20/12/2022

Assignment 3 – TCP CC

חלק א – תיאור מערכת

תוכן עניינים

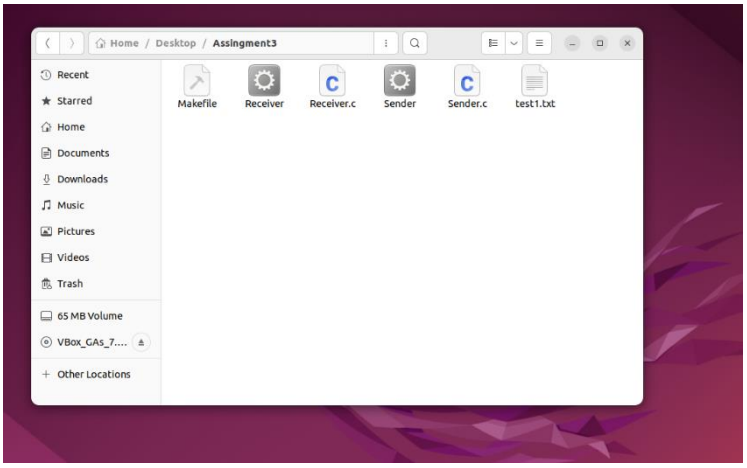
הקדמה:

בחלק זה של המטלה נתבקשנו לבנות שרת ולקוח בשפת C, להסביר את אופן הרצת התוכנה ולתאר את הקוד שכתבנו תוך הסבר של השלבים בתהליך.

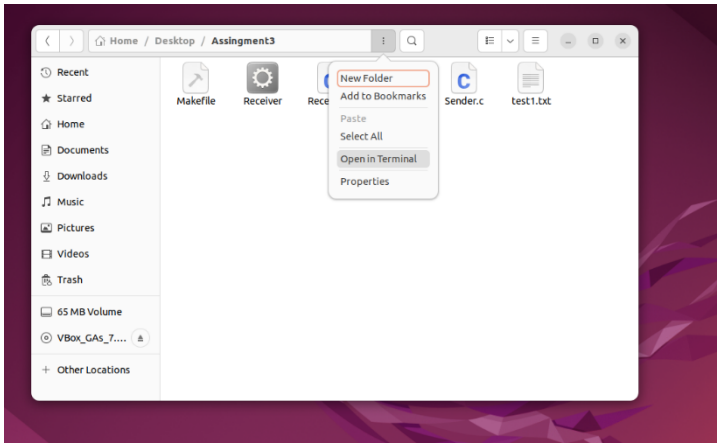
תיאור מערכת:

- הרצת התוכנה:

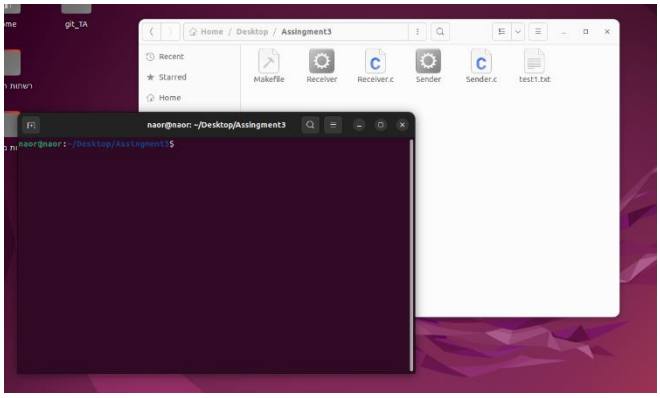
1. נפתח את התיקייה בה נמצאים קבצי ה- C ביחד עם קובץ הטקסט אותו יצרנו.



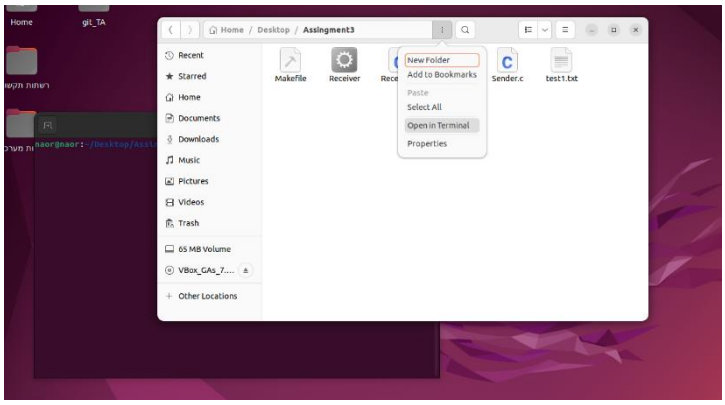
2. נלחץ על Open in Terminal



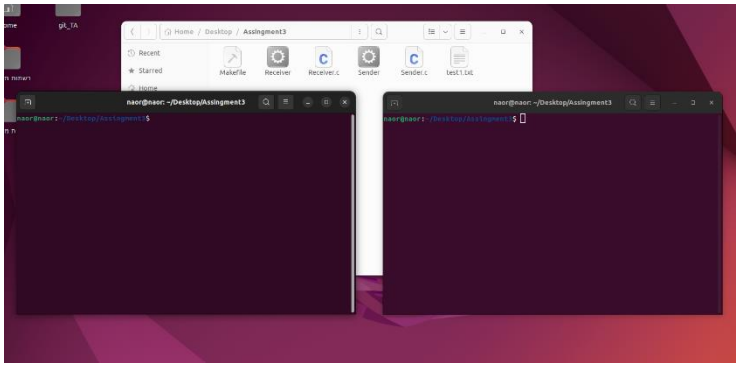
3. ייפתח לנו ה- Terminal בכתובת של התיקייה



