ISSN No:-2456-2165

Encrypted and Distributed Data Transfer Protocol

Vivek Vaishya
Department of Computer Engineering,
Pillai College of Engineering, New Panvel, India

Abstract:- Distributed Web is a relatively new invention of the Internet. As opposed to the current model followed by the Internet, called Server- Client model, Distributed Web has almost no concept of a Dedicated Server and no one is truly regarded as a Client. Every element is a Peer and thus the name Peer 2 Peer. The model introduced here serves the content in P2P manner and enforces every participant to contribute equally thus by balancing the network. The former concept is derived from the Bittorrent network while the latter is based upon the equality concept of Blockchain. This paper describes how this possible model can be implemented to be the new Distributed Web.

Keywords:- *Cryptocurrency, Blockchain.*

I. INTRODUCTION

The *Blockchain* based technologies like *Ethereum Dapps* and *IPFS*[1] are quite promising and can be seen as a viable future to the current Web but haven't gone into widespread use, yet. The developers require you to first sign up for the service using some existing *Cryptocurrency* and a novice user doesn't want to get into trouble of setting up these *wallets.* Cryptocurrencies like Bitcoin have gained unclean reputation due to legal concerns surrounding them in the past.[2] To build any new technology based on Blockchain, one requires in depth knowledge of the working of the Blockchain itself.[3] While publishing a website on the internet, we generally do not worry about the rendering and distribution. Both of these jobs are responsibilities of Web Hosting services. So developers need only write their blog and not worry about anything else. This model is not possible with the advent of the new Blockchain based *Distributed Internet* or what creators refer to as *Web 3.0.*[4] This paper talks about an alternative approach to achieving this goal of distributed web in widespread use for a wide audience.

The most basic entity of the distributed network is considered to be a peer. A peer is an end node which is contributing to the working of the network. It helps other peers download the data that they already have downloaded. Each node has its own public address (as well as a username) to identify it over the network. An overlay routing scheme using Distributed Hash Table is the backbone of the entire network structure. Any node can join the network any time and depart without affecting the working structure of the network.

When a new node joins the network, it will first request a Bootstrapping Router closest to it, which is a volunteer router always having a static IP address. Every node in the network has a corresponding routing table, called Distributed Hash Table (DHT),[5] which connects it to up to n other nodes. Any node, say A, wants to communicate with any given node, say B, will request the nearest node, say P, for the destination node's IP address. Node P will either give the destination node's IP address or of another possible closest node which might have node B's IP address. For a network with 2^n nodes, at least n nodes will have an IP address of node B provided B is currently active. The Bootstrapping Router is responsible for bootstrapping the DHT routing table of any newly attached node in the network, since its routing table will either be empty or might be outdated.[6][7] The BS Router will give the new node the address of other nodes in the network so the new node will position itself as if it has always been there.

The DDT (Distributed Data Transfer) address assignment is randomly calculated with the aid of BS routers distributed around the world when a new node joins. The node will own an address for a definite period so long as it is advertised as active. After 7 days of going inactive, the address will be reallocated to any other joining node. All the M*essage Chunks* will be reallocated and re-replicated to other nodes on the network to keep failover. This way there is no dependency on a single node in the network.

For a node or user to communicate on the network, it must first identify the receiver node. For this identification purpose, it must first acquire the public username through some other means. No search engine is part of the implementation as of this paper (for there is no central database). Once it obtains the public username, it'll send the request on the network to grab the receiver's public communication key. The intermediate nodes will use overlay routing to acquire the receiver's public communication key. This key is not directly propagated to the sender but the receiver gets a request, whether it wants to communicate with the sender. This request message consists of the sender's public username as well as its public DDT address. This DDT address is used for establishing a connection between the two parties before each communication begins. Receiver, if it approves, will directly send its DDT address to the sender to finalize the connection establishment process. The two nodes will keep these public DDT addresses in their key loggers and will keep them private. The paper suggests that DDT address is only part of DHT routing and is independent of the communication IP addresses which is usually provided by the ISP.

After the address exchange, two nodes can start communicating by exchanging their cryptographic keys for secure communication. For each new *Cryptographic Key Exchange*, both the parties need to pay fees in the form of *APT coins*. This payment will establish a secure session between the two and nodes may communicate thereafter. The protocol provides end to end encryption via a *Distributed Encryption Scheme*.

Protocol can also handle channels as in Telegram Messenger for providing multicast contents. These multicast channels are subscription based online content providers administered by multiple admins. Multicast channels are allocated a predetermined storage initially and this storage increases as the user base increases, though this cannot grow exponentially because *Karma* rules the entire network. Multicast channels can be of two types, public or private. Private Channels act as *WhatsApp* groups while Public Channels deliver subscription based content. The network is equipped with support for broadcast messages which will serve as traditional websites. Hosting a website is a matter of buying a public *DDT Domain* from the network and assigning it an initial supporting node which will serve as genesis nodes for that website. The Genesis nodes can be bought from the network through public trading as all the nodes are willing to share their network capacity. In both the cases, Broadcast or Multicast, Messages will be downloaded from the nearest node cache and will be stored if it becomes a very frequently asked content over the network.

