Date / /

Assignment - B4

Title :- Peer to Peer & Multiuser chat.

Problem statement :-
write a program using UDP/TCP sockets for wired network to implement
(a) peer to peer chat
(b) Multiuser chat
Demonstrate the packets captured traces using wireshark packet analyzer tool for peer to peer mode

Objective :- To understand the implementation of peers-to-peers & multiuser chat.

Outcome :- Students will be able to implement peer-to-peer & multiuser chat & understand its theory

S/w & H/w :- python, IDE, wireshark, linux/windows OS, eclipse

Theory :-
Network Socket :-
It is an internal end-point for sending/recieving data at a single node in a computer network

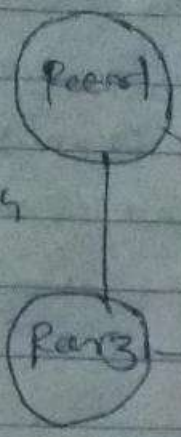
conceptually. It is representation of an endpoint in networking software (protocol stack), such as entry in table. It is a form of system resource.
- Similarly, the term 'port' is used for external endpoints at a node, and the term 'socket' is also used for an internal end-point of local inter-process communication (IPC).

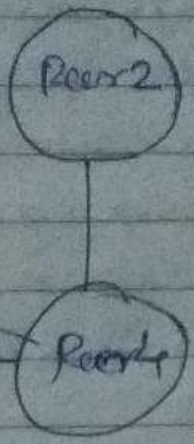
Peer-to-peer chat :-

The principle of communication can generally be categorized into two, client server & peer-to-peer. In client-server environment there is a dedicated servers, while rest of other nodes are acting as clients throughout the whole communication
- whereby peer to peer mode, a node can either be a client or a server depending whether it is a requestor or provider of the service at that specific time.
- Examples of client servers technology : web access, network time & windows login, This however causes an idea of single point of failure, which may cause a devasted damage in case of breakings.

Index table of Peer 1

username, password, IP add, port no for P3 & P4





Index table of peer 2

username, password, IP address, port no for P4

Index table of Port 3

username, password, IP address, port no, for P1 & P4.

Index table of peer 4

username, password, IP address, port no for P1, P2, P3

Multichat :

With multiuser chat, chat rooms are designed, where multiple users can converse simultaneously.

- Similar to IRC, ie chat room have different stat users (visible, hidden, password, protected) & role of participants.
- However MUC has many advantages over IRC. This is how go online in MUC with its globally unique Jabber Identifier, which cannot be occupied by somebody else. Therefore it doesn't need auxiliary some service (such as NickServ) to uniquely identify users

Functions:- MUC offers various functions this allows the server to create a log file over a room, if designed.
- In addition, each user can have different privelages in a chat, one can write in a room or change the subject depending on his/her privelleges.

Wireshark Packet Analyzer tool:-

It is a network packet analyzer, which will capture network packets & tries to display that packet data, as detailed as possible.

Testcase:-

I/P	O/P	Expected O/P	Result
P-to-P clientside: Hi! server, need help Serverside:- Hi client! glad to help	on client side: Hi client! server: glad to help on server side: client: Hi server need help	—11—	Success

Conclusion :-

Successfully implemented a peer-to-peer & multichat program over a wired network & captured the packets using wireshark

