**19CSE301- COMPUTER NETWORK**

**Case Study**

**Traffic Network System**

**Name: Suryanarayan.B**

**Roll no. :CB.EN.U4CSE19056**

|  |  |  |  |
| --- | --- | --- | --- |
| **Registration No** | **Name** | **Email ID** | **Contribution** |
| CB.EN. U4CSE19060 | S. Vigneshwararaj |  | Control Room |
| CB.EN. U4CSE19056 | Suryanarayan.B |  | Patrol and Toll |
| CB.EN. U4CSE19001 | Adithya Varshan |  | Judicial |
| CB.EN. U4CSE19008 | B. Sudarshan |  | RTO |

**Introduction**

Traffic Management and Monitoring systems play a huge role in everyday life of a commuter. They maintain order to ensure safety of the commuter as well as make sure they reach their destination. The Traffic Monitoring and Management Systems are part of the following departments within the Traffic police

* Response Units or Enforcement Units
  + - Patrol Units
      * Escorts
      * Highway and Road Policing
    - Outposts or Check posts
    - Highway Tolls
    - Medical Units
* Control Room
  + - Traffic Systems
    - Traffic Routing Management
* Judicial (Mobile Courts)
  + - Penalties
    - Fines
* RTO
  + - Licensing
    - Vehicle Registration
      * Commercial Vehicles
      * Industrial Vehicles
    - Tax Collection

**Road Patrol and Toll dept:**

Highway toll and Check post department monitors the traffic and vehicle counts which passes through either a toll booth or a check post which is normally located in the forest regions (Tiger reserves, sanctuaries). It also escorts the CM or PM travel.

**Component Minimum Recommended:**

**Processor**

Minimum: Pentium 1400 MHz Pentium III compatible

Recommended: Intel 7th gen core i7 7700. Quad core, octa thread. Base 3.6GHz, 8MB RAM: 32GB DDR4 2133mhz.

**RAM**

Minimum: 4GB

Recommended: 8 GB or more

**Video resolution**

Minimum: 800 x 600

Recommended: 1920 x 1080

**Server Configuration (ia64 hp server rx2660):**

2 Intel(R) Itanium 2 9100 series processors

23.98 GB RAM

12 MB cache/cpu

10 FC disk (450 GB each)(EMC Box)

4.5 TB Storage (EMC Box)

Dual external storage connectivity

DVD ROM drive

**Software/Operating System used:**

* **OS used:**
  + Windows 10
  + Windows DOS
* **Software used:**
  + Visual Studio Code
  + Cisco Packet Tracer
* **Language:**
  + Python

**Network Devices:**

The various devices used in the data network other than the nodes on which the applications reside, along with their interface specifications are given below:

1. Modems

2. Switches

3. Routers

**Types of Servers used:**

1. DNS Server
2. Web Server
3. Mail Server
4. FTP Server

**Why measure network performance?**

The demands on networks are increasing every day, and the need for proper network performance measurement is more important than ever before. Effective network performance translates into improved user satisfaction, whether that be internal employee efficiencies, or customer-facing network components such as an e-commerce website, making the business rationale for performance testing and monitoring self-evident.

When delivering services and applications to users, bandwidth issues, network down time, and bottlenecks can quickly escalate into IT crisis mode. Proactive network performance management solutions that detect and diagnose performance issues are the best way to guarantee ongoing user satisfaction.

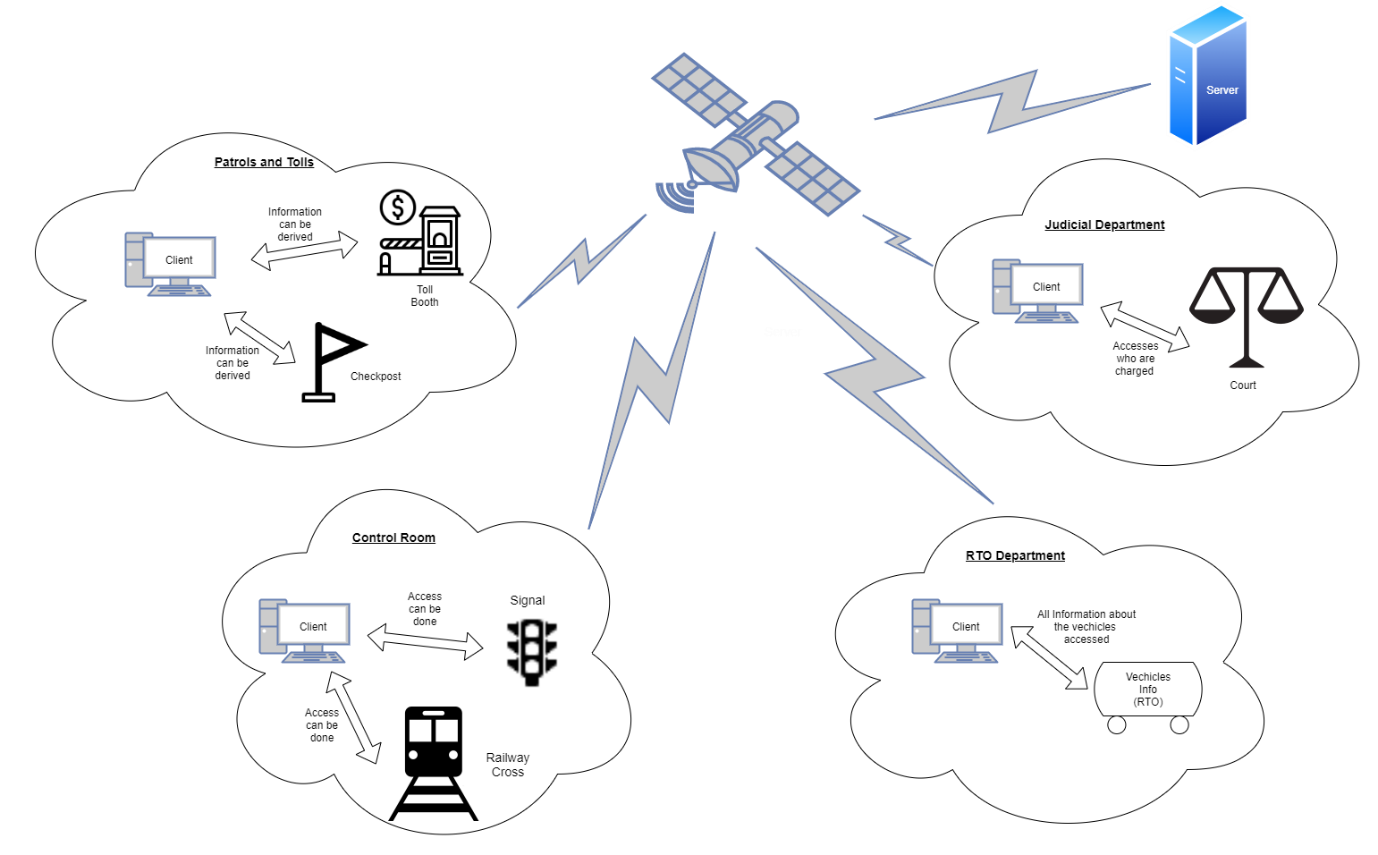
The performance of a network can never be fully modelled, so measuring network performance before, during, and after updates are made and monitoring performance on an ongoing basis are the only valid methods to fully ensure network quality. While measuring and monitoring network performance parameters are essential, the interpretation and actions stemming from these metrics are equally important.

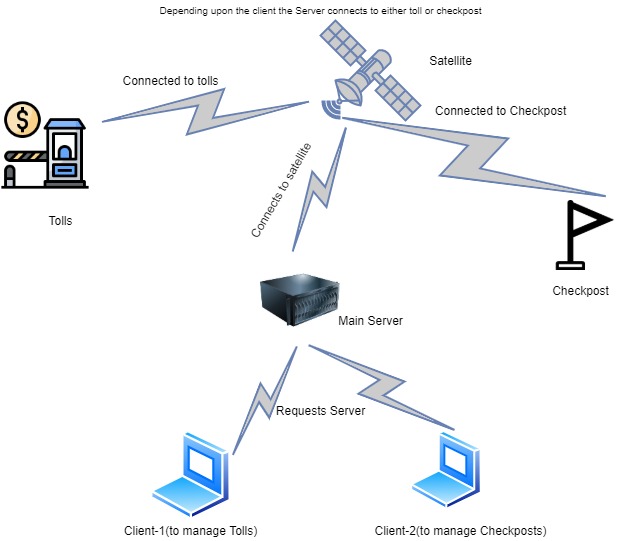
**Performance parameters:**

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Meaning** | **Formula** |
| **Bandwidth** | Bandwidth is the capacity of a wired or wireless network communications link to transmit the maximum amount of data from one point to another over a computer network or internet connection in a given amount of time | Expressed as [bits](https://web.archive.org/web/20190816003233/https:/whatis.techtarget.com/definition/bit-binary-digit) per second ([bps](https://web.archive.org/web/20190816003233/https:/searchnetworking.techtarget.com/definition/bits-per-second)), modern network links have greater capacity, which is typically measured in millions of bits per second ([megabits per second](https://web.archive.org/web/20190816003233/https:/searchnetworking.techtarget.com/definition/Mbps), or Mbps) or billions of bits per second ([gigabits per second](https://web.archive.org/web/20190816003233/https:/whatis.techtarget.com/definition/Gbps-billions-of-bits-per-second), or Gbps). |
| **Throughput** | Throughput measures the percentage of data packets that are successfully being sent; a low throughput means there are a lot of failed or dropped packets that need to be sent again. |  |
| **Packet Loss** | Packet loss occurs when one or more packets of data travelling across a computer network fail to reach their destination. Due to network congestion | Efficiency = 100% \* (transferred - retransmitted) / transferred  Network Loss = 100 - Efficiency |
| **Transmission time** | The time required for transmission of a message depends on the size of the message and the bandwidth of the channel. | Transmission time=Message size / Bandwidth |
| **Propagation Time** | Propagation time measures the time required for a bit to travel from the source to the destination. The propagation time is calculated by dividing the distance by the propagation speed. | Propagation time = Distance /Propagation speed |
| **Processing Delay** | Time taken by the processor to process the data packet is called processing delay. |  |
| **Queuing Delay** | Time spent by the data packet waiting in the queue before it is taken for execution is called queuing delay. |  |
| **Jitter** | Jitter is defined as the variation in time delay for the data packets sent over a network. This variable represents an identified disruption in the normal sequencing of data packets. Jitter is related to latency, since the jitter manifests itself in increased or uneven latency between data packets, which can disrupt network performance and lead to packet loss and network congestion. Although some level of jitter is to be expected and can usually be tolerated, quantifying network jitter is an important aspect of comprehensive network | Latency=sum of all delays    To measure Jitter, we take the difference between samples, then divide by the number of samples (minus 1). |

**Architecture Diagram:**

**Overall Architecture**



**For Toll and Checkpost department**

**Socket Programming:**

**List of operations:**

**For Toll:**

1)Entering data

2)Checking the no. of vehicles passing through each toll location

3)Total Fee collected

**For Checkpost:**

1)Entering Data

2) No. of cars passed through each checkpoint:

3) No. of people passing through the chechpost each day:

**Server:**

*import* socket

*import* random

*import* pandas *as* pd

*import* numpy *as* np

*import* string

*from* pandas.core.indexes.base *import* Index

host=socket.gethostname()

port=19008

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

data=pd.DataFrame(pd.read\_csv('toll.csv'))

data.set\_index(['Vehicle-no.','Location','Time','Amount','Way','Highway'])

data1=pd.DataFrame(pd.read\_csv('check.csv'))

data1.set\_index(['Vehicle','Time Entered','Time Exited','Location','Date Entered','Date Exited'])

def serverBrain(clientsocket,addr,host,data,data1):

    x="--------------Welcome to Highway Control room---------------------"

    clientsocket.send(x.encode())

    ques=clientsocket.recv(1024).decode()

    print(ques," \nConnected to client")

    ques='Choose a department :\n 1. Toll\n 2.Checkpost'

    clientsocket.send(ques.encode())

    ans=clientsocket.recv(1024).decode()

    intAns=int(ans)

*if*(intAns==1):

        ques='Select an operation :\n 1.Enter Details  \n2.No of cars passed through each toll \n3. Total amount Collected'

        clientsocket.send(ques.encode())

        ans=clientsocket.recv(1024).decode()

        intAns=int(ans)

*if*(intAns==1):

            s=[]

            enterData="Vehicle-no.: "

            clientsocket.send(enterData.encode())

            vehicle=clientsocket.recv(1024).decode()

            s.append(vehicle)

            enterData="Location: "

            clientsocket.send(enterData.encode())

            loc=clientsocket.recv(1024).decode()

            s.append(loc)

            enterData="Time: "

            clientsocket.send(enterData.encode())

            tiem=clientsocket.recv(1024).decode()

            s.append(tiem)

            enterData="Amount: "

            clientsocket.send(enterData.encode())

            amt=clientsocket.recv(1024).decode()

            s.append(amt)

            enterData="Way: "

            clientsocket.send(enterData.encode())

            way=clientsocket.recv(1024).decode()

            s.append(way)

            enterData="Highway: "

            clientsocket.send(enterData.encode())

            hw=clientsocket.recv(1024).decode()

            s.append(hw)

            n=len(data)

            data.loc[n]=s

            i=data.to\_string()

            clientsocket.send(i.encode())

            data.to\_csv('toll.csv',index=False)

*elif*(intAns==2):

            x=data['Location'].value\_counts()

            x=x.to\_string()

            clientsocket.send(bytes(x,"utf-8"))

*elif*(intAns==3):

            x=data['Amount'].sum()

            x=str(x)

            clientsocket.send(bytes(x,"utf-8"))

