מבוא לרשתות תקשורת

תרגיל 1

עידו ברקאי ויוגב אברבאנל

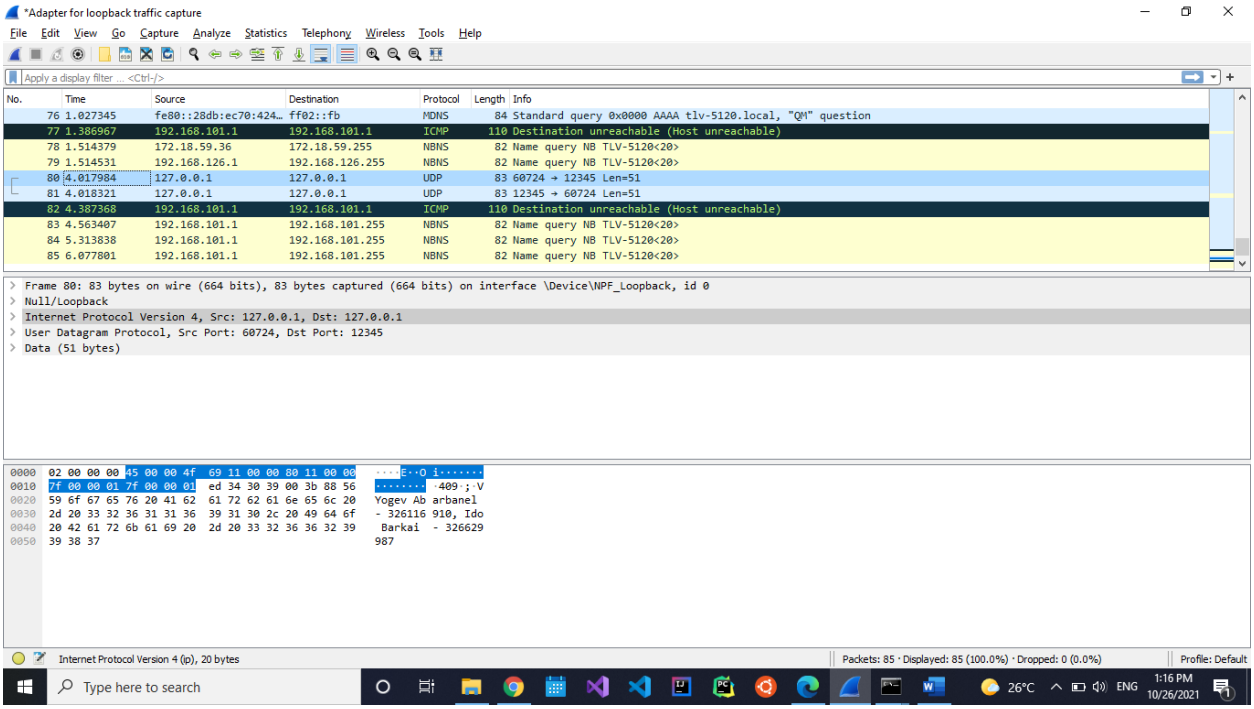
חלק א

1) ראשית, שינינו את הודעת השליחה לשמות והת״ז בקובץ הלקוח:

```
def main():
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

    s.sendto(b'Yogev Abarbanel - 326116910, Ido Barkai - 326629987', ('127.0.0.1', 12345))
    data, addr = s.recvfrom(1024)
    print(str(data), addr)
```

לאחר מכן הרצנו משני טרמינלים את קוד השרת והלקוח והסנפנו את התעבורה על כרטיס ה loopback (זאת מאחר והשרת והלקוח הורצו בשני טרמינלים שונים על גבי אותה המכונה):



2) היה צריך לסנן את החבילות שהן לא של הלקוח והשרת, אז הוספנו את השורה הבאה ב-filter:

udp && udp.port == 12345 && ip.addr == 127.0.0.1

- מסנן חבילות שלא בפרוטוקול תקשורת של השרת והלקוח
- מסנן חבילות מסוג מסויים שגם בפרוטוקול הנ״ל אבל לא של השרת והלקוח

