Hema Bahirwani

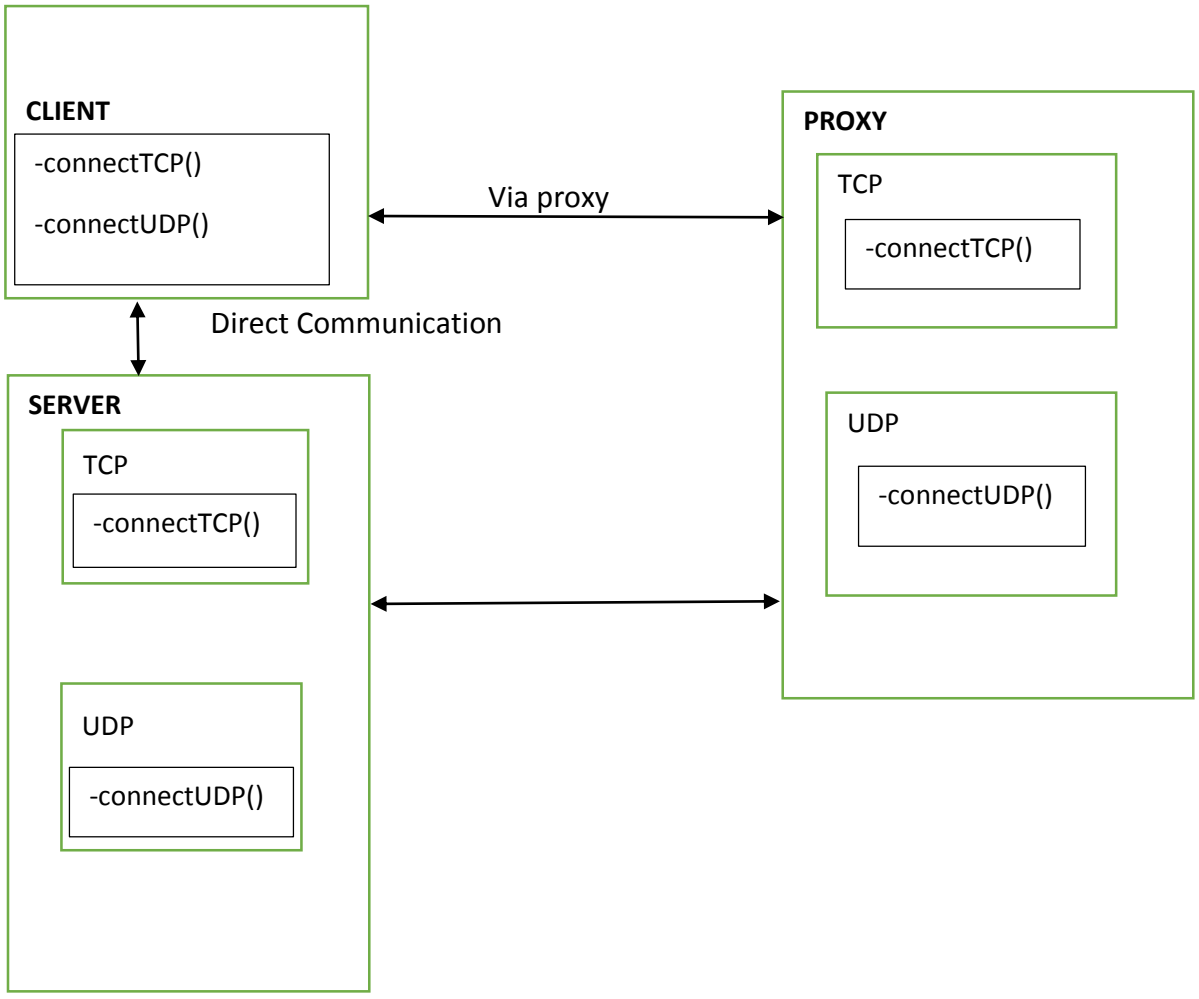
Foundation of Computer Networks

Project -1

Socket Programming

1. INTRODUCTION:

The goal of this project is to familiarize with the socket programming API and design a protocol with a client, server, and proxy server applications. The flow of the program is as follows.



The server and proxy are set up, and would be listening on both the TCP and UDP for any connections. The client would connect to either server or proxy to query time from the server. Based on the connection types, the client would connect and send the message to the server or the proxy. The total round trip time is calculated and displayed.

2. IMPLEMENTATION:

SERVER:

The server thread would be listening on both the ports, namely, TCP and UDP. In order to setup the server, the application has to mention the operator type as "-s" and set the initial time of the server. Username and password fields could be mentioned, and be used for client authentication. While setting up the server, it is mandatory to provide both the TCP and UDP ports, without which the server won't be set up.

```
C:\Users\Hema\workspace\socketProgramming\src>java tsapp -s -T 5 5000 5001
Sever!
Server: Listening on TCP: 5001
Server: Listening on UDP: 5000
```

As soon as, the server is up, it starts the threads of TCP and UDP, these two classes, connect the server on respective ports and wait for the client or proxy to connect. The server receives the message, checks if the client wants to change the set time, authenticates the client, and if the client is legit, it sets the new time as required by the client. It then sends the message back to the client with time and other fields.

