מימוש מודל שרת/לקוח בשפת TCP,
בניית ומימוש פרוטוקול RUDP, ביצוע
בדיקות השוואה

ספר פרויקט – מטלה 3

קורס רשתות תקשורת

אילון יעקב קטן -325883957
אורי קיאלי - 318375318

RUDP_API.H

```
#define TIMES_TO_SEND 100
#define BITS_TO_BYTES 8

#define TIMEOUT_MICROSECS 700000
#define BUFFER_SIZE 65000
#define SERVER_IP_ADDRESS "127.0.0.1"
```

נגדיר את הקבועים הבאים:

TIMES_TO_SEND - כמות הפעמים לביצוע שליחה חוזרת.

BITS_TO_BYTES - המרה בין בייט לביט, למען הגדרת מבנה הפקטה.

TIMEOUT_MICROSECS - כמות זמן שבמיקור-שניות שנקבעה כזמן המתנה לביצוע שליחה בפרוטוקול

BUFFER_SIZE - גודל הבאפר המכיל את ההודעות בפרוטוקול.

SERVER_IP_ADDRESS - כתובת הIP של השרת הפנימי בו נשתמש בפרויקט.

```
22  typedef struct _RUDP_Header
23  {
24      unsigned short length : BITS_TO_BYTES + BITS_TO_BYTES;
25      unsigned short checksum : BITS_TO_BYTES + BITS_TO_BYTES;
26      unsigned char ack : BITS_TO_BYTES;
27      unsigned char fin : BITS_TO_BYTES;
28      unsigned char syn : BITS_TO_BYTES;
29      unsigned short seq : BITS_TO_BYTES;
30  } RUDP_Header;
31
32  typedef struct _RUDP_Packet
33  {
34      // Header for RUDP
35      RUDP_Header header;
36
37      // Message to deliver
38      char mes[BUFFER_SIZE];
39
40  } RUDP_Packet;
```

מבנה הפקטה:

הפקטה מכילה שני שדות: באפר (שדה ההודעה) וheader (שדה הגדרת אופי הפקטה)

- ה header שלנו מכיל את השדות הבאים:

- Length - גודל הפקטה.

- Checksum - תוצאת חישוב הchecksum המהווה בדיקת תקינות למידע אותו הפקטה מחזיקה. נשווה את ערך הchecksum שהתקבל מהפקטה עם חישוב שנבצע בצד המקבל.

- Ack - שדה המגדיר כי הפקטה היא פקטת ack.

- Fin - שדה המגדיר כי הפקטה היא פקטת fin.

- Syn - שדה המגדיר כי הפקטה היא פקטת syn.

- Seq - שדה המכיל את מספר הsequence של הפקטה, המסמן לצד השני איזה חתיכות מידע הוא צריך להעביר.

- Mes - ההודעה עצמה, הבאפר (65000).

```
 struct that represents RUDP Socket
def struct _rudp_socket

int socket_fd;               // UDP socket file descriptor
bool isServer;               // True if the RUDP socket acts like a server, false for client.
bool isConnected;            // True if there is an active connection, false otherwise.
struct sockaddr_in dest_addr; // Destination address. Client fills it when it connects via rudp_connect(), server fills it when it accepts a connection via rudp_accept().
DP_Socket;
```

<u>מבנה הסוקט:</u>

- Socket_fd - מזהה סוקט.
- isServer - שדה המגדיר האם הsocket הינה מסוג שרת/לקוח.
- isConnected - שדה המגדיר האם הsocket מחוברת או לא.
- dest_arr - מגדיר struct ששומר בתוכו את כתובת היעד.

```
// A struct that represents a packet
RUDP_Packet *create_Packet(void);

// A function that sets the values of the packet
void set_Packet(RUDP_Packet* packet, char ack, char fin, char syn, short seq, char mes[BUFFER_SIZE]);

// Allocates a new structure for the RUDP socket (contains basic information about the socket itself). Also creates a UDP socket as a baseline for the RUDP.
// isServer means that this socket acts like a server. If set to server socket, it also binds the socket to a specific port.
RUDP_Socket *rudp_socket(bool isServer, unsigned short int listen_port);

// Tries to connect to the other side via RUDP to given IP and port. Returns 0 on failure and 1 on success. Fails if called when the socket is connected/set to server.
int rudp_connect(RUDP_Socket *sockfd, const char *dest_ip, unsigned short int dest_port);

// Accepts incoming connection request and completes the handshake, returns 0 on failure and 1 on success. Fails if called when the socket is connected/set to client.
int rudp_accept(RUDP_Socket *sockfd);

// Receives data from the other side and put it into the buffer. Returns the number of received bytes on success, 0 if got FIN packet (disconnect), and -1 on error.
// Fails if called when the socket is disconnected.
int rudp_recv(RUDP_Socket *sockfd, void *buffer, unsigned int buffer_size);

// Sends data stores in buffer to the other side. Returns the number of sent bytes on success, 0 if got FIN packet (disconnect), and -1 on error. Fails if called when the socket is disconn
int rudp_send(RUDP_Socket *sockfd, void *buffer, unsigned int buffer_size,unsigned short seq);

// Disconnects from an actively connected socket. Returns 1 on success, 0 when the socket is already disconnected (failure).
int rudp_disconnect(RUDP_Socket *sockfd);

// This function releases all the memory allocation and resources of the socket.
int rudp_close(RUDP_Socket *sockfd);

//this function calculates the checksum
unsigned short int calculate_checksum(void *data, unsigned int bytes);
```

```c
// our functions//

// This function Sends fyn
int send_fin(RUDP_Socket *sockfd);

// This function Sends ack
int send_ack(RUDP_Socket *sockfd, int seq);

// This function calculates the checksum
unsigned short int calculate_checksum(void *data, unsigned int bytes);

//this function creates a packet
RUDP_Packet *create_Packet(void);

//this function sets the packet values
void set_Packet(RUDP_Packet *packet, char ack, char fin, char syn, short seq, char *mes);

//this function free the packet
void free_packet(RUDP_Packet *p);

```

- חתימות הפונקציות בתמונה הראשונה הינן ע"פ קובץ ה header שהסגל פרסם (תודה, עזר מאוד😊) : rudp_socket ,rudp_connect, rudp_accept, rudp_recv, rudp_send, Create_packet, rudp_disconnect rudp_close.

בנוסף, קיימות גם פונקציות העזר הבאות:

- Send fin - פונקציה השולחת הודעת fin.
- Send ack - פונקציה השולחת הודעת ack.
- Calculate_checksum - פונקציה המחשבת את הchecksum של הפקטה הנוכחית.
- Set_packet - פונקציה המקבלת מצביע לפקטה, ערכים ומזינה אותם בשדות המתאימים.
- Free_packet - פונקציה המשחררת את הזיכרון שהוקצה לפקטה בCreate_packe

RUDP_API.C

```c
RUDP_Packet *create_Packet(void)
{
    RUDP_Packet *packet = calloc(1, sizeof(RUDP_Packet));

    if (packet == NULL)
    {
        perror("no memory");
        exit(EXIT_FAILURE);
    }
    return packet;
}
```

יצירת פקטה – ניצור פקטה חדשה ונאפס בה את כל השדות. במידה ואין מספיק זיכרון נעדכן את המשתמש שקרתה שגיאה ונצא מהתוכנית.

```
16  void set_Packet(RUDP_Packet *packet, char ack, char fin, char syn, short seq, char *mes)
17  {
18
19      packet->header.fin = fin;
20      packet->header.syn = syn;
21      packet->header.seq = seq;
22      packet->header.ack = ack;
23      if (strchr(mes, '\0') == NULL)
24      { // check if the messenge is ending with \0
25          perror("messenge too long "); // because its not ending with \0 it is to long
26          exit(EXIT_FAILURE);
27      }
28      strcpy(packet->mes, mes);
29      packet->header.length = strlen(mes);
30      packet->header.checksum = calculate_checksum((void *)packet, packet->header.length), sizeof(short);
31  }
```

השמה של השדות המבוקשים בפקטה, כולל בדיקה שההודעה לא חורגת מהגודל האפשרי.

```
33  int send_fin(RUDP_Socket *sockfd)
34  {
35      RUDP_Packet *pack = create_Packet();
36      set_Packet(pack, 0, 1, 0, 0, "FIN");
37      int bytes_send = sendto(sockfd->socket_fd, (void *)pack, BUFFER_SIZE, 0, (const struct sockaddr *)&sockfd->dest_addr, sizeof(sockfd->dest_addr));
38      if (bytes_send == 0) // nice
39      {
40          printf("connection closed.\n");
41          free(pack);
42          return -1;
43      }
44      else if (bytes_send < 0)
45      {
46          perror("send(2)");
47          free(pack);
48          return 0;
49      }
50      free(pack);
51      return 1;
52  }
```

שליחת פקטת fin כולל בדיקה של שגיאות וסיווגן במידת הצורך, כולל הדפסה לכל סוג שגיאה (הקשר נסגר/בעיה בפונ' send).

```
54  int send_ack(RUDP_Socket *sockfd, int seq)
55  {
56      RUDP_Packet *pack = create_Packet();
57      set_Packet(pack, 1, 0, 0, seq + 1, "ACK");
58      int bytes_send = sendto(sockfd->socket_fd, (void *)pack, BUFFER_SIZE, 0, (struct sockaddr *)&sockfd->dest_addr, sizeof(sockfd->dest_addr));
59      if (bytes_send == 0)
60      {
61          printf("connection closed.\n");// Connection closed
62          free(pack);
63          return -1;
64      }
65      else if (bytes_send < 0)
66      {
67          perror("send(2)"); // error
68          free(pack);
69          return 0;
70      }
71      free(pack);
72      return 1;
73  }
```

שליחת פקטת ack כולל בדיקה של שגיאות וסיווגן במידת הצורך, כולל הדפסה לכל סוג שגיאה (הקשר נסגר/בעיה בפונ' send).

```
75  void free_packet(RUDP_Packet *p)
76  {
77      if (p == NULL)
78      {
79          return;
80      }
81      free(p);
82  }
```

