פרוייקט גמר ברשתות תקשורת

תומר שור – 325511541

אדם סין – 322453689

מתן קליין – 214968240

נועם לוין – 326662574

(אם יש בעיה כלשהי עם קובץ בזיפ, הנה קישור לגיט:
https://github.com/AdamSinale/Networking_final_project.git)

שאלות פתוחות:

שאלה 1 –

א. בעיה ראשונה, הקשר בין CC (Congestion control) לבין שליחה אמינה: מטרת ה-CC הינה לשלוט בכמות הפאקטות שמועברות ברשת וב-TCP הוא בעצם "ננעל" על מימוש של חלון שליחה שתוכנן במקור לצד המקבל בכדי לבצע בקרת זרימה על הצד השולח ובכך לוודא שיש שליחה אמינה. מכיוון שהוא "נעול" על מימוש ספציפי, כל איבוד של חבילה (פאקטה) יגביל את ההתקדמות והגדילה של החלון עד שהחבילות שאבדו יגיעו מחדש. בזמן שאנו נזהה את האובדן ונשלח מחדש את המידע, רוב החבילות שבחלון היו יכולות כבר להתקבל אצל הצד המקבל, כלומר "יצאו מהרשת". נתאר דוגמה שתמחיש לנו את הבעיה: נניח שהמקבל קובע את חלון בקרת הזרימה שלו ל-8 חבילות ושגם חלון ה-CC הינו בגודל 8. כאשר חבילה מספר 1 תגיע החלון של השליחה יתקדם לטווח שבין חבילה 2 ל-9. אם למשל כעת חבילה 2 נאבדה אז זה מונע מאיתנו להמשיך לקדם את החלון גם אם חבילות 3 עד 9 הגיעו ולא אבדו בדרך. מכך נוצר מצב בו כמות החבילות שלא שלחנו עליהן ACK לא משקפת את כמות החבילות שנמצאות ברשת שכן חלקן התקבלו כבר ולא צריך לשלוח אותן מחדש.

ב. בעיה שנייה, Head of line blocking: בגלל ש-TCP מבטיח העברה רציפה של מידע, האובדן של חבילה אחת בלבד תחסום את הקבלה של כל החבילות הבאות עד שהחבילה שאבדה תגיע מחדש. נסתכל על דוגמה שתמחיש את הבעיה: נניח ששלחנו את חבילות 1 עד 6. חבילה 2 הגיעה אל המקבל אך חבילה 1 נאבדה בדרך. המידע שיש בחבילה 2 לא יכול לעבור לשיכבת האפליקציה עד שחבילה 1 תשלח מחדש, תתקבל ותישלח קודם לשיכבת האפליקציה כדי להבטיח העברה ששומרת על הסדר. הבעיה הזו דומה מאוד לבעיה הראשונה שהצגנו אך פה התייחסנו לבעיה שמתרחשת אצל המקבל ומקודם התייחסנו לבעיה אצל השולח.

ג. עיכוב (delay) בעקבות הקמת קשר: ב-TCP כל פעם ששתי נקודות קצה רוצות לתקשר ביניהן חייבת להתקיים לחיצת יד משולשת. אם בנוסף לכך נרצה לאבטח את הקשר על ידי TLS אנו נזדקק ל-RT נוסף בכדי להחליף אישורי אבטחה בין שתי הנקודות. הקמת הקשר בצורה כזו לוקחת זמן רב שמעכב את שליחת המידע.

ד. **הגבלות שנובעות מ-header קבוע ל-TCP:** ה-header של TCP הינו אוסף של שדות שקבועים בגודלם. 3 שדות ספציפיים ב-header של TCP הושפעו ישירות מהעלייה במהירות הרשת לאורך זמן – #seq ו-ACK ששניהם בגודל של 4 בתים, וגודל חלון בקרת הזרימה שגודלו הינו 2 בתים. הגדלים הקטנים האלו מגבילים את הביצועים של TCP במהירות גבוהה ברשת. ה-#seq וה-ACK "נעטפים" מהר מידי מה שהופך אותם לכבר לא ייחודיים, וגודל חלון הזרימה מגביל בצורה ישירה את תפוקת החיבור של TCP אשר חסומה על ידי גודל החלון כפול ה-RTT.

ה. **מזהה ייחודי לחיבור ולכתובת ה-IP:** כתובת ה-IP של כל אחת מנקודות הקצה בחיבור TCP עלולה להשתנות המהלך חייו של החיבור בעקבות שלל סיבות – אירוח של מספר משתמשים, ניידות או ריצה מאחורי NAT. בגלל ש-TCP משתמש בשילוב של כתובות ה-IP ומספרי ה-port של שתי נקודות הקצה כמזהה לחיבור, כל שינוי בכתובת ה-IP של אחת מנקודות הקצה תשבור את החיבור הקיים מה שיגרום לכל המידע שהועבר על כה "להיזרק". בכדי להתאושש ממצב זה אנו נזדקק ללחיצת יד משולשת חדשה בכדי לפתוח קשר חדש.

שאלה 2 –

א. **הגדרת מזהה לחיבור של הקשר ומזהה למידע שעובר –** על מנת שנדע לזהות ולהבדיל בין חיבורים שונים ובין מידע שונה.

ב. **ניהול חיבורי התעבורה –** שזה בין היתר להגדיר מזהה חיבור ייחודי שמקשר בין 2 רכיבי קצה, שליטה במידע שמוחלף, הגדרת מצב חיבור וניתוק ותמיכה בשינוי ה- IP של המארח.

ג. **העברת מידע באופן אמין –** מזהה מידע ייחודי על מנת להעביר את המידע באופן אמין בין רכיבי הקצה, ושליטה בחלון זרימה בשביל העברה אמינה על מנת שלא ניתקע בבעיה שהפאקטה הראשונה בתור נתקעת וע"י כך מונעת מפאקטות אחרות שאחריה לעבור עיבוד או להישלח(נקראת head of line blocking problem).

ד. **בקרת עומסים –** שליטה שבפאקטות שבתוך הרשת על מנת שלא להעמיס על רכיבי הקצה ועל הרשת עצמה.

ה. **אבטחה –** חיבורים מוצפנים והעברת מידע באופן מאובטח.

