P2P
NETWORK DOCUMENT

Melih Yesilyurt

Contents

1. Peer to Peer Network

In its simplest form, a peer-to-peer (P2P) 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.

Legal status of peer-to-peer software

With peer-to-peer software, users can share all kinds of digital content with those who use the applications. Although this process is legal with this definition, when the shared content is copyrighted and monetized content (commercial music albums, production films, licensed software, etc.), sharing these content and obtaining it from the people who share it becomes illegal. But it's not the software and technology that becomes illegal, it's its abuse.

The history of P2P (peer-to-peer) networks

The precursor of peer-to-peer networks appears to be USENET, which was developed in 1979. It was a system that allowed users to read and post messages/news. It was a network system similar to the online forums today, but with the difference that USENET did not rely on a central server or administrator. USENET copied the same message/news to all the servers found in the network. Similarly, peer-to-peer networks distribute and use all the resources available to them.

The next big thing in the history of P2P was the year 1999 when Napster came to life. Napster was file-sharing software that was used by people to distribute and download music. The music shared on Napster was usually copyrighted and thus illegal to distribute. However, that did not stop people from getting it. Although Napster was the one that got P2P into the mainstream, Napster ultimately failed and was shut down by authorities because of all the content that was shared illegally on it. Nowadays, P2P remains one of the most popular technologies for sharing files over the internet, both lawfully and unlawfully.

Unstructured P2P networks

Unstructured P2P networks do not have a specific node organization. Participants communicate randomly with each other. These systems are considered to be resistant to high I/O movements (eg, often several nodes join and leave the network).

While unstructured P2P networks are easier to build, they require higher CPU and memory usage because search queries are sent to as many peers as possible. This causes the network to be flooded with queries, especially if the queried content is served by a small number of nodes.

Structured P2P networks

Structured P2P networks have an organized architecture, allowing nodes to efficiently search for files even on content that is not widely available. In most cases, hash functions are used to facilitate database searches to achieve this.

Although structured networks are more efficient, they tend to be more centralized and often require higher installation and maintenance costs. In addition, structured networks are more vulnerable to high I/O mobility.

Hybrid P2P networks

Hybrid P2P networks combine the traditional client-server model with some features of peer-to-peer architecture. For example, they may include a central server design that supports peer-to-peer connectivity.

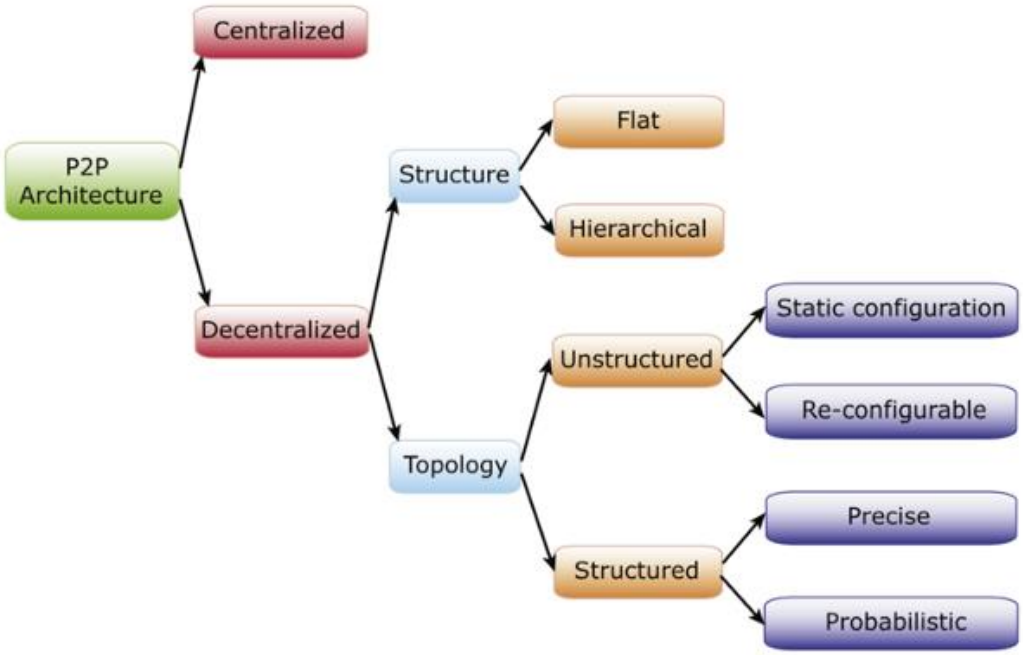
Compared to the other two types, hybrid models have better overall performance. Combining the main advantages of both approaches, they can offer significant efficiency and decentralization at the same time.

