פרויקט גמר
רשתות תקשורת

מימוש פרוטוקול QUIC בהיבט מדידת RTT ובקרת עומס

מגישים:

יאיר כהן 206386708

עמית גיני 207275215

צופיה טויטו 324953900

מנשה אשכנזי 326648532

חלק "יבש"

<u>שאלה 1:</u> תארו במילים שלכם 5 חסרונות/ מגבלות של TCP.

1) החיבור בין בקרת העומס ברשת לבין בקרת האמינות

גודל חלון ה-TCP (או חלון הגודש) מייצג את כמות הנתונים שניתן לשלוח מבלי לקבל אישור מהמקבל. הטיפול בבקרת העומס ברשת ובקרת האמינות מתבצעת ע"י הקטנת החלון.

בקרת עומס ברשת מתייחסת למנגנונים המשמשים לניהול כמות הנתונים המועברים ברשת כדי למנוע עומס ולהבטיח שימוש יעיל במשאבים. בעוד בקרת האמינות מבטיחה מסירת נתונים אמינה.

כאשר בקרת העומס מתרחשת, הקטנת גודל החלון מבטיחה שהשולח לא ישלח יותר נתונים ממה שהרשת יכולה להתמודד. גודל חלון קטן יותר יכול לשפר את המהימנות בתנאים מסוימים על ידי הבטחת אישור נתונים בתדירות גבוהה יותר, ובכך לאפשר זיהוי ושידור חוזר מהיר יותר של מנות אבודות.

אם אישור (ACK) עבור חבילה שנשלחה לא מתקבל בתוך תקופה מסוימת (RTO), TCP מניח שהמנה אבדה. **ההנחה העיקרית של הפרוטוקול לגבי הסיבה לאובדן מנות היא גודש ברשת.** הנחה זו היא עיקרון בסיסי העומד בבסיס מנגנוני בקרת הגודש של TCP.

בכך, קורה מצב שTCP מפרש את אובדן החבילות שאינם קשורים לגודש (אלא בעקבות גורמים אחרים כגון: שגיאות שידור, טעות במסלול, חבילה לא תקינה) כסימנים של גודש ומקטין שלא לצורך את חלון הגודש, מה שמוביל לביצועים לא אופטימליים מתוך הנחה לא נכונה מכיוון שמספר החבילות שלא קיבלו ACK לא מייצג בצורה מדוייקת את כמות החבילות ברשת.

על ידי הפחתת קצב השידור בתגובה לאובדן מנות מסיבות אחרות שהן לא עומס (כמו למשל כשלי קישור, שגיאות ברשת), TCP לא ינצל את רוחב הפס הזמין של הרשת בצורה טובה. זה בעייתי ברשתות עם רוחב פס גבוה וזמן אחזור גבוה, בהן העלות של הפחתת חלון הגודש היא משמעותית.

2) חסימת ראש קו

חסימת HOL (Head-of-Line) ב-TCP (Transmission Control Protocol) מתייחסת למקרה בו פקטות מתעכבות או נחסמות מכיוון שהן תקועות מאחורי מנה שאבדה או מתעכבת בשידור- כלומר שומר על הגעה לפי סדר השליחה – ועיכוב במקרה שלא הגיע. במקרה כזה מתעכבת הגעתן של פקטות דבר הפוגע ישירות בתפוקה.

3) עיכוב עקב הגדרת חיבור

תהליך לחיצת היד של TCP כולל שלושה שלבים: SYN, SYN-ACK ו-ACK.
כלומר, לפני שניתן לבצע העברת נתונים ממשית, יש עיכוב של לפחות זמן נסיעה אחד הלוך ושוב

עד להשלמת לחיצת היד 1.5 RTT . עיכוב זה יכול להשפיע על יישומים או שירותים בזמן אמת הדורשים העברת נתונים מיידית. עיכוב זה נוצר עוד לפני העברת המידע.

4) מגבלות של כותרת פרוטוקול קבועה (Header)

לכותרת ה-TCP יש שדות בגודל קבוע כמו שדות מספר הרצף, שדות ACK (בגודל 4 בתים) וגודל חלון בקרת הזרימה (בגודל 2 בתים) קבועים

בשל מגבלות גודל הכותרת, ייתכן ש-TCP יצטרך לדחוס מידע ACK כך שיתאים לשטח הפנוי בכותרת. תהליך דחיסה זה יכול להשפיע על הפירוט והדיוק של משוב ACK, ועלול להפחית את היעילות של מנגנוני בקרת גודש ואסטרטגיות שידור חוזר.

בנוסף, ככל שמהירויות הרשת עולות והרשת מתפתחת, השדות בגדלים הקבועים הנ"ל הופכים למגבילים יותר. מספרי רצף ושדות ACK מגיעים לשיאם ומתאפסים מהר, וגודל חלון בקרת הזרימה מגביל ישירות את התפוקה .

5) תלות בכתובות IP קבועות (זיהוי קשר ייחודי)

בהקמת הקשר בפרוטוקול TCP, לחיצת ידיים משולשת-

SYN : הלקוח שולח חבילת SYN (סנכרון) לשרת, המציין את הרצון ליצור חיבור. חבילה זו כוללת את מספר הרצף הראשוני של הלקוח ואת כתובת ה-IP והיציאה של הלקוח.

SYN-ACK : השרת מגיב עם חבילת SYN-ACK (סנכרון-אישור). חבילה זו כוללת את מספר הרצף הראשוני של השרת ומאשרת את חבילת ה-SYN של הלקוח. זה כולל גם את כתובת ה-IP והיציאה של השרת.

ACK : הלקוח שולח חבילת ACK (אישור) חזרה לשרת, ומאשר את ה-SYN-ACK של השרת.

בשלב זה, החיבור נוצר, והן הלקוח והן השרת יכולים להתחיל לשלוח נתונים.

ניתן לראות כמתואר לעיל כי הקמת הקשר מתבססת על כתובות IP ומספרי הפורט של שני הצדדים.

כתובות IP קבועות הן סטטיות ומוקצות למכשירים או מארחים ספציפיים.

ייתכן שבמהלך ההתקשרות כאשר צד אחד נמצא בסביבה דינמית שבה מכשירו ישנה את מיקומו או את תצורת הרשת שלו ואז התקשורת תאבד לגמרי, ויצטרך להקים קשר חדש ולהתחיל מהתחלה את העברת המידע.

<u>שאלה 2:</u> ציינו 5 תפקידים שפרוטוקול תעבורה צריך למלא.

1) אמצעי זיהוי

פרוטוקול תעבורה צריך להבחין בין זרמי תקשורת שונים ולנתב כל זרם ליישום המתאים, וע"מ ליצור ולשמר את החיבור בין 2 הצדדים. זה חיוני מכיוון שמספר יישומים יכולים לפעול על מארח יחיד, כל אחד דורש ערוץ תקשורת ייחודי משלו.

(זה מושג בדרך כלל באמצעות מזהים כגון מספרי יציאה ב-TCP או מזהי חיבור בפרוטוקולים כמו QUIC) כאשר מנה מגיעה, פרוטוקול התחבורה משתמש במזהים אלה כדי לקבוע לאיזה אפליקציה או שירות הנתונים שייכים, ומבטיח שהנתונים מועברים לנקודת הקצה הנכונה.

2) העברת נתונים אמינה

על פרוטוקול התחבורה לדאוג למסירת נתונים אמינה אשר תבטיח שהנתונים הנשלחים מהשולח יגיעו למקבל בצורה מדויקת ובסדר בהם נשלחו, ללא אובדן או אי דיוק במידע.
מספרי רצף או מזהים דומים משמשים כדי לעקוב אחר כל פיסת נתונים שנשלחת, מה שמאפשר למקלט לזהות מנות חסרות או משוכפלות.
המקבל שולח אישורים עבור מנות שהתקבלו, והשולח משדר מחדש כל מנות שאינן מאושרות בתוך מסגרת זמן מסוימת. טכניקה זו מנהלת את קצב העברת הנתונים על סמך יכולתו של המקלט לעבד את הנתונים, ומונעת חסימת ראש-קו שבו חבילה אחת מושהית יכולה לעכב את משלוח החבילות הבאות.
(ב-TCP, לכל בייט של נתונים מוקצה מספר רצף. אם קטע (מנה) אבד, המקלט יכול לבקש שידור חוזר של אותו קטע ספציפי, תוך הבטחת מסירת נתונים מלאה ונכונה).

3) ניהול קשר תעבורתי.

הקמה, תחזוקה וסיום קשרים בצורה מבוקרת ויעילה. זה כולל הגדרה של פרמטרי חיבור ראשוניים, תחזוקה שוטפת של מצב החיבור וסגירה נכונה של החיבור. מזהים ייחודיים מבטיחים שניתן לנהל ולהפנות כל חיבור באופן מובהק, גם אם קיימים מספר חיבורים בין אותם מארחים.

נהלים להפעלה (לחיצת יד) ולסיום (סגירת) חיבורים מבטיחים שהקצאת משאבים והקצאת משאבים מטופלים בצורה נקייה. ובמהלך חיי החיבור, מחליפים מידע בקרה כגון עדכוני בקרת זרימה, גדלי חלונות ואותות בקרת גודש כדי לנהל את מצב החיבור.

4) בקרת עומס ברשת.

הפרוטוקול צריך לשלוט בבקרת הגודש ולמנוע גודש ברשת על ידי שליטה בקצב שליחת הנתונים. מה שמבטיח שהרשת תוכל לטפל בנתונים ללא אובדן מנות ועיכובים משמעותיים.

5) אבטחה.

הפרוטוקול צריך ליישם מנגנוני אבטחה אשר יגנו על הנתונים במהלך השידור מפני גישה בלתי מורשית, שיבוש הקשר והאזנת סתר, למנוע גישה בלתי רצויה לנתונים המועברים,
לאמת את זהותם של הצדדים המתקשרים כדי להבטיח שהנתונים מוחלפים בין שני הצדדים אשר ייסדו את הקשר. מבטיח שהנתונים לא שונו במהלך השידור.
(פרוטוקולי אבטחה של שכבת תחבורה כמו TLS מספקים הצפנה לנתונים המועברים באמצעות TCP)

שאלה 3: תארו את אופן פתיחת הקשר ב-QUIC . כיצד הוא משפר חלק מהחסרונות של TCP שתיארתם בסעיף 1 ?

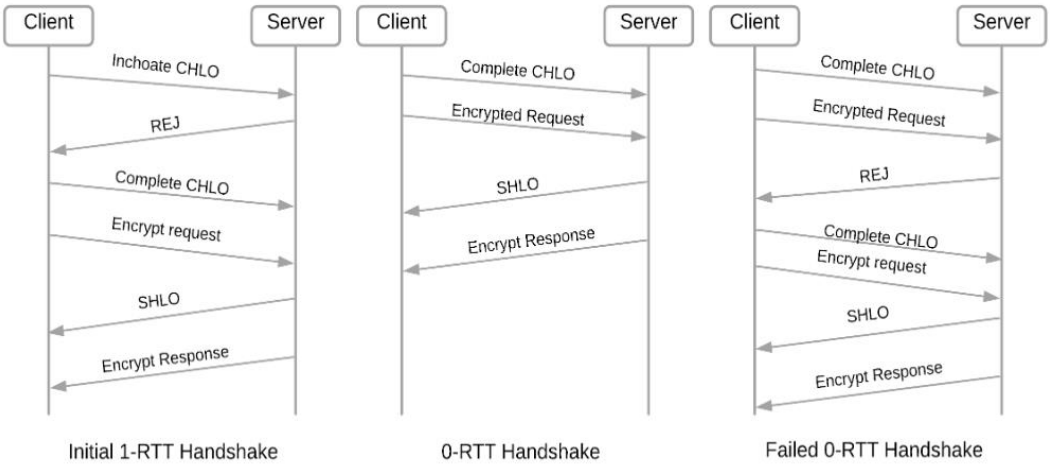
1) הלקוח שולח חבילת Initial לשרת המכילה הודעת ClientHello כדי ליזום את לחיצת היד של TLS. הודעה זו כוללת את פרמטרים של QUIC, חלק מהמפתח הציבורי שלו להחלפת מפתחות ואת חבילות הצופן הנתמכות, חילופי מפתחות Diffie-Hellman : גם לקוח וגם שרת מחליפים פרמטרים של Diffie-Hellman כדי ליצור מפתח מוצפן ולאפשר החלפה באופן מאובטח.

2. השרת מגיב עם שליחת חבילת Initial ללקוח המכילה הודעת ServerHello להמשך התהליך.

3. הלקוח משלים את הצד שלו בלחיצת היד של TLS, ומאשר את הקמת מפתחות הצפנה ואת הקשר האבטחה. השרת משלים את הצד שלו בלחיצת היד של TLS, ומסיים את הגדרת החיבור המאובטח.

הפרוטוקול משפר חלק מהחסרונות של פרוטוקול TCP:

1. **תלות בכתובת IP:** מאפשר במהלך התהליך המתואר לעיל לנהל את משאבי החיבור connection IDE's כך שבמקרה של שינוי כתובת IP של אחד הצדדים במהלך ההתקשרות, החיבור ימשיך לעבוד .
2. **עיכוב בהקמת החיבור:** מקטין את העיכוב בהקמת החיבור (ביחס ל-TCP) ע"י איחוד לחיצת היד של האבטחה והתחבורה ל-1RTT (לחיצת יד QUIC - TLS משלב את לחיצת היד של TLS ישירות בפרוטוקול שלו, כלומר המשא ומתן ההצפנה להקמת חיבור מאובטח מתרחש במקביל ללחיצת היד התחבורה (מה שמקטין את מספר הנסיעות הלוך ושוב הנדרשות בהשוואה ללחיצות היד הנפרדות ב-TCP + TLS*).
3. **העברת נתונים ב 0-rtt:** לאחר יצירת החיבור, השרת אינו דורש חיבור נוסף במידה והלקוח מתנתק ובכך הוא "זוכר אותו" ומדלג על החיבור ההתחלתי בניגוד לTCP.



שאלה 4: תארו בקצרה את מבנה החבילה של QUIC . כיצד הוא משפר חלק מהחסרונות של TCP שתיארתם בסעיף א?

בפרוטוקול QUIC קיים שימוש במספרי זהות ייחודיים לשתי הקצוות שנקרא Connection ID בשביל שתתאפשר זיהוי הצדדים הקשר כך שכל צד אינו תלוי בIP שלו והקשר נשמר גם אם התחלף הIP באמצע התהליך (למשל מעבר מ WIFI לסלולר). בכך הפרוטקול מוודא שהחבילות אכן נשלחות ליעדן הרצוי.

