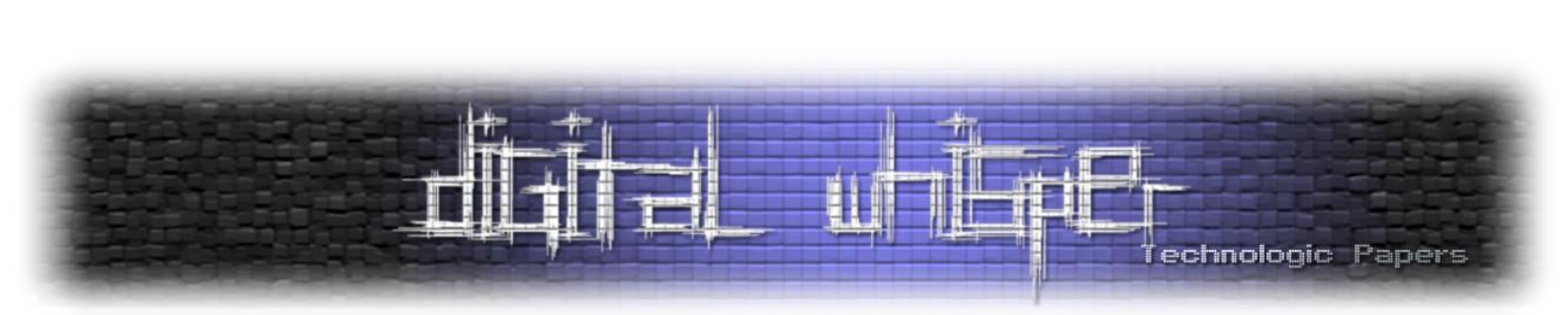


על QUIC

מאת ברק גונן

הקדמה

פרק זה הוא הפרק האחרון מהספר "רשתות מחשבים חלק ב" שיצא זה עתה בהוצאת המרכז לחינוך סייבר. הספר מעדכן בהתפתחויות טכנולוגיות שהיו ב-12 השנים שעברו מאז יציאת הספר "רשתות מחשבים" והוא עשוי לסייע למבקשים להתמיין ליחידות טכנולוגיות בצבא.

פרוטוקול QUIC לא היה קיים בעת יציאת הספר הראשון וכיום תופס נתח הולך וגדל מהאינטרנט. קשה למצוא חומרי לימוד על פרוטוקול זה באנגלית, קל וחומר בעברית. לכן ראיתי חשיבות בהנגשת החומר לקהל הטכנולוגי הישראלי.

אני מבקש להודות לעורכי Digital Whisper אפיק קסטיאל וספיר פדרובסקי על הרוח הגבית והפרסום שנתנו לספר, כמו גם על המפעל החינוכי ארוך השנים שלהם.

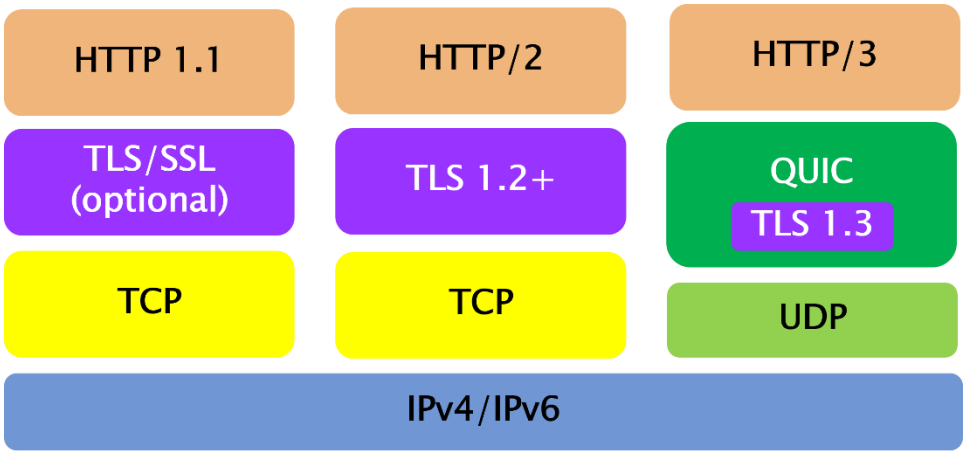
בפרקים הקודמים כיסינו את ההתקדמות הטכנולוגית שהיתה בין אמצע שנות ה-90, כאשר תעבורת האינטרנט היתה לא מאובטחת, ועד 2018, השנה שבה פורסם תקן TLS בגרסה 1.3.

בפרק זה נתקדם עוד כמה שנים קדימה, לשנת 2021 ולפרוטוקול חדש שפורסם ואשר נכון למועד כתיבת שורות אלה הוא הפרוטוקול העדכני והמתקדם ביותר. כאשר נסיים ללמוד אותו נהיה בנקודה שבה יש לנו הבנה בפרוטוקול החדש ביותר שיוצר סוקטים מאובטחים.

שם הפרוטוקול - QUIC. ראשי תיבות של Quick UDP Internet Connections.

רגע אחד! UDP? הרי UDP אינו פרוטוקול אמין. כל מה שלמדנו עד כה הביא אותנו לתובנה שכל פרוטוקול שדורש אמינות חייב להיות מעל TCP. ובכן, נכון, פרוטוקול QUIC שובר הנחת יסוד משמעותית שלמדנו בכך שהוא עובד מעל UDP. ולא רק זאת, אלא ש-QUIC גם שובר את מודל השכבות המוכר לנו. פרוטוקול שנמצא מעל שכבת התעבורה, סופח אליו כמעט את כל התפקידים שהיו עד כה שמורים ל-TCP.

התמונה הבאה נותנת מושג על השינויים שעברנו ועל השינוי שאנחנו הולכים לעסוק בו בפרק זה:



מצד שמאל: האינטרנט הלא מאובטח, או המאובטח בגרסת TLS ישנה כלשהי. רואים יפה את מודל השכבות המוכר, כאשר מעל IP יש TCP, מעליו יש TLS ישן כלשהו (או אין, תלוי כמה אחורה הולכים בזמן), ומעליו HTTP גרסה 1.1.

באמצע- המעבר ל-TLS גרסה 1.2 או 1.3, ויחד איתו גרסה 2 של HTTP. עדיין השכבות למטה אינן משתנות. IP ומעליו TCP.

מצד ימין- תוהו ובוהו. לקחנו את המבנה הקודם, ששירת אותנו עשרות שנים, ושברנו אותו. החלפנו את TCP ב-UDP. מעליו יש משהו שנקרא QUIC, שבאופן מוזר ביותר כולל **בתוכו** את TLS 1.3. מעליו - גרסה חדשה של HTTP, גרסה 3. הפרוטוקול היחידי שנותר מערימת הפרוטוקולים המקורית הוא IP.

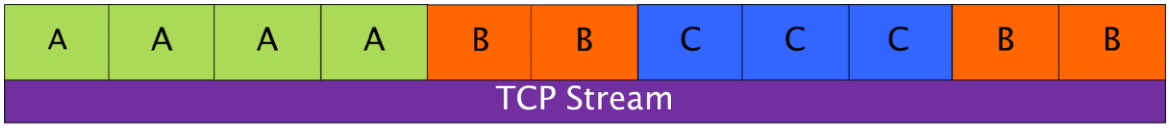
במהלך הפרק נענה על אוסף שאלות מעניינות שיחולקו למספר חלקים. בחלקו הראשון של הפרק נבין מה היו החסרונות והבעיות של HTTP/2 מעל TLS 1.3 מעל TCP. כך נבין מדוע המעבר ל-UDP הוא מוצלח ואף הכרחי. חלקו השני של הפרק ידון בפרוטוקול TCP, מכיוון שאם רוצים לבטל את TCP אז צריך להבין מה היכולות של TCP שנצטרך למצוא להן תחליף. בחלק השלישי נסקור את QUIC בליווי הסנפה, ונראה כיצד מיושמים העקרונות של TCP בלי שימוש ב-TCP. נעבור על כל המוטיבציות לשינוי מהמצב ששרר לפני QUIC ונראה כיצד השינויים עונים על הבעיות. בחלק הרביעי נסקור את השינויים שחלו ב-HTTP בין גרסה 2 לגרסה 3, ונסיים בצפיה ב-DNS מעל QUIC.

HTTP/2 - מקומות לשיפור

בפרק הקודם הצגנו את ההתקדמות המשמעותית שהיתה בין HTTP 1.1 ל-HTTP/2. כעת נסקור את הדברים המרכזיים שצריך לשפר ב-HTTP/2, רובם המכריע לא קשורים ישירות ל-HTTP/2 אלא לפרוטוקול TCP, שנמצא בחבילת הפרוטוקולים עליהם HTTP/2 מתבסס.

בעיית ה-Head Of Line Blocking

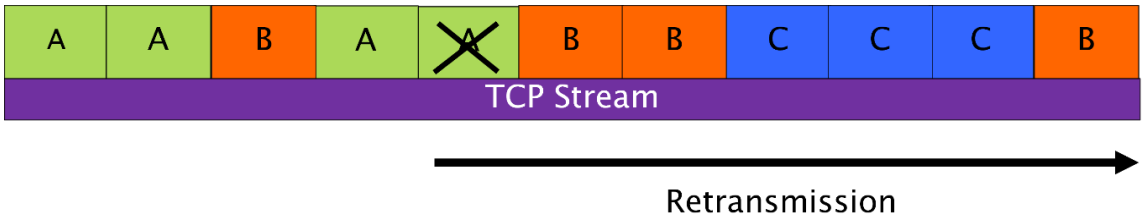
פרוטוקול HTTP/2 משתמש, כפי שראינו, בקישור TCP יחיד שעליו מועברים זרמי מידע שונים. כדי להפריד בין זרמי המידע, לכל זרם יש stream ID שהינו חלק מפרוטוקול HTTP/2, ואשר מאפשר לצד המקבל לאסוף יחד פקטות ששייכות לאותו זרם. דבר זה מאפשר לנצל בצורה מיטבית את הערוץ,מכיוון שגם אם משאב אחד "תקוע", או נמצא בהמתנה, אפשר לנצל את הערוץ כדי להעביר משאבים אחרים.



נראה שהשגנו ניתוק בין הזרמים השונים, כל זרם מידע יכול לעבור בלי תלות בזרמי המידע האחרים. אך האמנם זה באמת המצב?

דמיינו תור של אנשים הממתינים לשירות של קופאי בבנק. הקופאי מטפל בלקוח, אבל בגלל בעיה טכנית הפעולה מתארכת. כל הממתינים בתור לקופאי צריכים לחכות, למרות שיכול להיות שהם צריכים שירותים אחרים, שאין בעיה טכנית לבצע אותם. מי שנמצא בראש התור יוצר "פקק" לכל התור. דבר זה נקרא Head of Line Blocking.

וכעת נחזור לעולם התקשורת. האיור הבא ממחיש מה קורה כאשר פקטת TCP אחת נפלה.



למרות שכל הפקטות שאחריה הגיעו בצורה תקינה ליעד, הן עלולות להיזרק והשולח יצטרך לשלוח אותן מחדש. כן, גם פקטות שנושאות Stream ID של משאב B או של משאב C, שאין שום בעיה לשמור אותן ולהשתמש בהן, עלולות להזרק. אם הדבר אינו ברור לכם, בהמשך הפרק נסביר על ה-Cumulative ACK של TCP ונבין מדוע אובדן של פקטה מעכב את הטיפול בפקטות הבאות. פרוטוקול TCP מייצר תלות בסיסית בין זרמי המידע, בצורה שגם הפרדה מאוחרת יותר בשכבת האפליקציה לא יכולה לתקן. פקטה שאבדה היא כמו לקוח שנתקע אצל הקופאי, יוצרת Head Of Line Blocking.

TCP Ossification

האם אפשר איכשהו לתקן את TCP, או להוסיף לו שיפור כלשהו, שיפתור את בעיית ה-Head of Line Blocking? תאורטית כן, מעשית - בעיה גדולה. כדי להסביר מדוע זה קשה מאד, נסקור את תהליך ה-Ossification של TCP וניתן דוגמה לשיפור יפה של TCP, שיכל להיות קיים, אך כשל את האינטרנט.

המונח Ossification פירושו "התגרמות", מהמילה "גֶרֶם", שפירושה "עצם". לדוגמה, מישהו שהוא גרום הוא מישהו רזה במידה שהעצמות בולטות ממנו. תהליך ההתגרמות הוא תהליך טבעי שבו רקמת סחוס הופכת לעצם. תינוקות נולדים עם עצמות קטנות והרבה סחוס ביניהן. בתהליך הגדילה וההתבגרות הסחוס מתגרם לעצם, והתהליך מסתיים בתום ההתבגרות כאשר אין כמעט רווח בין העצמות.

