FavorX Protocol

The Unified Hybrid Data Transport Network For Web3

Abstract

Web3 is in a rapid development process, and the business value brought by decentralized technology is continuously being discovered and recognized by the industry, while the DEFI and NFT markets on Ethereum has also proved the value of decentralized finance. The emergence of GameFI, Web3, the concept of Metaverse, and corresponding prototype products indicate that decentralized technology is gradually expanding to the mass market. When decentralization enters the mass market, the decentralized data transmission system will become an important infrastructure. In early blockchain projects such as public blockchains, in order to ensure the censorship resistance of transactions, the only transaction data was transmitted in a network-wide broadcasting manner. In Web3 systems, clients not only interact with the public chains, but also need to interact with various decentralized application service backends, such as storage backend and database backend. The BitTorrent network, IPFS and Swarm/BZZ have already made very useful explorations and laid the foundation for the decentralized uniform communication protocol - FavorX Protocol .

FavorX Protocol is a complete decentralized data transmission protocol that turns the CDN into a decentralized trading marketplace, in which tokens (FavDs) are circulated. Miners provide bandwidth resources to obtain token revenues, while applications pay tokens to use the data relay services provided by miners. When miners provide data forwarding services, they will get corresponding token rewards according to the ratio of the bandwidth computing power they provide across the whole network to the total bandwidth computing power of the whole network, which is similar to the block height reward mechanism of Bitcoin/Ethereum. Therefore, in the FavorX network, miners derive their revenues from two sources: the data traffic revenues generated by providing data relays and the token rewards generated by the bandwidth computing power.

The FavorX Protocol brings decentralized blockchain capabilities to mobile terminals, and the FavorX infrastructure and applications are separated. The nodes provided by miners have routing, data and message relay functions, and are rarely offline after going online. These nodes are called full nodes in the network, constituting the basic decentralized network. Meanwhile, the mobile node is always connected to the full nodes, sending requests or obtaining data through the full nodes, and never providing data and message relay capabilities to the network. This ability is suitable for the current mobile terminal that will go online and offline repeatedly, so that the system will not be unstable due to too many mobile terminals.

FavorX Protocol is a decentralized data communication network, which implements different processing modes and accounting models according to different data types: 1. To improve efficiency while transferring file fragments, a local data cache can be established to enable repeated access to that file in a short time, with the cost of file transmission being charged on a per-byte basis. 2. In the transmission of command requests/responses, the charges are based on the number of frames and the total length. 3. When messages are transmitted, the charges will be assessed on the number of messages and the total number. 4. Furthermore, the protocol will allow the application to customize and use a specific accounting scheme. The flexible accounting scheme allows for a better balance between miners' revenues and application costs.

To meet the high-performance and low-cost on-chain computing requirements for accounting, FavorX Protocol adopts an off-chain accounting and on-chain settlement scheme: Upon receiving a response, the data requester issues a receipt to the service provider, and the data service provider collects, merges and Reduces the receipts of the same requester. Every once in a while, the Reduced receipts of each requester are merged again to form a receipt set and submitted to the chain for settlement. To meet this settlement requirement, FavorX Protocol has designed a dedicated chain for processing this dataset, which can operate on its own or as the Layer2 on an existing public chain (such as BSC/Polygon) for higher security.

FavorX's work includes:

1) A routing system over P2P adds multicast and unicast capabilities to message and command relaying on the basis of P2P network-wide broadcasting
2) Optimizing the performance of DHT content distribution to meet the real-time playback capability of streaming media under high concurrency
3) Defining application-level protocols according to different types of data, including content distribution, command relaying, message subscription and real-time streaming, which can meet the data transmission requirements of all current Web3 applications
4) Accounting for content distribution and message/command relaying, calculating the bandwidth computing power, and then rewarding it
5) A dedicated chain of FavorX Protocol, the performance and cost of which can meet the requirements of decentralized accounting, rewarding, and operational information

1. Introduction

P2P networks are the underlying communication frameworks of current decentralized systems, and currently there are mainly two kinds of data transmission approaches in P2P networks: broadcast-based data communication schemes and distributed hash table (DHT)-based data transmission schemes. In the system based on public chains, since only transactions and block contents need to be transmitted on the network, and each node needs to verify all the transactions and block contents, broadcasting communication is used to transmit information. In decentralized file transmission systems, such as IPFS and SWARM,

since the goal is to transmit a large amount of content to the target endpoint, it is only necessary to relay and transmit content at the necessary nodes, and it is not necessary or should not let all nodes read the content, so DHT-based data transmission schemes are adopted.

With the booming development of industries and blockchain projects, Cosmos and Polka, as representative products of cross-chain solutions, focus on providing data transit services between different chains or projects, and different P2P networks, in order to satisfy the communication needs among different projects via a relay chain. On the other technical route, attempts are made to provide multiple functionalities such as unicast and multicast of messages by processing communication protocols, to realize the goal of integrating different projects into the same P2P network. FavorX Protocol, belonging to the latter solution, adds routing tables to each node based on the DHT routing scheme, which records the recent paths to the target node. In addition, it also introduces the concept of virtual connection, which enables two physically non-adjacent nodes to form a connection logically. The introduction of the concept of a group enables virtual connections to be formed among nodes within the group, enabling broadcast of messages within the group, while also allowing external nodes to send requests to the group and receive responses.

FavorX Protocol is a communication protocol running on a decentralized network, based on routing tables and multicast protocols, which can meet the needs of different projects. It can group data that needs to be broadcasted (such as public blockchains) into a group and broadcast within the group; for situations where large amounts of data need to be transmitted to the target node, the routing table can be used to find the fastest transmission path; and for data that may be read repeatedly, local caching is used to reduce the possibility of repeated reading.

The FavorX Protocol comprises the following components:

1) A decentralized routing table, implemented on top of a DHT network for virtual connection.
2) Unicast and multicast on P2P, based on the decentralized routing table for message orientation and limited range transmission.
3) The foundation layer, driven by a built-in special-purpose chain that can be accessed by applications via multicasting:
    a) Custom High-performance Low-cost Blockchain
    b) Name Service Protocol
    c) Authorization Access Protocol
    d) Accouting and Incentive System
4) The application-layer protocols, including:
    a) File Transfer Protocol
    b) Live Streaming Protocol
    c) Web Service Protocol

2. Relevant Works

This section has reviewed the existing important technological implementations used by FavorX. It has focused on describing IPFS and SWARM, two globally distributed storage systems.

DHT

Distributed Hash Table (DHT) is a widely used device and node addressing scheme in P2P networks, which adopts the architecture of no centralized server to divide the whole network into multiple subnets. Each node is responsible for part of the routing and storage, thus implementing the addressing and storage of the whole DHT network.

Distributed Hash Table (DHT) is a decentralized distributed system composed of a keyspace and a mask network, which has the characteristics of automatic decentralization, powerful fault tolerance, and good scalability support. DHT does not specify a specific algorithm implementation, so there are various implementation plans, such as Chord, Pastry, Kademlia, etc. Kademlia and its variants have become the mainstream scheme for DHT implementation due to their excellent performance and implementation schemes.

KAD

Kademlia is a typical structured P2P overlay network, which organizes nodes and data in the same way and implements the distributed storage and query of data through the XOR calculation of logical distance. It distributes all nodes into 160 buckets based on logical distance, and can quickly find the nearest transmission or query node according to the data key without caring about which node the value information is stored on.

S-BUCKET

In KAD network, 160 buckets are used to store all the node information of the network, and the number of nodes in each bucket decreases exponentially. When only the adjacent nodes are recorded, a large number of empty buckets will appear, thus increasing the difficulty of routing and the probability of failure.

In order to solve this problem, SWARM proposed the concept of "saturation": setting a threshold K, starting from the first bucket (saturation 0).

1. If the number of nodes in the bucket is greater than or equal to K, S is the serial number of the bucket, until:

2. If the total number of nodes in the bucket is less than K, S remains unchanged, ending; or

3. If the total number of nodes in all buckets greater than S is less than K, S is reduced by 1 (if S is not 0), ending.

The S-BUCKET algorithm divides the number of connections of the node into saturated region and unsaturated region, with the bucket numbered as S as the boundary, each bucket in the saturated region is treated as a virtual bucket, and the unsaturated region is merged into a virtual bucket; during the addressing process, according to the addressing target, the corresponding virtual bucket is searched to find the corresponding neighbor node to forward the data or query the relay. In FavorX, a maximum of 16 buckets are used.

2.1. SWARM

SWARM SWARM is a distributed storage project on Ethereum, whose design goal is to provide a decentralized storage system, which adopts KAD addressing scheme and S-BUCKET scheme to achieve low failure rate and high read speed.

2.1.1.Responsibility Zone

If the nearest node is the sole storage node and has gone offline, the content will be irrecoverable, and this basic scenario can be addressed by implementing redundant storage. SWARM designed the concept of "responsibility zone" which defines it in fully connected areas. If the block falls within the responsibility zone of the node, the memory node will assume the responsibility of storing it. As long as each node in the minimal field of R endpoints can store all the blocks in the area, the content can be mathematically guaranteed to be retrievable even if R-1 storage nodes are disconnected at the same time.

2.1.2. Streaming Synchronization

SWARM employs data dissemination and stream synchronization, which automatically propagates data according to its key value to the nearest node and establishes an input stream and an output stream between the node and all the nodes in the bucket. It also tags the data with sequence numbers in the cache to avoid data being propagated multiple times. Then:

1. When the data in the cache is updated, the node updates its output stream with a new sequence number (NC);
2. Nodes need to record the latest updated sequence number Ni of all input streams;
3. When a new sequence number NC is received from the input stream, search for all the sequence numbers between [Ni, ..., NC], and request the transmission of the corresponding data for these sequence numbers from the input stream;
4. When receiving the data corresponding to the sequence number, update the value of Ni.

Streaming synchronization can quickly and non-repeatedly synchronize data between two nodes, but in a decentralized system, there are security issues: any node may attack by writing invalid junk data, resulting in wasting precious network bandwidth and storage space.

Another disadvantage of SWARM is that it can design arbitrary key values for the data value. This scheme can store different values for the same key, which is an important foundation for

the PSS module of SWARM, but in a decentralized network, malicious nodes can arbitrarily modify the value, resulting in the inability to effectively guarantee the authenticity of the data.

2.2. IPFS

IPFS is a content-addressed, distributed hypermedia transport protocol aimed at building a fast, secure, and open global file system. It utilizes a DAG graph structure like GIT to organize folders and create a decentralized file system, which has attracted widespread global attention.

2.2.1. Coral DSHT

In SWARM, the data values are directly synchronized and stored to Distributed Hash Table (DHT) nodes, which can lead to unnecessary node storage and transmission of data and thus waste of storage space and network bandwidth. IPFS adopts a new approach of first evaluating the connectivity of all nodes and then dividing them into several different priority levels according to the Round-Trip Time.

Coral DSHT is a distributed M-Hash Table used to implement soft-state key-value retrieval, allowing multiple values to be stored for the same key and mapping the given key to Coral server address, which can be used to query domain servers closer to the user, servers with specific website caching information, and locating nearby nodes to minimize request latency.

2.2.2. GIT

Git utilizes a directed acyclic graph to store file contents, with each file's history associated to its directory structure and root node and ultimately to a commit node which may have one or more parent nodes. This data structure has the characteristics of facilitating the browsing of file history and contents:

i. When content nodes (files or directories) have the same identifier (in Git represented by a SHA code) in the directed acyclic graph, even if they are in different commit nodes, it guarantees that their contents are the same, thus making Git more efficient in difference comparison.

Ii. When merging two branches, essentially what is being done is merging two directed acyclic graph nodes. Directed acyclic graph allows for Git to efficiently determine their common parent node.

Git provides a scheme of using directed acyclic graph to store folders in objects, which minimizes the object storing changes when the content of the folder is changed.

2.2.3. BITSWAP

The BitSwap protocol in IPFS is inspired by Bit Torrent, distributing data by exchanging data blocks between peer nodes. Like Bit Torrent, each peer node continuously uploads the data

already downloaded while downloading. Unlike the Bit Torrent protocol, however, BitSwap is not limited to data blocks in a single torrent file. That is, downloading a large file or directory (divided into multiple small blobs and lists for storage), one of which is part of other files. The BitSwap protocol contains a permanent marketplace composed of all the nodes. This marketplace contains all the blocks wanted by each node, including all the block data that each node wants to retrieve. Regardless of what these blocks are (e.g. part of a torrent file), the block data may come from files that are completely unrelated in the file system.

Every node in IPFS maintains two lists: the list of blocks it already has (have_list) and the list of blocks it wants (want_list). The BITSWAP credit mechanism is used to prevent water holing attacks (empty-load nodes never share blocks). Node downloads and share files should be balanced, and the bytes received and sent are recorded in the BITSWAP ledger. When A and B are connected, A sends the part of the local ledger related to B first, that is, the record of the blocks exchanged between A and B before. If the same, a link will be established; if different, node B will send an empty ledger to node A. If a node's ledger is found to be different often, the other nodes will remember it and not link to it in the future. Moreover, after the link is established, if the blocks sent by B to A are found to not be the ones desired, the connection will be broken and the ledger will not be updated.

2.2.4. CID Content Addressing

Content Identifier (CID) is also referred to as a tag in the IPFS network that points to the actual data entity. It does not represent the actual location where the data is stored in the network but rather a form of address representation based on the data itself. No matter how large the data content is, the representation of CID is very short.

CID is a content-based encrypted hash. It has two main features:

1) Any modification to the data will produce a different CID.

2) Appending the same content to two IPFS nodes in the same manner will generate the same CID.

