P2P NETWORK
And
HOW IT WORKS

Prepared By:
Melih Yesilyurt

Contents

1. What is Peer to Peer Network?

What does P2P do?

In its simplest form, a peer-to-peer (P2P)[2] network is created when two or more PCs are connected and share resources without going through a separate server computer. A P2P network can be an temporary connection, a couple of computers connected via a Universal Serial Bus to transfer files. A P2P network also can be a permanent infrastructure that links a half-dozen computers in a small office over copper wires. Or a P2P network can be a network on a much grander scale in which special protocols and applications set up direct relationships among users over the Internet.

Peers can make some of their own resources, such as processing power, disk storage, or network bandwidth, directly available to other network participants, without the need for central coordination by servers or fixed computers. In contrast to the traditional client-server model, where only servers are suppliers and clients are consumers, peers are both suppliers and consumers. P2P networks are one of the most affordable methods of distributing content because they use the bandwidth of peers, not the bandwidth of the content's creator. There are some basic challenges all P2P networks need to address. These challenges are:

- Connectivity
- Addressability
- Routability

Who is responsible for managing the network?

It is important to note that although the P2P architecture is inherently distributed, there are also levels of decentralization. So not all P2P networks are decentralized.

In fact, most systems need a central authority to direct network movements, which makes them centralized to some degree. For example, some P2P file sharing systems allow users to search and download files from other users, but these users cannot be involved in other processes such as handling search queries.

In addition, it can be said that small networks controlled by a limited number of user bases with common goals have a high degree of centrality even though they do not have a central network infrastructure.

Unstructured P2P networks

- Unstructured P2P networks do not have a specific node organization. Participants communicate randomly with each other.
- Unstructured P2P networks are easier to build, they require higher CPU and memory

Structured P2P networks

- Structured P2P networks have an organized architecture.
- Although structured networks are more efficient, they tend to be more centralized and often require higher installation and maintenance costs.

Hybrid P2P networks

- Hybrid P2P networks combine the traditional client-server model with some features of peer-to-peer architecture.
- Compared to the other two types, hybrid models have better overall performance.

Why are peer-to-peer networks useful?

- It's hard to take them down. Even if one of the peers is shut down, the others are still operating and communicating. For a P2P (peer-to-peer) network to stop working, you have to close down all its peers.
- Peer-to-peer networks are incredibly scalable. Adding new peers is easy as you don't need to do any central configuration on a central server.
- When it comes to file-sharing, the larger a peer-to-peer network is, the faster it is. Having the same file stored on many of the peers in a P2P network means that when someone needs to download it, the file is downloaded from multiple locations simultaneously.

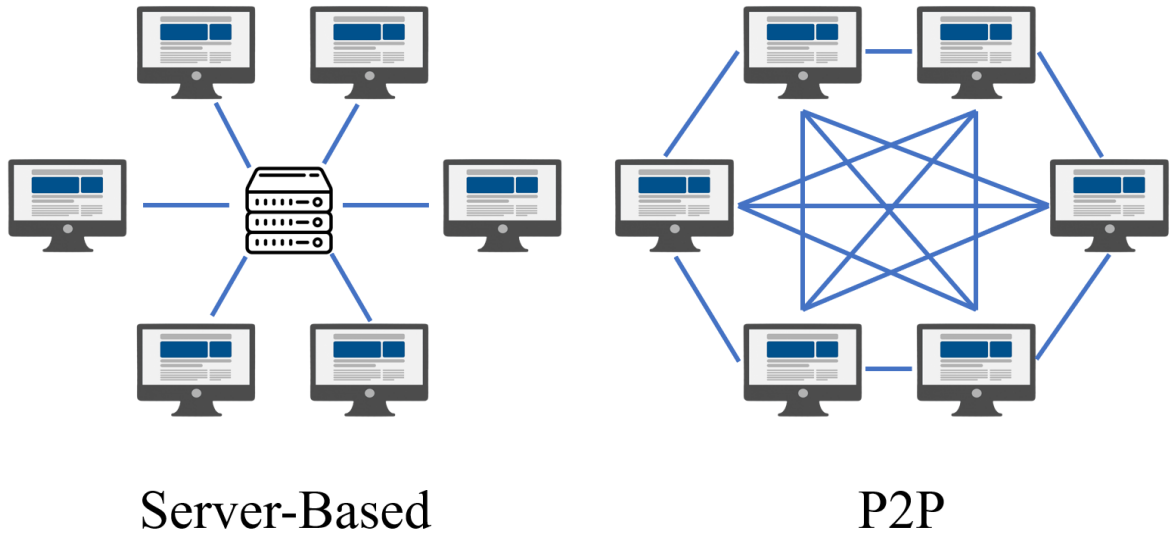

Image 1: Comparison of P2P and Server-Based Network

Is P2P Safe?

In P2P networks, computers connect with each other. They do not connect with the central server. They use various protocols in their connection with each other what they do. Many service providers operate using a central server because they centralize all data and are easier to control. But there are also disadvantages, for example, if more than one user tries to connect to a computer, a crash will occur. This can also happen in a DDoS attack. In P2P networks, this does not happen because there is no concept of server and each computer establishes a connection between itself. In P2P networks, data continues to be stored and protected by the majority of the network if any computer is hacked or has technical problems.

