מטלה 3

-

רשתות תקשורת

אמרי שי - 213023500
עדי פסח - 326627635

Part A – Transmission Control Protocol (TCP)

בחלק זה נבנה בנינו שתי תוכניות: TCP_Receiver.c, TCP_Sender.c. התוכנית TCP_Sender תשלח קובץ בגודל 3MB לTCP_Receiver, עם אפשרות לשלוח מספר פעמים, ולבסוף תשלח הודעת exit. התוכנית TCP_Receiver תאסוף את המידע ותמדוד את הזמן שלקח לו להגיע ואת מהירות השליחה.

Sender

1) Read the created file.
2) Create a TCP socket between the Sender and the Receiver.
3) Send the file.
4) User decision: Send the file again?
   a. If yes, go back to step 3.
   b. If no, continue to step 5.
5) Send an exit message to the receiver.
6) Close the TCP connection.
7) Exit.

Receiver

1) Create a TCP connection between the Receiver and the Sender.
2) Get a connection from the sender.
3) Receive the file, measure the time it took and save it.
4) Wait for Sender response:
   a. If Sender resends the file, go back to step 3.
   b. If Sender sends exit message, go to step 5.
5) Print out the times (in milliseconds), and the average bandwidth for each time the file was received.
6) Calculate the average time and the total average bandwidth.
7) Exit.

הוראות הפעלה

ראשית יש לבנות את הפרויקט עם make. לאחר מכן יש להריץ את TCP_Receiver בדרך הבאה:

./TCP_Receiver -p PORT - algo ALGO

כאשר ALGO הוא אלגוריתם בקרת הגודש הרצוי (reno או cubic) ו-PORT הוא מספר הפורט הרצוי. ואין חשיבות לסדר (אפשר לכתוב קודם אלגוריתם ואז פורט או להפך).

לאחר מכן יש להריץ את TCP_Sender בדרך הבאה:

./TCP_Sender -p PORT -algo ALGO -ip 127.0.0.1

כאשר ALGO הוא אלגוריתם בקרת הגודש הרצוי (reno או cubic) ו-PORT הוא מספר הפורט הרצוי, אבל יש לכתוב את אותו הPORT בו הרצנו את TCP_Receiver וכתובת הIP של TCP_Receiver שבחרנו היא 127.0.0.1 ולכן נשלח אליה. אין חשיבות לסדר.

הסבר הקוד

TCP_Sender.c

```
#define FILE_SIZE 3000000
```

נגדיר את גודל הקובץ להיות 3,000,000B, כלומר 3MB.

```
/*
* @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( nmemb: size,  size: sizeof(char));
    // Error checking.
    if (buffer == NULL)
        return NULL;
    // Randomize the seed of the random number generator.
    srand( seed: time( timer: NULL));
    for (unsigned int i = 0; i < size; i++)
        *(buffer + i) = ((unsigned int) rand() % 256);
    return buffer;
}
```

פונקציה שלקוחה מ- Appendix C ומייצרת מידע רנדומלי בגודל נתון.

```
void setBits(char *array, long number, int offset) {
    for (int i = 0; i < 32; i++) {
        array[i+offset] = (number >> (31 - i)) & 1;
    }
}
```

פונקציה שמקבלת מספר ומערך של char והופכת את הביטים החל מoffset לביטים של המספר.
אנו משתמשים בפונקציה זו על מנת לשנות את תחילת ההודעה לזמן השליחה.

```c
int main(int argc, char *argv[]) {
    char *file = util_generate_random_data( size: FILE_SIZE);
    puts( s: "Starting Sender...\n");

    char *SERVER_IP = (char *) malloc( size: sizeof(char) * (16));
    int port;
    char *algo = (char *) malloc( size: sizeof(char) * 10);
    if (argc != 7) {
        puts( s: "invalid command");
        return 1;
    }
    for (int i = 1; i < argc; i++) {
        if (strcmp(argv[i], "-p") == 0) { port = atoi( nptr: argv[i + 1]); }
        else if (strcmp(argv[i], "-ip") == 0) { strcpy( dest: SERVER_IP,  src: argv[i + 1]); }
        else if (strcmp(argv[i], "-algo") == 0) { strcpy( dest: algo,  src: argv[i + 1]); }
    }
```

ניצור את הקובץ הרנדומלי ונקבל את הערכים משורת הפקודה ונשמור אותם כמשתנים.

```c
    struct sockaddr_in senderAdress;
    memset( s: &senderAdress,  c: 0,  n: sizeof(senderAdress));
    senderAdress.sin_family = AF_INET;
    senderAdress.sin_port = htons( hostshort: port);
```

נאתחל את senderAdress, מבנה ששומר את כתובת השולח.

```c
int socketfd = -1; // Initializing the socket
socketfd = socket( domain: AF_INET,  type: SOCK_STREAM,  protocol: 0);
if (socketfd == -1) {
    perror( s: "socket(2)");
    free( ptr: SERVER_IP);
    free( ptr: algo);
    exit( status: 1);
}

// Setting the CC algo to the input algo.
if (setsockopt( fd: socketfd,  level: IPPROTO_TCP,  optname: TCP_CONGESTION,  optval: algo,  optlen: strlen( s: algo)) != 0) {
    perror( s: "setsockopt(2)");
    free( ptr: SERVER_IP);
    free( ptr: algo);
    exit( status: 1);
}
free( ptr: algo);
```

נאתחל את הסוקט ונבחר את שיטת בקרת בגודש.

```
if (inet_pton( af: AF_INET,  cp: SERVER_IP,  buf: &senderAdress.sin_addr) <= 0) { // setting up the IP address
    perror( s: "inet_pton(3)");
    close( fd: socketfd);
    free( ptr: SERVER_IP);
    return 1;
}
if (connect( fd: socketfd,  addr: (struct sockaddr *) &senderAdress,  len: sizeof(senderAdress)) < 0) { // connecting to receiver
    puts( s: "failed");
    perror( s: "connect(2)");
    close( fd: socketfd);
    free( ptr: SERVER_IP);
    return 1;
}
puts( s: "sending file");
```

נתחבר לכתובת ה IP שקיבלנו ונפתח קשר עם ה Receiver.

```
// adding start to file
struct timeval start, end;
gettimeofday(&start, NULL);
setBits( array: file,  number: start.tv_sec,  offset: 0);
setBits( array: file,  number: start.tv_usec,  offset: 32);
```

נשמור את הזמן הנוכחי כזמן תחילת השליחה. נשמור את הזמן הזה בתחילת ההודעה.

```
long bytes_sent = send( fd: socketfd,  buf: file,  n: FILE_SIZE,  flags: 0);
if (bytes_sent <= 0) {
    perror( s: "send(2)");
    close( fd: socketfd);
    free( ptr: SERVER_IP);
    return 1;
}
puts( s: "file sent");
```

נשלח את הקובץ.

```c
int input = -1;
while (input) {

    puts("Press 1 to resend the file, press 0 to exit.");
    scanf("%d", &input);
    if (input == 1) {
        gettimeofday(&start, NULL);
        setBits(file, start.tv_sec, 0);
        setBits(file, start.tv_usec, 32);

        bytes_sent = send(socketfd, file, FILE_SIZE, 0);
        if (bytes_sent <= 0) {
            perror("send(2)");
            close(socketfd);
            free(SERVER_IP);
            return 1;
        }
        gettimeofday(&end, NULL);
        timeTaken = (end.tv_sec - start.tv_sec) * 1000.0;
        timeTaken += (end.tv_usec - start.tv_usec) / 1000.0;
        printf("time taken: %0.3lf ms\n",timeTaken);
    }
    else if (input != 0) {
        puts("Invalid input.");
    }
}
```

נשאל את המשתמש האם הוא רוצה לשלוח את הקובץ שוב, ונמשיך לשלוח כל עוד הוא לוחץ 1. כל פעם נשמור את זמן השליחה ונצרף להודעה ובנוסף נשמור את זמן הסיום ונדפיס את כמות הזמן שלקחה השליחה.

```
bytes_sent = send( fd: socketfd,  buf: "exit",  n: 4,  flags: 0);
if (bytes_sent <= 0)
{
    perror( s: "send(2)");
    close( fd: socketfd);
    free( ptr: SERVER_IP);
    return 1;
}
close( fd: socketfd);
free( ptr: SERVER_IP);
return 0;
```

לאחר שהמשתמש בחר 0 נשלח הודעת exit ונסגור את הקשר.