IPFS by default employs the sha-256 hashing algorithm, while also supporting a variety of other algorithms. The Multihash project aims to provide other programs with hashing algorithms, and allows for the coexistence of multiple hashing algorithms.

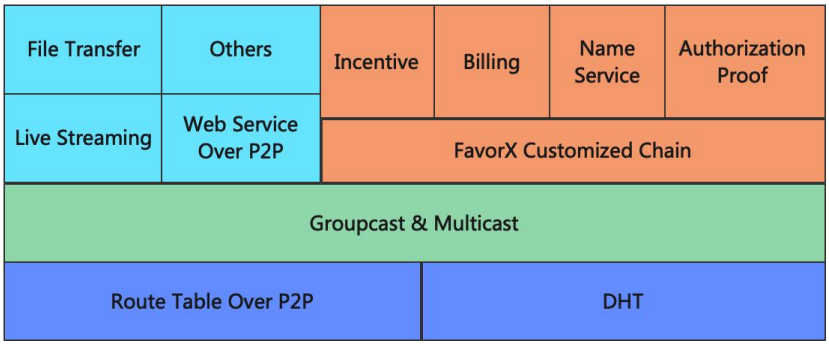
3. Design of FavorX System

The design of FavorX is aimed at solving the network isolation problem in Web3 systems by restricting the transmission of messages to necessary nodes on a peer-to-peer network and transmitting various protocols through this message, thus providing decentralized services for multiple underlying modules for Web3 applications.

In mobile networks, node online/offline behavior is affected by device startup/shutdown and application foreground/background state. When the device is on and the application is in the foreground, the node will join the decentralized network; when the device is off or the application enters the background, it will automatically disconnect the network connection of the application, resulting in the node going offline. If a large number of mobile nodes are used to provide routing functions in the system, it will result in a large number of invalid routes, thus degrading the network performance and causing network jitter. The same mobile devices are also limited by the battery, so even if they are online, the nodes do not want to keep providing uplink bandwidth in order to avoid extra power consumption. In FavorX, nodes are divided into full nodes and light nodes, where full nodes are equipped with the functions of routing and data transmission, while light nodes only connect with full nodes and read data from them.

All full nodes in the FavorX network are placed in an equal position without any privilege. Every node offers data caching and command forwarding services to other nodes, and thus is rewarded with corresponding rewards and FavorX network computing power, and thus gets corresponding rewards based on its network computing power.

3.1. Protocol Architecture



An agreement architecture is adopted on the basis of existing DHT addressing, which enables each full node to create and maintain its own routing table to achieve high performance and reliability of message and data relay. Based on this routing table, unicast and multicast messaging functions are implemented, allowing lightweight nodes to send requests to the group and get a response only with the knowledge of the group information.

The introduction of multicasting capabilities enables the FavorX Protocol to create different functional modules or connect to existing blockchain modules. For example, within the FavorX Protocol, the light nodes can send requests or subscribe to messages from the 19-node-formed FavorX Customized Chain to obtain chain information. Additionally, applications can also create a listening group for existing blockchains to relay the blockchain messages heard to subscribers. For example, a listening group of nine nodes can be used to monitor information on Ethereum, while a light node can simply subscribe to the group to get information from Ethereum or send commands to it through the group.

On FavorX's customized chain, a decentralized trading market is established, with all nodes providing message transmission and data relay services to the system, for which requesters need to pay. On FavorX network, these fees are always measured in the form of data traffic fees. When the full nodes of FavorX transmit verifiable data traffic, they will gain network power and receive corresponding rewards through the Incentive and Billing modules according to the proportion of the node to the total network power.

FavorX's customized chain also has a name service system and authorization proof system, where the name service system can realize the mapping of general URL to specific information related to the service, such as the CID of the file when the file is transferred, and the group name of the corresponding service for the decentralized service.

In addition to the above protocol components as the foundation of FavorX, FavorX has also defined most of the basic technical modules required by Web3 applications, including file transfer, real-time media streaming and Web service protocols on P2P networks.

FavorX adopts the strategy of dividing and segmenting files in units of 256KB for file transmission, and each full node has data caching capability, so the same file fragment may have multiple data sources. FavorX defines how to select the most suitable one from different data sources for data transmission, which optimizes the strategy and enables FavorX to achieve high-definition video real-time playback capability with high concurrency users and greatly improves the efficiency of file transmission.

Unlike data transfer protocols in which data is read from the client side, real-time streaming media protocols have the source node actively pushing data to all subscribers. Due to the limited bandwidth of the source node, it is necessary to assist in forwarding the data by using a relay node, that is, the data is first sent from the source node to multiple relay nodes, and then these relay nodes deliver it to the subscribers. FavorX's real-time media streaming protocol supports dynamic adjustment of the relay nodes according to the number of subscribers to optimize network bandwidth, providing subscribers with a continuous and uninterrupted playback experience.

The basic design concept of Web services in existing Web systems is to build a platform-independent, low-coupled, self-contained, programmable Web application, usually adopting the Server/Client architecture, in which the Client accesses the interface provided by the Server via the HTTP protocol. In order to be as compatible as possible with existing Web development, FavorX provides a function of proxying HTTP services through P2P networks.

FavorX's basic P2P network is implemented based on LibP2P, thus fully taking advantage of its modular features, while addressing and data flow formats conform to the conventions of Lib2P2P: different transport layer protocols are identified by different ProtocolIds, while data is encoded by ProtocolBuf.

3.2. Nodes and Groups

3.2.1.Nodes and Identities

In FavorX, each node is endowed with an address and an identity that is created through the use of S/Kademlia's static cryptographic puzzles involving public-key cryptography hashes, wherein the node stores its public and private keys (encrypted by cryptography) and its overlay address is generated by hashing the public key with sha256. Additionally, each node possesses a type identifier to indicate whether it is a full node or a light node in the system. When nodes establish a connection, they need to exchange public keys, and check if sha256(peer.publicKey) is consistent with peer.NodeId; if not, the connection will be immediately terminated.

The connection strategy and type of the node are related, light nodes always only connect to full nodes and not to other light nodes, and full nodes will also take different strategies to manage other nodes that connect to it according to the node type.

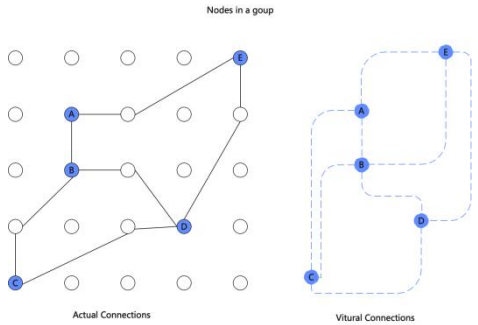
Connection Strategies:

Connected or Not	Full node	Light node
Full node	Y	Y
Light node	Y	N

When all nodes receive a connection request, different handling approaches will be taken according to the type of the incoming node: if it is a full node, it will be managed by S-Bucket; if it is a light node, it will be managed according to the maximum number of allowed connected light nodes.

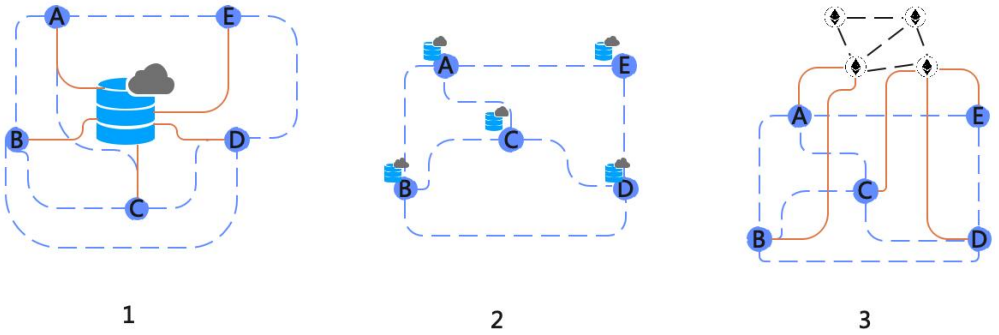
3.2.2.Grouping

Any nodes in FavorX can create groups, which consist of a set of logically-related nodes connected through virtual links. In most cases, data will be synchronized within the group, and if the way of adding nodes within the group has the characteristics of decentralization, then the group is decentralized.



One of the basic components of FavorX is grouping, which can:

1) Encapsulate a single point of service as a group, enabling clients to access the single point of service in a decentralized manner for the network.
2) Decentralize existing services by deploying the same service - e.g., database service, application service - on multiple nodes and implementing data synchronization across multiple nodes using intra-group broadcast functionality.
3) It is also possible to connect a decentralized technology module such as a public chain or decentralized storage to a set of nodes, providing customers with another connection option in addition to RPC connections.



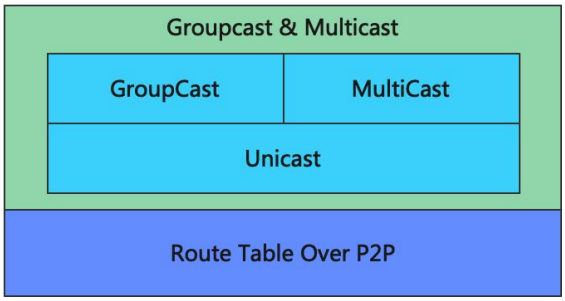
The communication scheme within the group is defined as multicast and groupcast. Multicast is to send a message that needs to be broadcast within the group to multiple nodes in the group; while groupcast is to send a command to the group to obtain a response, which only needs to be processed and responded by a node in the group, without broadcasting in the group. Evidently, the primary application scenario of multicast is data synchronization among nodes within a group, while that of broadcast is interaction between clients and services within a group.

Due to the non-trusting nature of nodes, when a client requests data from a server, it needs to check the validity of the acquired data. If the validity of the data is to be determined jointly by the server and the client, it is not specified in FavorX.

3.3. Transport Layer Protocol

As mentioned above, in addition to having the ability to broadcast messages to the network, FavorX also defines multicast and unicast: Multicast is to broadcast messages within the same multicast group, since the nodes belonging to the same multicast group may not necessarily be adjacent nodes, the broadcast messages in the multicast group may need to be relayed through other nodes. The Unicast feature is the ability to send messages from one node to a specific node, which may also require relaying through other nodes. Aside from the two aforementioned transmission protocols, there is also a Multicast protocol: namely, sending commands to or getting data from a group, in fact, this Multicast protocol is realized through Unicast protocol, that is to say, the requester firstly needs to find a node in the group,

then send the request to the node and get the response. Unicast capability relies on the implementation of RouteTable. Therefore, the underlying framework of the transport layer protocol is illustrated in the following diagram:



3.3.1. Protocol Formats

In FavorX, the protocol format is always "ProtocolId/protocol data", and the following ProtocolIds are currently defined in FavorX:

- Routing Protocol
- Unicast Protocol
- Multicast Protocol

Where, protocol is a five-tuple (D,T,L,P,S)

Thereinto:

1) D → Dest NodeID, the destination address to send to
2) T → TTL, each hop is decremented by 1, and no more forwarding when it reaches 0, preventing infinite loopback of data
3) P → Payload, application layer data protocol
4) S → Signature, the signature of the requester

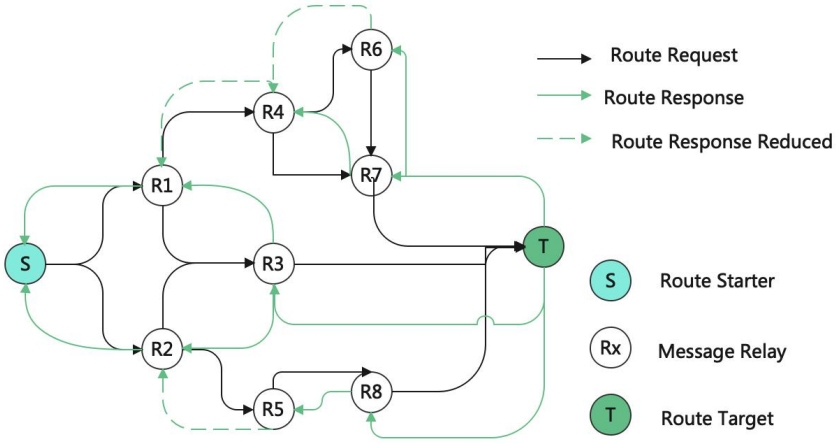
All data content is encoded with ProtocolBuf, and since the transport layer does not define a generic protocol format for the payload, the protocol handler encodes and decodes the content in the payload.

3.3.2. Routing Protocol

Brief Description

A routing protocol is a technique for discovering the next hop address to a particular destination on a network, with the key being the ability to record the latest connection information that can reach the destination address. When data is successfully transmitted through the nodes of a routing table entry, the routing table entry should be refreshed; and when a routing table entry has not been used for a period of time, the entry will become invalid. Therefore, through the routing table, the application end can establish a virtual connection. When a node needs to establish a connection with another node, it can send heartbeat data regularly within the timeout period of the routing table item to maintain the validity of the item.

The routing protocol is divided into routing discovery and response. The following diagram is a schematic diagram of the routing discovery process.



The figure shows an operation process based on a route discovery protocol, where node S sends out a Route Request R_{req} and searches for the closest two nodes R1 and R2 among its neighbor nodes, then sends the R_{req} to the two nodes, and so on until the last step which pushes the R_{req} to the destination node T.

When the destination node receives the Route Request R_{req}, if it finds the destination node is itself, it will generate a Route Response R_{resp} and push it back to the node that sends out R_{req}. In the above figure, R7 and R8 are the nodes that issue R_{req}, and thus will receive R_{resp}. The same procedure is repeated until the final step of pushing R_{resp} back to the initial node S.