<u>פונקציית שחרור הזיכרון</u> - כולל בדיקה האם P הוא NULL (fool-safe)

```
84  unsigned short int calculate_checksum(void *data, unsigned int bytes)
85  {
86      unsigned short int *data_pointer = (unsigned short int *)data;
87      unsigned int total_sum = 0;
88      // Main summing loop
89      while (bytes > 1)
90      {
91          total_sum += *data_pointer++;
92          bytes -= 2;
93      }
94      // Add left-over byte, if any
95      if (bytes > 0)
96          total_sum += *((unsigned char *)data_pointer);
97      // Fold 32-bit sum to 16 bits
98      while (total_sum >> 16)
99          total_sum = (total_sum & 0xFFFF) + (total_sum >> 16);
00      return (~((unsigned short int)total_sum));
01  }
```

<u>חישוב ה-checksum</u> - (הפונ' ניתנה ע"י הסגל במטלה עצמה).

```
106 RUDP_Socket *rudp_socket(bool isServer, unsigned short int listen_port)
107 {
108     RUDP_Socket *rudpSocket = malloc(sizeof(RUDP_Socket));
109     // rudpSocket->socket_fd = -1;
110
111     if ((rudpSocket->socket_fd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) == -1)
112     {
113         perror("socket(2)");
114         exit(1);
115     }
116
117     rudpSocket->isConnected = false; // sock is > 0 so we are connected
118     rudpSocket->isServer = isServer; // following the input (true/false)
119
120     memset(&rudpSocket->dest_addr, 0, sizeof(rudpSocket->dest_addr));
121     rudpSocket->dest_addr.sin_family = AF_INET;
122
123     // if we create the socket from the server we will bind here
124     if (isServer)
125     {
126         rudpSocket->dest_addr.sin_port = htons(listen_port);
127         // inet_pton(AF_INET, (const char *)SERVER_IP_ADDRESS, &(rudpSocket->dest_addr.sin_addr));
128
129         if (bind(rudpSocket->socket_fd, (struct sockaddr *)&rudpSocket->dest_addr, sizeof(rudpSocket->dest_addr)) == -1)
130         {
131             perror("bind(2)");
132             close(rudpSocket->socket_fd);
133             exit(1);
134         }
135     }
136
137     return rudpSocket;
138 }
```

הפונק' מקצה מקום לסוקט ומאתחלת את השדות שלו. במידה ואנו יוצרים לקוח RUDP נחזיר את המצביע לסוקט, אך אם אנו יוצרים שרת נגדיר את הפורט המתאים לפי האינפוט (שורה 126).

ננסה להגדיר את החיבור בין כתובת הIP של הצד השני והפורט, ובמידה וכשלנו נחזיר שגיאה.

```
142  int rudp_connect(RUDP_Socket *sockfd, const char *dest_ip, unsigned short int dest_port)
143  {
144      RUDP_Packet *sendPack = create_Packet();
145      RUDP_Packet *recvPack = create_Packet();
146
147      // Set timeout before entering the loop
148      struct timeval timeout;
149      timeout.tv_sec = 0;
150      timeout.tv_usec = TIMEOUT_MICROSECS;
151      if (setsockopt(sockfd->socket_fd, SOL_SOCKET, SO_RCVTIMEO, &timeout, sizeof(timeout)) < 0)
152      {
153          perror("setsockopt(2)");
154          free_packet(sendPack);
155          return -1;
156      }
157
158      if (sockfd->isConnected)
159      {
160          perror("Socket already connected\n");
161          close(sockfd->socket_fd); // Close the socket in case of error
162          free_packet(sendPack);
163          return -2;
164      }
165      else if (sockfd->isServer)
166      {
167          perror("Can't connect a server\n");
168          close(sockfd->socket_fd);
169          free_packet(sendPack);
170          return -3;
171      }
```

הפונקציה מנסה להתחבר לצד השני בהינתן הסוקט וכתובת IP. ראשית ניצור פקטות שליחה וקבלה (שורות 5-144). לאחר מכן, נאתחל את זמן ההמתנה לשליחה חוזרת ונבצע השמה לסוקט לפי הערכים הרצויים.
במידה וכשלנו נחזיר 1-. במידה והשדה isConnected דלוק, כלומר יש כבר חיבור בסוקט, נסגור את הסוקט כי יש כנראה שגיאה, נשחרר הפקטה ונחזיר 2-.

במידה והשדה isServer דלוק אז ארעה שגיאה, זו פונ' שהשרת מפעיל ולכן לא יכול להיות שהצד השני הוא גם שרת במודל שלנו.

```
172
173         sockfd->dest_addr.sin_port = htons(dest_port);// set the port
174         if (inet_pton(AF_INET, dest_ip, &(sockfd->dest_addr.sin_addr)) <= 0)
175         {
176             perror("error, inet_pton(3)");
177             free_packet(sendPack);
178             return -4;
179         }
180         int flag1 = 0, i = 69;
181         set_Packet(sendPack, 0, 0, 1, 0, "SYN");
182
```

נעדכן את הפורט (שורה 173) וננסה לעדכן את הIP. במידה וכשלנו נחזיר שגיאה.

תהליך לחיצת הידיים לביסוס קשר בו בחרנו במימוש הינו Two Way Handshake: צד השרת ישלח SYN בניסיון לפתוח קשר, צד הלקוח יחזיר SYN-ACK כהסכמה לכך, ולאחר מכן שני הצדדים יוכלו להתחיל להעביר ביניהם מידע. שלב הראשון בלחיצת הידיים הוא הSYN לכן נגדיר לפקטה את הערכים המתאימים (1 בשדה הsyn וההודעה תהיה SYN).

```
for (i = 0; i < TIMES_TO_SEND; i++)
{
    int ack = sendto(sockfd->socket_fd, sendPack, sizeof(RUDP_Packet), 0,
                     (struct sockaddr *)&sockfd->dest_addr, sizeof(sockfd->dest_addr));
    if (ack == 0)
    {
        printf("send failed.\n");
        continue; // Retry on send failure
    }
    else if (ack == -1)
    {
        perror("send(2)");
        free_packet(sendPack);
        return -5;
    }
    int j = 0;
    while (j < 100)
    {
        int bytes_recv = recv(sockfd->socket_fd, recvPack, sizeof(RUDP_Packet), 0);
        if (bytes_recv == 0)
        {
            printf("Receive timed out.\n");
            j++;
            continue; // Retry on timeout
        }
        else if (bytes_recv == -1)
        {
            perror("recvfrom");
            free_packet(recvPack);
            return -6;
        }

        if (!(recvPack->header.syn && recvPack->header.ack))
        {
            printf("Didn't receive SYN-ACK.\n");
            j++;
            continue; // Retry if not SYN-ACK
        }

        // Successful handshake, break out of the loop
        flag1 = 1;
        break;
    }
```

לולאה זו מבצעת שליחה חוזרת של הודעת SYN לשרת בכדי ליצור מולו חיבור. הלולאה תשלח 100 פעמים את הפקטה, ועל כל שליחה תנסה 100 פעמים לקבל בחזרה ACK. במקרה של חריגה מכמות הניסיונות החוזרים, כישלון של הפונקציה recvfrom או שנגמר הזמן המוקצב, הפונ' תדפיס הודעת שגיאה בהתאם ותחזיר את ערך השגיאה.

```
    if (j == 100)
    {
        printf("Connection request not received\n");
        free_packet(recvPack);
        free_packet(sendPack);
        return 0;
    }
    if (flag1 = 1)
    {
        break;
    }
}
if (i == TIMES_TO_SEND) // Maximum attempts reached
{
    perror("Maximum send attempts reached");
    free_packet(recvPack);
    free_packet(sendPack);
    return -7;
}

sockfd->isConnected = true;
free_packet(sendPack);
free_packet(recvPack);
return 1;
```

```
int rudp_accept(RUDP_Socket *serverSock)
{
    struct sockaddr_storage their_addr;
    socklen_t addr_size = sizeof(their_addr);
    fd_set readfds;
    struct timeval timeout;
    RUDP_Packet *recvPacket = create_Packet();
    // Set timeout for receiving connection request
    timeout.tv_sec = 1;
    timeout.tv_usec = TIMEOUT_MICROSECS;
    int i = 0, is_i_goot = 0;

    while (1) // 200 tries
    {
        // Data available on the socket (serverSock->sockfd)
        int bytes_recvd = recvfrom(serverSock->socket_fd, recvPacket, BUFFER_SIZE, 0, (struct sockaddr *)&their_addr, &addr_size);
        // Check for errors during receive
        if (bytes_recvd < 0)
        {
            if (errno == EWOULDBLOCK)
            { // Timeout occurred

                continue;
            }
            else
            {
                perror("recvfrom");
                return -2;
            }
        }
```

בקטע הקוד הזה נגדיר את פונקציית האישור של צד הלקוח, כאשר ניצור לולאה המחכה לקבלת SYN מהשרת, ומטפלת במצבי חריגות שונים.

```
        // Process received data (assuming valid RUDP packet format)
        if (recvPacket->header.syn)
        {
            // Valid connection request (SYN packet)
            serverSock->dest_addr.sin_addr = ((struct sockaddr_in *)&their_addr)->sin_addr;
            serverSock->dest_addr.sin_family = ((struct sockaddr_in *)&their_addr)->sin_family;
            serverSock->dest_addr.sin_port = ((struct sockaddr_in *)&their_addr)->sin_port;
            is_i_goot = 1;
            break;
        }
        else
        {
            printf("Unexpected packet received during connection setup, ignoring\n");
        }
        i++;
    }
    // Send a SYN-ACK response to the sender
    set_Packet(recvPacket, 1, 0, 1, 0, "SYN-ACK");
    if (sendto(serverSock->socket_fd, recvPacket, BUFFER_SIZE, 0, (struct sockaddr *)&serverSock->dest_addr, addr_size) < 0)
    {
        perror("sendto");
        return -3;
    }

    serverSock->isConnected = true;
    printf("Connection established with client.\n");
    return 1; // Return the received SYN packet
```

