"מבט מהיר על QUIC"

כללי:

QUIC הוא הפיתוח האחרון בפרוטוקולי תקשורת, עם הפוטנציאל להחליף את פרוטוקול TCP בהמשך הזמן. במקום לתאר את אופן הפעולה של QUIC צעד-צעד, מאמר זה מכוון להסביר את QUIC מהעקרונות שהעיצוב שלו מבוסס עליהם.
נתאר קודם את TCP, פרוטוקול התקשרות הראשון שהופץ וזה שהכי נפוץ בשימוש עד כה, כדי להסביר את הפעולות הבסיסיות שפרוטוקול תקשורת מבצע ואת הבעיות העולות מהפעולה של TCP. נדון גם בתובנות העולות מהעיצוב של כמה פרוטוקולי תקשורת אחרים.
לאחר מכן, נתאר לפרטים את העיצוב שך QUIC, תוך התמקדות ביסודות של העקרונות הכי חשובים שלו:

1) השילוב של יצירת חיבור ואבטחה,
2) שימוש במזהי חיבור כתובת IP עצמאיים ומצב חיבור מתמיד,
3) מנגנוני חלון נפרדים לבקרת עומס ושליחה אמינה של מידע,
4) חיבור קרוב לאפליקציות,

והדגמה איך תכונות העיצוב החדשות האלה עונות על הבעיות המזוהות בפרוטוקול תקשורת קיימים.

1 מבוא:

QUIC הוא פרוטוקול התקשורת העדכני הנקלט במהירות. יש שצופים שהוא יחליף את TCP עם הזמן [13]. למרבה הצער, ההגדרות של QUIC לא רק ארוכות, הן גם מתמקדות בעיקר בתיאור הפעולות של הפרוטוקול, בעוד ההסברים *למה* קבור עמוק בפרטים של הפרוטוקול או פשוט חסרים. עבור הרבה אנשים שרוצים ללמוד על QUIC במהירות, צלילה ישירות לתוך הפרטים של QUIC לא נראית כמו דרך יעילה להשגת המטרה. המסמך הזה מכוון להציע מבט מובן ומלא תובנות לתוך QUIC.

QUIC נראה שונה מאד מ-TCP שאנשים מכירים, אבל מה הם ההבדלים בדיוק? ויותר חשוב, *למה* ההבדלים האלה? מאיפה הרעיונות האלה הגיעו? למרות שהעיצוב ופיתוח של QUIC עדיין בתהליכי התקדמות, ולכן מאפיינים ספציפיים שונים עלולים להשתנות בעתיד. בכל זאת, אנחנו מצפים שהחלקים הבסיסיים במאמר הזה יישארו עם הפרוטוקול.

נתחיל את ההסבר קודם על ידי סקירה של TCP, הפעולות הבסיסיות שלו, בעיות שזוהו, והמסקנות שנלמדו לאורך שנים של עיצוב פרוטוקולי תקשורת (2). לאחר מכן, נעבור לסקירה של פרוטוקול QUIC בעצמו, בהפרדה ל:

1) חיבורי QUIC; (3)
2) פקטות (חבילות) QUIC; (4)
3) שחזור QUIC; (5)
4) אבטחת QUIC; (6)
5) ואפליקציות מעל QUIC. (7)

בסוף, נתאר את האתגרים הנותרים שעומדים בפני QUIC בפרט ופרוטוקולי תקשורת עתידיים בכלל. (8)

2 מבוא לפרוטוקולי תעבורה:

בחלק הזה, נשתמש בTCP כדוגמה כדי לזהות את הקבוצה של פעולות בסיסיות שפרוטוקול תעבורה צריך לתמוך בהם. אנשים שמכירים TCP יכולים לדלג לחלק 2.3, שמתאר כמה פרוטוקולי תעבורה אחרים שהעיצוב של QUIC למד מהם.

2.1 TCP:

כפרוטוקול תעבורה, TCP מבצע את 3 הפעולות הבסיסיות הבאות:

1) פענוח (Demultiplexing) של מידע תוך שימוש במספרי פורטים;
2) שליחה אמינה של זרם בתים (Byte stream) עם בקרת זרימה מבוססת חלון[1];
3) בקרת עומס על מנת לשלו על מספר החבילות בתוך הרשת.

ראשית, הפרוטוקול TCP/IP מגשר על הפער בין שמות אפליקציות/תהליכים בעלי משמעות סמנטית והפרוטוקולים ברמה נמוכה יותר, על ידי שימוש במספרי פורטים. כל אפליקציה או תהליך סטנדרטי מקבל מספר פורט תעבורה מסוים [5, 12]. כל פרוטוקולי התעבורה הקיימים משתמשים בשילוב של מספרי פורט מוצא ויעד כדי לשלוח חבילות נכנסות לתהליכים הנכונים.

שנית, כדי לספק שירות העברת מידע אמין בין שתי נקודות קצה, כל חלק מידע (במקרה של TCP, כל בית מידע) צריך לקבל מזהה ייחודי, שמאפשר לשני הקצוות להבין אם כל החלקים נשלחו. כל פרוטוקולי התעבורה משיגים את המטרה הזו ע"י קודם נתינת מזהה חיבור ייחודי, ובתוך החיבור כל יחידת מידע מקבלת מזהה ייחודי שהוא מספר סדרתי עולה מונוטונית באופן כללי. עם מספרים סדרתיים מונוטוניים עולים, המקבל יכול ליידע את השולח לגבי כל המידע שהתקבל עד מספר מסוים n ע"י אישור מסכם כללי, ACK(n).
TCP משתמש בשילוב של כתובות הIP ומספרי הפורט של שני הקצוות כדי לייצר מזהה חיבור ייחודי באופן גלובלי, ותהליך יצירת חיבור אמין כדי לאפשר לשני הצדדים להסכים על מספר סדרה התחלתי להשתמש בו בהתחלת התקשורת. TCP גם משתמש בשיטה דומה בפירוק חיבור כדי לתת לכל צד ליידע את השני לגבי המספר הסדרתי הסופי שמזהה את בית המידע האחרון שהוא שולח. גם התהליך של יצירת החיבור וגם של פירוק החיבור מבצעים לחיצת ידיים משולשת סטנדרטית.
ליצירת החיבור, כדי למזער את הסיכוי שמספר סדרתי בחיבור החדר יתנגש עם מספר סדרתי מחיבורים ישנים[2], TCP נותן לכל אחד מהצדדים לבחור מספר אקראי בתור המספר הסתדרתי ההתחלתי שלו. במקרה הזה, הצד המתחיל (לקוח) יכול להתחיל לשלוח מידע עם ההודעה השלישית של לחיצת הידיים, ולכן יש עיכוב של RTT אחד כדי ליידע אחד את השני לגבי המספרים הסדרתיים ההתחלתיים.

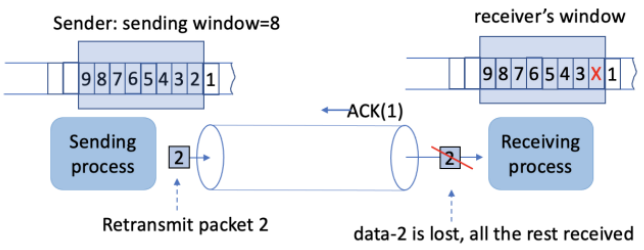
שלישית, המטרה של בקרת עומס היא להגביל את מספר החבילות *בתוך* הרשת. TCP לא ייועד לבצע בקרת עומס בהגדרתו [1], המימוש של TCP שונה באמצע שנות ה-80 כדי להוסיף בקרת עומס מעל המנגנון בקרת זרימה (מבוסס חלון) שהיה לו. בקרת עומס של TCP נותנת לשולח לשמר *חלון בקרת עומס*, ולשלוח מידע בהתאם למינימום מתוך שני החלונות.

---

[1] על מנת להבטיח שירות שליחה אמינה של מידע גנרי לאפליקציות שונות, TCP משתמש ב*בית (Byte)* בתור יחידת המידה הבסיסית שלו, כדי להבטיח שליחה לפי הסדר של הזרם בתים.
[2] כדי למנוע מצב שבו מידע מחיבור קודם נחשב בטעות למידע מהחיבור הנוכחי, למקרה שבו לשני החיבורים יש כתובת IP ומספר פורט זהים.