During the process of pushing R_{req} and R_{resp}， all the intermediate nodes will update their routing tables according to R_{req} and R_{resp} to reduce redundant route entries. When there are multiple routes to the destination address, the intermediate nodes will select one or more optimal routes and delete other redundant route entries. Generally speaking, the valid route with the least number of hops is usually retained. Taking R1 to T as an example, there are two routes for the routing, 1) R1→R4→R7→T, 2)R1→R3→T. If we keep only one route, then route 1) will be deleted while route 2) will be preserved.

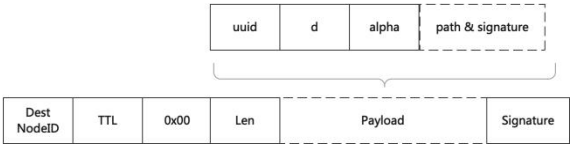
3.3.2.1.Routing Discovery

Definition

Routing discovery R_{req} is a quintuple created by a node S with ($uuid_{t,s}$,d,α,p,s), where:

5) uuid → creation time and the creator's signature for that time, this can be used as a unique identifier for route discovery and response
6) d → The overlay address of the target
7) α→ The fan-out value, i.e., in search of several neighboring nodes to forward this request
8) p→ The path information during the process from the start node to this node, each path is signed by the corresponding node, used to verify whether the path is valid, this information can be used to update the local route

9) s → The signature of the node in the path to verify that the frame has not been forged or tampered with, in this case the value is signed by the initiator and the path is empty.



Initiate

When a node needs to find the route to a specific node, it generates the frame and requests for relay from its neighboring node, and the frame will be modified continuously in the process of relay.

Receiver Processing

Algorithm 1 Procedure on RReq Frame.

Validate $R_{req}.s$;
Check $R_{req}.s > 0$
Check $R_{req} \notin pend\_list$
Update Route Table by $R_{req}.s > 0$
if $R_{req}.s == self.addr$ **then**
  Create R_{resp};
  Send R_{resp} to source node;
else
  $R_{req}.ttl \Leftarrow R_{req}.ttl - 1$;
  Update $R_{req}.ttl$;
  Generate $R_{req}.s$;
  $pend\_list \leftarrow pend\_list \cup R_{req}$;
  Send R_{req} to max $R_{req}.\alpha$ neighbours ;
end if

When the node receives the route discovery request, it first checks the validity of the request, whether it has been repeated, and then checks whether the target node is itself. If it is itself, it will generate the route response R_{resp} and return it to the other party. Otherwise, modify the request as necessary and search for a suitable node from the neighboring nodes to forward this modified routing request.

Cost Calculation

Algorithm 2 Payment of Route Request

D ←Max(dist(self.addr,$R_{req}.d$), $R_{Req}.ttl$)
P ←$P_{base} * R_{Req}.\alpha^{D}$
return P

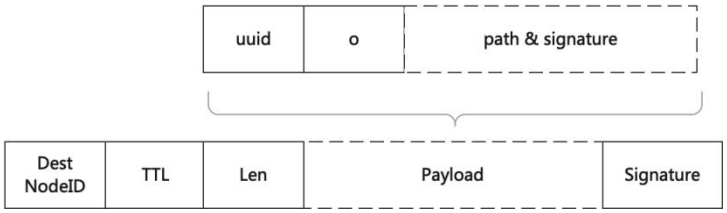
The requester needs to pay the forwarding node a fee proportional to $R_{req}.\alpha$ and the distance between this node and the target node, where $R_{req}.\alpha$ is a variable factor. As $R_{req}.\alpha$ increases or the distance between this node and the target node increases, the fee to be paid will also increase.

3.3.2.2.Route Response

Definition

Route Response frame R_{resp} is an four-tuple ($uuid_{t,s}$,o,p,s)created by the destination node D when receiving a route discovery request, where:

10) $uuid_{t,s}$ →the uuid of this R_{resp} corresponding to the R_{req}, the identifier of route discovery response
11) o→ This R_{resp} corresponds to the overlay address of the R_{req} initiator
12) p→ The path information during the process from the response node to this node, each path is signed by the corresponding node, used to verify whether the path is valid, this information can be used to update the local route
13) s → The signature of the node in the path to verify that the frame has not been forged or tampered with



Initiate

When a node receives a route request with its own node as the destination node, it generates the frame and returns it to the requester.

Receiver Processing

Algorithm 3 Process R_{resp}

Input: R_{resp}
  Validate R_{resp}
  if $R_{resp}.uuid \in pend\_list$ **then**
    modify $R_{resp}.p$
    modify $R_{resp}.ttl$
    Generate signature for R_{resp}
    Send R_{resp} to max $R_{resp}.\alpha$ neighbours ;
  else
    Ignore this response
  end if

Cost Calculation

None, processing route response frames does not incur any costs, however if the route is successful, the node may benefit from relaying data by collecting data flow fees, which is the potential gain of processing route response frames.

3.3.2.3.Fault-tolerant Handling

Routing Failure

In S-Bucket based routing discovery process, if the fan-out number [1] of each node is N, and the number of nodes in the whole network is M, the average hops of the whole network is $log_N M$; Considering the success rate of TCP transmission and the success rate of route response, the probability of routing discovery and successfully getting response is p^{2H}. In order to improve the success rate, a way of forwarding each request to multiple next-hop nodes with the number of α can be adopted, and the success rate of each hop TCP transmission is:

$$p' = (1 - (1 - p)^{\alpha}$$

Thus, the probability of discovering the route and successfully obtaining a response will increase, i.e. p'^{2H}.

In a typical network with 10 million nodes, N=20 and H=6, considering the bandwidth occupancy of application systems may lead to point-to-point communication timeout or failure, p=0.999, when α=1, the transmission success rate is

$$p = 0.999^{2\times5} = 98.8\%$$

and this failure rate is relatively high, which will have an adverse effect on applications. When α = 2, the transmission success rate is

$$p = 0.999999^{2\times5} = 99.9988\%$$

which can meet the requirements of the actual application.

Amplification and Routing Spoofing

FavorX is designed for a trustless network where routing requests originating from source nodes are forwarded by multiple nodes in the form of α^{ttl}, making malicious nodes capable of attacking the network by using large α and ttl values. To prevent such attacks, FavorX requires the nodes issuing the routing requests to pay the corresponding fees to the other nodes. This fee is paid by the initiator of the route to the forwarding neighbors and, in turn, by the forwarding neighbors to the forwarding nodes when they forward again.

When a neighboring node receives a routing discovery request and fee, in order to obtain genuine routing information, he will need to pay a corresponding fee to his forwarding neighbor node, which will lead to the neighboring node having a motivation to not forward the request normally, but to generate a false routing response to the requester, so the routing response needs to be verifiable. FavorX adopts signature verification to realize path authentication, that is, each hop node will sign its

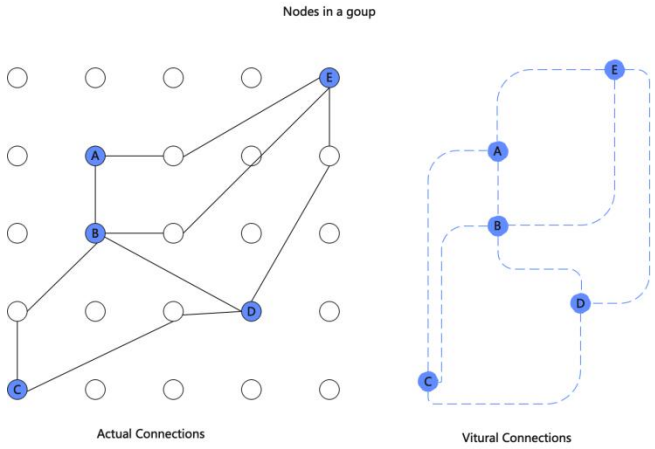
[1] refers to the subordinate full nodes connected by a node; in P2P networks, this value is 1/2 of the node's average full node connections

path. This signature scheme prevents nodes from forging routing responses while increasing routing complexity.

3.3.3. Transport Protocols

3.3.3.1. Unicast Protocol

Unicast protocol is a point-to-point communication protocol based on routing, which establishes a virtual connection between two points, and uses routing protocol to find the optimal route between the source node and the destination node, thus realizing efficient and reliable data transmission. In the process of system implementation, communication status of the virtual connection can be recorded. When the virtual connection is in idle state for a long time, heartbeat frames can be sent to maintain the routing table entries in each node on the path, thus effectively ensuring the success of subsequent data transmission.



In certain scenarios, nodes can send unicast messages without routing, i.e. after node A sends a request to node B, node B can directly respond without route discovery, because the routing table entries from A to B are already up-to-date during the request process.
All application layer command and message protocols, including multicast and broadcast protocols, are based on the unicast protocol, which contains the payload of the unicast protocol. The data frame format of the unicast protocol is shown in the figure below:

Definition

Unicast discovery is a datagram with ProtocolId "/favx/1x", where x is an optional range of "0-9" and "A-F". The payload can be any generic data format and the following:

1) 10 is the heartbeat frame, and the rest is defined by the application
2) The data fee of unicast is always paid by the source node

Initiate

When a node needs to send a multicast data Payload to another node, a unicast data frame is created and sent to the corresponding neighbor node.

Algorithm 4 Process Unicast frame

Input: $unicastframefr$
  Validate fr
  Check $fr \notin known\_frames$
  if $fr.d == self.addr$ **then**
    if $fr.c == HEART\_BEAT$ **then**
      Ignore
    else
      $sink \leftarrow GetSinkFromCmd(fr.c)$
      $sink.process(fr)$
    end if
  else
    $fr.ttl \leftarrow fr.ttl - 1$
    $known\_frames \leftarrow known \cup hash(fr)$
    $next\_hops \leftarrow GetRouteItems(fr.d)$
    Send fr to next_hops
  end if

When the node receives the unicast data frame, it checks the validity of the frame and then gets the next hop neighbor node from the routing table according to the destination node address, and then sends it to the neighbor node.

In order to prevent data loopback, for each data frame, the hash will be recorded in a *known_frames*, if a data frame has been recorded, the node will receive it and directly discard the frame. When implementing the system, the total capacity of *know_frames* can be restricted, and only the data frames of a certain period of time will be recorded, instead of being permanently recorded, thus avoiding the capacity of *know_frames* expanding infinitely.

For the data sent to itself, it should be checked if it is a heartbeat. If it is a heartbeat, which is only used by the other side to maintain the connection, it can be simply discarded; otherwise, the data should be handed over to the upper layer for processing.

Cost Calculation

Algorithm 5 Payment of Unicast

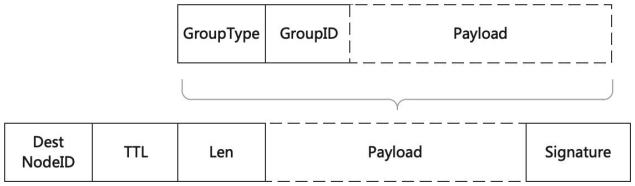
Input: $Unicast frame fr$
  if $fr.d \neq self.addr$ **then**
    $credit[src\_node] \leftarrow credit[src\_node] + P_{base} * fr.ttl$
    $debit[next\_hop] \leftarrow debit[next\_hop] + P_{base} * (fr.ttl - 1)$
  else
    if $fr.c = HEARTBEAT$ **then**
      $credit[src\_node] \leftarrow credit[src\_node] + P_{base} * fr.ttl$
    else
      ignore
    end if
  end if

Different protocols at the application layer have different payment methods, for example: when reading files, payment is made according to the number of bytes; when instant messaging messages, payment is made according to the number of frames or bytes; in unicast protocols, if it is only a relay node, the data is charged per frame; if it is sent to itself, only the heartbeat data frame is charged; the other multicast or broadcast protocols correspond to the protocol layer for fee settlement.

3.3.3.2. Multicast Protocol

Multicast protocol is a broadcasting mechanism among nodes in a group. Since the nodes in the group may not be directly connected, the data is relayed through non-group nodes. From the perspective of the nodes in the group, it always transmits the messages to be broadcasted through virtual connections. When the nodes in the group receive the multicast messages sent to themselves, they are assigned to the corresponding group processor according to the target group ID, which is managed by each group processor.

The data frames of multicast protocols are within the payload of unicast protocols:



Definition

Multicast protocols are a triplet (GT,GID,P), wherein:

14) GT → GroupType, the type of the group, which each type having a dedicated processing module
15) GID → GoupID, the ID of the group, according to which the processing module processes differently
16) P → Payload, the data protocols within each group

Initiate

Multicast messages could be initiated by nodes within the group--when data needs to be synchronized with other nodes in the group, or by nodes outside the group--when a functional module needs to transmit data to another decentralized functional module. Therefore, when a node needs to broadcast data to a group, create the data frame and send it to any node in the group.

Receiver Processing

Algorithm 6 Process multicast frame

Input: multicast frame(payload of unicast) fr
  Validate fr
  $proc\_of\_gt \leftarrow FindProc(fr.gt)$
  if $proc\_of\_gt \neq NIL$ **then**
    $need\_broadcast \leftarrow proc\_of\_gt(fr.gid, fr.payload)$
    if $need\_broadcast = true$ **then**
      $nodes \leftarrow GetNeighbours(fr.gid)$
      Send fr as multicast to nodes
    else
      Ignore
    end if
  else
    Ignore
  end if

When a node in the group receives a multicast frame, it first verifies the validity of the data and then, according to the GroupType in the frame, finds the corresponding processor registered in the system. If exists, the processor is invoked to process the data. After the processor completes the processing, it needs to return a status value to the system indicating whether the frame is valid and whether it needs to be forwarded. If the status value is true, the multicast component will forward the multicast frame to other nodes in the same group.