במידה והפקטה שהתקבלה היא אכן פקטת SYN, נשמור על הסוקט את הכתובת של המשתמש להמשך תקשורת, ונשלח הודעת SYN-ACK ללקוח בכדי להשלים את לחיצת היד הכפולה.

```
int rudp_recv(RUDP_Socket *sockfd, void *buffer, unsigned int buffer_size) {
    //printf("%s\n", sockfd->dest_addr.sin_addr.s_addr);
    if (!sockfd->isConnected) {
        return -1;
    }
```

כאשר נתחיל את פונקציית הקבלה, ראשית נבדוק האם הסוקט כבר מחוברת ונפעל בהתאם במידה וכן.

```
while (1) {
    if(!sockfd->isConnected){
        printf("Socket is disconnected.\n");
        free_packet(receivePacket);
        return -1;
    }
    bytes_rec = recvfrom(sockfd->socket_fd, receivePacket, sizeof(RUDP_Packet), 0,
                         (struct sockaddr *)&sockfd->dest_addr, &dest_addr_len);
    if (bytes_rec < 0) {
        // Handle recvfrom error (log details using errno)
        printf("recvfrom error: %s.\n", strerror(errno));
        retries++;
        if (retries >= TIMES_TO_SEND) {  // Define a reasonable maximum
            free_packet(receivePacket);
            return -1;
        }
        timeout.tv_usec *= 2;  // Increase timeout on retries (exponential backoff)
    } else if (bytes_rec == 0) {
        free_packet(receivePacket);
        fprintf(stdout, "%s:%d disconnected.\n", inet_ntoa(sockfd->dest_addr.sin_addr), (int)ntohs(sockfd->dest_addr.sin_port));
        close(sockfd->socket_fd);
        exit(1);
    } else {
        break;  // Received data, exit the loop
    }
}
```

בלולאה זו נגדיר את ניסיונות הקבלה של הפקטה שצד הלקוח שולח, תוך בדיקת זמן חיבור, כמות ניסיונות, והאם הפקטה נשלחה כראוי.

```
// Reset timeout to initial value for future receives
timeout.tv_usec = initial_timeout;

// Check if the checksum is correct
if (calculate_checksum(receivePacket->mes , receivePacket->header.length) != receivePacket->header.checksum)
{
    printf("Checksum wasn't the same as the calc, error.\n");
    free_packet(receivePacket);
    return -1;
}

if (receivePacket->header.fin)
{
    free_packet(receivePacket);
    return -2;
}

strncpy(buffer, receivePacket->mes, buffer_size);

send_ack(sockfd, receivePacket->header.seq + bytes_rec);
free_packet(receivePacket);

return receivePacket->header.seq + bytes_rec;
```

במידה ואכן התקבלה פקטה, נחשב את הchecksum של המידע שקיבלנו ונשווה אותו לערך השדה הנ"ל בפקטה, ונטפל בשגיאות. במידה והכל עבר בצורה חלקה, נשלח הודעת ACK לצד הלקוח, ובה מספר הSequence Number הבא אותו אנו מצפים לקבל.

```
int rudp_send(RUDP_Socket *sockfd, void *buffer, unsigned int buffer_size, unsigned int seq) {
    int bytes_sent = 0;
    int timeout_oc = 0;
    RUDP_Packet *pack = create_Packet();
    struct timeval timeout;
    timeout.tv_sec = 0;
    timeout.tv_usec = 500;
    socklen_t dest_addr_len = sizeof(struct sockaddr);
    unsigned int seqNum = 0;
    int i = 0;

    if (!sockfd->isConnected) {
        printf("Socket isn't connected.\n");
        close(sockfd->socket_fd);
        free_packet(pack);
        return 0;
    }

    else if (sockfd->isServer) {
        printf("Server can't send data.\n");
        close(sockfd->socket_fd); // Just close the socket
        free_packet(pack);
        return 0;
    }

    set_Packet(pack, 0, 0, 0, seq, buffer);
```

בתחילת פונקציית השליחה נבצע בדיקות תקינות לsocket, ולאחר מכן נגדיר פקטת RUDP חדשה בהתבסס על המידע ומספר הseq שהתקבלו בקריאה לפונקציה.

```
for (i = 0; i < TIMES_TO_SEND; i++) {
    bytes_sent = sendto(sockfd->socket_fd, pack, BUFFER_SIZE, SO_REUSEADDR, (const struct sockaddr *)&sockfd->dest_addr, sizeof(sockfd->dest_ad

    if (bytes_sent == 0) {
        printf("Send failed.\n");
        free_packet(pack);
        return 0;
    } else if (bytes_sent == -1) {
        // Handle specific errors
        if (errno == EWOULDBLOCK || errno == EAGAIN) {
            timeout_oc = 1;
        } else if (errno == EHOSTUNREACH || errno == ENETUNREACH) {
            // Potential issue with destination address
            printf("Error sending: Destination unreachable (%s). Retrying...\n", strerror(errno));
        } else {
            perror("sendto");
            free_packet(pack);
            return 0;
        }
    }
```

בתחילת הלולאה, נבצע 100 פעמים את שליחת הפקטה שיצרנו, ונטפל בבעיות שונות שעלולות להיווצר, כגון בעיות בסוקט ובtimeout. במידה ואכן התרחש TO, נסגור את החיבור.

```
    if (timeout_oc) {
        perror("Timeout occured - Aborting Connection\n");
        free_packet(pack);
        return -2;
    }

    int ack = recvfrom(sockfd->socket_fd, pack, sizeof(RUDP_Packet), 0, (struct sockaddr *)&sockfd->dest_addr, &dest_addr_len);
    seqNum = pack->header.seq;
    if (ack > 0) {
        // Check if received packet is an ACK for the current sequence number
        if (pack->header.ack) {
            // Received ACK, break out
            break;
        }
    } else if (ack < 0) {
        if (ack == 0) {
            printf("Receive failed.\n");
            free_packet(pack);
            return -1;
        } else {
            continue;
        }

    }
}
```

לאחר מכן, ננסה לקבל את הודעה הACK שנשלחה ע"י השרת, ונשמור את מספר הSEQ בכדי שנוכל להחזיר אותו, כדי לשלוח את החלק הבא של הקובץ. נוודא כי אכן קיבלנו הודעת ACK ושקבלת ההודעה התבצעה כראוי. לבסוף נוודא כי אכן התהליך התבצע ללא חריגה מכמות האיטרציות הרצויה ונחזיר את הSEQ.

```
        if (i == TIMES_TO_SEND)
        {
            return -1;
        }
        free_packet(pack);
        return seqNum;
```

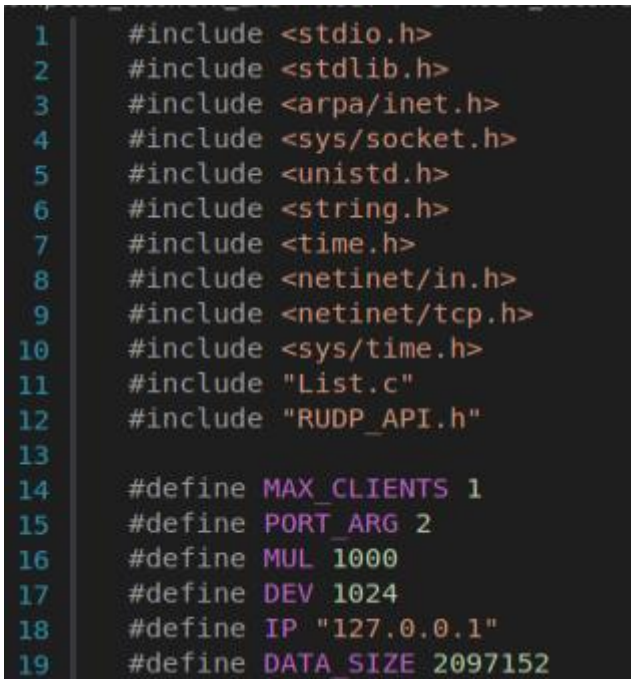
```
// Disconnects from an actively connected socket. Returns 1 on success, 0 when the socket is already disconnected (failure).
int rudp_disconnect(RUDP_Socket *sockfd)
{
    RUDP_Packet *pack = create_Packet();

    if (!sockfd->isConnected)
    {
        perror("Socket already disconnected");
        free_packet(pack);
        return 0;
    }

    int sent = send_fin(sockfd);

    if (sent)
    {
        sockfd->isConnected = false;
        free_packet(pack);
        return 1;
    }
}
```

בכדי להתנתק מהקשר, נשלח הודעת FIN לצד השני בכדי להודיע על סיום הקשר.

```
// This function releases all the memory allocation and resources of the socket.
int rudp_close(RUDP_Socket *sockfd)
{
    if (sockfd == NULL)
    {
        return -1; // Or handle the error case differently
    }
    close(sockfd->socket_fd);
    return 0;
}
```

לאחר מכן נשחרר את הזיכרון המוקצה ונסגור את הסוקט.

RECEIVER.C



```c
#include <stdio.h>
#include <stdlib.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <unistd.h>
#include <string.h>
#include <time.h>
#include <netinet/in.h>
#include <netinet/tcp.h>
#include <sys/time.h>
#include "List.c"
#include "RUDP_API.h"

#define MAX_CLIENTS 1
#define PORT_ARG 2
#define MUL 1000
#define DEV 1024
#define IP "127.0.0.1"
#define DATA_SIZE 2097152
```

כל ה include הם לצורכי הקוד, בנוסף יש לנו את הקבועים הבאים:

MAX_CLIENTS - יש לנו בפועל רק לקוח יחיד (sender).

PORT_ARG - הפורט מועבר בארגומנט השני משורת הפקודה

MUL, DEV - שני קבועים שנשתמש בהם בהמשך עבור חישוב זמן העברת הקובץ ואורך הפס.

IP- כתובת ה ip שנשתמש בה.

