מטלה 3
רשתות תקשורת

מגישים:
אילן שמחון ת.ז 212036396
תומר גוזלן ת.ז 314770058

הקדמה - מהות המטלה

מטרתנו במטלה זו הינה לחקור את איכות האלגוריתמים reno וcubic אשר מטרתם הינה לנהל את זרימת הנתונים בחיבורי TCP ולוודא התנהלות תקינה של העברת נתונים, תוך התמודדות עם בעיות התעכבות ואיבוד מידע
ברצוננו לענות על השאלה איזה מהם איכותי מהיר יותר, נסביר כיצד נבדוק זאת:

הקוד בנוי בשני קבצים: sender.py , receiver.py

תחילה ניצור חיבור client-server בין השולח של הקובץ לבין המקבל שלו
לאחר שנוצר חיבור בין הsender ל receiver הsender שולח לו קובץ טקסט (בגודל מעל MB2) בשני חלקים שונים כאשר בחלק הראשון מופעל אלגוריתם reno ואילו בחלק השני מופעל אלגוריתם cubic, הreceiver שומר את הזמנים שלקח לו לקבל את כל אחד מהחלקים על מנת לראות בעזרת איזה אלגוריתם זה נעשה יותר מהר

אנו נריץ את התכנית מספר פעמים כאשר בכל פעם נשנה את אחוז איבוד הפאקטות (בים 0 ל 20 אחוז) ונבחן את התנהגות התוכנה בתנאים אלו ובעיקר את זמן הריצה שלוקח לכל אלגוריתם

חלק א' – מהלך הקוד

נעבור כעת על מהלך הקוד בצורה משולבת בין 2 הקבצים ונראה מה קורה בכל קובץ במקביל

1) הגדרת משתנים ופונקציות שישמשו אותנו בתכנית:

בשלב זה נגדיר משתנים בשביל בדיקת התקינות, נפתח את הfile בכל אחד מהקבצים במצב המתאים ונגדיר משתנים ליצירת החיבור (ת.ז מספר 2 נגמרת ב0058 ולכן זה מוגדר להיות רק 58)

receiver.py

```python
import socket
import time
from statistics import mean

ID1 = 6396
ID2 = 58

file_path = "receiver_file.txt"
file = open(file_path, 'wb')

part1_time = []
part2_time = []

receiver_ip = '127.0.0.1'
receiver_port = 9999
```

Sender.py

```python
import socket
import os

ID1 = 6396
ID2 = 58

2 usages
def check_got(xor):
    bool = (xor == (ID1 ^ ID2))
    return bool

file_path = "myFile1.txt"
file = open(file_path, "r")

file_len = os.path.getsize(file_path)
half_len = file_len // 2

receiver_ip = '127.0.0.1'
receiver_port = 9999
```

2) יצירת חיבור בין הsender לrecreiver וקריאת הקובץ

בחלק זה ניצור חיבור TCP בין הsender לבין הreceiver, הreceiver יוצר חיבור:

```python
receiver_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
buffer_size = receiver_socket.getsockopt(socket.SOL_SOCKET, socket.SO_RCVBUF)

print("Creating a connection...")
receiver_socket.bind((receiver_ip,receiver_port))

receiver_socket.listen(1)
print("listening on IP:",receiver_ip," Port:",receiver_port)
```

הsender מתחבר אל החיבור שיצר הreceiver

```
sender_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
buffer_size = sender_socket.getsockopt(socket.SOL_SOCKET, socket.SO_RCVBUF)

print("Connecting to receiver...")
sender_socket.connect((receiver_ip, receiver_port))
```

הreceiver מאשר את בקשת החיבור של הsender והחיבור מושלם:

```
sender_socket, sender_address = receiver_socket.accept()
print("connection Succeeded!")
```

הsender מחלק את הקובץ ל2 חלקים ושולח לreceiver את גודל הקובץ:

```
file_read = file.read()

part1 = file_read[:half_len]
part2 = file_read[half_len:]

file.close()
print("Sending the len of the file")
sender_socket.send(str(file_len).encode())
```

הreceiver מקבל גודל הקובץ מהsender:

```
file_len = int(sender_socket.recv(buffer_size))
```

3) כעת ניכנס ללולאה הראשית בה נשלח בכל איטרציה את הקובץ פעם אחת
הsender מגדיר אצלו את האלגוריתם CC להיות reno (עבור החלק הראשון בלבד) ואז שולח את החלק הראשון לreceiver:

```
send = True
while send is True:

    print("Defines the CC algorithm be Reno.")
    sender_socket.setsockopt(socket.IPPROTO_TCP, socket.TCP_CONGESTION, b'reno')


    print("Sending the first part of the file..")
    sender_socket.send(part1.encode())
```

שליחת
החלק הראשון

במקביל, הreceiver מגדיר גם אצלו את האלגוריתם CC להיות reno ואז מקבל בלולאה את החלק הראשון של הקובץ מהsender, כשהוא מסיים לקרוא את החלק הראשון הוא מחשב את הזמן שזה לקח ושומר אותו בlist , לאחר מכן הוא שולח לreceiver אישור שהקובץ התקבל בהצלחה ע"י שליחת xor של 2 הת"ז כפי שנדרש (ת.ז מספר 2 נגמרת ב0058 ולכן זה מוגדר להיות רק 58)

קבלת החלק הראשון

שליחת אימות

```
while file_len != -1:
    print("Defines the CC algorithm be reno.")
    receiver_socket.setsockopt(socket.IPPROTO_TCP, socket.TCP_CONGESTION, b'reno')

    start_time = time.time()

    data_size1 = 0
    print("receives the first part of the file")
    print(file_len // 2)
    while data_size1 < file_len // 2:
        chunk = sender_socket.recv(buffer_segment)
        data_size1 += len(chunk.decode())
        file.write(chunk)

    first_part_time = time.time() - start_time
    part1_time.append(first_part_time)

    print("Sending OK to the sender")
    sender_socket.send((str(ID1 ^ ID2)).encode())
```

כעת, הsender מקבל מהreceiver את התוצאה של הxor ושולח אותה לפונקצייה שבודקת האם זה מה שהיה צריך להתקבל, אם זה שגוי הוא יזרק שגיאה

בדיקת אימות

```
    xor = float(sender_socket.recv(buffer_size).decode())
    if check_got(xor) is False:
        Exception
    else:
        print("Succeeded!")
```