Comparison of centralized and decentralized

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.

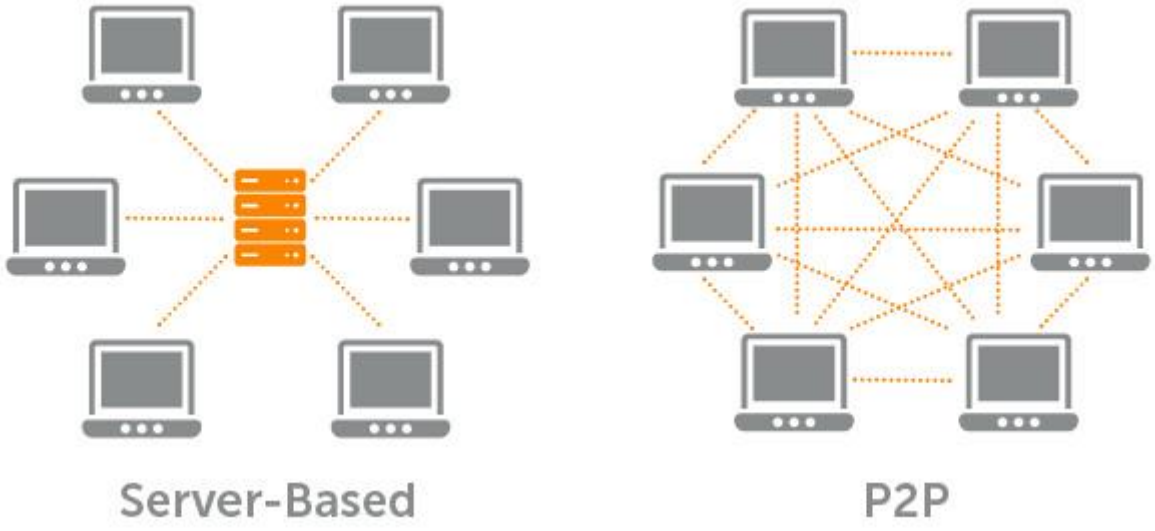
In fact, most systems need a central authority to direct network movements, which makes them centralized to some degree. For example, some P2P file sharing systems allow users to search and download files from other users, but these users cannot be involved in other processes such as handling search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



Why are peer-to-peer networks useful?
P2P networks have some characteristics that make them 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.



The role of P2P in blockchains
Satoshi Nakamoto defined Bitcoin as “Peer-to-Peer Electronic Cash System” in its early days. Bitcoin was created as a type of digital money. It can be transferred from one user to another via a P2P network managed by a distributed ledger called blockchain.
In this context, it is the P2P architecture inherent in the blockchain that enables Bitcoin and other cryptocurrencies to be transferred worldwide without the need for intermediaries or any central server. Also, anyone who wants to participate in the process of verifying and confirming blocks can operate as a Bitcoin node.

Therefore, there is no bank in the Bitcoin network that records and processes transactions. Instead, the blockchain acts as a digital ledger that publicly records all transactions. Basically, each node keeps a copy of the blockchain and compares it to other nodes to ensure the accuracy of the data. Malicious acts or inaccuracies are quickly rejected by the network.

In the context of cryptocurrency blockchains, nodes can have different roles. For example, full nodes ensure network security by verifying transactions according to the system's consensus rules.
Each full node maintains a complete and updated copy of the blockchain, which can thus contribute to the collaborative effort to verify the true state of the distributed ledger. However, it is important to note that not all fully validator nodes are miners.

