DATA COMMUNICATION AND COMPUTER NETWORKS LAB

ASSIGNMENTS SET 1

Name: Debnil Sarkar

Roll: 002210503005

ASSIGNEMENT 1

PROBLEM STATEMENT:

Write a TCP Day-Time server program that returns the current time and date. Also write a TCP client program that sends request to the server to get the current time and date. Choose your own formats for the request/reply messages.

DESIGN OF REQUEST/REPLY PROTOCOL:

Server-Side:

1. The server listens on a designated port for incoming client connections.
2. Upon receiving a client connection, the server reads the incoming request continuously.
3. If the received request matches the expected "GetTime" request, the server generates the current time and date and sends a response with the "OK" status code and the current time and date in the desired format.
4. If the request is invalid or any error occurs during processing, the server sends a response with the "ERROR" status code and an error message.
5. The server closes the connection for client after getting “Exit” as message from the client.
6. Server keeps listening to the any other client request again.

Client-Side:

1. The client establishes a connection to the server using the server's IP address and port.
2. The client sends the "GetTime" request.
3. The client reads the response from the server.
4. If the status code is "OK", the client extracts and displays the current time and date provided in the response.
5. Client can send as many as request they want to the server as “GetTime”.
6. If the status code is "ERROR", the client displays the error message from the response.
7. The client can exit by typing “Exit” on the console.

SOURCE CODE:

Server-Side code:

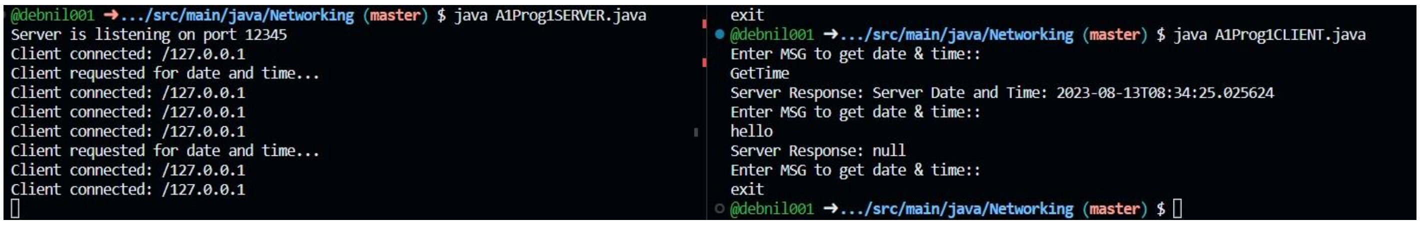
```
import java.io.*;
import java.net.*;

public class DayTimeServer {
    public static void main(String[] args) {
        int port = 12345; // Choose a suitable port number
        BufferedReader br=null;
        try (ServerSocket serverSocket = new ServerSocket(port)) {
            System.out.println("Server is listening on port " + port);

            while (true) {
                try (Socket clientSocket = serverSocket.accept()) {
                    System.out.println("Client connected: " +
clientSocket.getInetAddress());
                    br=new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
```

```
                    String msg=br.readLine();
                    if(msg==null)   continue;
                    if(msg.equals("GetTime")){
                        // Get current time and date
                        String dateTime =
java.time.LocalDateTime.now().toString();
                        System.out.println("Client requested for date and
time...");
                        // Send the response to the client
                        PrintWriter out = new
PrintWriter(clientSocket.getOutputStream(), true);
                        out.println("Server Date and Time: " + dateTime);
                    }
                    // Close the client socket
                    else if(msg.equals("exit")){
                        clientSocket.close();
                    }
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Client-Side code:

```
import java.io.*;
import java.net.*;
import java.util.*;


public class DayTimeClient {
    public static void main(String[] args) {
        String serverAddress = "localhost"; // Change to the server's IP address
if needed
        int serverPort = 12345; // Same port as the server
        while(true){
        try (Socket socket = new Socket(serverAddress, serverPort)) {
            // Create input and output streams
            System.out.println("Enter MSG to get date & time::");
            String msg=new Scanner(System.in).nextLine();
            if(msg.equals("exit"))  break;
            BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));

            // Send a request to the server
            PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
            out.println(msg);

            // Receive and print the server's response
            String response = in.readLine();
            System.out.println("Server Response: " + response);

        } catch (IOException e) {
            e.printStackTrace();
        }
        }
        }
    }
```

OUTPUT:



```
@debnil001 ➜.../src/main/java/Networking (master) $ java A1Prog1SERVER.java
Server is listening on port 12345
Client connected: /127.0.0.1
Client requested for date and time...
Client connected: /127.0.0.1
Client connected: /127.0.0.1
Client connected: /127.0.0.1
Client requested for date and time...
Client connected: /127.0.0.1
Client connected: /127.0.0.1
```

```
exit
@debnil001 ➜.../src/main/java/Networking (master) $ java A1Prog1CLIENT.java
Enter MSG to get date & time::
GetTime
Server Response: Server Date and Time: 2023-08-13T08:34:25.025624
Enter MSG to get date & time::
hello
Server Response: null
Enter MSG to get date & time::
exit
@debnil001 ➜.../src/main/java/Networking (master) $
```

<u>Output description:</u> Server is turned on at first and after getting a client joining request it accepts the request and waits for “GetTime” command to return back the current date time to client.

PROBLEM STATEMENT:

Write a TCP Math server program that accepts any valid integer arithmetic expression, evaluates it and returns the value of the expression. Also write a TCP client program that accepts an integer arithmetic expression from the user and sends it to the server to get the result of evaluation. Choose your own formatsfor the request/reply messages.

DESIGN OF REQUEST/REPLY PROTOCOL:

Server-Side:

1. The server listens on a designated port for incoming client connections.
2. Upon receiving a client connection, the server reads the incoming request.
3. The server attempts to evaluate the received mathematical expression using a user defined calculate function.
4. If the expression is valid and can be evaluated successfully, the server sends the result of the evaluation.
5. If the expression is invalid or evaluation fails, the server sends a response and an error message “Invalid Expression”.
6. The server keeps listening to the connected client or for any new request.

Client-Side:

1. The client establishes a connection to the server using the server's IP address and port.
2. The client reads a mathematical expression from the user until user gives “Exit” as message.
3. The client sends the expression to the server.
4. The client reads the response from the server until it encounters a “Exit”.
5. If the status code is "OK", the client extracts and displays the result provided in the response.
6. If invalid expression is sent by the client, then it displays the error message from the response.
7. The client closes the connection if “Exit” is typed on the console.

SOURCE CODE:

<u>Server-Side code:</u>

```
package Networking;
import java.io.*;
import java.net.*;
import java.util.*;
class Server {
    //calculate function for the expression evaluation
    public static int calculate(String s) {
        int len=s.length();
        int res=0;
        char sign='+';
        Stack<Integer> st=new Stack<>();
        int i;
        for(i=0;i<len;i++){
            char c=s.charAt(i);
            if(Character.isDigit(c)){
                int val=c-48;
                while(i+1<len && Character.isDigit(s.charAt(i+1))){
                    val=val*10+(s.charAt(i+1)-48);
                    i++;
                }
                if(sign=='+'){
                    st.push(val);
                }
                else if(sign=='-'){
                    st.push(-val);
                }
                else if(sign=='*'){
                    res=st.pop();
                    st.push(val*res);
                }
                else if(sign=='/'){
                    res=st.pop();
                    st.push(res/val);
                }
            }
            else if(c!=' '){//must be sign
             if(i>0 && s.charAt(i-1)!=' ' && !Character.isDigit(s.charAt(i-1))){
                    System.out.println("Invalid Expression!");
                    break;
                }
                sign=c;
            }
        }
        if(i!=len)  return Integer.MAX_VALUE;
        res=0;
        while(!st.isEmpty()){
            res+=st.pop();
        }
        return res;
    }
    public static void main(String[] args) throws IOException{
        ServerSocket ss=new ServerSocket(50000);
//server is initialized on port 50000 with default loop back address
        System.out.println("Server started...ready to accept client");
        PrintWriter out=null;//to send output to the client side
        BufferedReader in=null;//to get input from the client side
        while(true){
            Socket client=ss.accept();//ready to listen to the client
            System.out.println("Client connected"+client);//new client has
arrived
            out=new PrintWriter(client.getOutputStream(),true);//setting output
stream with the client
```

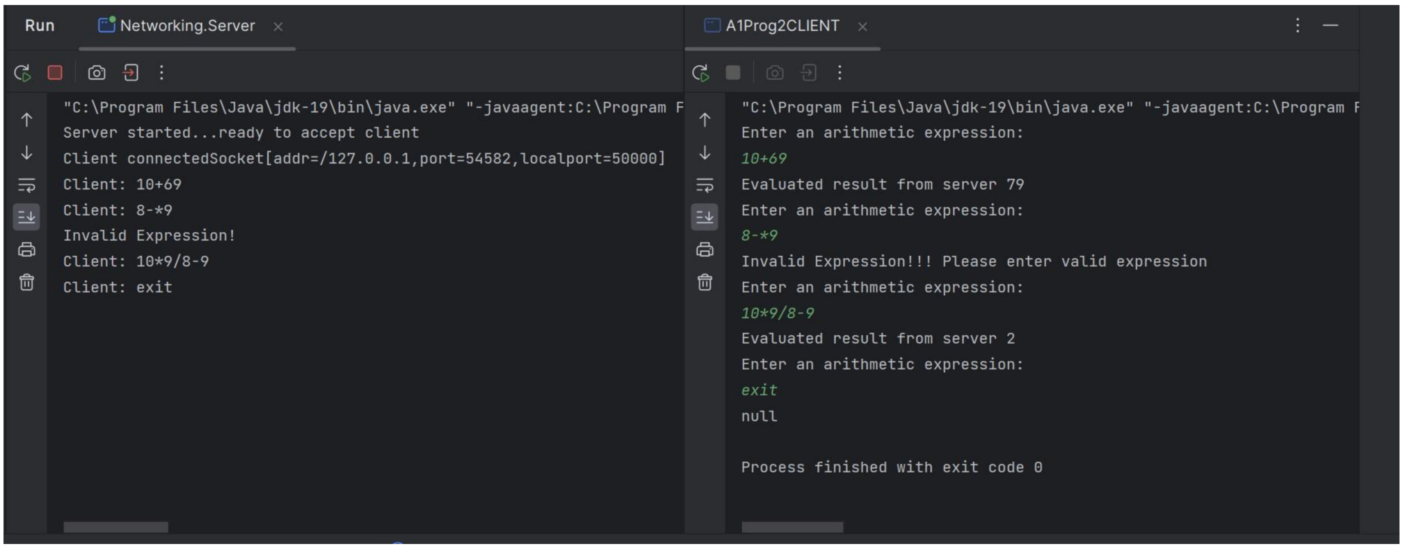
```
            in=new BufferedReader(new 
InputStreamReader(client.getInputStream()));//setting input stream with the 
client

