

מטלה מס' 2 – אילאי בן יהושע ונועם ליבוביץ

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

```
tcp_server.py X
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 X
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 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	Proto	Length	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 –

The image shows a Wireshark packet capture of a TCP connection. The top pane displays a list of packets, with packet 1 highlighted. The middle pane shows the details of packet 1, which is a SYN packet from 192.168.135.1 to 192.168.135.128. The bottom pane shows the raw data of the packet, which is a 74-byte frame. A red arrow points to the 'Flags: 0x0002 (SYN)' field in the details pane.

Packet 1: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface \Device\NPF{F050E847...} Ethernet II, Src: VMware_6f:d1:51 (00:0c:29:6f:d1:51), Dst: VMware_c0:00:08 (08:50:56:c0:00:08) Internet Protocol Version 4, Src: 192.168.135.128, Dst: 192.168.135.1 Transmission Control Protocol, Src Port: 52970, Dst Port: 12345, Seq: 0, Len: 0

Destination Port: 12345

[Stream index: 0]

[Conversation completeness: Complete, WITH_DATA (31)]

[TCP Segment Len: 0]

Sequence Number: 0 (relative sequence number)

Sequence Number (raw): 1035627461

[Next Sequence Number: 1 (relative sequence number)]

Acknowledgment Number: 0

Acknowledgment number (raw): 0

1010 ... = Header length: 40 bytes (10)

Flags: 0x0002 (SYN)

Window: 64240

[Calculated window size: 64240]

Checksum: 0xff73 [unverified]

[Checksum Status: Unverified]

Urgent Pointer: 0

Options: (20 bytes), Maximum segment size, SACK permitted, Timestamps, No-Operation (NOP), Window s...

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 החס, 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 scale
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, המss שלי הוא 1460b".

חבילה מס' 2 –

The image shows a Wireshark packet capture of a TCP SYN-ACK packet. The packet list shows a packet from 192.168.13 to 192.168.13 on port 52970. The packet details pane shows the following information:

- Frame 2: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface \Device\NPF_{ED60E847...}
- Ethernet II, Src: VMware_c0:00:08 (00:50:56:c0:00:08), Dst: VMware_6f:d1:51 (00:0c:29:6f:d1:51)
- Internet Protocol Version 4, Src: 192.168.135.1, Dst: 192.168.135.128
- Transmission Control Protocol, Src Port: 12345, Dst Port: 52970, Seq: 0, Ack: 1, Len: 0
 - Source Port: 12345
 - Destination Port: 52970
 - [Stream index: 0]
 - [Conversation completeness: Complete, WITH_DATA (31)]
 - [TCP Segment Len: 0]
 - Sequence Number: 0 (relative sequence number)
 - Sequence Number (raw): 3834986605
 - Max Sequence Number: 1 (relative sequence number)
 - Acknowledgment Number: 1 (relative ack number)
 - Acknowledgment number (raw): 1035627462
 - 1010
 - Header Length: 40 bytes (10)
 - Flags: 0x012 (SYN, ACK)
 - Window: 65535
 - [Calculated window size: 65535]
 - Checksum: 0xcbbd [unverified]
 - [Checksum Status: Unverified]
 - Urgent Pointer: 0
 - Options: (20 bytes), Maximum segment size, No-Operation (NOP), Window scale, SACK permitted, Timestamps

אנו רואים פניה שמתבצעת הפעם בכיוון ההפוך מ-PORT 12345 (השרת), אשר מיודעת לפורט 52970 (הלקוח). הפניה (או יותר נכון 'תגובה') מצד השרת הינה מסוג syn, ack (גם פה ניתן לראות את שני הדגלים דלוקים), כלומר השרת ראשית מכיר בסוכרון של הלקוח איתו, אך בנוסף הוא מבקש להסתנכרן גם הוא עם הלקוח. זוהי למעשה תפקידה של החבילה הנ"ל עם דגלי הack, syn, 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 phantom-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 המss שלו – 1460b, כדי שזה ידע את הנתון כאשר הוא שולח אליו הודעות.

```
Options: (20 bytes), Maximum segment size, No-Operation (NOP), Window scale, SACK permitted, Timestamps
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 –

Wireshark packet capture showing a TCP ACK segment. The packet list shows a SYN segment followed by an ACK segment. The packet details pane shows the ACK segment's fields: Source Port: 52970, Destination Port: 12345, Sequence Number: 1, Acknowledgment Number: 1, and Window: 502. The packet bytes pane shows the raw data of the segment.

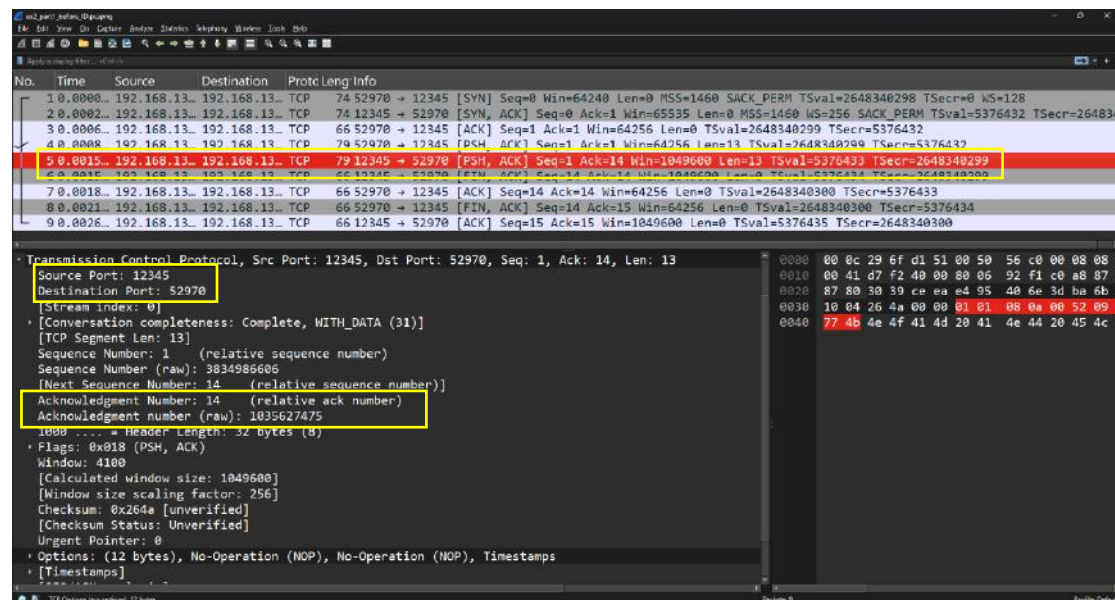
כעת ניתן לראות פניה נוספת מהלקוח 52970 (הלקוח), אל השרת. הפניה מצד השרת הינה מסוג ack (שוב הדגל דלוק ולמעשה מעתה והילך הוא יהיה דלוק עד סוף התקשורת, אז נפסיק לציין זאת...), דהיינו הלקוח גם הוא מכיר בהכרת השרת בסנכרון שלו, ולמעשה תפקידה של החבילה הזו מהווה את השלב השלישי והאחרון בטקס 'לחיצת הידיים המשולשת', 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 האקט ששולח הלקוח, והוא – '3834986606', באופן לא מפתיע זה 1 יותר מהseq שבחר השרת (שוב אנחנו מגדילים בגלל הפאנטום ביט של האקט). הseq לעומת זאת לא השתנה שכן לא נשלח עוד דאטא, וכתוצאה מכך שוב החל 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 –

Wireshark packet capture showing a TCP PSH segment. The packet list shows a SYN segment followed by a PSH segment. The packet details pane shows the PSH segment's fields: Source Port: 52970, Destination Port: 12345, Sequence Number: 1, Acknowledgment Number: 1, and Window: 502. The packet bytes pane shows the raw data of the segment.

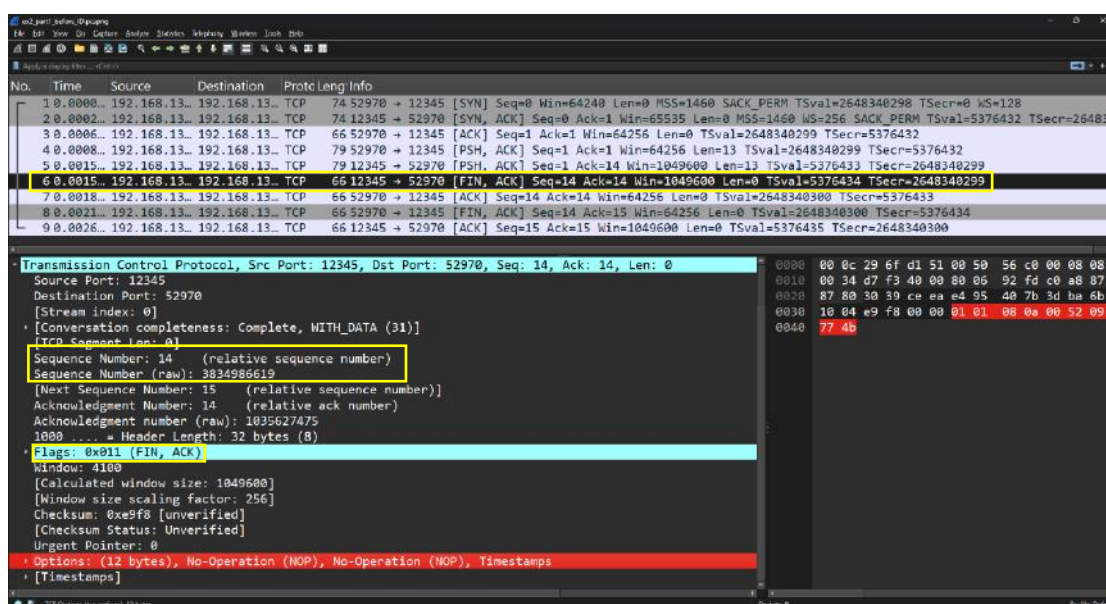
סוף כל סוף הגענו לחלק המעניין ('פסגת הקורס' לפי חלק מהשיטות), אנו רואים הודעה מהלקוח (לפי הפורט 52970 שראינו קודם שהוא הפורט של הלקוח), השולח 13 בתים (לפי ההנחה), שאנחנו יודעים שהם הבתים של המחרוזת 'Noam and Elay' (כנדרש במטלה), ובנוסף לדגל האק (שאמרנו שמעתה והילך תמיד יהיה דלוק), גם כן דלוק דגל הpush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

