QUIC PROTOCOL

פרויקט רשתות תקשורת - ריבוי זרימות

מגישים: דניאל קוריס 214539397 , שירה יוגב 325877108 , אורין גבריאל 318774338 , אור גורן 314619115.

<u>עבדנו בסביבת עבודה Windows</u>

קצת על הפרוטוקול:

פרוטוקול QUIC הוא פרוטוקול רשת מהיר ובטוח המבוסס על UDP ומשמש להעברת נתונים בין צד לקוח ושרת באופן מהיר ובטוח. הפרוטוקול משלב את היתרונות של TCP ו UDP- ומספק עמידות בפני אובדן חבילות, מנגנון זרימה רקעי, פחות עקביות רשתית ותגובה מהירה יותר לפניות.

בקצרה על הקוד:

הקוד מאפשר שליחת קבצים גדולים בצורה מהירה ויעילה על רשת. התהליך מתחיל עם יצירת קובץ אקראי בין 1 MB ל-2 MB ובחירת מספר זרימות מ 1 עד 10 לשליחת הקובץ (בסדר הזה על מנת ליצור גרף שלם). בכל זרימה נשלח הקובץ. הקובץ נשלח בחלקים. כאשר הקליינט מוכן לשלוח קובץ חדש, הוא ממתין לאישור מהשרת ורק לאחר קבלת האישור ממשיך לשלוח. לסיום, לאחר השלמת המשימה, הקוד מנקה את הקבצים שנוצרו במהלך השליחה. באמצעות התהליך הזה, הקוד מאפשר למשתמש לשלוח ולקבל קבצים בצורה מסודרת ובטוחה.

server

הקבועים של השרת:

- **local_port** - הפורט המקומי שבו השרת מקבל נתונים.
- **packet_size** - גודל הפאקטה בבתים.
- **ready_message** – השרת שולח הודעת READY ללקוח כדי להודיע שהוא מוכן להתחיל איטרציה חדשה.
- **finished_message** - השרת שולח הודעת FINISHED ללקוח כדי להודיע שהוא סיים לקבל את כל הנתונים.

- **timeout_duration-** זמן ההמתנה לקבלת נתונים.
- - **num_iterations-** מספר האיטרציות שהשרת יריץ

אתחול השרת:

1. **יצירת אובייקט סוקט**:**socket-**
   - יוצר סוקט UDP
2. **קשירת הסוקט לפורט מקומי** :**socket.bind(('', self.local_port))-**
   - קושר את הסוקט לפורט מקומי מסוים.
3. **משתנה לאחסון נתוני הזרם** :**stream_data.-**
   - יוצר מילון לאחסון נתוני זרמים.
4. **ספירת הבייטים שהתקבלו** :**total_bytes_received-**
   - יוצר משתנה לעקוב אחר המספר הכולל של בייטים שהתקבלו.
5. **ספירת המנות (packets) שהתקבלו** :**total_packets_received** -
   - יוצר משתנה לעקוב אחר המספר הכולל של פאקטות שהתקבלו.
6. **זמן התחלה של קבלת נתונים** :**start_time** -
   - יוצר מילון לשמירת זמן התחלת קבלת הנתונים לכל זרם.
7. **זמן סיום של קבלת נתונים** :**end_time-**
   - יוצר מילון לשמירת זמן סיום קבלת הנתונים לכל זרם.
8. **מספר האיטרציה הנוכחית** :**iteration-**
   - יוצר משתנה לעקוב אחר מספר האיטרציה הנוכחית.
9. **מדדים לאיטרציה** :**iteration_metrics-**
   - יוצר רשימה לאחסון מדדים עבור כל איטרציה.
10. **הגדרת זמן התפוגה של הסוקט** :**socket.settimeout(self.timeout_duration)-**

- מגדיר את זמן התפוגה של הסוקט, כלומר כמה זמן לחכות לפני שיגמר הזמן במידה ואין נתונים המתקבלים.

פונקציות- שרת:

1. **close_connection:** סוגרת את החיבור שנפתח על ידי השרת באמצעות הקריאה ל־close() על הסוקט של השרת.

2. **process_initial_message:** המטרה של הפונקציה הזו היא לעבד את ההודעה הראשונית שנשלחה מהלקוח ולהשיב פרטי התחברות. כאשר הלקוח מתחיל את התקשורת עם השרת, הוא שולח הודעה שמכילה את פרטי התקשורת הבסיסיים, ipה של הלקוח, מספר הזרמים וגודל הקובץ הנשלח. הפונקציה משתמשת ב־recvfrom כדי לקבל את ההודעה מהלקוח, ומשתמשת ב־struct.unpack כדי להפוך את הנתונים שנשלחו מהלקוח לפרמטרים נוחים לשימוש. כשהפונקציה מקבלת את הנתונים האלו, היא מחזירה אותם בתור ערכים נפרדים, כך שניתן יהיה להשתמש בהם בתהליך התקשורת המתמשך.

3. **update_statistics**: פונקציה זו מתקדמת בתהליך הקבלת הנתונים מהלקוח ומעדכנת את הסטטיסטיקות בהתאם. כאשר השרת מקבל נתונים מהלקוח, הוא רוצה לעקוב אחרי ההתקדמות של הקשר ולסכום את הנתונים שהוא מקבל. פונקציה זו מתעדכנת בכמות הנתונים שהשרת קיבל, ומוסיפה אותם לתוך מבני הנתונים המתאימים.