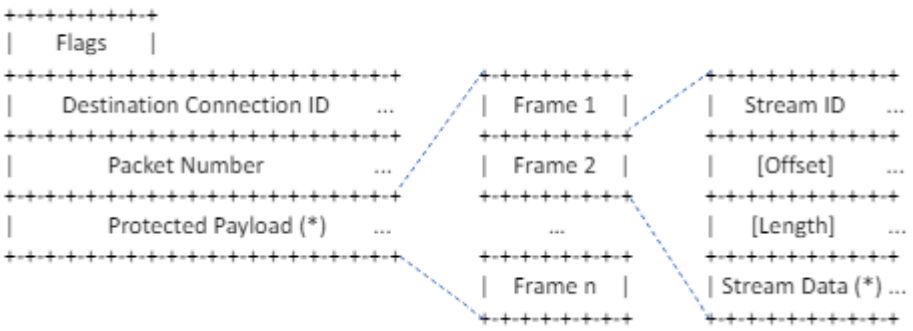
שאלה 3 –

הפרוטוקול QUIC משלב העברת מידע ולחיצת יד ובכך משיג את המידע הנדרש ב-RTT אחד בלבד. למעשה, זה כולל ביצוע אמין של החלפת מפתח (בין נקודות הקצה) של TLS 1.3 באותו הזמן עם החלפה אמינה של פרמטרים לטובת העברת מידע. הצורה הזו של לחיצת היד מקטינה את העיכוב בהקמת קשר מאובטח כמו שדיברנו עליו בליחת היד של TCP. בעוד ש-TCP מפריד בין לחיצת היד לבין ביסוס האבטחה בין שתי נקודות הקצה, אשר גורר כמו שציינו בשאלה 1 לפחות שני RTT, QUIC מקטין זאת ל-RTT אחד. QUIC משתמש בפאקטה (חבילה) התחלתית בכדי לדון ב-ConnectionID על מנת לפתוח חיבור (קשר) חדש. כל נקודת קצה תאכלס את השדה שלה של ה-C-ID בערך שנבחר וערך זה ישמש כל צד כ-ID ליעד בשליחת הפאקטות בהמשך. בקבלת הפאקטה ההתחלתית השרת יכול לבחור לאמת את כתובת הלקוח בעזרת שליחה של פאקטה חוזרת (Retry packet) שמכילה token רנדומלי, אשר אמור להתקבל בחזרה על ידי הלקוח בפאקטה התחלתית חדשה על מנת להמשיך בתהליך לחיצת היד. גם ההודעות שקשורות ל-TCP 1.3 מוטמעות בפאקטות ההתחלתיות כך שהן יכולות לשמש כביסוס של "סוד משותף" על מנת לשמור על אבטחה ואמינות של פאקטות עתידיות שישלחו, ב-RTT יחיד. ה-C-ID שנבחר יכלל בפרמטרים של ההעברה והוא יאומת במהלך תהליך לחיצת היד של ה-TLS. בכך ש-QUIC דן על ה-C-ID, הוא תומך ביצירה אמינה של קשר חדש בצורה דומה לפונקציונאליות שאנו מקבלים מ-TCP. QUIC מאפשר ללקוח לשלוח ב-0-RTT מידע מוצפן בפאקטה הראשונה שהוא שולח על ידי שימוש חוזר בפרמטרים מהחיבור הקודם, ומפתח ל-TLS 1.3 שנחשף מוקדם יותר על ידי השרת. בעיה קטנה היא שמידע שמועבר ב-0-RTT אינו מוגן מפני התקפות replay. בתמיכה בשליחה של מידע ב-0-RTT, QUIC יכול גם להתמודד עם מקרים בהם נדרש T/TCP.

שאלה 4 –

ל-QUIC יש שני סוגים של headers. סוג אחד בשביל הקמת החיבור וסוג שני (וקצר יותר) לאחר הקמת החיבור ("לחיצת יד").

להלן תמונה של הפאקטה הקצרה יותר-



כלומר, בכל פאקטה אפשר להכניס frame אחד או יותר.

כמו כן, הפאקטה ששולחת selective ACK יכולה לשלוח עד 256 ACK-ים לעומת 3 ב TCP.

השיפורים שבאים לידי ביטוי בין היתר: הפאקטה ששולחים אחרי שהקשר נוצר הרבה יותר קצרה מב-TCP ובכך מונעת שליחה מבוזבזת של מידע. הפאקטה תומכת ב-streams ובכך מונעת את בעיית ה-Head of line.

שאלה 5 –

ראשית, נתחיל בזיהוי של איחור או איבוד ב-QUIC על ידי ACK. כל חבילה של QUIC מכילה בתוכה "frames" כך שאנו יכולים להתייחס לכל "frame" כפאקטת IP. QUIC מזהה איבוד חבילות על בסיס פאקטות אלו. על כל חבילה שקיבלה ACK, כל ה-"frames" שהיו באותה החבילה נחשבים ככאלו שהגיעו בהצלחה. "frames" נחשבים ככאלו שאבדו אם החבילה שבתוכה הם נשלחו עדיין לא קיבלה ACK כאשר חבילה שנשלחה אחרי כבר קיבלה ואנו עוברים איזה סף מסויים. ל-QUIC יש שני סוגים של סף לקביעת איבוד חבילות – אחד, על פי מספר החבילה: כל החבילות שנשלחו והמספר הסידורי שלהן (#seq) קטן בכמות מסויימת מהחבילה שקיבלנו עלייה ACK. שתיים, על פי זמן: החבילה שנשלחה כבר נשלחה לפני יותר מה-RTT המקסימלי שמדדנו על כה, לפני החבילה שכבר קיבלנו עליה ACK. סוגי הסף האלו מספקים לנו קצת "זמן של חסד" לפני קביעה של איבוד חבילה בכדי שלא יווצר לנו מצב בו אנו שולחים מחדש חבילות ללא צורך אמיתי. בנוסף הם באים גם למנוע ירידה בביצועים שנגרמת מה-CC כאשר יש זיהוי של איבוד חבילה. בכדי לזהות איבוד של "חבילות זנב", QUIC מאתחל טיימר בכל פעם שחבילה שאמורה לקבל ACK נשלחת, כאשר הוא כולל את ה-RTT המשוער, את השונות בין דגימות ה-RTT, ואת הזמן המקסימלי שהמקבל עשוי לחכות עד שהוא שולח ACK. כאשר הטיימר שלנו מתאפס (נגמר), השולח יישלח חבילה חדשה שדורשת ACK לבדיקה כך שאפשר גם כבר להתחיל לכלול בה מידע שאנו צריכים לשלוח שוב ובכך להוריד את כמות השליחות מחדש. כעת נראה כיצד QUIC מתאושש מאיבוד חבילות. לאחר שזוהה איבוד, ה-"frames" האבודים מוכנסים לחבילה חדשה שתקבל מספר שונה ללא קשר לחבילות שנאבדו. סך הכל אנו רואים ש-QUIC מספק לנו סדר אמין של שליחה וקבלה של מידע בדומה למה ש-TCP מספק לנו.

