פרויקט גמר

קורס רשתות תקשורת

מדעי המחשב, אוניברסיטת אריאל שבשומרון

מגישים: אוראל שלם, יאיר מרגלית, אליחי זרגריאן, אביחי בן דוד

תוכן עניינים

הקדמה

מה זה פרוטוקול QUIC?

פרוטוקול QUIC (Quick UDP Internet Connections) הוא פרוטוקול תקשורת שפותח על ידי גוגל במטרה לשפר את ביצועי העברת המידע באינטרנט. הוא נועד להחליף או לשפר את הפרוטוקול הסטנדרטי TCP תוך שימוש ב UDP כבסיס להעברת המידע. הפרוטוקול כולל שיפורים שנועדו להקטין את זמן החביון (latency) ולהגביר את הביצועים הכוללים של חיבורי האינטרנט.

נמצא כיום בשימוש נרחב בכמה מהשירותים של גוגל, כולל YouTube ו Google Search, והוא תופס תאוצה גם בקרב חברות נוספות ושירותים שונים ברחבי האינטרנט. הפרוטוקול גם אומץ כבסיס לפרוטוקול HTTP/3 גרסה חדשה של HTTP שנועדה להחליף את HTTP/2 ולהביא את היתרונות של QUIC לשימוש רחב יותר.

פרוטוקול QUIC מביא עמו שיפורים משמעותיים בהשוואה לפרוטוקולים הוותיקים כמו TCP, הן מבחינת ביצועים, אמינות, ואבטחה. בזכות תכונותיו החדשניות, הוא יכול לספק חווית משתמש טובה יותר בעבודה עם אפליקציות ושירותי אינטרנט מודרניים.

לטובת פרויקט גמר בקורס רשתות תקשורת בחרנו לממש את התכונה של ריבוי זרימות ב-Python.

קובץ זה מאגד את התשובות לחלק היבש והסבר על המימוש.

פתרון חלק יבש

5 חסרונות של פרוטוקול TCP:

---

1. קשר בין בקרת גודש להעברה אמינה:
כל איבוד חבילה מקפיא את התקדמות החלון עד לשחזור האובדן, מה שגורם לעיכובים ולא משקף את המצב האמיתי ברשת[1].

2. חסימת ראש התור (Head of Line Blocking):
איבוד חבילה אחת מונע העברת כל החבילות הבאות עד לשחזור החבילה האבודה, מה שגורם לעיכוב במשלוח הנתונים לאפליקציה.

3. שיהוי בהקמת חיבור:
כל חיבור חדש דורש תהליך של שלושה שלבים (3-way handshake), ואם רוצים לאבטח את החיבור עם TLS, נדרש סבב נוסף של לחיצת ידיים, מה שמוסיף זמן שיהוי.

4. מגבלות כותרת קבועה:
הכותרת של TCP מוגבלת בגודל השדות שלה, מה שמקשה על הוספת פונקציות חדשות ועלול להגביל ביצועים ברשתות מהירות.

5. מזהה חיבור תלוי בכתובת IP:
שינוי בכתובת IP של אחד הקצוות במהלך חיי החיבור גורם לשבירת החיבור, מצריך הקמת חיבור מחדש וגורם לאובדן נתונים קודמים.

---

[1] השולח לא יכול לדעת בדיוק כמה חבילות נמצאות ברשת כרגע או כמה חבילות נוספות הוא יכול לשלוח בלי לגרום לעומס יתר. זה יוצר מצב שבו השולח מוגבל ביכולת לשלוח חבילות נוספות, למרות שהרשת עשויה להיות פנויה לקבל חבילות נוספות.

5 תפקידים שפרוטוקול תעבורה צריך למלא:

1. מזהה חיבור ומזהה נתונים ייחודיים: פרוטוקול תעבורה חייב לספק מזהה ייחודי לכל חיבור ולכל נתונים כדי לאפשר זיהוי וניהול יעיל.

2. ניהול חיבורי תחבורה: הפרוטוקול צריך לאפשר התקנה וסיום של חיבור, ניהול מצבי חיבור, ותמיכה בהעברת מידע בקרה בין קצוות החיבור. ותמיכה בשינויי כתובת IP של שני הצדדים.

3. העברת נתונים אמינה: הפרוטוקול חייב לספק מסגרת להעברת נתונים אמינה, כולל בקרת זרימת חלון כדי למנוע חסימת ראש התור.

4. בקרת גודש: הפרוטוקול צריך לכלול מנגנון לבקרת הגודש, כולל שליטה במספר החבילות ברשת וניהול הכנסות לרשת.

5. אבטחה: יש לספק אבטחה מתאימה לפרוטוקול, כולל ייעוץ באבטחת הפרוטוקול, קריפטוגרפיה (הצפנה) ואימות של הצד המרוחק.

אופן פתיחת קשר בפרוטוקול QUIC וכיצד הוא משפר את החיסרון בפרוטוקול TCP:

התחלת הקשר ע"י הלקוח: הלקוח שולח חבילה ראשונה עם הודעת ClientHello. בחבילה זו מופיעים פרטי צפנים נתמכים, מפתח פומבי של Diffie-Helman ופרמטרי תעבורה של QUIC.

התגובה מהשרת: השרת שולח חבילה ראשונה עם הודעת ServerHello. בתגובה זו השרת בוחר פרטי צפנים לחיבור, מפתח פומבי של Diffie-Helman ואולי הרחבות נוספות של TLS.

אימות וסיכום הקשר: השרת שולח הודעות נוספות הכוללות Certificate ו-EncryptedExtensions. מאשר את המפתח הפרטי שלו באמצעות הודעת CertificateVerify. לבסוף מסיים את התהליך עם הודעת סיום שמאמת את הקשר האבטחתי.

