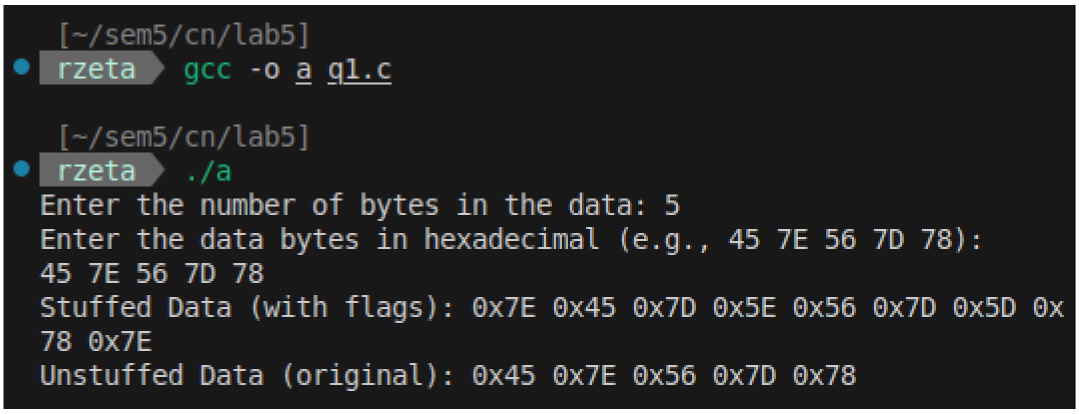
COMPUTER NETWORKS
LAB – 5

ROHAN G
CS22B1093

Q1)
```
//CS22B1093 ROHAN G
#include <stdio.h>
#include <string.h>

#define FLAG 0x7E   // Define the flag byte
#define ESCAPE 0x7D // Define the escape byte

void byteStuffing(char *input, int length, char *stuffed, int *stuffedLength) {
   int i, j = 0;

   stuffed[j++] = FLAG;

   // Stuff the data
   for (i = 0; i < length; i++) {
      if (input[i] == FLAG) {
         stuffed[j++] = ESCAPE;
         stuffed[j++] = FLAG ^ 0x20; // XOR with 0x20 = 0x5E
      } else if (input[i] == ESCAPE) {
         stuffed[j++] = ESCAPE;
         stuffed[j++] = ESCAPE ^ 0x20; // XOR with 0x20 = 0x5D
      } else {
         stuffed[j++] = input[i];
      }
   }

   // Add the ending flag byte
   stuffed[j++] = FLAG;

   *stuffedLength = j;
}

// Function to perform byte unstuffing
void byteUnstuffing(char *stuffed, int stuffedLength,char *unstuffed, int *unstuffedLength) {
   int i = 1, j = 0;

   while (i < stuffedLength - 1) {
      if (stuffed[i] == ESCAPE) {
         i++;
         if (stuffed[i] == (FLAG ^ 0x20)) { // XOR with 0x20 = 0x5E
            unstuffed[j++] = FLAG;
         } else if (stuffed[i] == (ESCAPE ^ 0x20)) { // XOR with 0x20 = 0x5D
            unstuffed[j++] = ESCAPE;
         }
      } else {
         unstuffed[j++] = stuffed[i];
      }
```

```c
        i++;
    }
    *unstuffedLength = j;
}

int main() {
    char input[100];
    char stuffed[200];
    char unstuffed[100];
    int unstuffedLength;

    // Get the user input
    int inputLength;
    printf("Enter the number of bytes in the data: ");
    scanf("%d", &inputLength);

    printf("Enter the data bytes in hexadecimal (e.g., 45 7E 56 7D 78):\n");
    for (int i = 0; i < inputLength; i++) {
        int byte;
        scanf("%x", &byte);
        input[i] = (char)byte;
    }

    int stuffedLength = 0;

    byteStuffing(input, inputLength, stuffed, &stuffedLength);

    printf("Stuffed Data (with flags): ");
    for (int i = 0; i < stuffedLength; i++) {
        printf("0x%X ", stuffed[i]);
    }
    printf("\n");

    byteUnstuffing(stuffed, stuffedLength, unstuffed, &unstuffedLength);

    printf("Unstuffed Data (original): ");
    for (int i = 0; i < unstuffedLength; i++) {
        printf("0x%X ", unstuffed[i]);
    }
    printf("\n");

    return 0;
}
```

Output -



```
  [~/sem5/cn/lab5]
● rzeta  gcc -o a q1.c

  [~/sem5/cn/lab5]
● rzeta  ./a
Enter the number of bytes in the data: 5
Enter the data bytes in hexadecimal (e.g., 45 7E 56 7D 78):
45 7E 56 7D 78
Stuffed Data (with flags): 0x7E 0x45 0x7D 0x5E 0x56 0x7D 0x5D 0x
78 0x7E
Unstuffed Data (original): 0x45 0x7E 0x56 0x7D 0x78
```

Q2)

