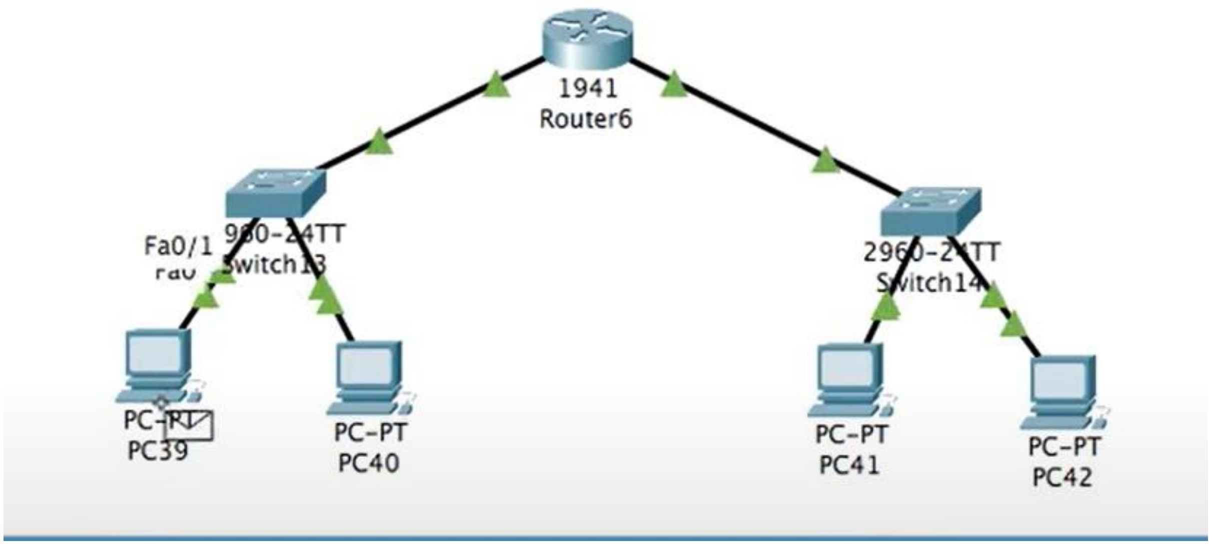
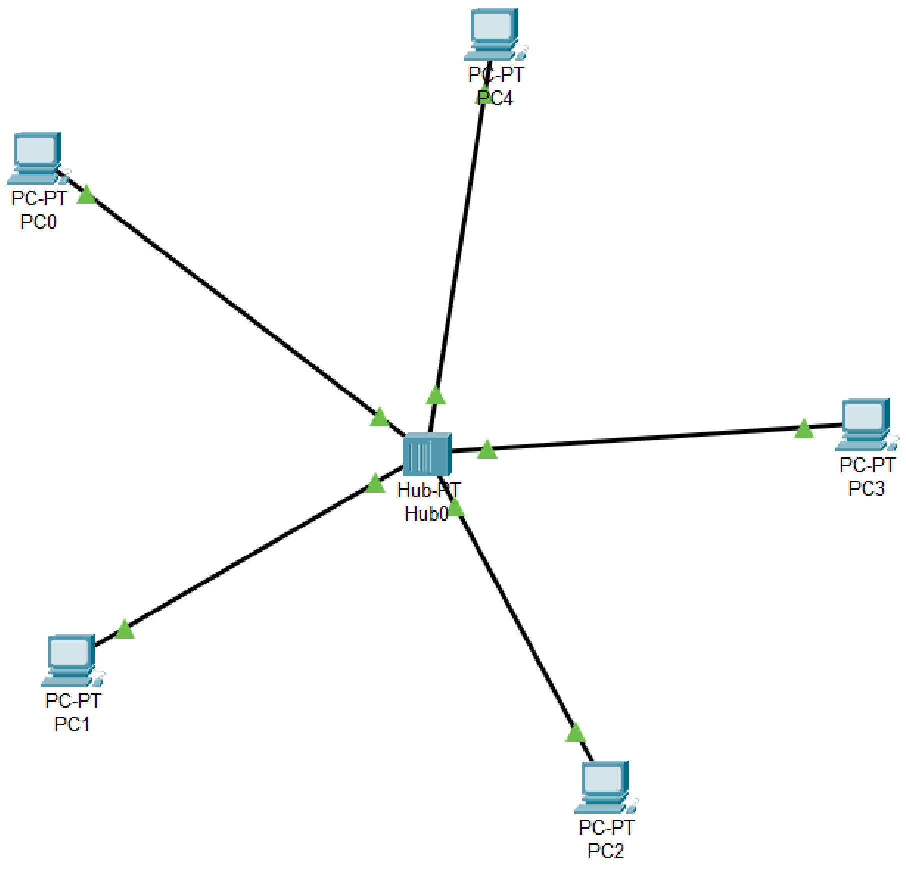
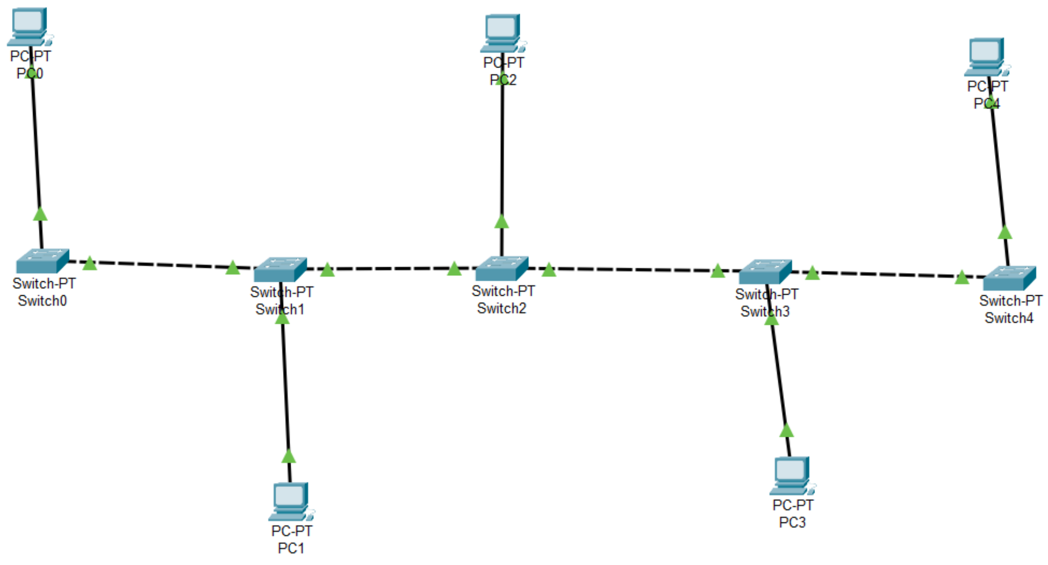
1. Configuration of Network Devices using Packet Tracer tools (Hub, Switch, Ethernet, Broadcast).



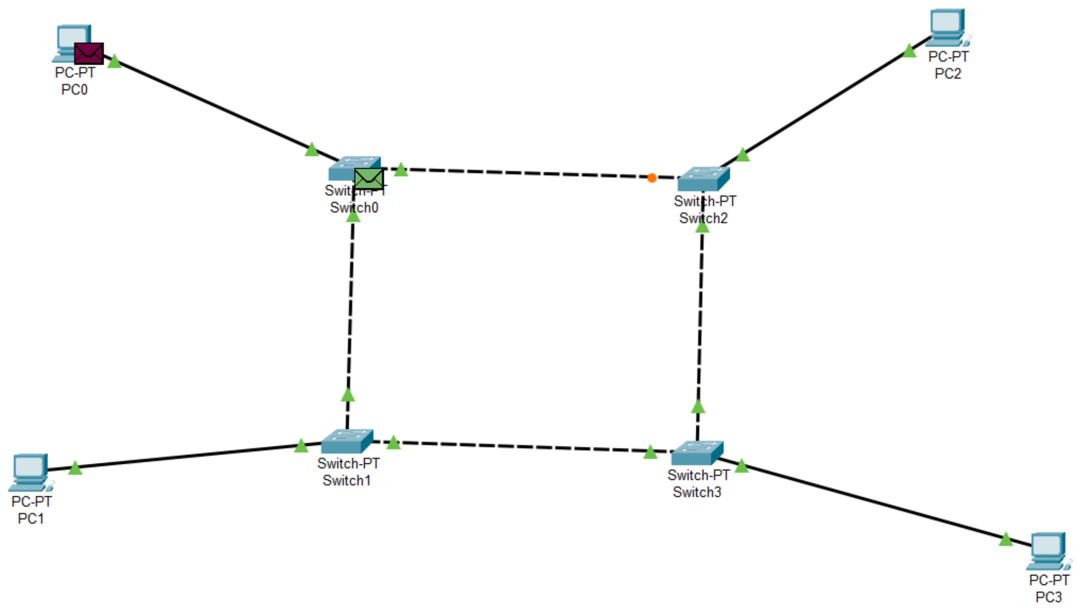
2. Design and Configuration of Star Topologies using Packet Tracer.



