רשתות תקשורת פרויקט גמר

מגישים:

בן שלו- 204818538

מאיר יוסט- 318342037

צבי רכל - 316289495

נועם גלינסקי- 206884751

חלק יבש

שאלה 1: תארו במלים שלכם 5 חסרונות/מגבלות של TCP

1. TCP משתמש באותו מנגנון חלון עבור בקרת עומס ואמינות. כתוצאה מכך, מקרה של אבדן של חבילה אחת יעצור את התקדמות החלון וימנע מחבילות חדשות להגיע עד שהחבילה שאבדה תשוחזר, על אף שרוב החבילות בחלון כבר יצאו מהרשת והיה ניתן לשלוח חבילות נוספות. זה גורם לניצול לא יעיל של המשאבים ומאט את הקצב.
2. בנוסף לחיסרון הראשון, מכיוון שחבילות נשלחות על פי סדר, אבדן של חבילה אחת ימנע מחבילות אחרות אחריה שכבר הגיעו למקבל לעבור לשכבת האפליקציה ולממש את המידע שבהן.
3. בTCP נדרש תהליך של שלושה שלבים על מנת להקים חיבור(לחיצת יד משולשת). בנוסף לכך, אם מאבטחים את החיבור נדרשים עוד שני שלבים על מנת להחליף אישורי אבטחה. תהליך זה מעכב את ההעברה של הנתונים עצמם.
4. לheader של TCP יש שדות בגודל קבוע שמגבילים את היכולת שלו להתמודד עם מהירויות רשת גבוהות. למשל ה- sequence number והACK הם בגודל של4 בתים וכאשר המהירות גבוהה השדות הללו עלולים להגיע למספר המקסימלי ולהתאפס וכך בעצם הם אינם ייחודיים לכל חבילה.
5. TCP משתמש בשילוב של כתובות IP ומספרי פורטים כמזהה חיבור. כתוצאה מכך, שינוי בכתובת IP של אחד הצדדים מנתק את החיבור הקיים, גורם לאבדן של המידע שהיה בדרך ודורש הקמת חיבור חדש.

שאלה 2:ציינו 5 תפקידים שפרוטוקול תעבורה צריך למלא

1. הגדרת מזהי חיבור ונתונים - פרוטוקול תעבורה צריך לספק מזהה חיבור ייחודי שמחבר בין שני הקצוות של החיבור, מזהה נתונים ייחודי שיאפשר זיהוי וסידור של החבילות.
2. ניהול חיבור תעבורה - ניהול חיבור כולל הגדרת מצב החיבור, הקמה וניתוק של חיבורים, והחלפת מידע בקרה בין הקצוות. בנוסף בתמיכה בשינויים בכתובת הIP.
3. אמינות - הפרוטוקול צריך לספק מנגנונים לזיהוי אובדן חבילות, שחזור חבילות אבודות, וסידור נכון של המידע המתקבל.
4. בקרת עומס - הפרוטוקול צריך לשלוט בכמות החבילות ברשת ולהגביל אותה בהתאם לעומס הקיים ברשת.
5. אבטחה - הפרוטוקול צריך לספק מנגנוני אבטחה על מנת להבטיח סודיות של הנתונים שמעוברים.

<u>שאלה 3</u>:אופן פתיחת הקשר ב-QUIC

1. QUIC משלב את תהליכי ה-handshake של התעבורה והאבטחה יחד:

הלקוח שולח חבילת Initial הכוללת:

- הודעת ClientHello של TLS
- פרמטרי תעבורה של QUIC
- מפתח ציבורי לחילוף מפתחות Diffie-Hellman

השרת מגיב עם:

- חבילת Initial הכוללת הודעת ServerHello של TLS
- חבילת Handshake הכוללת:
- EncryptedExtensions עם פרמטרי תעבורה של השרת
- אישור השרת (Certificate)
- אימות האישור (CertificateVerify)
- הודעת Finished של TLS

הלקוח משיב עם:

- חבילת Handshake הכוללת הודעת Finished של TLS

השרת מאשר סיום התהליך עם:

- חבילת RTT-1 הכוללת מסגרת HANDSHAKE_DONE

2. תמיכה ב-0-RTT:

QUIC מאפשר ללקוח לשלוח נתונים מוצפנים כבר בחבילה הראשונה, תוך שימוש במפתחות ופרמטרים מחיבור קודם.

