לק"י

רשתות תקשורת – פרויקט סיום

Communication Networks - Project - DQUIC - GitHub

מגישים:

עידו מנגדי - 315310250

אסף חדד - 209088152

אוריה פרל - 322522806

שחר בבקוף - 324207935

חלק יבש – תשובות:

<u>שאלה 1:</u> חסרונות של TCP :

1. הקשר בין שליטה בעומס למסירה אמינה: שליטה בעומס ב-TCP ובקרת הזרימה, מתבססים על אותו מנגנון חלון. שימוש בחלון זה אומר שכל אובדן של חבילה עוצר את התקדמות החלון עד שהחבילה האבודה משוחזרת ורק אז יזוז חלון השולח והוא ישלח את החבילה הבאה בתור. הדבר הזה מוביל לחוסר יעילות מבחינת מסירת חבילות, במיוחד כאשר החבילה האבודה כבר עזבה את הרשת אך אובדנה מעכב את כל החבילות הבאות אחריה בחלון השולח (הזמן של תהליך זיהוי הנפילה והשידור מחדש גורע ממהירות השידור)
2. Head of line blocking : בגלל שפרוטוקול TCP מעביר את הפקאטות אחת אחרי השנייה עם חשיבות לסדר, אז אם חבילה בודדת לא הגיעה לצד המקבל, תהליך השליחה ייעצר והפאקטות שכבר הגיעו לצד המקבל שהן אחרי הפאקטה האבודה, לא יועברו לשכבת האפליקציה והמידע בהן לא ימומש עד שהחבילה האבודה משוחזרת ומועברת, מה שגורם לעיכובים אפשריים במימוש המידע הנשלח, אפילו אם החבילות הבאות הגיעו ליעד.
3. עיכוב עקב הקמת חיבור: ל-TCP נדרשת תהליך חיבור של שלושה שלבים (לחיצת היד המשולשת) כדי להקים חיבור, מה שיוצר עיכוב לפני שניתן לשלוח את הנתונים. בנוסף לכך אם נדרשת גם הצפנה של החיבור באמצעות TLS, נדרש סבב נוסף (עוד RTT) להחלפת אישורי האבטחה, מה שמגדיל עוד יותר את זמן הקמת החיבור.
4. הגבלות בגלל שדות קבועים בפרוטוקול: ה-TCP מורכב משדות בגודל קבוע, עם שדה נוסף אופציונלי שמוגבל ל-40 בתים. בגלל השיפור במהירות של תעבורת הרשת, שדות מסוימים, שבעבר היה מספיק מספר מצומצם של בתים בשביל לייצג אותם, כעת מצריכים יותר בתים (שאין) וזה גורם לכך שלא מנצלים את כל יכולות הרשת. לדוגמה : The sequence number field שבעבר היה צריך ארבעה בתים והיום בגלל מהירות האינטרנט מצריך הרבה יותר.
5. תלות בכתובת IP לזיהוי חיבור: פרוטוקול TCP משתמש בכתובות IP ומספרי פורטים כדי לזהות חיבור. אם כתובת ה-IP של נקודת קצה משתנה (בשל ניידות, Multi-Homing או NAT), החיבור הקיים יפסק ואז יהיה צריך לעשות הקמת קשר חדשה באמצעות לחיצת היד המשולשת, וזה גורם לעוד תעבורת רשת (עיכוב בזמן).

<u>שאלה 2:</u> חמשת התפקידים של פרוטוקול תעבורה הם:

1. הגדרת מזהה חיבור ומזהה נתונים.
2. ניהול חיבור התעבורה: קביעת ID ייחודי המחבר בין שני קצוות החיבור ובכך מבטיח תקשורת אמינה בין שני הקצוות, הקמה ופירוק של החיבור, תמיכה בשינויים בכתובות IP ובקרה על החלפת מידע בין שתי הקצוות.
3. מסירת נתונים אמינה: מבטיח שהנתונים עוברים באופן אמין בין שתי נקודות קצה, בעזרת מנגנונים כמו בקרת זרימה והגדרת חלון.
4. בקרת עומס: ניהול מספר החבילות ברשת כדי למנוע עומס יתר.
5. אבטחה: מבטיח חיבור מוצפן ובכך מספק סודיות של הנתונים.

<u>שאלה 3:</u>

פתיחת הקשר בQUIC מבצעת העברת פרמטרי חיבור והחלפת פרטי אבטחה בRTT יחיד (הלקוח שולח את הנתונים הנ"ל, ומקבל עליהם תשובה מהשרת), לעומת זאת ב TCP שני התהליכים קורים בנפרד בשני RTT.

