מטלה מס' 2 – אילאי בן יהושע ונועם ליבוביץ

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

tcp_server.py

```python
import socket
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind(('', 12345))
server.listen(5)

while True:
    client_socket, client_address = server.accept()
    print('Connection from: ', client_address)
    data = client_socket.recv(100)
    print('Received: ', data)
    client_socket.send(data.upper())
    client_socket.close()
    print('Client disconnected')
```

tcp_client.py

```python
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(('192.168.135.1', 12345))
s.send(b'Noam and Elay')
data = s.recv(100)
print("Server sent: ", data)
s.close()
```

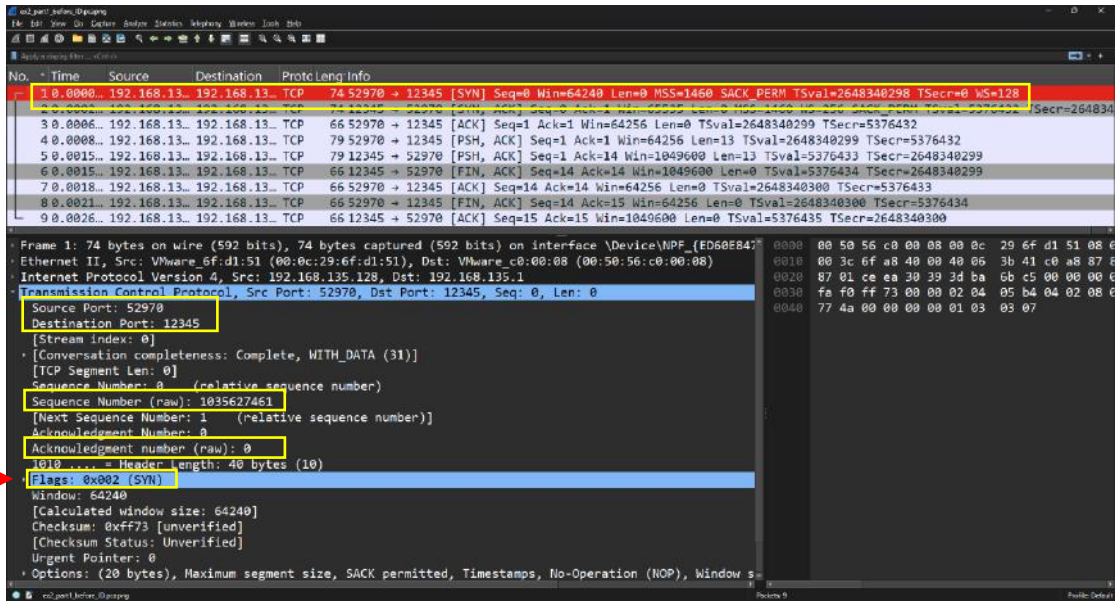
כעת נריץ את הקודים המעודכנים הנ"ל, נסניף את המידע בעזרת "כריש-הכבל" (בלעז – wireshark, מעתה והילך כאשר נכתוב 'כריש' בקיצור נתייחס למינוח הזה), וננתח את מה שקרה פה בהתאם לעקרונות TCP שראינו בהרצאה ובתרגול.

זה מה שמוצג לנו בכריש לאחר הרצת הקודים -

No.	Time	Source	Destination	Protc	Leng	Info
1	0.0000…	192.168.13…	192.168.13…	TCP	74	52970 → 12345 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=2648340298 TSecr=0 WS=128
2	0.0002…	192.168.13…	192.168.13…	TCP	74	12345 → 52970 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM TSval=5376432 TSecr=264834
3	0.0006…	192.168.13…	192.168.13…	TCP	66	52970 → 12345 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2648340299 TSecr=5376432
4	0.0008…	192.168.13…	192.168.13…	TCP	79	52970 → 12345 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=13 TSval=2648340299 TSecr=5376432
5	0.0015…	192.168.13…	192.168.13…	TCP	79	12345 → 52970 [PSH, ACK] Seq=1 Ack=14 Win=1049600 Len=13 TSval=5376433 TSecr=2648340299
6	0.0015…	192.168.13…	192.168.13…	TCP	66	12345 → 52970 [FIN, ACK] Seq=14 Ack=14 Win=1049600 Len=0 TSval=5376434 TSecr=2648340299
7	0.0018…	192.168.13…	192.168.13…	TCP	66	52970 → 12345 [ACK] Seq=14 Ack=14 Win=64256 Len=0 TSval=2648340300 TSecr=5376433
8	0.0021…	192.168.13…	192.168.13…	TCP	66	52970 → 12345 [FIN, ACK] Seq=14 Ack=15 Win=64256 Len=0 TSval=2648340300 TSecr=5376434
9	0.0026…	192.168.13…	192.168.13…	TCP	66	12345 → 52970 [ACK] Seq=15 Ack=15 Win=1049600 Len=0 TSval=5376435 TSecr=2648340300

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

חבילה מס' 1 –



אנו רואים פניה שמתבצעת לPORT - 12345 (אשר אותו נתנו ידנית לשרת), אשר מגיעה מפורט מקור 52970 אשר ניתן דינאמית ללקוח ע"י מ"ה (כיוון שלא צמדנו אותו לפורט ספציפי). הפנייה היא מסוג syn (ניתן לראות את הדגל הדלוק), כלומר הלקוח מבקש להסתנכרן עם השרת וזוהי תפקידה של החבילה הנ"ל עם דגל הsyn, אשר מהווה את השלב הראשון בטקס 'לחיצת הידיים המשולשת'. בנוסף לכך אנו רואים את ערך האופסט הגולמי (= raw) של הseq

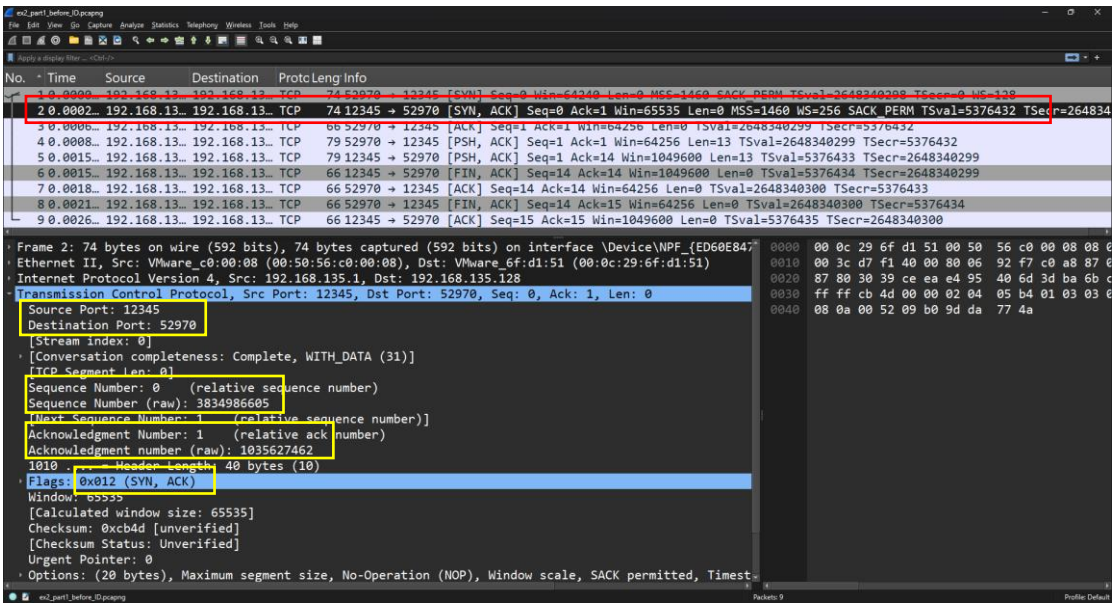
שבחר הלקוח, כאשר ניתן לראות שהוא החליט שתחילת התקשורת תתחיל מאופסט – '1305627461', ואילו ברור לנו שערך הack שלו כרגע יהיה '0', כיוון שהack שלו, זה הseq של השרת (אשר טרם נבחר..). כמובן שהlen הוא 0 שהרי אין דאטא בהודעת syn.

דבר **מעניין** נוסף ניתן לראות כאשר נפתח את הoptions –

```
- Options: (20 bytes), Maximum segment size, SACK permitted, Timestamps, No-Operation (NOP), Window s
  › TCP Option - Maximum segment size: 1460 bytes
  › TCP Option - SACK permitted
  › TCP Option - Timestamps: TSval 2648340298, TSecr 0
  › TCP Option - No-Operation (NOP)
  › TCP Option - Window scale: 7 (multiply by 128)
› [Timestamps]
```

ובו, הלקוח מציין בפני השרת – "רק שתדע, הmss שלי הוא 1460b.".

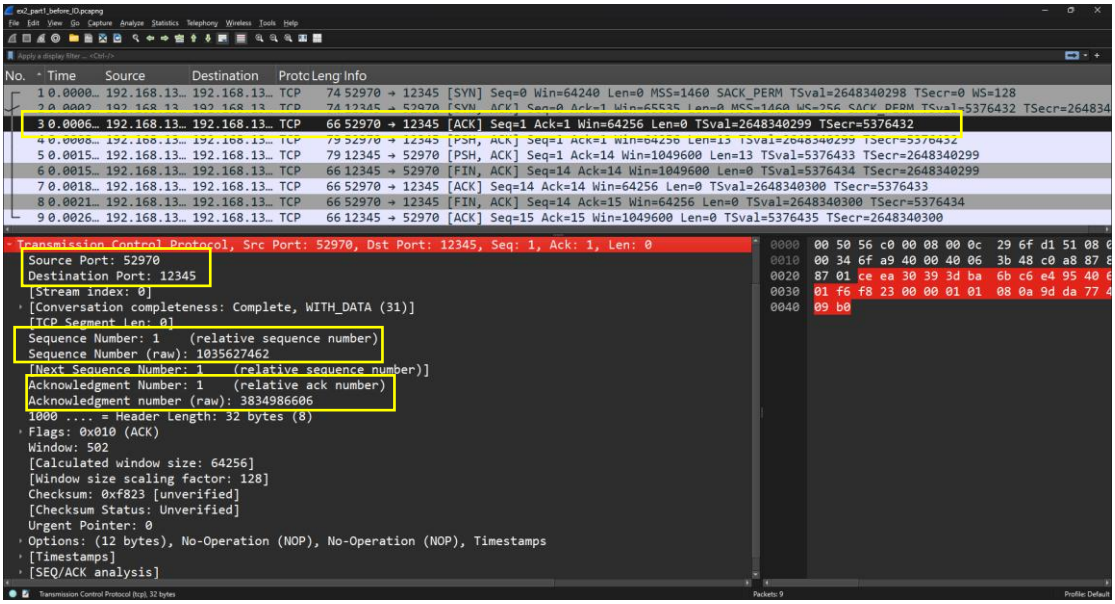
<u>חבילה מס' 2 –</u>



אנו רואים פניה שמתבצעת הפעם בכיוון ההפוך מPORT - 12345 (השרת), אשר מיודעת לפורט 52970 (הלקוח). הפנייה (או יותר נכון 'תגובה') מצד השרת הינה מסוג syn, ack (גם פה ניתן לראות את **שני** הדגלים דלוקים), כלומר <u>השרת ראשית מכיר בסנכרון של הלקוח איתו, אך בנוסף הוא מבקש להסתנכרן גם הוא עם הלקוח.</u> זוהי למעשה תפקידה של החבילה הנ"ל עם דגלי הsyn, ack, אשר מהווה את השלב השני בטקס 'לחיצת הידיים המשולשת'. ראוי לציין כי השלב הנ"ל עשוי להתבצע לעיתים ב2 חבילות שונות (1 לack ואחת לsyn) אך כמו שאנו רואים כאן, אלו יכולות להישלח גם כן יחדיו בחבילה אחת. בנוסף לכך אנו רואים את ערך האופסט הגולמי (= raw) של הseq שבחר השרת, כאשר ניתן לראות שהוא החליט שתחילת התקשורת תתחיל מאופסט – '3834986605', ואילו נשים לב שערך הack שלו יהיה '1305627462', שזה 1 יותר משהיה הseq של הלקוח בגלל הphantom-bit. גם כאן הlen הוא 0 שהרי אנחנו עדיין לא בשלב של העברת דאטא, אך מאוד מתקרבים לשם. אם נפתח את הoptions נשים לב שגם השרת מציין בפני הלקוח מהו הmss שלו – 1460b, כדי שזה ידע את הנתון כאשר הוא שולח אליו הודעות.