2.2 הבעיות של TCP:

הבעיות המרכזיות בשימוש בTCP שזוהו לאורך השנים:



איור 1: כל החבילות יצאו מהרשת; האובדן של חבילה 2 מונע מחלון הזרימה להתקדם

1) השילוב של בקרת עומס ושליחה אמינה:
המטרה של בקרת עומס היא לשלוט במספר החבילות *בתוך הרשת*, והיא נוספה למנגנון שייועד במקור לאפשר למקבל לבצע בקרת זרימה על השולח, כדי להבטיח שליחה אמינה. לכן, כל אובדן חבילות יגביל את התקדמות החלון עד שמשלימים את האבידה; בזמן של גילוי אבידה ושליחה מחדש, יכול להיות שרוב (אם לא כל) החבילות בתוך החלון כבר התקבלו, כלומר יצאו מהרשת. כמו שמתואר באיור 1, כשהמקבל קבע את גודל החלון ל-8 חבילות. ניתן להניח שחלון בקרת העומס הוא גם 8 חבילות. בעוד ההגעה של חבילה 1 מאפשרת לחלון להתקדם לחבילות 2-9, האובדן של חבילה 2 מונע מהחלון להתקדם גם כאשר אין חבילות בתוך הרשת. לכן, מספר החבילות שטרם אושרו לא מייצג את מספר החבילות בתוך הרשת. מספר תיקונים של "ניפוח וצמצום" חלון פותחו כדי להקל על הבעיה, עם תוספת של סיבוך ותועלת מוגבלת [3].

איור 2: חסימת ראש תור ב-TCP

2) חסימת ראש תור – Head of line blocking:
בגלל שTCP מבטיח שליחת זרם בתים לפי הסדר, האובדן של חבילה אחת יחסום את שליחת כל החבילות העוקבות עד שהחבילה האבודה תתקבל. איור 2 מראה דוגמה שבה חבילה 2 נשלחה אבל חבילה 1 נאבדה בדרך. המידע בחבילה 2 לא יכול להגיע לאפליקציה עד שהמידע של חבילה 1 נשלח מחדש ומתקבל, ומועבר לאפליקציה קודם, כדי להבטיח שליחה לפי הסדר של כל המידע. איורים 1 ו-2 נראים דומים, כאשר הראשון משקף בעיה אצל השולח (שנמנע מלשלוח עוד חבילות) והשני בעיה אצל המקבל (מידע שכבר התקבל נחסם בkernel).

3) עיכוב בגלל בניית חיבור:
כל פעם שקצה אחד רוצה לתקשר עם השני בעזרת TCP, לחיצת-יד משולשת נדרשת כדי לבנות חיבור TCP לפני שמידע יכול להישלח. ואם רוצים לאבטח את החיבור בעזרת TLS, נדרש עוד הלוך-חזור כדי ששני הקצוות יחליפו אישורי אבטחה.

4) הגבלות בגלל כותרת באורך מקובע:
הכותרת של TCP היא אוסף של שדות בגודל קבוע. קיים שדה אפשרויות, שמוגבל ל-40 בתים לכל היותר. בנוסף לנשיאת המידע בין שני הקצוות, פרוטוקול TCP גם מאגד את כל פעולות השליטה לתוך אותה כותרת של 20 בתים: בניית החיבור, פירוק, ואיפוס; ואישור מידע. כאשר מזוהים עוד פעולות – לדוגמה ACK סלקטיבי – הם נוספים לתוך שדה האפשרויות שהוא בעצמו מוגבל ל-40 בתים לכל היותר.
שלושה שדות ספציפיים בכותרת של TCP הושפעו ישירות מהעלייה המתמדת של מהירות הרשת במשך הזמן. המספרים הסדרתיים ומספרי האישור שניהם באורך 4 בתים; גודל חלון בקרת הזרימה הוא באורך 2 בתים בלבד. השדות בעלי הגודל הקטן והקבוע מגבילים את הביצועים של TCP ברשתות בעלי מהירות גבוהה: השניים הראשונים מתאפסים מהר מדי, כך שהם כבר לא ייחודיים; וגודל קטן של חלון בקרת זרימה מגביל ישירות את התפוקה של חיבור ה-TCP, שחסום מלמעלה על ידי הנוסחה: (גודל חלון)X(RTT). גם פה, נעשה שימוש בשדה האפשרויות כדי להגדיל את האורכים שלהם.

5) מזהה חיבור ייחודי וכתובות IP:
כתובת ה-IP של כל אחד מהקצוות בחיבור TCP עלול להשתנות בזמן החיבור, בגלל מגוון סיבות: רב-גישות של שרת (multihoming), ניידות, או הרצה מאחורי NAT. בגלל שTCP משתמש בשילוב של כתובות ה-IP ומספרי פורט של שני הקצוות בשביל מזהה החיבור, כל שינוי בכתובת ה-IP של אחד הצדדים ינתק את החיבור הקיים, וכל המידע ששותף עד אותו זמן ייזרק. כדי לשחזר את החיבור שנכשל, צריך לבצע לחיצת ידיים חדשה.

2.3 פרוטוקולי תקשורת אחרים:

אמנם TCP הוא הפרוטוקול השולט בינתיים, נעשה שימוש בעוד כמה פרוטוקולים אחרים, שכל אחד מהם ממלא איזשהו צורך שחסר בTCP.

2.3.1 [4] T/TCP – Transaction TCP: פותח כדי לתמוך באפליקציות שצריכות לבצע התקשרות בזמן אמת, בלי לשלם את המחיר של עיכוב ותקורה מתהליך בניית החיבור של TCP. T/TCP שומר את מצב החיבור אחרי שהוא נוצר, בפרט את מזהה החיבור עם המספר הסידורי המשויך בשני הצדדים. שמירת המידע הזה דורשת עוד זיכרון אבל נמנעת מלחיצת ידיים משולשת להעברות קצרות אחר כך, שיכול לעבור זמן רב ביניהן בלי תקשורת.

2.3.2 SCTP – Stream Control Transmission Protocol: [18] הקדים את QUIC ביותר מ-10 שנים. העיצוב שלו עונה על 3 מהבעיות שזוהו ב-TCP.
עבור בעיה #2 – HLB, הפרוטוקול מאפשר לכל חיבור להכיל תתי-זרמים של מידע, ובכך מונע את הבעיה בין זרמים שונים. חסימה בזרם אחד לא תשפיע על השני, ובכך הבעיה נתחמת בתוך כל זרם בנפרד, שעדיין נשלח לפי הסדר. למרות זאת, SCTP לא עונה על הבעיה בצורה מלאה; כל חיבור SCTP משתמש ב-TSN (transmission sequence number – מספר שידור סידורי) יחיד עבור שליחה אמינה של מידע ועבור בקרת עומס, באותה דרך כמו TCP. לכן, כאשר חלון ה-TSN חסום על ידי חבילות אבודות, השולח לא יכול לשלוח מידע חדש באף זרם, בעיה דומה ל-TCP.
עבור בעיה #4 – כותרת קבועה, SCTP מגדיר *סוגי גושים*, כל גוש מוגדר על ידי כותרת בפורמט משלו, וגושים שונים יכולים להישלח לתוך חבילה אחת (במגבלות ה-MTU). כל פקודה של ניהול החיבור (בנייה, פירוק, איפוס) מוגדרת בתור גוש שליטה נפרד, וגם ה-ACK הסלקטיבי. המידע של האפליקציה נשלח בגושי מידע בתור ADU-ים, כל אחד מזוהה על ידי [מזהה זרם, מספר סידורי בזרם][3]. העיצוב הזה מאפשר ל-SCTP גמישות בהגדרת פקודות שליטה חדשות פשוט על ידי הוספת סוג גוש חדש.
SCTP עונה על בעיה #5 – שינוי כתובות IP – באופן חלקי. כל קצה של החיבור יכול להכיל מספר כתובות IP, והקבוצה יכולה להשתנות בזמן שהחיבור קיים. בכל זאת, מזהי חיבור של SCTP עדיין תלויים בכתובות ה-IP.

