

1. אם כן, ביצענו פתיחת client+server והעברת מסר תעודות הזהות שלנו לשרת.

ניתן לראות בתמונה את שליחת המסר (הפעלת קובץ clienta מתוך הטרמינל):

```
david@david-XPS-15-7590:~/Documents/Misc Coding Projects$ python3 client.py
b'Server: Hello DAVID DINKEVICH (ID: 584698174), ROEI GEHASI (ID: 208853754)' ('127.0.0.1', 12345)
david@david-XPS-15-7590:~/Documents/Misc Coding Projects$
```

אם כן, ניתן לראות בטרמינל השני כי קיבלנו תשובה חזרה מהserver:

```
david@david-XPS-15-7590:~/Documents/Misc Coding Projects$ python3 server.py
Server: b'David Dinkevich (ID: 584698174), Roei Gehasi (ID: 208853754)' ('127.0.0.1', 60438)
david@david-XPS-15-7590:~/Documents/Misc Coding Projects$
```

כעת, תוך כדי העברת המסרים הסנפנו את התעבורה באמצעות Wireshark.

בכדי לראות בדיוק את החבילות המבוקשות סיננו את החבילות באמצעות שורת הקוד שנמצאת בתמונה שכללה

No.	Time	Source	Destination	Protocol	Length	Info
2774	9.529136119	127.0.0.1	127.0.0.1	UDP	104	49428 → 12345 Len=60
2775	9.529402184	127.0.0.1	127.0.0.1	UDP	118	12345 → 49428 Len=74

באמצעות הפקודה: `udp && udp.port == 12345` פקודה זו מסננת לנו את כל החבילות שהועברו מהפורט 12345 וייצאו אליו.

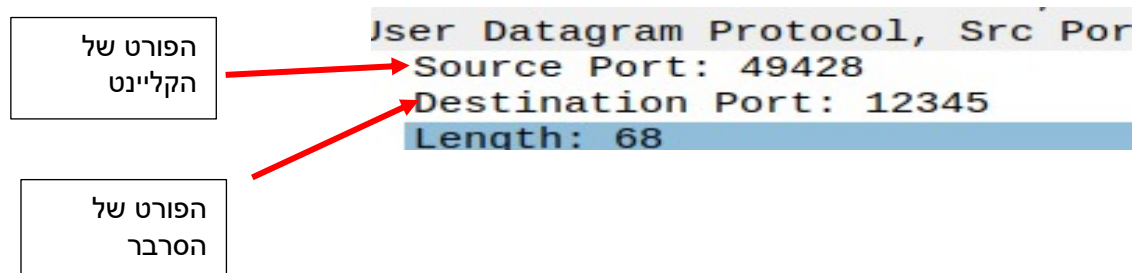
כעת נסביר את שימושי הפורט בין השרת לסרבר.

אם כן, הסרבר פותח socket, שתפקידו לייצר מעין "שקע התחברות" לפורטים—כלומר מאפשר מעבר מיידע בין פורטים שונים.

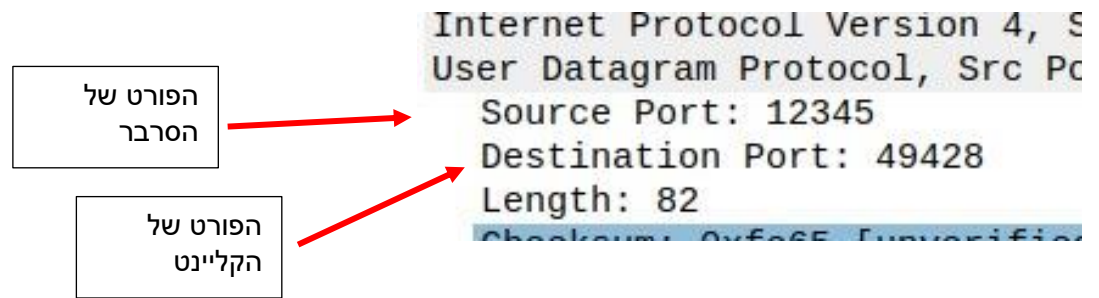
לסוקט עושים bind לפורט פנוי כלשהו (מנסים עד שמוצאים פורט פנוי).

אם כן, אצלינו, הסרבר פותח סוקט אליו יוכלו להתחבר באמצעות הפורט 12345.

2. ניתן לראות כי הקליינט שולח לפורט שהסרבר שלח, ואילו הסרבר מאזין לכל פורט באשר הוא ומחכה לקבל חבילות לפורט שהוא פתח (אותו פורט 12345).