DATA_SIZE- גודל הקובץ (בערך 2 מגה בייט).

```c
if (strcmp(argv[PORT_ARG - 1], "-p"))
{
    printf("Invalid Arguments!\n");
    exit(EXIT_FAILURE);
}

regex_t regex;
int result;
char port_number[] = "443"; // Replace with the port number to validate

// Compile the regular expression
result = regcomp(&regex, port_regex, REG_EXTENDED);
if (result != 0)
{
    fprintf(stderr, "Error compiling regex: %s\n", strerror(result));
    return 1;
}
```

```c
int main(int argc, char *argv[])
{
    double total_t = 0;
    int bytes_received = 0;
    struct timeval start, end;
    char buffer[DATA_SIZE] = {0}; // Buffer to store the message from the client.
    List *dataList = List_alloc();// List to store the data.
    RUDP_Socket *serverSock = rudp_socket(true, (short)atoi(argv[PORT_ARG]));// Create a new RUDP socket.
    printf("Starting Receiver.\n");
    printf("Waiting for RUDP connection...\n");

    // try to connect
    int connect = rudp_accept(serverSock);
    if (connect < 1)
    {
        printf("Couldn't accepted client, aborting...\n");
        rudp_close(serverSock);
        return -1;
    }

    printf("Sender connected, beginning to receive file...\n");
```

תחילת הmain: נוודא כי אכן קיבלנו את הערכים המתאימים משורת הפקודה, נאתחל ונגדיר את כל המשתנים הרצויים לצורך תחילת קשר (זמן, באפר, הרשימה שמכילה את המידע, סוקט), וננסה ליצור קשר בעזרת rudp_accept.

```c
bytes_received = 0;

gettimeofday(&start, NULL);

while (bytes_received < DATA_SIZE)
{
    // Receive a message from the client and store it in the buffer.
    int currBytes = rudp_recv(serverSock, buffer + bytes_received, DATA_SIZE - bytes_received);

    // If the message receiving failed, print an error message and return 1.
    if (currBytes < 0)
    {
        printf("Receive was unsuccessful.\n");
        rudp_close(serverSock);
        return 1;
    }

    else if (currBytes == 0)
    {
        fprintf(stdout, "Client disconnected.\n");
        rudp_close(serverSock);
        break;
    }

    bytes_received = currBytes; // Increment the number of bytes received.
}
```

ננסה לקבל מידע - לולאה חיצונית שקיימת כל עוד לא נסגר קשר / כל ההודעה הגיעה (בעזרת הלולאה הפנימית) / אירעה שגיאה.

נוודא שהתו האחרון מסיים הודעה (אחרת נוסיף לו בסוף"\0").

```
77          gettimeofday(&end, NULL);
78          total_t = ((end.tv_sec - start.tv_sec) * MUL + ((double)(end.tv_usec - start.tv_usec) / MUL));// Ca
79          double bandwith = ((double)(DATA_SIZE / DEV) / DEV) / (total_t / MUL);// Calculate the bandwith.
80          List_insertLast(dataList, total_t, bandwith);// Insert the total time and bandwith into the list.
81
82          printf("Waiting for Sender response...\n");
83
84          int fin = rudp_recv(serverSock, buffer, sizeof(buffer));// Receive a message from the client and  c
85          if (fin == -2)
86          {
87
88              printf("Sender sent exit message.\n");
89              rudp_close(serverSock);
90              break;
91          }
92      }
93      // Print the data list and free the list.
94      List_print(dataList);
95      List_free(dataList);
96
97      printf("Receiver end.\n");
```

נחשב את הזמן ואת רוחב הפס ונוסיף אותם לרשימה לאיבר המתאים (כלומר הריצה המתאימה).

במידה ונסגר הקשר - קיבלנו פקטת fin אזי נסגור גם מהצד שלנו.

נדפיס את הרשימה ונשחרר את הזיכרון.

SENDER.C

```c
#include <stdio.h>
#include <time.h>
#include <string.h>
#include <netinet/in.h>
#include <netinet/tcp.h>
#include <arpa/inet.h>
#include "stdlib.h"
#include <sys/socket.h>
#include <unistd.h>
#include "RUDP_API.h"

#define IP_ARG 2
#define PORT_ARG 4
#define FILE_SIZE 2097152
```

כל ה include הם לצורכי הקוד, בנוסף יש לנו את הקבועים הבאים:

IP_ARG - הIP מועבר בארגומנט השני שהתכנית מקבלת

PORT_ARG - הפורט מועבר בארגומנט הרביעי שהתכנית מקבלת.

FILE_SIZE - הגודל של הקובץ הוא 2 מגה בייט.

```c
/*
 * @brief A random data generator function based on srand() and rand().
 * @param size The size of the data to generate (up to 2^32 bytes).
 * @return A pointer to the buffer.
 */
char *util_generate_random_data(unsigned int size)
{
    char *buffer = NULL;
    // Argument check.
    if (size == 0)
        return NULL;
    buffer = (char *)calloc(size, sizeof(char));
    // Error checking.
    if (buffer == NULL)
        return NULL;
    // Randomize the seed of the random number generator.
    srand(time(NULL));
    for (unsigned int i = 0; i < size; i++)
        *(buffer + i) = ((unsigned int)rand() % 256);
    return buffer;
}
```

פונ' היוצרת מידע רנדומלי (ניתן על ידי הסגל)

```
38  int main(int argc, char *argv[])
39  {
40      int seq = 0;
41      int bytesSent = 0;
42      char ans[PORT_ARG] = "yes";
43      RUDP_Socket *sock = rudp_socket(false, argv[PORT_ARG]);//create a socket
44      char *message = util_generate_random_data(FILE_SIZE);//generate random data
45
46      printf("Starting Sender.\n");
47
48      printf("Connecting to Reciever...\n");
49      int connect = rudp_connect(sock, argv[IP_ARG], (short)atoi(argv[PORT_ARG]));//connect to reciever
50      if (connect < 0)
51      {
52          printf("Failed to connect.\n");
53          rudp_close(sock);
54          return 1;
55      }
```

תחילת הmain: נוודא כי אכן כל הפרמטרים משורת הפקודה עומדים בפורמט המתאים, נאתחל ונגדיר את כל המשתנים הרצויים לצורך תחילת קשר, ניצור סוקט, נחולל הודעה רנדומית וננסה ליצור קשר בעזרת rudp_connect.

```
57      while (1)
58      {
59          seq = 0;
60          bytesSent = 0;
61
62          printf("Reciever connected, beginning to send file...\n");
63
64          while (bytesSent < FILE_SIZE)//as long as the file is not fully sent
65          {
66
67              int cur_sent = rudp_send(sock, message + bytesSent, FILE_SIZE - bytesSent, seq);//send the each chunk of the file
68              if (cur_sent == -1)
69              {
70                  printf("send failed.\n");
71                  rudp_close(sock);
72                  return 1;
73              }
74              else if (cur_sent == 0)
75              {
76                  printf("peer has closed the RUDP connection prior to send().\n");
77                  rudp_close(sock);
78                  return 1;
79              }
80              else if (cur_sent == -2) //if the file is not fully sent
81              {
82                  continue;
83              }
84              else
85              {
86                  bytesSent += (cur_sent - seq); //increment the bytes sent
87              }
88
89              seq = cur_sent;
```

ננסה לשלוח מידע כל עוד יש עוד מידע לשלוח (הבדיקה בשורה 64) / לא אירעה שגיאה.

במידה והקובץ עוד לא נשלח במלואו, נקדם המספר פקטה שנובע מכמות שנשלחה (ראו בפונ' rudp_send את אופציות ההחזרה) או שקיבלנו אחת מהשגיאות שמיוצג על ידי מספר שלילי.

```
90          }
91          printf("File was successfully sent.\n");
92          printf("Do you want to resend the file? [yes/no]: ");
93          scanf("%s", ans);
94          if (ans[0] != 'y')
95          {
96              break;
97          }
98      }
99      //free the memory and close the socket
100     free(message);
101     rudp_disconnect(sock);
102     rudp_close(sock);
103
104     printf("Sender end.\n");
105
106     return 0;
107 }
```

נשאל את האם לשלוח שוב את הקובץ. אם לא, נצא מהלולאה (בדיקה בשורה 94), נשחרר את הזיכרון, נתנתק מהסוקט ונדפיס שה sender סיים את עבודתו. אם כן, הלולאה החיצונית תשלח שוב את הקובץ.

חלק 3: מחקר

התמונות של כל הרצה לפי פרוטוקול יהיו מסודרות בסדר יורד (0 ראשון, אחריו 2...).