סיום לחיצת היד: הלקוח גם שולח הודעת סיום מאשר את סיום התהליך באמצעות הודעת DONE_HANDSHAKE שנשלחת לשרת.

התחלת המעבר לנתוני האפליקציה: לאחר האימות שני הצדדים יכולים לשלוח נתוני אפליקציה עם חבילות 1-RTT.

בזכות היכולות של QUIC לשלב את ה-handshake התחבורתי והקריפטוגרפי ביחד, ניתן להקים חיבור מאובטח ולהתחיל בהעברת נתונים מאוד מהר, תוך חיסכון בזמן ובמשאבים בהשוואה לפתרונות קודמים כמו TCP.

יתרונות שמציע QUIC לעומת חסרונות שיש בTCP:

- שיפור ביצועי התעבורה המוצפנת:
  - חיסרון ב-TCP לא תוכנן מראש לטפל בתעבורה מוצפנת ולכן יכול לסבול מביצועים נמוכים יותר כשמשתמשים ב-TLS.
  - יתרון ב-QUIC: QUIC מתוכנן מראש לשפר את ביצועי התעבורה המוצפנת באמצעות שילוב של פרוטוקול UDP וגישה ישירה להצפנה עם TLS 1.3, שמבטיחים תעבורה מוצפנת יעילה יותר.
  - ציטוט: " QUIC is a transport protocol originally designed by Google to improve transport performance for encrypted traffic."

- יכולת ניהול קשר עמיד לשינויים בכתובות רשת:
  - חיסרון ב-TCP: שינוי כתובת IP של אחד מהקצוות יקטע את הקשר ב-TCP כיוון שהפרוטוקול משתמש בכתובות ה-IP ובמספרי הפורטים כמזהה ייחודי לקשר.
  - יתרון ב-QUIC: QUIC משתמש במזהי חיבור (Connection ID) ייחודיים המאפשרים להמשיך את הקשר גם במקרים של שינוי כתובות IP. כך, גם אם הכתובת של אחד מהקצוות משתנה (למשל עקב מעבר בין רשתות), הקשר נשמר והתקשורת לא נקטעת.
  - ציטוט: " QUIC connection has the ability to survive changes in underlying protocol addresses with the usage of Connection ID."

- הקמת קשר מאובטח ומהירה יותר:
  - חיסרון ב-TCP: הקמת קשר ב-TCP דורשת 3-way handshake, והוספת TLS דורשת RTT נוסף, מה שמוביל להשהיה גבוהה יותר.
  - יתרון ב-QUIC: QUIC משלב את תהליך ה-handshake התחבורתי והקריפטוגרפי יחד, מה שמאפשר הקמת קשר מאובטח בסיבוב אחד בלבד (1-RTT), במקום בשני סיבובים כמו ב-TCP ו-TLS. זה מפחית את ההשהיה הנדרשת להקמת קשר מאובטח.
  - ציטוט: " QUIC combines transport and cryptographic handshakes together, acquiring the information necessary for both in 1-RTT."

- יכולת שינוי כתובת דינמית ותמיכה בניתוב מחדש:
  - חיסרון ב-TCP: שינוי כתובת IP במהלך הקשר יקטע את החיבור וידרוש פתיחה מחדש של החיבור באמצעות 3-way handshake חדש.
  - יתרון ב-QUIC: QUIC תומך בשינוי כתובת דינמית על ידי שימוש במזהי חיבור חלופיים שניתנים מראש. כך, כאשר קצה אחד משנה את הכתובת שלו, הוא יכול להשתמש במזהי החיבור החלופיים כדי להמשיך את התקשורת מבלי לאבד את הנתונים.
  - ציטוט: " QUIC endpoint can provide its peer with alternative connection IDs in advance and a migrating endpoint can use different connection IDs when sending data from different addresses."

מבנה החבילה של QUIC:

QUIC משתמש בשני סוגים של כותרות לחבילות – כותרת ארוכה בעת הקמת החיבור וכותרת קצרה לאחר ההקמה. הדבר מאפשר גמישות ויעילות רבה יותר בתקשורת לאחר ההקמה, בניגוד לכותרת הקבועה של TCP.

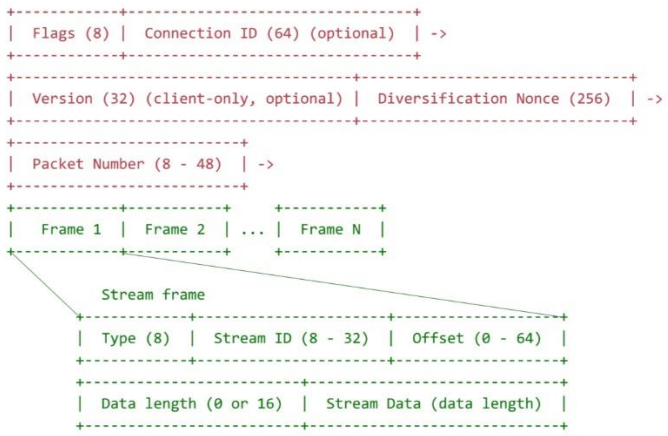


Figure 5: Structure of a QUIC packet, as of version 35 of Google's QUIC implementation. Red is the authenticated but unencrypted public header, green indicates the encrypted body. This packet structure is evolving as QUIC gets standardized at the IETF [2].

ציטוט: " Unlike TCP where the packet header format is fixed, QUIC has two types of packet headers. QUIC packets for connection establishment need to contain several pieces of information, it uses the long header format. Once a connection is established, only certain header fields are necessary, the subsequent packets use the short header format for efficiency."