שיפורים לעומת חסרונות TCP:

1. עיכוב בפתיחת קשר:

- QUIC משלים את ה-handshake ב-RTT-1 במקום RTT-3 של TCP+TLS.
- תמיכה ב-0-RTT מאפשרת שליחת נתונים מיידית בחיבורים חוזרים.

2. חוסר גמישות בזיהוי חיבורים:

- QUIC משתמש במזהה חיבור ייחודי במקום צמד כתובות IP/פורטים.
- מאפשר החלפת כתובות IP תוך כדי החיבור (למשל במעבר בין רשתות)

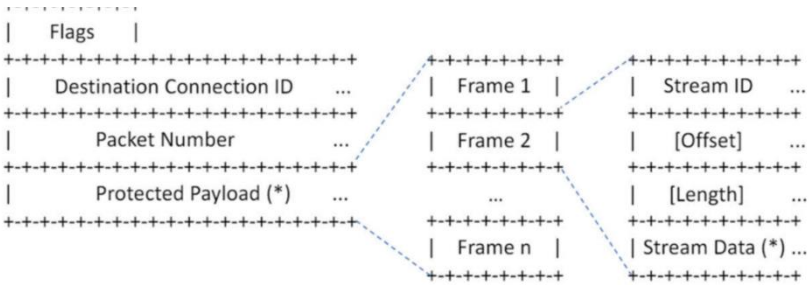
3. שילוב אבטחה:

- QUIC משלב אבטחת TLS 1.3 כחלק אינטגרלי מהפרוטוקול.
- מצפין את רוב תוכן החבילה, כולל מספרי רצף ואישורי קבלה.

<u>שאלה 4</u>:מבנה החבילה של QUIC

ל QUIC יש שני סוגים של כותרות בחבילה. חבילות להקמת חיבור המכילות פריטים של מידע, ולכן חבילות אלו משתמשות בכותרת ארוכה. לאחר שנוצר החיבור רק שדות מסויימים נחוצים והחבילות הבאות משתמשות בפורמט כותרת קצר.

Flag – מייצג את סוג החבילה וגרסת הפרוטוקול
Destination connection ID– מספר סידורי של החיבור
Packet Number– מייצג את מספר החבילה
Protected payload– זהו למעשה תוכן החבילה כאשר התוכן מחולק למספר frameים כאשר לכל frame יש מזהה זרם, היסט מתחילת החבילה(לזיהוי המיקום שלו), אורך ולבסוף תוכן



תמונה 4: פורמט חבילה עם כותרת קצרה ב-QUIC

שיפורים לעומת חסרונות TCP:

1. גמישות וניידות:

- מזהה חיבור ייחודי במקום צמד כתובות IP/פורטים
- מאפשר החלפת כתובות IP תוך כדי החיבור

שיפור: מתגבר על בעיית הניידות של TCP

2. יעילות ואמינות:

- מספרי חבילה חד כיווניים וגדלים מונוטונית
- מונע בעיות עמימות בשידור חוזר

שיפור: מאפשר זיהוי אובדן חבילות ושידור חוזר יעיל יותר מ-TCP

3. יעילות בשימוש ברוחב פס:

- תמיכה בתתי זרמים בתוך חבילה אחת
- מאפשר העברת נתונים מזרמים שונים במקביל

שיפור: מונע חסימת ראש תור (HOL Blocking) ברמת הזרמים, בניגוד ל-TCP

4. גמישות ויכולת התפתחות:

- שימוש במסגרות מטופסות בגוף החבילה
- מאפשר הוספת סוגי מסגרות חדשים בעתיד

שיפור: מונע התאבנות פרוטוקול ומאפשר הרחבות קלות יותר מ-TCP

<u>שאלה 5:</u> מה QUIC עושה כאשר חבילות מגיעות באיחור או לא מגיעות כלל?