RUDP:

```
eylon@VM1:~/C/computer_Network_EX3/RUDP$ ./RUDP_Receiver -p 5060
5060 is a valid port number
Starting Receiver.
Waiting for RUDP connection...
Connection established with client.
Sender connected, beginning to receive file...
File transfer completed.
Waiting for Sender response...
File transfer completed.
Waiting for Sender response...
File transfer completed.
Waiting for Sender response...
File transfer completed.
Waiting for Sender response...
File transfer completed.
Waiting for Sender response...
Sender sent exit message.

---------------------------------
-          * Statistics *       -
- Run #1 Data: Time=8.59ms; Speed=232.77MB/s
- Run #2 Data: Time=6.89ms; Speed=290.23MB/s
- Run #3 Data: Time=7.68ms; Speed=260.31MB/s
- Run #4 Data: Time=9.63ms; Speed=207.68MB/s
- Run #5 Data: Time=9.63ms; Speed=207.58MB/s
-
- Average time: 8.49ms
- Average bandwith: 239.72MB/s
---------------------------------
Receiver end.
eylon@VM1:~/C/computer_Network_EX3/RUDP$
```

```
eylon@VM1:~/C/computer_Network_EX3/RUDP$ ./RUDP_Sender -ip 127.0.0.1 -p 5060
Valid IP address: 127.0.0.1
Valid port number: 5060
Starting Sender.
Connecting to Reciever...
entering the while loop
Reciever connected!
Beginning to send file...
File was successfully sent.
Do you want to resend the file? [yes/no]: yes
Beginning to send file...
File was successfully sent.
Do you want to resend the file? [yes/no]: yes
Beginning to send file...
File was successfully sent.
Do you want to resend the file? [yes/no]: yes
Beginning to send file...
File was successfully sent.
Do you want to resend the file? [yes/no]: yes
Beginning to send file...
File was successfully sent.
Do you want to resend the file? [yes/no]: no
Sender end.
eylon@VM1:~/C/computer_Network_EX3/RUDP$
```

```
on@VM1:~/C/computer_Network_EX3/RUDP$ ./RUDP_Sender -ip 127.0.0.1 -p 5060
id IP address: 127.0.0.1
id port number: 5060
rting Sender.
necting to Reciever...
iever connected!
inning to send file...
e was successfully sent.
you want to resend the file? [yes/no]: yes
inning to send file...
e was successfully sent.
you want to resend the file? [yes/no]: yes
inning to send file...
e was successfully sent.
you want to resend the file? [yes/no]: yes
inning to send file...
e was successfully sent.
you want to resend the file? [yes/no]: yes
inning to send file...
e was successfully sent.
you want to resend the file? [yes/no]: no
der end.
on@VM1:~/C/computer_Network_EX3/RUDP$
```

```
eylon@VM1:~/C/computer_Network_EX3/RUDP$ ./RUDP_Receiver -p 5060
5060 is a valid port number
Starting Receiver.
Waiting for RUDP connection...
Connection established with client.
Sender connected, beginning to receive file...
File transfer completed.
Waiting for Sender response...
File transfer completed.
Waiting for Sender response...
File transfer completed.
Waiting for Sender response...
File transfer completed.
Waiting for Sender response...
File transfer completed.
Waiting for Sender response...
Sender sent exit message.

---------------------------------
-          * Statistics *       -
- Run #1 Data: Time=735.19ms; Speed=2.72MB/s
- Run #2 Data: Time=1442.44ms; Speed=1.39MB/s
- Run #3 Data: Time=1439.08ms; Speed=1.39MB/s
- Run #4 Data: Time=7.55ms; Speed=264.97MB/s
- Run #5 Data: Time=1436.35ms; Speed=1.39MB/s
-
- Average time: 1012.12ms
- Average bandwith: 54.37MB/s
---------------------------------
Receiver end.
eylon@VM1:~/C/computer_Network_EX3/RUDP$
```

```
eylon@VM1:~/C/computer_Network_EX3/RUDP$ ./RUDP_Sender -ip 127.0.0.1 -p 5060
Valid IP address: 127.0.0.1
Valid port number: 5060
Starting Sender.
Connecting to Reciever...
Reciever connected!
Beginning to send file...
File was successfully sent.
Do you want to resend the file? [yes/no]: yes
Beginning to send file...
File was successfully sent.
Do you want to resend the file? [yes/no]: yes
Beginning to send file...
File was successfully sent.
Do you want to resend the file? [yes/no]: yes
Beginning to send file...
File was successfully sent.
Do you want to resend the file? [yes/no]: yes
Beginning to send file...
File was successfully sent.
Do you want to resend the file? [yes/no]: no
Sender end.
eylon@VM1:~/C/computer_Network_EX3/RUDP$
```

```
eylon@VM1:~/C/computer_Network_EX3/RUDP$ ./RUDP_Receiver -p 5060
5060 is a valid port number
Starting Receiver.
Waiting for RUDP connection...
Connection established with client.
Sender connected, beginning to receive file...
File transfer completed.
Waiting for Sender response...
File transfer completed.
Waiting for Sender response...
File transfer completed.
Waiting for Sender response...
File transfer completed.
Waiting for Sender response...
File transfer completed.
Waiting for Sender response...
Sender sent exit message.

---------------------------------
-          * Statistics *       -
- Run #1 Data: Time=2865.43ms; Speed=0.70MB/s
- Run #2 Data: Time=1416.70ms; Speed=1.41MB/s
- Run #3 Data: Time=2830.87ms; Speed=0.71MB/s
- Run #4 Data: Time=2147.06ms; Speed=0.93MB/s
- Run #5 Data: Time=2141.94ms; Speed=0.93MB/s
-
- Average time: 2280.40ms
- Average bandwith: 0.94MB/s
---------------------------------
Receiver end.
eylon@VM1:~/C/computer_Network_EX3/RUDP$
```

```
ylon@VM1:~/C/computer_Network_EX3/RUDP$ ./RUDP_Sender -ip 127.0.0.1 -p 5060
alid IP address: 127.0.0.1
alid port number: 5060
tarting Sender.
onnecting to Reciever...
eciever connected!
eginning to send file...
ile was successfully sent.
o you want to resend the file? [yes/no]: yes
eginning to send file...
ile was successfully sent.
o you want to resend the file? [yes/no]: yes
eginning to send file...
ile was successfully sent.
o you want to resend the file? [yes/no]: yes
eginning to send file...
ile was successfully sent.
o you want to resend the file? [yes/no]: yes
eginning to send file...
ile was successfully sent.
o you want to resend the file? [yes/no]: no
ender end.
ylon@VM1:~/C/computer_Network_EX3/RUDP$
```

```
eylon@VM1:~/C/computer_Network_EX3/RUDP$ ./RUDP_Receiver -p 5060
5060 is a valid port number
Starting Receiver.
Waiting for RUDP connection...
Connection established with client.
Sender connected, beginning to receive file...
File transfer completed.
Waiting for Sender response...
File transfer completed.
Waiting for Sender response...
File transfer completed.
Waiting for Sender response...
File transfer completed.
Waiting for Sender response...
File transfer completed.
Waiting for Sender response...
Sender sent exit message.

---------------------------------
-         * Statistics *        -
- Run #1 Data: Time=4978.09ms; Speed=0.40MB/s
- Run #2 Data: Time=8454.90ms; Speed=0.24MB/s
- Run #3 Data: Time=2113.74ms; Speed=0.95MB/s
- Run #4 Data: Time=2843.79ms; Speed=0.70MB/s
- Run #5 Data: Time=7128.83ms; Speed=0.28MB/s
-
- Average time: 5103.87ms
- Average bandwith: 0.51MB/s
---------------------------------
Receiver end.
eylon@VM1:~/C/computer_Network_EX3/RUDP$
```

:TCP RENO

```
eylon@VM1:~/C/computer_Network_EX3/TCP$ ./TCP_Receiver -p 5060 -algo reno
Starting Receiver.
Waiting for TCP connection...
Sender connected, beginning to receive file...
File transfer completed.
Waiting for Sender response...
Sender connected, beginning to receive file...
File transfer completed.
Waiting for Sender response...
Sender connected, beginning to receive file...
File transfer completed.
Waiting for Sender response...
Sender connected, beginning to receive file...
File transfer completed.
Waiting for Sender response...
Sender connected, beginning to receive file...
File transfer completed.
Waiting for Sender response...
Sender sent exit message.

---------------------------------
-          * Statistics *        -
- Run #1 Data: Time=2.06ms; Speed=972.29MB/s
- Run #2 Data: Time=1.60ms; Speed=1252.35MB/s
- Run #3 Data: Time=0.93ms; Speed=2155.17MB/s
- Run #4 Data: Time=1.00ms; Speed=2008.03MB/s
- Run #5 Data: Time=1.21ms; Speed=1658.37MB/s
-
- Average time: 1.36ms
- Average bandwith: 1609.24MB/s
---------------------------------
Receiver end.
```

```
eylon@VM1:~/C/computer_Network_EX3/TCP$ ./TCP_Sender -ip 127.0.0.1 -p 5060 -algo reno
Starting Sender.
Connecting to Reciever...
Reciever connected, beginning to send file...
File was successfully sent.
Do you want to resend the file? [yes/no]: yes
Reciever connected, beginning to send file...
File was successfully sent.
Do you want to resend the file? [yes/no]: yes
Reciever connected, beginning to send file...
File was successfully sent.
Do you want to resend the file? [yes/no]: yes
Reciever connected, beginning to send file...
File was successfully sent.
Do you want to resend the file? [yes/no]: yes
Reciever connected, beginning to send file...
File was successfully sent.
Do you want to resend the file? [yes/no]: no
Sender end.
eylon@VM1:~/C/computer_Network_EX3/TCP$
```

```
lon@VM1:~/C/computer_Network_EX3/TCP$ ./TCP_Receiver -p 5060 -algo reno
arting Receiver.
iting for TCP connection...
nder connected, beginning to receive file...
le transfer completed.
iting for Sender response...
nder connected, beginning to receive file...
le transfer completed.
iting for Sender response...
nder connected, beginning to receive file...
le transfer completed.
iting for Sender response...
nder connected, beginning to receive file...
le transfer completed.
iting for Sender response...
nder connected, beginning to receive file...
le transfer completed.
iting for Sender response...
nder sent exit message.

---------------------------------
         * Statistics *        -
Run #1 Data: Time=2.97ms; Speed=674.31MB/s
Run #2 Data: Time=1.11ms; Speed=1793.72MB/s
Run #3 Data: Time=7.58ms; Speed=263.71MB/s
Run #4 Data: Time=57.68ms; Speed=34.67MB/s
Run #5 Data: Time=1.18ms; Speed=1699.24MB/s

Average time: 14.10ms
Average bandwith: 893.13MB/s
---------------------------------
ceiver end.
```

```
eylon@VM1:~/C/computer_Network_EX3/TCP$ ./TCP_Sender -ip 127.0.0.1 -p 5060 -algo reno
Starting Sender.
Connecting to Reciever...
Reciever connected, beginning to send file...
File was successfully sent.
Do you want to resend the file? [yes/no]: yes
Reciever connected, beginning to send file...
File was successfully sent.
Do you want to resend the file? [yes/no]: yes
Reciever connected, beginning to send file...
File was successfully sent.
Do you want to resend the file? [yes/no]: yes
Reciever connected, beginning to send file...
File was successfully sent.
Do you want to resend the file? [yes/no]: yes
Reciever connected, beginning to send file...
File was successfully sent.
Do you want to resend the file? [yes/no]: no
Sender end.
eylon@VM1:~/C/computer_Network_EX3/TCP$
```

```
eylon@VM1:~/C/computer_Network_EX3/TCP$ ./TCP_Receiver -p 5060 -algo reno
Starting Receiver.
Waiting for TCP connection...
Sender connected, beginning to receive file...
File transfer completed.
Waiting for Sender response...
Sender connected, beginning to receive file...
File transfer completed.
Waiting for Sender response...
Sender connected, beginning to receive file...
File transfer completed.
Waiting for Sender response...
Sender connected, beginning to receive file...
File transfer completed.
Waiting for Sender response...
Sender connected, beginning to receive file...
File transfer completed.
Waiting for Sender response...
Sender sent exit message.

---------------------------------
-          * Statistics *        -
- Run #1 Data: Time=1.49ms; Speed=1344.09MB/s
- Run #2 Data: Time=213.11ms; Speed=9.38MB/s
- Run #3 Data: Time=0.76ms; Speed=2642.01MB/s
- Run #4 Data: Time=48.43ms; Speed=41.30MB/s
- Run #5 Data: Time=1.05ms; Speed=1902.95MB/s
-
- Average time: 52.97ms
- Average bandwith: 1187.95MB/s
---------------------------------
Receiver end.
```

```
eylon@VM1:~/C/computer_Network_EX3/TCP$ ./TCP_Sender -ip 127.0.0.1 -p 5060 -algo reno
Starting Sender.
Connecting to Reciever...
Reciever connected, beginning to send file...
File was successfully sent.
Do you want to resend the file? [yes/no]: yes
Reciever connected, beginning to send file...
File was successfully sent.
Do you want to resend the file? [yes/no]: yes
Reciever connected, beginning to send file...
File was successfully sent.
Do you want to resend the file? [yes/no]: yes
Reciever connected, beginning to send file...
File was successfully sent.
Do you want to resend the file? [yes/no]: yes
Reciever connected, beginning to send file...
File was successfully sent.
Do you want to resend the file? [yes/no]: no
Sender end.
eylon@VM1:~/C/computer_Network_EX3/TCP$
```

```
eylon@VM1:~/C/computer_Network_EX3/TCP$ ./TCP_Receiver -p 5060 -algo reno
Starting Receiver.
Waiting for TCP connection...
Sender connected, beginning to receive file...
File transfer completed.
Waiting for Sender response...
Sender connected, beginning to receive file...
File transfer completed.
Waiting for Sender response...
Sender connected, beginning to receive file...
File transfer completed.
Waiting for Sender response...
Sender connected, beginning to receive file...
File transfer completed.
Waiting for Sender response...
Sender connected, beginning to receive file...
File transfer completed.
Waiting for Sender response...
Sender sent exit message.

---------------------------------
-          * Statistics *        -
- Run #1 Data: Time=3.74ms; Speed=535.05MB/s
- Run #2 Data: Time=466.73ms; Speed=4.29MB/s
- Run #3 Data: Time=215.07ms; Speed=9.30MB/s
- Run #4 Data: Time=524.37ms; Speed=3.81MB/s
- Run #5 Data: Time=1.26ms; Speed=1581.03MB/s
-
- Average time: 242.24ms
- Average bandwith: 426.69MB/s
---------------------------------
Receiver end.
eylon@VM1:~/C/computer_Network_EX3/TCP$
```

```
eylon@VM1:~/C/computer_Network_EX3/TCP$ ./TCP_Sender -ip 127.0.0.1 -p 5060 -algo reno
Starting Sender.
Connecting to Reciever...
Reciever connected, beginning to send file...
File was successfully sent.
Do you want to resend the file? [yes/no]: yes
Reciever connected, beginning to send file...
File was successfully sent.
Do you want to resend the file? [yes/no]: yes
Reciever connected, beginning to send file...
File was successfully sent.
Do you want to resend the file? [yes/no]: yes
Reciever connected, beginning to send file...
File was successfully sent.
Do you want to resend the file? [yes/no]: yes
Reciever connected, beginning to send file...
File was successfully sent.
Do you want to resend the file? [yes/no]: no
Sender end.
eylon@VM1:~/C/computer_Network_EX3/TCP$
```