ובכן, סיימנו לשלוח בהצלחה את החלק הראשון ונעבור לשלוח את החלק השני
נשנה את האלגוריתם CC להיות cubic
לאחר מכן נשלח את החלק השני ואז נקבל מהreceiver שוב את האות המוסכם שמאשר כי המידע הנכון הגיע אליו :

שליחת החלק השני

בדיקת אישור

```
print("Defines the CC algorithm be Cubic")
sender_socket.setsockopt(socket.IPPROTO_TCP, socket.TCP_CONGESTION, b'cubic')

print("Sending the second part of thr file..")
sender_socket.send(part2.encode())

xor = float(sender_socket.recv(buffer_size).decode())
if check_got(xor) is False:
    raise Exception
```

במקביל יתרחש תהליך זהה אצל הreceiver, הוא יקבל בלולאה את החלק השני של הקובץ מהsender,
לאחר שיסיים ישמור את הזמנים ויישלח הודעת אישור לsender

שליחת אישור

```
while data_size2 < file_len - data_size1:
    chunk = sender_socket.recv(buffer_size)
    data_size2 += len(chunk)
    file.write(chunk)

second_part_time = time.time() - start_time
part2_time.append(second_part_time)

print("Sending OK to the sender")
sender_socket.send((str(ID1 ^ ID2)).encode())
```

כעת, על המשתמש לקבל החלטה האם לשלוח את הקובץ שוב, או לסגור את החיבור ולסיים את התכנית.
הsender שואל את המשתמש שמחליט האם להמשיך ושולח את התשובה לreceiver.
במידה והוחלט להמשיך, הsender חוזר לתחילת הלולאה. במידה והוחלט לסיים הוא יוצא מהלולאה וסוגר את החיבור.

סגירת הלולאה להמשך

הבלת הודעת מה-receiver

```
again = input("Should you send the file again? (y/n):")
if again == "n":
    send = False
    sender_socket.send("Stop sending the file".encode())
    sender_socket.recv(buffer_size).decode()
else:
    sender_socket.send("Sending the file again!".encode())

sender_socket.close()
```

במקביל אצל הreceiver, הוא מקבל את תוצאת החלטת המשתמש מהsender האם הוא מקבל את הקובץ שוב או לא.
אם הוחלט להמשיך ולשלוח שוב את הקובץ, הוא חוזר לתחילת הלולאה ומתחיל לקלוט את הקובץ מההתחלה

אחרת, הוא יוצר מהלולאה ומדפיס את כל הזמנים שלקח לו לקבל את הקובץ מהsender, כולל ממוצע לכל חלק בנפרד וכולל ממוצע כללי

לאחר מכן, הreceiver סוגר את החיבור לsender ואת הקובץ שאליו כתב.

קבלת ההחלטה

גורם לסיום הלולאה

```
    decision = sender_socket.recv(buffer_size).decode()
    print(decision)
    if decision == "Stop sending the file":
        file_len = -1
        sender_socket.send("Thank you, goodbye!".encode())

for i in range(1, len(part1_time)+1):
    print("Time of receiving number ", i, ": ")
    print("part 1: ", part1_time[i-1], " seconds")
    print("part 2: ", part2_time[i-1], " seconds")

print("Average of part 1:", mean(part1_time), "seconds")
print("Average of part 2:", mean(part2_time), "seconds")
print("Total Average: ", mean(part1_time+part2_time), "seconds")
print("Close the connection..")
receiver_socket.close()
file.close()
```

הדפסת סטטיסטיקה

סגירת חיבור וקבצ

חלק ב – הרצה כללית

בחלק זה נריץ את הקוד בצורה רגילה ללא איבוד פאקטות על מנת להראות את ההתנהלות התקינה של ההרצה ואת ההדפסות במהלך הריצה, במהלך ההרצה נבקש מהsender לשלוח 2 פעמים את הקובץ כיוון שהיא מיועדת רק על מנת להמחיש את התהליך

נריץ את ה receiver

```
(venv) ilan@ilan-Latitude-5420:~/PycharmProjects/pythonProject$ python3 receiver.py
Creating a connection...
listening on IP: 127.0.0.1  Port: 9999
connection Succeeded!

Defines the CC algorithm be reno.
receives the first part of the file
Sending OK to the sender

Defines the CC algorithm be Cubic..
receives the second part of the file
Sending OK to the sender
```

קבלת חלק ראשון

קבלת חלק שני

בתמונה זו אנו רואים חיבור ושליחה ראשונה של הקובץ, כעת הreceiver מחכה לעדכון מהsender האם הוא שולח פעם נוספת את הקובץ או לא

נריץ את הsender

```
(venv) ilan@ilan-Latitude-5420:~/PycharmProjects/pythonProject$ python3 sender.py
Connecting to receiver...
connection Succeeded!

Sending the len of the file

Defines the CC algorithm be Reno.
Sending the first part of the file..
Succeeded!

Defines the CC algorithm be Cubic
Sending the second part of thr file..
Succeeded!

Should you send the file again? (y/n):
```

שליחת חלק ראשון

שליחת חלק שני

הודעת המשתמש

בתמונה זו אנו רואים את מהלך החיבור של הsender לreceiver ואת שליחה הקובץ בפעם הראשונה, לבסוף שואלת התכנית את המשתמש האם לשלוח את הקובץ שוב, במידה והמשתמש בוחר לשלוח שוב התכנית תשלח את הקובץ פעם נוספת עד שהמשתמש יחליט לעצור

הצעה על שיחה חוצרת

נציג בקשה לשלוח את הקובץ פעם נוספת:

```
Should you send the file again? (y/n):y

Defines the CC algorithm be Reno.
Sending the first part of the file..
Succeeded!

Defines the CC algorithm be Cubic
Sending the second part of thr file..
Succeeded!

Should you send the file again? (y/n):n

Close the connection..
```

Sender.py

```
Sending the file again!

Defines the CC algorithm be reno.
receives the first part of the file
Sending OK to the sender

Defines the CC algorithm be Cubic..
receives the second part of the file
Sending OK to the sender

Stop sending the file

Time of receiving number  1 :
part 1:  0.0029425621032714844  seconds
part 2:  0.0014846324920654297  seconds

Time of receiving number  2 :
part 1:  0.004225969314575195  seconds
part 2:  0.003233194351196289  seconds

Average of part 1: 0.00358426570892334 seconds
Average of part 2: 0.0023589134216308594 seconds
Total Average:  0.00297158956527709 96 seconds

Close the connection..
(venv) ilan@ilan-Latitude-5420:~/PycharmProjects/pythonProject$
```

receiver.py

בתמונה הימנית, רואים את התכנית שואלת את המשתמש של הsender האם לשלוח את הקובץ פעם נוספת, המשתמש עונה בחיוב ולכן התכנית מעדכנת את הreceiver ואכן רואים בתמונה השמאלית את הreceiver מדפיס הודעה שהקובץ עומד להישלח שוב וכן ניתן לראות גם בsender וגם בreceiver את תהליך שליחת הקובץ שוב, לאחר מכן התכנית שואלת את המשתמש האם לשלוח שוב (שלחנו את הקובץ כבר פעמיים כמו שרצינו בדוגמא זו ולכן נענה בשלילה) ואכן רואים את הsender ואת הreceiver מדפיסים הודעה על סגירת החיבור וסיום התכנית.

