**פרויקט סיום ברשתות תקשורת**

**מגישים: בן לכנר, שרון רוד, נבו ביטון ולידור בורופקר**

.

(הערה: החלק היבש נמצא בקובץ הזה, בסופו)

הפרויקט שאנחנו מגישים היום הוא תוצאה של כל מה שלמדנו בקורס רשתות תקשורת. בפרוייקט הזה למדנו לעבוד עם הפרוטוקול QUIC, פרוטוקול שתופס תאוצה בעולם התקשורת והוא בעל פוטנציאל לעקוף את TCP בכמות המשתמשים בעתיד הנראה לעין, משום שהוא פותר את כל הבעיות הקיימות בשני הפרוטוקולים הבסיסיים TCP וUDP, ומשתמש בדרכים מחוכמות על מנת לשפר את הפתרונות שהוא מציע לשני הפרוטוקולים הוותיקים.

אנחנו בחרנו להתמקד בנושא 2 של הפרוייקט – אמינות. מימשנו את אמינות הפרוטוקול באמצעות שיטת מספרי פקטות ושיטת הזמן, כפי שנסביר בהרחבה בהמשך.

כעת ניגש להסביר את תהליך העבודה על הפרויקט.

התחלנו את העבודה על הפרוייקט בקריאת המאמר על QUIC והבנתו. בהתחלה בן ושרון התקשו קצת עם המאמר שכן הוא כתוב באנגלית, אבל קצת עזרה מחברי הקבוצה, מהגוגל טרנסלייט ומהיוטיוב העניקו להם את ההבנה שהם היו צריכים.

מכאן פנינו היו אל החלק היבש. סיימנו את החלק הזה יחסית מהר והוא אילץ אותנו לחקור לעומק על הנושא של QUIC ואיך הוא עובד, ההבדלים בינו לבין הפרוטוקולים שהוא שואף להחליף, ועוד.

לאחר מכן הכנסנו את הרגליים למים בכך שיצרנו שרת ולקוח פשוטים עם פרוטוקול QUIC.

מכיוון שפרוטוקול QUIC בנוי בבסיסו על UDP, ניסינו קודם כל ליצור שרת UDP ולהתקדם משם. עוד לא התרכזנו בלהעביר פקטות, אלא רק התמקדנו בליצור חיבור בין השרת ללקוח. משום שQUIC נבנה על פרוטוקול UDP, הרשנו לעצמנו לקחת חלק מהמאפיינים של UDP ולהשתמש בהם לקוד שלנו.

בהתחלה שלחנו הודעות טקסט פשוטות של UDP, ומשם התקדמנו כדי להמיר את השליחות האלה להיות שליחות של פקטות QUIC.

פקטות QUIC הם פקטות מיוחדות ולכן היינו צריכים לבנות אותן בעצמנו. יצרנו שני סוגי פקטות – פקטה גדולה ופקטה קטנה (לפי המאפיין של QUIC שמשתמש בפקטות עם גודל HEADER משתנה). כל פקטה מכילה מספר זהות של הפקטה, סוגה, ומידע. יצרנו גם פונקציות הממירות את המידע בפקטה לטקסט, וממירות מטקסט חזרה לפקטה.

זה הזמן לנסות ולהעביר חבילות מידע QUIC ממשיות ביניהם. חילקנו את כל הפקטות לקבוצות של 5 (מצאנו כי זה החלוקה האידיאלית לפקטות). אם כל 5 הפקטות עוברות בהצלחה נמשיך ל5 הבאות, אך אם לפחות אחת מ-5 החבילות הלכו לאיבוד נשלח את אותן 5 החבילות שוב עד שכולן יגיעו בהצלחה ורק אז נמשיך ל5 הבאות. לפני שיצרנו מנגנון לאיבוד חבילות(פאקט לוס) יצרנו חבילות שהמידע בהן אקראי ורצינו להעביר אותן בזרימה אחת. כדי לבדוק אם אכן החבילות הגיעו באותו הסדר שהן יצאו יצרנו משתנה שנקרא sequence\_number. לכל חבילה נבדוק האם מספר החבילה שווה לsequence\_number. אם כן, נעבור לחבילה הבאה ונגדיל את sequence\_number ב1. אם לא סימן שחבילה מסוימת הגיעה לפני או אחרי שהיא אמורה להגיע וכאן ניכנס למנגנון איבוד החבילות.