- מסנן לפי כתובת ה-ip של הlopback

3) בשרת יש שימוש במספר פורט כאן:

```
s = socket.socket(socket.AF_INET, sock
s.bind(('', 12345))
```

השימוש הוא בתפיסת הפורט במספר הנ"ל, כך שכאשר מישהו ירצה לשלוח הודעה לשרת, הוא יוכל לשלוח למחשב ולפי מספר הפורט הוא ידע לאיזה אפליקציה לשלוח את ה-pacet. בקוד השרת הוא נמצא בשליחת ההודעה, כדי שהיא תגיע לאפליקציה הנכונה כפי שהסברנו:

```
s.sendto(b'Yogev Abarbanel - 326116910, Ido Barkai - 326629987', ('127.0.0.1', 12345))
data, addr = s.recvfrom(1024)
```

הפורט נמצא בשכבת התעבורה, מספר הפורט הוא חלק מההדר של שכבת התעבורה ונמצא בהודעה:

```
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
User Datagram Protocol, Src Port: 57522, Dst Port: 12345
Data (51 bytes)
```

```
000  02 00 00 00 45 00 00 4f  69 0f 00 00 80 11 00 00   ····E··O i·******
010  7f 00 00 01 7f 00 00 01  e0 b2 30 39 00 3b 94 d8   ******** ··09·;··
020  59 6f 67 65 76 20 41 62  61 72 62 61 6e 65 6c 20   Yogev Ab arbanel
030  2d 20 33 32 36 31 31 36  39 31 30 2c 20 49 64 6f   - 326116 910, Ido
040  20 42 61 72 6b 61 69 20  2d 20 33 32 36 36 32 39    Barkai  - 326629
050  39 38 37                                           987
```

הפורטים מסומנים בכחול, זאת הודעה שנשלחה מהלקוח לשרת, כיוון שבלקוח אין bind לפורט מסוים, מערכת ההפעלה הקצתה לו אחד שהיה פנוי (src port: 57522), לעומת זאת כפי שעשינו בbind הפורט של השרת הוא האחד שהקצאנו לו (dst port: 12345).

4) כפי שניתן לראות בכחול :

```
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> User Datagram Protocol, Src Port: 57522, Dst Port: 12345
v Data (51 bytes)
    Data: 596f676576204162617262616e656c202d20333236313136393130 2c2049646f20426
    [Length: 51]

0000  02 00 00 00 45 00 00 4f  69 0f 00 00 80 11 00 00   ····E··O i······ 
0010  7f 00 00 01 7f 00 00 01  e0 b2 30 39 00 3b 94 d8   ····· ··09·;·· 
0020  59 6f 67 65 76 20 41 62  61 72 62 61 6e 65 6c 20   Yogev Ab arbanel 
0030  2d 20 33 32 36 31 31 36  39 31 30 2c 20 49 64 6f   - 326116 910, Ido 
0040  20 42 61 72 6b 61 69 20  2d 20 33 32 36 36 32 39    Barkai  - 326629 
0050  39 38 37                                           987
```

החבילה הזאת הגיעה מה-ip 127.0.0.1 ונשלחה ל-ip 127.0.0.1. (בדומה החבילה השנייה)

```
yogev_a@MININT-GLCA489:~$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.27.88.109  netmask 255.255.240.0  broadcast 172.27.95.255
        inet6 fe80::215:5dff:fe84:1ad2  prefixlen 64  scopeid 0x20<link>
        ether 00:15:5d:84:1a:d2  txqueuelen 1000  (Ethernet)
        RX packets 660  bytes 252980 (252.9 KB)
        RX errors 0  dropped 3  overruns 0  frame 0
        TX packets 195  bytes 13320 (13.3 KB)
        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 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

yogev_a@MININT-GLCA489:~$
```

כפי שניתן לראות בכרטיסי הרשת, זהו הכרטיס של lopback, כלומר ההודעה נשלחה מהמחשב הזה 127.0.0.1 מציין זאת, ונשלחה אל עצמו 127.0.0.1.

פרוטוקול שכבת האפליקציה

כדי להתמודד עם זריקת החבילות והדילי, הפרוטוקול שלנו הוסיף מספר סידורי לחבילה וטיימאאוט בלקוח.

