

## מקרה מס' 2 – אילאי בן יהושע ונועם ליבוביץ

ראשית נתאים ונשנה את הקוד כך שישלוח את שמותינו לשרת, באופן הבא –

```

tcp_server.py > ...
1 import socket
2 server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
3 server.bind(('', 12345))
4 server.listen(5)
5
6 while True:
7     client_socket, client_address = server.accept()
8     print('Connection from: ', client_address)
9     data = client_socket.recv(100)
10    print('Received: ', data)
11    client_socket.send(data.upper())
12    client_socket.close()
13    print('Client disconnected')

```

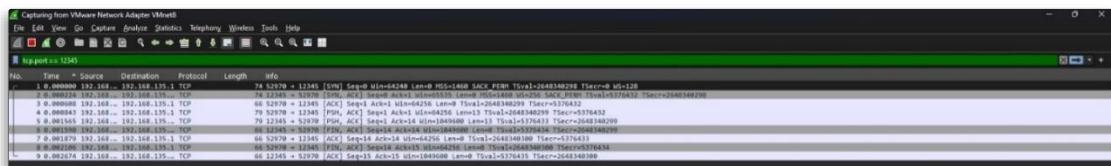
```

tcp_client.py > ...
1 import socket
2 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
3 s.connect(('192.168.135.1', 12345))
4 s.send(b'Noam and Elay')
5 data = s.recv(100)
6 print("Server sent: ", data)
7 s.close()

```

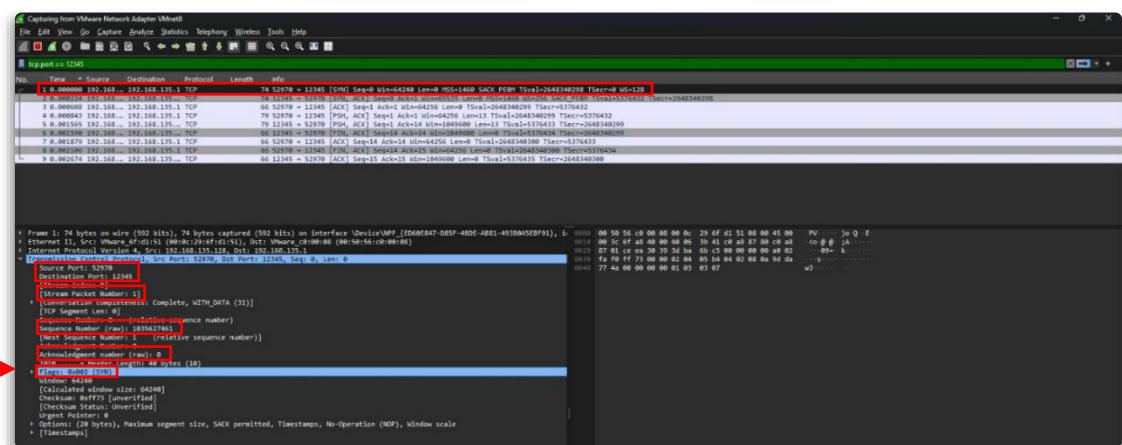
כעת נריץ את הקודים המעודכנים הנ"ל, נסניף את המידע באמצעות "כרייש-הכבל" (בליעז – wireshark, מעתה והילך כאשר נכתב 'כרייש' בקיצור נתייחס למינוח זהה), וננתח את מה שקרה פה בהתאם לעקרונות TCP שראינו בהרצאה ובתרגול.

זה מה שמוצג לנו בכרייש לאחר הריצת הקודים -



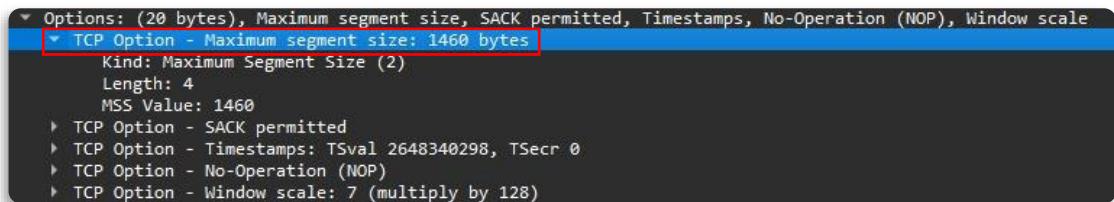
מעבר על כל חבילה וחבילה על מנת לתאר במדויק כל שלב בתקשורת.

## חביבה מס' 1 –



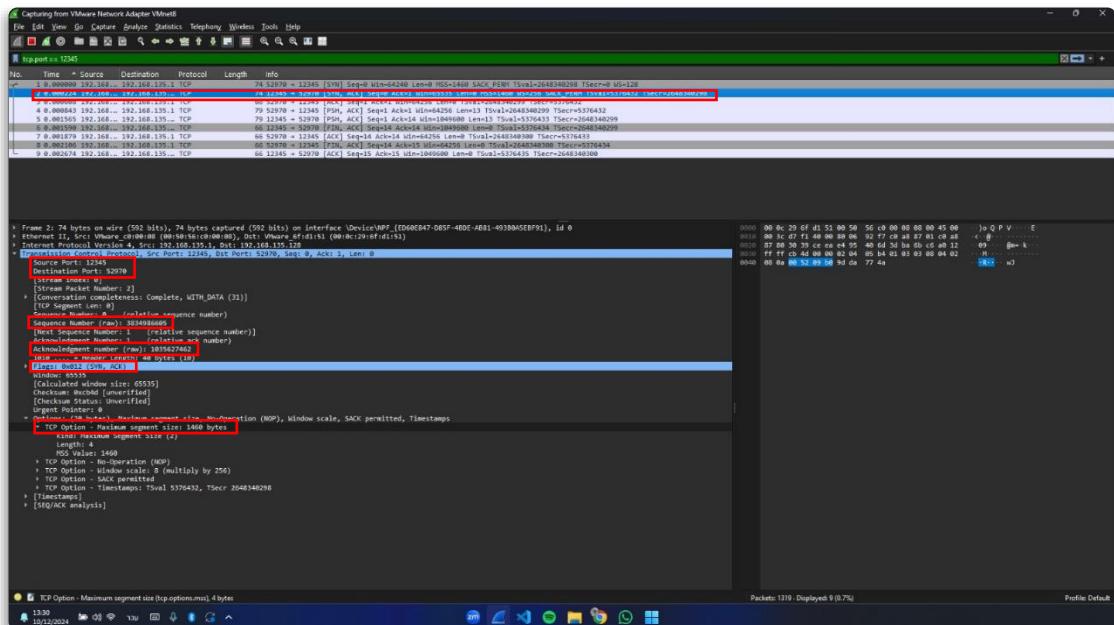
אנו רואים פניה שמתבצעת לPORT 12345 – (אשר אותו נתנו ידנית לשרת), אשר מגיעה מפורט מקור 52979 אשר ניתן דינאמית ללקוח ע"י מ"ה (כיון שלא צמדנו אותו לפורט ספציפי). הפנייה היא מסוג chs (ניתן לראות את הדגל הדלוק), ככלומר הלקוח מבקש להסתנן עט השרת וזו הפקידה של החביבה הנ"ל עם דגל החsus, אשר מהוות את השלב הראשון בטקס 'לחיצת הידיים המשולשת'. בנוסף לכך אנו רואים את ערך האופטט הגולמי (= raw) של seqnum (raw) של הלקוח, כאשר ניתן לראות שהוא החוליט שתחלת התקשרות תתחילה מואופטט – שבחור הלקוח, כאשר לנו שערך ack שלו כרגע יהיה '0', משום שהseqnum שלו, זה seqnum של השרת (אשר טרם נבחר.). כמובן שהseqnum הוא 0 שהרי אין DATA בהודעת chs.

דבר מעניין נוסף ניתן לראות כאשר נפתח את הoptions –



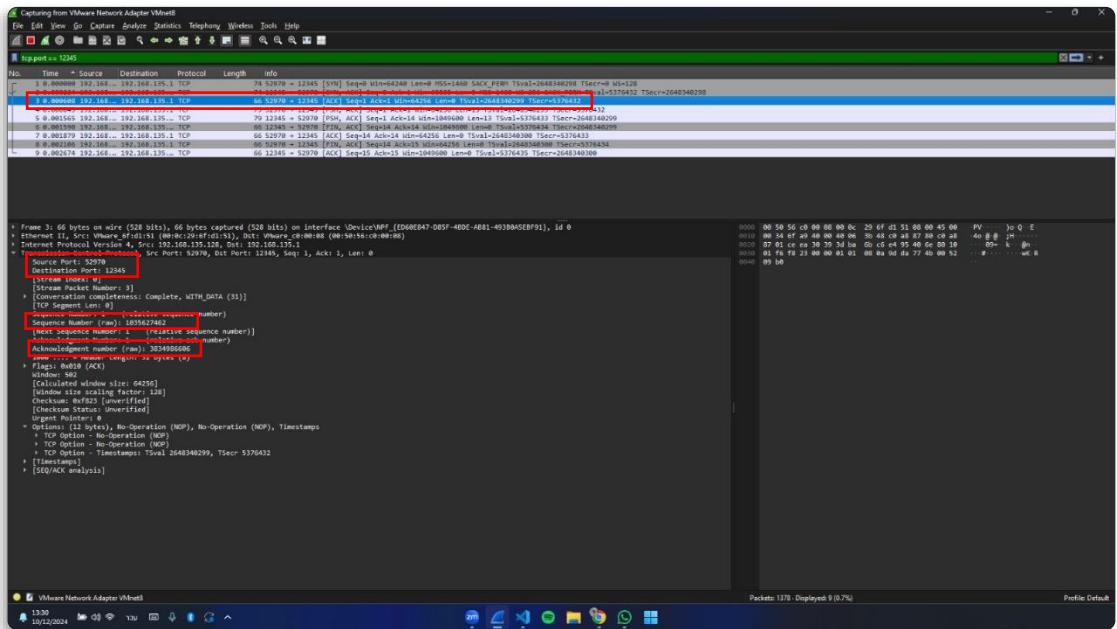
ובו, הלקוח מציין בפני השרת – "רק שטdue, התסס שלו הוא 1460b".

## חvíלה מס' 2 –



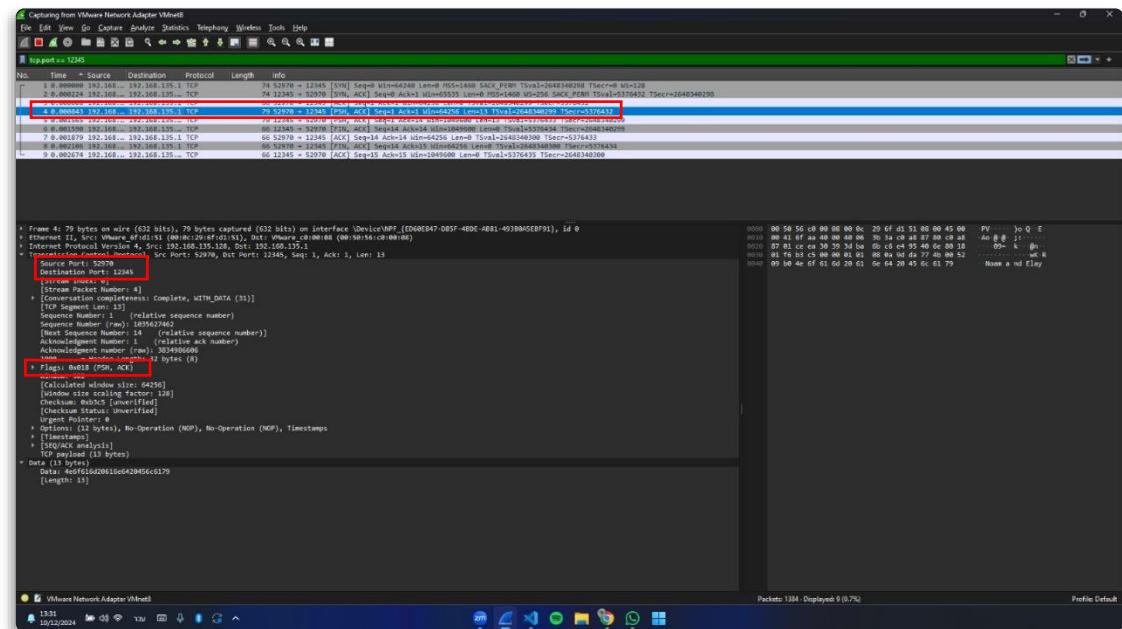
אנו רואים פניה שמתבצעת הפעם בכיוון הפהוף מ-PORT 12345 (השרת), אשר מיודעת לפורט 52970 (הלקוח). הפניה (או יותר נכון 'tagובה') מצד השירות הינה מסוג syn, ack (גם פה ניתן לראות את שני הדגלים דלוקים), כלומר **השרת ראשית מכיר בסנסרנו של הלקוח איתנו**, אך בנוסך הוא מבקש להסתنصرנו גם הוא עם הלקוק. זהה למשזה תפקידה של החvíלה הנ"ל עם דגלי ack, syn, אשר מהוות את השלב השני בטקס 'לחיצת הידיים המשולשת'. ראיו לציין כי בשלב הנ"ל עשוי להתבצע לעיתים 2 חבילות שונות (1 ל-ack ו-1 ל-syn) אך כמו שאנו רואים כאן, אלו יכולות להישלח גם כן ייחודי בחvíלה אחת. בנוסף לכך אנו רואים את ערך האופסוט הגולמי (= raw) של seq (raw) של 3834986605 – '3834986605', ואילו נשים לב שערך ack שלו יהיה התקשורת תחילית מאופסוט – '1305627462', שזה 1 יותר מאשר הסהיה seq של הלקוק בגל ה-bit-bit-phantom. גם כאן הchen הוא 0 שחיי אנחנו עדין לא בשלב של העברת DATA, אך מאוד מתקרבים לשם. אם נפתח את ה-opts option נשים לב שגם השירות מציין בפני הלקוק מהו התסס שלו – 1460b, כדי שזה דעת את הנתון אשר הוא שולח אליו הודעות.

