פרויקט גמר ברשתות תקשורת - מימוש QUIC עם ריבוי זרימות

מגישים:

ליאור טלמן 323831842, אליהו פרידמן 211691159, מתן מרקוביץ 322318080

חלק יבש

שאלה 1:

1. **חסימת ראש הקו (Head of Line Blocking)** :
ב TCP חבילות נתונים נשלחות ומתקבלות בסדר קפדני כדי להבטיח רצף נכון של המידע. כאשר חבילה אחת הולכת לאיבוד במהלך השידור, כל החבילות שנשלחו אחריה חייבות להמתין לשחזור החבילה האבודה לפני שהן יכולות להימסר לאפליקציה. חסימת ראש הקו גורמת לעיכוב משמעותי בתהליך העברת המידע, מאחר שהאפליקציה לא יכולה לקבל ולעבד את המידע שהגיע עד שהחבילה האבודה מתקבלת מחדש ומועברת בסדר הנכון. בעיה זו מדגישה את התלות הקריטית של TCP בסדר הגעת החבילות ומובילה לאובדן פוטנציאלי של ביצועים כאשר קיימים שיבושים ברשת.

2. **עיכוב בעקבות הגדרת חיבור (Connection Establishment Delay)** :
כל פעם שמתקיים חיבור חדש באמצעות TCP יש צורך בתהליך לחיצת יד משולשת (3-way handshake) שמבטיח את אמינות החיבור. תהליך זה כולל חילופי הודעות בין השולח למקבל לפני תחילת השידור של נתוני האפליקציה. למרות שההליך הזה מבטיח תקשורת אמינה, הוא מוסיף עיכוב של לפחות זמן סבב RTT נוסף בתחילת כל חיבור חדש. עיכוב זה מורגש במיוחד ברשתות עם זמן סבב גבוה או כאשר נדרשים חיבורים קצרים ותכופים, מה שמוביל ליעילות נמוכה ביישומים בזמן אמת.

3. **מגבלות בשל גודל קבוע של כותרת הTCP (Fixed Header Size Limitations) :**
הכותרת של TCP כוללת שדות קבועים בגודלם, מה שמגביל את היכולת להרחיב את הפרוטוקול ולהוסיף פונקציות חדשות. לדוגמה, השדה האופציונלי המוגבל ל-40 בתים בלבד, מגביל את האפשרות להוסיף הרחבות לפרוטוקול מבלי לחרוג מהמגבלות של הכותרת הקיימת. בנוסף, גודל השדה הקטן של מספר הרצף (Sequence Number) וה ACK- עלול להפוך ללא

מספק ברשתות עם מהירות גבוהה, מה שמוביל לבעיות של "עטיפה" של מספרי הרצף וכתוצאה מכך לקשיים בניהול תקשורת אמינה.

4. **תלות בכתובת IP לזיהוי חיבור TCP (Dependency on IP Address for Connection Identification):**
מזהה חיבורים באמצעות שילוב של כתובות ה IP-ומספרי הפורטים של השולח והמקבל. שינוי בכתובת ה IP-של אחד מהצדדים במהלך חיי החיבור (למשל, בעקבות נדידה, שינוי ,NAT או מעבר לרשת אחרת) יגרום לניתוק החיבור הקיים ולזריקת כל הנתונים שהועברו עד לאותו רגע. הבעיה הזו מדגישה את החולשה של TCP בניהול חיבורים במצבים דינמיים, בהם ייתכנו שינויים בכתובת ה IP של המכשירים המחוברים.

5. **ביצועים מוגבלים במהירויות רשת גבוהות (Limited Performance in High-Speed Networks):**
ההגבלות על גודל חלון הבקרה (Congestion Window) והשימוש ב ACK בודד עבור כל חבילה מגבילים את הביצועים של TCP ברשתות בעלות מהירות גבוהה. כאשר מהירות הרשת עולה, חלון הבקרה עלול להתמלא במהירות, מה שמוביל להאטה בהעברת הנתונים ולחוסר ניצול מלא של רוחב הפס הזמין. מצב זה בעייתי במיוחד ברשתות מודרניות עם קצבים גבוהים, שכן TCP אינו מותאם במלואו להתמודדות עם מהירויות אלו, מה שמוביל להורדה משמעותית ביעילות התקשורת.

שאלה 2:

מסירת נתונים אמינה:
פרוטוקול תעבורה חייב להבטיח שכל הנתונים שנשלחים אכן יגיעו ליעדם בצורה נכונה ובסדר הנכון שבו נשלחו. זה כולל שימוש במנגנונים כמו מספרי סדרתיים (sequence numbers) ובאישורי קבלה (ACKs) המאפשרים מעקב אחר כל חבילה שנשלחת ברשת. במקרה שחבילה מסוימת לא מגיעה ליעדה או מגיעה פגומה, הפרוטוקול יזהה זאת ויוזם שליחה מחדש של הנתונים הפגומים או החסרים. כך, הפרוטוקול מבטיח שהמידע מועבר באופן שלם ואמין בין שני הצדדים.

