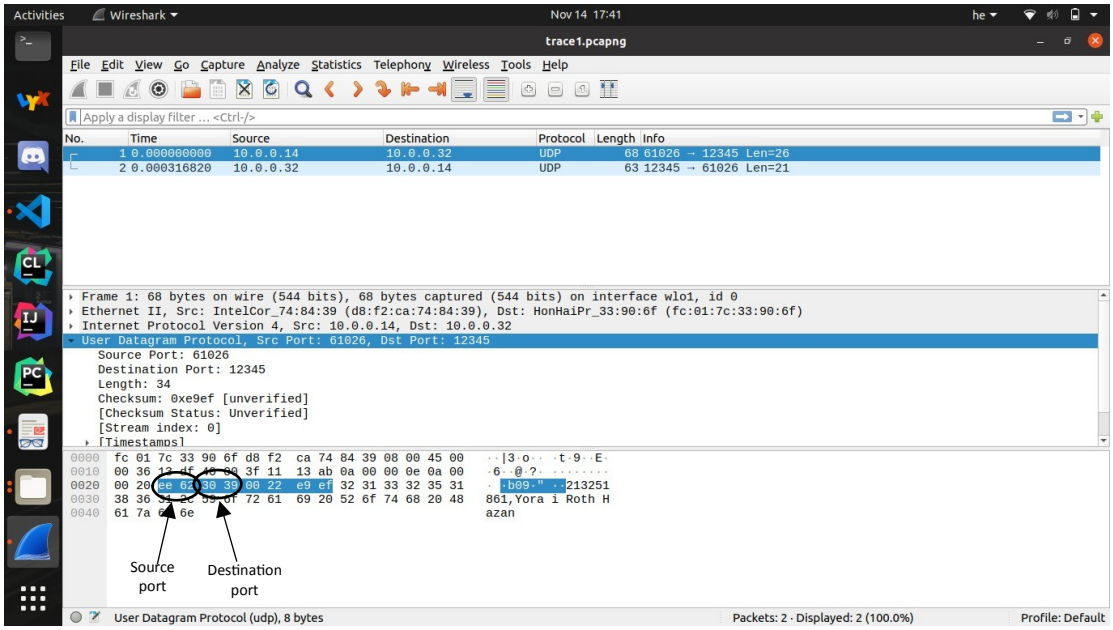
יוראי רוט חזן ואורי דאבוש – רשתות תקשורת 1

חלק א

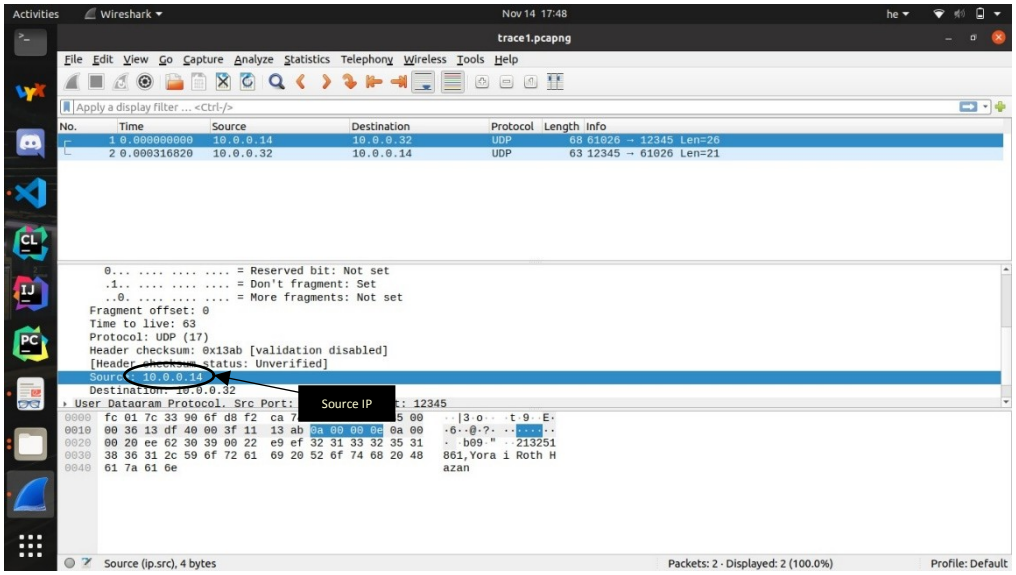
2) סיננו את החבילות בעזרת כתיבת בשורת הסינון ip.addr == ל-Ip של המחשבים שלנו שגילינו בעזרת ifconfig (לינוקס).