בשרת, אנו מתחזקים מונה, בהגעת חבילה:
1.בודק האם המספר הסידורי של החבילה תואם למונה:
א. אם כן,
-מדפיס אותה
-מעלה את המונה ב1
-שולח ack (החבילה עצמה)
ב. אם לא, שולח ack (החבילה עצמה)

לקוח:
1. שולח חבילה ומפעיל טיימאאוט
2. מחכה לack:
א. אם לא קיבל והטיימאאוט נגמר אז הוא שולח אותה שוב(1)
ב. אם קיבל, בודק את מספרה הסידורי:
-אם שונה מהמונה, זורק אותה
-אחרת:
*מעלה את המונה
*שולח את החבילה הבאה (1 שוב)

נכונות הפרוטוקול:
1. במקרה של חבילות עד גודל 100 טפלנו בעת יצירת ההודעה.
2. עם זריקת פקטות טפלנו על ידי הטיימאאוט, במקרה שהודעה נזרקה הטיימאאוט יגיע לסיומו והיא תשלח שוב.
כיוון שיתכן ממקרה שבו, הטיימאאוט יסתיים בגלל שה ack עדיין לא הגיע (מקצב השליחה ברשת או זריקת ה-ack על ידי foo), ואז תקרה שליחה חוזרת של הודעה אשר לא נזרקה,
הוספנו את המספר הסידורי כדי שהשרת יוכל למנוע שליחת ack כפול על חבילה יחידה.
3. במקרים של עיכוב, טפלנו בדיעבד על ידי טיפול ב-2, אם יש עיכוב של חבילה, כלומר הסדר לא נכון, אנו נתעלם בשרת מכל הפקטות עם מספר סידורי לא תואם ורק נחזיר עליהן ack (למקרה שה-ack הקודם נזרק ולא הגיע ללקוח), וגם בקוד לקוח אם ה-ack לא תואם למונה נתעלם.
כיוון שאנו עובדים בשיטת stop & wait, יכולות להגיע פקטות רק מהמספר הסידורי הדרוש ומטה, לכן בהחזרת ack, אם הוא קטן יותר זה לא ישפיע כי הלקוח יתעלם אבל אם זאת החבילה הנוכחית הלקוח יתיחס ויקדם את המונה.

חלק ב

נפרט על פעולתו של הקוד בכל אחד מן השלבים[1]:

לאורך כל ההסבר שלושת התוכנות רצו על טרמינלים שונים כמתואר בטבלה הבאה:

	server	foo	client
port	23456	12345	רנדומלי, מוקצה ע"י מערכת ההפעלה
ip	127.0.0.1	127.0.0.1	127.0.0.1

שלב 1

בשלב זה foo "מתנהג יפה" ומעביר את החבילות כמו שצריך בתנאי שאינן מכילות יותר מ-100 בתים.

נצרף דוגמת הרצאה של התכנית:

server

```
>> python3 server.py 23456
```

foo

```
>> python3 foo.py 12345 127.0.0.1 23456 1
```

client

```
>> python3 client.py 127.0.0.1 12345 input.txt
```

נסניף את התעבורה המתאימה ב wireshark המגיעה ויוצאת מן foo[2]:

No.	Time	Source	Destination	Protocol	Length	Info
7	1.791669	127.0.0.1	127.0.0.1	UDP	132	58498 → 12345 Len=100
8	1.793332	127.0.0.1	127.0.0.1	UDP	132	12345 → 23456 Len=100
9	1.794421	127.0.0.1	127.0.0.1	UDP	132	23456 → 12345 Len=100
10	1.794856	127.0.0.1	127.0.0.1	UDP	132	12345 → 58498 Len=100
11	1.795070	127.0.0.1	127.0.0.1	UDP	132	58498 → 12345 Len=100
12	1.795928	127.0.0.1	127.0.0.1	UDP	132	12345 → 23456 Len=100
13	1.797126	127.0.0.1	127.0.0.1	UDP	132	23456 → 12345 Len=100
14	1.797462	127.0.0.1	127.0.0.1	UDP	132	12345 → 58498 Len=100
15	1.797538	127.0.0.1	127.0.0.1	UDP	132	58498 → 12345 Len=100
16	1.798020	127.0.0.1	127.0.0.1	UDP	132	12345 → 23456 Len=100
17	1.798627	127.0.0.1	127.0.0.1	UDP	132	23456 → 12345 Len=100
18	1.798949	127.0.0.1	127.0.0.1	UDP	132	12345 → 58498 Len=100
19	1.799023	127.0.0.1	127.0.0.1	UDP	132	58498 → 12345 Len=100
20	1.800151	127.0.0.1	127.0.0.1	UDP	132	12345 → 23456 Len=100
21	1.800680	127.0.0.1	127.0.0.1	UDP	132	23456 → 12345 Len=100