שליטה בעומס ובקרת זרימה:
שליטה בעומס היא תפקיד מרכזי נוסף של פרוטוקול תעבורה, שנועדה למנוע עומס יתר ברשת. פרוטוקול תעבורה משתמש במנגנונים כמו בקרת זרימה מבוססת חלון-זמן (window-based flow control) ובקרת עומסים (congestion control) כדי לווסת את כמות הנתונים שנשלחים ברשת, בהתאם

לקיבולת שלה ולמצב הנוכחי של התקשורת. בקרת הזרימה מבטיחה שהשולח לא ישלח יותר נתונים ממה שהמקבל יכול לעבד, ובקרת עומסים נועדה למנוע הצפת הרשת בחבילות כאשר יש עומס תעבורתי גבוה.

ניהול שגיאות:
במהלך העברת הנתונים, עלולות להתרחש שגיאות שונות – אם זה כתוצאה מהפרעות בתקשורת, תקלות בחומרה או בעיות אחרות. פרוטוקול תעבורה נדרש לזהות שגיאות אלה ולהגיב בהתאם. הוא משתמש במנגנונים כמו בדיקת CRC (Cyclic Redundancy Check) לזיהוי שגיאות בחבילות, ומנגנוני תיקון עצמי, כמו שליחה מחדש של חבילות, כדי להבטיח שהנתונים המועברים יהיו מדויקים ונכונים.

ניהול חיבורים:
פרוטוקול תעבורה אחראי גם על הקמה, ניהול וסיום של חיבורים בין מכשירים ברשת. זה כולל יצירת מזהה חיבור (connection ID) ייחודי שמקשר בין שני רכיבי קצה, ניהול מצב החיבור, וטיפול בשינויים בתנאי הרשת כמו שינוי כתובת הIP . ניהול חיבורים כולל גם שליטה בהעברת הנתונים בין שני הצדדים, תמיכה במצבים מיוחדים כמו הפסקות חיבור זמניות, והבטחת תקשורת יציבה לאורך כל זמן הקשר.

תמיכה בשידור עבור מספר זרמים במקביל:
פרוטוקול תעבורה מודרני, כמו QUIC, מאפשר העברת מספר זרמי נתונים במקביל באותו חיבור. תפקיד זה חשוב במיוחד עבור יישומים כמו דפדפני אינטרנט או סטרימינג, שבהם יש צורך לנהל מספר רב של זרמים במקביל בצורה יעילה וללא הפרעות זה בזה. התמיכה בשידור מקבילי כוללת ניהול של מזהי זרמים (stream IDs) ייחודיים, מנגנוני תזמון חבילות (packet scheduling) ובקרת זרימה נפרדת לכל זרם.

שאלה 3:

בפרוטוקול QUIC- תהליך לחיצת הידיים (Handshake) משלב בתוכו גם את הליך ההצפנה וגם את הליך התעבורה, ובכך מאפשר לקיים את שניהם תוך זמן עיכוב של RTT אחד בלבד. הדבר נעשה באמצעות שילוב של מידע קריפטוגרפי ומידע תעבורתי בפקטה הראשונה שנשלחת בין הלקוח לשרת, כך שכל צד יכול להגדיר את מזהי החיבור וההצפנה הנדרשים להמשך התקשורת.

ב TCP - תהליך לחיצת הידיים מורכב משני חלקים נפרדים – לחיצת יד תעבורתית (הכוללת שלושה שלבים) ולחיצת יד קריפטוגרפית. (במקרה של שימוש בTLS) כך נוצר מצב שבו פתיחת החיבור והצפנתו דורשים לפחות שני RTT נפרדים, דבר

המוביל לעיכוב בזמן החיבור הראשוני. לעומת זאת, ב QUIC כל התהליך הזה מתבצע במהלך RTT אחד בלבד, דבר שמקטין את זמן ההשהיה ומייעל את ביצועי הרשת.

בנוסף לכך QUIC ,משפר את אחד החסרונות המשמעותיים של - TCP אי היכולת להתמודד עם שינויי כתובת IP או פורט במהלך החיבור. בעוד ש TCP-מתבסס על כתובת IP ופורט לקביעת החיבור, שינוי בכתובת או בפורט מחייב פתיחה מחדש של החיבור. ב QUIC לעומת זאת, לכל צד בחיבור יש מזהה חיבור ייחודי שנשמר לאורך כל התקשורת.
כך, גם אם הכתובת או הפורט משתנים במהלך החיבור, ניתן להמשיך את התקשורת מבלי להפסיק את החיבור ולהתחיל מחדש. היכולת הזו מאפשרת לפרוטוקול QUIC לספק חוויית משתמש טובה יותר במצבים של רשתות לא יציבות או במקרים שבהם כתובת ה IP-משתנה (למשל, במעבר בין רשתות Wi-Fi וסלולר).

