the P2P network is not secure, the attacker can take that advantage to perform consensus-level attacks, such as double-spending attacks, selfish mining attacks. Additionally, the P2P network should also be resistant to traffic analysis. The curious observers should not be able to trace down which node originally provide the data. For each example, if an attacker knows which transactions are generated by a victim node, the attacker may be able to delay the time that those transactions are added to the ledger.

Another important aspect of the blockchain P2P network is performance. Since nodes may have heterogeneous bandwidth capacity. The data distribution in the P2P network can introduce a bottleneck. Indeed, one of the main issues in blockchain systems is scalability. For example, Bitcoin, Ethereum can only process 4.1, 15 transactions per second, respectively, while Visa has a throughput of $1,700$ transactions per second. Increasing the number of transactions the system processes will require each node to consume more resources such as bandwidth and storage. For example, Bitcoin will be vulnerable to weaker attackers if the throughput increases. The main reason is that blocks containing more transactions take more time to propagate through the P2P network [10]. For the same reason, it is also important to improve the block propagation time (i.e., latency). Note that, the latency does not only depend on the nodes' bandwidth but also the network topology. We will discuss more details on the throughput and latency in Section 5.

In Section 6, we formalize an an optimization problem to improve performance (throughput and latency) in a blockchain network and present a solution to achieve optimal performance. We show the physical limit of the throughput based on the bandwidth of nodes in the network and summarize a throughput-optimal scheme in [13] that is designed for a single source node. Based on this scheme, we propose a new throughput-optimal scheme that is designed for multiple source nodes.

3 Network topology

In a blockchain system, nodes forward data through the P2P network. To construct the P2P network, nodes first find other nodes' addresses by using a peer discovery process. The peer discovery process typically relies on static information sources and/or distributed hashtable (DHT) approaches. After discovering addresses of other peers, nodes select some peers to establish *outbound connections*. Nodes also accept *inbound connections*, i.e., outbound connections from other peers. In this section, we present the process to construct the P2P network in Bitcoin and Ethereum.

3.1 Bitcoin P2P Network

In the Bitcoin network, all nodes are "equal" in the sense that there is no server, centralized service, or hierarchy within the network. However, they may take different roles based on their functionalities. A Bitcoin node may support one or more of the four following main functions [19]: routing, the blockchain database, mining, and wallet services. Depending on the functions that nodes support, they may download different types of data. For example, a node that supports the blockchain database function should download all blocks on the blockchain with the transactions included. While a node that only supports wallet services (Simplified Payment Verification or SPV node) only download block headers and does not download the transactions included in each block.

Peer discovery. When a new node boots up, it must discover other Bitcoin nodes on the network in order to establish connections. When a node joins the network for the first time, it discovers other nodes by making DNS queries to DNS seed servers. A *DNS seed server* is a server that responds to DNS queries from bitcoin nodes with a list of IP addresses for bitcoin nodes. The size of the list is limited by constraints on DNS; it turns that the maximum possible number of IP addresses that can be returned by a single DNS query is around 4000 [20]. The DNS seed servers are hardcoded into the Bitcoin core client software. At the time of writing (April 2021), there are currently 9 seed addresses listed in the bitcoin software. The DNS seed servers are maintained by Bitcoin community members. For example, some of them provide dynamic DNS seed servers which automatically get IP addresses of active nodes by scanning the network. In this way, the hardcoded DNS seed servers act as the trusted, authoritative source for initial nodes. After that, as the node interacts on the network it builds up a local list of active nodes.

Once a node establishes a connection with another node, the node will query it for a list of network nodes that it is aware of by sending a GETADDR message [21]. The node will reply with an *ADDR message*, containing up to 1000 IP address and their timestamps, which are used to obtain network information from peers. If more than 1000 addresses are sent in an ADDR message, the peer who sent the message is blacklisted. Nodes accept unsolicited ADDR messages. An ADDR message is solicited only upon establishing an outgoing connection with a peer; the node responds with up to three ADDR messages, each containing up to 1000 addresses randomly selected from its local list of addresses.

Furthermore, every day, a node sends its own IP address in an ADDR message to each peer. Also, when a node receives an ADDR message with no more than 10 addresses, it forwards the ADDR message to two randomly selected connected peers. To choose these peers, the node takes the hash of each connected peer's IP address and a secret nonce associated with the day, selects the peers with the lexicographically first and second hash values. Finally, to prevent stale ADDR messages from endlessly propagating, each node keeps a known list of the addresses it has sent to or learned from each of its connected peers, and never sends addresses on the known list to its peer. The known lists are flushed daily.

Each node stores addresses it has already seen in two tables: tried and new. The tried table, which consists of 64 buckets, each of which can store up to 64 unique addresses, stores peer addresses the node has already connected to. While the new table, which consists of 256 buckets, each of which can store up to 64 unique addresses, stores the nodes' addresses that the node has received from other peers.

Since the tables of each node can only store a bounded number of addresses, an attacker can fill up the tables with its address. In this case, for any connection rule, the node has no choice but to connect to the nodes that are controlled by the attacker. The attacker now can manipulate which data the node can receive. We will discuss more on this issue the SubsectionSince the tables of each node can only store a bounded number of addresses. An attacker can fill up the tables with its address. In this case, for any connection rule, the node has no choice but to connect to the nodes that are controlled by the attacker. The attacker now can manipulate which data the node can receive. We will discuss more on this issue the Subsection 4.1 (Eclipse attacks).

Connection rules. Each node selects 8 random addresses in its tried and new table to establish *outbound connections*. In detail, for the i-th ($i \in [8]$) outbound connection, a node selects a random address in the tried table with probability

$$\frac{\sqrt{\rho}(10-i)}{i+\sqrt{\rho}(10-i)}.$$

Furthermore, the node selects a random address from the table, with a bias towards addresses with fresher timestamps, i.e., addresses that join the table earlier have less chance to be selected. Each node may also accept *inbound connections* (up to 125 by default).

For the purpose of analyzing the connectivity, there are 2 types of nodes in the Bitcoin network: *private nodes* that do not accept inbound connections and *public nodes* that do accept inbound connections. However, once they have joined the network, public and private nodes are indistinguishable in their operation: both node types perform transaction and block validation and relay valid transactions and blocks to their peers.

Characteristics. Many works have discovered and explored the characteristics of the Bitcoin network. In [22], by running the peer discovery protocol for 45 days during the time horizon from 2018/12/10 until 2019/01/23 and extracts over 162, 000 nodes and 136, 023 unique IP addresses from the Bitcoin main network (some nodes may use the same IP address). 87, 652 IP addresses are reachable (public nodes). The authors also pointed out that the Bitcoin network shows more community structures compared with what should be expected from a random graph network. The top three largest communities consist of almost 40% of nodes.

In [23], using IP geolocation databases, the authors geolocate the address of nodes in the Bitcoin network by country. Most of the nodes are located in the US (23.7%) and EU – Germany (19%), France (6.8%), Netherlands (4.9%), whereas a smaller share is located in China (6.7%).

Although most Bitcoin nodes are located in EU and US, the mining power is actually concentrated in China. Almost 50% of all BTC blocks are mined by 4 major mining pools in China. Plus, in terms of incoming distribution, 4.5% of all nodes holding more than 85% of all mined coins so far.

3.2 Ethereum's P2P Network.

Nodes in the Ethereum use a Distributed Hash Table (DHT) approach (Kademlia [16]) to make connections. In the DHT approach, an overlay network is constructed by assigning nodes with different identifiers. For any message that is assigned with a key, nodes can efficiently choose some other nodes to forward the message based on the identifiers that are stored in the DHT. By using the DHT approach, we can guarantee that any message can be delivered to any nodes in $O(\log n)$ steps.

Peer discovery. The peer discovery in the Ethereum network is quite similar to the one in the Bitcoin network. When a new node joins the Ethereum network for the first time, it connects to some bootstrap nodes that maintain a list of all nodes that are connected to them in a period of time. (At the time of writing, April 2021, there are currently 8 bootstrap nodes in the main Ethereum network.) Then, the bootstrap nodes share the lists of peers with the new node. Finally, the new node synchronizes with the peers to obtain the list of all nodes.

Connection rule. Nodes in the Ethereum network use Kademlia [16] to establish connections to the discovered nodes. Each node is assigned with a NodeID in the 160-bit identifier space, and {key,value} pairs are stored on nodes with IDs close to the key. Here, keys are also 160-bit identifiers. A NodeID-based routing algorithm will be used to locate nodes near a destination key.