2.3.3 RTP – Real-time Transport Protocol: כמו שהשם שלו מרמז, RTP [9] מעוצב כדי לתמוך באפליקציות זמן אמת, כמו שיחות וידאו. אפליקציות של העברת מולטימדיה בזמן אמת דורשות העברת מידע בזמן, ולכן יכולות להכיל אבדן חבילות עד רמה מסוימת. הפרוטוקול גם מספק שירותים של multicast package delivery, jitter compensation. בגלל ההבדלים בינו לבין פרוטוקולים קודמים – הם מתמקדים בשליחה *אמינה* של מידע, ולא במהירות – RTP פיתח רעיונות חדשים בעיצוב של פרוטוקולי תעבורה.
ראשית, RTP הוא הפרוטוקול הראשון שבשימוש נרחב שרץ על UDP, והתוצאה של זה היא שהמימוש של RTP הוא *מחוץ ל-kernel*. אפשר לראות את UDP [16] כפרוטוקול NO-OP בנפרד משימוש במספרי פורטים כדי לקבל ולמיין חבילות נכנסות לאפליקציות היעד (demultiplexing), ולקבל checksum אופטימלי. אבל, UDP כן מספק יתרון שבדרך כלל לא שמים לב אליו: הוא מאפשר לאפליקציות איזה מידע נכנס לכל חבילת IP, ובכך מאפשר מימוש של application data unit (ADU) [7]. זה מאפשר לRTP להיות קשור למימוש של האפליקציה כדי להשתמש בADU.
שנית, RTP הוא פרוטוקול התקשורת הראשון שמשתמש ב- IP multicast delivery. פעילות של multicast RTP מזוהה לפי כתובת ה- IP multicast בתוספת זוג פורטים של UDP – אחד ל-RTP, אחד ל-RTCP.
בנוסף, חבילות המידע של RTP נושאות חותמות זמן ומספרים סדרתיים, בשביל סדר וזיהוי אבידות.

---

[3] נשים לב שכל מספר סידורי בזרם מזהה גוש מידע שניתן על ידי האפליקציה, שגודלו יכול להיות יותר מה-MTU של הרשת. לכן, SCTP צריך לתמוך בפירוק והרכבה של הגושים, בניגוד למודל הבתים של TCP, שבו המספר הסידורי של כל בית מאפשר לפרק את המידע של האפליקציה לפי כל חלוקה של בתים.

2.4 אבטחת פרוטוקולי תקשורת: TLS ו- DTLS.

אפליקציות אינטרנט צריכות אבטחה קריפטוגרפית, שבדרך כלל נוסף מעל פרוטוקול תעבורה קיים.
היתרון המרכזי של Transport Layer Security – TLS הוא שהוא מספק ערוץ תקשורת שקוף. כך, נוח לאבטח פרוטוקול של אפליקציה על ידי זה שנכניס TLS בין שכבת האפליקציה לשכבת התעבורה, בדרך כלל TCP. TLS מצפין את העברות המידע של חיבור הTCP, ונותן אימות ואבטחה בין הצדדים המתקשרים [17]. TLS רץ במרחב המשתמש ומסתמך על TCP בשביל שליחה אמינה של יחידות המידע שלו שהוא משתמש בהן להצפנת המידע. יש צורך בתהליך לחיצת ידיים כדי לקבוע פרמטרים נחוצים להצפנה לפני שאפשר לשלוח מידע בחיבור הTCP.
פרוטוקול DTLS – Datagram Transport Layer Security [8] מעוצב לאבטח אפליקציות שמשתמשות בUDP.
פילוסופיית העיצוב הבסיסית של DTLS היא לבנות "TLS מעל תמיכה בdatagram". TLS דורש שליחה אמינה של חבילות ואי אפשר להשתמש בו ישירות בסביבה של UDP שבה חבילות יכולות ללכת לאיבוד. DTLS עושה רק שינויים קלים בTLS כדי לתקן את הבעיה הזאת. DTLS משתמש בטיימר שידור פשוט כדי להתמודד עם אובדן חבילות, מספר סדרתי כדי להתמודד עם סידור מחדש, ומבצע פירוק והרכבה של המידע לפי הצורך. במהות, DTLS מבצע את כל המשימות של TLS עם הבטחת אמינות דומה ל-TCP. כתוצאה מכך, שימוש בDTLS כדי לאבטח datagrams של אפליקציות דורש הרבה הלוך-חזור.

2.5 סיכום: פעולות פרוטוקול תעבורה על IP:

בהתבסוסת על התבוננות בעיצובי פרוטוקולי תעבורה והשימוש שלהם בעשורים האחרונים, נעשה סיכום של הפעולות הבסיסיות שפרוטוקול תעבורה שרץ מעל IP צריך לספק, ושאלות העיצוב הנלוות. נשים לב שכל פרוטוקולי התעבורה הקיימים משתמשים במספרי פורטים בשביל demultiplexing בתוך הhost בצורה ישירה, ולכן לא נזכיר אותו ברשימה הבאה:

1. **הגדרת מזהה חיבור ומזהה מידע.**
2. **ניהול חיבור תעבורה:**
   - **מזהה חיבור ייחודי באופן גלובלי** שמקשר את שני צידי החיבור, ורוצה להיות עצמאי מIP אבל צריך להיות ממופה בצורה אמינה לכתובת IP.
   - ייצור ופירוק של מצב החיבור.
   - **שליטה על החלפת מידע** בין שני הקצוות; רוצים גמישות בהגדרת הודעות שליטה חדשות.
   - תמיכה בשינוי של כתובת IP של הhost.
3. **שליחת מידע בצורה אמינה:**
   - **מזהה מידע ייחודי** שצריך להיות משותף בצורה אמינה עם הצד השני.
   - **בקרת זרימה מבוססת חלון לשליחה אמינה**, כדי להימנע מחסימת ראש תור.
4. **בקרת עומס:**[4] כדי לשלוט על מספר החבילות בתוך הרשת.
5. **אבטחה.** ליתר דיוק, אנשים רואים חיבורים מאובטחים בתור *אבטחת רשת*, למרות שערוצים עם אבטחת TLL נותנים רק הסתרת מידע. אימות של גופים חיצוניים מנוהל דרך גופי אישורים צד שלישי (Certificate Authorities – CAs).

יצירת ואבטחת חיבור היו שלבים נפרדים; לדוגמה, נייצר קודם חיבור TCP כדי לאפשר העברת מידע בצורה אמינה, ואז נבנה אסוציאציית TLS מעליו. DTLS מאחד את השניים.
בנוסף, בדרך כלל פרוטוקולי תקשורת היו ממומשים בתוך הkernel, וסיבה אחת היא ליעילות של ביצועים. הצד השני הוא שיש פחות שליטה על הדרך שבה אורזים את המידע וקושי בשינוי דברים בפרוטוקול. בדרך שבה RTP רץ מעל UDP עושה את זה מחוץ לkernel ותומך בADU. כמו שנראה, QUIC משתמש באותה שיטה.

---

[4] אידיאלית, בקרת העומס צריכה להיות פונקציה של שכבת הnetwork. זה נחת על הTCP בגלל ש-IP הוא open-loop ולכן אין לו דרך אפקטיבית לשלוט על שליחת חבילות.

3 חיבורי QUIC:

QUIC הוא פרוטוקול תעבורה שפותח ע"י גוגל כדי לשפר ביצועים לתקשורת מוצפנת ולאפשר ייצוא מהיר ולתמוך בהתפתחות מהירה של מנגנוני תעבורה. [6] QUIC משתמש בUDP כפרוטוקול מאחורי הקלעים, והמימוש שלו רץ במרחב המשתמש. ככה שנוח לעדכן אותו בעתיד בלי צורך לשנות לעדכון של הkernel. כלקח מעיצובים קודמים של פרוטוקולים, QUIC עונה על הבעיות שיש בTCP שהזכרנו ב-2.2. חיבור QUIC הוא מצב משותף בין שרת ולקוח, שתמיד מתחיל עם לחיצת יד שבה שני הצדדים קובעים פרמטרים לחיבור.

3.1 מזהה חיבור:

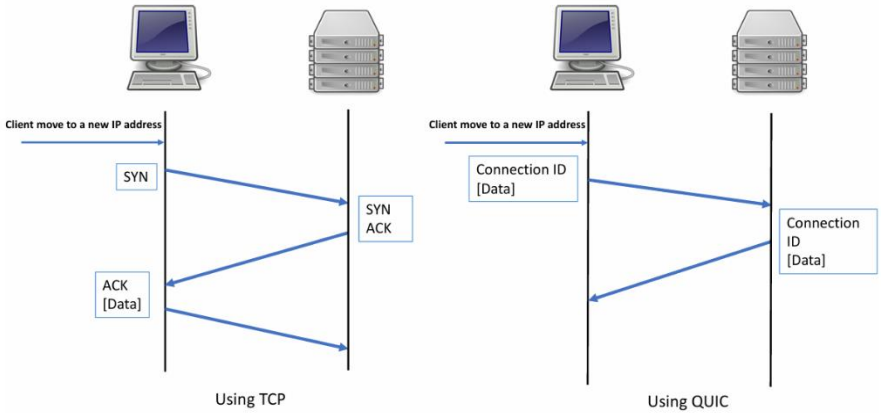
פרוטוקול QUIC משתמש בשילוב של שני מספרים, אחד מכל צד, כדי לייצר זוג של מזהי חיבור. ה- Connection ID (CID) משמש כמזהה ייחודי לחיבור, כדי להבטיח ששינויים ברמות נמוכות יותר בפרוטוקול לא יגרמו לחבילות להישלח ליעד שגוי. בכך QUIC מאפשר יכולת demultiplexing דומה ל-TCP.

