

# פרויקט רשתות

## תקשורת מחשבים

הערות עבור קבצי Wireshark:

1. בקובץ jupyter\_wireshark\_capture.pcap יש להשתמש במסנן:  
`ip.addr == 127.0.0.1 && tcp.port == 80`

2. בקובץ server\_clients\_wireshark\_capture.pcap יש להשתמש במסנן: `tcp.port == 8888`

## חלק 1 – אריזת נתונים ולכידת מנות בעזרת Wireshark

סקירה כללית:

חלק זה של הפרויקט מתמקד בהבנת מודל השכבות של פרוטוקול TCP/IP. המטרה היא להדגים כיצד הודעת טקסט פשוטה משכבת האפליקציה (Application Layer) עוברת תהליך של אריזה (Encapsulation) בשכבות השונות עד להפיכתה למסגרת (Frame) המוכנה לשידור ברשת.

הכנת נתוני הקלט (CSV):

עבור הניסוי, יצרתי קובץ נתונים במבנה CSV המדמה תעבורת HTTP אמיתית בין לקוח לשרת. הקובץ כולל 30 הודעות המייצגות ששן גלישה מלא.

אופן יצירת הקובץ: הנתונים נוצרו בעזרת בינה מלאכותית (Gemini), הצגתי לו את הדרישות עבור הדמיית תעבורה וביקשתי ממנו שידמה תעבורה אמיתית ובנוסף לדמות תרחישי קצה (כגון שגיאות 404 והפניות 302) ושימוש בפורטים דינמיים שונים.

שדות חובה: לכל הודעה הוגדרו msg\_id, פרוטוקול, פורט מקור, פורט יעד, ותוכן ההודעה.

הקובץ CSV ששומש למהלך התקשורת:

msg_id	app_protocol	src_port	dst_port	message	timestamp
0	HTTP	50001	80	GET /index.html	0.010
1	HTTP	50002	80	GET /login.html	0.015
2	HTTP	50003	80	GET /api/status	0.020
3	HTTP	80	50001	HTTP/1.1 200 OK	0.025
4	HTTP	50001	80	GET /style.css	0.030
5	HTTP	80	50002	HTTP/1.1 200 OK (Login Page)	0.035
6	HTTP	50002	80	POST /login_attempt user=admin	0.040
7	HTTP	80	50003	HTTP/1.1 200 OK	0.045
8	HTTP	50004	80	GET /downloads/	0.050
9	HTTP	50005	80	GET /favicon.ico	0.055
10	HTTP	80	50001	HTTP/1.1 200 OK	0.060
11	HTTP	50001	80	GET /script.js HTTP/	0.065
12	HTTP	80	50002	HTTP/1.1 302 Found (Redirect)	0.070
13	HTTP	50002	80	GET /dashboard	0.075
14	HTTP	80	50004	HTTP/1.1 200 OK	0.080
15	HTTP	50003	80	GET /api/updates	0.085
16	HTTP	80	50005	HTTP/1.1 404 Not	0.090
17	HTTP	80	50001	HTTP/1.1 200 OK	0.095

18	HTTP	50004	80	GET /downloads/	0.100
19	HTTP	80	50002	HTTP/1.1 200 OK (Dashboard)	0.110
20	HTTP	50005	80	GET /index.html	0.115
21	HTTP	80	50003	HTTP/1.1 200 OK (No Update)	0.120
22	HTTP	50002	80	GET /user/profile	0.125
23	HTTP	80	50004	HTTP/1.1 200 OK	0.130
24	HTTP	80	50005	HTTP/1.1 200 OK	0.135
25	HTTP	50001	80	GET /logout HTTP/	0.140
26	HTTP	80	50001	HTTP/1.1 200 OK (Logged out)	0.150
27	HTTP	50003	80	GET /api/logout	0.155
28	HTTP	80	50003	HTTP/1.1 200 OK	0.160
29	HTTP	80	50002	HTTP/1.1 200 OK (Profile Data)	0.170

## תיאור תהליך האריזה (Encapsulation):

תהליך האריזה הוא אבן היסוד של מודל ה-TCP/IP. בתהליך זה, המידע עובר מהשכבה העליונה (Application) כלפי מטה. בכל שלב, השכבה הנוכחית מתייחסת למידע שקיבלה מהשכבה שמעליה כאל Payload ומוסיפה לו כותרת (Header) המכילה נתוני בקרה הרלוונטיים לאותה שכבה. התוצאה הסופית היא "חבילה בתוך חבילה", המאפשרת לכל רכיב ברשת לקרוא רק את שכבת המידע הרלוונטית אליו. תהליך האריזה בוצע באמצעות מחברת Jupyter המדמה את פעולת ה-Network Stack במערכת ההפעלה. עבור כל שורה ב-CSV, בוצעו השלבים הבאים:

1. **Application Layer (L7):** קבלת הודעת ה-HTTP (לדוגמה: GET /index.html).
2. **Transport Layer (L4):** הוספת כותרת TCP. בשלב זה נקבעו מספרי הפורטים (למשל Port 80 כיעד) ונוספו דגלי בקרה (Flags) כגון PSH ו-Ack.
3. **Network Layer (L3):** הוספת כותרת IP (מחלקה IPv4). כאן מוגדרות כתובות ה-IP של המקור והיעד לצורך ניתוב לוגי.
4. **Data Link Layer (L2):** אריזת החבילה בתוך Ethernet Frame והוספת כתובות MAC פיזיות.

תיאור מפורט :

## קטע קוד האחראי על התעבורה ב wireshark:

```
for index, row in messages_df.iterrows():

    # Extract message details from the DataFrame row

    message = row['message']

    message = f"test message {index}" if not message else message

    transport.send(message.encode(), flags=0x18) # Example with PSH+ACK flags

    time.sleep(0.1) # Optional delay between messages
```

פונקציה זאת ניגשת לכל שורה בקובץ CSV ושולחת את המידע בקצב של 0.1 שניות, ובשימוש דגלים של PSH & ACK עבור ערך של 0x18.

הערך 0x18 הוא שילוב של שני דגלים: PSH (ערכו 8) ו-ACK (ערכו 16). השימוש ב-PSH (Push) מבטיח שהנתונים יועברו לאפליקציה מיד עם הגעתם ולא יישמרו בבאפר.

## פונקציה האחראית על השליחה:

### def send(self, data: bytes, flags: int=0x02):

הפונקציה מקבלת את המידע משכבת האפליקציה שאותו יש לשלוח מהקובץ CSV, בנוסף את הדגלי TCP ברירת מחדל זה 0x02 שזה בעצם SYN, מטרתו ליצור קשר (סינכרון) בפרוטוקול.

התנאי הראשון בודק באיזה מערכת הפעלה התוכנית פועלת ובהתאם למערכת הפעלה השידור מתבצע. אני משתמש ב Mac לכן אני פועל לפי התנאי הראשון:  
נוצרת חבילה באמצעות self. encapsulate(data, flags=flags) שמוסיפה ידנית את הביטים עבור : TCP Header & IP Header.

לאחר מכן קוראים ל self.sock.sendto(pkt, (self.dst\_ip, 0)) שאחראית לשליחת החבילה לכרטיס הרשת.

לאנשים שיש מערכת הפעלה Windows יפעלו לפי ה else הראשון מאחר ו- Windows חוסמת את raw sockets אז הם משתמשים בספריית Scapy לצורך בניית ה TCP & IP Stack.

## פונקציית האחראית על האריזה של המידע:

### def encapsulate(self, data: bytes, flags: int=0x02) -> bytes:

הפונקציה קוראת לשתי פונקציות : בניית הכותרת של TCP ובניית הכותרת של IP ומחזירה את המידע ארוז.

ה - IP header נמצא בראש החבילה (אחראי על כך שהנתבים ידעו לאן להעביר את החבילה ) לאחר מכן מופיע ה - TCP header (כדי שהמחשב ביעד ידע לאיזה פורט להעביר ולבדוק תקינות) ולאחר מכן ה - payload (המידע מהקובץ CSV).

