**דוח מסכם – עבודה רשתות תקשורת מחשבים**

**מגישות: רוני לוי 325588184 , ליאן מאור 325162980**

**חלק א:**

1. **חלק 1: אריזה ולכידת המנות**

**דרך יצירת הקובץ עם הודעות בשכבת היישום**

בשלב הראשון הכנו קובץ csv שמהווה לנו כקובץ הקלט. הקובץ (group05\_DNS\_Input.csv) נוצר לצורך יצירת התוכן והבטחת מבנה הודעות תקין בשכבת האפליקציה. הקובץ מכיל הודעות המדמות פרוטוקול .HTTP

כל שורה בקובץ מייצגת הודעה אחת, ומכילה את השדות הבאים:

msg\_id – מזהה ייחודי לכל הודעה

app\_protocol – שם הפרוטוקול בשכבת היישום (למשל: HTTP)

src\_app – שם היישום השולח (לדוגמה: client\_browser)

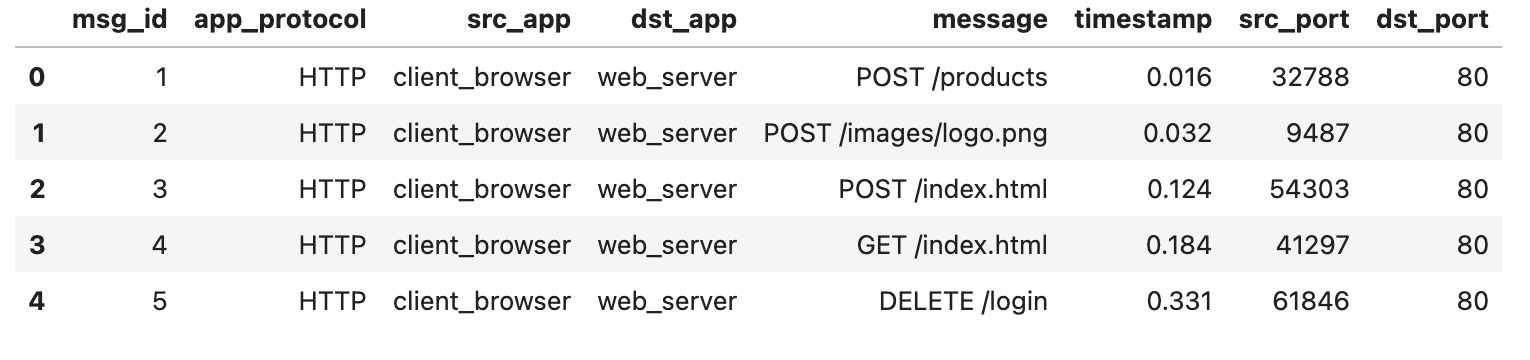
dst\_app – שם היישום המקבל (לדוגמה: web\_server)

message – תוכן ההודעה בשכבת היישום

timestamp – זמן שליחת ההודעה (ביחידות זמן יחסיות)

**מטרת הקובץ:** הקובץ משמש כ"חומר הגלם" לתקשורת. הוא מאפשר לנו להגדיר מראש אילו הודעות יישלחו ברשת, מי המקור (src\_app), מי היעד (dst\_app) ומהו תוכן המידע (Message)..

הקובץ נוצר באמצעות בינה מלאכותית השדות הוגדרו לפי דרישות הפרויקט: מזהה הודעה, שם הפרוטוקול, אפליקציית מקור ויעד, תוכן ההודעה וזמן.



התמונה שלפניכם מראה חלק מהמחברת שמציין את 5 השורות הראשונות של הקובץ שנטען בהצלחה.

**תיאור והסבר של תהליך אריזת המנות:**

תהליך אריזת המנות מדמה את האופן שבו הודעה לוגית משכבת היישום הופכת בפועל לחבילת רשת הנשלחת דרך המחשב.

בפרוייקט שלנו, כל שורה בקובץ מייצגת הודעת יישום עצמאית. לאחר טעינת הקובץ במחברת, כל שורה נקראת והופכת להודעה בשכבת היישום. בשלב זה ההודעה עדיין ״לוגית״ – היא קיימת כמחרוזת טקסט המייצגת פעולה של יישום.

עבור כל הודעה, המחברת מדמה את תהליך האריזה לפי מודל tcp/ip , באופן מדורג שכבה אחר שכבה:

**: (Application Layer – HTTP) שכבת היישום**

בשלב זה קיימת ההודעה המקורית כפי שהיא מופיע ב csv . זוהי הודעה ברמת היישום.

**שכבת התעבורה (Transport Layer – TCP):**

ההודעה נארזת בתוך מקטע של tcp. בשלב זה מתווספים: פורט המקור , פורט היעד ושדות הבקרה של tcp (רצף מספרי , ack , מנגנונים למינהם).

שכבה זו אחראית לוודא שהמידע יועבר בצורה אמינה, לפי הסדר, ויגיע ליישום הנכון בצד המקבל.

**שכבת הרשת (Network Layer – IP) :**

מקטע הtcp נארז בתוך חבילת ip. בשלב זה מתווספים כתובת המקור והיעד. (אצלנו בפרוייקט 127.0.0.1).

במימוש שלנו במחברת הפונקצייה build\_ip\_header בונה את כותרת הip הכוללת: כתובת ip מקור, כתובת ip יעד, אורך החבילה, סוג הפרוטוקול (tcp).

**שכבת הקישור (Link Layer):**

לבסוף, חבילת ה-IP עטופה במסגרת (Frame) המתאימה לממשק הרשת. רק לאחר שלב זה ניתן לשדר את המידע בפועל דרך המערכת. לצורך השליחה המעשית, המחברת משתמשת במחלקה RawTcpTransport, אשר שולחת את המידע דרך ממשק ה-Loopback של המחשב.

לסיכום, לאחר טעינת הקובץ csv במחברת: כל שורה מומרת להודעה בשכבת היישום, ההודעה נארזת במקטע tcp הכולל פורטים ושדות בקרה, מקטע ה tcp נארז בחבילת ip עם כתובת מקור ויעד ולבסוף החבילה עטופה במסגרת ומוכנה לשידור דרך ממשק הרשת.

