YourRDTF: Application Layer Reliable Data Transfer Protocol
Experiment No: AV-341-2025-Lab-9

Saurabh Kumar
SC22B146
April 17, 2025

Date and Time of experiment: April 16, 2025, 15:00 IST

Objectives

- To create an unreliable transport layer protocol session by establishing a User Datagram Protocol (UDP) client server session and simulate packet loss over that UDP session.
- To create a simple Application Layer protocol for reliable data transfer over the unreliable UDP transport.
- The system should contain one client and one server.
- The request and response packets should follow the following packet format.

8 bits	1 bit	8 bits	8 bits	20 Bytes
S No.	Type	Src. port no.	D. port no.	Data

  - S. No.: Packet sequence number
  - Type:
    - 0 - Data packet (message from client)
    - 1 - Acknowledgment packet (ACK from server)
  - Src. port no: Source port number
  - D. port no.: Destination port number
  - Data: Message to send

- **Description of the Experiment:**
  You are required to develop a C-based application layer implementation as part of YourRDTF, a reliable data transfer (RDT) protocol. Create a C-based receiver (server) application that receives packets from a C-based application layer sender (client).

Tools Used

- PC: 12th Gen Intel(R) Core(TM) i5-1240P 1.70 GHz, Windows 11 / Ubuntu 24.04 dual-booted, 64-bit, (reduced to) 4 GB RAM
- OS: Ubuntu 24.04
- Software used: VS Code, VS-Code in-built bash terminal, gcc compiler

Phase 1 (UDP session)

- Establish a single client-server UPD session.
- Use the given packet format for the UDP session.
- In the UDP session, send n packets.

Procedure

1. This part is similar to the lab-5 UDP experiment. The packet is made of C Struct in the given format.

```
struct packet
{
    uint8_t seq_no;
    uint8_t type;
    uint8_t src_port;
    uint8_t dest_port;
    char data[20];
};
```

2. On the client side, the user is prompted to enter data for 10 packets which are sent to the server in loop.

```
struct packet pkt;
char input_buf[100]; // user input buffer
for (int i = 0; i < N; i++)
{
    printf("Enter message %d: ", i + 1);
    fgets(input_buf, sizeof(input_buf), stdin);
    input_buf[strcspn(input_buf, "\n")] = '\0';

    pkt.seq_no = i;
    pkt.type = 0;
    pkt.src_port = CLIENTPORT & 0xFF;
    pkt.dest_port = SERVERPORT & 0xFF;
    strncpy(pkt.data, input_buf, sizeof(pkt.data));

    // Send packet
    bytes_sent = sendto(sock, &pkt, sizeof(pkt), 0, struct sockaddr
    *)&s_server, si_len);
    if (bytes_sent < 0)
    {
        printf("Error sending data\n");
        return 1;
    }
    printf("Sent packet %d: %s\n", pkt.seq_no, pkt.data);
}
```

   Note that the port no.s are 16 bit but are truncated to LSB 8 bit as per the design requirement.

3. The server is simply receiving the packets in while loop.

```c
struct packet pkt;
while ((bytes_received = recvfrom(sock, &pkt, sizeof(pkt), 0, (
    struct sockaddr *)&s_client, &si_len)) > 0)
{
     printf("Received packet:\n");
     printf("Seq No: %d, Type: %d, Src Port: %d, Dest Port: %d, Data
    : %s\n", pkt.seq_no, pkt.type, pkt.src_port, pkt.dest_port, pkt.
    data);
}
```

