רשתות תקשורת מטלה 3

אליאן אילוק- 214787483

אריאל לבוביץ- 326535242

חלק ב-

צילומי מסך של הקוד-

בקבצי הקוד עצמם יש מגוון רחב של הערות שמסבירות את הקוד בצורה הטובה ביותר, נצרף לפה צילומי מסך של הקוד עצמו ועם הסבר קטן שלנו שמראה מה קורה שם:

הקובץ RUDP_Receiver.c:

חילוץ הפרמטרים הנחוצים משורת הפקודה,יצירת הסוקט,ויצירת הheader:

```c
#include "RUDP_API.h"
#include <time.h>

// Function prototypes
void usage(char *progname);

int main(int argc, char *argv[]) {
    if (argc != 3 || strcmp(argv[1], "-p") != 0) {  //checking the command line
        usage(argv[0]);
        return FAILURE;
    }

    printf("~~~~~~~~ RUDP Receiver ~~~~~~~~\n");
    int port = atoi(argv[2]);//receive the port
    if (port <= 0) {//check if the post is ok
        printf("Invalid port number\n");
        return FAILURE;
    }

    // Create the RUDP socket
    int sockfd = rudp_socket(AF_INET, SOCK_DGRAM, 0, 0);
    if (sockfd < 0) {//if fail, exit the program
        printf("Failed to create RUDP socket.\n");
        return FAILURE;
    }
    printf("Socket created.\n");

    // Bind the socket to the provided port
    struct sockaddr_in my_addr;
    memset(&my_addr, 0, sizeof(my_addr));
    my_addr.sin_family = AF_INET;
    my_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    my_addr.sin_port = htons(port);
```

חיבור הקשר בין המקבל לשולח, יצירת קובץ הסטטיסטיקות ויצירת המשתנים הדרושים:

```c
if (bind(sockfd, (struct sockaddr *)&my_addr, sizeof(my_addr)) < 0) {//chech if the bind work and ok
    printf("Bind failed");
    close(sockfd);
    return FAILURE;
}

printf("Connection established.\n");
printf("Listening on port %d...\n", port);

FILE* file = fopen("stats", "w+");//open the file for statistics
if (file == NULL) {
    printf("Error opening  stats file!\n");
    return 1;
}
fprintf(file, "\n\n~~~~~~~~ Statistics ~~~~~~~~\n");
double average_time = 0;//to calculate the time for the statistics
double average_speed = 0;//to calculate the speed for the statistics
clock_t start, end;

char totalData[4194304] = {0};  // 4 MB
int run = 1; // a number of sent files.

start = clock();
end = clock();

// Buffer for receiving data, parameters for the receiving data
char buffer[RUDP_BUFFER_SIZE];
struct sockaddr_storage sender_addr;
socklen_t addr_len = sizeof(sender_addr);
ssize_t received_bytes;
int rcv_status;
int seqnum = 0;
```

תחילת הלולאה- קבלת המידע ובדיקת התקינות:

```c
do {
    rcv_status = 0;//parameter for the status of the receiving data
    // Receive data chunks
    received_bytes = rudp_recv(sockfd, (struct sockaddr *)&sender_addr, &addr_len, buffer, sizeof(buffer), seqnum, &rcv_sta
    if (received_bytes < 0) {
        printf("Receive error\n");
        break;
    }
    if (rcv_status == -2) {  // if the connection was closed by the sender
        printf("Connection closed\n");
        break;
    }

    if (rcv_status == -1) {//if there was an error receiving the data
        printf("Error receiving data\n");
        return -1;
    }

    // start the timer for the first received packet.
    if (start < end) {
        start = clock();
    }

    // if this is not the last data packet, add it to the total data
    if (rcv_status == 1) {
        strcat(totalData, buffer);
        seqnum++;
    }
```

מדידת הזמנים, יצירת הסטטיסטיקות והדפסתם וסגירת הקשר:

```c
    if (rcv_status == 2) {
        // if we got the last data packet, take it and write the stats it to the file
        seqnum = 0;
        strcat(totalData, buffer);
        printf("Received total data length: %zu\n", strlen(totalData));
        end = clock();
        double time_taken = ((double)(end - start)) / CLOCKS_PER_SEC;
        average_time += time_taken;
        float rcvdDataInMb = strlen(totalData) / (1024*1024);
        double speed = rcvdDataInMb / time_taken;
        average_speed += speed;
        fprintf(file, "Run #%d Data: Time=%f S ; Speed=%f MB/S\n", run,
                time_taken, speed);
        memset(totalData, 0, sizeof(totalData));
        run++;
    }
} while (rcv_status >= 0);
// print stats
// add the average time and speed to the file
fprintf(file, "\n");    (char [20])"Average time: %f S\n"
fprintf(file, "Average time: %f S\n", average_time / (run - 1));
fprintf(file, "Average speed: %f MB/S\n", average_speed / (run - 1));
// end the file with nice message
fprintf(file, "\n\n----------------------------\n");
fprintf(file, "Thank you for using our RUDP service\n");
rewind(file);
char print_buffer[100];
while (fgets(print_buffer, 100, file) != NULL) {
    printf("%s", print_buffer);
}
// Close the file
fclose(file);
remove("stats");

return 0;
```

קובץ RUDP_Sender.c:

חילוץ המשתנים משורת הפקודה והשמתם במשתנים, יצירת הסוקט ואתחול הheader:

```c
#include "RUDP_API.h"
#include <fcntl.h> // For file operations
#include <errno.h>

// Function prototypes
void usage(char *progname);

int main(int argc, char *argv[]) {
    if (argc != 6 || strcmp(argv[1],"-ip") != 0 || strcmp(argv[3],"-p") != 0 ) {//cheking the command line
        usage(argv[0]);
        return FAILURE;
    }
    //placement of the command line parameters to new variables
    const char *ip = argv[2];
    int port = atoi(argv[4]);
    const char *filename = argv[5];

    // Create the RUDP socket
    int sockfd = rudp_socket(AF_INET, SOCK_DGRAM, 0, 1);
    if (sockfd < 0) {
        printf("Failed to create RUDP socket\n");
        return FAILURE;
    }

    // Set up the destination address
    struct sockaddr_in dest_addr;
    memset(&dest_addr, 0, sizeof(dest_addr));
    dest_addr.sin_family = AF_INET;
    dest_addr.sin_port = htons(port);
    if (inet_pton(AF_INET, ip, &dest_addr.sin_addr) <= 0) {
        printf("Invalid IP address format\n");
        return FAILURE;
    }

```

לחיצת ידיים(שליחת SYN), פתיחת הקובץ והתחלה של שליחת הקובץ למקבל:

```c
    // Buffer for reading from the file
    char buffer[4194304]; // 4 MB
    ssize_t read_bytes, sent_bytes;

    // handshake with the receiver (send SYN)
    char buf[1];
    int attempts = 0;
    while (attempts < MAX_RETRANS_ATTEMPTS) {//sending until it too much attempts
        sent_bytes = rudp_send(sockfd, (struct sockaddr *)&dest_addr, sizeof(dest_addr), buf, 1, SYN); //sending ack
        if (sent_bytes < 0) {
            attempts++;
        } else {
            break;
        }
    }
    if (sent_bytes < 0) {//if the handshake wasnt success
        printf("Failed to handshake with the receiver\n");
        return FAILURE;
    }

    printf("Sending the file\n");
    int keepSending = 0;
    // Send the file in chunks
    do {
        // Open the file
        int file_fd = open(filename, O_RDONLY);
        if (file_fd < 0) {
            printf("Failed to open file");
            return FAILURE;
        }

```

שליחת הקובץ למקבל, בודקים האם רוצים לשלוח שוב את הקוב. וסגירת הקשר:

```c
        read_bytes = read(file_fd, buffer, sizeof(buffer));//reading sizeof a buffer from the file
        // close the file so that it can be reopen in the next loop iteration.
        close(file_fd);

        //send the data from the file to the receiver
        if (read_bytes > 0) {
            sent_bytes = rudp_send(sockfd, (struct sockaddr *)&dest_addr, sizeof(dest_addr), buffer, read_bytes, 0);

            if (sent_bytes < 0) {
                printf("Failed to send data\n");
                break;
            }
        }
        //cheking ig the client want to send the file again
        printf("Do you want to send the file again? (1 - yes, any other character - no): ");
        scanf("%d", &keepSending);
    } while (keepSending == 1);

    // send FIN to indicate the receiver to stop listening.
    sent_bytes = rudp_send(sockfd, (struct sockaddr *)&dest_addr, sizeof(dest_addr), buf, 1, FIN);

    // Close RUDP connection
    printf("Closing the connection\n");
    rudp_close(sockfd);
    printf("Connection closed\n");

    return 0;
}
//example how to write in the command line
void usage(char *progname) {
    printf("Usage: %s -ip <IP> -p <Port> <Filename>\n", progname);
    printf("Example: %s -ip 127.0.0.1 -p 12345 myfile.txt\n", progname);
}
```

קובץ RUDP_API.h:

ייבוא הספריות הדרושות, הגדרת הקבועים והדגלים ויצירת הrudp header:

```c
#ifndef RUDP_API_H
#define RUDP_API_H
//libaries for functions we use in our code
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/time.h>
//constants we use many times in our code
#define RUDP_PORT 12345
#define RUDP_BUFFER_SIZE 1500
#define ACK_TIMEOUT 1 // 1 seconds
#define SENDER_TIMEOUT 2
#define SYN_TIMEOUT 10
#define INIT_SOCKET_TIMEOUT 60
#define ENDLESS_TIMEOUT 10000000
#define MAX_RETRANS_ATTEMPTS 5
#define FAILURE -1
#define SUCCESS 0

// Define flags for RUDP header
#define SYN 0x01
#define ACK 0x02
#define FIN 0x04
#define DATA 0x08
//struct for the RUDP header, contains its length,checksum for each packet,flags and sequence number for the packet loss
typedef struct {
    uint16_t length;
    uint16_t checksum;
    uint8_t flags;
    uint16_t seqnum;
} rudp_header;

```

הגדרת הפונקציות אותם נממש בקובץ RUDP_API.c:

```c
int rudp_socket(int domain, int type, int protocol, int isSender);
ssize_t rudp_send(int sockfd, const struct sockaddr *dest_addr, socklen_t addrlen, const void *buf, size_t len, int sendFin);
ssize_t rudp_recv(int sockfd, struct sockaddr *src_addr, socklen_t *addrlen, void *buf, size_t len, int seqnum, int *status);
ssize_t rudp_recv(int sockfd, struct sockaddr *src_addr, socklen_t *addrlen, void *buf, size_t len, int seqnum, int *status);
int rudp_close(int sockfd);
```

הקובץ RUDP_API.c:

בקובץ זה יש מספר פונקציות בהם השתמשנו כדי לממש את הקשר בין השולח למקבל, נרשום בקצרה מה כל פונקציה עושה, הקוד עצמו של כל פונקציה מתועד עד לפרטים הקטנים ולכן לא נרשום אותו כאן שוב.