- ממוצע זמן בין קבלת חבילות- כמה זמן עובר בין קבלת חבילה אחת לחבילה הבאה זה יכול להעיד על מהירות התגובה של השרת ועל עומס העבודה שלו.
- זיהוי של הזרם או הלקוח ששלח את הנתונים- אם התקבלו נתונים ממקורות שונים, נרצה לשמור על סטטיסטיקות נפרדות עבור כל מקור.
- גודל זרם ממוצע- מהו גודל הזרם הממוצע שנשלח על כל זרם. זה יכול לתת מושג על הכמות הממוצעת של הנתונים שנשלחים דרך כל זרם.
- נתוני המתחילים והמסיימים של כל זרם- מתי התחילה ומתי הסתיימה קבלת הנתונים על כל זרם. זה עוזר לנו לקבוע את תפקודו של כל זרם בזמן.
- סטטוס קבלת הנתונים- האם הנתונים התקבלו בהצלחה או לא, האם הייתה איזו שגיאה בתהליך הקבלה.

4. **receive_packets:** פונקציה זו אחראית על קבלת החבילות מהלקוח ואיסוף הנתונים מהן. השרת יכול לקבל מספר גדול של חבילות מהלקוח, ולכן חשוב להיות מוכן לעבדן באופן אפקטיבי. פונקציה זו משתמשת ב־select.select כדי להמתין לחבילות נכנסות מהלקוח במשך זמן קצר, בעזרת select, השרת משאיר את עצמו במצב נעילה עד שחבילה נכנסת מגיעה מהלקוח, ורק אז הוא מתעורר לקבלתה. ואז הפונקציה משתמשת ב־recvfrom כדי לקבל אותן. היא מנהלת את החבילות בהתאם לתוכן שלהן, מזהה את הזרם שבו הן שולחות ומעדכנת את הנתונים בהתאם.
   - התמודדות עם כמות גדולה של חבילות: הפונקציה מיועדת להתמודד עם קבוצה גדולה של חבילות שמגיעות מהלקוח. זה חיוני במיוחד בריבוי הזרמים, שבו כנראה יהיה מספר גדול של זרמים פתוחים במקביל.
   - זיהוי הזרם: כאשר חבילה מגיעה, השרת מזהה את הזרם שבו היא נשלחה. הזיהוי נעשה על פי הID של הזרם שמצוין בחבילה עצמה.
   - עדכון הנתונים: לאחר שהחבילה מזוהה ונמצא הזרם שבו היא שלוחה, הנתונים מעודכנים בהתאם. זה כולל כמות הנתונים המקבלים בחבילה, ועדכון ספירת החבילות שהתקבלו מהזרם.
5. **send_ready_message ו־send_finished_message:** שתי פונקציות אלו שולחות הודעות ללקוח כדי להודיע לו על התקדמות התהליך. במקרה הזה, הם משתמשים בפרטי ההתחברות שנקבעו בפונקציה process_initial_message ושולחים הודעות בהתאם.
6. **calculate_metrics:** המשימה של פונקציה זו היא לחשב מדדים לנתונים שהתקבלו מהלקוח. בעזרת הנתונים שנאספו והזמנים שנבחרו, היא יכולה לחשב מדדים כגון כמות הנתונים שהועברו, מהירות העברת הנתונים, ועוד. כך ניתן לקבוע את ביצועי השרת ולקבוע אם הם תואמים את הציפיות.
   - חישוב מדדים פעם אחת אחרי כל האיטרציות: הפונקציה לוקחת את המידע שנשמר ביומנים ורשימות, ומחשבת את הממוצע על ידי חילוק של המידע בזמן.

- חישוב מדדים ספציפיים לכל זרם: הפונקציה יכולה לחשב מדדים מפורטים לכל זרם בנפרד, כגון כמות הנתונים שהועברו בזרם, מהירות העברת הנתונים בזרם, זמן התחלה וסיום של הזרם, ועוד.

- חישוב מדדים כוללים לכל איטרציה: לצד המדדים הספציפיים לכל זרם, הפונקציה עשויה לחשב גם מדדים כוללים לכל איטרציה, כגון סך כל הנתונים שהועברו, מהירות העברת הנתונים באופן כולל, וכדומה.

- שמירת המדדים: לאחר שהמדדים נחשבו, ניתן לשמור אותם במבנה הנתונים המתאים, כגון מערך, מילון או מבנה נתונים אחר.

- החזרת המדדים: בסיום החישובים, הפונקציה יכולה להחזיר את המדדים החשובים למטרת עיבוד נוסף, הדפסה, או הצגה למשתמש. הפונקציה מספקת את הנתונים המדוייקים והמספיקים להערכת ביצועי השרת והתאמתם לציפיות.

7. **print_statistics:** פונקציה זו משמשת להדפסת הסטטיסטיקות שנחשבו על ידי הפונקציה לחישוב המדדים. היא מקבלת את הנתונים הנחוצים ומדפיסה אותם למסך, כך שניתן יהיה להבין את התוצאות בצורה ברורה ונוחה.

8. **plot_metrics:** פונקציה זו משמשת ליצירת גרפים המציגים את המדדים שחושבו. כשיש כמות גדולה של נתונים, לפעמים קשה להבין אותם בצורה חזותית, ולכן נוצרים גרפים כדי לאפשר לקורא לראות ולהבין את הנתונים בצורה ברורה יותר.

9. **run:** run מנהל את כל החלק של התקשורת ואסיפת הנתונים.

