network-exp03-leftside

Program:

TCP Server

```
#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
void main() {
    struct sockaddr_in client_address, server_address;
    int socket_descriptor_1, socket_descriptor_2;
    char message_1[10] = "", message_2[10] = "";
    socket_descriptor_1 = socket(AF_INET, SOCK_STREAM, 0);
    server_address.sin_family = AF_INET;
    server_address.sin_port = 2000;
    server_address.sin_addr.s_addr = inet_addr("127.0.0.1");
    bind(socket_descriptor_1, (struct sockaddr *)&server_address, sizeof(server_address));
    listen(socket_descriptor_1, 1);
    int size_of_client_address = sizeof(client_address);
    socket_descriptor_2 = accept(socket_descriptor_1,
                                 (struct sockaddr *)&client_address,
                                 &size_of_client_address);
    while (1) {
        recv(socket_descriptor_2, message_1, sizeof(message_1), 0);
        if (strcmp(message_1, "end") == 0)
            break;
        printf("\nClient : %s", message_1);
        printf("\nServer : ");
        scanf("%s", message_2);
        send(socket_descriptor_2, message_2, sizeof(message_2), 0);
        if (strcmp(message_2, "end") == 0)
            break;
    }
    close(socket_descriptor_2);
    close(socket_descriptor_1);
}
```

TCP Client

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

```
#include <sys/socket.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
void main() {
    struct sockaddr_in server_address;
    int socket_descriptor;
    char message_1[10] = "", message_2[10] = "";
    socket_descriptor = socket(AF_INET, SOCK_STREAM, 0);
    server_address.sin_family = AF_INET;
    server_address.sin_port = 2000;
    server_address.sin_addr.s_addr = inet_addr("127.0.0.1");
    connect(socket_descriptor, (struct sockaddr *)&server_address,
            sizeof(server_address));
    while (1) {
        printf("\nClient : ");
        scanf("%s", message_2);
        send(socket_descriptor, message_2, sizeof(message_2), 0);
        if (strcmp(message_2, "end") == 0)
            break;
        recv(socket_descriptor, message_1, sizeof(message_1), 0);
        if (strcmp(message_1, "end") == 0)
            break;
        printf("\nServer : %s", message_1);
    }
    close(socket_descriptor);
}
```

Output:

Terminal 1 (Server)

```
ubuntu@ubuntu:~/Desktop$ gcc tcp-server.c
ubuntu@ubuntu:~/Desktop$ ./a.out

Client : Hi
Server : Hello

Client : Nothing
Server : Fine
ubuntu@ubuntu:~/Desktop$
```

Terminal 2 (Client)

```
ubuntu@ubuntu:~/Desktop$ gcc tcp-client.c
ubuntu@ubuntu:~/Desktop$ ./a.out

Client : Hi

Server : Hello
```

```
Client : Nothing

Server : Fine
Client : end
ubuntu@ubuntu:~/Desktop$
```

network-exp04-leftside

Program:

UDP Server

```
#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <sys/types.h>
void main() {
    struct sockaddr_in client_address, server_address;
    int socket_descriptor;
    char message[10];
    socket_descriptor = socket(AF_INET, SOCK_DGRAM, 0);
    server_address.sin_family = AF_INET;
    server_address.sin_port = 3000;
    server_address.sin_addr.s_addr = inet_addr("127.0.0.1");
    bind(socket_descriptor, (struct sockaddr *)&server_address, sizeof(server_address));
    int size_of_client_address = sizeof(client_address);
    while (1) {
        recvfrom(socket_descriptor, message, sizeof(message), 0,
                 (struct sockaddr *)&client_address, &size_of_client_address);
        if (!(strcmp(message, "end")))
            break;
        printf("Client : %s\n", message);
        printf("Server : ");
        scanf("%s", message);
        sendto(socket_descriptor, message, sizeof(message), 0,
               (struct sockaddr *)&client_address, size_of_client_address);
    }
}
```

UDP Client

```
#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <sys/types.h>
void main() {
    struct sockaddr_in server_address;
    int socket_descriptor;
    char message[10];
    socket_descriptor = socket(AF_INET, SOCK_DGRAM, 0);
    server_address.sin_family = AF_INET;
    server_address.sin_port = 3000;
    server_address.sin_addr.s_addr = inet_addr("127.0.0.1");
    bind(socket_descriptor, (struct sockaddr *)&server_address, sizeof(server_address));
    while (1) {
        printf("Client : ");
        scanf("%s", message);
        sendto(socket_descriptor, message, sizeof(message), 0,
               (struct sockaddr *)&server_address, sizeof(server_address));
        if (!(strcmp(message, "end")))
            break;
        recvfrom(socket_descriptor, message, sizeof(message), 0, NULL, NULL);
        printf("Server : %s\n", message);
    }
}
```

Output:

Terminal 1 (Server)

```
ubuntu@ubuntu:~/Desktop$ gcc udp-server.c
ubuntu@ubuntu:~/Desktop$ ./a.out
Client : Hello
Server : Hi
Client : Nothing
Server : Fine
ubuntu@ubuntu:~/Desktop$
```

Terminal 2 (Client)

```
ubuntu@ubuntu:~/Desktop$ gcc udp-client.c
ubuntu@ubuntu:~/Desktop$ ./a.out
Client : Hello
Server : Hi
Client : Nothing
Server : Fine
Client : end
ubuntu@ubuntu:~/Desktop$
```

network-exp05-leftside

Program:

FTP Server

```
#include <arpa/inet.h>
#include <fcntl.h>
#include <stdio.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>
#define BUFSIZE 1024
#define PORT_ADDR 6000
void main() {
    struct sockaddr_in server_address, client_address;
    int i, socket_descriptor_1, socket_descriptor_2, size_of_client_address,
        file_descriptor, character_count;
    char buffer_1[BUFSIZE], buffer_2[BUFSIZE];
    socket_descriptor_1 = socket(AF_INET, SOCK_STREAM, 0);
    server_address.sin_family = AF_INET;
    server_address.sin_port = htons(PORT_ADDR);
    server_address.sin_addr.s_addr = htonl(INADDR_ANY);
    bind(socket_descriptor_1, (struct sockaddr *)&server_address, sizeof(server_address));
    listen(socket_descriptor_1, 1);
    size_of_client_address = sizeof(client_address);
    socket_descriptor_2 = accept(socket_descriptor_1, (struct sockaddr *)&client_address,
                                 &size_of_client_address);
    printf("\nRequest accepted...\n");
    close(socket_descriptor_1);
    character_count = read(socket_descriptor_2, buffer_1, BUFSIZE);
    buffer_1[character_count] = '\0';
    printf("\nFilename is %s", buffer_1);
    if ((file_descriptor = open(buffer_1, O_RDONLY)) >= 0) {
        printf("\n\nFile opened...");
        character_count = read(file_descriptor, buffer_2, BUFSIZE);
        while (character_count > 0) {
            printf("\n\nSending file...");
            write(socket_descriptor_2, buffer_2, character_count);
            printf("\n\nPrinting content of file:\n");
            for (i = 0; i < character_count; i++)
                printf("%c", buffer_2[i]);
            printf("\n");
            if (i == character_count)
                break;
        }
    } else
        printf("\n\nRequested file is not present...\n\n");
    close(file_descriptor);
    close(socket_descriptor_2);
}
```

FTP Client

```
#include <arpa/inet.h>
#include <fcntl.h>
#include <netinet/in.h>
#include <stdio.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
void main() {
    struct sockaddr_in server_address;
    int socket_descriptor, character_count, file_length, i;
    char buffer_1[3024], buffer_2[3000];
    socket_descriptor = socket(AF_INET, SOCK_STREAM, 0);
    server_address.sin_family = AF_INET;
    server_address.sin_port = htons(6000);
    server_address.sin_addr.s_addr = inet_addr("127.0.0.1");
    connect(socket_descriptor, (struct sockaddr *)&server_address,
            sizeof(server_address));
    printf("\nEnter the filename:\n");
    character_count = read(0, buffer_1, sizeof(buffer_1));
    buffer_1[character_count - 1] = '\0';
    write(socket_descriptor, buffer_1, character_count);
    file_length = read(socket_descriptor, buffer_2, sizeof(buffer_2));
    if (file_length == 0)
        printf("\n\nFile not present....!!!");
    else {
        printf("\n\nPrinting content of file:\n");
        for (i = 0; i < file_length; i++)
            printf("%c", buffer_2[i]);
        printf("\nReceived requested file\n\n");
    }
    close(socket_descriptor);
}
```

Output:

Terminal 1 (Server)

```
ubuntu@ubuntu:~/Desktop$ gcc ftp-server.c
ubuntu@ubuntu:~/Desktop$ ./a.out

Request accepted...

Filename is test.txt

File opened...

Sending file...

Printing content of file:
```

```
hello world
this is a test file
have a good day

ubuntu@ubuntu:~/Desktop$
```

Terminal 2 (Client)

```
ubuntu@ubuntu:~/Desktop$ gcc ftp-client.c
ubuntu@ubuntu:~/Desktop$ ./a.out

Enter the filename:
test.txt

Printing content of file:
hello world
this is a test file
have a good day

Received requested file

ubuntu@ubuntu:~/Desktop$
```

network-exp06-distance-vector-leftside

Program:

```
#include <stdio.h>
void main() {
    int number_of_nodes, i, j, k, x;
    printf("Enter the number of nodes: ");
    scanf("%d", &number_of_nodes);
    int routing_table[number_of_nodes][number_of_nodes];
    printf("Enter the routing table:\n");
    for (int i = 0; i < number_of_nodes; i++)
        for (int j = 0; j < number_of_nodes; j++) {
            printf("[%d][%d]: ", i, j);
            scanf("%d", &routing_table[i][j]);
        }
    for (int x = 0; x < number_of_nodes; x++)
        for (int i = 0; i < number_of_nodes; i++)
            for (int j = 0; j < number_of_nodes; j++)
                for (int k = 0; k < number_of_nodes; k++)
                    if (routing_table[i][j] > routing_table[i][k] + routing_table[k][j])
                        routing_table[i][j] = routing_table[i][k] + routing_table[k][j];
    printf("\nDistance Vector Table:\n");
    for (int i = 0; i < number_of_nodes; i++) {
        for (int j = 0; j < number_of_nodes; j++)
            printf("%d\t", routing_table[i][j]);
        printf("\n");
    }
}
```

Output:

```
Enter the number of nodes: 4
Enter the routing table:
[0][0]: 0
[0][1]: 3
[0][2]: 2
[0][3]: 5
[1][0]: 3
[1][1]: 0
[1][2]: 8
[1][3]: 4
[2][0]: 2
[2][1]: 8
[2][2]: 0
```

```
[2][3]: 1
[3][0]: 5
[3][1]: 4
[3][2]: 1
[3][3]: 0

Distance Vector Table:
0       3       2       3
3       0       5       4
2       5       0       1
3       4       1       0
```

network-exp07-sliding-window-leftside

Program:

a) Stop-and-Wait

Server:

```
#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>
int main() {
    printf("\nWaiting for client...\n");
    struct sockaddr_in servaddr, cliaddr;
    int sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    memset(&servaddr, 0, sizeof(servaddr));
    memset(&cliaddr, 0, sizeof(cliaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = INADDR_ANY;
    servaddr.sin_port = htons(8080);
    bind(sockfd, (const struct sockaddr *)&servaddr, sizeof(servaddr));
    int len = sizeof(cliaddr);
    int frames[100], n;
    recvfrom(sockfd, &n, sizeof(n), 0, (struct sockaddr *)&cliaddr, &len);
    printf("\nClient connected successfuly.\n");
    printf("\nWaiting for total number of frames...\n");
    recvfrom(sockfd, &n, sizeof(n), 0, (struct sockaddr *)&cliaddr, &len);
    int p, ack;
    while (1) {
        recvfrom(sockfd, &p, sizeof(n), 0, (struct sockaddr *)&cliaddr, &len);
        if (p == -99)
            return 0;
        printf("\nReceived frame-%d ", p);
        printf("\nEnter 1 for +ve ack and -1 for -ve ack.\n");
        scanf("%d", &ack);
        sendto(sockfd, &ack, sizeof(n), 0, (struct sockaddr *)&cliaddr, sizeof(cliaddr));
    }
    return 0;
}
```