Server -

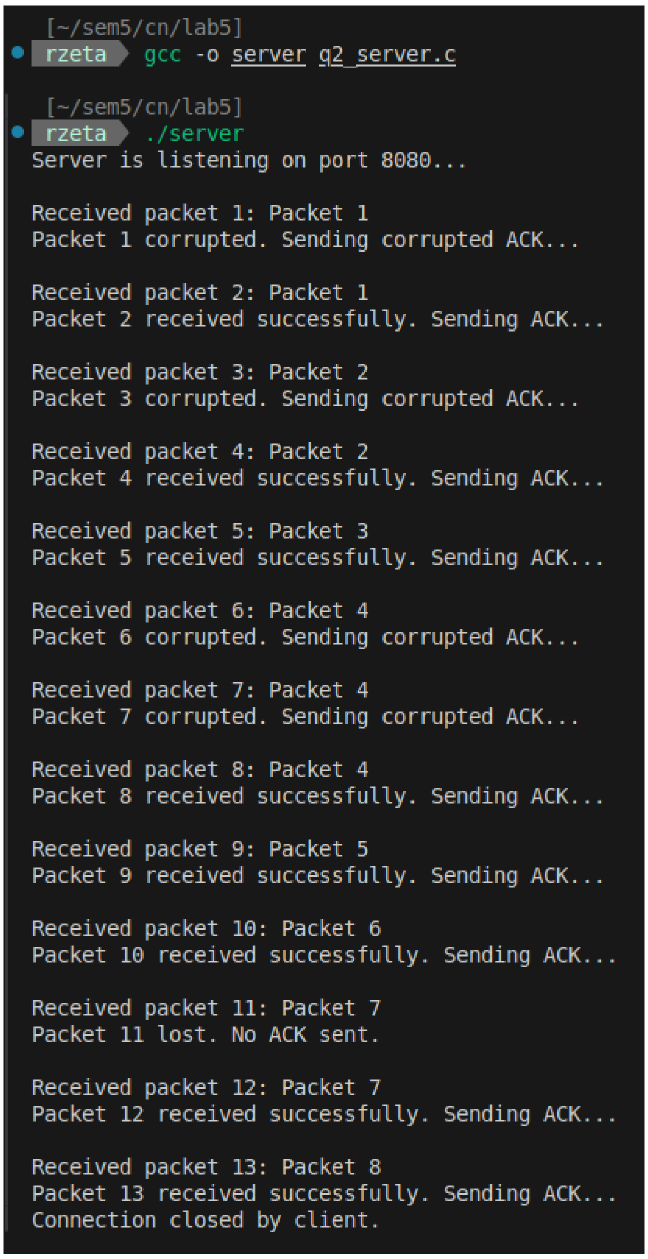
```
// server.c
// CS22B1093 ROHAN G
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <time.h>

#define PORT 8080
#define MAX_BUFFER 1024
#define PACKET_LOSS_PROBABILITY 20 // 20% chance of packet loss or corruption

// Function to simulate packet loss or corruption
int simulatePacketLossOrCorruption() {
    // Generate a random number to simulate packet loss or corruption
    int randomValue = rand() % 100;
    return (randomValue < PACKET_LOSS_PROBABILITY);
}

int main() {
    int server_fd, new_socket;
    struct sockaddr_in address;
    int addrlen = sizeof(address);
    char buffer[MAX_BUFFER] = {0};
    char ack[] = "ACK";  // Normal acknowledgment
    char corruptedAck[] = "CORRUPTED_ACK";  // Simulated corrupted ACK

    srand(time(0));  // Seed for random number generation

    // Create socket file descriptor
    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");
    exit(EXIT_FAILURE);
}

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

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

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

int packetNumber = 1;
while (1) {
    memset(buffer, 0, MAX_BUFFER);
    // Read packet from client
    int valread = read(new_socket, buffer, MAX_BUFFER);

    if (valread == 0) {
        printf("Connection closed by client.\n");
        break;
    }

    printf("\nReceived packet %d: %s\n", packetNumber, buffer);

    // Simulate packet loss or corruption
    if (simulatePacketLossOrCorruption()) {
        int corruptPacket = rand() % 2; // Decide between loss or corruption
        if (corruptPacket == 0) {
            printf("Packet %d lost. No ACK sent.\n", packetNumber);
        } else {
            printf("Packet %d corrupted. Sending corrupted ACK...\n", packetNumber);
            send(new_socket, corruptedAck, strlen(corruptedAck), 0);  // Send corrupted ACK
        }
    } else {
        printf("Packet %d received successfully. Sending ACK...\n", packetNumber);
        send(new_socket, ack, strlen(ack), 0); // Send valid ACK
    }

    packetNumber++;
    }

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

<u>Client - </u>

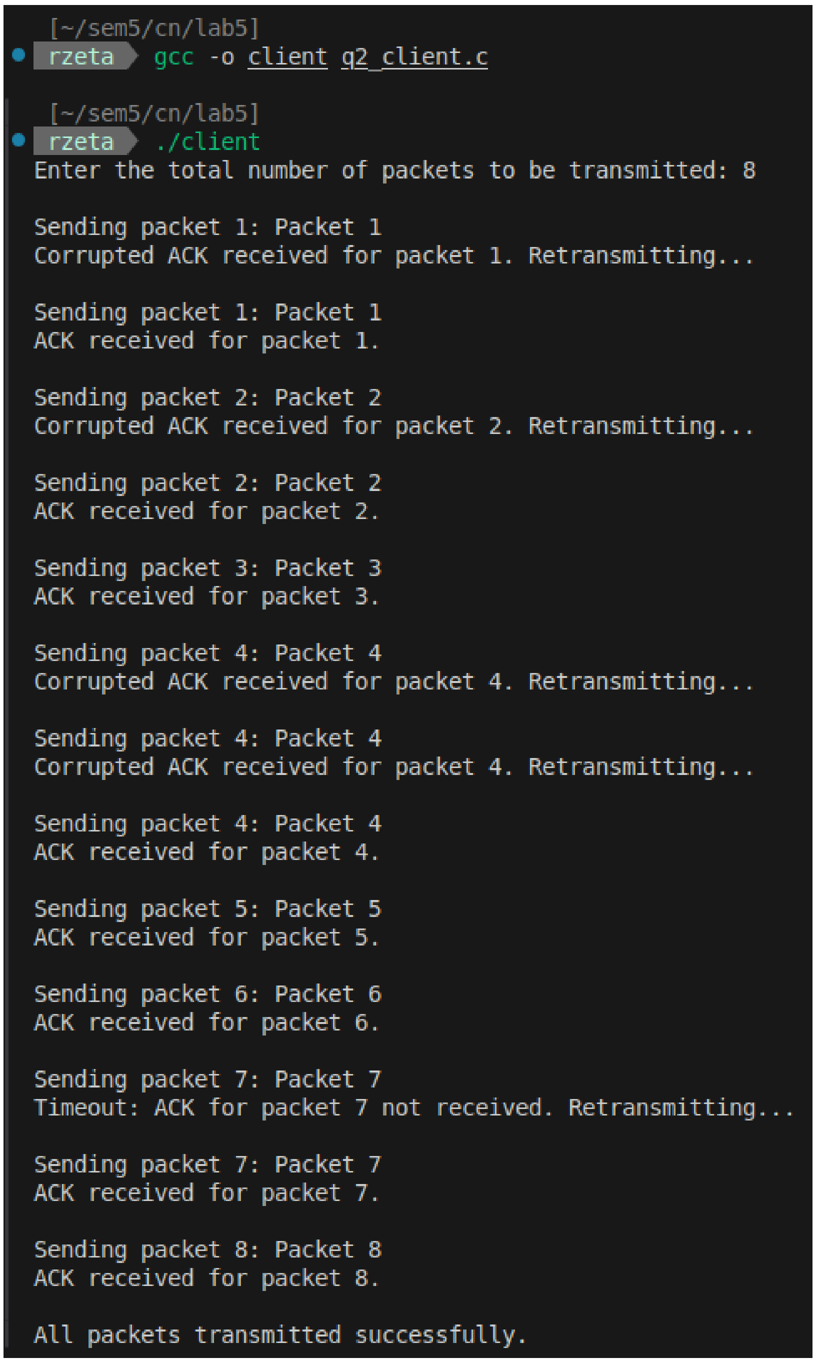
```
// client.c
// CS22B1093 ROHAN G
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/select.h>
#include <time.h>