3.2 לחיצת יד של QUIC:

QUIC משלב לחיצות יד של תעבורה והצפנה ביחד, ומקבל את המידע הנחוץ לשניהם ב- RTT אחד. בפרט, זה דורש ביצוע של החלפת מפתחות TLS ביחד עם החלפת פרמטרים של תעבורה באותו זמן. זה ממזער את ההשהיה שנצרכת כדי לבנות חיבור מאובטח. בעוד TCP רגיל מפריד בין החלפות המידע של הפרמטרים של אבטחה ותעבורה, ודורש 2 RTT כדי לייצר חיבור מאובטח.
QUIC משתמש בחבילה הראשונה כדי לקבוע את מזהי החיבור בשביל החיבור החדש. כל צד נותן ערך שהוא בוחר לשדה ה- Source Connection ID בחבילה הראשונה, והצד השני משתמש בID הזה כדי לקבוע לאן לשלוח חבילות בעתיד. כשהשרת מקבל את החבילה הראשונה, הוא יכול לבחור לאמת את הכתובת של הלקוח על ידי שליחת חבילת retry שיש בה token רנדומלי, שהלקוח צריך להחזיר בחבילה ראשונה חדשה כדי להמשיך את תהליך לחיצת הידיים. הודעות לחיצת ידיים של TLS גם נמצאים בחבילות הראשונות האלה, שמאפשר יצירת סוד משותף כדי לספק אימות חבילות עתידיות ב-1 RTT. מזהי החיבור שנבחרים יכללו בפרמטרי התעבורה של QUIC, שיאומתו בזמן לחיצת הידיים של TLS. בזמן השיחות על מזהי החיבור, QUIC תומך ביצירה אמינה של חיבורים בדומה לTCP.
QUIC מאפשר ללקוח לשלוח מידע אפליקציה מוצפן ב-0 RTT בחבילה הראשונה על ידי שימוש בפרמטרים מחיבור קודם ומפתח TLS שהשרת שיתף קודם לכן, אבל המידע 0 RTT הזה לא מוגן בפני reply attack. בכך שהוא תומך בשליחה ב0 RTT, QUIC יכול לתמוך במצבים שבהם נדרש T/TCP. נדון בהרחבה בחלק הקריפטוגרפי של תהליך לחיצת הידיים ב-6.1.

3.3 נדידת חיבור:

בשונה מTCP שבו השילוב של כתובות ה-IP ומספרי הפורט של שני הקצוות משמשים ליצירת מזהה החיבור, חיבור QUIC יכול לשרוד שינויים בכתובות הפרוטוקולים מאחורי הקלעים בגלל שימוש במספר זהות של חיבור – connection ID. אחרי שינוי ברשת, הקצה שנדד יכול לשלוח חבילות עם מזהה החיבור הקודם עם הכתובת החדשה שלו כדי לחדש את החיבור. אחרי שהצד השני מקבל את החבילה, הוא יבצע אימות מסלול כדי לאמת את הבעלות של הצד השני על הכתובת החדשה על ידי שליחה של challenge frame – שליחה של מידע רנדומלי לכתובת החדשה והמתנה לשליחה חוזרת של המידע, ואז שני הצדדים יכולים להמשיך לשלוח מידע. כדי למנוע מצופה פסיבי להסיק את הפעילות של קצה אחד בין מסלולי רשת שונים, קצה של QUIC יכול לספק לצד השני מזהי חיבור אלטרנטיביים מראש וקצה נודד יכול להשתמש במזהי חיבור שונים בזמן שליחת מידע מכתובות שונות. QUIC גם מאפשר לשרת לקבל חיבורים באותו IP, ולבקש מהלקוח לנדוד לכתובת שרת אחרת על ידי שימוש בפרמטרי התעבורה כדי לתת את הכתובת המועדפת בזמן תהליך לחיצת הידיים. אחרי שלחיצת הידיים מאומתת, הלקוח מבצע אימות מסלול על הכתובת המועדפת של השרת ואם הוא מצליח, הוא ישלח את כל החבילות העתידיות לכתובת החדשה.

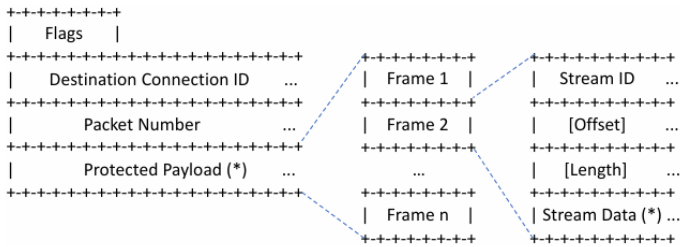


איור 3: QUIC מול TCP אחרי שינוי כתובת. בTCP, אחרי שינוי החיבור הישן נזרק ויש צורך בלחיצת ידיים חדשה (1 RTT) לפני שמידע חדש יכול להישלח. בQUIC, אפשר להשתמש שוב בחיבור הישן ואפשר לשלוח מידע מיד אם השרת זיהה את הכתובת של הלקוח בעבר (לדוגמה, אם הלקוח חוזר לכתובת ישנה).

4 חבילת QUIC:

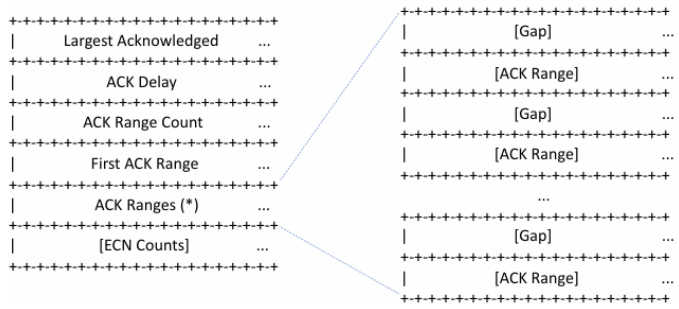
4.1 מבנה החבילה:

בשונה מTCP שבו המבנה של כותרת החבילה קבוע, ל-QUIC יש שני סוגים של כותרת. חבילות שקשורות ליצירת חיבור צריכות להכיל מגוון חלקי מידע, ומשתמשים במבנה כותרת ארוכה. אחרי שהחיבור נבנה, רק חלק מהחלקים בכותרת נחוצים ושאר החבילות משתמשות בכותרת קצרה יותר, בשביל יעילות. המבנה הקצר מודגם באיור 4. כל חבילה יכולה להכיל אחד או יותר מהמסגרות (frames) והמסגרות לא צריכות להיות מאותו סוג, כל עוד הם בטווח של הMTU.



איור 4: מבנה של כותרת קצרה. מסגרת 2 היא מסגרת סטרים שמכילה מידע אפליקציה.

לכל חבילה בQUIC יש מספר סידורי ייחודי. המספר הזה עולה באופן מונוטוני, מסמן את סדר השליחה של החבילות ולא קשור לאחזור אבידות5. כך ניתן בקלות וביעילות לדעת כמה חבילות עלולות להיות בתוך הרשת, בניגוד לבקרת עומס של TCP שחולק את אותו החלון עם בקרת הזרימה (אמינות שליחת המידע).
המקבל בQUIC מאשר את מספר החבילה הגדול ביותר שהתקבל עד כה, ביחד עם אישורים סלקטיביים (selective ACKs) – אישור של כל החבילות שהתקבלו מתחתיו, בקידוד לפי טווחי מספרים רציפים, כמו שניתן לראות באיור 5. השימוש במסגרות שמיועדות לACK יכול לתמוך בעד 256 בלוקים של אישורים במסגרת אחת, בניגוד ל3 הטווחים שיש לTCP בגלל מגבלת הגודל של שדה הoptions. זה מאפשר לQUIC לאשר חבילות שהתקבלו במסגרות ACK מרובות, שמעלה את העמידות מול סידור מחדש ואיבוד חבילות. כשחבילת QUIC מקבלת אישור, זה אומר שכל המסגרות שהגיעו באותה החבילה הגיעו.