## חvíלה מס' 3 –



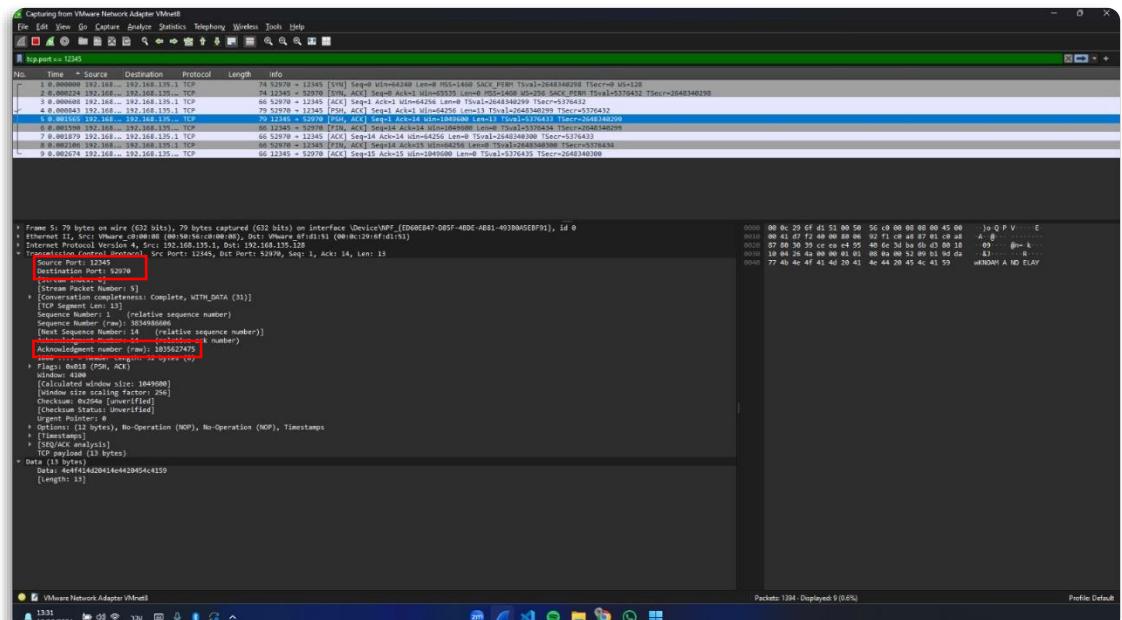
כעת ניתן לראות פניה נוספת מהלך 52970 (הלקוח), אל השרת. הפניה מצד הרשת הינה מסוג ack (שוב הדגל דлок ולמענה מעתה והילך הוא יהיה דлок עד סוף התקשרות, אז נפסיק לציין זאת...), דהיינו הלקוח גם הוא מכיר בהכרת הרשת בסנכרון שלו, ולמענה תפקידה של החvíלה הזאת את השלב השלישי **והאחרון** בטקס 'לחיצת הידיים המשולשת', שכן שני הצדדים הסתנכרנו אחד עם השני, וכל אחד מהם הכיר בסנכרון המשותף, ומכאן והילך הקשר שරיר ונitin להעביר דרכו הودעות. שוב אנו רואים את ערך האופטט הגלומי (= raw) של ack שהשליח הלקוח, והוא – '06'3834986606', באופן לא מפתיע זה 1 יותר מהפזז שבחר הרשת (שוב אנחנו מגדילים בגל הפאנטום ביט של ack). הseq לעומת זאת לא השתנה שכן לא נשלח עוד נתונים, וכתוואה מכך שוב החלו הוא 0 (אך לא עוד הרבה זמן). אם נפתח את הoptions נשים לב שההסז שלו כבר לא נשלח, שכן שני הצדדים כבר יודעים ומודעים למוגבלות כל אחד של השני.

## חvíלה מס' 4 –



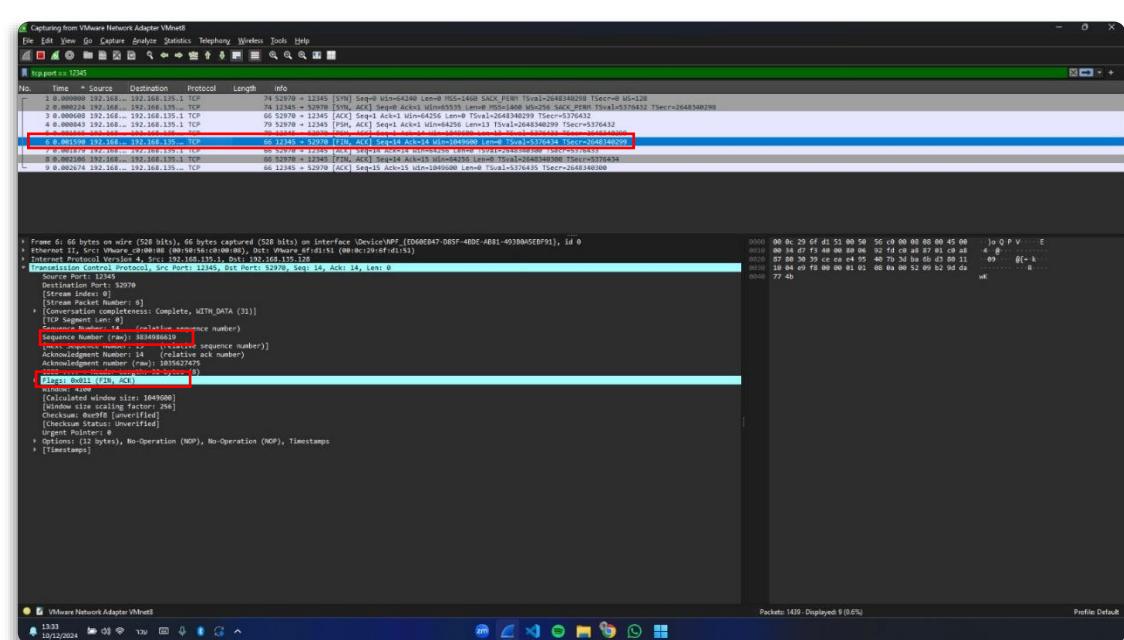
סוף כל סוף הגיענו לחלק המעניין ('פsegת הקורס' לפי חלק מהשיעור), אנו רואים הودעה מהלוקו (לפי הפורט 52970 שראינו קודם שהוא הפורט של הלוקו), השולח 13 בתים (לפי *len*), שנחננו יודעים שהם הבטים של המחרוזת 'Noam and Elay' (כנדרש במתלה), ובנוסף לדגל *ack* (שאמרנו שמעתה והילך תמיד היה דילוק), גם כן דילוק דגל ההשיק שכאילו אומר להעביר בדחיפות לאפליקציה את המידע שהגיע. נשים לב שעכשיו יש לנו גם את שכבת *data* בהודעה (הבטים שלחנו מואפן מוקודד).

## – 5 'חבילה מס'



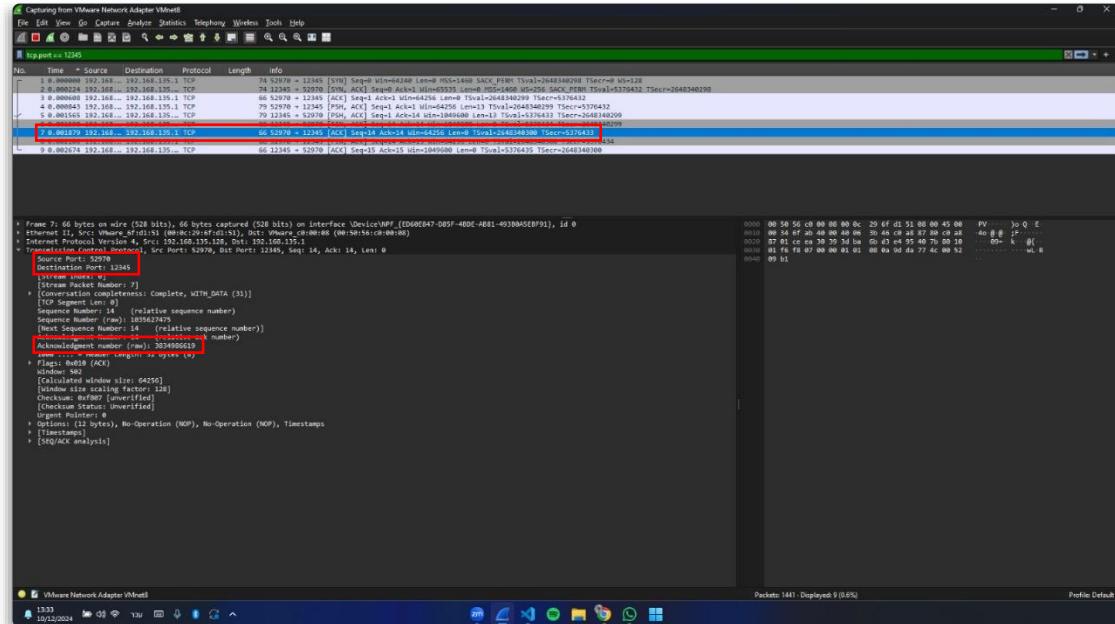
כעת נראה את תגובת השירות להודעה שקיבל מהלוקו בחבילה הקודמת שסקרנו. הרשות הראשית **מעדכן** את המספר בשדה *ack* להיות 13 יותר מאשר – '1035627475' שכן עכשו הוא אומר ללוקו – "שמעו?! קיבלתיכי כבר עד בית מס' 1035627475 תן לי ממוני והלאה.". אמןם *seq* לא משתנה כיון שהוא טרם שלח נתונים עצמו ללוקו, אך כעת הוא מכרף להודעת *ack* את התוכן שקיבל בזאת (ושוב לנו לא מופתעים שההט *len* הוא 13).

