ARUN M
R6B 76

Networking lab assignment

I. a) Implement Stop-and-Wait ARQ flow control protocol.

server

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

#define PORT 12345
#define BUFFER_SIZE 1024

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

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

    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");
        exit(EXIT_FAILURE);
    }

    if (listen(server_fd, 3) < 0) {
        perror("Listen failed");
```

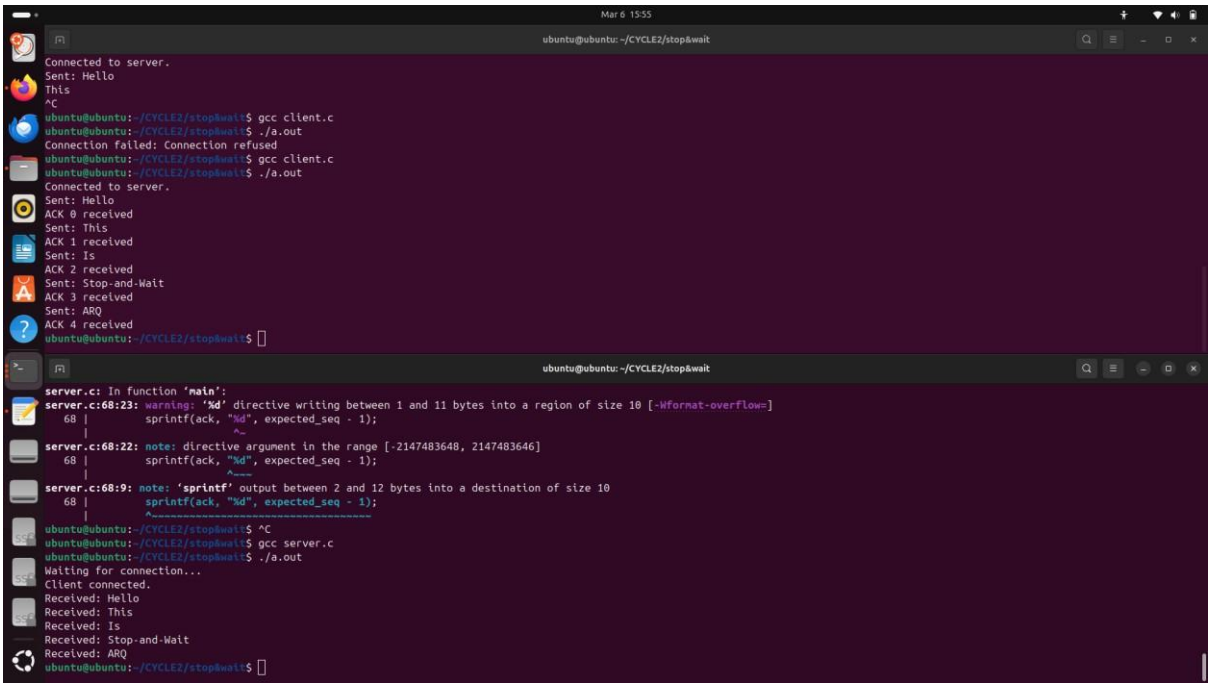
```
        exit(EXIT_FAILURE);
    }

    printf("Waiting for connection...\n");
    new_socket = accept(server_fd, (struct sockaddr *)&address, (socklen_t*)&addrlen);
    if (new_socket < 0) {
        perror("Accept failed");
        exit(EXIT_FAILURE);
    }
    printf("Client connected.\n");

    while (1) {
        memset(buffer, 0, BUFFER_SIZE);
        int bytes_received = recv(new_socket, buffer, BUFFER_SIZE, 0);
        if (bytes_received <= 0) {
            break;
        }

        int seq_num;
        char message[BUFFER_SIZE];
        sscanf(buffer, "%d:%s", &seq_num, message);

        if (seq_num == expected_seq) {
            printf("Received: %s\n", message);
            expected_seq++;
        } else {
            printf("Duplicate packet detected, resending last ACK.\n");
        }

        char ack[12];
        sprintf(ack, "%d", expected_seq - 1);
        send(new_socket, ack, strlen(ack), 0);
    }

