בס"ד

רשתות תקשורת מטלה 3 | ת.ז 325408409

! ההקלטות מצורפות בקובץ הzip. שם ההקלטות של tcp מהצורה
tcp_**s**<cubic/reno>_**r**<cubic/reno>_<loss%>.pcapng : לדוגמה tcp_scubic_rreno_2 זה
tcp, **s** = sender משתמש בcubic, **r** = receiver משתמש בreno, וההקלטה היא של 2% איבוד.

חלק A – TCP:

דוגמת הרצה:

```
ayelet@ayelet-VirtualBox:~/Documents/network_communications_3$ ./TCP_Sender -ip 127.0.0.1 -p 12345 -algo reno
Sender Connected To Receiver...
Data sent.
Do you wish to resend the data? (1 for yes, 0 for no): 1
Data sent.
Do you wish to resend the data? (1 for yes, 0 for no): 0
ayelet@ayelet-VirtualBox:~/Documents/network_communications_3$
```

```
ayelet@ayelet-VirtualBox:~/Documents/network_communications_3$ ./TCP_Receiver -p 12345 -algo reno
Starting Receiver...
Waiting for TCP connection...
TCP connection established
Received 32741 bytes
Received 65482 bytes
Received 130964 bytes
Received 196446 bytes
Received 261928 bytes
Received 491115 bytes
Received 523856 bytes
Received 589339 bytes
Received 654822 bytes
Received 720305 bytes
Received 785788 bytes
Received 851271 bytes
Received 916754 bytes
Received 982237 bytes
Received 1047720 bytes
Received 1113203 bytes
Received 1178686 bytes
Received 1244169 bytes
Received 1309652 bytes
Received 1375135 bytes
Received 1440618 bytes
Received 1506101 bytes
Received 1571584 bytes
Received 1637067 bytes
Received 1702550 bytes
Received 1768033 bytes
Received 1833516 bytes
Received 1898999 bytes
Received 1964482 bytes
Received 2029965 bytes
Received 2095448 bytes
Received 2097153 bytes
Received 4194306 bytes
Received 4 bytes
Data received and written
Total received: 4194310 bytes
Time taken: 8.42 seconds
Average Bandwidth: 486.57 KB/s
--------------------------------
Average Time: 8.42 seconds
Average Bandwidth: 486.57 KB/s
ayelet@ayelet-VirtualBox:~/Documents/network_communications_3$
```

בס"ד

הסבר מימוש קוד – TCP_Sender.c :

```c
#include <netinet/tcp.h>
#include <netdb.h>
#include <time.h>

//#define MAX_SIZE 1024

// 2MB = 2^21 bytes = 2097152 bytes. Therefore buffer
// is rounded up to be more then 2MB as required
#define FILE_BUFFER_SIZE 2097153

  /* * @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. */
 // CONTAINS CALLOC! REMEMBER TO FREE
char* util_generate_random_data(unsigned int size) {
    char *buffer = NULL;
    if (size == 0) return NULL; // Argument check.
    buffer = (char *)calloc(size, sizeof(char));
    if (buffer == NULL) return NULL;  // Error checking.
    srand(time(NULL)); // Randomize the seed of the random number generator.
    for (unsigned int i = 0; i < size; i++) {*(buffer + i) = ((unsigned int)rand() % 256);}
    return buffer;
}
```

הגדרת באפר בגודל בדיוק מעבר ל2MB, ופונקציית יצירת קובץ עם תוכן רנדומי בגודל רצוי שניתנה כנספח במטלה.

```c
int main(int argc, char *argv[]) {
    if (argc < 7) {
        fprintf(stderr, "Usage: %s -ip <IP> -p <PORT> -algo <ALGO>\n", argv[0]);
        return 1;
    }
    const char* IP = argv[2];// CHECK NUMBERS CORECTNESS // added atoi
    const int PORT = atoi(argv[4]);
    const char* ALGO = argv[6];

    if (PORT <= 0 || PORT > 65535) {
        fprintf(stderr, "Invalid port number: %d\n", PORT);
        return 1;
    }
```

main של השולח (sender) – קודם בדיקה ששלחנו כמות ארגומנטים נכונה ואחר כך שמירת הארגומנטים (פורט, IP, אלגוריתם בקרה וכו) במשתנים. בדיקה שהפורט שהוזן לגיטימי.

```c
int main(int argc, char *argv[]) {
    int soc = socket(AF_INET, SOCK_STREAM, 0);
    if (soc < 0){
        perror("error opening socket");
        return 1;
    }
    struct sockaddr_in receiver;
    memset(&receiver, 0, sizeof(receiver));
    receiver.sin_family = AF_INET;
    receiver.sin_port = htons(PORT);
    if (inet_pton(AF_INET, IP, &receiver.sin_addr) <0){//IP=argv[2]. here: convert ip adress from text to binary form
        close(soc);
        return 1;
    }
    if (strcmp(ALGO, "reno") != 0 && strcmp(ALGO, "cubic") != 0) {
        close(soc);
        return 1;
    }
    if (setsockopt(soc, IPPROTO_TCP, TCP_CONGESTION, ALGO, strlen(ALGO)) < 0) {
        close(soc);
        return 1;
    }
    if (connect(soc, (struct sockaddr*)&receiver, sizeof(receiver)) < 0){ // establishes TCP connection to the receiver
        close(soc);
        return 1;
    }

    printf("Sender Connected To Receiver...\n");
```

יצירת סוקט ובדיקת תקינות, יצירת כתובת אינטרנט סוקט לקוח שקיבלנו בארגומנטים, המרת כתובת IP לבינארי, בדיקת תקינות שקיבלנו שם אלגוריתם תקין, והגדרה לסוקט שיהיה מוגדר להיות בגרסת

בס"ד

IP 4 וסטרים, לTCP, עם congestion control אלגוריתם – הסוג שקיבלנו. לבסוף connect עושה initialization לחיבור הסוקט עם כתובת הreceiver.

