Online chat

Introduction:

The project represents a multiplayer messenger app(using TCP), while at the same time the possibility of reliable transferring files (RDT) Over UDP.

The project shows our capabilities in using sockets, unittest, object-oriented programming, and GUI.

The application should consist of a client program and a server program. The server program controls every request and send back the response to specific client, or to all online sockets(e.g., send a message to all the chat) and even transfer files using A protocol I created myself inspired by famous protocols that will be explained later.

The code was written in the familiar MVC design from design patterns (Model, View, Controller)

Explanation about the protocol-

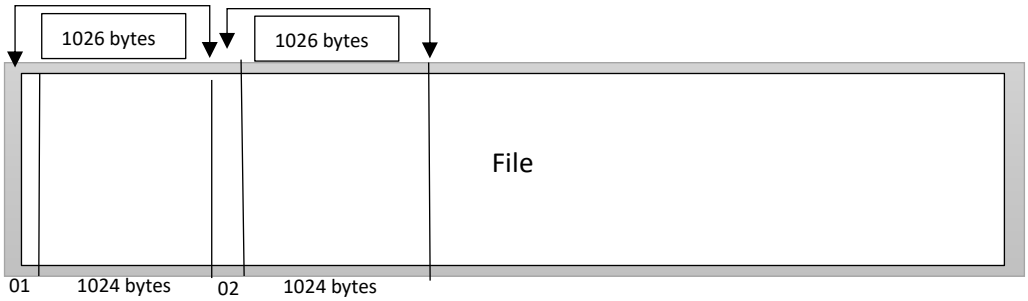
The client can ask the server which file it can transfer, the server responds to this request, and now the client can ask the server to download a specific file from the list of files.

The server has a method that divides the file into sections (we select 1024 bytes per segment (for example, 3kb/1024bytes = 3 segment, and the last segment is less than or equal to 1024 bytes)) and a unique number for each segment. The server starts sending it for the client (in a specific format that I did to easily unload it on the client server (for example, ID: Bytes)).

When the client has finished receiving the files, he repeats the receiving structure and checks what is received and what is lost, and then sends back to the server that he has lost several packets (e.g. a dose with serial number :3, 11,20).

The server receives a request to send back all the missing packets, the client check it again and when it is finished receiving all the packets it sends back to the server that it has finished receiving and then the connection between them is gone.

This is how the file should look like after the process



Explanation about the classes:

Server.py:

- **Server (Controller):**
  - Variables:
    - viewContoller : GUI, Pygame
    - serverTCP: TCP Socket
    - serverUDP: UDP Socket
    - clientList: {sock: Client()} Data structure that represent all online clients
    - fileList: Data structure of all files in server dictionary (max size of file to download is 64kb)
    - portUDPList: Data structure to organize all UDP ports for transferring files
  - Method:
    - Run(): Start the program

- **viewController (Visualization):**
  - Constructor:
    - Buttons: Start server, Exit server
    - Labels: Start server, Exit server
    - Rectangle: Surrounding the caption
  - Method:
    - Drawscreen(): Activates the draw method on all objects

- **Classes (Model):**
  - Client: (name, address)
  - ClientList: {connection: Client}
  - FileList: {filename: filesize}
  - PortUDP: (isUsed: bool, client_addr: (ip,port), sock: socket)
  - PortUDPList: {port: PortUDP}
  - ReliableUDP: This class arranges the data to before moving the file

- **Methods:**
    - SendFileUDPReliable(): Sending a file using the method I created to reliably transfer files
    - FileToFrames(): Breaks the file into packets with serial number for each packet
    - Handle_call(): Handle all the request and response calls

Client.py:

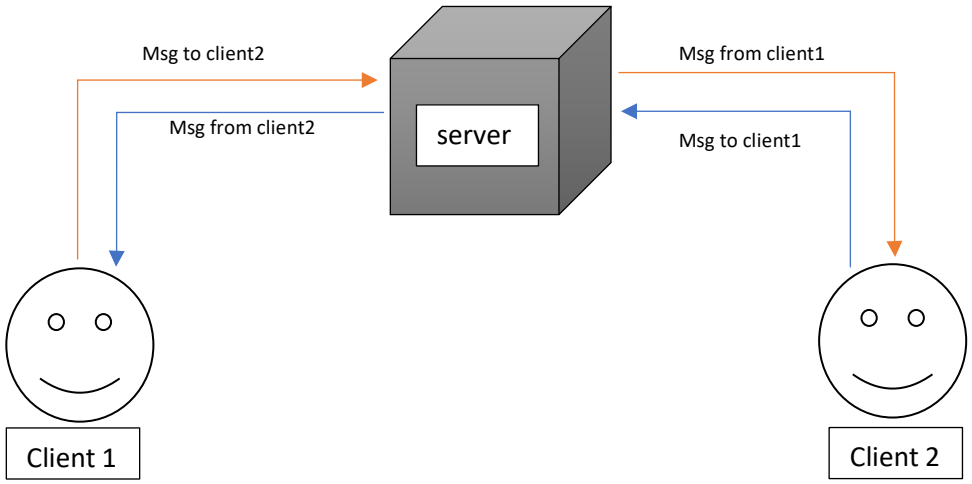
- Client (Controller)
    - Variables:
        - socketTCP: TCP Socket
        - socketUDP: UDP Socket
        - viewController: GUI, pygame
        - name: The name written in the username text field
        - messageList: List of messages (‘client0’: ‘Hello’)
    - Method:
        - Run(): Start the program