*elif*(intAns==2):

        ques='Select an operation :\n 1.Enter Details \n2. No of cars passed through each checkpoint \n3.no. of people passed through each day'

        clientsocket.send(ques.encode())

        ans=clientsocket.recv(1024).decode()

        intAns=int(ans)

*if*(intAns==1):

            u=[]

            enterData="Vehicle-no.: "

            clientsocket.send(enterData.encode())

            vehicle=clientsocket.recv(1024).decode()

            u.append(vehicle)

            enterData="Time Entered: "

            clientsocket.send(enterData.encode())

            te=clientsocket.recv(1024).decode()

            u.append(te)

            enterData="Time Exited: "

            clientsocket.send(enterData.encode())

            tex=clientsocket.recv(1024).decode()

            u.append(tex)

            enterData="Location: "

            clientsocket.send(enterData.encode())

            loc=clientsocket.recv(1024).decode()

            u.append(loc)

            enterData="Date Entered: "

            clientsocket.send(enterData.encode())

            de=clientsocket.recv(1024).decode()

            u.append(de)

            enterData="Date Exited: "

            clientsocket.send(enterData.encode())

            dex=clientsocket.recv(1024).decode()

            u.append(dex)

            n=len(data)

            data.loc[n]=u

            i=data.to\_string()

            clientsocket.send(i.encode())

            data.to\_csv('check.csv',index=False)

*elif*(intAns==2):

            x=data1['Location'].value\_counts()

            x=x.to\_string()

            clientsocket.send(bytes(x,"utf-8"))

*elif*(intAns==3):

            x=data1['Date Entered'].value\_counts()

            y=data1['Date Exited'].value\_counts()

            a="No.of people entered during a particular day\n"

            clientsocket.send(a.encode())

            x=x.to\_string()

            y=y.to\_string()

            clientsocket.send(bytes(x,"utf-8"))

            b="No.of people exited during a particular day\n"

            clientsocket.send(b.encode())

            clientsocket.send(bytes(y,"utf-8"))

*else*:

        errorMsg="Wrong Option! Please Select any of the above options only"

        clientsocket.send(errorMsg.encode())

    clientsocket.close()

def initiate():

    s.bind((host,port))

    s.listen(5)

    x=True

*while*(x):

        c,addr=s.accept()

        serverBrain(c,addr,host,data,data1)

        ch=input("\nDo You want to continue(y/n): ")

*if*(ch=='y'):

*pass*

*else*:

            x=False

print('----------------Server Initiated !-----------------')

initiate()

**Client:**

*import* threading

*import* socket

sck=socket.socket()

host=socket.gethostname()

port=19008

sck.connect((host,port))

print(sck.recv(1024).decode())

sck.send(bytes("-----Client Initiated-----","utf-8"))

print(sck.recv(1024).decode())

ch=input()

*if*(ch=="1"):

    sck.send(bytes(ch,"utf-8"))

    rply=sck.recv(1024).decode()

    print(rply)

    ch=input()

*if*(ch=="1"):

        sck.send(bytes(ch,"utf-8"))

        rply=sck.recv(1024).decode()

        print(rply)

        vno=input()

        sck.send(bytes(vno,"utf-8"))

        rply=sck.recv(1024).decode()

        print(rply)

        loc=input()

        sck.send(bytes(loc,"utf-8"))

        rply=sck.recv(1024).decode()

        print(rply)

        t=input()

        sck.send(bytes(t,"utf-8"))

        rply=sck.recv(1024).decode()

        print(rply)

        a=input()

        sck.send(bytes(a,"utf-8"))

        rply=sck.recv(1024).decode()

        print(rply)

        w=input()

        sck.send(bytes(w,"utf-8"))

        rply=sck.recv(1024).decode()

        print(rply)

        hw=input()

        sck.send(bytes(hw,"utf-8"))

        rply=sck.recv(1024).decode()

        print(rply)

*elif*(ch=="2"):

        sck.send(bytes(ch,"utf-8"))

        rply=sck.recv(1024).decode()

        print(rply)

*elif*(ch=="3"):

        sck.send(bytes(ch,"utf-8"))

        rply=sck.recv(1024).decode()

        print(rply)

*elif*(ch=='2'):

    sck.send(bytes(ch,"utf-8"))

    rply=sck.recv(1024).decode()

    print(rply)

    ch=input()

*if*(ch=="1"):

        sck.send(bytes(ch,"utf-8"))

        rply=sck.recv(1024).decode()

        print(rply)

        vno=input()

        sck.send(bytes(vno,"utf-8"))

        rply=sck.recv(1024).decode()

        print(rply)

        te=input()

        sck.send(bytes(te,"utf-8"))

        rply=sck.recv(1024).decode()

        print(rply)

        tex=input()

        sck.send(bytes(tex,"utf-8"))

        rply=sck.recv(1024).decode()

        print(rply)

        loc=input()

        sck.send(bytes(loc,"utf-8"))

        rply=sck.recv(1024).decode()

        print(rply)

        de=input()

        sck.send(bytes(de,"utf-8"))

        rply=sck.recv(1024).decode()

        print(rply)

        dex=input()

        sck.send(bytes(dex,"utf-8"))

        rply=sck.recv(1024).decode()

        print(rply)

*elif*(ch=="2"):

        sck.send(bytes(ch,"utf-8"))

        rply=sck.recv(1024).decode()

        print(rply)

*elif*(ch=="3"):

        sck.send(bytes(ch,"utf-8"))

        rply=sck.recv(1024).decode()

        rply1=sck.recv(1024).decode()

        rply2=sck.recv(1024).decode()

        rply3=sck.recv(1024).decode()

        print(rply ,'\n',rply1,'\n',rply2,'\n',rply3)

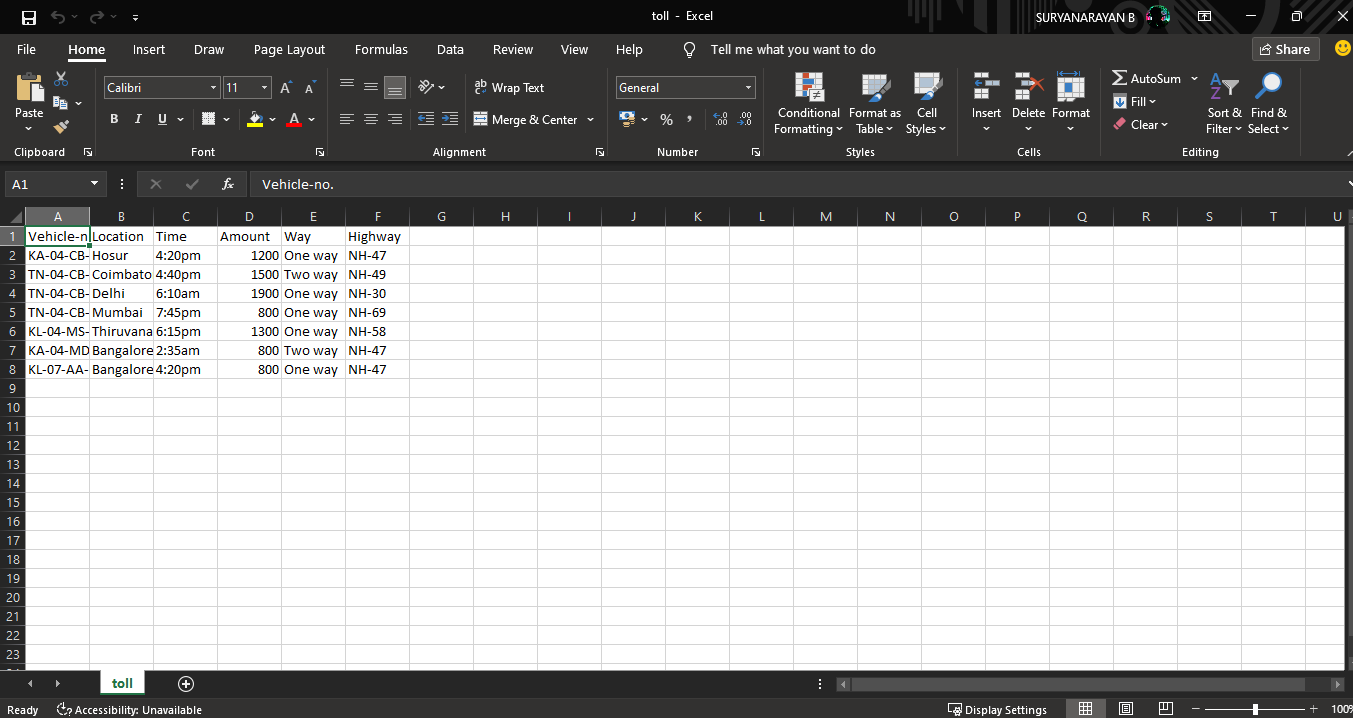
*else*:

*pass*

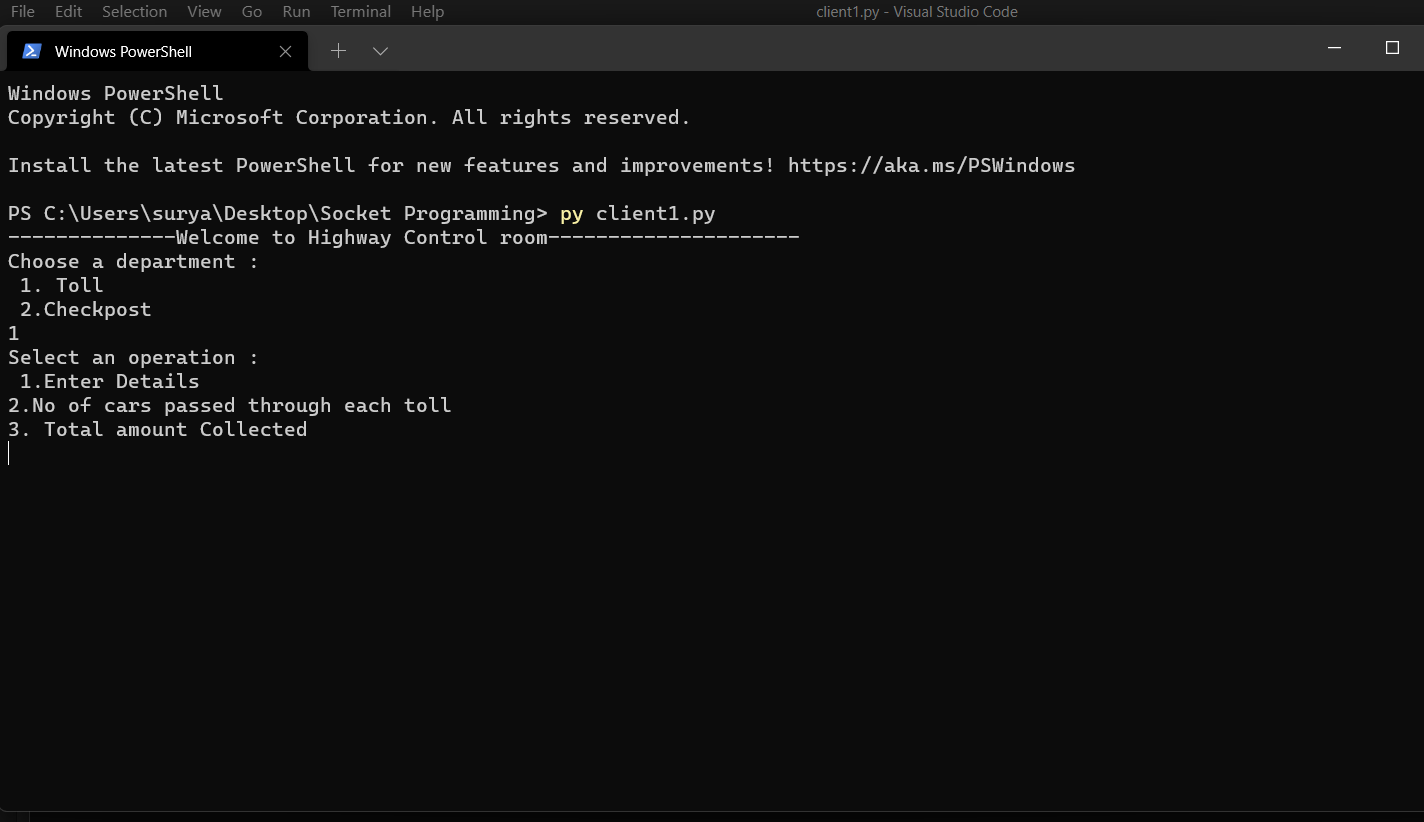
sck.close()