כיצד QUIC מתמודדת עם בקרת העומס:

- חלונות עומס: QUIC משתמשת במנגנון של חלונות עומס כדי להגביל את הכמות של הנתונים שהשולח יכול לשלוח ללא אישור חזרה. כלומר, יש לכל שולח חלון מסוים שמציין את הכמות המרבית של הנתונים שהוא יכול לשלוח ללא קבלת אישור לפני שהחלון נסגר.
  - ציטוט: " QUIC utilizes a window-based congestion control scheme that limits the maximum number of bytes the sender might have in transit at any time."

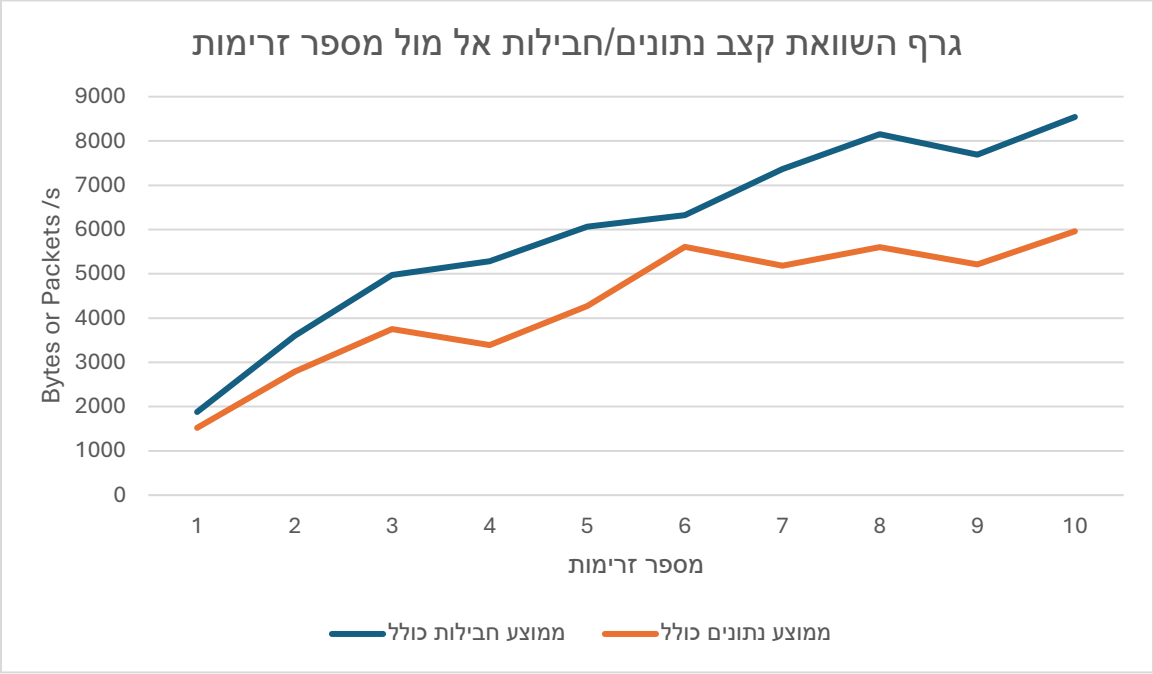
- מניעת צפיפות עומס מתמשכת: QUIC מנסה לא להפחית את חלונות העומס במקרים של אובדן חבילות רגיל. זאת במטרה למנוע צפיפות עומס מיותרת שעלולה להפריע לביצועי הרשת.
  - ציטוט: " To avoid unnecessary congestion window reduction, QUIC does not collapse the congestion window unless it detects persistent congestion."

- פיזור שליחת חבילות: QUIC מפזרת את השליחה של החבילות על מנת למנוע צפיפות עומס קצרת טווח. השליחה מתבצעת בהתאם למדדי ה-RTT וגודל החלון העומס.
  - ציטוט: "A QUIC sender will pace its sending to reduce the chances of causing short-term congestion by ensuring its inter-packet sending interval exceeds a limit calculated based on the average RTT (smoothed_rtt), the congestion window size, and the packet size."

- זיהוי ושחזור אובדני חבילות: QUIC מזהה ומשחזר חבילות שנאבדו באמצעות התנהלות מתקדמת עם עומס מתמשך ובעזרת טיימר PTO. המנגנון מספק רכיב חסינות מתקדם מבוסס על הזיהוי המהיר והשחזור של חבילות שנאבדו.
  - ציטוט: " To detect the loss of tail packets, QUIC will initialize a timer for the Probe Timeout (PTO) period whenever a packet requiring acknowledgment is sent"

גרפים:

הגרף הבא מתאר את השינוי בקצב העברת הנתונים/חבילות אל מול מספר הזרימות:



נשים לב שהגדלת הזרימות עד 5 זרימות לא מועילה בהורדת קצב הנתונים לשנייה. אך מ-6 זרימות ומעלה ניתן להבחין בירידה בממוצע הנתונים לשנייה וזו אודות למקביליות השליחה שמתרחשת בעת שליחת הקבצים.

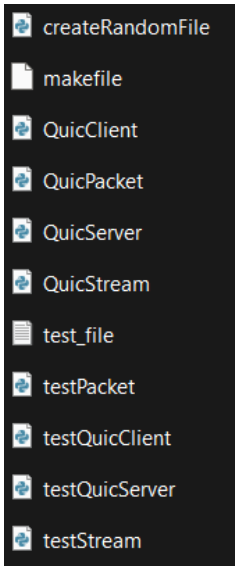
*נזכיר שעבור כל זרימה אנו שולחים קובץ אקראי (בגודל בין 1000 ל-2000 בתים).

