Jadavpur University

Data Communication and

Computer Networks

Lab Report

Name : Himangshu Bauri

Roll : 002210503009

Dept. : CSE

Class : MCA 2nd Year- 1st Sem

**Assignment 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.

GET\_TIME

CLIENT ---------------------------------------------→ SERVER

Date and time

CLIENT ←-----------------------------------------------SERVER

disconnect

CLIENT ---------------------------------------------→ SERVER

connection closed

CLIENT ←-------------------------------------------SERVER

**Code:**

Server side

import socket

import time

#create a socket object

server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

#bind the port

server\_socket.bind(('127.0.0.1', 12345))

#queue upto 5 request

server\_socket.listen(5)

while True:

print("Waiting for connection")

#establish a connection

client\_socket, addr = server\_socket.accept()

print("Got a connection from %s" %str(addr))

currentTime = time.ctime(time.time()) + "\r\n"

try:

client\_socket.send(currentTime.encode('ascii'))

except:

print("Exited by the user")

client\_socket.close()

server\_socket.close();

Client Side

import socket

#create a socket object

client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

#connection to hostname on the port

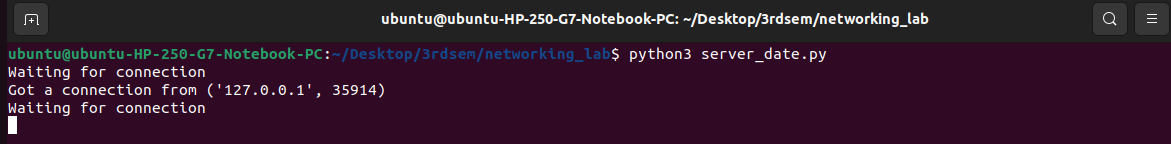
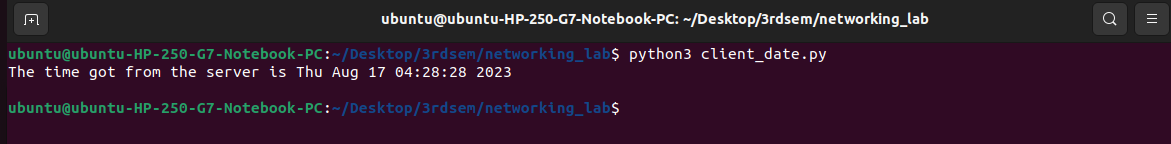
client\_socket.connect(('127.0.0.1', 12345))

data = client\_socket.recv(1024) #receive the data from the server

client\_socket.close()

print("The time got from the server is %s" %data.decode('ascii'))

**Sample Run :**

****

**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 formats

for the request/reply messages.

Equation

CLIENT ---------------------------------------------→ SERVER

Result

CLIENT ←-----------------------------------------------SERVER

disconnect

CLIENT ---------------------------------------------→ SERVER

connection closed

CLIENT ←-------------------------------------------SERVER

**Code:**

Server side

import socket

import ast

def evaluate\_expression(expression):

try:

# Using the 'eval' function to evaluate the mathematical expression received from the client.

result = eval(expression)

return str(result)

except Exception as e:

return "Error: " + str(e)

def main():

host = '127.0.0.1'

port = 5050

#create a socket object

server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

#bind the port

server\_socket.bind((host, port))

server\_socket.listen(5) # Allow up to 5 queued connections

print("Server is listening on {}:{}".format(host, port))

while True:

#establish a connection

client\_socket, client\_address = server\_socket.accept()

print("Connection established with {}:{}".format(client\_address[0], client\_address[1]))

try:

while True:

data = client\_socket.recv(1024).decode('utf-8')

if data.lower() == "disconnect":

print("Client {}:{} disconnected.".format(client\_address[0], client\_address[1]))

break

result = evaluate\_expression(data)

“”” Converting the result of the expression evaluation to a UTF-8 encoded byte string and sending it back to the client over the network using the 'client\_socket'.”””

client\_socket.sendall(result.encode('utf-8'))

except Exception as e:

print("Error occurred while handling client request:", e)

finally:

# Close the client socket

client\_socket.close()

print("Connection with {}:{} closed\n".format(client\_address[0], client\_address[1]))

if \_\_name\_\_ == "\_\_main\_\_":

main()

Client Side

import socket

def main():

host = '127.0.0.1'

port = 5050

#create a socket object

client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

try:

#connection to hostname on the port

client\_socket.connect((host, port))

while True:

expression = input("Enter an arithmetic expression (e.g., 2+3) or 'Disconnect' to quit: ")

if expression.lower() == 'disconnect':

“”” Converting the result of the expression evaluation to a UTF-8 encoded byte string and sending it back to the client over the network using the 'client\_socket'.”””

client\_socket.sendall(expression.encode('utf-8'))

print("Disconnected from the server.")

break

“”” Converting the result of the expression evaluation to a UTF-8 encoded byte string and sending it back to the client over the network using the 'client\_socket'.”””

client\_socket.sendall(expression.encode('utf-8'))

result = client\_socket.recv(1024).decode('utf-8')

print("Result:", result)

except Exception as e:

print("Error occurred:", e)

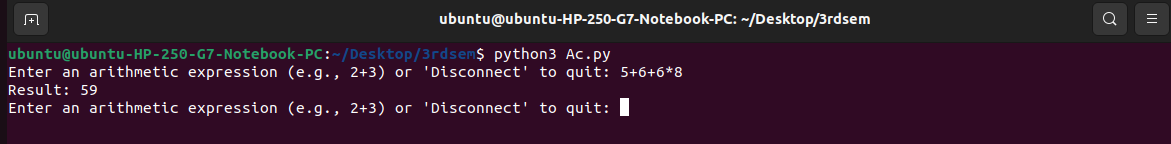
finally:

client\_socket.close()

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Sample Run :**

****

**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.

Name

CLIENT ---------------------------------------------→ SERVER

Address

CLIENT ←-----------------------------------------------SERVER

disconnect

CLIENT ---------------------------------------------→ SERVER

connection closed

CLIENT ←-------------------------------------------SERVER

**Code:**

Server side

import socket

def find\_address(student\_name):

try:

with open('student\_addresses.txt', 'r') as file:

for line in file:

name, address = line.strip().split(',')

if name.lower() == student\_name.lower():

return address

except Exception as e:

print("Error:", e)

return "Address not found"

def main():

#create a socket object

server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

host = '127.0.0.1'

port = 12345

#bind the port

server\_socket.bind((host, port))

print(f"UDP server listening on {host}:{port}")

while True:

# Receiving data from the client using the UDP socket 'server\_socket'.

data, client\_address = server\_socket.recvfrom(1024)

# Decoding the received data, which is in bytes, into a UTF-8 encoded string.

student\_name = data.decode('utf-8')

if student\_name.lower() == 'exit':

print("Client disconnected.")

break

address = find\_address(student\_name)

“”” Converting the 'address' string into a UTF-8 encoded byte strin and sending it back to the client using the 'server\_socket'. “””

server\_socket.sendto(address.encode('utf-8'), client\_address)

server\_socket.close()

if \_\_name\_\_ == '\_\_main\_\_':

main()

Client Side

import socket

def main():

#create a socket object

client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

host = '127.0.0.1'

port = 12345

print("UDP client running.")

while True:

student\_name = input("Enter student's name (or 'exit' to quit): ")

“”” Converting the 'student\_name' string into a UTF-8 encoded byte string

and sending it to the server using the 'client\_socket'. “””

client\_socket.sendto(student\_name.encode('utf-8'), (host, port))

if student\_name.lower() == 'exit':

print("Exiting.")

break

# Receiving data from the server using the UDP socket 'client\_socket'.

address, server\_address = client\_socket.recvfrom(1024)

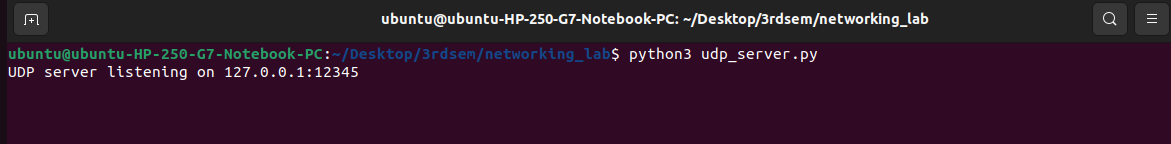
print(f"Permanent address of {student\_name}: {address.decode('utf-8')}")