חבילה מחולקת לFrames שנשלחים במספר זרימות.

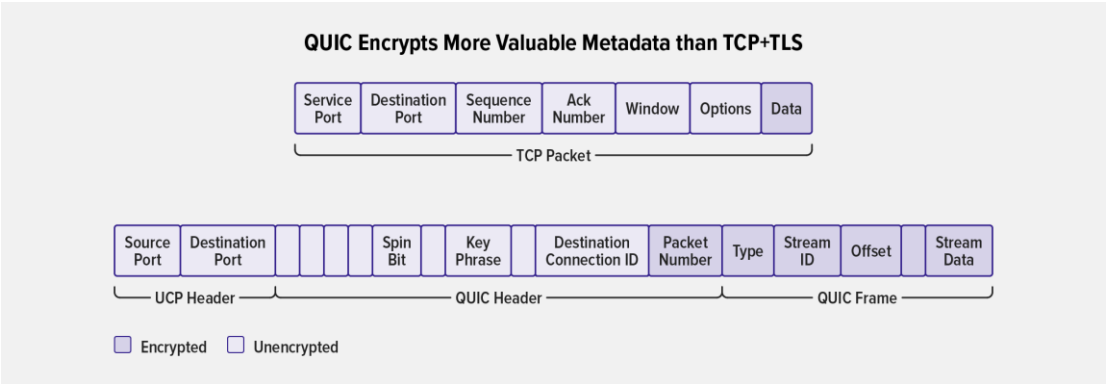
מבנה החבילה של QUIC מורכב משני פורמטים עיקריים : פורמט הכותרת הארוכה ופורמט הכותרת הקצרה .

1. long header - משמש להקמת החיבור, פורמט הכותרת הארוכה מכיל מידע חיוני להגדרת הקשר. הוא כולל בתוכו נתונים נחוצים לתקשורת ראשונית :

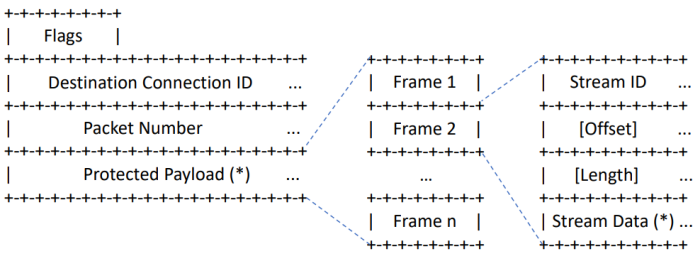
- Source connection ID (מזהה חיבור של המקור)
- Destination connection ID (מזהה חיבור של היעד)
- Packet type סוג החבילה (Initial; Close connection; Handshake)
- Version (גרסת QUIC של החבילה)

2. short header - לאחר יצירת החיבור, מנות עוקבות משתמשות בפורמט הכותרת הקצרה לצורך יעילות. פורמט זה כולל רק את שדות הכותרת החיוניים הדרושים לתקשורת שוטפת, וכולל בתוכו :

- Destination connection ID (מזהה חיבור של היעד)
- packet number – מס' הפקטה (כפי שיוסבר בשאלה 5)



Frames:



- Sream ID – מספר זיהוי הזרימה שבה מועבר הframe
- Offset – מס׳ סידורי של הפריים ב-stream (משמש לשחזור סדר הframes בסטרים)
- Length – גודל הframe (בביטים)
- Stream data – המידע (עצמו שנשלח)

מבנה של חבילת הQUIC משפרת חלק מהחסרונות שציינו לגבי TCP בכמה אופנים:

1. חיבור בין בקרת העומס ברשת לבין בקרת האמינות: בעוד שבTCP טיפול בבקרת העומס ברשת לבין בקרת האמינות מתבצעת במנגנון אחד (הקטנה גודל החלון), פרוטוקול QUIC מתמודד עם זאת בשני מנגנונים שונים. עם בקרת העומס – מתמודד באמצעות packet number המנהל מעקב על מס׳ חבילות וסדר הגעתן ובכך יודע בדיוק איזו חבילה חסרה ולא יעכב את שאר החבילות (בנוסף, QUIC תומך בסטרימים מרובים בתוך אותו חיבור, וכל סטרים מנוהל באופן עצמאי מבחינת בקרת העומס. כך, אם סטרים אחד מתעכב או יש אובדן חבילות, זה לא משפיע על שאר הסטרימים באותו החיבור). עם בקרת האמינות מתמודד באמצעות frame offset (שחזור סדר הframes ע״מ לראות שאכן כל החבילה הגיעה ואם לא, אז איזה חלק בדיוק חסר).
2. חסימת ראש קו - QUIC מאפשר הטמעת מסגרות מרובות בתוך חבילה, מה שמאפשר שידור של סוגי נתונים מגוונים בתוך אותה חבילה תוך הישארות בתוך מגבלת יחידת השידור המקסימלית (MTU) בכך אם נאבדה פקטה אחת, היא לא חוסמת את שאר הפקטות.
3. תלות בכתובות IP קבועות - QUIC משתמש במזהי חיבור (CID) ב-header לניתוב מנות לשרתי היעד שלהם. גישה זו מסייעת להבטיח שהמנות מועברות בצורה מדויקת גם בתרחישים שבהם כתובות הפרוטוקול הבסיסיות משתנות, ומספקת חוסן ואמינות במסירת מנות, ונותן פתרון למקרה בו כתובת IP תשתנה במהלך החיבור. (חיסרון של TCP בו ישנה תלות בכתובת ה-IP לקיום הקשר).
4. יעילות כותרות הפרוטוקול - ב-QUIC יש שני סוגים של long header – headers ארוכות ליצירת חיבור ו- short header להעברת נתונים עוקבים. שימוש בסוג ע״פ מידת הצורך נותנת פתרון יעיל בזיכרון בשליחת המידע הרלוונטי עבור כל מקרה לגופו.

שאלה 5: מה QUIC עושה כאשר חבילות מגיעות באיחור או לא מגיעות בכלל ?

לפרוטוקול QUIC שני מנגנונים לפיהם יחליט האם להכריז על אובדן החבילה או לא.

- מבוסס מספר חבילה - לכל חבילת QUIC מוקצה מספר חבילה ייחודי (packet number) שגדל באופן מונוטוני, המציין את סדר השידור של מנות. מערכת מספור ייחודית זו מסייעת במעקב ובניהול מדויק של העברת מנות בתוך החיבור. כאשר receiver מקבל חבילה, הוא שולח ACK בחזרה לשולח עם מספרי חבילה של החבילה שקיבל, וכאשר מגיע ACK, QUIC בודק את המספר הסידורי של חבילה עבורה התקבל האישור ויכריז על אובדן כל החבילות ממספר סידורי נמוך יותר מזו שהתקבלה עבורן טרם נשלח ACK.

- מבוסס זמן – מנגנון שבו המערכת מזהה שחבילה אבדה אם היא לא קיבלה אישור (ACK) בפרק זמן מסוים. בכל חיבור QUIC, השולח מודד את זמן ה-RTT, שזה הזמן שלוקח לחבילה להגיע ליעדה ולקבל אישור (ACK) בחזרה. הפרוטוקול משתמש במושג של RTT smoothed (זמן סיבוב ממוצע מחושב) ו-RTT variance (שונות בזמן הסיבוב) כדי לקחת בחשבון את התנודתיות ב-RTT, ולהתאים את מנגנון זיהוי החבילות האבודות. אם לא מתקבל ACK לחבילה בפרק זמן מסוים (Timeout) שנקבע על פי ה-RTT הנוכחי, השולח מניח שהחבילה אבדה. ה-Timeout מחושב בדרך כלל ע״פ ערכי הRTT שציינו כעת. לדוגמא, אם ה-RTT הממוצע הוא 100 מילי-שניות, והשונות היא 20 מילי-שניות, אז ה-Timeout יהיה 100 + (4 × 20) = 180 מילי-שניות.

כאשר מוכרז על אובדן חבילה, ה frames שאבדו יהיה בחבילות חדשות (סידור ה frames יתבצע ע״י סדר הoffset) עם מספר חבילה שונה כלומר יוקצה לשידורה החוזר מספר חבילה חדש היא תשלח. חבילות חדשות עוקבות ממשיכות מהמספר הבא אחרי החבילה האחרונה ששודרה (לדוגמא, אם חבילה 5 הייתה החבילה המשודרת מחדש, החבילה החדשה הבאה תהיה 6).

על ידי הקצאת מספרי מנות חדשים לכל חבילה חדשה או משודרת מחדש, QUIC מבטיח מעקב וזיהוי אובדן יעיל ותהליכי שידור חוזר. שיטה זו עוזרת למנוע בלבול בין מנות מקוריות לשידור חוזר ושומרת על שלמות רצף השידור.

שאלה 6: תארו את בקרת העומס (congestion control) של QUIC .

בקרת הגודש ב-QUIC היא מנגנון המנהל את קצב זרימת הנתונים בכדי למנוע עומס ברשת ולהבטיח העברת נתונים יעילה.

נזכיר כי QUIC משתמש במספרי מנות לבקרת גודש, ו offset frame לבקרת אמינות.

בדומה לבקרת גודש TCP, QUIC משתמש בסכימת בקרת גודש המבוססת על חלון המגבילה את המספר המרבי של בתים שיהיו לשולח במעבר בכל עת.

QUIC לא שואפת לפתח אלגוריתמים חדשים משלה לבקרת גודש, ולא להשתמש באף אחד ספציפי (למשל, Cubic). השרת מספק אותות גנריים לבקרת גודש, והלקוח חופשי ליישם מנגנוני בקרת גודש משלו. כדי למנוע הפחתת חלון גודש מיותר, הפרוטוקול אינו ממוטט את חלון הגודש אלא אם כן הוא מזהה גודש מתמשך.

כאשר שתי חבילות הדורשות אישור מוכרזות כאבודות, עומס מתמשך ייווצר אם אף אחת מהחבילות שנשלחות ביניהן לא תאושר, ההפרש בין זמן שליחתן לבין משך זמן הגודש המתמשך {המחושב על סמך ה-RTT הממוצע (smoothed_rtt), הסטייה של דגימות ה-(rttvar) RTT והזמן המרבי שהשולח עשוי לעכב את שליחת האישור} שרת הQUIC יקטין את הסיכוי לגרום לגודש לטווח קצר על ידי הבטחת מרווח שליחת המנות ע״י חישוב על סמך RTT שהזכרנו קודם.

בפרויקט מימשנו את אלגוריתם NEW RENO עליו יפורט בהרחבה בהמשך.

חלק "רטוב"

חלק א: הסבר על מימוש הקוד.

בחלק זה, נעסוק בהסבר מפורט על מימוש הקוד שבוצע בפרויקט. מטרת חלק זה היא לספק תובנה מעמיקה על אופן העבודה של התהליכים והמבנים שהוגדרו במסגרת הפרויקט, עם דגש על ההיגיון שמאחורי הבחירות התכנוניות שבוצעו.

נבאר מושגים טכניים ונבצע סקירה של המרכיבים המרכזיים בקוד, כגון מבני נתונים, אלגוריתמים לבקרת עומס, מנגנוני פתיחת קשר, בניית סוגי חבילות, מימוש מעקב RTT ועוד. ההסבר יתמקד הן בתיאור המבני של הקוד והן בתהליך המימוש.

קובץ QUIC_API.py:

קובץ זה מהווה את הליבה של הפרויקט, ומטרתו לספק ממשק API למימוש פרוטוקול QUIC. בפרויקט זה, הקובץ משמש כבסיס לתקשורת מאובטחת ומהירה בין לקוח לשרת, תוך התמקדות בניהול תעבורה יעיל, בקרת עומס, וניהול חבילות ברמת דיוק גבוהה.

הקובץ כולל מימוש של פונקציות מרכזיות לניהול חיבורי QUIC בין לקוח ושרת כגון יצירה וסגירת חיבורים, שליחת וקבלת נתונים וכהרחבה לכך, שליחת וקבלת קבצים והצפנה של תעבורת הנתונים.

פונקציות:

1. **פונקציות הצפנה encrypt ופענוח decrypt:**
אמנם, נדרשנו לתת מימוש מנוון לפרוטקול אך ברצוננו לתת דימוי זהה ככל הניתן לפרוטקול והצפנת ופענוח נתוני החבילות הן חלק מרכזי בפרוטוקול QUIC אשר נעשה ע"י החלפת מפתחות הצפנה ציבוריים ליצירת מפתח סימטרי באמצעות Diffie Hellman שמאפשר לשני הצדדים להעביר נתונים באופן מאובטח.

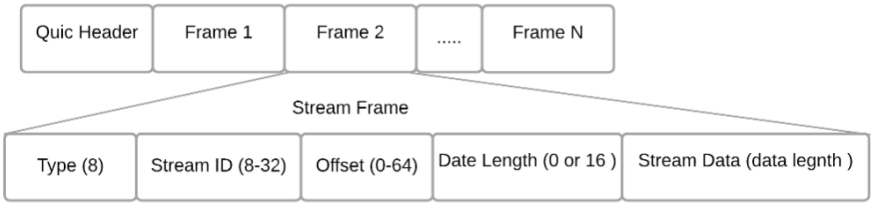
מימוש ההצפנה: הפונקציה encrypt אחראית על הצפנת הנתונים באמצעות מפתח הצפנה (key) ו-Nonce (מספר חד-פעמי). היא יוצרת אובייקט cipher באמצעות ספריית cryptography, ומבצעת את ההצפנה על המידע הנכנס. הפונקציה מחזירה את הטקסט המוצפן (ciphertext) ואת תג האימות (tag), המשמשים לאימות שלמות הנתונים במהלך הפענוח.

מימוש הפענוח: הפונקציה decrypt מבצעת את הפענוח של הנתונים המוצפנים, תוך שימוש באותם מפתח הצפנה ו-Nonce, יחד עם ה-tag שאומת במהלך ההצפנה.
הפונקציה יוצרת אובייקט cipher דומה לזה שבפונקציית ההצפנה, ומשתמשת בו כדי לפענח את הטקסט המוצפן חזרה לנתונים המקוריים.

2. **פונקציית generate_cid:**

יצירת מזהה ייחודי Connection ID אשר ישתמשו השרת והלקוח (תזכורת: CID הוא מזהה ייחודי שנועד להבחין בין חיבורים שונים בפרוטוקול QUIC. הוא חשוב במיוחד בסביבה שבה ניתן לשנות כתובות IP במהלך חיבור, כמו בעת מעבר בין רשתות (למשל, מ-WiFi לנתוני סלולר).