Observations

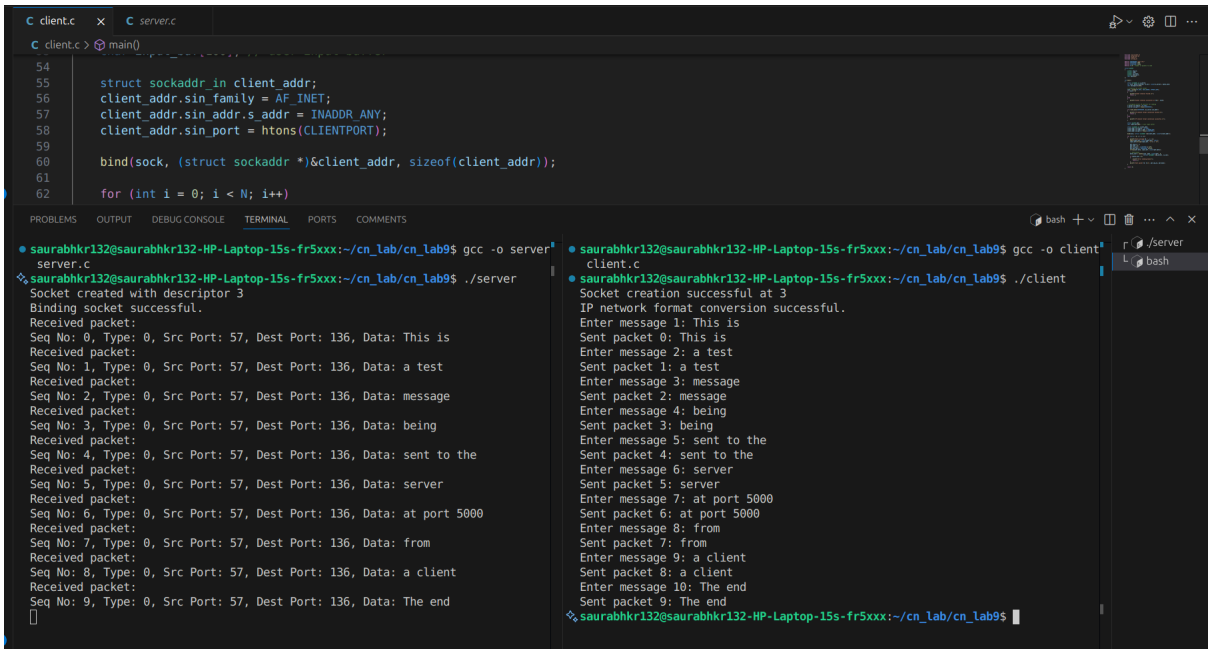

Figure 1: Server-client UDP Communication

Phase 2 (Implementation of acknowledgment)

- The server receives data packets from the client and sends out an ACK for each packet correctly received.
- The client, after each packet transmission, may set a timer with a certain reasonable timeout value to receive acknowledgments in response to the data packets.
- The client waits for acknowledgment from server, if received correctly, the client sends the next packet.
- If an acknowledgement is not received within that time, the packet is lost and is retransmitted. If an acknowledgment does arrive within the timeout, the timer is cancelled.

Procedure

1. The timeout on client side is implemented using `select()` system call, which waits for activity on file descriptors (like sockets). `fd_set` watches the socket for readability.

```
int ack_received = 0;

while (!ack_received) {
    bytes_sent = sendto(sock, &pkt, sizeof(pkt), 0, (struct
   sockaddr *)&s_server, si_len);
    printf("Sent packet %d: %s\n", pkt.seq_no, pkt.data);

    // Set timeout
    fd_set readfds;
    struct timeval timeout;

    FD_ZERO(&readfds);
    FD_SET(sock, &readfds);
    timeout.tv_sec = 5;

    int ready = select(sock + 1, &readfds, NULL, NULL, &timeout);

    if (ready == -1) {
        perror("select error");
        continue;
    } else if (ready == 0) { // timeout
        printf("Timeout! Retrying packet %d...\n", pkt.seq_no);
    } else {
        bytes_received = recvfrom(sock, &server_pkt, sizeof(
   server_pkt), 0, (struct sockaddr *)&s_server, &si_len);
        if (bytes_received > 0 && server_pkt.seq_no == pkt.seq_no
   && server_pkt.type == 1) {
            printf("ACK received for packet %d\n", pkt.seq_no);
            ack_received = 1;
        } else {
            printf("Incorrect ACK or packet corrupted. Retrying...\
   n");
        }
    }
}
```