הסבר על המימוש:

הקדמה

בחרנו לכתוב את המימוש לריבוי ב- Python לאור הנוחות בכתיבה, קריאות והבנה טובה יותר, ושימוש בספריות. לצורך מימוש ההיבט של ריבוי זרימות ישנו צורך בשימוש במקביליות לכך, השתמשנו ב- Threads.

מבנה הפרויקט



- **`createRandomFile`** לצורך ייצור קבצים אקראיים לשליחה.
- **`Makefile`** פונקציונלי לטובת חזרות השליחה.
- **`QuicClient`** מימוש צד לקוח.
- **`QuicServer`** מימוש צד שרת.
- **`QuicStream`** מימוש אובייקט זרימה.
- **`QuicPacket`** מימוש אובייקט חבילה.
- וקבצי טסטים.

הרצה

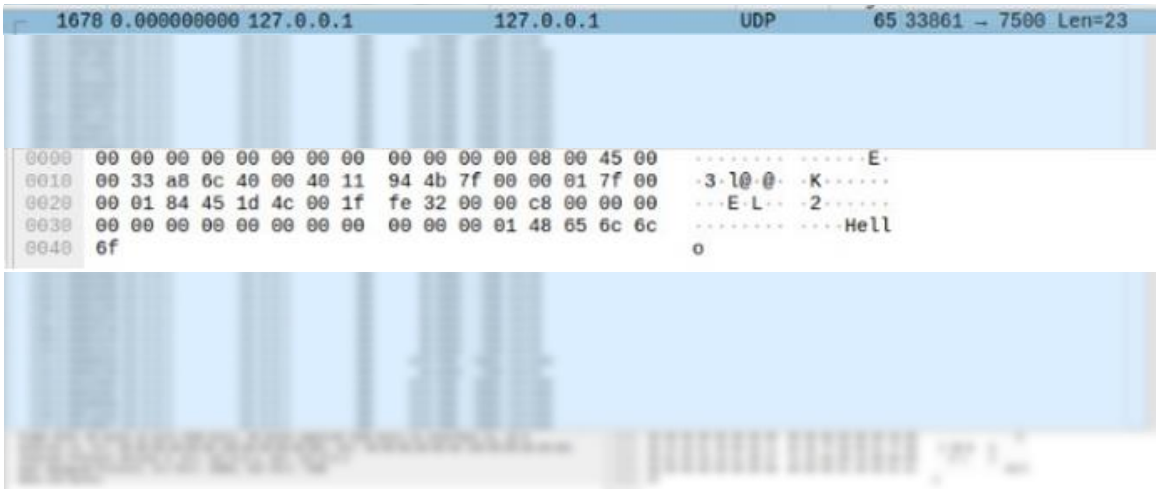
פקודות הרצה שימושיות:

- **`$make run`** מריץ לקוח ושרת ומבצע של שליחה של x קבצים בהתאם לקובץ makefile
- **`$make kill`** לצורך הפסקת תהליכים במקרה של שגיאה
- **`$make clean`** ניקוי הקבצים האקראיים, סטטיסטיקות ושאר הקבצים הזמניים.

*בתוך ה- makefile ניתן לערוך את הפרמטר **`$(ITERATIONS)`** על מנת לקבוע את מס' הקבצים/זרימות.

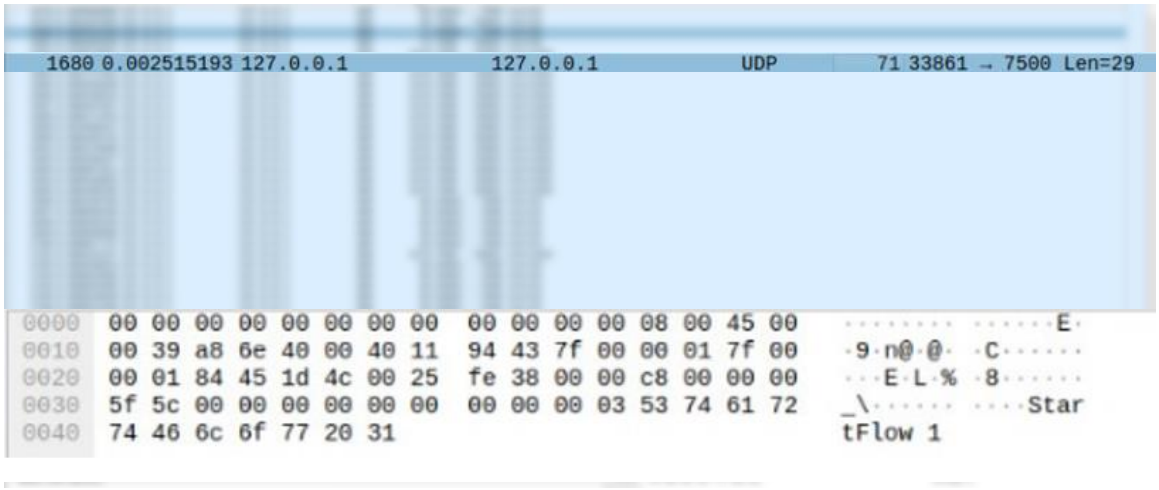
צילומי wireshark והסברים:

יצירת קשר עם השרת ע"י שליחת Hello:

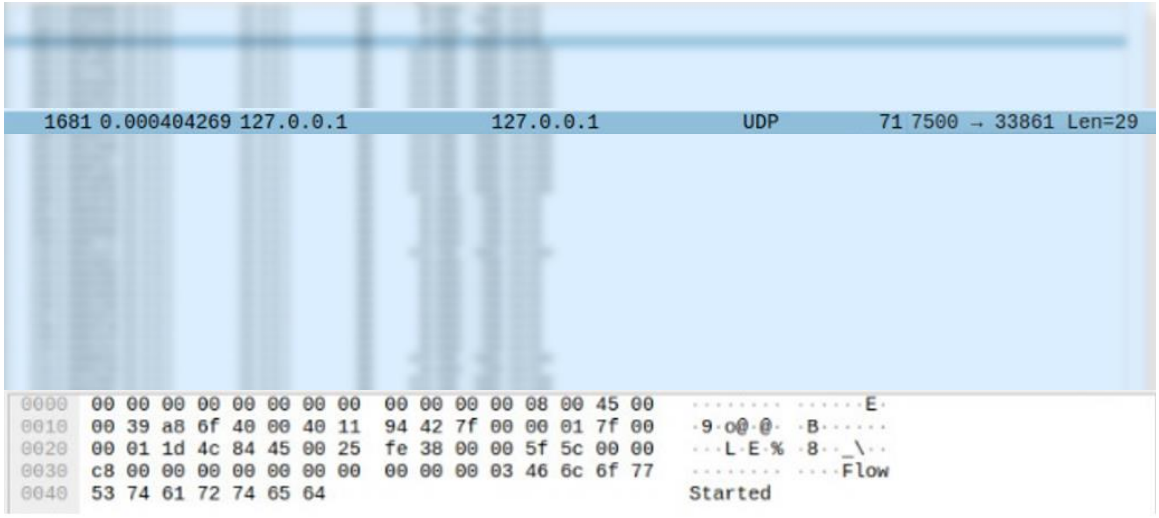


השרת שולח Hello בחזרה ללקוח:

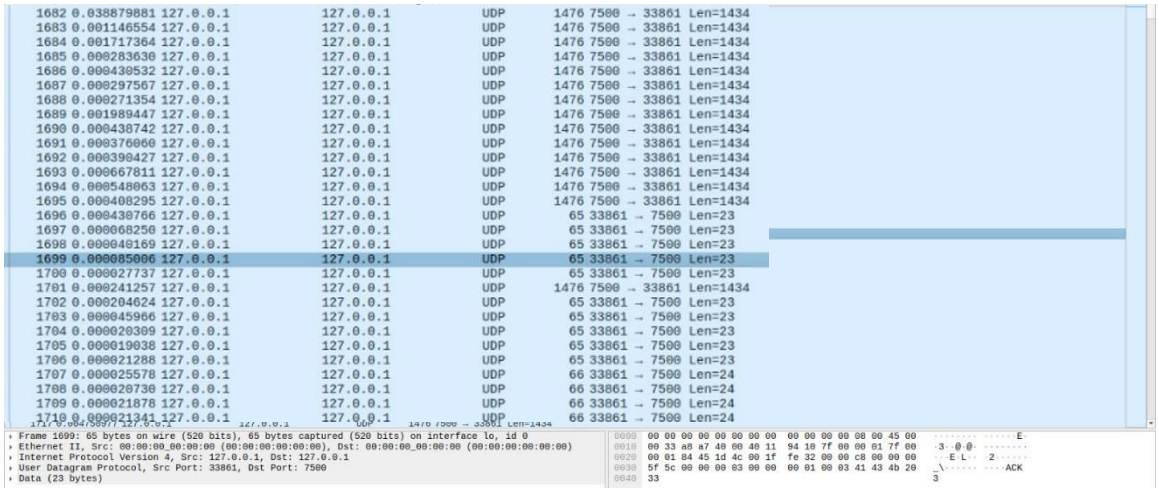
הלקוח מבקש מהסרבר להתחיל בזרימה של שליחת קובץ בודד לזרימה מס 1 ("Start flow1"):



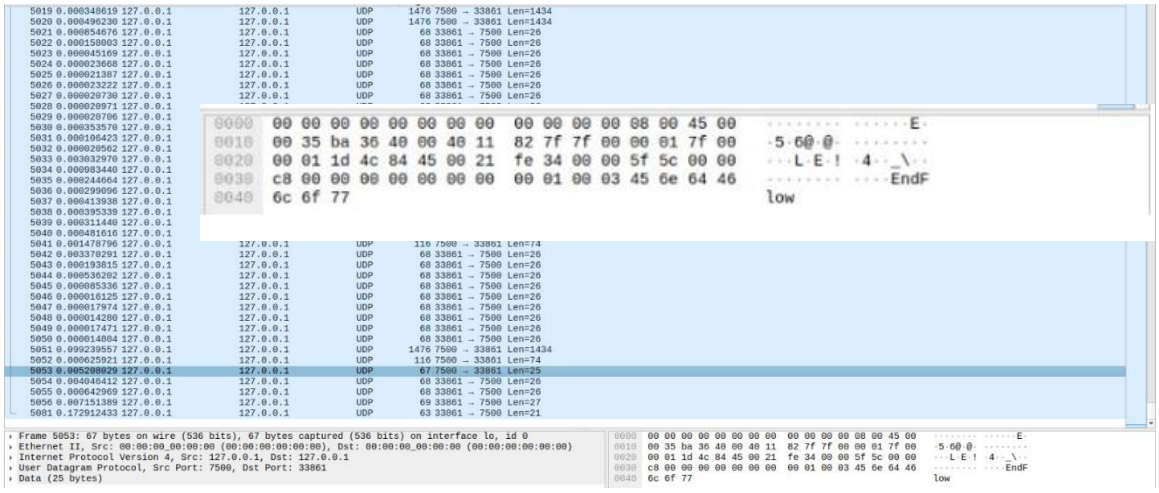
השרת מחזיר שישלח קובץ אחד (בזרימה אחת) "Flow Started":



