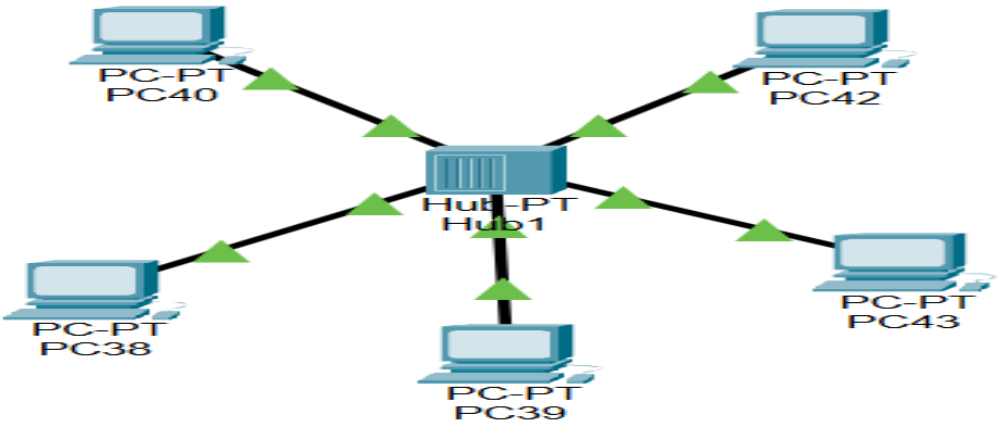
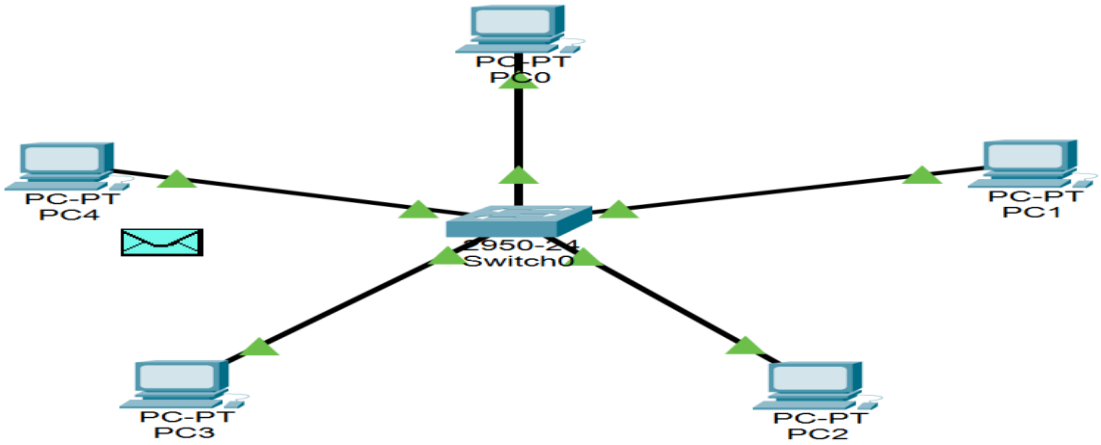
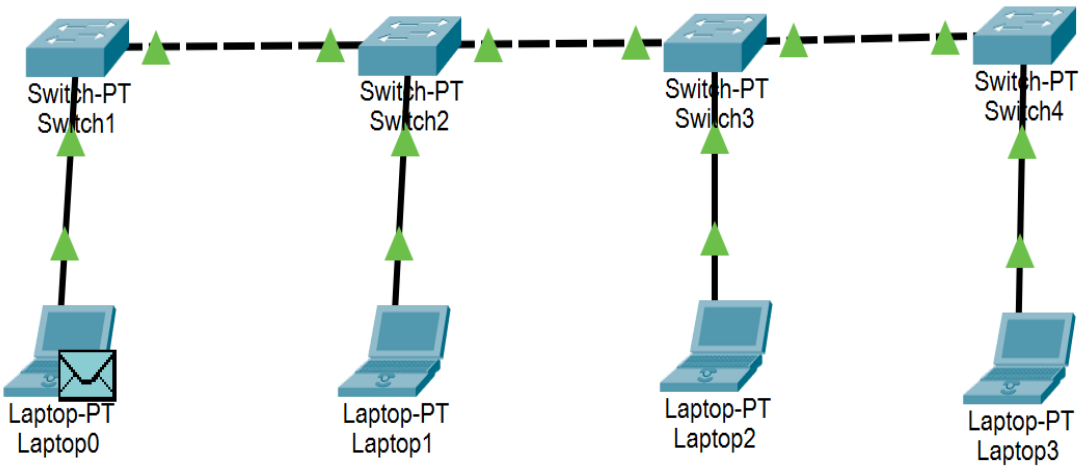
CONFIGURATION OF NETWORK DEVICES