איור 5: מבנה של מסגרת ACK ב-QUIC. השדה של Largest Acknowledged מתאר את מספר החבילה הגדול ביותר שהשולח מכיר. טווח אישורים (ACK range) הוא מספר החבילות הרציפות שאושרו לפני מספר החבילה הגדול ביותר שאושר. הפער הוא מספר החבילות שלא אושרו ברצף לפני כל טווח אישורים.

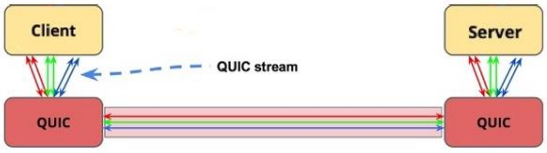
4.2 זרם – Stream:

QUIC אימץ מספר דברים ישירות מ-HTTP/2, ואחד מהם זה הכנסה של stream multiplexing לתוך שכבת התעבורה. באותה דרך שבה ב-HTTP/2 מספר זרמים יכולים להתקיים בחיבור TCP אחד, כל חיבור QUIC יכול להכיל מספר זרמים במקביל. הרעיון הזה גם לוקח השראה מהמבנה של זרמים של SST [11]. בכלל, QUIC גם אימץ את הרעיון של לפרק את המידע למסגרות ושימוש בהם כיחידות התקשורת הבסיסיות.
כל זרם ב-QUIC מזוהה על ידי מזהה זרם ייחודי, כאשר שני הביטים הכי קטנים (LSB) שלו מייצגים את הקצה שהתחיל את הזרם והאם הזרם הוא דו-כיווני. כל זרם דומה לחיבור TCP, שמספק שליחה לפי הסדר של זרם בתים (Byte stream). הזרם מחולק למסגרות, באופן דומה לסגמנטים של TCP. Stream frame offset – כמו מספרים סדרתיים של TCP, משמשים לסידור של המידע וזיהוי ושליחה מחדש של אבידות. כל מסגרת מזוהה על ידי [frame offset, stream ID]. QUIC משתמש במסגרות הSTREAM כדי לשדר מידע אפליקציה, ומספר מסגרות מזרמים שונים יכולים להיות ארוזים בתוך חבילה אחת.
הקצוות בQUIC יכולים להחליט איך לחלק את הרוחב פס בין זרמים שונים, ואיך לתת סדר עדיפויות למסגרות זרמים לפי מידע מהאפליקציה. זה מבטיח שחזור אבידות יעיל, בקרת עומס, ובקרת זרימה, שמאד משפיעים על הביצועים של האפליקציה.

---

[5] QUIC משתמש במרחבי מספרים שונים לכל רמת הצפנה (חבילות ראשונות, לחיצת יד, מידע של אפליקציה). מספרי חבילות הם ייחודיים בכל מרחב מספרים, וחבילות מקבלות אישורים במרחבים האלה. זה מאפשר הפרדה של מידע קריפטוגרפי בין מרחבי חבילות שונים.

חסימת ראש תור – Head of Line Blocking: פרוטוקול HTTP/2 ניסה להשתמש ב- stream multiplexing כדי לפתור את הבעיה (לקוח של HTTP יכול לפתוח רק מספר מסוים של חיבורי TCP לשרת במקביל, וכאשר מגיעים לגבול הזה כל בקשה חדשה צריכה לחכות עד שבקשה קודמת תסתיים). אבל, בגלל ש-HTTP/2 עושה multiplexing על חיבור TCP יחיד הוא עדיין יסבול מאותה בעיה של חסימת ראש תור ב-TCP.
מכיוון שQUIC משתמש בזרמים בלתי-תלויים מרובים, הוא נמנע מבעיית HLB שנגרמת מהמתנה לחבילות אבודות בTCP. כשחבילת נאבדת, רק הזרמים עם מסגרות שמוכלות בחבילה הזאת יצטרכו לחכות לשליחה מחדש. שאר הזרמים לא ייחסמו.



לדוגמה, באיור 6 יש 3 זרמי QUIC מסומנים באדום, ירוק, וכחול. הם מהווים חיבור יחיד. במקרה של אובדן חבילה בזרם האדום, הוא לא יחסום את השליחה בזרמים הירוק והכחול.

4.3 שליחה לא אמינה של Datagram:

אפליקציות מסוימות, בפרט כאלה שצריכות לשלוח מידע בזמן אמת, מעדיפות לשלוח מידע בלי שליחה אמינה. כיום אפשר לתמוך באפליקציות האלה על ידי שימוש בUDP, או DTLS. QUIC תומך בשליחה לא אמינה אבל מוצפנת עם מסגרות DATAGRAM, שלא יישלחו מחדש גם בגילוי אבדן [10]. עם תמיכה בdatagram לא אמין, QUIC יכול לשפר את הגישה הנ"ל עם לחיצת יד אמינה ומאובטחת, ואז שליחה מאובטחת אך לא אמינה של datagrams. חבילות QUIC שמכילות רק מסגרות DATAGRAM גם גורמות לACK, אז האפליקציה יכולה לבדוק האם המסגרת נשלחה או לא.

5 שחזור בQUIC:

5.1 שערוך RTT:

ACK של QUIC מקודדים את העיכוב בין הקבלה של החבילה ושליחת הACK שלה, שמאפשר למקבל של הACK לחשב את הזמן בפועל שלוקח לשדר חבילה ברשת. אז כשקצה מקבל מסגרת ACK, הוא יכול לייצר דוגמה של הRTT של המסלול ברשת של ידי חישוב של הזמן שעבר מאז השליחה של החבילה הכי גדולה שאושרה.
QUIC משתמש ב3 הערכים הבאים כדי לייצר תיאור סטטיסטי של הRTT של המסלול ברשת: RTT מינימלי (min_rtt), ממוצע משוקלל מעריכי דינאמי של RTT (smoothed_rtt), והשונות הממוצעת של דוגמאות הRTT שנצפו (rttvar). על ידי שימוש במספרי חבילות מונוטוניים עולים, השליחה מחדש בQUIC נמנעת מבעיית ה"שליחה מחדש לא ברורה" של TCP, שנגרמת מזה שלחבילה שנשלחת מחדש יש את אותו מספר כמו החבילה המקורית.

5.2 בקרת עומס:

הפרדה של בקרת עומס מבקרת אמינות: QUIC משתמש במספרי חבילות לבקרת עומס, ו-stream frame offset לבקרת אמינות.
שימוש באלגוריתמים קיימים: בדומה לבקרת עומס ב-TCP, QUIC משתמש בבקרת עומס מבוססת חלון שמגבילה את מספר הבתים שהשולח יכול לשלוח בפרק זמן מסוים. QUIC לא מנסה לפתח אלגוריתמים משל עצמו לבקרת עומס, ולא להשתמש באחד ספציפי. QUIC מספק סימנים גנריים לבקרת עומס, והמשתמש חופשי להשתמש במנגנונים משלו. אלגוריתם בקרת עומס שמתועדת בסטנדרט של QUIC מתואר בנספח A.
כדי להימנע מהקטנה מיותרת של חלון בקרת העומס, QUIC לא מקטין את החלון אלא אם כן הוא מזהה *עומס מתמיד*. כאשר שתי חבילות שצריכות ACK מוגדרות כאבודות, נגדיר מצב של עומס מתמיד רק אם:

- אף אחת מהחבילות ביניהן לא קיבלה ACK,
- קיימת דוגמה RTT מלפני שהן נשלחו,
- וההבדל בין זמני השליחה שלהן גדול מזמן העומס המתמיד - persistent congestion duration.

זמן העומס המתמיד מחושב על ידי שילוב של ה- smoothed_rtt, rttvar, והזמן המקסימלי שמקבל עלול להשתהות לפני שליחת אישור.
שולח בQUIC יקציב את השליחה כדי למזער את הסיכוי לגרום לעומס בטווח הקצר, על ידי ווידוא שההפרשים בין שליחת החבילות הם יותר מהזמן המחושב לפי ה-smoothed_rtt, גודל החלון, וגודל החבילה.

