פרויקט סייבר במסגרת תכנית גבהים

Reliable UDP

איתן גרוניך

318904703

תיכון ליאו באק

הגנת סייבר

מנחים

שרית לולב

אלון בר־לב

תוכן עניינים

מבוא

בפרויקט זה מימשתי פרוטוקול RUDP – Reliable UDP, וסרבר RUDP המתקשר באמצעותו.
Reliable UDP הוא פרוטוקול אמין מעל פרוטוקול UDP, שמטרתו להוות תחליף גמיש וקל להתאמה לפרוטוקול TCP, בתווכים לא אמינים. מטרת המערכת כולה היא לבצע, בעזרת מספר סרברים, proxying בפרוטוקול RUDP בין שני משתמשי קצה המתקשרים ב-TCP. כלומר, שני משתמשי קצה המעוניינים לבצע ביניהם proxying ב-RUDP, לא יתקשרו זה עם זה ישירות, אלא יתקשרו עם סרברי ה-RUDP, כל אחד עם סרבר ה-RUDP שלו. סרברי ה-RUDP יעבירו ביניהם את המידע המתקבל מהמשתמשים, תוך שמירה על אמינות.
הפרויקט כולל מימוש אסינכרוני של רכיבי התקשורת ראשית, הסרבר מתקשר עם משתמשי הקצה באמצעות פרוטוקול TCP, וזאת על מנת לשלוח להם מידע גולמי (data). הסרבר מתקשר עם אותם משתמשי הקצה גם בפרוטוקול נוסף, מעל פרוטוקול TCP, שגם אותו מימשתי, ונקרא Control Protocol. באמצעות פרוטוקול זה, מבקשים המשתמשים לאתחל חיבורים למשתמשים מרוחקים, וגם מבקשים מידע וסטטיסטיקה לגבי החיבורים שמנהל הסרבר.
הסרבר מנהל גם תקשורת בפרוטוקול RUDP מול סרברי RUDP אחרים. באמצעות פרוטוקול זה, מנהלים הסרברים זה עם זה חיבורים ושולחים מידע שהתקבל ממשתמשי הקצה.
בנוסף, הסרבר כולל בתוכו סרבר HTTP, המשרת אתר אינטרנט אינטראקטיבי ומעוצב, שבו ניתן לבקש פתיחת חיבורים ולחזות במידע וסטטיסטיקה על החיבורים הקיימים. תוכנו של אתר אינטרנט זה דומה לתוכן שניתן להפיק משימוש בפרוטוקול ה-Control, אך הוא ויזואלי ומיועד למשתמש אנושי.
על מנת לבדוק את המערכת, מימשתי גם תכניות בדיקה מקיפות.

פרק תיאורטי

הגדרת מושגים בסיסיים

מטרת פרויקט זה היא מימוש פרוטוקול אמין מעל פרוטוקול UDP. בכדי לעשות זאת נסקור את השיטות הקיימות להבטיח אמינות ברשת. אך לפני כן, יש להגדיר כמה מושגים בסיסיים שהבנתם קריטית להמשך:

מודל השכבות של OSI – מודל של ארגון התקינה הבינלאומי (ISO), שמתאר את הפעולות השונות הנדרשות כדי להעביר נתונים ברשת תקשורת, ואת הסדר ביניהן. המודל הוא בעל 7 שכבות: השכבה הפיזית, שכבת הקו, שכבת הרשת, שכבת התעבורה, שכבת השיחה, שכבת הייצוג ושכבת היישום. כל שכבה מבצעת חלק מסוים מהפעולות הדרושות לביצוע התקשורת. בכל רכיב ברשת המבצע תקשורת, כל שכבה מתקשרת עם השכבה המקבילה לה ברכיבי הרשת האחרים.

למידע נוסף: https://en.wikipedia.org/wiki/OSI_model

שכבת התעבורה (Transport Layer) – שכבת תקשורת המאפשרת תקשורת בין אפליקציות מסוימות, שרצות על רכיבי רשת שאינם בהכרח מחוברים בחיבור פיזי ישיר.

למידע נוסף: https://en.wikipedia.org/wiki/Transport_layer

פרוטוקול UDP – פרוטוקול תקשורת בשכבת התעבורה, ואחד הפרוטוקולים הנפוצים ביותר בשכבה זו. UDP הוא פרוטוקול חסר-חיבור (Connectionless), כלומר כל יחידת מידע (פאקטה) נשלחת בפני עצמה, ללא כינון מראש של ערוץ מידע בלעדי. כמו כן, UDP הוא פרוטוקול חסר-אמינות (unreliable), כלומר הוא אינו מבטיח הגעתן של פאקטות ליעדן, או את הגעתן בסדר שבו נשלחו. הודות למאפיינים אלה, פרוטוקול UDP הוא פרוטוקול פשוט המעביר מעט מידע פרט למידע הגולמי (data) – יש לו overhead נמוך. לכן פרוטוקול זה מעמיס פחות על הרשת ומבטיח זמן דיוור (זמן השהיה או latency) קצר יחסית לפרוטוקולים אמינים. משום כך, פרוטוקול UDP מתאים למערכות שמתעדפות מהירות על פני אמינות. דוגמה למערכת כזו היא הזרמת מדיה – "streaming" – שבה עדיף לאבד לעתים פריים (frame) כדי להבטיח מהירות דיוור גבוהה.

למידע נוסף: https://en.wikipedia.org/wiki/User_Datagram_Protocol

RFC: https://tools.ietf.org/html/rfc768

פרוטוקול TCP – פרוטוקול תקשורת בשכבת התעבורה, ואחד הפרוטוקולים הנפוצים ביותר בשכבה זו. TCP הוא פרוטוקול מבוסס חיבור (Connection-oriented). כלומר, לפני שליחת מידע בין שני צדדים, על שני הצדדים להקים ביניהם "ערוץ מידע" לוגי. בערוץ המידע (או ה"חיבור") יועבר מידע גולמי, אך גם מידע נוסף, המשפר את אמינות החיבור. בעזרת מידע נוסף זה, פרוטוקול TCP מבטיח הגעת פאקטות ליעדן והגעתן בסדר בו נשלחו. צדו השני של המטבע הוא ה-overhead הגדול שפרוטוקולים אמינים, ובפרט TCP, מעמיסים על הרשת. Overhead

זה גורם לזמן השהיה גבוה יחסית לפרוטוקולים לא אמינים, ולכן ב-TCP ישתמשו מערכות המתעדפות אמינות על פני מהירות, דוגמת גלישה ברשת.

בפרק זה נסקור את השיטות העיקריות של פרוטוקול TCP להבטיח אמינות כנגד איבוד פאקטות, נגדיר את פרוטוקול RUDP שמימשתי בפרויקט זה, ונשווה את בין מנגנוני האמינות של שני הפרוטוקולים. כמו כן, נשווה מבחינה מעשית את ההתמודדות של שתי המערכות עם איבוד פאקטות, ונסיק מכך מסקנות.

אז ראשית – כיצד פרוטוקול TCP מבטיח אמינות כנגד איבוד פאקטות?

למידע נוסף: https://en.wikipedia.org/wiki/Transmission_Control_Protocol

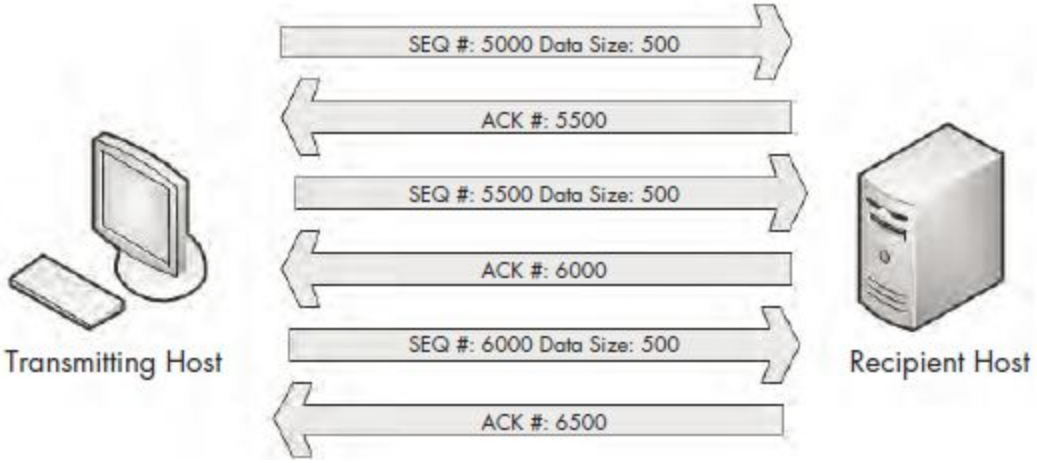
RFC: https://tools.ietf.org/html/rfc793

מספרים סידוריים וack-ים

כל פאקטה שנשלחת ב-TCP מאופיינת במספר סידורי – sequence number. המספר הסידורי ההתחלתי של כל צד בחיבור נקבע באופן אקראי, בסדרת האתחול הידועה כ- "Three Way Handshake":

1. יוזם החיבור (משתמש א') שולח פאקטה המכונה "פאקטת SYN". בפאקטה זו מציין השולח את המספר הסידורי ההתחלתי האקראי אותו הוא בחר – נקרא לו A.
2. מקבל החיבור (משתמש ב') שולח פאקטה המכונה "SYN-ACK". באמצעות פאקטה זו מבצע משתמש ב' שתי פעולות: ראשית, הוא שולח למשתמש א' אישור הגעה (acknowledgement או ack בקיצור) על הפאקטה ששלח משתמש א'. במסגרת אישור הודעה זו, מציין משתמש ב' מספר – A + 1 – שמייצג את המספר הסידורי של הפאקטה הבאה שמשתמש ב' מצפה לקבל ממשתמש א' (הסיבה תתואר בהמשך). שנית, משתמש ב' מציין את המספר הסידורי ההתחלתי האקראי שלו – נקרא לו B.
3. משתמש א' מחזיר למשתמש ב' פאקטת ACK בעלת ערך B+1 – שהוא המספר הסידורי של הפאקטה הבאה שמשתמש ב' מצפה לקבל ממשתמש א'.

מעתה, המספר הסידורי של כל פאקטה מציין למעשה את המספר הסידורי של בית המידע הראשון שלה, ביחס למספר הסידורי ההתחלתי. לדוגמה, אם A נבחר להיות 100, ואחרי האתחול שלח משתמש א' שתי פאקטות שכל אחת מהן באורך 100 בתים, המספר הסידורי של הראשונה תהיה 101, ושל השנייה תהיה 201. בכל פעם שמשתמש מקבל פאקטת מידע, הוא מחזיר לצד השני פאקטת ACK, שערכה הוא המספר הסידורי של הפאקטה הבאה שהמשתמש מצפה לקבל. בדוגמה שהבאתי לעיל: אחרי קבלת הפאקטה הראשונה ששלח משתמש א', משתמש ב' ישלח פאקטת ACK בעלת ערך 201. אכן, לפאקטה הבאה שמשתמש ב' קיבל מספר סידורי 201, והפעם יחזיר משתמש ב' פאקטת ACK בעלת ערך 301. מכניקת פעולה זו מוסברת היטב בתרשים הבא:



מהי התועלת שבציון ערך המספר הסידורי המצופה הבא בפאקטות ה-ACK?

ובכן, שיטה זו מבטיחה בקלות זיהוי שגיאות (Error Detection) הנובעות מאיבוד פאקטות, ולכן מאפשרת התאוששות משגיאות אלו (Error Recovery).

נשתמש בדוגמה שבתרשים לעיל. נניח שמשתמש א' (ה-Transmitting Host בתרשים) שלח שלוש פאקטות ברצף: פאקטה בעלת מספר סידורי 5000 ואורך 500, פאקטה בעלת מספר סידורי 5500 ואורך 500, ופאקטה בעלת מספר סידורי 6000 ואורך 500. משתמש ב' קיבל את פאקטות 5000 ו-6000, אך פאקטה 5500 נאבדה. כעת יחזיר משתמש ב' למשתמש א' פאקטת ACK, אך יציין בה את הערך 5500, כיוון שהוא מצפה עדיין לקבל את פאקטה 5500. בצורה זו יוודע משתמש א' לעובדה שפאקטה 5500 נאבדה.

במקום לשלוח מיד את פאקטה 5500 שוב, ימשיך משתמש א' לשלוח פאקטות בסדר הנכון. זאת משום שלעיתים הרשת עלולה לסדר מחדש את הפאקטות או לגרום לעיכוב בהגעתה של פאקטה, וייתכן שמשתמש ב' יקבל את פאקטה 5500 בעתיד הקרוב ללא צורך בשליחה מחדש. רק כאשר משתמש א' יקבל ממשתמש ב' שלוש פאקטות ACK ברצף שמציינות את אותו המספר (Duplicate Acks או Dupack) – במקרה זה 5500 – הוא ישלח מחדש (retransmit) את פאקטה 5500, ואת כל הפאקטות ששלח אחריה. סף ה-Dupacks המתקבלים לפני שידור מחדש, שנקרא ה-threshold (בתיאור לעיל – שלוש), קבוע במערכת.

אמנם שיטה זו מבטיחה התאוששות במקרה של איבוד פאקטה, אך התאוששות זו עלולה להיות מאוד לא יעילה. נסתכל לדוגמה על התרחיש הבא: משתמש א' שלח למשתמש ב' 10 פאקטות, כל אחת באורך 1000 בתים. משתמש ב' קיבל את כולן חוץ מהראשונה והשלישית. לאחר שמשתמש א' יקבל שלושה Dupacks, הוא ייאלץ לשלוח שוב את כל עשר הפאקטות, כאשר למעשה מה שחסר למשתמש ב' הוא רק שתי פאקטות מתוך העשר. כדי להתגבר על חוסר יעילות זו, חלק מחיבורי ה-TCP מיישמים שיטה המכונה "אישורים סלקטיביים" (Selective Acknowledgement או SACK). שיטת הSACK מאפשרת למשתמש לציין בפאקטות ACK מקטעים (לאו דווקא רציפים) של פאקטות שכן התקבלו באופן תקין. זאת בנוסף לציון המספר הסידורי המצופה הבא, כמו בפאקטת ACK רגילה. במקרה זה, משתמש ב' ישלח למשתמש א' פאקטה המציינת שני מקטעי SACK (SACK

1000 :(blocks עד 1999 ו-3000 עד 9999. בערך ה-ACK הרגיל יציין 0. בכך יידע משתמש א' לשלוח מחדש רק את פאקטה 1 ו-3, כלומר רק את המספרים הסידוריים 0-999 ו-2000-2999. השימוש ב-SACK אינו חובה בחיבורי TCP, וחייב להיות מוצהר על ידי שני הצדדים בעת אתחול החיבור, אך כיום השימוש בו נפוץ מאוד.

טיימאאוט

עד כה עסקנו בהתאוששות של פרוטוקול TCP מאיבוד פאקטות מידע. אך נשאלת השאלה – מה קורה כאשר נאבדת פאקטת ACK?

כדי להתמודד עם סוגיה זו מממש פרוטוקול TCP שיטת זיהוי ותיקון שגיאות המבוססת על זמנים. בכל שליחת פאקטה, מאתחל השולח טיימר, המודד פרק זמן מסוים שנקרא ה-RTO (retransmission timeout). אם פג הטיימר ולא הגיעה לשולח פאקטת ACK על הפאקטה שנשלחה, השולח ישלח שוב את הפאקטה, אך הפעם יכפיל את ה-RTO. כך ימשיך השולח לשלוח מחדש את הפאקטה ולהכפיל את ה-RTO, עד שיקבל פאקטת ACK על הפאקטה ששלח, או שיגיע למספר המקסימלי של ניסיונות שידור הקבוע במערכת ויסגור את החיבור. בווינדוס ברירת המחדל למספר מקסימלי זה היא 5 ניסיונות וברוב מערכות לינוקס ברירת המחדל היא 15 ניסיונות.

הכפלת ה-RTO נועדה, בין השאר, להגן מפני תקיפות רשת מסוג man-in-the-middle, שבו יעלים התוקף פאקטות ACK במטרה לגרום לשולח להציף את הנמען בשידורים חוזרים.

ערך ה-RTO נקבע לכל חיבור בנפרד. בעת אתחול החיבור, הערך נלקח מערך הקבוע במערכת. בשידורים הראשונים של כל משתמש בחיבור, מודד השולח את הזמן שחולף מרגע שליחת הפאקטה עד להגעת ACK על פאקטה זו (RTT או round-trip time). לאחר מדידת מספר ערכים כאלה, השולח בונה מהם ממוצע מתמטי ומשתמש בערך זה כערך ה-RTO הבסיסי לשידורים שלו בחיבור זה.

הגדרת פרוטוקול RUDP

פרוטוקול RUDP, הממומש בפרויקט זה, הוא פרוטוקול מבוסס חיבור (connection-oriented), המבטיח אמינות בהגעת פאקטות ליעדן בסדר הנכון. הפרוטוקול משמש לתקשורת בין שני שרתי RUDP, שכל אחד מהם מחובר בחיבור TCP למשתמש קצה. כל משתמש קצה מתקשר עם השרת שלו ב-TCP, והשרתים עושים proxy למידע זה בפרוטוקול RUDP. כלומר, זרימת המידע הבסיסית בתצורה זו היא:

משתמש א' -----TCP-----> שרת RUDP----- RUDP 1 -----> שרת TCP--- RUDP 2 -------> משתמש ב'.

על כך יורחב בהמשך.

ההנחה הבסיסית בתקשורת RUDP היא שהחיבור בין שני שרתי ה-RUDP הוא החיבור הבעייתי מבחינת אמינות, והחיבורים בין המשתמשים לשרתים אמינים.

לפאקטות בפרוטוקול RUDP חמישה מרכיבים:

- אורך – אורך הפאקטה (4 בתים)

- Connection ID/CID או מזהה חיבור – מספר המזהה את החיבור מבין אוסף החיבורים בין שני השרתים (4 בתים)
- Flag – מספר המזהה את סוג הפאקטה (בית 1). אפשרויות: פאקטת מידע, פאקטת ACK, פאקטת סגירה, פאקטת אתחול, ופאקטת keep-alive.
- Sequence Number או מספר סידורי (4 בתים) – מספר סידורי של הפאקטה. כל שרת בחיבור שומר מספר סידורי משל עצמו (בדומה ל-TCP).
- Data או מידע גולמי (1024 בתים לכל היותר) – המידע של הפאקטה

כל התיאורים להלן הינם, כמובן, ללא הגבלת הכלליות, ויכולים להתרחש באופן סימטרי לחלוטין בהיפוך תפקידי השרתים והמשתמשים המשתתפים בחיבור (במקום שרת א' – שרת ב', במקום משתמש א' – משתמש ב' וכו').

אתחול חיבור

1. אתחול החיבור מתחיל כאשר משתמש קצה א' מתחבר לשרת RUDP א', בפורט שהוקצה לכך קודם. בבקשתו את פתיחת הפורט קודם לכן, ביקש משתמש א' להתחבר למשתמש ב' דרך שרת ב'. לכן, כאשר משתמש א' מתחבר לפורט, שרת א' שולח פאקטת אתחול לשרת ב', ובה מציין את הכתובת של משתמש א' ואת הכתובת של משתמש ב'.
2. שרת ב' שולח מיד פאקטת ACK לשרת א'.
3. שרת ב' מנסה להתחבר למשתמש ב'.
4. במקרה של הצלחה בחיבור למשתמש ב', שולח שרת ב' לשרת א' פאקטת אתחול - אישור חיבור. במקרה של כישלון, שולח שרת ב' לשרת א' פאקטת סגירה והחיבור נסגר.
5. שרת א' שולח לשרת ב' פאקטת ACK על פאקטת האתחול (לא תקף למקרה של כישלון בחיבור).

סגירת חיבור

1. השרת המעוניין לסגור את החיבור ישלח פאקטת סגירה. לא נדרש ACK על פאקטת סגירה.
2. שרת המקבל פאקטת סגירה יסגור את החיבור שבו קיבל את הפאקטה.

שידור וקבלת מידע

1. כאשר שרת א' שולח לשרת ב' פאקטה (שאינה פאקטת סגירה או פאקטת ACK), הוא ימתין עד לקבלת ACK משרת ב' על פאקטה זו לפני שליחת פאקטות נוספות. אם לא קיבל ACK לאחר זמן מסוים (זמן זה נקרא טיימאאוט השידור מחדש), ישדר שוב את הפאקטה. שרת א' ימשיך לשדר מחדש את הפאקטה, עד שיקבל ACK או עד שהגיע למספר השידורים המקסימלי שלו. אם הגיע למספר השידורים המקסימלי, ולא קיבל ACK, יסגור שרת א' את החיבור.

2. כל פאקטה (שאינה פאקטת ACK) שנשלחת על ידי שרת א', ואינה שידור מחדש, היא בעלת מספר סידורי גבוה ב-1 מהפאקטה הקודמת.
3. כאשר שרת ב' מקבל משרת א' פאקטה (שאינה פאקטת סגירה או פאקטת ACK), ישלח מיד ACK לשרת א', עם מספר סידורי זהה לפאקטה שהתקבלה. לאחר מכן יבדוק על ידי בדיקת המספר הסידורי, האם פאקטה זו היא שכפול של פאקטה שכבר קיבל. לפי כך יידע האם להתייחס לפאקטה או לא.

Keep-Alive

1. כל זמן מסוים (נקרא טיימאאוט הkeep-alive), שבו החיבור קיים ושרת א' לא קיבל או שלח אף פאקטה (כולל פאקטה מכל סוג שהוא), ישלח שרת א' פאקטת keep-alive לשרת ב', ויצפה לקבל עליה ACK כפי שתואר בסעיף 1 של "שידור וקבלת מידע".
2. טיימאאוט ה-keep-alive שונה בכל שרת, וזאת על מנת למנוע שידורים בו-זמניים של פאקטות keep-alive.

השוואה בין מנגנוני האמינות של הפרוטוקולים

לפרוטוקול RUDP עקרונות דומים ל-TCP, אך הוא פשוט יותר. ההבדל העיקרי בין הפרוטוקולים, הוא שפרוטוקול השיחה של RUDP (שנקרא פרוטוקול מסוג "פינג-פונג") פחות מורכב מ-TCP, כיוון שהוא לא מאפשר שליחת פאקטות נוספות עד קבלת ACK על פאקטה שנשלחה. כלומר, בכל רגע נתון, שרת RUDP מחכה ל-ACK רק על פאקטה אחת בחיבור מסוים.

בנוסף, טיימאאוט השידור מחדש, המקביל ל-RTO של TCP, אינו משתנה, וזאת משום חוסר הצורך בפרויקט בהתמודדות עם התקפות man-in-the-middle.

השוואה מעשית

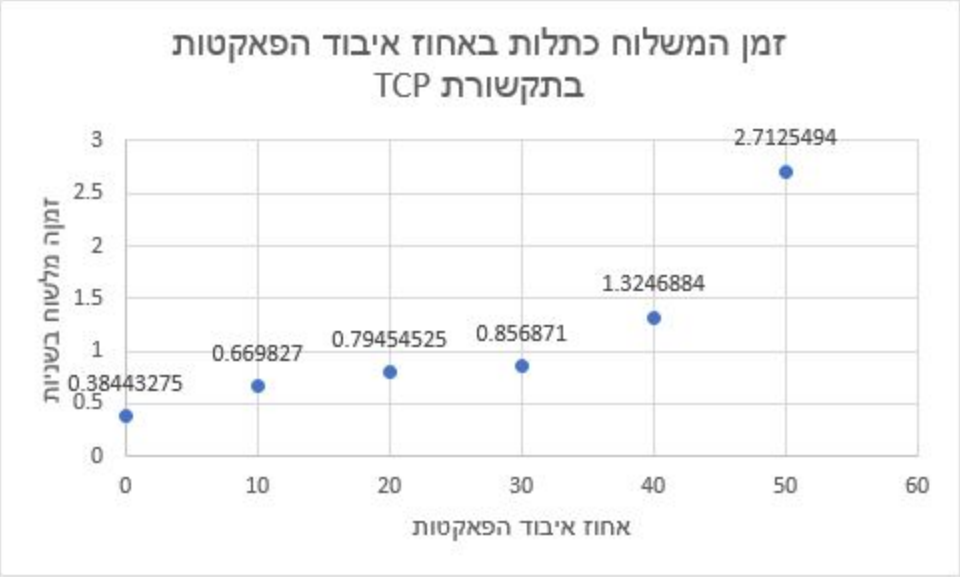
כעת נעבור להשוואה מעשית בין יכולת ההתאוששות של TCP על איבוד פאקטות, לבין יכולת ההתאוששות של פרוטוקול RUDP שמיממשתי על איבוד פאקטות.

כדי לבצע השוואה זו, נעזרתי בכלי שכתב חברי לכיתה, אופק צנזור, במסגרת הפרויקט שלו. הכלי שכתב אופק מסוגל להכניס "רעשים" מלאכותיים לחיבור TCP ובכך ליצור הדמיה של תווך לא אמין.