- **viewContoller (Visualization):**
    - Constructor:
        - Buttons
        - Labels
        - Rectangles
        - InputFields: e.g., usernamefield, addressfield,
        - ChatWindow
        - DownloadBar
    - **Method:**
        - Drawscreen(): Activates the draw method on all objects

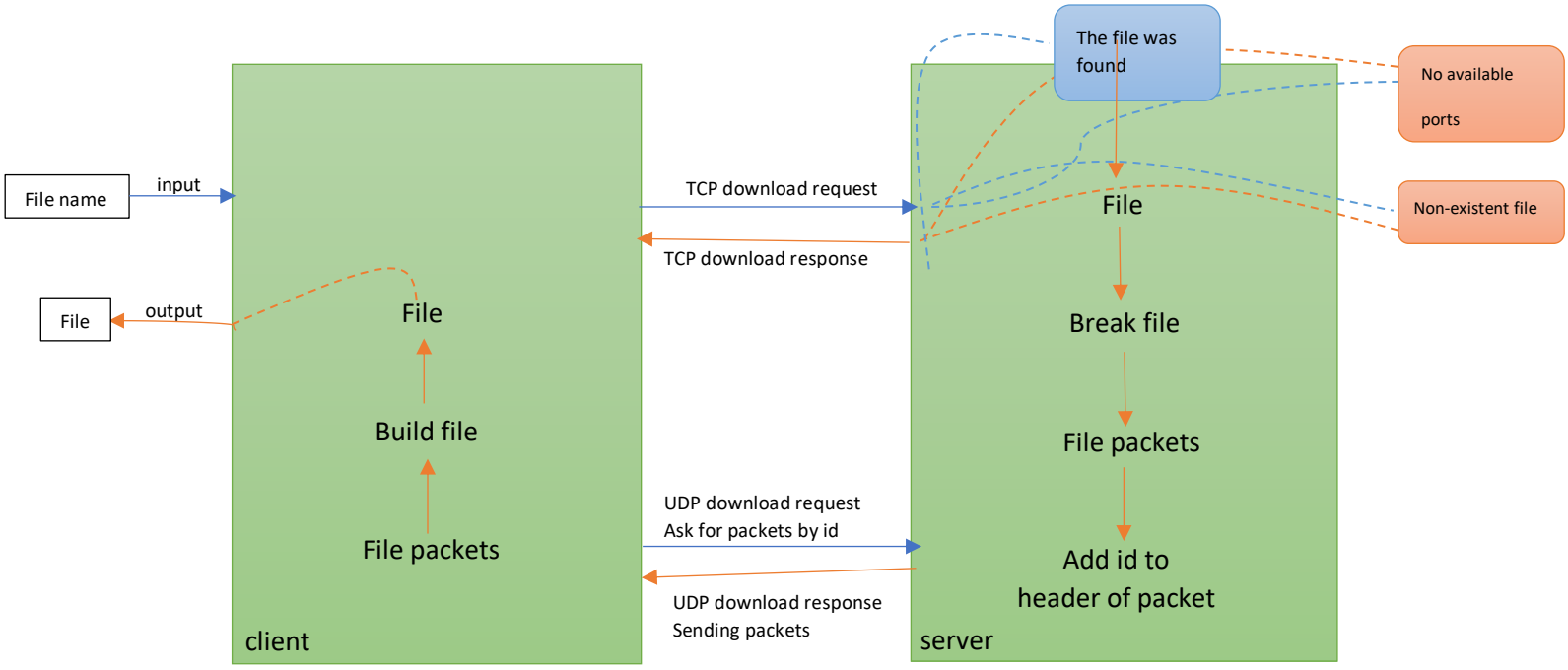
- **Classes (Model)**
    - Message: (name, message)
    - MessageList: List of messages
- **Methods:**
    - requestTCP(): handle with request from client to server
    - responseTCP(): handle with response from server to client
    - fileToFrames(): Build a data structure by given numOfPackets and fileSize, but the value initially applied there is None
    - receiveFileUDP(): This method handles reliably transferring the file from the server, in a separate process