כיצד מנגנון איבוד החבילות עובד? כאשר זיהינו שהחבילה שציפינו לה אינה הגיעה או שהגיעה אחרת במקומה, אנחנו נשכח מהחמישייה האחרונה ונשלח אותה שוב עד שכל החמש יגיעו כמו שצריך. אנחנו נוריד את sequence\_number למספר המתחלק ב5 הקרוב ביותר ונשלח 5 משם. זה אמור להחזיר לנו את אותה חמישייה. באג אחד רציני שנתקלנו בו בחלק הזה היה שכאשר זיהינו שחבילה אחת לא הגיעה והורדנו את הsequence\_number למספר המתחלק ב5 הקרוב ביותר, פתאום שאר החבילות מהחמישייה ההיא הגיעו והפריעו לסדר. כדי לתקן את זה הוספנו משתנה בוליאני שנקרא stop\_it שהמטרה שלו הייתה לעצור את החבילות האלו ואחרי קצת שינויים בקוד הצלחנו לגרום להכל לעבוד כמו שצריך.

לכאן הוספנו את אלמנט הזמן.

לאחר יצירת הקשר הגדרנו לשרת timeout של 0.5 שניות. אם בזמן 0.5 שניות האלה הוא לא מקבל חבילה מידע, הוא מניח שנאבדה לו פקטת ACK ולכן הלקוח לא יודע לשלוח את החבילה הבאה. הוא בודק האם הוא באמת צריך לשלוח פקטת ACK (זה מצוין לפי הדגל של send\_ack שנדלק לאחר ששלחנו ACK בפעם הראשונה ונכנס לפעולה רק כאשר שגיאת ה-timeout הופעלה).

במקביל, גם ללקוח הוגדר timeout. אם הוא אינו מקבל חבילת ACK מהשרת, הוא מניח שהיא נאבדה ולכן הולך 5 פקטות אחורה ושולח אותן שוב.

ככה המצב מתגלגל עד אשר כל הפקטות הגיע ליעדן בבטחה.

בהתחלה, לפני שקבענו timeout לקשרים, היו מצבים בהם השרת והלקוח היו נתקעים כי פשוט אחד מהם היה מחכה לחבילה של השני שהייתה נאבדה (במצב של איבוד פקטות מכוונת). מאז שקבענו timeout הבעיה הזאת נעלמה.

חשוב לציין כי שמנו לב כי לפעמים שליחת ACK או שליחת חלון הפקטות הבא קורים מהר מדי ו-"דןרכים" אחד על השני, ולכן במקומות מסויימים החלטנו לשים דיליי (sleep) של כמה מאיות שנייה על מנת למנוע דבר כזה.

בעצם עד עכשיו יצרנו template שידע להעביר חבילות מידע בשיטת זרימה יחידה ולהתמודד עם איבוד חבילות אך עד עכשיו תוכן החבילה היה לא רלוונטי. כל מה שנשאר היה לפרק קבצים לחבילות ולהלביש את החבילות הללו על הtemplate שיצרנו. בעצם יצרנו משתנה מחרוזת בשם file אצל הלקוח, שמקבל ערכים רנדומליים מהגודל הרצוי שלנו. הלקוח עובר על המחרוזת באמצעות באפר של 1000 (כלומר 1000 תווים) ומכניס אותם לתוך פקטת QUIC. הפקטה נשלחת, ואחריה נשלחות עוד 4 פקטות שמהוות את כלל החלון של הפקטות שנשלחות יחד.

במקביל, לאחר פתיחת הקשר השרת פותח פייל חדש. כעת, השרת מחכה עד שכל חמשת הפקטות יגיעו (ואם לא כולן מגיעות הוא מתמודד עם זה כמו שהסברנו לעיל). רק אחרי שהן מגיעות הוא רושם אותן לתוך פייל, ושולח ACK. הדבר הזה נמשך עד אשר הלקוח שולח בקשת terminate, ואז השרת סוגר ושומר את הקובץ.

כך בעצם עובדת השיטה שלנו לשלוח קובץ שלם על גבי חיבור QUIC.

ניסויים

כבר כשהתחלנו לגשש עם הקוד חיפשנו את הפרמטרים המתאימים. ניסינו גדלים שונים של חלונות (חבורת פקטות שנשלחת ביחד), באפרים (כמות המידע שנכנס לפקטנה אחת) וגודל קובץ. בסוף הגענו לכך שהפרמטרים הכי נוחים לנו הם גודל קובץ 25mb, חלון של 5 פקטות ובאפר של 1000 תווים. כך עבדנו רוב הפרויקט, ולמעט שינויים קטנים הפרמטרים נשארו איתנו עד עכשיו, לניסויים

תוצאות הניסויים:

ניתן לראות כי הזמנים די דומים וחגים סביב ה-5 שניות. ניתן לראות שכאשר שני השיטות מופעלות הקובץ נשלח הרבה יותר מהר, ובעיקר כאשר אין הרבה סיכוי לאיבוד פאקטות. ניתן להסיק כי כאשר שתי השיטות עובדות יחד הן "מחפות" אחת על השנייה וגורמות בממוצע לקצב שליחה מהיר יותר.

כאשר שני השיטות רצות בנפרד, ניתן לראות שהשינויים טיפה פחות דרמטיים. למרות זאת ניתן לראות כי סה"כ שיטת הזמן היא יותר מהירה מאשר מספרי הפקטות, בניגוד למה שאנחנו האמנו שיקרה. ההשערה שלנו היא שבמקום שהחלון ידרוש מאיתנו לחשב כמה פקטות אנו צריכים לחזור אחורה ולשלוח עליהם הודעה שלא הגיעו, הרבה יותר מהיר פשוט לחכות 0.1 שניות ולשלוח את הפקטות מחדש.

אנו מניחים כי ככל שאחוז איבוד הפקטות יעלה, המגמה הזאת תישמר. שכן במקרה של מספור פקטות יכולים להיאבד פקטות ו-ACK של פקטות, מה שיגרום לבלאגן גדול ולזמן ריצה הולך ותופח, זאת לעומת שיטת הזמן שפשוט לא מחכה לאישור ושולחת שוב את הפקטה אם היא חושדת שהיא לא הגיעה בזמן.

חלק יבש:

1. נפרט את חמשת החסרונות של TCP.

**חיסרון ראשון** טמון בחיבור בין אמינות העברת החבילות בTCP לבין הדרך שבה TCP מטפל בעומסים.

ב – TCP, על מנת לטפל בעומסים, מוגדר חלון בגודל מסויים שמטרתו היא לתרום לשליטת הזרימה ברשת, החלון עובר חבילה חבילה וכאשר חבילה נשלחת ומתקבלת כראוי, הוא ממשיך לחבילה הבאה. אך כאשר יש עומס, הוא מקטין או מגדיל אותו בדרכים שונות בהתאם לסיטואציה והאלגוריתם שהוגדר.

מצד שני, אמינות העברת חבילות ב – TCP דואגת לשלוח בשנית חבילות שנאבדו כדי לוודא את שלמות הdata ברשת.

כיוצא מזאת, כל סוג של איבוד חבילה, עלול לגרום להגבלה בתנועת חלון הזרימה קדימה, אפילו אם הסיבה לא נובעת מעומס בכלל, בנוסף לכך, החלון יפסיק להתקדם עד אשר האובדן ישוחזר, דבר זה יוצר עיכובים בצד השרת. לכן השילוב בין שני ההיבטים האלה גורם לעיכוב ברשת.

מעבר לכך שכל הסיבות האלה גורמות להגדלת הdelay ברשת, עיכובים, ואי ניצול מירבי של רוחב הפס הנתון, כמות החבילות שלא קיבלו ack, לא משקפת את כמות החבילות ברשת.

**חיסרון שני** קשור בעיקרו ליצירות של צוואר בקבוק בהעברת המידע הנובע מתוך מנגנון ההעברה האמינה של TCP. כאשר נאבדת חבילה, השרת שולח אותה שוב ללקוח, אך מכיוון שב-TCP החבילות ממוספרות, הצד המקבל לא יהיה מוכן לקבל את החבילות לא בסדר הנכון, כלומר אם ברצף מסויים נאבדה חבילה הלקוח לא ימשיך לקבל חבילות עד אשר תגיע אליו החבילה הבאה בסדר הנכון. כלומר חבילה אחת שנאבדת יוצרת צוואר בקבוק ועוצרת את זרימת המידע ללקוח של כל החבילות שמגיעות אחריה, דבר זה יוצר עיכובים ברשת.

**חיסרון שלישי** הוא לחיצת היד של TCP שנדרשת בכל יצירת חיבור בין שני צדדים, לחיצת היד משולשת היא הכרחית ולא יכול להישלח מידע לפני שהיא קוראת, כלומר גם אם הלקוח והשרת תיקשרו כבר פעם אחת, פנייה נוספת חדשה תדרוש שוב לחיצת יד משולשת, יתר על כן, אם אחד הצדדים ירצה גם לבטח את הקשר באמצעות TLS, תידרש עוד שליחה וקבלה של חבילות על מנת להחליף פרטי אבטחה בין הצדדים. כלומר, שלבים רבים חוזרים על עצמם ביצירת קשר בTCP כחלק ממנגנון האבטחה ואמינות הקשר שלו, דבר זה חשוב, אך בכל זאת יוצר trade off ב- delay ברשת.

**חיסרון רביעי** קשור בheader של TCP, ה-header מכיל את כל פונקציות השליטה שדרושות לפרוטוקול בתוך אותם 20 בתים, הבעיה בכך היא שיש לו גודל קבוע לפונקציות האלה ולכן הוא מושפע מאוד מעלייה במהירות הרשת, כלומר הגודל הקבוע של זיכרון עבור הפונקציות מגביל מאוד את הביצועים של TCP במהירות גבוהה, בTCP אכן יש עוד 40 בתים אופציונליים לשימוש אך גם כמות זאת מוגבלת ובמהירות מסוימת תגביל גם את הביצועים של TCP במהירויות גבוהות.

**חיסרון חמישי** קשור בדרך שבה TCP מבדיל בין קשרים.

TCP מתאים לכל צד בקשר זוג סדור של כתובת IP ו – PORT, הבעיה בכך היא שבמידה ו – IP פתאום משתנה מסיבה כלשהי (ולא חסר) כל המידע שהוחלף עד לשינוי ה – IP נעלם, מכיוון שאחד הצדדים כבר לא עונה על הidentifier ש – TCP הצמיד לו כדי לזהות אותו. לכן שני הצדדים יצטרכו להקים קשר מחדש ולחיצת יד נוספת תידרש מההתחלה.

2. חמשת התפקידים שפרוטוקול תעבורה צריך למלא הם:

1. להגדיר מזהי חיבור ומזהי מידע, מזהי חיבור הם לדוגמה כתובות IP, PORTS כמו שהפרוטוקולים שלמדנו עליהם עד כה משתמשים.

מזהי מידע הם דרך לסמן את סדר המידע הנכון, לדוגמה כמו שTCP ממספר חבילות בצורה סדרתית ובכך יודע את סדרן.

2. ניהול תעבורת החיבור

בעצם איזו שהיא רשימת חוקים שמתארת מה הפרוטוקול עושה, איך הוא מחבר בין שני צדדים, איך הוא מקים חיבור או סוגר אותו, איך תעבורת המידע תתנהל פרקטית בין שני צדדים לפי הפרוטוקול, דברים שהפרוטוקול תומך בהם בחיבור, למשל ריבויי כתובות IP.

3. העברת מידע אמינה

איך המידע מסומן ברשת ( כדי לדעת את סדר שליחתו או קבלתו), דרך העברת המידע.

4. טיפול בעומסים

בסופו של דבר הפרוטוקול צריך להגיע לידי שליטה בכמות החבילות שיש ברשת.

5. אבטחה

הכוונה באבטחה היא יצירת חיבורים מוצפנים, בדרך כלל הקמת החיבור מתרחשת ועליה מוסיפים אבטחה והצפנה של הקשרים.

3.

פרוטוקול QUIC, בשונה מ- TCP משתמש בדרך שונה ללחיצת יד, ראשית הוא לא משתמש בIP או PORT בתור מזהי חיבור, אלא בID פשוט.

הQUIC שולח את הID שלו למקבל והמקבל מחזיר לו ID, וכך בשתי שליחות מהירות הצלחנו כבר להקים חיבור.

לעומת זאת, בבעיה שתיארנו קודם בTCP, כל פעם שרוצים להקים קשר, צריך לחיצת ידיים משולשת של חבילת SYN, חבילת ACK-SYN וחבילת ACK. בנוסף לכך, QUIC תומך בלחיצת יד אחת, ולאחר מכן מאפשר חיבורים שכבר קרו ללא לחיצת יד, בשונה לTCP שכן מבקש לחיצת יד בכל הקמת קשר שהיא.

לכן אנו מבינים שQUIC אכן משפר חלק מהחסרונות של TCP בהקשר של לחיצת היד שלו, שכן היא יותר קצרה וקומפקטית, וכמובן דורשת פחות RTT מלחיצת היד של TCP.

4.

מבנה חבילת QUIC מחולק לשני סוגים של חבילות header, סוג אחד נועד עבור חבילות של הקמת חיבור, שהוא הפורמט הארוך יותר, סוג זה משומש רק בהקמת החיבור, לאחר מכן החבילות שעוקבות להם בחיבור משתמשות בסוג השני, בסוג הקצר יותר מטעמי יעילות.

פורמט הheader הקצר מכיל את הדגלים הבאים: ID של היעד, מספר פאקטה, מטען של פריימים של מידע שכל פריים מכיל Stream ID, אורך המידע, offset ואת המידע עצמו, סך משקל הפריים לא חוצה את גבול הMTU הנוכחי.

ניתן לראות שQUIC אכן משפר את TCP במקרה זה. בעוד שלTCP היה גודל קבוע בheader שלו, QUIC הרבה יותר גמיש ומאפשר שני גדלים עבור תפקידים שונים ואכן במהירויות גבוהות יותר ישיג כנראה ביצועים טובים יותר בשל ניצול טוב יותר של מקום לתפקיד מתאים.

בנוסף לכך, QUIC מאפשר עד ל256 בלוקים שלACK בתוך פריים יחיד, בעוד שTCP מאפשר 3 SACK עקב הגודל הקבוע והמגביל שלו. זה מאפשר לQUIC להחזיר ACK עבור חבילות שהתקבלו בקלות רבה. עובדה זאת מהווה שיפור גדול של QUIC על TCP במקרים של אובדן חבילות או בעיות בסדר קבלת המידע.

5.

בפרוטוקול QUIC, כל פאקטה מכילה מספר פאקטה. בעזרת מספר זה הפרוטוקול יודע לזהות איבוד חבילות או איחור שלהן. בנוסף לכך, כל פאקטה מכילה מספר frames של מידע, כלומר אם הפאקטה נאבדה, כך גם כל הframes שבה נחשבים כאבודים.

ברגע שעובר ברשת רצף פאקטות בפרוטוקול הQUIC, הפרוטוקול סופר את הפאקטות שהגיעו ומוודא שסדר הספירה באמת נשמר וכך כל הפאקטות מגיעות. ברגע שהרצף הסידורי דילג על מספר, למשל מספור הפאקטות תועד כך: 1,2,3,5.

אנו יודעים שחבילה מספר 4 מגיעה באיחור (או לא מגיעה) ישנם מספר thresholds שבעזרתם הפרוטוקול מחליט אם פאקטה מגיעה באיחור או נאבדה, אם רצף השליחה התקדם מעבר למספר החבילה שאיחרה מעל לאיזה שהוא x שקובע הפרוטוקול, החבילה תיחשב אבודה, אם החבילה תישלח בטווח סביר מתחת ל

ברגע שהפרוטוקול מזהה חבילה שנחשבת אבודה, הוא שולח את הframes שהכילה מחדש בעזרת חבילות חדשות שממשיכות כרגיל את רצף מספור החבילות, כך הפרוטוקול חוזר לסדר שליחה תקין לאחר איבוד או איחור הגעה של חבילה.

6.

פרוטוקול QUIC פועל בדומה לTCP כשזה מגיע לטיפול בעומסים ברשת, הדמיון בניהם מתבטא בכך ששניהם מטפלים בעומסים בעזרת חלון עומסים, שמטרתו להגביל את כמות החבילות האפשרית שתעבור ברשת בזמן נתון.

בניגוד לTCP, פרוטוקול QUIC מנסה להימנע מהקטנות מיותרות של החלון, שעלולות לפגוע בביצועים, בכך שהוא מקטין את חלון העומסים רק במקרה של עומס ממושך ברשת.

הכוונה בעומס ממושך הוא שמספר חבילות רצופות לא מגיעות ולכן לא מקבלות ACK, כאשר מגיעים למצב כזה, הקטנת חלון העומסים יותר סבירה שתטפל בעומס ברשת, ולכן הפרוטוקול פועל כך.

בנוסף לכך, הפרוטוקול נותן לשולח את האפשרות לממש אלגוריתמים לטיפול עומסים בעצמו ולשלב אותם בהעברת המידע.

קישור לעמוד הGithub של הפרויקט:

https://github.com/Lidx7/computerNetworking\_FinalProject