A) P2P program

1. Server.java

```
import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;
import java.net.SocketException;
public class Server
{
        private ServerSocket severSocket = null;
        private Socket socket = null;
        private InputStream inStream = null;
        private OutputStream outStream = null;
        public Server()
        {
        }
        public void createSocket() {
        try {
        ServerSocket serverSocket = new ServerSocket(3339);
        while (true) {
        socket = serverSocket.accept();
        inStream = socket.getInputStream();
        outStream = socket.getOutputStream();
        System.out.println("Connected");
        createReadThread();
        createWriteThread();
        }
        } catch (IOException io) {
        io.printStackTrace();
        }
        }
        public void createReadThread() {
        Thread readThread = new Thread() {
        public void run() {
        while (socket.isConnected()) {
        try {
        byte[] readBuffer = new byte[200];
        int num = inStream.read(readBuffer);
        if (num > 0) {
        byte[] arrayBytes = new byte[num];
        System.arraycopy(readBuffer, 0, arrayBytes, 0, num);
        String recvedMessage = new String(arrayBytes, "UTF-8");
        System.out.println("Received message :" + recvedMessage);
        } else {
        notify();
        }
        ;
        //System.arraycopy();
        } catch (SocketException se) {
        System.exit(0);
        } catch (IOException i) {
        i.printStackTrace();
```

```
}
}
}
};
readThread.setPriority(Thread.MAX_PRIORITY);
readThread.start();
}
public void createWriteThread() {
Thread writeThread = new Thread() {
public void run() {
while (socket.isConnected()) {
try {
BufferedReader inputReader = new BufferedReader(new InputStreamReader(System.in));
sleep(100);
String typedMessage = inputReader.readLine();
if (typedMessage != null && typedMessage.length() > 0) {
synchronized (socket) {
outStream.write(typedMessage.getBytes("UTF-8"));
sleep(100);
}
}/* else {
notify();
}*/
;
//System.arraycopy();
} catch (IOException i) {
i.printStackTrace();
} catch (InterruptedException ie) {
ie.printStackTrace();
}
}
}
};
writeThread.setPriority(Thread.MAX_PRIORITY);
writeThread.start();
}
public static void main(String[] args) {
Server chatServer = new Server();
chatServer.createSocket();
}
}
```

2. Client.java

```
import java.io.*;
import java.net.Socket;
import java.net.SocketException;
import java.net.UnknownHostException;
public class Client {
private Socket socket = null;
private InputStream inStream = null;
private OutputStream outStream = null;
```

```
public Client() {
}
public void createSocket() {
try {
socket = new Socket("localHost", 3339);
System.out.println("Connected");
inStream = socket.getInputStream();
outStream = socket.getOutputStream();
createReadThread();
createWriteThread();
} catch (UnknownHostException u) {
u.printStackTrace();
} catch (IOException io) {
io.printStackTrace();
}
}
public void createReadThread() {
Thread readThread = new Thread() {
public void run() {
while (socket.isConnected()) {
try {
byte[] readBuffer = new byte[200];
int num = inStream.read(readBuffer);
if (num > 0) {
byte[] arrayBytes = new byte[num];
System.arraycopy(readBuffer, 0, arrayBytes, 0, num);
String recvedMessage = new String(arrayBytes, "UTF-8");
System.out.println("Received message :" + recvedMessage);
}/* else {
// notify();
}*/
;
//System.arraycopy();
}catch (SocketException se){
System.exit(0);
} catch (IOException i) {
i.printStackTrace();
}
}
}
};
readThread.setPriority(Thread.MAX_PRIORITY);
readThread.start();
}
public void createWriteThread() {
Thread writeThread = new Thread() {
public void run() {
while (socket.isConnected()) {
try {
BufferedReader inputReader = new BufferedReader(new InputStreamReader(System.in));
sleep(100);
String typedMessage = inputReader.readLine();
if (typedMessage != null && typedMessage.length() > 0) {
```

```
synchronized (socket) {
outStream.write(typedMessage.getBytes("UTF-8"));
sleep(100);
}
}
;
//System.arraycopy();
} catch (IOException i) {
i.printStackTrace();
} catch (InterruptedException ie) {
ie.printStackTrace();
}
}
}
};
writeThread.setPriority(Thread.MAX_PRIORITY);
writeThread.start();
}
public static void main(String[] args) throws Exception {
Client myChatClient = new Client();
myChatClient.createSocket();
/*myChatClient.createReadThread();
myChatClient.createWriteThread();*/
}
}
```