Diagrams: (Part B)

How the messages system work:



Process of download file:



How a system overcomes package loss:

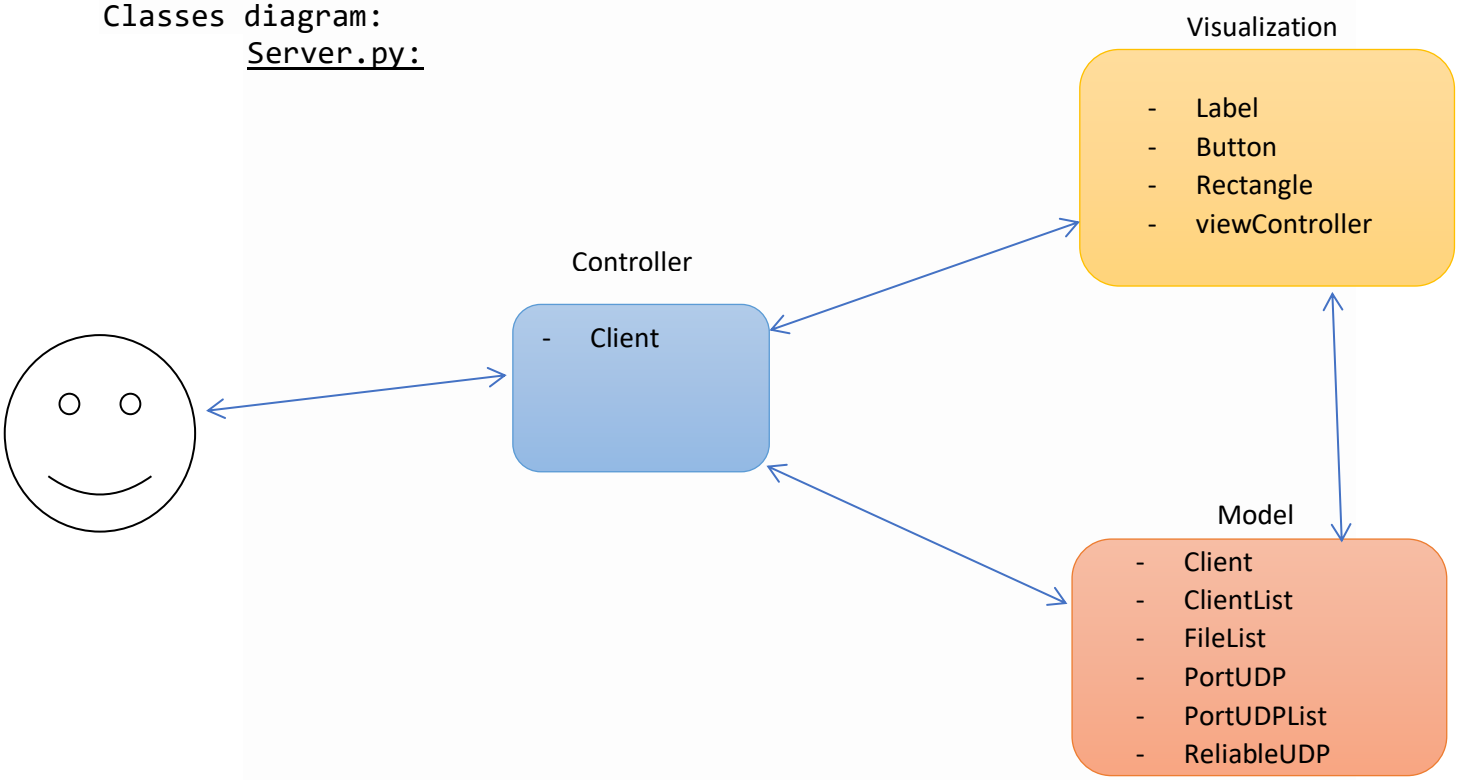
- The client wants to download a file from the explorer
- The server checks to see if the file exists, so the server sends back to the client details about the file such as the number of pacts, the file size, etc.
- Immediately after the message is sent to the client , the server begins the process of dismantling the file into 1026-byte sections (2 bytes for id, 1024 for data)
- The first 2 bytes in each segment represented a serial number per segment (e.g., 00,01,02…)
- After you break the file into small parts, The server begins the sending process to the client
- After each segment received from the server to the client
  The client sends back ACK with the serial number (e.g. ACK01)
- The server fills in the bool data structure of each segment and marks correctly or incorrectly
- After you send the entire file. The server goes through its bool data structure, checking whether it is marked that no ACK is received for one of the pacts, and immediately sends it back
- The client always goes through the data structure that contains all the packets, if after sending the file it is found that one of the packets is None, the client requests that the server send the specific packet again
- This process is repeated over and over again until the file is fully received

