

מגישים:
ארד בן מנשה 207083353
עפרי לייפר 206391054

נושא הפרויקט: HTTP Redirect Server App Using Custom RUDP

תוכן עניינים:

רשתות תקשורת - פרויקט גמר

1. עמוד 01 - תוכן עניינים
2. עמוד 02 - תיאור המערכת (כולל את הפעלת התוכנה ותרשים זרימה)
3. עמוד 04 - תשובות לשאלות בפרויקט
4. עמוד 06 - שרת-h-DHCP ושרת-h-DNS
5. עמוד 07 - שרת האפליקציה והשרות DATA
6. עמוד 09 - פרוטוקול-h-RUDP
7. עמוד 13 - מימוש של - Reliability, Congestion Control, Flow Control ב RUDP
8. עמוד 18 - הרצאות קוד עם ווירשארק של שרת DNS ושרת DHCP
9. עמוד 22 - הרצאות קוד עם ווירשארק של (1)
10. עמוד 25 - הרצאות קוד עם ווירשארק של (2)
11. עמוד 38 -ביבליוגרפיה

מצורפים לקובץ PDF:

- תיקיה עם 9 הקלטות ווירשארק
- תיקיה עם סרטוני הריצה, שמסבירים איך להריץ את הפרויקט במצבים שונים.
- קבצי הריצה של שרת DNS ושרת DHCP (모ולים בכל שאר התיקיות הריצה של הפרויקט).
- (1)תיקיה המכילה קבצי הריצה של השרת והלקוח המבוססים על פרוטוקול TCP, בהם שרת האפליקציה מפנה לשרתים מוקומיים
- (2)תיקיה המכילה קבצי הריצה של השרת והלקוח המבוססים על פרוטוקול RUDP, בהם שרת האפליקציה מפנה לשרתים מוקומיים (כדי שהעברת המידע תוכל להתבצע עם פרוטוקול RUDP שהכו - שרתים באינטראנט כموן לא יודעים לעבוד על ה프וטוקול שמיישנו) + 5 קבצי טסטים(בכל טסט יש הסבר ליעודו)
- תיקיה המכילה קבצי הריצה של השרת והלקוח המבוססים על פרוטוקול TCP, בהם שרת האפליקציה מפנה לשרתים חיצוניים באינטראנט

קישור GIT של הפרויקט: <https://github.com/leifer98/NetworkProject.git>

חלק ראשון - תיאור המערכת

על המערכת -

המערכת מכילה שישה שירותים - DHCP,DNS,APPLICATION SERVER ו-3 שירותי מידע. לאחר הריצת השירותים כל לקוח שמחבר לרשת, יכול לבצע את הפעולות הבאות:

- לקבל כתובת IP מה-DHCP SERVER
 - לקבל כתובת של ה-APP SERVER מה-APP SERVER
 - לפנות ל-APPLICATION SERVER ולבקש תמונה של חתול
- שירות האפליקציה מפנה את הלקוח לאחד משלשות השירותים המקומיים כscal אחד מהם מכיל תמונה אחרת של חתול, ומוריד את התמונה.

ניתן לבצע את התקשרות במערכת בין הלקוח והשירות מידע והאפליקציה מינה על בסיס פרוטוקול TCP או על ידי פרוטוקול RUDP שנאנו יצרנו. ה-RUDP הוא פרוטוקול אשר מבוסס על UDP עם פיצ'רים חדשים עליהם נרחב בהמשך.

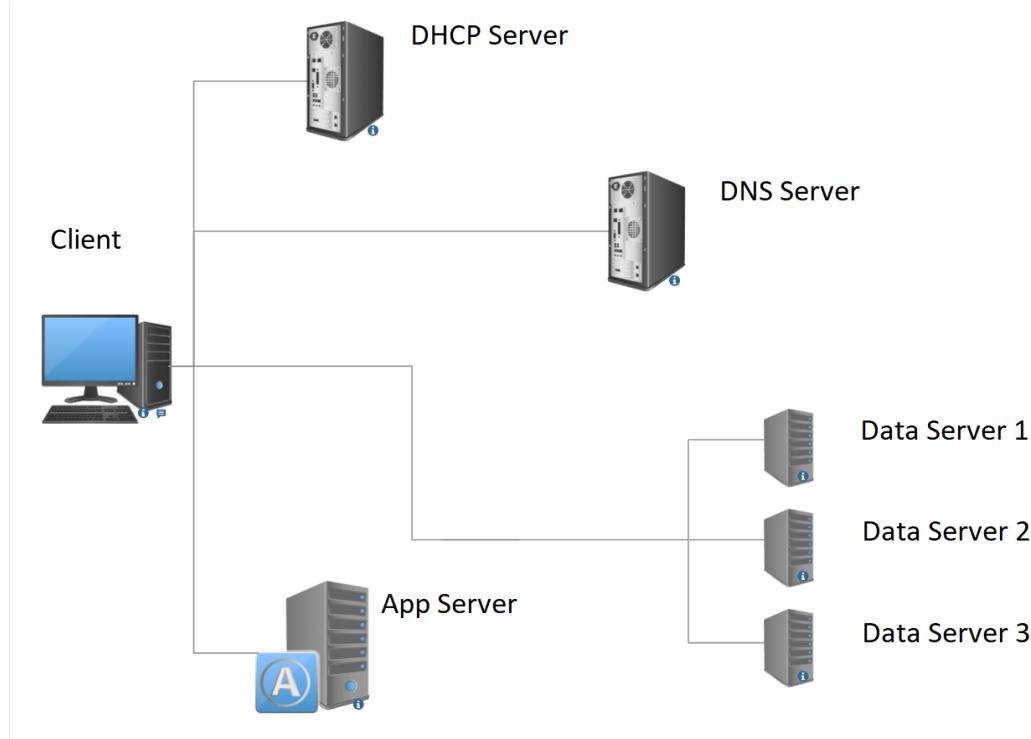
הפעלת המערכת -

- יש לעשות CLONE GIT לפרויקט בקישור הבא:
<https://github.com/EladKatz/Project-1>
- בפרויקט יש 3 תיקיות המכילות וריאציה של המערכת. בכל תיקייה יש את כל הקבצים הרלוונטיים כדי להריץ את המערכת לפי בהתאם לנדרש, והוריאציות האפשרות הם:
 1. מערכת מבוססת פרוטוקול RUDP עם האפליקציה | TCP עם שירותי מקומיים
 2. מערכת מבוססת פרוטוקול TCP עם שירותי מקומיים
 3. מערכת מבוססת פרוטוקול TCP עם שירותי באינטרנט
- יש לבחור את התיקייה של המערכת, לפתח טרמינל במקומם של התיקייה ולהריץ את הקבצים הבאים (למשל לDNS.py יש להריץ python3 DNS.py):
 1. DHCP.py
 2. DNS.py
 3. app_server.py
 4. multi_server.py
 - במידה ופותחים את המערכת שפונה לשירותים באינטרנט אין צורך להריץ את .multi_server
 - במידה ופותחים מערכת שלא פונה לשירותים באינטרנט, מומלץ לכבות את האינטרנט לפני הריצה.

תלויות -

- Python 3.10
- Python libraries: Scapy, Pillow, BytesIO
- Windows 10(sniffing interfaces can be affected by switching operating system)

תרשים זרימה של הרשת (עם שירותי מקומיים) -



סדר פעולות ברשת הנ"ל

- נניח כי הלוקו מתחבר לרשת פיזית. ומטרתו היא לקבל תמונה של חתול בסוף הריצה כאשר כל השירותים רצים.
1. הלוקו מתחבר ופיזית ושולח בקשה DHCP DISCOVER ברשת (עשה FLOOD).
 2. השירות DHCP מעבד את הבקשה, מכין DHCP OFFER לכתובת MAC שפנה אליו, ו"מציף" את הרשת עם הפקטה.
 3. הלוקו מעבד את הבקשת OFFFER ושולח בקשה DHCP REQUEST לשרת DNS SERVER.
 4. השירות DHCP שולח בחרה ACK עם כתובת DNS SERVER לлокו.
 5. כעת, לлокו יש כתובת IP שה-ACK הקenza לו, ואת הכתובת של השירות DNS.
 6. הלוקו מכין בקשה DNS REQUEST לשרת DNS בבקשת לגשת לאתר "the-famous-cat.com".
 7. השירות DNS מעבד את הבקשה מכין DNS RESPONSE ושולח את הכתובת של האתר לлокו.
 8. הלוקו מכין ושולח בקשה HTTP GET לשרת האפליקציה לבקש תמונה של חתול.
 9. השירות מעבד את הבקשה ומחזיר לлокו HTTP REDIRECT, לאחד מהסרים ברשימה שלו המכילים תמונה של החתול.
 10. הלוקו פונה לשרת שקיבל משרת האפליקציה וմבקש את התמונה עם בקשה HTTP GET.
 11. השירות עם התמונה מעבד את הבקשה ושולח תמונה של החתול בחזרה לлокו.

חלק שני - תשובות לשאלות בפרויקט

שאלה 1

הבדלים עיקריים בין פרוטוקול TCP ל-QUIC:

1. פרוטוקול QUIC לא צריך לבצע לחיצת יד משולשת בכל חיבור אלא רק בראשונה לעומת פרוטוקול TCP שעושה בכל תקשורת.
2. ה프וטוקול TCP עובד דרך שכבת התעבורה (TRANSPORT) לעומת הפרוטוקול QUIC שעבוד דרך שכבת התעבורה וגם דרך שכבת האפליקציה.
3. כאשר לquoח מתחילה חיבור במכשיר כלשהו וממשיך אותה במכשיר אחר, ב-QUIC לא צריך לסגור ואז לפתח חיבור חדש אלא פשוט להשתמש בPACKET NUMBER הייחודי ולהמשיך את התקשרות. לעומת TCP צריך לסגור ולפתח מחדש את התקשרות אם משנים את הרשות שימושים בה.
4. ב-TCP נקבל head of line blocking כאשר פקטה אחת מתעכבות וגורמת לעיכוב של שאר הפקודות האחריה. לעומת UDP אין חשיבות לסדר שהפקודות נשלחות, ואם נאבדת פקטה בדרך אז החיבור אינו נפסק ורק הפקטה הספציפית תשלוח בתור פקטה חדשה לגמרי בלי לעזר את שליחת הפקודות הבאות.

שאלה 2

הבדלים עיקריים בין Cubic ל-Vegas:

1. האלגוריתם Vegas מודד את ה RTT של פקודות כדי לקבוע את העומס ברשות. לעומת Cubic, משנה את גודל החלון של הפקודות שנשלחות בהתאם לתיקינות התנועה של החבילות ברשות.
2. האלגוריתם Vegas הוא משנה את הקצב של שליחת פקודות, לכל מי שמחובר אליו בקרה אחרת, לעומת Cubic תקשורת חלון השילחה הוא יותר אישי ויכול להשתנות בקרה פרטנית.

שאלה 3

פרוטוקול (Border Gateway Protocol) BGP הוא פרוטוקול ניטוב, כלומר לניבב המיצים את הਪרוטוקול יש טבלה של הרשותות שמחוברות אליו ואת הקשרים ביניהן לבין רשותות אחרות ומבצע החלטות ניטוב על בסיס הקשרים בין הרשותות ומידניות המוכתבת בקרה ידנית על ידי מנהל הרשות. הפרוטוקול מיושם על ידי נתבי BGP שנמצאים באינטראנט ומיכליים טבלאות של AS שנitin להגיאן אליום דרכם (Autonomous System = AS כמו למשל: ספק האינטראנט בזק, HOT וכו'...). אוסף כי הוא נחשה כ"דבק" שמחבר את האינטראנט כיוון שהמשתמש מבקש להגיאן כתובות שלא נמצאת ברשות המקומית חייב לעשות בעזרתו הפרויקט הנ"ל.

הפרוטוקול (Open Shortest Path First) OSPF הוא פרוטוקול intra-AS והפרוטוקול BGP הוא inter-AS. כאמור, פרוטוקול OSPF פועל בתחום מערכות אוטונומיות (AS) לעומת BGP שפועל בין מערכות אוטונומיות (AS).

הפרוטוקול לא עובד על מסלולים היכי קצרים מנוקודה לנוקודה אלא הוא עובד על פי המדיניות של ספק האינטראנט בדרך ליעד. יכול להיות שיש ספקי אינטראנט שלא רוצים שייעברו דרכם או שלא ניתן לעברו דרכם, והפרוטוקול רק מփש דרך להגיאן לעד על פי הדרכים שכל טבלת BGP מכילה ואני בהכרח שזו תהיה הדרך היכי קצרה.

שאלה 4

פורט הלקוח לפי הנדרש: 20054
 פорт השירות לפי הנדרש: 30353

<u>Application</u>	<u>Port src</u>	<u>Port Des</u>	<u>IP Src</u>	<u>IP Des</u>	<u>Mac Src</u>	<u>Mac Des</u>
app_server.py	30353	20054	Assigned by DHCP server	127.0.0.1	-	client random generated MAC

<u>Application</u>	<u>Port src</u>	<u>Port Des</u>	<u>IP Src</u>	<u>IP Des</u>	<u>Mac Src</u>	<u>Mac Des</u>
DHCP.py	67	68	Assigned by DHCP server	192.168.1.100	00:00:00:00:00:01	client random generated MAC

<u>Application</u>	<u>Port src</u>	<u>Port Des</u>	<u>IP Src</u>	<u>IP Des</u>	<u>Mac Src</u>	<u>Mac Des</u>
DNS.py	53	20054	Assigned by DHCP server	192.168.1.200	00:00:00:00:00:02	client random generated MAC

כיוון שכל השירותים עובדים על הרשת הפנימית, לא יהיו שינויים בהודעות אם יהיה NAT בין הלקוק לשירותים כיוון שהוא מוזר למפות בין כתובות חיצונית לרשת לבין כתובות ברשת, ככלומר מחשב שנמצא ברשת הפנימית יקבל לו מסויים אבל כשהוא ירצה לגשת לcom.google למשל אז הכתובת שתיהיה מוצגת, תהיה של הקוטר שמייצג את כל הראות. אם נשתמש ב프וטוקול QUIC אז המידע ישב מעל פרוטוקול udp, لكن ההודעות יראו כהודעות פרוטוקול UDP.

אם היינו פונים לשרת אינטרנט חיצוני אז ה-NAT כן יהיה משפייע על כל הכתובות הנ"ל.

שאלה 5

הבדלים בין פרוטוקול ARP ל-DNS:

1. הפרוטוקול ARP פועל בשכבה הפיזית לעומת DNS שפועל בשכבה של האפליקציה.
2. הפרוטוקול ARP נדרש למפות כתובות פיזיות (MAC) של מחשבים ברשת מקומית לכתובות IP, לעומת הפהוטוקול DNS שmaps שמות דומיין לכתובות IP.
3. הפרוטוקול ARP פועל בכל רכיב אינטרנט על הרשת שהוא מחובר אליה ושומר את הכתובת של כל אחד ברשת המקומית שלו עם השם של הכתובת הפיזית, לעומת DNS שmaps שמות של אתרים לכתובות והוא נמצא ברשת המקומית הרגילה, אלא הוא משוייר לספק אינטרנט לרוב.

חלק שלישי - שרת DHCP ושרת DNS

שרת DHCP

בפרויקט שלנו אנחנו בונים שרת DHCP שיודיע לקבל שני סוגי בקשות: DHCP DISCOVER וDHCP REQUEST וידע לטפל בשניהם.

במקרה של DHCP DISCOVER השרת בודק תחילה אם הכתובת MAC שימושית לлокוח כבר קיימת במילון של השירות. אם לא, השירות מציע חזרה לлокוח שפנה אליו DHCP OFFER עם IP שלא בשימושו כרגע על ידי שום MAC אחר שכבר מחובר.

במקרה של DHCP REQUEST השירות מחזיר DHCP ACK לлокוח ומאשר לו סופית שזו הכתובת שלו.

ביחד עם הפקות הנ"ל שהשרות מחזיר, כתובות השירות DNS גם מוחזרת כחלק מה OPTIONS בחייבת DHCP וכן הלה את הכתובת של השירות DNS המקומי בראשה.

שרת DNS

בונים בפרויקט שרת DNS שבעת הפעלה מರחילה פקודות מסוג DNS REQUEST שמגיעות לכתובת בה הוא נמצא, ומחזיר לשולח פקעת DNS RESPONSE עם הכתובת של האתר אותו הוא בקש. במידה והבקשת אינה חוקית או שהכתובת לא נמצאת במאגר הוא יחזיר שגיאה (CODE==3). הכתובות DNS נמצאות במאגר קבוע.

```
DNS_SERVER_IP = "192.168.1.200"
DNS_SERVER_PORT = 53
DNS_MAC_ADDRESS = "00:00:00:00:00:02"
domains_list = [("the_famous_cat.com.", "127.0.0.1"),
                 ("www.google.com.", "8.8.8.8")]
```

לאחר שקיבלנו את הכתובת של השירות אפליקציה מהשרת DNS, אנחנו נרצה לפנות לשרת האפליקציה ולבקש תמונה של חתול.

בקובץ `client.py` אנחנו יוצרים תקשורת עם שני השירותים - תחיליה קיבלנו כתובת IP מה DHCP שגם מחזיר את הכתובת של שירות DNS ולאחר מכן בעזרת הכתובת פנינו לשרת DNS לקלוט את הכתובת של שירות הפאליקציה הרלוונטי שאצלנו נקרא "the_famous_cat.com". נציג שבסהשר הפוך, אנחנו משתמשים בכתובת מקומית "localhost" או 127.0.0.1 לתקשורת עם שרת האפליקציה ושרת המידע המקומיים.

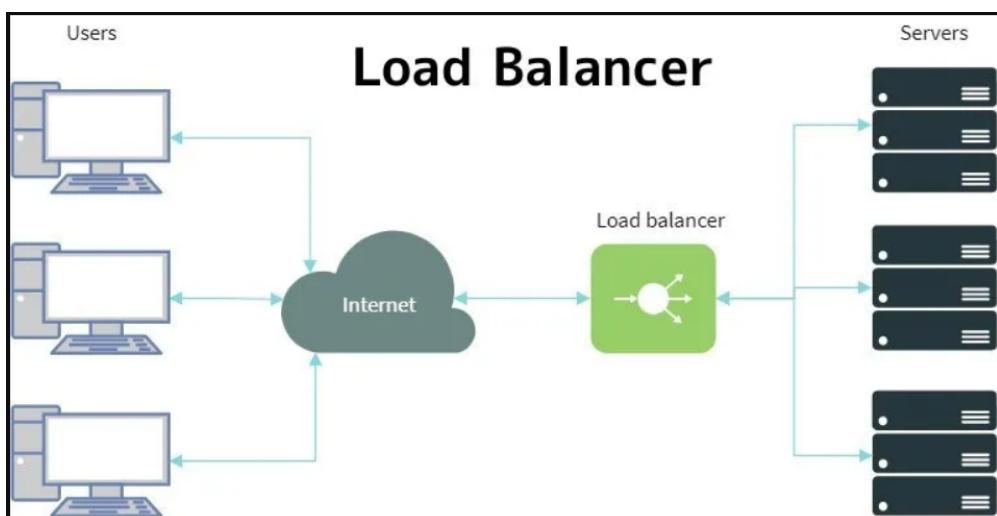
חלק רביעי - שירות האפליקציה והשירותי DATA

בפרויקט שלנו נדרשו לבנות שירות שיעודו לעשות REDIRECT לשרת חיצוני, ממנה הלקוח יוריד קובץ. אנחנו בחרנו בפרויקט למשרר רענון שמיועד לרוב למערכות מידע גדולות והוא Load Balancer (מאזן עומס).