צד לקוח:

קבועים:

1. **LOCAL_IP**: כתובת ה IP-המקומית שבה מריצה הלקוח את השרת.
2. **LOCAL_PORT**: פתח התקשורת המקומי שבו השרת והלקוח מתקשרים.
3. **PACKET_SIZE**: גודל חבילת הנתונים שנשלחת ונקבעת על ידי הפרוטוקול.

4. :**READY_MESSAGE**הודעה שהשרת שולח ללקוח כדי להודיע לו שהוא מוכן להרצת הרצה חדשה (לזיהוי סוף איטרציה נוכחית והתחלת איטרציה חדשה).

5. :**FINISHED_MESSAGE**הודעה שהשרת שולח ללקוח כדי להודיע לו כי הוא השלים את כל האיטרציות.

6. :**NUM_ITERATIONS**מספר האיטרציות שהלקוח צפוי להריץ.

אתחול:

streams: רשימה שמשמשת לאחסון מספרי הזרמים (streams). בהמשך הקוד, נשתמש ברשימה זו כדי לשלוח את הנתונים על כמה זרמים לשרת.

iteratio: משתנה שסופר לנו את מספר האיטרציה הנוכחית

פונקציות:

1. **generate_random_data & generate_random_file:** שתי הפונקציות אחראיות ליצירת נתונים אקראיים, קבצים או מידע, הדרושים לשליחה.

2. **create_streams:** פונקציה זו יוצרת רשימת זרמים בעלת מספר נתון של זרמים. היא מקבלת כארגומנט את מספר הזרמים ויוצרת רשימה של מספרים שמייצגים את הזרמים.

3. **send_file_over_streams:** משמשת לשליחת קובץ על כמה זרמים. היא מתחילה על ידי פתיחת הקובץ האקראי שירצנו. לאחר מכן, היא קוראת את תוכן הקובץ ומחלקת אותו לפאקטות בגודל קבוע, כדי לאפשר שליחה על פי נפרד על כל זרם. בכל איטרציה, עבור כל פאקטה מתוך הקובץ, הפונקציה משלחת את הנתונים הללו על הזרם המתאים באמצעות הפונקציה send data. עם סיום השליחה של כל הפאקטות, הפונקציה שולחת הודעת סיום עבור כל זרם, כדי לסיים את השליחה עליו. פונקציה זו מסיימת את עבודתה כאשר כל הקובץ נשלח בהצלחה על כל הזרמים.

4. **send_initial_message:** שולחת את הודעת ההתחלה לשרת, כוללת את מספר הזרמים וגודל הקובץ.

5. **send_data:** משמשת לשליחת נתונים על זרם נתון. בעת קבלת הנתונים לשליחה, הפונקציה מקבלת את מספר הזרם (`stream_id`) ואת הנתונים עצמם. לפני שליחת הנתונים על הזרם, הפונקציה משדרת את מספר הזרם בתחילת ההודעה. זה מאפשר לצד המקבל לזהות את הזרם בו הגיעו הנתונים ולעבד אותם בהתאם לזרם המתאים. בסיום השליחה של הנתונים, הפונקציה מסיימת את עבודתה וחוזרת.
6. **send_end_of_stream:** משמשת לשליחת הודעת סיום של זרם מסוים. היא מקבלת את מספר הזרם (stream_id) ושולחת הודעת סיום לזרם זה. הודעת סיום מסמנת כי אין עוד נתונים המגיעים על הזרם המתאים, ובכך מסיימת את השליחה על הזרם. השליחה של הודעת הסיום מאפשרת לצד המקבל לזהות שסיום הנתונים על הזרם הושג, ולסגור את הזרם בהתאם. כשהודעת הסיום נשלחת, הפונקציה מסיימת את עבודתה וחוזרת.

7. **wait_for_ready_message** ו- **wait_for_finished_message:** מחכות להודעות מהשרת, אחת שמסמנת על כך שהשרת מוכן להתחיל את הפעולה הבאה, והשנייה מסמנת על סיום כל הפעולות המתוכננות.

8. **Run** : פונקציה זו מפעילה את הלקוח ונותנת חזרה לו להתחיל לעבוד. היא יוצרת קובץ אקראי ושולחת אותו כמספר פעמים באמצעות הזרמים השונים. היא משתמשת בפונקציות קודמות כדי לשלוח את הנתונים ולקבל את הודעות התחלה וסיום מהשרת.

מסקנה- גרף

המסקנה מהפרויקט על ריבוי זרימות היא הגרף.

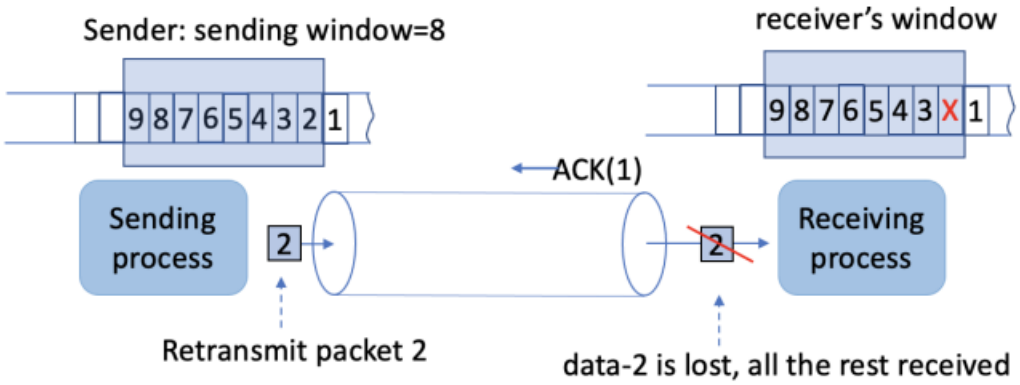
פונקציית ()sleep שסייעה במימוש של הלקוח גרמה לכך שהאטנו במעט את השולח בגלל שנוצר לנו מצב בו איבדנו עד 50% מהפאקטות בכל זרימה. לכן, כמו שניתן להסיק מהגרף - בגלל הביצועים הנמוכים היינו צריכים להאט את קצב השליחה כדי שהמקבל יוכל להתמודד עם כל הפקטות בו זמנית.