3) יש שימוש במספרי פורט בקבצי השרת בפעולה bind שבה השרת כורך את עצמו לפורט הנ"ל על מנת שהלקוח ידע לאן לשלוח את הודעותיו ובקובץ הלקוח בפעולה sendto הלקוח שולח לפורט הרשום בו את ההודעה. השימוש בפורט הוא על מנת שהלקוח ידע היכן השרת יושב והשרת משתמש בפורט של הלקוח על מנת להחזיר לו תשובה. כל זה מתבצע בשכבת התעבורה (Transport) וניתן לראות זאת בתמונה הבאה: (סימנו את הheader של שכבת הטרנספורט וניתן לראות שהפורטים נמצאים שם)

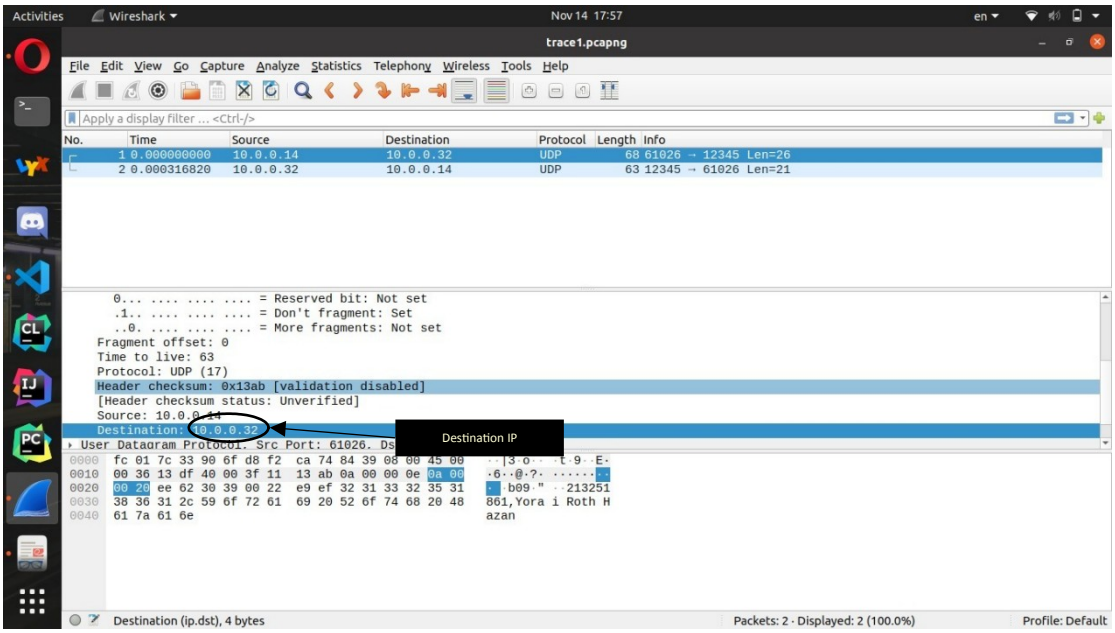


4) כתובות ה-IP ב-wireshark:

כתובת המקור (מסומנת בחבילה עצמה ובעיגול):



כתובת היעד (מסומנת בחבילה עצמה ובעיגול):



כתובות ה-IP בטרמינל (כדי לראות שהן אכן מתאימות לכתובות ב-wireshark):

(הכתובת היא של המחשב המארח את ה-VM, בטרמינל של ה-VM מוצגת הכתובת 10.0.2.15)

```
Wireless LAN adapter Wi-Fi:

   Connection-specific DNS Suffix  . : Home
   IPv6 Address. . . . . . . . . . . : 2001:4df4:187:400:2966:b89f:1d76:e88e
   Temporary IPv6 Address. . . . . . : 2001:4df4:187:400:2196:c44c:d561:8b9a
   Link-local IPv6 Address . . . . . : fe80::2966:b89f:1d76:e88e%5
   IPv4 Address. . . . . . . . . . . : 10.0.0.14
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : fe80::2b8:c2ff:fe06:f2bc%5
                                       10.0.0.138
```

```
dabushori@dabushori-computer:~$ ifconfig
eno1: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether b4:b6:86:bc:9c:ef  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 10083  bytes 5809371 (5.8 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 10083  bytes 5809371 (5.8 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlo1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.32  netmask 255.255.255.0  broadcast 10.0.0.255
        inet6 fe80::e200:a1dd:ff93:5939  prefixlen 64  scopeid 0x20<link>
        inet6 2001:4df4:187:400:37b4:7711:6594:11d1  prefixlen 64  scopeid 0x0<global>
        inet6 2001:4df4:187:400:258c:10e2:670c:8a80  prefixlen 64  scopeid 0x0<global>
        ether fc:01:7c:33:90:6f  txqueuelen 1000  (Ethernet)
        RX packets 330780  bytes 357929928 (357.9 MB)
        RX errors 0  dropped 586  overruns 0  frame 0
        TX packets 110170  bytes 14040156 (14.0 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

חלק ב

תיעוד הקבצים:

Server.py:

השרת מקבל פורט, ip ו-port של שרת האב שלו ושם של קובץ המידע והוא משתמש בפורט על מנת לעשות בינד לסוקט שהוא פתח.