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

    char* data = util_generate_random_data(FILE_BUFFER_SIZE); // inner calloc! remmeber to free
    if (data == NULL) {
        perror("error generating data");
        close(soc);
        return 1;
    }

    int resend = 1;
    do {
        int bytes_sent = send(soc, data, FILE_BUFFER_SIZE, 0);
        if (bytes_sent < 0) {
            perror("error sending data");
            close(soc);
            return 1;
        }
        printf("Data sent.\n");
        printf("Do you wish to resend the data? (1 for yes, 0 for no): ");
        scanf("%d", &resend);
    } while (resend == 1);

    if (send(soc, "EXIT", strlen("EXIT"), 0) < 0) {//check strlen for bugs here because of ""
        perror("error sending exit message");
        close(soc);
        return 1;
    }

    free(data);
    close(soc);
    return 0;

}
```

ואחרי סידור החיבור מג'נרטים קובץ בגודל הבאפר (קצת יותר מMB2), בודקים שבאמת הוחזר מהפונקציה פוינטר לקובץ (ולא פוינטר "ריק" = לנאל), שולחים את הקובץ, אם המשתמש מכניס לטרמינל 1, שהוא רוצה לשלוח שוב הלולאה ממשיכה, עד שהוא יגיד לא (מכניסים 0 לresend ובבדיקה בסוף ה-do while יתקיים תנאי יציאה), כשמסיימים לשלוח שולחים הודעה EXIT, מעין FIN בלי ACK.

הסבר מימוש קוד – TCP_Receiver.c :

```c
C TCP_Receiver.c > main(int, char * [])

void print_stats(struct timeval start, struct timeval end, int totalReceived);
int main(int argc, char *argv[]) {
    if (argc < 5) {
        fprintf(stderr, "Usage: %s -p <PORT> -algo <ALGO>\n", argv[0]);
        return 1;
    }

    const int PORT = atoi(argv[2]); // from string to int
    const char* ALGO = argv[4];

    if (PORT <= 0 || PORT > 65535) {
        fprintf(stderr, "Invalid port number: %d\n", PORT);
        return 1;
    }
```

הצהרת פונקציה, בדיקת ושמירת נתונים מהארגומנטים, בדיקת לוגיות כתובת פורט

```c
int soc = socket(AF_INET, SOCK_STREAM, 0);
if (soc < 0){
    return 1;
}
struct sockaddr_in server; // internet socket adress struct has three fields: sin_family = ipv4 or ipv6,
//sin_port - port numbrt, sin_addr - internet ip address
memset(&server, 0, sizeof(server)); // first fill in memory with 0
server.sin_family = AF_INET;
server.sin_port = htons(PORT); // port we want to connect to (16-bit int convert to network byte order)
if (inet_pton(AF_INET, argv[2], &server.sin_addr) <0){ // convert target ipv4 address to binary form
    close(soc);
    return 1;
}

if(bind(soc, (struct sockaddr*)&server, sizeof(server)) < 0){ // bind: give the socket soc the adress of this server
    close(soc);
    return 1;
}
// prepare to accept incoming connections on the socket soc, 1 - max number of waiting connections queued
if (listen(soc, 1) < 0){
    close(soc);
    return 1;
}
printf("Starting Receiver...\n");
printf("Waiting for TCP connection...\n");
struct sockaddr_in client; //(client = sender)
memset(&client, 0, sizeof(client)); // zero filling in case memory was used before
socklen_t client_addr_len = sizeof(struct sockaddr_in);
//Await a connection on socket soc. When it arrives, open new socket to communicate with it,
//set *CLIENT-ADDRESS to the address of the connecting peer and return the new socket's descriptor.
int client_sender_soc = accept(soc, (struct sockaddr*)&client, &client_addr_len);
if (client_sender_soc < 0){
    close(soc);
    return 1;
}
printf("TCP connection established\n");
```

בס"ד

פתיחת סוקט ובדיקה, עם הכתובת בipV4, המרה לצורה בינארית של IP, מחברים ומטפלים בזיכרון לסטרים סוקט. בaccept מחזירים סוקט לתקשורת עם השולח שם נקבל ממנו את הקובץ כשישלח.

הגדרת תקינות שם אלגוריתם וויסות, והגדרתו לסוקט.

```
if (strcmp(ALGO, "reno") != 0 && strcmp(ALGO, "cubic") != 0) {
    perror("Invalid algorithm\n");
    close(soc);
    close(client_sender_soc);
    return 1;
}
// Set TCP congestion control algorithm
if (setsockopt(soc, IPPROTO_TCP, TCP_CONGESTION, ALGO, strlen(ALGO)) < 0) {
    perror("Setting TCP congestion control algorithm failed");
    close(soc);
    close(client_sender_soc);
    return 1;
}
```

הכנות לספור זמן וסטטיסטיקות של כמה בתים התקבלו ונכתבו לקובץ. פותחים קובץ – לתוכו נכתוב את הקובץ שקיבלנו. בודקים אם צריך לפתוח באפר אקסטרה להשלים את השאר.

