COMPUTER NETWORKS
LAB ASSIGNMENT-03

REDDIPALLI SAI CHARISH

CS22B1095

QUESTION 1:

```
//CS22B1095 REDDIPALLI SAI CHARISH
#include <stdio.h>
#include <string.h>
#define FLAG 0x7E  // Flag byte
#define ESCAPE 0x7D // Escape byte
// Function to perform byte stuffing
void byte_stuffing(unsigned char data[], int len, unsigned char stuffed_data[], int *stuffed_len) {
   int j = 0;
 // Add FLAG byte at the beginning of the stuffed data
   stuffed_data[j++] = FLAG;

   for (int i = 0; i < len; i++) {
      if (data[i] == FLAG) {
         stuffed_data[j++] = ESCAPE;
         stuffed_data[j++] = 0x5E; // Replace 0x7E with 0x7D 0x5E
      } else if (data[i] == ESCAPE) {
         stuffed_data[j++] = ESCAPE;
         stuffed_data[j++] = 0x5D; // Replace 0x7D with 0x7D 0x5D
      } else {
         stuffed_data[j++] = data[i];
      }
   }

   // Add FLAG byte at the end of the stuffed data
   stuffed_data[j++] = FLAG;

   *stuffed_len = j;
}

// Function to perform byte de-stuffing
void byte_destuffing(unsigned char stuffed_data[], int stuffed_len, unsigned char destuffed_data[], int *destuffed_len) {
   int j = 0;
```

```
    // Ignore the FLAG byte at the beginning
    for (int i = 1; i < stuffed_len - 1; i++) { // Ignore first and last FLAG byte
        if (stuffed_data[i] == ESCAPE) {
            if (stuffed_data[i + 1] == 0x5E) {
                destuffed_data[j++] = FLAG; // Convert 0x7D 0x5E back to 0x7E
            } else if (stuffed_data[i + 1] == 0x5D) {
                destuffed_data[j++] = ESCAPE; // Convert 0x7D 0x5D back to 0x7D
            }
            i++; // Skip the next byte as it is part of the escape sequence
        } else {
            destuffed_data[j++] = stuffed_data[i];
        }
    }
    *destuffed_len = j;
}

int main() {
    unsigned char data[100];  // To store user input data
    int len;

    printf("Enter the number of data bytes: ");
    scanf("%d", &len);

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

    unsigned char stuffed_data[200];  // To store the stuffed data
    int stuffed_len;

    unsigned char destuffed_data[100];  // To store the destuffed data
    int destuffed_len;

    printf("Original data: ");
```

```
    for (int i = 0; i < len; i++) {
        printf("0x%X ", data[i]);
    }
    printf("\n");

    // Perform byte stuffing
    byte_stuffing(data, len, stuffed_data, &stuffed_len);

    printf("Stuffed data with flag: ");
    for (int i = 0; i < stuffed_len; i++) {
        printf("0x%X ", stuffed_data[i]);
    }
    printf("\n");

    // Perform byte de-stuffing
    byte_destuffing(stuffed_data, stuffed_len, destuffed_data, &destuffed_len);

    printf("De-stuffed data: ");
    for (int i = 0; i < destuffed_len; i++) {
        printf("0x%X ", destuffed_data[i]);
    }
    printf("\n");

    return 0;
}
```

```
charish@LAPTOP-GFCS9LJ9:~/cn$ gcc byte_stuff.c
charish@LAPTOP-GFCS9LJ9:~/cn$ ./a.out
Enter the number of data bytes: 5
Enter the data bytes in hexadecimal (e.g., 45 7E 56):
45 7E 56 7D 69
Original data: 0x45 0x7E 0x56 0x7D 0x69
Stuffed data with flag: 0x7E 0x45 0x7D 0x5E 0x56 0x7D 0x5D 0x69 0x7E
De-stuffed data: 0x45 0x7E 0x56 0x7D 0x69
```

Question 2:

SERVER CODE:

```
// Server Code (Receiver) - stop_and_wait_server.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <time.h>

#define SERVER_PORT 8080
#define LOSS_PROBABILITY 20 // 20% packet loss probability

// Function prototypes
int simulate_packet_loss();
void send_ack(int sockfd, struct sockaddr_in *client_addr);

int main() {
    int sockfd;
    char buffer[1024];
    struct sockaddr_in server_addr, client_addr;
    socklen_t addr_len = sizeof(client_addr);

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

    // Create UDP socket
    if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    memset(&server_addr, 0, sizeof(server_addr));
    memset(&client_addr, 0, sizeof(client_addr));

    // Server information
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(SERVER_PORT);
```

```
    server_addr.sin_addr.s_addr = INADDR_ANY;

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

    printf("Server is listening on port %d...\n", SERVER_PORT);
while (1) {
        int n = recvfrom(sockfd, buffer, sizeof(buffer), 0, (struct sockaddr *)&client_addr, &addr_len);
        buffer[n] = '\0';
        printf("\nReceived: %s\n", buffer);

        // Simulate packet loss
        if (simulate_packet_loss()) {
            printf("Simulated packet loss for %s. Not sending ACK.\n", buffer);
        } else {
            send_ack(sockfd, &client_addr);
        }
    }

    close(sockfd);
    return 0;
}

// Simulates packet loss with a 20% chance
int simulate_packet_loss() {
    int random_value = rand() % 100;
    return random_value < LOSS_PROBABILITY;
}

// Simulates sending an acknowledgment (ACK)
void send_ack(int sockfd, struct sockaddr_in *client_addr) {
    char ack[] = "ACK";
    sendto(sockfd, ack, strlen(ack), 0, (struct sockaddr *)client_addr, sizeof(*client_addr));
    printf("ACK sent.\n");
}
```

CLIENT CODE:

```
// Client Code (Sender) - stop_and_wait_client.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include<sys/select.h>
#include <time.h>

#define SERVER_PORT 8080
#define SERVER_ADDR "127.0.0.1"
#define TIMEOUT 3 // Timeout in seconds

// Function prototypes
void send_packet(int sockfd, struct sockaddr_in *server_addr, int packet);
int receive_ack(int sockfd);

int main() {
   int sockfd, total_packets, packet = 1, ack_received;
   struct sockaddr_in server_addr;
   socklen_t addr_len = sizeof(server_addr);

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

   // Create UDP socket
   if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
      perror("Socket creation failed");
      exit(EXIT_FAILURE);
   }

   memset(&server_addr, 0, sizeof(server_addr));

   // Server information
   server_addr.sin_family = AF_INET;
   server_addr.sin_port = htons(SERVER_PORT);
```

```
    server_addr.sin_addr.s_addr = inet_addr(SERVER_ADDR);

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

    while (packet <= total_packets) {
        printf("\nSending Packet %d...\n", packet);
        send_packet(sockfd, &server_addr, packet);

        ack_received = 0;

        for (int attempts = 0; attempts < TIMEOUT; attempts++) {
            ack_received = receive_ack(sockfd);
            if (ack_received) {
                break;
            } else {
                printf("Timeout or lost packet. Retransmitting Packet %d...\n", packet);
                send_packet(sockfd, &server_addr, packet);
            }
        }

        if (ack_received) {
            printf("Acknowledgment received for Packet %d\n", packet);
            packet++;
        } else {
            printf("Failed to receive acknowledgment for Packet %d\n", packet);
        }
    }

    printf("\nTransmission complete.\n");

    close(sockfd);
    return 0;
}

// Simulates sending a packet
void send_packet(int sockfd, struct sockaddr_in *server_addr, int packet) {
    char buffer[1024];
```

```
    snprintf(buffer, sizeof(buffer), "Packet %d", packet);
    sendto(sockfd, buffer, strlen(buffer), 0, (struct sockaddr *)server_addr, sizeof(*server_addr));
    printf("Packet %d sent.\n", packet);
}

// Wait for acknowledgment from the server
int receive_ack(int sockfd) {
    char buffer[1024];
    struct sockaddr_in from_addr;
    socklen_t from_len = sizeof(from_addr);

    // Set socket timeout for receiving ACK
    struct timeval timeout;
    timeout.tv_sec = TIMEOUT;
    timeout.tv_usec = 0;
    setsockopt(sockfd, SOL_SOCKET, SO_RCVTIMEO, &timeout, sizeof(timeout));

    int n = recvfrom(sockfd, buffer, sizeof(buffer), 0, (struct sockaddr *)&from_addr, &from_len);
    if (n > 0) {
        buffer[n] = '\0';
        if (strcmp(buffer, "ACK") == 0) {
            return 1;
        }
    }
    return 0;
}
```

```
charish@LAPTOP-GFCS9LJ9:~/cn$ gcc swarq_server.c
charish@LAPTOP-GFCS9LJ9:~/cn$ ./a.out
Server is listening on port 8080...

Received: Packet 1
ACK sent.

Received: Packet 2
ACK sent.

Received: Packet 3
ACK sent.

Received: Packet 4
ACK sent.

Received: Packet 5
Simulated packet loss for Packet 5. Not sending ACK.

Received: Packet 5
ACK sent.
```

```
charish@LAPTOP-GFCS9LJ9:~/cn$ gcc swarq_client.c
charish@LAPTOP-GFCS9LJ9:~/cn$ ./a.out
Enter the total number of packets to be transmitted: 5

Sending Packet 1...
Packet 1 sent.
Acknowledgment received for Packet 1

Sending Packet 2...
Packet 2 sent.
Acknowledgment received for Packet 2

Sending Packet 3...
Packet 3 sent.
Acknowledgment received for Packet 3

Sending Packet 4...
Packet 4 sent.
Acknowledgment received for Packet 4

Sending Packet 5...
Packet 5 sent.
Timeout or lost packet. Retransmitting Packet 5...
Packet 5 sent.
Acknowledgment received for Packet 5

Transmission complete.
```

