Thadomal Shahani Engineering College

Bandra (W.), Mumbai- 400 050.

CERTIFICATE

Certify that ~~Mr.~~/Miss Niyati S. Savant

of Computer Department, Semester V with

Roll No. 2103156 has completed a course of the necessary

experiments in the subject Computer Network under my

supervision in the **Thadomal Shahani Engineering College**

Laboratory in the year 2023 - 2024

Barodi 18/10/23

Teacher In-Charge

Head of the Department

Date 18/10/23

Principal

THADOMAL SHAHANI TSEC ENGINEERING SOLLEGE

CONTENTS

THADOMAL SHAHANI
TSEC
ENGINEERING COLLEGE

CONTENTS

Niyati Savant
C31 - TE
2103156

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE
241 7123

Experiment - 01

Aim: Study of RJ45 and CAT6 cabling and connection using crimping tool.

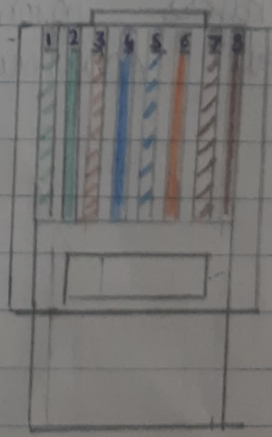
Theory:

RJ i.e Registered Jack is a standard format of telecommunication network interface used for connecting voice and data equipment. One of its main functionality is to connect different data equipment and telecommunication devices with service provided by telephone exchanges like long and short-exchange carrier.

The connector RJ45 is available in two standards ie. T568A and T568B which work as pin IN and pin Out for ethernet cable to perform data transfer.

T568A is commonly used pinout standard for ethernet cables. The highlight is ~~its~~ it is backward compatible with one-pair as well as two-pair USOC (Universal service ordering code). The color code table is -

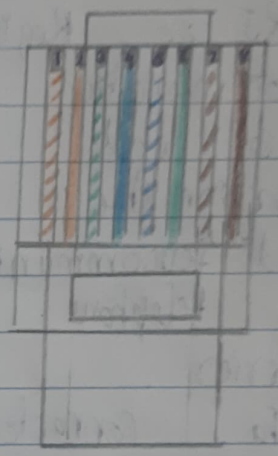
i) White Green
ii) Green
iii) White orange
iv) Blue
v) White blue
vi) Orange
vii) White Brown
viii) Brown



THADOMAL SHAHANI
TSEC
ENGINEERING COLLEGE

The RJ45 Pin Out T568B gives better protection from noise and isolates signal more effectively. However it is only backward compatible with one-pair USOC wiring scheme. The color code is -

i) White Orange
ii) Orange
iii) White Green
iv) Blue
v) White Blue
vi) Green
vii) White Brown
viii) Brown



Some characteristics of RJ45 connector:

- Excellent sealing and waterproof performance for usage in multiple environments
- Strong signal transmission due to complete shielding system
- Safety locking system to ensure connector is not detached while in use
- Transfers information at very high-speed to help achieve maximum

Niyati Savant
C31 - TE
2103156

Cat 6 cable is derived from Category 6, it is a twisted pair cable for ethernet that is backward compatible with Cat 3, Cat 5 and Cat 5e cable standards. It is designed for more improved performance as compared to Cat 5 and it is also better at handling crosstalk

Some features of Cat 6 cable -

1) It is costlier than Cat 5e as it has a thicker copper wire.
2) Frequency varies from 0-250 MHz to 500 MHz
3) For slower networks, cable length is 100 m and for gigabit ethernet it is 33 m & 55 m for low crosstalk.
4) It offers better speed but is comparatively less flexible.
5) Its construction is more robust as it has a splint inside its jacket

Working -

i) Prepare the cable : Strip about 1-1.5 inches of the outer insulation off the Cat 5 cable using a cable stripper without damaging the inner wires
ii) Untwist pairs : Untwist wire pairs within cable making sure they remain twisted as close to connector as possible
iii) Arrang wires : Align wires according to the T568A

or T568B wiring standards to determine order of colored wires

iv) Trim Wires: Trim wires to an even length, leaving about 0.5 inches of exposed wire beyond outer insulation

v) Insert wires: Carefully insert wires into RJ-45 connector ensuring each wire goes to corresponding slot. Push the wires so that they make contact with pins inside connector

vi) Check Alignment: Double check that wires are in correct order and fully seated in the connector and visible through the clear plastic

vii) Crimp Connector: Use a crimping tool specifically designed for RJ-45 connectors to secure the connector onto the cable.

viii) Test: Once connector is crimped, perform connectivity test to ensure cable is properly terminated. A cable tester is used to check if cable works as expected

Remarks
24/7/23

(15)

Niyati Savant
C31-TE
2103156

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

31/7/23

Experiment-02

Aim: Implementation of Hamming code for Error detection and correction

Theory:-

Hamming code is a set of error-correction codes that can be used to detect and correct the errors that occurs when data is moved from sender to receiver. It was developed by R.W Hamming for error correction. Redundant bits (Extra binary bits) are generated and added to the information-carrying bits of data transfer to ensure that no bits were lost.

The formula to find number of redundant bits is -

$$2^r \geq m + r + 1$$

where m : data bit

r : redundant bit

The value of these parity/redundant bits is such that an even or odd parity is maintained. In case of even parity, if the count of number of 1's is odd, parity bit is 1 else 0. For odd parity, if the count of 1's is odd, parity bit is 0 else 1.

The redundant bits are placed at positions that are numbered corresponding to the power of 2 i.e 1, 2, 4, 8 .... and so the data bits take up positions 3, 5, 6, 7 ....

To assign the bit value for a redundant bit we do the following.

- Parity bit P_1 checks data bits which have 1 in the LSB ie (001, 011, 101, 111 ... etc) Thus P_1 checks bits 1, 3, 5, 7, 9, 11 etc.
- Parity bit P_2 checks bits which have 1 in the second least significant position (010, 011, 110, 111 etc) Thus P_2 checks bits 2, 3, 6, 7 etc
- Parity bit P_4 checks bits with 1 in the binary at third least significant place ie (100, 101, 110, 111) ie bit location 4, 5, 6, 7 etc.

When the data is send, these parity bits are rechecked and if they voilate their parity their value is set to 1 else set to 0. These parity bits are then checked in reverse order to find the error bit which is inverted to get the correct code.

Some of its features are

- It can detect and correct single bit errors but only detect double bit errors.
- It is a relatively simple and efficient technique which makes it ideal for low-power and low-bandwidth communication networks
- Has wide variety of applications including telecommunications, computer networks & data storage systems

iyati Savant
C31 - TE
2103156

Example - Consider the data "1011" to be shared

As there are 4 data bits, we need 3 redundant bits since

$2^3 \geq 3 + 4 + 1$ (as $8 \geq 8$)

Thus parity bits are P1, P2 and P4
The values of these parity bits are -

P1 - Check positions 1, 3, 5, 7 ie 1 at 3 positions.
For Even parity P1 is set to 1

P2 - Check positions 2, 3, 6, 7 ie 1 at 2 positions and to maintain parity P2 is set to 0

P4 - Check positions 4, 5, 6, 7 ie 1 at 2 positions and P4 is 0

Thus, hamming code (has m+r ie 4+3 = 7 bits):

D7	D6	D5	P4	D3	P2	P1
1	0	1	0	1	0	1

Suppose, due to some error, the data bit D7 was changed and the data ~~sent that was~~ received was "0010101".

At the receiver end, the parity bits are checked again.

For P1, ~~there are~~ we check bits 1,3,5,7, as there are 3 1's, the bit value is set to 1.

For P2 we check bits 2,3,6,7 and there ~~are~~ is one 1 thus bit value is set to 1.

For P4, we check 4,5,6,7 there is only one 1 and bit value is 1.

Thus there is error in bit 111 ie bit 7
Therefore corrected code is 1010101.

Aim -

Program: Hamming Code For Error Detection And Correction

Code:

```
#include <stdio.h>
#include <math.h>
int total_bits,hamming[20],received_data[20];

int pow_of_two(int x)
{
   for(int i=0;i<10;i++)
   {
      if(pow(2,i)==x)
         return 1;
   }
   return 0;
}

int parity_bit(int a)
{
   int i=a,count_no=a;
   int b=0;
   int bit[20];
   int arr_len = total_bits-a+1,count_one=0; //9-2+1=8
   for(b=0;b<arr_len ;b++,i++)
      bit[b]=i;
   b=0;
   while(b<arr_len)
   {
      if(count_no!=0)
      {
         if(hamming[bit[b]]==1)
```

```
        {
            count_one++;
        }
        count_no--;
        b++;
      }
      else
      {
        count_no=a;
        b+=a;
      }
   }
   if(count_one%2==0)
      return 0;
   else
      return 1;
}

int bit_checker(int a)
{
   int i=a,count_no=a;
   int b=0;
   int bit[20];
   int arr_len = total_bits-a+1,count_one=0; //9-2+1=8
   for(b=0;b<arr_len ;b++,i++)
      bit[b]=i;
   b=0;
   while(b<arr_len)
   {
      if(count_no!=0)
      {
```

```
            if(received_data[bit[b]]==1)
            {
               count_one++;
            }
            count_no--;
            b++;
         }
         else
         {
            count_no=a;
            b+=a;
         }
      }
      if(count_one%2==0)
         return 0;
      else
         return 1;
}

int main()
{
   int p,i,input_data[15],n,k,j,x,error[5],error_bit;
   for(i=0;i<5;i++)
      error[i]=0;
   printf("Niyati's program for Hamming Code \n");
   printf("Enter the number of data bits: ");
   scanf("%d",&n);

   printf("Enter data bits- \n");
   for(i=1;i<=n;i++)
      scanf("%d",&input_data[i]);
```

```
/*No of Parity Bits */
for(i=1;i<10;i++)
{
   if(pow(2,i)>=(n+i+1))
   {
      p=i;
      break;
   }
}
printf("\nNo. of parity bits = %d \n",p);

// Finding Hamming Code
k=1;
total_bits = n+p;
for(i=1;i<=total_bits;i++)
{
   hamming[i]=111;
}
printf("Hamming Code is --\n");
for(i=total_bits;i>=1;i--)
{
   //Find Parity bits
   if (pow_of_two(i))
   {
      printf(" P%d",i);
      hamming[i]=parity_bit(i);
   }
   else
   {
```

```
        hamming[i]=input_data[k];
        printf(" D%d",i);
        k++;
    }
}
printf("\n");
for(i=total_bits;i>=1;i--)
    printf("  %d",hamming[i]);

printf("\nEnter received code:");
for(i=total_bits;i>=1;i--)
    scanf("%d",&received_data[i]);


printf("\nError bit in binary is: ");
k=5;
for(i=total_bits;i>=1;i--)
{
    if (pow_of_two(i))
    {
        error[k]=bit_checker(i);
        k--;
    }
}
for(i=5;i>k;i--)
    printf("%d",error[i]);
    //k=2

error_bit=0;
for(i=k+1,j=0;i<=5;i++,j++)
{
```

```
        error_bit +=error[i]*pow(2,j);
    }
    printf(" i.e in binary bit D%d ",error_bit);

    if (received_data[error_bit]==0)
        received_data[error_bit]=1;
    else
        received_data[error_bit]=0;
    printf("\nThus the corrected code is : ");
    for(i=total_bits;i>=1;i--)
        printf("%d",received_data[i]);

    return 0;
}
```

Output:

```
Niyati's program for Hamming Code
Enter the number of data bits: 4
Enter data bits-
1 0 1 1
No. of parity bits = 3
Hamming Code is --
 D7 D6 D5 P4 D3 P2 P1
  1  0  1  0  1  0  1
Enter received code:0 0 1 0 1 0 1
Error bit in binary is: 111 i.e in binary bit D7
Thus the corrected code is : 1010101
```

Niyati Savant
TE - C31
2103156

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

7/8/23 (11)

Experiment -03

Aim: Implementation of CRC (Cyclic Redundancy Code) for Error Detection

Theory-

CRC or Cyclic Redundancy Check is a method of detecting accidental changes / errors. It includes only enough redundancy to allow receiver to deduce that an error has occurred but not which error and have it request a retransmission

~~The data is~~

It uses a generator polynomial which is available to both sender and receiver side.

The generator is of type $x^3 + x + 1$ i.e 1011 or $x^2 + 1$ shows 101.

n : No of bits in data

k : no of bits in key obtained by generator polynomial.

Steps for data encoding at sender side:

i) Binary data is augumented by adding (k-1) zeros at the end

ii) Use modulo-2 binary division to divide binary data by key and store remainder of division

iii) Append the remainder at end of data to form encoded data and send it

Steps for decoding data at receiver side

i) Perform modulo-2 division again and if remainder is 0, there are no errors.

Example data - 100100 and generator polynomial is $x^3 + x^2 + 1$ or 1101.

→ As $k = 4$, we need to append 3 zeros at the end.

Thus dividend = 100100 000
divisor = 1101

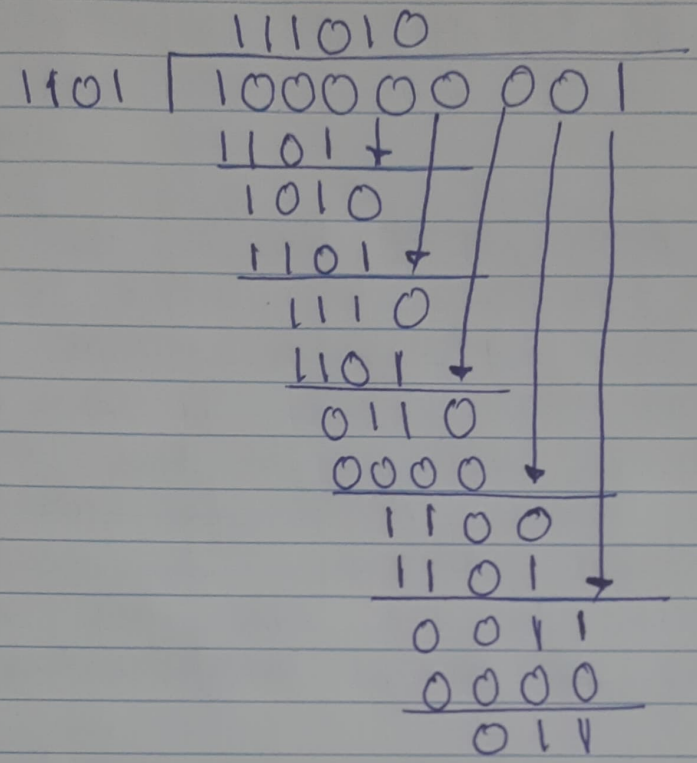
```
      111101
    ------------
1101 | 100100 000
       1101
       ----
        1000
        1101
        ----
         1010
         1101
         ----
          1110
          1101
          ----
          0110
          0000
          ----
          1100
          1101
          ----
           001
```

In case, left most bit is 0 → we divide by 0 and not the divisor

The encoded data is - 100100 001.

Suppose there is some error in transmission media and data received at other end is 100000 001

Niyati Savant
TE - C31
2103156



Since remainder is Not all zeros, error is detected at receiver side.

A

Barcode
2103123
156

Program: Cyclic Redundancy Check(CRC) Code For Error Detection

Code:

```
#include<stdio.h>
#include<string.h>
#define N strlen(gen_poly)
char data[30],check_value[30], gen_poly[10];
int data_length,i,j;

void XOR(){
   for(j = 1;j < N; j++)
   check_value[j] = (( check_value[j] == gen_poly[j])?'0':'1');
}

void crc(){
   for(i=0;i<N;i++)
      check_value[i]=data[i];
   do{

      if(check_value[0]=='1')
         XOR();

      for(j=0;j<N-1;j++)
         check_value[j]=check_value[j+1];

      check_value[j]=data[i++];
   }while(i<=data_length+N-1);

}

void receiver(){
```

```
    printf("\n \nEnter the received data: ");
    scanf("%s", data);

    crc();

    for(i=0;(i<N-1) && (check_value[i]!='1');i++);
        if(i<N-1)
            printf("\nError detected\n");
        else
            printf("\nNo error detected\n");
}


int main()
{
    printf("\n Niyati's program for CRC Code");
    printf("\nEnter data to be transmitted: ");
    scanf("%s",data);
    printf("\n Enter the Generating polynomial aka divisor: ");

    scanf("%s",gen_poly);

    data_length=strlen(data);

    for(i=data_length;i<data_length+N-1;i++)
        data[i]='0';

    printf("\n Data padded with n-1 zeros aka divident : %s",data);
```

```
    crc();

    printf("\nCRC or Check value is : %s",check_value);

    for(i=data_length;i<data_length+N-1;i++)
        data[i]=check_value[i-data_length];


    printf("\n Final data to be sent : %s",data);

    receiver();
        return 0;
}
```

Output:

Niyati's program for CRC Code

Enter data to be transmitted: 101001001

Enter the Generating polynomial aka divisor: 101011

Data padded with n-1 zeros aka divident : 10100100100000

CRC or Check value is : 10111

Final data to be sent : 10100100110111

Enter the received data: 11100100110111

Error detected

Niyati Savant
C31 - TE
2103156

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

Experiment - 04

Aim: Simulation of Go Back N flow control Algorithm

Theory:

In Stop-and-wait protocol, sender can send only one frame at a time but cannot send the next frame without receiving acknowledgment of previously sent frame. whereas in case of sliding window protocol, multiple frames can be sent at a time. 'Go-Back-N' is a variation of sliding window protocol.
N denotes the sender's window size, i.e N frames can be sent at a time before expecting an acknowledgment from the receiver.

The frames are numbered sequentially as multiple frames are send at a time and the numbering approach is used to distinguish the frame and these numbers are called sequential numbers. If the acknowledgement of a frame is not received within an agreed-upon time, or an error in any of the frames is received, all frames in current window are retransmitted. It is an approach to deal with errors in the presence of pipelining. It discards all subsequent frames following a damaged frames and sending no acknowledgements. Eventually as sender times out and retransmits all unacknowledged frames in order starting with the damaged or lost one.

Example-
Assume there are 2 bits in sequence number. Thus window size ie 2^n is 4 i.e. 4 frames can be transmitted

Sending Window [0123] 0123.....
Sender has sent 4 frames from 0 to 3 and waits for acknowledgement. Suppose acknowledgement for 0 and 1 is received.
The window slides to right: [2301] 2301
The Sender sendes next two frames 01. After receiving acknowledgemet for 2,3 window slides to right: [0123] 01
Suppose acknowledgement for fram 0 is received but frame 1 is lost however frames 2,3 are received. Even then, receiver discards 2,3 Thus the receiving station does not acknowledge each received frame explecitly. If a sending station received an acknowledgement for frame i and later receives an acknowledgement for frame k it assumes all frames between i and k have been received correctly. Thus it reduces no. of acknowledge- and thereby the network traffic

If k are the no. of bits in sequence number then frames are numbered from 0 to $2^k - 1$.

- If sender's window size is greater than 2^k

Niyati Savant
C31 - TE
2103156

Eg - K = 3, i.e $2^k = 8$ but if window size be 9
∴ Sender's window → |0 1 2 3 4 5 6 7 0| 1 2 3 . . . .

When sender receives an ACK0, it does not know if it is for the first frame or last.

- If window size is 2^k

Eg ∴ Window size is 8
Sender's window → |0 1 2 3 4 5 6 7| 0 1 2 3 . . . .

Suppose at time t_1, A sends frame 0-7 to B and B receives each one correctly. At t_2, B sends acknowledgment for most recent frame ACK7 but it gets lost. Next frame expected by B is 0. At t_3, as A does not get acknowledgment it resends frames 0 to 7. At t_4, B gets frame 0 and it accepts that. However,

A sends: 0 1 2 3 4 5 6 7, 0 1 2 3
and B was expecting 0 1 2 3 4 5 6 7, 0 1 2 3 4 5 6 7,
Thus, protocol fails as B accepts a duplicate frame

- If window size is less than 2^k
ie window size is 7
senders window → |0 1 2 3 4 5 6| 7 0 1 2 3 4

As seen in previous scenario, B receives each

frames 0 to 6 correctly and sends ACK 6 but it gets lost. B now expects frame 7. As A does not received any acknowledgement, it retransmits frame 0 to 6 and as B was expecting 7, it ignores them and eventually sends ACK 6 which is received by A and it advances its window.