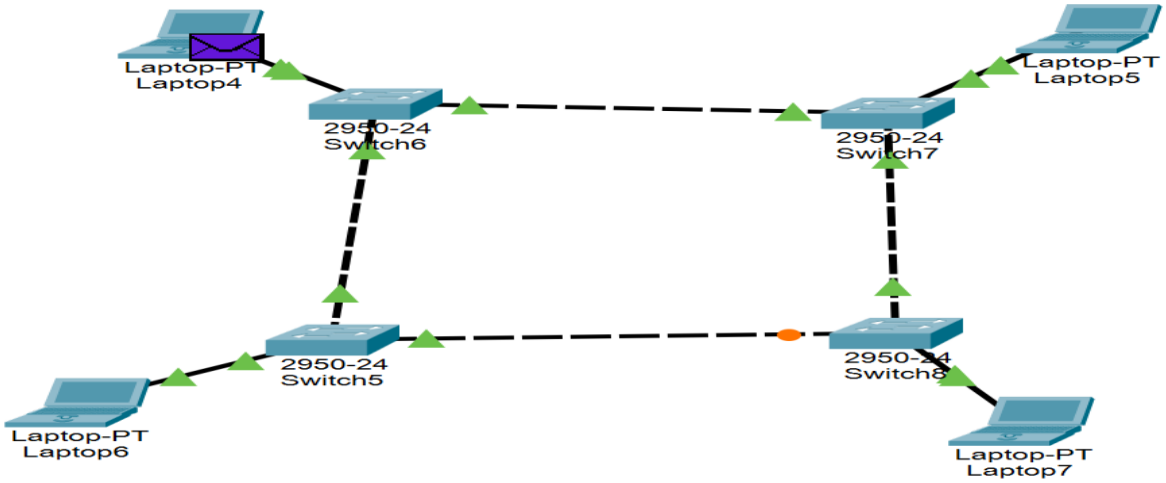
STAR TOPOLOGY



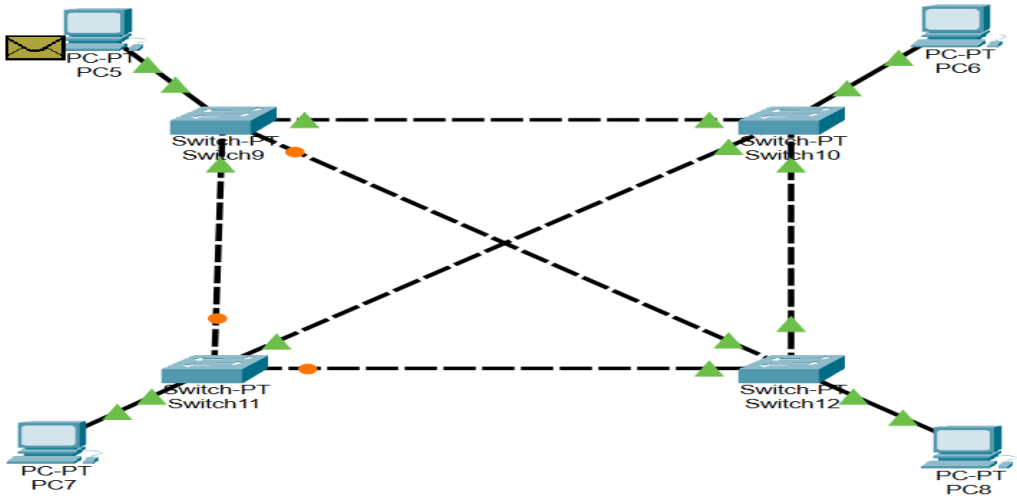
BUS TOPOLOGY



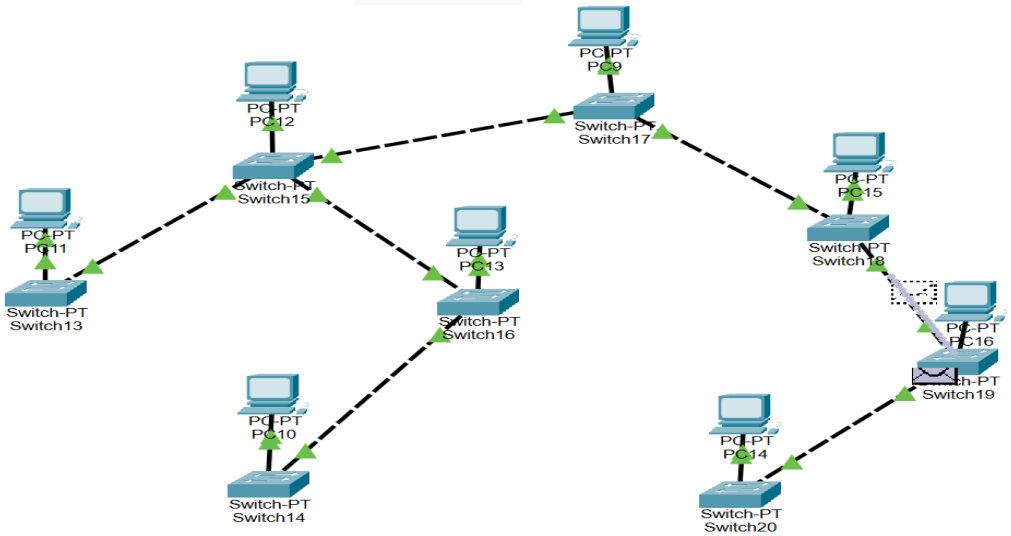
RING TOPOLOGY

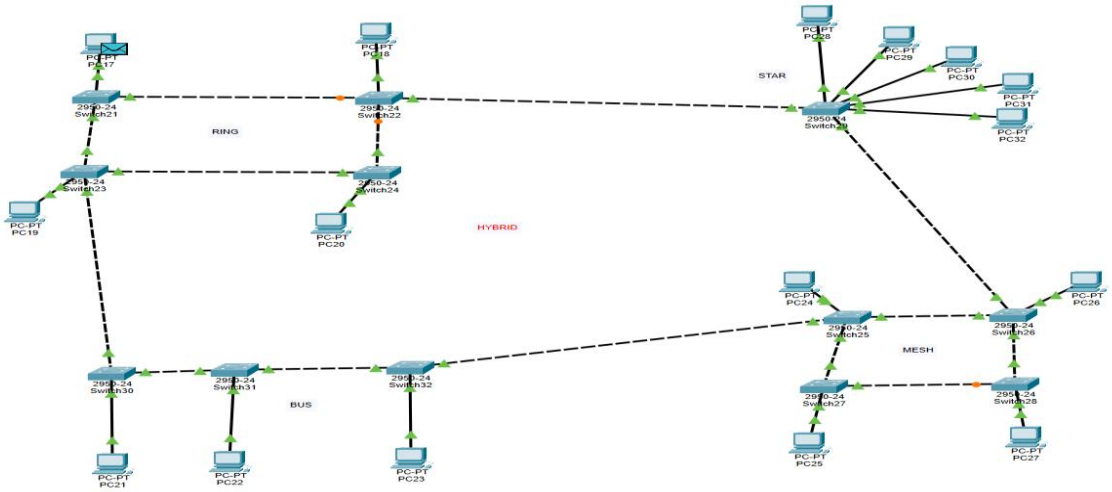


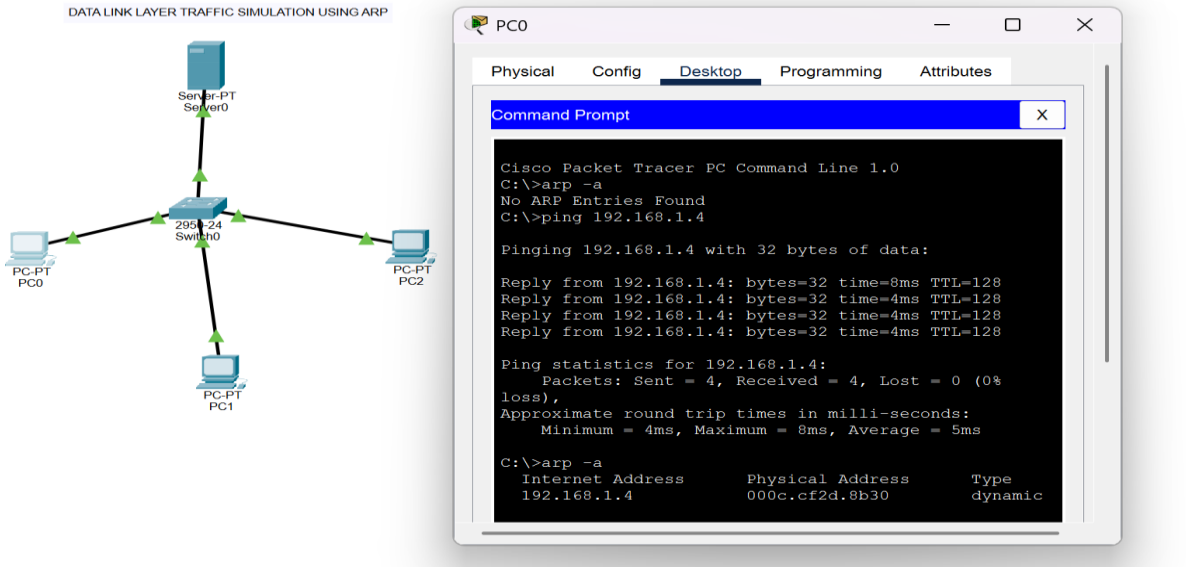
MESH TOPOLOGY



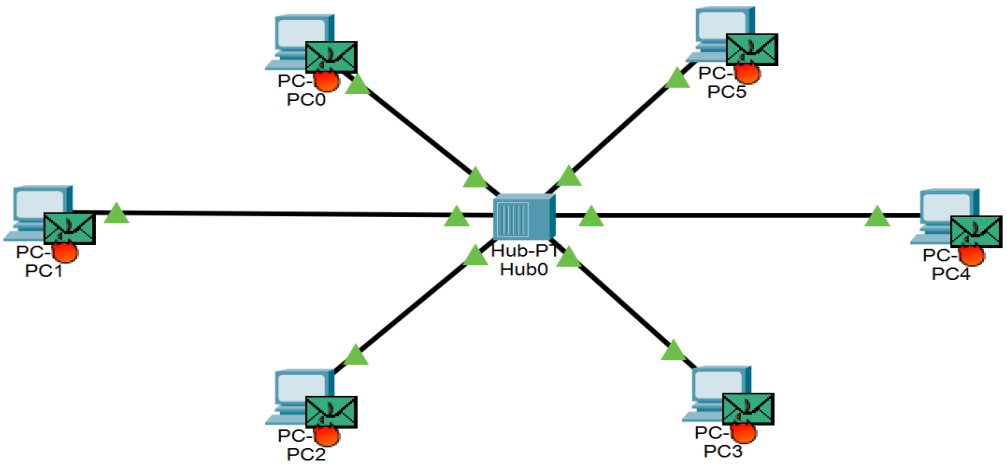
TREE TOPOLOGY

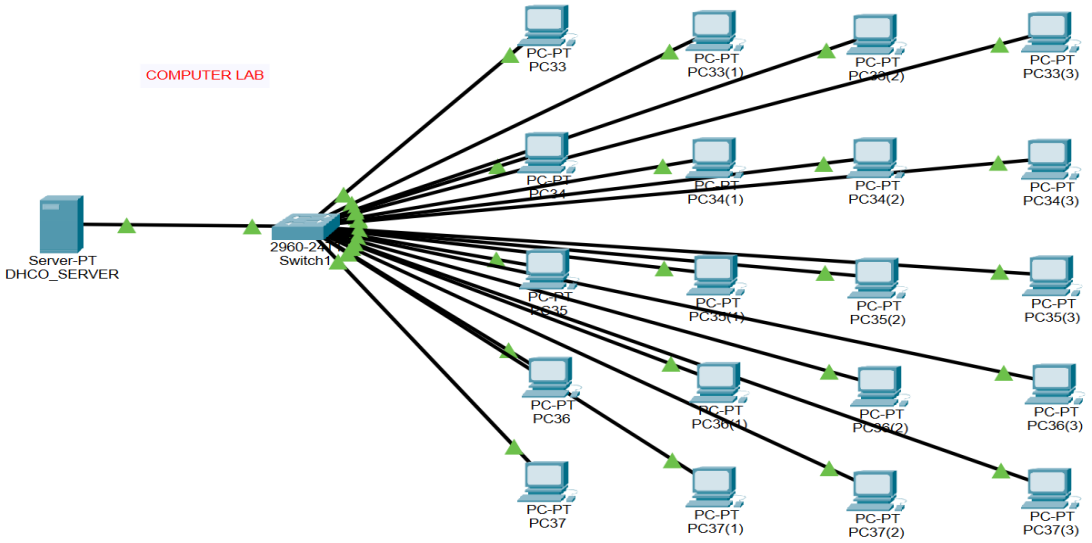


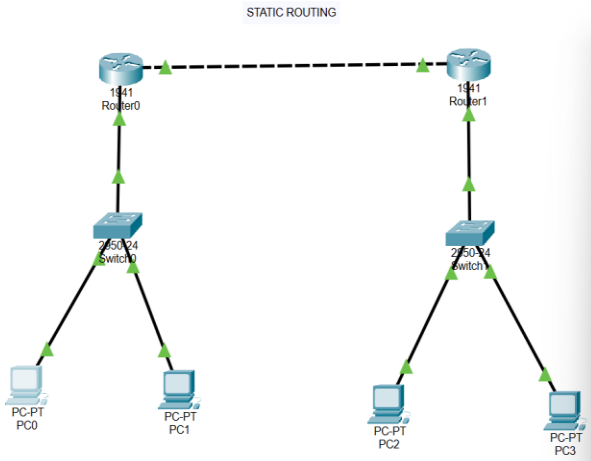


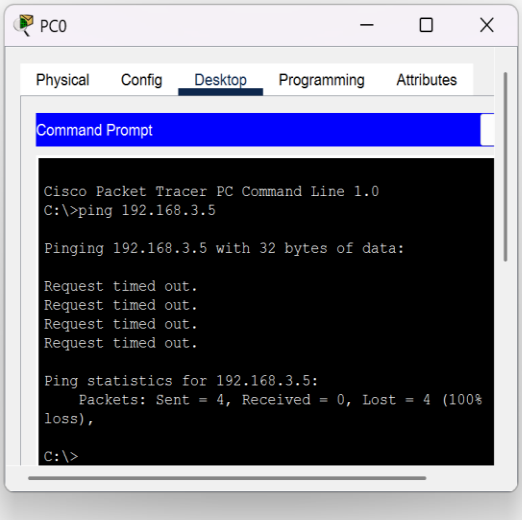