לאחר מכן הוא מחכה שהלקוח ישלח לו כתובת של אתר ואם הכתובת קיימת בקובץ המידע שאת שמו קיבלנו כארגומנט וגם לא עבר ה-ttl שלו אז נחזיר את ה-ip של האתר ונחכה לכתובת נוספת בצורה חוזרת.

יכולים לקרות המקרים הבאים:

אם האתר לא קיים בקובץ שקיבלנו אז בעזרת הפונקציה getInfoFromServer שהולכת לחפש את השם של האתר בקובץ של האב (השרת מתנהג כמו לקוח של האב) ואם הוא גם לא נמצא שם הוא ימשיך לעלות לאבא שלו עד אשר ימצא את הקובץ המבוקש ואז יתבצע תהליך של למידה לכל ההורים עד הילד הראשון כך שנוסיף בקבצים שלהם שורה חדשה עם ה-ttl שכאשר הוא יעבור נצטרך לעלות שוב לשרתי אב לחפש את האתר.

כאשר השרת מקבל קישור ראשית הוא ראשית עובר על כל השורות בקובץ המידע ובודק לכל קישור אם עבר ה-ttl שלו. אם כן, הוא מוחק אותו. כך כאשר הלקוח יבקש קישור שעבר ה-ttl שלו הוא לא יהיה קיים בקובץ המידע וכך השרת לא יחזיר את השורה שנמחקה כתשובה.

אם השרת הגיע לסוף קובץ המידע ולא מצא את הקישור המבוקש הוא יצטרך לקרוא לשרת האב. שרת האב יחזיר לו את המידע הדרוש, השרת ילמד אותו (ויכתוב את השעה בה המידע נלמד) ויחזיר תשובה ללקוח.

Client.py:

הלקוח מקבל פורט ו-ip של השרת אליו הוא צריך להתחבר, לאחר מכן הלקוח מבקש כתובת של אתר ואז הוא מתחבר לשרת שהוא קיבל את ה-ip והפורט שלו כארגומנטים ומבקש ממנו את ה-ip של הכתובת שהוא קיבל ולאחר שהוא מקבל את ה-ip שהשרת החזיר לו הוא מבצע את הבקשה לכתובת שוב והפעולה חוזרת חלילה (ההתחברות לשרת נעשית בתוך המתודה getInfoFromServer).

הרצה: (התוכן הבא יכלול סקרינשוטים)

בדוגמת ההרצה שלנו הרצנו שלושה תוכניות – לקוח, שרת בן ושרת אב.