פונקציה הקובעת את הtimeout ופונקציה המחשבת את הchecksum:

```c
// set timeout for the socket
int set_timeout(int socket, int time) {
  struct timeval timeout;
  timeout.tv_sec = time;
  timeout.tv_usec = 0;

  if (setsockopt(socket, SOL_SOCKET, SO_RCVTIMEO, &timeout, sizeof(timeout)) < 0) {
    printf("Error setting timeout for socket");
    return FAILURE;
  }
  return SUCCESS;
}

// Helper function for a simple checksum calculation.
uint16_t simple_checksum(void *data, size_t len) {
    uint8_t *bytes = (uint8_t *)data; // Cast data pointer to a byte pointer for byte-by-byte processing
    uint32_t sum = 0; // Use a 32-bit integer to accumulate the sum to prevent overflow

    for (size_t i = 0; i < len; ++i) {
        sum += bytes[i];
    }

    // Reduce the sum to 16 bits by adding the carry (if any) back into the sum
    while (sum >> 16) {
        sum = (sum & 0xFFFF) + (sum >> 16);
    }

    return (uint16_t)sum; // Cast and return the sum as a 16-bit value
}
```

פונקציה היוצרת את הסוקט עם פרוטוקול UDP:

```c
//creating the RUDP socket
int rudp_socket(int domain, int type, int protocol, int isSender) {
    int sockfd = socket(domain, SOCK_DGRAM, protocol); // Use UDP
    if (sockfd < 0) {//if fail to create the socket, exit the program
        printf("Socket creation failed");
        return FAILURE;
    }
    //setting the timeout
    int timeout = (isSender == 1) ? SENDER_TIMEOUT : INIT_SOCKET_TIMEOUT;
    set_timeout(sockfd, timeout);

    return sockfd;
}
```

פונקציה שמחכה לקבלת ack ומדפיסה את התוצאה בהתאם:

```c
/**
 * Waits for an acknowledgment after sending a packet.
 *
 * @param sockfd The socket file descriptor.
 * @param seqnum The sequence number of packet we expect to get
 * @param timeout The timeout
 * @return 0 on success, -1 on failure (timeout or error). *
 */
int wait_for_ack(int sockfd, int seqnum, int timeout) {
    clock_t starttime = clock();
    rudp_packet packet;
    struct sockaddr_storage sender_addr;
    socklen_t addr_len = sizeof(sender_addr);
    //trying until reaching timeout
    while ((double)(clock() - starttime) / CLOCKS_PER_SEC < timeout) {
        ssize_t recv_len = recvfrom(sockfd, &packet, sizeof(packet), 0, (struct sockaddr *)&sender_addr, &addr_len);

        if (recv_len == -1) {//fail to receive ack
            printf("wait_for_ack(%d) returned FAILURE\n", seqnum);
            return FAILURE;
        }
        printf("wait_for_ack(%d), packet.header.seqnum=%d\n", seqnum, packet.header.seqnum);
        printf("packet.header.flags & ACK=%d\n", packet.header.flags & ACK);
        // if packet.header.seqnum > seqnum, it means
        // the receiver has already got the packet with the seqnum
        // but the sender didn't get the ack for the seqnum
        if (packet.header.seqnum >= seqnum && (packet.header.flags & ACK) == ACK) {
            printf("wait_for_ack(%d) returned SUCCESS\n", seqnum);
            return SUCCESS;
        }
    }
    printf("wait_for_ack(%d) returned FAILURE\n", seqnum);
    return FAILURE; // Timeout or error
}
```