3. **פונקציית build_frame:**



פונקציה זו מממשת את המבנה הבסיסי של פריים ב-QUIC, עם כותרת שמכילה מידע חשוב על סוג הפריים, הסטרים אליו הוא שייך, המיקום של הנתונים בסטרים (Offset), ואורך הנתונים. בסוף התהליך, היא מחזירה את הפריים השלם המוכן לשידור.
הפונקציה מבטיחה שכל נתון שנשלח נמצא במיקום הנכון בתוך הסטרים, ושומרת על הסדר הנכון של הנתונים במסגרת הסטרים.
frame_type: סוג הפריים מקודד כבתים בודדים (1 byte).
stream_id: מזהה הסטרים מיוצג כ-4 בתים (4 bytes).
offset: מיקום הנתונים (Offset) בסטרים, מיוצג כ-8 בתים (8 bytes).
data_length: אורך הנתונים בפריים, מיוצג כ-2 בתים (2 bytes).

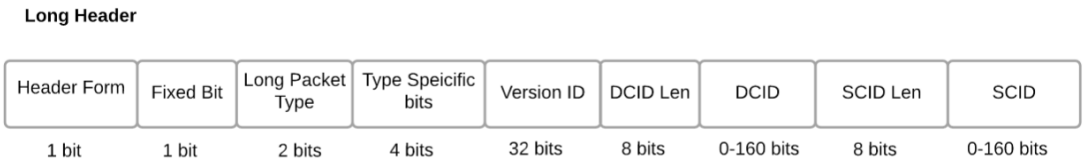
4. **פונקציית parse_frame:**

פונקציה זו אחראית על פירוק פריים (Frame) שנשלח בפרוטוקול QUIC, והפקת המידע החשוב שנמצא בתוכו, כך שניתן יהיה להשתמש בו לצורך המשך עיבוד הנתונים.
הפונקציה מתחילה בפענוח סוג הפריים (frame_type), הממוקם בתו הראשון של הפריים. לאחר מכן, היא מפענחת את מזהה הסטרים (stream_id) מהביתים הבאים (בין 1 ל-5). היא ממשיכה לפענוח המיקום בסטרים (offset), הממוקם בין הביתים 5 ל-13.
לבסוף, היא מפענחת את אורך הנתונים (data_length) מהביתים 13 ו-14.

שליפת הנתונים: לאחר פענוח הכותרת, הפונקציה שולפת את הנתונים עצמם (data) מהפריים, בהתאם לאורך שנקבע בכותרת.

החזרת המידע: הפונקציה מחזירה את כל המידע שנשלף מהפריים: סוג הפריים, מזהה הסטרים, המיקום (offset), והנתונים.

5. פונקציית build_long_header_packet:



פונקציה זו נועדה לבנות חבילה עם כותרת ארוכה (Long Header) בפרוטוקול QUIC, המיועדת לשימוש בחיבורי רשת מורכבים, כמו חיבורי התחלה (initialization) או בעת מעבר בין גרסאות פרוטוקול.

קביעת הכותרת (Header) הראשונית: הפונקציה מתחילה בהגדרת ה-header form, אשר מסמן כי מדובר בכותרת ארוכה (Long Header), ובסימון bit מיוחד (fixed_bit) שתמיד מוגדר כ-1 עבור חבילות תקפות.

היא ממשיכה בקביעת סוג החבילה (packet_type) וביטויים ספציפיים לסוג החבילה (type_specific_bits).

אריזת הכותרת: הכותרת כוללת את הגרסה של הפרוטוקול (version) אשר מוגדר כקבוע לשם התרגול כ1 version, ואורכי המזהים של היעד (dest_cid) ושל המקור (source_cid). המזהים עצמם נכללים גם הם בכותרת. בנוסף, נכלל מספר החבילה (Packet Number) בתוך הכותרת כדי לעקוב אחרי סדר החבילות.

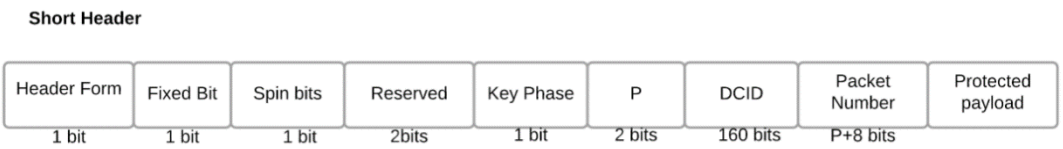
הצפנה ויצירת מטען מוצפן (Payload Encryption): הפונקציה יוצרת nonce (מספר חד-פעמי) אקראי לשם הצפנה בטוחה. לאחר מכן, היא מאחדת את כל ה-frames לתוך מטען אחד ומצפינה אותו באמצעות הפונקציה encrypt. הפלט המוצפן (ciphertext) והתווית (tag) משולבים יחד עם ה-header כדי ליצור את החבילה. לשם דימוי הפרוטוקול השתמשנו בlow_key קבוע אשר ישמש את חבילות הlong header מכיוון שעדיין עוד לא נוצר מפתח סימטרי.

החזרת החבילה: לבסוף, החבילה המוגמרת כוללת את הכותרת הארוכה, ה-nonce, הנתונים המוצפנים, ותווית האימות (tag), ומוחזרת כערך הפונקציה.

6. פונקציית parse_long_header_packet:

פונקציה זו מממשת את תהליך פירוק (parsing) של חבילה עם כותרת ארוכה בפרוטוקול QUIC. תפקידה הוא לפרק את החבילה ולהפיק את המידע הקריטי שנמצא בתוכה לצורך עיבוד נוסף. הפונקציה מחזירה את סוג החבילה (long_packet_type), גרסת הפרוטוקול (version), מזהי היעד והמקור (dest_cid ו-source_cid), מספר החבילה (packet_number), ומערך ה-frames המפוענחים.

7. **פונקציית build_short_header_packet:**



פונקציה זו מיועדת לבנות חבילה עם כותרת קצרה (Short Header) בפרוטוקול QUIC. חבילות עם כותרת קצרה משמשות בעיקר להעברת נתונים לאחר שהחיבור הוקם, והן נועדו להיות קלות ויעילות יותר מאשר חבילות עם כותרת ארוכה, שכן הן כוללות פחות מידע בכותרת. חבילות עם כותרת קצרה משמשות בעיקר לתקשורת מתמשכת, שבה אין צורך לשאת את כל המידע הנוסף שנדרש בשלב ההתחלתי של החיבור.

שלבי הפעולה של הפונקציה:

הפונקציה מתחילה ביצירת nonce אקראי, המשמש לצורך הצפנת הנתונים בצורה בטוחה. הנתונים שיש לשדר מאוחדים ל-payload אחד גדול המורכב מכל ה-frames.

בהתאם לערך של key_phase, נעשה שימוש במפתח הצפנה מתאים (ack_key או quic_key_symmetric) לצורך הצפנת הנתונים. פעולה זו מוודאת שהנתונים מוצפנים ומאובטחים לפני שליחתם.

קביעת אורך מספר החבילה: אורך מספר החבילה נקבע כ-2 בתים (2 bytes), וזה משוקף ב-2 הביטים המייצגים את אורך המספר בכותרת החבילה.

בנייה של הבייט הראשון של הכותרת: הפונקציה מגדירה את מאפייני הכותרת, כולל בחירה ב-header form שמציין שמדובר בכותרת קצרה, ה-fixed bit שמסמן שהחבילה תקינה, ה-spin bit שמייצג את הזמן (אמצעי לניטור הRTT), ה-key phase bit שמציין את המפתח שנבחר להצפנה, והייצוג של אורך מספר החבילה.

בנייה של הכותרת הקצרה: הפונקציה בונה את הכותרת הקצרה באמצעות ספריית struct, עם השדות הבאים: ה-byte הראשון, מזהה היעד (dest_cid), ומספר החבילה.

הפונקציה מחזירה את החבילה המתקבלת.

8. **פונקציית parse_short_header_packet:**

פונקציה זו מממשת את תהליך פירוק (parsing) של חבילה עם כותרת קצרה (Short Header) בפרוטוקול QUIC. תפקידה הוא לחלץ את כל המידע החשוב שנמצא בחבילה לצורך עיבוד נוסף.

9. **פונקציית send_ack:**

הפונקציה send_ack אחראית על שליחת חבילת ACK (אישור קבלה) לשרת. היא בונה Frame של ACK באמצעות הפונקציה build_frame, ולאחר מכן אורזת אותו בחבילה עם כותרת קצרה (Short Header) באמצעות build_short_header_packet. החבילה נשלחת

לשרת דרך ה-socket שצוין, והפונקציה מחזירה את גודל החבילה שנשלחה, או False אם לא כל החבילה נשלחה בהצלחה.

10. **פונקציית send_data:**

הפונקציה send_data שולחת חבילת נתונים לשרת. היא יוצרת Frames של נתונים (Data Frames) בעזרת build_frame מהdata שקיבל ולאחר מכן אורזת אותם בחבילה עם כותרת קצרה באמצעות build_short_header_packet. החבילה נשלחת לשרת דרך ה-socket, והפונקציה מחזירה את גודל החבילה שנשלחה. אם שליחת החבילה נכשלה, הפונקציה מדפיסה הודעת שגיאה ומחזירה False.

11. **פונקציית receive_data:**

הפונקציה receive_data מקבלת נתונים משרת. היא מאזינה לחבילה שנשלחת דרך ה-socket, ומפענחת אותה באמצעות הפונקציה parse_short_header_packet. לאחר הפענוח, הפונקציה מחזירה את ה-frames שהתקבלו, יחד עם מזהה היעד (dest_cid) ומספר החבילה (packet_number). אם לא מתקבלים נתונים, היא מחזירה False ומדפיסה הודעת שגיאה.

12. **פונקציית receive_ack:**

הפונקציה receive_ack נועדה לקבל חבילת ACK משרת. היא מאזינה לחבילה שנשלחת דרך ה-socket, מפענחת אותה באמצעות parse_short_header_packet, ובודקת אם ה-frame הראשון שהתקבל הוא ACK. אם כן, היא מחזירה packet_number (בכך מי ששלח את הנתונים, ישווה בין הpacket_number של החבילה ששלח לבין הערך שקיבל מהפונקציה לוודא שאין איבוד פקטות) ואם לא, היא מדפיסה הודעת שגיאה ומחזירה False.

תהליך הHandshake:

הפונקציות הבאות מיישמות את תהליך ה Handshake-ב QUIC - הכולל שליחה וקבלת הודעות חיוניות בין הלקוח לשרת להקמת חיבור מאובטח. כל פונקציה ממוקדת בשלב מסוים בתהליך, וכולן יחד מאפשרות הקמת חיבור מהיר ויעיל, תוך שמירה על אבטחת המידע המועבר.

תהליך ה-Handshake בפרוטוקול QUIC הוא תהליך קריטי להקמת חיבור מאובטח בין לקוח לשרת. התהליך מבוצע בצורה מהירה ומאובטחת יותר בהשוואה לפרוטוקולים אחרים כמו TCP. במהלך ה-Handshake, מתבצע החלפה של מפתחות הצפנה, אימות זהויות, וקביעת פרמטרים לשימוש במהלך החיבור. ה-Handshake ב-QUIC מאפשר גם התחלה מהירה של תקשורת נתונים, באמצעות מה שנקרא RTT-0 (זמן אפס לחזרה), שבו הלקוח יכול לשלוח נתונים כבר עם ההודעה הראשונה.

צד הלקוח:

13. **פונקציית send_first_chlo:**

פונקציה זו שולחת את ההודעה הראשונה בתהליך ה-Handshake, המכונה (CHLO) "ClientHello". ההודעה נבנית כ-Frame מסוג FCHLO, ומקודדת בתוך חבילה עם כותרת ארוכה (Long Header). החבילה נשלחת לשרת, והפונקציה מחזירה את גודל החבילה שנשלחה או False במקרה של כשל.

14. **פונקציית receive_rej:**

פונקציה זו מקבלת את ההודעה השנייה בתהליך ה-Handshake, המכונה "REJ" (Rejected). היא מאזינה לחבילה מהשרת, מפענחת אותה, ובודקת האם מדובר בחבילת REJ. אם כן, הפונקציה מחזירה את מזהה החיבור של השרת (server CID), אחרת מחזירה False ומדפיסה הודעת שגיאה.

15. **פונקציית send_complete_chlo:**

פונקציה זו שולחת את ההודעה המלאה של "(CHLO) "ClientHello" לאחר קבלת ה-REJ מהשרת. ההודעה כוללת את המפתח הציבורי של הלקוח ומספר הפעמים שהוא רוצה לשלוח את הקובץ לשרת. הפונקציה בונה את ההודעה כ-Frame מסוג SCHLO ושולחת אותה לשרת.

16. **פונקציית receive_shlo:**

פונקציה זו מקבלת את ההודעה השלישית בתהליך ה-Handshake, המכונה (SHLO) "ServerHello". היא מאזינה לחבילה מהשרת, מפענחת אותה, ובודקת האם מדובר בחבילת SHLO. אם כן, היא מחלצת את המפתח הציבורי של השרת מהחבילה ומחזירה אותו ובכך נחתם תהליך הhandshake.

צד השרת:

17. **פונקציית receive_first_chlo:**

פונקציה זו מיועדת לקבל את ההודעה הראשונה מהלקוח, המכונה "ClientHello" (CHLO). היא מאזינה לחבילה מהלקוח, מפענחת אותה ובודקת אם היא מהסוג INITIAL_REQUEST. הפונקציה מחזירה את מזהה החיבור של הלקוח (client CID) ואת הכתובת שלו.

18. **פונקציית send_rej:**

פונקציה זו שולחת את ההודעה "REJ" ללקוח, לאחר קבלת ה-CHLO. ההודעה נבנית כ-Frame מסוג REJ, ונשלחת כתגובה להודעת ה-CHLO הראשונה שנשלחה מהלקוח.