שאלה 6 –

QUIC משתמש במספר פאקטה לבקרת עומסים ול-offset ב-frames של ה-streams לצורך בקרת אמינות. כמו TCP יש ל QUIC חלון שמגביל את המקסימום בתים שהשולח יכול להעביר בכל רגע נתון. QUIC לא מחפש ליצור אלגוריתמים של בקרת עומסים או לממש קיימים, הוא מספק קווים כלליים לבקרת עומסים ונותנת לשולח חופש למשש בעצמו. בשביל למנוע צמצום לא הכרחי של החלון הוא מקריס את החלון רק אם הוא מזהה עומס קבוע: במידה ושני פאקטות אבדו וגם אף אחת מהחבילות שנשלחו ביניהם לא אושרו. השולח יאט את השליחה בשביל לצמצם את הסיכויים לגרימת עומס בטווח הקצר ע"י שליחת הפאקטות שיעלו על הזמן של החישובים שמבוססים על ה RTT הממוצע, גודל החלון וגודל הפאקטה.

חלק "רטוב"

ראשית נסביר את מימוש ודרך עבודת הקוד שלנו.

חבילת QUIC:

ישנם 4 אובייקטים:

דגלים - 4 ביטים שמסמלים FIN,DATA,SYN,ACK. יש לה גם פונקציית סריאליזציה דסריאליזציה שיהפכו את האובייקט ל4 ביטים המייצגים את הנתונים וידעו להחזיר אותם מביטים לאובייקט זהה.

Header – יש בו אובייקט דגלים בנוסף למספר הconnection id, ומספר החבילה. בנוסף הוא עושה סריאליזציה על ידי קריאה לסריאליזציה של הדגלים, והוספת מספר מוגדר של כ4 ביטים לכל אחד משאר הנתונים שלו. בדסריאליזציה הוא יקרא את מספר הביטים המוגדר ויצור מהם את האובייקט מחדש(9 בתים קבועים)

Frame – מכיל את מספר הזרימה, הoffset של הנתונים(מקומם בקובץ), אורך הנתונים בחבילה, והנתונים עצמם. כאן בסריאליזציה אנו מגדירים את גודל מספר הזרימה להיות 2 בתים, ומספר הבתים של offset ואורך תלוי בהאם החבילה מוגדרת עם data = 0. אם כן הגודל יהיה 2 ויספיק, במקרה אחר יגדיר 20 בתים כאורך ובכך יאפשר חבילת עם מידע באורך 1024 עד 2047. כך הוא ידע איך לעשות את הדסריאליזציה.

QUIC – מכיל באותו Header וpayload שמוגדר להיות מערך פריימים. כך בסריאליזציה ודסריאליזציה יקרא לפונקציות אלה בשאר האובייקטים.

השולח:

בנאי- השולח מקבל PORT,IP ויוצר גם SOCKET.

פונקציית handshake - שתיצור חבילת QUIC עם SYN=1 דלוק, והודעת "שלום" קצרה אשר נשלחת אל המקבל, הפונקציה קוראת ל()wait_for_ack.

פונקציית wait_for_ack – תחכה לקבל מהשולח חבילת ACK/SYN ACK במקרה של response/handshake, בהתאמה. היא תעשה לו דסריאליזציה ובמידה ואין ack או נגמר הזמן המוגדר תודיע בצורת false.

פונקציית generate_data_sets – תקבל מספר קבצים ליצור, מספר גודל כל קובץ, ותיצור מערך עם נתונים רנדומלים לשם שליחה.

פונקציית udp_send – תקבל את הנתונים, תוודא שאנחנו באמת שולחים קבצים לא ריקים(מחרוזות בגודל 1+)ת וכל עוד לא שלחנו את כולם, תיצור חבילות לשלוח למקבל (create_packet), ותחכה בין כל חבילה מספר מוגדר של זמן (0.0005)

פונקציית create_packet – מקבלת offsets, נתונים, ומספר חבילה, יוצרת ומחזירה חבילה חדשה של נתונים, בו היא שולחת 60% ממספר הקבצים שנשארו באופן רנדומלי. היא בודקת ש frame מסוים שולח את כל המקום המוגדר לו ובמידה ולא, תקרא לadd_to_min_frame שיוסיף לפריים שנשאר הכי מאחורה נתונים כמספר המקום הנשאר בחבילה. פונקציית יצירת החבילה תעדכן בהתאם את הנתונים שקיבלה ובמידה וזיהתה שסיימה עם קובץ, תמחק אותו.

פונקציית finish_connection – תקבל את מספר החבילה הסופית, תיצור ותשלח חבילה עם fin=1 והודעה קצרה של הפרדות.

פונקציית start – מחברת את שאר הפונקציות לשלם הנוחות ולדאוג שאין טעויות מהמשתמש, יעשה handshake, ייצור את הנתונים וישלח אותם. בסוף ישלח חבילת סיום, יסגור את הקשר ויחזיר את הנתונים כך שנוכל להשוות את הנתונים שנוצרו והנתונים שקיבל המקבל.

המקבל:

בנאי – מקבל host, port ויוצר סוקט חדש ומערך לשמירת קבצים

פונקציית listen – תחכה לhandsake בפונקציית wait_for_connection. אם הכל היה תקין, תתחיל לקבל חבילות (בעזרת פונקציית receive שתחכה לחבילה, תעשה לה דסריאליזציה ותחזיר אותה) ולמדוד את הזמן שלקח לה לחשב כל חבילה שהגיעה בנוסף לפרטים שלה. כמובן שגם תשמור את הנתונים שנשלחו. במידה והגיעה חבילת עם מספר זרימה חדש, היא תקצה מקום נוסף במערך ותשים את הנתונים שם. היא תעשה אותו דבר למערכים ששומרים מידע על כל זרימה. לכל חבילה היא תשלח חבילת ack קטנה(שאת השולח לא באמת מעניינת). כאשר תקבל חבילת סיום, תסגור את החיבור ותקרא לprint_stats עם הנתונים ששמרה שידפיס ויחזיר את הנתונים.