P2P (peer-to-peer) network examples

We all use peer-to-peer networks to connect computers and devices without the need to configure a server. Having to create a server for everything is expensive and difficult to manage, so in some situations, it's easier and more affordable to use P2P networks. Here are some examples of common use cases for peer-to-peer networks:

- Windows 10 updates are delivered both from Microsoft's servers and through P2P.
- Sharing large files over the internet is often done using a P2P (peer-to-peer) network architecture. For example, 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. Others, like Steam or GOG, choose not to use P2P and prefer maintaining dedicated download servers around the world.
- 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.
- If you're using Windows 7, Windows 8.1, or a Windows 10 version before Version 1803, you can connect the computers in your home to a Homegroup, thus creating a peer-to-peer network between them. The Homegroup is a small group of computers that are connected between themselves to share storage and printers. This is one of the most common uses for peer-to-peer technology. Some people might say that Homegroups can't be peer-to-peer because the computers in the network are connected to a router. However, keep in mind that the router has nothing in common with managing what the computers from the Homegroup share among themselves. The router does not work as a server but merely as an interface or gate between the local network and the internet.

2. Napster

Napster is a set of three music-focused online services. It was founded in 1999 as a pioneering peer-to-peer (P2P) file sharing Internet software that emphasized sharing digital audio files, typically audio songs, encoded in MP3 format. As the software became popular, the company ran into legal difficulties over copyright infringement. It ceased operations and was eventually acquired by Roxio. Napster became an online music store until it was merged with Rhapsody from Best Buy on December 1, 2011.

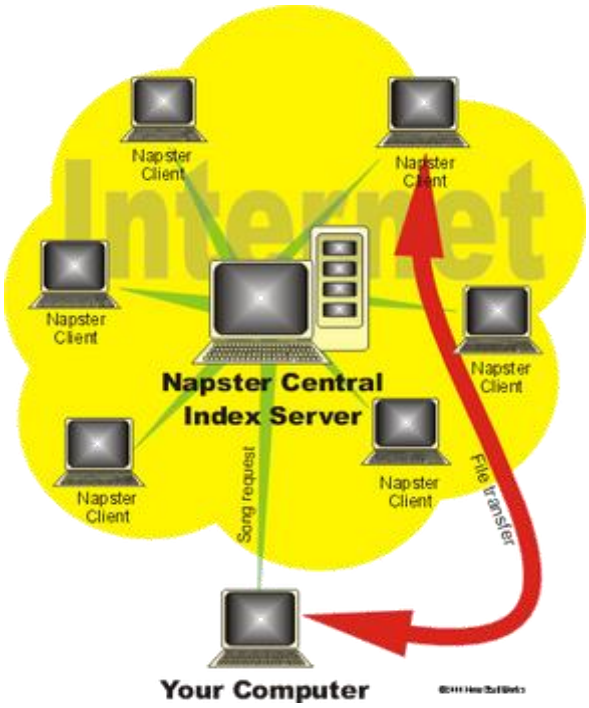
Although there were already networks that facilitated the distribution of files across the Internet, such as IRC, Hotline, and Usenet, Napster specialized in MP3 files of music and a user-friendly interface. At its peak the Napster service had about 80 million registered users. Napster made it relatively easy for music enthusiasts to download copies of songs that were otherwise difficult to obtain, such as older songs, unreleased recordings, studio recordings, and songs from concert bootleg recordings. Napster paved the way for streaming media services and transformed music into a public good for a brief period of time.

High-speed networks in college dormitories became overloaded, with as much as 61% of external network traffic consisting of MP3 file transfers. Many colleges blocked its use for this reason, even before concerns about liability for facilitating copyright violations on campus.

How the Napster Worked?

Napster (Napster was Fanning's nickname in high school, because of his hair) is a different way to distribute MP3 files. Instead of storing the songs on a central computer, the songs live on users' machines. This is called peer to peer sharing, or P2P. When you want to download a song using Napster, you are downloading it from another person's machine, and that person could be your next-door neighbor or someone halfway around the world.