            while(true){// as long as client sends expression, server will 
evaluate until exit is sent
                String exp=in.readLine();//reading the expression from the 
client side
                System.out.println("Client: "+exp);//printing the current 
expression on the console
                if(exp.equals("exit")) break;//if exit is sent
                int res=Server.calculate(exp);//evaluating the result for the 
given expression
                if(res==Integer.MAX_VALUE)
                    out.println("Invalid Expression!!! Please enter valid 
expression");
                else
                    out.println("Evaluated result from server "+res);//sending 
the result back to the client
            }
            //freeing up all the resources
            client.close();
            out.close();
            in.close();
        }
    }
}
```

<u>Client-Side code:</u>

```
package Networking;

import java.io.*;
import java.net.Socket;
import java.util.Scanner;

public class A1Prog2CLIENT {
    public static void main(String[] args) throws IOException {
        Socket sc=new Socket("localhost",50000);//creating connection to the 
server's port and address
        PrintWriter out=null;
        BufferedReader in=null;
        while(true){
            out=new PrintWriter(sc.getOutputStream(),true);// output stream 
object for getting output from server
            in=new BufferedReader(new InputStreamReader(sc.getInputStream()));//
input stream object for getting input from server
            Scanner console=new Scanner(System.in);
            System.out.println("Enter an arithmetic expression:");
            String exp=console.next();//reading expression from console of the 
client
            out.println(exp);//sending to the server in form of output stream

            System.out.println(in.readLine());//getting the result and printing 
on the screen

            if(exp.equals("exit"))  break;//if the console message is exit then 
exit from the program
        }
    }
```

}

OUTPUT:



```
"C:\Program Files\Java\jdk-19\bin\java.exe" "-javaagent:C:\Program F
Server started...ready to accept client
Client connectedSocket[addr=/127.0.0.1,port=54582,localport=50000]
Client: 10+69
Client: 8-*9
Invalid Expression!
Client: 10*9/8-9
Client: exit
```

```
"C:\Program Files\Java\jdk-19\bin\java.exe" "-javaagent:C:\Program F
Enter an arithmetic expression:
10+69
Evaluated result from server 79
Enter an arithmetic expression:
8-*9
Invalid Expression!!! Please enter valid expression
Enter an arithmetic expression:
10*9/8-9
Evaluated result from server 2
Enter an arithmetic expression:
exit
null

Process finished with exit code 0
```

Output Description: Server is turned on and waits for client to join. After getting a client server waits for expression from client to evaluate. If client sends valid expression to the server, then it returns back result otherwise it sends error message as "Invalid Expression".

PROBLEM STATEMENT:

Implement a UDP server program that returns the permanent address of a student upon receiving a request from a client. Assume that, a text file that stores the names of students and their permanent addresses is available local to the server. Choose your own formats for the request/reply messages.

DESIGN OF REQUEST/REPLY PROTOCOL:

Server-Side:

1. The server listens on a designated port for incoming client connections.
2. Upon receiving a client connection, the server reads the incoming request.
3. The server attempts to search the provided name on the local storage.
4. If the provided name is not found then server sends "Student not found" to the server.
5. If name is present in the local file, it sends the corresponding address of the given student's name by searching into the file.
6. The server keeps listening for new clients.

Client-Side:

1. The client establishes a connection to the server using the server's IP address and port.
2. The client reads a student name from console.
3. The client sends the name to the server.
4. The client gets only one chance to get the address.
5. If the given student is present in the server, client will get the address of the specified student.
6. If the name is not present in the server it will get a message from server as "Student not found"
7. In either case client exits from the program.

SOURCE CODE:

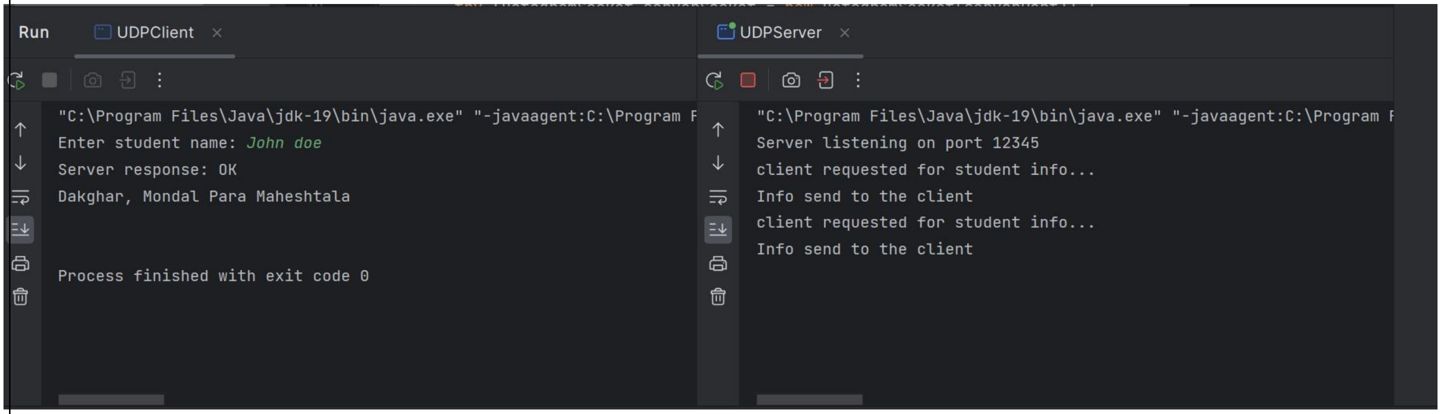
Server-Side code:

```
package Networking;
import java.io.*;
import java.net.*;

class UDPServer {
    public static void main(String[] args) {
        final int serverPort = 12345;
        final String studentInfoFile = 
"C:\\Users\\HP\\Documents\\NetBeansProjects\\JavaSocket\\src\\main\\java\\Studen
t.txt";
        //student file location in local machine
        try (DatagramSocket serverSocket = new DatagramSocket(serverPort)) {
            System.out.println("Server listening on port " + serverPort);//start 
listening the client req if any

            byte[] receiveData = new byte[1024];

            while (true) {
                DatagramPacket receivePacket = new DatagramPacket(receiveData, 
receiveData.length);
                serverSocket.receive(receivePacket);//data received from client
                System.out.println("client requested for student info...");
                String studentName = new String(receivePacket.getData(), 0, 
receivePacket.getLength());//converting the student name from bytes to string

                String response = findStudentAddress(studentName, 
studentInfoFile);//function call to search in file

                InetAddress clientAddress = receivePacket.getAddress();
                int clientPort = receivePacket.getPort();//get the clients port 
to send the data

                byte[] sendData = response.getBytes();
                DatagramPacket sendPacket = new DatagramPacket(sendData, 
sendData.length, clientAddress, clientPort);
                serverSocket.send(sendPacket);//packet sent to the client
                System.out.println("Info send to the client");
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    //function to search in file
    private static String findStudentAddress(String studentName, String 
studentInfoFile) {
        try (BufferedReader br = new BufferedReader(new 
FileReader(studentInfoFile))) {
            String line;
            while ((line = br.readLine()) != null) {
                String[] parts = line.split(":");
                if (parts.length == 2 && 
parts[0].trim().equalsIgnoreCase(studentName)) {
                    return "OK\n" + parts[1].trim() + "\n";
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }

        return "ERROR\nStudent not found.\n";
    }
}
```

<u>Client-Side code:</u>

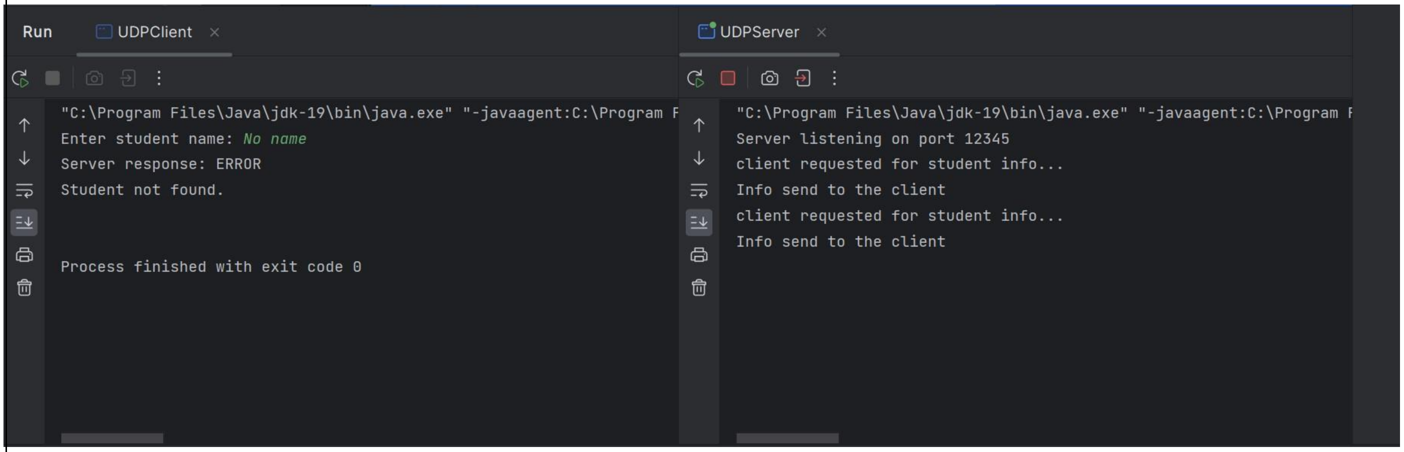
```
package Networking;
import java.io.*;
import java.net.*;

class UDPClient {
    public static void main(String[] args) {
        final String serverAddress = "127.0.0.1"; // server's IP address
        final int serverPort = 12345; //server port

        try (DatagramSocket clientSocket = new DatagramSocket()) {//creating UDP
connection
            BufferedReader userInput = new BufferedReader(new
InputStreamReader(System.in));

