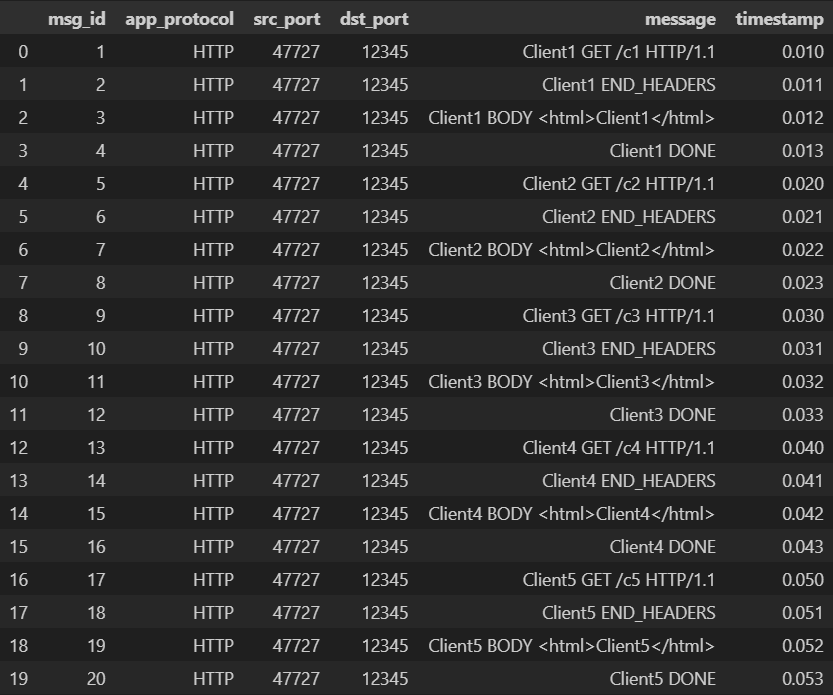
**חלק 1 – אריזת נתונים ולכידת מנות בעזרת Wireshark**

**דרך יצירת קובץ ה-CSV** בשלב הראשון, הכנו את קובץ הנתונים (CSV) שישמש כקלט לסימולציה. הקובץ נבנה בהתאם לפורמט הנדרש וכולל את כל שדות החובה:

* **msg\_id**  - מזהה ייחודי לכל הודעה.
* **app\_protocol**  - שם הפרוטוקול (HTTP).
* **src\_port /** **dst\_port** - פורטים לוגיים.
* **message** - תוכן ההודעה עצמו.
* **timestamp** - חותמת זמן לסימולציה.

דאגנו למלא את הקובץ בתוכן שמדמה שיחה הגיונית (כגון בקשות GET ותגובות), כדי שנוכל לזהות את המידע בקלות לאחר מכן ב-Wireshark.