## – 6 'חבילה מס'



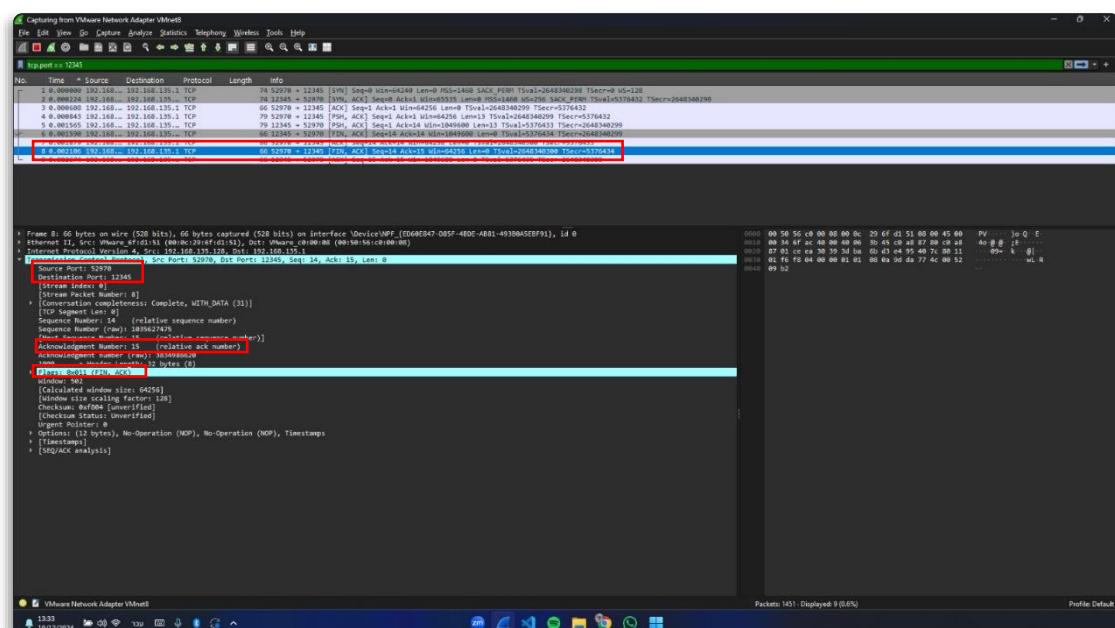
מיד לאחר שהשרת שולח את הודהה **ack** יחד עם התוכן בזקוקס, הוא שולח הודעה **fin** (ניתן לראות דגל דולק), שכן הוא עשה את שלו, ותפקיד ההודהה זה להודיע על כך שהוא מעוניין לסיים את מערכת היחסים עם הלקוח. נשים לב שעכשו הseq של השרת גדל ב-13 גם הוא ערך – '3834986619'.

## – 7' חבילה מס'



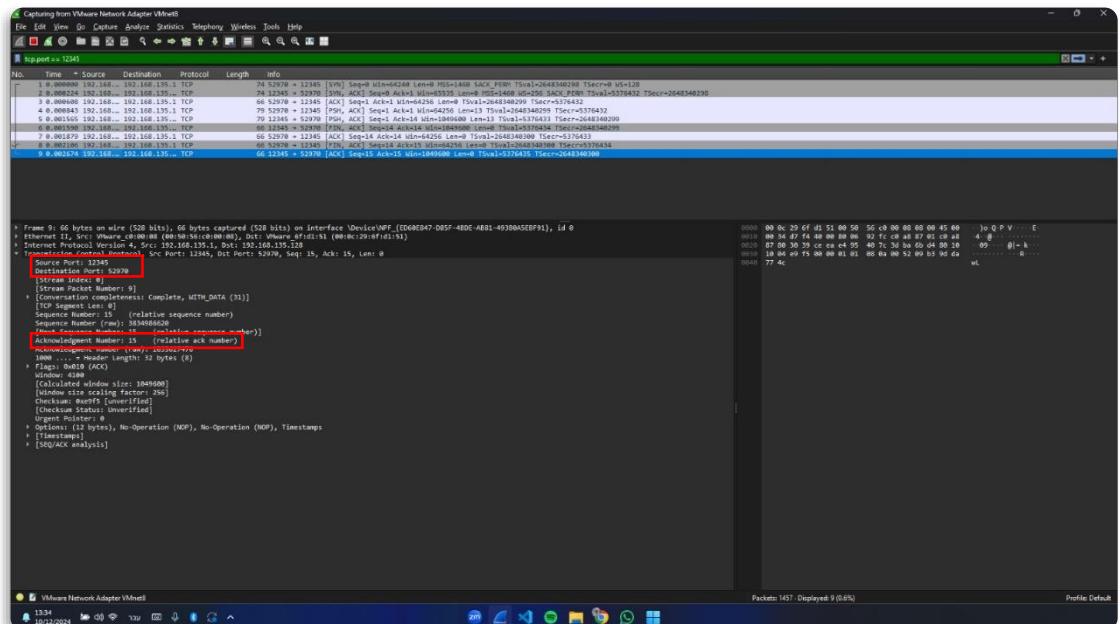
לקוח מקבל את הדטא חזרה מהשרת בזקוקס, בהתאם הוא מגדיל את ערך **acked** ב-13 בתים נוספים לערך – '3834986619'. כמו כן נשים לב שגם הseq המזמין בהודהה גדול ב-13 מהפעם האחרון בה ראיינו חבילה נשלחת מהלקוח, וcutut הוא עומד על – '1035627475' (שוקינג זה הseq של השרת..).

## – 8' חבילה מס'



הלקוח כעת שלוח הודעת ack לשרת, ולמענה מעדכן שהוא קיבל את ההודעה שלו ברגע לסיום ההתקשרות עימיו. כיוון שהוא הבין שנגמר הקשר, הוא מסיים אותו גם מבחינתו ומצרף זאת להודעה (דגל fin דלוק בה). דהיינו **תפקיד ההודעה לומר לשרת שהסיום הינו הדדי**. מעבר לזה אין מידע מעניין למעט העובדה שערך ack של הלוקוח גדול ב-1 כתוצאה מהפאנטום ביט של הודעה החfin שמכורמת רק עכשו.

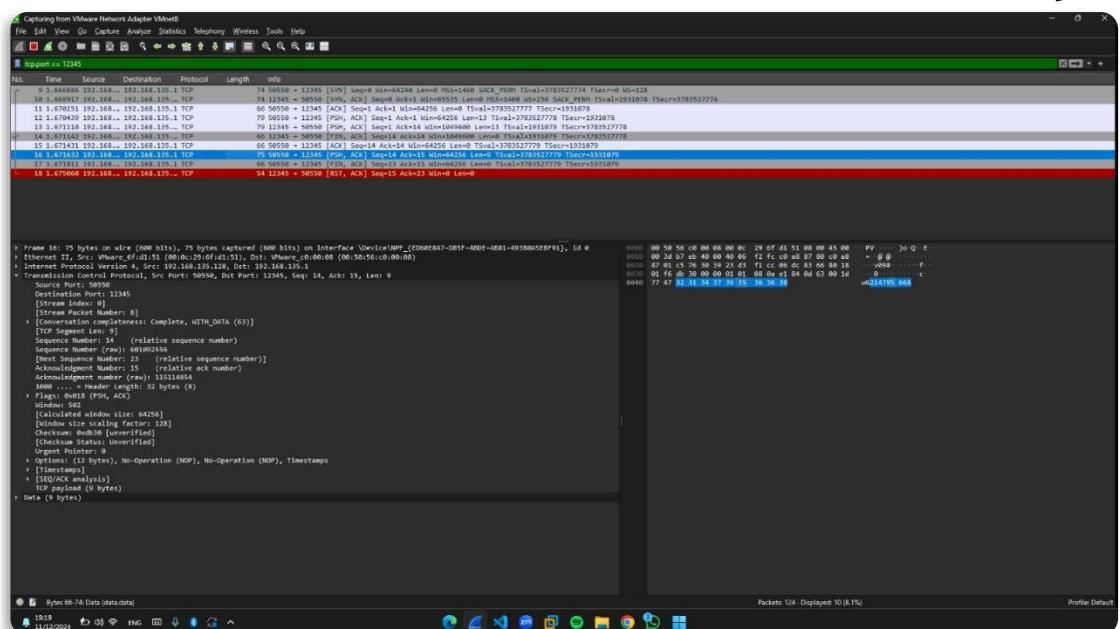
## ח毕לה מס' 9 –



בח毕לה الأخيرة (😢) שלנו אנו רואים את השרת מכיר בסיום ההתקשרות מצד הלוקוח גם הוא בהודעת ack, ובאופן פרקטן **תפקיד ההודעה הינו לסייע לחולוטן בהחלטת ערך התקשרות שהיא בינהם**. נשים לב שרגע לפני שהוא אומר "ב'י'", הוא מגיד גם הוא את ערך ack כתוצאה מהפאנטום ביט של החfin שהגיע מהлокוח, סה"כ ערך ack של השרת בסיום התקשרות עומד על – '1035627476'.

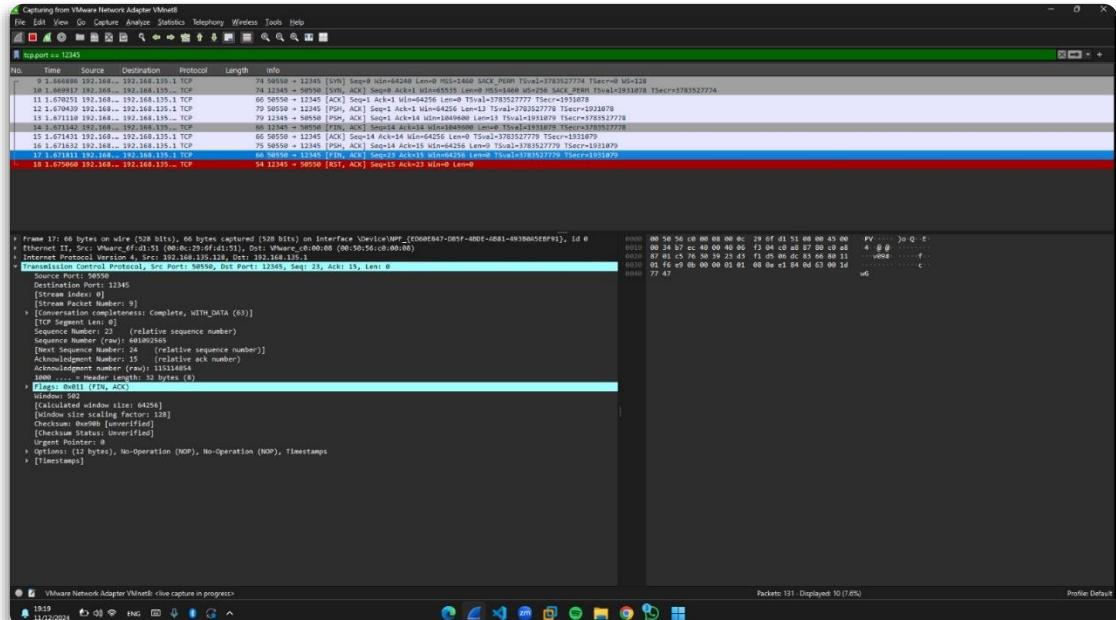
סימנו לנתח את התקשרות בחלק הראשון. נסיף כעת את שליחת ה-tz. של אילאי לאחר שהשרת משיב את תשובתו ונראה את השינויים.

כל ההודעות בהתחלה תתנהגנה באופן זהה (לכן נחשוך ונתחיל רק מהמקום בו השינוי קורה), אבל עכשו נשים לב –

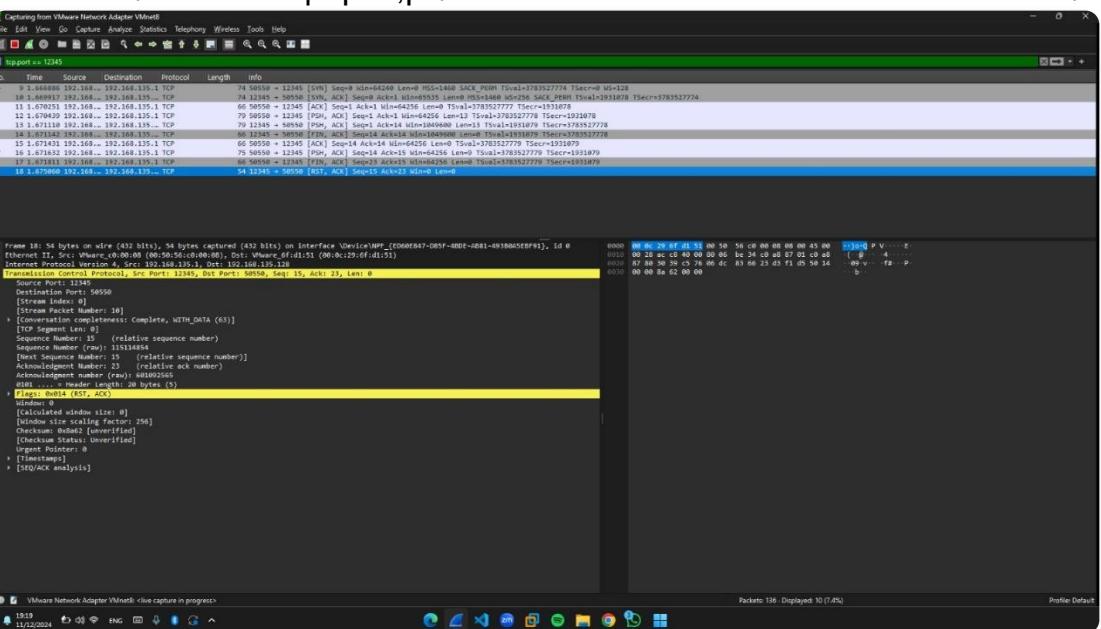


לאחר שהלkipן החזיר ack על החזיר fin של השרת (פאנטום בית מגדל ב-1 את ה-edged), הוא מנסה לשלוח הודעה נוספת נוספת ולבסוף לה במקומ לשלוח גם הוא fin (כמו שקרה קודם), והוא מנסה לשוחה הודעה נוספת ולבסוף לה skp לאפליקציה (הלווא היא הת.ז.). ורק לאחר שההודעה הזו נשלחת, הוא שולח גם fin

מצידן –



הseq שלו גדול ב-9, והוא מסכים עכשו לסיים את התקשרות. אלא שבגלל שהשרת לא ידע – rst – נקלט ממנו הודעת –



כאיו רוצה לומר – "חדש אח שלי", חדש. מה שהיא אני כבר לא שם, ואין מי שישמע ויקבל את מה שיש לך לשולח. אם אתה רוצה בוא נתחיל מההתחלת, ומשם נשתמע הלאה...".

ובכן סימנו לנתח גם את המקירה בו הודעה נשלחת לאחר שהשרת עושה קלוז קודם לרגע בו היא מגיעה.

## סעיף 2:

כעת, ננתח בקצירה את הקודים של הגרסאות השונות ואת התעבורה בהן:

### גרסה 1:

The figure shows two terminal windows and a Wireshark capture of network traffic.

**server.py** (Terminal 1):

```
versions > versions 2 > v1 > server.py > ...
1 import socket,sys
2
3 TCP_IP = '0.0.0.0'
4 TCP_PORT = int(sys.argv[1])
5 BUFFER_SIZE = 1024
6
7 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
8 s.bind((TCP_IP, TCP_PORT))
9 s.listen(0)
10
11 while True:
12     conn, addr = s.accept()
13     print('New connection from:', addr)
14
15     conn1, addr1 = s.accept()
16     print('New connection from:', addr1)
17
18     data = conn1.recv(BUFFER_SIZE)
19     print("received:", data)
20     conn.send(data[0:7])
21
22     data = conn.recv(BUFFER_SIZE)
23     print("received:", data)
24     conn1.send(data[0:5])
25
26     conn.close()
27     conn.close()
```

**client.py** (Terminal 2):

```
versions > versions 2 > v1 > client.py > ...
1 import socket,sys
2 from time import sleep
3
4 TCP_IP = sys.argv[1]
5 TCP_PORT = int(sys.argv[2])
6 BUFFER_SIZE = 1024
7 MESSAGE = b'Foo, World!'
8 MESSAGE1 = b'bar, Hello'
9
10 s1 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
11 s1.connect((TCP_IP, TCP_PORT))
12
13 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
14 s.connect((TCP_IP, TCP_PORT))
15 s.send(MESSAGE)
16
17 sleep(5)
18
19 s1.send(MESSAGE1)
20 data = s1.recv(BUFFER_SIZE)
21 s1.close()
22
23 data = s.recv(BUFFER_SIZE)
24 s.close()
```

**Wireshark Capture:**

This Wireshark capture shows the TCP handshake and data exchange between the client and server. The client sends "Foo, World!" and receives "bar, Hello". The server sends "bar, Hello" and receives "Foo, World!". The traffic includes SYN, ACK, and FIN packets along with data payloads.

הלקוח מקבל את ה-IP והפורט מהארוגמנטים לתוכנית, לאחר מכן הוא מתחבר בשני חיבורים בפרדימט בعزيزת שני סוקטיפ נפרדים לשרת (שורות 4 עד 9). הלקוח תחילת שולח הודעה דריך החיבור שני, ממחכה 5 שניות ושלח הודעה אחרית דריך החיבור הראשון. לבסוף, הלקוח מקבל את שתי ההודעות מהסתוקט בסדר הפוך לסדר בו התבצעה הש寥חה, ולאחר מכן סגור את החיבורים. צד שני, הרשת מזין לכל החיבורים כיוון שהוא בחר בכתובת 0.0.0.0 ומתקבל פורט קלט. הוא מרים את השרת לאוואר כאשר הוא מוכן שייהי ברגע נתון לכל היותר חיבור 1 בהמתנה (לייטן שווה 0). הוא עושה accept לשני חיבורים (מדפיס מי המ), מקצר את הודעה השנייה ל-7 תווים הראשונים שנקלטו ושלח אותה לחיבור הראשון, ואת הודעה הראשונה ל-5 תווים הראשונים שנקלטו ושלח אותה לחיבור השני. לבסוף, יוזם ניתוק בקעם החיבור הראשון שהוא פתח.

נשים לב שבפועל בכרייש קרה משהו מעניין. לאחר האתחול עד שורה 9, נשלחת ההודעה מהסתוקט שנוצר שני בלקוח, והשרת לפניו שהוא מחזיר עליה ack לאוטו סוקט שני, קודם שלוח את ההודעה המקוצרת (7 בתים), לחיבור הראשוני (שורה 11). יתרה מזאת, הוא מקבל מהחיבור הראשוני ack על ההודעה המקוצרת (שורה 12), ורק אז מחזיר ack על עצם זה שהוא קיבל את ההודעה המקורי מהחיבור השני. לאחר מכן השרת מקבל את ההודעה שהוא קיבל את ההודעה המקורי מהחיבור השני. לאחר מכן השרת משלוח fin מהחיבור השני רק עכשו על הchnfin מצד השרת, ועל הדרך הוא גם נותן לו ack (רק בשלב הזה) על החבילה שנשלחה על ידו, ומיד הודעה נוספת נספתחת של ack על הchnfin עם הפאנטום ביט. החיבור השני מחזיר ack רק עכשו על ההודעה שהוא קיבל (5 הבטים), החיבור הראשוני מחזיר ack על הודעה הchnfin שקיבל מהשרת (שוב פאנטום ביט), ולבסוף השרת מקבל fin מהחיבור השני ומשיב על כך בchnack (הוא לא שלוח לו fin שכן בקוד החון ששולח השרת הינו עבור החיבור הראשוני, שכבר נסגר מיזמתו קודם לכך..).

## גרסה 2:

The screenshot shows two terminal windows and a Wireshark network traffic capture.

**server.py:**

```

server.py > ...
1 import socket,sys
2
3 TCP_IP = '0.0.0.0'
4 TCP_PORT = int(sys.argv[1])
5 BUFFER_SIZE = 5
6
7 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
8 s.bind((TCP_IP, TCP_PORT))
9 s.listen(1)
10
11 while True:
12     conn, addr = s.accept()
13     print('New connection from:', addr)
14     while True:
15         data = conn.recv(BUFFER_SIZE)
16         if not data: break
17         print("received:", data)
18         conn.send(data.upper())
19     conn.close()
20
21

```

**client.py:**

```

client.py > ...
1 import socket,sys
2
3 TCP_IP = sys.argv[1]
4 TCP_PORT = int(sys.argv[2])
5 BUFFER_SIZE = 1024
6 MESSAGE = b'World! Hello, World!'
7
8 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
9 s.connect((TCP_IP, TCP_PORT))
10 s.send(MESSAGE)
11 data = s.recv(BUFFER_SIZE)
12 s.close()
13
14 print("received data:", data)
15
16

```

**Wireshark Capture:**

The Wireshark capture shows the following sequence of frames:

- Frame 4: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 'Device\Device\MPF [ED9E847-0B5F-4B0E-A881-4930865EB91]', id 0
  - Source: 192.168.1.35 (192.168.1.35)
  - Destination: 192.168.1.10 (192.168.1.10)
  - Protocol: TCP (6)
  - Length: 74 Info: Seq=4217713162 Win=64248 Len=48 MSS=1468 SACK\_PERM TS=0.1-1771713162 TSecr=<0 WS=128
- Frame 6: 48 bytes on wire (384 bits), 48 bytes captured (384 bits) on interface 'Device\Device\MPF [ED9E847-0B5F-4B0E-A881-4930865EB91]', id 1
  - Source: 192.168.1.10 (192.168.1.10)
  - Destination: 192.168.1.35 (192.168.1.35)
  - Protocol: TCP (6)
  - Length: 48 Info: Seq=1 Ack=4217713162 Win=64256 Len=0 TSval=1771713162 TSecr=>21771713162
- Frame 8: 48 bytes on wire (384 bits), 48 bytes captured (384 bits) on interface 'Device\Device\MPF [ED9E847-0B5F-4B0E-A881-4930865EB91]', id 2
  - Source: 192.168.1.10 (192.168.1.10)
  - Destination: 192.168.1.35 (192.168.1.35)
  - Protocol: TCP (6)
  - Length: 48 Info: Seq=1 Ack=4217713162 Win=64256 Len=0 TSval=1771713162 TSecr=>21771713162
- Frame 9: 48 bytes on wire (384 bits), 48 bytes captured (384 bits) on interface 'Device\Device\MPF [ED9E847-0B5F-4B0E-A881-4930865EB91]', id 3
  - Source: 192.168.1.10 (192.168.1.10)
  - Destination: 192.168.1.35 (192.168.1.35)
  - Protocol: TCP (6)
  - Length: 48 Info: Seq=21 Ack=4217713162 Win=64256 Len=0 TSval=1771713163 TSecr=>21771713163
- Frame 10: 48 bytes on wire (384 bits), 48 bytes captured (384 bits) on interface 'Device\Device\MPF [ED9E847-0B5F-4B0E-A881-4930865EB91]', id 4
  - Source: 192.168.1.10 (192.168.1.10)
  - Destination: 192.168.1.35 (192.168.1.35)
  - Protocol: TCP (6)
  - Length: 48 Info: Seq=21 Ack=4217713166 Win=64256 Len=0 TSval=1771713166 TSecr=>21771713166
- Frame 11: 48 bytes on wire (384 bits), 48 bytes captured (384 bits) on interface 'Device\Device\MPF [ED9E847-0B5F-4B0E-A881-4930865EB91]', id 5
  - Source: 192.168.1.10 (192.168.1.10)
  - Destination: 192.168.1.35 (192.168.1.35)
  - Protocol: TCP (6)
  - Length: 48 Info: Seq=22 Ack=4217713167 Win=64256 Len=0 TSval=1771713167 TSecr=>21771713167
- Frame 13: 48 bytes on wire (384 bits), 48 bytes captured (384 bits) on interface 'Device\Device\MPF [ED9E847-0B5F-4B0E-A881-4930865EB91]', id 6
  - Source: 192.168.1.10 (192.168.1.10)
  - Destination: 192.168.1.35 (192.168.1.35)
  - Protocol: TCP (6)
  - Length: 48 Info: Seq=21 Win=4096 Len=0 TSval=1771713167 TSecr=>21771713167
- Frame 14: 48 bytes on wire (384 bits), 48 bytes captured (384 bits) on interface 'Device\Device\MPF [ED9E847-0B5F-4B0E-A881-4930865EB91]', id 7
  - Source: 192.168.1.35 (192.168.1.35)
  - Destination: 192.168.1.10 (192.168.1.10)
  - Protocol: TCP (6)
  - Length: 48 Info: Seq=22 Ack=4217713168 Win=64256 Len=0 TSval=1771713168 TSecr=>21771713168
- Frame 15: 48 bytes on wire (384 bits), 48 bytes captured (384 bits) on interface 'Device\Device\MPF [ED9E847-0B5F-4B0E-A881-4930865EB91]', id 8
  - Source: 192.168.1.35 (192.168.1.35)
  - Destination: 192.168.1.10 (192.168.1.10)
  - Protocol: TCP (6)
  - Length: 48 Info: Seq=22 Win=4096 Len=0 TSval=1771713168 TSecr=>21771713168
- Frame 16: 48 bytes on wire (384 bits), 48 bytes captured (384 bits) on interface 'Device\Device\MPF [ED9E847-0B5F-4B0E-A881-4930865EB91]', id 9
  - Source: 192.168.1.35 (192.168.1.35)
  - Destination: 192.168.1.10 (192.168.1.10)
  - Protocol: TCP (6)
  - Length: 48 Info: Seq=22 Win=4096 Len=0 TSval=1771713169 TSecr=>21771713169

The Wireshark interface shows the following details:

- Frame 4: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 'Device\Device\MPF [ED9E847-0B5F-4B0E-A881-4930865EB91]', id 0
  - Source: 192.168.1.35 (192.168.1.35)
  - Destination: 192.168.1.10 (192.168.1.10)
  - Protocol: TCP (6)
  - Length: 74 Info: Seq=4217713162 Win=64248 Len=48 MSS=1468 SACK\_PERM TS=0.1-1771713162 TSecr=<0 WS=128
- Frame 6: 48 bytes on wire (384 bits), 48 bytes captured (384 bits) on interface 'Device\Device\MPF [ED9E847-0B5F-4B0E-A881-4930865EB91]', id 1
  - Source: 192.168.1.10 (192.168.1.10)
  - Destination: 192.168.1.35 (192.168.1.35)
  - Protocol: TCP (6)
  - Length: 48 Info: Seq=1 Ack=4217713162 Win=64256 Len=0 TSval=1771713162 TSecr=>21771713162
- Frame 8: 48 bytes on wire (384 bits), 48 bytes captured (384 bits) on interface 'Device\Device\MPF [ED9E847-0B5F-4B0E-A881-4930865EB91]', id 2
  - Source: 192.168.1.10 (192.168.1.10)
  - Destination: 192.168.1.35 (192.168.1.35)
  - Protocol: TCP (6)
  - Length: 48 Info: Seq=1 Ack=4217713162 Win=64256 Len=0 TSval=1771713162 TSecr=>21771713162
- Frame 9: 48 bytes on wire (384 bits), 48 bytes captured (384 bits) on interface 'Device\Device\MPF [ED9E847-0B5F-4B0E-A881-4930865EB91]', id 3
  - Source: 192.168.1.10 (192.168.1.10)
  - Destination: 192.168.1.35 (192.168.1.35)
  - Protocol: TCP (6)
  - Length: 48 Info: Seq=21 Ack=4217713162 Win=64256 Len=0 TSval=1771713163 TSecr=>21771713163
- Frame 10: 48 bytes on wire (384 bits), 48 bytes captured (384 bits) on interface 'Device\Device\MPF [ED9E847-0B5F-4B0E-A881-4930865EB91]', id 4
  - Source: 192.168.1.10 (192.168.1.10)
  - Destination: 192.168.1.35 (192.168.1.35)
  - Protocol: TCP (6)
  - Length: 48 Info: Seq=21 Ack=4217713166 Win=64256 Len=0 TSval=1771713166 TSecr=>21771713166
- Frame 11: 48 bytes on wire (384 bits), 48 bytes captured (384 bits) on interface 'Device\Device\MPF [ED9E847-0B5F-4B0E-A881-4930865EB91]', id 5
  - Source: 192.168.1.10 (192.168.1.10)
  - Destination: 192.168.1.35 (192.168.1.35)
  - Protocol: TCP (6)
  - Length: 48 Info: Seq=22 Ack=4217713167 Win=64256 Len=0 TSval=1771713167 TSecr=>21771713167
- Frame 13: 48 bytes on wire (384 bits), 48 bytes captured (384 bits) on interface 'Device\Device\MPF [ED9E847-0B5F-4B0E-A881-4930865EB91]', id 6
  - Source: 192.168.1.10 (192.168.1.10)
  - Destination: 192.168.1.35 (192.168.1.35)
  - Protocol: TCP (6)
  - Length: 48 Info: Seq=21 Win=4096 Len=0 TSval=1771713167 TSecr=>21771713167
- Frame 14: 48 bytes on wire (384 bits), 48 bytes captured (384 bits) on interface 'Device\Device\MPF [ED9E847-0B5F-4B0E-A881-4930865EB91]', id 7
  - Source: 192.168.1.35 (192.168.1.35)
  - Destination: 192.168.1.10 (192.168.1.10)
  - Protocol: TCP (6)
  - Length: 48 Info: Seq=22 Ack=4217713168 Win=64256 Len=0 TSval=1771713168 TSecr=>21771713168
- Frame 15: 48 bytes on wire (384 bits), 48 bytes captured (384 bits) on interface 'Device\Device\MPF [ED9E847-0B5F-4B0E-A881-4930865EB91]', id 8
  - Source: 192.168.1.35 (192.168.1.35)
  - Destination: 192.168.1.10 (192.168.1.10)
  - Protocol: TCP (6)
  - Length: 48 Info: Seq=22 Win=4096 Len=0 TSval=1771713169 TSecr=>21771713169

כמו בתוכנית הקודמת, הלקוח מקבל את IP והפורט מהארגומנטים לתוכנית, לאחר מכן הוא מתחבר לשרת ושולח לו את ההודעה המצוינת. הוא עושה recv מהשרת, ולאחר מכן מקבל את ההודעה שהגיעה חזרה מהשרת הוא מדפיס אותה. מצד שני, כמו בתוכנית הקודמת, השרת מאזור לכל החיבורים כיון שהוא בחר בכתבota 0.0.0.0 ומתקבל בקלט. הוא פותח חיבור, מקבל את ההודעה בצדדים של בתים 5 (כוגדול הבאför), עד שבתיים מפסיקים להגיע, מדפיס כל צאנק, ושולח אותו חזרה ללקוח באופן של zeros. לבסוף, הוא סגור את החיבור.

כמו התפיסה האחורונה, ניתן לראות את לחיצת הידיים בין הלקוח לשרת (שורות 4 ו-5). אך בשונה מהתפיסה האחורונה, השרת קולט את המידע ש מגיע אליו מחיבור אחד בחתיכות של 5 תווים. לכן רואים בכריש את הצאנק הראשון מגיע ומוחזר מהשרת (בזאתםן כמובן), ולפניהם רואים את המשך קבלת הצאנקים בשרת, אנו רואים את ack שהלקוח החזיר על הצאנק הראשון. משום מה שרת מוחדר את 2 הצאנקים הבאים ושולח אותם יחדיו ללקוח, כאשר האחרון מוחזר עליהם ack. עם זאת, מכיוון שיש רק recv אחד בלקוח, הוא ממשיר בשלו ולאחר קבלת הצאנק הראשון הוא שולח fin כדי לסיים את ההתקשרות. השרת מקבל את fin ומוחזר עליה ack, אולם מעתה והילך שאר הצאנקים שהשרת שולח זוכים לمعנה של rst כיון שאין אף אחד בצד השני ששמע וזמן לקבל. ככל שהשרת שולח – הכל מקבל rst, שכן הלקוח ניתק בצד השני...

### גרסת 3:

```

server.py
1 import socket,sys
2
3 TCP_IP = '0.0.0.0'
4 TCP_PORT = int(sys.argv[1])
5 BUFFER_SIZE = 1024
6
7 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
8 s.bind((TCP_IP, TCP_PORT))
9 s.listen(1)
10
11 while True:
12     conn, addr = s.accept()
13     print('New connection from:', addr)
14     while True:
15         data = conn.recv(BUFFER_SIZE)
16         if not data: break
17         print("received:", data)
18         conn.send(data.upper()*1000)
19     conn.close()

```

```

client.py > ...
1 import socket,sys
2
3 TCP_IP = sys.argv[1]
4 TCP_PORT = int(sys.argv[2])
5 BUFFER_SIZE = 1024
6 MESSAGE = b'Hello, World!'
7
8 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
9 s.connect((TCP_IP, TCP_PORT))
10 s.send(MESSAGE)
11 data = s.recv(BUFFER_SIZE)
12 print("received data:", data)
13 data = s.recv(BUFFER_SIZE)
14 print("received data:", data)
15 s.close()

```

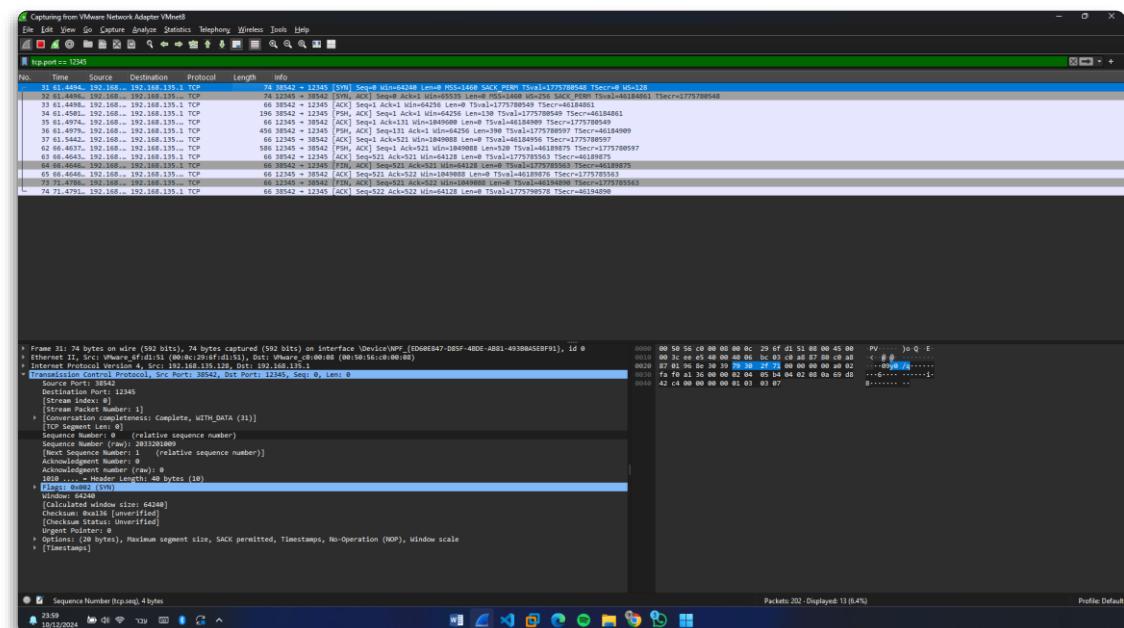
כמו בתוכנית הקודמת, הקוד לא השתנה המון – הלקוח מקבל את התוכנית, לאחר מכן הוא מתחבר לשרת ושולח לו הודעה, אך בשונה מהתוכנית הקודמת, לאחר שקיבל את הודעה חזירה מהשרת הוא מושך ומדפיס את 1024 בתים הראשונים ולאחר מכן שוב הוא מנסה למשוך עוד 1024 בתים. כמו כן בצד שני, כמו בתוכנית הקודמת, השרת מאיין לכל החיבורים כיוון שהוא בחר בכטובת 0.0.0.0 ומקבל פורט נקלט. הוא פותח חיבור, מקבל את 1024 הבטים הראשונים ושולח אותם חזירה ללקוח באoitיות גדולות בהכפלה של 1000 פעמים הלקוח. לבסוף, סוגר את החיבור.

כמו בתפיסה האחורונה, ניתן לראות את לחיצת הידיים בין הלקוח לשרת. מיד לאחר מכן את 13 הבטים שהלקוח שולח, ואז השרת קולט את כל המידע ש מגיע אליו, ושולח אותו באoitיות גדולות כפול 1000 פעמים, لكن יש הרבה שליחות של 12345 (השרת) ל-44550 (הלקוח). השרת שולח את החבילה הראשונה, ולאחר מכן עוד מלא חבילות. הלקוח מאשר את קבלת החבילה הראשונה (1024 הבטים הראשונים) וכן את השניה (אלו שבאים אחריו), מחזיר ack לאופן מאוחד על 13000 הבטים שהגינו, וסגור את החיבור. אבל יש עוד חבילות בבאפר שלא נקבעו ע"י האפליקציה כאשר היא נסגרת, לכן נשלחת הודעה stz לשרת שידע שהלקוח ניתק מבלי לקרוא עד הסוף.