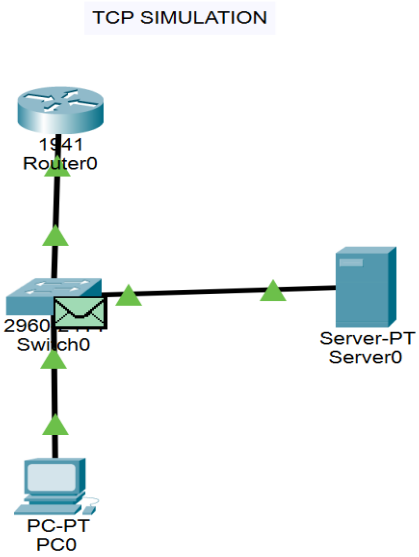
Data Link Layer Traffic Simulation of CSMA/CD

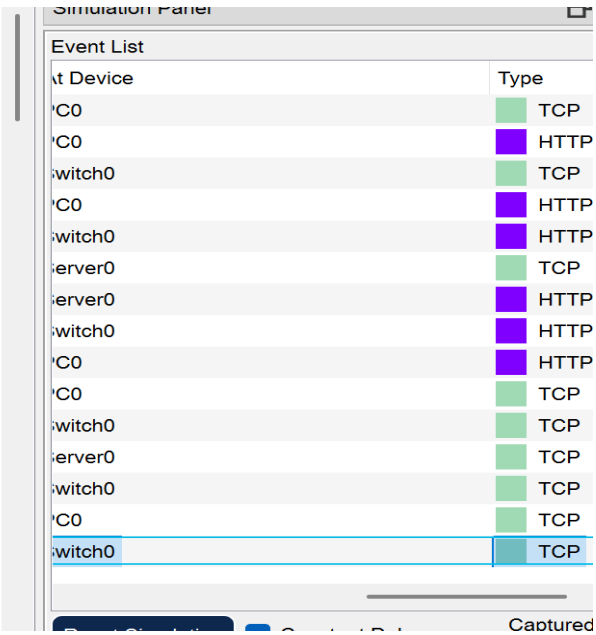


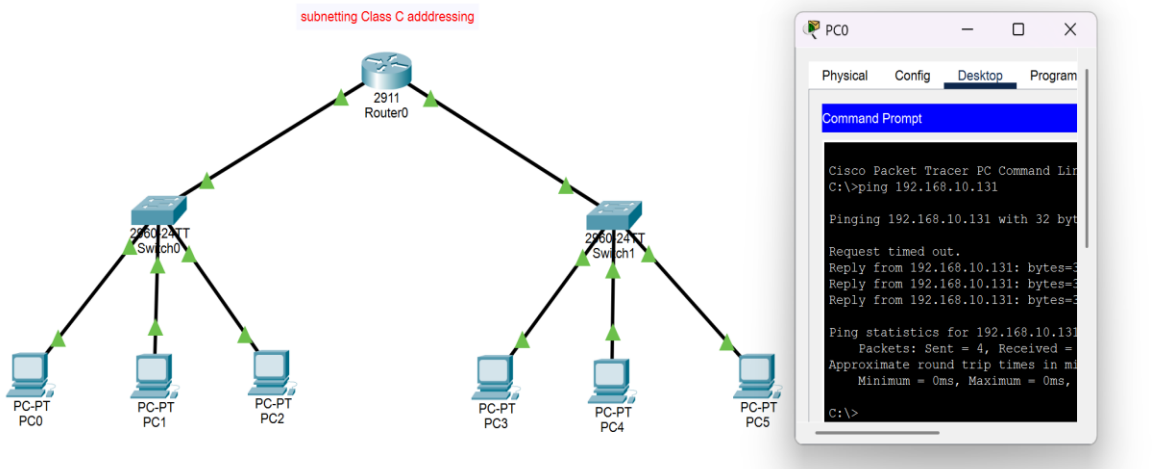

COMPUTER LAB
Server-PT
DHCO_SERVER
2960-24TT
Switch1
PC-PT
PC33
PC-PT
PC33(1)
PC-PT
PC33(2)
PC-PT
PC33(3)
PC-PT
PC34
PC-PT
PC34(1)
PC-PT
PC34(2)
PC-PT
PC34(3)
PC-PT
PC35
PC-PT
PC35(1)
PC-PT
PC35(2)
PC-PT
PC35(3)
PC-PT
PC36
PC-PT
PC36(1)
PC-PT
PC36(2)
PC-PT
PC36(3)
PC-PT
PC37
PC-PT
PC37(1)
PC-PT
PC37(2)
PC-PT
PC37(3)