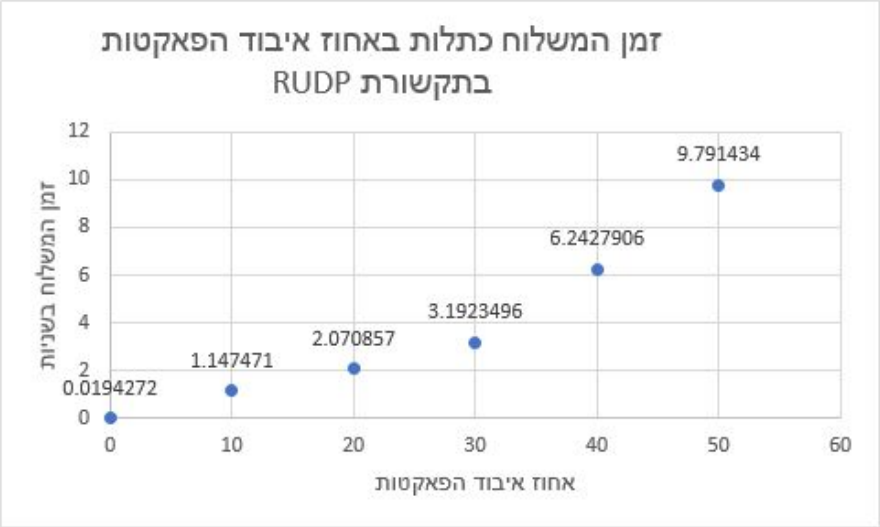
למטרת ההשוואה כתבתי שתי תכניות קצה פשוטות. הראשונה שולחת קובץ ליעד ומתעדת את זמן תחילת השליחה, והשנייה מקבלת קובץ ומתעדת את זמן סיום הקבלה. הערך המתקבל בניסוי הוא ההפרש בין זמנים אלו, כלומר – פרק הזמן שחולף מתחילת שליחת הקובץ בצד אחד ועד סיום קבלתו בצד השני. בניסוי עבדתי עם קובץ קבוע – קובץ טקסטואלי אקראי באורך 17 קילובייט.

ראשית מדדתי את הזמן שלוקח להעביר קובץ זה בין שני משתמשים המתקשרים ביניהם ב-TCP, ללא כל התערבות של המערכת שלי. הזמן נמדד בתווך נטול רעש, ולאחר מכן, בעזרת הכלי של אופק, בתווכים בעלי

אחוזי איבוד פאקטות שונים: 10% איבוד אקראי של פאקטות, 20%, 30%, 40% ו-50%. בכל תווך נעשו 5 בדיקות על מנת לשפר את סטטיסטיקת הנתונים, ונלקח מהם הממוצע. להלן התוצאות:



לאחר מכן ביצעתי בדיקה דומה, אך הפעם העברתי את הקובץ דרך פרוטוקול RUDP שלי, כלומר: התכנית השולחת התחברה לשרת RUDP אחד שלי ושלחה לו את המידע בסביבת TCP נטולת רעשים, שרת ה-RUDP שלח לשרת RUDP שני את המידע בסביבה עם רעשים, ושרת ה-RUDP השני שלח את המידע לתכנית המקבלת. מערכת ה-RUDP קונפגה לכל אורך הבדיקה ב-0.5 שניות מהתמנה עד לשידור חוזר, ו-15 ניסיונות שידור. להלן התוצאות:



מסקנות

פרט לתקשורת בעלת סביבה נטולת רעשים לחלוטין, פרוטוקול TCP רושם זמן דיוור קצר יותר בכל דרגות הרעש. את היתרון של RUDP בסביבה נטולת רעשים ניתן לייחס לאופן הבדיקה: בעוד שהבדיקה של התאוששות פרוטוקול TCP, בעזרת הכלי של אופק, התרחשה ב"רשת וירטואלית" בין "virtual machines", שמקים הפרויקט של אופק, הבדיקה של פרוטוקול RUDP נעשתה באמצעות שליחה בין כתובות ב-localhost על מחשב אחד בלבד. אמנם ההשפעה של הבדל זה אינה גדולה, אך היא מספיק משמעותית כדי ליצור פער בסביבה נטולת רעשים לחלוטין.

היתרון של TCP בשאר דרגות הרעש ככל הנראה נובע מיעילות גבוהה יותר של הפרוטוקול על פני RUDP. בעוד שב-TCP הטיימאאוט (RTO) מותאם לתווך, הטיימאאוט בפרוטוקול RUDP קבוע ויעיל פחות. בנוסף, פרוטוקול ה"פינג-פונג" של RUDP, שבו המשדר מחכה ל-ACK לפני שליחת פאקטה נוספת, איטי יחסית לפרוטוקול ה Sequence Num/Ack של TCP, שמאפשר שליחת פאקטות רבות לפני קבלת ACK-ים. יש להניח גם שתקשורת ה-TCP השתמשה ב-Selective Ack, מה שמגביר עוד יותר את יעילותה לעומת RUDP.

למרות זאת, כאמור במבוא, היתרון הגדול של פרוטוקול RUDP הוא גמישותו. בעוד ששינוי פרמטרים ב-TCP, כמו RTO התחלתי, מספר ניסיונות שידור חוזר ודupack threshold, דורשים גישה למערכת ההפעלה, את פרוטוקול RUDP ניתן להתאים בקלות וביעילות לכל סוג תווך. יש לשער, שבעזרת אלגוריתמים מתוחכמים יותר (זמני שידור מחדש גמישים ופרוטוקול שיחה יעיל יותר מ"פינג-פונג"), וקונפיגורציה ידנית לסוג התווך, פרוטוקול RUDP יוכל לעלות על TCP ביעילותו בתווכים מסוימים. הודות לפתיחות מערכת RUDP, ניתן אפילו לכתוב פרוטוקול שלם המותאם לסוג תווך מסוים, ובכך להפוך את RUDP מפרוטוקול אחד למשפחת פרוטוקולים הממומשים מעל UDP.

בנוסף, ניתן לממש בקלות יחסית מערכת שידור multicast אמינה מעל פרוטוקול RUDP, שתאפשר שידור לנמענים רבים. מימוש multicast אמין זה ייעשה באמצעות שליחת פאקטות לכל הנמענים, תוך שמירה בזיכרון של מספר מסוים של בתים שנשלחו (window), המשותף לכל הנמענים. במקום לשלוח פאקטת ACK על כל פאקטה, ישלחו הנמענים חזרה לשולח רק את המספרים הסידוריים של החלקים החסרים להם. מימוש פרוטוקול שכזה אינו אפשרי ב-TCP מפני שפרוטוקול TCP מהגדרתו מנהל חיבור נפרד עבור כל נמען, ולכן שומר עבור כל נמען window נפרד. כאשר מדובר במספר נמענים גדול ומידע רב, שיטה זו אינה ישימה, לעומת הפרוטוקול המוצע מעל RUDP. זהו יתרון נוסף של RUDP על פני TCP.

טכנולוגיות בפרויקט ומושגים נוספים

שפת python - שפת תכנות עילית ודינמית נפוצה. הרעיונות העיקריים מאחורי פייתון הם קריאות הקוד, וסינטקס המאפשר תכנות קצר, מתומצת וברור. פייתון מאפשרת תכנות מונחה עצמים, ויש לה ספרייה סטנדרטית ענפה. תוכניות פייתון רצות לאחר שהן עוברות אינטרפרטציה על ידי תוכנית שנקראת python interpreter.
קיימים python interpreters עבור מערכות הפעלה רבות, מה שמאפשר לפייתון לרוץ על מגוון רחב של מערכות.
למידע נוסף: (https://en.wikipedia.org/wiki/Python_(programming_language
הפרויקט שלי רץ בעיקר בשפת פייתון.

POSIX - אוסף תקנים שהוגדר על ידי ארגון ה-IEEE, ומיועדים לשמירה על תאימות בין מערכות הפעלה, במיוחד מערכות דמויות Unix. התקנים מגדירים למערכות ההפעלה בין השאר API וסביבות קומנד ליין (command line shells).
למידע נוסף: https://en.wikipedia.org/wiki/POSIX
פרויקט זה רץ באמולטור cygwin, שמדמה סביבת posix וירטואלית בתוך מערכת ההפעלה ווינדוס.

Async IO - צורת עיבוד I/O (קלט-פלט) שמתירה לחישובים נוספים להתרחש לפני שפעולת ה-I/O הסתיימה. צורת עבודה זו קריטית לכל תכנית העובדת עם מספר רב של file descriptors המבצעים I/O. לא ניתן להמתין לסיום פעולות ה-I/O, שהן בדרך כלל איטיות מאוד יחסית לעיבוד נתונים, ולהשאיר את משאבי המערכת במצב לא מנוצל.
למידע נוסף: https://en.wikipedia.org/wiki/Asynchronous_I/O
פרויקט זה משתמש בקריאות המערכת poll ו-select על מנת להבטיח Async IO.

פרוטוקול HTTP - Hypertext Transfer Protocol - פרוטוקול תקשורת בשכבת האפליקציה, הממומש מעל TCP. הפרוטוקול משמש להעביר דפי HTML ואובייקטים שהם מכילים ברשת האינטרנט. שרתי התוכן העיקריים של הפרוטוקול הם שרתי HTTP, ולקוחות התוכן העיקריים הם דפדפנים. התקשורת ב-HTTP בנויה מסדרה של בקשות מצד הלקוח ותשובות מצד השרת.
למידע נוסף: https://he.wikipedia.org/wiki/Hypertext_Transfer_Protocol
RFC: https://tools.ietf.org/html/rfc2616
פרויקט זה כולל שרת HTTP המשרת דפי תוכן.

HTML - Hypertext Markup Language - השפה הסטנדרטית לתיאור דפי אינטרנט ומרכיביהם. השפה מורכבת מזוגות של תגיות - תגית פתיחה ותגית סיום - המציינים מרכיבים בדף האינטרנט. קבצי HTML בדרך כלל נשלחים לדפדפנים משרתי HTTP, ומוצגים באמצעות הדפדפן באופן ויזואלי.
למידע נוסף: https://he.wikipedia.org/wiki/HTML
פרויקט זה כולל עמודי תוכן שנבנו בשפת HTML.

CSS - Cascading Style Sheets - שפת עיצוב המשמשת לעיצוב מרכיבי תוכן ב-HTML, XML, XHTML, וכל שפה אחרת שהיא markup language. ההפרדה בין התוכן, המוצג בשפת ה-markup, לבין העיצוב, המוגדר ב-CSS, מאפשר גמישות גדולה יותר בעיצוב דפי אינטרנט ומוריד את המורכבות של עיצוב זה.
למידע נוסף: https://en.wikipedia.org/wiki/Cascading_Style_Sheets
פרויקט זה כולל קבצי CSS המתארים עמודי HTML.

קריאת מערכת - System Call - בקשה שמבצעת תוכנת מחשב מליבת מערכת ההפעלה (ה-kernel), כיוון שאינה יכולה לבצע אותה בעצמה. קריאות מערכת האחראיות על החיבור בין המשתמש ל-kernel.
למידע נוסף: https://en.wikipedia.org/wiki/System_call
פרויקט זה משתמש במספר קריאות מערכת, כגון: poll, select, stat, fork

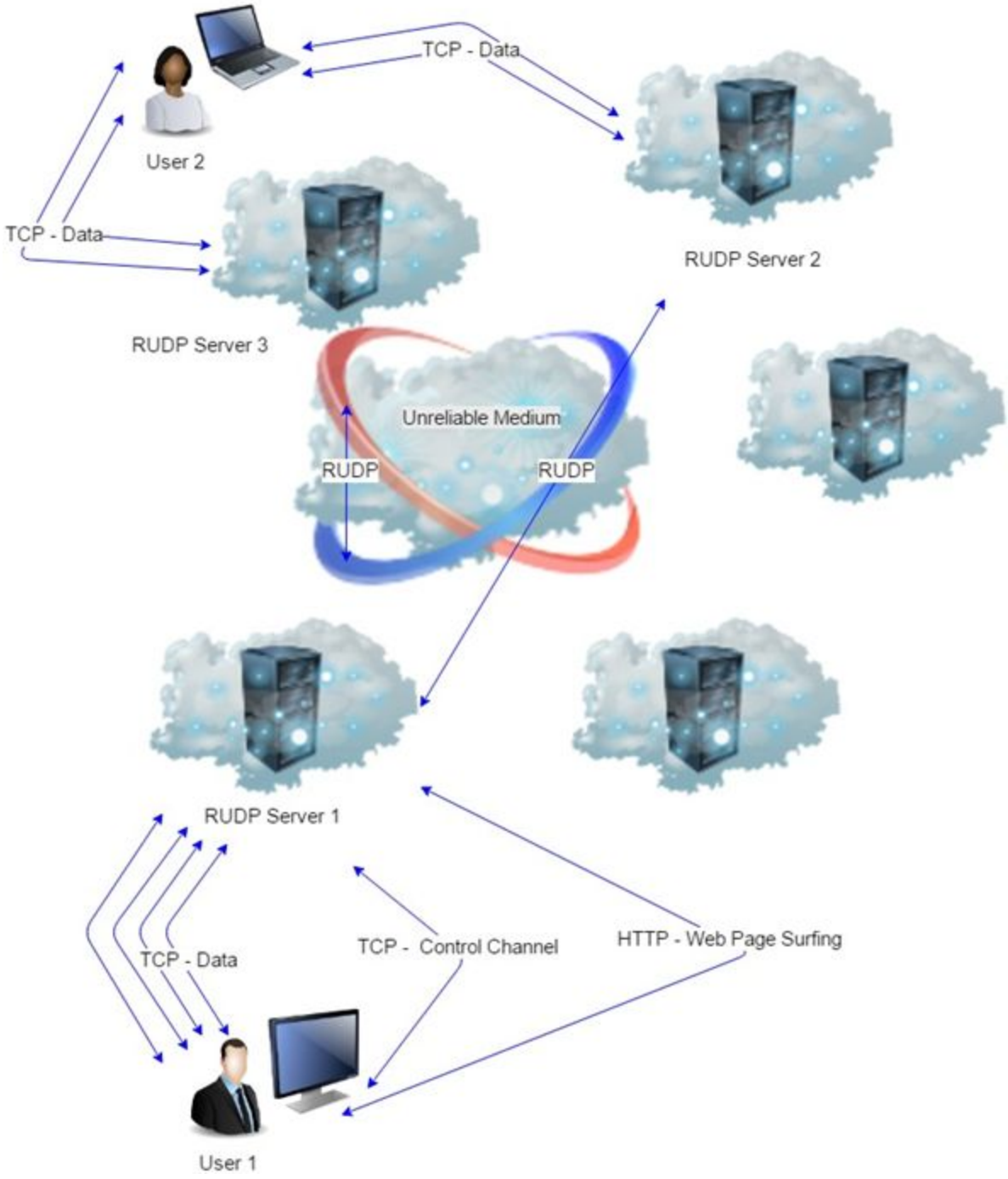
תכנית daemon - תכנית שרצה כתכנית רקע במערכת ההפעלה, ולא נתונה לשליטתו של משתמש מסוים.
למידע נוסף: (https://en.wikipedia.org/wiki/Daemon_(computing
פרויקט זה כולל אפשרות להריץ את שרת ה-RUDP כתכנית daemon.

אותות (signals) - אותות הם צורת תקשורת בין תהליכים במערכות הפעלה הפועלים לפי תקני POSIX.
האותות הם התראות אסינכרוניות שנשלחות לתהליך מסוים על מנת להודיע לו על אירוע שקרה.
למידע נוסף: https://en.wikipedia.org/wiki/Unix_signal
פרויקט זה משתמש באות SIGINT כדי להבטיח סגירה מסודרת של התכנית.

קובץ log - קובץ בו מתעדת תוכנית את האירועים המתרחשים בעת הרצתה.
למידע נוסף: https://en.wikipedia.org/wiki/Logfile
פרויקט זה כולל אפשרות לתיעוד בקובץ log, ומשתמש ברמות log שונות (debug, info, warning, error, critical) על מנת ליצור תיעוד מסודר של הרצת התכנית.

תכנות מונחה עצמים (Object Oriented Program) - תכנות המבוסס על רעיון ה-"עצמים", המכילים נתונים בצורת תכונות (attributes) וקוד בצורת פעולות (methods). תכנות מסוג זה הופך את המערכת למסודרת יותר, מובנת יותר ופשוטה יותר לשינוי.
למידע נוסף: https://en.wikipedia.org/wiki/Object-oriented_programming
פרויקט זה מגדיר עצמים רבים המייצגים את כל רכיבי המערכת.

ארכיטקטורת הפתרון



בדיאגרמה שלעיל ניתן להתרשם מארכיטקטורת הפתרון שלי:

1. משתמשי הקצה מתקשרים עם שרתי ה-RUDP בשלושה ערוצים שונים:

a. מידע גולמי המועבר בפרוטוקול TCP
b. פרוטוקול בקרה (control) למידע, סטטיסטיקות ובקשות חיבורים מעל פרוטוקול TCP
c. גלישה בפרוטוקול HTTP בעמוד האינטרנט הויזואלי, המספק מידע, סטטיסטיקות ובקשות חיבורים

2. שרתי ה-RUDP מתקשרים זה עם זה בפרוטוקול RUDP, מעל תווך לא אמין.

3. כל שרת RUDP מסוגל לנהל חיבורים מול יותר משרת RUDP אחד, ומול כל שרת הוא מסוגל לנהל יותר מחיבור אחד.

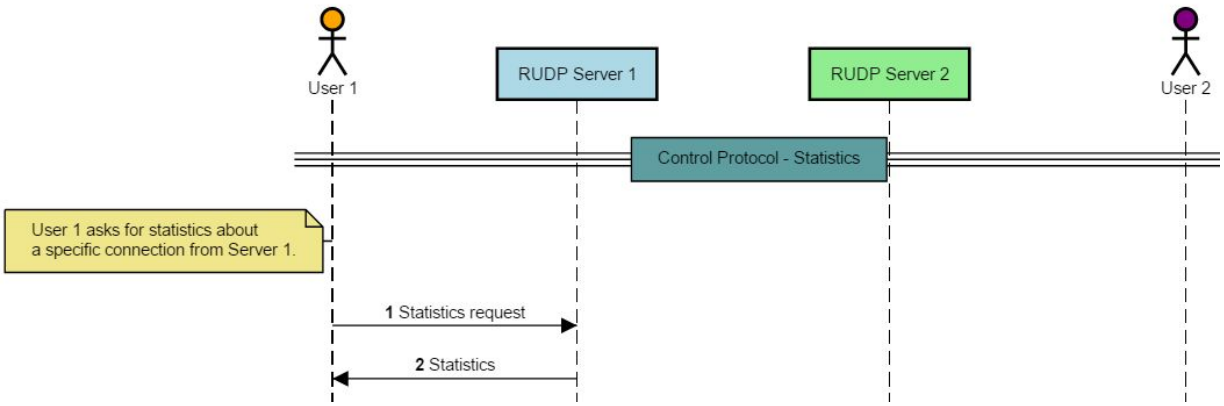
פרק הרצף

בפרק זה אתאר את הרצפים העיקריים המתרחשים בין רכיבי הארכיטקטורה שלי בפרויקט.

אתחיל מהתקשורת בערוץ הבקרה - Control Protocol:

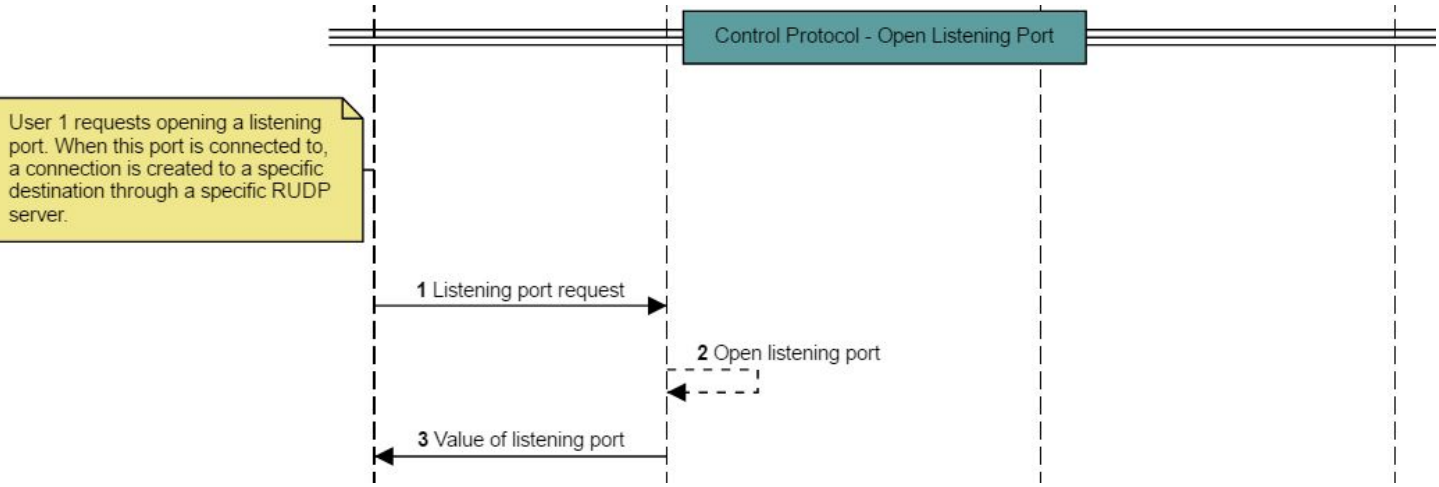
סטטיסטיקות

תהליך פשוט של בקשת סטטיסטיקות מהשרת



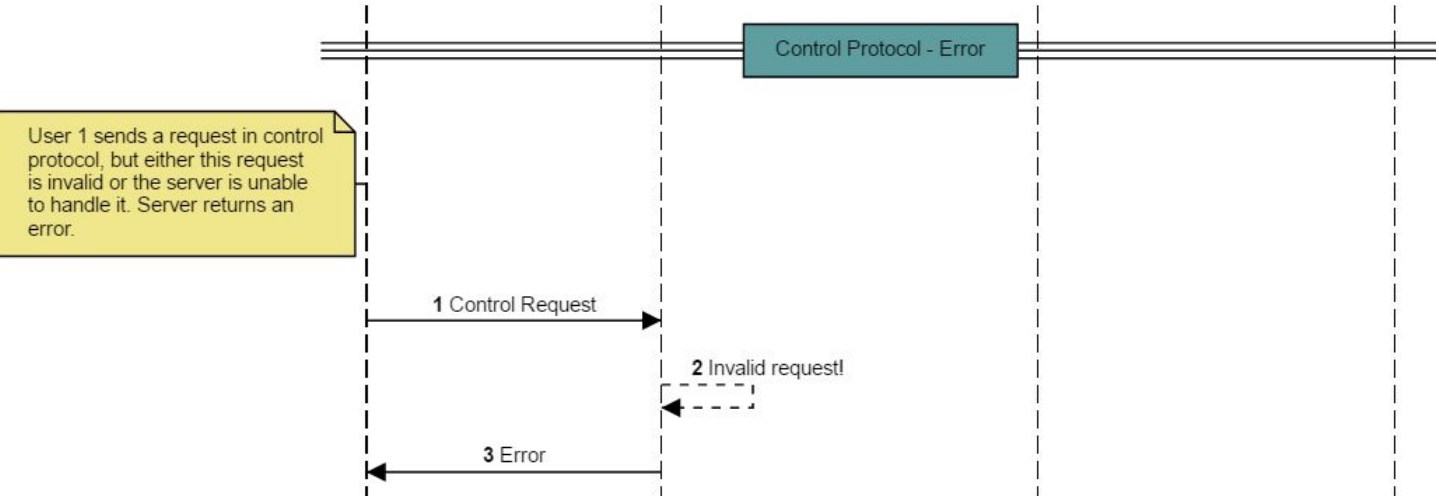
פתיחת פורט (בקשת חיבור)

תהליך בקשת החיבור. למעשה, המשתמש אינו מבקש "חיבור" אלא מבקש מהשרת לפתוח פורט, שכאשר יתחבר אליו המשתמש, יאותחל חיבור ליעד הרצוי (משתמש 2), דרך שרת המעבר הרצוי (שרת 2):



