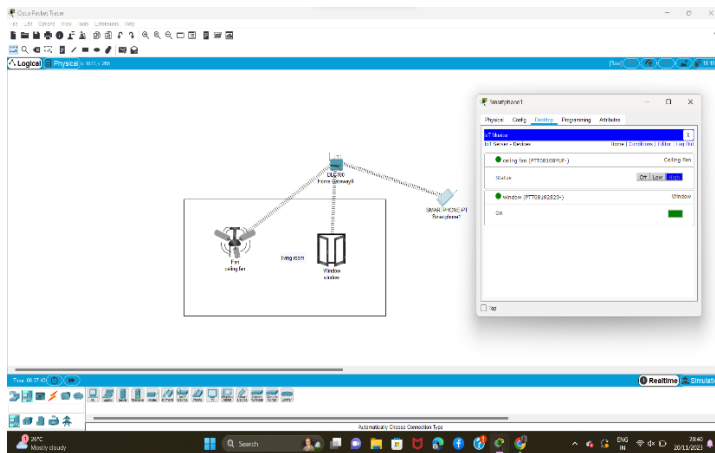
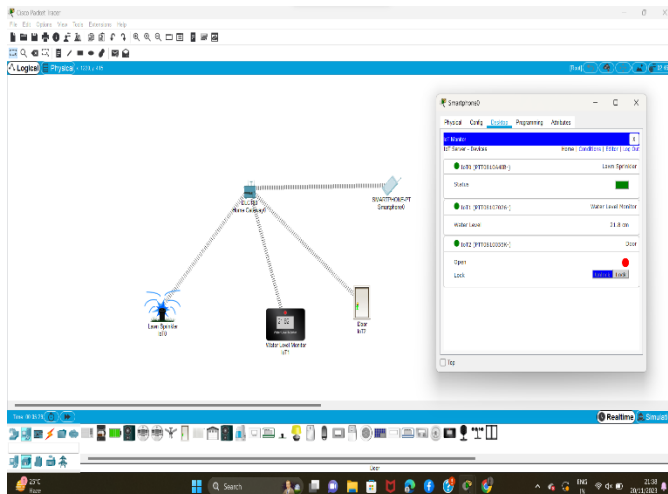


