National Institute of Technology,Raipur

Department of Computer Science & Engineering

A Term Project on Network Programming

"TCP OVER UDP"

GitHub Project Link:

https://github.com/rahul2307/TCP_Over_UDP

Submitted By:

Roll no:14115017 Anushree Ghosh
Roll no:14115038 Harshil Gupta
Roll no:14115041 Inakollu Neha Priyanka
Roll no:14115053 Meenakshi Choudhary
Roll no:14115073 Rahul Baghel

ABSTRACT

In this project, we have implemented a TCP transport layer protocol with UDP socket (UDP implementation of TCP). Here we have used Java programming language for network socket programming. In this project, we are going to use UDP socket API to implement TCP.

Description: As you know, TCP includes many features to support reliable transmission, flow control and congestion control. In congestion control, we can use TCP well known procedures like slow start, fast retransmission and additive increase/multiplicative decrease. The 3–way handshake for TCP connection setup is required but special cases need not be supported. To protect packets against bit level errors, use a checksum mechanism similar to TCP. We need to take a look at TCP RFC (RFC 793) for details.

The TCP like mechanism chosen for reliable transmission in this project is cumulative acknowledgment of the last in-order delivered byte for every packet received. Pay attention that the delayed acknowledgement method is not required so you will acknowledge every packet arrived at the receiver. The sequence numbers are measured in bytes like TCP with an initial random sequence number (Note that your protocol will totally work on bytes rather than messages).

When a packet ACK times out, window size is set to 2 packets. For RTO calculation we will use the options field of TCP to send a timestamp. Note that you should also add padding to the header to align it to 32 bits and also cover the options in checksum. You should update your RTT calculation and RTO for every packet.

We will implement an API for applications to be able to use it for sending and receiving data.

TABLE OF CONTENTS

ABSTRACT 2
TABLE OF CONTENTS 3
TABLE OF FIGURES 3
INTRODUCTION 4
Motivation 4
Problem 4
PROJECT DESCRIPTION 5
Classes made 5
OUTPUT SCREENSHOTS 7
CONCLUSION & FUTURE SCOPE 9
REFERENCES 9

TABLE OF FIGURES

Figure 1 State.java's Class Definition 5
Figure 2 Packet.java's Class Definition 5
Figure 3 toString Method Definition 5
Figure 4 Output of toString Method 5
Figure 5 3-way Hand Shaking Definition 6
Figure 6 4-way Handshaking Definition 6
Figure 7 ServerSocket Definition 7
Figure 8 Output of Server Side - 1 7
Figure 9 Server Side Output - 2 8
Figure 10 Client Side Output - 1 8
Figure 11 Client Side Final Output 9

INTRODUCTION

A TCP connection is established via a three way handshake, which is a process of initiating and acknowledging a connection. Once the connection is established data transfer can begin. After transmission, the connection is terminated by closing of all established virtual circuits.

UDP uses a simple transmission model without implicit hand-shaking dialogues for guaranteeing reliability, ordering, or data integrity. Thus, UDP provides an unreliable service and datagrams may arrive out of order, appear duplicated, or go missing without notice. UDP assumes that error checking and correction is either not necessary or performed in the application, avoiding the overhead of such processing at the network interface level. Unlike TCP, UDP is compatible with packet broadcasts (sending to all on local network) and multicasting (send to all subscribers).

We have implemented a Sliding Window Protocol and simulated TCP (Transmission Control Protocol--which ensures reliable data transfer) over a UDP connection (which occasionally dropped data packets) in Java. The Sliding Window ensures reliable transmission using acknowledgements (ACKs) for received data packets on the server. Furthermore, in adding Congestion Control, it allows for reliable packet transmission and avoids bandwidth congestion by adjusting the size of the sliding window dynamically based on the level of congestion on the network.

Motivation

We were taught about TCP and UDP protocols individually in our lecture classes and were made aware of the differences in both of these protocols. We were intrigued by the idea of developing a code that would use the concepts of one protocol and convert it into the other. Hence as soon as a group was formed, we discussed this idea together and upon approval from all, we distributed the work and were all set to build the project. Thus this is how we started upon the project.

Problem

While working on this project we were faced with the following problems:

1. Implementation of Three-way and Four-way handshake was quite a challenge for us as we had studied it only in theory and had no practical experience with it. Finally through diligent efforts and a bit of help from online communities we successfully implemented it.
2. For sending and receiving string values through packets, the heterogeneity of data like flags and acknowledgement bits caused us to brain storm on how to retrieve those values from incoming string. But then we solved the issue by use of string split class.