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

client

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

```
#define PORT 12345
#define SERVER_IP "127.0.0.1"
#define BUFFER_SIZE 1024

int main() {
    int sock;
    struct sockaddr_in server_addr;
    char buffer[BUFFER_SIZE];
    int seq_num = 0;
    char *messages[] = {"Hello", "This", "Is", "Stop-and-Wait", "ARQ"};
    int num_messages = sizeof(messages) / sizeof(messages[0]);


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

    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(PORT);
    inet_pton(AF_INET, SERVER_IP, &server_addr.sin_addr);

    if (connect(sock, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0) {
        perror("Connection failed");
        exit(EXIT_FAILURE);
    }
    printf("Connected to server.\n");

    for (int i = 0; i < num_messages; i++) {
        while (1) {
            sprintf(buffer, "%d:%s", seq_num, messages[i]);
            send(sock, buffer, strlen(buffer), 0);
            printf("Sent: %s\n", messages[i]);

            memset(buffer, 0, BUFFER_SIZE);
            recv(sock, buffer, BUFFER_SIZE, 0);

            int ack;
            sscanf(buffer, "%d", &ack);

            if (ack == seq_num) {
                printf("ACK %d received\n", ack);
                seq_num++;
                break;
            } else {
                printf("ACK mismatch, resending...\n");
```

```
            sleep(1);
        }
    }
  }

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




b) Implement Go-Back--N ARQ flow control protocol.
server

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

#define PORT 12346
#define SERVER_IP "127.0.0.1"
#define BUFFER_SIZE 1024
#define WINDOW_SIZE 3

int main() {
  int sock;
```

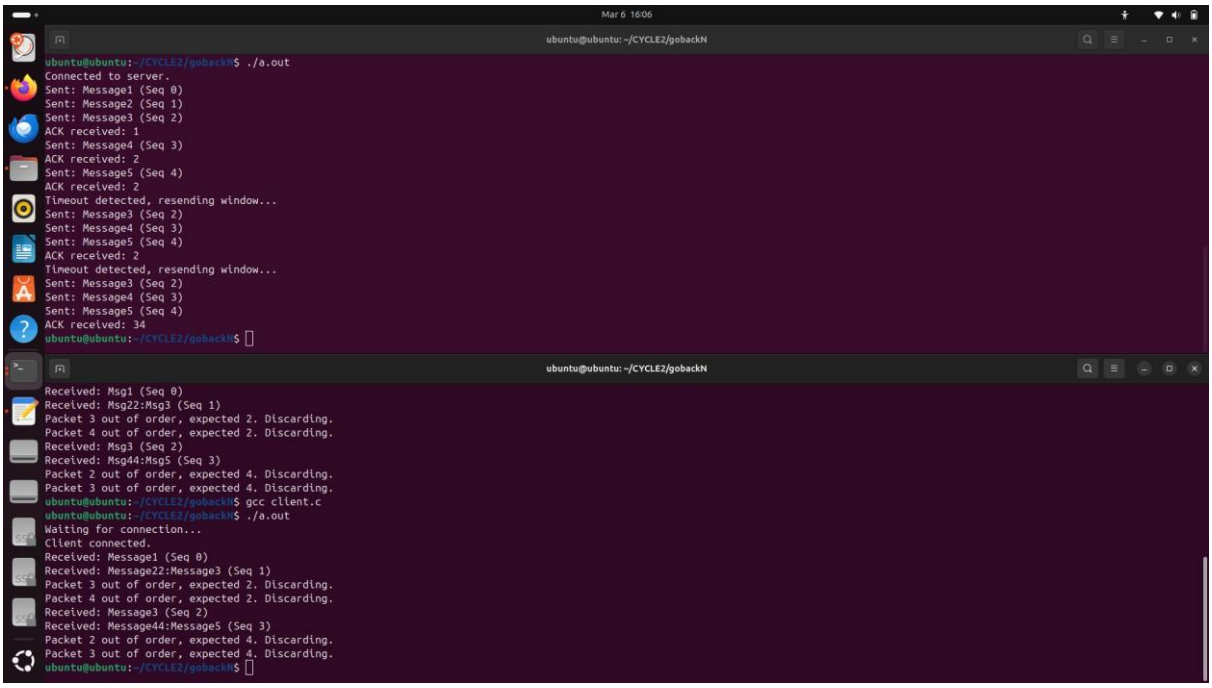
```
struct sockaddr_in server_addr;
char buffer[BUFFER_SIZE];
int base = 0, next_seq = 0;
char *messages[] = {"Message1", "Message2", "Message3", "Message4", "Message5"};
int num_messages = sizeof(messages) / sizeof(messages[0]);


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


server_addr.sin_family = AF_INET;
server_addr.sin_port = htons(PORT);
inet_pton(AF_INET, SERVER_IP, &server_addr.sin_addr);


if (connect(sock, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0) {
   perror("Connection failed");
   exit(EXIT_FAILURE);
}
printf("Connected to server.\n");

while (base < num_messages) {
   // Send packets within the window
   while (next_seq < base + WINDOW_SIZE && next_seq < num_messages) {
      sprintf(buffer, "%d:%s", next_seq, messages[next_seq]);
      send(sock, buffer, strlen(buffer), 0);
      printf("Sent: %s (Seq %d)\n", messages[next_seq], next_seq);
      next_seq++;
   }

   // ereceive ACK
   memset(buffer, 0, BUFFER_SIZE);
   recv(sock, buffer, BUFFER_SIZE, 0);
   int ack;
   sscanf(buffer, "%d", &ack);
   printf("ACK received: %d\n", ack);

   if (ack > base) {
      base = ack;
   } else {
      printf("Timeout detected, resending window...\n");
      next_seq = base; // retransmit from base
   }
   sleep(1);
```

```
    }

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

client

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

#define PORT 12346
#define BUFFER_SIZE 1024

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


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


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

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


    if (listen(server_fd, 3) < 0) {
        perror("Listen failed");
        exit(EXIT_FAILURE);
```

```
    }

    printf("Waiting for connection...\n");
    new_socket = accept(server_fd, (struct sockaddr *)&address, (socklen_t*)&addrlen);
    if (new_socket < 0) {
        perror("Accept failed");
        exit(EXIT_FAILURE);
    }
    printf("Client connected.\n");

    while (1) {
        memset(buffer, 0, BUFFER_SIZE);
        int bytes_received = recv(new_socket, buffer, BUFFER_SIZE, 0);
        if (bytes_received <= 0) {
            break;
        }

        int seq_num;
        char message[BUFFER_SIZE];
        sscanf(buffer, "%d:%s", &seq_num, message);

        if (seq_num == expected_seq) {
            printf("Received: %s (Seq %d)\n", message, seq_num);
            expected_seq++;
        } else {
            printf("Packet %d out of order, expected %d. Discarding.\n", seq_num, expected_seq);
        }

        char ack[10];
        sprintf(ack, "%d", expected_seq);
        send(new_socket, ack, strlen(ack), 0);
    }

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




c. Selective Repeat ARQ Protocol :