5.3 גילוי ושחזור אובדן:

גילוי אבדות לפי ACK: כמו שמתואר לעיל, כל חבילת QUIC מכילה כמה מסגרות, שכל אחת מהן יכולה להיחשב כמקבילה לפקטת IP. QUIC מגלה אבידות לפי החבילות האלה (כלומר, מקביל לאוסף של פקטות IP): לכל חבילה שמקבלת ACK, כל המסגרות בחבילה הזאת נחשבות שקיבלו ACK. המסגרות נחשבות אבודות אם החבילה לא קיבלה ACK, או אם חבילה מאוחרת יותר קיבלה ACK ונחצה רף מסוים. QUIC משתמש בשני סוגי רפים כדי לקבוע האם חבילה תיחשב אבודה:

1) לפי מספר חבילה: אם קיבלנו ACK על חבילה עם מספר גבוה בפער מסוים מהחבילה האבודה. לדוגמה, אם מספר החבילה המאושרת הכי גדול הוא X והרף הוא T, אז כל החבילות "באוויר" שיש להן מספר קטן מ X-T ייחשבו אבודות.
2) לפי זמן: אם החבילה האבודה נשלחה לפני לפחות מספר פעמים המקסימום מבין הRTT המשוער של הרשת והRTT האחרון שנמדד. כלומר, משווים בין הRTT המשוער והRTT האחרון שנמדד, ולוקחים את המקסימום. נקרא לו m. יש קבוע מסויים, נקרא לו c, שמגדיר את הרף. אז הרף יהיה $m \cdot c$.
נניח שחבילה קיבלה ACK בזמן t והרף הוא t_0. אז כל החבילות "באוויר" שנשלחו לפני $t - t_0$ ייחשבו אבודות.

הרפים האלה נותנים איזשהו מרווח בשביל סידור מחדש של חבילות ונמנע משליחה מחדש מיותרת. זה גם נועד למנוע ירידה בביצועים שנגרמת מבקרת העומס בגלל גילוי אבידות.
כדי לגלות אובדן של חבילות "זנב", QUIC מאתחל טיימר בשביל ה- (Probe Timeout Period (PTO בכל זמן שחבילה שדורשת ACK נשלחת. ה-PTO מוגדר כמו זמן העומס המתמיד (5.2). כשהטיימר של ה-PTO נגמר, השולח ישלח עוד חבילה שדורשת ACK בתור גישוש, שיכול לחזור על מידע שכבר באוויר כדי למנוע שליחות מחדש.

שחזור אובדן: אחרי שמגלים אבידה, המסגרות האבודות מוכנסות לתוך חבילות יוצאות חדשות (שיקבלו מספרים חדשים, בלי קשר לחבילות האבודות).
עם גילוי ושחזור אבידות, QUIC מאפשר שליחה של זרם בתים בצורה מסודרת ואמינה, בדומה לTCP.

6 אבטחת QUIC:

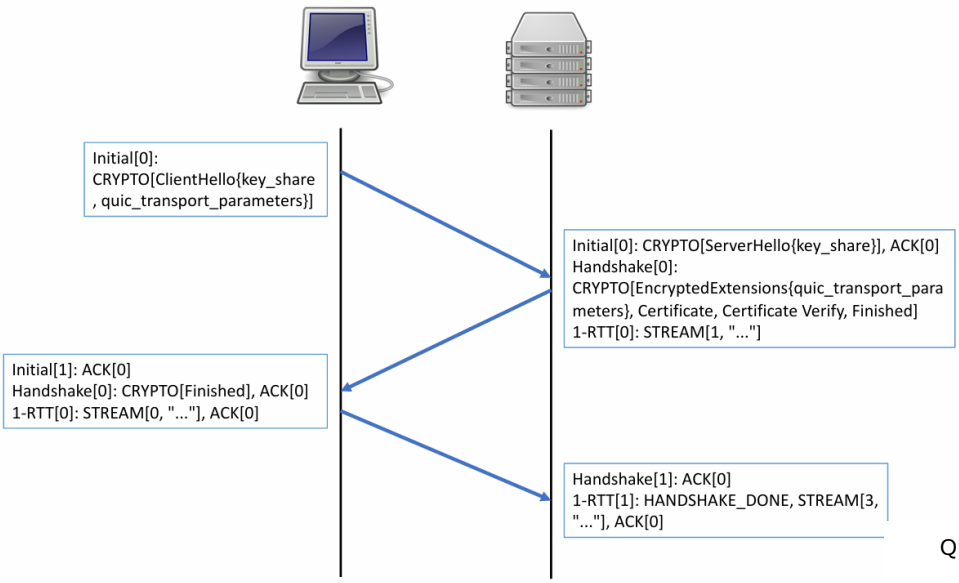
אבטחה היא בעדיפות ראשונה בפרוטוקול. כתוצאה מכך, QUIC מצפין כמעט הכל בתוך הפרוטוקול (חוץ משדות שהכרחיים לרשת, כמו כתובות מקור ויעד), ביחד עם שילוב הדוק של אבטחה. חלק מהשילוב הזה הוא איחוד לחיצות הידיים של התעבורה והאבטחה: זה מאפשר להוריד RTT אחד מהזמן שדרוש לבנות את החיבור. למרות שבאופן נאיבי נראה שזה עובר על העקרון של הפרדת רכיבים, ברוב המקרים שבהם משתמשים בTCP, משתמשים בTLS מעליו, ככה שאין פה הקרבה בכלל.

6.1 לחיצת ידיים קריפטוגרפית בQUIC:

QUIC משתמש במפתחות שנגזרים מלחיצת ידיים של TLS כדי להגן על הסודיות של החבילות [19] וההודעות של לחיצת הידיים של הTLS נשלחות במסגרות CRYPTO בחבילות הראשונות וחבילות לחיצת הידיים, שמחוברות לתהליך לחיצת הידיים של התעבורה. אפשר לתאר את היחס בין QUIC ל-TLS כך: QUIC לוקח מידע מ-TLS (הודעות לחיצת יד, מפתחות וכו') ובתמורה נותן לTLS זרם אמין. התהליך הכולל ללחיצת ידיים קריפטוגרפית בRTT 1 בלי אימות של הלקוח מתוארת באיור 7 ועובדת כך:

- הלקוח מאתחל את תהליך לחיצת היד הקריפטוגרפית על ידי שליחת הודעת ClientHello של TLS בחבילה הראשונית שלו, שכוללת את הcipher suites שהלקוח תומך בהם, ה- public share of its ephermal Diffie-Helman key, (החלק הפומבי של מפתח הDH החולף) ופרמטרי התעבורה שלו. עוד תוספות של TLS כמו אינדיקציה לשם השרת יכולות גם להיכלל בהודעת ה ClientHello.
- אחרי שהשרת מקבל את החבילה הראשונית של הלקוח, הוא עונה עם חבילה ראשונית שיש בה:
הודעת הServerHello של TLS, שכוללת את הcipher suite שנבחרה על ידי השרת לחיבור הזה,
החלק הפומבי של המפתח Diffie-Helman של השרת,
ואולי עוד הרחבות של TLS שהכרחיות ליצירת ההקשר הקריפטוגרפי. בשלב הזה, השרת יכול לגזור את הMaster Secret עם המפתח הפרטי שלו והמפתח הציבורי של הלקוח, וישתמש במפתחות האלה כדי להגן על החבילות העתידיות. אחרי זה, השרת שולח חבילת לחיצת יד שיש בו את הודעת לחיצת היד הבאה:
הודעת ה- EncryptedEx tensions כוללת את פרמטרי התעבורה של השרת, עוד תוספות של TLS שלא נדרשות בשביל לייצר הקשר קריפטוגרפי, הודעה הכוללת את ה certificate chain של השרת, הודעת certificate Verify שמוכיחה בעלות על המפתח הפרטי, והודעת Finished שנותנת אימות של ללחיצת היד והמפתחות [17].
השרת יכול עכשיו להתחיל לשלוח מידע עם חבילות RTT 1 למרות שעוד לא ווידאנו שהלקוח "חי".
- אחרי שהלקוח קיבל את החבילה הראשונית וחבילת לחיצת היד של השרת, הוא גם יכול להסיק את ה Master Secret של החיבור ולחשב את המפתחות כדי להגן על החבילות העתידיות. הוא ישלח את הודעת הFinished של TLS בחבילת לחיצת יד כדי לאשר את לחיצת היד. הוא גם יכול להתחיל לשלוח מידע לשרת בחבילות RTT 1.
- אחרי שהקבלה על הודעת הFinished מהלקוח, השרת ישלח מסגרת HANDSHAKE_DONE בחבילת RTT 1 כדי לאשר את סיום לחיצת היד.