**תיאור והסבר של תהליך אריזה של מנות (Encapsulation):**

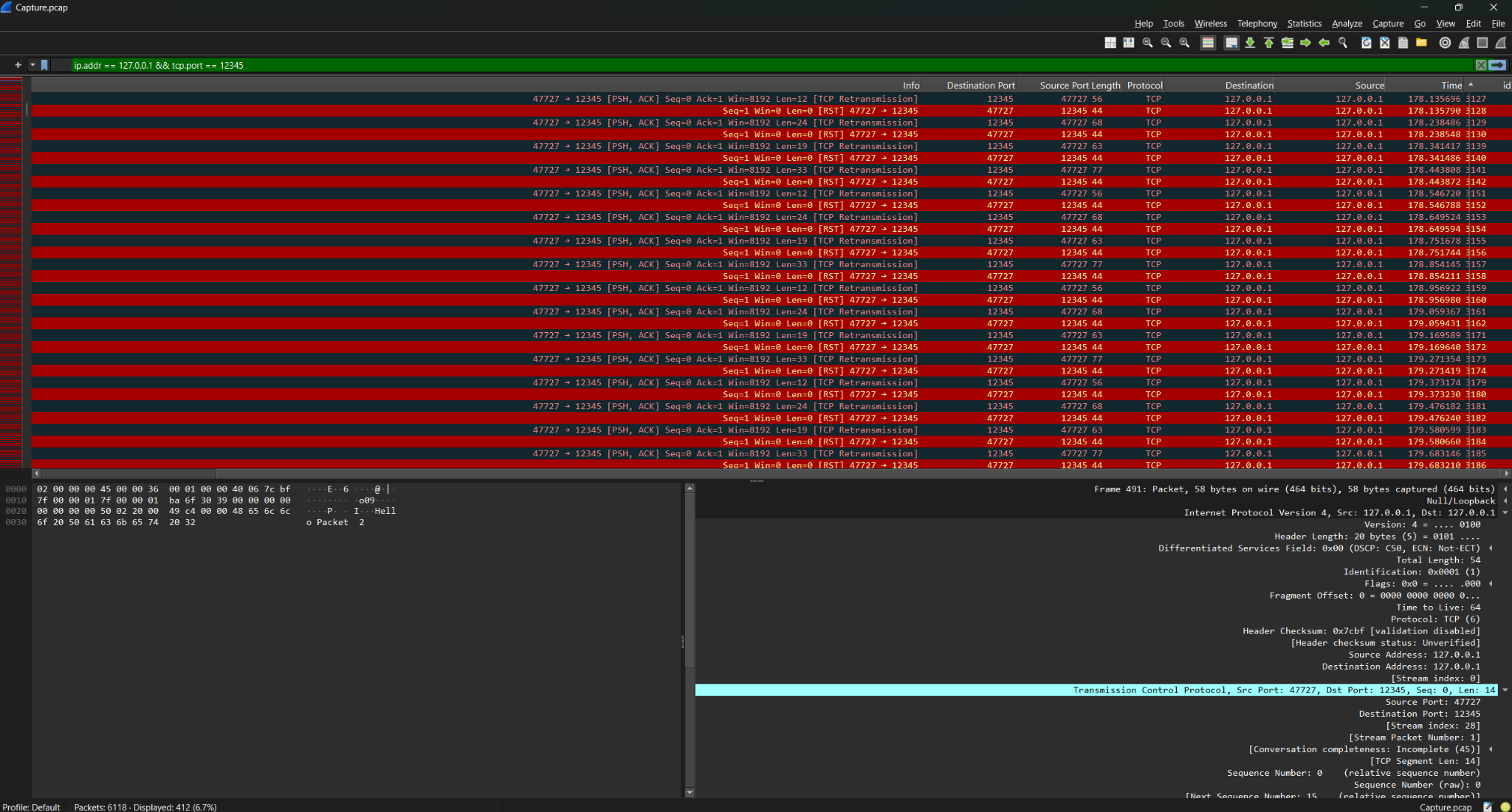
את תהליך האריזה ביצענו באמצעות מחברת ה-Jupyter שסופקה לנו. התהליך מדמה את מה שקורה במחסנית הרשת (Network Stack) במערכת ההפעלה: הסקריפט קרא את ההודעות מתוך קובץ ה-CSV שלנו, וביצע עליו תהליך כימוס:

1. **שכבת האפליקציה (Application Layer)** - הקוד קרא כל שורה מקובץ ה-CSV ושלף את תוכן ה-message כ-Payload גולמי (בתים).
2. **שכבת התעבורה (Transport Layer - TCP) -** יצרנו כותרת (TCP Header) באופן ידני. הכותרת כללה את פורט המקור והיעד (מה-CSV), מספרי רצף (Sequence Numbers) אקראיים, ודגלים (Flags) מתאימים. התוצאה הייתה סגמנט TCP המכיל את המידע מהשכבה הקודמת.
3. **שכבת הרשת (Network Layer – IP)** הסגמנט נארז בתוך חבילת IPv4 Packet. הגדרנו ידנית את כתובות המקור והיעד כ-127.0.0.1 (localhost) כדי לאפשר תעבורה פנימית, וחישבנו את אורך החבילה הכולל Header + Payload.

