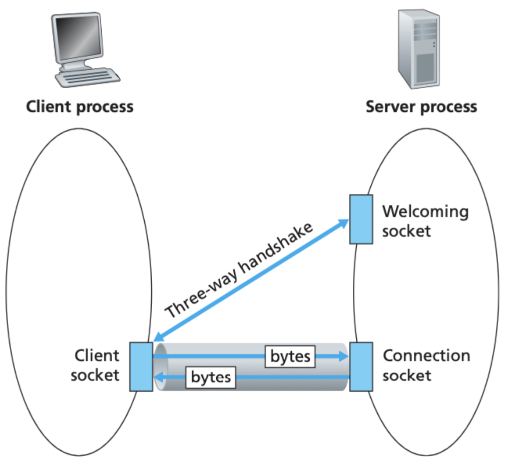
3D3 Computer Networks
Project 2 report

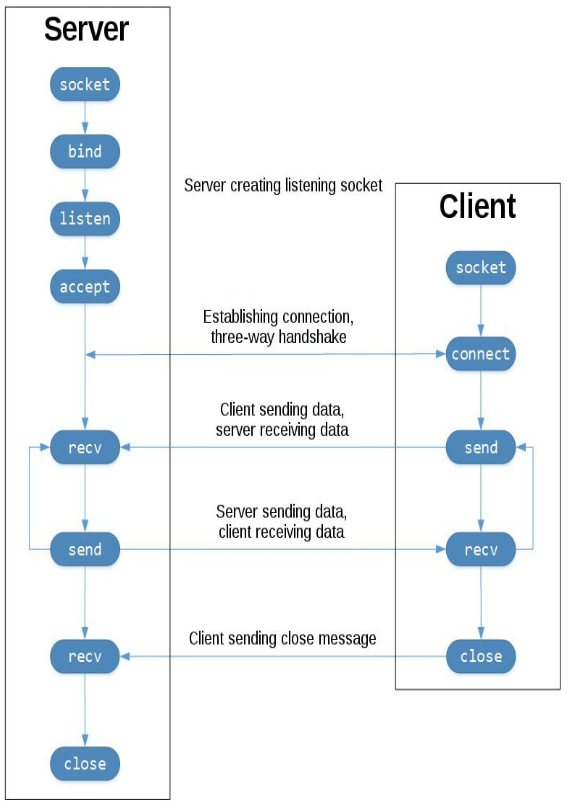
Karan Rajiv (20303100)
Siddharth Lala (20315464)

Use Case

We have seen so many wars throughout our lives and how it takes a toll on the entire country's resources and infrastructure. Taking reference from the recent calamities that are the earthquake in Turkey and Syria as well as the war currently taking place between Russia and Ukraine. These tragedies destroyed the respective country's infrastructure and in such situations, the communication channels also get disrupted. This idea has been selected as it aids in facing issues to assist humanity in conflict zones. The purpose of this project was to help the soldiers to communicate with different base camps and inform them about any movement across the border. The plan of this project is to help communicate with other people easily without the use of complex connections with the server. Subsequently, we plan to develop the sending of encrypted messages between the soldiers to maintain the privacy of any sensitive data.

Picture / Diagram / Logical Diagram




Server
socket
bind
listen
accept
recv
send
recv
close
Server creating listening socket
Establishing connection, three-way handshake
Client sending data, server receiving data
Server sending data, client receiving data
Client sending close message
Client
socket
connect
send
recv
close

High-Level Messaging / Functionality / Functional description / Purpose

This model utilises peer-to-peer design which is significantly beneficial since it formulates direct communication between the hosts. TCP has been chosen for the model since it is connection-oriented and has higher reliability. It also has more advanced security features in comparison to UDP which is important for the selected test case since it involves the exchange of sensitive information among soldiers and base camps.

TCP socket programming uses a client-server architecture. For communication between two hosts using TCP, a connection is established prior to data exchange. A three-way handshake is used to form this connection. The server first creates a socket on which it will wait for an incoming connection request from the TCP client. This is the welcoming socket. The client also creates a socket specifying the server's name and port number to which the socket is going to be connected. The creation of this socket by the client leads to a connection request message which is sent from within the transport layer.