בנוסף, לאחר ההודעה מהsender על סגירת החיבור הreceiver מדפיס את כל הזמנים וסיכום הנתונים.

חלק ג – איבוד פאקטות

בחלק זה נבחן את ההבדלים שנגרמים בהרצת הקוד כתוצאה מאיבוד של פאקטות בתהליך התקשורת, לפי הנתונים שנקבל תחת הרצות אלו נוכל לקבוע איזה אלגוריתם טוב יותר

אנו מריצים את הקוד 4 פעמים, בכל פעם עם אחוז איבוד פאקטות שונה בהתאם להוראות המטלה (20%,15%,10%,0%)

כזכור, בהרצה יש למשתמש אפשרות לבחור האם לשלוח את הקובץ שוב או לסגור את ההתחברות, בכל הרצה נבחר לשלוח את הקובץ 5 פעמים על מנת לקבל נתונים מספיק מבוססים בנוגע לזמן השליחה על פי כל אלגוריתם ולהבדלים שביניהם

נריץ את הקוד, נציג את הזמנים שקיבלנו ונוסיף צילומי מסך רלוונטיים מהקלטות ה wireshark

הרצה ראשונה – ללא איבוד פאקטות

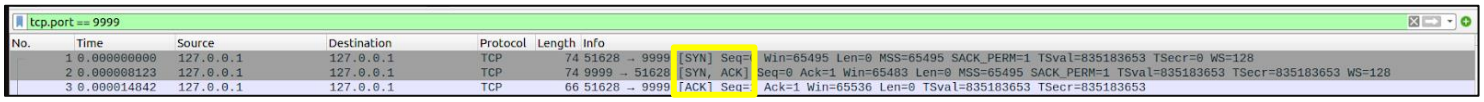
בהדפסת הזמנים נקבל אינדיקציה ראשונית כמה זמן לוקח לתכנית לרוץ ואיזה אלגוריתם הוא טוב יותר ללא ההתערבות שלנו באובדן פאקטות

ובכן כפי שניתן לראות בצילום ההרצה, מהנתונים שאנו מקבלים לאחר שליחת הקובץ 5 פעמים, מהירות השליחה באמצעות כל אחד מהאלגוריתמים קרובה מאוד, עם יתרון קל לאלגוריתם cubic

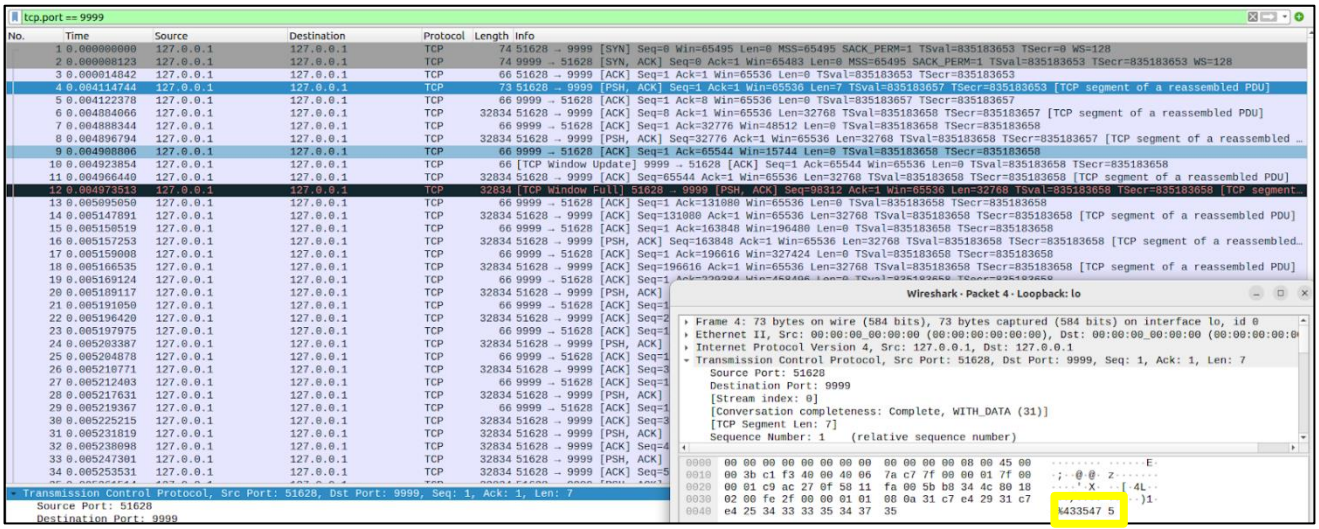
```
Stop sending the file

Time of receiving number  1 :
part 1:  0.002384185791015625  seconds
part 2:  0.0011560916900634766  seconds

Time of receiving number  2 :
part 1:  0.004258155822753906  seconds
part 2:  0.003230571746826172  seconds

Time of receiving number  3 :
part 1:  0.0036559104919433594  seconds
part 2:  0.0033414363861083984  seconds

Time of receiving number  4 :
part 1:  0.0038270950317382812  seconds
part 2:  0.003756284713745117  seconds

Time of receiving number  5 :
part 1:  0.005266427993774414  seconds
part 2:  0.003618001937866211  seconds

Average of part 1: 0.003878355026245117 seconds
Average of part 2: 0.003020477294921875 seconds
Total Average:  0.003449416160583496 seconds

Close the connection..
(venv) ilan@ilan-Latitude-5420:~/PycharmProjects/pythonProject$
```