CLIENT:

The client thread will connect to the server directly or via proxy. In order to set up the client, use the operator "-c", and provide the port number and server address which are the mandatory fields.

```
C:\Users\Hema\workspace\socketProgramming\src>java tsapp -c 127.0.0.5 5000
Client!
Total Round Trip Time: 225 milliseconds
Time is: 5 seconds
Number of Hops: 1
Hop 1 Address: Hema-PC/192.168.0.29 Time taken: 205 milliseconds
```

The user can specify the connection type of the client with the server as "-t" or "-u" for TCP and UDP, respectively. If no connection type is mentioned, the default is taken as UDP. The client can set the time of the server by using "-T" operator. To set the time, it is mandatory for the client to specify the user name and password, which would be

validated at the server, and if legit, the server will allow the client to set the new time. The client is also given the option to specify the number of times it wants to query the server, using the operator “-n”. The number specified after –n is the number of times the client will ask the server for time. “-z” operator enables the client to ask for the time in UTC format.

PROXY:

The proxy thread acts as the medium between the server and the client. It mediates the message to and fro from client to server and vice versa. To set up proxy, use the operator “p”, specify the mandatory fields such as the server address it wants to connect to, TCP port, UDP port, these are used by client to connect to the proxy, TCP and UDP port for proxy to connect to the server. The proxy can mention the connection type between the server and itself using operators “-t” and “-u”
For TCP and UDP connection, respectively.

```
C:\Users\Hema\workspace\socketProgramming\src>java tsapp -p 127.0.0.5 --proxy-ud
p 5000 --proxy-tcp 5001 4000 4001
Proxy!
Proxy: Listening on TCP: 4001
Proxy: Listening on UDP: 4000
```

If the proxy has mentioned any connection type with the server, it is mandatory for the proxy to mention the respective ports, else it would throw an exception. If no connection type is specified, the proxy uses the connection type between client and itself, to connect to the server.

MESSAGE OBJECT:

This class is used to define the message sent to and fro from client to server, and vice versa. This message contains information about the client, such as, username, password, if the client wants to set the time, if the user requested the time in UTC format, error, if any etc.

EXAMPLE:

Server:

```
C:\Users\Hema\workspace\socketProgramming\src>java tsapp -s -T 5 --user usr --pa
ss pw 5000 5001
Sever!
Server: Listening on TCP: 5001
Server: Listening on UDP: 5000
Client/Proxy connected via UDP
Server: Listening on UDP: 5000
Client/Proxy connected via UDP
Server: Listening on UDP: 5000
```

Proxy:

```
C:\Users\Hema\workspace\socketProgramming\src>java tsapp -p 127.0.0.5 --proxy-ud
p 5000 --proxy-tcp 5001 4000 4001
Proxy!
Proxy: Listening on TCP: 4001
Proxy: Listening on UDP: 4000
Client connected via UDP
Client connected via UDP
```

CLIENT:

```
C:\Users\Hema\workspace\socketProgramming\src>java tsapp -c 127.0.0.4 -n 2 4000
Client!
Total Round Trip Time: 341 milliseconds
Time is: 5 seconds
Number of Hops: 2
Hop 1 Address: Hema-PC/192.168.0.29 Time taken: 185 milliseconds
Hop 2 Address: Hema-PC/192.168.0.29 Time taken: 323 milliseconds
Total Round Trip Time: 12 milliseconds
Time is: 5 seconds
Number of Hops: 2
Hop 1 Address: Hema-PC/192.168.0.29 Time taken: 3 milliseconds
Hop 2 Address: Hema-PC/192.168.0.29 Time taken: 8 milliseconds
```

Here the client queries the server twice through proxy. Since no connection type is mentioned, client uses UDP to connect to proxy. The connection between proxy and server is also via UDP, same as client connection type, since no connection type is mentioned.

3. MESSAGE FORMAT

The message used in this project is as shown below:

Fields	DataType	Description
Number of Hops	Int	Number of hops it took to reach the server
Time	Int	Time sent by the server
Username	String	Username of the client
Password	String	Password of the client
Set Time	Int	Time the client wants to set
UTC	Boolean	Whether time is requested in UTC format
IP Addresses	List<String>	IP address of each hop it takes to reach the server
Timings	List<long>	Time taken to reach the next hop
Error	String	Error, if any
Client Connection Type	String	Connection type of the client, used by the proxy to determine its connection with the server

The Byte format of the message is as shown below:

BYTE FORMAT:

Fields	Description
AC ED	Specifies that this is a serialization protocol
00 05	This is the serialization version
0x 73	Specifies that this is a new object
0x 72	Specifies that this is a new class
00 0A	Length of the class name
0x 02	Specifies that the object supports serialization
00 10	Number of fields in the class
65 46 76 36 26 36 47 58 57 47	The name of the class
Number of Hops	Number of hops it took to reach the server
Time	Time sent by the server
Username	Username of the client
Password	Password of the client
Set Time	Time the client wants to set
UTC	Whether time is requested in UTC format
IP Addresses	IP address of each hop it takes to reach the server
Timings	Time taken to reach the next hop
Error	Error, if any
Client Connection Type	Connection type of the client, used by the proxy to determine its connection with the server

When we serialize an object, there are extra bytes added by the serialization algorithm. This information is needed to recreate the object at the receiver's end. The serialization algorithm writes the metadata of the class to the object and recursively writes the data of the instance.

The message contains data sent by the client, the server modifies the message as per request, and send it back to the client. It illustrates the number of hops the message goes through to reach the server. At each hop, the IP address and the time taken to reach is added to the message. These are used to calculate the delay to reach the next hop. If the client wishes to set the time at the server, the user name, password and the time to be set are stored in the message. We, altypeso, take in to account any error occurred during the process. The message also tell us, if the client has requested time in UTC format. It contains the client connection type information. This information is used by the proxy to determine its connection with the server, if no connection type has been mentioned between the two.

A part of the message explanation has been referred from http://www.javaworld.com/article/2072752/the-java-serialization-algorithm-revealed.html.