```
int totalReceived = 0;
int counter = 1;
struct timeval start, end;
char filename[40];
sprintf(filename, "received%d.txt", counter);
FILE *file = fopen(filename, "w");

if (file == NULL) {
    perror("Error opening file");
    close(client_sender_soc);
    close(soc);
    return 1;
}

char buffer[BUFFER_SIZE];
int bytesRead;
int onerecv = 0;
int currentsize = BUFFER_SIZE;
if(BUFFER_SIZE % MAX_SIZE != 0){
    currentsize = BUFFER_SIZE+MAX_SIZE;
}
gettimeofday(&start, NULL);
```

בלולאה עד שהשולח מסיים לשלוח הכל מקבלים את הנתונים, וכותבים אותם לקובץ received. אם בוחרים לשולח יותר מקובץ אחד, בלולאה יוצרים קבצי המשך אם צריך. בסוף סוגרים את הקובץ והסוקט. מודדים זמן סיום. שולחים נתונים זמנים ובתים לפונקציה מודדת סטטיסטיקות. סוגרים סוקטים.

```
int main(int argc, char *argv[]) {
    while ((bytesRead = recv(client_sender_soc, buffer, sizeof(buffer), 0)) > 0) {
        if (onerecv > currentsize - MAX_SIZE){
            counter++;
            sprintf(filename, "received%d.txt", counter);
            fclose(file);
            onerecv = 0;

        }
        if (onerecv == 0 && bytesRead>0){
            file = fopen(filename, "w");
            if (file == NULL) {
                perror("Error opening file");
                close(client_sender_soc);
                close(soc);
                return 1;
            }
        }
        fwrite(buffer, sizeof(char), bytesRead, file);
        totalReceived += bytesRead;
        onerecv += bytesRead;
        printf("Received %d bytes\n", onerecv);

    }

    if (bytesRead < 0) {
        perror("Receive failed");
    }
    printf("Data received and written\n");

    printf("Total received: %d bytes\n", totalReceived);
    fclose(file);

    gettimeofday(&end, NULL);
    print_stats(start, end, totalReceived);
    close(client_sender_soc);
    close(soc);

    return 0;
}
```

בס"ד

פונקציית חישוב הסטטיסטיקות וההדפסה שלהן לטרמינל.

```c
void print_stats(struct timeval start, struct timeval end, int totalReceived) {
    double time_taken = (end.tv_sec - start.tv_sec) + (end.tv_usec - start.tv_usec) / 1e6;
    double bandwidth = (totalReceived / 1024.0) / time_taken;
    static double avgBandwidth = 0;
    avgBandwidth += bandwidth;
    static double avgTime = 0;
    avgTime += time_taken;
    static int counter = 0;
    printf("Time taken: %.2f seconds\n", time_taken);
    printf("Average Bandwidth: %.2f KB/s\n", bandwidth);
    printf("--------------------------------\n");
    printf("Average Time: %.2f seconds\n", avgTime / ++counter);
    printf("Average Bandwidth: %.2f KB/s\n", avgBandwidth / counter);
}
```

חלק B – RUDP:

דוגמת הרצה:

```
Packet checksum: 2050
Acknowledge sent.
Packet received.
Packet checksum: 2051
Acknowledge sent.
Received data
Current Time taken: 0.26 seconds
Current Average Bandwidth: 8018.95 KB/s
--------------------------------
Average Time: 0.26 seconds
Average Bandwidth: 8018.95 KB/s
```

RECEIVER

```
Acknowledge received.
Packet sent.
Acknowledge received.
Packet sent.
Acknowledge received.
Packet sent.
Acknowledge received.
Packet sent.
Acknowledge received.
Packet sent.
Acknowledge received.
Packet sent.
Acknowledge received.
Packet sent.
Acknowledge received.
Do you want to send more data? (y/n)
n
ayelet@ayelet-VirtualBox:~/Documents/network_communications_3$
```

SENDER

<u>הסבר על הקוד:</u>

החלק העיקרי נכתב ב <u>RUDP_API.c</u> לכן קודם אראה אותו

```c
#define MAX_PAYLOAD_SIZE 1024
// 2MB = 2^21 bytes = 2097152 bytes. therefore buffer is rounded up to be more then 2MB as required
#define FILESIZE 2100000
#define TIMEOUT_SEC 2
#define MAX_RETRIES 3

// different packets flags defining:
#define REGFLAG 0x01
#define SYNFLAG 0x02
#define ACKFLAG 0x03
#define FIN 0x04


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;
}
```

בס"ד

הגדרת גודל מקסימלי של דאטה בפקטה וגודל קובץ שבחרנו (גדול מ2MB.)

timeout הוא הגבלת זמן שהsender יחכה לACK עד שישלח שוב, ומקסימום כמות נסיונות חוזרים. הפונקציה היא של ג'ינרות הקובץ בגודל מבוקש שצורף במטלה.

סטראקט למבנה כותרת של חבילת RUDP. מגדירים אורך, מחשבים צ'קסם כדי לבדוק אמינות, דגלים רלוונטים ואחר כך תוכן הדאטה. יהיה שימוש

