ASSIGNMENT 1

Name: Imon Raj
Class: BCSE III
Roll: 002010501098
Section: A3
Subject: Computer Networks Lab Report

PROBLEM STATEMENT: Design and implement an error detection module which has four schemes namely LRC, VRC, Checksum and CRC. Please note that you may need to use these schemes separately for other applications (assignments). You can write the program in any language. The Sender program should accept the name of a test file (contains a sequence of 0,1) from the command line. Then it will prepare the data frame (decide the size of the frame) from the input. Based on the schemes, codeword will be prepared. Sender will send the codeword to the Receiver. Receiver will extract the dataword from codeword and show if there is any error detected. Test the same program to produce a PASS/FAIL result for following cases (not limited to).

DESIGN: In this assignment we are going to implement different error detection modules. It will have functionalities of LRC, VRC, CRC, CHECKSUM techniques.

We will use Java as a programming language to create the modules of this assignment. We will use Object Oriented Programming approach to solve the problems.

Basically, we will have three important classes – **Sender**, **Receiver**, **Channel** and one helper class **BinaryArithmetic.**

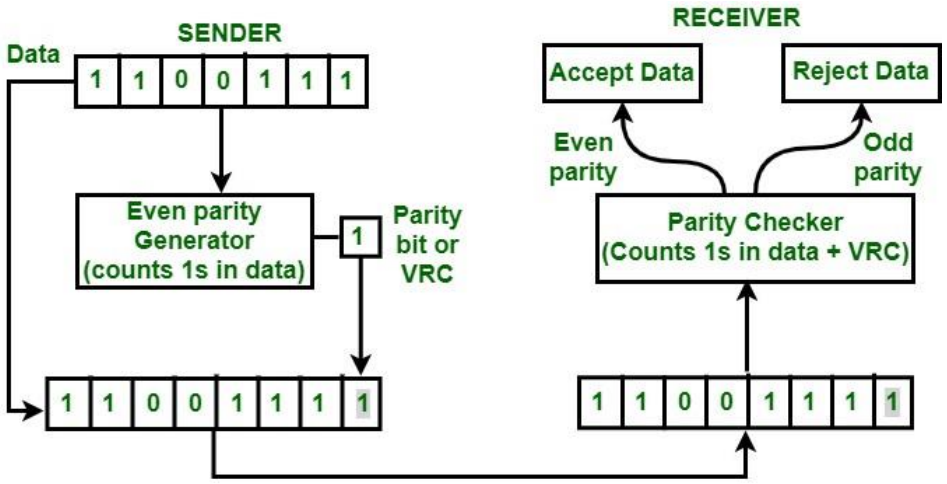
- **Sender** class will have different functions for loading the raw data from a specified file, creating codewords using CRC, LRC, VRC, CheckSum methods and sending them to the channel.
- **Channel** class will simulate the transmission impairments by flipping any number of bits in the codeword sent by the Sender. It will use Java's random function to generate the random integer.
- **Receiver** class will have similar methods as the Sender to validate the codeword received from the Channel. If the channel had inserted any error, the receiver may or may not detect it.

DIFFERENT ERROR DETECTION TECHNIQUES:

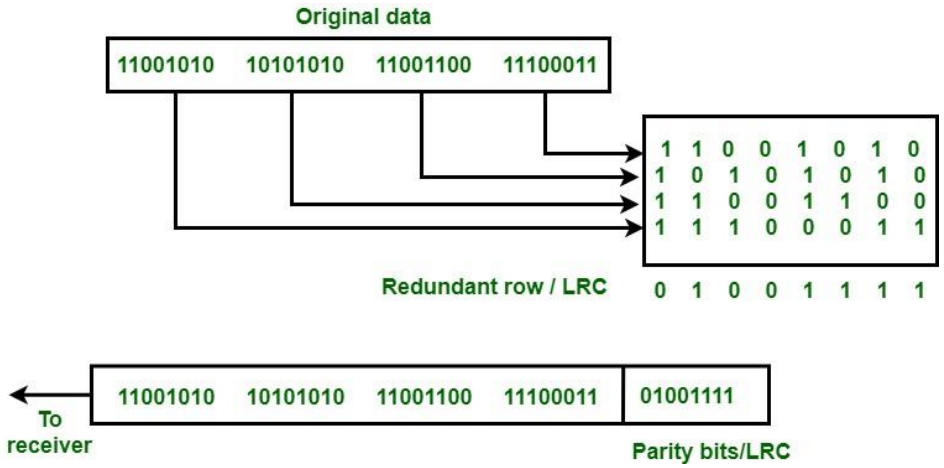
The sender and receiver are bound to maintain certain rules and protocols during data communication. Sender prepares the codeword using any of the techniques and the receiver uses the same rule to detect the error. Now the different techniques are described below briefly –

1. Vertical Redundancy Check(VRC): In this error detection technique, a redundant bit called parity bit is appended to every data unit so that

total number of 1's in the unit (including parity bit) becomes even. At the receiver, all received bits are checked through even parity checking function. If it counts even 1's data unit passes. If it counts odd number of 1's, it means error has been introduced in the data somewhere. Hence receiver rejects the whole data unit.



2. Longitudinal Redundancy Check(LRC): In this error detection technique, the sender divides the complete dataword into multiple packets of same size and then, putting them one below another, parity bits are calculated which finally generates a parity group same as data packet size and this extra packet is then added to dataword to form the codeword. The receiver follows the same process and if it gets all zeros then it detects no error, otherwise there was error during transmission.



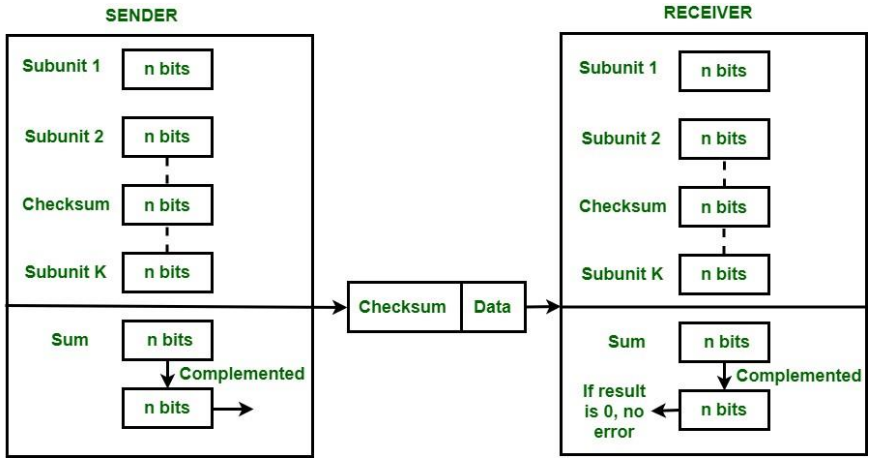
3. Cyclic Redundancy Check(CRC): This is another error detection technique used during data communication. CRC uses Generator Polynomial which is available on both sender and receiver side. An example generator polynomial is of the form like x3 + x + 1. This generator polynomial represents key 1011. Then we append n zero bits at the end of the dataword where n is the degree of the generator polynomial. After that, modulus 2 division is done to this bit sequence to collect the final remainder, which is finally appended to the dataword to create the codeword to be transferred. Receiver uses the same technique to find the final remainder

and if the remainder has all zero bits, then no error is detected and otherwise there is some error in the received bit sequence and it is rejected.

Example – Consider data – 100100 and Polynomial – 1101, final code to be sent - 100100001

```
          111101
       ____________
1101  | 100100000
        1101
        ----
         1000
         1101
         ----
          1010
          1101
          ----
           1110
           1101
           ----
            0110
            0000
            ----
             1100
             1101
             ----
              001
              ---
```

4. CheckSum: This is another way of adding redundancy to the end of dataword for error detection purpose. In this process dataword is divided into list of packets of a pre-defined same size and then they are added in binary arithmetic, any carry generated during this process is added to the sum. The final sum is complemented using one's complement and this is the checksum for that dataword. This is appended to the dataword to send. Receiver uses the same way and if it gets a checksum with all zero bits then no error is detected and if not, then there is error in the received code and thereby rejected.



IMPLEMENTATION(Various Functions Used):

loadData(): Used to load any binary data from file using java.io.*

.

createCheckSumData() & **validateCheckSumData():** First one used in sender class and second one at Receiver class. Both does the similar work of calculating the checksum given a binary sequence and data-packet size. Firstly the whole binary sequence is divided into a group of data-packet-sized strings. This list of binary strings are then added using binary arithmetic in the BinaryArithmetic class.

BinaryArithmetic class contains two necessary functions for this operation – addValues() and addTwoValues(). The answer returned by the addValues() function is complemented to finalize the checksum. In case of Sender it just appends the checksum to the data and in case of Receiver it validates the checksum to detect any error.

createCRCdata() & validateCRCdata(): First one is present in Sender class and second one is at Receiver class. Both does the work of calculating CRC, where polynomial is given. For the division, it again uses a function from BinaryArithmetic class which takes divisor and dividend(appended zeros to the actual sequence), does the division and finally returns the remainder which is nothing but the CRC for a given binary sequence. Sender appends this CRC to raw data and Receiver in its case validates the CRC to detect any error.

createLRCdata() & validateLRCdata(): Both does the work of calculating LRC for a given binary sequence. Binary sequence is divided into multiple groups of same size as the decided data-packet size. Considering them putting one below another, parity bits are calculated longitudinally, taking even parity as convention. Finally, we get the LRC. LRC used by Sender to append to the data and, Receiver validates the calculated LRC using even-parity only.

createVRCdata() & validateVRCdata(): This scheme implementation is easiest of all. One parity bit is added to every data packet considering the even parity in that packet. Receiver also validates each packet whether even parity is maintained or not.

introduceRandomError(): This function present in Channel class simulates the noise or impairments which are present in the actual data-communication scenario. It takes a random number of errors to be inserted and those many errors are inserted into the binary sequence in random indexes by flipping the bits. This function internally uses a function randomInteger() which returns a random integer within the given start and end points.

xOr(): Present in BinaryArithmetic class. It does xOr of two bits.

addTwoValues(): Present in BinaryArithmetic class. Adds two binary strings and if any carry is generated added to the sum to get the final answer.

addValues(): This is a function to add list of binary strings using the function mentioned above.

Comments:

This assignment indirectly helped me to polish my concepts of OOP in Java. Instead of files, socket or pipe programming could be used to implement the simulation for data communication.

Single bit errors can be detected by all the schemes. In case of burst error schemes may or may not detect them. As an example – 10110 is data then for VRC parity will be 1 so sent code will be 101101 and if two bits flip to become 001111, then this error couldn't be detected by VRC. Similarly – for LRC, if multiple bits in same longitudinal line flip in such a way that evenparity doesn't violates then, no error is detected. In CRC, if width of bits flipped is more than the degree of polynomial then, error may or may not be detected. In case of checksum, suppose two data packets increased and decreased by same amount because of flipped bits then, no error is detected by checksum.

ASSIGNMENT 2

Name: Imon Raj
Class: BCSE III
Roll: 002010501098
Section: A3
Subject: Computer Networks Lab Report

PROBLEM STATEMENT: Implement three data link layer protocols, Stop and Wait, Go Back N Sliding Window and Selective Repeat Sliding Window for flow control.

Sender, Receiver and Channel all are independent processes. There may be multiple Transmitter and Receiver processes, but only one Channel process. The channel process introduces random delay and/or bit error while transferring frames. Define your own frame format or you may use IEEE 802.3 Ethernet frame format.

DESIGN: This problem was quite interesting and hard to implement. For this assignment I have used Object-oriented approach with Java. In places, I have extensively used Multithreading of Java. For data sending and receiving purpose Socket is used. As it is an assignment on FlowControl, I haven't sent any actual binary data, rather I am only bothered about sending frames. Now, let me share my design approaches –

Stop and Wait: The general theory of this protocol says –

Sender sends frames one by one. After sending one packet sender must wait for the acknowledgement of that packet from the receiver. Sender will wait only for a previously decided timeout period, if no acknowledgement is got, sender will resend the packet, supposing that either packet or acknowledgement is lost. In this way sender continues. Sender assigns consecutive 0 and 1 as sequence numbers to the frames. That means if acknowledgement for 0 is not got, it will resend 0 and so on.

In case of receiver, if it gets a frame, it will instantly send an acknowledgement. It will expect frames in alternative sequence of 0 and 1. If it gets one frame twice it will send the acknowledgement again, as it means that previous acknowledgement is lost.

For implementation, the sender and receiver , each of them runs in one thread. Sender sends a frame to the socket and waits for acknowledgement from the receiver. The receiver sometimes sends acknowledgement for the frame, otherwise it sends no acknowledgement behaving as if it hasn't got the frame. We have used Java random int function to decide whether to send acknowledgement or not.

Go Back N ARQ: This protocol is a little complicated –

Sender at a time sends N number of frames and always maintains a N size window. As soon as it receives acknowledgement for the first frame of the window, it shifts the window by one position, and sends one new frame

maintaining N-sized window. But if after a timeout, it receives no acknowledgement, it will resend the all the frames of the window again.

Receiver sends acknowledgement after receiving each frame. If one frame is sent repeatedly, receiver sends the acknowledgement again, because probably sender is resending the complete window again.

Here for implementation, sender has two threads running concurrently- one for frame for sending frames, one for receiving acknowledgement. Receiver has one thread only. Receiver, here also, sends acknowledgements with a random probability. There are two sockets at two different ports. One socket is to send frames by sender, another to send acknowledgements by receiver.

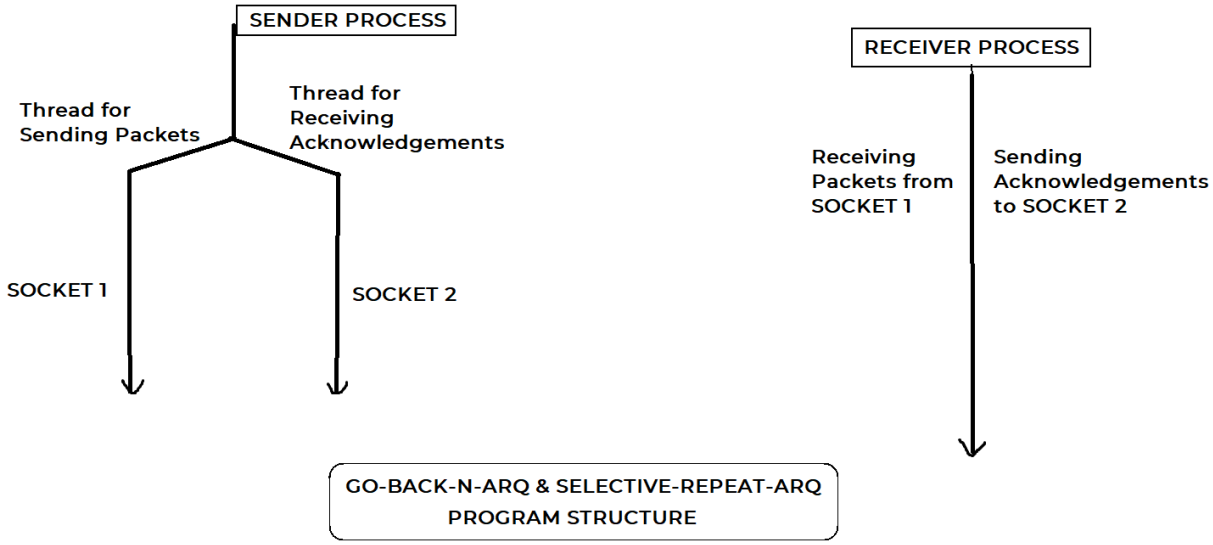
Selective Repeat ARQ: This is similar to the previous one. Here also a window is maintained of size N, but here if acknowledgement of first frame of window is not received by sender, sender only sends those frames within the window whose acknowledgements are not at all received.

Receiver sends an acknowledgement after receiving a frame. If one frame is sent twice, receiver sends acknowledgement again.

For implementation, most things will be same as the previous protocol. The change will be in resending frames. In this case, only unacknowledged frames are retransmitted after a timeout.

IMPLEMENTATION:

The threading and socket structure of Go-Back-N-Arq and Selective-Repeat-Arq is given below. As, Stop-and-wait is so simple, with sender-receiver each using one thread and one common socket, its figure is avoided.



GO-BACK-N-ARQ & SELECTIVE-REPEAT-ARQ PROGRAM STRUCTURE

All related codes written in Java are attached herewith (In the Drive-shared-folder).