Question 3:

SERVER CODE:

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

#define SERVER_PORT 8080
#define LOSS_PROBABILITY 20 // 20% packet loss probability

// Function prototypes
int simulate_packet_loss();
void send_ack(int sockfd, struct sockaddr_in *client_addr, int ack_num);

int main() {
    int sockfd;
    char buffer[1024];
    struct sockaddr_in server_addr, client_addr;
    socklen_t addr_len = sizeof(client_addr);
    int expected_packet = 0;

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

    // Create UDP socket
    if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    memset(&server_addr, 0, sizeof(server_addr));
    memset(&client_addr, 0, sizeof(client_addr));

    // Server information
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(SERVER_PORT);
```

```
    server_addr.sin_addr.s_addr = INADDR_ANY;

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

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

    while (1) {
        int n = recvfrom(sockfd, buffer, sizeof(buffer), 0, (struct sockaddr *)&client_addr, &addr_len);
        buffer[n] = '\0';

        int received_packet;
        if (sscanf(buffer, "Packet %d", &received_packet) == 1) {
            printf("\nReceived: %s\n", buffer);

            // Simulate packet loss
            if (simulate_packet_loss()) {
                printf("Simulated packet loss for %s. Not sending ACK.\n", buffer);
            } else {
                if (received_packet == expected_packet) {
                    // Send cumulative ACK
                    send_ack(sockfd, &client_addr, expected_packet);
                    expected_packet++;
                } else {
                    // Send cumulative ACK for the last correctly received packet
                    send_ack(sockfd, &client_addr, expected_packet - 1);
                }
            }
        }
    }

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

```
// Simulates packet loss with a 20% chance
int simulate_packet_loss() {
    int random_value = rand() % 100;
    return random_value < LOSS_PROBABILITY;
}

// Simulates sending an acknowledgment (ACK)
void send_ack(int sockfd, struct sockaddr_in *client_addr, int ack_num) {
    char ack[50];
    snprintf(ack, sizeof(ack), "ACK %d", ack_num);
    sendto(sockfd, ack, strlen(ack), 0, (struct sockaddr *)client_addr, sizeof(*client_addr));
    printf("ACK %d sent.\n", ack_num);
}
```