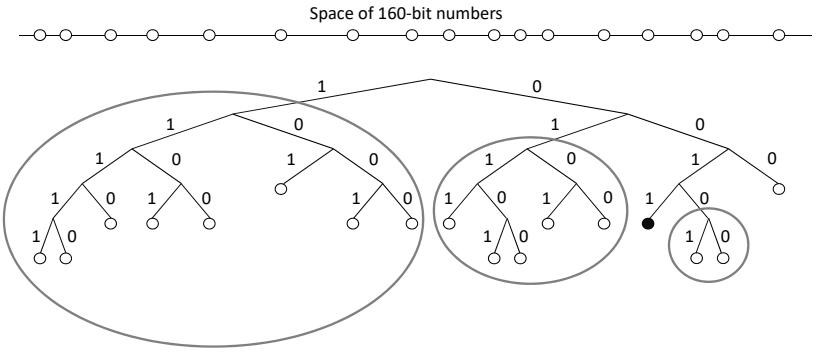


Fig. 2: Kademilia binary tree. The black dot shows the location of the node with NodeID = $0011\cdots$ in the tree. Each gray oval shows a bucket of NodeID as a subtree. (Reproduced from [16]).

To locate {key,value} pairs, node relies on the notion of distance between two identifiers. Given two 160-bit identifiers, a and b, it defines the distance between them as their XOR value, i.e., $d(id_u, id_v) = id_u \oplus id_v = d(id_v, id_u)$, for all pair of identifiers id_u, id_v. XOR also offers the triangle inequality property $d(id_u, id_v) + d(id_v, id_x) \geq d(id_u, id_x)$. Furthermore, XOR is unidirectional, i.e., for any given identifier id_u and distance $d > 0$, there is exactly one identifier id_v such that $d(id_u, id_v) = d$ (the identifier may not be used by any node in the network, but it does exist). The unidirectional approach makes sure that all lookups for the same key converge along the same path, regardless of the originating node.

The node in the network stores a list of {IP address, NodeID} tuples for nodes of distance between 2^i and 2^{i+1} from itself. These lists are called k-buckets. Fig. 2 shows an example of k-buckets of a NodeID. Each k-bucket is kept sorted by last time seen, i.e., least recently accessed node at the head, most recently accessed at the tail. Nodes in the Kademlia routing protocol can send 4 types of messages:

- PING check if a peer is active.
- STORE store a {key,value} pair in a node's table.
- FIND_NODE takes a 160-bit ID, and returns {IP address, NodeID} tuples for the k nodes it knows that are closest to the target ID.
- FIND_VALUE is similar to FIND_NODE: it returns {IP address, NodeID} tuples, except in the case when a node receives a STORE for the key, in which case it just returns the stored value.

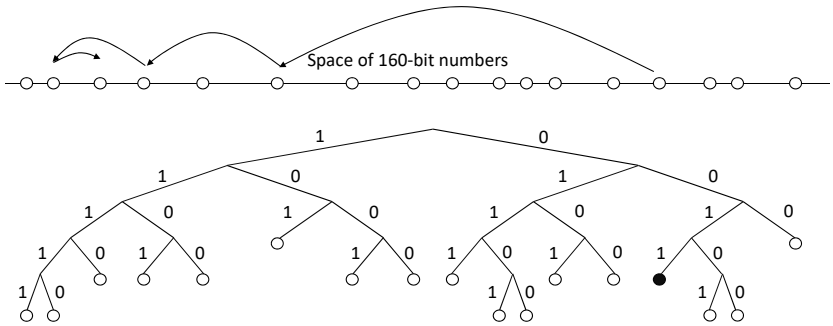


Fig. 3: Node performs a FIND_VALUE lookup to find the k peers. Here, nodes with prefix = $0011\cdots$ finds the nodes with prefix 1110 by successively querying closer and closer nodes. The line segment on top shows how the lookups converge to the target node. (Reproduced from [16]).

Each node locates the k closest peers to some given NodeID. This lookup initiator starts by picking some nodes from its closest non-empty k-bucket, and then sends FIND_NODE messages to the nodes it has chosen. If FIND_NODE fails to return a

node that is any closer than the closest nodes already seen, the initiator resends the FIND_NODE to all of the k closest nodes it has not already queried. It can route for lower latency because it has the flexibility to choose any one of k nodes to forward a request. To find a {key,value} pair, a peer starts by performing a FIND_VALUE lookup to find the k peers with IDs closest to the key. Fig. 3 shows an example of a node performing a FIND_VALUE lookup to find the k nodes with NodeIDs closest to the key.

To join the network, a node u inserts the nodes that it found with peer discovery protocol into the appropriate k-bucket. Then, node u lookup for its own NodeID to select which addresses to establish outbound connections.

Note that, since nodes in the Ethereum network establish outbound connections based on the NodeIDs, the attackers can select some carefully selected NodeIDs to ensure that a victim node will establish outbound connections to those addresses with the NodeIDs. Thus, the attacker can isolate a victim node without filling up the address tables. This makes the Ethereum network more vulnerable to this kind of attack. We will discuss more on this issue the Subsection 4.1.

Characteristics. In studies in [24] discover 769,000 Ethereum active nodes. Most of the nodes only have one IP address. There are 1,268 nodes changing their communication address once within the day. The node that changes its IP addresses most frequently possesses 514 addresses. Similar to another real-world network, the degree distribution in the Ethereum network follows a power-law distribution. The mean of the indegree (or outdegree) is 118.75, the maximum of the indegree is 986, while the number of outdegree is 586.

Similar to Bitcoin, in Ethereum network [25], a few large communities contain a large number of nodes. 43.2% of the nodes operate in the US and 12.9% in China. Furthermore, most of the nodes in Ethereum network runs on several major cloud service. More than 50% of nodes run on the top 8 cloud service providers.

3.3 Data forwarding

We now present the scheme that nodes in a blockchain network are used to forward data over the P2P network. Note that, to prevent malicious nodes from spamming the network with invalid blocks. Nodes use a store-and-forward propagation to ensure that they only forward valid data. To be precise, the origin node, that introduces data into the network, will send the packet (block or transaction) to all of its neighbors, who will verify and forward further to the neighbors of neighbors, and so on.

Transaction/block propagation. To avoid sending packets to a node that already received them, nodes use a *flooding mechanism* to forward the packets instead of forwarding them directly. As shown in Fig. 4, when a node u receives and verifies the validity of a new packet, it advertises the packet to *all* neighbors, except those who already advertised the packets, by sending an invite message[5], containing a set

[5] In Ethereum, the invite messages are equivalent to the $NewBlockHashes$ or $NewPooledTransactionHashes$ messages.

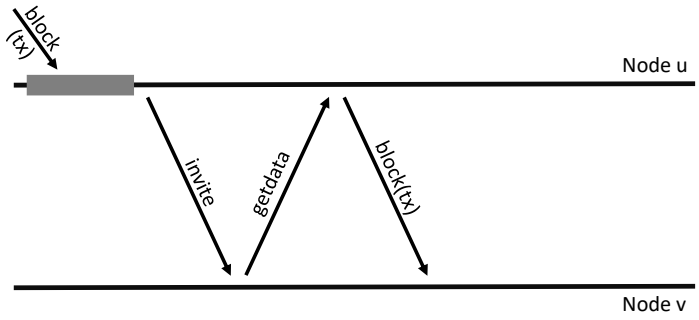


Fig. 4: Data forwarding between nodes. Node u verify the block/transaction before sending the invite message to node v. (Reproduced from [26]).

of packets' hash values. A node v, receiving an invite message for packets that it does not have, will send a getdata message containing the hashes of the needed packets. After that, the actual transfer of the packet is done via individual block or tx messages.