הכוונה בפרוטוקול שעבר Ossification היא שהוא הגיע למצב של בגרות כזו, שאיבד את הגמישות שלו וקשה להכניס לתוכו תוספות או שינויים.

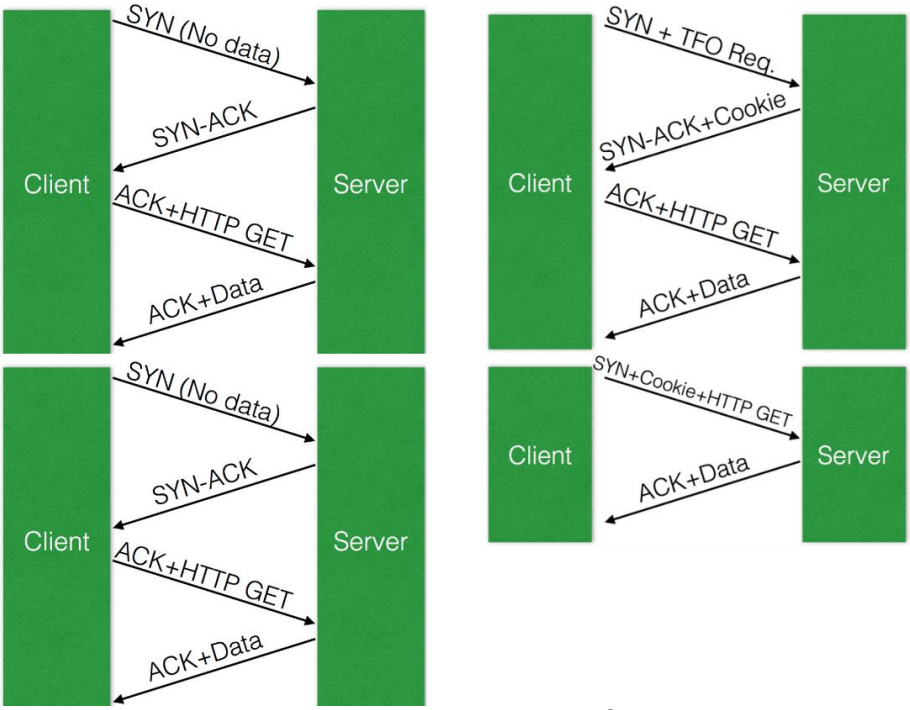
כדי להבין למה ואיך פרוטוקול TCP עבר Ossification צריך להפנות את המבט אל הרכיבים שנמצאים בין השרת והלקוח, ה-Middleboxes. כל אותם רכיבים כגון Firewall, NAT, רכיבים שמבצעים תרגום בין פרוטוקולים, רכיבי DPI - Deep packet inspection - ועוד ועוד. רכיב כזה עשוי לעשות את עבודתו במשך שנים רבות ואז להתקל בפקטה שכוללת פרוטוקול לא מוכר, או איזושהי המצאה חדשה שמבוססת על פרוטוקול מוכר אך עובדת קצת אחרת. במקרה זה הרכיב עשוי להעביר את הפקטה כמו שהיא, אך הוא עלול גם להשליך אותה או לשנות אותה. וזה עלול להיות קטלני להמצאות חדשות.

כבר כאשר דנו בשדות של TLS record ראינו שאחד השדות הוא גרסת ה-TLS וראינו שהערך של שדה זה אינו תמיד תואם את הגרסה האמיתית של פרוטוקול ה-TLS שהוא מעביר. זו היתה דוגמה להתחלה של Ossification של פרוטוקול TLS.

פרוטוקול TCP הוא ותיק בעשרות שנים מ-TLS ואותם Middleboxes למדו אותו היטב, מה יש בו ומה אין בו. וכאשר מנסים לשלוח פקטת TCP עם משהו חדש, המשהו החדש הזה לא תואם לידע של מה יש ומה אין בפרוטוקול. לכן, יש סיכוי לא מבוטל ששינויים וחידושים ב-TCP ייכשלו במבחן המעשי של זרימה ברשת האינטרנט.

דוגמה קלאסית לכך הוא מה שנקרא TCP Fast Open, או בקיצור TFO. הרעיון של TFO הוא לחסוך RTT במקרה של הקמת סוקט שנסגר לאחרונה, באמצעות קיצור ה-Three Way Handshake.

האיור הבא מדגים את התהליך. מצד שמאל המצב כיום, הקמת קישור, סגירה שלו, הקמת קישור חדש. מימין - TFO. הקמת קישור ואז קישור מקוצר:



[מקור: reproducingnetworkresearch]

המצב ללא TFO הוא שלקוח מתחבר לשרת לאחר שמקיים Three Way Handshake, בעל RTT של 1. נניח שהלקוח התנתק ורוצה ליצור קישור חדש, עליו לחזור על ה-Three Way Handshake ולשלם ב-RTT נוסף. ב-TFO, ה-Three Way Handshake הראשון עדיין לוקח RTT אחד, אבל הוא שונה במקצת. *על גבי* פקטת ה-TCP Syn, הלקוח יוסיף מידע שמשמעותו "בקשת TFO". במידה והשרת תומך ב-TFO, השרת יענה עם TCP Syn-Ack הכולל מחרוזת כלשהי, שהינה ייחודית לכל לקוח ולכל הקמת קישור. מחרוזת זו נקראת TFO Cookie.

כעת הלקוח התנתק ורוצה ליצור קישור מחדש. הוא שולח בקשת TCP Syn שכוללת את ה-TFO Cookie שהשרת שלח. ה-Cookie מציין "היי שרת, אנחנו כבר מכירים, בוא נדלג על ה-Three Way Handshake". בנוסף הלקוח כבר שולח על ה-TCP Syn מידע של שכבת האפליקציה, כגון HTTP GET. אם הקמת הקשר המקוצר מצליחה, אז הפקטה הבא שהשלח ישלח תהיה TCP Ack שכבר כוללת את התגובה לבקשה של הלקוח. קבלנו RTT-0 (ניזכר כי הכוונה שיש אפס הלוך ושוב לפני שהלקוח יכול לשלוח מידע של שכבת האפליקציה).

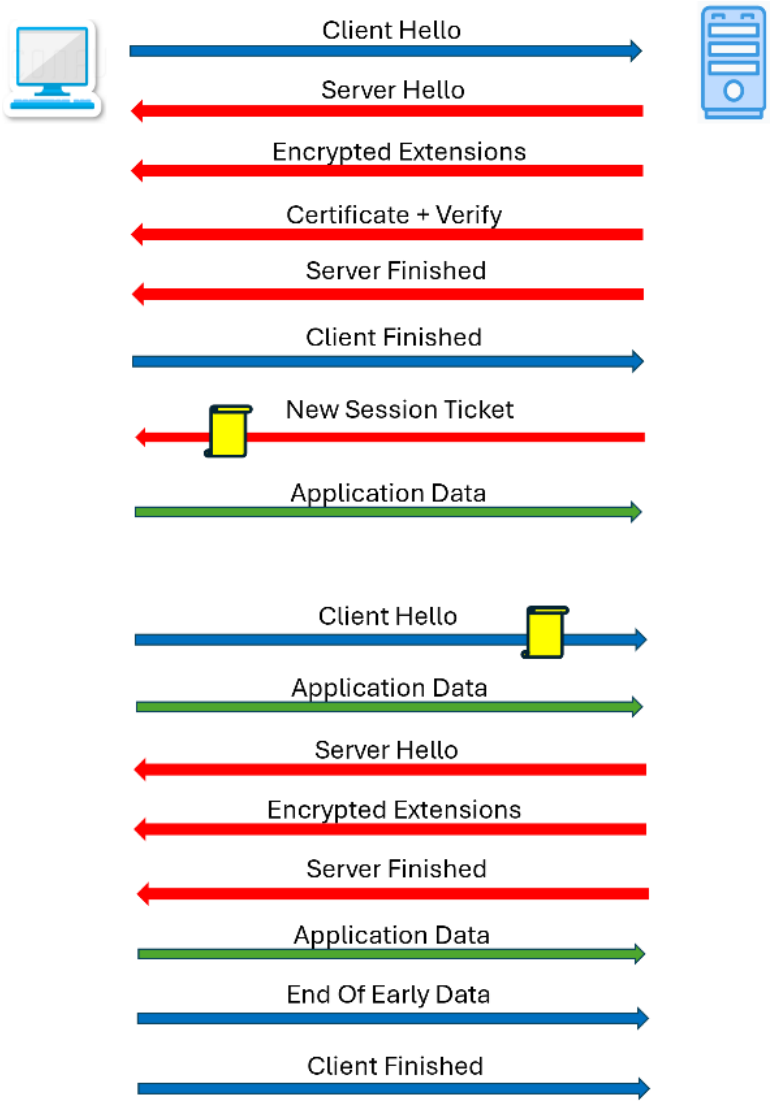
כאן נכנס לתמונה גורם ה-Ossification. הרעיון היפה של ה-TFO לא עבד בשטח. בערך 20% מהנסיונות להקים קישורי TFO לא עבדו. הסיבה לכך היא שאותם Middleboxes סברו "היי, מה פתאום יש מידע על גבי פקטת TCP Syn או TCP Syn Ack? זה ממש לא איך שפרוטוקול TCP אמור לעבוד. סכנה! זו עלולה להיות מתקפה, מוטב שאעיף את הפקטה הזו". וכך יצא שרעיון יפה, שהיה יכול להאיץ את האינטרנט, בוטל.

לסיכום, אין קשר ישיר בין TFO ל-QUIC. ה-TFO הוא דוגמה לכך שמי שרוצה לפתור את בעיות כגון TCP Head of Line Blocking עלול להתקשות מאד לעשות זאת באמצעות שינויים והגדרות של TCP.

הורדת כמות ה-RTT

הורדת כמות ה-RTT היתה ונשארה שאיפה שמלווה אותנו בכל תהליך השיפור של הפרוטוקולים. המטרה היא תמיד להקטין את כמות ההלוך-ושוב שהלקוח עושה עם השרת עד שהלקוח יכול לשלוח לשרת את ה-HTTP GET הנכסף, או כל מידע אחר של שכבת האפליקציה.

כפי שראינו, גרסת TLS 1.3 עשתה עבודה יפה בהורדה של RTT מגרסה 1.2. כאשר מבצעים Handshake רגיל מחכים RTT-1, ואילו עבור Session Resumption מחכים RTT-0, כפי שמראה האיור הבא מתוך הפרק על TLS 1.3:



[Session Resumption ולאחריו TLS 1.3 Handshake]

אלא שבאיור יש פרט קטנטן שחסר בו... כל עוד TLS 1.3 עובד מעל TCP, נוסף גם ה-RTT-1 של ה- TCP Handshake. היינו רוצים לבטל אותו, ואם אי אפשר לגמרי אז לפחות עבור Session Resumption. אין ספק ש-TFO היה יכול להשתלב פה בצורה נהדרת, היה אפשר להשיג RTT-0 אמיתי, כאשר פקטת ה- TCP Syn ששולח הלקוח כבר מכילה גם את ה-Client Hello וגם את ה-HTTP GET.

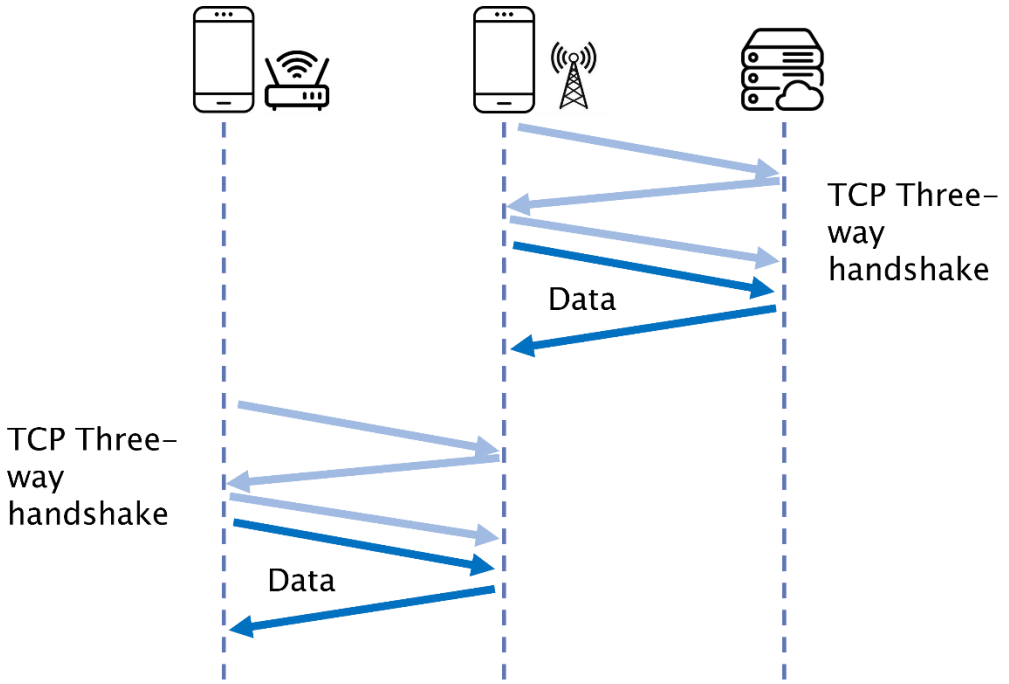
אבל TFO לא עובד. כל עוד אנחנו מחוברים ל-TCP, אנחנו צריכים לשלם את המחיר של ה- Three Way Handshake.