#define PORT 8080
#define MAX_BUFFER 1024
#define TIMEOUT 3  // Timeout duration in seconds

int main() {
    int sock = 0, valread;
    struct sockaddr_in serv_addr;
    char buffer[MAX_BUFFER] = {0};
    char ack[MAX_BUFFER] = {0};
    int totalPackets, packetNumber = 1;

    srand(time(0));  // Seed for random number generation

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

    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 the server
    if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
        printf("\nConnection failed\n");
        return -1;
    }
    // Get the total number of packets to be sent
    printf("Enter the total number of packets to be transmitted: ");
    scanf("%d", &totalPackets);

    while (packetNumber <= totalPackets) {
        // Create a packet message
        snprintf(buffer, MAX_BUFFER, "Packet %d", packetNumber);
        printf("\nSending packet %d: %s\n", packetNumber, buffer);

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

        fd_set readfds;
        struct timeval tv;
        int retval;

        // Initialize the timeout structure
        tv.tv_sec = TIMEOUT;
        tv.tv_usec = 0;

        // Set up the file descriptor set
        FD_ZERO(&readfds);
        FD_SET(sock, &readfds);

        // Wait for the ACK from the server or timeout
        retval = select(sock + 1, &readfds, NULL, NULL, &tv);

        if (retval == -1) {
            perror("select() error");
            close(sock);
            return -1;
        } else if (retval == 0) {
            printf("Timeout: ACK for packet %d not received. Retransmitting...\n", packetNumber);
            continue; // Retransmit the same packet
        } else {
            // ACK received
            memset(ack, 0, MAX_BUFFER);
            valread = read(sock, ack, MAX_BUFFER);
            if (valread > 0) {
                if (strcmp(ack, "ACK") == 0) {
                    printf("ACK received for packet %d.\n", packetNumber);
                    packetNumber++; // Move to the next packet
                } else if (strcmp(ack, "CORRUPTED_ACK") == 0) {
                    printf("Corrupted ACK received for packet %d. Retransmitting...\n", packetNumber);
                } else {
                    printf("No ACK or incorrect ACK received. Retransmitting packet %d...\n",
packetNumber);
                }
            } else {
                printf("Error receiving ACK. Retransmitting packet %d...\n", packetNumber);
            }
        }
    }

    printf("\nAll packets transmitted successfully.\n");
    close(sock);
    return 0;
}
```

Terminal – 1(Server)



```
  [~/sem5/cn/lab5]
● rzeta  gcc -o server q2_server.c

  [~/sem5/cn/lab5]
● rzeta  ./server
Server is listening on port 8080...

Received packet 1: Packet 1
Packet 1 corrupted. Sending corrupted ACK...

Received packet 2: Packet 1
Packet 2 received successfully. Sending ACK...

Received packet 3: Packet 2
Packet 3 corrupted. Sending corrupted ACK...

Received packet 4: Packet 2
Packet 4 received successfully. Sending ACK...

Received packet 5: Packet 3
Packet 5 received successfully. Sending ACK...

Received packet 6: Packet 4
Packet 6 corrupted. Sending corrupted ACK...

Received packet 7: Packet 4
Packet 7 corrupted. Sending corrupted ACK...

Received packet 8: Packet 4
Packet 8 received successfully. Sending ACK...

Received packet 9: Packet 5
Packet 9 received successfully. Sending ACK...

Received packet 10: Packet 6
Packet 10 received successfully. Sending ACK...

Received packet 11: Packet 7
Packet 11 lost. No ACK sent.

Received packet 12: Packet 7
Packet 12 received successfully. Sending ACK...