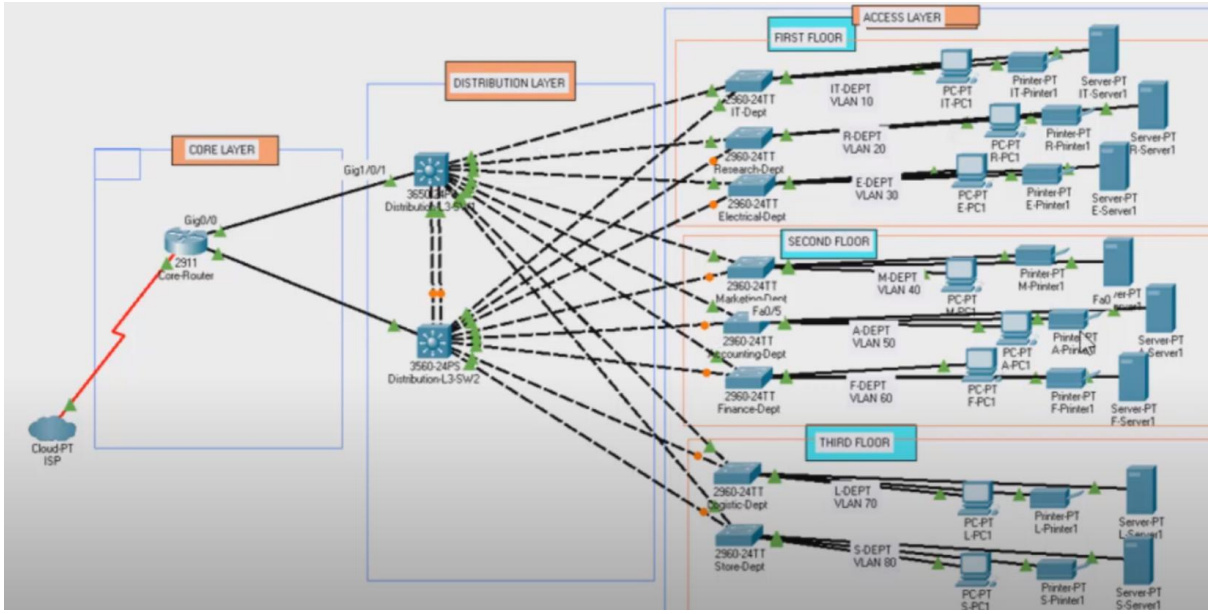

STATIC ROUTING
1941
Router0
1941
Router1
2950-24
Switch0
2950-24
Switch1
PC-PT
PC0
PC-PT
PC1
PC-PT
PC2
PC-PT
PC3

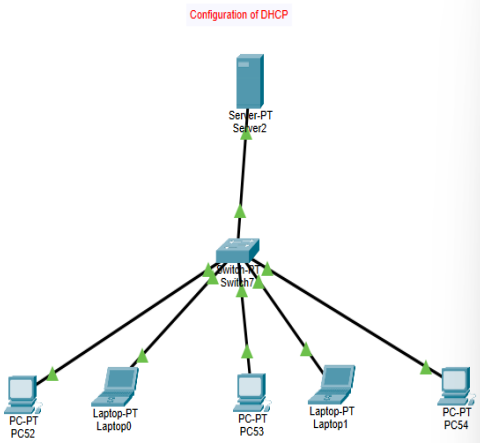

PC0
Physical
Config
Desktop
Programming
Attributes
Command Prompt
Cisco Packet Tracer PC Command Line 1.0
C:\>ping 192.168.3.5
Pinging 192.168.3.5 with 32 bytes of data:
Request timed out.
Request timed out.
Request timed out.
Request timed out.
Ping statistics for 192.168.3.5:
Packets: Sent = 4, Received = 0, Lost = 4 (100%
loss),
C:\>

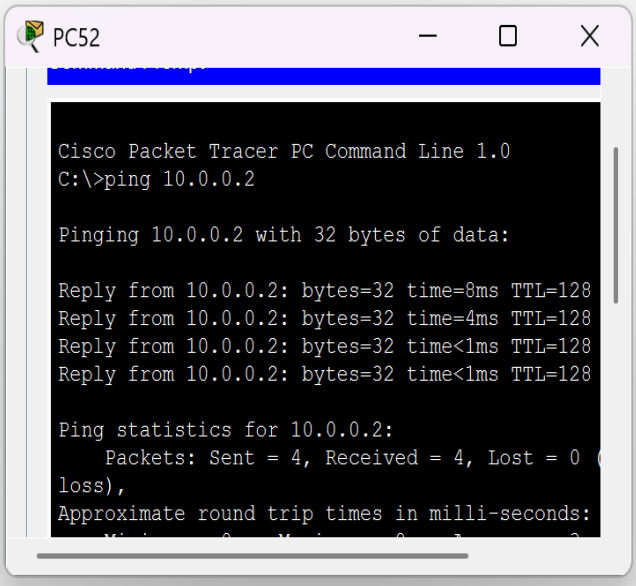

TCP SIMULATION
1941
Router0
2960-24TT
Switch0
Server-PT
Server0
PC-PT
PC0

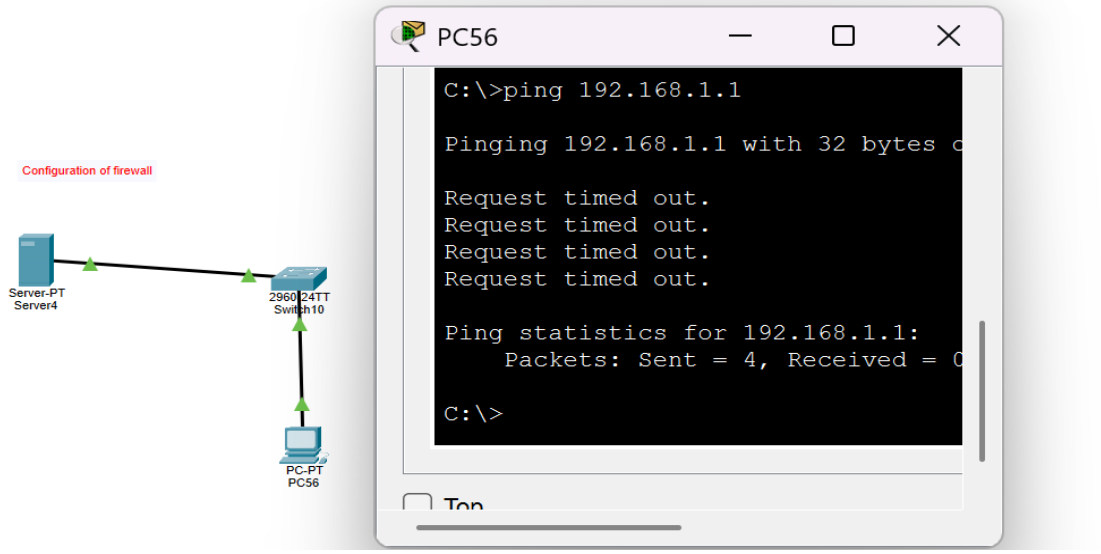

Event List
t Device
Type
C0
TCP
C0
HTTP
witch0
TCP
C0
HTTP
witch0
HTTP
erver0
TCP
erver0
HTTP
witch0
HTTP
C0
HTTP
C0
TCP
witch0
TCP
erver0
TCP
witch0
TCP
C0
TCP
witch0
TCP
Captured

