# An analysis of Blockchain System

Monday 18th November, 2024

## 1 Introduction

The Bitcoin system [1] is built around a public ledger that records and stores every transaction that has ever occurred. A peer-to-peer network of computers works to reach a consensus on the order and correctness of the transactions before validating and storing them in the ledger. The consensus mechanism ensures that the distributed ledger is consistent and up-to-date across the network's nodes. The Bitcoin system also makes use of a network to facilitate currency exchange. Users can send and receive payments directly from one another using this system, eliminating the need for a middleman or centralized entity. The Bitcoin system is a secure and reliable way to exchange currency in a digital world because of the Nakamoto Consensus, decentralized network, and peer-to-peer system. With the advancement of blockchain technology, blockchain networks have been used in a variety of scenarios, including decentralized finance and applications implemented with Ethereum smart contracts [2], as well as decentralized data storage achieved with Filecoin [3]. While many studies have been conducted on the network conditions of Bitcoin and Ethereum, there hasn't been much in-depth research on the Filecoin network, despite the fact that its P2P network is used by the majority of blockchains.

Filecoin [4], as the IPFS [5] incentive layer, incentivizes the authentic and effective storage of on-chain data via its token $fil$. It has assets worth 1,520 million [1] in the blockchain storage field, with BitTorrent coming in second with 721 million. Filecoin binds miners' rights to generate blocks and their actual storage capacity. It ensures that miners can provide long-term and authentic storage for on-chain data through complex and time-consuming calculations for effective data storage and fast verification. Filecoin allows miners and users to negotiate the cost of off-chain data storage and retrieval. However, for large-scale Filecoin user applications, negotiating with storage providers (SP) each time to obtain complete data can cause delays in webpage refresh and reduce user experience. The requests are then distributed to the highest-scoring nodes in that region, ensuring that the best-performing nodes are assigned to the task. This contributes to the data being retrieved quickly and accurately, as well as the network's ability to handle requests efficiently.

[1] https://coinmarketcap.com/

Filecoin network adopts Expected Consensus, a probabilistic Byzantine fault-tolerant protocol, the elected peers each epoch could submit a block consisting of the proof of election, the proof of storage, and the proof of space and time. Because Filecoin provides decentralized data storage and retrieval services to all users, there are many nodes using Filecoin services within local networks. As a result, the main distinction between the Filecoin P2P network and the Ethereum network is that the number of connections for nodes is not fixed, but is adjusted based on the node's running memory and node's performance. Before delving into the specifics of the Filecoin network, let's first discuss node initialization, including memory management and cold start issues. Nodes partition their memory by binding protocol services, limiting resource consumption in libp2p for each service or protocol. Cold start issues, on the other hand, must be considered. Cold start issues occur when a node is (re)started and the network must wait for the node to (re)connect to the blockchain's current nodes. They exchange blockchain state by sending hello protocol packets after obtaining remote node initialization information. A connection can be established only when two nodes have the same genesis block hash value. The node then confirms its own status and the height difference between the latest blocks in the hello packet and quickly synchronizes to the latest node via the graphsync protocol, which ensures that the peer can quickly retrieve and verify the data stored on the network in order to sync up with the current state of the blockchain. After successful synchronization, the node can participate in consensus and transaction processing.

In [6], a total eclipse of all connections that disrupt the IPFS network has been proposed that exploits peers' reputation mechanism and peers management strategy in Kademlia-based distributed hash table (DHT) in go-ipfs $\leq$ v0.4.23. The DHT lower buckets see more candidates (the lowest bucket could store half of the network peers), but all peers are trimmed based on the score regardless of their bucket position even the lower bucket peers are long-lived relatively with competing more candidates. In the case of a full bucket, the table evicts the peer if its last useful time is more than the grace time. The adversary occupies higher buckets easily with prepared 29TB nodes data and takes over the lower bucket gradually by tricking the peer reputation system to disconnect these honest peers. Gradually, these disconnected honest peers will be evicted from the table in the full bucket, the adversary occupies the whole table eventually and interacts with the victim periodically to keep connections stable. The later updated IPFS system introduces three main countermeasures to increase the difficulty and attack cost to occupy all table peers.

In proof-of-work blockchain (Bitcoin, Ethereum) mining is a Poisson process where it solves a puzzle with the block head's double hash less than the difficulty, therefore the mining time follows an exponential distribution. Blockchain miners blocks transactions in parallel and there are maybe some conflicting blocks at the same block height. Notice that the node just verifies the first received block with the same parent block hash, transactions sorted in the verified blocks are recorded to the distributed ledger, whereas the conflicting block's transactions will be released to the waiting pool. Hence, blocks are propagated quickly on

the network is important for saving computation electrical power and increasing mining efficiency.