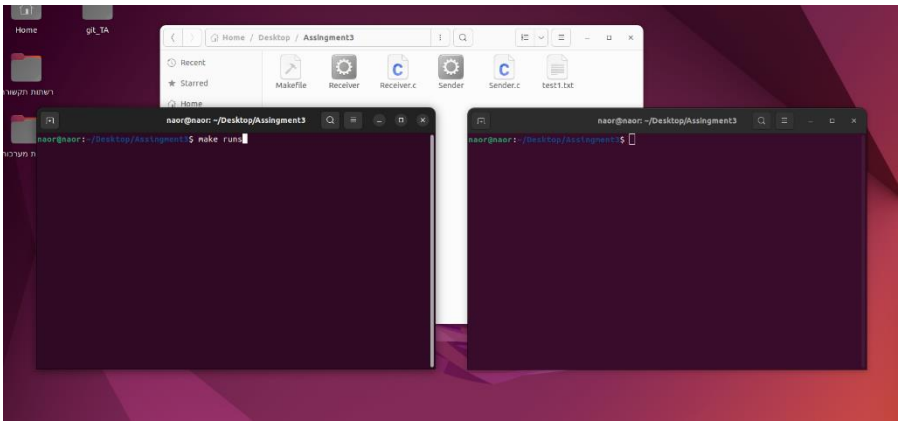
4. כעת נפתח Terminal נוסף



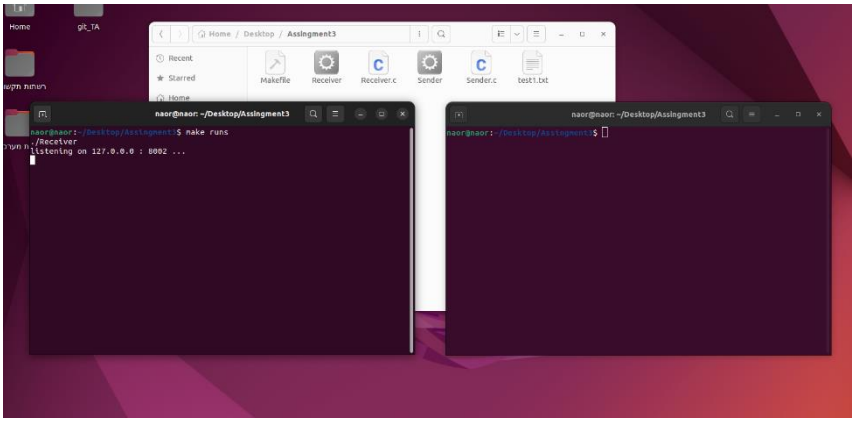
5. בשלב זה יש בפנינו שתי חלוניות של ה- Terminal



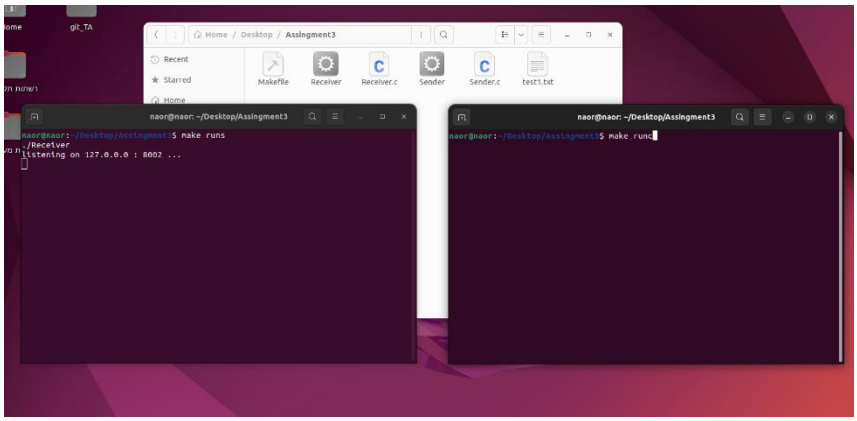
6. בחלון השמאלי נריץ תחילה את ה- Receiver ע"י הפקודה make runs.



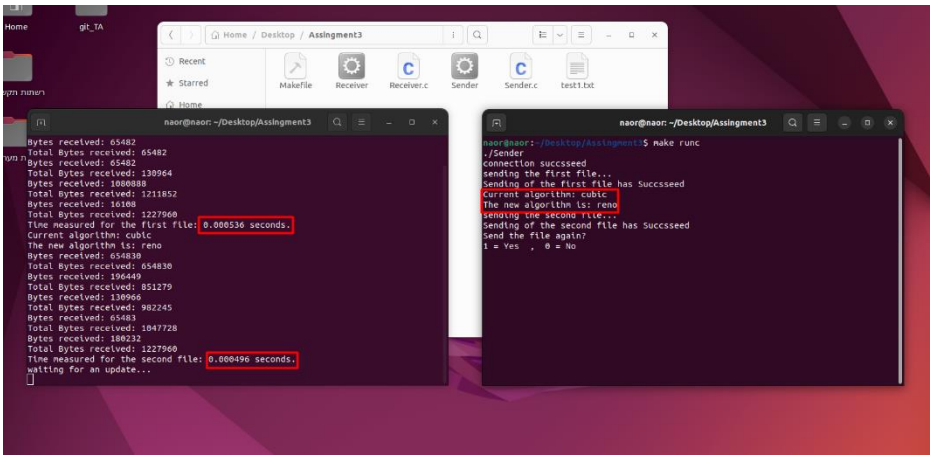
7. נשים לב כי אכן ה- Receiver רץ ברקע



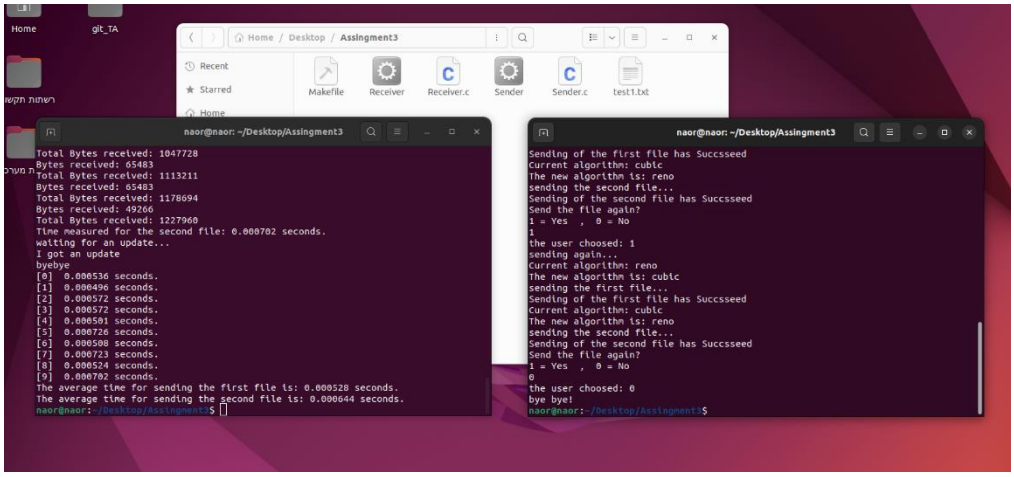
8. כעת נריץ את ה- Sender בחלון הימני ע"י הפקודה make runc.



9. כעת ניתן להבחין בהתקשרות שנוצרה בין ה- Receiver ל- Sender ובתהליך שליחת הקובץ. כדי לשלוח את הקובץ שוב נלחץ 1.



10. יציאה מהתוכנית תיעשה ע"י לחיצה על 0.



- ייעודיות התוכנה:

התוכנה משמשת ככלי עזר על מנת לחקור את השוני בין האלגוריתמים – cubic ו- reno.
היא עושה זאת באמצעות יצירת התקשרות בין שרת (Sender) ללקוח (Receiver),
שלאחריה מתבצעת שליחת קובץ טקסט בגודל 2.5 מגה בייט בשני חלקים.
התוכנה מודדת את הזמנים שלקחו ללקוח לקבל את כל החלק הראשון של הקובץ דרך שימוש באלגוריתם cubic,
ולאחר מכן מודדת את הזמנים שלקחו ללקוח לקבל את כל החלק השני של הקובץ דרך שימוש באלגוריתם reno.