PROJECT DESCRIPTION

Classes made

- **State.java :**In this each state is assigned a number by using enum

```
public enum State {
    NONE, SYN_SEND, SYN_RECV, ESTABLISHED, FIN_SEND, FIN_RECV, FIN_ACKD, DISCONNECTED
}
```

Figure 1 State.java's Class Definition

- **packet.java:**In this get and set methods are used to get the current status of flags like Sync Flag,Ack ,Flag,Fin flag etc.. and to set their values.

```
public boolean isSyncFlag() {
    return syncFlag;
}

public void setSyncFlag(boolean sYN_FLAG) {
    syncFlag = sYN_FLAG;
}
```

Figure 2 Packet.java's Class Definition

Append of String Builder class is used to append all the details of flags and window size in a single line in output.

```
public String toString(){
    StringBuilder builder = new StringBuilder();
    builder.append("SYNF=").append(syncFlag?1:0).append(";").
    append("SYNN=").append(syncNum).append(";").
    append("ACKF=").append(ackFlag?1:0).append(";").
    append("ACKN=").append(ackNum).append(";").
    append("FINF=").append(finFlag?1:0).append(";").
    append("WINDOW=").append(windowSize).append(";").
    append("DATA=").append(data);

    return builder.toString();
}
```

Figure 3 toString Method Definition

Output:

```
Data Recieved SYNF=0;SYNN=0;ACKF=1;ACKN=4738;FINF=0;WINDOW=14;DATA=null

RCVD: SYNF=0;SYNN=0;ACKF=1;ACKN=4738;FINF=0;WINDOW=14;DATA=null
```

Figure 4 Output of toString Method

- **Client.java** :Client Side code

```
    Packet syncPacket = new Packet();
    syncPacket.setSyncFlag(true);
    SYNC_NUM = ThreadLocalRandom.current().nextInt(1, 5000);
    syncPacket.setSyncNum(SYNC_NUM);
    send(syncPacket.toString());
    state = State.SYN_SEND;
    System.out.println("Threeway handshake 1/3.");
}
private static List<Packet> BUFFER = new ArrayList<Packet>();
private static Thread listener = new Thread(new Runnable(){
    @Override
    public void run() {
        while(true){
            byte[] buff = new byte[MAXIMUM_BUFFER_SIZE];
            DatagramPacket packet = new DatagramPacket(buff,buff.length);

            try {
                clientSocket.receive(packet);
                Packet p = Packet.valueOf(new String(buff));

                Thread t = timerThread(1);
                t.start();
                try{t.join();}catch(InterruptedException ie){}
                System.out.println("Data Recieved "+p.toString());


                if(state == State.SYN_SEND){
                    if(p.getAckNum() == SYNC_NUM +1){
                        System.out.println("Threeway handshake 2/3.");
```

Figure 5 3-way Hand Shaking Definition

```
        BUFFER.clear();
        if(p.isFinFlag()){
            System.out.println("Fourway handshake 1/4");

            state = State.FIN_RECV;
            ack = new Packet();
            ack.setFinFlag(true);
            ack.setAckFlag(true);
            send(ack.toString());
            System.out.println("Fourway handshake 2/4");
            System.out.println("Fourway handshake 3/4");
        }else{

            ack.setAckNum(INITIAL_SEGMENT+DATA.length());
            ack.setWindowSize(WINDOW_SIZE-BUFFER.size());
            send(ack.toString());
        }
    }
    SYNC_NUM = INITIAL_SEGMENT + DATA.length()-1;
}else if(state == State.FIN_RECV){
    if(p.isAckFlag() && p.getAckNum()==0){
        System.out.println("Fourway handshake 4/4");

        t = timerThread(5);
        t.start();
        try{t.join();}catch(InterruptedException ie){}
        clientSocket.close();
        break;
    }
}
```

Figure 6 4-way Handshaking Definition

- **Server.java** :Server Side code

ServerSocket

```
try {
    serverSocket.receive(packet);
    Packet p = Packet.valueOf(new String(buff));
    InetAddress clientAddress = packet.getAddress();
    int clientPort = packet.getPort();


    Thread t = timerThread(1);
    t.start();
    try{t.join();}catch(InterruptedException ie){}
    System.out.println("Data Recieved "+p.toString());

    if(state == State.NONE){

        if(p.isSyncFlag()){
            System.out.println("Threeway handshake 1/3.");
```

