UDP Peer to Peer Secure Data Transfer

How to run the code

- Prerequisite: Python 3, Node JS, npm, pip
- You can do so by running the below commands on your ubuntu systems

```
sudo apt update
sudo apt install python3
sudo apt install python3-pip
curl -fsSL https://deb.nodesource.com/setup_14.x | sudo -E bash -
sudo apt install -y nodejs
```

Instructions for the sender and receiver

1. Clone the repository https://github.com/DatafoundationSystem/UDP-Peer-to-Peer-Secure-Data-Transfer_Ilomolo
2. Extract the file
3. Open a terminal and navigate to 'Frontend' folder and run `npm i`.
4. Run the Frontend by executing `npm start` command.
5. Go to Backend folder and run

```
pip install cryptography fastapi uvicorn
```

5. Run the backend by executing `uvicorn middleware:app --host 0.0.0.0 --port 8000 --reload` in backend directory.

The website will open on your browser on localhost:3000 while the port for the backend is 8000

Replicate the above steps on another device and now you will be able to send and receive data within these devices.

- Multiple receivers can be present at once for one sender.

Instructions for the server

1. Copy the server.py file present in the github repository and run it on the system you will be using as the server by running `python3 server.py`
2. Copy the IP of the server and paste it on the server_ip.txt file present in the backend folder on both sender and receiver

Some important points

- The instructions are given wrt Ubuntu OS.
- It is assumed that all the receivers, sender and the server are under same network (different IP is fine)

- In some ubuntu devices Firewall disables the ports being open, in that case you need to run `sudo ufw disable` before running the code and after the transfer enable it again by running `sudo ufw enable`
- In some windows systems to run the backend below code is needed to be executed `python -m uvicorn middleware:app --host 0.0.0.0 --port 8000 --reload`
- To change the buf size you can use the below code to check the maximum buffer size that can be allowed in your case `ping -f -l <packet_size> <destination_ip>` for windows `ping -s <packet_size> -M do -c 4 <destination_ip>` for ubuntu
- Mac works similar to Ubuntu

Errors while running the UDP Transfer code and how to handle

1. If you get this error at any point just run the code again

```
Traceback (most recent call last):
  File "/Users/akhilgupta/Downloads/git/dfs_server_receiver.py", line 384, in <module>
    receive_data()
  File "/Users/akhilgupta/Downloads/git/dfs_server_receiver.py", line 133, in receive_data
    sender_details.append(line.decode('utf-8'))
UnicodeDecodeError: 'utf-8' codec can't decode byte 0x8a in position 64: invalid start byte
```

This means that there might have been an issue with sending data from client side. Try restarting the front end or reinstalling it. This error occurs when there is an issue with decoding data from bytes to string. It might occur due to different encoding used or corrupted data.

About the project

Overview

This project was developed as part of my Data Foundation System course. We had to implement high speed UDP Peer to Peer Transfer of files with no middleman to store our files.

The project involves creating customized files for the sender and the receivers having their data including their IP addresses, PORT in which they will be receiving and transferring, Room password so that no intruder can come and receive the data. Some more additional data of the sender like File size, file name, number of threads, buffer size are also stored.

Requirements

- High speed transfer of data from sender to receiver
- Using FastAPI for the backend
- Encrypting the data before sending across the network

- Using parity check for checking the received packets and resending if its corrupted
- Using hashing for checking the integrity of the file
- Transferring files of any extension .jpg, .mkv, .mp4, .zip etc
- Constant update about the progress of the file transfer
- Having multiple receivers in the queue for the same file
- Interactive UI interface for the sender and receiver
- 24/7 working server

Software Requirements

- Frontend is coded in React JS while the backend uses FastAPI, Python
- Server uses Python
- Several libraries are used
  - socket for creating sender and receiver sockets
  - threading to implement multi threading
  - hashlib for hashing of the file
  - cryptography for using AES encryption for security

Hardware Requirements

- This requires atleast three systems. One will be acting as a clent who will be having the main file. Second will be the client for receiving that file while the third one will be the server which will help to facilitate the transfer of the data betweeen them.
- A good internet connection, preferably a strong WiFi connection or 5G mobile network.
- System with good performance (preferably with 4 cores or more) since receiving packets at a high rate requires high performance.

Workflow of the system

1. After setting up all the three systems and running them the sender will send a file with the below details in it.
   - ip - IP of the sender
   - file_size - The size of the file to be sent
   - file_name - The name of the file
   - key - Decryption key
   - n_threads - Number of threads used to trnafer the files
   - buf - Size of the buffer for sending and receiving the packets
   - port - Port in which sender will be listening
   - flag - telling if its a sender or a receiver
   - password - unique for a room

- room_cap - Capacity of the number of receiver for one file

This file will be sent to the server.

2. Now the receiver will send a file having its own details to the server having the below data
    - ip - IP of the receiver
    - port - Port for listening
    - flag - to tell if its a sender or a receiver
    - password - to tell which room it wants to enter
3. Now the server will use the Sender's IP to send the receivers (can be multiple) details to the sender
4. Server will send the sender's details to all the receivers in that room
5. Now sender will create the file_hash and send it to the receiver to be checked later
6. Sender will create chunks of the file of buf size and then dividing all that chunks and giving equal number of chunks to all the threads
7. Each thread will send the packets of buf size to the receiver having a unique sequence number to identify the packet.
8. Sender will add a parity bit to the packet to check if its not corrupted.
9. The packet will now be padded to the buffer size and encrypted using AES cipher to security reasons
10. All the packets will be received randomly to the receiver which it will store.
11. Now receiver will find sequence numbers of all the packets which are missing/lost or corrupted and will send those sequence numbers to the sender and then wait for them to be received.
12. Sender will wait to get the signal if any packets were lost or not.
13. If they were lost it will receive the sequence numbers and try again else sender's work is done.
14. The receiver will sort the packets and create a file to get the full received data.
15. The hash will be checked to check the integrity of the received file.
16. If the hash matches then the transfer is successfully completed
17. The server will wait for yet another sender to come.

Use cases for this project

This can be useful when you want to share large files over the internet without worrying about the bandwidth usage as it uses only small amount of bandwidth.

- This can be used to transfer movies across nations to reduce the cases of piracy and losing any information
- Scientific data between different institutes can be sent via this technique
- Companies, big industries can use these customized file transferring technique to reduce any information leakage

Schedule

Task	Started	Status
Understanding prev code	September	Completed
Requirement Doc	1st week October	Completed
Code from scratch	Mid october	Completed
Frontend	End of October	Completed
UDP basic code	End of October	Completed
Multithreading	November	Completed
Parity check	November	Completed
Encryption and Hash	November	Completed
Flask to FastAPI	November	Completed
Testing	December	Completed
Multiple Receivers	December	Completed
Final testing	Decemeber	Completed
Final Documentation	December	Completed
Github Update	December	Completed