TCP_Receiver.c

```c
#define FILE_SIZE 2000000
#define MAX_RUNS 100
```

נגדיר את גודל הקובץ להיות 2,000,000B, כלומר 2MB ומספר הריצות, כלומר מספר הפעמים שהמשתמש בחר לשלוח שוב את הקובץ להיות 100.

```c
void printStats(double times[100], double speeds[100], int run) {
    double timeSum = 0;
    double sum1 = 0;
    double avgTime = 0;
    double avgSpeed = 0;
    puts( s: "-------------------------------------------\n");
    puts( s: "(*) Times summary:\n\n");
    for (int i = 1; i < run; i++) {
        timeSum += times[i];
        printf( format: "(*) Run %d, Time: %0.3lf ms\n", i, times[i]);
    }
    printf( format: "\n(*) Speeds summary:\n\n");
    for (int i = 1; i < run; i++) {
        sum1 += speeds[i];
        printf( format: "(*) Run %d, Speed: %0.3lf MB/s\n", i, speeds[i]);
    }
    if (run > 0) {
        avgTime = (timeSum / (double) (run-1));
        avgSpeed = (sum1 / (double) (run-1));
    }
    printf( format: "\n(*) Time avarages:\n");
    printf( format: "(*) Avarage transfer time for whole file: %0.3lf ms\n", avgTime);
    printf( format: "(*) Avarage bandwidth: %0.3lf MB/s\n", avgSpeed);
    printf( format: "-------------------------------------------\n");

}
```

פונקציה שמדפיסה כמה זמן לקח לכל הודעה להגיע בשלומתה ואת מהירות השליחה הכוללת שלה. לבסוף היא מדפיסה את ממוצע הזמנים וממוצע המהירויות (רוחב הפס).

```c
long recreateNumber(char *array, int offset) {
    long number = 0;
    for (int i = 0; i < 32; i++) {
        number |= (array[i+offset] & 1) << (31 - i);
    }
    return number;
}
```

פונקציה שמקבלת מערך של char ונקודת התחלה (offset) ובונה מספר מ32 הביטים החל מנקודת ההתחלה.

```c
puts( s: "Starting Receiver...\n");
if (argc != 5) {
    puts( s: "invalid command");
    return 1;
}
int port = 0;
char algo[10] = {0};
for (int i = 1; i < argc; i++) {
    if (strcmp(argv[i], "-p") == 0) {
        port = atoi( nptr: argv[i + 1]);
    }
    else if (strcmp(argv[i], "-algo") == 0) {
        strcpy( dest: algo,  src: argv[i + 1]);
    }
}
```

נקבל את הערכים משורת הפקודה ונשמור אותם כמשתנים.

```
struct sockaddr_in receiverAddress, senderAddress;
struct timeval start, end;

memset( s: &receiverAddress,  c: 0,  n: sizeof(receiverAddress));
memset( s: &senderAddress,  c: 0,  n: sizeof(senderAddress));
receiverAddress.sin_family = AF_INET;
receiverAddress.sin_addr.s_addr = inet_addr( cp: "127.0.0.1");
receiverAddress.sin_port = htons( hostshort: port);
```

נאתחל את המבנים בהם נשמור את כתובת המקבל וכתובת השולח. בחרנו שהתוכנית תעבוד על IPv4 ושכתובת ה-IP של המקבל היא 127.0.0.1.

```
int socketfd = -1;
socketfd = socket( domain: AF_INET,  type: SOCK_STREAM,  protocol: 0);
if (socketfd == -1) {
    perror( s: "socket(2)");
    exit( status: 1);
}
if (setsockopt( fd: socketfd,  level: IPPROTO_TCP,  optname: TCP_CONGESTION,  optval: algo,  optlen: strlen( s: algo)) != 0) {
    perror( s: "setsockopt(2)");
    exit( status: 1);
}
```

נאתחל את הסוקט ונבחר את שיטת בקרת בגודש.

```
if (bind( fd: socketfd,  addr: (struct sockaddr *) &receiverAddress,  len: sizeof(receiverAddress)) == -1) {
    perror( s: "bind(2)");
    exit( status: 1);
}

if (listen( fd: socketfd,  n: 1) < 0) {
    perror( s: "listen(2)");
    close( fd: socketfd);
    return 1;
}
```

נקשור את הסוקט ל-IP של המקבל ולפורט שנבחר ונעביר אותו למצב הקשבה כך שהוא יכול לקבל בו זמנית עד לקוח אחד.

```
socklen_t clientAddressLen = sizeof(senderAddress);
double times[MAX_RUNS];
double speeds[MAX_RUNS];
int currentRun = 1;
```

נאתחל מבנים על מנת לשמור סטטיסטיקות מהריצות.

```
puts("Waiting for TCP connection...");
while (1) {

    int client_sock = accept(socketfd, (struct sockaddr *) &senderAddress, &clientAddressLen);
    if (client_sock < 0) {
        perror("accept(2)");
        close(socketfd);
        return 1;
    }
```

נחכה שייצרו קשר עם הסוקט ונאשר את יצירת הקשר (נשלים את לחיצת היד).

```
puts("Sender connected, beginning to receive file...\n");
char buffer[FILE_SIZE] = {0};
char data[FILE_SIZE] = {0};
ssize_t totalRecv = 0;
int first = 1;
```

נאתחל באפר בו נשמור את הביטים שהתקבלו בכל פקטה ו data בו נשמור את כל ההודעה המפורקת לפקטות.
נצטרך לספור כל פעם כמה ביטים קיבלנו כבר על מנת להסיק מתי קיבלנו את כל המידע.

```
while (currentRun < MAX_RUNS) {
    ssize_t bytes_received = recv(client_sock, buffer, FILE_SIZE, 0);
    gettimeofday(&end, NULL);
    if (first) {
        start.tv_sec = recreateNumber(buffer, 0);
        start.tv_usec = recreateNumber(buffer, 32);
        first = 0;
        strcpy(data, buffer+64);
    } else {
        strcpy(data+totalRecv-64, buffer);
    }
    totalRecv += bytes_received;
```

בכל ריצה, עד שנגיע למספר הריצות המקסימלי, נקבל פקטה ונשמור מתי קיבלנו אותה על מנת לקבל את זמן הסיום של הפקטה האחרונה. אם זוהי הפקטה הראשונה ברצף, נקח מראש הפקטה את זמן השליחה אותו צירפנו בשליחה על מנת לקבל את זמן השליחה של הפקטה הראשונה. נעתיק את המידע שקיבלנו בפקטה למערך של כל המידע data.

```
if (buffer[0]=='e'&&buffer[1]=='x'&&buffer[2]=='i'&&buffer[3]=='t' && bytes_received < 10) {
    puts("Sender sent exit message.\n");
    printStats(times, speeds, currentRun,algo);
    close(client_sock);
    close(socketfd);
    exit(1);
}
```

נבדוק האם ההודעה שקיבלנו היא הודעת יציאה. אם כן נדפיס את הסטטיסטיקות ונסגור את הקשר.

```
if (totalRecv != FILE_SIZE) {
    continue;
}
```

נחזור על צעדים אלו כל עוד כמות הביטים שקיבלנו לא תואמת את כמות הביטים של הקובץ.

```
totalRecv = 0;
first = 1;
puts("File transfer completed.\n");

if (bytes_received < 0) {
    perror("recv(2)");
    close(client_sock);
    close(socketfd);
    return 1;
}

times[currentRun] = (end.tv_sec - start.tv_sec) * 1000.0;
times[currentRun] += (end.tv_usec - start.tv_usec) / 1000.0;
speeds[currentRun] = (FILE_SIZE / times[currentRun]) / 1000;
currentRun++;
puts("Waiting for Sender response...");
```

כאשר קיבלנו את כל הביטים, נאפס את המשתנים לקראת הריצה הבאה, ונשמור את הזמן והמהירות מהריצה הנוכחית במערך על ידי חיסור זמן הסיום מזמן ההתחלה כמו שעשינו.