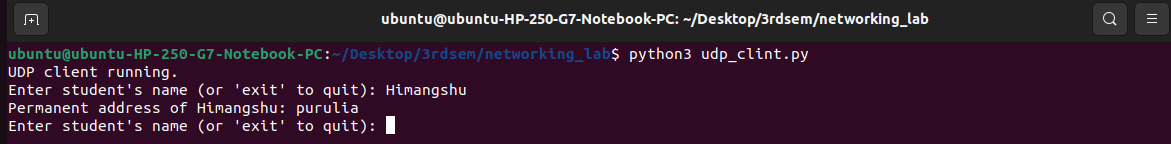
client\_socket.close()

if \_\_name\_\_ == '\_\_main\_\_':

main()

**Sample Run :**

****

****

**Assignment 2**

**Problem Statement**

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](http://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 it’s 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.

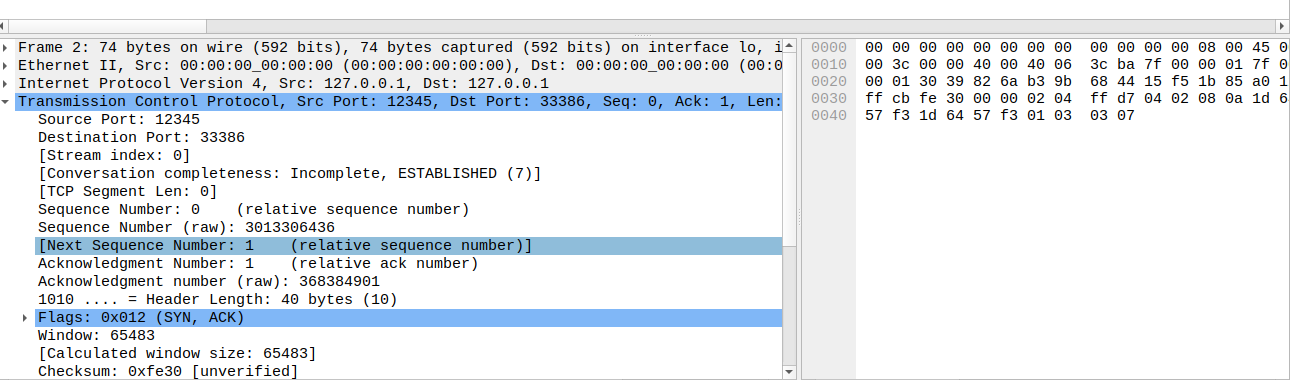
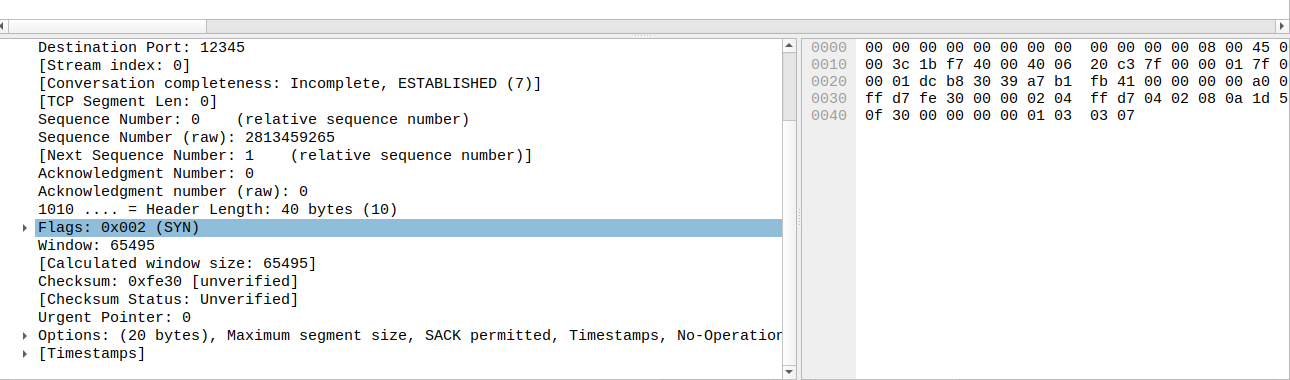
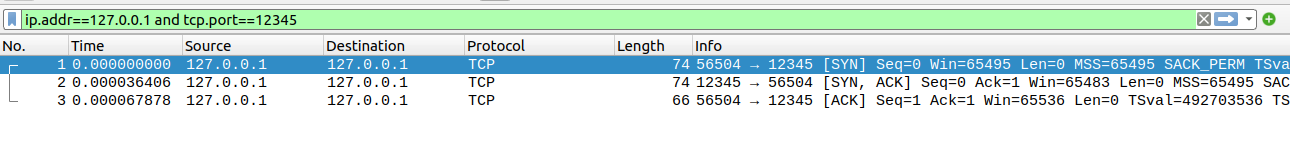
For all the above three segments record the values of the sequence number and acknowledgment fields. Draw a