כך המערכת מדמה בצורה מלאה את תהליך האריזה לפי מודל TCP/IP, עד להפיכת טקסט לוגי מחוברת ה-Jupyter לחבילת רשת אמיתית הניתנת ללכידה ב-Wireshark

**תיאור והסבר של תהליך לכידת התעבורה**

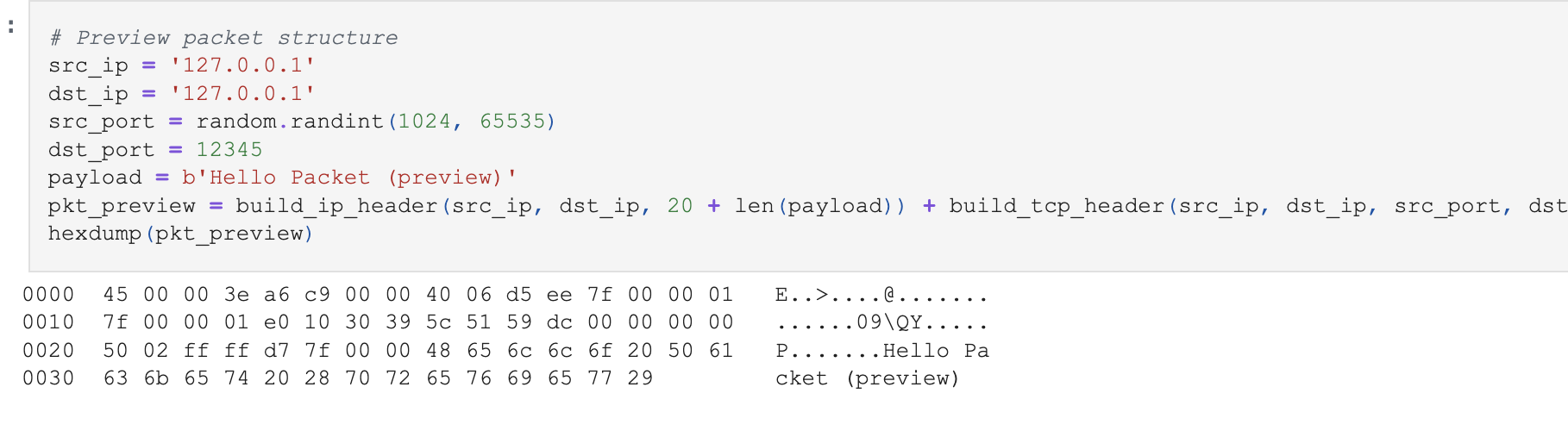
לאחר יצירת קובץ ה-CSV, המכיל את ההודעות בשכבת היישום (Application Layer), הרצנו את הקוד אשר קורא את הקובץ ושולח כל הודעה כ־payload בתוך חבילת רשת. טענו אותו למחברת **Jupyter** שסופקה לנו. במחברת ביצענו הדמיה של תהליך האריזה, המדמה את מעבר המידע במורד שכבות מודל TCP / IP. הקוד מדמה שליחת הודעות ברשת באמצעות יצירת מנות TCP באופן יזום, כאשר כל שורה מה-CSV נשלחת כהודעת טקסט בתוך גוף המנה.

לפני ההרצה פתחנו את Wireshark ובחרנו בממשק ה-Loopback, מאחר והתקשורת מתבצעת בין localhost 127.0.0.1. הגדרנו מסנן תצוגה מתאים (ip.addr == 127.0.0.1 && tcp.port == 12345) והתחלנו לכידה.

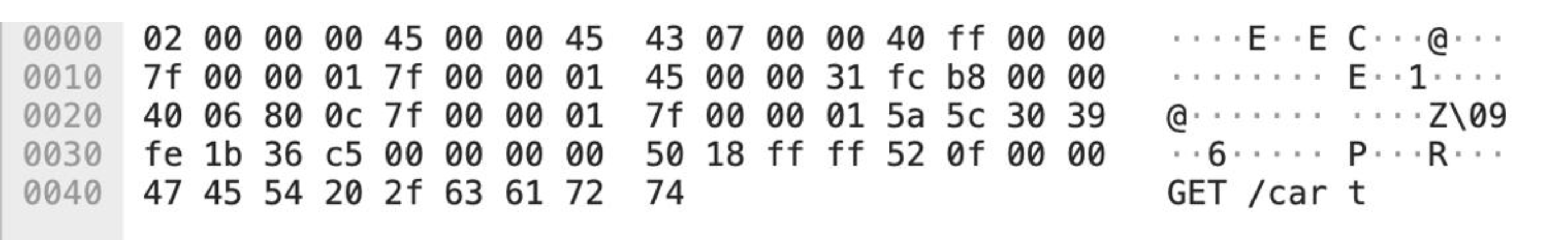
לצורך לכידת התעבורה:

1. הופעל Wireshark ונבחר ממשק רשת פעיל (( Loopback
2. הוגדר מסנן תצוגה מתאים לפרוטוקול הנבחר. (tcp port 80)
3. בזמן ש-Wireshark פועל במצב Capture, הופעלה מחברת ה Jupyter
4. המחברת שלחה את המנות שנוצרו מתוך קובץ ה-CSV אל שכבת הרשת.
5. Wireshark לכד את המנות בזמן אמת והציג אותן כרשומות תעבורה.
6. בסיום ההרצה, הלכידה נעצרה והקובץ נשמר בפורמט .pcap.

ממצאים מהמחברת:

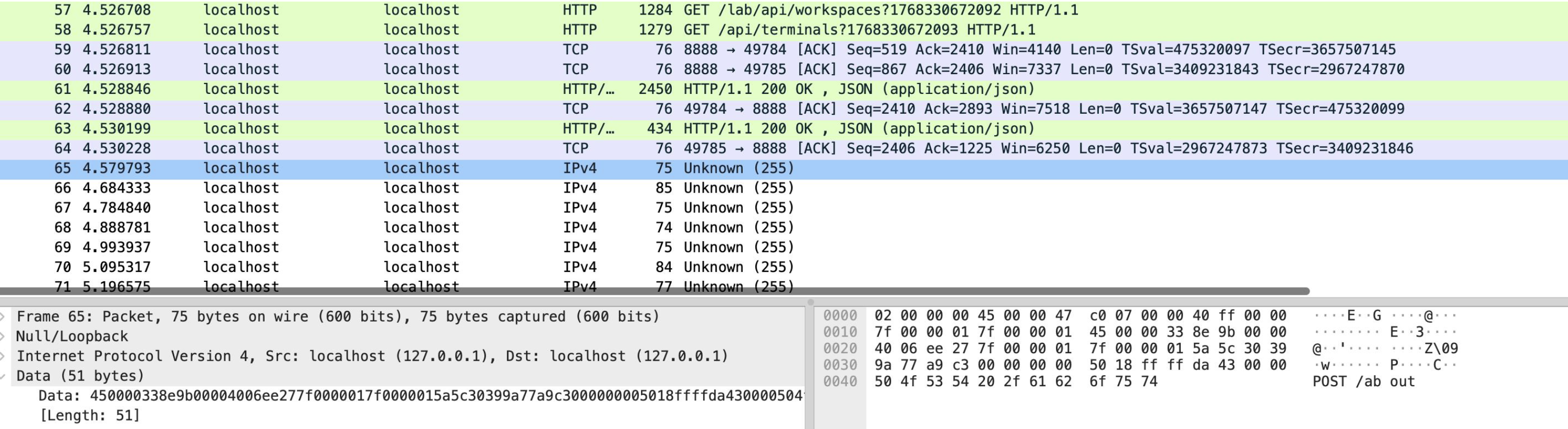


ממצאים מה wireshark:



בתמונה שלפניכם ניתן לראות את הלכידה מאחת מן ההודעות מהקובץ csv: (GET / car)

**תיאור והסבר של התעבורה שנלקדה מה wireshark וניתוחה:**



בתמונה ניתן לראות רצף של חבילות שנשלחו בין localhost ל־localhost, כלומר בין

127.0.0.1 → 127.0.0.1, דרך ממשק ה-Loopback. זה מאשר שהמחברת אכן שולחת מנות אמיתיות דרך מחסנית הרשת של מערכת ההפעלה, גם אם התקשורת היא מקומית.

בחלק העליון של Wireshark מופיעות שורות מסוג: HTTP GET /lab/api/workspaces … , HTTP/1.1 200 OK, JSON (application/json), מנות TCP עם דגלים כמו [ACK], מנות IPv4 Unknown (255).

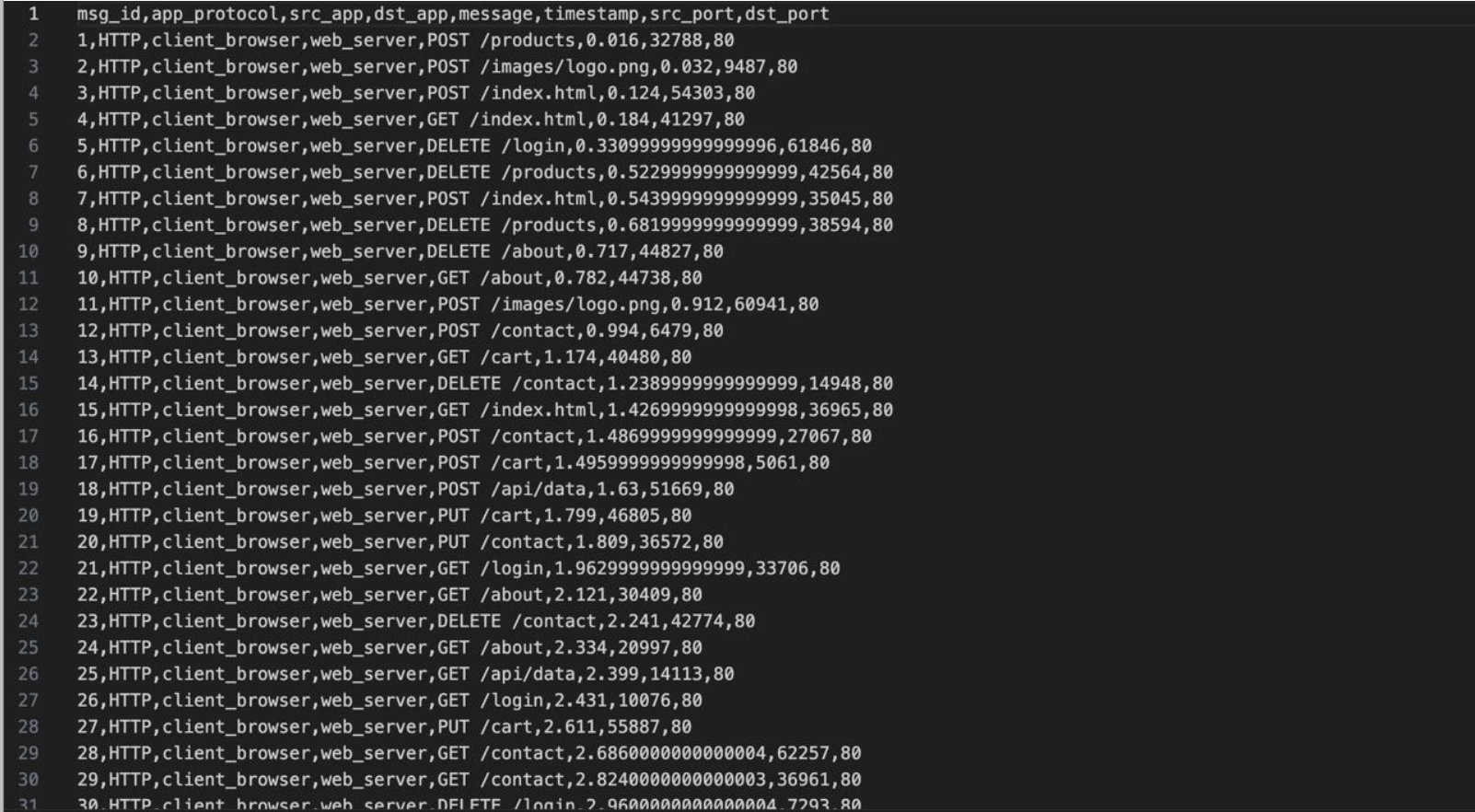
IPv4 Unknown מדובר בפרוטוקול לא מזוהה על ידי הwireshark ולא שגיאה.

הדבר מצביע על כך שבמהלך ההרצה מתקיימת גם תעבורת HTTP פנימית של סביבת ה-Jupyter, וגם המנות שנוצרו על ידי הקוד שלנו.