```
// server.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <time.h>
#include <arpa/inet.h>
#define PORT 9000
#define MAX 1024

int main() {
    int server_sock, client_sock;
    struct sockaddr_in server_addr, client_addr;
    socklen_t addr_size = sizeof(client_addr);
    char buffer[MAX];
    int expected_seq = 0;

    server_sock = socket(AF_INET, SOCK_STREAM, 0);
    if (server_sock < 0) {
        perror("Socket creation failed");
        exit(1);
    }

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

    bind(server_sock, (struct sockaddr*)&server_addr, sizeof(server_addr));
    listen(server_sock, 5);
    printf("Server listening on port %d...\n", PORT);

    client_sock = accept(server_sock, (struct sockaddr*)&client_addr, &addr_size);
    printf("Client connected.\n");
```

```
    srand(time(NULL));
    while (1) {
        memset(buffer, 0, sizeof(buffer));
        recv(client_sock, buffer, sizeof(buffer), 0);
        if (strcmp(buffer, "END") == 0) break;

        int seq;
        sscanf(buffer, "SEQ:%d", &seq);

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

        // Randomly drop some packets (simulate error/loss)
        if (rand() % 4 == 0) {
            printf("Simulating loss of packet SEQ:%d\n", seq);
            continue;  // No ACK sent
        }

        // Send ACK
        char ack[32];
        sprintf(ack, "ACK:%d", seq);
        send(client_sock, ack, strlen(ack), 0);
        printf("Sent %s\n", ack);
    }

    close(client_sock);
    close(server_sock);
    return 0;
}

// client.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/time.h>
#include <stdbool.h>
#define PORT 9000
#define MAX 1024
#define WINDOW_SIZE 4
#define TOTAL_PACKETS 10
#define TIMEOUT_SEC 2

typedef struct {
    int seq;
    bool acked;
    char data[MAX];
} Packet;

int main() {
    int sock;
    struct sockaddr_in server_addr;
    char buffer[MAX];
    Packet packets[TOTAL_PACKETS];

    sock = socket(AF_INET, SOCK_STREAM, 0);
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(PORT);
    server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
```

```
    connect(sock, (struct sockaddr*)&server_addr, sizeof(server_addr));
    printf("Connected to server.\n");