B) Multiuser chat (in python)

1. server.py

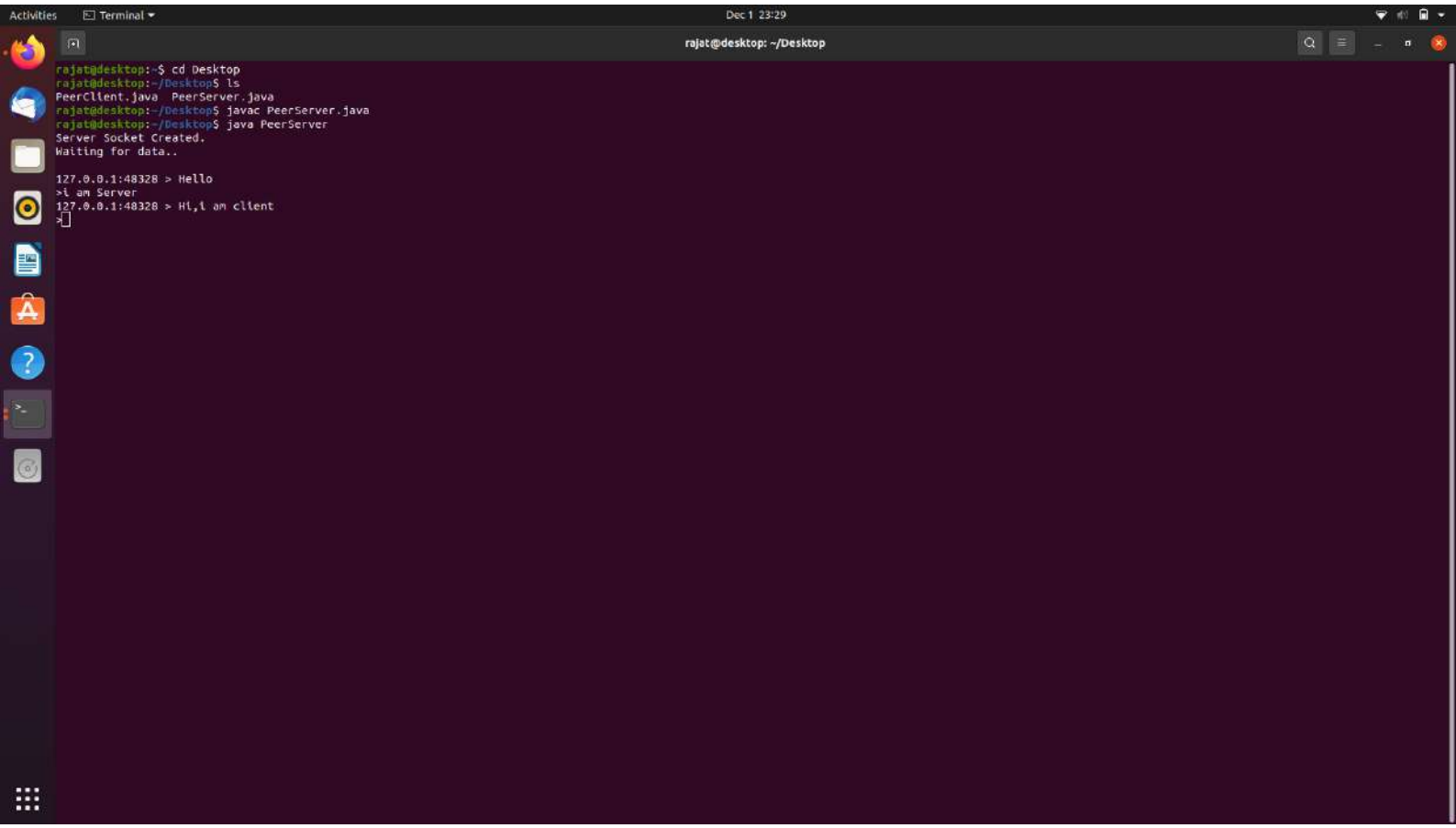
```
import socket
import threading

# Connection Data
host = '127.0.0.1'
port = 55555

# Starting Server
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind((host, port))
server.listen()

# Lists For Clients and Their Nicknames
clients = []
nicknames = []

# Sending Messages To All Connected Clients
def broadcast(message):
    for client in clients:
        client.send(message)

# Handling Messages From Clients
```