מהו Load Balancer

כאשר צריכים כמות גדולה מאוד של משתמשים את אותו פרט מידע, השירות בוודד לא יכול להתמודד בעומס, והתגובה שלו תהיה איטית מאוד ואף יתכן שיאבדו בקשות. לכן, אנו נרצה להוריד את העומס משרת אחד, וליצור שירות ביןיהם המחבר בין השירותים אחרים שכולים מכילים את אותו מידע.

כלומר, משתמש אשר פונה לשרת הכללי וմבקש מידע, יכול לקבל את המידע משרת מס' 2, שירות מס' 3 או שירות מס' 4, והמידע שיחזור למשתמש יהיה אותו המידע.



מימוש השירות Load Balancer

אצלנו בפרויקט Load Balancer הוא שירות האפליקציה שיעודו לקבל בקשות HTTP GET, והוא מפנה לאחד משלושת השירותים האחרים (REDIRECT) המכילים תמונה של Chatbot.

* לשם ההדגמה, בפרויקט שלנו לכל שירות יש תמונה של Chatbot ייחודית אליו - כדי לבדוק בבדיקות שאכן הכל פועל כנדרש.

** שירות האפליקציה אינם אובדים כמו שירות פרויקט, אין עליו מידע - הוא רק אחראי לנתח את הבקשות לשירותים אחרים.

*** הבחירה שלנו לנתח בפרויקט היא רנדומלית(בין שלושת השירותים), אך בפועל ניתן לשנות בצורה חכמה יותר על הנתח של השירותים לביצוע יעיל יותר.

**** מימשנו את אתamazon עמוס שלנו בשני דרכי: פעם אחת בעזרת שירות INTERNET CHICAGO (אשר דורשים חיבור לINTERNET) ופעם אחרת שירותי מקומיים שאנו יוצרים עם תМОנות שאנו בחרנו. את שאר הפרויקט נציג לפי הדרך השנייה כדי שלא יהיה צורך בחיבור לINTERNET בעת ההגשה, וגם כדי להווסף עוד למורכבות, בעמוד הבא ניתן לראות פירוט על השירותים המקומיים.

שרת ה-DATA המקוומיים

בפרויקט יצרנו קובץ file_server.py שמקבל 3 ערכים:

1. כתובת IP או localhost
2. פורט עליו השרת ירוץ
3. קובץ שהשרת מקבל

בעת הריצה של הקובץ, יפעל שרת שמתחבר לפורט של הכתובת שקיבל, והוא ידע להחזיר לבקשת HTTP את התמונה שנבחרה בעת הרצת הקובץ.

בנוסף לקובץ הנ"ל, הכנו את הקובץ file_server.py שבעת הרצתו, הוא ניגש לfile_server.py ומריץ אותו כל פעם עם פרמטרים שונים, ומקבלים 3 שרתים אשר מכילים כל אחד תוכן שונה (תמונה של חתול עם מספר בהתאם) שרצים על כתובת ופורט שונה.

שרת האפליקציה - Load Balancer

```

servers = [
    ("localhost", "30354", "cat1.png"),
    ("localhost", "30355", "cat2.png"),
    ("localhost", "30356", "cat3.png"),]

if __name__ == "__main__":
    print("starting servers")
    os.chdir(os.path.dirname(os.path.abspath(__file__)))
    for server in servers:
        command = ["python3", "file_server_socket.py"] + list(server)
        subprocess.Popen(command)

```

שרת האפליקציה לא מכיר קבצים. השרת יודע לקבל בקשות חתול, והוא יודיע לשלווח חזרה ללקוח HTTP REDIRECTHTTP שמנפה את הלוקוח לשרת ממנו ניתן להוריד את התמונה. כל לוקוח שניגש אליו יוכל לקבל לפי החלטת שרת האפליקציה (אצלנו זה רנדומלי) שרת שונה ממנה שקיבל קודמו להורדת התמונה.

כאשר אנו נרים את הלוקוח אנחנו מקבלים רק פעם אחת אותה התמונה שנקבל כי החלטנו לעשות בכל שרת DATA מקומי, יכול תמונה שונה של חתול לשם זיהוי ובדיקות.

חלק חמישי - פרוטוקול ה-RUDP

בפרויקט אנחנו משתמשים בשני פרוטוקולים שונים של תקשורת בין הלקוח והשרתים שהוא פונה אליהם.

הפרוטוקול הראשון הוא TCP. פרוטוקול מוכר בעל פיצ'רים כמו CONGESTION CONTROL, FLOW CONTROL, RELIABILITY ועוד מגוון רחב. הפרוטוקול מבצע את הפעולות הללו בלבד, בצורה דיפולטיבית כמו שהגדירו אותו, ככלומר בעת חיבור חדש שאנו אצור עם שרת, באופן אוטומטי הפרוטוקול ידע לשנות את גודל החלון העברת מידע, לבדוק אם חבילה לא הגעה ליעד ולעשות בהתאם RETANSMITION וכו'.

הפרוטוקול השני הוא UDP. הפרוטוקול הנ"ל הוא פרוטוקול המבוסס על פרוטוקול UDP המוכר, ובא לתקן בעיות מוכרות בו ולהפוך אותו לאמין.

כאשר אנו שולחים בטעות פרוטוקול UDP חבילה, אנחנו לא יודעים אם חבילה הגיעה או אבדה בדרך, אם אנחנו שולחים כמה חבילות ביחד, אנחנו לא יכולים לדעת אם אחת לא הגעה וצריך לשוחח אותה שוב, או שהכל הגיע כמו שציריך? בכלל, כמה זמן בכלל לחכות עד שאני אשלח שוב אם לא קיבלתי תשובה?

כדי לפתרור את הבעיה הנ"ל נרצה להשתמש בפרוטוקול RUDP שיפורו אותן וيسפר את הפרוטוקול UDP המוכר.

לשם כך, אנחנו נבנה שכבה תעבורת משלנו שתתלבש על הפרוטוקול UDP, וכל שרת ולקוח שיכיר את ה-HEADER של הפרוטוקול - ידעו איך לבצע פעולות כמו:

- הקטינה והגדלה של חלון העברת המידע.
- להחזיר בקשה לNONESSION של חבילות שלא הגיעו.
- קבלת אישור ack שהמידע שלחתי עד כה התקבל ועובד.

ועוד ידובר עליהם בהמשך.

אנחנו השתמשנו בספריית scapy על מנת ליצור שכבה נוספת מתחת לשכבה UDP המוכרת, שכבה זאת נועדה לאחסן פרמטרים שהשרות והלקוח יכירו כפי שמתיואר הנ"ל.
בחרנו להשתמש במבנה עזר של SCTP, כדי להעביר מידע בין השירות ללקוח.

השתמשנו ב 6 סוגי של שכבות SCTP שונות, יש לציין שהשימוש בפרמטרים של כל שכבה אינו תואם את איך שSCTP אמרור להיות, אלא תואם את מה שנדרש בפרויקט זהה לפי מה שראינו לנו.

להלן סוג הsharpes:

חבילת sctp.SCTPChunkInit

0	1	2	3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1			
+++++	+++++	+++++	+++++
TYPE	FLAGS	LEN	
+++++	+++++	+++++	+++++
INIT TAG			
+++++	+++++	+++++	+++++
A RWND			
+++++	+++++	+++++	+++++
N OUT STREAMS	N IN STREAMS		
+++++	+++++	+++++	+++++
INIT TSN			
+++++	+++++	+++++	+++++
PARAMS			
+++++	+++++	+++++	+++++

סוג החבילה `type=1`

scapy, גודל השכבה, מאותחל אוטומטי ע"י

`init_tag`, תגיית אתחול, ייחודית לכל session בין הלקוח לשרת, מקבלת ערך רנדומלי בשילחה `rwnd_a`, חלון העברת מידע מפורסם, מימוש של FLOW CONTROL, הלקוח שולח כמה מידע הוא מוכן לקבל, מאותחל ל 50

`tsn_init`, מספר רצף שידור לאתחול (TSN), ייחודית לכל פקטה והו ack שלו, מקבלת ערך רנדומלי בשילחה

`params, n_out_streams, n_in_streams, flags`
חבילת sctp.SCTPChunkInitAck

0	1	2	3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1			
+++++	+++++	+++++	+++++
TYPE	FLAGS	LEN	
+++++	+++++	+++++	+++++
INIT TAG			
+++++	+++++	+++++	+++++
A RWND			
+++++	+++++	+++++	+++++
N OUT STREAMS	N IN STREAMS		
+++++	+++++	+++++	+++++
INIT TSN			
+++++	+++++	+++++	+++++
PARAMS			
+++++	+++++	+++++	+++++

סוג החבילה `type=2`

scapy, גודל השכבה, מאותחל אוטומטי ע"י

`init_tag`, תגיית אתחול, ייחודית לכל session בין הלקוח לשרת, מקבלת ערך לפי חבילת האתחול שהלקוח שלחו

`rwnd_a`, חלון העברת מידע מפורסם, מימוש של FLOW CONTROL, השרת שולח כמה מידע הוא מוכן לקבל, מאותחל ל 50

`tsn_init`, מספר רצף שידור לאתחול (TSN), ייחודית לכל פקטה והו ack שלו, מקבלת ערך לפי חבילת האתחול שהלקוח שלחו `params, n_out_streams, n_in_streams, flags`

חבילת sctp.SCTPChunkData

```

0           1           2           3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+
|      TYPE      |RESERVE|D|U|B|E|          LEN
+-----+-----+-----+-----+
|                                TSN
+-----+-----+-----+-----+
|      STREAM ID     |      STREAM SEQ
+-----+-----+-----+-----+
|                                PROTO ID
+-----+-----+-----+-----+
|      DATA      |
+-----+-----+-----+

```

proto_id, *reserved*, *unordered*, *stream_id*, *payload*, *data*, *stream_seq*, *stream*, *tsn*, *ack*, *len*, *scapy*, גודל השכבה, מאותחל אוטומטי ע"י *send*, *beginning*, *ending*, האם זהה פקחת מידע האחרון של פעולה ה *send*, אם כן 1, אם לא 0 *delay_sack*, האם עברו פקחת המידע ההזוז השרת אמר לחדר *ack*, אם כן 1, אם לא 0 *type=0*, סוג החבילה

: sctp.SCTPChunkSACK

`type=3`, סוג החבילה
`len`, גודל השכבה, מאותחל אוטומטי ע"י scapy
`cumul_tsn_ack`, מספר רצף שידור, ייחודית לכל פקטה וו ack שלה, בהתאם ל `tsn` של פקetta
`ack`. המידיע עליה צריך להחזיר ack.
`a_rwnd`, חילון העברת המידע העדכני של המקלט, רלוונטי ל FLOW CONTROL
`n_gap_ack`, `n_dup_tsn`, `gap_ack_list`, `dup_tsn_list`, `flags`

חבילה 7 : sctp.SCTPChunkShutdown

0	1	2	3
0 1 2 3 4 5 6 7 8 9 0	1 2 3 4 5 6 7 8 9 0	1 2 3 4 5 6 7 8 9 0	1
+-----+-----+-----+-----+	+-----+-----+-----+-----+	+-----+-----+-----+-----+	+-----+-----+-----+-----+
TYPE FLAGS LEN			
+-----+-----+-----+-----+	+-----+-----+-----+-----+	+-----+-----+-----+-----+	+-----+-----+-----+-----+
CUMUL TSN ACK			
+-----+-----+-----+-----+	+-----+-----+-----+-----+	+-----+-----+-----+-----+	+-----+-----+-----+-----+

סוג החבילה $type=7$,
scapy, גודל השכבה, מאותחל אוטומטי ע"י len
מספר רצף שידור אחרון $cumul_tsn_ack$
לא בשימוש $flags$

חבילה 8 : sctp.SCTPChunkShutdownAck

0	1	2	3
0 1 2 3 4 5 6 7 8 9 0	1 2 3 4 5 6 7 8 9 0	1 2 3 4 5 6 7 8 9 0	1
+-----+-----+-----+-----+	+-----+-----+-----+-----+	+-----+-----+-----+-----+	+-----+-----+-----+-----+
TYPE FLAGS LEN			
+-----+-----+-----+-----+	+-----+-----+-----+-----+	+-----+-----+-----+-----+	+-----+-----+-----+-----+

סוג החבילה $type=8$,
scapy, גודל השכבה, מאותחל אוטומטי ע"י len
לא בשימוש $flags$

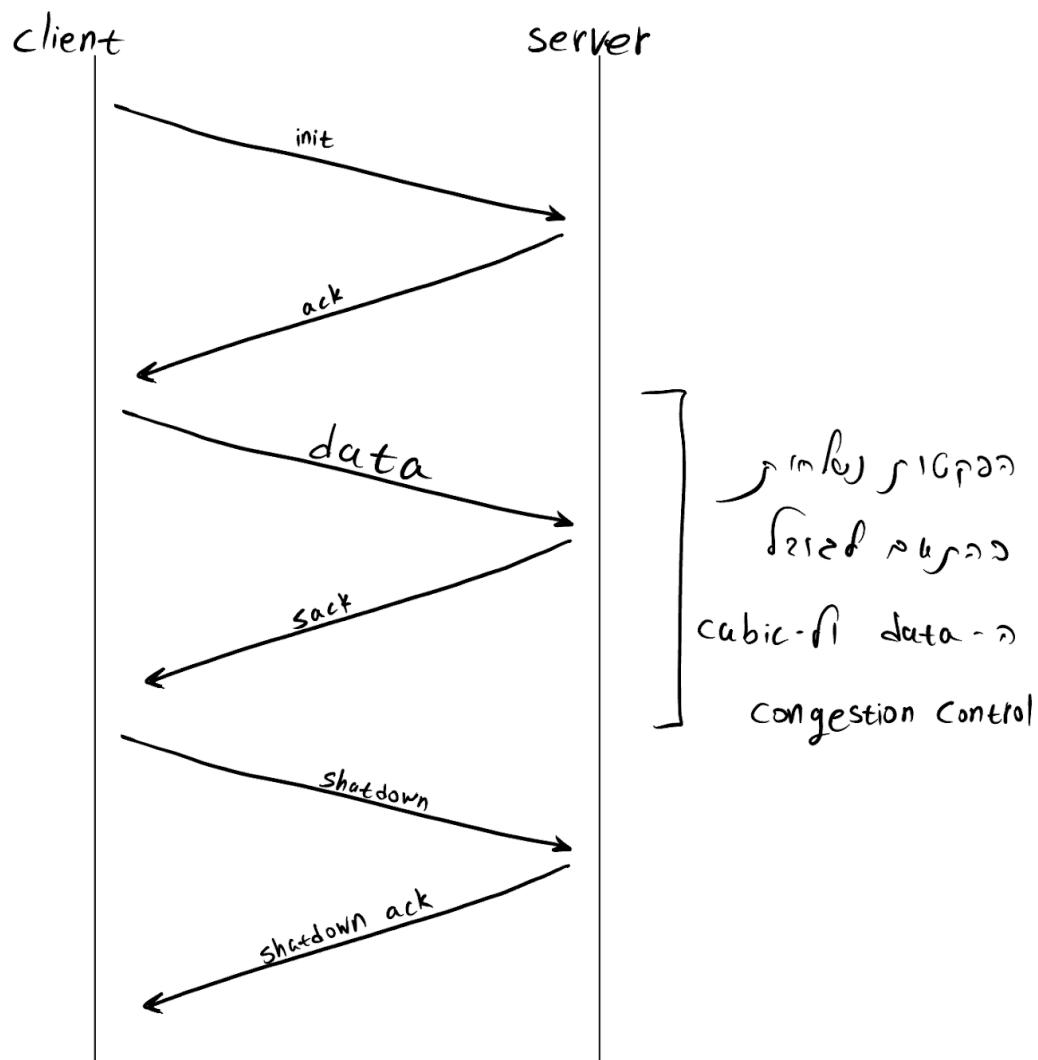
חלק שלישי - Reliability, Congestion Control, Flow Control ב-RUDP

מימוש של Reliability-Retransmission

כיצד המערכת מתגברת על איבוד חבילות?

המנגנון העיקרי העיקרי נמצא ב-RUDP והוא משתמש ב-ACK כדי לאמת שהמידע הושלט והועבר בהצלחה. כאשר הנמען מקבל חבילה, הוא שולח חזרה לשלוח ACK עם המספר הסידורי של החבילה (ts) שהוא קיבל. השילוח של ts ACKים מאפשר לשולח לדעת שהחביבה הגיעו לעדיה בהצלחה ולהתאים את זרימת הנתונים על פי זה (Congestion control).

בסק הכל, המנגנון של RUDP דומה למנגנון המבוצע ב-TCP. ב-TCP גם נעשה שימוש ב-ACKs על מנת לאמת שהמידע הועבר בהצלחה, ובעזרתם המקבל מאשר לשולח כי המידע הועבר בהצלחה. בשני המקרים, נעשה שימוש במנגנון של ACKs כדי לאמת שהנתונים הועברו ולפעול על מנת להשלים את הנתונים שלא הגיעו ליעדם.



עוד מנגנון נמצא בפעולה `start_timer` מתחילה את הטימר עבור חבילת נתונים או קבוצת חבילות כדי למדוד את הזמן ש עבר מאז שהן נשלחו. אם החבילות נמסרו לפונקציה, היא מ afsht את המונה שמחכה לתשובה ל-0 ו מגדרה את זמן ההתחלה לזמן הנוכחי. אחרת, היא מגדרה את זמן ההתחלה לזמן הנוכחי ללא איפוס של המונה שמחכה לתשובה. הפעולה נקראת לאחר שליחת כל חבילת נתונים או קבוצת חבילות.

הפעולה `check_timer` אחראית לבדוק אם הטימר כבר לא תקין עבור החבילות שנשלחו לפני כן. אם הטימר פג תוקף, זה אומר שהמקבל לא החזיר תשובה, בין אם זה ack או חבילת ה `data` הבאה, וכן השולח מניח שהחבילות אבדו ושלח אותן מחדש. זה נעשה על ידי שליחת החבילות שנשלחו מחדש, עד שלא עברו יותר מ-3 נסיבות, עם עיכוב בין כל השלחיות. אם המונה עולה מעל ל-2, זה אומר שמספר גדול של חבילות אבדו, והשלוח מתייחס לזה כתקלה בזמן אמת, מה שמשמעותו את ניהול העומס - `timeout`, אם החבילות נשלחו מחדש בהצלחה, הטימר מתאפשר.