Algorithm / Pseudo Code / Structure / Logical thinking

First set up the server connection

```
Import libraries
Main()
Set up the server host and port number
Create the server socket
Sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.bind
sock.listen

once the socket is created
print server is waiting for connections
create new thread to wait for connections
.thread()
Start()

Declare the setup variables for holding information about the connection
    Connections[]

Class name 'client' which has the socket and address that is associated with the items along
with an assigned ID and a name chosen by the client.
    Define the class variables – socket, address, id, name, signal



Get data from the client
If unable to get data, assume client has disconnected and remove him from server data
If able to get data from the client, print it in the server and send it back to every client
node.
    If data received from the signal
        Decode the message (.decode('utf-8')) {this converts the byte data into a
printable string}
        Print the data on the server
        Encode data and send it to clients

Set up for client connection

Import libraries
Receive message from server
Def receive
    Data is received
    Decode the data using .decode('uft-8')

    Print the data

Set up the host address and port number

Connection to server
Try:
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect(host address, port number)
print connected to server

except:
    print could not connect to the server
exit

create a new thread to wait for new data
threading process
.thread(target, args)
Threading start()

Send data to server
While true:
    Message= input is received from server
    print message using .encode(message)
else if message is exit
break
```

Networking / Comms model

The prototype constructed involves each server dealing with multiple clients. However, every client interacts with only one server. Hence it portrays a one-to-many form of communication.
The model also contains the flow control feature supported by TCP. This feature ensures that the volume of data sent by the sender to the receiver doesn't exceed the receiver's buffer.
It also features congestion control which avoids the overfilling of packets at various nodes.

Implementation details

The code is written in python3 script. To implement this idea, we choose to set up a tcp transport layer protocol with client server program. TCP is a connection-oriented protocol. This means that before the client and server can start to send data to each other, the client first needs to establish a connection with the server. One end of the TCP connection is attached to the client socket and the other end is attached to a server socket.
For this project we have worked on tcp layer socket programming to implement the idea. In tcp socket programming, a socket is an endpoint of a two-way communication link between two computers on a network. A socket is identified by an IP address and a port number and is used to send and receive data across the network.
Before sending any messages, a connection must be established and to do so we need to create a socket on the server and the client side. The client socket initiates the connection by sending a request to the server socket, which accepts the request and creates a new socket to communicate with the client. To create a socket, we have used AF_INET (IPv4) domain and the domain type is sock_stream.
After the connection is established, the client can send messages to the server. In TCP socket programming, messages are sent in the form of byte streams, which means that text messages need to be converted into byte arrays before sending. The client creates a buffer to hold the message and writes the message into the buffer. Then, the buffer is sent to the server using the socket connection.

On the server side, a buffer is created to hold the incoming message. The server reads the incoming data from the socket connection and stores it in the buffer. Once the entire message has been received, the server can convert the byte array back to a string representation of the original text message.

After all the messages have been sent and received, the connection needs to be closed. This is done by sending a termination signal to the other side of the connection. Once both sides have acknowledged the termination signal, the socket connection is closed.

After creating a socket, we bind the socket to the local address and port number of the operating system. Binding a socket ensures that incoming connections are directed to the correct socket.

This application of send and receiving messages has no data loss and the throughput is elastic.
Highlights/Unique Selling Points of your system/implementation.

Our model supports interaction between multiple clients through the server which is a unique selling point for our prototype. It is not restricted to a single client-server relation.

Changes, advances and/or improvements from Project 1

For project 1, Siddharth's idea was to use satellite communication to communicate with people from different regions. Similarly, Karan's idea for project 1 was to use a satellite mode of communication to send and receive encrypted messages. The setting for it was how soldiers send messages from one base camp to another about the movements around the border. After the first pitch, we soon realised that we cannot use satellite communication in a war or conflict setting since the infrastructure for that is not feasible.

Bibliography

https://stackoverflow.com/questions/55661626/with-socket-socketsocket-af-inet-socket-sock-stream-as-s-get-error-attribut

https://pythonprogramming.net/server-chatroom-sockets-tutorial-python-3/

James W. Kurose, Keith W. Ross - Computer Networking_ A Top-Down Approach-Pearson (2021)