כעת, הסרבר קיבל את החבילה מאת הקליינט ולכן רוצה להחזיר לו חזרה שהוא אכן קיבל את החבילה.  
ניתן לראות בתמונה הבאה כי השרת אכן קיבל את החבילה בכך שהוא שולח בעצמו חבילה חוזרת לאותו קליינט:



3. המסר שנשלח התבצע בשכבת התעבורה באמצעות הפרוטוקול UDP, ניתן לראות זאת בתמונה הבאה:

User Datagram Protocol, Src Port: 49428, Dst Port: 12345  
Source Port: 49428  
Destination Port: 12345  
Length: 68  
Checksum: 0xfe57 [unverified]  
[Checksum Status: Unverified]  
[Stream index: 6]  
[Timestamps]

בקיצור: UDP

4. כעת ניתן לראות כי התקשורת בין הקו השונים בהם הועברו החבילות היו בין המחשב לעצמו.

ניתן לראות זאת על ידי ה source וה destination בתמונה הבאה:

udp && udp.port == 12345						
No.	Time	Source	Destination	Protocol	Length	Info
2774	9.529136119	127.0.0.1	127.0.0.1	UDP	104	49428 → 12345 Len=60
2775	9.529402184	127.0.0.1	127.0.0.1	UDP	118	12345 → 49428 Len=74

כעת נשווה ונסביר כתובות אלו כפי שהן מופיעות תחת הפקודה `ifconfig`.

כתובת הקו של המחשב "האני"

כרטיס הרשת של המחשב הוירטואלי


```
David@david-XPS-15-7596:~/Documents/Misc Coding Projects$ ifconfig
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 52135 bytes 4670916 (4.6 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 52135 bytes 4670916 (4.6 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

vlp59s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
```

## חלק ב

הדגמת הריצה של חלק 1 עבור חלק ב.

כלומר, ריצת `foo` עם ארגומנט רביעי (1).



The screenshot shows a terminal window with two tabs. The active tab is titled "david@david-XPS-15-7590: ~/Documents/git projects/Computer-Networki...". The terminal content shows the command `python3 client.py 127.0.0.1 12346 roei.txt` being executed. A red arrow points from the text "client.py" in the previous block to the `client.py` in the terminal command.

הרצת הקליינט בטרמינל, כפי שניתן לראות הארגומנט הראשון הינו כתובת הקו של הקליינט והארגומנט השני הוא הפורט של .foo

Roei.txt זהו קובץ הטקסט שאותו אנו קוראים ושולחים לסרבר (דרך foo).

```
david@david-XPS-15-7590:~/Documents/git_projects/Computer-Networking---Ex1/Part 2$ python3 foo.py 12346 127.0.0.1 12345 1
Playing nice
Phiiiii, no drop.... 96
Yay, no sleep.... 36 100
Forwarded b'\x00\x00\x00\x00\x00\x00\x00\x00\x00a\aaa\aaaa\naaaa\naaaaaa\naaaaaaa\naaaaaaaa\naaaaaaaaa\naaaaaaaaa\naaaaa' from ('127.0.0.1', 34489) to ('127.0.0.1', 12345)
aaaaaaaa\naaaaaaaaa\naaa' from ('127.0.0.1', 12345) to ('127.0.0.1', 34489)
Phiiiii, no drop.... 7
Yay, no sleep.... 86 100
Forwarded b'\x00\x00\x00\x00\x00\x00\x00\x00\x00a\aaa\aaaa\naaaa\naaaaaa\naaaaaaa\naaaaaaa\naaaaaaaa\naaaaaaaaa\naaaaa' from ('127.0.0.1', 12345) to ('127.0.0.1', 34489)
aaaaaaaa\naaaaaaaaa\naaa' from ('127.0.0.1', 12345) to ('127.0.0.1', 34489)
Phiiiii, no drop.... 27
Yay, no sleep.... 13 100
Forwarded b'\x01\x00\x00\x00\x00\x00\x00\x00\x00a\naaaa\naaaaaaa\naaaaaaa\naaaaa\naaaa\naaa\naaa\nna\nn' from ('127.0.0.1', 34489) to ('127.0.0.1', 12345)
Phiiiii, no drop.... 58
Yay, no sleep.... 68 100
Forwarded b'\x01\x00\x00\x00\x00\x00\x00\x00\x00a\naaaa\naaaaaaa\naaaaaaa\naaaaa\naaaa\naaa\naaa\nna\nna\nn' from ('127.0.0.1', 12345) to ('127.0.0.1', 34489)
```

הפעלת פונקציית foo עם הארגומנטים הרלוונטיים בהתאמה.