כאשר חבילות מגיעות באיחור, QUIC מטפל בהן בעזרת מנגנונים המונעים שליחה מחדש מיותרת של החבילה: בצד השולח, קיימים שני ספים שמנחים האם לשדר מחדש את החבילה- על פי מספר החבילה(מספר סידורי קטן של החבילה האבודה ביחס למספר החבילה האחרונה שהתקבלה) ועל פי זמן העיכוב. הגעה מאוחרת של ACK שלא חורגת מהמגבלות של הספים הללו, לא תגרום לשליחה מחדש. אם אכן זוהה אובדן של חבילה, הפריימים שאבדו נארזים מחדש בתוך חבילה חדשה עם מספר שונה ללא קשר לחבילה שאבדה.

<u>שאלה 6:</u> תארו את בקרת העומס(control congestion) שלQUIC

בבקרת העומס של Quic אין לנו איזה שהם אלגוריתמים חדשים, אלא Quic נותן לנו את הקווים הכלליים של בקרת העומס ומאפשר לנו לממש בעצמנו את הבקרה עצמה. בקרת העומס של QUIC משתמשת במספרי חבילות, ובתכונה של TCP:

היא מגבילה את מספר הבתים שהשולח מעביר בכל זמן נתון. ההגבלה של העברת החבילה תתבצע רק כאשר נזהה עומס יתר בעקביות שזה קורה כאשר שני חבילות נאבדו לנו (לא הגיעו ליעד) ובנוסף כל החבילות שנשלחו ביניהם לא אושרו. במקרה כזה Quic יאט את קצב השליחה על ידי הגדלת מרווח השליחה של החבילות (החישוב של קצב ההאטה מבוסס על RTT הממוצע) במטרה להוריד את רמת העומס המיידי.

חלק רטוב

בחלק זה בחרנו לממש את סעיף 1- **ריבוי זרימות**. בחלק זה, פיתחנו פרוטוקול בשם MyQUIC שמבצע העברת נתונים דרך מספר זרימות במקביל על גבי קשר יחיד. בפרויקט השתמשנו בפרוטוקול שבנינו לשם העברת נתונים בין לקוח לשרת.

מבנה הפרויקט

הפרויקט כולל 4 קבצי קוד:

- **MyQUIC**-כולל את כל הפונקציות והמחלקות ששייכות לפרוטוקול.
- **MyServer**-תכנית שרת להעברת קבצים ללקוח תוך שימוש בפרוטוקול.
- **MyClient**-תכנית לקוח המבקשת קבצים מהשרת תוך שימוש בפרוטוקול.
- **Test**- בדיקות עבור הפרוטוקול.

הקובץ MyQUIC כולל את המחלקות הבאות:

PacketHeader

- מייצגת את הheader של חבילת MyQUIC.

- כוללת שדות `type` ו-`number` לזיהוי סוג החבילה ומספרה.

- מספקת פונקציות לסריאליזציה ודסריאליזציה של הheader .

Frame

- מייצגת header של frame בתוך חבילת MyQUIC.

- שדות:

- type- סוג הframe
- offset- הoffset של הdata לאחר הheader
- streamId- מספר הזרם של הframe
- length- אורך הdata לאחר הheader

בנוסף היא כוללת פונקציות לעדכון הlength והoffset, וכן לסריאליזציה ולדסריאליזציה.