            System.out.print("Enter student name: ");
            String studentName = userInput.readLine();//getting the student name
to search in server

            byte[] sendData = studentName.getBytes();//converting the data into
bytes
            InetAddress serverIPAddress = InetAddress.getByName(serverAddress);
            DatagramPacket sendPacket = new DatagramPacket(sendData,
sendData.length, serverIPAddress, serverPort);
            clientSocket.send(sendPacket);//sending the data in form of packet
through UDP connection

            byte[] receiveData = new byte[1024];
            DatagramPacket receivePacket = new DatagramPacket(receiveData,
receiveData.length);
            clientSocket.receive(receivePacket);//response received from client

            String response = new String(receivePacket.getData(), 0,
receivePacket.getLength());
            System.out.println("Server response: " + response);//printing the
response
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

OUTPUT:



```
"C:\Program Files\Java\jdk-19\bin\java.exe" "-javaagent:C:\Program F
Enter student name: John doe
Server response: OK
Dakghar, Mondal Para Maheshtala


Process finished with exit code 0
```

```
"C:\Program Files\Java\jdk-19\bin\java.exe" "-javaagent:C:\Program F
Server listening on port 12345
client requested for student info...
Info send to the client
client requested for student info...
Info send to the client
```



```
"C:\Program Files\Java\jdk-19\bin\java.exe" "-javaagent:C:\Program F
Enter student name: No name
Server response: ERROR
Student not found.


Process finished with exit code 0
```

```
"C:\Program Files\Java\jdk-19\bin\java.exe" "-javaagent:C:\Program F
Server listening on port 12345
client requested for student info...
Info send to the client
client requested for student info...
Info send to the client
```

<u>Output description:</u> As the connection is established using UDP, Server does not acknowledge the client. If client sends a name which is present in the server's local storage it will return back student's corresponding address to the client otherwise it sends "No record found".

ASSIGNMENT 2

The objective of this laboratory exercise is to look at the details of the Transmission Control Protocol (TCP). TCP is a transport layer protocol. It is used by many application protocols like HTTP, FTP, SSH etc., where guaranteed and reliable delivery of messages is required.
To do this exercise you need to install the Wireshark tool. This tool would be used to capture and examine a packet trace. Wireshark can be downloaded from www.wireshark.org.

Step1: Capture a Trace

(i) Launch Wireshark
(ii) From Capture→Options select Loopback interface
(iii) Start a capture with a filter of "ip.addr==127.0.0.1 and tcp.port==xxxx", where xxxx is the port number used by the TCP server.
(iv) Run the TCP server program on a terminal.
(v) Run two instances of the TCP client program on two separate terminals and send some dummy data to the sever.
(vi) Stop Wireshark capture

Step2: TCP Connection Establishment

To observe the three-way handshake in action, look for a TCP segment with SYN flag set. A" SYN" segment is the start of the three-way handshake and is sent by the TCP client to the TCP server. The server then replies with a TCP segment with SYN and ACK flag set. And finally, the client sends an" ACK" to the server.
For all the above three segments record the values of the sequence number and acknowledgment fields. Draw a time sequence diagram of the three-way handshake for TCP connection establishment in your trace. Do it for all the client connections.

Step3: TCP Data Transfer

For all data segments sent by the client, record the value of the sequence number and acknowledge number fields. Also, record the same for the corresponding acknowledgements sent by the server. Draw a time sequence diagram of the data transfer in your trace. Do it for all the client connections.

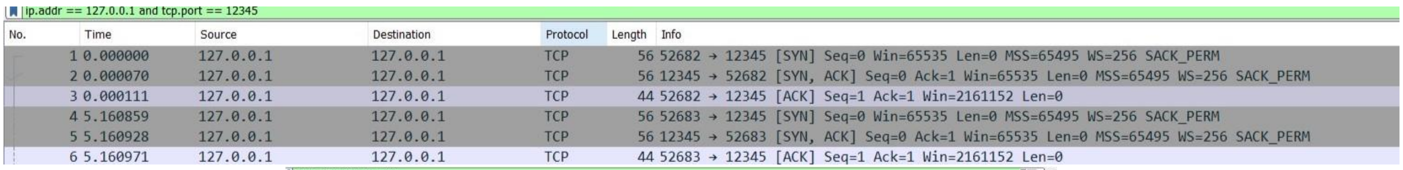
Step4: TCP Connection Termination

Once the data transfer is over, the client initiates the connection termination by sending TCP segment with FIN flag set, to the server. Server acknowledges it and sends its own intention to terminate the connection by sending a TCP segment with FIN and ACK flags set. The client finally sends an ACK segment to the server.
For all the above three segments record the values of the sequence number and acknowledgment fields. Draw a time sequence diagram of the three-way handshake for TCP connection termination in your trace. Do it for all the client connections.

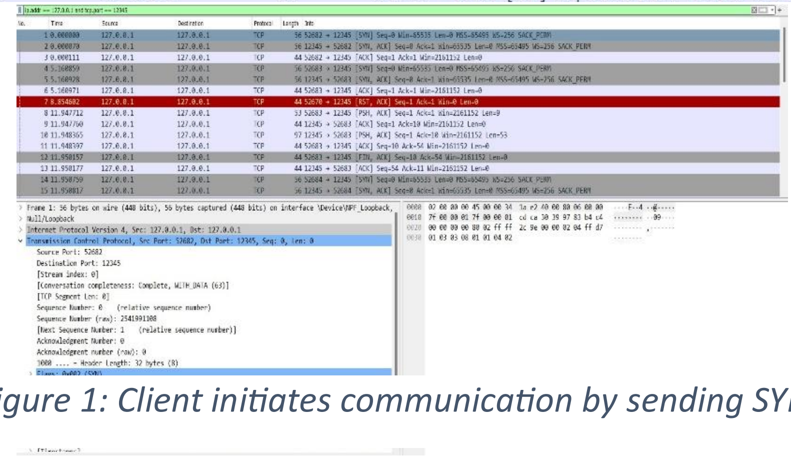
Step 1:

Connection establishment:

For client1 and client2 (SYN – SYN+ACK - ACK)

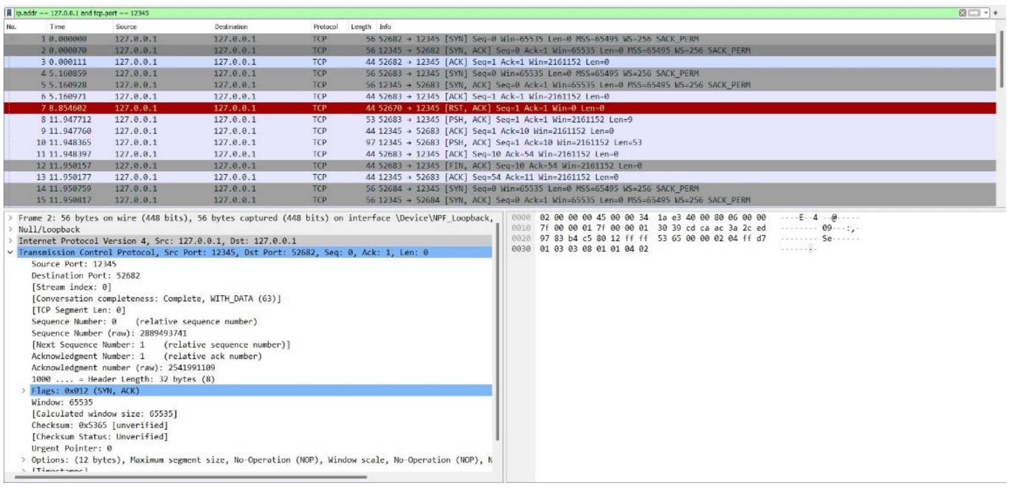


```
ip.addr == 127.0.0.1 and tcp.port == 12345
No.  Time      Source     Destination  Protocol  Length  Info
  1 0.000000  127.0.0.1  127.0.0.1    TCP       56 52682 → 12345 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
  2 0.000070  127.0.0.1  127.0.0.1    TCP       56 12345 → 52682 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
  3 0.000111  127.0.0.1  127.0.0.1    TCP       44 52682 → 12345 [ACK] Seq=1 Ack=1 Win=2161152 Len=0
  4 5.160859  127.0.0.1  127.0.0.1    TCP       56 52683 → 12345 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
  5 5.160928  127.0.0.1  127.0.0.1    TCP       56 12345 → 52683 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
  6 5.160971  127.0.0.1  127.0.0.1    TCP       44 52683 → 12345 [ACK] Seq=1 Ack=1 Win=2161152 Len=0
```