יתרון נוסף של QUIC על פני TCP הוא התמיכה בשליחת נתונים ב-0 ,RTT תכונה המאפשרת ללקוח לשלוח נתוני אפליקציה מוצפנים כבר בפאקטה הראשונה שהוא שולח. זה מתאפשר על ידי שימוש במפתחות הצפנה שנשמרו מחיבור קודם, דבר שמצמצם עוד יותר את ההשהיה ומאפשר תחילת עבודה מהירה יותר של האפליקציה. TCP לעומת זאת, אינו תומך ביכולת זו, ולכן תמיד ישנו עיכוב נוסף לפני שניתן להתחיל לשלוח נתונים מוצפנים.

תהליך לחיצת הידיים המשולב של ,QUIC הכולל את העברת המפתחות הקריפטוגרפיים והגדרת החיבור במהלך RTT אחד בלבד, יחד עם היכולת לשמור על חיבור פעיל גם במקרים של שינויי כתובת IP ופורט, הופכים אותו לפרוטוקול יעיל יותר במגוון רחב של תרחישי שימוש, במיוחד בתנאי רשת מאתגרים.

שאלה 4:

מבנה החבילה של QUIC : פרוטוקול QUIC מתאפיין במבנה חבילות גמיש ודינמי המותאם לצרכים שונים. החבילות בפרוטוקול זה מחולקות לשני סוגים עיקריים של headers :

1. **Long Header** - משמש בעיקר בעת יצירת החיבור הראשוני או בעת שינויי פרמטרים משמעותיים במהלך חיי החיבור.
   Header זה מכיל שדות קריטיים לתהליך ההתחברות כמו סוג החבילה (Type), גרסת הפרוטוקול , מזהי חיבור ייחודיים ליעד ולמקור מספר הפאקטה ומידע מוצפן.
2. **Short Header -** לאחר יצירת החיבור הראשוני (Handshake) וההתקשרות בין הצדדים, נעשה שימוש ב Header קצר יותר, המכיל רק את השדות

החיוניים ביותר להמשך התקשורת, כמו מזהה החיבור, מספר הפאקטה ומטען מוצפן. Header זה מיועד לפקטות המכילות נתוני אפליקציה ונועד לחסוך במשאבים כמו רוחב פס וזמן עיבוד.

שיפורים לעומת TCP : פרוטוקול QUIC פותח במטרה להתגבר על חלק מהחסרונות הבולטים של פרוטוקול .TCP הנה כמה שיפורים מרכזיים:

1. **גמישות המבנה** :בניגוד לTCP , שבו מבנה החבילה קבוע וכולל שדות בגודל קבוע שלעתים אינו מנוצל במלואו, QUIC מאפשר הרחבה ושינוי של ה Headers בהתאם לצרכים של החיבור הנוכחי. גמישות זו מאפשרת ל QUIC להתאים את עצמו למגוון תרחישים ותנאים, ולהתפתח בקלות רבה יותר.

2. **שיפור האבטחה** :כל החבילות ב QUIC מוצפנות, כולל ה Headers עצמם, דבר שמגביר את אבטחת המידע ומונע גישה או שינוי מצד גורמים בלתי מורשים. הדבר מאפשר ל QUIC לספק חוויית גלישה בטוחה ומוגנת יותר בהשוואה ל TCP שבו ה Headers אינם מוצפנים.

3. **התמודדות עם אובדן פאקטות** :ב TCP אובדן של חבילה אחת משפיע על מעבר כל הפקטות הבאות בתור, שכן הן תלויות בה. כלומר, לא ניתן לעבד פקטות עוקבות עד שהחבילה האבודה משוחזרת. ב QUIC לעומת זאת, ניתן לקיים מספר זרמים באותו חיבור, וכל זרם מתקדם באופן עצמאי. כתוצאה מכך, אובדן חבילה בזרם אחד אינו משפיע על זרמים אחרים, דבר שמשפר את הביצועים והמהירות הכוללת של החיבור.

4. **שיפור ניהול ה ACK** : בעוד שב TCP קיים שדה אופציונלי המאפשר תמיכה ב-3 ACKs בלבד, ב QUIC קיימת תמיכה בעד 256 סוגי ACKs שונים. עובדה זו מאפשרת ל QUIC לספק הערכה מדויקת יותר על מצב החבילות ברשת, ומסייעת בזיהוי מהיר של חבילות שאבדו או עוכבו.

5. **מניעת בעיית ה Head-of-Line Blocking**: אחת הבעיות המשמעותיות ב TCP היא תופעת ה Head-of-Line Blocking שבה אובדן חבילה מעכב את עיבוד כל הפקטות הבאות. ב QUIC כאמור, תופעה זו מופחתת משמעותית הודות לאפשרות לנהל מספר זרמים בלתי תלויים בתוך אותו חיבור, מה שמאפשר למידע להגיע ליעדו בזמן ולמנוע עיכובים מיותרים.

שאלה 5:

בQUIC כל זרם נתונים מנוהל בנפרד, כך שאובדן של חבילה אחת אינו משפיע על זרמים אחרים באותו חיבור. זהו שיפור משמעותי בהשוואה ל TCP שבו אובדן של

חבילה יחידה יכול לחסום את העיבוד של חבילות עוקבות, מה שגורם להשהיות נוספות ולאיבוד ביצועים.