CLIENT CODE:

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

#define SERVER_PORT 8080
#define SERVER_ADDR "127.0.0.1"
#define LOSS_PROBABILITY 20 // 20% packet loss probability
#define TIMEOUT 3 // Timeout in seconds
#define MAX_RETRIES 5 // Maximum number of retries for each packet

// Function prototypes
void send_packet(int sockfd, struct sockaddr_in *server_addr, int packet);
int receive_ack(int sockfd, int expected_ack);

int main() {
    int sockfd, total_packets, window_size, base = 0, next_seq_num = 0, i, ack_received;
    struct sockaddr_in server_addr;
    socklen_t addr_len = sizeof(server_addr);

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

    // Create UDP socket
    if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    memset(&server_addr, 0, sizeof(server_addr));

    // Server information
    server_addr.sin_family = AF_INET;
```

```
server_addr.sin_port = htons(SERVER_PORT);
server_addr.sin_addr.s_addr = inet_addr(SERVER_ADDR);

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

while (base < total_packets) {
  // Send packets in the window
  for (i = base; i < base + window_size && i < total_packets; i++) {
    printf("\nSending Packet %d...\n", i);
    send_packet(sockfd, &server_addr, i);
  }

  // Wait for ACKs
  int retries;
  for (i = base; i < base + window_size && i < total_packets; i++) {
    retries = 0;
    ack_received = 0;
    while (retries < MAX_RETRIES) {
      ack_received = receive_ack(sockfd, i);
      if (ack_received) {
        printf("Acknowledgment received for Packet %d\n", i);
        break;
      } else {
        printf("Timeout or packet loss detected. Retransmitting Packet %d...\n", i);
        send_packet(sockfd, &server_addr, i);
        retries++;
      }
    }
    if (retries == MAX_RETRIES && !ack_received) {
      printf("Failed to receive acknowledgment for Packet %d after %d retries.\n", i, MAX_RETRIES);
    }
  }

  if (ack_received) {
    base += (i - base); // Slide the window
```

```
        }
    }

    printf("\nTransmission complete.\n");

    close(sockfd);
    return 0;
}
// Simulates sending a packet
void send_packet(int sockfd, struct sockaddr_in *server_addr, int packet) {
    char buffer[1024];
    snprintf(buffer, sizeof(buffer), "Packet %d", packet);
    sendto(sockfd, buffer, strlen(buffer), 0, (struct sockaddr *)server_addr, sizeof(*server_addr));
    printf("Packet %d sent.\n", packet);
}

// Wait for acknowledgment from the server
int receive_ack(int sockfd, int expected_ack) {
    char buffer[1024];
    struct sockaddr_in from_addr;
    socklen_t from_len = sizeof(from_addr);

    // Set socket timeout for receiving ACK
    struct timeval timeout;
    timeout.tv_sec = TIMEOUT;
    timeout.tv_usec = 0;
    setsockopt(sockfd, SOL_SOCKET, SO_RCVTIMEO, &timeout, sizeof(timeout));

    int n = recvfrom(sockfd, buffer, sizeof(buffer), 0, (struct sockaddr *)&from_addr, &from_len);
    if (n > 0) {
        buffer[n] = '\0';
        int ack_num;
        if (sscanf(buffer, "ACK %d", &ack_num) == 1 && ack_num == expected_ack) {
            return 1;
        }
    }
    return 0;
}
```

```
charish@LAPTOP-GFCS9LJ9:~/cn$ gcc gobackn_server.c
charish@LAPTOP-GFCS9LJ9:~/cn$ ./a.out
Server is listening on port 8080...

Received: Packet 0
ACK 0 sent.

Received: Packet 1
ACK 1 sent.

Received: Packet 2
ACK 2 sent.

Received: Packet 3
ACK 3 sent.

Received: Packet 4
ACK 4 sent.

Received: Packet 5
Simulated packet loss for Packet 5. Not sending ACK.

Received: Packet 6
ACK 4 sent.