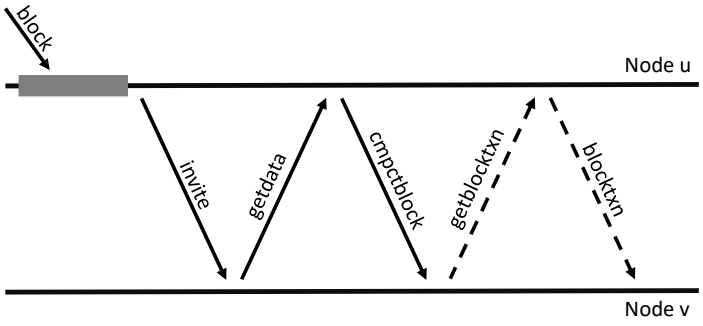
Compact block relay proposal. Forwarding full blocks (that consists of the transactions) is not efficient. The transactions in the block are already forwarded to most of the nodes in the network. In other words, each transaction is forwarded twice, once as an individual transaction and once in a block. Furthermore, this also causes bandwidth spikes when new blocks are generated. When such spikes occur, the network is swarmed and the propagation process may be delayed.

Matt Corallo proposes an improvement, called *compact block relay* (BIP152 [26]), with the goal of decreasing the bandwidth used during block forwarding. Since nodes in the network may download some transactions in the full blocks, nodes forward lightweight compact blocks, instead of forwarding the full blocks, to receiving nodes. These compact blocks include the following information: the 80-byte header of the new block, shortened transaction identifiers (hash values of transactions), and some full transactions which the nodes predict the receiving nodes have not received yet.

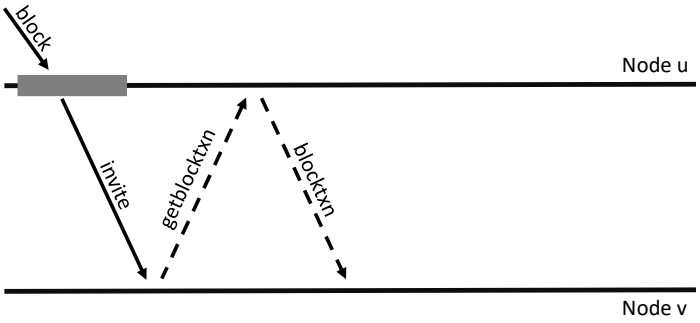
The receiving node then tries to reconstruct the entire block using the received information and the transactions already in its memory pool. If the receiving node is still missing some transactions, it will request those from the node that forwards the compact block.

With compact block relay improvement, in the best case, transactions only need to be forwarded once, when they are originally broadcasted from the users. This provides a large reduction in overall bandwidth consumption. A node can reconstruct a full block of size $1MB$ by receiving a compact block of size $9KB$.

There are two different modes in the compact block relay proposal, *low-throughput* mode and *high-throughput* mode (see Fig. 5). In *low-throughput* mode, after verifying the validity of the block, node u sends new block announcements with the usual invite messages. Then the receiving node v requests the block using a getdata message, which will receive a response of the compact block that consists of the



(a) Low-throughput mode



(b) High-throughput mode

Fig. 5: Data forwarding between nodes using compact block relay. (Reproduced from [26]).

header and short transaction IDs via a cmpctblock. If node *u* already received all transactions in the compact block, it can reconstruct the block immediately. Otherwise, if some transactions are missing, node *v* sends a getblocktxn to request the missing transactions from node *u*. Finally, node *u* will respond by a blocktxn message that contains the lists of missing transactions.

A node can also enable *high-throughput* mode for a few peers. In this mode, node *u* can send the compact block to node *v* with verification or sending invite message. This significantly improves the latency. In the best case, where node v already receives all the transactions in the compact block, it only takes 0.5 round time trip (RTT) to forward data from node u to node v. In the worst, it will take 1.5 RTT which is the same to the legacy data forwarding as in Fig. 4. Note that, node *v* may receive the duplicate compact block from multiple nodes. However, since the size of compact blocks is small, this duplication can be simply ignored.

Note that, although the goal of the compact block relay proposal is not to reduce the latency, it does improve the latency since the size of compact blocks is significantly smaller than the size of the full blocks. (See Subsection 5.2 for more details).

4 Attacks on blockchain P2P networks

In this section, we discuss the security issues in blockchain P2P networks. The security of blockchain critically depends on the security of the P2P network. The attacker can break the security of a blockchain by attacking its P2P network.

4.1 Eclipse attacks

In an eclipse attack, the goal of the attacker is to obscure some target node by attempting to have all the target's connections to the attacker-controlled nodes. Upon success, the attacker can eclipse the information from the target's view and conduct further attacks such as double spending, selfish mining attacks, and 51% attacks.

Flooding address tables attacks. In the Bitcoin network, nodes store the addresses of other nodes in the tried and new tables. In an eclipse attack [27], the attacker fills up the victim's tables with the attacker's addresses. To be precise, the attacker keeps advertising its addresses to the victim. Note that, when an address table of the victim is full, to add a new address to the table, the victim needs to remove an old address. Eventually, all the addresses from honest nodes will be removed from the tables of the victim. Then, when the victim reboot, it can only make connections to the attacker's addresses. The attacker can either actively perform Distributed-Denial-of-Service (DDoS) attacks or simply wait out until the victim restart.

Many countermeasures were proposed in [27] with several of them already implemented in Bitcoin software. One obvious countermeasure is to increase the number of connections to improve the chance honest nodes will be connected to each other. Other countermeasures are to increase the size of the tables or reduce the number of addresses in ADDR messages. This reduces the chance that the addresses of honest nodes get kicked out of the tables. The remaining countermeasures aim to address the vulnerability in adding and removing nodes from the address tables and preventing the adversary from flooding the target's address table with trash addresses. For example, before removing an address from the tried table, the node briefly attempts to connect to the address. If the connection is successful, then the node will not remove the address and add a new address to the table. The proposed countermeasures substantially reduce the chance that the adversary can successfully perform an eclipse attack.

Stubborn mining attacks. In [28], the authors investigate new stubborn mining attacks which combine eclipse attacks with selfish mining [29] attacks. In this work, the authors consider the same model against users who are also eclipsed in the network and show the effect to which eclipsed users help a stubborn mining attacker. Overall, eclipse attacks empower adversarial agents with a larger strategy space to continue running attacks, and when paired with stubborn mining strategies, enable an attacker to improve their relation fraction of block rewards beyond traditional selfish mining strategies.

Eclipse attacks on Ethereum network. The works in [30, 31] show that the Ethereum network is more vulnerable to eclipse attacks. Since nodes in the Ethereum network use Kademlia to select nodes to connect, the attackers can simulate the connection rule of the victims. Thus, it is easier for attackers to perform eclipse attacks.

Newly connected nodes attacks. Due to the block propagation design of Ethereum, a node that newly connects to the network may receive a chain that is longer than the main chain but has a lower total difficulty. To perform this attack [30], the attacker creates a long blockchain starting from the genesis block by decreasing the difficulty for each block. Then, the attacker connects to the victim and advertises a high total difficulty (higher than the difficulty on the main chain). Finally, the victim will request to download the chain from the attacker. The authors point out an implementation bug in Ethereum's difficulty calculation. The attacker can use this bug to present the victim to download the main chain.

Zero-outbound-connections issue. In the Ethereum network, a node may node establish outbound connections if it accepts too many inbound connections. Based on this connection rule, when a victim reboots, the attacker can immediately initiate inbound connections to the victim from its addresses. In this case, the victim is eclipsed since all connections of the victim are from the addresses of the attackers. A simple countermeasure to this attack is enforcing an upper limit on the number of inbound connections.

Kademlia-based attacks in the Ethereum network. Even if an upper limit on the number of inbound connections is enforced, the attacker can use a carefully-crafted set of node's IDs to repeatedly ping the victim (since a node establishes outbound connections to the nodes with IDs that are closest to the ID of that node). When the victim restarts, the victim establishes all outgoing connections to the attacker's address with high probability. To complete the eclipse, the attacker monopolizes the remaining connection slots by initiating inbound connections to the victim. Several countermeasures was proposed to increase the difficulty to perform eclipse attacks. A simple countermeasure is to make the node ID deterministic to the address. Another countermeasure is to make the lookup process non-public. This prevents the attacker from predicting set of nodes' IDs that the victim will connect to when it reboots.

4.2 Network partitioning attacks

Partitioning the blockchain network can have severe consequences. Partitions affect the ability of participants to exchange data. Thus, nodes in different parts may maintain different chains. When all nodes are connected, based on the consensus protocol, only one chain will survive and as a result, the mining power on the other chains is wasted. Plus, many transactions are reverted (and potentially double-spent).

The attacker can partition the blockchain network by using *routing attacks*. The attackers control the Border Gateway Protocol (BGP) advertisements to manipulate

the connections of nodes. Recent studies [32] show that the attackers can isolate 50% of the Bitcoin mining power by hijacking less than 100 IP prefixes.

