Pradeep KM
3122225001092

Exercise 7 : Flow Control

Server algorithm

1. Initialize socket and bind to address.
2. Continuously receive packets using recvfrom().
3. If packet sequence number matches expected_seq, increment expected_seq.
4. If out of sequence, print expected sequence number.
5. Send ACK (last correctly received sequence) using sendto().

Client algorithm

1. Initialize socket and server address.
2. Input data, source IP, and destination IP.
3. Calculate total packets based on data size.
4. Set window_start and window_end based on WINDOW_SIZE.
5. For each packet in the window:
    a. Extract data chunk.
    b. Create a packet with sequence number, source IP, and destination IP.
    c. Ask user if packet should be sent.
    d. If "Y", send packet via sendto().
6. Wait for ACK using recvfrom().
7. If ACK >= window_start, update window_start and window_end.
8. Ask if transmission should end. If "Y", exit.
9. Close socket.

server.c

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

#define PORT 8080
#define MAX_PACKET_SIZE 4

typedef struct {
    char src_ip[16];              // Source IP address as a string
    char dest_ip[16];             // Destination IP address as a string
    int sequence_number;          // Sequence number
    char data[MAX_PACKET_SIZE + 1]; // Data + 1 for null terminator
    char fcs;                     // Frame Check Sequence (dummy for now)
} Packet;

int main() {
    int sockfd;
    struct sockaddr_in server_addr, client_addr;
    socklen_t addr_size;
```

```
    Packet packet;
    int expected_seq = 0, ack;

    sockfd = socket(AF_INET, SOCK_DGRAM, 0);

    memset(&server_addr, 0, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(PORT);
    server_addr.sin_addr.s_addr = INADDR_ANY;

    bind(sockfd, (struct sockaddr*)&server_addr, sizeof(server_addr));
    addr_size = sizeof(client_addr);

    while (1) {
        recvfrom(sockfd, &packet, sizeof(Packet), 0, (struct sockaddr *)&client_addr, &addr_size);
        printf("Received Packet: Seq %d, Data %s\n", packet.sequence_number, packet.data);

        if (packet.sequence_number == expected_seq) {
            printf("Packet %d is in sequence.\n", packet.sequence_number);
            expected_seq++;
        } else {
            printf("Packet %d is out of sequence, expecting %d.\n", packet.sequence_number,
expected_seq);
        }

        ack = expected_seq - 1;
        sendto(sockfd, &ack, sizeof(ack), 0, (struct sockaddr *) &client_addr, addr_size);
        printf("Sent ACK %d\n", ack);
    }

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

client.c

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

#define PORT 8080
#define MAX_PACKET_SIZE 4
#define WINDOW_SIZE 4

typedef struct {
    char src_ip[16];              // Source IP address as a string
    char dest_ip[16];             // Destination IP address as a string
    int sequence_number;          // Sequence number
    char data[MAX_PACKET_SIZE + 1]; // Data + 1 for null terminator
```

```
    char fcs;                    // Frame Check Sequence (dummy for now)
} Packet;

void create_packet(Packet *packet, int seq, const char *src, const char *dest, const char *data) {
    strcpy(packet->src_ip, src);
    strcpy(packet->dest_ip, dest);
    packet->sequence_number = seq;
    strncpy(packet->data, data, MAX_PACKET_SIZE);
    packet->data[MAX_PACKET_SIZE] = '\0';
    packet->fcs = 'F';
}

void send_packet(int sockfd, struct sockaddr_in server_addr, Packet *packet) {
    sendto(sockfd, packet, sizeof(Packet), 0, (struct sockaddr *) &server_addr, sizeof(server_addr));
}

int main() {
    int sockfd;
    struct sockaddr_in server_addr;
    socklen_t addr_size;
    Packet packet;
    char data[16];
    char src_ip[16], dest_ip[16];
    int window_start = 0, window_end = WINDOW_SIZE - 1, seq = 0, ack;

    sockfd = socket(AF_INET, SOCK_DGRAM, 0);

    memset(&server_addr, 0, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(PORT);
    server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
    printf("Enter the data to send (in 8-bit chunks): ");
    scanf("%s", data);

    printf("Enter source IP address: ");
    scanf("%s", src_ip);

    printf("Enter destination IP address: ");
    scanf("%s", dest_ip);

    int total_packets = (strlen(data) + MAX_PACKET_SIZE - 1) / MAX_PACKET_SIZE;

    while (window_start < total_packets) {
        for (seq = window_start; seq <= window_end && seq < total_packets; seq++) {
            char packet_data[MAX_PACKET_SIZE + 1] = {0};
            strncpy(packet_data, data + seq * MAX_PACKET_SIZE, MAX_PACKET_SIZE);

            create_packet(&packet, seq, src_ip, dest_ip, packet_data);

            printf("Send packet %d (Y/N)? ", seq);
            char send_decision;
```

```
            scanf(" %c", &send_decision);

            if (send_decision == 'Y' || send_decision == 'y') {
                send_packet(sockfd, server_addr, &packet);
                printf("Sent Packet: Seq %d, Data %s\n", packet.sequence_number, packet.data);
            } else {
                printf("Packet %d not sent.\n", seq);
            }
        }

        recvfrom(sockfd, &ack, sizeof(ack), 0, (struct sockaddr *)&server_addr, &addr_size);
        printf("Received ACK %d\n", ack);

        if (ack >= window_start) {
            window_start = ack + 1;
            window_end = window_start + WINDOW_SIZE - 1;
        }

        printf("End transmission (Y/N)? ");
        char end_decision;
        scanf(" %c", &end_decision);

        if (end_decision == 'Y' || end_decision == 'y') {
            break;
        }
    }
    close(sockfd);
    return 0;
}
```

Output