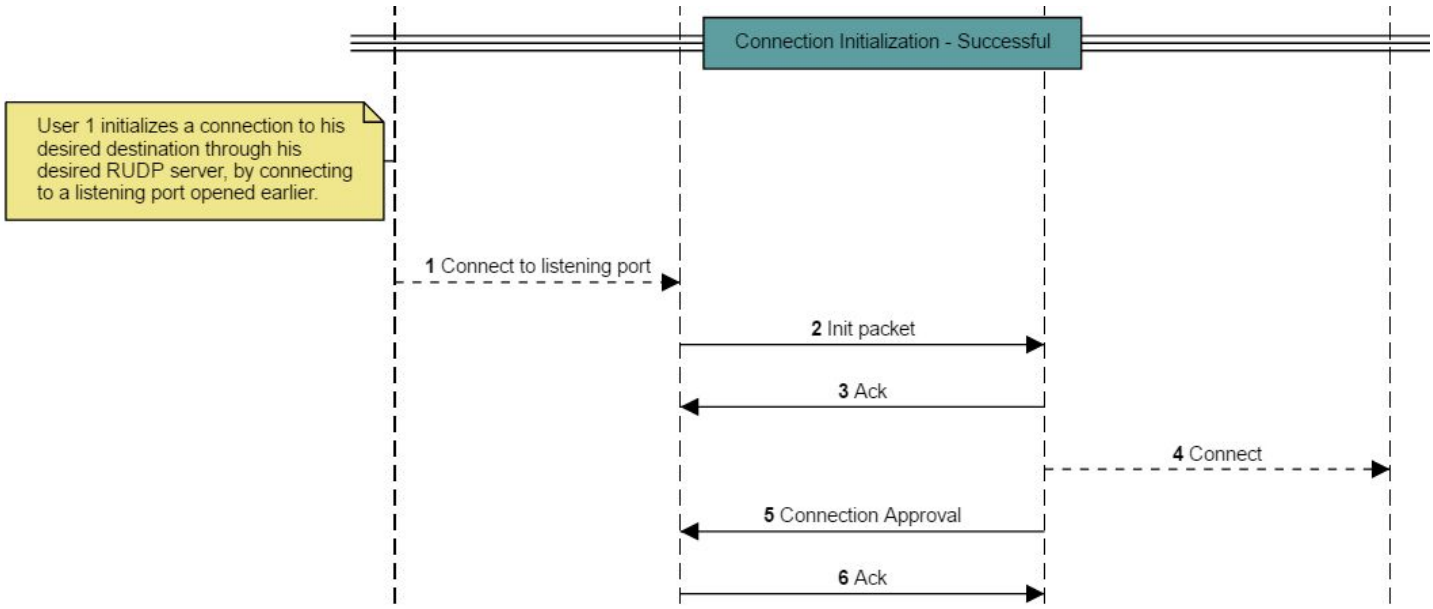
תקלה בערוץ הבקרה

כעת נראה מה קורה כאשר יש תקלה בבקשה בערוץ הבקרה. תקלות אלו נובעות מבקשה לא תקינה של המשתמש. חוסר התקינות מתבטא ב-syntax שגוי של הבקשה, או בקשה שאינה אפשרית (לדוגמה: בקשת מידע על חיבור שאינו קיים):



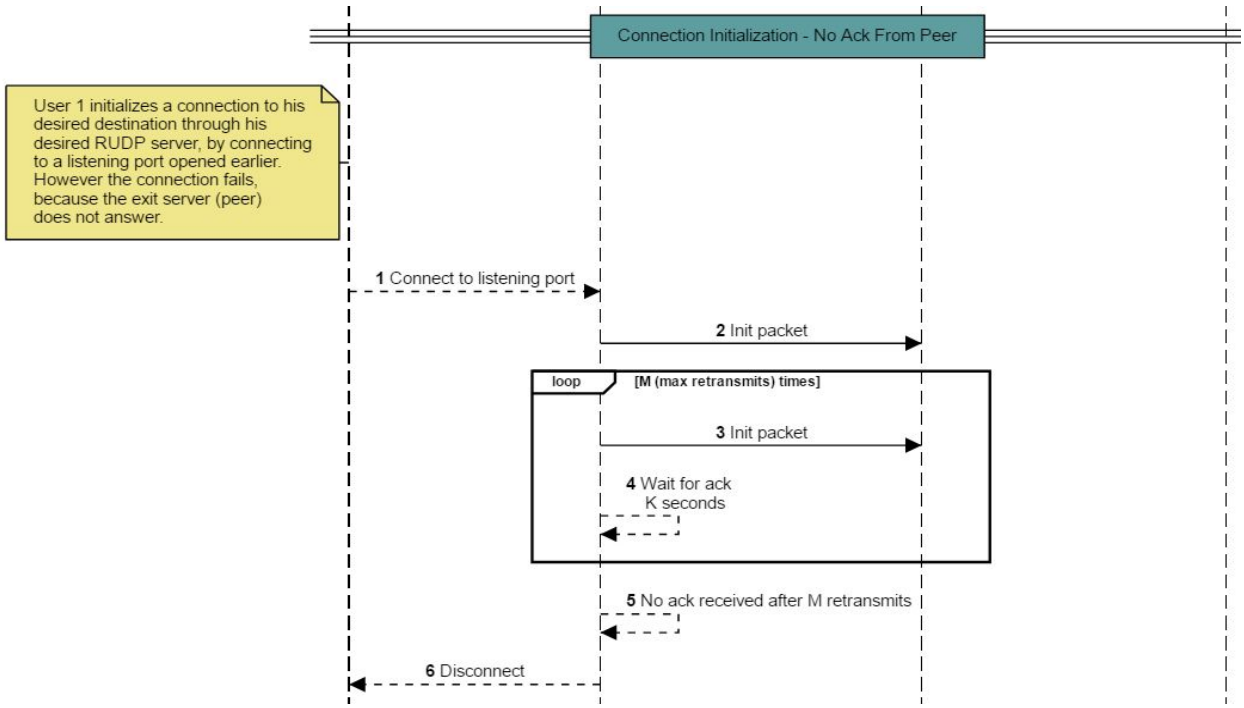
אתחול מוצלח של חיבור

רצף התקשורת בעת אתחול מוצלח של חיבור, לפי כל השלבים המתוארים בפרוטוקול. יש לשים לב, שמשתמש 1 מתחבר לפורט מסוים, שכבר נפתח בעבורו, ושמוגדרים לו יעד סופי ושרת מעבר מסוימים:



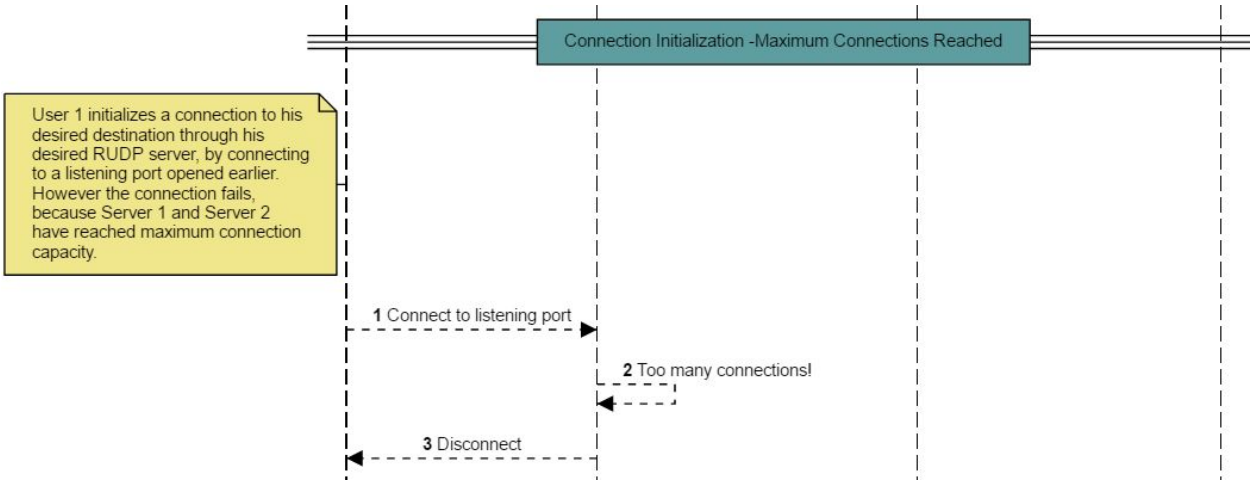
אתחול נכשל - אין מענה

שרת 1 אינו מקבל ACK משרת 2 על פאקט האתחול שלו. ייתכן ששרת 2 אינו קיים בכתובת שמנסה שרת 1 לשלוח אליה. ייתכן גם שהרשת עד כדי כך לא אמינה, שכל הפאקטות ששלח שרת 1 נאבדו, או שפאקטות ה-ACK של שרת 2 נאבדו. בכל מקרה, התוצאה היא סגירת החיבור והתנתקות ממשתמש 1. ברצף זה, M מייצג את מספר השידורים החוזרים המקסימלי המקונפג במערכת, ו-K מייצג את טיימאאוט השידור החוזר, או ה-RTO.



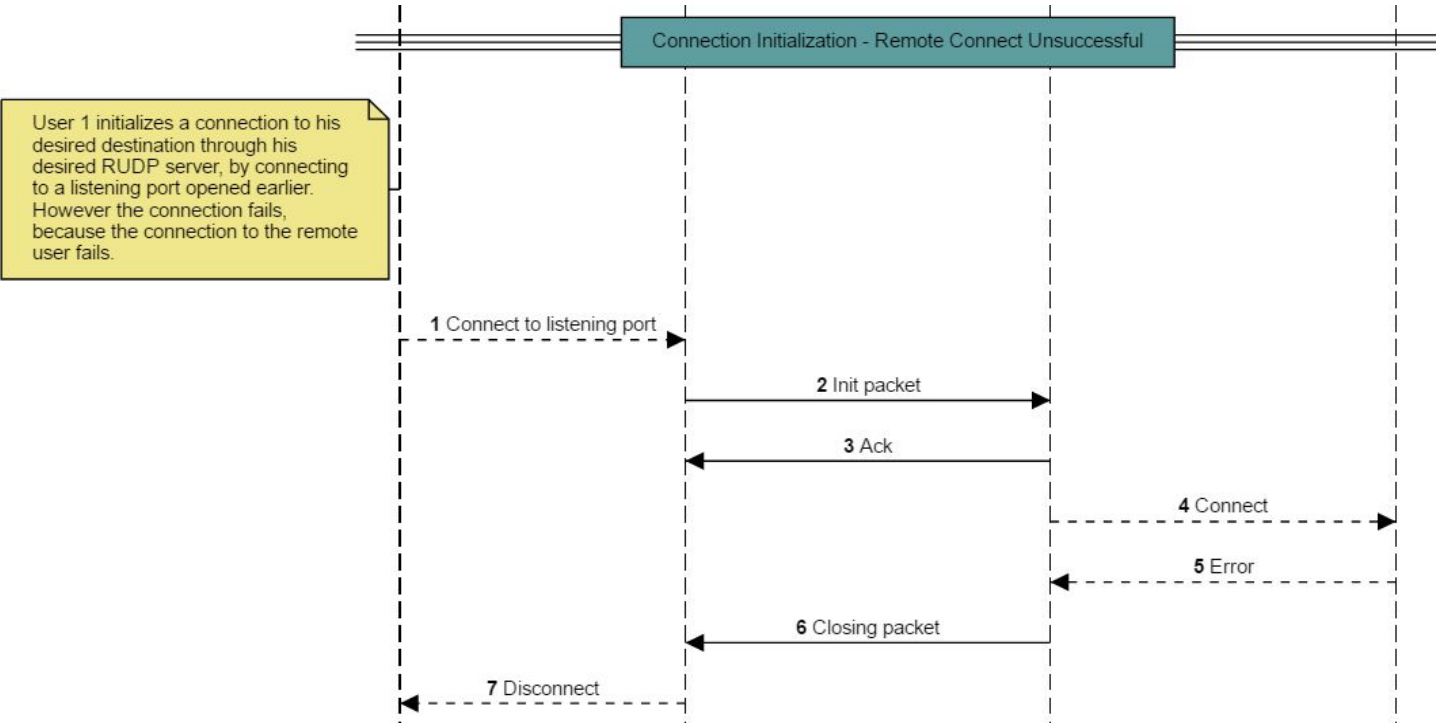
אתחול נכשל - מקסימום חיבורים

אתחול החיבור נכשל משום ששני השרתים הגיעו למספר המקסימלי של חיבורים ביניהם, ולא יכולים לקבל חיבור נוסף. מספר מקסימלי זה זהה למספר האפשרויות לCID-ים (מזהי חיבור) שונים, וברירת המחדל שלו היא 65,536 (2 בחזקת 16).



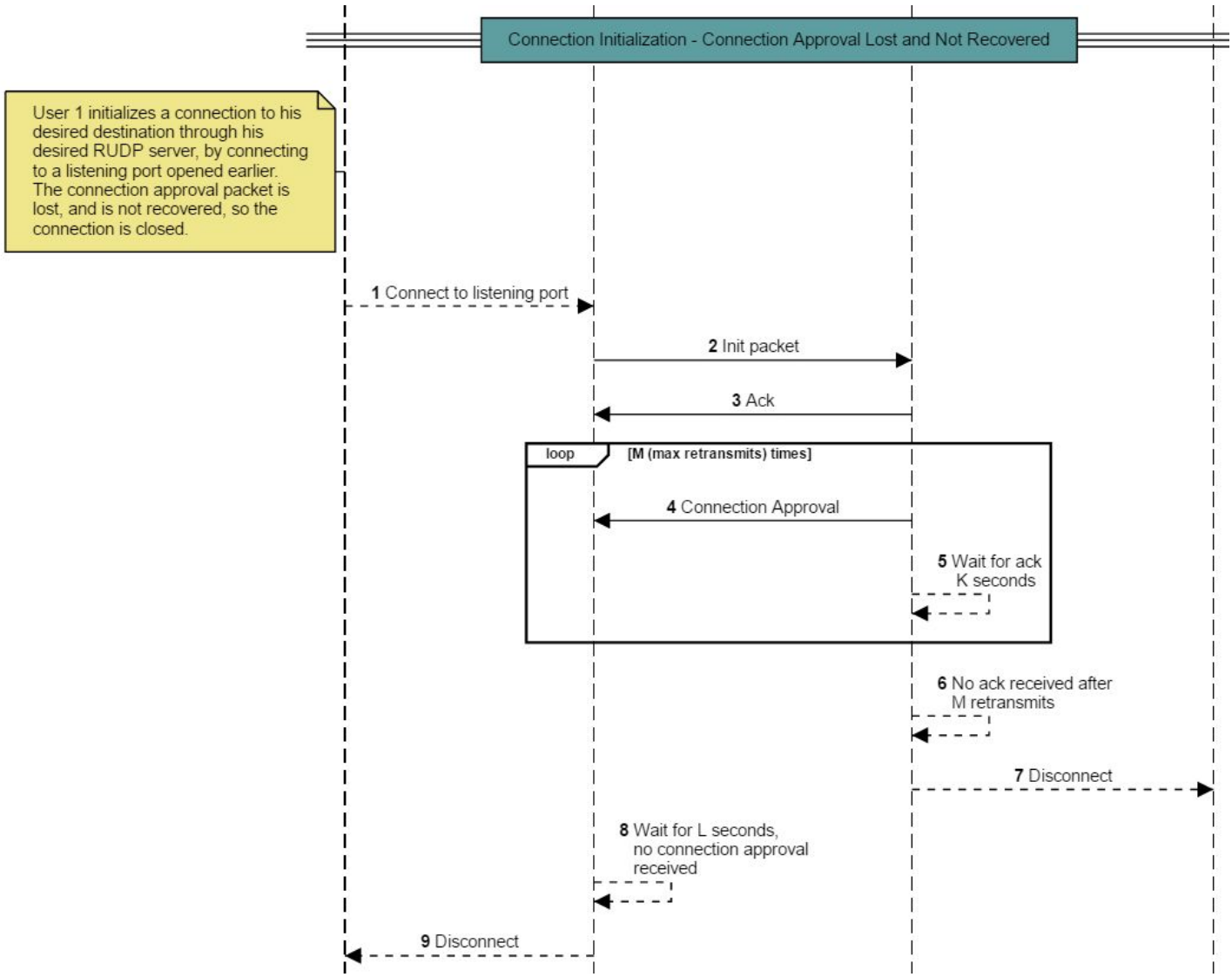
אתחול נכשל - חיבור כושל ליעד

אתחול החיבור נכשל משום שהחיבור ב-TCP למשתמש היעד, משתמש 2, נכשל. הסיבות לכישלון זה הן מגוונות וקשורות כולן לפרוטוקול TCP. כאשר החיבור נכשל, שרת 2 שולח לשרת 1 פאקטת סגירה והחיבור נסגר.



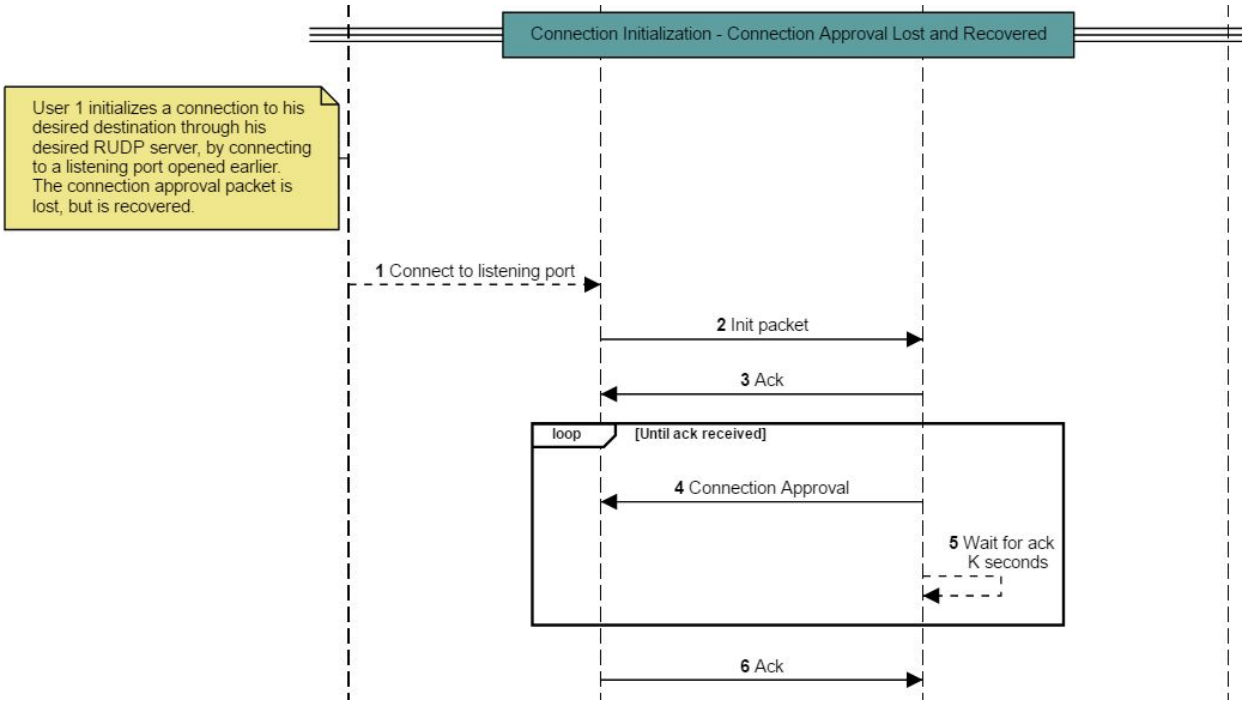
אתחול נכשל - אישור החיבור נאבד

אתחול החיבור נכשל משום שפאקטת אישור החיבור (connection approval) נאבדת והמערכת אינה מצליחה להתאושש. ברצף זה, K מייצג את ה-RTO של המערכת, M את המספר השידורים החוזרים המסקימלי, ו-L את פרק הזמן שמחכה מאתחל החיבור לאישור החיבור לפני שהוא מתנתק.



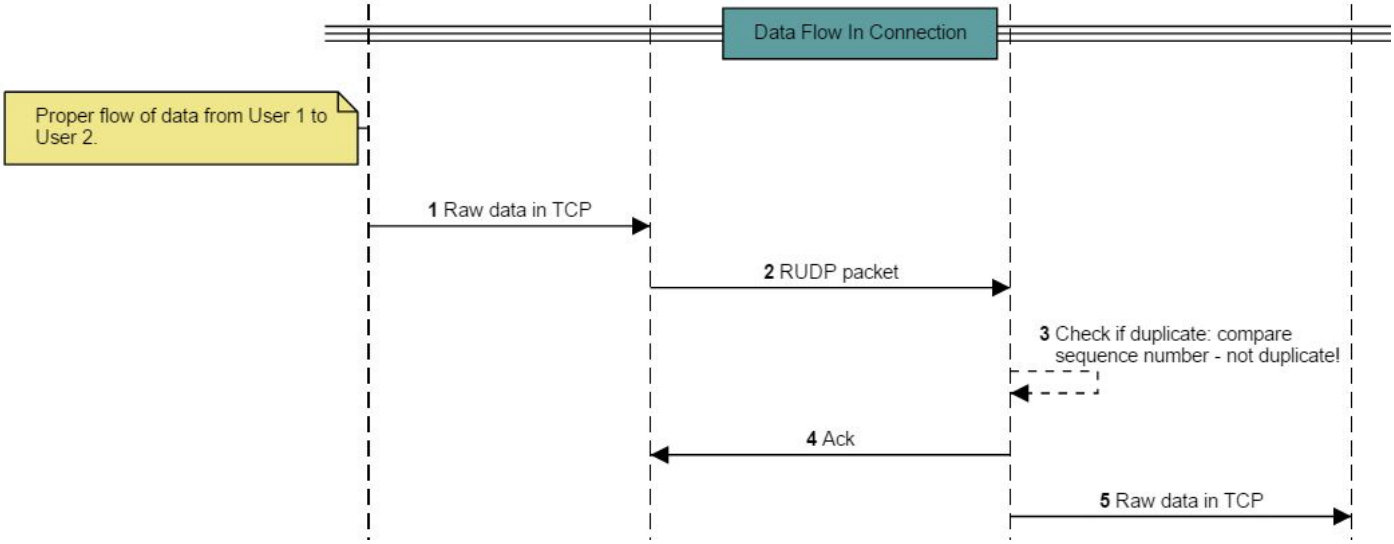
אתחול מתאושש

ייתכן גם שפאקטת אישור החיבור תיאבד, אך החיבור יצליח להתאושש,והאתחול יצליח, הודות למנגנון השידורים החוזרים. K מייצג את ה-RTO של המערכת (בעמוד הבא):



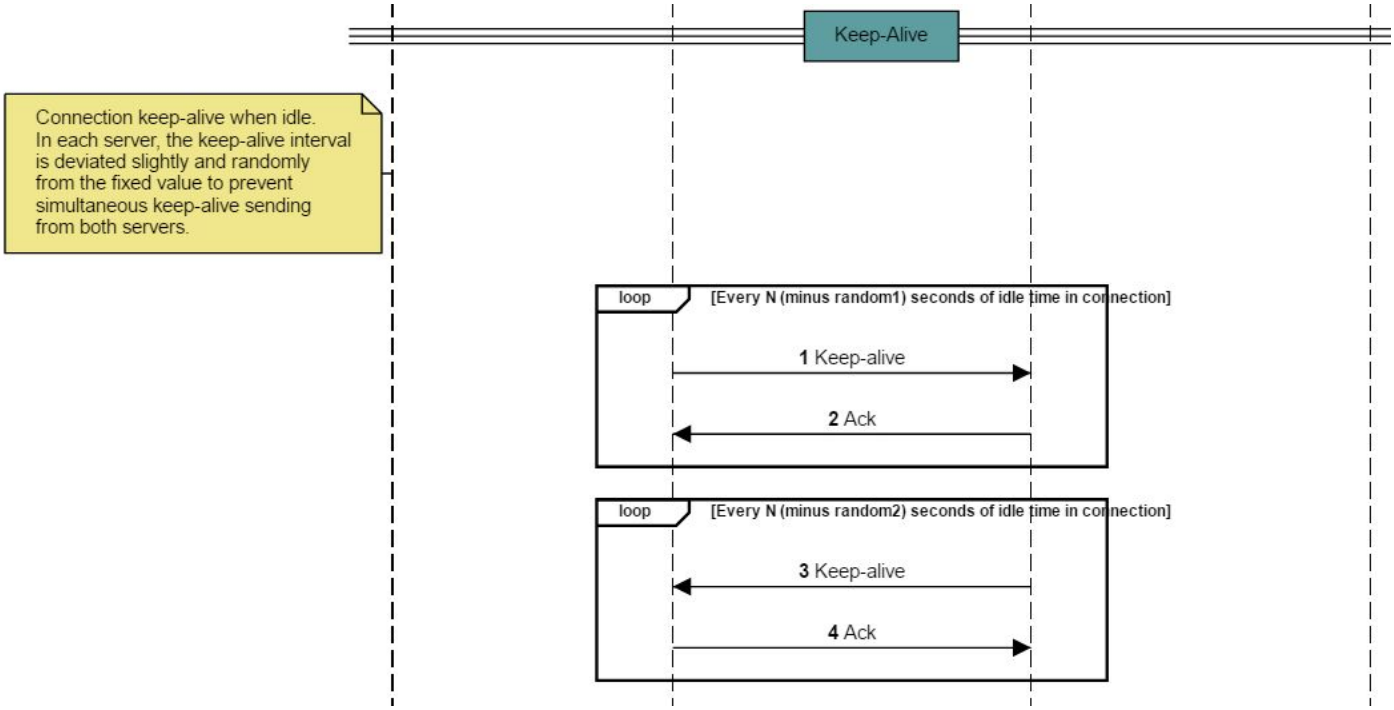
זרימת מידע תקינה

זרימת מידע בחיבור, ממשתמש 1 למשתמש 2, כולל פאקטת ACK.