:TCP CUBIC

```
eylon@VM1:~/C/computer_Network_EX3/TCP$ ./TCP_Receiver -p 5060 -algo cubic
Starting Receiver.
Waiting for TCP connection...
Sender connected, beginning to receive file...
File transfer completed.
Waiting for Sender response...
Sender connected, beginning to receive file...
File transfer completed.
Waiting for Sender response...
Sender connected, beginning to receive file...
File transfer completed.
Waiting for Sender response...
Sender connected, beginning to receive file...
File transfer completed.
Waiting for Sender response...
Sender connected, beginning to receive file...
File transfer completed.
Waiting for Sender response...
Sender sent exit message.

---------------------------------
-          * Statistics *        -
- Run #1 Data: Time=0.80ms; Speed=2487.56MB/s
- Run #2 Data: Time=0.83ms; Speed=2406.74MB/s
- Run #3 Data: Time=0.71ms; Speed=2805.05MB/s
- Run #4 Data: Time=0.68ms; Speed=2923.98MB/s
- Run #5 Data: Time=1.38ms; Speed=1452.43MB/s
-
- Average time: 0.88ms
- Average bandwith: 2415.15MB/s
---------------------------------
Receiver end.
eylon@VM1:~/C/computer_Network_EX3/TCP$
```

```
eylon@VM1:~/C/computer_Network_EX3/TCP$ ./TCP_Sender -ip 127.0.0.1 -p 5060 -algo cubic
Starting Sender.
Connecting to Reciever...
Reciever connected, beginning to send file...
File was successfully sent.
Do you want to resend the file? [yes/no]: yes
Reciever connected, beginning to send file...
File was successfully sent.
Do you want to resend the file? [yes/no]: yes
Reciever connected, beginning to send file...
File was successfully sent.
Do you want to resend the file? [yes/no]: yes
Reciever connected, beginning to send file...
File was successfully sent.
Do you want to resend the file? [yes/no]: yes
Reciever connected, beginning to send file...
File was successfully sent.
Do you want to resend the file? [yes/no]: no
Sender end.
eylon@VM1:~/C/computer_Network_EX3/TCP$
```

```
lon@VM1:~/C/computer_Network_EX3/TCP$ ./TCP_Receiver -p 5060 -algo cubic
rting Receiver.
iting for TCP connection...
der connected, beginning to receive file...
e transfer completed.
iting for Sender response...
der connected, beginning to receive file...
e transfer completed.
iting for Sender response...
der connected, beginning to receive file...
e transfer completed.
iting for Sender response...
der connected, beginning to receive file...
e transfer completed.
iting for Sender response...
der connected, beginning to receive file...
e transfer completed.
iting for Sender response...
der sent exit message.

------------------------------
        * Statistics *        -
un #1 Data: Time=1.72ms; Speed=1160.77MB/s
un #2 Data: Time=1.86ms; Speed=1075.85MB/s
un #3 Data: Time=0.99ms; Speed=2022.24MB/s
un #4 Data: Time=0.76ms; Speed=2635.05MB/s
un #5 Data: Time=0.94ms; Speed=2139.04MB/s

verage time: 1.25ms
verage bandwith: 1806.59MB/s
------------------------------
eiver end.
lon@VM1:~/C/computer_Network_EX3/TCP$
```

```
eylon@VM1:~/C/computer_Network_EX3/TCP$ ./TCP_Sender -ip 127.0.0.1 -p 5060 -algo cubic
Starting Sender.
Connecting to Reciever...
Reciever connected, beginning to send file...
File was successfully sent.
Do you want to resend the file? [yes/no]: yes
Reciever connected, beginning to send file...
File was successfully sent.
Do you want to resend the file? [yes/no]: yes
Reciever connected, beginning to send file...
File was successfully sent.
Do you want to resend the file? [yes/no]: yes
Reciever connected, beginning to send file...
File was successfully sent.
Do you want to resend the file? [yes/no]: yes
Reciever connected, beginning to send file...
File was successfully sent.
Do you want to resend the file? [yes/no]: no
Sender end.
eylon@VM1:~/C/computer_Network_EX3/TCP$
```

```
eylon@VM1:~/C/computer_Network_EX3/TCP$ ./TCP_Receiver -p 5060 -algo cubic
Starting Receiver.
Waiting for TCP connection...
Sender connected, beginning to receive file...
File transfer completed.
Waiting for Sender response...
Sender connected, beginning to receive file...
File transfer completed.
Waiting for Sender response...
Sender connected, beginning to receive file...
File transfer completed.
Waiting for Sender response...
Sender connected, beginning to receive file...
File transfer completed.
Waiting for Sender response...
Sender connected, beginning to receive file...
File transfer completed.
Waiting for Sender response...
Sender sent exit message.

---------------------------------
-          * Statistics *        -
- Run #1 Data: Time=13.48ms; Speed=148.38MB/s
- Run #2 Data: Time=1.41ms; Speed=1415.43MB/s
- Run #3 Data: Time=62.74ms; Speed=31.88MB/s
- Run #4 Data: Time=211.13ms; Speed=9.47MB/s
- Run #5 Data: Time=0.77ms; Speed=2614.38MB/s
-
- Average time: 57.91ms
- Average bandwith: 843.91MB/s
---------------------------------
Receiver end.
eylon@VM1:~/C/computer_Network_EX3/TCP$
```