Received packet 13: Packet 8
Packet 13 received successfully. Sending ACK...
Connection closed by client.
```

Terminal – 2(client)



```
 [~/sem5/cn/lab5]
● rzeta  gcc -o client q2_client.c

 [~/sem5/cn/lab5]
● rzeta  ./client
Enter the total number of packets to be transmitted: 8

Sending packet 1: Packet 1
Corrupted ACK received for packet 1. Retransmitting...

Sending packet 1: Packet 1
ACK received for packet 1.

Sending packet 2: Packet 2
Corrupted ACK received for packet 2. Retransmitting...

Sending packet 2: Packet 2
ACK received for packet 2.

Sending packet 3: Packet 3
ACK received for packet 3.

Sending packet 4: Packet 4
Corrupted ACK received for packet 4. Retransmitting...

Sending packet 4: Packet 4
Corrupted ACK received for packet 4. Retransmitting...

Sending packet 4: Packet 4
ACK received for packet 4.

Sending packet 5: Packet 5
ACK received for packet 5.

Sending packet 6: Packet 6
ACK received for packet 6.

Sending packet 7: Packet 7
Timeout: ACK for packet 7 not received. Retransmitting...

Sending packet 7: Packet 7
ACK received for packet 7.

Sending packet 8: Packet 8
ACK received for packet 8.

All packets transmitted successfully.
```

Q3)

Server -

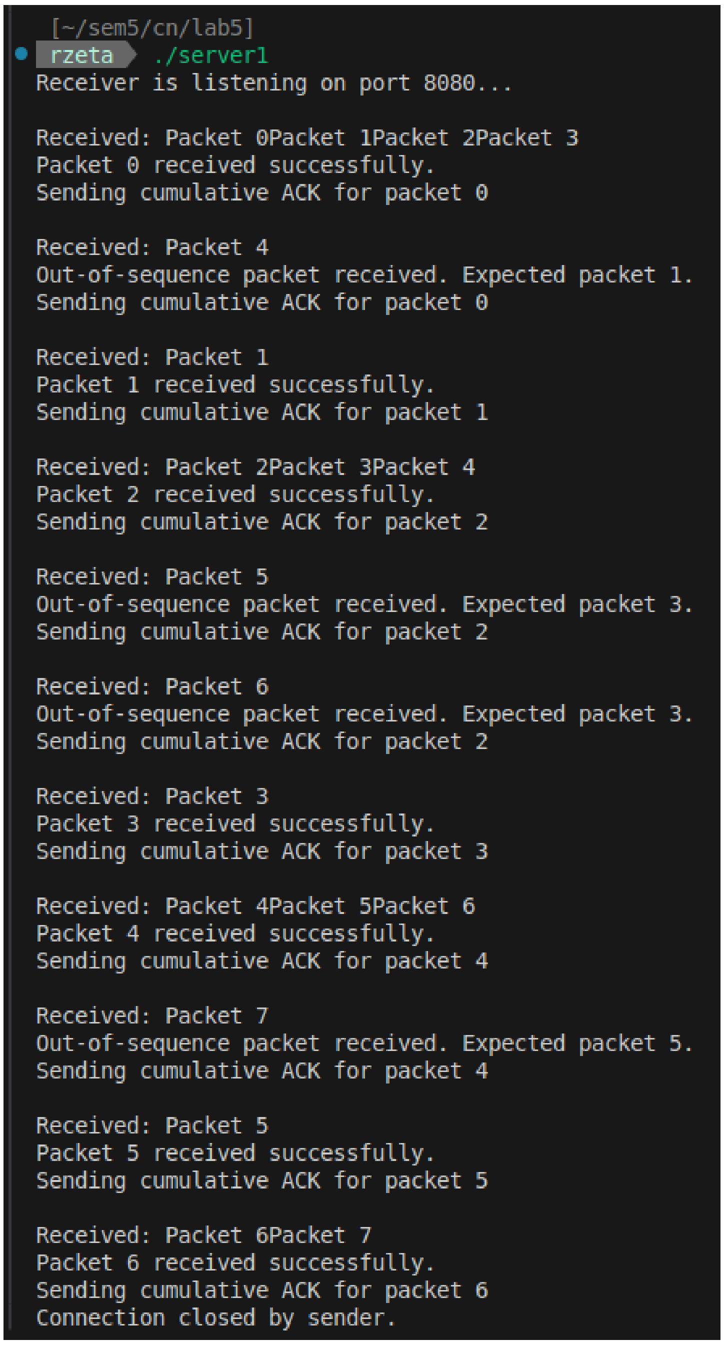
```
// Receiver program for Go-Back-N protocol using TCP
//CS22B1093 ROHAN G
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <time.h>

#define PORT 8080
#define MAX_BUFFER 1024
#define PACKET_LOSS_PROBABILITY 20 // 20% chance of packet loss or corruption

int simulatePacketLossOrCorruption() {
    return rand() % 100 < PACKET_LOSS_PROBABILITY;
}

int main() {
    int server_fd, new_socket;
    struct sockaddr_in address;
    int addrlen = sizeof(address);
    char buffer[MAX_BUFFER] = {0};
    int expectedSeqNum = 0;  // The next packet the receiver expects

    srand(time(0));  // Seed for random number generation

    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        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("Receiver is listening on port %d...\n", PORT);

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

    while (1) {
        memset(buffer, 0, MAX_BUFFER);
        int valread = read(new_socket, buffer, MAX_BUFFER);
        if (valread == 0) {
            printf("Connection closed by sender.\n");
            break;
        }