שרת האב רץ בפורט 55555. שרת האב שלו רץ בפורט וכתובת ip מינוס אחד, כלומר לא היה לו שרת אב. קובץ המידע שלו (parent.txt) הכיל את המידע הבא:

www.google.co.il,8.8.8.8,2

mail.google.co.il,9.9.9.9,5

שרת הבן רץ בפורט 12345 . שרת האב שלו רץ בפורט 12345 ובכתובת ip שהיא 127.0.0.1, כלומר באותו המחשב. קובץ המידע שלו (ips.txt) הכיל את המידע הבא:

www.biu.ac.il,1.2.3.4,180

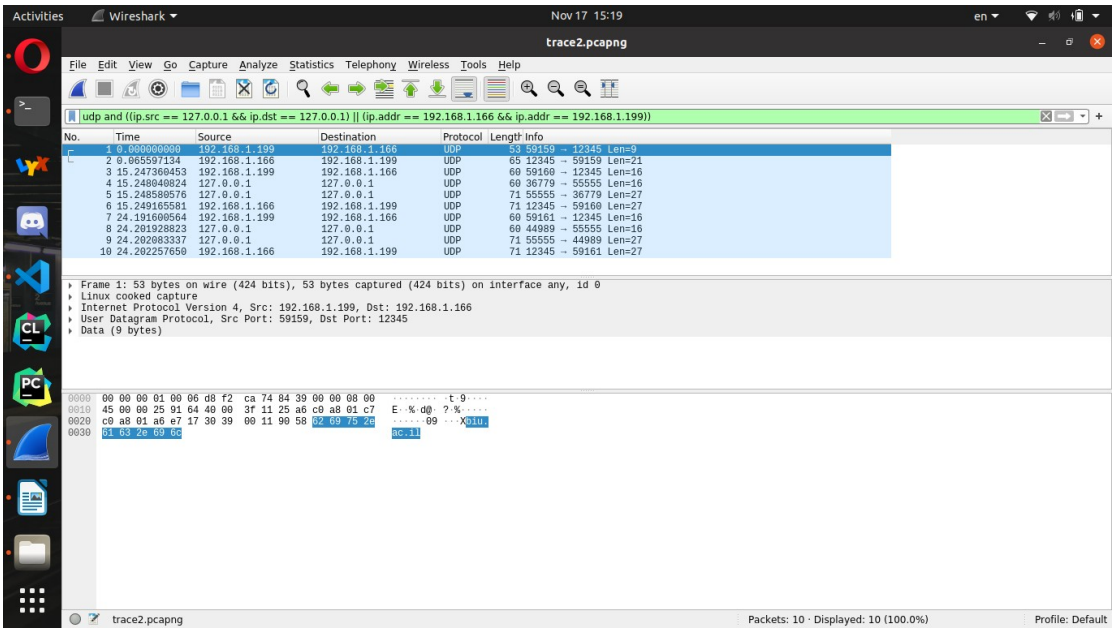
mail.biu.ac.il,1.2.3.5,240

biu.ac.il,1.2.3.4,180

הלקוח רץ כשנתוני השרת שלו הם פורט 12345 וכתובת ip שהיא 192.168.1.166, שזוהי כתובת ה-ip של המחשב בה הורצו השרתים.

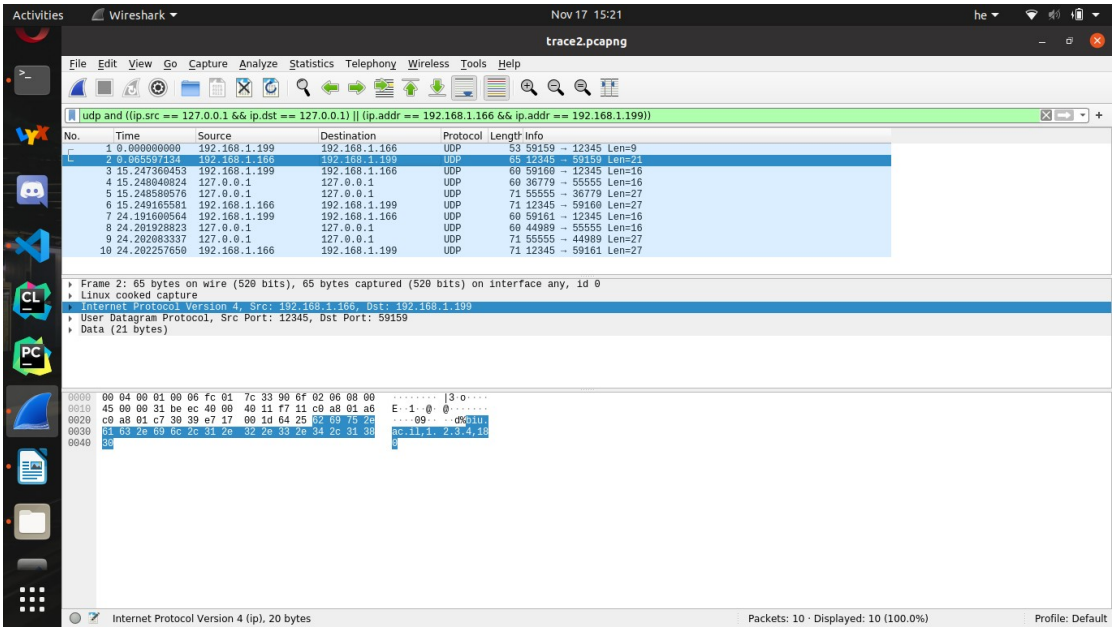
ראשית הכנסנו ללקוח את הכתובת biu.ac.il. הפלט במסך היה 1.2.3.4 כמצופה.