מסקנה, לפי הגרף אפשר לראות שככל שיש יותר זרימות, יש יותר עומס ומידע בו זמנית, ולכן הביצועים הופכים להיות פחות טובים.

חלק "יבש":

1. 5 חסרונות/מגבלות של TCP :

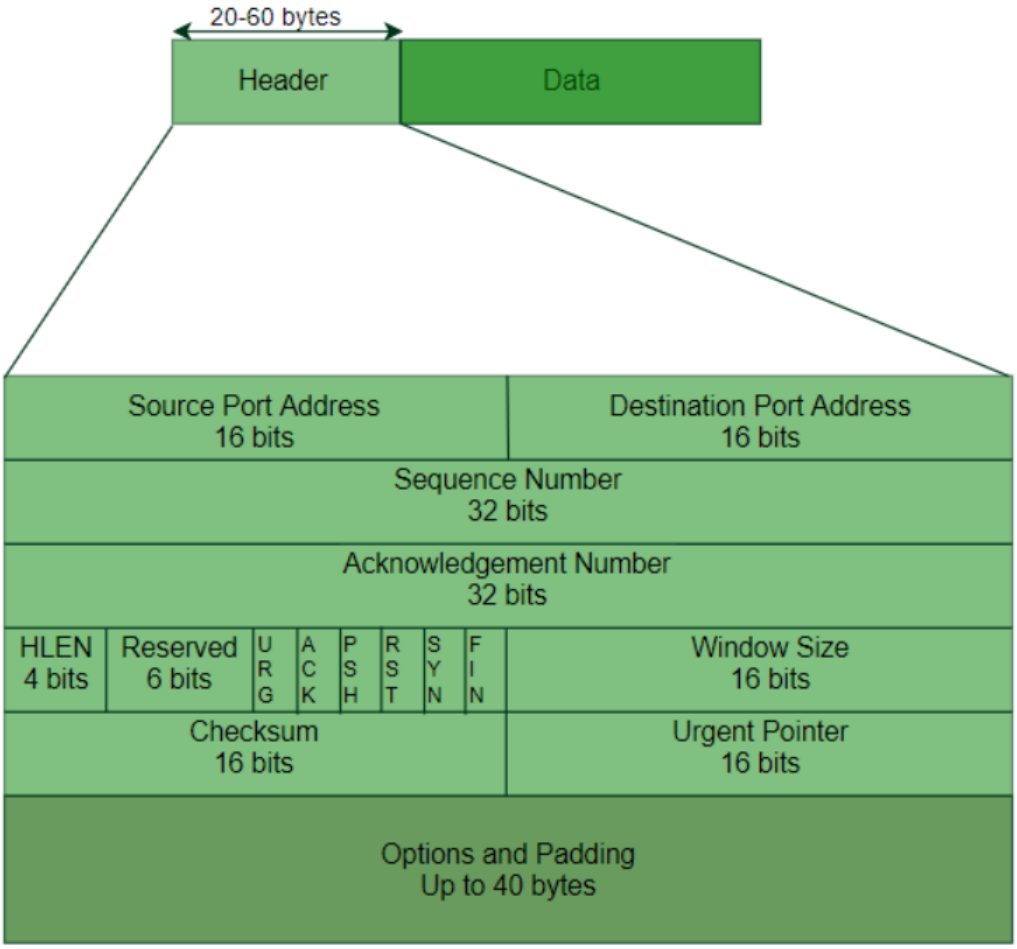
1. בקרת עומס למול בקרת אמינות - TCP משתמש באותו מנגנון חלון הן לבקרת עומס והן לבקרת אמינות. כתוצאה מכך, אובדן חבילה יכול להגביל את ההתקדמות ולגרום לחלון העומס להיתקע גם כאשר אין חבילות ברשת דבר הפוגע ביעילות השימוש ברוחב הפס וכן לא להציג במדויק את החבילות שנמצאות ברשת .
בתמונה המצורפת ניתן לראות דוגמה בה לאחר אובדן פקטה 2 מונע מחלון הזרימה להתקדם.



2. חסימת ראש השורה (HLB)- כיוון שTCP מבטיח מסירה רציפה של זרם הבתים, אובדן של חבילה אחת יגרום לחסימת המסירה של כל החבילות הבאות עד שהחבילה האבודה תשוחזר.
בתמונה המצורפת ניתן לראות כי אובדן פקטה 1 חסם את קבלת שאר הפקטות.

3. עיכוב עקב הקמת חיבור – כאשר נקודות קצה רוצות לתקשר דרך TCP, דרוש תהליך "לחיצת ידיים" בן 3 שלבים (3-way handshake) כדי להקים חיבור TCP לפני ששליחת נתוני אפליקציה תתאפשר. אם רוצים לאבטח את החיבור עם TLS, דרוש עוד סיבוב תקשורת (round trip) כדי שהצדדים יחליפו אישורי אבטחה מה שיגרום לעיכוב נוסף.