כשחבילות מגיעות באיחור או לא מגיעות כלל QUIC ,משתמש במנגנון המאפשר לזהות חבילות שאבדו ולהגיב בהתאם. הזיהוי מתבצע באמצעות מנגנון של ACK כך ש QUIC- שולח על כל חבילה שהתקבלה בהצלחה. אם חבילה מאוחרת יותר קיבלה ACK אך לא התקבל ACK על החבילה שקדמה לה, נחשב כי החבילה הקדומה אבדה. כמו כן, הפרוטוקול מתבסס על שני סוגי Thresholds כדי לקבוע אם חבילה אבדה ויש לשדר אותה מחדש.

1. **סף מבוסס מספר סידורי (Sequence Number) :**
   ה QUIC ממספר את החבילות שנשלחות, כך שכל חבילה מקבלת מספר סידורי ייחודי. כאשר חבילה מקבלת ACK ומספרה הסידורי גבוה מהחבילות שטרם קיבלו ,ACK נחשב כי החבילות שביניהן אבדו ויש לשדר אותן מחדש.
2. **סף מבוסס זמן (Time-Based Threshold):**
   לכל חבילה יש זמן משוער להגעה המבוסס על חישובי RTT קודמים. אם חלף הזמן המוערך ולא התקבל ACK עבור חבילה מסוימת, נחשב כי החבילה אבדה ויש לשדר אותה מחדש.

בנוסף, כאשר מתגלה אובדן של חבילות, QUIC שולח מחדש את המידע האבוד במסגרת חבילות חדשות, כאשר כל חבילה חדשה מקבלת מספר סידורי חדש ללא קשר למספר הסידורי של החבילה האבודה. בכך, הפרוטוקול מוודא שהמידע יגיע בסופו של דבר ליעדו, תוך כדי שמירה על סדר נתונים תקין ומניעת שליחות מיותרות.

במקרים של הגעת חבילה באיחור, QUIC ממשיך להשתמש בחבילה כל עוד היא רלוונטית ומתאימה לסדר הנתונים. ה ACK שנשלח לאחר מכן יאשר גם חבילות שנשלחו לאחר החבילה המאוחרת אך הגיעו לפניה. מנגנון זה מאפשר למנוע שידורים חוזרים מיותרים ולהפחית את העומס על הרשת.

השילוב של ניהול זרמים עצמאי, מנגנוני זיהוי ואיחוי חבילות גמישים, והתאמות דינמיות לקצב השידור מאפשר ל QUIC להתמודד בצורה יעילה עם בעיות של אובדן חבילות והגעתן באיחור. כל זאת תוך שיפור ביצועים והפחתת ההשפעה של בעיות רשת על החיבור הכולל.

שאלה 6:

בקרת העומס בפרוטוקול QUIC היא מרכיב מרכזי שמאפשר לפרוטוקול להתמודד עם התנאים המשתנים של הרשת בצורה אפקטיבית, תוך שמירה על ביצועים גבוהים ויציבות בקצב השידור. כמו בפרוטוקול TCP , גם ב QUIC קיימת מערכת של

חלונות שליחה (Congestion Window) , שמגבילה את כמות הנתונים שהשולח יכול לשלוח לפני שהוא מקבל אישור מהמקבל. עם זאת, QUIC מבצע הפרדה בין בקרת העומס לבקרת האמינות, בכך שהוא משתמש במספרי חבילות לבקרת עומס וב offset- למסגרות זרם לצורך בקרת אמינות.

בקרת העומס ב QUIC פועלת בשיטת חלון, שמשמעותה היא הגבלה על כמות הנתונים שהשולח יכול לשלוח בזמן נתון. זה נעשה כדי למנוע הצפה של הרשת במידע רב מדי בו זמנית, מה שעלול לגרום לאובדן חבילות ולירידה בביצועי הרשת.

הפרוטוקול עושה שימוש במדידות זמן תגובה לסיבוב (RTT) כדי להעריך את המצב הנוכחי של הרשת ולהתאים את גודל החלון בהתאם. מדידת ה RTT מתבצעת באופן קבוע על ידי השוואת הזמן שעבר מאז שנשלחה חבילה מסוימת ועד שהתקבל עליה אישור. נתונים אלו מאפשרים ל QUIC לחשב את הזמן המרבי שהחבילה יכולה לשהות ברשת לפני שתיחשב כ"אבודה". במקרים בהם מתגלה עומס יתר עקבי ברשת, לדוגמה, כאשר שתי חבילות או יותר אבדו והזמן שעבר בין שליחתן עולה על זמן מוגדר שנקבע על סמך ה RTT והסטייה בו ,ה QUIC מקטין את גודל חלון השליחה כדי למנוע החמרה במצב.

כאשר מתגלה עומס יתר זמני או אובדן חבילות, QUIC מסוגל להגיב בצורה מהירה ויעילה על ידי האטת קצב השידור, מה שמאפשר לשולח להימנע מהגברת העומס על הרשת. בנוסף, כאשר מזוהה איבוד חבילות, ה QUIC משלב מחדש את החבילות האבודות בחבילות חדשות עם מספרי חבילות חדשים, ומבצע את השידור מחדש שלהן. הגישה הזו מאפשרת שמירה על רציפות ואמינות השידור, בלי לגרום לצמצום מיותר של גודל חלון השידור.