חבילה מס' 5 –



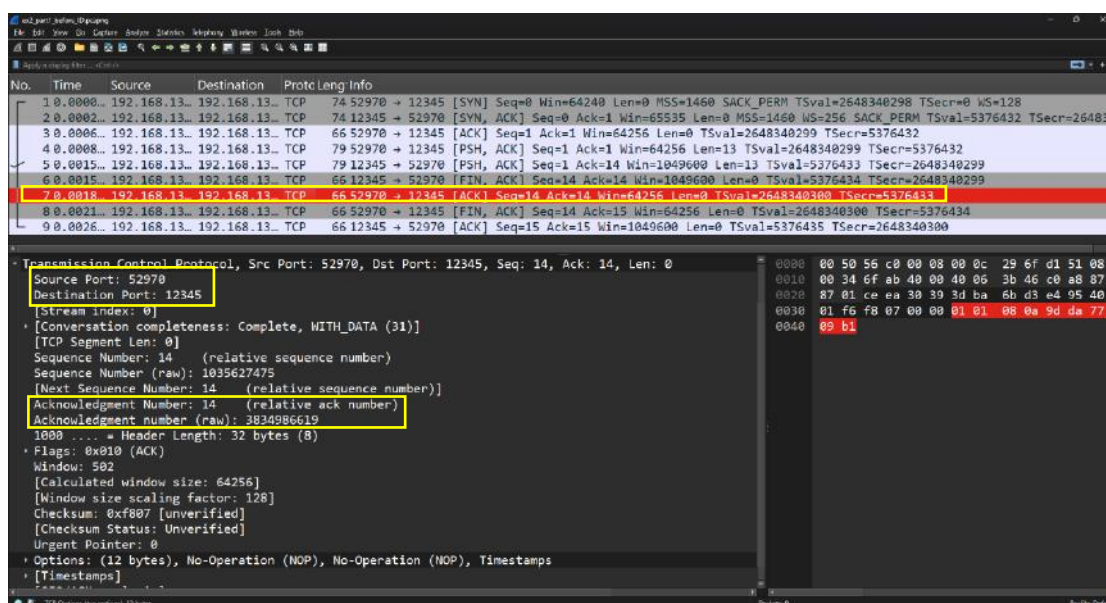
כעת נראה את תגובת השרת להודעה שקיבל מהלקוח בחבילה הקודמת שסקרנו. השרת ראשית מעדכן את המספר בשדה האק להיות 13 יותר משהיה – '1035627475' שכן עכשיו הוא אומר ללקוח – "שומע?! קיבלתי כבר עד בית מס' 1035627475 תן לי ממנו והלאה". אמנם seq לא משתנה כיוון שהוא טרם שלח דאטא משל עצמו ללקוח, אך כעת הוא מצרף להודעת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

חבילה מס' 6 –



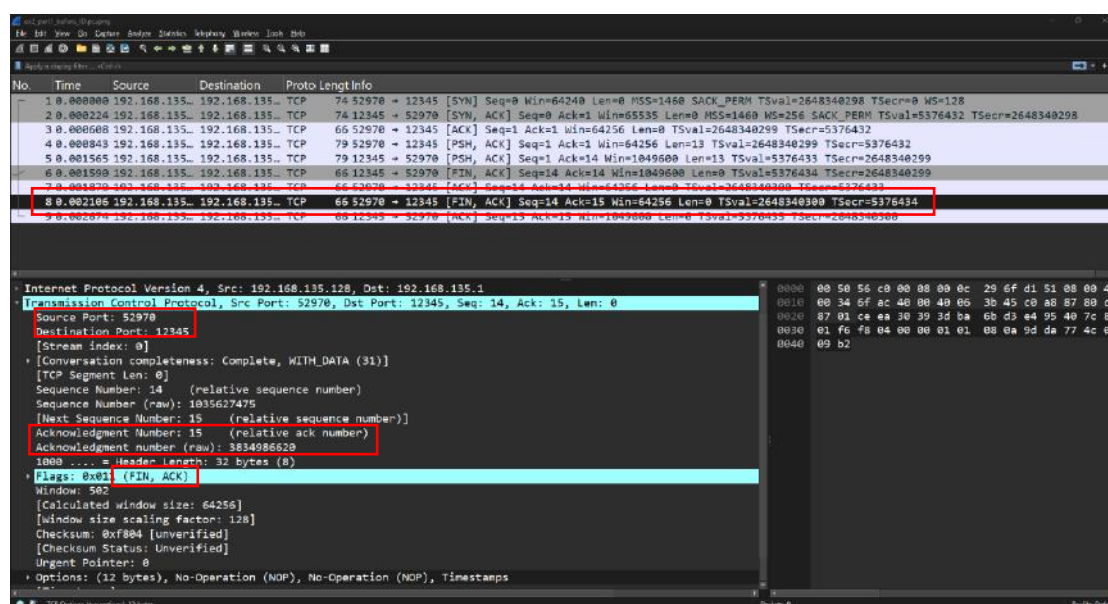
מיד לאחר שהשרת שולח את הודעת ack יחד עם התוכן בupper, הוא שולח הודעת fin (ניתן לראות דגל דולק), שכן הוא עשה את שלו, ותפקיד ההודעה הזו להודיע על כך שהוא מעוניין לסיים את מערכת היחסים עם הלקוח. נשים לב שעכשיו seq של השרת גדל ב3 גם הוא לערך – '3834986619'.

חבילה מס' 7 –



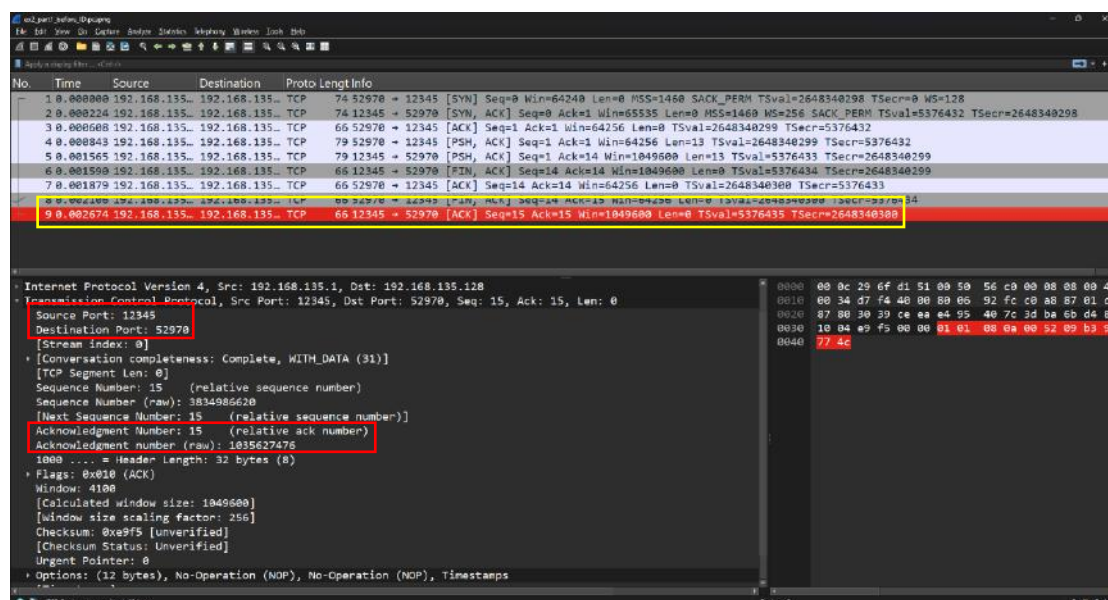
הלקוח מקבל את הדאטא חזרה מהשרת בupper, בהתאמה הוא מגדיל את ערך הackedn ב13 בתים נוספים לערך – '3834986619'. כמו כן נשים לב שגם ערך הseq המצוין בהודעה גדל ב-13 מהפעם האחרונה בה ראינו חבילה נשלחת מהלקוח, וכעת הוא עומד על – '1035627475' (שוקינג זה ack של השרת..).

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 החיף דלוק בה). דהיינו תפקיד ההודעה לומר לשרת שהסיום הינו הדדי. מעבר לזה אין מידע מעניין למעט העובדה שערך האקס של הלקוח גדל ב-1 כתוצאה מהפאנטום ביט של הודעה החיף שמוכרת רק עכשיו.

חבילה מס' 9 –



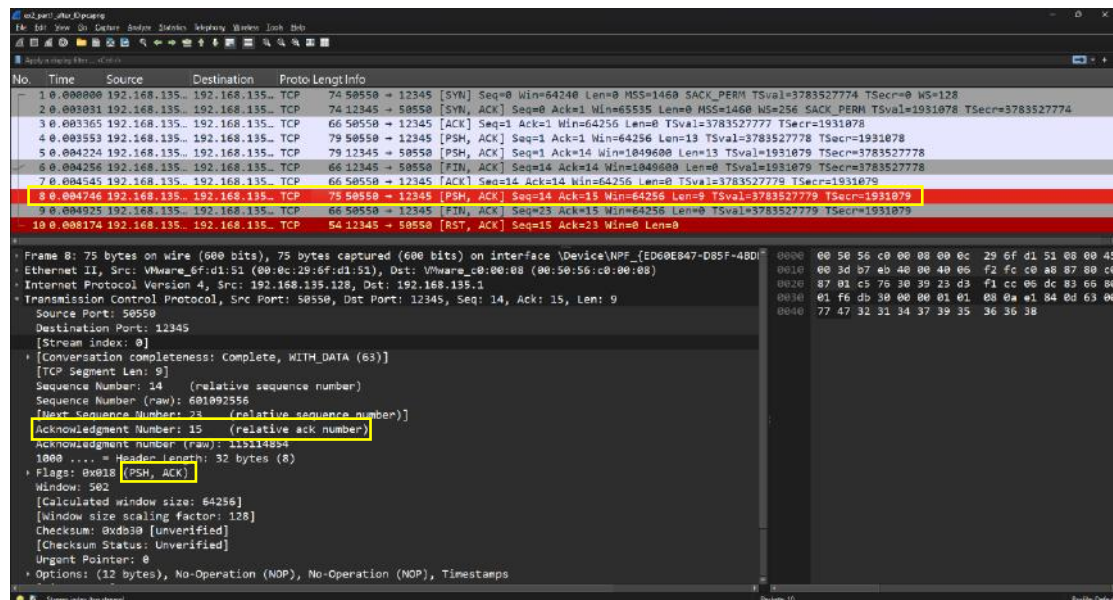
בחבילה האחרונה (🙄) שלנו אנו רואים את השרת מכיר בסיום ההתקשרות מצד הלקוח גם הוא בהודעת ack, ובאופן פרקטי תפקיד ההודעה הינו לסיים לחלוטין בהחלט את ערוץ התקשורת שהיה ביניהם. נשים לב שרגע לפני שהוא אומר "ביי", הוא מגדיל גם הוא את ערך האקס כתוצאה מהפאנטום ביט של החיף שהגיע מהלקוח, סה"כ ערך האקס של השרת בסיום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

סיימנו לנתח את התקשורת בחלק הראשון. נסיף כעת את שליחת ה-t.z. של אילאי לאחר שהשרת משיב את תשובתו ונראה את השינויים.