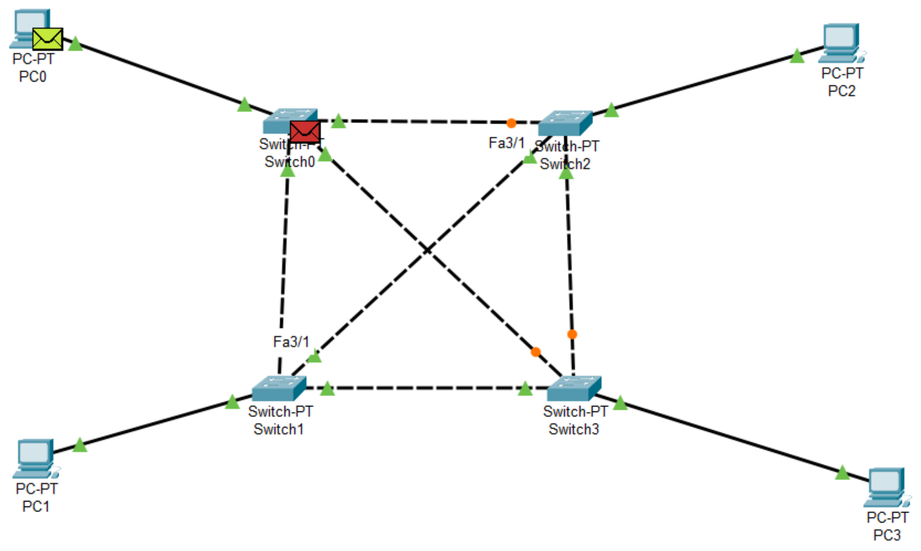
3. Design and Configuration of BUS Topologies using Packet Tracer.



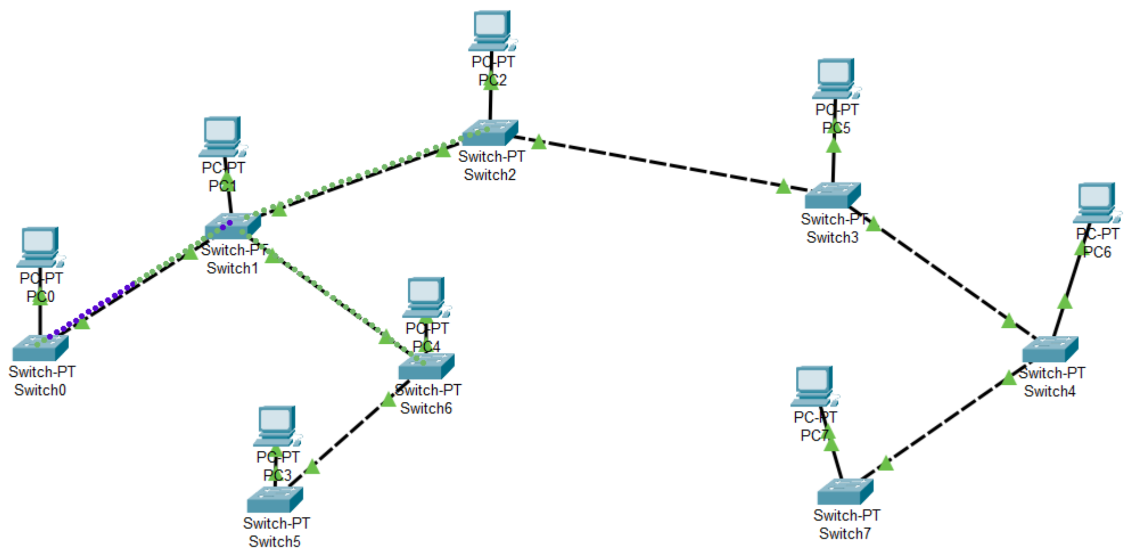
4. Design and Configuration of RING Topologies using Packet Tracer.



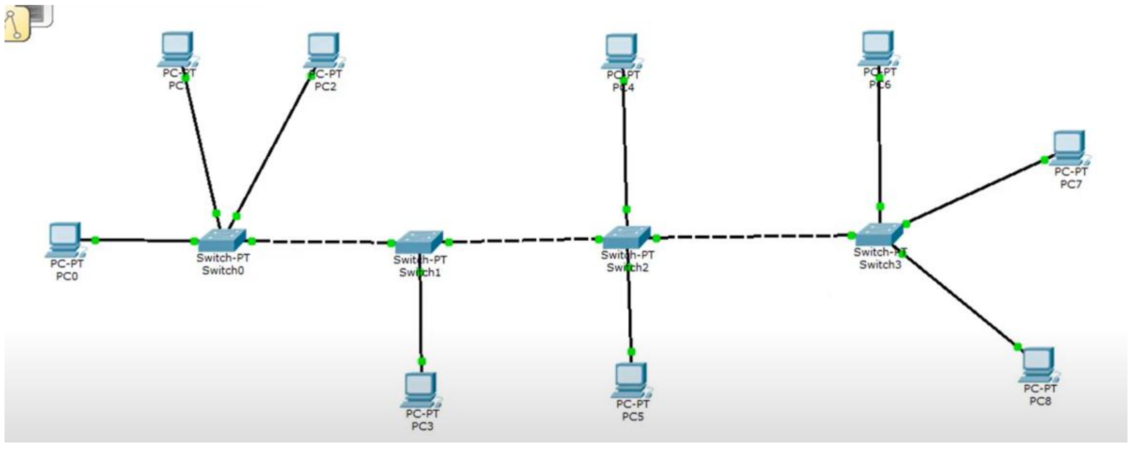
5. Design and Configuration of Mesh Topologies using Packet Tracer.



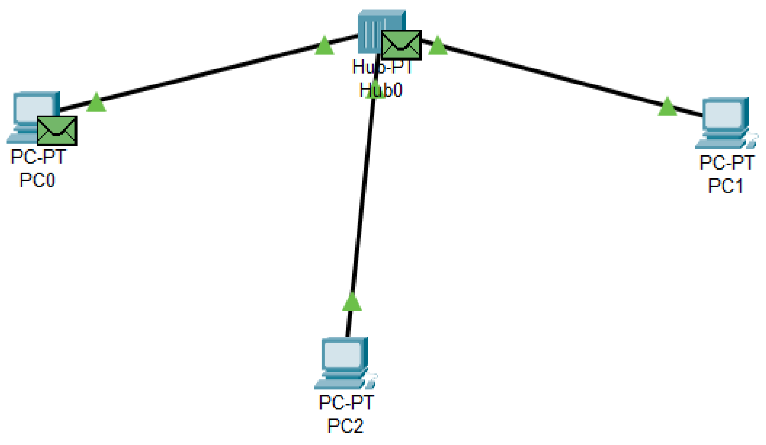
6. Design and Configuration of Tree Topologies using Packet Tracer.



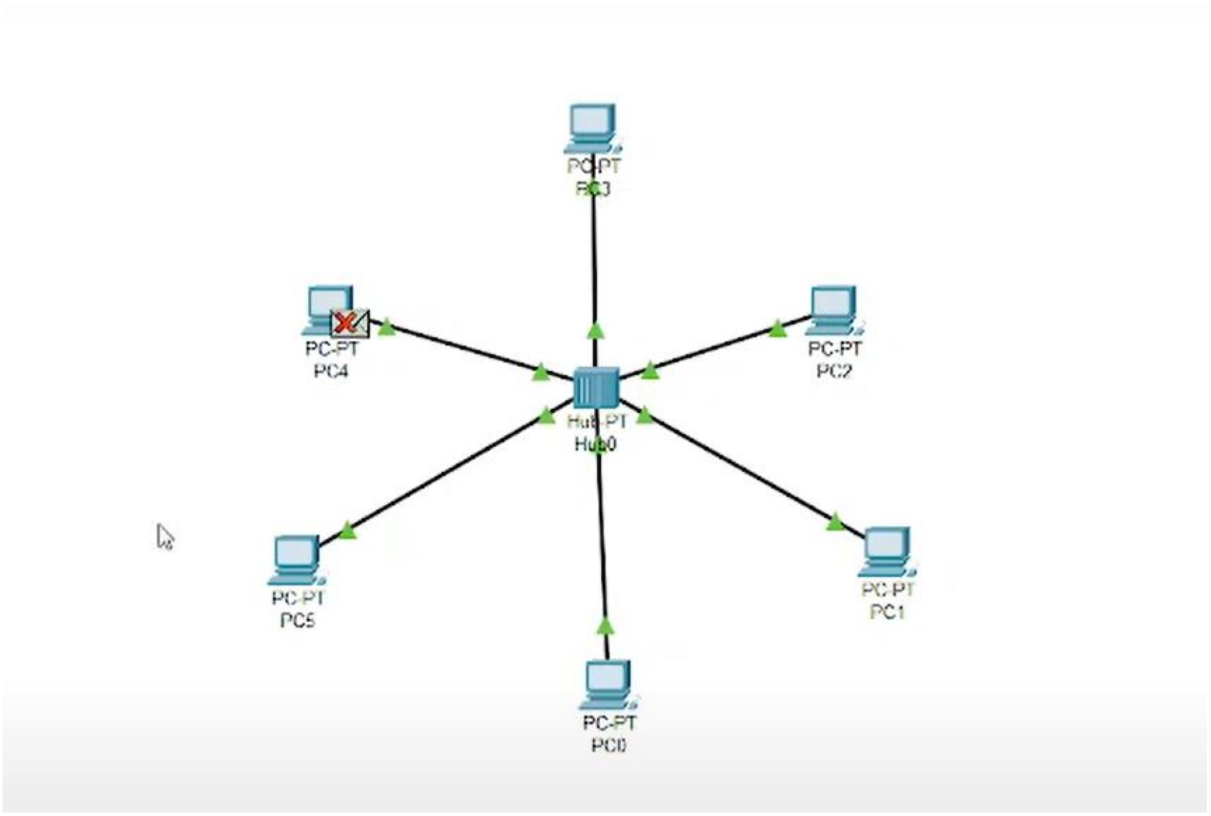
7.Design and Configuration of Hybrid Topologies using Packet Tracer.