עוד מנגנון שהוא משתמש בפイルוט חריגה של חבילה מסויימת, המנגנון מבצע שליחה נוספת של החבילת החסורה על מנת לשפר את ה-Y-`RELIABILITY` של התקשרות. המנגנון מנסה למנוע את אובדן המידע על ידי שליחת חבילות נוספות, כך שהמקבל יקבל את המידע הנדרש, זאת מתוך הבנה של השליחה שהתקבלת מסמלת שהתשובה המczופה לה לא התקבלה.

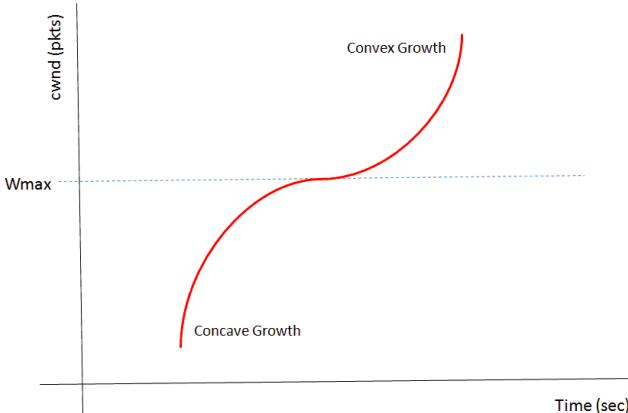
מנגנון זה גם תורם ל-Y-`RELIABILITY` של UDP על ידי כך שהוא מאפשר עוד דרך להתמודד עם איבוד חבילות. המנגנון מקפיד על חשיבות האמינות והשלמת המידע על מנת למנוע אובדן נתונים ופגיעה באיכות התקשרות.

הפונקציות והסבירים שצינו קודם קודם אפשרים ל-UDP להיות פרוטוקול העברת נתונים אמין ולעומוד בעקרונות הרלוונטיים לתקשרות רשת.

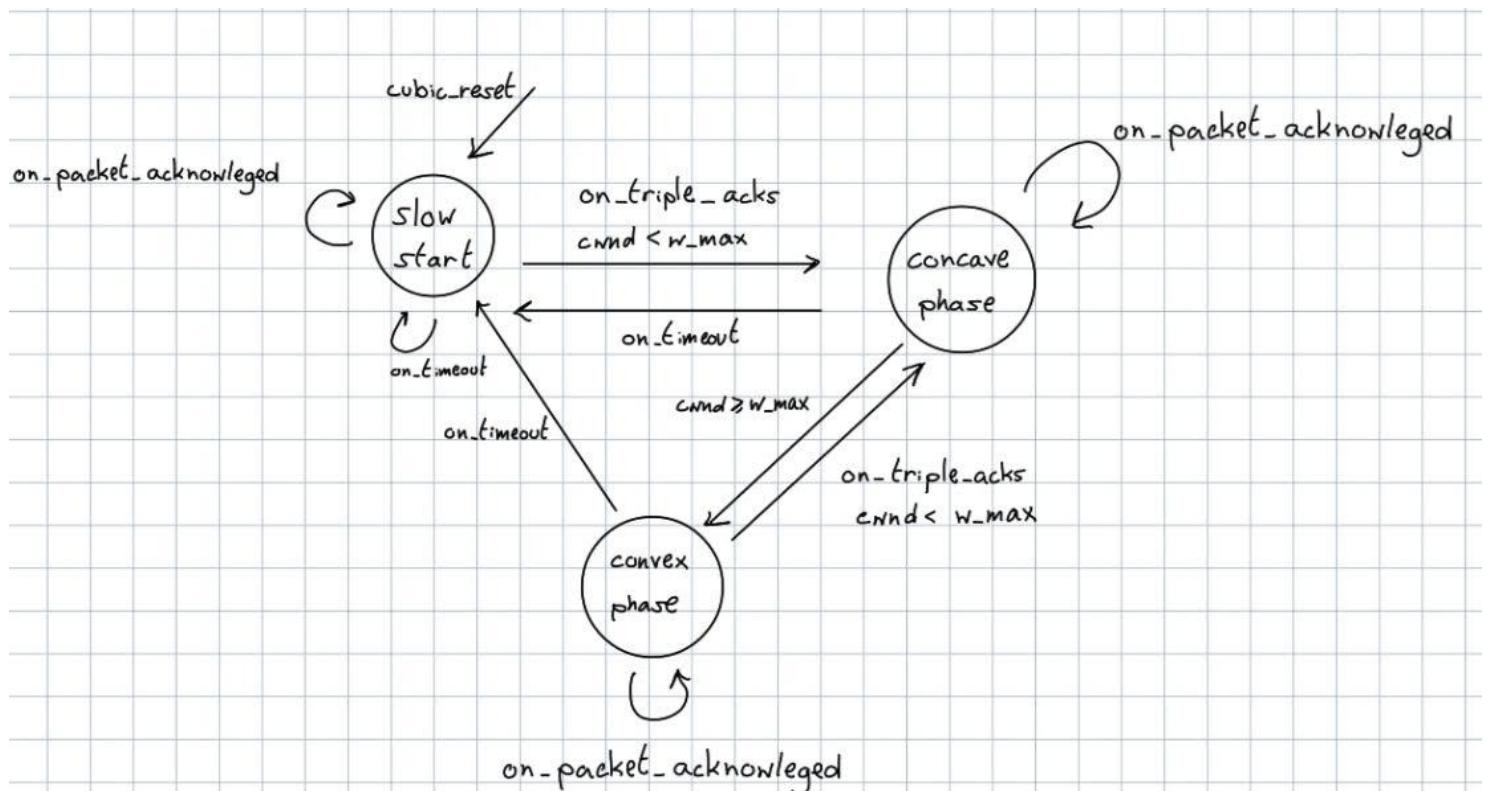
מימוש של Congestion Control: Cubic

כיצד המערכת מתגברת על בעיות latency?

- במוקם להשתמש ב cubic המקורי יצרנו גרסה משלנו אשר בהתנהגות שלו יש דמיון ל cubic.
- האלגוריתם מגיב לאירועים כמו:
- קבלת ack - מסמל עבור השולח שסדרת החבילות הקודמת הגיעה למקבל, כלומר שניתן להגדיל את חלון העברת המידע. בנוסף לכך, מתקבל מידע על הגבולות מצד המქבל (wnd_a) לגבי כמה מידע הוא מסוגל לעבד.
 - קבלת 3 dup acks - מסמל שבסדרת החבילות האחורונה היה פספוס שהמქבל זיהה או שהקצב השולח כל כך מהיר שהמქבל זיהה זאת ורוצה להתריע לשולח להאט, כאשר המქבל מזיהה זאת הוא שולח 3 dup, שהשולח מקבל זאת הוא מקטין את חלון העברת המידע.
 - timeout, כאשר השולח מחכה ל ack מהמქבל על מנת להמשיך לשולח מידע, ועובד זמן מוגדר מראש, זאת מסמל שהמქבל לא עומד בקצב הנוכחי ויש לאפס את חלון העברת המידע על מנת שהמქבל יחזור לשולח ack ים, לפיכך בעת timeout, השולח מאפס את חלון העברת המידע שלו.



להלן דיאגרמת מצבים המתארת את האלגוריתם:



להלן אלגוריתם:

```
class CubicCC:  
    def __init__(self):  
        self.cwnd = 1  
        self.int_cwnd = self.cwnd  
        self.w_min = 1  
        self.w_max = 0  
        self.peer_w_max = 0  
        self.jump = 1  
  
    def cubic_reset(self):  
        self.cwnd = 1  
        self.int_cwnd = self.cwnd  
        self.w_min = 1  
        self.w_max = 0  
        self.jump = 1  
  
    def on_packet_sent(self):  
        pass  
  
    def on_packet_acknowledged(self, a_rwnd):  
        if self.cwnd >= self.w_max:  
            self.cwnd += self.jump  
            self.jump *= 2  
            self.w_max = min(self.cwnd, a_rwnd)  
        else:  
            self.cwnd += max((self.w_max-self.cwnd)/2,1)  
  
        self.peer_w_max = a_rwnd  
        self.w_min = max(self.w_max * 0.4,1)  
        self.cwnd = min(self.peer_w_max,self.cwnd)  
        self.int_cwnd = int(self.cwnd)  
  
    def on_triple_ack(self):  
        self.jump = 1  
        self.w_max = self.cwnd  
        self.w_min = max(self.w_max * 0.4,1)  
        self.cwnd = self.w_min  
        self.int_cwnd = int(self.cwnd)  
  
    def on_timeout(self):  
        self.cubic_reset()
```

המשתנים המיצגים גודל של חלון כגון `cwnd` ו `w_max`, מדודים על ידי מספר פאקטות ולא על ידי כמות בייטים כפי שמקובל בtcp. בחרנו לעשות את זה כך מכיוון שהוא אלגוריתם ופרוטוקול שאנו בונים מאפס וכך ראיינו לנכון. כאשר מס הפקודות של חלון העברה הנוכחי לא שלם, הוא יועגל למיטה, בהתחשב בכך שרצינו שכל פקודה שנשלחה תהיה מלאה במידע כמה שאפשר.

מימוש של Flow Control - Stop-and-Wait

האובדן של חבילות הקשורות flow מכיוון שיכול להיות מגיע מכך שהמקבל מועמד עם כמות גדולה מדי של נתונים מהשולח. מנגנון window control כמו sliding windows משמשים למניעת אובדן חבילות על ידי הבטחת שהשולח לא ישלח נתונים בקצב שהמקבל לא מסוגל לעמוד בו, אנחנו בחרנו במנגנון "Stop-and-Wait ARQ" או "Stop-and-Wait Flow Control".

היא צורת מניעה לזרימת נתונים פשוטה וקלת ביצוע. בשיטה זו, המסר או הנתונים מתחלקים לפריטים קטנים ומספריים, והמקבל מציין שהוא מוכן לקבל סדרת חבילות מסוים. רק כאשר המקלט קיבל סדרת חבילות הנוכחית, השולח ישלח את סדרת חבילות הבא. שהפעולה תימשך עד שהשולח ישלח את סדרת חבילות האחורנית (ending), המסיימים את סוף העברה.

בנוסף לכך, אם המקלט יזהה חבילות כפולות, כMOVN שהוא התעלם מהם, אבל אם יזהה מספר חריג של פקודות כפולות, יקטין את חלון העברת המידע שלו (wndw_a), ובכך יגרום לשולח לשלוח פחות חבילות על מנת למנוע כפליות ולהסדיר את הקצב, הוא יעדכן את השולח בכך על ידי ack והפרמטרים שלו כפי שתואר קודם.

השיטות הללו כוללות תחת CONTROL FLOW, מכיוון שהן משומשת להבטיח שהמקבל לא יוכל לקבל יותר סדרות חבילות ממה שהוא מסוגל לטפל בהם בזמן אמת.

חלק שני - הרכזת קוד עם ווירשארק של השירות DHCP והשתת DNS

* היהות בכל הרצה, התקשרות ל-DHCP SERVER ול-DNS SERVER היא אותו הדבר, אنته פה
פעם אחת את התקשרות בווירשארק.

** קובץ ההורלה נקרא `only_dhcp_dns.pcapng`
הוראות לפילטר בווירשארק כדי לראות רק את החבילות הרצויות:

-on ethernet interface:

for DHCP AND DNS and for http connection with internet:
(udp && ip.dst == 255.255.255.255 && dhcp) || (dns && ip.addr == 127.0.0.1 || http)

בקובץ שרת DNS ובקובץ שרת DHCP

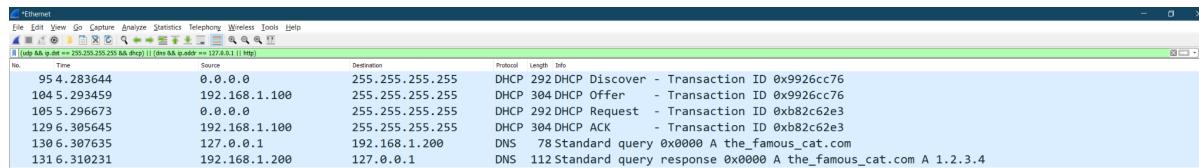
```
DNS_SERVER_IP = "192.168.1.200"
DNS_SERVER_PORT = 53
DNS_MAC_ADDRESS = "00:00:00:00:00:02"
domains_list = [("the_famous_cat.com.", "127.0.0.1"),
                 ("www.google.com", "8.8.8.8")]
```

```
3 | DHCP_SERVER_IP = "192.168.1.100"
4 | DNS_SERVER_IP = "192.168.1.200"
5 | DHCP_PORT = 67
```

פרטי הלוקוח כפי שהוגדר לנו:

APP_SERVER_P = 30353
CLIENT_P = 20054

ווירשארק לאחר הרצה עם הפילטר הנ"ל:



טרמינל של הלוקוח לאחר הרצה (רכ של DNS&DHCP):
הלוקוח הגיריל כתובת MAC חוקי ולאחר מכן יצר תקשורת עם ה-DHCP ועם ה-DNS. בהמשך
נשים לב לכתובת שמוועה פה בהקלות ווירשארק ובשרת ה-DHCP-P.

```
\Desktop\asdasd\Reliable_UDP\online_servers_and_client-tcp\client.py
Client MAC address: 00:16:3e:30:13:d4
Press Enter to continue...
.
Sent 1 packets.
DHCP discover sent, waiting for offer...
DHCP offer received, IP address is 10.100.102.76, DNS server is
192.168.1.200
10.100.102.76
.
Sent 1 packets.
DHCP request sent, waiting for ack...
Assigned IP address: 10.100.102.76
DNS server: 192.168.1.200
.
Sent 1 packets.
IP address of the_famous_cat.com: 1.2.3.4
PS C:\Users\user\Desktop\asdasd\Reliable_UDP>
```

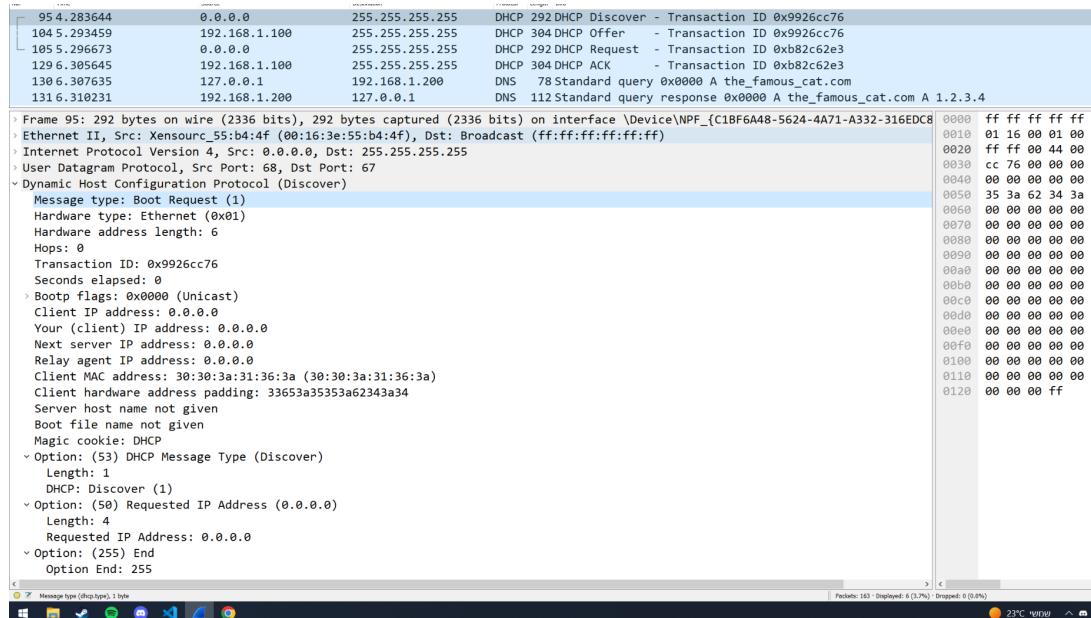
ניטוח שרת DHCP

לאחר הרצה נקלט בצד שרת DHCP

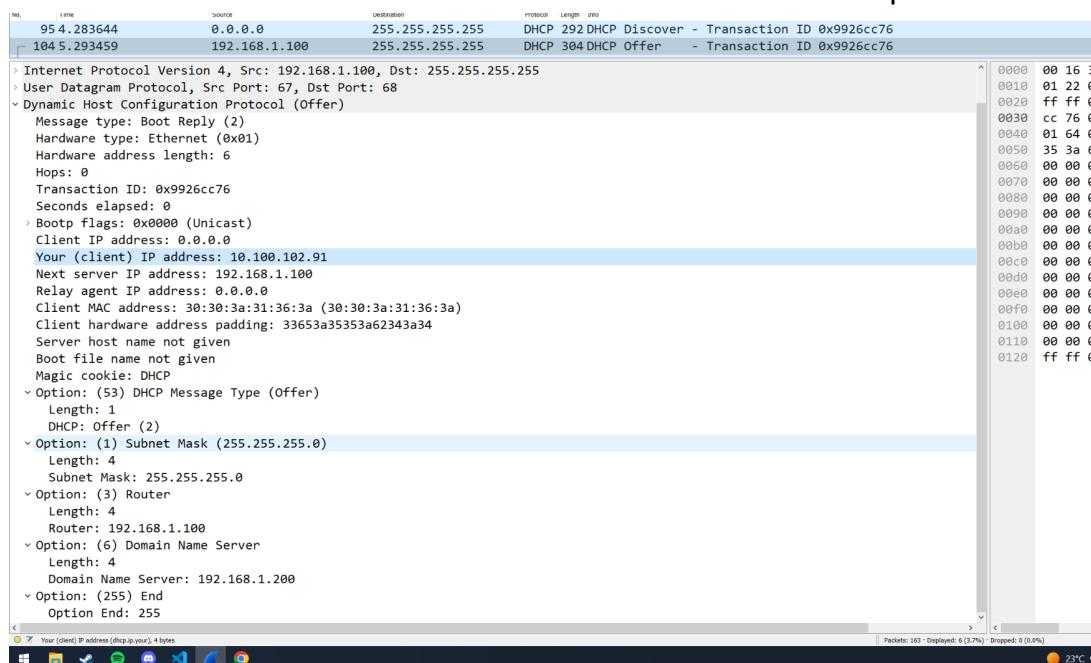
```
ktop\asdasd\Reliable_UDP\online_servers_and_client-tcp\DHCP.py
Starting DHCP server
DHCP discover received
Offering IP address 10.100.102.76 to client 00:16:3e:30:13:d4
.
Sent 1 packets.
Assigning IP address 10.100.102.76 to client 00:16:3e:30:13:d4
.
Sent 1 packets.
Current dic is:
{'00:16:3e:30:13:d4': '10.100.102.76'}
Sent ack. waiting for next packet...
```

אנו רואים שהוא מקבל DHCP DISCOVER, מזהה את כתובת ה-MAC של הלוקו ומציע לו כתובת חוקית רנדומלית שלא נמצאת במילון של כל הכתובות MAC וכותבות IP שהוא כבר חילק.