Connection Migration

אתם גולשים בסמארטפון תוך כדי נסיעה (לא נהיגה כמובן). הגעתם הביתה. הסמארטפון שלכם מזהה את הרשת הביתית ומתחבר אליה. מה קרה לכתובת ה-IP שלכם?

השתנתה כמובן. כל עוד הייתם ברשת הסלולרית, כתובת ה-IP שלכם הוקצתה על ידי הרשת הסלולרית. כעת היא מוקצית על ידי שרת ה-DHCP הביתי, שהוא גם הראוטר שלכם.

סוקט מורכב כזכור מארבעה מזהים, אחד מהם הוא כתובת ה-IP של הלקוח. שיניתם כתובת IP, טאק, כל קישורי ה-TCP שלכם התנתקו וצריך להקים אותם מחדש. נצטרך תוך כדי הגלישה לבצע Three Way Handshake חדש, וצפוי גם שהדבר יגרום לכך שחלק מהמשאבים שנשלחו אלינו לא יתקבלו ותידרש הורדה מחדש.



כלומר, TCP לא יודע לזהות Connection Migration ולאפשר מעבר חלק. אין ללקוח אפשרות לומר לשרת "היי, זה עדיין אני, בוא נמשיך מאיפה שהיינו".

מבוא ועקרונות QUIC

פרוטוקול QUIC, קיצור של Quick UDP Internet Connection. הגרסה הראשונית שלו פותחה על ידי גוגל ב-2012. גוגל רצו ליצור פרוטוקול שיאיץ את הגלישה לשירותים שלהם, לדוגמה יוטיוב.

כאשר חברה מסחרית מפתחת פרוטוקול, היא יכולה לשמור אותו לעצמה או לנסות להפוך אותו לתקן. אם החברה בוחרת לשמור אותו לעצמה היא מרוויחה יתרון תחרותי, אף אחד לא יכול להשתמש בו חוץ ממנה. במקרה של גוגל, הכסף שהיא מרוויחה מגיע לא מהדפדפן אלא מפרסומות, בין היתר ביוטיוב. לכן לגוגל אין רווח כלכלי מיוחד מכך שרק הדפדפן שלה יתמוך בפרוטוקול החדש שהם המציאו. לעומת זאת, אם יצרני דפדפנים נוספים ישתמשו בפרוטוקול שלהם, אז הגלישה ליוטיוב תהיה יותר מהירה מה שיעלה את כמות הצופים ביוטיוב ויוריד גם עלויות של שרתים. בשנת 2016 גוגל מחליטים לשתף את ה-IETF, צוות המשימה של האינטרנט, ברעיונות שלהם לגבי הפרוטוקול החדש, כדי שיהפכו אותו לתקן. תקן פירושו שיש מסמך בשם RFC (קיצור של Request For Comments) שבו מתוארים כל הפרטים הקשורים לפרוטוקול.

בשנת 2021 ה-IETF מסיים את גיבוש התקן, ומתוך הכרה בחשיבות ובחדשנות שלו מעניק ל-RFC של QUIC את המספר העגול 9000. התקן מגדיר את פורט 443 בתור הפורט של QUIC.

נסקור בקצרה את השינויים מרכזיים שמציג QUIC:

התקשורת עוברת מעל UDP. המשמעות היא ש-QUIC לא יכול יותר להתבסס על האמינות שמספק TCP, אלא צריך לדאוג בעצמו לכך שהתקשורת תהיה אמינה. את הרעיון הבסיסי של יצירת אמינות QUIC לקח מ-TCP ולכן נזהה אותו מיד. זרם המידע שנשלח מחולק לחלקים ממוספרים. הצד המקבל מרכיב אותם יחד לפי הסדר והצד השולח מצפה לאישורי קבלה.

TLS 1.3 משולב בתוך QUIC. מה המשמעות של "משולב בתוך"? תאורטית היה אפשר ש-QUIC יבנה שכבה של אמינות מעל UDP, ומעל השכבה הזו יעבור TLS 1.3 או כל פרוטוקול אבטחה אחר. אבל לא כך. מה שמיד נראה בהסנפה, הוא ש-QUIC מעביר ממש רשומות של TLS 1.3. נזהה את הרשומות המוכרות לנו מתהליך ה-Handshake. כלומר פרוטוקול שנמצא מעל QUIC מקבל ממנו סוקט שהוא גם אמין וגם מאובטח.

הצפנה של ה-Header. ב-TLS 1.3, למרות ההצפנה, היו דברים מסויימים שאפשר היה לקרוא בגלוי. לדוגמה, את ה-TCP Header. מה אפשר לעשות עם מספרי SEQ ו-ACK? הרי מידע לא עובר שם. ובכן יש מה לעשות. אפשר להשתמש בהם כדי לסדר לפי הסדר את זרם המידע המוצפן. זה אמנם לא מפענח את

ההצפנה, אבל עוזר בתור שלב ראשון והכרחי לשבירת המפתח. פעולה נוספת שאפשר לבצע היא מעקב אחרי מעבר של תקשורת מערוץ פיזי אחד לערוץ פיזי אחר. למרות המעבר, צפוי שה-SEQ וה-ACK ימשיכו מהמקום שבו הם היו. מחקרים שונים מצאו גם שאפשר לזהות תעבורה של שרתים מסויימים, לדוגמה נטפליקס, לפי מאפיינים של TCP.

גרסה חדשה של HTTP. גרסת HTTP/2 היתה צריכה לטפל בדברים שלא היו קיימים בשכבות שמתחתיה, אבל QUIC כן עושה. הנה שלוש דוגמאות:

- ב-HTTP/2 יש כפי שראינו Stream ID. אך QUIC כבר כולל Stream ID.
- למדנו על HPACK, דחיסה של ה-Header שקיימת ב-HTTP/2. ב-QUIC יש דחיסה משלו, QPACK.
- HTTP/2 ביצע לפעמים PINGים ברמת שכבת האפליקציה, בתור Keep Alive, במקום המנגנון הלא תמיד עובד של TCP. גם זה משהו ש-QUIC כבר מבצע.

מסיבות אלו וסיבות נוספות, מעל QUIC עוברת גרסה חדשה, HTTP/3.

מעבר מ-TCP ל-QUIC

פרוטוקול QUIC עובד מעל פורט 443 של UDP. הבעיה הראשונה של לקוחות היא לדעת אילו שרתים תומכים ב-QUIC?

ללקוח יש שתי אפשרויות. האחת, לזכור ששרת מסויים תומך ב-QUIC. אם כבר ביצענו התקשרות QUIC מול שרת, בפעם הבאה ננסה אותו שוב ב-UDP 443.

האפשרות השניה היא להקים התקשרות TCP רגילה עם TLS מגרסה 1.2 או 1.3, ולבחון את שדה ה- Alt Svc, קיצור של Alternative Service. שדה זה הוא אחת מהאופציות של פרוטוקול HTTP/2. שרת שתומך ב-QUIC יודיע שם על תמיכה ב-"h3", כמו בדוגמה הבאה:

```
Header: alt-svc: h3=":443"; ma=2592000,h3-29=":443"; ma=2592000
    Name Length: 7
    Name: alt-svc
    Value Length: 46
    Value: h3=":443"; ma=2592000,h3-29=":443"; ma=2592000
```

[QUIC Header]

הורידו את קובץ ההסנפה ואת קובץ המפתחות שבלינק הבא:

https://data.cyber.org.il/networks/QUICfiles.zip

בשלב ראשון העלו את קובץ ההסנפה *ללא* המפתחות.

פקטה 3180 היא נקודת ההתחלה שלנו. הלקוח פונה לשרת שתומך ב-QUIC ומתחיל בהקמת קשר.

נקליק קליק ימני על הפקטה ונבחר Follow UDP Stream.

נפתח את הפקטה. אנחנו מבחינים בגישה לפורט 443 מעל UDP, ומעל זה QUIC IETF. כזכור IETF הוא צוות המשימה שהפך את פרוטוקול QUIC לתקן. כלומר Wireshark מזהה שזוהי הגרסה שתוקננה ולא גרסה שונה, כגון ה-QUIC של גוגל:

```
> Frame 3180: Packet, 1292 bytes on wire (10336 bits), 1292 bytes captu
> Ethernet II, Src: HonHaiPrecis_25:11:f9 (9c:30:5b:25:11:f9), Dst: Tec
> Internet Protocol Version 4, Src: 192.168.1.103, Dst: 142.250.75.110
> User Datagram Protocol, Src Port: 62487, Dst Port: 443
> QUIC IETF
```

נפתח את QUIC:

```
v QUIC IETF
  > QUIC Connection information
    [Packet Length: 1250]
    1... .... = Header Form: Long Header (1)
    .1.. .... = Fixed Bit: True
    ..00 .... = Packet Type: Initial (0)
    [.... 00.. = Reserved: 0]
    [.... ..00 = Packet Number Length: 1 bytes (0)]
    Version: 1 (0x00000001)
    Destination Connection ID Length: 8
    Destination Connection ID: a288782ad708fa67
    Source Connection ID Length: 0
    Token Length: 0
    Length: 1232
    [Packet Number: 1]
    Payload [...]: dc700a244edace6c1aff272ec2c1c013b70f43b96092
```

זהו ה-Header של QUIC. נעבור על השדות המעניינים:

- סוג ה-Header: Long Header משמש בהקמת קשר. לאחר מכן יש שימוש ב-Short Header חסכוני יותר בבתים.
- סוג הפקטה: לפנינו סוג Initial, שמשמש לחלק הלא מוצפן בתהליך ה-Handshake. כזכור ב-TLS 1.3 יש מעבר למצב מוצפן מיד לאחר קביעת הסוד המשותף. ב-QUIC, הפקטות הלא מוצפנות הן מסוג Initial ואילו המוצפנות נקראות Handshake. מיד נראה אותן.
- Destination Connection ID (בקיצור DCID): השדה המעניין ביותר ב-Header. התקשורת מתבצעת מעל UDP ולכן צריך מזהה כלשהו של הקישור, שיאפשר לקשר בין פקטות של אותו קישור. המזהה

נקבע אקראית על ידי הלקוח, אך כאשר השרת יחזיר תשובה הוא יבחר במזהה אחר והלקוח ימשיך עם המזהה שהשרת בחר. גם את זה נראה מיד.

- שדה אורך של ה-DCID. כפי שרואים, שדה האורך הוא 8 ואכן ה-DCID כולל שמונה בתים (שש עשרה ספרות הקסדצימליות).

QUIC Frames

נעבור למידע עצמו, שם נמצאים הדברים המעניינים. הדבר הראשון ששמים אליו לב, הוא שלמרות שמדובר עדיין במידע לא מוצפן, הוא עבר תהליך ששינה אותו. זו לא הצפנה, אלא ערבול של הבתים באופן שמוגדר בתקן. לכן Wireshark יכול לשחזר את המידע גם בלי מפתחות הצפנה. אם נבחר בתצוגה באפשרות Decrypted QUIC נראה את הבתים של המידע הלא מעורבל:

```
0000  01 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 01 00
Packet (1292 bytes)   Decrypted QUIC (1215 bytes)
```

מה יש במידע עצמו?

המידע מחולק לחלקים שנקראים פריימים - Frames. יש פריימים מסוגים שונים, כאשר בפקטה הנוכחית יש שלושה סוגים:

- PING: זוהי המקבילה ל-HTTP PING. בית בודד שערכו x010 ומטרתו היא פשוט לשמור על Keep Alive עם השרת.
- PADDING: ריפוד באפסים, כדי שהפקטה תגיע לאורך מסויים.
- CRYPTO: זהו הפריים המעניין אותנו. פריימים של קריפטו נושאים את ה-Handshake של TLS.

נתבונן בפריים הקריפטו הראשון:

```
CRYPTO
    Frame Type: CRYPTO (0x0000000000000006)
    Offset: 976
    Length: 827
    Crypto Data
    [Reassembled PDU in frame: 3181]
```

לאחר השדה של סוג הפריים, יש את ההיסט ואת האורך. מה זה אומר?

כדי להבין זאת נבצע תרגיל מחקר קטן. נעבור על פקטה 3180 ועל פקטה 3181 ונחפש את כל הפריימים מסוג קריפטו.