4. מגבלות הheader – הheader של ה-TCP מורכב משדות באורך קבוע (בנוסף לשדות אלו, יש שדה אופציונלי המוגבל ל-40 בתים לכל היותר).
שדות מסוימים בheader מושפעים מהגידול המתמשך במהירויות הרשת לאורך הזמן : מספר סידורי (sequence number) ושדה האישור (ACK) הם באורך 4 בתים כל אחד ופועלים במהירות רבה ולעומתם שדה גודל חלון ובקרת הזרימה הם באורך 2 בתים ומגבילים את ביצועי TCP במהירויות רשת גבוהות.
בתמונה המצורפת ניתן לראות את מבנה הheader



5. זיהוי החיבור תלוי בכתובת IP - כתובת ה-IP של כל אחד ממשתמשי הקצה בחיבור TCP עשויה להשתנות במהלך החיבור בגלל מגוון סיבות. מכיוון ש-TCP משתמש בשילוב של כתובות ה-IP ומספרי הפורט של שני מארחי הקצה כמזהה החיבור, כל שינוי בכתובת IP ישבור את החיבור הקיים, מה שיגרום לכל המידע שהוחלף עד לאותו רגע להימחק. על מנת ליצור חיבור חדש , תידרש לחיצת ידיים חדשה.

2. 5 תפקידים שפרוטוקול תעבורה צריך למלא :

א. הגדרת מזהה חיבור ומזהה מידע - פרוטוקול תעבורה צריך להגדיר מזהה חיבור ייחודי גלובלי שקושר בין שני צדדי החיבור (רצוי שיהיה בלתי תלוי בכתובת IP כיוון שזו יכולה להשתנות במהלך החיבור אך נדרשת למיפוי אמין לכתובות IP), וכמו כן מזהה מידע ייחודי שחייב להיות מוחלף באופן אמין עם הצד השני.

ב. ניהול חיבור תעבורה – פרוטוקול תעבורה אחראי לקשירה בין שני צדדי החיבור, הקמת החיבור וסיום החיבור, בקרה על החלפת המידע בקרה בין הצדדים ותמיכה בשינויים בכתובת IP של המארח.

ג. אמינות במסירת המידע - נדרש מזהה ייחודי למידע אשר באמצעותו המידע יועבר באמינות בין המשתמשים, וכן בקרת זרימה למסירה אמינה (רצוי להימנע מחסימת HOL).

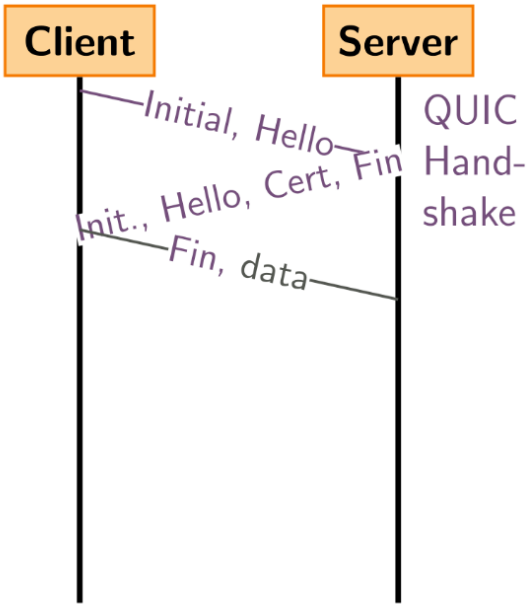
ד. בקרת עומס – פרוטוקול התעבורה אחראי לבקרה על הפקטות ברשת.

ה. אבטחה - נדרשת אבטחה מקצה לקצה (end-to-end). עם זאת, יצירת חיבור ואבטחה הינם שני תהליכים נפרדים- לדוגמה, באמצעות TCP נוצר חיבור ועליו נוספת שכבת האבטחה TLS .

3. QUIC משלב את "לחיצות הידיים" של התעבורה וההצפנה יחד, ומשיג את המידע ב-RTT-1. בהתאם לכך, תהליך לחיצת הידיים הראשוני כולל גם את העברת התעבורה וגם את מסירתה באופן מוצפן בו זמנית ובכך מקטין את זמן הקמת החיבור המאובטח. בכך, בא לידי ביטוי השיפור מחיבור הTCP אשר מטפל בנפרד בשלב התעבורה ובשלב האבטחה .

QUIC משתמש בחבילה הראשונית לזיהוי חיבורים חדשים. בפקטה הראשונית לכל נקודת קצה קיים מזהה חיבור ייחודי באמצעותו המשתמש השני יוכל לשלוח פקטות חדשות למשתמש היעד . בעת קבלת הפקטה הראשונית , המשתמש יכול לאמת את כתובת הלקוח באמצעות שליחת חבילה חוזרת עם token אותו יצטרך לאמת הלקוח על מנת להמשיך את תהליך החיבור. בדרך זו , QUIC מאפשר חיבור אמין .

בנוסף, QUIC מאפשר ללקוח לשלוח לשרת נתוני אפליקציה מוצפנים ב-RTT-0 כבר בחבילה הראשונה ע"י שימוש חוזר בפרמטרים מהחיבור הקודם .
באמצעות תמיכה בשליחת נתונים בRTT-0 , ניתן לטפל גם במקרים בהם נדרש T/TCP.