בנוסף, גם הלקוח וגם השרת בוחרים מזהי חיבור באופן עצמאי במהלך לחיצת היד, המשמשים לאורך כל החיבור וזה מאפשר לפאקטות לעבור בצורה נכונה גם אם כתובת הIP ההתחלתי משתנה. לעומת זאת, ב TCP המתבסס על מזהה של כתובת IP, שינוי של כתובת הIP מצריך פתיחת חיבור מחדש.

QUIC תומך גם בשליחת נתונים ב- RTT-0, שבהם לקוח יכול לשלוח נתונים בחבילה הראשונה שלו בהתבסס על פרמטרים שנקבעו בהפעלה קודמת (שכמובן דבר זה אינו נתמך בTCP)

<u>שאלה 4:</u>

QUIC משתמש במבנה פאקטה גמיש שנועד לתמוך במגוון צרכים ואופטימיזציות. המבנה הבסיסי מבדיל בין שני סוגי headers .

Long Header: משתמש בעיקר במהלך הגדרת החיבור הראשוני או כאשר מתבצעים שינויים משמעותיים בפרמטרים של החיבור.

זה כולל:

סוג: המטרה הספציפית של החבילה.

לק"י
גרסה: הגרסה של הפרוטוקול QUIC שבה נעשה שימוש.
מזהה חיבור יעד ומקור: מזהים ייחודיים עבור הקצוות של החיבור.
מספר פאקטה: לכל פאקטה יש מספר זיהוי ובכך אנחנו יכולים לבצע מעקב אחר הפאקטות (מי נשלח ומי נאבד).
מטען (Payload): מידע מוצפן.

Short Header: משמש לאחר יצירת החיבור עבור פאקטות המכילות נתוני אפליקציה.
זה כולל:
מזהה חיבור יעד: לזיהוי החיבור המתמשך.
מספר פאקטה: בדומה ל Long Headerאך כאן הוא משמש לחיבורים שכבר נוצרו.
מטען (Payload): הוא מוצפן ומכיל בתוכו רצף של frames .

ל-TCP יש מבנה פאקטה קבוע, הכולל שדות בגודל קבוע ולעתים קרובות לא מנוצל. זה מגביל את האופן שבו TCP יכול להתפתח ולהשתנות.
לעומת זאת בפרוטוקול QUIC ה- headers של הפאקטות בנויים כך שהם ניתנים להרחבה ,מה שמאפשר דינמיות לפי המצב והצרכים הנוכחיים של החיבור.

בנוסף לכך בTCP אובדן חבילה אחת משפיע על מעבר כל הפאקטות שאחריו כלומר לא ניתן לעבד פאקטות עוקבות עד לשחזור החבילה האבודה, לעומת זאת QUIC מאפשר בתוך אותו חיבור למספר זרמים להתקדם באופן עצמאי ובכך אובדן בזרם אחד אינו משפיע על זרמים אחרים.

<u>שאלה 5:</u>
טיפול בהגעה מאוחרת של פאקטה: מצד השולח, קיימים שני סוגי ספים שעל פיהם נקבע האם לשדר מחדש את החבילה, כך שהגעה מאוחרת של ACK, אבל לא מאוחרת מדי, לא תגרום לשליחה מחדש של החבילה. מצד המקבל, אם חבילה מגיעה באיחור ניתן עדיין להשתמש בה, הACK שיישלח לאחר מכן ייאשר גם את החבילות האחרות שנשלחו אחריה אבל הגיעו לפניה. ספי השליחה מחדש, השימוש בחבילה זו והאישור שלה מונעים שידורים חוזרים מיותרים.

אובדן פאקטה: כמו שהזכרנו לעיל לגבי הספים של שליחה מחדש, אחד מהם הוא זמן המשוערך בהתאם לזמני RTT קודמים. כאשר הT.O הזה קופץ, המידע בסטרימים הספציפיים שהיו בחבילה הקודמת ואבדו, מועברים שוב בתוך חבילה חדשה שאמורה לצאת (יכול להיות שבחבילה היוצאת החדשה יש DATA של עוד סטרימים).

