Node Communication Protocol

1.1. P2P Sync Algorithm:

2. Nodes first start in either as a full **node** or **client node.**
3. Nodes read known peers from the config file. At least one peer node (i.e., bootstrap node) must be configured.
4. Nodes then query bootstrap nodes to retrieve other known peers list, and then updates its own list
5. If a node is set as a **full node** (name might change) It also listens in a port, when a request to get peer list is received, it responds with its known peer list.

1.2. Message Format:

1. The full message format:

   {

   Code: uint64, // message type

   Size: uint32, // size of the payload

   Payload: []byte,

   ReceivedAt: Time,

   Name: string,

   Version: uint,

   meterCode: uint64, // message code for egress metering

   meterSize: uint32, // compressed message size for ingress metering

   }

2. A node, let's say **A**, sends a peer sync request with the following json format to its known peer list.

   {
   "Code": 0,
   "mode": "full-node"
   }

   Type 0 indicating a peer sync request. The key "mode" indicates if it's a full node or not.

3. The peer receiving the request, let's say **B**, will then respond with message of following format

   {

```
"type": 1,
"peers": [{
                "ip":"1.1.1.1",
                "port": "5050"
        }]
}
```

Type 1 indicating peer sync response.
Peers field contains information about the list of peers it knows about. If the node **A** is a full node, in this case it is, the responder adds the requester, i.e., node **A**, at its own **known-peers.json** file.

4. The node **A** will then update its local **known-peers.json** file.
5. After a certain amount of time, node **A** again queries all of its known updated peers, and the process continues.

1.3. P2P sync Algorithm (UDP hole punching included/Not implemented yet.)

NOTE: This is not the UDP hole punching protocol itself, rather a modified implementation of it. It should be easy to implement this protocol with minimum modification to existing p2p protocol. That being said, we have to add additional fields and update communication protocols.

1. Let us assume 2 hosts that want to communicate **A** (**full-node**) and **B(client-node)** which are both in a private network. Let us assume a bootstrap node server **S**, globally accessible with public IP Address. Let the NAT Router for A **Na** and for B **Nb**.
2. **A** sends a UDP packet to **S**. Data format:

```
{
        "type": 0,
        “mode”: “full-node”,
        “id”: “<globally-unique-address-like-wallet-address>”,
        “ip”: “<ip-address>”,
        “port”: <listening_port_number>
}
```

- An extra value added **ID** is added here. **A** must discover its public IP, if it is hosted in a public server, that's no issue, can use open source services.
- **S** sees that the node is running as **“full-node”**, **S** then checks if **ID** has previously recorded IP to Port mapping, If the ip has changed, it updates it. else the S records the public IP and port as a new known-peer.
- **We assume that the NAT router will map the IP-Port to A**. Let's say the IP and port is 1.1.1.1:9090 for now.
- If A does not reply, **A** assumes that **Na** has experienced a port collision and **S** could not communicate with **A**, so it listens to another random port, and tries to communicate again.

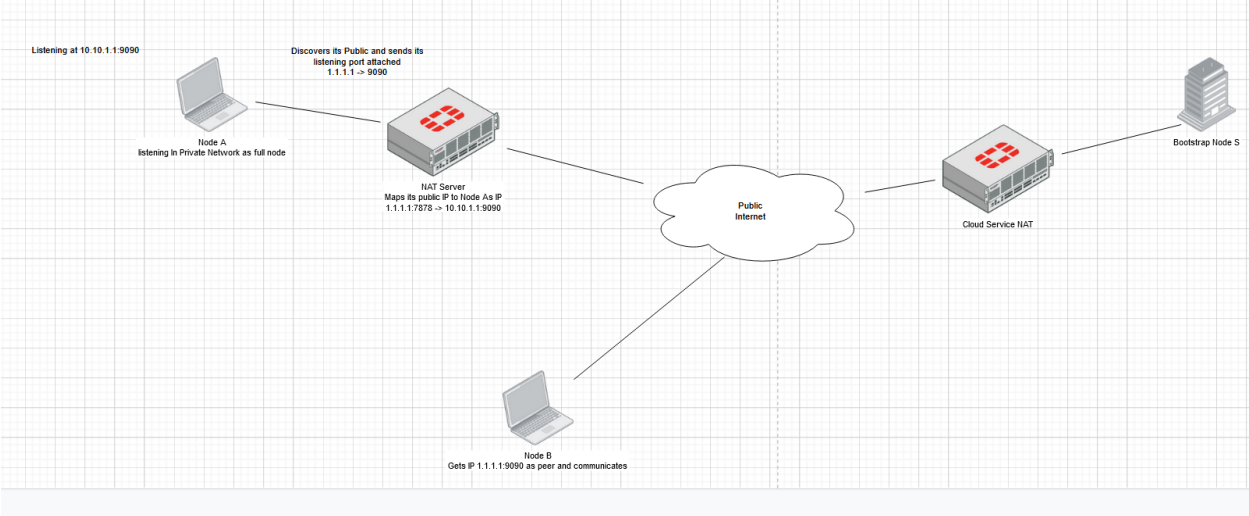
3. **S** responds with the usual peer list (If it could communicate).
4. **B** being a client-node, requests peer-list to **S**, **B** receives peer-list along with **A**.
5. If **Na** has the IP and port still mapped in its memory, **B** should now be able to communicate with **A**.

6. … Additional Block sync steps may happen here …
7. If **B** could not communicate with **A** (tries five times?)**,** it sends following message to all of its peers

```
{
        "type": -1,
        “mode”: “client-node”,
        “hosts”: [{
                “id”: “<globally-unique-address-like-wallet-address>”,
                “ip”: “<ip-address>”,
                “port”: <port_nuumber>
        }]
}
```

   type=-1 indicating dead hosts.

8. After receiving such a message, all of the hosts, including **S**, tries to communicate with **A**.
9. If the rest of the network can not communicate with **A**, **A** is dropped from their individual peer list as well, else the message is ignored.
10. **Pros:** Pretty simple to implement.
11. **Cons:** Port collision problems may occur, keep alive packets have to be sent. As referenced it is not supposed to work on [Restricted cone NATs] or [Symmetric NATs].



12. References:
    https://en.wikipedia.org/wiki/UDP_hole_punching
    https://www.youtube.com/watch?v=s_-UCmuiYW8

DHT, how the distance between two nodes is calculated? to find the closest node i mean? GUID used in papers, but how the GUID is calculated?

It seems most blockchain are built on unstructured overlays of peers, so might want to implement this later?

Articles; https://en.bitcoinwiki.org/wiki/Distributed_hash_table

classical: https://jenkov.com/tutorials/p2p/chord.html

https://en.bitcoinwiki.org/wiki/Kademlia

2.1. Block Sync Algorithm

// update here same as above