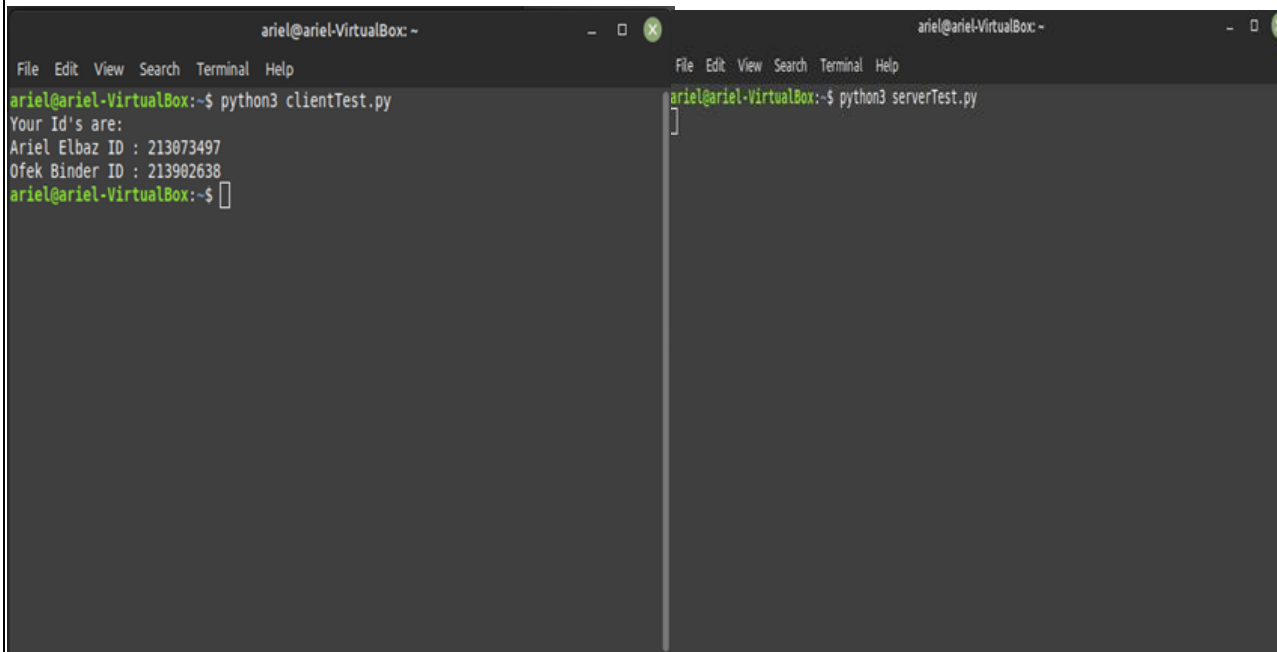


דוח עבודה 1 במבוא לרשתות:

חלק א -

סעיף 1 –

נראה שהצלחנו ליצור קשר בין השרת ללקוח-



The image shows two terminal windows side-by-side. The left window, titled 'ariel@ariel-VirtualBox: -', shows the execution of 'python3 clientTest.py'. It outputs 'Your Id's are:', 'Ariel Elbaz ID : 213073497', and 'Ofek Binder ID : 213902638'. The right window, also titled 'ariel@ariel-VirtualBox: -', shows the execution of 'python3 serverTest.py'. It outputs a single '[' character, indicating it has received the client's ID list.

ניתן

לראות צילום של הרצת 2 process שונים

אחד מייצג לנו את הלקוח (השמאלי) השני מייצג לנו את השרבר(הימני).

מה שקורה במהלך הרצת 2 process זה שהסוקט של השרת מופעל ומחכה בלולאה אינסופית עד שלקוח כלשהו יפנה אליו.

לאחר מכן כשהלקוח שלנו מתחיל לפעול הוא יוצר קשר עם השרת ושולח לו את תעודות הזהות של אריאל ושל אופק והשרת מוסיף את הפתיח "Your id's are:" ומחזיר את המחרוזת השלמה אל הלקוח.

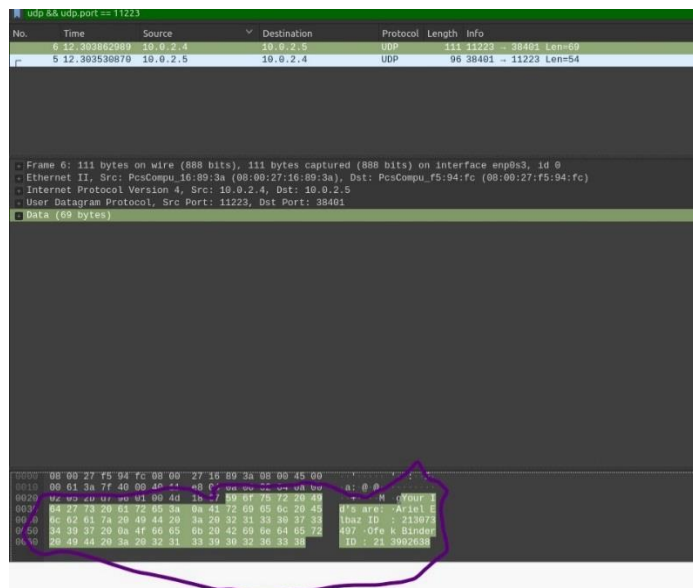
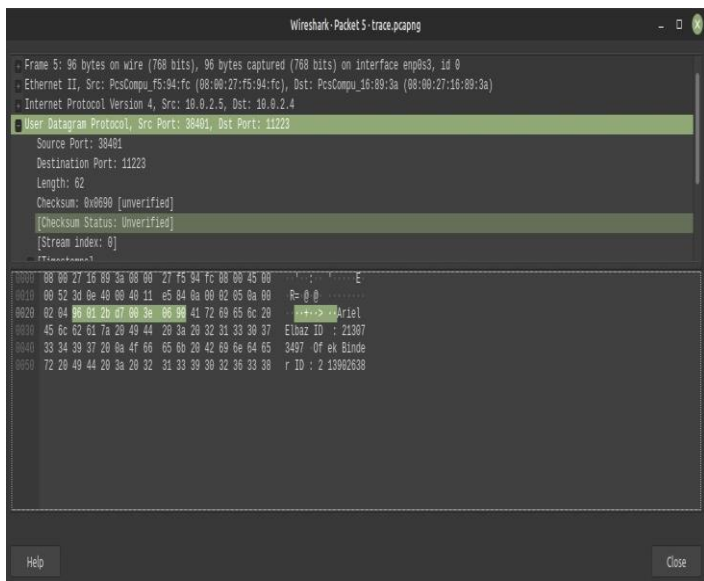
בנוסף השרת מדפיס הודעת אישור (למסך שלו) על כך שהוא קיבל את ההודעה מהלקוח.

לאחר מכן הלקוח מדפיס את המחרוזת השלמה שקיבל ולאחר מכן סוגר את הסוקט.

לאחר ביצוע הפעולות הללו השרת שלנו ממשיך לעבוד בלולאה האינסופית ומחכה לפנייה של לקוחות אחרים אליו (כפי ששרת אמור לנהוג) וכך נוכל להרחיב את הרשת ולבנות לקוחות נוספים שיצרו קשר עם השרת שלנו.

ניתן לראות לפי ההדפסות שהקשר בין הלקוח לשרת צלח.

עתה נוסיף צילומי מסך של הסנפות מן ה Wireshark אשר קלטו את החבילות שעברו בין הלקוח אל השרת ולהפך.



ניתן לראות בתמונה הימנית שלאחר סינון (נפרט על הסינון בשלב ב) תפסנו 2 חבילות אשר מתאימות לתנאים שהצבנו שהן אכן 2 החבילות שנשלחו בין הלקוח לשרת הראשונה מן הלקוח אל השרת והשנייה מן השרת אל הלקוח (כמתואר בהסבר על הקוד שנכתב בעמוד הקודם).

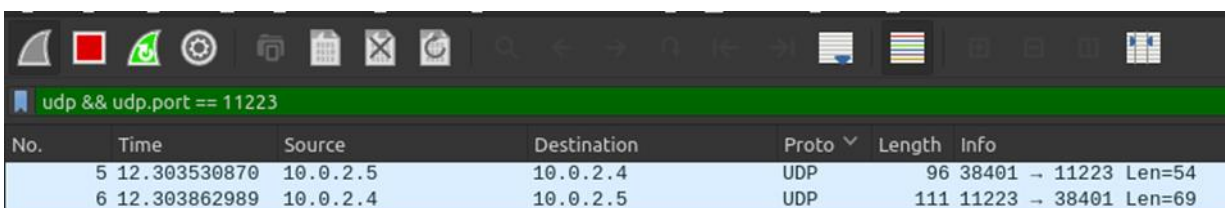
ניתן לראות לפי port המקור והיעד שכתובים שתחילה שלחנו הודעה מהלקוח אל השרת ובחודעה השנייה מן השרת אל הלקוח.

בתמונה הימנית ניתן לראות בחלק התחתון המסומן בסגול את מידע ההודעה שנשלחה וזוהי אכן ההודעה שנשלחה בקוד על ידי השרת אל הלקוח.

בתמונה השמאלית ניתן לראות את פירוט ההודעה הראשונה שנשלחה מן הלקוח אל השרת (בעצם הקשר הראשוני שנוצר בין 2 הסוקטים) וזהו אכן פירוט ההודעה המתואר בקוד.

בעבודה זו השתמשנו בכרטיס הרשת שעל המחשב שמספרו הוא enps03 תפקידו של כרטיס הרשת הוא לאפשר למחשב להתחבר לרשת המחשבים וזאת על מנת שנוכל ליצור קשר עם מחשבים אחרים (למרות שבחלק זה השתמשנו רק במחשב שלנו והחיבור נעשה בצור וירטואלית).

סעיף 2 –



No.	Time	Source	Destination	Proto	Length	Info
5	12.303530870	10.0.2.5	10.0.2.4	UDP	96	38401 → 11223 Len=54
6	12.303862989	10.0.2.4	10.0.2.5	UDP	111	11223 → 38401 Len=69

תוכנת האפליקציות שהרצנו Wireshark קולטת כל תקשורת שמבוצעת עם המחשב שלנו אך בכל רגע נתון המחשב שלנו מקבל חבילות רבות מהמון גורמים שונים (כאשר הוא מחובר לרשת אינטרנט מסויימת) לכן אנו צריכים לסנן ולמצוא את ההודעות אותן אנו שלחנו ועליהם אנו רוצים להסתכל.

לכן נאלץ למצוא דרך לחפש את החבילות עליהם אנו רוצים להסתכל.