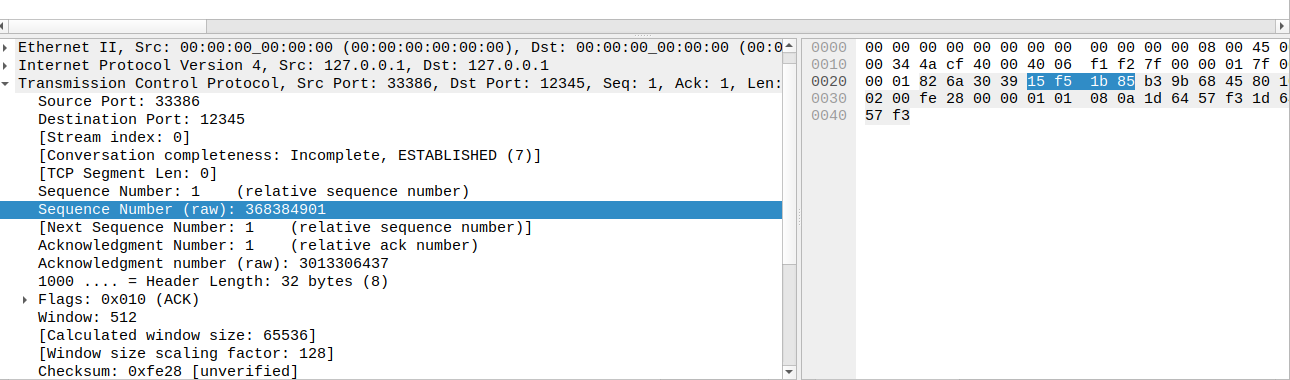
time sequence diagram of the three-way handshake for TCP connection termination in your trace. Do it for all the

client connections.