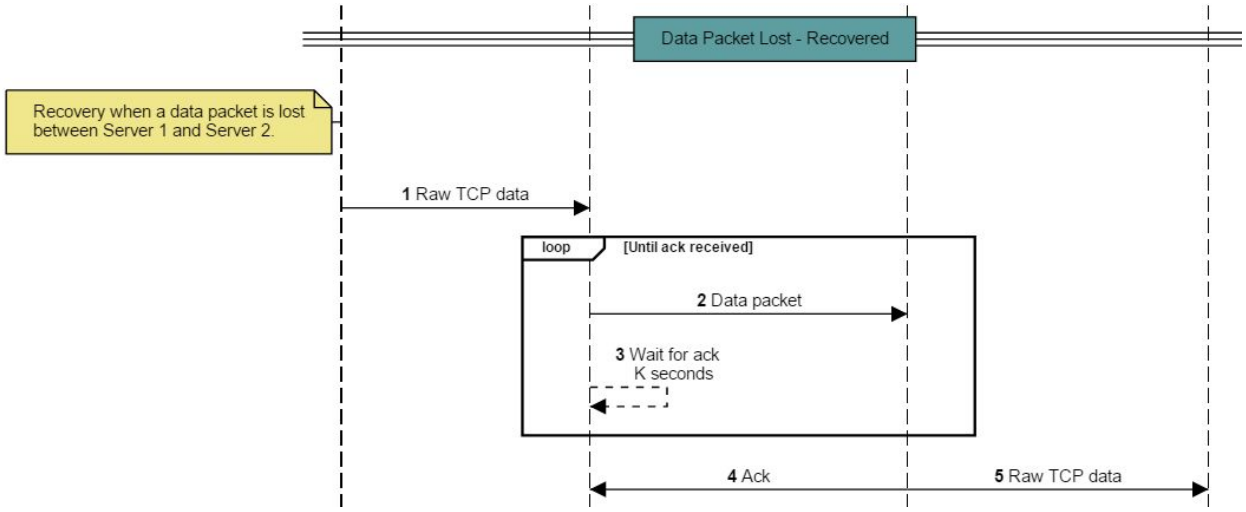
שידור Keep-Alive

שידור פאקטות keep-alive בין שני השרתים על מנת לשמור על החיבור. N מייצג את טיימאאוט ה-keep-alive הקבוע בממערכת. יש לשים לב שבכל שרת, טיימאאוט זה שונה במעט כדי למנוע שידורי keep-alive בו-זמניים.


Keep-Alive
Connection keep-alive when idle.
In each server, the keep-alive interval is deviated slightly and randomly from the fixed value to prevent simultaneous keep-alive sending from both servers.
loop
[Every N (minus random1) seconds of idle time in connection]
1 Keep-alive
2 Ack
loop
[Every N (minus random2) seconds of idle time in connection]
3 Keep-alive
4 Ack

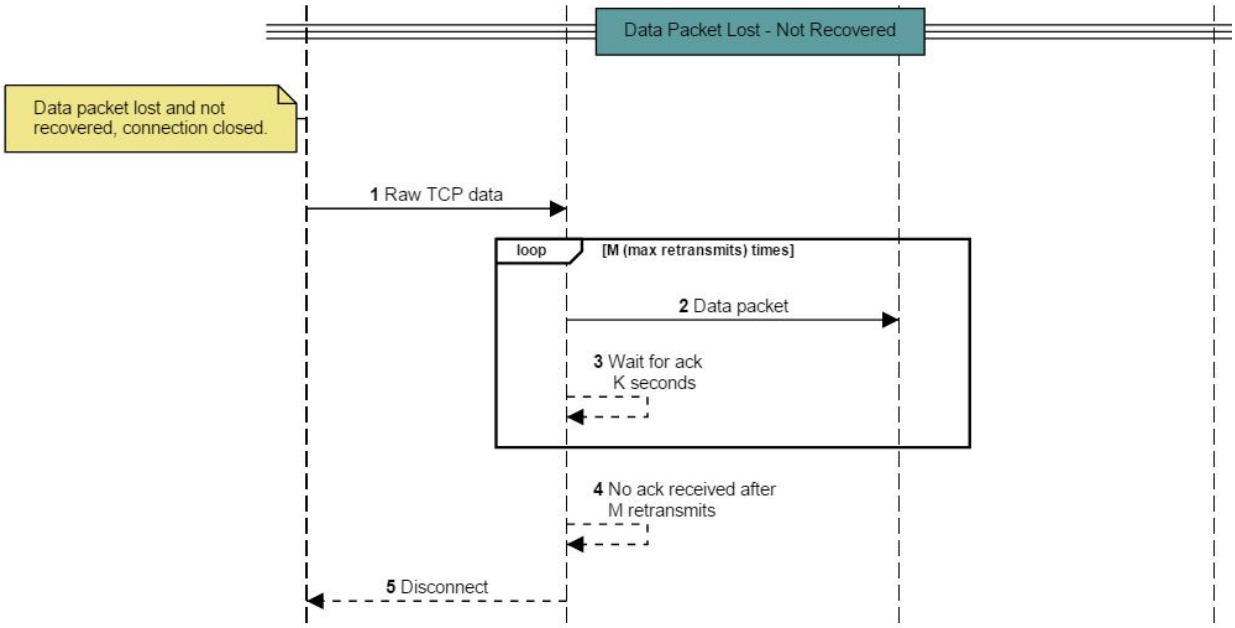
התאוששות מאיבוד מידע

פאקטת מידע נאבדת ברשת, והמערכת מצליחה להתאושש. היכולת של המערכת להתמודד עם איבוד מידע שכזה חשובה מאוד, כיוון שפרוטוקול RUDP מבטיח אמינות. K מייצג ברצף זה את ה-RTO של המערכת:



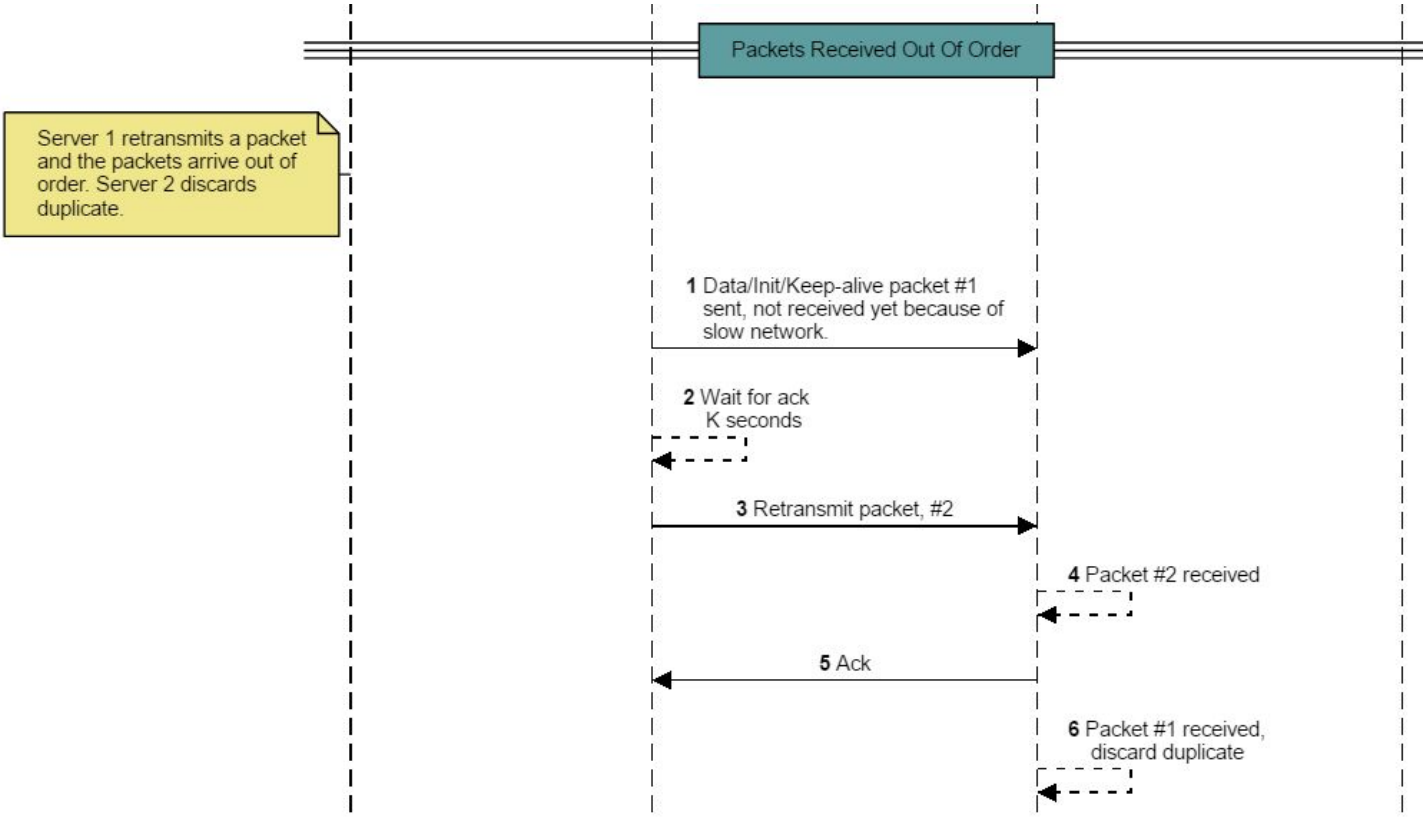
חוסר התאוששות מאיבוד מידע - סגירת חיבור

לא תמיד המערכת מצליחה להתאושש מאיבוד פאקטות. ייתכן ששרת 2 התנתק ללא הודעה, או שהרשת לא מאפשרת בכלל העברה של שדרים בין השרתים. לאחר M ניסיונות (המספר המקסימלי), החיבור נסגר. K מייצג את ה-RTO של המערכת.



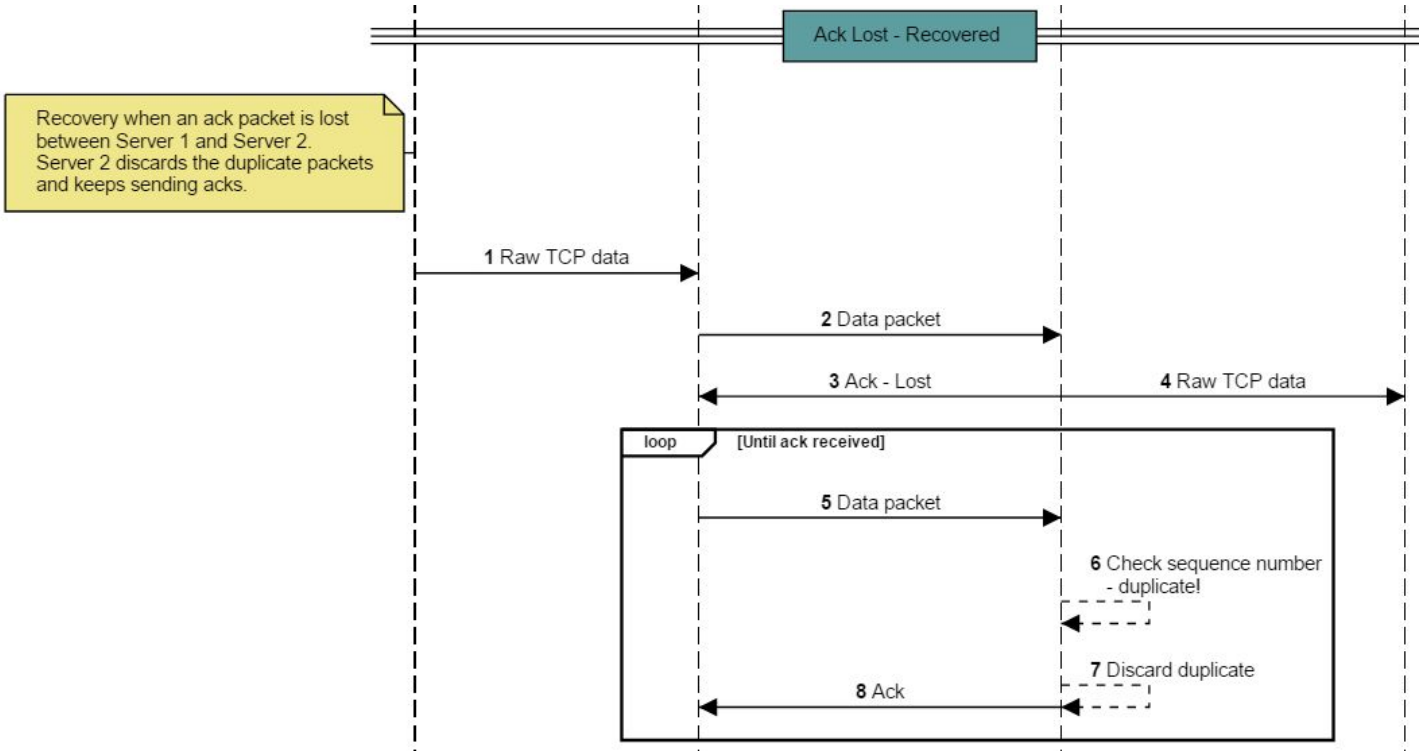
פאקטות מגיעות בסדר הלא נכון

הגעתן של שתי פאקטות מידע, אתחול או keep-alive בסדר הלא נכון, בגלל עיכובים ברשת. היות והן זהות לכל דבר ועניין, שרת 2 יתייחס אל הראשונה שקיבל וישלח עליה ACK, ומהשנייה יתעלם.



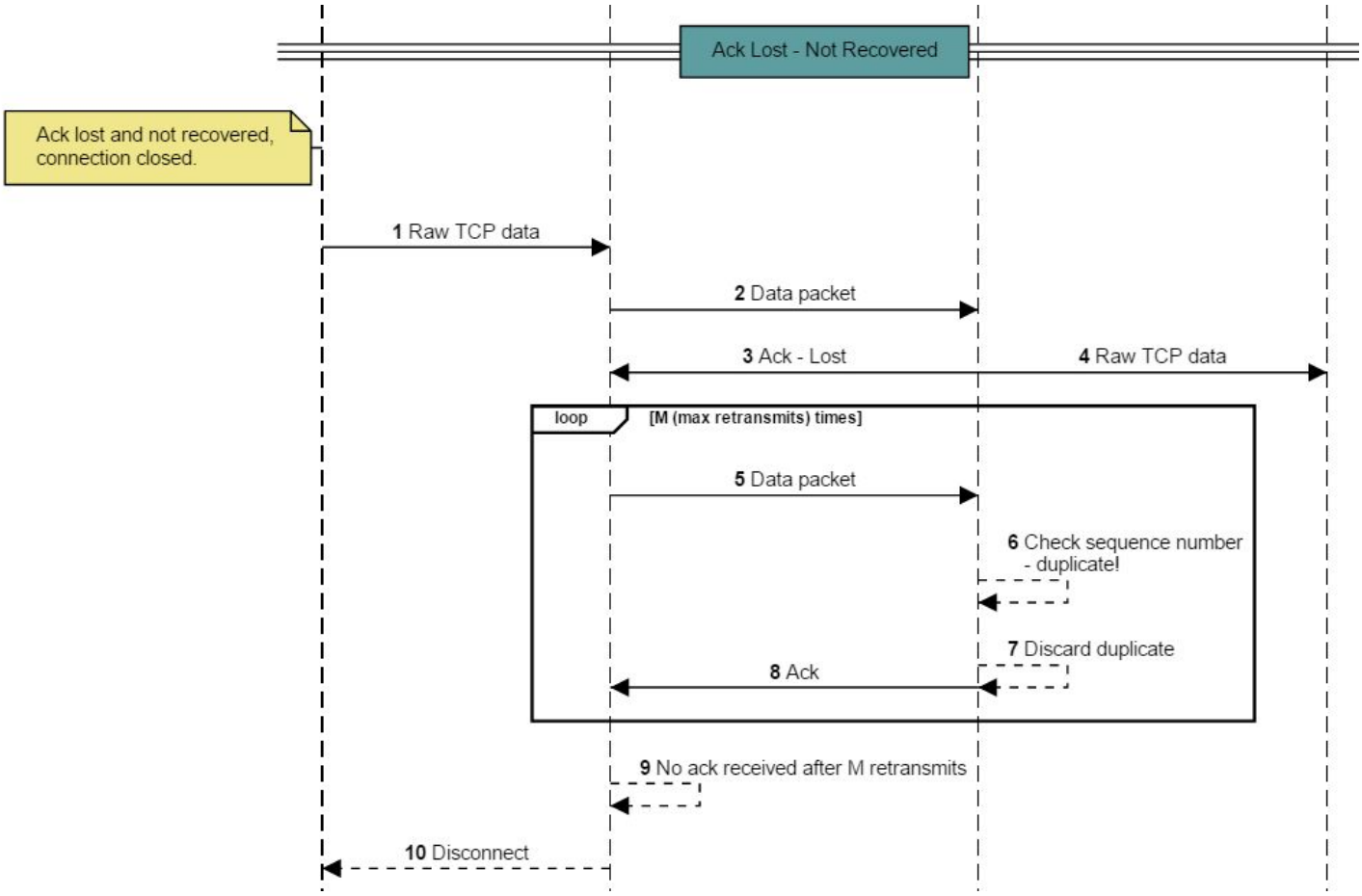
התאוששות מאיבוד פאקטת ACK

כפי שסקרנו בחלק התיאורטי, גם פאקטת ACK יכולה להיאבד. ברצף הבא מתואר תהליך ההתאוששות של המערכת מאיבוד של פאקטת ACK. שרת א' לא מקבל את ה-ACK ולכן משרד שוב את הפאקטה עד לקבלת ACK. יש לשים לב, ששרת 2 מזהה, לפי בדיקת המספרים הסידוריים של הפאקטות, שהפאקטות שהוא מקבל זהות לפאקטות שקיבל בעבר, ולכן פרט לשליחת ACK עליהן הוא אינו מתייחס אליהן.



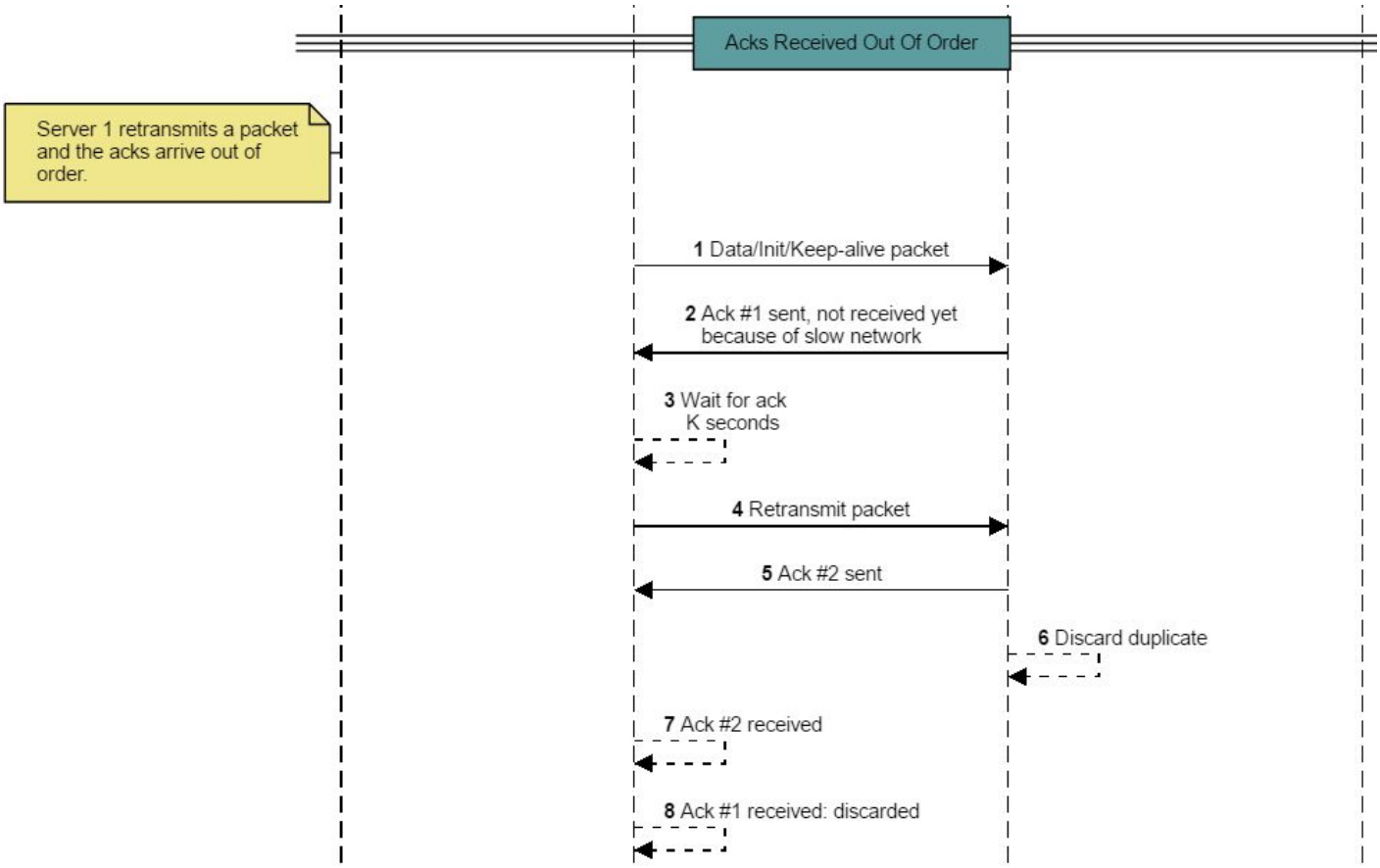
חוסר התאוששות מאיבוד פאקטת ACK - סגירת חיבור

בדומה לפאקטת מידע, גם פאקטת ACK יכולה להיאבד ברשת מספיק פעמים על מנת ששרת 1 ינתק את החיבור.



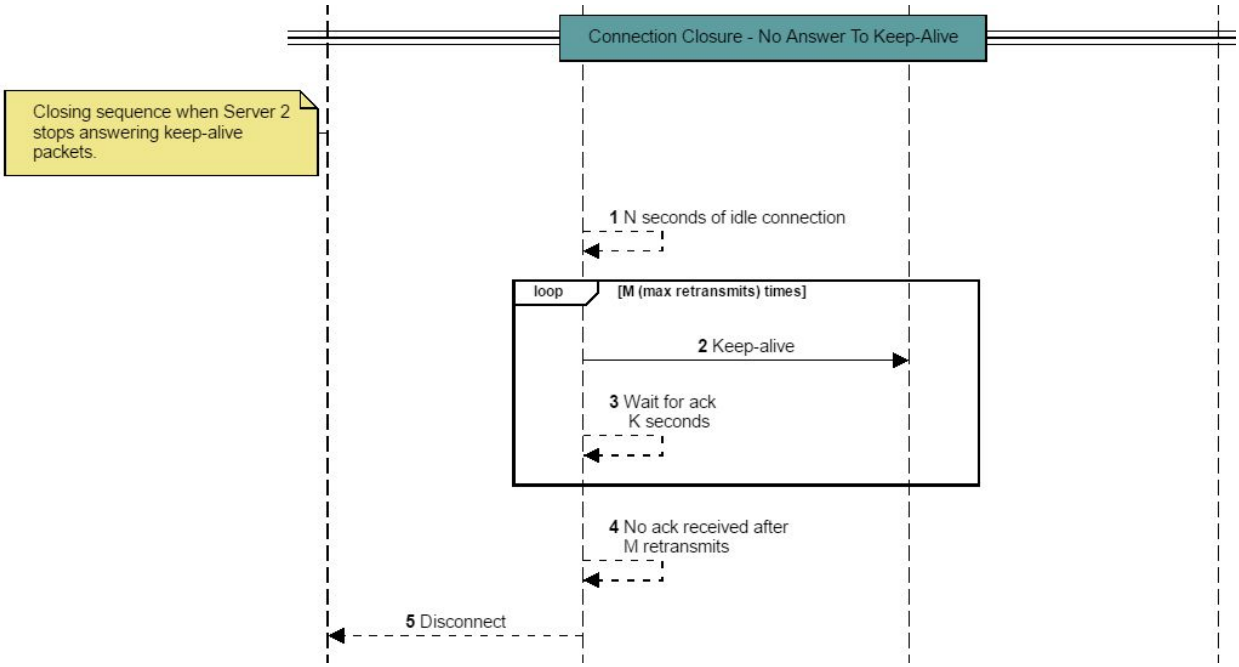
פאקטות ACK מגיעות בסדר הלא נכון

בדומה למתרחש בפאקטות מידע, גם פאקטות ACK יכולות להגיע בסדר הלא נכון. שוב, היות והן זהות, שרת 1 יתייחס לראשונה שיקבל, ויתעלם מהשנייה. כמו כן, שרת 2, שמקבל פעמיים את הפאקטה, יזהה את הכפילות על ידי המספרים הסידוריים ויתעלם מהשידור השני.



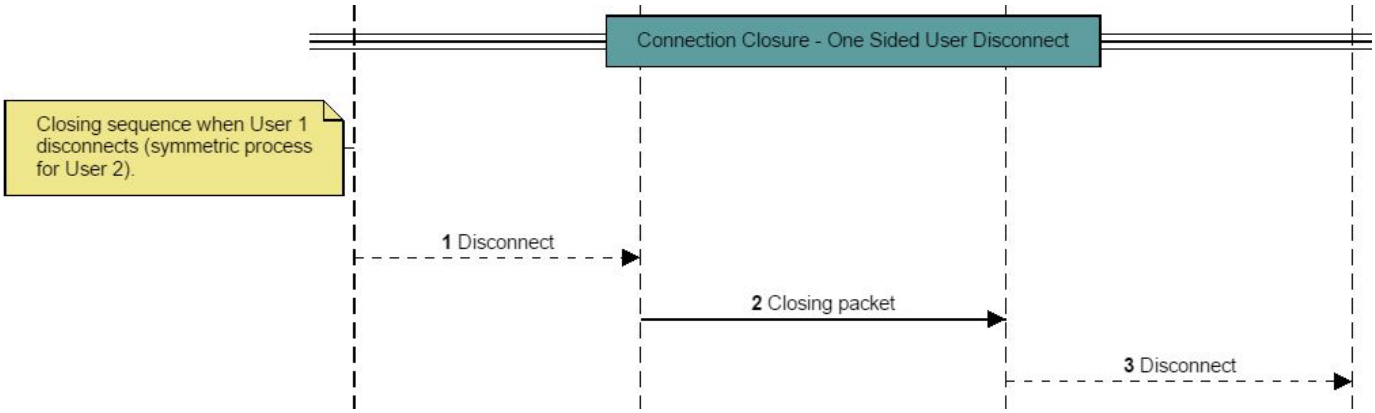
סגירת חיבור - אין מענה ל-Keep-Alive

שרת 1 לא מקבל מענה על פאקטות keep-alive שהוא שולח. לאחר מספר הניסיונות המקסימלי הוא מנתק את החיבור.