Part B – Reliable User Datagram Protocol (Reliable UDP or RUDP)

בחלק זה נבנה בנינו שלוש תוכניות: RUDP_Receiver, RUDP_Sender, RUDP_API. התוכנית RUDP_Sender תשלח קובץ בגודל 2MB לRUDP_Receiver, עם אפשרות לשלוח מספר פעמים. התוכנית RUDP_Receiver תאסוף את המידע ותמדוד את הזמן שלקח לו להגיע ואת מהירות השליחה. שתי תוכניות אלה משתמשות ב RUDP_API שכתבנו.

The programs will do the following:

Sender

1) Read the created file.
2) Create a UDP socket between the Sender and the Receiver.
3) Send the file via the RUDP protocol.
4) User decision: Send the file again?
   a. If yes, go back to step 3.
   b. If no, continue to step 5.
5) Send an exit message to the receiver.
6) Close the TCP connection.
7) Exit.

Receiver

1) Create a UDP connection between the Receiver and the Sender.
2) Get a connection from the sender, by the custom RUDP protocol you've built.
3) Receive the file, measure the time it took and save it.
4) Wait for Sender response:
   a. If Sender resends the file, go back to step 3.
   b. If Sender sends exit message, go to step 5.
5) Print out the times (in milliseconds), and the average bandwidth for each time the file was received.
6) Calculate the average time and the total average bandwidth.
7) Exit.

הוראות הפעלה

ראשית יש לבנות את הפרויקט עם make. לאחר מכן יש להריץ את RUDP_Receiver בדרך הבאה:

./RUDP_Receiver -p PORT

כאשר PORT הוא מספר הפורט הרצוי.

לאחר מכן יש להריץ את RUDP_Sender בדרך הבאה:

./RUDP_Sender -p PORT -ip 127.0.0.1

כאשר PORT הוא מספר הפורט הרצוי, אבל יש לכתוב את אותו הPORT בו הרצנו את RUDP_Receiver וכתובת הIP של RUDP_Receiver שבחרנו היא 127.0.0.1 ולכן נשלח אליה (בחרנו לעבוד בIPv4). אין חשיבות לסדר.

הסבר הקוד

RUDP_API.h

בקובץ זה נכתוב את הAPI של ה RUDP כך שיהיה מינימלי למשתמש. ישנם עוד פונקציות ב RUDP_API.c שלא רלוונטיות למשתמש ולכן לא נשים אותם בRUDP_API.h.

```
#define MAXLINE 2048
#define FILE_SIZE 2000000
#define MAX_RUNS 100
#define MAX_WAIT_TIME 10000000

#define FLAG_ACK 1
#define FLAG_SYN 2
#define FLAG_FIN 4

#define TIMEOUT 5000 // microseconds
```

נגדיר קבועים לתוכנית שלנו. MAXLINE הוא כמות הביטים המקסמילית הניתן לשלוח כמידע של פקטה.
FILE_SIZE הוא הגודל של הקובץ אותו שולחים. MAX_RUNS הוא כמות הריצות המקסימלית.
MAX_WAIT_TIME הוא כמות הזמן המקסימלית שנתן לשרת לחכות לבקשת פתיחת קשר לפני שייסגר.
לכל דגל נתנו ביט משלו, ACK- הביט הראשון, SYN הביט השני, FIN הביט השלישי.
הקבוע TIMEOUT הוא כמות הזמן (במיקרושניות) עד שמוחלט שפקטה נאבדה.

```
typedef struct _RudpPacket {
    uint16_t length;
    uint16_t checksum;
    uint8_t flags;
    int seq_num;
    ulong sec;
    ulong usec;
    char data[MAXLINE];

} RudpPacket;
```

זהו header הפקטה שלנו. כל פקטה שומרת ב length את כמות המידע שהיא מכילה. ב checksum את סכום הביטים של המידע שהיא מחזיקה (שומרים את ערך זה על מנת לוודא שהפקטה הגיעה תקינה), דגלים, מספר סדרתי, הזמן שליחה שלה (בשניות ומיקרו שניות) ואת המידע.

```
int rudp_send_file(char* file, int sockfd, struct sockaddr_in receiver_addr,int seqNum);
```

פונקציה לשליחת קובץ לשרת שכבר יצרנו קשר איתו.

```
int rudp_rcv_file(char* file, int sockfd, struct sockaddr_in receiver_addr, int seqNum,double* time, int* run);
```

פונקציה לקבלת קובץ מלקוח שכבר יצרנו קשר איתו. הפונקציה מעדכנת גם את הזמן שלקח לקובץ להגיע.

```
/*
 * Creating a RUDP socket and creating a handshake between two peers.
 */
int rudp_socket();
```

פונקציה לאתחול סוקט.

```
/*
 * Offer a handshake in order to establish a connection with a peer.
 * Returns -1 if failure. Returns -2 if no answer.
 */
int rudp_connect(int sockfd, struct sockaddr_in *dest_addr, socklen_t addrlen, int side);
```

פונקציה ליצירת קשר עם עמית. מחזירה 1- אם יצירת הקשר נכשלה. מחזירה 2- אם לא מקבלת בקשת יצירת קשר לזמן מה או אם מנסה לפנות לשרת ולא מקבלת תשובה.

```
/*
 * Closes a connection from a peer
 * Returns -1 if failure.
 */
int rudp_close(int sockfd, struct sockaddr_in *dest_addr, socklen_t addrlen, int side);
```

פונקציה ליצירת קשר עם עמית. מחזירה 1- אם יצירת הקשר נכשלה.