```
struct RUDP_Header {
    uint16_t length;  // Length of the packet
    uint16_t checksum; // Checksum for error detection
    uint8_t flags;    // Flags for various control information
    char value[MAX_PAYLOAD_SIZE];      // Data payload
};
```

Reliable UDP Header proposed sketch			
Byte 0	Byte 1	Byte 2	Byte 3
Length (2 Bytes)		Checksum (2 Bytes)	
Flags (1 Byte)			

Where:

- ***Length*** is the length of the data itself, without the RUDP header.
- ***Checksum*** is a 16bit number that validates the correctness of the data.
- ***Flags*** is a special byte where we classify the packet itself (SYN, ACK, etc.).

בדומה להצעה במטלה <-

```
// Function to build an RUDP packet with data
void packetConstruct(struct RUDP_Header *header, char *data, uint16_t dataLength, uint16_t checksum, uint8_t flags) {
    header->length = htons(sizeof(struct RUDP_Header)); // Total length of packet
    header->checksum = htons(checksum); // Convert checksum to network byte order
    header->flags = flags; // 8 bit single byte no need for htons conversion
    memcpy(header->value, data, dataLength);
}

int send_receiveAck(int sockfd, struct sockaddr_in *addr, int isSend) {
    struct RUDP_Header ackHeader;

    if (isSend) {
        packetConstruct(&ackHeader, "ACK", sizeof("ACK"), 0, ACKFLAG);
        if (sendto(sockfd, &ackHeader, sizeof(ackHeader), 0, (const struct sockaddr *)addr, sizeof(*addr)) < 0) {
            perror("sendto failed");
            return -1;
        }
        printf("Acknowledge sent.\n");
    } else {
        struct timeval timeout;
        timeout.tv_sec = TIMEOUT_SEC;
        timeout.tv_usec = 0;
        setsockopt(sockfd, SOL_SOCKET, SO_RCVTIMEO, (const char *)&timeout, sizeof(timeout));

        char ackBuffer[sizeof(struct RUDP_Header)];
        socklen_t addrLen = sizeof(*addr);
        int numBytesReceived = recvfrom(sockfd, ackBuffer, sizeof(ackBuffer), 0, (struct sockaddr *)addr, &addrLen);
        if (numBytesReceived < 0) {
            printf("Acknowledge not received. Retransmitting...\n");
            return 0; // Acknowledge not received
        }
        printf("Acknowledge received.\n");
    }

    return 1; // Acknowledge sent or received successfully
}
```

שתי פונקציות – הראשונה מעיין פונקציית בנאי שמקבלת פרמטים נחוצים ופוינטר לסטרקט הפקטה, וממלאת אותו. פוקציה נטו לנוחות כדי לא להכניס את זה בקוד כל פעם.

פונקציה שניה לשליחת וקבלת אישורים – יש דגל isSend להבדיל בין שליחה וקבלה. כששולחים פשוט בונים פקטה עם הודעת ACK ושולחים עם הסוקט והכתובת שקיבלנו בפונקציה לצד השני. כשמקבלים הופכים את הסוקט ללא תלוי פרוטוקול, ומגדירים לו את הטיימאאוט. מקבלים מהכתובת שהועברה בפונקציה.

בס"ד

```c
int performHandshake(int sockfd, struct sockaddr_in *serverAddr) {
    struct RUDP_Header handshakePacket;

    // Send handshake packet to the receiver
    packetConstruct(&handshakePacket, "SYN", sizeof("SYN"), 0, SYNFLAG);
    if (sendto(sockfd, &handshakePacket, ntohs(handshakePacket.length), 0, (const struct sockaddr *)serverAddr, sizeof(*serverAddr)) < 0) {
        perror("sendto failed");
        return -1; // Error sending handshake packet
    }
    printf("Handshake packet sent.\n");

    // Wait for acknowledgment from the receiver
    if (!send_receiveAck(sockfd, serverAddr, 0)) {
        printf("Acknowledgment for handshake not received. Handshake failed.\n");
        return 0; // Handshake failed
    }

    send_receiveAck(sockfd, serverAddr,1);//sendind ack

    printf("Handshake successful.\n");
    return 1; // Handshake successful
}
```

פונקציית אתחול לחיצת ידיים בשביל הsender, מתחיל לחיצת ידיים ומחכה לאישור מהreceiver.

```c
int receiveHandshake(int sockfd, struct sockaddr_in *clientAddr) {
    struct RUDP_Header handshakePacket;

    // Wait for handshake packet from the sender
    socklen_t clientAddrLen = sizeof(*clientAddr);
    int numBytesReceived = recvfrom(sockfd, &handshakePacket, sizeof(handshakePacket), 0, (struct sockaddr *)clientAddr, &clientAddrLen);
    if (numBytesReceived < 0) {
        perror("recvfrom failed");
        return -1; // Error receiving handshake packet
    }

    // Send acknowledgment back to the sender
    send_receiveAck(sockfd, clientAddr,1); // sending ack

    printf("Handshake packet received and acknowledged.\n");
    printf("Waiting for acknowledgment...\n");
    if(!send_receiveAck(sockfd, clientAddr,0)) {
        printf("Acknowledgment for handshake not received. Handshake failed.\n");
        return 0; // Handshake failed
    }
    return 1; // Handshake successful
}
```

פונקציה שהרסיבר משתמש בה, היא מקבלת את הפאקטה של לחיצת היד ומחזירה אישור ואז מחכה לאישור של השולח שהוא קיבל את האישור.

