Computer Networks Lab
CS302

Assignment 5 Report

Ragul N S - 191CS146 &
Rakshith H R – 191CS148

1) Develop a code to illustrate a secure socket connection between client and server.

Solution: we have used the libraries such as ssl, os, socket and time in python to do this program. We have attached the code below.

Server.py:

```
# Develop a code to illustrate a secure socket connection between
client and server.

import ssl
import socket
import datetime
import time

ipAddress = "127.0.0.1"
port = 15001

serverSocket = socket.socket()
serverSocket.bind((ipAddress, port))

serverSocket.listen()
print("Server listening:")

while(True):
    (clientConnection, clientAddress) = serverSocket.accept()
    secureClientSocket = ssl.wrap_socket(clientConnection,
server_side=True, ca_certs="./DemoCA.pem",
certfile="./DemoSvr.crt",
                                         keyfile="./DemoSvr.key",
cert_reqs=ssl.CERT_REQUIRED, ssl_version=ssl.PROTOCOL_TLSv1_2)

    client_cert = secureClientSocket.getpeercert()
    clt_subject = dict(item[0] for item in client_cert['subject'])
    clt_commonName = clt_subject['commonName']
    if not client_cert:
        raise Exception("Unable to get the certificate from the
client")
```

```
    if clt_commonName != 'DemoClt':
        raise Exception("Incorrect common name in client
certificate")
    t1 = ssl.cert_time_to_seconds(client_cert['notBefore'])
    t2 = ssl.cert_time_to_seconds(client_cert['notAfter'])
    ts = time.time()
    if ts < t1:
        raise Exception("Client certificate not yet active")
    if ts > t2:
        raise Exception("Expired client certificate")
    serverTimeNow = "%s" % datetime.datetime.now()
    secureClientSocket.send(serverTimeNow.encode())
    print("Securely sent %s to %s" % (serverTimeNow,
clientAddress))
    secureClientSocket.close()
```

Client.py:

```
# Develop a code to illustrate a secure socket connection between
client and server.

import socket
import ssl
import os
import time


sslServerIP = "127.0.0.1"
sslServerPort = 15001


context = ssl.SSLContext()
context.verify_mode = ssl.CERT_REQUIRED


context.load_verify_locations("./DemoCA.pem")
```

```
context.load_cert_chain(certfile="./DemoClt.crt",
keyfile="./DemoClt.key")

clientSocket = socket.socket()
secureClientSocket = context.wrap_socket(clientSocket)
secureClientSocket.connect((sslServerIP, sslServerPort))

server_cert = secureClientSocket.getpeercert()
subject = dict(item[0] for item in server_cert['subject'])
commonName = subject['commonName']

if not server_cert:
    raise Exception("Unable to retrieve server certificate")

if commonName != 'DemoSvr':
    raise Exception("Incorrect common name in server certificate")

notAfterTimestamp =
ssl.cert_time_to_seconds(server_cert['notAfter'])
notBeforeTimestamp =
ssl.cert_time_to_seconds(server_cert['notBefore'])
currentTimeStamp = time.time()

if currentTimeStamp > notAfterTimestamp:
    raise Exception("Expired server certificate")

if currentTimeStamp < notBeforeTimestamp:
    raise Exception("Server certificate not yet active")

msgReceived = secureClientSocket.recv(1024)
print("Secure communication received from server:%s" %
msgReceived.decode())

secureClientSocket.close()
clientSocket.close()
```

2.Capture TCP Packets and:

A. Analyze the three-way handshake during the establishment of the communication.

B. Identify if there are any retransmitted segments.

Solution:

A) TCP uses a three-way handshake to establish a reliable connection. The connection is full duplex, and both sides synchronize (SYN) and acknowledge (ACK) each other. The exchange of these four flags is performed in three steps: SYN, SYN-ACK, ACK