RUDP_API.c

```c
int rudp_socket() {
    return socket(domain: AF_INET, type: SOCK_DGRAM, protocol: 0);
}
```

אתחול סוקט UDP.

```c
ssize_t rudp_send(int sockfd, const RudpPacket *rudp_packet, struct sockaddr_in *serv_addr, socklen_t addrlen) {
    return sendto(fd: sockfd, buf: (const char *) rudp_packet, n: sizeof(RudpPacket) + rudp_packet->length, flags: 0,
                  addr: (struct sockaddr *) serv_addr, addr_len: addrlen);
}
```

פונקציה ששולחת פקטה לכתובת המבוקשת בסוקט נתון.

```c
ssize_t rudp_rcv(int socketfd, RudpPacket *rudp_packet, struct sockaddr_in *src_addr, socklen_t *addrlen) {
    return recvfrom(fd: socketfd, buf: rudp_packet, n: sizeof(RudpPacket) + rudp_packet->length, flags: 0, addr: (struct sockaddr *) src_addr,
                    addr_len: addrlen);
}
```

פונקציה שמקבלת פקטה בסוקט נתון.

```c
/*
 * Sending data to the peer. The function should wait for an acknowledgment packet,
 * and if it didn't receive any in timeout microseconds, retransmits the data.
 * Return the bits sent.
 */
ssize_t
rudp_send_with_timer(int sockfd, const RudpPacket *rudp_packet, struct sockaddr_in *serv_addr, socklen_t addrlen,
                     int timeout);
```

פונקציה ששולחת פקטה לכתובת המבוקשת בסוקט הנתון, מפעילה טיימר ומחכה לACK עם מספר סדרתי זהה.
אם הטיימר סיים לפני שהגיע ACK כזה, נשלח את הפקטה שוב ונחכה שוב עד שנקבל ACK מתאים.

```c
ssize_t bytes_sent;
struct timeval tv;
fd_set readfds;

// Set timeout for socket
tv.tv_sec = timeout / 1000000;
tv.tv_usec = timeout % 1000000;
setsockopt(fd: sockfd, level: SOL_SOCKET, optname: SO_RCVTIMEO, optval: (const char *) &tv, optlen: sizeof(tv));

// Send the packet
bytes_sent = rudp_send(sockfd, rudp_packet, serv_addr, addrlen);
if (bytes_sent < 0) {
    perror(s: "sendto failed");
    return -1;
}
```

נאתחל משתנים ואת הטיימר ונשלח את הפקטה.

```
// Wait for ACK with the same seqnum
while (1) {
    FD_ZERO(&readfds);
    FD_SET(sockfd, &readfds);

    int ready = select(nfds: sockfd + 1, &readfds, writefds: NULL, exceptfds: NULL, timeout: &tv);
    if (ready < 0) {
        perror(s: "select failed");
        return -1;
    } else if (ready == 0) { // Timeout reached, resend the packet and reset the timer
        bytes_sent = rudp_send(sockfd, rudp_packet, serv_addr, addrlen);
        if (bytes_sent < 0) {
            perror(s: "sendto failed");
            return -1;
        }
        // Reset the timer
        tv.tv_sec = timeout / 1000000;
        tv.tv_usec = timeout % 1000000;
        printf(format: "Timeout for packet %d\n", rudp_packet->seq_num);
        continue; // Continue waiting for ACK
```

נכנס ללואה אינסופית ונבדוק האם הטיימר סיים, אם כן נשלח את ההודעה שוב ונתחיל מהתחלה.

```
} else {
    // Socket is ready to read, check for ACK
    RudpPacket ack_packet;
    ssize_t bytes_received = rudp_rcv(socketfd: sockfd, rudp_packet: &ack_packet, src_addr: serv_addr, &addrlen);
    if (bytes_received < 0) {
        perror(s: "recvfrom failed");
        return -1;
    }
    // Check if received packet is an ACK with the same seqnum
    if ((ack_packet.flags & FLAG_ACK) && ack_packet.seq_num == rudp_packet->seq_num) {
        // ACK received, return
        //printf("Received ACK %d\n", ack_packet.seq_num);
        return bytes_received;
    }
    // If not an ACK with the same seqnum, continue waiting
```

אם הטיימר לא הסתיים וקיבלנו הודעה, נבדוק שהיא הודעת ACK ושהיא ACK על המספר הסידורי שלנו. אם כן נחזיר את הכמות הביטים שקיבלנו ונסיים.

```
/*
 * Receive data from a peer.
 * If no data received in timeout microseconds returns 0.
 */
int
rudp_rcv_with_timer(int sockfd, RudpPacket *rudp_packet, struct sockaddr_in *src_addr, socklen_t *addrlen, int timeout);
```

פונקציה שמפעילה טיימר ומחכה לקבלת פקטה בסוקט הנתון. אם הטיימר סיים לפני שהגיעה פקטה, נחזיר ACK. 0. נשתמש בפונקציה זו כאשר נרצה לחכות לשידורים חוזרים במידה שיש.

```
struct timeval tv;
fd_set readfds;

// Set timeout for socket
tv.tv_sec = timeout / 1000000;
tv.tv_usec = timeout % 1000000;

// Initialize read file descriptor set
FD_ZERO(&readfds);
FD_SET(sockfd, &readfds);
```

נאתחל את המשתנים ואת הטיימר.

```
// Wait for data to be available on the socket or timeout
int ready = select(nfds: sockfd + 1, &readfds, writefds: NULL, exceptfds: NULL, timeout: &tv);
if (ready < 0) {
    perror(s: "select failed");
    return -1;
} else if (ready == 0) {
    // Timeout reached, no data available
    return 0;
} else {
    // Data available, receive the packet
    ssize_t bytes_received = rudp_rcv(socketfd: sockfd, rudp_packet, src_addr, addrlen);
    if (bytes_received < 0) {
        perror(s: "recvfrom failed");
        return -1;
    }
    return bytes_received;
}
```

אם התקבלה פקטה לפני שהטיימר סיים נחזיר את כמות הביטים של הפקטה. אם הטיימר סיים לפני נחזיר 0.

```c
int rudp_send_file(char *file, int sockfd, struct sockaddr_in receiver_addr, int seqNum) {
    RudpPacket packet;
    packet.seq_num = seqNum;
    struct timeval tv;
    for (int i = 0; i < FILE_SIZE; i += MAXLINE) {
        memcpy(dest:packet.data, src:file + i, n:MAXLINE);
        packet.length = htons(hostshort:MAXLINE);
        packet.flags = 0;
        packet.checksum = calculate_checksum(packet.data, bytes:packet.length);
        //printf("Sending packet %d\n",packet.seq_num);
        gettimeofday(&tv, tz:NULL);
        packet.usec = tv.tv_usec;
        packet.sec = tv.tv_sec;
        if (rudp_send_with_timer(sockfd, rudp_packet:&packet, serv_addr:&receiver_addr, addrlen:sizeof(receiver_addr), timeout:TIMEOUT) < 0) {
            perror(s:"sendto failed");
            //free(packet);
            return -1;
        }
        //printf("Sent packet %d\n", packet.seq_num);
        packet.seq_num++;
    }
    return 0;
}
```

פונקציה לשליחת קובץ שלם. נפרק את הקובץ לפקטות כך שכל פקטה תחזיק את כמות המידע המקסימלית שהיא יכולה. נאתחל את השדות ב header של הפקטה, כולל checksum על המידע וזמן השליחה. נשלח את הפקטה ונחכה לACK בעזרת הפונקציה הקודמת. כך נוודא שהפקטות הגיעו ליעד. אם סיימנו בהצלחה נחזיר 0.

```c
int rudp_rcv_file(char *file, int sockfd, struct sockaddr_in sender_addr, int seqNum, double times[], int *currentRun) {
    RudpPacket packet;
    char *current = &file[0];
    socklen_t sender_addr_len = sizeof(sender_addr);
    struct timeval start, end;
    int first = 1;
    while (1) {
        //printf("Waiting for packet %d\n",seqNum);
        ssize_t bytes_recv = rudp_rcv(socketfd:sockfd, rudp_packet:&packet, src_addr:&sender_addr, addrlen:&sender_addr_len);
        while (bytes_recv < 0) {
            bytes_recv = rudp_rcv(socketfd:sockfd, rudp_packet:&packet, src_addr:&sender_addr, addrlen:&sender_addr_len);
        }
```

פונקציה זו מקבלת קובץ ושומרת אותו. ראשית נאתחל את המשתנים ונחכה לקבלת פקטה.

```c
if (first) {
    first = 0;
    start.tv_sec = packet.sec;
    start.tv_usec = packet.usec;
}
```