## פונקציית האחראית על בניית כותרת ה-TCP:

**def build\_tcp\_header(src\_ip: str, dst\_ip: str, src\_port: int, dst\_port: int, payload: bytes=b'', seq: Optional[int]=None, ack\_seq: int=0, flags: int=0x02, window: int=65535) -> bytes:**

הפונקציה מקבלת IP Src & IP Dst & Ports כדי לדעת מאיפה ולכן לשלוח את החבילה, את ה-data כדי לחשב את ה-Checksum שמוכנס לתוך ה- כותרת ודגלים.

הפונקציה מתחילה בבדיקה האם קיים מספר סידורי (Sequence Number) לתעבורת המידע כדי לעקוב אחר החבילות. הקוד מייצר מספר רנדומלי בין 0 לבין  $2^{32}-1$ .

לאחר מכן מגדירים את אורך הכותרת שתהיה בגודל 20 בתים ובמיקום הנכון שתואם את הפרוטוקול של TCP.

שימוש ב-struct.pack עם התו ! מבטיח שהנתונים ייארזו בפורמט Big Endian כמקובל ברשתות. לחישוב בדיקת התקינות (Checksum), נעשה שימוש ב-Pseudo Header הכוללת נתונים משכבת ה-IP, מה שמבטיח הגנה מפני ניתוב שגוי של החבילה. לאחר מכן מחשבים pseudo\_header חישוב עזר עבור ה-checksum (בדיקת תקינות החבילה).

## פונקציה האחראית על בניית כותרת ה-IP:

**def build\_ip\_header(src\_ip: str, dst\_ip: str, payload\_len: int, proto:int=socket.IPPROTO\_TCP) -> bytes:**

הפונקציה מקבלת את ה-IP Src & IP Dst כדי שהנתב ידע לאן לנתב את המידע והפונקציה מקבלת את כל מה שבא אחריה (גם ה-TCP Header וגם ה-Data) נחשב כ-Payload.

הפעולה הראשונה בפונקציה קובע את גרסת ה-IP להיות IPv4 ואת גודל הכותרת של ה-IP בגודל 20 בתים.

לאחר מכן כמו ב-TCP בונים את הכותרת של ה-IP בכך שהופכים את המידע לביטים בגודל 20 בתים לפי הפרוטוקול של IP אך עם checksum 0.

לאחר מכן מריצים את החישוב checksum לכותרת ואז בונים את הכותרת שוב וזה מה שמחזירים.

## תיאור והסבר של תהליך הלכידה (Capture Process)

לאחר ביצוע אריזת הנתונים (Encapsulation) באמצעות הקוד, השלב הבא הוא "להאזין" לרשת ולתפוס את המנות בזמן שהן עוברות באמצעות Wireshark.

### 1. הגדרת ממשק הלכידה (Network Interface)

מאחר שהתקשורת התבצעה בין הלקוח לשרת על גבי אותו המחשב (מול הכתובת 127.0.0.1), החבילות אינן עוזבות את המחשב לכרטיס הרשת הפיזי (Wi-Fi או Ethernet).

- **הפעולה:** להאזין לממשק ה-Loopback (במערכות Mac כמו שלי נקרא lo0).
- **Loopback:** ממשק וירטואלי המאפשר למערכת ההפעלה להעביר נתונים בין תהליכים פנימיים.

### 2. סינון התעבורה (Traffic Filtering)

כדי לבודד את 30 ההודעות מתוך מאות החבילות האחרות שעוברות במערכת ההפעלה בכל רגע (כגון עדכוני תוכנה או שירותי מערכת), השתמשתי במסננים:

- **סינון לפי כתובת (ip.addr == 127.0.0.1):** מסנן זה וידא שנראה אך ורק תעבורה שנוצרה ונשלחה בתוך המחשב המקומי (Loopback), ובכך סינן תעבורת אינטרנט חיצונית.
- **סינון לפי פורט (tcp.port == 80):** מסנן זה השאיר מתוך תעבורת ה-Loopback רק את החבילות המשויות לפרוטוקול HTTP.
- **הסבר:** השילוב ביניהם (בביטוי `ip.addr == 127.0.0.1 && tcp.port == 80`) יצר סביבת עבודה מסוננת, שבה כל חבילה שנלכדה היא בוודאות חלק מהפרוייקט. פעולה זו אפשרה לראות את הדו-שיח המלא בצורה עוקבת ומסודרת: את בקשות ה-GET שנשלח ואת תגובות השרת.

### 3. תהליך ה-Decapsulation (פענוח המנות)

תהליך הלכידה מאפשר לנו לבצע את הפעולה ההפוכה לאריזה – Decapsulation. ה-Wireshark מקבל את רצף הביטים הגולמי (Hex) מהרשת ומפרק אותו חזרה לשכבות:

1. הוא מזהה את ה-Ethernet Frame (שכבה 2).
2. הוא "מקלף" את כותרת ה-IP (שכבה 3) ומציג לנו את כתובות המקור והיעד.
3. הוא "מקלף" את כותרת ה-TCP (שכבה 4) ומציג לנו את הפורטים והדגלים.
4. בסוף התהליך נחשף ה-Payload – הודעת ה-HTTP המקורית כפי שהופיעה בקובץ ה-CSV.

