

Mini Project Report

of

Computer Networks LAB

Simulation of OSI Model

SUBMITTED

BY

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Registraction NO | Roll No | Section |
| Abhishek Jhawar | 200905144 | 25 | A |
| Sujeet Amberkar | 200905092 | 15 | A |

1. Introduction

1.1General Introduction :

The Open Systems Interconnection (OSI) model describes seven layers that computer systems use to communicate over a network. It was the first standard model for network communications, adopted by all major computer and telecommunication companies.

Hardware and Software Requirements

Python3

Linux Operating System

Objectives

The main theme of the project is to stepwise simulate the OSI models. Here, our aim is to mainly simulate OSI model layers with the help of python programming language.

ACKNOWLEDGEMENT

We want to express our most significant appreciation to all the individuals who have helped and supported us throughout the project. We are thankful to our COMPUTER NETWORKS LAB teachers for their ongoing support during the project, from initial advice and encouragement, which led to the final report of this project. We would also like to thank Mr. Radhakrishna Bhat sir , who always guided us in the computer lab and also in the theory classes. A special acknowledgment goes to our classmates who helped us complete the project by exchanging interesting ideas and sharing their experience

We wish to thank our parents for their undivided support and interest in our interests and well-being, inspiring us and encouraging us to go our own way. Without their support and blessing, we wouldn't be in this position.In the end, we want to thank our friends who displayed appreciation for our work and motivated us to continue our work.

This life is like a steps of study. Till the end of life, there is something to study. This project is just a small try by us in this very short span. But honestly learnt a lot of things from this. We have the hope that your guidelines will always be with us so that we would go in a right path both in subject and also in life.

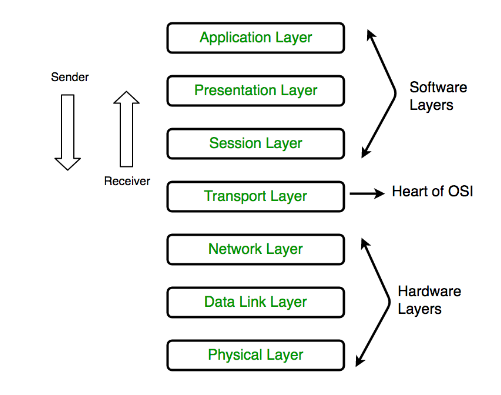
Thanking You All

Table of Content

**Chapter 1:**

**OSI Model layers**

* 1. **Pictographic representation of OSI models**



1.2.1.

Application Layer

The application layer is used by end-user software such as web browsers and email clients. It provides protocols that allow software to send and receive information and present meaningful data to users. A few examples of application layer protocols are the Hypertext Transfer Protocol (HTTP), File Transfer Protocol (FTP), Post Office Protocol (POP), Simple Mail Transfer Protocol (SMTP), and Domain Name System (DNS).

**1.2.2.**

**Presentation Layer**

The presentation layer prepares data for the application layer. It defines how two devices should encode, encrypt, and compress data so it is received correctly on the other end. The presentation layer takes any data transmitted by the application layer and prepares it for transmission over the session layer.

**1.2.3**

**Session Layer**

The session layer creates communication channels, called sessions, between devices. It is responsible for opening sessions, ensuring they remain open and functional while data is being transferred, and closing them when communication ends. The session layer can also set checkpoints during a data transfer—if the session is interrupted, devices can resume data transfer from the last checkpoint.

**1.2.4  
Transport Layer**  
The transport layer takes data transferred in the session layer and breaks it into “segments” on the transmitting end. It is responsible for reassembling the segments on the receiving end, turning it back into data that can be used by the session layer. The transport layer carries out flow control, sending data at a rate that matches the connection speed of the receiving device, and error control, checking if data was received incorrectly and if not, requesting it again.

**1.2.5  
Network Layer**The network layer has two main functions. One is breaking up segments into network packets, and reassembling the packets on the receiving end. The other is routing packets by discovering the best path across a physical network. The network layer uses network addresses (typically Internet Protocol addresses) to route packets to a destination node.

**1.2.6**

**Data Link Layer**  
The data link layer establishes and terminates a connection between two physically-connected nodes on a network. It breaks up packets into frames and sends them from source to destination. This layer is composed of two parts—Logical Link Control (LLC), which identifies network protocols, performs error checking and synchronizes frames, and Media Access Control (MAC) which uses MAC addresses to connect devices and define permissions to transmit and receive data.