בחלק ה-Bytes של אותה חבילה ניתן לזהות את תוכן הנתונים, ובצד ימין מופיע הטקסט: POST /about. כלומר, ניתן לראות ישירות בתוך החבילה את תוכן ההודעה שנשלחה – הטקסט שמקורו במחברת ובקובץ ה-CSV. זה מוכיח שההודעה משכבת היישום אכן נארזה, הועברה דרך שכבות TCP ו-IP, והגיעה כחבילת רשת מלאה שניתנת ללכידה וניתוח.

מתוך הלכידה ניתן להסיק כי: המחברת שולחת בפועל מנות TCP/IP דרך מערכת ההפעלה , כל הודעה מה-CSV הופכת ל-payload בתוך חבילת רשת, החבילות נשלחות מ-127.0.0.1 ל-127.0.0.1 דרך ה-Loopback , Wireshark לוכד אותן בזמן אמת ומאפשר לראות גם את שכבות הפרוטוקול וגם את תוכן ההודעה עצמה.

כך הלכידה מאמתת שהפרויקט מדמה בהצלחה את תהליך המעבר מהודעה לוגית במחברת Jupyter לחבילת רשת אמיתית.



קובץ ה-CSV מכיל רצף של הודעות יישום, כאשר כל שורה מייצגת פעולה של לקוח מול שרת HTTP.נכל שורה כוללת בין היתר: מזהה הודעה (msg\_id) ,הפרוטוקול היישום (HTTP), סוג הלקוח והשרת, סוג הבקשה (GET, POST, PUT, DELETE) , הנתיב (למשל /about, /products, /login), חותמת זמן, פורט מקור, פורט יעד (80 – שרת HTTP)

לדוגמה, בקובץ מופיעות שורות כגון:



בעת הרצת המחברת, כל שורה בקובץ ה-CSV נקראת והופכת להודעת יישום אחת.

ההודעה מועברת כ-payload למחלקת RawTcpTransport, אשר אורזת אותה בתוך מקטע TCP, ולאחר מכן בתוך חבילת IP.

בלכידה ב-Wireshark ניתן לראות התאמה ישירה בין תוכן הקובץ לבין התעבורה:

בשדה ה-Bytes מופיעים מחרוזות כמו POST /about, GET /contact, DELETE /products.

אלו הם בדיוק הערכים שמופיעים בעמודת message בקובץ ה-CSV.

כלומר, כל שורה מהקובץ הופכת בפועל לחבילת רשת שנשלחת דרך מערכת ההפעלה ונלכדת ב-Wireshark.

כך מתקבל חיבור ברור בין:

קובץ CSV → הודעת יישום במחברת → אריזה ב-TCP/IP → חבילת רשת → לכידה ב-Wireshark

והלכידה מוכיחה שהתוכן הלוגי שבקובץ אכן עבר את כל שכבות המודל והפך לתעבורת רשת אמיתית.

**חלק 2: יצירת יישום וניתוח תעבורה**

**הסבר כללי על המערכת ומבנה הקוד**

המערכת שפיתחנו היא מערכת צ’אט המבוססת על TCP Sockets. היא בנויה משרת מרכזי ומספר לקוחות, כאשר השרת משמש כמתווך בין הלקוחות: הלקוחות לא מתקשרים אחד עם השני ישירות, אלא כל ההודעות עוברות דרך השרת.

הקוד מחולק לשלושה קבצים עיקריים:

1) server.py – השרת

השרת מאזין לחיבורים נכנסים ומאפשר חיבור של מספר לקוחות במקביל. כל לקוח שמתחבר מקבל טיפול בתהליך מקביל (Thread), כך שהשרת יכול לשרת בו־זמנית לפחות 5 לקוחות כפי שנדרש. השרת שומר את הלקוחות המחוברים לפי שם ייחודי, ומחזיק גם מידע על “מי משוחח עם מי” בצ’אט הנוכחי.

כדי לפתוח שיחה, השרת מצפה שהלקוח יבקש שיחה עם לקוח אחר לפי שם. אם הלקוח המבוקש מחובר וזמין, השרת “מקשר” ביניהם ומתחיל להעביר הודעות מהאחד לשני. אם היעד לא קיים/לא מחובר/כבר בשיחה, השרת מחזיר הודעת שגיאה מתאימה.

בנוסף, השרת כולל טיפול בשגיאות ובניתוק לא צפוי: אם לקוח מתנתק באמצע, השרת סוגר את החיבור, מסיר אותו מרשימת המחוברים, ואם הוא היה בשיחה—מנתק גם את השיחה ומודיע לצד השני.

2) client.py – הלקוח

הלקוח מתחבר לשרת דרך TCP, ומאפשר למשתמש:

* להזדהות בשם
* לבקש פתיחת צ’אט עם לקוח אחר לפי שם
* לשלוח הודעות בזמן אמת
* לקבל הודעות בזמן אמת

כדי לאפשר “זמן אמת”, הלקוח משתמש בשני זרמי עבודה במקביל:

זרם אחד לקריאת קלט מהמשתמש ושליחת הודעות

זרם נוסף שמאזין כל הזמן להודעות מהשרת ומדפיס אותן מיד למסך

כך מתקבלת חוויית צ’אט בסיסית: אפשר גם להקליד וגם לקבל הודעות בלי לחסום את התוכנית.

3) protocol.py – שכבת הפרוטוקול

כדי לפשט ולתקנן את התקשורת, השתמשנו בפרוטוקול טקסטואלי פשוט: כל הודעה היא שורה אחת שמסתיימת בירידת שורה (newline).

הקובץ הזה מרכז את פעולות השליחה והקבלה כך שגם השרת וגם הלקוח “מדברים” באותו פורמט. זה גם מקל מאוד על ניתוח ב־Wireshark, כי רואים את הטקסט בבירור בתוך ה־payload של TCP.

**הוראות הרצת השרת לקוח**

דרישות מקדימות

כל הריצה מתבצעת על אותו מחשב, באמצעות localhost 127.0.0.1, אך בשישה טרמינלים נפרדים, המדמים שישה תהליכים שונים: שרת ו5 לקוחות.

שלושת קבצי הפרויקט: server.py, client.py, protocol.py

server.py – מריץ שרת tcp שמאזין לחיבורים נכנסים.

client.py – מייצג לקוח שמתחבר לשרת, שולח הודעות ומקבל תגובות וכולל את הui.

protocol.py – ממש את הפרוטוקול של התקשורת : שליחת שורה וקבלת שורה על גבי socket.