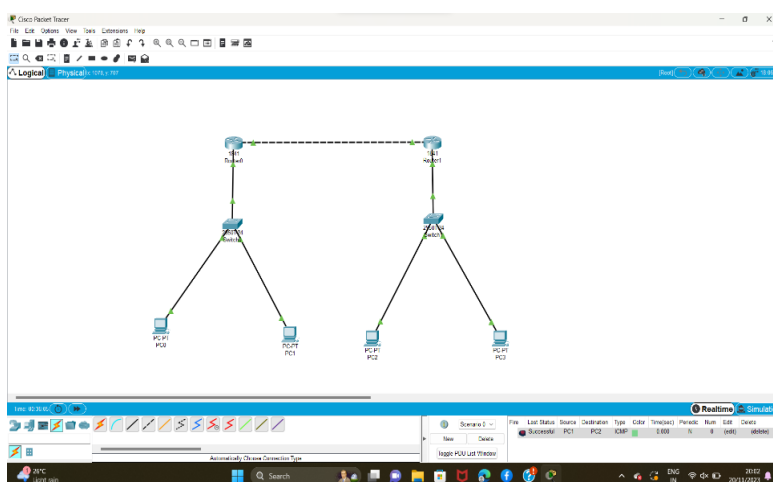
Lab ex output



lot network device fan and window

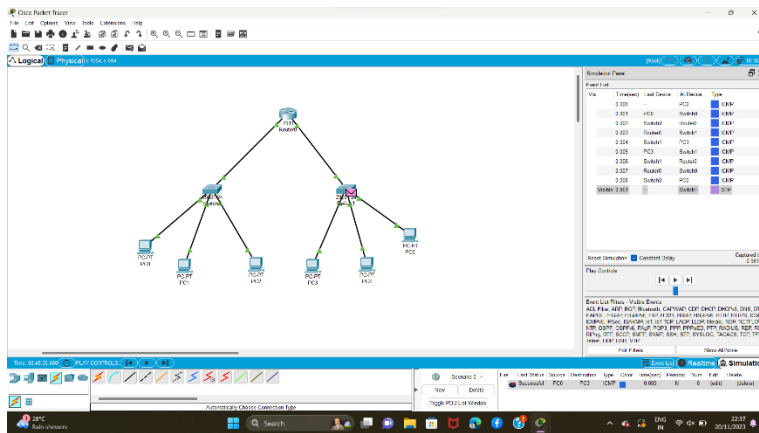


lot spring smart garden

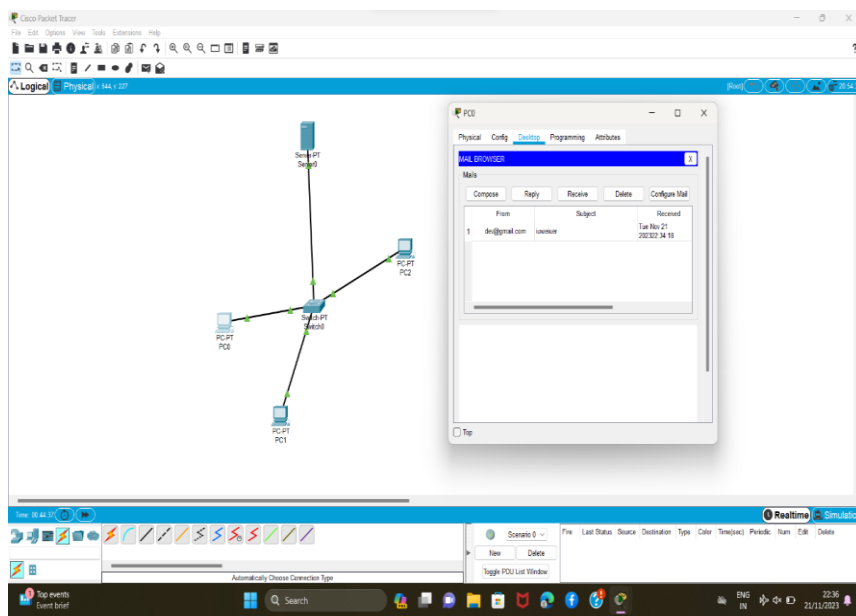