Client:

```
#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>
int main() {
    printf("\nSearching for server...\n");
    struct sockaddr_in servaddr;
    int sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    memset(&servaddr, 0, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(8080);
    servaddr.sin_addr.s_addr = INADDR_ANY;
    int n = -1;
    sendto(sockfd, &n, sizeof(n), 0, (struct sockaddr *)&servaddr, sizeof(servaddr));
    printf("\nServer connected successfully.\n");
    printf("\nEnter the total number of frames: ");
    scanf("%d", &n);
    sendto(sockfd, &n, sizeof(n), 0, (struct sockaddr *)&servaddr, sizeof(servaddr));
    int len, ack;
    for (int i = 1; i <= n; i++) {
        ack = -1;
        do {
            printf("\nSending frames-%d ", i);
            sendto(sockfd, &i, sizeof(n), 0, (struct sockaddr *)&servaddr,
                   sizeof(servaddr));
            printf("\nWaiting for ACK...\n");
            recvfrom(sockfd, &ack, sizeof(n), 0, (struct sockaddr *)&servaddr, &len);
            if (ack == -1) {
                printf("\nNegative ack recieved.. resending...\n");
            }
        } while (ack == -1);
    }
    n = -99;
    sendto(sockfd, &n, sizeof(n), 0, (const struct sockaddr *)&servaddr,
           sizeof(servaddr));
    printf("\nSuccessfully sent all the frames.\n");
    close(sockfd);
    return 0;
}
```

b) Go-Back-N

Server:

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

```
#include <sys/socket.h>
#include <unistd.h>
#define WINDOW_SIZE 4
int main() {
    int socket_1, socket_2, length, i, j, status, client_address_length;
    char buffer[20], frame[20], temp[20], ack[20];
    struct sockaddr_in server_address, client_address;
    socket_1 = socket(AF_INET, SOCK_STREAM, 0);
    bzero(&server_address, sizeof(server_address));
    server_address.sin_family = AF_INET;
    server_address.sin_addr.s_addr = htonl(INADDR_ANY);
    server_address.sin_port = htons(5465);
    bind(socket_1, (struct sockaddr *)&server_address, sizeof(server_address));
    listen(socket_1, 5);
    client_address_length = sizeof(&client_address);
    socket_2 = accept(socket_1, (struct sockaddr *)&client_address,
                      &client_address_length);
    printf("Enter the text:\n");
    scanf("%s", buffer);
    i = 0;
    while (i < strlen(buffer)) {
        memset(frame, 0, 20);
        strncpy(frame, buffer + i, WINDOW_SIZE);
        // Transmitting Frames
        printf("\nTransmitting frames: ");
        length = strlen(frame);
        for (j = 0; j < length; j++) {
            printf("%d ", i + j);
            sprintf(temp, "%d", i + j);
            strcat(frame, temp);
        }
        printf("\n");
        write(socket_2, frame, sizeof(frame));
        read(socket_2, ack, 20);
        // Read acknowledgement
        sscanf(ack, "%d", &status);
        if (status == -1)
            printf("Transmission is successful\n");
        else {
            printf("Received error in %d\n", status);
            printf("Retransmitting frames: ");
            // Retransmitting Frames
            for (j = 0;;) {
                frame[j] = buffer[j + status];
                printf("%d ", j + status);
                j++;
                if ((j + status) % WINDOW_SIZE == 0)
                    break;
            }
            printf("\n");
            frame[j] = '\0';
            length = strlen(frame);
            for (j = 0; j < length; j++) {
                sprintf(temp, "%d", j + status);
                strcat(frame, temp);
            }
            write(socket_2, frame, sizeof(frame));
        }
        i += WINDOW_SIZE;
    }
    write(socket_2, "Exit", sizeof("Exit"));
    printf("\nExiting\n");
    close(socket_2);
    close(socket_1);
}
```

Client:

```
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>
int main() {
    int socket_1, choice;
    char buffer[20], error[20];
    struct sockaddr_in server_address;
    socket_1 = socket(AF_INET, SOCK_STREAM, 0);
    bzero(&server_address, sizeof(server_address));
    server_address.sin_family = AF_INET;
    server_address.sin_addr.s_addr = htonl(INADDR_ANY);
    server_address.sin_port = htons(5465);
    connect(socket_1, (struct sockaddr *)&server_address, sizeof(server_address));
    while (1) {
        // Receive Frame
        read(socket_1, buffer, 20);
        if (!strcmp(buffer, "Exit")) {
            printf("\nExiting\n");
            break;
        }
        printf("\nReceived %s", buffer);
        printf("\nDo you want to report error? (1 - Yes, 0 - No): ");
        scanf("%d", &choice);
        if (!choice)
            write(socket_1, "-1", sizeof("-1"));
        else {
            printf("Enter the sequence of frame where error has occured: ");
            scanf("%s", error);
            // Send acknowledgement
            write(socket_1, error, sizeof(error));
            read(socket_1, buffer, 20);
            printf("\nReceived the retransmitted frames: %s\n", buffer);
        }
    }
}
```