ניצור רשימה של כל ההיסטים וכל האורכים שיש בהם, ונחשב גם את הסכום של ההיסט + אורך:

Offset	Length	Total
976	827	1803
70	1	71
0	70	70
71	1	72
853	33	886
208	34	242
886	17	903
632	221	853
72	11	83
242	7	249
945	5	950
950	26	976
903	42	945
249	383	632
83	125	208

נמיין את השורות על פי הערך בטור ההיסט, מהקטן לגדול:

Offset	Length	Total
0	70	70
70	1	71
71	1	72
72	11	83
83	125	208
208	34	242
242	7	249
249	383	632
632	221	853
853	33	886
886	17	903
903	42	945
945	5	950
950	26	976
976	827	1803

רואים שכל פריים מתחיל בהיסט שבו מסתיים הפריים שלפניו. המסקנה היא ש-QUIC מחלק את המידע בין פריימים שונים, אך כולל בתוכו את המידע הדרוש כדי שהצד הקולט יוכל להרכיב מחדש את הפאזל. מדוע מתבצעת החלוקה הזו? כדי להקשות על מי שמאזין לנו. כרגע הפריימים אינם מוצפנים ולכן אפשר לעקוב אחרי שדה ה-Offset ולהרכיב את המידע בסדר הנכון, אך כאשר הפריימים יעברו למוצפן זה יהיה בלתי אפשרי אפילו לסדר את המידע לפי הסדר.

אנחנו גם רואים פה שימוש ראשון ל-DCID. הצד המקבל קיבל שתי פקטות UDP, שרק החיבור שלהן מחלץ את המידע. הצד המקבל ידע לקשור את הפקטות זו לזו באמצעות ה-DCID שלהן.

כדי לצפות בתוכן ה-TLS Handshake נעבור לפקטה 3181:

```
CRYPTO
  Frame Type: CRYPTO (0x0000000000000006)
  Offset: 83
  Length: 125
  Crypto Data
  [15 Reassembled QUIC CRYPTO Data Fragments (1803 bytes):
  TLSv1.3 Record Layer: Handshake Protocol: Client Hello
    Handshake Protocol: Client Hello
      Handshake Type: Client Hello (1)
```

בסוגריים המרובעים, Wireshark מציין בפנינו שפריים זה הוא הרכבה של פריימים קודמים, בדיוק כפי שראינו. לאחר מכן מופיע המידע שהורכב מכל הפריימים, ואז מתברר שזו הרשומה המוכרת לנו של Client Hello.

ה-SNI, Server Name Indication, הוא www.youtube.com, מתברר מהו האתר שאיתו אנחנו מקימים קשר. שימו לב שאנחנו עדיין לא במוצפן, המידע לאן אנחנו גולשים ב-QUIC נגיש לכל מי שיש לו גישה לפקטות שלנו.

רשומה זו כוללת נוסף על השדות המוכרים לנו שני שדות מעניינים.

שדה ALPN - Application Layer Protocol Negotiation. נכנס ונגלה כי הפרוטוקול המוצע על ידי הלקוח הוא H3, קיצור של HTTP/3.

שדה quic_transport_parameters מגדיר הגדרות שונות כגון כמות המידע המקסימלית שלקוח יכול לשלוח לשרת, כמות זרמי המידע שאפשר להקים מול השרת בו זמנית. לקוח היה רוצה כמובן לשלוח לשרת כמה שיותר מידע ולעבוד מול השרת בכמה שיותר ערוצים במקביל, אך השרת משרת לקוחות רבים ולא רוצה שלקוח יחיד "יתעלק" עליו ולא ישאיר לו משאבים ללקוחות אחרים.

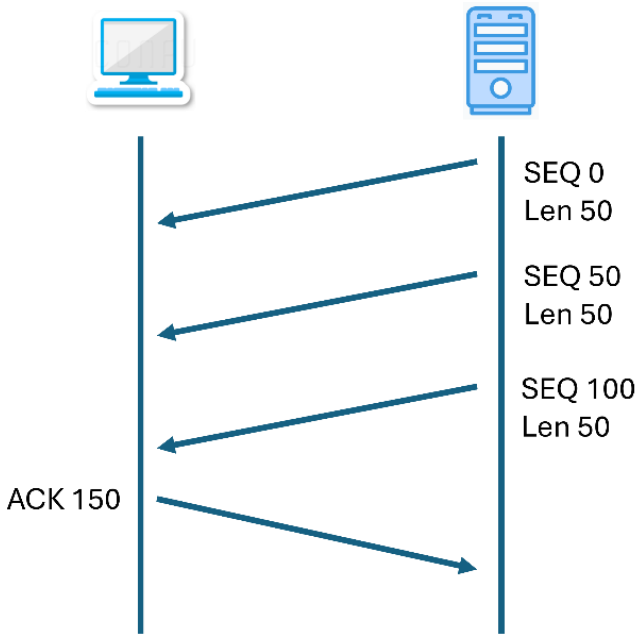
אנחנו עוברים לפקטה הבאה בזרם, פקטה 3216. השרת עונה ללקוח.

הפריים הוא מסוג חדש - ACK. לפנינו מנגנון חלופי ל-ACK של TCP. לכן, לפני שנסביר על השדות השונים, נעצור לרגע כדי להסביר על ה-ACK של TCP והבעיות שיש בו ושאותן QUIC רוצה לפתור.

TCP ACK

מנגנון האמינות של TCP מבוסס על כך שלכל בית (Byte) של שכבת האפליקציה יש מספר סידורי עוקב, Sequential number, או בקיצור SEQ. הצד המקבל את הבתים מאשר - Acknowledge או בקיצור ACK - את המספר הסידורי האחרון שהוא קיבל באופן רציף. לדוגמה, אם נשלח ACK 150 לדוגמה, זה אומר שהתקבלו כל הבתים עד 149 וכעת הצד המקבל מצפה לבית מספר 150. עד כאן מה שלמדנו בפרק אודות TCP. אך זה לא הסוף.

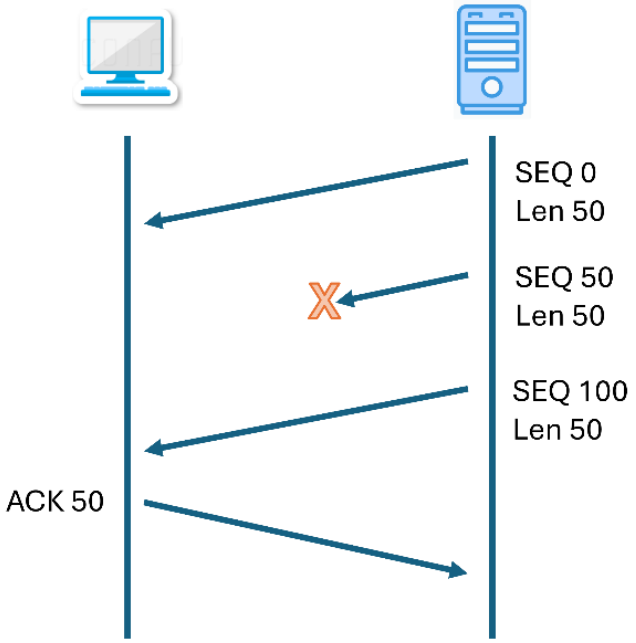
ה-ACK הוא Cumulative, או "שיתופי". נניח שהיו שלוש פקטות, כל אחת באורך של 50. הראשונה התחילה ב-SEQ 0 (והסתיימה ב-SEQ 49), השניה התחילה ב-SEQ 50 והשלישית ב-SEQ 100. במקרה זה ACK 150 מאשר את כל שלושת הפקטות. גם אם לא נשלחו עליהן ACKים, או שה-ACKים נפלו בדרך עקב תקלה, מי ששלח את שלושת הפקטות יכול להיות בטוח ששלושתן התקבלו:



החסרון של ה-Cumulative ACK, הוא במקרה שבו פקטה הולכת לאיבוד.

אם לצד המקבל יש "חור" בפקטות שהוא קיבל, אין לו דרך להתקדם עם ה-ACK-ים. נחזור לדוגמה של שלוש הפקטות ממקודם. נניח שהפקטה השניה לא התקבלה. הצד המקבל יכול לשלוח ACK 50 על הפקטה הראשונה. מה לגבי השלישית? אם הצד המקבל ישלח ACK 150, הצד השולח יסיק מזה שגם הפקטה השניה התקבלה והוא לא ישלח אותה שוב.

לכן הצד המקבל "תקוע" על 50. התקיעה הזו, שנגרמת בעקבות פקטה שנפלה, היא שגורמת לתופעת ה-Head of Line Blocking שסקרנו:

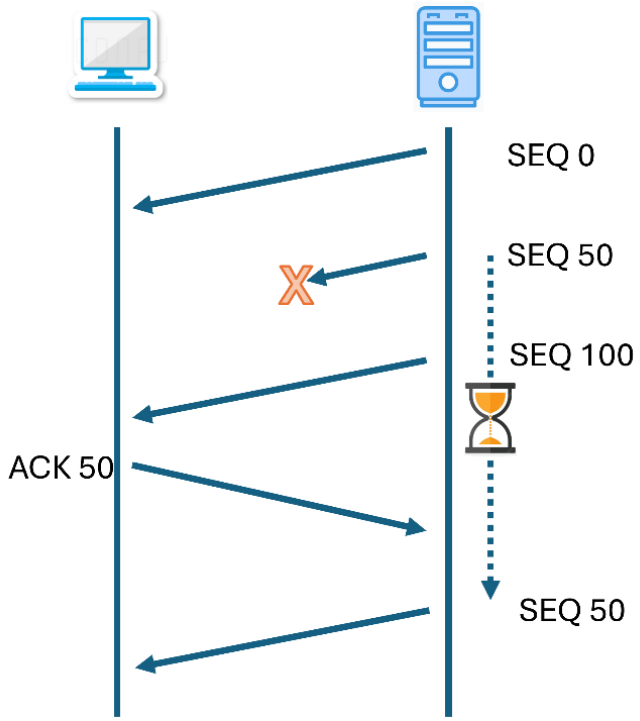


במימושים ישנים של TCP, הצד הקולט היה זורק את כל הפקטות שהגיעו אחרי הפקטה החסרה, לא שולח עליהן ACK גם אם הפקטה החסרה הושלמה, ובכך מאלץ את השולח לשלוח אותן שוב גם אם הן הגיעו תקין. כיום לא סביר שהפקטות החדשות ייזרקו, במקום זאת הן יישמרו בזיכרון וימתינו לכך שהפקטה החסרה תגיע. אז, יבוצע Cumulative ACK על כל הפקטות שהגיעו עד כה. ועדיין גם במימושים אלו בעיית ה-Head of Line Blocking קיימת. בעקבות ההמתנה לפקטה החסרה, התעכבו כל הפקטות הבאות, כולל פקטות של Streamים שאינם קשורים לפקטה החסרה. TCP לא מעביר את המידע לשכבת האפליקציה עד שהפקטה החסרה לא הושלמה.

נסקור מה עושה הצד השולח בתרחיש כזה. כיצד הוא יודע שפקטה נפלה בדרך?

עבור כל פקטה, הצד השולח "פותח שעון" ומחכה פרק זמן מוגדר שנקרא Timeout. אם עד ה-Timeout לא מגיע עליה ACK, הצד השולח מבצע Retrasmit, שליחה חוזרת.

האיור הבא ממחיש Retransmit של פקטה שלא התקבל עליה ACK לאחר Timeout:



מה שמעלה את השאלה: כמה זמן ממתינים ל-ACK? כיצד קובעים Timeout "טוב"?

נחשוב על המקרים השונים. אם קבענו זמן המתנה ארוך מדי, אז נבזבז זמן בהמתנה ל-ACK. יש זמן שאם עד אז ה-ACK הגיע, הוא כבר לא יגיע. מצד שני אם קבענו זמן המתנה קצר מדי, מה שנקרא Premature Timeout, אז סביר שנכריז על Retransmit גם על חבילות שהגיעו ליעד תקין. ה-ACK של חבילות אלו בדרך ופשוט טרם התקבל. גם זו בעיה, כי השולח יבזבז משאבים על שליחות חוזרות, במקום להתקדם עם שליחה של פקטות חדשות.

הפתרון הוא שהשולח מנסה להעריך את ה-RTT בינו לבין המקבל, ולקבוע זמן המתנה שהוא ארוך במקצת מה-RTT. לדוגמה, אם ה-RTT בין הצדדים הוא שניה, אז לא הגיוני לחכות ל-ACK עשר שניות וגם לא הגיוני לחכות חצי שניה. זמן המתנה של שניה ועוד ספייר קטן כגון עשירית שניה הוא זמן מוצלח. הספייר נדרש גם בגלל שלעיתים פקטות מתעכבות קצת בדרך (נתקלנו בכך כאשר בתחילת לימודי הרשתות ביצענו פינג לשרת כלשהו, ראינו שהזמן שמתקבל עד לתשובה יכול להיות שונה בין פקטה לפקטה) וגם בגלל שמסובך להעריך בצורה מדוייקת את ה-RTT בין הצדדים.