## תיאור התעבורה ב - Wireshark

ip.addr == 127.0.0.1 && tcp.port == 80							
No.	Time	Source	Destination	Protocol	Length	Info	
443	89.150941	127.0.0.1	127.0.0.1	TCP	88	50001 → 80 [PSH, ACK] Seq=1 Ack=1 Win=65535 Len=24	
456	89.255718	127.0.0.1	127.0.0.1	TCP	88	50002 → 80 [PSH, ACK] Seq=1 Ack=1 Win=65535 Len=24	
457	89.361682	127.0.0.1	127.0.0.1	TCP	88	50003 → 80 [PSH, ACK] Seq=1 Ack=1 Win=65535 Len=24	
458	89.467707	127.0.0.1	127.0.0.1	TCP	87	80 → 50001 [PSH, ACK] Seq=3031137395 Ack=3277438791 Win=65535 Len=23	
459	89.571225	127.0.0.1	127.0.0.1	TCP	87	[TCP ACKed unseen segment] [TCP Retransmission] 50001 → 80 [PSH, ACK] Seq=3500837797 Ack=1 Win=65535 Len=23	
460	89.677304	127.0.0.1	127.0.0.1	TCP	92	[TCP ACKed unseen segment] 80 → 50002 [PSH, ACK] Seq=2164000912 Ack=559638271 Win=65535 Len=28	
461	89.783294	127.0.0.1	127.0.0.1	TCP	94	[TCP ACKed unseen segment] [TCP Retransmission] 50002 → 80 [PSH, ACK] Seq=2641863961 Ack=1 Win=65535 Len=30	
462	89.889447	127.0.0.1	127.0.0.1	TCP	88	[TCP ACKed unseen segment] 80 → 50003 [PSH, ACK] Seq=2754845582 Ack=1440852898 Win=65535 Len=24	
463	89.995300	127.0.0.1	127.0.0.1	TCP	89	50004 → 80 [PSH, ACK] Seq=1 Ack=1 Win=65535 Len=25	
464	90.101020	127.0.0.1	127.0.0.1	TCP	89	50005 → 80 [PSH, ACK] Seq=1 Ack=1 Win=65535 Len=25	
471	90.204280	127.0.0.1	127.0.0.1	TCP	85	[TCP Retransmission] 80 → 50001 [PSH, ACK] Seq=1290733062 Ack=3277438791 Win=65535 Len=21	
472	90.306462	127.0.0.1	127.0.0.1	TCP	87	[TCP ACKed unseen segment] [TCP Retransmission] 50001 → 80 [PSH, ACK] Seq=4174739075 Ack=1 Win=65535 Len=23	
473	90.410524	127.0.0.1	127.0.0.1	TCP	93	[TCP ACKed unseen segment] [TCP Previous segment not captured] 80 → 50002 [PSH, ACK] Seq=3859986207 Ack=559638271 Win=65535 Len=29	
474	90.514920	127.0.0.1	127.0.0.1	TCP	87	[TCP ACKed unseen segment] [TCP Retransmission] 50002 → 80 [PSH, ACK] Seq=4153672459 Ack=1 Win=65535 Len=23	
475	90.618347	127.0.0.1	127.0.0.1	TCP	88	[TCP ACKed unseen segment] 80 → 50004 [PSH, ACK] Seq=286331188 Ack=899502297 Win=65535 Len=24	
476	90.723263	127.0.0.1	127.0.0.1	TCP	89	[TCP ACKed unseen segment] [TCP Retransmission] 50003 → 80 [PSH, ACK] Seq=2438689089 Ack=1 Win=65535 Len=25	
493	90.820853	127.0.0.1	127.0.0.1	TCP	86	80 → 50005 [PSH, ACK] Seq=3769306679 Ack=3945313040 Win=65535 Len=22	
494	90.930802	127.0.0.1	127.0.0.1	TCP	84	[TCP Previous segment not captured] 80 → 50001 [PSH, ACK] Seq=352400798 Ack=3277438791 Win=65535 Len=20	
495	91.035171	127.0.0.1	127.0.0.1	TCP	88	[TCP Retransmission] 50004 → 80 [PSH, ACK] Seq=3224759170 Ack=1 Win=65535 Len=24	
496	91.141125	127.0.0.1	127.0.0.1	TCP	91	[TCP ACKed unseen segment] [TCP Retransmission] 80 → 50002 [PSH, ACK] Seq=22807707425 Ack=559638271 Win=65535 Len=27	
497	91.243754	127.0.0.1	127.0.0.1	TCP	88	[TCP ACKed unseen segment] [TCP Previous segment not captured] 50005 → 80 [PSH, ACK] Seq=900270520 Ack=1 Win=65535 Len=24	
498	91.346939	127.0.0.1	127.0.0.1	TCP	91	[TCP ACKed unseen segment] [TCP Previous segment not captured] 80 → 50003 [PSH, ACK] Seq=2807957298 Ack=1440852898 Win=65535 Len=27	
499	91.451144	127.0.0.1	127.0.0.1	TCP	90	[TCP ACKed unseen segment] [TCP Retransmission] 50002 → 80 [PSH, ACK] Seq=2593121671 Ack=1 Win=65535 Len=26	
500	91.555415	127.0.0.1	127.0.0.1	TCP	79	[TCP ACKed unseen segment] [TCP Retransmission] 80 → 50004 [PSH, ACK] Seq=2607668501 Ack=899502297 Win=65535 Len=15	
501	91.660438	127.0.0.1	127.0.0.1	TCP	79	[TCP Previous segment not captured] 80 → 50005 [PSH, ACK] Seq=663216124 Ack=3945313040 Win=65535 Len=15	
502	91.763144	127.0.0.1	127.0.0.1	TCP	84	[TCP Previous segment not captured] 50001 → 80 [PSH, ACK] Seq=481655279 Ack=1 Win=65535 Len=20	
503	91.868759	127.0.0.1	127.0.0.1	TCP	92	[TCP Previous segment not captured] 80 → 50001 [PSH, ACK] Seq=1221150086 Ack=3277438791 Win=65535 Len=28	
504	91.971837	127.0.0.1	127.0.0.1	TCP	88	[TCP ACKed unseen segment] [TCP Previous segment not captured] 50003 → 80 [PSH, ACK] Seq=212334184 Ack=1 Win=65535 Len=24	
505	92.077671	127.0.0.1	127.0.0.1	TCP	79	[TCP ACKed unseen segment] [TCP Retransmission] 80 → 50003 [PSH, ACK] Seq=666754239 Ack=1440852898 Win=65535 Len=15	
506	92.181561	127.0.0.1	127.0.0.1	TCP	94	[TCP ACKed unseen segment] [TCP Retransmission] 80 → 50002 [PSH, ACK] Seq=2165410508 Ack=559638271 Win=65535 Len=30	

בצילום המסך המצורף, ניתן לראות את לכידת התעבורה שנוצרה על ידי הקוד. הנקודות המרכזיות העולות מהניתוח:

- זיהוי כתובות ה-IP: ניתן לראות כי התעבורה מתבצעת על כתובת ה-(127.0.0.1) Loopback, מה שמעיד על שימוש ב-Interface הפנימי במחשב לצורך הבדיקה.
- זיהוי הפורטים (Port Numbers):

- Source Port: הלקוח משתמש בפורטים 50001-50005, כפי שהוגדר ב - CSV.
- Destination Port: היעד הוא פורט 80 (Default HTTP Port), כפי שהוגדר ב-CSV.

השימוש בפורטים שונים עבור אותו ה-IP מדגים את יכולת ה-Multiplexing של שכבת התעבורה, המאפשרת למכשיר הקצה לנהל מספר שיחות (Sessions) במקביל מול אותו שרת, תוך הפרדה מוחלטת ביניהן.

- שימוש בדגלים (Flags): כפי שניתן לראות בעמודת ה-Info, כל החבילות מסומנות ב-[PSH, ACK]. זהו תרגום ישיר של הערך 0x18 שהוגדר בקוד (0x08 עבור PSH ו-0x10 עבור ACK). ה-PSH מבטיח שהמידע יידחף לאפליקציה באופן מיידי.
- ניתוח ה-Sequence Number (Seq): ניתן לראות שבכל שורה מופיע מספר Seq גבוה ושונה (למשל Seq= 3031137395). אלו הם המספרים הרנדומליים שיצרה הפונקציה build\_tcp\_header כדי לנהל את המעקב אחר החבילות.
- אורך הנתונים (Len): ניתן לראות ערכים משתנים של Len (כמו 23, 24, 28). אלו הם אורכי מחרוזות הטקסט השונות שנלקחו מה-CSV ונשלחו כ-Payload.

כעת נצלול פנימה עבור שידור אחד של port 50001 לפורט 80 וקבלת תשובה מ- port 80 ל-port 50001 תוך הסתכלות בקובץ CSV בהשוואה ל - Wireshark.

## הודעת CSV:

שידור של port 50001 לפי קובץ CSV: HTTP/1.1,0.01 GET /index.html HTTP,50001,80.

ניתוח מרכיבי ההודעה:

- HTTP (Protocol): מציין שזוהי תעבורה מסוג Web. למרות שאנחנו בונים את החבילה בתוך TCP ו-IP, התוכן עצמו שייך לפרוטוקול השכבה השביעית (Application Layer).
- 50001 (Source Port): זהו הפורט הלוגי של הלקוח שיוזם את הקשר.
- 80 (Destination Port): פורט היעד הסטנדרטי לשרתי HTTP. זה מבטיח שהחבילה תגיע ל"שירות האינטרנט" בשרת ולא לאפליקציה אחרת.
- GET /index.html HTTP/1.1 (Message Payload): הלקוח אומר לשרת: "אני רוצה לקבל (GET) את הקובץ הראשי של האתר (index.html) לפי הסטנדרט של גרסה 1.1".
- 0.01 (Timestamp): הזמן היחסי מתחילת הניסוי שבו ההודעה נשלחה.

## הודעת Wireshark:

### Request from port 50001 to port 80:

443	89.150941	127.0.0.1	127.0.0.1	TCP	88	50001 → 80	[PSH, ACK] Seq=1 Ack=1 Win=65535 Len=24
-----	-----------	-----------	-----------	-----	----	------------	---

בתמונה זו ניתן לראות את שורת הסיכום של חבילה 443. התמונה מדגימה כיצד ה-Wireshark מפענח את כל הנתונים שהוזנו ידנית לקוד: פורט מקור 50001, דגלי PSH/ACK, ואורך הודעה של 24 בתים. זהו אישור ויזואלי לכך שה-Encapsulation הידני יצר חבילה חוקית לחלוטין לפי סטנדרט ה-TCP/IP.

שכבת היישום (application layer):

0000	02 00 00 00 45 00 00 54	5f 53 00 00 40 ff 00 00	...E..T..S..@...
0010	7f 00 00 01 7f 00 00 01	45 00 00 40 bc e9 00 00	.....E..@.....
0020	40 06 bf cc 7f 00 00 01	7f 00 00 01 c3 51 00 50	@.....Q.P
0030	3c a6 40 ba 00 00 00 00	50 18 ff ff c9 07 00 00	<@.....P.....
0040	47 45 54 20 2f 69 6e 64	65 78 2e 68 74 6d 6c 20	GET /index.html
0050	48 54 54 50 2f 31 2e 31		HTTP/1.1

בתמונה זו ניתן לראות את ה-Payload של שכבת ה-Application:

מציג את ה-Payload כפי שהוא נראה על הרשת:

ניתן לראות את הטקסט מה-HTTP/1.1 GET /index.html CSV.