על מנת לעשות זאת נקליד בשורת הסינון את הקלט הבא `udp&&udp.port==11223` קלט זה עוזר לנו לסנן את החבילות כתלות ב2 דברים פרוטוקול התקשורת שלנו ומספר הפורט המעורב. כך נקבל רק את החבילות המייצגות את 2 התנאים הבאים:

1. החבילות אשר פועלות לפי פרוטוקול התעבורה `udp` בו עבדנו. `Udp` הוא פרוטוקול הנמצא בשכבת התעבורה ומשתמשים בו להעברת מידע. היחוד של פרוטוקול זה הוא שהוא מהיר יותר מאשר שאר הפרוטוקולים הדומים לו וזאת משום שהוא בניגוד לפרוטוקולים אחרים (כמו `tcp`) אינו מבצע בדיקה שהחבילה הגיעה ליעדה ללא שגיאות, אינו מספק תהליך שבדק אם המגיע שהגיע הוא אמין, מבנה החבילה שלו קטן וכולל פתיח באורך 64 סיביות. החסרון בפרוטוקול זה הוא שבשל הגורמים שהעלנו קודם הוא נחשב כפרוטוקול אינו אמין משום שאין לנו אף הבטחה שהמידע יגיע בשלמותו או יגיע כלל. השימוש העיקרי בפרוטוקול זה הוא לאפליקציות אשר מעדיפות שהמידע יעבור במהירות על חשבון האמינות למשל באפליקציות של שיחות וידאו (במידע ונדרוש שהמידע שיגיע יהיה אמין יקח למידע זמן רב יותר לעבור ויהיה קשה לתקשר ככה בשיחה). לסיכום פרוטוקול `udp` הוא פרוטוקול תעבורה אינו אמין אך מפצה על כך במהירות מעבר החבילות.

2. Port היעד או המקור חייב להיות 11223 וזאת משום שזהו הפורט של השרת שלנו. אז אנחנו יודעים בהכרח שהוא חייב להיות היעד או המקור כי אנו מעוניינים רק בחבילות שעוברות מהשרת או אל השרת.

ניתן לראות בתמונה המצורפת שביצענו חיפוש ואכן קיבלנו את 2 החבילות המתאימות (שהן החבילות היחידות ששלחנו בין ה 2 process) שבשניהם פרוטוקול התקשורת בו עובדים 2 הצדדים הוא פרוטוקול `udp`. בנוסף ניתן לראות שבחבילה הראשונה הפורט שהצבנו (שהוא של השרת) הוא פורט היעד ובחבילה השנייה הוא פורט המקור.

בנוסף מצורף קובץ `trace1` אשר מכיל את ה `trace` המסונן

סעיף – 3

השתמשנו בקוד שלנו במספרי ה port של השרת ושל הלקוח.

עתה נראה בכל אחד מהם איפה השימוש במספרי port ולמה הוא נועד-

קוד השרת (server)-

```
def runServer():  
    # serverName = 'ArielServer'  
    serverPort = 11223  
    serverSocket = socket(AF_INET, SOCK_DGRAM)  
    serverSocket.bind(('', serverPort))
```

ראשית לפני שנכנס ללולאה האינסופית שמרנו את מספר ה port של השרת תחת המשתנה serverPort וזאת על מנת שבפקודת ה bind נוכל לקשור את אובייקט ה socket שלנו אל הכתובת המתאימה שהיא כתובת ה port שלנו. (זה יעזור בכך שכאשר לקוח כלשהו יפנה לכתובת פורט זאת הוא יגיע באמת אל הסרבר שלנו).

```
while True:  
    msg, clientAddress = serverSocket.recvfrom(1024)
```

עתה אנו נפעיל לולאה אינסופית שתפעל בצורה הבאה. ראשית נקבל חבילה כלשהי וזאת בעזרת הפקודה recvform מקבלת מספר מסויים המתאר את מספר הביטים המקסימלי אותו ניתן לקבל בפקודה זו. בעזרת הפקודה זו וקבלת החבילה נקבל 2 דברים שימשו אותנו בהמשך:

1. את ההודעה שנשלחה לנו (במקרה הזה תעודות הזהות שנשלחו מהלקוח) שנשמור אותה תחת המשתנה msg (ובהמשך נראה מה נעשה עם הודעה זו)
2. את פרטי השולח שבמקרה זה הוא הלקוח שלנו אשר ישמרו בזוג הכולל את מספר ה port של הלקוח ומספר ה ip של הלקוח אשר ישמרו תחת השם clientAddress (בהם נשתמש בהמשך על מנת להחזיר הודעה אל הלקוח).

```
print(f"server got the msg:\n{msg.decode()}")  
modifiedMsg = f"Your Id's are:\n{msg.decode()}"  
serverSocket.sendto(bytes(modifiedMsg, 'utf-8'), clientAddress)
```

בשלב הבא נדפיס שקיבלנו את ההודעה. לאחר מכן נוסיף לפני ההודעה שקיבלנו את הקידמות שהתבקשנו (:Your Id's are). ובשלב הבא אנו נחזיר ללקוח את ההודעה השלמה בעזרת הפקודה sendto נחזיר את ההודעה מקודדת בקידוד הנקרא utf-8. עתה השרת כבר קיבל את הפרטים של הלקוח אשר נשמרו בזוג תחת המשתנה clientAddress (את ה port קיבלנו בשכבת התעבורה ואת ה ip בשכבת הרשת).

לאחר שהשרת ישלח בחזרה את ההודעה השלמה הוא ימשיך לרוץ עד אינסוף כדי לקבל הודעות נוספות.

קוד הלקוח (client)-

```
def runClient():  
    serverIP = '10.0.2.5'  
    serverPort = 11223
```

בקוד הלקוח ראשית שמרנו את מספרי הport וה ip של הסרבר במשתנים וזאת משום שאנו נצטרך אותם כדי ליצור קשר עם הסרבר אליו אנו רוצים להגיע.

```
clientSocket = socket(AF_INET, SOCK_DGRAM)  
msg = 'Ariel Elbaz ID : 213073497 \nOfek Binder ID : 213902638'
```

השלב הבא הוא פתיחת סוקט חדש שבעזרתו נוכל ליצור קשר עם השרת ובנוסף נכתוב את ההודעה אותה אנו רוצים לשלוח.

```
clientSocket.sendto(msg.encode(), (serverIP, serverPort))
```

עתה בעזרת פקודת ה sendto נשלח לשרת את ההודעה אותה אנו רוצים לשלוח. פקודה זו עובדת בכך שהיא מקבלת את ההודעה המיועדת לשליחה מקודדת לביטים ובנוסף היא מקבלת זוג המייצגים את ה ip וה port של היעד אליו אנו רוצים לשלוח את החבילה.

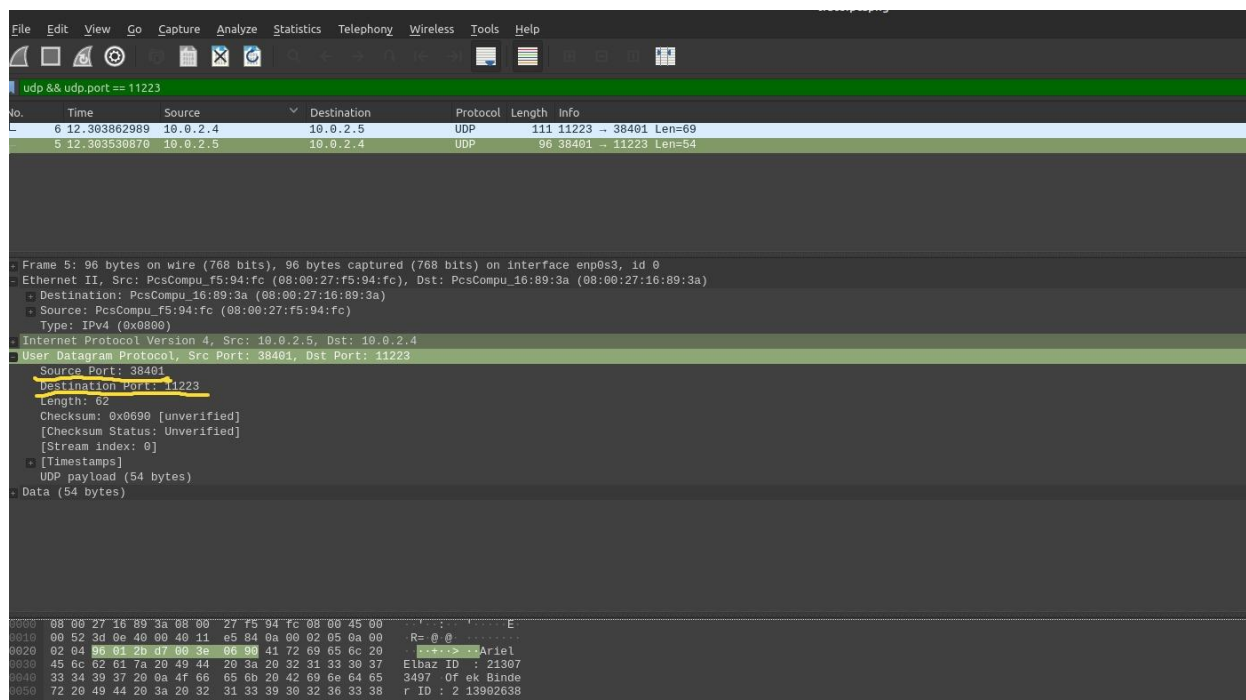
חשוב לציין שאין לנו צורך לבצע bind בין הלקוח אל socket מסויים משום שאם לא מבצעים זאת מערכת ההפעלה מבצעת זאת בעצמה ומשייכת port פנוי כלשהו עבור הלקוח (לנו אין זה משנה מה הport משום שהשרת מחזיר הודעה ללקוח רק לאחר שמקבל הודעה מהלקוח ולכן הוא מקבל את ערכי ה port שלו ללא תלות במה מערכת ההפעלה או אנחנו נבחר).

```
data, serverAddress = clientSocket.recvfrom(1024)  
print(data.decode())  
clientSocket.close()
```

בשלב הבא לאחר ששלחנו את ההודעה הלקוח מחכה לקבלת ההודעה מהסרבר בעזרת הפקודה recvfrom עתה קיבלנו 2 ערכים הראשון הוא ההודעה שקיבלנו מן השרת והשני הוא פרטי השרת שמגיעים לנו כזוג של port ו ip . לאחר מכן אנו נדפיס את ההודעה שקיבלנו ונסגור את ה socket בעזרת הפקודה close .

ניתן לראות שבמהלך הקוד השתמשנו מספר פעמים ב מספרי ה port של הלקוח ושל השרת וזאת על מנת שנוכל לשלוח חבילות מאחד לשני.

ענה נראה כיצד רואים זאת בחבילות מן wire shark –



```
Frame 2: 111 bytes on wire (888 bits), 111 bytes captured (888 bits) on interface enp0s3, id 0
Ethernet II, Src: PcsCompu_16:89:3a (08:00:27:16:89:3a), Dst: PcsCompu_f5:94:fc (08:00:27:f5:94:fc)
Internet Protocol Version 4, Src: 10.0.2.4, Dst: 10.0.2.5
User Datagram Protocol, Src Port: 11223, Dst Port: 38401
  Source Port: 11223
  Destination Port: 38401
  Length: 77
  Checksum: 0x1867 [unverified]
  [Checksum Status: Unverified]
  [Stream index: 0]
  [Timestamps]
  UDP payload (69 bytes)
  Data (69 bytes)
```

בתצלום זה של חבילה ב wire shark מן הלקוח אל השרת ניתן לראות את מספרי פורט היעד ופורט המקור תחת הכיתוב – user Datagram Protocol

השכבה בה מתבצע השימוש בפורט היא שכבת התעבורה (transport) משום שבשלב זה יש שימוש במספרי הפורט המייצגים פרוססים (אפליקציות ותכנות הרצות) במחשב .

שכבת התעבורה היא אחת מהשכבות המרכיבות את מודלי התקשורת (פעילותה זהה ב2 המודלים המוכרים OSI ו TCP/IP) תפקידה של שכבת התעבורה הוא לוודא שכאשר המידע מגיע למחשב הוא

ימצא את דרכו אל האפליקציה המתאימה במחשב (כמעט תמיד במחשב רצות מספר אפליקציות).
שכבת התעבורה מוצאת לאיזה אפליקציה צריך להגיע המידע בעזרת מספר ה port שנמצא בחבילה
ואומר לאן היא צריכה להגיע, כידוע מספר ה port ייחודי לכל אפליקציה.