c) Selective-Repeat

Server:

```
#include <arpa/inet.h>
#include <netdb.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
#define cls() printf("Clearing")
// Structure definition for packet
struct frame {
    int packet[40];
};
// Structure definition for ack
struct ack {
    int acknowledge[40];
};
int main() {
    int socket_fd;
    struct sockaddr_in serveraddr, clientaddr;
    socklen_t len;
    struct frame f1;
    int windowsize, totalpackets, totalframes, i = 0, j = 0, framessend = 0, k, m, n, l;
    int repacket[40];
    struct ack acknowledgement;
    char req[50];
    socket_fd = socket(AF_INET, SOCK_DGRAM, 0);
    bzero((char*)&serveraddr, sizeof(serveraddr));
    serveraddr.sin_family = AF_INET;
    serveraddr.sin_port = htons(5018);
    serveraddr.sin_addr.s_addr = INADDR_ANY;
    bind(socket_fd, (struct sockaddr*)&serveraddr, sizeof(serveraddr));
    bzero((char*)&clientaddr, sizeof(clientaddr));
    len = sizeof(clientaddr);
    // Establishing connection
    printf("\nWaiting for client connetion \n");
    recvfrom(socket_fd, req, sizeof(req), 0, (struct sockaddr*)&clientaddr, &len);
    printf("\nClient connection obtained\t%s\n", req);
    // Sending request for window size
    printf("\nSending request for window size\n");
    sendto(socket_fd, "REQUEST FOR WINDOW SIZE", sizeof("REQUST FOR WINDOW SIZE"), 0,
           (struct sockaddr*)&clientaddr, sizeof(clientaddr));
    // Obtaining window size
    printf("\nWaiting for the window size\n");
    recvfrom(socket_fd, (char*)&windowsize, sizeof(windowsize), 0,
             (struct sockaddr*)&clientaddr, &len);
    cls();
    printf("The window size obtained as: \t%d\n", windowsize);
    printf("\nObtaining packets from network layer\n");
    printf("\nTotal packets obtained \t%d\n", (totalpackets = windowsize * 5));
    printf("\nTotal frames or windows to be transmitted\t%d\n", (totalframes = 5));
    // Sending details to client
    printf("\nSending total number of packets\n");
    sendto(socket_fd, (char*)&totalpackets, sizeof(totalpackets), 0,
           (struct sockaddr*)&clientaddr, sizeof(clientaddr));
    recvfrom(socket_fd, req, sizeof(req), 0, (struct sockaddr*)&clientaddr, &len);
    printf("\nSending total number of frames\n");
    sendto(socket_fd, (char*)&totalframes, sizeof(totalframes), 0,
           (struct sockaddr*)&clientaddr, sizeof(clientaddr));
    recvfrom(socket_fd, req, sizeof(req), 0, (struct sockaddr*)&clientaddr, &len);
    printf("\nPress enter to start the process\n");
    fgets(req, 2, stdin);
    cls();
    j = 0;
    l = 0;
    // Starting sending process
    while (l < totalpackets) {
        // Initializing transmit buffer
        bzero((char*)&f1, sizeof(f1));
        printf("Initializing the transmit buffer \n");
        printf("The frame to be send is %d with packets\n", framessend);
        // Building the frame
        for (m = 0; m < j; m++) {
            // Including the packets for which negative ack was received
            printf("%d", repacket[m]);
            f1.packet[m] = repacket[m];
        }
        while (j < windowsize && i < totalpackets) {
            printf("%d", i);
            f1.packet[j] = i;
            i++;
            j++;
        }
        printf("Sending frame %d\n", framessend);
        // Sending the frame
        sendto(socket_fd, (char*)&f1, sizeof(f1), 0, (struct sockaddr*)&clientaddr,
               sizeof(clientaddr));
        // Waiting for ack
        printf("Waiting for the acknowledgement\n");
        recvfrom(socket_fd, (char*)&acknowledgement, sizeof(acknowledgement), 0,
                 (struct sockaddr*)&clientaddr, &len);
        cls();
        // Checking ack of each packet
        j = 0;
        m = 0;
        k = 0;
        n = 1;
        while (m < windowsize && n < totalpackets) {
            if (acknowledgement.acknowledge[m] == -1) {
                printf("\nNegetive acknowledgement received for packet:%d\n",
                       f1.packet[m]);
                k = 1;
                repacket[j] = f1.packet[m];
                j++;
            } else {
                l++;
            }
            m++;
            n++;
        }
        if (k == 0) {
            printf("\nPositive acknowledgement received for all packets,
                   waiting for frame: %d\n ", framessend);
        }
        framessend++;
        printf("\nPress enter to proceed\n");
        fgets(req, 2, stdin);
        cls();
    }
    printf("All frames sent successfully.\nClosing connection with client");
    close(socket_fd);
}
```