אם זוהי הפקטה הראשונה נשמור את זמן השליחה שלה.

```
if ((packet.flags & FLAG_FIN)) {
    //printf("Received FIN packet %d.\n",packet.seq_num);
    gettimeofday(tv:&end, tz:NULL);
    times[*currentRun] = (end.tv_sec - start.tv_sec) * 1000.0;
    times[*currentRun] += (end.tv_usec - start.tv_usec) / 1000.0;
    (*currentRun)++;
    return 0;
}
```

אם קיבלנו פקטת FIN נשמור את זמן הסיום ונסגור את הקשר.

```
packet.flags = FLAG_ACK;
//if the packet is not the expected one or the checksum is not correct, ask for retransmission
if (packet.seq_num != seqNum || calculate_checksum(packet.data, bytes:packet.length) != packet.checksum) {
    packet.seq_num = seqNum - 1;
    //printf("Packet had error. Sending ACK With previous seqnum %d\n",packet.seq_num);
    rudp_send(sockfd, rudp_packet:&packet, serv_addr:&sender_addr, addrlen:sizeof(sender_addr));
} else //if the packet is the expected one and the checksum is correct, send an ack, and update the sequence number for the next packet
```

נכין חבילת ACK. אם המספר הסדרתי של החבילה שהתקבלה שונה מהמספר הסדרתי הרצוי או שהchecksum שגוי נשלח ACK עם מספר סדרתי נמוך ב1 על מנת לבקש שידור חוזר.

```
} else //if the packet is the expected one and the checksum is correct, send an ack, and update the sequence number for the next packet
{
    //acked = 1;
    packet.seq_num = seqNum;
    //printf("Packet is good. Sending ACK %d\n",packet.seq_num);
    rudp_send(sockfd, rudp_packet:&packet, serv_addr:&sender_addr, addrlen:sizeof(sender_addr));
    seqNum++;
    char *data = packet.data;
    for (int i = 0; i < packet.length; i++) {
        *current = data[i];
        current++;
    }
}
```

אחרת נשלח חבילת ACK עם המספר הסדרתי הנוכחי, נקדם את המספר הסדרתי ונוסיף את המידע שהתקבל למערך.

```
int rudp_connect(int sockfd, struct sockaddr_in *dest_addr, socklen_t addrlen, int side) {
    if (side == 1) {
        return rudp_accept(sockfd, dest_addr, addrlen);
    }
```

בפונקציה זו ננהל את לחיצת היד להקמת קשר. אם זהו הצד שפותח את הקשר (השולח) נכניס side=0 ואם הצד שמקבל את הבקשה לפתיחת קשר (המקבל) נכניס side=1. אם זהו הצד שמקבל בקשה לפתיחת קשר נשתמש בפונקציה rudp_accept עליה נסביר בהמשך.

```
RudpPacket syn_packet, synack_packet, ack_packet;
//socklen_t* len = &addrlen;
ssize_t bytes_sent;
struct timeval tv;
fd_set readfds;

// Set timeout for socket
tv.tv_sec = TIMEOUT / 1000000;
tv.tv_usec = TIMEOUT % 1000000;
setsockopt(fd: sockfd, level: SOL_SOCKET, optname: SO_RCVTIMEO, optval: (const char *) &tv, optlen: sizeof(tv));
```

נאתחל את המשתנים והטיימר.

```
// Prepare SYN packet
syn_packet.length = htons(hostshort: 0); // No data in SYN packet
syn_packet.flags = FLAG_SYN;
syn_packet.seq_num = 0;

// Send the packet
bytes_sent = rudp_send(sockfd, rudp_packet: &syn_packet, serv_addr: dest_addr, addrlen);
if (bytes_sent < 0) {
    perror(s: "sendto failed");
    return -1;
}
printf(format: "Sent SYN %d\n", syn_packet.seq_num);
```

נכין חבילת SYN ונשלח אותה.

```
int timeoutCount = 0;
// Wait for ACK with the same seqnum
while (1) {
    FD_ZERO(&readfds);
    FD_SET(sockfd, &readfds);

    int ready = select( nfds: sockfd + 1, &readfds, writefds: NULL, exceptfds: NULL, timeout: &tv);
    if (ready < 0) {
        perror( s: "select failed");
        return -1;
    } else if (ready == 0) { // Timeout reached, resend the packet and reset the timer
        bytes_sent = rudp_send(sockfd, rudp_packet: &syn_packet, serv_addr: dest_addr, addrlen);
        if (bytes_sent < 0) {
            perror( s: "sendto failed");
            return -1;
        }
        // Reset the timer
        tv.tv_sec = TIMEOUT / 1000000;
        tv.tv_usec = TIMEOUT % 1000000;
        printf( format: "Timeout for SYN %d\n", syn_packet.seq_num);
        timeoutCount++;
        if (timeoutCount > 10) {
            return -2;
        }
        continue; // Continue waiting for ACK
```

נתחיל לחכות לSYNACK. אם הטיימר נגמר לפני נשלח את הSYN שוב ונאתחל את הטיימר. אם נכשלנו 10 פעמים נבין שיש תקלה ונוותר על הקשר.

```
} else {
    // Socket is ready to read, check for ACK
    ssize_t bytes_received = rudp_rcv( socketfd: sockfd, rudp_packet: &synack_packet, src_addr: dest_addr, &addrlen);
    if (bytes_received < 0) {
        perror( s: "recvfrom failed");
        return -1;
    }
    // Check if received packet is an ACK with the same seqnum
    if ((synack_packet.flags & FLAG_SYN) && (synack_packet.flags & FLAG_ACK) &&
        synack_packet.seq_num == syn_packet.seq_num) {
        // ACK received, return
        printf( format: "Received SYNACK %d\n", synack_packet.seq_num);
        break;
    }
```

אחרת אם קיבלנו פקטה לפני שהטיימר סיים, נבדוק שאכן קיבלנו SYNACK. מאחר שהאלגוריתם שלנו הוא לא TCP, נוכל לוותר על הACK השלישי. זאת מאחר ואנחנו מחכים לשידורים חוזרים של SYN לאחר ששלחנו את SYNACK (נראה את זה בהמשך), ולכן יש לנו וודאות שSYNACK הגיע.

```c
int rudp_accept(int sockfd, struct sockaddr_in *dest_addr, socklen_t addrlen) {
    RudpPacket syn_packet, synack_packet;
    socklen_t *len = &addrlen;

    // Receive SYN packet
    int status = rudp_rcv_with_timer(sockfd, rudp_packet:&syn_packet, src_addr:dest_addr, addrlen:len, timeout:MAX_WAIT_TIME);
    if (status < 0) {
        perror(s:"rcv failed");
        return -1;
    } else if (status == 0) {
        return -2;
    }
```

בפונקציה זו נפעיל קבלה של בקשת יצירת קשר. נאתחל את המשתנים ונמתין לקבלת פקטת SYN. אם לא התקבל תוך הזמן שקבענו, סימן שיש תקלה ונוותר על הקשר.

```c
// Check if it's a valid SYN packet
if (syn_packet.flags & FLAG_SYN) {
    // Prepare SYNACK packet
    synack_packet.length = htons(hostshort:0); // No data in ACK packet
    synack_packet.flags = FLAG_ACK + FLAG_SYN;
    synack_packet.seq_num = syn_packet.seq_num;
    // Send SYNACK packet
    //printf("Sending SYNACK packet\n");
    if (rudp_send(sockfd, rudp_packet:&synack_packet, serv_addr:dest_addr, addrlen) < 0) {
        perror(s:"sendto failed");
        return -1;
    }
} else {
    perror(s:"Invalid SYN packet\n");
    return -1; // Handshake failed
}
```