<u>שאלה 6:</u>
בקרת עומס ב-QUIC מתבצעת באופן שמפריד בין בקרת אמינות שמתבצעת לפי offsets לכל סטרים בנפרד לבין בקרת עומס שמתבצעת לפי מספר חבילה. באופן עקרוני הCC של QUIC מתבצע על בסיס חלון שליחה כדי להגביל את כמות המידע שהשולח ישדר לרשת. QUIC מספק נתונים על מצב הרשת ונקודת קצה לא מוגבלת מבחינת שימוש באלגוריתמי CC (כמו QUBIC, RENO וכדומה).
כדי לא לצמצם לחינם את גודל חלון השליחה, QUIC מחכה לעומס יתר עקבי כדלהלן:
אם בזמן נתון, שתי חבילות שלא אושרו מוגדרות כאבודות, כל החבילות שנשלחו ביניהן לא אושרו, והזמן שעבר בין שליחת שתי החבילות הוא יותר מה- "persistent congestion duration" = זמן שמחושב על פי הRTT הממוצע, הסטייה בין מדידת הRTT'ים, והזמן המקסימלי שיכול להידרש לשלוח לעבד את החבילות לפני שליחת ACK (הראשונים תלויים בכך שהייתה כבר מדידת RTT אחת לפחות. האחרון הוא נתון שהמקבל שולח עם הACK).
ורק אם התקבל עומס יתר עקבי כמו שתיארנו הוא יקטין את חלון השליחה. לכן השולח בQUIC ינסה לא להגיע אפילו לעומס יתר זמני על ידי הגדלת הפער שבין שליחת חבילות כך שהוא יעלה על המגבלה שמחושבת על סמך RTT ממוצע, גודל החלון וגודל החבילה.

לק"י

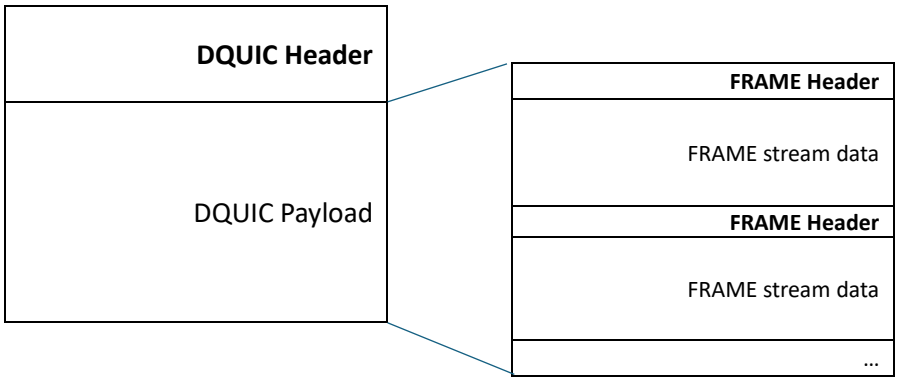
חלק רטוב - פרוטוקול DQUIC: DQUIC - Github

בחרנו לממש את החלק הראשון, ריבוי זרימות וקראנו לפרוטוקול שלנו Demo QUIC (בקיצור DQUIC) .

חלק א': **מבנה הפאקטות**

DQUIC Header כולל בתוכו:
packet_type – סוג החבילה.
packet_number – מספר החבילה לחיבור הספציפי.
DQUIC Frame כולל בתוכו:
stram_id – מספר סידורי של הזרימה.
frame_type – סוג הframe.
offset – מסמן את מספר הבתים שהתקבלו ואושרו על ידי הצד המקבל.
length – אורך המידע שנמצא מיד לאחר הframe בפאקטה הנוכחית.

המחשה:



חלק ב': **מחלקות**

Class DQUICHeader
במחלקה זו מימשנו מבנה של header של פאקטת DQUIC .
שדות: packet_type , packet_nuumber המשמשים לזיהוי סוג החבילה והמספר הסידורי שלה לחיבור הנוכחי.

Class DQUICFrame
במחלקה זו מימשנו מבנה של header של frame ספציפי בתוך פאקטת DQUIC.
שדות: stream_id, frame_type, offset, length (ההסבר בקוד כמשמעות שם השדה).

Class Connection
מחלקה זו מייצגת חיבור. כאשר נוצר קשר בין שתי נקודות קצה, הסוקט שומר נתונים על כל חיבור באמצעות המחלקה הזו.
שדות: addr, conn_id, sent_packet_number, recv_packet_number המייצגים נתונים פשוטים על כל חיבור.
stream_bytes_ack – מילון המחזיק את נתוני הקבלה עבור כל stream בפורמט: (stream_id : bytes_received)
stream_bytes_sent – מילון המחזיק את נתוני השליחה עבור כל stream בפורמט: (stream_id : bytes_sent)

Class DQUIC
המחלקה העיקרית של הפרויקט.