Given the problem, the later updated libp2p network prefers old connections, it anchors all bucket peers by setting them irreplaceable per 60 seconds, there is no way to evict peers with consolidation if they keep online all the time. New DHT protects the front two bucket peers from trimming. The bucket 0 and 1 with overwhelming candidates, there are big chances to be replaced even just they are offline for a little while. Thus, peers who stayed in the front two buckets are mostly long-lived peers and more trustworthy to some degree.

# 2 Background

The initialization process for the node's memory and initial connections, Libp2p implements a number of protocols for establishing and managing nodes based on node status and connection performance.

## 2.1 Bitcoin Peer to Peer Network

**Connection Initialization.** Bitcoin nodes initialize connections by connecting encoded DNS seeds[7] with the limitation of connecting at most 125 nodes(115 inbound and 10 outbound nodes), the DNS seeds are maintained by the Bitcoin community members[2]. However, the reconnected peers prefer his persistent peers' records: (1) if there are no peers in its peers' table, query all DNS seeds with no delay. (2) if it has stored more than 1,000 peers, then it connects 10 peers chosen with 50% from its two tables and waits for 300 seconds. (3) in the last case, it has stored less than 1,000 peers, thus waiting for 11 seconds. The peer connection time out is 5 seconds, thus 11 seconds is enough to try to connect three peers before the timeout and maybe connect 2 peers successfully. In (2)and(3), it queries three DNS seeds once time until it gets enough outbound connections if it has only connected less than 2 peers successfully. Bitcoin node sends *GetAddr* to the neighbor for discovering more once they have built connections, the queried peer will response *Addr* including min(23% $*$ $TotolConnectedNodes$,
1000) peers and propagates it *Addr* peers package per 24 hours. If some peer responds to a package more than 1,000 peers will be marked misbehavior and punished with adding 20 scores whereas a peer will be considered disconnected if has a score of more than 100. Bitcoin system encodes 600 bootstrapping nodes[8] in case it fails to connect nodes by querying DNS seeds.

**Message propagation.** Bitcoin node floods message announcement to all connected neighbours and then responds to the full message only if their neighbour request it, all message receivers do likewise. To limit duplicated message propagation and waste bandwidth resources, Bitcoin nodes send the message

[2]seed.bitcoin.sipa.be, dnsseed.bluematt.me, dnsseed.bitcoin.dashjr.org, seed.bitcoinstats.com, seed.bitcoin.jonasschnelli.ch, seed.btc.petertodd.org, seed.bitcoin.sprovoost.nl, dnsseed.emzy.de, seed.bitcoin.wiz.biz

hash firstly as $Ihave$, then it sends the full message back following $Iwant$ from its neighbour who hasn't stored this message. The message propagation time delay in bitcoin mainly combines with announcement time and full message transmission time. For example, miners A produces a new block and then floods the hash of the block message to its all neighbours by $Ihave$, a message receiver neighbour B will request the full message if it cannot find it locally by $Iwant$, then A sends the block back. The total block propagation delay consists of three $TTL$, but it gravely reduces the resource consumption by flooding full messages directly and decreases the risk of being isolated from the network only if it has one honest neighbour. Thus the security of the Bitcoin network is mostly based on its connections.

**Network peers storage.** Bitcoin maintains a tried and new table with 256 and 1024 buckets separately, each bucket contains 64 positions. The tried table stores all connected peers, whereas the new table stores saw peers from the $ADDR$ package. The node periodically pings new table peers and translates them to the tried table if the dialled peer responds in time, refreshing the tried table and evicting trash peers in the new table. Both the tried and new table bucket filters a maximum of 8 peers with the same /16 prefix IP group, but the new table also filters a maximum of 32 peers in peer's senders /16 IP group. A new peer will be pushed into the tried table following: (1) randomly choose 8 buckets based on its /16 IP group. (2) choose a bucket based on its full IP address. (3) A ping-before-eviction replacement strategy in the full bucket. The evicted peer is pushed into the new table. Whereas the peers in the new table are marked with IP group and source IP group based on the peer who sends it back:(1) Choose 64 buckets based on the sender /16 IP group and choose the specific bucket based on the IP group itself and the bucket position is chosen by the full IP address, every peer could occur in 8 buckets increasing selection chance. (2) The unchecked peers included in $ADDR$ will replace the terrible old peers. The peer is terrible if (1)its timestamp is 30 days old or 10 minutes in advance. (2)there are 7 days from the last successfully response and it has never succeeded in response 10 times. (3)the firstly asked peer with 0 response time fails to respond 3 times.