```c
int rudp_socket() {
    int sockfd;

    // Create UDP socket
    if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
        perror("socket creation failed");
        exit(EXIT_FAILURE);
    }
    return sockfd;
}
void rudp_close(int sockfd) {
    close(sockfd);
}
```

שתי פונקציות של פתיחת סוקט, והשניה סגירה לשו. בפתיחה זאת פתיחת סוקט סטנדרטי של IPv4, וUDP.

```c
void rudp_send(int sockfd, struct sockaddr_in *serverAddr, struct RUDP_Header *packetHeader) {
    int retries = 0;
    while (retries < MAX_RETRIES) {
        if (sendto(sockfd, packetHeader, ntohs(packetHeader->length), 0, (const struct sockaddr *)serverAddr, sizeof(*serverAddr)) < 0) {
            perror("sendto failed");
            exit(EXIT_FAILURE);
        }
        printf("Packet sent.\n");
        if (send_receiveAck(sockfd, serverAddr,0)) {
            break; // Packet sent successfully
        }
        retries++;
    }
}
```

הפונקציה שולחת פקטה עד שהיא מקבלת אישור עליה או עד שהגיע למספר נסיונות מקסימלי.
משתמשת בסוקט שנשלח לה.

בס"ד

```c
uint16_t rudp_recv(int sockfd, struct sockaddr_in *clientAddr, struct RUDP_Header *packetHeader) {
    socklen_t clientAddrLen = sizeof(*clientAddr);
    int numBytesReceived = recvfrom(sockfd, packetHeader, sizeof(*packetHeader), 0, (struct sockaddr *)clientAddr, &clientAddrLen);
    if (numBytesReceived < 0) {
        perror("recvfrom failed");

        return 0xFFFF; // Error receiving packet
        //exit(EXIT_FAILURE);
    }
    printf("Packet received.\n");
    printf("Packet checksum: %d\n", ntohs(packetHeader->checksum));
    return ntohs(packetHeader->checksum);
}
```

זאת הפונקציה שמקבלת פקטות, אמרו שלא חשוב לשמור על התוכן אז פשוט מחזירה את הchecksum. משתמשת בסוקט ששולחים לה.

RUDP_Sender:

```c
#include "RUDP_API.c"

int main(int argc, char *argv[]) {
    if (argc != 3) {
        fprintf(stderr, "Usage: %s <Receiver IP> <Port>\n", argv[0]);
        exit(EXIT_FAILURE);
    }
    char* ip = argv[1];
    int port = atoi(argv[2]);
    int sockfd = rudp_socket();
    struct sockaddr_in serverAddr;
    struct RUDP_Header packetHeader;
    uint16_t textLength = MAX_PAYLOAD_SIZE;

    // Configure server address
    memset(&serverAddr, 0, sizeof(serverAddr));
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_port = htons(port);
    if (inet_pton(AF_INET, ip, &serverAddr.sin_addr) < 0) {
        perror("inet_pton failed");
        exit(EXIT_FAILURE);
    }

    // Perform handshake with the receiver
    if (performHandshake(sockfd, &serverAddr) < 0) {
        printf("Handshake failed. Exiting.\n");
        rudp_close(sockfd);
        return 1;
    }
```

קבלת ארגומנטים מהטרמינל, הגדרת שימוש בכתובות ipv4, המרה כתובת לייצוג בינארי, וביצוע לחיצת ידיים לreceiver.

```c
    // Generate random data to send and extract it from a file to array
    char *fdata = util_generate_random_data(FILESIZE);
    FILE *file = fopen("random.txt", "w");

    if (file == NULL) {
        perror("Error opening file");
        return 1; // Return an error code
    }
    fwrite(fdata, sizeof(char), FILESIZE, file);
    fclose(file);
    FILE *fr = fopen("random.txt", "r");
    if (fr == NULL) {
        perror("Error opening file");
        return 1; // Return an error code
    }
    fseek(file, 0, SEEK_END);
    long fileSize = ftell(file);
    fseek(file, 0, SEEK_SET);
    printf("File size: %ld\n", fileSize);
    char *data = (char *)malloc(fileSize + 1);
    if (data == NULL) {
        perror("Error allocating memory");
        fclose(file);
        return 1;
    }
    size_t bytesRead = fread(data, sizeof(char), fileSize, file);
    data[bytesRead] = '\0';
    fclose(file);
```

בס"ד

יוצרים את הטקסט הרנדומלי ופותחים קובץ, רואים את הגודל.