-Sender
שרת ששולח קובץ טקסט ללקוח בשני חלקים כאשר בין השליחה הראשונה לשנייה הוא מבצע שינוי של האלגוריתם, ומקבל קלט מהמשתמש על מנת לדעת אם להמשיך לשליחה נוספת או לצאת מהתוכנית.

-Receiver
לקוח שמקבל את שני חלקי הקובץ כאשר בין הקבלה הראשונה לשנייה הוא מבצע שינוי של האלגוריתם ובנוסף מודד את הזמנים שלקח לכל קובץ להגיע ושומר אותם. הלקוח מקבל הודעת עדכון מהשרת האם המשתמש בחר לשלוח פעם נוספת את הקובץ או לצאת מהתוכנית. במידה והמשתמש בחר לצאת הלקוח ידפיס את הזמנים לפי הסדר, ובנוסף יחשב וידפיס את הזמן הממוצע שלקח לכל חלק להגיע בהתאם לאלגוריתם שבו הוא נשלח.

- תיאור קוד:

<u>Sender:</u>
בשלב הראשון התוכנה קוראת את קובץ הטקסט בגודל 2.5 מגה בייט שיצרנו אל תוך מערך Char file[] בגודל הקובץ. לאחר מכן התוכנה מפצלת את המערך לשני מערכים- f1[] ו- f2[] שכל אחד מכיל חצי מהתוכן של המערך הגדול.

```
int main(){

   // reading the file

    FILE *ptr;
    char file[2455920];
    ptr = fopen("test1.txt","r");
    fread(&file, 1, 2455920, ptr);
    fclose(ptr);

    // spliting the file to two files
    char f1[1227960];
    char f2[1227960];
    for(int i = 0; i < 1227960; i++){
        f1[i] = file[i];
        f2[i] = file[1227960 + i];
    }
```

בשלב השני נפתח קשר TCP בין ה- Sender לבין ה- Receiver:

1. יצירת socket
2. אתחול כתובת ip ו- port עבור ה- socket באמצעות struct sockaddr_in:
   - שימוש ב- memeset על מנת לאפס בזיכרון את הבתים שיוקצו ל- server_address
   - שימוש ב- htons על מנת להמיר את ה- port number מ- HostBytesOrder ל- NetworkBytesOrder
   - שימוש ב- inet_pton על מנת להמיר את כתובת ה- ip מהצורה הטקסטואלית לצורה הבינארית.
3. התחברות אל ה- Receiver באמצעות הפונקציה connect.

```
// creat a socket
int sockfd = socket(AF_INET, SOCK_STREAM, 0);
if(sockfd == -1){
    printf("Eror in creating socket");
    exit(1);
}


// initialize an address for the socket
struct sockaddr_in server_address;
memset(&server_address, 0, sizeof(server_address));
server_address.sin_family = AF_INET;
server_address.sin_port = htons(8002);
inet_pton(AF_INET, "127.0.0.1", &server_address.sin_addr);


// create a connection
int connection_status =  connect(sockfd, (struct sockaddr *) &server_address, sizeof(server_address));
if(connection_status == -1){
    printf("Eror in connecting\n");
    exit(1);
}
printf("connection succsseed\n");
```

בשלב השלישי לאחר שהחיבור צלח נשלח את החצי הראשון של הקובץ ונבדוק שה- Receiver קיבל את המידע כמו שצריך ע"י שליחת אותנטיקציה חזרה מה- Receiver אל ה- Sender.

```
// Send the file

while(1){

    // send the first half of the file
    printf("sending the first file...\n");
    int send_status = send(sockfd, f1, sizeof(f1), 0);
    if(send_status == -1){
        printf("Eror in sending the file");
        exit(1);
    }


    // check the authentication from the receiver
    char authentication[17] = "0000010001010110";
    char receiver_response[17];
    int result = -1;

    while(result != 0){
        recv(sockfd, &receiver_response, sizeof(receiver_response), 0);
        result = strcmp(authentication, receiver_response);
    }

    printf("Sending of the first file has Succsseed\n");
```

בשלב הרביעי נשנה את האלגוריתם. קודם כל נבדוק מהו האלגוריתם הנוכחי ע"י הפונקציה getsockopt. במידה והוא reno נשנה אותו ל- cubic או הפוך ע"י שימוש בפונקציה setsockopt.

```
// change the cc algorithm
char reno[6] = "reno";
char algorithm[256];
socklen_t len;
len = sizeof(algorithm);


int get_status = getsockopt(sockfd, IPPROTO_TCP, TCP_CONGESTION, algorithm, &len);
if(get_status != 0){
    printf("Eror in getting the current cc algorithm\n");
    exit(1);
}

int ans = strcmp(algorithm, reno);

if(ans == 0){
    printf("Current algorithm: %s\n", algorithm);
    strcpy(algorithm, "cubic");
    int set_status = setsockopt(sockfd, IPPROTO_TCP, TCP_CONGESTION, algorithm, strlen(algorithm));
    if(set_status != 0){
        printf("Eror in changing the current cc algorithm\n");
        exit(1);
    }
    printf("The new algorithm is: cubic\n");

}else{
    printf("Current algorithm: %s\n", algorithm);
    strcpy(algorithm, "reno");
    int set_status = setsockopt(sockfd, IPPROTO_TCP, TCP_CONGESTION, algorithm, strlen(algorithm));
    if(set_status != 0){
        printf("Eror in changing the current cc algorithm\n");
        exit(1);
    }
    printf("The new algorithm is: reno\n");
}
```

בשלב החמישי נשלח את החצי השני של הקובץ. בשליחה זאת <u>לא נוודא</u> אותנטיקציה.

```
//send the second half of the file
printf("sending the second file...\n");
send_status = send(sockfd, f2, sizeof(f2), 0);
if(send_status == -1){
    printf("Eror in sending the file");
    exit(1);
}
printf("Sending of the second file has Succsseed\n");
```

בשלב השישי נשאל את המשתמש האם הוא מעוניין לשלוח את הקובץ שוב.
1 = שליחה נוספת. 0 = סגירה של התוכנית.

```
//asking the user
printf("Send the file again?\n");
printf("1 = Yes  ,  0 = No\n");
scanf("%d", &user_decision);
printf("the user choosed: %d\n", user_decision);
```