Let's take a look at what was necessary for you to download a song that you are interested in using the old Napster.com:



You needed:

- A copy of the Napster utility installed on your computer
- A directory on your computer that has been shared so that remote users can access it
- Some type of Internet connection

The provider of the song needed:

- A copy of the Napster utility installed on his computer
- A directory on his computer that has been shared so that someone else could access it
- Some type of Internet connection that was "on"
- A copy of the song in the designated, shared directory

Here is what happened when you decided to look for the song:

- You opened the Napster utility.
- Napster checked for an Internet connection.
- If it found a connection, Napster logged you onto the central server. The main purpose of this central server was to keep an index of all the Napster users currently online and connect them to each other. It did not contain any of the MP3 files.
- You typed in the title or artist of the song you were looking for.
- The Napster utility on your computer queried the index server for other Napster computers online that had the song you requested.
- Whenever a match was found, the Napster server informed your computer where to find the requested file.
- When the server replied, Napster built a list of these systems in the results window.
- You clicked on the file(s) that interested you and then chose Download.
- Your copy of Napster attempted to establish a connection with the system hosting the file you selected.
- If a connection was successfully made, the file began downloading.
- Once the file was downloaded, the host computer broke the connection with your system.
- You opened up your MP3 player software and listened to the song.

How to Create Your Own Decentralized File Sharing Service Using Python

In this section, we use python and leverage the help of sockets to create a peer to peer network. We make our module in such a way that when a server disconnects it automatically makes another peer a server.

Server.py

We first create a socket and then bind it to a Host and port. The port I use is 5000 and the host is 0.0.0.0 which basically listens to all incoming traffic. We accept the connections to the server and add the list of connections to a list. Then we have a method called the handler method that deals with the uploading of bytes. We run this method in a different thread so that it does not bottleneck the entire program. The client sends a request to receive data called REQUEST_STRING. When the server sees this request, it starts uploading the data. We have to be sure that the data we are uploading is in the format of bytes. We also make sure that when we receive the signal 'q' from the peer we know that, that peer is disconnecting from the network.

```
"""
This method deals with sending info to the clients
This methods also closes the connection if the client has left
:param: connection -> The connection server is connected to
:param: a -> (ip address, port) of the system connected
"""
def handler(self, connection, a):

    try:
        while True:
            # server recieves the message
            data = connection.recv(BYTE_SIZE)
            for connection in self.connections:

                # The peer that is connected wants to disconnect
                if data and data.decode('utf-8')[0].lower() == 'q':

                    # disconnect the peer
                    self.disconnect(connection, a)
                    return
                elif data and data.decode('utf-8') == REQUEST_STRING:
                    print("-" * 21 + " UPLOADING " + "-" * 21)
                    # if the connection is still active we send it back the data
                    # this part deals with uploading of the file
                    connection.send(self.msg)
                    #convert_to_music(self.msg)
    except Exception as e:
        sys.exit()
```

Whenever a peer gets connected or disconnected, we broadcast the list of peers to all the peers in thus creating a decentralized architecture.

```
class p2p:
    # make ourself the default peer
    peers = ['127.0.0.1']
```

Client.py

This file also first creates a socket and then connects to the peer that is acting as the server. The peer sends a request to the peer acting as the server requesting the file. The server responds to this request and starts uploading the file. The client also starts receiving the bytes on a different thread.

```
"""
    This thread will deal with printing the recieved message
"""
def recieve_message(self):
    try:
        print("Recieving -------")
        data = self.s.recv(BYTE_SIZE)

        print("\nRecieved ..............................")

        if self.previous_data != data:
            fileIO.create_file(data)
            self.previous_data = data

        return data
    except KeyboardInterrupt:
        self.send_disconnect_signal()
```