---

[1] קובץ הטקסט מכיל חלק מן הסיפור "אליסה בארץ הפלאות"

[2] הסינון בוצע ע"י הפילטר udp&&udp.port==12345

ונתסכל על השכבות השונות בפקטה:

<u>שכבת התעבורה</u> – בשכבת התעבורה עבדנו עם פרוטוקול UDP, לכן ניתן לראות בשכבה זו פורט מקור ופורט יעד:

```
User Datagram Protocol, Src Port: 57738, Dst Port: 12345
    Source Port: 57738
    Destination Port: 12345
```

תמונה 1.1: פקטה מ client אל foo

```
User Datagram Protocol, Src Port: 12345, Dst Port: 23456
    Source Port: 12345
    Destination Port: 23456
```

תמונה 1.2: הפקטה מועברת ל server

```
User Datagram Protocol, Src Port: 23456, Dst Port: 12345
    Source Port: 23456
    Destination Port: 12345
```

תמונה 1.3: פקטת תגובה חוזרת מ server ל foo

```
User Datagram Protocol, Src Port: 12345, Dst Port: 57738
    Source Port: 12345
    Destination Port: 57738
```

תמונה 1.4: הפקטה חוזרת ל client

כמו שניתן לראות, פורט המקור ופורט היעד משתנים בהתאם למסלול הפקטה

<u>שכבת הרשת</u> – בשכבת הרשת נראה את כתובות ה-ip, מאחר והרצנו את התוכנות בטרמינלים שונים באותה המכונה נראה כי כולם יהיו 127.0.0.1

```
Source Address: 127.0.0.1
Destination Address: 127.0.0.1
```

תמונה 1.5: כתובת המקור והיעד בפקטות

<u>שכבת אפליקציה</u>

בשכבת האפליקציה ניתן לראות את המידע שמועבר בפקטות, הכתוב בפרוטוקול שהגדרנו לעיל[3]. לדוגמה:

```
Data (100 bytes)
    Data: 416c69636527732041647665 6e747572657320696e20576f6e6465726c616e640a0a2020…
    Text: Alice's Adventures in Wonderland\n\n                ALICE'S ADVENTURES IN WONDERLAND\n\n
    [Length: 100]
```

תמונה 1.6: הפקטה הראשונה שנשלחה מ client ל foo

[3] בשלב זה עוד לא היה צריך למספר את הפקטות

שלב 2

בשלב זה foo בנוסף לתנאי של גודל החבילות זורק באקראי אחוז מסויים של החבילות בשני כיוונים. על כן נראה תעבורה יותר מסיבית – הרבה יותר פקטות יישלחו.

נצרף דוגמת הרצאה של התכנית:

server

>> python3 server.py 23456

foo

>> python3 foo.py 12345 127.0.0.1 23456 2

client

>> python3 client.py 127.0.0.1 12345 input.txt

נסניף את התעבורה המתאימה ב wireshark המגיעה ויוצאת מן foo:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	UDP	132	57690 → 12345 Len=100
2	2.000459	127.0.0.1	127.0.0.1	UDP	132	57690 → 12345 Len=100
3	2.001664	127.0.0.1	127.0.0.1	UDP	132	12345 → 23456 Len=100
4	2.002742	127.0.0.1	127.0.0.1	UDP	132	23456 → 12345 Len=100
5	4.000964	127.0.0.1	127.0.0.1	UDP	132	57690 → 12345 Len=100
6	4.002132	127.0.0.1	127.0.0.1	UDP	132	12345 → 23456 Len=100
7	4.002358	127.0.0.1	127.0.0.1	UDP	132	23456 → 12345 Len=100
8	4.002933	127.0.0.1	127.0.0.1	UDP	132	12345 → 57690 Len=100
9	4.003116	127.0.0.1	127.0.0.1	UDP	132	57690 → 12345 Len=100
10	4.003498	127.0.0.1	127.0.0.1	UDP	132	12345 → 23456 Len=100
11	4.004703	127.0.0.1	127.0.0.1	UDP	132	23456 → 12345 Len=100
12	4.005654	127.0.0.1	127.0.0.1	UDP	132	12345 → 57690 Len=100
13	4.005846	127.0.0.1	127.0.0.1	UDP	132	57690 → 12345 Len=100
14	4.006645	127.0.0.1	127.0.0.1	UDP	132	12345 → 23456 Len=100
15	4.007236	127.0.0.1	127.0.0.1	UDP	132	23456 → 12345 Len=100
49	6.006211	127.0.0.1	127.0.0.1	UDP	132	57690 → 12345 Len=100
71	8.006750	127.0.0.1	127.0.0.1	UDP	132	57690 → 12345 Len=100
72	8.007998	127.0.0.1	127.0.0.1	UDP	132	12345 → 23456 Len=100
73	8.008245	127.0.0.1	127.0.0.1	UDP	132	23456 → 12345 Len=100
74	8.009813	127.0.0.1	127.0.0.1	UDP	132	12345 → 57690 Len=100

התעבורה אכן יותר גדולה – 114 פקטות לעומת 68 בשלב 1

נתסכל על השכבות השונות בפקטה:

<u>שכבת התעבורה</u> – בשכבת התעבורה לא השתנה הפרוטוקול. עבדנו עם פרוטוקול UDP, לכן ניתן לראות בשכבה זו פורט מקור ופורט יעד:

```
User Datagram Protocol, Src Port: 57738, Dst Port: 12345
    Source Port: 57738
    Destination Port: 12345
```

תמונה 1.1: פקטה מ client אל foo

```
User Datagram Protocol, Src Port: 12345, Dst Port: 23456
    Source Port: 12345
    Destination Port: 23456
```

תמונה 1.2 : הפקטה מועברת ל server

```
User Datagram Protocol, Src Port: 23456, Dst Port: 12345
    Source Port: 23456
    Destination Port: 12345
```

תמונה 1.3 : פקטת תגובה חוזרת מ server ל foo

```
User Datagram Protocol, Src Port: 12345, Dst Port: 57738
    Source Port: 12345
    Destination Port: 57738
```

תמונה 1.4 : הפקטה חוזרת ל client

כמו שניתן לראות, פורט המקור ופורט היעד משתנים בהתאם למסלול הפקטה

<u>שכבת הרשת</u> – גם בשכבת הרשת לא השתנה הפרוטוקול. נראה את כתובות ה-ip, מאחר והרצנו את התוכנות בטרמינלים שונים באותה המכונה נראה כי כולם יהיו 127.0.0.1

```
Source Address: 127.0.0.1
Destination Address: 127.0.0.1
```

תמונה 1.5 : כתובת המקור והיעד בפקטות

<u>שכבת אפליקציה</u>

בשכבת האפליקציה ניתן לראות את המידע שמועבר בפקטות, הכתוב בפרוטוקול שהגדרנו לעיל – בשלב זה כבר מספרנו כל פקטה, נוכל לראות זאת בתחילת חלק המידע. לדוגמה :

```
Data (100 bytes)
    Data: 310a416c69636527732041647665 6e747572657320696e20576f6e6465726c616e640a0a…
    Text: 1\nAlice's Adventures in Wonderland\n\n                ALICE'S ADVENTURES IN WONDERLAND\n\n
    [Length: 100]
```

תמונה 1.6 : הפקטה הראשונה שנשלחה מ client ל foo

שלב 3+4

שלבים אלו עובדים באותו פרוטוקול של שלב 2 בשכבת האפליקציה (כמפורט לעיל), הרשת והתעבורה ולכן נראה מידע דומה בפקטות.

אכן נשים לב שהתעבורה תהיה גדולה ככל שפרמטרי העיכוב וזריקה יגדלו.