SABRE [33] presents a transparent relay network protecting Bitcoin clients from routing attacks by providing them with an extra secure channel for learning and propagating the latest mined block. SABRE is easy to deploy and can run alongside the existing P2P network. The IP addresses of the SABRE relay nodes will be publicly known (e.g., via a website), and that everyone can connect to the relay nodes. SABRE is designed to efficiently handle extremely high load and resistance to Denial of Service attacks. The authors use properties of BGP to predict where would be a good place to host relay nodes – locations that are inherently protected against routing attacks and on paths that are economically preferred by the majority of Bitcoin clients. In addition, they provide resiliency through the use of caching, and partially hardware implementation in programmable network devices. This enables SABRE relay nodes to sustain large (D)DoS attackers.

4.3 DDoS attacks

In a blockchain system, nodes can generate as many transactions as they wish if they are able to pay the transaction fees. An attacker can perform a DDoS attack by generating a large number of transactions and forwarding them to some nodes in the network. Then, nodes will unconditionally forward those transactions to the entire network. The main cost for the attacker is the transaction fees. However, it is possible that the attacker does not need to pay those fees. In fact, if transactions are propagated to the entire network but are not included on the blockchain, the fees are not collected, i.e., the attacker can perform a DDoS attack without any cost. Thus, in Bitcoin, miners only forward transactions that pay a sufficient fee and are likely to be included in a block.

4.4 Man-in-the-middle attacks

In [34], the authors study the impact of Man-In-The-Middle Attacks on Ethereum. Based on the properties of the Ethereum public blockchain topology, the authors build a simulated network to mimic the top 10 biggest mining pools of Ethereum. Then, the authors perform BGP hijacking and ARP spoofing to partition the network before issuing a double spending attack. The results demonstrate the attack will be almost impossible in public blockchains (nodes rely on the Internet for communication). However, consortium blockchains (nodes rely on the multiple organization networks that are connected by the Internet) and private blockchains (nodes rely on a single organization network) are suffering from this attack. The attacker can successfully perform a double-spending attack with a probability of 80% with 12 minutes attack duration.

4.5 Deanonymization attacks

An attacker may wish to identify which node originally generates a transaction. By actively connecting to several nodes, it is possible that the attack will trace back to the source node that generates the transaction. Then, the attack may make an attempt to censor the transactions of a victim node.

Biryukov et al. [35] The paper presents a deanonymization method for the Bitcoin network, which allows linking IP addresses of nodes to their pseudonyms. This way, we are able to find out where the transactions are generated. The method explicitly targets nodes behind NAT (Network Address Translation) or firewall. The method also works on nodes that use anonymity services like Tor. Plus, it can distinguish between nodes with the same IP address. The key idea of this deanonymization is that each node can be uniquely identified by a set of nodes he connects to (entry nodes). Each transaction is mapped to a set of entry nodes, which is associated with a node with a similar set. To avoid this deanonymization technique, the authors suggest frequently changing the set of entry nodes.

In [36] Neudecker and Hartenstein evaluate the deanonymization in Bitcoin network in the form of address clustering. The goal of address clustering is to group the addresses into clusters so that the addresses in each cluster are controlled by a single user. The authors compare the blockchain information (e.g., the public keys of users in the transactions) based clustering approaches and the network information based clustering approaches. The majority of nodes have no correlation between network information and the clustering performed on blockchain information. However, a small number of nodes (8%) exhibit correlations that might make them susceptible to network-based deanonymization attacks.

The Dandelion protocol [37] is a transaction relay protocol to improve the anonymity in the presence of honest-but-curious attackers. The Dandelion protocol consists of two phases. In the first phase, each transaction is propagated on a random path, i.e., each node forwards the transaction to exactly one random neighbor for a random number of hops. Then, in the second phase, the transaction will be broadcast using the same flooding mechanism as in Bitcoin. Dandelion++ [38] extends Dandelion to defend against active attackers that can divert from the protocol. Instead of forwarding transactions through a random path (in the first phase), nodes forward transactions over one of two intertwined paths on a 4-regular graph. The second phase remains the same as in Dandelion.

5 Performance

Another important aspect of a blockchain system is performance. In this work, we focus on the throughput, i.e., the number of transactions per second the system can process, and the latency, i.e., the time it takes for a block to propagate to all nodes in the network.

5.1 Throughput

One of the main issues of blockchain is scalability. Most of existing legacy blockchains suffer from very low throughput. For example, in Bitcoin, for every 10 minutes (600 seconds), a new block, that consists of $2,467$ transactions on average[6], is generated. In other words, on average, the Bitcoin system can process 4.1 transactions per second. Ethereum can achieve a better throughput (15 transactions per second). However, comparing with Visa, which does around $1,700$ transactions per second on average, the throughput of the blockchain systems is still too low. Thus, it is essential to make blockchain systems more scalable.

One simple solution that may come to mind is to increase the block size or decrease the block generation time. However, this solution comes with consequences. Without sacrificing the security, to increase the block size or decrease the block generation time, nodes in the network are required a higher bandwidth capacity which some nodes cannot afford. Presumably, to increase the throughput by this solution, fewer and fewer nodes can participate in the system, leading to the increase of centralization. Thus, to improve the throughput without sacrificing security, we should have a more efficient way to propagate data.

Reducing redundancy. Although the flooding mechanism is needed for security purposes, it has a bad effect on the performance. Using the flooding mechanism, nodes are required to send many redundant transaction announcements. Indeed, by using the flooding mechanism, each node sends an announcement on each of the links except the one where that announcement originally arrived. In other words, each link sees each announcement once, if no two nodes ever send the same announcement to each other simultaneously, and more than once if they do. Therefore, in Bitcoin, each announcement is sent at least as many times as the number of links. On the other hand, optimally, each node would receive each announcement exactly once, the number of times each announcement is sent should be equal to the number of nodes. As the size of an invite message is $32B$ (while the average size of a transaction is $459B$ [39]), this redundancy is quite big. According to [40], in Bitcoin, nodes use 48% of bandwidth to send invite, getdata messages.

Erlay [40] minimizes the redundant transaction announcements. Instead of announcing every transaction on each link (flooding), a node advertises it to a subset of peers (low-fanout flooding). This low-fanout flooding help to reduce the number of invite messages for advertising transactions. Note that, to defend against timing attacks (where the attacker can use the timing of transactions arrival to guess whether a transaction originated at its peers), node relay transaction via outbound connections.

Furthermore, to make sure that all transactions reach the entire network, nodes periodically engage in an interactive protocol to discover announcements that were missed, and request missing transactions. This can be done using set reconciliation. Each node performs set reconciliation by computing a local set sketch with a predetermined capacity. When the number of elements in the set does not exceed the

[6] According to [39], at the time of this writing (April 2021), the average block size is $1.08MB$ and the average transaction size is $459B$

capacity, it is always possible to recover the entire set from the sketch. Plus, a sketch of the symmetric difference between the two sets can be obtained by XORing the bit representation of sketches of those sets.

Optimizing throughput of the P2P network. Many works have been studied to optimize the throughput for data propagation in the P2P network. We can borrow those techniques to optimize the throughput of the blockchain P2P network. There are two classes of approaches to propagate data via a P2P network, tree-based or mesh-based. Tree-based approaches explicitly construct multiple spanning trees, connecting the source node to all receivers. In mesh-based approaches, nodes exchange packets with several of their neighbors without explicitly constructing the spanning trees.

We now present a tree-based approach in [15] that can achieve near-optimal throughput. Note that, in this construction, we only consider the single source problem, where a single source node broadcast data to other nodes in the network.

