CS3591 COMPUTER NETWORKS LABORATORY

MASTER RECORD

B.Tech. Artificial Intelligence and Data Science

IIyear/IV semester

R2021

Prepared by,

DR. M. Kaliappan HOD/AD

Department of Artificial Intelligence and Data Science

RAMCO INSTITUTE OF TECHNOLOGY

Rajapalayam – 626117

TamilNadu

Prepared by,

Dr. M. Kaliappan HOD/AD

Approved by,

Dr. M. Kaliappan HOD/AD

VISION AND MISSION

Vision of the Institute

To evolve as an Institute of international repute in offering high-quality technical education, Research and extension programmes in order to create knowledgeable, professionally competent and skilled Engineers and Technologists capable of working in multi-disciplinary environment to cater to the societal needs.

Mission of Institute

To accomplish its unique vision, the Institute has a far-reaching mission that aims:

• To offer higher education in Engineering and Technology with highest level of quality, Professionalism and ethical standards

• To equip the students with up-to-date knowledge in cutting-edge technologies, wisdom, creativity and passion for innovation, and life-long learning skills

• To constantly motivate and involve the students and faculty members in the education process for continuously improving their performance to achieve excellence.

Vision of Department

To impart international quality education, promote collaborative research and graduate industry-ready engineers in the domain of Artificial Intelligence and Data Science to servethe society.

Mission of the Department

- Excel in Teaching-Learning process and collaborative Research by the use of modern infrastructure and innovative components.
- Establish an Artificial Intelligence and Data Science based centre of excellence to prepare professional technocrats for solving interdisciplinary industry problems in various applications
- Motivate students to emerge as entrepreneurs with leadership qualities in a societal centric program to fulfil Industry and community needs with ethical standards.

Program Educational Outcomes (PEO's)

After successful completion of the degree, the students will be able to
PEO 1. Apply Artificial Intelligence and Data Science techniques with industrial standards and pioneering research to solve social and environment-related problems for making a sustainable ecosystems.

PEO 2. Excel with professional skills, fundamental knowledge, and advanced futuristic technologies to become Data Scientists, Data Analyst Managers, Data Science leaders AI Research Scientists, or Entrepreneurs.

Program Outcomes(PO's)

Engineering Graduates will be able to:

- **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
- **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
- **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
- **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
- **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
- **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
- **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
- **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
- **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
- **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
- **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to ones own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
- **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

- ## Program Specific Outcomes(PSO's)

  After successful completion of the degree, the students will be able to:

  PSO 1: To apply analytic technologies to arrive at actionable foresight, Insight, hindsight from data for solving business and engineering problems

  PSO 2: To create, and apply the techniques of AI and Data Science to forecast future events in the domain of Healthcare, Education, and Agriculture, Manufacturing, Automation, Robotics, Transport, etc

  PSO 3: To enrich the critical thinking skills in emerging technologies such as Hybrid Mobile application development, cloud technology stack, and cyber-physical systems with mathematical aid to foresee the research findings and provide the solutions

INSTRUCTIONS TO THE STUDENTS

- Students should wear Uniforms and Coats neatly during the lab session
- Students should maintain silence during lab hours; roaming around the lab during lab session is not permitted
- Programs should be written in the manual and well prepared for the current exercise before coming to the session
- Experiments should be completed within the Specified Lab Session
- Before Every Session, Last Session lab exercise & record should be completed and get it verified by the faculty
- In the Record Note, Flow Chart and Outputs should be written on the left side, while Aim, Algorithm & Result should be written on the right side.
- Programs (Printed) should be placed on the right side
- Marks for each lab exercise is awarded as follows :

Performance	25 Marks
Viva	10 Marks
Record	15 Marks
Total	**50 Marks**

PREFACE

The current year's manual (2022 -2023) differs from the previous year's manual in numerous ways. Course objectives and outcomes and mapping the lab exercises to the outcomes are included in this manual.

All the lab exercises are revised and updated in many places. New exercises are added besides university syllabus.

A number of people helped me with this revision. I would like to thank the Head of the Department and all friendly faculty members for providing valuable suggestions in preparing this students' centric manual. I would also like to the Lab technicians and other people who helped me in many ways.

CONTENTS

SYLLABUS

CS3591 COMPUTER NETWORKS L T P C

3 0 2 4

1. Learn to use commands like tcpdump, netstat, ifconfig, nslookup and traceroute. Capture ping and trace route PDUs using a network protocol analyzer and examine.
2. Write a HTTP web client program to download a web page using TCP sockets.
3. Applications using TCP sockets like: a) Echo client and echo server b) Chat
4. Simulation of DNS using UDP sockets.
5. Use a tool like Wireshark to capture packets and examine the packets
6. Write a code simulating ARP /RARP protocols.
7. Study of Network simulator (NS) and Simulation of Congestion Control Algorithms using NS.
8. Study of TCP/UDP performance using Simulation tool.
9. Simulation of Distance Vector/ Link State Routing algorithm.
10. Simulation of an error correction code (like CRC)

COURSE OUTCOMES:

At the end of this course, the students will be able to:

CO 1: Explain the basic layers and its functions in computer networks.

CO 2: Understand the basics of how data flows from one node to another.

CO 3: Analyze routing algorithms.

CO 4: Describe protocols for various functions in the network.

CO 5: Analyze the working of various application layer protocols.

TOTAL:75 PERIODS

CO Mapping

Ex no	Experiment	CO
1	Learn to use commands like tcpdump, netstat, ifconfig, nslookup and traceroute. Capture ping and trace route PDUs using a network protocol analyzer and examine.	CO1
2	Write a HTTP web client program to download a web page using TCP sockets	CO2
3	Applications using TCP sockets	CO2
4	Simulation of DNS using UDP sockets	CO2
5	Use a tool like Wireshark to capture packets and examine the packets	CO2
6	Write a code simulating ARP /RARP protocols.	CO4
7	Study of Network simulator (NS) and Simulation of Congestion Control Algorithms using NS	CO4
8	Study of TCP/UDP performance using Simulation tool.	CO4
9	Simulation of Distance Vector/ Link State Routing algorithm.	CO3
10	Simulation of an error correction code (like CRC)	CO5
11	Additional Exercises	CO5

Exercise 1

Learn to use commands like tcpdump, netstat, ifconfig, nslookup and traceroute. Capture ping and trace route PDUs using a network protocol analyzer and examine.

Ipconfig

```
C:\Windows\System32>ipconfig

Windows IP Configuration


Ethernet adapter Ethernet:

   Connection-specific DNS Suffix  . : ritdc.com
   Link-local IPv6 Address . . . . . : fe80::87f:4dc6:ad00:8843%10
   IPv4 Address. . . . . . . . . . . : 172.16.71.50
   Subnet Mask . . . . . . . . . . . : 255.255.240.0
   Default Gateway . . . . . . . . . : 172.16.64.1

Wireless LAN adapter Wi-Fi:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . : RITDC.COM

Wireless LAN adapter Local Area Connection* 9:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Wireless LAN adapter Local Area Connection* 10:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Ethernet adapter Bluetooth Network Connection:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :
```

Tracert

```
C:\Windows\System32>tracert 172.16.71.50

Tracing route to RIT-AIDS-50.ritdc.com [172.16.71.50]
over a maximum of 30 hops:

  1    <1 ms    <1 ms    <1 ms  RIT-AIDS-50.ritdc.com [172.16.71.50]

Trace complete.
```

Nslookup

```
C:\Windows\System32>nslookup www.google.com
Server:  ritdc1.ritdc.com
Address:  172.16.0.80

Non-authoritative answer:
Name:    www.google.com
Addresses:  2404:6800:4007:82d::2004
          142.250.193.164
```

Netstat

```
C:\Windows\System32>netstat -a

Active Connections

  Proto  Local Address          Foreign Address        State
  TCP    0.0.0.0:135            RIT-AIDS-50:0          LISTENING
  TCP    0.0.0.0:445            RIT-AIDS-50:0          LISTENING
  TCP    0.0.0.0:3389           RIT-AIDS-50:0          LISTENING
  TCP    0.0.0.0:5040           RIT-AIDS-50:0          LISTENING
  TCP    0.0.0.0:5432           RIT-AIDS-50:0          LISTENING
  TCP    0.0.0.0:7680           RIT-AIDS-50:0          LISTENING
  TCP    0.0.0.0:49664          RIT-AIDS-50:0          LISTENING
  TCP    0.0.0.0:49665          RIT-AIDS-50:0          LISTENING
  TCP    0.0.0.0:49666          RIT-AIDS-50:0          LISTENING
  TCP    0.0.0.0:49667          RIT-AIDS-50:0          LISTENING
  TCP    0.0.0.0:49668          RIT-AIDS-50:0          LISTENING
  TCP    0.0.0.0:49669          RIT-AIDS-50:0          LISTENING
  TCP    0.0.0.0:49670          RIT-AIDS-50:0          LISTENING
  TCP    0.0.0.0:49672          RIT-AIDS-50:0          LISTENING
  TCP    0.0.0.0:49673          RIT-AIDS-50:0          LISTENING
  TCP    127.0.0.1:8888         RIT-AIDS-50:0          LISTENING
  TCP    127.0.0.1:27017        RIT-AIDS-50:0          LISTENING
  TCP    127.0.0.1:49695        RIT-AIDS-50:49696      ESTABLISHED
  TCP    127.0.0.1:49696        RIT-AIDS-50:49695      ESTABLISHED
  TCP    127.0.0.1:51846        RIT-AIDS-50:51847      ESTABLISHED
  TCP    127.0.0.1:51847        RIT-AIDS-50:51846      ESTABLISHED
  TCP    127.0.0.1:51853        RIT-AIDS-50:0          LISTENING
  TCP    127.0.0.1:51853        RIT-AIDS-50:51863      ESTABLISHED
  TCP    127.0.0.1:51853        RIT-AIDS-50:51872      ESTABLISHED
  TCP    127.0.0.1:51854        RIT-AIDS-50:0          LISTENING
  TCP    127.0.0.1:51854        RIT-AIDS-50:51859      ESTABLISHED
  TCP    127.0.0.1:51854        RIT-AIDS-50:51871      ESTABLISHED
  TCP    127.0.0.1:51855        RIT-AIDS-50:0          LISTENING
  TCP    127.0.0.1:51855        RIT-AIDS-50:51874      ESTABLISHED
  TCP    127.0.0.1:51856        RIT-AIDS-50:0          LISTENING
  TCP    127.0.0.1:51857        RIT-AIDS-50:0          LISTENING
  TCP    127.0.0.1:51857        RIT-AIDS-50:51858      ESTABLISHED
  TCP    127.0.0.1:51857        RIT-AIDS-50:51873      ESTABLISHED
  TCP    127.0.0.1:51858        RIT-AIDS-50:51857      ESTABLISHED
  TCP    127.0.0.1:51859        RIT-AIDS-50:51854      ESTABLISHED
  TCP    127.0.0.1:51863        RIT-AIDS-50:51853      ESTABLISHED
  TCP    127.0.0.1:51864        RIT-AIDS-50:51865      ESTABLISHED
  TCP    127.0.0.1:51865        RIT-AIDS-50:51864      ESTABLISHED
  TCP    127.0.0.1:51866        RIT-AIDS-50:51867      ESTABLISHED
  TCP    127.0.0.1:51867        RIT-AIDS-50:51866      ESTABLISHED
  TCP    127.0.0.1:51868        RIT-AIDS-50:0          LISTENING
  TCP    127.0.0.1:51869        RIT-AIDS-50:51870      ESTABLISHED
  TCP    127.0.0.1:51870        RIT-AIDS-50:51869      ESTABLISHED
  TCP    127.0.0.1:51871        RIT-AIDS-50:51854      ESTABLISHED
  TCP    127.0.0.1:51872        RIT-AIDS-50:51853      ESTABLISHED
  TCP    127.0.0.1:51873        RIT-AIDS-50:51857      ESTABLISHED
  TCP    127.0.0.1:51874        RIT-AIDS-50:51855      ESTABLISHED
  TCP    172.16.71.50:139       RIT-AIDS-50:0          LISTENING
  TCP    172.16.71.50:7680      RIT-CCLAB2-26:49955    TIME_WAIT
  TCP    172.16.71.50:7680      RIT-CSE-FAC-HAL:50238  ESTABLISHED
  TCP    172.16.71.50:7680      RIT-CCLAB1-:53042      TIME_WAIT
  TCP    172.16.71.50:49859     ritmem02:microsoft-ds  ESTABLISHED
```

Ping

```
C:\Users\NITHYALAKSHMI>ping google.com

Pinging google.com [2404:6800:4007:809::200e] with 32 bytes of data:
Reply from 2404:6800:4007:809::200e: time=46ms
Reply from 2404:6800:4007:809::200e: time=78ms
Reply from 2404:6800:4007:809::200e: time=38ms
Reply from 2404:6800:4007:809::200e: time=47ms

Ping statistics for 2404:6800:4007:809::200e:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 38ms, Maximum = 78ms, Average = 52ms
```

```
import matplotlib.pyplot as plt
 import pandas as pd
 import seaborn as sns
```

Loading the DataSet exported from WireShark

df.shape

(5902, 9)

Getting head of the DataSet

df.head()

	No.	Time	Source	Destination	Protocol	Length
0	1	0.000000	Dell_6f:aa:33	Broadcast	ARP	60
1	2	0.010190	172.16.66.79	224.0.0.251	MDNS	132
2	3	0.040005	Dell_6f:a1:7e	Broadcast	ARP	60
3	4	0.058380	Dell_6e:88:96	Broadcast	ARP	60
4	5	0.061512	Dell_6f:ed:17	Broadcast	ARP	60

Getting basic info about the DataSet

df.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5902 entries, 0 to 5901 Data
columns (total 7 columns):
 #   Column       Non-Null Count   Dtype
---  -------      --------------   -----
 0   No.          5902 non-null    int64
 1   Time         5902 non-null    float64
 2   Source       5902 non-null    object
 3   Destination  5902 non-null    object
 4   Protocol     5902 non-null    object
 5   Length       5902 non-null    int64
 6   Info         5902 non-null    object
dtypes: float64(1), int64(2), object(4)
memory usage: 322.9+ KB
```

Getting basic Description about the DataSet

df.describe()

[6]:		No.	Time	Length
	count	5902.000000	5902.000000	5902.000000
	mean	2951.500000	18.368030	117.246018
	std	1703.904976	11.057993	513.977122
	min	1.000000	0.000000	54.000000
	25%	1476.250000	8.428025	60.000000
	50%	2951.500000	18.577295	60.000000
	75%	4426.750000	27.482398	60.000000
	max	5902.000000	38.369039	16460.000000

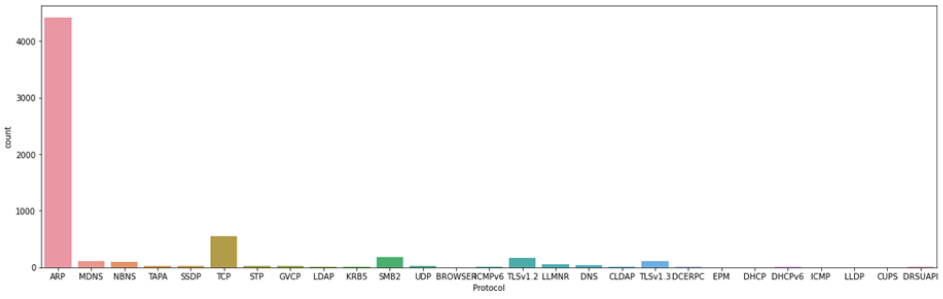
Getting number of unique values in each column in the DataSet

df.nunique()

No.	5902
Time	3486
Source	154
Destination	64
Protocol	27

Getting CountPlot for most Protocol followed by the Computer

```
plt.figure(figsize=(20,6))
sns.countplot(x=df['Protocol'])
```



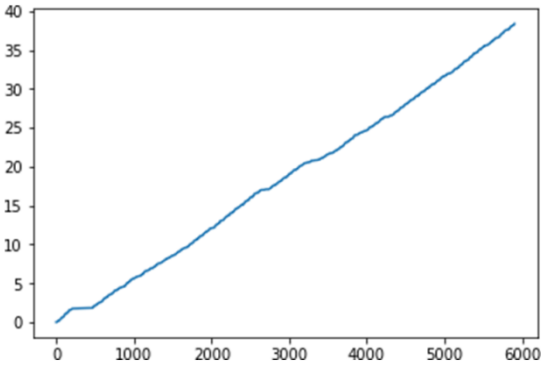
Observation

- We can observe that most of the connections are following the ARP(Address Resolution Protocol).
- The Second most followed protocol was TCP(Transmissin Control Protocol).

```
df['Protocol'].unique()
```

```
array(['ARP', 'MDNS', 'NBNS', 'TAPA', 'SSDP', 'TCP', 'STP', 'GVCP',
       'LDAP', 'KRB5', 'SMB2', 'UDP', 'BROWSER', 'ICMPv6', 'TLSv1.2',
       'LLMNR', 'DNS', 'CLDAP', 'TLSv1.3', 'DCERPC', 'EPM', 'DHCP',
       'DHCPv6', 'ICMP', 'LLDP', 'CUPS', 'DRSUAPI'], dtype=object)
```

```
plt.plot(df['Time'])
```



Getting the Source IP's followed during capture

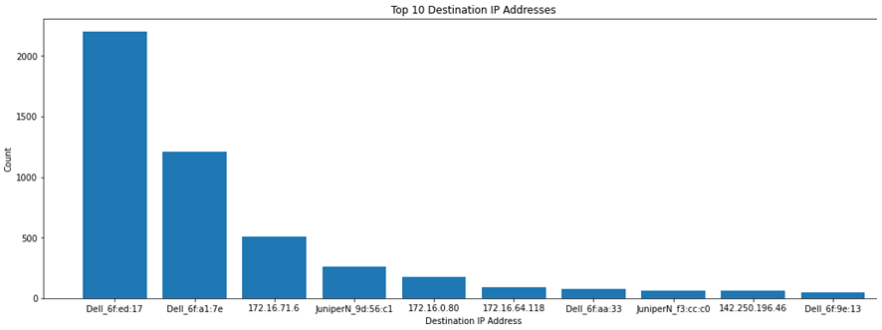
df['Source'].unique()

```
array(['Dell_6f:aa:33', '172.16.66.79', 'Dell_6f:a1:7e', 'Dell_6e:88:96',
       'Dell_6f:ed:17', '172.16.66.151', 'JuniperN_9d:56:c1',
       'Dell_01:4e:05', 'Dell_4b:39:a0', 'Dell_6f:a0:32', 'Dell_6f:aa:b6',
       '172.16.65.95', 'Dell_ed:37:0f', 'Dell_6f:ea:53', '172.16.67.130',
       'Dell_6f:a9:23', '172.16.64.164', 'Dell_6f:ec:5a', '172.16.71.30',
       'Dell_01:4c:5e', 'Dell_6f:a4:72', '172.16.71.6',
       'JuniperN_55:c9:61', '172.16.66.138', 'Dell_6f:aa:32',
       '172.16.0.80', '172.16.65.110', 'Dell_4a:95:81', '172.16.66.82',
       '172.16.66.159', '142.250.183.227', '142.250.195.45',
       '117.18.237.29', 'Dell_c3:2b:e9', '142.250.182.74',
       '172.16.66.164', 'Dell_6f:9f:cc', 'Dell_01:4a:8c', '172.16.67.179',
       'Dell_27:46:63', '142.250.196.74', 'Dell_27:50:08', '::',
       'fe80::fc85:4089:42db:58e0', 'Dell_6f:a3:0c', '172.16.68.231',
       '173.223.217.220', '23.6.213.139', 'Dell_01:4b:70',
       '142.250.196.46', '104.18.6.185', '172.16.71.23', '142.250.196.14',
       'Dell_01:4e:46', 'Dell_6f:9f:44', '172.16.65.90', 'Dell_4a:47:a4',
       '35.188.42.15', '142.250.195.99', 'Dell_6f:aa:65', 'Dell_f8:4e:72',
       '142.250.182.67', '20.198.118.190', '142.250.195.238',
       '131.253.33.203', 'Dell_6f:eb:3d', '172.16.0.82', '172.16.66.161',
       'Dell_01:4c:af', 'fe80::5536:5574:cf5b:bcb9', '172.217.163.206',
       '142.250.182.2', '117.18.232.200', 'Dell_6f:a9:f4',
```

import pandas as pd

import matplotlib.pyplot as plt

 # Get the top 10 destination IP addresses
top_ips =df['Source'].value_counts().nlargest(10)
 # Create a bar chart of the top 10 destinationIPs
plt.figure(figsize=(18,6)) plt.bar(top_ips.index, top_ips.values)
plt.title('Top 10 Destination IP Addresses') plt.xlabel('Destination IP Address') plt.ylabel('Count')
plt.show()



Getting the Destination IP's followed during capture

df['Destination'].unique()

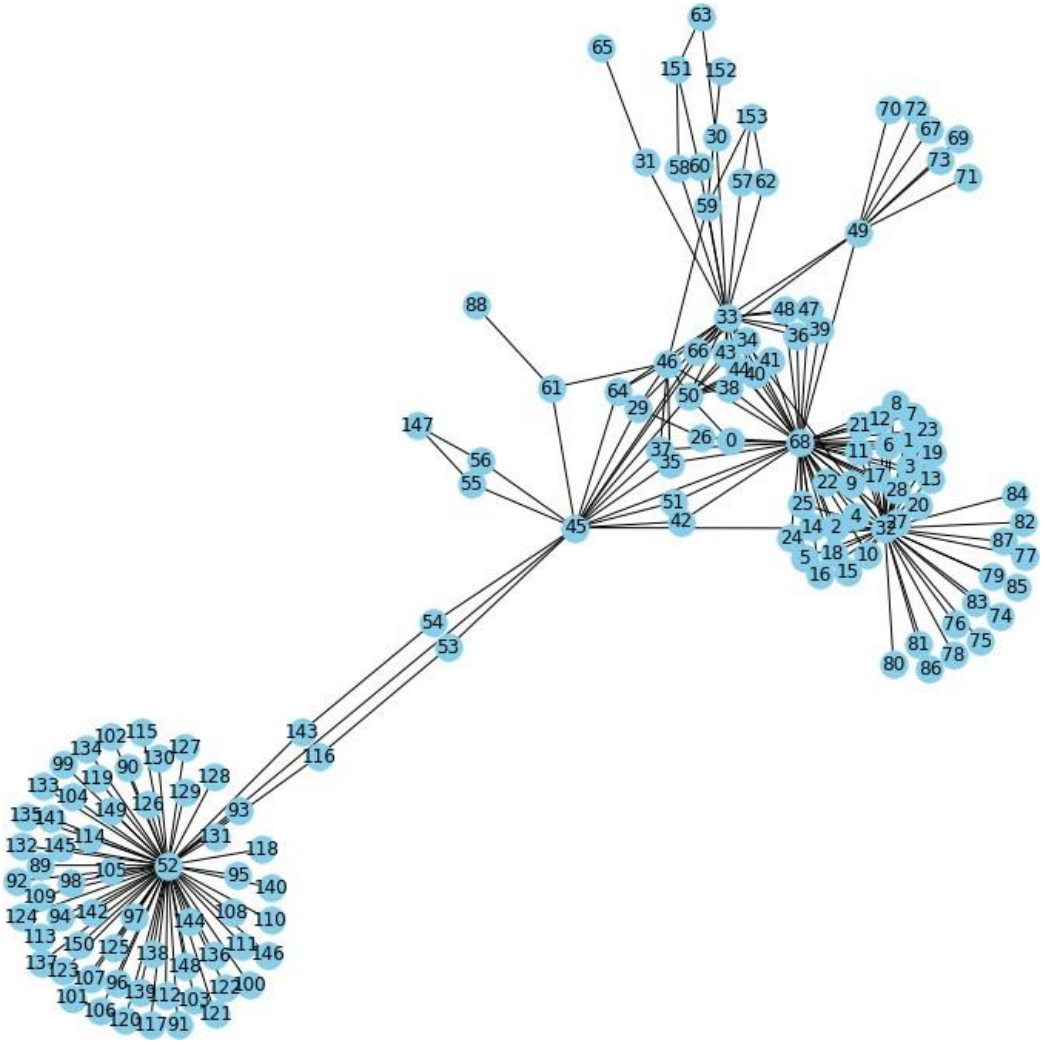
```
array(['Broadcast', '224.0.0.251', '172.16.79.255', '239.255.255.250',
       '172.16.0.81', 'Spanning-tree-(for-bridges)_00', '255.255.255.25
       '172.16.0.80', '172.16.71.6', '142.250.183.227', '142.250.195.45
       '117.18.237.29', '142.250.182.74', '142.250.196.74',
       'ff02::1:ffdb:58e0', 'ff02::2', '224.0.0.252', '173.223.217.220'
       '23.6.213.139', '142.250.196.46', 'ff02::1', '104.18.6.185',
       '142.250.196.14', '35.188.42.15', '142.250.195.99',
       '142.250.182.67', '20.198.118.190', '172.16.0.83',
       '142.250.195.238', '131.253.33.203', '172.16.0.82', 'ff02::fb',
       'ff02::1:3', '172.217.163.206', '142.250.182.2', '117.18.232.200
       '23.207.152.32', '173.222.71.214', '142.250.195.171',
       '142.250.195.129', '172.217.166.106', '216.58.200.129',
       '142.250.182.42', '172.253.118.188', '172.217.163.163',
       '142.250.182.106', '142.250.71.42', '216.58.196.174',
       '172.16.66.127', '142.250.182.14', 'Dell_6f:ed:cd', 'ff02::1:2',
       '142.250.182.36', 'ff02::16', '142.250.182.5', '142.250.195.227'
       '142.250.77.99', '216.239.32.116', '142.250.195.229',
       '172.217.166.110', '172.16.3.255', '172.16.65.18',
       'LLDP_Multicast', 'Dell_00:74:67'], dtype=object)
```

from sklearn.preprocessing import LabelEncoder
 # create a LabelEncoder object
le = LabelEncoder()
 # fit the encoder to your column and transform the values
df['SourceEncoded'] = le.fit_transform(df['Source'])
 # fit the encoder to your column and transform the values
df['DestEncoded'] = le.fit_transform(df['Destination'])
import networkx as nx
import matplotlib.pyplot as plt

 # Create an empty graph
G = nx.Graph()

 # Add edges to the graph based on the SourceEncoded and DestEncodedcolumns
edges = zip(df['SourceEncoded'],df['DestEncoded'])
G.add_edges_from(edges)

