דוח מסכם: ניתוח תעבורה ואריזת מנות בפרוטוקול TCP/IP

מגישים: שחר סידון, שי כהן, איסבל דייצ׳ב | **קורס:** רשתות תקשורת מחשבים

1. מבוא ומטרת הפרויקט

חלק זה של הפרויקט עוסק במימוש ידני של תהליך אריזת הנתונים (Encapsulation) במודל ה-TCP/IP. המטרה היא להדגים הבנה מעמיקה של מבנה הכותרות (Headers) בשכבות השונות, ולבצע ניתוח תעבורה (Traffic Analysis) של פרוטוקול HTTP. השתמשנו בקוד של מחברת jupyter כדי לבנות את המנות (Raw Sockets), להזריק אותן לממשק הרשת המקומי (Loopback), וללכוד אותן באמצעות Wireshark.

2. ניתוח תרחיש היישום (קובץ ה-CSV)

הקלט לפרויקט הוא קובץ נתונים (group212252217_http_input.csv) המייצג תעבורת רשת בשכבת היישום אשר נוצר בעזרת סוכן AI.

פרוטוקול HTTP

הנתונים מבוססים על פרוטוקול **HTTP (HyperText Transfer Protocol)** בגרסה 1.1. זהו פרוטוקול במודל שרת-לקוח (Client-Server), הפועל בצורה של request-response. הפרוטוקול הוא Stateless, כלומר כל בקשה עומדת בפני עצמה, אך השימוש ב-Cookies או Tokens מאפשר ניהול Session.

ניתוח תוכן ה-CSV

הקובץ מכיל תרחיש גלישה מלא באתר מסחר אלקטרוני. להלן ניתוח של המהלכים המרכזיים בקובץ:

- **זיהוי הישויות:**
  - **הלקוח (client_browser):** פועל מפורט מקור **12345**.
  - **השרת (web server):** פועל בפורט היעד הסטנדרטי **80** (HTTP).
- **סוגי הבקשות (Methods):**
  - GET: בקשת משאבים.
  - POST: שליחת מידע לשרת.
- **קודי תגובה (Status Codes):**
  - 200 OK:הרוב המוחלט של הבקשות נענו בהצלחה.
  - 404 not found
  - 304 not modified: בהודעה מס' 76, השרת מעדכן את הלקוח שהמשאב לא השתנה ולכן ניתן להשתמש בגרסה השמורה במטמון (Cache).
  - 201 created: בהודעה מס' 20, אישור על יצירת משאב חדש (למשל, רשומה חדשה ב-LOG).

3. תהליך אריזת המנות (Encapsulation)

תהליך האריזה בוצע באמצעות מחברת Jupyter, בשימוש במחלקת RawTCPTransport. הקוד ביצע סימולציה של ירידת הנתונים במורד המודל השכבתי:

שכבת התעבורה (Transport Layer - TCP)

הקוד יצר כותרת TCP (באורך 20 בתים) עבור כל הודעה מה-CSV.

- **פורטים:** הוגדרו ידנית (Source: 12345, Dest: 5074).
- **Checksum:** בוצע חישוב מתמטי לאימות שלמות המידע, הכולל "Pseudo Header" (כתובות ה-IP ופרטי הפרוטוקול) כדי לקשור בין שכבת הרשת לשכבת התעבורה.
- **דגלים (Flags):** נקבעו לערך 0x18 (שהם psh+ack). הסבר מורחב על כך מופיע בפרק הניתוח.

שכבת הרשת (Network Layer - IP)

כותרת ה-TCP נעטפה בכותרת IPv4.

- **כתובות:** מכיוון שהסימולציה היא מקומית, הוגדרו כתובות 127.0.0.1 (Loopback).
- **פרוטוקול:** שדה הפרוטוקול הוגדר כ-6 (TCP).