Static router

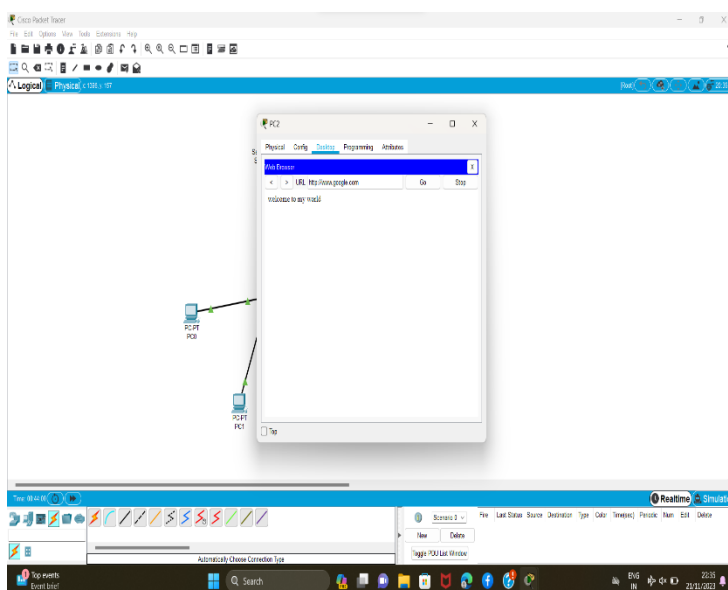
Lab ex output



Subnetting



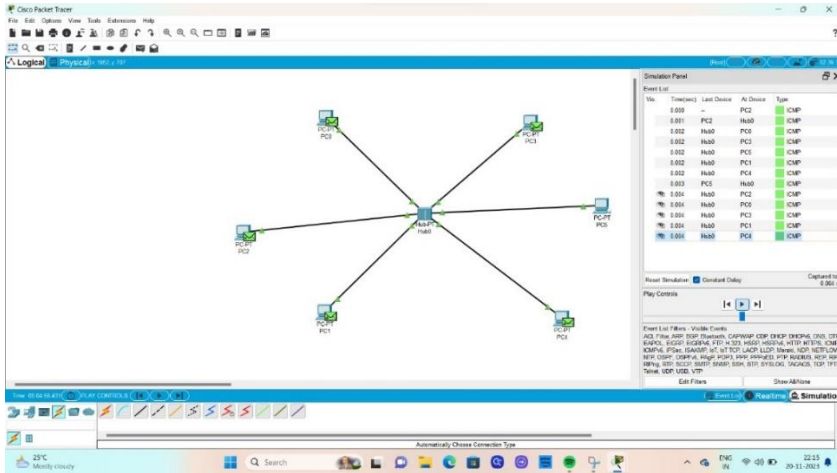
Tcp



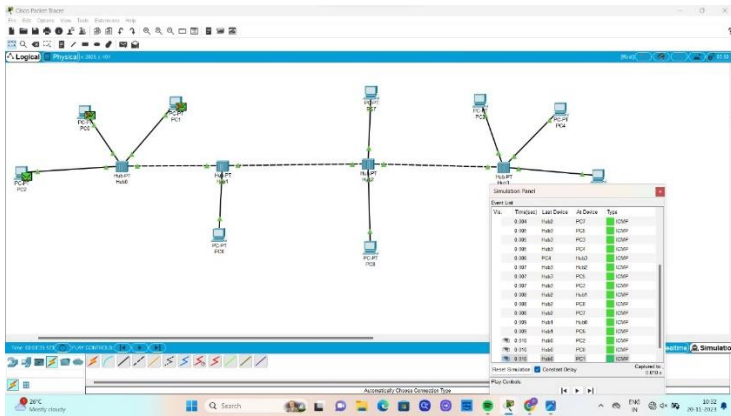