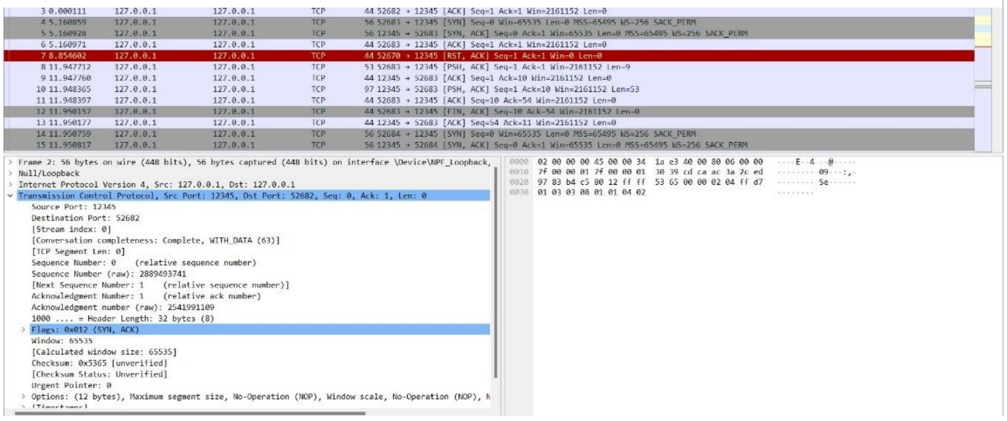
```
ip.addr == 127.0.0.1 and tcp.port == 12345
No.  Time       Source     Destination  Protocol  Length  Info
  1 0.000000   127.0.0.1  127.0.0.1    TCP       56 52682 → 12345 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
  2 0.000070   127.0.0.1  127.0.0.1    TCP       56 12345 → 52682 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
  3 0.000111   127.0.0.1  127.0.0.1    TCP       44 52682 → 12345 [ACK] Seq=1 Ack=1 Win=2161152 Len=0
  4 5.160859   127.0.0.1  127.0.0.1    TCP       56 52683 → 12345 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
  5 5.160928   127.0.0.1  127.0.0.1    TCP       56 12345 → 52683 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
  6 5.160971   127.0.0.1  127.0.0.1    TCP       44 52683 → 12345 [ACK] Seq=1 Ack=1 Win=2161152 Len=0
  7 8.854682   127.0.0.1  127.0.0.1    TCP       44 52670 → 12345 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
  8 11.947712  127.0.0.1  127.0.0.1    TCP       53 52683 → 12345 [PSH, ACK] Seq=1 Ack=1 Win=2161152 Len=9
  9 11.947760  127.0.0.1  127.0.0.1    TCP       44 12345 → 52683 [ACK] Seq=1 Ack=10 Win=2161152 Len=0
 10 11.948365  127.0.0.1  127.0.0.1    TCP       97 12345 → 52683 [PSH, ACK] Seq=1 Ack=10 Win=2161152 Len=53
 11 11.948397  127.0.0.1  127.0.0.1    TCP       44 52683 → 12345 [ACK] Seq=10 Ack=54 Win=2161152 Len=0
 12 11.950157  127.0.0.1  127.0.0.1    TCP       44 52683 → 12345 [FIN, ACK] Seq=10 Ack=54 Win=2161152 Len=0
 13 11.950177  127.0.0.1  127.0.0.1    TCP       44 12345 → 52683 [ACK] Seq=54 Ack=11 Win=2161152 Len=0
 14 11.950759  127.0.0.1  127.0.0.1    TCP       56 52684 → 12345 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
 15 11.950817  127.0.0.1  127.0.0.1    TCP       56 12345 → 52684 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM

> Frame 1: 56 bytes on wire (448 bits), 56 bytes captured (448 bits) on interface \Device\NPF_Loopback,
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
v Transmission Control Protocol, Src Port: 52682, Dst Port: 12345, Seq: 0, Len: 0
    Source Port: 52682
    Destination Port: 12345
    [Stream index: 0]
    [Conversation completeness: Complete, WITH_DATA (63)]
    [TCP Segment Len: 0]
    Sequence Number: 0    (relative sequence number)
    Sequence Number (raw): 2541991108
    [Next Sequence Number: 1    (relative sequence number)]
    Acknowledgment Number: 0
    Acknowledgment number (raw): 0
    1000 .... = Header Length: 32 bytes (8)
  > Flags: 0x002 (SYN)
  > [Timestamps]

0000  02 00 00 00 45 00 00 34  1a e2 40 00 80 06 00 00   ....E..4 ..@.....
0010  7f 00 00 01 7f 00 00 01  cd ca 30 39 97 83 b4 c4   ........ ..09....
0020  00 00 00 00 80 02 ff ff  2c 9e 00 00 02 04 ff d7   ........ ,.......
0030  01 03 03 08 01 01 04 02                            ........
```

Figure 1: Client initiates communication by sending SYN flag



```
ip.addr == 127.0.0.1 and tcp.port == 12345
No.   Time        Source      Destination  Protocol  Length  Info
    1 0.000000    127.0.0.1   127.0.0.1    TCP       56 52682 → 12345 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
    2 0.000070    127.0.0.1   127.0.0.1    TCP       56 12345 → 52682 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
    3 0.000111    127.0.0.1   127.0.0.1    TCP       44 52682 → 12345 [ACK] Seq=1 Ack=1 Win=2161152 Len=0
    4 5.160859    127.0.0.1   127.0.0.1    TCP       56 52683 → 12345 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
    5 5.160928    127.0.0.1   127.0.0.1    TCP       56 12345 → 52683 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
    6 5.160971    127.0.0.1   127.0.0.1    TCP       44 52683 → 12345 [ACK] Seq=1 Ack=1 Win=2161152 Len=0
    7 8.854602    127.0.0.1   127.0.0.1    TCP       44 52670 → 12345 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
    8 11.947712   127.0.0.1   127.0.0.1    TCP       53 52683 → 12345 [PSH, ACK] Seq=1 Ack=1 Win=2161152 Len=9
    9 11.947760   127.0.0.1   127.0.0.1    TCP       44 12345 → 52683 [ACK] Seq=1 Ack=10 Win=2161152 Len=0
   10 11.948365   127.0.0.1   127.0.0.1    TCP       97 12345 → 52683 [PSH, ACK] Seq=1 Ack=10 Win=2161152 Len=53
   11 11.948397   127.0.0.1   127.0.0.1    TCP       44 52683 → 12345 [ACK] Seq=10 Ack=54 Win=2161152 Len=0
   12 11.950157   127.0.0.1   127.0.0.1    TCP       44 52683 → 12345 [FIN, ACK] Seq=10 Ack=54 Win=2161152 Len=0
   13 11.950177   127.0.0.1   127.0.0.1    TCP       44 12345 → 52683 [ACK] Seq=54 Ack=11 Win=2161152 Len=0
   14 11.950759   127.0.0.1   127.0.0.1    TCP       56 52684 → 12345 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
   15 11.950817   127.0.0.1   127.0.0.1    TCP       56 12345 → 52684 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM

> Frame 2: 56 bytes on wire (448 bits), 56 bytes captured (448 bits) on interface \Device\NPF_Loopback,
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
v Transmission Control Protocol, Src Port: 12345, Dst Port: 52682, Seq: 0, Ack: 1, Len: 0
    Source Port: 12345
    Destination Port: 52682
    [Stream index: 0]
    [Conversation completeness: Complete, WITH_DATA (63)]
    [TCP Segment Len: 0]
    Sequence Number: 0    (relative sequence number)
    Sequence Number (raw): 2889493741
    [Next Sequence Number: 1    (relative sequence number)]
    Acknowledgment Number: 1    (relative ack number)
    Acknowledgment number (raw): 2541991109
    1000 .... = Header Length: 32 bytes (8)
  > Flags: 0x012 (SYN, ACK)
    Window: 65535
    [Calculated window size: 65535]
    Checksum: 0x5365 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  > Options: (12 bytes), Maximum segment size, No-Operation (NOP), Window scale, No-Operation (NOP), N

0000  02 00 00 00 45 00 00 34  1a e3 40 00 80 06 00 00   ····E··4 ··@·····
0010  7f 00 00 01 7f 00 00 01  30 39 cd ca ac 3a 2c ed   ········ 09···:,·
0020  97 83 b4 c5 80 12 ff ff  53 65 00 00 02 04 ff d7   ········ Se······
0030  01 03 03 08 01 01 04 02                            ········
```

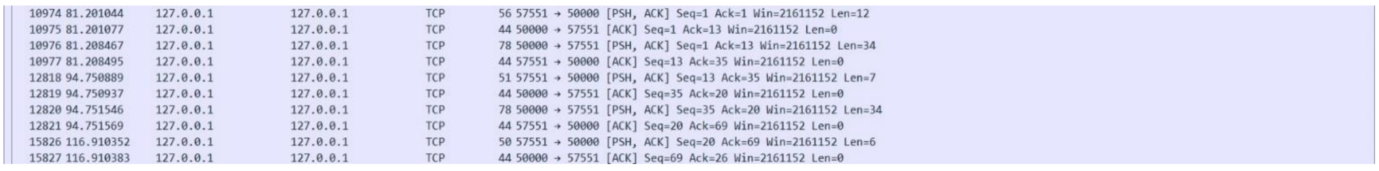
Figure 2: Server response back with SYN+ACK flag