MyQUIC

זו המחלקה המרכזית המיישמת את הפרוטוקול MyQUIC. כיוון שquic מתבסס על פרטוקול UDP, המחלקה פותחת סוקט UDP ומשתמשת בו לאורך התכנית. מחלקה זו כוללת שדות לניהול ומעקב על העברת הנתונים, ופונקציות לקבלת ושליחת נתונים על פי עיקרון ריבוי הזרימות.

<u>פונקציות עיקריות:</u>

sendData

הפונקציה אחראית על שליחת נתונים אל הכתובת שמתקבלת.
הפונקציה מקבלת כתובת ומבנה נתונים מסוג מילון כאשר הkey הוא מספר הstream והvalue זה הקובץ שנדרש להעביר דרך הstream.
הפונקציה מגרילה לכל stream גודל מסוים, שישאר קבוע לאורך כל העברת הנתונים.
לאחר מכן מתחיל תהליך בניית הפקטות ושליחתן:
עבור כל stream הפונקציה יוצרת frame, מוסיפה בתים מהקובץ ומקבצת את כל הframe לחבילה ומנסה לשלוח את החבילה אל הלקוח.
כפי הנדרש, הגדרנו את מספר הזרמים המקסימלי לחבילה ל-6. לכן, במקרה בו כמות הזרמים גדולה מ-6, כל פעם יכנסו לחבילה frames של 6 זרמים שונים באופן אקראי. הפונקציה ממתינה לACK מהלקוח כדי לאשר שהנתונים התקבלו בהצלחה וממשיכה לשלוח את החבילות הבאות.
בסוף תהליך השליחה, הפונקציה מדפיסה את הסטטיסטיקות שנדרשו: מספר הבתים שעברו בכל זרימה(שזה בעצם גודל הקובץ), מספר החבילות ששייכות לכל זרימה, קצבי ההעברת הנתונים לכל זרימה ובאופן כללי.
לבסוף, הפונקציה מחזירה את סך הבתים של הנתונים שנשלחו.

ReceiveData

הפונקציה אחראית על קבלת נתונים ושליחת ACK בחזרה.
הפונקציה מעבדת את החבילה שהתקבלה לפי הheader של הframes.
עבור כל frame, היא בודקת אם הנתונים שהתקבלו הם בסדר הצפוי. אם כן, היא מעדכנת את מספר הבתים שהתקבלו עבור הזרם המתאים.
הפונקציה יוצרת frame מסוג ACK עבור כל frame שהתקבל, מאגדת אותם לחבילת ACKאחת ושולחת את הACK בחזרה ללקוח.
הפונקציה שומרת את הנתונים שהתקבלו במבנה נתונים מילון, כאשר הkey הוא מספר הstream והvalue זה הבתים שהתקבלו.
הפונקציה מחזירה את כתובת השולח ואת מילון הנתונים שהתקבלו.

<u>דוגמת הרצה והקלטות wireshark</u>

בנינו תכנית שרת ותכנית לקוח המשתמשות בפרוטוקול שכתבנו לצורך העברת קבצים, כפי הנדרש.
לקוח: מזין מספר בין 1 ל-10 על מנת לבחור את כמות הקבצים שהוא מעוניין לקבל מהשרת.
לאחר מכן בוחר באופן רנדומלי מספר סטרים עבור כל קובץ, למשל קובץ מספר 1 יעבור דרך סטרים מספר 3 וכד'.

שרת: יוצר 10 קבצים אקראיים(רצף של bytes), מקבל את הבקשה מהלקוח ושולח לו את הקבצים המבוקשים על הסטרימים המתאימים.