כאשר ניתן לראות כי הפורט אליו foo שולח את המידע הוא הפורט אליו 'יאזין הסרבר' (כמו כן הפורט אליו foo מאזין הוא הפורט לו שולח הקליינט. ניתן לראות זאת בתמונה אחת מעל).

אם כן, קיבלנו את פלט הריצה עבור ארגומנט 1.

כעת נראה מה קיבלנו בטרמינל בו האזין הסרבר:

[illegible]

קיבלנו את קובץ הטקסט אותו שלחנו בצורה תקינה.

כעת נדגים עבור החלק הראשון את התעבורה בwireshark :

Wireshark packet capture analysis of a UDP packet. The packet list shows a packet from 127.0.0.1 to 127.0.0.1 on ports 34489 to 12346. The packet details show it's a User Datagram Protocol packet with source port 34489 and destination port 12346. The packet data is 62 bytes long. The packet bytes are shown in hexadecimal and ASCII. Annotations in Hebrew explain the packet structure and the data content.

סינון התעבורה הרלוונטי עבור הפורטים אותם אנו מחפשים

פורט המקור (הלקוח) פורט היעד (foo)

Datan עצמו שהועבר

Datan עצמו שהועבר בתרגום לASCII

כעת נדגים עבור תת חלק 2 בחלק ב

אם כן בחלק זה, נזרקו לנו חלק מההודעות באופן רנדומלי (אחוז מוגרל) מ-2 הכיוונים. גם מכיוון הסרבר וגם מכיוון הלקוח, foo זרקה לנו מידע.

אם כן, התמודדנו עם בעיה זו על ידי כך שעידכנו את הלקוח וגם את הסרבר שאם לא נשלחה הודעה, עליהם לשלוח שוב את אותה ההודעה בתוך זמן מסוים.

בכל פעם קראנו 90 בתים מהקובץ כך שלאותם 90 בתים הוספנו 10 בתים שישמשו למיספור ההודעה שנשלחה.

כך ידענו לבדוק האם אותה הודעה נשלחה כבר לסרבר או שזוהי הודעה חדשה.

אם זוהי הודעה חדשה—שלחנו חזרה עידכון ללקוח (כלומר, אכן ההודעה הקודמת שלנו הגיעה ללקוח וכעת אנו מצפים להודעה חדשה).





[illegible]

ניתן לראות את ריצת הfoo, ריצת הclient והסרבר נותנת את אותה תוצאה כפי שקיבלנו לעיל.

כלומר:

```
david@david-XPS-15-7590:~/Documents/git projects/Computer-Networking---Ex1/Part 2$ python3 server.py 12345  
a  
aa  
aaa  
aaaa  
aaaaa  
aaaaaa  
aaaaaaa  
aaaaaaaa  
aaaaaaaaa  
aaaaaaaaa  
aaaaaaaaa  
aaaaaaaaa  
aaaaaaaaa  
aaaaaaaaa  
aaaaaaaaa  
aaaaaaaaa  
aaaaaaaaa  
aaaaaaa  
aaaaaaa  
aaaaaaa  
aaaaa  
aaaaa  
aaa  
aa  
a
```

אם כן, בכדי להתמודד עם בעיית הדילי, הוספנו טיימר בהגדרת socket של 10 שניות. קל לראות שהזמן המקסימלי לדילי הוא 5 שניות (לכל כיוון).

כלומר, במידה ונשלחה הודעה מהלקוח אל פו, ופו נתן לאותה הודעה דיליי של 5 שניות, שלח לסרבר, הסרבר שלח חזרה לפו. פו יכול שוב לבצע דיליי של 5 שניות עד שישלח חזרה את ההודעה ללקוח.

כלומר יכולות לעבור 10 שניות שלמות עד שהלקוח יקבל חזרה תשובה מהסרבר. לכן עלינו לתת זמן השהייה ללקוח שהוא המינימום מעל 10 שניות.

וכך פעלנו.

חלק 4 : שילוב של שניהם (דיליי וזריקת מידע)

[illegible]

באותה מידה כפי שעשינו בקודמים, אנו מוודאים בעזרת הדילי ובעזרת השוואת מספר ההודעה שכבר נשלחה להודעה שכעת קיבלנו בסרבר, שסדר ההודעות נשמר (אם ההודעה זזה להודעה שכבר קיבלנו, הסרבר לא מחזיר הודעה ללקוח).

כך נשמר סדר ההודעות. כפי שכבר הסברנו לעיל בחלקים 3 ו2. סה"כ השתמשנו בשיטת Stop and Wait.