שדות:
sock – סוקט UDP דרכו בפועל מתבצעת התקשורת.
connections – רשימת כל החיבורים של הסוקט.

פונקציות:

bind – מתפקדת כמו bind של UDP, מחברת את הסוקט לכתובת נתונה.

לק"י

Send_to – אחת מהפונקציות העיקריות של הפרוטוקול. מקבלת מילון בפורמט (stream_id : data) ושולחת את המידע לפי הסטרימים המתאימים תוך מימוש עקרון הריבוי זרימות של פרוטוקול QUIC.
שלב א – connection. תחילה הפונקציה בודקת אם הכתובת כבר נמצאת ברשימת החיבורים של האובייקט, אם לא היא מוסיפה אותה בתור connection חדש.
שלב ב – הכנות. עבור כל סטרים שנדרש לשלוח הפונקציה מגרילה גודל סטרים בבתים, יוצרת עבורו פריים, ומוסיפה את הסטרים לרשימת הסטרימים של החיבור (כלומר כל connection של האובייקט שומר כמה בתים נשלחו בכל סטרים).
שלב ג – שליחה וקבלת ack. הפונקציה שולחת את האובייקטים מכל סטרים שעדיין לא סיים לשלוח את האובייקטים שלו. עבור כל חבילה שנשלחת הפונקציה מחכה לקבלת ack, מעדכנת את החיבור והפריים שהמקבל קיבל את המידע וממשיכה לשליחה של עוד מידע. הפונקציה מגבילה את כמות הסטרימים שיכולים להיכנס בכל פאקטה (לפי ערך נתון) ולכן לפני כל שליחה של פאקטה מתבצעת הגרלה על איזה סטרים 'יכנס' לחבילה.
שלב ד – מדדים. בסיום השליחה הפונקציה מדפיסה את המדדים של השליח

Recv_from – אחת מהפונקציות העיקריות של הפרוטוקול. הפונקציה מקבלת את מספר הבתים המקסימלי לקבל. מחזירה את כתובת השולח ומילון של האובייקטים בפורמט: (stream_id : data).
שלב א - connection. תחילה הפונקציה בודקת אם הכתובת כבר נמצאת ברשימת החיבורים של האובייקט, אם לא היא מוסיפה אותה בתור connection חדש.
שלב ב – פענוח החבילה לפי סטרימים, עדכון השדות הרלוונטים והחזרת פאקטת ack.
שלב ג – החזרת כתובת השולח והמידע שהגיע.

Close – הפונקציה סוגרת את סוקט הUDP.

<u>חלק ג':</u> **מימוש במודל שרת ולקוח + הקלטת WireShark**

כמו שניתן לראות בקבצים המצורפים, מימשנו מודל שרת-לקוח שמשתמשים בפרוטוקול בצורה הבאה:
שלב א – הלקוח מבקש מספר אובייקטים כאשר לכל אובייקט שהוא מבקש יש מספר stream שעליו הלקוח מבקש לקבל אותו.
שלב ב – השרת מקבל את הבקשה, מעבד אותה ושולח ללקוח את האובייקטים על מספרי ה stream שהוא ביקש.
(הערה: לא הסברנו פרטים טכניים שלא רלוונטים לשימוש של הצדדים בפרוטוקול כמו איזה בדיוק פריטים הלקוח מבקש וכדומה. ניתן לראות את הכל בצורה מפורטת בקוד)

נתבונן בשני השלבים של התקשורת:

```
PS C:\Users\User\PycharmProjects\DQUIC> python .\server.py
Generating 10 objects...
Generating objects complete!
Generating DQUIC socket...
DQUIC socket is up!
Waiting for requests...
Detailed client request:
Stream:5, Object:2, Actual size: 1577269
Stream:8, Object:3, Actual size: 2057245
total objects size: 3634514
Sending objects...


------------------------------------- STATES -------------------------------------

(a)+(b)+(c): Streams info
Stream: 5, Stream size: 1991 bytes, Total bytes sent: 1577269, Pace: 4430365.38 B/s, 2224.64 Packet/s
Stream: 8, Stream size: 1139 bytes, Total bytes sent: 2057245, Pace: 2282039.34 B/s, 2003.34 Packet/s

(d)+(e): Connection info:
Received data pace: 4031655.89 Bytes/s, 2006.67 Packets/s

----------------------------------------------------------------------------------

Sending finishing msg
Socket closed
PS C:\Users\User\PycharmProjects\DQUIC>
```

הרצת השרת, קבלת בקשה מהלקוח ושליחת הקבצים.
בסיום, הדפסת מדדים.