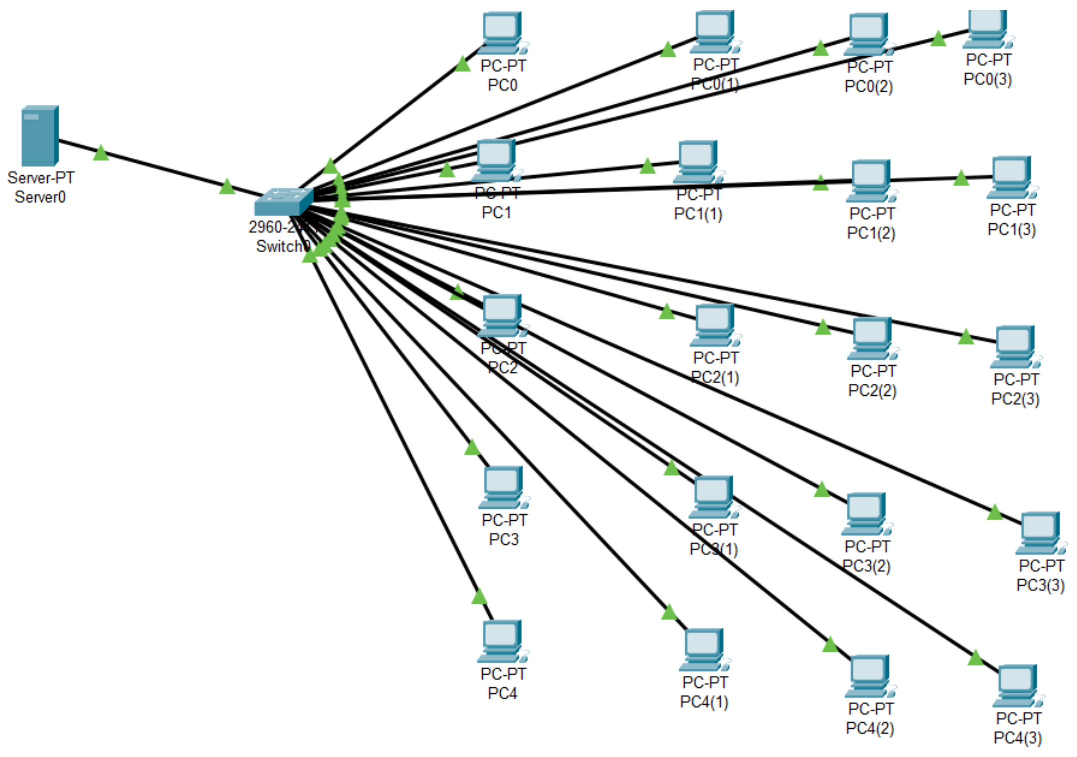
8.Data Link Layer Traffic Simulation using Packet Tracer Analysis of ARP.



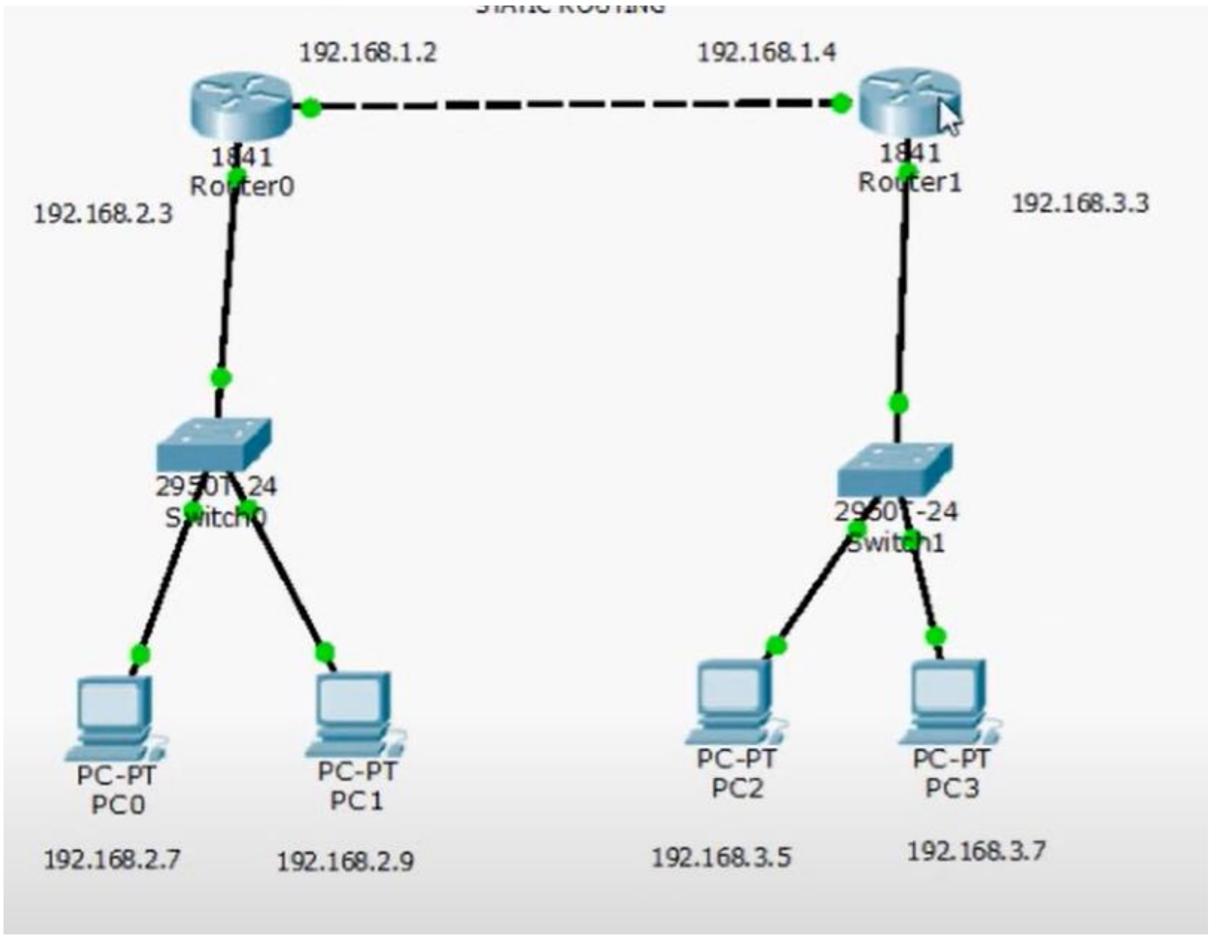
9.Data Link Layer Traffic Simulation using Packet Tracer Analysis of CSMA/CD & CSMA/CA.



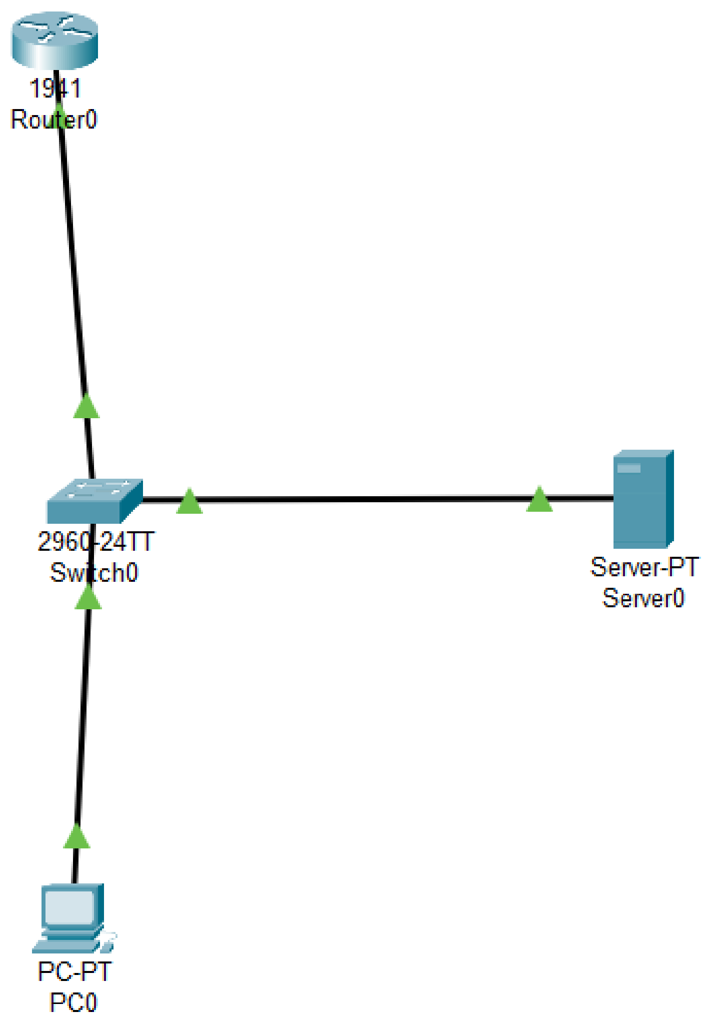
10. Making Computer Lab in Cisco Packet Tracer.