**1.2.7  
Physical Layer**The physical layer is responsible for the physical cable or wireless connection between network nodes. It defines the connector, the electrical cable or wireless technology connecting the devices, and is responsible for transmission of the raw data, which is simply a series of 0s and 1s, while taking care of bit rate control

**1.3  
Python Code**

import random

import ipaddress

from colorama import Fore, Back, Style

def application\_layer(msg):

print(Fore.CYAN+'\n\n\n------------------APPLICATION LAYER------------------\n\n\n')

msg =' (APPLICATION) - ' + msg

print(Fore.GREEN+'\t'+msg+'\t')

#print(,"\n")

presentation\_layer(msg)

return

def presentation\_layer(msg):

# choose a protocol (telnet)

print(Fore.RED+'\n\n\n------------------PRESENTATION LAYER------------------\n\n\n')

pres\_header = ' (PRESENTATION TELNET) - '

msg = pres\_header + msg

print(Fore.GREEN+'\t'+msg+'\t')

#print("\n")

session\_layer(msg)

return

def session\_layer(msg):

# choose a protocol (rpc)

print(Fore.RED+'\n\n\n------------------SESSION LAYER----------------------\n\n\n')

sesh\_header = ' (SESSION RPC) - '

msg = sesh\_header + msg

print(Fore.GREEN+'\t'+msg+'\t')

transport\_layer(msg)

return

def transport\_layer(msg):

# assume port 20 (00010100)

print(Fore.RED+'\n\n\n------------------TRANSPORT LAYER----------------------\n\n\n')

port=(int)(input(Fore.WHITE+"\nEnter the Port Number : \n"))

p='{0:08b}'.format(port)

tr\_header = "\n(" +str(p)+ ") "+ ' TRANSPORT - '

msg = tr\_header + msg

print(Fore.GREEN+'\t'+msg+'\t')

network\_layer(msg)

return

def network\_layer(msg):

# assume IP address of 10.217.204.85

# (equivalent to 00001010 11011001 11001100 01010101)

print(Fore.RED+'\n\n\n------------------NETWORK LAYER----------------------\n\n\n')

ip=input(Fore.WHITE+"\nEnter the Ip address : \n");

net\_header = '\n(NETWORK - ' +bin(int(ipaddress.IPv4Address(ip)))+ ") -"

msg = net\_header + msg

print(Fore.GREEN+' '+msg+'\n')

datalink\_layer(msg)

return

def datalink\_layer(msg):

# convert everything to bits

print(Fore.RED+'\n\n\n------------------DATA-LINK LAYER----------------------\n\n\n')

msg\_in\_bits = ""

for c in msg:

# convert each letter to ascii

ascii\_number = ord(c)

# convert each ascii to an 8-bit word (pad with zeros)

eight\_bits = '{:08b}'.format(ascii\_number)

msg\_in\_bits += eight\_bits

msg = msg\_in\_bits

# use bit stuffing

# placing a 0 after 5 consecutive 1s is a simple find and replace

msg.replace('11111', ' 111110 ')

# add header

msg = '\n(Header - '+get\_random\_header() +')\n\nMESSGAGE - '+ msg

print(Fore.GREEN+' '+msg+'\t')

physical\_layer(msg)

return

def physical\_layer(msg):

print("\n\n")

print(Fore.RED+'\n\n\n-----------------PHYSICAL LAYER----------------------\n\n\n')

print(Fore.GREEN+"\nFinal result :\n", msg)

print(Fore.BLUE+"\nsize : ", len(msg))

print("\n\n")

return

def main(message):

application\_layer(message)

return

def get\_random\_header():

header = ""

# generate a random 32-bit header

for i in range(0, 32):

header += str(random.choice([0,1]))

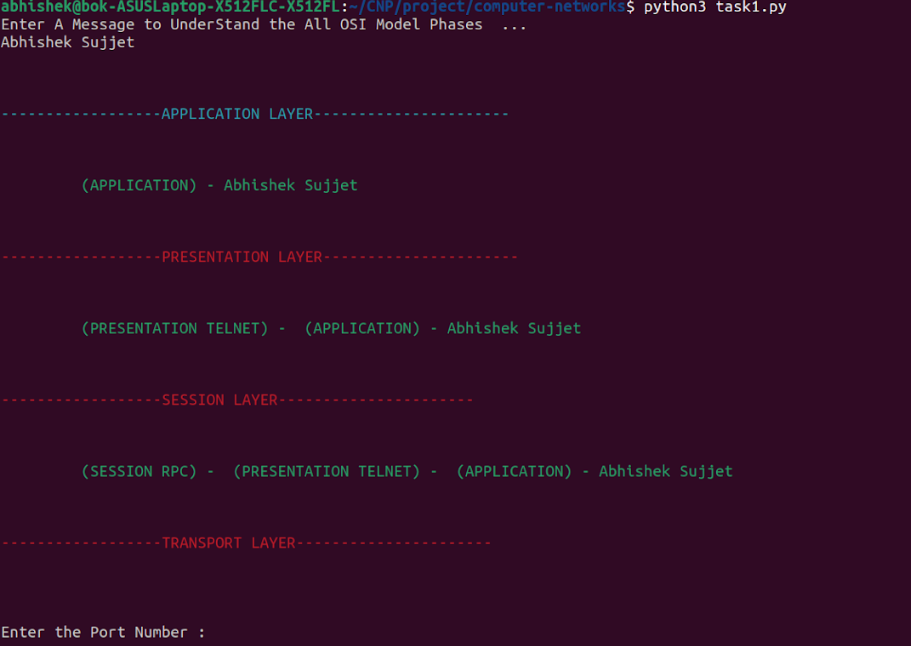
return header;