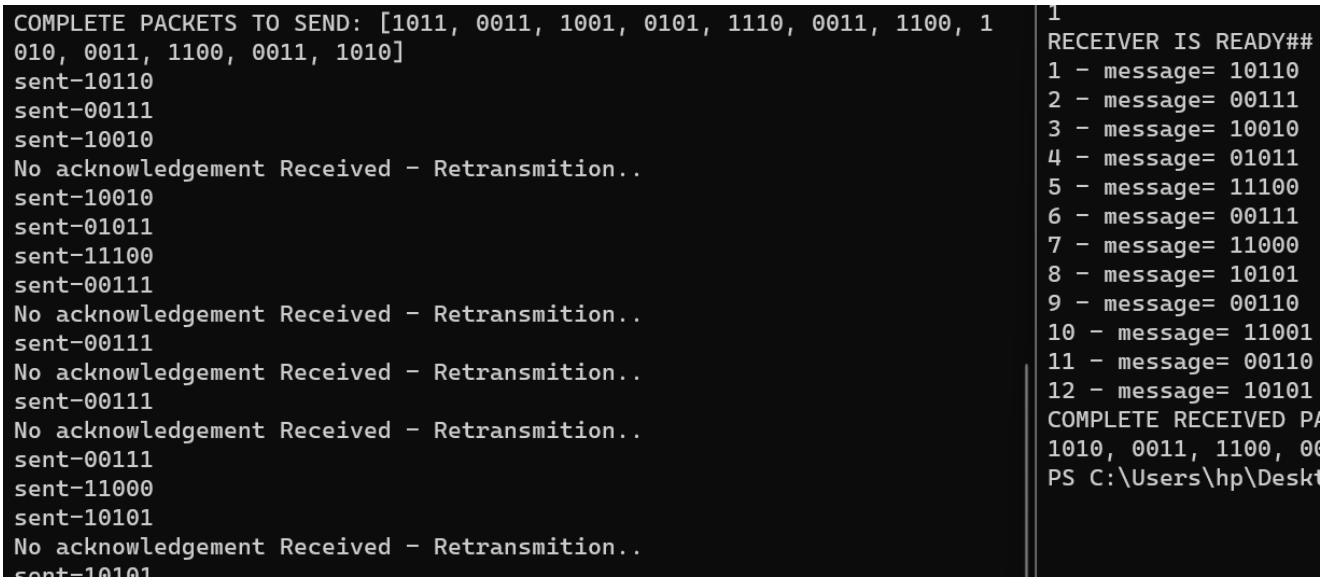
TEST CASES:

- There shouldn't be any Deadlock condition, i.e-> In the multithreaded environment, No thread of either Sender or Receiver should wait indefinitely.
- The Program should correctly terminate, i.e-> All sockets used must be closed properly, there should not be any crash in the program.
- After all, the frame sending sequences should follow the rules of the protocols.

RESULTS & ANALYSIS:

Program can fulfil all the test cases conditions provided above. Given randomness in acknowledgement sending, different sequence of output is being generated, but all are correct. However, I can see improvement possibilities in the acknowledgement procedure.

SAMPLE OUTPUT(STOP AND WAIT)



SAMPLE OUTPUT(selective repeat)

```
PS C:\Users\hp\Desktop\3rd_year_lab\COMPUTER NETWORKS\ASSIGNMENT_2> javac Main.java
PS C:\Users\hp\Desktop\3rd_year_lab\COMPUTER NETWORKS\ASSIGNMENT_2> java Main
1
==========SELECTIVE REPEAT==========
Frame 1: 0011
ack sent for frame: 1
Frame 3: 0101
ack sent for frame: 3
Frame 0: 1011
ack sent for frame: 0
Frame 2: 1001
ack sent for frame: 2
Frame 5: 0011
ack sent for frame: 5
Frame 7: 1010
ack sent for frame: 7
Frame 4: 1110
ack sent for frame: 4
Frame 6: 1100
ack sent for frame: 6
Frame 8: 0011
ack sent for frame: 8
Frame 9: 1100
ack sent for frame: 9
Frame 10: 0011
ack sent for frame: 10
Frame 11: 1010
ack sent for frame: 11
PS C:\Users\hp\Desktop\3rd_year_lab\COMPUTER NETWORKS\ASSIGNMENT_2>
```

```
COMPLETE PACKETS TO SEND: [1011, 001
010, 0011, 1100, 0011, 1010]
Sent Frame- 0
Sent Frame- 1
Ack Get - 1
Sent Frame- 2
Sent Frame- 3
Ack Get - 3
Timeout Resending Frame#
Sent Frame- 0
Sent Frame- 2
Timeout Resending Frame#
Sent Frame- 0
Ack Get - 0
Sent Frame- 2
Ack Get - 2
Sent Frame- 4
Sent Frame- 5
Ack Get - 5
Sent Frame- 6
Sent Frame- 7
Ack Get - 7
Timeout Resending Frame#
Sent Frame- 4
Ack Get - 4
Sent Frame- 6
Ack Get - 6
Sent Frame- 8
Ack Get - 8
Sent Frame- 9
Ack Get - 9
```

Output of GO-back-N is somewhat similar to upper one.

COMMENTS:

This assignment took a lot of time for me to complete. The Stop-and-wait was simple but, the other two protocols' implementation made me think deeply about various approaches. I faced bug few times, and was able to fix them one by one. This assignment helped to polish my concepts on multithreading and synchronization techniques. According to me, this assignment was a little hard but yes, it was interesting to complete it.

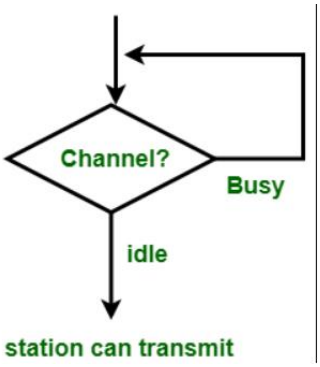
ASSIGNMENT 3

Name: Imon Raj
Class: BCSE III
Roll: 002010501098
Section: A3
Subject: Computer Networks Lab Report

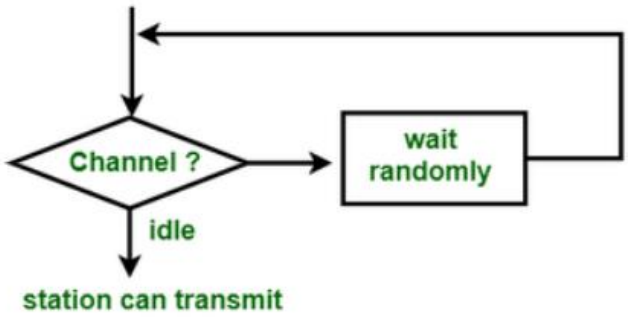
PROBLEM STATEMENT: In this assignment, you have to implement 1-persistent, non-persistent and p-persistent CSMA techniques. Measure the performance parameters like throughput (i.e., average amount of data bits successfully transmitted per unit time) and forwarding delay (i.e., average end-to-end delay, including the queuing delay and the transmission delay) experienced by the CSMA frames (IEEE 802.3). Plot the comparison graphs for throughput and forwarding delay by varying p. State your observations on the impact of performance of different CSMA techniques.

DESIGN: CSMA stands for Carrier Sense Multiple Access. It is one of the randomized MAC layer protocols. Like other MAC layer protocols, it is used in situations where multiple stations are using shared media. No station is superior to another. Any station can send frames whenever it wants, with some restrictions to be followed. Carrier sense indicates that a station must sense the medium before sending a frame. It can only send(using some persistence methods) a frame if the medium is idle. Carrier sensing reduces the chance of collision and improves throughput. There are three types of persistence methods for CSMA.

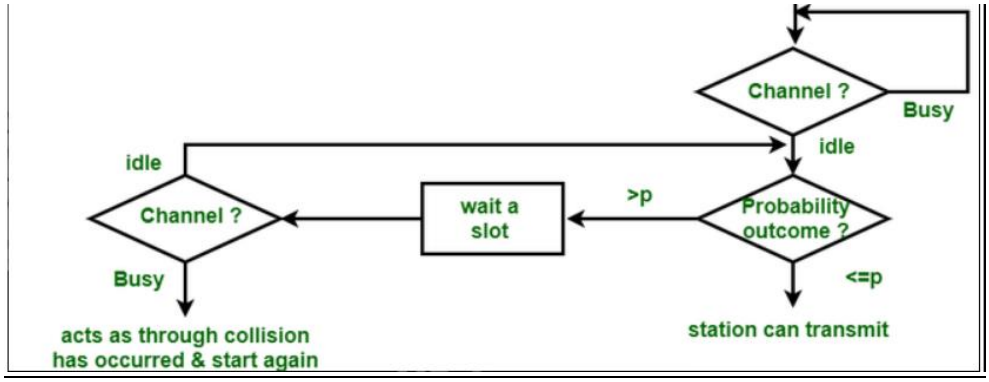
1. One-persistent: The station continuously senses the channel to check its state i.e. idle or busy so that it can transfer data. In case the channel is busy, the station will wait for the channel to become idle. When the channel is found idle, it transmits the frame to the channel without any delay. It transmits the frame with probability 1. Due to probability 1, it is called 1-persistent CSMA. The problem with this method is that there are a large number of chances for a collision to occur because two or more stations might find the channel in an idle state and the transmit frames at the same time. When a collision occurs the station has to wait for back-off time duration and then sense again.



2. Non-persistent: When there is a frame to send, a station senses the medium. If the channel is idle, station will immediately send frame. In case the channel is busy, it will wait for the random time and again senses the station whether idle or busy. In this method, the station does not continuously sense the channel for only the purpose of capturing it when it detects the end of the previous transmission. The main advantage of using this method is that it reduces the chances of collision. The problem with this is that it reduces the efficiency of the network. It may so happen that channel is completely idle but no station is sending, all are waiting, causes underutilization of the channel.



3. P-persistent: This is the method that is used when channel has time-slots and that time-slot duration is equal to or greater than the maximum propagation delay time. When the station is ready to send the frames, it will sense the channel. If the channel found to be busy, the channel will wait for the next slot. If the channel found to be idle, it transmits the frame with probability p, thus for the left probability i.e. q which is equal to 1-p the station will wait for the beginning of the next time slot. In case, when the next slot is also found idle it will transmit or wait again with the probabilities p and q. This process is repeated until either the frame gets transmitted or another station has started transmitting.



IMPLEMENTATION:

For the implementation of the three persistence methods of CSMA, I have used Java as a programming language. There will be two classes – Channel and Station. The Station class extends the Thread class so that multiple stations can run parallelly. Station class will have three methods for three persistence rules. The overridden run() method will execute one of these persistence methods. The Channel class will have a variable isBusy(to check whether channel is busy), a Semaphore mutex(to synchronize the access and modification of the isBusy).

All related codes written in Java are attached herewith (In the Drive-shared-folder).

TEST CASES:

The program should be able to assure that –

- No station should send when another is sending.
- The value of isBusy variable is consistent(synchronized).
- The persistence methods are obeyed properly by all stations.

RESULTS & ANALYSIS:

Program can fulfil all the test cases provided above. Currently, I have not found any bug, as the program is not very complicated. Snapshots of outputs of the three persistence methods are given below.

1-PERSISTENT:

```
Station 0 starts to sense medium..
Station 0 finds medium IDLE.. Medium Acquired..
Station 0 sends Frame..
Station 3 starts to sense medium..
Station 0 releases Medium..
==
Station 3 finds medium IDLE.. Medium Acquired..
Station 3 sends Frame..
Station 1 starts to sense medium..
Station 3 releases Medium..
==
Station 1 finds medium IDLE.. Medium Acquired..
Station 1 sends Frame..
Station 2 starts to sense medium..
Station 4 starts to sense medium..
Station 1 releases Medium..
==
Station 2 finds medium IDLE.. Medium Acquired..
Station 2 sends Frame..
Station 2 releases Medium..
==
Station 4 finds medium IDLE.. Medium Acquired..
Station 4 sends Frame..
Station 3 starts to sense medium..
Station 0 starts to sense medium..
Station 4 releases Medium..
```

NON-PERSISTENT:

```
Station 1 senses medium..
Station 1 finds medium IDLE.. Medium Acquired..
Station 1 sends Frame..
Station 2 senses medium..
Station 2 finds medium busy.. waits randomly..
Station 1 releases Medium..
==
Station 0 senses medium..
Station 0 finds medium IDLE.. Medium Acquired..
Station 0 sends Frame..
Station 4 senses medium..
Station 4 finds medium busy.. waits randomly..
Station 0 releases Medium..
==
Station 3 senses medium..
Station 3 finds medium IDLE.. Medium Acquired..
Station 3 sends Frame..
Station 2 senses medium..
Station 2 finds medium busy.. waits randomly..
Station 3 releases Medium..
==
Station 4 senses medium..
Station 4 finds medium IDLE.. Medium Acquired..
Station 4 sends Frame..
Station 4 releases Medium..
==
```

P-PERSISTENT:

```
Station 0 starts to sense medium..
Station 0 finds medium IDLE. generates -> (0.11) -> WILL SEND ACQUIRE MEDIUM..
Station 0 sends Frame..
Station 1 starts to sense medium..
Station 4 starts to sense medium..
Station 3 starts to sense medium..
Station 0 releases Medium..
==
Station 4 finds medium IDLE. generates -> (0.77) -> WILL NOT SEND..
Station 1 finds medium IDLE. generates -> (0.54) -> WILL NOT SEND..
Station 4 waits backoff..
Station 1 waits backoff..
Station 3 finds medium IDLE. generates -> (0.28) -> WILL NOT SEND..
Station 3 waits backoff..
Station 2 starts to sense medium..
Station 2 finds medium IDLE. generates -> (0.10) -> WILL SEND ACQUIRE MEDIUM..
Station 2 sends Frame..
Station 2 releases Medium..
==
Station 1 starts to sense medium..
Station 1 finds medium IDLE. generates -> (0.95) -> WILL NOT SEND..
Station 1 waits backoff..
Station 0 starts to sense medium..
Station 0 finds medium IDLE. generates -> (0.56) -> WILL NOT SEND..
Station 0 waits backoff..
Station 4 starts to sense medium..
Station 4 finds medium IDLE. generates -> (0.51) -> WILL NOT SEND..
Station 4 waits backoff..
Station 3 starts to sense medium..
```

I can see some improvement possibilities which are not present in my code. In my code, there is a global isBusy variable, but actually a station can only sense the medium in its point. Also in my code, no actual packet is being sent. My code is more about – station acquiring the medium, holding for sometimes and then release. In these mentioned topics, improvement can be done.

COMMENTS:

This assignment was not that lengthy. Like all the other assignments, it also encouraged me to think of an efficient solution. And coding this assignment was mostly fun and interesting.

ASSIGNMENT 4

Name: Imon Raj
Class: BCSE III
Roll: 002010501098
Section: A3
Subject: Computer Networks Lab Report

PROBLEM STATEMENT: In this assignment you have to implement CDMA for multiple access of a common channel by n stations. Each sender uses a unique code word, given by the Walsh set, to encode its data, send it across the channel, and then perfectly reconstruct the data at n stations

DESIGN: This problem was quite easy to understand and implement. CDMA stands for Code Division Multiple Access and it is one of the MAC protocols. Here all n stations in a shared media can use the complete bandwidth anytime. Here the trick is that, each station will have a code associated with it. Any two such codes will be orthogonal to each other. These codes for stations can be generated using WalshTable.

The station which sends a bit, will do a calculation. It will consider 0 bit as -1 and 1 as 1. Then, if it's code is C_i, then C_i is multiplied with the bit value. The generated result is sent as signal sequence onto the media. If any $station_i$ wants to receive data from $station_j$ then, $station_i$ will get signal values from the media and multiply then with the code C_j. Then the result is divided by no of stations to get the bit sent by $station_j$.

Generation of Walsh Table is interesting. We will start with a (2 x 2) walshTable and populate it to our desired size. One thing to note that, for any given station count n, we have to calculate a K>=n such that K=2^p. Now we will populate the table to size (K x K). Surely, we will have K codes, we will only use n of them for n stations.

IMPLEMENTATION:

All related codes written in Java are attached herewith (In the Drive-shared-folder).

TEST CASES:

- The K for a given n must be calculated properly.
- The system should be able to detect correctly bit 0, 1 and no-bit receiving.
- walshTable generation should be as time efficient as possible.