**תיאור והסבר של תהליך הלכידה**:

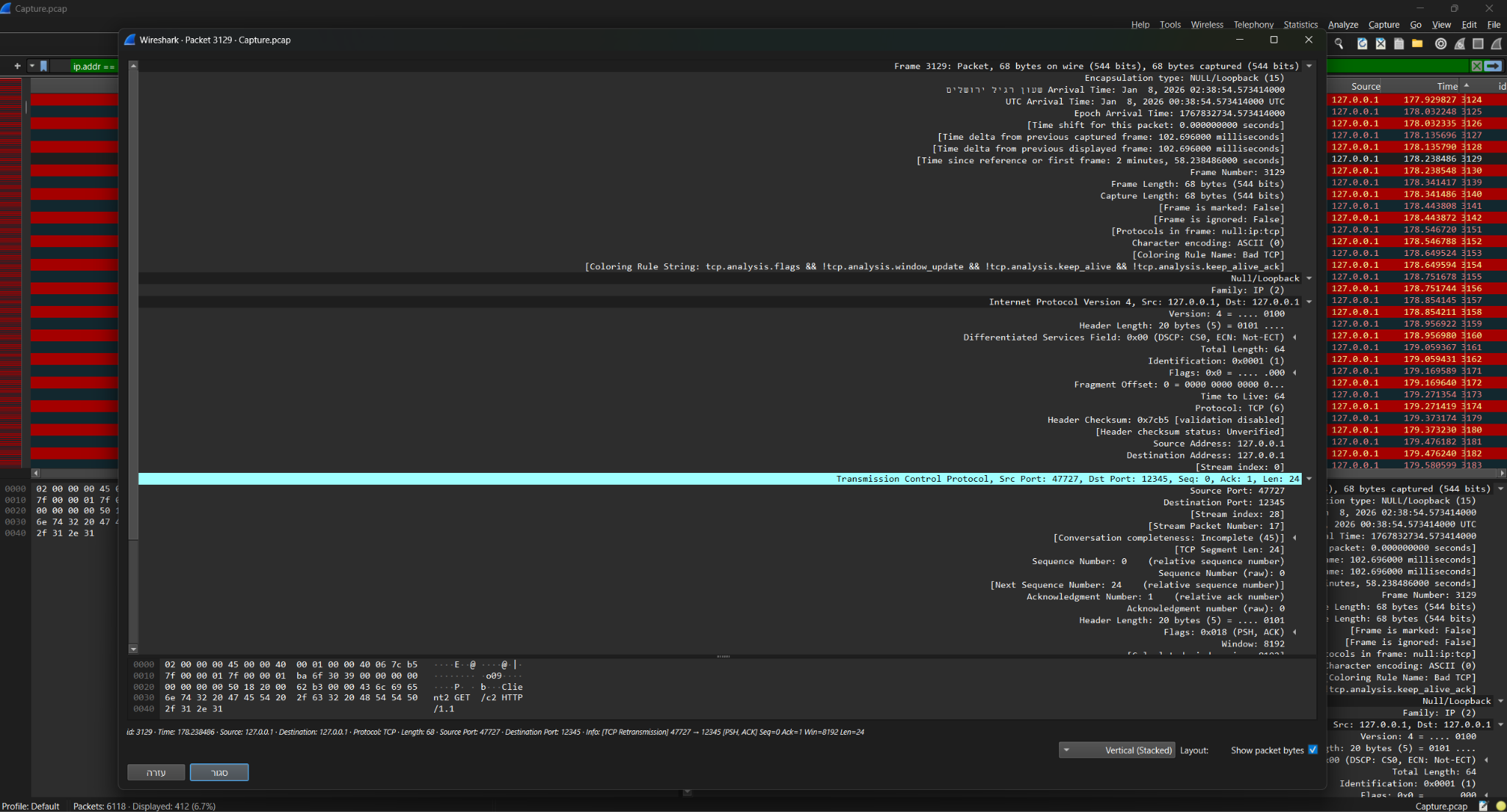
1. מכיוון שהתעבורה היא פנימית (Localhost), בחרנו להאזין לממשק Adapter for Loopback traffic capture.
2. כדי לסנן רעשי רקע ולראות רק את המנות שלנו, הפעלנו את הפילטר: tcp.port == 12345 (פורט היעד שהוגדר בסימולציה).
3. לאחר שהלכידה הייתה פעילה, הרצנו את תא הסימולציה במחברת, אשר הזריק את החבילות שיצרנו לממשק הרשת בזמן אמת.

השרת מאזין על פורט 12345. בצד הלקוח מערכת ההפעלה יכולה להשתמש בפורט זמני נוסף/שונה בחלק מהחבילות, ולכן ברשימת המנות מופיעים גם פורטים כמו 47727 בכיוון ההפוך.