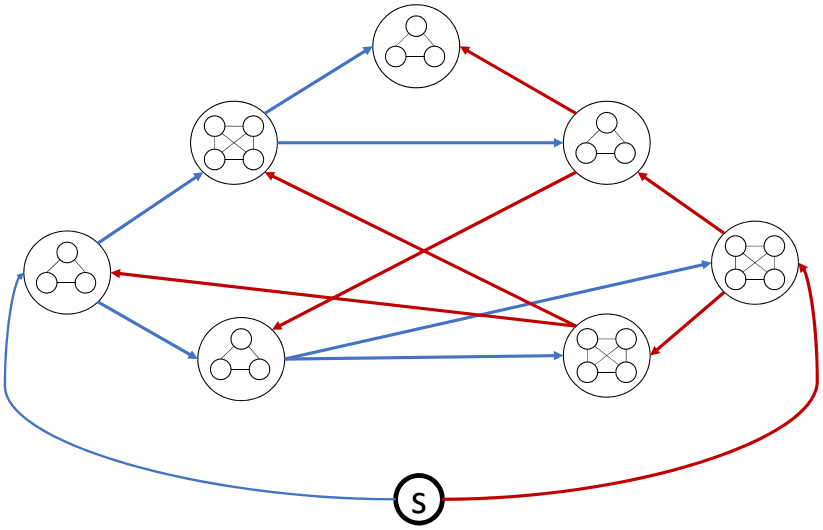


Fig. 6: In this example, the tracker groups the peers into 7 clusters. The server s forms two interior-node-disjoint trees, each having tree degree 2, to distribute video to all clusters. Within each cluster, the peers form a full mesh and locally broadcast the video among themselves. There are not enough clusters left to build a third tree with disjoint interior clusters; hence, some clusters, e.g., the cluster on the top, do not get the chance to serve as interior clusters.

The work in [15] solves single-source broadcast problem by constructing multiple broadcast trees over clusters instead of individual nodes (see Fig. 6). Recall that, with the SplitStream construction, we can achieve near-optimal throughput if the capacity of all nodes are roughly the same. Taking this advantage, in [15], nodes are divided

into clusters of size $O(\log n)$ such that the total capacity of all nodes in each cluster are roughly the same. Nodes in each cluster are fully connected to each other in order to obtain optimal throughput in each cluster. (For example, we can use the construction in [13]). Nodes in each cluster are fully connected to each other in order to obtain optimal throughput in each cluster. Clusters are then abstracted as super-nodes and d broadcast trees are then built (using SplitStream [14]) over those super-nodes. In detail, we construct d broadcast trees that aim to balance the transmission loads of clusters. Here, a cluster only forwards the data equals to the data that the cluster received. In the context of broadcasting, all clusters in the network forward the same data. To be precise, we form d broadcast trees with the same capacity in which, in each tree, each interior cluster has exact d links to forward data to other clusters. Furthermore, if a cluster is an interior cluster in one tree, the will be a leaf in all other trees.

Since the capacity of the clusters is almost the same, we can achieve within a factor $(1-\epsilon)$ of the optimal throughput. We can also achieve the latency of $O(\log n)$. Let k be the depth of a broadcast tree. Since each interior cluster has exact d links, the depth of each broadcast tree is $O(\log n)$. Plus, all nodes in each cluster connect to each other. Thus the length of the propagation path in each cluster is $O(1)$. Thus, we can achieve the latency of $O(\log n)$.

5.2 Latency

The latency of the P2P network affects the security of the systems. The security analysis of Bitcoin [41] is based on the assumption that all packets can be delivered to all nodes with a bounded delay. By reducing the latency (i.e., the bounded delay), we can reduce the confirmation time, i.e., the time it takes for a transaction on the blockchain becomes irreversible.

Even though the goal of the compact block relay proposal (as we presented in the paragraph in Section 3.3) is not to reduce the latency, it does improve the latency since the size of compact blocks is significantly smaller than the size of the full blocks. In [42], the authors run simulations of Bitcoin with and without the compact block relay proposal, with 2019 Internet parameters on a network of $9,000$ nodes and the block size is $1MB$. Without compact block relay proposal, the 50%, 90% latency (the time it takes for a block to reach 50%, 90% of nodes in the network) is $6.4s$ and $9.4s$, respectively. With the compact block relay proposal, the latency reduces to $1.3s$ and $2.4s$, respectively. The authors also measure the number of orphan blocks. With the compact block relay proposal, the fraction of orphan blocks reduces from 0.95% to 0.19%.

Relay network is a network that attempts to minimize the latency in the transmission of blocks between miners. The *original Bitcoin relay network* [43] was created by core developer Matt Corallo in 2015 to enable fast synchronization of blocks between miners with very low latency. The network consisted of multiple gateways

infrastructure around the world and served to connect the majority of nodes in the network.

The original Bitcoin Relay Network was replaced in 2016 with the introduction of the *Fast Internet Bitcoin Relay Engine* or FIBRE [44], also created by core developer Matt Corallo. FIBRE uses a similar architecture while using UDP-based transmission instead of TCP-based. TCP-based is designed to provide reliable transmission at reasonable bandwidth across medium-large amounts of data, it is incredibly bad at low-latency relay of small amounts of data. It is generally tuned to send packets (each just under 1500 bytes) once and to only discover that some packets were lost after getting a response from the other side. Only then will the sender retransmit the lost packets, allowing the receiver to (potentially) reconstruct the original transmission. Thus, in order to have minimal latency block transmission, we must avoid the need for retransmissions at all costs. In order to do so, we must transmit enough extra data that the receiving peer can reconstruct the entire block even though some packets were lost on the way. The common solution is UDP-based transmission with some relatively simple linear algebra to send data which can fill in gaps of lost packets efficiently.

Falcon [45] is a relay network that uses cut-through-routing instead of store-and-forward propagation model to reduce latency. Here, nodes propagate parts of blocks as soon as they arrive rather than waiting for the entire block to arrive.

BloxRoute [11] is a high-capacity, low-latency Blockchain Distribution Network that is optimized to quickly propagate transactions and blocks for blockchain systems. Contrary to the relay networks, the BloxRoute propagates data without knowing the content of the data. This prevents BloxRoute from censor the specific data by intentionally delaying the propagation of the data.

6 Performance improvement as an optimization problem

In this section, we formalize the performance (throughput and latency) improvement problem in a blockchain network as an optimization problem and present a solution to achieve optimal performance.

6.1 Optimization problem

Consider a blockchain system with n participants that are modeled as a set of nodes $V = \{1, 2, \ldots, n\}$. Each node $i \in V$ has an upload bandwidth $c(i)$, or c_i for short. A node i can transmit simultaneously to each neighbor j with rate g_{ij} as long as $\sum_j g_{ij} \leq c(i)$. We refer to this as the *capacity constraint*.

During the execution of the system, some source node i may produce new data (e.g., block) to other nodes. We denote the *arrival rate* λ_i as the average data that is produced by node i. For any node i, the arrival rate cannot exceed the upload

throughput of, i.e., $\lambda_i \leq c_i, \forall i \in V$. For example, in Bitcoin, since the size of compact blocks is relatively small (the average size of compact blocks is 9KB, while the average size of transactions is 459KB [39]), the arrival rate can be approximated as the transaction generation rate.

Throughput. The throughput is the data nodes can broadcast to all nodes. We say a propagation scheme can achieve a throughput TP iff there exists an arrival rate $\lambda = (\lambda_1, \cdots, \lambda_n)$ such that $\sum_{i \in V} \lambda_i = \mathsf{TP}$, the propagation scheme, that satisfies the capacity constraint, is able to deliver such data to all nodes in the network.

Lemma 1 *Consider the set of node $V = (1, \cdots, n)$ with the upload capacity $C = (c_1, \cdots, c_n)$. Let OPT_{TP} be the optimal throughput the network can achieve. The upper-bound on the optimal throughput can be simplified into*

$$\mathsf{OPT}_{TP} \leq \frac{\sum_{i \in V} c_i}{n-1}. \tag{1}$$

Proof Since each node needs to forward its data to $n-1$ other nodes, to achieve a throughput of OPT_{TP}, the total data nodes need to forward is $(n-1)\mathsf{OPT}_{TP}$. The total amount of data nodes can forward is $\sum_{i \in V} c_i$. Thus, we have,

$$(n-1)\mathsf{OPT}_{TP} \leq \sum_{i \in V} c_i.$$

Jumping ahead, in subsection 6.3, we show by construction that we can achieve the optimal throughput $\mathsf{OPT}_{TP} = \frac{\sum_{i \in V} c_i}{n-1}$.

Latency. The latency is the time for any source nodes to transmit the data to all other nodes. In our model, for simplicity, we measure the latency based on the number of hops on the propagation paths. To be precise, the latency equals the length of the longest path that a packet travels from any source node to any target node.

The goal of our problem is to construct a propagation scheme that can support high throughput, while maintaining low latency.