## שכבת התעבורה (Transport Layer - TCP):

```
Transmission Control Protocol, Src Port: 50001, Dst Port: 80, Seq: 1, Ack: 1, Len: 24
Source Port: 50001
Destination Port: 80
[Stream index: 10]
[Stream Packet Number: 1]
> [Conversation completeness: Incomplete (8)]
[TCP Segment Len: 24]
Sequence Number: 1 (relative sequence number)
Sequence Number (raw): 1017528506
[Next Sequence Number: 25 (relative sequence number)]
Acknowledgment Number: 1 (relative ack number)
Acknowledgment number (raw): 0
0101 .... = Header Length: 20 bytes (5)
v Flags: 0x018 (PSH, ACK)
  000. .... = Reserved: Not set
  ...0 .... = Accurate ECN: Not set
  .... 0... = Congestion Window Reduced: Not set
  .... .0.. = ECN-Echo: Not set
  .... ..0. = Urgent: Not set
  .... ...1 = Acknowledgment: Set
  .... .... 1... = Push: Set
  .... .... .0.. = Reset: Not set
  .... .... ..0. = Syn: Not set
  .... .... ...0 = Fin: Not set
  [TCP Flags: .....AP...]
Window: 65535
[Calculated window size: 65535]
[Window size scaling factor: -1 (unknown)]
Checksum: 0xc907 [unverified]
[Checksum Status: Unverified]
Urgent Pointer: 0
v [Timestamps]
  [Time since first frame in this TCP stream: 0.000000000 seconds]
  [Time since previous frame in this TCP stream: 0.000000000 seconds]
[Client Contiguous Streams: 4]
[Server Contiguous Streams: 4]
TCP payload (24 bytes)
TCP segment data (24 bytes)
```

- Ports: פורט מקור 50001 ויעד 80.
- Sequence Numbers: מוצג ה-Sequence Number הגולמי (Raw): 1017528506.
- Header Length: מופיע הערך 20 (5 bytes). זהו בדיוק ה-5 >> 4 מהקוד, המייצג 5 מילים של 32 ביט.
- Flags: מופיע (PSH, ACK) 0x018. ה-Wireshark מפרק זאת ומראה שביט ה-Push וביט ה-Acknowledgment מאופשרים, בדיוק כפי שהוגדר בקוד 0x18.



## שכבת הרשת (IPv4 - Network Layer):

```
✓ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 64
    Identification: 0xbce9 (48361)
  > 000. .... = Flags: 0x0
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 64
    Protocol: TCP (6)
    Header Checksum: 0xbfcc [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 127.0.0.1
    Destination Address: 127.0.0.1
    [Stream index: 0]
```

- Version & IHL: מופיע גרסה 4 ואורך כותרת 20 bytes. זהו המימוש של ה-version\_ihl מהקוד.
- Identification: מופיע מזהה ייחודי (48361) 0xbce9, המאפשר למערכת ההפעלה לעקוב אחר החבילה.
- Protocol: מופיע (6) TCP. זהו השדה שמורה לשכבת ה-IP ש-Payload שהיא נושאת הוא מסוג TCP.
- Addresses: כתובות המקור והיעד הן 127.0.0.1.

## response from port 80 to port 50001:

458 89.467707 127.0.0.1 127.0.0.1 TCP 87 80 → 50001 [PSH, ACK] Seq=3031137395 Ack=3277438791 Win=65535 Len=23

בתמונה ניתן לראות את שורת הסיכום של חבילה 458. זוהי חבילת ה-TCP שנשלחה מהשרת חזרה למחשב. פורט מקור 80, דגלי PSH/ACK, ואורך הודעה של 23 בתים. ה-ACK מאשר את קבלת ה-GET, וה-PSH מצוין העברת נתונים מיידית.

0000	02 00 00 00 45 00 00 53	c2 a7 00 00 40 ff 00 00	....E..S....@...
0010	7f 00 00 01 7f 00 00 01	45 00 00 3f 24 ea 00 00	.....E..?\$....
0020	40 06 57 cd 7f 00 00 01	7f 00 00 01 00 50 c3 51	@.W.....P.Q
0030	b4 ab 7c 72 00 00 00 00	50 18 ff ff e1 f4 00 00	.. r....P.....
0040	48 54 54 50 2f 31 2e 31	20 32 30 30 20 4f 4b 20	HTTP/1.1 200 OK
0050	28 49 6e 64 65 78 29		(Index)

בתמונה זו ניתן לראות את ה-Payload של שכבת ה-Application:

מציג את ה-Payload כפי שהוא נראה על הרשת:

ניתן לראות בבירור את הטקסט מה-CSV: HTTP/1.1 200 OK (Index).

תגובה המציינת שהבקשה ל-index.html הצליחה והשרת מחזיר את התוכן.

```
Transmission Control Protocol, Src Port: 80, Dst Port: 50001, Seq: 3031137395, Ack: 3277438791, Len: 23
Source Port: 80
Destination Port: 50001
[Stream index: 10]
[Stream Packet Number: 2]
> [Conversation completeness: Incomplete (8)]
[TCP Segment Len: 23]
Sequence Number: 3031137395 (relative sequence number)
Sequence Number (raw): 3031137394
[Next Sequence Number: 3031137418 (relative sequence number)]
Acknowledgment Number: 3277438791 (relative ack number)
Acknowledgment number (raw): 0
0101 .... = Header Length: 20 bytes (5)
Flags: 0x018 (PSH, ACK)
  000. .... = Reserved: Not set
  ...0 .... = Accurate ECN: Not set
  ....0... = Congestion Window Reduced: Not set
  ....0... = ECN-Echo: Not set
  ....0... = Urgent: Not set
  ....1... = Acknowledgment: Set
  ....1... = Push: Set
  ....0... = Reset: Not set
  ....0... = Syn: Not set
  ....0... = Fin: Not set
[TCP Flags: .....AP...]
Window: 65535
[Calculated window size: 65535]
[Window size scaling factor: -1 (unknown)]
Checksum: 0xe1f4 [unverified]
[Checksum Status: Unverified]
Urgent Pointer: 0
[Timestamps]
  [Time since first frame in this TCP stream: 316.766000 milliseconds]
  [Time since previous frame in this TCP stream: 316.766000 milliseconds]
> [SEQ/ACK analysis]
  [Client Contiguous Streams: 4]
  [Server Contiguous Streams: 4]
  TCP payload (23 bytes)
  TCP segment data (23 bytes)
```

- Ports: פורט מקור 80 ויעד 50001.
- Sequence Numbers: מוצג ה-Sequence Number הגולמי (Raw): 3031137395.
- Header Length: מופיע הערך 20 (5 bytes). זהו בדיוק ה-5 >> 4, המייצג 5 מילים של 32 ביט.
- Flags: מופיע (PSH, ACK) 0x018. ה-Wireshark מפרק זאת ומראה שביט ה-Push וביט ה-Acknowledgment מאופשרים, בדיוק לפי הגדרת ה-0x18.

```

v Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
> Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 63
  Identification: 0x24ea (9450)
> 000. .... = Flags: 0x0
  ...0 0000 0000 0000 = Fragment Offset: 0
  Time to Live: 64
  Protocol: TCP (6)
  Header Checksum: 0x57cd [validation disabled]
  [Header checksum status: Unverified]
  Source Address: 127.0.0.1
  Destination Address: 127.0.0.1
  [Stream index: 0]

```

- Version & IHL: מופיע גרסה 4 ואורך כותרת 20 bytes. זהו המימוש של ה-version\_ihl מהקוד.
- Identification: מופיע מזהה ייחודי (9450) 0x24ea, המאפשר למערכת ההפעלה לעקוב אחר החבילה.
- Protocol: מופיע (6) TCP. זהו השדה שמורה לשכבת ה-IP ש-Payload שיהא נושאת הוא מסוג TCP.
- Addresses: כתובות המקור והיעד הן 127.0.0.1.