אנו רואים התאמה מלאה בין ה-CSV ללכידה:

* בצילום רואים תקשורת 127.0.0.1 ל- 127.0.0.1 (Loopback).
* בשכבת TCP רואים dst\_port = 12345, src\_port = 47727 ו-PSH ,ACK שמעיד על שליחת נתונים.
* ב-Packet Bytes מופיע ה-Payload שהוא בדיוק ה-message מה-CSV.
* ה־payload שמופיע הוא "Client2 GET /c2 HTTP/1.1" ולכן הוא תואם לשורה msg\_id=5 בקובץ ה־CSV.



**חלק 2: יצירת יישום וניתוח תעבורה (Chat Application)**

**הסבר כללי על מערכת ומבנה הקוד -** בחלק זה פיתחנו מערכת צ'אט מלאה בתצורת שרת-לקוח (Client-Server) הפועלת על גבי פרוטוקול TCP. המערכת תומכת בריבוי משתמשים (Multi-client) ומאפשרת שיחה בזמן אמת. המערכת מורכבת משני רכיבים:

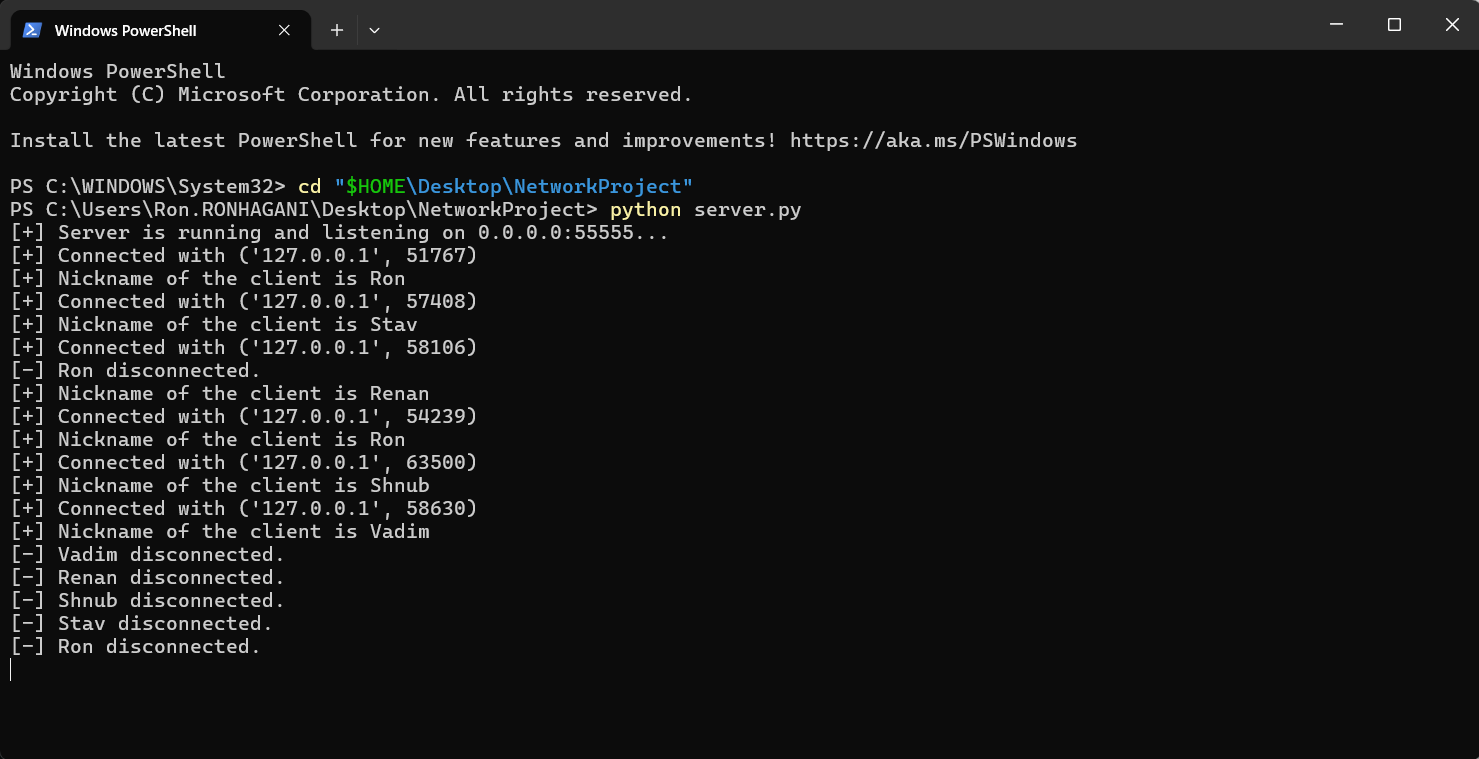
1. **השרת server.py)):** משמש כ"רכזת" התקשורת. הוא מאזין בפורט 55555 ומנהל רשימת לקוחות מחוברים. עבור כל לקוח שמתחבר, השרת פותח תהליכון נפרד (Thread) המאזין להודעות ממנו. ברגע שמתקבלת הודעה, השרת מבצע "שידור" (Broadcast) ומפיץ אותה לכל שאר הלקוחות המחוברים. כמו כן, השרת מטפל במקרי קצה כמו התנתקות פתאומית של לקוח ומעדכן את שאר המשתתפים.
2. **הלקוח client.py)):** צד המשתמש. במסגרת מטלת הבונוס, שדרגנו את הלקוח לכלול ממשק גרפי (GUI) ידידותי. הלקוח מריץ שני תהליכים במקביל: אחד אחראי על תצוגת הממשק ושליחת הודעות, והשני מאזין ברקע להודעות נכנסות מהשרת ומעדכן את מסך הצ'אט.