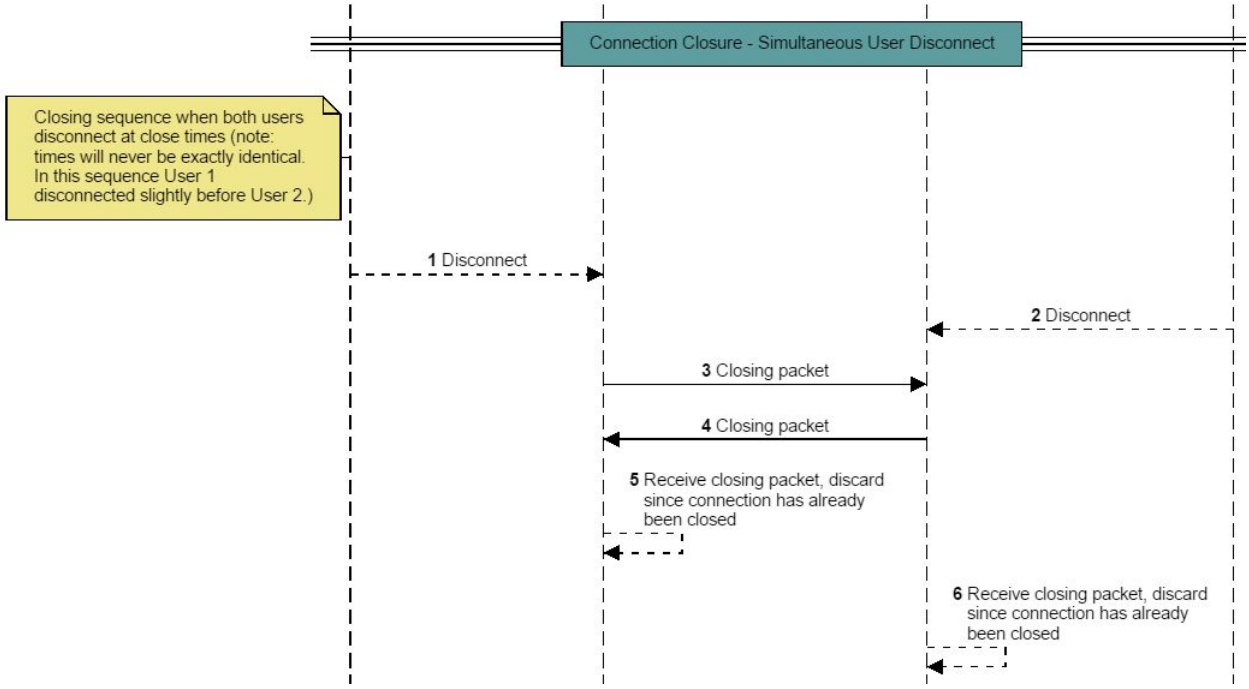
סגירת חיבור - ניתוק משתמש אחד

אחד ממשתמשי הקצה מתנתק.



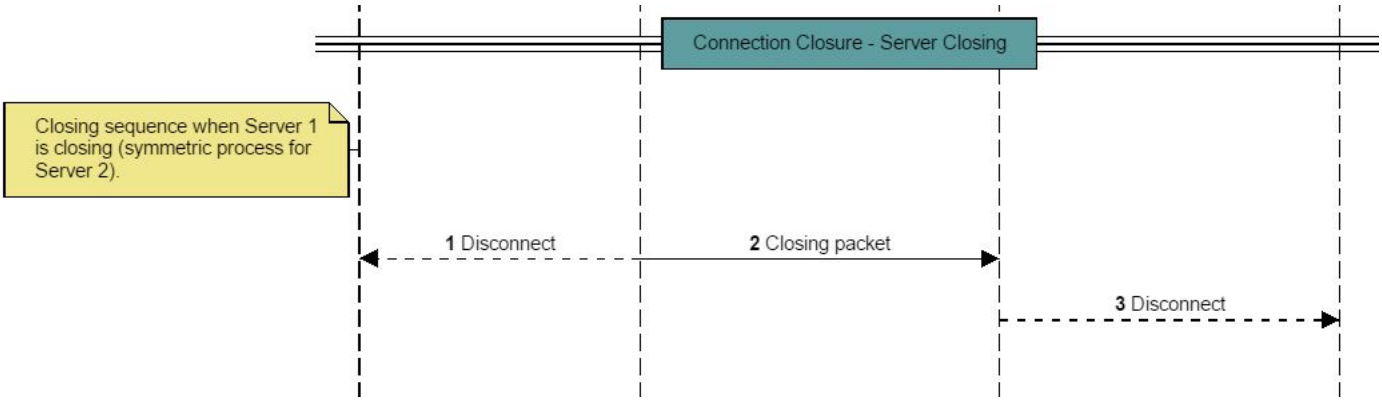
סגירת חיבור - שני המשתמשים מתנתקים

ייתכן גם ששני המשתמשים יתנתקו בזמנים קרובים מאוד זה לזה, כך ששני השרתים ישלחו פאקטות סגירה לפני קבלת פאקטות סגירה מהשרת השני. במקרה זה הם מתעלמים מהפאקטות המיותרות:



סגירת חיבור - שרת נסגר

לבסוף, נעבור על רצף האירועים שמתרחש כאשר שרת נסגר, מפאת שגיאה או סגירה מסודרת.

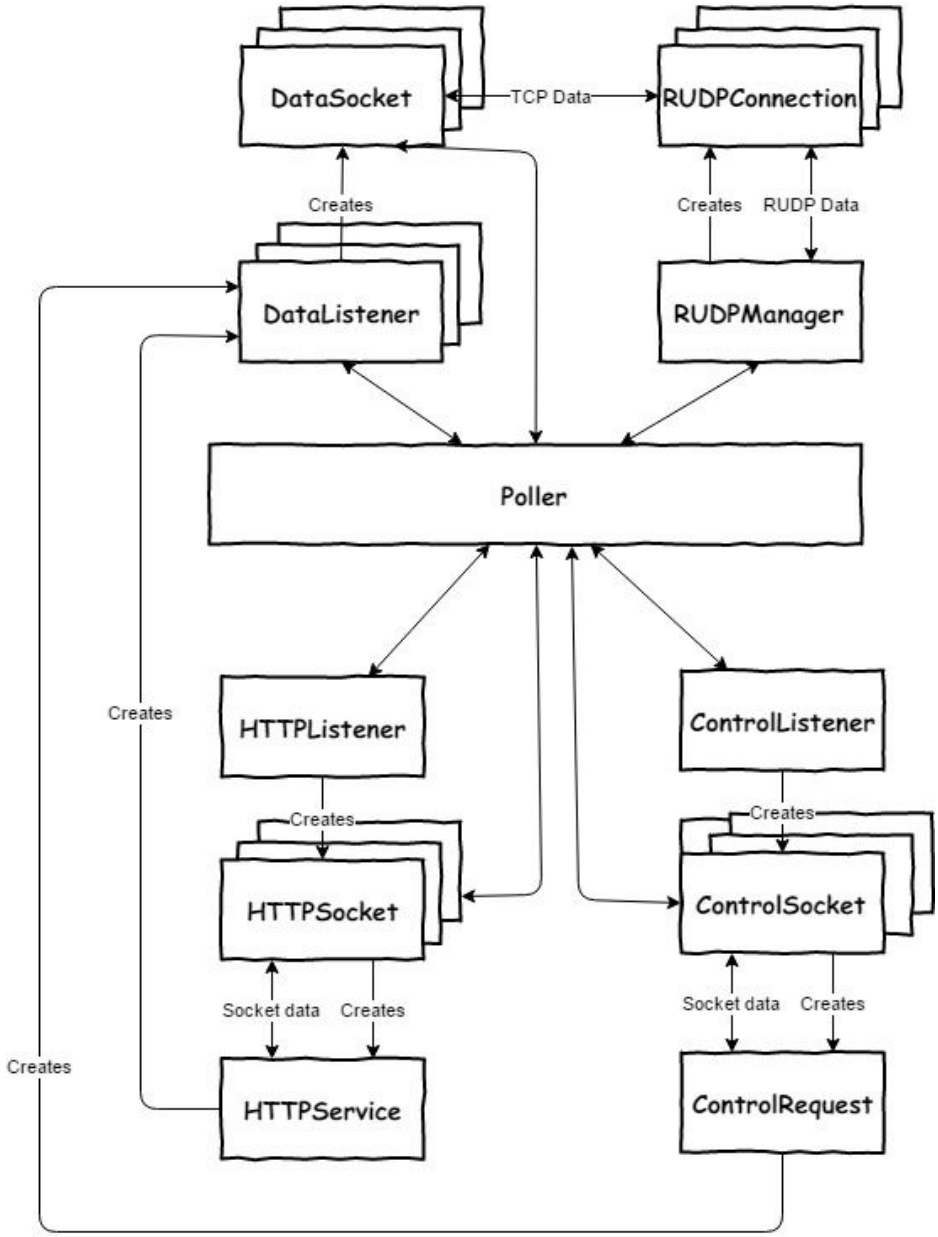


מימוש

עצמים ויחסי הגומלין ביניהם

העצמים העיקריים בפרויקט, ויחסי הגומלין ביניהם, מתוארים בדיאגרמות הבאות:

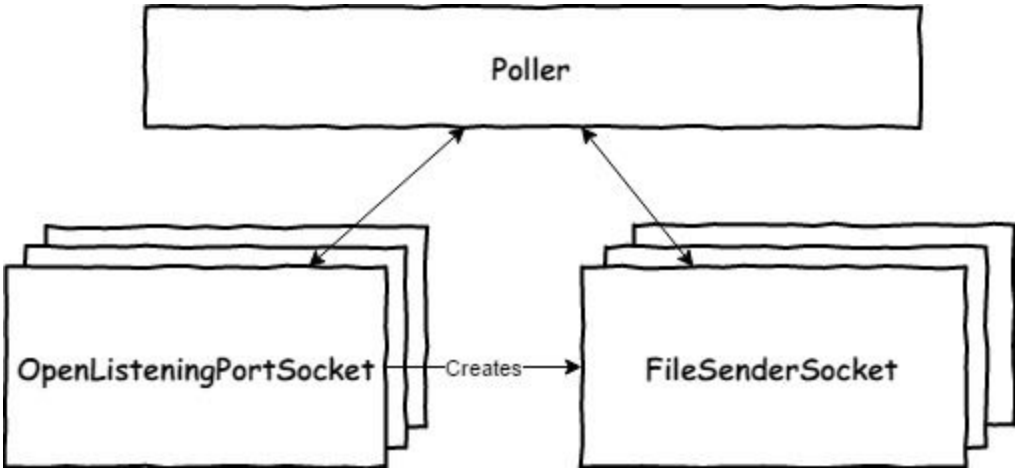
תכנית השרת - Server



עצם	תיאור
Poller	מנהל את הלולאה המרכזית של תכנית. ה-Poller שומר רישום של כל הסוקטים במערכת ומבצע בלולאה קריאת poll עליהם. בכך מבטיח ריצה אסינכרונית של התכנית. כאשר הוא מקבל מקריאת ה-poll אירועי I/O, הוא מעביר אותם לאובייקטים המתאימים.
RUDPManager	מנהל את חיבורי ה-RUDP לשרתים אחרים: פותח אותם, סוגר אותם, ומקבל ושולח עבורם את כל פאקטות ה-RUDP כיוון שהוא העצם המחזיק בסוקט ה-UDP.
RUDPConnection	מייצג חיבור RUDP בודד, המחובר מצד אחד לשרת RUDP אחר ומהצד השני למשתמש TCP. מנהל את הלוגיקה של החיבור - שליחה וקבלה של פאקטות, המתנה לACK-ים, שידורים חוזרים, שליחת Keep-Alive וכו'. שליחה וקבלה של הפאקטות עצמן נעשית על ידי ה-RUDPManager. מקבל מ-DataSocket מידע שהתקבל מהמשתמש, ומעביר ל-DataSocket מידע שהתקבל עבור המשתמש.
DataListener	סוקט המאזין לחיבורי Data. נוצר על ידי בקשת פורט, דרך פרוטוקול הבקרה או דרך עמוד האינטרנט (בקשת HTTP). יש לו זמן חיים מוגדר, המוקצב על ידי הבקשה. כאשר מתחברים אליו, יוצר DataSocket.
DataSocket	סוקט המנהל חיבור מידעי עם משתמש הקצה. מקבל מידע ממשתמש הקצה ומעביר אותו ל-RUDPConnection המתאים. מקבל מידע מ-RUDPConnection ומעביר אותו למשתמש הקצה.
ControlListener	סוקט המאזין לחיבורי בקרה. כאשר מתחברים אליו, יוצר ControlSocket.
ControlSocket	סוקט המנהל חיבור בקרה עם משתמש הקצה. מקבל ממשתמש הקצה בקשות בפרוטוקול הבקרה, ויוצר בהתאם לסוג הבקשה ControlRequest מתאים. שולח למשתמש הקצה מענה בהתאם ללוגיקה של עצם ה-ControlRequest.
ControlRequest	עצם כללי המייצג בקשת בקרה. קיימים שני עצמים שונים היורשים ממנו - StatisticsRequest ו-ConnectRequest. מקבלים את הבקשה מ-ControlSocket, מפרשים אותו על פי הלוגיקה של הבקשה ומעבירים ל-ControlSocket מענה, שנשלח למשתמש. StatisticsRequest בודק סטטיסטיקות בעוד שConnectRequest יוצר DataListener לבקשת המשתמש.

HTTPListener	סוקט המאזין לחיבורי HTTP. כאשר מתחברים אליו, יוצר HTTPSocket.
HTTPSocket	סוקט המנהל חיבור HTTP עם הדפדפן. מקבל מהדפדפן בקשות בפרוטוקול HTTP, ויוצר בהתאם לסוג הבקשה HTTPService מתאים. שולח לדפדפן מענה בהתאם ללוגיקה של עצם ה-HTTPService.
HTTPService	עצם כללי המייצג שירות HTTP. לכל דף באתר האינטרנט קיים Service משלו: ConnectionsService המשרת את דף המידע על החיבורים, DataPortService המשרת את בקשת הפורט, ו-FileService המשרת כל קובץ סטטי. אלה מקבלים את הבקשה מ-HTTPSocket, מפרשים אותה על פי פרוטוקול HTTP ולוגיקת השירות, ומעבירים ל-HTTPSocket מענה, שנשלח לדפדפן. DataPortService יוצר DataListener בהתאם לבקשת המשתמש.

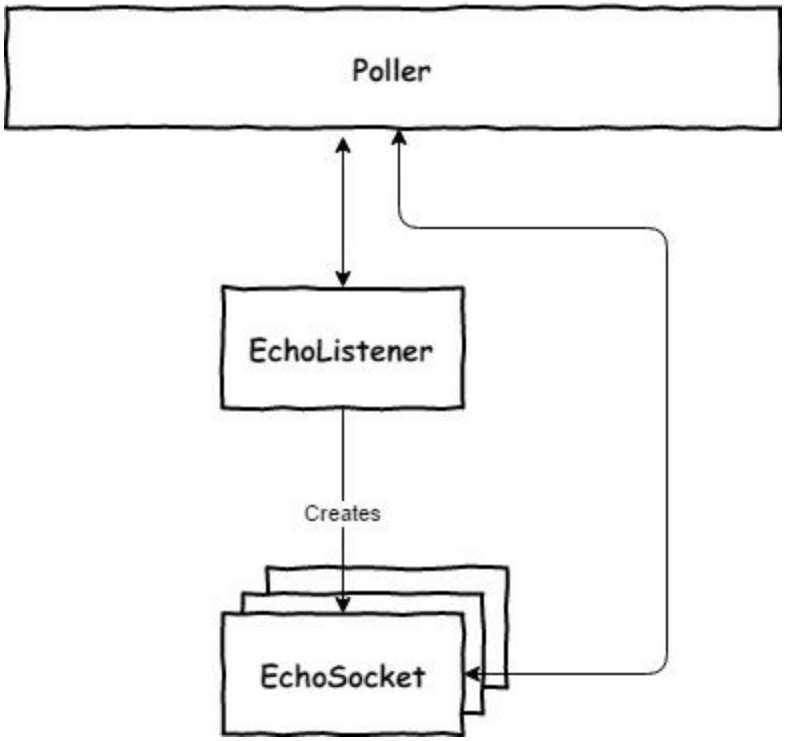
תכנית הבדיקה השולחת - Sender



עצם	תיאור
Poller	מנהל את הלולאה המרכזית של תכנית. ה-Poller שומר רישום של כל הסוקטים במערכת ומבצע בלולאה קריאת poll עליהם. בכך מבטיח ריצה אסינכרונית של התכנית. כאשר הוא מקבל מקריאת ה-poll אירועי I/O, הוא מעביר אותם לאובייקטים המתאימים.

OpenListeningPortSocket	סוקט המתחבר לשרת RUDP בערוץ הבקרה, ומבקש פתיחת פורט ליעד מסוים דרך שרת מעבר מסוים. לאחר קבלת הפורט יוצר מספר עצמי FileSenderSocket עם הפורט הנ"ל.
FileSenderSocket	סוקט המתחבר לפורט מאזין בשרת ה-RUDP. לאחר החיבור שולח קובץ ומחכה לקבלת תשובה. אם התשובה שקיבל אינה זהה לקובץ ששלח, מעלה שגיאה.

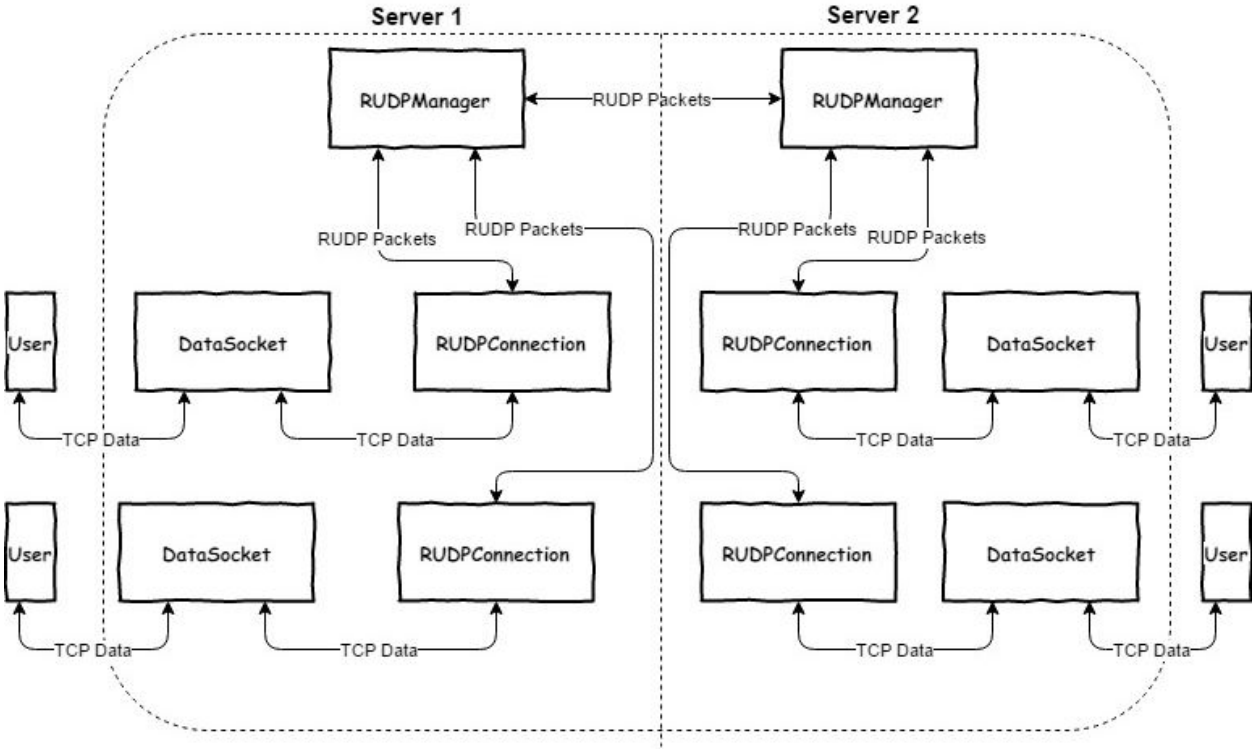
תכנית הבדיקה המהדהדת - Echoer



עצם	תיאור
Poller	מנהל את הלולאה המרכזית של תכנית. ה-Poller שומר רישום של כל הסוקטים במערכת ומבצע בלולאה קריאת poll עליהם. בכך מבטיח ריצה אסינכרונית של התכנית. כאשר הוא מקבל מקריאת ה-poll אירועי I/O, הוא מעביר אותם לאובייקטים המתאימים.
EchoListener	סוקט המאזין לחיבורים. כאשר מתחברים אליו, יוצר EchoSocket/
EchoSocket	סוקט המחזיר את כל המידע שהוא מקבל.

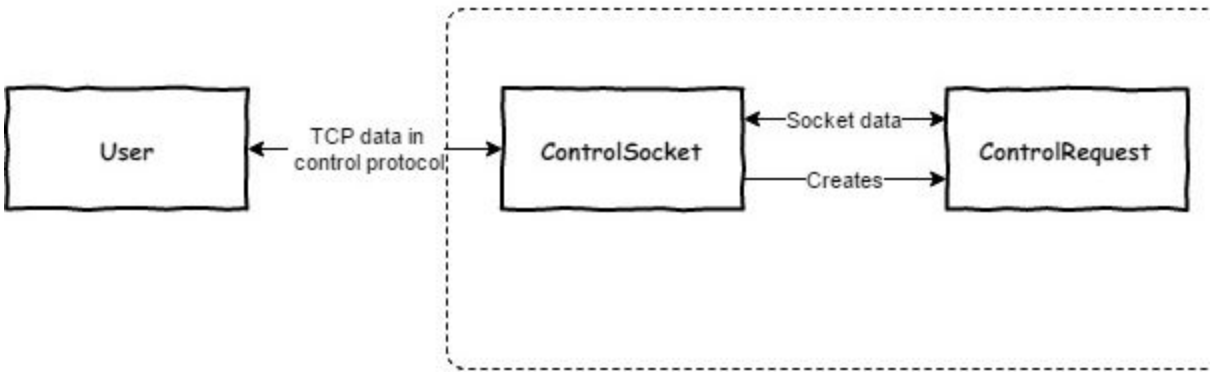
עצמים בתכנית ה-Server המעורבים בחיבור RUDP

העצמים בתוך הצורה המקווקוות הם עצמים של תכנית ה-Server, ועצמים הנמצאים מחוץ לצורה המקווקוות הם עצמים חיצוניים (משתמשי הקצה). הדיאגרמה מתארת שני חיבורים, בין ארבעה משתמשים שונים, העוברים דרך אותם שני השרתים. היחסים בין העצמים תוארו כבר בפתיח הפרק.



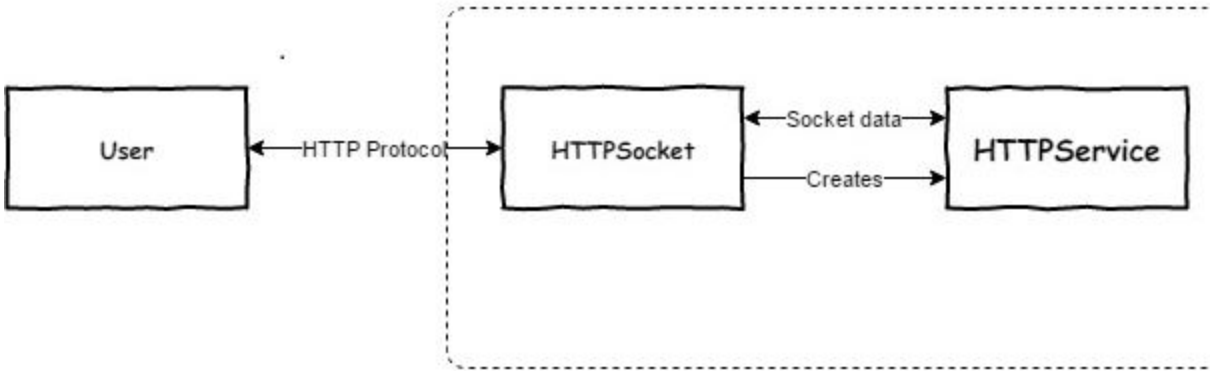
עצמים בתכנית ה-Server המעורבים בחיבור בקרה

העצמים בתוך הצורה המקווקוות הם עצמים של תכנית ה-Server, ועצמים הנמצאים מחוץ לצורה המקווקוות הם עצמים חיצוניים (משתמשי הקצה). היחסים בין העצמים תוארו כבר בפתיח הפרק.