#### גרסה 4:

```
server.py > ...
1 import socket,sys,time
2
3 TCP_IP = '0.0.0.0'
4 TCP_PORT = int(sys.argv[1])
5 BUFFER_SIZE = 1024
6
7 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
8 s.bind((TCP_IP, TCP_PORT))
9 s.listen(1)
10
11 while True:
12     conn, addr = s.accept()
13     print('New connection from:', addr)
14     while True:
15         time.sleep(5)
16         data = conn.recv(BUFFER_SIZE)
17         if not data: break
18         print("received:", data)
19         conn.send(data.upper())
20     conn.close()
```

```
client.py > ...
1 import socket,sys
2
3 TCP_IP = sys.argv[1]
4 TCP_PORT = int(sys.argv[2])
5 BUFFER_SIZE = 1024
6 MESSAGE = b'Hello, World!'
7
8 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
9 s.connect((TCP_IP, TCP_PORT))
10 s.send(MESSAGE*10)
11 s.send(MESSAGE*10)
12 s.send(MESSAGE*10)
13 s.send(MESSAGE*10)
14 data = s.recv(BUFFER_SIZE)
15 s.close()
16
17 print("received data:", data)
```



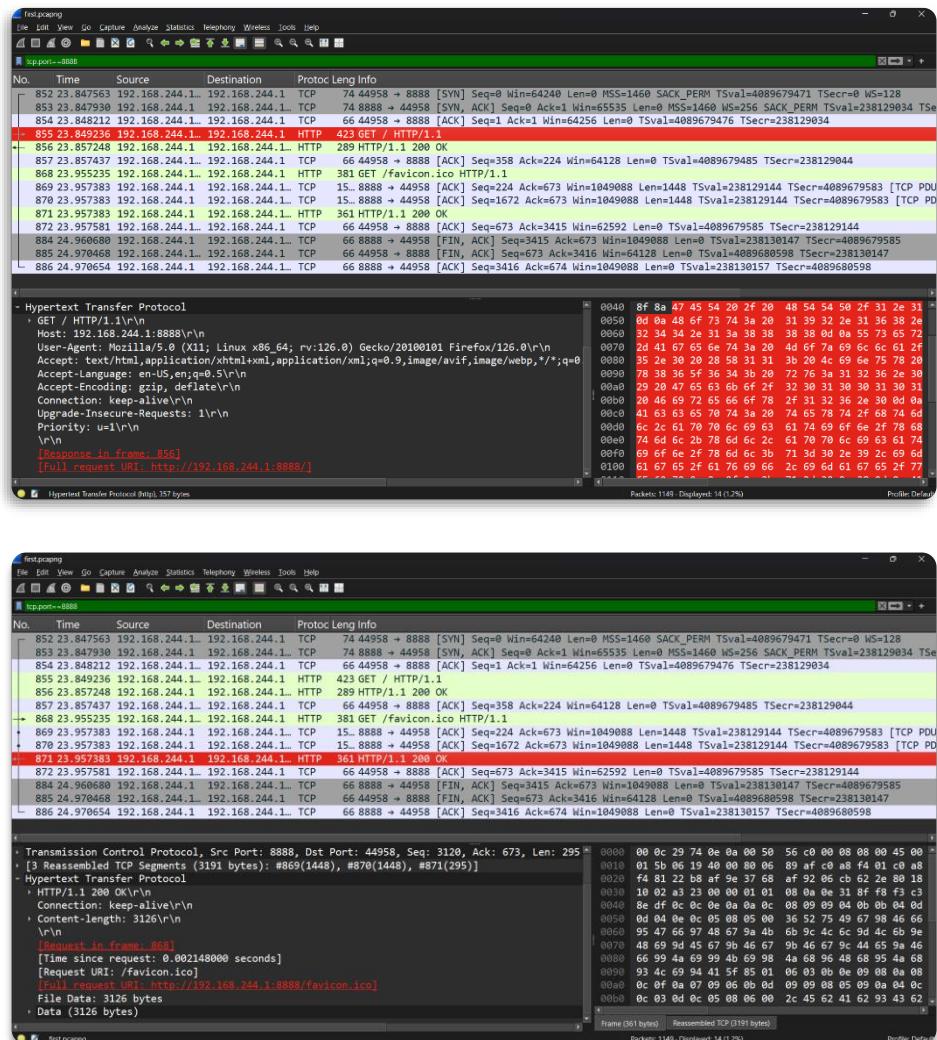
כרגיל הלוקו מקבל את ה-IP והפורט מהארגוני לתוכנית, לאחר מכן הוא מתחבר לשרת ושולח לו 4 הודעות שכל הודעה היא 10 פעמים "Hello, World", השוני הפעם מהתוכנית הקודמת, הוא שלאחר שקיבל את ההודעה חוזרת מהשרת הוא מדפיס את 1024 בתים הראשונים. בצד שני, כרגיל השירות מażין לכל החיבורים כיון שהוא בחר בכתובת 0.0.0.0 ומקבל פорт קקלט. הוא פותח חיבור מכחכה 5 שניות, מקבל את 1024 הבטים הראשונים ושולח אותם חוזרת ללקוח באוטיות גדולות. הוא חוזר על התהיליך עד שהוא מפסיק לקבל מידע מה לקוח ולבסוף סגור את החיבור.

בכרייש אנו רואים תחילת את לחיצת הידיים בין הלקוח לשרת. אך בשונה מהתפיסה האחורונה, השירות מכחכה 5 שניות ולכן התגובה שלו מתעכבות. ניתן לראות כי השילחה (של הלקוח) והתגובה (של השירות) קורית פה 2 פעמים (כנראה 3 השילוחות האחרונות אוחדו בעט שליחתן). בכל פעם השירות (שנמצא בדעתן timeout ב一封יקציה ולן לא קורא מהבאפר), מছיר ack שהוא קיבל את ההודעה. השירות שולח את כל ה-520 בתים בחזרה בפעם אחת (מאוחדת) באוטיות גדולות, ומתקבל ack לעלייה מהשרת. מכאן והילך טקס הפרידה הסטנדרטי של-fin-ack בין השירות ללקוח, כאשר כל אחד מעלה את הפאנטום בית, ונפרדים לשלום. כאן לא הייתה בעיה שלrst בשל העבודה שכל המידע נקרא מהבאפר לפני האפליקציה סיימה את הריצה.

## תיק ב'

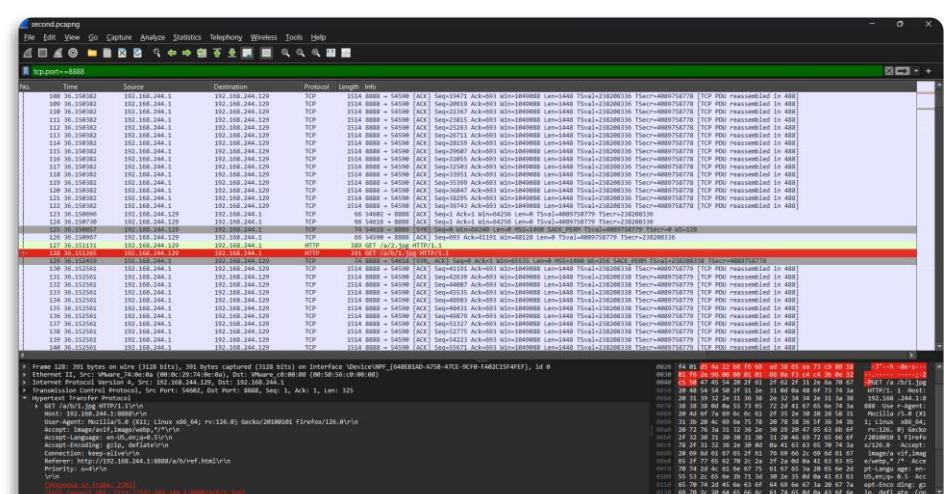
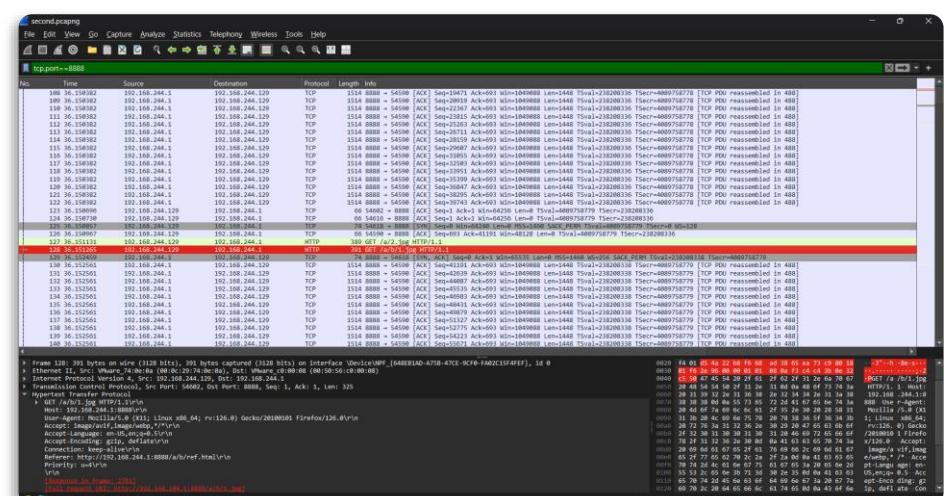
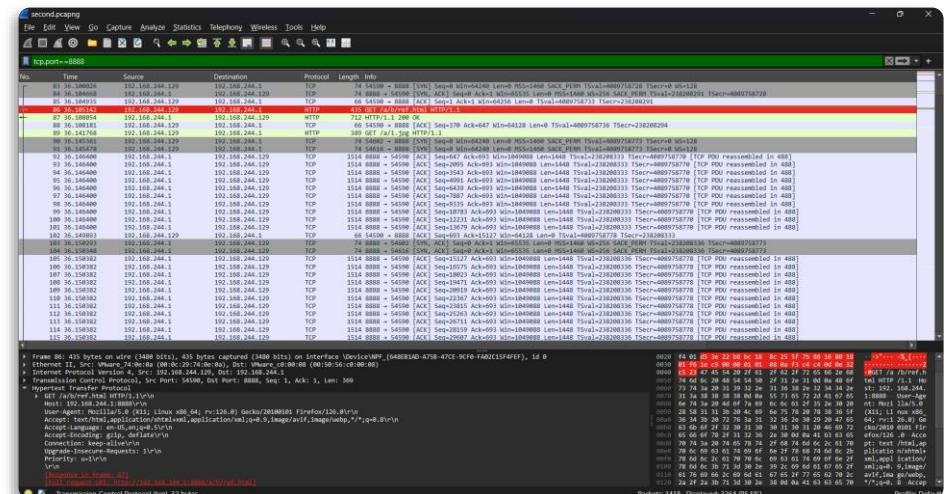
נסים חלק זה של התרגיל בהציג הכריש בזמן ריצת קוד המשרת שלנו, כאשר האחרון מקבל בקשות מהדפסן (אשר רץ בVM).