שכבת התעבורה מגיעה אחרי שכבת הרשת ובכך היא "סומכת" על שכבת הרשת שהביאה את המידע
למחשב הסופי.

בשכבת התעבורה המידע המועבר נקרא Segment.

סעיף 4 –

ענה נראה בעזרת ה wire shark מאיזה כתובת ip לאיזה כתובת ip אנחנו שולחים

```

udp && udp.port == 11223
No.    Time           Source            Destination       Protocol  Length  Info
5 12.303530870 10.0.2.5          10.0.2.4          UDP        96 38401 → 11223 Len=54
6 12.303862989 10.0.2.4          10.0.2.5          UDP       111 11223 → 38401 Len=69

Frame 6: 111 bytes on wire (888 bits), 111 bytes captured (888 bits) on interface enp0s3, id 0
Ethernet II, Src: PcsCompu_16:89:3a (08:00:27:16:89:3a), Dst: PcsCompu_f5:94:fc (08:00:27:f5:94:fc)
  Destination: PcsCompu_f5:94:fc (08:00:27:f5:94:fc)
  Source: PcsCompu_16:89:3a (08:00:27:16:89:3a)
  Type: IPv4 (0x0800)
Internet Protocol Version 4, Src: 10.0.2.4, Dst: 10.0.2.5
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 97
  Identification: 0x3a7f (14975)
  Flags: 0x40, Don't fragment
  ...0 0000 0000 0000 = Fragment Offset: 0
  Time to Live: 64
  Protocol: UDP (17)
  Header Checksum: 0xe804 [validation disabled]
  [Header checksum status: Unverified]
  Source Address: 10.0.2.4
  Destination Address: 10.0.2.5
User Datagram Protocol, Src Port: 11223, Dst Port: 38401
0000 08 00 27 f5 94 fc 08 00 27 16 89 3a 08 00 45 00  ..a...:E
0010 00 61 3a 7f 40 00 40 11 e8 04 0a 00 02 04 0a 00  ..a:0:0:.....
0020 02 05 2b d7 96 01 00 4d 18 67 59 6f 75 72 20 49  ..t..M.gYour I
0030 64 27 73 29 61 72 65 3a 0a 41 72 69 65 6c 20 45  d's are: Ariel E
0040 6c 62 61 7a 20 49 44 20 3a 20 32 31 33 30 37 33  lbaz ID : 213073
0050 34 39 37 20 0a 4f 66 65 6b 20 42 69 6e 64 65 72  497 0fe k Binder
0060 20 49 44 20 3a 20 32 31 33 39 30 32 36 33 38      ID : 21 3902638
  
```

זוהי החבילה מן ip 10.0.2.4

אל ip 10.0.2.5

```

Frame 2: 111 bytes on wire (888 bits), 111 bytes captured (888 bits) on interface enp0s3, id 0
Ethernet II, Src: PcsCompu_16:89:3a (08:00:27:16:89:3a), Dst: PcsCompu_f5:94:fc (08:00:27:f5:94:fc)
Internet Protocol Version 4, Src: 10.0.2.4, Dst: 10.0.2.5
    Version: 4 = .... 0100
    Header Length: 20 bytes (5) = 0101 ....
    Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 97
    Identification: 0x3a7f (14975)
    Flags: 0x2, Don't fragment = .... 010
    Fragment Offset: 0 = 0000 0000 0000 0...
    Time to Live: 64
    Protocol: UDP (17)
    Header Checksum: 0xe804 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 10.0.2.4
    Destination Address: 10.0.2.5
User Datagram Protocol, Src Port: 11223, Dst Port: 38401
Data (69 bytes)

```

זוהי החבילה מ 10.0.2.5

אל 10.0.2.4

עתה נראה בעזרת הפקודה ip שהם אכן ה ip של הסרבר ושל הלקוח

```

ariel@ariel-VirtualBox:~$ ip -c a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:f5:94:fc brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.5/24 brd 10.0.2.255 scope global dynamic noprefixroute enp0s3
        valid_lft 467sec preferred_lft 467sec
    inet6 fe80::419c:6fb6:3309:59d2/64 scope link noprefixroute
        valid_lft forever preferred_lft forever

```



```

ariel@ariel-VirtualBox:~$ ip -c a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:16:89:3a brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.4/24 brd 10.0.2.255 scope global dynamic noprefixroute enp0s3
        valid_lft 483sec preferred_lft 483sec
    inet6 fe80::c48e:2a1d:d43d:98e8/64 scope link noprefixroute
        valid_lft forever preferred_lft forever

```

ניתן לראות שבתמונה הראשונה כתובת ה ip היא 10.0.2.5

ובתמונה השנייה כתובת ה ip היא 10.0.2.4

השכבה בה אנו משתמשים בכתובת ה ip היא שכבת הרשת (network) וזה משום בשלב זה יש לנו שימוש בכתובות ה ip של המכשירים השולחים והמקבלים.

שכבת הרשת היא אחת מהשכבות המרכיבות מודל תקשורת (תפקידה זהה ב 2 המודלים המוכרים OSI ו TCP/IP). תפקידה המרכזי של שכבת הרשת הוא לדאוג כיצד החבילה תגיע אל המחשב הסופי. שכבת הרשת יודעת לאיזה מחשב החבילה צריכה להגיע בעזרת כתובות ה ip אשר ייחודיות לכל מחשב (כאשר הן באותה רשת יש דמיון רב ביניהם).

שכבת הרשת מגיעה ישר אחרי שכבת הקו והיא סומכת על שכבה זו שתעשה את עבודתה ותדאג שהחבילה תגיע אל התחנה הקרובה והנכונה ביותר - בדרך אל היעד הסופי. לאחר שכבת הרשת החבילה עוברת לשכבת התעבורה.

המידע בשכבת הרשת נקרא Packet.

חלק ב –

בחלק זה נציג מימוש לצ'אט פשוט בצורה דומה לקבוצת ווטסאפ שבה כל הודעה שמישהו כותב נשלחת לשאר חברי הקבוצה.

כאשר אחד מהלקוחות שולח הודעה לקבוצה השרת מקבל אותה ורק כאשר לקוח נוסף יפנה לשרת השרת יספק עבורו את ההודעות המתאימות בהתאם להודעה שהוא שלח לשרת.

בחלק זה השתמשנו בארבעה מכונות ווירטואליות שונות, עבור שלושה לקוחות: client1, client2, client3 ומכונה אחת עבור השרת – ArielHost.

כאשר יצרנו את המכונות ווידאנו שכל אחת מהמכונות תקבל כתובת MAC שונה ובנוסף גם IP שונה.

את ה IP הגדרנו ע"י כך שבהגדרות קישרנו כל כרטיס רשת של המכונה הווירטואלית NAT-network (NAT הינו פרוטוקול בשכבת הרשת שנלמד עליו בהמשך הקורס).

בתמונה הבאה מסומנות בירוק כתובות ה IP שעובדות בשכבת הרשת ובאדום כתובות ה MAC שעובדות בשכבת הlink וניתן להבחין כי כל הכתובות שונות:

```
ariel@ariel-VirtualBox:~$ ip -c a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 08:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid lft forever preferred_lft forever
2: enp8s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:16:83:30 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic noprefixroute enp8s3
        valid lft 503sec preferred_lft 503sec
    inet6 fe80::c49e:2a1d:8431:9647/64 scope link noprefixroute
        valid lft forever preferred_lft forever

ariel@ariel-VirtualBox:~$

ariel@ariel-VirtualBox:~$ ip -c a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 08:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid lft forever preferred_lft forever
2: enp8s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:16:83:30 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic noprefixroute enp8s3
        valid lft 503sec preferred_lft 503sec
    inet6 fe80::c49e:2a1d:8431:9647/64 scope link noprefixroute
        valid lft forever preferred_lft forever

ariel@ariel-VirtualBox:~$

ariel@ariel-VirtualBox:~$ ip -c a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 08:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid lft forever preferred_lft forever
2: enp8s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:16:83:30 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic noprefixroute enp8s3
        valid lft 497sec preferred_lft 497sec
    inet6 fe80::9502:6430:eabb:b647/64 scope link noprefixroute
        valid lft forever preferred_lft forever

ariel@ariel-VirtualBox:~$

ariel@ariel-VirtualBox:~$ ip -c a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 08:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid lft forever preferred_lft forever
2: enp8s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:16:83:30 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic noprefixroute enp8s3
        valid lft 320sec preferred_lft 320sec
    inet6 fe80::8032:c335:7d16:e853/64 scope link noprefixroute
        valid lft forever preferred_lft forever

ariel@ariel-VirtualBox:~$
```

כעת נריץ את השרת ואת שלושת הלקוחות ונדגים את התהליך בצאת הווטסאפ כאשר במקביל אנו מסניפים את התעבורה במכונה הווירטואלית של השרת באמצעות wireshark.

נציין כעת בקצרה כי הפורט אליו מקושר הסוקט של השרת הינו 12345 ובהמשך נרחיב עליו ועל אופן שימושם בשכבת התעבורה, כאשר נבחר את מס חבילות שהסנפנו ונדגים עליהן.

1. Ariel נרשם
2. Ofek נרשם
3. חמי מנסה להרשם בצורה שגויה
4. אריאל שולח הודעה.
5. תצוגה עד עכשיו:

```
ariel@ariel-VirtualBox:~$ python3 server.py 12345
ariel@ariel-VirtualBox:~$ python3 client.py 10.0.2.4 12345
1 Ariel
2 Hi Ofek !!!
```

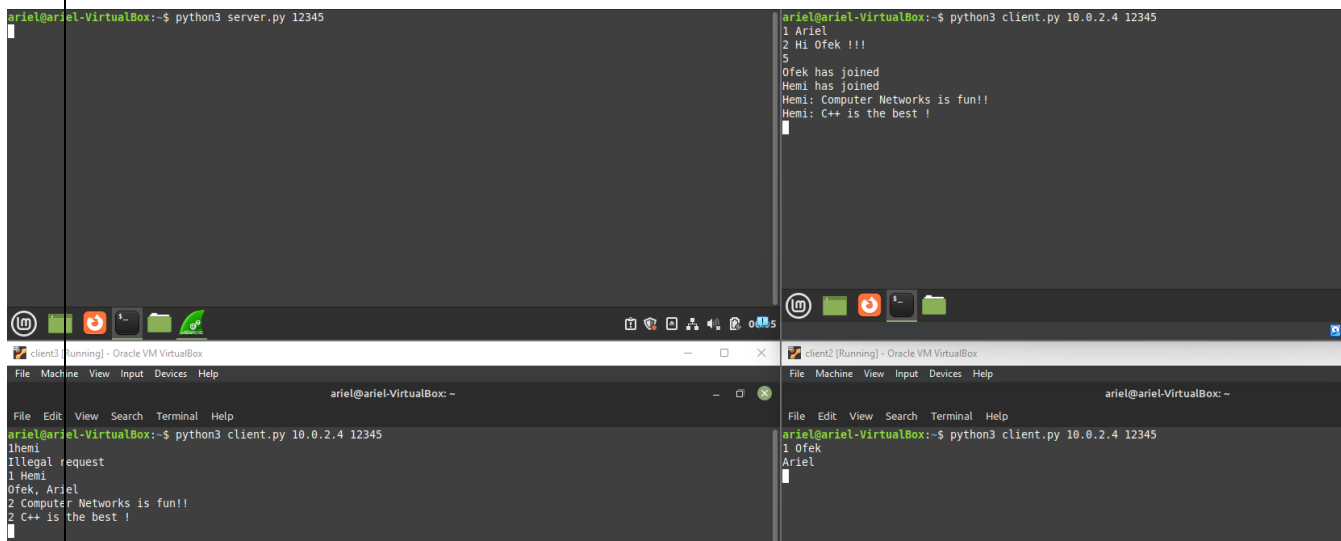
6. חמי הצטרף כמו שצריך :)
7. חמי שולח 2 הודעות לאופק ואריאל.
8. תצוגה עד עכשיו :

```
ariel@ariel-VirtualBox:~$ python3 server.py 12345
ariel@ariel-VirtualBox:~$ python3 client.py 10.0.2.4 12345
1 Ofek
2 Hi Ofek !!!
```

```
ariel@ariel-VirtualBox:~$ python3 client.py 10.0.2.4 12345
1 Ofek
2 Hi Ofek !!!
3 Computer Networks is fun!!
4 C++ is the best !
```