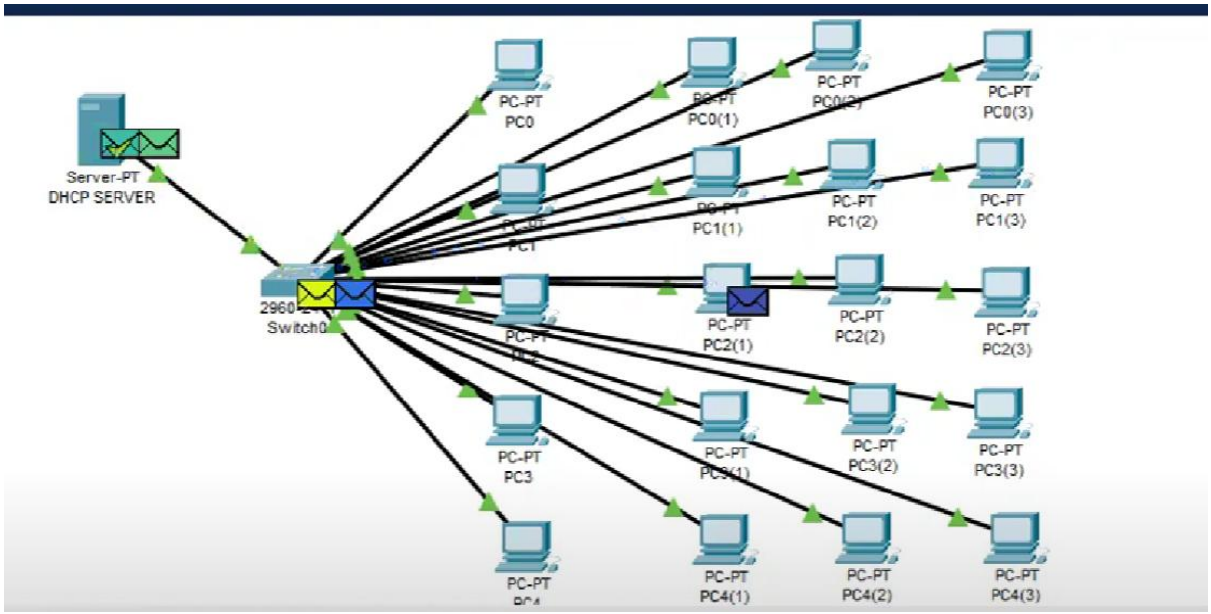

subnetting Class C adddressing
2911
Router0
2960-24TT
Switch0
2960-24TT
Switch1
PC-PT
PC0
PC-PT
PC1
PC-PT
PC2
PC-PT
PC3
PC-PT
PC4
PC-PT
PC5
PC0
Physical
Config
Desktop
Program
Command Prompt
Cisco Packet Tracer PC Command Lin
C:\>ping 192.168.10.131
Pinging 192.168.10.131 with 32 byt
Request timed out.
Reply from 192.168.10.131: bytes=
Reply from 192.168.10.131: bytes=
Reply from 192.168.10.131: bytes=
Ping statistics for 192.168.10.13
Packets: Sent = 4, Received =
Approximate round trip times in mi
Minimum = 0ms, Maximum = 0ms,
C:\>

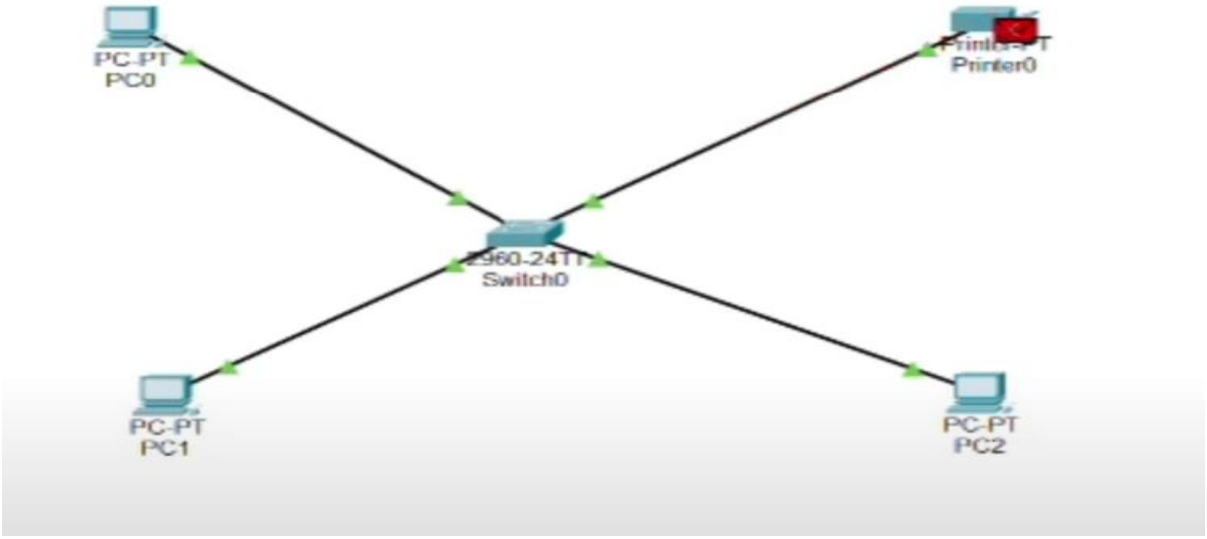

ACCESS LAYER
FIRST FLOOR
DISTRIBUTION LAYER
CORE LAYER
SECOND FLOOR
THIRD FLOOR
2911
Core-Router
Cloud-PT
ISP
3560-24PS
Distribution-L3-SW2
IT-DEPT
VLAN 10
R-DEPT
VLAN 20
E-DEPT
VLAN 30
M-DEPT
VLAN 40
A-DEPT
VLAN 50
F-DEPT
VLAN 60
L-DEPT
VLAN 70
S-DEPT
VLAN 80

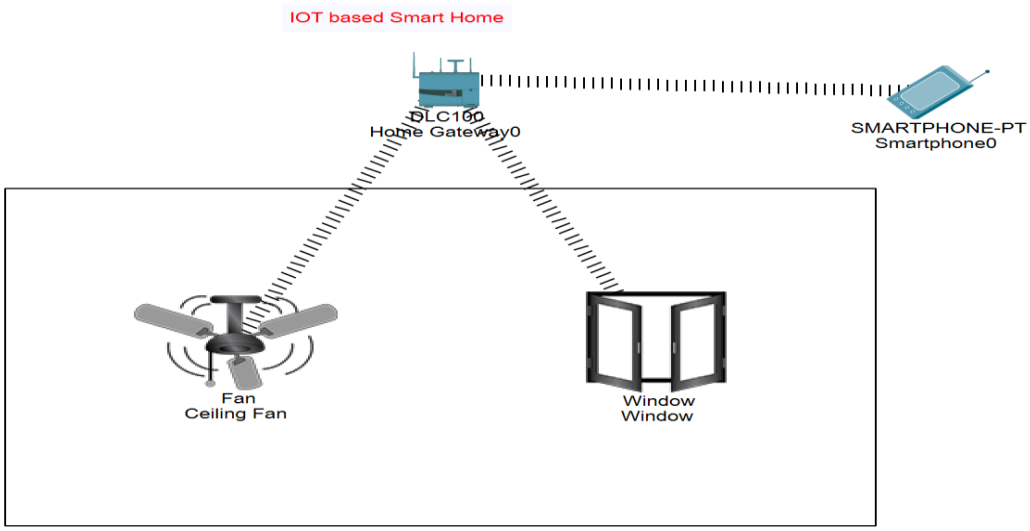

Configuration of DHCP
Server-PT
Server2
Switch-PT
Switch7
PC-PT
PC52
Laptop-PT
Laptop0
PC-PT
PC53
Laptop-PT
Laptop1
PC-PT
PC54

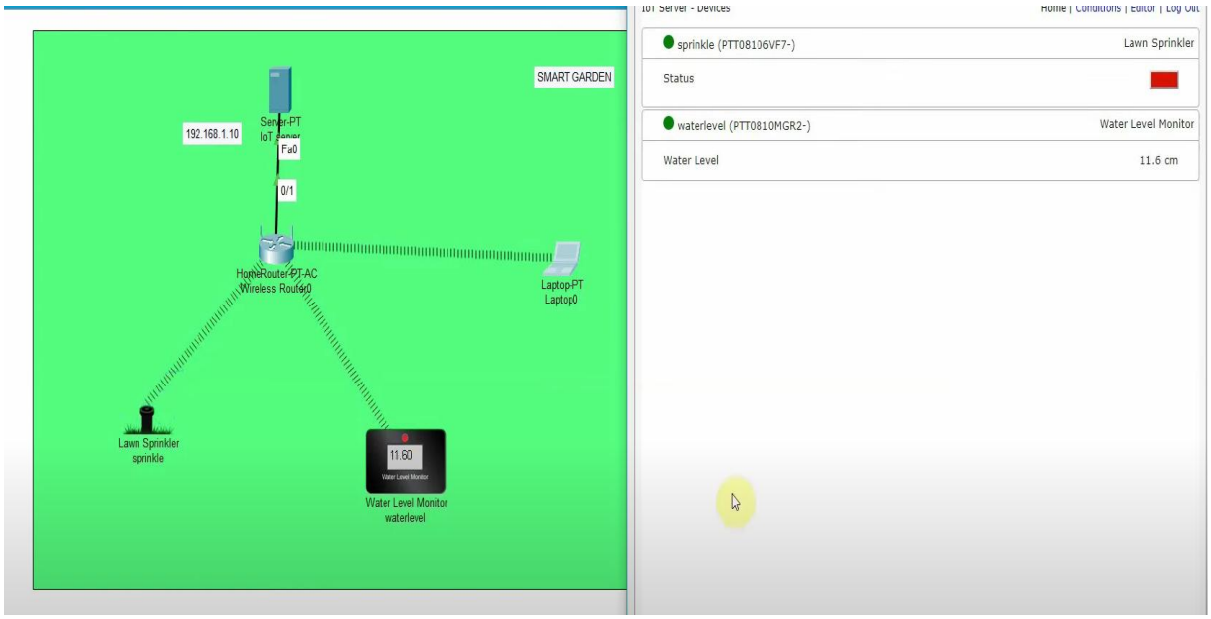

PC52
Cisco Packet Tracer PC Command Line 1.0
C:\>ping 10.0.0.2
Pinging 10.0.0.2 with 32 bytes of data:
Reply from 10.0.0.2: bytes=32 time=8ms TTL=128
Reply from 10.0.0.2: bytes=32 time=4ms TTL=128
Reply from 10.0.0.2: bytes=32 time<1ms TTL=128
Reply from 10.0.0.2: bytes=32 time<1ms TTL=128
Ping statistics for 10.0.0.2:
Packets: Sent = 4, Received = 4, Lost = 0
loss),
Approximate round trip times in milli-seconds:

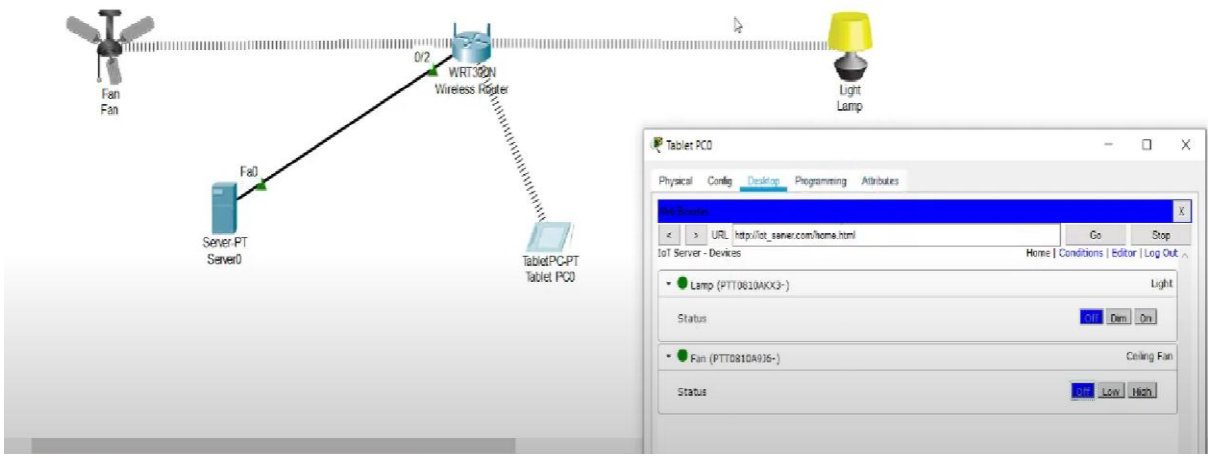

Configuration of firewall
Server-PT
Server4
2960-24TT
Switch10
PC-PT
PC56
PC56
C:\>ping 192.168.1.1
Pinging 192.168.1.1 with 32 bytes
Request timed out.
Request timed out.
Request timed out.
Request timed out.
Ping statistics for 192.168.1.1:
Packets: Sent = 4, Received =
C:\>

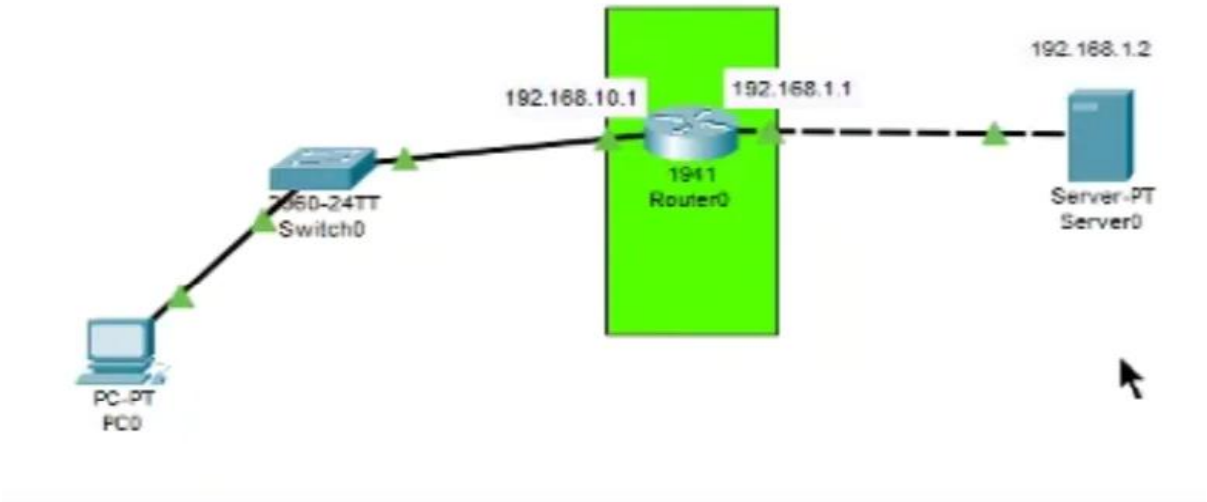

Server-PT
DHCP SERVER
2960-24TT
Switch0
PC-PT
PC0
PC-PT
PC0(1)
PC-PT
PC0(2)
PC-PT
PC0(3)
PC-PT
PC1
PC-PT
PC1(1)
PC-PT
PC1(2)
PC-PT
PC1(3)
PC-PT
PC2
PC-PT
PC2(1)
PC-PT
PC2(2)
PC-PT
PC2(3)
PC-PT
PC3
PC-PT
PC3(1)
PC-PT
PC3(2)
PC-PT
PC3(3)
PC-PT
PC4
PC-PT
PC4(1)
PC-PT
PC4(2)
PC-PT
PC4(3)


PC-PT
PC0
Printer0
2960-24TT
Switch0
PC-PT
PC1
PC-PT
PC2


IOT based Smart Home
DLC100
Home Gateway0
SMARTPHONE-PT
Smartphone0
Fan
Ceiling Fan
Window
Window


SMART GARDEN
192.168.1.10
Server-PT
IoT Server
Fa0
0/1
HomeRouter-PT-AC
Wireless Router0
Laptop-PT
Laptop0
Lawn Sprinkler
sprinkle
11.60
Water Level Monitor
waterlevel
sprinkle (PTT08106VF7-)
Lawn Sprinkler
Status
waterlevel (PTT0810MGR2-)
Water Level Monitor
Water Level
11.6 cm


Fan
Fan
0/2
WRT300N
Wireless Router
Light
Lamp
Fa0
Server-PT
Server0
TabletPC-PT
Tablet PC0
Tablet PC0
Physical
Config
Desktop
Programming
Attributes
URL
http://iot_server.com/home.html
Go
Stop
IoT Server - Devices
Home | Conditions | Editor | Log Out
Lamp (PTT0810AKK3-)
Light
Status
Off
Dim
On
Fan (PTT0810A9J6-)
Ceiling Fan
Status
Off
Low
High