פונקציה ששולחת את המידע בעזרת פרוטוקול UDP:_

```c
//send data with RUDP protocol
ssize_t rudp_send(int sockfd, const struct sockaddr *dest_addr, socklen_t addrlen, const void *buf, size_t len, int sendFin) {
    size_t totalSent = 0; // Total bytes sent
    int attempts = 0; // Retransmission attempts
    int seqnum = 0;//starting the counting

    while (totalSent < len) {//runing until sendinf all the data
        size_t chunkSize = (sendFin > 0 || len - totalSent > MAX_PAYLOAD_SIZE) ? MAX_PAYLOAD_SIZE : len - totalSent;
        rudp_packet packet;
        // Prepare the packet
        packet.header.length = htons(chunkSize);
        if (sendFin > 0) {
            packet.header.flags = sendFin;
        } else {
            packet.header.flags = DATA;
            // if we send the last packet, set the FIN flag
            if ((len - totalSent) <= sizeof(rudp_packet)) {
                packet.header.flags = packet.header.flags | FIN;
            }
        }
        packet.header.seqnum = seqnum;//setting the sequence number for each packet
        packet.header.checksum = 0; // set to zero to calculate checksum
        if (sendFin > 0) {
            memset(packet.payload, 0, MAX_PAYLOAD_SIZE);
        } else {
            memcpy(packet.payload, (char *)buf + totalSent, chunkSize);
        }
        packet.header.checksum = simple_checksum(&packet, sizeof(rudp_header) + chunkSize);//calculate checksum
        // Send the packet
        printf("sending the packet(%d) via sendto\n", seqnum);
        ssize_t sent = sendto(sockfd, &packet, sizeof(rudp_header) + chunkSize, 0, dest_addr, addrlen);
        if (sent < 0) {//if send fail, terminate the program
            printf("sendto failed\n");
            return FAILURE;
        }
        // Wait for acknowledgment
```

```c
        // Wait for acknowledgment
        printf("calling wait_for_ack(%d)\n", seqnum);
        if (wait_for_ack(sockfd, seqnum, ACK_TIMEOUT) == 0) {
            printf("sender got seqnum=%d\n", seqnum);
            totalSent += (sizeof(rudp_header) + chunkSize); // Acknowledged
            seqnum++;
            attempts = 0; // Reset attempts after successful send
        } else {//if ack fail to be delivered
            attempts++;
            if (attempts >= MAX_RETRANS_ATTEMPTS) { // Maximum retransmission attempts
                printf("Maximum retransmission attempts reached.\n");
                return FAILURE;
            }
        }
    }
    return totalSent;
}
```

פונקציה ששולחת ack:

```c
ssize_t send_ack(int sockfd, const struct sockaddr *dest_addr, socklen_t addrlen, uint8_t ack_flags, int seqnum) {
    rudp_packet ack_packet;
    memset(&ack_packet, 0, sizeof(rudp_packet));
    ack_packet.header.flags = ack_flags; // Set flags
    ack_packet.header.seqnum = seqnum;//set sequence number
    ack_packet.header.checksum = 0;//set to zero to calculate checksum
    ack_packet.header.checksum = simple_checksum(&ack_packet, sizeof(rudp_packet));//clculate checksum

    // Send the ACK packet
    ssize_t sent = sendto(sockfd, &ack_packet, sizeof(rudp_packet), 0, dest_addr, addrlen);
    if (sent < 0) {//if fail to send ack,print FAIL, else print SUCCESS
        printf("sendto (ACK) failed\n");
    }
    printf("send_ack sent seq_num=%d\n", seqnum);
    return sent;
}
```

פונקציה שמקבלת את המידע ומעבדת אותו בהתאם:

```
        if (send_ack(sockfd, src_addr, *addrlen, ACK | SYN, seqnum) < 0) {
            return FAILURE; // Failed to send ACK
        }
        return received;
    } else {
        // Send an acknowledgment back to the sender
        if (send_ack(sockfd, src_addr, *addrlen, ACK, seqnum) < 0) {
            return FAILURE; // Failed to send ACK
        }
    }

    if (header->seqnum == seqnum) {//check that the sequence number match
        if (seqnum == 0 && (header->flags & DATA) == DATA) {
            set_timeout(sockfd, SYN_TIMEOUT);
        }
        if (header->flags == (DATA | FIN)) {  // last packet
            *status = 2;
            set_timeout(sockfd, ENDLESS_TIMEOUT);
        } else if (header->flags == DATA) { // data packet, not last
            *status = 1;
        } else if (header->flags == FIN) {  // close request
            *status = -2;
            set_timeout(sockfd, SYN_TIMEOUT);
        }
    } else {
        *status = 0; // we got a packet with the wrong sequence number.
    }
    return received; // Return the length of the received data (including the header)
}
```

```
//receive the data
ssize_t rudp_recv(int sockfd, struct sockaddr *src_addr, socklen_t *addrlen, void *buf, size_t len, int seqnum, int *status) {
    *status = -1;
    char packet[RUDP_BUFFER_SIZE];//buffer that will contain the data, sizeof a buffer
    ssize_t received = recvfrom(sockfd, packet, sizeof(packet), 0, src_addr, addrlen);//receive the data
    if (received < 0) {
        printf("recvfrom failed\n");
        return FAILURE;
    }

    // Extract the header and data from the received packet
    rudp_header *header = (rudp_header*) packet;
    void *data = packet + sizeof(rudp_header);
    ssize_t data_len = received - sizeof(rudp_header);

    // Verify the checksum
    uint16_t received_checksum = header->checksum;
    header->checksum = 0; // Temporarily set to zero to calculate checksum
    uint16_t calculated_checksum = simple_checksum(packet, received);//calculate checksum
    if (received_checksum != calculated_checksum) {//if checksum doesnt match,return FALIURE
        printf("Checksum mismatch. Packet corrupted.\n");
        return FAILURE;
    }

    // If the packet is valid, copy the data to the user buffer
    if (data_len > len) {
        printf("Buffer too small to hold received data.\n");
        return FAILURE;
    }
    memcpy(buf, data, data_len);

    if ((header->flags & SYN) == SYN) {  // connection request
        printf("Received connection request\n");
        *status = 0;
        // Send an acknowledgment back to the sender
```

פונקציה שסוגרת את הקשר בין השולח למקבל:

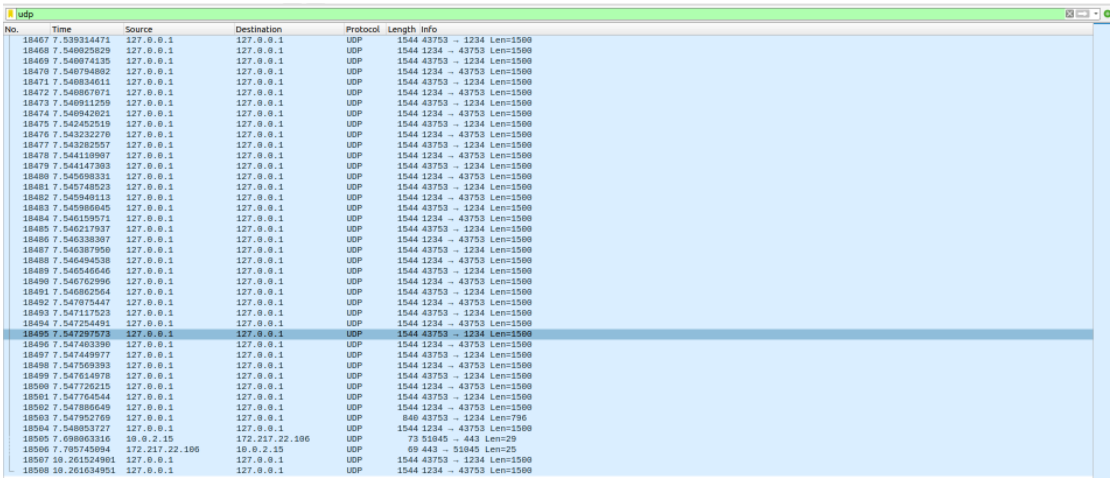
```
//close the connection
int rudp_close(int sockfd) {
    return close(sockfd);
}
```

צילומי מסך מה-wireshrark של הפקטות:

*כל הפקטות נמצאות בקובץ זיפ המצורף

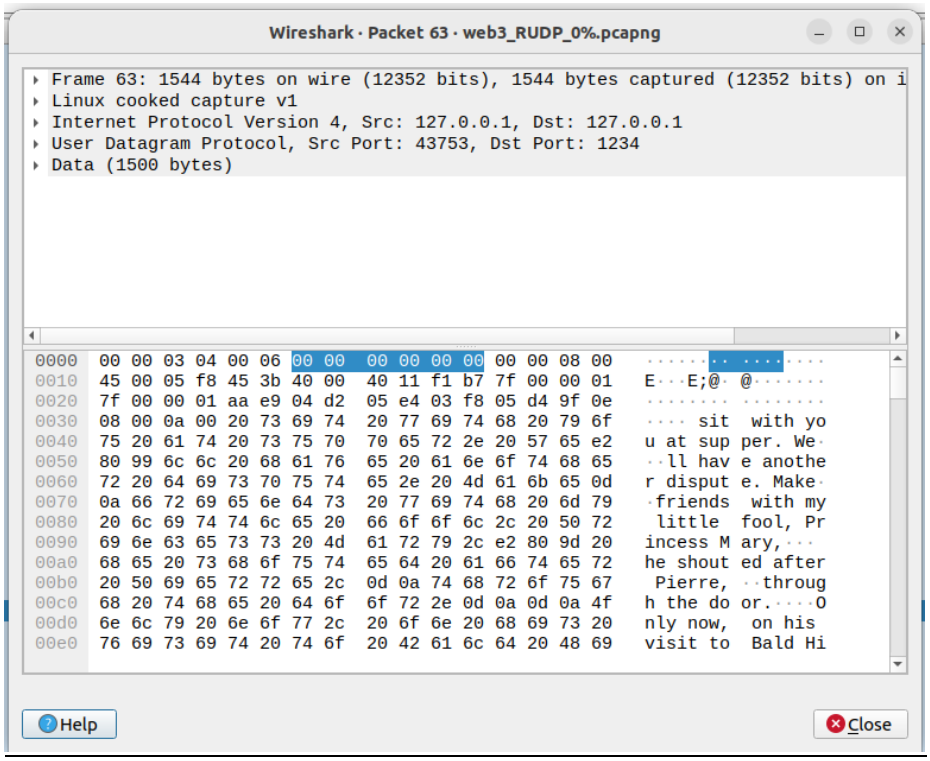
בזמן הסנפת הפקטות ניתן לראות את המידע עובר משולח למקבל בעזרת פרוטוקול UDP, כל פעם עובר מידע באורך 1500 בתים(אורך הבאפר שהגדרנו) והוא עובר מהsource ל-destination.

זהו דוגמא עבור הרצה עם 0% איבודי פקטות:



No.	Time	Source	Destination	Protocol	Length	Info
18467	7.539314471	127.0.0.1	127.0.0.1	UDP	1544	43753 → 1234 Len=1500
18468	7.540025829	127.0.0.1	127.0.0.1	UDP	1544	1234 → 43753 Len=1500
18469	7.540074135	127.0.0.1	127.0.0.1	UDP	1544	43753 → 1234 Len=1500
18470	7.540794802	127.0.0.1	127.0.0.1	UDP	1544	1234 → 43753 Len=1500
18471	7.540834611	127.0.0.1	127.0.0.1	UDP	1544	43753 → 1234 Len=1500
18472	7.540867071	127.0.0.1	127.0.0.1	UDP	1544	1234 → 43753 Len=1500
18473	7.540911259	127.0.0.1	127.0.0.1	UDP	1544	43753 → 1234 Len=1500
18474	7.540942021	127.0.0.1	127.0.0.1	UDP	1544	1234 → 43753 Len=1500
18475	7.542452519	127.0.0.1	127.0.0.1	UDP	1544	43753 → 1234 Len=1500
18476	7.543232270	127.0.0.1	127.0.0.1	UDP	1544	1234 → 43753 Len=1500
18477	7.543282557	127.0.0.1	127.0.0.1	UDP	1544	43753 → 1234 Len=1500
18478	7.544119967	127.0.0.1	127.0.0.1	UDP	1544	1234 → 43753 Len=1500
18479	7.544147303	127.0.0.1	127.0.0.1	UDP	1544	43753 → 1234 Len=1500
18480	7.545698331	127.0.0.1	127.0.0.1	UDP	1544	1234 → 43753 Len=1500
18481	7.545748523	127.0.0.1	127.0.0.1	UDP	1544	43753 → 1234 Len=1500
18482	7.545948113	127.0.0.1	127.0.0.1	UDP	1544	1234 → 43753 Len=1500
18483	7.545986045	127.0.0.1	127.0.0.1	UDP	1544	43753 → 1234 Len=1500
18484	7.546159571	127.0.0.1	127.0.0.1	UDP	1544	1234 → 43753 Len=1500
18485	7.546217937	127.0.0.1	127.0.0.1	UDP	1544	43753 → 1234 Len=1500
18486	7.546338307	127.0.0.1	127.0.0.1	UDP	1544	1234 → 43753 Len=1500
18487	7.546387950	127.0.0.1	127.0.0.1	UDP	1544	43753 → 1234 Len=1500
18488	7.546494538	127.0.0.1	127.0.0.1	UDP	1544	1234 → 43753 Len=1500
18489	7.546546646	127.0.0.1	127.0.0.1	UDP	1544	43753 → 1234 Len=1500
18490	7.546762996	127.0.0.1	127.0.0.1	UDP	1544	1234 → 43753 Len=1500
18491	7.546862564	127.0.0.1	127.0.0.1	UDP	1544	43753 → 1234 Len=1500
18492	7.547075447	127.0.0.1	127.0.0.1	UDP	1544	1234 → 43753 Len=1500
18493	7.547117523	127.0.0.1	127.0.0.1	UDP	1544	43753 → 1234 Len=1500
18494	7.547254491	127.0.0.1	127.0.0.1	UDP	1544	1234 → 43753 Len=1500
18495	7.547297573	127.0.0.1	127.0.0.1	UDP	1544	43753 → 1234 Len=1500
18496	7.547403390	127.0.0.1	127.0.0.1	UDP	1544	1234 → 43753 Len=1500
18497	7.547449977	127.0.0.1	127.0.0.1	UDP	1544	43753 → 1234 Len=1500
18498	7.547569393	127.0.0.1	127.0.0.1	UDP	1544	1234 → 43753 Len=1500
18499	7.547614978	127.0.0.1	127.0.0.1	UDP	1544	43753 → 1234 Len=1500
18500	7.547726215	127.0.0.1	127.0.0.1	UDP	1544	1234 → 43753 Len=1500
18501	7.547764544	127.0.0.1	127.0.0.1	UDP	1544	43753 → 1234 Len=1500
18502	7.547886649	127.0.0.1	127.0.0.1	UDP	1544	1234 → 43753 Len=1500
18503	7.547952769	127.0.0.1	127.0.0.1	UDP	840	43753 → 1234 Len=796
18504	7.548053727	127.0.0.1	127.0.0.1	UDP	1544	1234 → 43753 Len=1500
18505	7.698063316	10.0.2.15	172.217.22.106	UDP	73	51045 → 443 Len=29
18506	7.795745094	172.217.22.106	10.0.2.15	UDP	69	443 → 51045 Len=25
18507	10.261524901	127.0.0.1	127.0.0.1	UDP	1544	43753 → 1234 Len=1500
18508	10.261634951	127.0.0.1	127.0.0.1	UDP	1544	1234 → 43753 Len=1500

דוגמא לפקטה שעוברת מהשולח למקבל- בתוכה יש את המידע שעבר, לפי איזה פרוטוקול היא עברה(UDP), את ה- source port שמוגדר להיות 43753 וה- destination port אותו הגדרנו בהרצת הקוד להיות 1234, בנוסף לכתובת המחשב שהוגדר ל 127.0.0.1.



חלק ג-

איבודי פקטות:

*כל הפקטות נמצאות בקובץ זיפ המצורף

בשלב זה עשינו את הבונוס והרצנו את הקוד שלנו על כל אחד מהאפשרויות המבוקשות עבור TCP ועבור RUDP והפקטות המבוקשות נמצאות בקובץ הזיפ המצורף(הפקטות עבור חלק הTCP מופיע בקובץ הזיפ שהגשנו עבור חלק א, כמו כן גם ההבדלים בין TCP reno לבין TCP cubic, הסטטיסטיקות עבור כל אחד מהם ועם סוגים שונים של איבודי פקטות(10%,5%,2%,0%).