עצמים בתכנית ה-Server המעורבים בחיבור HTTP

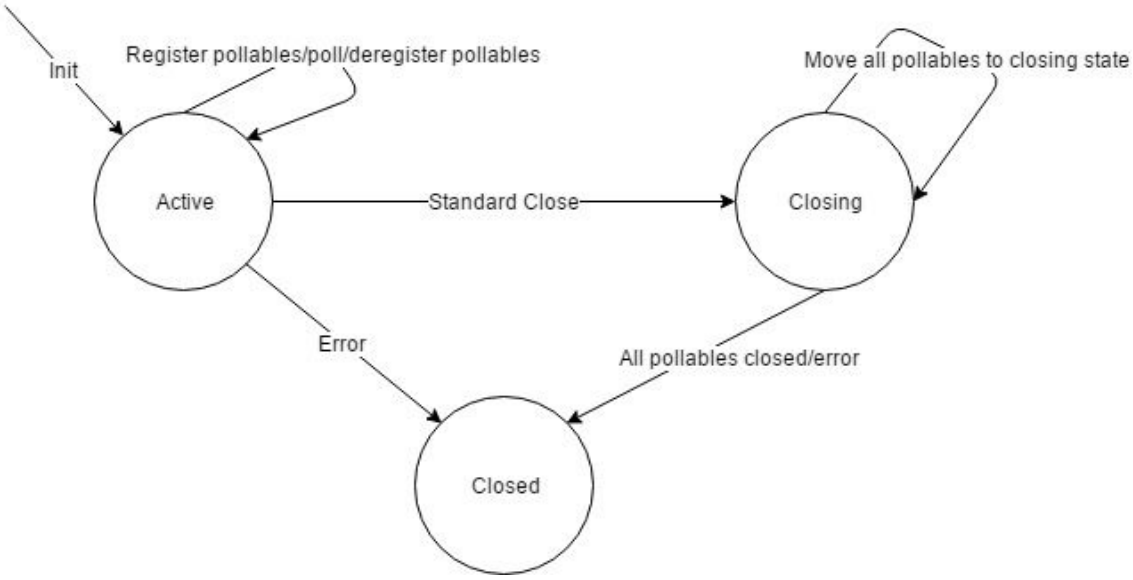
העצמים בתוך הצורה המקווקוות הם עצמים של תכנית ה-Server, ועצמים הנמצאים מחוץ לצורה המקווקוות הם עצמים חיצוניים (משתמשי הקצה). היחסים בין העצמים תוארו כבר בפתיח הפרק.



מכונות מצבים

רוב העצמים בפרויקט המבצעים לוגיקה מסתמכים על תצורת State Machine, או מכונת מצבים. לפי תצורה זו, העצם נמצא בכל זמן נתון במצב מסוים, ועובר בין מצבים בהתאם לקלט שהוא מקבל, לפלט שהוא שולח ולחלוף הזמן.

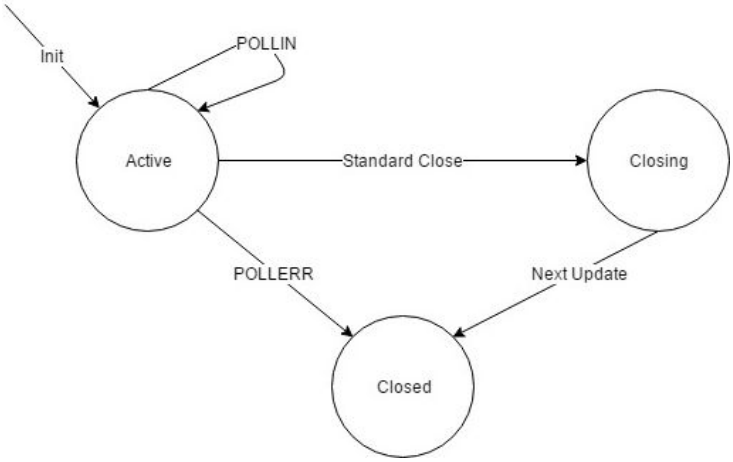
ה-Poller



מצב	תיאור
Active	המצב הפעיל. במצב זה ה-Poller נמצא בלולאה הראשית שלו: רושם עצמים (Pollables), מעדכן אותם, קורא קריאות poll עליהם ונותן להם לבצע את הפעולות הנדרשות על כל אירוע שמתקבל. עצם שמעלה שגיאה נסגר, אך אינו גורם לסגירת ה-Poller עצמו.
Closing	במצב זה ה-Poller נמצא במצב סגירה. הוא עדיין נמצא בלולאה הראשית, אך מעביר את כל העצמים למצב הסגירה שלהם.
Closed	ה-Poller סגור והתכנית הסתיימה.
מעבר	**תיאור**
Register pollables	רישום עצמים (pollables) ל-Poller
Poll	קריאת Poll על כל הסוקטים והעברת האירועים לPollables
Deregister pollables	מחיקת עצמים מה-Poller במקרה של שגיאה או סגירה מסודרת שלהם.
Standard Close	סגירה מסודרת של התכנית - Ctrl-C.
Move all pollables to closing state	העברת כל העצמים למצב הסגירה שלהם.
All pollables closed	כל העצמים נסגרו ואפשר לסגור את התכנית
Error	שגיאה ב-Poller עצמו ולא באחד מהעצמים (ה-Pollables)

סוקט מאזין - Listener Socket

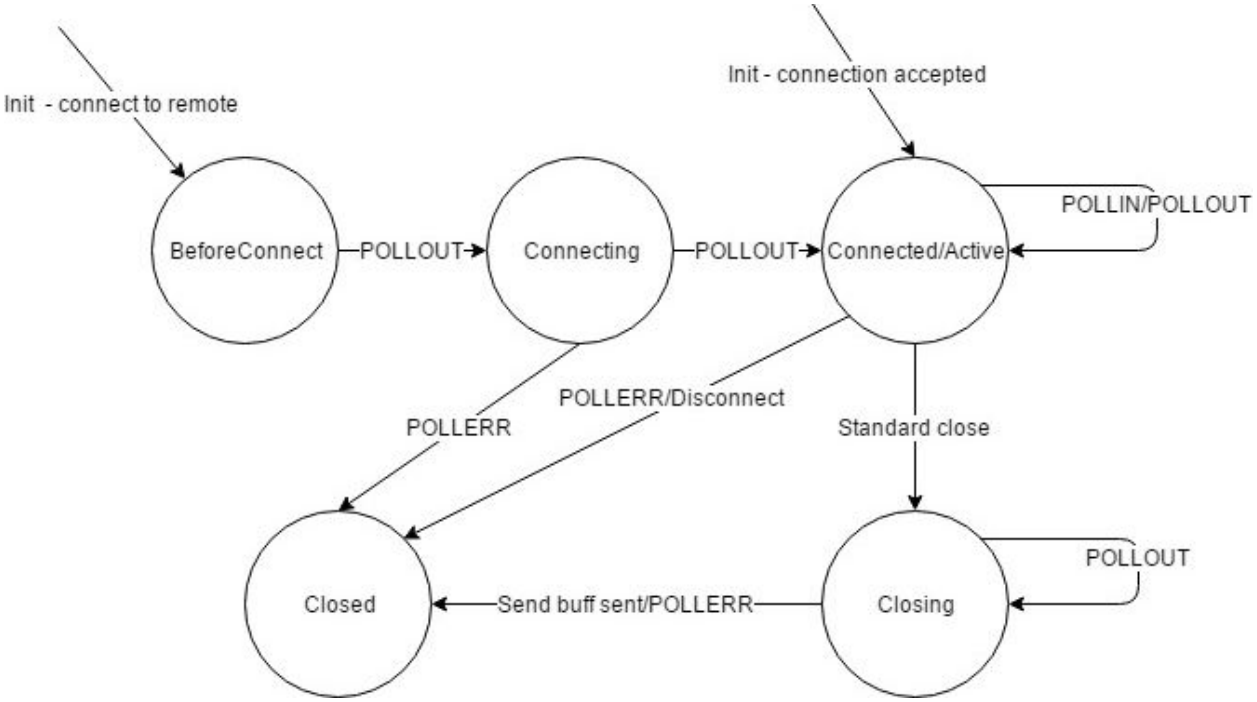
עצם מסוג Pollable המחזיק בסוקט שמאזין לחיבורים:
DataListener, ControlListener, HTTPListener, EchoListener



מצב	תיאור
Active	המצב הפעיל. העצם מאזין לחיבורים.
Closing	העצם נמצא במצב סגירה, ולא מקבל שום סוג של אירועים.
Closed	העצם סגור.
מעבר	**תיאור**
POLLIN	אירוע מסוג קריאת קלט
POLLERR	אירוע מסוג שגיאה בסוקט
Standard Close	סגירה מסודרת של העצם, בדרך כלל מתוך הלולאה הראשית של ה-Poller
Next Update	העדכון הבא של העצם, במסגרת הלולאה הראשית של ה-Poller

סוקט שאינו מאזין - Non-Listener Socket

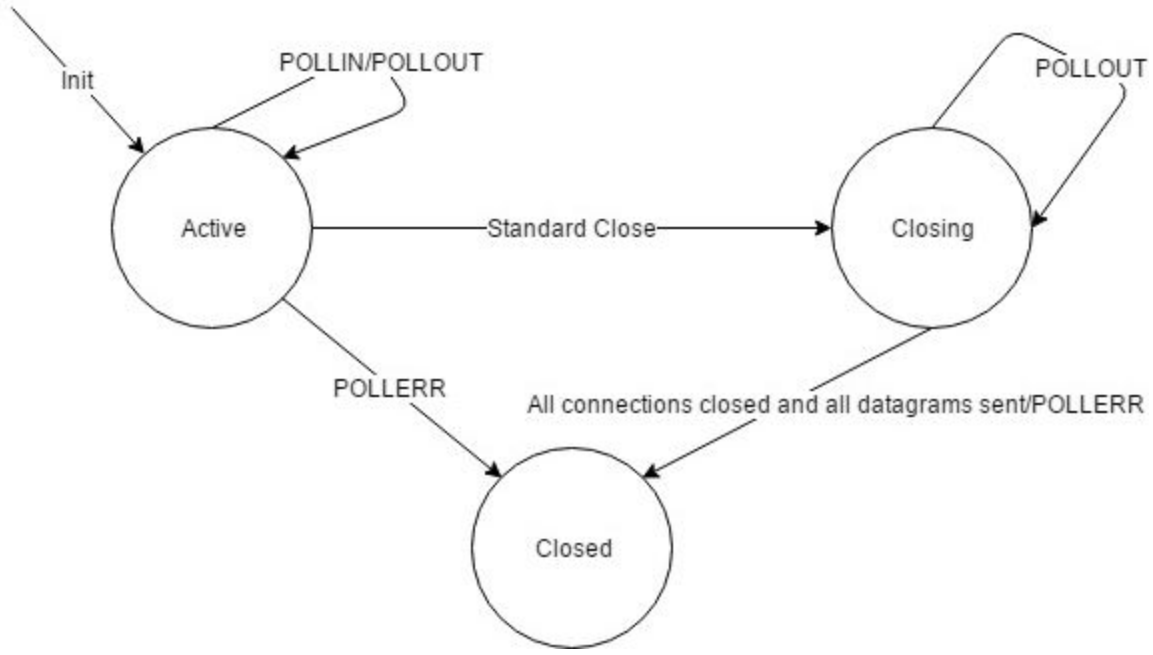
עצם מסוג Pollable המחזיק בסוקט ששולח ומקבל מידע:
DataSocket, ControlSocket, HTTPSocket, EchoSocket, OpenListenerPortSocket, FileSenderSocket



מצב	תיאור
BeforeConnect	המצב ההתחלתי כאשר הסוקט לא התקבל מהאזנה לחיבור, אלא נוצר כדי להתחבר ליעד.
Connecting	הסוקט נמצא בתהליך חיבור ליעד.
Connected/Active	המצב הפעיל. העצם שולח ומקבל מידע. מצב זה הוא המצב ההתחלתי כאשר הסוקט מתקבל מהאזנה לחיבור.
Closing	העצם נמצא במצב סגירה, ומקבל רק אירועי POLLOUT כדי לסיים לשלוח את ה-buffer המחכה לשליחה שלו.
Closed	העצם סגור.
מעבר	**תיאור**
POLLIN	אירוע מסוג קריאת קלט
POLLOUT	אירוע מסוג כתיבת פלט
POLLERR	אירוע מסוג שגיאה בסוקט
Disconnect	התנתקות של הצד השני בחיבור.
Standard Close	סגירה מסודרת של העצם, בדרך כלל מתוך הלולאה הראשית של ה-Poller
Send buff sent	העצם סיים לשלוח את ה-buffer המחכה לשליחה שלו.

מנהל חיבורים - RUDP Manager

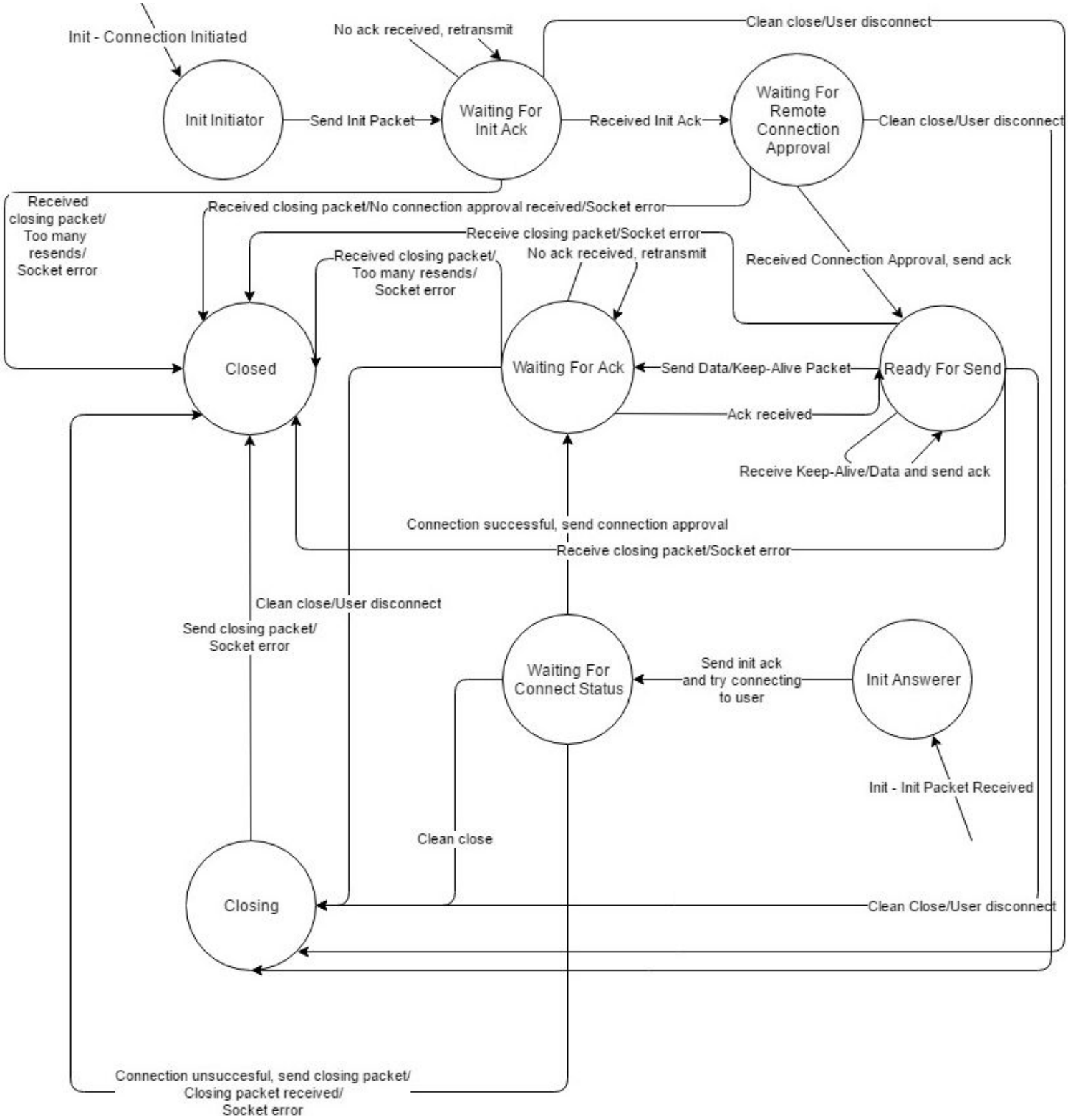
עצם מסוג Pollable המחזיק בסוקט UDP.



מצב	תיאור
Active	המצב הפעיל. העצם שולח ומקבל פאקטות, יוצר חיבורים חדשים וסוגר חיבורים קיימים.
Closing	העצם נמצא במצב סגירה, ומקבל רק אירועי POLLOUT כדי לסיים לשלוח את הפאקטות שנותרו לשליחה.
Closed	העצם סגור.
מעבר	**תיאור**
POLLIN	אירוע מסוג קריאת קלט
POLLOUT	אירוע מסוג כתיבת פלט
POLLERR	אירוע מסוג שגיאה בסוקט
Standard Close	סגירה מסודרת של העצם, בדרך כלל מתוך הלולאה הראשית של ה-Poller
All connections closed and all datagrams sent	העצם סיים לסגור את כל החיבורים שלו ולשלוח את כל הפאקטות שנותרו לשליחה.

חיבור - RUDP Connection

עצם מסוג חיבור שאינו מחזיק בסוקט ולכן אינו Pollable, אך מנהל את רוב הלוגיקה של פרוטוקול RUDP.

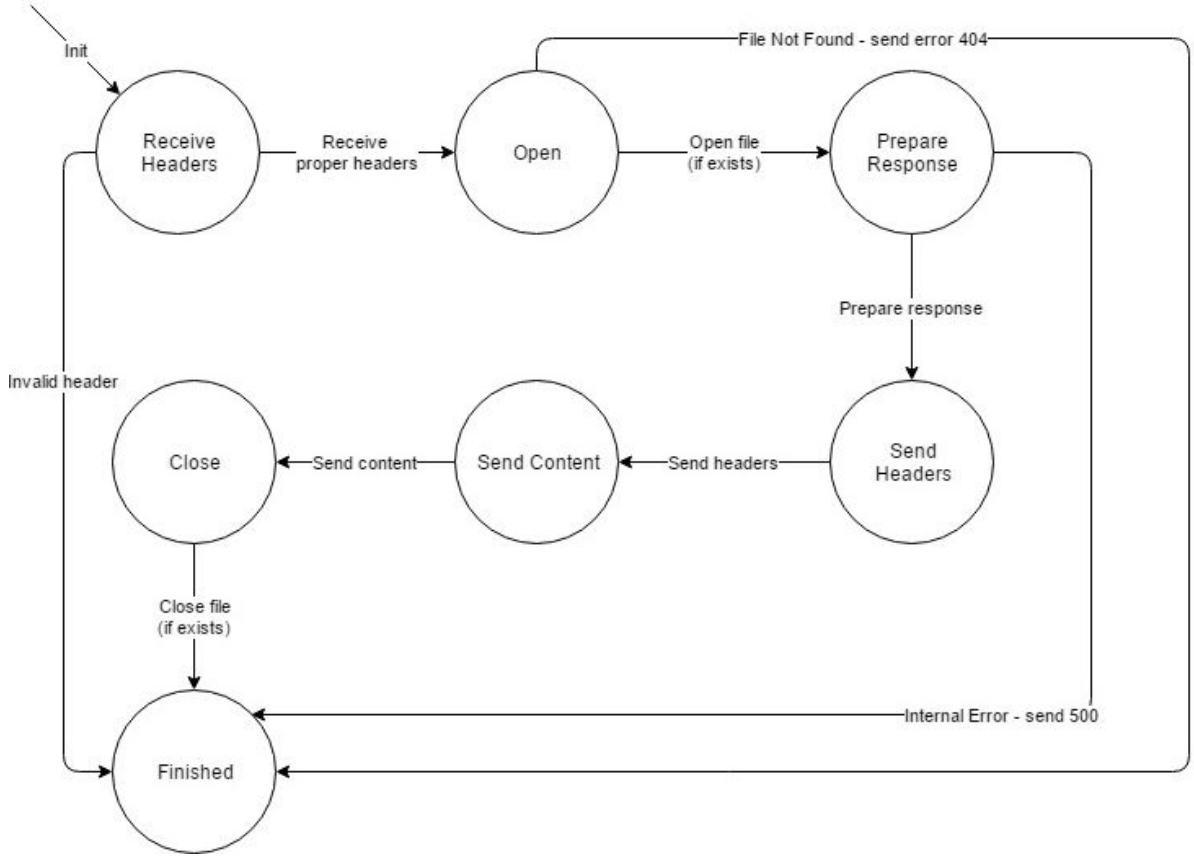


מצב	תיאור
Init Initiator	המצב ההתחלתי בו עצם החיבור הוא חלק מהשרת שיוזם את החיבור, ולא חלק מהשרת שמגיב ליוזמת חיבור.
Init Answerer	המצב ההתחלתי בו עצם החיבור הוא חלק מהשרת שמגיב ליוזמת חיבור, ולא חלק מהשרת שיוזם את החיבור.
Waiting For Init Ack	עצם החיבור מחכה לפאקטת ACK על פאקטת האתחול.
Waiting For Remote Connection Approval	עצם החיבור מחכה לאישור חיבור מהשרת המרוחק.
Ready For Send	עצם החיבור מוכן לשליחת פאקטות.
Waiting For Ack	עצם החיבור מחכה לפאקטת ACK על פאקטה ששלח.
Waiting For Connect Status	עצם החיבור יצר DataSocket שמטרתו להתחבר למשתמש הקצה המבוקש, ומחכה לתוצאת ניסיון החיבור
Closing	העצם נמצא במצב סגירה.
Closed	העצם סגור.
מעבר	**תיאור**
Connection initiated	החיבור אותחל כחיבור יוזם
Init packet received	החיבור אותחל כחיבור מגיב
Send Init Packet	שליחת פאקטת אתחול לשרת המרוחק
No ack received, retransmit	לאחר זמן ההמתנה הקבוע במערכת, לא התקבלה פאקטת ACK על הפאקטה שנשלחה ולכן היא שודרה מחדש
Received Init Ack	קבלת פאקטת ACK על פאקטת האתחול
Received connection approval, send ack	עצם החיבור קיבל אישור חיבור מהשרת המרוחק ושולח על כך ACK
No connection approval received	לאחר זמן ההמתנה הקבוע במערכת, לא התקבל אישור חיבור מהשרת המרוחק
Receive keep-alive/data and send ack	עצם החיבור מקבל פאקטת keep-alive או פאקטת מידע ושולח עליה ACK לשרת המרוחק.
Send keep-alive/data packet	עצם החיבור שולח פאקטת keep-alive או פאקטת מידע
Ack received	התקבלה פאקטת ACK על פאקטה שנשלחה

Received closing packet	התקבלה פאקטת סגירה
Too many resends	עצם החיבור הגיע למספר המקסימלי של שידורים חוזרים של פאקטה, ולא קיבל פאקטת ACK עליה
User Disconnect	משתמש הקצה המחובר לשרת התנתק
Clean Close	העצם נסגר בצורה מסודרת, בדרך כלל בגלל סגירה של ה-Poller
Send init ack and try connecting to user	החיבור שולח פאקטת ACK על פאקטת האתחול, ומנסה להתחבר למשתמש הקצה המבוקש על ידי יצירת DataSocket
Connection successful, send connection approval	החיבור למשתמש הקצה המבוקש צלח ועצם החיבור שולח אישור חיבור לשרת המרוחק
Connection unsuccessful, send closing packet	החיבור למשתמש לא צלח, ועצם החיבור שולח פאקטת סגירה לשרת המרוחק
Socket error	שגיאה בסוקט ה-UDP של ה-RUDP Manager

שירות רשת - HTTP Service

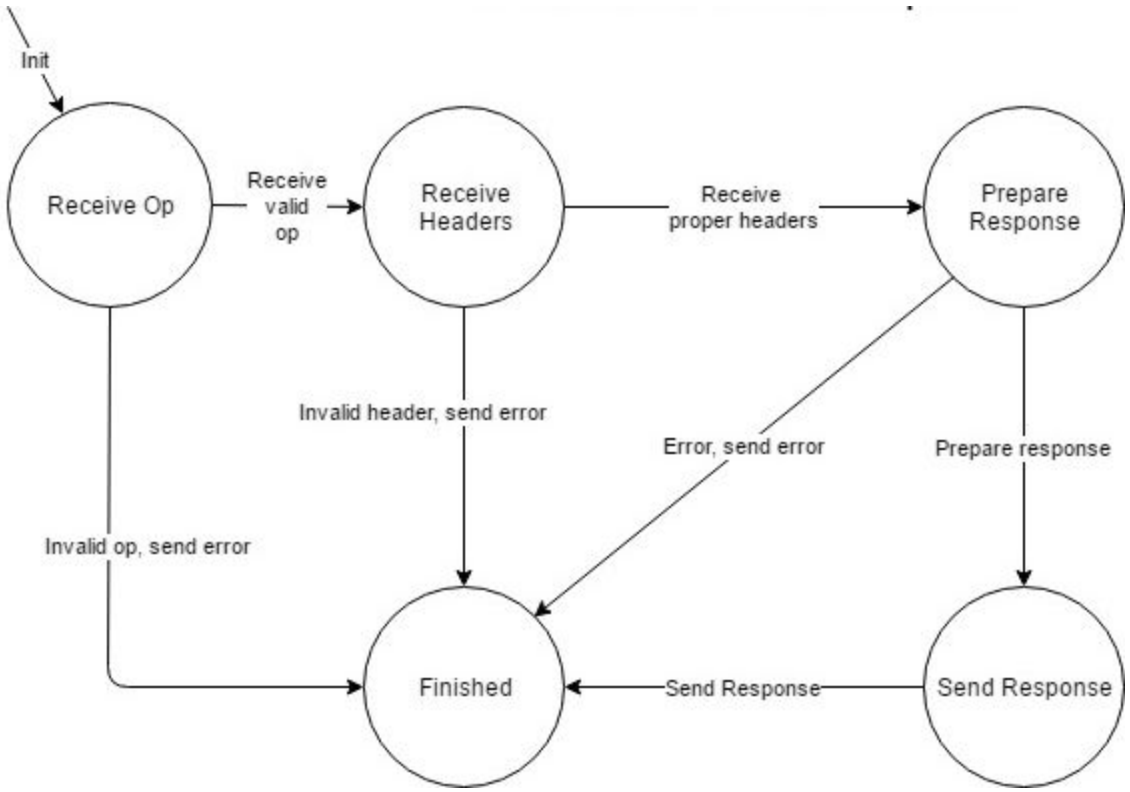
לוגיקה של טיפול בבקשות HTTP.