Client:

```
#include <netdb.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#define cls() printf("Clearing")
// Structure definition for packet
struct frame {
    int packet[40];
};
// Structure definition for ack
struct ack {
    int acknowledge[40];
};
int main() {
    int client_socket_fd;
    struct sockaddr_in serveraddr;
    socklen_t len;
    struct hostent* server;
    struct frame f1;
    int windowsize, totalpackets, totalframes, i = 0, j = 0, framesreceived = 0, k, m,
        n, l;
    int repacket[40];
    struct ack acknowledgement;
    char req[50];
    client_socket_fd = socket(AF_INET, SOCK_DGRAM, 0);
    bzero((char*)&serveraddr, sizeof(serveraddr));
    serveraddr.sin_family = AF_INET;
    serveraddr.sin_port = htons(5018);
    server = gethostbyname("127.0.0.1");
    bcopy((char*)server->h_addr, (char*)&serveraddr.sin_addr.s_addr,
          sizeof(server->h_addr));
    // Establishing connection
    printf("Sending request to the server\n");
    sendto(client_socket_fd, "HI IAM CLIENT.", sizeof("HI IAM CLIENT."), 0,
           (struct sockaddr*)&serveraddr, sizeof(serveraddr));
    printf("\nWaiting for reply\n");
    recvfrom(client_socket_fd, req, sizeof(req), 0, (struct sockaddr*)&serveraddr, &len);
    printf("The server has send:It %s \n", req);
    // accepting window size from the user.
    printf("\nEnter the window size \n");
    scanf("%d", &windowsize);
    // sending the window size.
    printf("Sending window size\n");
    sendto(client_socket_fd, (char*)&windowsize, sizeof(windowsize), 0,
           (struct sockaddr*)&serveraddr, sizeof(serveraddr));
    cls();
    // collecting details from server.
    printf("\nWaiting for the server response\n");
    recvfrom(client_socket_fd, (char*)&totalpackets, sizeof(totalpackets), 0,
             (struct sockaddr*)&serveraddr, &len);
    printf("The total packets are:%d\n", totalpackets);
    sendto(client_socket_fd, "RECEIVED.", sizeof("RECEIVED."), 0,
           (struct sockaddr*)&serveraddr, sizeof(serveraddr));
    recvfrom(client_socket_fd, (char*)&totalframes, sizeof(totalframes), 0,
             (struct sockaddr*)&serveraddr, &len);
    printf("The frames of windows are:\t%d\n", totalframes);
    sendto(client_socket_fd, "RECEIVED.", sizeof("RECEIVED."), 0,
           (struct sockaddr*)&serveraddr, sizeof(serveraddr));
    // starting the process.
    printf("Starting the process of receiving:\n");
    l = 0;
    j = 0;
    while (l < totalpackets) {
        // initialising the receive buffer.
        printf("Initializing the receiver buffer \n");
        printf("The Expected frame is %d with packets:", framesreceived);
        for (m = 0; m < j; m++) {
            // readjusting for packets with negative acknowledgemen
            printf("%d", repacket[m]);
        }
        while (j < windowsize && i < totalpackets) {
            printf("%d", i);
            i++;
            j++;
        }
        printf("\nWaiting for the frame \n");
        // accepting the frame.
        recvfrom(client_socket_fd, (char*)&f1, sizeof(f1), 0,
                 (struct sockaddr*)&serveraddr, &len);
        printf("Received frame %d \n\nEnter to send negative acknowledgement for
               the following packets.\n ", framesreceived);
        // constructing the acknowledgement frame.
        j = 0;
        m = 0;
        k = l;
        while (m < windowsize && k < totalpackets) {
            printf("\nPacket:%d\n", f1.packet[m]);
            // accepting acknowledgement from the user.
            scanf("%d", &acknowledgement.acknowledge[m]);
            if (acknowledgement.acknowledge[m] == -1) {
                repacket[j] = f1.packet[m];
                j++;
            } else {
                l++;
            }
            m++;
            k++;
        }
        framesreceived++;
        // sending acknowledgement to the server.
        sendto(client_socket_fd, (char*)&acknowledgement, sizeof(acknowledgement), 0,
               (struct sockaddr*)&serveraddr, sizeof(serveraddr));
        cls();
    }
    printf("\nAll frames received successfully.\n\nClosing connection with the
           server.\n ");
    close(client_socket_fd);
}
```

Output:

a) Stop-and-Wait

Terminal 1 (Server)

```
Waiting for client...

Client connected successfuly.

Waiting for total number of frames...

Received frame-1
Enter 1 for +ve ack and -1 for -ve ack.
1

Received frame-2
Enter 1 for +ve ack and -1 for -ve ack.
1

Received frame-3
Enter 1 for +ve ack and -1 for -ve ack.
-1

Received frame-3
Enter 1 for +ve ack and -1 for -ve ack.
1
```

Terminal 2 (Client)

```
Searching for server...

Server connected successfully.

Enter the total number of frames: 3
```

```
Sending frames-1
Waiting for ACK...

Sending frames-2
Waiting for ACK...

Sending frames-3
Waiting for ACK...

Negative ack recieved.. resending...

Sending frames-3
Waiting for ACK...

Successfully sent all the frames.
```

b) Go-Back-N

Terminal 1 (Server)

```
Enter the text:
HELLO

Transmitting frames: 0 1 2 3
Received error in 2
Retransmitting frames: 2 3

Transmitting frames: 4
Transmission is successful

Exiting
```

Terminal 2 (Client)

```
Received HELL0123
Do you want to report error? (1 - Yes, 0 - No): 1
Enter the sequence of frame where error has occured: 2

Received the retransmitted frames: LL23

Received O4
Do you want to report error? (1 - Yes, 0 - No): 0

Exiting
```

c) Selective Repeat

Terminal 1 (Server)

```
Waiting for client connetion

Client connection obtained  HI IAM CLIENT.

Sending request for window size

Waiting for the window size
Clearing
The window size obtained as:  3

Obtaining packets from network layer

Total packets obtained  6

Total frames or windows to be transmitted 3

Sending total number of packets

Sending total number of frames

Press enter to start the process

Clearing
Initializing the transmit buffer
The frame to be send is 0 with packets
012
Sending frame 0
Waiting for the acknowledgement
Clearing
Negetive acknowledgement received for packet:0

Press enter to proceed

Clearing
Initializing the transmit buffer
The frame to be send is 1 with packets
034
Sending frame 1
Waiting for the acknowledgement
Clearing
Positive acknowledgement received for all packets, waiting for frame: 1

Press enter to proceed

Clearing
Initializing the transmit buffer
The frame to be send is 2 with packets
5
Sending frame 2
Waiting for the acknowledgement
Clearing
Positive acknowledgement received for all packets, waiting for frame: 2

Press enter to proceed
```

```
Clearing
All frames sent successfully.
Closing connection with client
```

Terminal 2 (Client)

```
Sending request to the server

Waiting for reply
The server has send:It REQUEST FOR WINDOW SIZE

Enter the window size
3
Sending window size
Clearing
Waiting for the server response
The total packets are:6
The frames of windows are:  3
Starting the process of receiving:
Initializing the receiver buffer
The Expected frame is 0 with packets:012
Waiting for the frame
Received frame 0

Enter to send negative acknowledgement for the following packets.

Packet:0
-1

Packet:1
1

Packet:2
1
Clearing
Initializing the receiver buffer
The Expected frame is 1 with packets:034
Waiting for the frame
Received frame 1

Enter to send negative acknowledgement for the following packets.

Packet:0
1

Packet:3
1

Packet:4
1
Clearing
Initializing the receiver buffer
The Expected frame is 2 with packets:5
Waiting for the frame
```

```
Received frame 2

Enter to send negative acknowledgement for the following packets.

Packet:5
1
Clearing
All frames received successfully.

Closing connection with the server.
```

network-exp08-leaky-bucket-leftside

Program:

```
#include<stdio.h>
int main() {
    int incoming, outgoing, buck_size, n, store = 0;
    printf("Enter bucket size, outgoing rate and number of inputs: ");
    scanf("%d %d %d", & buck_size, & outgoing, & n);
    while (n != 0) {
        printf("Enter incoming packet size: ");
        scanf("%d", & incoming);
        printf("Incoming packet size is %d\n", incoming);
        if (incoming <= (buck_size - store)) {
            store += incoming;
            printf("Bucket buffer size is %d out of %d\n", store,
                   buck_size);
        } else {
            printf("Dropped %d no of packets\n",
                   incoming - (buck_size - store));
            store = buck_size;
            printf("Bucket buffer size is %d out of %d\n", store,
                   buck_size);
        }
        store = store - outgoing;
        if (store < 0)
            store = 0;
        printf("After outgoing, %d packets left out of %d
                in buffer\n", store, buck_size);
        n--;
    }
}
```