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

        if (simulatePacketLossOrCorruption()) {
            printf("Packet %d lost or corrupted. No ACK sent.\n", expectedSeqNum);
            continue;
        }

        int receivedSeqNum = atoi(buffer + 7);  // Extract packet number

        if (receivedSeqNum == expectedSeqNum) {
            printf("Packet %d received successfully.\n", receivedSeqNum);
            expectedSeqNum++;
        } else {
            printf("Out-of-sequence packet received. Expected packet %d.\n", expectedSeqNum);
        }

        char ack[MAX_BUFFER];
        snprintf(ack, MAX_BUFFER, "%d", expectedSeqNum - 1);
        printf("Sending cumulative ACK for packet %d\n", expectedSeqNum - 1);
        send(new_socket, ack, strlen(ack), 0);
    }

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

<u>Client - </u>

```
// Simulate Go-Back-N ARQ protocol using TCP
//CS22B1093 ROHAN G
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/select.h>
#include <time.h>

#define PORT 8080
#define MAX_BUFFER 1024
#define TIMEOUT 3  // Timeout duration in seconds
```

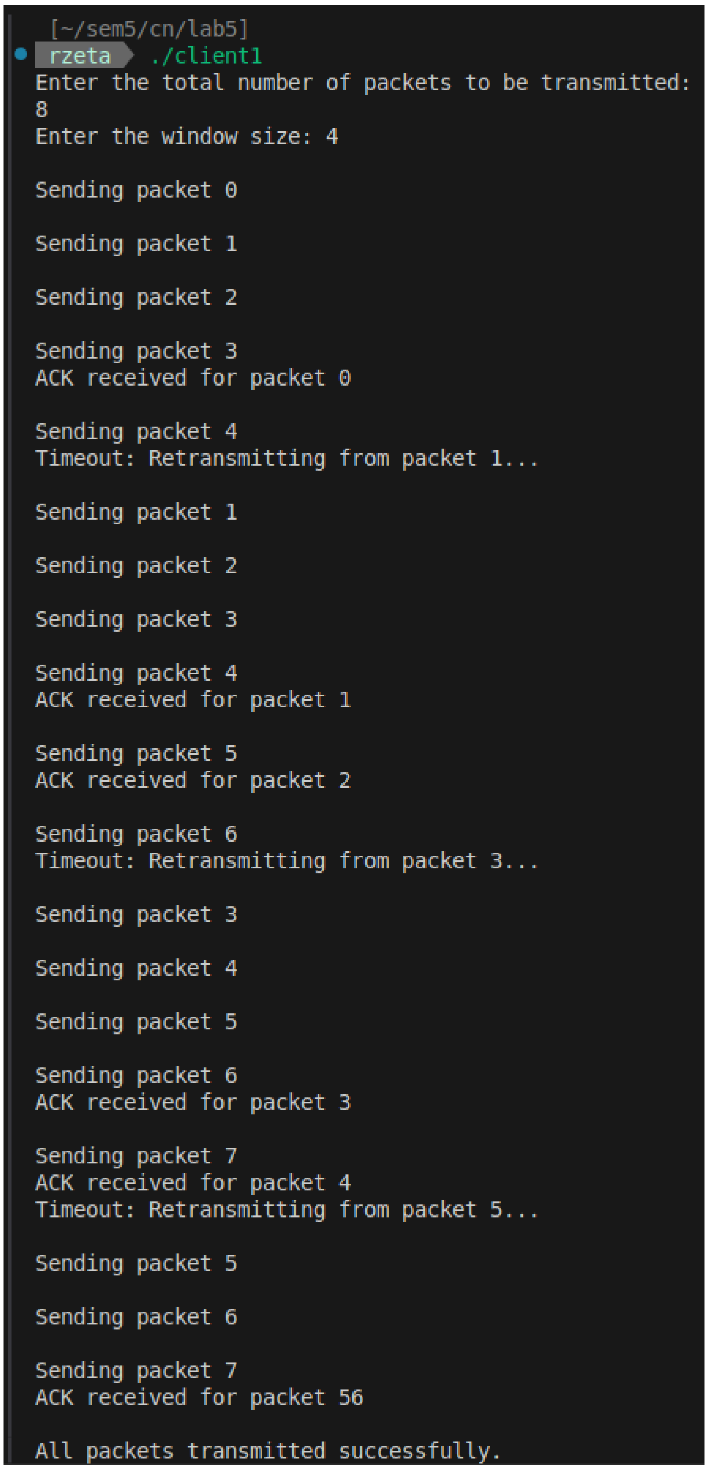
```
#define PACKET_LOSS_PROBABILITY 20 // 20% chance of packet loss or corruption

int simulatePacketLossOrCorruption() {
    return rand() % 100 < PACKET_LOSS_PROBABILITY;
}

int main() {
    int sock = 0, valread;
    struct sockaddr_in serv_addr;
    char buffer[MAX_BUFFER] = {0};
    char ack[MAX_BUFFER] = {0};
    int totalPackets, windowSize;
    int base = 0, nextSeqNum = 0;  // Go-Back-N window variables
    int ackNum = 0;

    srand(time(0));  // Seed for random number generation

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

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

    if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr) <= 0) {
        printf("\nInvalid address/ Address not supported\n");
        return -1;
    }

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

    printf("Enter the total number of packets to be transmitted: ");
    scanf("%d", &totalPackets);
    printf("Enter the window size: ");
    scanf("%d", &windowSize);

    while (base < totalPackets) {
        // Send packets within window
        while (nextSeqNum < base + windowSize && nextSeqNum < totalPackets) {
            snprintf(buffer, MAX_BUFFER, "Packet %d", nextSeqNum);
            printf("\nSending packet %d\n", nextSeqNum);
            send(sock, buffer, strlen(buffer), 0);
            nextSeqNum++;
        }

        // Wait for ACK or timeout
        fd_set readfds;
        struct timeval tv;
        int retval;

        tv.tv_sec = TIMEOUT;
        tv.tv_usec = 0;
        FD_ZERO(&readfds);
        FD_SET(sock, &readfds);

        retval = select(sock + 1, &readfds, NULL, NULL, &tv);

        if (retval == -1) {
            perror("select() error");
            close(sock);
            return -1;
        } else if (retval == 0) {
            // Timeout occurred
            printf("Timeout: Retransmitting from packet %d...\n", base);
            nextSeqNum = base;  // Go back to base for retransmission
        } else {
            // Read ACK from receiver
            memset(ack, 0, MAX_BUFFER);
            valread = read(sock, ack, MAX_BUFFER);
            if (valread > 0) {
                ackNum = atoi(ack);
                if (ackNum >= base) {
                    printf("ACK received for packet %d\n", ackNum);
                    base = ackNum + 1;  // Slide window forward
                }
            }
        }
    }

    printf("\nAll packets transmitted successfully.\n");
    close(sock);
    return 0;
}
```