ראשית הלקוח (שרץ במחשב שכתובת ה-ip שלו היא 192.168.1.199) העביר את החבילה הבאה לשרת הבן שרץ בכתובת 192.168.1.166 בפורט 12345:



חבילה זו מכילה את הקישור biu.ac.il כפי שמסומן בתמונה.

לאחר מכן השרת החזיר ללקוח את החבילה הבאה:



שבה ניתן למצוא את תשובת השרת, כלומר את השורה המתאימה בקובץ המידע (המכילה את הקישור, כתובת ה-ip ואת ה-ttl). מתוך שורת המידע הזו הלקוח מדפיס את כתובת ה-ip למסך.

כעת שלחנו לשרת את הכתובת www.google.co.il שלא נמצאת בקובץ המידע של השרת אלא בקובץ המידע של שרת האב.

החבילות שנשלחו הן הבאות:

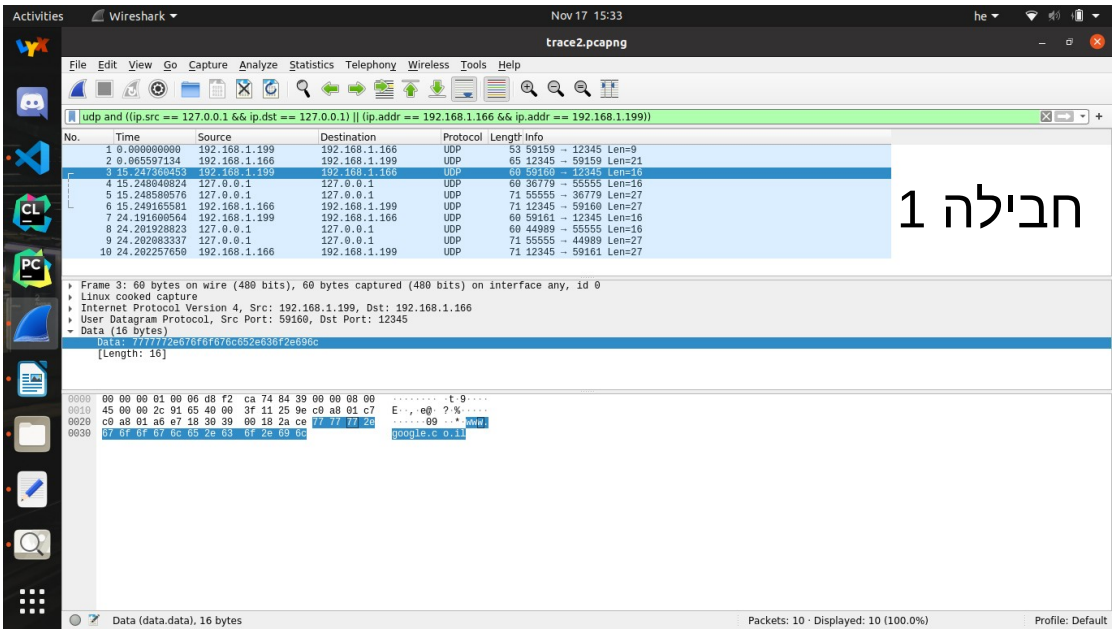
1. הלקוח (כתובת 192.168.1.199 ופורט כלשהו שהלקוח קיבל ממערכת ההפעלה) שולח לשרת (כתובת 192.168.1.166 ופורט 12345) את הקישור המבוקש.