בשלב השביעי נפעל לפי התשובה של המשתמש. במידה והתשובה היא 1, נשלח הודעת עדכון (again) ל- Receiver לשליחה נוספת של הקובץ. נשנה בחזרה את האלגוריתם ונחזור שוב לשלב השלישי.

```
//send an update message to the receiver

if(user_decision == 1){
    char msg3[7] = "again.";
    send_status = send(sockfd, msg3, sizeof(msg3), 0);
    if(send_status == -1){
        printf("Eror in sending the file");
        exit(1);
    }
    printf("sending again...\n");

    //change the algorithm again
    get_status = getsockopt(sockfd, IPPROTO_TCP, TCP_CONGESTION, algorithm, &len);
    if(get_status != 0){
        printf("Eror in getting the current cc algorithm\n");
        exit(1);
    }

    ans = strcmp(algorithm, reno);

    if(ans == 0){
        printf("Current algorithm: %s\n", algorithm);
        strcpy(algorithm, "cubic");
        int set_status = setsockopt(sockfd, IPPROTO_TCP, TCP_CONGESTION, algorithm, strlen(algorithm));
        if(set_status != 0){
            printf("Eror in changing the current cc algorithm\n");
            exit(1);
        }
        printf("The new algorithm is: cubic\n");

    }else{
        printf("Current algorithm: %s\n", algorithm);
        strcpy(algorithm, "reno");
        int set_status = setsockopt(sockfd, IPPROTO_TCP, TCP_CONGESTION, algorithm, strlen(algorithm));
        if(set_status != 0){
            printf("Eror in changing the current cc algorithm\n");
            exit(1);
        }
        printf("The new algorithm is: reno\n");
    }
```

במידה והתשובה היא 0, נשלח הודעת יציאה (byebye) ל- Receiver, נצא מהלולאה ונסגור את ה- socket.

```
        }else if (user_decision == 0){
            char msg4[7] = "byebye";
            send_status = send(sockfd, msg4, sizeof(msg4), 0);
            if(send_status == -1){
                printf("Eror in sending the file");
                exit(1);
            }
            printf("bye bye!\n");
            break;
        }

    }

    //close the socket
    close(sockfd);

    return 0;

}
```

Receiver:

בשלב הראשון נפתח קשר TCP בין ה- Receiver לבין ה- Sender:

1. יצירת socket

2. אתחול כתובת ip ו- port עבור ה- socket באמצעות struct sockaddr_in:

- שימוש ב- memeset על מנת לאפס בזיכרון את הבתים שיוקצו ל- server_address

- שימוש ב- htons על מנת להמיר את ה- port number מ- HostBytesOrder ל- NetworkBytesOrder

- שימוש ב- inet_pton על מנת להמיר את כתובת ה- ip מהצורת הטקסטואלית לבינארית

4. חיבור בין ה- socket ל- server_address שיצרנו ע"י הפונקציה bind.
5. פתיחת האזנה לבקשות מצד אחר ע"י הפונקציה listen.

```
int main(){

    // creat a socket
    int sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if(sockfd == -1){
        printf("Eror in creating socket\n");
        exit(1);
    }

    // initialize an address for the socket
    struct sockaddr_in server_address;
    memset(&server_address, 0, sizeof(server_address));
    server_address.sin_family = AF_INET;
    server_address.sin_port = htons(8002);
    inet_pton(AF_INET, "127.0.0.1", &server_address.sin_addr);

    // create connection
    bind(sockfd, (struct sockaddr *) &server_address, sizeof(server_address));

    // Prepare to accept connections on socket FD
    int listen_status = listen(sockfd, 1);
    if(listen_status == -1){
        printf("Eror in getting connection");
        exit(1);
    }
    printf("listening on 127.0.0.0 : 8002 ...\n");
```

הקטע קוד הבא אינו מהווה שלב בתהליך אלא רק מגדיר משתני עזר שישרתו את הפעולות שיבואו בהמשך.

```
// tools for receving and for authentication//
char authentication[17] = "0000010001010110";
char msg3[7] = "byebye";


// tools for measuring time //
struct timespec begin, end;
long seconds;
long nanoseconds;
double elapsed;

// tools for saving the times //
double times[10];
int i = 0;
double sum1;
double sum2;
double average1;
double average2;

// tools for changing algorithm //
char reno[6] = "reno";
char algorithm[256];
socklen_t len;
len = sizeof(algorithm);
int ans;
```

בשלב השני נקבל באמצעות הפונקציה accept את ה- socket descriptor של השרת.

```
// wait for an incoming connection
int sender_socket;
sender_socket = accept(sockfd, NULL, NULL);
if(sender_socket == -1){
    printf("Eror in opennig new socket");
    exit(1);
}
```

בשלב השלישי נקבל את החצי הראשון של הקובץ ונמדוד כמה זמן לקח לקבל את כל המידע שלו. לאחר מכן נשמור את הזמן. על מנת לבדוק שאכן כל הבתים התקבלו, נגדיר שני משתני עזר:
count1- מחזיק את מספר הבתים שהתקבלו ברגע הנתון שמוחזר ע"י recv.
total1- מחזיק את סך מספר הבתים שהתקבלו עד עתה.

נעזר בלולאת while שתרוץ כל עוד סך הבתים שהתקבלו אינו שווה לגודל הקובץ הראשון כלומר ל- sizeof(buf1), וכך נדע שאכן הקובץ התקבל במלואו. על מנת שהמידע שיתקבל יישמר ב- buf1 כמו שצריך, בכל איטרציה של הלולאה נגדיר שהמידע יישמר ב- buf1[total1]. כמו כן מספר הבתים שנותר לקבל ישתנה בכל איטרציה ולכן נגדיר אותו – sizeof(buf1) total1. לצורך מדידת זמן הקבלה נשתמש בפונקציית clock_gettime.

```
while (1){


// Buffers for saving the data that will be sent
    char buf1[1227960];
    char buf2[1227960];
    char buf3[7];

   // veriables for calculate the number of bytes that been receivd
    int count1 = 0;
    int total1 = 0;


    // Start measuring time
    clock_gettime(CLOCK_REALTIME, &begin);

    while(total1 != sizeof(buf1)){
        count1 = recv(sender_socket, &buf1[total1], sizeof(buf1) - total1, 0);
        printf("Bytes received: %d\n", count1);
        total1 += count1;
        printf("Total Bytes received: %d\n", total1);
    // At this point the buffer is valid from 0..total-1, if that's enough then process it and break, otherwise continue
    }

    // Stop measuring time
    clock_gettime(CLOCK_REALTIME, &end);
    seconds = end.tv_sec - begin.tv_sec;
    nanoseconds = end.tv_nsec - begin.tv_nsec;
    elapsed = seconds + nanoseconds*1e-9;
    printf("Time measured for the first file: %.6f seconds.\n", elapsed);

    //save the time
    times[i] = elapsed;
    i++;
```

בשלב הרביעי נשלח אותנטיקציה ל- sender שמסמלת כי אכן הקובץ הראשון התקבל בהצלחה.

```
//send an authentication
int send_status = send(sender_socket, authentication, sizeof(authentication), 0);
if(send_status == -1){
    printf("Eror in sending the authentication");
    exit(1);
}
```