Output:

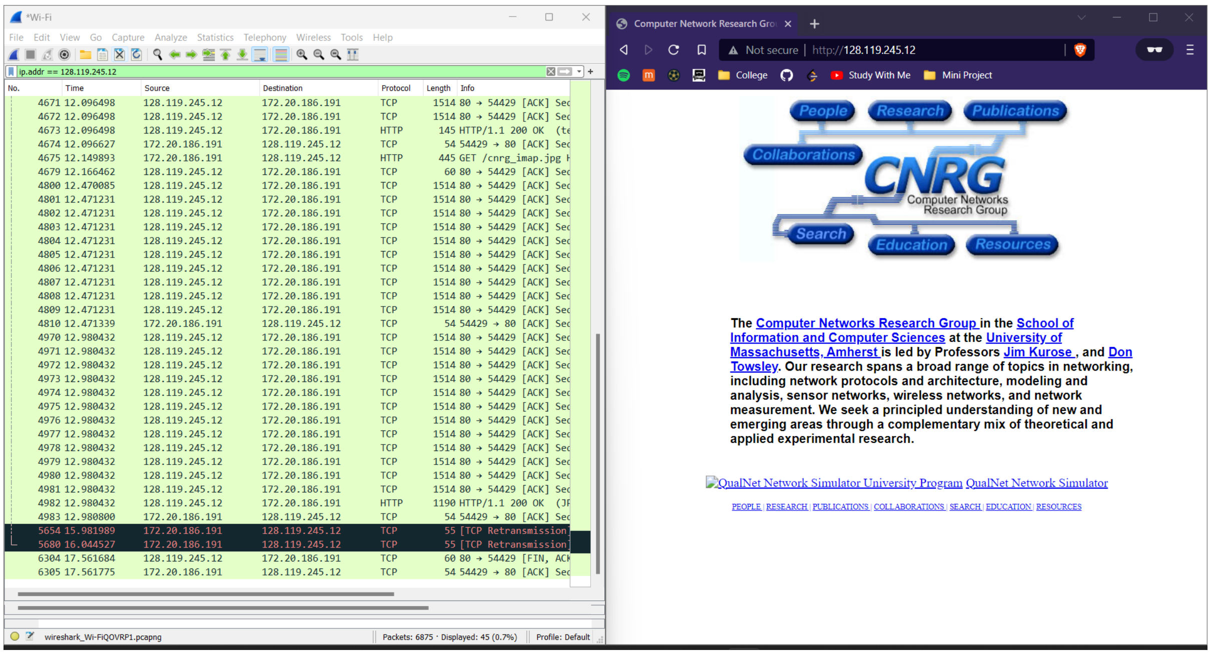
```
Enter bucket size, outgoing rate and number of inputs: 25 10 4
Enter incoming packet size: 20
Incoming packet size is 20
Bucket buffer size is 20 out of 25
After outgoing, 10 packets left out of 25 in buffer
Enter incoming packet size: 20
Incoming packet size is 20
Dropped 5 no of packets
Bucket buffer size is 25 out of 25
After outgoing, 15 packets left out of 25 in buffer
```

```
Enter incoming packet size: 5
Incoming packet size is 5
Bucket buffer size is 20 out of 25
After outgoing, 10 packets left out of 25 in buffer
Enter incoming packet size: 10
Incoming packet size is 10
Bucket buffer size is 20 out of 25
After outgoing, 10 packets left out of 25 in buffer
```

network-exp09-wireshark-leftside

Output:

a) Observing TCP connection


*Wi-Fi
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Computer Network Research Gro
Not secure | http://128.119.245.12
College
Study With Me
Mini Project
People
Research
Publications
Collaborations
CNRG
Computer Networks Research Group
Search
Education
Resources