Cost Calculation

FavorX does not impose restrictions on the protocol layer of multicast applications for accounting purposes. In most cases, multicasting is used for underlying data synchronization, so there is no mutual accounting.

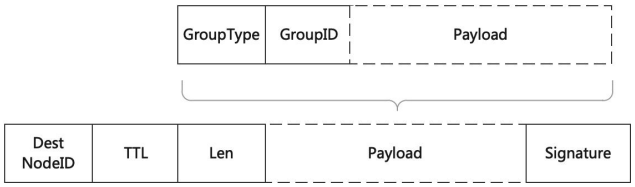
3.3.3.3. Multicast Protocol

Multicast protocols refer to applications (clients or group-external nodes) sending requests to a specific group to perform certain operations or obtain certain data. For example, in FavorX, a group can be created to wrap an Ethereum service, and clients send Ethereum transactions

to this group, which then submit the transactions to Ethereum, thus realizing the proxy Ethereum function. Additionally, data services can be provided to clients by a group, and the clients can send Create/Retrieve/Update/Delete (CRUD) commands to the group to achieve decentralized access to the database.

In a typical implementation of FavorX, the client first seeks to establish a virtual connection with the nodes in the group, and then sends a multicast request to the node. When nodes in the group receive requests, the requests will be forwarded to the actual server according to the actual service it proxies, or data will be read from the server and returned to the client. When nodes in the group receive requests, the requests will be forwarded to the actual server according to the actual service it proxies, or data will be read from the server and returned to the client.

Like multicast protocols, the multicast protocol is also implemented on the basis of unicast protocol, and its unicast command code is 0x20.



Definition

Multicast protocols are Group (GT,GID,P), wherein:

17) GT → GroupType, the type of the group, which each type having a dedicated processing module
18) GID → GoupID, the ID of the group, according to which the processing module processes differently
19) P → Payload, the data protocols within each group

Initiate

Multicast messages originate from nodes outside the group and are delivered to a node within the group.

Receiver Processing

Algorithm 7 Process groupcast frame

Input: groupcast frame(payload of unicast) fr
  Validate fr
  $proc\_of\_gt \leftarrow FindProc(fr.gt)$
  if $proc\_of\_gt \neq NIL$ **then**
    $response \leftarrow proc\_of\_gt(fr.gid, fr.payload)$
    if $response \neq nil$ **then**
      $resp\_frame \leftarrow CreateFrame(fr.src\_node, CMD\_GROUPCAST, response)$
      Send resp_frame as fr.src_node
    else
      Ignore
    end if
  else
    Ignore
  end if

When a node in the group receives a multicast data frame, it should first validate its validity. Then, based on the registered GroupType, the handler corresponding to it will be retrieved from the system and invoked to process the data frame. If the handler returns data that needs to be responded to, the system should create a multicast frame with the data and return it to the source node.

Cost Calculation

The multicast protocol in FavorX is used for clients to send commands or read data to the servers that provide services in the form of a group, thus the application layer is responsible for computing the related fees. For example, when the clients send requests to the database service, the server will charge the initiator (i.e. the client) instead of charging the relaying nodes in the upper level. Additionally, the database service can also adopt multiple different payment modes, such as the pay-per-times mode, the package monthly payment mode, etc. FavorX entrusts the application layer to implement these payment modes.

3.3.3.4. Fault-tolerant Handling

Multicast Failure Detection

The initiator of the multicast message must be aware of the possibility of "packet loss" and "non-forwarding" of data, where "packet loss" refers to the data not arriving correctly at a node in the group during the transmission process, and "non-forwarding" refers to the node in the group receiving this data packet without forwarding it. Given the decentralized nature of the group, the initiator of the multicast needs to detect the reliability of the message and the retransmission scheme based on its own business logic. for example, in FavorX's accounting processing scheme, a customized chain is adopted as the service group, and the service nodes with bandwidth periodically submit the receipt sets that need to be billed to this group, but

there may be cases of packet loss or non-forwarding. FavorX monitors the information on the accounting chain to determine whether the receipt set has been processed, and if it has not been processed, the data of this time will be merged together and resent in the next sending to achieve effective accounting of the data traffic. For details of the specific scheme, please refer to Chapter "Data Traffic Accounting".

Amplification Attack

When the creator of a multicast message creates the data, it is broadcasted within a group, which may lead to the risk of an amplification attack, i.e., malicious nodes can exploit limited resources to make the system consume a large amount of resources by creating a multicast message and letting nodes within the group propagate the message. FavorX adopts two mechanisms to avoid amplification attacks:

1) Consumption payment: When it is required to transmit a multicast message, the node needs to pay a fee to the relay node, which increases the attack cost.

2) Application Verification: Nodes within the group adopt message validity verification to process group-type messages to ensure the correctness of the message, and broadcast the message again if the content is valid.

Service Fraud

Upon a multicast request being issued by a user, the request is relayed to a node within the group, and a response is obtained from this node. Given that FavorX employs a trustless mechanism, malicious nodes can potentially forge response data in order to deceive the requester. Given the variety of application service forms, FavorX cannot implement a consistent anti-fraud function at the application layer. A typical anti-fraud scheme is to use a KZG promise, in which the services in the group are provided to the users in the form of a KZG promise, and when the user requests certain data, the service node needs to return the data and the corresponding proof to the user simultaneously. The user uses the pre-set KZG promise to verify the correctness of the response data. For a detailed description of KZG promise, please refer here [2].

3.4. Application Protocol

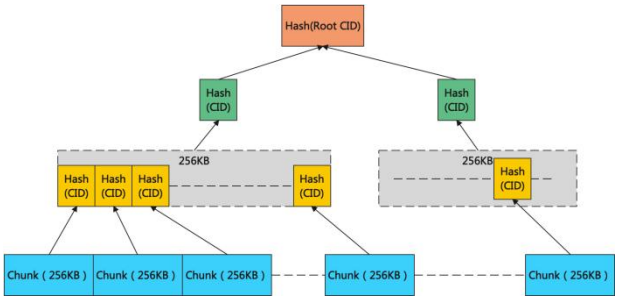
3.4.1.File Distribution Protocol

File Origination

Inspired by the IPFS and SWARM/BZZ projects, FavorX implements file and folder distribution in a decentralized network. In FavorX, a file is a piece of data and the result of a split hash tree represented in pyramid form. Specifically, the file content is divided into

[2] KZG polynomial commitments – https://dankradfeist.de/ethereum/2020/06/16/kate-polynomial-commitments.html

chunks of 256KB, and all these chunks are hashed and arranged, and then sliced again when reaching 256KB, and so on until only one hash is left, each of which is referred to as a Content IDentifier (CID), and the top-level hash is referred to as the Root CID.



In the FavorX system, the file originals are stored in what are called storage nodes in the form of an entirety. FavorX supports multiple ways to match files with storage nodes, such as IPFS nodes for file storage, and a decentralized storage marketplace within FavorX to enable file requesters and servers to automatically match in the market.

In this system, the part of the file original image other than the actual content is referred to as a file hash tree, which can be read separately to obtain the hash information of all segments of the file.

File System

Starting from IPFS, not only can single files be stored on decentralized networks, but folders can also be stored and distributed, thus making the whole network look like a huge file system. FavorX has borrowed these concepts and also achieved folder management, in which folder management is realized through manifest files in FavorX.

Manifest can be used to represent the mapping of file paths and has the ability to map URLs to the directory tree of the file system. By providing detailed routing conventions, URLs can be mapped to files in a standardized way, thus allowing for a simulated site-map/routing table, and FavorX can also be used as a virtual cloud hosting server.

A manifest is a structure that defines the mapping relationship between any paths and files, used to handle a set of related files. In addition, it also contains metadata associated with the set and its objects (files). Most importantly, the manifest entries specify the MIME type of the files so that the browser can properly handle them. The manifest can be seen as a (1) routing table, (2) index, or (3) directory tree, which enables FavorX to implement a website, database, or filesystem directory. The manifest also provides the main mechanism for URL-based addressing in FavorX. The domain part of the URL is mapped to the manifest, and the path part of the URL is matched in the manifest to find the file entry to be served.

Currently, the manifest is presented in the form of a compressed Trie (http://en.wikipedia.org/wiki/Trie), where each Trie node is serialized in JSON. The JSON structure has the minimum manifest entry array, which contains the path and reference (hash

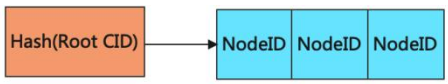
root address). The path portion is used to match the URL path, and if the path is a common prefix among multiple paths in the set, then references can point to the embedded manifest. When a file is retrieved via URL, FavorX resolves the domain into a reference to the root manifest which is traversed recursively to find the matching path.

The higher level APIs of the manifest provide the functionality to upload and download a single document as a file, a collection (manifest) as a directory, and an interface to add documents to and delete from a collection on the path. It should be noted that the deletion here only implies the creation of a new list with the document related paths deleted.

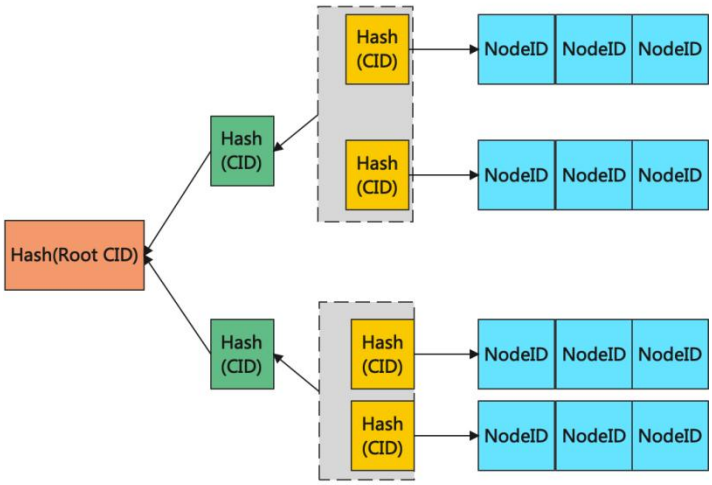
In FavorX data transmission, the CID index is defined in the form of {CID: ParentCID} in the form of fragments, which is used to indicate and quickly locate the location of a CID in the list path, thus allowing the root CID to be traced up one level.

Storage Information

In FavorX, file information refers to the information stored on which nodes a file or chunk corresponding to a CID is stored, including "original image storage information" and "chunk storage information". The former is statically retained and needs to be stored in a specific way; while the latter is passively recorded and maintained during the data transmission process. When the client reads the file or folder content, it needs to rely on the file information to find the corresponding node and extract the data.



Preimage Storage Infomation



Chunk Storage Information

There are multiple ways to store the "original image storage information", including storing it on a centralized server (such as the BitTorrent seed server scheme), pushing it to a node through distributed hash table (DHT) technology (IPFS adopts this scheme), or storing it on the blockchain (the scheme used by FileCoin).

FavorX network has designed a "Chunk Storage Information" service group, which acts as a prophet for the client. The client can send the target file to the service group in a specific format, and the service group will return the "Chunk Storage Information" to the client. The client can use this Chunk Storage Information to read data.

Chunk Storage Information is maintained by each node. When node B reads a Chunk corresponding to a hash H from node A, a record H→NodeB will be added in node A. Consequently, node A will record the node information of all Chunks with hash value H read in a certain period of time, forming a record information H→[NodeId, NodeId, …], and this information is called "Chunk Storage Information".

Chunk storage information is cached in nodes to discover multiple data in the system. The application side looks for certain Chunk storage information to node A, and gets possible Chunk cached nodes B, C, ..., and so on. Then, further queries are made to nodes B and C, and through iteration, a table of all the Chunk cache nodes information of a certain original image file is formed. When reading each Chunk in the follow-up, the node information table is used to select the node for data transmission according to the ant colony algorithm.

Optimization Strategies

According to the previous section, with the increase of the number of clients reading the same file, the number of chunks of the file cached by the relay nodes will also increase correspondingly, providing more sources of data for subsequent nodes to choose from. In the case of multiple data sources, optimizing the client's selection strategy will lead to better reading and playback effects. FavorX optimizes the selection of read data sources through Ant Colony Optimization (ACO) algorithm.

In the 1990s, Italian scholars Dorigo, Maniezzo, and others first proposed the Ant System (AS) or Ant Colony System (ACS). They found that the ant colony could exhibit some intelligent behaviors when observing the process of ants searching for food, that is, finding the optimal path to the food source in different environments. Further research found that this is because the ants release a substance called "pheromone", and other ants can also sense the "pheromone", so they will choose the path with high "pheromone" concentration to go forward; and then the other ants going to that path will also release "pheromone", forming a positive feedback mechanism. In the end, the whole ant colony will be able to find the optimum path to the food source.

In FavorX, the food source is nodes with data slices, where the file information records which nodes have data slices. In the reading process, data is read preferentially from nodes with good connection quality to maximize the utilization of network bandwidth between nodes. To this end, we introduce an initial heuristic value e, which depends on the previous value or a random number between [x, x + r); at the same time, an N value is considered to measure the current node's busy degree. Rank all available nodes according to the processing of e and N, and select the node ranked at the front to read data. After the reading is completed, update e in the reverse direction according to the information pheromone concentration composed of data reading size/delay to form a positive feedback loop.