## 2.2 Ethereum Peer to Peer Network

**Connection initialization.** Ethereum nodes initialize connections with a maximum peers size $S$ (default 50) and a limitation factor $\alpha$ (default 3) which limits the outbound and inbound peers to $\lfloor S/\alpha \rfloor$ and $S - S/\alpha$, and all outbound peers are found by encoded 8 bootstrapping nodes[9] or its persistent table. A restarted node prefers its nonempty peers' database. Generally, DHT is empty when a node restarts, it inserts 30 seeding nodes younger than 5 days randomly from DB into the DHT with filters at most 2 and 10 peers separately in bucket and table with the same /24 prefix group. No inbound peers will be pushed into the DHT until the seeding process is finished. Then, the peer connects these seeding and bootstrapping nodes, and tries to push all seen peers has passed IP filters into the bucket's front position. If it has already been pushed, its

timestamp will be updated and moved to the front. If the bucket is full, it will be tried to be pushed into the bucket replacement list. The list keeps the same IP filters and contains 10 peers in FIFO strategies. Whenever it loses outbound connections with $T = 16 - OC$, $OC$ is the outbound connections now, it connects $\lfloor \frac{T}{2} \rfloor$ peers randomly from DHT and the other half peers are chosen from DHT discovery buffer.

**Message propagation.** Ethereum has an expected time interval (13s on average) including message propagation and validation time. Ethereum nodes randomly choose the square of their connected peers, sending them full blocks and transactions directly and sending the others announcements like Bitcoin. Ethereum nodes connect 50 peers by default, which means it sends the message directly to 7 neighbours at most. This reduces 1 TTL compared to announcement firstly, however, the limited connections have more isolation risks. Thus, Ethereum nodes more rely on the highly secure peer management strategy.

**Network peers storage.** Ethereum nodes adopt a modified distributed hash table based on the *Kademlia* protocol, which contains 17 buckets, each bucket stores 16 nodes and also keeps a replacement list. Ethereum node checks the last node's aliveness in a randomly selected bucket per 10 seconds, if the selected node responded successfully, it is refreshed timestamp and moved to the bucket front, otherwise, it will be replaced with peers randomly chosen from the replacement list. The whole DHT is refreshed per 30 minutes, the refresh process consists of loadSeeds, lookupSelf and lookupRandom process. (i) loadSeeds process loads 30 seeding nodes randomly from the database and tries to insert them into the DHT in case of the table is empty. (ii) lookupSelf process discoveries new remote peers to keep buckets full. It chooses the 16 DHT closest peers by sorting the distance between DHT peers and nodes themselves. Then it sends them $FINDNODE$ and waits for their $NODE$ package. The process runs until it has queried all the front closest peers from sorted responded peers. (iii) lookupRandom process runs three lookup processes with three different random targets to refresh DHT buckets.

## 2.3 Filecoin Libp2p network

**Connection initialization.** The resource manager initializes node resources based on the node machine's available memory and connects system resources to protocols, services, and connections. The system scope manages global resources such as total (in)outbound connections and streams, available memory, and file descriptors. Protocols abound in libp2p networks, and all services are built on them; peers must negotiate specific protocols by opening a transient stream. Following the negotiation, the stream resource is removed from the transient scope and translated to the connections and protocols scopes by binding protocols services. Streams are linked to peers associated with this stream and their previously negotiated protocols, and the stream owner binds protocol services to this stream. The node tightly controls protocol-related resources, such as limiting old-version protocols and preventing services and streams from becoming overloaded. For example, consider the identity service, which sends

full identity messages containing the system version, ipfs protocols, and network state of the peer and is bound to three protocols: identity.ID, identity.IDDelta, and identity.IDPush. The identity service configures corresponding resources for each of the three protocols, and it can also limit the use of old protocols in the future through resource configuration when updating.

Each neighbour connection has its multiple communication streams based on negotiated protocols services.The negotiated stream is bi-directional, both nodes could write or read the message in a FIFO order through the stream.It also could be one-way closed, which means it only supports reading or writing for defending trash message flooding stream.

The most secure way for a node to operate is to create a trusted node list and to check and reconnect to these trusted nodes on a regular basis, ensuring that the current node can receive and broadcast network messages in a timely and efficient manner. However, because the connections' memory resources are tightly constrained, these honest nodes cannot be connected if all resources are occupied by attackers. The core of the allowlist scope shields white peers from resource constraints. During the connection negotiation process, the node checks whether the peer's IP address is protected if he fails to reserve resources for opening connections; if the peer belongs to the white peers, the connection is connected directly; otherwise, the connection is terminated.

### 2.3.1 Init peers: Exploiting the seeding peers

Filecoin networks are configured with 16 DNS servers, the node mappers the multiple IP addresses with the node identity by parsing the server and trying to connect the mapped node. The seeding peers each DHT discovery round are the second source used by Filecoin nodes to fill outbound connection slots, sorted by distance to the target each discovery round. The discovery process following $FINDPEER$ requests selects the top closest 20 peers as new seeding peers and runs until the node has connected to the top 3 seeding peers. Lotus node then boosts its outbound connections by connecting these seeding peers each round.