איך הצד השולח יכול להעריך את ה-RTT בינו לבין הצד המקבל בצורה טובה ככל האפשר?

השיטה מבוססת על מדידת הזמן שלקח להודעות קודמות שלו לקבל ACK.

בתחילת התקשורת השולח יקבע זמן Timeout ארוך יחסית, או שייקח אומדן ראשוני מפרק הזמן שלקח לעשות Three Way Handshake. ככל שמתקבלים יותר ACKים כך השולח יכול לדייק את ההערכה שלו על ה-RTT וכך לשפר את קצב התקשורת בין הצדדים.

אך ההערכה של ה-RTT אינה פשוטה. ב-TCP יש מספר דברים שמקשים על החישוב. נסקור את הבעיות, וחלק זה חשוב במיוחד להבנה מכיוון שכאשר נסקור את מנגנון ה-ACK של QUIC נראה כיצד הוא נבנה כדי להתמודד עם בעיות אלו.

בעיה ראשונה, **ה-Cumulative ACK "מותח" את זמן ה-ACK של חלק מהפקטות**. כאשר סקרנו את Cumulative ACK, נתנו דוגמה שבה ACK יחיד מאשר שלוש פקטות. הפקטה בעלת SEQ 0 קיבלה את ה-ACK רק לאחר ששתי פקטות נוספות הגיעו לצד המקבל בהצלחה. הזמן שלקח האישור שלה מוטה כלפי מעלה ולא משקף את ה-RTT.

בעיה שניה, **לצד הקולט לוקח זמן מרגע שהוא מקבל את הפקטה ועד שהוא שולח את ה-ACK**. הזמן שלוקח לצד השולח לקבל את ה-ACK על פקטה מורכב מחיבור של ה-RTT עם זמן נעלם, שהוא כמות הזמן שלוקח לצד הקולט לשלוח את ה-ACK. לדוגמה, שלחנו פקטה וקיבלנו ACK אחרי 5 מילישניות. יכול להיות שה-RTT הוא 5 מילישניות, אבל אם לצד השני לקח 2 מילישניות לענות לנו, אז ה-RTT הוא בעצם רק 3 מילישניות. למה יש שיהוי בצד המקבל? קודם כל, כל חישוב לוקח זמן כלשהו ויכולים להיות עיכובים בגלל עומס. אבל יכול גם להיות שהצד המקבל משתהה בכוונה בשליחת ה-ACK. הצד המקבל יכול לקוות שאם הוא יחכה עוד קצת, תתקבל עוד פקטה מהשולח ואז פקטת ה-ACK שלו כבר תאשר את כמה פקטות יחד, דבר שיחסוך לו שליחה של פקטה. אפשרות נוספת שעומדת בפני הצד המקבל היא להעלות את שדה ה-ACK בפקטת המידע הבאה שהוא ישלח, וכך לחסוך שליחה של פקטת ACK נפרדת. הצד המקבל יכול לחשוב "יש לי פקטה שאני עומד לשלוח תיכף, אז במקום לשלוח עכשיו פקטת ACK, שווה להמתין רגע".

בעיה שלישית, **ב-ACK על Retrasmit, אי אפשר לדעת בבטחון האם ה-ACK הוא על הפקטה המקורית או על הפקטה החוזרת**. השדות השונים של TCP יהיו זהים ולצד המקבל אין דרך לדעת על איזו פקטה התקבל ה-ACK. נכון שרוב הסיכויים הם שה-ACK הוא על הפקטה החוזרת, אבל תמיד יכולות להיות הפתעות. כמובן שלהחלטה על איזו פקטה התקבל ה-ACK יש השפעה גדולה על הסקת ה-RTT.

נסכם את הלקחים עבור מי שמתכננים פרוטוקול חדש:

- מועיל לשמר את היכולת לאשר בבת אחת מספר פקטות. זה מקטין את העומס שיוצרים ה-ACKים וחוסך שידורים חוזרים של פקטות שה-ACK שלהן נפל
- מצד שני, צריך לתקן את הבעיה ש"חור" עוצר את העיבוד של הפקטות הבאות, גם אם הן שייכות למשאב אחר, שכל הפקטות שלו הגיעו תקין.
- כדאי לעזור לצדדים לגלות איפה יש "חורים" בקבלת הפקטות ולהשלים רק אותם.

- כדי לשפר את האומדן של ה-RTT, כדאי להעביר מידע כמה זמן ה-ACK השתהה. כלומר כמה זמן עבר מרגע שהפקטה התקבלה ועד שיצא עליה ACK.
- במקרה של Retransmit, צריך למצוא דרך להבדיל בין ACK על הפקטה המקורית לבין ACK על השידור החוזר שלה.

כעת כשהבנו את המטרות הללו, אפשר להתקדם ולבחון איך נראית פקטת ACK של QUIC.

QUIC ACK

חזרנו מסקירת ה-TCP אל עולם ה-QUIC.

הדבר הראשון שנבחין בו ב-QUIC הוא שאין מספרי SEQ. במקום זאת, לכל פקטה יש מספר סידורי. המספר לא נשלח בצורה גלויה אך Wireshark יודע לחלץ אותו ולהציג אותו. בשונה מ-TCP, שבו ה-ACKים תואמים לכמות הבתים שהתקבלו, ב-QUIC ה-ACKים תואמים למספרי הפקטות.

הבדל נוסף מ-TCP, הוא שמספרי פקטות לא חוזרים על עצמם כאשר משדרים מחדש פקטה שנפלה בדרך. זהו תיקון של המנגנון הבעייתי שסקרנו ב-TCP, שבו פקטה שנשלחה מחדש נראית זהה לפקטה המקורית, מה שמקשה על הצד שקיבל עליה ACK לדעת אם הוא על השליחה המחודשת או על המקור. אנחנו כבר רואים שהשינוי הזה מתקן את אחד הלקחים מ-TCP.

פריים מסוג ACK מעביר את הדברים הבאים:

- כמה זמן הצד המקבל השתהה בין קבלת הפקטה ועד ששלח את ה-ACK. המידע הזה מתקן לקח נוסף מ-TCP, ומאפשר חישוב מדויק יותר של ה-RTT.
- לאילו פקטות יש אישור קבלה
- אילו "חורים" יש, כלומר אילו מספרי פקטות חסרים בצד המקבל

לדוגמה, נניח שלצד המקבל הגיעו פקטות 1,2,3,4,7,8. פקטות 5,6 חסרות. ה-ACK של QUIC ידווח הן על הפקטות שהגיעו והן על אלו שלא הגיעו. הדרך שבה הדיווח יתבצע תהיה באמצעות שדות שיענו על השאלות הבאות:

- מהי הפקטה בעלת המספר הגבוה ביותר שהתקבלה?
- כמה פקטות התקבלו לפניה, בלי "חורים"?

בדוגמה שלנו, התשובה לשאלה הראשונה היא 8, והתשובה לשאלה השניה - 1. זאת מכיוון שישנה פקטה אחת, פקטה מספר 7, שהתקבלה לפני פקטה 8 בצורה צמודה, בלי "חור" באמצע.

- כמה אזורים של חורים ישנם?

כעת יש שתי אפשרויות. או שיש "חור" או שאין. אם אין חור, הכמות תהיה 0. אם יש אזורים של חורים, המספר שלהם ידווח פה. שימו לב שחורים צמודים מדווחים בתור אזור אחד. בדוגמה שלנו, חסרות פקטות 5,6. הן צמודות ולכן הערך בדיווח יהיה "1".

עבור כל אזור של חורים:

- כמה פקטות נמצאות בחור?
- כמה פקטות התקבלו תקינות לפני החור?

הדרך שבה המספרים הללו מועברים היא קצת טריקית להבנה, אבל הדוגמה שלנו תבהיר אותה.

המידע שקבלנו עד כה מה-ACK מוסר לנו שפקטות 7,8 התקבלו תקין ושישנו חור אחד. המסקנה היא שפקטה 6 לא התקבלה. כדי למסור שגם פקטה 5 לא התקבלה, הדיווח יהיה "1". כלומר, חסרה פקטה אחת נוסף על הפקטה שבסוף החור.

המסקנה הבאה חייבת להיות שפקטה 4 כן התקבלה תקין. כדי למסור שגם שלושת הפקטות לפניה התקבלו תקין, ידווח "3".

כלומר בדוגמה שלנו הדיווח יהיה כך:

- פקטה אחרונה שהתקבלה - 8
- כמות פקטות צמודות לפניה - 1
- כמות רווחים - 1
- כמות פקטות צמודות לסוף הרווח - 1
- כמות פקטות צמודות לסוף האזור הבא שאינו חור - 3

נבחן את פקטה 3229:

```
ACK
  Frame Type: ACK (0x0000000000000002)
  Largest Acknowledged: 4
  ACK Delay: 62
  ACK Range Count: 0
  First ACK Range: 3
```

שדה ה-Largest Acknowledged מציין "4", כלומר זו הפקטה עם המספר הגבוה ביותר שהתקבל.

ה-First ACK Range הוא 3, כלומר התקבלו 3 פקטות צמודות לפקטה שמספרה הוא 4. כלומר מאושרות הפקטות 1,2,3,4.

ה-ACK Range Count הוא 0, מכיוון שאין חורים עד כה.

ה-ACK Delay הוא 62, מספר זה מאפשר לחשב את כמות המיקרו שניות (מליוניות השניה) שהמקבל התעכב בשליחה.

בזרם המידע שלפנינו יש גם חורים. אך כדי לראות דיווח ACK עם חורים נצטרך להוסיף ל-Wireshark את קובץ המפתחות שלנו. נבצע זאת ונבחן את פקטה 3401:

```
ACK
  Frame Type: ACK (0x0000000000000002)
  Largest Acknowledged: 12
  ACK Delay: 1
  ACK Range Count: 1
  First ACK Range: 2
  Gap: 0
  ACK Range: 3
```

התקבלה פקטה מקסימלית 12.

ה-First ACK Range הוא 2, כלומר התקבלו שתי פקטות לפני פקטה 12. מכאן שפקטות 10,11,12 התקבלו.

ה-ACK Range Count מדווח על חור אחד. כלומר פקטה 9 היא חור.

ה-Gap הוא 0, כלומר יש אפס פקטות לפני פקטה 9 שהן חור. כלומר פקטה 8 התקבלה.

ה-ACK Range הוא 3, כלומר יש 3 פקטות שהתקבלו לפני פקטה 8. מכאן שפקטות 5,6,7,8 התקבלו.

ומה לגבי הפקטות שקודמות לפקטה 5? הן כבר קיבלו ACK, אין צורך לחזור על כך, ויותר מזה - QUIC לא מאפשר לדווח בתור "חור" על פקטה שכבר דווח עליה ACK.

נסכם את מה שראינו ונשווה לדברים שרצינו לשפר ב-TCP:

1. **לקח מ-TCP**: מועיל לשמר את היכולת לאשר בבת אחת מספר פקטות. זה מקטין את העומס שיוצרים ה-ACKים וחוסך שידורים חוזרים של פקטות שה-ACK שלהן נפל. **פתרון של QUIC**: שימרנו את היכולת לאשר כמה פקטות בו זמנית.
2. **לקח מ-TCP**: מצד שני, צריך לתקן את הבעיה ש"חור" עוצר את העיבוד של הפקטות הבאות, גם אם הן שייכות למשאב אחר, שכל הפקטות שלו הגיעו תקין. **פתרון של QUIC**: טרם ראינו זאת בהסנפה, אך לכל משאב שנשלח יש Stream ID, בדומה למה שראינו ב-HTTP/2. באמצעות שימוש ב- Stream ID, QUIC יכול להעלות לשכבת האפליקציה פקטות ששייכות לאותו משאב. כך, גם במקרה של "חור" בקבלה, משאב שהפקטות שלו לא שייכות ל"חור" יכול להמשיך עיבוד בלי שיהוי.
3. **לקח מ-TCP**: כדאי לעזור לצדדים לגלות איפה יש "חורים" בקבלת הפקטות ולהשלים רק אותם. **פתרון של QUIC**: ה-ACK כולל גם מידע לגבי חורים.

4. **לקח מ-TCP**: כדי לשפר את האומדן של ה-RTT, כדאי להעביר מידע כמה זמן ה-ACK השתהה. כלומר כמה זמן עבר מרגע שהפקטה התקבלה ועד שיצא עליה ACK. **פתרון שלQUIC** : זמן השיהוי בשליחת ה-ACK נשלח כחלק מה-ACK.