6.2 Throughput-optimal propagation scheme for single-source problem

In [13], Kumar and Ross present a throughput-optimal propagation scheme for single-source problem. In the single-source problem, there is only one source node s can provide data to other nodes, i.e., for any, node $v \in V \setminus \{s\}$ the arrival rate $\lambda_v = 0$.

Note that, since the arrival rate is bounded by the capacity, i.e., $\lambda_s \leq c_s$, the optimal throughput in the single-source problem is also bound by the capacity of the source node s, i.e.,

$$\min \left\{ c_s, \frac{\sum_{i=1}^{n} c_i}{n-1} \right\}.$$

The source node s first a small data to all other nodes. We denote p_v as the data node s forwards to node v. Then, each node v forwards $q_v \leq p_v$ data to all node

$u \in V \setminus \{s, v\}$. Here, each node needs to received λ_s data, i.e.,

$$p_v + \sum_{u \in V \setminus \{v,s\}} q_v = \lambda_s, \forall v \in V \setminus \{s\}.$$

Plus, the capacity constraint also need to be satisfied, i.e.,

$$\sum_{v \in V \setminus \{s\}} q_v \leq c_s,$$
$$(n-2)q_v \leq c_v, \forall v \in V \setminus \{s\}.$$

We consider two cases in the bound of optimal throughput.

- *Case 1:* $c_s \leq \frac{\sum_{i \in V} c_i}{n-1}$. In this case, we can achieve a throughput of c_s, i.e., $\lambda_s = c_s$. For all $v \in V \setminus \{s\}$, we set

$$q_v = p_v = \lambda_s \frac{c_v}{\sum_{i \in V \setminus \{s\}} c_i}.$$

- *Case 2:* $c_s > \frac{\sum_{i \in V} c_i}{n-1}$. In this case, we can achieve a throughput of $\frac{\sum_{i \in V} c_i}{n-1}$, i.e., $\lambda_s = \frac{\sum_{i \in V} c_i}{n-1}$. For all $v \in V \setminus \{s\}$, we set

$$q_v = \frac{c_v}{n-2}$$
$$p_v = q_v + \frac{(n-1)c_s - \sum_{i \in V} c_i}{(n-1)(n-2)}$$

We omit the analysis of this propagation scheme since it can be considered as a subcase of the propagation scheme in Subsection 6.3.

6.3 Throughput-optimal propagation scheme for blockchain data forwarding problem

We now modify the propagation scheme in [13] to solve the blockchain data forwarding problem where the arrival rate of any node $s \in V$ can be bigger than zero. Here, the bound on the capacity of the source node is removed.

Similar to the single-source problem, each source node s forwards a small amount of data to all other nodes. We denote p_{sv} as the data node s forwards to node v. Then, each node v forwards the same data from node s to all node $u \in V \setminus \{s, v\}$. Each node v needs to receive an λ_s data from s, i.e.,

$$p_{sv} + \sum_{u \in V \setminus \{s,v\}} q_{su} = \lambda_s.$$

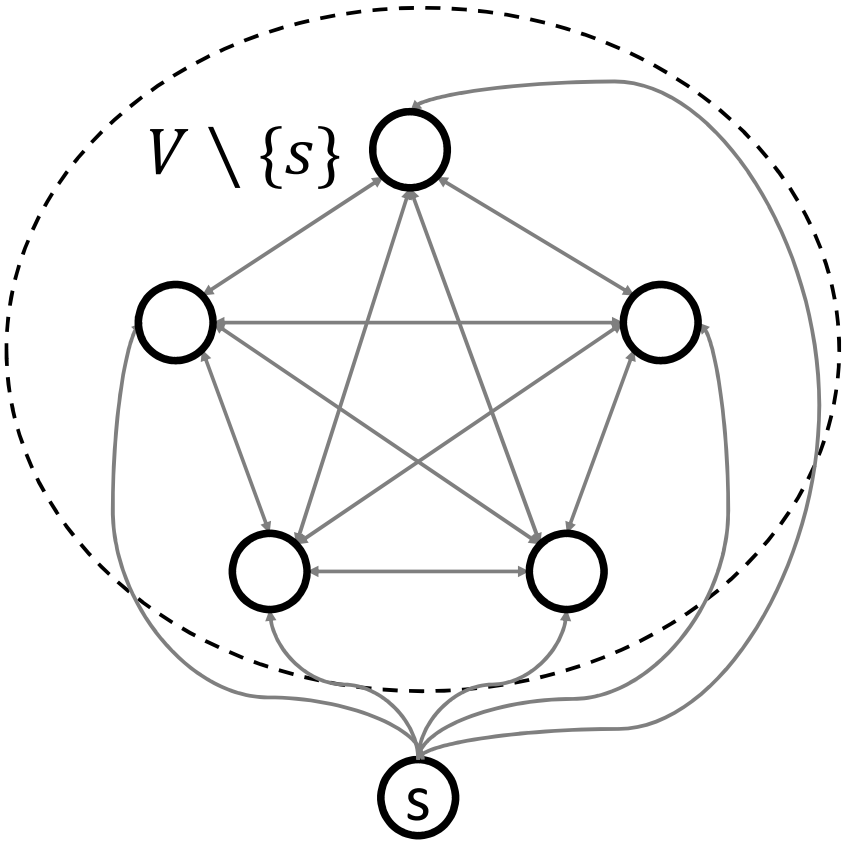


Fig. 7: Each source node first forwards a small amount of data to all other nodes the desire to receive the data. Then, all nodes exchange data of that source node with each other. Note that, data can be forwarded from any source node.

For the capacity constraint, when s is the source node, it sends p_{sv} to all other nodes v. Plus, when a node $v \in V \setminus \{s\}$ is the source node, s send $(n-2)$ packets of size q_{vs} to all nodes $u \in V \setminus \{s, v\}$. Thus, the capacity constraint can be written as

$$\sum_{v \in V \setminus \{s\}} (p_{sv} + q_{vs}\,(n-2)) \leq c_s$$

In detail, for each source node $s \in V$ and a node $v \in V \setminus \{s\}$, we assign p_{sv} and q_{sv} as in Algorithm 1. We denote $c_v^{(s)}$ as the remaining capacity after nodes forwarded data for the first s source nodes. Note, at the beginning, each node s reserves λ_s of its capacity. This ensure that node s have enough capacity to forward its own data.

Algorithm 1: Constructing propagation scheme.

Input : Given the set of node $V = (1, \cdots, n)$, the upload capacity $C = (c_1, \cdots, c_n)$, and the arrival rate $\lambda = (\lambda_1, \lambda_2, \cdots, \lambda_n)$.

Output : For each source node $s \in V$, and node $v \in V \setminus \{s\}$, assign q_{sv} is the data s send to v and p_{sv} is the data v forward data from s to all node $u \in V \setminus \{s, v\}$.

1 **for** $v = 1$ *to* n **do**
2 $\quad c_v^{(0)} = c_v - \lambda_v$
3 **for** $s = 1$ *to* n **do**
4 $\quad$ **for** $v = 1$ *to* n **do**
5 $\quad\quad$ **if** $\lambda_s \leq \frac{\sum_{i \in V \setminus \{s\}} c_i^{(s-1)}}{n-2}$ **then**
6 $\quad\quad\quad p_{sv} = q_{sv} = \lambda_s \frac{c_v^{(s-1)}}{\sum_{i \in V \setminus \{s\}} c_i^{(s-1)}}$
7 $\quad\quad$ **else**
8 $\quad\quad\quad q_{sv} = \frac{c_v^{(s-1)}}{n-2}$
9 $\quad\quad\quad p_{sv} = q_{sv} + (\lambda_s - \frac{\sum_{i \in V \setminus \{s\}} c_i^{(s-1)}}{n-2})$
10 $\quad$ **for** $v \in V \setminus \{s\}$ **do**
11 $\quad\quad c_v^{(s)} = c_v^{(s-1)} - q_{sv}\,(n-2)$
12 $\quad c_s^{(s)} = c_s^{(s-1)} + \lambda_s - \sum_{v \in V \setminus \{s\}} p_{sv}$
13 Return p, q

At lines 1-2, we set $c_v^{(0)} = c_v - \lambda_v$. Then, we iterate through all node in V. For the source node s, we consider two cases.