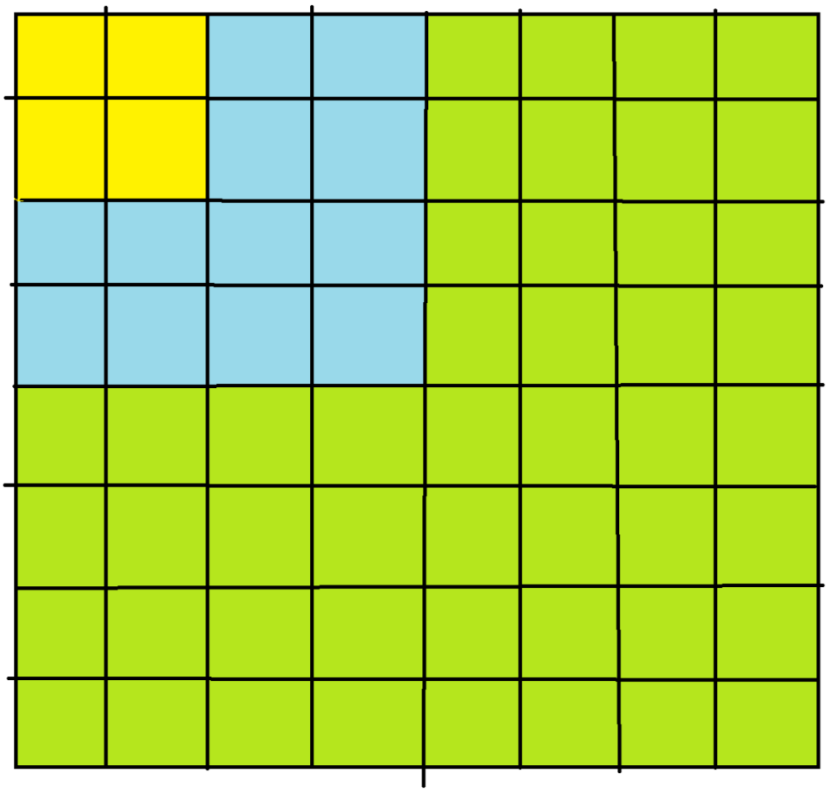
One sample output is given below –

```
Enter the number of stations:
13

----------WALSH TABLE---------
1       1       1       1       1       1       1       1       1       1       1       1       1       1       1       1
1       -1      1       -1      1       -1      1       -1      1       -1      1       -1      1       -1      1       -1
1       1       -1      -1      1       1       -1      -1      1       1       -1      -1      1       1       -1      -1
1       -1      -1      1       1       -1      -1      1       1       -1      -1      1       1       -1      -1      1
1       1       1       1       -1      -1      -1      -1      1       1       1       1       -1      -1      -1      -1
1       -1      1       -1      -1      1       -1      1       1       -1      1       -1      -1      1       -1      1
1       1       -1      -1      -1      -1      1       1       1       1       -1      -1      -1      -1      1       1
1       -1      -1      1       -1      1       1       -1      1       -1      -1      1       -1      1       1       -1
1       1       1       1       1       1       1       1       -1      -1      -1      -1      -1      -1      -1      -1
1       -1      1       -1      1       -1      1       -1      -1      1       -1      1       -1      1       -1      1
1       1       -1      -1      1       1       -1      -1      -1      -1      1       1       -1      -1      1       1
1       -1      -1      1       1       -1      -1      1       -1      1       1       -1      -1      1       1       -1
1       1       1       1       -1      -1      -1      -1      -1      -1      -1      -1      1       1       1       1
---------------------------
Station 0 generates no data.
Station 1 generates no data.
Station 2 generates data bit 1
Station 3 generates data bit 0
Station 4 generates data bit 1
Station 5 generates no data.
Station 6 generates no data.
Station 7 generates no data.
Station 8 generates data bit 1
Station 9 generates data bit 0
Station 10 generates data bit 0
Station 11 generates data bit 0
Station 12 generates data bit 1
---------------------------
Channel Data:
0       6       4       2       -4      2       0       -2      2       0       -2      -4      2       0       -2      -4
---------------------------

Which station you want to see:
4
Data bit from station 4 is : 1
```

RESULTS & ANALYSIS:

Program can fulfil all the test cases conditions provided above. Currently, I haven't found any bug, as the program is not very complicated.

Let's analyse the Time complexity of the walshTable generation function. Notice carefully the image given below –



At the beginning we will have only the yellow portion's code. Using this yellow portion we will calculate all the blue portions, then using blue and yellow we will generate the green portion and so on. Now for any given N(2^p form) time complexity will be –

$T(N) = 3*(4 + 16 + 64 + \ldots\ldots\ldots + N^2)$

$= 3*(2^2 + 4^2 + 8^2 + \ldots\ldots\ldots..+ N^2)$

$= 3*4*(1^2 + 2^2 + 4^2 + \ldots\ldots\ldots..+ (N/2)^2)$

$= 12 * 1 * (4^p - 1)/(4 - 1)$ [put N = 2^p]

$= 4 * (4^p - 1)$

$= 4 * (N^2 - 1)$

$= \mathbf{O(N^2)}$

So, we can see the time complexity is O(N^2). We can even think easily that each slot in that matrix is calculated once, so we have total N*N operations, which leads us to the same result.

COMMENTS:

This assignment was not that lengthy. Like all the other assignments, it also encouraged me to think of an efficient solution. And coding this assignment was mostly fun and interesting.

ASSIGNMENT 5

Name: Imon Raj
Class: BCSE III
Roll: 002010501098
Section: A3
Subject: Computer Networks Lab Report

PROBLEM STATEMENT: **Install Wireshark in local machine and do traffic and packet analysis according to the given questions**

QUESTIONS:

1.) Generate some ICMP traffic by using the Ping command line tool to check the connectivity of a neighbouring machine (or router). Note the results in Wireshark. The initial ARP request broadcast from your PC determines the physical MAC address of the network IP Address, and the ARP reply from the neighbouring system. After the ARP request, the pings (ICMP echo request and replies) can be seen.

Answer:

```
PS C:\Users\hp> ping 192.168.29.27

Pinging 192.168.29.27 with 32 bytes of data:
Reply from 192.168.29.27: bytes=32 time=253ms TTL=64
Reply from 192.168.29.27: bytes=32 time=288ms TTL=64
Reply from 192.168.29.27: bytes=32 time=85ms TTL=64
Reply from 192.168.29.27: bytes=32 time=103ms TTL=64

Ping statistics for 192.168.29.27:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 85ms, Maximum = 288ms, Average = 182ms
PS C:\Users\hp> ping 192.168.29.27

Pinging 192.168.29.27 with 32 bytes of data:
Reply from 192.168.29.27: bytes=32 time=195ms TTL=64
Reply from 192.168.29.27: bytes=32 time=213ms TTL=64
Reply from 192.168.29.27: bytes=32 time=234ms TTL=64
Reply from 192.168.29.27: bytes=32 time=245ms TTL=64

Ping statistics for 192.168.29.27:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 195ms, Maximum = 245ms, Average = 221ms
```

Source	Destination	Protocol	Length	Info
AzureWav_68:10:67	b2:2a:7f:38:2b:17	ARP	42	Who has 192.168.29.27? Tell 192.168.29.243
b2:2a:7f:38:2b:17	AzureWav_68:10:67	ARP	42	192.168.29.27 is at b2:2a:7f:38:2b:17

No.	Time	Source	Destination	Protocol	Length	Info
21	0.7192…	192.168.29.243	192.168.29.27	ICMP	74	Echo (ping) request id=0x0001, seq=5/1280, ttl=128 (reply in 61)
61	0.9728…	192.168.29.27	192.168.29.243	ICMP	74	Echo (ping) reply id=0x0001, seq=5/1280, ttl=64 (request in 21)
138	1.7313…	192.168.29.243	192.168.29.27	ICMP	74	Echo (ping) request id=0x0001, seq=6/1536, ttl=128 (reply in 139)
139	2.0193…	192.168.29.27	192.168.29.243	ICMP	74	Echo (ping) reply id=0x0001, seq=6/1536, ttl=64 (request in 138)
140	2.7388…	192.168.29.243	192.168.29.27	ICMP	74	Echo (ping) request id=0x0001, seq=7/1792, ttl=128 (reply in 141)
141	2.8241…	192.168.29.27	192.168.29.243	ICMP	74	Echo (ping) reply id=0x0001, seq=7/1792, ttl=64 (request in 140)
142	3.7451…	192.168.29.243	192.168.29.27	ICMP	74	Echo (ping) request id=0x0001, seq=8/2048, ttl=128 (reply in 143)
143	3.8483…	192.168.29.27	192.168.29.243	ICMP	74	Echo (ping) reply id=0x0001, seq=8/2048, ttl=64 (request in 142)
148	5.9035…	192.168.29.243	192.168.29.27	ICMP	74	Echo (ping) request id=0x0001, seq=9/2304, ttl=128 (reply in 149)
149	6.0983…	192.168.29.27	192.168.29.243	ICMP	74	Echo (ping) reply id=0x0001, seq=9/2304, ttl=64 (request in 148)
150	6.9099…	192.168.29.243	192.168.29.27	ICMP	74	Echo (ping) request id=0x0001, seq=10/2560, ttl=128 (reply in 151)
151	7.1225…	192.168.29.27	192.168.29.243	ICMP	74	Echo (ping) reply id=0x0001, seq=10/2560, ttl=64 (request in 150)
152	7.9147…	192.168.29.243	192.168.29.27	ICMP	74	Echo (ping) request id=0x0001, seq=11/2816, ttl=128 (reply in 153)
153	8.1486…	192.168.29.27	192.168.29.243	ICMP	74	Echo (ping) reply id=0x0001, seq=11/2816, ttl=64 (request in 152)
157	8.9197…	192.168.29.243	192.168.29.27	ICMP	74	Echo (ping) request id=0x0001, seq=12/3072, ttl=128 (reply in 161)
161	9.1645…	192.168.29.27	192.168.29.243	ICMP	74	Echo (ping) reply id=0x0001, seq=12/3072, ttl=64 (request in 157)

2.) Generate some web traffic and

a. find the list the different protocols that appear in the protocol column in the unfiltered packet-listing window of Wireshark.

Ans:

192.168.29.243	136.232.79.144	HTTP	586	GET /jums_exam/misc/footer.js
136.232.79.144	192.168.29.243	HTTP	384	HTTP/1.1 200 OK (text/html)
192.168.29.243	136.232.79.144	TCP	54	2942 → 80 [ACK] Seq=1050 Ack=
85.14.245.45	192.168.29.243	TCP	66	443 → 2940 [SYN, ACK] Seq=0 A
192.168.29.243	85.14.245.45	TCP	54	2940 → 443 [ACK] Seq=1 Ack=1
192.168.29.243	136.232.79.144	TCP	54	2939 → 80 [ACK] Seq=1680 Ack=
192.168.29.243	85.14.245.45	TLSv1.3	571	Client Hello
85.14.245.45	192.168.29.243	TCP	54	443 → 2940 [ACK] Seq=1 Ack=51
85.14.245.45	192.168.29.243	TLSv1.3	298	Server Hello, Change Cipher S

The different protocols I can see are – HTTP, TCP, TLS, TLSv1.3 etc.

b. How long did it take from when the HTTP GET message was sent until the HTTP OK reply was received? (By default, the value of the Time column in the packet-listing window is the amount of time, in seconds, since Wireshark tracing began. To display the Time field in time-of-day format, select the Wireshark View pull down menu, then select Time Display Format, then select Time-of-day.)

ans:

Time	Source	Destination	Protocol	Length	Info
12.932066	192.168.29.243	136.232.79.144	HTTP	700	GET /jums_exam/student_odd_2023
13.092242	136.232.79.144	192.168.29.243	HTTP	130	HTTP/1.1 200 OK (text/html)

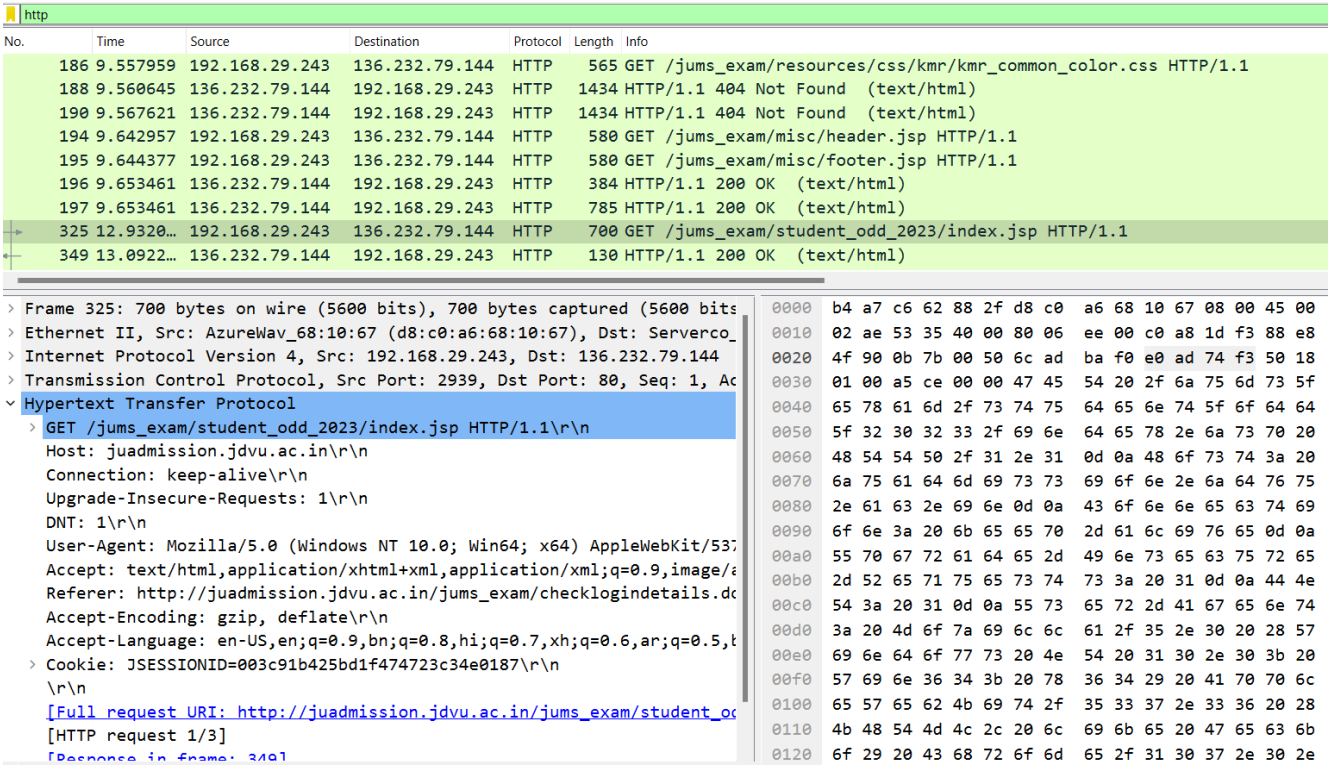
As we can see, that GET message was sent in time 12.932066 and OK response was received in time 13.092242. So the time taken is – (13.092242-12.932066) = 0.160176 seconds.

c. What is the Internet address of the website? What is the Internet address of your computer?

Ans: From the previous screenshot, we can see that the internet(IP) address of the website is – 136.232.79.144 and IP address of my computer is – 192.168.29.243.

d. Search back through your capture, and find an HTTP packet containing a GET command. Click on the packet in the Packet List Panel. Then expand the HTTP layer in the Packet Details Panel, from the packet.

Ans:



e. Find out the value of the Host from the Packet Details Panel, within the GET command.

Ans: The value of host from the previous screenshot is – juadmission.jdvu.ac.in

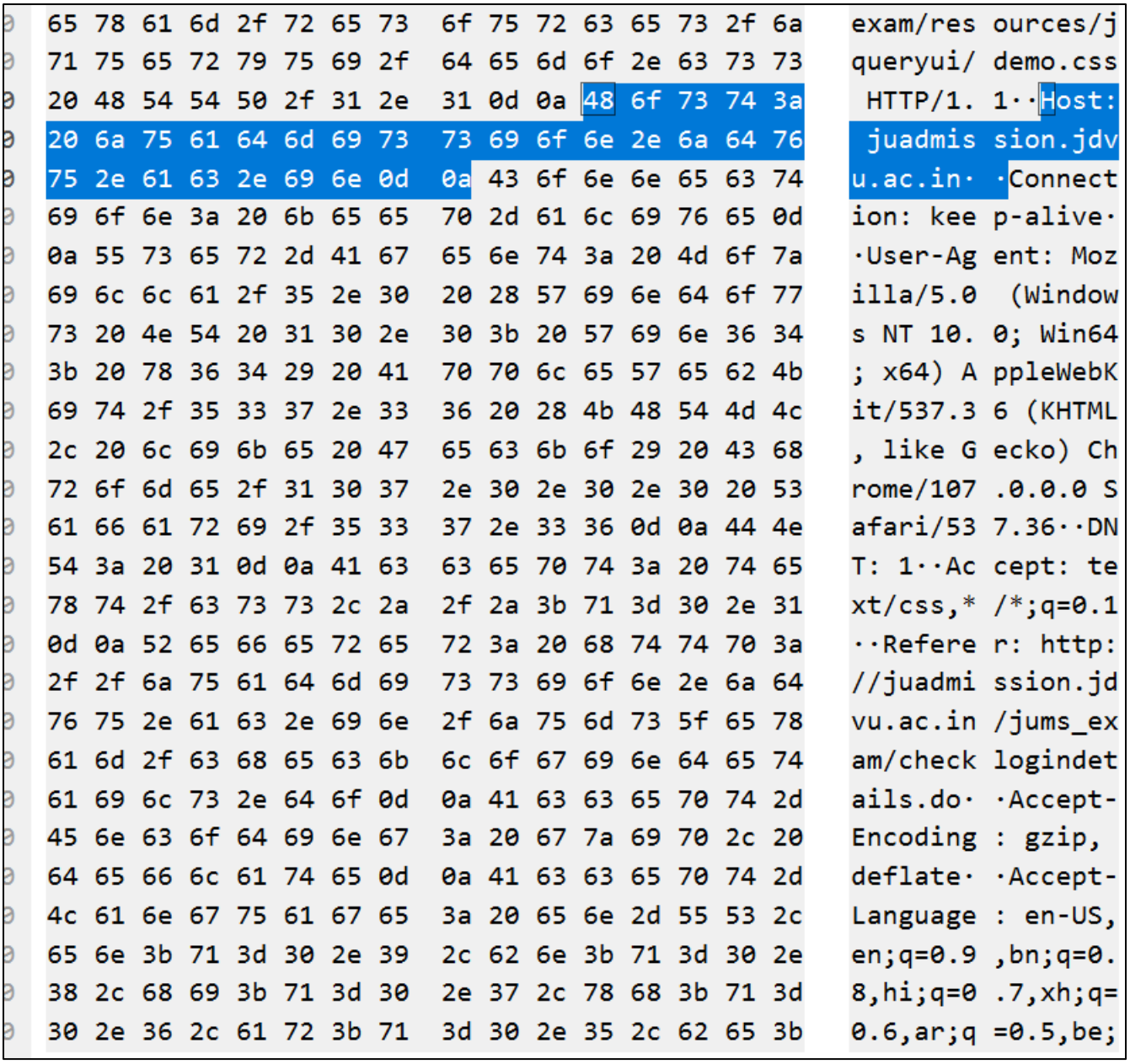
3.) Highlight the Hex and ASCII representations of the packet in the Packet Bytes Panel