The ant colony algorithm is realized by a set of interrelated processes:

1. Node selection;

Algorithm 8 ACO: Node selection

Input: nodes
Output: Best node
for $i \in nodes$ **do**
$e^{'}_{i} \leftarrow \frac{e_i}{n_i+1}$
end for
$e^{''} \leftarrow sort(e^{'})$
return $e^{''}[0]$

2. Recording information at the beginning of reading;

Algorithm 9 ACO: Start Read

Input: nodes
Output: Best node
$N_i \leftarrow N_i + 1$
$T_i \leftarrow \{when : time.Now(), len : len\_of(target)\}$

3. Updating information at the end of reading;

Algorithm 10 ACO: Update On Read Finished

$N_i \leftarrow N_i - 1$ { Update }
$e_i \leftarrow e_i + \frac{T_i.len}{time.Now()-T_i.len()}$
if $e_i > MAX\_E$ **then**
$e_i \leftarrow MAX\_E$
end if

4. Information pheromone evaporation on a regular basis;

Algorithm 11 ACO: Timed Process

for $i \in nodes$ **do**
$e_i \leftarrow e_i \times 0.9$
if $e_i \leq MIN\_E$ **then**
$e_i \leftarrow MIN\_E$
end if
end for

Through the above set of algorithms, nodes can track the data transmission state between themselves and other data sources in real time, and select the best transmission mode from all available nodes. In this scheme, the transmission speed between two nodes is the only criterion for judgment, regardless of physical location and network topology.

File Oracle

FavorX has realized the unified access and data distribution to different storage backends through its file oracle technology, currently offering built-in support for decentralized storage systems, IPFS storage systems, and centralized cloud storage systems, and capable of extending support to other storage backends through the development of corresponding plugins.

The file oracle provides a decentralized query service for the original image storage information in the form of multicast to the client, and the client sends the query request in the standard format to the oracle, which is in the form of a simplified URL:

<scheme>://<user>:<password>@<host>:<port>/<path>

scheme: The protocols, where common ones are HTTP/HTTPS for centralized cloud storage backends, IPFS for IPFS storage backends, and FXFS for the built-in storage backends of FavorX.

The username (user) is an optional item and is available for HTTP/HTTPS protocols, while IPFS protocols don't need it and in FXFS it will be the user's account address.

password: The password, optional, which is currently applied to http/https only.

host: The host, in HTTP/HTTPS refers to the hostname; in IPFS it is not necessary, if exists it is the universal gateway domain name for IPFS; for FXFS, if exists it indicates the use of a name resolution system, if not exists it does not use.

port: Port is an optional item that in HTTP/HTTPS refers to the port of the host; in IPFS, it refers to the port of the gateway; while in FXFS, it does not need to be used.

path: The path information of the file, in fact, is defined by the corresponding storage backend.

The data format returned by oracle is always consistent and does not differ depending on the different storage backends. Therefore, the client can parse it in a unified way and proceed with subsequent operations. The data format returned by the oracle is as follows:

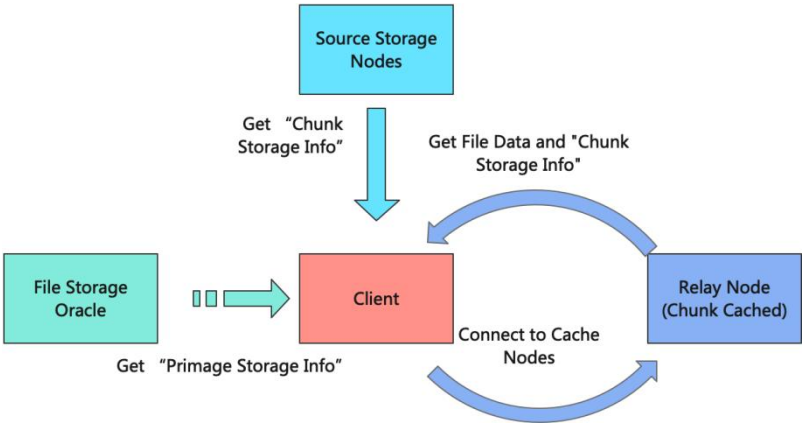
`<RootCID>:[<NodeID>]`

The oracle responds to the RootCID and NodeID information of the original image file, generally speaking, the original image file has multiple storage nodes. The NodeID in the response must be the FavorX node, and the client can access these nodes through routing.

Distribution Process

During the reading process, the client will continuously exchange specific CID-corresponding Chunk storage information with the nodes in order to obtain the node information that has cached the Chunk and request data from the corresponding nodes

through the relay nodes. When client A reads the data corresponding to a specific CID from the storage node C through the relay node B, B will cache the data locally, and C will also add B as a new cached node of Chunk storage information to its local machine. When other client D requests the corresponding Chunk storage information from C, C will also send B as an available data source to D, and D will select to read data from C or B according to the connection situation.



After obtaining the "Original Image Storage Information" from the file oracle, the client reads the file "Original Image Information

Distribution Protocol" from the storage node. It is a process of obtaining actual data from the FavorX network according to the obtained file CID. In FavorX, file distribution is divided into three stages:

1) Obtaining the original image storage information corresponding to CIDs from the oracle group
2) Obtaining the file storage node from the original image storage information, and obtaining the Chunk storage information from the file storage node
3) Obtaining the cache node of the Chunk data from the Chunk storage information, and obtaining the data content and the remaining Chunk storage information from the cache node

Protocol Descriptions

FavorX's file distribution protocol consists of four operation primitives, namely QueryFileInfo, QueryHashTree, QueryChunkInfo, and GetChunkData. QueryFileInfo is used to query the original image storage information to the oracle group, QueryHashTree is used to request the corresponding hash tree of a file from the node, QueryChunkInfo is used to read the Chunk storage information from a node, and GetChunkData is used to read the Chunk fragments. These four primitives all contain two parts of request and response, and have different data formats.

QueryFileInfo

➔ Request:

Data Format:

<scheme>://<user>:<password>@<host>:<port>/<path>, described in detail in the section "File Oracle".

Sending Mode:

Multicast, from the client to the oracle group

Receiver Processing Process

Algorithm 12 Process QueryFileInfo

Input: $FileURL$
  $protocol, fileInfo \leftarrow GetProtocol(FileURL)$
  $procedure \leftarrow GetRegisteredProc(protocol)$
  if $procedure \neq NIL$ **then**
    $resp \leftarrow procedure(fileInfo)$
  else
    $resp \leftarrow ERROR\_NOT\_FOUND$
  end if
  $frame \leftarrow CreateQFIResp(resp)$
  $send\_unicast(src\_node, frame)$

When a node or relay node in the file oracle group receives a request, it first parses out the protocol information and target file information, then finds the registered processor according to the protocol information, if it finds the processor, it obtains the original image storage information, if not, it sets the result to a NOT_FOUND error, after the processing is completed, the result is sent back to the requester via unicast.

➔ Response:

Data Format:

{RootCID:[nodeId,NodeId,....,NodeId],proof}, described in detail in the section “Storage Information”

Sending Mode:

Unicast, back to the requester from the nodes of the oracle

Receiver Processing Algorithm:

Algorithm 13 Process QueryFileInfoResp

Input: $QFIResp, commitment$

$passed \leftarrow ZKVerify(QFResp.result, QFResp.proof, commitment$

if $passed \neq false$ **then**

  Enter Next State

else

  Retry to retrieve from next node or reture error

end if

Upon receiving the response, the receiver verifies the correctness of the returned value via the known commitment value. If correct, it enters the next step--selecting some of these nodes to request reading the HashTree and subsequent work.

QueryHashTree

➔ Request:

Data Format:

RootCID, files parsed from RootCID

Sending Mode:

Unicast, from client to file storage nodes

Receiver Processing Process

After the receiver (file storage nodes) or relay nodes receive the request, they will check if there is information corresponding to the RootCID in the local cache. If so, a response is generated and returned.

➔ Response:

Data Format:

CID:[SUB_CID], this is an object, with each object's value being an array of sub-hashes until a particular SUB_CID corresponds to data.

Sending Mode:

Unicast, sent back to the requester by either the file storage node or the relay node

Receiver Processing Algorithm:

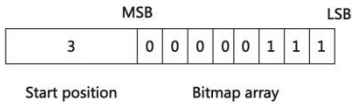
Upon receipt of the response, the receiver checks the integrity of the hash tree, and if it is correct, proceeds to request "Chunk Storage Information" from the source node as needed.

QueryChunkInfo

➔ Request:

Data Format:

[start_pos, bitmap_array], this is the bitmap of the file hash tree, and the part of the "Chunk storage information" should be given by the other side, with '1' if this position is available, otherwise '0'. For example, if there is a 2560K byte file that has been divided into 10 Chunks, and node A queries the information of the third, fourth, and fifth Chunks to node B, then the content of QCIReq is: [3, 0x07]:



It is obvious that the length of the Bitmap is related to the total number of Chunks to be queried at one time, and the total number of Chunks queried at one time is related to the cache prepared for streaming media playback, and this part of the implementation details is determined by the application.

Sending Mode:

Unicast, from client to oracle group

Receiver Processing Process

Algorithm 14 Process QueryChunkStorageInfo

Input: $start_pos, bitmap_array$
  $nodes\_ret \leftarrow []$
  for $bit\_i \in bitmap\_array$ **do**
    if $test_bit(bitmap\_array, bit\_i) = TRUE$ **then**
      $chunk\_cid \leftarrow GetCID(start_pos, bit\_i)$
      $nodes \leftarrow GetNodesOfDelivered(chunk\_cid)$
      $nodes\_ret \leftarrow nodes\_ret \cup nodes$
    end if
  end for
  $payload \leftarrow CreateQFIResp(start\_pos, bitmap\_array)$
  $send\_unicast(src\_node, payload)$

When a node in the file oracle group receives the request, it first parses out the protocol information and the target file information, then looks up the registered handlers according to the protocol information, if the handler is found, use this handler to obtain the original image storage information, if not, set the result as NOT_FOUND error, after processing is completed, send the result to the requester via unicast.

➔ Response:

Data Format:

{RootCID:[nodeId,NodeId,....,NodeId],proof}, described in detail in the section “Storage Information”

Sending Mode:

Unicast, back to the requester from the nodes of the oracle

Receiver Processing Algorithm:

Upon receiving the response, the receiver selects appropriate nodes according to the Ant Colony Algorithm, and then invokes the GetChunkData primitive to read the data.

GetChunkData

➔ Request:

Data Format:

CID, hash of the data chunk that needs to be read.

Sending Mode:

Multicast, from client to the cache nodes of the chunk

Receiver Processing Process

Algorithm 15 Process GetChunkReq

Input: $target\_node, cid$
  $chunk\_exist \leftarrow CheckChunkInCache(cid)$
  if $chunk\_exist = TRUE$ **then**
    if $NodeTypeOf(prev\_hop\_node) = FULL\_NODE$ **then**
      $delivered\_chunks[cid] \leftarrow delivered\_chunks[cid] \cup PrevHop$
    end if
    $payload \leftarrow CreateGetChunkResp(src\_node, cid, GetChunkFromCache(cid))$
    $send\_unicast(prev\_hop\_node, payload)$
  else
    $cid\_in\_pending \leftarrow pending\_chunks[cid] \neq \emptyset$
    $pending\_chunks[cid] \leftarrow pending\_chunks[cid] \cup prev\_hop\_node$
    if $cid\_in\_pending = FALSE$ **then**
      $payload \leftarrow CreateGetChunk(cid)$
      $next\_hop \leftarrow GetNextHopFromRouteTable(target\_node)$
      $send\_neighbour(next\_hop, payload)$
    end if
  end if

When the target cache nodes or relay nodes of the chunk receive the request, they first check if they have the data locally. If they do, they process it locally, otherwise they record the incomplete request and then forward it to the next hop in their routing table. The steps of local processing involve two: 1. creating a response frame to respond to the previous hop address (prev_hop_node); 2. recording the transmission to prev_hop_node in the Chunk storage information.

Note that forwarding to the next hop is always performed here in order to avoid the problem of data redundancy when responding. For example: Node A and Node B consecutively requested data from Node C and Node D to Node E, and with the solution of FavorX, Node C received the two requests from Node A and Node B but only sent one to Node D; similarly, Node D only requested from Node E once and returned the data to Node C, then Node C cached it and sent it to Node A and B separately. This scheme avoided the redundancy of Node D receiving requests from Node C and returning data.

➔ Response:

Data Format:

{cid, chunk_data}, where cid = hash(chunk_data) is the information of CID and the actual data content

Sending Mode:

Unicast, sent from the previous hop.

Receiver Processing Algorithm:

Algorithm 16 Process GetChunkResp

Input: $cid, chunk\_data$
$UpdateCache(cid, chunk\_data)$
$payload \leftarrow CreateChunkResp(cid, chunk\_data)$
for node in pending_chunks[cid] **do**
$send\_neighbour(node, payload)$
end for
$pending\_chunks[cid] \leftarrow \emptyset$

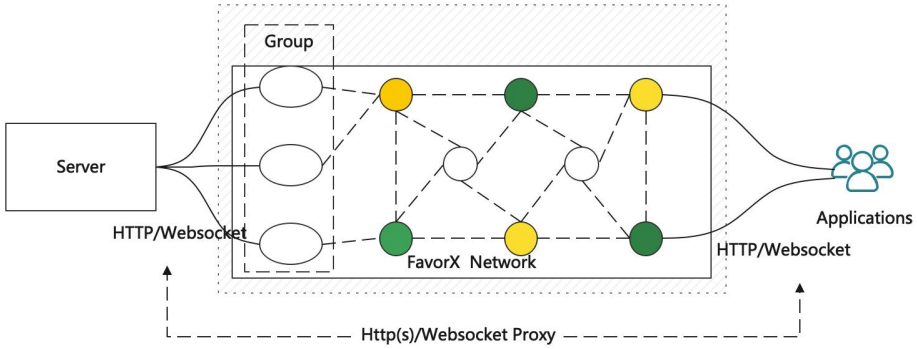
After the receiver receives the data response, it first updates the local cache, then checks which previous hop neighbor nodes are waiting for the response, and responds to the Chunk data to them one by one.