P2P Network Algorithms

The most commonly known P2P network algorithms are:

- Chord
- Kademlia
- Tapestry
- Pastry

All of the P2P algorithms mentioned above are similar to each other. The logic of all of them is that their peers are organized into a single virtual ring, with each peer holding a routing table that points to a subset of the other peers in the network. It is the reference to different peers in the largest routing table between the rings.

P2P (peer-to-peer) network examples

- Windows 10 updates are delivered both from Microsoft's servers and through P2P.
- Some online gaming platforms use P2P for downloading games between users. Blizzard Entertainment distributes Diablo III, StarCraft II, and World of Warcraft using P2P. Another large publisher, Wargaming, does the same with their World of Tanks, World of Warships, and World of Warplanes games.
- Many Linux operating systems are distributed via BitTorrent downloads using P2P transfers. Such examples are Ubuntu, Linux Mint, and Manjaro.
- In Windows 7 and Windows 8.1, when you create an ad-hoc network between two computers, you create a peer-to-peer network between them.
- Bitcoin, Blockchain.
- Napster

2.How to Create P2P

The network starts with a peer and that peer is referred to as the boot peer. When P2P network is created for the first time, P2P network needs boot peer. The boot peer allows other peers to join the network. Here is the join process:

- Joining peer sends a "join" request to the boot peer, and gets a GUID back.
- Joining peer sends a "copy routing table" to the boot peer.
- Joining peer finds the correct peers it should have in its own routing table.

GUID = Globally Unique identifiers for nodes and stored objects.

Routing Table = Table containing routes to where the router should forward that packet when a packet arrives.

The Join Request

The first peer to join the network apart from the boot peer, will connect to the boot peer and send a “join” message. The boot peer responds with a new GUID to the joining peer.

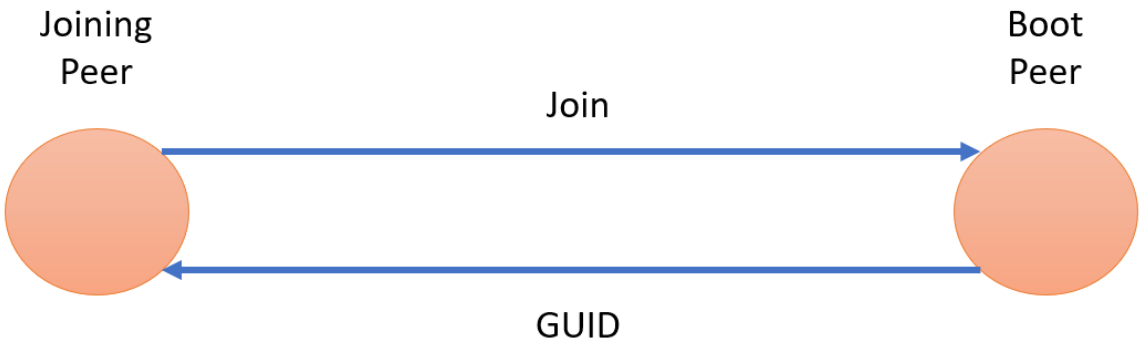

Image 2: Connecting of peers in a P2P network

The Copy Routing Table Request

After getting a GUID, the joining peer requests to get a copy of the boot peers routing table.

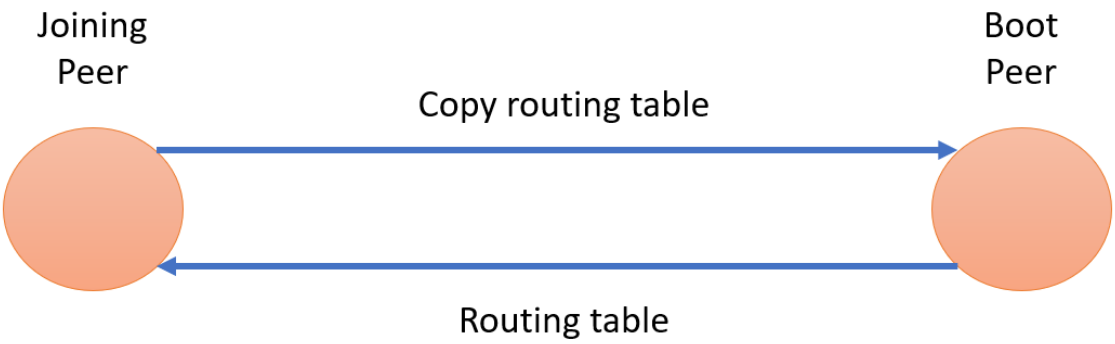

Image 3: Connecting of peers in a P2P network

Finding the Correct Peers for the Routing Table
After having received a copy of the boot peers routing table, the joining peer uses this routing table to find the peers it should really have in its routing table.

When a peer no longer wants to be part of a P2P network, it will send a "leave" request to all peers in its routing table. In this way, each of these peers can remove the leaving peer from their routing table.

3.Napster P2P Background
Napster's architecture is based on the centralized version of the P2P network. Its central server is used to route traffic between users. In addition, its central server maintains the address of file directories shared by users. These directories are updated each time a user logs in or out of the Napster server network. Clients are automatically connected to the internally designated "metaserver" that assists in public connectivity. This metaserver assigns a randomly available, light-loaded server from one of the clusters. The servers are clustered around five to a geographic site and internet stream and can each handle up to 15,000 users. The client registers with the assigned server and provides identity and shared file information for the server's local database. In turn, the client receives data from the server about connected users and available files from the user. Although officially organized around a user directory, the Napster app is very data-centric.