```
    int check = 1;
    int i = 1;
    while (check) {
        i = 1;
        int amount = fileSize / MAX_PAYLOAD_SIZE;
        if(fileSize % MAX_PAYLOAD_SIZE != 0)
        {
            amount++; //find out how many packets we need to send
        }
        while (i <= amount)
        {
            char txt[MAX_PAYLOAD_SIZE];
            for (size_t s = 0; s < MAX_PAYLOAD_SIZE; s++)
            {
                txt[s] = data[i*MAX_PAYLOAD_SIZE + s]; //split text to smaller buffers so they can fit in packets
            }
            packetConstruct(&packetHeader, txt, textLength, i, REGFLAG); //build packet
            rudp_send(sockfd, &serverAddr, &packetHeader); //send packet
            i++;
        }
        char act;
        printf("Do you want to send more data? (y/n)\n");
        scanf(" %c", &act);
        if (act == 'n') check = 0;
        else if (act == 'y') check = 1;

    }
    free(data);
    free(fdata);
    rudp_close(sockfd);

    return 0;
}
```

בלולאה לוקחים את הטקסט, מפרקים לחתיכות עד גודל המקס' תוכן payload של הפקטה. בונים חבילה לפיסת הטקסט ושולחים לreceiver. אם רוצים לשלוח שוב נכנסים ללולאה שוב.

בסוף משחררים את כל הmalloc-ים.

RUDP_Reciever.c:

```
// rudp reciever
#include "RUDP_API.c"

struct timeval start, end;
void print_stats(struct timeval start, struct timeval end, int totalReceived) {
    double time_taken = (end.tv_sec - start.tv_sec) + (end.tv_usec - start.tv_usec) / 1e6;
    double bandwidth = (totalReceived / 1024.0) / time_taken;
    static double avgBandwidth = 0;
    avgBandwidth += bandwidth;
    static double avgTime = 0;
    avgTime += time_taken;
    static int counter = 0;
    printf("Current Time taken: %.2f seconds\n", time_taken);
    printf("Current Average Bandwidth: %.2f KB/s\n", bandwidth);
    printf("--------------------------------\n");
    printf("Average Time: %.2f seconds\n", avgTime / ++counter);
    printf("Average Bandwidth: %.2f KB/s\n", avgBandwidth / counter);
}
```

הפונקציה שמחשבת סטטיסטיקות ובסוף תדפיס אותן.

בס"ד

```c
int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "Usage: %s <Port>\n", argv[0]);
        exit(EXIT_FAILURE);
    }
    int port = atoi(argv[1]);
    int sockfd = rudp_socket();
    struct sockaddr_in serverAddr, clientAddr;
    struct RUDP_Header packetHeader;

    // Configure server address
    memset(&serverAddr, 0, sizeof(serverAddr));
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_addr.s_addr = INADDR_ANY;
    serverAddr.sin_port = htons(port);

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

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

    if (receiveHandshake(sockfd, &clientAddr) <= 0) {
        printf("Handshake failed.\n");
        rudp_close(sockfd);
        exit(EXIT_FAILURE);

    }
```

קבלת ארגומנטים, השמה לipv4, " לינקוז' " הסוקט עם הפורט. קבלת לחיצת הידיים שהסנדר שולח.

```c
    int amount = FILESIZE / MAX_PAYLOAD_SIZE;
    if (FILESIZE % MAX_PAYLOAD_SIZE != 0)  {amount++;}
    gettimeofday(&start, NULL);

    while (1) {
        int i = 1;
        gettimeofday(&start, NULL);
        while(i <= amount)
        {
            while (1)
            {
                uint16_t receivedChecksum = rudp_recv(sockfd, &clientAddr, &packetHeader);
                if (receivedChecksum == i)
                {
                    send_receiveAck(sockfd, &clientAddr, 1);//sending ack
                    i++;
                    break;
                }
            }

        }
        gettimeofday(&end, NULL);
        printf("Received data\n");
        print_stats(start, end, FILESIZE);


    }
    rudp_close(sockfd);

    return 0;
}
```

חישוב כמה חבילות אמורות להגיע, תחילת מדידת זמן, בלולאה מקבלים מהסנדר חבילות, אם הצ'קסם מסתדר שולחים ACK.

כשמסיימים לקבל דוגמים שוב את הזמן ושולחים לחישוב סטטיטיקות. סוגרים סוקט.

בס"ד

חלק C:

עשיתי את אפשרות הבונוס – large dataset.

להלן צילומי מסך לתוצאות ההרצות השונות, ההקלטות מצורפות בקובץ zip, תשובות לשאלות הפתוחות בסוף.

<u>0% איבוד פקטות:</u>

RECEIVER & SENDER RENO:

```
ayelet@ayelet-VirtualBox:~/Documents/network_communications_3$ ./TCP_Receiver -p 12345 -al
go reno
Starting Receiver...
Waiting for TCP connection...
TCP connection established
Received 65482 bytes
Received 130964 bytes
Received 196446 bytes
Received 1604795 bytes
Received 2096315 bytes
Received 838 bytes
Received 524702 bytes
Received 2097991 bytes
Received 196449 bytes
Received 654830 bytes
Received 2097153 bytes
Received 2097153 bytes
Received 2097153 bytes
Received 4 bytes
Data received and written
Total received: 10485769 bytes
Time taken: 10.39 seconds
Average Bandwidth: 986.01 KB/s
--------------------------------
Average Time: 10.39 seconds
Average Bandwidth: 986.01 KB/s
```

```
ayelet@ayelet-VirtualBox:~/Documents/network_communications_3$ ./TCP_Sender -ip 127.0.0.1
-p 12345 -algo reno
Sender Connected To Receiver...
Data sent.
Do you wish to resend the data? (1 for yes, 0 for no): 1
Data sent.
Do you wish to resend the data? (1 for yes, 0 for no): 1
Data sent.
Do you wish to resend the data? (1 for yes, 0 for no): 1
Data sent.
Do you wish to resend the data? (1 for yes, 0 for no): 1
Data sent.
Do you wish to resend the data? (1 for yes, 0 for no): 0
```

RECEIVER & SENDER CUBIC:

```
Received 2097153 bytes
Received 2097153 bytes
Received 2097153 bytes
Received 130966 bytes
Received 2097153 bytes
Received 4 bytes
Data received and written
Total received: 10485769 bytes
Time taken: 7.21 seconds
Average Bandwidth: 1421.02 KB/s
--------------------------------
Average Time: 7.21 seconds
Average Bandwidth: 1421.02 KB/s
ayelet@ayelet-VirtualBox:~/Documents/network_communications_3$
```

(receiver)

RECEIVER RENO – SENDER CUBIC:

```
Total received: 10485769 bytes
Time taken: 9.08 seconds
Average Bandwidth: 1128.22 KB/s
--------------------------------
Average Time: 9.08 seconds
Average Bandwidth: 1128.22 KB/s
ayelet@ayelet-VirtualBox:~/Docume
```

בס"ד

:RECEIVER CUBIC – SENDER RENO

```
Total received: 10485769 bytes
Time taken: 8.46 seconds
Average Bandwidth: 1210.32 KB/s
--------------------------------
Average Time: 8.46 seconds
Average Bandwidth: 1210.32 KB/s
ayelet@ayelet-VirtualBox:~/Docume
```

2% איבוד פקטות:

:RECEIVER & SENDER RENO

```
Data received and written
Total received: 10485769 bytes
Time taken: 4.28 seconds
Average Bandwidth: 2391.36 KB/s
--------------------------------
Average Time: 4.28 seconds
Average Bandwidth: 2391.36 KB/s
ayelet@ayelet-VirtualBox:~/Docum
```

:RECEIVER & SENDER CUBIC

```
Data received and written
Total received: 10485769 bytes
Time taken: 6.36 seconds
Average Bandwidth: 1610.31 KB/s
--------------------------------
Average Time: 6.36 seconds
Average Bandwidth: 1610.31 KB/s
ayelet@ayelet-VirtualBox:~/Docume
```

:RECEIVER RENO – SENDER CUBIC

```
Total received: 10485769 bytes
Time taken: 7.36 seconds
Average Bandwidth: 1391.79 KB/s
--------------------------------
Average Time: 7.36 seconds
Average Bandwidth: 1391.79 KB/s
ayelet@ayelet-VirtualBox:~/Docume
```

:RECEIVER CUBIC – SENDER RENO

```
Total received: 10485769 bytes
Time taken: 8.37 seconds
Average Bandwidth: 1224.07 KB/s
--------------------------------
Average Time: 8.37 seconds
Average Bandwidth: 1224.07 KB/s
ayelet@ayelet-VirtualBox:~/Docume
```

5% איבוד פקטות:

:RECEIVER & SENDER RENO

```
Total received: 10485769 bytes
Time taken: 6.96 seconds
Average Bandwidth: 1472.31 KB/s
--------------------------------
Average Time: 6.96 seconds
Average Bandwidth: 1472.31 KB/s
ayelet@ayelet-VirtualBox:~/Document
```

:RECEIVER & SENDER CUBIC

בס"ד

```
Data received and written
Total received: 10485769 bytes
Time taken: 6.62 seconds
Average Bandwidth: 1545.89 KB/s
--------------------------------
Average Time: 6.62 seconds
Average Bandwidth: 1545.89 KB/s
ayelet@ayelet-VirtualBox:~/Docume
```

:RECEIVER RENO – SENDER CUBIC

```
Data received and written
Total received: 10485769 bytes
Time taken: 7.42 seconds
Average Bandwidth: 1380.12 KB/s
--------------------------------
Average Time: 7.42 seconds
Average Bandwidth: 1380.12 KB/s
ayelet@ayelet-VirtualBox:~/Docum
```

:RECEIVER CUBIC – SENDER RENO

```
Data received and written
Total received: 10485769 bytes
Time taken: 7.13 seconds
Average Bandwidth: 1435.31 KB/s
--------------------------------
Average Time: 7.13 seconds
Average Bandwidth: 1435.31 KB/s
ayelet@ayelet-VirtualBox:~/Docume
```

10% איבוד פקטות:

:RECEIVER & SENDER RENO

```
Data received and written
Total received: 10485769 bytes
Time taken: 8.58 seconds
Average Bandwidth: 1193.87 KB/s
--------------------------------
Average Time: 8.58 seconds
Average Bandwidth: 1193.87 KB/s
ayelet@ayelet-VirtualBox:~/Docume
```

:RECEIVER & SENDER CUBIC

```
Data received and written
Total received: 10485769 bytes
Time taken: 6.73 seconds
Average Bandwidth: 1521.72 KB/s
--------------------------------
Average Time: 6.73 seconds
Average Bandwidth: 1521.72 KB/s
ayelet@ayelet-VirtualBox:~/Docume
```

:RECEIVER RENO – SENDER CUBIC

```
Data received and written
Total received: 10485769 bytes
Time taken: 8.41 seconds
Average Bandwidth: 1218.23 KB/s
--------------------------------
Average Time: 8.41 seconds
Average Bandwidth: 1218.23 KB/s
ayelet@ayelet-VirtualBox:~/Documen
```

:RECEIVER CUBIC – SENDER RENO