Tcp browsers

Lab ex output

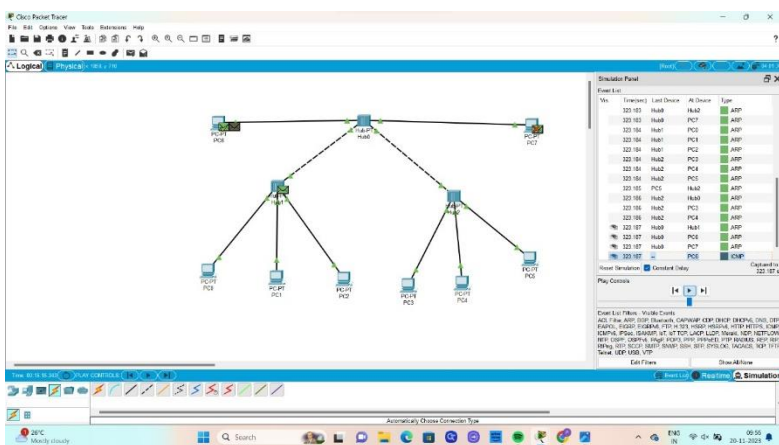
Topology



Star

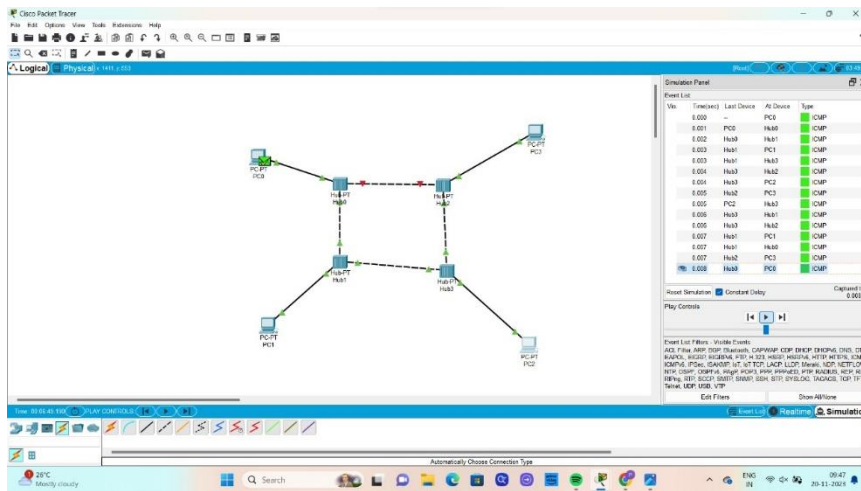


Hybrid

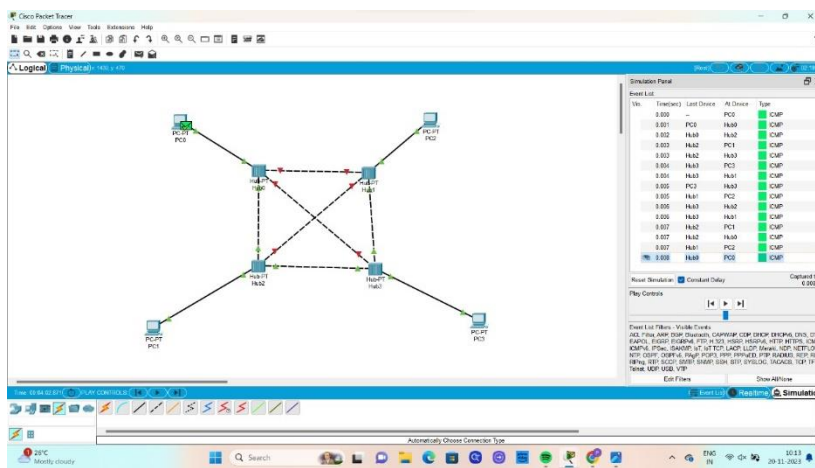


Tree