- *Case 1:* $\lambda_s \leq \frac{\sum_{i \in V \setminus \{s\}} c_i^{(s-1)}}{n-2}$. In this case, for all $v \in V \setminus \{s\}$, we set

$$p_{sv} = q_{sv} = \lambda_s \frac{c_v^{(s-1)}}{\sum_{i \in V \setminus \{s\}} c_i^{(s-1)}}.$$

- *Case 2:*$\lambda_s > \frac{\sum_{i \in V \setminus \{s\}} c_i^{(s-1)}}{n-2}$. In this case, for all $v \in V \setminus \{s\}$, we set

$$q_{sv} = \frac{c_v^{(s-1)}}{n-2}$$

$$p_{sv} = q_{sv} + (\lambda_s - \frac{\sum_{i \in V \setminus \{s\}} c_i^{(s-1)}}{n-2})$$

Then, we deduct the capacity of each node by the amount of data it uses to forward the data from the source node s. To be precise, each node $v \in V \setminus \{s\}$ forward an q_{sv} data to $n-2$ nodes. Thus, we set,

$$c_v^{(s)} = c_v^{(s-1)} - q_{sv}(n-2).$$

The source node s send p_{sv} data to each node v. Thus, we set,

$$c_s^{(s)} = c_s^{(s-1)} + \lambda_s - \sum_{v \in V \setminus \{s\}} p_{sv}.$$

Here, we plus λ_s as the reserved data for node s to forward data of its own.

Now, we prove that, by assigning p and q as in Algorithm 1, each node receive λ_s data from each source node s (see Lemma 2), and the data each node s forwarded does node exceed c_v (see Lemma 3).

Lemma 2 *Consider the set of node $V = (1, \cdots, n)$, the upload capacity $C = (c_1, \cdots, c_n)$, and the arrival rate $\lambda = (\lambda_1, \lambda_2, \cdots, \lambda_n)$ such that $\sum_{i \in V} \lambda_i \leq \frac{\sum_{i \in V} c_i}{n-1}$.. For each source $s \in V$ and a node $v \in V \setminus \{s\}$, p_{sv} and q_{sv} is assigned as in Algorithm 1. Then, for each source node s, each node $v \in v \setminus s$ receives λ_s data from s, i.e.,*

$$p_{sv} + \sum_{u \in V \setminus \{s,v\}} q_{su} = \lambda_s, \forall s \in V, v \in V \setminus \{s\}$$

Proof We consider two cases as in Algorithm 1.

- *Case 1:* $\lambda_s \leq \frac{\sum_{i \in V \setminus \{s\}} c_i^{(s-1)}}{n-2}$. In this case, we have

$$p_{sv} + \sum_{u \in V \setminus \{s,v\}} q_{su} = \sum_{u \in V \setminus \{s\}} \left(\lambda_s \frac{c_v^{(s-1)}}{\sum_{i \in V \setminus \{s\}} c_i^{(s-1)}} \right) = \lambda_s$$

- *Case 2:*$\lambda_s > \frac{\sum_{i \in V \setminus \{s\}} c_i^{(s-1)}}{n-2}$. In this case, for all $v \in V \setminus \{s\}$, we set

$$p_{sv} + \sum_{u \in V \setminus \{s,v\}} q_{su} = (\lambda_s - \frac{\sum_{i \in V \setminus \{s\}} c_i^{(s-1)}}{n-2}) + \sum_{u \in V \setminus \{s\}} \frac{c_v^{(s-1)}}{n-2} = \lambda_s$$

Lemma 3 *Consider the set of node $V = (1, \cdots, n)$, the upload capacity $C = (c_1, \cdots, c_n)$, and the arrival rate $\lambda = (\lambda_1, \lambda_2, \cdots, \lambda_n)$ such that $\sum_{i \in V} \lambda_i \leq \frac{\sum_{i \in V} c_i}{n-1}$.. For each source $s \in V$ and a node $v \in V \setminus \{s\}$, p_{sv} and q_{sv} is assigned as in algorithm 1. Then, the data each node $s \in V$ forward does node exceed c_s, i.e.,*

$$\sum_{v \in V \setminus \{s\}} (p_{sv} + q_{vs}\,(n-2)) \leq c_s, \forall s \in V \tag{2}$$

Proof Recall from lines 8-10 in Algorithm 1, for each source node s, we deduct the data that each node v uses to forward data from the node s. The capacity constraint in Eq. 2 is violated iff the remaining capacity of any node v after nodes forwarded data from all source nodes is smaller than zero, i.e., $c_v^{(n)} < 0$.

We will prove $c_v^{(n)} \geq 0, \forall v \in V$ by showing that

- For all $s \in [n]$, $\lambda_s \leq \frac{\sum_{i \in V} c_i^{(s-1)} + \lambda_s}{n-1}$. From lines 8-10 in Algorithm 1, we have

$$\sum_{i \in [n]} c^{(s)} = \sum_{i \in [n]} c^{(s-1)} - \lambda_s (n-2)$$
$$\Rightarrow \sum_{i \in [n]} c^{(s)} = \sum_{i \in [n]} (c_i - \lambda_i) - \sum_{i \in [s]} (\lambda_i (n-2))$$
$$(\text{Since } c_i^{(0)} = c_i - \lambda_i))$$

  Thus, for any $s \in [0..n-1]$, we have

$$\begin{aligned}\frac{\sum_{i \in V} c_i^{(s)} + \lambda_s}{n-1} &= \frac{\sum_{i \in [n]} (c_i - \lambda_i) - \sum_{i \in [s]} (\lambda_i (n-2)) + \lambda_s}{n-1} \\ &\geq \frac{(n-2) \sum_{i \in [n]} \lambda_i - (n-2) \sum_{i \in [s]} \lambda_i + \lambda_s}{n-1} \\ &= \frac{(n-2) \sum_{i \in [s..n]} \lambda_i + \lambda_s}{n-1} \geq \lambda_s.\end{aligned}$$

- For all $s \in [n]$, if $\lambda_s \leq \frac{\sum_{i \in V} c_i^{(s-1)} + \lambda_s}{n-1}$, then $c_v^{(s)} \geq 0, \forall v \in V$. We prove by induction as follows. Since $\lambda_v \leq c_v, \forall v \in V$, we have $c_v^{(0)} \geq 0, \forall v \in V$. Now, assuming $c_v^{(s-1)} \geq 0, \forall v \in V$. We consider two cases as in Algorithm 1.
  - *Case 1:* $\lambda_s \leq \frac{\sum_{i \in V \setminus \{s\}} c_i^{(s-1)}}{n-2}$. For the source node s, we have

$$\begin{aligned}c_s^{(s)} &= c_s^{(s-1)} + \lambda_s - \sum_{v \in V \setminus \{s\}} p_{sv} \\ &= c_s^{(s-1)} + \lambda_s - \sum_{v \in V \setminus \{s\}} \left(\lambda_s \frac{c_v^{(s-1)}}{\sum_{i \in V \setminus \{s\}} c_i^{(s-1)}} \right) \\ &= c_s^{(s-1)} + \lambda_s - \lambda_s = c_s^{(s-1)} \geq 0.\end{aligned}$$

    For each node $v \in V \setminus \{s\}$, we have,

$$\begin{aligned}c_v^{(s)} &= c_v^{(s-1)} - q_{sv}(n-2) \\ &= c_v^{(s-1)} - \lambda_s \frac{(n-2) c_v^{(s-1)}}{\sum_{i \in V \setminus \{s\}} c_i^{(s-1)}} \\ &\geq c_v^{(s-1)} - c_v^{(s-1)} = 0 \text{ (since } \lambda_s \leq \frac{\sum_{i \in V \setminus \{s\}} c_i^{(s-1)}}{n-2})\end{aligned}$$

  - *Case 2:*$\lambda_s > \frac{\sum_{i \in V \setminus \{s\}} c_i^{(s-1)}}{n-2}$. For the source node s, we have

$$
\begin{aligned}
c_s^{(s)} &= c_s^{(s-1)} + \lambda_s - \sum_{v \in V \setminus \{s\}} p_{sv} \\
&= c_s^{(s-1)} + \lambda_s - \sum_{v \in V \setminus \{s\}} \left(\frac{c_v^{(s-1)}}{n-2} + \lambda_s - \frac{\sum_{i \in V \setminus \{s\}} c_i^{(s-1)}}{n-2} \right) \\
&= c_s^{(s-1)} + \lambda_s - (n-1)\lambda_s + \frac{n-1}{n-2} \sum_{i \in V \setminus \{s\}} c_i^{(s-1)} \\
&= \sum_{i \in V} c_i^{(s-1)} - (n-2)\lambda_s + \frac{1}{n-2} \sum_{i \in V \setminus \{s\}} c_i^{(s-1)} \\
&\geq \frac{1}{n-2} \sum_{i \in V \setminus \{s\}} c_i^{(s-1)} \geq 0 \quad (\text{Since } \lambda_s \leq \frac{\sum_{i \in V} c_i^{(s-1)} + \lambda_s}{n-1})
\end{aligned}
$$