```
Data received and written
Total received: 10485769 bytes
Time taken: 6.67 seconds
Average Bandwidth: 1536.32 KB/s
--------------------------------
Average Time: 6.67 seconds
Average Bandwidth: 1536.32 KB/s
ayelet@ayelet-VirtualBox:~/Docume
```

בס"ד

RUDP 0%

```
ayelet@ayelet-VirtualBox:~/Documents/network_communications_3$ ./RUDP_Recei
ver 12345
Server listening on port 12345
Handshake packet received and acknowledged.
Waiting for acknowledgment...
Received data
Current Time taken: 0.18 seconds
Current Average Bandwidth: 11421.85 KB/s
--------------------------------
Average Time: 0.18 seconds
Average Bandwidth: 11421.85 KB/s
recvfrom failed: Resource temporarily unavailable
recvfrom failed: Resource temporarily unavailable
Received data
Current Time taken: 5.00 seconds
Current Average Bandwidth: 410.47 KB/s
--------------------------------
Average Time: 2.59 seconds
Average Bandwidth: 5916.16 KB/s
Received data
Current Time taken: 1.91 seconds
Current Average Bandwidth: 1073.15 KB/s
--------------------------------
Average Time: 2.36 seconds
Average Bandwidth: 4301.82 KB/s
Received data
Current Time taken: 1.29 seconds
Current Average Bandwidth: 1595.55 KB/s
--------------------------------
Average Time: 2.09 seconds
Average Bandwidth: 3625.25 KB/s
recvfrom failed: Resource temporarily unavailable
Received data
Current Time taken: 2.58 seconds
Current Average Bandwidth: 795.08 KB/s
--------------------------------
Average Time: 2.19 seconds
Average Bandwidth: 3059.22 KB/s
```

סה"כ 10.96~ שניות ל10485769 בתים, רוחב פס ממוצע 3059.22 KB/S.

2%

```
Received data
Current Time taken: 393.29 seconds
Current Average Bandwidth: 5.21 KB/s
--------------------------------
Average Time: 393.29 seconds
Average Bandwidth: 5.21 KB/s
```

די קרס, לקח 6+ דקות. בהנחה שהיינו שולחים חמישה קבצים ככה היינו מגיעים לכ30 דקות. ביצוע נוראי.

5%

```
Received data
Current Time taken: 744.18 seconds
Current Average Bandwidth: 2.76 KB/s
--------------------------------
Average Time: 744.18 seconds
Average Bandwidth: 2.76 KB/s
```

המספרים די מספרים הכל. 12~ דקות. רוחב פס 5.21. בשליחת קובץ יחיד. אם היינו עושים חמישה יכול להגיע בהערכה גסה לזמן המתנה של שעה.

למען האמת אחרי פסק זמן מסוים של 10% כבר פשוט ויתרתי לחכות לכן לא מצורף.

בס"ד

שאלות פתוחות חלק C:

1) בכללי ביצועי CUBIC לעומת RENO – ב0% איבוד פקטות הCUBIC נתן ביצוע מרשים הרבה יותר מהRENO גם מבחינת הזמן גם מבחינת רוחב הפס הממוצע לאורך השליחה.
ב2% הRENO היה הרבה יותר מרשים, גם במחינתם שני הפרמטרים. ב5% שני אלגוריתמי הויסות הגיעו לתוצאות קרובות מאוד, ב10% שוב הCUBIC היה יותר טוב.
ככה שבאופן כללי ההבדלים לא דרסטיים אבל הCUBIC הראה מהירות ורוחב פס טובים יותר מהRENO, אבל הדאטהסט למרות ההרחבה של הבונוס עדיין קטן מידי כדי לדעת בודאות.
באחוזי איבוד פקטות גבוהים הCUBIC נתן תוצאות מוצלחות יותר, גם במהירות גם ברוחב פס (צילומי מסך לעיל, הקלטות מצורפות בzip).

2) TCP לעומת RUDP: ממה שראיתי בהרצות ללא אחוזי איבוד זמן הריצה שלהם היה כמעט זהה, אבל הRUDP שמר על ממוצע רוחב פס גדול פי 3 מהTCP מה שדי יפה אם הזמנים שווים. באחוזים איבוד פקטות אבל הRUDP היה נוראי לעומת הTCP. הTCP אפילו לא הוסיף הרבה לזמן שלו, רוחב הפס של הTCP באחוזים גבוהים שמר על ממוצע 1493 קילובייט לשניה. הRUDP בכלל לא מומלץ כאשר צפויים אחוזי איבוד.
לכן למרות שהRUDP היה מרשים בהתחלה, באחוזי איבוד נמוכים וגבוהים כאחד עדיף TCP (אולי מימושים אחרים יתנו תוצאות הרבה יותר טובות אבל ככה נראה מהניסוי הזה).

3) מהנראה מההרצות הנוכחיות הRUDP נראה מומלץ כאשר לא צפויים אחוזי איבוד פקטות, אך על אחוזי איבוד פקטות היה צריך להשתמש בTCP, אפשר לראות בצילומי המסך איזה תוצאות הRUDP בגרסה הזאת נתן. מאוד לא מומלץ.