נתבונן בהקלטות הwireshark

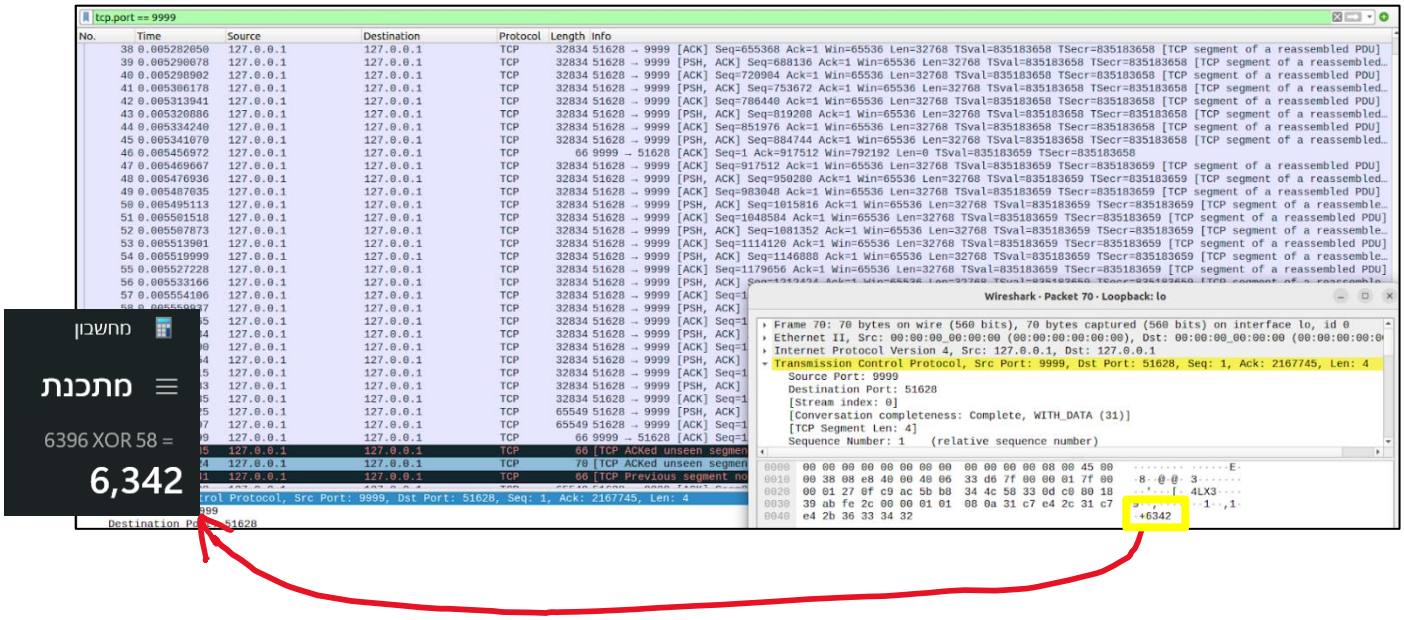
נראה בתחילה את הפאקטות המעידות על יצירת חיבור בין הsender לreceiver , ניתן לזהות אותם לפי סוגם, פאקטות SYN המעידות על בקשה לחיבור ופאקטת ACK המאשרת את החיבור



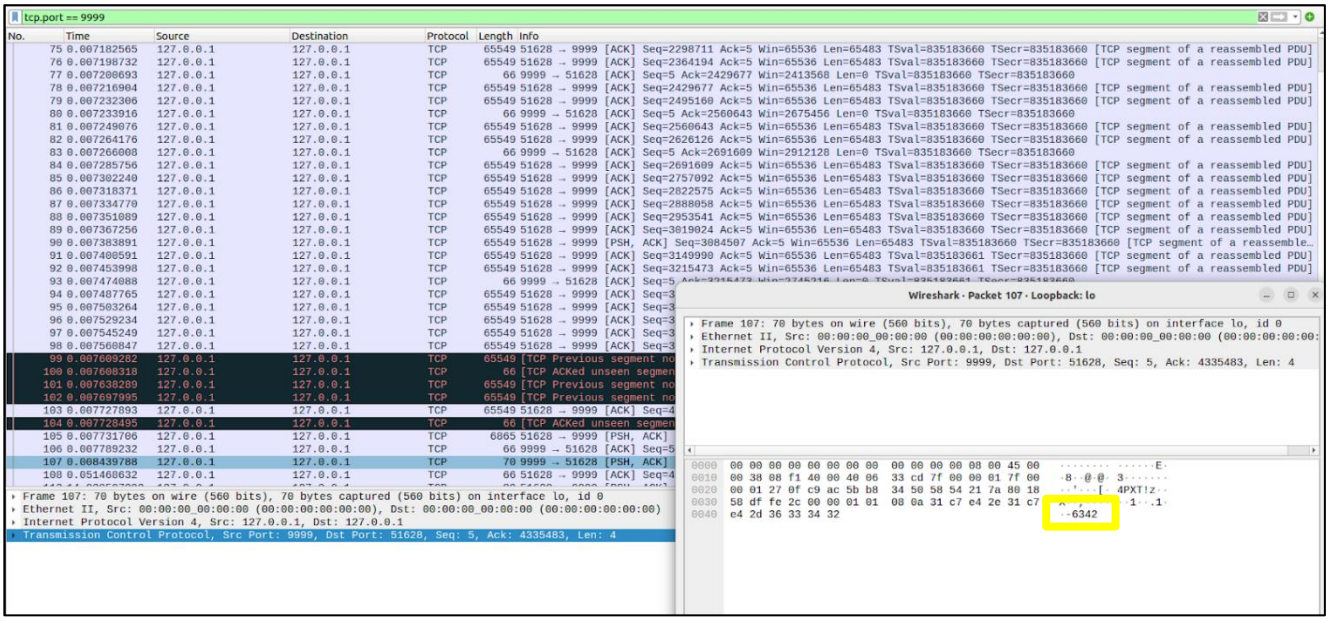
כעת, יישלחו פאקטות רבות המעבירות את המידה, ניתן לזהות את העברת המידע מתחילה בפאקטה הרביעית ע״י זה שנראה כי הפאקטה מעבירה את אורך הקובץ לreceiver, ולאחר מכן המידע מתחיל לעבור



כעת נבחין בפאקטה (מספר 70) המעבירה את הxor של תעודות הזהות שלנו (6396 ^ 0058), פאקטת האוטנטיקאצייה לחלק הראשון של הקובץ