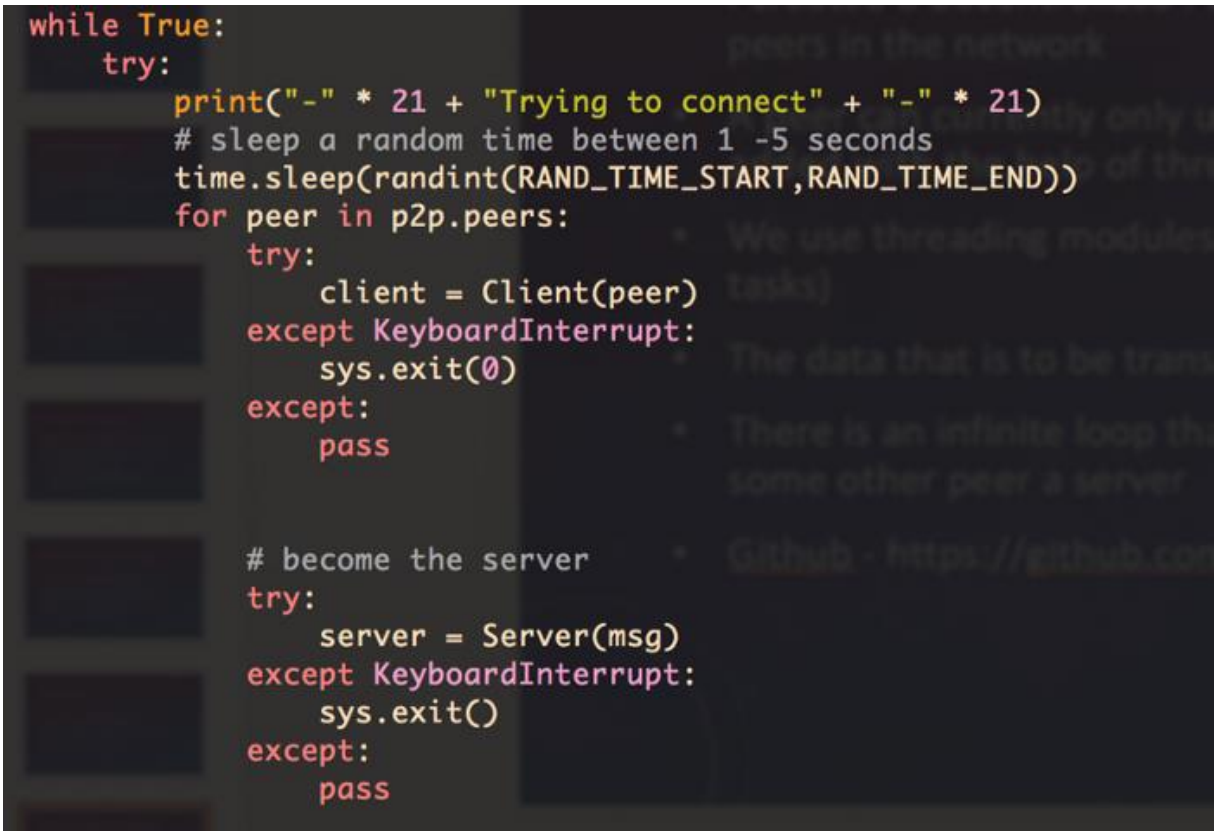
How do we sync the peers and differentiate it from the data being sent as bytes?

The way we deal with this problem is every instance of the program has its P2P class as mentioned earlier. The data that contains a list of peers is appended with a byte "\x11'. The client checks on receiving the data that weather it has this byte in front of it use the data to decode the bytes and update the list of peers in that instance of the program.

```
            break

        elif data[0:1] == b'\x11':
            print("Got peers")
            # first byte is the byte '\x11 we
ave peers
            self.update_peers(data[1:])
```

Finally, the driver of our program deals with creating a peer as the server and then when the server disconnects it takes care making another peer as the server. This is done by constantly looping over the list of peers and trying to make all the peers clients and then try making itself as the server. So, when that particular instance of the program disconnects another instance tries to become the server.



```
while True:
    try:
        print("-" * 21 + "Trying to connect" + "-" * 21)
        # sleep a random time between 1 -5 seconds
        time.sleep(randint(RAND_TIME_START,RAND_TIME_END))
        for peer in p2p.peers:
            try:
                client = Client(peer)
            except KeyboardInterrupt:
                sys.exit(0)
            except:
                pass

            # become the server
            try:
                server = Server(msg)
            except KeyboardInterrupt:
                sys.exit()
            except:
                pass
```

6. References

https://tr.wikipedia.org/wiki/Peer-to-peer

https://www.digitalcitizen.life/what-is-p2p-peer-to-peer/

https://en.wikipedia.org/wiki/Napster

https://computer.howstuffworks.com/napster.htm#:~:text=Instead%20of%20storing%20the%20songs,someone%20halfway%20around%20the%20world.

https://medium.com/@amannagpal4/how-to-create-your-own-decentralized-file-sharing-service-using-python-2e00005bdc4a