<u>החלק הרטוב:</u>

<u>מימוש צד הלקוח:</u>

הלקוח מממש את הצד השולח של פרוטוקול. המטרה המרכזית של הלקוח היא ליצור מספר זרמי נתונים (streams) שבהם ישלחו קבצים מרובים במקביל, ולוודא שכל זרם נתונים מחולק לקטעים שווים (chunks) הנשלחים במקטעים קטנים יותר (frames)בתוך כל חבילה.

תיאור הפונקציות המרכזיות:

generate_random_files:

הפונקציה יוצרת קבצים אקראיים בגודל קבוע של 2 MB כל אחד, שישמשו כזרמי נתונים (streams) שישלחו לשרת. כל קובץ נשמר כקובץ טקסט זמני בזיכרון לצורך שליחתו לשרת.

מימוש פונקציה זו מאפשר לבחון את התמודדות הלקוח והשרת עם קבצים גדולים ושונים, ומספק תשתית לבדיקה של העברת נתונים במספר זרימות.

create_packet:

הפונקציה אחראית על יצירת פקטות QUIC שבהן מוקצים חלקים מנתוני כל זרימה למסגרות (frames) על בסיס קבוע של גודל חבילה (chunk size). היא מוודאת שכל חבילה תכיל נתונים מזרמים מרובים, כלומר מקבצים שונים, בצורה מסודרת.

הפונקציה מקבלת מספר סידורי של החבילה (stream_id), את הנתונים עצמם (data), ואת נקודת ההתחלה (offset) של כל זרימה, כך שתדע מאיפה להתחיל לשלוח את החבילה הבאה מבלי שתהינה דריסות של נתונים וכדי לשמור על רצף הקובץ למרות שליחתו במספר מקטעים.

כל פקטה שנוצרת מכילה אחוז מסוים ממספר הזרימות (במקרה זה 60%). כלומר, במקום לשלוח נתונים מכל הזרימות בו-זמנית, שולחים נתונים רק מאחוז מסוים מהן, כדי למנוע עומס על הרשת.

הפונקציה בוחרת את גודל המקטעים (frames) הנשלחים עבור כל זרימה, כך שיתאימו לגודל ה chunk-המוקצה לחבילה. הקצאת גודל זה נעשית לפי מספר הזרימות הפעילות.

לבסוף, הפונקציה בודקת אם כל הנתונים נשלחו, ואם כן – היא מסירה את הזרימה מהרשימה.

send_packet:

הפונקציה שולחת את חבילת ה QUIC-לשרת ואחראית על שליחת הפקטות בצורה מסודרת תוך שמירה על המשכיות בין החבילות שנשלחות לשרת.

הפונקציה מבצעת סריאליזציה (המרת הפקטה למערך בייטים) ושולחת אותה לשרת באמצעות פרוטוקול UDP.

send_syn, receive_ack, send_fin_message:

פונקציות אלו מטפלות בניהול ההתחברות והסיום של התקשורת בין הלקוח לשרת.

send_syn שולחת חבילת SYN לשרת על מנת להקים את החיבור הראשוני (handshake). זוהי חבילת השליחה הראשונה, שמשמשת להודיע לשרת על כוונת הלקוח לשלוח חבילות נתונים.

receive_ack מקבלת חבילת ACK מהשרת, המאשרת את קבלת חבילת ה- SYN ומאשרת שהשרת מוכן לקבל נתונים.

send_fin_message בסיום העברת הנתונים, הפונקציה שולחת חבילת FIN לשרת לסגירת החיבור.

מימוש השרת:

השרת מממש את הצד המקבל של פרוטוקול QUIC, ומטרתו היא לקבל את החבילות שנשלחות מהלקוח, לעבד אותן, ולשמור את הנתונים שקיבל בצורה מסודרת. בנוסף, השרת שומר סטטיסטיקות של קצב הנתונים שהתקבלו וקצב החבילות שהתקבלו, לצורך ניתוח ביצועים. כדי לתפוס שגיאות ולהקל על בדיקות האמינות השרת בנוסף משווה את הקבצים שהתקבלו לעומת הקבצים שנשלחו.

תיאור הפונקציות המרכזיות:

:handle_packet

הפונקציה המרכזית האחראית על ניהול קבלת חבילות מהלקוח. השרת מאזין לחבילות שמתקבלות (בין אם הן חבילות SYN, DATA, או FIN) ומטפל בהן בהתאם לסוג החבילה.

כאשר מתקבלת חבילת SYN, השרת שולח חבילת SYN-ACK ללקוח לאישור קבלת הבקשה להתחברות.

עבור חבילת DATA, הפונקציה תשלח את הנתונים לפונקציה **process_data_packet.**

כאשר מתקבלת חבילת FIN, השרת מסיים את ההתקשרות עם הלקוח וסוגר את החיבור.

:process_data_packet

הפונקציה מעבדת חבילות נתונים שמתקבלות מהלקוח ומחשבת את הסטטיסטיקות הנלוות לה, כמו ספירת הביטים, החבילות והזמן הדרוש לעיבוד כל זרימה.