9. אריאל מבקש עדכון מהשרת

10. תצוגה עד עכשיו :



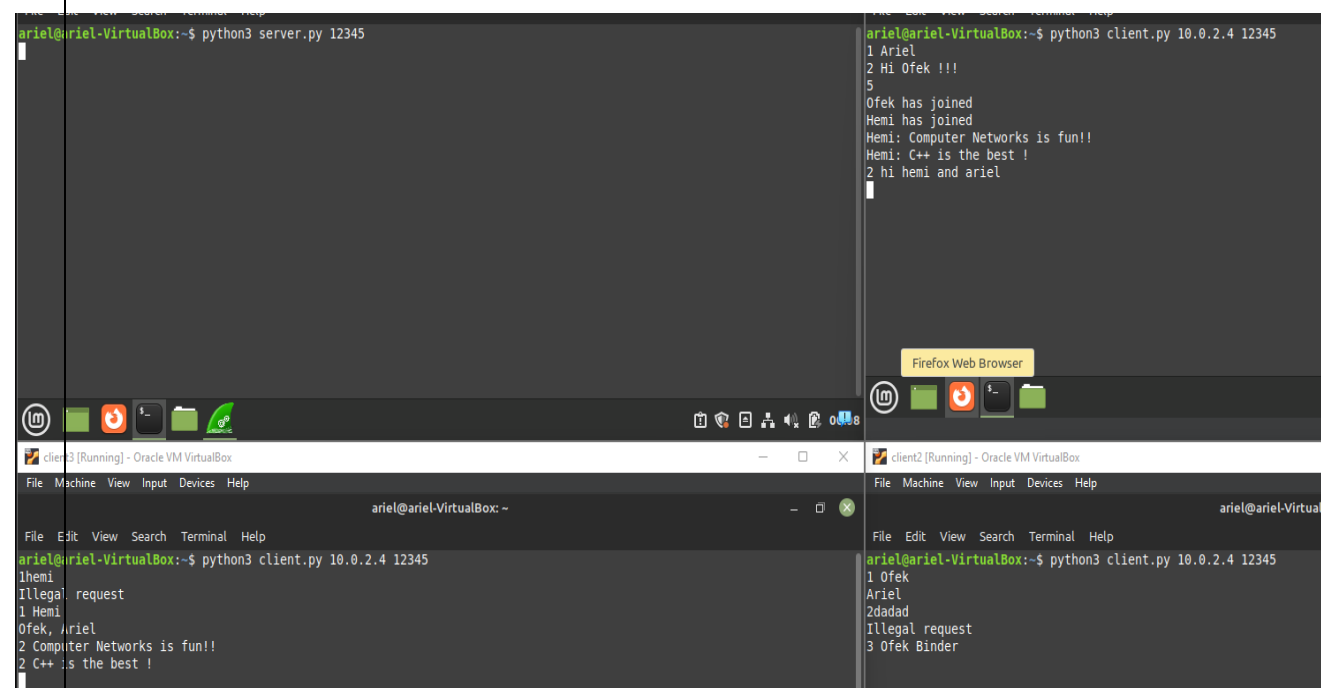
```
ariel@ariel-VirtualBox:~$ python3 server.py 12345
ariel@ariel-VirtualBox:~$ python3 client.py 10.0.2.4 12345
1 Ariel
2 Hi Ofek !!!
5
Ofek has joined
Hemi has joined
Hemi: Computer Networks is fun!!
Hemi: C++ is the best !
```

11. אריאל שולח הודעה

12. אופק שולח הודעה לא תקינה

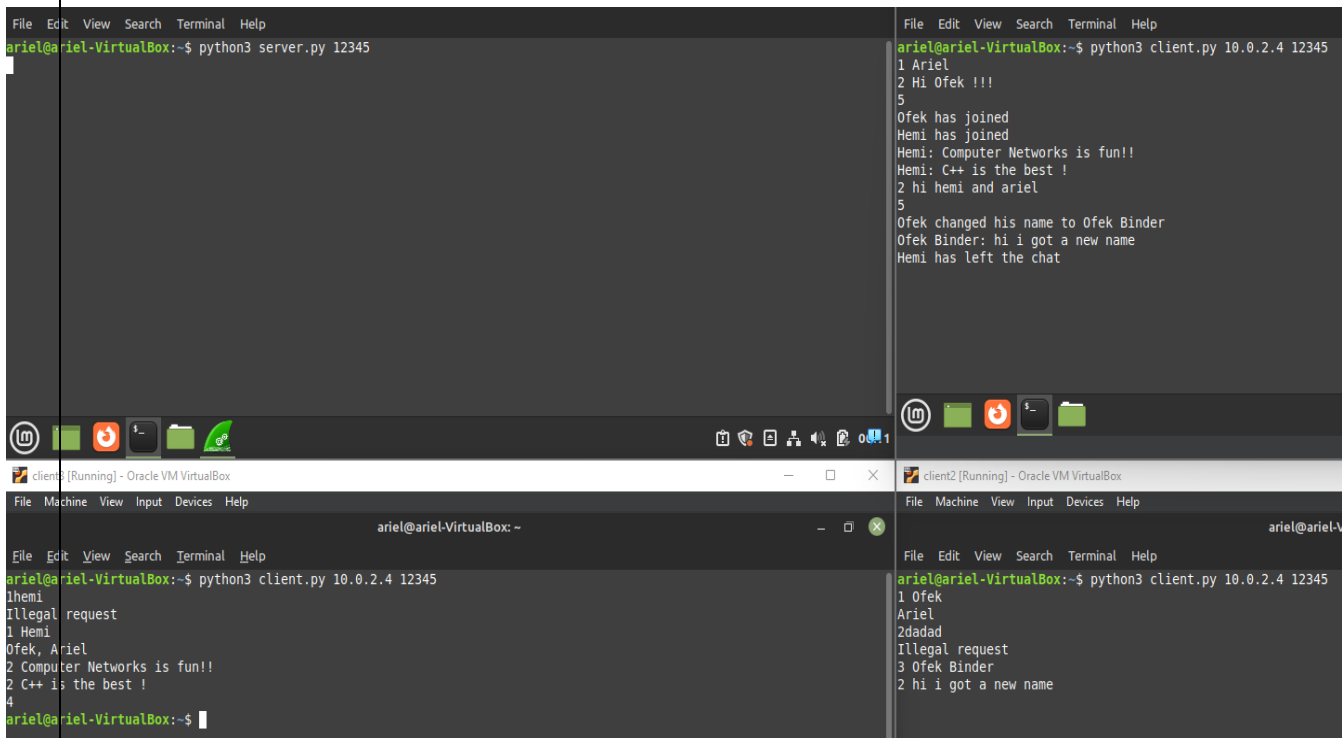
13. אופק משנה שם לאופק בינדר

14. תצוגה עד עכשיו :



```
ariel@ariel-VirtualBox:~$ python3 server.py 12345
ariel@ariel-VirtualBox:~$ python3 client.py 10.0.2.4 12345
1 Ariel
2 Hi Ofek !!!
5
Ofek has joined
Hemi has joined
Hemi: Computer Networks is fun!!
Hemi: C++ is the best !
2 hi hemi and ariel
```

- 15. אופק שולח הודעה
- 16. חמי עוזב את הקבוצה
- 17. אריאל מבקש עדכון
- 18. תצוגה עד עכשיו



```
ariel@ariel-VirtualBox:~$ python3 server.py 12345
ariel@ariel-VirtualBox:~$ python3 client.py 10.0.2.4 12345
1 Ariel
2 Hi Ofek !!!
5
Ofek has joined
Hemi has joined
Hemi: Computer Networks is fun!!
Hemi: C++ is the best !
2 hi hemi and ariel
5
Ofek changed his name to Ofek Binder
Ofek Binder: hi i got a new name
Hemi has left the chat

ariel@ariel-VirtualBox:~$ python3 client.py 10.0.2.4 12345
1 hemi
Illegal request
1 Hemi
Ofek, Ariel
2 Computer Networks is fun!!
2 C++ is the best !
4
ariel@ariel-VirtualBox:~$

ariel@ariel-VirtualBox:~$ python3 client.py 10.0.2.4 12345
1 Ofek
Ariel
2dadad
Illegal request
3 Ofek Binder
2 hi i got a new name
```

עד כה הדגמנו את מהלך התוכניות ואת השימוש בצ'אט , כעת אציג שלוש חבילות שעברו בין הלקוחות לשרת ולהפך.

את חבילות אלו נציג באמצעות התוכנה Wireshark שבאמצעותה הסנפנו את התעבורה וסיננו את החבילות על ידי `serverPort = 12345` ועל ידי כך שהראנו רק חבילות שמוצגות ע"י פרוטוקול התעבורה UDP בדומה לחלק א של הדוח.

חבילה ראשונה:

No.	Time	Source	Destination	Protocol	Len	Info
1	0.00000000	10.0.2.8	10.0.2.4	UDP	70	48921 → 12345 Len=28
2	125.277176494	10.0.2.4	10.0.2.5	UDP	130	12345 → 57025 Len=88
3	456.427095744	10.0.2.4	10.0.2.5	UDP	134	12345 → 57025 Len=92

Frame 1: 70 bytes on wire (560 bits), 70 bytes captured (560 bits) on interface enp0s3, id 0	
• Ethernet II, Src: PcsCompu_af:d3:da (08:00:27:af:d3:da), Dst: PcsCompu_16:89:3a (08:00:27:16:89:3a)	
• Internet Protocol Version 4, Src: 10.0.2.8, Dst: 10.0.2.4	
• User Datagram Protocol, Src Port: 48921, Dst Port: 12345	
• Data (28 bytes)	

ניתן לראות את הפריים כולו לאחר שירד משכבת הרשת

0000	08 00 27 16 89 3a 08 00 27 af d3 da 08 00 45 00E..
0010	00 38 dd 86 40 00 40 11 45 23 0a 00 02 08 0a 00	..8..0..0..E#.....
0020	02 04 bf 19 30 39 00 24 37 3a 32 20 43 6f 6d 7009..\$ 7:2 Comp
0030	75 74 65 72 20 4e 65 74 77 6f 72 6b 73 20 69 73	uter Net works is
0040	20 66 75 6e 21 21	fun!!