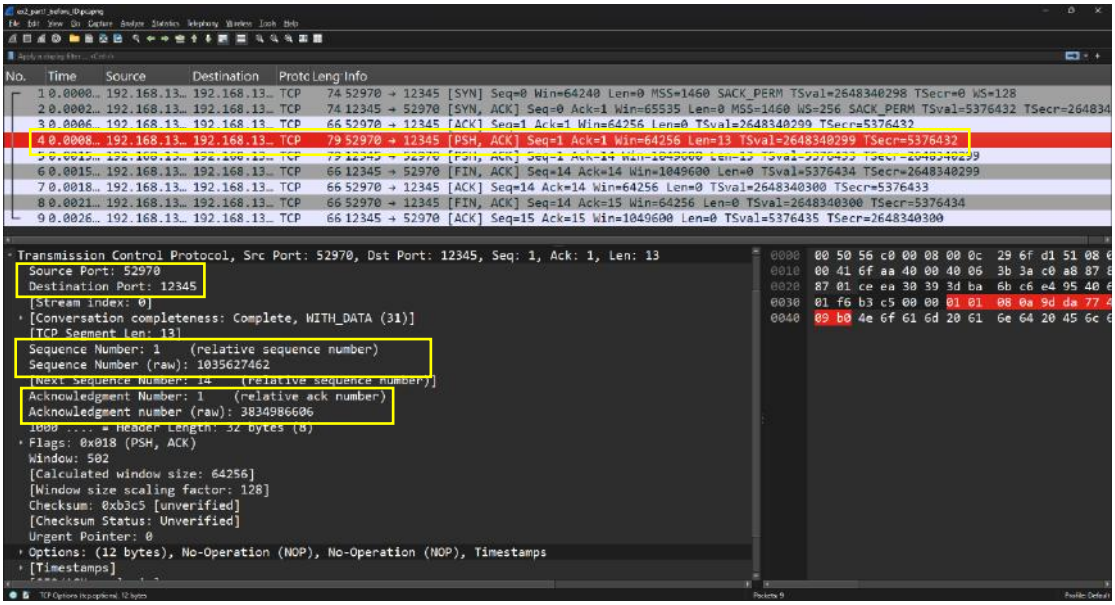
```
  Urgent Pointer: 0
- Options: (20 bytes), Maximum segment size, No-Operation (NOP), Window scale, SACK permitted, Timest
  › TCP Option - Maximum segment size: 1460 bytes
  › TCP Option - No-Operation (NOP)
  › TCP Option - Window scale: 8 (multiply by 256)
  › TCP Option - SACK permitted
  › TCP Option - Timestamps: TSval 5376432, TSecr 2648340298
› [Timestamps]
› [SEQ/ACK analysis]
```

חבילה מס' 3 –



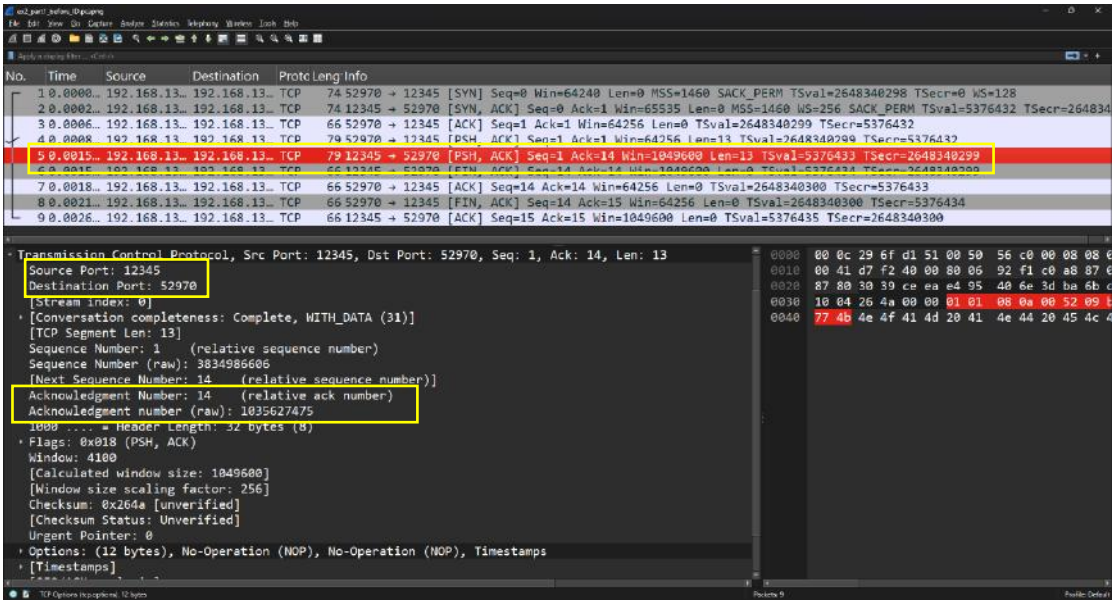
כעת ניתן לראות פניה נוספת מהלקוח 52970 (הלקוח), אל השרת. הפנייה מצד השרת הינה מסוג ack (שוב הדגל דלוק ולמעשה מעתה והילך הוא יהיה דלוק עד סוף התקשורת, אז נפסיק לציין זאת...), דהיינו <u>הלקוח גם הוא מכיר בהכרת השרת בסנכרון שלו</u>, ולמעשה תפקידה של החבילה הזו מהווה את השלב השלישי **והאחרון** בטקס 'לחיצת הידיים המשולשת', שכן שני הצדדים הסתנכרנו אחד עם השני, וכל אחד מהם הכיר בסנכרון המשותף, ומכאן והילך הקשר שריר וניתן להעביר דרכו הודעות. שוב אנו רואים את ערך האופסט הגולמי (= raw) של הack ששולח הלקוח, והוא – '3834986606', באופן לא מפתיע זה 1 יותר מהseq שבחר השרת (שוב אנחנו מגדילים בגלל הפאנטום ביט של הack). הseq לעומת זאת לא השתנה שכן לא נשלח עוד דאטא, וכתוצאה מכך שוב הlen הוא 0 (אך לא לעוד הרבה זמן). אם נפתח את הoptions נשים לב שהmss שלו כבר לא נשלח, שכן שני הצדדים כבר יודעים ומודעים למגבלות כל אחד של השני.

חבילה מס' 4 –



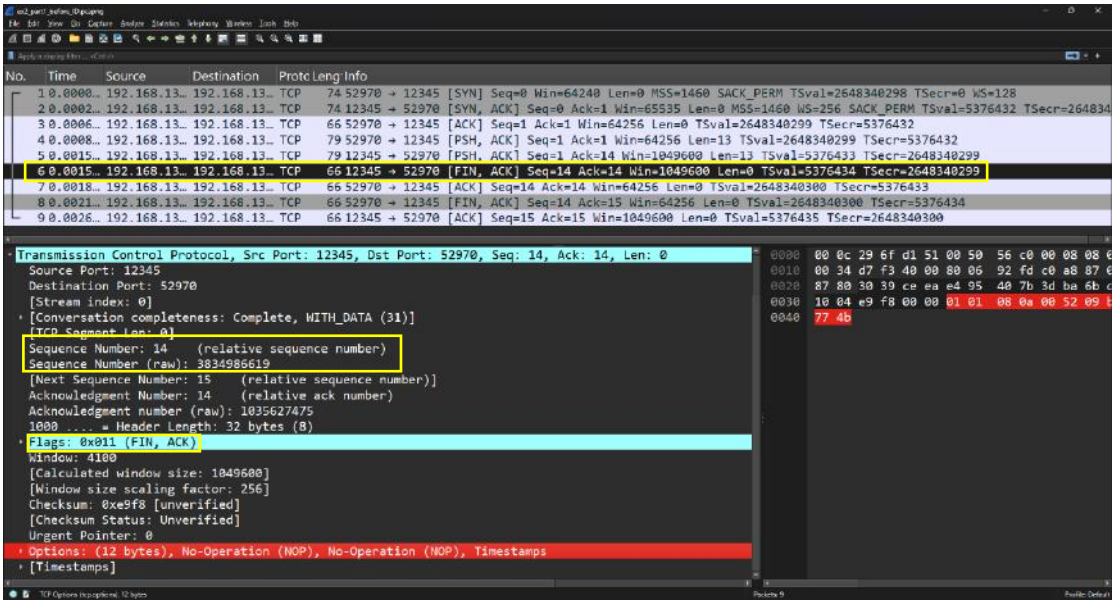
סוף כל סוף הגענו לחלק המעניין ('פסגת הקורס' לפי חלק מהשיטות), אנו רואים הודעה מהלקוח (לפי הפורט 52970 שראינו קודם שהוא הפורט של הלקוח), השולח 13 בתים (לפי הlen), שאנחנו יודעים שהם הבתים של המחרוזת 'Noam and Elay' (כנדרש במטלה), ובנוסף לדגל הack (שאמרנו שמעתה והילך תמיד יהיה דלוק), גם כן דלוק דגל הpush שכאילו אומר להעביר בדחיפות לאפליקציה את המידע שהגיע. נשים לב שעכשיו יש לנו גם את שכבת הdata בהודעה (הבתים ששלחנו מאופן מקודד).

<u>חבילה מס' 5 –</u>



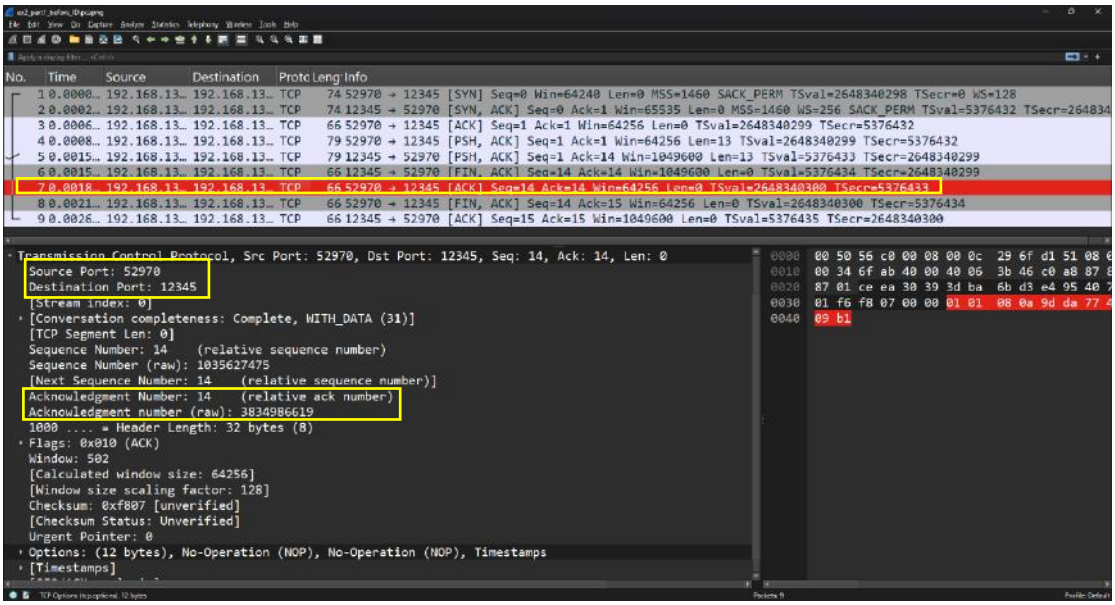
כעת נראה את תגובת השרת להודעה שקיבל מהלקוח בחבילה הקודמת שסקרנו. השרת ראשית **מעדכן** את המספר בשדה הack להיות 13 יותר משהיה – '1035627475' שכן עכשיו הוא אומר ללקוח – "שומע?! קיבלתי כבר **עד** בית מס' 1035627475 תן לי ממנו **והלאה**.". אמנם הseq לא משתנה כיוון שהוא טרם שלח דאטא משל עצמו ללקוח, אך כעת הוא מצרף להודעת הack את התוכן שקיבל בupper (ושוב אנו לא מופתעים שהlen הוא 13).