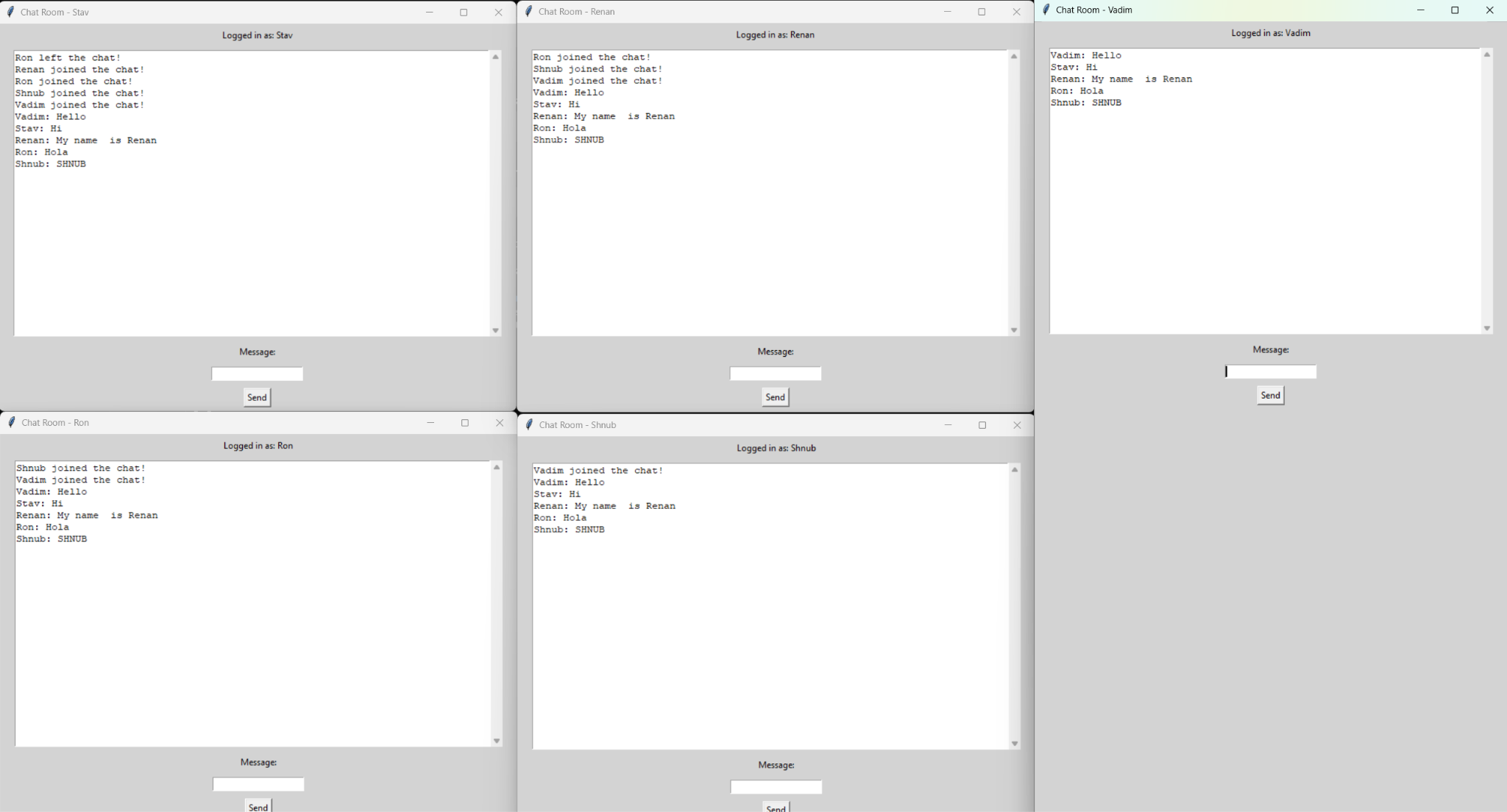
**הוראות התקנה והרצה -** היישום נכתב ב-Python ואינו דורש התקנות מיוחדות:

1. פתח מסוף (Terminal) והרץ את השרת: python server.py. וודא שמתקבלת הודעה שהשרת מאזין.
2. פתח מסוף חדש (או מספר מסופים) והרץ את הלקוח: python client.py.
3. בחלון שיפתח, הזן שם משתמש (Nickname) והתחל לשוחח.

**דוגמאות קלט ופלט** בתמונות המצורפות ניתן לראות הדגמה של המערכת בפעולה:

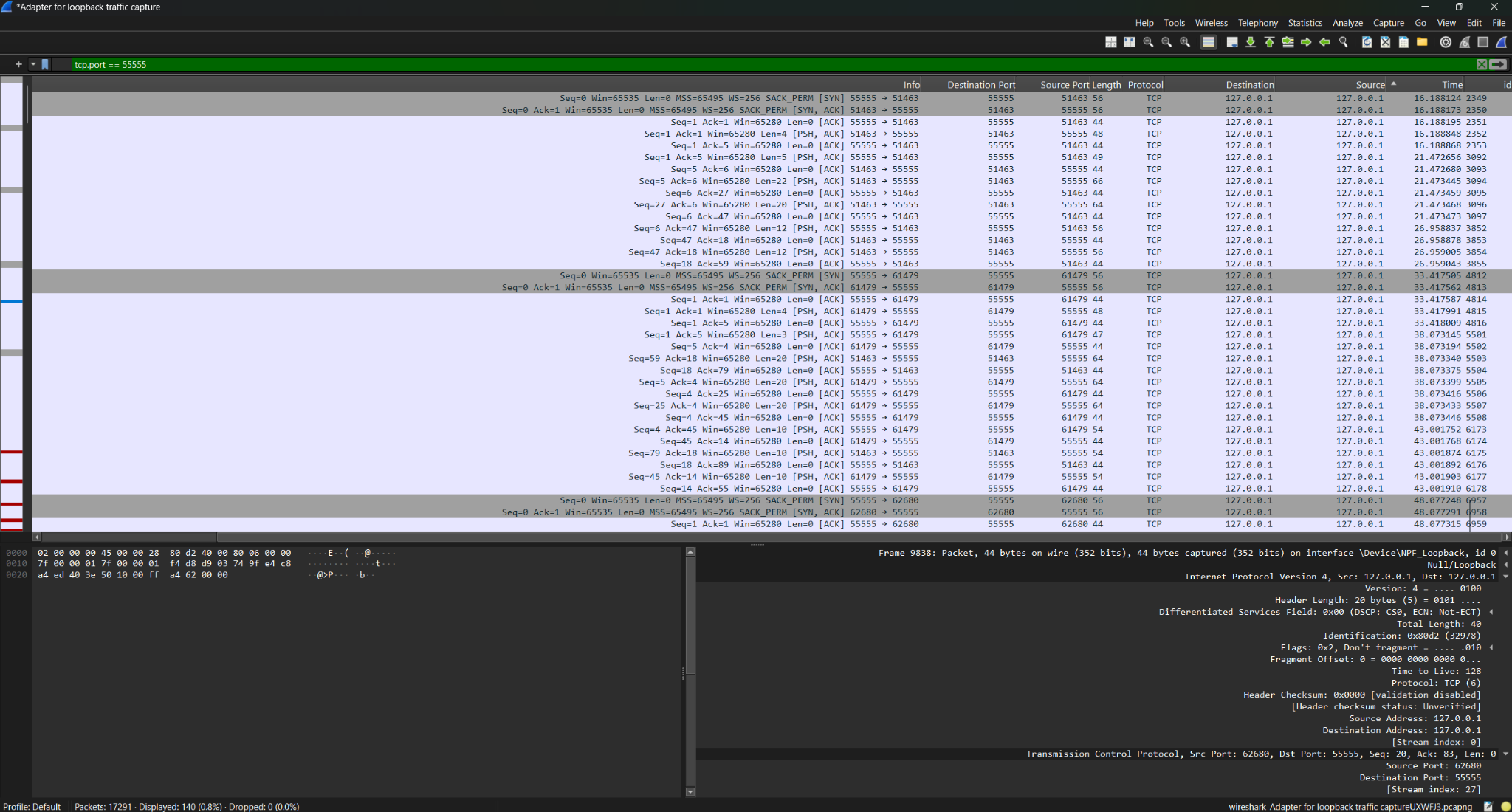
* בצד השרת: לוגים המראים התחברות של לקוחות (עם כתובת IP ופורט) והודעות סטטוס.
* בצד הלקוח: ממשק גרפי המציג את היסטוריית השיחה, כאשר לדוגמא הודעות שנשלחות מלקוח אחד ("Ron") מופיעות מיד אצל הלקוח השני ("Stav").