**איך נפעיל את השרת לקוח?**

**שלב א**

במחשב אחד פתחנו שישה חלונות טרמינל נפרדים: טרמינל 1- שרת וחמשת האחרים לקוחות.

הרצנו את השרת המתאים לכל טרמינל , לאחר מכן בטרמינל של השרת אנחנו נראה את ההודעה: Server listening on 0.0.0.0:6543. זה אומר שהמרכזייה שלנו פתוחה ומחכה.

**שלב ב**

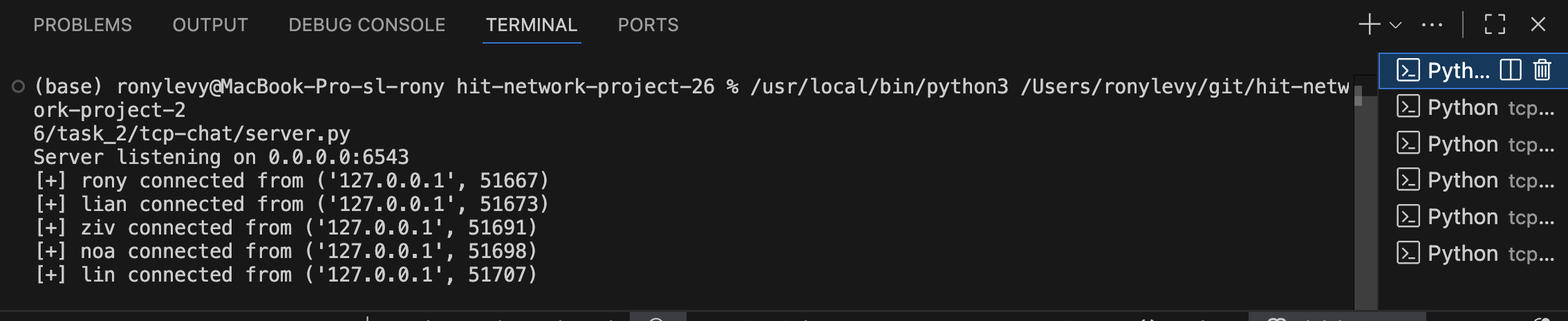
חמשת הטרמינלים האחרים הם ה"לקוחות". כדי שהם ידעו לאן להתחבר, אנחנו חייבים לעדכן את הקוד שלהם.

הלקוח יוצר חיבור לשרת (127.0.0.1:6543) ומאפשר הקלדת הודעות.

**שלב ג**

התחברות: כל אחד מהטרמינלים של הלקוחות יריץ את client.py ויזין את השם שלו (אנחנו עשינו עם השמות ״lian״ ו״"noa" , "ziv" ,"lin", "rony).

בדיקת השרת: נסתכל על המסך של מחשב השרת – הופיע לנו הודעות של [+] connected עבור כל אחד מאיתנו.



הזדהות ופתיחת צ’אט

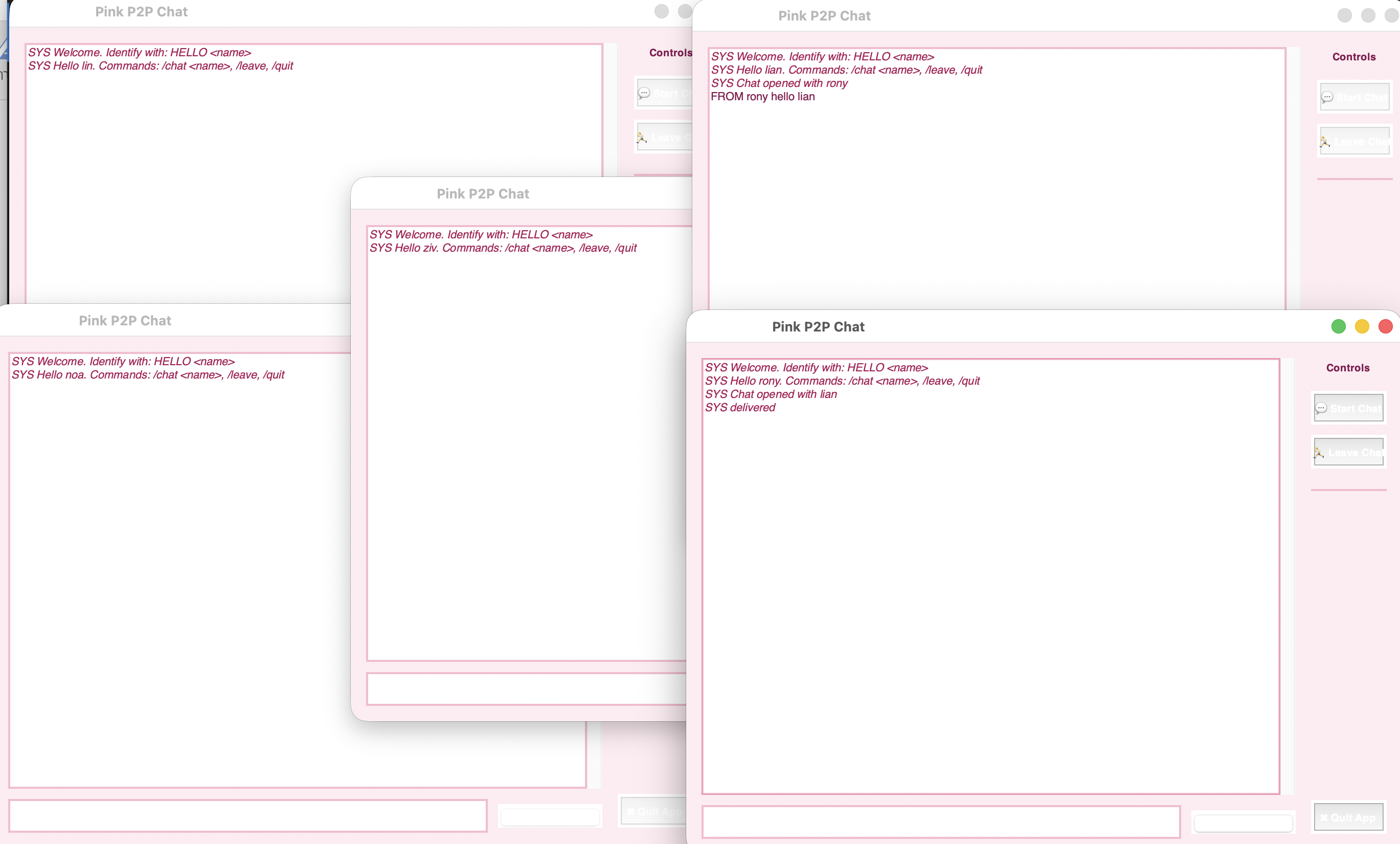
לאחר חיבור הלקוח לשרת, יש להזדהות בשם ייחודי באמצעות: HELLO <rony>

לאחר ששני הלקוחות הזדהו, ניתן לפתוח צ’אט עם לקוח אחר לפי שמו:

<chat <ziv

מרגע זה ניתן לשלוח הודעות טקסט חופשיות, אשר יועברו דרך השרת ללקוח השני בזמן אמת.

