

בעלי הפרויקט:

דני איסקוב, 206530073
קורן עבדוש, 209271535
יהונתן ברוך, 211465786

החלק ה"יבש" בפרויקט

5 חסרונות/מגבלות של TCP

1. חוסר יעילות שנובע משימוש באותו אלגוריתם עבור Congestion Control וגם Reliable Delivery
TCP שואף לשלוט במדויק במספר הפקטות שנמצאות בתוך הרשת ומסתמן שהוא משתמש באותו האלגוריתם חלון עבור flow control וגם reliable delivery שזה לא אופטימלי - לדוגמא, אם פקטה הולכת לאיבוד אז חלון שלם של פקטות "יזרק" אפילו אם חלק מהם כבר הגיעו ליעד בהצלחה.
2. Head of Line Blocking - HoL
TCP מוודא שהדאטא מדולבר בסדר מסוים, אך אם פקטה אחת הולכת לאיבוד, אז כל הפקטות שאחריה יחכו עד שהפקטה שהלכה לאיבוד תשלח שוב ותתקבל ובכך להאט את התפוקה של ההעברה.
3. חיבור ראשוני איטי
לפני ש-TCP יכול לשלוח דאטא, חייבת להתקיים לחיצת יד משולשת (3-way handshake), תהליך שלוקח זמן. הדילי הזה יכול להיות בעיה, במיוחד שדרוש חיבור מהיר. בנוסף, פרוטוקולים מתקדמים של TCP יממשו גם תהליכי אימות ואבטחה לפני החיבור, מה שאפילו יאט את החיבור הראשוני עוד יותר.
(לדוגמא בעת שימוש באינטרנט כנראה שנצטרך גם לבצע לחיצת יד TLS על מנת להבטיח חיבור מאובטח)
4. Header קבוע בגודלו
ל-TCP יש Header בגודל קבוע, מה שמגביל כמה עם כמה מידע הוא יכול לעבוד בצורה יעילה, במיוחד במהירויות גבוהות.
5. שינוי כתובת IP
חיבור TCP מאופיין ע"י כתובת IP ומס' הפורט הרלוונטי. אם כתובת ה-IP תשתנה בעת חיבור, נצטרך להתחבר מחדש - מה שגורר handshake חדש וכפי שאמרנו ב-3. זה תהליך שהוא איטי.

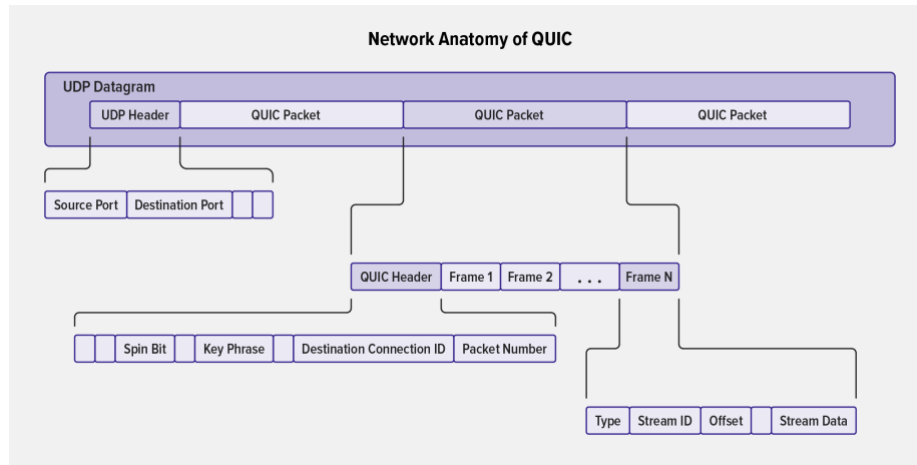
5 תפקידים שפרוטוקול תקשורת צריך למלא

1. Reliability: הפרוטוקול צריך להבטיח שהנתונים אכן מועברים ויוכל לשדר מחדש נתונים שאבדו.
2. Efficiency: הזמן שלוקח לעיבוד והעברת נתונים צריך להיות מהיר יחסית.
3. Scalability: הפרוטוקול צריך לתמוך בגדלי רשת שונים, במיוחד בכמות גדולה.