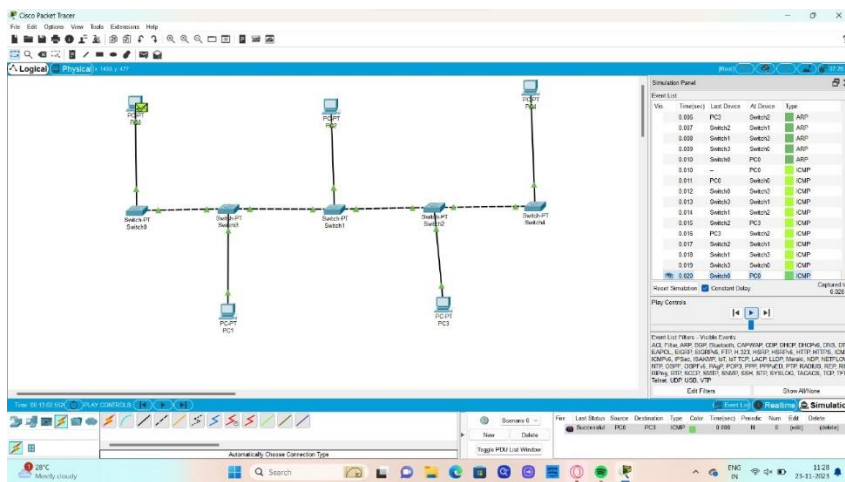
Lab ex output



Ring

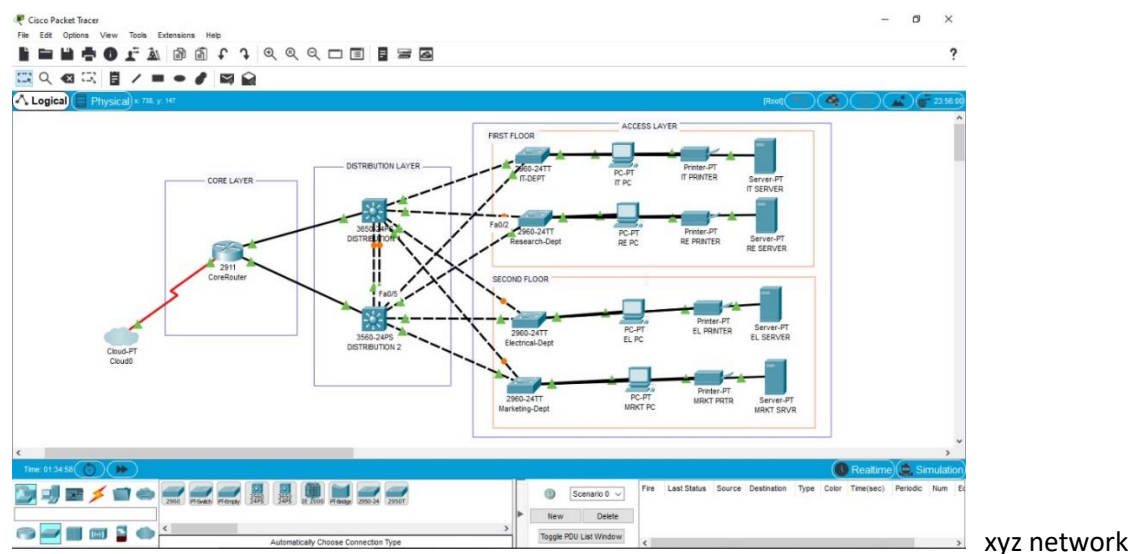
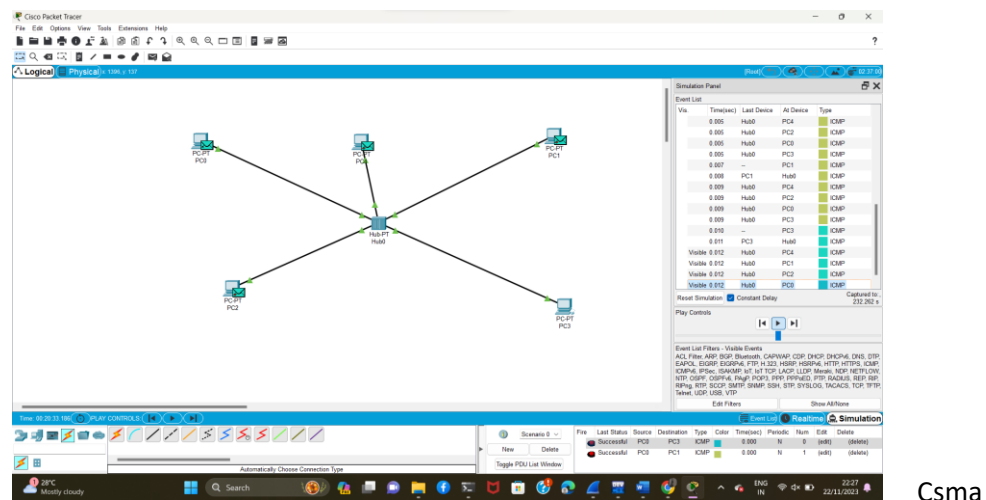
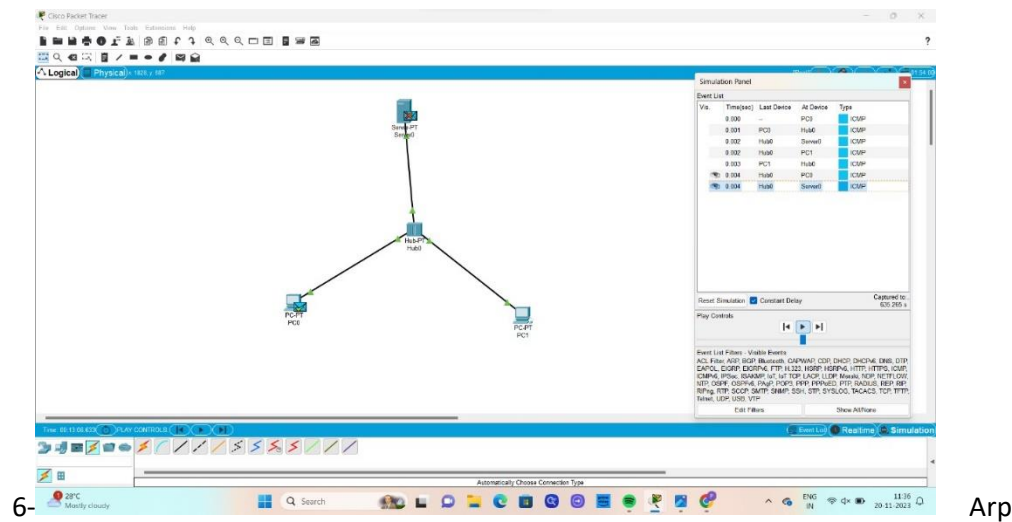