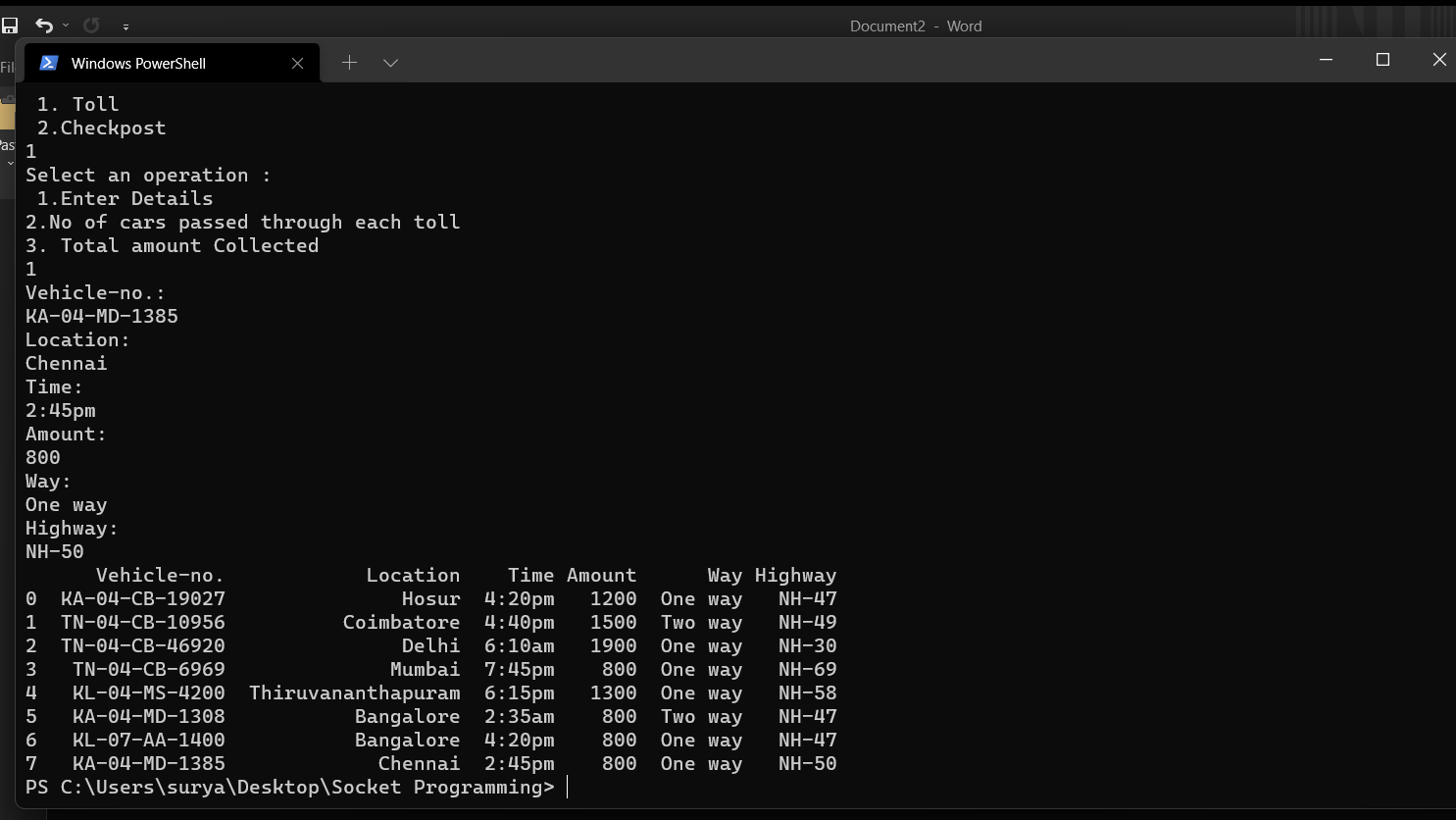
**Output:**

**Toll:**

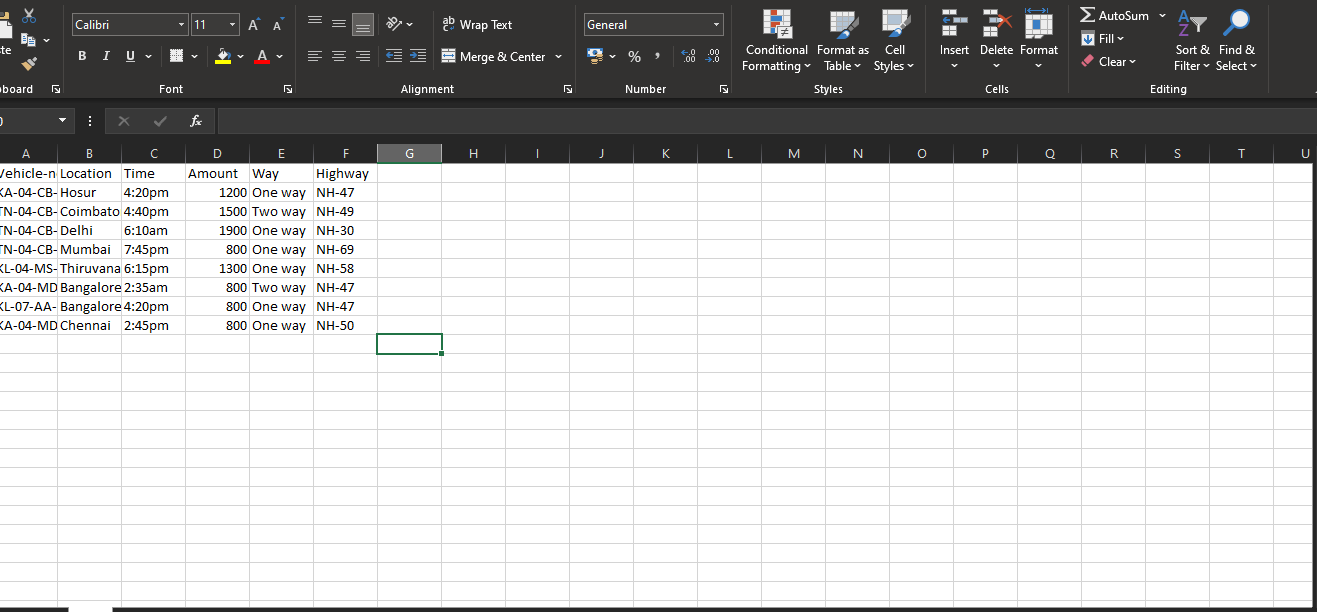
Toll csv before input:

Toll client:





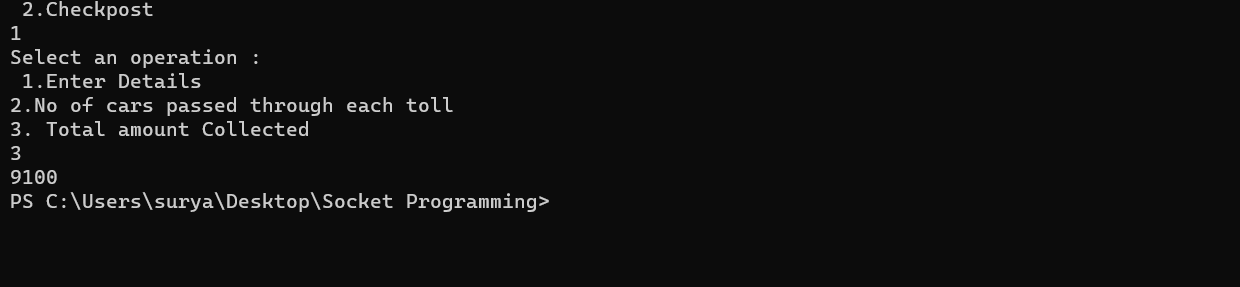
Csv after Entry:



No. of cars passed through each toll:

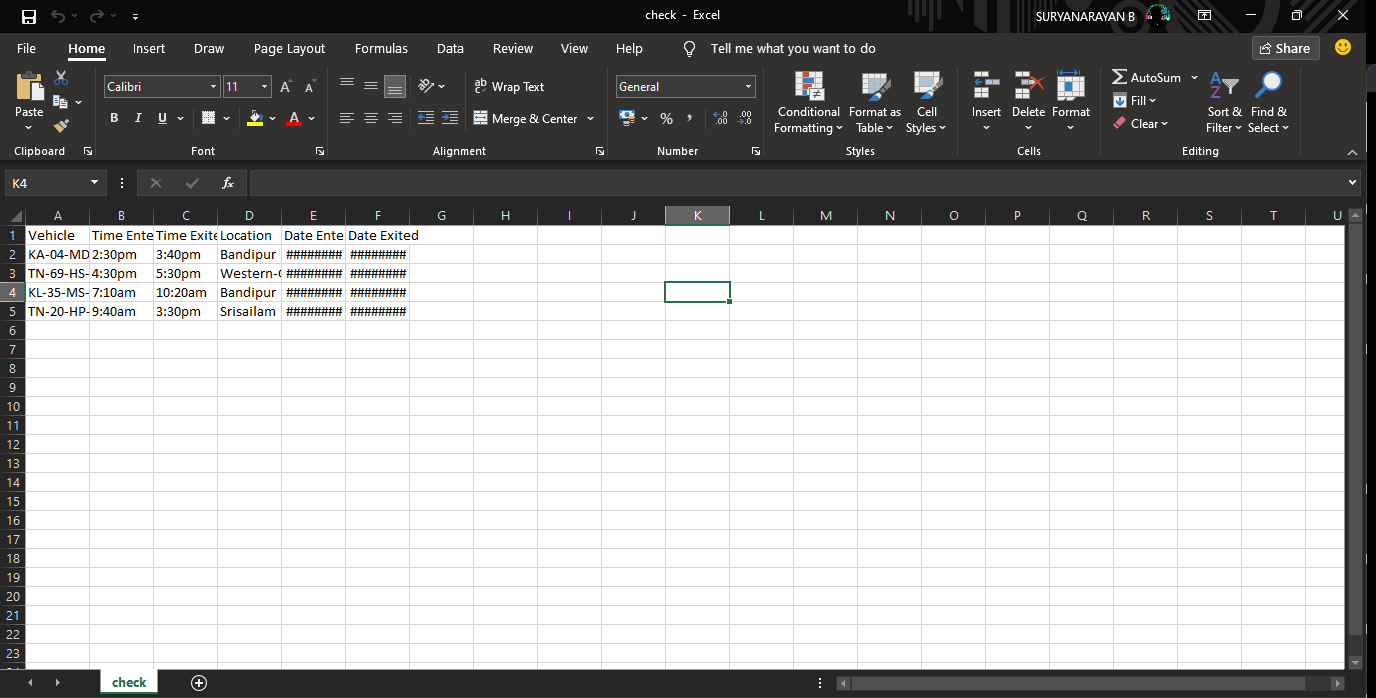


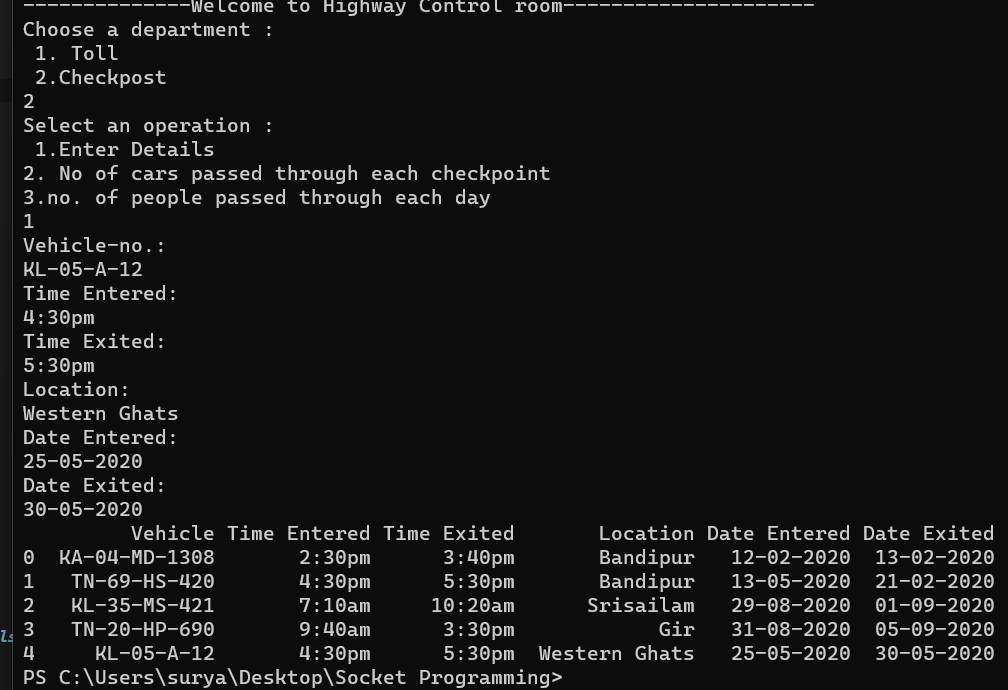
Total Amount collected:

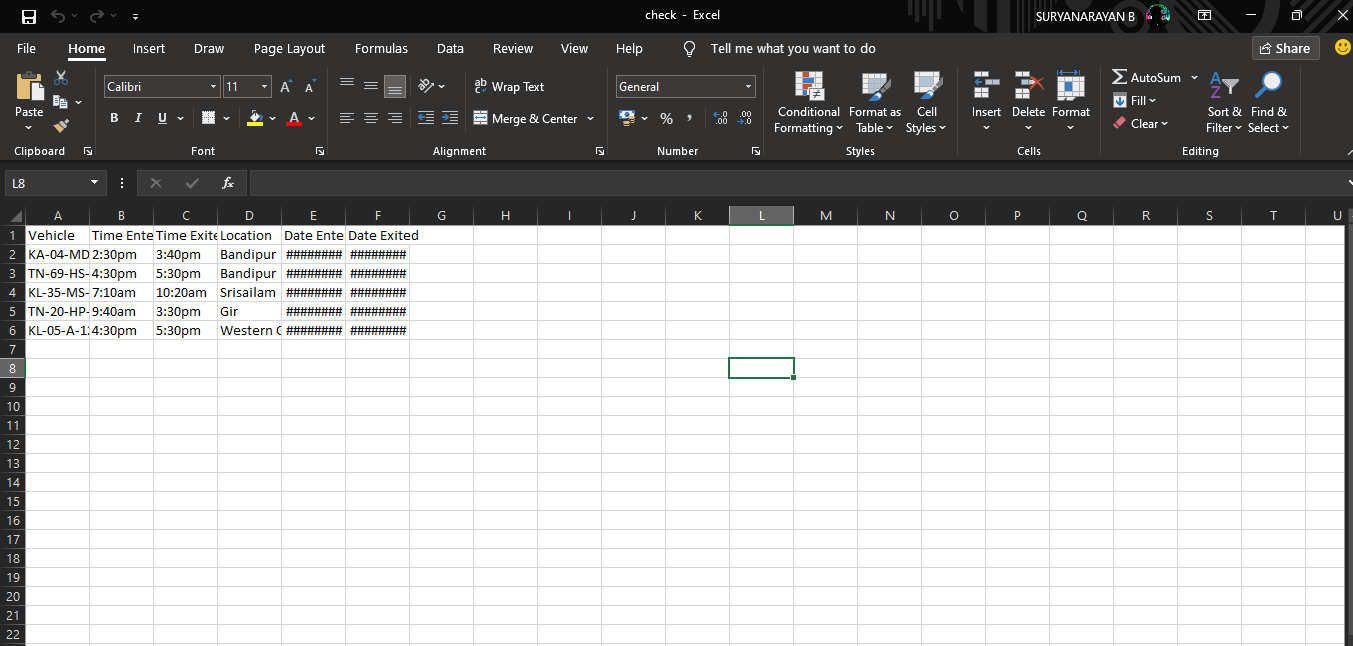


**Checkpoint:**

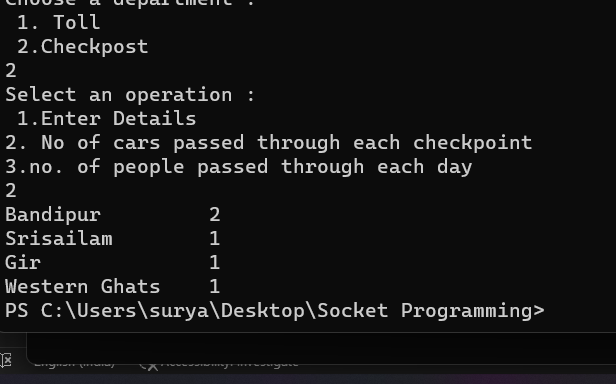
Checkpoint csv before input:

****

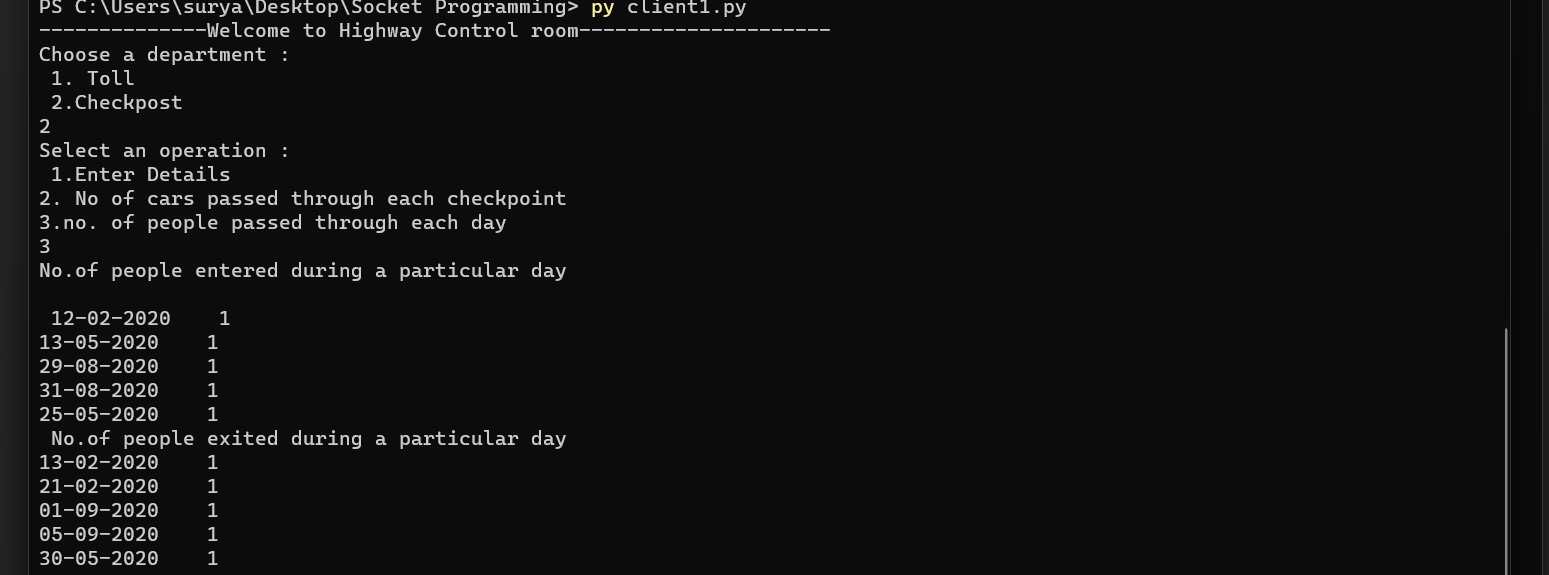
Checkpoint client:Csv after Modification:



No. of cars passed through each checkpoint:

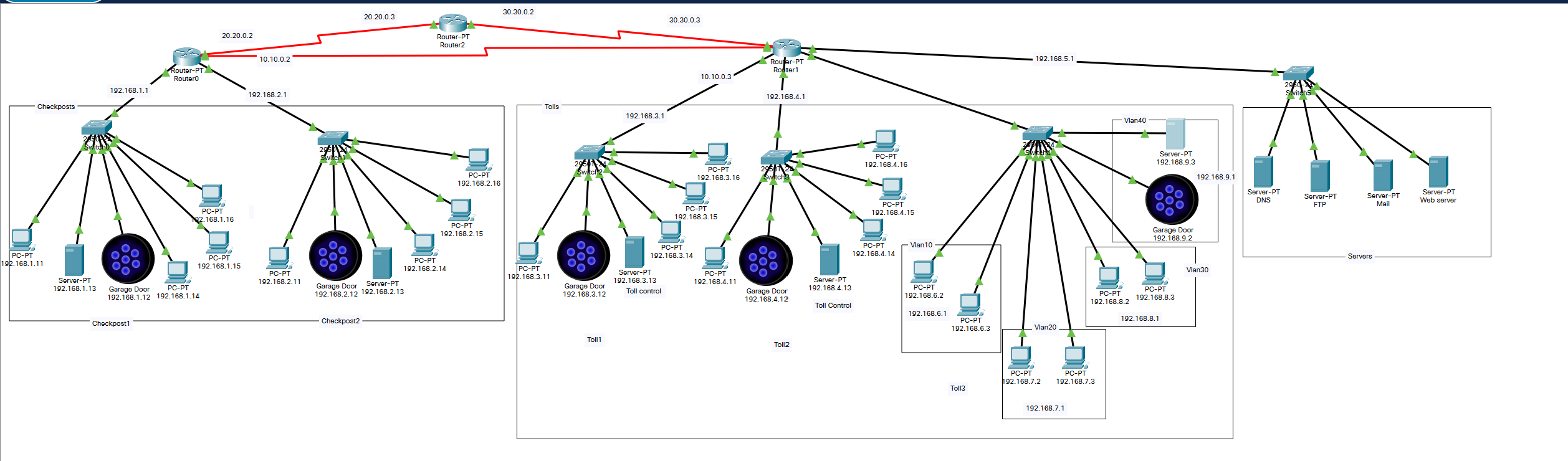


No. of people passing through the chechpost each day:

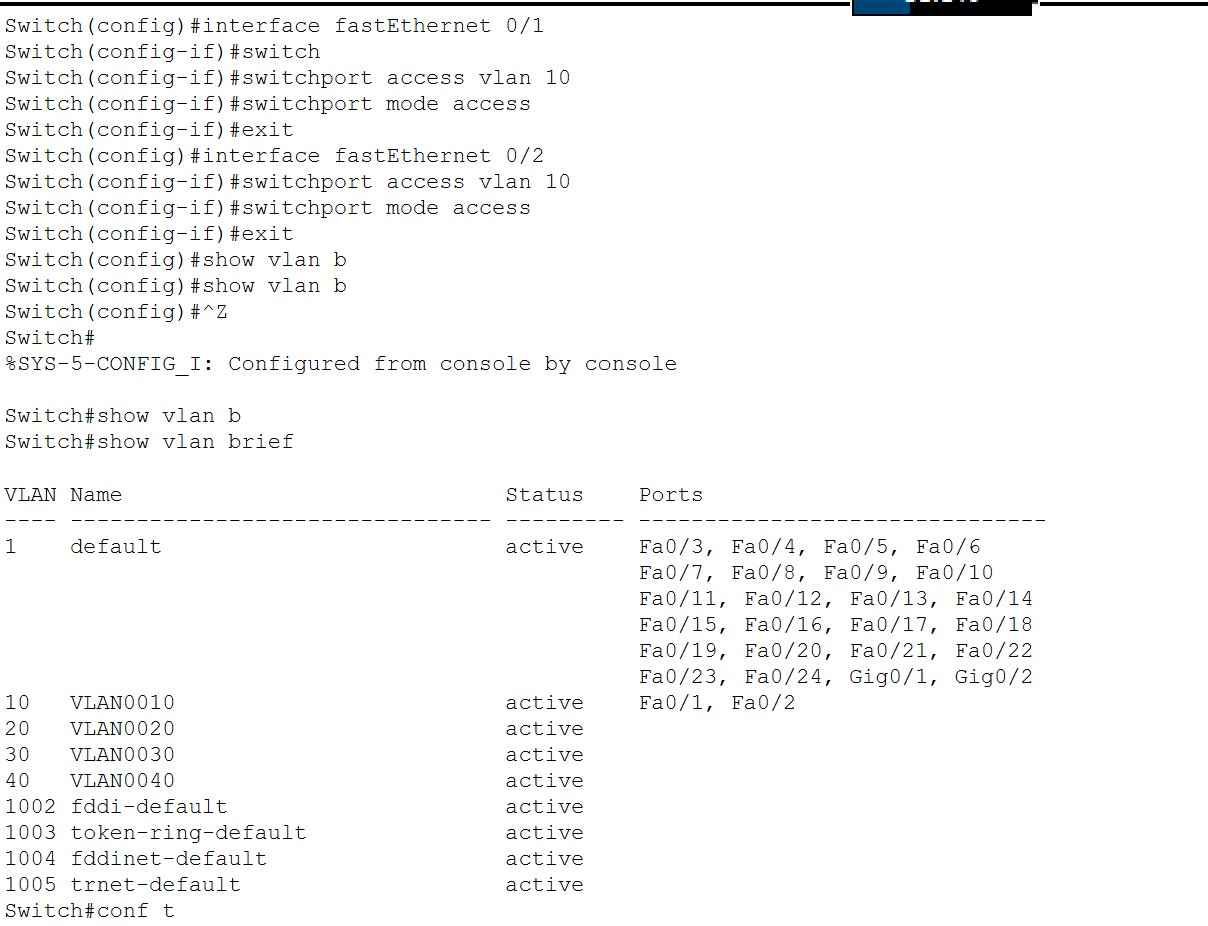


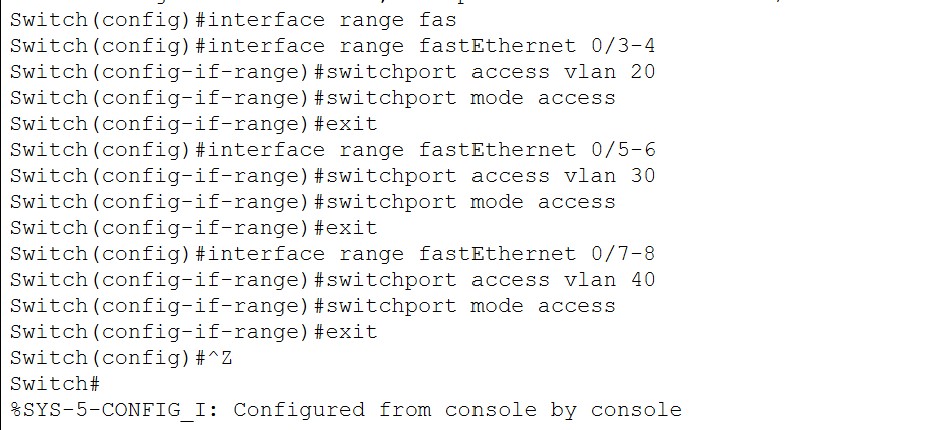
**Cisco Packet Tracer**

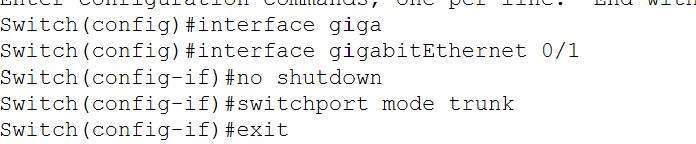
Checkpost and Toll Model

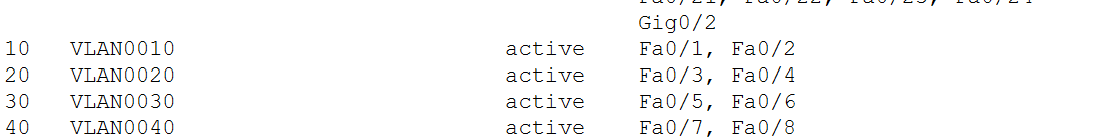


Vlan:

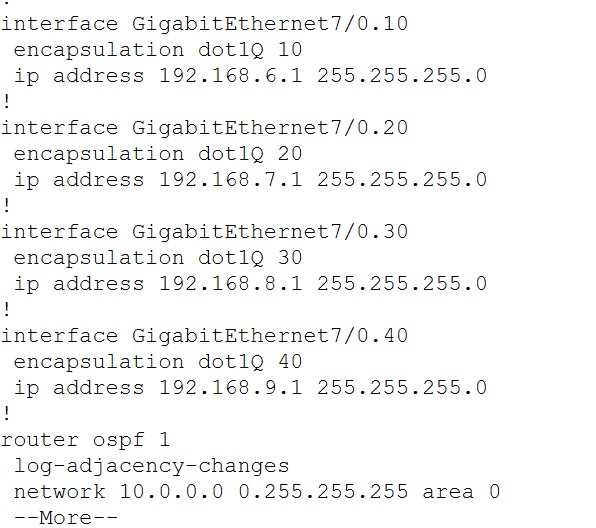


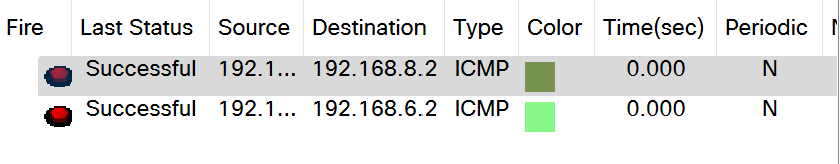






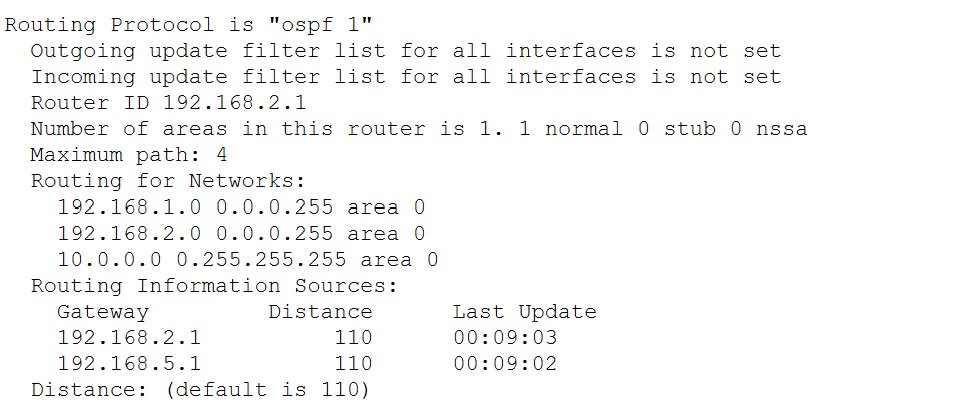
Router:



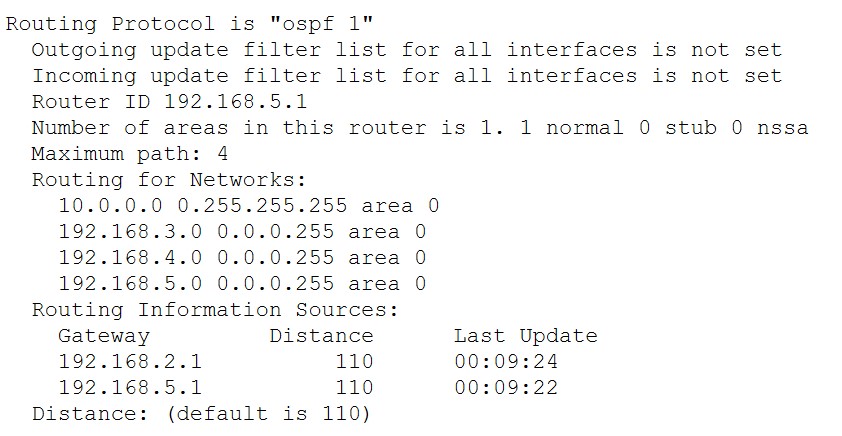


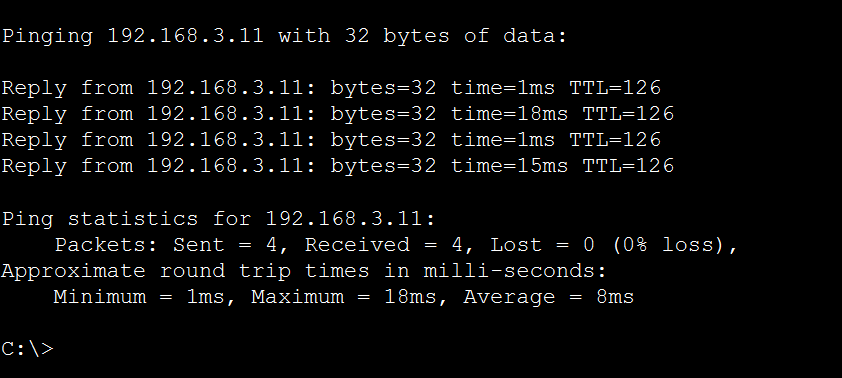
OSPF:

Router 1

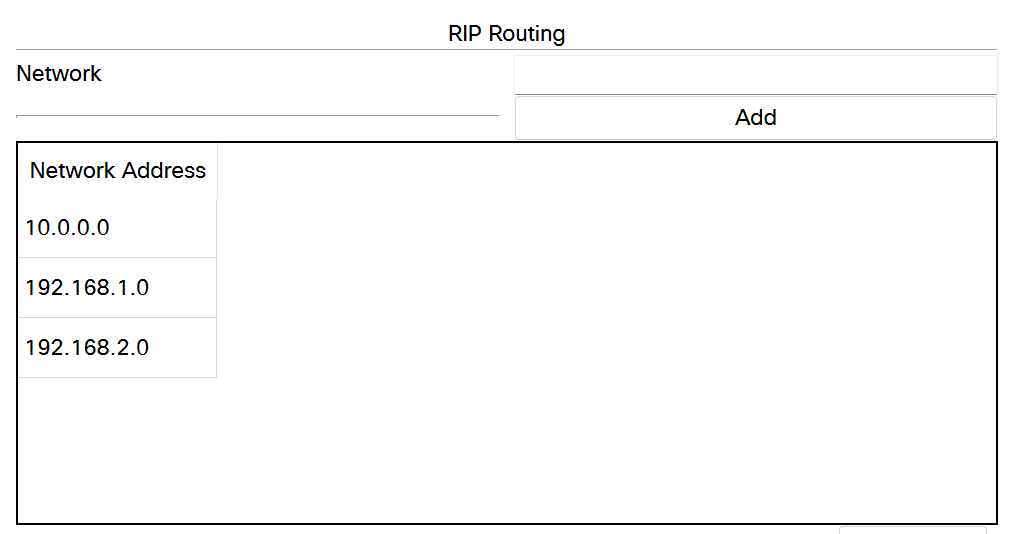


Router 2

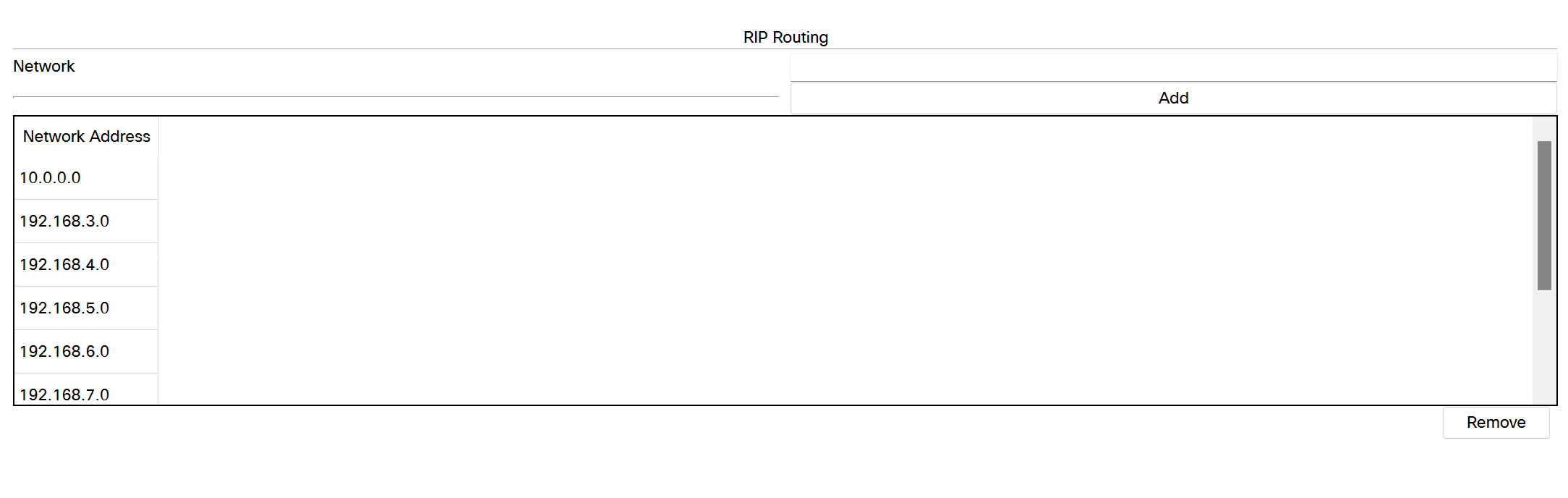


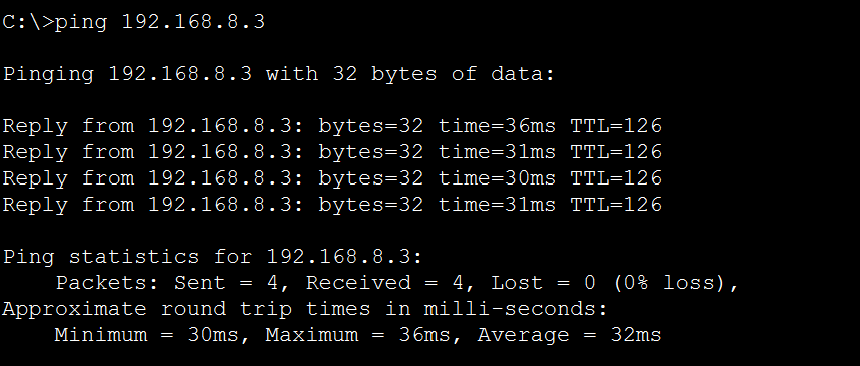


RIP:

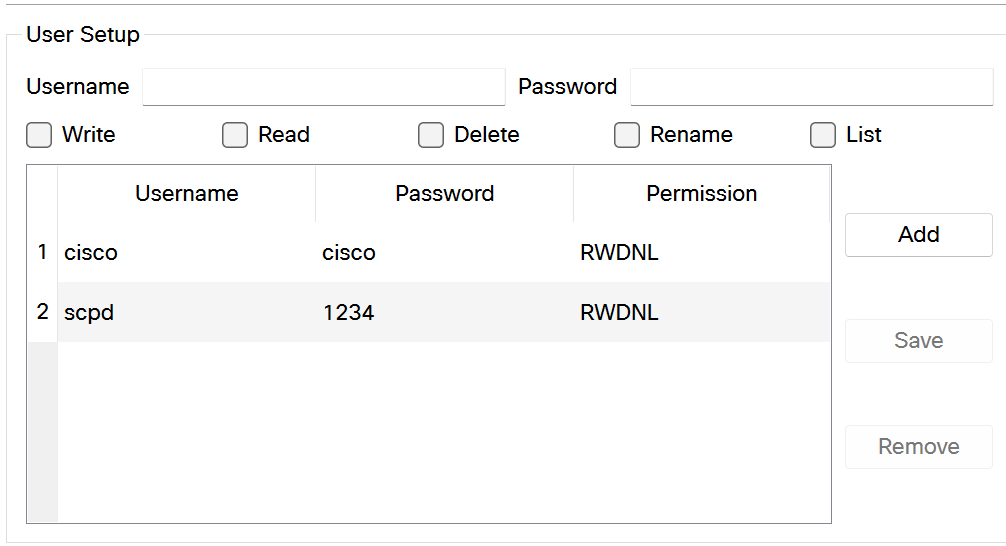
Router 1  


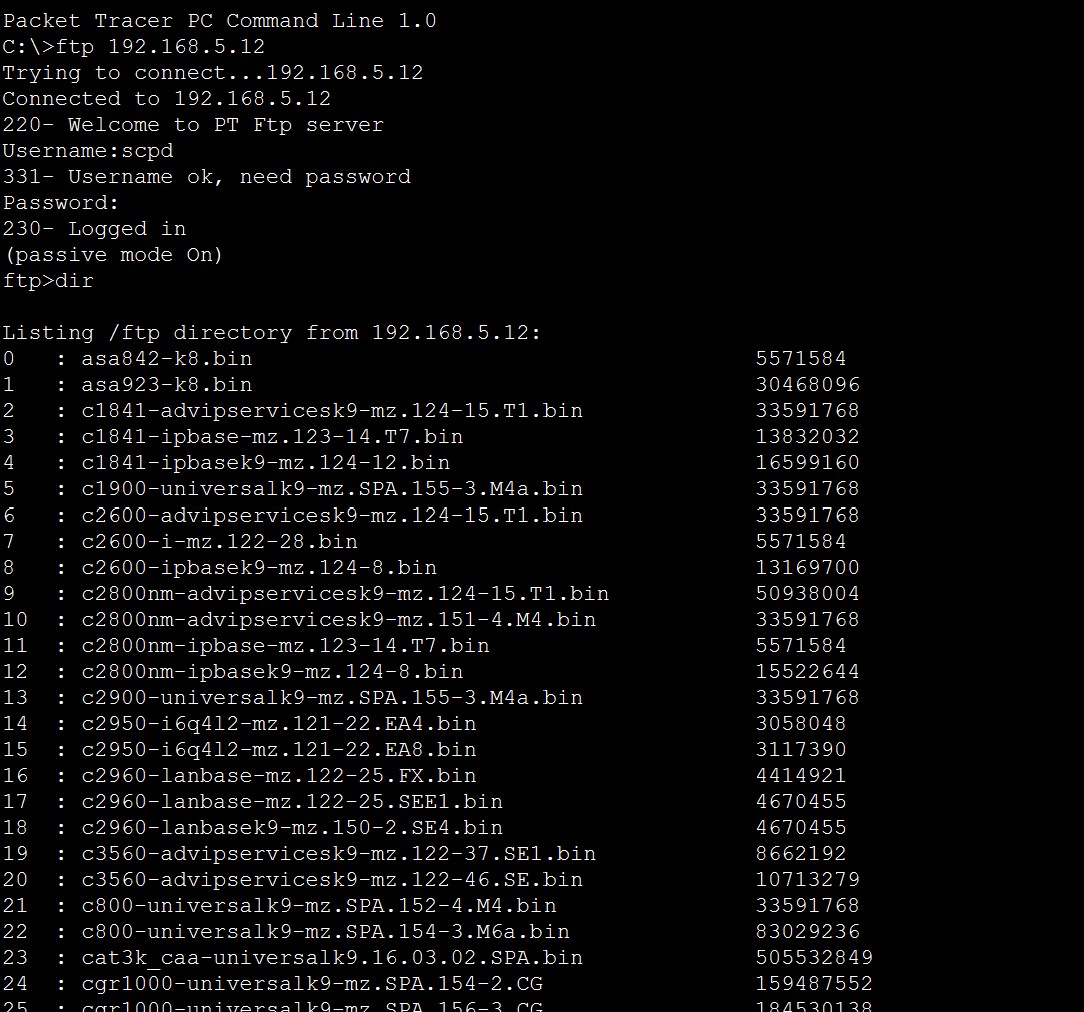
Router 2:



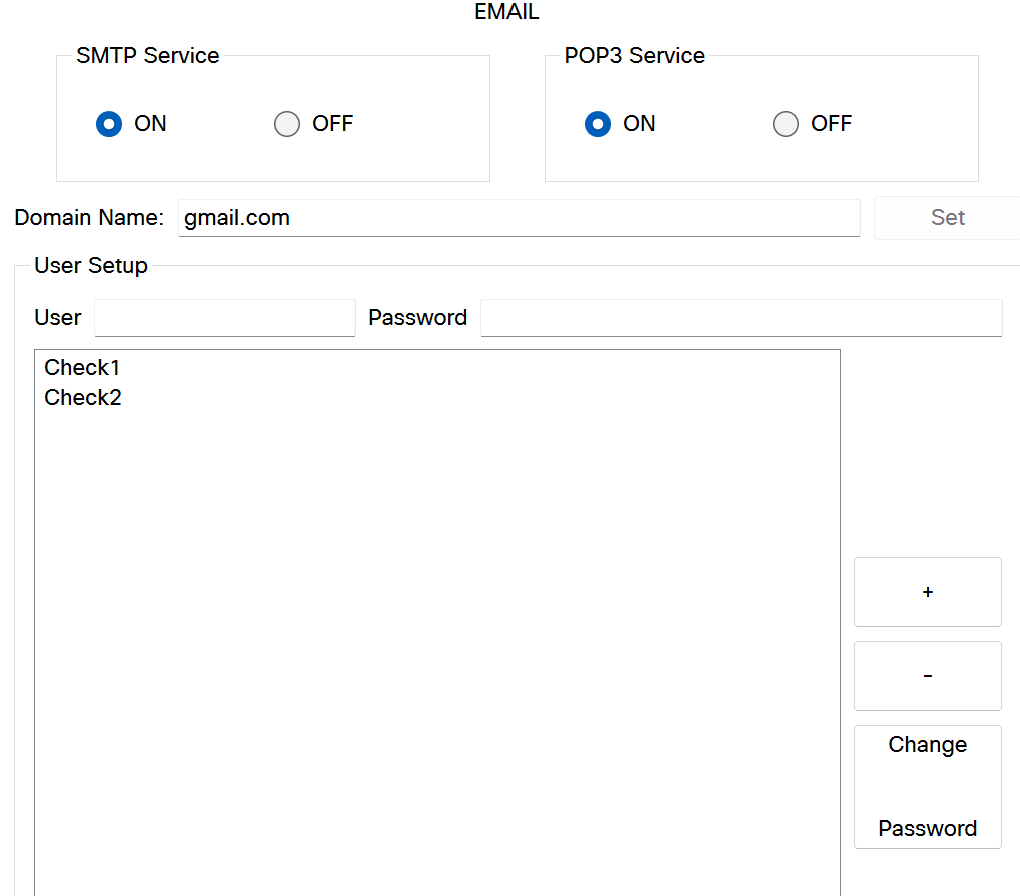


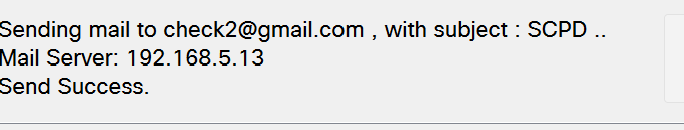
FTP:

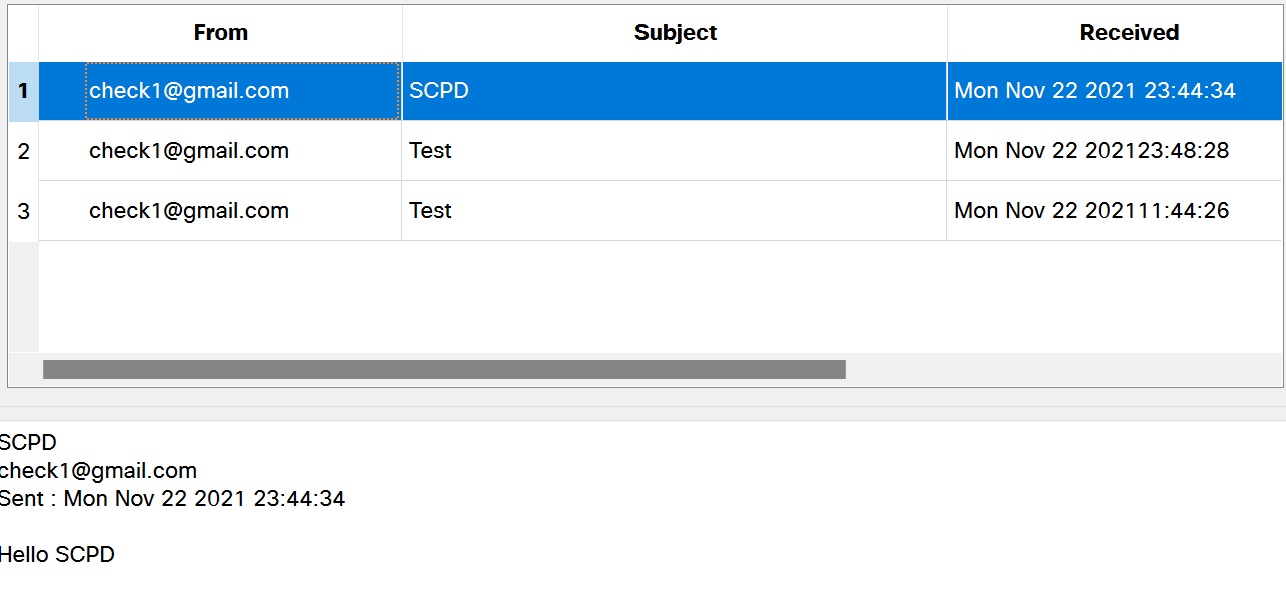




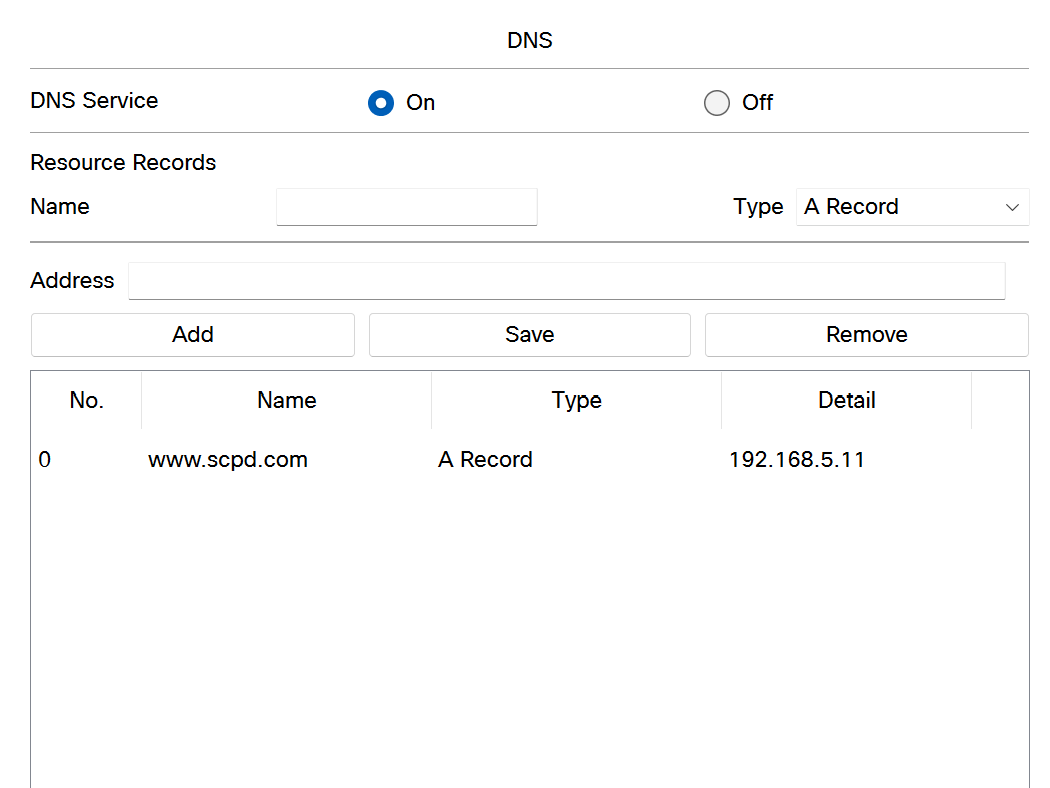
Mail Server:

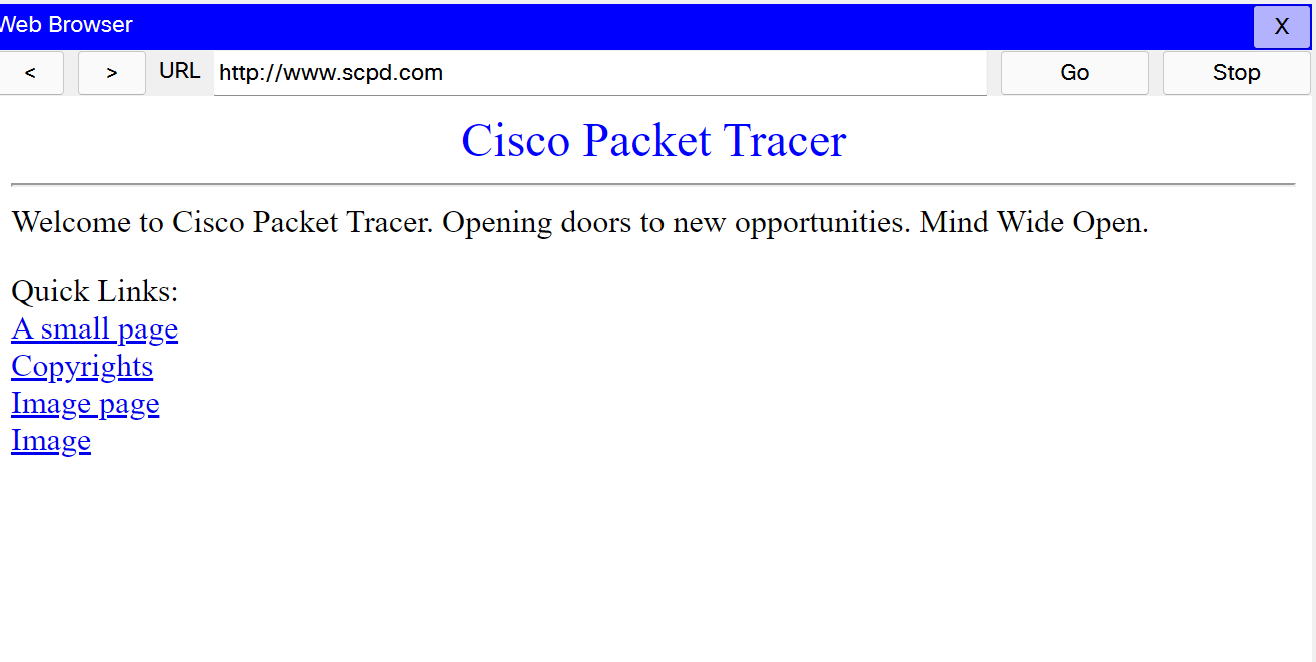






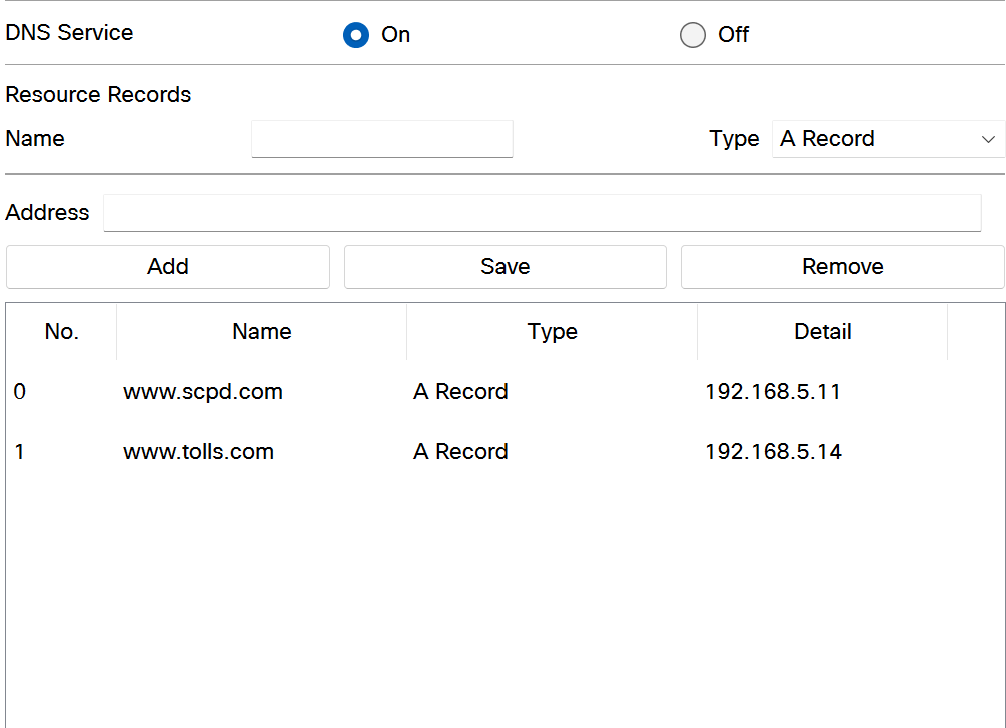
DNS:

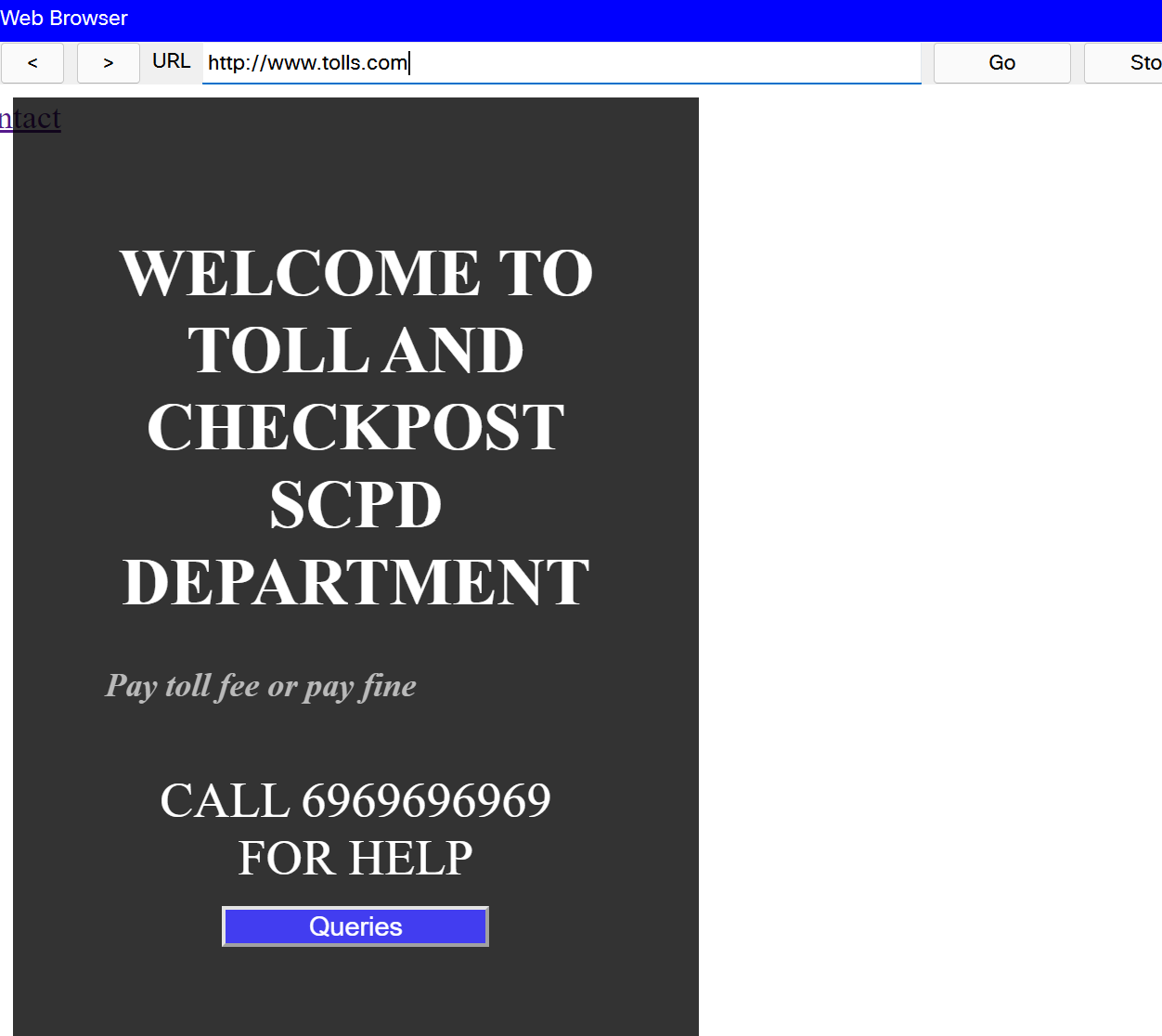




**Web Server:**

**Configured Web server:**

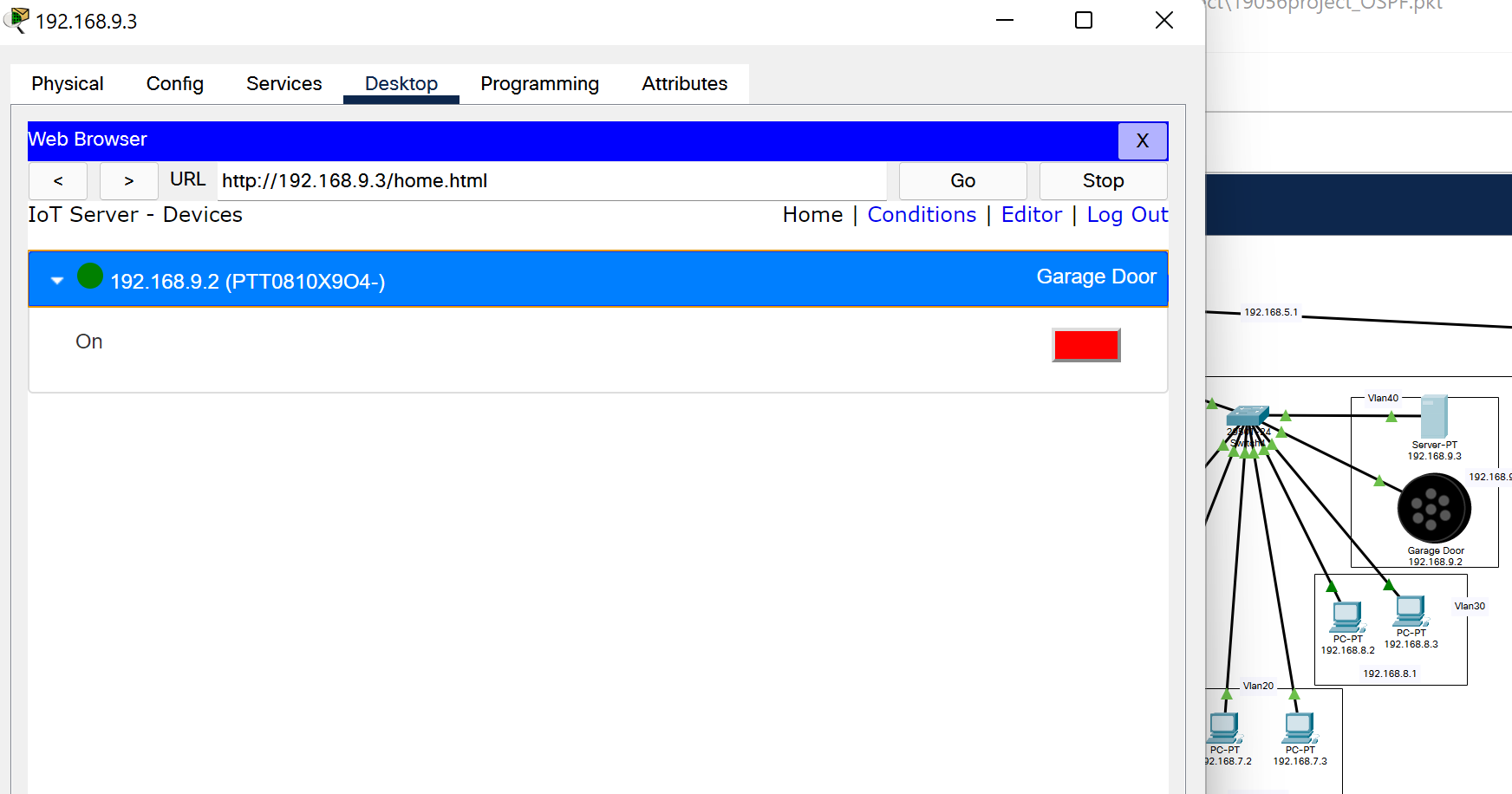
****

**Output:**

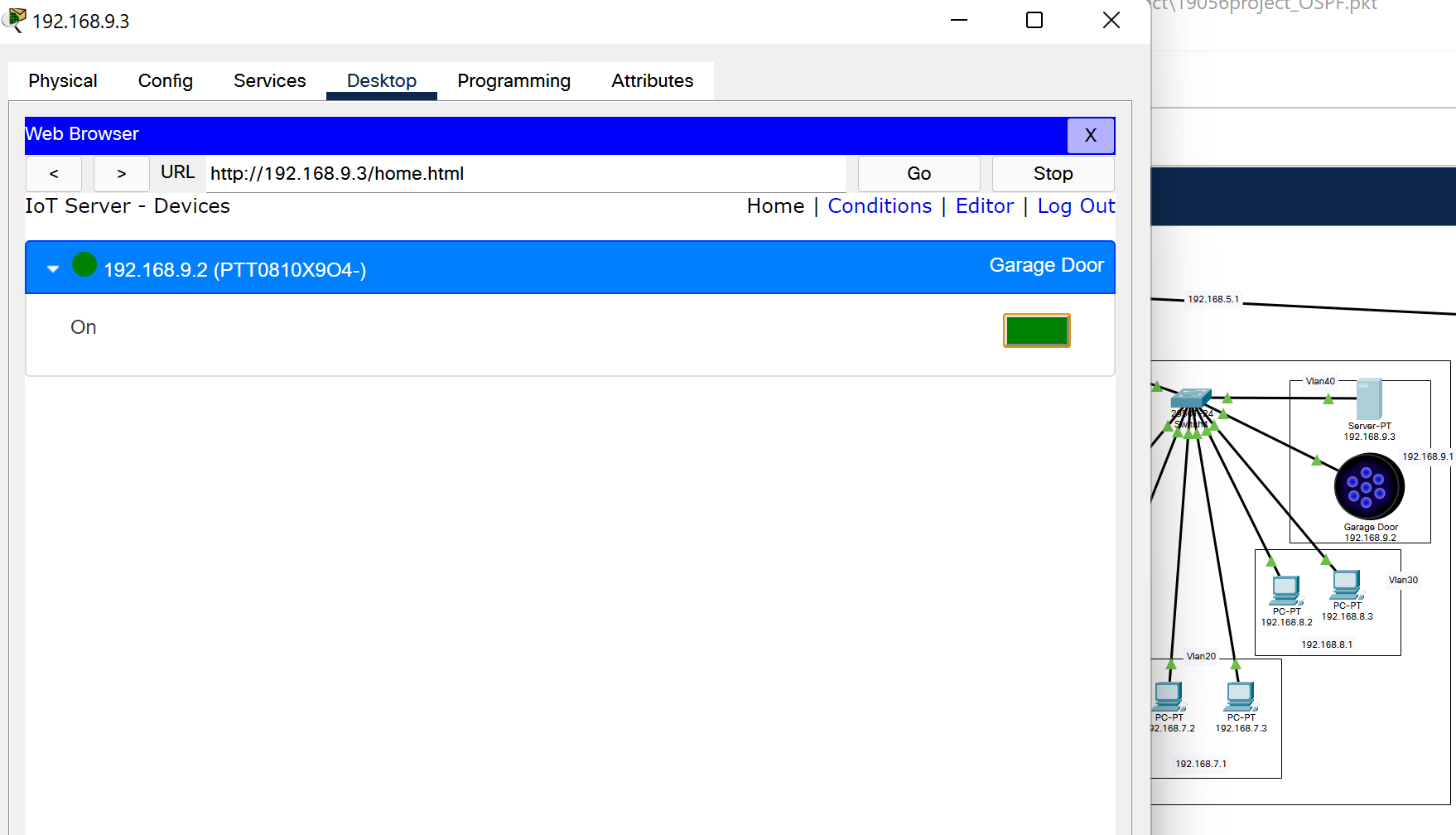
**IOT Devices:**

**To show the closing and opening of Toll and Checkpost Gates**

**Gate Closed:**



Gate Open:



**Subnetting**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Name | Hosts Needed | Hosts Available | Network Address | Mask | Usable Range | Broadcast |
| Permit Issuance Division | 7 | 14 | 192.168.1.0 | 255.255.255.240 | 192.168.1.1 - 192.168.1.14 | 192.168.1.15 |
| License Issuance Division | 7 | 14 | 192.168.1.16 | 255.255.255.240 | 192.168.1.17 - 192.168.1.30 | 192.168.1.31 |
| License & Permit Servers | 4 | 6 | 192.168.1.32 | 255.255.255.248 | 192.168.1.33 - 192.168.1.38 | 192.168.1.39 |

**GobackN and Selective Repeat**

**Client.java**

import java.lang.System;

    import java.net.\*;

    import java.io.\*;

    public class Client {

        static *Socket* connection;

        public static *void* main(*String* *a*[]) throws *SocketException* {

            try {

*int* v[] = new *int*[6];

                //int g[] = new int[8];

*int* n = 0;

*InetAddress* addr = InetAddress.getByName("Localhost");

                System.out.println(addr);

                connection = new Socket(addr, 8011);

*DataOutputStream* out = new DataOutputStream(

                        connection.getOutputStream());

*DataInputStream* in = new DataInputStream(

                        connection.getInputStream());

*int* p = in.read();

                System.out.println("No of frame is:" + p);

                for (*int* i = 0; i < p; i++) {

                    v[i] = in.read();

                    System.out.println(v[i]);