בדוגמה המצורפת, הלקוח בוחר לקבל 3 קבצים. לאחר מכן הוא שולח בקשה לשרת עם מספרי הסטרימים ומספרי הקבצים שנבחרו.
בסיום ניתן לראות כי הוא אכן קיבל את הנתונים המתאימים.

```
PS C:\Users\שחר\Desktop\QUIC\MyQUIC> python .\MyClient.py
Please enter the number of files (1-10): 3
You entered: 3
Client starting...
Sending request: 4->10 3->6 9->8
Waiting for data...

Receiving data...
Data reception complete!

Total packets received: 1023
Stream: 4, File number: 10, File size: 1566628 bytes
Stream: 3, File number: 6, File size: 1628629 bytes
Stream: 9, File number: 8, File size: 1921136 bytes
```

בצילום הבא ניתן לראות את ההדפסות בצד השרת- השרת מפענח את בקשת הלקוח, ושולח לו את הקבצים דרך הסטרימים הנדרשים.
בסיום השליחה, מודפסות הסטטיסטיקות.

```
PS C:\Users\שחר\Desktop\QUIC\MyQUIC> python .\MyServer.py
Creating files...
Files creation complete!
Starting server...
Server is ready!
Waiting for client connections...
Client request details:
Stream: 4, file: 10, Actual size: 1566628 bytes
Stream: 3, file: 6, Actual size: 1628629 bytes
Stream: 9, file: 8, Actual size: 1921136 bytes
Total response size: 5116393 bytes
Sending response data...


STATISTICS:

Streams details:
Stream: 4, Size: 1932 bytes, Sent: 1566628 bytes, Sent in 810 different packets, Pace: 3998044.14 B/s, 2067.12 Packets/s
Stream: 3, Size: 1594 bytes, Sent: 1628629 bytes, Sent in 1021 different packets, Pace: 3181084.39 B/s, 1994.25 Packets/s
Stream: 9, Size: 1948 bytes, Sent: 1921136 bytes, Sent in 986 different packets, Pace: 3838531.73 B/s, 1970.08 Packets/s

General details:
Data pace: 9993484.03 B/s, 1998.15 Packets/s
```

הקלטות

ניתן לראות כאן את התהליך שמתבצע- בשתי החבילות הראשונות הלקוח שולח את הבקשה ומקבל עליה ACK מהשרת. לאחר מכן השרת מתחיל בהעברת הקבצים.

ניתן לראות בפקטה הראשונה את הבקשה שמעביר הלקוח לשרת(stream id->file id) :

```
0000  02 00 00 00 45 00 00 43  fb 7e 00 00 80 11 00 00  ····E··C ·~******
0010  7f 00 00 01 7f 00 00 01  ea 39 04 bc 00 2f 8f 34  ******** ·9···/·4
0020  03 00 00 00 00 00 00 00  12 00 00 00 05 00 00 00  ******** ********
0030  00 00 00 00 00 00 00 00  0e 35 2d 3e 38 20 39 2d  ******** ·5->8 9-
0040  3e 33 20 31 2d 3e 31                              >3 1->1
```

כאמור, אנחנו בחרנו את מספר הזרמים המקסימלי בחבילה ל-6. לכן כאשר מספר הזרמים קטן או שווה ל-6, גודל החבילה יהיה קבוע כיוון שגודל כל stream מוגרל ונקבע בתחילת התכנית, וכל פעם חבילה תכלול frames ששייכים לאותם הזרמים. כאן הקלטנו ריצה עם 3 סטרימים ואכן ניתן לראות שגודל החבילות אינו משתנה. בנוסף לכך ניתן לראות את חבילת הACK שמתקבלת לאחר כל חבילת נתונים :