How the system overcomes latency problems:

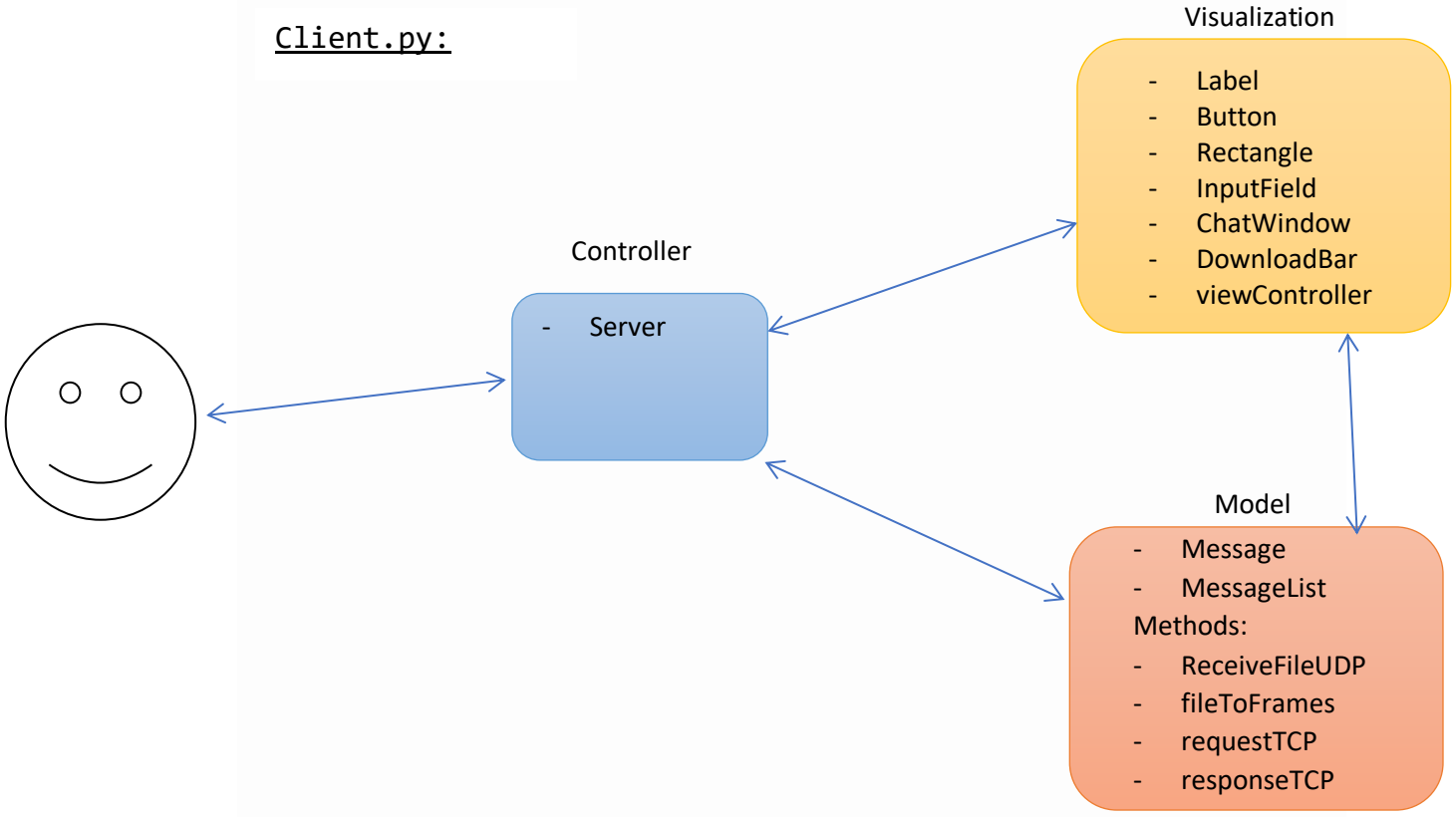
- The method we chose, we encountered a delay problem that caused the system to crash and other problems.
  To address this issue, we had to run several conditions in the system to avoid further delay issues.

Classes diagram:

Server.py:



Client.py:



Requirements:

- Math
- Threading
- Pygame
- Socket
- Select
- Time

How to use:

- Download the code
- Open cmd on the same directory as the project
- Run ‘python3 server.py’
- Then click on ‘start server’

- Run ‘python3 client.py’
- If you on the same pc, then write the address that appears in the window of the server
  If you run the client on separate pc , then ask the guy who ran the server for IP address
- After enter IP address and user name press ‘Login’ button

Video:

ChatApp-How to use

Authors:

@Orcohen33 , id-206585515
@Shlomi-Lantser id-209322379