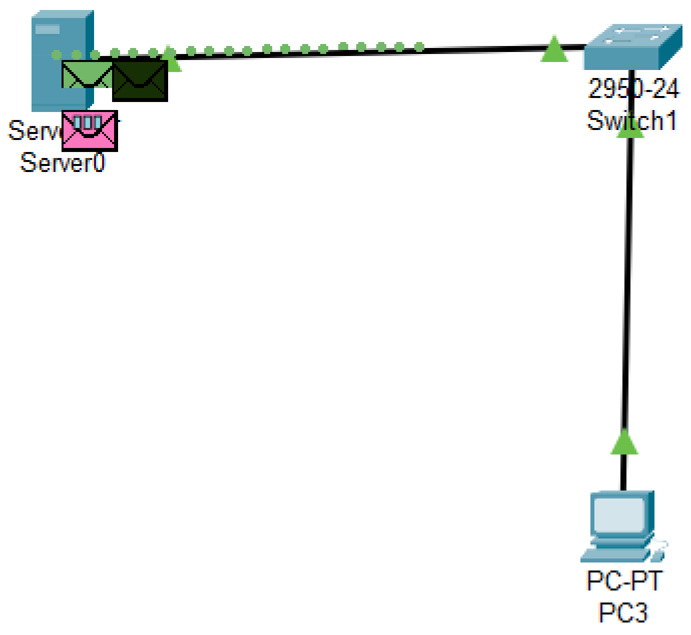
11. Designing two different network with Static Routing techniques using Packet Tracer.



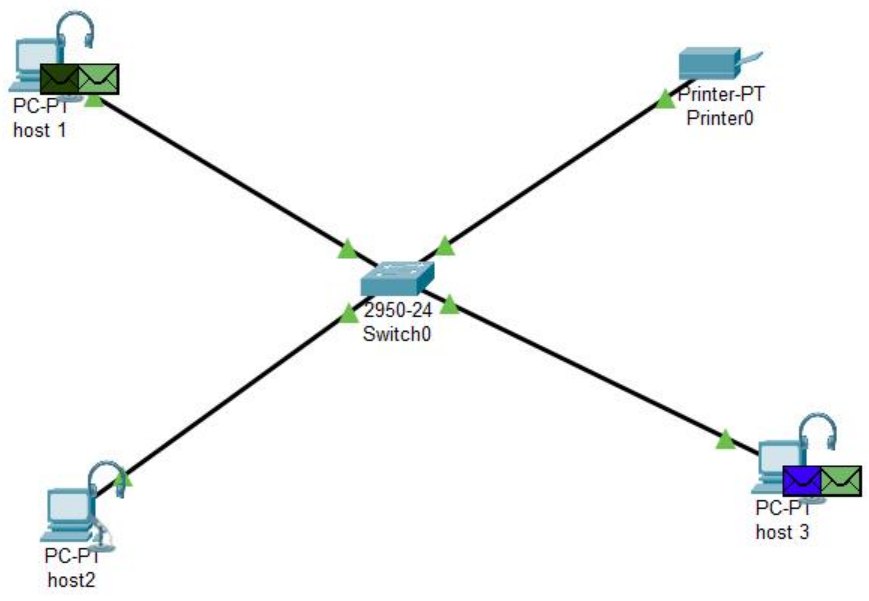
12. Design the Functionalities and Exploration of TCP using Packet Tracer.



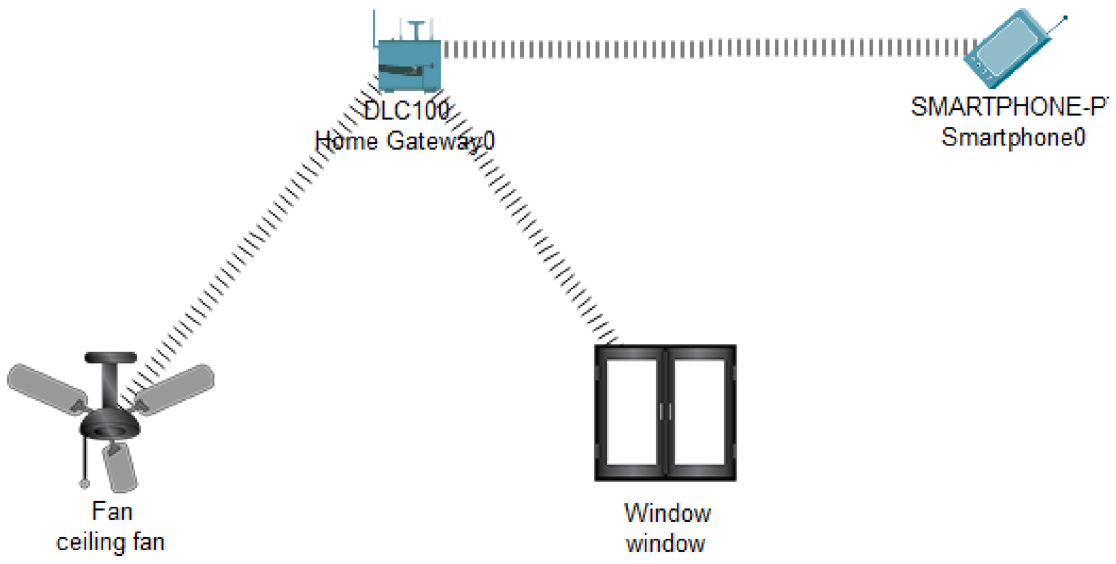
16. Configuration of firewall in packet tracer.



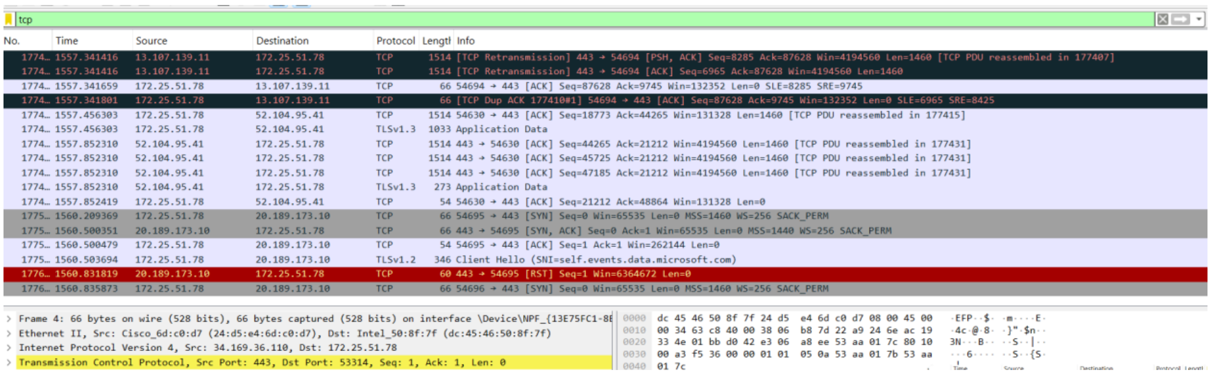
18. Simulate a Multimedia Network in Cisco Packet Tracer.