1	0.000000	127.0.0.1	127.0.0.1	UDP	71	59961 → 1212 Len=39
2	0.000198	127.0.0.1	127.0.0.1	UDP	57	1212 → 59961 Len=25
3	0.000619	127.0.0.1	127.0.0.1	UDP	4388	1212 → 59961 Len=4356
4	0.000679	127.0.0.1	127.0.0.1	UDP	97	59961 → 1212 Len=65
5	0.000730	127.0.0.1	127.0.0.1	UDP	4388	1212 → 59961 Len=4356
6	0.000823	127.0.0.1	127.0.0.1	UDP	97	59961 → 1212 Len=65
7	0.000924	127.0.0.1	127.0.0.1	UDP	4388	1212 → 59961 Len=4356
8	0.001013	127.0.0.1	127.0.0.1	UDP	97	59961 → 1212 Len=65
9	0.001106	127.0.0.1	127.0.0.1	UDP	4388	1212 → 59961 Len=4356
10	0.001168	127.0.0.1	127.0.0.1	UDP	97	59961 → 1212 Len=65
11	0.001213	127.0.0.1	127.0.0.1	UDP	4388	1212 → 59961 Len=4356
12	0.001244	127.0.0.1	127.0.0.1	UDP	97	59961 → 1212 Len=65
13	0.001278	127.0.0.1	127.0.0.1	UDP	4388	1212 → 59961 Len=4356
14	0.001308	127.0.0.1	127.0.0.1	UDP	97	59961 → 1212 Len=65
15	0.001438	127.0.0.1	127.0.0.1	UDP	4388	1212 → 59961 Len=4356
16	0.001482	127.0.0.1	127.0.0.1	UDP	97	59961 → 1212 Len=65
17	0.001592	127.0.0.1	127.0.0.1	UDP	4388	1212 → 59961 Len=4356
18	0.001659	127.0.0.1	127.0.0.1	UDP	97	59961 → 1212 Len=65
19	0.001753	127.0.0.1	127.0.0.1	UDP	4388	1212 → 59961 Len=4356
20	0.001802	127.0.0.1	127.0.0.1	UDP	97	59961 → 1212 Len=65
21	0.001896	127.0.0.1	127.0.0.1	UDP	4388	1212 → 59961 Len=4356
22	0.001943	127.0.0.1	127.0.0.1	UDP	97	59961 → 1212 Len=65
23	0.002059	127.0.0.1	127.0.0.1	UDP	4388	1212 → 59961 Len=4356
24	0.002119	127.0.0.1	127.0.0.1	UDP	97	59961 → 1212 Len=65

למרות זאת, ניתן לראות שלקראת הסיום גודל החבילה קטן מאחר שחלק מהקבצים כבר הועברו בשלמותם ולכן פחות נתונים עוברים בשלב זה :

3603	0.669223	127.0.0.1	127.0.0.1	UDP	1112	1212 → 59961 Len=1080
3604	0.669634	127.0.0.1	127.0.0.1	UDP	57	59961 → 1212 Len=25
3605	0.669673	127.0.0.1	127.0.0.1	UDP	1112	1212 → 59961 Len=1080
3606	0.670268	127.0.0.1	127.0.0.1	UDP	57	59961 → 1212 Len=25
3607	0.670310	127.0.0.1	127.0.0.1	UDP	1112	1212 → 59961 Len=1080
3608	0.670717	127.0.0.1	127.0.0.1	UDP	57	59961 → 1212 Len=25
3609	0.670755	127.0.0.1	127.0.0.1	UDP	1112	1212 → 59961 Len=1080
3610	0.671093	127.0.0.1	127.0.0.1	UDP	57	59961 → 1212 Len=25
3611	0.671134	127.0.0.1	127.0.0.1	UDP	1112	1212 → 59961 Len=1080
3612	0.671477	127.0.0.1	127.0.0.1	UDP	57	59961 → 1212 Len=25
3613	0.671541	127.0.0.1	127.0.0.1	UDP	298	1212 → 59961 Len=266
3614	0.671938	127.0.0.1	127.0.0.1	UDP	57	59961 → 1212 Len=25
3615	0.672908	127.0.0.1	127.0.0.1	UDP	60	1212 → 59961 Len=28
3616	0.672993	127.0.0.1	127.0.0.1	UDP	57	59961 → 1212 Len=25

כאשר הרצנו את התכנית עם מספר זרמים גדול מ-6(במקרה כאן עם 10) ניתן לראות שגודל החבילה משתנה מפעם לפעם, כיוון שכל פעם מוגרלים מספרי הזרמים שיכללו בחבילה הבאה:

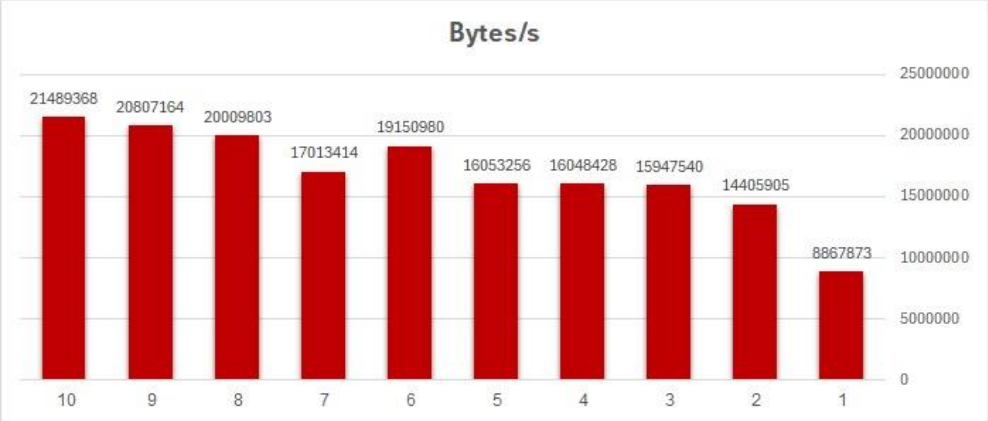
1	0.000000	127.0.0.1	127.0.0.1	UDP	106	59696 → 1212 Len=74
2	0.000133	127.0.0.1	127.0.0.1	UDP	57	1212 → 59696 Len=25
3	0.000828	127.0.0.1	127.0.0.1	UDP	9239	1212 → 59696 Len=9207
4	0.000899	127.0.0.1	127.0.0.1	UDP	157	59696 → 1212 Len=125
5	0.000968	127.0.0.1	127.0.0.1	UDP	8821	1212 → 59696 Len=8789
6	0.001013	127.0.0.1	127.0.0.1	UDP	157	59696 → 1212 Len=125
7	0.001068	127.0.0.1	127.0.0.1	UDP	8684	1212 → 59696 Len=8652
8	0.001104	127.0.0.1	127.0.0.1	UDP	157	59696 → 1212 Len=125
9	0.001279	127.0.0.1	127.0.0.1	UDP	8895	1212 → 59696 Len=8863
10	0.001340	127.0.0.1	127.0.0.1	UDP	157	59696 → 1212 Len=125
11	0.001402	127.0.0.1	127.0.0.1	UDP	9437	1212 → 59696 Len=9405
12	0.001444	127.0.0.1	127.0.0.1	UDP	157	59696 → 1212 Len=125
13	0.001536	127.0.0.1	127.0.0.1	UDP	8094	1212 → 59696 Len=8062
14	0.001588	127.0.0.1	127.0.0.1	UDP	157	59696 → 1212 Len=125
15	0.001643	127.0.0.1	127.0.0.1	UDP	8979	1212 → 59696 Len=8947
16	0.001701	127.0.0.1	127.0.0.1	UDP	157	59696 → 1212 Len=125
17	0.001756	127.0.0.1	127.0.0.1	UDP	8859	1212 → 59696 Len=8827
18	0.001799	127.0.0.1	127.0.0.1	UDP	157	59696 → 1212 Len=125
19	0.001850	127.0.0.1	127.0.0.1	UDP	8213	1212 → 59696 Len=8181
20	0.001885	127.0.0.1	127.0.0.1	UDP	157	59696 → 1212 Len=125
21	0.001954	127.0.0.1	127.0.0.1	UDP	9156	1212 → 59696 Len=9124
22	0.001992	127.0.0.1	127.0.0.1	UDP	157	59696 → 1212 Len=125
23	0.002057	127.0.0.1	127.0.0.1	UDP	8815	1212 → 59696 Len=8783
24	0.002091	127.0.0.1	127.0.0.1	UDP	157	59696 → 1212 Len=125
25	0.002146	127.0.0.1	127.0.0.1	UDP	8377	1212 → 59696 Len=8345
26	0.002189	127.0.0.1	127.0.0.1	UDP	157	59696 → 1212 Len=125