4. ל-QUIC יש שני סוגי header (בניגוד ל-TCP שבו הheader קבוע).
הפקטות להקמת החיבור צריכות להכיל כמה פריטי מידע, ולכן הן משתמשות בheader ארוך.
לאחר הקמת החיבור, רק שדות מסויימים נדרשים ולא כולם ולכן החבילות הבאות ישתמשו בheader קצר יותר ובכך ישנה יעילות גבוהה יותר וישנו שיפור מפרוטוקול TCP בו הheader קבוע. בתמונה המצורפת ניתן לראות את שני סוגי הheaders .
בנוסף, לכל חבילה ב-QUIC מוקצה מספר ייחודי שעולה באופן מונוטוני בהתאם לשידור החבילות והוא אינו תלוי בשחזור אובדן פקטה. כך, ניתן לדעת במדויק כמה חבילות נמצאות בתוך הרשת. הדבר משפר את בקרת העומס בTCP שמשתמשת באותו מנגנון לאמינות ולעומס.

Long Header

Header Form	Fixed Bit	Long Packet Type	Type Speicific bits	Version ID	DCID Len	DCID	SCID Len	SCID
1 bit	1 bit	2 bits	4 bits	32 bits	8 bits	0-160 bits	8 bits	0-160 bits

Short Header

Header Form	Fixed Bit	Spin bits	Reserved	Key Phase	P	DCID	Packet Number	Protected payload
1 bit	1 bit	1 bit	2bits	1 bit	2 bits	160 bits	P+8 bits	

5. QUIC מזהה אובדן על בסיס חבילות.
עבור כל חבילה שקיבלה ACK, כל המסגרות שהיא הכילה נחשבות שהתקבלו.
המסגרות נחשבות לאבודות אם החבילה לא קיבלה אישור כאשר חבילה שנשלחה מאוחר יותר כן קיבלה אישור, ובמדיה שזוהו אחד מהבאים:
1. אם המספר של החבילה הנוכחית בתעבורה קטן ממספר החבילה האחרונה שקיבלה אישור.
2. אם חבילה בתעבורה נשלחה לפני זמן מסוים ביחס למועד קבלת האישור על חבילה מאוחרת יותר.

אחרי שאובדן זוהה, המסגרות האבודות יוצאות בפקטות חדשות בעלות מספרי חבילה חדשים (ללא תלות בחבילות האבודות). בכך, QUIC תומך במסירה אמינה בדומה ל- TCP ומאפשר זיהוי ושחזור חבילות אבודות .

6. בקרת העומס ב-QUIC מפרידה בין בקרת העומס לבין בקרת האמינות. הQUIC משתמש במספרי חבילה לצורך בקרת עומס, ובoffset של מסגרת הזרם (stream frame) לצורך בקרת אמינות- דבר הפותר את הבעיה הקיימת ב-TCP שמשתמש באותו המנגנון לשניהם.

בדומה לבקרת העומס של TCP, פרוטוקול הQUIC משתמש בבקרת עומס שמגבילה את המספר המרבי של בתים שהשולח יכול להחזיק בתעבורה בכל רגע נתון. QUIC אינו מייצר אלגוריתמים חדשים משלו לבקרת עומס וגם לא מחייב שימוש באלגוריתם ספציפי אלא מאפשר למשתמש ליישם בעצמו מנגנון בקרת עומס כרצונו.

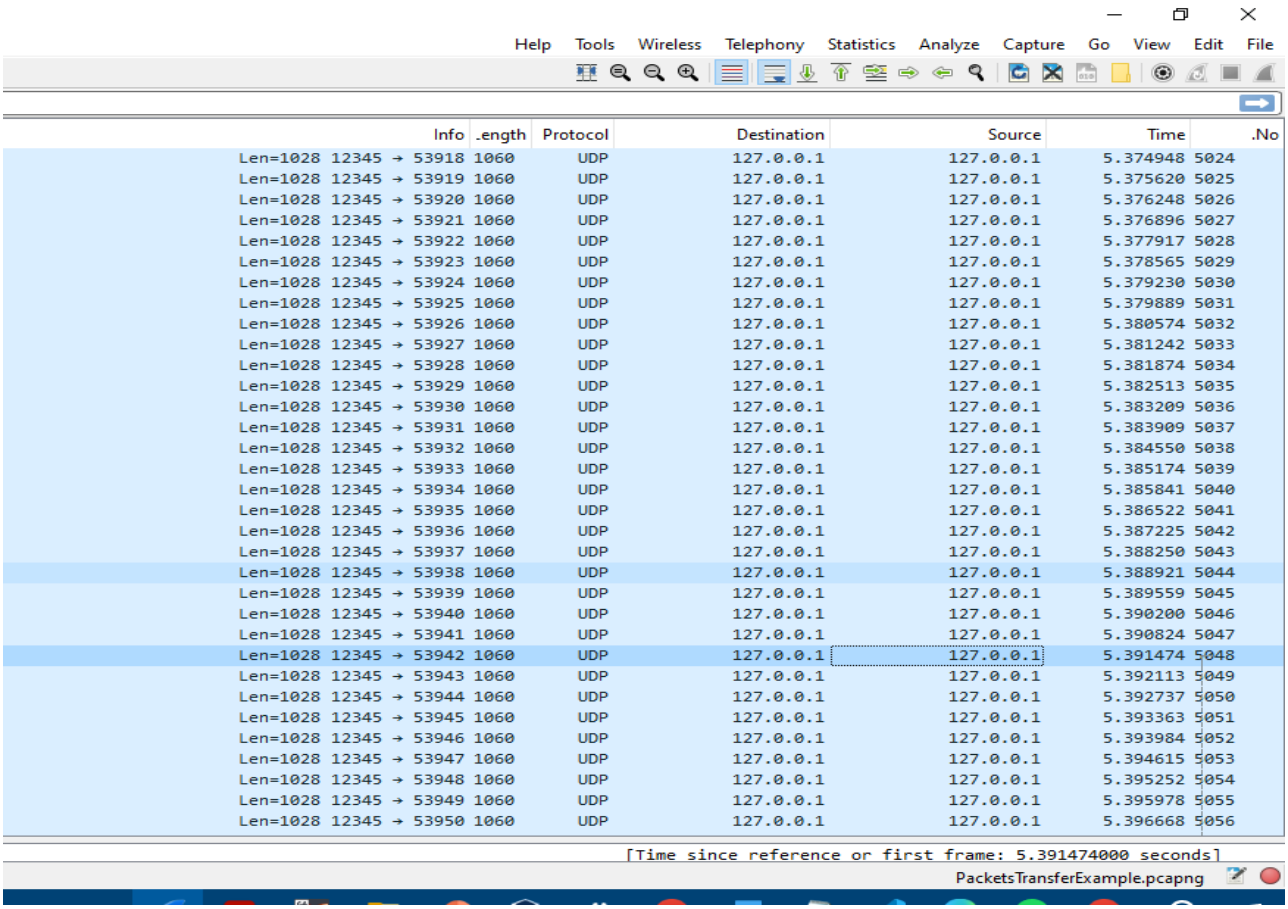
כדי למנוע הפחתה מיותרת של חלון העומס, QUIC לא מקטין את חלון העומס אלא במקרה של זיהוי עומס מתמשך. עומס מתמשך מזוהה כאשר שתי חבילות הדורשות אישור אובדות ואף אחת מהחבילות שנשלחו ביניהן לא קיבלה אישור, הייתה דגימת RTT לפני שהן נשלחו, וההפרש בין זמני השליחה שלהן עולה על משך העומס המתמשך.