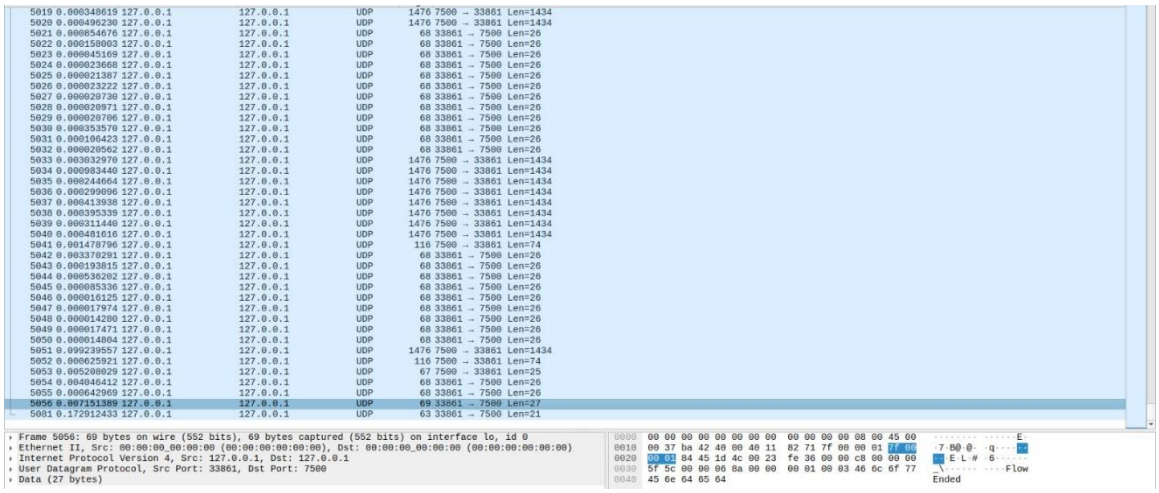
נשלחות 14 חבילות שעבור כל אחד יש ACK מהלקוח עבור כל חבילה, לדוגמה בתצלום הבא הלקוח נותן חיווי שחבילה מס' 3 הגיעה (3 ACK):



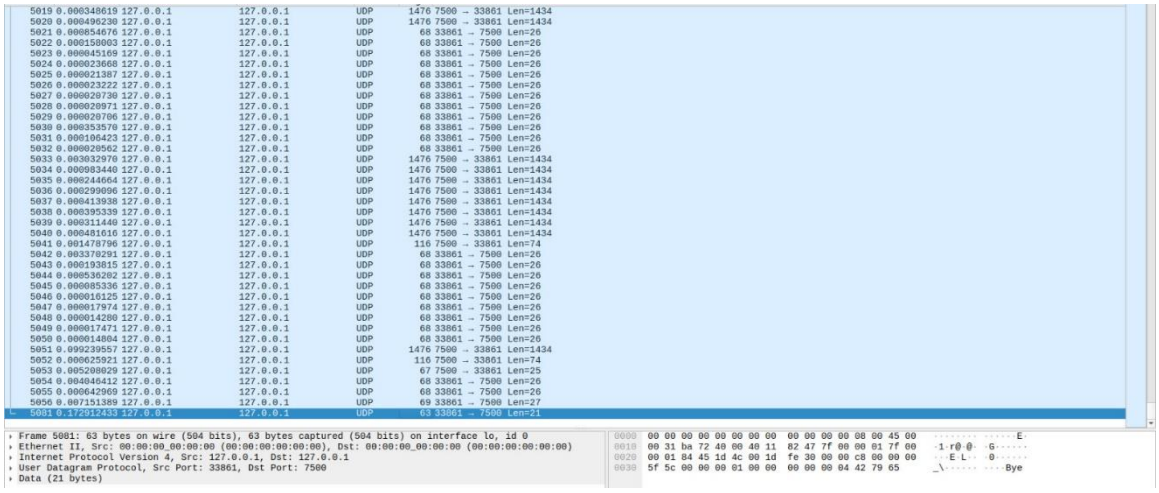
השרת שולח ללקוח שהוא סיים לשלוח את הקובץ:



הלקוח נותן חיווי לשרת שהוא קיבל את סיום השליחה:

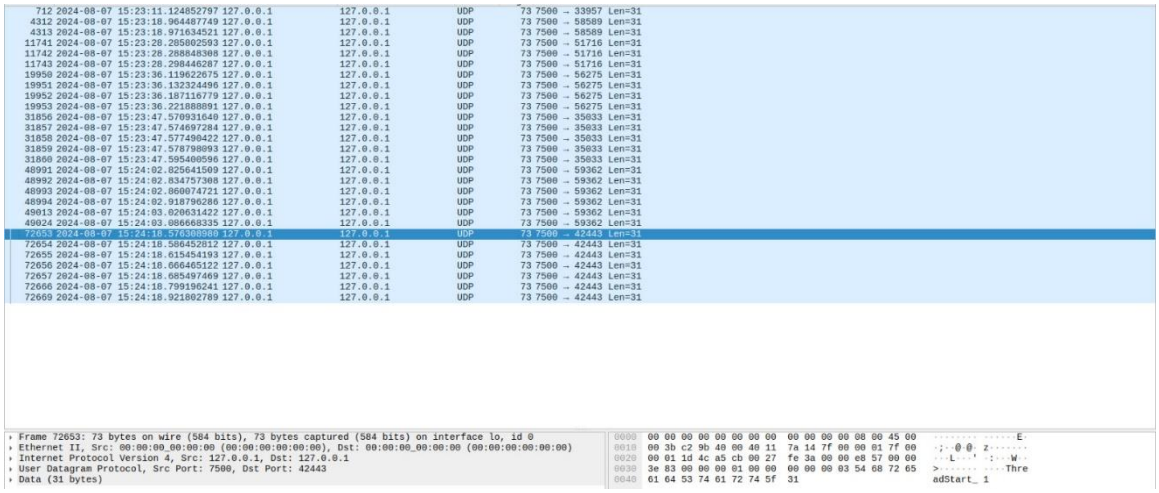


הלקוח מודיע על סיום הקשר ושולח "Bye":