Users are completely anonymous. The user directory can never be queried. What users can do is simply search for content and specify a node to download. That's why things like directory, computer ID, IP address just run in the background and therefore users stay anonymous. When a user of the P2P file sharing system sends a request or searches for a particular file, the central server cross-checks the request with the server's database to create a list of files that match the search request. The requesting user can then select the desired file from the list and click on that file at that time can open a direct HTTP connection with the individual computer that owns it. The download of the actual file takes place directly from one network user to another without the intervention of the central server. Files are never stored on the server.

Napster Messages Data Structures

Image 4: This is each message format to/from the Napster central server

Where:

- Length specifies the length of the payload.
- Function defines the message type of the packet
- Payload this portion of the message is a plain ASCII string

Every block of header and payload is separated by "blanks" which make the synchronization of the incoming messages possible. Most of blocks have no fixed length. The blanks make separation of data blocks in incoming bit streams possible.

Initialisation

Image 5: A registered Napster host, acting like a client, sends to the server a LOGIN(0x02) message

Let's say the user's name is nick.

- Nick & Password identify the user
- Port is the port which the client is listening on for data transfer.
- Client_Info is a string containing the client version info.
- Link_Type is a integer indicating the client's bandwidth.

The IP address of the host should not be included in the message. However, the server can automatically extract it from the TCP packet in which the message is packaged for communication. An unregistered host sends a New User Login (0x06) similar to the Login (0x02) format, with the email address appended at the end. The server sends a Login Ack (0x03) to the client after a successful login. If Nick is registered, the e-mail address provided during registration is sent back to the user.

Client Notification of Shared File (0x64)

With the Client Notification of Shared File (0x64) message, the client sends the files it wants to share in order.

Image 7: Client Notification of Shared File (0x64) message format

- MD5 is the hash of the shared file. The MD5 algorithm generates a 128-bit password. It is impossible to generate the same passwords. The purpose of the MD5 algorithm is to ensure the security of users.
- Size is the file size in bytes
- Bitrate is the bit rate of the mp3 in kbps
- Frequency is the sample rate of the mp3 in Hz
- Time is the duration of the music file in seconds

File Request

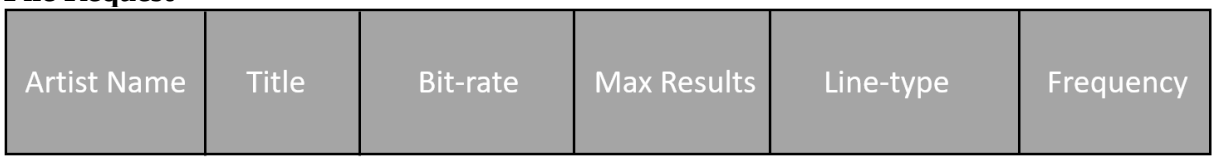

Image 8: The downloading client will first send either a Search (0xC8) or Browse(0xD3) and this is format of message.

- Max is the maximum number of results.
- Link-Type Range is the rage of link-types.
- Bit-rate Range is the rage of bit-rate.
- Frequency Range is the range of sample frequencies in Hz.

The artist name and the song title are checked from the file name only. ID3 TAGs can contain a lot of information such as Song name, Artist name, album name, filena etc. but Napster does not make use of the ID3 in mp3 files in its search criteria. The payload of the Browse (0xD3) message does only contains the of the host. It requests a list of the host's shared files.

Response and Browse Response

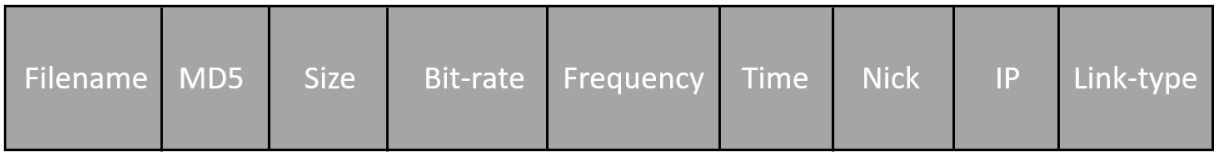

Image 9: The server answers respectively with a Search Response (0xC9) or a Browse Response (0xd4) with the this formats.

Where:

- MD5 hash value of the requested file
- Size file size in bytes
- Bitrate bit rate of the mp3 in kbps
- Frequency sample rate of the mp3 in Hz
- Time specify the length of the file
- Nick identify the user who shares the file
- IP 4 Bytes integer representing the IP address of the user with the file.
- Link-Type Refer to Login Message.

Download Request

To request a download, a DOWNLOAD REQUEST (0xCB) message is sent to the server. The client requests to download from other clients.

Image 10: DOWNLOAD REQUEST (0xCB)

Download ACK
The server will answer with a DOWNLOAD ACK (0xCC) containing more information about the file (Linespeed, Port Number, etc).

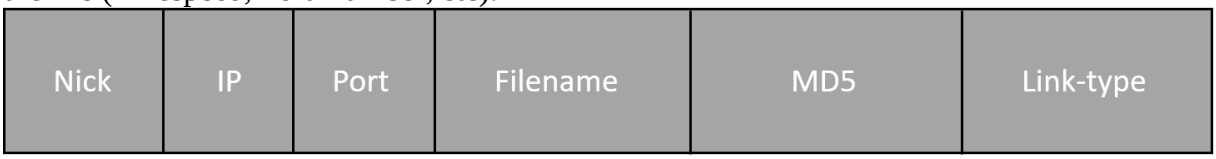

Image 11: DOWNLOAD ACK (0xCC)

File Transfer
From here on, hosts do not send messages to the server. The host requesting the file establishes a TCP connection from the server to the data port specified in the 0xCC message. It sends HTTP messages to request the file it wants to download. The content of the sent message is a "GET" string in a single packet.