The key to the discovery process is the selection of seeding peers: First, the node calculates the distance to the target and gets the relevant bucket seeding peers. Second, the seeding peers are the top 20 peers from the sorted all received remote peers.

Node A sends FIND_NODE requests to each seeding node B, C, D, E. Each seeding node calculates the distance to node A and returns the bucket's peers relevant to the distance. Node A has no idea about the distance between the returned peers between himself, such as all PEERS_B have a regular distance to B and these are the closest peers in B's opinion, but actually these peers have no difference to A with any one remote peer in the network.

A (re)start peer firstly inserts all connected peers into the DHT and expands the table vertically and horizontally by querying these initial nodes twice. Note that the table expands peers for each bucket in the $fresh\_bucket$ process per 10 minutes, which means in the second table refresh process, the node elongates the table vertically through $fresh\_for\_itself$ process by querying the last bucket

peers and the newly split buckets because the front buckets are refreshed within 10 minutes. From the crawled table date, the majority of lower buckets keep peers until anchor after 75 seconds, and the last bucket peers are fulfilled after 2 minutes in the seconds DHT refresh process.

The node boosts outbound connections by connecting the seeding peers in DHT discovery multiple rounds. In the P2P network, it's crucial to ensure the security of the peer's initialization process for the re(start) nodes. The node connects bootstrapping nodes only when it hasn't connected to any peers, otherwise, it executes the $expandDHTpeeres$ process to find more peers.

### 2.3.2 Membership management

The core of the distributed system is the consistency of the network ledger synchronized between the network peers (blocks in the blockchains). A node syncs and views the network through the received propagated messages, but the validity depends on whether it has connected the majority($\geq 51\%$) adjacent connections. Therefore, the security of the P2P network corresponds to the peer's connectivity [1], the more valid peers connected, the lower the risk of being attacked with isolation or a biased network ledger. Filecoin network nodes initialize the first batch of neighbour peers by connecting the encoded bootstrapping nodes/server and finding more remote peers by later DHT discovery process. However, the table is empty when the nodes (re)start, therefore the system has to ensure the peer's initialization process execution at the beginning.

In comparison to the other p2p networks, the security of Libp2p nodes heavily depends on the peer's initialization process. Ethereum maintains an in-memory peers table, but the former nodes copy table peers who have stayed for more than 5 minutes to a persistent database on disk per 30 seconds, where a (re)started Ethereum node has enough peers backup on a persistent disk. Thus the core of the Ethereum security builds on both the database and the table.

The Bitcoin nodes store peers with two persistent tables: the tried table stores connected peers (256 * 20 == 5120 peers), and the new table contains all newly found peers (1024 * 20 = 20480 peers) from neighbours. For the (re)start and each absent outbound connection in Bitcoin, the node selects candidates randomly from these two table or connects bootstrapping servers directly if it fails to connect candidates successfully for a while.

Libp2p network uses a distributed hash table (DHT) to store peers by multiple bucket lists and find the specific peers based on the distance to the unique target DHT identity. For each node, the Filecoin system generates dissymmetric key pairs hashes the public key to a self-identified node ID afterwards hashes the node ID to a unique 256-bit DHT ID to avoid collision. The distance is the number of the leading zero of $XOR(dhtID, targetID)$ results of two DHT IDs ranging from 0 (dhtID and targetID begin with a different prefix) and 256 (dhtID equals to targetID).

Based on the network configuration, only the public IP peers could be connected directly, whereas the private peers cannot be connected due to the Network Address Translator(NAT) and/or network firewalls [10]. Filecoin system

configures two DHT modes: server and client, which is distinguished by the DHT protocol with the prefix $/fil/kad/$ where the network only allows public peers to handle the DHT protocol and server inbound peers or content discovering requests as a server mode, which achieves Pareto efficiency in the whole network.

The node inserts a new public peer into the table with six parameters $pID, LastUsefulAt, LastSuccessfulOutboundQueryAt,$
$AddedAt, dhtID, relaceable$. The $pID$ is the peer identity by hashing its public key, $dhtID$ is the DHT identity by hashing $pID$. The node records the time interval between the start and end of each request and updates the $LastUsefulAt = now$ for the valuable peers, where some peers are tagged valuable if they respond to the request within a double minimal period of responding time. The $AddedAt$ is the time when the peer is inserted into the table. The $relaceable$ marks whether the peer could be replaced in the full bucket.