Mesh

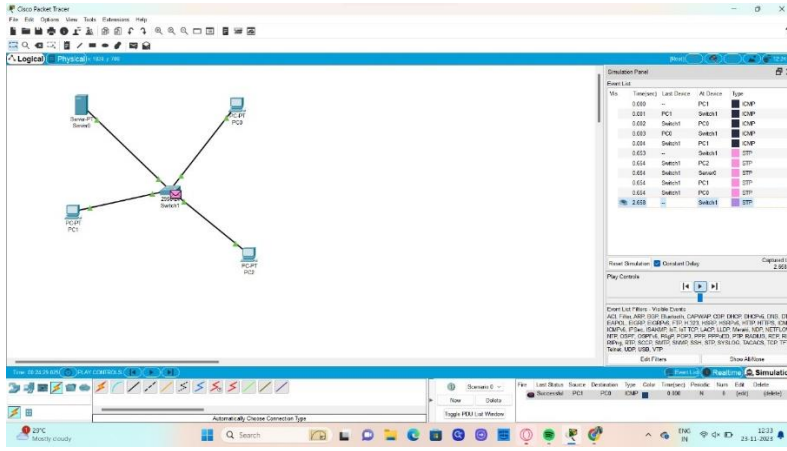


bus

Lab ex output



Lab ex output



fire wall

Wire shark

The screenshot displays the Wireshark network protocol analyzer interface. The top menu bar includes File, Edit, View, Go, Capture, Analyze, Statistics, Telephony, Wireless, Tools, and Help. The toolbar contains icons for various functions like opening files, saving, and capturing. The main window is divided into three panes: Packet List, Packet Details, and Packet Bytes.

Packet List: Shows a single captured packet (No. 170) at time 172.034894. It is an ARP packet (Protocol: ARP) with a length of 40 bytes. The source is 1a:3a:58:2b:c8:8e and the destination is CloudNet_3c:b5:5f.

Packet Details: The selected packet is expanded, showing the following structure:

- Ethernet II, Src: 1a:3a:58:2b:c8:8e (1a:3a:58:2b:c8:8e), Dst: CloudNet_3c:b5:5f (30:03:c8:b3:c8:5f)
- Address Resolution Protocol (request)

Packet Bytes: The raw data of the selected packet is displayed in hexadecimal and ASCII. The hexadecimal data is: 0000 30 03 c8 b3 c8 5f 1a 3a 58 2b c8 8e 00 00 01 00 00 00 04 00 01 1a 3a 58 2b c8 8e c0 a4 a7 81 00 00 00 00 00 00 c0 a4 a4 40. The ASCII data is:X.....X.....X.....K.

The bottom status bar indicates the current filter is 'Address Resolution Protocol: Protocol' and shows 1537 packets displayed, with 16 (1.0%) selected.

arp

Lab ex output

The image shows a Wireshark packet capture of a TCP connection. The packet list on the left shows several packets, with packet 68 selected. The packet details pane on the right shows the structure of the selected packet, including Ethernet II, Internet Protocol Version 4, and Transmission Control Protocol. The packet bytes pane on the right shows the raw data in hexadecimal and ASCII.