Sender's window: 0 1 2 3 4 5 6 | 7 0 1 2 3 4 . . .

As frames are received in order, receiver's window size must not be greater than 1

The drawback of Go-back-N is that receiver discards all the correct frames transmitted after the bad one and the channel bandwidth is wasted on retransmitted frames.

Remark
25/9/23

(15)

Program: Go Back N Flow Control Protocol

Code:

```
#include <stdio.h>
int main()
{
 int total_frames, window_size, i, error_frame, k = 1, count = 0, ack_rec = 0;
printf("\n Niyati's program for Go Back N protocol ");
 printf("Enter the number of frames: ");
 scanf("%d", &total_frames);
 printf("Enter the Window Size: ");
 scanf("%d", &window_size);
 printf("\n");

 // k-no. of frames for which ack. is received
 while (ack_rec != total_frames)
 {
  for (i = ack_rec + 1; i <= ack_rec + window_size && i != total_frames + 1; i++)
  {
   printf("Send Frame %d \n", i);
   count++;
  }
  printf("\n");

  printf("Enter Frame number that has error(In case of no error,enter 100): ");
  scanf("%d", &error_frame);

  if (error_frame == 100)
  {
   for (i = k; i < k + window_size && i != total_frames + 1; i++)
   {
```

```
      printf("Acknowledgement for Frame %d \n", i);
      ack_rec++;
     }
     k = i;
     printf("\n");
    }

    else
    {
     for (i = k; i < error_frame && i != total_frames + 1; i++)
     {
      printf("Acknowledgement for Frame %d \n", i);
      ack_rec++;
     }

     k = i;

     printf("Frame %d Not received \n", error_frame);
     printf("Retransmitting Window \n");

     printf("\n");
    }
  }
  printf("\n Total Transmissions:%d", count);
  return 0;
}
```

Output :

Niyati's program for Go Back N protocol

Enter the number of frames: 10

Enter the Window Size: 3

Send Frame 1

Send Frame 2

Send Frame 3

Enter Frame number that has error (In case of no error, enter 100): 2

Acknowledgement for Frame 1

Frame 2 Not received

Retransmitting Window

Send Frame 2

Send Frame 3

Send Frame 4

Enter Frame number that has error (In case of no error, enter 100): 100

Acknowledgement for Frame 2

Acknowledgement for Frame 3

Acknowledgement for Frame 4

Send Frame 5

Send Frame 6

Send Frame 7

Enter Frame number that has error(In case of no error, enter 100): 6

Acknowledgement for Frame 5

Frame 6 Not received

Retransmitting Window

Send Frame 6

Send Frame 7

Send Frame 8

Enter Frame number that has error (In case of no error, enter 100): 100

Acknowledgement for Frame 6

Acknowledgement for Frame 7

Acknowledgement for Frame 8

Send Frame 9

Send Frame 10

Enter Frame number that has error (In case of no error, enter 100): 10

Acknowledgement for Frame 9

Frame 10 Not received

Retransmitting Window

Send Frame 10

Enter Frame number that has error (In case of no error, enter 100): 100

Acknowledgement for Frame 10

Total Transmissions:15

Niyati Savant
Niyati Savant
C-31
2103156

Experiment - 05

Aim - Build a simple network topology and configure it for static routing protocol using packet tracer. Setup a network and configure IP addressing, subnetting, masking

Theory -

Routing is a process performed by network layer devices to deliver the packet by choosing an optimal path from one network to another. Static routing is a process in which we have to manually add routes to the routing table. Thus, there is no overhead for router CPU, adds security and there is no bandwidth usage between routers. However, for large networks, manually adding each route is hectic and the administrator must have a good knowledge of topology.

Steps to Configure and Verify Two router connections in Cisco Packet Tracer:

1) Open cisco packet tracer desktop and select the devices given below:
   a) PC → 4
   b) PT-Switch → 2
   c) PT-Router → 2

IP addressing Table for PCs:

Sr no	Device	IPv4 Address	Subnet mask	Default Gateway
1	PC0	192.168.1.2	255.255.255.0	192.168.1.1
2	PC1	192.168.1.3	255.255.255.0	192.168.1.1
3	PC2	192.168.2.2	255.255.255.0	192.168.2.1
4	PC3	192.168.2.3	255.255.255.0	192.168.2.1

Then create the network topology and use connecting cable to connect the devices with others

2) Configure PCs (hosts) with IPv4 address and Subnet Mask according to the IP addressing table above.
- To assign an IP address in PC0, click on PC0
- Then, go to desktop and then IP configuration and there you will have IPv4 configuration.
- Fill IPv4 address and subnet mask

3) Assigning IP address using ipconfig command
- We can also assign an IP address using command.
- Go to command terminal of PC
- Then, type - Ipconfig <IPv4 address> <subnet mask> <default gateway>
- Repeate the same with other PCs

Niyati Savant
C31
2103156

4) Configure router with IP address and subnet mask

Sr no	Device	Interface	IPv4 Addressing	Subnet Mask
1	router0	Fast Ethernet 0/0	192.168.1.1	255.255.255.0
		Serial 2/0	11.0.0.1	255.255.255.0
2	router1	Fast Ethernet 0/0	192.168.2.1	255.255.255.0
		serial 2/0	11.0.0.2	255.255.255.0

- To assign an IP address in router 0, click on router 0.
- Then, go to config and then Interfaces.
- Then, configure the IP address in Fast Ethernet and serial ports as per table.
- Fill IPv4 address and subnet mask
- Repeat for other routers

5) After configuring all of the devices we need to assign routes to the routers
- First, click on router0 then Go to CLI
- Then type the command -
  ip route <network id> <subnet mask> <next hop>

6) Verify network by pinging the IP address

of any PC. We use ping command to do so.
- First click on PC1 then go to the Command prompt
- Then type ping <IP address of target node>
- If we get replies, the connection is working fine

For simulation we send a PDU (protocol data unit) from PC0 to PC2

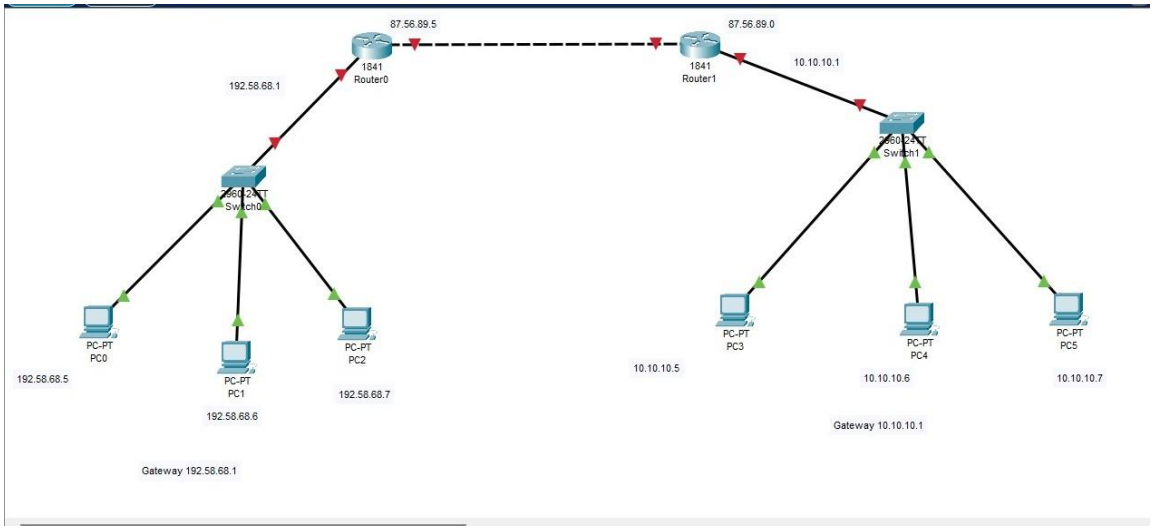
Signed 31/10/22

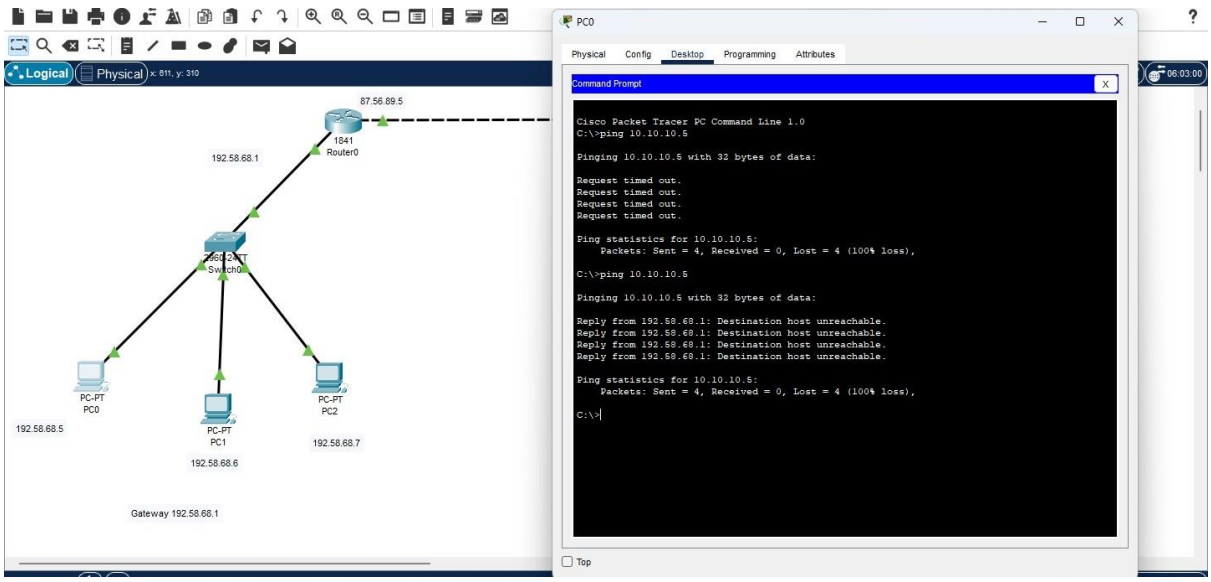
15

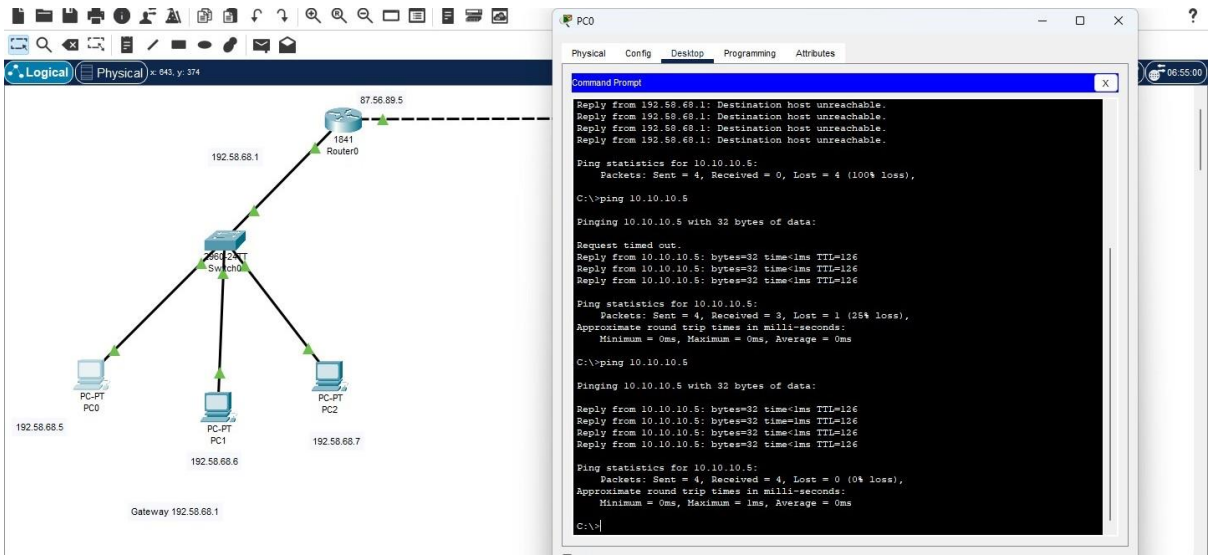
Program: Network Topology using Packet Tracer

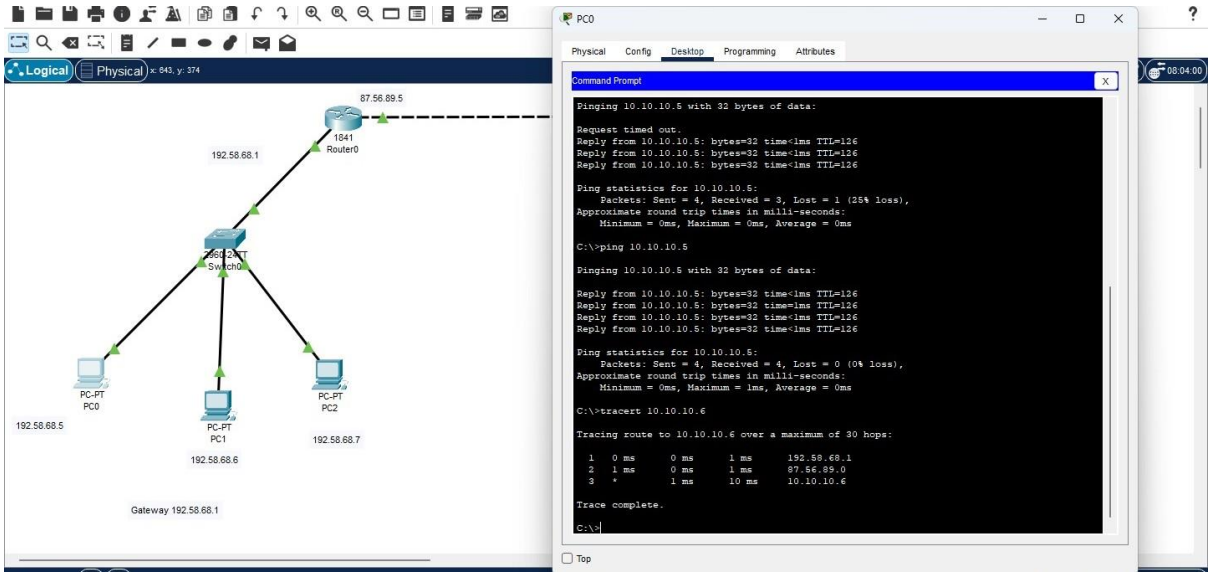
Aim- Build a simple network topology and configure it for static routing protocol using packet tracer set up a network and configure IP addressing subnetting masking.

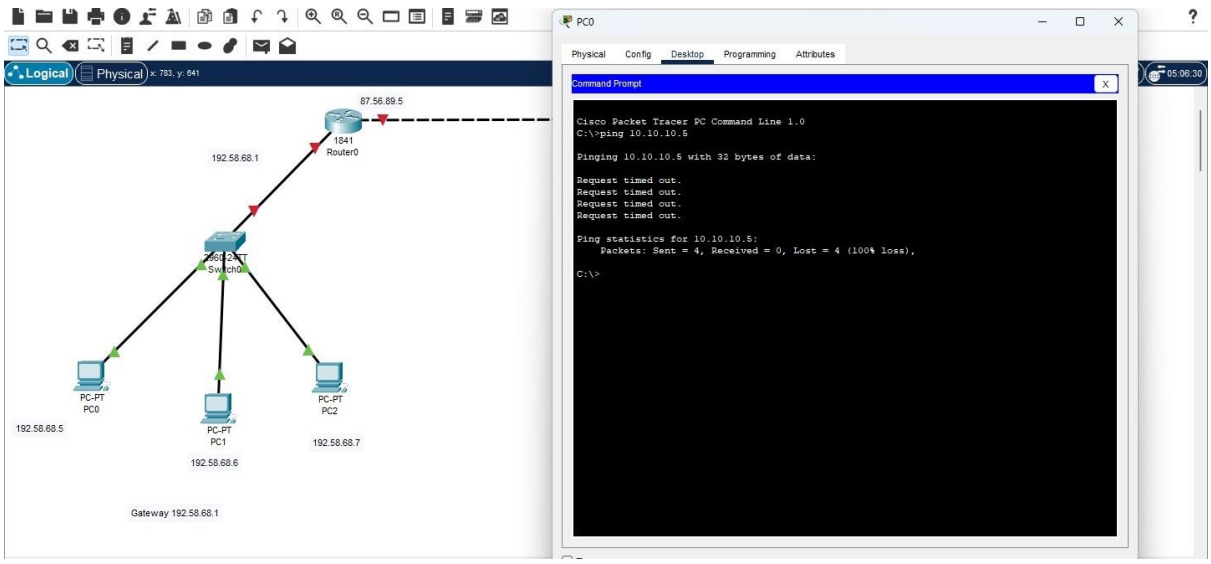
Output:

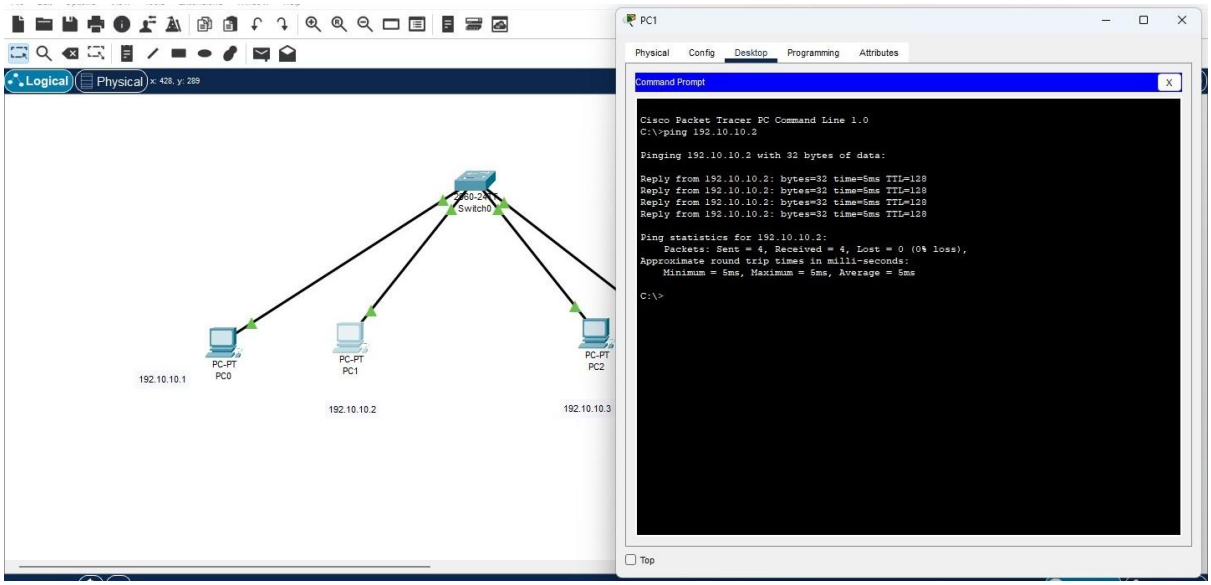





Logical
Physical
x: 843, y: 374
87.56.89.5
1841
Router0
192.58.68.1
PC-PT
PC0
PC-PT
PC1
PC-PT
PC2
192.58.68.5
192.58.68.6
192.58.68.7
Gateway 192.58.68.1
PC0
Physical
Config
Desktop
Programming
Attributes
Command Prompt
Reply from 192.58.68.1: Destination host unreachable.
Reply from 192.58.68.1: Destination host unreachable.
Reply from 192.58.68.1: Destination host unreachable.
Reply from 192.58.68.1: Destination host unreachable.
Ping statistics for 10.10.10.5:
Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),
C:\>ping 10.10.10.5
Pinging 10.10.10.5 with 32 bytes of data:
Request timed out.
Reply from 10.10.10.5: bytes=32 time<1ms TTL=126
Reply from 10.10.10.5: bytes=32 time<1ms TTL=126
Reply from 10.10.10.5: bytes=32 time<1ms TTL=126
Ping statistics for 10.10.10.5:
Packets: Sent = 4, Received = 3, Lost = 1 (25% loss),
Approximate round trip times in milli-seconds:
Minimum = 0ms, Maximum = 0ms, Average = 0ms
C:\>ping 10.10.10.5
Pinging 10.10.10.5 with 32 bytes of data:
Reply from 10.10.10.5: bytes=32 time<1ms TTL=126
Reply from 10.10.10.5: bytes=32 time<1ms TTL=126
Reply from 10.10.10.5: bytes=32 time<1ms TTL=126
Reply from 10.10.10.5: bytes=32 time<1ms TTL=126
Ping statistics for 10.10.10.5:
Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
Minimum = 0ms, Maximum = 1ms, Average = 0ms
C:\>