לק"י

```
PS C:\Users\User\PycharmProjects\DQUIC> python .\client.py 2
Generating DQUIC socket...
Sending request: 5:2 8:3
Waiting for response...

Start receiving
Receiving completed!

total packets received: 1809
In stream:5, object number:2 object size:1577269
In stream:8, object number:3 object size:2057245

Socket closed
PS C:\Users\User\PycharmProjects\DQUIC>
```

הרצת הלקוח עם ארגומנט 2 (יבקש מהשרת 2 קבצים כלשהם).
שליחת הבקשה וקבלת המידע.
הדפסת מדדים.

שלב הבקשה מהשרת, ניתן לראות שגודל החבילה הוא קטן (קיים גם ACK על הבקשה).
כאמור, הWireShark לא מכיר את הפרוטוקול שלנו לכן אנחנו רואים כאילו הכל נשלח על UDP (זה באמת מה שקורה)

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	UDP	64	51426 → 9999 Len=32
2	0.000229	127.0.0.1	127.0.0.1	UDP	57	9999 → 51426 Len=25

```
▸ Frame 1: 64 bytes on wire (512 bits), 64 bytes captured (512 bits) on interface \Device\NPF_Loopback, id 0
▸ Null/Loopback
▸ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▸ User Datagram Protocol, Src Port: 51426, Dst Port: 9999
▾ Data (32 bytes)
    Data: 0300000000000000420000000500000000000000000000000007353a3220383a33
    [Length: 32]
```

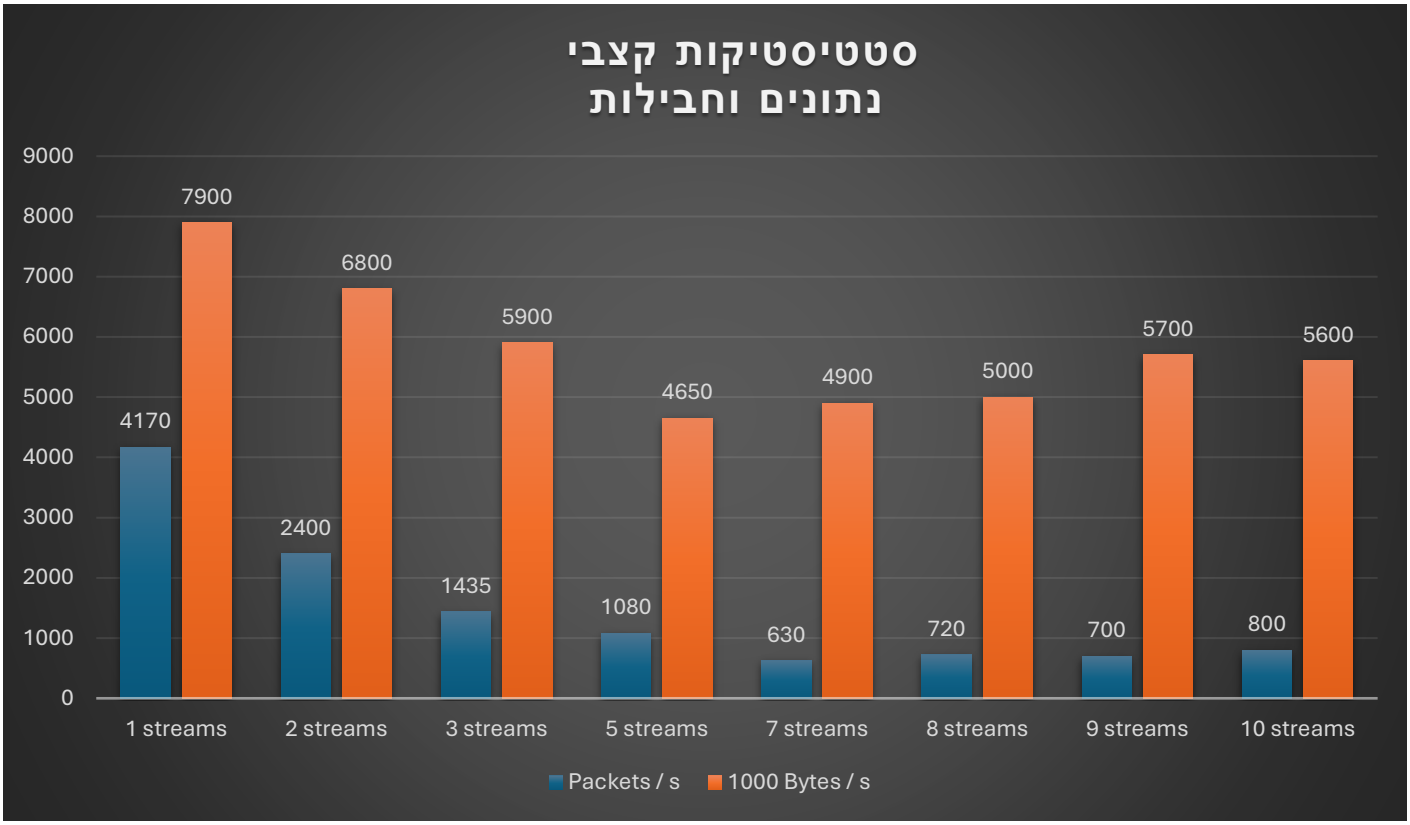
דוגמא לשתיים מהפאקטות שנשלחו ובנוסף, פאקטת הACK שהוחזרה על ידי הלקוח (בתמונה רואים רק על הפאקטה הראשונה). ניתן לראות שגודל הפאקטות נשאר זהה וזה קורה כי הגדרנו לכל סטרים גודל קבוע (רנדומי) וכל חבילה יכולה להכיל עד 7 סטרימים, לכן במקרה הזה הגודל הוא קבוע

