

Peer-to-Peer File-sharing

“Distributed Systems and Cloud Computing” Project

Students: Bundela Kuldeep, Lecini Fehemi, Savev David

Introduction

In the beginning of Internet's era, the web pages were all static, acting as an information source for readers or just for reading contents and nothing more. Later on, things start to change and companies began to provide their services on the web pages, making it possible to download contents, buy products and so on. This was a revolutionary era for Internet and its use started to expand all over the globe. The spread of internet users was even more significant also due to a new network technology invented in 1999: P2P network.

P2P Network



Peer to peer networks were created for the aim of sharing the files among the users and each computer acts as both a server and a client—supplying and receiving files—with bandwidth and processing distributed among all members of the network. All the nodes in the network are called peers and the network itself is quite reliable as if one node suddenly disconnects, the content we are looking for might still be offered from someone else.

The first peer-to-peer network was introduced in 1999 from an american college student, who created the first ever music-sharing service, namely Napster. He used a centralized index server, which users would query based on the song name. If the index located the song on any computer that was connected to the network, a user could download a copy and simultaneously offer his or her own files for other users.

However, this was only the first step of the new technology because not much time later the centralized P2P network evolved in a decentralized one, where there was no need to have a central server (and so a single point of failure) who was in charge of coordinating the peers. Probably one of the most famous p2p decentralized protocols nowadays is Gnutella and it is the basis of the Software called BitTorrent.

Project idea

In our project we implemented a console-based P2P application used for exchanging every type of digital content between peers by adopting a special version of Gnutella; the application was written by using the high level programming language Python.

In order to join the overlay network, users or peers have to know at least one peer, who is already part of the network. We used two different type of message protocols each with a different purpose: on one side we opted for UDP protocol for the exchange of the messages like 'Ping', 'Pong', 'Query' and 'Queryhit', since it is a pretty lightweight protocol and avoids the "Three-Way-Handshake" overhead added by the TCP protocol, which will cause a definitely higher network traffic.

Basically, with UDP we make the distinction of the type of request a peer is receiving (by adoption of IF-IF ELSE conditional statements) and based on the type of message that we call 'Command' in the code, we take a proper action.

On the other side, we use the reliable TCP protocol for establishing a connection between the peer that is looking for a specific file and the peer where he or she wants to download the file from. Also the file download is made through TCP, since it provides mechanisms to prevent the corruption of big files, such as the re-sending of missed packages and the guarantee that the packages arrive in the correct order.

All the messages itself are sent as JSON files, since Python provides libraries which make this data-exchange really easier in comparison to other approaches and definitely in comparison to other programming languages.

The mission was to develop a responsive application, able to quickly exchange data and messages; in order to achieve this goal and so to add the least possible overhead on top of the Network we opted to use the lowest possible communication technique: UDP, TCP Sockets and Multithreading.

The Software was designed in order to be functional in both, LAN and also WAN networks, however with the only difference that within WANs some special extra-configuration is needed on the Gateway to be able to manage properly the NAT-connected issues (e.g. Port-Forwarding, Firewall rules) and so to be able at the end to route correctly data frames and packets within and out from the LAN.