The protocol supports *Live Streaming* of media content as well. For Live Streaming, the media content is first uploaded to the nearest Bootstrapping router and then the content is downloaded by consumer nodes. As soon as a node finishes downloading a Chunk of Message from the Bootstrapping Router, it will broadcast itself as one of the content providers to BS router thus by increasing its credits. BS router will keep a list of all these nodes and will evaluate them based on their latency. Novel joining nodes will be redirected to the node with least latency instead of BS router to consume the streaming content. This structure balances the bandwidth sharing across the network. Irrespective of downloading the Message chunks from any of the nodes, the hash is always downloaded from the main streamer and checked against each chunk. Public streaming has authenticity and non-repudiation because of hashing but doesn't provide confidentiality because of no end to end encryption.

II. PROTOCOL SPECIFICATION

This section focuses on how the network architecture will perform various operations.

A. Components of the Network

i) *Nodes:* These are standard and authentic users sharing their bandwidth and storage on the network. *Authenticity* of the users is guaranteed by their traffic states. Upon entering the network, users can opt for a *Username* which is unique to each node. This Username is a 128 bit *Alphanumeric* text that nodes can opt for themselves. These UserNames serve as a gate for nodes to be identified on the network. This information is stored in a decentralized manner in *Bootstrapping Routers*. An XOR routing table address is also assigned to each user similar to how an IP address is assigned to any communicating node over the internet. This address is called *DDT (Distributed Data Transfer) Address* which is of 160 bits as that in original DHT based trackerless Bittorrent implementation. The only difference is that this address space is different from that of the Message Address space.

ii) Bootstrapping Routers: These are dedicated routers situated at various geographical areas of the world to help the nodes find nodes on the network. The Bootstrapping Routers are needed to avoid Nodes from sending unnecessary broadcast messages on the network and flooding the pipe wasting its bandwidth. Each Bootstrapping Router covers a geographical zone in the world as explained in Section. The Bootstrapping Routers are only needed once the node enters the network either when it is newborn or when it has come from a *Deep Hibernation*. When a node sends the request to a Bootstrapping Router to find another node on the network, it will send it a list of possible nodes which know or are close enough to the destination node. Then it will continue digging in the search using DHT until it gets close enough to the destination.

iii) *Helpers:* These are also special nodes but are contributing their network bandwidth for identifying other nodes on the network and helping other nodes in the authentication process. When two nodes communicate with each other in Unicast mode, they must first exchange a cryptography key in the distributed manner as described in *Section 3*, this requires intermediary nodes, Helpers serve this purpose. Furthermore, these also help in routing *Request Messages* during node search.

iv) *Peers:* These are also ordinary nodes but are contributing their storage for storing temporary request data from a Unicast message or acting as permanent storage for Multicast or Broadcast messages. During a Unicast message, possibility of the receiver node getting down is more than the sender node. In this case, Peers which are possibly closest to the receiver, store the request message in encrypted form for the period until the receiver is back online. When the receiver comes back online, it will broadcast its new state to all the nodes in its DHT routing table, one of which will have the message for the receiving node. There is a limit on the size of the message stored in Peers and also the time for which they remain stored there. These two limits are implementation dependent.

v) *Award for Participation Token (APT):* These are very similar to Cryptocurrencies in Blockchain networks like Bitcoin and Ethereum. Authenticated Nodes are rewarded with these currencies for contributing towards

the network. These rewards can be increased by contributing more network bandwidth and storage and can be used in turn for purchasing personal storage from the network. APT coins can be purchased for real currencies. The value of the currency is directly proportional to the number of active users and inversely proportional to the amount of bandwidth and storage that they are contributing. This value fluctuation is created to ensure trust in the reward system and no single authority can accumulate a very high amount of these APT coins.

vi) *Messages:* Any data sent over the network is considered a message. In a Unicast domain, these messages are encrypted using *4 Layered Symmetric Encryption* methodology while messages in the Multicast domain are protected through *Hash Key Matching* method. This is further explained in detail in Section 3. Messages in the broadcast domain are publicly available and can be viewed or downloaded depending upon the operator's policies. Each public Message,

B. The relationship among these components

i) *Zone Distribution:* The entire network infrastructure is divided into Zones of a finite number of nodes governed by the Bootstrapping Routers. These routers aid in easy identification and searching of end nodes on the network when a new node joins the network or is coming back online after a *Routing Table Expiration* time. Zones are mostly based on geographical area each of which can only contain a maximum of 1024 nodes. The number is implementation dependent and can be changed based on the infrastructure provided by the libraries and the security parameters available at the lower layers of the network (below application layer).

ii) *Bootstrapping:* When a new node enters the network, after the sign up stage, it requests to join one of the zones from the nearest Bootstrapping Router. The Router will add this node after confirming its verification and will announce its presence in the network. The zone area may increase or decrease depending upon the number of Bootstrapping Routers available in the area and amount of routing information sent per unit time.

