Online Chat Assignment - Python

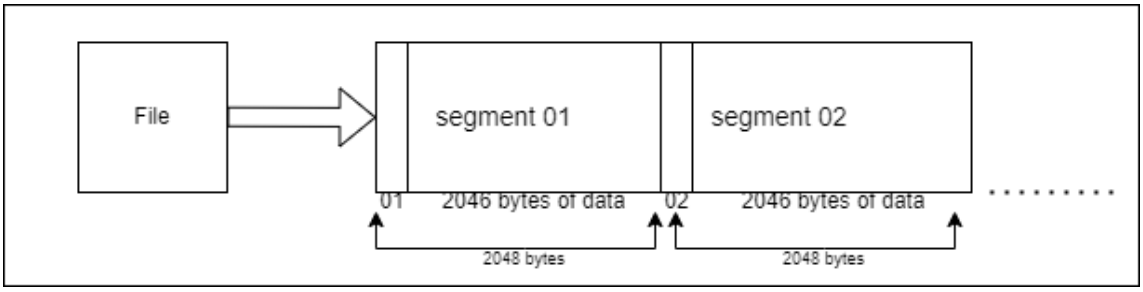
This project revolves around implementing an online chat including reliable file transfer over udp using the knowledge we aquired during the Networking Course in Ariel-University.

Introduction

First and foremost we implemented our online chat room which is a server-multiple client application. The chat room allows sending and receiving messages over TCP connections. Each user may send messages to the entire chat room or private messages to other specific users. Furthermore, we were tasked with implementing reliable file transfer over UDP. Reliable Data Transfer (RDT) over UDP (User Datagram Protocol), in our case data being files. The reliability is achieved using Selective Repeat Protocol - SRP. Unlike TCP, UDP is an unreliable data transfer protocol in the transport layer. Our goal is transferring files fast and reliably over UDP connections. Required Libraries - Select.

Selective Repeat Protocol - SRP

By requesting a certain file the server transmits the relevant data for the download proccess, within that data the client finds the file size, he then knows to expect filesize/2048 amount of **different** packets heading his way. Each packet has that Sequence Number header, that way the client reveals which packets he gets and which packets are still missing. Using SRP protocol the client and server are communicating back and forth accordingly until the client receives every packet sequence starting with 01 up to the amount of total packets.



Reliable UDP

- We begin with the client sending the server the file-request command.
- The server then sends the relevant data for the transfer process (aka File size, Available port, etc)
- Based on the data received from the server, the client then knows how many segments of data to look forward to.
- The server breaks the File's data into data segments. each segment consists of 2 bytes that represents the segment's sequence number (e.g 00,01,02,...,99) and another 2046 bytes of the file's data.
- According to the number of required segments, the client then sends which missing packets are needed to completely download the file.
- To which the server responds by sending back all of the missing segments that were sent by the client's request.

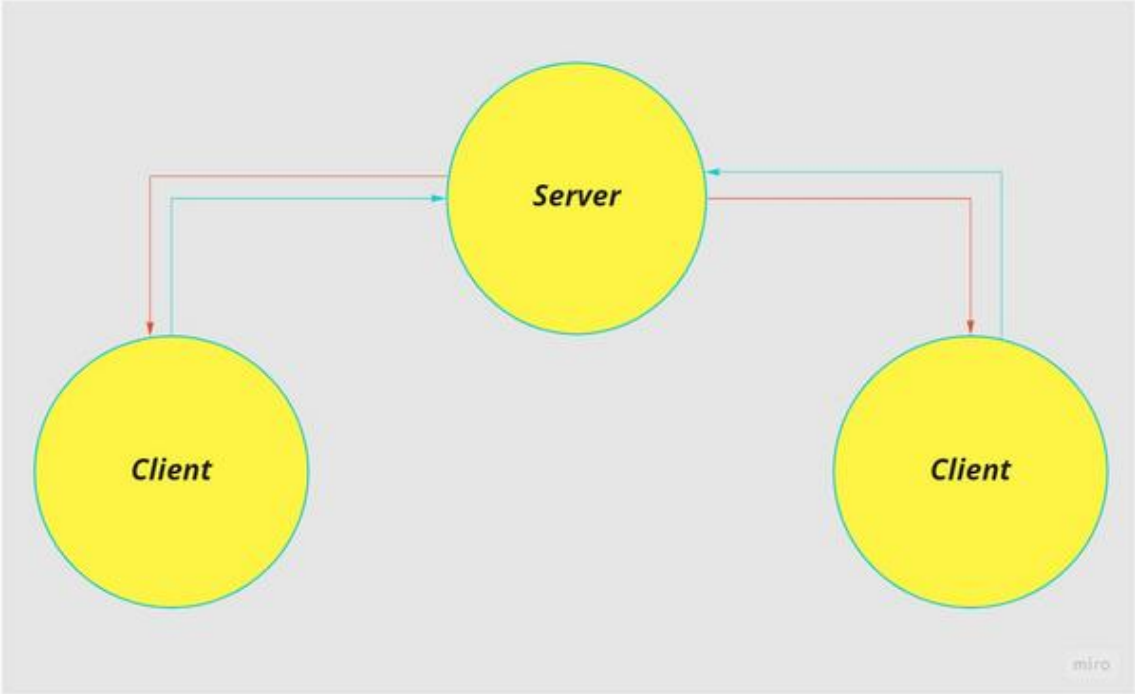
- This method prevents packet lose because the back and forth between the client and the server continues until the client recevied every segment.
- The two above stages works in a loop up until the client received all the data needed, when that happens the client sends 'check' message to the server to finish the download process.