ANS:

```
b4 a7 c6 62 88 2f d8 c0  a6 68 10 67 08 00 45 00   ···b·/·· ·h·g··E·
02 1c 53 1f 40 00 80 06  ee a8 c0 a8 1d f3 88 e8   ··S·@··· ········
4f 90 0b 71 00 50 e5 bb  21 09 cb 66 80 f3 50 18   O··q·P·· !··f··P·
01 00 33 27 00 00 47 45  54 20 2f 6a 75 6d 73 5f   ··3'··GE T /jums_
65 78 61 6d 2f 72 65 73  6f 75 72 63 65 73 2f 6a   exam/res ources/j
71 75 65 72 79 75 69 2f  64 65 6d 6f 2e 63 73 73   queryui/ demo.css
20 48 54 54 50 2f 31 2e  31 0d 0a 48 6f 73 74 3a    HTTP/1. 1··Host:
20 6a 75 61 64 6d 69 73  73 69 6f 6e 2e 6a 64 76    juadmis sion.jdv
75 2e 61 63 2e 69 6e 0d  0a 43 6f 6e 6e 65 63 74   u.ac.in· ·Connect
69 6f 6e 3a 20 6b 65 65  70 2d 61 6c 69 76 65 0d   ion: kee p-alive·
0a 55 73 65 72 2d 41 67  65 6e 74 3a 20 4d 6f 7a   ·User-Ag ent: Moz
69 6c 6c 61 2f 35 2e 30  20 28 57 69 6e 64 6f 77   illa/5.0  (Window
73 20 4e 54 20 31 30 2e  30 3b 20 57 69 6e 36 34   s NT 10. 0; Win64
3b 20 78 36 34 29 20 41  70 70 6c 65 57 65 62 4b   ; x64) A ppleWebK
69 74 2f 35 33 37 2e 33  36 20 28 4b 48 54 4d 4c   it/537.3 6 (KHTML
2c 20 6c 69 6b 65 20 47  65 63 6b 6f 29 20 43 68   , like G ecko) Ch
72 6f 6d 65 2f 31 30 37  2e 30 2e 30 2e 30 20 53   rome/107 .0.0.0 S
61 66 61 72 69 2f 35 33  37 2e 33 36 0d 0a 44 4e   afari/53 7.36··DN
54 3a 20 31 0d 0a 41 63  63 65 70 74 3a 20 74 65   T: 1··Ac cept: te
78 74 2f 63 73 73 2c 2a  2f 2a 3b 71 3d 30 2e 31   xt/css,* /*;q=0.1
0d 0a 52 65 66 65 72 65  72 3a 20 68 74 74 70 3a   ··Refere r: http:
2f 2f 6a 75 61 64 6d 69  73 73 69 6f 6e 2e 6a 64   //juadmi ssion.jd
76 75 2e 61 63 2e 69 6e  2f 6a 75 6d 73 5f 65 78   vu.ac.in /jums_ex
61 6d 2f 63 68 65 63 6b  6c 6f 67 69 6e 64 65 74   am/check logindet
61 69 6c 73 2e 64 6f 0d  0a 41 63 63 65 70 74 2d   ails.do· ·Accept-
45 6e 63 6f 64 69 6e 67  3a 20 67 7a 69 70 2c 20   Encoding : gzip,
64 65 66 6c 61 74 65 0d  0a 41 63 63 65 70 74 2d   deflate· ·Accept-
```

4.) Find out the first 4 bytes of the Hex value of the Host parameter from the Packet Bytes Pane

Ans:



```
65 78 61 6d 2f 72 65 73  6f 75 72 63 65 73 2f 6a   exam/res ources/j
71 75 65 72 79 75 69 2f  64 65 6d 6f 2e 63 73 73   queryui/ demo.css
20 48 54 54 50 2f 31 2e  31 0d 0a 48 6f 73 74 3a    HTTP/1. 1··Host:
20 6a 75 61 64 6d 69 73  73 69 6f 6e 2e 6a 64 76   juadmis sion.jdv
75 2e 61 63 2e 69 6e 0d  0a 43 6f 6e 6e 65 63 74   u.ac.in· ·Connect
69 6f 6e 3a 20 6b 65 65  70 2d 61 6c 69 76 65 0d   ion: kee p-alive·
0a 55 73 65 72 2d 41 67  65 6e 74 3a 20 4d 6f 7a   ·User-Ag ent: Moz
69 6c 6c 61 2f 35 2e 30  20 28 57 69 6e 64 6f 77   illa/5.0  (Window
73 20 4e 54 20 31 30 2e  30 3b 20 57 69 6e 36 34   s NT 10. 0; Win64
3b 20 78 36 34 29 20 41  70 70 6c 65 57 65 62 4b   ; x64) A ppleWebK
69 74 2f 35 33 37 2e 33  36 20 28 4b 48 54 4d 4c   it/537.3 6 (KHTML
2c 20 6c 69 6b 65 20 47  65 63 6b 6f 29 20 43 68   , like G ecko) Ch
72 6f 6d 65 2f 31 30 37  2e 30 2e 30 2e 30 20 53   rome/107 .0.0.0 S
61 66 61 72 69 2f 35 33  37 2e 33 36 0d 0a 44 4e   afari/53 7.36··DN
54 3a 20 31 0d 0a 41 63  63 65 70 74 3a 20 74 65   T: 1··Ac cept: te
78 74 2f 63 73 73 2c 2a  2f 2a 3b 71 3d 30 2e 31   xt/css,* /*;q=0.1
0d 0a 52 65 66 65 72 65  72 3a 20 68 74 74 70 3a   ··Refere r: http:
2f 2f 6a 75 61 64 6d 69  73 73 69 6f 6e 2e 6a 64   //juadmi ssion.jd
76 75 2e 61 63 2e 69 6e  2f 6a 75 6d 73 5f 65 78   vu.ac.in /jums_ex
61 6d 2f 63 68 65 63 6b  6c 6f 67 69 6e 64 65 74   am/check logindet
61 69 6c 73 2e 64 6f 0d  0a 41 63 63 65 70 74 2d   ails.do· ·Accept-
45 6e 63 6f 64 69 6e 67  3a 20 67 7a 69 70 2c 20   Encoding : gzip,
64 65 66 6c 61 74 65 0d  0a 41 63 63 65 70 74 2d   deflate· ·Accept-
4c 61 6e 67 75 61 67 65  3a 20 65 6e 2d 55 53 2c   Language : en-US,
65 6e 3b 71 3d 30 2e 39  2c 62 6e 3b 71 3d 30 2e   en;q=0.9 ,bn;q=0.
38 2c 68 69 3b 71 3d 30  2e 37 2c 78 68 3b 71 3d   8,hi;q=0 .7,xh;q=
30 2e 36 2c 61 72 3b 71  3d 30 2e 35 2c 62 65 3b   0.6,ar;q =0.5,be;
```

As we can see first four bytes of the Hex value of the Host parameter is: 48 6f 73 74

5. Filter packets with http, TCP, DNS and other protocols. a. Find out what are those packets contain by following one of the conversations (also called network flows), select one of the packets and press the right mouse button..click on follow

Ans:

TCP:

Source	Destination	Protocol	Length	Info
192.168.29.243	140.82.114.25	TCP	55	2561 → 443 [ACK] Seq=1 Ack=1 Win=253 Len=1 [TCP segm
140.82.114.25	192.168.29.243	TCP	66	443 → 2561 [ACK] Seq=1 Ack=2 Win=70 Len=0 SLE=1 SRE=
192.168.29.243	85.14.245.45	TCP	54	2924 → 443 [FIN, ACK] Seq=1 Ack=1 Win=255 Len=0
192.168.29.243	85.14.245.45	TCP	54	2923 → 443 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
85.14.245.45	192.168.29.243	TCP	54	443 → 2924 [FIN, ACK] Seq=25 Ack=2 Win=501 Len=0
192.168.29.243	85.14.245.45	TCP	54	2924 → 443 [RST, ACK] Seq=2 Ack=25 Win=0 Len=0
192.168.29.243	136.232.79.144	TCP	66	2929 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=25
192.168.29.243	136.232.79.144	TCP	66	2930 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=25
136.232.79.144	192.168.29.243	TCP	66	80 → 2929 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS
136.232.79.144	192.168.29.243	TCP	66	80 → 2930 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS
192.168.29.243	136.232.79.144	TCP	54	2929 → 80 [ACK] Seq=1 Ack=1 Win=65536 Len=0
192.168.29.243	136.232.79.144	TCP	54	2930 → 80 [ACK] Seq=1 Ack=1 Win=65536 Len=0
136.232.79.144	192.168.29.243	TCP	54	80 → 2929 [ACK] Seq=1 Ack=584 Win=30464 Len=0
136.232.79.144	192.168.29.243	TCP	1514	80 → 2929 [ACK] Seq=1 Ack=584 Win=30464 Len=1460 [TC
136.232.79.144	192.168.29.243	TCP	1514	80 → 2929 [ACK] Seq=1461 Ack=584 Win=30464 Len=1460

DNS:

Source	Destination	Protocol	Length	Info
2405:201:8012:…	2405:201:8012:…	DNS	91	Standard query 0xbac4 A cssdeck.com
2405:201:8012:…	2405:201:8012:…	DNS	91	Standard query 0x6383 AAAA cssdeck.com
2405:201:8012:…	2405:201:8012:…	DNS	91	Standard query 0x188a HTTPS cssdeck.com
2405:201:8012:…	2405:201:8012:…	DNS	123	Standard query response 0xbac4 A cssdeck.com A 172.67.162.
2405:201:8012:…	2405:201:8012:…	DNS	119	Standard query response 0x6383 AAAA cssdeck.com AAAA 2606:
2405:201:8012:…	2405:201:8012:…	DNS	142	Standard query response 0x188a HTTPS cssdeck.com HTTPS
2405:201:8012:…	2405:201:8012:…	DNS	98	Standard query 0x4ae8 A www.jaduniv.edu.in
2405:201:8012:…	2405:201:8012:…	DNS	98	Standard query 0xdb05 AAAA www.jaduniv.edu.in
2405:201:8012:…	2405:201:8012:…	DNS	98	Standard query 0x0e9c HTTPS www.jaduniv.edu.in
2405:201:8012:…	2405:201:8012:…	DNS	114	Standard query response 0x4ae8 A www.jaduniv.edu.in A 136.
2405:201:8012:…	2405:201:8012:…	DNS	169	Standard query response 0xdb05 AAAA www.jaduniv.edu.in SOA
2405:201:8012:…	2405:201:8012:…	DNS	169	Standard query response 0x0e9c HTTPS www.jaduniv.edu.in SO
2405:201:8012:…	2405:201:8012:…	DNS	109	Standard query 0xaf2e A d27xxe7juh1us6.cloudfront.net
2405:201:8012:…	2405:201:8012:…	DNS	109	Standard query 0xe864 AAAA d27xxe7juh1us6.cloudfront.net
2405:201:8012:…	2405:201:8012:…	DNS	109	Standard query 0x76f8 HTTPS d27xxe7juh1us6.cloudfront.net
2405:201:8012:…	2405:201:8012:…	DNS	189	Standard query response 0xe864 AAAA d27xxe7juh1us6.cloudfr
2405:201:8012:…	2405:201:8012:…	DNS	189	Standard query response 0x76f8 HTTPS d27xxe7juh1us6.cloudf
2405:201:8012:…	2405:201:8012:…	DNS	173	Standard query response 0xaf2e A d27xxe7juh1us6.cloudfront

6. Search through your capture, and find an HTTP packet coming back from the server (TCP Source Port == 80). Expand the Ethernet layer in the Packet Details Panel.

On expanding packet in the Packet Details Panel, the following results are obtained.

```
Frame 67: 521 bytes on wire (4168 bits), 521 bytes captured (4168 bits) on interface
  Section number: 1
> Interface id: 0 (\Device\NPF_{A1D23659-979B-4254-9862-F6ECA398591B})
  Encapsulation type: Ethernet (1)
  Arrival Time: Nov 19, 2022 19:37:12.088244000 India Standard Time
  [Time shift for this packet: 0.000000000 seconds]
  Epoch Time: 1668866832.088244000 seconds
  [Time delta from previous captured frame: 0.001353000 seconds]
  [Time delta from previous displayed frame: 0.034173000 seconds]
  [Time since reference or first frame: 4.026843000 seconds]
  Frame Number: 67
  Frame Length: 521 bytes (4168 bits)
  Capture Length: 521 bytes (4168 bits)
  [Frame is marked: False]
  [Frame is ignored: False]
  [Protocols in frame: eth:ethertype:ip:tcp:http:data-text-lines]
  [Coloring Rule Name: HTTP]
  [Coloring Rule String: http || tcp.port == 80 || http2]
```

7.) What are the manufacturers of your PC's Network Interface Card (NIC), and the servers NIC?

Ans:

```
> Destination: Serverco_62:88:2f (b4:a7:c6:62:88:2f)
> Source: AzureWav_68:10:67 (d8:c0:a6:68:10:67)
```

Manufacturer's NIC: AzureWav_68:10:67 (d8:c0:a6:68:10:67)

Server's NIC: Serverco_62:88:2f (b4:a7:c6:62:88:2f)

8.) What are the Hex values (shown in the raw bytes panel) of the two NICS Manufacturers OUIs?

Ans:

For Laptop's Manufacturer :- d8:c0:a6:68:10:67

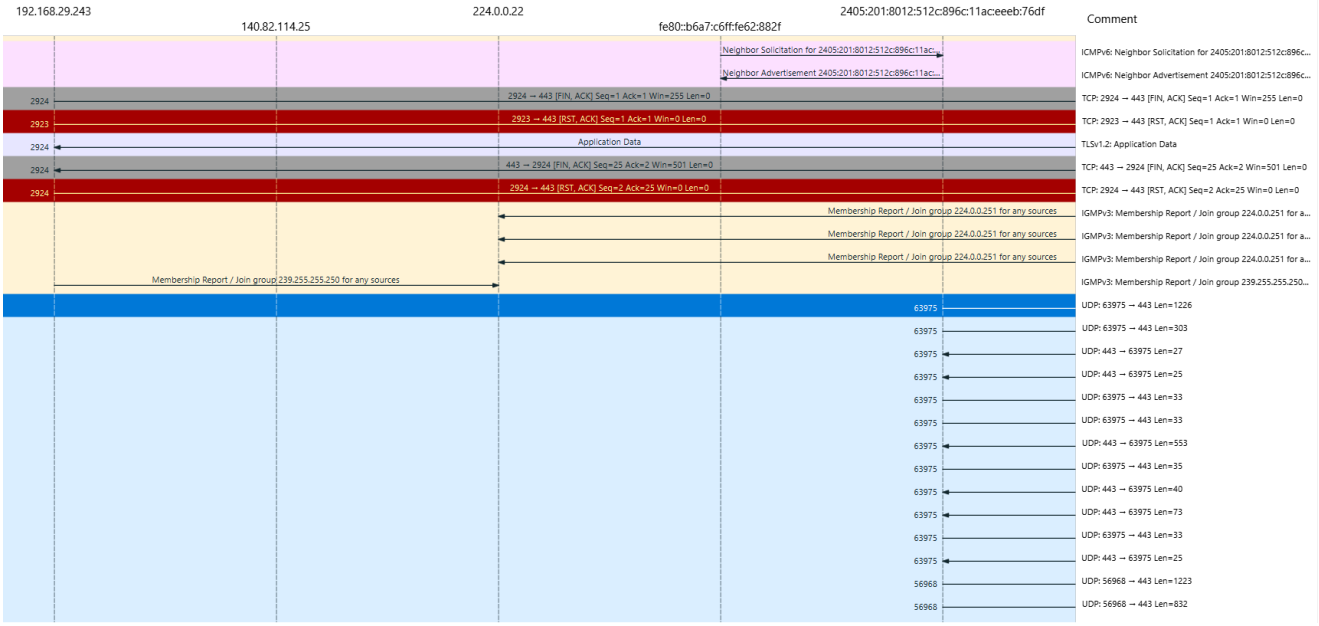
For server's Manufacturer :- b4:a7:c6:62:88:2f

10.) Find the traffic flow Select the Statistics->Flow Graph menu option. Choose General Flow and Network Source options, and click the OK button.