19. IoT based smart home applications.



23. Transport layer protocol header analysis using Wire shark- TCP and UDP.

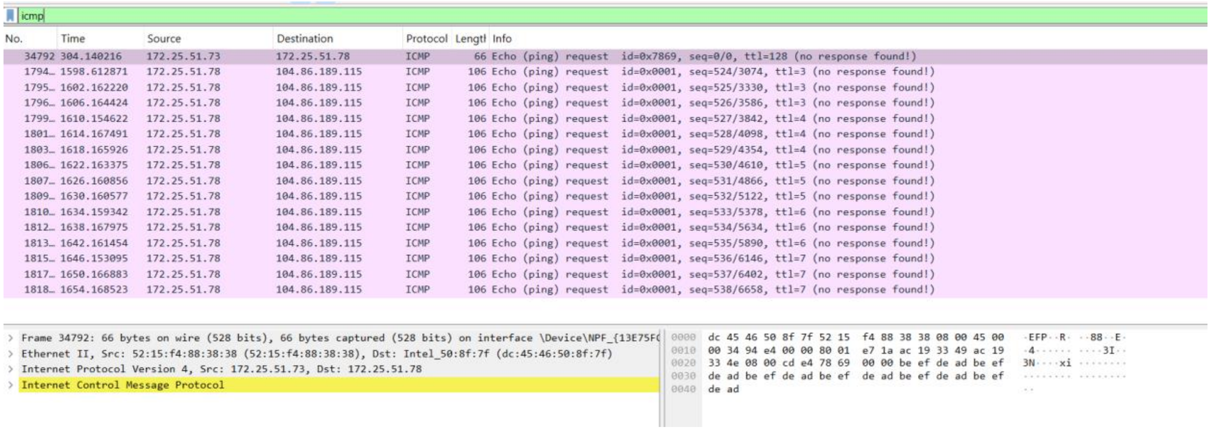


```
tcp
No.      Time          Source           Destination      Protocol Length Info
1774…    1557.341416   13.107.139.11    172.25.51.78     TCP      1514 [TCP Retransmission] 443 → 54694 [PSH, ACK] Seq=8285 Ack=87628 Win=4194560 Len=1460 [TCP PDU reassembled in 177407]
1774…    1557.341416   13.107.139.11    172.25.51.78     TCP      1514 [TCP Retransmission] 443 → 54694 [ACK] Seq=6965 Ack=87628 Win=4194560 Len=1460
1774…    1557.341659   172.25.51.78     13.107.139.11    TCP        66 54694 → 443 [ACK] Seq=87628 Ack=9745 Win=132352 Len=0 SLE=8285 SRE=9745
1774…    1557.341801   172.25.51.78     13.107.139.11    TCP        66 [TCP Dup ACK 177410#1] 54694 → 443 [ACK] Seq=87628 Ack=9745 Win=132352 Len=0 SLE=6965 SRE=8425
1774…    1557.456303   172.25.51.78     52.104.95.41     TCP      1514 54630 → 443 [ACK] Seq=18773 Ack=44265 Win=131328 Len=1460 [TCP PDU reassembled in 177415]
1774…    1557.456303   172.25.51.78     52.104.95.41     TLSv1.3  1033 Application Data
1774…    1557.852310   52.104.95.41     172.25.51.78     TCP      1514 443 → 54630 [ACK] Seq=44265 Ack=21212 Win=4194560 Len=1460 [TCP PDU reassembled in 177431]
1774…    1557.852310   52.104.95.41     172.25.51.78     TCP      1514 443 → 54630 [ACK] Seq=45725 Ack=21212 Win=4194560 Len=1460 [TCP PDU reassembled in 177431]
1774…    1557.852310   52.104.95.41     172.25.51.78     TCP      1514 443 → 54630 [ACK] Seq=47185 Ack=21212 Win=4194560 Len=1460 [TCP PDU reassembled in 177431]
1774…    1557.852310   52.104.95.41     172.25.51.78     TLSv1.3   273 Application Data
1774…    1557.852419   172.25.51.78     52.104.95.41     TCP        54 54630 → 443 [ACK] Seq=21212 Ack=48864 Win=131328 Len=0
1775…    1560.209369   172.25.51.78     20.189.173.10    TCP        66 54695 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM
1775…    1560.500351   20.189.173.10    172.25.51.78     TCP        66 443 → 54695 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1440 WS=256 SACK_PERM
1775…    1560.500479   172.25.51.78     20.189.173.10    TCP        54 54695 → 443 [ACK] Seq=1 Ack=1 Win=262144 Len=0
1775…    1560.503694   172.25.51.78     20.189.173.10    TLSv1.2   346 Client Hello (SNI=self.events.data.microsoft.com)
1776…    1560.831819   20.189.173.10    172.25.51.78     TCP        60 443 → 54695 [RST] Seq=1 Win=6364672 Len=0
1776…    1560.835873   172.25.51.78     20.189.173.10    TCP        66 54696 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM

> Frame 4: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface \Device\NPF_{13E75FC1-8E
> Ethernet II, Src: Cisco_6d:c0:d7 (24:d5:e4:6d:c0:d7), Dst: Intel_50:8f:7f (dc:45:46:50:8f:7f)
> Internet Protocol Version 4, Src: 34.169.36.110, Dst: 172.25.51.78
> Transmission Control Protocol, Src Port: 443, Dst Port: 53314, Seq: 1, Ack: 1, Len: 0

0000  dc 45 46 50 8f 7f 24 d5  e4 6d c0 d7 08 00 45 00   ·EFP··$· ·m····E·
0010  00 34 63 c8 40 00 38 06  b8 7d 22 a9 24 6e ac 19   ·4c·@·8· ·}"·$n··
0020  33 4e 01 bb d0 42 e3 06  a8 ee 53 aa 01 7c 80 10   3N···B·· ··S··|··
0030  00 a3 f5 36 00 00 01 01  05 0a 53 aa 01 7b 53 aa   ···6···· ··S··{S·
0040  01 7c
```

24. Network layer protocol header analysis using Wire shark – SMTP and ICMP.

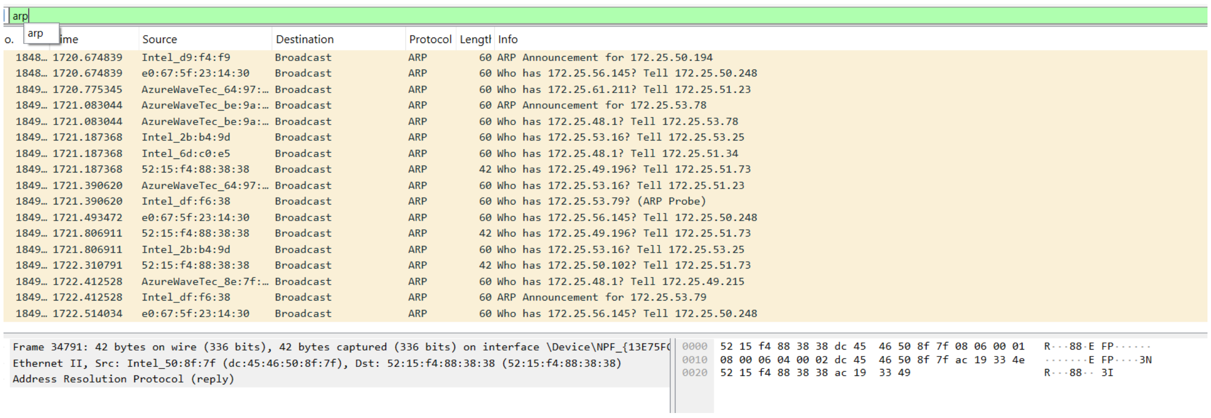


```
icmp
No.      Time          Source           Destination      Protocol Length Info
34792    304.140216    172.25.51.73     172.25.51.78     ICMP       66 Echo (ping) request  id=0x7869, seq=0/0, ttl=128 (no response found!)
1794…    1598.612871   172.25.51.78     104.86.189.115   ICMP      106 Echo (ping) request  id=0x0001, seq=524/3074, ttl=3 (no response found!)
1795…    1602.162220   172.25.51.78     104.86.189.115   ICMP      106 Echo (ping) request  id=0x0001, seq=525/3330, ttl=3 (no response found!)
1796…    1606.164424   172.25.51.78     104.86.189.115   ICMP      106 Echo (ping) request  id=0x0001, seq=526/3586, ttl=3 (no response found!)
1799…    1610.154622   172.25.51.78     104.86.189.115   ICMP      106 Echo (ping) request  id=0x0001, seq=527/3842, ttl=4 (no response found!)
1801…    1614.167491   172.25.51.78     104.86.189.115   ICMP      106 Echo (ping) request  id=0x0001, seq=528/4098, ttl=4 (no response found!)
1803…    1618.165926   172.25.51.78     104.86.189.115   ICMP      106 Echo (ping) request  id=0x0001, seq=529/4354, ttl=4 (no response found!)
1806…    1622.163375   172.25.51.78     104.86.189.115   ICMP      106 Echo (ping) request  id=0x0001, seq=530/4610, ttl=5 (no response found!)
1807…    1626.160856   172.25.51.78     104.86.189.115   ICMP      106 Echo (ping) request  id=0x0001, seq=531/4866, ttl=5 (no response found!)
1809…    1630.160577   172.25.51.78     104.86.189.115   ICMP      106 Echo (ping) request  id=0x0001, seq=532/5122, ttl=5 (no response found!)
1810…    1634.159342   172.25.51.78     104.86.189.115   ICMP      106 Echo (ping) request  id=0x0001, seq=533/5378, ttl=6 (no response found!)
1812…    1638.167975   172.25.51.78     104.86.189.115   ICMP      106 Echo (ping) request  id=0x0001, seq=534/5634, ttl=6 (no response found!)
1813…    1642.161454   172.25.51.78     104.86.189.115   ICMP      106 Echo (ping) request  id=0x0001, seq=535/5890, ttl=6 (no response found!)
1815…    1646.153095   172.25.51.78     104.86.189.115   ICMP      106 Echo (ping) request  id=0x0001, seq=536/6146, ttl=7 (no response found!)
1817…    1650.166883   172.25.51.78     104.86.189.115   ICMP      106 Echo (ping) request  id=0x0001, seq=537/6402, ttl=7 (no response found!)
1818…    1654.168523   172.25.51.78     104.86.189.115   ICMP      106 Echo (ping) request  id=0x0001, seq=538/6658, ttl=7 (no response found!)

> Frame 34792: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface \Device\NPF_{13E75F(
> Ethernet II, Src: 52:15:f4:88:38:38 (52:15:f4:88:38:38), Dst: Intel_50:8f:7f (dc:45:46:50:8f:7f)
> Internet Protocol Version 4, Src: 172.25.51.73, Dst: 172.25.51.78
> Internet Control Message Protocol

0000  dc 45 46 50 8f 7f 52 15  f4 88 38 38 08 00 45 00   ·EFP··R· ··88··E·
0010  00 34 94 e4 00 00 80 01  e7 1a ac 19 33 49 ac 19   ·4······ ····3I··
0020  33 4e 08 00 cd e4 78 69  00 00 be ef de ad be ef   3N····xi ········
0030  de ad be ef de ad be ef  de ad be ef de ad be ef   ········ ········
0040  de ad                                              ··
```