המחשה למקביליות השליחה:

להלן השרת אומר ללקוח שקובץ ישלח בThread:

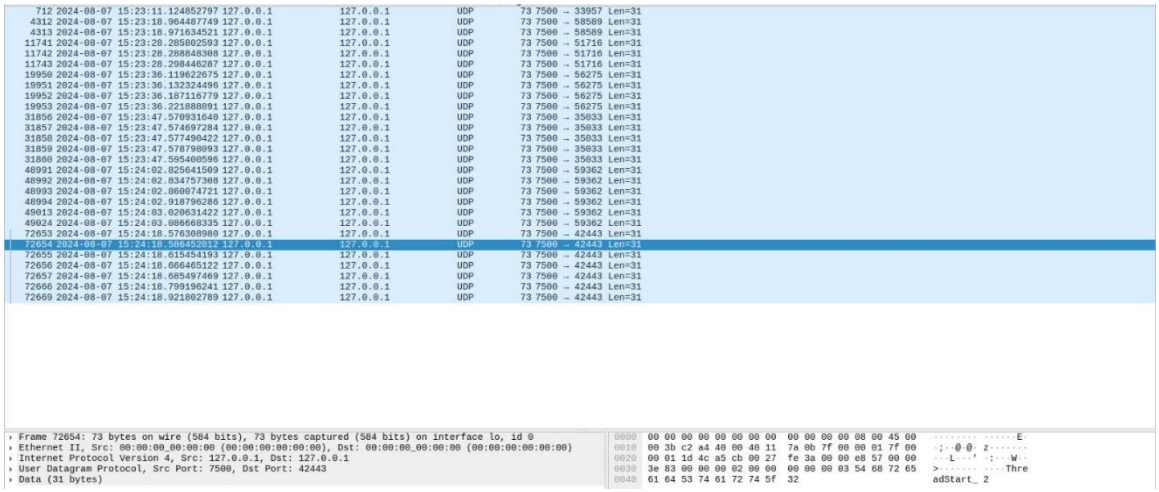


712	2024-08-07 15:23:11.124852797	127.0.0.1	127.0.0.1	UDP	73 7500 → 33957 Len=31
4312	2024-08-07 15:23:18.964487749	127.0.0.1	127.0.0.1	UDP	73 7500 → 58589 Len=31
4313	2024-08-07 15:23:18.971634521	127.0.0.1	127.0.0.1	UDP	73 7500 → 58589 Len=31
11741	2024-08-07 15:23:28.285802593	127.0.0.1	127.0.0.1	UDP	73 7500 → 51716 Len=31
11742	2024-08-07 15:23:28.288848308	127.0.0.1	127.0.0.1	UDP	73 7500 → 51716 Len=31
11743	2024-08-07 15:23:28.298446287	127.0.0.1	127.0.0.1	UDP	73 7500 → 51716 Len=31
19950	2024-08-07 15:23:36.119622675	127.0.0.1	127.0.0.1	UDP	73 7500 → 56275 Len=31
19951	2024-08-07 15:23:36.132324496	127.0.0.1	127.0.0.1	UDP	73 7500 → 56275 Len=31
19952	2024-08-07 15:23:36.187116779	127.0.0.1	127.0.0.1	UDP	73 7500 → 56275 Len=31
19953	2024-08-07 15:23:36.221888891	127.0.0.1	127.0.0.1	UDP	73 7500 → 56275 Len=31
31856	2024-08-07 15:23:47.570931640	127.0.0.1	127.0.0.1	UDP	73 7500 → 35033 Len=31
31857	2024-08-07 15:23:47.574697284	127.0.0.1	127.0.0.1	UDP	73 7500 → 35033 Len=31
31858	2024-08-07 15:23:47.577490422	127.0.0.1	127.0.0.1	UDP	73 7500 → 35033 Len=31
31859	2024-08-07 15:23:47.578798093	127.0.0.1	127.0.0.1	UDP	73 7500 → 35033 Len=31
31860	2024-08-07 15:23:47.595400596	127.0.0.1	127.0.0.1	UDP	73 7500 → 35033 Len=31
48991	2024-08-07 15:24:02.825641509	127.0.0.1	127.0.0.1	UDP	73 7500 → 59362 Len=31
48992	2024-08-07 15:24:02.834757308	127.0.0.1	127.0.0.1	UDP	73 7500 → 59362 Len=31
48993	2024-08-07 15:24:02.860074721	127.0.0.1	127.0.0.1	UDP	73 7500 → 59362 Len=31
48994	2024-08-07 15:24:02.918796286	127.0.0.1	127.0.0.1	UDP	73 7500 → 59362 Len=31
49013	2024-08-07 15:24:03.020631422	127.0.0.1	127.0.0.1	UDP	73 7500 → 59362 Len=31
49024	2024-08-07 15:24:03.086668335	127.0.0.1	127.0.0.1	UDP	73 7500 → 59362 Len=31
72653	2024-08-07 15:24:18.576308980	127.0.0.1	127.0.0.1	UDP	73 7500 → 42443 Len=31
72654	2024-08-07 15:24:18.586452812	127.0.0.1	127.0.0.1	UDP	73 7500 → 42443 Len=31
72655	2024-08-07 15:24:18.615454193	127.0.0.1	127.0.0.1	UDP	73 7500 → 42443 Len=31
72656	2024-08-07 15:24:18.666465122	127.0.0.1	127.0.0.1	UDP	73 7500 → 42443 Len=31
72657	2024-08-07 15:24:18.685497469	127.0.0.1	127.0.0.1	UDP	73 7500 → 42443 Len=31
72666	2024-08-07 15:24:18.799196241	127.0.0.1	127.0.0.1	UDP	73 7500 → 42443 Len=31
72669	2024-08-07 15:24:18.921802789	127.0.0.1	127.0.0.1	UDP	73 7500 → 42443 Len=31

```
▸ Frame 72653: 73 bytes on wire (584 bits), 73 bytes captured (584 bits) on interface lo, id 0
▸ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▸ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▸ User Datagram Protocol, Src Port: 7500, Dst Port: 42443
▸ Data (31 bytes)

0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ······· ····· E·
0010  00 3b c2 9b 40 00 40 11  7a 14 7f 00 00 01 7f 00   ·;··@·@· z·····
0020  00 01 1d 4c a5 cb 00 27  fe 3a 00 00 e8 57 00 00   ···L···' ·:···W··
0030  3e 83 00 00 00 01 00 00  00 00 00 03 54 68 72 65   >····· ····Thre
0040  61 64 53 74 61 72 74 5f  31                        adStart_ 1
```