4. Secure: הפרוטוקול צריך להיות עמיד מפני גישה לא רצויה ולהגן על הנתונים שלו באופן מסוים (הצפנה).

5. Adaptive: הפרוטוקול אמור להיות מסוגל להתאים את עצמו לתנאי הרשת על מנת למקסם את הביצועים שלו.

ב TCP גודל header הוא קבוע. אך ב QUIC ישנם שני סוגים שונים של headers. במהלך יצירת החיבור, משתמש ב header גדול כדי לכלול מידע הכרחי. לאחר יצירת החיבור, QUIC עובר לתבנית header קצרה יותר שמכילה רק את המידע ההכרחי ובכך יעיל יותר. בנוסף, לכל פקטה ב QUIC ישנה מזהה ייחודי שמאפשר לשמור על סדר השידור ועוזר בתהליך שחזור הפקטה במקרה והלכה לאיבוד.

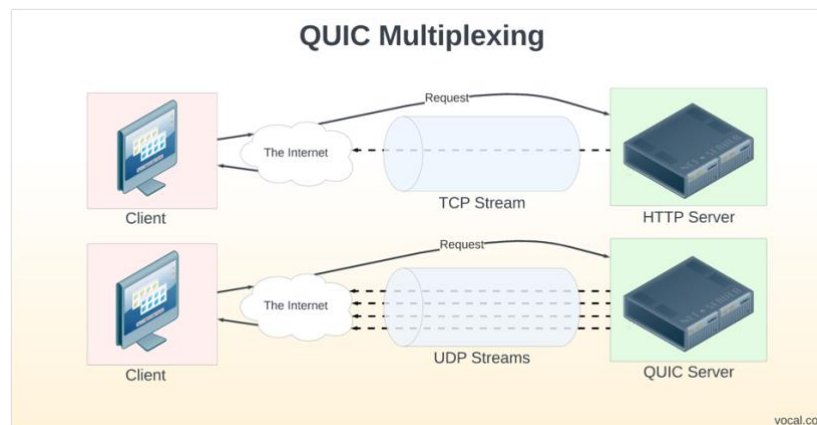


(Taken from https://www.f5.com/content/dam/f5-com/nginx-import/primer-QUIC-networking-encryption_stream-anatomy.png)

זרימות:

QUIC משלב ריבוי זרימות אל תוך השכבה התעבורתית בדומה ל-HTTP/2. ב-QUIC לכל חיבור יכול להיות מספר זרמים בו-זמנית, המזוהים ע"י stream ids ייחודיים. כל זרם כזה מתנהג בדומה לחיבור TCP כך שזרימת המידע הינה מסודרת. כל מידע בתוך זרם מפוצל לפריימים וכל פריים מזוהה לפי שילוב של id והoffset שניתן לו - מה שעוזר למנגנוני loss detection של הפרוטוקול לזהות מידע שאבד ולהשלים אותו במידת הצורך.

בנוסף, נשים לב שהשימוש בזרימות מנטרלת את בעיית ה-Head-of-Line (HOL) הקיימת ב-TCP כפי שציינו ב-1. במקרה של איבוד פקטות (packet loss) רק הזרם שבו הייתה אבדה יתעכב ואילו שאר הזרימות יזרמו כרגיל.



(Taken from <https://vocal.com/wp-content/uploads/2024/02/QUIC-Multiplexing-2048x1066.png>)

תמיכה ב-unreliable datagram delivery:

ישנן אפליקציות, במיוחד אלו שמשמשות במידע בזמן אמת, המעדיפות לשדר מידע מאובטח באופן לא אמין בתמורה למהירות. אפליקציות מסוג זה, יעדיפו להשתמש ב-UDP או ב-DTLS. QUIC גם תומך בשידור מידע מאובטח אך לא אמין באמצעות Datagram Frames. בניגוד לפריימים רגילים, אלו פריימים שלא ישודרו פעם נוספת במקרה וילכו לאיבוד. בהשוואה ל-UDP או DTLS, QUIC מבצעת לחיצת יד המאפשרת שידור מאובטח אך לא אמין.

מה QUIC עושה כאשר פקטות מגיעות באיחור או לא מגיעות בכלל?

QUIC משתמש בזיהוי אובדן מבוסס ACK בכך שכל פקטה מפוצלת למספר של פריימים וכשפקטה היא ACK, הוא מאשר את קבלת הפקטה כולה וכל הפריימים שלה מתקבלים. אם פקטה לא מקבלת ACK אך פקטה מאוחרת יותר תקבל ACK, הפקטה המוקדמת והפריימים שלה ילכו לאיבוד.