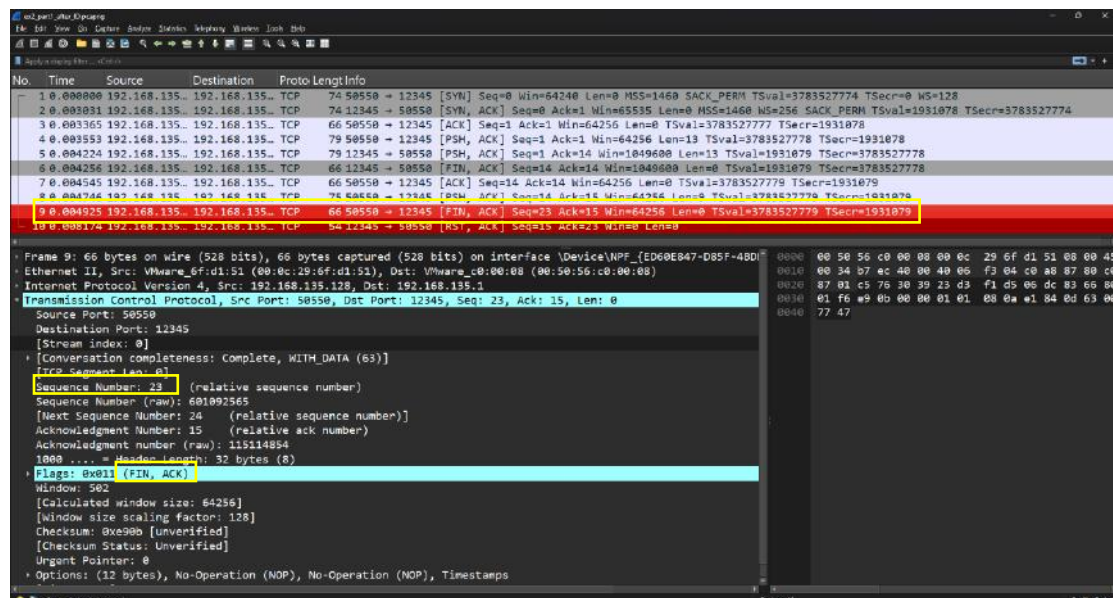
```
tcp_server.py
1 import socket
2 server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
3 server.bind(('', 12345))
4 server.listen(5)
5
6 while True:
7     client_socket, client_address = server.accept()
8     print('Connection from: ', client_address)
9     data = client_socket.recv(100)
10    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.send(b'214795668')
8 s.close()
```

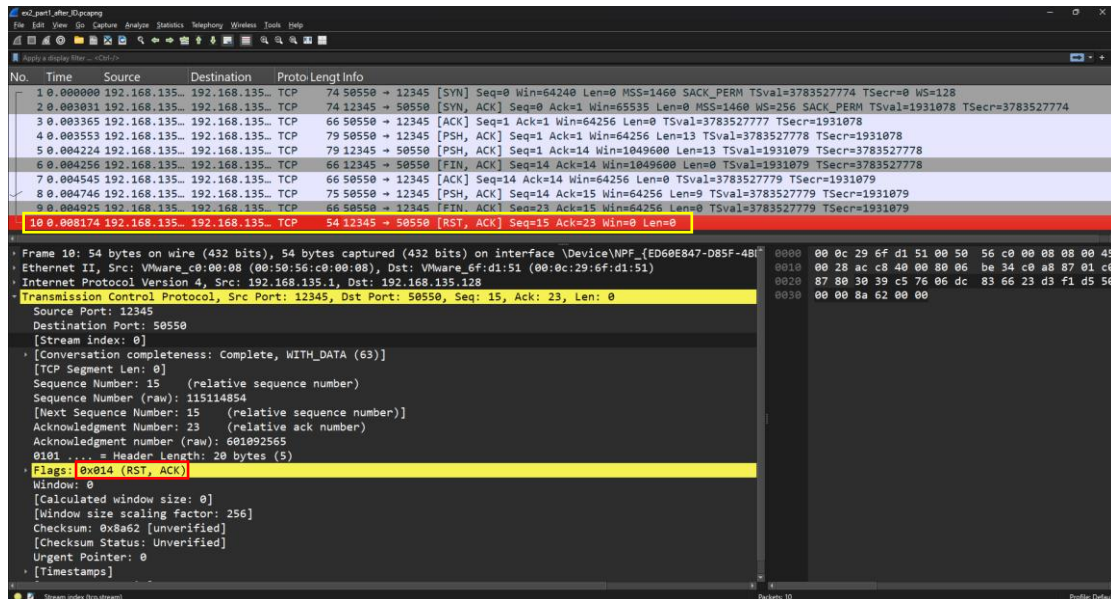
כל ההודעות בהתחלה תתנהגנה באופן זהה (לכן נחסוך ונתחיל רק מהמקום בו השינוי קורה), אבל עכשיו נשים לב –



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 לעשות לה psh לאפליקציה (הלוא היא ה-t.z.), ורק לאחר שההודעה הזו נשלחת, הוא שולח גם כן fin מצידו –



seq 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:

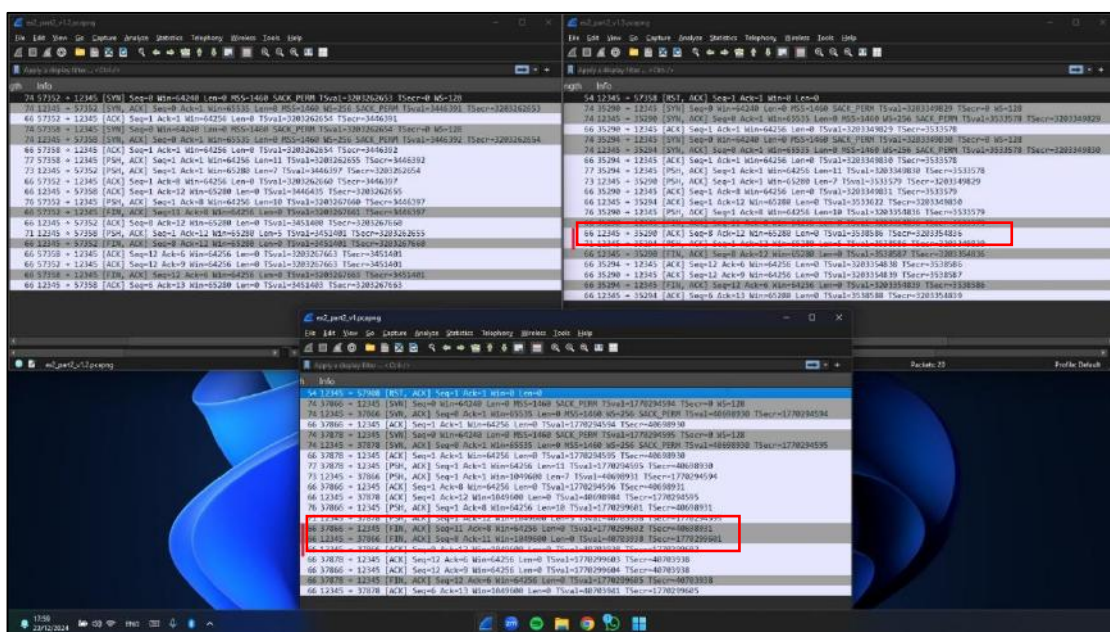
גרסה v1:

The first screenshot shows the code for server.py. It imports socket and sys, sets TCP_IP to '0.0.0.0' and TCP_PORT to 5798. It binds a socket to these values and listens. In a while loop, it accepts connections, prints the address, receives data, and sends it back. It also handles a second connection. The second screenshot shows client.py, which imports socket and sys, sets TCP_IP and TCP_PORT from command-line arguments, connects to the server, sends a message, and receives a response. The third screenshot is a Wireshark packet capture showing a TCP connection from 192.168.135.1 to 192.168.135.128. It highlights the SYN, ACK, and FIN packets, and shows the details of the first packet, including the source and destination ports and the sequence and acknowledgment numbers.

הלקוח מקבל את ה IP והפורט מהארגומנטים לתוכנית, לאחר מכן הוא מתחבר בשני חיבורים נפרדים בעזרת שני סוקטים נפרדים (שורות 4 עד 9). הלקוח תחילה שולח הודעה דרך החיבור השני, מחכה 5 שניות ושולח 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.