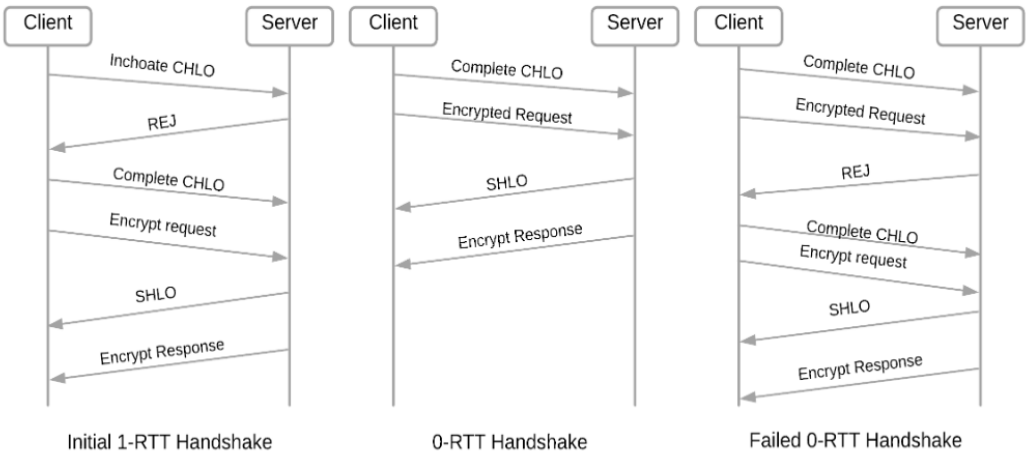
19. **פונקציית receive_complete_chlo:**

פונקציה זו מקבלת את ההודעה השלמה של "ClientHello" מהלקוח, לאחר שליחת ה-REJ. היא מאזינה לחבילה מהלקוח, מפענחת אותה, ובודקת אם מדובר בהודעת

INITIAL_COMPLETE. הפונקציה מחזירה את המפתח הציבורי של הלקוח ואת מספר הפעמים שהלקוח מבקש לשלוח את הקובץ לשרת.

20. פונקציית send_shlo:

פונקציה זו שולחת את ההודעה (SHLO) "ServerHello" ללקוח, לאחר קבלת ההודעה השלמה של CHLO מהלקוח. היא בונה את ההודעה כ-Frame מסוג SHLO, הכולל את המפתח הציבורי של השרת, ושולחת אותה ללקוח.



פונקציות שליחה וקבלת קובץ:

21. פונקציית send_file:

הפונקציה אחראית על שליחת קובץ מהלקוח לשרת בפרוטוקול QUIC. הפונקציה מחלקת את הקובץ לחלקים קטנים בגודל של KB1 כל אחד ושולחת אותם בזה אחר זה לשרת, תוך ניהול נכון של מספרי החבילות וקבלת אישור (ACK) עבור כל חלק שנשלח.

שלבי הפעולה של הפונקציה:

בדיקת מפתח סימטרי: הפונקציה מתחילה בבדיקה אם המפתח הסימטרי הוקם בהצלחה. אם לא, הפונקציה מדפיסה הודעת שגיאה ומפסיקה את הביצוע.

קריאת הקובץ ושליחת נתונים: הקובץ נפתח לקריאה במצב בינארי, ומחולק לחלקים בגודל של KB1. כל חלק שנקרא מהקובץ נשלח לשרת באמצעות הפונקציה send_data, שתשלח את החבילה לשרת ותמתין לקבלת אישור (ACK) חזרה.

ניהול רשימת החבילות: לאחר שליחת כל חלק מהקובץ, הפונקציה רושמת את זמן שליחת החבילה וזמן קבלת ה-ACK ברשימת החבילות (packet_list), כדי לעקוב אחר ביצועי ההעברה.

טיפול בשגיאות: אם הקובץ לא נמצא, הפונקציה מדפיסה הודעת שגיאה מתאימה ומפסיקה את הביצוע. אם יש בעיה בשליחת החבילה או בקבלת ה-ACK, הפונקציה תדפיס הודעת שגיאה ותפסיק את הביצוע.
השלמת תהליך שליחת הקובץ: בסיום שליחת כל חלקי הקובץ וקבלת כל האישורים, הפונקציה מדפיסה הודעה שהקובץ נשלח בהצלחה ומחזירה את מספר החבילה האחרון שנשלח.

22. פונקציית receive_file:

הפונקציה מיועדת לקבל קובץ מהלקוח, לשמור אותו בשרת, ולשלוח חזרה אישורי קבלה (ACK) לכל חבילה שהתקבלה. הפונקציה מתחילה בחישוב מספר החבילות הצפויות להתקבל בהתבסס על גודל הקובץ (MB2) וגודל כל חלק (KB1).

הקובץ החדש נפתח במצב כתיבה בינארי, וכל חבילה שמתקבלת נכתבת לתוכו.

הפונקציה מקבלת את החבילות מהלקוח אחת אחרי השנייה באמצעות הפונקציה שפירטנו עליה קודם receive_data, ומפענחת את ה-frames המתקבלים. כל frame מכיל את חלק הנתונים שמיועד להיכתב לקובץ.

לאחר כתיבת הנתונים מהחבילה, הפונקציה שולחת ללקוח חבילת ACK כדי לאשר שהחבילה התקבלה בהצלחה.
בדיקת סיום קבלת הקובץ: מספר החבילות שהתקבלו נספר, ובסיום קבלת כל החבילות הצפויות, הפונקציה מסיימת את התהליך.
טיפול בשגיאות: אם הקובץ לא נמצא או מתרחשת שגיאה במהלך הקבלה, הפונקציה מדפיסה הודעת שגיאה ומפסיקה את הביצוע.

23. פונקציית send_close:

הפונקציה נועדה לסגור את החיבור בין הלקוח לשרת בצורה מסודרת בפרוטוקול QUIC. היא מבטיחה שהחיבור ייסגר בצורה תקינה ושכל החבילות שקשורות לסגירה יישלחו בהצלחה לשרת, תוך שמירה על עקביות במבנה החבילות ומספרי החבילות.

24. פונקציית receive_close:

קבלת הודעת CLOSE ובכך השרת מודע לכך לתהליך סגירת החיבור.

25. פונקציית simulate_network_conditions:

הפונקציה נועדה לדמות תנאי רשת משתנים במהלך העברת נתונים. באמצעות סימולציה זו, ניתן לבחון כיצד מערכת התקשורת מתמודדת עם עיכובים (delays) שונים ברשת, וכיצד הדבר משפיע על ביצועי ההעברה. נצטרך פונקציה זו ע"מ לבצע ניסוי במתבקש בנושא שונות RTT. הניסוי יקרה בצד הserver ועליו נרחיב בהמשך.

קובץ Server.py :

קובץ server.py מיישם את תהליך ההתקשרות והתקשורת של שרת בפרוטוקול QUIC מצד השרת ללקוח, כולל תהליך ה-handshake, ניהול בקרת העומס (Congestion Control), וקבלה של קבצים בצורה מאובטחת באמצעות הפונקציות שציינו קודם לכן בקובץ Quic_api.py . מטרת הקובץ היא ליצור חיבור אמין ומאובטח עם הלקוח, לנהל את בקרת העומס בצורה יעילה, ולטפל בקבצים הנשלחים בצורה מאובטחת תוך שימוש בהצפנה.

לפני שהשרת סוגר את החיבור עם הלקוח, מבצע את ניסוי **שונות ב RTT** ומפיק גרף מהנתונים שהתקבלו באמצעות הרשימה המקושרת. נרחיב על הניסוי בהמשך.

1. קבלת פרמטרים והכנה: הקובץ מתחיל בפרסור של פרמטרי ההרצה (כגון פורט). אם הקלט אינו תקין, מתבצע ניסיון חוזר עד 5 פעמים לפני יציאה מהתוכנית. מתבצע יצירת תיקייה לשמירת הקבצים המתקבלים מהלקוח ופתיחת סוקט UDP לשמירה על חיבור עם הלקוח.
2. תהליך ה-Handshake עם הלקוח: התהליך מתחיל בקבלת הודעת ClientHello הראשונה מהלקוח וכולל שליחת הודעת REJ ללקוח. השרת מוודא קבלת הודעות ACK עם מספר חבילה מתאים מהלקוח ומעדכן את רשימת החבילות עם זמני השליחה והקבלה של ההודעות. לאחר מכן, השרת מקבל הודעת ClientHello מלאה מהלקוח ומבצע יצירת מפתח סימטרי לצורך הצפנה. לאחר מכן, השרת שולח הודעת ServerHello שבו מצויין המפתח הציבורי של השרת ללקוח ומוודא קבלת ACK סופית להשלמת ה-handshake.
3. קבלת קבצים מהלקוח: השרת ממתין לקבלת הקובץ מהלקוח מספר פעמים כפי שהוגדר בפרמטרים הנשלחים. כל קובץ מתקבל ונשמר בתיקייה שהוגדרה מראש, והשרת שולח הודעות ACK לאחר כל קבלה מוצלחת של קובץ.
4. קבלת הודעת סגירה ושליחת ACK סופי: לאחר קבלת הקבצים, השרת ממתין להודעת סגירה מהלקוח ושולח ACK אחרון כדי לסגור את החיבור בצורה מסודרת.

מדידת RTT (Round Trip Time): השרת מחשב את ערכי ה-RTT ומבצע ניסוי סימולציה ליצירת גרף המתאר את ה-RTT באמצעות רשימת זמני חבילות מזויפת המדמה תנאי רשת משתנים (אופן המדידה והניסוי עצמו יורחב בהמשך).

סגירת השרת וניקוי קבצים: לבסוף, השרת סוגר את הסוקט, מוחק את הקבצים והתיקייה שנוצרו במהלך ההרצה, ומסיים את הפעולה.

קובץ Client.py :

הקובץ client.py מייצג את הצד של הלקוח בפרוטוקול QUIC. תפקידו העיקרי של קובץ זה הוא ליזום תקשורת עם השרת, לנהל את תהליך ה-Handshake לאבטחת התקשורת, לשלוח קובץ מספר פעמים לשרת (ע"פ הפרמטר שקיבל), ולסיים את החיבור בצורה מסודרת. לאחר סיום החיבור יוצגו נתוני RTT על אופן שליחת וקבלת פקטות תוך כדי מימוש בקרת עומס NEWRENO, את הנתונים ניתן לראות בגרף שהלקוח יוצר.

1. קריאת הפרמטרים והכנת הנתונים : הלקוח קורא את כתובת ה-IP של השרת, הפורט ומספר הפעמים שבהן הקובץ ישלח לשרת לאחר מכן נוצר קובץ אקראי בגודל MB2 לצורך השידור.
2. חיבור ו-Handshake עם השרת : הלקוח יוצר מזהה חיבור (CID) ומחבר את עצמו לשרת באמצעות UDP socket. תהליך ה-Handshake מתחיל בשליחת הודעת CHLO ראשונה לשרת. הלקוח מקבל את הודעת ה-ACK מהשרת, מבצע שליחת ACK נוספת, ומקבל הודעת REJ מהשרת כדי להמשיך את תהליך החיבור.תהליך ה-Handshake מסתיים לאחר קבלת הודעת ה-SHLO כאשר בתוכה קיימת המפתח הציבורי של השרת ע"מ הקמת מפתח סימטרי עבור הצפנה ופענוח הנתונים.
3. שליחת הקובץ לשרת : הלקוח שולח את הקובץ לשרת מספר פעמים לפי הדרישה. כל שידור כולל חלוקה של הקובץ לפריימים וניהול מספרי חבילות וספירת offset. לאחר כל שליחה, הלקוח מחכה לקבלת הודעת ACK מהשרת כדי לאשר את קבלת החבילה כאשר בודק אם מספר החבילה שמקבל מACK תואם למספר החבילה שנשלח.
4. סגירת החיבור : לאחר השלמת שליחת הקבצים, הלקוח שולח הודעת סגירה (CLOSE) לשרת ומחכה לאישור סופי (ACK) לסיום התהליך.
5. חישוב RTT ויצירת גרף : הלקוח מחשב את ערכי ה-RTT מהזמנים שנמדדו במהלך התקשורת עם השרת ומציג גרף שממחיש את ערכי ה-RTT לאורך השידור באמצעות מבנה נתונים packetlist שעליו נרחיב בהמשך.
6. סגירת הקובץ והחיבור : לאחר סיום כל הפעולות, הלקוח סוגר את הסוקט ומוחק את הקובץ שנוצר.

קובץ LinkedList.py :

הקובץ linkedlist.py הוא חלק עיקרי וחשוב במימוש התוכנית שלנו, שכן הוא אחראי על ניהול בקרת עומס באמצעות אלגוריתם NewReno בפרוטוקול QUIC, וממלא תפקיד מרכזי בניתוח מדדים כמו Round-Trip Time (RTT) ו-Congestion Window (cwnd). בקובץ זה, אנו יוצרים רשימה מקושרת (Linked List) המורכבת ממספר מחלקות ופונקציות המרכזיות לניהול פקטות מידע, ניתוח ביצועים בזמן אמת, **וסימולצית ניסוי של בקרת עומס ואובדן פקטות.**

Node :

מחלקת Node מהווה את הבסיס למימוש הרשימה המקושרת (Linked List) בקובץ linkedlist.py. כל אובייקט של מחלקה זו מייצג פקטה (Packet) בודדת שנשלחה והתקבלה במסגרת התקשורת בפרוטוקול QUIC. המחלקה מתעדת מידע חשוב עבור ניתוח ביצועי הרשת וניהול בקרת העומס באמצעות אלגוריתם NewReno.

מאפיינים עיקריים של מחלקת Node :

- packet_number : מזהה ייחודי של הפקטה, אשר משמש למעקב אחר סדר הפקטות שנשלחו וקיבלו אישור (ACK).
- ack_receive_time : זמן קבלת אישור ה-ACK עבור הפקטה, נמדד בזמן אמת עם קבלת התשובה מהנמען.
- packet_send_time : הזמן בו הפקטה נשלחה לראשונה. ערך זה נדרש לחישוב ה-RTT.
- sample_rtt : ה-RTT המדגמי המחושב כהפרש בין ack_receive_time ל-packet_send_time. ערך זה מספק מידע על הזמן שלוקח לפקטה להגיע לנמען ולקבל אישור חזרה.
- packet_size : גודל הפקטה, אשר משמש לבקרה על גודל חלון העומס (cwnd) במסגרת בקרת העומס.
- experiment_lost_flag : משתנה בוליאני המצביע אם הפקטה סומנה כאבודה במסגרת ניסוי.
- rtt_min : הערך המינימלי של ה-RTT עבור פקטה זו, כאשר התחלתי זהה ל-sample_rtt.
- rtt_smooth: ממוצע המוחלק לפי נוסחה עליה נרחיב בהמשך, המספק מידע יציב יותר על ביצועי הרשת. RTT
- rtt_var : משתנות RTT המשמשת לחישוב סטיות מהערך הממוצע ומסייעת בזיהוי מצבי עומס מתמשך.
- next : מצביע על ה-Node הבא ברשימה המקושרת, המאפשר התקדמות בין הפקטות השונות המאוגדות ברשימה.