בשלב החמישי נשנה את האלגוריתם. קודם כל נבדוק מהו האלגוריתם הנוכחי ע"י הפונקציה getsockopt. במידה והוא reno נשנה אותו ל- cubic או ההיפך ע"י שימוש בפונקציה setsockopt.

```
// change the cc algorithm
int get_status = getsockopt(sockfd, IPPROTO_TCP, TCP_CONGESTION, algorithm, &len);
if(get_status != 0){
    printf("Eror in getting the current cc algorithm\n");
    exit(1);
}

int ans = strcmp(algorithm, reno);

if(ans == 0){
    printf("Current algorithm: %s\n", algorithm);
    strcpy(algorithm, "cubic");
    int set_status = setsockopt(sockfd, IPPROTO_TCP, TCP_CONGESTION, algorithm, strlen(algorithm));
    if(set_status != 0){
        printf("Eror in changing the current cc algorithm\n");
        exit(1);
    }
    printf("The new algorithm is: cubic\n");

}else{
    printf("Current algorithm: %s\n", algorithm);
    strcpy(algorithm, "reno");
    int set_status = setsockopt(sockfd, IPPROTO_TCP, TCP_CONGESTION, algorithm, strlen(algorithm));
    if(set_status != 0){
        printf("Eror in changing the current cc algorithm\n");
        exit(1);
    }
    printf("The new algorithm is: reno\n");
}
```

בשלב השישי נחזור על אותו תהליך של קבלת הקובץ השני בדיוק כמו בשלב השלישי.

```
// veriables for calculate the number of bytes that receivd
int count2 = 0;
int total2 = 0;


// Start measuring time
clock_gettime(CLOCK_REALTIME, &begin);

while(total2 != sizeof(buf2)){
    count2 = recv(sender_socket, &buf2[total2], sizeof(buf2) - total2, 0);
    printf("Bytes received: %d\n", count2);
    total2 += count2;
    printf("Total Bytes received: %d\n", total2);
// At this point the buffer is valid from 0..total-1, if that's enough then process it and break, otherwise continue
}

// Stop measuring time
clock_gettime(CLOCK_REALTIME, &end);
seconds = end.tv_sec - begin.tv_sec;
nanoseconds = end.tv_nsec - begin.tv_nsec;
elapsed = seconds + nanoseconds*1e-9;
printf("Time measured for the second file: %.6f seconds.\n", elapsed);

//save the time
times[i] = elapsed;
```

בשלב השביעי נחכה לקבלת הודעת עדכון מה- sender. לאחר שנתקבלה נשווה את תוכן ההודעה למחרוזת "byebye" שמשמעותה לצאת מהתוכנית.

```
printf("waiting for an update...\n");
//receive an update message
recv(sender_socket, &buf3, sizeof(buf3), 0);
printf("I got an update\n");
printf("%s\n", buf3);

int result = strcmp(buf3, msg3);
```

בשלב השמיני נפעל בהתאם להודעת העדכון שקיבלנו. אם המחרוזת שקיבלנו שווה למחרוזת "byebye" ז"א שהמשתמש בחר לצאת מהתוכנית ולכן נדפיס את כל הזמנים ששמרנו במערך []times לפי הסדר. לאחר מכן נחשב את סך הזמנים שלקח לכל חלק להתקבל:

sum1- מחזיק את סך הזמנים שלקח לחלק הראשון להתקבל.
sum2- מחזיק את סך הזמנים שלקח לחלק השני להתקבל.

נחשב את הזמן הממוצע שלקח לכל חלק להתקבל ונדפיס אותו. בסיום נצא מהלולאה ונסגור את ה- socket.

```
if(result == 0){
    for (int i = 0; i < 10; i++){
        printf("[%d]  %.6f seconds.\n", i, times[i]);
    }
    for (int i = 0; i < 10; i += 2){
        sum1 += times[i];
        sum2 += times[i+1];
    }
    average1 = sum1 / 5;
    average2 = sum2 / 5;
    printf("The average time for sending the first file is: %.6f seconds.\n", average1);
    printf("The average time for sending the second file is: %.6f seconds.\n", average2);
    break;
```

```
    }

    //close the socket
    close(sockfd);

    return 0;

}
```

אם המחרוזת שקיבלנו אינה שווה ל- "byebye" ז"א שהמשתמש בחר לשלוח פעם נוספת את הקובץ.
נבדוק מהו האלגוריתם הנוכחי ונשנה אותו, ולאחר מכן נמשיך לאיטרציה נוספת.

```
}else{
    // change the cc algorithm
    int get_status = getsockopt(sockfd, IPPROTO_TCP, TCP_CONGESTION, algorithm, &len);
     if(get_status != 0){
        printf("Eror in getting the current cc algorithm\n");
        exit(1);
    }

    int ans = strcmp(algorithm, reno);

     if(ans == 0){
        printf("Current algorithm: %s\n", algorithm);
        strcpy(algorithm, "cubic");
        int set_status = setsockopt(sockfd, IPPROTO_TCP, TCP_CONGESTION, algorithm, strlen(algorithm));
        if(set_status != 0){
            printf("Eror in changing the current cc algorithm\n");
            exit(1);
        }
        printf("The new algorithm is: cubic\n");

    }else{
        printf("Current algorithm: %s\n", algorithm);
        strcpy(algorithm, "reno");
        int set_status = setsockopt(sockfd, IPPROTO_TCP, TCP_CONGESTION, algorithm, strlen(algorithm));
        if(set_status != 0){
            printf("Eror in changing the current cc algorithm\n");
            exit(1);
        }
        printf("The new algorithm is: reno\n");
    }

}
```

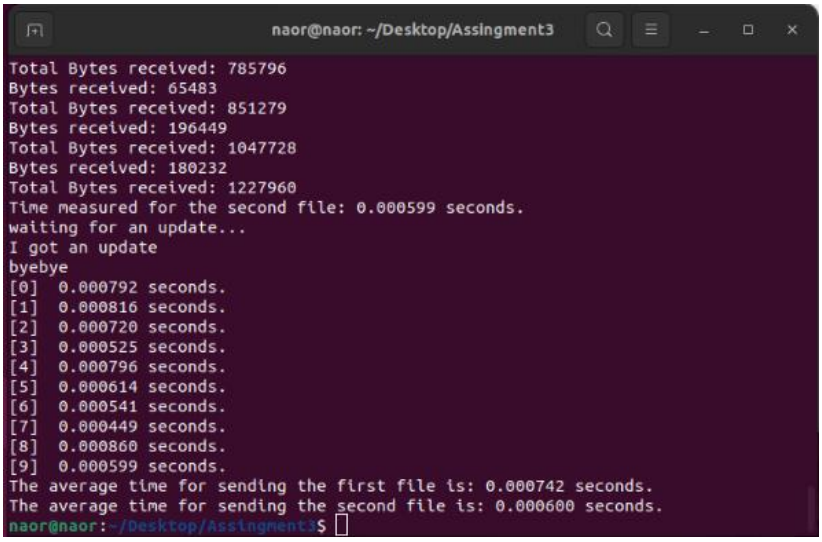
חלק ב - מחקר

הקדמה:

בחלק זה של המטלה נתבקשנו לערוך מחקר בעזרת התוכנה שבנינו כדי לבחון את השוני בין האלגוריתמים שלמדנו- cubic ו- reno. את המחקר ערכנו במערכת הפעלה Ubuntu 22.04. נעזרנו בכלי tc לאיבוד פקטות ברשת כאשר בכל איטרציה שינינו את אחוז איבוד הפקטות- 0%, 10%, 15%, 20%, 25%.