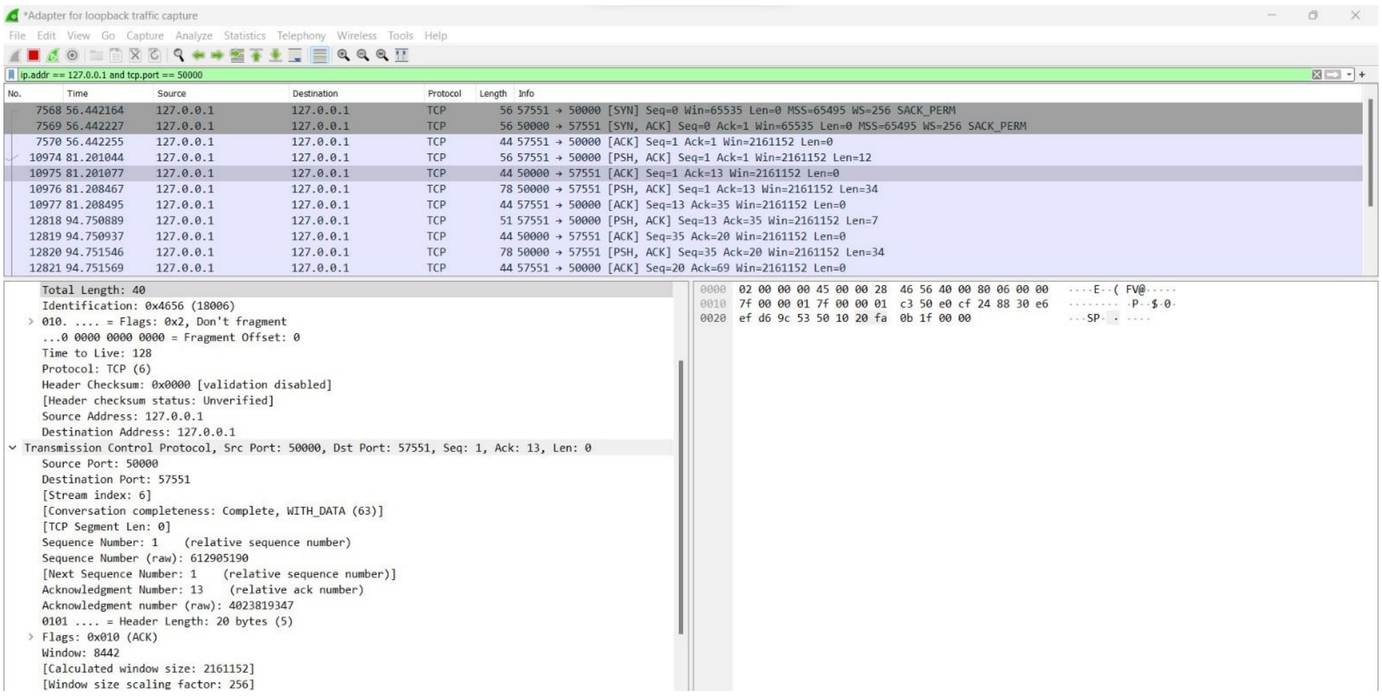
```
    3 0.000111    127.0.0.1   127.0.0.1    TCP       44 52682 → 12345 [ACK] Seq=1 Ack=1 Win=2161152 Len=0
    4 5.160859    127.0.0.1   127.0.0.1    TCP       56 52683 → 12345 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
    5 5.160928    127.0.0.1   127.0.0.1    TCP       56 12345 → 52683 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
    6 5.160971    127.0.0.1   127.0.0.1    TCP       44 52683 → 12345 [ACK] Seq=1 Ack=1 Win=2161152 Len=0
    7 8.854602    127.0.0.1   127.0.0.1    TCP       44 52670 → 12345 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
    8 11.947712   127.0.0.1   127.0.0.1    TCP       53 52683 → 12345 [PSH, ACK] Seq=1 Ack=1 Win=2161152 Len=9
    9 11.947760   127.0.0.1   127.0.0.1    TCP       44 12345 → 52683 [ACK] Seq=1 Ack=10 Win=2161152 Len=0
   10 11.948365   127.0.0.1   127.0.0.1    TCP       97 12345 → 52683 [PSH, ACK] Seq=1 Ack=10 Win=2161152 Len=53
   11 11.948397   127.0.0.1   127.0.0.1    TCP       44 52683 → 12345 [ACK] Seq=10 Ack=54 Win=2161152 Len=0
   12 11.950157   127.0.0.1   127.0.0.1    TCP       44 52683 → 12345 [FIN, ACK] Seq=10 Ack=54 Win=2161152 Len=0
   13 11.950177   127.0.0.1   127.0.0.1    TCP       44 12345 → 52683 [ACK] Seq=54 Ack=11 Win=2161152 Len=0
   14 11.950759   127.0.0.1   127.0.0.1    TCP       56 52684 → 12345 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
   15 11.950817   127.0.0.1   127.0.0.1    TCP       56 12345 → 52684 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM

> Frame 2: 56 bytes on wire (448 bits), 56 bytes captured (448 bits) on interface \Device\NPF_Loopback,
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
v Transmission Control Protocol, Src Port: 12345, Dst Port: 52682, Seq: 0, Ack: 1, Len: 0
    Source Port: 12345
    Destination Port: 52682
    [Stream index: 0]
    [Conversation completeness: Complete, WITH_DATA (63)]
    [TCP Segment Len: 0]
    Sequence Number: 0    (relative sequence number)
    Sequence Number (raw): 2889493741
    [Next Sequence Number: 1    (relative sequence number)]
    Acknowledgment Number: 1    (relative ack number)
    Acknowledgment number (raw): 2541991109
    1000 .... = Header Length: 32 bytes (8)
  > Flags: 0x012 (SYN, ACK)
    Window: 65535
    [Calculated window size: 65535]
    Checksum: 0x5365 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  > Options: (12 bytes), Maximum segment size, No-Operation (NOP), Window scale, No-Operation (NOP), N

0000  02 00 00 00 45 00 00 34  1a e3 40 00 80 06 00 00   ····E··4 ··@·····
0010  7f 00 00 01 7f 00 00 01  30 39 cd ca ac 3a 2c ed   ········ 09···:,·
0020  97 83 b4 c5 80 12 ff ff  53 65 00 00 02 04 ff d7   ········ Se······
0030  01 03 03 08 01 01 04 02                            ········
```

Figure 3:Client sends ACK to Server

Data Transfer(PSH+ACK - ACK)

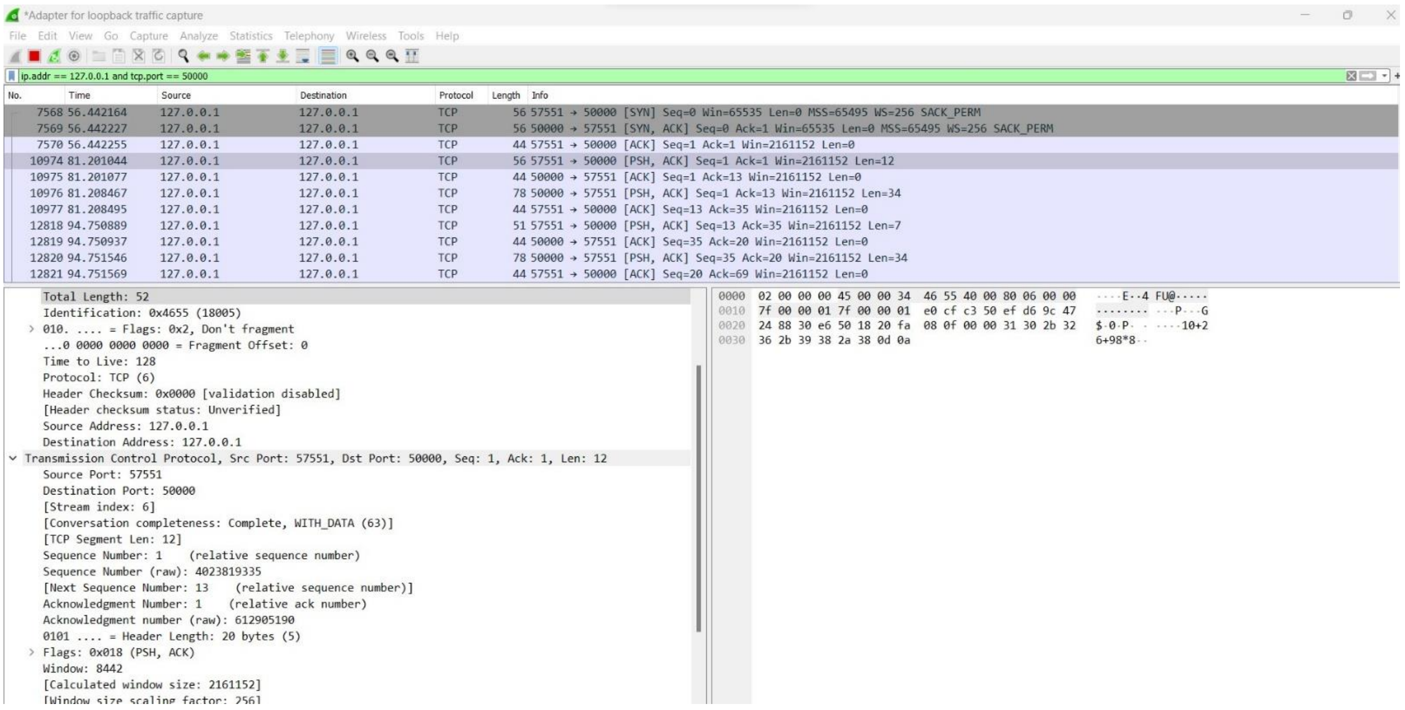


```
10974 81.201044    127.0.0.1   127.0.0.1   TCP   56 57551 → 50000 [PSH, ACK] Seq=1 Ack=1 Win=2161152 Len=12
10975 81.201077    127.0.0.1   127.0.0.1   TCP   44 50000 → 57551 [ACK] Seq=1 Ack=13 Win=2161152 Len=0
10976 81.208467    127.0.0.1   127.0.0.1   TCP   78 50000 → 57551 [PSH, ACK] Seq=1 Ack=13 Win=2161152 Len=34
10977 81.208495    127.0.0.1   127.0.0.1   TCP   44 57551 → 50000 [ACK] Seq=13 Ack=35 Win=2161152 Len=0
12818 94.750889    127.0.0.1   127.0.0.1   TCP   51 57551 → 50000 [PSH, ACK] Seq=13 Ack=35 Win=2161152 Len=7
12819 94.750937    127.0.0.1   127.0.0.1   TCP   44 50000 → 57551 [ACK] Seq=35 Ack=20 Win=2161152 Len=0
12820 94.751546    127.0.0.1   127.0.0.1   TCP   78 50000 → 57551 [PSH, ACK] Seq=35 Ack=20 Win=2161152 Len=34
12821 94.751569    127.0.0.1   127.0.0.1   TCP   44 57551 → 50000 [ACK] Seq=20 Ack=69 Win=2161152 Len=0
15826 116.910352   127.0.0.1   127.0.0.1   TCP   50 57551 → 50000 [PSH, ACK] Seq=20 Ack=69 Win=2161152 Len=6
15827 116.910383   127.0.0.1   127.0.0.1   TCP   44 50000 → 57551 [ACK] Seq=69 Ack=26 Win=2161152 Len=0
```