Accounting Description

Only GetChunkDataResp is chargeable, according to the actual number of bytes.

3.4.2. Web Service Proxy Protocol

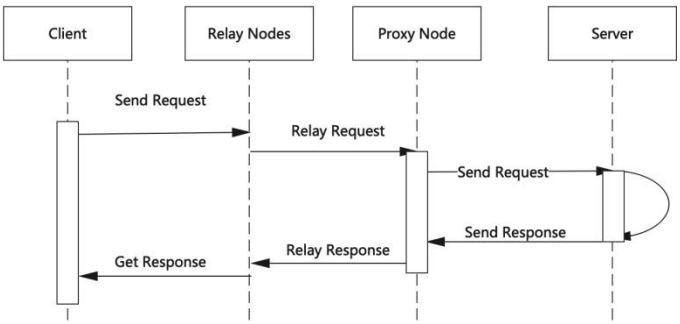
FavorX provides a new decentralized network architecture that allows developers to achieve decentralization of applications without the need for a server, using P2P technology. FavorX encapsulates using the RESTful protocol to establish virtual connections between two nodes and provides HTTP and WebSocket proxy services for customer codes on these nodes. Moreover, FavorX also allows developers to develop interface protocols fully compliant with the VPN standard to directly leverage FavorX's decentralized features for Web2 applications. At present, HTTP(S) and WebSocket proxies are supported in the FavorX system.



Provides proxy services for applications
FavorX provides two types of proxy services, stateful and stateless, where the former can implement push, which means sending data from the server side to the application side and vice versa with the same effect as WebSocket, and the latter provides requests/responses for applications with the same effect as HTTP.

Http Proxy

This feature enables applications to acquire Http services in a decentralized way and, due to the stateless characteristic of Http services, nodes in the proxy group of FavorX do not need to maintain the information and status of clients, thus improving the utilization efficiency of the nodes.



➔ Request:

Data Format:

Data format: Same as the Http multipart [3] uploading format

Sending Mode:

Multicast, from the client to the proxy node group

Receiver Processing Process

Algorithm 17 Process HttpProxyReq

Input: $src\_node, data$ {This should be execute in a separated thread}
$method, mimedata \leftarrow decode(data)$
await resp = SendHttpData(server, method, mimedata)
$frame \leftarrow CreateHttpRespFrame(encode(resp))$
$send\_unicast(src\_node, frame)$

When the nodes within the proxy group receive the request, the data to be proxied is parsed from the frame and then submitted to the actual server in the form of Http multipart, and then wait for the response. When the data response from the server arrives, it is pushed to the requester by unicast.

➔ Response:

Data Format:

Encoded HTTP Response format, with status code encoded.

Sending Mode:

Unicast, ient back to the requester from a node within the proxy group

Receiver Processing Algorithm:

Algorithm 13 Process QueryFileInfoResp

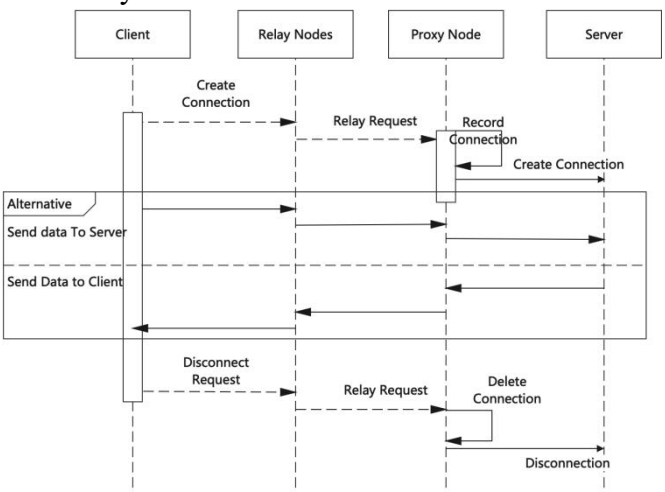
Input: $QFIResp, commitment$
$passed \leftarrow ZKVerify(QFResp.result, QFResp.proof, commitment$
if $passed \neq false$ **then**
  Enter Next State
else
  Retry to retrieve from next node or reture error
end if

Upon receiving the response, the receiver (optional) verifies the integrity of the return value through the known commitment value, and if correct, parses the response data for application processing.

[3] https://www.w3.org/Protocols/rfc1341/7_2_Multipart.html

WebSocket Proxy

This feature enables applications to gain WebSocket services in a decentralized manner. WebSocket is a stateful service, thus clients can establish connections with proxy nodes, and the proxy nodes will create a WebSocket connection with the server according to the requests of the client nodes. In the subsequent process, the client can directly send data to the Server, and the Server can also actively send data to the Client.



➔ Create/Disconnect a Connection:

Data Format:

Data Format: the subprotocol coding, “create/destroy”

Sending Mode:

Multicast, from the client to the proxy node group

Receiver Processing Process

Creation: When a node in the proxy group receives a request, a connection is created with the Server, and the src_node of the request and the connection relationship is recorded in the connection information table.

Destruction: When a node in the proxy group receives a request, the connection with the Server is disconnected, and the information about the connection is removed from the connection information table.

➔ Data Transmission

Data Format:

Any data format.

Sending Mode:

When sending from the client to the server, it is a multicast mode.

When sending from the server to the client, it is a unicast mode.

Receiver Processing Algorithm:

The client is treated in the same way as the original websocket.

Accounting Scheme

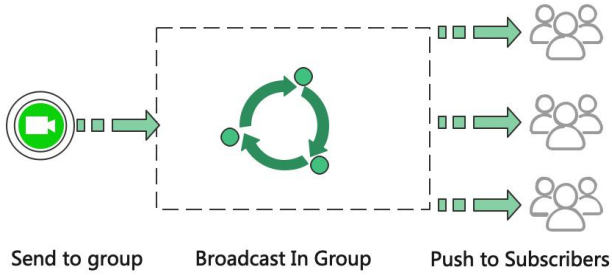
The server can adopt a pay-per-use or record the purchase situation on the server side to decide the response content according to whether the client is authorized. Therefore, a server-side charge scheme that is consistent with existing application modes can be adopted. The implementation process of accounting varies according to different accounting schemes. If it is pay-per-use, then the same can be realized by offline small payments and on-chain charges. If prepaid or other schemes are used, smart contracts can be used for payments and chain-based authorization information. This part of the scheme is implemented by the server.

Attack and Defense

Due to the decentralized nature, any node in the network can tamper with the data in the middle transmission, so the data needs to be signed to confirm the validity of the data. In these proxy services, the HTTPS encryption model is used, which enables the service and client to sign the data when initiating, to ensure the security of the service data. The WSS scheme can also be used to establish an encrypted and reliable channel between the two sides, and the data is transmitted through this reliable channel to ensure the security of the data.

3.4.3. Live Stream Acceleration Protocol

The Live Acceleration Protocol is the live forwarding function on FavorX. The live streaming acceleration protocol has three roles: the live source, the forwarding group, and the client. Its data is broadcasted within the forwarding group after being emitted from the live source node and then pushed to the subscribers of the group by the nodes in the forwarding group.



Due to the varying number of viewers for each live streaming, the number of viewers could also fluctuate during the streaming, thus the forwarding group should have the ability of node elastic scalability, i.e. automatically expanding the number of nodes within the group when the number of viewers increases, and reducing the number of members in the group when the number of viewers decreases.

An elastic scalability algorithm is constructed during the subscription process. If the relay node has enough bandwidth, it can choose to join the distribution group of the live broadcast. Once it joins the group, it will have more chances to be subscribed by the client node and serve more clients, thus gaining more traffic revenue. If all users of a node have unsubscribed, then it can choose to exit the distribution group.

Since broadcasting of video content is required in the distribution group, a large amount of bandwidth resources will be consumed. In order to avoid data duplication, the broadcasting of data in the distribution group adopts a push/pull scheme, that is, when data needs to be broadcast, the source node first pushes the hash value of the data to the target node. When the target node receives the hash value, it checks whether the data exists. If the data does not exist, it requests the data from the source node of the data, otherwise, it sends the existing message to the node without reading the data.

In order to avoid malicious nodes sending invalid data to multicast groups to waste system bandwidth for amplification attacks, all broadcast data within the group must be signed. To enable offline verification of these data, FavorX makes the following provisions for the Live Acceleration Protocol.

1. Each live source + forwarding group is referred to as a live channel, each of which can have a name determined by the live source, and no two channels with the same name can have the same live source.
2. The forwarding ID for each live channel is $\text{Signature}_{\text{owner}}(\text{sha256}(\text{group\_name}))$, that is, the signature of the live source for the hashed name of the live channel.
3. Users obtain and record the broadcasting name, source address, and forwarding group ID in some ways and can verify the correctness of these data.
4. For each data segment Chunk in the broadcasting, the source needs to hash it and then sign it to send in the group. The receiving nodes in the group need to verify the correctness of the data and then decide whether to forward it or not.

Protocol Overview

The Live Streaming Protocol on FavorX consists of five operations primitives. These five primitives are divided into two groups: the first group is user-related, including StartLive, StopLive and SubscribeLive. The second group is for internal use among the forwarding group, PushChunkHash and GetChunkByHash. The broadcaster starts or stops the live streaming with StartLive and StopLive, respectively. Meanwhile, the users subscribe to or unsubscribe from the video with SubscribeLive and UnsubscribeLive. The nodes within the forwarding group achieve live streaming data synchronization through PushChunkHash and GetChunkByHash.

StartLive

➔ Request:

Data Format:

None.

Sending Mode:

Multicast, initiated by the live source, broadcasted within the forwarding group.

Receiver Processing Process

The nodes within the forwarding group receiving the message will set to allow the data from the live source to be converted and forwarded to other nodes.

StopLive

➔ Request:

Data Format:

None.

Sending Mode:

Multicast, initiated by the live source, broadcasted within the forwarding group.

Receiver Processing Process

When the nodes in the forwarding group receive the message, they broadcast it to notify the nodes in the group that the live stream has ended. When the nodes in the group receive the message, they notify all subscribed nodes that the live stream has ended.

Upon receiving the message, the subscribed nodes can unsubscribe the message.

SubscribeLive

➔ Request:

Data Format:

The ID of live stream group.

Sending Mode:

Multicast, from the client to the forwarding group

Receiver Processing Process

Algorithm 18 Process SubscribeLive

Input: $src\_node, target\_node, liveID$
$resp \leftarrow ERR_OUT_RESOURCE$
$enough\_band \leftarrow CheckBandwidth()$
if $enough\_band$ **then**
  if $self.addr = target\_node$ **then**
    $resp \leftarrow ERR_OK$
  else
  await result $\leftarrow JoinGroup(liveID)$
  end if
  if result = TRUE **then**
    $resp \leftarrow ERR_OK$
  end if
end if
$frame \leftarrow CreateSubscibeResp(encode(resp))$
$send\_unicast(src\_node, frame)$

After the relay node receives this message, it decides whether it can participate in the forwarding group according to its local resources (especially the upstream bandwidth usage). If it can, it first joins the forwarding group. After successful joining, it returns to the requester that the node is already a node in the broadcasting group.

Upon receiving this request, the nodes in the forwarding group will respond with success if they agree to subscribe from the host, or return ERR_OUT_RESOURCE if they do not agree.

Due to the relay nodes in the routing process from the client node to the broadcast group node, the client may receive multiple responses, so the client can select one or more of these nodes to connect. The client can take data from one data source, and the other several can serve as alternative data sources. When the data from the main data source is not smooth, it can switch to the alternative data source to read the data.

➔ Response:

Data Format:

Live Stream ID

Sending Mode:

Unicast, sends back to the requestor from the forwarding node

Receiver Processing Algorithm:

Algorithm 19 Process SubscribeLiveResp

Input: $src\_id, liveID, ret\_code$
await handle $\leftarrow CreateSocketProxy(src\_id, liveID)$**if** live_connections[liveID] $= \emptyset$ **then**
  $live\_connections[liveID] \leftarrow live\_connections[liveID] \cup \{src\_id, handle, T\}$
else
  $live\_connections[liveID] \leftarrow live\_connections[liveID] \cup \{src\_id, handle, F\}$
end if

Upon receiving the response, if the return value is ERR_OK, the node is recorded and based on its own status, the node is determined to be the primary data source, then sends a request to establish a Websocket connection and stores the connection information.

PushChunkHash

➔ Request:

Data Format:

{chunk_hash,signature}.

Sending Mode:

Multicast, broadcast between nodes in the forwarding group. Unicast, push from the forwarding group to the client.

Receiver Processing Process

Algorithm 20 Process PushChunkHash

Input: $src\_id, liveID, chunkhash, signature$
  $valid \leftarrow VerifySignature(chunkhash, signature$
  if $valid$ **then**
    if $seen\_hashes[chunkhash] = \emptyset$ **then**
      $frame \leftarrow CreateGetChunkFrame(hash)$
      $send\_unicast(src\_id, frame)$
    end if
    $frame \leftarrow CreateGetChunkFrame(chunkhash, signature)$
    for $node \in group_nodes[liveID]$ **do**
      $send\_unicast(src\_id, frame)$
    end for
  end if

The receiver first verifies the validity of the signature. If valid, it checks if there is data corresponding to the hash locally. If yes, it simply broadcasts the message, otherwise, it requests to read the data and then broadcasts the message.