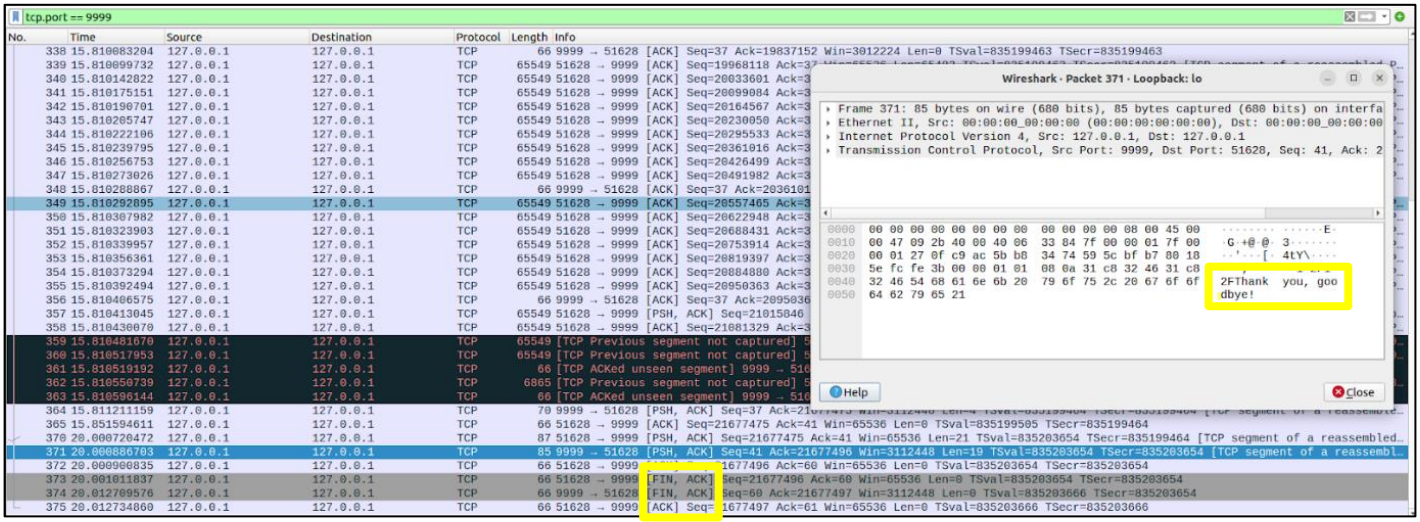


לאחר מכן החלק השני של הקובץ ממשיך להישלח, עד שבפאקטה מספר 107 נבחין שוב בxor שלנו, המאשר את קבלת החלק השני



כך נשלח את הקובץ שוב ושוב במשך 5 פעמים ובסופם נזהה את הפאקטה האחרונה של העברת מידע בה הreceiver שולח לsender הודעת תודה ולהתראות לאחר שהsender הודיע לו על סיום החיבור

לאחר מכן יופיעו פאקטות FIN המעידות על סגירת חיבור קיים

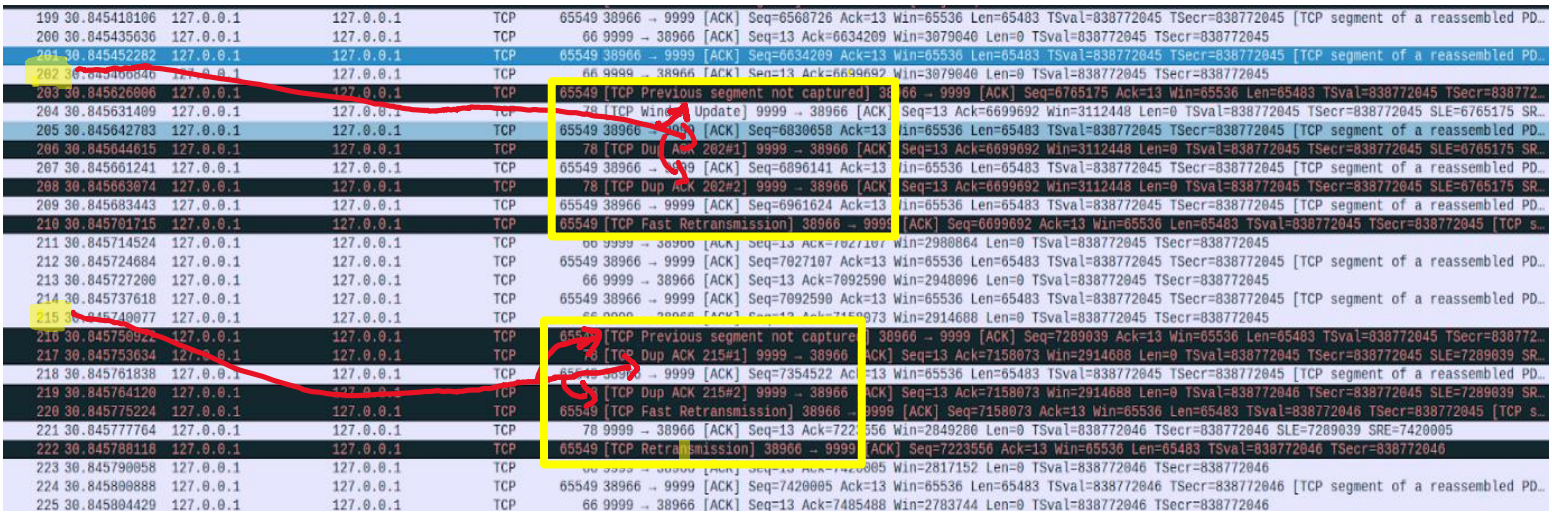


הרצה שנייה – 10% איבוד פאקטות

ניתן להבחין בבירור בהאטה משמעותית ביותר של שליחת הקבצים בהשוואה לחלק הראשון, בנוסף מבחינים כי הפער בין האלגוריתמים נהיה משמעותי יותר כאשר cubic מהיר יותר בצורה משמעותית מאשר reno

```
Stop sending the file

Time of receiving number  1 :
part 1:  0.22402334213256836  seconds
part 2:  0.003892660140991211  seconds

Time of receiving number  2 :
part 1:  0.004171133041381836  seconds
part 2:  0.2207174301147461  seconds

Time of receiving number  3 :
part 1:  0.2729053497314453  seconds
part 2:  0.0023365020751953125  seconds

Time of receiving number  4 :
part 1:  0.6241581439971924  seconds
part 2:  0.002256631851196289  seconds

Time of receiving number  5 :
part 1:  0.026320934295654297  seconds
part 2:  0.003447294235229492  seconds

Average of part 1: 0.23031578063964844 seconds
Average of part 2: 0.04653010368347168 seconds
Total Average:  0.13842294216156006 seconds

Close the connection..
(venv) ilan@ilan-Latitude-5420:~/PycharmProjects/pythonProject$
```