בגלל שחבילות רבות יכולות להיכלל ב UDP datagram יחיד, הפעולות הנ"ל יכולות להתבצע ב4 Datagrams בלבד.

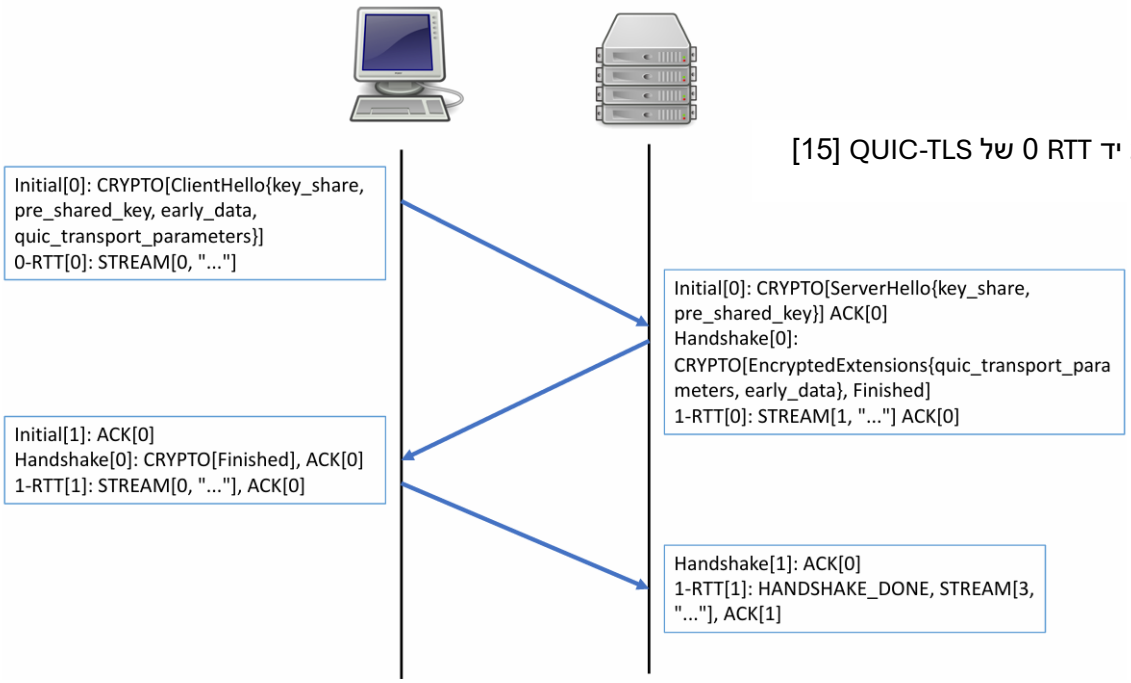


איור 7: לחיצת יד RTT 1 של QUIC-TLS

שליחה של מידע 0 RTT: QUIC מאפשר ללקוח לשלוח מידע אפליקציה מוצפן לפני שלחיצת הידיים הושלמה על ידי שימוש בפרמטרים מחיבור קודם [19]. אחרי שחיבור נקבע, שרת QUIC יכול לייצא זהות מפתח שכבר שותף – Pre Shared Key, (PSK) – שמקושר ל resumption secret של החיבור דרך הודעת NewSessionTicket של TLS עם גודל מידע מוקדם מקסימלי מיוחד כדי לסמן שהוא יקבל מידע RTT 0. לקוח QUIC יכול לזכור את זהות הPSK הזו ביחד עם פרמטרים חיוניים כדי שיוכל לעשות לחיצת יד פשוטה יותר בפעם הבאה. התהליך הכללי של לחיצת יד RTT 0 מתוארת באיור 8 ועובדת כך:

- הלקוח יוזם את לחיצת היד על ידי שליחת חבילה ראשונית בדומה לזו בתהליך לחיצת יד של RTT 1, עם תוספות TLS מסוימות. הלקוח ישתמש בתוספת pre_shared_key כדי להגיד לשרת את זהות הPSK וישתמש בתוספת הearly_data- כדי לסמן שיש לו מידע RTT 0 לשלוח. המידע אפליקציה נכלל בחבילת RTT 0 שמוגן על ידי ה-resumption secret, שיכול להיכנס לתוך UDP datagram יחיד ביחד עם החבילה הראשונית.
- השרת משתמש במחסנית הTLS כדי לבדוק אם החבילה אמינה. אם כן, השרת משתמש במחסנית הQUIC ופרוטוקול האפליקציה כדי לבדוק את האמינות של החבילה. חלק מהבדיקה של מחסנית הQUIC היא לוודא שאיזשהו מצב תעבורה מחובר ל-session ticket, מעל ומעבר לדרישות של TLS.
אם השרת בוחר לא לקבל מידע RTT 0, הוא יחזור ללחיצת יד RTT 1 ואין צורך בפעולות מיוחדות.
- השרת ישלח חבילה ראשונית וחבילת לחיצת יד בדומה ללחיצת יד RTT 1 עם כמה שינויים. בהודעת הServerHello, השרת יכלול את תוספת ה pre_shared_key כדי לסמן שהוא מצפה לזהות הPSK. בחבילת לחיצת היד, תוספת הTLS early_data תתווסף להודעת ה-EncryptedExtensions כדי לסמן שמידע ה RTT 0 מקובל, והשרת לא צריך לשלוח certificate או הודעת Certificate Verify כי הזהות שלו כבר אומתה. כדי לספק אבטחה קדימה לחיבור החדש, השרת ינדוד ל- Master Secret חדש שמשלב את ה- Resumption Secret ואת ה- Shared Secret שנגזר מהחלפת מפתחות ה-DH החדשים. השרת גם ישלח מסגרת ACK כדי לאשר את חבילת ה RTT 0.
- הלקוח ינדוד ל Master Secret החדש חרי שהוא קיבל את החלק הפומבי של מפתח הDH החולף של השרת. שאר התהליך זהה ללחיצת יד של RTT 1.

בכל זאת, יש כמה דברים שכדאי לעשות באופן כללי. לדוגמה, כשמשתמשים בRTT 0, עדיף להשתמש במפתחות RTT 0 כדי להגן רק על מידע אימפוטנטי בגלל שחבילות RTT 0 אינן מוגנות בפני reply attack. בנוסף, כדי למזער פגיעות אבטחה, כשהמידע השמור של הלקוח פג תוקף, השרת צריך לדחות את חיבור ה RTT 0 ולשלוח את מידע האימות שלו כמו ביצירת חיבור בפעם הראשונה. זה מובנה בפרוטוקול QUIC, והלקוח אמור לעבור ישירות לתהליך הזה בלי בעיות.



איור 8: לחיצת יד RTT 0 של QUIC-TLS [15]

6.2 כותרת ומטען מאומת ומוצפן:

פרוטוקול QUIC מצפין את כל החלקים הפרקטיים של החבילה, בכוונה. למרות שיש פה פשרה – לדוגמה, להסתרת מידע מהISP יש יתרונות וחסרונות – זה מאפשר הגנה חדשה למשתמשים. בפרט, QUIC מאמת את כל הכותרות ומטענים שלו (חוץ מחבילות שקובעות גרסה), וגם מצפין את רוב המידע שעובר. איור 4 מראה את מבנה החבילה של QUIC, כותרות של חבילות אינן מוצפנות כי הן משמשות להכוונה או פענוח של המטען. גוף החבילה, שמכיל מסגרות, מוצפן. כל מה שיש בכותרת המוצפנת צריך להישאר ב-plaintext בשביל פעילות תקינה. הכותרת מכילה דגלים שמגדירים איזה שדות קיימים ובאיזה אורך. מזהה החיבור מכוון את החבילה לשרת היעד ובו זמנית משמש כמזהה למצב החיבור. מספר החבילה דרוש בשביל אימות ופענוח, ולכן לא יכול להיות מוצפן.
להצפנה הזו יש גם יתרון בכך שהיא מבטיחה שיחסית נוח לשפר ולעדכן את QUIC. התאבנות של פרוטוקולים היא בעיה ידועה, קשה לעדכן middleboxes כדי להתאים לשינויים בפרוטוקול, שמגביל את הגמישות בעיצוב פרוטוקולי רשת. חבילות QUIC ברובן מוצפנות, מה שמונע שינוי על ידי middleboxes, ומגביל את ההתאבנות של הפרוטוקול.