Terminal – 1 (server)



```
  [~/sem5/cn/lab5]
● rzeta  ./server1
Receiver is listening on port 8080...

Received: Packet 0Packet 1Packet 2Packet 3
Packet 0 received successfully.
Sending cumulative ACK for packet 0

Received: Packet 4
Out-of-sequence packet received. Expected packet 1.
Sending cumulative ACK for packet 0

Received: Packet 1
Packet 1 received successfully.
Sending cumulative ACK for packet 1

Received: Packet 2Packet 3Packet 4
Packet 2 received successfully.
Sending cumulative ACK for packet 2

Received: Packet 5
Out-of-sequence packet received. Expected packet 3.
Sending cumulative ACK for packet 2

Received: Packet 6
Out-of-sequence packet received. Expected packet 3.
Sending cumulative ACK for packet 2

Received: Packet 3
Packet 3 received successfully.
Sending cumulative ACK for packet 3

Received: Packet 4Packet 5Packet 6
Packet 4 received successfully.
Sending cumulative ACK for packet 4

Received: Packet 7
Out-of-sequence packet received. Expected packet 5.
Sending cumulative ACK for packet 4

Received: Packet 5
Packet 5 received successfully.
Sending cumulative ACK for packet 5

Received: Packet 6Packet 7
Packet 6 received successfully.
Sending cumulative ACK for packet 6
Connection closed by sender.
```

Terminal – 2 (client)



```
 [~/sem5/cn/lab5]
● rzeta  ./client1
Enter the total number of packets to be transmitted:
8
Enter the window size: 4

Sending packet 0

Sending packet 1

Sending packet 2

Sending packet 3
ACK received for packet 0

Sending packet 4
Timeout: Retransmitting from packet 1...

Sending packet 1

Sending packet 2

Sending packet 3

Sending packet 4
ACK received for packet 1

Sending packet 5
ACK received for packet 2

Sending packet 6
Timeout: Retransmitting from packet 3...

Sending packet 3

Sending packet 4

Sending packet 5

Sending packet 6
ACK received for packet 3

Sending packet 7
ACK received for packet 4
Timeout: Retransmitting from packet 5...

Sending packet 5

Sending packet 6

Sending packet 7
ACK received for packet 56

All packets transmitted successfully.
```

Q4)

Server -

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

#define PORT 8080
#define MAX_BUFFER 1024
#define PACKET_LOSS_PROBABILITY 20  // 20% chance of packet loss or corruption

int simulatePacketLossOrCorruption() {
    return rand() % 100 < PACKET_LOSS_PROBABILITY;
}

int main() {
    int server_fd, new_socket;
    struct sockaddr_in address;
    int addrlen = sizeof(address);
    char buffer[MAX_BUFFER] = {0};
    int expectedSeqNum = 0;  // Expected packet number for Selective Repeat ARQ

    srand(time(0));  // Seed for random number generation

    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        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("Receiver is listening on port %d...\n", PORT);

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

    while (1) {
        memset(buffer, 0, MAX_BUFFER);
        int valread = read(new_socket, buffer, MAX_BUFFER);
        if (valread == 0) {
            printf("Connection closed by sender.\n");
            break;
        }

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

        if (simulatePacketLossOrCorruption()) {
            printf("Packet %d lost or corrupted. No ACK sent.\n", expectedSeqNum);
            continue;
        }

        int receivedSeqNum = atoi(buffer + 7);  // Extract packet number

        if (receivedSeqNum >= expectedSeqNum) {
            printf("Packet %d received successfully.\n", receivedSeqNum);
            expectedSeqNum = receivedSeqNum + 1;
        } else {
            printf("Out-of-sequence packet received. Expected packet %d.\n", expectedSeqNum);
        }

        char ack[MAX_BUFFER];
        snprintf(ack, MAX_BUFFER, "%d", receivedSeqNum);
        printf("Sending ACK for packet %d\n", receivedSeqNum);
        send(new_socket, ack, strlen(ack), 0);
    }