פונקציית wait_for_connection – תחכה פרק זמן מוגדר לחבילת syn. אם קיבלה תחזיר חבילת syn ack ותתחיל להאזין, אם לא תקבל או שנגמר הזמן תחזיר false שיסיים את ההאזנה

Main:

בעזרת threads, יפעיל במקביל את השולח והמקבל, ויריץ את התוכניות בצורה הנדרשת בעזרת לולאה כאשר בכל פעם הוא שולח לstart מספר אחר של קבצים ליצור (1-10). הוא ישמור במערכים את מספר הביטים הממוצע לשנייה הכולל וכנ"ל עם מספר החבילות, וידפיס בסיום הלולאה את הגרפים שנראה בהמשך. הוא גם ייצור קובץ טקסט עם הנתונים שהגרפים משתמשים בהם לכל הרצה.

הטסטים:

ניסינו להתייחס למקרים שהקוד הרגיל לא מתייחס אליהם, הוא שולח מספר של קבצים בגודל זהה ומצורת מחרוזת. הטסטים בודקים מקרים בהם יש טיפוסים שאנו לא מוכנים לקבל כקובץ כמו מספרים וכל טיפוס אחר.

הטסטים גם בודקים מקרה בו הקובץ הנשלח הוא מחרוזת ריקה שלא אמורה להישלח.

רצינו לבדוק גם מקרים בהם הקבצים הם בגדלים שונים, חלקם בגדלים שמתחלקים אחד בשני וחלקם לא.

לבסוף ערבבנו בין טיפוסים וגדלים שונים כדי לוודא שאין מערך מידע שלא יעבור במידה וצריך, ושמערך עם קבצים ריקים או מערך ריק לא יפיל את הקוד

נציג לפניכם את הסטטיסטיקות עם כמה דגשים:

1) בשלבי המימוש של השולח והמקבל, נתקלנו בבעיה הנובעת מחוסר התקשורת בין השולח והמקבל:

בעוד המקבל מחכה לחבילה, מקבל אותה ועושה פעולות של שמירת הנתונים במקום הנחוץ, וחישוב סטטיסטיקות לפני שיוכל לקבל את החבילה הבאה,

השולח ממשיך לשלוח חבילות בקצב שגורר חבילות רבות שנשלחו למקבל אבל הוא היה עסוק מכדי לקבל אותם. נוצר מצב של איבוד של כ65%~ מהחבילות!

אנו ניגשנו לפתור את הבעיה הזאת על ידי הקצעת זמן בו השולח יחכה לאחר ששלח חבילה, מהפעם שיוכל לשלוח את החבילה הבאה. עם קצת ניסוי וטעיה בחיפוש אחר הזמן המינימלי שיבטיח חוסר מוחלט באיבוד חבילות מצאנו ש0.0005 הוא הזמן האידיאלי.

ברור שזה משפיע על הריצה הכוללת של השולח והמקבל אבל לא משפיע על הסטטיסטיקות שכן זה לא היה בטווח הקוד אותו אנחנו מודדים.

2) משום שהמטרה של QUIC היא למנוע שליחת כל המידע בחבילה אחת, דבר העשוי לעכב קבצים מסוימים כאשר חבילה נאבדת גם אם הן לה קשר לקובץ, החלטנו שבכל שליחת חבילה נבחר 60% מהקבצים הקיימים. כך שבתיאוריה, גם אם תאבד חבילה, הקבצים שלא נשלחו בחבילה הנאבדת יוכלו להמשיך ולהישלח.

3) דגש נוסף הוא אי בירור כיצד לפרש את ממוצע הביטים/חבילות לשנייה. ישנם 2 דרכים שונים להסתכל על חישוב זה:

א) חישוב מספר הביטים לשנייה לכל חבילה, וחישוב הממוצע בין כל החבילות.

ב) חישוב מספר הביטים הכולל, והזמן הכולל שלקח, וחילוק ביניהם.

דוגמה:

חבילה 1: 100 בתים, חושבה ב0.01 שניות

חבילה 2: 150 בתים, חושבה ב0.03 שניות

חבילה 3: 200 בתים, חושבה ב0.06 שניות

ממוצע החבילות = $\frac{\frac{100}{0.1}+\frac{150}{0.3}+\frac{200}{0.6}}{3}$ = 611.11 (דרך א')

ממוצע החבילות = $\frac{100+150+200}{0.1+0.3+0.6}$ = 450 (דרך ב')

בשביל להיות בטוחים, הצגנו סטטיסטיקות לשתי הדרכים.

דרך א':

העברנו את הדפסות הקוד לטבלה בה ניתן לראות את הנתונים של כל זרימה בכל הרצה. הרצה 1 מסמלת את ההרצה בה שלחנו קובץ יחיד והרצה 10 את ההרצה בה שלחנו 10 קבצים ב10 זרימות שונות:

		Run 1	Run 2	...	Run 10
Total	Average num of bytes per second	134065944.1803061	92684082.55670367		19270310.70843328
	Average num of packets per second	72907.57237589273	50403.37801019834		10497.162315154495
Stream 1	total num of bytes	1048589	1048589		1048589
	total num of packets	571	571		3411
	average num of bytes per second	134065944.1803061	93605678.9528709		3293468.3564966125
	average num of packets per second	72907.57237589273	50904.28057964331		10757.560164237088
Stream 2	total num of bytes	X	1048589		1048589
	total num of packets		571		3397
	average num of bytes per second		91762486.16053617		3182121.620446689
	average num of packets per second		49902.47544075348		10387.733818945178
...					
Stream 10	total num of bytes	X	X		1048589
	total num of packets				3427
	average num of bytes per second				3266751.3697198206
	average num of packets per second				10675.71368535687

חשוב לציין שבגלל הגדרתנו לשלוח 60% מהקבצים בכל שליחת חבילה בין הרצות מסוימות יכול להיות שמספר החבילות הנשלחות בין כל זרימה יהיה זהה.