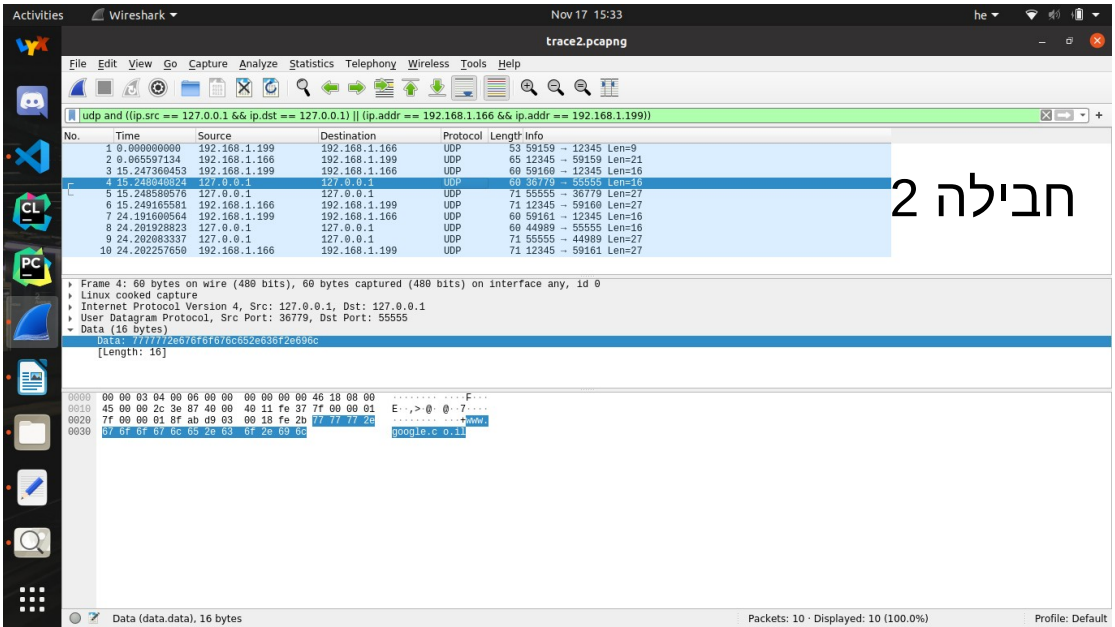
2. השרת (כתובת 127.0.0.1 ופורט כלשהו שקיבל ממערכת ההפעלה) שולח לשרת האב (כתובת 127.0.0.1 ופורט 55555) את הקישור המבוקש.

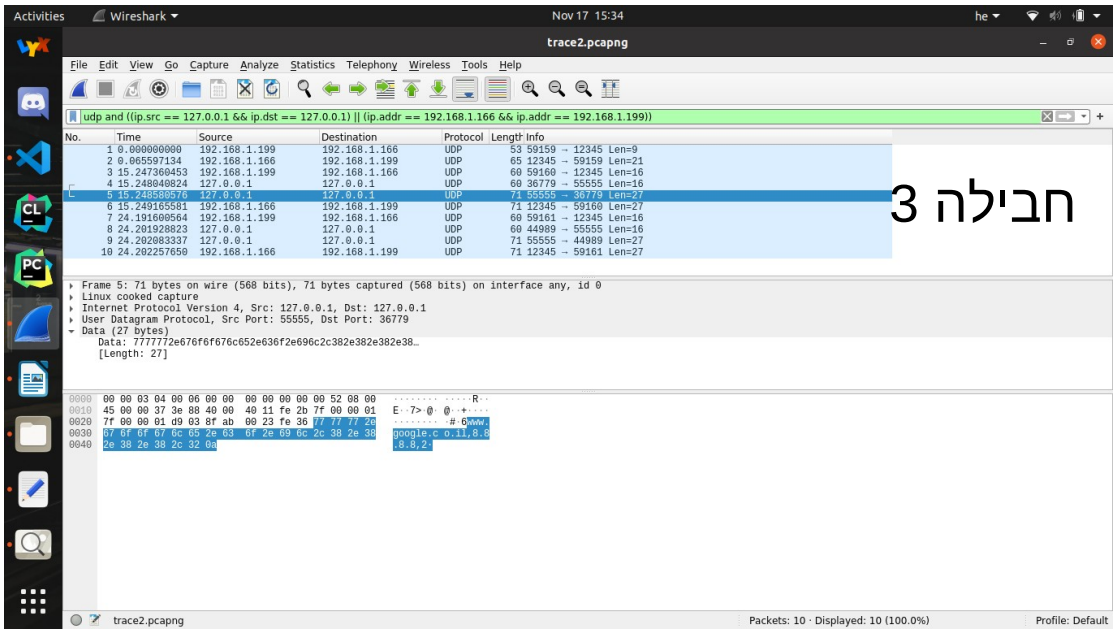
3. שרת האב (כתובת 127.0.0.1 ופורט 55555) מחזיר לשרת (כתובת 127.0.0.1 ופורט ממנו שלח את הבקשה) את התשובה (שורת המידע).