GetChunkByHash

➔ Request:

Data Format:

{chunkhash}.

Sending Mode:

Unicast, from client to the data sending end

Receiver Processing Process

After the receiver receives this request, it sends the data corresponding to the hash to the requester.

Accounting Scheme

Apart from the live data, other commands are charged on a per-frame basis.

For live data, the live source will be deducted for the traffic fee.

The requester of the data needs to pay the traffic fee.

Attack and Defense

In order to avoid the forwarding nodes requesting extra traffic fees by forging PushChunkHash, the Hash in the command needs to be signed.

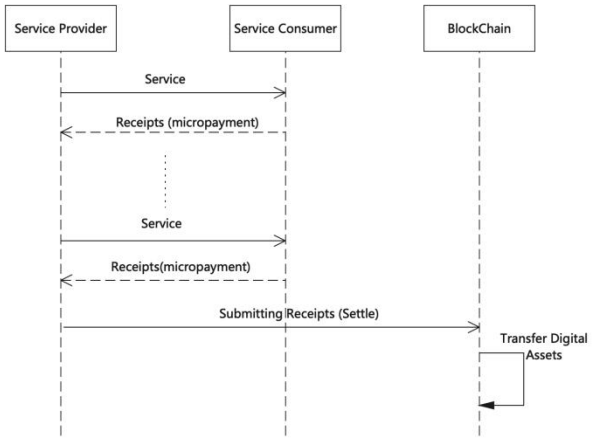
Any node in the group will record and evaluate the credit value of other nodes in the group locally. If the actual data and corresponding hashes transmitted from other nodes are inconsistent, in addition to not forwarding the data, the credit value of the node will be reduced and the traffic fee will be refused.

3.4.4. Protocol Expansion

3.4.4.1. Data Traffic Accounting

Accounting Basis

The accounting in FavorX is composed of two parts: fuel and unit price. The fuel in the accounting is calculated according to the frame length, and the unit price will change with the degree of network congestion, so the total cost paid by the payer is Gas* Gas_price. In order to reduce the workload of on-chain accounting, FavorX adopts a scheme of off-chain small payment and on-chain settlement.



FavorX has created Reducible Receipts, a mechanism that greatly reduces the storage and accounting requirements on the chain, allowing service providers to continuously provide services to consumers while consumers only need to provide the latest reducible receipt to gain the benefits of all services. In addition, service providers can also merge receipts from multiple consumers at the same time into a single set of receipts to reduce storage and accounting costs on the chain.

Due to the delayed settlement of data, it may result in insufficient balance in the consumer's account when settlement, leading to the failure of digital asset transfer, so nodes providing services should expect a small part of the income to be unreceived, and in the system, accounts with arrears will be recorded as well. Furthermore, accounts are divided into fund accounts and application accounts in FavorX, with a small part of assets that cannot be transferred agreed to be applicable to application payment only.

Definition

The payment of traffic charges from the client node to the service node is the implementation of P2P charging, which is done by paying a cheque from the client node to the service node. The payment process is broken down into the following steps:

1. The total traffic DT transmitted by the confirmed service nodes of each client node is recorded in the system (on the chain), with the DT value defaulting to 0 if no interaction has taken place.
2. When the client node and the service node are connected, the DT value interactively recorded by both parties is actually always presented by the service node to the client node;
3. After each segment of data T (256K by default) is transmitted by the service node, the client node presents a signed receipt Receipts to the service node with the value DT+nT where n is accumulated constantly
4. When the service node receives a new Receipts, it replaces the original Receipts of the client node with the new Receipts
5. The service node presents the Receipts to the chain, and after the chain reads the value in this Receipts and calculates the difference with the last traffic:
    a. Deduct the traffic charges from the client node
    b. Add the corresponding traffic charges to the service node account
    c. Deduct the corresponding commission charges from the service node account
    d. The total traffic on the chain is recorded as DT+nT

The reason for deducting the commission charges is to allow the service node to submit the transaction receipt of the traffic to the chain after the traffic charges reach a certain level。

3.4.4.2. Bandwidth Reward

Each node, depending on the amount of token it has pledged, will be assigned a certain energy value. When the node pays for a traffic fee to the server after reading data from the server, part of the energy value can be transferred to the service node, and the energy value

obtained by the service node will form its own computing power. With this computing power, the system reward can be obtained. This part of the reward is called the bandwidth reward. Definition:

This system rewards tokens to nodes based on their computational power at each block height, and the probability of the reward each node can get is consistent with the ratio of its own computational power to the total computational power of the network. Nodes can get computational power by providing bandwidth service and receiving traffic fees from other nodes.

Reward Algorithm:

1. The energy of node m

$$E_m = \begin{cases} E_{base} & \text{if } Balance_m > B_{base} \\ 0 & \text{if } x \leq B_{base} \end{cases}$$

   The energy value of a node is related to its current balance value. When the balance value is greater than a certain basic data, the node energy value is E_{base}, otherwise it is 0. The fixed energy value E_{base} is set to 10K in the implementation of FavorX. E_{base} is the balance of non-transferable assets in the node application account.

   When the node pays for the data flow by signing the receipt, the node can transfer energy to the service node. The transfer formula is as follows. Assuming that the node m has signed receipts for K service nodes in the agreed time, the data flow of each receipt is DT_i, and the total data flow is as follows:

$$Dm = \sum_{i=1}^{K} DT_i$$

   The energy value obtainable by service node n from node m is:

$$E(n,m) = \frac{DT_n}{D_m}$$

2. Energy value obtained by the service node

   When service node n serves L nodes in a certain period of time, the total energy value obtained can be calculated:

$$En = \sum_{i=1}^{L} E_{(n,i)} ,$$

   In the entire network, there are a total of J service nodes in this period of time, and the total computing power can be calculated as follows:

$$E = \sum_{i=1}^{J} E_i$$

3. Reward Acquisition Method

   Service node n tracks the hash H of each block height, then signs with its own private key $SH = Sign_{pk}(H)$, and calculates:

$$\frac{SH}{2^{256}} \leq \frac{E_n}{E}$$

   When the above formula is established, the node sends a reward request to the chain: (block height Height, block hash H, own signature SH), the chain node receives the request, verifies whether the Height and H blocks are correct, then according to the signature SH obtain the corresponding node address, find out the energy value of the node and the total energy value of the system at that time, calculate whether the above formula is established, if so, add a designated digital asset as a reward to the corresponding node address.

3.4.4.3. Attack and Defense

Attackers can create a large number of client nodes to provide computing power for conspiring service nodes in order to achieve a Sybil attack. The system adopts the following scheme to increase the cost of Sybil attacks and make the attackers' returns lower than the costs:

- Staking

Each node must possess a certain staking value (Bbase) in order to possess transferable energy value. Therefore, generating a large number of nodes requires paying the corresponding staking costs.

- Miners' Node Revenue Share

The system takes a 3% commission on the data flow fees collected from service nodes, so that when the client nodes collude with the server nodes to cheat with a large amount of false data flow, they will pay a higher cost. If the data flow fee of the transaction is less than a certain value $Payment_{min}$, a fixed fee will be deducted from the server node, which prevents the client node from obtaining revenue for the server node at a low cost by using a small amount of targeted data flow receipts.

- Transferable Energy

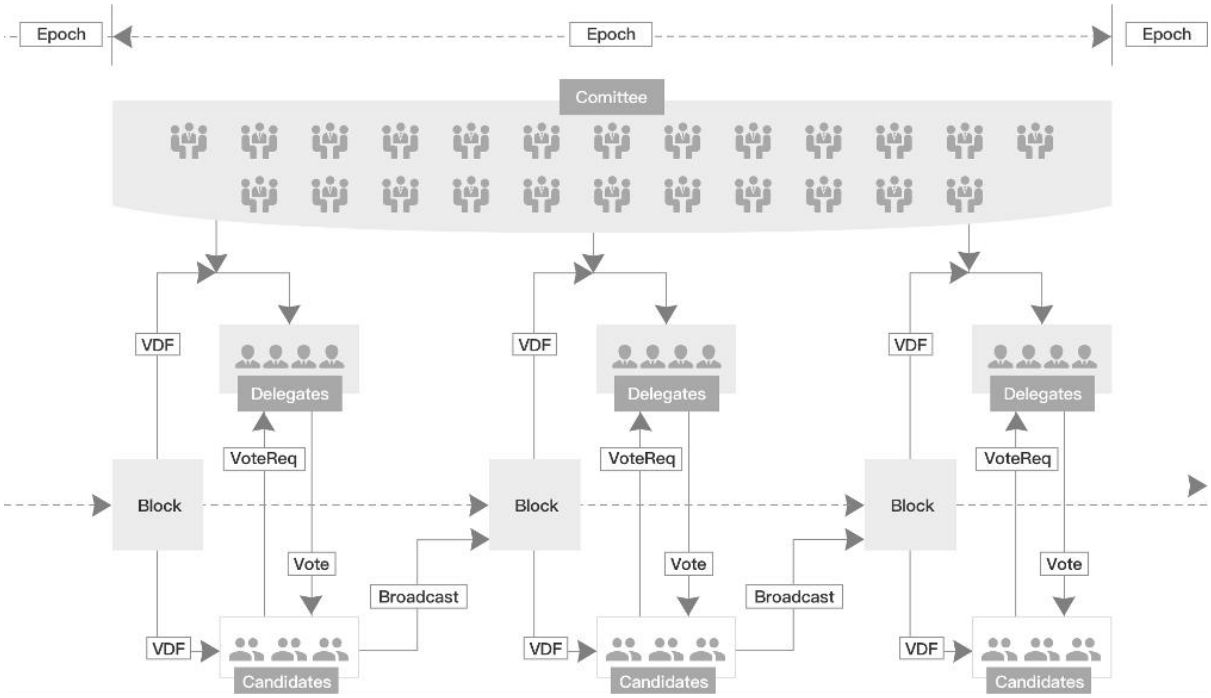
Once an attacker has gained energy through colluding client nodes and miners, after a period of time, malicious nodes can withdraw their stakes. If the miner's energy values remain constant, the attacker will gain permanent benefits after paying a one-time cost. The system solves this problem by introducing transferable energy: once a miner's reward request is

passed on the chain and rewards have been obtained, the energy value for this portion will be reset to zero.

3.5. Custom Chain

FavorX's in-built high-performance chain provides the system with decentralized storage matching, data traffic accounting and reward capabilities. This chain is enabled with a dedicated Favor consensus algorithm, featuring:
1) High Performance; 2) Low Bandwidth Requirement; 3) Fast Finalization; 4) Easily Verifiable.

3.5.1.Consensus System Model



A certain amount of block time is divided into Epochs by the Favor consensus. Before the start of each Epoch, at least a certain number of nodes will participate in the election, and all successful nodes are called committee members (committee[a1]); in each block cycle, some committee members can vote,[and these members are called Delegates. In each block cycle, each node first calculates its own chance value for block production through the VRF function, and then decides whether it needs to to send a vote request (VoteReq) to the agent based on the chance value. The agent ranks the vote requests according to the chance values for block production in all requests within a certain time period, and then sends the ranking results back to the candidates. After collecting enough votes, the candidates calculate their own weights based on the ranking results in their votes, and then propagate the blocks with weight proofs to the network. The nodes in the network select the blocks with the highest

weight to be included in their own chains. The system model is based on the following assumptions:

- All nodes are able to get the current total computing power of the whole network. The computing power of each node can be obtained on the chain.
- The number of malicious nodes that can be aggregated does not exceed 25% of all the nodes in the whole network.
- Dishonest agents can vote in any way they want, but cannot vote for two candidates with the same ranking priority without being detected.
- All agent nodes are likely to be bribed to become malicious agents.
- As the network environment is asynchronous, multiple block production requests and multiple blocks may be generated in the network at the same time. A specific node may not receive a particular block broadcast,
- and some nodes may also refuse to send data that should be sent: for example, the agent does not send the voting results, and the candidate does not broadcast the block, etc.

3.5.2.Keywords

Next, we'll briefly describe the key items in the system as follows:

Epoch: We call a certain number of block cycles an epoch, in which a fixed committee is elected from the block and different proxies are selected from the committee. The epoch scheme enables each node to obtain proxy information in each block cycle in a low-cost and clear manner.The epoch scheme enables each node to obtain agent information in a low-cost and unambiguous way during each block cycle.

Candidates: Any node can obtain a new random number through the block information and predefined VDF algorithm when the previous block is completed, calculate the block opportunity, and determine whether to send a block request based on the value.T he node that sends the block request is called a candidate. Within a block cycle, there are multiple candidates.

Committee: The set of nodes that participated in the election successfully in the previous period is called the committee.

Delegates: Nodes elected from the committee in a certain block cycle to handle all "vote requests" and then vote to select different block makers are called delegates.

VoteReq: When a candidate believes it is sufficient and has a sufficient probability of block success, it sends a block request to the delegate, and the request sent is called the vote request.

Vote: A delegate verifies each incoming voting request periodically, and sorts them based on their value. The voting requests at the top of the list are signed and sent to the candidates, and this process is called voting.

Campaign Cycle: The delegate waits for a period of time, during which they collect voting requests from each candidate. This period is called the campaign cycle, and it determines the block generation time.

Block: After collecting the voting results, each candidate forms a new block, which we call a block. Since the set of agents may give multiple signatures for different candidates in different orders, there will be multiple blocks in the same round.

Block Weight: The voting information of each block obtained in each round is called weight, and the smaller the serial number, the higher the weight; within the same block height, the block with a higher weight is more likely to be added to the blockchain.

Sign: The BLS signature scheme is adopted in the system. Thanks to the linear characteristics of the BLS signature function, the signatures of multiple agents in the block can be combined into one [a4] to save transmission bandwidth.