2. The server is receiving the packet, copies it and changes its type ('1' to indicate acknowledgement) and swap the ports, and send it back to the client.

```
struct packet server_pkt;
struct packet pkt;

while ((bytes_received = recvfrom(sock, &pkt, sizeof(pkt), 0, (
   struct sockaddr *)&s_client, &si_len)) > 0)
{
    printf("Received packet:\n");
    printf("Seq No: %d, Type: %d, Src Port: %d, Dest Port: %d, Data
   : %s\n", pkt.seq_no, pkt.type, pkt.src_port, pkt.dest_port, pkt.
   data);
    memcpy(&server_pkt, &pkt, sizeof(pkt));
    server_pkt.type = 1;
```

```
    server_pkt.dest_port = pkt.src_port;
    server_pkt.src_port = pkt.dest_port;
    sendto(sock, &server_pkt, sizeof(server_pkt), 0, (struct
   sockaddr *)&s_client, si_len);
}
```

Observations

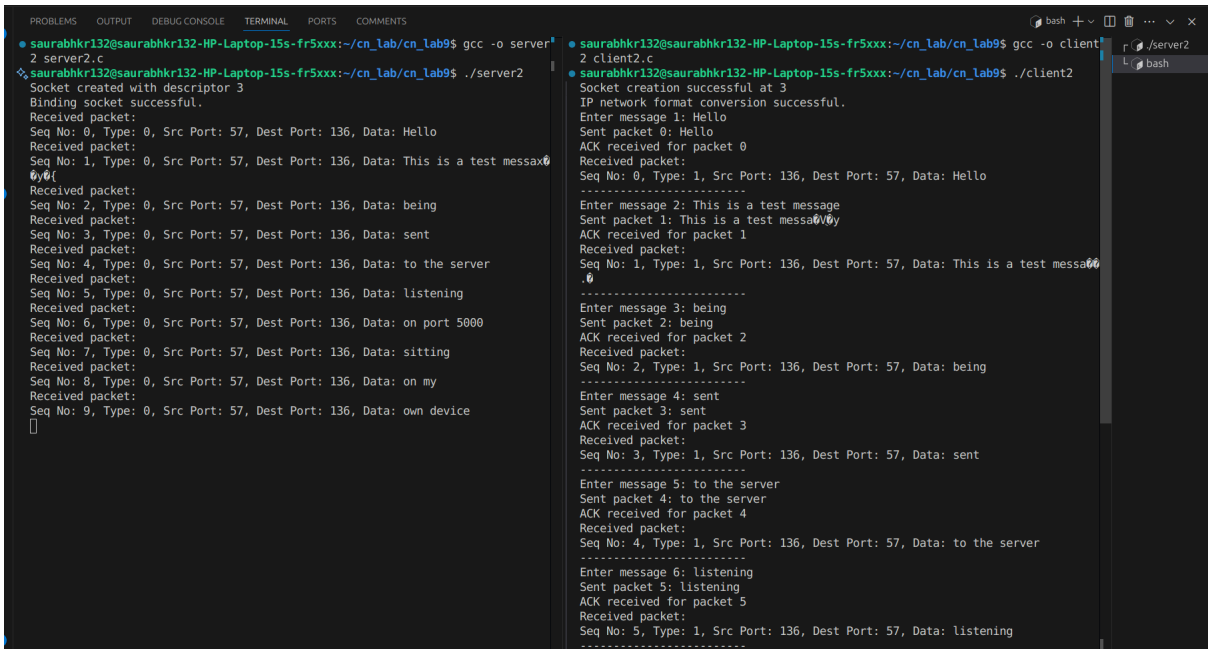

Figure 2: Server-client UDP Communication with ACK

Phase 3 (Simulate packet loss)

- Do the phase-2 with a packet loss mechanism and through acknowledgment retransmit the lost packets.
- To simulate packet loss, the client may reject/drop random packets. Randomly, one out of five packets (20% of packets) may be discarded by the client.
- When a packet is rejected, the server may timeout and retransmit the packet.
- The client needs to display all the events on screen, such as receive packets, discarded packets, retransmission of packets, and acknowledgments.
- Completion of phase-3 with above instructions.