חבילה מס' 6 –



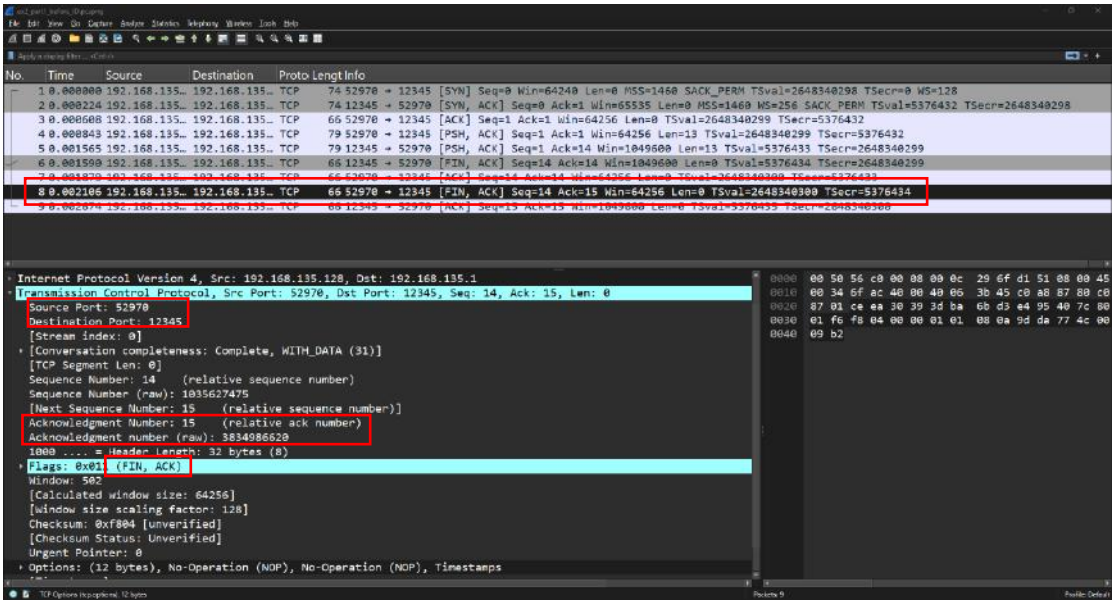
מיד לאחר שהשרת שולח את הודעת הack יחד עם התוכן בupper, הוא שולח הודעת fin (ניתן לראות דגל דולק), שכן הוא עשה את שלו, <u>ותפקיד ההודעה הזו להודיע על כך שהוא מעוניין לסיים את מערכת היחסים עם הלקוח</u>. נשים לב שעכשיו הseq של השרת גדל ב13 גם הוא לערך – '3834986619'.

חבילה מס' 7 –



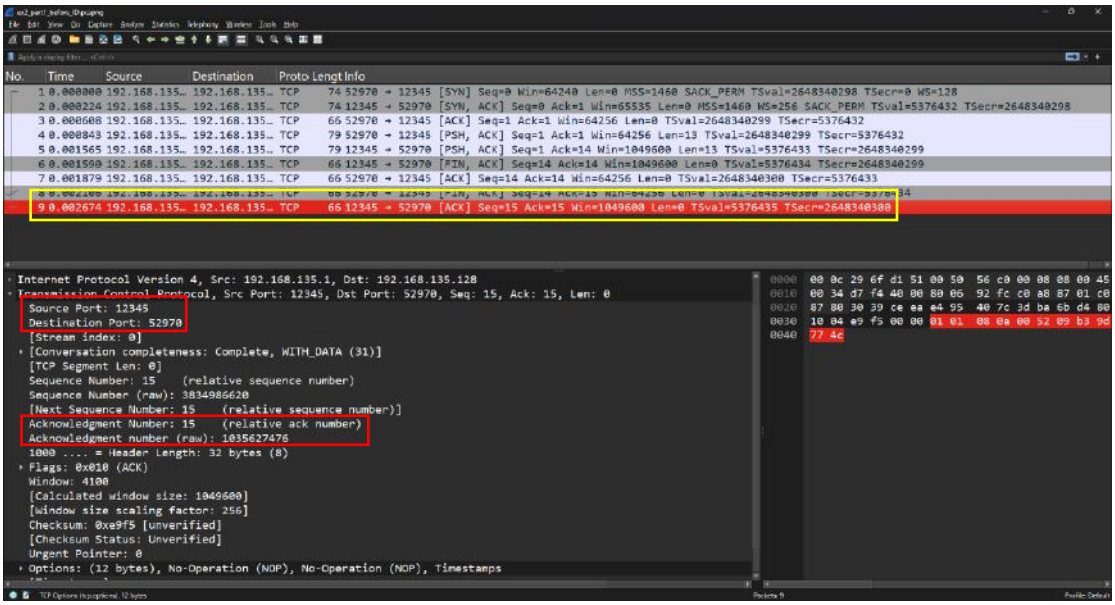
הלקוח מקבל את הדאטא חזרה מהשרת בupper, בהתאמה הוא מגדיל את ערך הacked ב13 בתים נוספים לערך – '3834986619'. כמו כן נשים לב שגם ערך הseq המצוין בהודעה גדל ב-13 מהפעם האחרונה בה ראינו חבילה נשלחת מהלקוח, וכעת הוא עומד על – '1035627475' (שוקינג זה הack של השרת..).

חבילה מס' 8 –



הלקוח כעת שולח הודעת ack לשרת, ולמעשה מעדכן שהוא קיבל את ההודעה שלו בנוגע לסיום ההתקשרות עימו. 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 הלקוח גדל ב-1 כתוצאה מהפאנטום ביט של הודעה fin שמוכרת רק עכשיו.

חבילה מס' 9 –



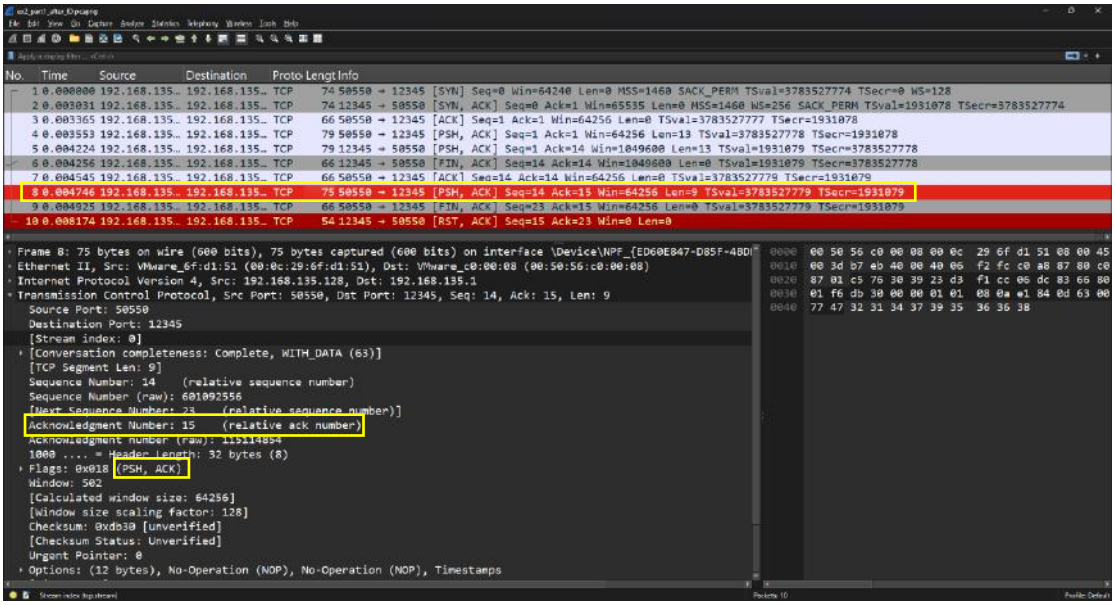
בחבילה האחרונה (☹️) שלנו אנו רואים את השרת מכיר בסיום ההתקשרות מצד הלקוח גם הוא בהודעת ack, ובאופן פרקטי תפקיד ההודעה הינו לסיים **לחלוטין בהחלט** את ערוץ התקשורת שהיה ביניהם. נשים לב שרגע לפני שהוא אומר "בייי", הוא מגדיל גם הוא את ערך ack כתוצאה מהפאנטום ביט של fin שהגיע מהלקוח, סה"כ ערך ack של השרת בסיום התקשורת עומד על – '1035627476'.

סיימנו לנתח את התקשורת בחלק הראשון. נוסיף כעת את שליחת ה-ת.ז. של אילאי לאחר שהשרת משיב את תשובתו ונראה את השינויים.

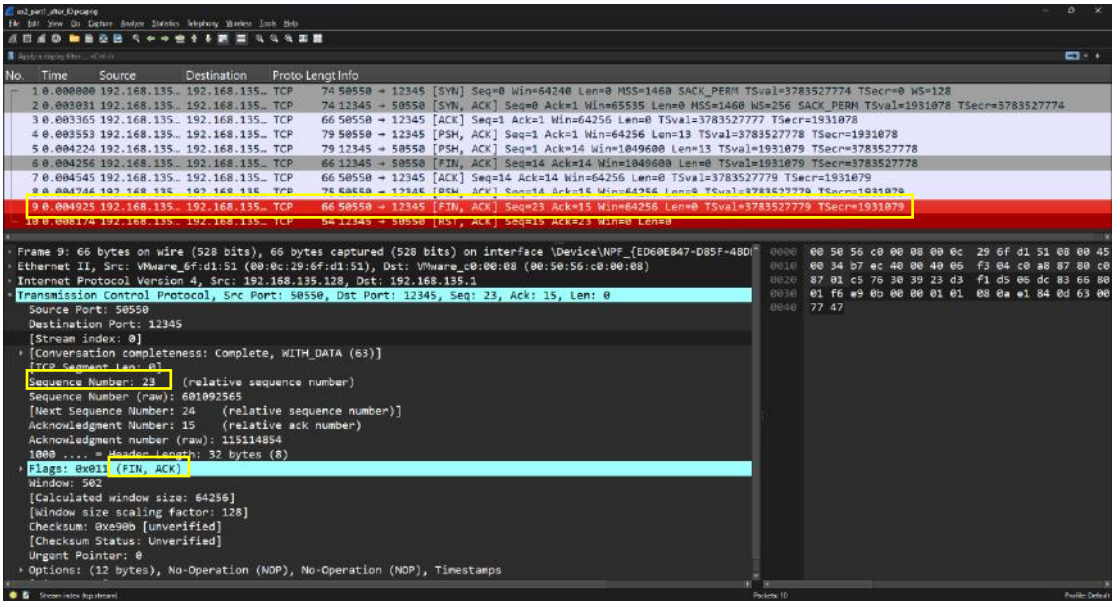
```python
import socket
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind(('', 12345))
server.listen(5)

while True:
    client_socket, client_address = server.accept()
    print('Connection from: ', client_address)
    data = client_socket.recv(100)
    print('Received: ', data)
    client_socket.send(data.upper())
    client_socket.close()
    print('Client disconnected')
```

```python
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(('192.168.135.1', 12345))
s.send(b'Noam and Elay')
data = s.recv(100)
print("Server sent: ", data)
s.send(b'214795668')
s.close()
```