כאשר התקבל SYN נבדוק שהוא אכן SYN ונשלח פקטת SYNACK.

```c
// wait for retransmit of SYN
while (rudp_rcv_with_timer(sockfd, rudp_packet:&syn_packet, src_addr:dest_addr, addrlen:len, timeout:TIMEOUT*2)) {
    if (!(syn_packet.flags & FLAG_SYN)) {
        break;
    }
    synack_packet.length = htons(hostshort:0); // No data in ACK packet
    synack_packet.flags = FLAG_ACK + FLAG_SYN;
    synack_packet.seq_num = syn_packet.seq_num;
    // Send SYNACK packet
    //printf("Sending SYNACK packet\n");
    if (rudp_send(sockfd, rudp_packet:&synack_packet, serv_addr:dest_addr, addrlen) < 0) {
        perror(s:"sendto failed");
        return -1;
    }
}

return 0;
```

כפי שהסברנו קודם, אנחנו מבצעים לחיצת יד כפולה ולא משולשת. לכן על מנת לקבל וודאות שפקטת SYNACK

הגיעה, נחכה לשידורים חוזרים של פקטת SYN. אם הגיעה פקטת SYN, נשלח שוב SYNACK. אחרת, סימן שהקמנו את הקשר בהצלחה.

```
int rudp_close(int sockfd, struct sockaddr_in *dest_addr, socklen_t addrlen,
               int side) { //side 0 means sender, 1 means receiver
    RudpPacket packet;
    if (side == 1) {
        packet.flags = FLAG_ACK;
        rudp_send(sockfd, rudp_packet: &packet, serv_addr: dest_addr, addrlen);
        puts(s: "Sent ACK for FIN");

        // wait a little to ensure no FIN retransmission
        while (rudp_rcv_with_timer(sockfd, rudp_packet: &packet, src_addr: dest_addr, &addrlen, timeout: TIMEOUT * 2)) {
            packet.flags = FLAG_ACK;
            rudp_send(sockfd, rudp_packet: &packet, serv_addr: dest_addr, addrlen);
            puts(s: "Sent ACK for FIN");
        }
    }
```

פונקציה זו מנהלת את סגירת הקשר. בתוכנית שלנו השולח ישלח FIN ראשון למקבל. לכן אם הפונקציה נקראת מהמקבל, הוא צריך קודם לשלוח ACK על הFIN, ולחכות לשידורים חוזרים של הFIN כדי לוודא שהACK התקבל.

```
RudpPacket fin_packet;

// Prepare FIN packet
fin_packet.length = htons(hostshort: 0); // No data in SYN packet
fin_packet.flags = FLAG_FIN;
fin_packet.seq_num = ((unsigned int) rand() % 256);

// Send FIN packet
if (rudp_send_with_timer(sockfd, rudp_packet: &fin_packet, serv_addr: dest_addr, addrlen, timeout: TIMEOUT) < 0) {
    perror(s: "sendto failed");
    return -1;
}
```

נשלח פקטת FIN ונחכה לACK עליה. בשביל השולח זה הדבר הראשון שיתבצע כראוי, בשביל המקבל זה יקרה רק לאחר שהשולח קיבל ACK על הFIN כראוי וכאן הוא סיים את תפקידו והקשר נסגר בהצלחה.

```
if (side == 0) { // if it's the sender wait for the FIN
    return rudp_close_sender(sockfd, receiver_addr: *dest_addr);
}
return 0;
```

השולח לא סיים את תפקידו כי עליו לחכות לFIN מהמקבל ולאשר אותו. זה ממומש בפונקציה הבאה:

```c
int rudp_close_sender(int sockfd, struct sockaddr_in receiver_addr) {
    // Wait for the FIN from receiver
    RudpPacket last_packet;
    socklen_t len = sizeof(receiver_addr);
    ssize_t bytes_recv = rudp_rcv(socketfd: sockfd, rudp_packet: &last_packet, src_addr: &receiver_addr, addrlen: &len);
    while (bytes_recv < 0) {
        bytes_recv = rudp_rcv(socketfd: sockfd, rudp_packet: &last_packet, src_addr: &receiver_addr, addrlen: &len);
    }
    if (!(last_packet.flags & FLAG_FIN)) {
        perror(s: "last packet received needs to be FIN");
        return -1;
    }
```

ראשית, השולח מחכה לקבלת פקטה ומוודא שהיא FIN.

```c
// ACK the FIN
last_packet.flags = FLAG_ACK;
if (rudp_send(sockfd, rudp_packet: &last_packet, serv_addr: &receiver_addr, addrlen: sizeof(receiver_addr)) < 0) {
    perror(s: "sendto failed");
    //free(packet);
    return -1;
}
```

לאחר שהתקבלה פקטת הFIN הוא ישלח עליה ACK.

```c
// wait a little to ensure no FIN retransmission
while (rudp_rcv_with_timer(sockfd, rudp_packet: &last_packet, src_addr: &receiver_addr, addrlen: &len, timeout: TIMEOUT * 2)) {
    if (!(last_packet.flags & FLAG_FIN)) {
        perror(s: "last packet received needs to be FIN");
        return -1;
    }

    //printf("Received FIN %d\n",last_packet.seq_num);
    // ACK the FIN
    last_packet.flags = FLAG_ACK;
    if (rudp_send(sockfd, rudp_packet: &last_packet, serv_addr: &receiver_addr, addrlen: sizeof(receiver_addr)) < 0) {
        perror(s: "sendto failed");
        return -1;
    }
}
return 0;
```

כעת נחכה לשידורים חוזרים של FIN על מנת לראות האם הACK התקבל בהצלחה. במידה ונקבל שידור חוזר של FIN נשלח שוב ACK עד שלא נקבל יותר שידורים חוזרים.

```c
unsigned short int calculate_checksum(void *data, unsigned int bytes) {
    unsigned short int *data_pointer = (unsigned short int *) data;
    unsigned int total_sum = 0;
    // Main summing loop
    while (bytes > 1) {
        total_sum += *data_pointer++;
        bytes -= 2;
    }
    // Add left-over byte, if any
    if (bytes > 0)
        total_sum += *((unsigned char *) data_pointer);
    // Fold 32-bit sum to 16 bits
    while (total_sum >> 16)
        total_sum = (total_sum & 0xFFFF) + (total_sum >> 16);
    return (~((unsigned short int) total_sum));
}
```

פונקציה לחישוב checksum הלקוחה מ- appendix D.

RUDP_Sender.c

```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(nmemb: size, size: sizeof(char));
    // Error checking.
    if (buffer == NULL)
        return NULL;
    // Randomize the seed of the random number generator.
    srand(seed: time(timer: NULL));
    for (unsigned int i = 0; i < size; i++)
        *(buffer + i) = ((unsigned int) rand() % 256);
    return buffer;
}
```

פונקציה שלקוחה מ- Appendix C ומייצרת מידע רנדומלי בגודל נתון.

```c
int main(int argc, char *argv[]) {


    struct timeval start, end;
    puts("Starting Sender...\n");

    if (argc != 5) {
        puts("invalid command");
        return 1;
    }
    int port = 0;
    char SERVER_IP[20] = {0};
    for (int i = 1; i < argc; i++) {
        if (strcmp(argv[i], "-p") == 0) {
            port = atoi(argv[i + 1]);
        } else if (strcmp(argv[i], "-ip") == 0) {
            strcpy(SERVER_IP, argv[i + 1]);
        }
    }
```

ניצור את הקובץ הרנדומלי ונקבל את הערכים משורת הפקודה ונשמור אותם כמשתנים.

```c
// Create socket
int sockfd = rudp_socket();
if (sockfd < 0) {
    perror("Socket creation error");
    exit(EXIT_FAILURE);
}
```