## חלק 2 – יצירת יישום וניתוח תעבורה

### תיאור כללי:

המערכת היא אפליקציית צ'אט רב-משתמשים המבוססת על ארכיטקטורת לקוח-שרת (Client-Server). המערכת מאפשרת למספר משתמשים לתקשר זה עם זה בזמן אמת דרך שרת מרכזי. התקשורת מתבצעת על גבי פרוטוקול TCP/IP, המבטיח העברת נתונים אמינה, ללא איבוד מידע ובסדר הנכון. כל הודעה מנותבת ספציפית ליעד שלה (Unicast) על פי לוגיקת הניתוב של השרת.

### רכיבי המערכת:

הפרויקט מחולק לשלושה מודולים עיקריים:

1. השרת (server.py): משמש כמקום בקרה של המערכת. הוא מנהל את חיבורי הלקוחות, מחזיק רישום של המשתמשים הפעילים ומנתב הודעות לפי הפרוטוקול שקבעתי (שם\_יעד:תוכן\_ההודעה).
2. הלקוח (client.py): ממשק המשתמש. הוא מתחבר לשרת, מזדהה מולו ומאפשר למשתמש לשלוח ולקבל הודעות בזמן-אמת.
3. סקריפט הסימולציה (main.py): כלי שנועד להריץ את כל הסביבה (שרת ושלושה לקוחות) בחלונות טרמינל נפרדים בלחיצת כפתור אחת, מה שמקל מאוד על תהליך הבדיקה (מיועד ל-macOS).

### ארכיטקטורת הקוד: תכנות מונחה עצמים (OOP):

בפרויקט השתמשתי בשתי מחלקות:

מחלקה **ChatServer**: מרכזת בתוכה את הסוקט של השרת, את מסד הנתונים של הלקוחות (Dictionary) ואת לוגיקת הריבוי-תהליכונים.

מחלקה **ChatClient**: מרכזת את מצב החיבור, את תהליכון הקבלה ואת ממשק המשתמש.

### עבודה במקביל (multi threading):

כדי לטפל במספר משתמשים בו-זמנית, המערכת משתמשת ב-Threads:

- **בשרת**: כל חיבור חדש מקבל "תהליכון" ייעודי. זה מונע ממצב שבו משתמש אחד "חוסם" את השרת בזמן שהוא מקליד או מחכה למידע.
- **בלקוח**: רצים שני תהליכונים במקביל:
  - התהליכון הראשי: מטפל בקלט מהמקלדת (input).
  - תהליכון הרקע: מאזין כל הזמן למידע שנכנס מהסוקט (recv) ומציג אותו מיד.

### מבנה בסיס הנתונים:

**מבנה המילון**: { "username": socket\_object }

- **המפתח (Key)**: מחרוזת (String) המייצגת את שם המשתמש הייחודי שהוזן בעת החיבור.
- **הערך (Value)**: אובייקט ה-Socket המשוויון לאותו לקוח.

השימוש במילון מאפשר שליפה וניתוב הודעות בזמן של  $O(1)$  (זמן קבוע), מה שמבטיח ביצועים מהירים גם כאשר מספר המשתמשים גדל. המילון מתעדכן דינמית בכל חיבור (connect) או ניתוק (disconnect).

## הוראות התקנה והרצה:

כדי לבחון את המערכת ולהריץ את הקוד, יש לעקוב אחר השלבים הבאים:

1. יש לפתוח טרמינל ולהוריד את הפרויקט למחשב המקומי בעזרת הפקודה:

```
git clone https://github.com/r0nik123/computer_networking_project.git
cd computer_networking_project
```

### 2. דרישות מערכת

- **Python 3.x:** המערכת נכתבה ב-Python 3 ומשתמשת בספריות מובנות בלבד (socket, threading, os, subprocess).
- **מערכת הפעלה:** סקריפט הסימולציה הותאם למערכות **macOS** (עושה שימוש ב-AppleScript לפתיחת חלונות טרמינל). להרצה במערכות אחרות (Windows/Linux), יש להפעיל את הקבצים ידנית כפי שיוסבר בהמשך.

3. **הרצת הסימולציה האוטומטית** על מנת לראות את המערכת בפעולה מלאה הכוללת שרת ושלושה לקוחות בו-זמנית, יש להריץ את הקובץ:

```
python3 main.py
```

- נפתח חלון טרמינל עבור השרת (server.py).
- המערכת ממתינה 2 שניות לאתחול ה-Socket של השרת.
- נפתחים 3 חלונות טרמינל נוספים עבור הלקוחות: יוסי (Yossi), דנה (Dana), ורוני (Roni).

### 4. הפעלה ידנית:

1. **הפעלת השרת:** `python3 server.py`

2. **הפעלת לקוח:** `python3 client.py <YourName>` (לדוגמה: `python3 client.py Dana`).

5. **שימוש בצ'אט (Format)** לאחר שכל החלונות פתוחים, ניתן לשלוח הודעות בין הלקוחות. הפרוטוקול מחייב ציון שם הנמען ולאחריו נקודתיים:

- **דוגמה:** בחלון של יוסי נקליד: `Dana: Hello Dana, how are you?`
- דנה תקבל מיד את ההודעה בעיצוב הבא: `[RECEIVE][Yossi]: Hello Dana, how are you?`
- לסגירת הלקוח, יש להקליד את המילה `quit`.

## דוגמאות קלט פלט:

```
project/server_clients/server.py
[SERVER] Listening on 127.0.0.1:8888...
[SERVER] Yossi connected from ('127.0.0.1', 63441)
[SERVER] Dana connected from ('127.0.0.1', 63442)
[SERVER] Roni connected from ('127.0.0.1', 63443)
```

```
--- Welcome to the chat, Yossi! ---    --- Welcome to the chat, Dana! ---    project/server_clients/client.py Roni
Send [Target:Message] or 'quit': █      Send [Target:Message] or 'quit': █      --- Welcome to the chat, Roni! ---
Send [Target:Message] or 'quit': █      Send [Target:Message] or 'quit': █      Send [Target:Message] or 'quit': █
```

ייזום השרת (Server Startup): בחלון השרת מופיעה ההודעה [SERVER] Listening on 127.0.0.1:8888.

זה מעיד על כך שהסוקט הראשי נוצר בהצלחה, עבר תהליך של Binding לפורט המבוקש ונמצא כעת במצב Listening.

ניתן לראות בלוגים של השרת שלוש שורות עוקבות של connected from...

- מאחורי הקלעים: כל לקוח (יוסי, דנה ורוני) ביצע תהליך TCP Three-Way Handshake מול השרת.
- הזדהות: מיד לאחר החיבור, כל לקוח שלח את שמו כחבילת המידע הראשונה. השרת קלט את השם ועדכן את ה-המילון הפנימי שלו. זהו שלב ה"זיהוי" שמאפשר ניתוב הודעות עתידיות.

בחלונות הלקוחות מופיעה ההנחיה: Send [Target:Message] or 'quit'.

- מצב אסינכרוני: בשלב זה, בכל לקוח כבר רץ תהליכון רקע (receive\_messages) הממתין על פקודת recv. הלקוח נמצא במצב של "האזנה כפולה" – הוא גם מוכן לקבל קלט מהמשתמש (Main Thread) וגם מוכן להציג הודעות שיגיעו מהשרת בכל רגע.