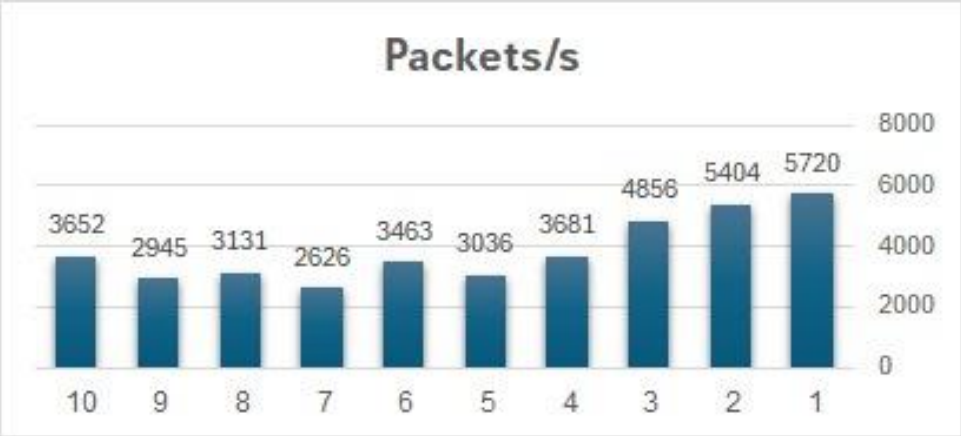
לסיום נציג צילום של חבילת הסיום המסמנת את סיום העברת הנתונים, עם הדגל FIN:

```
0000  02 00 00 00 45 00 00 38  09 9c 00 00 80 11 00 00   ····E··8 ********
0010  7f 00 00 01 7f 00 00 01  04 bc ea 39 00 24 3d d2   ******** ···9·$=·
0020  03 00 00 07 0f 00 00 00  51 00 00 00 05 00 00 00   ******** Q*******
0030  00 00 00 00 00 00 00 00  03 66 69 6e               ******** ·fin
```

<u>ניתוח הנתונים בניסוי</u>

הרצנו את התכנית 10 פעמים כאשר כל פעם מספר הזרמים גדל ב-1, ואלו התוצאות שקיבלנו:





ניתן לראות שככל שמספר הזרמים עולה, כך קצב החבילות הכולל יורד, כיוון שככל שיש יותר זרמים כך החבילות גדולות יותר ולכן לכל חבילה לוקח יותר זמן לעבור ברשת. כיוון שמספר הזרמים המקסימלי מוגדר ל-6, כאשר מספר הזרימות גדול או שווה ל6 הנתונים יחסית זהים, וההבדלים קשורים בין השאר לגודל הזרמים שהוגרל בכל ריצה ולשינויים כלשהם ברשת.

מצד שני, ראינו שככל שמספר הזרמים עולה כך קצב העברת הנתונים הכולל עולה(כאשר מה שנמדד הוא הנתונים ששייכים לקבצים עצמם). חשבנו שזה קורה מכיוון שככל שהחבילה קטנה יותר, הheader וbytes שאינם חלק מהנתונים עצמם תופסים חלק גדול יותר מהחבילה, מה שמפחית את היעילות של העברת הנתונים עצמם. וכאשר החבילות גדולות אזי זה מתחלק על פני כמות גדולה יותר של נתונים, מה שגורם באופן ישיר לכך שקצב העברת הנתונים עצמם יעלה.