iii) *Routing:* The network uses Overlay Network routing scheme for peer to peer communication. The overlay network finds the target node via XOR operations. Explaining the working of Overlay routing is beyond the scope of this paper. We are following the very identical routing scheme as that is followed in *Bittorrent* networks.[2]

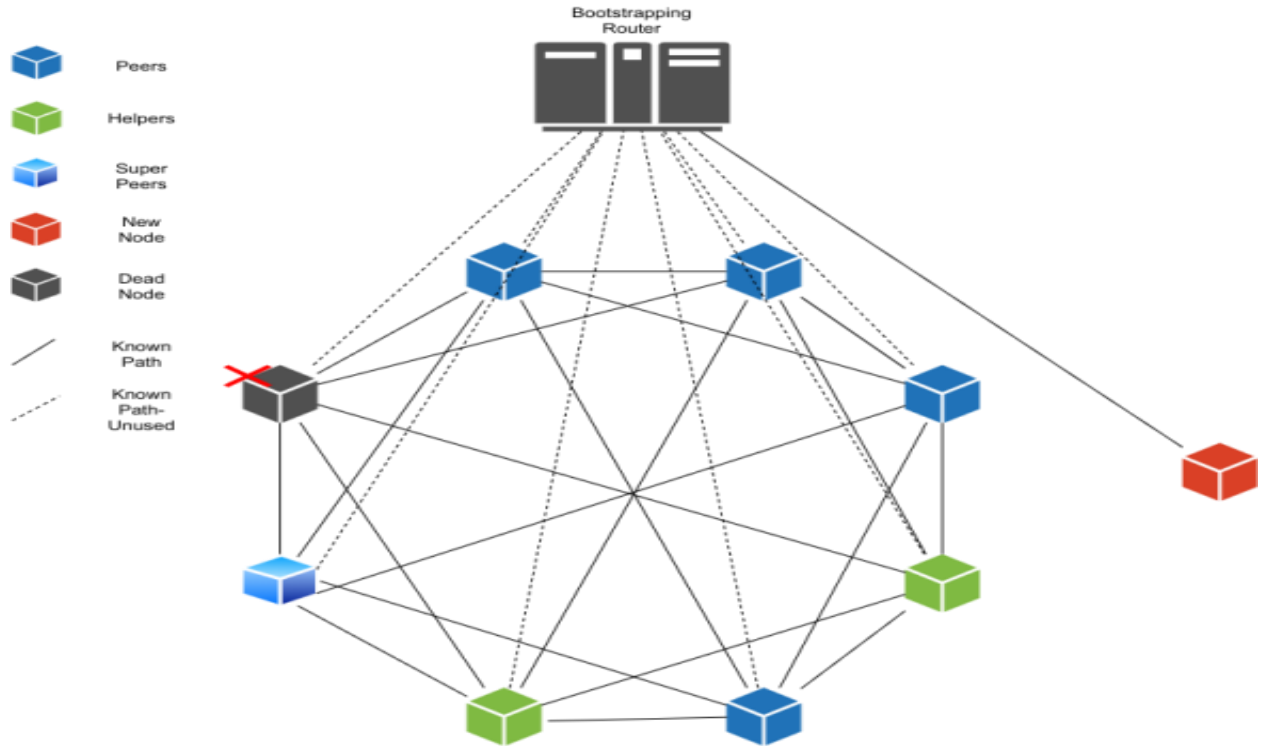

Fig. 1 A DDT network with 8 nodes of capacity 672 (+ 128) MB, Peers contributing 128MB storage and Helpers contributing 16MB, each having average network bandwidth of 1MBPS

C. Connection Establishment

i) Unicast Domain: After a node has joined the network, it must follow a tradition to establish a temporary connection between the receiver and itself in order to be able to transfer any message. In order to establish that temporary connection, it must follow the overlay network concept that we discussed in the previous section.

Joining the network doesn't mean one can contact anyone else in the network directly. They must first agree upon their connection to make message transfers.

ISSN No:-2456-2165

```
SendFriendRequest( Request):
# Request is a dictionary containing following key: value pairs
# [ username: (username of userA), address: (DDT Address of userA), targetname: (username of userB) ]
While( True) :
# Find closest peer to userB from the local DHT routing table, say it is peer
  If( peer != Request.userB):
    SendRequest( peer, Request)
    WaitForResult( newPeer)
    peer = newPeer
  else:
    returnedMsg = WaitForConfirmation()
    If( returnedMsg.grant == accepted):
      AddToRoutingTable(returnedMsg.userInfo)
      InitiateConnectionEstablishment( returnedMsg.userInfo)
    else:
      Error()
```

After becoming friends with the recipient, they both must exchange their security keys as explained in section *Encryption*. For every new session, there's a need of establishing a secure pathway between the two communicating nodes. Since Unicast messages are always encrypted, we do not worry about compromising data at the lower layers of the network. Messages are directly exchanged between the two nodes without any intermediary node once the two nodes have become aware of their public IP addresses.