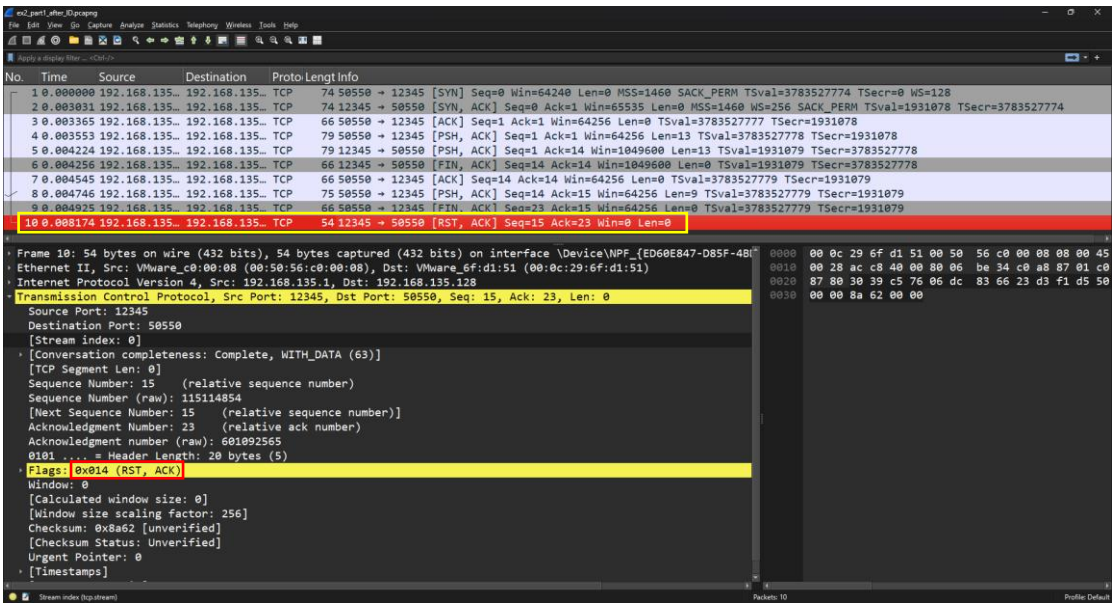
כל ההודעות בהתחלה תתנהגנה באופן זהה (לכן נחסוך ונתחיל רק מהמקום בו השינוי קורה), אבל עכשיו נשים לב –



לאחר שהלקוח החזיר ack על הfin של השרת (פאנטום ביט מגדיל ב1 את הacked), במקום לשלוח גם הוא fin (כמו שקרה קודם), הוא מנסה לשלוח הודעה נוספת ולעשות לה psh לאפליקציה (הלוא היא הת.ז.), ורק לאחר שההודעה הזו נשלחת, הוא שולח גם כן fin מצידו –



הseq שלו גדל ב-9, והוא מסכים עכשיו לסיים את התקשורת. אלא שבגלל שהשרת לא ידע שעליו לצפות לחבילה נוספת הוא כבר סיים וסגר את העניין, ולכן נקבל ממנו הודעת rst –



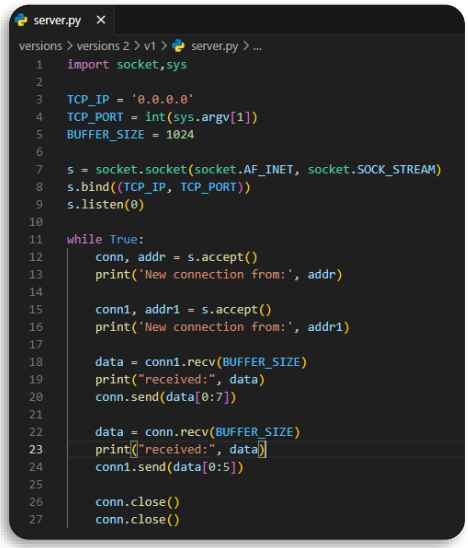
כאילו רוצה לומר – "חדש אח שלי, חדש. מה שהיה אני כבר לא שם, ואין מי שישמע ויקבל את מה שיש לך לשלוח. אם אתה רוצה בוא נתחיל מההתחלה, ומשם נשתמע הלאה..".

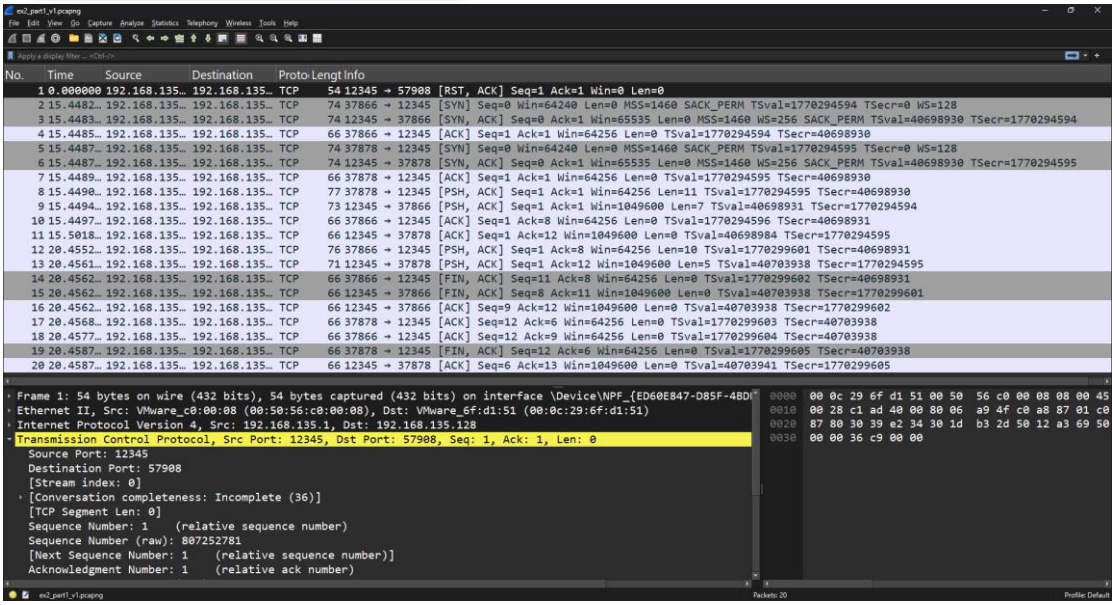
ובכך סיימנו לנתח גם את המקרה בו הודעה נשלחת לאחר שהשרת עושה קלוז קודם לרגע בו היא מגיעה.

סעיף 2:

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

גרסה v1:

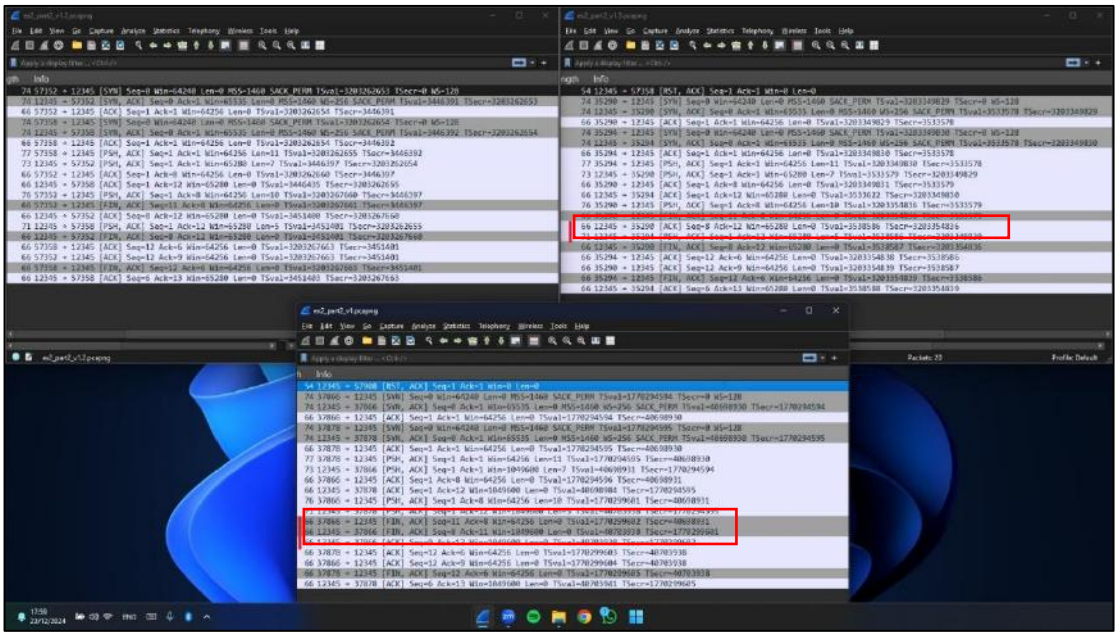




הלקוח מקבל את הIP והפורט מהארגומנטים לתוכנית, לאחר מכן הוא מתחבר בשני חיבורים נפרדים בעזרת שני סוקטים נפרדים לשרת (שורות 4 עד 9). הלקוח תחילה שולח הודעה דרך החיבור השני, מחכה 5 שניות ושלח הודעה אחרת דרך החיבור הראשון. לבסוף, הלקוח מקבל את שתי ההודעות מהסוקט בסדר הפוך לסדר בו התבצעה השליחה, ולאחר מכן סוגר את החיבורים. צד שני, השרת מאזין לכל החיבורים כיוון שהוא בחר בכתובת 0.0.0.0 ומקבל פורט כקלט. הוא מרים את השרת לאוויר כאשר הוא מוכן שיהיו ברגע נתון לכל היותר חיבור 1 בהמתנה (ליסטן שווה 0). הוא עושה accept לשני חיבורים (מדפיס מי הם), מקצר את ההודעה השנייה ל-7 תווים הראשונים שנקלטו ושולח אותה לחיבור הראשון, ואת ההודעה הראשונה ל-5 תווים הראשונים שנקלטו ושולח אותה לחיבור השני. לבסוף, יוזם ניתוק רק עם החיבור הראשון שהוא פתח.

נשים לב שבפועל בכריש קרה משהו מעניין. לאחר האתחול עד שורה 9, נשלחת ההודעה מהסוקט שנוצר שני בלקוח, והשרת לפני שהוא מחזיר עליה ack לאותו סוקט שני, קודם שולח את ההודעה המקוצרת (7 בתים), לחיבור הראשון (שורה 11). יתרה מזאת, הוא מקבל מהחיבור הראשון ack על ההודעה המקוצרת (שורה 12), ורק אז מחזיר ack על עצם זה שהוא קיבל את ההודעה המקורית מהחיבור השני. לאחר מכן השרת מקבל את ההודעה מהסוקט שנוצר ראשון בקליינט, ובאופן דומה לפני שהוא מחזיר עליה ack, הוא מעביר את 5 הבתים הראשונים מההודעה לחיבור סוקט השני. בשלב הזה מגיעה הודעת fin מהחיבור הראשון, אשר נענית בfin מצד השרת, ועל הדרך הוא גם נותן לו ack (רק בשלב הזה) על החבילה שנשלחה על ידו, ומיד הודעה נוספת של ack על הfin עם הפאנטום ביט. החיבור השני מחזיר ack רק עכשיו על ההודעה שהוא קיבל (5 הבתים), החיבור הראשון מחזיר ack על הודעות הfin שקיבל מהשרת (שוב פאנטום ביט), ולבסוף השרת מקבל fin מהחיבור השני ומשיב על כך בack (הוא לא שולח לו fin שכן בקוד הfin ששולח השרת הינו עבור החיבור הראשון, שכבר נסגר מיוזמתו קודם לכן..).

לאחר הרצה מס' פעמים לא היו שינויים משמעותיים –

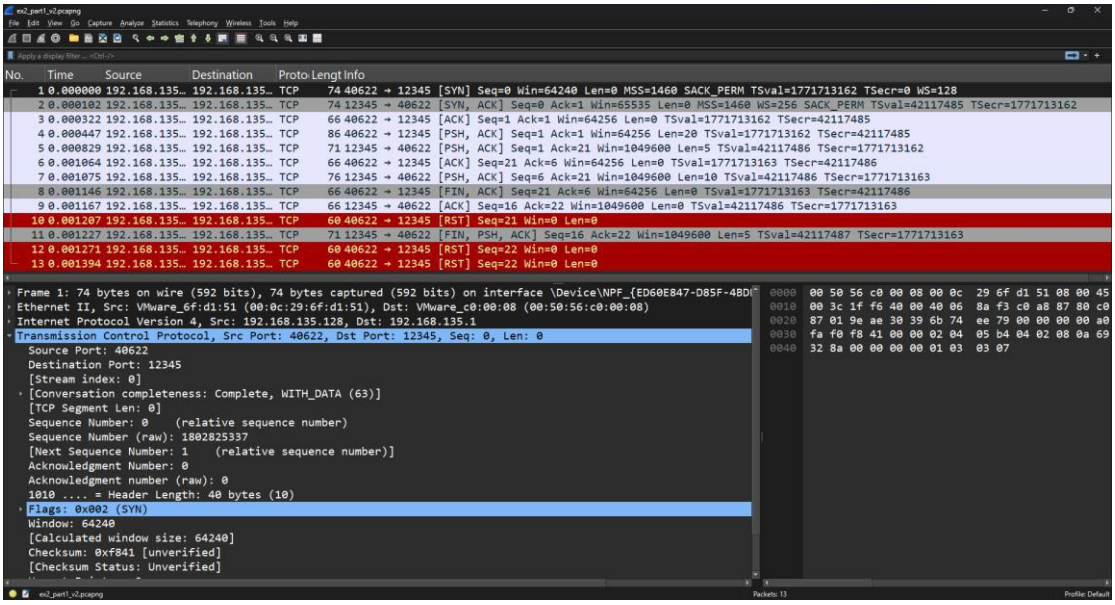


הדבר היחיד הוא שבאחת ההרצות (העליונה בימין) לאחר השליחה האחרונה לשרת מהקליינט הוא מיד שולח את הFIN, ורק לאחר מכן הוא מקבל תשובות מהשרת, בעוד שבהרצה אחרת (התחתונה), אנו רואים שהקליינט שולח את הבקשה **מקבל** תשובה מהשרת ורק אז הם שולחים FIN הדדי.

<u>גרסה v2</u>:

```python
import socket,sys

TCP_IP = '0.0.0.0'
TCP_PORT = int(sys.argv[1])
BUFFER_SIZE = 5

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((TCP_IP, TCP_PORT))
s.listen(1)

while True:
    conn, addr = s.accept()
    print('New connection from:', addr)
    while True:
        data = conn.recv(BUFFER_SIZE)
        if not data: break
        print("received:", data)
        conn.send(data.upper())
    conn.close()
```

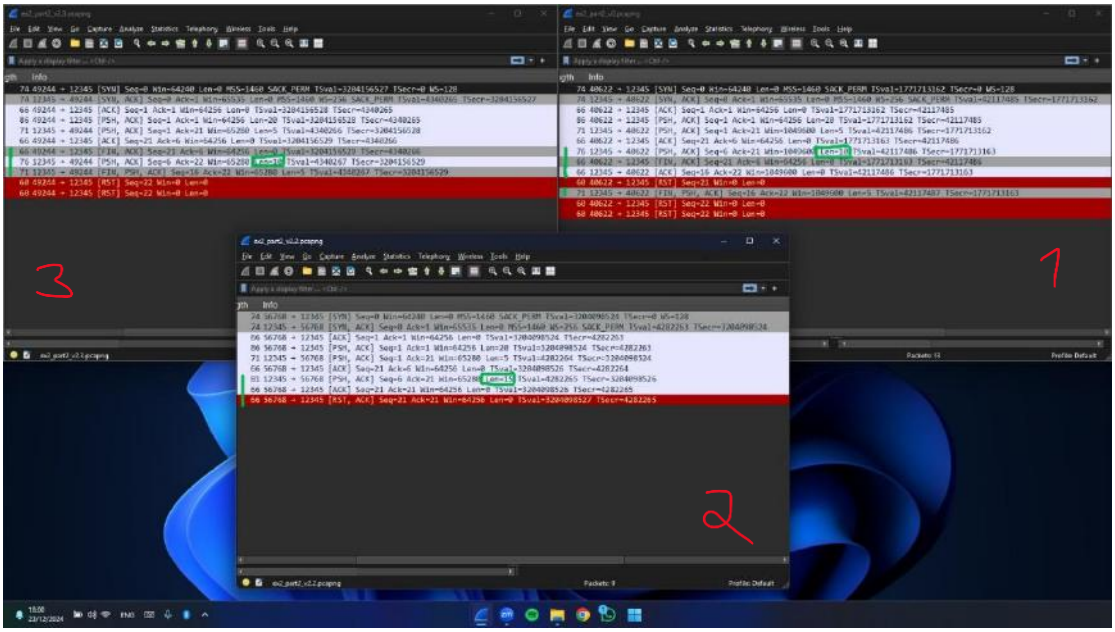
```python
import socket,sys

TCP_IP = sys.argv[1]
TCP_PORT = int(sys.argv[2])
BUFFER_SIZE = 1024
MESSAGE = b'World! Hello, World!'

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((TCP_IP, TCP_PORT))
s.send(MESSAGE)
data = s.recv(BUFFER_SIZE)
s.close()

print("received data:", data)
```



כמו בתוכנית הקודמת, הלקוח מקבל את הIP והפורט מהארגומנטים לתוכנית, לאחר מכן הוא מתחבר לשרת ושולח לו את ההודעה המצוינת. הוא עושה recv מהשרת, ולאחר שקיבל את ההודעה שהגיעה חזרה מהשרת הוא מדפיס אותה. מצד שני, כמו בתוכנית הקודמת, השרת מאזין לכל החיבורים כיוון שהוא בחר בכתובת 0.0.0.0 ומקבל פורט כקלט. הוא פותח חיבור, מקבל את ההודעה בצאנקים של בתים 5 (כגודל הבאפר), עד שבתים מפסיקים להגיע, מדפיס כל צאנק, ושולח אותו חזרה ללקוח באופן של upper. לבסוף, הוא סוגר את החיבור.

כמו התפיסה האחרונה, ניתן לראות את לחיצת הידיים בין הלקוח לשרת (שורות 4 5 6). אך בשונה מהתפיסה האחרונה, השרת קולט את המידע שמגיע אליו מחיבור אחד בחתיכות של 5 תווים. לכן רואים בכריש את הצאנק הראשון מגיע ומוחזר מהשרת (בupper כמובן), ולפני שאנו רואים את המשך קבלת הצאנקים בשרת, אנו רואים את הack שהלקוח החזיר על הצאנק הראשון. השרת מאחד את 2 הצאנקים הבאים (כמובן זה התנהגות שנובעת מאלגו' nagel) ושולח אותם יחדיו ללקוח, כאשר האחרון מחזיר עליהם ack. עם זאת, מכיוון שיש רק recv אחד בלקוח, הוא ממשיך בשלו ולאחר קבלת הצאנק הראשון הוא שולח fin כדי לסיים את ההתקשרות. השרת מקבל את fin ומחזיר עליה ack, אולם מעתה והילך שאר הצאנקים שהשרת שולח זוכים למענה של rst כיוון שאין אף אחד בצד השני ששומע וזמין לקבל. כנל לfin שהשרת שולח – הכל מקבל rst, שכן הלקוח ניתק בצד השני...

לאחר הרצה מס' פעמים אלו היו השינויים שראינו –



שמנו לב שאופן חלוקת המידע לפי השרת עונה ללקוח עשוי להשתנות מהרצה להרצה (גם הסדר של "מתי הלקוח שולח fin, לפי קבלת החבילה או אחרי..", זאת כמובן ראינו כבר בדוג' בגרסה הקודמת). בפעם השנייה שהרצנו ראינו שהשרת עשה "סטאק" ליותר בתים (15 במקום 10) ותכנן לשלוח אותם ביחד. שוב זה נובע מאלגו' נייגל. תחילה הוא שולח חבילה – צאנק יחיד, כיוון שאין משהו on the fly. ברגע שהוא מקבל עוד 5 בתים לשלוח הוא לא ממהר לשלוח אותם (שכן זה מעט בתים ועלות ההאדרים גבוהה), ולכן הוא ממתין לעוד בתים ע"מ שיהיה משתלם לשלם "דמי משלוח". בהרצה אחת הוא המתין ל-10, ואילו באחת ל-15. בכל אחד מהמקרים כבר לא היה מי שיאזין בצד השני (שכן תוכנית הלקוח כבר סיימה בשלב הזה, ולכן קיבלנו rstים בכל מצב). הריצה ה-3 זהה לראשונה, רק ששם הלקוח עשה FIN בטרם קיבל חבילה מהשרת (מה שתיארנו כבר בדוג' הקודמת).

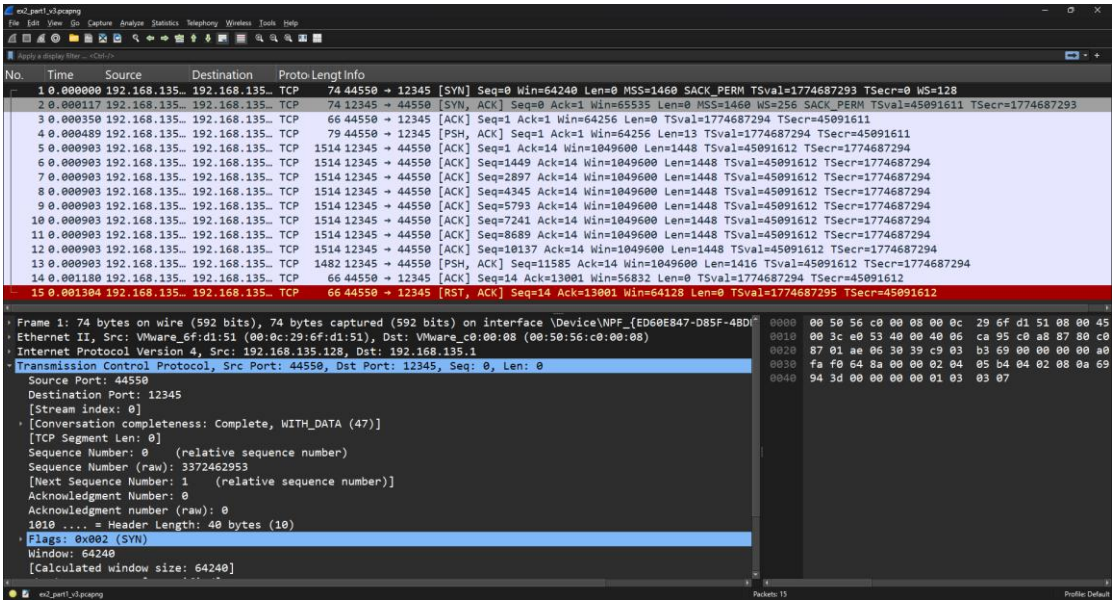
<u>גרסה v3</u>:

server.py

```python
import socket,sys

TCP_IP = '0.0.0.0'
TCP_PORT = int(sys.argv[1])
BUFFER_SIZE = 1024

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((TCP_IP, TCP_PORT))
s.listen(1)

while True:
    conn, addr = s.accept()
    print('New connection from:', addr)
    while True:
        data = conn.recv(BUFFER_SIZE)
        if not data: break
        print("received:", data)
        conn.send(data.upper()*1000)
    conn.close()
```