ברור כי כל התהליך שתיארנו בהרצה ללא איבוד הפאקטות מתרחש גם כאן וניתן לזהות אותו באופן זהה,
אלא שכאן המערכת צריכה להתמודד עם איבוד הפאקטות, נתבונן בהקלטות הwireshark ונלמד מהם
כיצד המערכת מגיבה לאיבוד הפאקטות



199	30.845418106	127.0.0.1	127.0.0.1	TCP	65549 38966 → 9999 [ACK] Seq=6568726 Ack=13 Win=65536 Len=65483 TSval=838772045 TSecr=838772045 [TCP segment of a reassembled PD…
200	30.845435636	127.0.0.1	127.0.0.1	TCP	66 9999 → 38966 [ACK] Seq=13 Ack=6634209 Win=3079040 Len=0 TSval=838772045 TSecr=838772045
201	30.845452282	127.0.0.1	127.0.0.1	TCP	65549 38966 → 9999 [ACK] Seq=6634209 Ack=13 Win=65536 Len=65483 TSval=838772045 TSecr=838772045 [TCP segment of a reassembled PD…
202	30.845466846	127.0.0.1	127.0.0.1	TCP	66 9999 → 38966 [ACK] Seq=13 Ack=6699692 Win=3079040 Len=0 TSval=838772045 TSecr=838772045
203	30.845626006	127.0.0.1	127.0.0.1	TCP	65549 [TCP Previous segment not captured] 38966 → 9999 [ACK] Seq=6765175 Ack=13 Win=65536 Len=65483 TSval=838772045 TSecr=838772…
204	30.845631409	127.0.0.1	127.0.0.1	TCP	78 [TCP Window Update] 9999 → 38966 [ACK] Seq=13 Ack=6699692 Win=3112448 Len=0 TSval=838772045 TSecr=838772045 SLE=6765175 SR…
205	30.845642783	127.0.0.1	127.0.0.1	TCP	65549 38966 → 9999 [ACK] Seq=6830658 Ack=13 Win=65536 Len=65483 TSval=838772045 TSecr=838772045 [TCP segment of a reassembled PD…
206	30.845644615	127.0.0.1	127.0.0.1	TCP	78 [TCP Dup ACK 202#1] 9999 → 38966 [ACK] Seq=13 Ack=6699692 Win=3112448 Len=0 TSval=838772045 TSecr=838772045 SLE=6765175 SR…
207	30.845661241	127.0.0.1	127.0.0.1	TCP	65549 38966 → 9999 [ACK] Seq=6896141 Ack=13 Win=65536 Len=65483 TSval=838772045 TSecr=838772045 [TCP segment of a reassembled PD…
208	30.845663074	127.0.0.1	127.0.0.1	TCP	78 [TCP Dup ACK 202#2] 9999 → 38966 [ACK] Seq=13 Ack=6699692 Win=3112448 Len=0 TSval=838772045 TSecr=838772045 SLE=6765175 SR…
209	30.845683443	127.0.0.1	127.0.0.1	TCP	65549 38966 → 9999 [ACK] Seq=6961624 Ack=13 Win=65536 Len=65483 TSval=838772045 TSecr=838772045 [TCP segment of a reassembled PD…
210	30.845701715	127.0.0.1	127.0.0.1	TCP	65549 [TCP Fast Retransmission] 38966 → 9999 [ACK] Seq=6699692 Ack=13 Win=65536 Len=65483 TSval=838772045 TSecr=838772045 [TCP s…
211	30.845714524	127.0.0.1	127.0.0.1	TCP	66 9999 → 38966 [ACK] Seq=13 Ack=7027107 Win=2980864 Len=0 TSval=838772045 TSecr=838772045
212	30.845724684	127.0.0.1	127.0.0.1	TCP	65549 38966 → 9999 [ACK] Seq=7027107 Ack=13 Win=65536 Len=65483 TSval=838772045 TSecr=838772045 [TCP segment of a reassembled PD…
213	30.845727200	127.0.0.1	127.0.0.1	TCP	66 9999 → 38966 [ACK] Seq=13 Ack=7092590 Win=2948096 Len=0 TSval=838772045 TSecr=838772045
214	30.845737618	127.0.0.1	127.0.0.1	TCP	65549 38966 → 9999 [ACK] Seq=7092590 Ack=13 Win=65536 Len=65483 TSval=838772045 TSecr=838772045 [TCP segment of a reassembled PD…
215	30.845740077	127.0.0.1	127.0.0.1	TCP	66 9999 → 38966 [ACK] Seq=13 Ack=7158073 Win=2914688 Len=0 TSval=838772045 TSecr=838772045
216	30.845750922	127.0.0.1	127.0.0.1	TCP	65549 [TCP Previous segment not captured] 38966 → 9999 [ACK] Seq=7289039 Ack=13 Win=65536 Len=65483 TSval=838772045 TSecr=838772…
217	30.845753634	127.0.0.1	127.0.0.1	TCP	78 [TCP Dup ACK 215#1] 9999 → 38966 [ACK] Seq=13 Ack=7158073 Win=2914688 Len=0 TSval=838772045 TSecr=838772045 SLE=7289039 SR…
218	30.845761838	127.0.0.1	127.0.0.1	TCP	65549 38966 → 9999 [ACK] Seq=7354522 Ack=13 Win=65536 Len=65483 TSval=838772045 TSecr=838772045 [TCP segment of a reassembled PD…
219	30.845764120	127.0.0.1	127.0.0.1	TCP	78 [TCP Dup ACK 215#2] 9999 → 38966 [ACK] Seq=13 Ack=7158073 Win=2914688 Len=0 TSval=838772046 TSecr=838772045 SLE=7289039 SR…
220	30.845775224	127.0.0.1	127.0.0.1	TCP	65549 [TCP Fast Retransmission] 38966 → 9999 [ACK] Seq=7158073 Ack=13 Win=65536 Len=65483 TSval=838772046 TSecr=838772045 [TCP s…
221	30.845777764	127.0.0.1	127.0.0.1	TCP	78 9999 → 38966 [ACK] Seq=13 Ack=7223556 Win=2849280 Len=0 TSval=838772046 TSecr=838772046 SLE=7289039 SRE=7420005
222	30.845788118	127.0.0.1	127.0.0.1	TCP	65549 [TCP Retransmission] 38966 → 9999 [ACK] Seq=7223556 Ack=13 Win=65536 Len=65483 TSval=838772046 TSecr=838772046
223	30.845790058	127.0.0.1	127.0.0.1	TCP	66 9999 → 38966 [ACK] Seq=13 Ack=7420005 Win=2817152 Len=0 TSval=838772046 TSecr=838772046
224	30.845800888	127.0.0.1	127.0.0.1	TCP	65549 38966 → 9999 [ACK] Seq=7420005 Ack=13 Win=65536 Len=65483 TSval=838772046 TSecr=838772046 [TCP segment of a reassembled PD…
225	30.845804429	127.0.0.1	127.0.0.1	TCP	66 9999 → 38966 [ACK] Seq=13 Ack=7485488 Win=2783744 Len=0 TSval=838772046 TSecr=838772046