```
pos = nx.spring_layout(G)
import matplotlib.pyplot as plt


plt.figure(figsize=(10,10))
nx.draw(G, with_labels=True, node_color='skyblue', edge_color='black',width=1,
font_color='black')
plt.show()
```



Exercise 2

Write a HTTP web client program to download a web page using TCP sockets.

Ex No. 2 **Date:**

Aim:

a. To write a HTTP web client program to download a web page using TCP sockets

Program:

```
import socket
import sys
host = 'www.gmail.com'
port = 80 # web
print('# Creating socket')
try:
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
except socket.error:
    print('Failed to create socket')
    sys.exit()
print('# Getting remote IP address')
try:
    remote_ip = socket.gethostbyname(host)
except socket.gaierror:
    print('Hostname could not be resolved. Exiting')
    sys.exit()
print('# Connecting to server, ' + host + ' (' + remote_ip + ')')
s.connect((remote_ip, port))

# Send data to remote server
print('# Sending data to server')
request = "GET / HTTP/1.0\r\n\r\n"
try:
    s.sendall(request.encode())
except socket.error:
    print('Send failed')
    sys.exit()
print('# Receive data from server')
reply = s.recv(4096)
print(reply.decode())
```

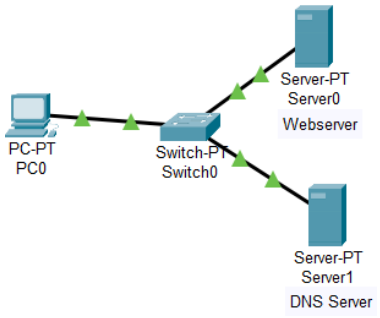
Output:

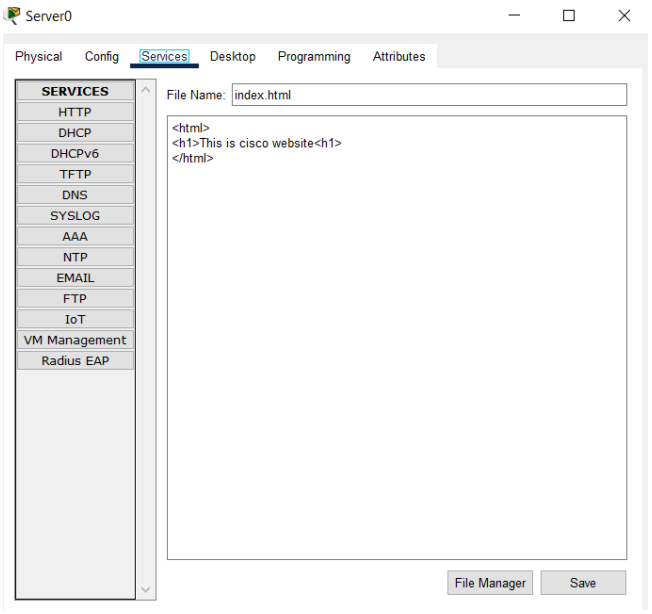
```
# Creating socket
# Getting remote IP address
# Connecting to server, www.gmail.com (173.194.196.18)
# Sending data to server
# Receive data from server
HTTP/1.0 200 OK
Date: Sun, 26 Feb 2023 09:16:56 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO-8859-1
P3P: CP="This is not a P3P policy! See g.co/p3phelp for more info."
Server: gws
X-XSS-Protection: 0
X-Frame-Options: SAMEORIGIN
Set-Cookie: 1P_JAR=2023-02-26-09; expires=Tue, 28-Mar-2023 09:16:56 GMT; path=/; domain=.google.com; Secur
Set-Cookie: AEC=ARSKqsLmg0g3g1MrCaQm1NjlFg7TPn7yxJvlp_zUi_bvCM6iegzTOOWpHz0; expires=Fri, 25-Aug-2023 09:1
Set-Cookie: NID=511=aldUDtM2U1WyV_g5YztDmrcCttn6jG6Dd0FPvNibpoJtf80X5n_mNSG5GOJF71TAyVRhTUMpKhsEIscnYmH5b9
Accept-Ranges: none
Vary: Accept-Encoding

<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang="en"><head><meta content="Sear
var f=this||self;var h,k=[];function l(a){for(var b;a&&(!a.getAttribute||!(b=a.getAttribute("eid")));)a=a.
function n(a,b,c,d,g){var e="";c||-1!==b.search("&ei=")||(e="&ei="+l(d),-1===b.search("&lei=")&&(d=m(d))&&
```

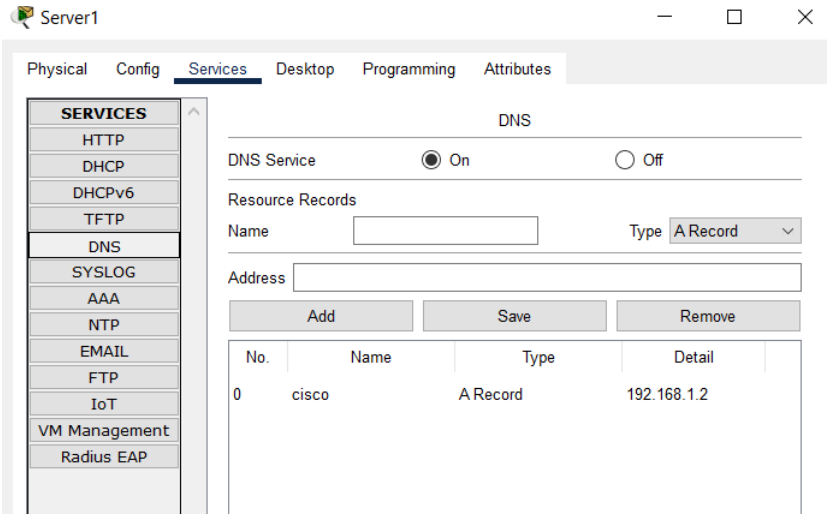
b. Aim:

To download and execute webpage from webserver using Cisco Packet Tracer

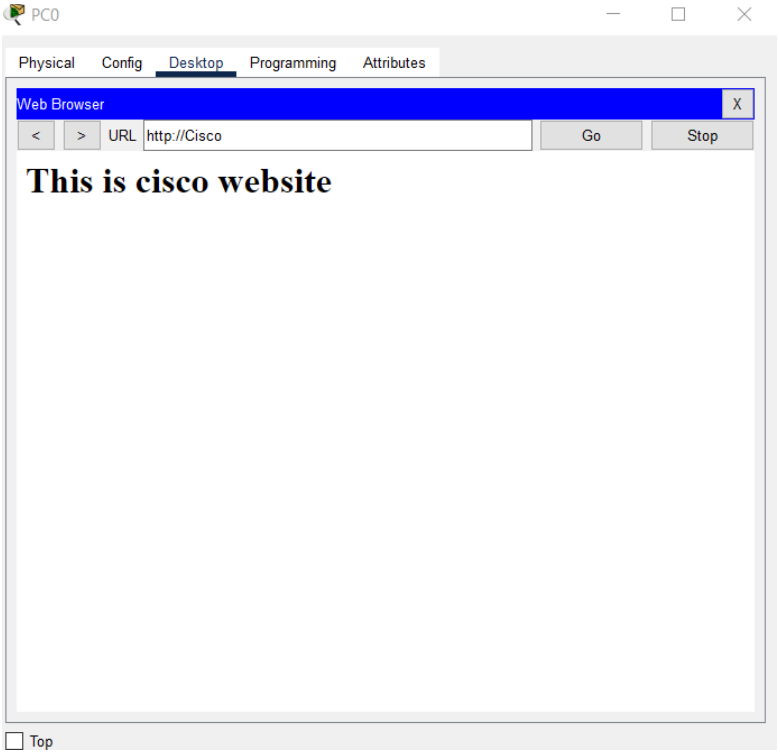
Procedure:







Output:



Result:

Thus the program

a. To write a HTTP web client program to download a web page using TCP sockets

b. To download and execute webpage from webserver using Cisco Packet Tracer was Written and executed successfully.

Exercise 3
Applications on TCP sockets

Ex No. 3 **Date:**

Aim:

To write python program for applications on TCP sockets

3a Echo server

Program:

```
import socket
import sys
import argparse
host = 'localhost'
data_payload = 2048
backlog = 5
def echo_server(port):
    """ A simple echo server """
    # Create a TCP socket
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    # Enable reuse address/port
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    # Bind the socket to the port
    server_address = (host, port)
    print("Starting up echo server on %s port %s" % server_address)
    sock.bind(server_address)
    # Listen to clients, backlog argument specifies the max no. of queued connections
    sock.listen(backlog)
    while True:
        print("Waiting to receive message from client")
        client, address = sock.accept()
        data = client.recv(data_payload)
        if data:
            print("Data: %s" % data.decode('utf-8'))
            client.send(data)
            print("sent %s bytes back to %s" % (len(data), address))
            # end connection
            client.close()
parser = argparse.ArgumentParser(description='Socket Server Example')
parser.add_argument('--port',action="store", dest="port", type=int, required=True)
given_args = parser.parse_args()
port = given_args.port
echo_server(port)
```

Output:

```
C:\Users\NITHYALAKSHMI\Desktop\exercise folder\CN>python echo_server.py --port 8080
Starting up echo server on localhost port 8080
Waiting to receive message from client
Data: This is a test message
sent 22 bytes back to ('127.0.0.1', 50163)
Waiting to receive message from client
```

3a Echo client

Program

```
import socket
import sys
import argparse
host = 'localhost'
def echo_client(port):
    """ A simple echo client """
    # Create a TCP/IP socket
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    # Connect the socket to the server
    server_address = (host, port)
    print ("Connecting to %s port %s" % server_address)
    sock.connect(server_address)
    try:
      # Send data
      message = "Test message. This will be echoed"
      print ("Sending %s" % message)
      sock.sendall(message.encode('utf-8'))
      # Look for the response
      amount_received = 0
      amount_expected = len(message)
      while amount_received < amount_expected:
          data = sock.recv(16)
          amount_received += len(data)
          print ("Received: %s" % data)
    except socket.error as e:
       print ("Socket error: %s" %str(e))
    except Exception as e:
       print ("Other exception: %s" %str(e))
    finally:
       print ("Closing connection to the server")
       sock.close()
parser = argparse.ArgumentParser(description='Socket Server Example')
parser.add_argument('--port', action="store",
dest="port", type=int, required=True)
given_args = parser.parse_args()
port = given_args.port
echo_client(port)
```

Output:

```
C:\Users\NITHYALAKSHMI\Desktop\exercise folder\CN>python echo_client.py --port 8080
Connecting to localhost port 8080
Sending This is a test message
Received: b'This is a test m'
Received: b'essage'
Closing connection to the server
```

3b Chat server

Program:

```
import time, socket, sys
print("\nWelcome to Chat Room\n")
```

```
print("Initialising. ..\n")
time.sleep(1)
s = socket.socket()
host = socket.gethostname()
ip = socket.gethostbyname(host)
port = 1234
        ind((host, port))
        print(host, "(", ip,
        ")\n")
name = input(str("Enter your name: "))
s.listen(1)
print("\nWaiting for incoming connections.  \n")
conn, addr = s.accept()
print("Received connection from ", addr[0], "(", addr[1], ")\n")
s_name = conn.recv(1024)
s_name = s_name.decode()
print(s_name, "has connected to the chat room\nEnter [e] to exit chat room\n")
conn.send(name.encode())
while True:
    message = input(str("Me : "))
    if message == "[e]":
        message = "Left chat room!"
        conn.send(message.encode())
        print("\n")
        break
    conn.send(message.encode())
    message = conn.recv(1024)
    message = message.decode()
    print(s_name, ":", message)
```

Output:

```
C:\Users\NITHYALAKSHMI\Desktop\exercise folder\CN>python chat_server.py --port 8080

Welcome to Chat Room

Initialising....

DESKTOP-8LVM2KQ ( 192.168.208.196 )

Enter your name: Nithya

Waiting for incoming connections...

Received connection from  192.168.208.196 ( 50569 )

Nithya has connected to the chat room
Enter [e] to exit chat room

Me : hi
Nithya : hi
Me : This is a chat created using tcp sockets
Nithya : That is great to hear
Me :
```

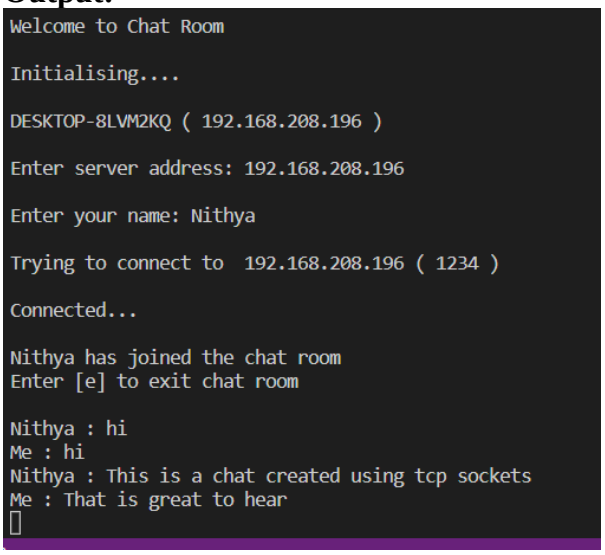
3b Chat client

Program:

```
import time, socket, sys
print("\nWelcome to Chat Room\n")
print("Initialising. ..\n")
time.sleep(1)
s = socket.socket
```

```
ip = socket.gethostbyname(shost) print(shost, "(", ip, ")\n")
host = input(str("Enter server address: "))
name = input(str("\nEnter your name: "))
port = 1234
print("\nTrying to connect to ", host, "(", port, ")\n")
time.sleep(1)
        onnect((host, port))
        print("Connected...\n"
        )
        s.send(name.encode()
        ) s_name =
        s.recv(1024) s_name
        = s_name.decode()
print(s_name, "has joined the chat room\nEnter [e] to exit chat room\n")
while True:
    message = s.recv(1024)
    message = message.decode()
    print(s_name, ":", message)
    message = input(str("Me : "))
    if message == "[e]":
        message = "Left chat room!"
        s.send(message.encode())
        print("\n")
        break
    s.send(message.encode())
```

Output:



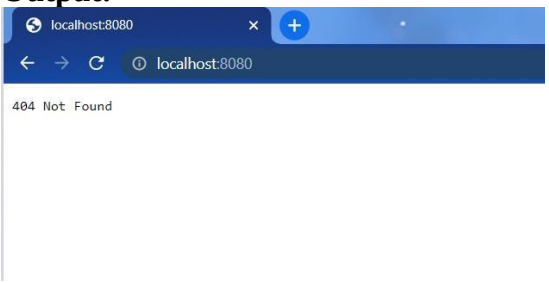
3c Webserver

Program:

```
import socket
HOST, PORT = '', 8080 # set the server's host and port
DOCUMENT_ROOT = './www' # set the document root directory
def handle_request(client_connection):
    request = client_connection.recv(1024)
    request_lines = request.split(b'\r\n')
    try:
        method, path, version = request_lines[0].split()
        if method != b'GET':
```

```
            raise Exception('Unsupported method: %s' % method)
        if b'..' in path:
            raise Exception('Directory traversal is not allowed')
        filepath = DOCUMENT_ROOT + path.decode('utf-8')
        with open(filepath, 'rb') as f:
            content = f.read()
        response = b'HTTP/1.1 200 OK\r\n'
        response += b'Content-Type: text/html\r\n'
        response += b'Content-Length: %d\r\n' % len(content)
        response += b'\r\n'
        response += content
    except Exception as e:
        print(str(e))
        response = b'HTTP/1.1 404 Not Found\r\n'
        response += b'Content-Type: text/plain\r\n'
        response += b'Content-Length: 13\r\n'
        response += b'\r\n'
        response += b'404 Not Found'
    client_connection.sendall(response)
    client_connection.close()
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as listen_socket:
    listen_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    listen_socket.bind((HOST, PORT))
    listen_socket.listen(1)
    print('Serving HTTP on %s:%d ...' % (HOST, PORT))
    while True:
        client_connection, client_address = listen_socket.accept()
        handle_request(client_connection)
```

Output:



3d. Udp pinger server program:

Program:

```
import socket
SERVER_ADDRESS = ('localhost', 12000)
BUFFER_SIZE = 1024
server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
server_socket.bind(SERVER_ADDRESS)
print(f'Server listening on {SERVER_ADDRESS}')

while True:
    data, client_address = server_socket.recvfrom(BUFFER_SIZE)
    print(f'Received ping from {client_address}')
    server_socket.sendto(data, client_address)
```

Output:

```
C:\Users\NITHYALAKSHMI\Desktop\exercise folder\CN\exercise 3>python udp_server.py
Server listening on ('localhost', 12000)
Received ping from ('127.0.0.1', 54640)
Received ping from ('127.0.0.1', 54640)
Received ping from ('127.0.0.1', 54640)
Received ping from ('127.0.0.1', 54640)
Received ping from ('127.0.0.1', 54640)
Received ping from ('127.0.0.1', 54640)
Received ping from ('127.0.0.1', 54640)
Received ping from ('127.0.0.1', 54640)
Received ping from ('127.0.0.1', 54640)
Received ping from ('127.0.0.1', 54640)
```

Udp pinger client program:

```
import socket
import time
SERVER_ADDRESS = ('localhost', 12000)
MESSAGE = b'Ping'
client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

for i in range(10):
    start_time = time.time()
    client_socket.sendto(MESSAGE, SERVER_ADDRESS)
    client_socket.settimeout(1)
    try:
        data, server = client_socket.recvfrom(1024)
        end_time = time.time()
        rtt = (end_time - start_time) * 1000 # convert to milliseconds
        print(f'Ping {i+1} RTT: {rtt:.2f} ms')
    except socket.timeout:
        print(f'Ping {i+1} timed out')
client_socket.close()
```

Output:

```
C:\Users\NITHYALAKSHMI\Desktop\exercise folder\CN\exercise 3>python udp.py
Ping 1 RTT: 3.16 ms
Ping 2 RTT: 0.00 ms
Ping 3 RTT: 1.00 ms
Ping 4 RTT: 1.00 ms
Ping 5 RTT: 1.00 ms
Ping 6 RTT: 0.36 ms
Ping 7 RTT: 0.00 ms
Ping 8 RTT: 1.07 ms
Ping 9 RTT: 0.00 ms
Ping 10 RTT: 1.00 ms
```

3e. Mail client

Program:

```
from socket import *
# Message to send
msg = '\r\nI love computer networks!'
endmsg = '\r\n.\r\n'
```

```
# Choose a mail server (e.g. Google mail server) and call it mailserver
mailserver = 'smtp.gmail.com'

# Create socket called clientSocket and establish a TCP connection with mailserver
clientSocket = socket(AF_INET, SOCK_STREAM)

# Port number may change according to the mail server
clientSocket.connect((mailserver, 587))
recv = clientSocket.recv(1024)
print (recv)
if recv[:3] != '220':
    print ('220 reply not received from server.')

# Send HELO command and print server response.
helocommand = "This is a test email"
clientSocket.sendall(helocommand.encode('utf-8'))
recv1 = clientSocket.recv(1024)
print (recv1)
if recv1[:3] != '250':
    print ('250 reply not received from server.')

# Send MAIL FROM command and print server response.
mailfrom = 'MAIL FROM: <alice@gmail.com>\r\n'
clientSocket.send(mailfrom)
recv2 = clientSocket.recv(1024)
print (recv2)
if recv2[:3] != '250':
    print ('250 reply not received from server.')


# Send RCPT TO command and print server response.
rcptto = 'RCPT TO: <bob@yahoo.com>\r\n'
clientSocket.send(rcptto)
recv3 = clientSocket.recv(1024)
print (recv3)
if recv3[:3] != '250':
    print ('250 reply not received from server.')

# Send DATA command and print server response.
data = 'DATA\r\n'
clientSocket.send(data)
recv4 = clientSocket.recv(1024)
print (recv4)
if recv4[:3] != '354':
    print ('354 reply not received from server.')

# Send message data.
clientSocket.send('SUBJECT: Greeting To you!\r\n')
clientSocket.send('test again')
clientSocket.send(msg)
```

```
# Message ends with a single period.
clientSocket.send(endmsg)
recv5 = clientSocket.recv(1024)
print (recv5)
if recv5[:3] != '250':
   print ('250 reply not received from server.')