3	0.000883	127.0.0.1	127.0.0.1	UDP	3207	9999 → 51426 Len=3175
4	0.001087	127.0.0.1	127.0.0.1	UDP	77	51426 → 9999 Len=45
5	0.001223	127.0.0.1	127.0.0.1	UDP	3207	9999 → 51426 Len=3175

```
▸ Frame 3: 3207 bytes on wire (25656 bits), 3207 bytes captured (25656 bits) on interface \Device\NPF_Loopback, id 0
▸ Null/Loopback
▸ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▸ User Datagram Protocol, Src Port: 9999, Dst Port: 51426
▾ Data (3175 bytes)
    Data [truncated]: 0300000001000000050000000500000000000000000000007c76ebdbece64184352c215f02dfec93759c5c9fc12ee1a7
    [Length: 3175]
```

הערה: קובץ ההקלטה המלא מצורף לקבצי ההגשה. ניתן לראות שאכן כמות הפאקטות שנשלחו (ללא ACK'ים) זהה במספר הפאקטות בהקלטה ובמספר הפאקטות שהתקבלו בהדפסה של צד הלקוח.

לק"י



בהתאם להוראות המשימה, הגדרנו את מספר הזרמים המקסימלי בכל חבילה ל-7. לכן ניתן לראות שעד שלא עברנו את ה-7 זרמים לחבילה, ככל שמספר הזרימות יעלה כך בהתאם קצבי שליחת החבילות והבתים ירדו. זאת מכיוון שהגדלת מספר הזרמים בכל חבילה משמעותו הגדלת מספר הבתים בכל חבילה באופן ישיר וברור שיקח יותר זמן לשלוח אותה. חשוב לציין שמכיוון שהמערכת שלנו עובדת עם מנגנון s&w הדבר מקטין עוד יותר את הקצבים.
מרגע שעברנו את את ה-7 זרמים לחבילה, הקצבים יישארו אותו דבר מכיוון שהקצבים נמדדים כמובן ביחס זמן ומכיוון שבכל חבילה יש בדיוק 7 זרמים,, לכן לאורך זמן הקצב יהיה זהה. יש לציין שיכולים להיות שינויים קלים מכמה סיבות: מצב הרשת בעת הניסוי הספציפי וגודל הזרמים שנבחר באקראיות עבור על ניסוי בנפרד. לאור זאת, שינויים קלים בין הניסוים עבור יותר מ-7 זרמים הינם זניחים.

הערות:
א. קבצי pcap ותמונות נוספות של הרצת הפרוטוקול למדידת הזמנים נמצאים בgithub.
ב. לפרוייקט הצמדנו קובץ טסטים כנדרש.
ג. לפרוייקט הוספנו קובץ README מעודכן ומפורט.