Initiating a Chat is a matter of collecting a Communication Token from the network in exchange for a promise that you'll share your bandwidth and storage for the growth of the network. This promise is collected in the form of APT coins. Collecting a Communication Token simply indicates that you're interested in message transfer with someone in the network and so your request should be queued now. This Communication Token is only needed at the sender side and is valid for the entire session of the sender. In this one session, the sender may send messages to multiple recipients. Further, Unicast messages are never stored on the network but in vaults of the participating parties. Participating parties may opt to buy storage from the network to backup their messages in the network.

```
InitiateConnectionEstablishment( UserInfo):
Let r = rand( 0, 4)
for i = 1 to 4:
  key, conMsg, user = GenerateConnectionMessage( UserInfo, r)
  StoreInKeyring( key, r)
  SendConnectionMessage( conmsg, user)
  r = ( r + 1) % 4
```

Both parties will know each other and have securely established a connection between each other. Following the sequence of keys in Keyring, messages are encrypted between the two nodes.

ii) Multicast Domain: Basic Communicating entities in Multicast Domain are called Channels. These channels are usually subscription based service providers which require a node special permission to join. Establishing a connection in Unicast domain in an encrypted form is a must as it needs end to end encryption. On the other hand, Channels do not require this encryption. So connection establishment is pretty straightforward in this case.

Incorporating a Channel- As channels will require permanent storage from the network, they must give a proof that they will contribute more bandwidth as well as storage towards the network. Depending upon how big the channel is planned to be, the creator must associate those many admin nodes with the channel (Since each node has a limit to how much it will contribute at once). Hence the most basic requirement for creating a channel is to provide more bandwidth towards the growth of the network and provide as much storage to the network as the content provided by the channel in addition to the basic tier.

We propose two types of channels-
Public Channels- As long as a user knows the channel ID, he can directly join the channel.

Private Channels- A single point of authority exist, namely administrator, regulating who can join the channel. The administrator may be a single person but all the associated nodes will be able to act as A*dmin Nodes.*

In the latter case, connection establishment is similar to making friends in Unicast domain. However friend requests in Public channels are automatically accepted.

Messages generated by channels are replicated across networks in multiples of 2, with every other frequency increment resulting in another replication. The 2 copies result in always availability of messages thus by increasing better data redundancy. New replication factor allows the network to give more importance to that message which has high demand. Replicas may get automatically reduced over time as the demand decreases.

Channels are meant to provide facilities provided by Bittorrent protocols. So it is quite expected that while downloading data from channels, the network will work more like Bittorrent network. The data integrity is a major concern in any kind of network and hence it is solved using Hashing.

iv) *Broadcast Domain:* Broadcast domains are websites whose contents are publicly available and their addresses are reserved. With distributed website caching, it is much faster to load a website than traditional Server- Client models. They can pay certain nodes on the network to keep better cache for faster website loading and downloading files. Every node on the network in addition to Bootstrapping nodes act as DNS resolver so the probability of a single point of failure is almost negligible.

ISSN No:-2456-2165

D. Message Transfer

Every single chunk of data transferred on the network is considered to be *a Message*. A message can be a file transferred in the form of an executable, a music, a video, a document, or even a chat text written by an individual. The differentiating feature of these Messages is how they are treated under different circumstances.

i) *Size:* The size of the Message is one of the biggest factors affecting how they are stored in the network. The standard defines the smallest unit of a Message as *Chunk* which has a size of 4MBytes [Implementation Dependent]. Whenever messages are transferred over the network, they are sent in Chunks to not over burden the network and avoid packet losses due to error in sequencing. During transfer, each Chunk is sequenced and encrypted, if needed, before transmitting over the network. If the size of a message is too big, it will be divided into multiple Chunks and transmitted or received individually (as dictated by Bittorrent protocol).

ii) Frequency of usage- Each message on the network is counted for its downloads across the network. For a Message which is frequently being downloaded, it will be cached on more nodes than a Message with less usage statistics. The Message caching decreases over time when the number of downloads in the network decreases. Again, each node is allowed to configure themselves to store only caches for those files which are older than a certain time in the network. Nodes can also dictate what kind of files they want to serve. Although nodes can control their individual statistics, they are not allowed to modify the cache since all the Message Chunks are hashed and the *Client Side Front-end Application* will first check the hash with other nodes before accepting a file as correct.

iii) Paid Promotion- If a Message is promoted by buying more priority from the network, it will automatically be replicated more often than a Message of less priority. The *Priority Purchasing* is implementation dependent but the paper governs that no message be given more priority than its usage frequency over the network after a certain period of time. Websites can buy more priority to have more availability in the network and faster load time.

iv) Type of Domain- For a message in Unicast domain, no more than a single replica is made other than the receiver and sender node. This is to guarantee safe and trusted message transfer over the network. For a message in a Multicast or Broadcast domain, at least 3 replicas are made at the time of creation. As the other conditions dictate, more replicas may be created over the network for individual Chunks. With each new download in later two cases, more replicas are automatically created and remain in the network until other conditions don't falsify.