25. Network layer protocol header analysis using Wire shark – ARP and HTTP.



```
arp

o.    Time          Source                 Destination   Protocol  Length  Info
1848… 1720.674839   Intel_d9:f4:f9         Broadcast     ARP       60 ARP Announcement for 172.25.50.194
1848… 1720.674839   e0:67:5f:23:14:30      Broadcast     ARP       60 Who has 172.25.56.145? Tell 172.25.50.248
1849… 1720.775345   AzureWaveTec_64:97:…   Broadcast     ARP       60 Who has 172.25.61.211? Tell 172.25.51.23
1849… 1721.083044   AzureWaveTec_be:9a:…   Broadcast     ARP       60 ARP Announcement for 172.25.53.78
1849… 1721.083044   AzureWaveTec_be:9a:…   Broadcast     ARP       60 Who has 172.25.48.1? Tell 172.25.53.78
1849… 1721.187368   Intel_2b:b4:9d         Broadcast     ARP       60 Who has 172.25.53.16? Tell 172.25.53.25
1849… 1721.187368   Intel_6d:c0:e5         Broadcast     ARP       60 Who has 172.25.48.1? Tell 172.25.51.34
1849… 1721.187368   52:15:f4:88:38:38      Broadcast     ARP       42 Who has 172.25.49.196? Tell 172.25.51.73
1849… 1721.390620   AzureWaveTec_64:97:…   Broadcast     ARP       60 Who has 172.25.53.16? Tell 172.25.51.23
1849… 1721.390620   Intel_df:f6:38         Broadcast     ARP       60 Who has 172.25.53.79? (ARP Probe)
1849… 1721.493472   e0:67:5f:23:14:30      Broadcast     ARP       60 Who has 172.25.56.145? Tell 172.25.50.248
1849… 1721.806911   52:15:f4:88:38:38      Broadcast     ARP       42 Who has 172.25.49.196? Tell 172.25.51.73
1849… 1721.806911   Intel_2b:b4:9d         Broadcast     ARP       60 Who has 172.25.53.16? Tell 172.25.53.25
1849… 1722.310791   52:15:f4:88:38:38      Broadcast     ARP       42 Who has 172.25.50.102? Tell 172.25.51.73
1849… 1722.412528   AzureWaveTec_8e:7f:…   Broadcast     ARP       60 Who has 172.25.48.1? Tell 172.25.49.215
1849… 1722.412528   Intel_df:f6:38         Broadcast     ARP       60 ARP Announcement for 172.25.53.79
1849… 1722.514034   e0:67:5f:23:14:30      Broadcast     ARP       60 Who has 172.25.56.145? Tell 172.25.50.248

Frame 34791: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface \Device\NPF_{13E75F(
Ethernet II, Src: Intel_50:8f:7f (dc:45:46:50:8f:7f), Dst: 52:15:f4:88:38:38 (52:15:f4:88:38:38)
Address Resolution Protocol (reply)

0000  52 15 f4 88 38 38 dc 45  46 50 8f 7f 08 06 00 01   R···88·E FP······
0010  08 00 06 04 00 02 dc 45  46 50 8f 7f ac 19 33 4e   ·······E FP····3N
0020  52 15 f4 88 38 38 ac 19  33 49                     R···88·· 3I
```

26. Implementation of date and time display from client to server using TCP sockets in java/C.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <netinet/in.h>
#include <time.h>

#define PORT 8080

int main() {
  int server_fd, new_socket;
  struct sockaddr_in address;
  int addrlen = sizeof(address);
  char buffer[1024] = {0};

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

  // Define server address
  address.sin_family = AF_INET;
  address.sin_addr.s_addr = INADDR_ANY;
  address.sin_port = htons(PORT);

  // Bind socket to the address
```

```
    if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) < 0) {
        perror("Bind failed");
        close(server_fd);
        exit(EXIT_FAILURE);
    }

    // Listen for incoming connections
    if (listen(server_fd, 3) < 0) {
        perror("Listen failed");
        close(server_fd);
        exit(EXIT_FAILURE);
    }

    printf("Server listening on port %d\n", PORT);

    // Accept client connection
    if ((new_socket = accept(server_fd, (struct sockaddr *)&address,
                        (socklen_t*)&addrlen)) < 0) {
        perror("Accept failed");
        close(server_fd);
        exit(EXIT_FAILURE);
    }

    // Get current time
    time_t t;
    struct tm *tm_info;
    char time_str[50];

    time(&t);
    tm_info = localtime(&t);

    strftime(time_str, 50, "%Y-%m-%d %H:%M:%S", tm_info);

    // Send current time to client
    send(new_socket, time_str, strlen(time_str), 0);
    printf("Date and Time sent to client: %s\n", time_str);

    close(new_socket);
    close(server_fd);
    return 0;
}
```

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

#define PORT 8080

int main() {
  int sock = 0;
  struct sockaddr_in serv_addr;
  char buffer[1024] = {0};

  // Create socket
  if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
    printf("\nSocket creation error \n");
    return -1;
  }

  // Define server address
  serv_addr.sin_family = AF_INET;
  serv_addr.sin_port = htons(PORT);

  // Convert IPv4 and IPv6 addresses from text to binary form
  if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr) <= 0) {
    printf("\nInvalid address/ Address not supported \n");
    return -1;
  }

  // Connect to server
  if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
    printf("\nConnection Failed \n");
    return -1;
  }

  // Read the date and time from server
  read(sock, buffer, 1024);
  printf("Current Date and Time from Server: %s\n", buffer);

  close(sock);
  return 0;
}
```

27. Implementation of a DNS server and client in java/C using UDP sockets.

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

#define PORT 8080
#define BUFFER_SIZE 1024

// Define a simple DNS table (hardcoded domain-IP mappings)
struct DNS_Table {
  char domain[100];
  char ip[100];
} dns_table[] = {
  {"example.com", "93.184.216.34"},
  {"google.com", "142.250.190.14"},
  {"yahoo.com", "74.6.143.25"},
  {"localhost", "127.0.0.1"}
};

// Function to get IP address for a given domain
const char* get_ip_from_domain(const char* domain) {
  int i;
  for (i = 0; i < sizeof(dns_table) / sizeof(dns_table[0]); i++) {
    if (strcmp(domain, dns_table[i].domain) == 0) {
      return dns_table[i].ip;
    }
  }
  return "Domain not found";  // Return this if domain is not in table
}

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

  // Create a UDP socket
  if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
    perror("Socket creation failed");
    exit(EXIT_FAILURE);
  }

  // Server address configuration
  memset(&server_addr, 0, sizeof(server_addr));
  memset(&client_addr, 0, sizeof(client_addr));

  server_addr.sin_family = AF_INET;        // IPv4
  server_addr.sin_addr.s_addr = INADDR_ANY;  // Bind to any available IP
  server_addr.sin_port = htons(PORT);      // Server port

  // Bind the socket to the server address
  if (bind(sockfd, (const struct sockaddr *)&server_addr, sizeof(server_addr)) < 0)
{
    perror("Bind failed");
    close(sockfd);
    exit(EXIT_FAILURE);
  }

  printf("DNS Server is running...\n");

  while (1) {
    // Receive domain name from client
    int n = recvfrom(sockfd, buffer, BUFFER_SIZE, 0, (struct sockaddr
*)&client_addr, &addr_len);
    buffer[n] = '\0';  // Null-terminate the received string

    printf("Received domain name: %s\n", buffer);

    // Look up the IP address for the domain
    const char* ip_address = get_ip_from_domain(buffer);