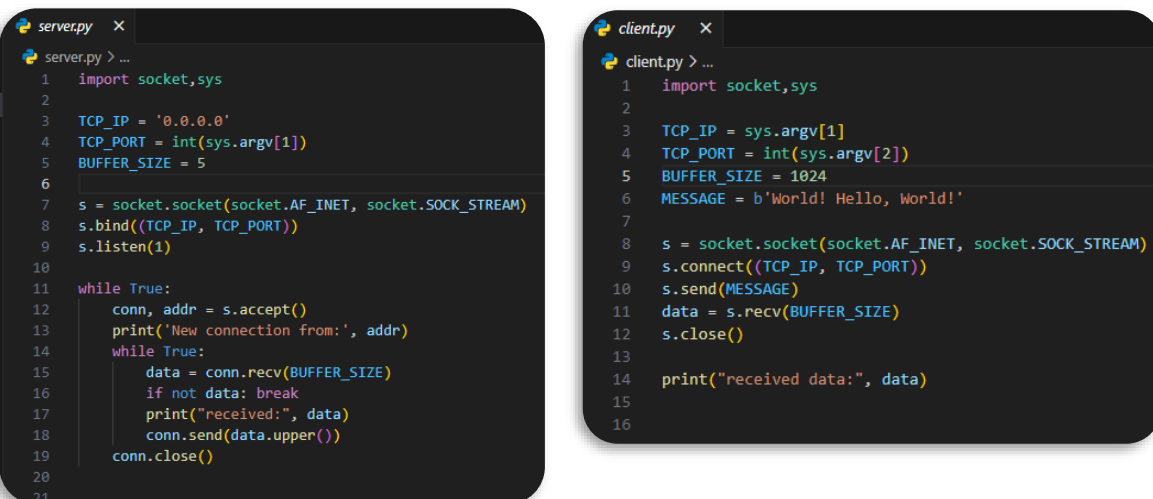
נשים לב שבפועל בכריש קרה משהו מעניין. לאחר האתחול עד שורה 9, נשלחת ההודעה מהסוקט שנוצר שני בלקוח, והשרת לפני שהוא מחזיר עליה ack לאותו סוקט שני, קודם שולח את ההודעה המקוצרת (7 בתים), לחיבור הראשון (שורה 11). יתרה מזאת, הוא מקבל מהחיבור הראשון ack על ההודעה המקוצרת (שורה 12), ורק אז מחזיר ack על עצם זה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

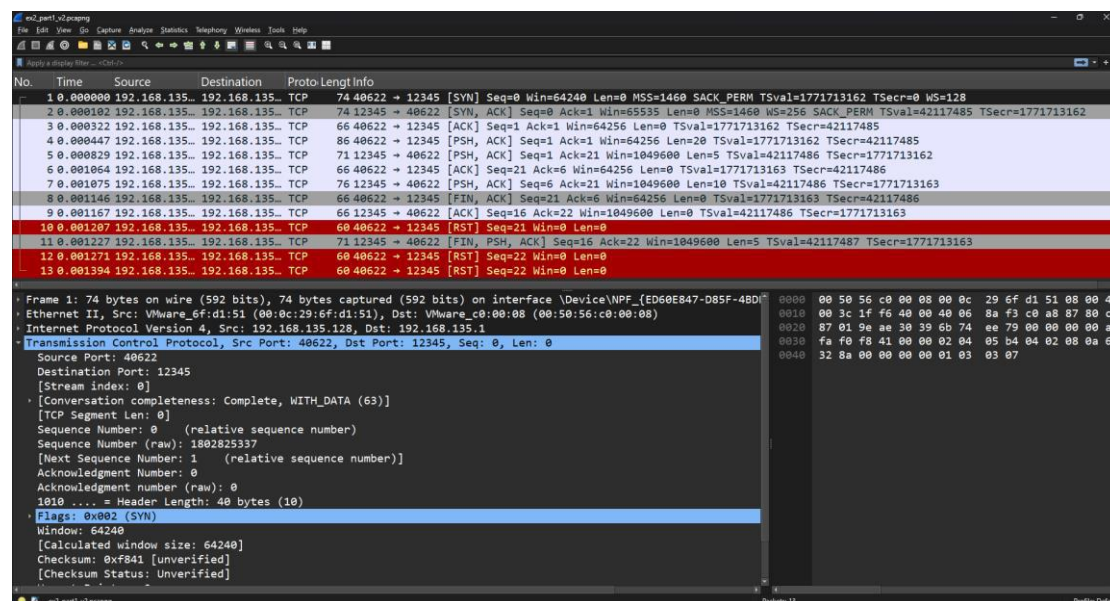
לאחר הרצה מס' פעמים לא היו שינויים משמעותיים –



הדבר היחיד הוא שבאחת ההרצות (העליונה בימין) לאחר השליחה האחרונה לשרת מהקליינט הוא מיד שולח את 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 את הFIN הדדי.

גרסה v2:

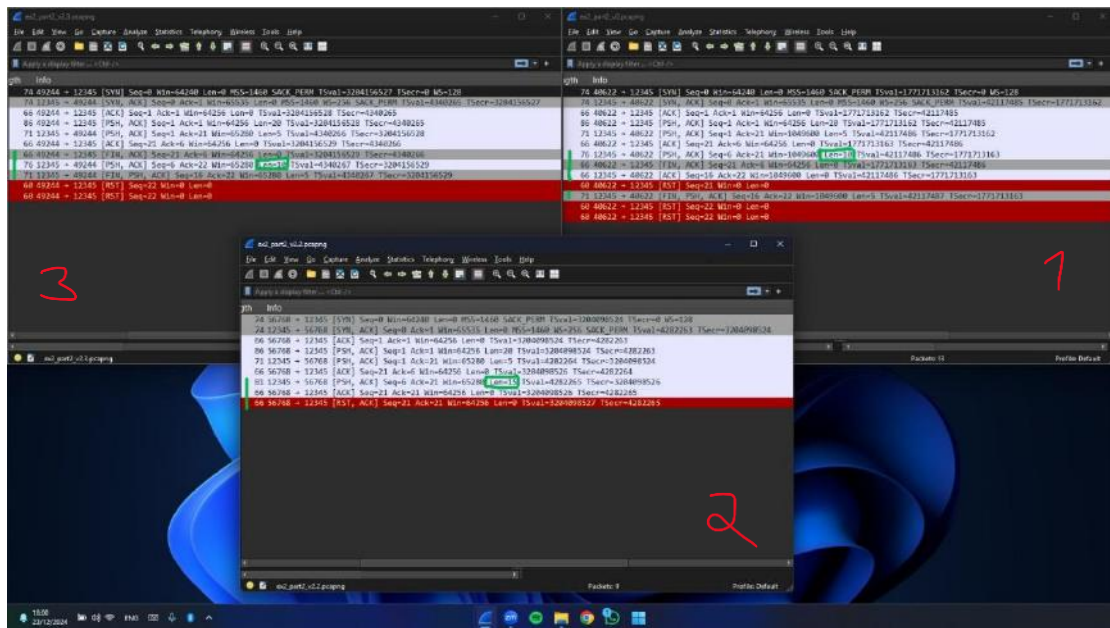




כמו בתוכנית הקודמת, הלקוח מקבל את ה IP והפורט מהארגומנטים לתוכנית, לאחר מכן הוא מתחבר לשרת ושולח לו את ההודעה המצוינת. הוא עושה recv מהשרת, ולאחר שקיבל את ההודעה שהגיעה חזרה מהשרת הוא מדפיס אותה. מצד שני, כמו בתוכנית הקודמת, השרת מאזין לכל החיבורים כיוון שהוא בחר בכתובת 0.0.0.0 ומקבל פורט כקלט. הוא פותח 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 האck 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 fin שהשרת שולח – הכל מקבל rst, שכן הלקוח ניתק בצד השני...

לאחר הרצה מס' פעמים אלו היו השינויים שראינו –

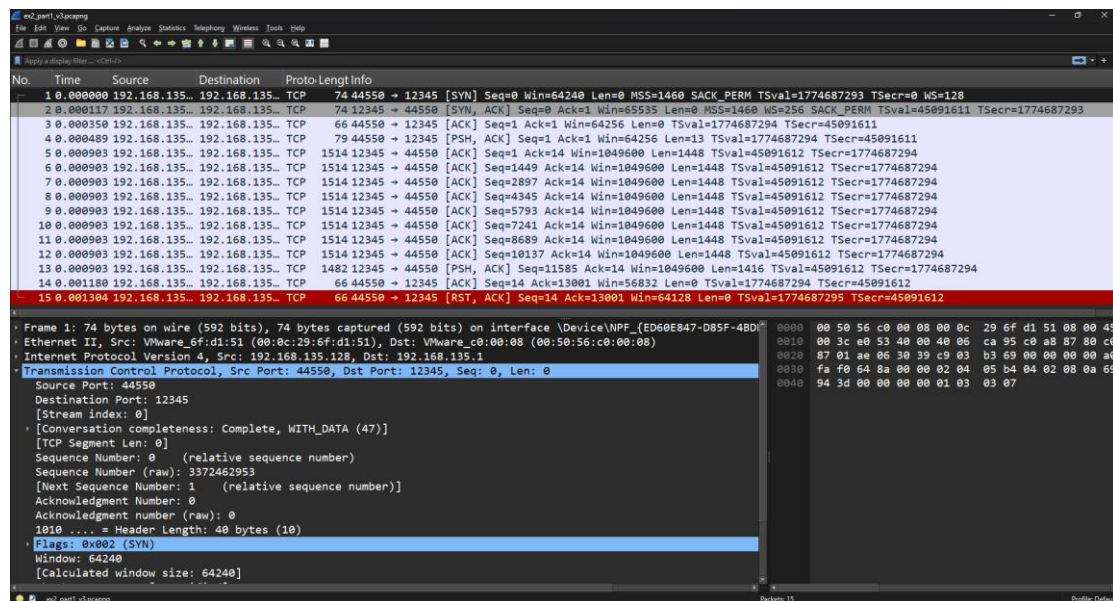


שמנו לב שאופן חלוקת המידע לפי השרת עונה ללקוח עשוי להשתנות מהרצה להרצה (גם הסדר של "מתי הלקוח שולח חי, לפי קבלת החבילה או אחרי..", זאת כמובן ראינו כבר בדוג' בגרסה הקודמת). בפעם השנייה שהרצנו ראינו שהשרת עשה "סטאק" ליותר בתים (15 במקום 10) ותכנן לשלוח אותם ביחד. שוב זה נובע מאלגו' נייגל. תחילה הוא שולח חבילה – צאנק יחיד, כיוון שאין משהו on the fly. ברגע שהוא מקבל עוד 5 בתים לשלוח הוא לא ממחר 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 rst בכל מצב). הריצה ה-3 זהה לראשונה, רק ששם הלקוח עשה FIN בטרם קיבל חבילה מהשרת (מה שתיארנו כבר בדוג' הקודמת).