# Send QUIT command and get server response.
quitcommand = 'QUIT\r\n'
clientSocket.send(quitcommand)
recv6 = clientSocket.recv(1024)
print (recv6)
if recv6[:3] != '221':
   print ('221 reply not received')
```

Output:

```
C:\Users\NITHYALAKSHMI\Desktop\exercise folder\CN\exercise 3>python mail_client.py
b'220 smtp.gmail.com ESMTP j18-20020aa78dd2000000b005938f5b7231sm4441455pfr.201 - gsmtp\r\n'
220 reply not received from server.
```

3f. Multithreaded web proxy server

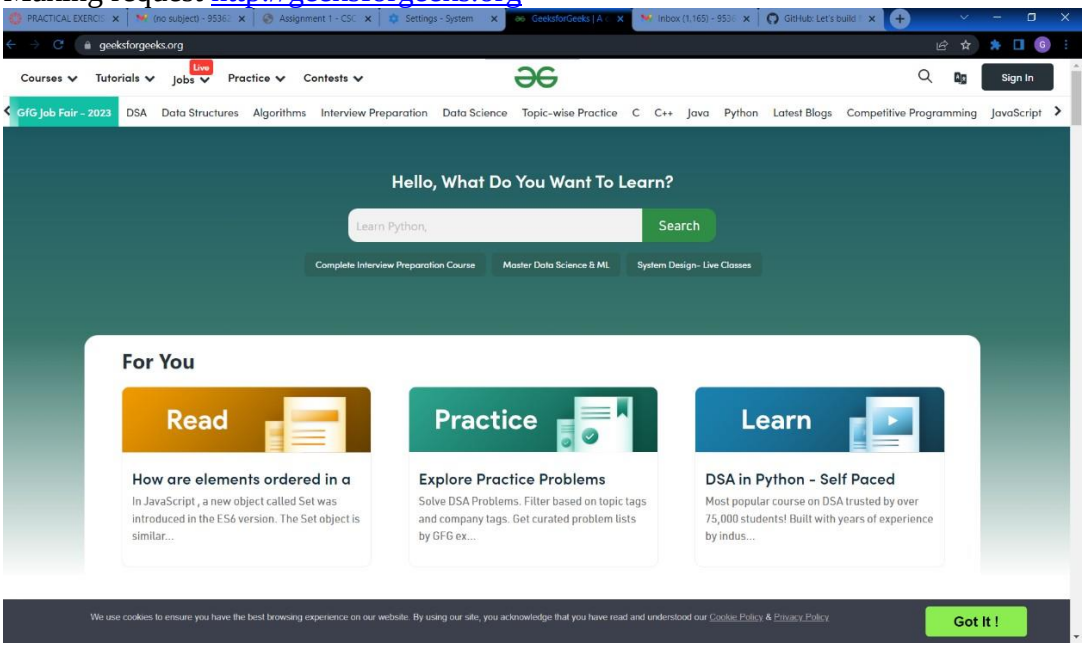
Program:

```
import socket
import threading
origin_server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
response_data = ""
def handle_request(client_socket):
   request_data = client_socket.recv(1024)
   # parse request_data and get host and port information
   # create a new socket and connect to the origin server
   # send the original request to the origin server
   origin_server_response = origin_server_socket.recv(1024)
   # parse the origin_server_response and get the data
   # create a new response to send back to the client
   client_socket.send(response_data)
   # close the sockets
   client_socket.close()
   origin_server_socket.close()

def start_server(host, port):
   server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
   server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
   server_socket.bind((host, port))
   server_socket.listen(5)
   print('Listening on {}:{}'.format(host, port))
   while True:
      client_socket, address = server_socket.accept()
      print('Accepted connection from {}:{}'.format(address[0], address[1]))
      t = threading.Thread(target=handle_request, args=(client_socket,))
      t.start()

if__name__== '_main_':
   start_server('localhost', 8080)
```

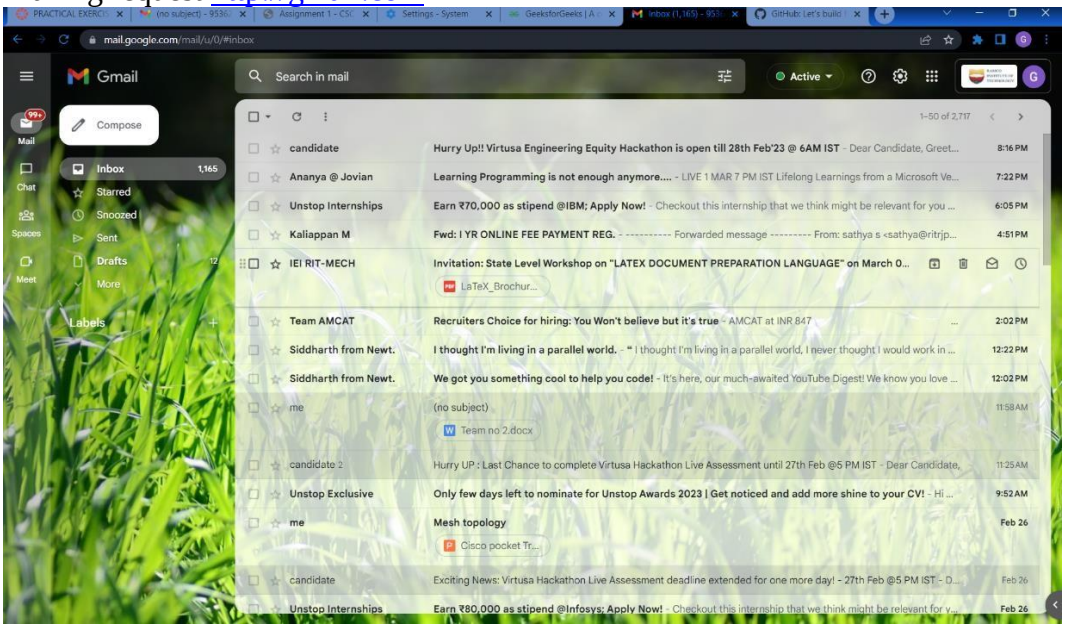
Output:

```
C:\Users\NITHYALAKSHMI\Desktop\exercise folder\CN\exercise 3>python webproxy.py
Listening on localhost:8080
```

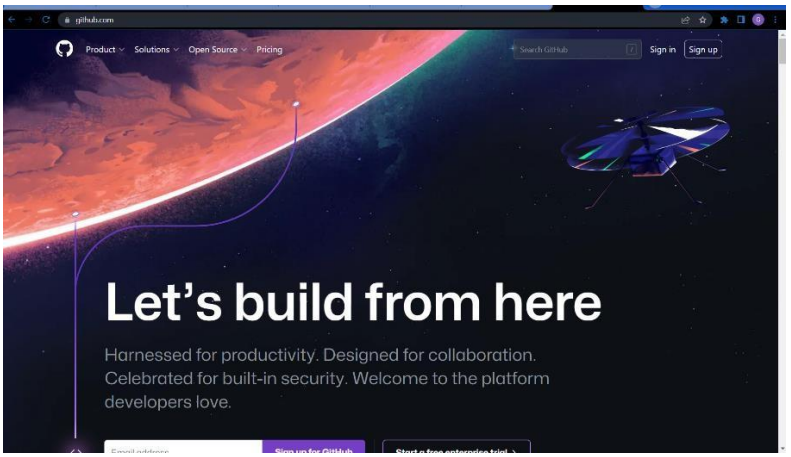
Making request http://geeksforgeeks.org



Making request http://gmail.com



Making request of http://github.com



Result:

Thus the program to write python program for application on TCP sockets was written and executed successfully.

Exercise 4
Simulate DNS server using UDP sockets

Ex.no.: 4 **Date:**

Aim:

To write a python program to simulate DNS server using UDP sockets

Algorithm:

Step 1: Start
Step 2: Initialize the dns table in the server program
Step 3: Get the socket from AF_INET and SOCK_DGRAM
Step4: Get the domain name from the client
Step 5: Decode the data and get the ipaddress from dns_table
Step 6: Receive the data from the server to client
Step 7: End

Program:

DNS server

```
import socket
dns_table = {"www.google.com":"192.165.1.1",
 "www.youtube.com":"192.165.1.2",
  "www.python.org": "192.165.1.3",
  "www.amazon.in": "192.165.1.4",
  "www.ritrjpm.ac.in":"192.165.1.5"}

s = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
print("Starting Server")
s.bind(("127.0.0.1",1234))
while True:
    data,address = s.recvfrom(1024)
    print(f"{address} wants to fetch data!")
    data = data.decode()
    ip = dns_table.get(data,"Not Found!").encode()
    send = s.sendto(ip,address)
```

Output:

```
C:\Users\NITHYALAKSHMI\Desktop\exercise folder\CN>python dns_server.py --port 80
80
Starting Server
('127.0.0.1', 60077) wants to fetch data!
('127.0.0.1', 60077) wants to fetch data!
```

DNS client

```
import socket
hostname = socket.gethostname()
ipaddr = "127.0.0.1"
s = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
addr = (ipaddr,1234)
c = "Y"
```

```
while c.upper() == "Y":
    req_domain = input("Enter domain name for which the ip is needed: ")
    send = s.sendto(req_domain.encode(),addr)
    data,address = s.recvfrom(1024)
    reply_ip = data.decode().strip()
    print(f"The ip for the domain name {req_domain}: {reply_ip}")
    c = input("Continue(y/n)")
s.close()
```

Output:

```
C:\Users\NITHYALAKSHMI\Desktop\exercise folder\CN>python dns_client.py --port 8080
Enter domain name for which the ip is needed: google.com
The ip for the domain name google.com: Not Found!
Continue(y/n)y
Enter domain name for which the ip is needed: ritrjpm.ac.in
The ip for the domain name ritrjpm.ac.in: Not Found!
Continue(y/n)n
```

Result:

Thus the program to write python program to simulate DNS server using UDP sockets was written and executed successfully.

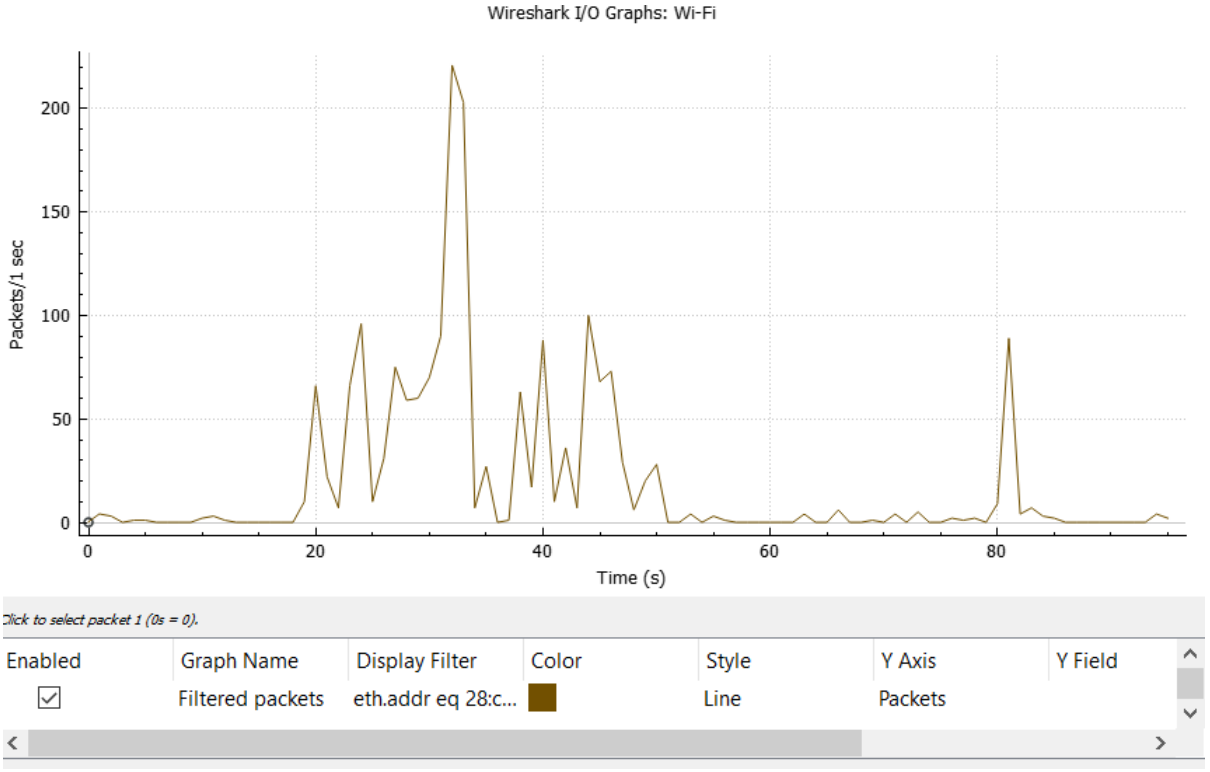
Exercise 5
Use a tool like Wireshark to capture packets and examine packets.

Ex.no.:5 **Date:**

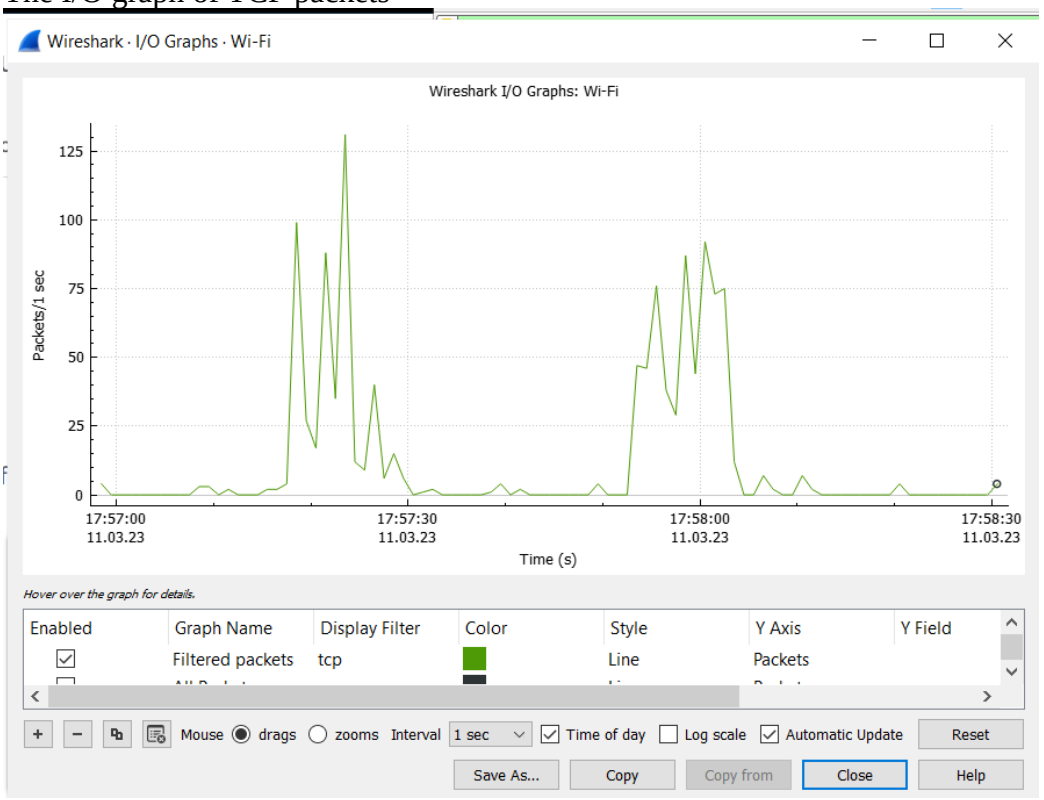
Aim:
To use a tool like Wireshark to capture packets and examine packets.

Algorithm:
Step 1: Start
Step 2: Open Wireshark tool
Step 3: Start capturing the packets for Wi-fi
Step 4: Apply various filters to analyze the packets
Step 5: End

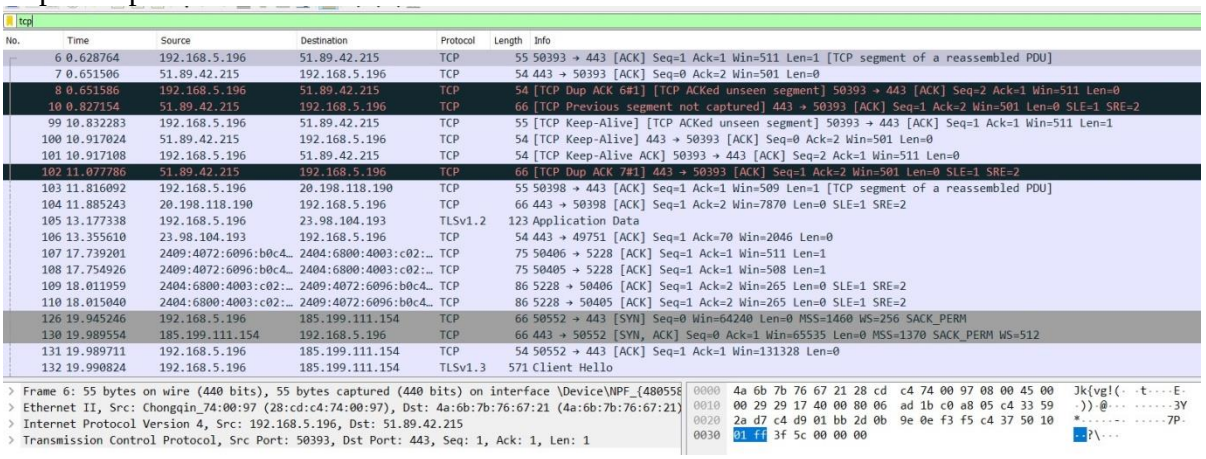
Procedure:
The I/O graph of UDP packets.



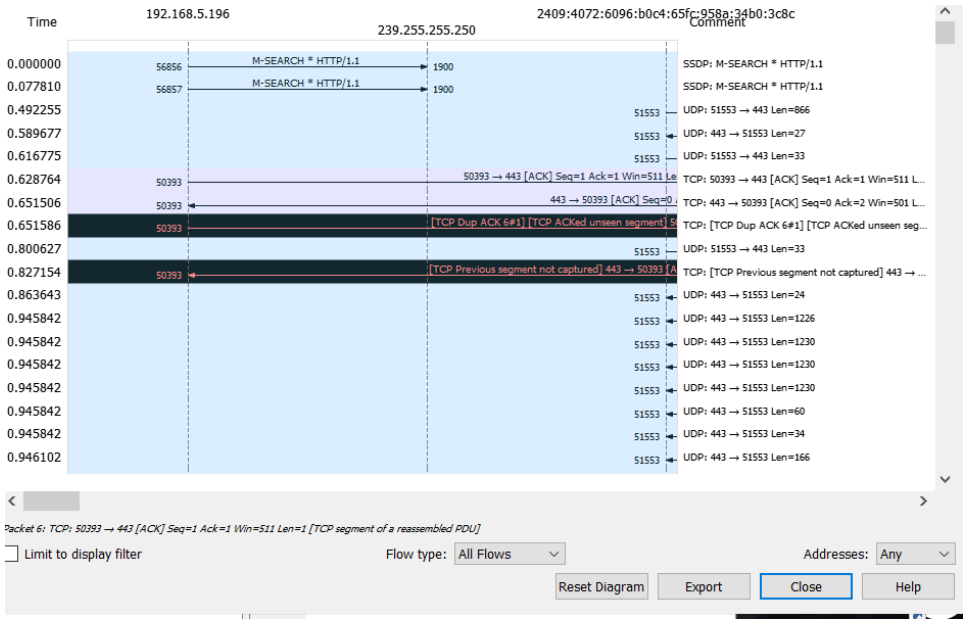
The I/O graph of TCP packets



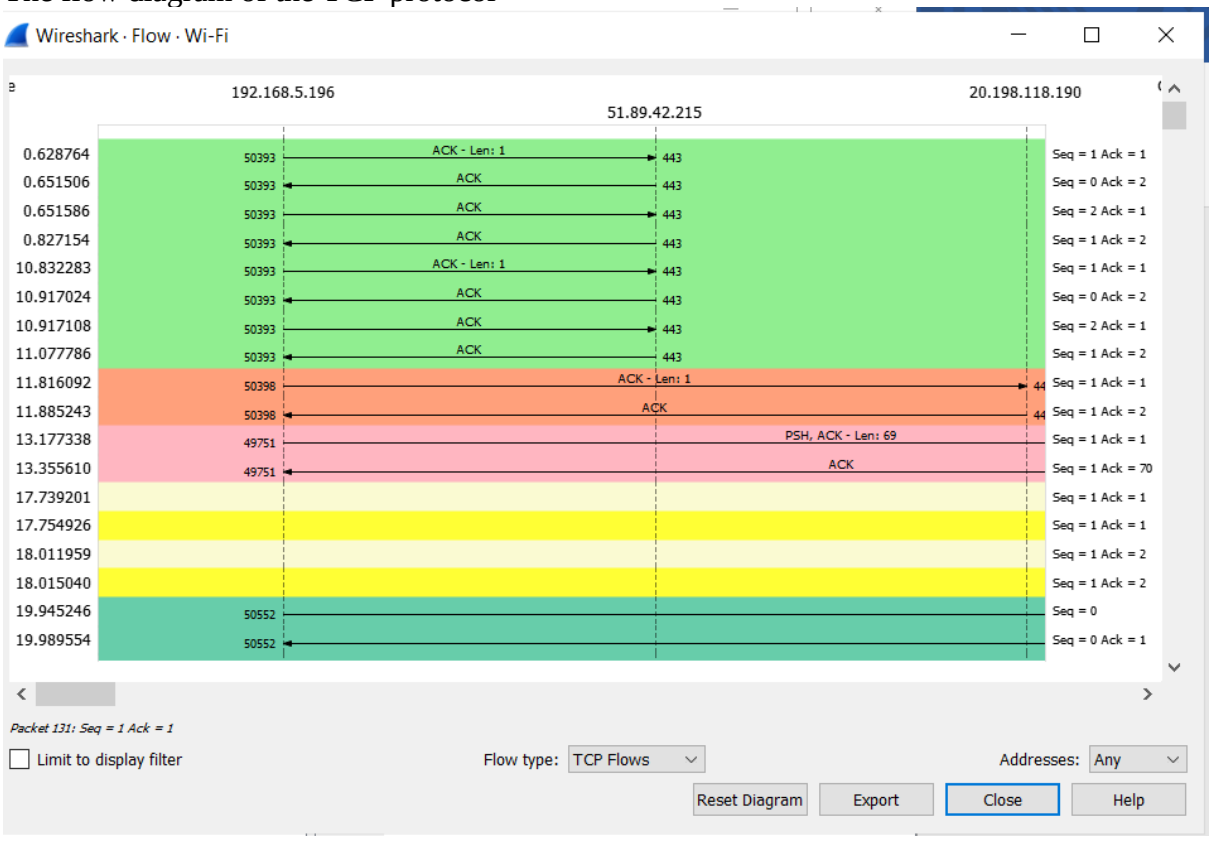
Captured packets



The flow diagram of the acknowledgements

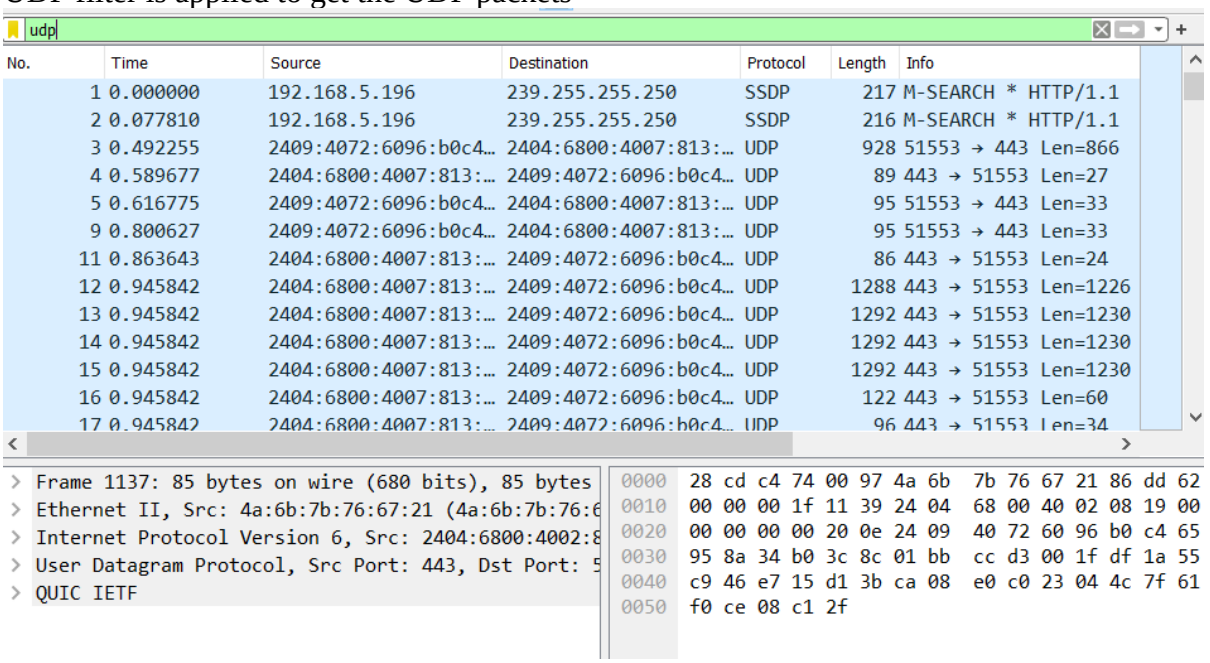


The flow diagram of the TCP protocol

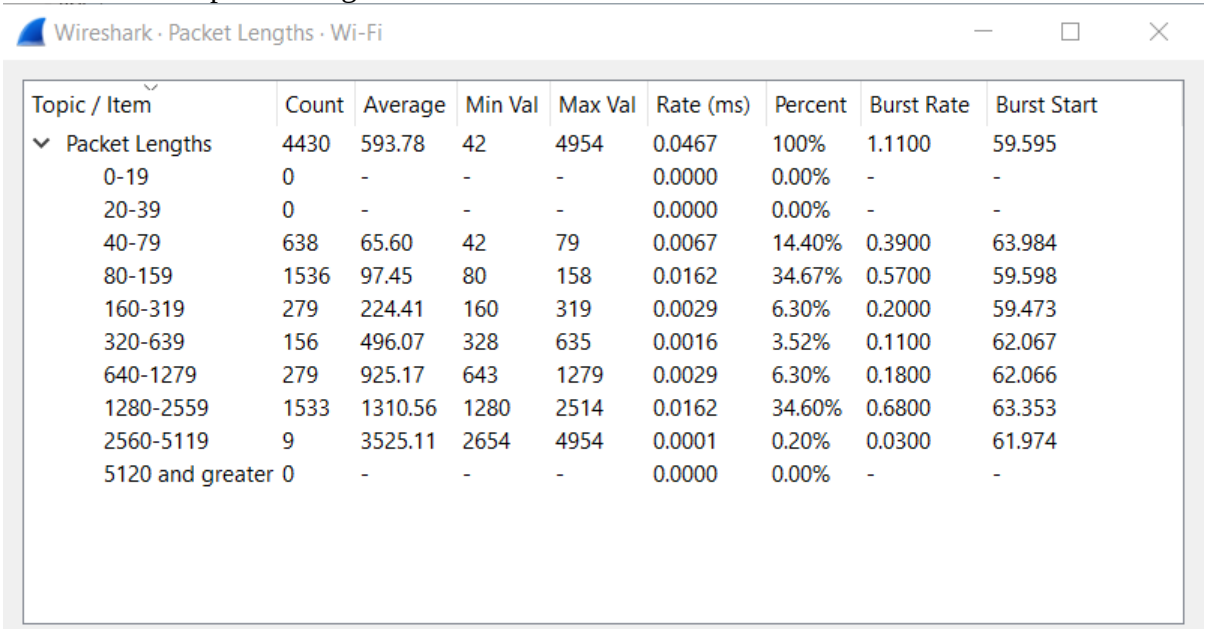


Observation: All the TCP requests made to the server 51.89.42.215 are successful and received acknowledgment.

UDP filter is applied to get the UDP packets



This show each packet lengths



Topic / Item	Count	Average	Min Val	Max Val	Rate (ms)	Percent	Burst Rate	Burst Start
Packet Lengths	4430	593.78	42	4954	0.0467	100%	1.1100	59.595
0-19	0	-	-	-	0.0000	0.00%	-	-
20-39	0	-	-	-	0.0000	0.00%	-	-
40-79	638	65.60	42	79	0.0067	14.40%	0.3900	63.984
80-159	1536	97.45	80	158	0.0162	34.67%	0.5700	59.598
160-319	279	224.41	160	319	0.0029	6.30%	0.2000	59.473
320-639	156	496.07	328	635	0.0016	3.52%	0.1100	62.067
640-1279	279	925.17	643	1279	0.0029	6.30%	0.1800	62.066
1280-2559	1533	1310.56	1280	2514	0.0162	34.60%	0.6800	63.353
2560-5119	9	3525.11	2654	4954	0.0001	0.20%	0.0300	61.974
5120 and greater	0	-	-	-	0.0000	0.00%	-	-

Observation: Maximum number of packets are 80-159 with a burst rate of 0.5700 and a rate of 0.0162. The average burst rate taken by the packets is 1.1100.

Final statistics of the packets received.

Statistics

Measurement	Captured	Displayed	Marked
Packets	4430	62 (1.4%)	—
Time span, s	94.839	40.794	—
Average pps	46.7	1.5	—
Average packet size, B	594	451	—
Bytes	2630463	27964 (1.1%)	0
Average bytes/s	27 k	685	—
Average bits/s	221 k	5484	—

Result:

Thus, using the wireshark tool to capture and examine packets was written and executed successfully.

Exercise 6
Simulate ARP/RARP protocols

Ex.no:6 **Date:**

Aim:
To write a python program to simulate ARP/RARP protocols

Algorithm:
Step 1: Start
Step 2: Build a server side program to listen to client
Step 3: Bind the IP address to 1234 and receive the message from client
Step 4: Build the client side program
Step 5: Get the input as ARP or RARP protocol
Step 6: If the input is ARP protocol then enter the IP address else input the MAC address
Step 7: End

Program:
Server:

```
import socket
table = {
   '192.168.1.1':'1E.5E.6R.99',
   '192.168.2.1': '2R.5T.6Y.11',
   '192.168.1.3' : '5F.4R.5E.80'
}
s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
s.bind((',1234))
s.listen()
clientsocket,address = s.accept()
print("Connection from",address,"has established")
ip = clientsocket.recv(1024)
ip = ip.decode("utf-8")
mac = table.get(ip,"No entry for given address")
clientsocket.send(mac.encode())
```

Output:

```
Z:\CN\exercise5>python arp_server.py --port 8080
Connection from ('127.0.0.1', 50810) has established

Z:\CN\exercise5>
```

Client:

```
import socket
s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
s.connect(('localhost',1234))
a = input("ARP or RARP")
if (a=="ARP"):
   add = input("Enter IP: ")
elif(a=="RARP"):
   add = input("Enter MAC: ")
s.send(add.encode())
```

```
mac = s.recv(1024)
mac = mac.decode("utf-8")
if(a == "ARP"):
    print('MAC of',add,'is: ',mac)
else:
    print("IP of",add,"is: ",mac)
```

Output:

```
Z:\CN\exercise5>python arp_client.py --port 8080
ARP or RARPARP
Enter IP: 192.168.1.1
MAC of 192.168.1.1 is:  1E.5E.6R.99

Z:\CN\exercise5>
```

Result:

Thus the python program to simulate ARP/RARP protocols was written and executed successfully

Exercise 7
Study of Network simulator (NS) and Simulation of Congestion Control Algorithms using NS.

Ex. No. :7 **Date:**

Aim:

To study of Network simulator (NS) and Simulation of Congestion Control Algorithms using NS.

Algorithm:

Step 1: Start
Step 2: Create a simulator
Step 3: Create a simple topography
Step 4: Create the nodes of the topography
Step 5: A new TCP connection given to the nodes
Step 6: Create a trace file to write the output
Step 7: End

Software used:

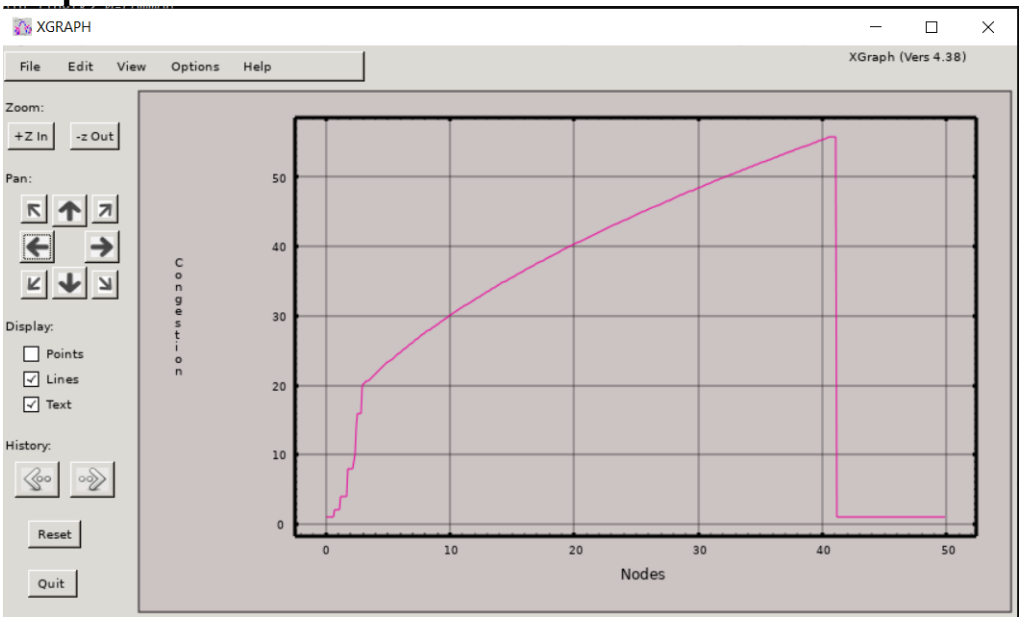
Network Simulator 2
Ubuntu

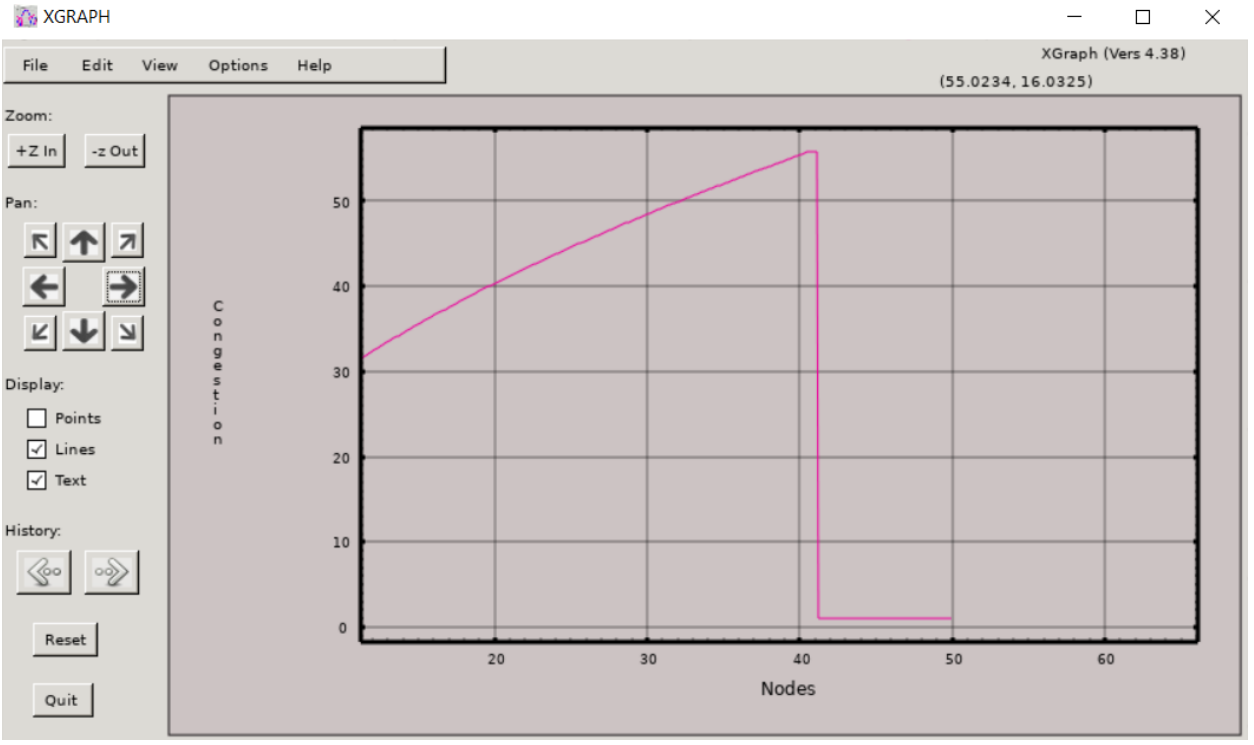
Program:

```
# create a simulator instance
set ns [new Simulator]
# create a topology
set topo [new Topography]
$topo load_flatgrid 100 100
create_nodes 100
$ns node-config -adhocRouting AODV
$ns node-config -llType LL
$ns node-config -macType Mac/802_11
$ns node-config -ifqType Queue/DropTail/PriQueue
$ns node-config -ifqLen 50
$ns node-config -antType Antenna/OmniAntenna
$ns node-config -propType Propagation/TwoRayGround
$ns node-config -phyType Phy/WirelessPhy
$ns node-config -channel Channel/WirelessChannel
$ns node-config -topoInstance $topo

# Create TCP flows
set tcp [new Agent/TCP]
$tcp set class_ 2
set sink [new Agent/TCPSink]
$ns attach-agent $node(0) $tcp
$ns attach-agent $node(99) $sink
$ns connect $tcp $sink
# Set congestion control algorithm
$tcp set window_ 10
$tcp set protocol Newreno
```

```
# Define simulation parameters
set DELAY 1ms
set STOP_TIME 10s
set now [$ns now]
$ns at $now+$DELAY "$tcp start"
$ns at $now+$STOP_TIME "$tcp stop"
$ns at $now+$STOP_TIME "stop"
proc stop {} {
    global ns tracefd
    $ns flush-trace
    close $tracefd
    $ns halt
}
# Start simulation
set tracefd [open out.tr w]
$ns trace-all $tracefd
$ns run
```

Output:





Result:

Thus to of Network simulator (NS) and Simulation of Congestion Control Algorithms using NS was written and executed successfully.

Exercise 8

Study of TCP/UDP performance using Simulation tool.

Ex.No.:8 **Date:**

Aim:

a) To study of TCP/UDP performance using Simulation tool (ns2)

b) To capture UDP packets using Wireshark and analyse them.

Algorithm:

Step 1: Start
Step 2: Create a new simulator
Step 3: Create duplex link for the nodes
Step 4: Create the TCP and UDP connection between the nodes
Step 5: Create the simulation
Step 6: End

Software used:

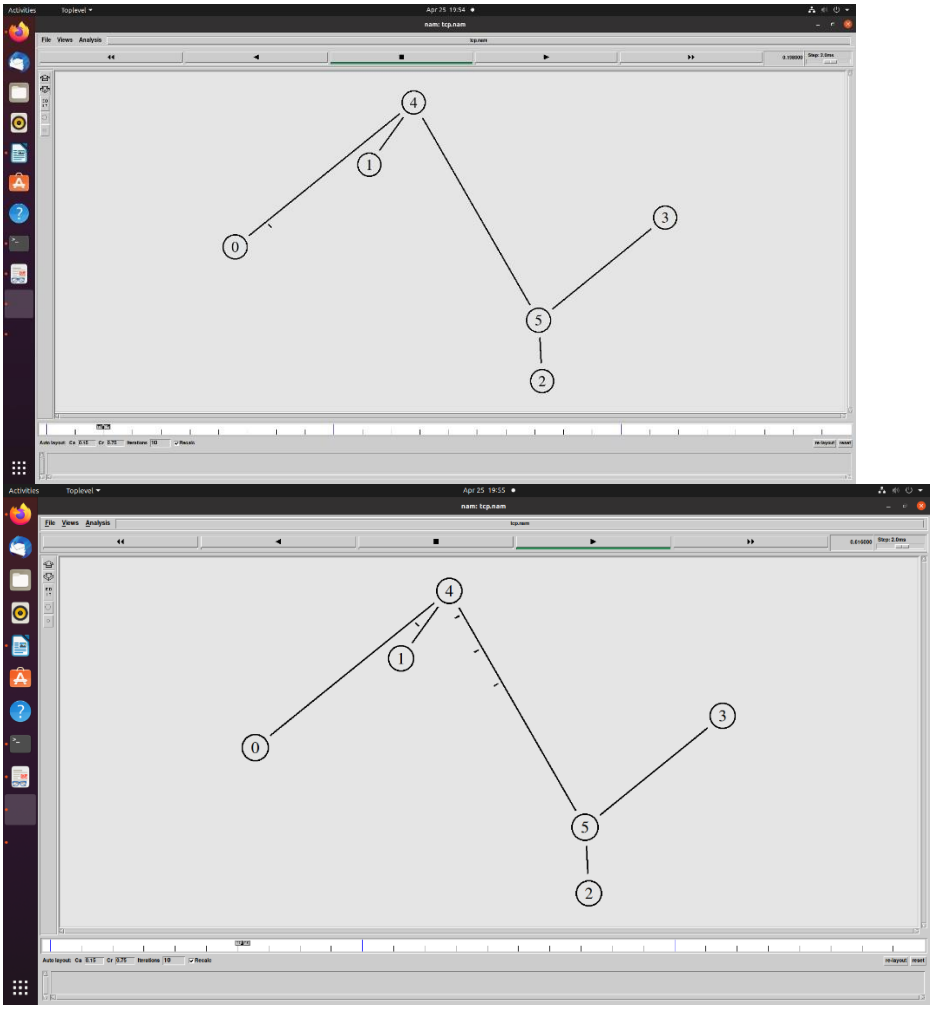
Network Simulator 2
Ubuntu
Wireshark

Program:

A) TCP Simulation:

```
set ns [new Simulator]
set nf [open tcp.nam w]
$ns namtrace-all $nf
set tf [open out.tr w]
$ns trace-all $tf
proc finish {} {
global ns nftf
$ns flush-trace
close  $nf
close $tf
exec namtcp.nam&
exit 0
}
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
$ns duplex-link $n0 $n4 1Mb 50ms DropTail
$ns duplex-link $n1 $n4 1Mb 50ms DropTail
$ns duplex-link $n2 $n5 1Mb 1ms DropTail
$ns duplex-link $n3 $n5 1Mb 1ms DropTail
$ns duplex-link $n4 $n5 1Mb 50ms DropTail
```

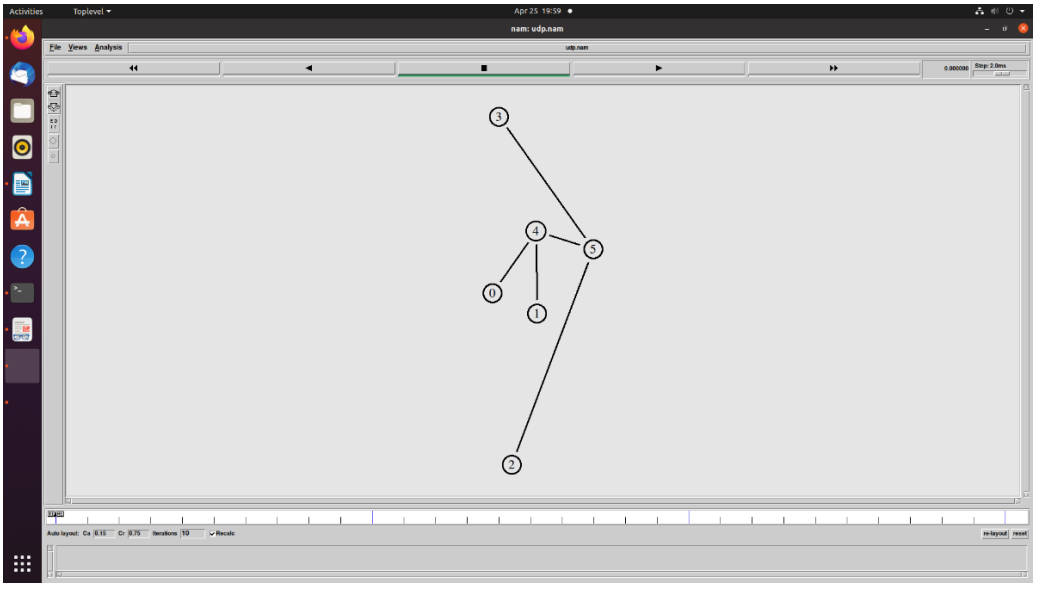
```
$ns duplex-link-op $n4 $n5 queuePos 0.5
set tcp [new Agent/TCP]
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n2 $sink
$ns connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns at 0.0 "$ftp start"
$ns at 2.5 "$ftp stop"
$ns at 3 "finish"
$ns run
```

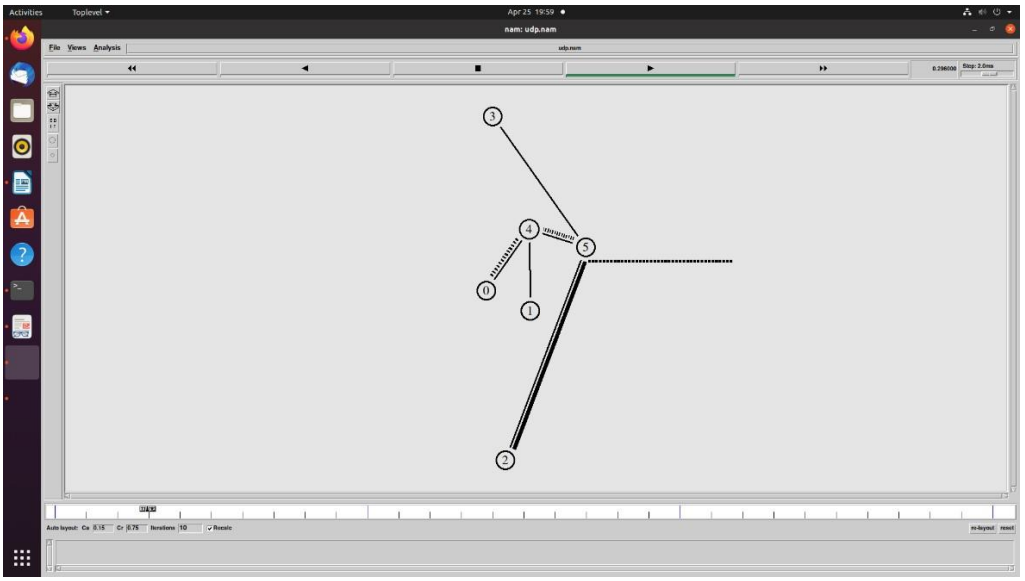


UDP simulation:

```
set ns [new Simulator]
set nf [open udp.nam w]
$ns namtrace-all $nf
set tf [open out.tr w]
$ns trace-all $tf
proc finish {} {
global ns nftf
$ns flush-trace
close  $nf
close $tf
```

```
exec namudp.nam&
exit 0
}
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
$ns duplex-link $n0 $n4 1Mb 50ms DropTail
$ns duplex-link $n1 $n4 1Mb 50ms DropTail
$ns duplex-link $n2 $n5 0.1Mb 1ms DropTail
$ns duplex-link $n3 $n5 1Mb 1ms DropTail
$ns duplex-link $n4 $n5 1Mb 50ms DropTail
$ns duplex-link-op $n2 $n5 queuePos 1
set tcp [new Agent/UDP]
$ns attach-agent $n0 $tcp
set sink [new Agent/Null]
$ns attach-agent $n2 $sink
$ns connect $tcp $sink
set ftp [new Application/Traffic/CBR]
$ftp attach-agent $tcp
$ns at 0.0 "$ftp start"
$ns at 2.5 "$ftp stop"
$ns at 3 "finish"
$ns run
```





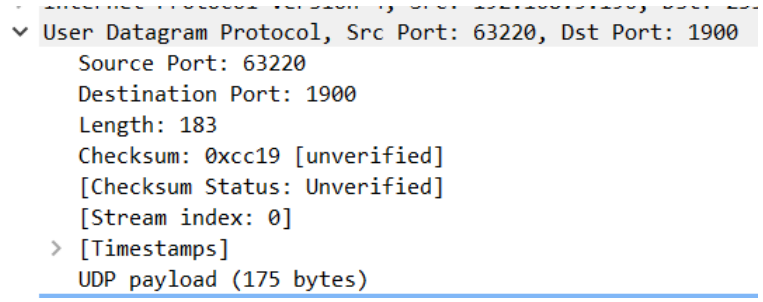
B) Capturing packets using wireshark

108	17.843388	192.168.5.196	192.168.5.194	DNS	78	Standard query 0x8832 A www.googleapis.com
109	17.843880	192.168.5.196	192.168.5.194	DNS	78	Standard query 0x74a8 AAAA www.googleapis.com
110	17.844215	192.168.5.196	192.168.5.194	DNS	78	Standard query 0x824a HTTPS www.googleapis.com
116	17.878576	192.168.5.194	192.168.5.196	DNS	334	Standard query response 0x8832 A www.googleapis.com A 142.250.196.10 A 142.250.196.42 A 142.250.196.74 A 142.250.196.170 A 1
117	17.892347	192.168.5.194	192.168.5.196	DNS	190	Standard query response 0x74a8 AAAA www.googleapis.com AAAA 2404:6800:4007:822::200a AAAA 2404:6800:4007:823::200a AAAA 2404
118	17.893035	192.168.5.194	192.168.5.196	DNS	135	Standard query response 0x824a HTTPS www.googleapis.com SOA ns1.google.com
205	18.253749	192.168.5.196	192.168.5.194	DNS	93	Standard query 0x8288 A appsitemsuggest-pa.googleapis.com
206	18.258146	192.168.5.196	192.168.5.194	DNS	93	Standard query 0xa9e0 AAAA appsitemsuggest-pa.googleapis.com
207	18.268304	192.168.5.196	192.168.5.194	DNS	93	Standard query 0xa3ed HTTPS appsitemsuggest-pa.googleapis.com
209	18.342153	192.168.5.194	192.168.5.196	DNS	349	Standard query response 0x8288 A appsitemsuggest-pa.googleapis.com A 216.58.196.170 A 142.250.76.42 A 142.250.195.74 A 142.2
210	18.342273	192.168.5.194	192.168.5.196	DNS	205	Standard query response 0xa9e0 AAAA appsitemsuggest-pa.googleapis.com AAAA 2404:6800:4007:817::200a AAAA 2404:6800:4007:818:

Nslookup command

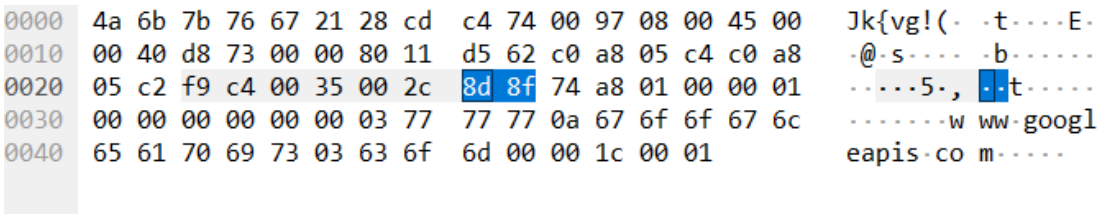
```
C:\Users\sys>nslookup
Default Server:  UnKnown
Address:  192.168.5.194

> nslookup ritrjpm.ac.in
Server:  ritrjpm.ac.in
Addresses:  107.154.163.176
          107.154.220.176

*** ritrjpm.ac.in can't find nslookup: BAD ERROR VALUE
> nslookup google.com
Server:  google.com
Addresses:  2404:6800:4007:82a::200e
          216.58.196.174

DNS request timed out.
    timeout was 2 seconds.
DNS request timed out.
    timeout was 2 seconds.
*** Request to google.com timed-out
>
```

1. **Select the first UDP segment in your trace. What is the packet number of this segment in the trace file? What type of application-layer payload or protocol message is being carried in this UDP segment? Look at the details of this packet in Wireshark. How many fields there are in the UDP header? What are the names of these fields?**



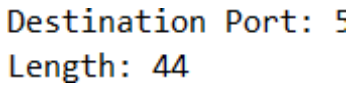
UDP header contains 4 fields: 1. source port; 2. destination port; 3. length; 4. Checksum

2. **By consulting the displayed information in Wireshark's packet content field for this packet (or by consulting the textbook), what is the length (in bytes) of each of the UDP header fields?**



The UDP header has a fixed length of 8 bytes. Each of these 4 header fields is 2 bytes long

3. **The value in the Length field is the length of what? Verify your claim with your captured UDP packet.**



The length field specifies the number of bytes in the UDP segment (header plus data). An explicit length value is needed since the size of the data field may differ from one UDP segment to the next. The length of UDP payload for selected packet is 32 bytes. 40 bytes - 8 bytes = 32 bytes.

4. **What is the maximum number of bytes that can be included in a UDP payload?**

The maximum number of bytes that can be included in a UDP payload is (2^16 – 1) bytes plus the header bytes. This gives 65535 bytes – 8 bytes = 65527 bytes.

5. **What is the largest possible source port number?**

   The largest possible source port number is (2^16 – 1) = 65535.

6. **What is the protocol number for UDP? Give your answer in both hexadecimal and decimal notation**.

The IP protocol number for UDP is 0x11 hex, which is 17 in decimal value.