כלומר בהרצות מספר 1 ו2 ניתן לראות אותו מספר חבילות לכל זרימה. זאת בגלל שכאשר ניקח 60% מהזרימות (ונעגל) נראה שנשלח כל פעם חבילה אחרת, ובגלל הגדלים הזהים של הקבצים המספר מתחלק שווה בשווה

ככל, מספר החבילות הנשלחות בכל זרימה הולך וגדל ככל שאנחנו שולחים יותר קבצים, כי ה60% הנשלחים הם מספר קבצים רב יותר, ובכך חילוק המקום בחבילה מתחלק לפריימים בגדלים קטנים יותר.

הגרף הבא מציג את מספר הביטים לשנייה הממוצע בכל הרצה (1-10).

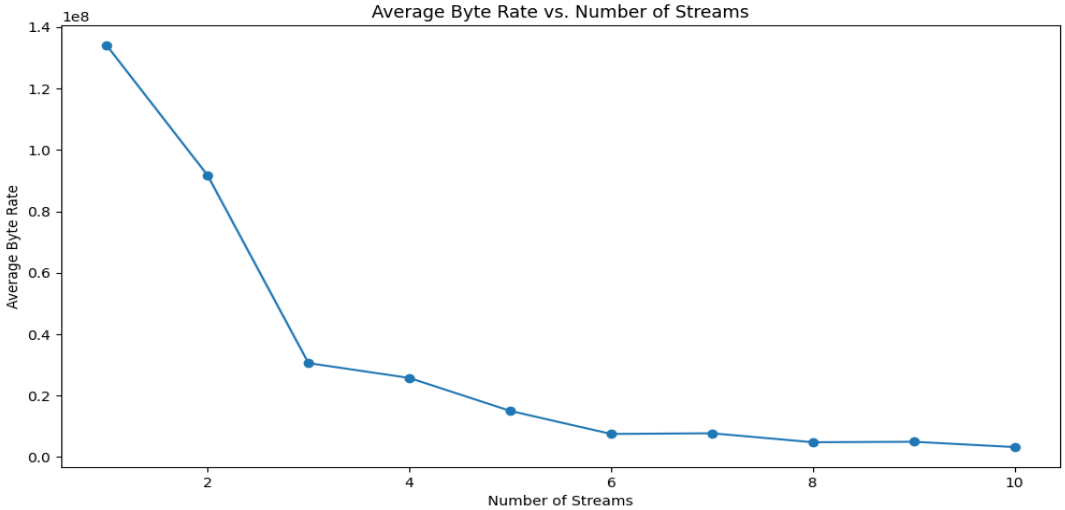
ניתן לראות בבירור שיש השפעה של מספר הקבצים הנשלחים לבין הקצב בוא הביטים נרשמים אצל המקבל. ככל פעם שמגדילים את מס' הזרימות קצב הנתונים הכולל יורד.

הסיבה לכך היא שככל שיש יותר קבצים קיימים יותר פריימים בחבילה. על המקבל למיין כל פריים במקומו כאשר הוא שומר את הקבצים ולהוסיף במקום נפרד את הנתונים הסטטיסטים עליו.

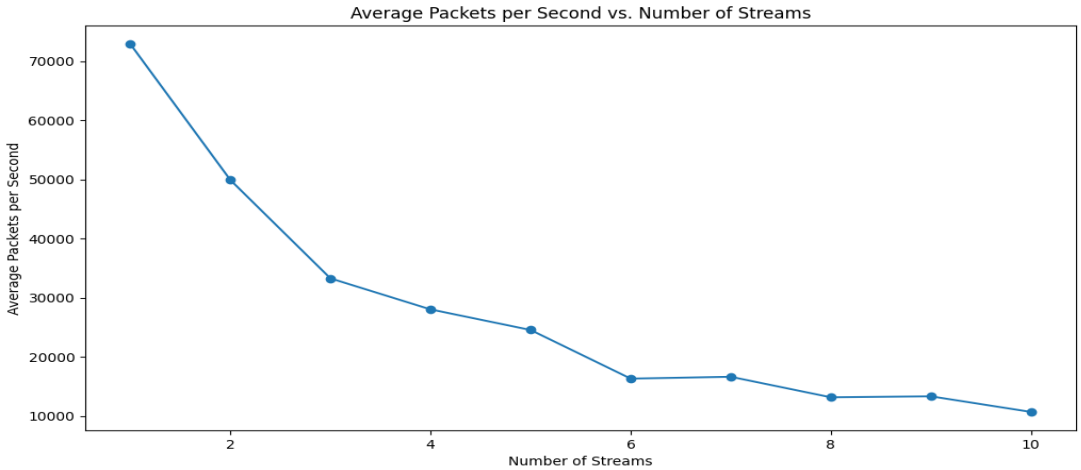
בנוסף הוא מאריך את מערכים אלה במידה ונתקל בקובץ חדש.

כל דברים אלה הם פעולות נוספות שהמקבל במקיים בכל פעם שמקבל חבילה, ונוספות כאשר יש יותר זרימות למיין.

למרות שזה מוסיף זמן חישובי לעומת פרוטוקול כמו TCP, הזמן שנחסך הוא הזמן שקבצים מסוימים מחכים במקרה שנאבדה חבילה, וכעת על כל הקבצים לחכות לשליחה וקבלה מחדש.



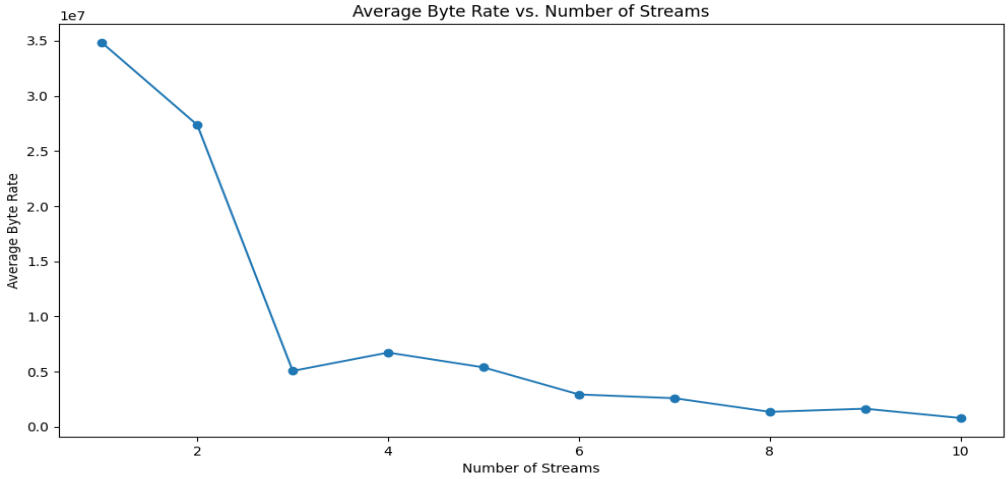
מאותה סיבה בדיוק, נראית ירידה במספר החבילות הנשלחות בשנייה גם כן:

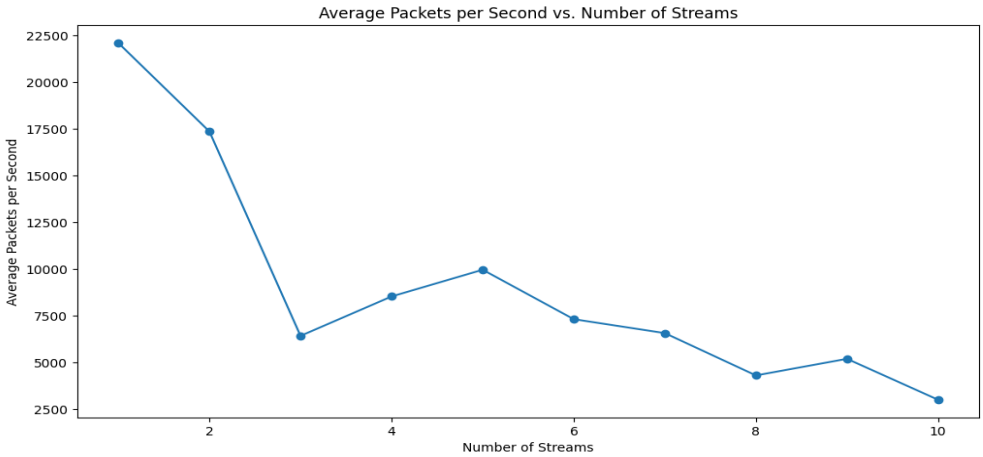


דרך ב':

כל הערה ותובנה מדרך א' תופסות גם כאן, אלו התוצאות:

		Run 1	Run 2	...	Run 10
Total	Average num of bytes per second	34834992.057436	24828018.113064244		4727826.335523504
	Average num of packets per second	22091.84887329062	15745.57051922891		2996.966557175855
Stream 1	total num of bytes	1048589	1048589		1048589
	total num of packets	665	665		3966
	average num of bytes per second	34834992.057436	22717334.5389247		798726.0770087006
	average num of packets per second	22091.84887329062	14407.005479158111		3020.96209422043
Stream 2	total num of bytes	X	1048589		1048589
	total num of packets		665		3926
	average num of bytes per second		27371084.91704115		805550.4743482079
	average num of packets per second		17358.34675915193		3016.044572555181
...					
Stream 10	total num of bytes	X	X		1048589
	total num of packets				3973
	average num of bytes per second				797273.4876928602
	average num of packets per second				3020.790382698782



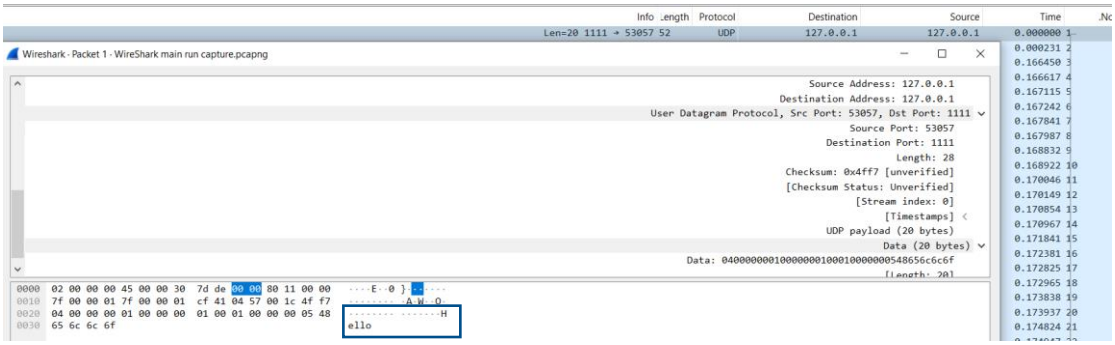


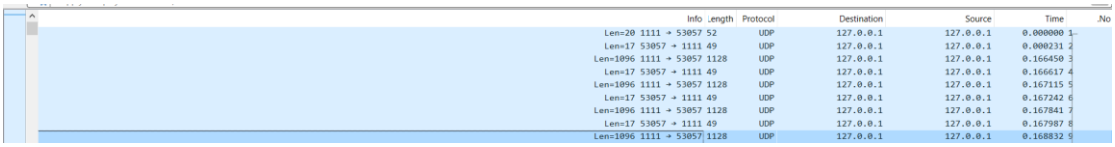
ניתוח ניתור חבילות:

נא לשים לב כי ההקלטות המלאות מצורפות בקובץ.

נביט בכמה צילומי מסך מרכזיים המלמדים אותנו על המימוש.

ראשית, נביט בחבילת פתיחת הקשר ששולח הלקוח לשרת:





בכחול מודגש תוכן החבילה – "Hello" אשר מסמל את חבילת בקשת פתיחת קשר עם השרת.

מרגע קבלת חבילה זו, השולח יתחיל לשלוח חבילות DATA לשרת.

כעת נביט בשליחת החבילות הרגילות:

Info	Length	Protocol	Destination	Source	Time	No.
Len=17 61849 → 1111	49	UDP	127.0.0.1	127.0.0.1	18.527940	24490
Len=1185 1111 → 61849	1217	UDP	127.0.0.1	127.0.0.1	18.528623	24491
Len=17 61849 → 1111	49	UDP	127.0.0.1	127.0.0.1	18.529348	24492
Len=1185 1111 → 61849	1217	UDP	127.0.0.1	127.0.0.1	18.529955	24493
Len=17 61849 → 1111	49	UDP	127.0.0.1	127.0.0.1	18.530594	24494
Len=1185 1111 → 61849	1217	UDP	127.0.0.1	127.0.0.1	18.530645	24495
Len=17 61849 → 1111	49	UDP	127.0.0.1	127.0.0.1	18.531795	24496
Len=1185 1111 → 61849	1217	UDP	127.0.0.1	127.0.0.1	18.531855	24497
Len=17 61849 → 1111	49	UDP	127.0.0.1	127.0.0.1	18.532105	24498
Len=1185 1111 → 61849	1217	UDP	127.0.0.1	127.0.0.1	18.532447	24499
Len=17 61849 → 1111	49	UDP	127.0.0.1	127.0.0.1	18.532734	24500
Len=1185 1111 → 61849	1217	UDP	127.0.0.1	127.0.0.1	18.533098	24501
Len=17 61849 → 1111	49	UDP	127.0.0.1	127.0.0.1	18.533393	24502
Len=1185 1111 → 61849	1217	UDP	127.0.0.1	127.0.0.1	18.534144	24503
Len=17 61849 → 1111	49	UDP	127.0.0.1	127.0.0.1	18.534492	24504
Len=1185 1111 → 61849	1217	UDP	127.0.0.1	127.0.0.1	18.535195	24505
Len=17 61849 → 1111	49	UDP	127.0.0.1	127.0.0.1	18.535597	24506
Len=1185 1111 → 61849	1217	UDP	127.0.0.1	127.0.0.1	18.536175	24507
Len=17 61849 → 1111	49	UDP	127.0.0.1	127.0.0.1	18.536564	24508
Len=1185 1111 → 61849	1217	UDP	127.0.0.1	127.0.0.1	18.537158	24509
Len=17 61849 → 1111	49	UDP	127.0.0.1	127.0.0.1	18.537510	24510
Len=1185 1111 → 61849	1217	UDP	127.0.0.1	127.0.0.1	18.537887	24511

בצילום מסך זה ניתן לראות חבילות אקראיות מתוך קובץ הניתור של ריצת התוכנית. נשים לב כי אכן כל חבילה היא בגודל החבילה שהוגרל אקראית בקוד. בנוסף לאחר הגעת כל חבילה, נשים לב שהשרת שולח חבילה חזרה לשולח (ACK), אך כפי שהסברנו לפני כן, הלקוח מתעלם לחלוטין מה-ACK וממשיך לשלוח חבילות לשרת. אין טעם לבחון את תוכן החבילה מאחר והוא מחרוזת אקראית וחסרת משמעות.

כעת נראה שליחה שונה עם מספר בייטים שונה:

Info	Length	Protocol	Destination	Source	Time	.No
Len=1274 1111 → 49667	1306	UDP	127.0.0.1	127.0.0.1	43.465563	66738
Len=17 49667 → 1111	49	UDP	127.0.0.1	127.0.0.1	43.465845	66739
Len=1274 1111 → 49667	1306	UDP	127.0.0.1	127.0.0.1	43.466404	66740
Len=17 49667 → 1111	49	UDP	127.0.0.1	127.0.0.1	43.466695	66741
Len=1274 1111 → 49667	1306	UDP	127.0.0.1	127.0.0.1	43.467262	66742
Len=17 49667 → 1111	49	UDP	127.0.0.1	127.0.0.1	43.467823	66743
Len=1274 1111 → 49667	1306	UDP	127.0.0.1	127.0.0.1	43.468108	66744
Len=17 49667 → 1111	49	UDP	127.0.0.1	127.0.0.1	43.468695	66745
Len=1274 1111 → 49667	1306	UDP	127.0.0.1	127.0.0.1	43.469187	66746
Len=17 49667 → 1111	49	UDP	127.0.0.1	127.0.0.1	43.469595	66747
Len=1274 1111 → 49667	1306	UDP	127.0.0.1	127.0.0.1	43.470194	66748
Len=17 49667 → 1111	49	UDP	127.0.0.1	127.0.0.1	43.470568	66749
Len=1274 1111 → 49667	1306	UDP	127.0.0.1	127.0.0.1	43.471190	66750
Len=17 49667 → 1111	49	UDP	127.0.0.1	127.0.0.1	43.471461	66751
Len=1274 1111 → 49667	1306	UDP	127.0.0.1	127.0.0.1	43.472200	66752
Len=17 49667 → 1111	49	UDP	127.0.0.1	127.0.0.1	43.472511	66753
Len=1274 1111 → 49667	1306	UDP	127.0.0.1	127.0.0.1	43.473202	66754
Len=17 49667 → 1111	49	UDP	127.0.0.1	127.0.0.1	43.473499	66755
Len=1274 1111 → 49667	1306	UDP	127.0.0.1	127.0.0.1	43.474225	66756
Len=17 49667 → 1111	49	UDP	127.0.0.1	127.0.0.1	43.474519	66757
Len=1274 1111 → 49667	1306	UDP	127.0.0.1	127.0.0.1	43.475184	66758
Len=17 49667 → 1111	49	UDP	127.0.0.1	127.0.0.1	43.475465	66759
Len=1274 1111 → 49667	1306	UDP	127.0.0.1	127.0.0.1	43.476232	66760
Len=17 49667 → 1111	49	UDP	127.0.0.1	127.0.0.1	43.476509	66761

כאן מספר הבייטים שהוגרל היה 1306. אפשר לראות שבכל הרצה מוגרל מספר בייטים שונה.

כעת נביט בחבילות האחרונות:

Info	Length	Protocol	Destination	Source	Time	.No
Len=17 49669 → 1111	49	UDP	127.0.0.1	127.0.0.1	69.816222	110481
Len=1141 1111 → 49669	1173	UDP	127.0.0.1	127.0.0.1	69.816281	110482
Len=17 49669 → 1111	49	UDP	127.0.0.1	127.0.0.1	69.816804	110483
Len=1141 1111 → 49669	1173	UDP	127.0.0.1	127.0.0.1	69.816953	110484
Len=17 49669 → 1111	49	UDP	127.0.0.1	127.0.0.1	69.817491	110485
Len=1141 1111 → 49669	1173	UDP	127.0.0.1	127.0.0.1	69.818223	110486
Len=17 49669 → 1111	49	UDP	127.0.0.1	127.0.0.1	69.818582	110487
Len=1141 1111 → 49669	1173	UDP	127.0.0.1	127.0.0.1	69.819651	110488
Len=17 49669 → 1111	49	UDP	127.0.0.1	127.0.0.1	69.819994	110489
Len=1141 1111 → 49669	1173	UDP	127.0.0.1	127.0.0.1	69.820652	110490
Len=17 49669 → 1111	49	UDP	127.0.0.1	127.0.0.1	69.820968	110491
Len=1141 1111 → 49669	1173	UDP	127.0.0.1	127.0.0.1	69.821729	110492
Len=17 49669 → 1111	49	UDP	127.0.0.1	127.0.0.1	69.822535	110493
Len=1203 1111 → 49669	1235	UDP	127.0.0.1	127.0.0.1	69.822642	110494
Len=17 49669 → 1111	49	UDP	127.0.0.1	127.0.0.1	69.825221	110495
Len=1096 1111 → 49669	1128	UDP	127.0.0.1	127.0.0.1	69.825266	110496
Len=17 49669 → 1111	49	UDP	127.0.0.1	127.0.0.1	69.825460	110497
Len=1096 1111 → 49669	1128	UDP	127.0.0.1	127.0.0.1	69.826052	110498
Len=17 49669 → 1111	49	UDP	127.0.0.1	127.0.0.1	69.826275	110499
Len=1096 1111 → 49669	1128	UDP	127.0.0.1	127.0.0.1	69.826657	110500
Len=17 49669 → 1111	49	UDP	127.0.0.1	127.0.0.1	69.827612	110501
Len=1096 1111 → 49669	1128	UDP	127.0.0.1	127.0.0.1	69.827659	110502
Len=17 49669 → 1111	49	UDP	127.0.0.1	127.0.0.1	69.827847	110503
Len=1096 1111 → 49669	1128	UDP	127.0.0.1	127.0.0.1	69.828270	110504
Len=17 49669 → 1111	49	UDP	127.0.0.1	127.0.0.1	69.828459	110505
Len=1096 1111 → 49669	1128	UDP	127.0.0.1	127.0.0.1	69.828867	110506
Len=17 49669 → 1111	49	UDP	127.0.0.1	127.0.0.1	69.829061	110507
Len=1096 1111 → 49669	1128	UDP	127.0.0.1	127.0.0.1	69.829471	110508
Len=17 49669 → 1111	49	UDP	127.0.0.1	127.0.0.1	69.829656	110509
Len=969 1111 → 49669	1001	UDP	127.0.0.1	127.0.0.1	69.830051	110510
Len=17 49669 → 1111	49	UDP	127.0.0.1	127.0.0.1	69.831131	110511
Len=588 1111 → 49669	620	UDP	127.0.0.1	127.0.0.1	69.831169	110512
Len=17 49669 → 1111	49	UDP	127.0.0.1	127.0.0.1	69.832310	110513
Len=23 1111 → 49669	55	UDP	127.0.0.1	127.0.0.1	69.832342	110514

נשים לב לכמה דברים:

1. בחבילות האחרונות (110510, 110512) ניתן לראות שהשולח ידע להסתדר עם מעט המידע שנשאר לו לשלוח, ולכן שלח חבילות קטנות יותר מגודל החבילה המקורי.
2. הפאקטה האחרונה שנשלחה על ידי השולח היא פאקטת סגירת קשר. ניתן לראות זאת לפי המידע שבחבילה שהוא המחזורת "Good Bye"

```
0000  02 00 00 00 45 00 00 33  2d 87 00 00 80 11 00 00   ····E··3 ********
0010  7f 00 00 01 7f 00 00 01  c2 05 04 57 00 1f b3 00   ******** ***W****
0020  02 00 00 00 01 00 00 00  01 00 01 00 00 00 08 47   ******** *******G
0030  6f 6f 64 20 42 79 65                               ood Bye
```

לאחר חבילה זו נסגר הקשר והופסקה שליחת החבילות.

סיכום הפרוייקט

נתחיל בלציין שהפרויקט היה לנו מאתגר ומלמד מאוד. במהלך ביצועו למדנו רבות על שלל דברים.

ראשית, לפני שבכלל שניגשנו לעבוד, נפגשנו וקראנו את הוראות הפרוייקט ומה דרוש מאיתנו. ראינו כי יש כמה נושאים אשר התבקשנו לממש רק אחד מהם. לפיכך, החלטנו שכל אחד ייקרא קודם כל את המאמר ואז נשב שוב בכדי לדון בתוכן המאמר ולהחליט איזה חלק אנו נממש בפרויקט.

שנית, לאחר שבחרנו איזה חלק אנו ממשים התחלנו בדיונים כיצד נממש, מה בדיוק אנחנו צריכים לבצע וכו'.

מכאן התחלנו לגשת לעבודה. תחילה ישבנו יחדיו על כתיבת הקוד בכדי שנתקבע על בסיס תואם בין כולם. לאחר כמה מפגשים כבר הגענו למצב שבו נוח לנו לפצל את העבודה מבלי לגרור חוסר תיאום בינינו. בפיצול העבודה ניגשנו גם למענה על החלק היבש מה שעזר לחידוד ההבנה שלנו ועזר לנו באופן מימוש הקוד.

העבודה על הפרויקט הייתה מאתגרת וקשה. שעות רבות הושקעו בכתיבתו ואף היו לא מעט ימים של דיונים עד השעות הקטנות של הלילה.

למרות כל הקשיים, נשארנו מאוחדים וממוקדים והתגברנו על המכשולים. האתגר הזה לימד אותנו רבות על החשיבות של התקשורת בעבודת צוות ועל כמה זה משפר את רמת ואיכות העבודה כאשר יש פתיחות לדעות והצעות של אחרים.

לאחר כל העבודה הקשה, קיבלנו את הפרויקט המוגמר ואנו גאים בו מאוד. אנו מאמינים שביצענו פרויקט ברמה גבוהה שמשקף ללא ספק את כמות ההשקעה שלנו.

כעת נעבור למעט מסקנות מהפרויקט:

אנו מאמינים שהמימוש שלנו לריבוי הזרימות שקורה בפרוטוקול QUIC הינו יעיל ביותר מכיוון שהוא בא ליצור מצב בו לא נצטרך לחכות לקובץ שלם לעבור בכדי להתחיל לשלוח את הבא אחריו. עם זאת, מכיוון שאנו התמקדנו במימוש של חלק קטן מתוך QUIC אז התוצאות שלנו כרגע לא מתחרות מספיק עם התוצאות שמקבלים משימוש ב-TCP (או פרוטוקולים בנויים אחרים). לדעתנו, בשילוב של מימוש מלא לפרוטוקול QUIC ושל המימוש שלנו לריבוי זרימות, אנו נקבל תוצאות מעולות שללא ספק ישפרו את צורת השליחה ב-TCP.