Ans:

Graph Obtained from General Flow and network source option of flow graphs:

P - 25


192.168.29.243
140.82.114.25
224.0.0.22
fe80::b6a7:c6ff:fe62:882f
2405:201:8012:512c:896c:11ac:eeeb:76df
Comment
Neighbor Solicitation for 2405:201:8012:512c:896c:11ac...
Neighbor Advertisement 2405:201:8012:512c:896c:11ac...
2924 → 443 [FIN, ACK] Seq=1 Ack=1 Win=255 Len=0
2923 → 443 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
Application Data
443 → 2924 [FIN, ACK] Seq=25 Ack=2 Win=501 Len=0
2924 → 443 [RST, ACK] Seq=2 Ack=25 Win=0 Len=0
Membership Report / Join group 224.0.0.251 for any sources
Membership Report / Join group 224.0.0.251 for any sources
Membership Report / Join group 224.0.0.251 for any sources
Membership Report / Join group 239.255.255.250 for any sources
63975
63975
63975
63975
63975
63975
63975
63975
63975
63975
63975
63975
56968
56968
ICMPv6: Neighbor Solicitation for 2405:201:8012:512c:896c...
ICMPv6: Neighbor Advertisement 2405:201:8012:512c:896c...
TCP: 2924 → 443 [FIN, ACK] Seq=1 Ack=1 Win=255 Len=0
TCP: 2923 → 443 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
TLSv1.2: Application Data
TCP: 443 → 2924 [FIN, ACK] Seq=25 Ack=2 Win=501 Len=0
TCP: 2924 → 443 [RST, ACK] Seq=2 Ack=25 Win=0 Len=0
IGMPv3: Membership Report / Join group 224.0.0.251 for a...
IGMPv3: Membership Report / Join group 224.0.0.251 for a...
IGMPv3: Membership Report / Join group 224.0.0.251 for a...
IGMPv3: Membership Report / Join group 239.255.255.250...
UDP: 63975 → 443 Len=1226
UDP: 63975 → 443 Len=303
UDP: 443 → 63975 Len=27
UDP: 443 → 63975 Len=25
UDP: 63975 → 443 Len=33
UDP: 63975 → 443 Len=33
UDP: 443 → 63975 Len=553
UDP: 63975 → 443 Len=35
UDP: 443 → 63975 Len=40
UDP: 443 → 63975 Len=73
UDP: 63975 → 443 Len=33
UDP: 443 → 63975 Len=25
UDP: 56968 → 443 Len=1223
UDP: 56968 → 443 Len=832

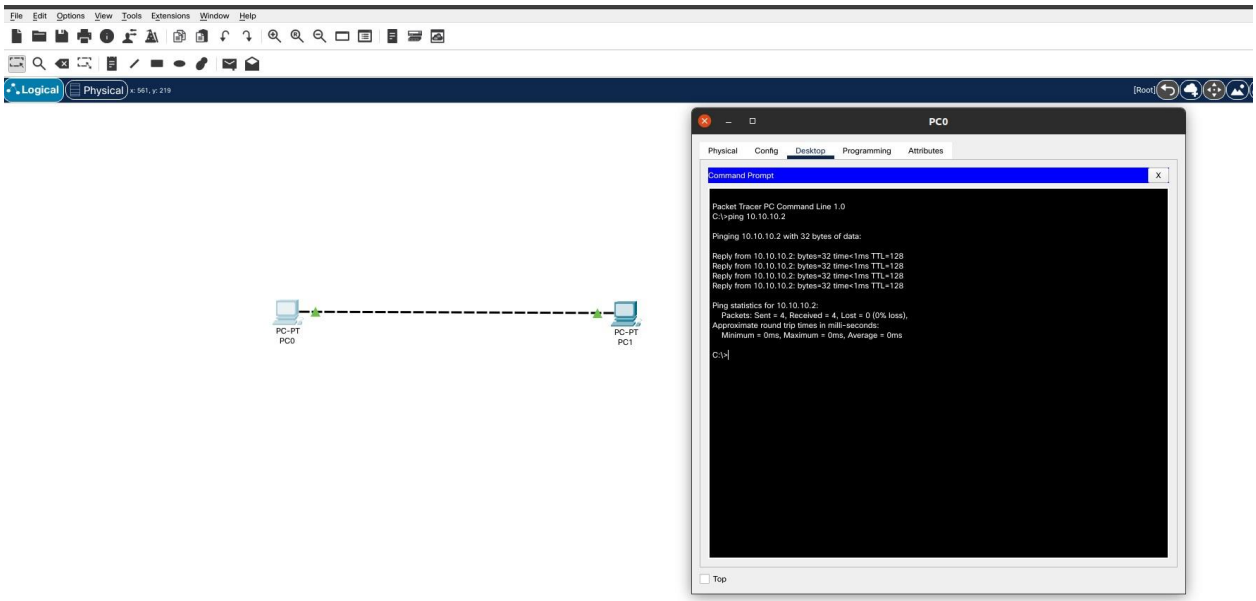
ASSIGNMENT 6

Name: Imon Raj
Class: BCSE III
Roll: 002010501098
Section: A3
Subject: Computer Networks Lab Report

PROBLEM STATEMENT: Use Cisco Packet Tracer software to do the following experiments.

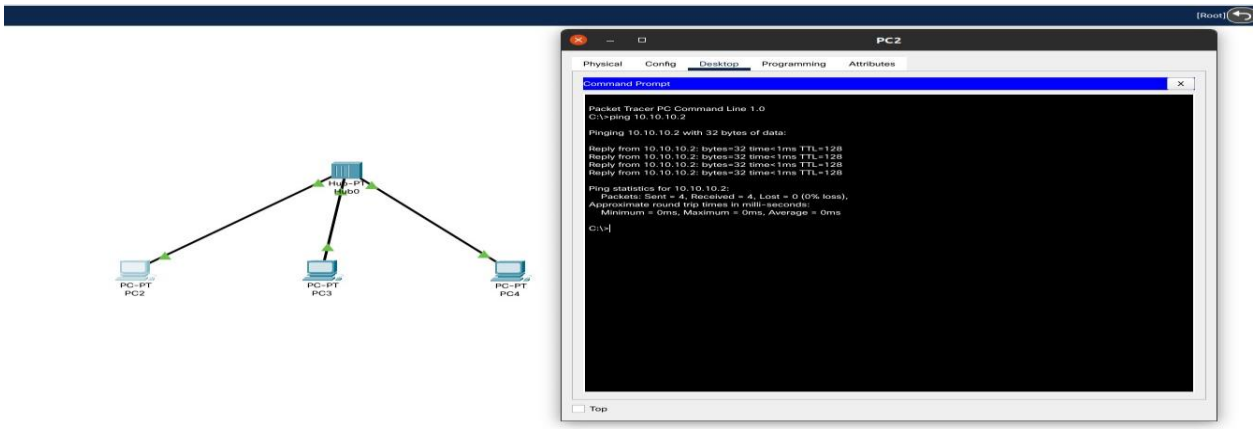
Overview : This entire assignment has been done using the CISCO Packet tracer tool.
System : Linux (Ubuntu 20.04)

Connect two hosts back-to-back with a crossover cable. Assign IP addresses, andsee whether they are able to ping each other.

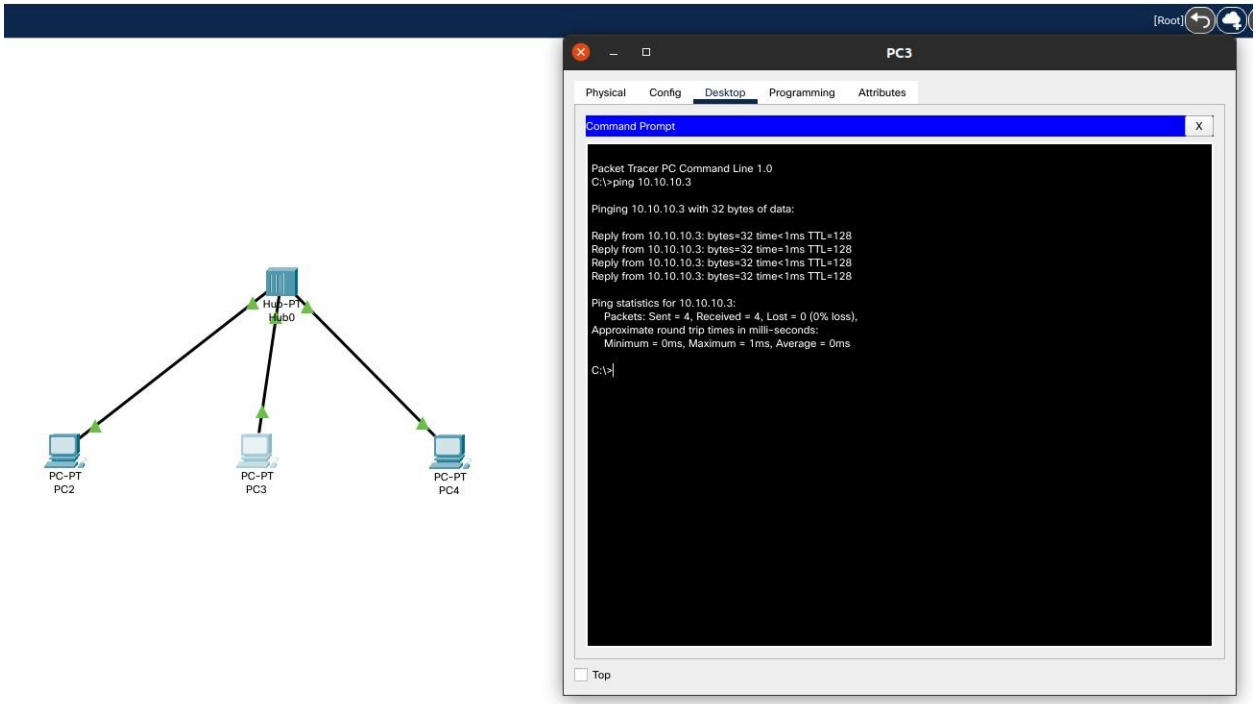


1. Create a LAN (named LAN-A) with 3 hosts using a hub. Ping each pair of nodes.

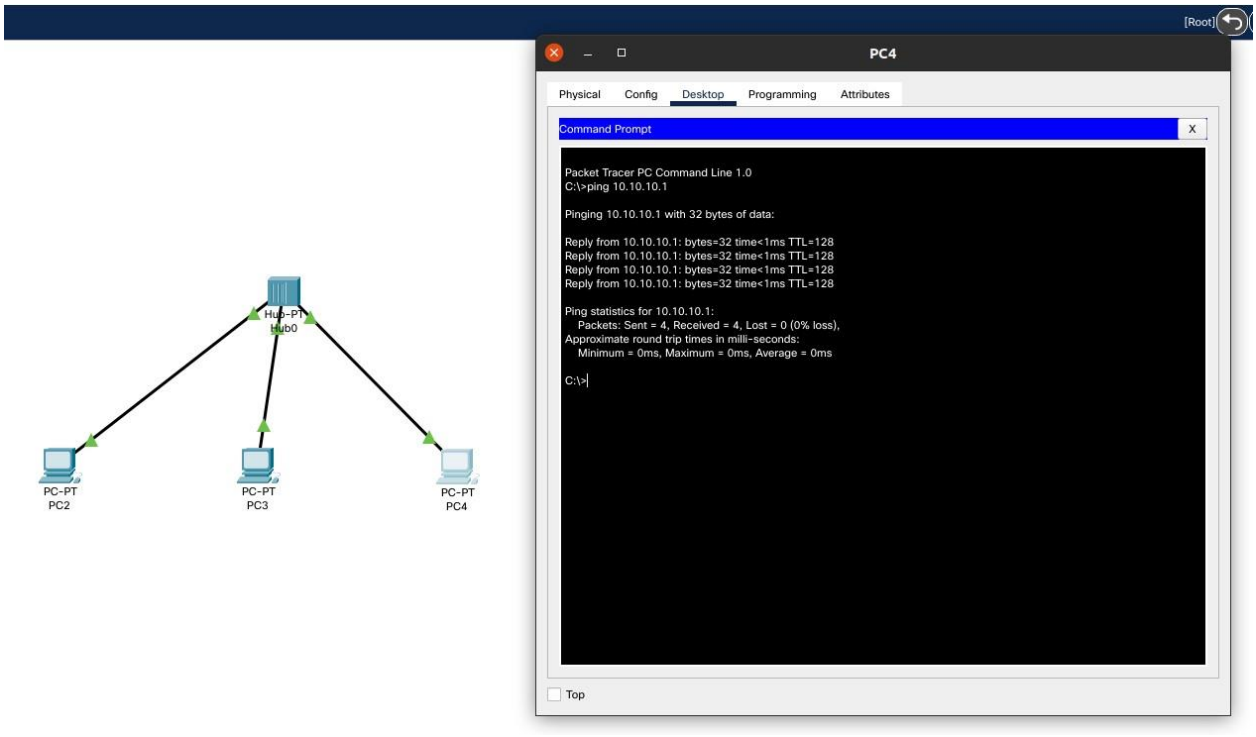
host with ip 10.10.10.1 pinging to 10.10.10.2



host with ip 10.10.10.2 pinging to 10.10.10.3

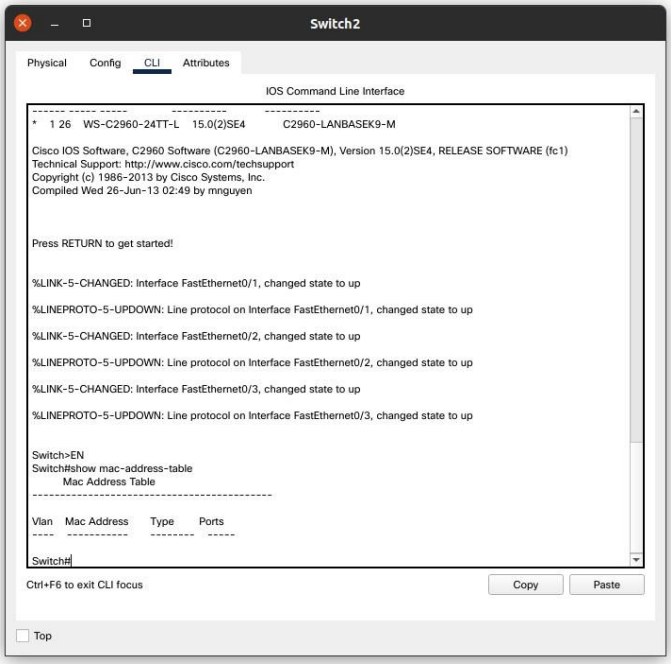


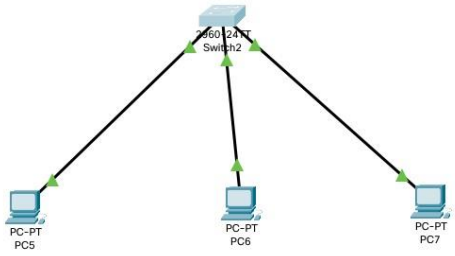
host with ip 10.10.10.3 pinging to 10.10.10.1



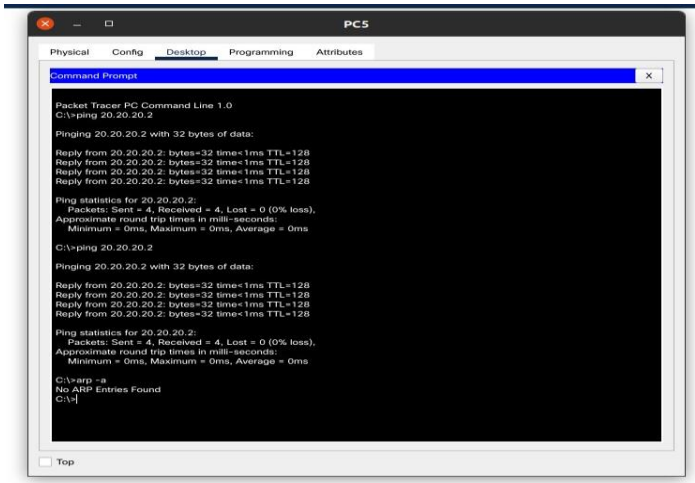
2. Create a LAN (named LAN-B) with 3 hosts using a switch. Record contents of theARP Table of end hosts and the MAC Forwarding Table of the switch. Ping each pair of nodes. Now record the contents of the ARP Table of end hosts and the MAC Forwarding Table of the switch again.