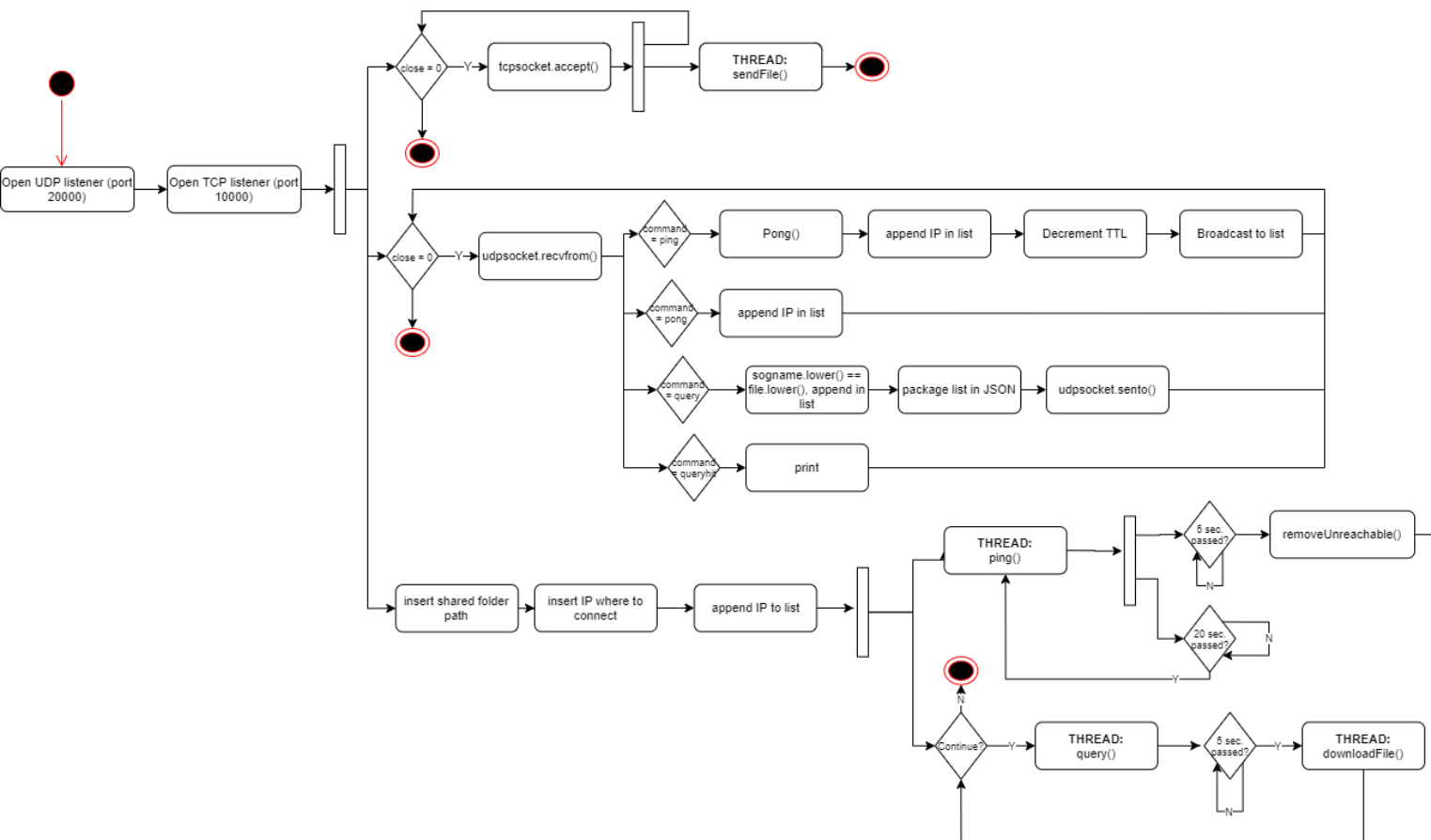
JSON messages structure:

The common structure of the JSON files exchanged between peers embeds “Command”, which specifies the type of message (ping, pong, query, queryhit) and the local or global IP address of the sender peer (depending whether is exchanging data with local or remote nodes).

Contents is specific for QueryHit messages, since it has to respond back to a Query message with the content matching a specific pattern, while TTL is specific for ping messages and is decremented whenever reaches a new peer before being broadcasted further.

COMMAND	IP Address (IPv4 version)	Contents (list with the complete name of the matching content)	TTL (Time to Live parameter value)
1 - Ping			
2 - Pong			
3 - Query			
4 - QueryHit	<i>Can be both global or private</i>	ONLY QUERYHIT	ONLY PING

High level statechart:



Ping & Pong

Once the user connects to the peer he knows (and so after checking that a valid IPv4 address was inserted), let it be “peer1”, he will also send a ping with a time to live and the peer1 on his side will forward the ping to his neighbors. Whenever a ping is sent, a pong is received back and once the

peer that generated the ping gets the pong, he checks if he already has that peer in his list. If so, then he does not add it to the list, otherwise yes.

The ping itself is performed every 20 seconds and was selected in an empirical manner after some real-time tests; the other peers have 5 seconds to respond with a pong to the ping message. If this does not happen then the peer that didn't respond will be deleted from the local list of peers, since it is considered to be unreachable. So basically, ping is used to discover the network and all the peers responding with a pong are added to the local list. The good thing with this approach is that we discover the network immediately, so every peer has the knowledge of the whole network and can interact directly with each single peer: hence, we have the guarantee that each single file matching with Query will be shown; however there are many messages that are sent, thus in a large network there might be the risk of flooding it with messages.

Query & Queryhit

The query starts whenever a peer in the network asks for a specific file; actually the application was deployed in a smart way in order to recognize as valid also partial and upper-lower case sentences, such for e.g. ESP inside despacito.mp3

Usually in a Gnutella protocol the query is forwarded from one peer to another and it clearly has a TTL (time to live), however, as we mentioned previously, we opted for a slightly different implementation: in our case the query request is sent to all the peers that we have in our local list, thus there is no need to forward it. If on one side there are many messages with the ping, on the other side with the query we generate less traffic. Once a peer gets a query, either he has or has not the file the other peer is looking for, he answers with a queryhit (empty in case of non-matching files). The peer that started the search gets all the IPs of the peers that have the searched file and the list of the files matching the pattern.

Download

After the user receives the queryhit from the different peers, he can select the IP he wants to download from and the content (in case of multiple files matching the query); at that point, a TCP connection with Client-Server topology is established: Client (requesting peer) and Server (sending peer).

As first, the Client notifies whether he has or not a partial version of the digital content (may due to network interruptions he has downloaded previously only a part of it) and eventually its size; the Server, at that point, estimates whether he has to send the whole content or not and sends to the Client the number of "chunks" he will transmit to the user (since there is the need of a strict coordination in their actions), which is simply found by performing $size\ file - partial\ file\ size$ where the dimension of each chunk is static with value 1024 bytes.