**DHT peers insertion.** In the Libp2p network, all nodes are identified with a distinctive $peerID$, but one peerID could map multiple IP addresses with higher fault tolerance. However, the multiple addresses within one IP could subvert the network by Sybil attacks, thus the Libp2p network groups the node's IP of multiple addresses as follows: (1) The IPv6 address is grouped by ASN. (2) The IPv4 address is grouped by the /16 prefix. The node filters each neighbour by allowing a maximum of 2 same groups in each bucket and 3 in the whole table. Each bucket has regular positions (default 20 in the Filecoin network), and the node inserts or evicts peers by the FIFO strategy as follows: if it belongs to the legacy class 8 [3], it is grouped by the/8 prefix, otherwise, Note that the node inserts the encoded white list peers directly without any restriction. The node inserts new IP-filtered remote peers into the specific bucket list based on the distance. The DHT table is in-memory storage of the neighbour peers, but it is empty after the node (re)starts until the node has inserted some bootstrapping nodes/servers successfully. Once the table has contained some remote peers, the node could discover more peers by the DHT discovery walks, thus at the beginning, there's a heavy bandwidth consumption to keep the bootstrapping nodes in the DHT. Table peers are anchored until it has been finished executing the DHT discovery process two times, which expects to replace all firstly batched bootstrapping nodes in the table, reducing their bandwidth.

> [(1)]Find bucket list. Find the bucket position *bpos* based on the distance. The last bucket could contain much longer distance peers to decrease the empty bucket positions and save the resources consumption considering there are fewer candidates in the higher buckets. Calculate the *distance* to the target node, where the $bucketNum$ returns the table bucket number and the distance equals to minimum values of the bucket number and the real distance. Find a position in a bucket. The peers could be inserted directly if there is a spare position in the bucket[*bpos*], otherwise, goes (3). In the full bucket. **Not the last bucket.** If the bucket[*bpos*] is not the last bucket, the node will traverse the entire bucket to replace

[3] 12.0.0.0/8, 17.0.0.0/8, 19.0.0.0/8, 38.0.0.0/8, 48.0.0.0/8, 56.0.0.0/8. 73.0.0.0/8, 53.0.0.0/8.

the last replaceable peer with the newly found peer. **The last bucket.** Otherwise, goes (4). Split full bucket. The node splits the bucket while keeping the *bpos* distance peers and shifts the other peers into the newly created bucket until the new bucket has the empty position or the table has already 256 buckets. After the split process, go (1) and find the target bucket list.

**DHT finds and manages peers.** The node connects the embedded bootstrapping nodes and establishes more connections to peers found by the DHT after the node has inserted the first connected peers into the table. However, it is clear that the majority of connected peers are or received from the bootstrapping nodes, which heavily burdens these nodes with overwhelming nodes repeating the peer's initialization process at every moment. Therefore, the Libp2p network anchors the table by making all table peers irreplaceable after the node has refreshed the table two times, which excepts replacing the firstly batched table peers before the anchor.

The node records the $LastSuccessfulOutboundQuery$ time of each table peer, which symbolises the latest time of asking the nodes for peers. In the refresh process, if the time difference is more than $l \times 10$ minutes, the node will try to connect to this peer and evict it from the table if the node fails to respond to the connection request by a $dial - before - evict$ strategy. $l$ in the aliveness limitation is related to the bucket size and concurrency $\alpha$ (45 minutes in the Filecoin network with a bucket size is 20 and $\alpha$ is 10).

Both the Bitcoin and Ethereum nodes have persistent peer backup, whereas the Filecoin network just stores peers in a memory table, which is empty for a (re)start peers. The core of the peer initialization in Filecoin is the security of the bootstrapping node/servers and the initialization process runs unconditionally for a (re)start peers. Ethereum system adopts *alwaysrunseeding* and *bondwaituntilseeding*, the Ethereum node will always insert bootstrapping nodes into the table firstly and not accept any inbound connections until it has finished the initialization process[11].

Besides 9 DNS servers, the Bitcoin system encodes 851 static bootstrapping nodes in case it connects to no peers from DNS servers. For absent outbound connections, bitcoin nodes select candidates randomly from a randomly persistent table or connect seeding nodes directly if they cannot connect their candidates successfully for a while.

Both Ethereum and Filecoin networks find new remote nodes by the DHT discovery process, they first retrieve the seeding nodes and then send them $FIND_NODES$ requests, thus the peer's discovery security depends on the DHT security. Ethereum Foundation adopts [11] multiple Countermeasures, the node refuses all inbound connections before he inserts enough seeding peers from his persistent database or bootstrapping nodes/servers into the DHT, keeping all DHT peers from being occupied by attackers by a size-unlimited peers database. But Filecoin nodes have no peer storage, the only way he finds new peers at the start is to connect bootstrapping nodes/servers, then he tries to insert these seeding nodes into his DHT and finds more remote peers based on them.

Thus the Libp2p network's security relies on the security of bootstrapping nodes/serves. Most importantly, the network should keep the initialization process execution unconditionally at first and the re-execution if he can not find more peers in a long period when he has fewer neighbours left.