QUIC משתמש ב-2 דרכים על מנת לקבוע איבוד פקטות:

1. לפי המספר שלהן (לכל פקטה יש מזהה ייחודי)
2. עבר זמן מסוים ולא התקבל ACK עבור פריימים מסוים

הדרכים הללו מאפשרות לסידור מחדש של הפקטות ובכך הפרוטוקול מונע שידור מחדש ללא צורך ממשי בכך. וכשפקטה הולכת לאיבוד, הפריימים שלה ישודרו מחדש בדומה ל-TCP - באופן אמין ומסודר.

ה Congestion Control של QUIC

QUIC מיישם CC באופן דומה ל-TCP. כפי שצינו קודם, QUIC ימספר כל פקטה וישתמש בframes offset על מנת להבטיח אמינות. בנוגע ל-CC, QUIC משתמש ב-CC מבוסס חלון, בדומה ל-TCP. ל-QUIC אין מימוש ספציפי של אלגוריתם CC מבוסס חלון, ומשאיר את המימוש בצד הלקוח - מה שהופך את התהליך לגמיש יותר.

כדי למנוע הקטנת חלון לא רצויה, QUIC יצמצם את החלון שלו אך ורק אם יזהה עומס. עומס מזהה אם שתי פקטות אובדות ואין פקטות ביניהן המקבלות ACK. המרווח זמן בין הפקטות האבודות הללו מחושב בשילוב ה-RTT הנוכחי כדי לקבוע עומס.

מצד הלקוח, הלקוח ימתן את כמות השליחות שלו ע"י שליחת הפקטות שלו במרווחים המבוססים על ה-RTT הממוצע ובכך יופחת הסיכוי לעומס.

החלק ה"רטוב" בפרויקט - מימוש ריבוי זרימות

מבוא

החלטנו לממש את חלק 1 — ריבוי זרימות כאשר ממשנו שתי תוכניות פייתון: client.py ו-server.py. החלטנו שבשרת יהיו מאוחסנים קבצים רבים בפורמט txt. בגדלים שונים וכאשר לקוח מתחבר לשרת, השרת יעביר את ללקוח את כל הקבצי הטקסט שמאוחסנים בו. למרות שאנחנו לא מממשים ריבוי זרימות כפי שהן ב-QUIC באופן ישיר, הגישה של השרת לטיפול בהעברות קבצים במקביל מדמה את הרעיון של זרמים ב-QUIC.

צד השרת ומימוש ריבוי הזרימות

השרת יוצר סוקט על פורט מסויים ואז מאזין לחיבורים של לקוחות. עם חיבור של לקוח השרת ישלח לו את כל הקבצים שזמינים להורדה. השרת תומך בעד 10 לקוחות במקביל.

לגבי ריבוי זרימות, נזכיר שאנחנו מנסים לזרז זרימות כפי שמתואר במאמר; כל קובץ שישלח ללקוח מהווה זרם, כאשר עבור כל זרם השרת יקצה thread בודד מה שמדמה את הרעיון של מספר זרימות תחת חיבור אחד כפי שהן ממומשות ב-QUIC.

השימוש בthreads שונים מייצל את הביצועים בצד השרת ומנטרל את בעיית HoL אך גם מביא אותנו לבעיה הבאה: הזרמים פועלים במקביל אחד לשני וגורמים למידע להישלח ללקוח בצורה לא מסודרת. נסביר כיצד פתרנו את זה בצד הלקוח בהמשך.

חשוב לדעת שתקשורת לקוח-שרת מוגבלת בדרך כלל על ידי רוחב הפס של הרשת. העברת מספר קבצים במקביל על פני threads שונים אינה מגדילה את רוחב הפס (המחשבים המודרניים יודעים למקסם את רוחב הפס..). ומה שקורה בפועל בפרויקט שלנו זה שהשרת מחלק את רוחב הפס הזמין בין threads הקיימים.

צד הלקוח

צד הלקוח הוא פשוט יחסית, הלקוח מתחבר לשרת בעזרת כתובת ופורט ומבקש את כל הקבצים שמאוחסנים בו. כפי שצינו מקודם, ישנה בעיה שצצה בעת שימוש במספר threads בצד השרת עבור כל קובץ: המידע שהלקוח יקבל אינו מסודר; ייתכן שהוא יקבל חבילה שעליה יש מידע ששייך לקובץ מס' 1 אחרי זה חבילה עם מידע ששייך לקובץ מס' 3 ואחרי זה שוב חבילה שקשורה לקובץ מס' 1.