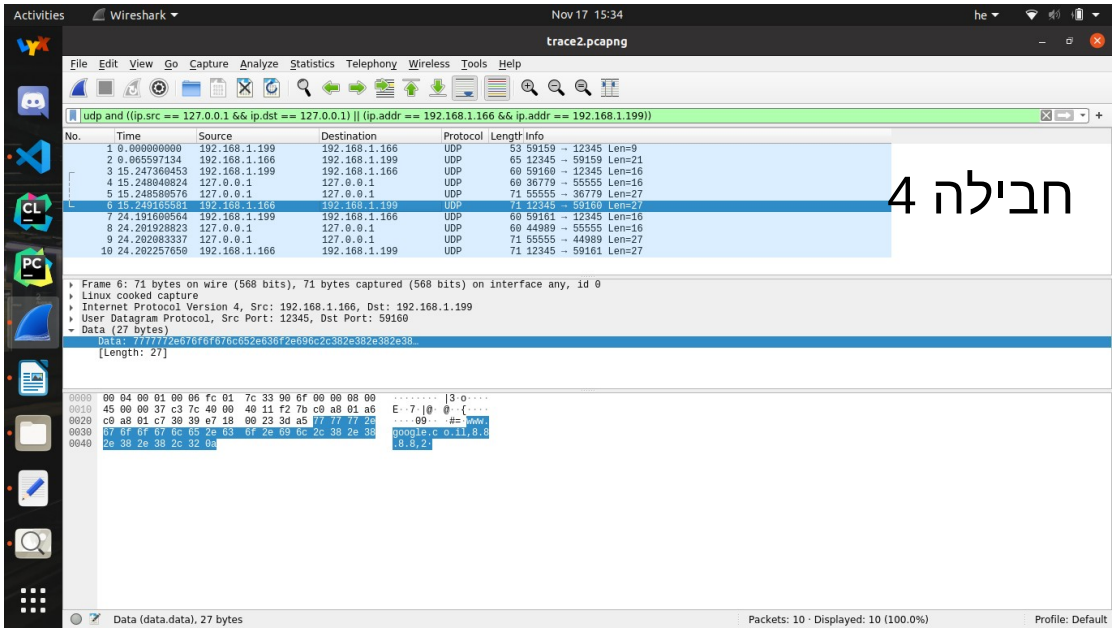
4. השרת (כתובת 192.168.1.166 ופורט 12345) שולח ללקוח (כתובת 192.168.1.199 ופורט כלשהו ממנו שלח את הבקשה) את התשובה (שורת המידע).

החבילות הנ"ל מוצגות בתמונות הבאות לפי הסדר:







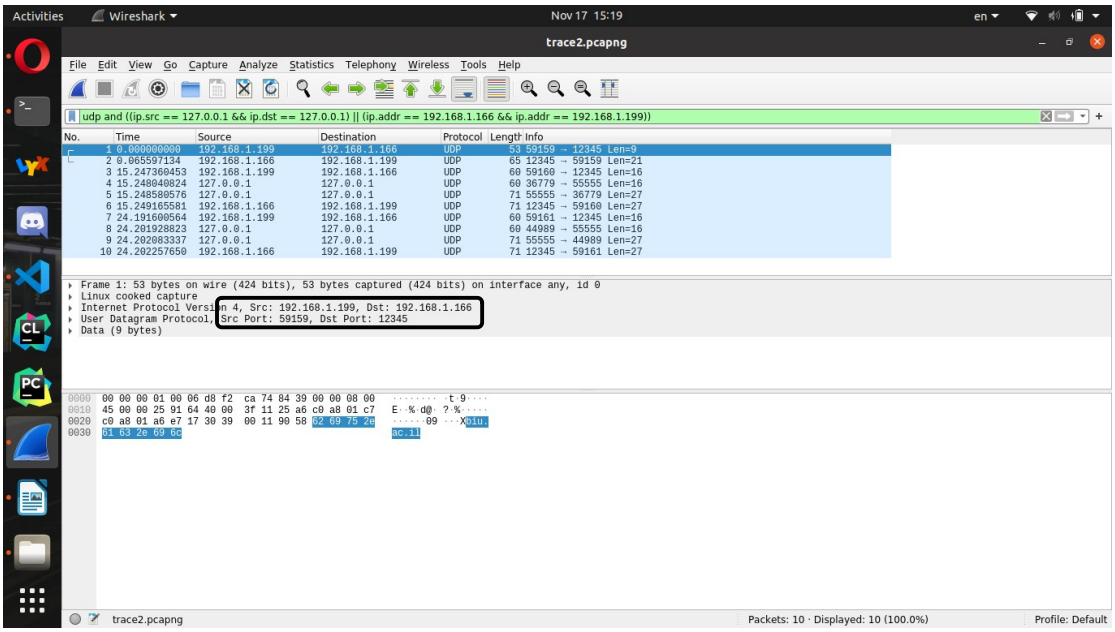


נשים לב שבנוסף לשליחת החבילות שרת הבן למד את התשובה משרת האב ולכן שמר את השורה הבאה בקובץ המידע שלו (ips.txt):

www.google.co.il,8.8.8.8,2,15:06:40

כעת ביקשנו שוב פעם את אותה הבקשה, כלומר הכנסנו כקלט ללקוח את הכתובת www.google.co.il. אותו התהליך קרה (ניתן לראות את החבילות בקובץ trace2 המצורף, לא ראינו צורך לצלם שוב 4 צילומי מסך כיוון שהחבילות מכילות את אותו התוכן). הדבר היחיד שהשתנה הוא שלפני שהשרת ביקש משרת האב את הקישור הוא עבר על הקישורים בקובץ שלו וניקה את אלו שה-ttl שלהם עבר, כלומר כאשר השרת עבר על השורות שלו כדי לבדוק אם הקישור קיים הוא כבר לא היה קיים כי ה-ttl שלו עבר ולכן פנה לשרת האב. לאחר ביצוע פעולה זו השורה הקודמת שהוכנסה לקובץ המידע (כלומר www.google.co.il,8.8.8.8,2,15:06:40) נמחקה ובמקומה נוספה השורה www.google.co.il,8.8.8.8,2,15:06:46 שמראה על למידה מחדש של השרת.