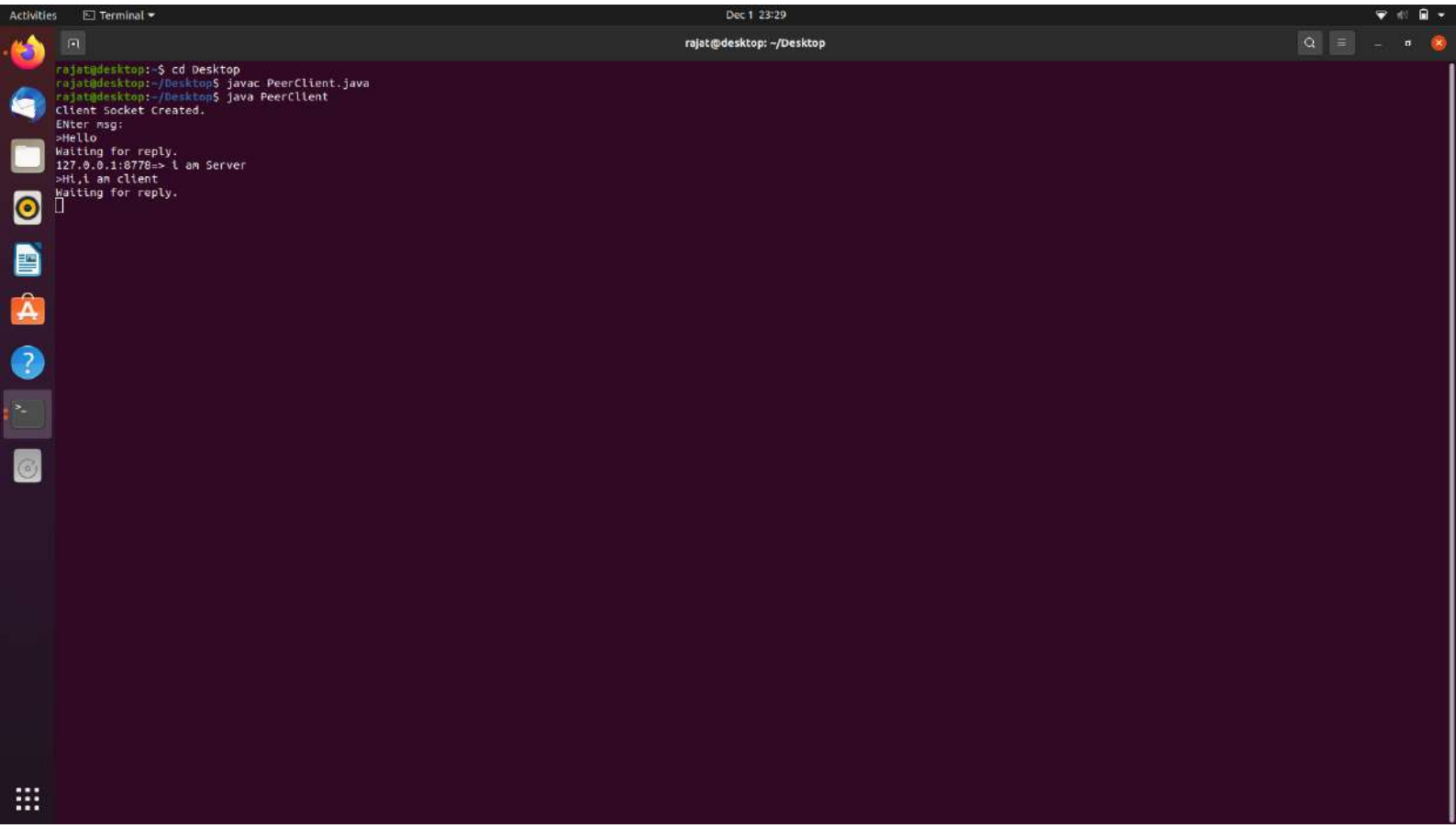
```
def handle(client):
    while True:
        try:
            # Broadcasting Messages
            message = client.recv(1024)
            broadcast(message)
        except:
            # Removing And Closing Clients
            index = clients.index(client)
            clients.remove(client)
            client.close()
            nickname = nicknames[index]
            broadcast('{} left!'.format(nickname).encode('ascii'))
            nicknames.remove(nickname)
            break

# Receiving / Listening Function
def receive():
    while True:
        # Accept Connection
        client, address = server.accept()
        print("Connected with {}".format(str(address)))

        # Request And Store Nickname
        client.send('NICK'.encode('ascii'))
        nickname = client.recv(1024).decode('ascii')
        nicknames.append(nickname)
        clients.append(client)

        # Print And Broadcast Nickname
        print("Nickname is {}".format(nickname))
        broadcast("{} joined!".format(nickname).encode('ascii'))
        client.send('Connected to server!'.encode('ascii'))

        # Start Handling Thread For Client
        thread = threading.Thread(target=handle, args=(client,))
        thread.start()

receive()
```