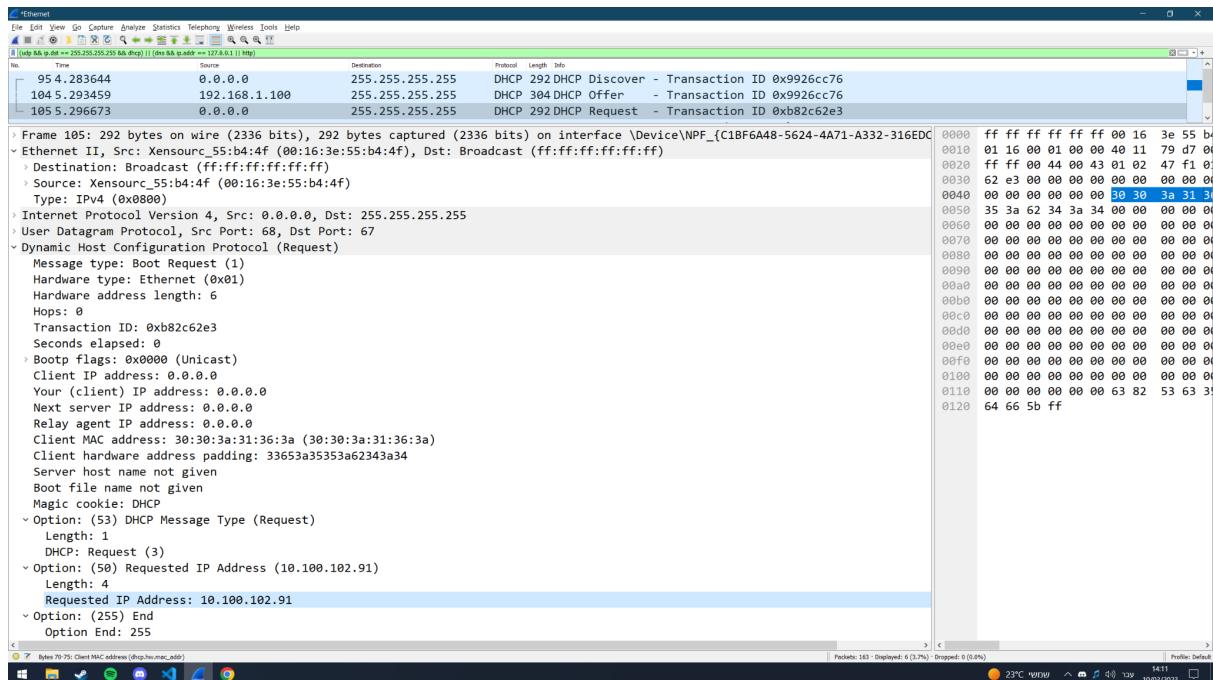
בוירשאך זה נראה כך:



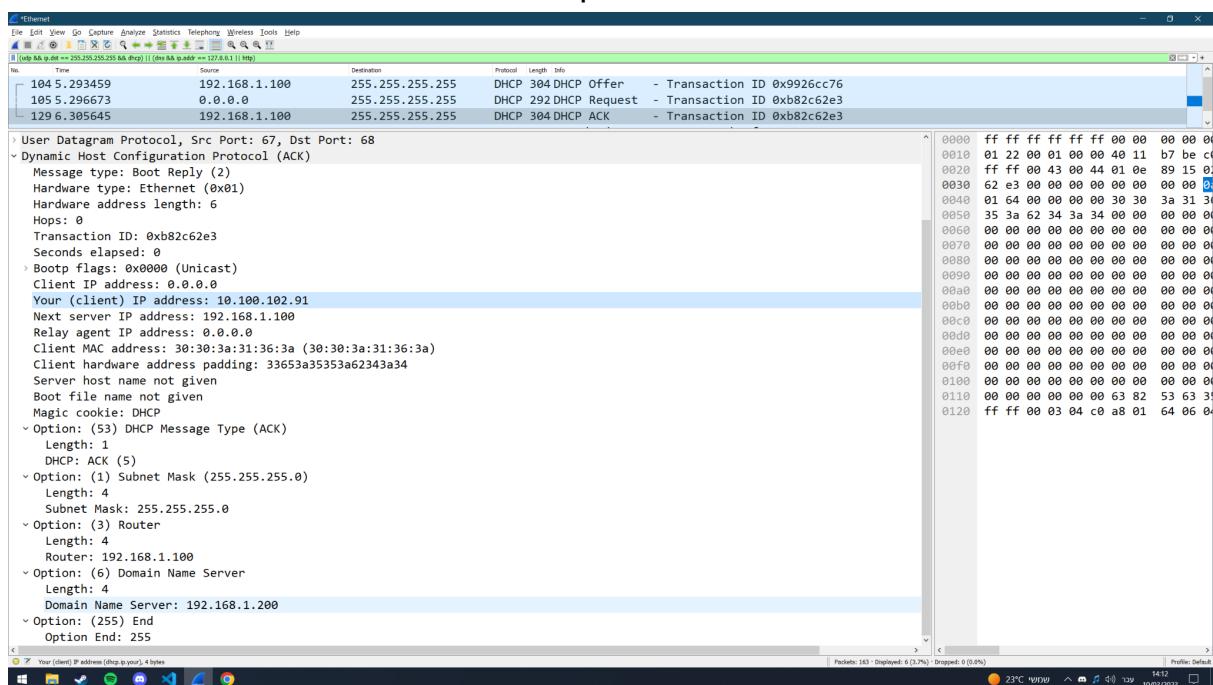
לאחר עיבוד הבקשה השרת מחייב DHCP OFFER



הלקוח שלוח DHCP REQUEST עם ה-IP שהשרת הציע לו

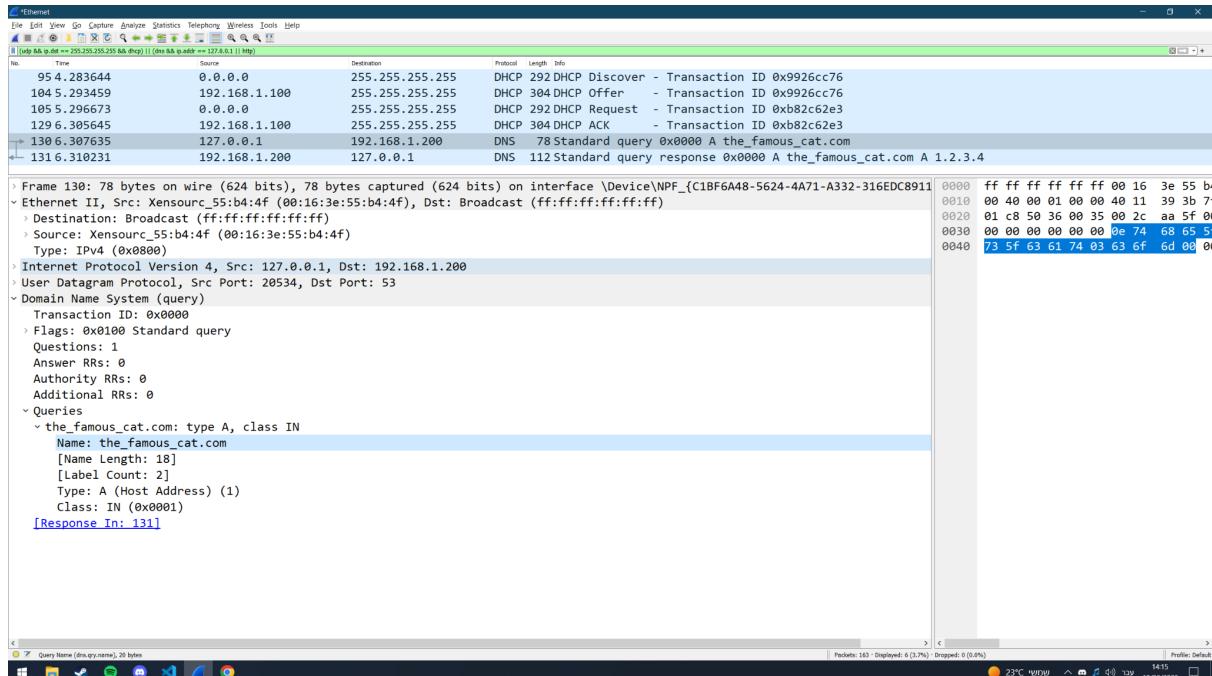


והשרת מחזיר ACK שהוא שומר את הכתובת במילון בידוד עם ה-IP

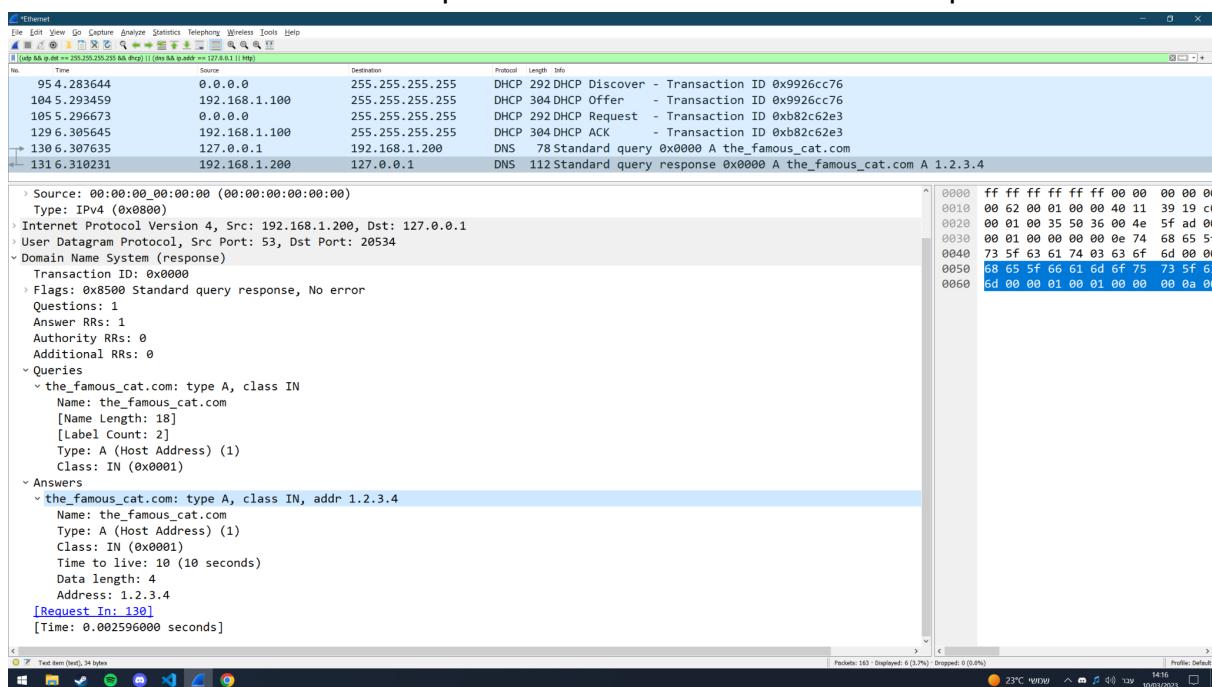


ניטוח שרת DNS

לאחר התקשרות של הלוקו עם השרת DHCP, הלוקו מקבל את הכתובת מה DHCP לשרת DNS. הלוקו פונה לשרת DNS לקבל את הכתובת של האתר "the_famous_cat.com" (שם סטמי לאתר שמכיר תמונות של חתול בשם הפרויקט).



השרת מעבד את הבקשה ומחזיר את הכתובת של שרת האפליקציה:



** לשם ההרצה זו שינו את המילון של DNS כך שכתובת לשרת האפליקציה 1.2.3.4. כפוי שניין לראות בתמונה בעמודים הקודמים.

חלק שמנוה- הרצה של שרת אפליקציה בפרוטוקול TCP עם שירותי DATA מקומיים

** כפי שרשمنו מוקדם, אנחנו ננתח רק את היכולות של השירותים המקומיים, כיוון שהשירותים החיצוניים זה רק וריאציה נוספת שהכנו לדוגמא

** לכל הרצה יש סרטון שסביר גם אנחנו מראים את הפROYKT

** שמota קבצי ההקלטה בוירשאך הם:

- local_servers_and_client_on_tcp-Ethernet.pcapng
- local_servers_and_client_on_tcp-loopback.pcapng

הוראות לפילטר בוירשאך כדי לראות רק את החבילות הרצויות:

On loopback interface:

for http connection with app server

tcp.port == 30355 || tcp.port == 30XXX WHERE XXX CAN BE 354,355,356

ה-loopback interface

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	TCP	56	20054 → 30353 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
2	0.000060	127.0.0.1	127.0.0.1	TCP	56	30353 → 20054 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
3	0.000103	127.0.0.1	127.0.0.1	TCP	44	20054 → 30353 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
4	0.000632	127.0.0.1	127.0.0.1	HTTP	79	GET / HTTP/1.1
5	0.000665	127.0.0.1	127.0.0.1	TCP	44	30353 → 20054 [ACK] Seq=1 Ack=36 Win=2619648 Len=0
6	0.000927	127.0.0.1	127.0.0.1	TCP	113	30353 → 20054 [PSH, ACK] Seq=1 Ack=36 Win=2619648 Len=69 [TCP segment of a reassembled PDU]
7	0.000953	127.0.0.1	127.0.0.1	TCP	44	20054 → 30353 [ACK] Seq=36 Ack=70 Win=2619648 Len=0
8	0.000986	127.0.0.1	127.0.0.1	HTTP	44	HTTP/1.1 307 Temporary Redirect
9	0.000999	127.0.0.1	127.0.0.1	TCP	44	[TCP Dup ACK 7#] 20054 → 30353 [ACK] Seq=36 Ack=70 Win=2619648 Len=0
10	0.001176	127.0.0.1	127.0.0.1	TCP	44	20054 → 30353 [ACK] Seq=36 Ack=71 Win=2619648 Len=0
11	0.001241	127.0.0.1	127.0.0.1	TCP	44	20054 → 30353 [FIN, ACK] Seq=36 Ack=71 Win=2619648 Len=0
12	0.001274	127.0.0.1	127.0.0.1	TCP	44	30353 → 20054 [ACK] Seq=71 Ack=37 Win=2619648 Len=0
13	0.001764	127.0.0.1	127.0.0.1	TCP	56	60232 → 30354 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
14	0.001803	127.0.0.1	127.0.0.1	TCP	56	30354 → 60232 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
15	0.001821	127.0.0.1	127.0.0.1	TCP	44	60232 → 30354 [ACK] Seq=1 Ack=56 Win=2619648 Len=0
16	0.002333	127.0.0.1	127.0.0.1	HTTP	98	GET / HTTP/1.1
17	0.002356	127.0.0.1	127.0.0.1	TCP	44	30354 → 60232 [ACK] Seq=1 Ack=55 Win=2619648 Len=0
18	0.003043	127.0.0.1	127.0.0.1	HTTP	65..Continuation	
19	0.003077	127.0.0.1	127.0.0.1	HTTP	65..Continuation	
20	0.003122	127.0.0.1	127.0.0.1	HTTP	65..Continuation	
21	0.003143	127.0.0.1	127.0.0.1	HTTP	65..Continuation	
22	0.003161	127.0.0.1	127.0.0.1	HTTP	65..Continuation	
23	0.003186	127.0.0.1	127.0.0.1	HTTP	65..Continuation	
24	0.003233	127.0.0.1	127.0.0.1	HTTP	65..Continuation	
25	0.003290	127.0.0.1	127.0.0.1	HTTP	65..Continuation	
26	0.003337	127.0.0.1	127.0.0.1	HTTP	65..Continuation	
27	0.003377	127.0.0.1	127.0.0.1	HTTP	65..Continuation	
28	0.003754	127.0.0.1	127.0.0.1	TCP	44	60232 → 30354 [ACK] Seq=55 Ack=654951 Win=2619648 Len=0
29	0.003801	127.0.0.1	127.0.0.1	HTTP	65..Continuation	
30	0.003839	127.0.0.1	127.0.0.1	HTTP	55..Continuation	
31	0.003921	127.0.0.1	127.0.0.1	TCP	44	60232 → 30354 [ACK] Seq=55 Ack=775773 Win=2619648 Len=0
32	0.074265	127.0.0.1	127.0.0.1	TCP	44	60232 → 30354 [FIN, ACK] Seq=55 Ack=775773 Win=2619648 Len=0
33	0.074289	127.0.0.1	127.0.0.1	TCP	44	30354 → 60232 [ACK] Seq=775773 Ack=56 Win=2619648 Len=0
34	0.074327	127.0.0.1	127.0.0.1	TCP	44	30354 → 60232 [FIN, ACK] Seq=775773 Ack=56 Win=2619648 Len=0
35	0.074339	127.0.0.1	127.0.0.1	TCP	44	60232 → 30354 [ACK] Seq=56 Ack=775774 Win=2619648 Len=0

בהרצה שעשינו אנחנו יכולים לראות את התקשרות בין הלקוח לשרת האפליקציה, ובין הלקוח לשרת המכיל את התמונה ששרת האפליקציה הפנה את הלקוח אליו. הפורטים רלוונטיים:

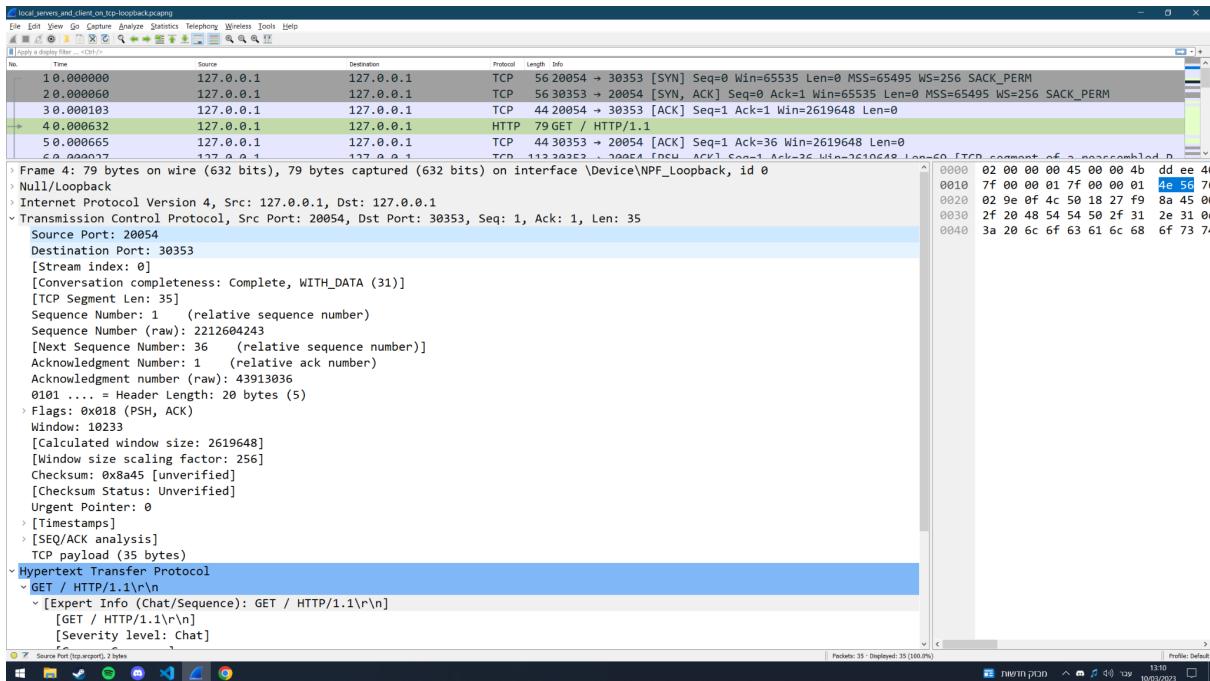
30353 - פורט עליון יושב שרת האפליקציה

20054 - פורט הלקוח

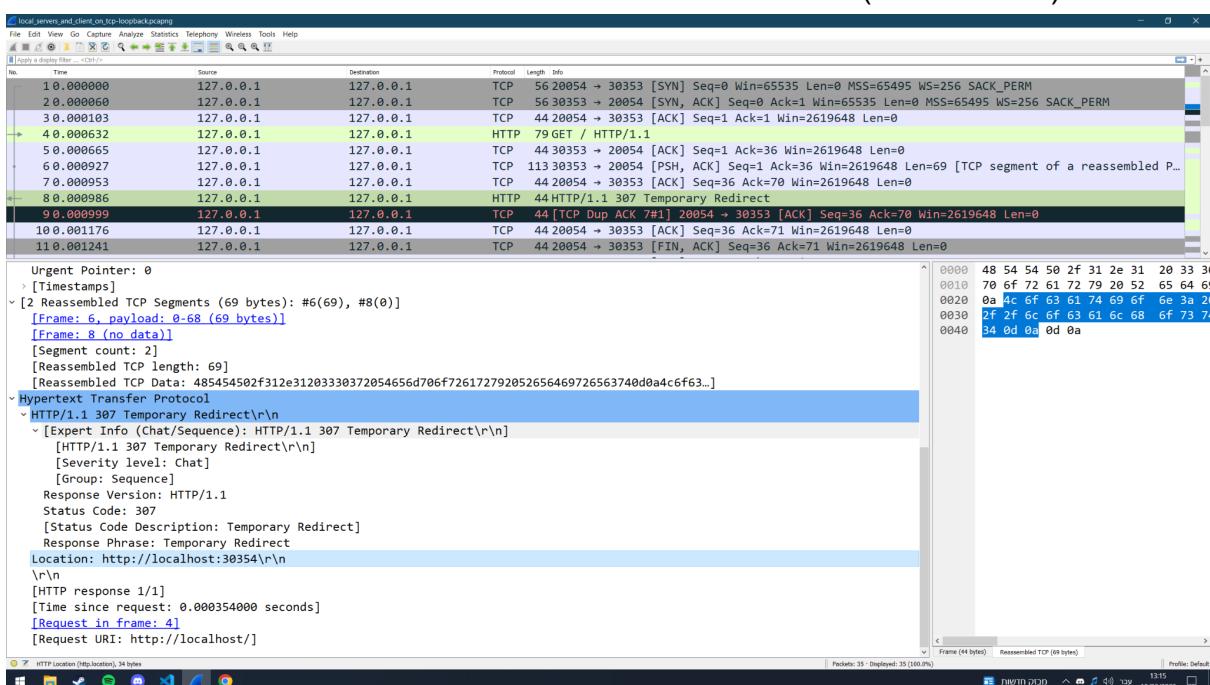
30354 - פורט השירות שמכיל את התמונה(יכול להיות גם 30355 או 30356)

ניתום

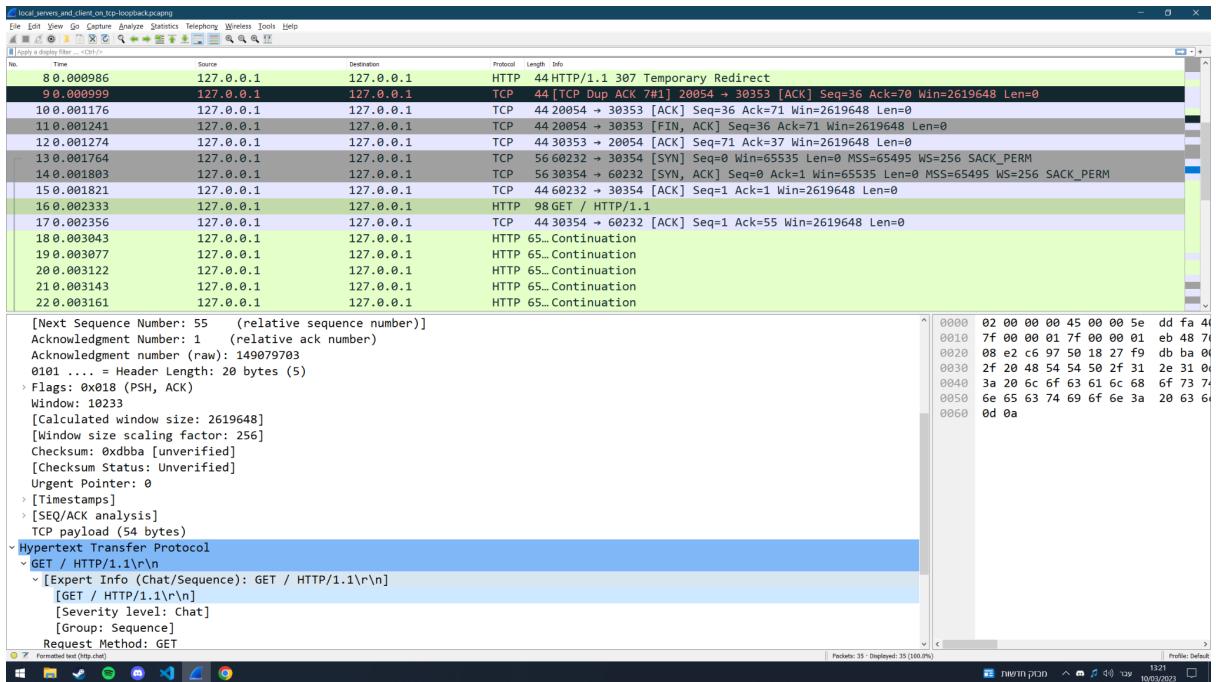
אם זואים בהתחלה את לחיצת היד'ם הראשונית ואז את הבקשת GET הנשלחת לשרת האפליקציה. ניתן לזרות לפי הפורטים (בתמונה הבאה).



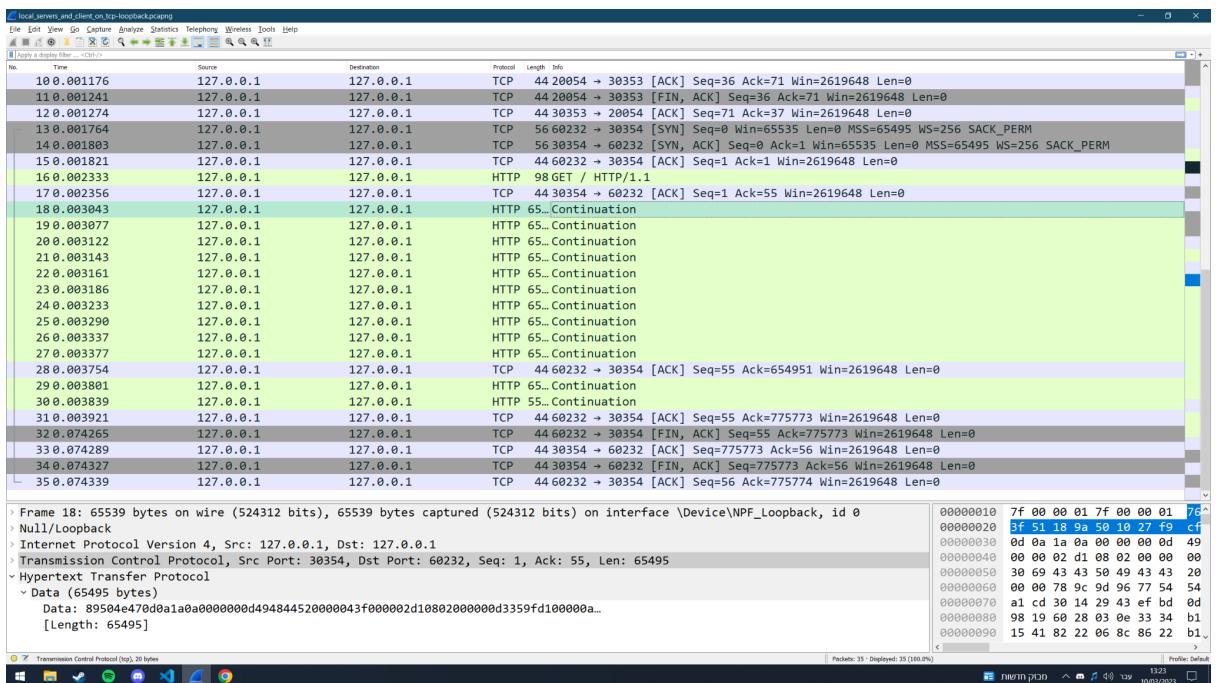
לאחר מכן נשלחה חזרה HTTP REDIRECT מהשרת אפליקציה עם הכתובת של השרת שמכיל את התמונה (בתמונה הבאה).



לאחר שלשרת יש הכתובת הוא יפנה אליה כדי לקבל את הקובץ, אך נוצר חיבור חדש בין הלקוח לשרת שקיבל מהשרת אפליקציה והלקוח שלוח שולח שוב בבקשת GET (בתמונה הבאה).



השרת מעבד את הבקשה ומתחיל לשלוח את התמונה בחלקים ללקוח (בתמונה הבאה).



לאחר מכן נסגרים שני החיבורים של ה-TCP על ידי חבילת FIN ACK שנשלחו מהשרותים, וחבילת ACK מהלקוח עצמה.

חלק תשיעי - רכזה של שרת אפליקציה בפרוטוקול RUDP עם שירותי DATA מקומיים

** קובץ ההוראה נקרא `rudp_0.pcapng`
הוראות לפילטר בוירשאך כדי לראות רק את החבילות הרצויות:

-on wifi and loopback interface:

```
(udp && ip.dst == 255.255.255.255 && dhcp) || (dns && ip.addr == 127.0.0.1) ||  
(udp.dstport==20054 || udp.srcport==20054) || (ip.addr==127.0.0.1 && !icmp)
```

מכיוון שאחד העקרונות של RUDP הוא Reliability, או בМИלים אחריות אמינות שליחת חבילות,
נבדק זאת על ידי הרכזת הקליינט עם אחוז אובדן חבילות שונים, נבדוק עם חיבור ה RUDP מצליח
להתגבר על החבילות שאבדו ובכל זאת להעביר את המידע בין השירות לקליינט וההפוך.

0% אובדן חבילות

ראשית נזודא שהפרמטרים של הסקוט מוגדרים היטב,
בקליינט נגדיר שלא יהיה איבוד פקודות, ונגדיר `pkt_printer` ל TRUE כך שיודפס מידע דלונטי, את
השאר נגדיר ל FALSE

```
105
106     matro68 *
107
108     def send_request(server_address, request):
109         # change parameters to print more or less data, and to change packet loss,
110         # default packet loss is -1 which means no packet loss
111         client_socket = SCTPSocket(packet_size=1024, pkt_printer=True, cc_printer=False, packet_loss=-1)
112         client_socket.bind(['localhost', CLIENT_P])
113         client_socket.connect(server_address)
114         client_socket.sendto(request)
```

בשרת האפליקציה נגדיר שלא יהיה איבוד פקודות, את השאר נגדיר ל FALSE

```
34
35
36     # Define a function to start the server and listen for incoming connections
37     matro68 *
38
39     def start_server():
40         # change parameters to print more or less data, and to change packet loss,
41         # default packet loss is -1 which means no packet loss
42         server_socket = SCTPSocket(packet_size=1024, pkt_printer=False, cc_printer=False, packet_loss=-1)
43         server_socket.bind(['localhost', server_port])
44         print(f'Server started and listening on server_port {server_port}...')
```

```

multi-server_socket
C:\Users\user\PycharmProjects\NetworkProject\venv\Scripts\python.exe C:
starting servers
ip is localhost', port is 30355', image is cat2.png
ip is localhost', port is 30354', image is cat1.png
ip is localhost', port is 30356', image is cat3.png

client
C:\Users\user\PycharmProjects\NetworkProject\venv\Scripts\python.exe C:
starting servers
ip is localhost', port is 30355', image is cat2.png
ip is localhost', port is 30354', image is cat1.png
ip is localhost', port is 30356', image is cat3.png

DNS
C:\Users\user\PycharmProjects\NetworkProject\venv\Scripts\python.exe C:
Starting DNS server on IP 192.168.1.200 and port 53

DHCP
C:\Users\user\PycharmProjects\NetworkProject\venv\Scripts\python.exe C:
Starting DHCP server on IP 192.168.1.200 and port 67

```

הרצה כל השרתים בקובץ והרצת וירשארק ברקע לפי ה필טר המתואר הנ"ל

```

app_server
started receiving
Returing response: HTTP/1.1 307 Temporary Redirect
Location: http://localhost:30355

multi-server_socket
C:\Users\user\PycharmProjects\NetworkProject\venv\Scripts\python.exe C:
starting servers
ip is localhost', port is 30355', image is cat2.png
ip is localhost', port is 30354', image is cat1.png
ip is localhost', port is 30356', image is cat3.png
connected to client on port: 30355
received: b'GET / HTTP/1.1\r\nHost: localhost\r\nConnection: close\r\n\r\n'
ended sending
ended thread
disconnected from client
Server started and listening on port 30355...
started threadwaiting

client
flags = 0x0
len = 20
init_tag = 0x81498
a_rwnd = 50
n_out_streams= 1
n_in_streams= 1
init_tsn = 0x54c07
\params \
-----sent-----
type- tsn=347144 seq=0 ds=1
waiting for ack seq=0 cmd=1 tsn=347144 ptsn=347144
-----received-----
type-3 tsn=347144 a_rwnd=50
waiting for data tsn=347144 ptsn=347144
-----received-----
type-0 tsn=347145 seq=0 ds=1
-----sent-----
type-3 tsn=347145 seq=0 a_rwnd=50
HTTP/1.1 307 Temporary Redirect
Location: http://localhost:30355

waiting for ack tsn=347146 ptsn=347146
ended thread
##[ SCTPchunkShutdownAck ]##
type_ = shutdown-ack
flags = 0x0
len = 4

session summary:
received packets: dict_keys([347143, 347144, 347145, 0, 347146])
sent packets: dict_keys([347143, 347144, 347145])
in flight packets: dict_keys([])
abandoned packets: dict_keys([347143, 347144, 347146])
('localhost', '30355')
got 18900 bytes

```

לאחר הרצה הקלינט הקוד יראה לנו את החטולה הרנדומלית ושומר אותה כקובץ בתיקייה.

No.	Time	Source	Destination	Protocol	Length	Info
1	2023-03-12 01:55:49	0.0.0.0	255.255.255.255	DHCP	292	DHCP Discover - Transaction ID 0xd87386d4
2	2023-03-12 01:55:50	192.168.1.100	255.255.255.255	DHCP	304	DHCP Offer - Transaction ID 0xd87386d4
3	2023-03-12 01:55:50	0.0.0.0	255.255.255.255	DHCP	292	DHCP Request - Transaction ID 0x3feeffcc
21	2023-03-12 01:55:51	192.168.1.100	255.255.255.255	DHCP	304	DHCP ACK - Transaction ID 0x3feeffcc
22	2023-03-12 01:55:51	127.0.0.1	192.168.1.200	DNS	78	Standard query 0x0000 A the_famous_cat.com
23	2023-03-12 01:55:51	192.168.1.200	127.0.0.1	DNS	112	Standard query response 0x0000 A the_famous_cat.com A 127.0.0.1
4	2023-03-12 01:55:51	127.0.0.1	127.0.0.1	UDP	64	20054 → 30353 Len=32
5	2023-03-12 01:55:51	127.0.0.1	127.0.0.1	UDP	64	30353 → 20054 Len=32
6	2023-03-12 01:55:51	127.0.0.1	127.0.0.1	UDP	96	20054 → 30353 Len=64
7	2023-03-12 01:55:51	127.0.0.1	127.0.0.1	UDP	60	30353 → 20054 Len=28
8	2023-03-12 01:55:51	127.0.0.1	127.0.0.1	UDP	132	30353 → 20054 Len=100
9	2023-03-12 01:55:51	127.0.0.1	127.0.0.1	UDP	60	20054 → 30353 Len=28
10	2023-03-12 01:55:51	127.0.0.1	127.0.0.1	UDP	52	20054 → 30353 Len=20
11	2023-03-12 01:55:52	127.0.0.1	127.0.0.1	UDP	48	30353 → 20054 Len=16
12	2023-03-12 01:55:52	127.0.0.1	127.0.0.1	TCP	56	63459 → 30356 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
13	2023-03-12 01:55:52	127.0.0.1	127.0.0.1	TCP	56	30356 → 63459 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
14	2023-03-12 01:55:52	127.0.0.1	127.0.0.1	TCP	44	63459 → 30356 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
15	2023-03-12 01:55:52	127.0.0.1	127.0.0.1	HTTP	98	GET / HTTP/1.1
16	2023-03-12 01:55:52	127.0.0.1	127.0.0.1	TCP	44	30356 → 63459 [ACK] Seq=1 Ack=55 Win=2619648 Len=0
17	2023-03-12 01:55:52	127.0.0.1	127.0.0.1	HTTP	65539	Continuation
18	2023-03-12 01:55:52	127.0.0.1	127.0.0.1	HTTP	65539	Continuation
19	2023-03-12 01:55:52	127.0.0.1	127.0.0.1	HTTP	65539	Continuation
20	2023-03-12 01:55:52	127.0.0.1	127.0.0.1	HTTP	17487	Continuation
24	2023-03-12 01:55:52	127.0.0.1	127.0.0.1	TCP	44	63459 → 30356 [ACK] Seq=55 Ack=213929 Win=2536704 Len=0
25	2023-03-12 01:55:52	127.0.0.1	127.0.0.1	TCP	44	[TCP Window Update] 63459 → 30356 [ACK] Seq=55 Ack=213929 Win=2602240 Len=0
26	2023-03-12 01:55:53	127.0.0.1	127.0.0.1	TCP	44	63459 → 30356 [FIN, ACK] Seq=55 Ack=213929 Win=2602240 Len=0
27	2023-03-12 01:55:53	127.0.0.1	127.0.0.1	TCP	44	30356 → 63459 [ACK] Seq=213929 Ack=56 Win=2619648 Len=0
28	2023-03-12 01:55:53	127.0.0.1	127.0.0.1	TCP	44	30356 → 63459 [FIN, ACK] Seq=213929 Ack=56 Win=2619648 Len=0
29	2023-03-12 01:55:53	127.0.0.1	127.0.0.1	TCP	44	63459 → 30356 [ACK] Seq=56 Ack=213930 Win=2602240 Len=0

בהרצת שעשינו אנחנו יכולים לראות את התקשרות בין הלקוח לשרת האפליקציה, ובין הלקוח לשרת המכיל את התמונה ששרת האפליקציה הפנה את הלקוח אליו. הפורטים רלוונטיים:

30353 - פורט עליו יושב שרת האפליקציה

20054 - פורט הלקוח

30356 - פורט השרת שמכיל את התמונה (יכול להיות גם 30355 או 30354) חבילות מס' 1,2,3,21,22,23 הן של התקשרות בין הקליינט ושרת DNS ושרת DHCP לפירוט והסבירים על כך יש לעבור לחלק השביעי.