נתחיל להסביר את מבנה החבילה ומה קורה ומה עובר בכל שכבה , נתחיל משכבת האפליקציה – הלקוח חמי שולח את ההודעה הבאה לשרת : 2 Computer Networks is fun!!

נסביר איך זה קורה על ידי המימוש של הקוד :

```
while clientSocket:
    msg = input()

    if msg:
        serverIP = sys.argv[1]
        serverPort = int(sys.argv[2])

        clientSocket.sendto(msg.encode(), (serverIP, serverPort))
```

הלקוח חמי פונה אל השרת הפורט והIP של השרת ידועים לו ושולח לו את ההודעה הרלוונטית שחמי הכניס לטרמינל מקודדת.

```
# receive message from client and save the client address - (IP, port).
msg, clientAddress = serverSocket.recvfrom(1024)
OPTION = msg.decode()[0]
```

צד השרת מחכה ללקוח שיפנה אליו ומקבל את המידע ושומר אותו בתוך . msg

ולאחר מכן מחלץ את האפשרות שהלקוח בחר (1-5) נשים לב שאכן ניתן לגשת למקום ה-0 במחרוזת כי בדקנו בצד הלקוח שהיא לא ריקה.

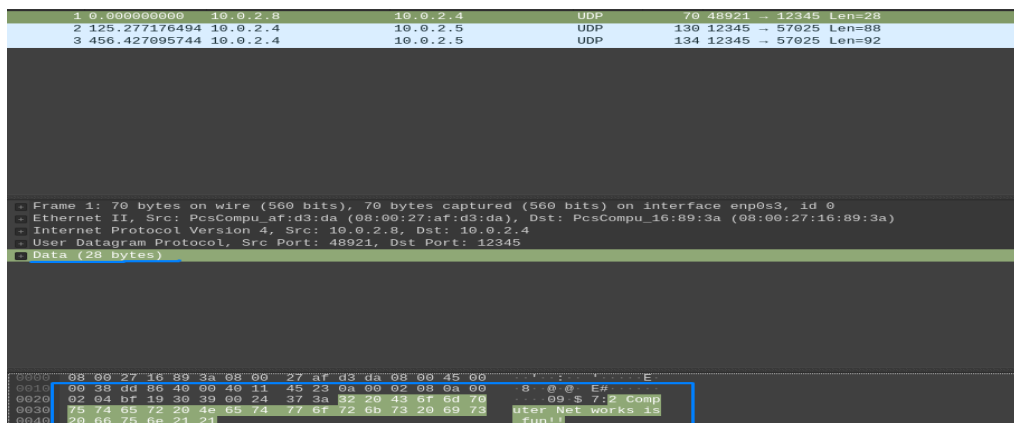
אחר כך בצד השרת נכנס לתנאי בקוד כי 2 = OPTION

```
elif OPTION == '2' and isJoined and len(msg.decode()) >= 3 and str(msg.decode())[1] == ' ':
    clientName = ''
    for nameOfClient, clientMsgDict in clientDict.items():
        if clientMsgDict['client'] == clientAddress:
            clientName = nameOfClient
    nameMsg = clientName + ': ' + str(msg.decode()[2:])
    for client in clientDict.values():
        if client['client'] == clientAddress:
            continue
        # Send to all the clients in the list that the user has joined.
        else:
            client['msgs'].append(nameMsg)
    serverSocket.sendto(bytes('', 'utf-8'), clientAddress)
```

נעבור על כל הלקוחות שנמצאים במילון בצד השרת ונמצא את הלקוח שלנו, מכיוון שידוע לנו רק ה(IP,PORT) של הלקוח או צריכים לחלץ את שמו.

לאחר מכן או עוברים על כל שאר הלקוחות שנמצאים במילון ומוסיפים לרשימה של ההודעות שממתינות להם שחמי שלח את ההודעה בפורמט המתאים (nameMsg).

ניתן לראות את ה data שנשלחה בחבילה בתמונה הבאה:



שכבת התעבורה- כאשר הלקוח משתמש במתודה sendTo הוא מעביר את החבילה עם המידע המתאים עם ה Port כפורט המקור ואת הפורט של הלקוח שנקבע ע"י מערכת ההפעלה במקרה הזה כפורט הלקוח.

כאשר או שולחים את החבילה אל השרת מהלקוח בצד השרת ווידאו שכל חבילה שתשלח אל ServerPort בהכרח תגיע לסוקט שיצרנו בשרת זאת מכיוון שעשינו bind לסוקט של השרת לפורט כמו שניתן לראות בתמונה הבאה:

```
serverPort = int(sys.argv[1])
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', serverPort))
```

ולכן כאשר או בשכבת התעבורה נקבל את DATAGRAM שמכיל את DATA שהועבר משכבת האפליקציה ובנוסף מכיל את פורט המקור (של הלקוח) והיעד (של השרת)

כפי שניתן לראות בתמונה הבאה:

```

+ Frame 1: 70 bytes on wire (560 bits), 70 bytes captured (560 bits) on interface enp0s3, id 0
+ Ethernet II, Src: PcsCompu_af:d3:da (08:00:27:af:d3:da), Dst: PcsCompu_16:89:3a (08:00:27:16:89:3a)
+ Internet Protocol Version 4, Src: 10.0.2.8, Dst: 10.0.2.4
+ User Datagram Protocol, Src Port: 48921, Dst Port: 12345
  Source Port: 48921
  Destination Port: 12345
  Length: 36
  Checksum: 0x373a [unverified]
  [Checksum Status: Unverified]
  [Stream index: 0]
  [Timestamps]
  UDP payload (28 bytes)
  Data (28 bytes)
    Data: 3220436f6d7075746572204e6574776f726b732069732066756e2121
    [Length: 28]
0000  08 00 27 16 89 3a 08 00 27 af d3 da 08 00 45 00  ..8..@.E#...
0010  00 38 dd 86 40 00 40 11 45 23 0a 00 02 08 0a 00  ..8..@.E#...
0020  02 04 bf 19 30 39 00 24 37 3a 32 20 43 6f 6d 70  ..0..@.E#...
0030  75 74 65 72 20 4e 65 74 77 6f 72 6b 73 20 69 73  ..0..@.E#...
0040  20 66 75 6e 21 21                                ..0..@.E#...
          user Net works is fun!!

```

המטען זה המידע שירד משכבת האפליקציה ונוסף עליו גם פורט הלקוח (48921) ופורט השרת (12345).

לאחר מכן אנו יורדים משכבת התעבורה אל שכבת הרשת.

שכבת הרשת -

בשכבת הרשת אנחנו מוסיפים למה שירד משכבת התעבורה את ה IP של הלקוח כמקור ואת ה IP של השרת כיעד.

ראינו שאנחנו בצד הלקוח שולח ל-10.0.2.4 serverip-10.0.2.8 מהאייפי של הלקוח – 10.0.2.8

הערה: כאשר הגדרנו את המכונות הווירטואליות יצרנו NAT network אשר משתמשת בפרוטוקול שעליו נלמד בהמשך מצורף תמונה שמסבירה את ההבדל בינו לבין פרוטוקולי רשת אחרים שמשתמשים בהם במכונות ווירטואליות.

Network type	Access to other VMs	Access to the host	Access to other physical machines	Internet Access
Nat	✗	One way	✗	✓
Nat network	✓	One way	✗	✓
Bridged network	✓	✓	✓	✓

```

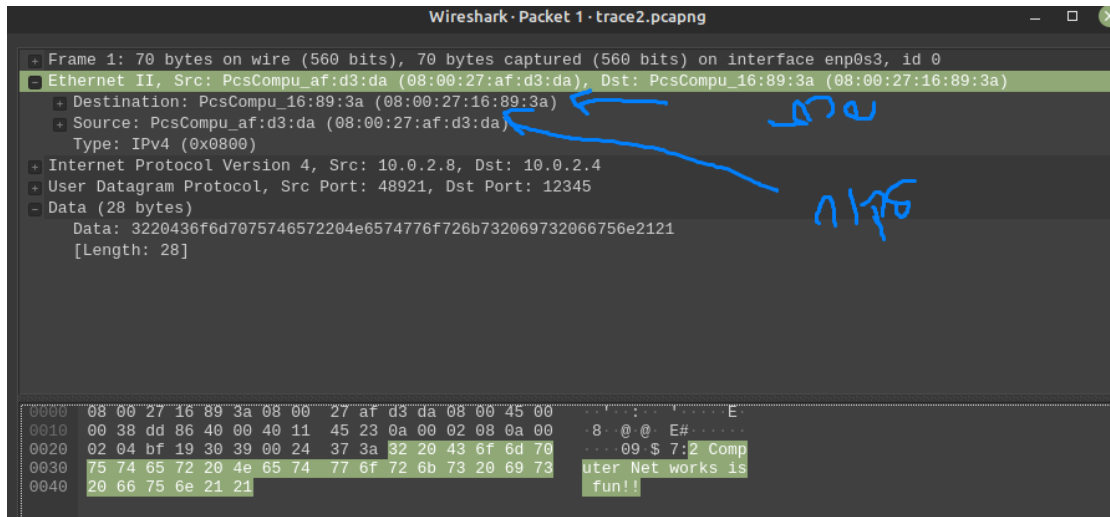
+ Frame 1: 70 bytes on wire (560 bits), 70 bytes captured (560 bits) on interface enp0s3, id 0
+ Ethernet II, Src: PcsCompu_af:d3:da (08:00:27:af:d3:da), Dst: PcsCompu_16:89:3a (08:00:27:16:89:3a)
+ Internet Protocol Version 4, Src: 10.0.2.8, Dst: 10.0.2.4
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  + Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 56
  Identification: 0xdd86 (56710)
  + Flags: 0x40, Don't fragment
  ...0 0000 0000 0000 = Fragment Offset: 0
  Time to Live: 64
  Protocol: UDP (17)
  Header Checksum: 0x4523 [validation disabled]
  [Header checksum status: Unverified]
  Source Address: 10.0.2.8
  Destination Address: 10.0.2.4
0000  08 00 27 16 89 3a 08 00 27 af d3 da 08 00 45 00  ..8..@.E#...
0010  00 38 dd 86 40 00 40 11 45 23 0a 00 02 08 0a 00  ..8..@.E#...
0020  02 04 bf 19 30 39 00 24 37 3a 32 20 43 6f 6d 70  ..0..@.E#...
0030  75 74 65 72 20 4e 65 74 77 6f 72 6b 73 20 69 73  ..0..@.E#...
0040  20 66 75 6e 21 21                                ..0..@.E#...
          user Net works is fun!!

```


שכבת הקישור – לאחר שסיימנו עם שכבת הרשת אנו נרד לשכבת הקישור אשר משתמשת בפרוטוקול Ethernet 2.