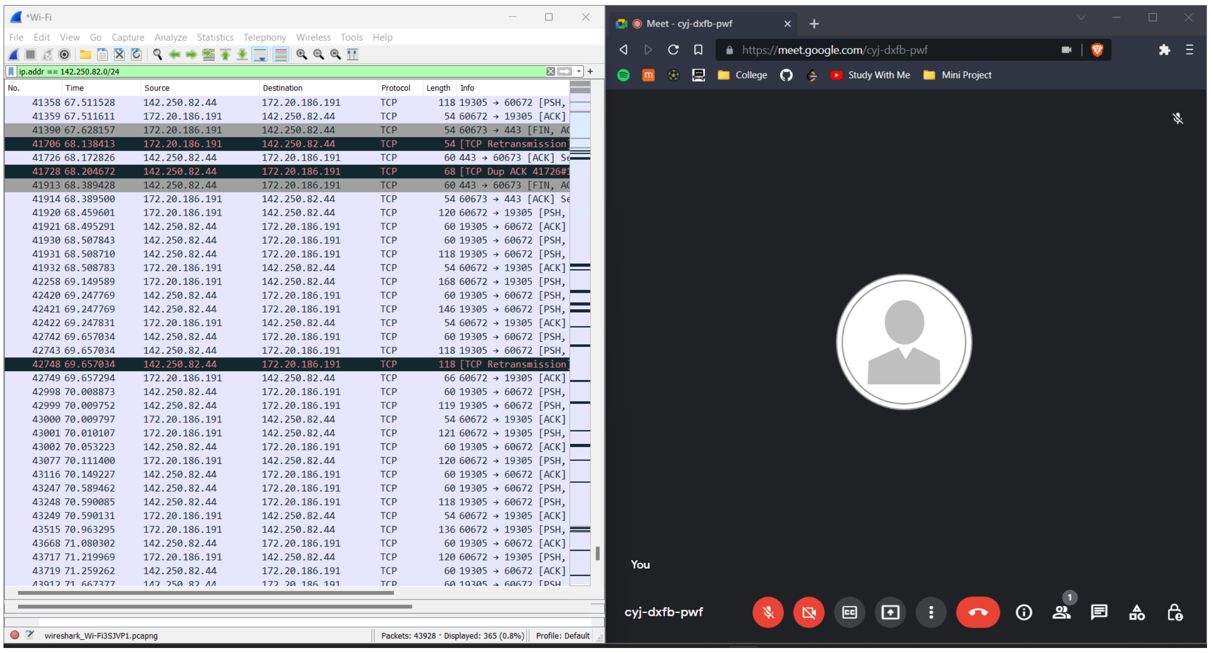
```
ip.addr == 128.119.245.12

No.    Time         Source           Destination      Protocol  Length  Info
  4671 12.096498    128.119.245.12   172.20.186.191   TCP       1514 80 → 54429 [ACK] Sec
  4672 12.096498    128.119.245.12   172.20.186.191   TCP       1514 80 → 54429 [ACK] Sec
  4673 12.096498    128.119.245.12   172.20.186.191   HTTP       145 HTTP/1.1 200 OK  (te
  4674 12.096627    172.20.186.191   128.119.245.12   TCP         54 54429 → 80 [ACK] Sec
  4675 12.149893    172.20.186.191   128.119.245.12   HTTP       445 GET /cnrg_imap.jpg H
  4679 12.166462    128.119.245.12   172.20.186.191   TCP         60 80 → 54429 [ACK] Sec
  4800 12.470085    128.119.245.12   172.20.186.191   TCP       1514 80 → 54429 [ACK] Sec
  4801 12.471231    128.119.245.12   172.20.186.191   TCP       1514 80 → 54429 [ACK] Sec
  4802 12.471231    128.119.245.12   172.20.186.191   TCP       1514 80 → 54429 [ACK] Sec
  4803 12.471231    128.119.245.12   172.20.186.191   TCP       1514 80 → 54429 [ACK] Sec
  4804 12.471231    128.119.245.12   172.20.186.191   TCP       1514 80 → 54429 [ACK] Sec
  4805 12.471231    128.119.245.12   172.20.186.191   TCP       1514 80 → 54429 [ACK] Sec
  4806 12.471231    128.119.245.12   172.20.186.191   TCP       1514 80 → 54429 [ACK] Sec
  4807 12.471231    128.119.245.12   172.20.186.191   TCP       1514 80 → 54429 [ACK] Sec
  4808 12.471231    128.119.245.12   172.20.186.191   TCP       1514 80 → 54429 [ACK] Sec
  4809 12.471231    128.119.245.12   172.20.186.191   TCP       1514 80 → 54429 [ACK] Sec
  4810 12.471339    172.20.186.191   128.119.245.12   TCP         54 54429 → 80 [ACK] Sec
  4970 12.980432    128.119.245.12   172.20.186.191   TCP       1514 80 → 54429 [ACK] Sec
  4971 12.980432    128.119.245.12   172.20.186.191   TCP       1514 80 → 54429 [ACK] Sec
  4972 12.980432    128.119.245.12   172.20.186.191   TCP       1514 80 → 54429 [ACK] Sec
  4973 12.980432    128.119.245.12   172.20.186.191   TCP       1514 80 → 54429 [ACK] Sec
  4974 12.980432    128.119.245.12   172.20.186.191   TCP       1514 80 → 54429 [ACK] Sec
  4975 12.980432    128.119.245.12   172.20.186.191   TCP       1514 80 → 54429 [ACK] Sec
  4976 12.980432    128.119.245.12   172.20.186.191   TCP       1514 80 → 54429 [ACK] Sec
  4977 12.980432    128.119.245.12   172.20.186.191   TCP       1514 80 → 54429 [ACK] Sec
  4978 12.980432    128.119.245.12   172.20.186.191   TCP       1514 80 → 54429 [ACK] Sec
  4979 12.980432    128.119.245.12   172.20.186.191   TCP       1514 80 → 54429 [ACK] Sec
  4980 12.980432    128.119.245.12   172.20.186.191   TCP       1514 80 → 54429 [ACK] Sec
  4981 12.980432    128.119.245.12   172.20.186.191   TCP       1514 80 → 54429 [ACK] Sec
  4982 12.980432    128.119.245.12   172.20.186.191   HTTP      1190 HTTP/1.1 200 OK  (JF
  4983 12.980800    172.20.186.191   128.119.245.12   TCP         54 54429 → 80 [ACK] Sec
  5654 15.981989    172.20.186.191   128.119.245.12   TCP         55 [TCP Retransmission]
  5680 16.044527    172.20.186.191   128.119.245.12   TCP         55 [TCP Retransmission]
  6304 17.561684    128.119.245.12   172.20.186.191   TCP         60 80 → 54429 [FIN, ACK
  6305 17.561775    172.20.186.191   128.119.245.12   TCP         54 54429 → 80 [ACK] Sec

wireshark_Wi-FiQOVRP1.pcapng    Packets: 6875 · Displayed: 45 (0.7%)    Profile: Default
```

The Computer Networks Research Group in the School of Information and Computer Sciences at the University of Massachusetts, Amherst is led by Professors Jim Kurose , and Don Towsley. Our research spans a broad range of topics in networking, including network protocols and architecture, modeling and analysis, sensor networks, wireless networks, and network measurement. We seek a principled understanding of new and emerging areas through a complementary mix of theoretical and applied experimental research.