מצב	תיאור
Receive Headers	עצם השירות מקבל את ה-Headers של בקשת ה-HTTP מעצם ה-HTTPSocket.
Open	עצם השירות פותח את הקובץ המבוקש (אם יש).
Prepare Response	עצם השירות מכין את המענה.
Send Headers	עצם השירות מעביר לעצם ה-HTTPSocket את ה-Header-ים של המענה.
Send Content	עצם השירות מעביר לעצם ה-HTTPSocket את תוכן המענה.
Close	עצם השירות סוגר את הקובץ שפתח (אם יש).
Finished	עצם השירות סיים.
מעבר	**תיאור**
Receive proper headers	עצם השירות קיבל header-ים תקינים לפי פרוטוקול HTTP
Invalid header	עצם השירות קיבל header לא תקין לפי פרוטוקול HTTP
Open file (if exists)	אם השירות המבוקש הוא שירות קובץ, עצם השירות סיים לפתוח את הקובץ.
File Not Found - Send 404	הקובץ המבוקש לא נמצא, נשלחת שגיאה לדפדפן.
Prepare reponse	עצם השירות סיים להכין את המענה
Send headers	עצם השירות סיים לשלוח את ה-header-ים של המענה.
Send content	עצם השירות סיים לשלוח את תוכן המענה.
Internal error, send 500	עצם השירות נתקל בשגיאה במהלך ביצוע הלוגיקה ושולח הודעת שגיאה לדפדפן
Close file (if exists)	אם השירות המבוקש הוא שירות קובץ, עצם השירות סיים לסגור את הקובץ

בקשת בקרה - Control Request

לוגיקה של טיפול בבקשות בקרה.



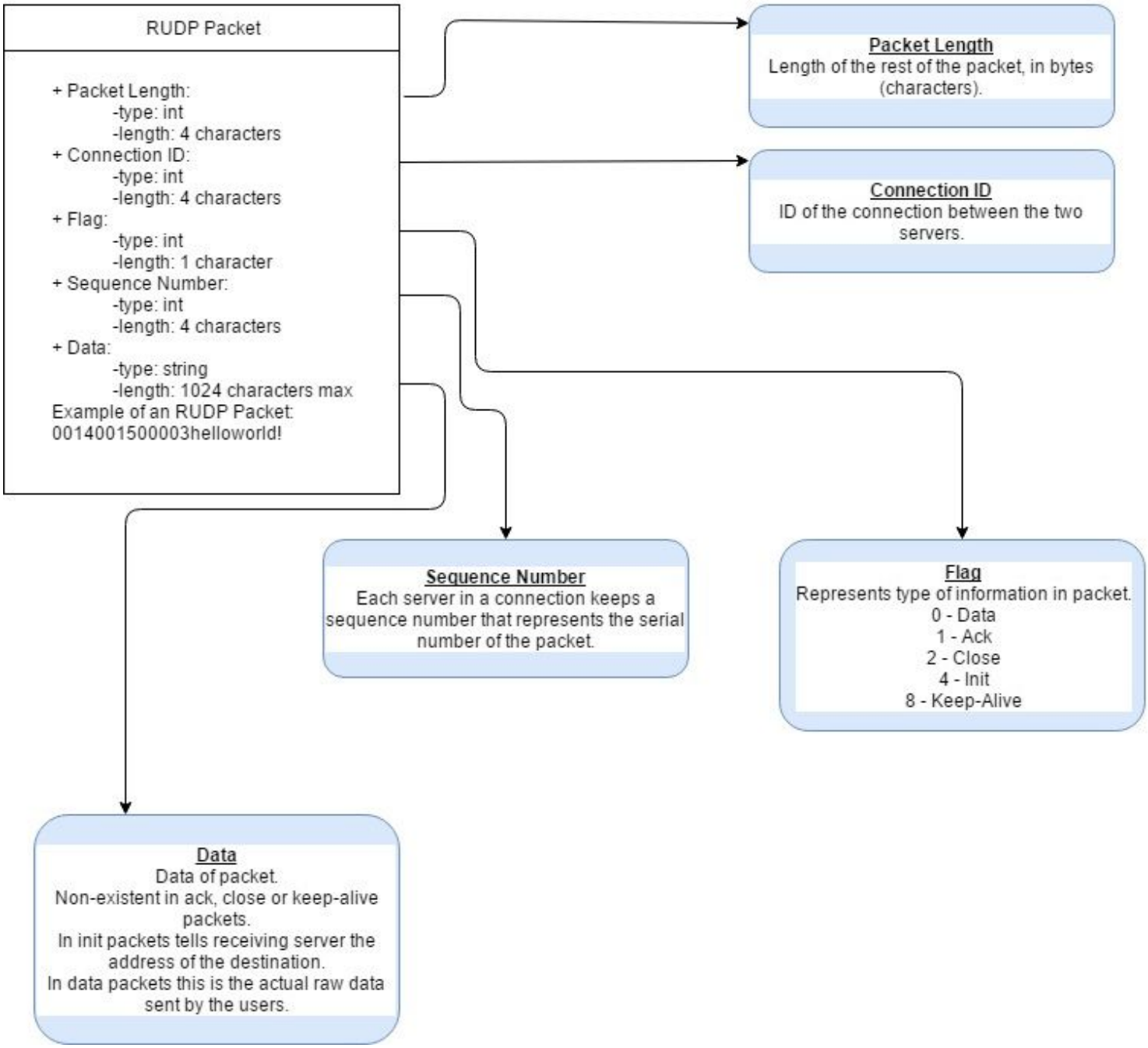
מצב	תיאור
Receive Op	קבלת סוג הבקשה (סטטיסטיקה או פתיחת חיבור).
Receive Headers	עצם הבקשה מקבל את ה-header-ים המתאימים לבקשה לפי פרוטוקול הבקרה.
Prepare Response	עצם הבקשה מכין מענה לבקשה.
Send Response	עצם הבקשה מעביר לעצם ה-ControlSocket את המענה לבקשה.
Finished	עצם הבקשה סיים.
מעבר	**תיאור**
Received valid OP	התקבל סוג בקשה תקין, נפתח עצם בקשת בקרה מסוג זה.
Invalid op, send error	התקבל סוג בקשה לא תקין, נשלחת שגיאה למשתמש

Receive proper headers	התקבלו header-ים תקינים ומתאימים לבקשה לפי פרוטוקול הבקרה.
Invalid hedaer, send error	התקבל header לא תקין, נשלחת שגיאה למשתמש.
Error, send error	התרחשה שגיאה בעת ביצוע הלוגיקה של הבקשה, נשלחת שגיאה למשתמש.
Prepare response	עצם הבקשה סיים להכין את הבקשה.
Send response	עצם הבקשה סיים לשלוח את הבקשה.

מבני נתונים בפרוטוקולים

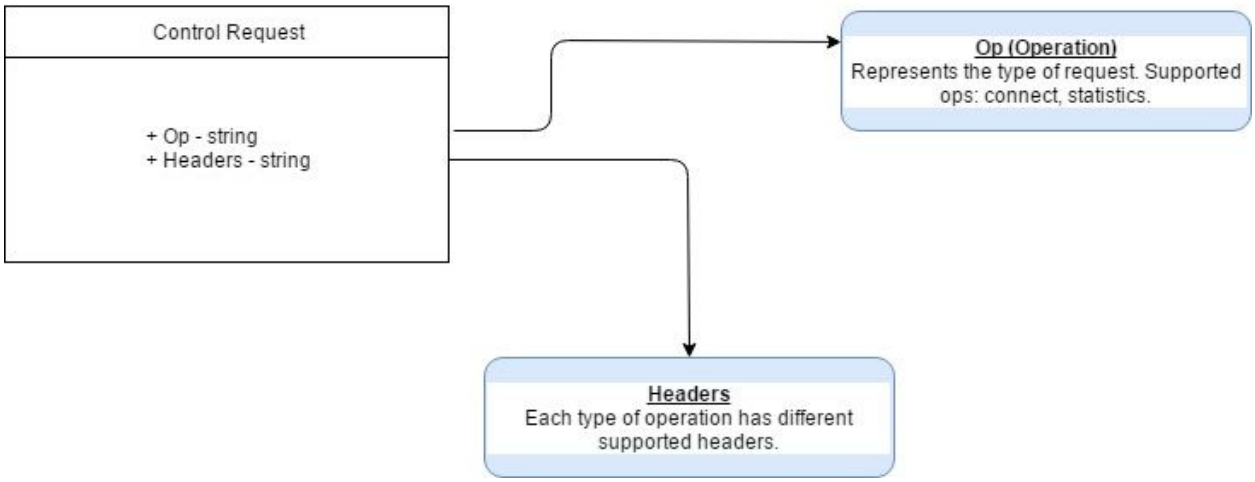
פאקטת RUDP

להלן דיאגרמה המתארת את מבנה פאקטת ה-RUDP, עם תיאורים של כל מרכיב בה.



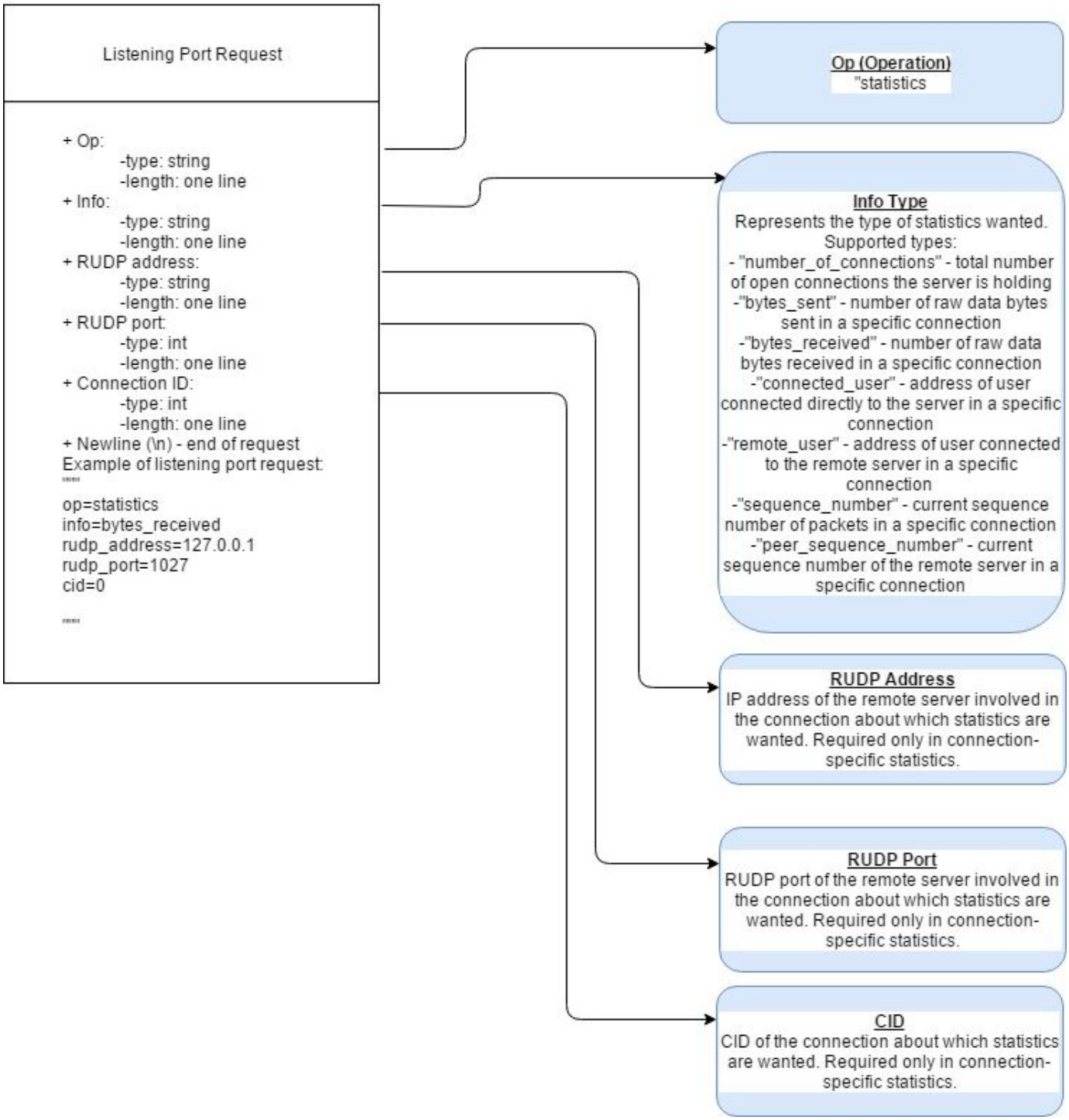
בקשת בקרה כללית

להלן דיאגרמה המתארת את מבנה הכללי של בקשת הבקרה, עם תיאורים של כל מרכיב בה.



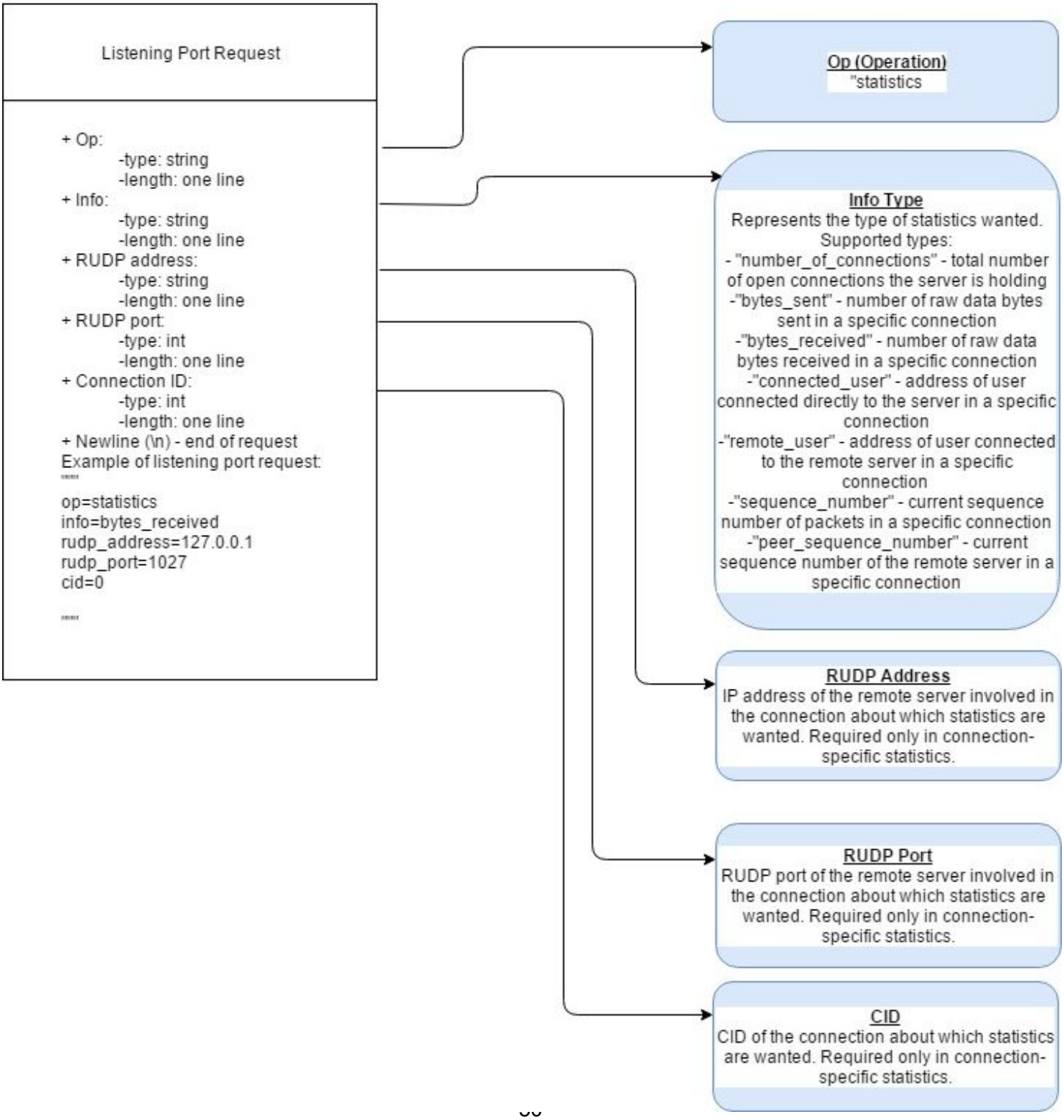
בקשת בקרה - סטטיסטיקה

להלן דיאגרמה המתארת את מבנה הכללי של בקשת הסטטיסטיקה, עם תיאורים של כל מרכיב בה.



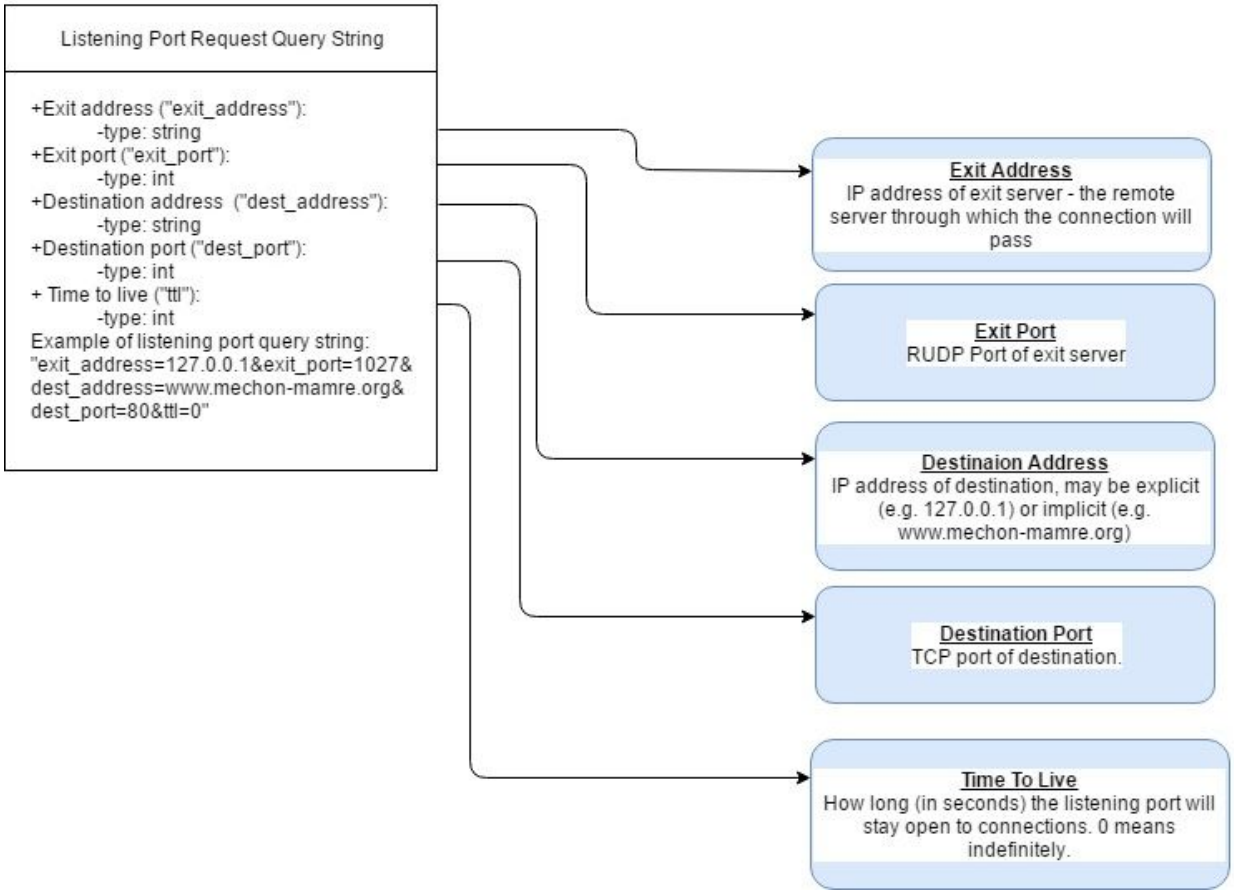
בקשת בקרה - פתיחת פורט (בקשת חיבור)

להלן דיאגרמה המתארת את מבנה הכללי של בקשת פתיחת פורט, עם תיאורים של כל מרכיב בה.



מבנה בקשת פורט ב-HTTP

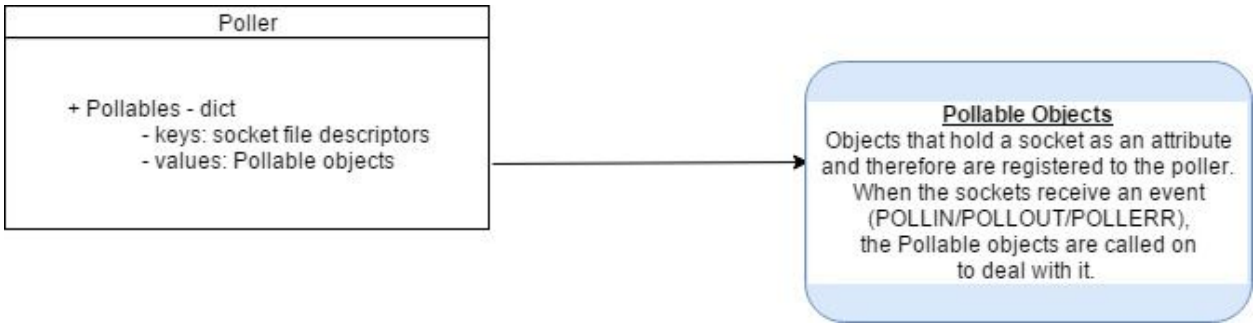
בקשת הפורט באתר האינטרנט מתבצעת באמצעות מילוי שאלון (Form), ששולח לשירות פתיחת הפורט ארגומנט בשם query string, שמבנהו מתואר להלן:



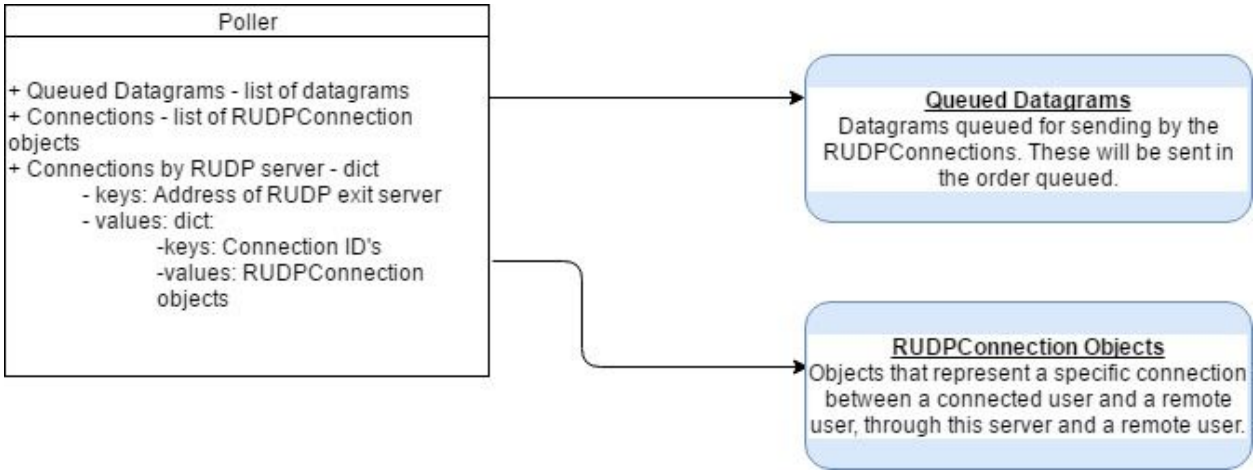
מבני נתונים בעצמים

מבני הנתונים שיוצגו בתת-פרק זה הינם מבני הנתונים המורכבים בלבד (מילונים ומערכים המכילים עצמים ומבנים מורכבים אחרים), ולא כל התכונות של העצמים בפרויקט.