תמונות המייצגות שיחה עם 5 לקוחות בו זמנית:



**ניתוח תעבורה של היישום עד שכבת הרשת**

לצורך ניתוח התעבורה ביצענו לכידה באמצעות Wireshark בזמן שהשרת והלקוחות פעלו ושלחו הודעות.

לצורך מיקוד בתעבורת היישום הופעל ב־ Wireshark המסנן: tcp.port == 6543

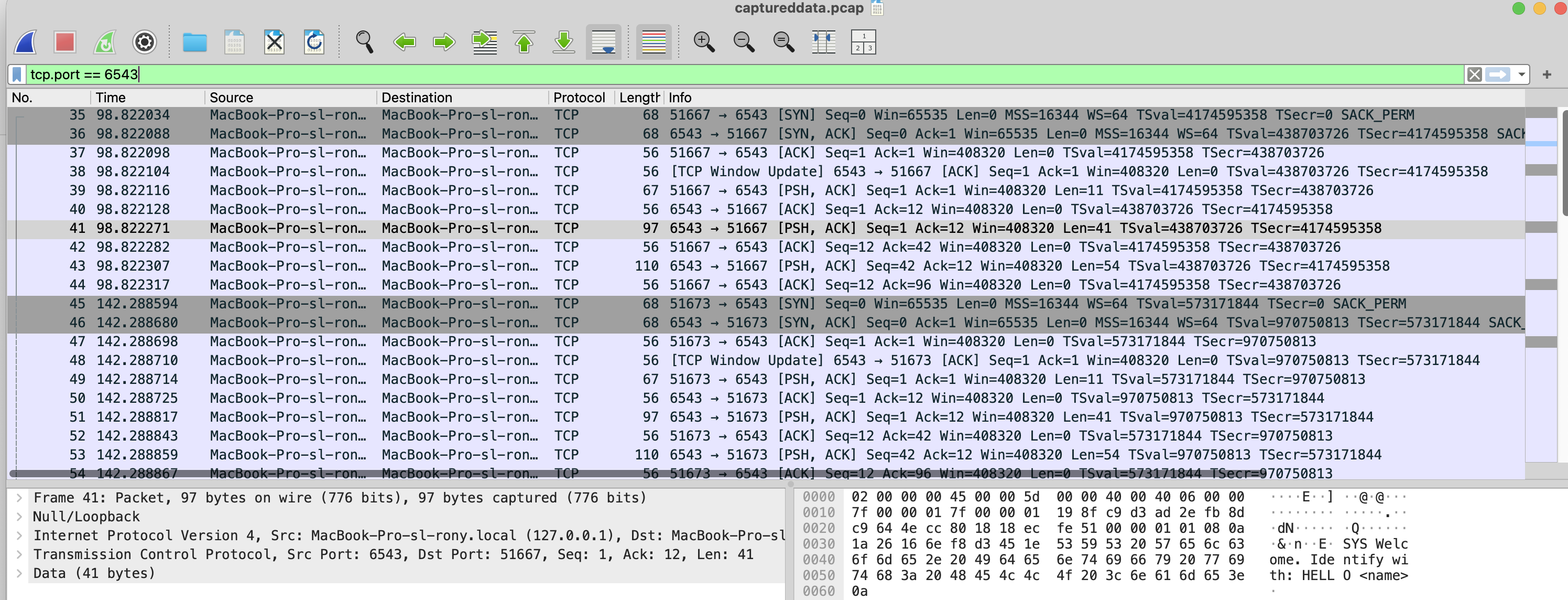
מסנן זה מציג רק מנות השייכות לתקשורת בין הלקוחות לשרת של מערכת הצ’אט, אשר מאזין לפורט 6543, ומסנן החוצה תעבורה פנימית אחרת של המחשב (כגון תעבורת מערכת או דפדפן).

1. הקמת חיבור TCP (3-Way Handshake)

בעת חיבור לקוח לשרת נצפה התהליך הסטנדרטי של TCP:

* הלקוח שולח SYN לשרת
* השרת מחזיר SYN-ACK
* הלקוח משלים עם ACK

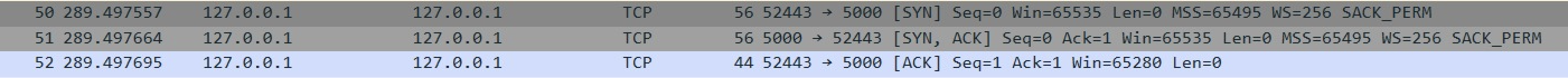
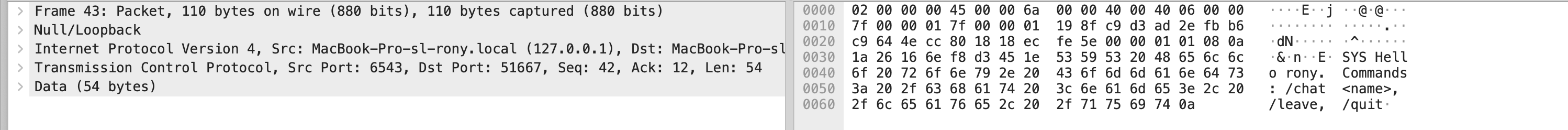
בתמונה מוצגת חבילת tcp שנלכדה לאחר הקמת החיבור. בחלון הפרטים ניתן לראות כתובת מקור ויעד (127.0.0.1) פורטים ( פורט 6543 – השרת , פורט 51667 – **לקוח 1**, פורט 51673 – **לקוח 2**, פורט 51691 - **לקוח 3** , פורט 51707- **לקוח 4 ,** פורט 51698 **– לקוח 5**) דגלים (psh ו ack) ואורך הנתונים ( len=11). בחלון הbytes מופיע הטקסט ״hello roni״ זהו בדיוק הטקסט שהוזן בטרמינל בשכבת היישום. התמונה ממחישה כיצד פקודת יישום (HELLO <name>) נארזת כ-payload בתוך מקטע TCP, ולאחר מכן נשלחת כחבילת רשת אמיתית שניתנת ללכידה וניתוח ב-Wireshark.