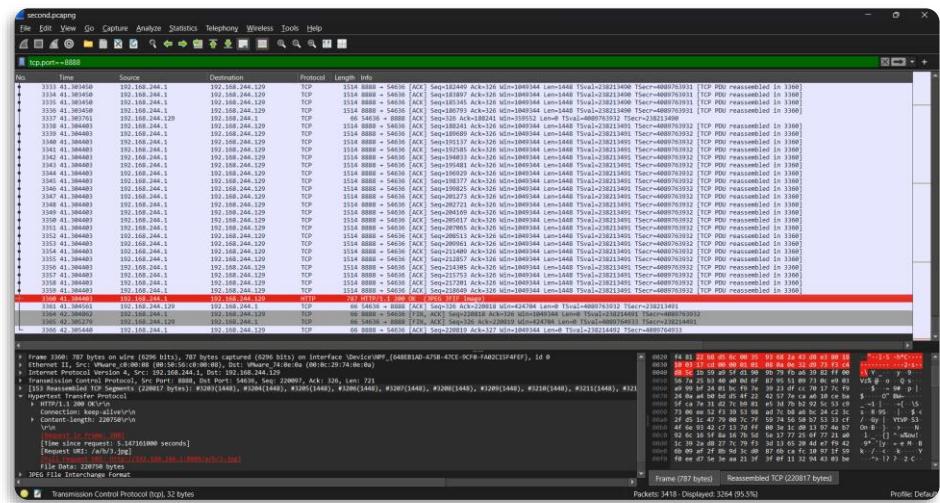
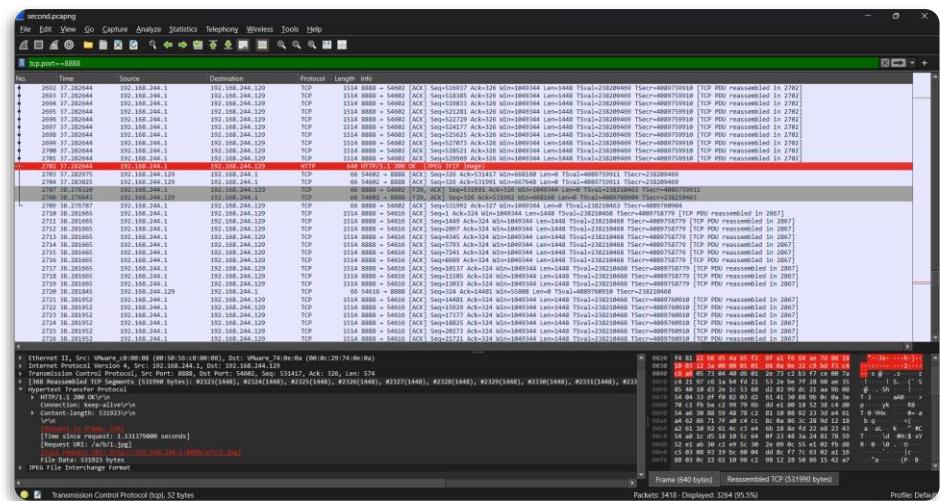
<http://192.168.244.1:8888/>



הכתובת פונה וambilת מהשרת את דף הבית, והוא מוחזיר בתמורה את – index.html. ניתן לראות בתמונה הראשונה שדף עושה מס' פניות מסווג פורט (44958), כאשר תחילתו שלlox את הבקשה של הקובץ, ומתקבל אליו חזרה את ה ok 200, מיד לאחר שהמחליפים ack ביןיהם אודות מעבר הקובץ באופן תקין, הדפסן שולח בקשה get נוספת עבור הקובץ favicon.ico הלוא הוא האיקון של "הarter" שלנו שאנו רואים בקצת הגרפיה בדף, גם על בקשה זו מתקבלת הודעה ok 200 (תמונה שנייה – רואים את הקובץ עליו הוחזר ok למטה body), והתקשרות נסגרת באופן הדדי בחו"ם משני הצדדים. ניתן לראות שנשליח המונן מידע עם הודעה get מדף, כמו סוג הדפסן, הגרסה שלו, סטטוס connection, עדיפות ועוד. מעבר לget, ההחזרה של המידע מהשרת – ok 200, אז הבקשה לאיקון אין מיוחד, ואין בקשה נוספת פורט.

<http://192.168.244.1:8888/a/b/ref.html>

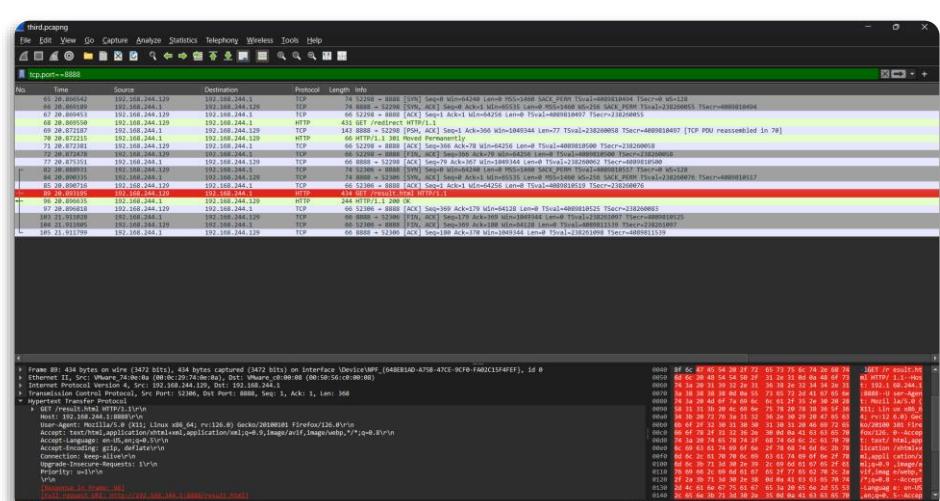
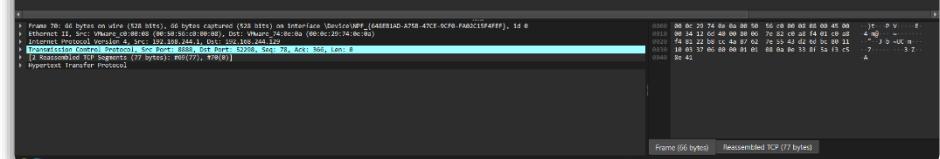
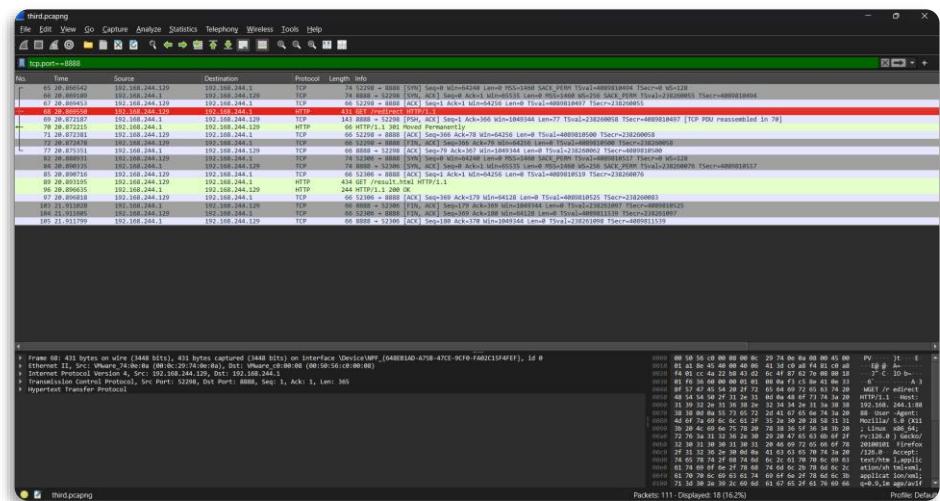




icut מבקש הדפסן את קובץ בתוכו מס' קבצים (תמונה) נוספים. ניתן לראות שהפעם הוא פותח מס' חיבורים (כ-שלואה) דרך כמה פורטים (תמונה ראשונה), אבל בפועל בשלב הראשון רק דרך החיבור הראשון עוברת התקשרות של הבקשה מהשרת. לאחר שהוא מבקש את הדף הראשון (זה שבקשו בכתוב), הוא מקבל אותו ומתחילה בקצבות כבדות מאוד של התמונות המופיעות באוטו קובץ. נשים לב שהתמונה מועברת בהמון צאנקים, אך לאחר כמה זמן הוא מנסה לבקש דרך אחד הפורטים האחרים שהוא פתח תמונה נוספת מזו שבקש דרך הפורט הראשי (תמונה שנייה), אך למרות זאת עדין התקשרות נספתחת בשונה. ואכן רואים כיצד התמונה שנטבעה בפורט השונה (45602) מתחילה להציג גם שלישית), ואכן רואים כיצד התמונה שנטבעה בפורט השונה (45602) מתחילה להציג גם היא. אותו דפוש פעולה חוזר על עצמו (בתמונה הריבועית אנו רואים את ה `get` על התמונה שהגיעה דרך הפורט (45602), כאשר מס' תמונות מגיעות דרך כמה פורטים שונים, אך אין ערבות והכל מתבצע בצאנקים, דהיינו רק לאחר שפורט קיבל את התמונות שלו ונסגר (`fin` הడד), אנחנו רואים שהתמונה שבקש פורט אחר מתחילה להציג (זה ועוד נובע מהמסטור במקבילות באופן שבו בנוינו את השרת, ובשל כך שאר הפורטים תקווים בחסונה כאשר האחד מקבל שירותים מהשרת). התמונות שהן די גדולות מועברות בהמון צאנקים (כנראה בגלל שאנחנו עובדים מחשב-VM), ואחת לכך זמן אנו רואים במהלך התקשרות ack שחודר עליהם מהלכו לשרת, כאשר בסיסים העבירה של כל קובץ תמונה נראה שmagui "מאסף". שוב בקצבות `get` מכך באתו מידע י Bush על הדפסן ושאר התשתיות. כך זה מתבצע באופן

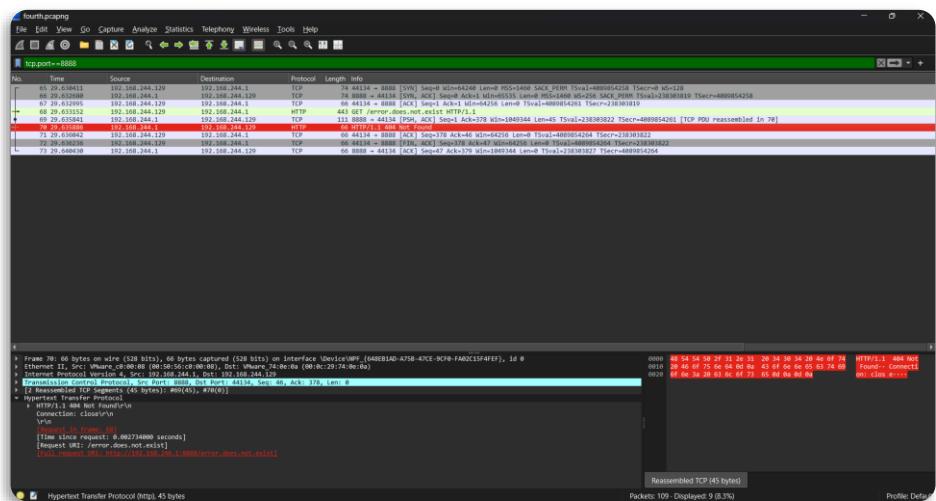
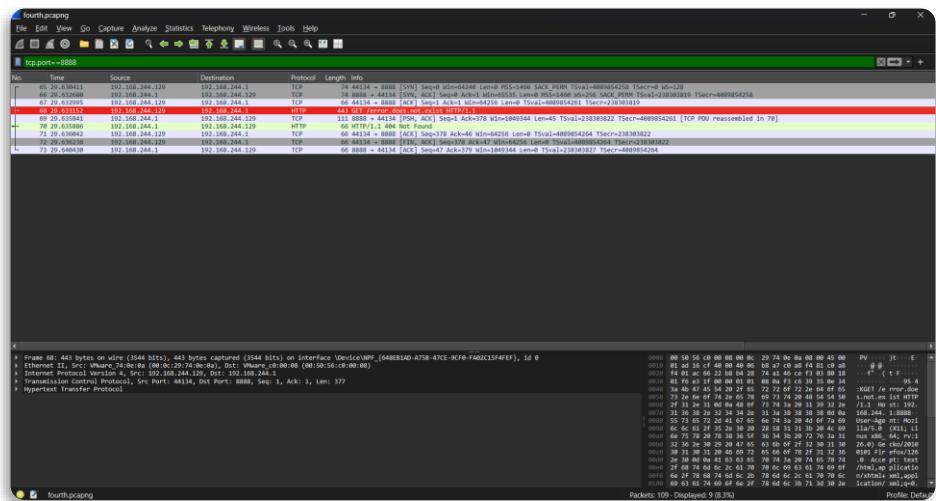
איטרטיבי עד שבסופו של דבר הפורט האחרון שבירקש את התמונה האחרונות נסגר גם הוא, והדף מציג את הדף בשלמותו (תמונה חמישית).