גרסה v3:

```
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()
16
17
18
19
```



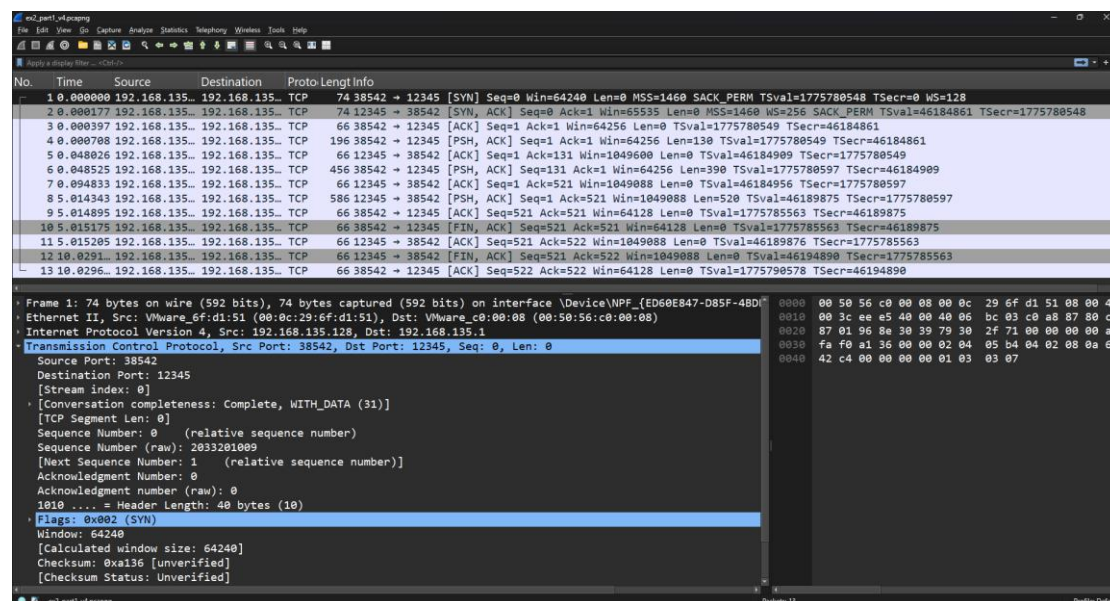
כמו בתוכנית הקודמת, הקוד לא השתנה המון – הלקוח מקבל את התוכנית, לאחר מכן הוא מתחבר לשרת ושולח לו הודעה, אך בשונה מהתוכנית הקודמת, לאחר שקיבל את ההודעה חזרה מהשרת הוא מושך ומדפיס את ה-1024 בתים הראשונים ולאחר מכן שוב הוא מנסה למשוך עוד 1024 בתים. כמו כן בצד שני, כמו בתוכנית הקודמת, השרת מאזין לכל החיבורים כיוון שהוא בחר בכתובת 0.0.0.0 ומקבל פורט קקלט. 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.

גרסה v4:

```
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)
```



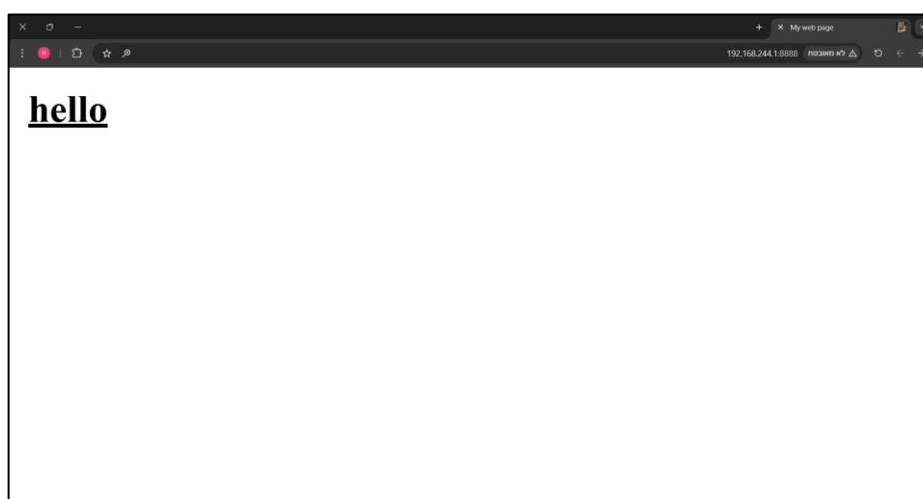
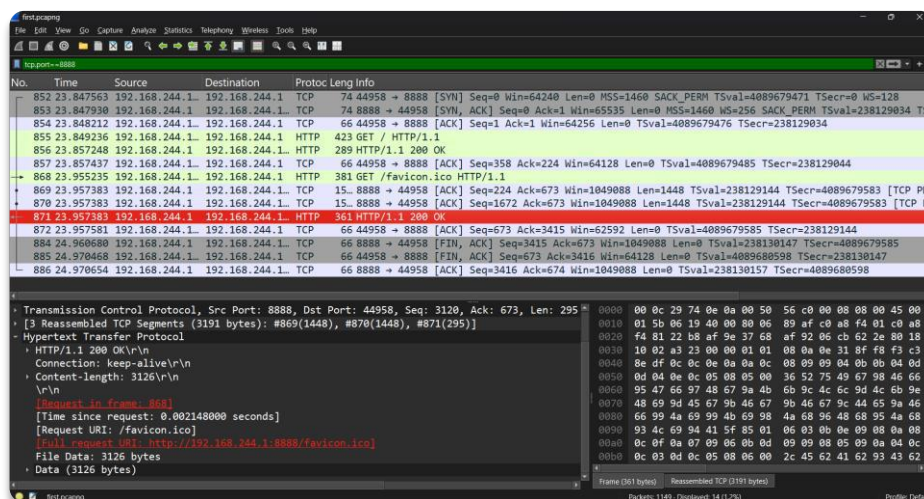
כרגיל הלקוח מקבל את ה IP והפורט מהארגומנטים לתוכנית, לאחר מכן הוא מתחבר לשרת ושולח לו 4 הודעות שכל הודעה היא 10 פעמים "Hello, World", השוני הפעם מהתוכנית הקודמת, הוא שלאחר שקיבל את ההודעה חזרה מהשרת הוא מדפיס את ה 1024 בתים הראשונים. בצד שני, כרגיל השרת מאזין לכל החיבורים כיוון שהוא בחר בכתובת 0.0.0.0 ומקבל פורט קקלט. הוא פותח חיבור מחכה 5 שניות, מקבל את ה 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 ב timeout באפליקציה ולכן לא קורא מהבאפר), מחזיר ack שהוא קיבל את ההודעה. השרת שולח את כל ה 520 בתים בחזרה בפעם אחת (מאוחדת) באותיות גדולות, ומקבל ack עליו מהשרת. מכאן והילך טקס הפרידה הסטנדרטי של fin-ack של 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, 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 (בתמונה רואים את הקובץ עליו הוחזר 200 למטה בbody), והתקשורת נסגרת באופן הדדי בחינים משני הצדדים. ניתן לראות שנשלח המון מידע עם הודעת get מדפדפן, כמו סוג הדפדפן, הגרסה שלו, סטטוס connection, עדיפות ועוד. מעבר לget, ההחזרה של המידע מהשרת – 200, ואז הבקשה לאייקון אין משהו מיוחד, ואין בקשה נוספת מאותו פורט.

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

Time	Source	Destination	Proto	Length	Info
0.0000...	192.168.24...	192.168.24...	TCP	66	54010 → 8888 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
0.0005...	192.168.24...	192.168.24...	TCP	66	8888 → 54010 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
0.0006...	192.168.24...	192.168.24...	TCP	54	54010 → 8888 [ACK] Seq=1 Ack=1 Win=131328 Len=0
0.0007...	192.168.24...	192.168.24...	HTTP	524	GET /c/Footube.html HTTP/1.1
0.0008...	192.168.24...	192.168.24...	TCP	60	8888 → 54010 [ACK] Seq=1 Ack=471 Win=64128 Len=0
0.0010...	192.168.24...	192.168.24...	TCP	15..	8888 → 54010 [ACK] Seq=1 Ack=471 Win=64128 Len=1460 [TCP PDU reassembled in 14]
0.0010...	192.168.24...	192.168.24...	TCP	15..	8888 → 54010 [ACK] Seq=1461 Ack=471 Win=64128 Len=1460 [TCP PDU reassembled in 14]
0.0010...	192.168.24...	192.168.24...	TCP	15..	8888 → 54010 [ACK] Seq=2921 Ack=471 Win=64128 Len=1460 [TCP PDU reassembled in 14]
0.0010...	192.168.24...	192.168.24...	TCP	15..	8888 → 54010 [ACK] Seq=4381 Ack=471 Win=64128 Len=1460 [TCP PDU reassembled in 14]
0.0010...	192.168.24...	192.168.24...	HTTP	360	HTTP/1.1 200 OK
0.0011...	192.168.24...	192.168.24...	TCP	54	54010 → 8888 [ACK] Seq=471 Ack=6147 Win=131328 Len=0
0.0047...	192.168.24...	192.168.24...	TCP	66	54011 → 8888 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
0.0053...	192.168.24...	192.168.24...	TCP	66	8888 → 54011 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
0.0054...	192.168.24...	192.168.24...	TCP	54	54011 → 8888 [ACK] Seq=1 Ack=1 Win=131328 Len=0
0.0023...	192.168.24...	192.168.24...	TCP	66	54012 → 8888 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
0.0026...	192.168.24...	192.168.24...	TCP	66	8888 → 54012 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
0.0029...	192.168.24...	192.168.24...	TCP	54	54012 → 8888 [ACK] Seq=1 Ack=1 Win=131328 Len=0
0.0030...	192.168.24...	192.168.24...	TCP	66	54013 → 8888 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
0.0031...	192.168.24...	192.168.24...	TCP	66	8888 → 54013 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
0.0035...	192.168.24...	192.168.24...	TCP	54	54013 → 8888 [ACK] Seq=1 Ack=1 Win=131328 Len=0
0.0039...	192.168.24...	192.168.24...	TCP	66	54014 → 8888 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
0.0040...	192.168.24...	192.168.24...	TCP	66	8888 → 54014 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
0.0042...	192.168.24...	192.168.24...	TCP	54	54014 → 8888 [ACK] Seq=1 Ack=1 Win=131328 Len=0
0.0042...	192.168.24...	192.168.24...	HTTP	429	GET /c/footube.css HTTP/1.1
0.0042...	192.168.24...	192.168.24...	HTTP	473	GET /c/img/1.jpg HTTP/1.1