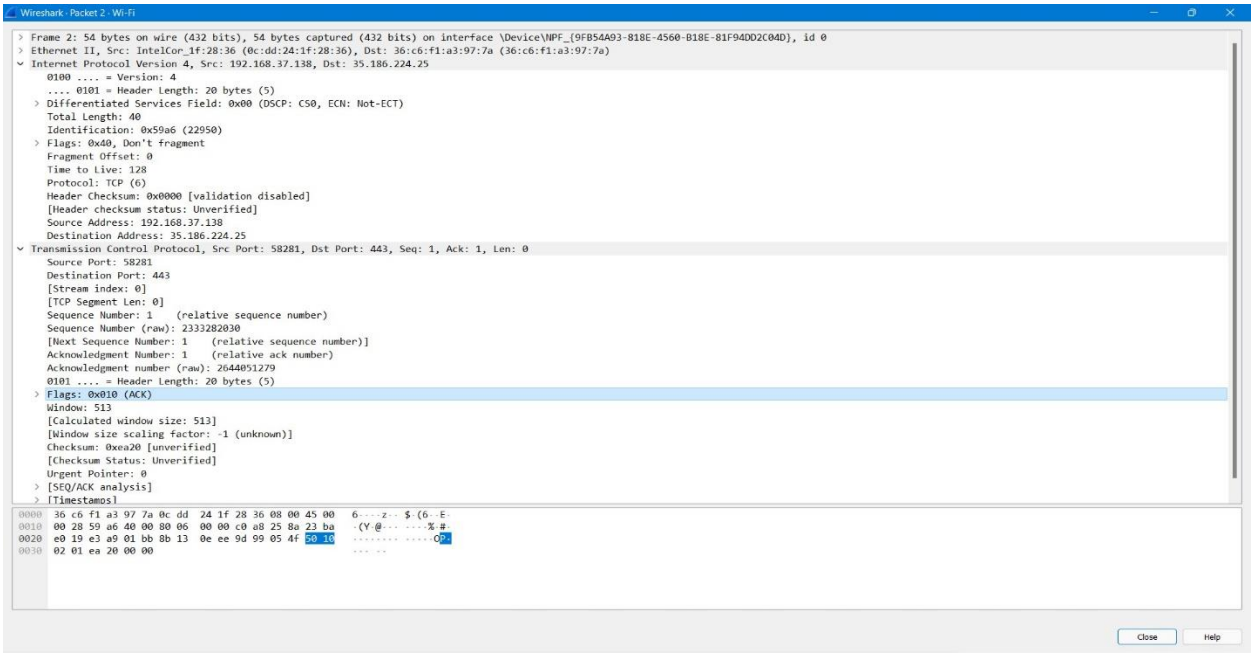
computer A transmits a SYNchronize packet to computer B, which sends back a SYNchronize-ACKnowledge packet to A. Computer A then transmits an ACKnowledge packet to B, and the connection is established.

We have added the screenshots of SYN, ACK, SYN-ACK packets captured from Wireshark.