הרצה מספר 1- 0% איבוד פקטות:

הרצנו את התוכנית 5 פעמים ברצף ולאחר מכן יצאנו.



ממצאים:

הזמן הממוצע שלקח לקבל את החלק הראשון באמצעות שימוש באלגוריתם cubic היה מעט גבוה יותר מהזמן הממוצע שלקח לקבל את החלק השני באמצעות שימוש באלגוריתם reno.

wireshark:

לא ראינו איבוד של פקטות.

<u>הרצה מספר 2- 10% איבוד פקטות:</u>

הרצנו את התוכנית 5 פעמים ברצף ולאחר מכן יצאנו.

```
naor@naor: ~/Desktop/Assingment3
Total Bytes received: 392898
Bytes received: 65483
Total Bytes received: 458381
Bytes received: 654830
Total Bytes received: 1113211
Bytes received: 114749
Total Bytes received: 1227960
Time measured for the second file: 0.224689 seconds.
waiting for an update...
I got an update
byebye
[0]  0.207661 seconds.
[1]  0.000976 seconds.
[2]  0.218570 seconds.
[3]  0.090430 seconds.
[4]  0.021126 seconds.
[5]  0.238767 seconds.
[6]  0.248377 seconds.
[7]  0.001124 seconds.
[8]  0.058648 seconds.
[9]  0.224689 seconds.
The average time for sending the first file is: 0.150876 seconds.
The average time for sending the second file is: 0.111197 seconds.
naor@naor:~/Desktop/Assingment3$
```

ממצאים:

הזמן הממוצע שלקח לקבל את החלק הראשון באמצעות שימוש באלגוריתם cubic היה מעט גבוה יותר מהזמן הממוצע שלקח לקבל את החלק השני באמצעות שימוש באלגוריתם reno.

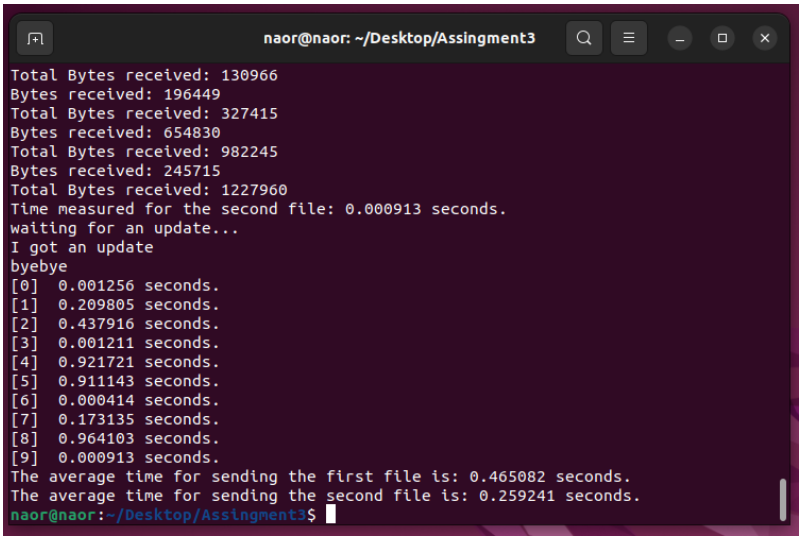
wireshark:

בתמונה ניתן לראות בבירור כי אכן יש איבוד של פקטות, לפי ההודעות מסוג TCP Dup ACK בשורות 65, 66 ולפי ההודעה מסוג TCP Retransmition בשורה 68.

53	0.002147161	127.0.0.1	127.0.0.1	TCP	32836	54042 → 8002 [PSH, ACK] Seq=1048172 Ack=
54	0.002162817	127.0.0.1	127.0.0.1	TCP	32836	54042 → 8002 [ACK] Seq=1080940 Ack=1 Wir
55	0.002181061	127.0.0.1	127.0.0.1	TCP	32836	54042 → 8002 [PSH, ACK] Seq=1113708 Ack=
56	0.002221699	127.0.0.1	127.0.0.1	TCP	68	8002 → 54042 [ACK] Seq=1 Ack=917100 Win=
57	0.002305288	127.0.0.1	127.0.0.1	TCP	32836	54042 → 8002 [ACK] Seq=1146476 Ack=1 Wir
58	0.002343910	127.0.0.1	127.0.0.1	TCP	80	[TCP Window Update] 8002 → 54042 [ACK] S
59	0.002368954	127.0.0.1	127.0.0.1	TCP	32836	54042 → 8002 [PSH, ACK] Seq=1179244 Ack=
60	0.002387145	127.0.0.1	127.0.0.1	TCP	80	[TCP Window Update] 8002 → 54042 [ACK] S
61	0.002414412	127.0.0.1	127.0.0.1	TCP	32836	[TCP Out-Of-Order] 54042 → 8002 [PSH, AC
62	0.002434436	127.0.0.1	127.0.0.1	TCP	80	[TCP Window Update] 8002 → 54042 [ACK] S
63	0.002447792	127.0.0.1	127.0.0.1	TCP	80	[TCP Window Update] 8002 → 54042 [ACK] S
64	0.002470888	127.0.0.1	127.0.0.1	TCP	32836	[TCP Out-Of-Order] 54042 → 8002 [ACK] Se
65	0.002496496	127.0.0.1	127.0.0.1	TCP	80	[TCP Dup ACK 56#1] 8002 → 54042 [ACK] Se
66	0.002508871	127.0.0.1	127.0.0.1	TCP	80	[TCP Dup ACK 56#2] 8002 → 54042 [ACK] Se
67	0.002574904	127.0.0.1	127.0.0.1	TCP	80	8002 → 54042 [ACK] Seq=1 Ack=949868 Win=
68	0.208306918	127.0.0.1	127.0.0.1	TCP	32836	[TCP Retransmission] 54042 → 8002 [ACK]
69	0.208338998	127.0.0.1	127.0.0.1	TCP	80	8002 → 54042 [ACK] Seq=1 Ack=1212012 Wir
70	0.208351977	127.0.0.1	127.0.0.1	TCP	16017	54042 → 8002 [PSH, ACK] Seq=1212012 Ack=
71	0.208361179	127.0.0.1	127.0.0.1	TCP	68	8002 → 54042 [ACK] Seq=1 Ack=1227961 Wir
72	0.208430431	127.0.0.1	127.0.0.1	TCP	85	8002 → 54042 [PSH, ACK] Seq=1 Ack=122796
73	0.208433277	127.0.0.1	127.0.0.1	TCP	68	54042 → 8002 [ACK] Seq=1227961 Ack=18 Wi
74	0.208619821	127.0.0.1	127.0.0.1	TCP	65551	54042 → 8002 [ACK] Seq=1227961 Ack=18 Wi
75	0.208653778	127.0.0.1	127.0.0.1	TCP	65551	54042 → 8002 [ACK] Seq=1293444 Ack=18 Wi
76	0.208671235	127.0.0.1	127.0.0.1	TCP	68	8002 → 54042 [ACK] Seq=18 Ack=1358927 Wi
77	0.208679934	127.0.0.1	127.0.0.1	TCP	65551	54042 → 8002 [ACK] Seq=1358927 Ack=18 Wi
78	0.208908230	127.0.0.1	127.0.0.1	TCP	65551	54042 → 8002 [ACK] Seq=1424410 Ack=18 Wi
79	0.208915994	127.0.0.1	127.0.0.1	TCP	68	8002 → 54042 [ACK] Seq=18 Ack=1489893 Wi

<u>הרצה מספר 3- 15% איבוד פקטות:</u>

הרצנו את התוכנית 5 פעמים ברצף ולאחר מכן יצאנו.