מבני נתונים ב-Poller



מבני נתונים ב-RUDP Manager



אתגרים במימוש ופתרונות

האתגר העיקרי במימוש התוכנית היה בדיקתו, שכן על מנת להבטיח אמינות עלינו לוודא שהמערכת עובדת גם במקרי קצה שונים ומשונים. תהליך המחשבה על שיטת הבדיקות אינו פשוט והוא דורש היכרות מעמיקה עם המערכת. אתאר שני סוגים עיקריים של בדיקות שבוצעו במהלך הפרויקט:

בדיקה שגרתית

זוהי בדיקה של המערכת כאשר היא איננה נמצאת תחת עומס מיוחד. בדיקה זו נועדה לוודא שהמערכת פעולת כראוי, לפחות במקרים שאינם קיצוניים.

על מנת לבדוק את המערכת בדיקה שגרתית, מצאתי אתר אינטרנט המשרת את הדפדפן בפרוטוקול HTTP פשוט וללא redirect-ים, לדוגמה www.mechon-mamre.org. במקום לגשת ישירות לאתר עם הדפדפן, הכנסתי את המערכת שלי בין הדפדפן לשרת. למעשה, הקמתי חיבור בין משתמשי הקצה: הדפדפן והשרת, שעובר דרך שני שרתים שלי.

אם המערכת שלי מבצעת את תפקידה באופן תקני, הרי שהגלישה באתר דרך השרתים שלי תהיה זהה לגלישה ישירות מהדפדפן לשרת. אכן זו הייתה תוצאת הבדיקה.

בדיקה תחת עומס

על מנת לבדוק את המערכת שלי תחת עומס, כתבתי שתי תכניות בדיקה: Sender ו-Echoer. שילובן של שתי תכניות אלו בודק בצורה מיטבית את ההתנהלות של המערכת שלי במצב עמוס.

התכנית Sender מקבלת שני פרמטרים חשובים: מספר פורטי החיבור לפתוח - n, ומספר הפעמים להתחבר לכל פורט - m. התכנית מתחברת לשרת RUDP, ומבקשת דרך ערוץ הבקרה לפתוח n פורטי חיבור לכתובת של תכנית Echoer דרך שרת RUDP נוסף. לאחר מכן, היא מתחברת לכל פורט כזה m פעמים. בכך למעשה נוצרים m*n חיבורים בין שני שרתי ה-RUDP.

בכל חיבור כזה, שולחת התכנית Sender קובץ טקסטואלי אקראי בגודל 17Kb. התכנית Echoer שולחת בחזרה לשרתים כל מידע שהיא מקבלת, ומידע זה מגיע חזרה ל-Sender. תכנית Sender משווה את תוצאות ה"הדהוד" בכל חיבור למידע קובץ ששלחה, ומעלה שגיאה אם יש הבדל ביניהם.

העומס על השרתים שמעמיסות תכניות הבדיקה מתבטא בכמה אופנים: מספר פורטים מאזינים גדול, מספר חיבורים גדול לכל פורט מאזין, ומידע רב שעובר בשני כיווני החיבור.

לכן, אם תכנית ה-Sender אינה מעלה שגיאה, נוכל לדעת שהשרתים עומדים תחת עומס גדול ואינם מפרים את אמינות העברת המידע, כלומר - שהמערכת עובדת כפי שדרוש.

בעיות ידועות ומגבלות

סביבת ריצה

השרת רץ בסביבת Linux, או בסביבת cygwin ב-Windows. ניתן להוסיף תמיכה ב-Windows, אך הוספת תמיכה ב-Windows תדרוש התאמת קריאות מערכת מסוימות לפעולתן ב-Windows.

תכנית הבדיקה - מספר חיבורים

תכנית הבדיקה מוגבלת במספר החיבורים שהיא מסוגלת לבדוק. כאשר מריצים את הבדיקה עם יותר מ-150 חיבורים (נניח - 15 פורטי חיבור ו-15 חיבורים בכל פורט חיבור), שרת ה-RUDP המתחבר לתכנית ה-Echoer מקבל בחלק מסוקטי המידע שלו אירועי שגיאה בעלי מספר 111, שמשמעותם Connection Refused, כלומר - החיבור סורב. זאת כנראה משום שתכנית ה-Echoer אינה מסוגלת לעמוד בעומס החיבורים, ואינה מספיקה להגיב לבקשת החיבור.

שרת - מספר חיבורים

השרת גם הוא מוגבל מבחינת כמות החיבורים שהוא מסוגל לשאת, וזאת בשל שתי מגבלות מובנות:

1. כאשר השרת רץ בסביבת cygwin, הוא מוגבל מבחינת מספר ה-file descriptors שמותר לו לפתוח. הגבול העליון שניתן לקנפג לכך בסביבת cygwin למספר הוא 3,200, ולכן גם הגבול העליון של מספר החיבורים שמסוגל לתחזק השרת יהיה בסביבות מספר זה.
2. קריאות המערכת select ו-poll, שתי הקריאות שמסוגלות לנהל מערכת סוקטים אסינכרונית, מוגבלות מבחינת מספר הסוקטים שהן יכולות לנהל. גבולות אלה תלויים במידה רבה במערכת עליה הן רצות. כאשר גבול זה נפרץ, השגיאות המתקבלות הן:

```
ValueError: filedescriptor out of range in select()
```

עבור select, ו-

```
error: (14, 'Bad address')
```

עבור poll.

התקנה ותפעול

התקנה

1. הורידו cygwin מהקישור הבא בהתאם לסוג המערכת (32-ביט או 64-ביט):
https://cygwin.com/install.html
2. אם לא מותקן אצלכם דפדפן מודרני (Google Chrome, Microsoft Edge, Mozilla Firefox) הורידו גוגל כרום: https://www.google.com/chrome/browser/desktop/index.html
3. בעת התקנת cygwin, בחרו להתקין גם את המרכיבים הבאים: python 2.7, git, netcat (nc).
4. פתחו חלון cygwin, והריצו בו את הפקודות הבאות:

```
cd /tmp
git clone https://github.com/EitanGronich/Reliable-UDP
```

כעת הפרויקט שלי מותקן אצלכם בתיקיית cygwin64/tmp.

תפעול

הרצת שני שרתים

1. פתחו שני חלונות של cygwin מחלון החיפוש בווינדוס.
2. בשני ה-terminal-ים הריצו את הפקודה:

```
cd /tmp/Reliable-UDP
```

3. הריצו בterminal אחד את השורה הבאה:

```
python -m Reliable-UDP.Server --rudp-port 1026 --control-port 1025 --http-port 8080
```

זהו השרת הראשון, נקרא לו שרת א'. סוקט ה-UDP שלו נמצא על פורט 1026, הסוקט המאזין לחיבורי בקרה מאזין לפורט 1025, והסוקט המאזין לחיבורי HTTP מאזין לסוקט 8080.

4. הריצו ב-terminal השני את השורה הבאה:

```
python -m Reliable-UDP.Server --rudp-port 1027 --control-port 1028 --http-port 8081
```

זהו השרת השני, נקרא לו שרת ב'. סוקט ה-UDP שלו נמצא על פורט 1027, הסוקט המאזין לחיבורי בקרה מאזין לפורט 1028, והסוקט המאזין לחיבורי HTTP מאזין לפורט 8081.

התחברות לעמוד האינטרנט של השרתים

1. פתחו את הדפדפן.

2. בלשונית אחת, גלשו לכתובת:

http://localhost:8080/home.html

זהו דף האינטרנט המציג מידע על שרת א'.

3. בלשונית שנייה, גלשו לכתובת:

http://localhost:8081/home.html

זהו דף האינטרנט המציג מידע על שרת ב'.

4. בכל שלב בהמשך הדגמת הפרויקט תוכלו לחזור לדפים אלו ולהתרשם מהמידע על החיבורים, תחת לשונית 'Connection Data'.

הדגמת המערכת באמצעות שתי תכניות netcat

בהדגמה זו תוכלו לראות את פעולת המערכת ב-proxying פשוט של חיבור בין שתי תכניות netcat, שישמשו בתור משתמשי הקצה של החיבור. netcat הוא כלי פשוט לתפעול המאפשר פעולות פשוטות ברשת, כגון קבלת חיבורים והתחברות.

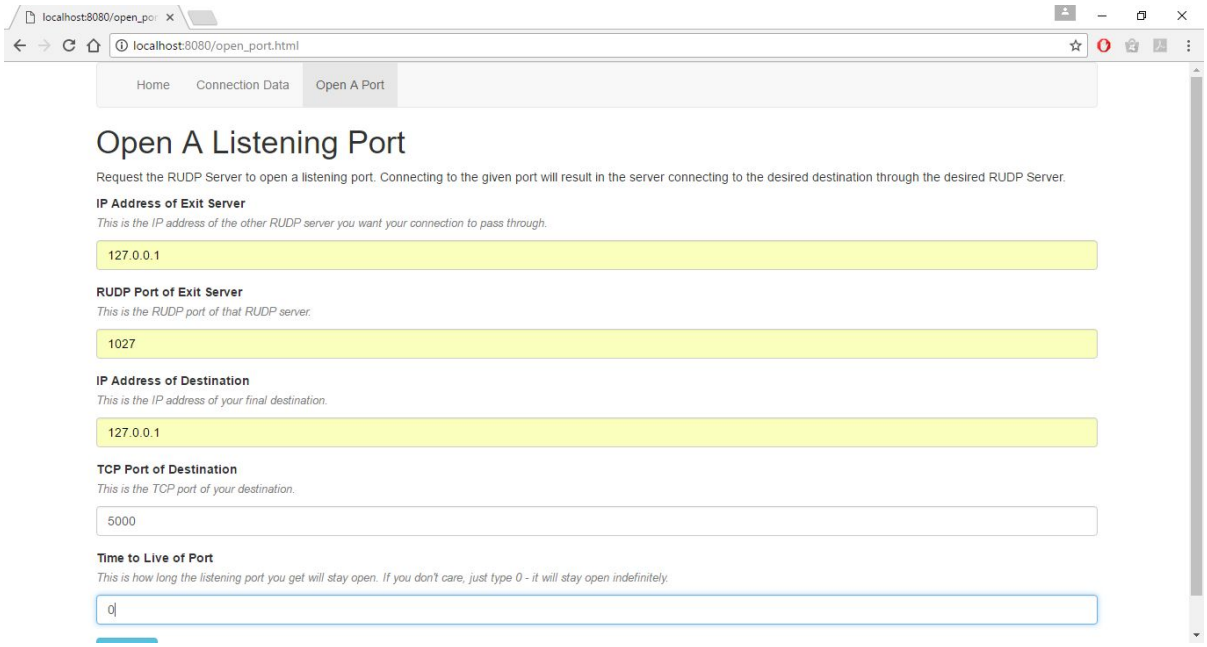
1. הריצו שני חלונות cygwin נוספים.
2. בחלון cygwin אחד, הריצו את הפקודה:

```
nc -l -p 5000
```

זוהי התכנית שמקבלת את החיבור - משתמש ב'. היא מקשיבה לפורט 5000.

4. כעת עליכם לפתוח אצל שרת א' פורט לחיבור למשתמש ב', דרך שרת ב'. תוכלו לעשות זאת בשתי דרכים:

א. אפשרות ראשונה: בלשונית בדפדפן שבה הכתובת localhost:8080/home.html, גלשו ל"Open A Port". שם מלאו את השדות בצורה הבאה:



לחצו על submit. מספר הפורט שמופיע על המסך באדום הוא מספר הפורט אליו תתחברו בהמשך. נקרא לו X.

ב. אפשרות שנייה: בחלון ה-cygwin השני, הריצו את השורה הבאה על מנת להתחבר לפורט הבקרה של שרת א':

```
nc localhost 1025
```

לאחר מכן כתבו בחלון ("\n" מייצג לחיצה על המקש אנטר):

```
op=connect\n
exit_address=127.0.0.1\n
exit_port=1027\n
dest_address=127.0.0.1\n
dest_port=5000\n
ttl=0\n
n\
```

קיבלתם הודעה מהצורה:

```
op=connect
code=0
port=X
```

X הוא מספר הפורט שאליו תתחברו.

כעת תוכלו להתנתק מחיבור הבקרה בעזרת לחיצה על ctrl-C, וחלון ה-cygwin יתפנה לכם לשלב הבא של התפעול.

5. הריצו בחלון ה-cygwin השני (X הוא הפורט שקיבלתם בסעיף הקודם):

```
nc localhost X
```

6. תוכלו לכתוב בכל אחד מחלונות הcygwin המריצים netcat וללחוץ על מקש האנטר, ולתקשר עם חלון ה-netcat השני בתקשורת דמויית צ'אט. שימו לב שכל מידע שנשלח מחלון ntecat עובר דרך שני השרתים בחלונות ה-cygwin בדרכו לחלון ה-netcat השני.

הדגמת המערכת באמצעות תכניות הבדיקה

1. פתחו שני חלונות cygwin נוספים.
2. בכל חלון cygwin הריצו את הפקודה:

```
cd /tmp/Reliable-UDP
```

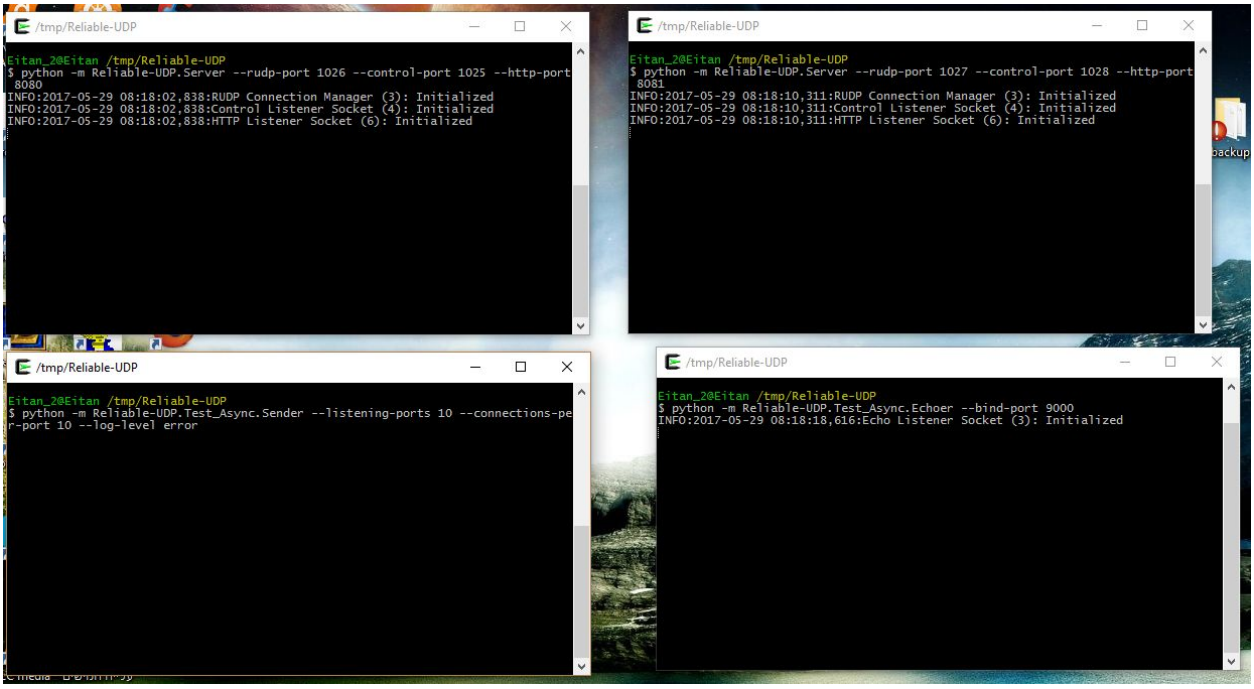
3. בחלון אחד הריצו את תכנית ה-Echoer, שתקשיב בפורט 9000:

```
python -m Reliable-UDP.Test_Async.Echoer --bind-port 9000
```

4. בחלון אחר הריצו את תכנית ה-Sender, שתפתח 100 חיבורים לתכנית ה-Echoer דרך שני השרתים, ותשלח בכל אחד מהם קובץ בגודל 17Kb:

```
\ python -m Reliable-UDP.Test_Async.Sender --listening-ports 10
                                    --connections-per-port 10
```

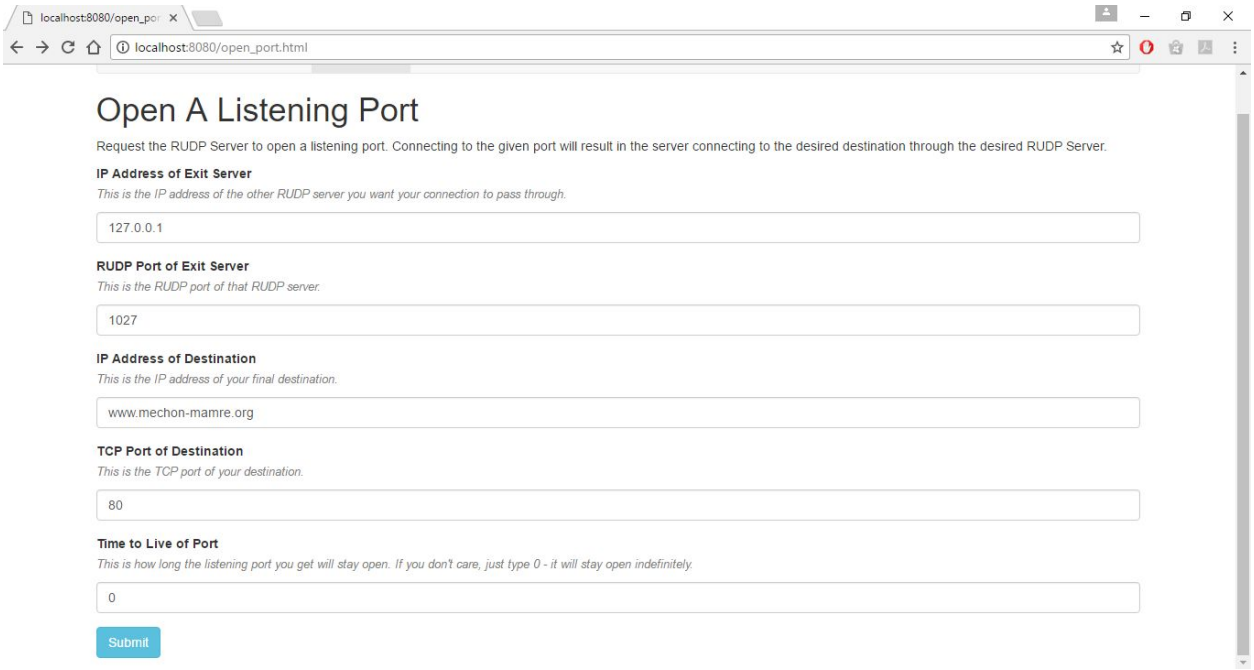
5. לפני הרצת ה-Sender, המערכת אמורה להיראות כך:



6. לאחר ההרצה, התרשמו משליחת המידע בין משתמשי הקצה והשרתים. חכו עד שתוכנית ה-Sender תסיים ותצא. הבדיקה הסתיימה.

הדגמת המערכת באמצעות proxying בין הדפדפן לאתר אינטרנט

1. בחרו אתר אינטרנט המשתמש בפרוטוקול HTTP פשוט, ללא redirects. אתר מומלץ שכזה הוא אתר www.mechon-mamre.org, אתר תנ"ך מנוקד.
2. בלשונית בדפדפן שבה הכתובת localhost:8080/home.html, גלשו ל"Open A Port". שם מלאו את השדות בצורה הבאה:



3. לחצו על submit. מספר הפורט שמופיע על המסך באדום הוא מספר הפורט אליו תתחברו בהמשך. נקרא לו X.
4. פתחו לשונית נוספת בדפדפן וגלשו לכתובת: localhost:8080/X
5. גלשו בחופשיות באתר. שימו לב שכל מידע שאתם מקבלים מועבר דרך שני השרתים בפרוטוקול RUDP.

תכניות לעתיד

פרויקט זה יכול להמשיך ולהתפתח בכיוונים אפשריים רבים.

מימוש Reliable Multicast

מימוש שידור פאקטות ב-multicast למספר רב של שרתי RUDP אחרים. על מנת להבטיח אמינות, השרת ששלח פאקטת multicast ישמור פאקטה זו עד שכל הכתובות להן שלח ישלחו לו ACK עליה. בנוסף, כדי להגביר את היעילות, יוכלו נמעני ה-multicast לבקש שידור מחדש של חלק מהמידע, בעזרת unicast שידור לשולח.

Socket API

מימוש API ל-"סוקט RUDP" בשפת פייתון שיפעל בדומה ל-API של סוקט TCP רגיל. API זה יקל על השימוש במערכת ה-RUDP, כיוון שלא ידרוש פעולה נוספת של בקשת פורט. מימוש ה-API ייעשה באמצעות שימוש ב-Unix Domain Socket.

שיפור הפרוטוקול

כפי שצוין בפרק התיאורטי, ניתן ליישם שיפורים לאלגוריתם של הפרויקט כדי לשפר את יעילותו. בין השאר, אפשרויות לשיפורים הן:

1. חישוב מותאם של ה-RTO של המערכת, על פי זמני RTT המתקבלים משידורים מעשיים ברשת.
2. פרוטוקול שיחה מהיר יותר מ"פינג-פונג", המאפשר לשלוח מספר פאקטות לפני קבלת ACK-ים עליהן.
3. selective acknowledgement, המגביר את יעילות הפרוטוקול לאחר יישום פרוטוקול שיחה מהיר.

פרק אישי

מימוש פרויקט זה היה עבורי תהליך משמעותי, מלמד ומהנה.

למעשה, זוהי הפעם הראשונה שבה אני מממש פרויקט בסדר גודל כזה, החל משלב הרעיון, דרך שלב האפיון והתכנון, עד שלב המימוש ולבסוף שלב כתיבת התיק. חוויה ראשונה ומיוחדת זו הציבה בפניי קשיים ונתנה לי תובנות במספר מישורים.

ראשית, במישור התיאורטי נחשפתי לעולם תוכן ספציפי ומורכב בתוך תחום מדעי המחשב. לימוד הנושא והמושגים הקשורים בו לא היה קל כלל, במיוחד עקב העובדה שרוב הלמידה הייתה עצמאית. למרות זאת, הנושא עניין אותי, ולכן הצלחתי להתגבר על הקושי ולהעמיק בנושא עד שהגעתי להבנה מספקת בו.

שנית, במישור המימוש, כתבתי בעצם לראשונה פרויקט בעל עשרות קבצים, עשרות מחלקות בעלת אינטראקציות מורכבות ביניהם, עצמים היורשים זה מזה, קובץ לוג מסודר, ועוד. מפאת חוסר הניסיון שלי במערכות בסדר הגודל הזה, נתקלתי בקשיים הקשורים למימוש נכון של מחלקות ותכניות, בניית מערכות של עצמים, לוגינג, ואפילו הרצת חבילות ומודולים. גם על קשיים אלה הייתי חייב להתגבר, תוך כדי שיפור מיומנויות תכנות ותכנון מבוסס-עצמים.

בנוסף, פרויקט זה היה הפעם הראשונה בה עבדתי עם מערכת ניהול הגרסאות git. הבנת עקרונות הפעולה של git ושימוש מושכל בהם לקחו לי זמן לא מבוטל, וגם כיום אינני מבין לגמרי את כל האפשרויות שמערכת זו מציעה, אך למדתי איך להשתמש בכלי באופן בסיסי על מנת לשמור גרסאות קוד, לחזור לגרסאות קודמות ולסנכרן בין שני repositories שונים. גם עצם למידת הכלי החדש הקנתה לי מיומנויות חשיבה ולמידה עצמאית.

הפרויקט תרם לי גם בנושא שאינו קשור ישירות למדעי המחשב, והוא פיתוח מיומנויות עבודה ותכנון. במהלך הפרויקט השתפרתי בלתכנן זמנים ולעבוד באופן יעיל ומרוכז. אין לי ספק שמיומנות זו חשובה ביותר להמשך דרכי בכל תחומי החיים, ולכן אני שמח שנפל בחלקי לבצע פרויקט ששיפר אותי בתחום זה.

לסיכום, הפרויקט היה בעיניי חוויה חיובית, הן בפן הלימודי והן בפן האישיותי, אך לא נטולת קשיים קוגניטיביים ומנטליים. אני מצפה לבצע עוד פרויקטים כאלו בעתיד.

בהזדמנות זו אני רוצה להודות למורים שרית לולב ואלון בר-לב, שליוו אותי לכל אורך הפרויקט ונתנו תמיכה אדירה בכל תחום!

קוד פרויקט

הקוד לפרויקט נמצא ב-release 1.0.0 של הפרויקט ב-github:
https://github.com/EitanGronich/Reliable-UDP/releases

קישור לדוקמנטציה

בדוקמנטציה ניתן לצפות בקובץ index.html שבתיקיית html בקובץ ה-zip בשם gen-docs המצורף לrelease.
gen docs/html/index.html