ה-Thread הבא ישלח מיד לאחר מכן:



712	2024-08-07 15:23:11.124852797	127.0.0.1	127.0.0.1	UDP	73 7500 → 33957 Len=31
4312	2024-08-07 15:23:18.964487749	127.0.0.1	127.0.0.1	UDP	73 7500 → 58589 Len=31
4313	2024-08-07 15:23:18.971634521	127.0.0.1	127.0.0.1	UDP	73 7500 → 58589 Len=31
11741	2024-08-07 15:23:28.285802593	127.0.0.1	127.0.0.1	UDP	73 7500 → 51716 Len=31
11742	2024-08-07 15:23:28.288848308	127.0.0.1	127.0.0.1	UDP	73 7500 → 51716 Len=31
11743	2024-08-07 15:23:28.298446287	127.0.0.1	127.0.0.1	UDP	73 7500 → 51716 Len=31
19950	2024-08-07 15:23:36.119622675	127.0.0.1	127.0.0.1	UDP	73 7500 → 56275 Len=31
19951	2024-08-07 15:23:36.132324496	127.0.0.1	127.0.0.1	UDP	73 7500 → 56275 Len=31
19952	2024-08-07 15:23:36.187116779	127.0.0.1	127.0.0.1	UDP	73 7500 → 56275 Len=31
19953	2024-08-07 15:23:36.221888891	127.0.0.1	127.0.0.1	UDP	73 7500 → 56275 Len=31
31856	2024-08-07 15:23:47.570931640	127.0.0.1	127.0.0.1	UDP	73 7500 → 35033 Len=31
31857	2024-08-07 15:23:47.574697284	127.0.0.1	127.0.0.1	UDP	73 7500 → 35033 Len=31
31858	2024-08-07 15:23:47.577490422	127.0.0.1	127.0.0.1	UDP	73 7500 → 35033 Len=31
31859	2024-08-07 15:23:47.578798093	127.0.0.1	127.0.0.1	UDP	73 7500 → 35033 Len=31
31860	2024-08-07 15:23:47.595400596	127.0.0.1	127.0.0.1	UDP	73 7500 → 35033 Len=31
48991	2024-08-07 15:24:02.825641509	127.0.0.1	127.0.0.1	UDP	73 7500 → 59362 Len=31
48992	2024-08-07 15:24:02.834757308	127.0.0.1	127.0.0.1	UDP	73 7500 → 59362 Len=31
48993	2024-08-07 15:24:02.860074721	127.0.0.1	127.0.0.1	UDP	73 7500 → 59362 Len=31
48994	2024-08-07 15:24:02.918796286	127.0.0.1	127.0.0.1	UDP	73 7500 → 59362 Len=31
49013	2024-08-07 15:24:03.020631422	127.0.0.1	127.0.0.1	UDP	73 7500 → 59362 Len=31
49024	2024-08-07 15:24:03.086668335	127.0.0.1	127.0.0.1	UDP	73 7500 → 59362 Len=31
72653	2024-08-07 15:24:18.576308980	127.0.0.1	127.0.0.1	UDP	73 7500 → 42443 Len=31
72654	2024-08-07 15:24:18.586452812	127.0.0.1	127.0.0.1	UDP	73 7500 → 42443 Len=31
72655	2024-08-07 15:24:18.615454193	127.0.0.1	127.0.0.1	UDP	73 7500 → 42443 Len=31
72656	2024-08-07 15:24:18.666465122	127.0.0.1	127.0.0.1	UDP	73 7500 → 42443 Len=31
72657	2024-08-07 15:24:18.685497469	127.0.0.1	127.0.0.1	UDP	73 7500 → 42443 Len=31
72666	2024-08-07 15:24:18.799196241	127.0.0.1	127.0.0.1	UDP	73 7500 → 42443 Len=31
72669	2024-08-07 15:24:18.921802789	127.0.0.1	127.0.0.1	UDP	73 7500 → 42443 Len=31

```
▸ Frame 72654: 73 bytes on wire (584 bits), 73 bytes captured (584 bits) on interface lo, id 0
▸ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▸ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▸ User Datagram Protocol, Src Port: 7500, Dst Port: 42443
▸ Data (31 bytes)

0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ······· ····· E·
0010  00 3b c2 a4 40 00 40 11  7a 0b 7f 00 00 01 7f 00   ·;··@·@· z·····
0020  00 01 1d 4c a5 cb 00 27  fe 3a 00 00 e8 57 00 00   ···L···' ·:···W··
0030  3e 83 00 00 00 02 00 00  00 00 00 03 54 68 72 65   >····· ····Thre
0040  61 64 53 74 61 72 74 5f  32                        adStart_ 2
```