5. **לקח מ-TCP**: במקרה של Retransmit, צריך למצוא דרך להבדיל בין ACK על הפקטה המקורית לבין ACK על השידור החוזר שלה. **פתרון של QUIC**: כל פקטה מקבלת מספר סידורי משלה, המספר לא חוזר על עצמו גם אם יש שידור חוזר. נשאלת אם ככה השאלה, אם מספר הפקטה שונה מהמספר המקורי, איך הצד המקבל יודע שמדובר ב-Retransmit? ובכן, השיוך מתבצע לפי מאפיינים אחרים שהזכרנו. לכל משאב יש Stream ID וכל משאב מחולק לחלקים, שנשלחים עם שדה שאומר מה ההיסט שלהם מתחילת המשאב. לכן מי שמקבל את הצירוף של היסט יחד עם Stream ID, יכול לדעת שזה מילוי של "חור".

תהליך ה-Handshake והמעבר למוצפן

כפי שרואים בהסנפה, לאחר פקטות מסוג Initial מופיעות פקטות מסוג Handshake. ההבדל ביניהן הוא המעבר למוצפן. פקטת ה-Server Hello כוללת כזכור גם את החלק של השרת ביצירת הסוד המשותף. לאחר פקטה זו הלקוח יכול כבר להצפין את התקשורת ומתחיל תהליך ה-Handshake המוצפן (אם אינכם רואים אותו, וודאו שהזנתם את קובץ המפתחות).

```
QUIC IETF
  QUIC Connection information
  [Packet Length: 1250]
  1... .... = Header Form: Long Header (1)
  .1.. .... = Fixed Bit: True
  ..10 .... = Packet Type: Handshake (2)
  [.... 00.. = Reserved: 0]
  [.... ..00 = Packet Number Length: 1 bytes (0)]
  Version: 1 (0x00000001)
  Destination Connection ID Length: 0
  Source Connection ID Length: 8
  Source Connection ID: e288782ad708fa67
```

ה-Handshake המוצפן מתחיל לאחר ה-Server Hello, פקטה מספר 3219, ומסתיים עם Client Finished, שמתקיים בפקטה 3402.

כעת אפשר להבין יותר טוב את הסיפור של פקטה מספר 9, אותה פקטה שראינו שהלקוח לא אישר אותה עם ACK. נבדוק מה השרת שלח. פקטה מספר 3374 כוללת שתי פקטות של QUIC בתוכה, פקטת QUIC

מספר 8 ופקטת QUIC מספר 9. אנחנו כבר לומדים מזה משהו: פקטת UDP יכולה לכלול יותר מפקטת QUIC אחת.

אם פקטת QUIC מספר 9 הגיעה, אז מדוע הלקוח שלח עליה "חור" במקום לאשר אותה? נשים לב לכך שהשרת שלח שתי פקטות HTTP/3 עוד לפני שה-Handshake הסתיים.

No.	Time	Source	Destination	Protocol	Info
3229	17.899649	192.168.1.103	142.250.75.110	QUIC	Initial, DCID=e288782ad708fa67, PKN: 3, ACK, PADDING
3272	17.912981	192.168.1.103	142.250.75.110	QUIC	Handshake, DCID=e288782ad708fa67, PKN: 5, PADDING, PING
3364	17.940155	192.168.1.103	142.250.75.110	QUIC	Handshake, DCID=e288782ad708fa67, PKN: 7, PADDING, PING
3368	17.944405	142.250.75.110	192.168.1.103	QUIC	Handshake, SCID=e288782ad708fa67, PKN: 5, CRYPTO
3370	17.944405	142.250.75.110	192.168.1.103	QUIC	Handshake, SCID=e288782ad708fa67, PKN: 6, CRYPTO
3372	17.944405	142.250.75.110	192.168.1.103	QUIC	Handshake, SCID=e288782ad708fa67, PKN: 7, CRYPTO
3374	17.944405	142.250.75.110	192.168.1.103	HTTP3	Protected Payload (KP0), PKN: 9, STREAM(3), SETTINGS
3375	17.944405	142.250.75.110	192.168.1.103	QUIC	Handshake, SCID=e288782ad708fa67, PKN: 10, ACK
3387	17.945140	192.168.1.103	142.250.75.110	QUIC	Handshake, DCID=e288782ad708fa67, PKN: 8, ACK
3390	17.945506	192.168.1.103	142.250.75.110	QUIC	Handshake, DCID=e288782ad708fa67, PKN: 9, ACK
3398	17.947346	142.250.75.110	192.168.1.103	QUIC	Handshake, SCID=e288782ad708fa67, PKN: 11, ACK, CRYPTO
3399	17.947346	142.250.75.110	192.168.1.103	HTTP3	Protected Payload (KP0), PKN: 13, STREAM(3), SETTINGS
3400	17.947749	192.168.1.103	142.250.75.110	QUIC	Handshake, DCID=e288782ad708fa67, PKN: 10, ACK
3401	17.947805	192.168.1.103	142.250.75.110	QUIC	Handshake, DCID=e288782ad708fa67, PKN: 11, ACK
3402	17.948218	192.168.1.103	142.250.75.110	QUIC	Handshake, DCID=e288782ad708fa67, PKN: 12, CRYPTO

השרת מקדם פקטות של שכבת האפליקציה, כך שברגע שתהליך ה-Handshake יסתיים כבר תהיה התחלה של מידע בידי הלקוח. הלקוח קיבל אותן אך לא מעבד אותן עד שהוא מסיים את העיבוד של ה-Handshake ולכן בשלב זה אין ביכולתו לדווח עליהן ACK. עם זאת הלקוח כן רוצה לדווח לשרת על קבלת יתר הפקטות של ה-Handshake. לפי התקן של QUIC הלקוח רשאי לדווח על חור ולעדכן על קבלה בהמשך.

לאחר ה-Handshake פקטות ה-QUIC נראות אחרת. נסקור את השינויים:

```
v QUIC IETF
  > QUIC Connection information
    [Packet Length: 70]
  v QUIC Short Header DCID=e288782ad708fa67 PKN=13
      0... .... = Header Form: Short Header (0)
      .1.. .... = Fixed Bit: True
      ..0. .... = Spin Bit: False
      [...0 0... = Reserved: 0]
      [.... .0.. = Key Phase Bit: False]
      [.... ..00 = Packet Number Length: 1 bytes (0)]
      Destination Connection ID: e288782ad708fa67
      [Packet Number: 13]
      Protected Payload: 1572b913a4fd3d3264e0df7de171f1f98f09620d25b346df114d9abd235cac986e2651b38d0b99
  v STREAM id=2 fin=0 off=0 len=42 dir=Unidirectional origin=Client-initiated
    > Frame Type: STREAM (0x0000000000000008)
    > Stream ID: 2
      Stream Data: 00041b018001000006800400000740643301c0000017c3013a2ca7f92a80c000000ab1ddd02e0320ac04
> Hypertext Transfer Protocol Version 3
```

ראשית, ה-Header משתנה מ-Long ל-Short. המידע היחידי שיש בו הוא ה-Destination Connection ID. חשוב להדגיש, ש*כל* מה שמופיע לאחר מכן הוא מוצפן.

שנית, סוג פריים חדש - STREAM. מתווסף לסוגים שהכרנו קודם לכן, CRYPTO, ACK, PING, PADDING.

שלישית, ל-Stream יש Stream ID. הרעיון של Stream ID מוכר לנו מ-HTTP/2. הוא מאפשר לייצר הפרדה בין משאבים שונים. ב-HTTP/2 ראינו שההפרדה בין המשאבים סובלת מבעיית ה- Head Of Line Blocking של TCP. המעבר ל-UDP מאפשר הפרדה מלאה. אם חסרות פקטות של משאב בעל Stream ID מספר 2, לדוגמה, אין לזה שום השפעה על Stream ID מספר 3. נציין לעצמנו שבעיית ה- Head Of Line Blocking נפתרה.

רביעית, HTTP/3. כבר אין צורך לבצע חלק מהדברים שביצע HTTP/2. כזכור ב-HTTP/2 יש שדה של Stream ID. כאשר QUIC לוקח על עצמו את המשימה, אפשר לייצר גרסה חדשה של HTTP, גרסה 3.

היררכיה של Connection, Stream, Packet, Frame

נעשה סדר בחלקים השונים שמרכיבים את QUIC.

פקטות UDP הן הבסיס. כדי לחבר בין פקטות UDP ששייכות לאותה תקשורת בין שרת ולקוח, קישור יש DCID, מזהה קישור. ה-DCID משותף הן לשרת והן ללקוח.

כל פקטת UDP יכולה להכיל פקטת QUIC אחת או יותר. לכל פקטת QUIC יש מספר, Packet Number, ייחודי.

כל פקטת QUIC מכילה פריים אחד או יותר של QUIC. פריימים יכולים להיות חלקים של מידע גדול יותר, ובמקרה כזה לכל פריים יהיה נתון ההיסט שלו מתחילת המידע ומה הגודל שלו. כך הצד המקבל יכול להרכיב בחזרה את המידע.

מידע של שכבת האפליקציה נשלח על גבי פריימים מסוג STREAM, שיש להם גם מזהה של Stream ID.

הדוגמה הבאה מראה פריים של Stream מסוים. נשים לב לכך שישנם שני מזהים שמאפשרים לצד המקבל להרכיב את המידע בצורה נכונה:

- ה-Stream ID שקובע לאיזה משאב שייך המידע בפריים
- ה-Offset, ההיסט, שקובע מה המיקום היחסי בתוך המשאב של המידע שבפריים

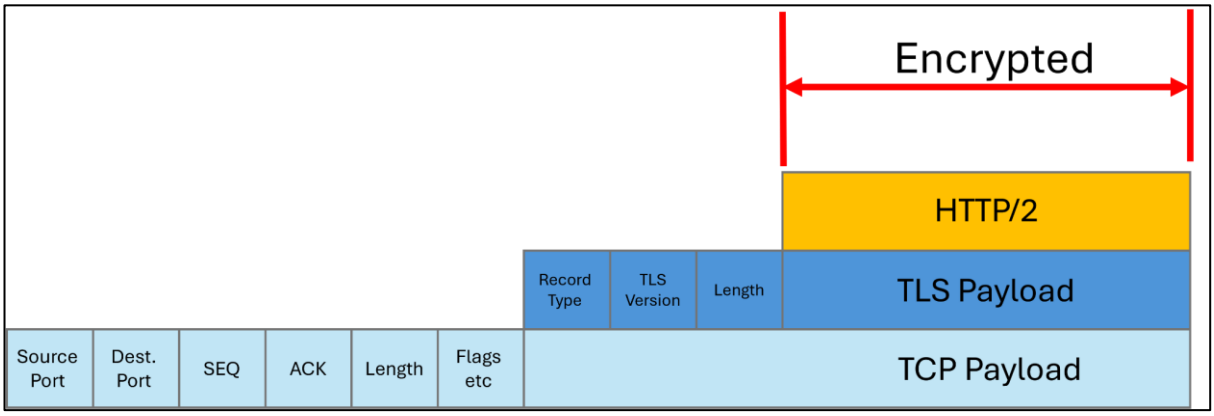
```
STREAM id=2 fin=0 off=42 len=12 dir=Unidirectional
  Frame Type: STREAM (0x000000000000000e)
  Stream ID: 2
  Offset: 42
  Length: 12
  Stream Data: 800f07000700753d302c2069
STREAM id=0 fin=0 off=0 len=1206 dir=Bidirectional
  Frame Type: STREAM (0x0000000000000008)
  Stream ID: 0
  Stream Data […]: 0149b10000d1508cf1e3c2fe8f6a6d8
```

במקרה של הפקטה שלפנינו, מדובר בפריים ששייך למשאב בעל המזהה Stream ID 2, ואשר המיקום היחסי שלו הוא 42 בתים מתחילת המשאב.

הצפנת Header-ים

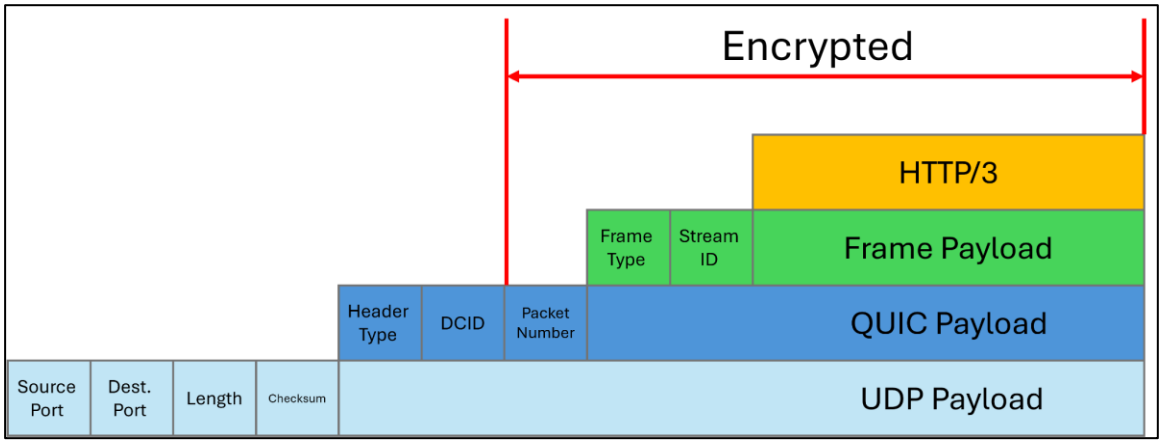
מה יכול לראות מי שמסניף QUIC בלי מפתחות ההצפנה?