                    //g[i] = v[i];

                }

                v[5] = -1;

                v[3] = -1;

                for (*int* i = 0; i < p; i++)

                 {

                    System.out.println("Received frame is: " + v[i]);

                    }

                for (*int* i = 0; i < p; i++)

                    if (v[i] == -1) {

                System.out.println("Request to retransmit packet no "

                                + (i+1) + " again!!");

                        n = i;

                        out.write(n);

                        out.flush();

                    }

                System.out.println();

                    v[n] = in.read();

                System.out.println("Received frame is: " + v[n]);

                System.out.println("quiting");

            } catch (*Exception* *e*) {

                System.out.println(e);

            }

        }

    }

**Server.java**

import java.io.DataInputStream;

    import java.io.DataOutputStream;

    import java.io.IOException;

    import java.net.ServerSocket;

    import java.net.Socket;

    import java.net.SocketException;

    public class Server {

        static *ServerSocket* Serversocket;

        static *DataInputStream* dis;

        static *DataOutputStream* dos;

        public static *void* main(*String*[] *args*) throws *SocketException* {

        try {

*int* a[] = { 3,5,7,8,10,12};

            Serversocket = new ServerSocket(8011);

            System.out.println("waiting for connection");

*Socket* client = Serversocket.accept();

            dis = new DataInputStream(client.getInputStream());

            dos = new DataOutputStream(client.getOutputStream());

            System.out.println("The number of packets sent is:" + a.length);

*int* y = a.length;

            dos.write(y);

            dos.flush();

            for (*int* i = 0; i < a.length; i++) {

                dos.write(a[i]);

                dos.flush();

            }

*int* k = dis.read();

            dos.write(a[k]);

            dos.flush();

            } catch (*IOException* *e*) {

                System.out.println(e);

            } finally {

                try {

                    dis.close();

                    dos.close();

                } catch (*IOException* *e*) {

                    // TODO Auto-generated catch block

                    e.printStackTrace();

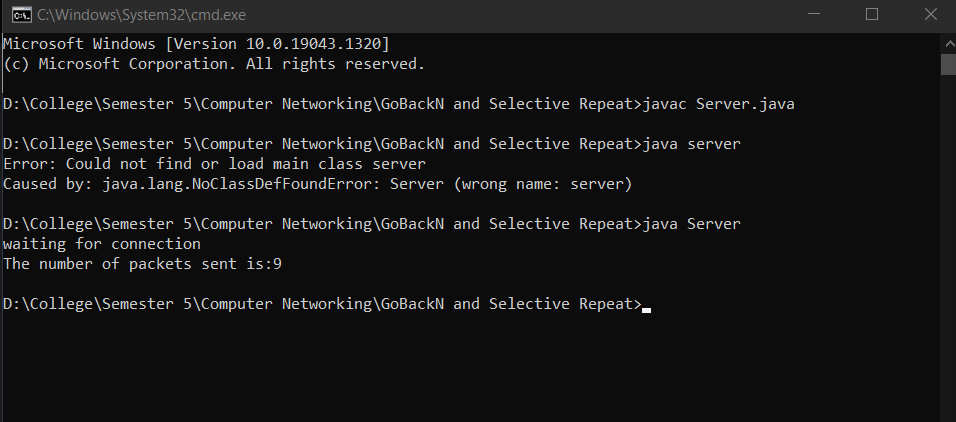
                }

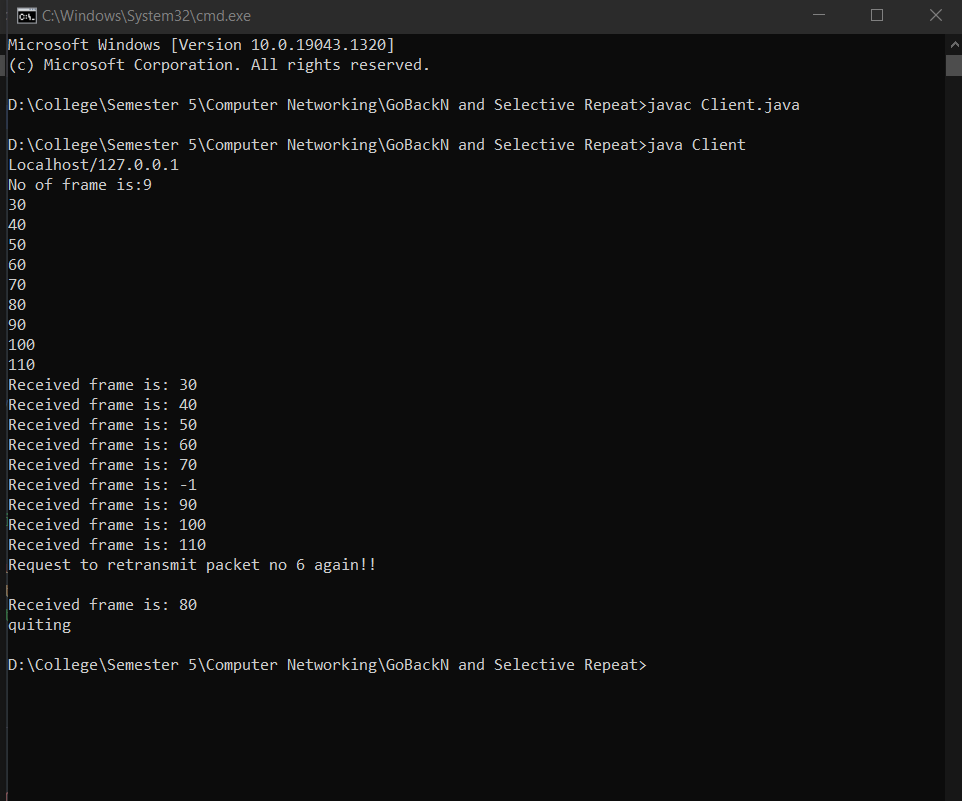
            }

        }

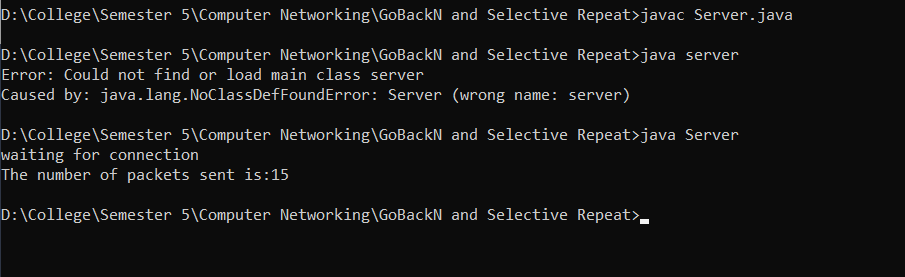
    }

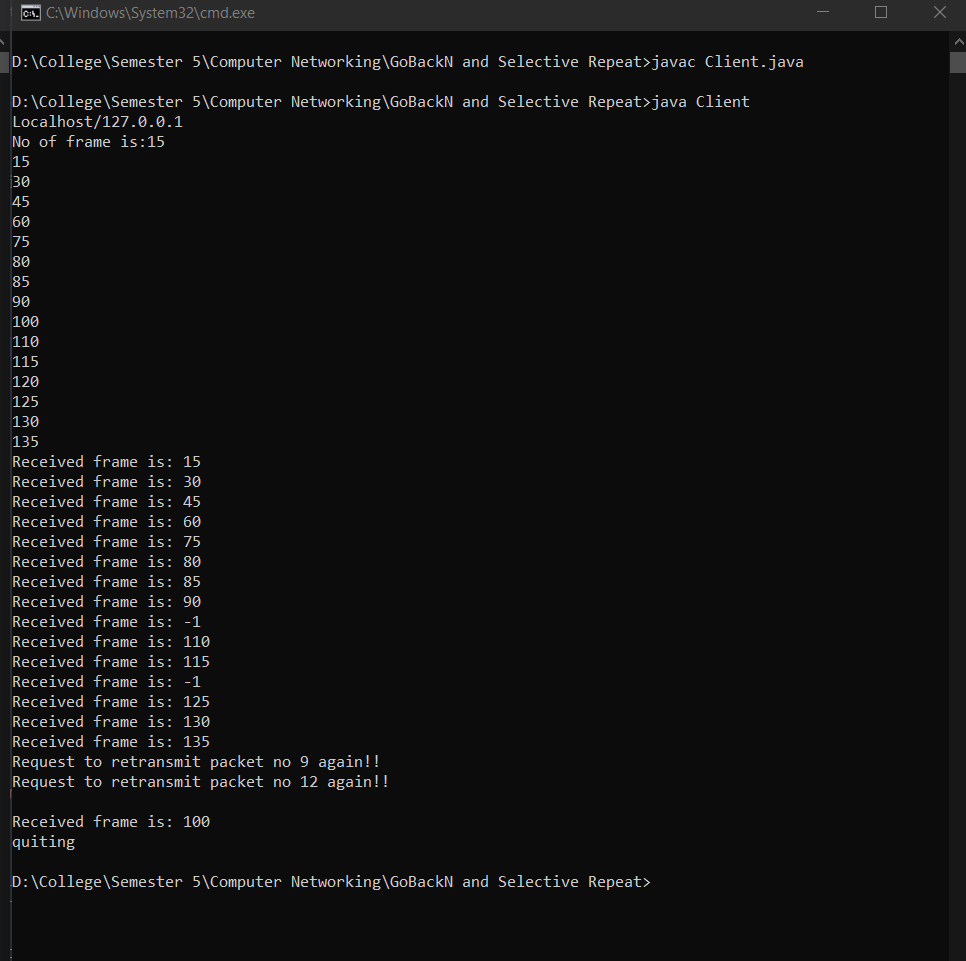
**Output 1**



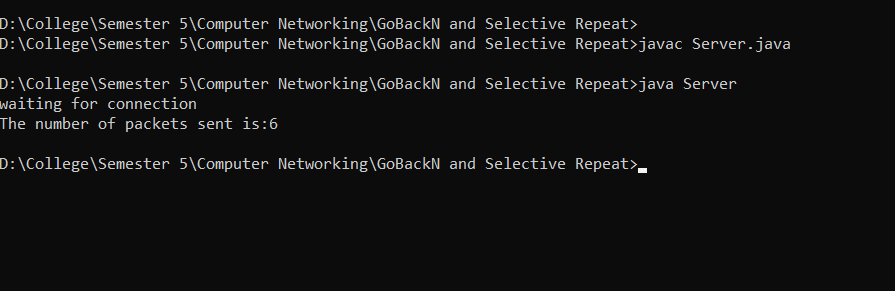
****

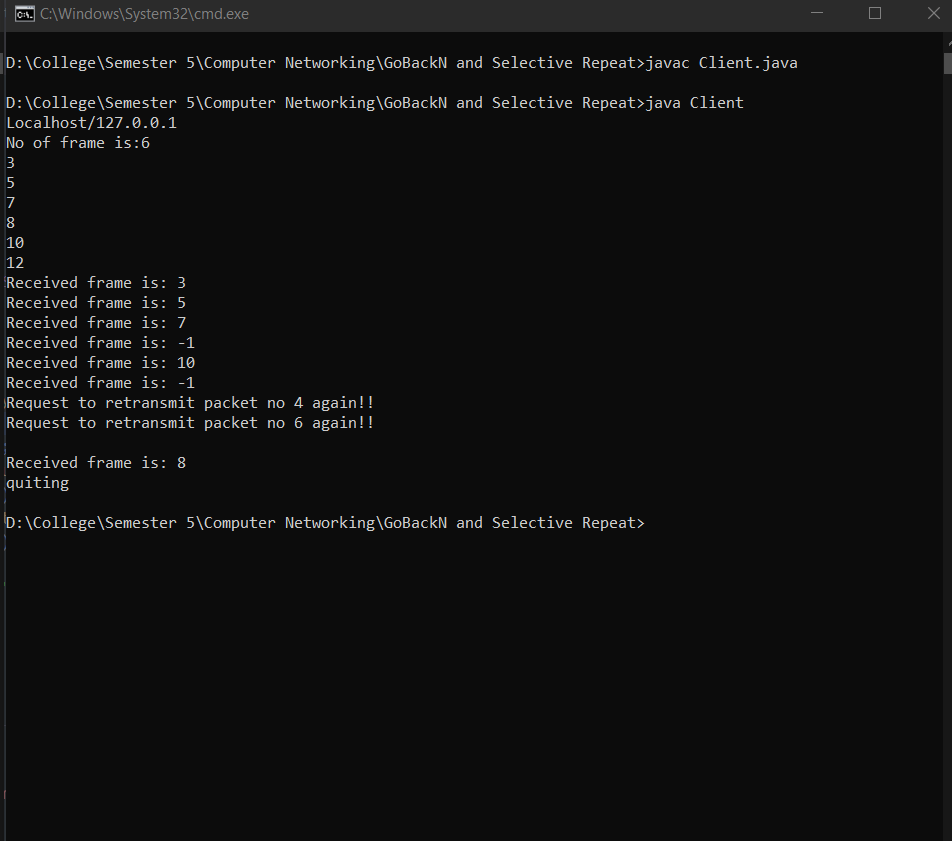
**Output 2**

****

****

**Output 3**

****



**Cloud Virtualization**

Virtualization enables users to disjoint operating systems from the underlying hardware, i.e, users can run multiple operating systems such as Windows, Linux, on a single physical machine at the same time. In the context of cloud computing, virtualization is a technique that makes a virtual ecosystem of storage devices and the server OS.

Cloud virtualization transforms the traditional computing methods such that the workload management is more efficient, economic and scalable. Since cloud computing is being considered as a service or an application, assisting a virtualized ecosystem that could either be private or public, so with virtualization, resources could be escalated, reducing the necessity for a physical system.

**Benefits of Cloud Virtualization Include:**

* Security
* Flexible Operations
* Economical
* Reduced system failures
* Flexible data transfer