Logical
Physical
x: 843, y: 374
87.56.89.5
1841
Router0
192.58.68.1
PC-PT
PC0
PC-PT
PC1
PC-PT
PC2
192.58.68.5
192.58.68.6
192.58.68.7
Gateway 192.58.68.1
PC0
Physical
Config
Desktop
Programming
Attributes
Command Prompt
Pinging 10.10.10.5 with 32 bytes of data:
Request timed out.
Reply from 10.10.10.5: bytes=32 time<1ms TTL=126
Reply from 10.10.10.5: bytes=32 time<1ms TTL=126
Reply from 10.10.10.5: bytes=32 time<1ms TTL=126
Ping statistics for 10.10.10.5:
Packets: Sent = 4, Received = 3, Lost = 1 (25% loss),
Approximate round trip times in milli-seconds:
Minimum = 0ms, Maximum = 0ms, Average = 0ms
C:\>ping 10.10.10.5
Pinging 10.10.10.5 with 32 bytes of data:
Reply from 10.10.10.5: bytes=32 time<1ms TTL=126
Reply from 10.10.10.5: bytes=32 time<1ms TTL=126
Reply from 10.10.10.5: bytes=32 time<1ms TTL=126
Reply from 10.10.10.5: bytes=32 time<1ms TTL=126
Ping statistics for 10.10.10.5:
Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
Minimum = 0ms, Maximum = 1ms, Average = 0ms
C:\>tracert 10.10.10.6
Tracing route to 10.10.10.6 over a maximum of 30 hops:
1 0 ms 0 ms 1 ms 192.58.68.1
2 1 ms 0 ms 1 ms 87.56.89.0
3 * 1 ms 10 ms 10.10.10.6
Trace complete.
C:\>
Top


Logical
Physical
x: 783, y: 641
87.56.89.5
1841
Router0
192.58.68.1
2960-24TT
Switch0
PC-PT
PC0
PC-PT
PC1
PC-PT
PC2
192.58.68.5
192.58.68.6
192.58.68.7
Gateway 192.58.68.1
PC0
Physical
Config
Desktop
Programming
Attributes
Command Prompt
Cisco Packet Tracer PC Command Line 1.0
C:\>ping 10.10.10.5
Pinging 10.10.10.5 with 32 bytes of data:
Request timed out.
Request timed out.
Request timed out.
Request timed out.
Ping statistics for 10.10.10.5:
Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),
C:\>


Logical
Physical
x: 428, y: 289
2960-24TT
Switch0
PC-PT
PC0
PC-PT
PC1
PC-PT
PC2
192.10.10.1
192.10.10.2
192.10.10.3
PC1
Physical
Config
Desktop
Programming
Attributes
Command Prompt
Cisco Packet Tracer PC Command Line 1.0
C:\>ping 192.10.10.2
Pinging 192.10.10.2 with 32 bytes of data:
Reply from 192.10.10.2: bytes=32 time=5ms TTL=128
Reply from 192.10.10.2: bytes=32 time=5ms TTL=128
Reply from 192.10.10.2: bytes=32 time=5ms TTL=128
Reply from 192.10.10.2: bytes=32 time=5ms TTL=128
Ping statistics for 192.10.10.2:
Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
Minimum = 5ms, Maximum = 5ms, Average = 5ms
C:\>
Top

Niyati Savant
C31
2103156

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

Experiment - 06

Aim - Design VPN and Configure RIP/OSPF using Packet tracer.

Theory -

Routing Information Protocol (RIP) is a dynamic routing protocol that uses hop count as a routing metric to find the best path between source and destination network. and is a distance-vector routing protocol. It uses port 520. Hop count is the number of routers occuring in between source and destination network. The best path is the one with lowest hop count and is thus placed in routing table. Maximum hop count allowed is 15.

Implementing RIP routing

1) Select the devices given below.
   - PC - 4
   - PT-Switch - 2
   - PT-Router - 2

   Create the network topology as shown and use connecting cables to connect them.

2) Configure PCs (hosts) with IPv4 address and subnet mask as per IP addressing in the table

Srno	Device	IPv4 Address	Subnet-Mask	Gateway
1	PC0	192.168.10.2	255.255.255.0	192.168.10.1
2	PC1	192.168.10.3	255.255.255.0	192.168.10.1
3	PC2	192.168.20.2	255.255.255.0	192.168.20.1
4	PC3	192.168.20.3	255.255.255.0	192.168.20.1

- Assign IP address using ipconfig command -
  ipconfig 192.168.10.2 255.255.255.0 192.168.10.1
- Repeat with other PCs
  Also ~~assign~~ another way is to click on the PC, go to desktop, IP Configuration where IPv4 address and subnet mask is filled.

3) Configure routers with - IP address and subnet mask

IP addressing Table Router

Srno	Device	Interface	IPv4 address	Subnet Mask
1	router 0	Fast Ethernet 0/0	192.168.10.1	255.255.255.0
		Serial 2/0	10.0.0.1	255.0.0.0
2	router 1	Fast Ethernet 0/0	192.168.20.1	255.255.255.0
		serial 2/0	10.0.0.2	255.0.0.0

To assign IP address in router0, click on it,

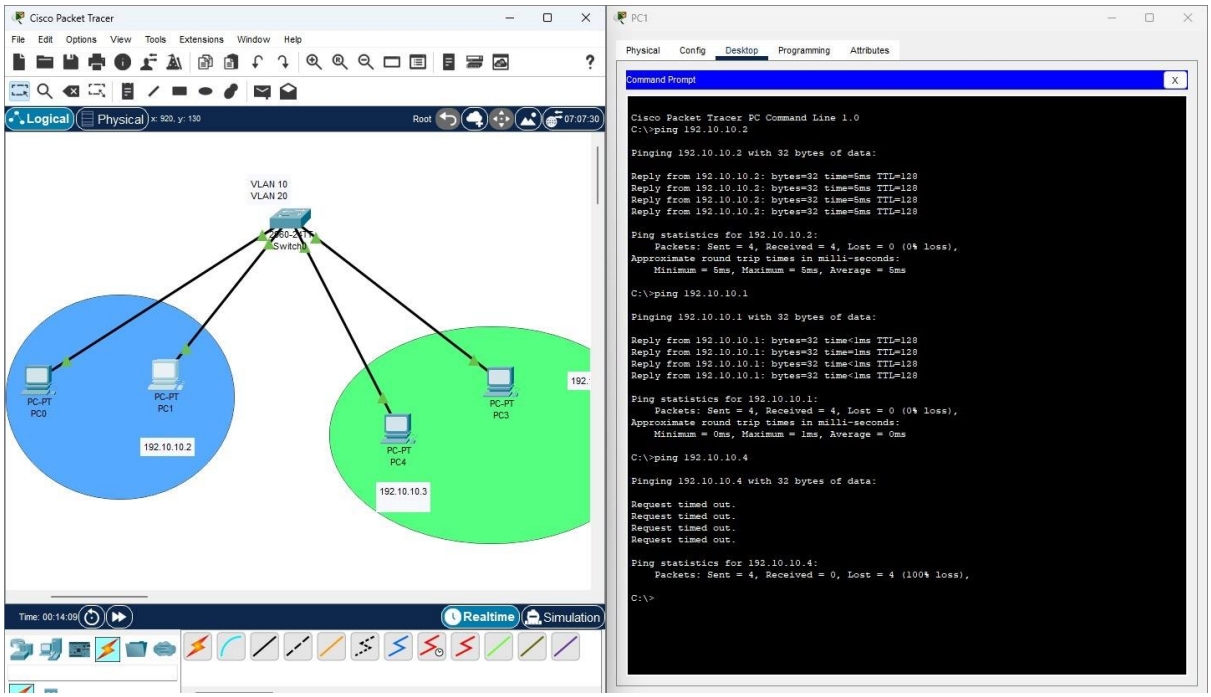
goto config and interfaces. Turn 'On' the ports then config IP address as per the table and fill IPv4 address and subnet mask Repeat with other routers

4) After configuring all devices we assign routes to routers.
First click on router 0 - go to CLI and type - network <network id>

5) Verify the network by pinging the IP address of any PC.
- Click on PC0, go to command prompt
- Then type ping <IP address of target>
- If we get replies, connection is working fine.

Karnade
3/10/22

Design VPN and configure RIP/OSPF using Packet tracer



Niyati Savant
C31
2103156

THADOMAL SHAHANI

THADOMAL SHAHANI
TSEC
ENGINEERING COLLEGE

Experiment - 07

(30)

Aim - Implementation of IPV4 addressing concept along with subnet masking.

Theory -

IP stands for Internet Protocol and v4 stands for Version Four. It was the primary version brought into action for production within the ARPANET in 1983. These are 32-bit integers which will be expressed in decimal notation. They are used in source address and destination address fields of IP header.

The IPv4 protocol has an address space and is defined as total number of addresses used by the protocol. If N number of bits are used for defining an address then address space will be 2^N addresses. For IPv4, N is 32 bits and hence its address space is 2^{32} ie 4,294,967,296. In dotted decimal notation, the range is from 0.0.0.0 to 255.255.255.255

The IPv4 address has 2 parts - network part that is used to identify network and the second part is the host id which identifies a host on that network. These are of variable lengths, depending on the class of address. The IP address space is divided

into the following 5 classes -

Class A - Network field is 7 bit long (numbered between 1 to 126) and 24 bit length for host field (to range from 0.0.0.0 to 127.255.255.255). Thus there are 126 types of network. It can be identified by "0" in the first field.

Class B - The first two fields identify network and number in the first field must be in the range 128-191. Network field has 14 bits and Host has 16 bits.

Class C - The network field has 21 bits and Host has 8 bits. The first block in Class C covers addresses from 192.0.0.0 to 192.0.0.255 and last block covers addresses from 223.255.255.0 to 223.255.255.255.

Class D - It starts with 1110 and then multicast address and also allows upto 2 million networks with upto 254 hosts each

Class E - It starts with 1110 and thus is reserved for future use

Thus, class can be identified by class A : 0-127, Class B: 128-191, Class C: 192-223, 224-239 for class D and class E: 240-255

The value of n for class A is 8, 16 for B and 24 for C.

Total number of IPv4 addresses in a block will be $N = 2^{(32-n)}$

The first address will be obtained by keeping leftmost n bits intact & setting (32-n) rightmost bit to 0's. The last address will be obtained by keeping leftmost "n" bits in address as it is and setting all (32-n) rightmost bits to 1.

A network mask in classful addressing is defined as a 32 bit number obtained by setting all the "n" leftmost bits to 1s and (32-n) bits to 0.

For example, a class B address has n = 16

∴ default mask will be 255.255.0.0 or

11111111 11111111 00000000 00000000

n=16 32-16=16

Subnetting is the principle of splitting a block of addresses into smaller blocks of addresses and each subnet has its own subnet. Each subnet has its own subnet address, net id and host id. To divide a network into 8 subnets, corresponding subnet mask has 3 extra 1's as $2^3 = 8$

158

Sarode 3/10/22

PROGRAM : IPv4 ADDRESSING AND SUBNET MASKING

Code:

```
import math

def findClass(ip):
    if 0 <= ip[0] <= 127:
        print("Network Address is : ", ip[0])
        print('No. of IP addresses possible : ', 2 ** 24)
        return "A", '255.0.0.0'
    elif 128 <= ip[0] <= 191:
        ip = [str(i) for i in ip]
        print("Network Address is : ", ".".join(ip[0:2]))
        print('No. of IP addresses possible : ', 2 ** 16)
        return "B", '255.255.0.0'
    elif 192 <= ip[0] <= 223:
        ip = [str(i) for i in ip]
        print("Network Id is : ", ".".join(ip[0:3]))
        print('No. of IP addresses possible : ', 2 ** 8)
        return "C", '255.255.255.0'
    elif 224 <= ip[0] <= 239:
        print("In this Class, IP address is not divided into Network and Host
ID")
        return "D"
    else:
        print("In this Class, IP address is not divided into Network and Host
ID")
        return "E"


def Subnetting(ip, num, className, ip_addresses):
    temp = 0
    if className == "A":
        place2 = ip_addresses / (256 ** 2)
        for i in range(num):
            print(f"Subnet {i} => ")
            print(temp)
            print("Subnet Address : ", ip[0] + '.' + str(temp) + '.0' + '.0')
            temp += int(place2)
            print("Broadcast address : ", ip[0] + '.' + str(temp - 1) + '.255'
+ '.255')
            print("Valid range of host IP address : ",
                  ip[0] + '.' + str(temp - int(place2)) + '.' + '0' + '.1' +
'\t-\t' + ip[0] + '.' + str(
                      temp - 1) + '.254' + '.254')
            print()
    elif className == "B":
```

```
        place2 = ip_addresses / 256
        for i in range(num):
            print(f"\nSubnet {i} => ")
            print("Subnet Address : ", ".".join(ip[0:2]) + '.' + str(temp) +
'.0')
            temp += int(place2)
            print("Broadcast address : ", ".".join(ip[0:2]) + '.' + str(temp -
1) + '.255')
            print("Valid range of host IP address : ",
                  ".".join(ip[0:2]) + '.' + str(temp - int(place2)) + '.1\t-
\t' + ".".join(ip[0:2]) + '.' + str(
                      temp - 1) + '.254')
            print()
    elif className == "C":
        for i in range(num):
            print(f"\nSubnet {i} => ")
            print("Subnet Address : ", ".".join(ip[0:3]) + '.' + str(temp))
            temp += int(ip_addresses)
            print("Broadcast address : ", ".".join(ip[0:3]) + '.' + str(temp -
1))
            print("Valid range of host IP address : ",
                  ".".join(ip[0:3]) + '.' + str(temp - int(ip_addresses) + 1)
+ '\t-\t' + ".".join(ip[0:3]) + '.' + str(
                      temp - 2))
            print()
    else:
        print("In this Class, IP address is not divided into Network and Host
ID")


def subnetmask(num, network_mask):
    var = '1' * int(math.log(num, 2))
    var1 = '0' * (8 - int(math.log(num, 2)))
    binary_num = var + var1
    network_mask = network_mask.split('.')
    network_mask = [i for i in network_mask if i != '0']
    network_mask.append(str(int(binary_num, 2)))
    while len(network_mask) < 5:
        network_mask.append('0')
    print('Subnet Mask - ', ".".join(network_mask[0:4]))


print("Niyati's code for IPv4 addressing ")
ip = input("Enter the IP address : ")
# ip = "192.168.123.0 "
ip = ip.split(".")

ip = [int(i) for i in ip]
lst = findClass(ip)
```

```
networkClass = lst[0]
print("Given IP address belongs to class : ", networkClass)
ip = [str(i) for i in ip]
network_mask = lst[1]
print('Network Mask : ', network_mask)
num_subnet = int(input('\nNo. of subnets(power of 2) : '))
num_ip = int(2 ** (8 * (68 - ord(networkClass))) / num_subnet)
print(num_ip)
print('The no. of bits in the subnet id : ', int(math.log(num_subnet, 2)))
if ord(networkClass) < 68:
    print('Total no. of IP addresses possible in each subnet : ', num_ip)
Subnetting(ip, num_subnet, networkClass, num_ip)
subnetmask(num_subnet, network_mask)
```

OUTPUT:

Niyati's code for IPv4 addressing

Enter the IP address : 192.168.123.0

Network Id is : 192.168.123

No. of IP addresses possible : 256

Given IP address belongs to class : C

Network Mask : 255.255.255.0

No. of subnets(power of 2) : 4

64

The no. of bits in the subnet id : 2

Total no. of IP addresses possible in each subnet : 64

Subnet 0 =>

Subnet Address : 192.168.123.0

Broadcast address : 192.168.123.63

Valid range of host IP address : 192.168.123.1 - 192.168.123.62

Subnet 1 =>

Subnet Address : 192.168.123.64

Broadcast address : 192.168.123.127

Valid range of host IP address : 192.168.123.65 - 192.168.123.126

Subnet 2 =>

Subnet Address : 192.168.123.128

Broadcast address : 192.168.123.191

Valid range of host IP address : 192.168.123.129 - 192.168.123.190

Subnet 3 =>

Subnet Address : 192.168.123.192

Broadcast address : 192.168.123.255

Valid range of host IP address : 192.168.123.193 - 192.168.123.254

Subnet Mask – 255.255.255.192

Experiment 8

Aim- Implement Networking Commands in Linux (NIYATI SAVANT)

Commands and their explanation

ifconfig

The 'ifconfig' command in Linux is used to manage network interfaces. When executed without arguments, it displays information about all active network interfaces, including their IP addresses and MAC addresses. You can also use it to enable or disable interfaces, set IP addresses, and configure netmasks. While 'ifconfig' is still available on many systems, modern Linux distributions often recommend using the 'ip' command for more advanced networking tasks.

```
practicalexampc29@LAB306PC32:~$ ifconfig
enp1s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.31.47  netmask 255.255.255.0  broadcast 192.168.31.255
        inet6 fe80::f49a:1df0:4642:17d7  prefixlen 64  scopeid 0x20<link>
        ether a4:ae:12:84:83:16  txqueuelen 1000  (Ethernet)
        RX packets 9353  bytes 10583209 (10.5 MB)
        RX errors 0  dropped 8  overruns 0  frame 0
        TX packets 3081  bytes 249128 (249.1 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 342  bytes 34799 (34.7 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 342  bytes 34799 (34.7 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

virbr0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 192.168.122.1  netmask 255.255.255.0  broadcast 192.168.122.255
        ether 52:54:00:2a:e9:6f  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

ping

The ping command in Linux is used to test the connectivity between your computer and another host on a network, typically using the Internet Control Message Protocol (ICMP). When you run ping followed by a host or IP address, it sends a series of ICMP echo request packets to the specified host. If the host is reachable and responsive, it will reply with ICMP echo reply packets. This command is commonly used to check network connectivity, measure network latency (ping time), and troubleshoot network issues. By

default, ping sends a series of packets and displays statistics about the packets sent and received, helping you assess the quality of the network connection.