נאתחל את הסוקט

```
// Set up receiver address
struct sockaddr_in receiver_addr;
//socklen_t rec_len = sizeof(receiver_addr);
memset(s:&receiver_addr, c:0, n:sizeof(receiver_addr));
receiver_addr.sin_family = AF_INET;
receiver_addr.sin_port = htons(hostshort:port);

if (inet_pton(af:AF_INET, cp:SERVER_IP, buf:&receiver_addr.sin_addr) <= 0) { // setting the SERVER_IP address
    perror(s:"inet_pton(3)");
    close(fd:sockfd);
    return 1;
}
```

נאתחל את המבנה בו נשמור את הכתובת של המקבל. נשלח לפורט שהוכנס ולכתובת ה- IP שהוכנסה.

```
char *file = util_generate_random_data(size:FILE_SIZE);
int status = 1;

while (status == 1) {
```

נאתחל את הקובץ אותו נשלח ואת מצב השליחה. כל עוד מצב השליחה הוא 1 נחזור על הצעדים הבאים:

```
// Perform handshake
int seqnum = rudp_connect(sockfd, dest_addr:&receiver_addr, addrlen:sizeof(receiver_addr), side:0);
if (seqnum == -1) {
    perror(s:"connection failed");
    return -1;
} else if (seqnum == -2) {
    perror(s:"Server can't receive packets.");
    break;
}
puts(s:"Connected successfully");
```

נשלח בקשה לחיבור למקבל. אם נקבל יותר מדי timeout-ים נבין שהמקבל לא מסוגל לקבל הודעות ונסגור את התוכנית.

```
gettimeofday(tv:&start,tz:NULL);
// Send data
if (rudp_send_file(file, sockfd, receiver_addr, seqnum) < 0) {
    perror(s:"send file failed");
    return -1;
}
gettimeofday(tv:&end,tz:NULL);
double timeTaken = (end.tv_sec - start.tv_sec) * 1000.0;
timeTaken += (end.tv_usec - start.tv_usec) / 1000.0;
printf(format:"time taken: %0.3lf ms\n",timeTaken);
```

נשמור את הזמן, נשלח קובץ ונשמור את הזמן לאחר מכן. כך נקבל את כמות הזמן שהשליחה לקחה. נדפיס את ערך זה.

```
puts(s:"Finished sending data. Closing connection");
//finish sending, and receive the last ack, close the rudp connection
//packet->flags = FLAG_FIN;
if (rudp_close(sockfd, dest_addr:&receiver_addr, addrlen:sizeof(receiver_addr), side:0) == -1) {
    perror(s:"close failed");

    return -1;
}
```

לבסוף נסגור את הקשר.

```
puts(s:"would you like to send the file again? (0 to exit, 1 to resend)");
scanf(format:"%d", &status);
```

נבקש מהמשתמש האם הוא רוצה לשלוח את הקובץ שוב. אם כן נחזור על הצעדים האחרונים ואם לא נצא מהלולאה.

RUDP_Receiver.c

```c
void printStats(double times[100], int run) {
    double timeSum = 0;
    double speedSum = 0;
    double avgTime = 0;
    double avgSpeed = 0;
    puts("-------------------------------------------");
    puts("(*) Times and speeds summary:\n");
    for (int i = 1; i < run; i++) {
        timeSum += times[i];
        double speed  = (FILE_SIZE / times[i]) / 1000;
        speedSum += speed;
        printf("(*) Run %d, Time: %0.3lf ms, Speed: %0.3lf MB/s\n", i, times[i], speed);
    }
    if (run > 0) {
        avgTime = (timeSum / (double) (run-1));
        avgSpeed = (speedSum / (double) (run-1));
    }
    printf("\n(*) Overall statistics:\n");
    printf("(*) Number of runs: %d\n", run-1);
    printf("(*) Average RTT: %0.3lf ms\n", avgTime);
    printf("(*) Average throughput: %0.3lf MB/s\n", avgSpeed);
    printf("(*) Total time: %0.3lf ms\n", timeSum);
    printf("-------------------------------------------\n");

}
```

פונקציה שמדפיסה כמה זמן לקח לכל הודעה להגיע בשלומתה ואת מהירות השליחה הכוללת שלה. לבסוף היא מדפיסה את ממוצע הזמנים וממוצע המהירויות (רוחב הפס).

```
double times[MAX_RUNS] = {0};
int currentRun = 1;

char file[FILE_SIZE] = {0};
puts("Starting Receiver...\n");
if (argc != 3)
{
    puts("invalid command");
    return 1;
}
int port = atoi(argv[2]);
```

נקבל את הערכים משורת הפקודה ונשמור אותם כמשתנים.

```
int sockfd;
struct sockaddr_in receiver_addr;
memset(&receiver_addr, 0, sizeof(receiver_addr));

// Create socket
sockfd = rudp_socket();
if (sockfd < 0)
{
    perror("Socket creation error");
    exit(EXIT_FAILURE);
}

// Bind socket to port
memset(&receiver_addr, 0, sizeof(receiver_addr));
receiver_addr.sin_family = AF_INET;
receiver_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
receiver_addr.sin_port = htons(port);
```

נאתחל את הסוקט ואת המבנה ששומר את כתובת המקבל. נעבוד עם IPv4 והכתובת של המקבל תהיה 127.0.0.1.

```
if (bind(fd:sockfd, addr:(struct sockaddr *)&receiver_addr, len:sizeof(receiver_addr)) < 0)
{
    perror(s:"Bind failed");
    close(fd:sockfd);
    exit(status:EXIT_FAILURE);
}

puts(s:"Waiting for RUDP connection...");
```

נקשור את הסוקט לפורט ולכתובת שקיבלנו ואתחלנו.

```
struct sockaddr_in sender_addr;
socklen_t sender_addr_len = sizeof(sender_addr);
memset(s:&sender_addr, c:0, n:sender_addr_len);

while (1) {
    // Wait for handshake
    int seqnum = rudp_connect(sockfd, dest_addr:&sender_addr, addrlen:sizeof(sender_addr), side:1);
    if (seqnum == -1) {
        perror(s:"connection failed");
        close(fd:sockfd);
        return -1;
    } else if (seqnum == -2) {
        break;
    }
    printf(format:"Connected successfully. seqnum: %d\n", seqnum);
```

נאתחל את המבנה ששומר את הכתובת של השולח ונכנס ללולאה. נחכה לקשר ואם לא קיבלנו תוך כמות הזמן שהגדרנו הבין שאין ניסיון ליצירת קשר ונסיים. אחרת נאשר את הקשר.

```
if (rudp_rcv_file(file, sockfd, receiver_addr, seqnum,time:times,run:&currentRun) < 0) {
    perror(s:"receiving failed");
    return -1;
}

printf(format:"Finished receiving\n");

if (rudp_close(sockfd, dest_addr:&sender_addr, addrlen:sizeof(sender_addr), side:1) == -1) {
    perror(s:"close failed");
    return -1;
}
```

לאחר מכן נקבל את הקובץ ונסגור את הקשר.

```
printStats(times, run: currentRun);

close(fd: sockfd);
return 0;
```

לבסוף נדפיס את הסטטיסטיקות.

Part C – Research

כל התמונות המידע וההקלטות נמצאות ומסודרות בתיקייה של DATASET המצורפת.
בכל הסנפה ניתן לראות כי אנו פותחים קשר באמצעות לחיצת יד משולשת, וכמו כן, סוגרים באמצעות FIN
FIN -> FINACK … FINACK→. בין לבין נמצאות הפקטות שנשלחות ומעבירות את המידע עצמו.
נשים לב כי בRUDP ישנן הרבה יותר פקטות מכיוון שהגבלנו את גודל המידע של הפקטה ל 2048.
נציין כי הDATASET שלנו גדול יותר ומכיל את הקומבינציות בשביל הבונוס.