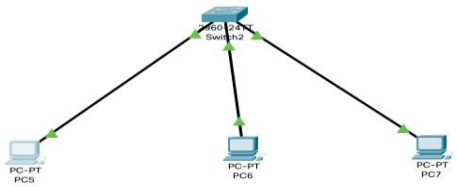
Initial contents of the MAC Address table for the switch



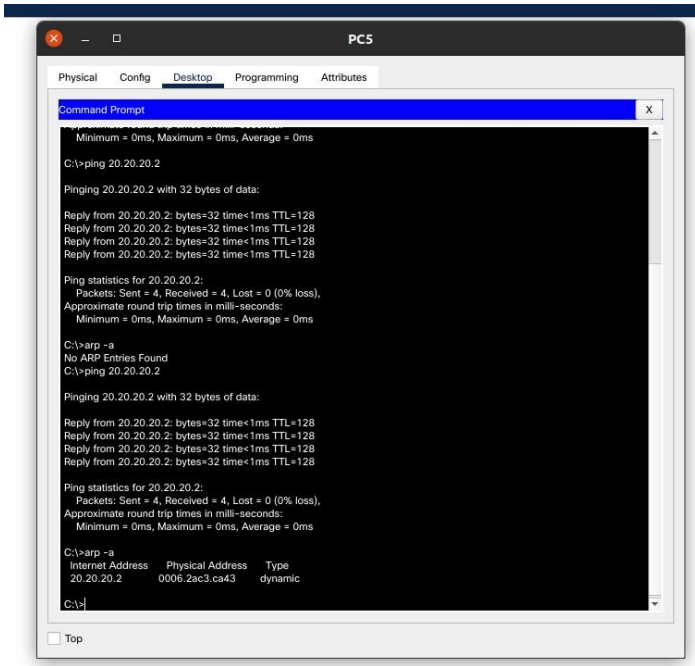


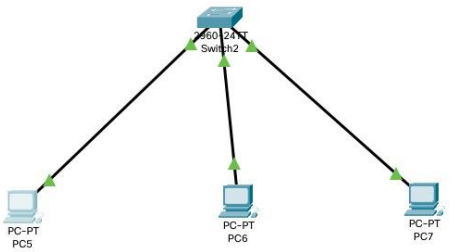
ARP Record of host 1 before sending



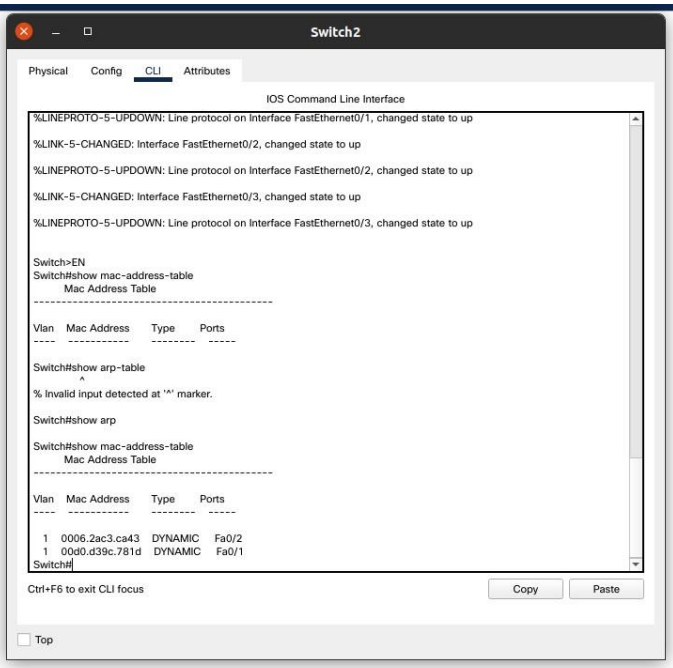


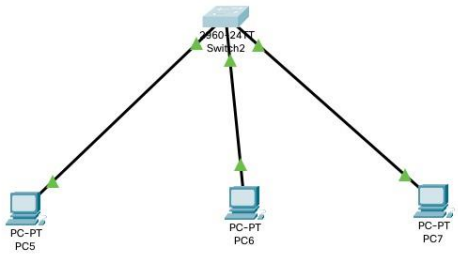
ARP Record of host 1 after pinging host 2





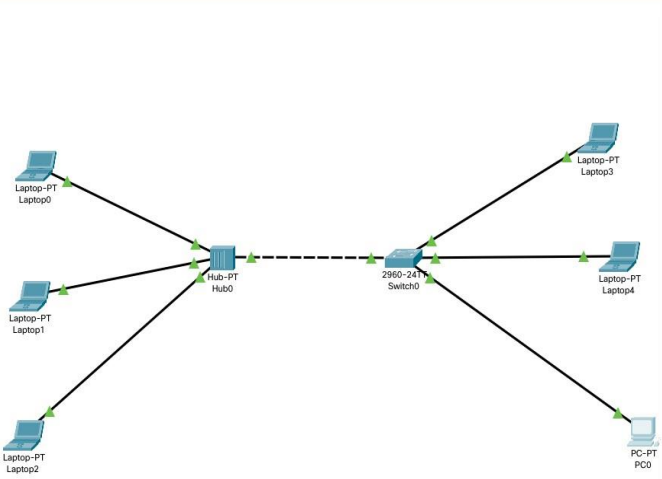
MAC Address Table of the switch after a ping operation from host1 -> host2

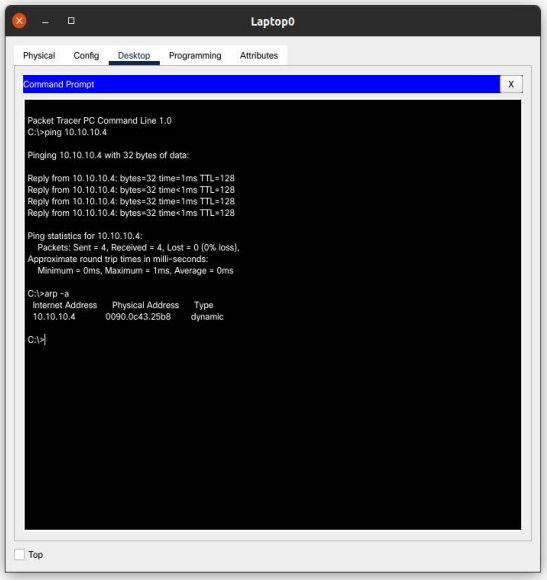


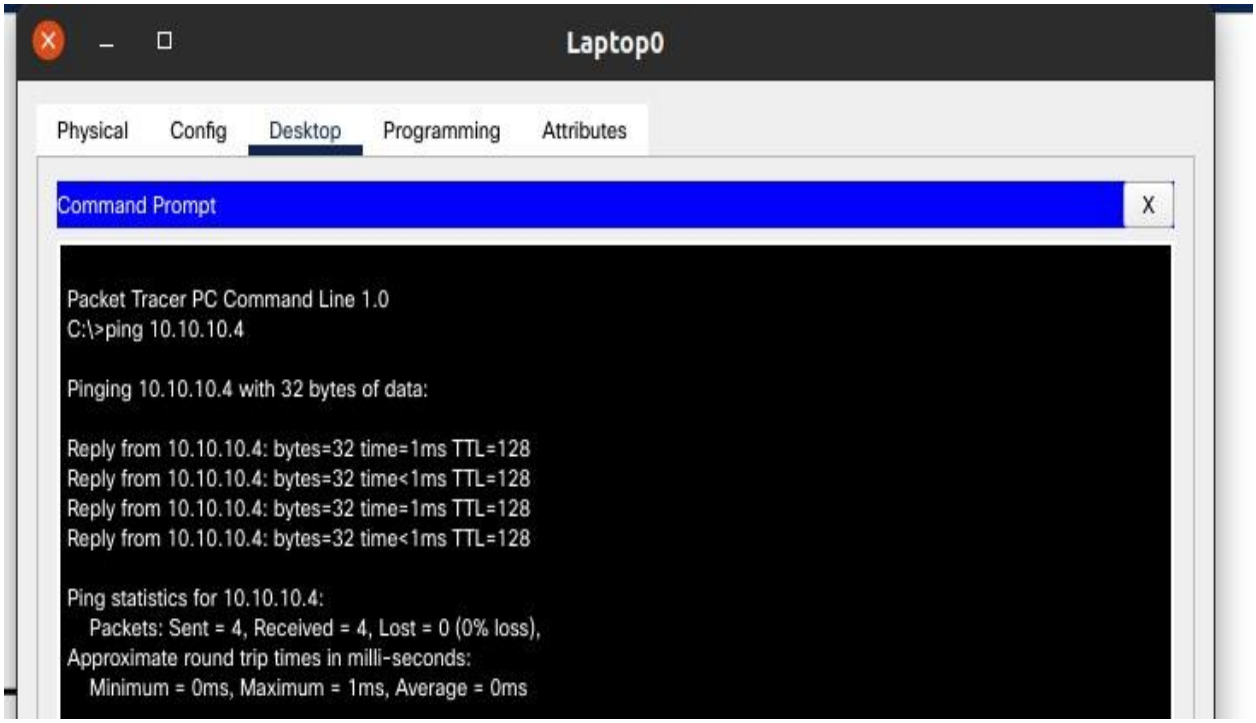


3. Connect LAN-A and LAN-B by connecting the hub and switch using a crossovercable. Ping between each pair of hosts of LAN-A and LAN-B. Now record the contents of the ARP Table of end hosts and the MAC Forwarding Table of the switch again.

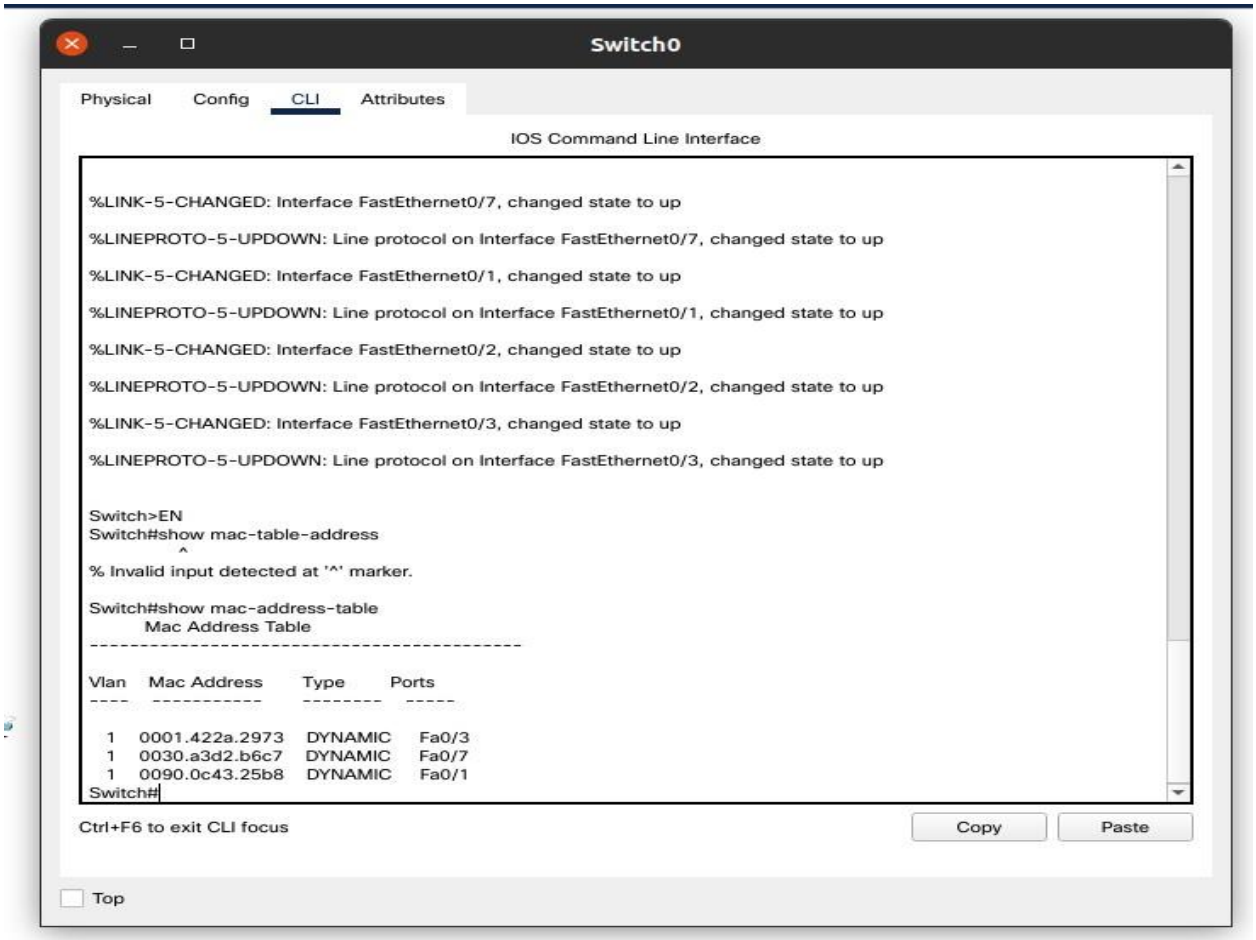
ARP Table after connecting LAN A and LAN B





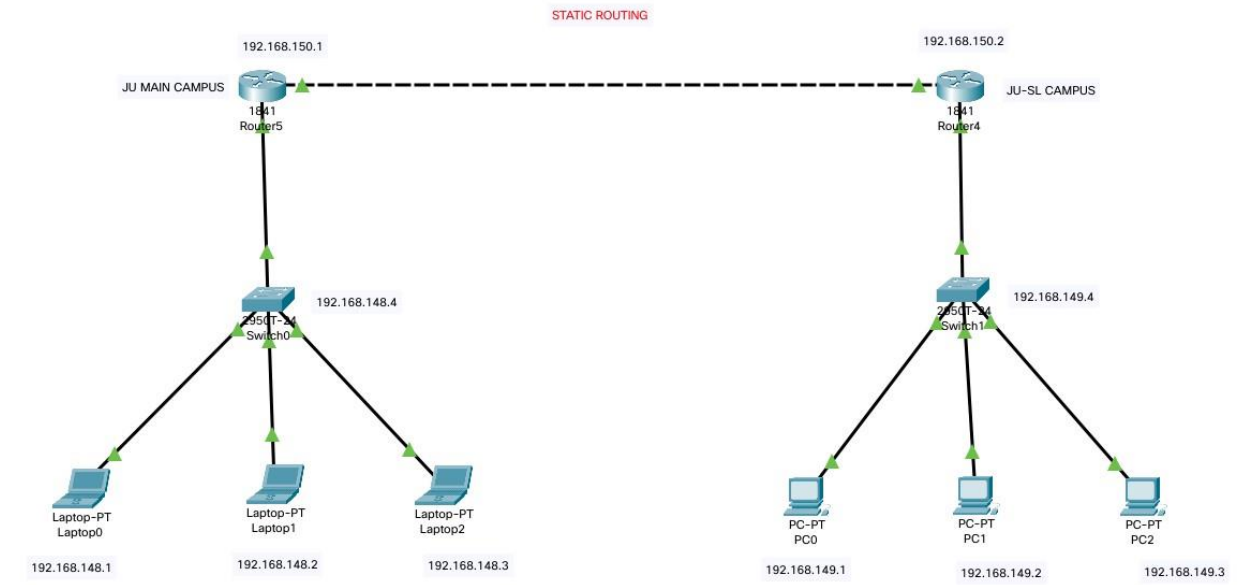


MAC Forwarding Table of the switch in LAN B



4. Create a LAN (named JU-Main) with three hosts connected via a layer-2 switch(Cisco 2950 switch PC-LAB1-Switch). Connect the switch to a router (Cisco 1818). Assign IP addresses to all the hosts and the router interface connected to this LAN from network 192.168.148.0/24. Configure the default gateway of each host as the IP address of the interface of the router which is connected to the LAN. Create another LAN (named JU-SL) with three hosts connected via a layer-2 switch (Cisco 2950 switch PC-LAB2-Switch). Connect this switch to another router (Cisco 1818). Assign IP addresses to all the hosts and the router interface connected to this LAN from network 192.168.149.0/24. Configure the default gateway of each host as the IP address of the interface of the router which is connected to the LAN. Connect the two routers through appropriate WAN interfaces. Assign IP addresses to the WAN interfaces from network 192.168.150.0/24. Add static route in both of the routers to route packets between two LANs.

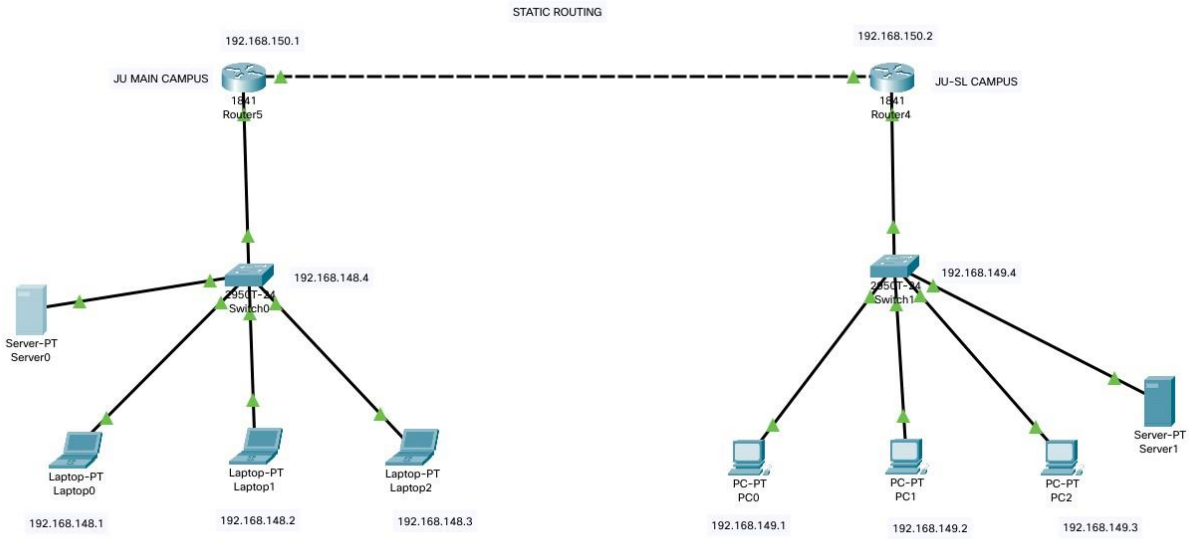
Network Layout



Pinging a host (192.168.149.2, JU-SL Campus) from a host (192.168.148.2, JU-MAIN Campus) using static routing.