כדי לפתור את זה, השרת שולח כל פריים בצורה הבאה:

file name	frame id	frame size	file data
-----------	----------	------------	-----------

כאשר החלק הירוק הוא header והחלק הסגול הוא data.

גודל header הוא קבוע בגודל 43 בייטים. הלקוח תחילה ישלוח בדיוק 43 בייטים מהסוקט שלו, ואז ישלוח שוב מידע בגודל frame_size מהסוקט (frame_size יוגרל בין 1000 ו2000 בייטים עבור כל קובץ בצד השרת). ולבסוף, הלקוח יסדר את המידע שהוא קיבל בצד שלו באמצעות file_name והframe_id. גישה זו דומה לאופן שבו QUIC מנהלת את המידע בתוך הזרמים שלה; באמצעות הרכבה נכונה בצד הלקוח בהתבסס על frame_id.

הפקטה הראשונה

```
E@ @ %n(U
P$R/file1.txt
000000117054Xo3eCqTKAJRXCjV6nwZ8mC1rjGC0MLEAIhQ7GWRDTYAthvnmDymbEKVI007L1dtysbC71oGcaafcMaHpQMcS7VYo5SAyESz2dzUsoYHe9YAwbL1L1J1DWI3vuQrZzWaICPToacIX56bEBdZr4LQeUeVPnc2824od7z9Am0y81dULyTbbjIm
ifI88gyDwfzGLaaAW7y01003MaVTEW8UdgSAuBK027z2znI3bSha8Jyyq1KvtzgVaQ07Z1RUH0eKNfAEGr3FcFLj0rDo2jfxMLcvFnRPpFb00c1LUSAmFDegefFwUQsvCZcstw42a1AT9PeFwgr48hRVZ1XaqQjHk2cIfuyA3UwVfoNb70QVrct6xsTnfp1
r6oPLIYLNjKq1F206CsJaK08m0ZNZGGWd83XGeK43hC7oJXmXbYsYYatPIB3y6rtTg130jh367wUcPEXLD1RNUphGPNQkvdaESFKEfx5MLDZ2mc8Wk8wo1MKoyxkQ9FmRx9Gc5mId4kqu1ajpX6kMQ220bywx2pbvys70ogT8Fck8hhK2LTyvDjfiwIVwG
DiwRvWench1LhG1JHxXaYSROQujeh4ZVphIwLsg5RFcr5GdX8wK2wXe4uISxTjfm18J0yyTpmMiiQgVtj7r9N2qCqAvR588qogMoMcL3EEtsfa2zUppqy1dJG7PMTreVWncpGvFUVWmHlms7KdVe7a1MXx7e0VGvNlXc0p4W0aJE50a8BfYPPpPH5ezKZtxAhKMa
mCaLSzQtEjI1LPBYhw7sup34DrBv11f5sJHcC9bGyQX5MNJ410yqQYqKvB7fEhpNTEnBgg2eNlyTSRpbXXShV8duXqLzqM5XpC4Ypxd1uRqkqepcfJFrB1hIYvSUrDjON1WdZqxAZ6QpZTnWd25pLAe7ZjtJHe4pJ29D09R8G5EuZxgE28Lg8kCDapeqyafFE2t
QKEudvP4yhFCyKpyKsmznT0tD7kmBNWQXnsI6L14M4rK3HMDrtqiaAMW1VSjBw6HCNUY81SfFue1eQYwSnNtP0d6G8rfHiebegXvdbW6WqVajhap13Gq3V0xbcmxHsnbfJXJ3fSaET5CdH3dN2WmPEwCguw6985x0YEY3YQHd74d1PHLHC8x3V06o5mFLRw08Bv
BMAQoQwdfuak8B8CUHaPaeJH36ptRjhMRpFkpPz0CwX7495f1uKUJCVmIPyMz61vMbX9PdKVE14GMFQC6ZbH42CgZS0ZLva0rF5Rpwj1r0U2U0nSKEt27eDNhZWh8csF0INvfc7wo7wL4uIrfDro5SYMjJ36zzJr0t19Qa6sdr5Z6FoFbyVhbsc83T
Zt9rN2ZvFdsNGW0HgFJ5x8QJXiEEAc1U6Hh1ZwSUSPL1widwPY77CEJ1qL800kZszKU41d818f0y9g11RtD19Uf08TSEK8FJBe7fJ199T0zfd1ZIM1Nw7QV0f88C0wFLrCKImYd1L3zzLTlCpXt2R1VqH10094HPYSNBmuSYQh6iRFJufNanqnIIS5oe0LLFg
bKt1KWCm8pdhplYFT2soGH8tIY1AC50Kry1A7fIgs5nF4296xQqbL6UTBf0fEpMBTaD9j14wydbGnr8ukA1VfxSSw2P1u3wJb4qBfM7fkyFNUlw8RPMcBgIYHf615X1d18uFfINaxlG15wEoLe7HqrXKFsNvKhj0b0
```