PacketList:

מחלקת PacketList מייצגת רשימה מקושרת שמכילה את המידע על כל הפקטות שנשלחו והתקבלו במהלך תקשורת בפרוטוקול QUIC. המחלקה משחקת תפקיד מרכזי במעקב אחר ביצועי הרשת, חישוב מדדים חשובים כמו RTT, וניהול בקרת העומס באמצעות אלגוריתם NewReno. המחלקה מנהלת את הפקטות בעזרת אובייקטים מסוג Node, כאשר כל Node מייצג פקטה אחת. שים לב כי בשיטה הזו בחרנו לממש את מדידת ה RTT למרות שהיה ניתן להיעזר עם הspin bit שכלול כחלק ממימוש מבנה חבילת הQUIC אך בחרנו במימוש אחר.

פונקציות:

1. **פונקציית insert:** אחראית להוספת Node חדש לרשימה המקושרת שמייצגת פקטה שנשלחה וקיבלה ACK. הפונקציה מבצעת עדכונים קריטיים למדדי הרשת ומפעילה את אלגוריתם בקרת העומס NewReno. לאחר הוספת ה-Node החדש לרשימה, מתבצעים חישובי RTT שונים על ידי קריאה לשתי פונקציות: calculate_smoothed_rtt המחשבת את ערך ה-RTT המוחלק של ה-Node הנוכחי ו-calculate_rtt_variance המחשבת את המשתנות של ה-RTT, המייצגת את הסטיות בין ערכי ה-RTT המדגמיים לערכים המוחלקים. לבסוף, הפונקציה מפעילה את אלגוריתם בקרת העומס NewReno על ה-Node החדש באמצעות הקריאה לפונקציה new_reno_congestion_control אלגוריתם זה משתמש במידע שנאסף בנוגע לפקטה (כגון זמני שליחה וקבלת ACK) לצורך ניהול חכם של חלון הגודש (cwnd) וההחלטה על מצב בקרת העומס (כגון Slow Start, Congestion Avoidance, או Recovery).

2. **פונקציית calculate_min_rtt:** משמשת לחישוב ערך ה-RTT (Round Trip Time) המינימלי עבור כל Node ברשימה המקושרת. ערך ה-RTT המינימלי הוא מדד חשוב בניהול ביצועי הרשת, שכן הוא מספק אינדיקציה לזמן המהיר ביותר שנדרש לפקטה לנוע הלוך ושוב בין השולח למקבל.

3. **פונקציית calculate_smoothed_rtt:** מתמקדת בחישוב ערך ה- RTT (Round Trip Time) המוחלק, שהוא מדד המשמש לאמוד את זמן ההעברה של פקטה הלוך וחזור ברשת בצורה מדויקת יותר על ידי החלקה של תנודות ערכי ה-RTT המדגמיים. המטרה המרכזית היא להעריך את הזמן הצפוי להעברת פקטות באופן שמפחית את השפעת התנודות והחריגות בערכי ה-RTT המתקבלים.

   הנוסחה המשמשת לחישוב ה-RTT המוחלק היא:

   node.rtt_smooth = (1 - ALPHA) * prev_node.rtt_smooth + ALPHA * node.sample_rtt

   כאשר RTT_smooth הוא ערך ה-RTT המוחלק הנוכחי, שמשקף הערכה של זמן ההעברה האמיתי. RTT_smooth_previous הוא ערך ה-RTT המוחלק מהחישוב הקודם. RTT_sample הוא הערך ה-RTT המדגמי הנוכחי, כלומר הזמן שנמדד עבור

הפקטה האחרונה. (Alpha) α הוא קבוע החלקה המגדיר את המשקל של ערך ה-RTT המדגמי הנוכחי לעומת הערך המוחלק הקודם. ערך זה נע בין 0 ל-1, כאשר ערכים נמוכים יותר נותנים משקל רב יותר לערך הקודם.

רעיון החישוב:

איזון בין נתונים חדשים לישנים: הנוסחה מעניקה איזון בין ערך ה-RTT הקודם (RTT_smooth_previous) לבין המדידה הנוכחית (RTT_sample).

הפחתת השפעת תנודות פתאומיות: על ידי שימוש במקדם α, הנוסחה מאפשרת לחלק בצורה מבוקרת בין ערכים היסטוריים וערכים חדשים, וכך היא מספקת אומדן יציב יותר שמקטין את השפעתם של עיכובים חריגים על החישוב.

4. **פונקציית calculate_rtt_variance:** מתמקדת בחישוב השונות של ה- RTT (Round Trip Time), המהווה מדד לחשיבות התנודות בערכי ה-RTT. השונות מספקת מידע נוסף על חוסר יציבות ואי-ודאות בקצבי התגובה ברשת, ועוזרת להבחין בין תנודות סטנדרטיות לבין עיכובים בלתי צפויים.

הנוסחה היא:

node.rtt_var = (1 - BETA) * prev_node.rtt_var + BETA * abs(node.rtt_smooth - node.sample_rtt)

כאשר BETA הוא מקדם השונות, השולט במשקל של הפער בין הערך המוחלק לערך המדגמי. ערך זה נע בין 0 ל-1, כאשר ערכים נמוכים יותר נותנים משקל רב יותר לשונות הקודמת.

רעיון החישוב:

איזון בין נתונים היסטוריים לבין הפערים הנוכחיים: הנוסחה מחשבת את השונות של ה-RTT בצורה שמאזנת בין השונות הקודמת לבין הפער בין ה-RTT המוחלק לערך המדגמי הנוכחי.

איתור תנודות בלתי צפויות: השימוש במקדם β מאפשר לנוסחה להגיב לשינויים חדים ב-RTT ולהתאים את השונות בהתאם לתנודות החדשות.

הגדלת הדיוק בבקרת עומס: חישוב השונות מסייע לפרוטוקול QUIC להבין עד כמה הערכים המוחלקים יכולים להיות בלתי צפויים, ובכך לאפשר לבקרת העומס לבצע התאמות נדרשות בקצב השליחה ובגודל החלון.

5. **פונקציית print_rtt_values:** מדפיס את ערכי ה RTT שחישבנו באמצעות הפונקציות שפירטנו עליהן.

6. **פונקציית Create_graph:** יוצר גרף אשר ממחיש וויזואלית את הנתונים שהתקבלו ומציג את ההבדלים בין הRTT הנמדד לעומת המוערך (כפי שהתבקשנו). ציר הX יהיה הזמן בשניות, וציר הY יהיה מדד הRTT במילישניות.

אלגוריתם NewReno והפונקציה הממשת:

אלגוריתם New Reno הוא אחד האלגוריתמים הנפוצים לשליטת עומס בפרוטוקול TCP, וכיום הוא מותאם גם לשימוש בפרוטוקולים כמו QUIC. המטרה העיקרית של האלגוריתם היא לווסת את קצב השליחה ברשת על ידי התאמת גודל חלון הקונגרסיה (cwnd) כדי למנוע עומס יתר ולשמור על יציבות ויעילות התקשורת.

אלגוריתם New Reno מבוסס על גישת חלון משתנה אשר גדל ומתכווץ בהתאם להתנהגות הרשת:

- Slow Start (התחלה איטית): מצב זה הוא שלב הפתיחה של האלגוריתם, בו ה-cwnd גדל בצורה אקספוננציאלית. המטרה היא לגלות את קיבולת הרשת בצורה מהירה, אך זה גם מגביר את הסיכון לעומס יתר.
- Congestion Avoidance (מניעת עומס): מצב זה נכנס לפעולה כאשר ה-cwnd חוצה את הסף (ssthresh). כאן ה-cwnd גדל בצורה מתונה ומבוססת על גישת AIMD (Additive Increase Multiplicative Decrease), כלומר ה-cwnd גדל בצורה לינארית עם כל אישור (ACK), ומתכווץ משמעותית בעת איבוד מנות.
- Recovery (שחזור): מצב זה נכנס לתוקף כאשר מתגלים איבודי מנות. כאן ה-cwnd מתכווץ למחצית מגודלו ונעשית התאמה של הסף ssthresh לערך המוקטן של ה-cwnd.
- Persistent Congestion (עומס מתמשך): כאשר מתגלה עומס מתמשך, ה-cwnd מצטמצם למינימום האפשרי והאלגוריתם חוזר למצב Slow Start כדי לנסות לחדש את התקשורת בצורה מבוקרת. מצב זה מזוהה כאשר יש איבוד רצוף של מנות בין שני אישורי ACK או כאשר אין אף אישור על מנות שנשלחו במהלך פרק זמן מסוים. אם ההפרש בין זמני השליחה של שתי מנות שלא קיבלו אישור גדול מזמן העומס המתמשך המחושב, (Persistent Congestion Duration) המצב מוכרז כעומס מתמשך.

** **סף ה (Slow Start Threshold) ssthresh -** הוא פרמטר המגדיר את הנקודה בה מתבצע מעבר ממצב Slow Start למצב Congestion Avoidance (מניעת עומס). זהו ערך גבול, וכאשר גודל חלון העומס (cwnd) מגיע או עובר את סף זה, האלגוריתם משנה את אופן גידולו מגידול מעריכי לגידול לינארי, על מנת למנוע עומס יתר ברשת.

אופן המימוש של האלגוריתם:

הפונקציה new_reno_congestion_control מבצעת את השלבים של New Reno ומעדכנת את מצבי השליטה על העומס בהתאם להתנהגות הרשת:

- מצב Slow Start: האלגוריתם מתחיל במצב זה כאשר cwnd גדל בצורה אקספוננציאלית על פי גודל החבילה של כל ACK שהתקבל. במצב זה, אם cwnd מגיע לערך של ssthresh, מתבצע מעבר למצב Congestion Avoidance. אם מתרחשת איבוד מנות (packet_lost), ssthresh מחושב מחדש כערך מחצית מגודל ה-cwnd הנוכחי, והמצב מתעדכן ל-Recovery.

- מצב Recovery : במצב זה ה-cwnd מתכווץ למחצית מערכו הנוכחי או למינימום המותר (MIN_CWND). אם מנות מתקבלות בהצלחה (packet_acknowledged), המצב עובר ל-Congestion Avoidance.
- מצב Congestion Avoidance : במצב זה ה-cwnd גדל בצורה איטית יותר, בהתאם לגישת AIMD. כל ACK גורם לעלייה קטנה בגודל החלון. במקרה של איבוד מנות, מתבצע מעבר למצב Recovery.
- זיהוי עומס מתמשך : אם מתגלה עומס מתמשך, האלגוריתם מקטין את ה-cwnd למינימום האפשרי ומחזיר את המצב ל-Slow Start, מתוך מטרה לחדש את התקשורת.

פונקציית packet_lost: נועדה לזהות אם חבילה מסוימת נחשבת כאבודה. זיהוי מדויק של איבוד חבילות קריטי לתפקוד תקין של אלגוריתם בקרת העומס, כיוון שאיבוד חבילות בדרך כלל מצביע על גודש ברשת. הפונקציה בודקת שני תנאים : אם ה-Sample RTT (הזמן שנמדד בפועל) גבוה מה-Smoothed RTT (הממוצע המשוקלל) בתוספת פי 4 של סטיית התקן (RTT Variance). תנאי זה מזהה מצבים שבהם ייתכן שהחבילה התעכבה יתר על המידה. אם experiment_lost_flag מוגדר כ-True, שמשמעותו שהחבילה סומנה כאבודה בניסוי (לצורך הניסוי שהתבקשנו לבצע).

פונקציית packet_acknowledged: מטרתה לקבוע אם חבילה התקבלה ואושרה בהצלחה. התייחסות זו חשובה לצורך מעבר בין מצבים כמו Recovery ל Congestion Avoidance באלגוריתם, לאחר שחבילה נחשבת כמאושרת. הפונקציה מחזירה True אם ה-Sample RTT נמוך מה-Smoothed RTT, כלומר אם זמן הגעת האישור (ACK) לחבילה היה מהיר מהרגיל. תנאי זה מצביע על כך שהחבילה נשלחה והתקבלה ללא עיכובים משמעותיים, ולכן ניתן להחשיב אותה כמאושרת. נשים לב כי כבר בדקנו במימוש לגבי כל חבילה אם מס' החבילה של הACK תואם למס' חבילה שנשלחה, הפונקציה פועלת לאחר שקיבלנו נתונים על זמני השליחה.

פונקציית persistent_congestion_detected: נועדה לזהות אם קיים גודש מתמשך ברשת, מה שמצריך פעולה מיוחדת כמו צמצום משמעותי של גודל חלון הגודש כדי למנוע קריסת הרשת. תחילה הפונקציה מגדירה ערך של persistent_congestion_duration אם הוא עדיין לא הוגדר, באמצעות מכפלת PERSISTENT_CONGESTION_MULTIPLIER בממוצע ה-RTT המשוקלל וסטיית התקן של ה-RTT. ערך זה משמש כסף שממנו נקבע אם מצב הגודש נמשך לאורך זמן. לאחר מכן, הפונקציה בודקת אם ה-Sample RTT גדול מ-persistent_congestion_duration. אם כן, משמע שהחבילה חוותה גודש משמעותי לאורך זמן.

** פונקציית הסימולציה ופונקציית ה main נועדו לניסוי שהתבקשנו לבצע לגבי בקרת עומס, לכן נרחיב עליהן בחלק הניסויים.

חלק ב: פקודות הרצה והצגת תוצאות

סביבת עבודה : Linux , שפת כתיבה : python, סביבת פיתוח : PyCharm.