    // Send IP address back to the client
    sendto(sockfd, ip_address, strlen(ip_address), 0, (const struct sockaddr
*)&client_addr, addr_len);
    printf("Sent IP address: %s\n", ip_address);
  }

  close(sockfd);
  return 0;
```

```
}
```
28. Developing a client that contacts a given DNS server to resolve a given hostname in java/C.

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

int main() {
  char hostname[256];
  printf("Enter hostname: ");
  scanf("%s", hostname);

  // Get host information
  struct hostent *host_info;
  host_info = gethostbyname(hostname);

  if (host_info == NULL) {
    printf("Error: Could not resolve hostname.\n");
    exit(1);
  }

  // Extract the IP address
  struct in_addr **addr_list = (struct in_addr **)host_info->h_addr_list;

  printf("IP addresses for %s:\n", hostname);
  for (int i = 0; addr_list[i] != NULL; i++) {
    printf("%s\n", inet_ntoa(*addr_list[i]));
  }

  return 0;
}
```
29. Creating the applications using TCP echo server and client in java/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 run_server() {
  int server_fd, client_socket;
  struct sockaddr_in address;
  char buffer[BUFFER_SIZE] = {0};
  socklen_t addr_len = sizeof(address);

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

  // Define server address
  address.sin_family = AF_INET;
  address.sin_addr.s_addr = INADDR_ANY;
  address.sin_port = htons(PORT);

  // Bind the socket to the address
  if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) < 0) {
    perror("Bind failed");
    close(server_fd);
    exit(EXIT_FAILURE);
  }

  // Listen for incoming connections
  if (listen(server_fd, 3) < 0) {
    perror("Listen failed");
    close(server_fd);
    exit(EXIT_FAILURE);
  }

  printf("Echo Server is listening on port %d...\n", PORT);

  // Accept a client connection
  if ((client_socket = accept(server_fd, (struct sockaddr *)&address, &addr_len))
< 0) {
    perror("Accept failed");
    close(server_fd);
    exit(EXIT_FAILURE);
  }

  // Read the message from the client
  int read_size = read(client_socket, buffer, BUFFER_SIZE);
  buffer[read_size] = '\0';  // Null-terminate the received string

  printf("Received message: %s\n", buffer);

  // Echo the message back to the client
  send(client_socket, buffer, strlen(buffer), 0);

  printf("Echoed message back to the client.\n");

  close(client_socket);
  close(server_fd);
}

void run_client() {
  int sock;
  struct sockaddr_in server_addr;
  char message[BUFFER_SIZE];
  char buffer[BUFFER_SIZE] = {0};

  // Create socket
  if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
    printf("Socket creation error\n");
    return;
  }

  server_addr.sin_family = AF_INET;
  server_addr.sin_port = htons(PORT);

  // Convert address to binary form
  if (inet_pton(AF_INET, "127.0.0.1", &server_addr.sin_addr) <= 0) {
    printf("Invalid address/ Address not supported\n");
    return;
  }

  // Connect to the server
  if (connect(sock, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0) {
    printf("Connection failed\n");
    return;
  }

  // Input message
  printf("Enter message to send: ");
  fgets(message, BUFFER_SIZE, stdin);
  message[strcspn(message, "\n")] = '\0';  // Remove trailing newline

  // Send message to the server
  send(sock, message, strlen(message), 0);

  // Receive the echoed message
  int valread = read(sock, buffer, BUFFER_SIZE);
  buffer[valread] = '\0';  // Null-terminate the string

  printf("Echoed message from server: %s\n", buffer);

  close(sock);
}

int main() {
  int choice;

  printf("Select mode: \n1. Server\n2. Client\n");
  scanf("%d", &choice);
  getchar();  // Consume the newline character after entering choice

  if (choice == 1) {
    run_server();
  } else if (choice == 2) {
    run_client();
  } else {
    printf("Invalid choice.\n");
  }

  return 0;
}
```

30. Creating the applications using TCP chat client and chat server in java/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

int client_sockets[MAX_CLIENTS];
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

void *handle_client(void *client_socket);

void run_server() {
  int server_fd, new_socket, i;
  struct sockaddr_in address;
  socklen_t addr_len = sizeof(address);

  // Initialize client sockets array
  for (i = 0; i < MAX_CLIENTS; i++) {
    client_sockets[i] = 0;
  }

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

  // Define server address
  address.sin_family = AF_INET;
  address.sin_addr.s_addr = INADDR_ANY;
  address.sin_port = htons(PORT);

  // Bind the socket to the address
  if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) < 0) {
    perror("Bind failed");
    close(server_fd);
    exit(EXIT_FAILURE);
  }
```

```
    // Listen for incoming connections
    if (listen(server_fd, 3) < 0) {
        perror("Listen failed");
        close(server_fd);
        exit(EXIT_FAILURE);
    }

    printf("Chat server is running on port %d...\n", PORT);

    while (1) {
        // Accept new client connection
        if ((new_socket = accept(server_fd, (struct sockaddr *)&address, &addr_len))
< 0) {
            perror("Accept failed");
            exit(EXIT_FAILURE);
        }

        // Add new socket to the array of client sockets
        pthread_mutex_lock(&mutex);
        for (i = 0; i < MAX_CLIENTS; i++) {
            if (client_sockets[i] == 0) {
                client_sockets[i] = new_socket;
                printf("New client connected, socket ID: %d\n", new_socket);
                pthread_t thread;
                pthread_create(&thread, NULL, handle_client, (void *)&client_sockets[i]);
                pthread_detach(thread);
                break;
            }
        }
        pthread_mutex_unlock(&mutex);
    }

    close(server_fd);
}

void *handle_client(void *client_socket) {
    int sock = *(int *)client_socket;
    char buffer[BUFFER_SIZE];
    int read_size;

    while ((read_size = recv(sock, buffer, BUFFER_SIZE, 0)) > 0) {
        buffer[read_size] = '\0';  // Null-terminate the message

        // Broadcast the message to all clients
        pthread_mutex_lock(&mutex);
        for (int i = 0; i < MAX_CLIENTS; i++) {
            if (client_sockets[i] != 0 && client_sockets[i] != sock) {
                send(client_sockets[i], buffer, strlen(buffer), 0);
            }
        }
        pthread_mutex_unlock(&mutex);
    }

    // Client disconnected
    pthread_mutex_lock(&mutex);
    for (int i = 0; i < MAX_CLIENTS; i++) {
        if (client_sockets[i] == sock) {
            client_sockets[i] = 0;
            break;
        }
    }
    pthread_mutex_unlock(&mutex);

    close(sock);
    return NULL;
}

void run_client() {
    int sock;
    struct sockaddr_in server_addr;
    char message[BUFFER_SIZE];
    char buffer[BUFFER_SIZE] = {0};
    pthread_t receive_thread;

    // Create socket
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        printf("Socket creation error\n");
        return;
    }

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

```
// Convert address to binary form
if (inet_pton(AF_INET, "127.0.0.1", &server_addr.sin_addr) <= 0) {
  printf("Invalid address/ Address not supported\n");
  return;
}

// Connect to the server
if (connect(sock, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0) {
  printf("Connection failed\n");
  return;
}

// Function to receive messages from the server
void *receive_messages(void *socket) {
  int sockfd = *(int *)socket;
  char recv_buffer[BUFFER_SIZE];
  int recv_size;

  while ((recv_size = recv(sockfd, recv_buffer, BUFFER_SIZE, 0)) > 0) {
    recv_buffer[recv_size] = '\0';  // Null-terminate the received message
    printf("Message from server: %s\n", recv_buffer);
  }

  return NULL;
}

// Create a thread to receive messages
pthread_create(&receive_thread, NULL, receive_messages, (void *)&sock);
pthread_detach(receive_thread);

// Main loop to send messages to the server
while (1) {
  printf("You: ");
  fgets(message, BUFFER_SIZE, stdin);
  message[strcspn(message, "\n")] = '\0';  // Remove trailing newline

  if (send(sock, message, strlen(message), 0) < 0) {
    printf("Send failed\n");
    break;
  }
```