Image 12: HTTP – messages Format

- Nick is the client's nick.
- Offset is the byte offset in the file to begin the transfer at.

The computer owning the file returns the file size and data stream in response. Communication between the computers receiving and sending the file uses P2P architecture. When Data Transfer is initiated, the downloader should send the message DOWNLOADING FILE (0xDA), notifying the server that it is downloading a file. When the transfer is complete, the client sends a DOWNLOAD COMPLETE (0xDB) message.

Firewalled Downloading
Napster also has method to allow clients behind firewalls to share their contents as well.It is like the normal "Download Request", only difference is that it is for use when he person sharing the file can only make outgoing TCP connection because of the firewall that is blocking the incoming messages. The ALTERNATE DOWNLOAD REQUEST (0x1F4) message should be used to request files from users who have specified their data port as '0'in their login message.

This ALTERNATE DOWNLOAD ACK (0x1F5) is sent to the uploader when its data port is set to 0 to indicate they are behind a firewall and need to push all data. The uploader is responsible for connecting to the downloader transfer the file. These alternatives help bypass the Firewall.

Implementation
Unlike its alternatives (Gnutella, etc.), Napster did not allow non-musical purchases and was trying to prevent them. Some alternative Napster servers, such as OpenNap, started out as "safe havens" for Napster users when Napster began filtering content, for a time starting to fill the void, interconnecting legacy Napster clients, clones, and variations with a new type of server that expanded to be a napster protocol to all file types. No matter how reconfigured the Napster model was, the basic requirement for a "Napster compatible" central server remained a serious constraint for a network based on this technology or any clone of it, and this problem prevented

the creation of a fully server-independent network. To circumvent this limitation, other protocols and architectural models are needed, such as serverless networks in the Gnutella style.

Gnutella

Gnutella [1] P2P is a file sharing protocol that communicates over a network. Like Napster, Gnutella is often used to share music files. It has created a huge problem for music publishing.

Unlike Napster, Gnutella is not a Web site. After installing and launching Gnutella, the user's computer becomes both a client and a server in the network, which is called GnutellaNet. Gnutella allows network members to share any file type, whereas Napster is limited to MP3 music files.

4. Static Entry Pusher API Use

We will create our own topology in the future. We will be able to create our own P2P network. In this network we will create, it is possible to direct the flow between hosts. We will examine this flow routing and other alternatives in this section. In the following sections, we will create our own P2P network.

We need to write the controller ip address that we use where it says "<controller_ip>". Since we usually run the controller on the computer we use, we need to use "127.0.0.1".

Adding a Flow

Although there is a connection between hosts in the network, we may want to ensure that the transferred data is transferred to different directions or to another direction. For this we need to add flow.

```
curl -X POST -d '{"switch":"00:00:00:00:00:00:00:01", "name":"flow-mod-1", "cookie":"0", "priority":"32768", "in_port":"1","active":"true", "actions":"output=2"}' http://<controller_ip>:8080/wm/staticentrypusher/json
```

If we examine the above command, there are places we need to change for each flow. These are in order:

- We can change the switch address according to where we will add the flow, to change the switch address, we must change the place where it says "00:00:00:00:00:00:00:01" according to the switch we want.
- The name of each flow should be unique and special. In order to change the name of these flows, we need to write the name we want to put in our flow where it says "flow-mod-1".
- We may want to change the priorities of the flows on the switch. In order to change this, it is necessary to change the number where it says "32768". The higher this number, the higher priority the flow will be, but the maximum can be 32768.
- To determine the starting port, we need to change the value where it says in port. In order to change the end port, we need to change where it says output. By changing these ports, we adjust the direction of the flow.
- We need to write the controller ip address that we use where it says "<controller_ip>". Since we usually run the controller on the computer we use, we need to use "127.0.0.1".

Adding a Group

Group is an abstraction that facilitates more complex and specialized packet operations that cannot easily be performed through a flow table entry. Adding a group is very similar to adding

a flow. The purpose of the group action is to further process these packets and assign a more specific forwarding action to them.

```
curl -X POST -d '{"switch":"00:00:00:00:00:00:00:01", "entry_type":"group",
"name":"group-mod-1", "active":"true", "group_type":"select",
"group_id":"1",  "group_buckets":[ {"bucket_id":"1", "bucket_watch_group":"any",
"bucket_weight":"50", "bucket_actions":"output=2"}, {"bucket_id":"2",
"bucket_watch_group":"any", "bucket_weight":"50", "bucket_actions":"output=3"}
]}' http://<controller_ip>:8080/wm/staticentrypusher/json
```

If we examine the above command, there are places we need to change for each group. These are in order:

- We can change the switch address according to where we will add the group, to change the switch address, we must change the place where it says "00:00:00:00:00:00:00:01" according to the switch we want.
- Buckets are ordered sequentially from lowest ID to highest ID.
- Although OpenFlow does not define bucket IDs, a group entry must have a unique bucket ID per bucket to define ordering of the buckets.
- bucket_actions shows which port to send to. To change this, we need to write the number of the port we want to forward instead of the number written there.
- We need to write the controller ip address that we use where it says "<controller_ip>". Since we usually run the controller on the computer we use, we need to use "127.0.0.1".