ע"מ להריץ את התוכנית וליצור חיבור בפרוטוקול QUIC יש לפעול לפי סדר הפעולות הבא :

- הרץ קודם את צד השרת ע"מ לפתוח תקשורת בפרוטוקול QUIC.
  הכנס פקודה זו לחלון הטרמינל : (ניתן להכניס פרמטר פורט אחר)

```
python3 Server.py -p 12347
```

- הרץ את צד הלקוח ע"מ ליצור קשר עם השרת ולהתחיל את התהליך.
  הכנס פקודה זו לחלון טרמינל שני : (ניתן להכניס פרמטרים ip, port, times אחרים)

```
python3 Client.py -ip 127.0.0.1 -p 12347 -t 2
```

- במידה ותרצה להריץ את הניסוי העוסק בבקרת עומס יש להריץ את קובץ
  LinkedList.py אשר בו מתרחש הניסוי שהתבקשנו לבצע.
  הכנס פקודה זו לחלון טרמינל שלישי :

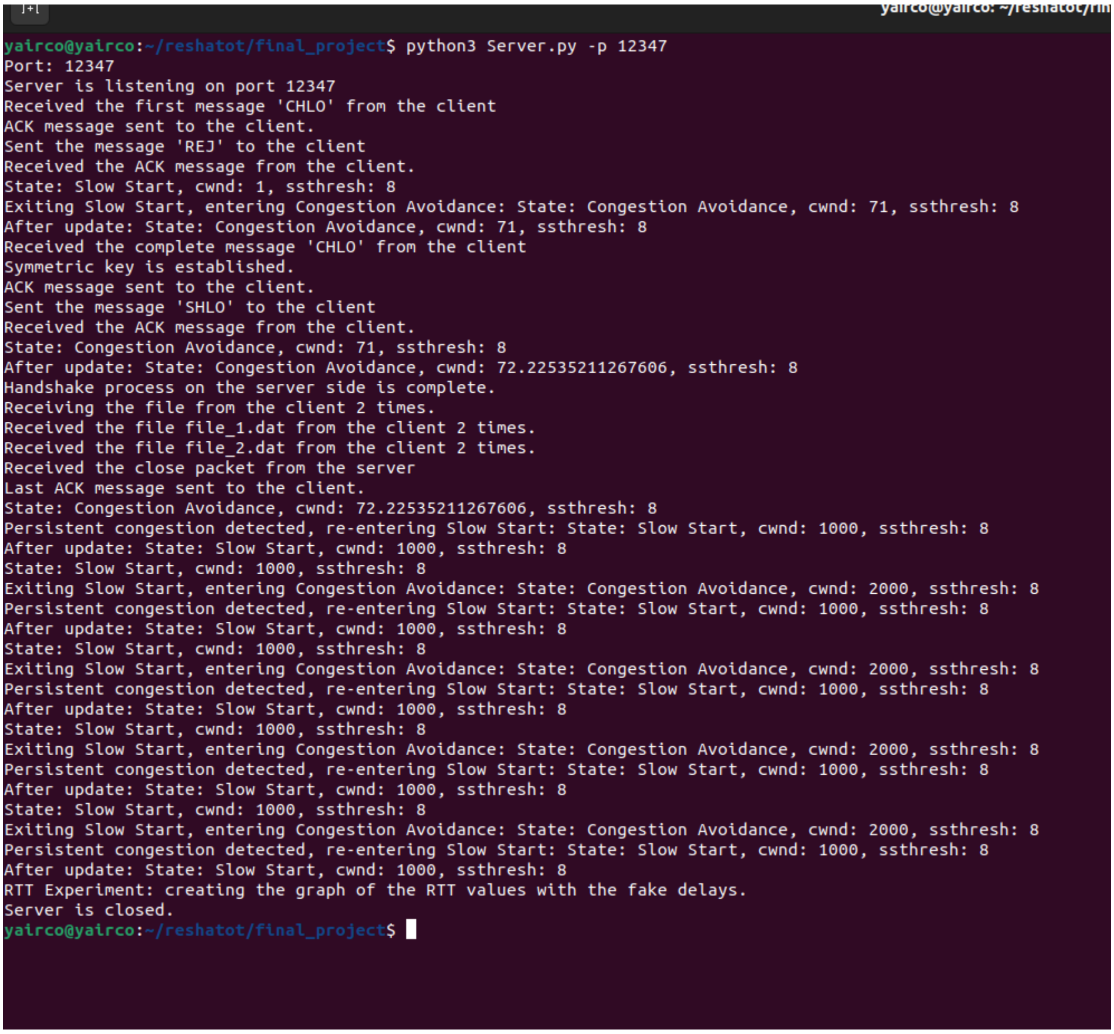
```
python3 LinkedList.py
```

- הרצת טסטים :

Python3 Test.py

- הצגת תוצאות (הצגת גרפי שני הניסויים יוצגו בחלק הניסויים):

צד השרת:



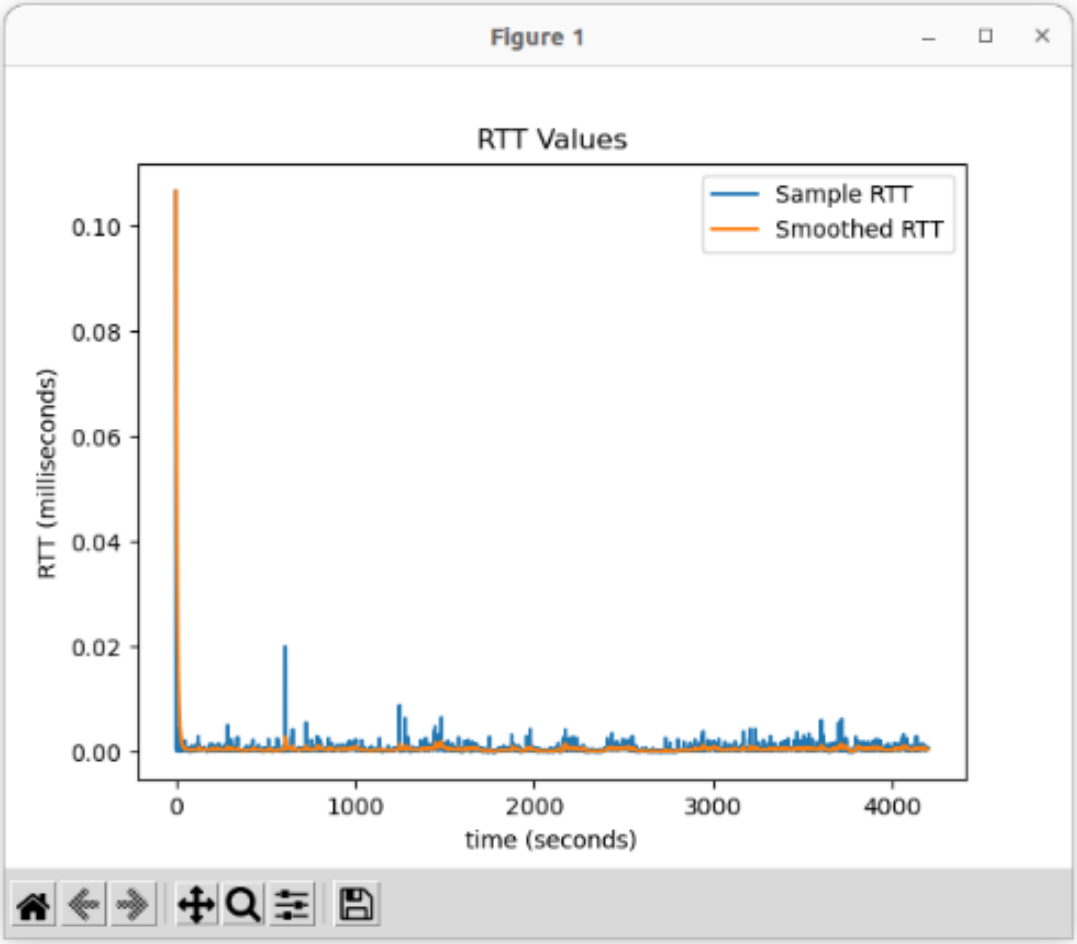
```
yairco@yairco:~/reshatot/final_project$ python3 Server.py -p 12347
Port: 12347
Server is listening on port 12347
Received the first message 'CHLO' from the client
ACK message sent to the client.
Sent the message 'REJ' to the client
Received the ACK message from the client.
State: Slow Start, cwnd: 1, ssthresh: 8
Exiting Slow Start, entering Congestion Avoidance: State: Congestion Avoidance, cwnd: 71, ssthresh: 8
After update: State: Congestion Avoidance, cwnd: 71, ssthresh: 8
Received the complete message 'CHLO' from the client
Symmetric key is established.
ACK message sent to the client.
Sent the message 'SHLO' to the client
Received the ACK message from the client.
State: Congestion Avoidance, cwnd: 71, ssthresh: 8
After update: State: Congestion Avoidance, cwnd: 72.22535211267606, ssthresh: 8
Handshake process on the server side is complete.
Receiving the file from the client 2 times.
Received the file file_1.dat from the client 2 times.
Received the file file_2.dat from the client 2 times.
Received the close packet from the server
Last ACK message sent to the client.
State: Congestion Avoidance, cwnd: 72.22535211267606, ssthresh: 8
Persistent congestion detected, re-entering Slow Start: State: Slow Start, cwnd: 1000, ssthresh: 8
After update: State: Slow Start, cwnd: 1000, ssthresh: 8
State: Slow Start, cwnd: 1000, ssthresh: 8
Exiting Slow Start, entering Congestion Avoidance: State: Congestion Avoidance, cwnd: 2000, ssthresh: 8
Persistent congestion detected, re-entering Slow Start: State: Slow Start, cwnd: 1000, ssthresh: 8
After update: State: Slow Start, cwnd: 1000, ssthresh: 8
State: Slow Start, cwnd: 1000, ssthresh: 8
Exiting Slow Start, entering Congestion Avoidance: State: Congestion Avoidance, cwnd: 2000, ssthresh: 8
Persistent congestion detected, re-entering Slow Start: State: Slow Start, cwnd: 1000, ssthresh: 8
After update: State: Slow Start, cwnd: 1000, ssthresh: 8
State: Slow Start, cwnd: 1000, ssthresh: 8
Exiting Slow Start, entering Congestion Avoidance: State: Congestion Avoidance, cwnd: 2000, ssthresh: 8
Persistent congestion detected, re-entering Slow Start: State: Slow Start, cwnd: 1000, ssthresh: 8
After update: State: Slow Start, cwnd: 1000, ssthresh: 8
State: Slow Start, cwnd: 1000, ssthresh: 8
Exiting Slow Start, entering Congestion Avoidance: State: Congestion Avoidance, cwnd: 2000, ssthresh: 8
Persistent congestion detected, re-entering Slow Start: State: Slow Start, cwnd: 1000, ssthresh: 8
After update: State: Slow Start, cwnd: 1000, ssthresh: 8
RTT Experiment: creating the graph of the RTT values with the fake delays.
Server is closed.
yairco@yairco:~/reshatot/final_project$ 
```

צד הלקוח:

```
yairco@yairco: ~/reshatot
yairco@yairco:~/reshatot/final_project$ python3 Client.py -ip 127.0.0.1 -p 12347 -t 2
Handshake process on the client side started.
Sent the first message 'CHLO' to the server
ACK message received from the server.
State: Slow Start, cwnd: 1, ssthresh: 8
Exiting Slow Start, entering Congestion Avoidance: State: Congestion Avoidance, cwnd: 79, ssthresh: 8
After update: State: Congestion Avoidance, cwnd: 79, ssthresh: 8
Received message 'REJ' from the server
ACK message sent to the server.
Sent the complete message 'CHLO' to the server
ACK message received from the server.
State: Congestion Avoidance, cwnd: 79, ssthresh: 8
After update: State: Congestion Avoidance, cwnd: 80.15189873417721, ssthresh: 8
Received message 'SHLO' from the server
Symmetric key established
ACK message sent to the server.
Handshake process on the client side is complete.
Sending the file to the server 2 times.
State: Congestion Avoidance, cwnd: 80.15189873417721, ssthresh: 8
After update: State: Congestion Avoidance, cwnd: 94.15031945432882, ssthresh: 8
State: Congestion Avoidance, cwnd: 94.15031945432882, ssthresh: 8
After update: State: Congestion Avoidance, cwnd: 106.06743249762836, ssthresh: 8
State: Congestion Avoidance, cwnd: 106.06743249762836, ssthresh: 8
After update: State: Congestion Avoidance, cwnd: 116.64560879151657, ssthresh: 8
State: Congestion Avoidance, cwnd: 116.64560879151657, ssthresh: 8
After update: State: Congestion Avoidance, cwnd: 126.26448781854772, ssthresh: 8
State: Congestion Avoidance, cwnd: 126.26448781854772, ssthresh: 8
After update: State: Congestion Avoidance, cwnd: 135.1505968059964, ssthresh: 8
State: Congestion Avoidance, cwnd: 135.1505968059964, ssthresh: 8
After update: State: Congestion Avoidance, cwnd: 143.45244693848667, ssthresh: 8
State: Congestion Avoidance, cwnd: 143.45244693848667, ssthresh: 8
After update: State: Congestion Avoidance, cwnd: 151.27385412912957, ssthresh: 8
State: Congestion Avoidance, cwnd: 151.27385412912957, ssthresh: 8
After update: State: Congestion Avoidance, cwnd: 158.6908661862313, ssthresh: 8
State: Congestion Avoidance, cwnd: 158.6908661862313, ssthresh: 8
After update: State: Congestion Avoidance, cwnd: 165.76121640212386, ssthresh: 8
State: Congestion Avoidance, cwnd: 165.76121640212386, ssthresh: 8
After update: State: Congestion Avoidance, cwnd: 172.52998912443616, ssthresh: 8
State: Congestion Avoidance, cwnd: 172.52998912443616, ssthresh: 8
After update: State: Congestion Avoidance, cwnd: 179.0332063662269, ssthresh: 8
State: Congestion Avoidance, cwnd: 179.0332063662269, ssthresh: 8
After update: State: Congestion Avoidance, cwnd: 185.30020019811334, ssthresh: 8
State: Congestion Avoidance, cwnd: 185.30020019811334, ssthresh: 8
After update: State: Congestion Avoidance, cwnd: 191.3552395278087, ssthresh: 8
State: Congestion Avoidance, cwnd: 191.3552395278087, ssthresh: 8
After update: State: Congestion Avoidance, cwnd: 197.21867970728152, ssthresh: 8
State: Congestion Avoidance, cwnd: 197.21867970728152, ssthresh: 8
After update: State: Congestion Avoidance, cwnd: 202.90779597996578, ssthresh: 8
State: Congestion Avoidance, cwnd: 202.90779597996578, ssthresh: 8
```

```
State: Congestion Avoidance, cwnd: 3067.041287200772, ssthresh: 8
After update: State: Congestion Avoidance, cwnd: 3067.407112077236, ssthresh: 8
State: Congestion Avoidance, cwnd: 3067.407112077236, ssthresh: 8
After update: State: Congestion Avoidance, cwnd: 3067.772893324721, ssthresh: 8
State: Congestion Avoidance, cwnd: 3067.772893324721, ssthresh: 8
After update: State: Congestion Avoidance, cwnd: 3068.138630958834, ssthresh: 8
State: Congestion Avoidance, cwnd: 3068.138630958834, ssthresh: 8
After update: State: Congestion Avoidance, cwnd: 3068.504324995172, ssthresh: 8
State: Congestion Avoidance, cwnd: 3068.504324995172, ssthresh: 8
After update: State: Congestion Avoidance, cwnd: 3068.869975449323, ssthresh: 8
State: Congestion Avoidance, cwnd: 3068.869975449323, ssthresh: 8
After update: State: Congestion Avoidance, cwnd: 3069.2355823368666, ssthresh: 8
State: Congestion Avoidance, cwnd: 3069.2355823368666, ssthresh: 8
After update: State: Congestion Avoidance, cwnd: 3069.601145673372, ssthresh: 8
State: Congestion Avoidance, cwnd: 3069.601145673372, ssthresh: 8
After update: State: Congestion Avoidance, cwnd: 3069.6904080846743, ssthresh: 8
File sent successfully to the server
File sent to the server 2 times.
Sent the close packet to the server
ACK message received from the server.
State: Congestion Avoidance, cwnd: 3069.6904080846743, ssthresh: 8
After update: State: Congestion Avoidance, cwnd: 3069.7106055612135, ssthresh: 8
Creating the graph of the RTT values.
Client Socket closed.
File deleted.
yairco@yairco:~/reshatot/final_project$
```

גרף מדידת RTT על חבילות שהלקוח שלח:



הסבר תוצאות צד השרת:

השרת מתחיל בהאזנה על הפורט שנבחר ומבצע את תהליך ה-Handshake עם הלקוח. במהלך תהליך ה-Handshake, השרת שולח ומקבל הודעות CHLO, REJ, ו-SHLO.