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

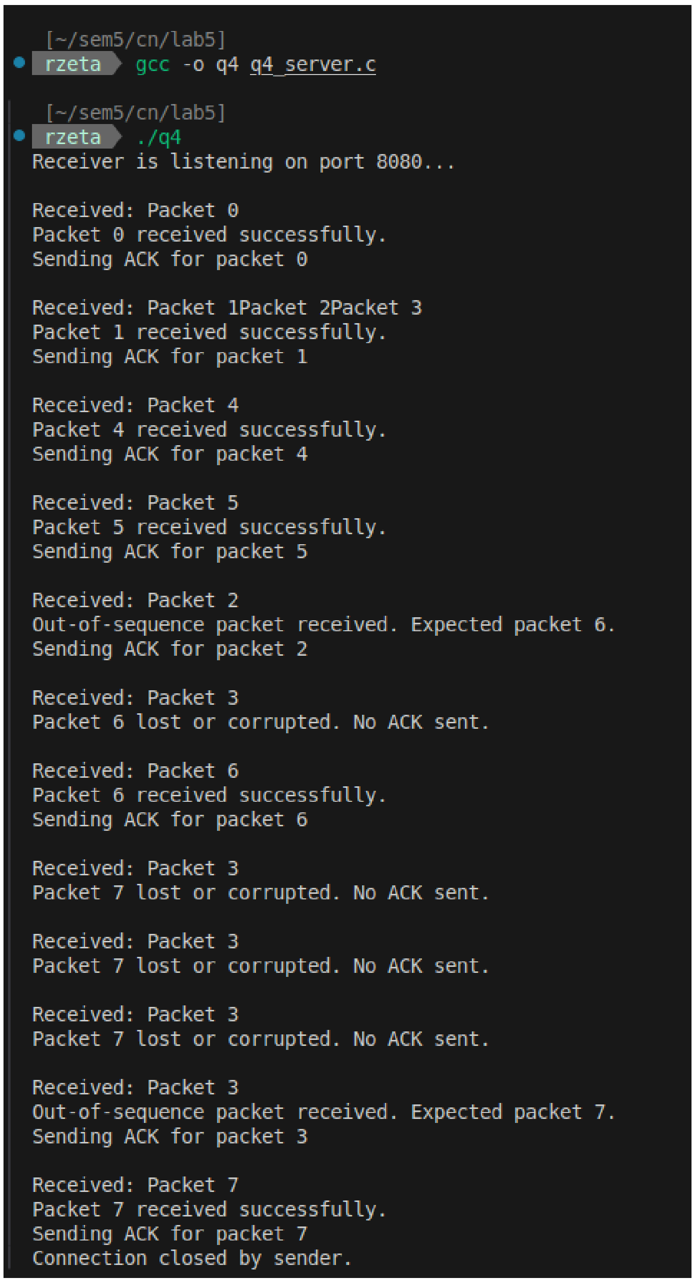
Client -

```
// sender_selective_repeat.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <time.h>
#include <sys/select.h>

#define PORT 8080
#define MAX_BUFFER 1024
#define TIMEOUT 3
#define PACKET_LOSS_PROBABILITY 20  // 20% chance of packet loss or corruption
```

```
int simulatePacketLossOrCorruption() {
   return rand() % 100 < PACKET_LOSS_PROBABILITY;
}

int main() {
   int sock = 0, valread;
   struct sockaddr_in serv_addr;
   char buffer[MAX_BUFFER] = {0};
   char ack[MAX_BUFFER] = {0};
   int totalPackets, windowSize;
   int base = 0, nextSeqNum = 0;  // Selective Repeat ARQ window variables
   int ackNum = 0;
   int *ackReceived;  // Track if packets have been acknowledged

   srand(time(0));  // Seed for random number generation

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

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

   if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr) <= 0) {
      printf("\nInvalid address/ Address not supported\n");
      return -1;
   }

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

   printf("Enter the total number of packets to be transmitted: ");
   scanf("%d", &totalPackets);
   printf("Enter the window size: ");
   scanf("%d", &windowSize);

   ackReceived = (int *)malloc(totalPackets * sizeof(int));
   memset(ackReceived, 0, totalPackets * sizeof(int));

   while (base < totalPackets) {
      // Send packets within window
      while (nextSeqNum < base + windowSize && nextSeqNum < totalPackets) {
         if (!ackReceived[nextSeqNum]) {
            snprintf(buffer, MAX_BUFFER, "Packet %d", nextSeqNum);
            printf("\nSending packet %d\n", nextSeqNum);
            send(sock, buffer, strlen(buffer), 0);
         }
         nextSeqNum++;
```