זה מה שרואים ב - Wireshark:

1	0.000000	127.0.0.1	127.0.0.1	TCP	68	63689 → 8888 [SYN] Seq=0 Win=65535 Len=0 MSS=16344 WS=64 TSval=3119097404 TSecr=0 SACK_PERM
2	0.000100	127.0.0.1	127.0.0.1	TCP	68	8888 → 63689 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=16344 WS=64 TSval=574648336 TSecr=3119097404 SACK_PERM
3	0.000120	127.0.0.1	127.0.0.1	TCP	56	63689 → 8888 [ACK] Seq=1 Ack=1 Win=408320 Len=0 TSval=3119097404 TSecr=574648336
4	0.000130	127.0.0.1	127.0.0.1	TCP	56	[TCP Window Update] 8888 → 63689 [ACK] Seq=1 Ack=1 Win=408320 Len=0 TSval=574648336 TSecr=3119097404
5	0.000149	127.0.0.1	127.0.0.1	TCP	61	63689 → 8888 [PSH, ACK] Seq=1 Ack=1 Win=408320 Len=5 TSval=3119097404 TSecr=574648336
6	0.000167	127.0.0.1	127.0.0.1	TCP	56	8888 → 63689 [ACK] Seq=1 Ack=6 Win=408320 Len=0 TSval=574648336 TSecr=3119097404
7	1.010205	127.0.0.1	127.0.0.1	TCP	68	63690 → 8888 [SYN] Seq=0 Win=65535 Len=0 MSS=16344 WS=64 TSval=1610830536 TSecr=0 SACK_PERM
8	1.010304	127.0.0.1	127.0.0.1	TCP	68	8888 → 63690 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=16344 WS=64 TSval=210469459 TSecr=1610830536 SACK_PERM
9	1.010326	127.0.0.1	127.0.0.1	TCP	56	63690 → 8888 [ACK] Seq=1 Ack=1 Win=408320 Len=0 TSval=1610830536 TSecr=210469459
10	1.010336	127.0.0.1	127.0.0.1	TCP	56	[TCP Window Update] 8888 → 63690 [ACK] Seq=1 Ack=1 Win=408320 Len=0 TSval=210469459 TSecr=1610830536
11	1.010347	127.0.0.1	127.0.0.1	TCP	60	63690 → 8888 [PSH, ACK] Seq=1 Ack=1 Win=408320 Len=4 TSval=1610830536 TSecr=210469459
12	1.010364	127.0.0.1	127.0.0.1	TCP	56	8888 → 63690 [ACK] Seq=1 Ack=5 Win=408320 Len=0 TSval=210469459 TSecr=1610830536
13	2.013131	127.0.0.1	127.0.0.1	TCP	68	63691 → 8888 [SYN] Seq=0 Win=65535 Len=0 MSS=16344 WS=64 TSval=2353658006 TSecr=0 SACK_PERM
14	2.013253	127.0.0.1	127.0.0.1	TCP	68	8888 → 63691 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=16344 WS=64 TSval=1097229661 TSecr=2353658006 SACK_PERM
15	2.013284	127.0.0.1	127.0.0.1	TCP	56	63691 → 8888 [ACK] Seq=1 Ack=1 Win=408320 Len=0 TSval=2353658006 TSecr=1097229661
16	2.013298	127.0.0.1	127.0.0.1	TCP	56	[TCP Window Update] 8888 → 63691 [ACK] Seq=1 Ack=1 Win=408320 Len=0 TSval=1097229661 TSecr=2353658006
17	2.013303	127.0.0.1	127.0.0.1	TCP	60	63691 → 8888 [PSH, ACK] Seq=1 Ack=1 Win=408320 Len=4 TSval=2353658006 TSecr=1097229661
18	2.013314	127.0.0.1	127.0.0.1	TCP	56	8888 → 63691 [ACK] Seq=1 Ack=5 Win=408320 Len=0 TSval=1097229661 TSecr=2353658006

בשלוש השורות הראשונות מתבצע תהליך ה- tcp three way handshake .

בשורה מספר 4 השרת שולח אישור על פתיחת החיבור.

בשורה מספר 5 הלקוח שולח ack על העדכון של השרת ושולח לו את השם של הלקוח(בדוגמא זה yossi).

בשורה מספר 6 השרת שולח אישור שהוא קיבל את השם.

כך מתבצע בשאר הלקוחות.

## דוגמת תקשורת בין שני לקוחות: רוני ודנה:

```
--- Welcome to the chat, Dana! ---  
Send [Target:Message] or 'quit': Roni: hii roni  
Send [Target:Message] or 'quit':  
  
[RECEIVE] [Roni]: hii dana  
Send [Target:Message] or 'quit': █
```

```
project/server_clients/client.py Roni  
--- Welcome to the chat, Roni! ---  
Send [Target:Message] or 'quit':  
  
[RECEIVE] [Dana]: hii roni  
Send [Target:Message] or 'quit': Dana: hii dana  
Send [Target:Message] or 'quit': █
```

ניתן לראות כי דנה יזמה את התקשורת ביניהם, רוני קיבל את ההודעה והחזיר לדנה גם הודעה שהתקבלה אצל דנה.

מה שרואים ב- Wireshark:

81	844.448971	127.0.0.1	127.0.0.1	TCP	70	63690 → 8888	[PSH, ACK] Seq=5 Ack=1 Win=408320 Len=14 TSval=1611673985 TSecr=210469459
82	844.449051	127.0.0.1	127.0.0.1	TCP	56	8888 → 63690	[ACK] Seq=1 Ack=19 Win=408320 Len=0 TSval=211312908 TSecr=1611673985
83	844.449436	127.0.0.1	127.0.0.1	TCP	73	8888 → 63691	[PSH, ACK] Seq=1 Ack=5 Win=408320 Len=17 TSval=1098072107 TSecr=2353658006
84	844.449476	127.0.0.1	127.0.0.1	TCP	56	63691 → 8888	[ACK] Seq=5 Ack=18 Win=408320 Len=0 TSval=2354500452 TSecr=1098072107
86	855.816054	127.0.0.1	127.0.0.1	TCP	70	63691 → 8888	[PSH, ACK] Seq=5 Ack=18 Win=408320 Len=14 TSval=2354511819 TSecr=1098072107
87	855.816106	127.0.0.1	127.0.0.1	TCP	56	8888 → 63691	[ACK] Seq=18 Ack=19 Win=408320 Len=0 TSval=1098083474 TSecr=2354511819
88	855.816286	127.0.0.1	127.0.0.1	TCP	73	8888 → 63690	[PSH, ACK] Seq=1 Ack=19 Win=408320 Len=17 TSval=211324275 TSecr=1611673985
89	855.816301	127.0.0.1	127.0.0.1	TCP	56	63690 → 8888	[ACK] Seq=19 Ack=18 Win=408320 Len=0 TSval=1611685352 TSecr=211324275

נדבר על התקשורת הראשונה מדנה לרוני כי זה חוזר על עצמו:

המידע הוא: Roni: hii roni:

שכבת היישום (application layer):

0000	02 00 00 00 45 00 00 42	00 00 40 00 40 06 00 00	...E..B..@..@..
0010	7f 00 00 01 7f 00 00 01	f8 ca 22 b8 9e 83 27 42	....."...'B
0020	92 b1 06 2d 80 18 18 ec	fe 36 00 00 01 01 08 0a	...-.....6.....
0030	60 10 31 81 0c 8b 82 53	52 6f 6e 69 3a 20 68 69	`1...S Roni: hi
0040	69 20 72 6f 6e 69		i roni

זוהי השכבה העליונה המכילה את התוכן שהמשתמש יצר:

- Payload: בתצוגת ה-Hex וה-ASCII ניתן לראות בבירור את הטקסט: "Roni: hii roni".
- Length: אורך הנתונים הוא 14 בתים, התואם בדיוק למספר התווים בהודעה (כולל רווחים וסימני פיסוק). זהו המידע שמועבר לפונקציית ה-recv() בקוד השרת שלנו.

## שכבת התעבורה (Transport Layer - TCP):

```
Transmission Control Protocol, Src Port: 63690, Dst Port: 8888, Seq: 5, Ack: 1, Len: 14
Source Port: 63690
Destination Port: 8888
[Stream index: 1]
[Stream Packet Number: 7]
> [Conversation completeness: Incomplete, DATA (15)]
[TCP Segment Len: 14]
Sequence Number: 5 (relative sequence number)
Sequence Number (raw): 2659395394
[Next Sequence Number: 19 (relative sequence number)]
Acknowledgment Number: 1 (relative ack number)
Acknowledgment number (raw): 2461074989
1000 .... = Header Length: 32 bytes (8)
> Flags: 0x018 (PSH, ACK)
Window: 6380
[Calculated window size: 408320]
[Window size scaling factor: 64]
Checksum: 0xfe36 [unverified]
[Checksum Status: Unverified]
Urgent Pointer: 0
> Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
> [Timestamps]
> [SEQ/ACK analysis]
[Client Contiguous Streams: 1]
[Server Contiguous Streams: 1]
TCP payload (14 bytes)
```

שכבה זו אחראית על ניהול השיחה (Session) ואמינות ההעברה:

- Ports: ניתן לראות את פורט המקור הדינמי 63690 ואת פורט היעד הקבוע של השרת 8888.
- Flags: מופיע הדגל 0x18 (PSH, ACK). ה-PSH (Push) קריטי ליישום צ'אט, כיוון שהוא מורה למערכת ההפעלה להעביר את המידע מיד ליישום ולא להמתין בבאפר.
- Sequence Number: ניתן לראות את המספר הסידורי (Seq=5) המאפשר ל-TCP לעקוב אחר סדר החבילות ולוודא שלא אבד מידע.