חבילות 4-11 הם השיחה בין הקליינט לשרת אפליקציית REDIRECT, כאשר השיחה מתבצעת על RUDP, בוירשאך זאת נראית UDP שכך אנחנו יצרנו את RUDP והוא אכן מוכר לוירשאך.

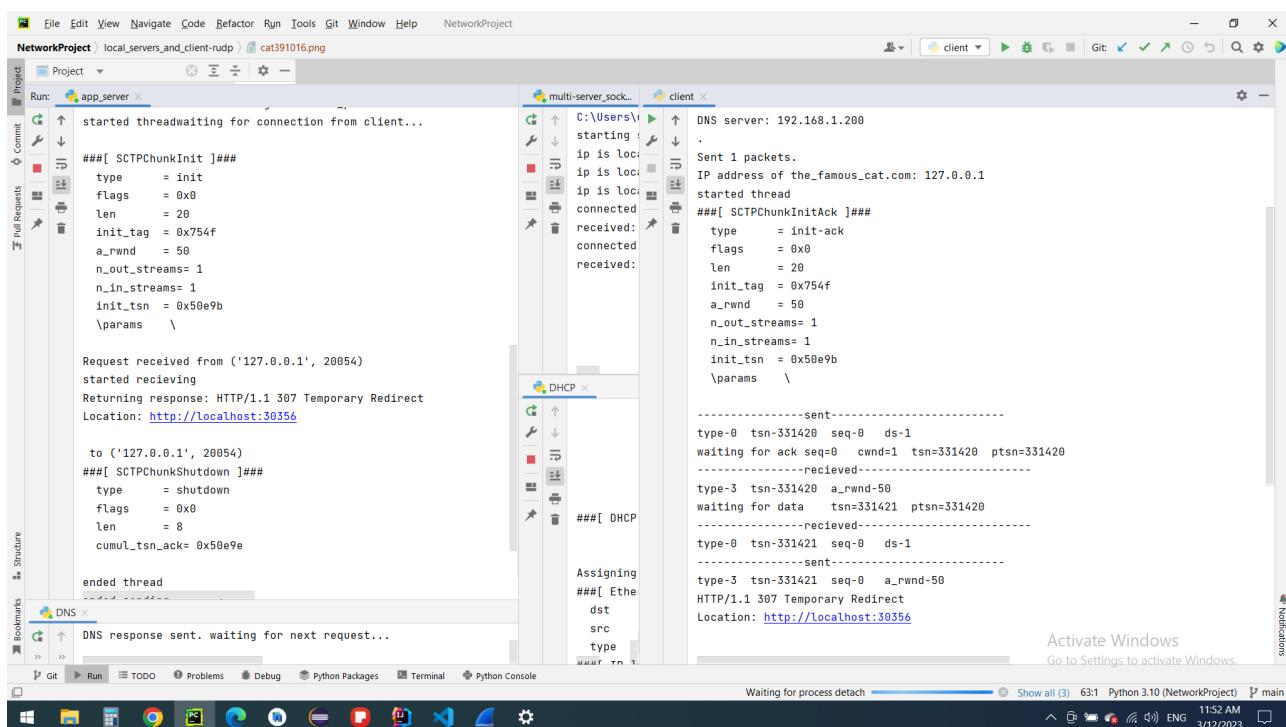
חבילות 12-29 הם השיחה בין הקליינט לשרת התמונה הרנדומלי, כאשר השיחה מתבצעת על TCP.

ניתנות

אנו רואים בהתחלה את לחיצת הידיים הראשונית על ידי חבילת INIT או INIT ACK בטרמינל ובהתאם להקלטת הוירשארק חבילות מס' 4 ו- 5, ואז את הבקשת HTTP GET נשלחת לשרת האפליקציה בטרמינל 331420 = tsn בוירשארק חבילה מס' 6), אחר כך את ה ack לבקשת ה HTTP GET, בטרמינל אותו tsn בוירשארק חבילה מס' 7.

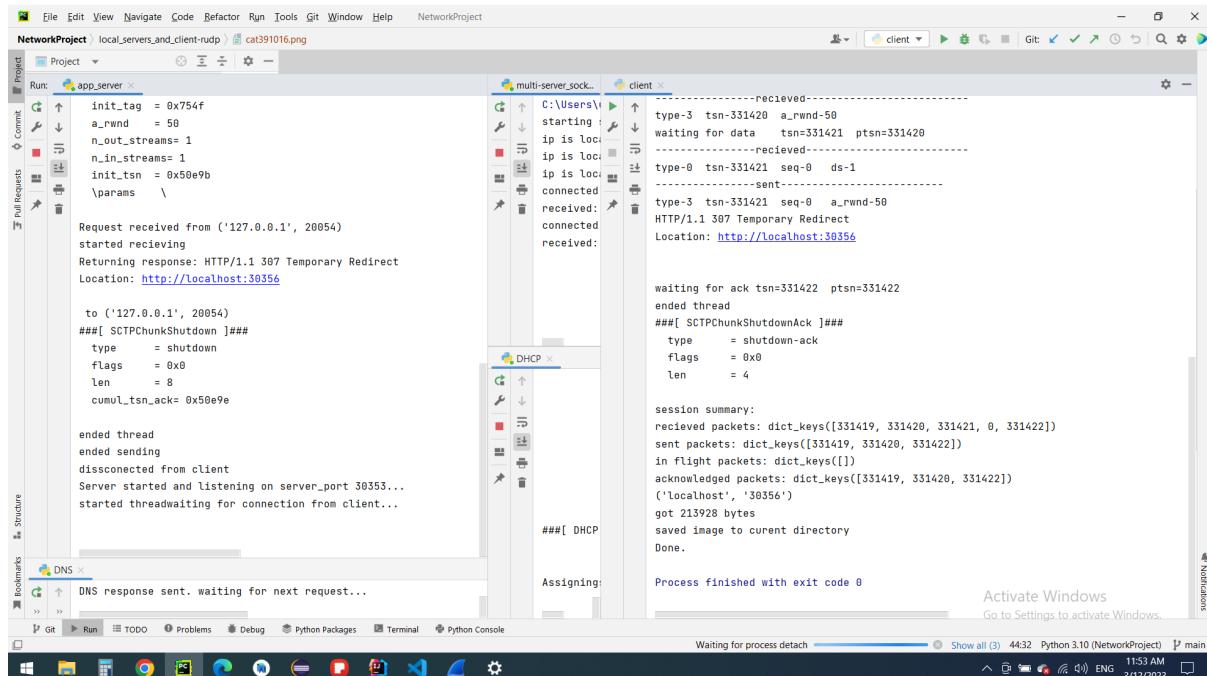
לאחר מכן נשלחה צדקה HTTP REDIRECT משרת אפליקציה עם הכתובת של השרת הרנדומלי שמכיל את התמונה, פה ספציפית קיבלנו את שרת התמונה עם פורט 30356.

את תשובה ה HTTP REDIRECT נשלחת לשרת האפליקציה בטרמינל 331421 tsn = 331421, בטרמינל אותו tsn בוירשארק חבילה מס' 8), אחר כך את ה ack לתשובה ה HTTP REDIRECT, בטרמינל אותו tsn בוירשארק חבילה מס' 9.

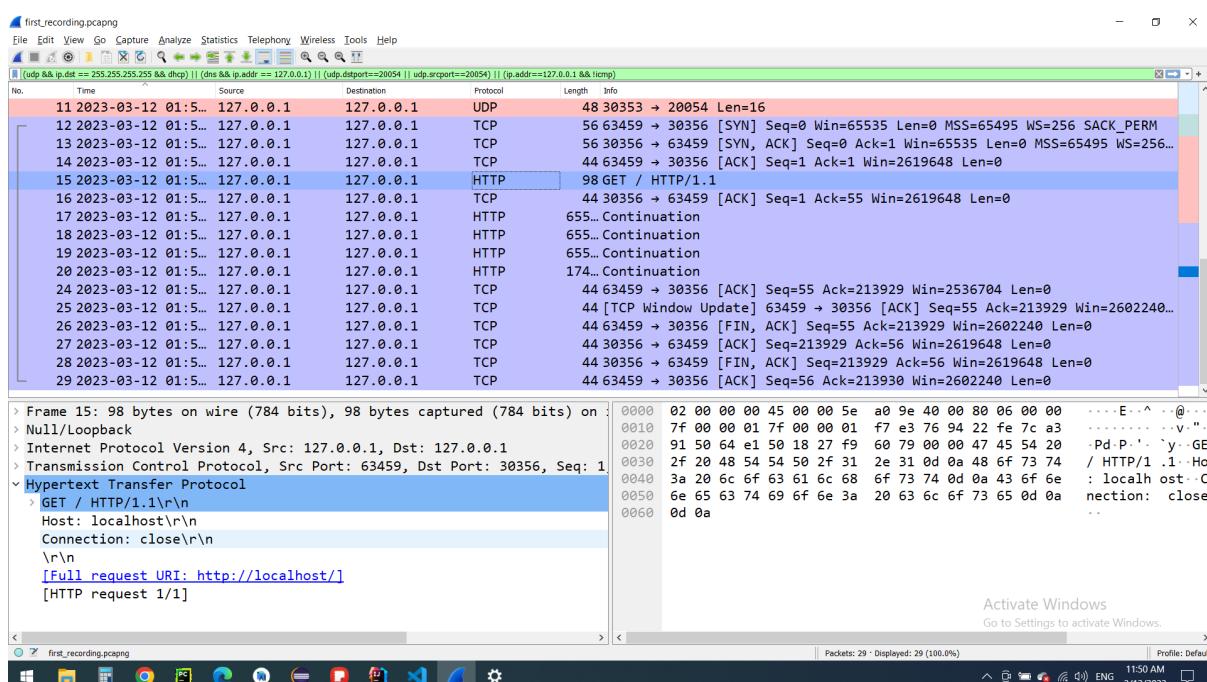


23 2023-03-12 01:55:51	192.168.1.200	127.0.0.1	DNS	112 Standard query response
4 2023-03-12 01:55:51	127.0.0.1	127.0.0.1	UDP	64 20054 → 30353 Len=32
5 2023-03-12 01:55:51	127.0.0.1	127.0.0.1	UDP	64 30353 → 20054 Len=32
6 2023-03-12 01:55:51	127.0.0.1	127.0.0.1	UDP	96 20054 → 30353 Len=64
7 2023-03-12 01:55:51	127.0.0.1	127.0.0.1	UDP	60 30353 → 20054 Len=28
8 2023-03-12 01:55:51	127.0.0.1	127.0.0.1	UDP	132 30353 → 20054 Len=100
9 2023-03-12 01:55:51	127.0.0.1	127.0.0.1	UDP	60 20054 → 30353 Len=28
10 2023-03-12 01:55:51	127.0.0.1	127.0.0.1	UDP	52 20054 → 30353 Len=20
11 2023-03-12 01:55:52	127.0.0.1	127.0.0.1	UDP	48 30353 → 20054 Len=16
12 2023-03-12 01:55:52	127.0.0.1	127.0.0.1	TCP	56 63459 → 30356 [SYN] Seq=

לבסוף אנו רואים את חבילות SHUTDOWN ACK ו SHUTDOWN בטרמינל ובהתקשרות בהקלטת הווירשאך חבילות מס' 10 ו- 11, אשר מסמלות על סיום חיבור בין הלוקו לשרת.



לאחר שלשרת יש הכתובת הוא יפנה אליה כדי לקבל את הקובץ, אך נוצר חיבור חדש בין הלוקו לשרת שקיבל מהשרת אפליקציה והlkoo שולח שוב בקשה GET. השרת מעבד את הבקשת ומחילה לשלוח את התמונה בחלקים לlkoo. לאחר מכן נסגר חיבור ה-TCP על ידי חבילת ACK שנשלחה מהשרת, וחבילת ACK מלהלוקה חוזרת.



10% אובדן חבילות

** קובץ הקלטה נקרא **rudp_10.pcapng**
ראשית נודע שהפרטטים של הסוקט מוגדרים היבש,
בקליינט נגדיר שיהיה 10% איבוד פקודות, ונגידר **pkt_printer** ו-**cc_printer** קר שיזופו
מידע רלוונטי

```

matroos* 
def send_request(server_address, request):
    # change parameters to print more or less data, and to change packet loss,
    # default packet loss is -1 which means no packet loss
    client_socket = SCTPSocket(packet_size=1024, pkt_printer=True, cc_printer=True, packet_loss=0.1)
    client_socket.bind(('localhost', CLIENT_P))
    client_socket.connect(server_address)
    client_socket.sendto(request)
    response = client_socket.recvfrom(1024).decode()

```

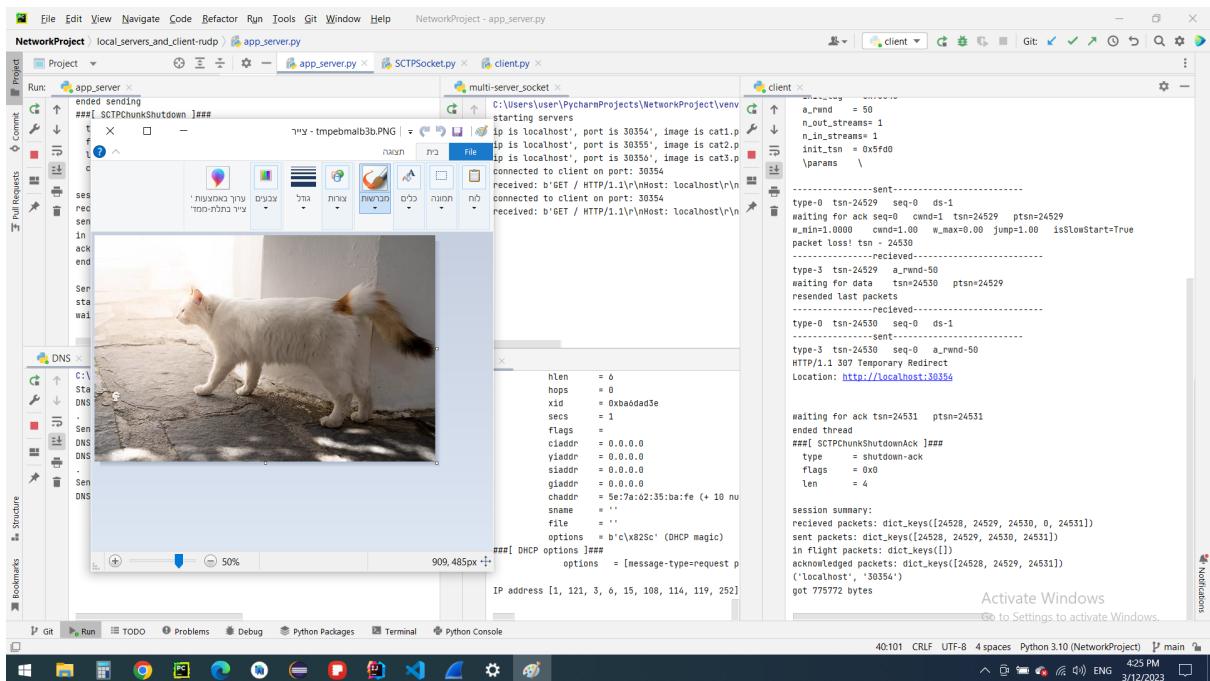
בשרת האפליקציה נגדיר באופן זהה

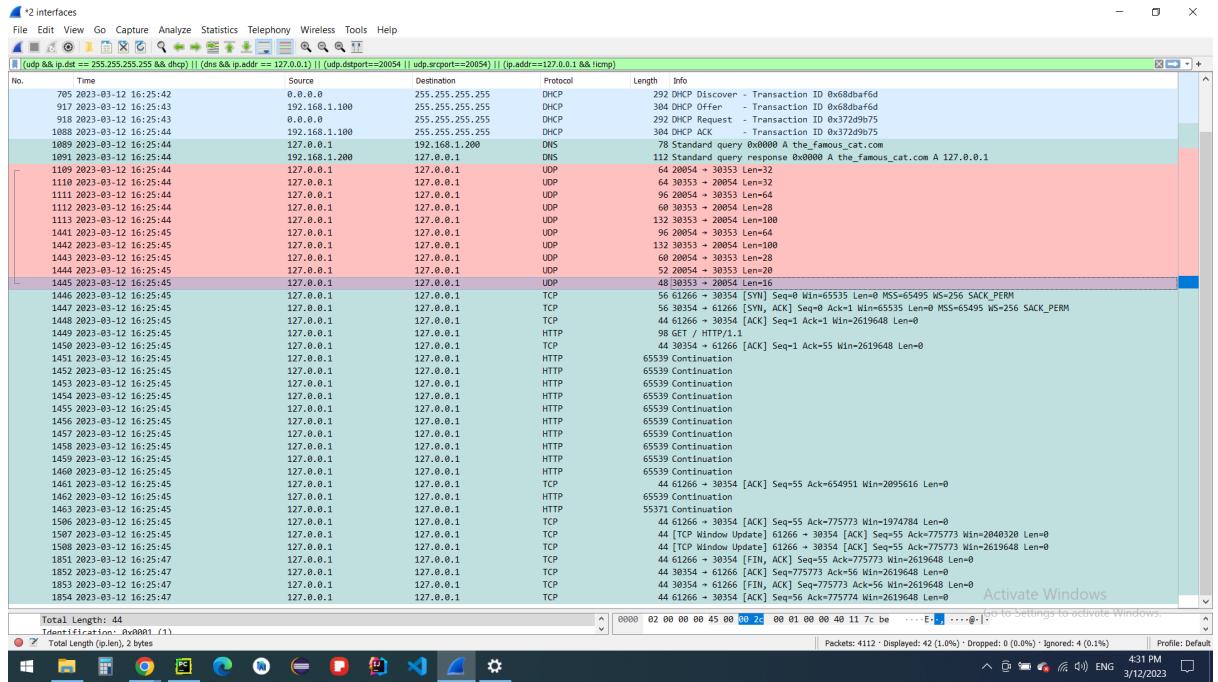
```

def start_server():
    # change parameters to print more or less data, and to change packet loss,
    # default packet loss is -1 which means no packet loss
    server_socket = SCTPSocket(packet_size=1024, pkt_printer=True, cc_printer=True, packet_loss=0.1)
    server_socket.bind(('localhost', server_port))
    print(f'Server started and listening on server_port {server_port}...')
    server_socket.listen()

```

הרצת כל השרתים בקובץ : DNS, DHCP, multi-server, app_server :
הרצת וירשארק ברקע לפי ה필טר המתואר הנ"ל
לאחר הרצת הקליינט הלקוח יראה לנו את החתולה הרנדומלית ושומר אותה כקובץ בתיקייה.