ממצאים:

הזמן הממוצע שלקח לקבל את החלק הראשון באמצעות שימוש באלגוריתם cubic היה גבוה ב- 0.2 שניות יותר מהזמן הממוצע שלקח לקבל את החלק השני באמצעות שימוש באלגוריתם reno.

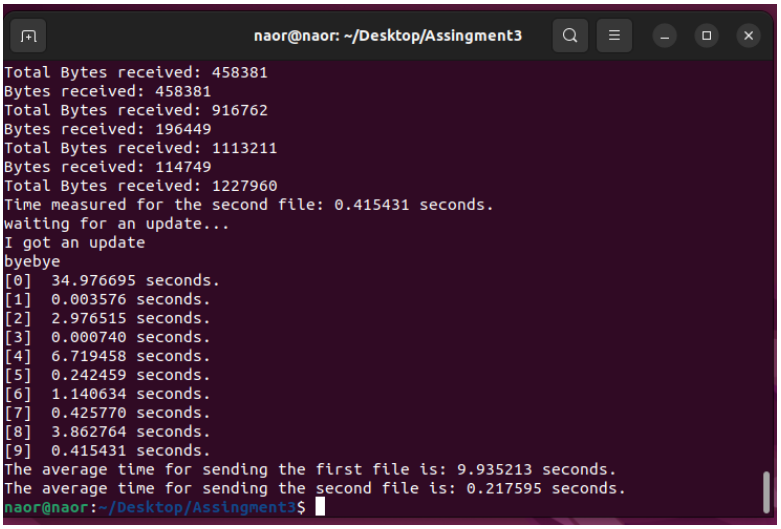
wireshark:

בתמונה ניתן לראות בבירור כי אכן יש איבוד של פקטות, לפי ההודעות מסוג TCP Dup ACK בשורות 335, 337 ולפי ההודעה מסוג TCP Retransmition בשורה 338.

331	33.815325923	127.0.0.1	127.0.0.1	TCP	65551	38798 → 8002 [ACK] Seq=11575533 Ack=86
332	33.815374561	127.0.0.1	127.0.0.1	TCP	80	8002 → 38798 [ACK] Seq=86 Ack=11444567
333	33.815385624	127.0.0.1	127.0.0.1	TCP	65551	38798 → 8002 [ACK] Seq=11641016 Ack=86
334	33.815392892	127.0.0.1	127.0.0.1	TCP	65551	38798 → 8002 [ACK] Seq=11706499 Ack=86
335	33.815406430	127.0.0.1	127.0.0.1	TCP	80	[TCP Dup ACK 332#1] 8002 → 38798 [ACK]
336	33.815416347	127.0.0.1	127.0.0.1	TCP	65551	38798 → 8002 [ACK] Seq=11771982 Ack=86
337	33.815425245	127.0.0.1	127.0.0.1	TCP	80	[TCP Dup ACK 332#2] 8002 → 38798 [ACK]
338	33.815433903	127.0.0.1	127.0.0.1	TCP	65551	[TCP Fast Retransmission] 38798 → 8002
339	33.815478995	127.0.0.1	127.0.0.1	TCP	68	8002 → 38798 [ACK] Seq=86 Ack=11837465
340	33.815494013	127.0.0.1	127.0.0.1	TCP	65551	38798 → 8002 [ACK] Seq=11837465 Ack=86
341	33.815503312	127.0.0.1	127.0.0.1	TCP	65551	38798 → 8002 [ACK] Seq=11902948 Ack=86
342	33.815512687	127.0.0.1	127.0.0.1	TCP	65551	38798 → 8002 [ACK] Seq=11968431 Ack=86
343	33.815617960	127.0.0.1	127.0.0.1	TCP	68	8002 → 38798 [ACK] Seq=86 Ack=12033914
344	33.815628739	127.0.0.1	127.0.0.1	TCP	65551	38798 → 8002 [ACK] Seq=12033914 Ack=86

<u>הרצה מספר 4- 20% איבוד פקטות:</u>

הרצנו את התוכנית 5 פעמים ברצף ולאחר מכן יצאנו.



ממצאים:

הזמן הממוצע שלקח לקבל את החלק הראשון באמצעות שימוש באלגוריתם cubic היה גבוה ב- 9 שניות יותר מהזמן הממוצע שלקח לקבל את החלק השני באמצעות שימוש באלגוריתם reno.

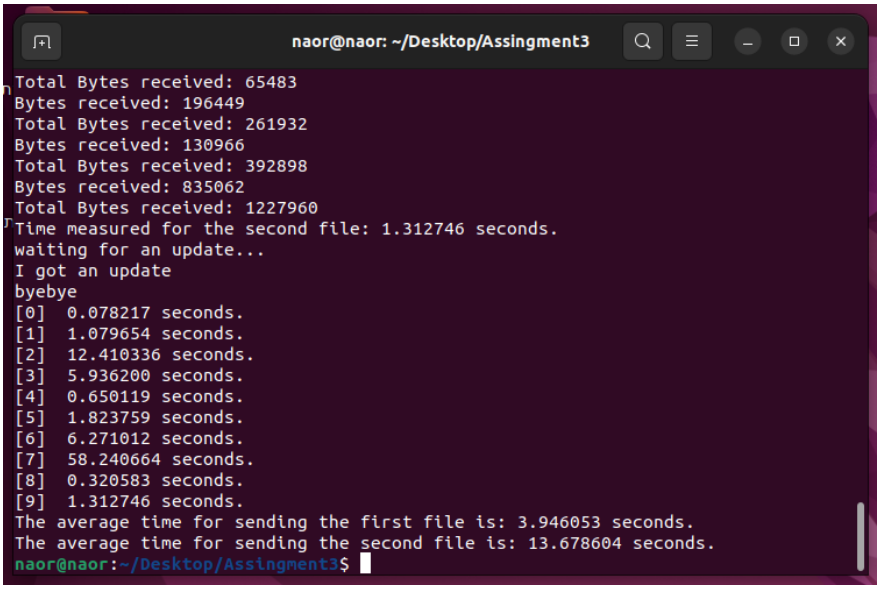
wireshark:

בתמונה ניתן לראות בבירור כי אכן יש איבוד של פקטות, לפי ההודעות מסוג TCP Dup ACK בשורות 55, 56 ולפי ההודעה מסוג TCP Retransmition בשורה 57.

51	8.353321023	127.0.0.1	127.0.0.1	TCP	32809	57116 → 8002 [PSH, ACK] Seq=818526 Ack=1
52	8.353342989	127.0.0.1	127.0.0.1	TCP	32809	57116 → 8002 [ACK] Seq=851267 Ack=1 Win=6
53	8.353363208	127.0.0.1	127.0.0.1	TCP	32809	57116 → 8002 [PSH, ACK] Seq=884008 Ack=1
54	8.353402667	127.0.0.1	127.0.0.1	TCP	32809	[TCP Out-Of-Order] 57116 → 8002 [ACK] Seq
55	8.353446416	127.0.0.1	127.0.0.1	TCP	80	[TCP Dup ACK 49#2] 8002 → 57116 [ACK] Seq
56	8.353468647	127.0.0.1	127.0.0.1	TCP	80	[TCP Dup ACK 49#3] 8002 → 57116 [ACK] Seq
57	11.424299460	127.0.0.1	127.0.0.1	TCP	32809	[TCP Retransmission] 57116 → 8002 [ACK] S
58	11.424612323	127.0.0.1	127.0.0.1	TCP	80	8002 → 57116 [ACK] Seq=1 Ack=916749 Win=2
59	11.424700567	127.0.0.1	127.0.0.1	TCP	65550	57116 → 8002 [PSH, ACK] Seq=916749 Ack=1

<u>הרצה מספר 5- 25% איבוד פקטות:</u>

הרצנו את התוכנית 5 פעמים ברצף ולאחר מכן יצאנו.



ממצאים:

הזמן הממוצע שלקח לקבל את החלק הראשון באמצעות שימוש באלגוריתם cubic היה נמוך ב- 10 שניות פחות מהזמן הממוצע שלקח לקבל את החלק השני באמצעות שימוש באלגוריתם reno.

wireshark:

בתמונה ניתן לראות בבירור כי אכן יש איבוד של פקטות, לפי ההודעות מסוג TCP Dup ACK בשורות 121, 123 ולפי ההודעה מסוג TCP Retransmition בשורה 124.

118	29.376372872	127.0.0.1	127.0.0.1	TCP	65551	52302 → 8002 [ACK] Seq=2914309 Ack=18 Win=65536
119	29.376453308	127.0.0.1	127.0.0.1	TCP	68	8002 → 52302 [ACK] Seq=18 Ack=2979792 Win=29474
120	29.376513216	127.0.0.1	127.0.0.1	TCP	65551	[TCP Previous segment not captured] 52302 → 800
121	29.376644172	127.0.0.1	127.0.0.1	TCP	80	[TCP Dup ACK 119#1] 8002 → 52302 [ACK] Seq=18 A
122	29.376694783	127.0.0.1	127.0.0.1	TCP	65551	52302 → 8002 [ACK] Seq=3110758 Ack=18 Win=65536
123	29.376726819	127.0.0.1	127.0.0.1	TCP	80	[TCP Dup ACK 119#2] 8002 → 52302 [ACK] Seq=18 A
124	29.376783501	127.0.0.1	127.0.0.1	TCP	65551	[TCP Fast Retransmission] 52302 → 8002 [ACK] Se
125	29.376827564	127.0.0.1	127.0.0.1	TCP	68	8002 → 52302 [ACK] Seq=18 Ack=3176241 Win=28485
126	29.376874996	127.0.0.1	127.0.0.1	TCP	65551	52302 → 8002 [ACK] Seq=3176241 Ack=18 Win=65536
127	29.376912280	127.0.0.1	127.0.0.1	TCP	65551	52302 → 8002 [ACK] Seq=3241724 Ack=18 Win=65536

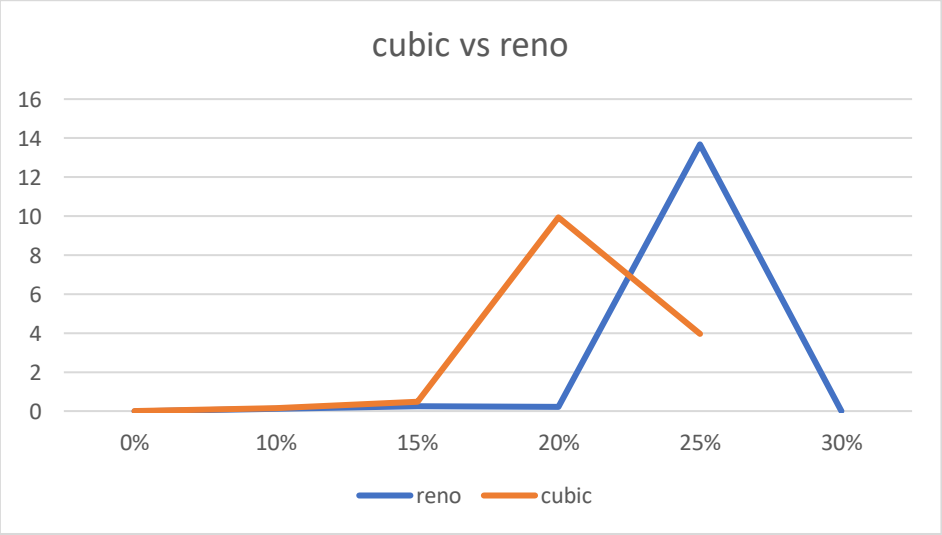
מסקנות:

ראשית הבחנו כי אם נריץ את התוכנה מספר פעמים ברצף אנחנו עשויים לקבל תוצאות שונות בזמנים וזאת כיוון שהכלי לאיבוד פקטות בו נעזרנו- tc משנה את חלון האיבוד באופן שאינו צפוי מראש.

למרות זאת לאור התוצאות שדגמנו שמנו לב שאכן קיים שוני בין האלגוריתמים. בעוד שהאלגוריתם של reno מגדיל את חלון השליחה שלו באופן איטי- ליניארי, cubic מגדיל את חלון השליחה באופן מהיר יותר ולפיכך יכול להגיע מהר יותר מ- reno לסף האופטימלי של קצב שליחה.

אם נסתכל על הגרף נוכל להבחין בעקומה של reno שטיפסה מהר יותר בשל איבוד הפקטות הפתאומי.
במידה והיינו ממשיכים לדגום מספר רב של פעמים בקצב איבוד של 25% ומחשבים את הממוצע ככל הנראה שהעקומה של reno הייתה מטפסת גבוה יותר.

Percentage of lost	Cubic average time	Reno average time
0%	0.000742 sec	0.000600 sec
10%	0.150 sec	0.111 sec
15%	0.465 sec	0.259 sec
20%	9.935 sec	0.217 sec
25%	3.946 sec	13.678 sec
30%	-	-



ביבליוגרפיה

/https://pandorafms.com/blog/tcp-congestion-control

https://witestlab.poly.edu/blog/tcp-congestion-control-basics

https://www.youtube.com/watch?v=rib_ujnMqcs&t=326s