Procedure

1. The packet simulation is simulated on the server side. The packet loss in real-life happen in between the server and the client, so its better to simulate it on the server which is receiving the packets and retransmitting with acknowledgement. This is done using rand() function. As a result, the client can't receive back few packets with ACK. Rest everything will be the same as the phase-2.

```
if (rand() % 5 == 0) {
    printf("Simulated packet loss: Dropped packet %d\n", pkt.seq_no
    );
    continue;
}
```

Observations

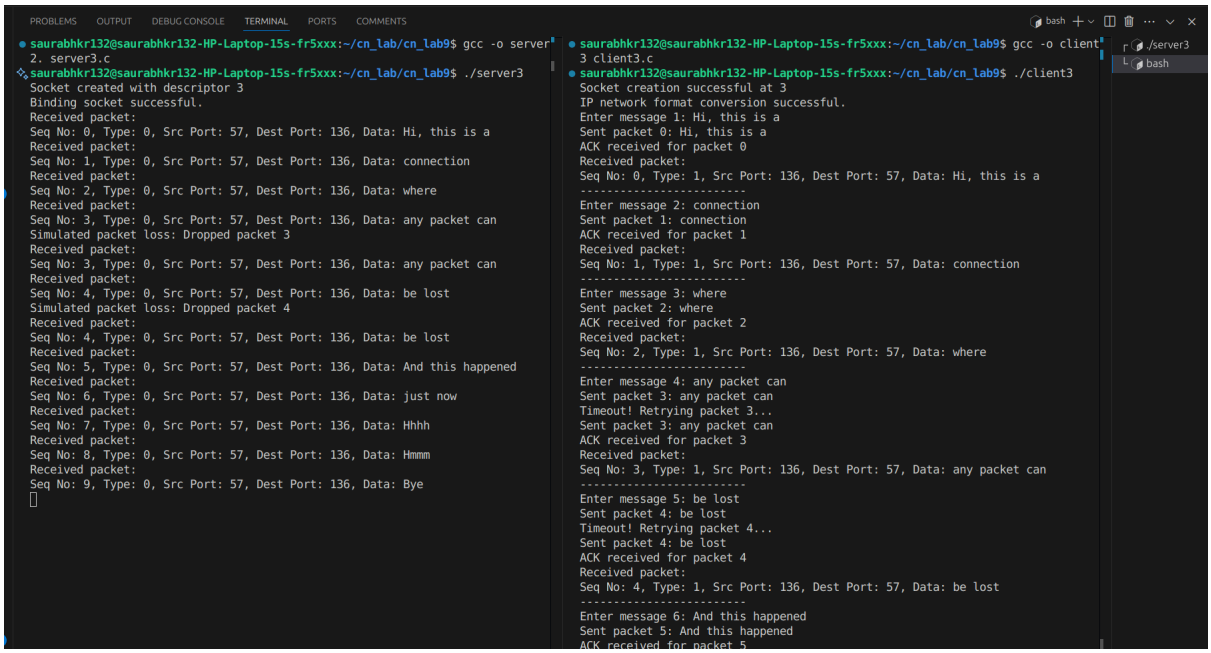

Figure 3: Server-client UDP Communication with packet loss simulation

Conclusions

- The experiment demonstrated a custom implementation of a reliable data transfer protocol over an unreliable UDP transport layer. In Phase 1, a basic client-server communication was established using UDP sockets, following a custom packet format that included sequence number, type, source & destination ports, and data.
- In Phase 2, reliability was introduced by implementing acknowledgments from the server and a retransmission mechanism in the client using timeouts. This ensured that lost packets were detected and resent.
- Finally, in Phase 3, packet loss was simulated on the server side with a 20% drop probability to mimic network unreliability, and the client correctly retransmitted dropped packets based on timeout and acknowledgment logic.
- The experiment implemented key networking concepts such as connectionless communication, reliable delivery at the application layer, and retransmission logic for error handling.