For each node $v \in V \setminus \{s\}$, we have,

$$
\begin{aligned}
c_v^{(s)} &= c_v^{(s-1)} - q_{sv}(n-2) \\
&= c_v^{(s-1)} - \frac{c_v^{(s-1)}}{n-2}(n-2) \\
&= c_v^{(s-1)} - c_v^{(s-1)} = 0
\end{aligned}
$$

References

1. M. Swan, *Blockchain: Blueprint for a new economy*. " O'Reilly Media, Inc.", 2015.
2. S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.
3. Ethereum, "ethereum/sharding." [Online]. Available: https://github.com/ethereum/sharding/blob/develop/docs/doc.md
4. Y. Chen and C. Bellavitis, "Blockchain disruption and decentralized finance: The rise of decentralized business models," *Journal of Business Venturing Insights*, vol. 13, p. e00151, 2020.
5. T. N. Dinh and M. T. Thai, "Ai and blockchain: A disruptive integration," *Computer*, vol. 51, no. 9, pp. 48–53, 2018.
6. K. Christidis and M. Devetsikiotis, "Blockchains and smart contracts for the internet of things," *Ieee Access*, vol. 4, pp. 2292–2303, 2016.
7. S. Huckle, R. Bhattacharya, M. White, and N. Beloff, "Internet of things, blockchain and shared economy applications," *Procedia computer science*, vol. 98, pp. 461–466, 2016.
8. X. Yue, H. Wang, D. Jin, M. Li, and W. Jiang, "Healthcare data gateways: found healthcare intelligence on blockchain with novel privacy risk control," *Journal of medical systems*, vol. 40, no. 10, p. 218, 2016.
9. A. Azaria, A. Ekblaw, T. Vieira, and A. Lippman, "Medrec: Using blockchain for medical data access and permission management," in *Open and Big Data (OBD), International Conference on*. IEEE, 2016, pp. 25–30.
10. C. Decker and R. Wattenhofer, "Information propagation in the bitcoin network," in *Peer-to-Peer Computing (P2P), 2013 IEEE Thirteenth International Conference on*. IEEE, 2013, pp. 1–10.

11. U. Klarman, S. Basu, A. Kuzmanovic, and E. G. Sirer, "bloxroute: A scalable trustless blockchain distribution network whitepaper," *IEEE Internet of Things Journal*, 2018.
12. A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun, "On the security and performance of proof of work blockchains," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 3–16.
13. R. Kumar and K. W. Ross, "Peer-assisted file distribution: The minimum distribution time," in *Hot Topics in Web Systems and Technologies, 2006. HOTWEB'06. 1st IEEE Workshop on*. IEEE, 2006, pp. 1–11.
14. M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "Splitstream: High-bandwidth multicast in cooperative environments," *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 298–313, 2003.
15. S. Liu, M. Chen, S. Sengupta, M. Chiang, J. Li, and P. A. Chou, "P2p streaming capacity under node degree bound," in *2010 IEEE 30th International Conference on Distributed Computing Systems*. IEEE, 2010, pp. 587–598.
16. P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the xor metric," in *International Workshop on Peer-to-Peer Systems*. Springer, 2002, pp. 53–65.
17. A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing*. Springer, 2001, pp. 329–350.
18. I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," *ACM SIGCOMM Computer Communication Review*, vol. 31, no. 4, pp. 149–160, 2001.
19. "Bitcoin book." [Online]. Available: https://github.com/bitcoinbook/bitcoinbook
20. E. Heilman, "How many ip addresses can a dns query return?" [Online]. Available: https://ethanheilman.tumblr.com/post/110920218915/how-many-ip-addresses-can-dns-query-return
21. "Bitcoin protocol documentation." [Online]. Available: https://en.bitcoin.it/wiki/Protocol_documentation
22. M. Essaid, S. Park, and H.-T. Ju, "Bitcoin's dynamic peer-to-peer topology," *International Journal of Network Management*, vol. 30, no. 5, p. e2106, 2020.
23. S. B. Mariem, P. Casas, M. Romiti, B. Donnet, R. Stütz, and B. Haslhofer, "All that glitters is not bitcoin–unveiling the centralized nature of the btc (ip) network," in *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2020, pp. 1–9.
24. Y. Gao, J. Shi, X. Wang, Q. Tan, C. Zhao, and Z. Yin, "Topology measurement and analysis on ethereum p2p network," in *2019 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 2019, pp. 1–7.
25. S. K. Kim, Z. Ma, S. Murali, J. Mason, A. Miller, and M. Bailey, "Measuring ethereum network peers," in *Proceedings of the Internet Measurement Conference 2018*, 2018, pp. 91–104.
26. M. Corallo, "Compact block relay," https://github.com/bitcoin/bips/blob/master/bip-0152.mediawiki.
27. E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, "Eclipse attacks on bitcoin's peer-to-peer network," in *24th {USENIX} Security Symposium ({USENIX} Security 15)*, 2015, pp. 129–144.
28. K. Nayak, S. Kumar, A. Miller, and E. Shi, "Stubborn mining: Generalizing selfish mining and combining with an eclipse attack," in *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2016, pp. 305–320.
29. I. Eyal and E. G. Sirer, "Majority is not enough: Bitcoin mining is vulnerable," in *International conference on financial cryptography and data security*. Springer, 2014, pp. 436–454.
30. K. Wüst and A. Gervais, "Ethereum eclipse attacks," ETH Zurich, Tech. Rep., 2016.
31. Y. Marcus, E. Heilman, and S. Goldberg, "Low-resource eclipse attacks on ethereum's peer-to-peer network." *IACR Cryptol. ePrint Arch.*, vol. 2018, p. 236, 2018.
32. M. Apostolaki, A. Zohar, and L. Vanbever, "Hijacking bitcoin: Routing attacks on cryptocurrencies," in *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 375–392.
33. M. Apostolaki, G. Marti, J. Müller, and L. Vanbever, "Sabre: Protecting bitcoin against routing attacks," *arXiv preprint arXiv:1808.06254*, 2018.

34. P. Ekparinya, V. Gramoli, and G. Jourjon, "Impact of man-in-the-middle attacks on ethereum," in *2018 IEEE 37th Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 2018, pp. 11–20.
35. A. Biryukov, D. Khovratovich, and I. Pustogarov, "Deanonymisation of clients in bitcoin p2p network," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, 2014, pp. 15–29.
36. T. Neudecker and H. Hartenstein, "Could network information facilitate address clustering in bitcoin?" in *International conference on financial cryptography and data security*. Springer, 2017, pp. 155–169.
37. S. Bojja Venkatakrishnan, G. Fanti, and P. Viswanath, "Dandelion: Redesigning the bitcoin network for anonymity," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 1, no. 1, p. 22, 2017.
38. G. Fanti, S. B. Venkatakrishnan, S. Bakshi, B. Denby, S. Bhargava, A. Miller, and P. Viswanath, "Dandelion++ lightweight cryptocurrency networking with formal anonymity guarantees," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 2, no. 2, pp. 1–35, 2018.
39. "Bitcoin historical data." [Online]. Available: https://tradeblock.com/bitcoin/historical/
40. G. Naumenko, G. Maxwell, P. Wuille, S. Fedorova, and I. Beschastnikh, "Bandwidth-efficient transaction relay for bitcoin," *arXiv preprint arXiv:1905.10518*, 2019.
41. R. Pass, L. Seeman, and A. Shelat, "Analysis of the blockchain protocol in asynchronous networks," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2017, pp. 643–673.
42. R. Nagayama, R. Banno, and K. Shudo, "Identifying impacts of protocol and internet development on the bitcoin network," in *2020 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 2020, pp. 1–6.
43. M. Corallo, "Bitcoin relay network."
44. "Fibre." [Online]. Available: http://bitcoinfibre.org/
45. S. Basu, I. Eyal, and E. Sirer, "Falcon," https://www.falcon-net.org/.