בתמונה הזו ניתן לראות שלוש חבילות עוקבות המייצגות את שלבי הקמת החיבור בין הלקוח לשרת:

הלקוח שולח SYN לפורט 6543, השרת משיב ב־SYN-ACK, והלקוח משלים ב־ACK.

רק לאחר שלב זה נוצר חיבור אמין, וניתן להתחיל בהעברת הודעות היישום.



1. פקודות והודעות בתוך TCP payload

לאחר הקמת החיבור:

* פקודת ההזדהות HELLO <name> נשלחת בתוך ה־payload של TCP
* פקודת פתיחת צ’אט /chat <name> נשלחת בתוך ה־payload
* הודעות צ’אט רגילות נשלחות גם הן בתוך ה־payload

ב־ Wiresharkניתן לראות בכלpacket :

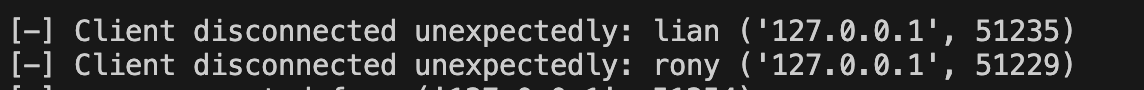
* שכבת IP:כתובת מקור ויעד
* שכבת: TCP פורט מקור/יעד, דגלים, מספרי רצף ואישורים
* שכבת Data (payload): הטקסט שנשלח בפועל, HELLO, /chat) הודעות(

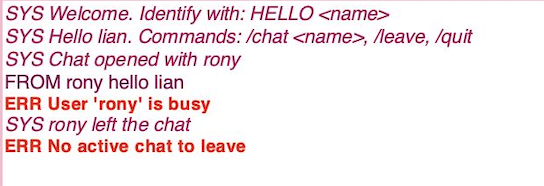
באמצעות התאמה בין הפלט במסכי הטרמינל לבין התוכן המופיע בשדה ה-payload ב-Wireshark ניתן לראות כי:

* שההודעות באמת עוברות דרך השרת
* שהשיחה נפתחת לפי שם לקוח
* שהתקשורת דו־כיוונית

התנהגות במקרה של ניתוק

כאשר לקוח מתנתק בצורה לא צפויה, נצפים packets שמייצגים סגירת חיבור TCP (למשל FIN/ACK או RST בהתאם לניתוק). במקביל, הצד השני מקבל הודעת מערכת מהשרת שמסמנת שהשיחה הסתיימה/הצד השני נותק.





**חלק 3 – תיאור שימוש בבינה מלאכותית:**

במהלך העבודה על הפרויקט נעזרתי בבינה מלאכותית בצ’אט ככלי עזר לימודי. הכלי סייע לי להבין מושגים תיאורטיים, לנסח הסברים בצורה ברורה ולבנות מבנה נכון לדוח. העזרה התמקדה בהכוונה, בהבהרת רעיונות ובשיפור הניסוח – אך כל ההבנה והיישום בוצעו על ידי. השימוש בבינה המלאכותית תרם ללמידה מעמיקה יותר ולהצגת הפרויקט בצורה מקצועית וברורה.

**מטרות השימוש:**

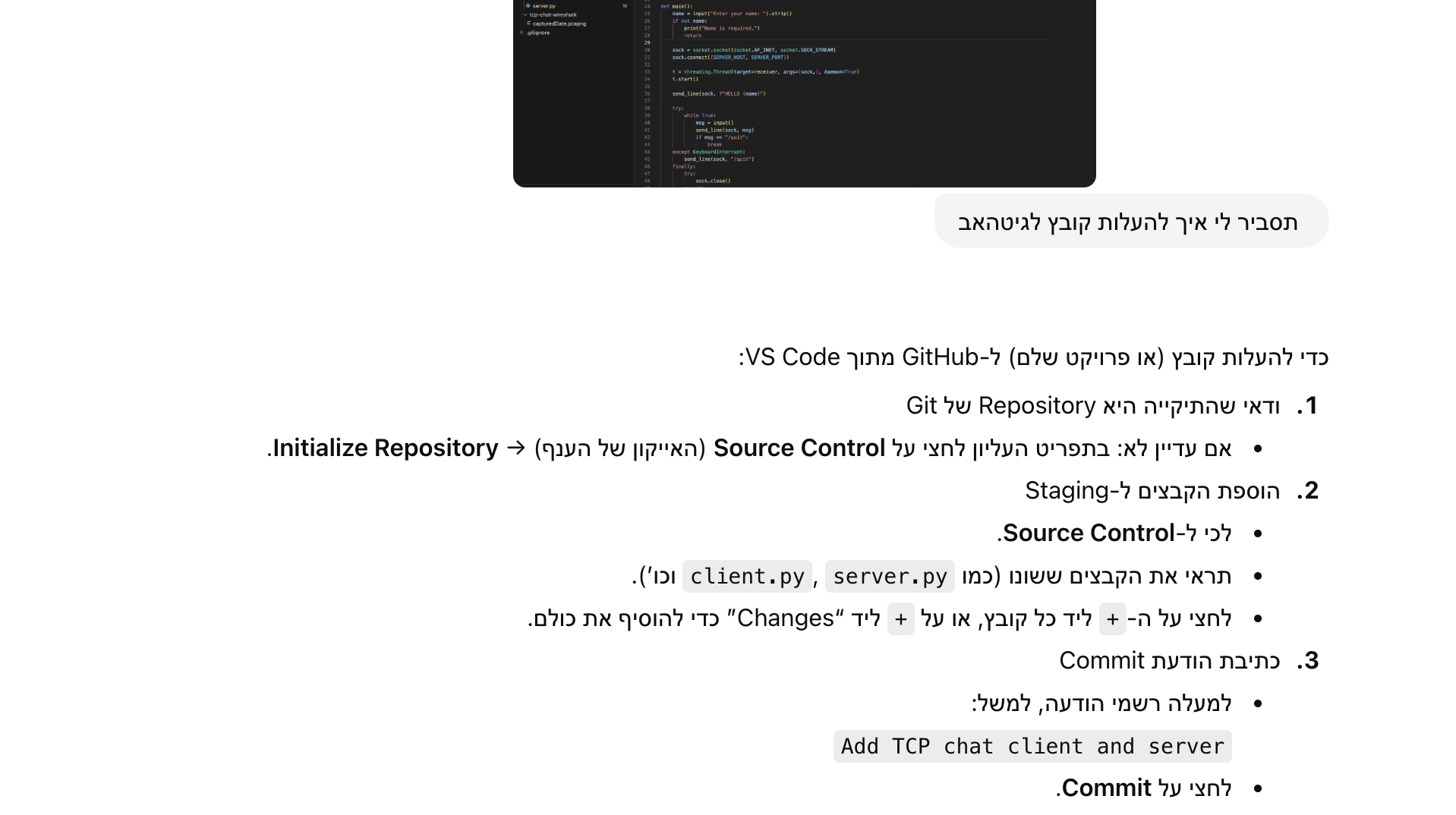
העמקת ההבנה התיאורטית: קבלת הסברים משלימים למושגים שנלמדו בקורס (כגון שכבות המודל, תהליכי אריזה וזרימת מידע ברשת), במטרה לחזק את ההבנה ולאפשר חיבור נכון בין התאוריה לבין המימוש בפועל בפרויקט.