Listing Entries

We can use it to list the entries made so far

```
curl http://<controller_ip>:8080/wm/staticentrypusher/list/00:00:00:00:00:00:00:01/json
```

If we write the above code by showing the switch address (00:00:00:00:00:00:00:01), it will only show the flows made by that switch. But if we write “all” instead of address as below, it shows all the flows of all switches. If we write the commands on the browser without using the "curl" text, we can examine the flows through the browser.

```
curl http://<controller_ip>:8080/wm/staticentrypusher/list/all/json
```

Clearing Entries

```
curl http://<controller_ip>:8080/wm/staticentrypusher/clear/00:00:00:00:00:00:00:01/json
```

If we write the above code by showing the switch address (00:00:00:00:00:00:00:01) like this, it will only delete the flows made for that switch. But if we write “all” instead of address as below, it will delete all flows of all switches.

```
curl http://<controller_ip>:8080/wm/staticentrypusher/clear/all/json
```

Deleting Entries

```
curl -X DELETE -d ‘{“name”:”flow-mod-1”}
http://<controller_ip>:8080/wm/staticentrypusher/json
```

By writing this code, we can delete the flow we created earlier. Since the name of each flow is special and unique, we need to write the name of the stream that we want to delete where it says "flow-mod-1".

Using Curl to Access the REST API

We need to write the controller ip address that we use where it says "<controller_ip>". Since we usually run the controller on the computer we use, we need to use "127.0.0.1".

An example REST call that retrieves data from Floodlight at IP address <controller-ip> is:

```
curl http://<controller-ip>:8080/wm/core/controller/switches/json
```

An example REST call that provides data to Floodlight at IP address <controller-ip> is:

```
curl http://<controller-ip>:8080/wm/core/switch/all/role/json -X POST -
d '{"role":"MASTER"}'
```

The output from Floodlight's REST API is in JSON. Python can be used to parse the JSON string returned:

```
curl http://<controller-ip>:8080/wm/core/controller/switches/json | python -m json.tool
```

5.Create Your Own Topology With Static Entry Pusher API

First we need to create a topology with 10 hosts and 10 switches to work on. In this example topology, we will not connect every switch each other. Connection methods will be as in the photo below. The topology we will use is the S10H10.py file in the folder.

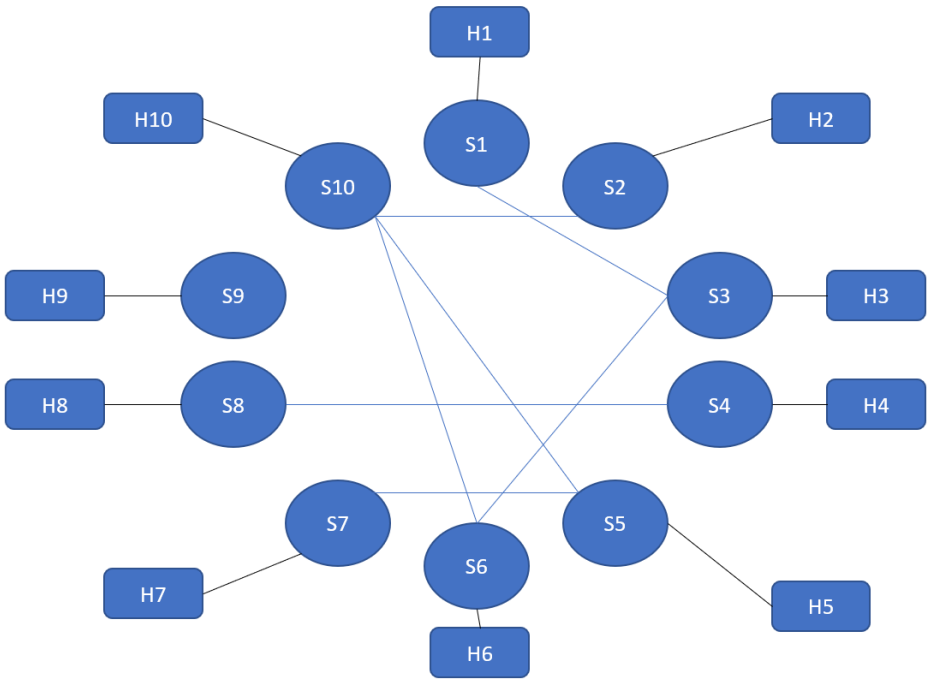

Image 13: Network Topology

Step 1:

Before running the topology, we need to run Floodlight. So we need to open a new terminal, open the Floodlight folder and write the following code.

```
java -jar target/floodlight.jar
```

Step 2:

Now we have to open a new terminal and navigate to the file location where our topology is and run the code below. In this way, our own topology will work.

```
sudo mn --custom S10H10.py --topo P2PNetwork --controller=remote,ip=127.0.0.1,port=6653
--switch ovsk,protocols=OpenFlow13
```