1)

בTCP, כאשר אחוזי האיבוד היו נמוכים,לא נראה הבדל משמעותי בין שימוש בTCP-RENO, לTCP-CUBIC. לעומת זאת, ככל שעלו אחוזי האיבוד ניתן היה לראות ששימוש בTCP-CUBIC משפר משמעותית את מהירות השליחה, ואת הRTT הממוצע.
את המידע הנ"ל הסקנו מתוך הDATASET שיצרנו, ע"י כך שעברנו על כל אחוז איבוד, וראינו מה הRTT הממוצע, ומה המהירות הממוצעת.

2)

מכיוון שמימשנו את RUDP ע"י פרוטוקל stop&wait, וללא בקרת זרימה ובקרת גודש, הוא משמעותית יותר איטי מהTCP, בכל אחוזי האיבוד, וללא תלות באלגוריתם בקרת הגודש שהשתמשנו. ניתן לראות את זה ע"י ההבדלים בזמן הממוצע ובמהירות הממוצעת, בין הTCP לבין הRUDP שבDATASET שלנו.

3)

אם נתייחס למידע שאספנו, לא נרצה להשתמש בRUDP בשום מצב, ונעדיף את TCP, זאת בגלל המימוש הפשוט שבחרנו, שבא על חשבון מהירות.
אם נאמר באופן כללי, ולא בהתחשב במידע שלנו, בדרך כלל נרצה להשתמש בRUDP ביישומים בהן נדרש העברת מידע בצורה אמינה בזמן אמת, וביישומים שרוצים לשלוט באמינות העברת המידע, כגון משחקי מחשב מקוונים, בעוד שבTCP נשתמש כשנצטרך להעביר מידע בצורה אמינה, אך בלי צורך שיהיה בזמן אמת, ואם חצי שנייה אחרי זה בסדר, כגון mail, file transfers, browsing…

שאלות בעברית:

שאלה 1

כאשר אנו במצב slow start, בכל פעם נשלח מספר פקטות, נחכה ל-ACK עליהן ואז נגדיל את חלון השליחה ונשלח יותר פקטות עד שנגיע ל-SSThreshold. במידה ונגדיל את SSThreshold, יישלחו יותר פקטות לפני שנעבור למצב congestion avoidance. כלומר, נשלח הרבה יותר פקטות מבלי לחכות ל- ACK. אמנם נוכל לשלוח הרבה מידע מהר מאוד, אבל אין לנו וודאות על האם הוא הגיע ובנוסף אנו עלולים ליצור עומס ברשת. נבחן את המקרים ונראה מתי עדיף לנו להגדיל את SSThreshold.
כאשר הרשת אמינה, כלומר מעט פקטות נאבדות, הגדלת SSThreshold תועיל לנו, מאחר וגם אם נשלח הרבה פקטות לפני שמחכים לאישורים עליהם, סביר שהן כן יגיעו לנמען.
כאשר הקשר ארוך, כלומר יש הרבה מידע לשלוח בקשר, הגדלת SSThreshold תועיל לנו. זאת מאחר וככל שיש לנו יותר מידע, כך גודל החלון יותר משפיע מכיוון ונצטרך לשלוח את ההודעות בחלון זה יותר פעמים. בנוסף, כאשר יש מעט מידע, יכול להיות שאפילו לא נגיע ל SSThreshold, אלא נסיים לשלוח את המידע לפני. לעומת זאת כאשר יש הרבה מידע, נרצה להישאר ב slow start יותר זמן ובכך להעביר הרבה יותר מידע בזמן קצר יותר.
כאשר ה RTT גדול, יקח הרבה זמן ל-ACKים לחזור. לכן הגדלת SSThreshold תועיל לנו כי נוכל לשלוח יותר פקטות מבלי להמתין ל ACK, מה שייקח אפילו יותר זמן במקרה זה. בנוסף, מכיוון שלוקח יותר זמן ל-ACKים לחזור ברשת עם RTT גדול, הגדלת SSTreshold מאפשרת לשולח להשאיר את הצינור מלא בנתונים, מה שמוביל לניצול יעיל יותר של רוחב הפס.
כמו שראינו, נעדיף להגדיל את SSThreshold במקרים של: רשת אמינה, קשר ארוך או RTT גדול.
לכן הגדלת SSThreshold תועיל במידה המירבית ב<u>מקרה 1: בקשר ארוך על גבי רשת אמינה עם RTT גדול.</u>

שאלה 2

נתאר את הקשר בין A ל-B. לאחר פתיחת הקשר שהנחנו שמתנהלת באופן תקין, מתחילים בתהליך slow start. במהלך slow start, החלון מתחיל בגודל 1MSS, ולאחר שהגיע ACK מגדיל את גודל החלון פי 2 עד שמגיעים ל-SSThreshold, במקרה שלנו - S*MSS, או שנקבל timeout או dupilcate ACK. על פי הנתונים לא היה אובדן חבילות וגם rwnd < S*MSS, כלומר כמות המידע ש-B יכל לקבל תמיד הייתה יותר ממה ש-A שלח ולכן B קיבל את כל החבילות ש-A שלח.
לכן, כל החבילות תמיד יגיעו ויתקבלו בהצלחה ולכן גודל החלון יעלה פי 2 כל פעם, כלומר כל RTT (כי זה הזמן שלוקח למידע להגיע מ-A ל-B ואז לACK להגיע מ-B ל-A), ולא נצא מslow start לפני שנגיע ל-SSThreshold.
מכך נובע כי כמות ה-RTT שלוקח לקשר היא $log_2(s)$. לכן ה<u>תשובה הנכונה היא 1</u>:

שזה בערך יוצא: $\frac{S*MSS}{log_2(s)*RTT*2}$.

שאלה 3

ספרת הביקורת הנמוכה יותר מהת.ז-ים שלנו היא 0, אבל מאחר והשאלה לא הגיונית עבור X=0, נחשב עבור <u>X=5</u>, ספרת הביקורת של הת.ז השני.

נתחיל מלחשב את RTT. הערך RTT הוא כמות הזמן שלוקח לפקטה להישלח ולחזור. במקרה שלנו נתון שקצב ההתפשטות הוא $2 \cdot 10^8$ m/sec, והמרחק בין התחנות הוא 1 KM, כלומר 1,000 מטר. לכן זמן העיכוב (Propagation delay) הוא:

$\frac{1,000}{2 \cdot 10^8} = 0.5 \cdot 10^{-5}$ שניות.

זמן השידור (trasmission time) הוא: $\frac{5 \cdot 8 \cdot 10^3}{8 \cdot 10^9} = \frac{5}{10^6}$ שניות.

לכן, סה"כ RTT הוא: $2 \cdot (0.5 \cdot 10^{-5} + \frac{5}{10^6}) = 0.00002.$ שניות.

מאחר ואין אובדן פקטות בכלל, כל הפקטות יגיעו והACK-ים יחזרו ללא שידורים חוזרים. לכן אנו יכולים למקסם את גודל החלון כדי לנצל את רוחב הפס המלא מבלי לדאוג לשידורים חוזרים. גודל החלון, כלומר המספר המרבי של פקטות הניתן לשלוח במהלך RTT אחד, הוא:

$$window\ size = \frac{communication\ rate \cdot RTT}{packet\ size} = \frac{8 \cdot 10^9 \cdot 2 \cdot (0.5 \cdot 10^{-5} + \frac{5}{10^6})}{5 \cdot 8 \cdot 10^3} = 4$$