Code Blocks

Commands

To compile the C-file:

```
gcc -o output_file_name file_to_compile.c
```

To run the compiled binary file:

```
./output_file_name
```

Phase 1

Server: (server.c)

```c
#include <sys/socket.h>
#include <arpa/inet.h>
#include <string.h>
#include <stdio.h>

#define PORT 5000

struct packet
{
    uint8_t seq_no;
    uint8_t type;
    uint8_t src_port;
    uint8_t dest_port;
    char data[20];
};

int main()
{
    int sock, recv_calls = 0;
    int bind_status, bytes_received, bytes_sent;

    struct sockaddr_in s_server, s_client;
    int si_len = sizeof(s_client);

    // Creating the socket
    sock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
    if (sock < 0)
    {
        printf("Socket creation failed.\n");
    }
    else
    {
        printf("Socket created with descriptor %d\n", sock);
    }

    // Initializing and binding the socket
    s_server.sin_family = AF_INET;
    s_server.sin_port = htons(PORT);
    s_server.sin_addr.s_addr = htonl(INADDR_ANY);
```

```
    bind_status = bind(sock, (struct sockaddr *)&s_server, sizeof(
   s_server));
    if (bind_status < 0)
    {
        printf("Binding socket failed.\n");
    }
    else
    {
        printf("Binding socket successful.\n");
    }

    struct packet pkt;
    while ((bytes_received = recvfrom(sock, &pkt, sizeof(pkt), 0, (
   struct sockaddr *)&s_client, &si_len)) > 0)
    {
        printf("Received packet:\n");
        printf("Seq No: %d, Type: %d, Src Port: %d, Dest Port: %d, Data
   : %s\n", pkt.seq_no, pkt.type, pkt.src_port, pkt.dest_port, pkt.data
   );
    }

    return 0;
}
```

Client: (client.c)

```
#include <sys/socket.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>

#define SERVERADDR "127.0.0.1"
#define SERVERPORT 5000
#define CLIENTPORT 12345
#define N 10 // Number of packets to send

struct packet
{
    uint8_t seq_no;
    uint8_t type;
    uint8_t src_port;
    uint8_t dest_port;
    char data[20];
};

int main()
{
    struct sockaddr_in s_server;
    int sock, bytes_received, si_len = sizeof(s_server), bytes_sent;

    // Creating the client socket
    sock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
    if (sock < 0)
    {
        printf("Socket creation failed.\n");
        return 1;
    }
    else
```

```
    {
        printf("Socket creation successful at %d\n", sock);
    }

    // Assigning server IP and port for sending
    s_server.sin_family = AF_INET;
    s_server.sin_port = htons(SERVERPORT);

    if (!inet_aton(SERVERADDR, &s_server.sin_addr))
    {
        printf("IP network format conversion failed.\n");
        return 1;
    }
    else
    {
        printf("IP network format conversion successful.\n");
    }

    struct packet pkt;
    char input_buf[100]; // user input buffer

    struct sockaddr_in client_addr;
    client_addr.sin_family = AF_INET;
    client_addr.sin_addr.s_addr = INADDR_ANY;
    client_addr.sin_port = htons(CLIENTPORT);

    bind(sock, (struct sockaddr *)&client_addr, sizeof(client_addr));

    for (int i = 0; i < N; i++)
    {
        printf("Enter message %d: ", i + 1);
        fgets(input_buf, sizeof(input_buf), stdin);
        input_buf[strcspn(input_buf, "\n")] = '\0';

        pkt.seq_no = i;
        pkt.type = 0;
        pkt.src_port = CLIENTPORT & 0xFF;
        pkt.dest_port = SERVERPORT & 0xFF;
        strncpy(pkt.data, input_buf, sizeof(pkt.data));

        // Send packet
        bytes_sent = sendto(sock, &pkt, sizeof(pkt), 0, (struct
   sockaddr *)&s_server, si_len);
        if (bytes_sent < 0)
        {
            printf("Error sending data\n");
            return 1;
        }
        printf("Sent packet %d: %s\n", pkt.seq_no, pkt.data);
    }

    return 0;
}
```