כאשר מתקבלת חבילת נתונים, הפונקציה מעדכנת את הסטטיסטיקות של כל זרימה בנפרד: כמה נתונים נשלחו, כמה חבילות הגיעו, ומה היה הזמן הנדרש לעיבוד הנתונים.

כמו כן, היא מוודאת שהנתונים נכנסים לזרימה הנכונה, ומעדכנת את המבנים בהתאם (מקטעים, offsets, וכו').

קוד הבדיקות:(TestClientServer)

run_server מפעיל את השרת בת'רד נפרד כדי להאזין לחבילות.

run_client מפעיל את הלקוח בת'רד נפרד ושולח את הנתונים.

run_both_for_testing מפעיל את השרת והלקוח במקביל ובודק אם הנתונים שהתקבלו זהים לאלו שנשלחו.

הטסטים- יש בדיקות שונות לבדוק מצבי קצה כמו קבצים קצרים, בינוניים, ארוכים, נתונים ריקים או לא חוקיים.

<u>סדר כרונולוגי של פעולות המחלקות והפונקציות:</u>

סדר הפעולות בקוד ה client:

1. <u>הפעלת הפונקציה start(num_flows) בלקוח:</u>
   - המטרה היא ליזום חיבור עם השרת ולשלוח מספר זרמים (קבצים) במקביל.
   - נוצר אובייקט של Client עם כתובת השרת והפורט (localhost, 12346).
2. <u>קריאה ל:send_syn()-</u>
   - הלקוח שולח לשרת חבילת SYN כדי להתחיל את תהליך החיבור. זהו שלב ה.Handshake-
   - נשלחת חבילת QUIC עם הדגל SYN (הדגל הראשון) שמאותת לשרת שהלקוח מעוניין לפתוח חיבור.
3. <u>קריאה ל:receive_ack()-</u>
   - הלקוח ממתין לחבילת ACK מהשרת. אם ה ACK-מתקבל בהצלחה, החיבור נפתח.
   - השרת שולח חבילת SYN-ACK כתגובה.
4. <u>קריאה ל:generate_random_files(num_flows)-</u>
   - הלקוח יוצר מספר קבצים רנדומליים (מספר הזרמים שנבחר). כל קובץ בגודל 2MB.
   - הקבצים מאוחסנים בזיכרון לשם שליחה.
5. <u>קריאה ל:send_all_packets(data)-</u>
   - הלקוח מתחיל לשלוח את הקבצים (הנתונים) על פני מספר זרמים (streams).
   - כל חבילה שמיוצרת מכילה חלקים שונים ממספר זרמים ומסומנת על ידי הלקוח בפריימים המתאימים.
   - יש שליחת ACK בין חבילת נתונים ללקוח.
6. <u>קריאה ל:send_fin_massage()-</u>

- לאחר שכל הנתונים נשלחו, הלקוח שולח חבילת FIN לסגירת החיבור.

7. קריאה ל:()close-
   - הלקוח סוגר את הסוקט.

סדר הפעולות בקוד ה server:

1. הפעלת :() Server. init
   - נוצר אובייקט של Serverעם כתובת IP ופורט.
   - סוקט UDP נפתח ומוכן לקבל נתונים.
2. הפעלת :()handle_packet
   - השרת נכנס ללולאה שמאזינה לחבילות שמגיעות מהלקוח.
   - חבילות מתקבלות ומעובדות אחת אחת.
3. קבלת חבילת SYN ב:()start-
   - השרת מקבל חבילת SYN מהלקוח, שמצביעה על בקשה לפתיחת חיבור.
   - לאחר קבלת ה ,SYN-השרת מגיב עם חבילת SYN-ACK כדי לאשר את החיבור.

4. קבלת חבילות נתונים ב:()process_data_packet-
   - השרת מקבל חבילות נתונים מהלקוח.
   - כל חבילה מכילה פריימים ממספר זרמים, כאשר כל פריים מכיל חלק מנתוני הזרם.
   - השרת שומר את הנתונים מהזרמים המתאימים, מעדכן את מספר הבייטים והחבילות שהתקבלו, וכן אוגר סטטיסטיקות על מהירות התעבורה.
5. שליחת ACK בחזרה ללקוח:
   - לאחר עיבוד כל חבילה, השרת שולח חבילת ACK ללקוח כדי לאשר את קבלת החבילה.
6. קבלת חבילת FIN ב:()start-

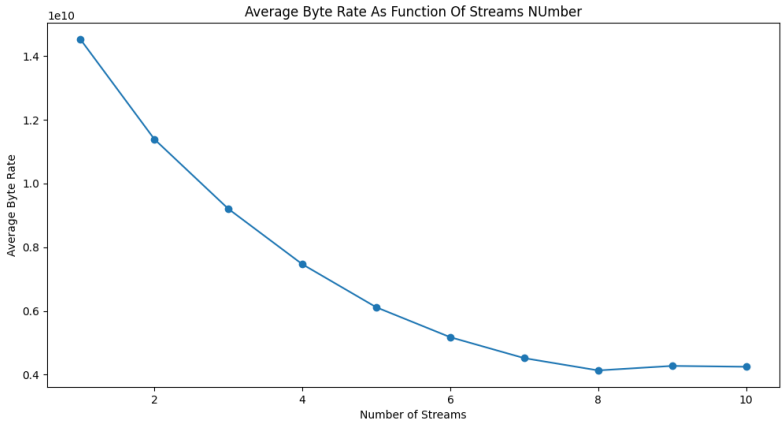
- כאשר השרת מקבל חבילת FIN, הוא מבין שהתהליך הסתיים והחיבור נסגר.