IPFS, Filecoin and Ethereum beacon chain are using libp2p as the network layer. Compared to beacon peers management, both IPFS and Filecoin rely on DHT to store peers without a persistent database. As for all p2p networks, bootstrapping nodes is an important security issue. IPFS team encoded 4 DNS seeds and 2 static nodes, whereas the Filecoin lotus team provides 16 DNS seeds, clearly both more dynamic/static bootstrapping nodes could effectively initialize more stable connections. In the IPFS network, the node checks his connected peers per 30 seconds and reconnects bootstrapping nodes if now he has connected less than 4 peers, whereas the lotus node reviews every 5 seconds and only does the reconnection initialization if he has no neighbours left.

#### 2.3.3 Peers reputation management

Any node could participate in or withdraw from a permissionless p2p network. Filecoin network has higher network churn[12] with 50% new connection leaving within 2 hours. However, the synchronization of the network ledger depends on the connected peers, thus it's critical to recognize nodes that are valuable or malicious. The Filecoin nodes select useful peers by contributions including propagating the new messages, retrieving peers or content, accelerating synchronization, and relaying connections. The Filecoin nodes propagate messages by a publish/subscribe(pub/sub) pattern. Subscribe is the process of binding the messages handler functions with a specific topic and publish means the node propagates the topic messages to specific peers while negotiating the subscribed topics. The message propagation scope focuses on the priority of relaying messages, the node rewards the first propagators 1 point for each unique message. However, some specific peers could inflate the number of new messages by forging overwhelming transactions. Therefore, the system limits the ceiling points to 15 for each topic message and decays 1 per 10 minutes for balancing the history contributions. For block messages, if the node has not stored these blocks in its local database and repeatedly received the same height new blocks 4 times, the node rewards the first propagator non-decaying 5 points for the contributions of providing a new chain synchronization head.

**Peers and content lookup.** The Libp2p nodes label the table peers with 5 points for the contributions of discovering remote peers and specific content. Filecoin network stores data by a directed acyclic graph (DAG) where the oversized data is divided into multiple slices and each slice is hashed to the leaf node of the DAG, all slices leaves make up the final content identity(cid). The leaf cid and the root cid are probably stored in the same node, whereas the DHT content lookup is independent and ignores the beneath relationship. Therefore, the Filecoin network develops the *bitswap* protocol to leverage the relationship between cid. For each retrieved content, the node sends $wantBlocks$ or $wantHave$ to all connected peers and batches the peers into a session that responds the

blocks or *have* back. The peers who send *too many donot have* will be evicted from the session and the node rewards peers who have inserted into the session 5 points. Note that each session is identified by a unique ID and each node could be inserted into multiple sessions with multiple points for the contributions of providing the necessary network ledger (blocks).

**Accelerate synchronization.** A node $B$ starts a new connection to the nodes $A$ by advertising its network weight and chain head blocks in the negotiated hello package. If $B$ advertises a lower weight than $A$ but with a non-genesis chain head, $A$ rewards $B$ 10 points because the Filecoin nodes prefer the synced nodes. Once $A$ discovers that $B$ has a higher weight, it will request the node $B$ for the entire chain head blocks. $A$ rewards $B$ 25 points if it responds with the valid blocks back for the contributions of assisting the node in accelerating the synchronization process.

**Relay connections.** According to the recent measurement of IPFS network [13], about 52% of peers are private peers. It's crucial to interact with them directly for the security of the whole network with the help of a relay node. The private nodes reserve connection slots to the relay node and acquire a relayed public address that is exchanged through the protocol with the prefix $/ipfs/id/push$. Thus the relay nodes deserve more rewards for the behind relayed private nodes, the attacks [6] trick the node into harming its connections by inflating the unbounded reputation of relaying overwhelming private nodes. Thus the Libp2p system disables the relaying services by default and configures a regular 5 points regardless of the number of the relayed nodes behind it.

### 2.3.4 Connections management

Libp2p nodes establish connections starting with opening a new bi-directional negotiation stream identified with a specific protocol for reserving memory resources of the distinct services. The Filecoin node adjusts runtime memory between 1GB and 4GB with scaling automatically based on the operating system's available memory. In the case of the maximum memory allocation situation, the node doubles inbound connections from 256 to 512 because it has more resources to handle more inside connections but halves outbound connections number to keep the total connections bandwidth balance. Considering the network churn, it has no limitation on the total number of inbound and outbound connections. However, the most consumer of the memory resource is the established connections, the node manages connections based on the peer's reputation and trims the lower score peers to $ConnMgrLow$ (default 150) per 10 seconds if the node has connected more than $ConnMgrHigh$ (default 180). In the Filecoin system, the lower table bucket peers and the gossipsub mesh and fanout peers and the newly connected peers within the $ConnMgrGrace$ (default 20) seconds are protected from trimming. These nodes are identified as protected to provide specific services including finding remote peers and propagating messages directly without involving the other contributions.