Phase 2

Server: (server2.c)

```c
#include <sys/socket.h>
#include <arpa/inet.h>
#include <string.h>
#include <stdio.h>

#define PORT 5000

struct packet
{
    uint8_t seq_no;
    uint8_t type;
    uint8_t src_port;
    uint8_t dest_port;
    char data[20];
};

int main()
{
    int sock, recv_calls = 0;
    int bind_status, bytes_received, bytes_sent;

    struct sockaddr_in s_server, s_client;
    int si_len = sizeof(s_client);

    // Creating the socket
    sock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
    if (sock < 0)
    {
        printf("Socket creation failed.\n");
    }
    else
    {
        printf("Socket created with descriptor %d\n", sock);
    }

    // Initializing and binding the socket
    s_server.sin_family = AF_INET;
    s_server.sin_port = htons(PORT);
    s_server.sin_addr.s_addr = htonl(INADDR_ANY);

    bind_status = bind(sock, (struct sockaddr *)&s_server, sizeof(
   s_server));
    if (bind_status < 0)
    {
        printf("Binding socket failed.\n");
    }
    else
    {
        printf("Binding socket successful.\n");
    }

    struct packet server_pkt;
    struct packet pkt;

    while ((bytes_received = recvfrom(sock, &pkt, sizeof(pkt), 0, (
   struct sockaddr *)&s_client, &si_len)) > 0)
    {
        printf("Received packet:\n");
```

```
        printf("Seq No: %d, Type: %d, Src Port: %d, Dest Port: %d, Data
    : %s\n", pkt.seq_no, pkt.type, pkt.src_port, pkt.dest_port, pkt.data
    );
        memcpy(&server_pkt, &pkt, sizeof(pkt));
        server_pkt.type = 1;
        server_pkt.dest_port = pkt.src_port;
        server_pkt.src_port = pkt.dest_port;
        sendto(sock, &server_pkt, sizeof(server_pkt), 0, (struct
    sockaddr *)&s_client, si_len);
    }

    return 0;
}
```