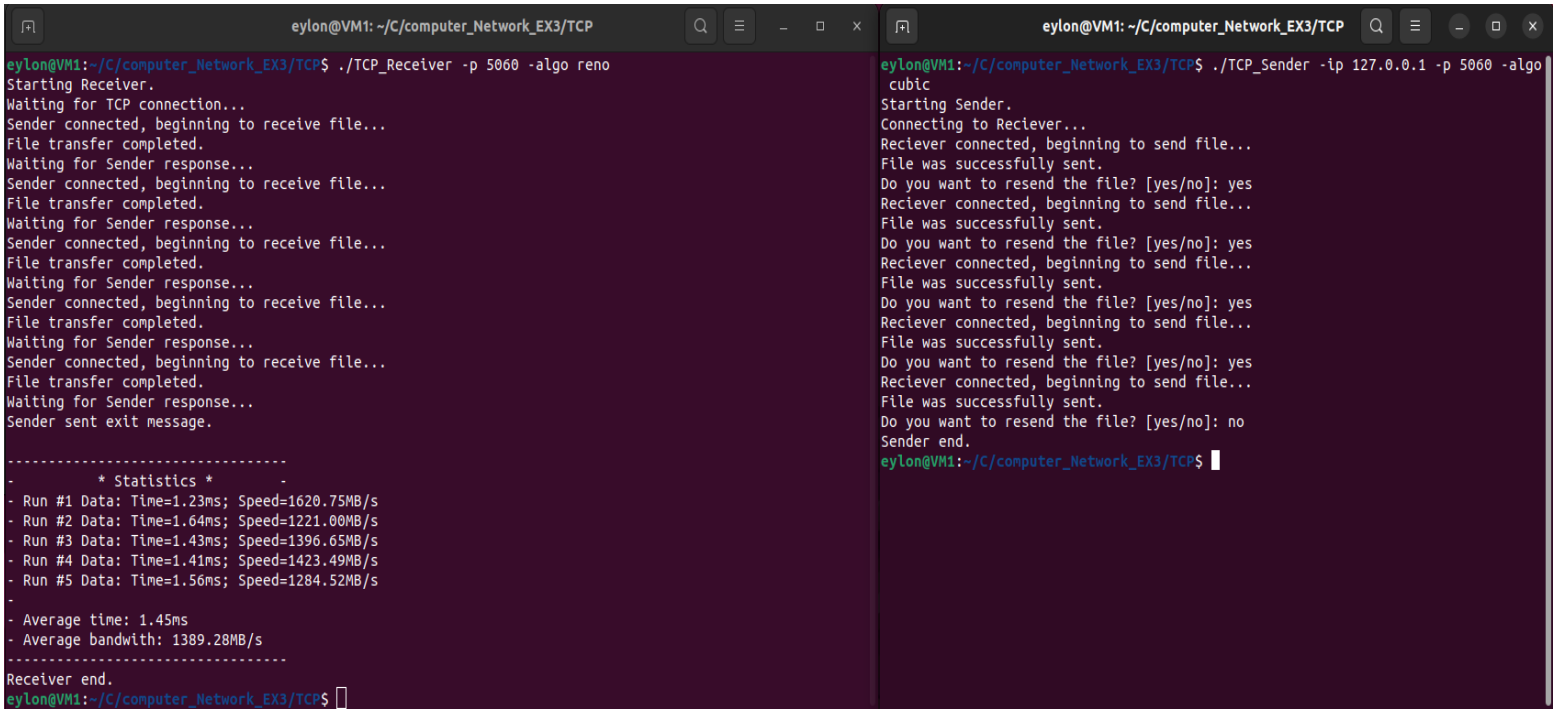
```
eylon@VM1:~/C/computer_Network_EX3/TCP$ ./TCP_Sender -ip 127.0.0.1 -p 5060 -algo cubic
Starting Sender.
Connecting to Reciever...
Reciever connected, beginning to send file...
File was successfully sent.
Do you want to resend the file? [yes/no]: yes
Reciever connected, beginning to send file...
File was successfully sent.
Do you want to resend the file? [yes/no]: yes
Reciever connected, beginning to send file...
File was successfully sent.
Do you want to resend the file? [yes/no]: yes
Reciever connected, beginning to send file...
File was successfully sent.
Do you want to resend the file? [yes/no]: yes
Reciever connected, beginning to send file...
File was successfully sent.
Do you want to resend the file? [yes/no]: no
Sender end.
eylon@VM1:~/C/computer_Network_EX3/TCP$
```

```
eylon@VM1:~/C/computer_Network_EX3/TCP$ ./TCP_Receiver -p 5060 -algo cubic
Starting Receiver.
Waiting for TCP connection...
Sender connected, beginning to receive file...
File transfer completed.
Waiting for Sender response...
Sender connected, beginning to receive file...
File transfer completed.
Waiting for Sender response...
Sender connected, beginning to receive file...
File transfer completed.
Waiting for Sender response...
Sender connected, beginning to receive file...
File transfer completed.
Waiting for Sender response...
Sender connected, beginning to receive file...
File transfer completed.
Waiting for Sender response...
Sender sent exit message.

---------------------------------
-          * Statistics *        -
- Run #1 Data: Time=1659.30ms; Speed=1.21MB/s
- Run #2 Data: Time=827.95ms; Speed=2.42MB/s
- Run #3 Data: Time=1.03ms; Speed=1943.63MB/s
- Run #4 Data: Time=1.46ms; Speed=1367.05MB/s
- Run #5 Data: Time=1.47ms; Speed=1361.47MB/s
-
- Average time: 498.24ms
- Average bandwith: 935.16MB/s
---------------------------------
Receiver end.
eylon@VM1:~/C/computer_Network_EX3/TCP$
```

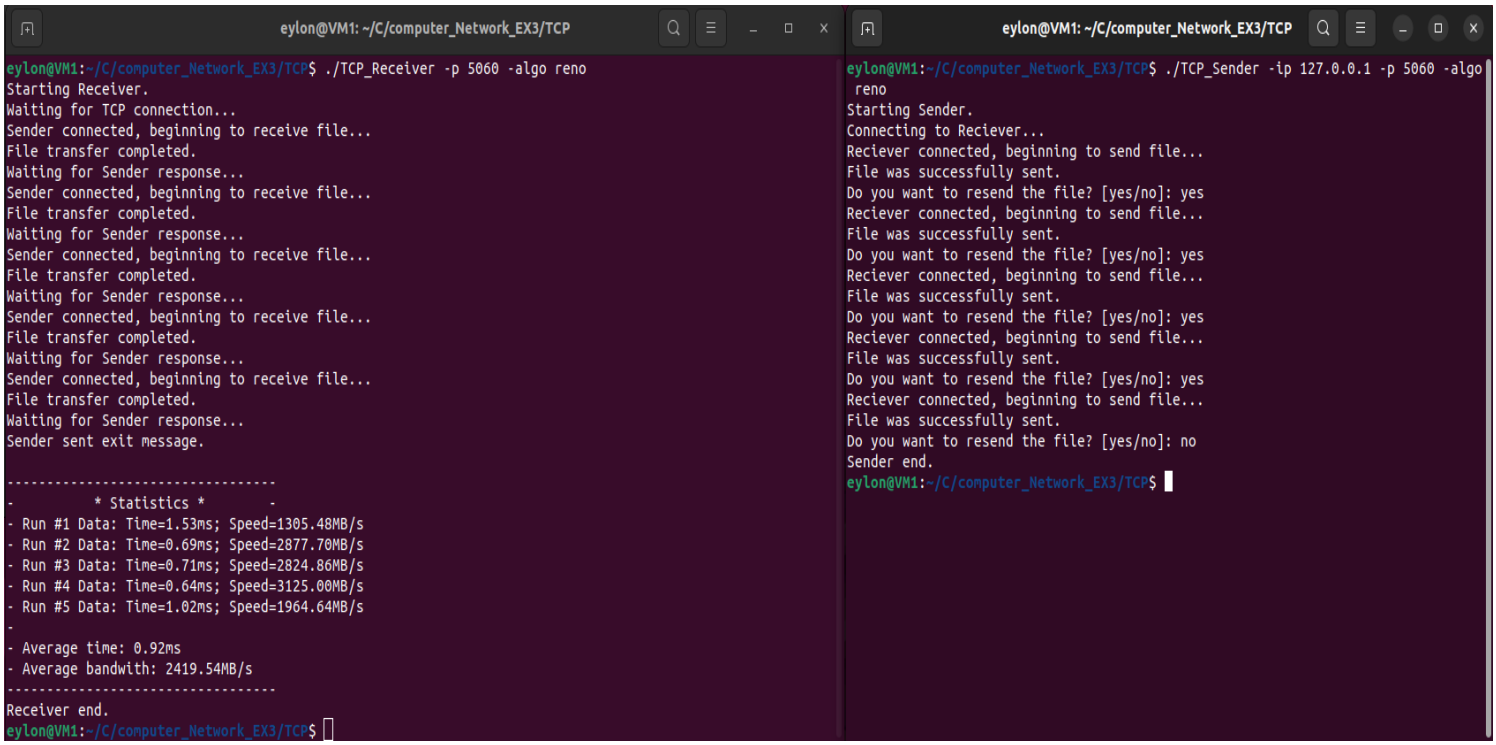
```
eylon@VM1:~/C/computer_Network_EX3/TCP$ ./TCP_Sender -ip 127.0.0.1 -p 5060 -algo cubic
Starting Sender.
Connecting to Reciever...
Reciever connected, beginning to send file...
File was successfully sent.
Do you want to resend the file? [yes/no]: yes
Reciever connected, beginning to send file...
File was successfully sent.
Do you want to resend the file? [yes/no]: yes
Reciever connected, beginning to send file...
File was successfully sent.
Do you want to resend the file? [yes/no]: yes
Reciever connected, beginning to send file...
File was successfully sent.
Do you want to resend the file? [yes/no]: yes
Reciever connected, beginning to send file...
File was successfully sent.
Do you want to resend the file? [yes/no]: no
Sender end.
eylon@VM1:~/C/computer_Network_EX3/TCP$
```

מובאים לעיל צילומי מסך של הרצות של השרת והלקוח, כל פעם עם אלגוריתם טיפול בעומס אחר:

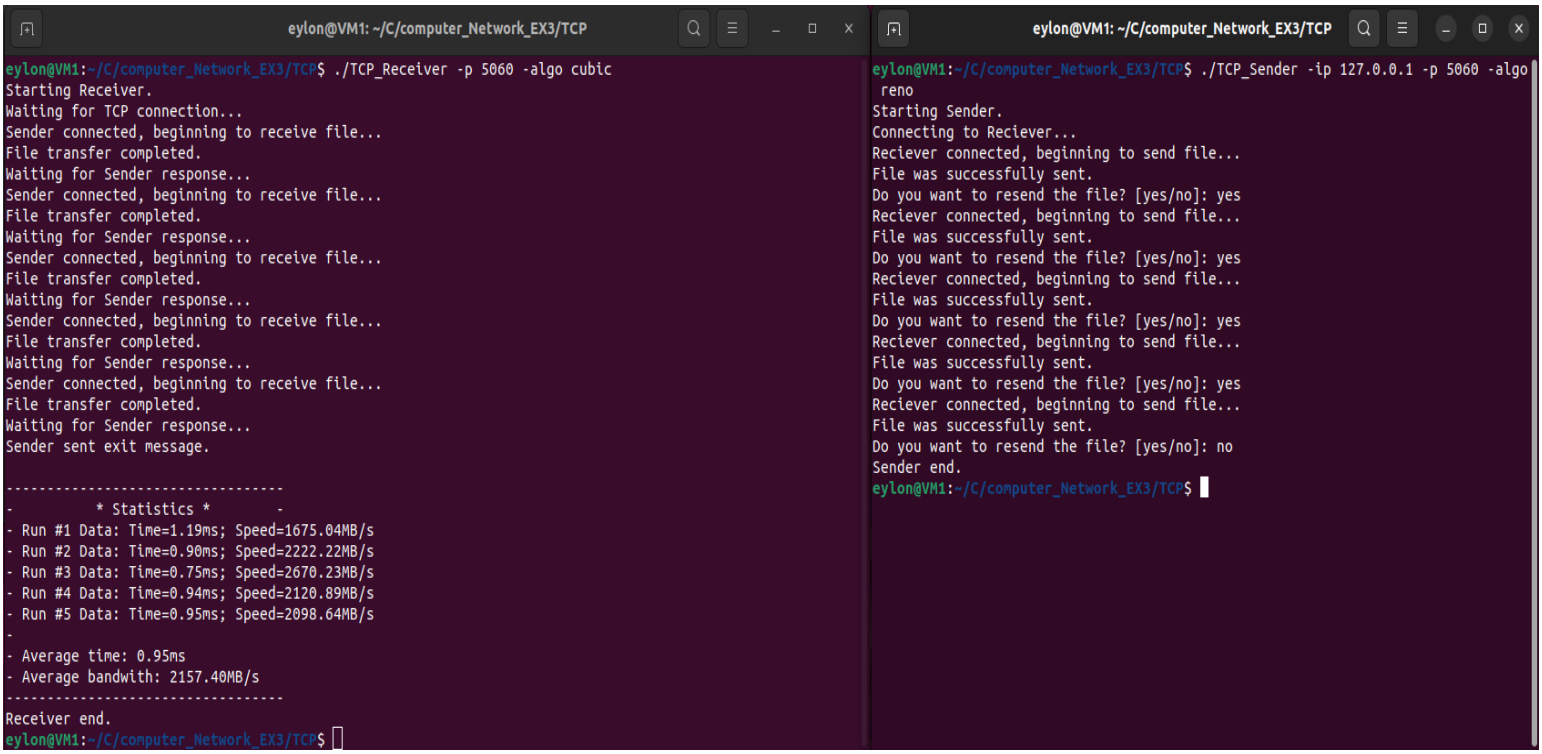
Reno → Cubic:



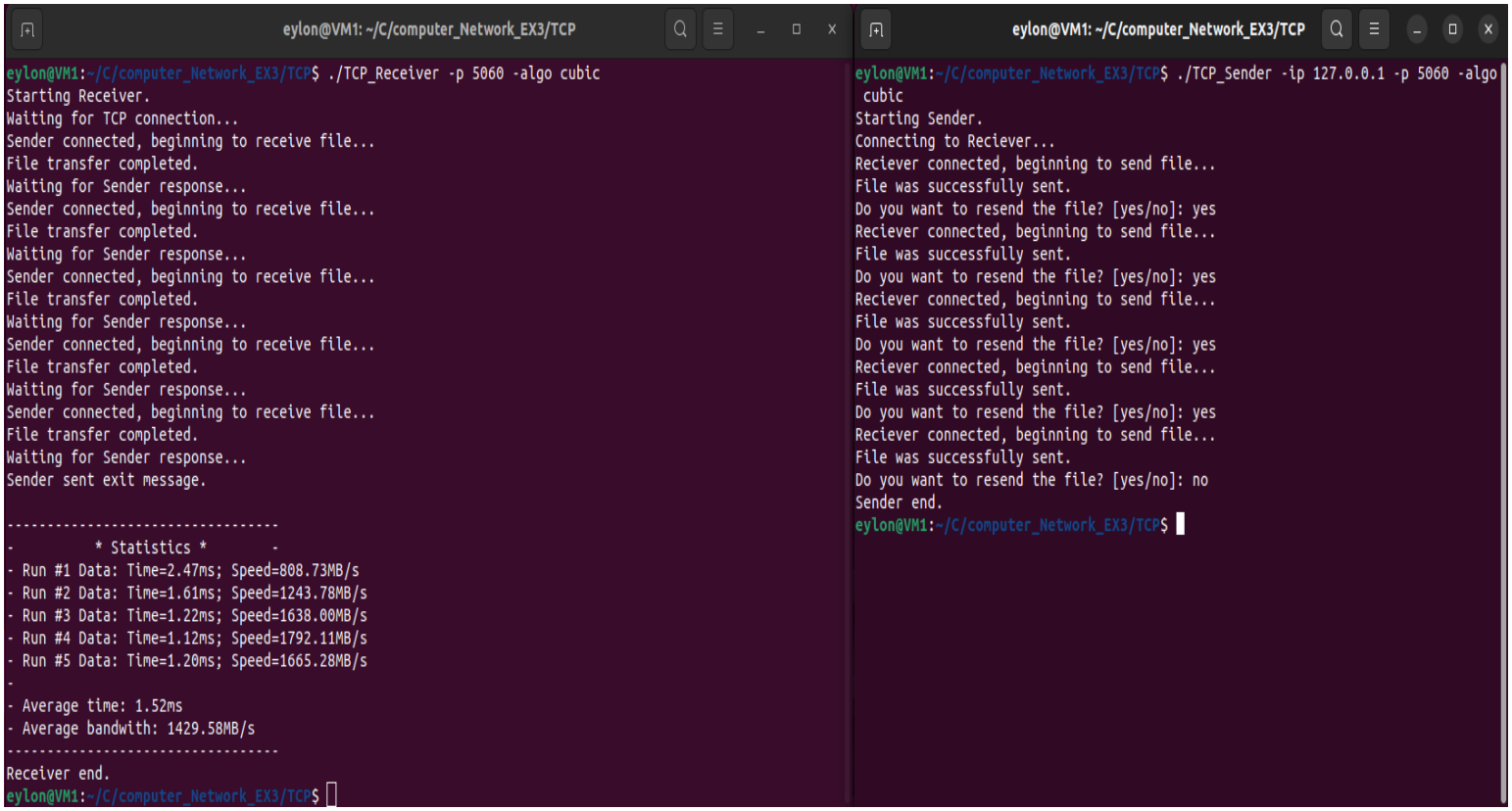
Reno → Reno:



Cubic → Reno:



Cubic → Cubic:



שאלות מחקר:

א) לפי המידע שנאסף, תוצאות ההרצות לא נותנות תשובה חד משמעית לגבי איזה אלגוריתם הוא הטוב יותר – במקרים שבהם ישנו איבוד יחסית נמוך של פקטות, נעדיף להשתמש בCUBIC, ובמקרים בהם איבוד הפקטות גבוה יותר, נעדיף את RENO. הדבר נובע מאופי האלגוריתם ובצורה בה הוא עובד.

ב) באיבוד פקטות נמוך, המימוש של RUDP עבד בצורה לא רעה, מעט פחות טובה מאשר TCP אך עדיין אמינה ומסוגלת לבצע את העבודה. לעומת זאת, כאשר אנו מתמודדים עם איבוד פקטות רב, נעדיף להשתמש באחד מאלגוריתמי השליטה של TCP, ובקורלציה לסעיף א' נעדיף את RENO.

ג) נעדיף להשתמש בTCP כאשר אנו רוצים להעביר מידע ששלמותו ותקינותו המלאה חשובה לנו, במצב בו הרשת אינה איכותית עד מאוד וקיים איבוד פקטות רב. נעדיף להשתמש בRUDP כאשר הרשת במצב טוב, ונרצה לשלוח קובץ ששלמותו ותקינותו אינם חשובים עד מאוד, אך צריכים להילקח בחשבון.

תשובות חלק עיוני (מעמ' 8):

- במצב 1: בקשר ארוך על גבי רשת אמינה עם RTT גדול.
הפתרון המוצע - להגדיל את הssthreshold - כלומר להתחיל עם לשלוח הרבה פקטות מבלי להמתין לאישור יניב תוצאה אופטימלית מהסיבות הבאות:
- בגלל שה round trip time– גדול הווה אומר הזמן שלוקח לפקטה להישלח ולקבל אישור הוא גדול גורר שככל שנשלח יותר פקטות מבלי לחכות לאישור יחסוך לנו יותר זמן
- הרשת אמינה ולכן גם אם נשלח הרבה פקטות מבלי לחכות לאישור יועיל כי ככל הנראה הן תתקבלנה
- הקשר ארוך (יש הרבה מידע לשלוח) אזי כמות פעמים שנשלח נגזרת ישירות מגודל החלון, ונרצה להישאר יותר זמן במצב ההתחלתי (אם הקשר "קצר" - מעט מידע - לא בטוח שנגיע בכלל ל ssthreshold) ולכן בקשר ארוך הוא יותר מועיל.

- נתאר את החישוב: בגלל שלא אובדות פקטות, וכל פעם הכמות מוכפלת, נשלח את כמות הפעמים שיקח אם מכפילים, לוג בבסיס 2 של S כפול הRTT (סה"כ זמן) וכל סגמנט הוא בגודל MSS אז נקבל:

$$\sum_{k=0}^{\log s} 2^k = 2 \times (S-1) \approx 2S$$

כלומר התשובה הראשונה היא הנכונה.

$$\frac{2S * MSS}{\log s * RTT}$$

- X של שנינו הוא 3
- - זמן השהייה = 1000(מרחק) לחלק ל8^10*2 (קצב התפשטות) =

$$\frac{10\text{^}3}{2 * 10\text{^}8} = \frac{1}{2 * 10\text{^}5}$$

-זמן שידור=$\frac{3*8*10\text{^}3}{8*10\text{^}9}$= $\frac{3}{1{,}000{,}000}$

-ואז נקבל RTT = $3 * (\frac{3}{10^6} + \frac{1}{2*10^5}) = \frac{3}{125{,}000}$

ומכך שאין איבוד פקטות ושידורים חוזרים נקבל שגודל החלון המקסימלי הוא (קצב התקשורת כפול RTT) לחלק לגודל הפקטה -

$$\frac{8 * 2 * 10^9 * \frac{3}{125{,}000}}{3 * 8 * 10^3} = \frac{384000}{3 * 8 * 10^3} = 16$$