```
    }

    close(sock);
}

int main() {
    int choice;

    printf("Select mode: \n1. Server\n2. Client\n");
    scanf("%d", &choice);
    getchar();  // Consume the newline character after entering choice

    if (choice == 1) {
        run_server();
    } else if (choice == 2) {
        run_client();
    } else {
        printf("Invalid choice.\n");
    }

    return 0;
}
```

31. Implementing ARP protocols in java/C.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <netinet/if_ether.h>
#include <net/if.h>
#include <sys/ioctl.h>
#include <sys/socket.h>
#include <linux/if_packet.h>
#include <net/ethernet.h>

// Function to print MAC address in readable format
void print_mac_address(unsigned char *mac) {
    printf("MAC Address: %02x:%02x:%02x:%02x:%02x:%02x\n",
           mac[0], mac[1], mac[2], mac[3], mac[4], mac[5]);
}
```

```
// Function to send ARP request
void send_arp_request(int sockfd, struct sockaddr_ll *socket_address, unsigned
char *source_mac, unsigned char *target_ip) {
  unsigned char buffer[42]; // ARP packet size
  struct ether_header *eth_header = (struct ether_header *)buffer;
  struct ether_arp *arp_request = (struct ether_arp *)(buffer + ETH_HLEN);

  // Create Ethernet frame
  memset(eth_header->ether_dhost, 0xff, 6); // Broadcast address
  memcpy(eth_header->ether_shost, source_mac, 6); // Source MAC address
  eth_header->ether_type = htons(ETH_P_ARP); // ARP protocol

  // Create ARP request
  arp_request->ea_hdr.ar_hrd = htons(ARPHRD_ETHER); // Hardware type:
Ethernet
  arp_request->ea_hdr.ar_pro = htons(ETH_P_IP);    // Protocol type: IP
  arp_request->ea_hdr.ar_hln = 6;              // Hardware address length
  arp_request->ea_hdr.ar_pln = 4;              // Protocol address length
  arp_request->ea_hdr.ar_op = htons(ARPOP_REQUEST); // Operation: ARP
request

  memcpy(arp_request->arp_sha, source_mac, 6); // Sender MAC address
  inet_pton(AF_INET, "192.168.1.1", arp_request->arp_spa); // Sender IP address
(change it)
  memset(arp_request->arp_tha, 0x00, 6); // Target MAC address (unknown)
  memcpy(arp_request->arp_tpa, target_ip, 4); // Target IP address

  // Send ARP request
  if (sendto(sockfd, buffer, sizeof(buffer), 0, (struct sockaddr *)socket_address,
sizeof(*socket_address)) < 0) {
    perror("ARP request send failed");
  } else {
    printf("ARP request sent.\n");
  }
}

int main() {
  int sockfd;
  struct ifreq ifr;
  struct sockaddr_ll socket_address;
  unsigned char source_mac[6];
```

```c
    unsigned char target_ip[4];

    // Create raw socket
    if ((sockfd = socket(AF_PACKET, SOCK_RAW, htons(ETH_P_ARP))) < 0) {
        perror("Socket creation failed");
        return -1;
    }

    // Get the index of the network interface
    strncpy(ifr.ifr_name, "eth0", IFNAMSIZ - 1); // Change "eth0" to your interface name
    if (ioctl(sockfd, SIOCGIFINDEX, &ifr) < 0) {
        perror("SIOCGIFINDEX failed");
        return -1;
    }
    socket_address.sll_ifindex = ifr.ifr_ifindex;

    // Get the MAC address of the interface
    if (ioctl(sockfd, SIOCGIFHWADDR, &ifr) < 0) {
        perror("SIOCGIFHWADDR failed");
        return -1;
    }
    memcpy(source_mac, ifr.ifr_hwaddr.sa_data, 6);

    printf("Source ");
    print_mac_address(source_mac);

    // Specify target IP (e.g., 192.168.1.2)
    inet_pton(AF_INET, "192.168.1.2", target_ip); // Change this to your target IP address

    // Send ARP request
    send_arp_request(sockfd, &socket_address, source_mac, target_ip);

    close(sockfd);
    return 0;
}
```

32. Implementation of Bit stuffing mechanism using C.

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

```c
#define MAX 100

void bit_stuffing(char input[], char stuffed[]) {
    int count = 0, j = 0;
    int len = strlen(input);

    for (int i = 0; i < len; i++) {
        if (input[i] == '1') {
            count++;
        } else {
            count = 0;
        }

        stuffed[j++] = input[i];

        if (count == 5) {
            stuffed[j++] = '0';  // Insert a '0' after 5 consecutive '1's
            count = 0;  // Reset count
        }
    }

    stuffed[j] = '\0';  // Null-terminate the stuffed string
}

int main() {
    char input[MAX], stuffed[MAX];

    // Input the binary string
    printf("Enter the binary data: ");
    scanf("%s", input);

    // Perform bit stuffing
    bit_stuffing(input, stuffed);

    // Output the result
    printf("After bit stuffing: %s\n", stuffed);

    return 0;
}
```

33. Implementing the applications using TCP file transfer in java/C.

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

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

#define PORT 8080
#define BUFFER_SIZE 1024

// Function to run the server
void run_server() {
  int server_fd, new_socket;
  struct sockaddr_in address;
  int addrlen = sizeof(address);
  char buffer[BUFFER_SIZE] = {0};
  FILE *file;

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

  // Bind the socket to the port
  address.sin_family = AF_INET;
  address.sin_addr.s_addr = INADDR_ANY;
  address.sin_port = htons(PORT);

  if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) < 0) {
    perror("Bind failed");
    close(server_fd);
    exit(EXIT_FAILURE);
  }

  // Listen for incoming connections
  if (listen(server_fd, 3) < 0) {
    perror("Listen failed");
    close(server_fd);
    exit(EXIT_FAILURE);
  }

  printf("Server is waiting for a connection on port %d...\n", PORT);
  // Accept incoming connection
  if ((new_socket = accept(server_fd, (struct sockaddr *)&address,
(socklen_t*)&addrlen)) < 0) {
    perror("Accept failed");
    close(server_fd);
    exit(EXIT_FAILURE);
  }

  // Open file to save the incoming data
  file = fopen("received_file.txt", "wb");
  if (file == NULL) {
    perror("File open error");
    close(new_socket);
    exit(EXIT_FAILURE);
  }

  // Receive the file from the client
  int bytes_received;
  while ((bytes_received = recv(new_socket, buffer, BUFFER_SIZE, 0)) > 0) {
    fwrite(buffer, sizeof(char), bytes_received, file);
  }

  printf("File received successfully!\n");
  fclose(file);
  close(new_socket);
  close(server_fd);
}

// Function to run the client
void run_client() {
  int sock = 0;
  struct sockaddr_in serv_addr;
  char buffer[BUFFER_SIZE] = {0};
  FILE *file;

  // Create client socket
  if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
    printf("\n Socket creation error \n");
    return;
  }
```

```
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);

    // Convert IPv4 address to binary form
    if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr) <= 0) {
        printf("\nInvalid address / Address not supported \n");
        return;
    }

    // Connect to the server
    if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
        printf("\nConnection Failed \n");
        return;
    }

    // Open the file to send
    file = fopen("send_file.txt", "rb");
    if (file == NULL) {
        perror("File open error");
        close(sock);
        return;
    }

    // Send the file to the server
    int bytes_read;
    while ((bytes_read = fread(buffer, sizeof(char), BUFFER_SIZE, file)) > 0) {
        send(sock, buffer, bytes_read, 0);
    }

    printf("File sent successfully!\n");
    fclose(file);
    close(sock);
}

int main() {
    int choice;

    printf("Select mode: \n1. Server\n2. Client\n");
    scanf("%d", &choice);
    getchar();  // Consume the newline character
    if (choice == 1) {
        run_server();
    } else if (choice == 2) {
        run_client();
    } else {
        printf("Invalid choice.\n");
    }

    return 0;
}
```

34. Implementing the simulation of error correction code - CRC in java/C.

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

#define POLYNOMIAL 0x9   // CRC-4 Polynomial: x^3 + x + 1 (binary: 1001)

void xor_operation(char *dividend, char *divisor, int len) {
    for (int i = 0; i < len; i++) {
        dividend[i] = dividend[i] == divisor[i] ? '0' : '1';
    }
}

void crc(char data[], char divisor[], char remainder[]) {
    int data_len = strlen(data);
    int divisor_len = strlen(divisor);
    char temp[20];

    strncpy(temp, data, divisor_len);

    for (int i = 0; i < data_len - divisor_len + 1; i++) {
        if (temp[0] == '1') {
            xor_operation(temp, divisor, divisor_len);
        }

        // Shift left and bring down the next bit
        memmove(temp, temp + 1, divisor_len - 1);
        temp[divisor_len - 1] = data[divisor_len + i];
    }

    strncpy(remainder, temp, divisor_len - 1);
}

int main() {
    char data[20], divisor[20] = "1001", transmitted_data[30], remainder[10];

    // Input data bits
    printf("Enter the data bits: ");
    scanf("%s", data);

    // Append zeros to data (length of divisor - 1)
    int data_len = strlen(data);
    strcpy(transmitted_data, data);
    for (int i = 0; i < strlen(divisor) - 1; i++) {
        strcat(transmitted_data, "0");
    }

    // Perform CRC
    crc(transmitted_data, divisor, remainder);

    // Add remainder (CRC) to the data
    strcat(data, remainder);

    // Output transmitted data with CRC
    printf("Transmitted data with CRC: %s\n", data);

    return 0;
}
```

35. Implementing the sliding window protocol in java/C.

```
#include <stdio.h>

#define WINDOW_SIZE 4
#define TOTAL_PACKETS 10

void sliding_window_protocol() {
  int ack = 0;

  for (int i = 0; i < TOTAL_PACKETS; i++) {
    if (i < ack + WINDOW_SIZE) {
      printf("Sending packet %d\n", i);
    }
    if (i % WINDOW_SIZE == WINDOW_SIZE - 1) {
            ack++;
            printf("ACK received for packets up to %d\n", ack + WINDOW_SIZE - 1);
        }
    }
}

int main() {
    sliding_window_protocol();
    return 0;
}
```