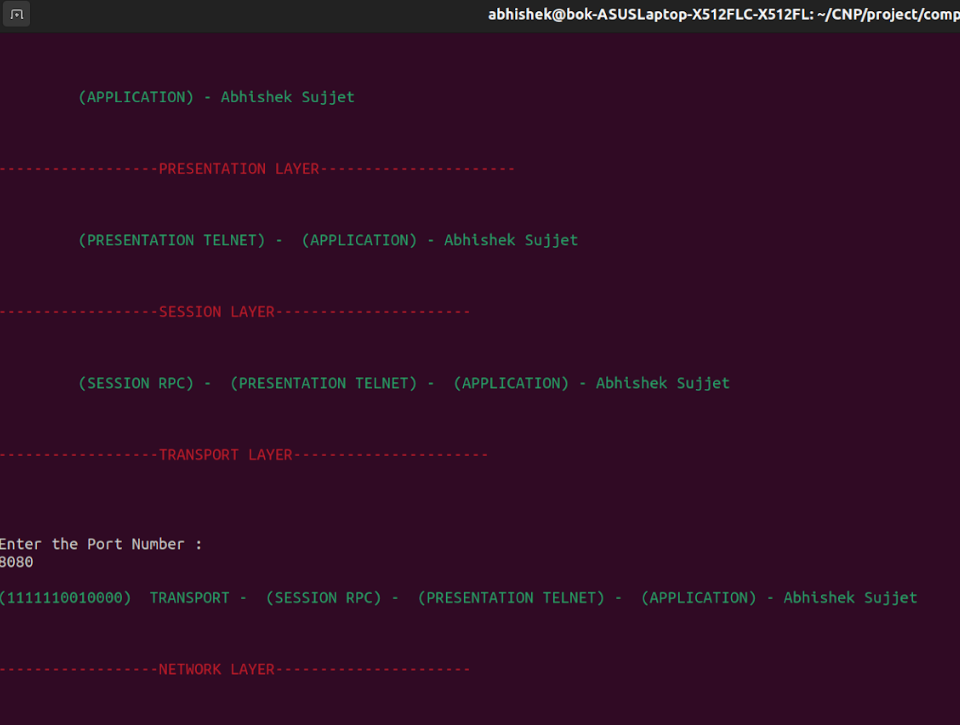
print("\n\n")

message = input(Fore.WHITE+"Enter A Message to UnderStand the All OSI Model Phases ...\n")

main(message)

**1.4  
Output**

****

****

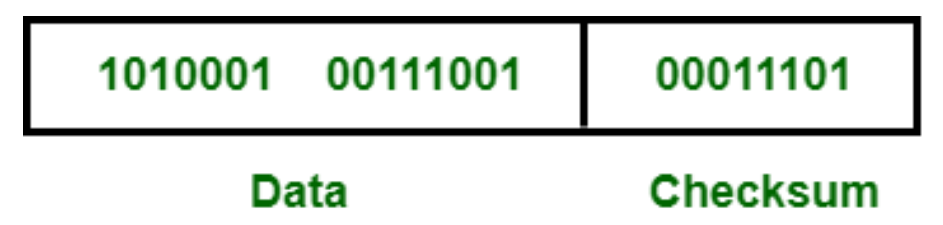
**Chapter 2  
Checksum**

**Introduction:  
Checksum** is the error detection method used by upper layer protocols and is considered to be more reliable than LRC, VRC and CRC. This method makes the use of **Checksum Generator** on Sender side and **Checksum Checker**on Receiver side.