נשים לב בנוסף שבכל תמונה של חבילה שצירפנו ניתן לראות את כתובות ה-ip של המקור והיעד ואת הפורטים של המקור והיעד. לדוגמא בחבילה הראשונה שנשלחה (בבקשה הראשונה מהלקוח) שנשלח מהלקוח (שרץ במחשב שכתובת ה-ip שלו היא 192.168.1.199) לשרת (שרץ בכתובת 192.168.1.166 בפורט 12345) ניתן לראות את המידע הנ"ל במיקום הבא:



כעת נסביר כיצד המידע נבנה בכל אחת מהשכבות.

שכבת האפליקציה מעלה את המידע המועבר, כלומר את הקישור או את שורת המידע התשובה של השרת).

שכבת ה-transport מוסיפה את ה-header שלה, בין היתר היא מוסיפה את הפורטים של המקור והיעד. היא משתמשת בפרוטוקול UDP.

שכבת ה-network מוסיפה את ה-header שלה, ובתוכו כותבת את כתובות ה-ip של המקור והיעד. היא משתמשת בפרוטוקול IPv4.

שכבת ה-link מוסיפה את ה-header וה-tail שלה בו כותבת את המידע ובין היתר את

כתובות ה-mac של המקור והיעד.

בכל שורה בחלק האפור בתמונות של החבילות כתוב את המידע הקשור לשכבה מסוימת, כאשר השכבה התחתונה ביותר היא שכבת האפליקציה והעליונה ביותר (אחת לפני העליונה ביותר..) היא שכבת ה-link.

הפלט של התוכנית (כלומר הפלט שמוצג ללקוח) מוצג בתמונה הבאה:

```
yorai@yorai-linux:~/Desktop$ python3 client.py 192.168.1.166 12345
Enter a url: biu.ac.il
1.2.3.4
Enter a url: www.google.co.il
8.8.8.8
Enter a url: www.google.co.il
8.8.8.8
Enter a url:
```