<http://192.168.244.1:8888/redirect>



הדף פונה כתע אל הנטייב /redirect של השרת שלנו. ניתן לראות שהפעם הוא פתוח תחילה חיבור יחיד של פורט אחד, אשר דרכו הוא שולח את הפניה (תמונה ראשונה). השרת שולח לו תשובה חזירה של 'moved permanently', מctrפ' לשובה שלו את הפניה/הכתובת של מקום החדש, והחיבור נסגר (תמונה שנייה). מיד לאחר מכן מוקם חיבור חדש מצד הדף, אשר דרכו מתבצעת הבקשה לאוטומטית location='result.html' חדש שהציג השרת לדף בבקשת הקודמת (תמונה שלישי). השרת מקבל את הבקשה השנייה, מחזיר את הקובץ שביקש בדף – result.html, ולאחר מכן רואים את הסגירה של התקשרות בחוף שגיאי משני הכוונים. ברמת המטא-דאטא אין שוני יוצא דופן מאשרינו כבר.

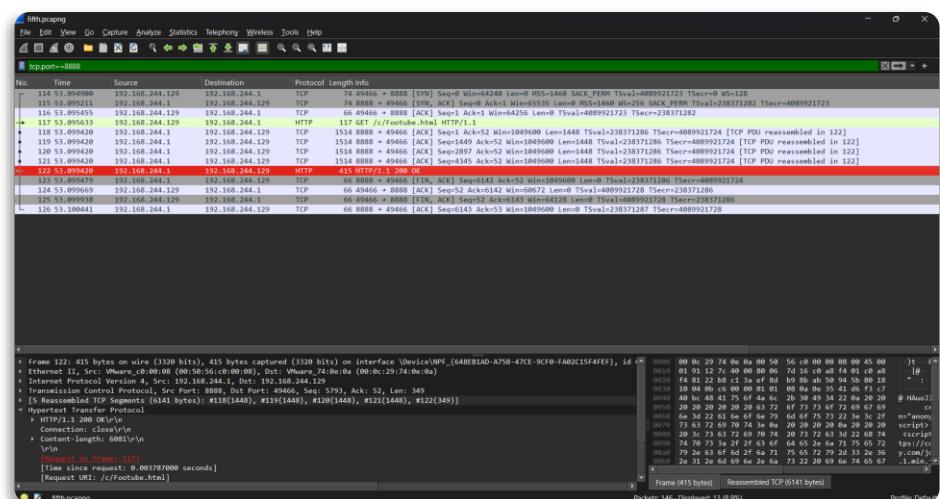
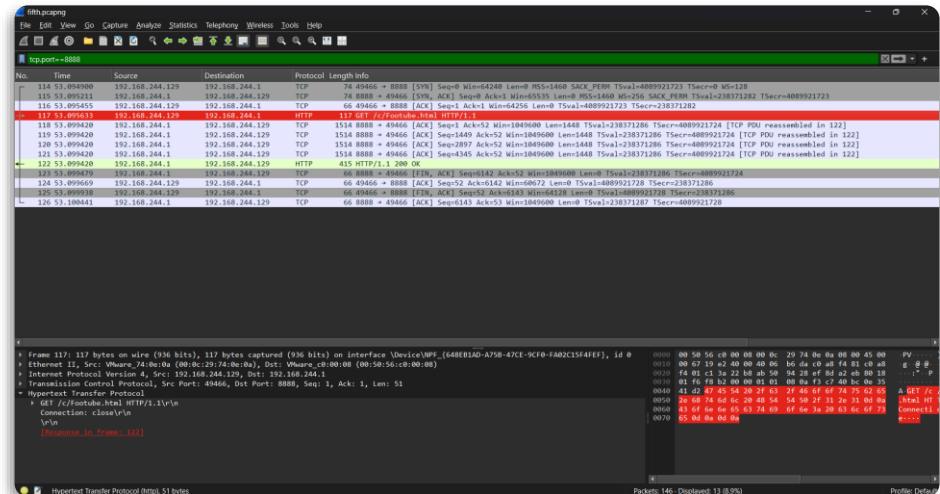
<http://192.168.244.1:8888/error.does.not.exist>



אנו רואים כתע ניסיון של הדף לפנהות לכתובת אשר לא קיימת מבינת השרת שלנו. אנו שונפתח שוב חיבור יחיד של פורט אחד, אשר דרכו נשלחת הבקשה. השרת אשר שולח על כך קיבל את הבקשה (תמונה ראשונה), מיד מחזיר לדף תשובה חזירה של status\_code='404 not found' לא נמצא – '404 not found' (תמונה שנייה). לאחר שהתשובה מוחזרת, התקשרות נסגרת בחוף. ברמת המטא-דאטא שוב אין שוני יוצא דופן מאשרינו כבר.

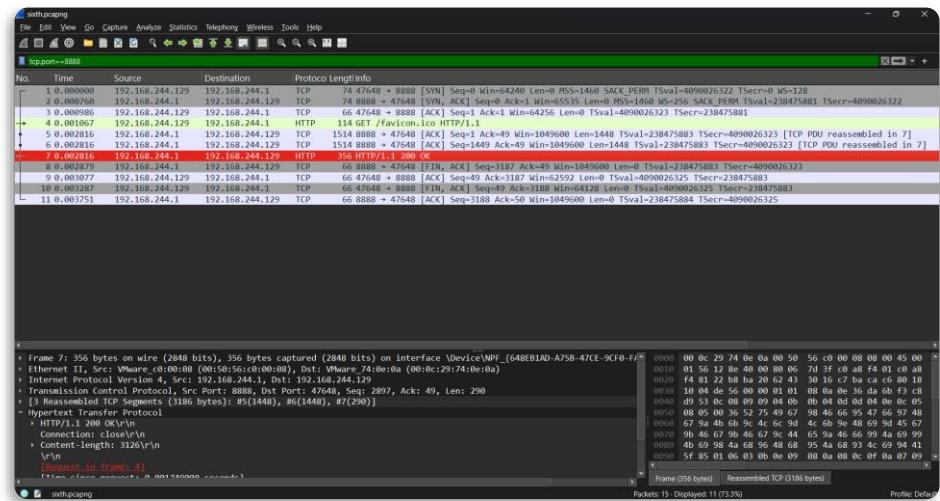
נסים בכך ש נראה מס' בקשות בכירש אשר הטענו אל הרשות אך הפעם לא מהדפסן אלא  
מיקוד הקליינט שאנו חנו כתבנו –

/c/Footube.html



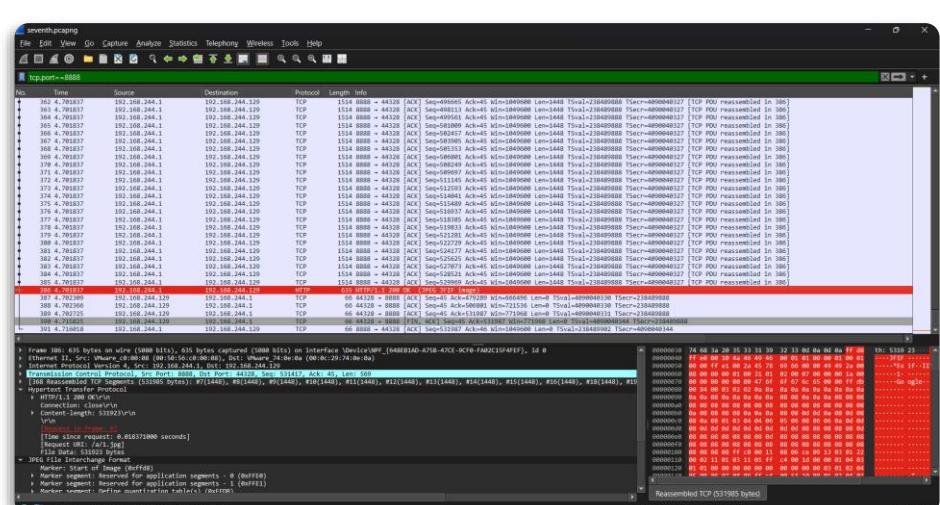
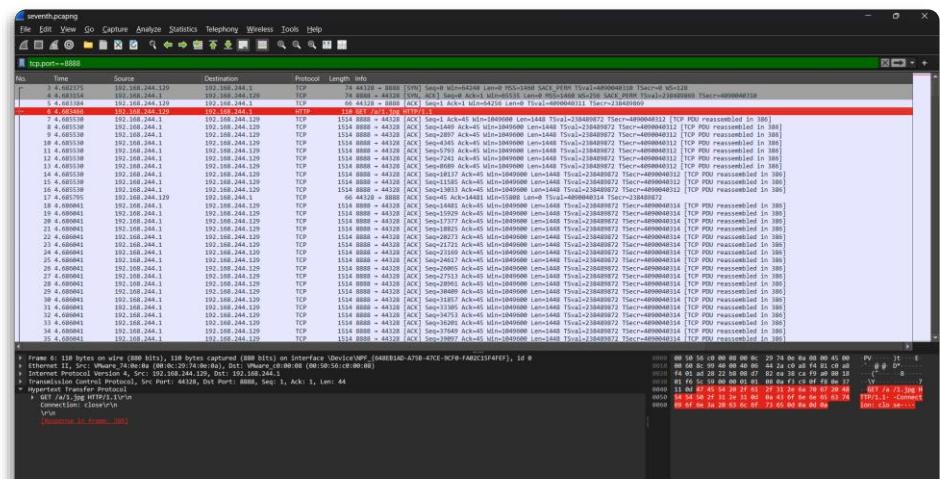
הלקוח מבקש את הקובץ [/c/Footube.html](#) עם כל "עמוד אינטרנט" עם שלל תמונות וקבצים המוכלים בו, אולם אין לנו רואים פורט יחיד אשר נפתח ע"י הלקוח ופונה לרשות, ויתר על מצאת לאחר שמתකבל הקובץ אין לא רואים שרשות של בקשות בדומה למה שראינו כאשר הפניה הגיעה מהדפסן, כיוון שפה הלקוח רק מבקש את הקובץ ולא מנסה להציגו, אולם לו היה מנסה היה נתקל בכל מיין קבצים ומשאים אשר אין, ובאמת הינו רואים שרשות בקשות היה המנסה ליבא את כל המשאים המוחזקים באופן זה או אחר אצל הרשות, אשר אלו דרישים באופן חינוי על מנת להציג את הקובץ במלואו. לאחר שהקובץ מגיע התקשורת נסגרת בחfine הדדי. יש לציין שניתן לראות שהheaders הרבה הרובה יותר דليل מזה שראינו בבקשת שהגיעו מהדפסן, שכן שאנו חנו בקוד הקליינט כתבנו בפניה רק את – סוג הפניה, שבו הקובץ המבוקש, הפרוטוקול בו אנו משתמשים, וכן סטטוס החסיטון.

/favicon.ico



דוג' פשוטה נוספת בה הלקוח מבקש את הקובץ של האיקון של הרשת (התמונה הקטנה המופיעה בראש הכרטיסיה), אנו רואים שהשרת מছזר את התוכן ב2 חבילות שונות (מפתחת גודל), ומאסף זאת בהודעתה נשלם, אנו רואים את הסגירה של התקשרות.

/a/1.jpg



דוג' אחרונה אשר בה הלקוח מבקש את קובץ של תמונה מהשרת. מכיוון שהתמונה מאד גדולה עבור השירות (וכן אנו עושים פה מעבר מהמחשב ל��ריז לינוקס), אנו רואים שהשרת מחזיר את התוכן בהמונ המונ חבילות שונות, ורק בסופו של דבר לאחר מעבר של חמיש מאות אלף בתיים, נשלחת הודעה הסיכום של ה- ok 200. לאחר שזו מגיעה נסגרת התקשרות באופן תקין. ניתן לשים לב אגב שאם אנו פותחים את הליבל של הקונטנט כאשר נשלחת תמונה, ניתן לראות כל מיני פרטיים עליה ועל סוגה על מנת שייהי ניתן לפתח ולפענה אותה בצד השני (מידע זה נשלח בודאי גם בדוג' הקודמות כאשר שלחנו תמונה בגין הודעה).