**Steps**

1. Convert the entire data into Binary (the data is divided into equal subunits of n bit length by the checksum generator, generally 16 bits )
2. Subunits are then added
3. The one's complement is calculated

|  |  |
| --- | --- |
| **10101001** | **Subunit 1** |
| **00111001** | **Subunit 2** |
| **11100010** | **Summation of subunits** |
| **00011101** | **Checksum (Complement of Sum )** |

****

**Code:**

# Function to find the Checksum of Sent Message

def findChecksum(SentMessage, k):

# Dividing sent message in packets of k bits.

c1 = SentMessage[0:k]

c2 = SentMessage[k:2\*k]

c3 = SentMessage[2\*k:3\*k]

c4 = SentMessage[3\*k:4\*k]

# Calculating the binary sum of packets

Sum = bin(int(c1, 2)+int(c2, 2)+int(c3, 2)+int(c4, 2))[2:]

# Adding the overflow bits

if(len(Sum) > k):

x = len(Sum)-k

Sum = bin(int(Sum[0:x], 2)+int(Sum[x:], 2))[2:]

if(len(Sum) < k):

Sum = '0'\*(k-len(Sum))+Sum

# Calculating the complement of sum

Checksum = ''

for i in Sum:

if(i == '1'):

Checksum += '0'

else:

Checksum += '1'

return Checksum

# Function to find the Complement of binary addition of

# k bit packets of the Received Message + Checksum

def checkReceiverChecksum(ReceivedMessage, k, Checksum):

# Dividing sent message in packets of k bits.

c1 = ReceivedMessage[0:k]

c2 = ReceivedMessage[k:2\*k]

c3 = ReceivedMessage[2\*k:3\*k]

c4 = ReceivedMessage[3\*k:4\*k]

# Calculating the binary sum of packets + checksum

ReceiverSum = bin(int(c1, 2)+int(c2, 2)+int(Checksum, 2) +

int(c3, 2)+int(c4, 2)+int(Checksum, 2))[2:]

# Adding the overflow bits

if(len(ReceiverSum) > k):

x = len(ReceiverSum)-k

ReceiverSum = bin(int(ReceiverSum[0:x], 2)+int(ReceiverSum[x:], 2))[2:]

# Calculating the complement of sum

ReceiverChecksum = ''

for i in ReceiverSum:

if(i == '1'):

ReceiverChecksum += '0'

else:

ReceiverChecksum += '1'

return ReceiverChecksum

# Driver Code

SentMessage = input('Enter the Data in Binary: \n')

k = 4

# Calling the findChecksum() function

# Calling the checkReceiverChecksum() function

# Printing Checksum

Checksum = findChecksum(SentMessage, k)

print("SENDER SIDE CHECKSUM: ", Checksum)

ReceivedMessage = SentMessage

ReceiverChecksum = checkReceiverChecksum(ReceivedMessage, k, Checksum)

print("RECEIVER SIDE CHECKSUM: ", ReceiverChecksum)

opt=int(input('Choose Correct Checksum or Wrong :\n'))

if opt==1:

ReceiverChecksum += '1'

print('Wrong Checksum ! \n')

exit()

finalsum=bin(int(Checksum,2)+int(ReceiverChecksum,2))[2:]

# Finding the sum of checksum and received checksum

finalcomp=''

for i in finalsum:

if(i == '1'):

finalcomp += '0'

else:

finalcomp += '1'

# If sum = 0, No error is detected

if(int(finalcomp,2) == 0):

print("Receiver Checksum is equal to 0. Therefore,")

print("STATUS: ACCEPTED")

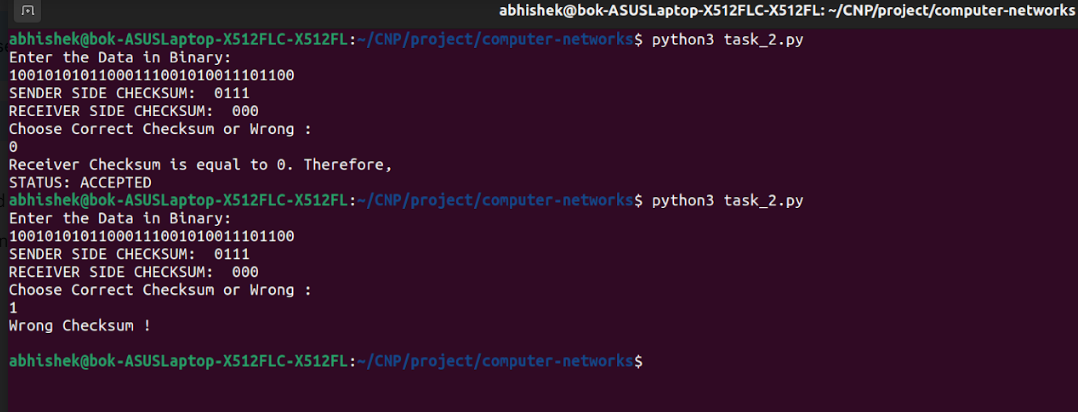
# Otherwise, Error is detected

else:

print("Receiver Checksum is not equal to 0. Therefore,")

print("STATUS: ERROR DETECTED")

**Output:**

****

**Chapter 3  
Network layer and Implementation**

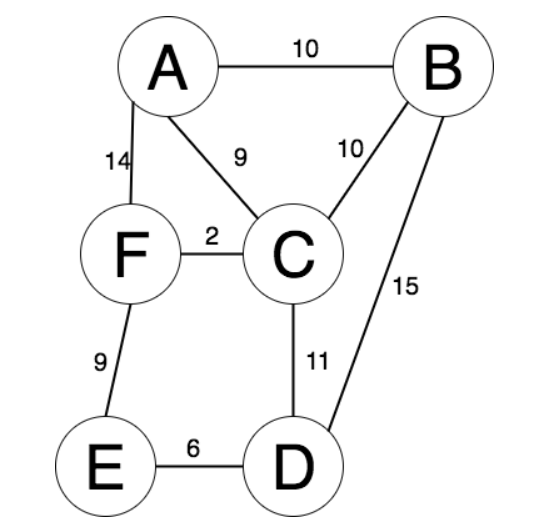
**Introduction:  
Network layer** is the third layer in the OSI model of computer networks. It’s main function is to transfer network packets from the source to the destination. It is involved both at the source host and the destination host. At the source, it accepts a packet from the transport layer, encapsulates it in a datagram and then deliver the packet to the data link layer so that it can further be sent to the receiver. At the destination, the datagram is decapsulated, the packet is extracted and delivered to the corresponding transport layer.

**Rules and Formula :**

1. Make sure there is no negative edges, set distance to source as 0 and set all other distances to infinity
2. Relax all the vertices adjacent to the current vertex
3. Choose the closest vertex as the next current vertex
4. Repeat rule 2 and 3 until the queue or reach the end of destination

**d(x, y) = d(x) + c(x, y) < d(y)**

**Graph**

****

**2.3  
Code**

class Vertex:

def \_\_init\_\_ (self, vertex):

self.name = vertex

self.adjacents = {}

def \_\_str\_\_ (self):

return str(self.name) + ' neighbour: ' + str([x.name for x in self.adjacents])

def add\_neighbour(self, neighbour, distance=0):

self.adjacents[neighbour] = distance

def get\_connections(self):

return self.adjacents.keys()

def get\_name(self):

return self.name

def get\_distance(self, neighbour):

return self.adjacents[neighbour]

class Graph:

def \_\_init\_\_ (self):

self.vert\_dict = {}

self.num\_vertices = 0

def \_\_iter\_\_ (self):

return iter(self.vert\_dict.values())

def add\_vertex(self, node):

self.num\_vertices = self.num\_vertices + 1

new\_vertex = Vertex(node)

self.vert\_dict[node] = new\_vertex

return new\_vertex

def get\_vertex(self, index):

if index in self.vert\_dict:

return self.vert\_dict[index]

else:

return None

def add\_edge(self, start, dest, distance = 0):

if start not in self.vert\_dict:

self.add\_vertex(start)

if dest not in self.vert\_dict:

self.add\_vertex(dest)

self.vert\_dict[start].add\_neighbour(self.vert\_dict[dest], distance)

self.vert\_dict[dest].add\_neighbour(self.vert\_dict[start], distance)

def get\_vertices(self):

return self.vert\_dict.keys()

def dijkstra(g, start, destination):

best\_path = {}

visited = {}

current = float('inf')

total = 0

current\_node = g.get\_vertex(start)

destination\_node = g.get\_vertex(destination)

print('\n\n(Starting Node is = %s)\n\n' %(current\_node.get\_name()))

while True:

print('\n\n(Current node is = %s)\n\n' %(current\_node))

for next\_node in current\_node.get\_connections():

#mark nodes that have been visited and put into best path

visited[current\_node.get\_name()] = current\_node.get\_name()

best\_path[current\_node.get\_name()] = current\_node.get\_name()

#check each node distance and ignore node if it is in visited array

if ((current\_node.get\_distance(next\_node) < current) and (next\_node.get\_name() not in visited) ):

current = current\_node.get\_distance(next\_node)

bestNodeSoFar = next\_node

#set new current node

total += current\_node.get\_distance(bestNodeSoFar)

current\_node = bestNodeSoFar

current = float('inf') # reset the current distance for a new node

#break out of loop if shortest path is found

if (current\_node == destination\_node):

print('\n(Final Node is = %s)' %(current\_node.get\_name()))

print('\n\nTotal Distance From %s to %s is: %s\n\n' %(start, destination, total))

best\_path[current\_node.get\_name()] = current\_node.get\_name()

break

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

g = Graph()

n=int (input('Enter the Number of Vertexs : \n'))

for i in range(0,n):

a=input('Enter the Vertex :\n')

g.add\_vertex(a)