לאחר השלמת ה-Handshake, השרת נכנס למצב Congestion Avoidance (מניעת עומס) לאחר יציאה מ-Slow Start.

תהליך קבלת הקובץ מהלקוח מתבצע פעמיים בהצלחה, עם קבלת הודעות סגירה ושיגור הודעות ACK מתאימות.

במהלך התהליך נצפו אירועי Persistent Congestion, המובילים לכניסה חוזרת למצב Slow Start, וחזרה למצב Congestion Avoidance לאחר מכן.

בנוסף גרף של ערכי ה-RTT נוצר על בסיס נתוני ניסוי שעליו נרחיב בהמשך.

ועובר למצב Recovery כדי לאפשר התאוששות הדרגתית ולהמשיך לשלוט בעומס.

הסבר תוצאות צד הלקוח:

הלקוח מתחיל בתהליך Handshake עם השרת, הכולל שליחת הודעות CHLO וקבלת הודעות REJ ו-SHLO. לאחר השלמת ה-Handshake והקמת מפתח סימטרי מוצלח, הלקוח שולח את הקובץ לשרת פעמיים בהצלחה. ההודעות נשלחות ונקלטות בסדר הנכון, והלקוח מאמת את קבלת ה-ACK מהשרת לכל ההודעות.

ניתן לראות כי מודפס מצב האלגוריתם NewReno ועדכון המצב שבו החלון נמצא כאשר נשלחה חבילה חדשה והתקבלה הודעת ACK עליה.

לבסוף הלקוח מייצר גרף על נתוני הRTT.

ניתוח הגרף:

הגרף שמוצג מראה את ערכי ה-(Round-Trip Time) RTT עבור החבילות שנשלחו מהלקוח, ומציג השוואה בין Sample RTT ל-Smoothed RTT לאורך זמן.

- **Sample RTT (הקו הכחול):** ה-Sample RTT מייצג את הזמן שנמדד בפועל שנדרש לכל חבילה לעבור לשרת ולחזור ללקוח. זהו מדד ישיר לעיכוב ברשת בזמן שליחת כל חבילה.
  הגרף מראה שה-Sample RTT מתנודד עם קפיצות מזדמנות, המשקפות את השינויים המיידיים במצב הרשת, כמו גודש זמני, שינויים בתנאי הרשת או עיכובים שנגרמים בגלל שידורים חוזרים של חבילות.
- **Smoothed RTT (הקו הכתום):** ה-Smoothed RTT הוא ממוצע משוקלל של ערכי ה-Sample RTT האחרונים, שמחושב באמצעות ממוצע נע אקספוננציאלי. ערך זה פחות תנודתי ומספק אומדן יציב יותר של זמן החביון הממוצע ברשת לאורך זמן.
  בתחילה, ה-Smoothed RTT מתחיל בגובה עקב ייצוב ראשוני של הרשת, ואז מתכנס מהר לערך נמוך ויציב. הדבר מצביע על כך שהתנאים ברשת מתייצבים לאחר תקופת תנודות ראשונית.

נשם לב כי ישנה פסגה בולטת בתחילת הגרף, שבה גם ה-Sample RTT וגם ה-Smoothed RTT גבוהים. מצב זה שכיח בשלבים הראשונים של חיבור כאשר הרשת מתאימה את עצמה לזרם התעבורה החדש. לאחר הפסגה הראשונית, הערכים יורדים משמעותית ומתייצבים, מה שמרמז על כך שמנגנון בקרת הגודש (כמו NewReno) מנהל את הזרימה של החבילות ביעילות.

לאורך הגרף, ה-Sample RTT מציג תנודות קלות סביב ה-Smoothed RTT. תנודות אלו צפויות ברשתות אמיתיות עקב שינויים קלים בגודש או במסלולי הניתוב.

מסקנות:

- יעילות בקרת הגודש: הגרף מדגים שמנגנון בקרת הגודש מצליח לייצב את ה-RTT במהירות, תוך צמצום השינויים בזמן ההעברה של החבילות.
- חביון נמוך ויציבות: הערכים הנמוכים של ה-RTT מרמזים על רשת יציבה ויעילה עם מעט גודש ברוב הזמן, דבר המועיל לשמירה על תפוקה גבוהה וחביון נמוך.

- התמודדות עם שינויים ברשת: ה-Smoothed RTT מסנן בהצלחה את רוב התנודות הקצרות ב-Sample RTT, מה שמעיד על כך שהמערכת יכולה להתמודד עם בעיות רשת זמניות מבלי לפגוע בביצועים בצורה דרסטית.

בסך הכול, הנתונים מראים כי תנאי הרשת נותרו נוחים להעברת נתונים במהירות גבוהה לאחר תקופת ההתאמה הראשונית. זה יתרון ביישומים הדורשים תקשורת עקבית ובעלת חביון נמוך, כמו סטרימינג וידאו או משחקים מקוונים.

חלק ג: הקלטות Wireshark

הגדרנו את ה port להיות 1234567 ולכן נסנן לפיו.

נשים לב כי בקוד שכתבנו החבילות הינן מוצפנות בעקבות עקרון ההצפנה של QUIC הקורה ביחד עם תהליך הHandshake ולכן המידע שנקבל מהWIRESHARK על החבילות שהועברו הוא מוצפן. נזהה בכל זאת את סוג ההודעות ע״פ הlength של החבילות שהגדרנו מראש בקוד .

תהליך פתיחת קשר Handshake :

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	QUIC	120	59260 → 12347 Len=78[Malformed Packet]
2	0.066484419	127.0.0.1	127.0.0.1	QUIC	107	Protected Payload (KP0)
3	0.067698242	127.0.0.1	127.0.0.1	QUIC	112	12347 → 59260 Len=70[Malformed Packet]
4	0.067948097	127.0.0.1	127.0.0.1	QUIC	107	Protected Payload (KP0)
5	0.067998905	127.0.0.1	127.0.0.1	QUIC	133	0-RTT, DCID=08c749595598b425
6	0.068327765	127.0.0.1	127.0.0.1	QUIC	107	Protected Payload (KP0)
7	0.068840039	127.0.0.1	127.0.0.1	QUIC	129	Retry[Malformed Packet]
8	0.069207982	127.0.0.1	127.0.0.1	QUIC	107	Protected Payload (KP0)
9	0.069842252	127.0.0.1	127.0.0.1	QUIC	1164	Protected Payload (KP0)

כמו שהסברנו תהליך ה handshake הממומש אצלנו פועל כך :

חבילה מס׳ 1 הלקוח שולח הודעת CHLO ראשונה ללא DEID לשרת ע״מ לבצע בקשה לפתיחת קשר ללא נתונים נוספים, חבילת Long header שגודלה 120.

חבילה מס׳ 2 השרת שולח הודעת ACK ראשונה ללקוח ע״מ לאשר שקיבל הודעה. ניתן לראות לאורך כל ההקלטה כי גודל הודעת ACK היא 107.

חבילה מס׳ 3 השרת שולח הודעת REJ ללקוח ע״מ להעביר לו פרטים שהלקוח צריך להעביר ומעביר את ה CID שלו חבילת Long header שגודלה 112.

חבילה מס׳ 4 הלקוח שולח ACK.

חבילה מס׳ 5 הלוקח שולח הודעת CHLO שנייה שבה מעביר את המפתח הציבורי שלו לשרת ע״מ יצירת מפתח סימטרי משותף להצפנת ההודעות.

חבילה מס׳ 6 השרת שולח ACK.

חבילה מס׳ 7 השרת שולח הודעת SHLO שבה בעצם מסתיימת תהליך ה HANDSHAKE ושולח ללקוח את המפתח הציבורי של השרת לשם אותה מטרה.

חבילה מס׳ 8 הלקוח שולח ACK.

מחבילה מס׳ 9 ניתן לראות שגודל החבילות שנשלחות הוא 1164 ו107 זאת מכיוון שהלקוח מתחיל לשלוח את הקובץ לשרת ע״פ הchunk_size שהגדרנו בsend_file כאשר כל חבילה מחולקת ל5 פריימים ובכך עקרון פרוטקול QUIC בא לידי ביטוי. לאחר כל שליחה מתבצע קבלת ACK. תהליך זה נמדד ע״פ ה RTT ומתבצע באמצעות אלגוריתם בקרת העומס NewReno אשר מחזיק משתנה זמן ssthresh שחלון העומס משנה מצבים ביחס למשתנה זה.

8396	4.460674811	127.0.0.1	127.0.0.1	QUIC	107	Protected Payload (KP0)
8397	4.461589086	127.0.0.1	127.0.0.1	QUIC	1164	Protected Payload (KP0)
8398	4.462995814	127.0.0.1	127.0.0.1	QUIC	107	Protected Payload (KP0)
8399	4.463864864	127.0.0.1	127.0.0.1	QUIC	316	Protected Payload (KP0)
8400	4.464098807	127.0.0.1	127.0.0.1	QUIC	107	Protected Payload (KP0)
8401	4.465020619	127.0.0.1	127.0.0.1	QUIC	104	Protected Payload (KP0)
8402	4.465217991	127.0.0.1	127.0.0.1	QUIC	107	Protected Payload (KP0)

לאחר שליחת הdata האחרונה שנשארה מהקובץ ב8399 frame , השרת שולח הודעת ACK.

לאחר מכן ניתן לראות שהלקוח שולח הודעת CLOSE בפריים מס׳ 8401 והשרת מחזיר לו בACK. בכך נסגרים הסוקטים של כל צד והסתיים החיבור בין השרת ללקוח.

חלק ד': ניסויים

ניסוי מדידת RTT:

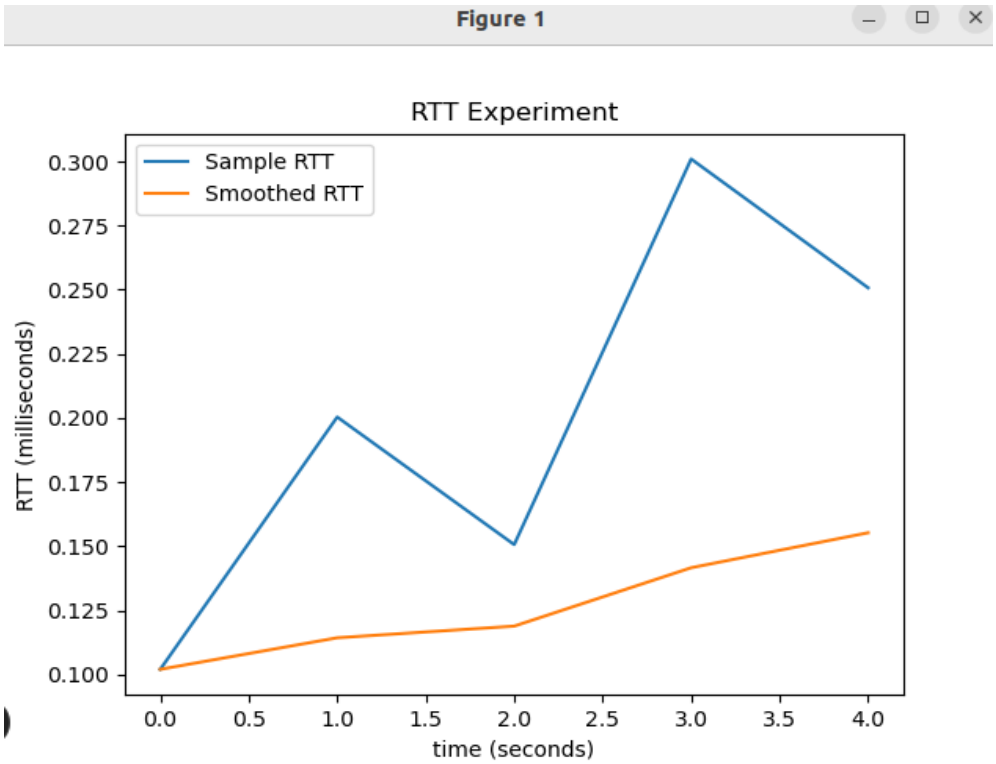
- **נושא הניסוי -**

ניסוי זה נועד לבחון את השפעת זמני עיכוב (RTT - Round-Trip Time) מלאכותיים על הביצועים והתגובה של פרוטוקול QUIC. מטרת הניסוי הייתה לדמות תנאי רשת מגוונים על ידי הוספת עיכובים משתנים בין שליחת החבילות לבין קבלת ה-ACK (אישור קבלה) ולהעריך כיצד משתנה ה-RTT הממוצע והחלק (Smoothed RTT) בתנאים אלו.
הניסוי מתבצע בצד השרת לאחר שהלקוח סיים לשלוח את הקבצים.