```
practicalexampc29@LAB306PC32:~$ ping google.com
PING google.com (142.250.182.206) 56(84) bytes of data.
64 bytes from bom07s28-in-f14.1e100.net (142.250.182.206): icmp_seq=1 ttl=58
time=3.50 ms
64 bytes from bom07s28-in-f14.1e100.net (142.250.182.206): icmp_seq=2 ttl=58
time=3.70 ms
64 bytes from bom07s28-in-f14.1e100.net (142.250.182.206): icmp_seq=3 ttl=58
time=3.58 ms
64 bytes from bom07s28-in-f14.1e100.net (142.250.182.206): icmp_seq=4 ttl=58
time=3.51 ms
^C
--- google.com ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3005ms
rtt min/avg/max/mdev = 3.499/3.572/3.699/0.080 ms


practicalexampc29@LAB306PC32:~$ ping 142.250.182.206
PING 142.250.182.206 (142.250.182.206) 56(84) bytes of data.
64 bytes from 142.250.182.206: icmp_seq=1 ttl=58 time=3.59 ms
64 bytes from 142.250.182.206: icmp_seq=2 ttl=58 time=3.65 ms
64 bytes from 142.250.182.206: icmp_seq=3 ttl=58 time=4.47 ms
64 bytes from 142.250.182.206: icmp_seq=4 ttl=58 time=3.53 ms
^C
--- 142.250.182.206 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3006ms
rtt min/avg/max/mdev = 3.532/3.810/4.470/0.383 ms
```

ping -c

The ping -c command is an extension of the standard ping command with the -c flag used to specify the number of ICMP echo request packets to send. When you use ping -c followed by a number, such as ping -c 5, it instructs ping to send a specific count of ICMP echo request packets to the target host or IP address. After sending the specified number of packets, ping will display a summary of the results, including statistics like packet loss, round-trip time (ping time), and more. This option is useful for running a specific number of ping tests to assess network connectivity or diagnose issues.

```
practicalexampc29@LAB306PC32:~$ ping -c 4 142.250.182.206
PING 142.250.182.206 (142.250.182.206) 56(84) bytes of data.
64 bytes from 142.250.182.206: icmp_seq=1 ttl=58 time=3.85 ms
64 bytes from 142.250.182.206: icmp_seq=2 ttl=58 time=3.47 ms
64 bytes from 142.250.182.206: icmp_seq=3 ttl=58 time=3.60 ms
64 bytes from 142.250.182.206: icmp_seq=4 ttl=58 time=3.39 ms

--- 142.250.182.206 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3005ms
rtt min/avg/max/mdev = 3.392/3.578/3.850/0.173 ms
```

ping -i5

The ping -i command is used to specify the interval between sending ICMP echo request packets when using the ping command. When you use ping -i followed by a number, such as ping -i 5, it determines the time interval (in seconds) between sending each ICMP echo request packet to the target host or IP address. Setting the interval with -i can be helpful when you want to space out the ping requests, especially in situations where you don't want to flood the network with too many requests in a short period. Adjusting the interval allows you to control the rate at which ping requests are sent, making it useful for network troubleshooting or monitoring.

```
practicalexampc29@LAB306PC32:~$ ping -i5 142.250.182.206
PING 142.250.182.206 (142.250.182.206) 56(84) bytes of data.
64 bytes from 142.250.182.206: icmp_seq=1 ttl=58 time=3.57 ms
64 bytes from 142.250.182.206: icmp_seq=2 ttl=58 time=3.66 ms
64 bytes from 142.250.182.206: icmp_seq=3 ttl=58 time=3.76 ms
^C
--- 142.250.182.206 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 10012ms
rtt min/avg/max/mdev = 3.571/3.663/3.757/0.075 ms
```

ping -R

The 'ping -R' command is used to enable record route functionality in the 'ping' command. When you use 'ping -R', it instructs the 'ping' command to include the record route option in the ICMP echo request packets it sends to the target host or IP address. This option is typically used for debugging and network analysis. When the record route option is enabled, each ICMP echo request packet includes a list of routers (hops) that the packet traverses on its way to the destination. The routers' IP addresses are recorded in the packet, allowing you to see the path that the packet takes through the network.

```
practicalexampc29@LAB306PC32:~$ ping -R 142.250.182.206
PING 142.250.182.206 (142.250.182.206) 56(124) bytes of data.
^C
--- 142.250.182.206 ping statistics ---
26 packets transmitted, 0 received, 100% packet loss, time 25592ms
```

ping -w6

The ping -w command is used to specify a timeout period for the ping command. When you use ping -w followed by a number, such as ping -w 6, it sets a timeout period in seconds for each ICMP echo request packet sent to the target host or IP address. In this example, ping will wait for up to 6 seconds for a response from the destination. If a response is not received within that time frame, ping will consider the packet lost and report it as such.Setting a timeout period with -w can be useful for controlling how long ping should wait for a response before moving on to the next packet. It's often used to adjust the behavior of the ping command based on the network conditions or specific testing requirements.

```
practicalexampc29@LAB306PC32:~$ ping -w6 142.250.182.206
PING 142.250.182.206 (142.250.182.206) 56(84) bytes of data.
64 bytes from 142.250.182.206: icmp_seq=1 ttl=58 time=3.64 ms
64 bytes from 142.250.182.206: icmp_seq=2 ttl=58 time=3.65 ms
```

64 bytes from 142.250.182.206: icmp_seq=3 ttl=58 time=3.58 ms
64 bytes from 142.250.182.206: icmp_seq=4 ttl=58 time=3.30 ms
64 bytes from 142.250.182.206: icmp_seq=5 ttl=58 time=3.41 ms
64 bytes from 142.250.182.206: icmp_seq=6 ttl=58 time=3.53 ms

--- 142.250.182.206 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5009ms
rtt min/avg/max/mdev = 3.303/3.517/3.652/0.125 ms

ping -w6 -c8

The 'ping -w' and 'ping -c' options can be used together to set both a timeout period and a specific count of ICMP echo request packets to send. 'ping -w6 -c8', will send 8 ICMP echo request packets to the target, waiting for a response for up to 6 seconds for each packet. After sending these 8 packets or when the 6-second timeout is reached for each, 'ping' will display a summary of the results, including statistics like packet loss, round-trip time (ping time), and more. This command can be useful for controlled testing and network troubleshooting with a specified timeout and packet count.

practicalexampc29@LAB306PC32:~$ ping -w6 -c8 142.250.182.206
PING 142.250.182.206 (142.250.182.206) 56(84) bytes of data.
64 bytes from 142.250.182.206: icmp_seq=1 ttl=58 time=3.60 ms
64 bytes from 142.250.182.206: icmp_seq=2 ttl=58 time=3.63 ms
64 bytes from 142.250.182.206: icmp_seq=3 ttl=58 time=3.57 ms
64 bytes from 142.250.182.206: icmp_seq=4 ttl=58 time=3.43 ms
64 bytes from 142.250.182.206: icmp_seq=5 ttl=58 time=3.78 ms
64 bytes from 142.250.182.206: icmp_seq=6 ttl=58 time=3.79 ms

--- 142.250.182.206 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5009ms
rtt min/avg/max/mdev = 3.425/3.631/3.785/0.125 ms

traceroute

The 'traceroute' command is used to trace the route that packets take from your computer to a destination host or IP address on a network. It displays a list of all the hops (routers or gateways) that the packets traverse, along with the round-trip times (RTT) for each hop. This tool is essential for network troubleshooting, diagnosing network issues, and understanding the path that data follows across the network.

practicalexampc29@LAB306PC32:~$ traceroute www.google.com
traceroute to www.google.com (142.250.183.196), 30 hops max, 60 byte packets
 1 _gateway (192.168.31.1) 1.039 ms 0.977 ms 0.948 ms
 2 203.212.25.1 (203.212.25.1) 2.229 ms 2.202 ms 2.178 ms
 3 203.212.24.53 (203.212.24.53) 2.153 ms 2.129 ms 2.104 ms
 4 10.10.226.153 (10.10.226.153) 3.313 ms * *
 5 72.14.196.213 (72.14.196.213) 5.647 ms 3.840 ms 5.599 ms
 6 108.170.248.209 (108.170.248.209) 4.392 ms 4.090 ms 4.027 ms
 7 142.251.64.11 (142.251.64.11) 3.420 ms 142.251.64.9 (142.251.64.9) 3.287 ms
3.228 ms
 8 bom07s33-in-f4.1e100.net (142.250.183.196) 3.203 ms 3.179 ms 3.727 ms

nslookup

The 'nslookup' command is used to query Domain Name System (DNS) servers to obtain information about domain names, IP addresses, and other DNS-related records. It is a tool for DNS-related tasks, such as looking up IP addresses associated with domain names (forward lookup) or finding domain names linked to IP addresses (reverse lookup). It's commonly used for DNS troubleshooting and verifying DNS configurations.

```
practicalexampc29@LAB306PC32:~$ nslookup www.google.com
Server:         127.0.0.53
Address:        127.0.0.53#53
Non-authoritative answer:
Name:www.google.com
Address: 142.250.183.196
Name:www.google.com
Address: 2404:6800:4009:826::2004
```

netstat -a

The 'netstat -a' command displays all active network connections and listening ports on a Linux system. It provides information about local and remote IP addresses, port numbers, and connection states, making it useful for network monitoring and troubleshooting. The '-a' option stands for "all" and shows both TCP and UDP connections along with listening ports.

```
practicalexampc29@LAB306PC32:~$ netstat -a
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 localhost:mysql         0.0.0.0:*               LISTEN
tcp        0      0 localhost:domain        0.0.0.0:*               LISTEN
tcp        0      0 LAB306PC32:domain       0.0.0.0:*               LISTEN
tcp        0      0 localhost:33060         0.0.0.0:*               LISTEN
tcp        0      0 localhost:ipp           0.0.0.0:*               LISTEN
tcp6       0      0 ip6-localhost:ipp       [::]:*                  LISTEN
udp        0      0 LAB306PC32:domain       0.0.0.0:*
udp        0      0 localhost:domain        0.0.0.0:*
udp        0      0 0.0.0.0:bootps          0.0.0.0:*
udp        0      0 LAB306PC32:bootpc       _gateway:bootps         ESTABLISHED
udp        0      0 0.0.0.0:631             0.0.0.0:*
udp        0      0 0.0.0.0:mdns            0.0.0.0:*
udp        0      0 0.0.0.0:43145           0.0.0.0:*
udp6       0      0 [::]:40047              [::]:*
udp6       0      0 [::]:mdns               [::]:*
raw6       0      0 [::]:ipv6-icmp          [::]:*                  7
Active UNIX domain sockets (servers and established)
Proto RefCnt Flags       Type       State         I-Node   Path
unix  2      [ ACC ]     STREAM     LISTENING     19696    /run/acpid.socket
unix  2      [ ACC ]     STREAM     LISTENING     19698    /run/avahi-daemon/socket
unix  2      [ ACC ]     STREAM     LISTENING     19702    /run/dbus/system_bus_socket
unix  2      [ ACC ]     STREAM     LISTENING     19704    /run/libvirt/libvirt-sock
unix  2      [ ACC ]     STREAM     LISTENING     19706    /run/uuidd/request
unix  2      [ ACC ]     STREAM     LISTENING     19708    /run/libvirt/virtlockd-sock
```

41

```
unix  2      [ ACC ]     STREAM     LISTENING     29645    /var/run/mysqld/mysqlx.sock
unix  2      [ ACC ]     STREAM     LISTENING     19710    /run/libvirt/virtlockd-admin-sock
unix  2      [ ACC ]     STREAM     LISTENING     19712    /run/libvirt/virtlogd-sock
unix  2      [ ACC ]     STREAM     LISTENING     29647    /var/run/mysqld/mysqld.sock
unix  2      [ ACC ]     STREAM     LISTENING     31767    /tmp/.X11-unix/X0
unix  2      [ ACC ]     STREAM     LISTENING     19714    /run/libvirt/virtlogd-admin-sock
unix  2      [ ACC ]     STREAM     LISTENING     19719    /run/libvirt/libvirt-admin-sock
unix  2      [ ACC ]     STREAM     LISTENING     19721    /run/libvirt/libvirt-sock-ro
unix  2      [ ACC ]     STREAM     LISTENING     21703
```

netstat -ap

The 'netstat -ap' command displays a list of all active network connections and listening ports on a Linux system, along with the associated process names. It provides information about local and remote IP addresses, port numbers, connection states, and the processes that are using those ports. This command is particularly helpful for identifying which processes are responsible for specific network connections or services running on the system. The '-a' option shows all connections, and the '-p' option displays the associated processes.

```
practicalexampc29@LAB306PC32:~$ netstat -ap
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program
name
tcp        0      0 localhost:mysql         0.0.0.0:*               LISTEN      -
tcp        0      0 localhost:domain        0.0.0.0:*               LISTEN      -
tcp        0      0 LAB306PC32:domain       0.0.0.0:*               LISTEN      -
tcp        0      0 localhost:33060         0.0.0.0:*               LISTEN      -
tcp        0      0 localhost:ipp           0.0.0.0:*               LISTEN      -
tcp6       0      0 ip6-localhost:ipp       [::]:*                  LISTEN      -
udp        0      0 LAB306PC32:domain       0.0.0.0:*                           -
udp        0      0 localhost:domain        0.0.0.0:*                           -
udp        0      0 0.0.0.0:bootps          0.0.0.0:*                           -
udp        0      0 LAB306PC32:bootpc       _gateway:bootps         ESTABLISHED -
udp        0      0 0.0.0.0:631             0.0.0.0:*                           -
udp        0      0 0.0.0.0:mdns            0.0.0.0:*                           -
udp        0      0 0.0.0.0:43145           0.0.0.0:*                           -
udp6       0      0 [::]:40047              [::]:*                              -
udp6       0      0 [::]:mdns               [::]:*                              -
raw6       0      0 [::]:ipv6-icmp          [::]:*                  7           -
Active UNIX domain sockets (servers and established)
Proto RefCnt Flags       Type       State         I-Node   PID/Program name     Path
unix  2      [ ACC ]     STREAM     LISTENING     19696    -                    /run/acpid.socket
unix  2      [ ACC ]     STREAM     LISTENING     19698    -
/run/avahi-daemon/socket
unix  2      [ ACC ]     STREAM     LISTENING     19702    -
/run/dbus/system_bus_socket
```

```
unix  2      [ ACC ]     STREAM     LISTENING     19704    -
/run/libvirt/libvirt-sock
unix  2      [ ACC ]     STREAM     LISTENING     19706    -
```

netstat -au

The 'netstat -au' command lists all active UDP (User Datagram Protocol) network connections on a Linux system. It displays information about local and remote IP addresses, port numbers, and connection states for UDP-based connections. This command is useful for monitoring and troubleshooting UDP-based network communication. The '-a' option shows all connections, and the '-u' option filters the output to display only UDP connections.

```
practicalexampc29@LAB306PC32:~$ netstat -au
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
udp        0      0 LAB306PC32:domain       0.0.0.0:*
udp        0      0 localhost:domain        0.0.0.0:*
udp        0      0 0.0.0.0:bootps          0.0.0.0:*
udp        0      0 LAB306PC32:bootpc       _gateway:bootps         ESTABLISHED
udp        0      0 0.0.0.0:631             0.0.0.0:*
udp        0      0 0.0.0.0:mdns            0.0.0.0:*
udp        0      0 0.0.0.0:43145           0.0.0.0:*
udp6       0      0 [::]:40047              [::]:*
udp6       0      0 [::]:mdns               [::]:*
```

netstat -tnl

The 'netstat -tnl' command displays a list of all listening TCP (Transmission Control Protocol) network ports on a Linux system. It provides information about local IP addresses and port numbers for services that are actively listening for incoming connections. This command is useful for identifying which network services are running and listening for incoming connections. The '-t' option filters the output to show only TCP connections, and the '-n' option displays numerical IP addresses and port numbers instead of resolving them to hostnames and service names.

```
practicalexampc29@LAB306PC32:~$ netstat -tnl
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.1:3306          0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN
tcp        0      0 192.168.122.1:53        0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:33060         0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN
tcp6       0      0 ::1:631                 :::*                    LISTEN
```

netstat -s

The 'netstat -s' command provides a summary of various network statistics on a Linux system. It displays cumulative statistics for different network protocols, including TCP, UDP, ICMP, and others. This command is valuable for monitoring network performance

and diagnosing network-related issues. It presents detailed information about network errors, packet types, and various protocol-specific statistics, allowing administrators to gain insights into the network's health and performance.

```
practicalexampc29@LAB306PC32:~$ netstat -s
Ip:
    Forwarding: 1
    6684 total packets received
    4 with invalid addresses
    0 forwarded
    0 incoming packets discarded
    6160 incoming packets delivered
    3576 requests sent out
    20 outgoing packets dropped
Icmp:
    403 ICMP messages received
    233 input ICMP message failed
    ICMP input histogram:
        destination unreachable: 68
        timeout in transit: 37
        echo requests: 233
        echo replies: 65
    230 ICMP messages sent
    0 ICMP messages failed
    ICMP output histogram:
        destination unreachable: 48
        echo requests: 182
IcmpMsg:
        InType0: 65
        InType3: 68
        InType8: 233
        InType11: 37
        OutType3: 48
        OutType8: 182
Tcp:
    28 active connection openings
    0 passive connection openings
    6 failed connection attempts
    0 connection resets received
    0 connections established
    4086 segments received
    2745 segments sent out
    18 segments retransmitted
    0 bad segments received
    3 resets sent
Udp:
    788 packets received
    48 packets to unknown port received
    0 packet receive errors
    638 packets sent
```

```
    0 receive buffer errors
    0 send buffer errors
    IgnoredMulti: 811
UdpLite:
TcpExt:
    12 TCP sockets finished time wait in fast timer
    3 delayed acks sent
    Quick ack mode was activated 1 times
    3805 packet headers predicted
    49 acknowledgments not containing data payload received
    8 predicted acknowledgments
    TCPLostRetransmit: 14
    TCPTimeouts: 18
    TCPLossProbes: 2
    TCPDSACKOldSent: 1
    2 connections aborted due to timeout
    TCPRcvCoalesce: 39
    TCPOFOQueue: 13
    TCPAutoCorking: 1
    TCPOrigDataSent: 90
    TCPDelivered: 110
    TCPAckCompressed: 6
    TcpTimeoutRehash: 16
IpExt:
    InNoRoutes: 2
    InMcastPkts: 342
    OutMcastPkts: 95
    InBcastPkts: 1044
    InOctets: 10322907
    OutOctets: 252812
    InMcastOctets: 87492
    OutMcastOctets: 19560
    InBcastOctets: 111165
    InNoECTPkts: 9609
    InECT1Pkts: 3
    InECT0Pkts: 3
MPTcpExt:
```

netstat -rn

The 'netstat -rn' command displays the routing table on a Linux system. It shows the routing information, including the destination network or host, gateway (next hop), and the network interface through which traffic is routed. This command is essential for viewing the current network routing configuration, helping administrators understand how network packets are directed within the system and where they will be forwarded.

```
practicalexampc29@LAB306PC32:~$ netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
0.0.0.0         192.168.31.1    0.0.0.0         UG        0 0          0 enp1s0
```

```
169.254.0.0     0.0.0.0         255.255.0.0     U       0 0        0 virbr0
192.168.31.0    0.0.0.0         255.255.255.0   U       0 0        0 enp1s0
192.168.122.0   0.0.0.0         255.255.255.0   U       0 0        0 virbr0
```

netstat -i

The 'netstat -i' command provides a listing of network interfaces on a Linux system along with various statistics associated with each interface. It displays information such as the interface name, packets transmitted and received, errors, drops, and more. This command is useful for monitoring network interface activity, identifying potential network issues, and assessing network performance at a per-interface level.

```
practicalexampc29@LAB306PC32:~$ netstat -i
Kernel Interface table
Iface      MTU    RX-OK RX-ERR RX-DRP RX-OVR    TX-OK TX-ERR TX-DRP TX-OVR
Flg
enp1s0    1500    10774      0      8 0          3304      0      0      0 BMRU
lo       65536      425      0      0 0           425      0      0      0 LRU
virbr0    1500        0      0      0 0             0      0      0      0 BMU
```

route -n

The 'route -n' command displays the routing table in a concise numeric format on a Linux system. It provides information about the network routes, including destination networks or hosts, gateway addresses, network masks, and interface names, all displayed in numerical form (IP addresses and numbers). This command is used to view the routing configuration and can be helpful for understanding how network traffic is routed within the system without hostname resolution.