# print('Enter the Connections and their Weigths : \n')

n1=int(input("Enter the Number of Connections : \n"))

for i in range(0,n1):

a=input('Enter the Start Vertex : \n')

b=input('Enter the End Vertex : \n')

w=int(input('Enter the Weigth : \n'))

g.add\_edge(a,b,w)

s=input('Enter the SRC Vertex : \n')

e=input('Enter the Destination Vertex : \n')

print('\n\n---------------------- Dijkstra Routing Algorithm ----------------------------\n\n\n')

dijkstra(g,s,e)

print("\n\n")

#for v in g:

# for w in v.get\_connections():

# vid = v.get\_name()

# wid = w.get\_name()

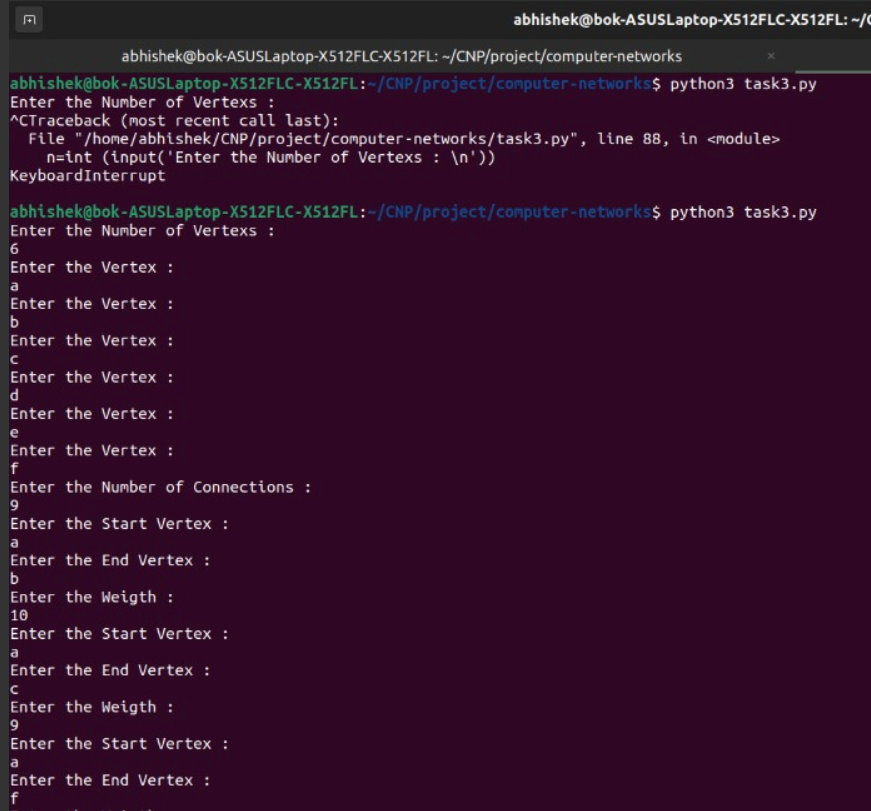
# print '( %s , %s, %3d)' % ( vid, wid, v.get\_distance(w))

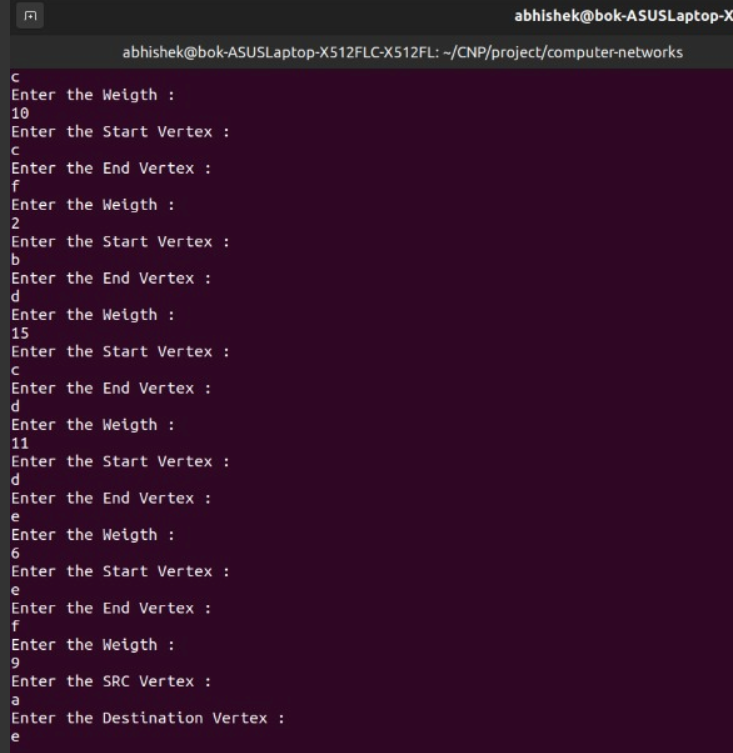
#for v in g:

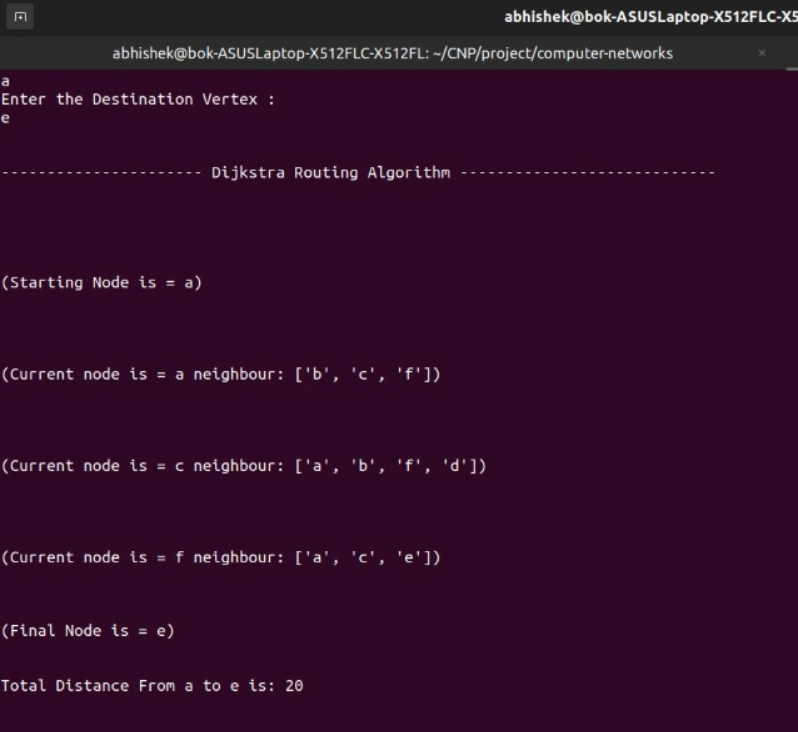
# print 'g.vert\_dict[%s]=%s' %(v.get\_name(), g.vert\_dict[v.get\_name()])

**2.4**

**Output**

****

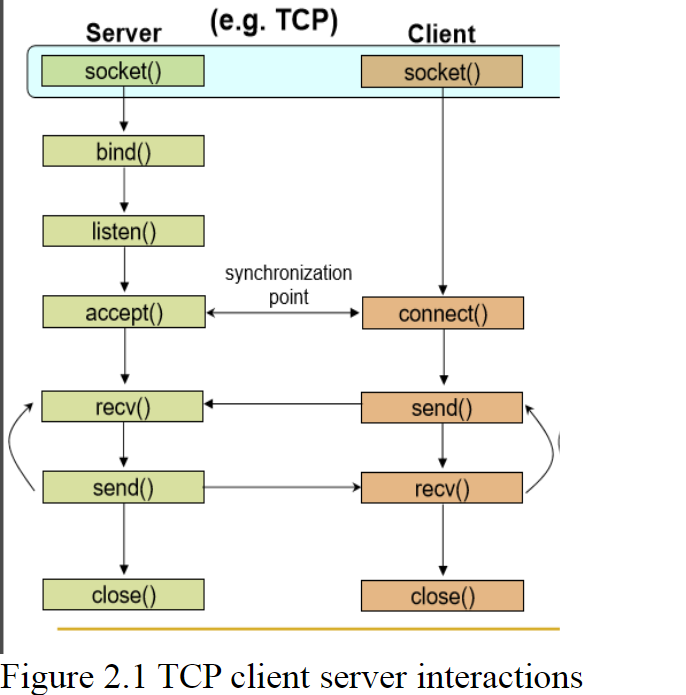
****

****

**Chapter 4:  
Transport layer**

**4.1 Introduction to Transport Layer**Transport Layer is the second layer in the TCP/IP model and the fourth layer in the OSI model. It is an **end-to-end** layer used to deliver messages to a host. It is termed an end-to-end layer because it provides a point-to-point connection **rather than** hop-to- hop, between the source host and destination host to deliver the services reliably. The unit of data encapsulation in the Transport Layer is a segment.