*Adapter for loopback traffic capture
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

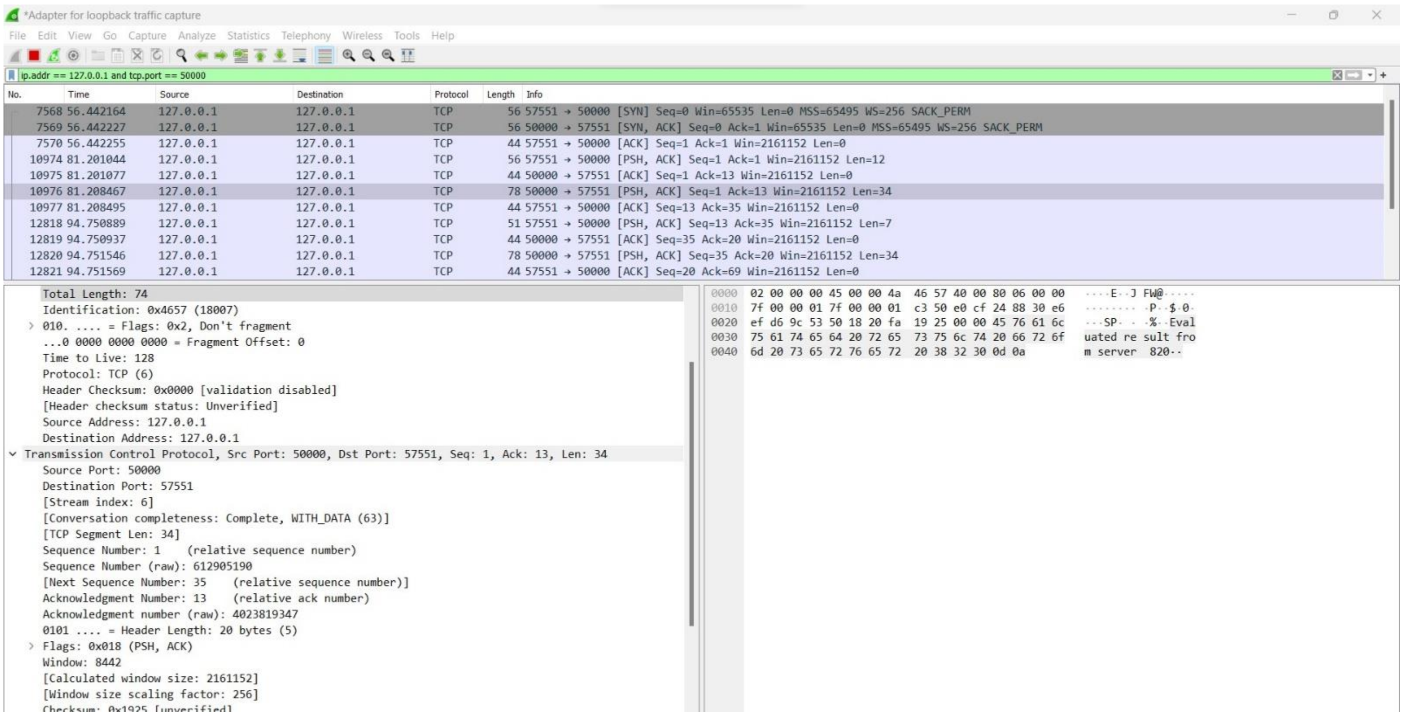
```
ip.addr == 127.0.0.1 and tcp.port == 50000
No.    Time         Source      Destination  Protocol  Length  Info
  7568 56.442164    127.0.0.1   127.0.0.1    TCP       56 57551 → 50000 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
  7569 56.442227    127.0.0.1   127.0.0.1    TCP       56 50000 → 57551 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
  7570 56.442255    127.0.0.1   127.0.0.1    TCP       44 57551 → 50000 [ACK] Seq=1 Ack=1 Win=2161152 Len=0
 10974 81.201044    127.0.0.1   127.0.0.1    TCP       56 57551 → 50000 [PSH, ACK] Seq=1 Ack=1 Win=2161152 Len=12
 10975 81.201077    127.0.0.1   127.0.0.1    TCP       44 50000 → 57551 [ACK] Seq=1 Ack=13 Win=2161152 Len=0
 10976 81.208467    127.0.0.1   127.0.0.1    TCP       78 50000 → 57551 [PSH, ACK] Seq=1 Ack=13 Win=2161152 Len=34
 10977 81.208495    127.0.0.1   127.0.0.1    TCP       44 57551 → 50000 [ACK] Seq=13 Ack=35 Win=2161152 Len=0
 12818 94.750889    127.0.0.1   127.0.0.1    TCP       51 57551 → 50000 [PSH, ACK] Seq=13 Ack=35 Win=2161152 Len=7
 12819 94.750937    127.0.0.1   127.0.0.1    TCP       44 50000 → 57551 [ACK] Seq=35 Ack=20 Win=2161152 Len=0
 12820 94.751546    127.0.0.1   127.0.0.1    TCP       78 50000 → 57551 [PSH, ACK] Seq=35 Ack=20 Win=2161152 Len=34
 12821 94.751569    127.0.0.1   127.0.0.1    TCP       44 57551 → 50000 [ACK] Seq=20 Ack=69 Win=2161152 Len=0

    Total Length: 40
    Identification: 0x4656 (18006)
  > 010. .... = Flags: 0x2, Don't fragment
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 128
    Protocol: TCP (6)
    Header Checksum: 0x0000 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 127.0.0.1
    Destination Address: 127.0.0.1
v Transmission Control Protocol, Src Port: 50000, Dst Port: 57551, Seq: 1, Ack: 13, Len: 0
    Source Port: 50000
    Destination Port: 57551
    [Stream index: 6]
    [Conversation completeness: Complete, WITH_DATA (63)]
    [TCP Segment Len: 0]
    Sequence Number: 1    (relative sequence number)
    Sequence Number (raw): 612905190
    [Next Sequence Number: 1    (relative sequence number)]
    Acknowledgment Number: 13    (relative ack number)
    Acknowledgment number (raw): 4023819347
    0101 .... = Header Length: 20 bytes (5)
  > Flags: 0x010 (ACK)
    Window: 8442
    [Calculated window size: 2161152]
    [Window size scaling factor: 256]

0000  02 00 00 00 45 00 00 28  46 56 40 00 80 06 00 00   ····E··( FV@·····
0010  7f 00 00 01 7f 00 00 01  c3 50 e0 cf 24 88 30 e6   ········ ·P··$·0·
0020  ef d6 9c 53 50 10 20 fa  0b 1f 00 00               ···SP· · ····
```


*Adapter for loopback traffic capture
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

```
ip.addr == 127.0.0.1 and tcp.port == 50000

No.   Time       Source      Destination  Protocol  Length  Info
 7568 56.442164  127.0.0.1   127.0.0.1    TCP       56 57551 → 50000 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
 7569 56.442227  127.0.0.1   127.0.0.1    TCP       56 50000 → 57551 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
 7570 56.442255  127.0.0.1   127.0.0.1    TCP       44 57551 → 50000 [ACK] Seq=1 Ack=1 Win=2161152 Len=0
10974 81.201044  127.0.0.1   127.0.0.1    TCP       56 57551 → 50000 [PSH, ACK] Seq=1 Ack=1 Win=2161152 Len=12
10975 81.201077  127.0.0.1   127.0.0.1    TCP       44 50000 → 57551 [ACK] Seq=1 Ack=13 Win=2161152 Len=0
10976 81.208467  127.0.0.1   127.0.0.1    TCP       78 50000 → 57551 [PSH, ACK] Seq=1 Ack=13 Win=2161152 Len=34
10977 81.208495  127.0.0.1   127.0.0.1    TCP       44 57551 → 50000 [ACK] Seq=13 Ack=35 Win=2161152 Len=0
12818 94.750889  127.0.0.1   127.0.0.1    TCP       51 57551 → 50000 [PSH, ACK] Seq=13 Ack=35 Win=2161152 Len=7
12819 94.750937  127.0.0.1   127.0.0.1    TCP       44 50000 → 57551 [ACK] Seq=35 Ack=20 Win=2161152 Len=0
12820 94.751546  127.0.0.1   127.0.0.1    TCP       78 50000 → 57551 [PSH, ACK] Seq=35 Ack=20 Win=2161152 Len=34
12821 94.751569  127.0.0.1   127.0.0.1    TCP       44 57551 → 50000 [ACK] Seq=20 Ack=69 Win=2161152 Len=0
```

```
    Total Length: 52
    Identification: 0x4655 (18005)
  > 010. .... = Flags: 0x2, Don't fragment
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 128
    Protocol: TCP (6)
    Header Checksum: 0x0000 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 127.0.0.1
    Destination Address: 127.0.0.1
v Transmission Control Protocol, Src Port: 57551, Dst Port: 50000, Seq: 1, Ack: 1, Len: 12
    Source Port: 57551
    Destination Port: 50000
    [Stream index: 6]
    [Conversation completeness: Complete, WITH_DATA (63)]
    [TCP Segment Len: 12]
    Sequence Number: 1    (relative sequence number)
    Sequence Number (raw): 4023819335
    [Next Sequence Number: 13    (relative sequence number)]
    Acknowledgment Number: 1    (relative ack number)
    Acknowledgment number (raw): 612905190
    0101 .... = Header Length: 20 bytes (5)
  > Flags: 0x018 (PSH, ACK)
    Window: 8442
    [Calculated window size: 2161152]
    [Window size scaling factor: 256]
```

```
0000  02 00 00 00 45 00 00 34  46 55 40 00 80 06 00 00   ····E··4 FU@·····
0010  7f 00 00 01 7f 00 00 01  e0 cf c3 50 ef d6 9c 47   ········ ···P···G
0020  24 88 30 e6 50 18 20 fa  08 0f 00 00 31 30 2b 32   $·0·P· · ····10+2
0030  36 2b 39 38 2a 38 0d 0a                            6+98*8··
```