```
practicalexampc29@LAB306PC32:~$ route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         192.168.31.1    0.0.0.0         UG    100    0        0 enp1s0
169.254.0.0     0.0.0.0         255.255.0.0     U     1000   0        0 virbr0
192.168.31.0    0.0.0.0         255.255.255.0   U     100    0        0 enp1s0
192.168.122.0   0.0.0.0         255.255.255.0   U     0      0        0 virbr0
```

route add default gw

The 'route add default gw' command is used to manually add a default gateway on a Linux system. The "default gateway" is the router or gateway that your Linux system uses to forward network traffic when the destination is outside of your local network. It's the route taken for all non-local traffic. The 'route add' part of the command is used to add a new route to the routing table. The 'default' specifies that this route is the default route, used when no other specific route matches the destination. 'gw' Indicates that you're specifying a gateway address.

```
practicalexampc29@LAB306PC32:~$ sudo su root
root@LAB306PC32:/home/practicalexampc29# route add default gw 192.168.31.253
root@LAB306PC32:/home/practicalexampc29# route -n
Kernel IP routing table
```

```
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         192.168.31.253  0.0.0.0         UG    0      0        0 enp1s0
0.0.0.0         192.168.31.1    0.0.0.0         UG    100    0        0 enp1s0
169.254.0.0     0.0.0.0         255.255.0.0     U     1000   0        0 virbr0
192.168.31.0    0.0.0.0         255.255.255.0   U     100    0        0 enp1s0
192.168.122.0   0.0.0.0         255.255.255.0   U     0      0        0 virbr0
```

route add -net

The 'route add -net' command is used to add a specific network route in a Linux system's routing table. It signals the addition of a network route and is followed by the destination network that you want to reach through this route, the network mask for the destination network, and the gateway (next hop) for reaching the specified destination network. Lastly, we specify the network interface (eth0) through which this route should be applied.

```
root@LAB306PC32:/home/practicalexampc29# route add -net 192.168.31.0 netmask
255.255.255.0 gw 192.168.31.253 eth 0
Usage: inet_route [-vF] del {-host|-net} Target[/prefix] [gw Gw] [metric M] [[dev] If]
       inet_route [-vF] add {-host|-net} Target[/prefix] [gw Gw] [metric M]
                         [netmask N] [mss Mss] [window W] [irtt I]
                         [mod] [dyn] [reinstate] [[dev] If]
       inet_route [-vF] add {-host|-net} Target[/prefix] [metric M] reject
       inet_route [-FC] flush      NOT supported
```

arp -a

The 'arp -a' command is used to display the ARP (Address Resolution Protocol) cache table on a Linux system. The ARP cache is a table that stores mappings between IP addresses and MAC (Media Access Control) addresses on your local network. It helps your system quickly resolve IP addresses to MAC addresses, a process necessary for proper network communication. When you run this command, it shows a list of entries in the ARP cache, including the IP addresses and corresponding MAC addresses of devices that your system has recently communicated with. This information is used for efficient data transfer within your local network.