7. <u>קריאה ל:()print_statistics-</u>
   - לאחר סיום תהליך העברת הנתונים, השרת מדפיס את הסטטיסטיקות:
     - מספר הבייטים שהתקבלו לכל זרם.
     - מספר החבילות לכל זרם.
     - מהירות הממוצעת לכל זרם (בבייטים לשנייה ובחבילות לשנייה).
   - השרת משווה בין הקבצים שהתקבלו לקבצים המקוריים בעזרת compare_files().
8. <u>סגירת הסוקט:</u>
   - בסיום כל תהליך ההעברה, השרת סוגר את הסוקט.

<u>תהליך כללי:</u>

1. הלקוח שולח חבילת SYN לפתיחת חיבור.
2. השרת מקבל את ה SYN-ושולח SYN-ACK לאישור.
3. הלקוח מקבל את ה ACK-ומתחיל לשלוח את הנתונים מחולקים לחבילות עם פריימים ממספר זרמים במקביל.
4. השרת מקבל את החבילות, מעבד את הפריימים, ושולח ACK חזרה ללקוח על כל חבילה.
5. לאחר סיום שליחת כל הנתונים, הלקוח שולח חבילת FIN לסגירת החיבור.
6. השרת מקבל את ה FIN-ומסיים את תהליך החיבור, תוך שהוא מציג את הסטטיסטיקות של ההעברה.

ניתוח סטטיטיקות:

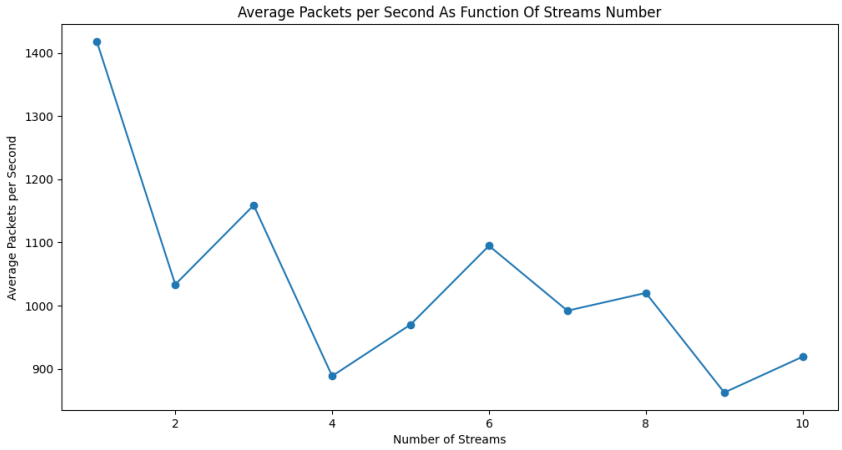


מצורף גרף הסטטיסטיקה שחושב עבור מספר הבייטים שעוברים בשניה כתוצאה של הגדלת מספר הזרימות.

ניתן לראות שככל שכמות הזרימות עולה, קצב העברת הנתונים (byte/sec) יורד.

הסיבה לכך היא תוצאה של מספר גורמים – עומס על המערכת, כלומר המערכת נאלצת לנהל יותר חבילות בו זמנית ועיבוד והקצאת זיכרון לכל זרימה בנפרד תיצור עומס.

בנוסף לכך, כאשר יש יותר זרימות שמתחרות על אותו רוחב פס, כל זרימה מקבלת רק חלק ממנו. אם נניח שכל זרימה משתמשת ברוחב פס מסוים, אז כמות הנתונים שכל זרימה יכולה להעביר יורדת ככל שמספר הזרימות גדל, פשוט כי יש פחות רוחב פס פנוי לכל זרימה בנפרד.



מאותה סיבה נצפה שגם מספר הפקטות לשניה ירד אך מכיוון שבמימוש הקוד שלנו גודל הזרימה משתנה מהרצה להרצה (כלומר בהרצת הmain לכל i זרימות יהיה גודל זרימה שונה) ולכן הגרפים לא יורדים בצורה אחידה.