Received: Packet 7
ACK 4 sent.

Received: Packet 5
Simulated packet loss for Packet 5. Not sending ACK.

Received: Packet 5
ACK 5 sent.

Received: Packet 6
ACK 6 sent.

Received: Packet 7
ACK 7 sent.
```

```
charish@LAPTOP-GFCS9LJ9:~/cn$ gcc gobackn_client.c
charish@LAPTOP-GFCS9LJ9:~/cn$ ./a.out
Enter the total number of packets to be transmitted: 8
Enter the window size: 4

Sending Packet 0...
Packet 0 sent.

Sending Packet 1...
Packet 1 sent.

Sending Packet 2...
Packet 2 sent.

Sending Packet 3...
Packet 3 sent.
Acknowledgment received for Packet 0
Acknowledgment received for Packet 1
Acknowledgment received for Packet 2
Acknowledgment received for Packet 3

Sending Packet 4...
Packet 4 sent.

Sending Packet 5...
Packet 5 sent.

Sending Packet 6...
Packet 6 sent.

Sending Packet 7...
Packet 7 sent.
Acknowledgment received for Packet 4
Timeout or packet loss detected. Retransmitting Packet 5...
Packet 5 sent.
Timeout or packet loss detected. Retransmitting Packet 5...
Packet 5 sent.
Acknowledgment received for Packet 5
Timeout or packet loss detected. Retransmitting Packet 6...
Packet 6 sent.
Acknowledgment received for Packet 6
```

```
Acknowledgment received for Packet 6
Timeout or packet loss detected. Retransmitting Packet 7...
Packet 7 sent.
Acknowledgment received for Packet 7

Transmission complete.
```

Question 4:

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<string.h>
#include<time.h>
#include<stdlib.h>
#include<ctype.h>
#include<unistd.h>
#include<arpa/inet.h>

#define W 5
#define P1 50
#define P2 10

char a[10];
char b[10];
void alpha9(int);
void alp(int);

int main()
{
   struct sockaddr_in ser,cli;
   int s,n,sock,i,j,c=1,f;
   unsigned int s1;
   s=socket(AF_INET,SOCK_STREAM,0);
   ser.sin_family=AF_INET;
   ser.sin_port=6500;
   ser.sin_addr.s_addr=inet_addr("127.0.0.1");
   bind(s,(struct sockaddr *) &ser, sizeof(ser));
   listen(s,1);
   n=sizeof(cli);
   sock=accept(s,(struct sockaddr *)&cli, &n);
   printf("\nTCP Connection Established.\n");
   s1=(unsigned int) time(NULL);
   srand(s1);
```

```
strcpy(b,"Time Out ");
recv(sock,a,sizeof(a),0);
f=atoi(a);
while(1)
{
   for(i=0;i<W;i++)
   {
      recv(sock,a,sizeof(a),0);
      if(strcmp(a,b)==0)
      {
         break;
      }
   }
   i=0;
   while(i<W)
   {
      L:
         j=rand()%P1;
      if(j<P2)
      {
         alp(c);
         send(sock,b,sizeof(b),0);
         goto L;
      }
      else
      {
         alpha9(c);
         if(c<=f)
         {
            printf("\nFrame %s Received ",a);
            send(sock,a,sizeof(a),0);
         }
         else
         {
            break;
         }
         c++;
      }
```

```
        if(c>f)
        {
           break;
        }
        i++;
      }
   }
   close(sock);
   close(s);
   return 0;
}

void alpha9(int z)
{
   int k,i=0,j,g;
   k=z;
   while(k>0)
   {
      i++;
      k=k/10;
   }
   g=i;
   i--;
   while(z>0)
   {
      k=z%10;
      a[i]=k+48;
      i--;
      z=z/10;
   }
   a[g]='\0';
}

void alp(int z)
{
   int k,i=1,j,g;
   k=z;
   b[0]='N';
```

```
  while(k>0)
  {
     i++;
     k=k/10;
  }
  g=i;
  i--;
  while(z>0)
  {
     k=z%10;
     b[i]=k+48;
     i--;
     z=z/10;
  }
  b[g]='\0';
}
```