**4.2 TCP:**Transmission Control Protocol (TCP) is a standard that defines how to establish and maintain a network conversation by which applications can exchange data. TCP works with the Internet Protocol (IP), which defines how computers send packets of data to each other

****

* 1. **Code**

Server Code  
import socket

from random import\*

import os

from os import system

os.system('clear')

print("My IP : \n")

print('-----------------------------------------------------------------------------\n')

os.system('ifconfig | tail -11 | head -2 | tail -1')

print('\n-----------------------------------------------------------------------------\n')

print("\n\n\n")

serversocket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

while True:

l=int(input('Enter Lower Limit : \n'))

if l>5 :

break

while True:

u=int(input('Enter Upper Limit : \n'))

if u>25:

break;

r=1000\*(u-l)+l

print("PORT" + str(r))

serversocket.bind(('192.168.1.9',r))

serversocket.listen(5) # become a server socket, maximum 5 connections

connected = False

file\_sent = False

while True:

connection,address = serversocket.accept()

connected=True

buff = connection.recv(64).decode()

p='Start'

connection.send(p.encode())

buf = connection.recv(64).decode()

print("Client is Requesting for this File : "+buf)

try:

file = open('source\_folder/'+buf,'r')

file\_contents = file.read()

#print(file\_contents)

connection.send(file\_contents.encode())

file\_sent = True

pass

except:

message="file doesnt exist"

connection.send(message.encode())

reply=connection.recv(64).decode()

if reply=='yes':

f=open("source\_folder/"+buf,"w+")

message2="file created"

connection.send(message2.encode())

name=connection.recv(64).decode()

branch=connection.recv(64).decode()

sec=connection.recv(64).decode()

hostel=connection.recv(64).decode()

f.write(name)

f.write(branch)

f.write(sec)

f.write(hostel)

f.close()

f=open("source\_folder/"+buf,"r")

print(f.read())

file\_sent = True

else:

break

buf = connection.recv(64).decode()

if connected == True and file\_sent == True and buf == 'done':

print('Client Request is Fulfilled and getting Closed !\n\n')

break

print("Server is Closing !\n\n")

**Client code:**import socket

from random import\*

import os

import sys

from os import system

os.system('clear')

max=1024

while True:

l=int(input('Enter Lower Limit : \n'))

if l>5 :

break

while True:

u=int(input('Enter Upper Limit : \n'))

if u>25:

break;

r=1000\*(u-l)+l

print("PORT" + str(r))

clientsocket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

#ip=input("Enter the Ip address to Connect : \n\n")

clientsocket.connect(('192.168.1.9', r))

while True:

reg=input('\nEnter The Registration Number : \n\n')

if len(reg)==9:

break;

#reg=input('\nEnter The Registration Number : \n\n')

print('Sending '+ reg +' to Server')

clientsocket.send(reg.encode())

connected = False

filename = reg+'.txt'

# listen for ack

while True:

buf = clientsocket.recv(64).decode()

if buf == 'Start':

# successfully connected

connected = True

# send the name of the file being requested

print('Requesting '+filename+' File From SERVER !!\n\n')

clientsocket.send(filename.encode())

if connected == True:

# receive the contents of the file

buf = clientsocket.recv(max).decode()

if buf=='file doesnt exist':

opt=input("Enter YES to Create the File "+filename+" or NO to Not Create the File "+filename+": \n")

if opt=="no":

quit()

break

else:

clientsocket.send(opt.encode())

b=clientsocket.recv(64).decode()

print("\n"+b+"\n")

name=input('Enter the Name : \n')

name='Name : '+name+'\n'

clientsocket.send(name.encode())

branch=input('Enter the Branch : \n')

branch='Branch : '+branch+'\n'

clientsocket.send(branch.encode())

section=input('Enter the Section : \n')

section='Section : '+section+'\n'

clientsocket.send(section.encode())

hostel=input('Enter the Hostel : \n')

hostel='Hostel : '+hostel+'\n'

clientsocket.send(hostel.encode())

print("\nYour Details are Updated in the File !!\n")

break

exit()

else:

print ('File Received From Server')

file\_contents = buf

#print(filename)

filename=str(filename);

file = open('destination\_folder/'+filename, 'w+')

file.write(file\_contents)

print('\nFILE RECEIVED \n')

'''f = open('destination\_folder/'+filename, 'r')

print(f.read())

#os.system('cat destination\_folder/'+filename)'''

file.close()

data1='done'

clientsocket.send(data1.encode())

os.system('cat destination\_folder/'+filename)

print("\n\n")

print('File Saved in Destination Folder')

print('Sending Ack Back To Server')

print('send done to server')

break

* 1. **Output**