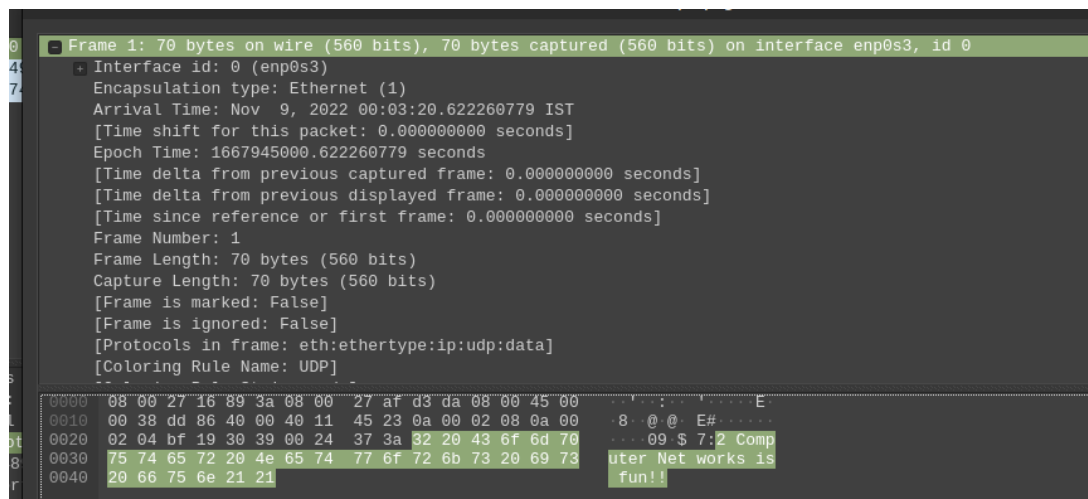
שכבה זו תוסיף את כתובות ה MAC לחבילה ובאמצעות כך נדע לאיזה כרטיס רשת לגשת בצד השרת כי ייתכן שיש באותה רשת כמה מחשבים שונים וכמה כרטיסי רשת.

כתובות ה MAC הן כתובות פיזיות! לעומת כתובות ה IP שהן כתובות לוגיות.



שכבת פיזית -

לבסוף נרד לשכבת ה Pysical שבה אנו מעבירים את כל הפריים כולו "באופן פיזי" ולאחר מכן כאשר הפריים יגיע לשרת הוא יתחיל מהשכבה הזאת ולאט לאט יפרק את החבילה בהתאם למה שיש בתוכה, הוא ידע מי שלח את החבילה ובאילו פרוטוקולים להשתמש.



לסיכום, נשים לב כי השכבות הנן נפרדות ובלתי תלויות אחת בשנייה וכל שכבה מוסיפה על השכבה האחרת את המידע הנדרש בלי קשר למידע שבאה מהשכבה האחרת.

חבילה שנייה:

.No	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.2.8	10.0.2.4	UDP	70	Len=28 12345 → 48921
2	125.277176494	10.0.2.4	10.0.2.5	UDP	130	Len=88 57025 → 12345
3	456.427095744	10.0.2.4	10.0.2.5	UDP	134	Len=92 57025 → 12345

Frame 2: 130 bytes on wire (1040 bits), 130 bytes captured (1040 bits) on interface enp0s3, id 0
 Ethernet II, Src: PcsCompu_16:89:3a (08:00:27:16:89:3a), Dst: PcsCompu_f5:94:fc (08:00:27:f5:94:fc)
 Internet Protocol Version 4, Src: 10.0.2.4, Dst: 10.0.2.5
 User Datagram Protocol, Src Port: 12345, Dst Port: 57025
 Data (88 bytes)

```

0000 08 00 27 f5 94 fc 08 00 27 16 89 3a 08 00 45 00  ..E:
0010 00 74 b3 db 40 00 40 11 6e 95 0a 00 02 04 0a 00  -t: @ n
0020 02 05 30 39 de c1 00 60 18 7a 4f 66 65 6b 20 68  ..09...zOfek h
0030 61 73 20 6a 6f 69 6e 65 64 0a 48 65 6d 69 20 68  as joine d:Hemi h
0040 61 73 20 6a 6f 69 6e 65 64 0a 48 65 6d 69 20 68  as joine d:Hemi:
0050 43 6f 6d 70 75 74 65 72 20 4e 65 74 77 6f 72 6b  Computer Network
0060 73 20 69 73 20 66 75 6e 21 21 0a 48 65 6d 69 3a  s is fun !!:Hemi:
0070 20 43 2b 2b 20 69 73 20 74 68 65 20 62 65 73 74  C++ is the best
0080 20 21
  
```

המידע הנשלח בהודעה

ניתן לראות בתמונות צילום של החבילה מן ה wire shark ועתה נסביר על מבנה החבילה ועל הדרך בה הוא מתבטא בקוד. נתחיל משכבת האפליקציות- השרת קיבל פקודה להחזיר עדכון של התצוגה עד עכשיו ולכן הוא שולח את ההודעות והעדכונים שהיו עד כה. ניתן לראות את המידע המועבר בתמונה המצורפת.

נראה איך זה קורה באמצעות הקוד-

```

elif OPTION == SHOW_UPDATES and isJoined and len(str(msg.decode())) == 1:
    for client in clientDict.values():
        if client['client'] == clientAddress:
            # if the client don't have msgs
            if not client['msgs']:
                serverSocket.sendto(bytes('', 'utf-8'), clientAddress)
                continue
            clientMsgs = '\n'.join(str(e) for e in client['msgs'])
            serverSocket.sendto(bytes(clientMsgs, 'utf-8'), clientAddress)
            client['msgs'] = []
  
```

זהו הקוד של השרת במצב שהוא קיבל מהלקוח את הפקודה 5 (אשר דורשת להחזיר עדכון על כל ההודעות עד כה) ובמצב זה הוא בודק את כל ההודעות שנשלחו עד כה ושומר אותם תחת המשתנה clientMsgs ולאחר מכן השרת שולח בעזרת פקודת ה sentto את clientMsgs אל הלקוח אשר פרטיו נשמרו לנו בעת שליחת ההודעה הראשונית על ידו. בנוסף הוא מוחק את שמירת ההודעות שנשלחו עד כה (משום שעדכון מתבצע רק על מה שנשלח מאז העדכון הקודם)

לאחר ששלחנו את המידע המידע נשלח (בהמשך יוסבר הליך השליחה לפי השכבות) השרת ממשיך לרוץ לנצח ולחכות להודעות הבאות.

עתה נראה מה קורה בקוד הלקוח שמקבל את החבילה-

```
data, serverAddress = clientSocket.recvfrom(2048)
if str(data.decode()) == '' and msg == '4':
    clientSocket.close()
    break
elif str(data.decode()) == '':
    continue
print(str(data.decode()))
else:
    print('Illegal request')
```

הלקוח מקבל את החבילה בפקודה `recvfrom` וכך הוא מקבל 2 משתנים אחד `data` המחזיק את המידע שנשלח והשני `serverAddress` המחזיק את המידע של השולח (השרת) בטופאל המכיל את ה `ip` וה `port` של השרת.

לאחר הקבלה הלקוח נכנס לשורה בה מדפיסים את הדאטה שקיבלנו (לאחר המרתו מביטים) וכך הדפסנו את היסטוריית ההודעות.

שכבת התעבורה-

בעזרת פקודת ה `send to` השרת מעביר את החבילה שלנו עם פורט המקור כפורט שלו (הפורט הוא 12345) ובפורט היעד נשים את פורט הלקוח אליו אנחנו שולחים אותו קיבלנו בעת קבלת ההודעה הראשונית מן הלקוח בעזרת פקודת ה `recvfrom` (כרגע עבורנו הפורט הוא 57025) כרגע המידע נמצא בטופל יחד עם ה `ip` של הלקוח תחת המשתנה `clientAddress`.

```
while True:
    # receive message from client and save the client address - (IP, port).
    msg, clientAddress = serverSocket.recvfrom(1024)
```

בצד הלקוח אליו אנחנו שולחים את החבילה מערכת ההפעלה ביצעה `bind` אוטומטית למספר פורט מסויים שהוא המספר שנשלח לשרת בשליחה הראשונית.

בנוסף בשכבת התעבורה נקבל `DATAGRAM` שמכיל את `DATA` שהועבר משכבת האפליקציה ובנוסף מכיל את פורט המקור(של השרת) והיעד(של הלקוח) כפי שניתן לראות בתמונה הבאה-

```
Frame 2: 130 bytes on wire (1040 bits), 130 bytes captured (1040 bits) on interface enp0s3, id 0
Ethernet II, Src: PcsCompu_16:89:3a (08:00:27:16:89:3a), Dst: PcsCompu_f5:94:fc (08:00:27:f5:94:fc)
Internet Protocol Version 4, Src: 10.0.2.4, Dst: 10.0.2.5
User Datagram Protocol, Src Port: 12345, Dst Port: 57025
    Source Port: 12345
    Destination Port: 57025
    Length: 96
    Checksum: 0x187a [unverified]
    [Checksum Status: Unverified]
    [Stream index: 1]
    [Timestamps]
    UDP payload (88 bytes)
    Data (88 bytes)
```

0000	08 00 27 f5 94 fc 08 00 27 16 89 3a 08 00 45 00	..'. '...:..E.
0010	00 74 b3 db 40 00 40 11 6e 95 0a 00 02 04 0a 00	.t..@..@. n.....
0020	02 05 30 39 de c1 00 60 18 7a 4f 66 65 6b 20 68	..09...`..zOfek h
0030	61 73 20 6a 6f 69 6e 65 64 0a 48 65 6d 69 20 68	as joine d·Hemi h
0040	61 73 20 6a 6f 69 6e 65 64 0a 48 65 6d 69 3a 20	as joine d·Hemi:
0050	43 6f 6d 70 75 74 65 72 20 4e 65 74 77 6f 72 6b	Computer Network
0060	73 20 69 73 20 66 75 6e 21 21 0a 48 65 6d 69 3a	s is fun !!·Hemi:
0070	20 43 2b 2b 20 69 73 20 74 68 65 20 62 65 73 74	C++ is the best
0080	20 21	!

עתה אנחנו נרד אל שכבת הרשת-

תפקידה של שכבת הרשת הוא לדאוג כיצד החבילה תגיע אל המחשב הסופי בפועל שכבת הרשת מוסיפה לחבילה שהגיעה אליה מן שכבת התעבורה את מספר ה קו של מחשב המקור (במקרה שלנו השרת בעל קו 10.0.4) ואת מספר ה קו של מחשב היעד (במקרה שלנו הלקוח שמספרו 10.0.4) כך החבילה יודעת לאיזה מחשב יעד סופי היא אמורה להגיע.

Frame 2: 130 bytes on wire (1040 bits), 130 bytes captured (1040 bits) on interface enp0s3, id 0	
Ethernet II, Src: PcsCompu_16:89:3a (08:00:27:16:89:3a), Dst: PcsCompu_f5:94:fc (08:00:27:f5:94:fc)	
Internet Protocol Version 4, Src: 10.0.2.4, Dst: 10.0.2.5	
Version: 4 = 0100	
Header Length: 20 bytes (5) = 0101	
Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)	
Total Length: 116	
Identification: 0xb3db (46043)	
Flags: 0x2, Don't fragment = 010	
Fragment Offset: 0 = 0000 0000 0000 0...	
Time to Live: 64	
Protocol: UDP (17)	
Header Checksum: 0x6e95 [validation disabled]	
[Header checksum status: Unverified]	
Source Address: 10.0.2.4	
Destination Address: 10.0.2.5	
User Datagram Protocol, Src Port: 12345, Dst Port: 57025	
Data (88 bytes)	

0000	08 00 27 f5 94 fc 08 00 27 16 89 3a 08 00 45 00	..'. '...:..E.
0010	00 74 b3 db 40 00 40 11 6e 95 0a 00 02 04 0a 00	.t..@..@. n.....
0020	02 05 30 39 de c1 00 60 18 7a 4f 66 65 6b 20 68	..09...`..zOfek h
0030	61 73 20 6a 6f 69 6e 65 64 0a 48 65 6d 69 20 68	as joine d·Hemi h
0040	61 73 20 6a 6f 69 6e 65 64 0a 48 65 6d 69 3a 20	as joine d·Hemi:
0050	43 6f 6d 70 75 74 65 72 20 4e 65 74 77 6f 72 6b	Computer Network
0060	73 20 69 73 20 66 75 6e 21 21 0a 48 65 6d 69 3a	s is fun !!·Hemi:
0070	20 43 2b 2b 20 69 73 20 74 68 65 20 62 65 73 74	C++ is the best
0080	20 21	!