**ניתוח תעבורה של היישום (Wireshark)** בזמן הרצת הצ'אט, ביצענו לכידת תעבורה ב-Wireshark תחת הפילטר tcp.port == 55555. מהניתוח עולה:

1. יצירת החיבור: ניתן לראות את תהליך "לחיצת היד המשולשת" (3-Way Handshake - SYN, SYN-ACK, ACK) בתחילת הלכידה, כאשר הלקוח יוצר קשר עם השרת.
2. העברת הנתונים: הודעות הצ'אט (למשל הטקסט "Hello") עוברות כסגמנטים של TCP עם דגל Push, המנחה את המקבל להעביר את המידע מיד לשכבת האפליקציה.
3. סיום החיבור: בעת סגירת חלון הצ'אט, נשלחת בקשת FIN לסגירה מסודרת של ה-Socket, והשרת מסיר את הלקוח מהרשימה.



**חלק 3: תיאור שימוש בבינה מלאכותית (AI)**

**מטרות השימוש** השתמשנו בAI ככלי עזר טכני לייעול הפיתוח בשלושה מוקדים:

1. **יצירת נתונים:** יצירה מהירה של שורות ה- CSV הודעות HTTP מגוונות) כדי לחסוך כתיבה ידנית סיזיפית.(
2. **דיבוג ותיקון:** איתור ותיקון באגים לוגיים בקוד השרת (כגון טיפול בניתוק לקוח) ומניעת קריסות.
3. **פיתוח הבונוס:** עזרה בכתיבת המעטפת הגרפית (GUI) ללקוח באמצעות ספריית, tkinter כדי לשדרג את חווית המשתמש.

**דוגמאות פרומפטים:**

* **ליצירת ה-CSV –** "צור קובץ CSV המדמה תעבורת HTTP עם עמודות msg\_id, app\_protocol, src\_port, dst\_port, message, timestamp וכלול לפחות 20 שורות של בקשות ותשובות הגיוניות."
* **לבונוס**" **- (GUI)** יש לי קוד לקוח TCP בסיסי בPython-. הפוך אותו ללקוח עם ממשק גרפי (GUI) פשוט לצ'אט הכולל חלון הודעות ושורת הקלדה."