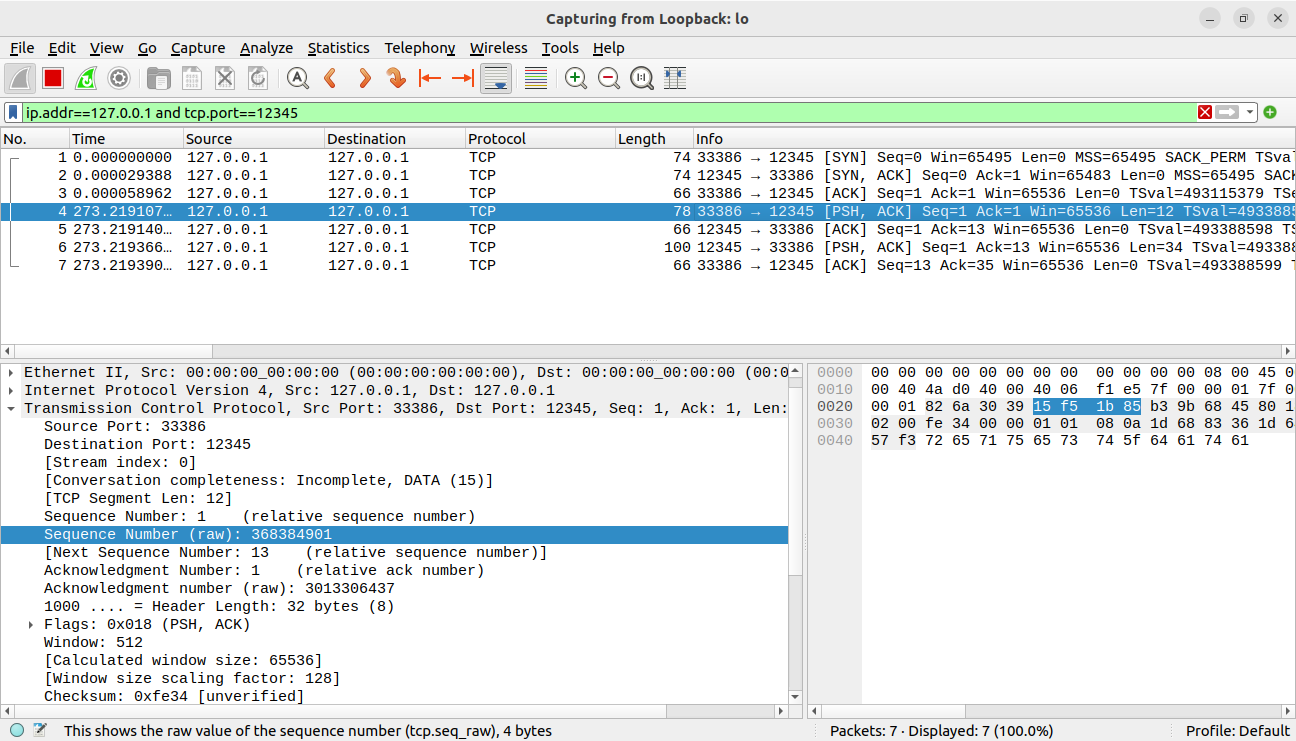
**Capturing in Wireshark**

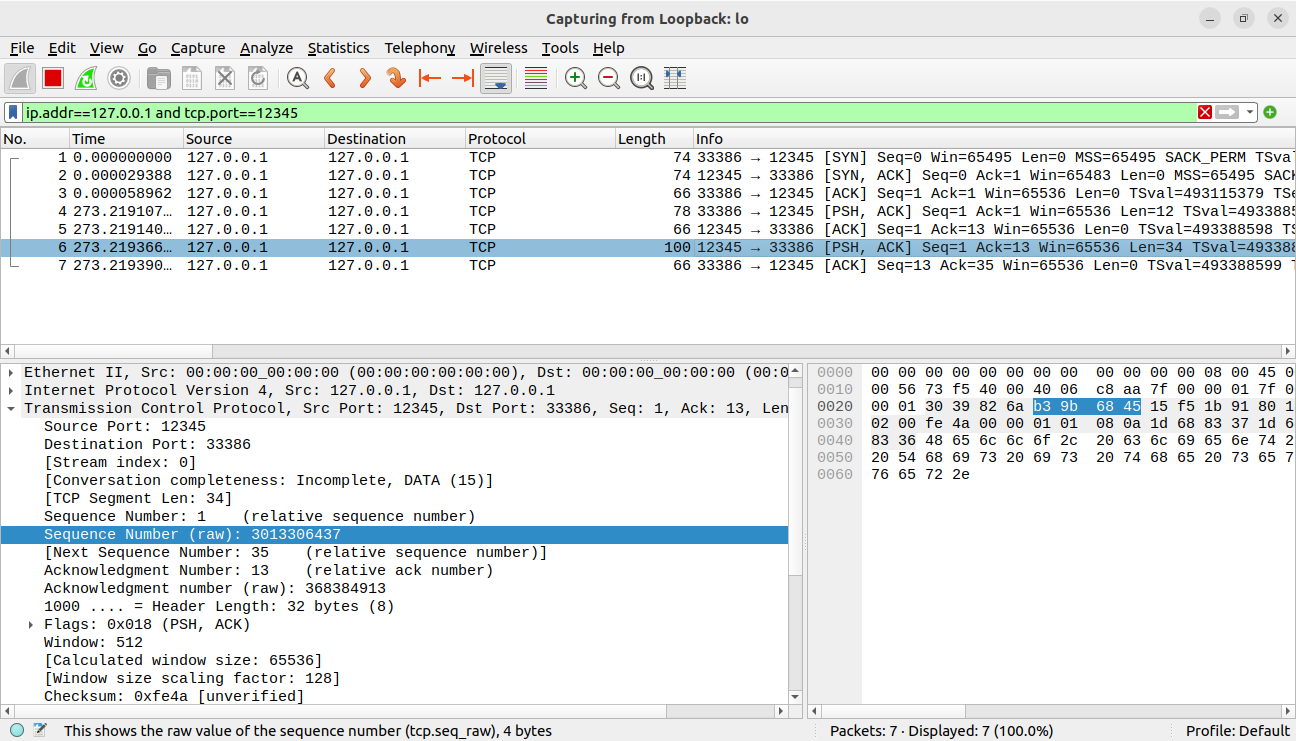
**TCP Connection Establishment:**



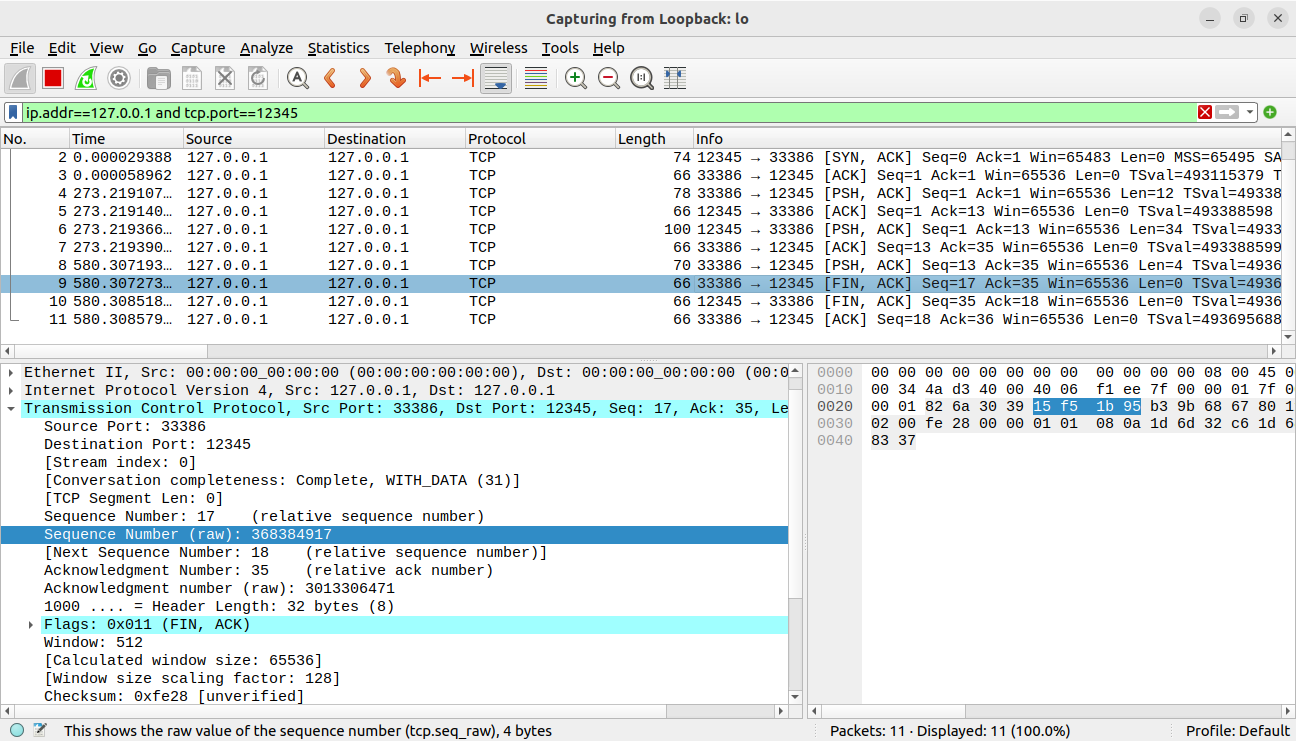


**TCP Data Transfer:**

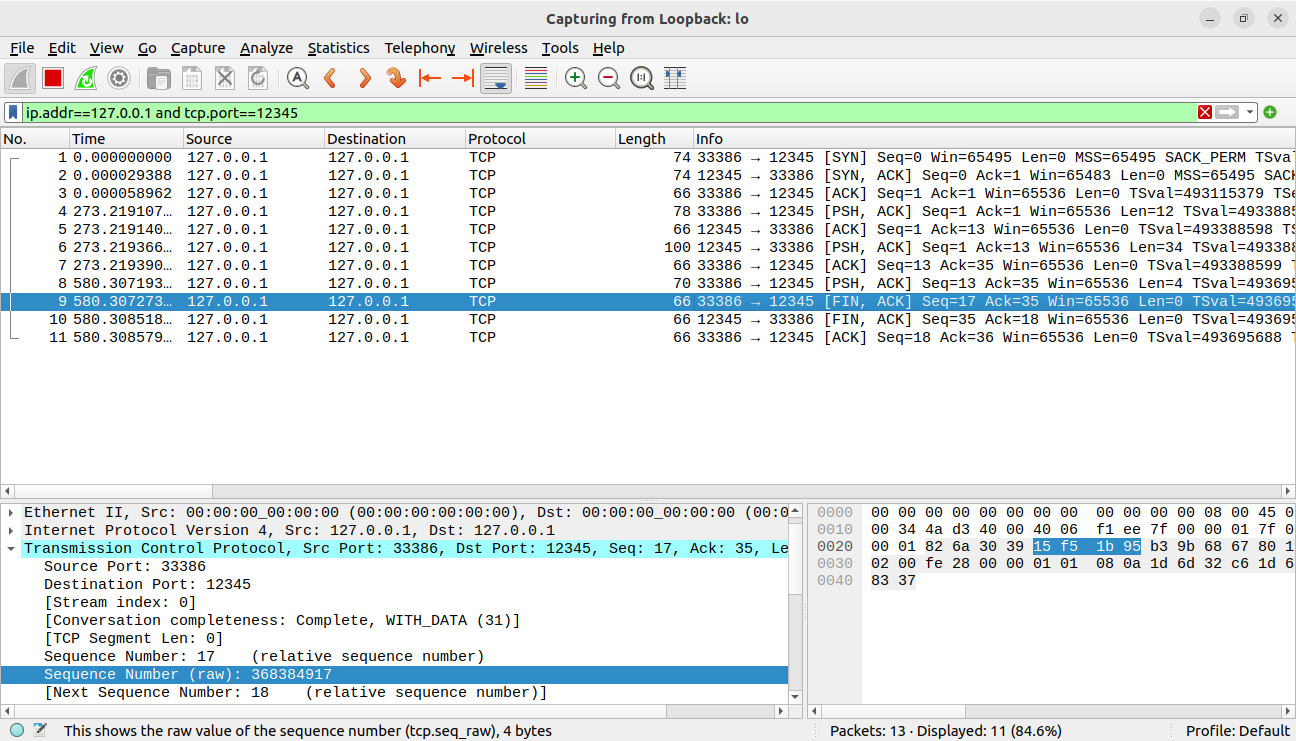


****

**TCP Connection Termination:**

****

**FINAL CAPTURE:**

****

**Time Sequence Diagram:**

**Connection Establishment:**

**Client Server**

SYN = 3945481377

SYN = 2219491575

ACK = 3945481378(SYN+1)

ACK = 2219491576(SYN+1)

**Data Transfer:**

**Client Server**

PSH = 902803996

ACK = 3541257455

ACK = 902804030

**Connection Termination:**

ACK = 902804030

FIN = 3541257459

ACK = 902804030

FIN = 3541257460

ACK = 3541257460

ACK = 902804031

**Assignment 3**

**Problem Statement**

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](http://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.

**Capturing in Wireshark**