(הפקטה נלקחה מהקלטה 3_files_test, מס' פקטה 1)

כבר בפקטה הראשונה, ה33 בייטים הראשונים מהווים את header (מסומן בצהוב בתמונה למעלה). client יקרא תחילה 43 בייטים בדיוק מהסוקט, ואז ישלוח שוב מהסוקט מידע בגודל 1705 וידע לקשר אותו לfile1.txt. לרוב בהקלטות, עבור כל פקטה, header יהיה בתחילת המידע. לפעמים יהיו מקרים שנראה כאילו המידע אכן מפוצל וללא header, לדוגמא:

עָוֹ, נְפִילָה
P(RZyDacDY8gOhHk6eRh1EzUuLPT7kVQ0N)pkSkAGrS6n6xZAgSITAQirR0HUyq4YgMLzbSTg9bbddEy5fHmF1F1DfJgAp8cYfwaA7eJlcZbVJpOFJYNIlg54ZTIPU9RnB38A18fReLbr9r518mw0AnY7gh0yLuF7QhWmaJtSp6i2S6GtKh79G6iXfS5oZz
PmVpB1M4XuLw0xZy7G6oLChSsnCrQW0iDGC0BAdhJTLBS7Wg19S70W1kxfZphjZFFthb\jIETowPwC31XhxbkHqH0z5Noudxwve0DVUv1tnH37mf1uTung10Pq7PqRHMqEvS2FFB1TG68TvmQ17U7W06KtcxyfYuaIpgdcmhBQKwhehWS0yJp1B
0AyrTianasoXTXb1G03hbBc1K2bZLWwbMaJqES6PXCkAcKkEbFhxVpKvuyJQVENZB54aBHzv613Jf2a0IaogZ1GYAFHqHephfV50q3FCXAKLa2mVLeThsJy0oL9JmEzYwUfKc080AMV4K48RvkqgDT0FY7n3C1qcdp0T9B3n1f18YB7U0t8mN2J
v3Nts2WmyfJdZ0Hqpg90u7aqb0y0Hf6p57CYkEL5rC1XyxwVcCgmabaeYMHhKHLgyugzgY3RgyKEM0BAAqgJ0C19WaqFwrrfUUF6p6j0m0ymhkrh

הפקטה נלקחה מההקלטה 3_files_test_3, מס' פקטה (100)

אין לנו שליטה במתי הפקטה תחתוך את הheader, בגלל שאנחנו משתשים בTCP כבסים.
אך בפועל זה לא משפיע על הלוקו - הheader כבר נשלח ומיד אחריו יגיע המידע נוסף הזה.

כיכוד ריבוי זרימות מתבטא ברשת?

P(RZyDacDY8gOhHk6eRh1EzUuLPT7kVQ0N)pkSkAGrS6n6xZAgSITAQirR0HUyq4YgMLzbSTg9bbddEy5fHmF1F1DfJgAp8cYfwaA7eJlcZbVJpOFJYNIlg54ZTIPU9RnB38A18fReLbr9r518mw0AnY7gh0yLuF7QhWmaJtSp6i2S6GtKh79G6iXfS5oZz
PmVpB1M4XuLw0xZy7G6oLChSsnCrQW0iDGC0BAdhJTLBS7Wg19S70W1kxfZphjZFFthb\jIETowPwC31XhxbkHqH0z5Noudxwve0DVUv1tnH37mf1uTung10Pq7PqRHMqEvS2FFB1TG68TvmQ17U7W06KtcxyfYuaIpgdcmhBQKwhehWS0yJp1B
0AyrTianasoXTXb1G03hbBc1K2bZLWwbMaJqES6PXCkAcKkEbFhxVpKvuyJQVENZB54aBHzv613Jf2a0IaogZ1GYAFHqHephfV50q3FCXAKLa2mVLeThsJy0oL9JmEzYwUfKc080AMV4K48RvkqgDT0FY7n3C1qcdp0T9B3n1f18YB7U0t8mN2J
v3Nts2WmyfJdZ0Hqpg90u7aqb0y0Hf6p57CYkEL5rC1XyxwVcCgmabaeYMHhKHLgyugzgY3RgyKEM0BAAqgJ0C19WaqFwrrfUUF6p6j0m0ymhkrh