- **מימוש הניסוי -**

במימוש הניסוי נוצרה רשימת חבילות (PacketList) המשמשת לעקוב אחר זמני השליחה והקבלה של החבילות. לשם הניסוי יצרנו פונקציה בשם simulate_network_conditions המשתמשת בעיכובים מלאכותיים שנקבעו מראש (0.1, 0.2, 0.15, 0.3, ו-0.25 שניות) כדי לדמות את תנאי הרשת. כל חבילה נשלחה עם עיכוב שנקבע באמצעות פקודת Sleep עם ערכי delay הנתונים במערך ולאחר מכן התקבלה ACK. בכל מחזור מתווספת לרשימה חבילה חדשה עם זמן שליחה וזמן קבלה מחושבים.
לאחר הדמיית התנאים, נוצר גרף המציג את ה-RTT בפועל של כל חבילה (Sample RTT) לעומת ה-RTT החלק (Smoothed RTT) שמחושב לפי נוסחא מבוססת משקל מתגלגל, שבו לכל דגימה יש השפעה מסוימת על החישוב של RTT החלק.

- **הגרף המתקבל -**



- **ניתוח צירי הגרף -**

ציר X - זמן (seconds): ציר זה מייצג את הזמן החולף בזמן הניסוי, כאשר כל נקודה על הציר מסמלת זמן שבו נמדד ערך RTT עבור חבילה שנשלחה והתקבל ACK עבורה. הציר מראה את סדר הזמנים לפי חבילות כפי שהן נשלחו במהלך הניסוי.

ציר Y - RTT (milliseconds): ציר זה מציג את ערכי ה-RTT במילי-שניות עבור כל מדידה שנלקחה. ה-RTT מתאר את הזמן הכולל שלקח לחבילה להגיע ליעדה ולקבל ACK חזרה, והוא מחולק לשתי עקומות:

Sample RTT (RTT דגימה): מייצג את זמן ה-RTT האמיתי שנמדד עבור כל חבילה שנשלחה. הקו כחול מציג את השינויים החדים והמהירים בזמני ה-RTT בין חבילה לחבילה, בעיקר כתוצאה מהעיכובים השונים שהוזרקו בכל שלב של הניסוי.

Smoothed RTT (RTT חלק): הקו הכתום מציג את ה-RTT הממוצע החלק שמחושב בעזרת אלגוריתם משקל מתגלגל. אפשר לראות שהוא עוקב אחרי ה-Sample RTT אבל בצורה מתונה וחלקה יותר, עם פחות תנודות חדות. עקומה זו נועדה להציג מגמות כלליות תוך דיכוי תנודות חדות.

- **תוצאות ומסקנות מן הגרף -**

תנודות ב-Sample RTT: ניתן לראות שה-Sample RTT משתנה בצורה חדה בין חבילות, במיוחד בשל תנאי רשת משתנים כמו עיכובים מלאכותיים שהוזרקו. זה מראה רגישות של ה-RTT לתנאים בזמן אמת שיכולים להשתנות במהירות.

יציבות ה-Smoothed RTT: ה-Smoothed RTT, לעומת זאת, מראה עלייה מתונה ומתמשכת ללא התנודות החדות. החישוב של ה-RTT החלק מסייע להקטין את ההשפעה של תנודות חדות ולהציג תמונה חלקה ויציבה יותר של המצב הכללי ברשת.

ניתן לראות שה-Smoothed RTT תמיד קטן או קרוב לערך ה-Sample RTT, במיוחד כאשר ה-Sample RTT חווה קפיצות חדות. משמעות הדבר היא שהחישוב של ה-RTT החלק מייצר תגובה מדורגת יותר לשינויים, מה שעוזר בשמירה על יציבות התנהגות הפרוטוקול.

שיפור יציבות בקרת עומס: היכולת לנטרל את התנודות הרגעיות ב-RTT מאפשרת לפרוטוקול להגיב בצורה שקולה ויציבה יותר לשינויים, מבלי לצמצם את חלון הגודש בצורה חפוזה. זהו יתרון משמעותי בניהול עומסים ברשתות עם תנאים משתנים.

ניהול אופטימלי של רוחב פס: באמצעות ה-Smoothed RTT, פרוטוקול QUIC מסוגל להקצות רוחב פס בצורה מושכלת יותר, תוך הימנעות מהקטנות חלון תכופות שעלולות

לפגוע בביצועים. שמירה על RTT יציב מונעת תגובות קיצוניות של הפרוטוקול כמו כניסה למצב של Slow Start בצורה בלתי מוצדקת, ובכך משפרת את היעילות הכללית של התקשורת.

לסיכום, הניסוי ממחיש כיצד חישוב RTT חלק מאפשר לבחון מגמות ולהבין טוב יותר את אופי התנהגות הרשת לאורך זמן, דבר שחשוב במיוחד בעת אבחון תקלות או ביצוע אופטימיזציה של רשתות תקשורת.

ניסוי בקרת עומס:

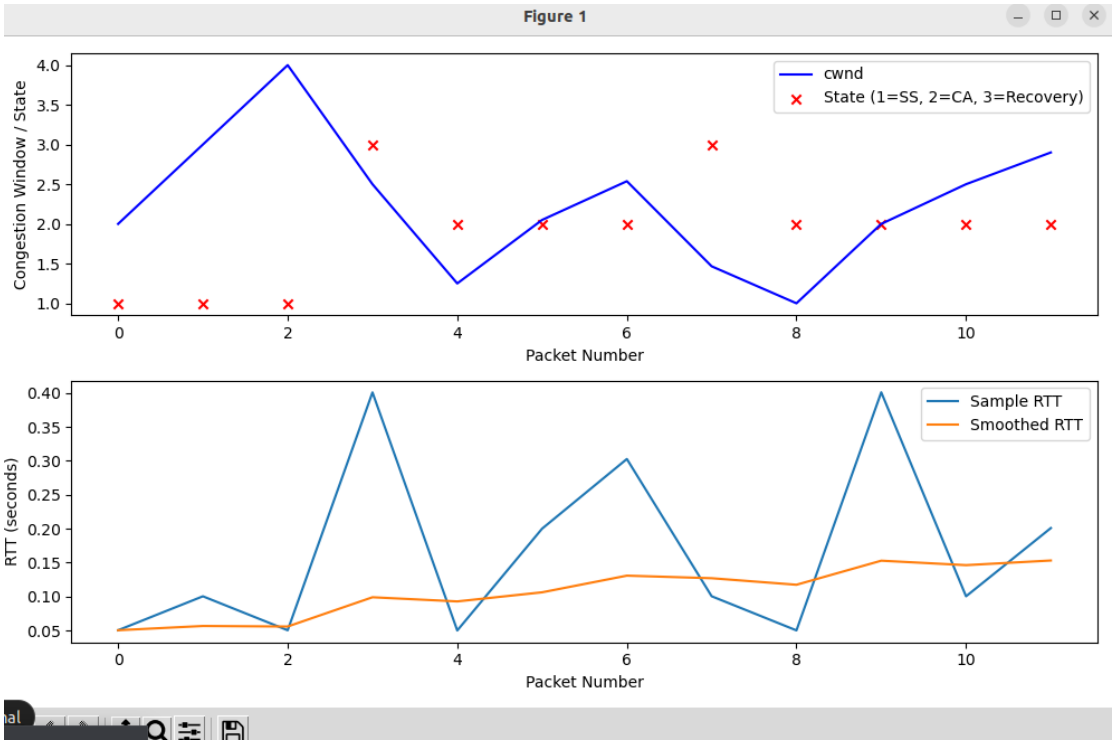
- **נושא הניסוי –**

בניסוי זה נבדקה התנהגות אלגוריתם בקרת העומס NewReno במצבים של איבוד חבילות ועיכוב זמנים. מטרת הניסוי היא להבין כיצד האלגוריתם מתמודד עם מצבים אלה ואיך שינויי המצב שלו משפיעים על גודל חלון העומס (cwnd) וזמני ה-RTT.
שים לב ע״מ להריץ ניסוי זה יש להריץ את קובץ LinkedList.py

- **מימוש הניסוי –**

בשביל להגשים את מטרת הניסוי, המימוש מבוצע באמצעות הפונקציה simulate_packet_transmission, שמדמה שליחת חבילות בתנאי רשת מוגדרים מראש:
הגדרת עיכובים ואובדני חבילות: רשימת עיכובים delays כוללת את זמני השהייה בכל שליחת חבילה, והמשתנה losses מגדיר אילו חבילות נחשבות לאבודות בניסוי.
תהליך השליחה והקבלה: כל חבילה נשלחת עם זמן שליחה שנרשם לפני ההשהייה (send_time) וזמן קבלה (ack_time) לאחר ההשהייה. אם החבילה מוגדרת כאבודה (experiment_lost_flag), היא מסומנת כך, ואלגוריתם ה-NewReno מגיב בהתאם למצב של אובדן.
עדכון מצבי האלגוריתם: בכל שלב, פונקציית ה-new_reno_congestion_control מופעלת כדי לעדכן את מצב האלגוריתם בהתאם לאירועים (כגון אישור חבילה או אובדן חבילה) ולבצע התאמה של חלון העומס (cwnd).

- **הגרף המתקבל -**



- **ניתוח צירי הגרפים –**

הגרף מתחלק לשני חלקים:

גרף עליון - חלון העומס ומצבי האלגוריתם:

1. ציר X: מייצג את מספרי החבילות שנשלחו.
2. ציר Y: מייצג את גודל חלון העומס (cwnd) (קו כחול) ואת מצב האלגוריתם (נקודות אדומות). כל נקודה אדומה מסמנת את מצב האלגוריתם שבו נמצא NewReno:

- Slow Start: מצב שבו חלון העומס גדל במהירות רבה כדי לנצל את רוחב הפס עד שהאלגוריתם מזהה את מגבלות הרשת.
- Congestion Avoidance: מצב שבו הגדלת חלון העומס נעשית בצורה איטית יותר, תוך שמירה על יציבות התקשורת.
- Recovery: מצב של התאוששות שמופעל בעקבות איבוד חבילה. במצב זה חלון העומס מצטמצם בצורה דרמטית ולאחר מכן מתחיל לעלות בהדרגה.

ניתן לראות בגרף ששני מצבים של איבוד חבילות (חבילות 3 ו-7) גורמים לכניסה למצב Recovery, שבו חלון העומס מצטמצם, והאלגוריתם עובר למצב של התאוששות.

גרף תחתון - זמני RTT:

1. ציר X: מייצג את מספרי החבילות.
2. ציר Y: מייצג את זמני RTT, עם עקומות שמייצגות את:

- Sample RTT (קו כחול): זמן החביון שנמדד בפועל עבור כל חבילה.
- Smoothed RTT (קו כתום): ממוצע נע של זמני ה-RTT שמחשב את זמן החביון בצורה חלקה יותר כדי למנוע רעשים חדים.

ניתן לראות ש-Sample RTT מציג תנודות חדות במיוחד בעת עיכובים מוגברים, בעוד ש-Smoothed RTT מייצג בצורה יותר מתונה את השינויים ומשקף את היציבות של הקשר בין חבילות.

- **תוצאות מן הגרף –**

האלגוריתם מתחיל במצב Slow Start, בו חלון העומס (cwnd) גדל באופן אקספוננציאלי עד שהוא מגיע לסף שהוגדר מראש (ssthresh). כל עוד אין איבוד חבילה, האלגוריתם ממשיך להגדיל את cwnd. עם זאת, בנקודה מסוימת (כפי שנראה בניסוי בחבילה מספר 3), התרחש איבוד חבילה והאלגוריתם נכנס למצב Recovery בו חלון העומס מצטמצם באופן דרסטי והאלגוריתם מתמקד בהחזרת הקשר למצב יציב. עם סיום ההתאוששות, האלגוריתם נכנס למצב Congestion Avoidance, שבו ההגדלה של חלון העומס מתבצעת בקצב איטי יותר, בהתאם לגישת AIMD.

הגרף העליון מציג את השינויים בחלון העומס ואת המצבים של NewReno לאורך הניסוי. ניתן לראות שהמעברים בין המצבים מתבצעים באופן תדיר בתגובה לאירועים שונים כמו אישור חבילה, אובדן, והתאוששות. חבילות 3 ו-7 אבדו בהתאם להגדרת הניסוי. ניתן לראות שינוי משמעותי במצב האלגוריתם בגרף העליון כאשר חבילות אלו מסומנות במצב Recovery. בכל מעבר למצב Recovery, חלון העומס cwnd מצטמצם כמחציתו, ומתחיל לעלות שוב כאשר חבילות מקבלות אישור תקין.

הגרף התחתון מציג את ערכי ה-RTT (Round-Trip Time). ערכי ה-Sample RTT נעים בין 0.05 ל-0.35 שניות ומציגים תנודות חדות בהתאם לעיכובים המתרחשים ברשת. ערכי ה-Smoothed RTT, לעומת זאת, מציגים תמונה חלקה יותר, אשר מסננת את השינויים החדים בזמנים ומייצגת טוב יותר את המגמות הכלליות של הקשר.

- **מסקנות –**

אלגוריתם NewReno מתמודד בצורה יעילה עם אובדן חבילות ועיכובים, תוך שמירה על יציבות על ידי התאמת חלון העומס במצבים שונים. המצב Recovery מסייע למנוע עומס יתר כאשר מתגלה איבוד חבילה, והמעבר למצב Congestion Avoidance מאפשר חזרה להגדלת קצב השידור בהדרגה.

החישוב של Smoothed RTT מאפשר לעקוב בצורה יציבה יותר אחר זמני החביון של הרשת ולמנוע תגובות יתר לקפיצות חדות ב-Sample RTT.
בניסוי ניתן לראות את החשיבות של בקרת העומס במצבי רשת דינמיים. אלגוריתם NewReno מספק מנגנונים חשובים להתמודדות עם מצבי עומס ברשתות תקשורת, ובכך תורם ליציבות ושיפור בביצועי השידור.
השימוש ב-Smoothed RTT מאפשר לזהות מצבי עומס מתמשכים ולהגיב בצורה מתאימה, דבר שמקטין את השפעת התנודות הקיצוניות בזמני ה-RTT המתקבלים.

חלק ה': ביבליוגרפיה

1. המאמר אשר סופק בקובץ המטלה:
https://web.cs.ucla.edu/~lixia/papers/UnderstandQUIC.pdf

2. הרכבת חבילות QUIC ותמונות מתהליך הHANDSHAKE :

https://www.cse.wustl.edu/~jain/cse570-21/ftp/quic/index.html