## שכבת הרשת (Network Layer - IPv4)

```
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
0100 .... = Version: 4
.... 0101 = Header Length: 20 bytes (5)
> Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
Total Length: 66
Identification: 0x0000 (0)
> 010. .... = Flags: 0x2, Don't fragment
...0 0000 0000 0000 = Fragment Offset: 0
Time to Live: 64
Protocol: TCP (6)
Header Checksum: 0x0000 [validation disabled]
[Header checksum status: Unverified]
Source Address: 127.0.0.1
Destination Address: 127.0.0.1
[Stream index: 0]
```

בשכבה זו מתבצע הניתוב הלוגי בין המארחים:

- Version: ניתן לראות שימוש ב-IPv4 (גרסה 4).
- Addresses: כתובת המקור והיעד הן 127.0.0.1 (Loopback), מה שמעיד על תקשורת פנימית במחשב.



- Protocol: מופיע הערך (6) TCP. זהו השדה שמורה לשכבת הרשת שהיא נושאת חבילה שייכת לפרוטוקול TCP בשכבה שמעליה.
- TTL (Time to Live): הערך הוא 64, המייצג את מספר ה"קפיצות" המקסימלי שהחבילה יכולה לעבור ברשת.

```
82 844.449051 127.0.0.1 127.0.0.1 TCP 56 8888 → 63690 [ACK] Seq=1 Ack=19 Win=408320 Len=0 TSval=211312908 TSecr=1611673985
```

שורה 82 ניתן לראות את חבילת ה-ACK שנשלחה מהשרת מיד לאחר קבלת הודעת הטקסט לדנה. חבילה זו, בעלת אורך נתונים של (Len=0), אינה נושאת מידע אפליקטיבי אלא משמשת את שכבת התעבורה לוודא אמינות. מספר ה- (19) Acknowledgment מעיד על כך שהשרת קיבל בהצלחה את כל 14 הבתים של ההודעה המקורית ומצפה לנתון הבא. זהו המנגנון המבטיח שבמקרה של איבוד חבילה ברשת, המערכת תדע לבצע שידור חוזר (Retransmission).

חשוב לציין כי עבור כל חבילת מידע שנשלחת בשכבת היישום, מוחזרת חבילת אישור (ACK) בשכבת התעבורה כחלק מסטנדרט פרוטוקול ה-TCP. חבילות אלו אינן מכילות מידע בשכבת היישום (Len=0), ולכן, כדי לשמור על רצף הניתוח ולמנוע כפילות, אמנע מלהסביר את שאר האישורים בהרחבה.

```
83 844.449436 127.0.0.1 127.0.0.1 TCP 73 8888 → 63691 [PSH, ACK] Seq=1 Ack=5 Win=408320 Len=17 TSval=1098072107 TSecr=2353658006
```

שורה 83 מראה את החלק המרכזי בקוד, המידע שהשרת קיבל מהלקוח "דנה" הוא מעביר אותו (פורט של השרת 8888) ללקוח "רוני" (פורט 63691).

שכבת היישום (Application Layer)

0000	02 00 00 00 45 00 00 45	00 00 40 00 40 06 00 00	...E..E..@..@..
0010	7f 00 00 01 7f 00 00 01	22 b8 f8 cb 10 7d 26 bb	....."}&.
0020	6e 71 a0 0a 80 18 18 ec	fe 39 00 00 01 01 08 0a	nq.....9.....
0030	41 73 40 2b 8c 49 f8 96	5b 44 61 6e 61 5d 3a 20	As@+·I· [Dana]:
0040	20 68 69 69 20 72 6f 6e	69	hii ron i

- עיבוד הנתונים: השרת לא רק מעביר את הטקסט, אלא מעבד אותו. בתצוגת ה-Hex/ASCII ניתן לראות שה-Payload שונה ל: [Dana]:hi roni
- אורך המידע: אורך ההודעה המעובדת הוא 17 בתים (Len=17), כפי שמופיע בשדה ה-Data.

## שכבת התעבורה (Transport Layer - TCP)

```
Transmission Control Protocol, Src Port: 8888, Dst Port: 63691, Seq: 1, Ack: 5, Len: 17
Source Port: 8888
Destination Port: 63691
[Stream index: 2]
[Stream Packet Number: 7]
> [Conversation completeness: Complete, WITH_DATA (31)]
[TCP Segment Len: 17]
Sequence Number: 1 (relative sequence number)
Sequence Number (raw): 276637371
[Next Sequence Number: 18 (relative sequence number)]
Acknowledgment Number: 5 (relative ack number)
Acknowledgment number (raw): 1852940298
1000 .... = Header Length: 32 bytes (8)
> Flags: 0x018 (PSH, ACK)
Window: 6380
[Calculated window size: 408320]
[Window size scaling factor: 64]
Checksum: 0xfe39 [unverified]
[Checksum Status: Unverified]
Urgent Pointer: 0
> Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
> [Timestamps]
> [SEQ/ACK analysis]
[Client Contiguous Streams: 1]
[Server Contiguous Streams: 1]
TCP payload (17 bytes)
```

- שינוי פורטים: בשלב זה, השרת הוא המקור (Src Port: 8888) והלקוח רוני הוא היעד (Dst Port: 63691).
- Flags: שוב מופיע הדגלים (PSH,ACK) נעשה שימוש בדגל PSH כדי להבטיח שההודעה תופיע בטרמינל של רוני באופן מיידי.

## שכבת הרשת (Network Layer - IPv4)

```
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
0100 .... = Version: 4
.... 0101 = Header Length: 20 bytes (5)
> Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
Total Length: 69
Identification: 0x0000 (0)
> 010. .... = Flags: 0x2, Don't fragment
...0 0000 0000 0000 = Fragment Offset: 0
Time to Live: 64
Protocol: TCP (6)
Header Checksum: 0x0000 [validation disabled]
[Header checksum status: Unverified]
Source Address: 127.0.0.1
Destination Address: 127.0.0.1
[Stream index: 0]
```

בשכבה זו מתבצע הניתוב הלוגי בין המארחים:

- Version: ניתן לראות שימוש ב-IPv4 (גרסה 4).
- Addresses: כתובת המקור והיעד הן 127.0.0.1 (Loopback), מה שמעיד על תקשורת פנימית במחשב. החבילה מנותבת הפעם ל"רוני" אבל מאחר והכל פנימי זה אותה כתובת. ההפרדה הלוגית בין הלקוחות בשכבת הרשת מתבצעת באמצעות (Source Ports) השונים שהוקצו לכל תהליך לקוח על ידי מערכת ההפעלה.
- Protocol: מופיע הערך (6) TCP. זהו השדה שמורה לשכבת הרשת שהיא נושאת שייך לפרוטוקול TCP בשכבה שמעליה.
- TTL (Time to Live): הערך הוא 64, המייצג את מספר ה"קפיצות" המקסימלי שהחבילה יכולה לעבור ברשת.

כעת נדבר על סגירת תקשורת בין הצד של הלקוח לשרת(כאשר הלקוח שולח "quit" לשרת):

```
--- Welcome to the chat, Roni! ---
Send [Target:Message] or 'quit':

[RECEIVE] [Dana]: hii roni
Send [Target:Message] or 'quit': Dana: hii dana
Send [Target:Message] or 'quit': quit
[SYSTEM] Closing connection...

[SYSTEM] Connection to server lost.
roni_macmini@rwnys-Mac-mini ~ %
```

```
project/server_clients/server.py
[SERVER] Listening on 127.0.0.1:8888...
[SERVER] Yossi connected from ('127.0.0.1', 63689)
[SERVER] Dana connected from ('127.0.0.1', 63690)
[SERVER] Roni connected from ('127.0.0.1', 63691)
[SERVER] Roni disconnected.
```

ניתן לראות בתמונות כי הלקוח "רוני" יזם סגירה מסודרת של ה - socket עם השרת.