4. תהליך הלכידה והניתוח ב-Wireshark

התעבורה נלכדה בממשק ה-Loopback תוך שימוש במסנן tcp.port == 5074

ניתוח מנה בודדת (Packet Analysis)

בתצלום המסך המצורף ניתן לראות את המבנה השכבתי שנלכד:

1. **Frame:** השכבה הפיזית/ערוץ (סה"כ הבתים שנלכדו).
2. **Internet Protocol (IP):** מראה את כתובות המקור והיעד (127.0.0.1).
3. **Transmission Control Protocol (TCP):** מראה את הפורטים (12345 -> 5074) ואת הדגלים.
4. **Data (Payload):** מכיל את טקסט ה-HTTP המקורי (למשל: GET /index.html HTTP/1.1), מה שמוכיח שהאריזה בוצעה בהצלחה והמידע עבר בשלמותו.

5. ניתוח מעמיק: משמעות הדגלים (Flags) ותופעת ה-RST

מדוע נשלחו PSH ו-ACK?

במקום לבצע תהליך "לחיצת יד משולשת" (Three-Way Handshake: SYN -> SYN-ACK -> ACK) עבור כל הודעה בסימולציה, המנות הוזרקו כאילו הן חלק משיחה שכבר קיימת ומעבירה מידע.

- **ACK (Acknowledgment):** מציין ששדה ה-Acknowledgment Number בכותרת הוא תקף. ב-TCP, כמעט כל מנה אחרי שלב יצירת הקשר נושאת דגל זה.
- **PSH (Push - דגל הדחיפה):** דגל זה מורה למקבל (מערכת ההפעלה בצד השני) **לא לשמור את המידע ב-Buffer** (במחסנית ההמתנה), אלא להעביר אותו מיד לשכבת האפליקציה. מכיוון שאנו שולחים פקודות HTTP שדורשות תגובה מיידית, השימוש ב-PUSH הוא הגיוני ונכון. לכן, ראינו את הצירוף [PSH ACK] על המנות שיצאו מהקוד שלנו.

אחת התופעות הבולטות בלכידה היא שלאחר כל מנה שהזרקנו (PSH ACK), הופיעה מיד מנה אדומה עם דגל **RST**.

ההסבר הטכני:

1. העדר מאזין: הקוד שלח מנות לפורט 5074 במחשב המקומי, אך בפועל לא הופעל שרת המאזין לפורט זה.

2. חוסר בתיאום חיבור (State Mismatch): גם לו היה שרת פעיל, הקוד שלנו ביצע הזרקת מנות עם דגלי PSH+ACK מבלי לבצע קודם לכן את תהליך "לחיצת היד המשולשת".

3. תגובת מערכת ההפעלה: כאשר מערכת ההפעלה (Kernel) מקבלת מנה לפורט סגור או מנה שאינה שייכת לחיבור קיים (Unsolicited Packet), הפרוטוקול מחייב שליחת הודעת RST כדי להודיע לשולח שיש לסגור את החיבור.

מסקנה: הופעת ה-RST היא הוכחה לכך שהצלחנו לייצר תעבורה, אך מכיוון שלא ביצענו לחיצת יד אמיתית מול ה-Kernel של מערכת ההפעלה, המערכת זיהתה את המנות שלנו כ"פולשות" או שגויות ודחתה אותן. בסימולציה זו, אנו מתעלמים מה-RST כיוון שמטרתנו הייתה רק להדגים את יכולת האריזה והשידור, ולא לנהל שיחה אמיתית מול דפדפן חי.

No.	Time	Source	Destination	Protocol	Length	Info
351	23.260163	127.0.0.1	127.0.0.1	TCP	62	[TCP Retransmission] 5074 → 12345 [PSH, ACK] Seq=1 Ack=1 Win=8192 Len=18
352	23.260206	127.0.0.1	127.0.0.1	TCP	44	12345 → 5074 [RST] Seq=1 Win=0 Len=0
353	23.361940	127.0.0.1	127.0.0.1	TCP	59	[TCP Retransmission] 5074 → 12345 [PSH, ACK] Seq=1 Ack=1 Win=8192 Len=15
354	23.361974	127.0.0.1	127.0.0.1	TCP	44	12345 → 5074 [RST] Seq=1 Win=0 Len=0
355	23.464677	127.0.0.1	127.0.0.1	TCP	69	[TCP Retransmission] 5074 → 12345 [PSH, ACK] Seq=1 Ack=1 Win=8192 Len=25
356	23.464712	127.0.0.1	127.0.0.1	TCP	44	12345 → 5074 [RST] Seq=1 Win=0 Len=0
357	23.572767	127.0.0.1	127.0.0.1	TCP	59	[TCP Retransmission] 5074 → 12345 [PSH, ACK] Seq=1 Ack=1 Win=8192 Len=15
358	23.572857	127.0.0.1	127.0.0.1	TCP	44	12345 → 5074 [RST] Seq=1 Win=0 Len=0
359	23.675413	127.0.0.1	127.0.0.1	TCP	69	[TCP Retransmission] 5074 → 12345 [PSH, ACK] Seq=1 Ack=1 Win=8192 Len=25
360	23.675470	127.0.0.1	127.0.0.1	TCP	44	12345 → 5074 [RST] Seq=1 Win=0 Len=0
361	23.780038	127.0.0.1	127.0.0.1	TCP	59	[TCP Retransmission] 5074 → 12345 [PSH, ACK] Seq=1 Ack=1 Win=8192 Len=15
362	23.780105	127.0.0.1	127.0.0.1	TCP	44	12345 → 5074 [RST] Seq=1 Win=0 Len=0
363	23.883036	127.0.0.1	127.0.0.1	TCP	70	[TCP Retransmission] 5074 → 12345 [PSH, ACK] Seq=1 Ack=1 Win=8192 Len=26
364	23.883073	127.0.0.1	127.0.0.1	TCP	44	12345 → 5074 [RST] Seq=1 Win=0 Len=0
365	23.984986	127.0.0.1	127.0.0.1	TCP	59	[TCP Retransmission] 5074 → 12345 [PSH, ACK] Seq=1 Ack=1 Win=8192 Len=15
366	23.985022	127.0.0.1	127.0.0.1	TCP	44	12345 → 5074 [RST] Seq=1 Win=0 Len=0
367	24.088367	127.0.0.1	127.0.0.1	TCP	68	[TCP Retransmission] 5074 → 12345 [PSH, ACK] Seq=1 Ack=1 Win=8192 Len=24
368	24.088427	127.0.0.1	127.0.0.1	TCP	44	12345 → 5074 [RST] Seq=1 Win=0 Len=0
369	24.191434	127.0.0.1	127.0.0.1	TCP	59	[TCP Retransmission] 5074 → 12345 [PSH, ACK] Seq=1 Ack=1 Win=8192 Len=15
370	24.191503	127.0.0.1	127.0.0.1	TCP	44	12345 → 5074 [RST] Seq=1 Win=0 Len=0
371	24.293258	127.0.0.1	127.0.0.1	TCP	72	[TCP Retransmission] 5074 → 12345 [PSH, ACK] Seq=1 Ack=1 Win=8192 Len=28
372	24.293296	127.0.0.1	127.0.0.1	TCP	44	12345 → 5074 [RST] Seq=1 Win=0 Len=0
373	24.396006	127.0.0.1	127.0.0.1	TCP	67	[TCP Retransmission] 5074 → 12345 [PSH, ACK] Seq=1 Ack=1 Win=8192 Len=23
374	24.396055	127.0.0.1	127.0.0.1	TCP	44	12345 → 5074 [RST] Seq=1 Win=0 Len=0
375	24.498528	127.0.0.1	127.0.0.1	TCP	74	[TCP Retransmission] 5074 → 12345 [PSH, ACK] Seq=1 Ack=1 Win=8192 Len=30
376	24.498573	127.0.0.1	127.0.0.1	TCP	44	12345 → 5074 [RST] Seq=1 Win=0 Len=0
377	24.601386	127.0.0.1	127.0.0.1	TCP	59	[TCP Retransmission] 5074 → 12345 [PSH, ACK] Seq=1 Ack=1 Win=8192 Len=15
378	24.601427	127.0.0.1	127.0.0.1	TCP	44	12345 → 5074 [RST] Seq=1 Win=0 Len=0
379	24.706126	127.0.0.1	127.0.0.1	TCP	73	[TCP Retransmission] 5074 → 12345 [PSH, ACK] Seq=1 Ack=1 Win=8192 Len=29
380	24.706182	127.0.0.1	127.0.0.1	TCP	44	12345 → 5074 [RST] Seq=1 Win=0 Len=0
381	24.807875	127.0.0.1	127.0.0.1	TCP	59	[TCP Retransmission] 5074 → 12345 [PSH, ACK] Seq=1 Ack=1 Win=8192 Len=15
382	24.807920	127.0.0.1	127.0.0.1	TCP	44	12345 → 5074 [RST] Seq=1 Win=0 Len=0
383	24.910743	127.0.0.1	127.0.0.1	TCP	79	[TCP Retransmission] 5074 → 12345 [PSH, ACK] Seq=1 Ack=1 Win=8192 Len=35
384	24.910793	127.0.0.1	127.0.0.1	TCP	44	12345 → 5074 [RST] Seq=1 Win=0 Len=0

תיאור: רשימת המנות שנלכדו. ניתן לראות את רצף בקשות ה-HTTP ואת תגובות ה-RST באדום.

```
▸ Null/Loopback
▸ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▾ Transmission Control Protocol, Src Port: 5074, Dst Port: 12345, Seq: 1, Ack: 1, Len: 18
    Source Port: 5074
    Destination Port: 12345
    [Stream index: 10]
    [Stream Packet Number: 109]
  ▸ [Conversation completeness: Incomplete (40)]
    [TCP Segment Len: 18]
    Sequence Number: 1    (relative sequence number)
    Sequence Number (raw): 0
    [Next Sequence Number: 19    (relative sequence number)]
    Acknowledgment Number: 1    (relative ack number)
    Acknowledgment number (raw): 0
    0101 .... = Header Length: 20 bytes (5)
  ▸ Flags: 0x018 (PSH, ACK)
    Window: 8192
    [Calculated window size: 8192]
    [Window size scaling factor: -1 (unknown)]
    Checksum: 0xb54e [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  ▸ [Timestamps]
  ▸ [SEQ/ACK analysis]
    [Client Contiguous Streams: 0]
    [Server Contiguous Streams: 1]
    TCP payload (18 bytes)
    Retransmitted TCP segment data (18 bytes)

0000  02 00 00 00 45 00 00 3a  00 01 00 00 40 06 7c bb   ····E··: ····@·|·
0010  7f 00 00 01 7f 00 00 01  13 d2 30 39 00 00 00 00   ........ ··09····
0020  00 00 00 00 50 18 20 00  b5 4e 00 00 47 45 54 20   ····P··· ·N··GET ·
0030  2f 62 6c 6f 67 20 48 54  54 50 2f 31 2e 31         /blog HT TP/1.1

No.: 351 · Time: 23.260163 · Source: 127.0.0.1 · Destination: 127.0.0.1 · Protocol: TCP · Length: 62 · Info: [TCP Retransmission] 5074 → 12345 [PSH, ACK] Seq=1 Ack=1 Win=8192 Len=18

☑ Show packet bytes      Layout: Vertical (Stacked)

                                                   Close    Help
```

תיאור: מבט על ה-Header של שכבת ה-TCP ועל תוכן ה-Data המציג את בקשת ה-HTTP קריאה.

חלק 2: פיתוח יישום צ'אט (Client-Server) וניתוח תעבורה

1. סקירה כללית ומבנה המערכת

במסגרת הפרויקט פותחה מערכת צ'אט מבוססת Sockets בפרוטוקול **TCP**, הפועלת בארכיטקטורת **שרת-לקוח**. המערכת מאפשרת למספר משתמשים להתחבר במקביל לשרת מרכזי, להזדהות באמצעות שם ייחודי, ולנהל שיחות פרטיות בזמן אמת.