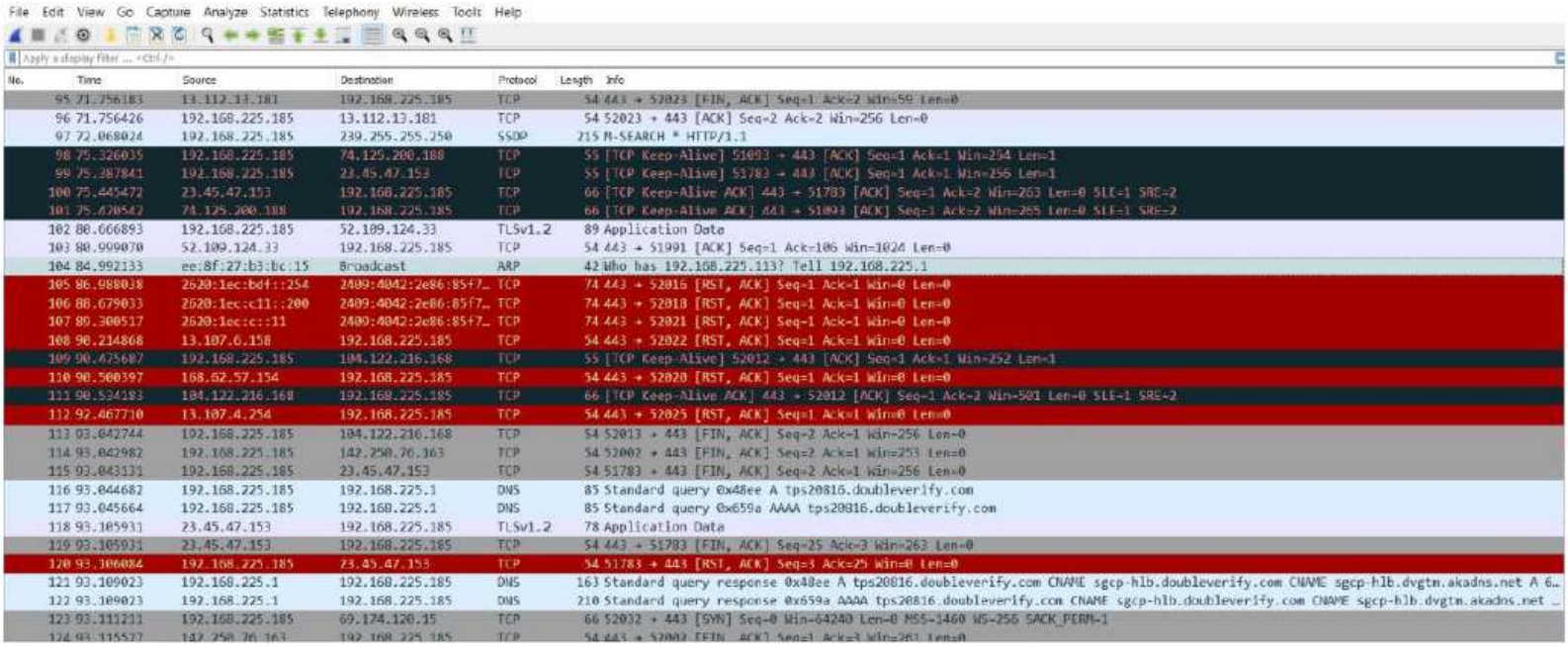
2. client.py

```
import socket
import threading

# Choosing Nickname
nickname = input("Choose your nickname: ")

# Connecting To Server
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client.connect(('127.0.0.1', 55555))
```

```
# Listening to Server and Sending Nickname
def receive():
    while True:
        try:
            # Receive Message From Server
            # If 'NICK' Send Nickname
            message = client.recv(1024).decode('ascii')
            if message == 'NICK':
                client.send(nickname.encode('ascii'))
            else:
                print(message)
        except:
            # Close Connection When Error
            print("An error occured!")
            client.close()
            break

# Sending Messages To Server
def write():
    while True:
        message = '{}: {}'.format(nickname, input(''))
        client.send(message.encode('ascii'))

# Starting Threads For Listening And Writing
receive_thread = threading.Thread(target=receive)
receive_thread.start()

write_thread = threading.Thread(target=write)
write_thread.start()
```