QualNet Network Simulator University Program QualNet Network Simulator

PEOPLE | RESEARCH | PUBLICATIONS | COLLABORATIONS | SEARCH | EDUCATION | RESOURCES

b) Observing UDP connection


*Wi-Fi
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Meet - cyj-dxfb-pwf
https://meet.google.com/cyj-dxfb-pwf
College
Study With Me
Mini Project
You
cyj-dxfb-pwf

```
ip.addr == 142.250.82.0/24

No.      Time        Source           Destination      Protocol  Length Info
   41358 67.511528   142.250.82.44    172.20.186.191   TCP       118 19305 → 60672 [PSH,
   41359 67.511611   172.20.186.191   142.250.82.44    TCP        54 60672 → 19305 [ACK]
   41390 67.628157   172.20.186.191   142.250.82.44    TCP        54 60673 → 443 [FIN, AC
   41706 68.138413   172.20.186.191   142.250.82.44    TCP        54 [TCP Retransmission]
   41726 68.172826   142.250.82.44    172.20.186.191   TCP        60 443 → 60673 [ACK] Se
   41728 68.204672   142.250.82.44    172.20.186.191   TCP        68 [TCP Dup ACK 41726#1
   41913 68.389428   142.250.82.44    172.20.186.191   TCP        60 443 → 60673 [FIN, AC
   41914 68.389500   172.20.186.191   142.250.82.44    TCP        54 60673 → 443 [ACK] Se
   41920 68.459601   172.20.186.191   142.250.82.44    TCP       120 60672 → 19305 [PSH,
   41921 68.495291   142.250.82.44    172.20.186.191   TCP        60 19305 → 60672 [ACK]
   41930 68.507843   142.250.82.44    172.20.186.191   TCP        60 19305 → 60672 [PSH,
   41931 68.508710   142.250.82.44    172.20.186.191   TCP       118 19305 → 60672 [PSH,
   41932 68.508783   172.20.186.191   142.250.82.44    TCP        54 60672 → 19305 [ACK]
   42258 69.149589   172.20.186.191   142.250.82.44    TCP       168 60672 → 19305 [PSH,
   42420 69.247769   142.250.82.44    172.20.186.191   TCP        60 19305 → 60672 [PSH,
   42421 69.247769   142.250.82.44    172.20.186.191   TCP       146 19305 → 60672 [PSH,
   42422 69.247831   172.20.186.191   142.250.82.44    TCP        54 60672 → 19305 [ACK]
   42742 69.657034   142.250.82.44    172.20.186.191   TCP        60 19305 → 60672 [PSH,
   42743 69.657034   142.250.82.44    172.20.186.191   TCP       118 19305 → 60672 [PSH,
   42748 69.657034   142.250.82.44    172.20.186.191   TCP       118 [TCP Retransmission]
   42749 69.657294   172.20.186.191   142.250.82.44    TCP        66 60672 → 19305 [ACK]
   42998 70.008873   142.250.82.44    172.20.186.191   TCP        60 19305 → 60672 [PSH,
   42999 70.009752   142.250.82.44    172.20.186.191   TCP       119 19305 → 60672 [PSH,
   43000 70.009797   172.20.186.191   142.250.82.44    TCP        54 60672 → 19305 [ACK]
   43001 70.010107   172.20.186.191   142.250.82.44    TCP       121 60672 → 19305 [PSH,
   43002 70.053223   142.250.82.44    172.20.186.191   TCP        60 19305 → 60672 [ACK]
   43077 70.111400   172.20.186.191   142.250.82.44    TCP       120 60672 → 19305 [PSH,
   43116 70.149227   142.250.82.44    172.20.186.191   TCP        60 19305 → 60672 [ACK]
   43247 70.589462   142.250.82.44    172.20.186.191   TCP        60 19305 → 60672 [PSH,
   43248 70.590085   142.250.82.44    172.20.186.191   TCP       118 19305 → 60672 [PSH,
   43249 70.590131   172.20.186.191   142.250.82.44    TCP        54 60672 → 19305 [ACK]
   43515 70.963295   172.20.186.191   142.250.82.44    TCP       136 60672 → 19305 [PSH,
   43668 71.080302   142.250.82.44    172.20.186.191   TCP        60 19305 → 60672 [ACK]
   43717 71.219969   172.20.186.191   142.250.82.44    TCP       120 60672 → 19305 [PSH,
   43719 71.259262   142.250.82.44    172.20.186.191   TCP        60 19305 → 60672 [ACK]
   43912 71.667377   142.250.82.44    172.20.186.191   TCP        60 19305 → 60672 [PSH,

wireshark_Wi-Fi3SJVP1.pcapng    Packets: 43928 · Displayed: 365 (0.8%)    Profile: Default
```

network-exp10-packet-tracer-rightside

Packet Tracer

Aim:

Design and configure a network with multiple subnets with wired LANs using required network devices. Configure commonly used services in the network.

Procedure:

Router 1:

```
Router>enable
Router#config terminal
Router(config)#interface fa0/0
Router(config-if)#ip address 192.168.10.1 255.255.255.0
Router(config-if)#no shutdown
Router(config-if)#exit
Router(config)#interface s0/0/0
Router(config-if)#ip address 222.222.22.1 255.255.255.0
Router(config-if)#clock rate 64000
Router(config-if)#no shutdown
Router(config-if)#exit
Router(config)#exit
Router#copy running-config startup-config

Router>enable
Router#config terminal
Router(config)#router rip
Router(config-router)#network 192.168.10.0
Router(config-router)#network 222.222.22.0
Router(config-router)#exit
Router(config)#exit
Router#copy running-config startup-config
```

Router 2:

```
Router>enable
Router#config terminal
Router(config)#interface fa0/0
Router(config-if)#ip address 192.168.20.1 255.255.255.0
Router(config-if)#no shutdown
Router(config-if)#exit
Router(config)#interface s0/0/0
Router(config-if)#ip address 222.222.22.2 255.255.255.0
Router(config-if)#clock rate 64000
Router(config-if)#no shutdown
Router(config-if)#exit
Router(config)#exit
Router#copy running-config startup-config

Router>enable
Router#config terminal
Router(config)#router rip
Router(config-router)#network 192.168.20.0
Router(config-router)#network 222.222.22.0
Router(config-router)#exit
Router(config)#exit
Router#copy running-config startup-config
```

Result:

Successfully designed and configured a network with multiple subnets using network devices.