זה מה שניתן לראות בתעבורת ה - wireshark:

251	2873.213719	127.0.0.1	127.0.0.1	TCP	56	63691 → 8888 [FIN, ACK] Seq=19 Ack=18 Win=408320 Len=0 TSval=2356529160 TSecr=1098083474
252	2873.213786	127.0.0.1	127.0.0.1	TCP	56	8888 → 63691 [ACK] Seq=18 Ack=20 Win=408320 Len=0 TSval=1100100815 TSecr=2356529160
253	2873.214697	127.0.0.1	127.0.0.1	TCP	56	8888 → 63691 [FIN, ACK] Seq=18 Ack=20 Win=408320 Len=0 TSval=1100100816 TSecr=2356529160
254	2873.214774	127.0.0.1	127.0.0.1	TCP	56	63691 → 8888 [ACK] Seq=20 Ack=19 Win=408320 Len=0 TSval=2356529161 TSecr=1100100816

נדבר על שורה 251:

שבו הלקוח מציין לשרת על הסגירה המסודרת של התקשורת ביניהם ניתן לראות דגל חדש בשם FIN המעיד על בקשת סיום תקשורת.

שכבת התעבורה (TCP)

```
Transmission Control Protocol, Src Port: 63691, Dst Port: 8888, Seq: 19, Ack: 18, Len: 0
Source Port: 63691
Destination Port: 8888
[Stream index: 2]
[Stream Packet Number: 11]
> [Conversation completeness: Complete, WITH_DATA (31)]
[TCP Segment Len: 0]
Sequence Number: 19 (relative sequence number)
Sequence Number (raw): 1852940312
[Next Sequence Number: 20 (relative sequence number)]
Acknowledgment Number: 18 (relative ack number)
Acknowledgment number (raw): 276637388
1000 .... = Header Length: 32 bytes (8)
> Flags: 0x011 (FIN, ACK)
Window: 6380
[Calculated window size: 408320]
[Window size scaling factor: 64]
Checksum: 0xfe28 [unverified]
[Checksum Status: Unverified]
Urgent Pointer: 0
> Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
> [Timestamps]
[Client Contiguous Streams: 1]
[Server Contiguous Streams: 1]
```

- דגלים (Flags): ניתן לראות את הדגל (FIN, ACK) 0x011. דגל ה-FIN (Finish) הוא הסיגנל ששולח הלקוח לשרת כדי להודיע: "סיימתי לשלוח נתונים, ברצוני לסגור את חצי החיבור שלי".
- אורך (Length): שים לב שערך ה-Len הוא 0. בניגוד להודעת טקסט, חבילת הניתוק היא חבילת בקרה בלבד ואינה נושאת מידע בשכבת היישום.
- סדר פעולות: הלקוח (פורט 63691) שולח את ה-FIN לשרת (פורט 8888).

## שכבת הרשת (IPv4)

```
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
> Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 52
  Identification: 0x0000 (0)
> 010. .... = Flags: 0x2, Don't fragment
  ...0 0000 0000 0000 = Fragment Offset: 0
  Time to Live: 64
  Protocol: TCP (6)
  Header Checksum: 0x0000 [validation disabled]
  [Header checksum status: Unverified]
  Source Address: 127.0.0.1
  Destination Address: 127.0.0.1
  [Stream index: 0]
```

- IP Addresses: החבילה עדיין עוברת ב-127.0.0.1 (Loopback).
- Total Length: האורך הכולל הוא 52 בתים (כותרת IP של 20 בתים + כותרת TCP של 32 בתים), ללא נתוני אפליקציה.

כעת נדבר על שורה 252 :

בשורה זו ניתן לראות כי השרת מאשר את בקשת הסגירת תקשורת של הלקוח "רוני".

שכבת התעבורה (TCP)

```
Transmission Control Protocol, Src Port: 8888, Dst Port: 63691, Seq: 18, Ack: 20, Len: 0
  Source Port: 8888
  Destination Port: 63691
  [Stream index: 2]
  [Stream Packet Number: 12]
> [Conversation completeness: Complete, WITH_DATA (31)]
  [TCP Segment Len: 0]
  Sequence Number: 18 (relative sequence number)
  Sequence Number (raw): 276637388
  [Next Sequence Number: 18 (relative sequence number)]
  Acknowledgment Number: 20 (relative ack number)
  Acknowledgment number (raw): 1852940313
  1000 .... = Header Length: 32 bytes (8)
> Flags: 0x010 (ACK)
  Window: 6380
  [Calculated window size: 408320]
  [Window size scaling factor: 64]
  Checksum: 0xfe28 [unverified]
  [Checksum Status: Unverified]
  Urgent Pointer: 0
> Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
> [Timestamps]
> [SEQ/ACK analysis]
  [Client Contiguous Streams: 1]
  [Server Contiguous Streams: 1]
```

- דגלים (Flags): ניתן לראות את הדגל (ACK) 0x010. השרת מאשר כאן שקיבל את בקשת ה-FIN מהלקוח.
- Acknowledgment Number: Ack=20. בחבילה הקודמת של רוני ה-Sequence Number היה 19. השרת מעלה את המספר ב-1 כדי לאשר שקיבל את ה-FIN (שנחשב כבית אחד לצורך הסנכרון), ובכך הוא "חותם" את הצד הזה של השיחה.

- אורך (Length): גם כאן Len=0, מכיוון שזו חבילת בקרה בלבד.

שכבת הרשת (IPv4):

```
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
> Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 52
  Identification: 0x0000 (0)
> 010. .... = Flags: 0x2, Don't fragment
  ...0 0000 0000 0000 = Fragment Offset: 0
  Time to Live: 64
  Protocol: TCP (6)
  Header Checksum: 0x0000 [validation disabled]
  [Header checksum status: Unverified]
  Source Address: 127.0.0.1
  Destination Address: 127.0.0.1
  [Stream index: 0]
```

- כיוון התעבורה: החבילה יוצאת מהשרת (Src: 127.0.0.1) אל הלקוח (Dst: 127.0.0.1) אותה כתובת כי אנחנו מבצעים הכל פנימי.

- Protocol: מופיע הערך (6) TCP, המזהה את סוג הנתונים בשכבה 4.

בשורות 253-254 מתבצע אותו תהליך של סגירת מסודרת של התקשורת אך הפעם מהצד של השרת מה שנקרא "TCP - Four way handsashake". כלומר השרת יזום את הסגירה והלקוח מאשר זאת. מבחינת תהליך היא תהיה מאוד דומה גם הדגלים, רק הכתובות ישתנו (ספציפית אני בתקשורת לוקאלית לכן בדוגמא זאת זה לא השתנה) והפורטים משתנים.

## סיכום:

באמצעות ניתוח ה-Wireshark הצלחנו לתעד את מעגל החיים המלא של תקשורת מבוססת TCP ביישום הצ'אט:

- Initialization: יצירת הקשר ב-Three-way handshake.
  - Data Transfer: העברת הודעות מעובדות בין השרת ללקוחות תוך שימוש בדגלי PSH.
  - Termination: סגירה מסודרת בארבעה שלבים (FIN/ACK) המבטיחה ששני הצדדים שחררו את המשאבים בצורה תקינה.
- עמידה בתהליכים אלו מוכיחה כי היישום מממש את עקרונות האמינות והסדר של מודל ה-TCP/IP ומטפל בצורה נכונה בניהול משאבי הרשת.

## **תיאור שימוש בבינה מלאכותית:**

מטרות השימוש:

- סיוע במבנה וארגון הדו"ח הטכני.
- דיוק בניתוח חבילות ב-Wireshark.
- סיוע בתכנות הקוד למציאת מקרי קצה והתמודדות עם אירועים מקביליים.

דוגמאות לפרומפטים:

- "תבדוק אם השתמשתי במונחים לא נכונים"
- "האם הניתוחים שעשיתי בשכבות התקשורת שנקלט ב - wireshark תקינים ומפורטים?"
- "האם בין הדוח שביצעתי לדרישות חסר משהו?"
- "האם הקוד עומד בדרישות שהוא יצליח גם להעביר הודעות וגם לקבל במקביל?"