Further, in a Unicast system, any message is always only stored at the two participating nodes' ends and never elsewhere. In case, if the receiver is unable to receive the message at the moment, a request will be sent in the network for the communication but the actual message will never be released. This request is stored in few of the network nodes that the sender assumes are direct neighbours to the receiver. As soon as the receiver indicates its active status by broadcasting to its direct neighbour nodes (ones in its routing table), the intermediary nodes, which have stored the previous request, will send it to the receiver. Receiver will then ask the sender to send the message. If the sender is offline, the process will continue.

E. Live Streaming

Supporting Live streaming of media content is always the most challenging aspect of any protocol. In a live environment, each moment is crucial and cannot be missed and thus a time critical aspect of Message routing is a necessary feature of any protocol. In other protocols, this is also regarded as Online Meeting, Conference Call or E-Lecture. Hence Live streaming is amended as one of the most integral parts of the EDDT protocol. Live Streaming feature is implemented in a Distributed manner. During a conference, Chunks of data are transmitted to all those nodes which are listening to the sender. While in traditional conferences, this requires an intermediary in the form of a dedicated server which will receive the Chunks of compressed Video data, possibly store it and then redistribute it to all those listening to the sender node. In this method, there is no possible way to ensure the reliability of dedicated servers since those are always closed source nodes. The method that we propose requires the intermediary only for initiation of the connection and then the communicating nodes take over the responsibility of handling the routing of these data Chunks.

When a Conference starts, after the usual connection key exchange procedure, the first sender node will identify the nearest Bootstrapping Router and will send all the receiver its address (IP). The first few packets are delivered to the Bootstrapping Router and then this router will redistribute these packets to all those requesting it. The protocol puts a restriction on how much bandwidth this Bootstrapping router will provide for each individual Conference. Since acquiring bandwidth from the network always costs APT, there is an absolute need to minimize it. To achieve so, participating parties have following options,

1. Pay more APTs and keep using Bootstrapping Router as intermediary.
2. Lower the quality of conference data thus by reducing the amount of APT paid.
3. Disconnect from Bootstrapping router and initiate conference in Peer 2 Peer mode.

While the first 2 methods sound interesting, they are only suitable for corporations which can pay for high bandwidth to the network. As discussed in the other sections of this paper, any ordinary node other than Bootstrapping nodes may also participate as an intermediary to earn more APTs and balance the network. To explain this, we expand the third method to discuss in general.

ISSN No:-2456-2165

There is a need to highlight why Bootstrapping Router as an intermediary node is so important. When a conference starts, none of the participating nodes have any idea of the network capacity of either of the other nodes. Network capacity includes precious download and upload speed as well *Relative Latency* for communicating with a particular node. Thus there is a need to calculate these and so in the conference initiation phase, Bootstrapping Router will calculate all the required attributes and will update the participating nodes regarding the same. After the knowledge of all the required attributes, participating nodes may continue conference in Peer 2 Peer chain.

In a P2P or distributed conference, not all nodes are equipped with identical networking equipment and thus an intelligent algorithm governs the message flow among all these nodes. This is explained using the next generalized algorithm.

SelectBestReceiverNode(NodeList):
NodeList is a list of all the Node objects communicating on the conference.
These Node objects contain all the network routing details such as network capacity details.
This unimplemented algorithm finds the node which has
1. Least overall relative latency among the data transmission between all the nodes.
2. Highest average as well as *Divided Download and Upload* Speed

The selected node will be sent the subsequent Chunks of Messages in following seconds and the redistribution of those chunks will happen from that node. It is yet unclear the performance of this structure as nothing is tested yet.