Transmission Control Protocol, Src Port: 54010, Dst Port: 8888, Seq: 1, Ack: 1, Len: 470

HyperText Transfer Protocol

GET /c/Footube.html HTTP/1.1\r\n\r\nHost: 192.168.244.129:8888\r\n\r\nConnection: keep-alive\r\n\r\nUpgrade-Insecure-Requests: 1\r\n\r\n

0030 02 01 cc 17 00 00 47 45 54 20 2f 63 2f 66 6f 6f 6f 0040 74 75 62 65 2e 68 74 6d 6c 20 48 54 50 31 2e 0050 2e 31 0d 0a 48 6f 73 74 3a 20 31 39 32 0060 38 2e 32 34 2e 31 32 39 3a 38 38 0070 43 6f 6e 6e 65 63 74 69 6f 6e 3a 20 6b 0080 2d 61 6c 69 76 65 0d 0a 55 70 67 72 61 61

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.244.1	192.168.244.1	TCP	54	54001 → 8888 [FIN, ACK] Seq=1 Ack=1 Win=1026 Len=0
2	0.000040	192.168.244.1	192.168.244.1	TCP	54	54002 → 8888 [FIN, ACK] Seq=1 Ack=1 Win=1026 Len=0
3	0.000201	192.168.244.1	192.168.244.1	TCP	60	8888 → 54001 [ACK] Seq=1 Ack=2 Win=501 Len=0
4	0.000231	192.168.244.1	192.168.244.1	TCP	60	8888 → 54002 [ACK] Seq=1 Ack=2 Win=501 Len=0
5	0.000373	192.168.244.1	192.168.244.1	TCP	66	54010 → 8888 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
6	0.000506	192.168.244.1	192.168.244.1	TCP	66	8888 → 54010 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
7	0.000613	192.168.244.1	192.168.244.1	TCP	54	54010 → 8888 [ACK] Seq=1 Ack=1 Win=131328 Len=0
8	0.000736	192.168.244.1	192.168.244.1	HTTP	524	GET /c/Footube.html HTTP/1.1
9	0.000856	192.168.244.1	192.168.244.1	TCP	60	8888 → 54010 [ACK] Seq=1 Ack=471 Win=64128 Len=0
10	0.001077	192.168.244.1	192.168.244.1	TCP	1514	8888 → 54010 [ACK] Seq=1 Ack=471 Win=64128 Len=1460 [TCP PDU reassembled in 14]
11	0.001082	192.168.244.1	192.168.244.1	TCP	1514	8888 → 54010 [ACK] Seq=1461 Ack=471 Win=64128 Len=1460 [TCP PDU reassembled in 14]
12	0.001085	192.168.244.1	192.168.244.1	TCP	1514	8888 → 54010 [ACK] Seq=2921 Ack=471 Win=64128 Len=1460 [TCP PDU reassembled in 14]
13	0.001087	192.168.244.1	192.168.244.1	TCP	1514	8888 → 54010 [ACK] Seq=4381 Ack=471 Win=64128 Len=1460 [TCP PDU reassembled in 14]
14	0.001089	192.168.244.1	192.168.244.1	HTTP	360	HTTP/1.1 200 OK
15	0.001186	192.168.244.1	192.168.244.1	TCP	54	54010 → 8888 [ACK] Seq=471 Ack=6147 Win=131328 Len=0
16	0.004777	192.168.244.1	192.168.244.1	TCP	66	54011 → 8888 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
17	0.005301	192.168.244.1	192.168.244.1	TCP	66	8888 → 54011 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
18	0.005466	192.168.244.1	192.168.244.1	TCP	54	54011 → 8888 [ACK] Seq=1 Ack=1 Win=131328 Len=0
19	0.002336	192.168.244.1	192.168.244.1	TCP	66	54012 → 8888 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
20	0.002606	192.168.244.1	192.168.244.1	TCP	66	8888 → 54012 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
21	0.002902	192.168.244.1	192.168.244.1	TCP	54	54012 → 8888 [ACK] Seq=1 Ack=1 Win=131328 Len=0
22	0.003037	192.168.244.1	192.168.244.1	TCP	66	54013 → 8888 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
23	0.003132	192.168.244.1	192.168.244.1	TCP	66	8888 → 54013 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
24	0.003519	192.168.244.1	192.168.244.1	TCP	54	54013 → 8888 [ACK] Seq=1 Ack=1 Win=131328 Len=0
25	0.003952	192.168.244.1	192.168.244.1	TCP	66	54014 → 8888 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
26	0.004083	192.168.244.1	192.168.244.1	TCP	66	8888 → 54014 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
27	0.004203	192.168.244.1	192.168.244.1	TCP	54	54014 → 8888 [ACK] Seq=1 Ack=1 Win=131328 Len=0
28	0.004229	192.168.244.1	192.168.244.1	HTTP	429	GET /c/footube.css HTTP/1.1
29	0.004292	192.168.244.1	192.168.244.1	HTTP	473	GET /c/img/1.jpg HTTP/1.1

Transmission Control Protocol, Src Port: 54010, Dst Port: 8888, Seq: 471, Ack: 6147, Len: 375

HyperText Transfer Protocol

GET /c/footube.css HTTP/1.1\r\n\r\nHost: 192.168.244.129:8888\r\n\r\nConnection: keep-alive\r\n\r\nUser-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.0.0 Safari/537\r\n\r\nAccept: text/css,*/*;q=0.1\r\n\r\nReferer: http://192.168.244.129:8888/c/Footube.html\r\n\r\n

0030 02 01 7f 3a 00 00 47 45 54 20 2f 63 2f 66 6f 6f 6f 0040 74 75 62 65 2e 68 73 73 20 48 54 50 31 2e 0050 31 0d 6a 48 6f 73 74 3a 20 31 39 32 0060 2e 32 34 2e 31 32 39 3a 38 38 38 0070 6f 6e 6e 65 63 74 69 6f 6e 3a 20 6b 65 65 70 2d 0080 61 6c 69 76 65 0d 0a 55 73 65 72 2d 41 67 65 6e 74 3a 20 6d 6f 74 69 6c 6e 61 2f 65 2e 30 2e 00a0 57 69 6e 64 6f 77 73 20 4a 54 20 31 30 2e 30 30

Time	Source	Destination	Protocol	Length	Info
5.0647...	192.168.24...	192.168.24...	TCP	15..	8888 → 54015 [ACK] Seq=511801 Ack=420 Win=64128 Len=1460 [TCP PDU reassembled in 2491]
5.0647...	192.168.24...	192.168.24...	TCP	15..	8888 → 54015 [ACK] Seq=512461 Ack=420 Win=64128 Len=1460 [TCP PDU reassembled in 2491]
5.0647...	192.168.24...	192.168.24...	TCP	15..	8888 → 54015 [ACK] Seq=513921 Ack=420 Win=64128 Len=1460 [TCP PDU reassembled in 2491]
5.0647...	192.168.24...	192.168.24...	TCP	15..	8888 → 54015 [ACK] Seq=515381 Ack=420 Win=64128 Len=1460 [TCP PDU reassembled in 2491]
5.0647...	192.168.24...	192.168.24...	TCP	15..	8888 → 54015 [ACK] Seq=516841 Ack=420 Win=64128 Len=1460 [TCP PDU reassembled in 2491]
5.0647...	192.168.24...	192.168.24...	TCP	15..	8888 → 54015 [ACK] Seq=518301 Ack=420 Win=64128 Len=1460 [TCP PDU reassembled in 2491]
5.0647...	192.168.24...	192.168.24...	TCP	15..	8888 → 54015 [ACK] Seq=519761 Ack=420 Win=64128 Len=1460 [TCP PDU reassembled in 2491]
5.0647...	192.168.24...	192.168.24...	TCP	15..	8888 → 54015 [ACK] Seq=521221 Ack=420 Win=64128 Len=1460 [TCP PDU reassembled in 2491]
5.0647...	192.168.24...	192.168.24...	TCP	15..	8888 → 54015 [ACK] Seq=522681 Ack=420 Win=64128 Len=1460 [TCP PDU reassembled in 2491]
5.0648...	192.168.24...	192.168.24...	TCP	15..	8888 → 54015 [ACK] Seq=524141 Ack=420 Win=64128 Len=1460 [TCP PDU reassembled in 2491]
5.0648...	192.168.24...	192.168.24...	TCP	15..	8888 → 54015 [ACK] Seq=525601 Ack=420 Win=64128 Len=1460 [TCP PDU reassembled in 2491]
5.0648...	192.168.24...	192.168.24...	TCP	15..	8888 → 54015 [ACK] Seq=527061 Ack=420 Win=64128 Len=1460 [TCP PDU reassembled in 2491]
5.0648...	192.168.24...	192.168.24...	TCP	15..	8888 → 54015 [ACK] Seq=528521 Ack=420 Win=64128 Len=1460 [TCP PDU reassembled in 2491]
5.0648...	192.168.24...	192.168.24...	TCP	15..	8888 → 54015 [ACK] Seq=529981 Ack=420 Win=64128 Len=1460 [TCP PDU reassembled in 2491]
5.0648...	192.168.24...	192.168.24...	HTTP	604	HTTP/1.1 200 OK (JPEG image)
5.0648...	192.168.24...	192.168.24...	TCP	54	54015 → 8888 [ACK] Seq=420 Ack=432161 Win=131328 Len=0
5.0648...	192.168.24...	192.168.24...	TCP	54	54015 → 8888 [ACK] Seq=420 Ack=446761 Win=131328 Len=0
5.0649...	192.168.24...	192.168.24...	TCP	54	54015 → 8888 [ACK] Seq=420 Ack=461361 Win=131328 Len=0
5.0650...	192.168.24...	192.168.24...	TCP	54	54015 → 8888 [ACK] Seq=420 Ack=475961 Win=131328 Len=0
5.0651...	192.168.24...	192.168.24...	TCP	54	54015 → 8888 [ACK] Seq=420 Ack=490561 Win=131328 Len=0
5.0652...	192.168.24...	192.168.24...	TCP	54	54015 → 8888 [ACK] Seq=420 Ack=505161 Win=131328 Len=0
5.0653...	192.168.24...	192.168.24...	TCP	54	54015 → 8888 [ACK] Seq=420 Ack=519761 Win=131328 Len=0
5.0655...	192.168.24...	192.168.24...	TCP	54	54015 → 8888 [ACK] Seq=420 Ack=531991 Win=131328 Len=0
6.0635...	192.168.24...	192.168.24...	TCP	60	8888 → 54015 [FIN, ACK] Seq=531991 Ack=420 Win=64128 Len=0
6.0637...	192.168.24...	192.168.24...	TCP	54	54015 → 8888 [ACK] Seq=420 Ack=531992 Win=131328 Len=0