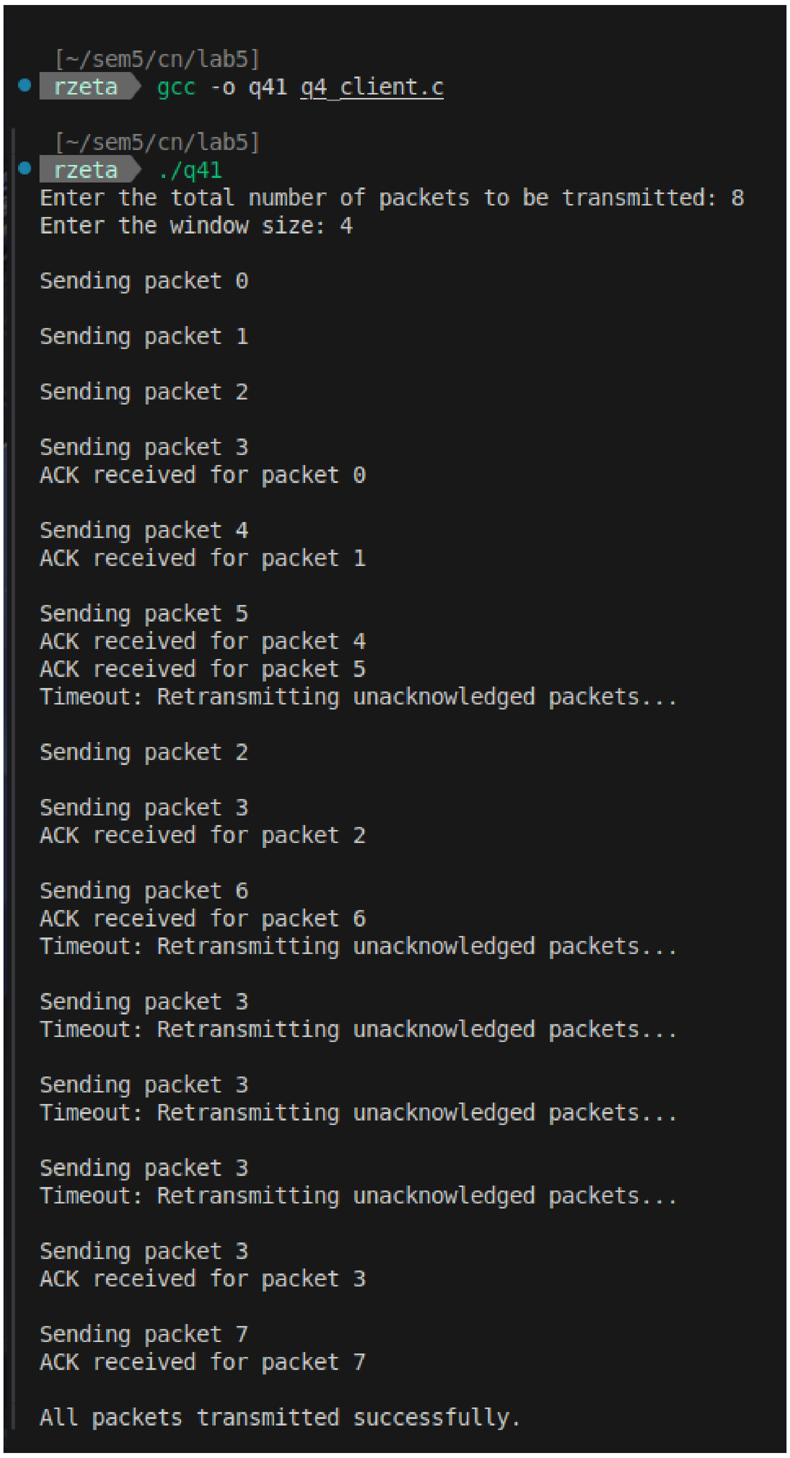
```
        }

        // Wait for ACK or timeout
        fd_set readfds;
        struct timeval tv;
        int retval;

        tv.tv_sec = TIMEOUT;
        tv.tv_usec = 0;
        FD_ZERO(&readfds);
        FD_SET(sock, &readfds);

        retval = select(sock + 1, &readfds, NULL, NULL, &tv);

        if (retval == -1) {
            perror("select() error");
            close(sock);
            return -1;
        } else if (retval == 0) {
            // Timeout occurred
            printf("Timeout: Retransmitting unacknowledged packets...\n");
            nextSeqNum = base;
        } else {
            // Read ACK from receiver
            memset(ack, 0, MAX_BUFFER);
            valread = read(sock, ack, MAX_BUFFER);
            if (valread > 0) {
                ackNum = atoi(ack);
                if (ackNum >= base && ackNum < totalPackets) {
                    printf("ACK received for packet %d\n", ackNum);
                    ackReceived[ackNum] = 1;

                    // Slide window if the base packet is acknowledged
                    while (ackReceived[base] && base < totalPackets) {
                        base++;
                    }
                }
            }
        }
    }

    printf("\nAll packets transmitted successfully.\n");
    free(ackReceived);
    close(sock);
    return 0;
}
```

Terminal – 1 (server):



```
 [~/sem5/cn/lab5]
● rzeta  gcc -o q4 q4_server.c

 [~/sem5/cn/lab5]
● rzeta  ./q4
Receiver is listening on port 8080...

Received: Packet 0
Packet 0 received successfully.
Sending ACK for packet 0

Received: Packet 1Packet 2Packet 3
Packet 1 received successfully.
Sending ACK for packet 1

Received: Packet 4
Packet 4 received successfully.
Sending ACK for packet 4

Received: Packet 5
Packet 5 received successfully.
Sending ACK for packet 5

Received: Packet 2
Out-of-sequence packet received. Expected packet 6.
Sending ACK for packet 2

Received: Packet 3
Packet 6 lost or corrupted. No ACK sent.

Received: Packet 6
Packet 6 received successfully.
Sending ACK for packet 6

Received: Packet 3
Packet 7 lost or corrupted. No ACK sent.

Received: Packet 3
Packet 7 lost or corrupted. No ACK sent.

Received: Packet 3
Packet 7 lost or corrupted. No ACK sent.

Received: Packet 3
Out-of-sequence packet received. Expected packet 7.
Sending ACK for packet 3

Received: Packet 7
Packet 7 received successfully.
Sending ACK for packet 7
Connection closed by sender.
```

Terminal – 2 (client) :



```
 [~/sem5/cn/lab5]
● rzeta  gcc -o q41 q4_client.c

 [~/sem5/cn/lab5]
● rzeta  ./q41
Enter the total number of packets to be transmitted: 8
Enter the window size: 4

Sending packet 0

Sending packet 1

Sending packet 2

Sending packet 3
ACK received for packet 0

Sending packet 4
ACK received for packet 1

Sending packet 5
ACK received for packet 4
ACK received for packet 5
Timeout: Retransmitting unacknowledged packets...

Sending packet 2

Sending packet 3
ACK received for packet 2

Sending packet 6
ACK received for packet 6
Timeout: Retransmitting unacknowledged packets...

Sending packet 3
Timeout: Retransmitting unacknowledged packets...

Sending packet 3
Timeout: Retransmitting unacknowledged packets...

Sending packet 3
Timeout: Retransmitting unacknowledged packets...

Sending packet 3
ACK received for packet 3

Sending packet 7
ACK received for packet 7

All packets transmitted successfully.
```