    // Prepare packets
    for (int i = 0; i < TOTAL_PACKETS; i++) {
        packets[i].seq = i;
        packets[i].acked = false;
        sprintf(packets[i].data, "SEQ:%d", i);
    }

    int base = 0;
    fd_set readfds;
    struct timeval timeout;

    while (base < TOTAL_PACKETS) {
        // Send all packets in the window that haven't been ACKed
        for (int i = base; i < base + WINDOW_SIZE && i < TOTAL_PACKETS; i++) {
            if (!packets[i].acked) {
                send(sock, packets[i].data, strlen(packets[i].data), 0);
                printf("Sent packet: %s\n", packets[i].data);
            }
        }

        // Wait for ACKs with timeout
        FD_ZERO(&readfds);
        FD_SET(sock, &readfds);
        timeout.tv_sec = TIMEOUT_SEC;
        timeout.tv_usec = 0;

        int activity = select(sock + 1, &readfds, NULL, NULL, &timeout);
        if (activity > 0 && FD_ISSET(sock, &readfds)) {
            memset(buffer, 0, sizeof(buffer));
            recv(sock, buffer, sizeof(buffer), 0);

            int ack_seq;
            if (sscanf(buffer, "ACK:%d", &ack_seq) == 1) {
                packets[ack_seq].acked = true;
                printf("Received ACK: %d\n", ack_seq);

                // Slide window
                while (base < TOTAL_PACKETS && packets[base].acked)
                    base++;
            }
        } else {
            printf("Timeout: Resending unacknowledged packets in window...\n");
        }
    }

    send(sock, "END", 3, 0);
    close(sock);
    return 0;
}
```

**Output:**

```
ubuntu@ubuntu:~$ gcc server.c -o s
ubuntu@ubuntu:~$ ./s
Server listening on port 9000...
Client connected.
Received packet: SEQ:0SEQ:1SEQ:2SEQ:3
Sent ACK:0
Received packet: SEQ:1
Simulating loss of packet SEQ:1
Received packet: SEQ:2SEQ:3SEQ:4
Simulating loss of packet SEQ:2
Received packet: SEQ:1
Sent ACK:1
Received packet: SEQ:2SEQ:3SEQ:4
Sent ACK:2
Received packet: SEQ:2SEQ:3SEQ:4SEQ:5
Simulating loss of packet SEQ:2
Received packet: SEQ:3
Sent ACK:3
Received packet: SEQ:4
Sent ACK:4
Received packet: SEQ:5SEQ:6SEQ:4SEQ:5SEQ:6SEQ:7
Sent ACK:5
Received packet: SEQ:5
Simulating loss of packet SEQ:5
Received packet: SEQ:6SEQ:7SEQ:8SEQ:6SEQ:7SEQ:8SEQ:9
Sent ACK:6
Received packet: SEQ:7
Simulating loss of packet SEQ:7
Received packet: SEQ:8SEQ:9
Sent ACK:8
Received packet: SEQ:7
Sent ACK:7
Received packet: SEQ:9
Sent ACK:9
Received packet: SEQ:9
Sent ACK:9
ubuntu@ubuntu:~$
```

```
ubuntu@ubuntu:~$ gcc client.c -o c
ubuntu@ubuntu:~$ ./c
Connected to server.
Sent packet: SEQ:0
Sent packet: SEQ:1
Sent packet: SEQ:2
Sent packet: SEQ:3
Received ACK: 0
Sent packet: SEQ:1
Sent packet: SEQ:2
Sent packet: SEQ:3
Sent packet: SEQ:4
Timeout: Resending unacknowledged packets in window...
Sent packet: SEQ:1
Sent packet: SEQ:2
Sent packet: SEQ:3
Sent packet: SEQ:4
Received ACK: 1
Sent packet: SEQ:2
Sent packet: SEQ:3
Sent packet: SEQ:4
Sent packet: SEQ:5
Received ACK: 2
Sent packet: SEQ:3
Sent packet: SEQ:4
Sent packet: SEQ:5
Sent packet: SEQ:6
Received ACK: 3
Sent packet: SEQ:4
Sent packet: SEQ:5
Sent packet: SEQ:6
Sent packet: SEQ:7
Received ACK: 4
Sent packet: SEQ:5
Sent packet: SEQ:6
Sent packet: SEQ:7
Sent packet: SEQ:8
Received ACK: 5
Sent packet: SEQ:6
Sent packet: SEQ:7
Sent packet: SEQ:8
Sent packet: SEQ:9
Received ACK: 6
Sent packet: SEQ:7
Sent packet: SEQ:8
```