הפקטה נלקחה מההקלטה 3_files_test_3, מס' פקטה (99)

כאן כבר נראות את השימוש בריבוי זרימות - על גבי פקטה אחת, נמצא מידע ששייך למס' קבצים שונים, באופן
לא מסודר. הלוקו יודע איך לסדר את הפריימים האלה אצלו ולשדר את הפקטה הדרושה בכוכן מרוויחים.
ניצול מקסימלי של כל פקטה בפועל.

שימוש בתקשורת בזמן 0-RTT

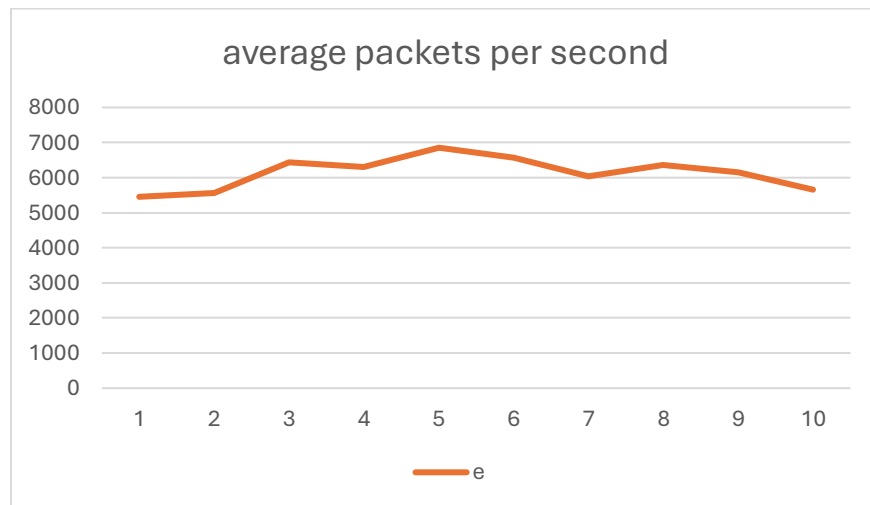
השרת שלנו מחקה תקשורת 0-RTT על ידי שליחת תונים ביחד כבר בחבילה הראשונה, ללא צורך בלחיצת יד.
מקטע זה מראה כיצד השרת מנצל את הזמן להחליף את השרת הראשון ודומה הודעה

ניסויי על מספר זרימות בין 1 ל-10

ראשית, לצורך אחידות הניסוי, יצרנו 10 קבצים זהים בגודל 4mb שאותם נעביר בפרוטוקול שלנו. תחילה, נציג את הנתונים שאספנו:

# of streams (files)	1	2	3	4	5	6	7	8	9	10
d	8294456	8604580	8206837	8633710	8737882	8798714	9015263	9125150	8889883	9528417
e	5454	5560	6430	6298	6853	6573	6042	6363	6155	5648