*Adapter for loopback traffic capture
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

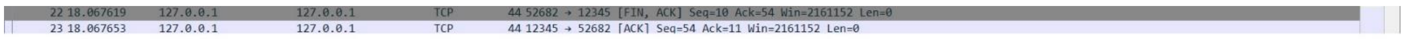
```
ip.addr == 127.0.0.1 and tcp.port == 50000

No.   Time       Source      Destination  Protocol  Length  Info
 7568 56.442164  127.0.0.1   127.0.0.1    TCP       56 57551 → 50000 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
 7569 56.442227  127.0.0.1   127.0.0.1    TCP       56 50000 → 57551 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
 7570 56.442255  127.0.0.1   127.0.0.1    TCP       44 57551 → 50000 [ACK] Seq=1 Ack=1 Win=2161152 Len=0
10974 81.201044  127.0.0.1   127.0.0.1    TCP       56 57551 → 50000 [PSH, ACK] Seq=1 Ack=1 Win=2161152 Len=12
10975 81.201077  127.0.0.1   127.0.0.1    TCP       44 50000 → 57551 [ACK] Seq=1 Ack=13 Win=2161152 Len=0
10976 81.208467  127.0.0.1   127.0.0.1    TCP       78 50000 → 57551 [PSH, ACK] Seq=1 Ack=13 Win=2161152 Len=34
10977 81.208495  127.0.0.1   127.0.0.1    TCP       44 57551 → 50000 [ACK] Seq=13 Ack=35 Win=2161152 Len=0
12818 94.750889  127.0.0.1   127.0.0.1    TCP       51 57551 → 50000 [PSH, ACK] Seq=13 Ack=35 Win=2161152 Len=7
12819 94.750937  127.0.0.1   127.0.0.1    TCP       44 50000 → 57551 [ACK] Seq=35 Ack=20 Win=2161152 Len=0
12820 94.751546  127.0.0.1   127.0.0.1    TCP       78 50000 → 57551 [PSH, ACK] Seq=35 Ack=20 Win=2161152 Len=34
12821 94.751569  127.0.0.1   127.0.0.1    TCP       44 57551 → 50000 [ACK] Seq=20 Ack=69 Win=2161152 Len=0
```

```
    Total Length: 74
    Identification: 0x4657 (18007)
  > 010. .... = Flags: 0x2, Don't fragment
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 128
    Protocol: TCP (6)
    Header Checksum: 0x0000 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 127.0.0.1
    Destination Address: 127.0.0.1
v Transmission Control Protocol, Src Port: 50000, Dst Port: 57551, Seq: 1, Ack: 13, Len: 34
    Source Port: 50000
    Destination Port: 57551
    [Stream index: 6]
    [Conversation completeness: Complete, WITH_DATA (63)]
    [TCP Segment Len: 34]
    Sequence Number: 1    (relative sequence number)
    Sequence Number (raw): 612905190
    [Next Sequence Number: 35    (relative sequence number)]
    Acknowledgment Number: 13    (relative ack number)
    Acknowledgment number (raw): 4023819347
    0101 .... = Header Length: 20 bytes (5)
  > Flags: 0x018 (PSH, ACK)
    Window: 8442
    [Calculated window size: 2161152]
    [Window size scaling factor: 256]
    Checksum: 0x1925 [unverified]
```

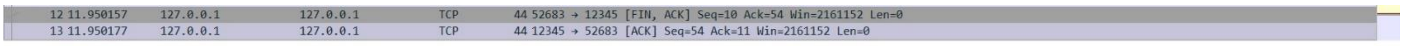
```
0000  02 00 00 00 45 00 00 4a  46 57 40 00 80 06 00 00   ····E··J FW@·····
0010  7f 00 00 01 7f 00 00 01  c3 50 e0 cf 24 88 30 e6   ········ ·P··$·0·
0020  ef d6 9c 53 50 18 20 fa  19 25 00 00 45 76 61 6c   ···SP· · ·%··Eval
0030  75 61 74 65 64 20 72 65  73 75 6c 74 20 66 72 6f   uated re sult fro
0040  6d 20 73 65 72 76 65 72  20 38 32 30 0d 0a         m server  820··
```

Figure 4 Client-Server Data Transferring with PSH+ACK

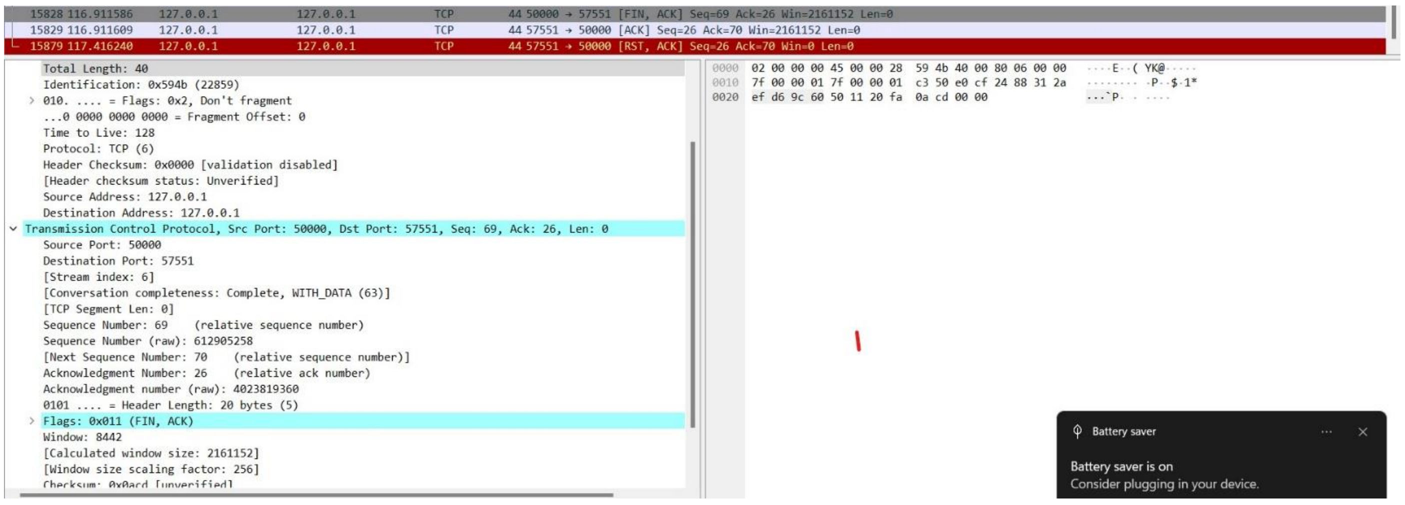
Finish and Acknowledgement:
For Client1 & client 2: (FIN+ACK - ACK)



```
22 18.067619   127.0.0.1   127.0.0.1   TCP   44 52682 → 12345 [FIN, ACK] Seq=10 Ack=54 Win=2161152 Len=0
23 18.067653   127.0.0.1   127.0.0.1   TCP   44 12345 → 52682 [ACK] Seq=54 Ack=11 Win=2161152 Len=0
```



```
12 11.950157   127.0.0.1   127.0.0.1   TCP   44 52683 → 12345 [FIN, ACK] Seq=10 Ack=54 Win=2161152 Len=0
13 11.950177   127.0.0.1   127.0.0.1   TCP   44 12345 → 52683 [ACK] Seq=54 Ack=11 Win=2161152 Len=0
```



```
15828 116.911586   127.0.0.1   127.0.0.1   TCP   44 50000 → 57551 [FIN, ACK] Seq=69 Ack=26 Win=2161152 Len=0
15829 116.911609   127.0.0.1   127.0.0.1   TCP   44 57551 → 50000 [ACK] Seq=26 Ack=70 Win=2161152 Len=0
15879 117.416240   127.0.0.1   127.0.0.1   TCP   44 57551 → 50000 [RST, ACK] Seq=26 Ack=70 Win=0 Len=0
```

```
    Total Length: 40
    Identification: 0x594b (22859)
  > 010. .... = Flags: 0x2, Don't fragment
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 128
    Protocol: TCP (6)
    Header Checksum: 0x0000 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 127.0.0.1
    Destination Address: 127.0.0.1
v Transmission Control Protocol, Src Port: 50000, Dst Port: 57551, Seq: 69, Ack: 26, Len: 0
    Source Port: 50000
    Destination Port: 57551
    [Stream index: 6]
    [Conversation completeness: Complete, WITH_DATA (63)]
    [TCP Segment Len: 0]
    Sequence Number: 69    (relative sequence number)
    Sequence Number (raw): 612905258
    [Next Sequence Number: 70    (relative sequence number)]
    Acknowledgment Number: 26    (relative ack number)
    Acknowledgment number (raw): 4023819360
    0101 .... = Header Length: 20 bytes (5)
  > Flags: 0x011 (FIN, ACK)
    Window: 8442
    [Calculated window size: 2161152]
    [Window size scaling factor: 256]
    Checksum: 0x0acd [unverified]
```