So, the Client peer iteratively stores and appends the received chunks into the file, while the Server peer in the same iterative way sends the chunks.

In case a peer suddenly disconnects for some reason and thus stops the file transferring, the Client peer looking and downloading chunks saves a local, partial copy of what he already downloaded and whenever he has the opportunity to start again the download, it will not start from scratch but from the chunk it was stopped. So only the chunks missing will be downloaded.

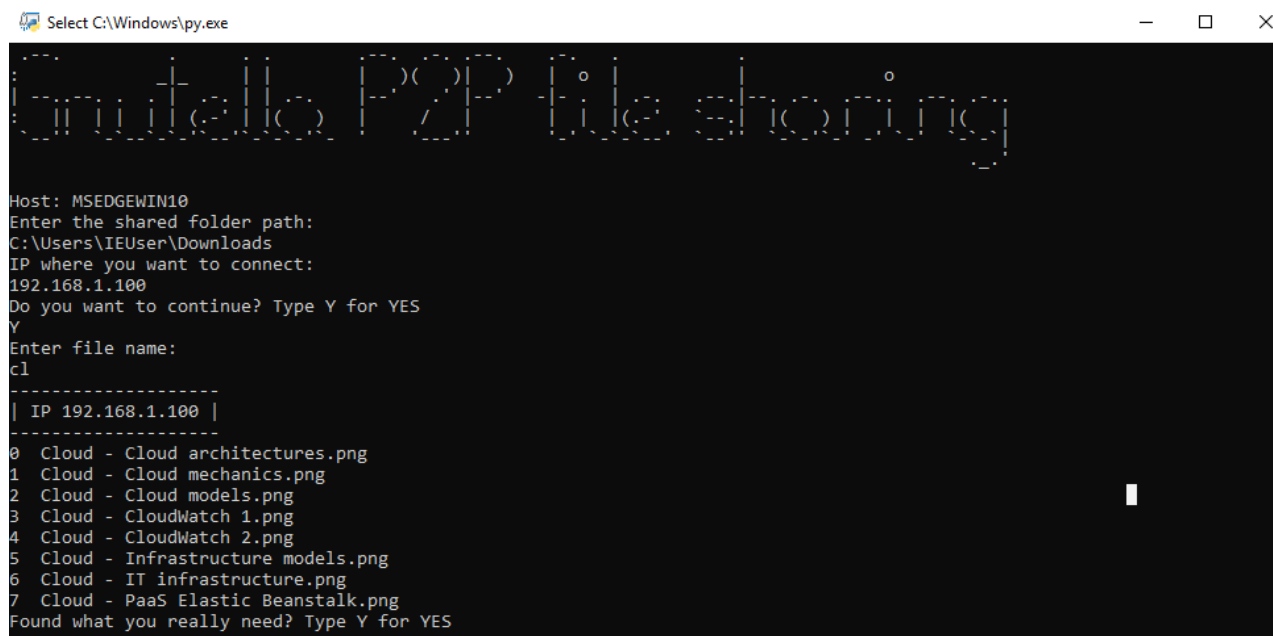
Future work

We were already aware of the fact that this is not really the traditional Gnutella protocol, as in that one the query is forwarded with a TTL, however we wanted to come up with our own solution. Anyways, we don't know how the system would react in a network with a high number of nodes and thus may lead the system to be unpredictable in case of a complex structure: that's why in the future could be possible to test the network with a bigger number of peers; for now, the tests were conducted with usage of 3 physical computers and 2 VMs, limiting so the number of nodes to 5.

Definitely, the application will also need further optimization to the ping and query timeouts and to the TTL parameter, which need a sort of "manual tuning" according to the network structure and size.

Screenshots:

Query example:



```
Select C:\Windows\py.exe

Gnutella P2P file sharing

Host: MSEdgeWIN10
Enter the shared folder path:
C:\Users\IEUser\Downloads
IP where you want to connect:
192.168.1.100
Do you want to continue? Type Y for YES
Y
Enter file name:
cl
-----
| IP 192.168.1.100 |
-----
0 Cloud - Cloud architectures.png
1 Cloud - Cloud mechanics.png
2 Cloud - Cloud models.png
3 Cloud - CloudWatch 1.png
4 Cloud - CloudWatch 2.png
5 Cloud - Infrastructure models.png
6 Cloud - IT infrastructure.png
7 Cloud - PaaS Elastic Beanstalk.png
Found what you really need? Type Y for YES
```

Download example:

```
Found what you really need? Type Y for YES
Y
Enter the IP address from where you want to download:
192.168.1.100
Enter the name of the file you want to download:
Cloud - IT infrastructure.png

Data has been received successfully!
Do you want to continue? Type Y for YES
Y
Enter file name:
jpg
No peer sent a response to your Query..
Do you want to continue? Type Y for YES
N
-
```