Packet 68: 74.443.49891 [ACK] Seq=180 Win=0 Len=0

Packet details:

- Ethernet II, Src: CloudNet_3c:b5:5f (30:03:c8:3c:b5:5f), Dst: fe:ea:97:1a:74:04 (fe:ea:97:1a:74:04)
- Internet Protocol Version 4, Src: 2401:4900:234e:c185:81b1b1b1b1b1b1b1, Dst: 2400:140f:4200::17c:3588
- Transmission Control Protocol, Src Port: 49893, Dst Port: 80, Seq: 113, Ack: 189, Len: 0

Packet bytes:

```
0000  fe ea 97 1a 74 04 30 03  c8 3c b5 54 86 d0 0e  ....t.....
0010  0a 11 00 14 06 3f 24 01  49 00 23 4e c1 85 81 f0  ....I-IR....
0020  b1 5f 6f 46 37 3d 26 00  14 0f 42 00 00 00 00 00  ....-d-...
0030  00 00 17 c9 35 88 c2 e5  00 50 e5 0b d9 18 e7 b4  ....P.....
0040  b3 da 50 10 01 02 1b f4  00 00  ....P.....
```

tcp

The image shows a Wireshark packet capture of a UDP connection. The packet list on the left shows several packets, with packet 299 selected. The packet details pane on the right shows the structure of the selected packet, including Ethernet II, Internet Protocol Version 4, and User Datagram Protocol. The packet bytes pane on the right shows the raw data in hexadecimal and ASCII.

Packet 299: 66.57021.57021 [ACK] Seq=180 Win=256 Len=0

Packet details:

- Ethernet II, Src: Intel_99:f7:f5 (e4:00:36:99:f7:f5), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
- Internet Protocol Version 4, Src: 192.168.220.172, Dst: 192.168.220.255
- User Datagram Protocol, Src Port: 57021, Dst Port: 57021

Packet bytes:

```
0000  ff ff ff ff ff ff e4 0d  36 99 f7 f5 08 00 45 00  ....K.....
0010  00 49 19 a1 00 00 00 11  00 00 c0 a8 dc ac c0 a8  ....H.....
0020  4c ff e1 15 e1 15 00 34  51 33 53 78 6f 7a 95 64  ....-K-7...
0030  78 30 4b b0 7d 37 70 c2  05 7f 00 01 00 04 48 95  ....pOK 77p...
0040  c2 03 f4 60 04 b5 a4 37  3a c5 06 3c 5c 7a 93 e3  ....-K-7...
0050  4f aa c8 03 00 00  ....O.....
```

udp

Lab ex output

The screenshot shows a Wireshark capture of an HTTP GET request. The packet list on the left shows a single packet (No. 1152) of length 1047 bytes, which is a reassembled PDU. The packet details pane shows the following structure:

- Frame 1152: 1047 bytes on wire (8376 bits), 1047 bytes captured (8376 bits) on interface \Device\NPF... (192.168.228.172)
- Ethernet II, Src: ae:d4:11:ab:10:54 (ae:d4:11:ab:10:54), Dst: Intel_99:f7:f5 (ae:d4:11:ab:10:54)
- Internet Protocol Version 4, Src: 192.168.228.172, Dst: 192.168.228.172
- Transmission Control Protocol, Src Port: 80, Dst Port: 58556, Seq: 1204801, Ack: 1480, Len: 993
- Hypertext Transfer Protocol
- Data (121393 bytes)

The packet bytes pane shows the raw data of the HTTP request, including the GET method and the URL.

http

The screenshot shows a Wireshark capture of an ICMP Echo (ping) request and response. The packet list on the left shows two packets (No. 580 and 581) of length 74 bytes each. The packet details pane for packet 580 shows the following structure:

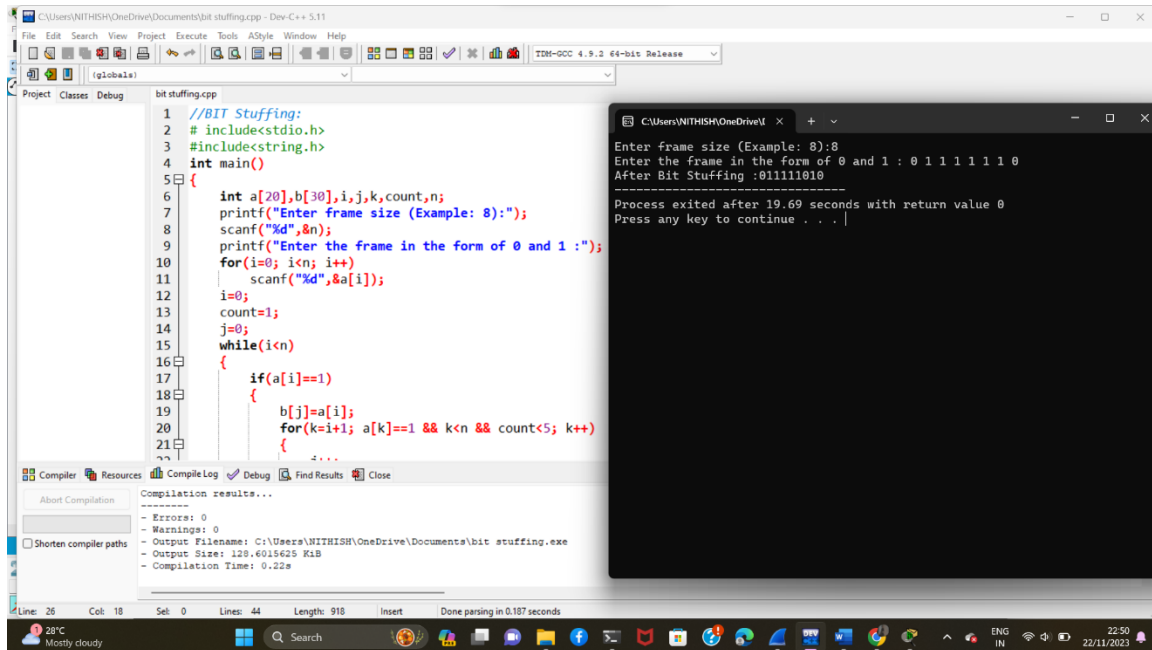
- Frame 580: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface \Device\NPF... (192.168.228.172)
- Ethernet II, Src: ae:d4:11:ab:10:54 (ae:d4:11:ab:10:54), Dst: Intel_99:f7:f5 (ae:d4:11:ab:10:54)
- Internet Protocol Version 4, Src: 192.168.228.172, Dst: 192.168.228.172
- Internet Control Message Protocol
- ICMP Echo (ping) request, id=600001, seq=46/11776, ttl=128

The packet bytes pane shows the raw data of the ICMP Echo request, including the type, code, and identifier.

icmp

Lab ex output

C program



The screenshot shows a C program in Dev-C++ titled 'bit stuffing.cpp'. The code implements a bit stuffing algorithm. It prompts the user to enter a frame size (n) and then a frame of 0s and 1s. It then displays the frame after bit stuffing, which appends a 0 to the original frame if the number of 1s is even, or a 1 if it is odd. The program also shows the compilation results and the execution output in a separate window.

```
1 //BIT Stuffing:
2 #include<stdio.h>
3 #include<string.h>
4 int main()
5 {
6     int a[20],b[30],i,j,k,count,n;
7     printf("Enter frame size (Example: 8):");
8     scanf("%d",&n);
9     printf("Enter the frame in the form of 0 and 1 :");
10    for(i=0; i<n; i++)
11        scanf("%d",&a[i]);
12    i=0;
13    count=1;
14    j=0;
15    while(i<n)
16    {
17        if(a[i]==1)
18        {
19            b[j]=a[i];
20            for(k=i+1; a[k]==1 && k<n && count<5; k++)
21            {
22                b[j+1]=a[k];
23                j++;
24            }
25            count++;
26            i++;
27        }
28    }
29    b[j]=0;
30    printf("After Bit Stuffing :");
31    for(i=0; i<j; i++)
32        printf("%d",b[i]);
33    printf("\n");
34    return 0;
35 }
```

Compilation results...

- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\NITHISH\OneDrive\Documents\bit stuffing.exe
- Output Size: 128.6015625 KiB
- Compilation Time: 0.22s

Execution output:

```
Enter frame size (Example: 8):8
Enter the frame in the form of 0 and 1 : 0 1 1 1 1 1 1 0
After Bit Stuffing :0111111010
Process exited after 19.69 seconds with return value 0
Press any key to continue . . .
```

8 bit stuffing