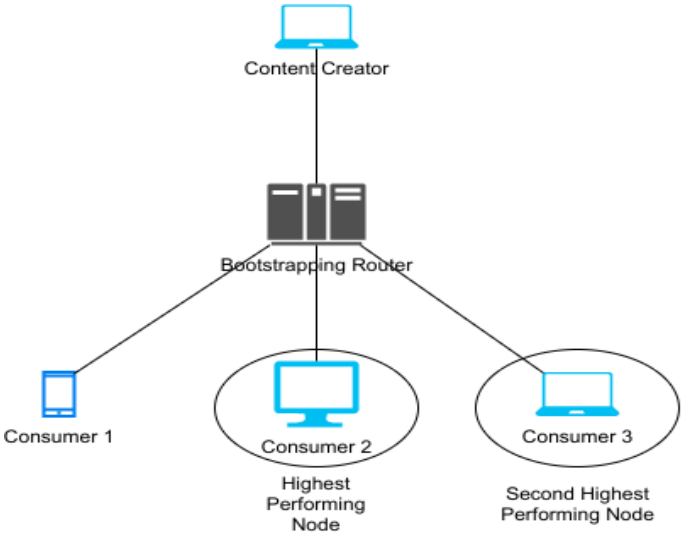

Fig. 2 Initiating Connection

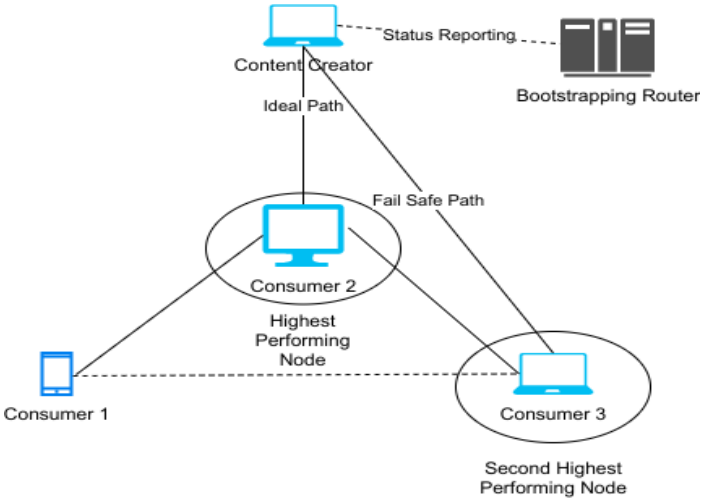

Fig. 3 Finding new Intermediary

F. Cryptocurrency Coin Mining or APT Coin Mining

For staying in the network, you're supposed to contribute towards the growth of the network. This contribution can come from either giving bandwidth or storage or both. In return, the contributor gets APT as a reward. Unlike Blockchain networks, like Bitcoin and Ethereum where miners are rewarded only when they contribute their computational power to the network, DDT dictates node participation and are rewarded for their networking capacity.

Upon signing up for the network service, each user is constrained to contribute a fair amount of storage and corresponding network bandwidth. This is a bare minimum amount of contribution needed to be part of the network. Failing to do so will result in loss of credit and eventually, they will be kicked out of the network. Beyond this storage limit, any further contribution will result in rewards of APT. This paper dictates that a Node must contribute 64MB to 128MB storage per 24 hour to have a valid credit for being part of the network. These criteria may change depending upon the overall network transactions. With each Chunk of Message replica stored beyond this capacity and a half of its network bandwidth, a node is credited with an APT coin. When uploading a Message over the network, a node is required to pay the same amount of APT coin to the network. In a Unicast domain, no APT coin beyond Connection Establishment per friend is needed. This promise dictates until a message is not forwarded. Which require a tiny portion of APT coin. Again, every time a connection is lost to a friend, there arises a need to re-establish the connection and hence new payment. The amount paid is dictated by the implementation.

In a Broadcast domain, website admins are needed to pay for hosting their content. This payment is received in terms of APT coins. Network will offer each Node of its network to host a particular site's content. In return, nodes will be awarded with APT which can be later used to make certain e-commerce purchases over the network. Dictated by conditions explained in the Message *Transfer* section, not all the nodes on the network may opt for hosting a site's content and not one single node is allowed to host beyond a

limit of content. This policy ensures that the entire infrastructure is fairly and equally distributed and no single point of authority dictates the entire network.

III. ENCRYPTION

A. Distributed Chained Encryption Scheme

The traditional encryption methods require only two parties, sender and receiver, to agree upon some predefined credentials to communicate in a secure and encrypted fashion. The RSA algorithm,[6] uses asymmetric key cryptography which requires two keys, Public Key for encryption and corresponding Private Key for decryption. There also exists Diffie Hellman Key exchange[7] algorithm which, in essence, is an asymmetric key cryptography technique but the same shared key is used on both the sides for encryption and decryption. These two encryption methods have received lots of criticism time to time. While the former one is pretty secure, it requires lots of computation power or in other words Time and Memory for encryption and decryption. The later one is faster but may, theoretically, suffer from Man in the Middle attacks. Hence we require a better approach for end to end encryption. We propose a new encryption method to be used for securing the communication chain, the Distributed Encryption Scheme.

This is a hybrid key exchange mechanism utilizing the best of RSA and DH key exchange algorithms. The requirement is sender, receiver and a number of Intermediary Transaction Resolvers. The ITR are selected at random from a pull of other nodes available on the network. Suppose Node A and Node B are two users on the network willing to communicate.

We use *10/3 Intermediary* scheme, which requires 3 intermediary nodes divided into 3 chains, while recipient is the 10^{th} node itself.

1. All nodes in the network generate their Public Key using RSA-2048. This key is available on the public transactional ledger and can be obtained using a DHT request.
2. While establishing a connection for the first time or after the encryption expiration, algorithm *GenerateConnectionMessage* will be called.
3. Each node, while serving in the network, may receive several kinds of requests. One of which is a *ConnectionEstablish* request.
4. Based on the result of *ConnectionEstablish* request, client may call *CompleteConnectionEstablishment* method or follow *ResendConnectionMessage*.

```
GenerateConnectionMessage( UserB, seq):
# UserB is a dictionary containing receiver's info Hash such as his username.
# seq is the sequence Number for the current key's usage sequence for encryption.
```

Let *conmsg* be a tuple of 4096 bits. This bit sequence is taken randomly.

Let *ranvec* be a vector of size 4.

Generate a key *k* of size 128 bits on random. This is one of the 3 keys used for symmetric encryption.