Step 3:
If we write “Net” on the terminal, we will see the ports where the Host and the switches are connected to each other. For example, eth1 means Port 1.

```
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s2-eth1
h3 h3-eth0:s3-eth1
h4 h4-eth0:s4-eth1
h5 h5-eth0:s5-eth1
h6 h6-eth0:s6-eth1
h7 h7-eth0:s7-eth1
h8 h8-eth0:s8-eth1
h9 h9-eth0:s9-eth1
h10 h10-eth0:s10-eth1
s1 lo:  s1-eth1:h1-eth0 s1-eth2:s3-eth2
s2 lo:  s2-eth1:h2-eth0 s2-eth2:s10-eth2
s3 lo:  s3-eth1:h3-eth0 s3-eth2:s1-eth2 s3-eth3:s6-eth2
s4 lo:  s4-eth1:h4-eth0 s4-eth2:s8-eth2
s5 lo:  s5-eth1:h5-eth0 s5-eth2:s7-eth2 s5-eth3:s10-eth3
s6 lo:  s6-eth1:h6-eth0 s6-eth2:s3-eth3 s6-eth3:s10-eth4
s7 lo:  s7-eth1:h7-eth0 s7-eth2:s5-eth2
s8 lo:  s8-eth1:h8-eth0 s8-eth2:s4-eth2
s9 lo:  s9-eth1:h9-eth0
s10 lo:  s10-eth1:h10-eth0 s10-eth2:s2-eth2 s10-eth3:s5-eth3 s10-eth4:s6-eth3
c0
```

Image 14: Connections on the topology

Step 4:
If we type “pingall” into the terminal, we can see which hosts can connect with each other.

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 X h5 h6 h7 X X h10
h2 -> h1 h3 X h5 h6 h7 X X h10
h3 -> h1 h2 X h5 h6 h7 X X h10
h4 -> X X X X X X h8 X X
h5 -> h1 h2 h3 X h6 h7 X X h10
h6 -> h1 h2 h3 X h5 h7 X X h10
h7 -> h1 h2 h3 X h5 h6 X X h10
h8 -> X X X h4 X X X X X
h9 -> X X X X X X X X X
h10 -> h1 h2 h3 X h5 h6 h7 X X
*** Results: 51% dropped (44/90 received)
```

Image 15: Indicates whether the hosts are communicating with each other.

What is Openflow?
The Openflow protocol, in SDN, allows the Controller to manage changes in the network's topology and filtering packets in the network. The most important components in this protocol are the controller and OpenFlow network switches. To briefly explain Openflow, it is a method of running a program on another server so that network packets can find the right path.

Static Entry Pusher API
Now let's make a connection between only H6-H5-H2 using flows.

Step 1:
Let's open a new terminal and write the following codes one by one, one by one. Let's first connect H5 to H6.

```
curl -X POST -d '{"switch":"00:00:00:00:00:00:00:06", "name":"flow-mod-1", "cookie":"0", "priority":"32768", "in_port":"1","active":"true", "actions":"output=3" }' http://localhost:8080/wm/staticentrypusher/json
```

The code above helps the 6th switch stream is regulated. The data coming from port 1 is transferred to port 3.

```
curl -X POST -d '{"switch":"00:00:00:00:00:00:00:06", "name":"flow-mod-2", "cookie":"0", "priority":"32768", "in_port":"3","active":"true", "actions":"output=1" }' http://localhost:8080/wm/staticentrypusher/json
```

The code above helps the 6th switch stream is regulated. The data coming from port 3 is transferred to port 1. In this way, a round trip is provided between these two ports

```
curl -X POST -d '{"switch":"00:00:00:00:00:00:00:10", "name":"flow-mod-3", "cookie":"0", "priority":"32768", "in_port":"4","active":"true", "actions":"output=3" }' http://localhost:8080/wm/staticentrypusher/json
```

The code above helps the 10th switch stream is regulated. The data coming from port 4 is transferred to port 3.

```
curl -X POST -d '{"switch":"00:00:00:00:00:00:00:10", "name":"flow-mod-4", "cookie":"0", "priority":"32768", "in_port":"3","active":"true", "actions":"output=4" }' http://localhost:8080/wm/staticentrypusher/json
```

The code above helps the 10th switch stream is regulated. The data coming from port 3 is transferred to port 4. In this way, a round trip is provided between these two ports

```
curl -X POST -d '{"switch":"00:00:00:00:00:00:00:05", "name":"flow-mod-5", "cookie":"0", "priority":"32768", "in_port":"3","active":"true", "actions":"output=1" }' http://localhost:8080/wm/staticentrypusher/json
```

The code above helps the 5th switch stream is regulated. The data coming from port 3 is transferred to port 1.