Activities
Terminal
Dec 1 23:29
rajat@desktop: ~/Desktop
rajat@desktop:~$ cd Desktop
rajat@desktop:~/Desktop$ ls
PeerClient.java PeerServer.java
rajat@desktop:~/Desktop$ javac PeerServer.java
rajat@desktop:~/Desktop$ java PeerServer
Server Socket Created.
Waiting for data..
127.0.0.1:48328 > Hello
>i am Server
127.0.0.1:48328 > Hi,i am client
>


Activities
Terminal
Dec 1 23:29
rajat@desktop: ~/Desktop
rajat@desktop:~$ cd Desktop
rajat@desktop:~/Desktop$ javac PeerClient.java
rajat@desktop:~/Desktop$ java PeerClient
Client Socket Created.
ENter msg:
>Hello
Waiting for reply.
127.0.0.1:8778=> i am Server
>Hi,i am client
Waiting for reply.


File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
No. Time Source Destination Protocol Length Info
95 71.756183 13.112.13.181 192.168.225.185 TCP 54 443 → 52023 [FIN, ACK] Seq=1 Ack=2 Win=59 Len=0
96 71.756426 192.168.225.185 13.112.13.181 TCP 54 52023 → 443 [ACK] Seq=2 Ack=2 Win=256 Len=0
97 72.068024 192.168.225.185 239.255.255.250 SSDP 215 M-SEARCH * HTTP/1.1
98 75.326035 192.168.225.185 74.125.200.188 TCP 55 [TCP Keep-Alive] 51093 → 443 [ACK] Seq=1 Ack=1 Win=254 Len=1
99 75.387841 192.168.225.185 23.45.47.153 TCP 55 [TCP Keep-Alive] 51783 → 443 [ACK] Seq=1 Ack=1 Win=256 Len=1
100 75.445472 23.45.47.153 192.168.225.185 TCP 66 [TCP Keep-Alive ACK] 443 → 51783 [ACK] Seq=1 Ack=2 Win=263 Len=0 SLE=1 SRE=2
101 75.670542 74.125.200.188 192.168.225.185 TCP 66 [TCP Keep-Alive ACK] 443 → 51093 [ACK] Seq=1 Ack=2 Win=265 Len=0 SLE=1 SRE=2
102 80.666893 192.168.225.185 52.109.124.33 TLSv1.2 89 Application Data
103 80.999070 52.109.124.33 192.168.225.185 TCP 54 443 → 51991 [ACK] Seq=1 Ack=106 Win=1024 Len=0
104 84.992133 ee:8f:27:b3:bc:15 Broadcast ARP 42 Who has 192.168.225.113? Tell 192.168.225.1
105 86.988038 2620:1ec:bd4::254 2409:4042:2e86:85f7… TCP 74 443 → 52016 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
106 88.679033 2620:1ec:c11::200 2409:4042:2e86:85f7… TCP 74 443 → 52018 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