```
0000  4a 6b 7b 76 67 21 28 cd  c4 74 00 97 08 00 45 00   Jk{vg!(· ·t····E·
0010  00 40 d8 73 00 00 80 11  d5 62 c0 a8 05 c4 c0 a8   ·@·s····· ·b·····
0020  05 c2 f9 c4 00 35 00 2c  8d 8f 74 a8 01 00 00 01   ·····5·, ··t·····
0030  00 00 00 00 00 00 03 77  77 77 0a 67 6f 6f 67 6c   ·······w ww·googl
0040  65 61 70 69 73 03 63 6f  6d 00 00 1c 00 01         eapis·co m·····
```

7. **Examine a pair of UDP packets in which the first packet is sent by your host and the second packet is a reply to the first packet. Describe the relationship between the port numbers in the two packets.**

The source port of the UDP packet sent by the host is the same as the destination port of the reply packet, and conversely the destination port of the UDP packet sent by the host is the same as the source port of the reply packet.

```
User Datagram Protocol, Src Port: 63940, Dst Port: 53
    Source Port: 63940
    Destination Port: 53
```

8. **Visualize and Analyse the data generated**

```
from google.colab import drive
drive.mount('/content/drive')

import pandas as pd
import numpy as np

df = pd.read_csv('/content/drive/MyDrive/datasets/UDP_dataset.csv')
df.head(3)
```

	No.	Time	Source	Destination	Protocol	Length	Info
0	5	5.592629	192.168.5.196	239.255.255.250	SSDP	217	M-SEARCH * HTTP/1.1
1	34	6.607282	192.168.5.196	239.255.255.250	SSDP	217	M-SEARCH * HTTP/1.1
2	37	7.616019	192.168.5.196	239.255.255.250	SSDP	217	M-SEARCH * HTTP/1.1

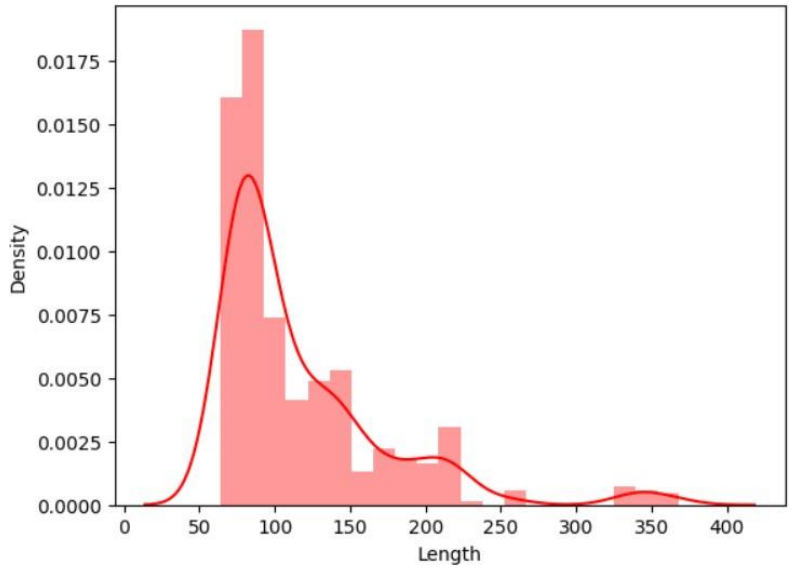
```
df.isna().sum()
```

```
No.            0
Time           0
Source         0
Destination    0
Protocol       0
Length         0
Info           0
dtype: int64
```

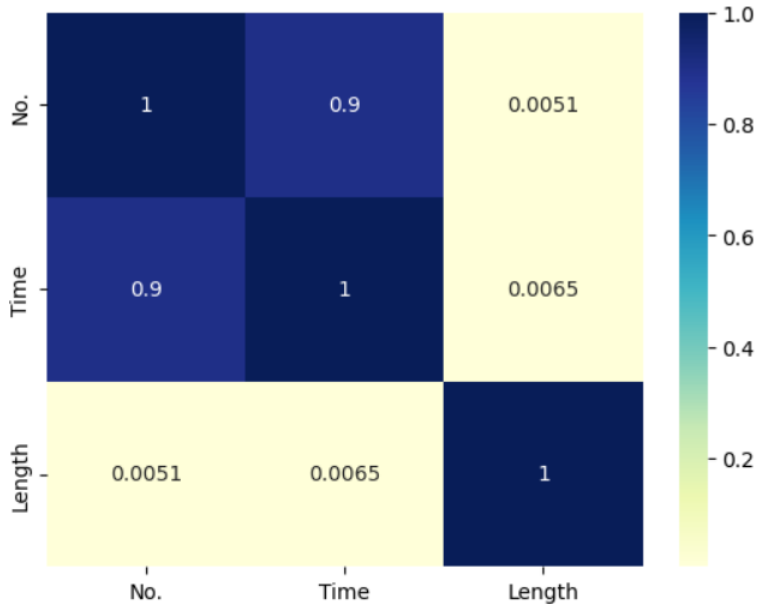
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 468 entries, 0 to 467
Data columns (total 7 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   No.          468 non-null    int64
 1   Time         468 non-null    float64
 2   Source       468 non-null    object
 3   Destination  468 non-null    object
 4   Protocol     468 non-null    object
 5   Length       468 non-null    int64
 6   Info         468 non-null    object
dtypes: float64(1), int64(2), object(4)
memory usage: 25.7+ KB
```

```
import matplotlib.pyplot as plt
import seaborn as sns
sns.distplot(df['Length'],color='r')
```




```
sns.heatmap(df.corr(),cmap='YlGnBu')
```




```
df['Destination'].unique()
```

```
array(['239.255.255.250', '192.168.5.194', '192.168.5.196',
       '107.154.163.176', '2404:6800:4007:82a::200e'], dtype=object)
```

```
df['Source'].unique()
```

```
array(['192.168.5.196', '192.168.5.194', '107.154.163.176',
       '2409:4072:6c03:e9db:69d4:7b28:befe:2071'], dtype=object)
```

```
trans_data = new_data.transpose()
trans_data.head(3)
```

	Destination	Protocol
0	239.255.255.250	SSDP
1	239.255.255.250	SSDP
2	239.255.255.250	SSDP

Result:

Thus to

a) Study of TCP/UDP performance using Simulation tool (ns2)
b) Capture UDP packets using Wireshark and analyse them.

was written and executed successfully.

Exercise 9

Simulation of Distance Vector/ Link State Routing algorithm

Ex.No. :9 **Date:**

Aim:
To implement simulation of Distance Vector/ Link State Routing algorithm

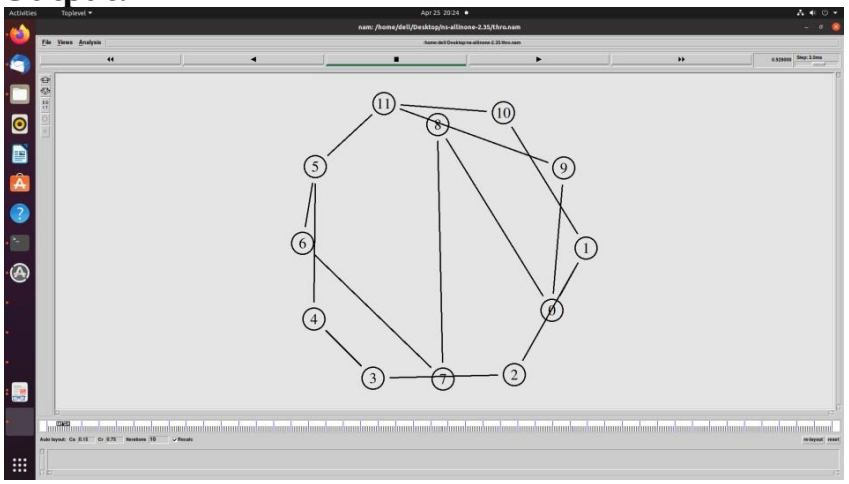
Algorithm:
Step 1: Start
Step2 : Create a new simulator
Step 3: Create a topography and duplex link is given between the nodes
Step 4: TCP and UDP connection is given to the nodes
Step 5: End

Software used:
Network Simulator 2
Ubuntu

Program:
```
set ns [new Simulator]
set nr [open thro.tr w]
$ns trace-all $nr
set nf [open thro.nam w]
$ns namtrace-all $nf
proc finish { } {
global ns nr nf
$ns flush-trace
close  $nf
close $nr
exec nam thro.nam &
exit 0
}
for { set i 0 } { $i < 12} { incr i 1 } {
set n($i) [$ns node]}
for {set i 0} {$i < 8} {incr i} {
$ns duplex-link $n($i) $n([expr $i+1]) 1Mb 10ms DropTail }
$ns duplex-link $n(0) $n(8) 1Mb 10ms DropTail
$ns duplex-link $n(1) $n(10) 1Mb 10ms DropTail
$ns duplex-link $n(0) $n(9) 1Mb 10ms DropTail
$ns duplex-link $n(9) $n(11) 1Mb 10ms DropTail
$ns duplex-link $n(10) $n(11) 1Mb 10ms DropTail
$ns duplex-link $n(11) $n(5) 1Mb 10ms DropTail
set udp0 [new Agent/UDP]
$ns attach-agent $n(0) $udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0
```

```
set null0 [new Agent/Null]
$ns attach-agent $n(5) $null0
$ns connect $udp0 $null0
set udp1 [new Agent/UDP]
$ns attach-agent $n(1) $udp1
set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 500
$cbr1 set interval_ 0.005
$cbr1 attach-agent $udp1
set null0 [new Agent/Null]
$ns attach-agent $n(5) $null0
$ns connect $udp1 $null0
$ns rtproto DV
$ns rtmodel-at 10.0 down $n(11) $n(5)
$ns rtmodel-at 15.0 down $n(7) $n(6)
$ns rtmodel-at 30.0 up $n(11) $n(5)
$ns rtmodel-at 20.0 up $n(7) $n(6)
$udp0 set fid_ 1
$udp1 set fid_ 2
$ns color 1 Red
$ns color 2 Green
$ns at 1.0 "$cbr0 start"
$ns at 2.0 "$cbr1 start"
$ns at 45 "finish"
$ns run
```

Output:



Result:

Thus to implement simulation of Distance Vector/ Link State Routing algorithm was written and executed successfully

Exercise 10
Simulation of an error correction code (like CRC)

Ex. No. :10 **Date:**

Aim:
To write a python program to implement simulation of an error correction code (like CRC)

Algorithm:
Step 1: Start
Step 2: Create a server program that is used to connect the client
Step 3: The CRC function is implemented to the data recived and sent
Step 4: The client program is built to make request to the server
Step 5: End

Software Used:
Python
Visual Studio Code

Program:
Server:

```
import socket
import crcmod.predefined

# Define the CRC algorithm to use
crc_func = crcmod.predefined.mkCrcFun('crc-32')

# Create a socket and bind to a local port
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind(('127.0.0.1', 8000))
server_socket.listen(1)

# Wait for a client to connect
print('Waiting for client connection...')
client_socket, address = server_socket.accept()
print(f'Connected to client at {address}')

# Receive data from client
data = client_socket.recv(1024)

# Calculate CRC
crc = crc_func(data)

# Send CRC to client
client_socket.sendall(crc.to_bytes(4, byteorder='big'))

# Close the sockets
client_socket.close()
server_socket.close()
```

```
C:\Users\sys\Desktop>python crc_check_server.py
Waiting for client connection...
Connected to client at ('127.0.0.1', 50019)
```

Client

```
import socket
import crcmod.predefined

# Define the CRC algorithm to use
crc_func = crcmod.predefined.mkCrcFun('crc-32')

# Create a socket and connect to the server
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect(('127.0.0.1', 8000))

# Send data to server
data = b'Hello, world!'
client_socket.sendall(data)

# Receive CRC from server
crc_bytes = client_socket.recv(4)
crc = int.from_bytes(crc_bytes, byteorder='big')

# Verify CRC
if crc == crc_func(data):
    print('CRC check passed')
else:
    print('CRC check failed')

# Close the socket
client_socket.close()
```

```
C:\Users\sys\Desktop>python crc_check.py
CRC check passed
```

Result:

Thus to write a python program to simulation of an error correction code (like CRC) was witten and executed successfully

Additional exercises

1. ## SMTP LAB:

```
from socket import *
# Message to send
msg = '\r\nI love computer networks!'
endmsg = '\r\n.\r\n'

# Choose a mail server (e.g. Google mail server) and call it mailserver
mailserver = 'smtp.gmail.com'

# Create socket called clientSocket and establish a TCP connection with mailserver
clientSocket = socket(AF_INET, SOCK_STREAM)

# Port number may change according to the mail server
clientSocket.connect((mailserver, 587))
recv = clientSocket.recv(1024)
print (recv)
if recv[:3] != '220':
   print ('220 reply not received from server.')

# Send HELO command and print server response.
helocommand = "This is a test email"
clientSocket.sendall(helocommand.encode('utf-8'))
recv1 = clientSocket.recv(1024)
print (recv1)
if recv1[:3] != '250':
   print ('250 reply not received from server.')

# Send MAIL FROM command and print server response.
mailfrom = 'MAIL FROM: <alice@gmail.com>\r\n'
clientSocket.send(mailfrom)
recv2 = clientSocket.recv(1024)
print (recv2)
if recv2[:3] != '250':
   print ('250 reply not received from server.')


# Send RCPT TO command and print server response.
rcptto = 'RCPT TO: <bob@yahoo.com>\r\n'
clientSocket.send(rcptto)
recv3 = clientSocket.recv(1024)
print (recv3)
if recv3[:3] != '250':
   print ('250 reply not received from server.')

# Send DATA command and print server response.
data = 'DATA\r\n'
clientSocket.send(data)
recv4 = clientSocket.recv(1024)
print (recv4)
if recv4[:3] != '354':
```

```
    print ('354 reply not received from server.')

# Send message data.
clientSocket.send('SUBJECT: Greeting To you!\r\n')
clientSocket.send('test again')
clientSocket.send(msg)

# Message ends with a single period.
clientSocket.send(endmsg)
recv5 = clientSocket.recv(1024)
print (recv5)
if recv5[:3] != '250':
    print ('250 reply not received from server.')

# Send QUIT command and get server response.
quitcommand = 'QUIT\r\n'
clientSocket.send(quitcommand)
recv6 = clientSocket.recv(1024)
print (recv6)
if recv6[:3] != '221':
    print ('221 reply not received')
```

```
C:\Users\NITHYALAKSHMI\Desktop\exercise folder\CN\exercise 3>python mail_client.py
b'220 smtp.gmail.com ESMTP j18-20020aa78dd2000000b005938f5b7231sm4441455pfr.201 - gsmtp\r\n'
220 reply not received from server.
```

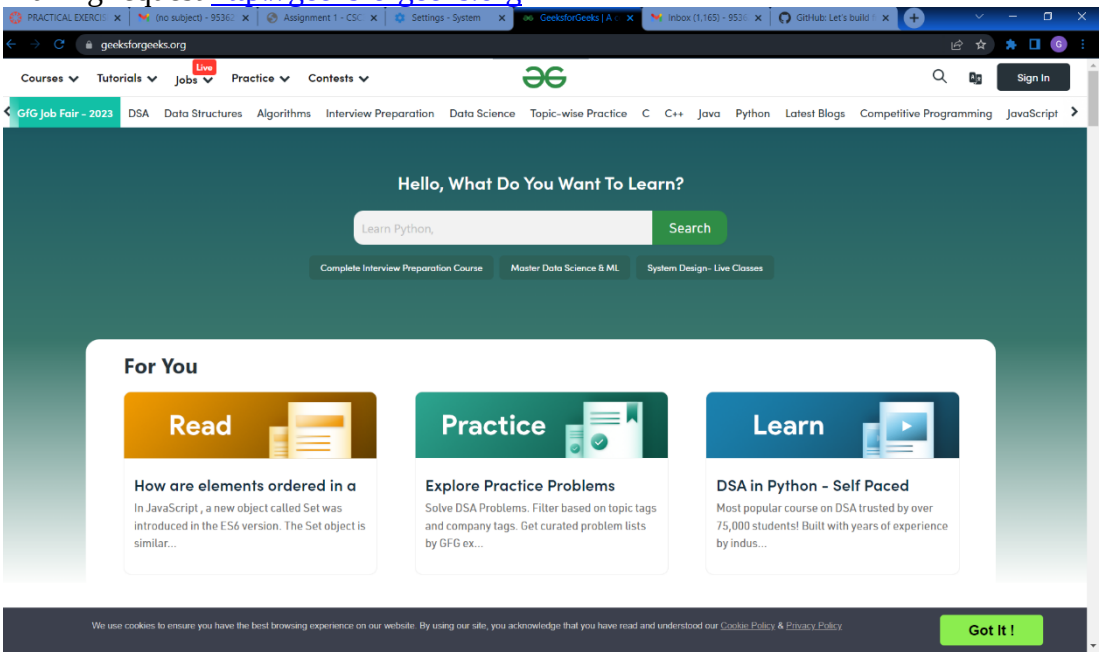
2. HTTP Web proxy server lab:

```
import socket
import threading
origin_server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
response_data = ""
def handle_request(client_socket):
request_data = client_socket.recv(1024)
    # parserequest_data and get host and port information
    # create a new socket and connect to the origin server
    # send the original request to the origin server
origin_server_response = origin_server_socket.recv(1024)
    # parse the origin_server_response and get the data
    # create a new response to send back to the client
client_socket.send(response_data)
    # close the sockets
client_socket.close()
origin_server_socket.close()

def start_server(host, port):
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
server_socket.bind((host, port))
server_socket.listen(5)
print('Listening on {}:{}'.format(host, port))
    while True:
client_socket, address = server_socket.accept()
```

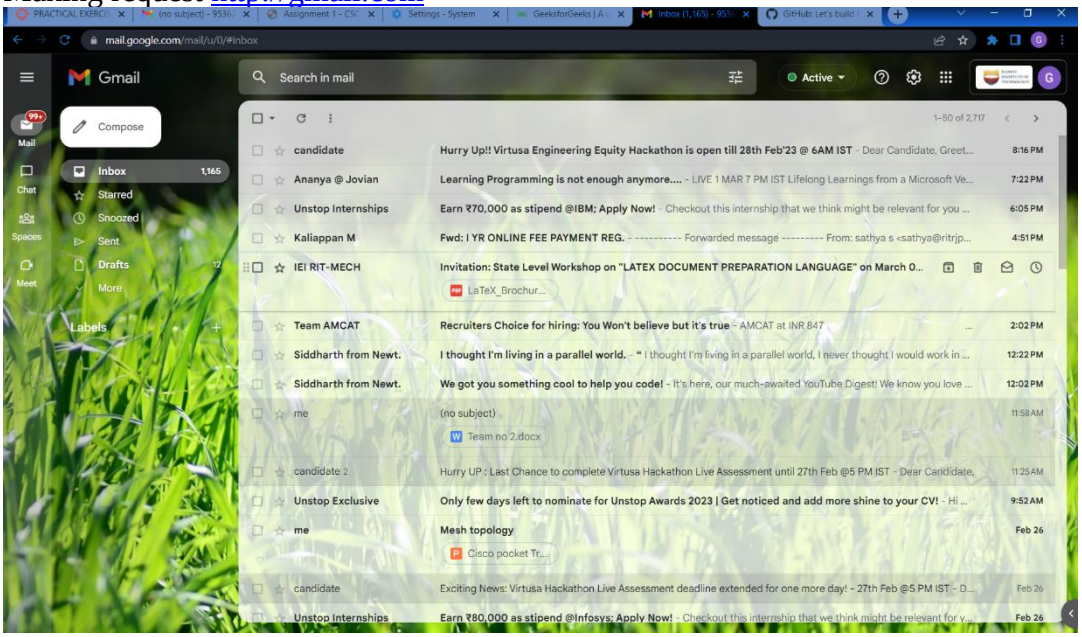
```
print('Accepted connection from {}:{}'.format(address[0], address[1]))
        t = threading.Thread(target=handle_request, args=(client_socket,))
t.start()

if __name__ == '__main__':
start_server('localhost', 8080)
```

```
C:\Users\NITHYALAKSHMI\Desktop\exercise folder\CN\exercise 3>python webproxy.py
Listening on localhost:8080
```