בנוסף, שולח QUIC יסדיר את קצב השליחה שלו כדי להפחית את הסיכויים לגרימת עומס בטווח הקצר, על ידי וידוא שהמרווח בין שליחת החבילות עולה על סף מסוים. סף זה מחושב על בסיס ה-RTT הממוצע, גודל חלון העומס, וגודל החבילה.

הקלטות wireshark: - אנחנו בסביבת עבודה Windows

```
Frame 5032: 1060 bytes on wire (8480 bits), 1060 bytes captured (8480 bits) on interface \Device\NPF_Loopback, id 0
Null/Loopback
Family: IP (2)
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
User Datagram Protocol, Src Port: 53926, Dst Port: 12345
Data (1028 bytes)
```

```
000  02 00 00 00 45 00 04 20  60 32 00 00 80 11 00 00   ····E··  `2·····
010  7f 00 00 01 7f 00 00 01  d2 a6 30 39 04 0c 8c b7   ····· ··09····
020  00 00 00 01 3b 52 d8 db  16 8e a6 f1 ec ff 9e 9c   ····;R·· ·····
030  ee 50 4c 91 95 84 93 66  16 3c a1 84 a8 cc df a1   ·PL····f ·<·····
040  15 d2 05 af dc e5 1d 82  34 a1 d4 25 46 77 38 c4   ····· 4··%Fw8·
050  97 12 52 e8 4c e6 97 14  11 c7 2d 3b 0f f9 52 68   ··R·L··· ··-;··Rh
060  94 31 95 1d 27 94 e7 d4  6a 29 37 5f 20 38 17 b0   ·1··'··· j)7_ 8··
070  35 59 09 4a 95 63 0c 4b  0c 03 98 79 55 46 97 6b   5Y·J·c·K ···yUF·k
080  d1 96 ea ae 8e 10 65 07  da fc 7d cf 64 76 22 dd   ·····e· ··}·dv"·
090  8e 78 42 f0 42 d2 10 fb  f9 12 7c 73 30 ea 25 53   ·xB·B··· ··|s0·%S
0a0  52 7f d0 d3 cf da 9b a5  bc 14 70 f3 82 a1 b9 7a   R····· ··p····z
0b0  cb 5e bc a9 86 89 03 fc  b5 92 de 3f 00 96 14 e9   ·^····· ···?····
0c0  02 6d 19 ee 9e 0e b3 9b  13 61 44 0e e0 58 5d 95   ·m····· ·aD··X]·
0d0  a7 6f 88 04 0b af 02 f5  d3 45 3a 1c 21 67 be 61   ·o····· ·E:·!g·a
0e0  61 cd 89 63 13 8c c3 34  c5 10 38 61 26 7e 1b cd   a··c···4 ··8a&~··
0f0  dc 6e 46 20 f6 d6 24 db  de 9e 3a e0 32 ec b8 bf   ·nF ··$· ··:·2···
2 · Time: 5.380574 · Source: 127.0.0.1 · Destination: 127.0.0.1 · Protocol: UDP · Length: 1060 · Info: 53926 → 12345 Len=1028
Show packet bytes
```