## 2.4 Libp2p Message propagation management

### 2.4.1 Pubsub pattern.

The publish/subscribe(pub/sub) pattern in the network layer is used to propagate specific messages to the target nodes, all messages are divided into different topics and the node subscribes to the topic means that it desires this message. During the connection negotiation, both peers exchange the subscribed topics each other with knowing which message should be propagated to this neighbour peer each other. Libp2p network supports floodsub and gossipsub. In the floodsub pattern, the node propagates the full message directly to all target peers, which has strong attack resilience and suits small networks due to the heavy redundancy. Whereas the gossipsub maintains a dynamic mesh list, the node propagates full messages to the mesh peers directly and randomly sends gossip(message digest) to the other nodes for balancing the bandwidth and attack resilience. This section introduces the message propagation pattern and mesh peers management in the gossipsub structure.

**Mesh.** The node propagates messages directly to the bi-directional mesh peers except the sender of the messages. Each *heartbeat*, the node manages the mesh peers based on the peer's contributions of relaying new messages and the connection type. At the same time, if the pruned peer's score is more than $acceptPXThreshold$, the node randomly chooses *prune* (default 16) peers who have subscribed to this topic from its connected peers for eviction compensation.

**Fanout.** The nodes could notify each other what they have not subscribed to but could participate in propagating this specific topic message by building a temporary fanout list with at most $D$ peers, the fanout will be dissolved if the node hasn't received or propagated relayed topic messages for $FanoutTTL$ (default 60s). Note that the fanout relationship is volunteer and uni-directional.

**Direct.** The node could preset a list of trustworthy white list peers (direct peers in the gossipsub) that stay outside of any mash and fanout lists, which assures to propagate its produced message even if all connections are malicious, thus the direct peers are protected from trimming and reconnected periodically. In most P2P networks, configuring trustworthy direct peers could defend against the majority of attacks, but it's difficult to differentiate who is trustworthy and this tends to form a more not decentralized network.

**Gossip.** Gossip peers are the remaining peers excluding the mesh, fanout, and direct lists that send the gossip (message unique hash) instead of the full messages to $\min\{Dlazy, Factor{*}Gossip\}$ peers three times with $Factor$ (default 0.1) and $Dlazy$ (default 6) where each gossip peer receives the message with $1-(1-0.1)^3 * 100\% = 27.1\%$ probability. When a node receives a gossip, it will request the messages through $IWant$ following the $IHave$ from the sender and decrease the sender's reputation if it fails to respond the full message back in 3 seconds. If the node misses some blocks, it requests blocks through the *bitswap* protocols and rewards the provider.

### 2.4.2 Peers reputation in the gossipsub structure

The network messages are propagated through the intricate connections between peers, but the overwhelming number of invalid messages harm the network and this malicious behaviour should be penalized. Libp2p node maintains a gossipsub peers reputation mechanism consisting of *topicScore* and *behaviorScore* which focuses on the priority of replaying new messages. Note that the weight parameters are diverse from message topics. For example, the block topic ceiling score is 50(100*5*0.1) with each block's weight being 5 and the total propagation cap being 100 and the blocks mesh with a weight of 0.1 whereas the transactions topic has a ceiling score of 5(100*0.5*0.1). The Filecoin network ledger corresponds to the blocks, thus the blocks matter more than the other messages, and the block's propagation number decays to 0 in 1 hour whereas 10 minutes of transactions.

### 2.4.3 Peers management in the gossipsub structure

The gossipsub manages peers based on the relaying contributions. Each $Heartbeat$, the node prunes negative scores mesh peers and grafts $D - Ms$ new peers to the mesh if mesh size $Ms<Dlo$ and otherwise prunes $Ms - D$ peers with keeping the front $Dscore$ (default 6) higher scores peers and randomly selecting $D - Dscore$ peers from the remaining mesh peers under the constraint that at least the mesh contains $Dout$ (default 2) outbound nodes for avoiding the mesh is occupied by all malicious inbound peers. Therefore how to monopoly the 2 outbound random peers is the core of occupying the whole mesh.

The node process the $Graft$ messages. The node inserts the node into its mesh if $Ms<Dhi$ but even in an oversized mesh, the node still inserts the outbound peers and expects to prune the redundant peers in the next *heartbeat*. Per 60 seconds, the node grafts automatically
$OpportunisticGraftPeers$ peers whose scores outweigh the median peer's score $medisco$ if $medisco<OpportunisticGraftThreshold$ (default 2.5).

## 2.5 Known attacks Strategies

Filecoin nodes interact under the decentralized Libp2p network without any central regulatory measures and institutions, which makes the Filecoin network susceptible to eclipsing attacks that view the inconsistent ledger of the network by monopolized connections.

The total eclipsing attack [11] tempts the Ethereum nodes to discover more remote peers by a non-empty DHT instead of connecting the encoded trustworthy bootstrapping nodes. The attackers establish connections to the victim node immediately after it (re)starts and non-empty its peer tables. This could destabilize all P2P networks because any nodes could participate in and withdraw from the network literately. Considering this attack in the peer initialization process, the Ethereum system refuses any inbound connection until the node has finished connecting the encoded nodes.