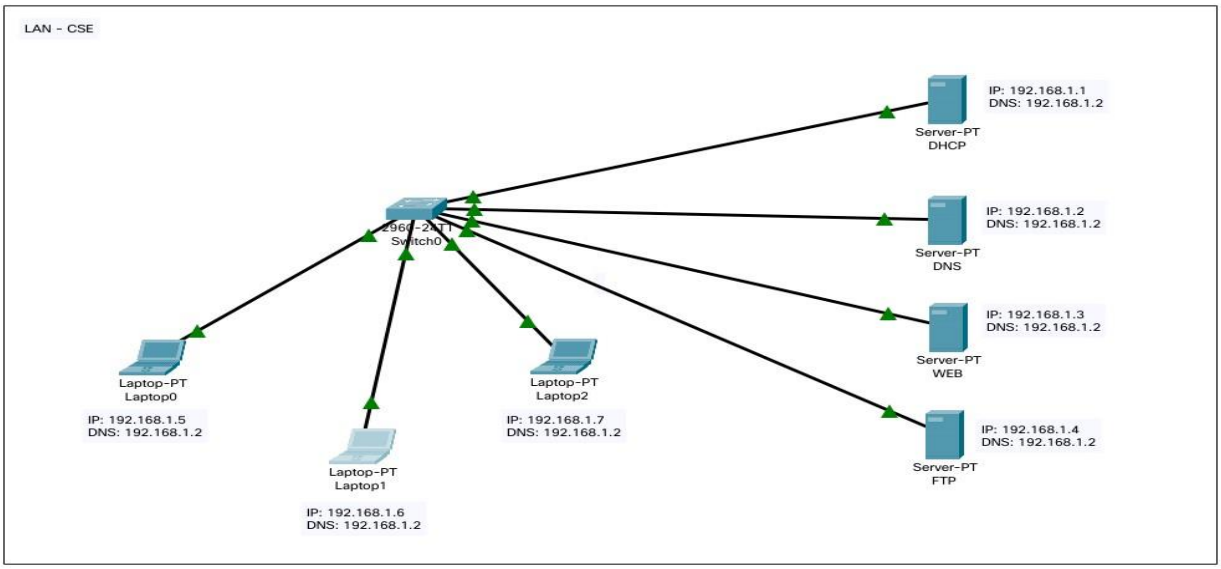
```
Packet Tracer PC Command Line 1.0
C:\>ping 192.168.149.2

Pinging 192.168.149.2 with 32 bytes of data:

Request timed out.
Reply from 192.168.149.2: bytes=32 time<1ms TTL=126
Reply from 192.168.149.2: bytes=32 time=1ms TTL=126
Reply from 192.168.149.2: bytes=32 time<1ms TTL=126

Ping statistics for 192.168.149.2:
    Packets: Sent = 4, Received = 3, Lost = 1 (25% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 1ms, Average = 0ms

C:\>
```

5. Add servers to the individual LANs (in problem 5) and configure them as a DHCPserver. Configure the hosts in the individual LAN to obtain IP addresses and address of the default gateway via this DHCP server.

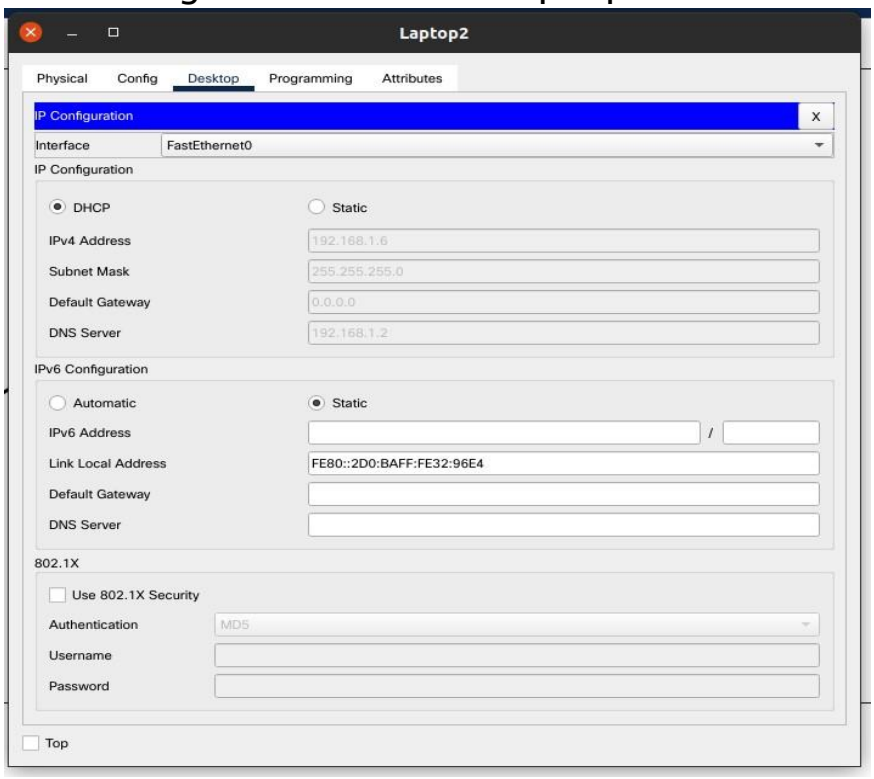


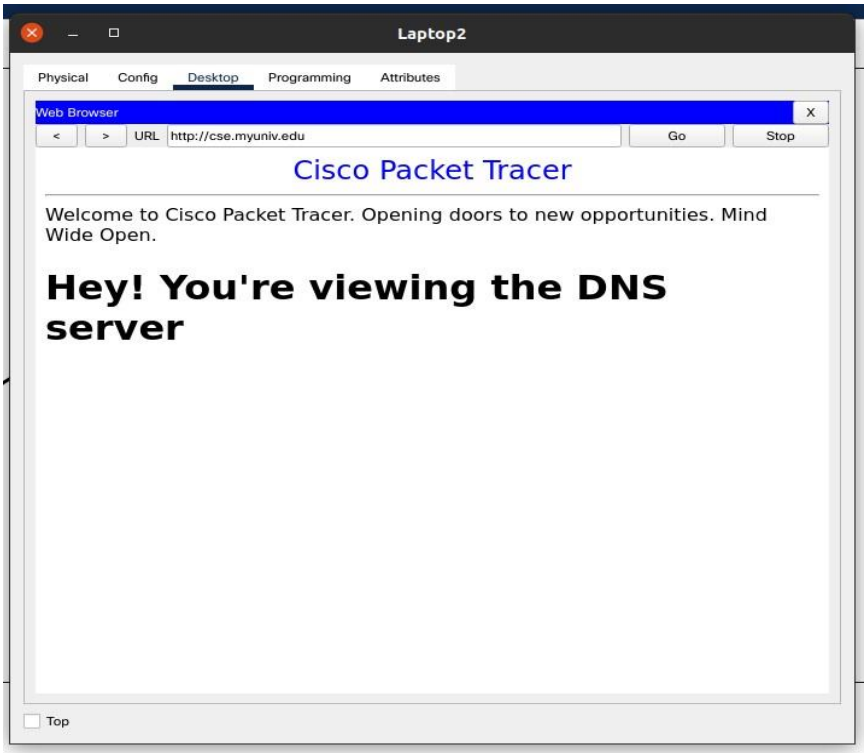
6. Create a LAN (CSE) with three hosts connected via a layer-2 switch (Cisco 2950 switch CSE-Switch). Also add a web server and a ftp server to this LAN. The hosts dynamically get their IP addresses from a local DHCP server. Servers are assigned fixed IP addresses. Configure the individual hosts to use the local DNS server for name resolution. Add a Domain Name Server (DNS) to this LAN. Create appropriate records in the DNS server for the individual servers in the LAN. The domain name of the LAN is cse.myuniv.edu. Configure the individual hosts to use the local DNS server for name resolution.

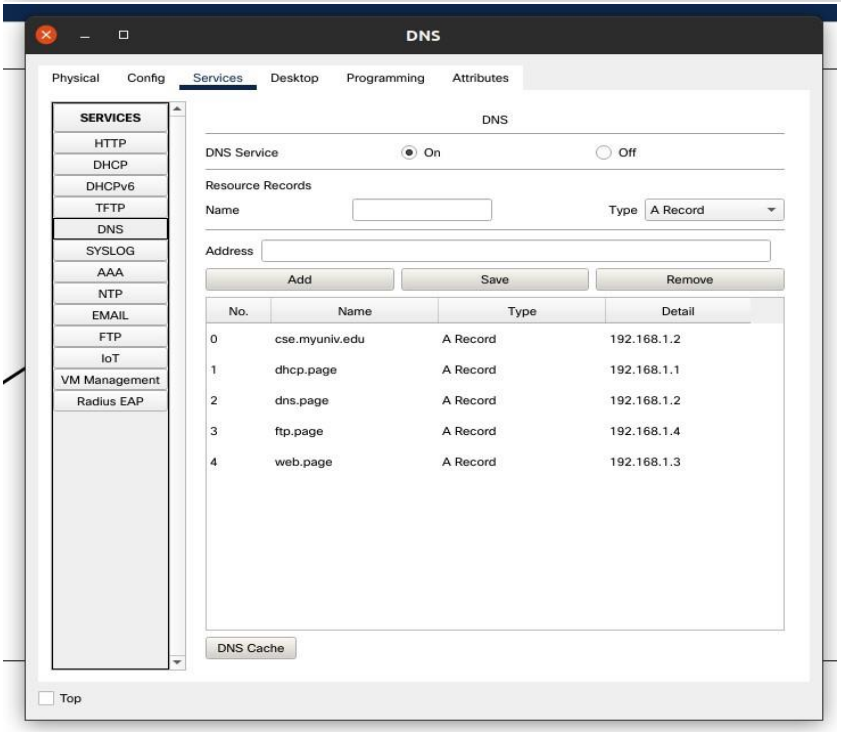
Network Layout

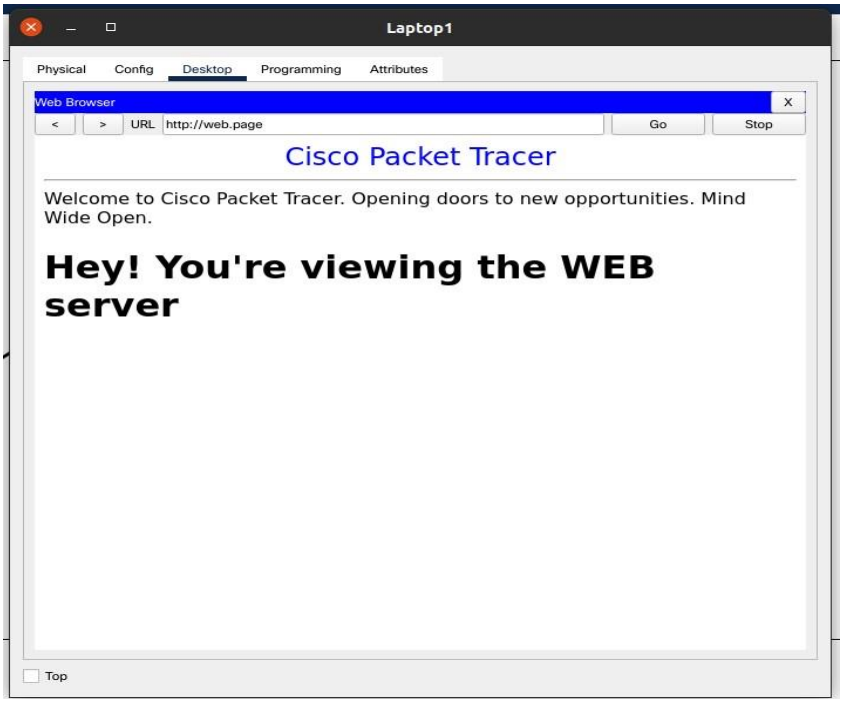


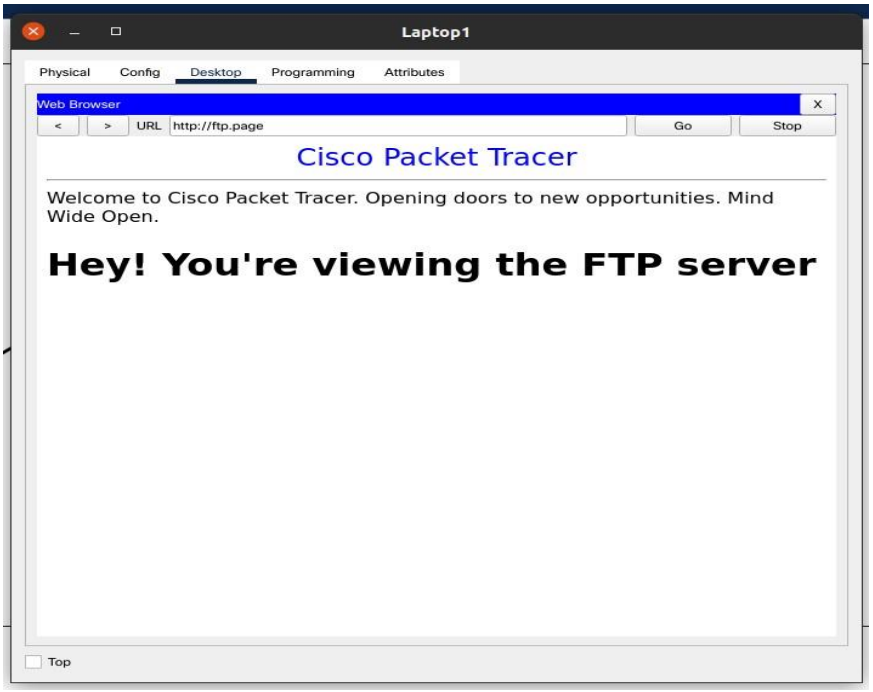
IP Configuration of a Laptop

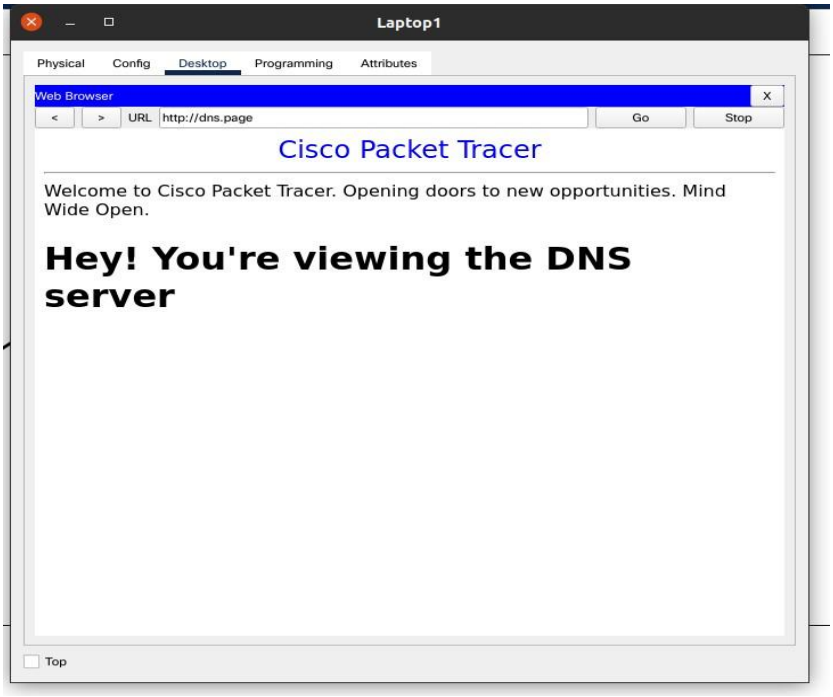


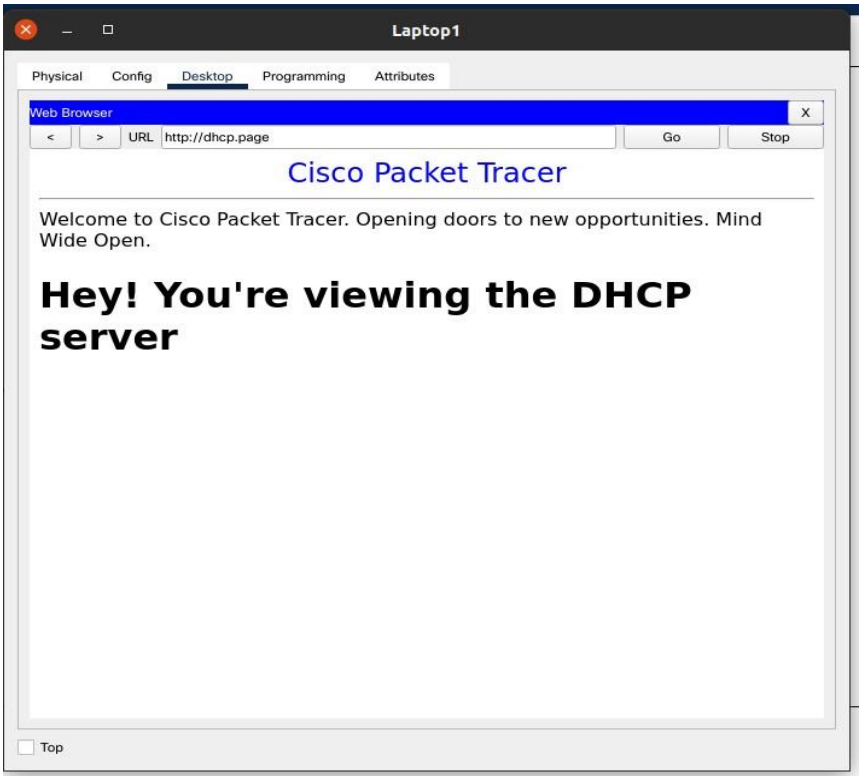

Laptop2
Physical
Config
Desktop
Programming
Attributes
Web Browser
X
<
>
URL
http://cse.myuniv.edu
Go
Stop
Cisco Packet Tracer
Welcome to Cisco Packet Tracer. Opening doors to new opportunities. Mind Wide Open.
Hey! You're viewing the DNS server
Top