ניתוח פקטות בעזרת WireShark:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	UDP	62	60216 → 12346 Len=30

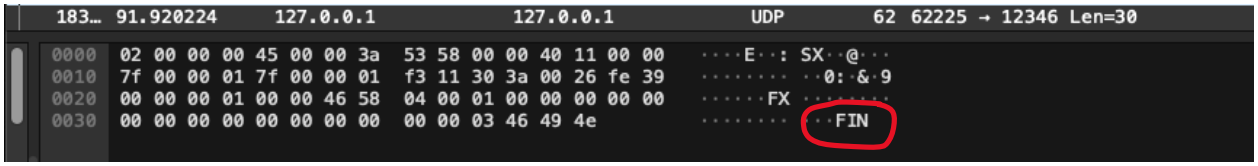
```
0000  02 00 00 00 45 00 00 3a  cc e8 00 00 40 11 00 00   ····E··: ····@···
0010  7f 00 00 01 7f 00 00 01  eb 38 30 3a 00 26 fe 39   ........ ·80:·&·9
0020  00 00 00 01 00 00 00 01  01 00 01 00 00 00 00 00   ........ ........
0030  00 00 00 00 00 00 00 00  00 00 03 53 59 4e         ........ ···SYN
```

No.	Time	Source	Destination	Protocol	Length	Info
2	0.000151	127.0.0.1	127.0.0.1	UDP	66	12346 → 60216 Len=34

```
0000  02 00 00 00 45 00 00 3e  7d 27 00 00 40 11 00 00   ····E··> }'··@···
0010  7f 00 00 01 7f 00 00 01  30 3a eb 38 00 2a fe 3d   ........ 0:·8·*·=
0020  00 00 00 01 00 00 00 01  03 00 01 00 00 00 00 00   ........ ........
0030  00 00 00 00 00 00 00 00  00 00 07 53 59 4e 5f 41   ........ ···SYN_A
0040  43 4b                                              CK
```

No.	Time	Source	Destination	Protocol	Length	Info
3	0.111269	127.0.0.1	127.0.0.1	UDP	1434	60216 → 12346 Len=1402

```
0000  02 00 00 00 45 00 05 96  6f f5 00 00 40 11 00 00   ····E··· o···@···
0010  7f 00 00 01 7f 00 00 01  eb 38 30 3a 05 82 03 96   ........ ·80:····
0020  00 00 00 01 00 00 00 01  02 00 00 00 00 00 00 00   ........ ........
0030  00 00 00 00 00 00 00 00  00 05 5f 6b 71 45 65 7a   ........ ··_kqEez
0040  6d 51 51 30 44 48 4a 63  33 68 69 45 4c 49 55 41   mQQ0DHJc 3hiELIUA
0050  56 45 6f 6d 6c 77 37 6b  37 42 4b 47 44 37 36 36   VEomlw7k 7BKGD766
0060  54 73 4f 4b 77 38 42 6d  72 68 50 72 4c 71 77 30   TsOKw8Bm rhPrLqw0
0070  77 64 56 61 33 73 34 4e  41 79 4c 4d 61 6c 67 4c   wdVa3s4N AyLMalgL
0080  65 42 61 4c 34 77 34 4c  6c 79 4f 76 44 6c 51 61   eBaL4w4L lyOvDlQa
0090  34 6e 42 73 62 6c 34 53  4f 4c 38 41 31 36 73 34   4nBsbl4S OL8A16s4
00a0  77 78 6a 4d 52 79 6a 4f  68 55 71 41 73 64 39 66   wxjMRyjO hUqAsd9f
00b0  79 79 6f 75 6a 67 69 5a  6f 68 39 49 4e 74 4a 74   yyoujgiZ oh9INtJt
00c0  66 6a 67 4c 4e 51 72 36  44 4b 75 68 54 57 6f 62   fjgLNQr6 DKuhTWob
00d0  41 57 72 48 76 56 6c 71  55 6c 41 5a 44 6b 6b 4d   AWrHvVlq UlAZDkkM
00e0  4e 4d 66 30 35 64 77 4e  55 6f 4e 7a 52 43 49 49   NMf05dwN UoNzRCII
00f0  77 44 72 4e 39 55 4d 6c  35 46 68 30 50 4c 52 43   wDrN9UMl 5Fh0PLRC
0100  6f 4e 41 64 41 65 46 30  6d 79 6d 47 39 76 48 56   oNAdAeF0 mymG9vHV
0110  68 74 64 53 36 54 59 7a  6c 39 6f 67 5a 34 66 39   htdS6TYz l9ogZ4f9
0120  31 56 70 6a 6a 50 5a 32  65 55 33 47 43 44 4e 46   1VpjjPZ2 eU3GCDNF
```



מתוך קובץ הקלטת הwireshark המצורפת אספנו ארבע דוגמאות לסוגי פקטות:

1. הפקטה הראשונה (SYN) נשלחה מהלקוח לשרת בבקשה לפתיחת קשר (12346 <- 60216)
2. הפקטה השניה(SYN-ACK) נשלחה מהשרת חזרה ללקוח ובכך מסיימת את לחיצת היד (60216 <- 12346)
3. הפקטה השלישית (DATA) נשלחת מהלקוח לשרת ובעצם מהווה את רצף העברת הנתונים הראשון מתוך קבצי הקלט של הלקוח.
4. הפקטה אחת לפני האחרונה (FIN) נשלחת מהלקוח לשרת ובכך מודיעה שרצף העברת הנתונים הסתיים ואפשר לסגור את הקשר.