```
root@LAB306PC32:/home/practicalexampc29# arp -a
_gateway (192.168.31.253) at <incomplete> on enp1s0
? (192.168.31.6) at d0:67:e5:1a:23:05 [ether] on enp1s0
? (192.168.31.27) at a4:ae:12:84:80:e1 [ether] on enp1s0
_gateway (192.168.31.1) at 9c:53:22:05:6a:19 [ether] on enp1s0
? (192.168.31.48) at a4:ae:12:84:81:df [ether] on enp1s0
```

156

18/10/23

Nayati Savant
C31 - TE
2103156

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

Experiment - 09

Aim - Use Wire shark to understand operation of TCP/IP layers - ethernet layer, data link layer, network layer, transport layer, Application layers.

Theory -

TCP is one of the most important protocol /standard to enable communication possible amongst devices present over a particular network. It has algorithms that solve complex errors arising in packet communications, ie corrupted packet, invalid packets, duplicates, etc. In order to start a communication, TCP first establishes a connection using three-way-handshake. &
Wireshark is the most widely used protocol analyzer. TO launch Wireshark the following steps are followed.

1) From menu, select capture → options → interfaces
2) In the interfaces choose a Ethernet (note IP) and click start.
3) Now we shall be capturing packets. Browse to a particular web address to generate traffic. Go to any website and return to Wireshark and stop the capture by selecting stop.
4) Now. captured packet list is on the screen we filter out TCP packets only by typing 'TCP' in the display bar filter and apply it

Niyati Savant
C31 – TE
2103156

THADOMAL SHAHANI
TSEC
ENGINEERING COLLEGE

5) Here we have list of TCP packets. The first 3 packets are part of 3-way handshake mechanism. The 3 steps of this mechanism are observed in these 3 packets where the packet types i.e. ACK, SYN, SYN-ACK is listed on their respective side.

6) To closely examine a packet, we select a packet and in the expert view in packet details section where we can see the TCP parameters like source port, destination port, TCP segment length, sequence number, next sequence number, acknowledgment number, header length

7) A major section of this packet is the flag section of a packet that gives in-depth information like CWR, ECN-echo, Urgent, Acknowledgement, Push, Reset, Syn, Fin.

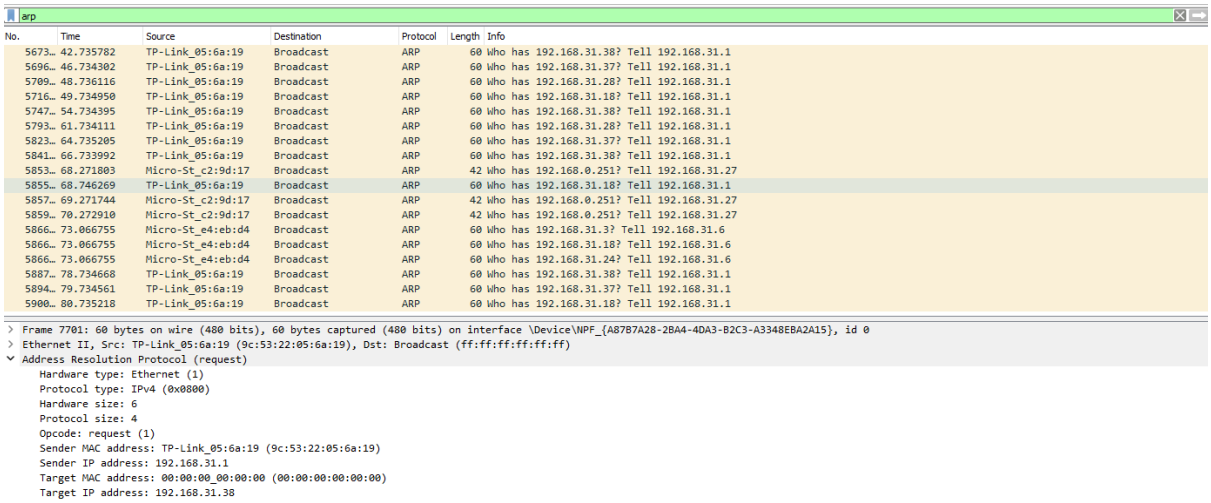
8) In the further subsections are have window size value, Checksum, checksum status

9) In the end we follow TCP termination handshake to close the connection

Approach
16/10/23

15/10

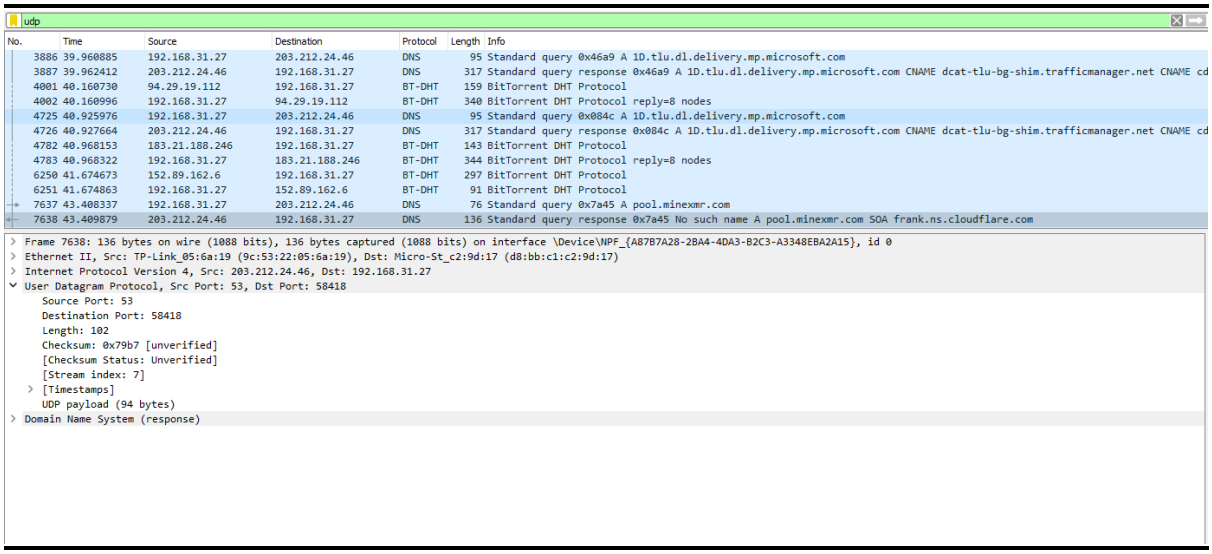
Wire Shark to understand operation of TCP/IP



arp

No.	Time	Source	Destination	Protocol	Length	Info
5673…	42.735782	TP-Link_05:6a:19	Broadcast	ARP	60	Who has 192.168.31.38? Tell 192.168.31.1
5696…	46.734302	TP-Link_05:6a:19	Broadcast	ARP	60	Who has 192.168.31.37? Tell 192.168.31.1
5709…	48.736116	TP-Link_05:6a:19	Broadcast	ARP	60	Who has 192.168.31.28? Tell 192.168.31.1
5716…	49.734950	TP-Link_05:6a:19	Broadcast	ARP	60	Who has 192.168.31.18? Tell 192.168.31.1
5747…	54.734395	TP-Link_05:6a:19	Broadcast	ARP	60	Who has 192.168.31.38? Tell 192.168.31.1
5793…	61.734111	TP-Link_05:6a:19	Broadcast	ARP	60	Who has 192.168.31.28? Tell 192.168.31.1
5823…	64.735205	TP-Link_05:6a:19	Broadcast	ARP	60	Who has 192.168.31.37? Tell 192.168.31.1
5841…	66.733992	TP-Link_05:6a:19	Broadcast	ARP	60	Who has 192.168.31.38? Tell 192.168.31.1
5853…	68.271803	Micro-St_c2:9d:17	Broadcast	ARP	42	Who has 192.168.0.251? Tell 192.168.31.27
5855…	68.746269	TP-Link_05:6a:19	Broadcast	ARP	60	Who has 192.168.31.18? Tell 192.168.31.1
5857…	69.271744	Micro-St_c2:9d:17	Broadcast	ARP	42	Who has 192.168.0.251? Tell 192.168.31.27
5859…	70.272910	Micro-St_c2:9d:17	Broadcast	ARP	42	Who has 192.168.0.251? Tell 192.168.31.27
5866…	73.066755	Micro-St_e4:eb:d4	Broadcast	ARP	60	Who has 192.168.31.3? Tell 192.168.31.6
5866…	73.066755	Micro-St_e4:eb:d4	Broadcast	ARP	60	Who has 192.168.31.18? Tell 192.168.31.6
5866…	73.066755	Micro-St_e4:eb:d4	Broadcast	ARP	60	Who has 192.168.31.24? Tell 192.168.31.6
5887…	78.734668	TP-Link_05:6a:19	Broadcast	ARP	60	Who has 192.168.31.38? Tell 192.168.31.1
5894…	79.734561	TP-Link_05:6a:19	Broadcast	ARP	60	Who has 192.168.31.37? Tell 192.168.31.1
5900…	80.735218	TP-Link_05:6a:19	Broadcast	ARP	60	Who has 192.168.31.18? Tell 192.168.31.1

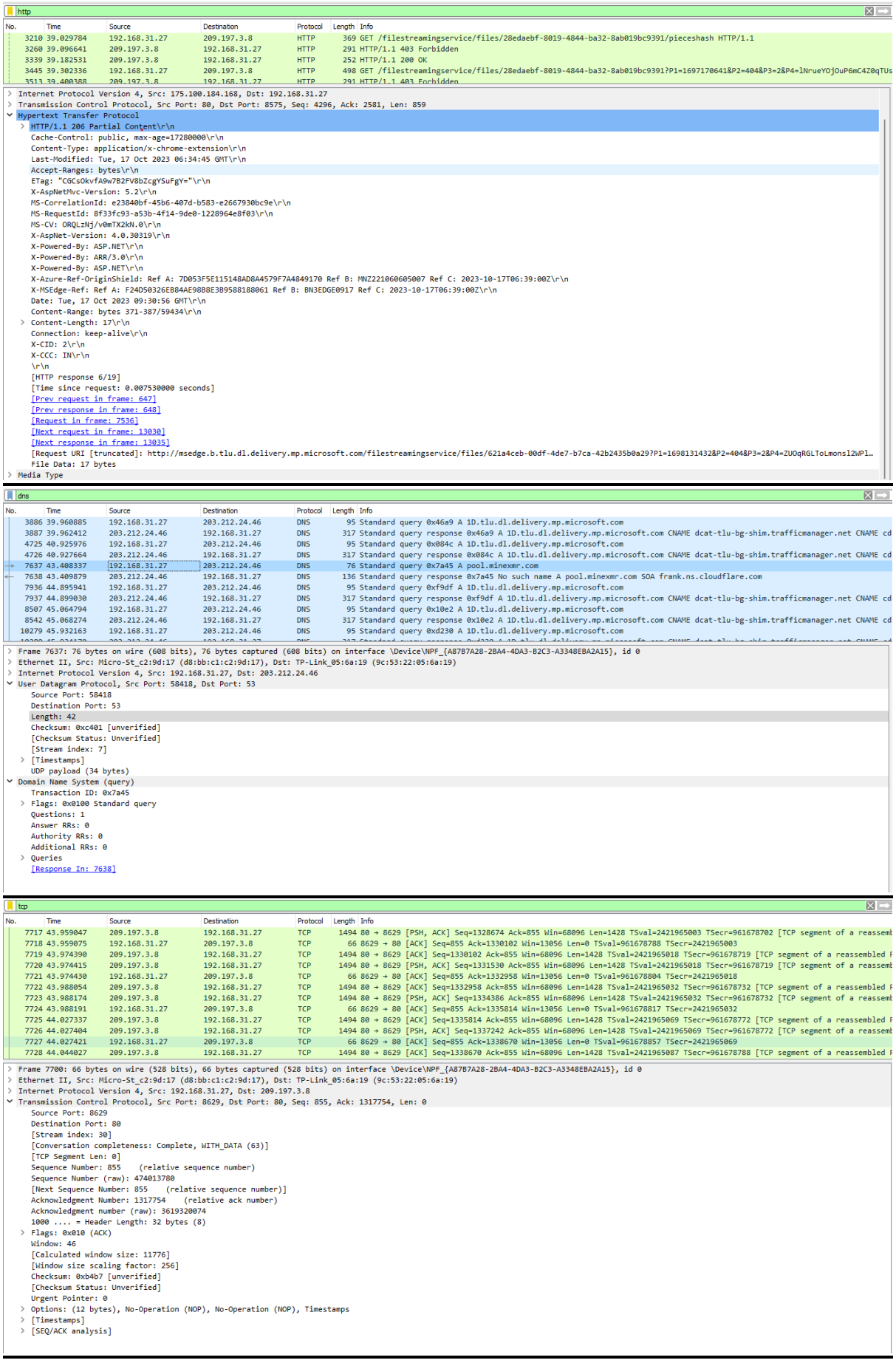
```
> Frame 7701: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface \Device\NPF_{A87B7A28-2BA4-4DA3-B2C3-A3348EBA2A15}, id 0
> Ethernet II, Src: TP-Link_05:6a:19 (9c:53:22:05:6a:19), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
v Address Resolution Protocol (request)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    Sender MAC address: TP-Link_05:6a:19 (9c:53:22:05:6a:19)
    Sender IP address: 192.168.31.1
    Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
    Target IP address: 192.168.31.38
```

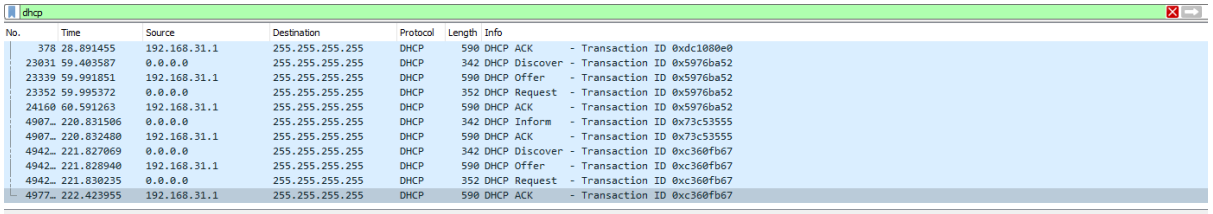


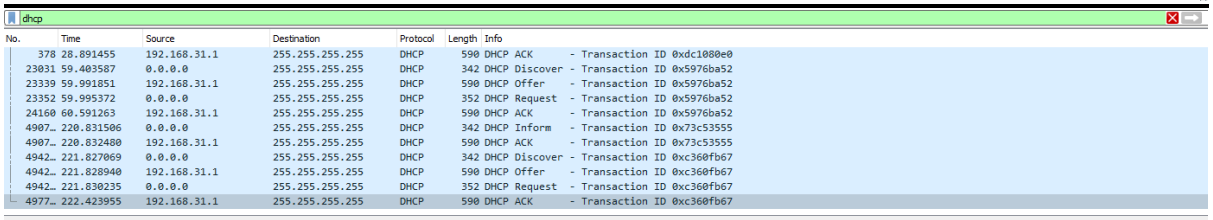
udp

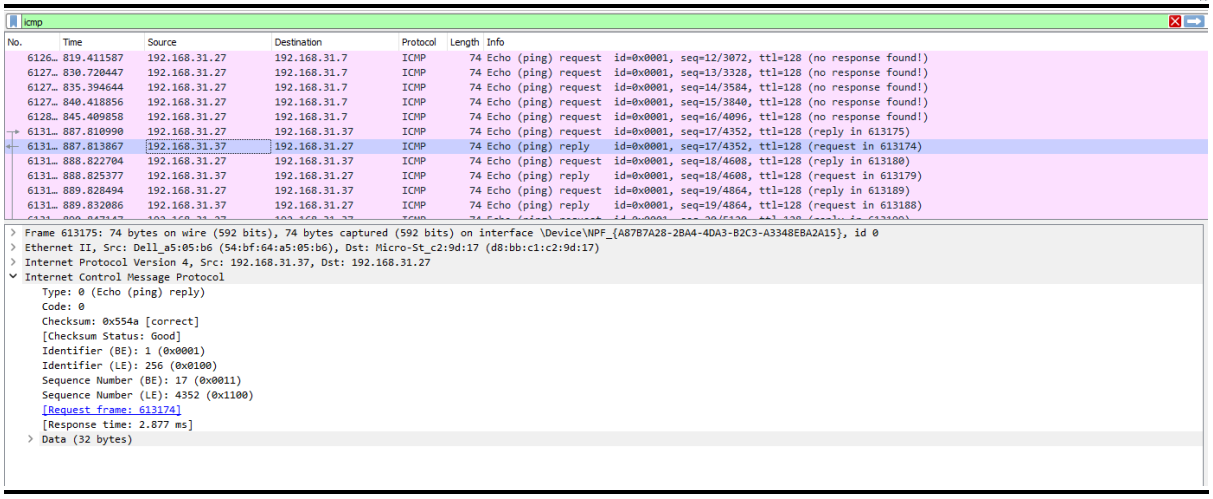
No.	Time	Source	Destination	Protocol	Length	Info
3886	39.960885	192.168.31.27	203.212.24.46	DNS	95	Standard query 0x46a9 A 1D.tlu.dl.delivery.mp.microsoft.com
3887	39.962412	203.212.24.46	192.168.31.27	DNS	317	Standard query response 0x46a9 A 1D.tlu.dl.delivery.mp.microsoft.com CNAME dcat-tlu-bg-shim.trafficmanager.net CNAME cd
4001	40.160730	94.29.19.112	192.168.31.27	BT-DHT	159	BitTorrent DHT Protocol
4002	40.160996	192.168.31.27	94.29.19.112	BT-DHT	340	BitTorrent DHT Protocol reply=8 nodes
4725	40.925976	192.168.31.27	203.212.24.46	DNS	95	Standard query 0x084c A 1D.tlu.dl.delivery.mp.microsoft.com
4726	40.927664	203.212.24.46	192.168.31.27	DNS	317	Standard query response 0x084c A 1D.tlu.dl.delivery.mp.microsoft.com CNAME dcat-tlu-bg-shim.trafficmanager.net CNAME cd
4782	40.968153	183.21.188.246	192.168.31.27	BT-DHT	143	BitTorrent DHT Protocol
4783	40.968322	192.168.31.27	183.21.188.246	BT-DHT	344	BitTorrent DHT Protocol reply=8 nodes
6250	41.674673	152.89.162.6	192.168.31.27	BT-DHT	297	BitTorrent DHT Protocol
6251	41.674863	192.168.31.27	152.89.162.6	BT-DHT	91	BitTorrent DHT Protocol
7637	43.408337	192.168.31.27	203.212.24.46	DNS	76	Standard query 0x7a45 A pool.minexmr.com
7638	43.409879	203.212.24.46	192.168.31.27	DNS	136	Standard query response 0x7a45 No such name A pool.minexmr.com SOA frank.ns.cloudflare.com

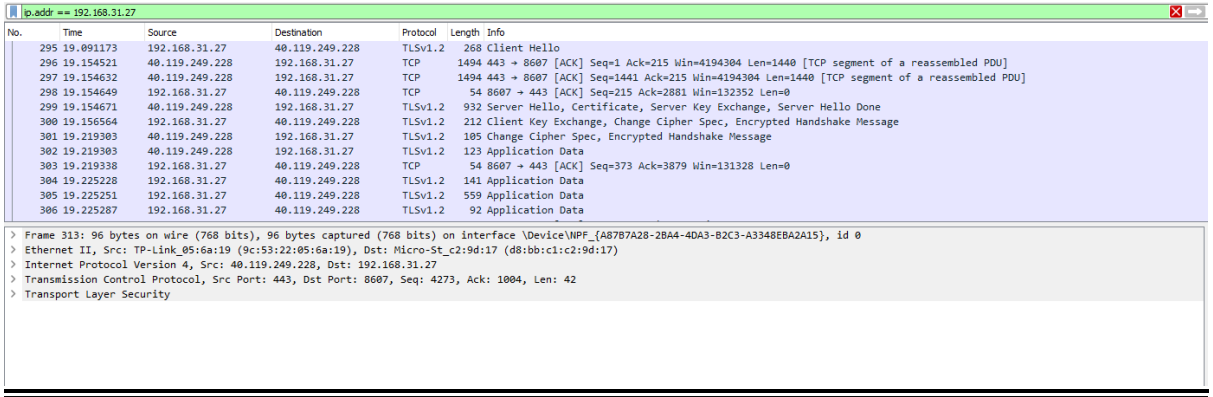
```
> Frame 7638: 136 bytes on wire (1088 bits), 136 bytes captured (1088 bits) on interface \Device\NPF_{A87B7A28-2BA4-4DA3-B2C3-A3348EBA2A15}, id 0
> Ethernet II, Src: TP-Link_05:6a:19 (9c:53:22:05:6a:19), Dst: Micro-St_c2:9d:17 (d8:bb:c1:c2:9d:17)
> Internet Protocol Version 4, Src: 203.212.24.46, Dst: 192.168.31.27
v User Datagram Protocol, Src Port: 53, Dst Port: 58418
    Source Port: 53
    Destination Port: 58418
    Length: 102
    Checksum: 0x79b7 [unverified]
    [Checksum Status: Unverified]
    [Stream index: 7]
  > [Timestamps]
    UDP payload (94 bytes)
> Domain Name System (response)
```


http
No. Time Source Destination Protocol Length Info
3210 39.029784 192.168.31.27 209.197.3.8 HTTP 369 GET /filestreamingservice/files/28edaebf-8019-4844-ba32-8ab019bc9391/pieceshash HTTP/1.1
3260 39.096641 209.197.3.8 192.168.31.27 HTTP 291 HTTP/1.1 403 Forbidden
3339 39.182531 209.197.3.8 192.168.31.27 HTTP 252 HTTP/1.1 200 OK
3445 39.302336 192.168.31.27 209.197.3.8 HTTP 498 GET /filestreamingservice/files/28edaebf-8019-4844-ba32-8ab019bc9391?P1=1697170641&P2=404&P3=2&P4=lNrueYOjOuP6mC4Z0qTUs
3513 39.400388 209.197.3.8 192.168.31.27 HTTP 291 HTTP/1.1 403 Forbidden
Internet Protocol Version 4, Src: 175.100.184.168, Dst: 192.168.31.27
Transmission Control Protocol, Src Port: 80, Dst Port: 8575, Seq: 4296, Ack: 2581, Len: 859
Hypertext Transfer Protocol
HTTP/1.1 206 Partial Content\r\n
Cache-Control: public, max-age=17280000\r\n
Content-Type: application/x-chrome-extension\r\n
Last-Modified: Tue, 17 Oct 2023 06:34:45 GMT\r\n
Accept-Ranges: bytes\r\n
ETag: "CGCsOkvfA9w7B2FV8bZcgYSuFgY="\r\n
X-AspNetMvc-Version: 5.2\r\n
MS-CorrelationId: e23840bf-45b6-407d-b583-e2667930bc9e\r\n
MS-RequestId: 8f33fc93-a53b-4f14-9de0-1228964e8f03\r\n
MS-CV: ORQLzNj/v0mTX2kN.0\r\n
X-AspNet-Version: 4.0.30319\r\n
X-Powered-By: ASP.NET\r\n
X-Powered-By: ARR/3.0\r\n
X-Powered-By: ASP.NET\r\n
X-Azure-Ref-OriginShield: Ref A: 7D053F5E115148AD8A4579F7A4849170 Ref B: MNZ221060605007 Ref C: 2023-10-17T06:39:00Z\r\n
X-MSEdge-Ref: Ref A: F24D50326EB84AE98B8E3B9588188061 Ref B: BN3EDGE0917 Ref C: 2023-10-17T06:39:00Z\r\n
Date: Tue, 17 Oct 2023 09:30:56 GMT\r\n
Content-Range: bytes 371-387/59434\r\n
Content-Length: 17\r\n
Connection: keep-alive\r\n
X-CID: 2\r\n
X-CCC: IN\r\n
\r\n
[HTTP response 6/19]
[Time since request: 0.007530000 seconds]
[Prev request in frame: 647]
[Prev response in frame: 648]
[Request in frame: 7536]
[Next request in frame: 13030]
[Next response in frame: 13035]
[Request URI [truncated]: http://msedge.b.tlu.dl.delivery.mp.microsoft.com/filestreamingservice/files/621a4ceb-00df-4de7-b7ca-42b2435b0a29?P1=1698131432&P2=404&P3=2&P4=ZUOqRGLToLmonsl2WPl…
File Data: 17 bytes
Media Type
dns
No. Time Source Destination Protocol Length Info
3886 39.960885 192.168.31.27 203.212.24.46 DNS 95 Standard query 0x46a9 A 1D.tlu.dl.delivery.mp.microsoft.com
3887 39.962412 203.212.24.46 192.168.31.27 DNS 317 Standard query response 0x46a9 A 1D.tlu.dl.delivery.mp.microsoft.com CNAME dcat-tlu-bg-shim.trafficmanager.net CNAME cd
4725 40.925976 192.168.31.27 203.212.24.46 DNS 95 Standard query 0x084c A 1D.tlu.dl.delivery.mp.microsoft.com
4726 40.927664 203.212.24.46 192.168.31.27 DNS 317 Standard query response 0x084c A 1D.tlu.dl.delivery.mp.microsoft.com CNAME dcat-tlu-bg-shim.trafficmanager.net CNAME cd
7637 43.408337 192.168.31.27 203.212.24.46 DNS 76 Standard query 0x7a45 A pool.minexmr.com
7638 43.409879 203.212.24.46 192.168.31.27 DNS 136 Standard query response 0x7a45 No such name A pool.minexmr.com SOA frank.ns.cloudflare.com
7936 44.895941 192.168.31.27 203.212.24.46 DNS 95 Standard query 0xf9df A 1D.tlu.dl.delivery.mp.microsoft.com
7937 44.899030 203.212.24.46 192.168.31.27 DNS 317 Standard query response 0xf9df A 1D.tlu.dl.delivery.mp.microsoft.com CNAME dcat-tlu-bg-shim.trafficmanager.net CNAME cd
8507 45.064794 192.168.31.27 203.212.24.46 DNS 95 Standard query 0x10e2 A 1D.tlu.dl.delivery.mp.microsoft.com
8542 45.068274 203.212.24.46 192.168.31.27 DNS 317 Standard query response 0x10e2 A 1D.tlu.dl.delivery.mp.microsoft.com CNAME dcat-tlu-bg-shim.trafficmanager.net CNAME cd
10279 45.932163 192.168.31.27 203.212.24.46 DNS 95 Standard query 0xd230 A 1D.tlu.dl.delivery.mp.microsoft.com
Frame 7637: 76 bytes on wire (608 bits), 76 bytes captured (608 bits) on interface \Device\NPF_{A87B7A28-2BA4-4DA3-B2C3-A3348EBA2A15}, id 0
Ethernet II, Src: Micro-St_c2:9d:17 (d8:bb:c1:c2:9d:17), Dst: TP-Link_05:6a:19 (9c:53:22:05:6a:19)
Internet Protocol Version 4, Src: 192.168.31.27, Dst: 203.212.24.46
User Datagram Protocol, Src Port: 58418, Dst Port: 53
Source Port: 58418
Destination Port: 53
Length: 42
Checksum: 0xc401 [unverified]
[Checksum Status: Unverified]
[Stream index: 7]
[Timestamps]
UDP payload (34 bytes)
Domain Name System (query)
Transaction ID: 0x7a45
Flags: 0x0100 Standard query
Questions: 1
Answer RRs: 0
Authority RRs: 0
Additional RRs: 0
Queries
[Response In: 7638]
tcp
No. Time Source Destination Protocol Length Info
7717 43.959047 209.197.3.8 192.168.31.27 TCP 1494 80 → 8629 [PSH, ACK] Seq=1328674 Ack=855 Win=68096 Len=1428 TSval=2421965003 TSecr=961678702 [TCP segment of a reassemb
7718 43.959075 192.168.31.27 209.197.3.8 TCP 66 8629 → 80 [ACK] Seq=855 Ack=1330102 Win=13056 Len=0 TSval=961678788 TSecr=2421965003
7719 43.974390 209.197.3.8 192.168.31.27 TCP 1494 80 → 8629 [ACK] Seq=1330102 Ack=855 Win=68096 Len=1428 TSval=2421965018 TSecr=961678719 [TCP segment of a reassembled P
7720 43.974415 209.197.3.8 192.168.31.27 TCP 1494 80 → 8629 [PSH, ACK] Seq=1331530 Ack=855 Win=68096 Len=1428 TSval=2421965018 TSecr=961678719 [TCP segment of a reassemb
7721 43.974430 192.168.31.27 209.197.3.8 TCP 66 8629 → 80 [ACK] Seq=855 Ack=1332958 Win=13056 Len=0 TSval=961678804 TSecr=2421965018
7722 43.988054 209.197.3.8 192.168.31.27 TCP 1494 80 → 8629 [ACK] Seq=1332958 Ack=855 Win=68096 Len=1428 TSval=2421965032 TSecr=961678732 [TCP segment of a reassembled P
7723 43.988174 209.197.3.8 192.168.31.27 TCP 1494 80 → 8629 [PSH, ACK] Seq=1334386 Ack=855 Win=68096 Len=1428 TSval=2421965032 TSecr=961678732 [TCP segment of a reassemb
7724 43.988191 192.168.31.27 209.197.3.8 TCP 66 8629 → 80 [ACK] Seq=855 Ack=1335814 Win=13056 Len=0 TSval=961678817 TSecr=2421965032
7725 44.027337 209.197.3.8 192.168.31.27 TCP 1494 80 → 8629 [ACK] Seq=1335814 Ack=855 Win=68096 Len=1428 TSval=2421965069 TSecr=961678772 [TCP segment of a reassembled P
7726 44.027404 209.197.3.8 192.168.31.27 TCP 1494 80 → 8629 [PSH, ACK] Seq=1337242 Ack=855 Win=68096 Len=1428 TSval=2421965069 TSecr=961678772 [TCP segment of a reassemb
7727 44.027421 192.168.31.27 209.197.3.8 TCP 66 8629 → 80 [ACK] Seq=855 Ack=1338670 Win=13056 Len=0 TSval=961678857 TSecr=2421965069
7728 44.044027 209.197.3.8 192.168.31.27 TCP 1494 80 → 8629 [ACK] Seq=1338670 Ack=855 Win=68096 Len=1428 TSval=2421965087 TSecr=961678788 [TCP segment of a reassembled P
Frame 7700: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface \Device\NPF_{A87B7A28-2BA4-4DA3-B2C3-A3348EBA2A15}, id 0
Ethernet II, Src: Micro-St_c2:9d:17 (d8:bb:c1:c2:9d:17), Dst: TP-Link_05:6a:19 (9c:53:22:05:6a:19)
Internet Protocol Version 4, Src: 192.168.31.27, Dst: 209.197.3.8
Transmission Control Protocol, Src Port: 8629, Dst Port: 80, Seq: 855, Ack: 1317754, Len: 0
Source Port: 8629
Destination Port: 80
[Stream index: 30]
[Conversation completeness: Complete, WITH_DATA (63)]
[TCP Segment Len: 0]
Sequence Number: 855 (relative sequence number)
Sequence Number (raw): 474013780
[Next Sequence Number: 855 (relative sequence number)]
Acknowledgment Number: 1317754 (relative ack number)
Acknowledgment number (raw): 3619320074
1000 .... = Header Length: 32 bytes (8)
Flags: 0x010 (ACK)
Window: 46
[Calculated window size: 11776]
[Window size scaling factor: 256]
Checksum: 0xb4b7 [unverified]
[Checksum Status: Unverified]
Urgent Pointer: 0
Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
[Timestamps]
[SEQ/ACK analysis]


dhcp
No. Time Source Destination Protocol Length Info
378 28.891455 192.168.31.1 255.255.255.255 DHCP 590 DHCP ACK - Transaction ID 0xdc1080e0
23031 59.403587 0.0.0.0 255.255.255.255 DHCP 342 DHCP Discover - Transaction ID 0x5976ba52
23339 59.991851 192.168.31.1 255.255.255.255 DHCP 590 DHCP Offer - Transaction ID 0x5976ba52
23352 59.995372 0.0.0.0 255.255.255.255 DHCP 352 DHCP Request - Transaction ID 0x5976ba52
24160 60.591263 192.168.31.1 255.255.255.255 DHCP 590 DHCP ACK - Transaction ID 0x5976ba52
4907… 220.831506 0.0.0.0 255.255.255.255 DHCP 342 DHCP Inform - Transaction ID 0x73c53555
4907… 220.832480 192.168.31.1 255.255.255.255 DHCP 590 DHCP ACK - Transaction ID 0x73c53555
4942… 221.827069 0.0.0.0 255.255.255.255 DHCP 342 DHCP Discover - Transaction ID 0xc360fb67
4942… 221.828940 192.168.31.1 255.255.255.255 DHCP 590 DHCP Offer - Transaction ID 0xc360fb67
4942… 221.830235 0.0.0.0 255.255.255.255 DHCP 352 DHCP Request - Transaction ID 0xc360fb67
4977… 222.423955 192.168.31.1 255.255.255.255 DHCP 590 DHCP ACK - Transaction ID 0xc360fb67
Frame 497788: 590 bytes on wire (4720 bits), 590 bytes captured (4720 bits) on interface \Device\NPF_{A87B7A28-2BA4-4DA3-B2C3-A3348EBA2A15}, id 0
Ethernet II, Src: TP-Link_05:6a:19 (9c:53:22:05:6a:19), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
Internet Protocol Version 4, Src: 192.168.31.1, Dst: 255.255.255.255
User Datagram Protocol, Src Port: 67, Dst Port: 68
Dynamic Host Configuration Protocol (ACK)
Message type: Boot Reply (2)
Hardware type: Ethernet (0x01)
Hardware address length: 6
Hops: 0
Transaction ID: 0xc360fb67
Seconds elapsed: 0
Bootp flags: 0x8000, Broadcast flag (Broadcast)
Client IP address: 0.0.0.0
Your (client) IP address: 192.168.31.12
Next server IP address: 0.0.0.0
Relay agent IP address: 0.0.0.0
Client MAC address: Micro-St_c2:a0:2e (d8:bb:c1:c2:a0:2e)
Client hardware address padding: 00000000000000000000
Server host name not given
Boot file name not given
Magic cookie: DHCP
Option: (53) DHCP Message Type (ACK)
Option: (54) DHCP Server Identifier (192.168.31.1)
Option: (51) IP Address Lease Time
Option: (6) Domain Name Server
Option: (1) Subnet Mask (255.255.255.0)
Option: (3) Router
Option: (255) End
Padding: 0000000000000000000000000000000000000000000000000000000000000000…


dhcp
No. Time Source Destination Protocol Length Info
378 28.891455 192.168.31.1 255.255.255.255 DHCP 590 DHCP ACK - Transaction ID 0xdc1080e0
23031 59.403587 0.0.0.0 255.255.255.255 DHCP 342 DHCP Discover - Transaction ID 0x5976ba52
23339 59.991851 192.168.31.1 255.255.255.255 DHCP 590 DHCP Offer - Transaction ID 0x5976ba52
23352 59.995372 0.0.0.0 255.255.255.255 DHCP 352 DHCP Request - Transaction ID 0x5976ba52
24160 60.591263 192.168.31.1 255.255.255.255 DHCP 590 DHCP ACK - Transaction ID 0x5976ba52
4907… 220.831506 0.0.0.0 255.255.255.255 DHCP 342 DHCP Inform - Transaction ID 0x73c53555
4907… 220.832480 192.168.31.1 255.255.255.255 DHCP 590 DHCP ACK - Transaction ID 0x73c53555
4942… 221.827069 0.0.0.0 255.255.255.255 DHCP 342 DHCP Discover - Transaction ID 0xc360fb67
4942… 221.828940 192.168.31.1 255.255.255.255 DHCP 590 DHCP Offer - Transaction ID 0xc360fb67
4942… 221.830235 0.0.0.0 255.255.255.255 DHCP 352 DHCP Request - Transaction ID 0xc360fb67
4977… 222.423955 192.168.31.1 255.255.255.255 DHCP 590 DHCP ACK - Transaction ID 0xc360fb67
Frame 497788: 590 bytes on wire (4720 bits), 590 bytes captured (4720 bits) on interface \Device\NPF_{A87B7A28-2BA4-4DA3-B2C3-A3348EBA2A15}, id 0
Ethernet II, Src: TP-Link_05:6a:19 (9c:53:22:05:6a:19), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
Internet Protocol Version 4, Src: 192.168.31.1, Dst: 255.255.255.255
User Datagram Protocol, Src Port: 67, Dst Port: 68
Dynamic Host Configuration Protocol (ACK)
Message type: Boot Reply (2)
Hardware type: Ethernet (0x01)
Hardware address length: 6
Hops: 0
Transaction ID: 0xc360fb67
Seconds elapsed: 0
Bootp flags: 0x8000, Broadcast flag (Broadcast)
Client IP address: 0.0.0.0
Your (client) IP address: 192.168.31.12
Next server IP address: 0.0.0.0
Relay agent IP address: 0.0.0.0
Client MAC address: Micro-St_c2:a0:2e (d8:bb:c1:c2:a0:2e)
Client hardware address padding: 00000000000000000000
Server host name not given
Boot file name not given
Magic cookie: DHCP
Option: (53) DHCP Message Type (ACK)
Option: (54) DHCP Server Identifier (192.168.31.1)
Option: (51) IP Address Lease Time
Option: (6) Domain Name Server
Option: (1) Subnet Mask (255.255.255.0)
Option: (3) Router
Option: (255) End
Padding: 0000000000000000000000000000000000000000000000000000000000000000…

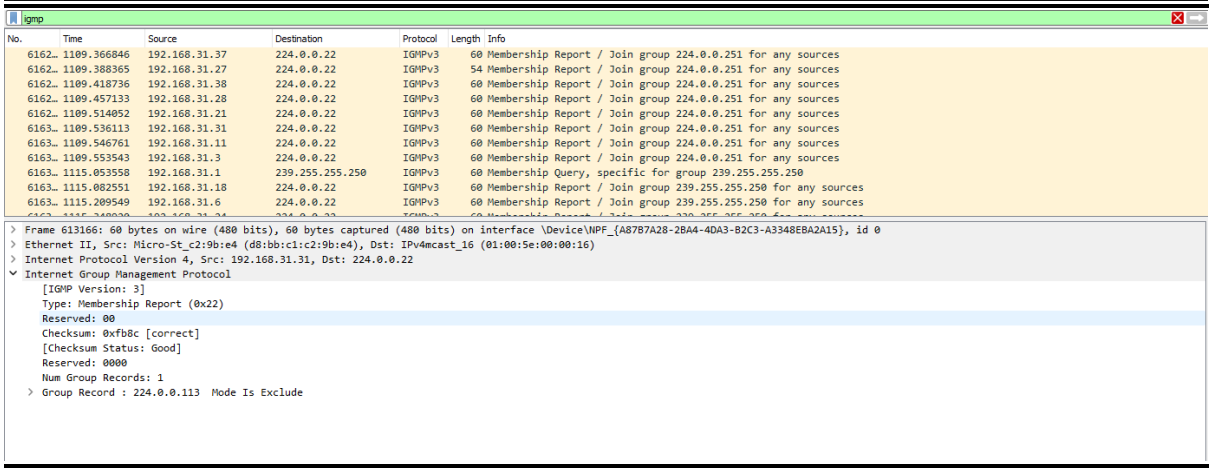

icmp
No. Time Source Destination Protocol Length Info
6126… 819.411587 192.168.31.27 192.168.31.7 ICMP 74 Echo (ping) request id=0x0001, seq=12/3072, ttl=128 (no response found!)
6127… 830.720447 192.168.31.27 192.168.31.7 ICMP 74 Echo (ping) request id=0x0001, seq=13/3328, ttl=128 (no response found!)
6127… 835.394644 192.168.31.27 192.168.31.7 ICMP 74 Echo (ping) request id=0x0001, seq=14/3584, ttl=128 (no response found!)
6127… 840.418856 192.168.31.27 192.168.31.7 ICMP 74 Echo (ping) request id=0x0001, seq=15/3840, ttl=128 (no response found!)
6128… 845.409858 192.168.31.27 192.168.31.7 ICMP 74 Echo (ping) request id=0x0001, seq=16/4096, ttl=128 (no response found!)
6131… 887.810990 192.168.31.27 192.168.31.37 ICMP 74 Echo (ping) request id=0x0001, seq=17/4352, ttl=128 (reply in 613175)
6131… 887.813867 192.168.31.37 192.168.31.27 ICMP 74 Echo (ping) reply id=0x0001, seq=17/4352, ttl=128 (request in 613174)
6131… 888.822704 192.168.31.27 192.168.31.37 ICMP 74 Echo (ping) request id=0x0001, seq=18/4608, ttl=128 (reply in 613180)
6131… 888.825377 192.168.31.37 192.168.31.27 ICMP 74 Echo (ping) reply id=0x0001, seq=18/4608, ttl=128 (request in 613179)
6131… 889.828494 192.168.31.27 192.168.31.37 ICMP 74 Echo (ping) request id=0x0001, seq=19/4864, ttl=128 (reply in 613189)
6131… 889.832086 192.168.31.37 192.168.31.27 ICMP 74 Echo (ping) reply id=0x0001, seq=19/4864, ttl=128 (request in 613188)
Frame 613175: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface \Device\NPF_{A87B7A28-2BA4-4DA3-B2C3-A3348EBA2A15}, id 0
Ethernet II, Src: Dell_a5:05:b6 (54:bf:64:a5:05:b6), Dst: Micro-St_c2:9d:17 (d8:bb:c1:c2:9d:17)
Internet Protocol Version 4, Src: 192.168.31.37, Dst: 192.168.31.27
Internet Control Message Protocol
Type: 0 (Echo (ping) reply)
Code: 0
Checksum: 0x554a [correct]
[Checksum Status: Good]
Identifier (BE): 1 (0x0001)
Identifier (LE): 256 (0x0100)
Sequence Number (BE): 17 (0x0011)
Sequence Number (LE): 4352 (0x1100)
[Request frame: 613174]
[Response time: 2.877 ms]
Data (32 bytes)