CLIENT CODE:

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<string.h>
#include<time.h>
#include<stdlib.h>
#include<ctype.h>
#include<unistd.h>
#include<arpa/inet.h>

#define W 5
```

```
char a[10];
char b[10];
void alpha9(int);
int con();

int main()
{
   int s,f,wl,c=1,x,i=0,j,n,p=0,e=0;
   struct sockaddr_in ser;
   s=socket(AF_INET,SOCK_STREAM,0);
   ser.sin_family=AF_INET;
   ser.sin_port=6500;
   ser.sin_addr.s_addr=inet_addr("127.0.0.1");
   connect(s,(struct sockaddr *) &ser, sizeof(ser));
   printf("\nTCP Connection Established.\n");
   printf("\nEnter the number of Frames: ");
   scanf("%d",&f);
   alpha9(f);
   send(s,a,sizeof(a),0);
   strcpy(b,"Time Out ");
   while(1)
   {
      for(i=0;i<W;i++)
      {
         alpha9(c);
         send(s,a,sizeof(a),0);
         if(c<=f)
         {
            printf("\nFrame %d Sent",c);
            c++;
         }
```

```
}
i=0;
wl=W;
while(i<W)
{
   recv(s,a,sizeof(a),0);
   p=atoi(a);
   if(a[0]=='N')
   {
      e=con();
      if(e<f)
      {
         printf("\nNAK %d",e);
         printf("\nFrame %d sent",e);
         i--;
      }
   }
   else
   {
      if(p<=f)
      {
         printf("\nFrame %s Acknowledged",a);
         wl--;
      }
      else
      {
         break;
      }
   }
   if(p>f)
   {
```

```
            break;
          }
          i++;
        }
        if(wl==0 && c>f)
        {
          send(s,b,sizeof(b),0);
          break;
        }
        else
        {
          c=c-wl;
          wl=W;
        }
    }
    close(s);
    return 0;
}

void alpha9(int z)
{
    int k,i=0,j,g;
    k=z;
    while(k>0)
    {
      i++;
      k=k/10;
    }
    g=i;
    i--;
    while(z>0)
```

```
    {
        k=z%10;
        a[i]=k+48;
        i--;
        z=z/10;
    }
    a[g]='\0';
}

int con()
{
    char k[9];
    int i=1;
    while(a[i]!='\0')
    {
        k[i-1]=a[i];
        i++;
    }
    k[i-1]='\0';
    i=atoi(k);
    return i;
}
```

```
TCP Connection Established.

Enter the number of Frames: 15

Frame 1 Sent
Frame 2 Sent
Frame 3 Sent
Frame 4 Sent
Frame 5 Sent
Frame 1 Acknowledged
Frame 2 Acknowledged
Frame 3 Acknowledged
NAK 4
Frame 4 sent
Frame 4 Acknowledged
Frame 5 Acknowledged
Frame 6 Sent
Frame 7 Sent
Frame 8 Sent
Frame 9 Sent
Frame 10 Sent
Frame 6 Acknowledged
NAK 7
Frame 7 sent
NAK 7
Frame 7 sent
Frame 7 Acknowledged
NAK 8
Frame 8 sent
Frame 8 Acknowledged
Frame 9 Acknowledged
Frame 10 Acknowledged
Frame 11 Sent
Frame 12 Sent
Frame 13 Sent
Frame 14 Sent
Frame 15 Sent
```

```
charish@LAPTOP-GFCS9LJ9:~/cn$ gcc selective_repeat_server.c
charish@LAPTOP-GFCS9LJ9:~/cn$ ./a.out

TCP Connection Established.

Frame 1 Received
Frame 2 Received
Frame 3 Received
Frame 4 Received
Frame 5 Received
Frame 6 Received
Frame 7 Received
Frame 8 Received
Frame 9 Received
Frame 10 Received
Frame 11 Received
Frame 12 Received
Frame 13 Received
Frame 14 Received
```