האיורים הבאים ממחישים את ההבדלים בין HTTP מעל TLS מעל TCP, לבין HTTP מעל QUIC.



ב-HTTP/2, שעובר מעל TLS ומעל TCP, רק המידע של שכבת האפליקציה מוצפן. כלומר כל השדות של TCP עוברים בגלוי, בין היתר ה-SEQ. דבר זה מאפשר למי שקולט את התקשורת לסדר את הפקטות לפי הסדר, גם אם אין בידיו את המפתח.

ומה לגבי HTTP/3, מעל QUIC?



אם ב-TLS מעל TCP החלק המוצפן הוא רק המידע, ב-QUIC מוצפנים גם רוב ה-Header-ים. מי שתופס תעבורת QUIC בלי מפתח ההצפנה יכול לראות רק את ה-UDP Header, שכולל את מספרי הפורטים של השרת והלקוח, ואת ה-Connection ID.

כל יתר המידע שעשוי לשמש אפילו לא לפענוח אלא רק לסידור המידע המוצפן לפי הסדר הנכון (כלומר מחולק ל-Streamים שבתוכם הפריימים נמצאים לפי הסדר), כל יתר המידע הזה מוצפן.

מה לגבי ה-DCID? לכאורה נראה שמה שכן אפשר לעשות זה לאסוף יחד את כל הפקטות בעלות אותו ה-Connection ID. אולי בדרך כזו או אחרת זה יכול להיות שימושי לצורך מעקב.

אך למעשה, פרוטוקול QUIC מצפין באופן עקיף גם את ה-DCID.

בשלב מסויים בהתקשרות, השרת שולח ללקוח פריים בשם NEW CONNECTION ID ושבו הוא מציע ללקוח מספר Connection ID נוספים לשימושו. הפריים הוא מוצפן כמובן, ולכן מי שאין לו את מפתחות ההצפנה ושומר את המידע רק לפי ה-Connection ID, יוכל לעקוב אחרי ההתקשרות רק עד הנקודה שבה הלקוח החליט לדלג ל-Connection ID החדש. המשמעות היא שלמרות שמזהה הקישור אינו מוצפן, קשה מאד לעקוב אחרי מי שמדלג.

```
NEW_CONNECTION_ID
   Frame Type: NEW_CONNECTION_ID (0x0000000000000018)
   Sequence: 1
   Retire Prior To: 0
   Connection ID Length: 8
   Connection ID: e388782ad708fa67
   Stateless Reset Token: 9854498c25cc14fdba98651bbff77676
```

0-RTT "אמיתי"

כאשר למדנו מהו RTT, הפרדנו בין RTT של TLS לבין RTT של פרוטוקולים אחרים, ובפרט TCP. באותה נקודת זמן זה היה נראה צעד מוזר. בין כה וכה אנחנו עובדים מעל TCP, לכן איזו סיבה יש לספור את ה-RTTים בלי ה-RTT הנוסף שלוקח TCP? כעת כשעברנו ל-UDP הסיבה לכך מתבררת.

איך נראה 0-RTT "אמיתי"?

TLS 1.3 מאפשר כזכור חידוש של קישור קיים. לקוח שרוצה להתחבר מחדש לשרת, יכול לעשות זאת תוך זמן מוגבל באמצעות שימוש ב-Session Ticket.

בהתחברות המחודשת, הלקוח שולח כבר מידע של שכבת האפליקציה. אמנם ניתן להשתמש במפתח ההצפנה הישן להעביר רק כמות מוגבלת של בתים, מה שנקרא Early Data, אך במקביל השרת והלקוח עובדים על יצירת מפתחות חדשים.

כל עוד עבדנו מעל TCP היינו צריכים לשלם RTT נוסף לטובת Three Way Handshake.

פרוטוקול QUIC עושה שימוש ב-TLS 1.3, כך שהיכולת של RTT-0 מוטמעת בתוכו מלכתחילה. דבר זה מאפשר RTT-0 "אמיתי", שבו לקוח שרוצה לחדש התקשרות עם שרת יכול לשלוח מידע של שכבת האפליקציה ממש מעל הפקטה הראשונה שהוא שולח.

הפקטה תראה כך: פרוטוקול UDP, מעליו פרוטוקול QUIC שיכיל בתוכו Client Hello של TLS 1.3. בתוך ה-Client Hello יהיה ה-Session Ticket. מעל QUIC, שכבת האפליקציה. לדוגמה בקשת GET של HTTP/3. וכל זאת בפקטה הראשונה שהלקוח שולח לשרת!

TLS13 over UDP

בסעיף זה נניח בצד לרגע את QUIC ונסקור בקצרה אפשרות אחרת להעברת מידע מוצפן מעל UDP. פרוטוקול DTLS, קיצור של Datagram TLS. פרוטוקול זה מוסיף ל-TLS את המינימום הנדרש כדי להעביר אותו מעל UDP.

הסיבה לקיומו של DTLS בעולם שבו קיים כבר QUIC היא פשטות ומהירות. ישנן אפליקציות שבשבילן חשוב יותר שהמידע יעבור מהר מאשר שיושלמו פקטות חסרות. לדוגמה, VoIP, קיצור של Voice over IP. נרצה שהשיחה תעבור מוצפן, אבל אם פקטה אבדה השיחה כבר התקדמה ומיותר לנסות להשלים את ההברה החסרה.

לפרוטוקול זה אין פורט קבוע, במקום זה הפורט שלו תואם - או לעיתים מזכיר - את הפורט המתאים לשכבת האפליקציה שעוברת מעליו. לדוגמה קישורי VPN, שמעבירים בדרך כלל HTTPS, עובדים בפורט 443. עבור אפליקציית DNS הפורט הוא 853 (פורט 53 מעל UDP תפוס כבר מן הסתם...).

לפרוטוקול DTLS יש Retransmission רק על פקטות ששייכות ל-TLS Handshake. הצד השולח משתמש ב-Timeout ואם לא מתקבל תשובה, הפקטה נשלחת מחדש. לעומת זאת, פקטות שנשלחות אחרי ה-Handshake כבר לא כוללות מנגנון שידור מחדש. לכל פקטה יש מספר סידורי, גם כדי שהצד המקבל יידע להרכיב מחדש את הפקטות לפי הסדר וגם כיוון שהמספר הסידורי הוא בעל חשיבות בהצפנה. נזכור ש-TLS משתמש ב-Counter Mode כדי למנוע Reply Attacks.

נקודה מעניינת ב-DTLS היא שימוש ב-Extension שלא סקרנו עד עכשיו - Cookie. לאחר שהלקוח פונה לשרת, השרת מחזיר לו Cookie בתוך Extension, ולא ממשיך בתהליך ה-Handshake עד שהלקוח מחזיר לו את ה-Cookie.

כדי להבין מדוע נדרש ההלוך ושוב עם ה-Cookie, שנראה מיותר במבט ראשון, ניזכר בכלי שהכרנו היטב בחלק הראשון של ספר רשתות - Scapy. כפי שראינו, אפשר לייצר פקטה שכתובת ה*שולח* שלה היא מה שנבחר, כולל כתובות לא קיימות. השרת שמקבל את כתובת ה-IP הזו מאת השולח כביכול, ישלח את התגובה לכתובת זו.

אם אנחנו עובדים עם TCP, התהליך לא יתקדם מעבר לפקטת ה-SYN ACK. השרת יענה לכתובת IP מזוייפת כלשהי, ולא יקבל בחזרה פקטת ACK (או יקבל בחזרה פקטת RST אם יש בכתובת זו מחשב, שלא ביקש לפתוח סוקט מול השרת). כלומר אם מישהו זייף כתובת IP של שולח, הוא גרם לשרת לשלוח פקטת SYN ACK יחידה.

לעומת זאת מעל UDP, אם מישהו זייף כתובת IP של שולח עלולה להיגרם לשרת טרחה לא קטנה. השרת צריך לשלוח Server Hello, סרטיפיקט, Server Key Exchange. כל אלו דורשים חישובים וחתימות. כלומר במקרה של זיוף IP היחס בין כמות הטרחה בצד המזייף לבין כמות הטרחה של בצד השרת הוא גבוה מאד. השיטה הזו קורצת למי שרוצים לבצע מתקפת מניעת שירות על השרת.

ה-Cookie מונע זאת, כיוון שהשרת לא יבצע שום דבר עד שלא וידא שהפקטה מגיעה מכתובת IP שעומד מאחריה לקוח אמיתי.

DNS over QUIC

בפרק הקודם סקרנו איך עובר DNS מעל HTTP/2, הקרוי גם DoH. העקרון של DNS מעל QUIC, הקרוי גם DoQ, הוא דומה. נראה בקשה מצד הלקוח, כאשר סוג הבקשה ושם הדומיין מקודדים Base64. תגובת שרת ה-DNS תופיע בתוך QUIC, מוצפן כמובן.

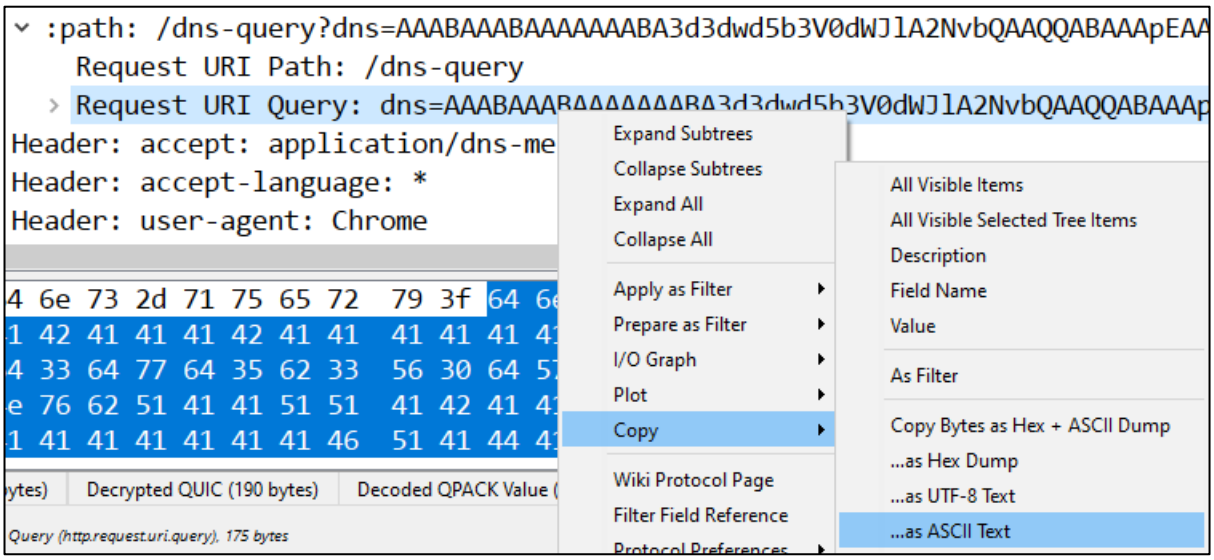
סיפורנו מתחיל בכך שהלקוח, הדפדפן, מקים קישור QUIC מול שרת ה-DNS המאובטח שהוגדר בדפדפן. לאחר הקמת הקישור, כל בקשה של הלקוח היא RTT-0, אין צורך בביצוע Handshake משום סוג. בהסנפה שאנחנו עובדים איתה, הלקוח מתקשר מול שרת ה-DNS של גוגל, התומך בבקשות DoQ.

נפתח את פקטה 3019.

```
Hypertext Transfer Protocol Version 3
  Request Stream
    HEADERS len=11
        [Stream ID: 12]
        Type: HEADERS (0x0000000000000001)
        Length: 11
        Frame Payload: 0c00d18ad781de88878680
        [Decoded Headers Length: 403]
        [Headers Count: 9]
      Header: :method: GET
      Header: :authority: dns.google
      Header: :scheme: https
      Header: :path: /dns-query?dns=AAABAAABAAAAAAAABA3d3dwd5b3V0dWJlA2NvbQAAQQABAAApEAAAAAAAA
      Header: accept: application/dns-message
```

הלקוח שולח בקשת GET של HTTP/3. המשאב המבוקש מקודד Base64.