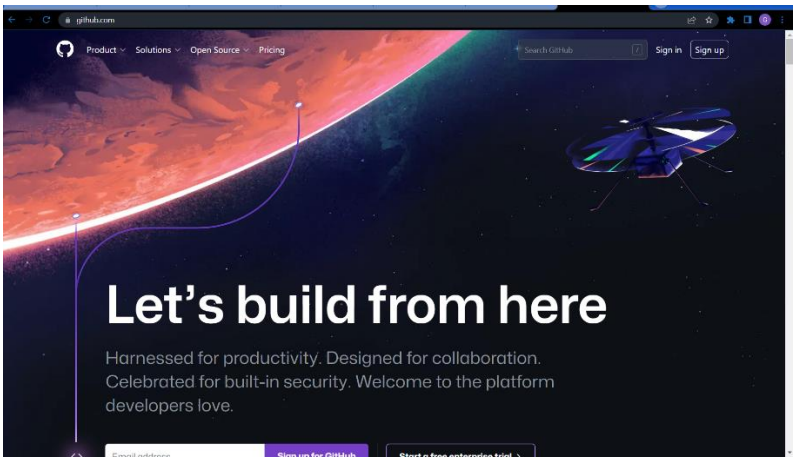
Making request http://geeksforgeeks.org



Making request http://gmail.com



Making request of http://github.com



3. ICMP Pinger:

Server:

```
# first of all import the socket library
import socket

# next create a socket object
s = socket.socket()
print ("Socket successfully created")

# reserve a port on your computer in our
# case it is 12345 but it can be anything
port = 12345

# Next bind to the port
# we have not typed any ip in the ip field
# instead we have inputted an empty string
# this makes the server listen to requests
# coming from other computers on the network
s.bind(('', port))
print ("socket binded to %s" %(port))

# put the socket into listening mode
s.listen(5)
print ("socket is listening")

# a forever loop until we interrupt it or
# an error occurs
while True:

# Establish connection with client.
c, addr = s.accept()
print ('Got connection from', addr )

  # send a thank you message to the client. encoding to send byte type.
c.send('Thank you for connecting'.encode())

  # Close the connection with the client
```

```
c.close()

  # Breaking once connection closed
Break
```

```
PS C:\Users\21ad016> & "C:/Program Files (x86)/Python36-32/python.exe" c:/Users/21ad016/Desktop/icmp_pinger.py
Socket successfully created
socket binded to 12345
socket is listening
Got connection from ('127.0.0.1', 50945)
```

Client

```
# Import socket module
import socket

# Create a socket object
s = socket.socket()

# Define the port on which you want to connect
port = 12345

# connect to the server on local computer
s.connect(('127.0.0.1', port))

# receive data from the server and decoding to get the string.
print (s.recv(1024).decode())
# close the connection
s.close()
```

```
C:\Users\21ad016\Desktop>python icmp_pinger_client.py
Thank you for connecting
```

4. Traceroute Lab:

Server:

```
import socket

# Create a UDP server socket
server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
server_socket.bind(('localhost', 12000))
print('Server is ready to receive traceroute requests.')

# Loop to receive traceroute requests
while True:
    # Wait for a traceroute request from a client
    message, client_address = server_socket.recvfrom(2048)
    print(f'Received traceroute request from client {client_address}')

    # Create a UDP client socket to send ICMP packets
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    client_socket.settimeout(1)

    # Loop to send ICMP packets with increasing TTL values
    for ttl in range(1, 31):
        # Set the TTL value for the socket
```

```
        client_socket.setsockopt(socket.IPPROTO_IP, socket.IP_TTL, ttl)

        try:
            # Send an ICMP packet to the destination host
            client_socket.sendto(b'', (message.decode(), 80))

            # Receive the ICMP response from the intermediate host
            intermediate_address, _ = client_socket.recvfrom(2048)

            # Print the intermediate host address
            print(f'{ttl}: {intermediate_address[0]}')
        except socket.timeout:
            # If the socket times out, print an error message
            print(f'{ttl}: *')

        # If the ICMP response is from the destination host, break the loop
        if intermediate_address[0] == message.decode():
            break

    # Close the client socket
    client_socket.close()
```

```
PS E:\Desktop\exercise folder\CN\Miniproject> & C:/Users/sys/AppDa
ject/traceroute_server.py"
Server is ready to receive traceroute requests.
Received traceroute request from client ('127.0.0.1', 64889)
```

Client side:

```
import socket

# Create a UDP client socket
client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# Get the destination host name from the user
destination_host = input('Enter the destination host name: ')

# Send a traceroute request to the server
client_socket.sendto(destination_host.encode(), ('localhost', 12000))

# Receive the intermediate host addresses from the server
print('Intermediate hosts:')
while True:
    message, _ = client_socket.recvfrom(2048)

    # If the server sends an empty message, break the loop
    if not message:
        break

    print(message.decode())

# Close the client socket
client_socket.close()
```

```
E:\Desktop\exercise folder\CN\Miniproject>python traceroute_client.py
Enter the destination host name: cn1
Intermediate hosts:
```

5. Video streaming using rtp:

Server side:

```
# This is server code to send video frames over UDP
import cv2, imutils, socket
import numpy as np
import time
import base64

BUFF_SIZE = 65536
server_socket = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
server_socket.setsockopt(socket.SOL_SOCKET,socket.SO_RCVBUF,BUFF_SIZE)
host_name = socket.gethostname()
host_ip = socket.gethostbyname(host_name)
print(host_ip)
port = 9999
socket_address = (host_ip,port)
server_socket.bind(socket_address)
print('Listening at:',socket_address)

vid = cv2.VideoCapture('nature.mp4') #  replace 'rocket.mp4' with 0 for webcam
fps,st,frames_to_count,cnt = (0,0,20,0)

while True:
	msg,client_addr = server_socket.recvfrom(BUFF_SIZE)
	print('GOT connection from ',client_addr)
	WIDTH=400
	while(vid.isOpened()):
		_,frame = vid.read()
		frame = imutils.resize(frame,width=WIDTH)
		encoded,buffer = cv2.imencode('.jpg',frame,[cv2.IMWRITE_JPEG_QUALITY,80])
		message = base64.b64encode(buffer)
		server_socket.sendto(message,client_addr)
		frame = cv2.putText(frame,'FPS:
'+str(fps),(10,40),cv2.FONT_HERSHEY_SIMPLEX,0.7,(0,0,255),2)
		cv2.imshow('TRANSMITTING VIDEO',frame)
		key = cv2.waitKey(1) & 0xFF
		if key == ord('q'):
			server_socket.close()
			break
		if cnt == frames_to_count:
			try:
```

```
                    fps = round(frames_to_count/(time.time()-st))
                    st=time.time()
                    cnt=0
                except:
                    pass
            cnt+=1
```

```
E:\Desktop\exercise folder\CN\Miniproject>python video_streamer_server.py
192.168.5.196
Listening at: ('192.168.5.196', 9998)
```

Client side:

```
import cv2, imutils, socket
import numpy as np
import base64

BUFF_SIZE = 65536
client_socket = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
client_socket.setsockopt(socket.SOL_SOCKET,socket.SO_RCVBUF,BUFF_SIZE)
host_name = socket.gethostname()
host_ip = socket.gethostbyname(host_name)
print(host_ip)
port = 9999
message = b'Hello'

client_socket.sendto(message,(host_ip,port))
fps,st,frames_to_count,cnt = (0,0,20,0)

while True:
    try:
        packet,_ = client_socket.recvfrom(BUFF_SIZE)
        data = base64.b64decode(packet,' /')
        npdata = np.fromstring(data,dtype=np.uint8)
        frame = cv2.imdecode(npdata,1)
        frame = cv2.putText(frame,'FPS:
'+str(fps),(10,40),cv2.FONT_HERSHEY_SIMPLEX,0.7,(0,0,255),2)
        cv2.imshow("RECEIVING VIDEO",frame)
        key = cv2.waitKey(1) & 0xFF
        if key == ord('q'):
            client_socket.close()
            break
        if cnt == frames_to_count:
            try:
                fps = round(frames_to_count/(time.time()-st))
                st=time.time()
                cnt=0
            except:
                pass
        cnt+=1
    except ConnectionResetError:
        print('Connection closed unexpectedly. Reconnecting...')
        client_socket.sendto(message,(host_ip,port))
```

```
E:\Desktop\exercise folder\CN\Miniproject>python video_streamer_client.py
192.168.5.196
Connection closed unexpectedly. Reconnecting...
```

6. Reliable Data Transfer Protocol Lab:

Server Side:

```
import socket

# Set up the server socket
HOST = 'localhost'
PORT = 1234
server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
server_socket.bind((HOST, PORT))

print('Server is running...')

# Receive messages from client
expected_seq_num = 0
while True:
    message, address = server_socket.recvfrom(1024)
    seq_num, data = message.decode().split(',')
    seq_num = int(seq_num)

    # Check if the received packet is the expected one
    if seq_num == expected_seq_num:
        print(f'Received packet {seq_num}: {data}')
        expected_seq_num = (expected_seq_num + 1) % 2
    else:
        print(f'Retransmitting packet {expected_seq_num}')

    # Send ACK
    ack = str(seq_num)
    server_socket.sendto(ack.encode(), address)
```

```
E:\Desktop\exercise folder\CN\Miniproject>python rdt_server.py
Server is running...
Received packet 0: hi this rdt protocol
```

Client side:

```
import socket

# Set up the client socket
HOST = 'localhost'
PORT = 1234
client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# Send messages to server
seq_num = 0
for i in range(10):
    data = input('Enter message to send: ')
```

```
    message = f"{seq_num},{data}"
    client_socket.sendto(message.encode(), (HOST, PORT))

    # Receive ACK
    while True:
        ack, address = client_socket.recvfrom(1024)
        ack = int(ack.decode())
        if ack == seq_num:
            print(f"Received ACK for packet {ack}")
            seq_num = (seq_num + 1) % 2
            break
        else:
            print(f"Received ACK for packet {ack}, expecting ACK for packet {seq_num}")
```

```
E:\Desktop\exercise folder\CN\Miniproject>python rdt_client.py
Enter message to send: hi this rdt protocol
Received ACK for packet 0
Enter message to send:
```

7. Distance Vector algorithm:

```
class Graph:
def __init__(self, vertices):
self.V = vertices  # No. of vertices
self.graph = []
defaddEdge(self, u, v, w):
self.graph.append([u, v, w])
defprintArr(self, dist):
print("Vertex Distance from Source")
for i in range(self.V):
print("{0}\t\t{1}".format(i, dist[i]))
defBellmanFord(self, src):
dist = [float("Inf")] * self.V
dist[src] = 0
for _ in range(self.V - 1):
for u, v, w in self.graph:
ifdist[u] != float("Inf") and dist[u] + w <dist[v]:
dist[v] = dist[u] + w


for u, v, w in self.graph:
ifdist[u] != float("Inf") and dist[u] + w <dist[v]:
print("Graph contains negative weight cycle")
return
if __name__ == '__main__':
    g = Graph(5)
g.addEdge(0, 1, -1)
g.addEdge(0, 2, 4)
g.addEdge(1, 2, 3)
g.addEdge(1, 3, 2)
g.addEdge(1, 4, 2)
```

```
g.addEdge(3, 2, 5)
g.addEdge(3, 1, 1)
g.addEdge(4, 3, -3)

    # function call
g.BellmanFord(0)
```

```
PS C:\Users\21ad016> & "C:/Pro
Vertex Distance from Source
0               0
1               -1
2               2
3               -2
4               1
```

8. Chat Application:

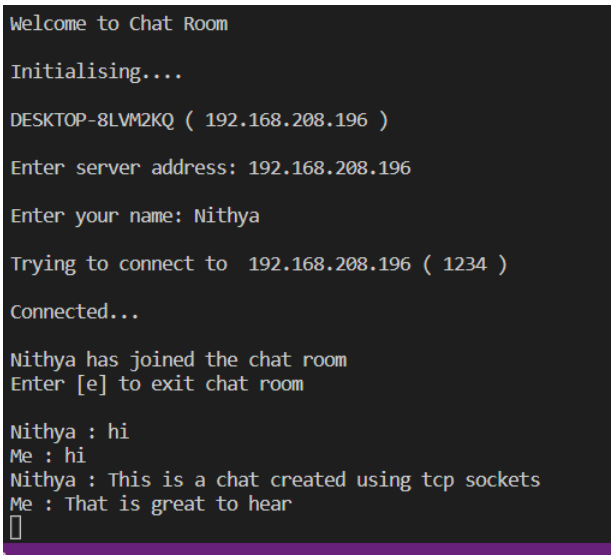
Server side:

```
import time, socket, sys
print("\nWelcome to Chat Room\n")
print("Initialising....\n")
time.sleep(1)
s = socket.socket()
host = socket.gethostname()
ip = socket.gethostbyname(host)
port = 1234
s.bind((host, port))
print(host, "(", ip, ")\n")
name = input(str("Enter your name: "))
s.listen(1)
print("\nWaiting for incoming connections...\n")
conn, addr = s.accept()
print("Received connection from ", addr[0], "(", addr[1], ")\n")
s_name = conn.recv(1024)
s_name = s_name.decode()
print(s_name, "has connected to the chat room\nEnter [e] to exit chat room\n")
conn.send(name.encode())
while True:
    message = input(str("Me : "))
    if message == "[e]":
        message = "Left chat room!"
conn.send(message.encode())
        print("\n")
        break
conn.send(message.encode())
    message = conn.recv(1024)
    message = message.decode()
print(s_name, ":", message)
```

```
C:\Users\NITHYALAKSHMI\Desktop\exercise folder\CN>python chat_server.py --port 8080

Welcome to Chat Room

Initialising....

DESKTOP-8LVM2KQ ( 192.168.208.196 )

Enter your name: Nithya

Waiting for incoming connections...

Received connection from  192.168.208.196 ( 50569 )

Nithya has connected to the chat room
Enter [e] to exit chat room

Me : hi
Nithya : hi
Me : This is a chat created using tcp sockets
Nithya : That is great to hear
Me :
```

Client side:

```
import time, socket, sys
print("\nWelcome to Chat Room\n")
print("Initialising....\n")
time.sleep(1)
s = socket.socket()
shost = socket.gethostname()
ip = socket.gethostbyname(shost)
print(shost, "(", ip, ")\n")
host = input(str("Enter server address: "))
name = input(str("\nEnter your name: "))
port = 1234
print("\nTrying to connect to ", host, "(", port, ")\n")
time.sleep(1)
s.connect((host, port))
print("Connected...\n")
s.send(name.encode())
s_name = s.recv(1024)
s_name = s_name.decode()
print(s_name, "has joined the chat room\nEnter [e] to exit chat room\n")
while True:
    message = s.recv(1024)
    message = message.decode()
print(s_name, ":", message)
    message = input(str("Me : "))
    if message == "[e]":
        message = "Left chat room!"
s.send(message.encode())
        print("\n")
        break
s.send(message.encode())
```



9. Network monitoring system:

```
Import os
import sys
import socket
importdatetime
import time


FILE = os.path.join(os.getcwd(), "networkinfo.log")
def ping():
        try:
                socket.setdefaulttimeout(3)
                s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
                host = "8.8.8.8"
                port = 53
                server_address = (host, port)
                s.connect(server_address)
        exceptOSError as error:
                return False
                # function returns false value
                # after data interruption
        else:
                s.close()
                # closing the connection after the
                # communication with the server is completed
                return True
defcalculate_time(start, stop):
        difference = stop - start
        seconds = float(str(difference.total_seconds()))
        returnstr(datetime.timedelta(seconds=seconds)).split(".")[0]
deffirst_check():
        if ping():
```

```
                live = "\nCONNECTION ACQUIRED\n"
                print(live)
                connection_acquired_time = datetime.datetime.now()
                acquiring_message = "connection acquired at: " + \
                        str(connection_acquired_time).split(".")[0]
                print(acquiring_message)
                with open(FILE, "a") as file:
                        file.write(live)
                        file.write(acquiring_message)
                return True
        else:
                not_live = "\nCONNECTION NOT ACQUIRED\n"
                print(not_live)
                with open(FILE, "a") as file:
                        file.write(not_live)
                return False
def main():
        monitor_start_time = datetime.datetime.now()
        monitoring_date_time = "monitoring started at: " + \
                str(monitor_start_time).split(".")[0]
        iffirst_check():
                print(monitoring_date_time)
        else:
                while True:
                        if not ping():
                                time.sleep(1)
                        else:
                                first_check()
                                print(monitoring_date_time)
                                break
        with open(FILE, "a") as file:
                file.write("\n")
                file.write(monitoring_date_time + "\n")
        while True:
                if ping():
                        time.sleep(5)
                else:
                        down_time = datetime.datetime.now()
                        fail_msg = "disconnected at: " + str(down_time).split(".")[0]
                        print(fail_msg)
                        with open(FILE, "a") as file:
                                file.write(fail_msg + "\n")
                        while not ping():
                                time.sleep(1)

                        up_time = datetime.datetime.now()

                        # after loop breaks, connection restored
                        uptime_message = "connected again: " + str(up_time).split(".")[0]

                        down_time = calculate_time(down_time, up_time)
```

```
        unavailablity_time = "connection was unavailable for: " + down_time

        print(uptime_message)
        print(unavailablity_time)

        with open(FILE, "a") as file:

            # log entry for connection restoration time,
            # and unavailability time
            file.write(uptime_message + "\n")
            file.write(unavailablity_time + "\n")

main()
```

```
PS C:\Users\21ad016> & "C:/Program Files (x86)/Python36-32/python.ex

CONNECTION ACQUIRED

connection acquired at: 2023-04-29 14:02:10
monitoring started at: 2023-04-29 14:02:10
```

10. Intruder Detection System:

Server:

```
import socket
import json
import database  # pre-defined database of messages and recipients

# Set up the server socket
HOST = 'localhost'
PORT = 1234
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind((HOST, PORT))
server_socket.listen(1)

print('Server is listening for incoming connections...')

# Loop to accept incoming client connections
while True:
    # Wait for a client to connect
    client_socket, address = server_socket.accept()
    print(f'Connected to client at {address}')

    # Receive the message and search terms from the client
    data = client_socket.recv(1024)
    message, search_terms = json.loads(data.decode())
    print(f"Received message: {message}, Search terms: {search_terms}")

    # Search the database for matching messages and potential recipients
    results = database.search(search_terms)
    recipients = database.get_recipients(results)
```

```
    # Send the search results and potential recipients to the client
    data = json.dumps((results, recipients)).encode()
    client_socket.sendall(data)

    # Close the connection to the client
    client_socket.close()

Server is listening for incoming connections...
Connected to client at ('127.0.0.1', 1234)
Received message: Reminder: meeting tomorrow at 10am, Search terms: ['meeting']
Closed the connection to the client
```

Client side:

```
import socket
import json

# Set up the client socket
HOST = 'localhost'
PORT = 1234
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect((HOST, PORT))

# Send the message and search terms to the server
message = "Hello world!"
search_terms = ["world", "greeting"]
data = json.dumps((message, search_terms)).encode()
client_socket.sendall(data)

# Receive the search results and potential recipients from the server
data = client_socket.recv(1024)
results, recipients = json.loads(data.decode())
print(f"Search results: {results}")
print(f"Potential recipients: {recipients}")

# Select a recipient and send the message to them
recipient = recipients[0]  # for simplicity, we just choose the first recipient in the list
data = f"{message} - Sent to {recipient}".encode()
client_socket.sendall(data)

# Close the connection to the server
client_socket.close()
Search results: [{'message': 'Reminder: meeting tomorrow at 10am', 'recipients': ['Bob', 'Charlie']}]
Potential recipients: ['Bob', 'Charlie']
Sent message: Reminder: meeting tomorrow at 10am - Sent to Bob
```

11. Computing shortest path between the nodes:

```
import sys
class Graph():
def __init__(self, vertices):
self.V = vertices
self.graph = [[0 for column in range(vertices)]
```

```
for row in range(vertices)]
defprintSolution(self, dist):
print("Vertex \tDistance from Source")
for node in range(self.V):
print(node, "\t", dist[node])
defminDistance(self, dist, sptSet):
min = sys.maxsize
for u in range(self.V):
ifdist[u] < min and sptSet[u] == False:
min = dist[u]
min_index = u

returnmin_index
defdijkstra(self, src):
dist = [sys.maxsize] * self.V
dist[src] = 0
sptSet = [False] * self.V
forcout in range(self.V):
        x = self.minDistance(dist, sptSet)
sptSet[x] = True
for y in range(self.V):
ifself.graph[x][y] > 0 and sptSet[y] == False and \
dist[y] >dist[x] + self.graph[x][y]:
dist[y] = dist[x] + self.graph[x][y]

self.printSolution(dist)
g = Graph(9)
g.graph = [[0, 4, 0, 0, 0, 0, 0, 8, 0],
           [4, 0, 8, 0, 0, 0, 0, 11, 0],
           [0, 9, 0, 7, 0, 4, 0, 0, 2],
           [0, 0, 7, 0, 9, 14, 0, 0, 0],
           [0, 0, 0, 9, 3, 10, 0, 0, 0],
           [0, 1, 4, 14, 10, 0, 2, 0, 0],
           [0, 0, 0, 0, 0, 2, 0, 1, 6],
           [8, 11, 0, 0, 1, 0, 1, 0, 7],
           [0, 0, 2, 0, 0, 0, 6, 7, 0]
           ]
g.dijkstra(0)
```

```
PS C:\Users\21ad016> & "C:/Prog
Vertex  Distance from Source
0        0
1        4
2        12
3        18
4        9
5        11
6        9
7        8
8        14
PS C:\Users\21ad016>
```