```
curl -X POST -d '{"switch":"00:00:00:00:00:00:00:05", "name":"flow-mod-6", "cookie":"0", "priority":"32768", "in_port":"1","active":"true", "actions":"output=3" }' http://localhost:8080/wm/staticentrypusher/json
```

The code above helps the 5th switch stream is regulated. The data coming from port 1 is transferred to port 3. In this way, a round trip is provided between these two ports

After writing the codes, let's type pingall and see if they can establish a connection.

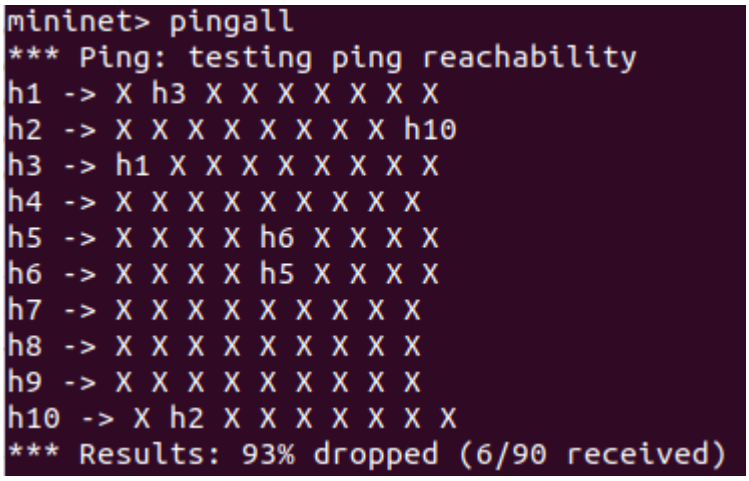

Image 16: Indicates whether the hosts are communicating with each other.

As you can see, a connection has been established between them.

Step 2:
Now let's connect H2 with H5 and H6. Let's write the following codes in order.

curl -X POST -d '{"switch":"00:00:00:00:00:00:00:02", "name":"flow-mod-7", "cookie":"0", "priority":"32768", "in_port":"1","active":"true", "actions":"output=2" }' http://localhost:8080/wm/staticentrypusher/json

The code above helps the 2th switch stream is regulated. The data coming from port 1 is transferred to port 2.

curl -X POST -d '{"switch":"00:00:00:00:00:00:00:02", "name":"flow-mod-8", "cookie":"0", "priority":"32768", "in_port":"2","active":"true", "actions":"output=1" }' http://localhost:8080/wm/staticentrypusher/json

The code above helps the 2th switch stream is regulated. The data coming from port 2 is transferred to port 1. In this way, a round trip is provided between these two ports

curl -X POST -d '{"switch":"00:00:00:00:00:00:00:10", "name":"flow-mod-9", "cookie":"0", "priority":"32768", "in_port":"2","active":"true", "actions":"output=1" }' http://localhost:8080/wm/staticentrypusher/json

The code above helps the 10th switch stream is regulated. The data coming from port 2 is transferred to port 1.

curl -X POST -d '{"switch":"00:00:00:00:00:00:00:10", "name":"flow-mod-10", "cookie":"0", "priority":"32768", "in_port":"1","active":"true", "actions":"output=2" }' http://localhost:8080/wm/staticentrypusher/json

The code above helps the 10th switch stream is regulated. The data coming from port 1 is transferred to port 2. In this way, a round trip is provided between these two ports

After writing the codes, let's type pingall and see if they can establish a connection.

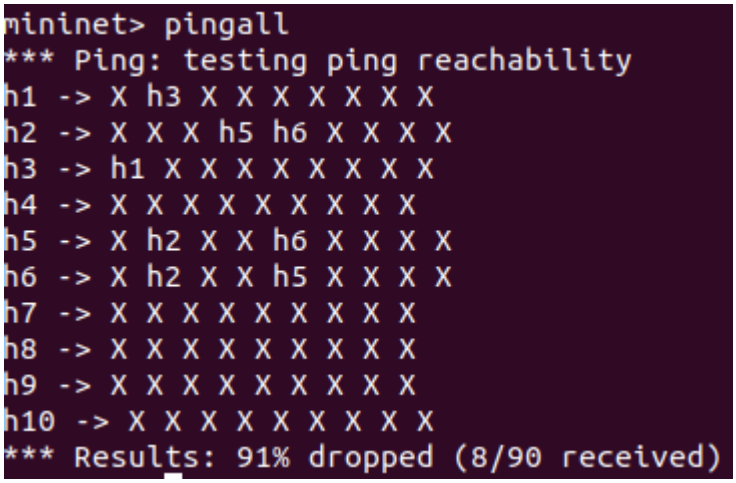

Image 17: Indicates whether the hosts are communicating with each other.

As you can see, a connection has been established between them.

Step 3:
Let's write the code below to list the entries, so we can see the flows we have done so far.