Client: (client2.c)

```
#include <sys/socket.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>
#include <sys/time.h>

#define SERVERADDR "127.0.0.1"
#define SERVERPORT 5000
#define CLIENTPORT 12345
#define N 10 // Number of packets to send

struct packet
{
    uint8_t seq_no;
    uint8_t type;
    uint8_t src_port;
    uint8_t dest_port;
    char data[20];
};

int main()
{
    struct sockaddr_in s_server;
    int sock, bytes_received, si_len = sizeof(s_server), bytes_sent;

    struct timeval start, end;

    // Creating the client socket
    sock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
    if (sock < 0)
    {
        printf("Socket creation failed.\n");
        return 1;
    }
    else
    {
        printf("Socket creation successful at %d\n", sock);
    }

    // Assigning server IP and port for sending
    s_server.sin_family = AF_INET;
    s_server.sin_port = htons(SERVERPORT);
```

```
    if (!inet_aton(SERVERADDR, &s_server.sin_addr))
    {
        printf("IP network format conversion failed.\n");
        return 1;
    }
    else
    {
        printf("IP network format conversion successful.\n");
    }

    struct packet pkt;
    struct packet server_pkt;
    char input_buf[100];

    struct sockaddr_in client_addr;
    client_addr.sin_family = AF_INET;
    client_addr.sin_addr.s_addr = INADDR_ANY;
    client_addr.sin_port = htons(CLIENTPORT);

    bind(sock, (struct sockaddr *)&client_addr, sizeof(client_addr));

    for (int i = 0; i < N; i++)
    {
        printf("Enter message %d: ", i + 1);
        fgets(input_buf, sizeof(input_buf), stdin);
        input_buf[strcspn(input_buf, "\n")] = '\0';

        pkt.seq_no = i;
        pkt.type = 0;
        pkt.src_port = CLIENTPORT & 0xFF;
        pkt.dest_port = SERVERPORT & 0xFF;
        strncpy(pkt.data, input_buf, sizeof(pkt.data));

        int ack_received = 0;

        while (!ack_received) {
            bytes_sent = sendto(sock, &pkt, sizeof(pkt), 0, (struct
sockaddr *)&s_server, si_len);
            printf("Sent packet %d: %s\n", pkt.seq_no, pkt.data);

            // Set timeout
            fd_set readfds;
            struct timeval timeout;

            FD_ZERO(&readfds);
            FD_SET(sock, &readfds);
            timeout.tv_sec = 5;

            int ready = select(sock + 1, &readfds, NULL, NULL, &timeout
);

            if (ready == -1) {
                perror("select error");
                continue;
            } else if (ready == 0) { // timeout
                printf("Timeout! Retrying packet %d...\n", pkt.seq_no);
            } else {
                bytes_received = recvfrom(sock, &server_pkt, sizeof(
```

```
    server_pkt), 0, (struct sockaddr *)&s_server, &si_len);
                if (bytes_received > 0 && server_pkt.seq_no == pkt.
    seq_no && server_pkt.type == 1) {
                    printf("ACK received for packet %d\n", pkt.seq_no);
                    ack_received = 1;
                } else {
                    printf("Incorrect ACK or packet corrupted. Retrying
    ...\n");
                }
            }
        }

        printf("Received packet:\n");
        printf("Seq No: %d, Type: %d, Src Port: %d, Dest Port: %d, Data
    : %s\n",
            server_pkt.seq_no, server_pkt.type, server_pkt.src_port,
    server_pkt.dest_port, server_pkt.data);
        printf("-------------------------\n");
     }

     return 0;
}
```

Phase 3

Server: (server3.c)

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

#define PORT 5000

struct packet
{
    uint8_t seq_no;
    uint8_t type;
    uint8_t src_port;
    uint8_t dest_port;
    char data[20];
};

int main()
{
    int sock, recv_calls = 0;
    int bind_status, bytes_received, bytes_sent;

    struct sockaddr_in s_server, s_client;
    int si_len = sizeof(s_client);

    // Creating the socket
    sock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
    if (sock < 0)
    {
        printf("Socket creation failed.\n");
```

```
    }
    else
    {
        printf("Socket created with descriptor %d\n", sock);
    }

    // Initializing and binding the socket
    s_server.sin_family = AF_INET;
    s_server.sin_port = htons(PORT);
    s_server.sin_addr.s_addr = htonl(INADDR_ANY);

    bind_status = bind(sock, (struct sockaddr *)&s_server, sizeof(
   s_server));
    if (bind_status < 0)
    {
        printf("Binding socket failed.\n");
    }
    else
    {
        printf("Binding socket successful.\n");
    }

    struct packet server_pkt;
    struct packet pkt;

    while ((bytes_received = recvfrom(sock, &pkt, sizeof(pkt), 0, (
   struct sockaddr *)&s_client, &si_len)) > 0)
    {
        printf("Received packet:\n");
        printf("Seq No: %d, Type: %d, Src Port: %d, Dest Port: %d, Data
   : %s\n", pkt.seq_no, pkt.type, pkt.src_port, pkt.dest_port, pkt.data
   );
        memcpy(&server_pkt, &pkt, sizeof(pkt));
        server_pkt.type = 1;
        server_pkt.dest_port = pkt.src_port;
        server_pkt.src_port = pkt.dest_port;

        if (rand() % 5 == 0) {
            printf("Simulated packet loss: Dropped packet %d\n", pkt.
   seq_no);
            continue;
        }
        sendto(sock, &server_pkt, sizeof(server_pkt), 0, (struct
   sockaddr *)&s_client, si_len);
    }

    return 0;
}
```

Client: (client3.c)