DNS
Physical
Config
Services
Desktop
Programming
Attributes
SERVICES
HTTP
DHCP
DHCPv6
TFTP
DNS
SYSLOG
AAA
NTP
EMAIL
FTP
IoT
VM Management
Radius EAP
DNS
DNS Service
On
Off
Resource Records
Name
Type
A Record
Address
Add
Save
Remove
No.
Name
Type
Detail
0
cse.myuniv.edu
A Record
192.168.1.2
1
dhcp.page
A Record
192.168.1.1
2
dns.page
A Record
192.168.1.2
3
ftp.page
A Record
192.168.1.4
4
web.page
A Record
192.168.1.3
DNS Cache
Top


Laptop1
Physical
Config
Desktop
Programming
Attributes
Web Browser
X
<
>
URL
http://web.page
Go
Stop
Cisco Packet Tracer
Welcome to Cisco Packet Tracer. Opening doors to new opportunities. Mind Wide Open.
Hey! You're viewing the WEB server
Top


Laptop1
Physical
Config
Desktop
Programming
Attributes
Web Browser
X
<
>
URL
http://ftp.page
Go
Stop
Cisco Packet Tracer
Welcome to Cisco Packet Tracer. Opening doors to new opportunities. Mind Wide Open.
Hey! You're viewing the FTP server
Top


Laptop1
Physical
Config
Desktop
Programming
Attributes
Web Browser
X
<
>
URL
http://dns.page
Go
Stop
Cisco Packet Tracer
Welcome to Cisco Packet Tracer. Opening doors to new opportunities. Mind Wide Open.
Hey! You're viewing the DNS server
Top



Comments :

The CISCO packet tracer is actually a pretty useful tool in simulating the Network Layout. It can help to plan the architecture, simulate events and then deploy it in real life so that we can be aware about the performance of the network in production. Further it helps experience a practical knowledge of how elements of an entire work are when deployed in an integrated fashion. Hoping to get more of such assignments to that learning becomes fun and not stressful.

ASSIGNMENT 7

Name: Imon Raj
Class: BCSE III
Roll: 002010501098
Section: A3
Subject: Computer Networks Lab Report

PROBLEM STATEMENT: Implement any two protocols using TCP/UDP Socket as suitable. 1. ARP 2. BOOTP 3. DHCP

DESIGN: This assignment is based on upper layers' protocols. Among the three mentioned, I have implemented ARP(Address resolution protocol) and DHCP(Dynamic host configuration protocol). I have used Java as my programming language and an Object-oriented approach.

- ARP has the work to translate the logical address(IP address) in a network to the corresponding device's MAC address. Here, if one host wants to know the MAC address of another host whose IP is known, it will send a broadcast to all the hosts requesting the MAC. The target host will respond to the requesting machine with its MAC address. We will have a static table that contains IP addresses and their corresponding MAC addresses.

- DHCP is used to assign an IP address to a host in the network, dynamically. In the case of DHCP, no host has a fixed(static) IP address in a network. Also if a host is inactive for a while its IP is revoked, such that other new hosts can accommodate in the network. For obvious reasons, a host must renew its already assigned IP address regularly according to the given rules.

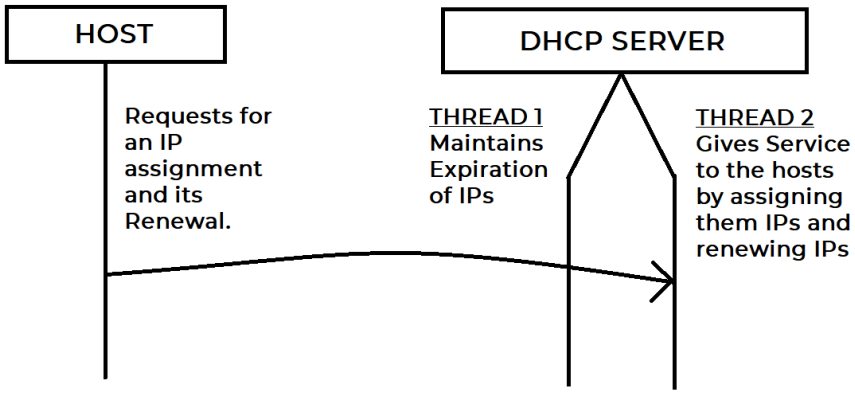
  For our implementation purpose, we will have a DHCP server. It will run two concurrent threads internally. One thread will just check the expiration of IP addresses, which are not renewed after a certain duration, and must be freed. And another thread will communicate directly with hosts(Clients). Any Client request for DHCPDISCOVER or DHCPRENEW will be served by this thread. For each IP, we will have a Boolean indicating its availability, one timestamp to indicate its last renewal and an Integer to indicate the host which has been assigned this address.

IMPLEMENTATION:

All Java codes are uploaded in the shared Drive folder.

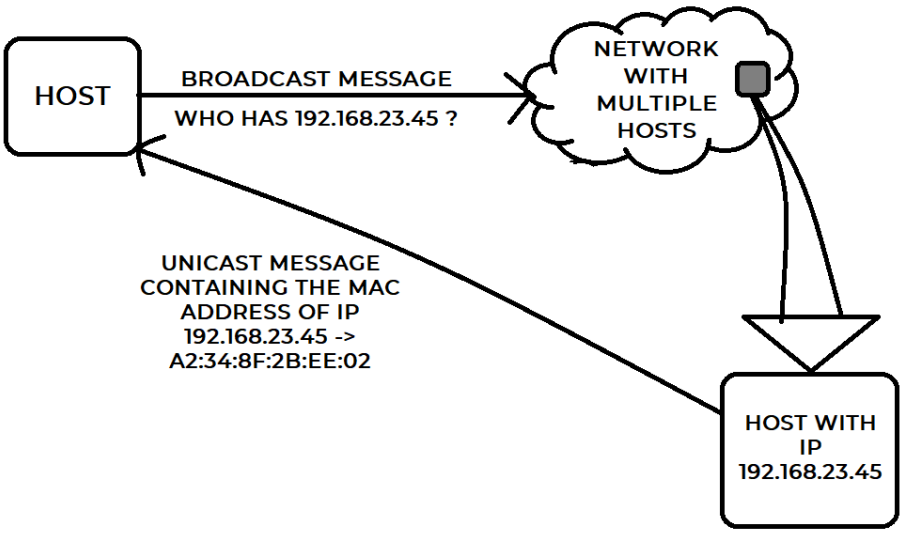
I have used TCP socket of Java for the implementations. I have used library classes – **ServerSocket** and **Socket** for creating Client-Server system. For reading data from socket and writing data into socket, **DataInputStream**, **DataOutputStream** classes are used. To keep track of time in case of DHCP, **TimeStamp** class is used.

To show the implementation model, I have attached a figure below for DHCP.



I have attached a figure for ARP implementation as well.

ARP IMPLEMENTATION



TEST CASES:

For ARP:

The program should be able to assure –

- Each IP in the network should have a corresponding MAC address.
- In case of a wrong(not present) IP, the program should show an error message.

For DHCP:

The program should be able to assure –

- If at least one IP is free, then a requesting host should be assigned.
- The DHCP server should continuously check the expirations of IPs.
- No host should be allowed to renew an IP which is already expired, or assigned to another host.

RESULTS & ANALYSIS:

DHCP OUTPUT:

```
---------------
1. DISCOVER NEW IP
2. RENEW IP
0. EXIT
ENTER CHOICE: 1
YOUR CLIENT_NO(AS IF PORT): 10
CLIENT 10 IS ASSIGNED 120.120.30.0

---------------
1. DISCOVER NEW IP
2. RENEW IP
0. EXIT
ENTER CHOICE: 1
YOUR CLIENT_NO(AS IF PORT): 20
CLIENT 20 IS ASSIGNED 120.120.30.1

---------------
1. DISCOVER NEW IP
2. RENEW IP
0. EXIT
ENTER CHOICE: 2
YOUR CLIENT_NO(AS IF PORT): 10
SORRY, IP EXPIRED..
```

```
IP 120.120.30.0 EXPIRED
IP 120.120.30.1 EXPIRED
```

ARP OUTPUT:

```
Enter the Logical address(IP):
165.165.80.80
The Physical Address is: 6A:08:AA:C2:A3:23
```

This program is so far able to provide all the functionalities which are mentioned in the test-cases section.

COMMENTS:

This program was interesting to implement. There is chance of improvement in case of ARP implementation. We can have different approach rather than using a static table. More or less both the protocols' implementations are complete.

ASSIGNMENT 8

Name: Imon Raj
Class: BCSE III
Roll: 002010501098
Section: A3
Subject: Computer Networks Lab Report

PROBLEM STATEMENT: Implement any two protocols using TCP/UDP Socket as suitable.
1. FTP 2. DNS 3. Telnet

DESIGN: This assignment is based on application layer protocols. Among the three mentioned, I have implemented FTP(File Transfer Protocol) and DNS(Domain Name System). I have used Java as my programming language and an Object-oriented approach.

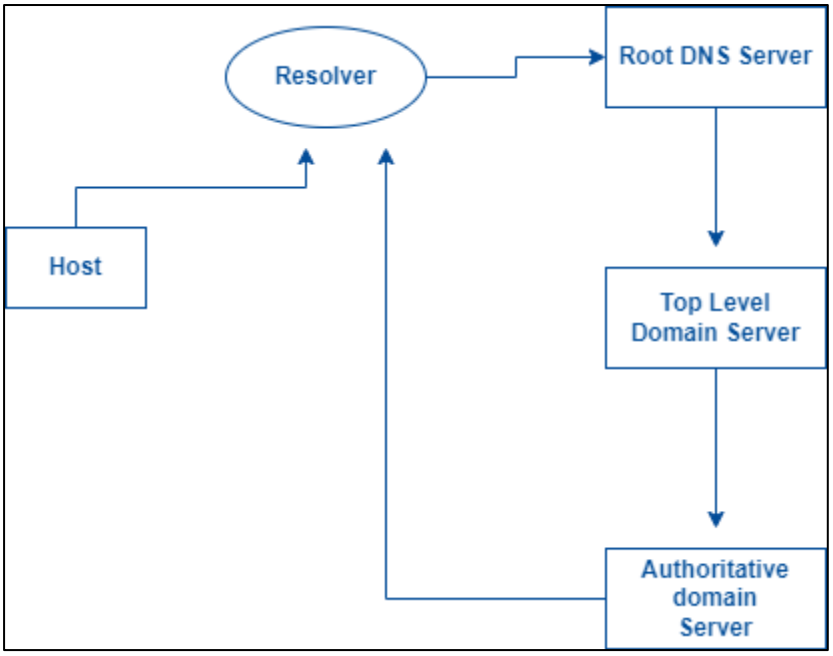
- FTP (File Transfer Protocol) is a network protocol for transmitting files between computers over Transmission Control Protocol/Internet Protocol (TCP/IP) connections. Within the TCP/IP suite, FTP is considered an application layer protocol.

  In an FTP transaction, the end user's computer is typically called the local host. The second computer involved in FTP is a remote host, which is usually a server. Both computers need to be connected via a network and configured properly to transfer files via FTP. Servers must be set up to run FTP services, and the client must have FTP software installed to access these services.

  Although many file transfers can be conducted using Hypertext Transfer Protocol (HTTP) -- another protocol in the TCP/IP suite -- FTP is still commonly used to transfer files behind the scenes for other applications, such as banking services. It is also sometimes used to download new applications via web browsers.

- DNS stands for Domain Name System. It is a naming system that is used to identify devices across the internet. It is an application layer protocol and is used to map the domain names to the IP addresses.

  The host requests for the IP address of a particular domain name to the DNS server and the IP address is returned to the host by the DNS server. The hierarchy of the resolution of the request is shown below.



IMPLEMENTATION:

All Java codes are uploaded in the shared Drive folder.

I have used TCP socket of Java for the implementations. I have used library classes – **ServerSocket** and **Socket** for creating Client-Server system. For reading data from socket and writing data into socket, **DataInputStream**, **DataOutputStream** classes are used.

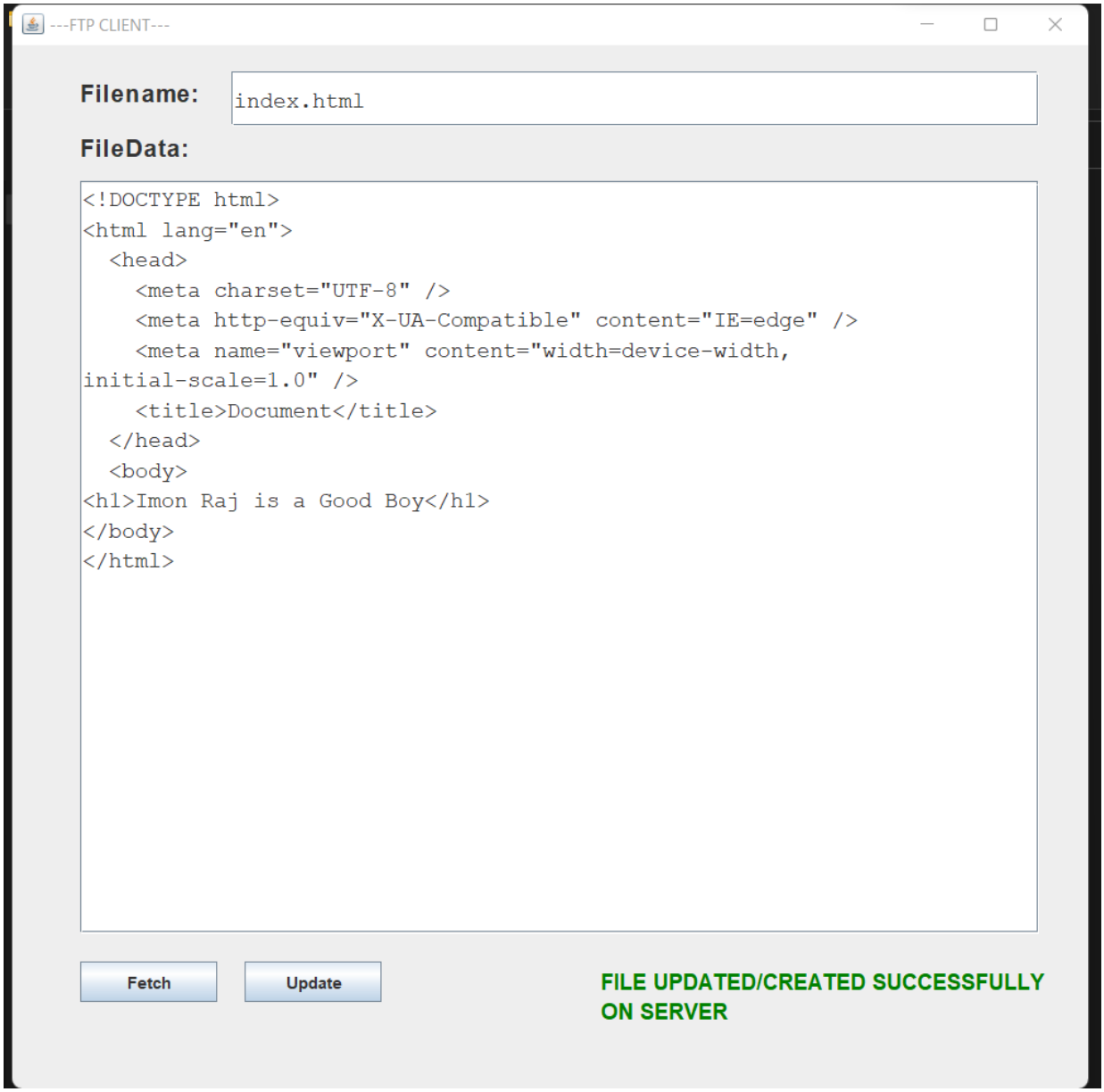
For FTP, I have developed GUI based application, similar to FTP clients like FileZilla. For GUI I have used **java.swing** library. Here Connection is created with remote FTP server and then file can be downloaded from server and uploaded(may be updated) to server.

For DNS, I have tried to use the iterative system. I have implemented the name resolution such that it can resolve domains with two levels.

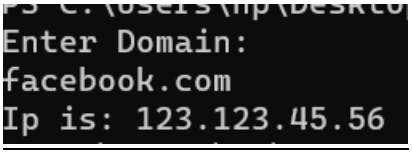
.

RESULTS & ANALYSIS:

FTP OUTPUT:



DNS OUTPUT:



COMMENTS:

This program was interesting to implement. There is chance of improvement in case of DNS implementation. We can have functionalities to resolve domains with multiple levels.