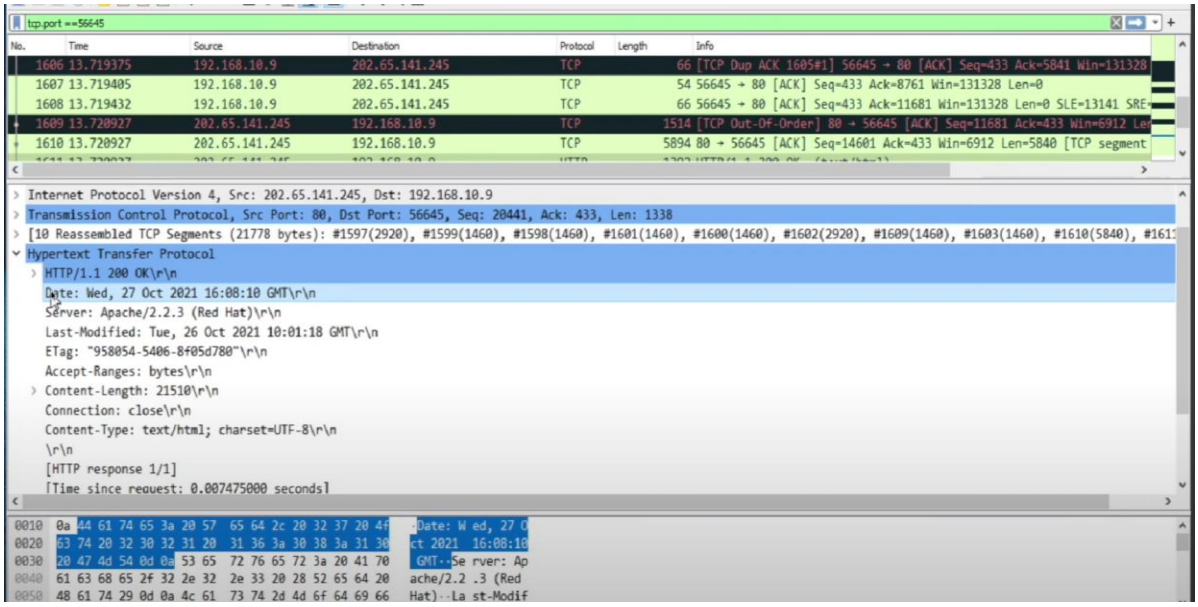
No.	Time	Source	Destination	Protocol	Length	Info
125	5.580331	192.168.3.153	146.66.71.198	TCP	66	33572 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
154	5.645496	146.66.71.198	192.168.3.153	TCP	66	80 → 33572 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PE
155	5.645569	192.168.3.153	146.66.71.198	TCP	54	33572 → 80 [ACK] Seq=1 Ack=1 Win=65536 Len=0
386	6.563605	192.168.3.153	146.66.71.198	HTTP	635	GET / HTTP/1.1
418	6.626732	146.66.71.198	192.168.3.153	TCP	54	80 → 33572 [ACK] Seq=1 Ack=582 Win=30464 Len=0
429	7.036925	146.66.71.198	192.168.3.153	TCP	1514	80 → 33572 [ACK] Seq=1 Ack=582 Win=30464 Len=1460 [TCP segment of
430	7.036935	146.66.71.198	192.168.3.153	TCP	1514	80 → 33572 [ACK] Seq=1461 Ack=582 Win=30464 Len=1460 [TCP segment
431	7.037267	192.168.3.153	146.66.71.198	TCP	54	33572 → 80 [ACK] Seq=582 Ack=2921 Win=65536 Len=0
432	7.037726	146.66.71.198	192.168.3.153	TCP	1514	80 → 33572 [ACK] Seq=2921 Ack=582 Win=30464 Len=1460 [TCP segment
433	7.037734	146.66.71.198	192.168.3.153	TCP	1514	80 → 33572 [ACK] Seq=4381 Ack=582 Win=30464 Len=1460 [TCP segment
434	7.037736	146.66.71.198	192.168.3.153	TCP	1514	80 → 33572 [ACK] Seq=5841 Ack=582 Win=30464 Len=1460 [TCP segment
435	7.037739	146.66.71.198	192.168.3.153	TCP	1514	80 → 33572 [ACK] Seq=7301 Ack=582 Win=30464 Len=1460 [TCP segment
436	7.037741	146.66.71.198	192.168.3.153	TCP	1514	80 → 33572 [ACK] Seq=8761 Ack=582 Win=30464 Len=1460 [TCP segment
437	7.037744	146.66.71.198	192.168.3.153	TCP	1514	80 → 33572 [ACK] Seq=10221 Ack=582 Win=30464 Len=1460 [TCP segment
438	7.037747	146.66.71.198	192.168.3.153	TCP	1514	80 → 33572 [ACK] Seq=11681 Ack=582 Win=30464 Len=1460 [TCP segment
439	7.037750	146.66.71.198	192.168.3.153	TCP	1514	80 → 33572 [ACK] Seq=13141 Ack=582 Win=30464 Len=1460 [TCP segment
440	7.038214	192.168.3.153	146.66.71.198	TCP	54	33572 → 80 [ACK] Seq=582 Ack=14601 Win=65536 Len=0
450	7.098733	146.66.71.198	192.168.3.153	TCP	1514	80 → 33572 [ACK] Seq=14601 Ack=582 Win=30464 Len=1460 [TCP segment

```
> Frame 450: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface 0
> Ethernet II, Src: Rosewill_12:2b:0f (68:1c:a2:12:2b:0f), Dst: IntelCor_42:70:89 (48:f1:7f:42:70:89)
> Internet Protocol Version 4, Src: 146.66.71.198, Dst: 192.168.3.153
v Transmission Control Protocol, Src Port: 80, Dst Port: 33572, Seq: 14601, Ack: 582, Len: 1460
    Source Port: 80
    Destination Port: 33572
```