בהריצה שעשינו אנחנו יכולים לראות את התקשרות בין הלוקו לשרת האפליקציה, ובין הלוקו לשרת המכיל את התמונה ששרת האפליקציה הפנה את הלוקו אליו. הפורטים רלוונטיים:

30353 - פורט עליון יושב שרת האפליקציה

20054 - פורט הלוקו

30354 - פורט השרת שמכיל את התמונה (יכול להיות גם 30355 או 30356) חבילות מס' 705, 918, 917, 1089, 1088, 1109, 1113, 1441-1445 הן של התקשרות בין הקליינט ושרת DNS ושרת DHCP לפירוט והסבירים על כך יש לעבור לחלק השביעי.

חbillות 1441-1445 הם השימוש בין הקליינט לשרת אפליקציית REDIRECT, כאשר השיחה מתבצעת על UDP, בוירשאך זאת נראה על UDP שcn אנחנו יצרנו את והוא איננו מוכר לוירשאך.

כל שאר החבילות הירוקות למטה הם השימוש בין הקליינט לשרת התמונה הרנדומלי, כאשר השימוש מתבצעת על TCP, לפירוט והסבירים יש לעבור להרצת 0% אובדן חבילות.

ניתוח:

אנו רואים בהתחלה את לחיצת הידיים הראשונית על ידי חבילת INIT או INIT ACK בטרמינל ובהתאם להקלטת הווירשארק חבילה מס' 1109 ו- 1110, ואז את הבקשת HTTP GET נשלחת לשרת האפליקציה בטרמינל חבילה מס' 1111, אחר כך את ה ack לבקשת ה GET, בטרמינל אותו ds בווירשארק חבילה מס' 1112.

לאחר מכן נשלחה צרצה HTTP REDIRECT משרת אפליקציה עם הכתובת של השרת הרנדומלי שמכיל את התמונה, פה ספציפית קיבלנו את שרת התמונה עם פורט 30354. נשים לב שבטרמינל הקליינט התרחש אובדן חבילה מס' 24530, ככלומר בקשה ה redirect הולכה לאיבוד.

```

File Edit View Navigate Code Refactor Run Tools Git Window Help NetworkProject - app_server.py
NetworkProject > local_servers_and_client-udp app_server.py
Project Run client app_server.py SCTPSocket.py client.py
Run Full Requests Project multi-server_sock... client
Full Requests Project multi-server_sock... client
Server started and listening on server_port 30353...
started threadWaiting for connection from client...
packet loss! tsn - 644481
packet loss! tsn - 644482
###[ SCTPChunkInit ]###
    type      = init
    flags     = 0x0
    len       = 20
    init_tag  = 0x73e4c
    a_rwnd   = 50
    n_out_streams= 1
    n_in_streams= 1
    init_tsn = 0x5fd0
    \params   \
Request received from ('127.0.0.1', 20054)
started receiving
waiting for data tsn=24529 ptsn=24529
resent last packets
-----recieved-----
type-0 tsn=24529 seq-0 ds-1
-----sent-----
type-3 tsn=24529 seq-0 a_rwnd=50
Returning response: HTTP/1.1 307 Temporary Redirect
Location: http://localhost:30354
DNS
Process finished with exit code -1
Git Run TODO Problems Debug Python Packages Terminal Python Console
Activate Windows
Go to Settings to activate Windows.
28:47 CRLF UTF-8 4 spaces Python 3.10 (NetworkProject) main
427 PM 3/12/2023
No. Time Source Destination Protocol Length Info
1109 2023-03-12 16:25:44 127.0.0.1 127.0.0.1 UDP 64 20054 → 30353 Len=32
1110 2023-03-12 16:25:44 127.0.0.1 127.0.0.1 UDP 64 30353 → 20054 Len=32
1111 2023-03-12 16:25:44 127.0.0.1 127.0.0.1 UDP 96 20054 → 30353 Len=64
1112 2023-03-12 16:25:44 127.0.0.1 127.0.0.1 UDP 60 30353 → 20054 Len=28
1113 2023-03-12 16:25:44 127.0.0.1 127.0.0.1 UDP 132 30353 → 20054 Len=100
1441 2023-03-12 16:25:45 127.0.0.1 127.0.0.1 UDP 96 20054 → 30353 Len=64
1442 2023-03-12 16:25:45 127.0.0.1 127.0.0.1 UDP 132 30353 → 20054 Len=100
1443 2023-03-12 16:25:45 127.0.0.1 127.0.0.1 UDP 60 20054 → 30353 Len=28
1444 2023-03-12 16:25:45 127.0.0.1 127.0.0.1 UDP 52 20054 → 30353 Len=20
1445 2023-03-12 16:25:45 127.0.0.1 127.0.0.1 UDP 48 30353 → 20054 Len=16
1446 2023-03-12 16:25:45 127.0.0.1 127.0.0.1 TCP 56 61266 → 30354 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 S...

```

את תשובה ה-HTTP REDIRECT נשלחת לשרת האפליקציה בטרמינל 24530 = tsn בוירשאرك חビלה מס' 1113, מכיוון שהלכה לאיבוד, השרת שמחכה ל-ack לאחר זמן מה ישלח מחדש את ה-HTTP REDIRECT בטרמינל 24530 = tsn בוירשאرك חビלה מס' 1442, באותו זמן הקלינט אשר שמחה לקבלת מידע בפועל recv, לאחר זמן מה ישלח מחדש את הפקטה האחרון, שהרי היא בבקשת GET HTTP בטרמינל 24529 = tsn בוירשאرك חビלה מס' 1441, ניתן לראות שגם חビלה זו אבדה בטרמינל של שרת האפליקציה.

ניתן לראות בהמשך שהקלינט כן קיבל את ה-HTTP REDIRECT השני שנשלח, ולכן שולח חוזה את ה-ack לתשובה ה-HTTP REDIRECT, בטרמינל אותו tsn בוירשאرك חビלה מס' 1443.

לעוד הסבר ופירוט לגבי השילוח מחדש או רשות מתבצע יש לעבור לחילק 6 ל-Reliability.

לבסוף אנו רואים את חבילות SHUTDOWN ACK ו-SHUTDOWN בטרמינל ובהתאמה בהקלטת הוירשאرك חビלות מס' 1444 ו-1445, אשר מסמלות על סיום חיבור בין הלקוח לשרת. לאחר שלשרת יש הכתובת הוא יפנה אליה כדי לקבל את הקובץ, لكن נוצר חיבור חדש בין הלקוח לשרת שקיבל מהשרת אפליקציה והלקוח שולח שוב בבקשת GET.

```

File Edit View Navigate Code Refactor Run Tools Git Window Help NetworkProject - app_server.py
NetworkProject > local_servers_and_client-rudp app_server.py SCTPSocket.py client.py
Project Run Full Requests Project Structure Bookmarks DNS
app_server x multi-server_sock... client x
to ('127.0.0.1', 20054)
-----sent-----
type=0 tsn=24530 seq=0 ds=1
waiting for ack seq=0 cwnd=1 tsn=24530 ptsn=24530
w_min=1.0000 cwnd=1.00 w_max=0.00 jump=1.00 isSlowStart=True
packet loss! tsn - 24529
resented last packets
-----received-----
type=3 tsn=24530 a_rwnd=50
ended sending
###[ SCTPChunkShutdown ]##
    type = shutdown
    flags = 0x0
    len = 8
    cumul_tsn_ack= 0x5fd3

session summary:
received packets: dict_keys([24528, 24529, 24530, 24531])
sent packets: dict_keys([24529, 24530])
in flight packets: dict_keys([644462])
acknowledged packets: dict_keys([24530])
ended threaddisconnected from client

Server started and listening on server_port 30353...
started thread
waiting for connection from client...

Process finished with exit code -1

```

```

C:\Users\...
starting :
ip is loc ip is loc ip is loc
connected
received:
connected
received:
Process f:
DHCP x
Assigning
###[ Ether
dst
src
type
###[ IP ];
vers:
ihl
tos
len
id
flag:
frag
ttl
packet loss! tsn - 24530
-----received-----
type=3 tsn=24529 a_rwnd=50
waiting for data tsn=24530 ptsn=24529
resented last packets
-----received-----
type=0 tsn=24530 seq=0 ds=1
-----sent-----
type=3 tsn=24530 seq=0 a_rwnd=50
HTTP/1.1 307 Temporary Redirect
Location: http://localhost:30354

waiting for ack tsn=24531 ptsn=24531
ended thread
###[ SCTPChunkShutdownAck ]##
    type = shutdown-ack
    flags = 0x0
    len = 4

session summary:
received packets: dict_keys([24528, 24529, 24530, 0, 24531])
sent packets: dict_keys([24528, 24529, 24530, 24531])
in flight packets: dict_keys({})
acknowledged packets: dict_keys([24528, 24529, 24531])
('localhost', '30354')
got 775772 bytes

Process finished with exit code -1

```

No.	Time	Source	Destination	Protocol	Length	Info
1109	2023-03-12 16:25:44	127.0.0.1	127.0.0.1	UDP	64	20054 → 30353 Len=32
1110	2023-03-12 16:25:44	127.0.0.1	127.0.0.1	UDP	64	30353 → 20054 Len=32
1111	2023-03-12 16:25:44	127.0.0.1	127.0.0.1	UDP	96	20054 → 30353 Len=64
1112	2023-03-12 16:25:44	127.0.0.1	127.0.0.1	UDP	60	30353 → 20054 Len=28
1113	2023-03-12 16:25:44	127.0.0.1	127.0.0.1	UDP	132	30353 → 20054 Len=100
1441	2023-03-12 16:25:45	127.0.0.1	127.0.0.1	UDP	96	20054 → 30353 Len=64
1442	2023-03-12 16:25:45	127.0.0.1	127.0.0.1	UDP	132	30353 → 20054 Len=100
1443	2023-03-12 16:25:45	127.0.0.1	127.0.0.1	UDP	60	20054 → 30353 Len=28
1444	2023-03-12 16:25:45	127.0.0.1	127.0.0.1	UDP	52	20054 → 30353 Len=20
1445	2023-03-12 16:25:45	127.0.0.1	127.0.0.1	UDP	48	30353 → 20054 Len=16
1446	2023-03-12 16:25:45	127.0.0.1	127.0.0.1	TCP	56	61266 → 30354 [SYN Seq=0 Win=65535 Len=0 MSS=65495 WS=256 S...

30% אובדן חבילות

** קובץ הקלטה נקרא **rudp_30.pcapng**
ראשית נודע שהפרטטים של הסוקט מוגדרים היבש
בקליינט נגדיר שיהי 30% איבוד פקודות, ונגדיר **pkt_printer** ל **cc_printer** כך שיודף
מידע רלוונטי

```

matro68 *
def send_request(server_address, request):
    # change parameters to print more or less data, and to change packet loss,
    # default packet loss is -1 which means no packet loss
    client_socket = SCTPSocket(packet_size=1024, pkt_printer=True, cc_printer=True, packet_loss=0.3)

```

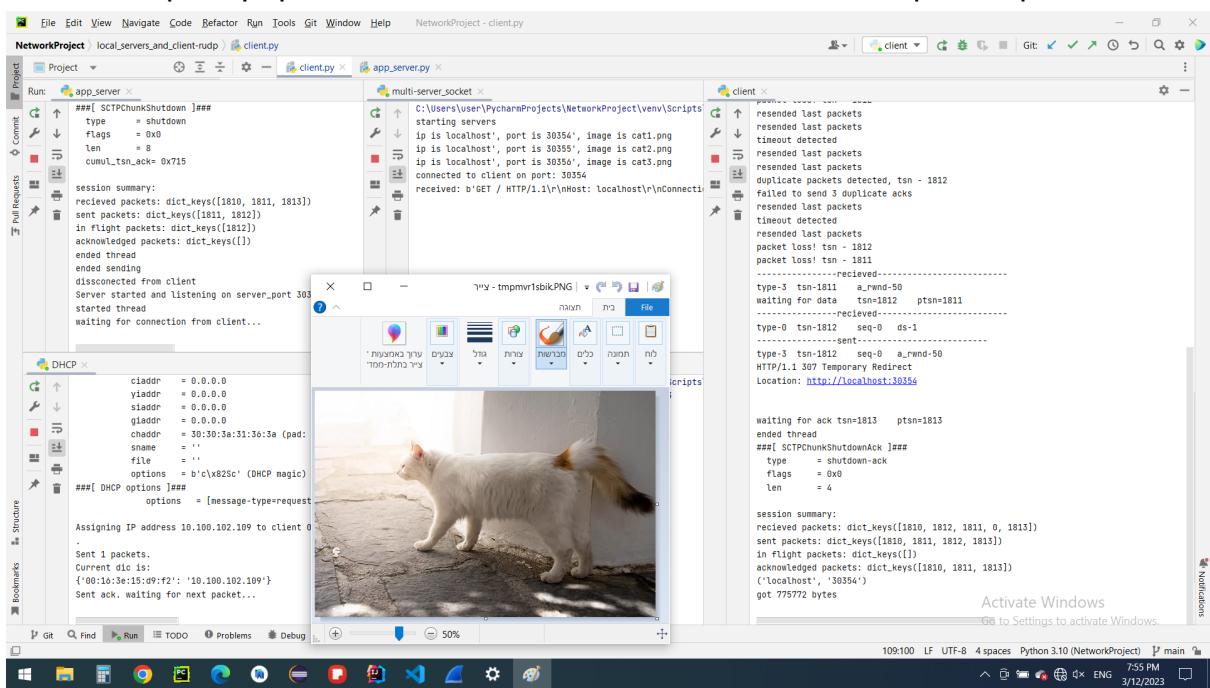
בשרת האפליקציה נגדיר באופן זהה

```

# Define a function to start the server and listen for incoming connections
matro68 *
def start_server():
    # change parameters to print more or less data, and to change packet loss,
    # default packet loss is -1 which means no packet loss
    server_socket = SCTPSocket(packet_size=1024, pkt_printer=True, cc_printer=True, packet_loss=0.3)
    server_socket.bind(('localhost', server_port))
    print(f'Server started and listening on server_port {server_port}...')
    server_socket.listen()
print("waiting for connection from client...")

```

הרצה כל השרתים בקובץ :
והרצה וירשארק ברקע לפי ה필טר המתואר להלן
לאחר הרצת הקליינט הקוד יראה לנו את החתולה הרנדומלית ושומר אותה כקובץ בתיקייה.



הקלטת וירשארק

No.	Time	Source	Destination	Protocol	Length	Info	Information
2	2023-03-12 19:55:06	0.0.0.0	255.255.255.255	DHCP	292	DHCP Discover - Transaction ID 0x14b15fef	
3	2023-03-12 19:55:07	192.168.1.100	255.255.255.255	DHCP	304	DHCP Offer - Transaction ID 0x14b15fef	
4	2023-03-12 19:55:07	0.0.0.0	255.255.255.255	DHCP	292	DHCP Request - Transaction ID 0x1561b927	
6	2023-03-12 19:55:08	192.168.1.100	255.255.255.255	DHCP	304	DHCP ACK - Transaction ID 0x1561b927	
8	2023-03-12 19:55:08	127.0.0.1	192.168.1.200	DNS	78	Standard query 0x0000 A the_famous_cat.com	
9	2023-03-12 19:55:08	192.168.1.200	127.0.0.1	DNS	112	Standard query response 0x0000 A the_famous_cat.com A 127.0.0.1	
5	2023-03-12 19:55:08	127.0.0.1	127.0.0.1	UDP	64	20054 → 30353 Len=32	
7	2023-03-12 19:55:08	127.0.0.1	127.0.0.1	UDP	64	30353 → 20054 Len=32	
10	2023-03-12 19:55:08	127.0.0.1	127.0.0.1	UDP	96	20054 → 30353 Len=64	
11	2023-03-12 19:55:08	127.0.0.1	127.0.0.1	UDP	60	30353 → 20054 Len=28	
12	2023-03-12 19:55:08	127.0.0.1	127.0.0.1	UDP	132	30353 → 20054 Len=100	
14	2023-03-12 19:55:10	127.0.0.1	127.0.0.1	UDP	96	20054 → 30353 Len=64	
15	2023-03-12 19:55:10	127.0.0.1	127.0.0.1	UDP	132	30353 → 20054 Len=100	
16	2023-03-12 19:55:11	127.0.0.1	127.0.0.1	UDP	96	20054 → 30353 Len=64	
17	2023-03-12 19:55:11	127.0.0.1	127.0.0.1	UDP	132	30353 → 20054 Len=100	
18	2023-03-12 19:55:13	127.0.0.1	127.0.0.1	UDP	96	20054 → 30353 Len=64	
19	2023-03-12 19:55:13	127.0.0.1	127.0.0.1	UDP	132	30353 → 20054 Len=100	
20	2023-03-12 19:55:14	127.0.0.1	127.0.0.1	UDP	96	20054 → 30353 Len=64	
21	2023-03-12 19:55:14	127.0.0.1	127.0.0.1	UDP	132	30353 → 20054 Len=100	
22	2023-03-12 19:55:16	127.0.0.1	127.0.0.1	UDP	96	20054 → 30353 Len=64	
23	2023-03-12 19:55:16	127.0.0.1	127.0.0.1	UDP	132	30353 → 20054 Len=100	
24	2023-03-12 19:55:17	127.0.0.1	127.0.0.1	UDP	96	20054 → 30353 Len=64	
25	2023-03-12 19:55:17	127.0.0.1	127.0.0.1	UDP	132	30353 → 20054 Len=100	
26	2023-03-12 19:55:19	127.0.0.1	127.0.0.1	UDP	96	20054 → 30353 Len=64	
27	2023-03-12 19:55:19	127.0.0.1	127.0.0.1	UDP	132	30353 → 20054 Len=100	
28	2023-03-12 19:55:19	127.0.0.1	127.0.0.1	UDP	60	30353 → 20054 Len=28	
29	2023-03-12 19:55:19	127.0.0.1	127.0.0.1	UDP	60	30353 → 20054 Len=28	
30	2023-03-12 19:55:19	127.0.0.1	127.0.0.1	UDP	60	20054 → 30353 Len=28	
31	2023-03-12 19:55:19	127.0.0.1	127.0.0.1	UDP	52	20054 → 30353 Len=20	
32	2023-03-12 19:55:19	127.0.0.1	127.0.0.1	UDP	60	30353 → 20054 Len=28	
33	2023-03-12 19:55:19	127.0.0.1	127.0.0.1	UDP	48	30353 → 20054 Len=16	
36	2023-03-12 19:55:21	127.0.0.1	127.0.0.1	TCP	56	49504 → 30354 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM	
37	2023-03-12 19:55:21	127.0.0.1	127.0.0.1	TCP	56	30354 → 49504 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM	
38	2023-03-12 19:55:21	127.0.0.1	127.0.0.1	TCP	44	49504 → 30354 [ACK] Seq=1 Ack=1 Win=2619648 Len=0	
39	2023-03-12 19:55:21	127.0.0.1	127.0.0.1	HTTP	98	GET / HTTP/1.1	
40	2023-03-12 19:55:21	127.0.0.1	127.0.0.1	TCP	44	30354 → 49504 [ACK] Seq=1 Ack=55 Win=2619648 Len=0	
41	2023-03-12 19:55:21	127.0.0.1	127.0.0.1	HTTP	65539 Continuation		

בהרצת שעשינו אנחנו יכולים לראות את התקשרות בין הלקוח לשרת האפליקציה, ובין הלקוח לשרת המכיל את התמונה ששרת האפליקציה הפנה את הלקוח אליו. הפורטים רלוונטיים:

30353 - פורט עליון יושב שרת האפליקציה

20054 - פורט הלקוח

30356 - פורט השרת שמכיל את התמונה (יכול להיות גם 30355 או 30354)

ה宦יות בצעע כחול וירוק בהתחלה הן של התקשרות בין הלקוח לשרת DNS ושרת DHCP לפחות והסבירים על כך יש לעבור לחלק השביעי.