It's recommended that the P2P network nodes maintain a known peers backup considering the higher network churn, the eclipsing attack [14] on bitcoind v0.10.1 focus on occupying the total Bitcoin node's peers lists by establishing overwhelming inbound connections and afterwards polluting the lists by exchanging idle peers packages. For the decentralized nature, Filecoin is susceptible to the eclipsing attack on the peer's backup as a structured P2P network. [11] advertised that the structured network adopts a private DHT algorithm by inserting randomly-generated deviation in XOR operations. However, it is not implemented in any P2P system because the offset influences discovery convergence. Filecoin nodes record and discovers remote peers by a DHT based on the XOR operation distance between interacted peers' unique identities. Since the public algorithms of generating the self-authorized peer ID and calculating the specific distance, anyone could take over the partial higher table buckets of the victim node by a brute-force calculation.

Considering the above attacks, Bitcoin has increased 4 times both table size and randomly checks outbound peers without bias towards choosing refresher nodes. Updated Bitcoin node prefers the old connected peers with configurable ping-before-eviction strategies instead of being replaced directly, it also checks new table peers' connectedness periodically and refreshes the tired table by pushing the enable connected peer and evict unconnected peers immediately.

The attack [15] on the geth version v1.9 realized the totally monopoly of the outbound connections to the Ethereum node with two distinct nodes by leveraging the DHT replacement and discovery sort process. Inbound peers with the same IP address need to wait for the 30s In branch two, the attacker nodes have to ensure their returned nodes are the closest to all responded peers. proposed an attack that relies on network churn instead of reboot. Due to the public DHT distance algorithm, the attacker could generate largely prepared peers focused on each bucket with only two distinct IP addresses, one generates 10 nodes and the other one generates 7 nodes with the same IP address separately. These attacker nodes firstly occupy all bucket replacement lists, then wait for the bucket empty position. When it is inserted into the bucket, the attacker node continuously interacts with the victim and keeps itself on top. In the experiment, 4.765 days and $5 * 10^6$ peers could eventually occupy the victim's all outbound connections. Based on the attacks above, the Ethereum developers double the max peers and select nodes randomly from the bucket.

# 3 Conclusion

The Libp2p network propagates Filecoin blocks and transaction messages in a pub-sub pattern. It realizes a p2p network with an integrated plug-and-play set of protocol modules and membership management and message propagation protocols. The Libp2p network provides all participants with a consistent view of the ledger by rapidly (re)accessing the network allowing for quick message propagation and decreasing the redundancies in propagating transactions compared to the BitCoin and Ethereum.

# References


[1] Satoshi Nakamoto. A peer-to-peer electronic cash system. 2008.

[2] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.

[3] Erik Daniel and Florian Tschorsch. Ipfs and friends: A qualitative comparison of next generation peer-to-peer data networks. *IEEE Communications Surveys & Tutorials*, 24(1):31–52, 2022.

[4] Juan Benet. Filecoin: A cryptocurrency operated file storage network. *Filecoin website*, 2014.

[5] Juan Benet. Ipfs-content addressed, versioned, p2p file system. *arXiv preprint arXiv:1407.3561*, 2014.

[6] Bernd Prünster, Alexander Marsalek, and Thomas Zefferer. Total eclipse of the heart–disrupting the {InterPlanetary} file system. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 3735–3752, 2022.

[7] https://github.com/bitcoin/bitcoin/src/chainparams.cpp, 2022.

[8] https://github.com/bitcoin/bitcoin/contrib/seeds/nodes$_main.txt$, 2022.

[9] https://github.com/ethereum/go-ethereum/blob/master/params/bootnodes.go, 2022.

[10] Marten Seemann, Max Inden, and Dimitris Vyzovitis. Decentralized hole punching.

[11] Yuval Marcus, Ethan Heilman, and Sharon Goldberg. Low-resource eclipse attacks on ethereum's peer-to-peer network. *Cryptology ePrint Archive*, 2018.

[12] Dennis Trautwein. Uptime and churn. https://github.com/protocol/network-measurements/blob/master/results/rfm2-uptime-and-churn.md, 2022.

[13] Sebastian Henningsen, Martin Florian, Sebastian Rust, and Björn Scheuermann. Mapping the interplanetary filesystem. In *2020 IFIP Networking Conference (Networking)*, pages 289–297. IEEE, 2020.

[14] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. Eclipse attacks on {Bitcoin's}{peer-to-peer} network. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 129–144, 2015.

[15] Sebastian Henningsen, Daniel Teunis, Martin Florian, and Björn Scheuermann. Eclipsing ethereum peers with false friends. *arXiv preprint arXiv:1908.10141*, 2019.