NSSA290-06
Documentation

Brandon Mok, Tyler Pache, Weibin Yang

Overview:

The TCP/UDP program provides the capability and functionality of allowing multiple client(s) and server(s) to exchange messages using the allowed communication methods: tcp or udp. The method of communication specified will create an instance of the respective file (i.e. TCPClient, TCPServer, UDPClient, UDPServer, etc.) and will handle the connection and data transfer as per method: i.e. if wanting to have a server communicating over a tcp connection, the main NSSA290Server file will create an instance of a TCPServer class file. It involves the startup of both a client and a server program that's accompanied by a "readme" file that contains information and instructions on running the program for the multithreaded server that can accept multiple communications at once. The client can send messages to the server with the server echoing back the received message alongside a timestamp of the data's arrival.

If communicating through a TCP connection, the method involves being connection oriented (IPs with sockets), slower mode of data transport due to a 3-way handshake of "syn," "ack," and "fin" messages, and possesses a beneficial feature of reassuring that the sent data does indeed get received. In the program, the client and server both contain a command-line interface requesting the necessary information to allow proper communication. Afterwards, the server opens a server socket for the client programs to connect to utilizing a socket, specified port, and IP address of the server. Once the port/socket is open, it remains in this state until the server is manually closed or the client enters "exit" to terminate the client program's connection. While a client is connected to the server it can send multiple messages. The server will record the connections of each new client (i.e. [timestamp][IP] connected!), the client's message ([timestamp][IP address] [message]), and then reply with the time and received message (i.e. [timestamp][sent message from client]).

If communicating through a UDP connection, the method involves being connectionless, a faster mode of data transfer, and can function without an available IP address. In the program, starting both the client and server programs and following the instructions of requested information from the command-line interface, the server opens a port and a datagram socket with the client also connecting with a datagram socket. The datagram socket allows for data to be exchanged between the client and server in the form of packets (datagram packet). The server continually looks for datagram packets being sent to the server and once a packet is detected, the data is recorded on the server (i.e. [timestamp][IP address][message]). The server then takes the client's IP address, time the message was received, and the message itself into a new datagram packet to be sent back to the client that are on the same datagram socket connection.

Workflow:

Server:

1) User runs the jar file from the command: java -jar NSSAServer.jar
2) User enters communication method (TCP or UDP)
    - If neither “tcp” nor “udp” entered, user is asked until correct input is entered.
3) User enters port to listen on
    - If port isn’t between 0 and 655355, user is asked again until correct input is entered.
4) The main server (NSSAServer.java class file) creates an instance of the respective communication method server file (i.e. TCP -> TCPServer.java or UDP -> UDPServer.java)
5) Communication is now setup and ready to listen for incoming messages with TCPServer and UDPServer handling communication differently.
    - TCPServer utilizes server sockets to connect with regular incoming sockets https://docs.oracle.com/javase/7/docs/api/java/net/ServerSocket.html
        - The TCP program utilizes Scanners and PrintWriters to exchange data
        - Scanner (https://docs.oracle.com/javase/7/docs/api/java/util/Scanner.html)
        - Printwriter(https://docs.oracle.com/javase/7/docs/api/java/io/PrintWriter.html)
    - UDPServer utilizes datagram sockets to connect https://docs.oracle.com/javase/7/docs/api/java/net/DatagramSocket.html
        - Datagram packets are used to send messages https://docs.oracle.com/javase/7/docs/api/java/net/DatagramPacket.html

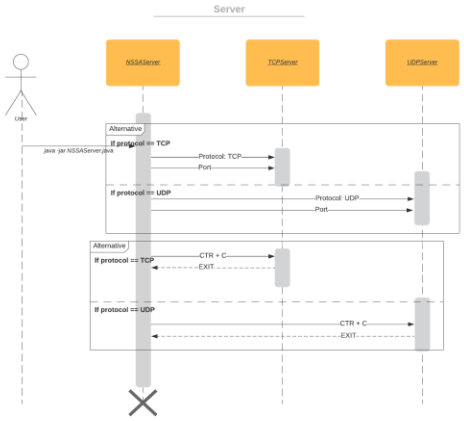

Figure 1: Instantiation of respective server

Client:

1) User runs the jar file from the command: java -jar NSSAClient.jar
2) User enters IP or hostname of the machine the server is running on
3) User enters communication method (TCP or UDP)
    - If neither “tcp” nor “udp” entered, user is asked until correct input is entered.
4) User enters port
    - If port isn’t between 0 and 655355, user is asked again until correct input is entered.
5) The main client (NSSAClient.java class file) creates an instance of the respective communication method client file (i.e. TCP -> TCPClient.java or UDP -> UDPClient.java)
6) Communication is now setup and ready to start sending messages to the server with TCPClient and UDPClient handling communication differently
    - TCPClient utilizes java sockets to communicate (https://docs.oracle.com/javase/7/docs/api/java/net/Socket.html)
        - The tcpclient utilizes Scanners and PrintWriters to communicate
        - Scanner (https://docs.oracle.com/javase/7/docs/api/java/util/Scanner.html)
        - Printwriter(https://docs.oracle.com/javase/7/docs/api/java/io/PrintWriter.html)
    - UDPClient utilizes datagram sockets to communicate https://docs.oracle.com/javase/7/docs/api/java/net/DatagramSocket.html
        - Datagram packets are used to send messages https://docs.oracle.com/javase/7/docs/api/java/net/DatagramPacket.html

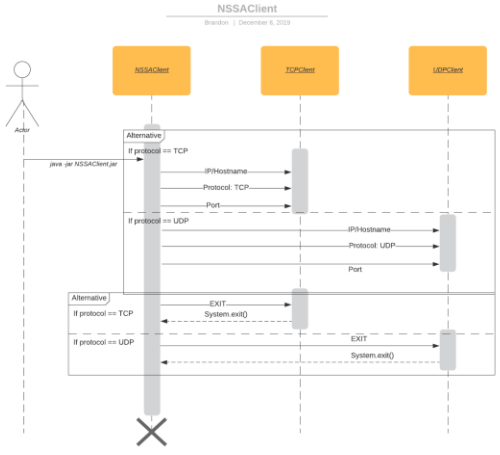

Figure 2: Instantiation of respective client

TCP Client-Server:

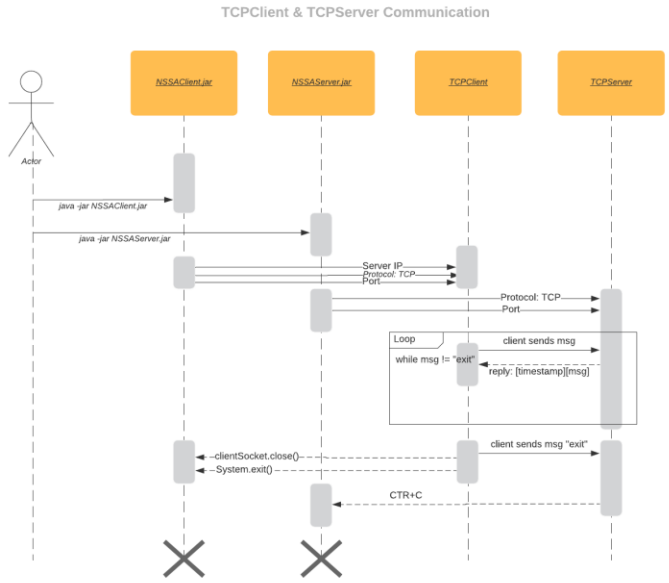

Figure 3: Communication between TCPClient and TCPServer

UDP Client-Server:

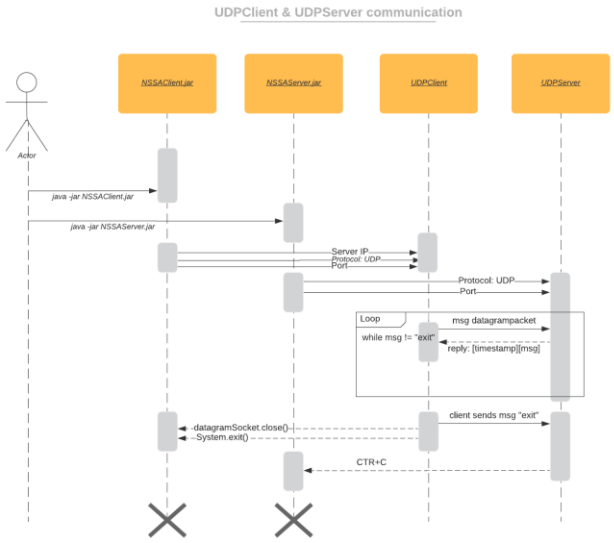

Figure 4: Communication between UDPClient and UDPServer