client.py > ...

```python
import socket,sys

TCP_IP = sys.argv[1]
TCP_PORT = int(sys.argv[2])
BUFFER_SIZE = 1024
MESSAGE = b'Hello, World!'

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((TCP_IP, TCP_PORT))
s.send(MESSAGE)
data = s.recv(BUFFER_SIZE)
print("received data:", data)
data = s.recv(BUFFER_SIZE)
print("received data:", data)
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 ההודעה חזרה מהשרת הוא מושך ומדפיס את ה1024 בתים הראשונים ולאחר מכן שוב הוא מנסה למשוך עוד 1024 בתים. כמו כן בצד שני, כמו בתוכנית הקודמת, השרת מאזין לכל החיבורים כיוון שהוא בחר בכתובת 0.0.0.0 ומקבל פורט כקלט. הוא פותח חיבור, מקבל את 1024 הבתים הראשונים ושולח אותם חזרה ללקוח באותיות גדולות בהכפלה של 1000 פעמים הקלט. לבסוף, סוגר את החיבור.

כמו בתפיסה האחרונה, ניתן לראות את לחיצת הידיים בין הלקוח לשרת. מיד לאחר מכן את 13 הבתים שהלקוח שולח, ואז השרת קולט את כל המידע שמגיע אליו, ושולח אותו באותיות גדולות כפול 1000 פעמים, לכן 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 הבתים הראשונים) וכן את השנייה (אלו שבאים אחרי), מחזיר ack באופן מאוחד על 13000 הבתים שהגיעו, וסוגר את החיבור. אבל יש עוד חבילות בבאפר שלא נקראו ע"י האפליקציה כאשר היא נסגרת, לכן נשלחת הודעת rst לשרת שידע שהלקוח ניתק מבלי לקרוא עד הסוף. לא ראינו שינויים בהרצה מס' פעמים של הקוד.

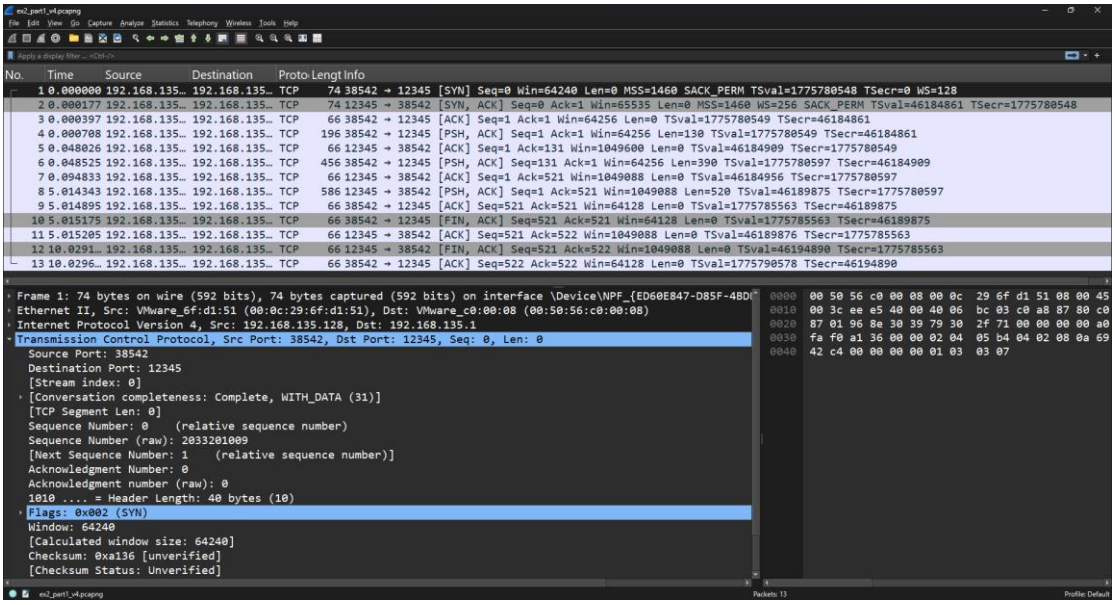
<u>גרסה v4</u>:

server.py > ...

```python
import socket,sys,time

TCP_IP = '0.0.0.0'
TCP_PORT = int(sys.argv[1])
BUFFER_SIZE = 1024

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((TCP_IP, TCP_PORT))
s.listen(1)

while True:
    conn, addr = s.accept()
    print('New connection from:', addr)
    while True:
        time.sleep(5)
        data = conn.recv(BUFFER_SIZE)
        if not data: break
        print("received:", data)
        conn.send(data.upper())
    conn.close()
```

client.py > ...

```python
import socket,sys

TCP_IP = sys.argv[1]
TCP_PORT = int(sys.argv[2])
BUFFER_SIZE = 1024
MESSAGE = b'Hello, World!'

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((TCP_IP, TCP_PORT))
s.send(MESSAGE*10)
s.send(MESSAGE*10)
s.send(MESSAGE*10)
s.send(MESSAGE*10)
data = s.recv(BUFFER_SIZE)
s.close()