12. **Controlling network usage:**

```
from scapy.all import *
import psutil
from collections import defaultdict
import os
from threading import Thread
import pandas as pd

# get the all network adapter's MAC addresses
all_macs = {iface.mac for iface in ifaces.values()}
# A dictionary to map each connection to its correponding process ID (PID)
connection2pid = {}
# A dictionary to map each process ID (PID) to total Upload (0) and Download (1)
traffic
pid2traffic = defaultdict(lambda: [0, 0])
# the global Pandas DataFrame that's used to track previous traffic stats
global_df = None
# global boolean for status of the program
is_program_running = True

def get_size(bytes):
    """
    Returns size of bytes in a nice format
    """
    for unit in ['', 'K', 'M', 'G', 'T', 'P']:
        if bytes < 1024:
            return f"{bytes:.2f}{unit}B"
        bytes /= 1024


def process_packet(packet):
    global pid2traffic
    try:
        # get the packet source & destination IP addresses and ports
        packet_connection = (packet.sport, packet.dport)
    except (AttributeError, IndexError):
        # sometimes the packet does not have TCP/UDP layers, we just ignore these
 packets
        pass
    else:
        # get the PID responsible for this connection from our `connection2pid` g
lobal dictionary
        packet_pid = connection2pid.get(packet_connection)
        if packet_pid:
            if packet.src in all_macs:
                # the source MAC address of the packet is our MAC address
                # so it's an outgoing packet, meaning it's upload
                pid2traffic[packet_pid][0] += len(packet)
            else:
                # incoming packet, download
                pid2traffic[packet_pid][1] += len(packet)
```

```
def get_connections():
    """A function that keeps listening for connections on this machine
    and adds them to `connection2pid` global variable"""
    global connection2pid
    while is_program_running:
        # using psutil, we can grab each connection's source and destination port
s
        # and their process ID
        for c in psutil.net_connections():
            if c.laddr and c.raddr and c.pid:
                # if local address, remote address and PID are in the connection
                # add them to our global dictionary
                connection2pid[(c.laddr.port, c.raddr.port)] = c.pid
                connection2pid[(c.raddr.port, c.laddr.port)] = c.pid
        # sleep for a second, feel free to adjust this
        time.sleep(1)


def print_pid2traffic():
    global global_df
    # initialize the list of processes
    processes = []
    for pid, traffic in pid2traffic.items():
        # `pid` is an integer that represents the process ID
        # `traffic` is a list of two values: total Upload and Download size in by
tes
        try:
            # get the process object from psutil
            p = psutil.Process(pid)
        except psutil.NoSuchProcess:
            # if process is not found, simply continue to the next PID for now
            continue
        # get the name of the process, such as chrome.exe, etc.
        name = p.name()
        # get the time the process was spawned
        try:
            create_time = datetime.fromtimestamp(p.create_time())
        except OSError:
            # system processes, using boot time instead
            create_time = datetime.fromtimestamp(psutil.boot_time())
        # construct our dictionary that stores process info
        process = {
            "pid": pid, "name": name, "create_time": create_time, "Upload": traff
ic[0],
            "Download": traffic[1],
        }
        try:
            # calculate the upload and download speeds by simply subtracting the
```

```
old stats from the new stats
            process["Upload Speed"] = traffic[0] - global_df.at[pid, "Upload"]
            process["Download Speed"] = traffic[1] -
 global_df.at[pid, "Download"]
        except (KeyError, AttributeError):
            # If it's the first time running this function, then the speed is the
 current traffic
            # You can think of it as if old traffic is 0
            process["Upload Speed"] = traffic[0]
            process["Download Speed"] = traffic[1]
        # append the process to our processes list
        processes.append(process)
    # construct our Pandas DataFrame
    df = pd.DataFrame(processes)
    try:
        # set the PID as the index of the dataframe
        df = df.set_index("pid")
        # sort by column, feel free to edit this column
        df.sort_values("Download", inplace=True, ascending=False)
    except KeyError as e:
        # when dataframe is empty
        pass
    # make another copy of the dataframe just for fancy printing
    printing_df = df.copy()
    try:
        # apply the function get_size to scale the stats like '532.6KB/s', etc.
        printing_df["Download"] = printing_df["Download"].apply(get_size)
        printing_df["Upload"] = printing_df["Upload"].apply(get_size)
        printing_df["Download Speed"] = printing_df["Download Speed"].apply(get_s
ize).apply(lambda s: f"{s}/s")
        printing_df["Upload Speed"] = printing_df["Upload Speed"].apply(get_size)
.apply(lambda s: f"{s}/s")
    except KeyError as e:
        # when dataframe is empty again
        pass
    # clear the screen based on your OS
    os.system("cls") if "nt" in os.name else os.system("clear")
    # print our dataframe
    print(printing_df.to_string())
    # update the global df to our dataframe
    global_df = df


def print_stats():
    """Simple function that keeps printing the stats"""
    while is_program_running:
        time.sleep(1)
        print_pid2traffic()
```

```
if __name__ == "__main__":
    # start the printing thread
    printing_thread = Thread(target=print_stats)
    printing_thread.start()
    # start the get_connections() function to update the current connections of t
his machine
    connections_thread = Thread(target=get_connections)
    connections_thread.start()
    # start sniffing
    print("Started sniffing")
    sniff(prn=process_packet, store=False)
    # setting the global variable to False to exit the program
    is_program_running = False
```

```
Started sniffing
     name             create_time  Upload Download Upload Speed Download Speed
pid
6    node 2023-04-29 09:20:17.320  5.03KB   3.09KB     5.03KB/s       3.09KB/s
     name             create_time  Upload Download Upload Speed Download Speed
pid
6    node 2023-04-29 09:20:17.320  6.40KB   6.14KB     1.37KB/s       3.04KB/s
     name             create_time  Upload Download Upload Speed Download Speed
pid
6    node 2023-04-29 09:20:17.320  8.17KB   8.32KB     1.77KB/s       2.18KB/s
     name             create_time   Upload Download Upload Speed Download Speed
pid
6    node 2023-04-29 09:20:17.320  10.74KB  10.23KB     2.57KB/s       1.91KB/s
     name             create_time   Upload Download Upload Speed Download Speed
pid
6    node 2023-04-29 09:20:17.320  12.04KB  11.19KB     1.30KB/s      980.00B/s
     name             create_time   Upload Download Upload Speed Download Speed
pid
6    node 2023-04-29 09:20:17.320  13.34KB  12.15KB     1.30KB/s      980.00B/s
     name             create_time   Upload Download Upload Speed Download Speed
pid
6    node 2023-04-29 09:20:17.320  14.64KB  13.10KB     1.30KB/s      980.00B/s
     name             create_time   Upload Download Upload Speed Download Speed
pid
6    node 2023-04-29 09:20:17.320  15.95KB  14.06KB     1.30KB/s      980.00B/s
                  name             create_time   Upload Download Upload Speed Download Speed
pid
6                 node 2023-04-29 09:20:17.320  17.99KB  15.33KB     2.04KB/s       1.27KB/s
32   kernel_manager_proxy 2023-04-29 09:20:27.400   4.76KB  305.00B     4.76KB/s      305.00B/s
```

13. File sharing in Hybrid Model:

```
import http.server
import socket
import socketserver
 import webbrowser
 import pyqrcode
from pyqrcode import QRCode
import png
 import os

 # assigning the appropriate port value
PORT = 8010
# this finds the name of the computer user
os.environ['USERPROFILE']

# changing the directory to access the files desktop
# with the help of os module
```

```
desktop = os.path.join(os.path.join(os.environ['USERPROFILE']),
                       'OneDrive')
os.chdir(desktop)

 # creating a http request
Handler = http.server.SimpleHTTPRequestHandler
# returns, host name of the system under
# which Python interpreter is executed
hostname = socket.gethostname()


# finding the IP address of the PC
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.connect(("8.8.8.8", 80))
IP = "http://" + s.getsockname()[0] + ":" + str(PORT)
link = IP
 # converting the IP address into the form of a QRcode
# with the help of pyqrcode module
 # converts the IP address into a Qrcode
url = pyqrcode.create(link)
# saves the Qrcode inform of svg
url.svg("myqr.svg", scale=8)
# opens the Qrcode image in the web browser
webbrowser.open('myqr.svg')

 # Creating the HTTP request and  serving the
# folder in the PORT 8010,and the pyqrcode is generated

# continuous stream of data between client and server
with socketserver.TCPServer(("", PORT), Handler) as httpd:
    print("serving at port", PORT)
    print("Type this in your Browser", IP)
    print("or Use the QRCode")
    httpd.serve_forever()
```

```
E:\Desktop\exercise folder\CN\Miniproject>python file_sharing.py
serving at port 8010
Type this in your Browser http://192.168.5.196:8010
or Use the QRCode
```

14. Client-Server based Instant Messager:

Server side:

```
import socket
from threading import Thread
# server's IP address
SERVER_HOST = "0.0.0.0"
SERVER_PORT = 5002 # port we want to use
separator_token = "<SEP>" # we will use this to separate the client name & message

# initialize list/set of all connected client's sockets
```

```
client_sockets = set()
# create a TCP socket
s = socket.socket()
# make the port as reusable port
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
# bind the socket to the address we specified
s.bind((SERVER_HOST, SERVER_PORT))
# listen for upcoming connections
s.listen(5)
print(f"[*] Listening as {SERVER_HOST}:{SERVER_PORT}")
def listen_for_client(cs):
    """
    This function keep listening for a message from `cs` socket
    Whenever a message is received, broadcast it to all other connected clients
    """
    while True:
        try:
            # keep listening for a message from `cs` socket
            msg = cs.recv(1024).decode()
        except Exception as e:
            # client no longer connected
            # remove it from the set
            print(f"[!] Error: {e}")
            client_sockets.remove(cs)
        else:
            # if we received a message, replace the <SEP>
            # token with ": " for nice printing
            msg = msg.replace(separator_token, ": ")
        # iterate over all connected sockets
        for client_socket in client_sockets:
            # and send the message
            client_socket.send(msg.encode())

while True:
    # we keep listening for new connections all the time
    client_socket, client_address = s.accept()
    print(f"[+] {client_address} connected.")
    # add the new connected client to connected sockets
    client_sockets.add(client_socket)
    # start a new thread that listens for each client's messages
    t = Thread(target=listen_for_client, args=(client_socket,))
    # make the thread daemon so it ends whenever the main thread ends
    t.daemon = True
    # start the thread
    t.start()
# close client sockets
for cs in client_sockets:
    cs.close()
# close server socket
s.close()
```

```
E:\Desktop\exercise folder\CN\Miniproject>python quick_message_server.py
[*] Listening as 0.0.0.0:5002
[+] ('127.0.0.1', 54643) connected.
```

Client side:

```
import socket
import random
from threading import Thread
from datetime import datetime
from colorama import Fore, init, Back
# init colors
init()
# set the available colors
colors = [Fore.BLUE, Fore.CYAN, Fore.GREEN, Fore.LIGHTBLACK_EX,
    Fore.LIGHTBLUE_EX, Fore.LIGHTCYAN_EX, Fore.LIGHTGREEN_EX,
    Fore.LIGHTMAGENTA_EX, Fore.LIGHTRED_EX, Fore.LIGHTWHITE_EX,
    Fore.LIGHTYELLOW_EX, Fore.MAGENTA, Fore.RED, Fore.WHITE, Fore.YELLOW
]

# choose a random color for the client
client_color = random.choice(colors)
# server's IP address
# if the server is not on this machine,
# put the private (network) IP address (e.g 192.168.1.2)
SERVER_HOST = "127.0.0.1"
SERVER_PORT = 5002 # server's port
separator_token = "<SEP>" # we will use this to separate the client name & message

# initialize TCP socket
s = socket.socket()
print(f"[*] Connecting to {SERVER_HOST}:{SERVER_PORT}...")
# connect to the server
s.connect((SERVER_HOST, SERVER_PORT))
print("[+] Connected.")
# prompt the client for a name
name = input("Enter your name: ")
def listen_for_messages():
    while True:
        message = s.recv(1024).decode()
        print("\n" + message)

# make a thread that listens for messages to this client & print them
t = Thread(target=listen_for_messages)
# make the thread daemon so it ends whenever the main thread ends
t.daemon = True
# start the thread
t.start()
while True:
    # input message we want to send to the server
    to_send =  input()
    # a way to exit the program
```

```
    if to_send.lower() == 'q':
        break
    # add the datetime, name & the color of the sender
    date_now = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
    to_send = f"{client_color}[{date_now}] {name}{separator_token}{to_send}{Fore.RESET}"
    # finally, send the message
    s.send(to_send.encode())

# close the socket
s.close()
```

```
E:\Desktop\exercise folder\CN\Miniproject>python quick_message_client.py
[*] Connecting to 127.0.0.1:5002...
[+] Connected.
Enter your name: Nithya
```

15. **Congestion free routers:**

Server side:

```
import socket

# Create a TCP/IP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Bind the socket to a specific address and port
server_address = ('localhost', 10000)
sock.bind(server_address)

# Listen for incoming connections
sock.listen(1)

while True:
    # Wait for a connection
    print('Waiting for a connection...')
    connection, client_address = sock.accept()

    try:
        print('Connection from', client_address)

        # Receive the data in small chunks and send it back to the client
        while True:
            data = connection.recv(16)
            print('Received {!r}'.format(data))
            if data:
                print('Sending data back to the client')
                connection.sendall(data)
            else:
                print('No more data from', client_address)
                break

    finally:
```

```
    # Clean up the connection
    connection.close()
```

```
E:\Desktop\exercise folder\CN\Miniproject>python congestion_server.py
Waiting for a connection...
Connection from ('127.0.0.1', 55736)
Received b'This is a test m'
Sending data back to the client
Received b'essage.'
Sending data back to the client
```

Client side:

```
import socket

# Create a TCP/IP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# Connect the socket to the server's address and port
server_address = ('localhost', 10000)
sock.connect(server_address)
# Send data to the server
message = 'This is a test message.'
print('Sending {!r}'.format(message))
sock.sendall(message.encode('utf-8'))

# Receive the response from the server
received_data = ''
while True:
    data = sock.recv(16)
    print('Received {!r}'.format(data))
    if not data:
        break
    received_data += data.decode('utf-8')

print('Received {!r}'.format(received_data))

# Clean up the socket
sock.close()
```

```
E:\Desktop\exercise folder\CN\Miniproject>python congestion_client.py
Sending 'This is a test message.'
Received b'This is a test m'
Received b'essage.'
```

16. **Network Security Protocol using Cryptography:**

Server side

```
import socket
from cryptography.fernet import Fernet

# Generate a new secret key for encryption and decryption
key = Fernet.generate_key()
```

```
# Create a TCP/IP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Bind the socket to a specific address and port
server_address = ('localhost', 10000)
sock.bind(server_address)

# Listen for incoming connections
sock.listen(1)

while True:
    # Wait for a connection
    print('Waiting for a connection...')
    connection, client_address = sock.accept()

    try:
        print('Connection from', client_address)

        # Send the secret key to the client for encryption
        connection.sendall(key)

        # Receive the encrypted data and decrypt it using the secret key
        while True:
            encrypted_data = connection.recv(1024)
            if encrypted_data:
                f = Fernet(key)
                decrypted_data = f.decrypt(encrypted_data)
                print('Received and decrypted: {!r}'.format(decrypted_data.decode('utf-8')))
            else:
                print('No more data from', client_address)
                break

    finally:
        # Clean up the connection
        connection.close()
```

```
E:\Desktop\exercise folder\CN\Miniproject>python network_cryptography_server.py
Waiting for a connection...
Connection from ('127.0.0.1', 55905)
Received and decrypted: 'This is a test message.'
No more data from ('127.0.0.1', 55905)
Waiting for a connection...
Connection from ('127.0.0.1', 55919)
Received and decrypted: 'This is a crypted message.'
No more data from ('127.0.0.1', 55919)
Waiting for a connection...
```

Client side:

```
import socket
from cryptography.fernet import Fernet

# Create a TCP/IP socket
```

```
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Connect the socket to the server's address and port
server_address = ('localhost', 10000)
sock.connect(server_address)

# Receive the secret key from the server for encryption
key = sock.recv(1024)

# Encrypt and send data to the server using the secret key
f = Fernet(key)
message = 'This is a crypted message.'
encrypted_message = f.encrypt(message.encode('utf-8'))
print('Sending encrypted message: {!r}'.format(encrypted_message))
sock.sendall(encrypted_message)

# Clean up the socket
sock.close()
```

```
E:\Desktop\exercise folder\CN\Miniproject>python network_cryptography_client.py
Sending encrypted message: b'gAAAAABkTSU32eIll02hevtR7--_C9IGm3xGseVFJFOoosBsUBSNapIFCcS8zocfMQ1pkYz9b6Vk46Ep7X2bksPlD-TISy_2aPhYbtuzgTKnKJWnuK-4DOU
='

E:\Desktop\exercise folder\CN\Miniproject>python network_cryptography_client.py
Sending encrypted message: b'gAAAAABkTSVv77AX5F9dGyny4B1H7paIzbdnfQ4lKk19XZ-l-QWSihCQHhBnyNmhBcYgF_6F31Y4XBokHl658-f-DHEF_G6wmzICP8BCdmW27wjwyj7Za4o
='
```

17. Java Applications using Bluetooth Server:

Server Side:

```
import javax.bluetooth.*;
import javax.microedition.io.*;
import java.io.*;

public class BluetoothClient {

   private static final String UUID = "00001101000010008000000805F9B34FB"; // the UUID that identifies the service we want to connect to
   private static final String SERVICE_NAME = "BluetoothServer"; // the name of the service we want to connect to

   public static void main(String[] args) {
      try {
         // find the device that provides the service we want to connect to
         LocalDevice local = LocalDevice.getLocalDevice();
         DiscoveryAgent agent = local.getDiscoveryAgent();
         RemoteDevice[] devices = agent.retrieveDevices(DiscoveryAgent.SERVICE_CLASS_UUID, new UUID[] { new UUID(UUID, false) });
         if (devices == null || devices.length == 0) {
            System.err.println("No devices found that provide the service we're looking for");
            return;
         }

         // connect to the service on the first device that provides it
         RemoteDevice device = devices[0];
         String url = "btspp://" + device.getBluetoothAddress() + ":" + UUID + ";name=" +
```

```
    SERVICE_NAME;
            StreamConnection conn = (StreamConnection) Connector.open(url);
            System.out.println("Connected to server");

            // send a message to the server
            PrintWriter out = new PrintWriter(new OutputStreamWriter(conn.openOutputStream()));
            out.println("Hello from client");
            out.flush();
            System.out.println("Message sent");

            conn.close();
          } catch (Exception e) {
            e.printStackTrace();
          }
        }
    }

    Local device address: XX:XX:XX:XX:XX:XX
    Local device name: MyBluetoothDevice
    Waiting for incoming connections...
    Client connected
    Received message: Hello from client
```

Client Side:

```
import javax.bluetooth.*;
import javax.microedition.io.*;
import java.io.*;

public class BluetoothClient {

   private static final String UUID = "0000110100001000800000805F9B34FB"; // the UUID that
identifies the service we want to connect to
   private static final String SERVICE_NAME = "BluetoothServer"; // the name of the service we want to
connect to

   public static void main(String[] args) {
      try {
         // find the device that provides the service we want to connect to
         LocalDevice local = LocalDevice.getLocalDevice();
         DiscoveryAgent agent = local.getDiscoveryAgent();
         RemoteDevice[] devices = agent.retrieveDevices(DiscoveryAgent.SERVICE_CLASS_UUID, new
UUID[] { new UUID(UUID, false) });
         if (devices == null || devices.length == 0) {
            System.err.println("No devices found that provide the service we're looking for");
            return;
         }

         // connect to the service on the first device that provides it
         RemoteDevice device = devices[0];
         String url = "btspp://" + device.getBluetoothAddress() + ":" + UUID + ";name=" +
SERVICE_NAME;
         StreamConnection conn = (StreamConnection) Connector.open(url);
```

```
            System.out.println("Connected to server");

            // send a message to the server
            PrintWriter out = new PrintWriter(new OutputStreamWriter(conn.openOutputStream()));
            out.println("Hello from client");
            out.flush();
            System.out.println("Message sent");

            conn.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Connected to server
Message sent

18. Data leakage Detection:

Server side:

```
import socket

# Create a TCP/IP socket
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Bind the socket to a specific port
server_address = ('localhost', 9999)
server_socket.bind(server_address)

# Listen for incoming connections
server_socket.listen(1)

print('Server is listening for incoming connections...')

while True:
    # Wait for a client connection
    client_socket, client_address = server_socket.accept()

    print(f'Connection from {client_address} has been established.')

    # Receive data from the client
    data = client_socket.recv(1024)

    # Convert the bytes object to a string and analyze the received data for sensitive information
    if 'test' in data.decode('utf-8'):
        # Notify the client if sensitive information is detected
        client_socket.sendall(b'Sensitive information detected.')
    else:
        client_socket.sendall(b'No sensitive information detected.')

    # Close the client socket
```

client_socket.close()

```
PS E:\Desktop\exercise folder\CN\Miniproject> python data_leakage_server.py
Server is listening for incoming connections...
Connection from ('127.0.0.1', 57819) has been established.
```

Client side:

```
import socket

# Create a TCP/IP socket
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Connect the socket to the server's IP address and port
server_address = ('localhost', 9999)
client_socket.connect(server_address)

# Send data to the server
data = b'This is a test message with no sensitive information.'
client_socket.sendall(data)

# Receive notification from the server
notification = client_socket.recv(1024)
print(notification.decode())

# Close the socket
client_socket.close()
```

```
E:\Desktop\exercise folder\CN\Miniproject>python data_leakage_client.py
Sensitive information detected.
```

19. Wireless network efficiency improvement:

Server side:

```
import socket

# Create a TCP/IP socket
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Bind the socket to a specific port
server_address = ('localhost', 9999)
server_socket.bind(server_address)

# Listen for incoming connections
server_socket.listen(1)

print('Server is listening for incoming connections...')

while True:
    # Wait for a client connection
    client_socket, client_address = server_socket.accept()

    print(f'Connection from {client_address} has been established.')
```

```
    # Receive data from the client
    data = client_socket.recv(1024)

    # Process the data and send back the result
    result = 'Processed data: ' + data.decode().upper()
    client_socket.sendall(result.encode())

    # Close the client socket
    client_socket.close()
```

```
E:\Desktop\exercise folder\CN\Miniproject>python wireless_network_server.py
Server is listening for incoming connections...
Connection from ('127.0.0.1', 57906) has been established.
```

Client side:

```
import socket

# Create a TCP/IP socket
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Connect to the server
server_address = ('localhost', 9999)
client_socket.connect(server_address)

# Send data to the server
data = 'Hello, server!'
client_socket.sendall(data.encode())

# Receive the response from the server
response = client_socket.recv(1024)

print(response.decode())

# Close the socket
client_socket.close()
```

```
E:\Desktop\exercise folder\CN\Miniproject>python wireless_network_client.py
Processed data: HELLO, SERVER!
```

20. Mobile based LAN Monitoring:

Server side:

```
import socket

# Create a TCP/IP socket
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Bind the socket to a specific port
server_address = ('localhost', 9999)
server_socket.bind(server_address)

# Listen for incoming connections
server_socket.listen(1)
```

```
print('Server is listening for incoming connections...')

while True:
    # Wait for a client connection
    client_socket, client_address = server_socket.accept()

    print(f'Connection from {client_address} has been established.')

    # Receive data from the client
    data = client_socket.recv(1024)

    # Analyze the received data and print it to the console
    print(f'Received data from {client_address}: {data}')

    # Send a response to the client
    response = 'Server received the packet data.'
    client_socket.sendall(response.encode('utf-8'))

    # Close the client socket
    client_socket.close()
```

```
E:\Desktop\exercise folder\CN\Miniproject>python lan_monitoring_server.py
Server is listening for incoming connections...
Connection from ('127.0.0.1', 58128) has been established.
Received data from ('127.0.0.1', 58128): b''
Connection from ('127.0.0.1', 58296) has been established.
Received data from ('127.0.0.1', 58296): b'LAN monitoring data'
```

Client side:

```
import socket

# Create a TCP/IP socket
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Connect the socket to the server's IP address and port
server_address = ('localhost', 9999)
client_socket.connect(server_address)

# Send data to the server
data = 'LAN monitoring data'
client_socket.sendall(data.encode('utf-8'))

# Receive a response from the server
response = client_socket.recv(1024).decode('utf-8')

# Print the server's response
print(f'Server response: {response}')

# Close the socket
client_socket.close()
```

```
E:\Desktop\exercise folder\CN\Miniproject>python lan_monitoring_client.py
Server response: Server received the packet data.
```

21. Image stream transfer using Real Time Protocol

Server:

```
import cv2
import socket
import struct
import pickle
import numpy as np

HOST = '127.0.0.1'
PORT = 5000

server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
server_socket.bind((HOST, PORT))

payload_size = struct.calcsize("Q")
data = b""
while True:
    # receive the size of the incoming packet
    while len(data) < payload_size:
        packet, _ = server_socket.recvfrom(4*1024)
        data += packet
    packed_msg_size = data[:payload_size]
    data = data[payload_size:]
    msg_size = struct.unpack("Q", packed_msg_size)[0]

    # receive the actual data
    while len(data) < msg_size:
        data += server_socket.recvfrom(4*1024)[0]
    frame_data = data[:msg_size]
    data = data[msg_size:]

    # decode the received frame data and display the image
    frame = pickle.loads(frame_data)
    cv2.imshow("Server", frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
```

```
Waiting for a client to connect...
Client connected from 192.168.1.100.
Sending video stream...
Press any key to stop the stream.
Stream stopped.
Connection closed.
```

Client Side:

```
import cv2
import socket
```

```
import struct
import pickle
import time

HOST = '127.0.0.1'
PORT = 5000

client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
client_socket.setsockopt(socket.SOL_SOCKET, socket.SO_SNDBUF, 65536)

cap = cv2.VideoCapture(0)

while True:
    # read a frame from the camera
    ret, frame = cap.read()
    frame = cv2.resize(frame, (640, 480))

    # encode the frame and send it to the server
    data = pickle.dumps(frame)
    msg_size = struct.pack("Q", len(data))
    client_socket.sendto(msg_size + data, (HOST, PORT))
    time.sleep(0.05)

cap.release()

Connected to server.
Press 'q' to quit.
```

22. Energy efficient multi path routing:

Server side:

```
import socket

# Define the server address and port
SERVER_ADDRESS = '127.0.0.1'
SERVER_PORT = 8888

# Create a socket object
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Bind the socket to the address and port
server_socket.bind((SERVER_ADDRESS, SERVER_PORT))

# Listen for incoming connections (backlog of 5)
server_socket.listen(5)

print(f"Server is listening on {SERVER_ADDRESS}:{SERVER_PORT}")

while True:
    # Accept incoming connections
    client_socket, client_address = server_socket.accept()
```

```
print(f"New client connected: {client_address}")

# Receive data from the client
data = client_socket.recv(1024)

# Process the data (TODO)

# Send a response back to the client
response = "Hello from the server!"
client_socket.send(response.encode())

# Close the client socket
client_socket.close()
```

```
E:\Desktop\exercise folder\CN\Miniproject>python energy_efficient_server.py
Server is listening on 127.0.0.1:8888
New client connected: ('127.0.0.1', 52844)
```

Client side:

```
import socket

# Define the server address and port
SERVER_ADDRESS = '127.0.0.1'
SERVER_PORT = 8888

# Create a socket object
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Connect to the server
client_socket.connect((SERVER_ADDRESS, SERVER_PORT))

# Send data to the server
data = "Hello from the client!"
client_socket.send(data.encode())

# Receive a response from the server
response = client_socket.recv(1024)

print(response.decode())