```
C:\Users\complab301pc25>ping 192.168.31.37

Pinging 192.168.31.37 with 32 bytes of data:
Reply from 192.168.31.37: bytes=32 time=2ms TTL=128
Reply from 192.168.31.37: bytes=32 time=2ms TTL=128
Reply from 192.168.31.37: bytes=32 time=3ms TTL=128
Reply from 192.168.31.37: bytes=32 time=2ms TTL=128

Ping statistics for 192.168.31.37:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 2ms, Maximum = 3ms, Average = 2ms
```



Niyati Savant
C31 - TE

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

2103156 Experiment - 10 21/08/23 (S2)

Aim: Socket Programming using TCP or UDP

Theory:

In computer networks, sockets are used for allowing the transmission of information between two processes of the same machine or different machines in a network. The socket is combination of IP address and software port number used for communication between multiple processes. It helps to recognize the address of the application to which data is to be sent using the IP address and port number. Thus sockets are the end of two-way communication between multiple applications. It provides Inter connected communication (IPC). The network can be logical, local network and physical network to connect external network.

The Socket module in Python is provides to make networking programs in Python. The socket() method helps to make socket INET stream and connect() method helps to connect the server.

Syntax: socket.socket(socket.AF_INET, socket.SOCK_STREAM)
where socket.AF_INET refers to the address-family ipV4
and socket.SOCK_STREAM refers to Connection Oriented TCP protocol

Syntax : socket.connect(IP address of Server, Port Number)
where IP address is of the server and port number is the one on which we have to connect.

To establish connection between client and server, we run the server program first and then the client otherwise we get a connection error. A client is a user or machine that generates a request to the server and gets response from server side. The server-side is a machine on which all operations are executed as requested by the client-side. To do so the server receives an input from client and performs the operations by itself. When result is ready, Python server sends output to client machine.

Some socket methods are.

a) Server socket methods
- s.bind - binds address hostname, port number to socket
- s.listen - Setups and start TCP listener
- s.accept - Passively accepts client connection, waiting until connection arrives blocking

b) Client socket methods
- s.connect - Actively initiates TCP server connection

c) General Socket methods
- s.recv - receives TCP message
- s.send - transmits TCP message
- s.close - closes socket

Ramesh 25/9/23

Program Socket programming using TCP/UDP

CODE:

Client code--

```
Import socket
SERVER = "127.0.0.1"
PORT = 8080
client = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
client.connect((SERVER, PORT))
print("input Example : 4 + 5")
print('For trigo function use the format- number operand another any number \n');
print("Type 'Over' to terminate")
while True:
        inp = input("Enter the operation in \ the form operand operator operand: ")
        if inp == "Over":
                break
        client.send(inp.encode())
        answer = client.recv(1024)
        print("Answer is "+answer.decode())
client.close()
```

Server code--

```
import math
import socket
LOCALHOST = "127.0.0.1"
PORT = 8080

server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind((LOCALHOST, PORT))
server.listen(1)
```

```
print("Server started")
print("Waiting for client request..")

clientConnection, clientAddress = server.accept()
print("Connected client:", clientAddress)

while True:
        data = clientConnection.recv(1024)
        msg = data.decode()

        if msg == 'Over':
                print("Connection is Over")
                break

        print("Equation is received")
        result = 0
        operation_list = msg.split()
        oprnd1 = float(operation_list[0])
        operation = operation_list[1]

        if operation == "+":
                oprnd2 = float(operation_list[2])
                result = oprnd1 + oprnd2
        elif operation == "-":
                oprnd2 = float(operation_list[2])
                result = oprnd1 - oprnd2
        elif operation == "/":
                oprnd2 = float(operation_list[2])
                if oprnd2 != 0:
```

```
                result = oprnd1 / oprnd2
            else:
                result = "Division by zero error"
        elif operation == "*":
            oprnd2 = float(operation_list[2])
            result = oprnd1 * oprnd2
        elif operation == "sin":
            result = math.sin(oprnd1)
        elif operation == "cos":
            result = math.cos(oprnd1)
        elif operation == "tan":
            result = math.tan(oprnd1)
        elif operation == "arcsin":
            result = math.asin(oprnd1)
        elif operation == "arccos":
            result = math.acos(oprnd1)
        elif operation == "arctan":
            result = math.atan(oprnd1)
        elif operation == "sqrt":
            result = math.sqrt(oprnd1)