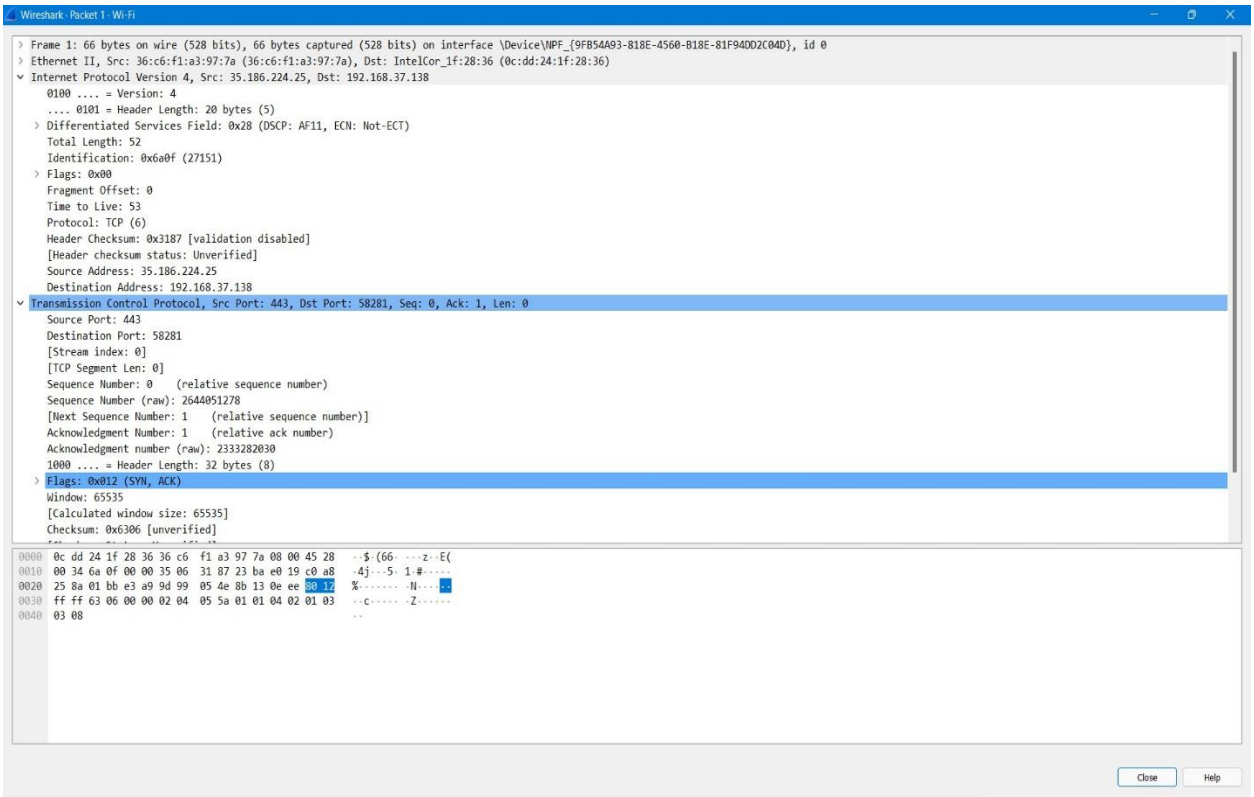
TCP message types

Message	Description
SYN	Used to initiate and establish a connection. It also helps you to synchronize sequence numbers between devices.
ACK	Helps to confirm to the other side that it has received the SYN.
SYN-ACK	SYN message from local device and ACK of the earlier packet.

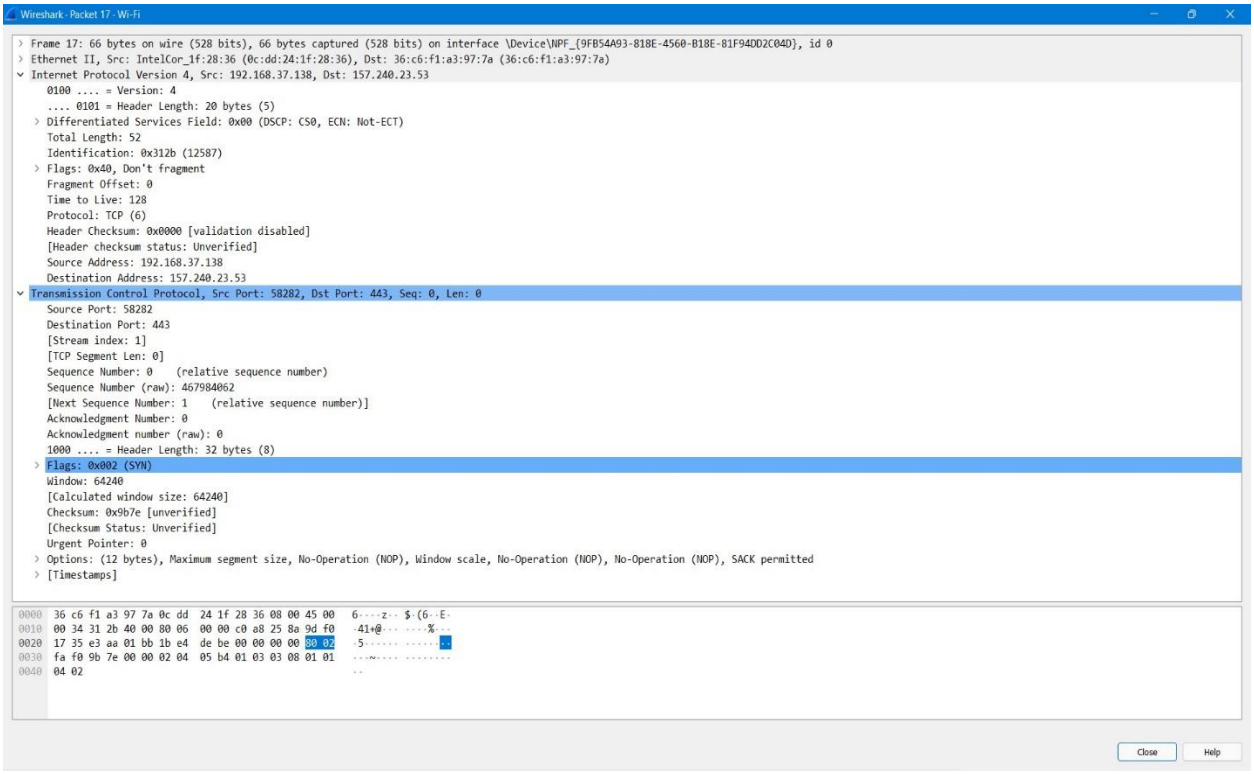
ACK Packet Screenshot:



SYN-ACK Packet Screenshot:



SYN Packet Screenshot:



B) There were no retransmission segments when we tried to analyze in the Wireshark. But retransmission will be there when a transferred packet gets lost in the middle and didn't reach the destination, then the source might retransmit the same packet again acknowledgement comes from the destination. In our case we hope none of the packets were lost as we had a reliable internet connection and that's why no retransmission of the segments was observer.

Thank You