Latency

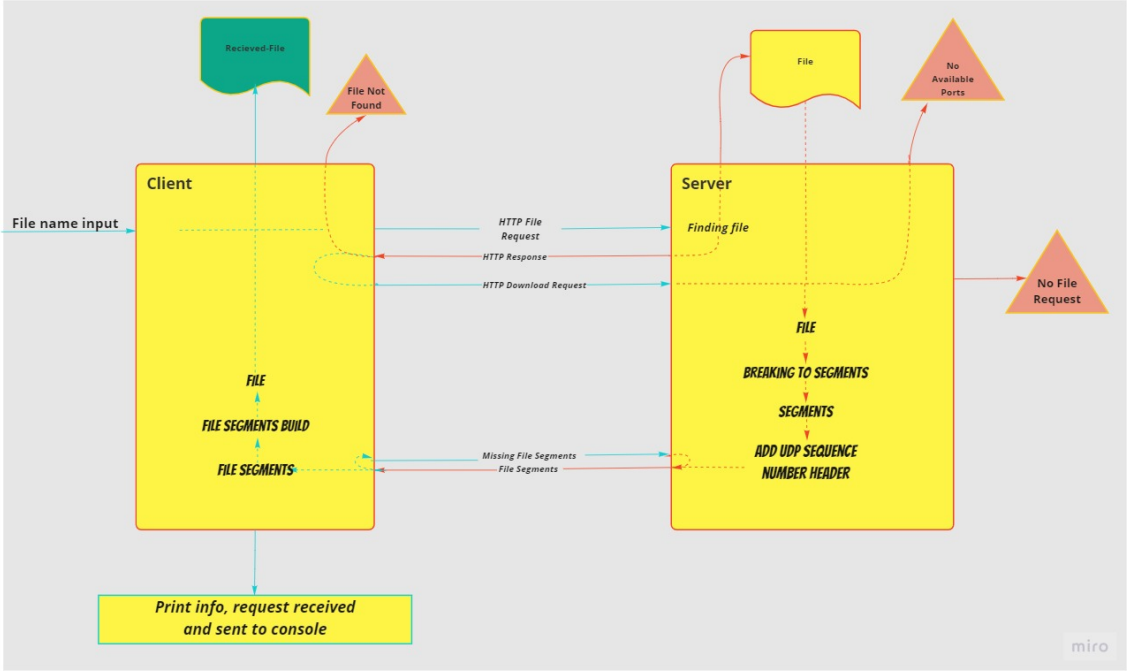
The issue we had with latency was: missing packets messages were chained together as the client sent more requests then the server recevied, this caused the server to receive multiple packets requests at once which caused KeyErrors and wrong packet numbers.

Diagrams

Messages Diagram:



Download Diagram:



Features

- Sending messages to all existing members on the server or to one or more specific members.
- Downloading existing files from the server.

Deployment

- Download the Code
- To run on Windows:

```
To run on localhost ignore the ip related instructions and enter 127.0.0.1 /
127.0.1.1 when requested to enter ip.
```

- Run Server.py.

```
PS C:\Users\97250\Desktop\MSN_Project> cd src
PS C:\Users\97250\Desktop\MSN_Project\src> python3 Server.py
---Server is Running---
Server IP - 192.168.138.1
```

- Open cmd (on the server's computer), type ipconfig and look for your IPv4 address under Wireless Lan Adapter.

```
Wireless LAN adapter WiFi:

   Connection-specific DNS Suffix  . : Home
   IPv6 Address. . . . . . . . . . . : 2a10:8006:3e1f:0:d94f:3f17:dc87:8ad1
   Temporary IPv6 Address. . . . . . : 2a10:8006:3e1f:0:18ef:aeee:26ad:ef4a
   Link-local IPv6 Address . . . . . : fe80::d94f:3f17:dc87:8ad1%10
   IPv4 Address. . . . . . . . . . . : 10.0.0.8
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : fe80::2b8:c2ff:fe3b:c020%10
                                       10.0.0.138
```

- Run Client.py (from any computer on the wifi network).

```
PS C:\Users\97250\Desktop\MSN_Project> cd src
PS C:\Users\97250\Desktop\MSN_Project\src> python client.py
Enter Server IP - Should be something like '10.0.0.8' -
```

- Enter the server's IPv4 address.

```
PS C:\Users\97250\Desktop\MSN_Project> cd src
PS C:\Users\97250\Desktop\MSN_Project\src> python client.py
Enter Server IP - Should be something like '10.0.0.8' - 10.0.0.8
You are connected from:('10.0.0.8', 63130)
Enter your user name:
```

- Enter your username.

```
Enter your user name:Lior
Thanks for signing in - you can chat now
---Client Manual---
To send a message use the syntax '@username:msg' or '@all:msg'
!users - get a list of current online users
!files - get a list of server files
!request 'file name' - make sure to enter full name including suffix. Example '!request dog.jpg'
!download 'save as name' - to download, make sure you request the file. Example '!download dog_copy
---Client Manual---
```

```
Make sure to run the server before the client.
Make sure to cd to src folder before running Server/Client.
On Linux the proccess is the same except the ipconfig command which is different, if
you dont know the linux version of ipconfig, google it. :)
```

Usage/Examples

Commands example:

```
!users
lior
!files
'dog.jpg', 'house_lo.mp3', 'player1.gif', 'textfile.txt'
!request dog.jpg
file - dog.jpg request received, to download use !download 'save as name'
!download dogcopy
```

https://user-images.githubusercontent.com/92747945/156892358-0d783984-f260-4715-a8a4-b65552770cd8.mp4

Authors

- @lior2k
- @chai9l