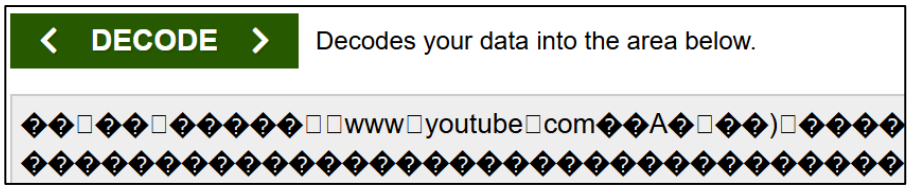
נעתיק אותו באמצעות קליק ימני - Copy - As ASCII Text.



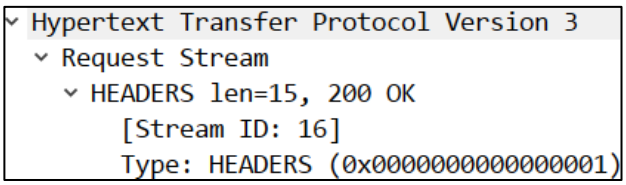
המחרוזת המקודדת היא:

```
AAABAAABAAAAAAAABA3d3dwd5b3V0dWJlA2NvbQAAQQABAAApEAAAAAAAAFQADABQAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

באמצעות שימוש במקודד Base64 נוכל לקרוא הן את הדומיין המבוקש והן את סוג הבקשה (במקרה זה, A, בקשת כתובת IPv4):



התשובה מגיעה בפקטה 3114 (כ-17 אלפיות שניה בסך הכל משליחת הבקשה). תגובת השרת מתחילה ב-200 OK:



לאחר מכן יש מידע מוצפן.

ניכנס אל ה-Decrypted QUIC ונראה שם הן חזרה על השאילתא (סוג A, www.youtube.com) והן את התגובה, עם כתובות ה-IP:

```
0030  89 88 80 ec 86 85 84 83  82 00 41 d4 00 00 81 80   ........ ..A.....
0040  00 01 00 07 00 00 00 01  03 77 77 77 07 79 6f 75   ........ .www.you
0050  74 75 62 65 03 63 6f 6d  00 00 01 00 01 c0 0c 00   tube.com ........
0060  05 00 01 00 00 00 aa 00  16 0a 79 6f 75 74 75 62   ........ ..youtub
0070  65 2d 75 69 01 6c 06 67  6f 6f 67 6c 65 c0 18 c0   e-ui.l.g oogle...
0080  2d 00 01 00 01 00 00 00  aa 00 04 8e fa 4b 6e c0   -------- .....Kn.
0090  2d 00 01 00 01 00 00 00  aa 00 04 8e fa 4b ae c0   -------- .....K..
00a0  2d 00 01 00 01 00 00 00  aa 00 04 8e fa 4b 4e c0   -------- .....KN.
00b0  2d 00 01 00 01 00 00 00  aa 00 04 8e fa 4b 8e c0   -------- .....K..
00c0  2d 00 01 00 01 00 00 00  aa 00 04 8e fa 4b ce c0   -------- .....K..
00d0  2d 00 01 00 01 00 00 00  aa 00 04 8e fa 4b 2e 00   -------- .....K..
00e0  00 29 02 00 00 00 00 00  01 26 00 0c 01 22 00 00   .)...... .&...".. 
00f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
Packet (588 bytes)  Decrypted QUIC (528 bytes)  Decoded QPACK Value (20 bytes)  Decompressed Header (516 bytes)
```

כתובת אחת מסומנת בריבוע, באמצעות מחשבון הקס' נמצא כי היא הכתובת 142.250.75.110. מתחת לכתובת זו ניתן להבחין בחמש כתובות דומות ששרת ה-DNS מספק על youtube.

בפקטה 3180 הלקוח מבצע את הפניה הראשונית לכתובת 142.250.75.110, זו היתה נקודת הפתיחה שלנו לפענוח תעבורת ה-QUIC. סגרנו את המעגל.

השוואה בין DoT, DoH, DoQ ו-DNS over DTLS

במהלך הלימוד סקרנו מספר שיטות לבצע שאילתות DNS, נוסף כמובן על ה-DNS הרגיל שעובר בפורט 53 מעל UDP. בשיטת DoH מעבירים DNS מעל HTTP מעל TCP. בשיטת DoT מעבירים DNS מעל TLS, בלי תיווך של HTTP. בשיטת DNS over DTLS, שנכנה אותה DoD, מעבירים DNS מעל TLS שעובר מעל UDP. ואת DoQ סקרנו כעת.

נסכם את השיטות בטבלה השוואתית:

	DoQ	DoD	DoH	DoT
שכבת תעבורה	UDP	UDP	TCP	TCP
פורט	443	853	443	853
האם גורם שיושב בין השרת והלקוח	אין אפשרות להפריד מגלישה, הבקשה	אפשר לזהות לפי הפורט הייחודי	אין אפשרות להפריד מגלישה, הבקשה	אפשר לזהות לפי הפורט הייחודי

יכול לחסום את השירות?	עוברת מוצפנת		עוברת מוצפנת	
שיהוי ותקורה	אפס שיהוי מול שרת שכבר הקמנו מולו קשר QUIC, תקורה בינונית	אפס שיהוי מול שרת שסיימנו איתו TLS Handshake, תקורה מינימלית	אפס שיהוי מול שרת שכבר הקמנו איתו סוקט מאובטח, פרוטוקול ה-HTTP גורם לתקורה גבוהה יחסית עקב השדות הרבים שב-Header	אפס שיהוי מול שרת שסיימנו איתו TLS Handshake, תקורה מינימלית
נפוצות	נתמך על ידי שרתי DNS הגדולים	פרוטוקול ניסיוני	נתמך על ידי שרתי DNS הגדולים	נתמך על ידי שרתי DNS הגדולים

סיכום

בתחילת הפרק הצגנו ארבע מוטיבציות עיקריות לפיתוח פרוטוקול חדש:

1. בעיית ה-Head Of Line Blocking- מעל TCP אי אפשר באמת ליצור חוסר תלות בין זרמי מידע שונים. זרם מידע אחד שמאבד פקטה, משהה את כל הזרמים שבאים אחריו.
2. בעיית ה-TCP Ossification. הפרוטוקול הגיע למצב שבו שינויים ושיפורים לא עוברים בהצלחה רכיבי רשת שאמורים להעביר אותם, כך שלמעשה כמעט לא מעשי להוסיף יכולות חדשות.
3. הורדת כמות ה-RTT. ראינו ש-TLS 1.3 לוקח בסך הכל RTT יחיד, אך נוסף עליו ה-RTT של ה- TCP Three Way Handshake. כך שסך הכל נדרשו RTT-2 לתחילת שליחת מידע של שכבת האפליקציה.
4. ביצוע Connection Migration - מעבר חלק בין תווך פיזי אחד לאחר (לדוגמה רשת סלולרית ל-WiFi) בלי צורך להקים מחדש את הקישור.

במהלך הפרק ראינו איך QUIC עונה על המוטיבציות הללו.

השימוש ב-UDP לבדו פתר את שלושת הבעיות הראשונות. על הדרך, השימוש ב-UDP הצריך מ-QUIC לקחת אחריות על יצירת ערוץ אמין ואיפשר ליישם שיפורים בדרך שבה עובד מנגנון ה-ACK. המנגנון החדש מאפשר אישור של מספר פקטות, תוך כדי דיווח על "חורים". שיפורים נעשו כדי לאפשר לצדדים להעריך את ה-RTT בצורה מדוייקת יותר לעומת מה שניתן לעשות עם TCP.

השימוש ב-Connection ID בתור אחד מהשדות של QUIC מייתר את הצורך בהקמת סוקט. כל הפקטות ששייכות לקישור מסויים מזוהות על ידי הצדדים לתקשורת. אפשר לעבור כתובת IP ועדיין הצד השני יידע שפקטה שהגיעה שייכת להמשך קישור קיים.

פרוטוקול QUIC מצפין לא רק את המידע אלא גם את ה-Header-ים. מי שעוקב אחרי תקשורת שאינה שלו, לא יצליח אפילו להרכיב יחד את רצף המידע, ששבור בין פריימים שונים.

בזאת סיימנו את הדיון ב-QUIC והשלמנו את ההתקדמות הטכנולוגית עד למועד כתיבת שורות אלה.

נסכם בהשוואה בין HTTP/1.1 לבין HTTP/2, HTTP/3:

	HTTP/3	HTTP/2	HTTP/1.1
שכבת תעבורה	UDP	TCP	TCP
פורט	443	443	443
אבטחה	TLS 1.3	TLS 1.2, TLS 1.3	לא חובה
דחיסת Header	QPACK	HPACK	אין
הורדת משאבים במקביל	אפשר, לכל משאב יש Stream ID משלו	אפשר, לכל משאב יש Stream ID משלו, אך עלול לקרות Head of Line Blocking בגלל השימוש ב-TCP	באמצעות פתיחת מספר סוקטים
RTT מתחילת התקשרות ועד שליחת בקשת GET	1, או 0 במקרה של Session Resumption	4, 3, או 2. מתוכם אחד עבור ה- TCP Three Way Handshake והיתר משתנים לפי גרסת TLS והאם יש Session Resumption	1 אם אין TLS, רק TCP Three Way Handshake. 3 או 4 אם יש גם TLS 1.2 Handshake, תלוי אם יש Session Resumption.
ניידות	יש. אפשר להחליף תווך פיזי ולעבור כתובת IP תוך כדי הקישור	אין. החלפת תווך פיזי, שגורמת לשינוי כתובת IP, דורשת הקמת סוקט מחדש	אין. החלפת תווך פיזי, שגורמת לשינוי כתובת IP, דורשת הקמת סוקט מחדש

מה צופן העתיד?

קשה לצפות איך ייראה חלקו השלישי של ספר רשתות מחשבים. אך הנה מספר תחזיות:

השינויים הטכנולוגיים הולכים ומאיצים ויחד איתם גם **קצב העדכון של הפרוטוקולים נהיה מואץ**. לפני עשור תעבורת HTTP לא מוצפנת היתה נפוצה. כיום גרסה 1.2 של TLS נחשבת מיושנת. אתרים שעדכנו לאחרונה גרסת TLS ל-1.3, כבר נמצאים בפיגור אחרי QUIC. סביר שהעדכונים הבאים יצטרכו להיות תכופים יותר.

כניסה של הצפנות פוסט קוואנטיות. מחשבים קוואנטיים יוכלו לפתור בזמן סביר את הבעיות המתמטיות שעומדות מאחרי RSA ו-DH. המעוניינים בקריאה נוספת מוזמנים לקרוא על אלגוריתם Shor. כבר כיום יש רעיונות והצעות של תקנים שיכולים לעמוד בפני מחשבים קוואנטיים. ייתכן שהחלק השלישי יצטרך להביא את התאוריה של ההצפנות הפוסט קוואנטיות ולהסביר כיצד תהליך ה-Handshake השתנה בהתאם.

שימוש גובר ב-QUIC יוביל למעבר לגרסאות מתקדמות יותר של הפרוטוקול. גרסת QUIC שאיתה בוצעה ההסנפה בפרק זה היא גרסה 1. גרסה 2 כבר קיימת, RFC 9369 מחודש מאי 2023.

הסתרת הדומיין אליו מתבצעת הגלישה. כיום ה-Client Hello מכיל שדה של SNI, Server Name Indication. גורמי מדינה שרוצים לצנזר תקשורת של אזרחים יכולים להשתמש בשדה זה כדי למנוע גישה לאתרים שונים. מסתמן שפרוטוקול ECH, Encrypted Client Hello, ייכנס לשימוש בקרוב. רשומות DNS יכילו שדה של מפתח ציבורי של השרת, כך שהלקוח יוכל להצפין חלק מה-Client Hello, ספציפית את שדה ה-SNI.

ותחזית אחרונה, שהיא אולי גם משאלת לב - **הבנה עמוקה של עולם המחשבים תמשיך להיות משמעותית**. נכון, תפקידים טכנולוגיים מסויימים יהיו מיותרים עקב בינה מלאכותית, אולם תפקידים שדורשים הבנה עמוקה לא רק שלא יוחלפו על ידי בינה מלאכותית אלא יהפכו משמעותיים יותר. כשם שהחלפת שפת אסמבלי בשפות עיליות לא הפכה את הידע באסמבלי למיותר אלא פתחה עולם חדש של חיפוש חולשות ופרצות אבטחה, כך כניסת הבינה המלאכותית תיצור אתגרים חדשים. מי שיידעו להבין דברים לעומק, לנתח ולחקור, להבין מתי הבינה המלאכותית נותנת תוצר שאינו אמין, הם יהיו הכוכבים והכוכבות של עולם המודיעין וההייטק המצפה לנו בטווח הזמן הנראה לעין.

כולי תקוה שספר זה קידם אתכם בדרך להבנה עמוקה.