נבחין פה בפאקטות הבעייתיות ונבין את המשמעות שלהם:

"TCP previous segment not captured" הפאקטה הקודמת עדיין לא התקבלה ולכן הreceiver מסמן לsender לשלוח לו אותה שוב

"TCP Dup ACK" הreceiver שולח לsender שיידע כי ישנה פאקטה שלא הגיעה אליו, או שהיא אבדה או שעדיין לא הגיעה (בתוספת x#y כאשר xמסמל את מספר הפאקטה y סופר כמה פעמים כבר הוא התריע על זה)

"TCP out-of-order" הפאקטה לא הגיעה במיקום הנכון שלה פי סדר השליחה

"TCP retransmission" - השולח לא מקבל אישור מהיעד כעבור זמן מסוים (טיימאאוט מוגדר) ולכן מבין שהפאקטה לא הגיעה כמו שצריך ושולח אותה מחדש

"TCP fast retransmission" השולח מבין על פי סדר האישורים שהוא מקבל מהיעד כי יש פאקטות נאבדו בדרך ולכן שולח אותם שוב עוד בטרם נגמר הטיימאאוט שהוקצב לקבלת האישור

נבחין כי אכן שורה 203 מציינת כי פאקטה 202 לא הגיעה ולאחר מכן נשלחת שוב ושוב בקשה מהreceiver לsender (פאקטות 206,208) עד שהוא שולח אותה שוב בשורה 210, לאחר מכן קורה תהליך דומה עם פאקטה 215 שלא מגיעה ליעדה.

בצורה זו מתמודדת המערכת עם אובדן הפאקטות ומנסה להמשיך את התנהלות התכנית בצורה רגילה ולשלוח שוב את הפאקטות החסרות על מנת להשלים את אובדן המידע

<u>ניתן כעת להבין מדוע לוקח יותר זמן לשליחת הקבצים, המערכת צריכה להקצות זמן ומשאבים למרדף אחרי הפאקטות שלא הגיעו ושליחתם מחדש ולכן זמן השליחה של הקבצים גדל</u>

הרצה שלישית – 15% איבוד פאקטות

שוב, הזמנים הכלליים של שליחה הולכים ועולים (הממוצע עלה בכ0.3 שניות) כאשר cubic שומר על יתרון לעומת reno אך הפער מצטמצם כאשר ב10% איבוד הוא היה מהיר יותר מפי 4 ואילו ב15% הוא מהיר בערך פי 3.
נמשיך לעקוב מה יקרה ב20%

```
Stop sending the file

Time of receiving number  1 :
part 1:  0.28852009773254395  seconds
part 2:  0.06921982765197754  seconds

Time of receiving number  2 :
part 1:  0.21334266662597656  seconds
part 2:  0.447465181350708  seconds

Time of receiving number  3 :
part 1:  0.6975066661834717  seconds
part 2:  0.045142173767089844  seconds

Time of receiving number  4 :
part 1:  0.5262246131896973  seconds
part 2:  0.2357017993927002  seconds

Time of receiving number  5 :
part 1:  1.4934086799621582  seconds
part 2:  0.26207995414733887  seconds

Average of part 1: 0.6438005447387696 seconds
Average of part 2: 0.2119217872619629 seconds
Total Average:  0.4278611660003662 seconds

Close the connection..
(venv) ilan@ilan-Latitude-5420:~/PycharmProjects/pythonProject$
```

באופן זהה לאיבוד של 10% המערכת מתמודדת בצורה של שליחת התרעות על איבוד פאקטות ושליחה שלהם מחדש, אולם כמות הפעמים שתהליך זה קורה עלתה ואנו מגיעים למספרי שיא של כמות פאקטות בתהליך השליחה (פאקטות רבות נשלחות על מנת להתמודד עם אובדן הפאקטות)
כפי שניתן לראות, סגירת החיבור וסיום ההתקשרות מגיעות רק בפאקטה מספר 632 (לעומת סגירה בפאקטה מספר 375 בהרצה ללא איבודים)

618	213.613906592	127.0.0.1	127.0.0.1	TCP	65549	37496 → 9999 [ACK] Seq=21474227 Ack=37 Win=65536 Len=65483 TSval=844132033 TSecr=844132033 [TCP segment of a reassembled P…
619	213.613910228	127.0.0.1	127.0.0.1	TCP	65549	37496 → 9999 [ACK] Seq=21539710 Ack=37 Win=65536 Len=65483 TSval=844132033 TSecr=844132033 [TCP segment of a reassembled P…
620	213.613913803	127.0.0.1	127.0.0.1	TCP	65549	37496 → 9999 [ACK] Seq=21605193 Ack=37 Win=65536 Len=65483 TSval=844132033 TSecr=844132033 [TCP segment of a reassembled P…
621	213.614011385	127.0.0.1	127.0.0.1	TCP	66	9999 → 37496 [ACK] Seq=37 Ack=21670676 Win=2684800 Len=0 TSval=844132033 TSecr=844132033
622	213.829435288	127.0.0.1	127.0.0.1	TCP	6865	37496 → 9999 [PSH, ACK] Seq=21670676 Ack=37 Win=65536 Len=6799 TSval=844132249 TSecr=844132033 [TCP segment of a reassembl…
623	213.829559868	127.0.0.1	127.0.0.1	TCP	70	9999 → 37496 [PSH, ACK] Seq=37 Ack=21677475 Win=3145728 Len=4 TSval=844132249 TSecr=844132249 [TCP segment of a reassemble…
624	213.873405931	127.0.0.1	127.0.0.1	TCP	66	37496 → 9999 [ACK] Seq=21677475 Ack=41 Win=65536 Len=0 TSval=844132293 TSecr=844132249
625	215.391540240	127.0.0.1	127.0.0.1	TCP	87	37496 → 9999 [PSH, ACK] Seq=21677475 Ack=41 Win=65536 Len=21 TSval=844133811 TSecr=844132249 [TCP segment of a reassembled…
626	215.391626049	127.0.0.1	127.0.0.1	TCP	85	9999 → 37496 [PSH, ACK] Seq=41 Ack=21677496 Win=3145728 Len=19 TSval=844133811 TSecr=844133811 [TCP segment of a reassembl…
627	215.391636488	127.0.0.1	127.0.0.1	TCP	66	37496 → 9999 [ACK] Seq=21677496 Ack=60 Win=65536 Len=0 TSval=844133811 TSecr=844133811
[illegible]	15.391680124	127.0.0.1	127.0.0.1	TCP	66	37496 → 9999 [FIN, ACK] Seq=2167[illegible] Win=65536 Len=0 TSval=844133811 TSecr=844133811
629	15.601408315	127.0.0.1	127.0.0.1	TCP	66	[TCP Retransmission] 37496 → 9999 [FIN, ACK] Seq=21677496 Ack=60 Win=65536 Len=0 TSval=844134021 TSecr=844133811
630	15.601419929	127.0.0.1	127.0.0.1	TCP	78	[TCP Previous segment not captured] 9999 → 37496 [ACK] Seq=61 Ack=21677497 Win=3145728 Len=0 TSval=844134021 TSecr=8441340…
631	15.625358086	127.0.0.1	127.0.0.1	TCP	66	[TCP Retransmission] 9999 → 37496 [FIN, ACK] Seq=60 Ack=21677497 Win=3145728 Len=0 TSval=844134045 TSecr=844134021
632	15.625373358	127.0.0.1	127.0.0.1	TCP	66	37496 → 9999 [ACK] Seq=21677497 [illegible] 536 Len=0 TSval=844134045 TSecr=844134045