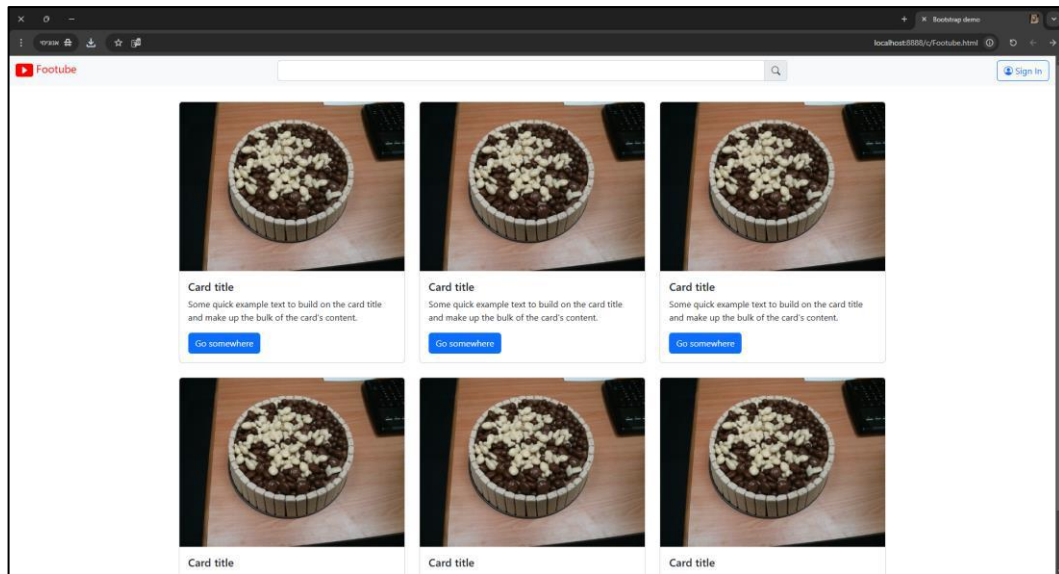
Frame 2491, 604 bytes on wire (4832 bits), 604 bytes captured (4832 bits) on interface \Device\NPF{...}

Ethernet II, Src: VMware_74:0e:0a (00:0c:29:74:0e:0a), Dst: VMware_c0:00:08 (00:50:56:c0:00:08)

Internet Protocol Version 4, Src: 192.168.244.129, Dst: 192.168.244.1

Transmission Control Protocol, Src Port: 8888, Dst Port: 54015, Seq: 531441, Ack: 420, Len: 550

[365 Reassembled TCP Segments (531990 bytes): #2095(1460), #2096(1460), #2097(1460), #2098(1460), #2099(1460), #2100(1460), #2101(1460), #2102(1460), #2103(1460), #2104(1460), #2105(1460), #2106(1460), #2107(1460), #2108(1460), #2109(1460), #2110(1460), #2111(1460), #2112(1460), #2113(1460), #2114(1460), #2115(1460), #2116(1460), #2117(1460), #2118(1460), #2119(1460), #2120(1460), #2121(1460), #2122(1460), #2123(1460), #2124(1460), #2125(1460), #2126(1460), #2127(1460), #2128(1460), #2129(1460), #2130(1460), #2131(1460), #2132(1460), #2133(1460), #2134(1460), #2135(1460), #2136(1460), #2137(1460), #2138(1460), #2139(1460), #2140(1460), #2141(1460), #2142(1460), #2143(1460), #2144(1460), #2145(1460), #2146(1460), #2147(1460), #2148(1460), #2149(1460), #2150(1460), #2151(1460), #2152(1460), #2153(1460), #2154(1460), #2155(1460), #2156(1460), #2157(1460), #2158(1460), #2159(1460), #2160(1460), #2161(1460), #2162(1460), #2163(1460), #2164(1460), #2165(1460), #2166(1460), #2167(1460), #2168(1460), #2169(1460), #2170(1460), #2171(1460), #2172(1460), #2173(1460), #2174(1460), #2175(1460), #2176(1460), #2177(1460), #2178(1460), #2179(1460), #2180(1460), #2181(1460), #2182(1460), #2183(1460), #2184(1460), #2185(1460), #2186(1460), #2187(1460), #2188(1460), #2189(1460), #2190(1460), #2191(1460), #2192(1460), #2193(1460), #2194(1460), #2195(1460), #2196(1460), #2197(1460), #2198(1460), #2199(1460), #2200(1460), #2201(1460), #2202(1460), #2203(1460), #2204(1460), #2205(1460), #2206(1460), #2207(1460), #2208(1460), #2209(1460), #2210(1460), #2211(1460), #2212(1460), #2213(1460), #2214(1460), #2215(1460), #2216(1460), #2217(1460), #2218(1460), #2219(1460), #2220(1460), #2221(1460), #2222(1460), #2223(1460), #2224(1460), #2225(1460), #2226(1460), #2227(1460), #2228(1460), #2229(1460), #2230(1460), #2231(1460), #2232(1460), #2233(1460), #2234(1460), #2235(1460), #2236(1460), #2237(1460), #2238(1460), #2239(1460), #2240(1460), #2241(1460), #2242(1460), #2243(1460), #2244(1460), #2245(1460), #2246(1460), #2247(1460), #2248(1460), #2249(1460), #2250(1460), #2251(1460), #2252(1460), #2253(1460), #2254(1460), #2255(1460), #2256(1460), #2257(1460), #2258(1460), #2259(1460), #2260(1460), #2261(1460), #2262(1460), #2263(1460), #2264(1460), #2265(1460), #2266(1460), #2267(1460), #2268(1460), #2269(1460), #2270(1460), #2271(1460), #2272(1460), #2273(1460), #2274(1460), #2275(1460), #2276(1460), #2277(1460), #2278(1460), #2279(1460), #2280(1460), #2281(1460), #2282(1460), #2283(1460), #2284(1460), #2285(1460), #2286(1460), #2287(1460), #2288(1460), #2289(1460), #2290(1460), #2291(1460), #2292(1460), #2293(1460), #2294(1460), #2295(1460), #2296(1460), #2297(1460), #2298(1460), #2299(1460), #2300(1460), #2301(1460), #2302(1460), #2303(1460), #2304(1460), #2305



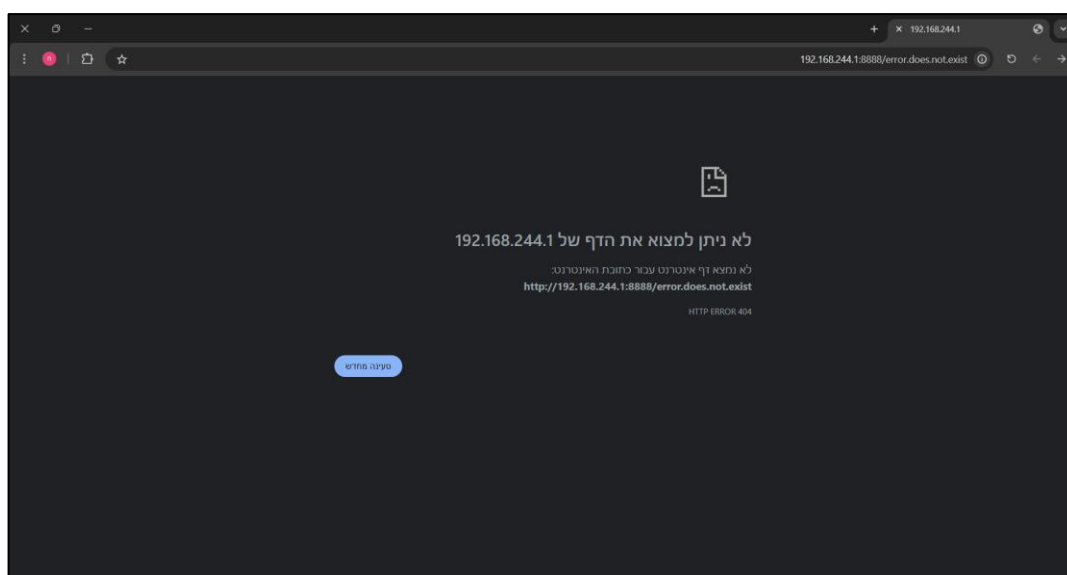
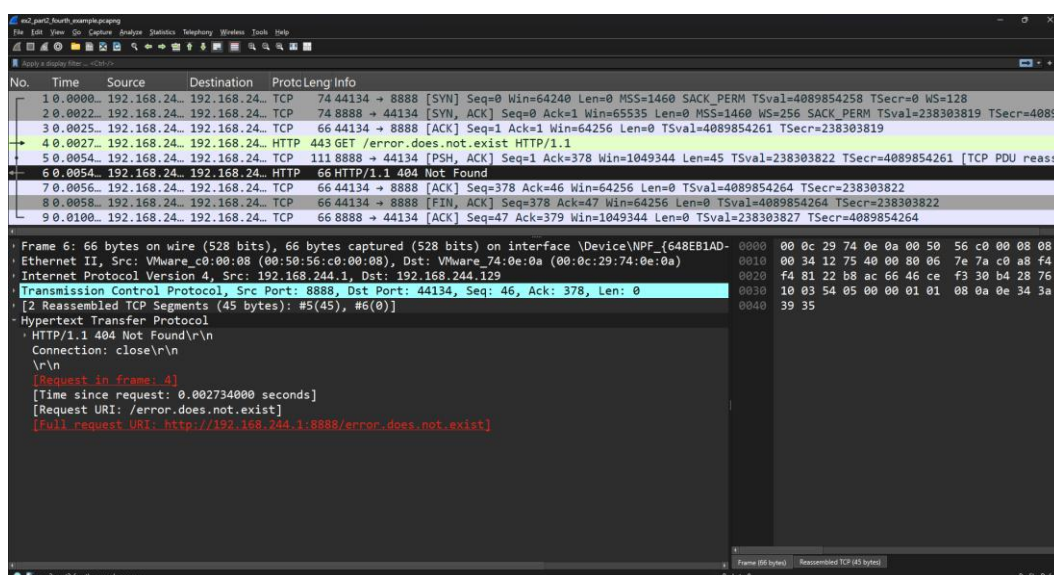
כעת מבקש הדפדפן את קובץ `/c/Footube.html` אשר מכיל בתוכו מס' קבצים (תמונות) נוספים. ניתן לראות שהפעם הוא פותח מס' חיבורים (כ-חמישה) דרך כמה פורטים (תמונה ראשונה), אבל בפועל בשלב הראשון רק דרך החיבור הראשון עוברת התקשורת של הבקשה מהשרת. לאחר שהוא מבקש את הדף הראשון (זה שביקשנו בכתובת), הוא מקבל אותו ומתחיל 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 התקשורת ממשיכה להגיע דרכו. זה גרם המקום להמחיש כדורש במטלה כיצד השרת שלנו תומך בkeep-alive שכן אנו רואים שבראש header 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 (בחן 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 הקצב 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 הטיפ שלה, גודל וכדו').