Figure 7 ServerSocket Definition

OUTPUT SCREENSHOTS

```
C:\Users\Asus>cd C:\Users\Asus\Desktop\TCP-UDP\TCP_Over_UDP

C:\Users\Asus\Desktop\TCP-UDP\TCP_Over_UDP>javac *.java

C:\Users\Asus\Desktop\TCP-UDP\TCP_Over_UDP>java Server 7777
Binding UDP server to port 7777...
Bind successful!
UDP listener for this server started!
Data Recieved SYNF=1;SYNN=1172;ACKF=0;ACKN=0;FINF=0;WINDOW=0;DATA=null


Threeway handshake 1/3.
Threeway handshake 2/3.
Data Recieved SYNF=1;SYNN=1173;ACKF=1;ACKN=4736;FINF=0;WINDOW=16;DATA=null


Threeway handshake 3/3.
```

Figure 8 Output of Server Side - 1

```
Data sent!
Fourway handshake 1/4
RCVD: SYNF=0;SYNN=0;ACKF=1;ACKN=4768;FINF=0;WINDOW=16;DATA=null


Data Recieved SYNF=0;SYNN=0;ACKF=1;ACKN=4769;FINF=0;WINDOW=15;DATA=null


Data Recieved SYNF=0;SYNN=0;ACKF=1;ACKN=4770;FINF=0;WINDOW=14;DATA=null


Data Recieved SYNF=0;SYNN=0;ACKF=1;ACKN=4771;FINF=0;WINDOW=13;DATA=null


Data Recieved SYNF=0;SYNN=0;ACKF=1;ACKN=4772;FINF=0;WINDOW=12;DATA=null


Data Recieved SYNF=0;SYNN=0;ACKF=1;ACKN=4773;FINF=0;WINDOW=11;DATA=null


Data Recieved SYNF=0;SYNN=0;ACKF=1;ACKN=0;FINF=1;WINDOW=0;DATA=null


Fourway handshake 2/4
Fourway handshake 3/4
Fourway handshake 4/4
```

Figure 9 Server Side Output - 2

```
C:\Users\Asus>cd C:\Users\Asus\Desktop\TCP-UDP\TCP_Over_UDP

C:\Users\Asus\Desktop\TCP-UDP\TCP_Over_UDP>java Client 4444 127.0.0.1 7777
Binding UDP server to port 4444...
Bind successful!
UDP listener for this client started!
Threeway handshake 1/3.
Data Recieved SYNF=1;SYNN=4735;ACKF=1;ACKN=1173;FINF=0;WINDOW=0;DATA=null



Threeway handshake 2/3.
Threeway handshake 3/3.
Data Recieved SYNF=0;SYNN=4736;ACKF=0;ACKN=0;FINF=0;WINDOW=0;DATA=A
```

Figure 10 Client Side Output - 1

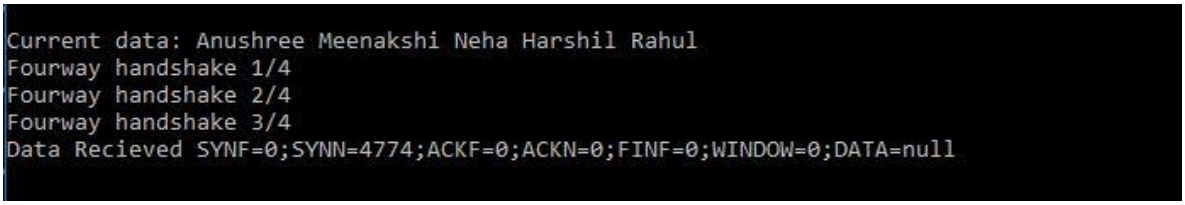

Figure 11 Client Side Final Output

CONCLUSION & FUTURE SCOPE

We have successfully implemented the project on "TCP over UDP".

We can further implement GUI (Graphical User Interfaces) in this project to make it more intuitive as the current project is basic.

REFERENCES

1. Data Communications and Networking By Behrouz A.Forouzan
2. Java Network Programming 4th Edition By Rusty Harold
3. StsckOverFlow : https://stackoverflow.com/questions/10680592/can-tcp-be-implemented-via-udp