print("received data:", data)
```



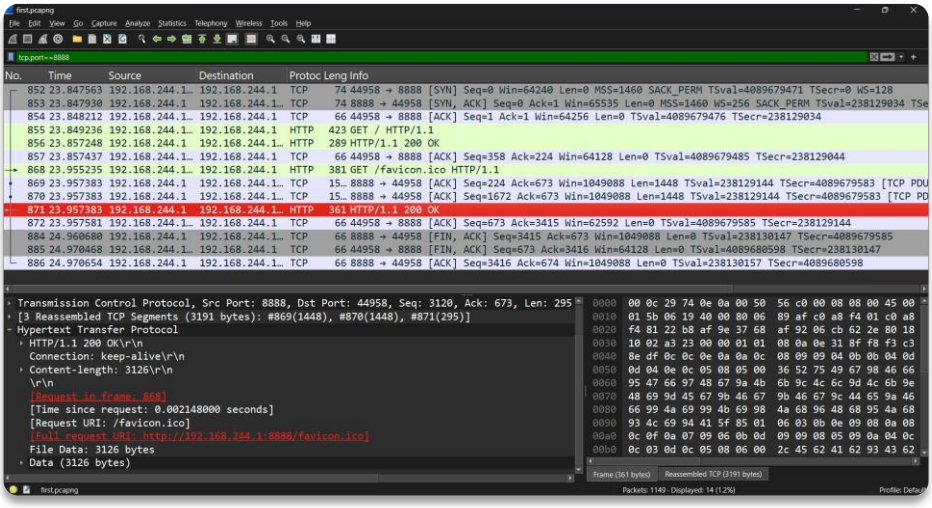
כרגיל הלקוח מקבל את הIP והפורט מהארגומנטים לתוכנית, לאחר מכן הוא מתחבר לשרת ושולח לו 4 הודעות שכל הודעה היא 10 פעמים "Hello, World", השוני הפעם מהתוכנית הקודמת, הוא שלאחר שקיבל את ההודעה חזרה מהשרת הוא מדפיס את ה1024 בתים הראשונים. בצד שני, כרגיל השרת מאזין לכל החיבורים כיוון שהוא בחר בכתובת 0.0.0.0 ומקבל פורט כקלט. הוא פותח חיבור מחכה 5 שניות, מקבל את 1024 הבתים הראשונים ושולח אותם חזרה ללקוח באותיות גדולות. הוא חוזר על התהליך עד שהוא מפסיק לקבל מידע מהלקוח ולבסוף סוגר את החיבור.

בכריש אנו רואים תחילה את לחיצת הידיים בין הלקוח לשרת. אך בשונה מהתפיסה האחרונה, השרת מחכה 5 שניות ולכן התגובה שלו מתעכבת. ניתן לראות כי השליחה (של הלקוח) והתגובה (של השרת) קורית פה 2 פעמים (כנראה 3 השליחות האחרונות אוחדו בעת שליחתן). בכל פעם השרת (שנמצא בtimeout באפליקציה ולכן לא קורא מהבאפר), מחזיר ack שהוא קיבל את ההודעה. השרת שולח את כל ה520 בתים בחזרה בפעם אחת (מאוחדת) באותיות גדולות, ומקבל ack עליו מהשרת. מכאן והילך טקס הפרידה הסטנדרטי של fin-ack-fin-ack בין השרת ללקוח, כאשר כל אחד מעלה את הפאנטום ביט, ונפרדים לשלום. כאן לא הייתה בעיה של rst בשל העובדה שכל המידע נקרא מהבאפר בטרם האפליקציה סיימה את הריצה. לא ראינו שינויים בהרצה מס' פעמים של הקוד.

חלק ב' –

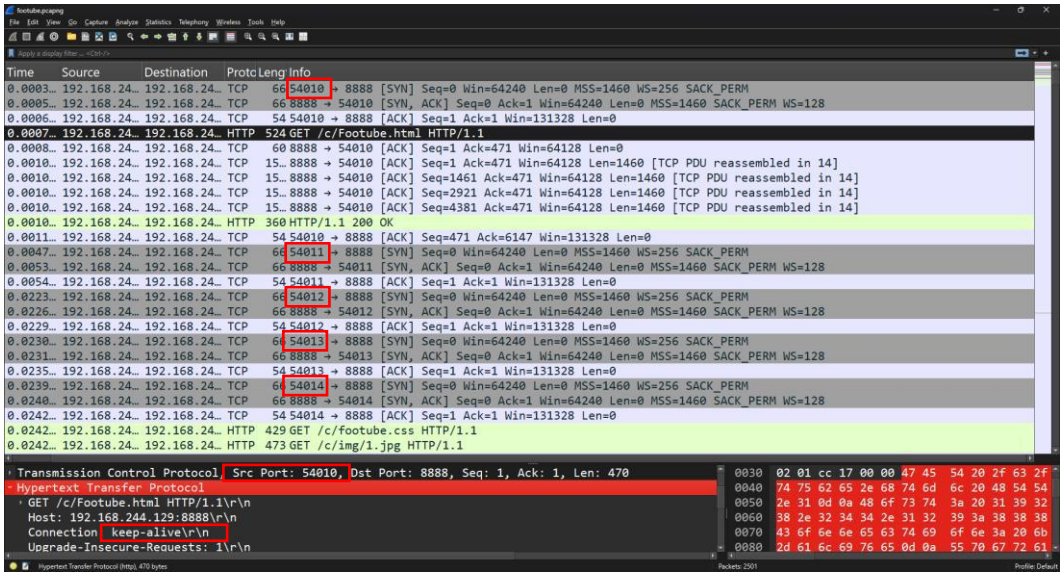
נסיים חלק זה של התרגיל בהצגת הכריש בזמן ריצת קוד השרת שלנו, כאשר האחרון מקבל בקשות מהדפדפן (אשר רץ בVM).

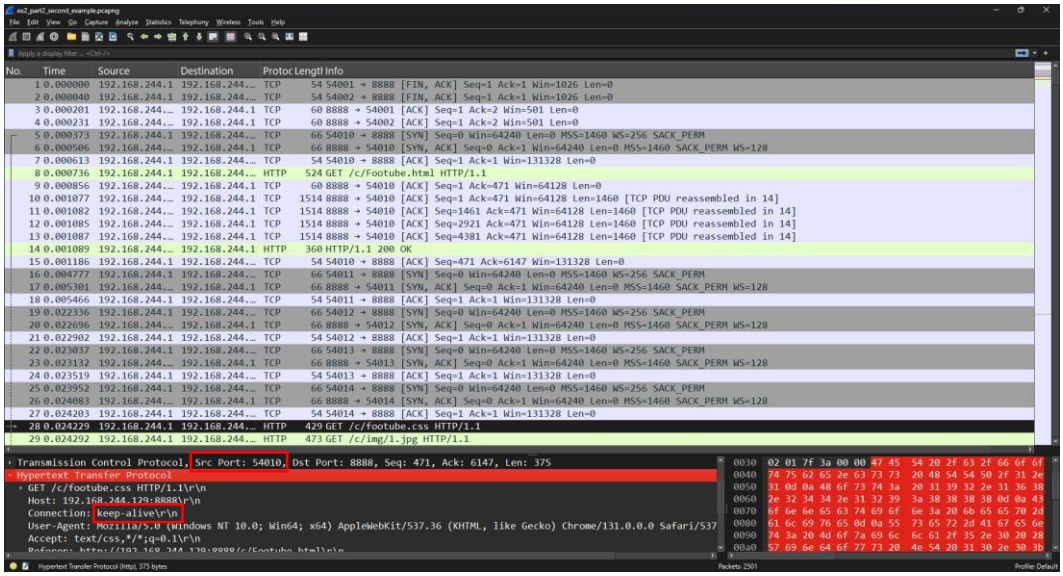
http://192.168.244.1:8888/

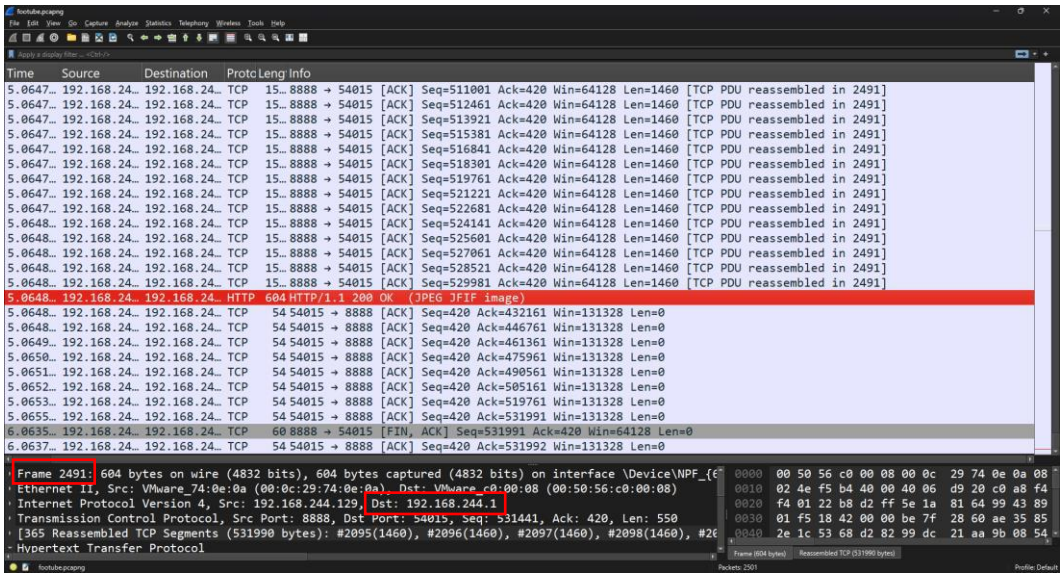


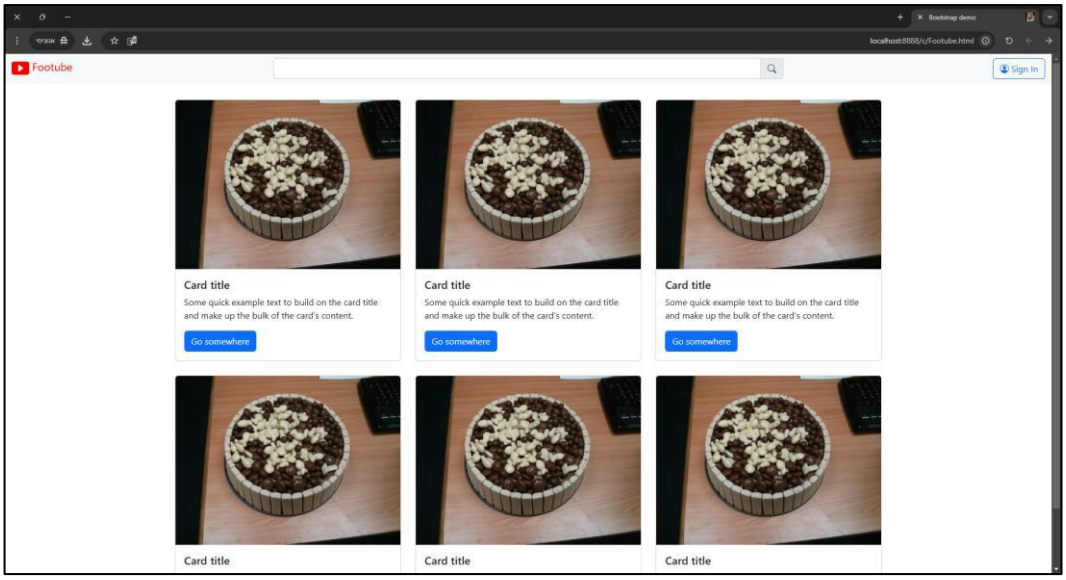
הכתובת פונה ומבקשת מהשרת את דף הבית, והוא מחזיר בתמורה את index.html. ניתן לראות בתמונה שהדפדפן עושה מס' פניות מאותו פורט (44958), כאשר תחילה הוא שולח את הבקשה של הקובץ, ומקבל אליו חזרה את ה 200 ok, מיד לאחר שהם מחליפים ack ביניהם אודות מעבר הקובץ באופן תקין, הדפדפן שולח בקשת get נוספת עבור הקובץ favicon.ico הלוא הוא האייקון של "האתר" שלנו שאנו רואים בקצה הכרטיסייה בדפדפן, גם על בקשה זו מתקבלת הודעת 200 ok (בתמונה רואים את הקובץ עליו הוחזר ok למטה בbody), והתקשורת נסגרת באופן הדדי בfinים משני הצדדים. ניתן לראות שנשלח המון מידע עם הודעת הget מדפדפן, כמו סוג הדפדפן, הגרסה שלו, סטטוס connection, עדיפות ועוד. מעבר לget, ההחזרה של המידע מהשרת – 200 ok, ואז הבקשה לאייקון אין משהו מיוחד, ואין בקשה נוספת מאותו פורט.

http://192.168.244.129:8888/c/Footube.html



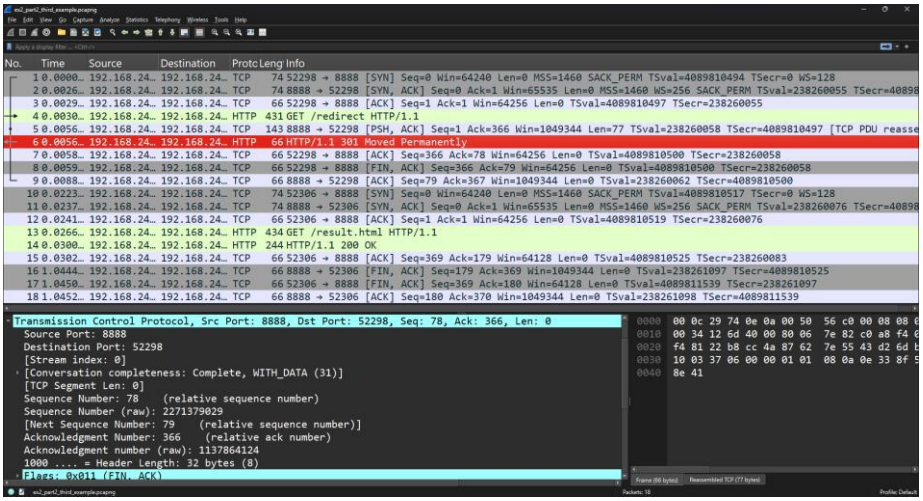


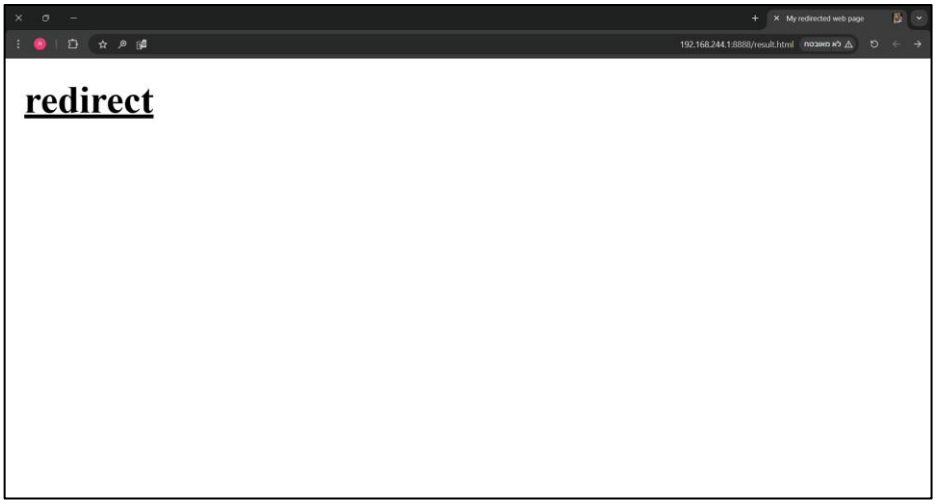




כעת מבקש הדפדפן את קובץ c/Footube.html/ אשר מכיל בתוכו מס' קבצים (תמונות) נוספים. ניתן לראות שהפעם הוא פותח מס' חיבורים (כ-חמישה) דרך כמה פורטים (תמונה ראשונה), אבל בפועל בשלב הראשון רק דרך החיבור הראשון עוברת התקשורת של הבקשה מהשרת. לאחר שהוא מבקש את הדף הראשון (זה שביקשנו בכתובת), הוא מקבל אותו ומתחיל ברצף בקשות כבדות מאוד של התמונות המופיעות באותו קובץ. נשים לב שהתמונה מועברת בהמון צאנקים, אך לאחר כמה זמן הוא מנסה לבקש דרך אחד הפורטים האחרים שהוא פתח תמונה נוספת שונה מזו שביקש דרך הפורט הראשי, אך למרות זאת עדיין התקשרות ממשיכה להגיע דרכו. זה גרם המקום להמחיש כדרוש במטלה כיצד השרת שלנו תומך בkeep-alive שכן אנו רואים שבheader של הבקשה מהדפדפן הוא מבקש להשאיר את החיבור פתוח, ובאמת לאחר שדף הhtml מגיע, הבקשה הבאה מתבצעת באותו חיבור על אותו הפורט וגם היא מציינת – 'keep-alive'. לאחר שמס' תמונות הגיעו מהפורט הראשי – הוא נסגר, ואז אנו רואים כיצד התמונה שנתבקשה בפורט השונה (54636) מתחילה להגיע גם היא. אותו דפוס פעולה חוזר על עצמו, כאשר מס' תמונות מגיעות דרך כמה פורטים שונים, אך אין ערבוב והכל מתבצע בצאנקים, דהיינו רק לאחר שפורט קיבל את התמונות שלו ונסגר (בfin כמובן), אנחנו רואים