הגרף של e כפונקציה של מספר זרימות

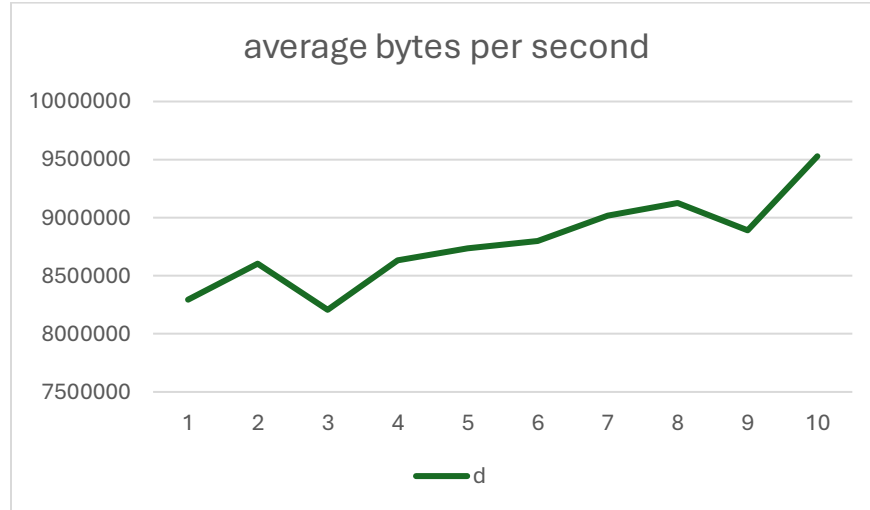


סיכום תוצאות הניסוי - האם יש קשר בין מספר הזרימות למספר הפקטות הממוצע לשנייה?

ניתן לראות תחילה שככול שנוספים זרמים, קצב מספר הפקטות הממוצע לשנייה משתפר. לאחר מכן, כאשר מגיעים למספר זרימות בגודל 5-6 נראה שקצב הרשת מגיע לסוג של מישור ולבסוף, הוספת זרמים נוספים (7-10) גורמות לירידה במספר הפקטות הממוצע לשנייה.

לסיכום, מסתמן שהוספת עד כ-5 זרימות משפרת את מספר הפקטות הממוצע לשנייה יביא אותנו לתוצאות האופטימליות ביותר. לאחר מכן, הוספת מס' קטן של זרימות נוספות לא תשפיע אך ככול שנוסיף יותר ויותר זרימות כך ביצועי הרשת ידעכו, ככול הנראה בגלל שהמערכת כבר הגיעה לגבולה ב~5 זרימות וכל זרימה לאחר מכן תגרום לעומס.

הגרף של d כפונקציה של מספר זרימות



סיכום תוצאות הניסוי - האם יש קשר בין מספר הזרימות למספר הבייטים הממוצע לשנייה?

נראה שהיו נפילות ספציפיות כאשר הוספנו זרימה נוספת בהינתן 2 זרימות ו-8 זרימות. קשה לקבוע מה הסיבה לנפילות אלו וסביר להניח שמדובר בנפילות מקריות בגלל תנאי רשת, כיוון שלא ניתן להניח איזושהו דפוס. סך הכל, מה שכן ניתן לראות בניסוי המצומצם שלנו זה שככול שמוסיפים יותר זרימות, קצב הבייטים הממוצע לשנייה עולה - כך מסתמן שביצועי הרשת ישתפרו (נוכל להעביר יותר מידע בכל שנייה). נצטרך לעשות ניסוי רחב יותר עם מספר גדול יותר של זרימות על מנת לקבוע מה הוא מספר הזרימות האופטימלי.

לקינות: אי-יעילות ידועות בפרויקט שלנו

- אי-יעילות בגודל הheader: בכל פקטה, אנו שולחים עימה סוג של header שמציין את שם הקובץ, גודל החבילה שהוגרל ע"י השרת ועוד. חלק מהפרטים אלו הם קבועים ולכן ניתן לפתור את האי-יעילות הזאת בקלות באמצעות שליחת שם הקובץ וגודל החבילה רק בפקטה הראשונה ואילו בפקטות הבאות להשתמש בheader קטן יותר שמציין רק את מזהה הקובץ והמידע בפקטה - כפי שקורה בפועל בפרוטוקול QUIC.
- כמות מרובה של קבצים: כאשר יהיו לנו כמות גדולה של קבצים, השרת יקצה לכל קובץ thread משלו מה שיצא מכלל שליטה באופן מהיר מאוד וכמובן שזה לא אופטימלי.
- שימוש בTCP כפרוטוקול: כמובן שלא התבקשנו לממש QUIC לגמרי אך חשוב לציין שהשימוש בTCP מגביל אותנו מאוד בהמון מובנים וגם בפועל QUIC בנוי מעל UDP.

נספחים

מרבית התשובות שלנו בחלק היבש מתבססות על המאמר "A Quick Look at QUIC"
<https://web.cs.ucla.edu/~lixia/papers/UnderstandQUIC.pdf>
<https://en.wikipedia.org/wiki/QUIC>
<https://blog.cloudflare.com/the-road-to-quic>
[/https://www.chromium.org/quic](https://www.chromium.org/quic)
ChatGPT