```
curl http://127.0.0.1:8080/wm/staticentrypusher/list/all/json
```

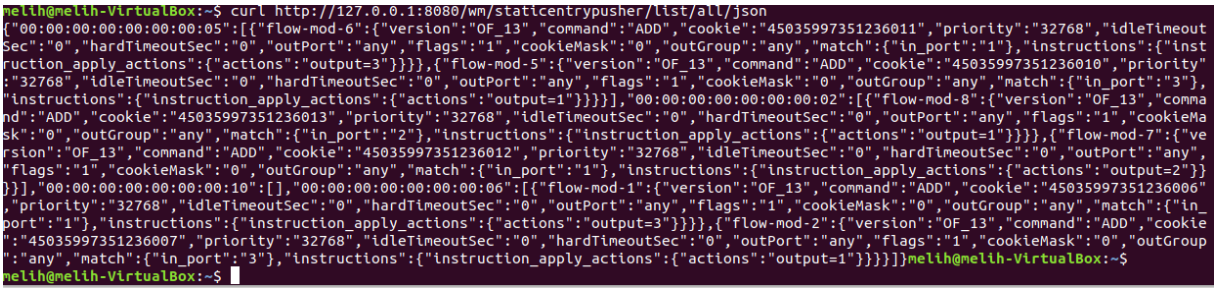

Image 18: Shows all entries.

6.How to Create Your Own P2P Chat App Using Python
We will use the network we prepared earlier. For our P2P messaging application [4], 1 server and 2 clients need hosts. We will set it to be H2 server, H5 and H6 client. Clients will only use the server to find each other, and then they will only talk among themselves. There will be no need to use the server.

Step 1:
We need to create the Client and Server files. You can find these files in the folder.

Step 2:
Now the most important part is that we need to run terminal for host 2 by typing "xterm h2" into the mininet. We need to run server.py from this terminal. We should do the same for H5 and H6, but client.py should be run on both.

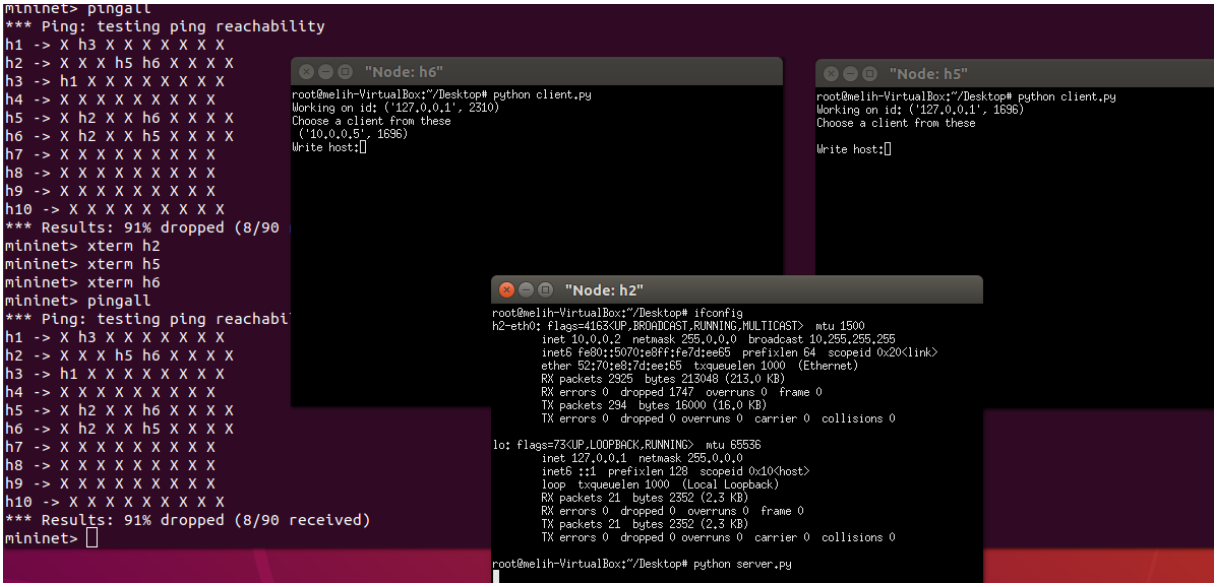

Image 19: Hosts are actively running but not connected to each other.

Step 3:

Now we can write Host number and Port number. Host number is IP address. As you can see, we can message between hosts.

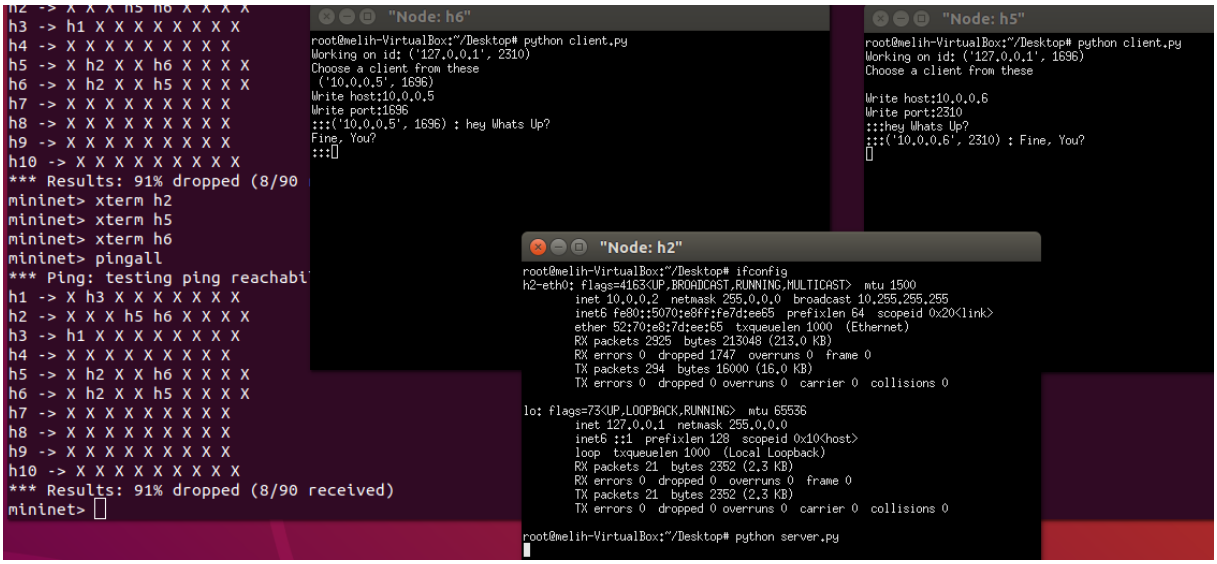

Image 20: Hosts are messaging each other.

7. References

[1] Karagiannis, Thomas, et al. "File-sharing in the Internet: A characterization of P2P traffic in the backbone." *University of California, Riverside, USA, Tech. Rep* (2003).

[2] Nobre, Jéferson Campos, et al. "A survey on the use of P2P technology for network management." *Journal of Network and Systems Management* 26.1 (2018): 189-221.

[3] Nguyen, Phi H. "Proposal for Peer-to-peer chat application using Hole-punching." (2020).

[4] Renner, Aaron. "Encrypted Peer-To-Peer Chat Application in Java."