Let *msg* be a tuple initialized as,

```
  msg[0] = 1
  msg[1, 128] = k
  msg[129, 2] = seq

Repeat 4 times:
  While(True):
    r = rand(0, 3944)
    If( contains(rv, r) == False)
      Putmsg( msg, conmsg, r)
      append(rv, r)
      If(i == 1):
        m = UserB
      else
        m = getMediary()

      Encrypt( RSA, conmsg[ 0, 4083], m.PublicRsaKey)
      msg[0] = 0
      msg[1, 128] = m.UserName
      msg[129, 140] = r
      Break

conmsg[4084, 4095] = r
Return k, conmsg, m
```

The returned *conmsg* tuple will be sent to node *m*. Upon receiving this request, he will call *ConnectionEstablish* method. The key *k* will be stored in the user's keyring with the current sequence number *seq.*

```
ConnectionEstablish( conmsg):
pos = conmsg[8084, 8095]
Let pr be the current user's private key for RSA.

Decrypt( RSA, conmsg[0, 8083], pr)

forme = conmsg[pos]

If( forme == 1):
  CompleteConnectionEstablishment(conmsg, pos)

Else:
  nextnode = conmsg[ pos + 1, 128]
  nextpos = conmsg[ pos + 129, 12]
  conmsg[4084, 4095] = nextpos
  SendConnectionMessage( conmsg, nextnode)
```

ISSN No:-2456-2165

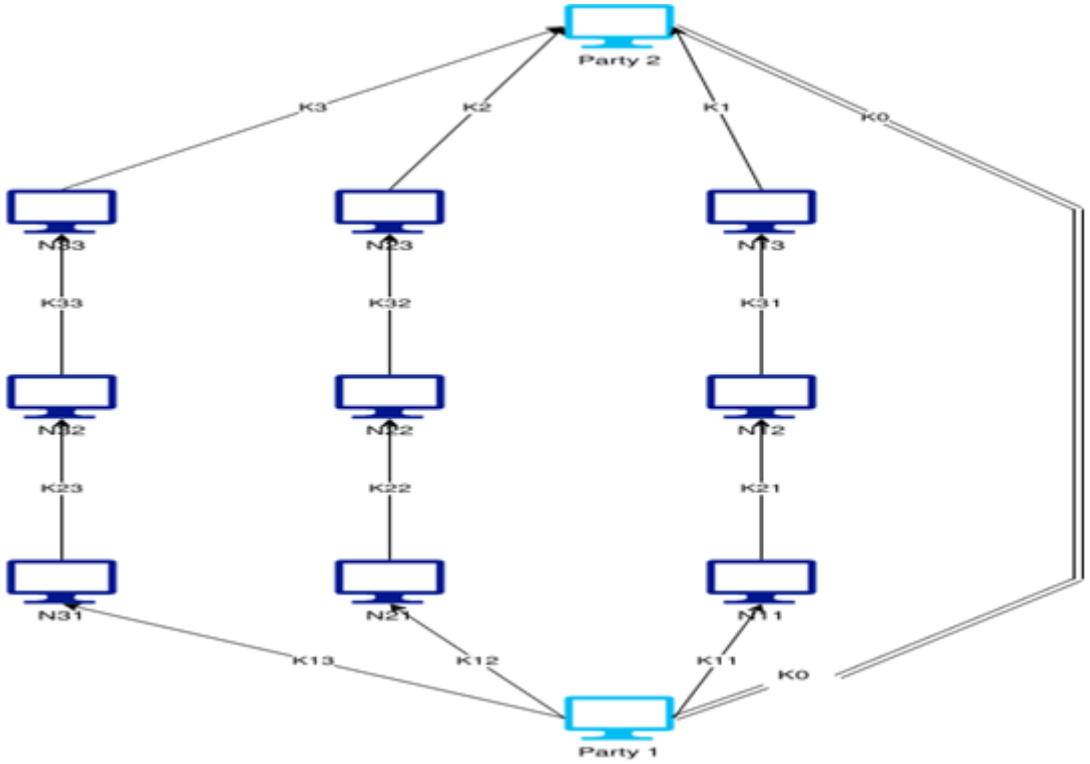

Fig. 4 Chained Encryption

CompleteConnectionEstablishment**(**conmsg, pos) :

key = conmsg[pos + 1, 128]
seq = conmsg[pos + 129, 2]
StoreInKeyring(key, seq)

SendConnectionMessage**(** conmsg, node):

hashId = ResolveUsername(node.Username)
ip = FindInDHT(hashId)
Send(conmsg, ip)

The chained structure was chosen so as to keep user anonymity on the network. It is the same method employed in Onion Routing for keeping the identity of the end user safe. Since any node is not aware of the origin and final destination of the encryption messages, they cannot track the sender or receiver. Though the sender knows the intermediary node's Public addresses, they cannot track them either as the intermediaries are always selected on random. While this method is not so intuitive, it shows how simply the encryption can be achieved without much computation power.