No.	Time	Source	Destination	Protocol	Length	Info
→ 33	29.633524	192.168.43.241	142.104.75.58	ICMP	74	Echo (ping) request id=0x0001, seq=289/8449, ttl=128 (reply …
← 34	30.249428	142.104.75.58	192.168.43.241	ICMP	74	Echo (ping) reply id=0x0001, seq=289/8449, ttl=41 (request…
40	30.641976	192.168.43.241	142.104.75.58	ICMP	74	Echo (ping) request id=0x0001, seq=290/8705, ttl=128 (reply …
46	31.209199	142.104.75.58	192.168.43.241	ICMP	74	Echo (ping) reply id=0x0001, seq=290/8705, ttl=41 (request…
47	31.649021	192.168.43.241	142.104.75.58	ICMP	74	Echo (ping) request id=0x0001, seq=291/8961, ttl=128 (reply …
49	32.169936	142.104.75.58	192.168.43.241	ICMP	74	Echo (ping) reply id=0x0001, seq=291/8961, ttl=41 (request…
50	32.654228	192.168.43.241	142.104.75.58	ICMP	74	Echo (ping) request id=0x0001, seq=292/9217, ttl=128 (reply …
51	33.129417	142.104.75.58	192.168.43.241	ICMP	74	Echo (ping) reply id=0x0001, seq=292/9217, ttl=41 (request…
52	33.682669	192.168.43.241	142.104.75.58	ICMP	74	Echo (ping) request id=0x0001, seq=293/9473, ttl=128 (reply …
53	34.090728	142.104.75.58	192.168.43.241	ICMP	74	Echo (ping) reply id=0x0001, seq=293/9473, ttl=41 (request…
54	34.695329	192.168.43.241	142.104.75.58	ICMP	74	Echo (ping) request id=0x0001, seq=294/9729, ttl=128 (reply …
55	35.054650	142.104.75.58	192.168.43.241	ICMP	74	Echo (ping) reply id=0x0001, seq=294/9729, ttl=41 (request…

```
v Frame 34: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface \Device\NPF_{AAF6E947-B49B-45AE-9916-FE3CA0C32315}, id 0
  > Interface id: 0 (\Device\NPF_{AAF6E947-B49B-45AE-9916-FE3CA0C32315})
    Encapsulation type: Ethernet (1)
    Arrival Time: Apr 28, 2020 01:40:30.839977000 India Standard Time
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1588018230.839977000 seconds
    [Time delta from previous captured frame: 0.615904000 seconds]
    [Time delta from previous displayed frame: 0.615904000 seconds]
    [Time since reference or first frame: 30.249428000 seconds]
    Frame Number: 34
    Frame Length: 74 bytes (592 bits)
    Capture Length: 74 bytes (592 bits)
    [Frame is marked: False]
```

```
0000  d0 c5 d3 3c 97 91 64 db  43 d8 fc 58 08 00 45 20   ···<··d· C··X··E
0010  00 3c 68 b7 00 00 29 01  62 ae 8e 68 4b 3a c0 a8   ·<h···)· b··hK:··
0020  2b f1 00 00 54 3a 00 01  01 21 61 62 63 64 65 66   +···T:·· ·!abcdef
0030  67 68 69 6a 6b 6c 6d 6e  6f 70 71 72 73 74 75 76   ghijklmn opqrstuv
0040  77 61 62 63 64 65 66 67  68 69                    wabcdefg hi
```




```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <unistd.h>
#include <arpa/inet.h>

#define PORT 5000

int main() {
    int server_fd, client_fd;
    struct sockaddr_in server_addr, client_addr;
    socklen_t addr_size;
    char buffer[256];
    time_t t;
    struct tm *time_info;

    server_fd = socket(AF_INET, SOCK_STREAM, 0);
    if (server_fd == -1) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = INADDR_ANY;
    server_addr.sin_port = htons(PORT);
```

```
Server is listening on port 5000...
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <netdb.h>

#define PORT 8080
#define BUFFER_SIZE 1024

void handle_dns_query(int sockfd, struct sockaddr_in
    *client_addr, socklen_t client_len) {
    char domain[BUFFER_SIZE];
    char response[BUFFER_SIZE];
    struct hostent *host_entry;

    // Receive domain name from client
    recvfrom(sockfd, domain, BUFFER_SIZE, 0, (struct sockaddr
        *)client_addr, &client_len);
    printf("Received domain request: %s\n", domain);

    // Resolve the domain name to an IP address
    host_entry = gethostbyname(domain);
    if (host_entry == NULL) {
```

```
DNS Server is running on port 8080...
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <netinet/in.h>

#define DNS_PORT 53
#define BUFFER_SIZE 512

// DNS Header structure
struct DNS_HEADER {
    unsigned short id;        // Identification number
    unsigned char rd : 1;     // Recursion desired
    unsigned char tc : 1;     // Truncated message
    unsigned char aa : 1;     // Authoritative answer
    unsigned char opcode : 4;// Purpose of message
    unsigned char qr : 1;     // Query/Response flag
    unsigned char rcode : 4; // Response code
    unsigned char cd : 1;     // Checking disabled
    unsigned char ad : 1;     // Authenticated data
    unsigned char z : 1;      // Reserved
    unsigned char ra : 1;     // Recursion available
    unsigned short q_count;   // Number of questions
    unsigned short ans_count;// Number of answers
    unsigned short auth_count; // Number of authority records
    unsigned short add_count; // Number of additional records
```

```
Usage: /tmp/uKm0Qut3S8/main.o <hostname> <DNS server IP>


=== Code Exited With Errors ===
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>

#define PORT 8080
#define BUFFER_SIZE 1024

int main() {
    int server_fd, new_socket;
    struct sockaddr_in server_addr, client_addr;
    socklen_t addr_len = sizeof(client_addr);
    char buffer[BUFFER_SIZE];

    // Create socket
    server_fd = socket(AF_INET, SOCK_STREAM, 0);
    if (server_fd == -1) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    // Configure server address structure
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = INADDR_ANY;
    server_addr.sin_port = htons(PORT);
```

```
Echo Server listening on port 8080...
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <pthread.h>

#define PORT 8080
#define MAX_CLIENTS 10
#define BUFFER_SIZE 1024

typedef struct {
    int socket;
    struct sockaddr_in address;
} Client;

Client clients[MAX_CLIENTS];
int client_count = 0;
pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;

void broadcast_message(char *message, int sender_socket) {
    pthread_mutex_lock(&lock);
    for (int i = 0; i < client_count; i++) {
        if (clients[i].socket != sender_socket) {
            send(clients[i].socket, message, strlen(message),
                0);
        }
```

```
Chat server listening on port 8080...
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <sys/ioctl.h>
#include <net/if.h>
#include <netinet/if_ether.h>
#include <netinet/ether.h>
#include <netpacket/packet.h>

#define BUFFER_SIZE 42  // ARP packet size

// Structure for ARP packet
struct arp_packet {
    struct ether_header eth_hdr;  // Ethernet header
    struct ether_arp arp_hdr;     // ARP header
};

// Function to get MAC address of the given interface
void get_mac_address(const char *iface, unsigned char *mac) {
    int sockfd;
    struct ifreq ifr;
```

```
Usage: /tmp/u4kMbRyfaV/main.o <interface> <target_ip>


=== Code Exited With Errors ===
```

```c
#include <stdio.h>
#include <string.h>

#define MAX_SIZE 100  // Maximum size for the bit sequence

// Function to perform Bit Stuffing
void bit_stuffing(int input[], int length, int stuffed[]) {
    int count = 0, j = 0;

    for (int i = 0; i < length; i++) {
        stuffed[j++] = input[i];

        if (input[i] == 1) {
            count++;
            if (count == 5) {  // Insert a 0 after five consecutive 1s
                stuffed[j++] = 0;
                count = 0;  // Reset count after stuffing
            }
        } else {
            count = 0;
        }
    }
}
```

```
Enter the number of bits: 8
Enter the bit sequence (0s and 1s only):
01001010
00101010
01010101
00001110
01001010
00011110
01110011
11001100

Stuffed Bit Sequence: 1001010 101010 1010101 1110 1001010 11110
    1110011 11001100 0 0 4210144 0 903878024 32764 0 0 -1735904544
    32140 0 0 -1736096128 32140 423972096 929612886 1 0 -1737658754
    32140 8388608 0 8064 196607 -1735904544 3 -1736034592 32140
    -2104929280 130207 -1735991851 32140 0 0 1 0 0 0 903883776 32764
    -1735905023 32140 -1739045056 32140 -1735903032 32140 -1735906608
    32140 -2107239214 130207 -1735904544 32140 -2105233031 130207 4 0
    0 0 0 0 255 255 0 -16777216 1096173906 1593853268 1651076191
    1634033507 1096173906 0 0 0 1593853268 1651076191 1634033507
    1601793138 96 0 0 0 0 0 0 0 791621423 791621423 791621423
    791621423 0 0 0 0
De-stuffed Bit Sequence: 1001010 101010 1010101 1110 1001010 11110
    1110011 11001100
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>

#define PORT 8080
#define BUFFER_SIZE 1024

void send_file(FILE *file, int socket) {
    char buffer[BUFFER_SIZE];

    while (fgets(buffer, BUFFER_SIZE, file) != NULL) {
        send(socket, buffer, strlen(buffer), 0);
        memset(buffer, 0, BUFFER_SIZE);
    }

    // Send a termination signal
    send(socket, "END", 3, 0);
}

int main() {
    int server_fd, new_socket;
    struct sockaddr_in server_addr, client_addr;
    socklen_t addr_size;
    char filename[BUFFER_SIZE];
```

```
Server is listening on port 8080...
```

```c
#include <stdio.h>
#include <string.h>

#define POLY "1101"  // Generator polynomial (CRC divisor)

// Function to perform XOR operation
void xorOperation(char dividend[], char divisor[], int start)
    {
    for (int i = 0; i < strlen(divisor); i++) {
        dividend[start + i] = (dividend[start + i] ==
            divisor[i]) ? '0' : '1';
    }
}

// Function to calculate CRC remainder
void calculateCRC(char data[], char divisor[], char
    remainder[]) {
    int dataLen = strlen(data);
    int divisorLen = strlen(divisor);
    char temp[20];

    // Copy data to temp for division
    strcpy(temp, data);
```

```
Enter binary data: 01010001
Encoded Data (Sent): 01010001000
Enter received data (with or without errors):
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>  // for sleep()
#include <time.h>    // for random number generation

#define WINDOW_SIZE 4  // Sliding window size
#define TOTAL_FRAMES 10 // Total frames to send

// Simulating frame transmission
void sendFrames(int startFrame, int windowSize) {
    printf("Sending frames: ");
    for (int i = startFrame; i < startFrame + windowSize && i
        < TOTAL_FRAMES; i++) {
        printf("[%d] ", i);
    }
    printf("\n");
}

// Simulating acknowledgment process
int receiveAcks(int startFrame, int windowSize) {
    int ackedFrames = windowSize;

    for (int i = startFrame; i < startFrame + windowSize && i
        < TOTAL_FRAMES; i++) {
        int lost = rand() % 10 < 2;  // Simulating 20%
            probability of lost ACK
```

```
Sliding Window Protocol Simulation (Go-Back-N ARQ)

Sending frames: [0] [1] [2] [3]
Acknowledgment received for frame [0]
Acknowledgment received for frame [1]
Acknowledgment received for frame [2]
Frame [3] lost! Retransmitting from here...

Sending frames: [3] [4] [5] [6]
Acknowledgment received for frame [3]
Acknowledgment received for frame [4]
Acknowledgment received for frame [5]
Acknowledgment received for frame [6]

Sending frames: [7] [8] [9]
Acknowledgment received for frame [7]
Acknowledgment received for frame [8]
Acknowledgment received for frame [9]

All frames transmitted successfully!


=== Code Execution Successful ===
```