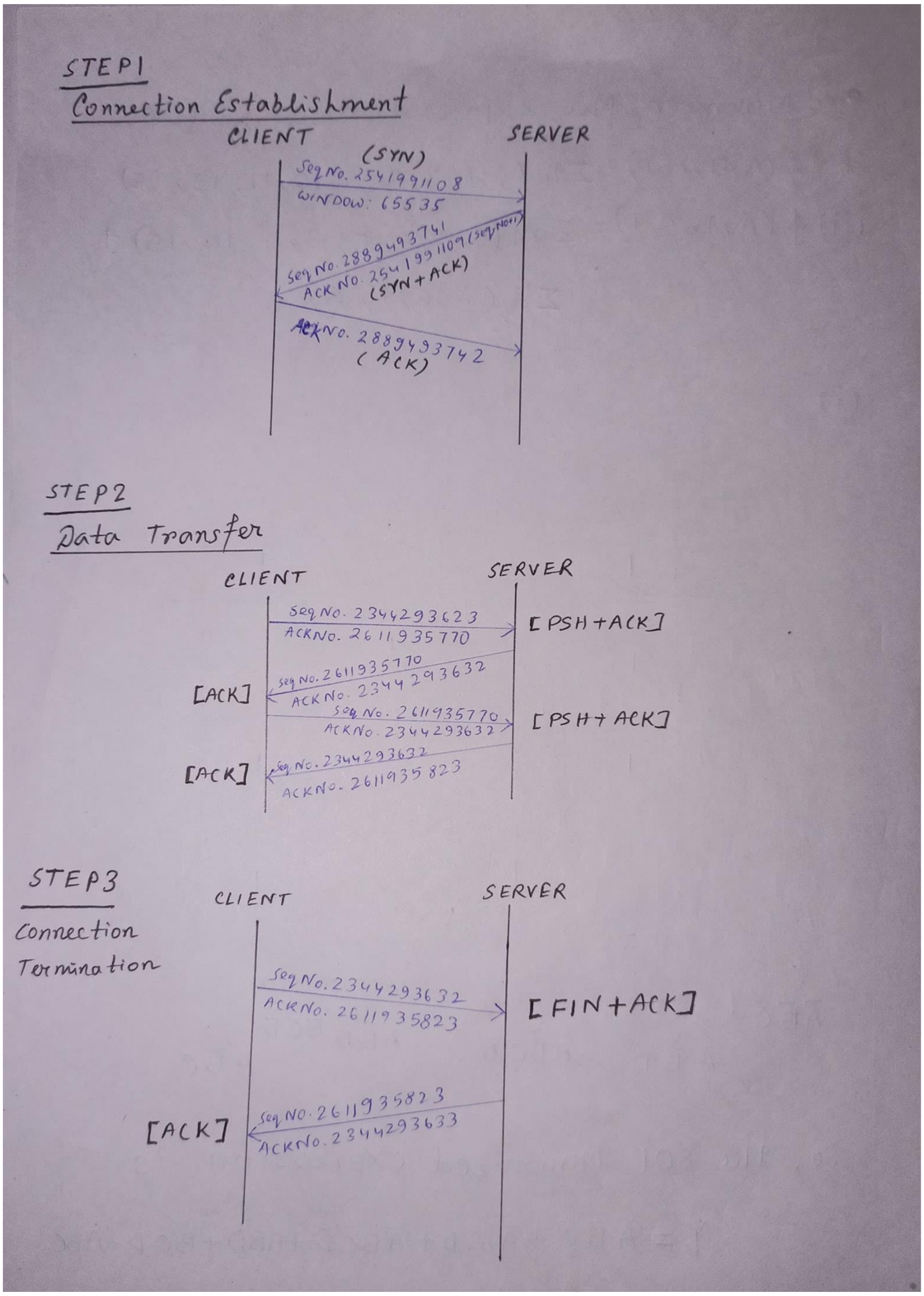
```
0000  02 00 00 00 45 00 00 28  59 4b 40 00 80 06 00 00   ····E··( YK@·····
0010  7f 00 00 01 7f 00 00 01  c3 50 e0 cf 24 88 31 2a   ········ ·P··$·1*
0020  ef d6 9c 60 50 11 20 fa  0a cd 00 00               ···`P·· ····
```

Battery saver

Battery saver is on
Consider plugging in your device.

Figure 5: Both client ends the connection by sending FIN+ACK

TIME SEQUENCE DIAGRAM:



Step1: Client initiates the communication by sending SYN flag with a seq. no to the server and server response back with ack no which is seq. no+1 and then client again sends back ack by adding 1 to the last seq. no. of server.

Step 2: Client sends data packets with PSH+ACK flag to the server and server response back with ACK no.

Step3: Client terminates the connection by sending FIN+ACK to server and server response back ACK to the client for final termination.

ASSIGNMENT 3

The objective of this laboratory exercise is to look at the details of the User Datagram Protocol (UDP). UDP is a transport layer protocol. It is used by many application protocols like DNS, DHCP, SNMP etc., where reliability is not a concern.
To do this exercise you need to install the Wireshark tool, which is widely used to capture and examine a packet trace. Wireshark can be downloaded from www.wireshark.org.

Step1: Capture a Trace

(i) Launch Wireshark
(ii) From Capture→Options select Loopback interface
(iii) Start a capture with a filter of "ip.addr==127.0.0.1 and udp.port==xxxx", where xxxx is the port number used by the UDP server.
(iv) Run the UDP server program on a terminal.
(v) Run multiple instances of the UDP client program on separate terminals and send requests to the sever.
(vi) Stop Wireshark capture

Step2: Inspect the Trace

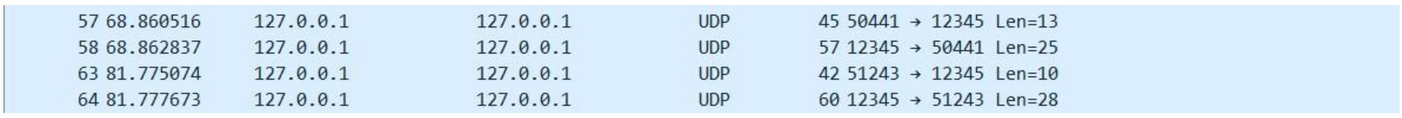
Select different packets in the trace and browse the expanded UDP header and record the following fields:

• Source Port: the port from which the udp segment is sent.
• Destination Port: the port to which the udp segment is sent.
• Length: the length of the UDP segment.

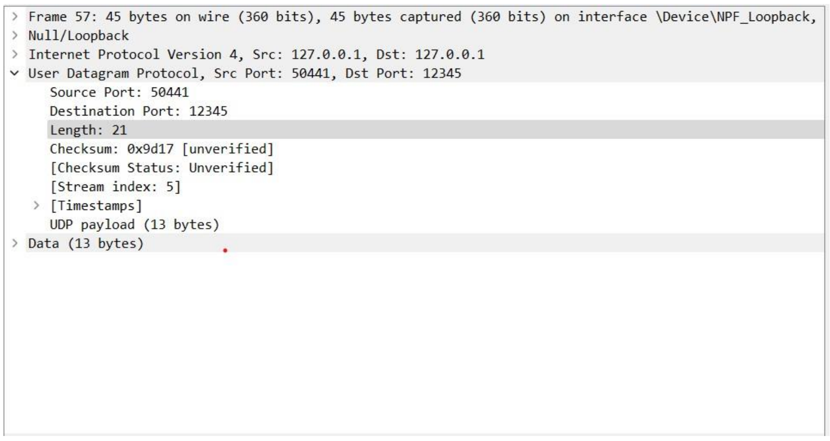
SOURCE PORT:50441

DESTINATION PORT:12345

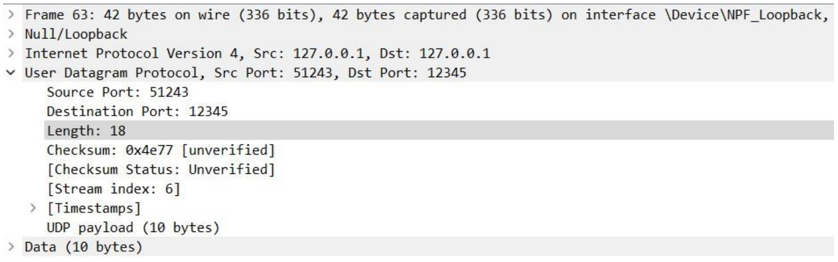
LENGTH:25



```
57 68.860516    127.0.0.1    127.0.0.1    UDP    45 50441 → 12345 Len=13
58 68.862837    127.0.0.1    127.0.0.1    UDP    57 12345 → 50441 Len=25
63 81.775074    127.0.0.1    127.0.0.1    UDP    42 51243 → 12345 Len=10
64 81.777673    127.0.0.1    127.0.0.1    UDP    60 12345 → 51243 Len=28
```



```
> Frame 57: 45 bytes on wire (360 bits), 45 bytes captured (360 bits) on interface \Device\NPF_Loopback,
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
v User Datagram Protocol, Src Port: 50441, Dst Port: 12345
    Source Port: 50441
    Destination Port: 12345
    Length: 21
    Checksum: 0x9d17 [unverified]
    [Checksum Status: Unverified]
    [Stream index: 5]
  > [Timestamps]
    UDP payload (13 bytes)
> Data (13 bytes)
```



```
> Frame 63: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface \Device\NPF_Loopback,
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
v User Datagram Protocol, Src Port: 51243, Dst Port: 12345
    Source Port: 51243
    Destination Port: 12345
    Length: 18
    Checksum: 0x4e77 [unverified]
    [Checksum Status: Unverified]
    [Stream index: 6]
  > [Timestamps]
    UDP payload (10 bytes)
> Data (10 bytes)
```