107 89.300517 2620:1ec:c::11 2409:4042:2e86:85f7… TCP 74 443 → 52021 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
108 90.214868 13.107.6.158 192.168.225.185 TCP 54 443 → 52022 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
109 90.475607 192.168.225.185 104.122.216.168 TCP 55 [TCP Keep-Alive] 52012 → 443 [ACK] Seq=1 Ack=1 Win=252 Len=1
110 90.500397 168.62.57.154 192.168.225.185 TCP 54 443 → 52020 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
111 90.534183 104.122.216.168 192.168.225.185 TCP 66 [TCP Keep-Alive ACK] 443 → 52012 [ACK] Seq=1 Ack=2 Win=501 Len=0 SLE=1 SRE=2
112 92.467710 13.107.4.254 192.168.225.185 TCP 54 443 → 52025 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
113 93.042744 192.168.225.185 104.122.216.168 TCP 54 52013 → 443 [FIN, ACK] Seq=2 Ack=1 Win=256 Len=0
114 93.042982 192.168.225.185 142.250.76.163 TCP 54 52002 → 443 [FIN, ACK] Seq=2 Ack=1 Win=253 Len=0
115 93.043131 192.168.225.185 23.45.47.153 TCP 54 51783 → 443 [FIN, ACK] Seq=2 Ack=1 Win=256 Len=0
116 93.044682 192.168.225.185 192.168.225.1 DNS 85 Standard query 0x48ee A tps20816.doubleverify.com
117 93.045664 192.168.225.185 192.168.225.1 DNS 85 Standard query 0x659a AAAA tps20816.doubleverify.com
118 93.105931 23.45.47.153 192.168.225.185 TLSv1.2 78 Application Data
119 93.105931 23.45.47.153 192.168.225.185 TCP 54 443 → 51783 [FIN, ACK] Seq=25 Ack=3 Win=263 Len=0
120 93.106084 192.168.225.185 23.45.47.153 TCP 54 51783 → 443 [RST, ACK] Seq=3 Ack=25 Win=0 Len=0
121 93.109023 192.168.225.1 192.168.225.185 DNS 163 Standard query response 0x48ee A tps20816.doubleverify.com CNAME sgcp-hlb.doubleverify.com CNAME sgcp-hlb.dvgtm.akadns.net A 6…
122 93.109023 192.168.225.1 192.168.225.185 DNS 210 Standard query response 0x659a AAAA tps20816.doubleverify.com CNAME sgcp-hlb.doubleverify.com CNAME sgcp-hlb.dvgtm.akadns.net …
123 93.111211 192.168.225.185 69.174.120.15 TCP 66 52032 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
124 93.115527 142.250.76.163 192.168.225.185 TCP 54 443 → 52002 [FIN, ACK] Seq=1 Ack=3 Win=263 Len=0