עבור RUDP:

עבור 0% איבודי פקטות:

לא היה איבודי פקטות בכלל ולכן לא היו שגיאות או תקלות במעבר מהשולח למקבל, הסטטיסטיקות שקיבלנו עבור החלק הזה הם:

```
~~~~~~~~ Statistics ~~~~~~~~
Run #1 Data: Time=0.051846 S ; Speed=19.287891 MB/S
Run #2 Data: Time=0.052799 S ; Speed=18.939753 MB/S
Run #3 Data: Time=0.045404 S ; Speed=22.024491 MB/S
Run #4 Data: Time=0.053282 S ; Speed=18.768064 MB/S
Run #5 Data: Time=0.051877 S ; Speed=19.276365 MB/S

Average time: 0.051042 S
Average speed: 19.659313 MB/S
```

עבור 2% איבודי פקטות:

היו קצת איבודי פקטות ועל כן ניתן לראות כי ישנם מספר פקטות שגיאה, הסטטיסטיקות עבור החלק הזה הם:

```
~~~~~~~~ Statistics ~~~~~~~~
Run #1 Data: Time=0.102277 S ; Speed=9.777369 MB/S
Run #2 Data: Time=0.112372 S ; Speed=8.899014 MB/S
Run #3 Data: Time=0.116598 S ; Speed=8.576476 MB/S
Run #4 Data: Time=0.097525 S ; Speed=10.253781 MB/S
Run #5 Data: Time=0.104770 S ; Speed=9.544717 MB/S
Run #6 Data: Time=0.104247 S ; Speed=9.592602 MB/S

Average time: 0.106298 S
Average speed: 9.440660 MB/S


-----------------------------
Thank you for using our RUDP service
vboxuser@ubuntu:~/web-3$
```

עבור 5% איבודי פקטות:

ניתן לראות שהחיבור בין השולח למקבל לא טוב ונאבדות פקטות בקצב גדול הרבה יותר, הסטטיסטיקות עבור החלק הזה הם:

```
~~~~~~~~ Statistics ~~~~~~~~
Run #1 Data: Time=0.140832 S ; Speed=7.100659 MB/S
Run #2 Data: Time=0.160849 S ; Speed=6.217011 MB/S
Run #3 Data: Time=0.161566 S ; Speed=6.189421 MB/S
Run #4 Data: Time=0.171546 S ; Speed=5.829340 MB/S
Run #5 Data: Time=0.162206 S ; Speed=6.165000 MB/S

Average time: 0.159400 S
Average speed: 6.300286 MB/S
```

עבור 10% איבודי פקטות:

ניתן לראות שהחיבור בין השולח למקבל רע מאוד, הפקטות נאבדות בקצב גדול מאוד, הסטטיסטיקות עבור החלק הזה הם:

```
~~~~~~~~ Statistics ~~~~~~~~
Run #1 Data: Time=0.186043 S ; Speed=5.375101 MB/S
Run #2 Data: Time=0.179826 S ; Speed=5.560931 MB/S
Run #3 Data: Time=0.216203 S ; Speed=4.625283 MB/S
Run #4 Data: Time=0.239608 S ; Speed=4.173483 MB/S
Run #5 Data: Time=0.209999 S ; Speed=4.761927 MB/S
Run #6 Data: Time=0.197425 S ; Speed=5.065215 MB/S

Average time: 0.204851 S
Average speed: 4.926990 MB/S
```

נשים לב לפי הסטטיסטיקות שככל שיש יותר איבודי פקטות כך לוקח יותר זמן לפקטות לעבור והם עוברים במהירות נמוכה הרבה יותר ואנחנו מקבלים זמנים גדולים הרבה יותר ומהירות נמוכה הרבה יותר. נסתכל על הזמן הממוצע שלקח עבור כל אחד מהאופציות של איבודי הפקטה-

אחוז איבודי פקטות-	0%	2%	5%	10%
זמן ממוצע	S0.051 ms51	S0.1062 ms106.2	S0.1594 ms159.4	S0.2048 ms204.8
מהירות ממוצעת-	mb/s19.659	mb/s9.44	mb/s6.3	mb/s4.926

נשווה את זה למשל לTCP reno:

אחוז איבודי פקטות-	0%	2%	5%	10%
זמן ממוצע	ms2.283	ms42.942	ms102.409	ms331.75
מהירות ממוצעת-	mb/s875.965	mb/s46.745	mb/s6.3	mb/s6.029

נשים לב ש TCP פועל הרבה יותר טוב עבור הרוב המוחלט של אפשרויות שיש להרצה(עם או בלי איבודי פקטות ועם איבודי פקטות כבדים או קלים).

שאלות הקשורות לTCP וRUDP:

1- נסתכל על הזמנים ועל המהירות של tcp reno ושל tcp cubic כאשר לא היה איבודי פקטות או שהיו קצת איבודי פקטות ונשים לב שtcp reno עושה עבודה קצת יותר טובה מtcp cubic וכאשר איבודי הפקטות היו גדולים קיבלנו שtcp cubic היה עדיף וטוב יותר ולכן נסיק לפי הנתונים שאספנו שכאשר יש לנו עומס גדול על המערכת או מספר גדול של איבודי פקטות אנחנו נעדיף להשתמש בtcp cubic וכאשר אין לנו עומס או שאין לנו איבודי פקטות(או שיש ממש קצת) אנחנו נעדיף להשתמש בtcp reno.