<http://192.168.244.1:8888/redirect>

The image displays a Wireshark packet capture and a browser window. The Wireshark interface shows a list of network packets. Packet 6, an HTTP 301 Moved Permanently response, is highlighted. The packet details pane shows the status bar as 'FIN, ACK'. The packet bytes pane shows the raw data. The browser window shows the URL 'http://192.168.244.1:8888/redirect' and the page content '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.

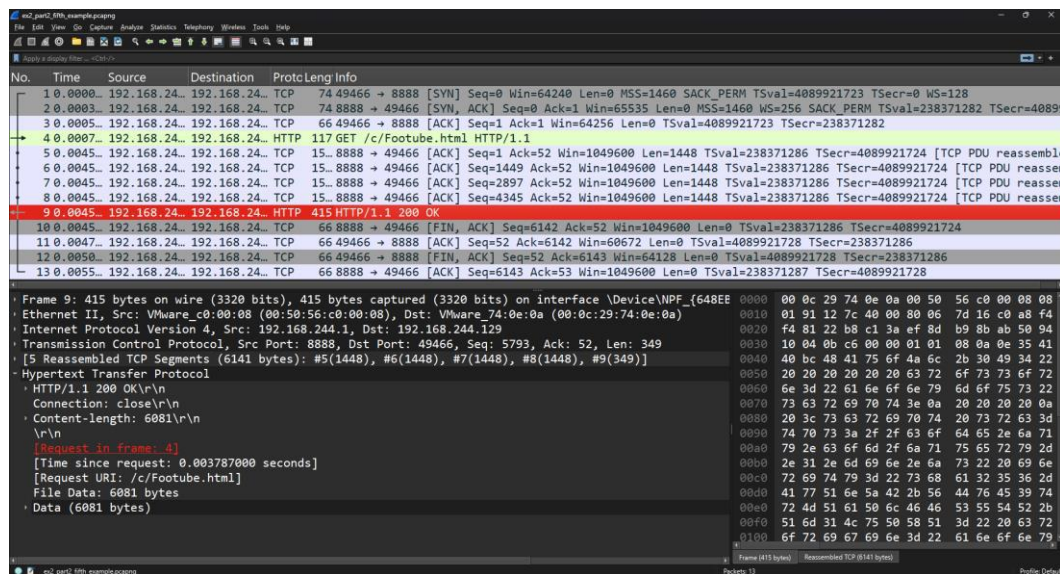
<http://192.168.244.1:8888/error.does.not.exist>



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 ב-finish. 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](http://c/Footube.html)



הלקוח מבקש את הקובץ /c/Footube.html אשר מכיל "עמוד אינטרנטי עם של 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 בחר fin 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

```

No.    Time    Source                Destination           Protocol Length Info
1 0.000000 192.168.244.1 → 192.168.244.1 TCP 74 47648 → 8888 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=4090026322 TSecr=0 WS=128
2 0.000760 192.168.244.1 → 192.168.244.1 TCP 74 8888 → 47648 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM TSval=238475881 TS
3 0.000986 192.168.244.1 → 192.168.244.1 TCP 66 47648 → 8888 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=4090026323 TSecr=238475881
4 0.001067 192.168.244.1 → 192.168.244.1 HTTP 114 GET /favicon.ico HTTP/1.1
5 0.002816 192.168.244.1 → 192.168.244.1 TCP 1514 8888 → 47648 [ACK] Seq=1 Ack=49 Win=1049600 Len=1448 TSval=238475883 TSecr=4090026323 [TCP PD
6 0.002816 192.168.244.1 → 192.168.244.1 TCP 1514 8888 → 47648 [ACK] Seq=1449 Ack=49 Win=1049600 Len=1448 TSval=238475883 TSecr=4090026323 [TCP PD
7 0.002816 192.168.244.1 → 192.168.244.1 HTTP 356 HTTP/1.1 200 OK
8 0.002879 192.168.244.1 → 192.168.244.1 TCP 66 8888 → 47648 [FIN, ACK] Seq=3187 Ack=49 Win=1049600 Len=0 TSval=238475883 TSecr=4090026323
9 0.003077 192.168.244.1 → 192.168.244.1 TCP 66 47648 → 8888 [ACK] Seq=49 Ack=3187 Win=62592 Len=0 TSval=4090026325 TSecr=238475883
10 0.003287 192.168.244.1 → 192.168.244.1 TCP 66 47648 → 8888 [FIN, ACK] Seq=49 Ack=3188 Win=64128 Len=0 TSval=4090026325 TSecr=238475883
11 0.003751 192.168.244.1 → 192.168.244.1 TCP 66 8888 → 47648 [ACK] Seq=3188 Ack=50 Win=1049600 Len=0 TSval=238475884 TSecr=4090026325

* Frame 4: 114 bytes on wire (912 bits), 114 bytes captured (912 bits) on interface \Device\NPF_{648EB1AD-0000-0050-56C0-000000000000}
* Ethernet II, Src: VMware_74:0e:0a (00:0c:29:74:0e:0a), Dst: VMware_c0:00:08 (00:50:56:c0:00:08)
* Internet Protocol Version 4, Src: 192.168.244.129, Dst: 192.168.244.1
* Transmission Control Protocol, Src Port: 47648, Dst Port: 8888, Seq: 1, Ack: 1, Len: 48
* Hypertext Transfer Protocol
  * GET /favicon.ico HTTP/1.1\r\n
  * Connection: close\r\n
  * \r\n
  * [Response in frame 7]

```

```

No.    Time    Source                Destination           Protocol Length Info
1 0.000000 192.168.244.1 → 192.168.244.1 TCP 74 47648 → 8888 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=4090026322 TSecr=0 WS=128
2 0.000760 192.168.244.1 → 192.168.244.1 TCP 74 8888 → 47648 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM TSval=238475881 TS
3 0.000986 192.168.244.1 → 192.168.244.1 TCP 66 47648 → 8888 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=4090026323 TSecr=238475881
4 0.001067 192.168.244.1 → 192.168.244.1 HTTP 114 GET /favicon.ico HTTP/1.1
5 0.002816 192.168.244.1 → 192.168.244.1 TCP 1514 8888 → 47648 [ACK] Seq=1 Ack=49 Win=1049600 Len=1448 TSval=238475883 TSecr=4090026323 [TCP PD
6 0.002816 192.168.244.1 → 192.168.244.1 TCP 1514 8888 → 47648 [ACK] Seq=1449 Ack=49 Win=1049600 Len=1448 TSval=238475883 TSecr=4090026323 [TCP PD
7 0.002816 192.168.244.1 → 192.168.244.1 HTTP 356 HTTP/1.1 200 OK
8 0.002879 192.168.244.1 → 192.168.244.1 TCP 66 8888 → 47648 [FIN, ACK] Seq=3187 Ack=49 Win=1049600 Len=0 TSval=238475883 TSecr=4090026323
9 0.003077 192.168.244.1 → 192.168.244.1 TCP 66 47648 → 8888 [ACK] Seq=49 Ack=3187 Win=62592 Len=0 TSval=4090026325 TSecr=238475883
10 0.003287 192.168.244.1 → 192.168.244.1 TCP 66 47648 → 8888 [FIN, ACK] Seq=49 Ack=3188 Win=64128 Len=0 TSval=4090026325 TSecr=238475883
11 0.003751 192.168.244.1 → 192.168.244.1 TCP 66 8888 → 47648 [ACK] Seq=3188 Ack=50 Win=1049600 Len=0 TSval=238475884 TSecr=4090026325

* Frame 7: 356 bytes on wire (2848 bits), 356 bytes captured (2848 bits) on interface \Device\NPF_{648EB1AD-0000-0050-56C0-000000000000}
* Ethernet II, Src: VMware_c0:00:08 (00:50:56:c0:00:08), Dst: VMware_74:0e:0a (00:0c:29:74:0e:0a)
* Internet Protocol Version 4, Src: 192.168.244.1, Dst: 192.168.244.129
* Transmission Control Protocol, Src Port: 8888, Dst Port: 47648, Seq: 2897, Ack: 49, Len: 290
* [3 Reassembled TCP Segments (3186 bytes): #5(1448), #6(1448), #7(290)]
* Hypertext Transfer Protocol
  * HTTP/1.1 200 OK\r\n
  * Connection: close\r\n
  * Content-length: 3126\r\n
  * \r\n
  * [Response in frame 4]
  * [Time since request: 0.001749000 seconds]
  * [Request URI: /favicon.ico]
  * File Data: 3126 bytes
  * Data (3126 bytes)

```

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 אס 200. 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.