מצורפת תמונה עם צילום מהוויר שארק המתארת את המידע שקיבלנו בשכבת הרשת. ניתן לראות מסומן בכחול את מספרי ה קו של המקור (השרת) ושל היעד (לקוח).

עתה לאחר מכן נרד אל שכבת הקישור

שכבת הקישור

אל שכבה זאת הגענו לאחר שכבת הרשת.

בשכבה זו אנו נשתמש בפרוטוקול Ethernet 2.

שכבה זו תוסיף את כתובות ה-mac המתאימות לחבילה. כלומר היא תוסיף את כתובת ה-mac של המקור שבמקרה זה הוא השרת שלנו ומוסיף גם את כתובת ה-mac של היעד.

בעזרת כתובת ה-mac נדע לאיזה כרטיס רשת צריך לגשת בצד השני (משום שיכול להיות ברשת אחת מספר רב של מחשבים וכרטיסים)

חשוב לציין כתובות ה-mac הן כתובות פיזיות של כל רכיב (כרטיס רשת), אשר מוטבעות עליו בעת ייצור המכשיר וזאת לעומת כתובות ip אשר יכולות להשתנות בהתאם לרשת בו מחובר המכשיר.

נצרך תמונה מן הוויר שארק ונסמן בכחול את ה-mac היעד והמקור.

```
Frame 2: 130 bytes on wire (1040 bits), 130 bytes captured (1040 bits) on interface enp0s3, id 0 <
Ethernet II, Src: PcsCompu_16:89:3a (08:00:27:16:89:3a), Dst: PcsCompu_f5:94:fc (08:00:27:f5:94:fc) >
    Destination: PcsCompu_f5:94:fc (08:00:27:f5:94:fc) <
    Source: PcsCompu_16:89:3a (08:00:27:16:89:3a) <
    Type: IPv4 (0x0800)
        Internet Protocol Version 4, Src: 10.0.2.4, Dst: 10.0.2.5 <
        User Datagram Protocol, Src Port: 12345, Dst Port: 57025 <
        Data (88 bytes) <
```

0000	08 00 27 f5 94 fc 08 00 27 16 89 3a 08 00 45 00	..'. '...E.
0010	00 74 b3 db 40 00 40 11 6e 95 0a 00 02 04 0a 00	.t..@.@. n.....
0020	02 05 30 39 de c1 00 60 18 7a 4f 66 65 6b 20 68	..09...` .zOfek h
0030	61 73 20 6a 6f 69 6e 65 64 0a 48 65 6d 69 20 68	as joine d·Hemi h
0040	61 73 20 6a 6f 69 6e 65 64 0a 48 65 6d 69 3a 20	as joine d·Hemi:
0050	43 6f 6d 70 75 74 65 72 20 4e 65 74 77 6f 72 6b	Computer Network
0060	73 20 69 73 20 66 75 6e 21 21 0a 48 65 6d 69 3a	s is fun !!·Hemi:
0070	20 43 2b 2b 20 69 73 20 74 68 65 20 62 65 73 74	C++ is the best
0080	20 21	!

שכבת פיזית -

בסיום לאחר מעבר על כל השכבות אנו נרד אל השכבה הפיזית. שכה זו מעבירה את כל הפריים (חבילה שלנו) בצורה מעשית ולאחר שנגיע אל היעד (במקרה שלנו הלקוח שלנו) הלקוח יחל לפרק את החבילה וזאת לפי סדר השכבות וכל פעם יפרק מידע אחר (מהשכבה הפיזית לשכבת הקשר לשכבת הרשת וכו.). ולאחר הפירוק הלקוח ידע מי שלח את החבילה ומה יש בתוכה.

נצרך צילום של הפריים הכולל של החבילה (מן הוויר שארק) שעובר בשכבה הפיזית

```
Frame 2: 130 bytes on wire (1040 bits), 130 bytes captured (1040 bits) on interface enp0s3, id 0
    Section number: 1
        Interface id: 0 (enp0s3)
        Encapsulation type: Ethernet (1)
        Arrival Time: Nov 9, 2022 00:05:25.899437273 Jerusalem Standard Time
            [Time shift for this packet: 0.000000000 seconds]
            Epoch Time: 1667945125.899437273 seconds
            [Time delta from previous captured frame: 125.277176494 seconds]
            [Time delta from previous displayed frame: 125.277176494 seconds]
            [Time since reference or first frame: 125.277176494 seconds]
        Frame Number: 2
        Frame Length: 130 bytes (1040 bits)
        Capture Length: 130 bytes (1040 bits)
        [Frame is marked: False]
        [Frame is ignored: False]
        [Protocols in frame: eth:ethertype:ip:udp:data]
        [Coloring Rule Name: UDP]
        [Coloring Rule String: udp]
    Ethernet II, Src: PcsCompu_16:89:3a (08:00:27:16:89:3a), Dst: PcsCompu_f5:94:fc (08:00:27:f5:94:fc)
    Internet Protocol Version 4, Src: 10.0.2.4, Dst: 10.0.2.5
    User Datagram Protocol, Src Port: 12345, Dst Port: 57025
    Data (88 bytes)
```

0000	08 00 27 f5 94 fc 08 00 27 16 89 3a 08 00 45 00	..'. '...E.
0010	00 74 b3 db 40 00 40 11 6e 95 0a 00 02 04 0a 00	.t..@. @. n.....
0020	02 05 30 39 de c1 00 60 18 7a 4f 66 65 6b 20 68	..09...` .zOfek h
0030	61 73 20 6a 6f 69 6e 65 64 0a 48 65 6d 69 20 68	as joine d.Hemi h
0040	61 73 20 6a 6f 69 6e 65 64 0a 48 65 6d 69 3a 20	as joine d.Hemi:
0050	43 6f 6d 70 75 74 65 72 20 4e 65 74 77 6f 72 6b	Computer Network
0060	73 20 69 73 20 66 75 6e 21 21 0a 48 65 6d 69 3a	s is fun !!.Hemi:
0070	20 43 2b 2b 20 69 73 20 74 68 65 20 62 65 73 74	C++ is the best
0080	20 21	!

ניתן לראות שכל שכבה הוסיפה לחבילה מידע שימושי ונוסף על מנת שנוכל להעביר את המידע בבטחה ובדיוק מירבי.

חבילה שלישית:

ראשית נצרף תמונה מן הוויר שארק של החבילה ואז נפרק את החבילה שכבה שכבה כפי שביצענו קודם.

	Info	Length	Protocol	Destination	Source	Time	.No
	Len=28 12345 → 48921 70		UDP	10.0.2.4	10.0.2.8	0.000000000	1
	Len=88 57025 → 12345 130		UDP	10.0.2.5	10.0.2.4	125.277176494	2
	Len=92 57025 → 12345 134		UDP	10.0.2.5	10.0.2.4	456.427095744	3

Frame 3: 134 bytes on wire (1072 bits), 134 bytes captured (1072 bits) on interface enp0s3, id 0
Ethernet II, Src: PcsCompu_16:89:3a (08:00:27:16:89:3a), Dst: PcsCompu_f5:94:fc (08:00:27:f5:94:fc)
Internet Protocol Version 4, Src: 10.0.2.4, Dst: 10.0.2.5
User Datagram Protocol, Src Port: 12345, Dst Port: 57025
Data (92 bytes)

0000	08 00 27 f5 94 fc 08 00 27 16 89 3a 08 00 45 00	..'. '...:..E.
0010	00 78 90 6c 40 00 40 11 92 00 0a 00 02 04 0a 00	..x.l@.@.
0020	02 05 30 39 de c1 00 64 18 7e 4f 66 65 6b 20 63	..09...d ~Ofek c
0030	68 61 6e 67 65 64 20 68 69 73 20 6e 61 6d 65 20	hanged h is name
0040	74 6f 20 4f 66 65 6b 20 42 69 6e 64 65 72 0a 4f	to Ofek Binder·O
0050	66 65 6b 20 42 69 6e 64 65 72 3a 20 68 69 20 69	fek Bind er: hi i
0060	20 67 6f 74 20 61 20 6e 65 77 20 6e 61 6d 65 0a	got a n ew name·
0070	48 65 6d 69 20 68 61 73 20 6c 65 66 74 20 74 68	Hemi has left th
0080	65 20 63 68 61 74	e chat

המידע הנשלח בהודעה

בתמונה הראשונה רואים את שלוש החבילות שעברנו עליהם וסימנו בכחול את החבילה שעליה עבדנו. 2 התמונות האחרות הם המידע הכללי על החבילה מן הוויר שארק.

בחבילה זו אנו נמצאים לאחר שהשרת קיבל פקודה 5 שמטרתה היא שהשרת ישלח ללקוח עדכון על ההודעות שנשלחו מאז העדכון האחרון.

עתה נראה את הקוד עבור כך.

ראשית נתחיל בשרת ששולח את החבילה-

```
elif OPTION == SHOW_UPDATES and isJoined and len(str(msg.decode())) == 1:
    for client in clientDict.values():
        if client['client'] == clientAddress:
            # if the client don't have msgs
            if not client['msgs']:
                serverSocket.sendto(bytes('', 'utf-8'), clientAddress)
                continue
            clientMsgs = '\n'.join(str(e) for e in client['msgs'])
            serverSocket.sendto(bytes(clientMsgs, 'utf-8'), clientAddress)
            client['msgs'] = []
        else:
            serverSocket.sendto(bytes('Illegal request', 'utf-8'), clientAddress)
```

השרת קיבל את הפקודה לשלוח עדכון של כל ההודעות שנשלחו עד כה אז הוא עובר על רשימת ההודעות שנשלחו עד כה ומכניס את כל ההודעות תחת משתנה אחד עם רווח של שורה בין כל אחת (ובנוסף מוחק את ההודעות שהיו שמורות עד כה כדי שבעדכון הבא נקבל את ההודעות רק מהעדכון)

הנוכחי). לאחר מכן השרת שולח את המשתנה הזה אל הלקוח בעזרת הפקודה sendto לשרת יש את פרטי הלקוח (ip ו port) שהתקבלו כשהוא קיבל את ההודעה הקודמת מן הלקוח.

לאחר מכן שהלקוח מקבל את החבילה הוא פועל בצורה הבאה-

```
data, serverAddress = clientSocket.recvfrom(2048)
if str(data.decode()) == '' and msg == '4':
    clientSocket.close()
    break
elif str(data.decode()) == '':
    continue
print(str(data.decode()))
```

הלקוח מקבל את החבילה בעזרת הפקודה recvfrom ובכך מקבל גם את תוכן ההודעה (תחת המשתנה data) וגם את פרטי השולח שהוא השרת במקרה שלנו. עתה בלקוח הקוד נכנס לשורה של print ולכן הוא מדפיס את המידע שקיבלנו מן השרת (מומר מביטים).