2- נשווה את הזמנים שקיבלנו ב TCP(זה לא משנה איזה אופציה של TCP) לעומת הזמנים שקיבלנו בRUDP- ניתן לראות לפי הזמנים, שה TCP עובד טוב יותר מהRUDP , הזמנים של השליחה בTCP היו קצרים הרבה יותר מהזמנים של השליחה בRUDP עבור כל סיטואציה עם ובלי איבודי פקטות בנוסף לכך שהTCP היה מהיר הרבה יותר בשליחת הפקטות ולכן בסך הכל נשים לב שTCP עבד הרבה יותר טוב מRUDP בכל האספקטים.

3- נשים לב לפי הסטטיסטיקות שאספנו שגם עם וגם בלי איבודי פקטות, הTCP עבד טוב יותר מRUDP תמיד, גם במהירות השליחה וגם בזמני השליחה ולכן נעדיף להשתמש תמיד בTCP לפי המימושים שלנו.

שאלות פתוחות:

חלק ב':

(ג) א. לא, מה שציינו ב- Stop-and-Wait יכול בעיקרון, לבוא בקופסא ב"כישר אבנים" מבלי שישרת בכלל תקינות לשלוח וניסה להבטיח אותה בדר לכן נניחה שב- Stop-and-Wait יכול בדוגה כמו שבמה הבטיח שעדרה תקינה נכונה כל פעם וצריך בשביל ל- Stop-and-Wait לשאר עם קבל תקינות לשלוח, בגיבה נאז נשלח את ה- Stop-and-Wait בגבה צה,אצ' נגיע אלין מבר ונעבור בקפה לאבל ימר את התקינות כאשר יש לנו עוד הבנה תקינות לשלוח.

בנוסף נגבה לבבע את כאבה בגיבה לא בגי כשאי אינה מבטיח שבגבה שבגר שבגל תקינות יכלו תאבדה לא מה שמשיקים בתפיקה שלנו בגלגלן שלא נאו נקשבים לתצר הבבה כמים אחרבה לשלוח את אותה תקינות שוב ושוב.

בנוסף לקשר עבדקים וכשאי אפיבה, נעבוד RTT בגבה בשאר נתם מבלין שעה RTT נתם ואמנו נקבה שהתקינה עלבה "מבר מבי" כאשר אנחנו לא מסכים לבבה אבל מבר התקינה הבאה שלחת לכן נעבוד לשמן את מה RTT בבוד לכן נתר מסלול (ב).

ב- התבנקה של הקשר מ- A ל- B:

כמן / נתה בנתונים שנשלחים בתבנה

הקשר יסתיים בזמן ב- Stop-and-Wait שבוא SSMS, נחשב את הכמן שלקח לשלוח את כל המקטעים בבחתה שנותים תקינה אחת נאז 2 ואז 4 עד שבינים ל Stop-and-Wait שבה כאשר שולחים 5 תקינות לכן סך כל הבתים שנשלחים זה $(5)\text{פול}$ בתים כי כל פעם שלחים כ' 2 תקינות מבבנים הבנבמה כל פעם הכמן שלקח הוא בסבר התקינות בגלל RTT לכן סך הכל הכמן שלקח לשלוח הוא $RTT\cdot(5)\text{פול}$. כאשר נגיע ל- Stop-and-Wait הקשר יסתיים, לכי הבבבבה בשאלה $SSMS=\text{Stop-and-Wait}$, נבבה לבבר את זה עם אנו שסתיים הקשר לכן נקלח $2\cdot SSMS$, אז אנו בוודאות לעבור את ה- Stop-and-Wait, לכן סך הכל בבה זה $RTT(5)\text{פול}\cdot\frac{2SSMS}{SSMS}$. (ד)

$X=3$ עבור

(ה) קצב התשדר: 8 Gbps

גודל תקינה: 3 KByte

קצב הבר בשדה: $2\cdot10^8\ \frac{m}{sec}$

המרחק בין תחנות: 1 km

נשים לב שגודל נתון מחושב בכ: $\frac{\text{קצב הבר בשדה}\cdot RTT}{\text{גודל התקינה}}$

בתרגיל כל אחד ואחד מקבל את כל הנתונים הנחוצים:

בנוסף התחנה נמצאת בהיא בעצמה 3, לאחר הפסד נקודת $3\cdot10^3$ לכיוון תחנות עוקבות ב- 10^8 נת תקשר את

הפסד הכיוון עדין נכון בדיוק נקודת $3\cdot10^3$)

קצב התקשרות נמדד לנו גם הוא וזה כמובן לאחר הפסד הפרישה נקודת $8\cdot10^9$

כעת נוכל לחשב את RTT שמתיחס גם: לעיכוב ·2 + זמן נאבדו תלה: RTT

$$\text{זמן השכ'} = \frac{\text{מרחק}}{\text{קצב התשדרות}} = \frac{1000}{2\cdot10^8} = 0.000005$$

$$\text{זמן נאבדו תל'} = \frac{\text{גודל הפקיטה}}{\text{קצב התשדרות}} = \frac{3\cdot10^3\cdot8}{8\cdot10^9} = 0.000003$$

$$RTT = 2\cdot(0.000003 + 0.000005) = 0.000016$$

$$\text{תפוקה} = \frac{RTT\cdot\text{קצב התשדרות}}{\text{גודל תגיל}} = \frac{0.000016\cdot8\cdot10^9}{3\cdot8\cdot10^3} = \frac{16}{3} = 5\frac{1}{3}$$