print("Send the result to client")
output = str(result)
clientConnection.send(output.encode())
clientConnection.close()
```

OUTPUT:

Server:

Server started

Waiting for client request..

Connected client: ('127.0.0.1', 54818)

Equation is received

Send the result to client

Equation is received

Send the result to client

Equation is received

Send the result to client

Equation is received

Send the result to client

Equation is received

Send the result to client

Equation is received

Send the result to client

Equation is received

Send the result to client

Equation is received

Send the result to client

Equation is received

Send the result to client

Equation is received

Send the result to client

Client :

input Example : 4 + 5

For trigo function use the format- number operand another any number

Type 'Over' to terminate

Enter the operation in \ the form operand operator operand: 1 + 5

Answer is 6.0

Enter the operation in \ the form operand operator operand: 20 / 2

Answer is 10.0

Enter the operation in \ the form operand operator operand: 2 * 4

Answer is 8.0

Enter the operation in \ the form operand operator operand: 0 cos 0

Answer is 1.0

Enter the operation in \ the form operand operator operand: 0 sin 0

Answer is 0.0

Enter the operation in \ the form operand operator operand: 45 tan 0

Answer is 1.6197751905438615

Enter the operation in \ the form operand operator operand: 1 arcsin 0

Answer is 1.5707963267948966

Enter the operation in \ the form operand operator operand: 3.14 sin 0

Answer is 0.0015926529164868282

Enter the operation in \ the form operand operator operand: 25 sqrt 1

Answer is 5.0

Enter the operation in \ the form operand operator operand: Over

Niyati Savant CN

Niyati Savant
2103156

TE-C31

Written Assignment - 01

Parvati 24/8/23 (15)

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE 17/7/23

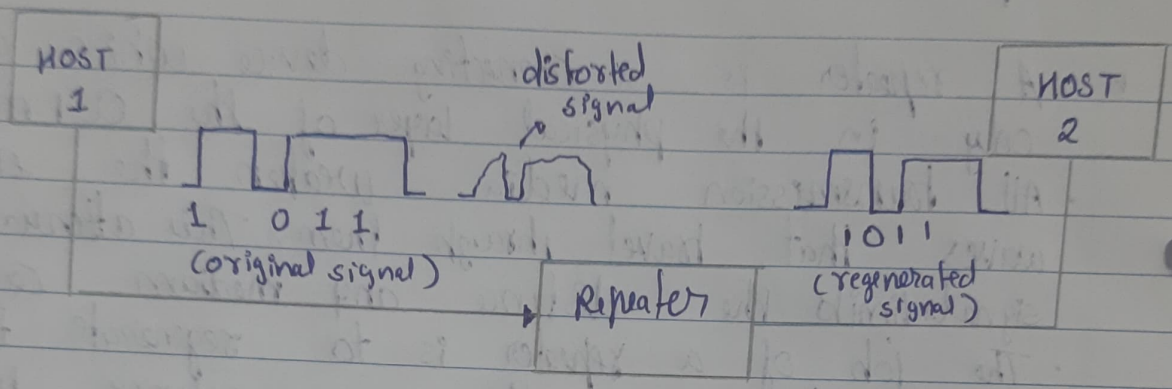
(1) (58)

Write a short note on the following networking devices:

1) Repeater

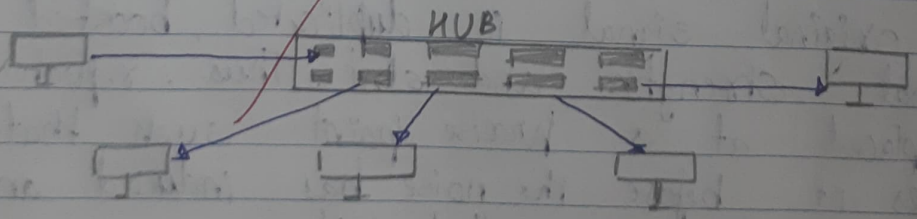
→ • A repeater is a connecting device which can operate only in the physical layer of the OSI model
• All transmission media weaken the electro magnetic waves that travel through them. This attenuation of signals limits the distance any medium can carry data.
• The job of a repeater is to regenerate the signal (original voltage value) over the same network before it gets too weak or corrupted so as to extend the length to which signal can be transmitted. Thus, it can be used to extend physical length of LAN by connecting only two devices in the same LAN as it is a 2 port devic
• Repeater is not an amplifier as it does not ~~simplis~~ simply amplify the entire incoming signal along with the noise Rather it creates an exact duplicate of the incoming data by identifying it amidst the noise, reconstructing it and retransmitting only desired information.
• The original signal is duplicated, boosted to its original strength and sent. Thus, repeater needs to be placed at a precise point such that the signal reaches it before the noise has induced an error in any of the transmitted bit

- A repeater has no filtering capability and thus cannot read sources and destination addresses and hence not capable of selective forwarding but uses flooding



2] Hub

→ - Hub is a multiport repeater and similar to the repeater, it operates only in the physical layer
- A signal received at any port is retransmitted on all other ports.
- Network segments that employ hubs are often described as having a star topology where hub forms the wiring center. As they don't filter data, packets are sent to all connected devices



- There are 3 types of hubs -

1) Passive
It simply combines signals of network segments as there is no processing or regeneration, thus it is merely a connector. Each computer receives signals sent from all other computers connected to the hub.

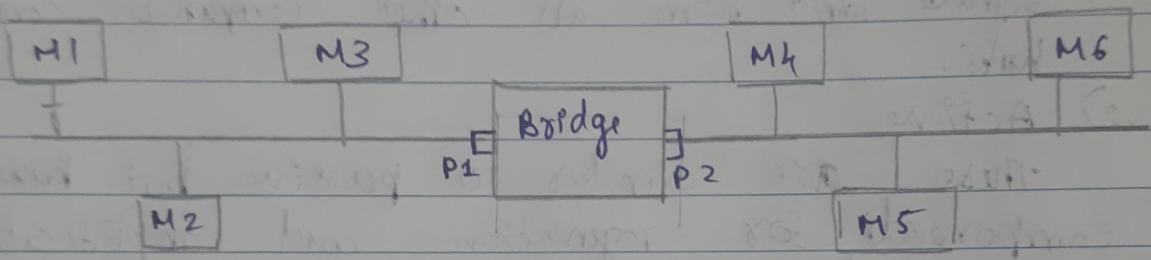
2) Active
These are similar to passive but have electronic components for regeneration and amplification thus increasing distance between devices. However, they also amplify noise along with the signal and are expensive

3) Intelligent
Also perform functions like network management and intelligent path selection ie chooses only the receiver port rather than sending signal along all paths

3. Bridges
A bridge operates at the data link layer and is a computer network device which creates a single aggregate network from multiple communication networks. It is a repeater with added functionality of filtering content by reading MAC addresses of source & destination and used to interconnect two LANs working on same protocol. It has a single input

and single output port i.e. a 2 port device.
- Bridge offers filtering capability ie it will check destination address of a frame and make a decision if the frame should be forwarded or dropped. If frame is not to be forwarded, bridge specifies the port over which it should forward.

4]

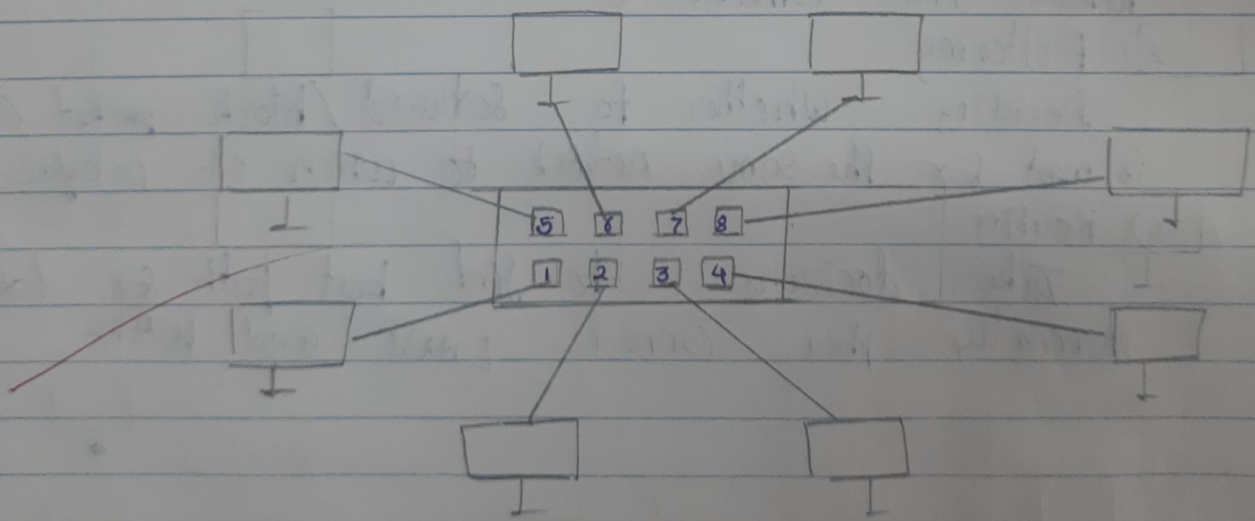


They are of two types
1) Static
The bridges maintain a table of MAC address - Port no. which is filled by the Network Admin. Thus if machine changes its interface, table must be updated manually
2) Dynamic
Initially, table is empty and bridge doesn't know which machine is connected to which interface
The bridge learns and makes entries as the data is passed and acknowledgment is received.

4] Switches
- A device that connects devices together on a network using packet switching to receive, process and forward data to destination. They operate at data link layer
- Also called multiport bridge as it provides bridging functionality with greater efficiency

to connect devices or segments in LAN. Switch has a buffer for each link to which it is connected.

- When it receives a packet, it stores packet in buffer of receiving link & checks address to find outgoing link, it then sends the frame. They also read MAC address and selectively forward packets
- They perform error checking before forwarding data & thus not forward packets with errors and forward good packets selectively to correct port only
- Types of switches
  1) Store-and-forward: stores the frame in input buffer till whole packet has arrived
  2) Cut-through: forwards packet to output buffer after getting destination address.



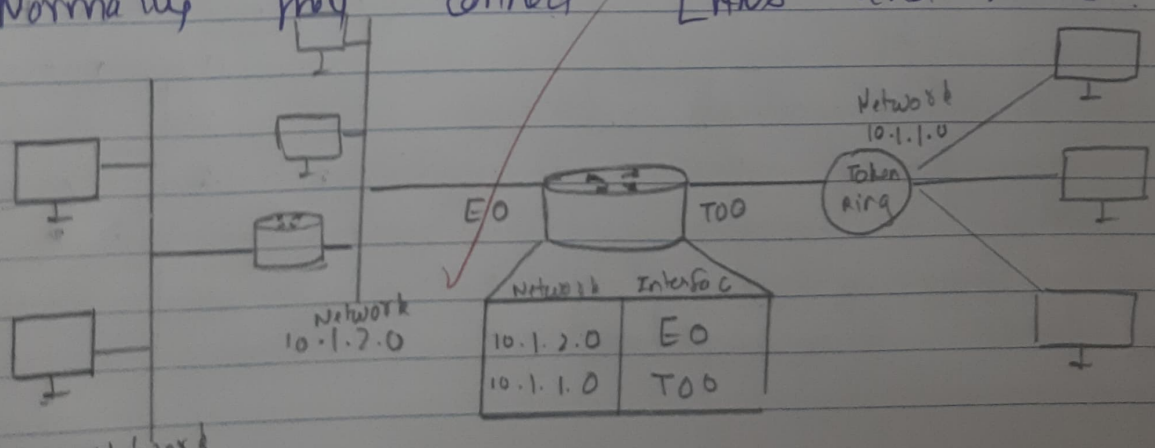
5] Router

- Networking device which forwards data packets between networks. They perform traffic directing function. The packets are routed based on their IP addresses and thus router acts on Network layer I
- Its functions are-

1) Forwarding
- If packet is to be send from a host on Network 1 to another host on network 2, packet has the source and destination IP address (MAC address used at layer 2 only in LAN, outside network need IP address). When packet reaches router, it forwards packet based on a inbuilt routing table that gives information about the networks its connected to

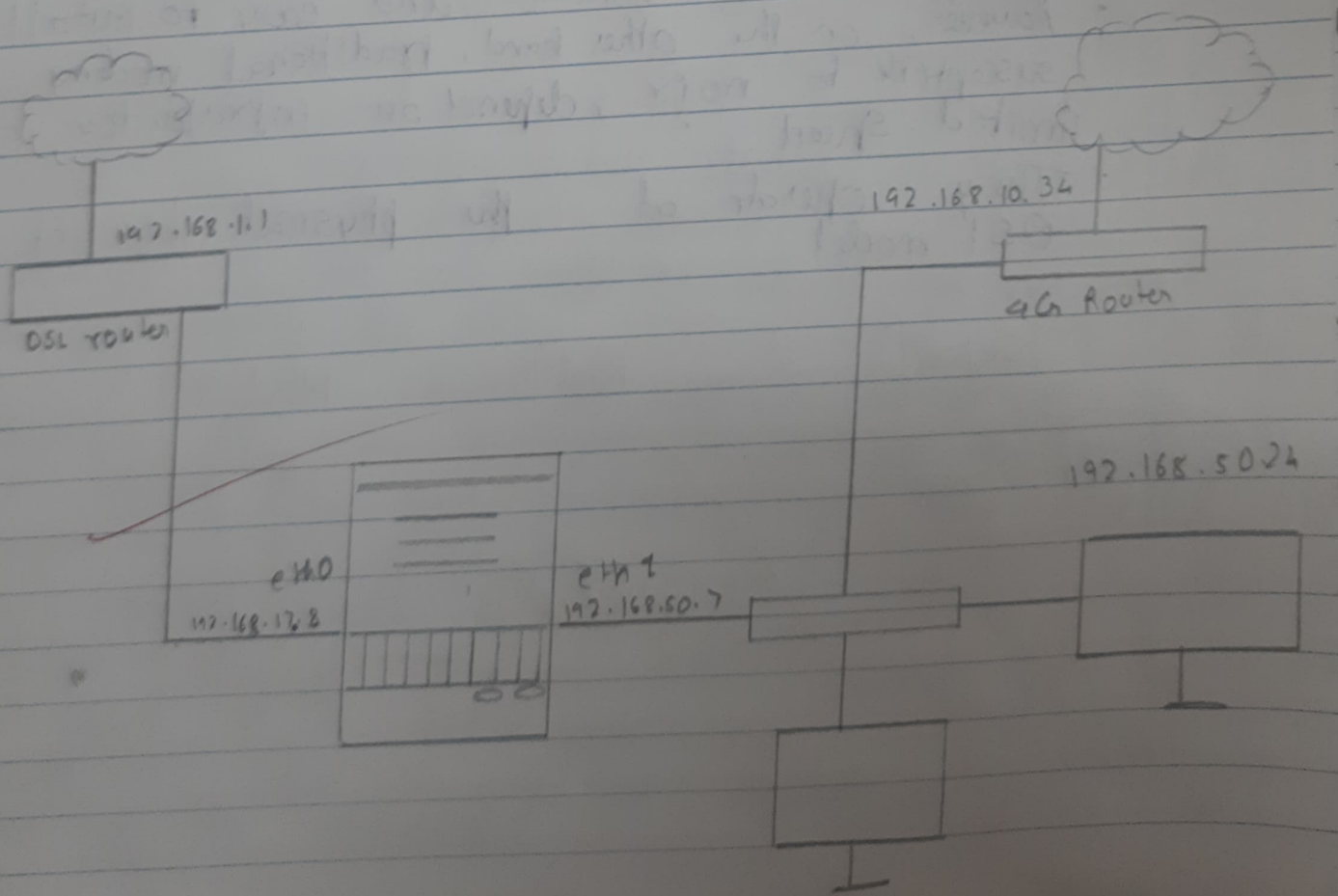
2) Filtering
Deciding whether to forward / block packet (if packet is meant for the same network to which it interfaces

3) Routing
Take decisions to find best path for data delivery
Normally they connect LANs and WANs



6] Gateways

- A gateway is a passage to connect two networks that may work upon different networking models. They work as messengers agents that take data from one system, interpret it and transfers it to another system.
- They operate at any network layer and are more complex than switches or routers. They are also called protocol converters.
- The default gateway is a router that connects host to remote network segments while the router delivers data within network.



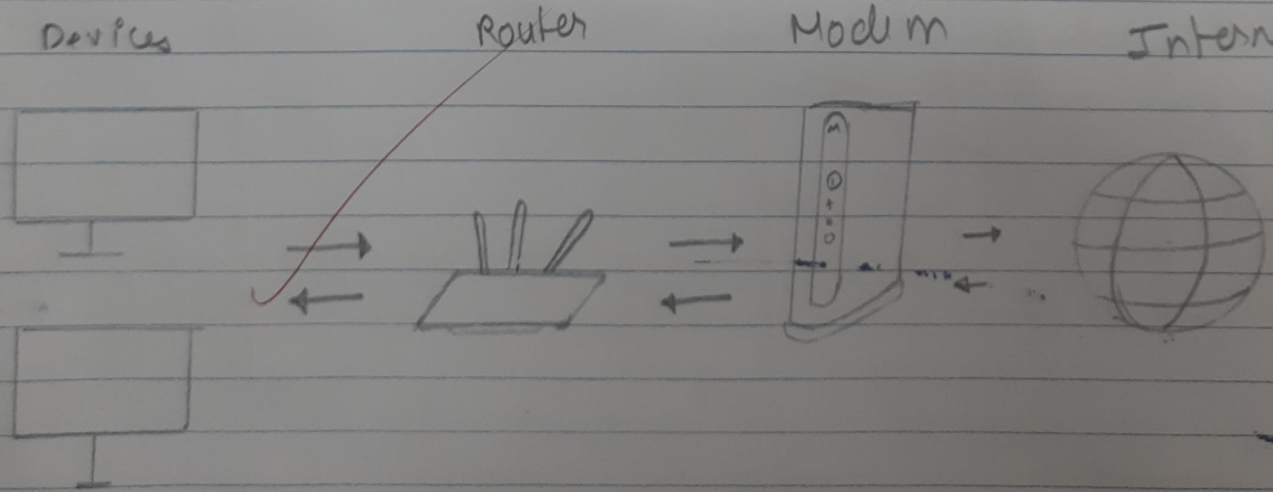
C3

CN

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

7] Modem

→ • A modem (modulator- demodulator) is a networking device that modulates digital data from a computer or network into analog signals that can be transmitted over analog communication channels like telephone lines and or cable system.
• It also demodulates incoming analog signals back to digital data for the receiving device.
• They are used to connect homes and businesses to Internet through connections like DSL (Digital Subscriber Line) or cable.
• Modems are widely available, compatible, are cost efficient and reliable and easy to install.
• However, on the other hand, traditional modems are susceptible to noise, depend on infrastructure and offers limited speed.
• They operate at the physical layer of the OSI model.



C3 CN

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

(1) 2103156 Written Assignment - 02 (2)

Q1] Write short note on

i) Ethernet

→ • Ethernet is the most widely used LAN technology since it is easy to understand, implement, maintain and allows low cost network implementation
• It offers flexibility in terms of topologies that are allowed. It operates at physical and data link layer.
• The protocol data unit is a frame which Access control mechanism to deal with collision is CSMA/CD
• Although it is largely replaced by wireless network, the large companies still prefer Ethernet due to its security and being immune to interference
• The types of Ethernet networks are.
a) Fast Ethernet.
Use CAT5 or twisted pair cable to transfers data at 100 Mbps.
b) Gigabit Ethernet
Use advance cables like CAT5 e, to transfer data at 10 Gbps
c) 10-Gigabit Ethernet
advanced and high-speed network that uses CAT6a or CAT7 cables to transmit 10 Gbps at a distance of 10,000 meter

d) Switch Ethernet

Involves using switches/hubs to improve netwo performance. Support a wide range of speeds from 10 Mbps to 10 Gbps.

ii) IPv6

→ It was developed by Internet Engineering Task Force to deal with the problem of IPv4 exhaustion. IPv6 is a 128 bit address thus having address space of 2^{128}. It uses the Hexa-Decimal format separated by colon (:).

The components in Address format are

a) There are 8 groups, each one representin 16 bits (2 bytes)

b) Each Hex-digit is a nibble (4 bit)

c) A colon (:) acts as delimiter

This hexadecimal format has 32 digits and is long. An abbreviation is used to shorten it by omming leading zeros in a section. Further, zero compression can be used to replace consecutive sections only having zeros by double colon

Eg: Unabbreviated address

BBFF:AC81:9840:0086:3210:000A:0000:FFFE

Abbreviated address

TO

BBFF : AC81 : 9840 : 86 : 3210 : A : 0 : FFFF

Abbreviated address

AC81 : 0 : 0 : 0 : 0 : BBFF : 0 : FFFF

↓

AC81 :: BBFF : 0 : FFFF

IPv6 supports unicast, multicast and anycast address

By looking at first few bits we can identify type of address

Eg: a prefix 0000 0000 that takes 1/256 fraction of address space is for reserved address prefix 1111 1111 with 1/256 supports multicast address

Thus, IPv6 resolves the issues of IPv4 by giving large address space, better header format, new options for additional functionalities, extension allowance, resource allocation support and more security

iii) ssh

→ SSH (Secure Shell) is access credential that is used in the SSH Protocol. It is a cryptographic network protocol that is used for transferring encrypted

data over network. It allows you to connect to a server/multiple servers without having to enter your password for each system ie to login remotely from one system to another. It always comes in key pair

- Public key - For encryption, everyone can see and no need to protect it
- Private key - For decryption, stays in computer and must be protected

Key pairs can be of the following types

1) User key - Both public key and private key remain with user
2) Host key - Both public key & private key are on remote system
3) session key - Used when large amount of data is to be transmitted

SSH uses asymmetric cipher for performing encryption and decryption

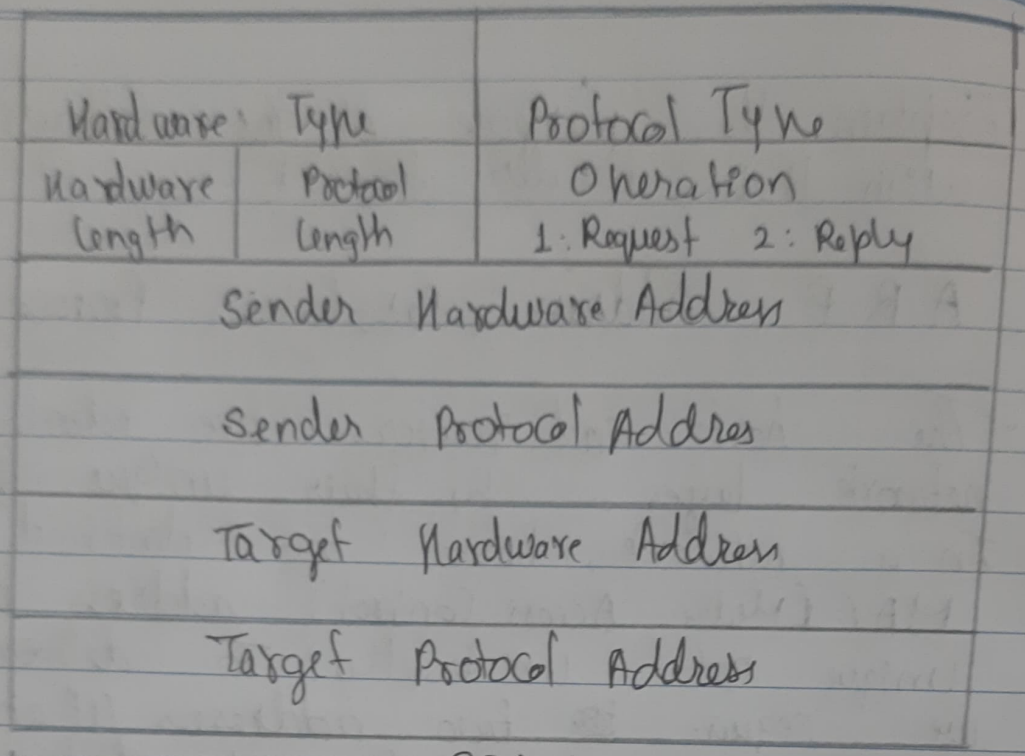
Q2] Explain the purpose of the following protocols with their header format

i) ARP - Address Resolution Protocol

→ • The host and routers are identified at the network layer by their unique IP addresses.
• In a network, hosts are identified by their MAC (Media Access Control) address which is locally unique not universally. To deliver a packet we require two addresses (IP and MAC)
• The domain names identify the destination and the DNS maps name to IP addresses which is thus known but not the MAC.
• ARP maps an IP address to its physical address
• When a host/router needs to find MAC address of another host/routers it broadcast an ARP query packet that includes physical and IP address of sender and IP address of receiver.
• Every host on network receives and processes the ARP query packet but the intended recipient recognizing its IP address ~~send~~ unicasts an ARP response packet that ~~includes~~ includes recipients IP and MAC address

The header format is -

(P.T.O)



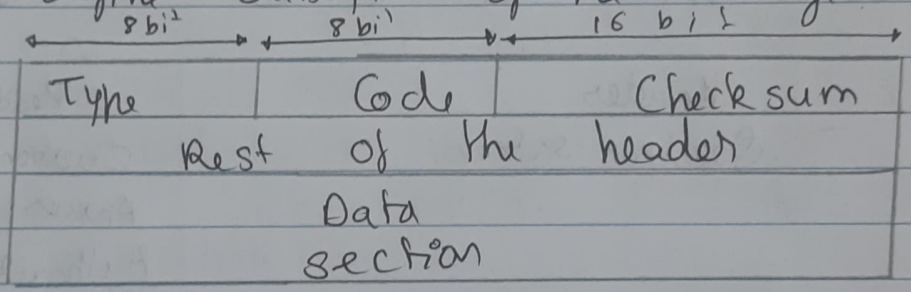
- Hardware type - 16 bit to define network type (Ethernet is 1)
- Protocol type - 16 bit to define protocol (IPv4 is 0800 H)
- Hardware length - 8 bit to define length in bytes (Ethanet is 6)
- Protocol length - 8 bit to define length in byte (IPv4 is 4)
- Operation - 16 bit to define type of packet
- Sender hardware - variable length (Ethernet is 6 byte)
- Sender protocol - variable length (IP is 4 byte)
- Target hardware - variable length (Ethernet is 6 byte)
- Target protocol - variable length (IPv4 is 4 byte)

ii) ICMP - Internet Control Message Protocol

- It has been designed to compensate for the lack of mechanism for host and management queries and no error-reporting or correcting mechanism of IP protocol.

It is a network layer protocol used by routers, intermediary devices and hosts to communicate error information or updates to other routers, hosts

- They can be categorized into error reporting (like redirection, time exceeded) and query messages (echo reply, time stamp request)
- For a error type message, it has a type that defines the type of message and a code to define subtype of the message



- Type - An 8 bit field that defines ICMP message type
- Code - 8 bit field to define subtype of message
- Checksum - 16 bit field to detect whether error exist

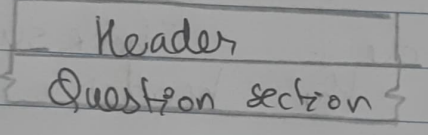
iii) DNS - Domain Name System

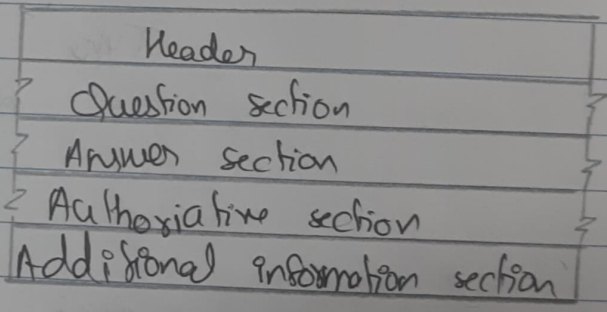
→ It is a translation service from hostname to IP address. It is a distributed database implemented in a hierarchy of name servers. It is an application layer protocol for message exchange between clients and servers.
DNS is used to convert domain name of

websites to their numerical IP address. The domain names could be generic (.com, .edu, .org), country (.in, .us), inverse (send domain name of website)

DNS has two types of messages -
a) Query b) Response

The format is similar to both with some fields set to 0 for query





Q3 Discuss Persistent and Non-Persistant protocols used in Transport and Application layers of TCP/IP protocol suite

→ The HTTP is an application-level protocol that uses TCP as an underlying transport and typically runs on port 80.
HTTP connections can be of two types Non-persistent and persistent

Non-Persistent Connection

→ These are the connections in which for each object we have to create a new connection for sending the object from source to destination. We can send a maximum of one object from one TCP connection
There are two types:-

1) Non-Persistent without parallel connection
Each objection takes two RTTs (assuming no window limit) one for TCP connection, other for HTTP image/text file
2) Non-Persistent-with parallel connection
They require an extra overhead in transferring data
These connection is more secure and has very less wastage of resources. However, they need a greater CPU overhead for the transmission of data.

Persistent Connection

The server leaves the connection open after sending a response. Subsequent HTTP messages between the same client/server are sent over an open connection. The client sends requests as soon as it encounters a

referenced object. The persistent connection is of two types

1) Non pipelined persistent connection
We first establish a connection that takes two RTTs then we send all the object's images / text files which take 1 RTT each (TCP for each object is not required)

2) Pipelined persistent Connection
In pipelined connection, 2RTT is for connection establishment and then 1RTT (assuming no window limit) for all the objects i.e images / text.

It has lower CPU and memory usage and allows HTTP pipelining of requests and responses. There is also reduced latency in subsequent requests. Resources may be kept occupied even when not needed.