```
#include <sys/socket.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>
#include <sys/time.h>

#define SERVERADDR "127.0.0.1"
#define SERVERPORT 5000
```

```c
#define CLIENTPORT 12345
#define N 10 // Number of packets to send

struct packet
{
    uint8_t seq_no;
    uint8_t type;
    uint8_t src_port;
    uint8_t dest_port;
    char data[20];
};

int main()
{
    struct sockaddr_in s_server;
    int sock, bytes_received, si_len = sizeof(s_server), bytes_sent;

    struct timeval start, end;

    // Creating the client socket
    sock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
    if (sock < 0)
    {
        printf("Socket creation failed.\n");
        return 1;
    }
    else
    {
        printf("Socket creation successful at %d\n", sock);
    }

    // Assigning server IP and port for sending
    s_server.sin_family = AF_INET;
    s_server.sin_port = htons(SERVERPORT);

    if (!inet_aton(SERVERADDR, &s_server.sin_addr))
    {
        printf("IP network format conversion failed.\n");
        return 1;
    }
    else
    {
        printf("IP network format conversion successful.\n");
    }

    struct packet pkt;
    struct packet server_pkt;
    char input_buf[100];
    uint16_t client_port;

    struct sockaddr_in client_addr;
    client_addr.sin_family = AF_INET;
    client_addr.sin_addr.s_addr = INADDR_ANY;
    client_addr.sin_port = htons(CLIENTPORT);

    bind(sock, (struct sockaddr *)&client_addr, sizeof(client_addr));

    for (int i = 0; i < N; i++)
```

```
    {
        printf("Enter message %d: ", i + 1);
        fgets(input_buf, sizeof(input_buf), stdin);
        input_buf[strcspn(input_buf, "\n")] = '\0';

        pkt.seq_no = i;
        pkt.type = 0;
        pkt.src_port = CLIENTPORT & 0xFF;
        pkt.dest_port = SERVERPORT & 0xFF;
        strncpy(pkt.data, input_buf, sizeof(pkt.data));

        int ack_received = 0;

        while (!ack_received) {
            bytes_sent = sendto(sock, &pkt, sizeof(pkt), 0, (struct
    sockaddr *)&s_server, si_len);
            printf("Sent packet %d: %s\n", pkt.seq_no, pkt.data);

            // Set timeout
            fd_set readfds;
            struct timeval timeout;

            FD_ZERO(&readfds);
            FD_SET(sock, &readfds);
            timeout.tv_sec = 5;
            timeout.tv_usec = 0;

            int ready = select(sock + 1, &readfds, NULL, NULL, &timeout
    );

            if (ready == -1) {
                perror("select error");
                continue;
            } else if (ready == 0) { // timeout
                printf("Timeout! Retrying packet %d...\n", pkt.seq_no);
            } else {
                bytes_received = recvfrom(sock, &server_pkt, sizeof(
    server_pkt), 0, (struct sockaddr *)&s_server, &si_len);
                if (bytes_received > 0 && server_pkt.seq_no == pkt.
    seq_no && server_pkt.type == 1) {
                    printf("ACK received for packet %d\n", pkt.seq_no);
                    ack_received = 1;
                } else {
                    printf("Incorrect ACK or packet corrupted. Retrying
    ...\n");
                }
            }
        }

        printf("Received packet:\n");
        printf("Seq No: %d, Type: %d, Src Port: %d, Dest Port: %d, Data
    : %s\n", server_pkt.seq_no, server_pkt.type, server_pkt.src_port,
    server_pkt.dest_port, server_pkt.data);
        printf("-------------------------\n");
    }

    return 0;
}
```