ה宦יות האדומות הן השיחה בין הקליינט לשרת אפליקציית REDIRECT, כאשר השיחה מתבצעת על UDP, RUDP, בוירשארק זאת נראה על UDP שכן יצרנו את RUDP והוא אכן מוכר לוירשארק.

ה宦יות הירוקות בסוף ההקלטה הן השיחה בין הקליינט לשרת התמונה הרנדומלי, כאשר השיחה מתבצעת על TCP. המשך הקלטת וירשארק:

No.	Time	Source	Destination	Protocol	Length	Info	Information
22	2023-03-12 19:55:16	127.0.0.1	127.0.0.1	UDP	96	20054 → 30353 Len=64	
23	2023-03-12 19:55:16	127.0.0.1	127.0.0.1	UDP	132	30353 → 20054 Len=100	
24	2023-03-12 19:55:17	127.0.0.1	127.0.0.1	UDP	96	20054 → 30353 Len=64	
25	2023-03-12 19:55:17	127.0.0.1	127.0.0.1	UDP	132	30353 → 20054 Len=100	
26	2023-03-12 19:55:19	127.0.0.1	127.0.0.1	UDP	96	20054 → 30353 Len=64	
27	2023-03-12 19:55:19	127.0.0.1	127.0.0.1	UDP	132	30353 → 20054 Len=100	
28	2023-03-12 19:55:19	127.0.0.1	127.0.0.1	UDP	60	30353 → 20054 Len=28	
29	2023-03-12 19:55:19	127.0.0.1	127.0.0.1	UDP	60	30353 → 20054 Len=28	
30	2023-03-12 19:55:19	127.0.0.1	127.0.0.1	UDP	60	20054 → 30353 Len=28	
31	2023-03-12 19:55:19	127.0.0.1	127.0.0.1	UDP	52	20054 → 30353 Len=20	
32	2023-03-12 19:55:19	127.0.0.1	127.0.0.1	UDP	60	30353 → 20054 Len=28	
33	2023-03-12 19:55:19	127.0.0.1	127.0.0.1	UDP	48	30353 → 20054 Len=16	
36	2023-03-12 19:55:21	127.0.0.1	127.0.0.1	TCP	56	49504 → 30354 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM	
37	2023-03-12 19:55:21	127.0.0.1	127.0.0.1	TCP	56	30354 → 49504 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM	
38	2023-03-12 19:55:21	127.0.0.1	127.0.0.1	TCP	44	49504 → 30354 [ACK] Seq=1 Ack=1 Win=2619648 Len=0	
39	2023-03-12 19:55:21	127.0.0.1	127.0.0.1	HTTP	98	GET / HTTP/1.1	
40	2023-03-12 19:55:21	127.0.0.1	127.0.0.1	TCP	44	30354 → 49504 [ACK] Seq=1 Ack=55 Win=2619648 Len=0	
41	2023-03-12 19:55:21	127.0.0.1	127.0.0.1	HTTP	65539 Continuation		
42	2023-03-12 19:55:21	127.0.0.1	127.0.0.1	HTTP	65539 Continuation		
43	2023-03-12 19:55:21	127.0.0.1	127.0.0.1	HTTP	65539 Continuation		
44	2023-03-12 19:55:21	127.0.0.1	127.0.0.1	HTTP	65539 Continuation		
45	2023-03-12 19:55:21	127.0.0.1	127.0.0.1	HTTP	65539 Continuation		
46	2023-03-12 19:55:21	127.0.0.1	127.0.0.1	HTTP	65539 Continuation		
47	2023-03-12 19:55:21	127.0.0.1	127.0.0.1	HTTP	65539 Continuation		
48	2023-03-12 19:55:21	127.0.0.1	127.0.0.1	HTTP	65539 Continuation		
49	2023-03-12 19:55:21	127.0.0.1	127.0.0.1	HTTP	65539 Continuation		
50	2023-03-12 19:55:21	127.0.0.1	127.0.0.1	HTTP	65539 Continuation		
51	2023-03-12 19:55:21	127.0.0.1	127.0.0.1	TCP	44	49504 → 30354 [ACK] Seq=55 Ack=654951 Win=2095616 Len=0	
52	2023-03-12 19:55:21	127.0.0.1	127.0.0.1	HTTP	65539 Continuation		
53	2023-03-12 19:55:21	127.0.0.1	127.0.0.1	HTTP	55371 Continuation		
54	2023-03-12 19:55:21	127.0.0.1	127.0.0.1	TCP	44	49504 → 30354 [ACK] Seq=55 Ack=775773 Win=1974784 Len=0	
55	2023-03-12 19:55:21	127.0.0.1	127.0.0.1	TCP	44	[TCP Window Update] 49504 → 30354 [ACK] Seq=55 Ack=775773 Win=2040320 Len=0	
56	2023-03-12 19:55:21	127.0.0.1	127.0.0.1	TCP	44	[TCP Window Update] 49504 → 30354 [ACK] Seq=55 Ack=775773 Win=2619648 Len=0	
57	2023-03-12 19:55:23	127.0.0.1	127.0.0.1	TCP	44	49504 → 30354 [FIN, ACK] Seq=55 Ack=775773 Win=2619648 Len=0	
58	2023-03-12 19:55:23	127.0.0.1	127.0.0.1	TCP	44	30354 → 49504 [ACK] Seq=55 Ack=775773 Win=2619648 Len=0	
59	2023-03-12 19:55:23	127.0.0.1	127.0.0.1	TCP	44	49504 → 30354 [FIN, ACK] Seq=56 Ack=56 Win=2619648 Len=0	
60	2023-03-12 19:55:23	127.0.0.1	127.0.0.1	TCP	44	49504 → 30354 [ACK] Seq=56 Ack=775774 Win=2619648 Len=0	

ניתוח

ניתן לראות מה שראינו בניתוחים הקודמים, אך בנוסף לכך נשים לב שבעקבות כך שהעלונו את אחוז אובדן החבילות, הודעות בטרמינל על אובדן חבילה למשל tsn - 1811 - packet loss! מופיעות בתדריות גבוהה יותר, ופעולות ההतאוששות של הלקוח והשרת אפליקציה של מנגןו ה Reliability resend last packets מופיעות גם הם בתדריות גבוהה יותר.

The screenshot shows the PyCharm IDE interface. On the left, the project structure for 'NetworkProject' is visible, containing 'app_server.py' and 'client.py'. The 'client.py' file is open, showing code for a client socket. The terminal window on the right displays the execution of the client and server scripts, along with their logs. The logs show the exchange of SCTP messages, including INIT, INIT-ACK, and data transmission, followed by a temporary redirect response. The terminal also shows periodic timeout and retransmission messages from the server.

```

File Edit View Navigate Code Refactor Run Tools Git Window Help NetworkProject - client.py
NetworkProject > local_servers_and_client-rudp client.py app_server.py
Project Run: app_server
C:\Users\user\PycharmProjects\NetworkProject\venv\Scripts\python.exe C:\Users\user\PycharmProjects\NetworkProject\local_servers_and_client-rudp\client.py
Server started and listening on server_port 30353...
started thread
waiting for connection from client...
###[ SCTPChunkInit ]###
    type      = init
    flags     = 0x0
    len       = 20
    init_tag  = 0x58401
    a_rwnd   = 50
    n_out_streams= 1
    n_in_streams= 1
    init_tsn = 0x712
    \params   \
Request received from ('127.0.0.1', 20054)
started receiving
waiting for data tsn=1811 ptsn=1811
resent last packets
-----recieved-----
type-0 tsn=1811 seq=0 ds=1
-----sent-----
type-3 tsn=1811 seq=0 a_rwnd=50
Returning response: HTTP/1.1 307 Temporary Redirect
Location: http://localhost:30353/
-----sent-----
options = [message-type=request requested_addr=10.100.102.109 end
DNS
#HWI-UNCPD options
Activate Windows
Go to Settings to activate Windows.
10:100 LF UTF-8 4 spaces Python 3.10 (NetworkProject) main
Find Run TODO Problems Debug Python Packages Terminal Python Console
10:100 LF UTF-8 4 spaces Python 3.10 (NetworkProject) main
7:58 PM 3/12/2023

```

ניתן לראות גם בהקלת הווירשארק את הכמות הגדולה יותר של חבילות שנשלחו בעקבות אובדן החבילות שהה.

The screenshot shows the Wireshark network traffic analysis tool. The packet list pane displays a large volume of captured data, primarily UDP and TCP traffic. The traffic is heavily concentrated between two hosts, with many UDP packets being exchanged. The packet details and bytes panes provide a detailed view of the captured data, showing the structure of the messages. The status bar at the bottom indicates the total number of packets (70), displayed packets (56), dropped packets (0), and the current time (8:01 PM).

No.	Time	Source	Destination	Protocol	Length	Info
8	2023-03-12 19:55:08	127.0.0.1	192.168.1.200	DNS	78	Standard query 0x0000 A the_famous_cat.com
9	2023-03-12 19:55:08	192.168.1.200	127.0.0.1	DNS	112	Standard query response 0x0000 A the_famous_cat.com A 127
5	2023-03-12 19:55:08	127.0.0.1	127.0.0.1	UDP	64	20054 → 30353 Len=32
7	2023-03-12 19:55:08	127.0.0.1	127.0.0.1	UDP	64	30353 → 20054 Len=32
10	2023-03-12 19:55:08	127.0.0.1	127.0.0.1	UDP	96	20054 → 30353 Len=64
11	2023-03-12 19:55:08	127.0.0.1	127.0.0.1	UDP	60	30353 → 20054 Len=28
12	2023-03-12 19:55:08	127.0.0.1	127.0.0.1	UDP	132	30353 → 20054 Len=100
14	2023-03-12 19:55:10	127.0.0.1	127.0.0.1	UDP	96	20054 → 30353 Len=64
15	2023-03-12 19:55:10	127.0.0.1	127.0.0.1	UDP	132	30353 → 20054 Len=100
16	2023-03-12 19:55:11	127.0.0.1	127.0.0.1	UDP	96	20054 → 30353 Len=64
17	2023-03-12 19:55:11	127.0.0.1	127.0.0.1	UDP	132	30353 → 20054 Len=100
18	2023-03-12 19:55:13	127.0.0.1	127.0.0.1	UDP	96	20054 → 30353 Len=64
19	2023-03-12 19:55:13	127.0.0.1	127.0.0.1	UDP	132	30353 → 20054 Len=100
20	2023-03-12 19:55:14	127.0.0.1	127.0.0.1	UDP	96	20054 → 30353 Len=64
21	2023-03-12 19:55:14	127.0.0.1	127.0.0.1	UDP	132	30353 → 20054 Len=100
22	2023-03-12 19:55:16	127.0.0.1	127.0.0.1	UDP	96	20054 → 30353 Len=64
23	2023-03-12 19:55:16	127.0.0.1	127.0.0.1	UDP	132	30353 → 20054 Len=100
24	2023-03-12 19:55:17	127.0.0.1	127.0.0.1	UDP	96	20054 → 30353 Len=64
25	2023-03-12 19:55:17	127.0.0.1	127.0.0.1	UDP	132	30353 → 20054 Len=100
26	2023-03-12 19:55:19	127.0.0.1	127.0.0.1	UDP	96	20054 → 30353 Len=64
27	2023-03-12 19:55:19	127.0.0.1	127.0.0.1	UDP	132	30353 → 20054 Len=100
28	2023-03-12 19:55:19	127.0.0.1	127.0.0.1	UDP	60	30353 → 20054 Len=28
29	2023-03-12 19:55:19	127.0.0.1	127.0.0.1	UDP	60	30353 → 20054 Len=28
30	2023-03-12 19:55:19	127.0.0.1	127.0.0.1	UDP	60	20054 → 30353 Len=28
31	2023-03-12 19:55:19	127.0.0.1	127.0.0.1	UDP	52	20054 → 30353 Len=20
32	2023-03-12 19:55:19	127.0.0.1	127.0.0.1	UDP	60	30353 → 20054 Len=28
33	2023-03-12 19:55:19	127.0.0.1	127.0.0.1	UDP	48	30353 → 20054 Len=16
36	2023-03-12 19:55:21	127.0.0.1	127.0.0.1	TCP	56	49504 → 30354 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256
37	2023-03-12 19:55:21	127.0.0.1	127.0.0.1	TCP	56	30354 → 49504 [SYN ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=256

בנוסף לכך אנו נחשפים לפועלות התואששות של מנגנון ה Reliability שהיא פועלה זו קורת כאשר מזוהה כפילות חריגה של חבילה שהתקבלה, מפורט על כך נרחבות בחלק 6 | Reliability

```

File Edit View Navigate Code Refactor Run Tools Git Window Help NetworkProject - client.py
NetworkProject > local_servers_and_client-rudp > client.py
Project Run: app_server
Run: app_server
-----sent-----
type=0 tsn=1812 seq=0 ds=1
waiting for ack seq=0 cwnd=1 tsn=1812 ptsn=1812
w_min=1.0000 cwnd=1.00 w_max=0.00 jump=1.00 isSlowStart=True
resended last packets
packet loss! tsn - 1811
resended last packets
packet loss! tsn - 1811
resended last packets
timeout detected
packet loss! tsn - 1811
resended last packets
packet loss! tsn - 1811
resended last packets
resended last packets
timeout detected
resended last packets
duplicate packets detected, tsn - 1811
sent 3 duplicate acks
packet loss! tsn - 1812
###[ SCTPChunkShutdown ]###
    type      = shutdown
    flags     = 0x0
    len       = 8
cumul_tsn_ack= 0x715

-----recieved-----
type=3 tsn=1811 a_rwnd=50
waiting for data tsn=1812 ptsn=1811
-----recieved-----
type=0 tsn=1812 seq=0 ds=1
-----sent-----
type=3 tsn=1812 seq=0 a_rwnd=50
HTTP/1.1 307 Temporary Redirect
Location: http://localhost:30354

```

לאחר זמן מה, המערכת התגברה על אובדן החבילות והצליחה להעביר את כל המידע הנדרש מהклиינט לשרת האפליקציה וההפר.

```

File Edit View Navigate Code Refactor Run Tools Git Window Help NetworkProject - client.py
NetworkProject > local_servers_and_client-rudp > client.py
Project Run: app_server
Run: app_server
-----recieved-----
type=0 tsn=1812 seq=0 ds=1
-----sent-----
type=3 tsn=1812 seq=0 a_rwnd=50
HTTP/1.1 307 Temporary Redirect
Location: http://localhost:30354

-----recieved-----
waiting for ack tsn=1813 ptsn=1813
ended thread
###[ SCTPChunkShutdownAck ]###
    type      = shutdown-ack
    flags     = 0x0
    len       = 4

-----recieved-----
recieved packets: dict_keys([1810, 1811, 1813])
sent packets: dict_keys([1811, 1812])
in flight packets: dict_keys([1812])
acknowledged packets: dict_keys([])
ended thread
ended sending
disconected from client
Server started and listening on server_port 30353...
started thread
waiting for connection from client...

-----recieved-----
recieved packets: dict_keys([1810, 1812, 1811, 0, 1813])
sent packets: dict_keys([1810, 1811, 1812, 1813])
in flight packets: dict_keys([])
acknowledged packets: dict_keys([1810, 1811, 1813])
('localhost', '30354')
got 775772 bytes
saved image to current directory
Done.

Process finished with exit code 0

```

ביבליוגרפיה:

1. <https://jcutrer.com/python/scapy-dhcp-listener>: used with build dhcp server
2. <https://stackoverflow.com/questions/41557596/scapy-dhcp-request-with-custom-options-for-dhcp-server>
3. <https://thepacketgeek.com/scapy/building-network-tools/part-09/>: used with dns server
4. <https://www.vsolcn.com/blogs-detail/what-is-dhcp> : info about dhcp servers
5. https://www.youtube.com/watch?v=DeFST8tvTul&ab_channel=NeuralNine : help building simple http server using HTTPSserver library
6. <https://medium.com/@lee5187415/concepts-you-should-know-about-large-system-design-c0a823c33a96> : load balancer info
7. <https://www.inbalsoft.com/newchange/teor.htm> : תיאור תוכנה של מערכת
8. <https://scapy.readthedocs.io/en/latest/api/scapy.layers.sctp.html> תיאור מבנה SCTP
9. <https://www.geeksforgeeks.org/flow-control-in-data-link-layer/> flow control
10. <https://www.cs.princeton.edu/courses/archive/fall16/cos561/papers/Cubic08.pdf> cubic algorithm
11. <https://www.scitepress.org/Papers/2019/79162/79162.pdf> cubic algorithm
12. <https://pandorafms.com/blog/tcp-congestion-control/> cubic algorithm
13. <https://xenodium.com/generating-a-random-mac-address/> how to create random MAC generator