Keep-alive Fork: Any node can send a voting request. Any node will decide whether to broadcast its own block according to the voting result. Therefore, there may be multiple blocks competing at a height, which will prevent the block from failing due to the highest-weight node failure. This measure to ensure activity is called a keep-alive fork.

Delayed Relay: When a node receives the block information, it forwards the information after a delay of some time based on the weight of the block, which is called the Delayed Relay. Delayed relay will promote the nodes with high weights to be propagated preferentially in the network and reduce the propagation speed and range of the blocks with low weights in the network.

3.5.3. Scheme Description

The whole Favor system is mainly divided into three parts: committee election part, secret election and voting part, and block broadcasting part.

1. The committee election is a process in which nodes collaborate with each other to generate a random number and use this number to create a committee on a decentralized network.

2. The secret election and voting is a process in which agent nodes and candidate nodes collaborate to generate blocks with weights.

3. The block broadcasting is a process in which the nodes in the network preferentially process the blocks with higher weights and suppress the propagation range of the blocks with lower weights.

The secret election and voting process is a process of potential block generation through interactive interaction between candidate nodes and proxy nodes. It is divided into three stages: the preparation stage, the election stage, and the voting stage.

1. Preparation Stage: In each block production cycle, the node first obtains the opportunity value of block production Op corresponding to its own ability value from the VDF of this round.

2. Campaign Stage: The node with a higher Op value sends a bill to the agent and requests the agent to vote, and the node requests the vote becomes a candidate.

3. Voting Stage: After receiving the voting response from the agent, the candidate calculates the weight of its votes and enters the waiting state of block production according to the delayed relay algorithm.

Next, we'll briefly introduce the key concepts involved in the solution:

P_m Value: Each node has a verifiable computing power across the network, which is defined in the application system and can be quickly verified by the nodes within the network. This capacity may be effective storage capacity, bandwidth capacity, or even mortaged tokens. In this scheme, effective storage is used as computing power. The capacity value of node M is noted as P_m.

OP: The node sighs the random number of the whole network to obtain a hash value:

$$H = Sha256(Sign_m(R))$$.

Node M's hash value of the Rth round is recorded as H_{mr} , and the block production opportunity of node M

$$Op_m = \frac{P_m}{\sum_{i=1}^{k} P_i} \div \frac{H_{mr}}{2^{256}}$$:

Where, Op_m is an arbitrary value from 0 to 2^{256}, we can make the following convention, the bigger the Op_m, the higher the probability of the node obtaining the block.

3.5.4. Analysis of Scheme

Favor Consensus has good adaptability, it can be used as an independent chain, or as a Layer2 of other chains, when it is used as a Layer2 of other chains, the committee's election can be conducted on Layer1. The consensus can also support two types of block production, one is that any node with computing power can produce blocks, and the other is that only members of the committee can produce blocks, which can control the scope of the block producers and the range of blocks and transactions broadcast, furthermore, the number of committee members can be reduced to the same number of proxy nodes, further reducing the scope of blocks and broadcasting to achieve higher TPS performance. Implementers can select according to their own performance and security requirements.

Another characteristic of the Favor consensus is its verifiability, since the agents sign the election results and these signatures and weights are placed in the block header, any node can obtain the validity of the block, and if it is finally determined by verifying the signature information in the block header.

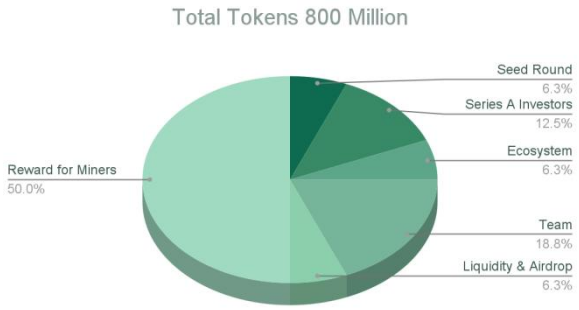
4. Economic Model

In FavorX, applications need to store pay storage fees, pay traffic fees for data distribution, or pay service fees for services. Miners in the system earn revenue by providing storage, bandwidth, or servers. Due to the bandwidth being a core resource in the system, miners will be rewarded for providing bandwidth.

FavorX's ecology will utilize FavX as a fundamental circulating tool, which will be used for: paying for data flow fees, paying for server API invocation fees, or paying for decentralized server fees.

In FavorX, miners' profits will have a 3% return to the platform national treasury, of which 1% will be allocated to the block nodes of the built-in chain, distributed according to each block height, while the rest will be allocated by the platform national treasury according to the strategy agreed by the community, such as for destruction, development, and operation, or additional rewards for miners, etc.

The total circulation of the system's total tokens is 800 million, of which 50% are generated by miners through providing bandwidth and performing data distribution services, while the rest are for early investors and liquidity providers. The distribution and release policy will be determined by the community agreement.



Allocation and Release Strategies

Stage	Tokens(M)	Release Model
Seed Round Investors	50	10% on TGE, then 5% per month
Round A Investors	100	10% on TGE, then 5% per month
ECO System	50	5% on TGE, then 5% permonth
Team	150	10% on TGE, then 5% per month

Liquidity & Airdrop	50	20% on TGE, then 10% per month
Reward for Miners	400	Halves per 1500 days. release coeffiency added

5. Case Studios

5.1. Decentralized Service

FavorX's proxy mechanism for HTTP and Websocket can enable applications to access services in a decentralized way by adding the following steps:

1) Develop the proxy function to forward requests from the application to the server and forward the server response to the application nodes
2) Generate some node instances and deploy them all over the world
3) Form a group with these nodes and configure the group
4) Notify the GroupID of the group to the front end
5) Front end transformation, modify the original access URL to http://localhost:1633/{original URL}, and keep the data format unchanged.

Through the above transformation, the application has the ability to access services in a decentralized manner, and since the services are hidden, it can also prevent attacks on the server.

After the above transformation, the server has the ability to access in a decentralized manner, but it is still decentralized in itself, and it can achieve decentralization of the service itself in another way.

The decentralization of the service itself lies in the decentralization of the database, and the simplest way is to create different replicas sets for the database on different nodes. The operations on the database can be transformed into operation logs, so that when different nodes receive the block information, they can directly transform it into operation logs to update the database on this node.

In order to ensure the consistency of the data, the node can generate a hash value for the local file according to a certain policy. When each block height is reached, the master node needs to generate this hash value, while the replica verifies the correctness of the hash rate.

5.2. Decentralized Metaverse Music Platform

SoundBox is an application for copyright management in the Metaverse. It has the ability to read and play music files in real time from a variety of different data sources -- Spotify, Dropbox, Metaverse's servers, as well as decentralized storage systems such as IPFS and Arweave.

In SoundBox, all music files are licensed for playback, and information about music players, times of play and play duration is also recorded. Player purchases, music licensing, and payment of fees are performed on FavorX.

Since the information is all recorded on the chain and payments are also made on the chain, it is possible to create a Royalty-Bearing NFT market -- FavorX tracks the owners of NFTs and distributes the revenue pro rata to the owners of NFTs when the musics corresponding to the NTFs earn revenue. This approach empowers a whole new concept for music NFTs and will drive sales and marketing of music NFTs.

5.3. Decentralized Video Creation Platform

5.3.1.Main Theme

Inspired by Mastodon and DTube, FavorTube is a decentralized infrastructure-as-a-service platform that supports content sharing, fan base management and patron services.

We believe that a true Web 3.0 platform should embrace the blockchain technology when handling specific type of transactions, but not building the service entirely around the blockchain & smart contract narrative. Peer-to-peer data transactions and distributed storage are deployed without the necessity of visiting any layer of the blockchain.

Note: We are not trying to be the decentralised version of Youtube, Tiktok or Patreon, but rather providing the construction framework and essential toolkits that enables community members to organize and

5.3.2.Core Pillars of FavorTube

- Basic Functions:
  - Decentralized ID to make sure of anonymous
  - Censorship resistant content rendering network mainly based on P2P, rewarded by Favor’s basic token.
  - Permissionless stable coin payment gateway enabling seamless transaction between users

- Advanced Features

  - Studio Sites / Instances Nodes allowing more sophisticated content creator to host its own channel and manages the fan base.
  - User committee that ensures governance over publicly available contents.
  - Algo-based content recommending flows
  - Private-key for decryption of special contents

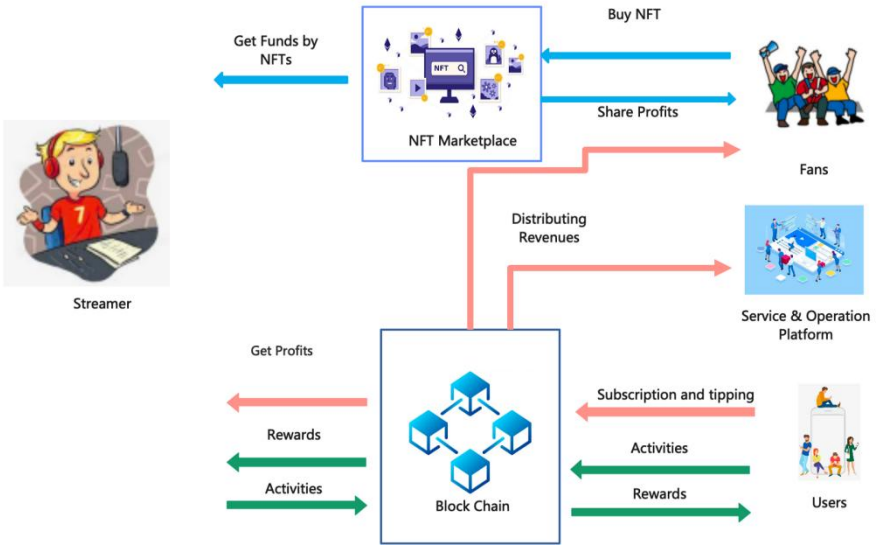
5.3.3. Economy Structure

Bandwidth sharer gets rewarded by Favor native tokens for net contribution to peer users. (Like a mining machine!)

A small portion of user transactions are made mandatory to purchase the native tokens.

5.4. Decentralized Live Streaming

The decentralized technology allows for increased interactions between live streamers and consumers, and enables them to share the revenue brought by growth together



5.4.1. Current Status

Recruitment, early companionship and customer acquisition are currently the most basic pain points in live shows. When a newly recruited live streamer joins, both the streamers and the operator are unsure whether the newcomer can become famous; in the early stage of the broadcast, many fans are reluctant to participate in the interaction due to the insufficient number of fans, leading to low confidence of the streamer and inability to persist; and the operator also needs to get more shares from other streamers to be able to reap the rewards.

5.4.2. Decentralized Operation Strategy

In a decentralized system, NFTs can be used to solve the customer acquisition and streamer recruitment challenges. When recruiting a live streamer, once both parties determine intentions, you can have the streamer create some photos and talent video as an introduction and take a certain percentage of future earnings as the subject matter to create NTFs . It is agreed with the NTF purchasers: When the NFT sales meet a certain base amount, the streamer will start broadcasting; if the sales cannot be completed within the specified period of time, the streamer will not start broadcasting, and the funds used to purchase NTFs will be returned.

In this way, the streamer knows his/her popularity before starting to broadcast, and meanwhile the sales process of NFTs is also the self-presentation and customer acquisition process for the streamer. For the operating platform, risks can also be avoided in advance, and therefore the share of the streamer can also be reduced. For fans, buying NFTs will no longer be just a sentimental purchase, but an investment.

After the streamers start broadcasting, interactions can be rewarded in a decentralized way, which will promote the interactions between users and streamers. For example, each fan has a certain amount of energy every day, and when a fan interacts with an streamer, the energy value can be transferred and distributed to the streamer and generate computing power for him/her. Similarly, each streamer also has an energy value every day, which can be transferred and distributed by the streamer to fans through interactions, generating computing power for the fans and allowing them to be rewarded by the system as well.

The above scheme of rewarding interactions will give fans a chance to earn rewards in the process of interacting with streamers, motivating the desire of fans to interact and also enhancing the confidence and willingness of the streamers to start broadcasting.

5.4.3. Decentralized Platform Strategy

Furthermore, the decentralized system would allow a streamer to own a platform entirely belonged to himself/herself, rather than having a platform on top of an existing one. The advantages for a streamer to have his/her own platform are self-evident: the streamer can have a fan base that is completely owned by himself/herself; the streamer is not subject to the control of third-party platforms and cannot be banned; the streamer doesn’t have to worry about being blocked for a decentralized access; and the streamer can gain revenue from subscriptions or rewards from around the world through the blockchain. As for the original operating platforms, they can earn revenue by providing aggregations, promotions, and search services for the platforms of these streamers, without paying for storage, bandwidth, and customer acquisition costs, and will also have high enough profits under the condition of low sharing.

6. Conclusion

This paper introduces the unified transmission protocol of FavorX's decentralized hybrid data. On the basis of P2P broadcasting transmission in decentralized networks, various data transmission models are discussed and designed to meet the transmission requirements of different data. Based on this, a accounting and reward scheme is designed to motivate miners to voluntarily provide resources for the network to achieve decentralized system construction.

On the basis of the completion of the scheme, this paper discusses the valuable application scenarios based on FavorX, and how to use these application scenarios to realize the killer application of Web3. The standardized service decentralized model allows developers to develop their own Web3 applications in existing ways, tools, and protocol stacks without considering the actual transmission process and implementation of data and messages.

FavorX allows application developers to connect existing various public chains, decentralize storage, or zero-knowledge proof modules, and combine the functions of these modules to provide a unified and complete service for the application.