שהתמונות שביקש פורט אחר מתחילות להגיע (זה וודאי נובע מהמחסור במקביליות באופן שבו בנינו את השרת, ובשל כך שאר הפורטים תקועים בlisten כאשר האחד מקבל שירותים מהשרת). התמונות שהן די גדולות מועברות בהמון צאנקים (כנראה בגלל שאנחנו עובדים מחשב-VM), ואחת לכמה זמן אנו רואים במהלך התקשורת ack שחוזר עליהן מהלקוח לשרת, כאשר בסיום העברה של כל קובץ תמונה נראה שמגיע get "מאסף". שוב בקשות הget מכילות באותו מידע יבש על הדפדפן ושאר התשתיות. כך זה מתבצע באופן איטרטיבי עד שבסופו של דבר הפורט האחרון שביקש את התמונה האחרונה נסגר גם הוא, והדפדפן מציג את הדף בשלמותו (תמונה חמישית). ניתן לראות בקובץ הpcap המלא את כלל ההסברים שכתבנו כאן באופן ממשי (אנחנו גבוליים עם גודל הזיפ לכן מצמצמים בתמונות מחילה). יש לציין שבתחתית כל חבילה המכילה קובץ PNG ישנם נתונים בהקשר הזה אשר אנו מניחים שעוזרים לפענח את התמונה (לפי הטייפ שלה, גודל וכדו').

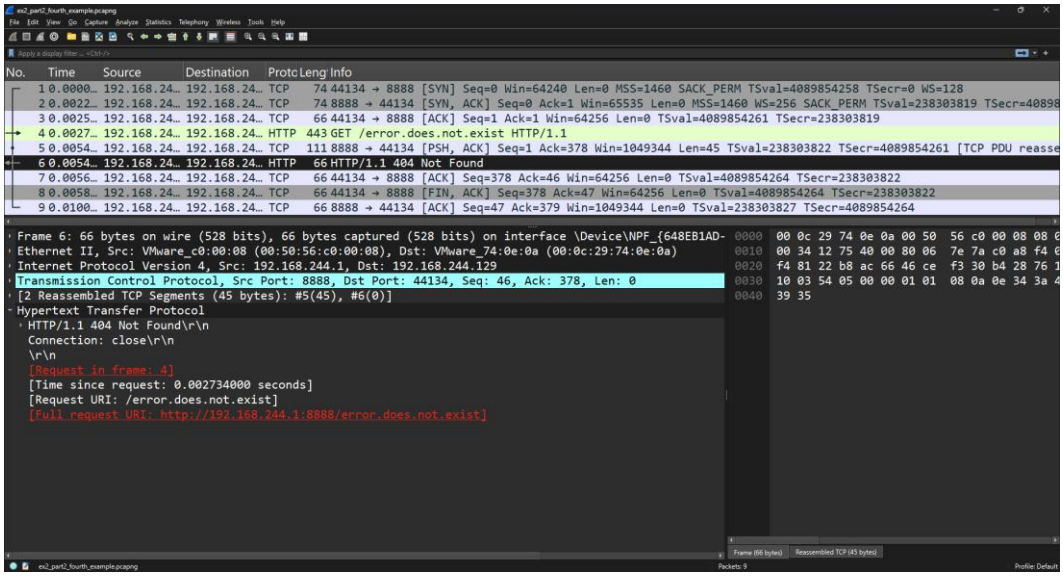
http://192.168.244.1:8888/redirect

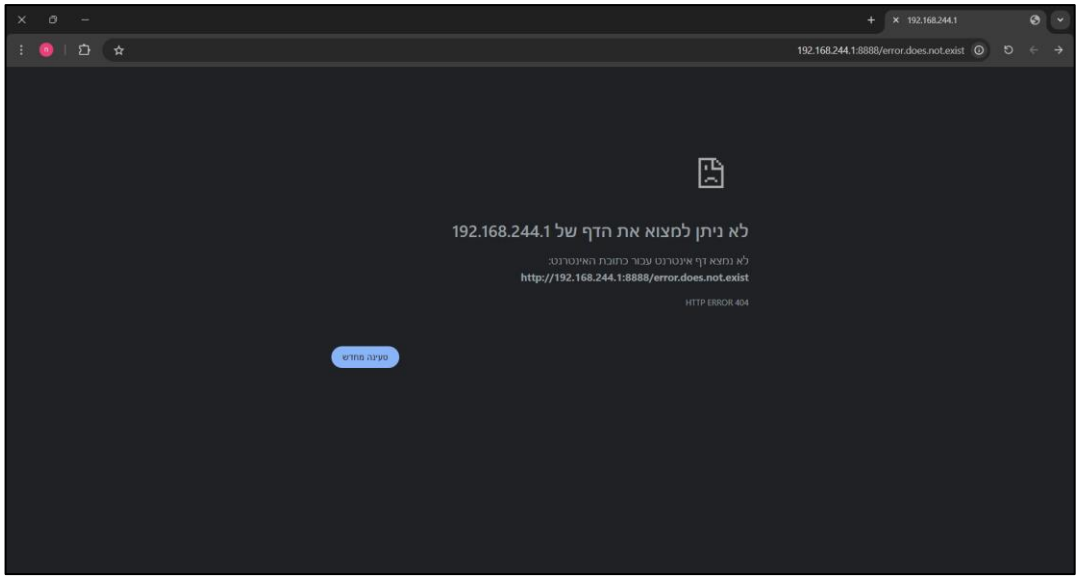




הדפדפן פונה כעת אל הנתיב redirect/ של השרת שלנו. ניתן לראות שהפעם הוא פותח תחילה חיבור יחיד של פורט אחד, אשר דרכו הוא שולח את הפניה (שורה צהובה עליונה). השרת שולח לו תשובה חזרה של 'moved permanently', מצרף לתשובה שלו את ההפניה/הכתובת של מקום החדש, והחיבור נסגר (תמונה ראשונה). מיד לאחר מכן מוקם חיבור **חדש** מצד הדפדפן, אשר דרכו מתבצעת הבקשה לאותו location-מיקום חדש שהחזיר השרת לדפדפן בבקשה הקודמת (שורה צהובה שלישית). השרת מקבל את הבקשה השנייה, מחזיר את הקובץ שביקש בדפדפן – result.html, ואחר כך שוב פעם אנו רואים את הסגירה של התקשורת בfin שמגיע משני הכיוונים. ברמת המטא-דאטא אין שוני יוצא דופן משראינו כבר.

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

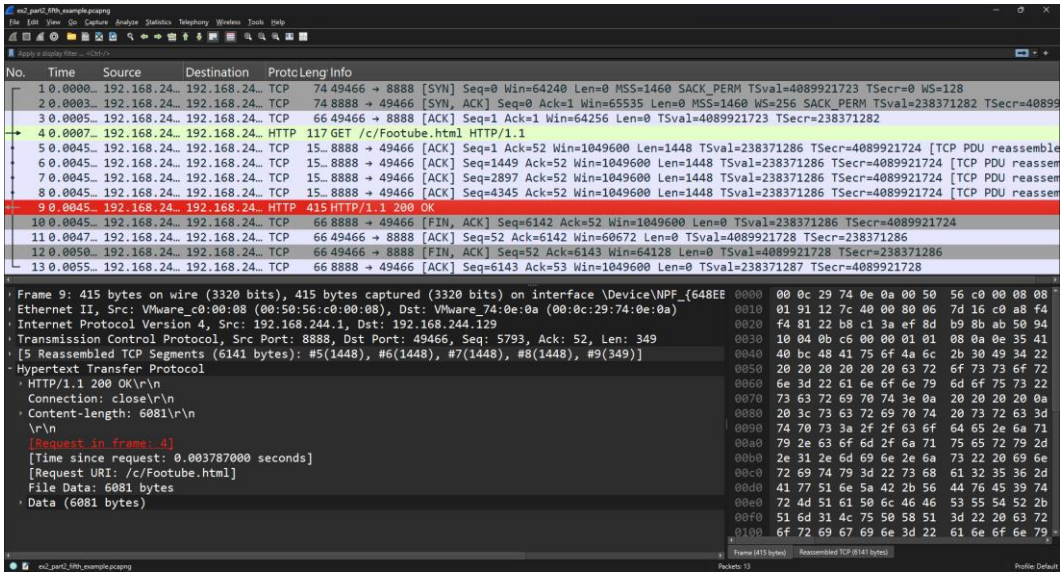




אנו רואים כעת ניסיון של הדפדפן לפנות לכתובת אשר לא קיימת מבחינת השרת שלנו. אנו רואים שנפתח שוב חיבור יחיד של פורט אחד, אשר דרכו נשלחת הבקשה (השורה בצהוב). השרת אשר שולח ack על כך שקיבל את הבקשה, ומיד מחזיר לדפדפן תשובה חזרה של שנותנת אינדיקציה לדפדפן שהדף לא נמצא – '404 not found' (תמונה שנייה). לאחר שהתשובה מוחזרת, התקשורת נסגרת בfin. ברמת המטא-דאטא שוב אין שוני יוצא דופן משראינו כבר.

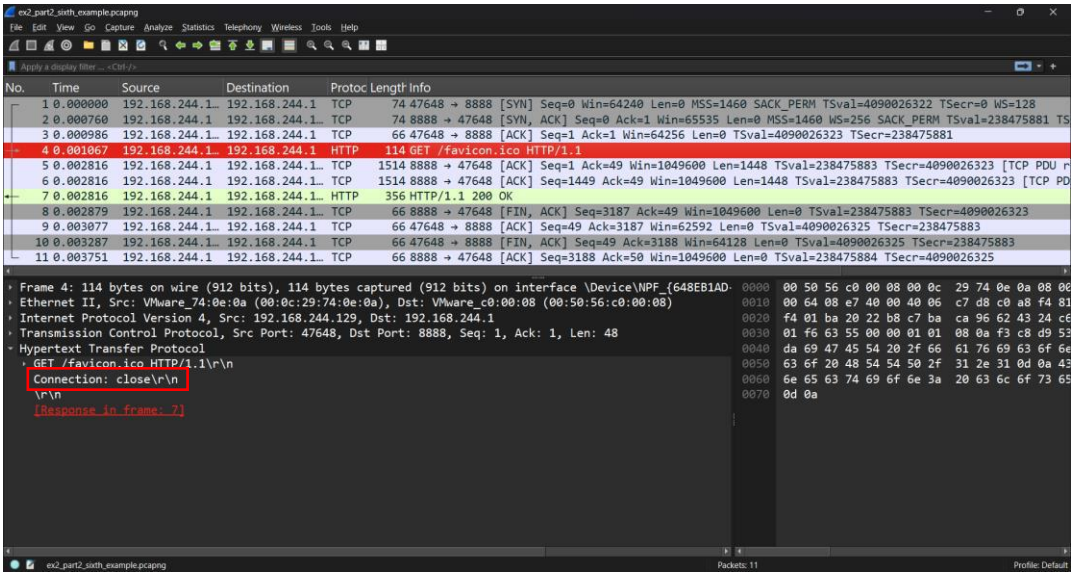
נסיים בכך שנראה מס' בקשות בכריש אשר התבצעו אל השרת אך הפעם לא מהדפדפן אלא מקוד הקליינט שאנחנו כתבנו –

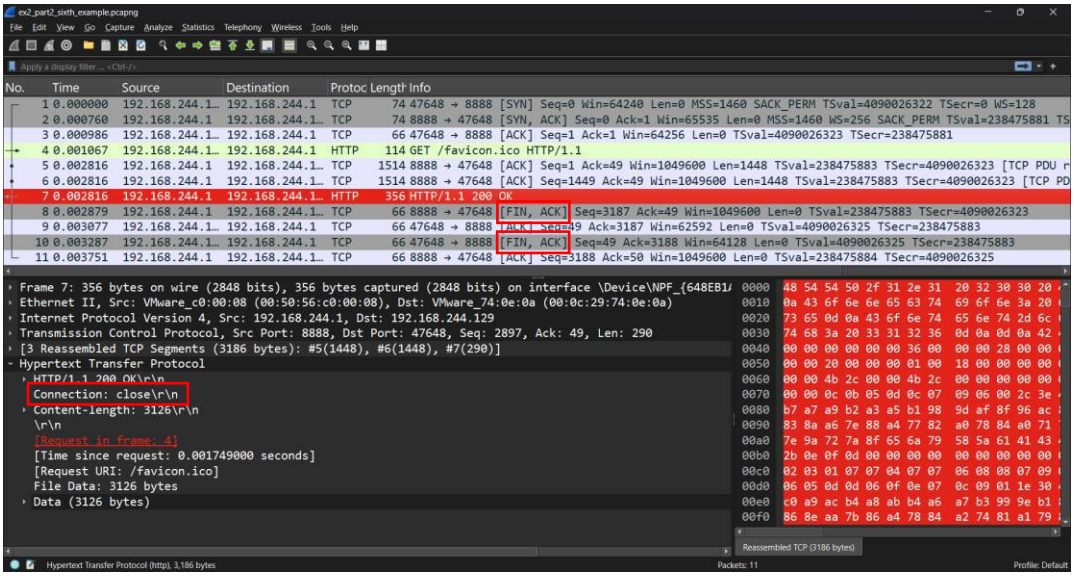
/c/Footube.html



הלקוח מבקש את הקובץ /c/Footube.html אשר מכיל "עמוד אינטרנטי עם שלל תמונות וקבצים המוכלים בו, אולם כאן אנו רואים פורט יחיד אשר נפתח ע"י הלקוח ופונה לשרת, ויתרה מזאת לאחר שמתקבל הקובץ אנו לא רואים שרשרת של בקשות בדומה למה שראינו כאשר הפניה הגיעה מהדפדפן, כיוון שפה הלקוח רק מבקש את הקובץ ולא מנסה להציגו, אולם לו היה מנסה היה נתקל בכל מיני קבצים ומשאבים אשר אין, ובאמת היינו רואים שרשרת בקשות המנסה לייבא את כל המשאבים המוחזקים באופן כזה או אחר אצל השרת, אשר אלו דרושים באופן חיוני על מנת להציג את הקובץ במלואו. לאחר שהקובץ מגיע התקשורת נסגרת בfin הדדי. יש לציין שניתן לראות שהheader הרבה יותר דליל מזה שראינו בבקשות שהגיעו מהדפדפן, שכן שאנחנו בקוד הקליינט כתבנו בפניה רק את – סוג הפניה, שוב הקובץ המבוקש, הפרוטוקול בו אנו משתמשים, וכן סטטוס הconnection.

/favicon.ico





דוג' פשוטה ואחרונה בה הלקוח מבקש את הקובץ של האייקון של השרת (התמונה הקטנה המופיעה בראש הכרטיסייה), אנו רואים שהשרת מחזיר את התוכן ב2 חבילות שונות (מפאת גודלו), ומאסף זאת בהודעת 200 ok. נמחיש פה את ההתנהגות מצד השרת כאשר זה מקבל סטאטוס של close בקונקשן. נשים לב שבהודעת הלקוח אכן הערך בheader הינו 'close', ואנו ממש רואים כי לאחר שמעבר ההודעה נשלם, בהודעת ה-OK מהשרת הוא משיב בחיווי של 'close' גם מצידו, ומיד לאחר מכן מתבצעת הסגירה של התקשורת בתצורת FIN הדדי.