Info	.ength	Protocol	Destination	Source	Time	.No
Len=1028 12345 → 53918	1060	UDP	127.0.0.1	127.0.0.1	5.374948	5024
Len=1028 12345 → 53919	1060	UDP	127.0.0.1	127.0.0.1	5.375620	5025
Len=1028 12345 → 53920	1060	UDP	127.0.0.1	127.0.0.1	5.376248	5026
Len=1028 12345 → 53921	1060	UDP	127.0.0.1	127.0.0.1	5.376896	5027
Len=1028 12345 → 53922	1060	UDP	127.0.0.1	127.0.0.1	5.377917	5028
Len=1028 12345 → 53923	1060	UDP	127.0.0.1	127.0.0.1	5.378565	5029
Len=1028 12345 → 53924	1060	UDP	127.0.0.1	127.0.0.1	5.379230	5030
Len=1028 12345 → 53925	1060	UDP	127.0.0.1	127.0.0.1	5.379889	5031
Len=1028 12345 → 53926	1060	UDP	127.0.0.1	127.0.0.1	5.380574	5032
Len=1028 12345 → 53927	1060	UDP	127.0.0.1	127.0.0.1	5.381242	5033
Len=1028 12345 → 53928	1060	UDP	127.0.0.1	127.0.0.1	5.381874	5034
Len=1028 12345 → 53929	1060	UDP	127.0.0.1	127.0.0.1	5.382513	5035
Len=1028 12345 → 53930	1060	UDP	127.0.0.1	127.0.0.1	5.383209	5036
Len=1028 12345 → 53931	1060	UDP	127.0.0.1	127.0.0.1	5.383909	5037
Len=1028 12345 → 53932	1060	UDP	127.0.0.1	127.0.0.1	5.384550	5038
Len=1028 12345 → 53933	1060	UDP	127.0.0.1	127.0.0.1	5.385174	5039
Len=1028 12345 → 53934	1060	UDP	127.0.0.1	127.0.0.1	5.385841	5040
Len=1028 12345 → 53935	1060	UDP	127.0.0.1	127.0.0.1	5.386522	5041
Len=1028 12345 → 53936	1060	UDP	127.0.0.1	127.0.0.1	5.387225	5042
Len=1028 12345 → 53937	1060	UDP	127.0.0.1	127.0.0.1	5.388250	5043
Len=1028 12345 → 53938	1060	UDP	127.0.0.1	127.0.0.1	5.388921	5044
Len=1028 12345 → 53939	1060	UDP	127.0.0.1	127.0.0.1	5.389559	5045
Len=1028 12345 → 53940	1060	UDP	127.0.0.1	127.0.0.1	5.390200	5046
Len=1028 12345 → 53941	1060	UDP	127.0.0.1	127.0.0.1	5.390824	5047
Len=1028 12345 → 53942	1060	UDP	127.0.0.1	127.0.0.1	5.391474	5048
Len=1028 12345 → 53943	1060	UDP	127.0.0.1	127.0.0.1	5.392113	5049
Len=1028 12345 → 53944	1060	UDP	127.0.0.1	127.0.0.1	5.392737	5050
Len=1028 12345 → 53945	1060	UDP	127.0.0.1	127.0.0.1	5.393363	5051
Len=1028 12345 → 53946	1060	UDP	127.0.0.1	127.0.0.1	5.393984	5052
Len=1028 12345 → 53947	1060	UDP	127.0.0.1	127.0.0.1	5.394615	5053
Len=1028 12345 → 53948	1060	UDP	127.0.0.1	127.0.0.1	5.395252	5054
Len=1028 12345 → 53949	1060	UDP	127.0.0.1	127.0.0.1	5.395978	5055
Len=1028 12345 → 53950	1060	UDP	127.0.0.1	127.0.0.1	5.396668	5056

ip 127.0.0.1 הקובץ נשלח מהמחשב לעצמו - כמו שהגדרנו בקוד

רואים שזהו פרוטוקול udp כמו שהגדרנו בקוד

```
Frame 5032: 1060 bytes on wire (8480 bits), 1060 bytes captured (8480 bits) on interface \Device\NPF_Loopback, id 0
                                                                                                        Null/Loopback
                                                Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
                                                                                          Version: 4 = .... 0100
                                                                                 Header Length: 20 bytes (5) = 0101 ....
                                                Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
                                                                                       Total Length: 1056
                                                                                 Identification: 0x6032 (24626)
                                                                                          Flags: 0x0 = .... .000
                                                                                 Fragment Offset: 0 = 0000 0000 0000 0...
                                                                                           Time to Live: 128
                                                                                          Protocol: UDP (17)
                                                                                 Header Checksum: 0x0000 [validation disabled]
                                                                                 [Header checksum status: Unverified]
                                                                                          Source Address: 127.0.0.1
                                                                                          Destination Address: 127.0.0.1
                                                User Datagram Protocol, Src Port: 53926, Dst Port: 12345
                                                                                                   Data (1028 bytes)
```

* הsource port זה המחשב שלי

* ה dest port זה כמו שהגדרנו בקוד 12345

* לכל זרימה יש ID משלה, אותו הצמדנו לפקטות כדי שנוכל לזהות איזה פקטה הגיעה מאיזה זרימה בשביל לחשב נתונים עבור זרימות/איטרציות

```
060 bytes on wire (8480 bits), 1060 bytes captured (8480 bits) on interface \Device\NPF_Loopback, id 0
                                                                                          Null/Loopback
                                   Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
                                      User Datagram Protocol, Src Port: 53926, Dst Port: 12345
                                                                                Source Port: 53926
                                                                           Destination Port: 12345
                                                                                       Length: 1036
                                                                     Checksum: 0x8cb7 [unverified]
                                                                     [Checksum Status: Unverified]
                                                                              [Stream index: 4345]
                                                                                     [Timestamps]
                                                        [Time since first frame: 0.000000000 seconds]
                                                      [Time since previous frame: 0.000000000 seconds]
                                                                         UDP payload (1028 bytes)
                                                                                   Data (1028 bytes)
```