שיפור ניסוח והצגה: שימוש בכלי לצורך חידוד ניסוחים, ארגון רעיונות ובניית מבנה ברור לדוח, כך שהתוכן ישקף בצורה מדויקת את תהליך העבודה, ההחלטות והפתרונות שנבחרו – מבלי להחליף את החשיבה או הביצוע העצמאיים.

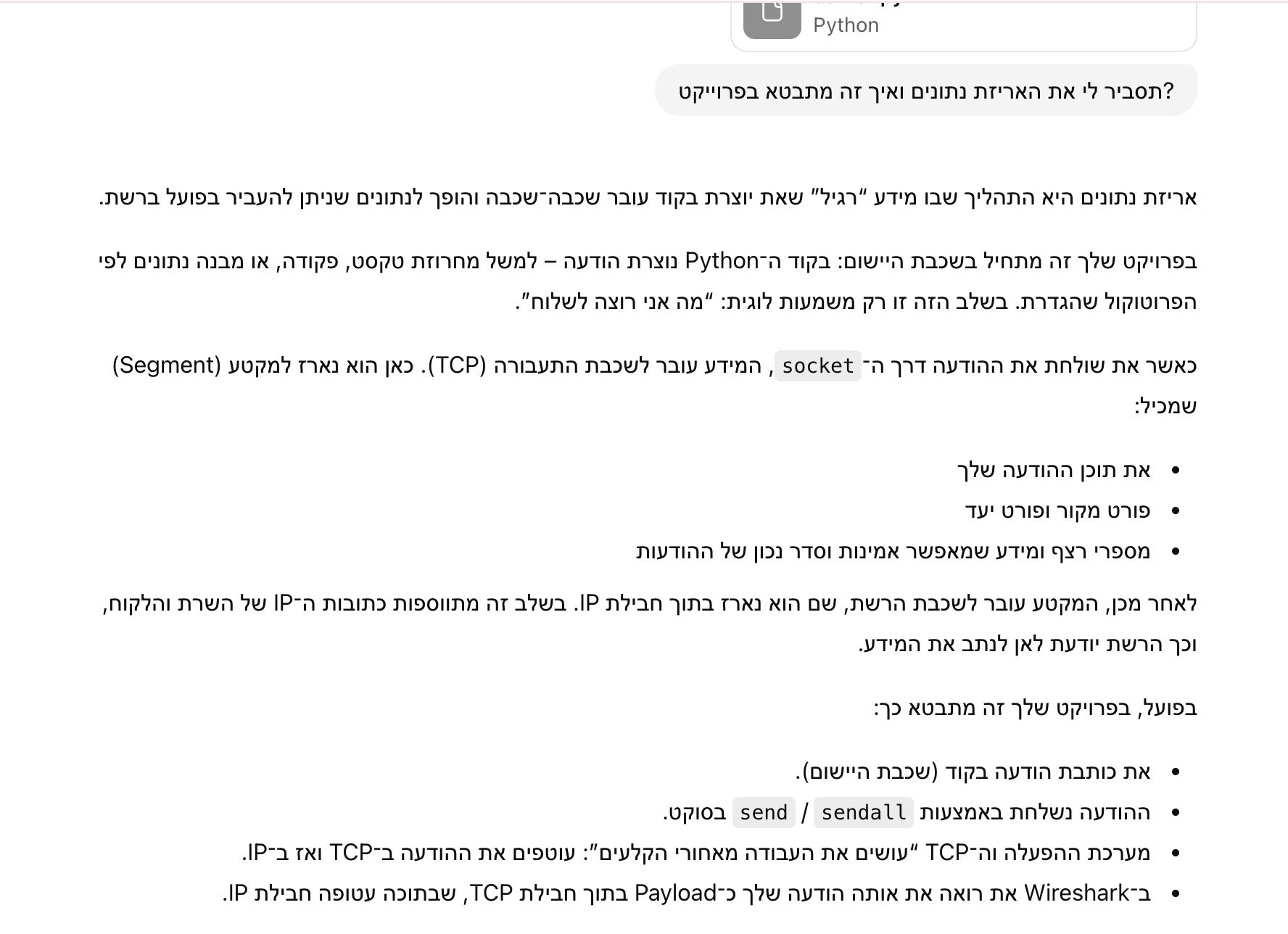
האצת תהליך זיהוי הבעיות במהלך הפיתוח, במיוחד בתקלות מורכבות הקשורות לתקשורת בין השרת ללקוח, כגון כשלים בחיבור, בעיות פתיחת פורטים או שיבושים בהעברת מידע, תוך הבנה ברורה של המקור הטכני לכל תקלה והשלכותיה על פעולת המערכת.

**דוגמאות שימוש:**

הסבר טכני על איך מעלים קבצים לגיטאהב:



הסבר תאורטי ומעמיק על איך אריזת נתונים עובדת ואיך זה מתבטא בפרוייקט שלנו:



הסבר על טיפול בשגיאות בפרוייקט:

