הסבר אופן פעולת RUDP

בחלק זה נתייחס להקלטה “RUDP segmentation and retransmission” ונסביר את אופן הפעולה הכללי של פרוטוקול RUDP, הקלטה זו נוצרה באמצעות תוכנות rudp\_sender\_test ו-rudp\_receiver\_test שבנינו על מנת לבדוק את הפרוטוקול, התוכנות מבצעות שליחה פשוטה של מחרוזת. לצורך ההדגמה הורדנו את MAX\_SEGMENT\_SIZE ל-10 (על מנת שתתבצע סגמנטציה) והגדרנו packet loss של 10% בעזרת tc על מנת להדגים retransmission.

ראשית נראה את כל התקשורת:

A screen shot of a computer

Description automatically generated

שימו לב שכל הודעה באורך 8 היא הודעה של header בלבד (SYN/FIN/ACK).

ראשית שתי ההודעות הראשונות הן לחיצת הידיים:

A screen shot of a computer

Description automatically generatedA black screen with white text

Description automatically generated

בהודעה הימנית (SYN) ניתן לראות שלאחר 4 ספרות ראשונות (אורך החבילה – 0 – בשני בתים) נמצאות שתי הספרות של ה-flag שהן 01 בהתאמה ל-flag SYN בלבד. ניתן לראות מהפורטים שההודעה נשלחה מה-sender ל-receiver (פורט ה-receiver הוגדר להיות 1025).

בהודעה השמאלית (SYN-ACK) ניתן לראות באותן ספרות של ה-flags 03 בהתאמה ל- SYN | ACK (00000001 | 0000010 = 00000011 = 0x03).

בנוסף ניתן לראות בכל אחת מההודעות checksum (לדוגמא 0xfeff) ו-segment\_num (0 בשניהם).

לאחר לחיצת הידיים ההודעה הבאה היא הסגמנט הראשון:

A black screen with white text

Description automatically generated

ניתן לראות שבשדה len קיים 0010 (0xa00 בהקסה ב-little-endian) מכיוון שזה האורך המקסימלי, שדה flags הוא 8 מכיוון שזה הערך של flag MOR שמסמן שזה אינו הסגמנט האחרון בהודעה, ה-checksum הוא איזשהו מספר שתלוי בכל ההודעה, ו-segment\_num הוא 0 (זה הסגמנט הראשון בהודעה). את שאר המידע נראה מצד ימין של החלון:



ניתן לראות את החלק הראשון של ההודעה ששלחנו “Hello thro”, נראה את שאר ההודעה בסגמנט הבא.

לאחר מכן קיימת הודעת ACK מה-receiver:

A screen shot of a computer

Description automatically generated

ניתן לראות שהיא דומה להודעת SYN-ACK רק שהשדה flags מכיל רק 2 (רק דגל ACK), שדה segment\_num מכיל 0 מכיוון שזה ACK לסגמנט הראשון.

לאחר שסיימנו עם ההודעה הראשונה מנגנון stop-and-wait עובר להודעה השנייה.

רואים שההודעה השנייה נשלחה 4 פעמים, ניתן להסיק שזאת מכיוון שבשלושת הפעמים הראשונות נאבד ה-ACK, נראה את אחת מהן:

A screen shot of a computer

Description automatically generated

ניתן לראות כאן דברים דומים להודעה הראשונה – אין דגלים והאורך הוא 10, אך בניגוד ניתן לראות ששני הבתים האחרונים של ה-header (0100) מכילים segment\_num 1 (ב-little endian) מכיוון שזה הסגמנט השני.

נראה את תוכן ההודעה בצד ימין של החלון:



ניתן לראות כאן את סוף ההודעה: “ugh RUDP!” ובסוף null-terminator (00 בבית האחרון).

כמו שאמרנו, ניתן לראות שאותה הודעה נשלחה 4 פעמים, עד שלבסוף התקבלה הודעת ACK:

A black screen with white text

Description automatically generated

ניתן לראות שבדומה ל-ACK הראשון ה-len הוא 0 ושדה flags הוא 2, אך כאן בדומה להודעה שזה ה-ACK שלה ב-segment\_num יש 1 (0x0100 ב-little-endian).

לבסוף קיימות שתי הודעות סגירה:

A black screen with white text

Description automatically generated A black screen with white text

Description automatically generated

ניתן לראות שהן דומות מאוד להודעות הפתיחה מלבד ששדה flags הוא 4 ב-FIN (צד ימין) ו-6 ב-FIN-ACK בהתאמה למה שהיינו מצפים מאיך שהשדות מוגדרים – שדה FIN הוא 4 ו-bitwise-or שלו עם שדה ACK שהוא 2 ייתן 6.

לאחר שראינו את התקשורת נגלה את פלט התכנות:

A screenshot of a computer

Description automatically generated

ניתן לראות שלאחר פתיחת הקשר ה-sender ביקש הודעה ונתנו לו את ההודעה “Hello through RUDP!”, הוא ניסה לשלוח אותה, כולל נסיונות חוזרים לאחר שהוא לא קיבל ACK 3 פעמים (כמו שראינו בהקלטה) ולבסוף ה-receiver קיבל את ההודעה והציג אותה בשלמותה (לאחר שפרוטוקול RUDP חיבר בחזרה את הסגמנטים).

לאחר מכן נתנו ל-sender את הפקודה “close” שגרמה לו לסגור את את התקשורת ושתי התוכנות דיווחו על סגירה.

**הסבר הקלטות TCP:**

ראשית נראה את הפעילות התקינה של התוכנה:

נפעיל את ה-TCP\_Receiver להאזנה בפורט 4000 בשימוש באלגוריתם reno.

הוספנו מוד format שבו יש פחות הדפסות. בהרצה הנוכחית המוד הזה כבוי.

לאחר מכן, נפעיל את ה-TCP\_Sender על אותו פורט עם אלגוריתם TCP Congestion זהה (נשים לב שבקוד של ה-TCP\_Sender הגדרנו IP דיפולטיבי במידה והמשתמש אינו מספק IP, לכן הצלחנו לתקשר עם ה-Receiver למרות שלא סופק IP כארגומנט).

נשלח את אותה חבילה בגודל 2MB חמישה פעמים ולבסוף נסגור את התקשורת ע"י סיום שליחת הקובץ.

בטרמינל של ה-TCP\_Receiver ניתן לראות ב-live את קבלת החבילות, את הזמן שלקח לכל חבילה להגיע, את המהירות ואת מס' החבילה (כאשר כל חבילה היא קובץ שלם, לא סגמנט מתוך קובץ).

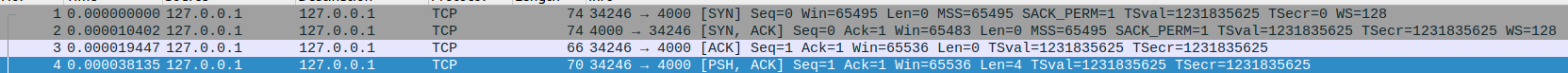
כאשר ה-Sender מסיים את התקשורת, נדפיס סיכום של הריצה וממוצעי מהירות, זמן ומס' ריצות כולל.

A screenshot of a computer program

Description automatically generatedA screenshot of a computer program

Description automatically generated

לאחר תחילת ההתקשרות בין ה-Sender ל-Receiver, ה-Sender שולח חבילה ראשונה שכוללת הצהרה על גודל הקובץ אותו היא תעביר בהמשך. זאת לטובת הקצאת זיכרון מתאים ב-buffer של ה-Receiver.



A screenshot of a computer

Description automatically generatedנשים לב שה-Data שנשלח הוא 2MB ביצוג שלהם בבייטים ב-Hexadecimal.

לאחר מכן תתחיל השליחה של הקובץ במס' סגמנטים מה-Sender ל-Receiver. לא נעלה תמונה של זה כי כל מה שרואים זה רצף של הודעות PSH והודעות ACK.

המשתמש יבחר כמה פעמים לשלוח את הקובץ. ה-Receiver ימשיך להאזין עד אשר תתקבל הודעת סיום התקשורת.

בסיום הריצה של ה-Sender נשלחת חבילה שכוללת את המלל “Closing connection” כדי להצהיר ל-Receiver על סיום התקשורת ביניהם (כדי שה-Receiver לא ימתין לקבלת חבילות נוספות).

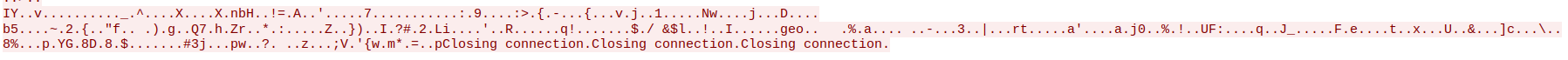
ניתן לראות ב-Wireshark את החבילה הזאת לפני סיום התקשורת (לפני ה-3 Way Handshake של TCP).

כמו כן, כל המלל שנמצא לפני ה-Closing connection הן שכבות של הפרוטוקולים שעטפו את החבילה (TCP, IP וכו').

A screenshot of a computer screen

Description automatically generated

ה-Sender ישלח את הודעת הסיום כמות מסוימת של פעמים (שמוגדרת כקבוע בקוד), ניתן לראות זאת בצורה ברורה ב-TCP Stream:



נזכיר שההודעה הזו נשלחת מס' פעמים כדי להתגבר על Packet Loss.

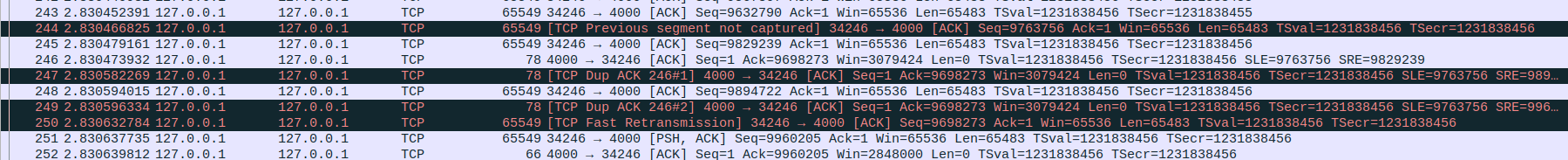
לאחר שהודעות הסגירה ישלחו, ה-Sender ישלח הודעת FIN (הודעת FIN של TCP) ושני ה-Sockets יבצעו לחיצת יד משולשת לפני סגירה:

A screenshot of a computer

Description automatically generated

בתמונה ניתן לראות את לחיצת היד המשולשת ואת הודעת ה-FIN הראשונה שבעצם מתחילה את סגירת התקשורת (כפי שניתן לראות ב-Expert Info).

כעת, נפעיל מצב Packet Loss בכדי לראות את ה-TCP Retranmission:

נריץ את ה-TCP\_Receiver ו-TCP\_Sender כמו מקודם וננסה לשלוח מס' חבילות.

נאפיין חבילה לפי 3 הספרות הראשונות ב-Seq number שלה.

נשים לב שבשורה 243 נשלחה החבילה ה-963. בשורה 244, Wireshark מתריע שלא התקבלה חבילה מכיוון שבשורה זו התקבלה חבילה 976 (Wireshark מזהה את הדילוג ולכן מתריע).

בשורה 246 ה-Receiver שולח ACK על כל החבילות עד 969, כלומר החבילה הבאה שה-Receiver מצפה לקבל היא 969 (שזו החבילה שקיבלנו התרעה שלא התקבלה מקודם).

בשורה 247 אנחנו שולחים ACK חוזר על כך שקיבלנו את כל החבילות על 969. זהו ACK על החבילה שקיבלנו בשורה 244, שאינה 969.

בשורה 248, ה-Sender ממשיך לשלוח לנו את החבילה הבאה (289), שכן היא נמצאת בחלון שלו.

בשורה 249, ה-Receiver שוב שולח ACK, בהתאמה לחבילה האחרונה. מס' ה-ACK תואם לחבילה 969 כדי להתריע ל-Sender על כך שה-Receiver עדיין מצפה לקבל את חבילה 969.

לבסוף, בשורה 250 מתבצע TCP Fast Retransmission בו ה-Sender שולח את חבילה 969.

בשורה 251, ה-Sender כבר מתקדם לחבילה הבאה ושולח את 996.

בשורה 252, ה-Receiver שולח ACK על חבילה 969. ניתן לראות שמס' ה-ACK הוא 996, שכן את כל החבילות בין 969 ל-996 ה-Receiver קיבל (חבילה 996 נשלחה מה-Sender במקביל ל-ACK שנשלח מה-Reciever).