שכבת התעבורה-

בעזרת פקודת ה send to השרת מעביר את החבילה שלנו עם פורט המקור כפורט שלו (הפורט הוא 12345) ובפורט היעד נשים את פורט הלקוח אליו אנחנו שולחים אותו קיבלנו בעת קבלת ההודעה הראשונית מן הלקוח בעזרת פקודת ה recvfrom (כרגע עבורנו הפורט הוא 57025) כרגע המידע נמצא בטופל יחד עם ה ip של הלקוח תחת המשתנה clientAdress.

```
msg, clientAddress = serverSocket.recvfrom(1024)
```

בצד הלקוח אליו אנחנו שולחים את החבילה מערכת ההפעלה ביצעה bind אוטומטית למספר פורט מסויים שהוא המספר שנשלח לשרת בשליחה הראשונית.

בנוסף בשכבת התעבורה נקבל DATAGRAM שמכיל את DATA שהועבר משכבת האפליקציה ובנוסף מכיל את פורט המקור(של השרת) והיעד(של הלקוח) כפי שניתן לראות בתמונה הבאה-

```
Frame 3: 134 bytes on wire (1072 bits), 134 bytes captured (1072 bits) on interface enp0s3, id 0
Ethernet II, Src: PcsCompu_16:89:3a (08:00:27:16:89:3a), Dst: PcsCompu_f5:94:fc (08:00:27:f5:94:fc)
Internet Protocol Version 4, Src: 10.0.2.4, Dst: 10.0.2.5
User Datagram Protocol, Src Port: 12345, Dst Port: 57025
```

```
Source Port: 12345
Destination Port: 57025
Length: 100
Checksum: 0x187e [unverified]
[Checksum Status: Unverified]
[Stream index: 1]
[Timestamps]
UDP payload (92 bytes)
```

Data (92 bytes)

0000	08 00 27 f5 94 fc 08 00 27 16 89 3a 08 00 45 00	..'. 'E.
0010	00 78 90 6c 40 00 40 11 92 00 0a 00 02 04 0a 00	.x.l@.@.
0020	02 05 30 39 de c1 00 64 18 7e 4f 66 65 6b 20 63	..09...d..Ofek c
0030	68 61 6e 67 65 64 20 68 69 73 20 6e 61 6d 65 20	hanged h is name
0040	74 6f 20 4f 66 65 6b 20 42 69 6e 64 65 72 0a 4f	to Ofek Binder.O
0050	66 65 6b 20 42 69 6e 64 65 72 3a 20 68 69 20 69	fek Bind er: hi i
0060	20 67 6f 74 20 61 20 6e 65 77 20 6e 61 6d 65 0a	got a n ew name.
0070	48 65 6d 69 20 68 61 73 20 6c 65 66 74 20 74 68	Hemi has left th
0080	65 20 63 68 61 74	e chat

עתה לאחר שצירפנו את הפורטים נרד אל שכבת הרשת-

שכבת הרשת- תפקידה של שכבת הרשת הוא לדאוג כיצד החבילה תגיע אל המחשב הסופי בפועל.

שכבת הרשת מוסיפה לחבילה שהגיעה אליה מן שכבת התעבורה את מספר ה ip של מחשב המקור (במקרה שלנו השרת בעל ip 10.0.4) ואת מספר ה ip של מחשב היעד (במקרה שלנו הלקוח שמספרו 10.0.4) כך החבילה יודעת לאיזה מחשב יעד סופי היא אמורה להגיע. (נסמן בכחול את קו היעד והמקור)

```
Frame 3: 134 bytes on wire (1072 bits), 134 bytes captured (1072 bits) on interface enp0s3, id 0
Ethernet II, Src: PcsCompu_16:89:3a (08:00:27:16:89:3a), Dst: PcsCompu_f5:94:fc (08:00:27:f5:94:fc)
Internet Protocol Version 4, Src: 10.0.2.4, Dst: 10.0.2.5
    Version: 4 = .... 0100
    Header Length: 20 bytes (5) = 0101 ....
    Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
        Total Length: 120
        Identification: 0x906c (36972)
        Flags: 0x2, Don't fragment = .... 010
        Fragment Offset: 0 = 0000 0000 0000 0...
        Time to Live: 64
        Protocol: UDP (17)
        Header Checksum: 0x9200 [validation disabled]
            [Header checksum status: Unverified]
            Source Address: 10.0.2.4
            Destination Address: 10.0.2.5
User Datagram Protocol, Src Port: 12345, Dst Port: 57025
    Data (92 bytes)
```

0000	08 00 27 f5 94 fc 08 00 27 16 89 3a 08 00 45 00	..'. '...:..E.
0010	00 78 90 6c 40 00 40 11 92 00 0a 00 02 04 0a 00	.x.l@. @.
0020	02 05 30 39 de c1 00 64 18 7e 4f 66 65 6b 20 63	..09...d ~Ofek c
0030	68 61 6e 67 65 64 20 68 69 73 20 6e 61 6d 65 20	hanged h is name
0040	74 6f 20 4f 66 65 6b 20 42 69 6e 64 65 72 0a 4f	to Ofek Binder.0
0050	66 65 6b 20 42 69 6e 64 65 72 3a 20 68 69 20 69	fek Bind er: hi i
0060	20 67 6f 74 20 61 20 6e 65 77 20 6e 61 6d 65 0a	got a n ew name.
0070	48 65 6d 69 20 68 61 73 20 6c 65 66 74 20 74 68	Hemi has left th
0080	65 20 63 68 61 74	e chat

עתה לאחר שסיימנו את שכבת הרשת נרד לשכבת הקישור-

שכבת הקישור

אל שכבה זאת הגענו לאחר שכבת הרשת.

בשכבה זו אנו נשתמש בפרוטוקול Ethernet 2.

שכבה זו תוסיף את כתובות ה-mac המתאימות לחבילה. כלומר היא תוסיף את כתובת ה-mac של המקור שבמקרה זה הוא השרת שלנו ומוסיף גם את כתובת ה-mac של היעד.

בעזרת כתובת ה-mac נדע לאיזה כרטיס רשת צריך לגשת בצד השני (משום שיכול להיות ברשת אחת מספר רב של מחשבים וכרטיסים)

חשוב לציין כתובות ה-mac הן כתובות פיזיות של כל רכיב (כרטיס רשת), אשר מוטבעות עליו בעת ייצור המכשיר וזאת לעומת כתובות ip אשר יכולות להשתנות בהתאם לרשת בו מחובר המכשיר.

נצרך תמונה מן הוויר שארק ונסמן בכחול את ה-mac היעד והמקור –

```
Frame 3: 134 bytes on wire (1072 bits), 134 bytes captured (1072 bits) on interface enp0s3, id 0 <
Ethernet II, Src: PcsCompu_16:89:3a (08:00:27:16:89:3a), Dst: PcsCompu_f5:94:fc (08:00:27:f5:94:fc) <
    Destination: PcsCompu_f5:94:fc (08:00:27:f5:94:fc) <
    Source: PcsCompu_16:89:3a (08:00:27:16:89:3a) <
    Type: IPv4 (0x0800)
Internet Protocol Version 4, Src: 10.0.2.4, Dst: 10.0.2.5 <
User Datagram Protocol, Src Port: 12345, Dst Port: 57025 <
Data (92 bytes) <
```

0000	08 00 27 f5 94 fc 08 00 27 16 89 3a 08 00 45 00	..'. ' . . . : . . E .
0010	00 78 90 6c 40 00 40 11 92 00 0a 00 02 04 0a 00	. x . l @ . @
0020	02 05 30 39 de c1 00 64 18 7e 4f 66 65 6b 20 63	. . 09 . . . d . ~ Ofek c
0030	68 61 6e 67 65 64 20 68 69 73 20 6e 61 6d 65 20	hanged h is name
0040	74 6f 20 4f 66 65 6b 20 42 69 6e 64 65 72 0a 4f	to Ofek Binder . O
0050	66 65 6b 20 42 69 6e 64 65 72 3a 20 68 69 20 69	fek Bind er: hi i
0060	20 67 6f 74 20 61 20 6e 65 77 20 6e 61 6d 65 0a	got a n ew name .
0070	48 65 6d 69 20 68 61 73 20 6c 65 66 74 20 74 68	Hemi has left th
0080	65 20 63 68 61 74	e chat

עתה נרד אל השכבה הפיזית

שכבת פיזית -

בסיום לאחר מעבר על כל השכבות אנו נרד אל השכבה הפיזית. שכה זו מעבירה את כל הפריים (חבילה שלנו) בצורה מעשית ולאחר שנגיע אל היעד (במקרה שלנו הלקוח שלנו) הלקוח יחל לפרק את החבילה וזאת לפי סדר השכבות וכל פעם יפרק מידע אחר (מהשכבה הפיזית לשכבת הקשר לשכבת הרשת וכו.) ולאחר הפירוק הלקוח ידע מי שלח את החבילה ומה יש בתוכה.

נצרך צילום של הפריים הכולל של החבילה (מן הוויר שארק) שעובר בשכבה הפיזית ניתן לראות בצילום מידע רב על החבילה כמו גודלה בביטים.

```
Frame 3: 134 bytes on wire (1072 bits), 134 bytes captured (1072 bits) on interface enp0s3, id 0
```

```
Section number: 1
```

```
Interface id: 0 (enp0s3) <
```

```
Encapsulation type: Ethernet (1)
```

```
Arrival Time: Nov 9, 2022 00:10:57.049356523 Jerusalem Standard Time
```

```
[Time shift for this packet: 0.000000000 seconds]
```

```
Epoch Time: 1667945457.049356523 seconds
```

```
[Time delta from previous captured frame: 331.149919250 seconds]
```

```
[Time delta from previous displayed frame: 331.149919250 seconds]
```

```
[Time since reference or first frame: 456.427095744 seconds]
```

```
Frame Number: 3
```

```
Frame Length: 134 bytes (1072 bits)
```

```
Capture Length: 134 bytes (1072 bits)
```

```
[Frame is marked: False]
```

```
[Frame is ignored: False]
```

```
[Protocols in frame: eth:ethertype:ip:udp:data]
```

```
[Coloring Rule Name: UDP]
```

```
[Coloring Rule String: udp]
```

```
Ethernet II, Src: PcsCompu_16:89:3a (08:00:27:16:89:3a), Dst: PcsCompu_f5:94:fc (08:00:27:f5:94:fc)
```

```
Internet Protocol Version 4, Src: 10.0.2.4, Dst: 10.0.2.5
```

```
User Datagram Protocol, Src Port: 12345, Dst Port: 57025
```

```
Data (92 bytes)
```

0000	08 00 27 f5 94 fc 08 00 27 16 89 3a 08 00 45 00	..'. '...E.
0010	00 78 90 6c 40 00 40 11 92 00 0a 00 02 04 0a 00	.x.l@.@.
0020	02 05 30 39 de c1 00 64 18 7e 4f 66 65 6b 20 63	..09...d ~Ofek c
0030	68 61 6e 67 65 64 20 68 69 73 20 6e 61 6d 65 20	hanged h is name
0040	74 6f 20 4f 66 65 6b 20 42 69 6e 64 65 72 0a 4f	to Ofek Binder.O
0050	66 65 6b 20 42 69 6e 64 65 72 3a 20 68 69 20 69	fek Bind er: hi i
0060	20 67 6f 74 20 61 20 6e 65 77 20 6e 61 6d 65 0a	got a n ew name.
0070	48 65 6d 69 20 68 61 73 20 6c 65 66 74 20 74 68	Hemi has left th
0080	65 20 63 68 61 74	e chat

ניתן לראות שכל שכבה הוסיפה לחבילה מידע שימושי ונוסף על מנת שנוכל להעביר את המידע בבטחה ובדיוק מירבי.