הרצה רביעית – 20% איבוד פאקטות

הזמנים שוב גדלו ממש בכמעט 4.5 שניות בממוצע מאשר 15%, אולם קורה כאן מהפך גדול בו אלגוריתם reno נהייה לראשונה מהיר יותר ועוד בצורה משמעותית (בממוצע 2.5 שניות מהיר יותר)

```
Stop sending the file

Time of receiving number  1 :
part 1:  8.277481317520142  seconds
part 2:  5.599479913711548  seconds

Time of receiving number  2 :
part 1:  0.6230425834655762  seconds
part 2:  0.9399144649505615  seconds

Time of receiving number  3 :
part 1:  0.2369091510772705  seconds
part 2:  1.8524489402770996  seconds

Time of receiving number  4 :
part 1:  6.867160320281982  seconds
part 2:  2.511601209640503  seconds

Time of receiving number  5 :
part 1:  0.015921592712402344  seconds
part 2:  18.908395290374756  seconds

Average of part 1: 3.2041029930114746 seconds
Average of part 2: 5.962367963790894 seconds
Total Average:  4.583235478401184 seconds

Close the connection..
```

מבחינת הקלטת wireshark, נבחין כי הקו ממשיך מהאיבודים של 15%,10% כאשר דרך הטיפול באיבוד פאקטות נשארת זהה, אולם כמות המקרים שאיבוד פאקטות קורה (וישירות מכך כמות הפעמים שהיה צריך לטפל באיבוד) עלתה משמעותית וניתן לראות זאת בצורה ישירה בכמות הפאקטות שהתהליך דרש (סגירת החיבור נעשתה רק בפאקטה 717!)

711	108.394503756	127.0.0.1	127.0.0.1	TCP	87	39444 → 9999 [PSH, ACK] Seq=21677475 Ack=41 Win=65536 Len=21 TSval=845394739 TSecr=845392529 [TCP segment of a reassembled…
712	108.394616302	127.0.0.1	127.0.0.1	TCP	85	9999 → 39444 [PSH, ACK] Seq=41 Ack=21677496 Win=3145728 Len=19 TSval=845394739 TSecr=845394739 [TCP segment of a reassembl…
[illegible]	108.394629943	127.0.0.1	127.0.0.1	TCP	66	39444 → 9999 [illegible] 21677496 Ack=60 Win=65536 Len=0 TSval=845394739 TSecr=845394739
714	108.394674145	127.0.0.1	127.0.0.1	TCP	66	39444 → 9999 [FIN, ACK] Seq=21677496 Ack=60 Win=65536 Len=0 TSval=845394739 TSecr=845394739
715	108.396989159	127.0.0.1	127.0.0.1	TCP	66	9999 → 39444 [FIN, ACK] Seq=60 Ack=21677497 Win=3145728 Len=0 TSval=845394742 TSecr=845394739
716	108.776090155	127.0.0.1	127.0.0.1	TCP	66	[TCP Retransmission] 9999 → 39444 [FIN, ACK] Seq=60 Ack=21677497 Win=3145728 Len=0 TSval=845395121 TSecr=845394739
717	108.776107062	127.0.0.1	127.0.0.1	TCP	66	39444 → 9999 [ACK] Seq=21677497 Ack=61 Win=65536 Len=0 TSval=845395121 TSecr=845394742

מבדיקה יסודית בהקלטה, נבחין כי ב20% איבוד לא נעשו בכלל פעולות TCP fast retransmission אלא רק TCP retransmission רגיל. ההגיון המסתבר הוא כי מרוב שהרבה פאקטות נאבדות אזי השולח לא מצליח להבין בצורה ברורה האם פאקטה מסוימת נאבדה רק לפי האישורים של הפאקטות שלאחריה ולכן הוא שולח אותה שוב רק במקרה של טיימאאוט

סיכום הנתונים שאספנו

	ללא איבוד	10% איבוד	15% איבוד	20% איבוד
חלק ראשון (reno)	0.00387 שניות	0.23031 שניות	0.64380 שניות	3.20410 שניות
חלק שני (cubic)	0.00302 שניות	0.04653 שניות	0.21192 שניות	5.96236 שניות
ממוצע	0.00344 שניות	0.13842 שניות	0.42786 שניות	4.58323 שניות

סיכום:

נראה כי התכנית הצליחה להתמודד בצורה טובה עם אובדן הפאקטות ולהשלים בצורה תקינה את שליחת הקבצים וסגירת החיבור גם באחוז איבוד גבוה יחסית כמו שהגדרנו.

מבחינת איכות האלגוריתמים, קיבלנו כי אלגוריתם cubic מהיר יותר כאשר התכנית פועלת בצורה תקינה וללא איבוד פאקטות משמעותי, אולם ברגע שהתכנית נהיית לא יציבה ומתאפיינת בכמות גדולה של איבוד פאקטות אזי אלגוריתם reno מתעלה עליו ונהיה טוב יותר ולכן עדיף במקרה כזה.