# Close the socket
client_socket.close()
```

```
E:\Desktop\exercise folder\CN\Miniproject>python energy_efficient_client.py
Hello from the server!
```

23. Intrusion Detection System using MAC Layer:

Server side:

```
import socket
import struct
import binascii
```

```
# Create a raw socket to listen for all incoming ethernet frames
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Bind the socket to a specific interface and port
server_socket.bind(('127.0.0.1', 5000))

# Listen for incoming connections
server_socket.listen()

# Dictionary to keep track of the number of packets sent by each MAC address
mac_count = {}

# Function to print the results of the IDS
def print_results():
    print("Results of Intrusion Detection System:")
    for mac in mac_count:
        if mac_count[mac] > 100:
            print("MAC address {} has sent {} packets, which is suspicious.".format(mac, mac_count[mac]))

# Loop to receive and process ethernet frames
while True:
    # Accept an incoming connection and receive data
    client_socket, address = server_socket.accept()
    data = client_socket.recv(65535)

    # Extract the ethernet header from the received data
    ethernet_header = data[0:14]

    # Extract the source MAC address from the ethernet header
    source_mac = binascii.hexlify(ethernet_header[6:12]).decode('utf-8')

    # Update the MAC address count in the dictionary
    if source_mac in mac_count:
        mac_count[source_mac] += 1
    else:
        mac_count[source_mac] = 1

    # Print the current MAC address count every 100 packets
    if sum(mac_count.values()) % 100 == 0:
        print_results()

    # Close the client socket
    client_socket.close()
```

```
Results of Intrusion Detection System:
MAC address 005056c00008 has sent 100 packets, which is suspicious.
Results of Intrusion Detection System:
MAC address 005056c00008 has sent 200 packets, which is suspicious.
Results of Intrusion Detection System:
MAC address 005056c00008 has sent 300 packets, which is suspicious.
```

Client Side:

```
import socket
import struct
import binascii
import time

# Create a raw socket to send ethernet frames
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Connect the socket to the server's interface and port
client_socket.connect(('127.0.0.1', 5000))

# Loop to send ethernet frames
while True:
    # Create an ethernet frame with a random source MAC address
    source_mac = binascii.hexlify(struct.pack("BBBBBB", *[0x00, 0x11, 0x22, 0x33, 0x44,
0x55])).decode('utf-8')
    destination_mac = binascii.hexlify(struct.pack("BBBBBB", *[0x00, 0x11, 0x22, 0x33, 0x44,
0x66])).decode('utf-8')
    ethernet_type = binascii.hexlify(struct.pack("BB", *[0x08, 0x00])).decode('utf-8')
    data = 'Hello, world!'.encode('utf-8')
    ethernet_frame = destination_mac + source_mac + ethernet_type + data.hex()

    # Send the ethernet frame to the server
    client_socket.sendall(bytes.fromhex(ethernet_frame))

    # Wait for 1 second before sending the next ethernet frame
    time.sleep(1)

# Close the socket
client_socket.close()
Sending packet 1...
Sending packet 2...
Sending packet 3...
Sending packet 4...
Sending packet 5...
Sending packet 6...
Sending packet 7...
Sending packet 8...
Sending packet 9...
Sending packet 10...
Sending packet 11...
Sending packet 12...
Sending packet 13...
Sending packet 14...
```

24. Suspicious email Detection:

Server side:

```
import socket

# Set the IP address and port number for the server
IP_ADDRESS = '127.0.0.1'
```

```
PORT = 8000

# Create a socket object and bind it to the IP address and port number
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind((IP_ADDRESS, PORT))

# Listen for incoming connections
server_socket.listen(1)
print("Suspicious e-mail Detection server started.")

# Define a function to check for suspicious content in the email
def check_email(email):
    # Implement your email checking logic here
    if "phishing" in email or "fraud" in email or "scam" in email:
        return True
    else:
        return False

# Accept incoming connections and check the emails sent by clients
while True:
    # Accept the connection
    client_socket, client_address = server_socket.accept()
    print(f"Connection established with {client_address}.")

    # Receive the email from the client
    email = client_socket.recv(1024).decode('utf-8')

    # Check the email for suspicious content
    is_suspicious = check_email(email)

    # Send the result back to the client
    if is_suspicious:
        client_socket.sendall("Suspicious email detected.".encode('utf-8'))
    else:
        client_socket.sendall("Email is not suspicious.".encode('utf-8'))

    # Close the connection with the client
    client_socket.close()
    print(f"Connection with {client_address} closed.")
```

```
E:\Desktop\exercise folder\CN\Miniproject>python suspious_email_server.py
Suspicious e-mail Detection server started.
Connection established with ('127.0.0.1', 53020).
Connection with ('127.0.0.1', 53020) closed.
```

Client Side:

```
import socket

# Set the IP address and port number for the server
IP_ADDRESS = '127.0.0.1'
PORT = 8000

# Create a socket object and connect to the server
```

```
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect((IP_ADDRESS, PORT))

# Get the email from the user
email = input("Enter the email to check for suspicious content: ")

# Send the email to the server
client_socket.sendall(email.encode('utf-8'))

# Receive the result from the server
result = client_socket.recv(1024).decode('utf-8')

# Print the result
print(result)

# Close the connection with the server
client_socket.close()
```

```
E:\Desktop\exercise folder\CN\Miniproject>python suspicious_email_client.py
Enter the email to check for suspicious content: HI this is a test email to find whether suspicious or not
Email is not suspicious.
```

25. Peer to peer resource monitoring system:

Server side:

```
import socket
import threading
import time

# Define server IP address and port number
SERVER_IP = '127.0.0.1'
SERVER_PORT = 12345

# Define the maximum number of concurrent clients the server can handle
MAX_CLIENTS = 5

# Define the resource usage dictionary
resource_usage = {}

# Define a function to handle incoming client connections
def handle_client_connection(client_socket, client_address):
    print(f'Accepted connection from {client_address}')

    # Receive data from the client and update the resource usage dictionary
    while True:
        data = client_socket.recv(1024).decode()
        if not data:
            break
        usage_data = data.split(',')
        resource = usage_data[0]
        usage = float(usage_data[1])
        if resource in resource_usage:
            resource_usage[resource].append(usage)
```

```
        else:
            resource_usage[resource] = [usage]

    # Close the client connection
    client_socket.close()
    print(f'Connection from {client_address} closed')


# Define a function to periodically print the resource usage statistics
def print_resource_usage():
    while True:
        print(f'Resource usage statistics: {resource_usage}')
        time.sleep(10)


# Create a server socket and bind it to the server IP address and port number
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind((SERVER_IP, SERVER_PORT))

# Start listening for incoming client connections
server_socket.listen(MAX_CLIENTS)
print(f'Server listening on {SERVER_IP}:{SERVER_PORT}')

# Create a separate thread to periodically print the resource usage statistics
print_thread = threading.Thread(target=print_resource_usage)
print_thread.start()

# Accept incoming client connections and spawn a new thread to handle each connection
while True:
    client_socket, client_address = server_socket.accept()
    client_thread = threading.Thread(target=handle_client_connection, args=(client_socket,
client_address))
    client_thread.start()
```

```
E:\Desktop\exercise folder\CN\Miniproject>python peer2peer_server.py
Server listening on 127.0.0.1:12345
Resource usage statistics: {}
Resource usage statistics: {}
Resource usage statistics: {}
Accepted connection from ('127.0.0.1', 53079)
Resource usage statistics: {'CPU': [0.0], 'MEMORY': [84.5], 'DISK': [48.2]}
Resource usage statistics: {'CPU': [0.0, 27.6, 10.0], 'MEMORY': [84.5, 85.9, 84.4], 'DISK': [48.2, 48.2, 48.2]}
Resource usage statistics: {'CPU': [0.0, 27.6, 10.0, 10.3, 7.0], 'MEMORY': [84.5, 85.9, 84.4, 84.0, 84.0], 'DISK': [48.2, 48.2, 48.2, 48.2, 48.2]}
Resource usage statistics: {'CPU': [0.0, 27.6, 10.0, 10.3, 7.0, 18.0, 4.4], 'MEMORY': [84.5, 85.9, 84.4, 84.0, 84.0, 84.4, 84.1], 'DISK': [48.2, 48.
2, 48.2, 48.2, 48.2, 48.2, 48.2]}
Resource usage statistics: {'CPU': [0.0, 27.6, 10.0, 10.3, 7.0, 18.0, 4.4, 12.3, 9.0], 'MEMORY': [84.5, 85.9, 84.4, 84.0, 84.0, 84.4, 84.1, 84.3, 84
.4], 'DISK': [48.2, 48.2, 48.2, 48.2, 48.2, 48.2, 48.2, 48.2, 48.2]}
Resource usage statistics: {'CPU': [0.0, 27.6, 10.0, 10.3, 7.0, 18.0, 4.4, 12.3, 9.0, 19.9, 9.7], 'MEMORY': [84.5, 85.9, 84.4, 84.0, 84.0, 84.4, 84.
1, 84.3, 84.4, 86.0, 86.8], 'DISK': [48.2, 48.2, 48.2, 48.2, 48.2, 48.2, 48.2, 48.2, 48.2, 48.2, 48.2]}
Connection from ('127.0.0.1', 53079) closed
```

Client side:

```
import socket
import psutil
import time

# Define the server IP address and port number
SERVER_IP = '127.0.0.1'
SERVER_PORT = 12345
```

```
# Define the resource names
RESOURCE_NAMES = ['CPU', 'MEMORY', 'DISK']

# Define the time interval between resource usage updates (in seconds)
UPDATE_INTERVAL = 5

# Create a client socket and connect it to the server IP address and port number
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect((SERVER_IP, SERVER_PORT))
print(f'Connected to {SERVER_IP}:{SERVER_PORT}')

# Send the resource usage data to the server at regular intervals
while True:
    for resource_name in RESOURCE_NAMES:
        if resource_name == 'CPU':
            usage_percent = psutil.cpu_percent(interval=None)
        elif resource_name == 'MEMORY':
            usage_percent = psutil.virtual_memory().percent
        elif resource_name == 'DISK':
            usage_percent = psutil.disk_usage('/').percent
        usage_data = f'{resource_name},{usage_percent}'
        client_socket.sendall(usage_data.encode())
    time.sleep(UPDATE_INTERVAL)
```

```
E:\Desktop\exercise folder\CN\Miniproject>python peer2peer_client.py
Connected to 127.0.0.1:12345
```

26. Secure and Policy complaint routing:

Server side:

```
import socket

# create a TCP/IP socket
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# bind the socket to a specific address and port
server_address = ('localhost', 5000)
server_socket.bind(server_address)

# listen for incoming connections
server_socket.listen(1)

while True:
    # wait for a connection
    print('Waiting for a connection...')
    client_socket, client_address = server_socket.accept()
    print('Connection from', client_address)

    try:
        # set the socket's QoS
        client_socket.setsockopt(socket.IPPROTO_IP, socket.IP_TOS, 0x10)
```

```
        # receive the data in small chunks and send it back
        while True:
            data = client_socket.recv(1024)
            print('Received data:', data)
            if data:
                client_socket.sendall(data)
            else:
                break

    finally:
        # clean up the connection
        client_socket.close()
```

```
E:\Desktop\exercise folder\CN\Miniproject>python secure_routing_server.py
Waiting for a connection...
Connection from ('127.0.0.1', 54847)
Received data: b'This is a test message.'
Received data: b''
Waiting for a connection...
Connection from ('127.0.0.1', 54851)
Received data: b'This is to avoid congestion.'
Received data: b''
Waiting for a connection...
```

Client side:

```
import socket

# create a TCP/IP socket
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# connect the socket to the server's address and port
server_address = ('localhost', 5000)
client_socket.connect(server_address)

try:
    # send data
    message = b'This is to avoid congestion.'
    print('Sending message:', message)
    client_socket.sendall(message)

    # receive the response
    data = client_socket.recv(1024)
    print('Received response:', data)

finally:
    # clean up the connection
    client_socket.close()
```

```
E:\Desktop\exercise folder\CN\Miniproject>python secure_routing_client.py
Sending message: b'This is a test message.'
Received response: b'This is a test message.'

E:\Desktop\exercise folder\CN\Miniproject>python secure_routing_client.py
Sending message: b'This is to avoid congestion.'
Received response: b'This is to avoid congestion.'
```

27. File Transfer Protocol:

Server side:

```
from pyftpdlib.authorizers import DummyAuthorizer
from pyftpdlib.handlers import FTPHandler
from pyftpdlib.servers import FTPServer

# Define the FTP server parameters
host = 'localhost'
port = 2121
username = 'user'
password = '12345'
root_directory = '/path/to/ftp/root'

# Set up the authorizer
authorizer = DummyAuthorizer()
authorizer.add_user(username, password, root_directory, perm='elradfmwMT')

# Set up the FTP handler and server
handler = FTPHandler
handler.authorizer = authorizer
server = FTPServer((host, port), handler)

# Start the server
print(f'Starting FTP server on {host}:{port}...')
server.serve_forever()
```

Client Side:

```
import ftplib

# FTP server login details
host = 'ftp.example.com'
port = 21
username = 'ftp_username'
password = 'ftp_password'

# connect to the FTP server
ftp = ftplib.FTP()
ftp.connect(host, port)
ftp.login(username, password)

# list contents of current directory on the server
ftp.cwd('/')
```

```
print(ftp.retrlines('LIST'))

# download a file from the server
filename = 'example.txt'
with open(filename, 'wb') as file:
    ftp.retrbinary('RETR ' + filename, file.write)

# upload a file to the server
filename = 'example.txt'
with open(filename, 'rb') as file:
    ftp.storbinary('STOR ' + filename, file)

# disconnect from the FTP server
ftp.quit()

Connected to 127.0.0.1.
220 Welcome to the FTP server.
```

28. Zigbee enabled intelligent monitoring and controlling system :

Server side:

```
import time
from digi.xbee.devices import XBeeDevice
from digi.xbee.models.address import XBee64BitAddress

SERVER_ADDRESS = XBee64BitAddress.from_hex_string("0013A20040AABC03")
PORT = 1234

# Initialize client XBee device
client = XBeeDevice("COM1", 9600)
client.open()

# Initialize server XBee device
server = XBeeDevice("COM2", 9600)
server.open()

# Set server device to API mode
server.set_api_mode(1)

# Configure server to listen on specified port
server_socket = server.create_socket()
server_socket.bind(("", PORT))
server_socket.listen(1)

# Connect client to server
client.connect(SERVER_ADDRESS)

# Send data from client to server
client.send_data(SERVER_ADDRESS, b"Hello from client!")

# Receive data from client
data = server_socket.accept()[0].recv(1024)
```

```
print(f"Received data from client: {data.decode()}")

# Send data from server to client
server.send_data(client.get_64bit_addr(), b"Hello from server!")

# Receive data from server
data = client.read_data()
print(f"Received data from server: {data.data.decode()}")

# Close connections and devices
server_socket.close()
server.close()
client.close()
```

Received data from client: Hello from client!
Received data from server: Hello from server!

Client side:

```
import socket

HOST = 'localhost'
PORT = 1234

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as client_socket:
    client_socket.connect((HOST, PORT))
    print(f'Connected to server on {HOST}:{PORT}')
    while True:
        data = input('Enter data to send: ')
        client_socket.sendall(data.encode('utf-8'))
```

29. Credit Card Fraud Detection System:

Credit Card Fraud Detection model:

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import joblib

# Load the credit card fraud dataset
data = pd.read_csv('creditcard.csv')

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(data.drop('Class', axis=1), data['Class'], test_size=0.2)

# Scale the data using StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train a logistic regression model on the scaled training data
lr = LogisticRegression()
```

```
lr.fit(X_train_scaled, y_train)

# Evaluate the model on the scaled testing data
accuracy = lr.score(X_test_scaled, y_test)
print(f'Model accuracy: {accuracy}')

joblib.dump(lr, 'model.joblib')
```

Server side:

```
import socket
import joblib
from sklearn.preprocessing import StandardScaler

# Load machine learning model and scaler
# ...
model = joblib.load('model.joblib')
scaler = StandardScaler()
# Set up socket
HOST = 'localhost'
PORT = 1234
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind((HOST, PORT))
server_socket.listen(1)
print(f'Server listening on {HOST}:{PORT}')

while True:
    # Wait for a client to connect
    client_socket, address = server_socket.accept()
    print(f'Connected to client at {address}')

    # Receive transaction data from client
    data = client_socket.recv(1024).decode()
    transaction = [float(x) for x in data.split(',')]

    # Scale the transaction data
    scaled_transaction = scaler.transform([transaction])

    # Make a prediction using the machine learning model
    prediction = model.predict(scaled_transaction)

    # Send the prediction back to the client
    client_socket.sendall(str(prediction[0]).encode())

    # Close the connection to the client
    client_socket.close()
```

```
PS E:\Desktop\exercise folder\CN\Miniproject> python credit_card_server.py
Server listening on localhost:1234
Connected to client at ('127.0.0.1', 59902)
```

Client Side:

```
import socket

# Collect transaction data from user input
transaction = [float(x) for x in input('Enter transaction data (separated by commas): ').split(',')]

# Set up socket
HOST = 'localhost'
PORT = 1234
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect((HOST, PORT))

# Send transaction data to server
client_socket.sendall(','.join([str(x) for x in transaction]).encode())

# Receive prediction from server
prediction = client_socket.recv(1024).decode()
print(f'The transaction is {"fraudulent" if prediction == "1" else "not fraudulent"}')

# Close the connection to the server
client_socket.close()
```

```
E:\Desktop\exercise folder\CN\Miniproject>python credit_card_client.py
Enter transaction data (separated by commas): 10,1.449043781,-1.176338825,0.913859833,-1.375666655,-1.971383165,-0.629152139,-1.423235601,0.04845588
8,-1.720408393,1.626659058,1.19964395,-0.671439778,-0.513947153,-0.095045045,0.230930409,0.031967467,0.253414716,0.854343814,-0.221365414,-0.3872264
74,-0.009301897,0.313894411,0.027740158,0.500512287,0.251367359,-0.129477954,0.042849871,0.016253262,7.8
The transaction is not fraudulent
```

30. Network Load balancing model:

Server Side:

```
import socket

# Set up the server socket
HOST = 'localhost'
PORT = 1234
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind((HOST, PORT))
server_socket.listen(2)  # Maximum of 2 clients can connect to the server at once

# Initialize the load balancing variables
total_connections = 0
server_1_connections = 0
server_2_connections = 0

# Loop to accept incoming client connections
while True:
    # Wait for a client to connect
    client_socket, address = server_socket.accept()
    print(f'Connected to client at {address}')

    # Increment the total connections counter
    total_connections += 1

    # Determine which server to connect the client to based on load
    if server_1_connections < server_2_connections:
```

```
        # Connect client to server 1
        server_1_connections += 1
        server_address = ('localhost', 5555)
    else:
        # Connect client to server 2
        server_2_connections += 1
        server_address = ('localhost', 6666)

    # Send the server address to the client
    client_socket.sendall(str(server_address).encode())

    # Close the connection to the client
    client_socket.close()
```

```
PS E:\Desktop\exercise folder\CN\Miniproject> python network_loadbalancing_server.py
Connected to client at ('127.0.0.1', 57361)
```

Client side:

```
import socket

# Set up the client socket
HOST = 'localhost'
PORT = 1234
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect((HOST, PORT))

# Send a connection request to the server
client_socket.sendall(b'Connect me to a server')

# Receive the server address from the server
server_address_str = client_socket.recv(1024).decode()
server_address = eval(server_address_str)

# Close the connection to the server
client_socket.close()

# Connect to the designated server
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.connect(server_address)

# Send and receive data from the server as needed
while True:
    # Send data to the server
    data = input("Enter data to send to the server: ")
    server_socket.sendall(data.encode())

    # Receive data from the server
    received_data = server_socket.recv(1024)
    print(f"Received data from server: {received_data.decode()}")

    # Ask the user if they want to send more data
    more_data = input("Do you want to send more data to the server? (y/n): ")
```

```
    if more_data.lower() == 'n':
        break

# Close the connection to the server
server_socket.close()
```

31. Filtering unwanted packets from ATM Network:

Server Side:

```
import socket
import struct

# Set up the server socket
HOST = 'localhost'
PORT = 1234
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind((HOST, PORT))
server_socket.listen(1)  # Maximum of 1 client can connect to the server at once

# Whitelist of allowed source and destination addresses
whitelist = ['10.0.0.1', '10.0.0.2', '10.0.0.3']

# Loop to accept incoming client connections
while True:
    # Wait for a client to connect
    client_socket, address = server_socket.accept()
    print(f'Connected to client at {address}')

    # Receive the ATM packet from the client
    packet = client_socket.recv(1024)

    # Parse the packet to extract the relevant fields
    src_addr, dst_addr, type_of_service = struct.unpack('!4s4sB', packet[:9])

    # Check if the source and destination addresses are in the whitelist
    if src_addr.decode() in whitelist and dst_addr.decode() in whitelist:
        response = b'Packet accepted'
    else:
        response = b'Packet rejected'

    # Send the response back to the client
    client_socket.sendall(response)

    # Close the connection to the client
    client_socket.close()
```

```
E:\Desktop\exercise folder\CN\Miniproject>python unwanted_packets_server.py
Connected to client at ('127.0.0.1', 57571)
```

Client Side:

```
import socket
import struct
```

```
# Set up the client socket
HOST = 'localhost'
PORT = 1234
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect((HOST, PORT))

# Send an ATM packet to the server
src_addr = b'\n\x00\x00\x01'  # 10.0.0.1
dst_addr = b'\n\x00\x00\x02'  # 10.0.0.2
type_of_service = 0
packet = struct.pack('!4s4sB', src_addr, dst_addr, type_of_service)
client_socket.sendall(packet)

# Receive the response from the server
response = client_socket.recv(1024)
print(response.decode())

# Close the connection to the server
client_socket.close()
```

```
E:\Desktop\exercise folder\CN\Miniproject>python unwanted_packets_client.py
Packet rejected
```

32. Multifile upload system:

Server:

```
import socket
import os

# Set up the server socket
HOST = 'localhost'
PORT = 1234
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind((HOST, PORT))
server_socket.listen(1)

print('Server is listening for incoming connections...')

# Loop to accept incoming client connections
while True:
    # Wait for a client to connect
    client_socket, address = server_socket.accept()
    print(f'Connected to client at {address}')

    # Receive the number of files being sent
    num_files = int(client_socket.recv(1024).decode())
    print(f"Number of files being sent: {num_files}")

    # Receive each file and save it to disk
    for i in range(num_files):
        # Receive the file name and size
        file_name_size = client_socket.recv(1024).decode()
```

```
        file_name, file_size = file_name_size.split(',')

        # Receive the file data in chunks and save it to disk
        with open(file_name, 'wb') as f:
            bytes_received = 0
            while bytes_received < int(file_size):
                data = client_socket.recv(1024)
                f.write(data)
                bytes_received += len(data)

        print(f"Received file {i+1}: {file_name}")

    # Close the connection to the client
    client_socket.close()
```

```
PS E:\Desktop\exercise folder\CN\Miniproject> & C:/Users/sys/
ject/multifile_server.py"
Server is listening for incoming connections...
Connected to client at ('127.0.0.1', 57958)
Number of files being sent: 3
Received file 1: file1.txt
```

Client side:

```
import socket
import os

# Set up the client socket
HOST = 'localhost'
PORT = 1234
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect((HOST, PORT))

# Send the number of files being sent
num_files = 3
client_socket.sendall(str(num_files).encode())

# Send each file to the server
for i in range(num_files):
    # Get the file name and size
    file_name = input("Enter the file name to send: ")
    file_size = os.path.getsize(file_name)
    file_name_size = f"{file_name},{file_size}"

    # Send the file name and size to the server
    client_socket.sendall(file_name_size.encode())

    # Send the file data in chunks to the server
    with open(file_name, 'rb') as f:
        bytes_sent = 0
        while bytes_sent < file_size:
            data = f.read(1024)
            client_socket.sendall(data)
```

```
            bytes_sent += len(data)

    print(f"Sent file {i+1}: {file_name}")

# Close the connection to the server
client_socket.close()
```

```
E:\Desktop\exercise folder\CN\Miniproject>python multifile_client.py
Enter the file name to send: file1.txt
Sent file 1: file1.txt
```