IV. EVENT DRIVING

With an expected network infrastructure to be as big as the internet today, Events must be drought sequentially. This was decided to avoid certain mistakes made during the creation of Web 2.0. With *Event Driving*, every single public event will be marked by a sequence number. Since the entire network is distributed, deriving the next sequence number might be time consuming, this sequence number is obtained from the nearest Zonal Bootstrapping Router. The sequence numbers are marked after the Bootstrapping Router's ID so every event in an area is closely related to the Bootstrapping Router of that area. This ledger is shared with other Bootstrapping nodes after a time limit to make a Blockchain like *Public Transactional Ledger* of events.

Events are always ordered by sequence numbers. Since Unicast messages are never publicly stored, they don't need any public transactional ledger. All the events in Unicast messages are always numbered privately and thus reside in the local system.

V. USAGE SPECIFICATION

A. Entering In

A simple Sign up process is needed to enter in the network and to verify the authenticity of the user. This process creates an account for that user which is linked with the device itself. A unique and system generated ID is provided at sign up which is essential to be identified in the network. This ID is used in overlay routing. Apart from this DDT address, a user may opt to choose a user-friendly Username for him. This username is used by web crawlers to help someone find a user. There exists a Username to DDT Address resolver system but the converse is not true. This ensures any reverse lookup and User safety. It is adamant that in no future implementations, the Reverse Lookup will be provided or endorsed anyhow. After signing into the network, the device will broadcast its unique ID to the nearest Bootstrapping Router. It will add the new node with all the announced details and will announce these details in the same zone immediately. Soon after, every node would have updated its routing table.

User is not allowed to send any message just right away since it has yet not provided the proof of network capacity. Unless a user contributes his part of the daily network capacity, it will remain a new node. With every new transaction taking place with the help of this new node, its credit increases. The more the credit of the user, the higher the priority of its messages will be. Credit is based on how much help that node is providing to the acceptance of other transactions on the network. This model ensures equal participation of everyone or in other words the Karma value.

B. Continue to Live

Each node is equal in the entire network. A Bootstrapping Router is also a node with a fixed IP address. The Overlay routing is similar to the implementation of Bittorrent protocol. Moreover, while implementing, it is advised to use existing Bittorrent DHT implementation to aid in the development process.

A few suggested changes are in how long a node is allowed to stay offline before it is considered dead. If a node doesn't receive any request from a particular node for 36 hours, it will mark it inactive. So the non- responding node will have to reestablish the connection with the network. This is to ensure that no dead nodes are marked as active in the network and to avoid storing any messages for those dead nodes in the network storage.

C. Departing

One can simply depart from the network without notifying as network balances itself within 36 hours as if that node was never there.

VI. IMPLEMENTATION

The proposal is planned to be implemented as drop in replacement for existing Bittorrent implementations while providing a Blockchain like feature set. Hence it is planned to implement the corresponding libraries like DHT using an existing Bittorrent DHT library and for encryption as well. This proposal will be implemented first as a library in JavaScript and run on an existing Browser's JavaScript engine. Corresponding app or plugin, mostly in the form of a native Web Browser cum Social Media app for various platforms will be launched. On Android and iOS, the target is to make a native app. On desktops, the protocol will be enabled via browser plugins so that the multicast and broadcast features of the protocol can easily be used.

VII. QUESTIONS

1. Why am I not using the Bittorrent swarm to inject this new structure?

Existing clients can't resolve encryption requests or serve the purpose of being intermediary since they are not equipped with libraries needed for encryption or decryption.

ACKNOWLEDGEMENT

I would like to acknowledge my mentor Prof. Dhiraj Amin of Pillai College of Engineering, New Panvel for helping me write this paper and being so supportive throughout the work. I would also like to thank Prof. Sagar Kulkarani of Pillai College of Engineering, New Panvel for reviewing the paper.

I would also like to acknowledge all the developers of the Blockchain/ Bitcoin and Bittorrent protocol. This paper would have never been though without those developers.

REFERENCES

[1]. State of the DApps. [Online]. Available: https://www.stateofthedapps.com/, Last access on
[2]. Legality of Bitcoin by country or territory - Wikipedia. [Online] Available: https://en.wikipedia.org/wiki/Legality_of_bitcoin_by_country_or_territory
[3]. The Ultimate Ethereum Dapp Tutorial. [Online]. Available: https://www.dappuniversity.com/articles/the-ultimate-ethereum-dapp-tutorial
[4]. Mat Zago, Why the Web 3.0 Matters and you should know about it. [Online]. Available:https://medium.com/@matteozago/why-the-web-3-0-matters-and-you-should-know-about-it-a5851d63c949
[5]. Distributed Hash Table - Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Distributed_hash_table
[6]. DHT Protocol – Bittorrent http://www.bittorrent.org/beps/bep_0005.htm
[7]. Simple basic explanation of a Distributed Hash Table (DHT) https://stackoverflow.com/questions/144360/simple-basic-explanation-of-a-distributed-hash-table-dht
[8]. RSA - Wikipedia. [Online]. https://en.wikipedia.org/wiki/RSA_(cryptosystem)
[9]. Diffie Hellman - Wikipedia. [Online]. https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange
[10]. Bittorrent DHT (Suggested Library) https://github.com/webtorrent/bittorrent-dht