7 אפליקציות מעל QUIC: HTTP/3 בתור דוגמה:

כפרוטוקול אפליקציות, HTTP מקודד אלמנטים מסוימים עם סמנטיקה עשירה, והתכונה הכי רלוונטית היא מודל הודעות בקשה-תגובה: לקוח דפדפן יכול לשלוח מספר בקשות בו זמנית, שהתגובות שלהן מצריכות תגובות בעדיפויות שונות כדי לשפר את חווית המשתמש. אבל אם HTTP רץ על חיבור TCP, שמאפשר שליחה על זרם בתים יחיד בלבד, התגובות יכולות להישלח רק בסדר שבו השרת קיבל את הבקשות.
העיצוב של QUIC תומך בזרמים מרובים כדי לענות על ההגבלה הזאת של TCP. בפרט, HTTP/3 (הגרסה העדכנית של HTTP שנועדה לרוץ על QUIC) משתמש בסמנטיקת הזרמים של QUIC כדי לאפשר לכל תגובת HTTP להישלח באופן בלתי תלוי ועם עדיפויות שונות.

- בתור התעבורה מאחורי הקלעים של HTTP/3, QUIC מספק שליחה אמינה לפי הסדר ברמת הזרם, ובקרת עומס ברמת החיבור. כל זוג בקשה-תגובה של HTTP ממופה לזרם עצמאי, וכך זוגות שונים לא יחסמו אחד את השני במקרים של אבידות. QUIC גם מספק אבטחה שווה ל TLS+TCP, ופחות עיכובים ביצירת חיבור.
- בקרת זרמים נשלטת ברמת התעבורה. QUIC מטפל בשליחה אמינה וסידור של המסגרות ושולח את המידע לאפליקציה.

8 סיכום:

QUIC מייצג את העיצוב הכי טוב של פרוטוקול תעבורה עד כה. הרעיונות הבסיסיים של העיצוב לא נפלו מהשמיים יום אחד, אלא QUIC מייצג הצטברות של לקחים שנלמדו מהתנסות ברשתות ועיצובי פרוטוקולים קודמים בעשורים האחרונים. לדוגמה, QUIC לומד מ- T/TCP ושומר וממחזר מצבי חיבור כדי לאפשר התקשרות ב RTT 0. QUIC גם מאמץ רעיונות מRTP, ורץ מעל UDP כדי להישאר מחוץ לkernel ומשתמש ברעיון ALF/ADU שמתואר ב [7]. בדומה ל SCTP ו- HTTP/2, QUIC משתמש בזרמים מרובים כדי להקל על חסימת ראש תור, ומסגרות מסוגים שונים כדי לתמוך במגוון התקשרויות שליטה. האימוץ של הרעיונות האלה ואיחודם לפרוטוקול אחד מאפשר לQUIC למזער עיכובים ובעיות אחרות.

- תמיכה בשליחה אמינה של מידע לגופים מרובים.
- תמיכה בהכלת עיכובים.

נספח א: אלגוריתם NEWRENO:

האלגוריתם מתועד בסטנדרטים של QUIC [14]. מכונת המצבים מתוארת באיור 9. בנוסף לחלון העומס, יש משתנה נוסף שנקרא רף התחלה איטית – slow start threshold. הוא מאותחל באינסוף. באלגוריתם יש שלושה שלבים:

1) **התחלה איטית – slow start:** השולח בQUIC מתחיל במצב התחלה איטית ויחזור למצב אם מתגלה עומס מתמיד. במצב הזה, חלון העומס יגדל בצורה מעריכים (אקספוננציאלית); בכל פעם מגדילים במספר הבתים החדשים שקיבלו אישור. השולח ייכנס למצב התאוששות (recovery) אם חבילה הוגדרה אבודה או הcounter של ECN-CE הוגדל.
2) **התאוששות – recovery:** בכל פעם שהשולח נכנס למצב התאוששות, חלון העומס יקטן בחצי ורף ההתחלה האיטית יוגדר הגודל החדש של חלון העומס. השולח ייכנס למצב הימנעות מעומס כשחבילה שנשלחה בזמן מצב התאוששות מקבלת אישור.
3) **הימנעות מעומס – congestion avoidance:** במצב הזה, נשתמש בגישת "הגדלה בחיבור, הקטנה בכפל" – additive increase, multiplicative decrease. (AIMD). לכל אישור של חלון עומס, ההגדלה של החלון תוגבל לdatagram אחד לכל היותר. השולח ייכנס למצב התאוששות כאשר חבילה מוגדרת אבודה או הcounter של ECN-CE הוגדל.
4) **טיפול בעומס מתמיד:** כשמוכרז עומס מתמיד, חלון העומס יוקטן למינימום והשולח יחזור למצב התחלה איטית.

```
                 New Path or      +------------+
            persistent congestion |   Slow     |
       (O)---------------------->|   Start    |
                                 +------------+
                                       |
                                Loss or |
                        ECN-CE increase |
                                       v
+------------+     Loss or       +------------+
| Congestion |  ECN-CE increase  |  Recovery  |
| Avoidance  |------------------>|   Period   |
+------------+                   +------------+
          ^                            |
          |                            |
          +----------------------------+
             Acknowledgment of packet
               sent during recovery
```

איור 9: מכונת מצבים של אלגוריתם New Reno

נספח ב: משא ומתן על גרסה:

בניגוד לפרוטוקולי תעבורה אחרים, QUIC תומך בקיום במקביל של גרסאות שונות. כדי לתמוך בתכונה הזו, השרת ולקוח יכולים לנהל משא ומתן כדי להסכים על גרסה נתמכת לפני יצירת החיבור. זה שימושי כדי לאפשר לפרוטוקולים להתפתח ועדיין לאפשר לצדדים להחליט באיזה גרסה להשתמש. בשביל לקוחות שתומכים בגרסאות מרובות, QUIC צריך לבחור את המקסימום מבין גודל החבילה המינימלי מכל הגרסאות הנתמכות בתור החבילה הראשונה. אם השרת לא מקבל את הגרסה, הוא יחזיר חבילה עם רשימה של גרסאות נתמכות. זה יוסיף עיכוב של 1 RTT לתהליך לחיצת הידיים.

[1] Transmission Control Protocol. RFC 793, September 1981.

[2] Mike Bishop. Hypertext transfer protocol version 3 (http/3). Rfc, February 2021.

[3] Ethan Blanton, Dr. Vern Paxson, and Mark Allman. TCP Congestion Control. RFC 5681, September 2009.

[4] Robert Braden. T/tcp– tcp extensions for transactions functional specification. RFC 1644, July 1994.

[5] Michelle Cotton, Lars Eggert, Dr. Joseph D. Touch, MagnusWesterlund, and Stuart Cheshire. Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry. RFC 6335, August 2011.

[6] AdamLangley et. al. The quic transport protocol: Design and internet scale deployment. In Proc. of SIGCOMM, 2017.

[7] David Clark et. al. Architectural considerations for a new generation of protocols. In Proc. of SIGCOMM, 1990.

[8] Eric Rescorla et. al. Datagram transport layer security version 1.2. RFC 6347, January 2012.

[9] Henning Schulzrinne et. al. Rtp: A transport protocol for real-time applications. RFC 3550, July 2003.

[10] Tommy Pault et. al. An unreliable datagram extension to quic. Rfc, March 2021.

[11] Bryan Ford. Structured streams: a new transport abstraction. In Proceedings of the 2007 conference on Applications, technologies, archi tectures, and protocols for computer communications, pages 361–372, 2007.

[12] IANA. Registry. Service Name and Transport Protocol Port Number https://www.iana.org/assignments/service-names-port numbers/service-names-port-numbers.xhtml, 2021 (Last Updated 2021-10-04).

[13] Jana Iyengar. The maturing of quic, fastly, industry insights. https://www.fastly.com/blog/maturing-of-quic, November 2019.

[14] Jana Iyengar and Ian Swett. QUIC Loss Detection and Congestion Control. RFC 9002, May 2021.

[15] Jana Iyengar and Martin Thomson. QUIC: A UDP-Based Multiplexed and Secure Transport. RFC 9000, May 2021.

[16] Jon Postel. User datagram protocol. RFC 768, August 1980.

[17] Eric Rescorla. The transport layer security (tls) protocol version 1.3. RFC 8446, August 2018.

[18] Randall Stewart. Stream control transmission protocol. RFC 4960, September 2007.

[19] Martin Thomson and Sean Turner. Using TLS to Secure QUIC. RFC 9001, May 2021.