המערכת פותחה בשפת **Python** ועומדת בדרישה לא להשתמש בספריות תקשורת מוכנות, אלא במימוש ישיר של socket ושימוש ב-threading לריבוי משימות.

מבנה הקוד והקבצים:

א. צד השרת (server.py) השרת משמש כצומת המרכזי המנהל את ה-State של המערכת.

- **ניהול משתמשים:** השרת מחזיק מילון (clients) הממפה בין שם המשתמש (String) לבין אובייקט ה-Socket הפעיל שלו.
- **מקביליות:** כדי לעמוד בדרישה לתמוך ב-5 לקוחות לפחות בו-זמנית, השרת משתמש בספריית threading. הפונקציה start_server מאזינה לחיבורים נכנסים, ועבור כל לקוח חדש נפתח תהליכון (Thread) נפרד המריץ את הפונקציה handle_client.
- **פרוטוקול יישום:** הוגדר פרוטוקול טקסטואלי פשוט בפורמט TargetName:Message. השרת מפענח את ההודעה (parse_chat_message), מוודא תקינות, ומנתב את ההודעה ל-Socket של הנמען.
- **טיפול בשגיאות:** השרת מזהה ניתוקים פתאומיים (ConnectionResetError), מסיר את המשתמש מהזיכרון ומעדכן את שאר המחוברים. כמו כן, במקרים שונים שכוללים פעולות על משתמשים יש טיפול במשתמשים שאינם פעילים.

ב. צד הלקוח (client.py) הלקוח הוא יישום CLI המורכב משני רכיבים עיקריים הפועלים במקביל:

1. **Thread האזנה (start_receiving):** רץ ברקע וממתין להודעות נכנסות מהשרת. ברגע שמתקבלת הודעה, היא מוצגת למשתמש. מנגנון זה מאפשר קבלת הודעות גם בזמן שהמשתמש מקליד הודעה חדשה.
2. **Thread ראשי (ממשק משתמש):** אחראי על קבלת קלט מהמקלדת, אימות הפורמט ושליחת ההודעה לשרת.

א. תצורות הפעלה (Modes of Operation)

המערכת פותחה כך שהיא תומכת בשני אופני הפעלה שונים, המשתמשים באותה לוגיקת תקשורת בסיסית:

1. **מצב CLI (ממשק שורת פקודה):** ממשק טקסטואלי קליל ומהיר, הרץ ישירות בחלון המסוף (Terminal/Console).
2. **מצב GUI (ממשק גרפי):** ממשק משתמש ויזואלי מלא, הכולל חלונות, כפתורים ותצוגה צבעונית.

ב. צד השרת (Server)

השרת משמש כצומת המרכזי המנהל את ה-State של המערכת.

- **ניהול משתמשים:** השרת מחזיק בזיכרון מילון (clients) הממפה בין שם המשתמש (String) לבין אובייקט ה-Socket הפעיל שלו, ומאפשר שליפה מהירה לניתוב הודעות.
- **מקביליות:** כדי לתמוך במספר רב של לקוחות בו-זמנית (Multi-Client), השרת משתמש בספריית threading. הפונקציה start_server מאזינה לחיבורים נכנסים, ועבור כל לקוח חדש נפתח **Thread נפרד** המריץ את הפונקציה handle_client. כך, פעילות של לקוח אחד לא חוסמת את האחרים.
- **פרוטוקול יישום:** התקשורת מתבצעת בפרוטוקול טקסטואלי בפורמט TargetName:Message. השרת מפענח את ההודעה באמצעות parse_chat_message, מוודא תקינות, ומנתב את ההודעה ל-Socket של הנמען.
- **טיפול בשגיאות:** השרת מזהה ניתוקים פתאומיים והודעות לא תקינות, מסיר את המשתמש מהרשימה במידת הצורך ומעדכן את שאר המחוברים.

ג. צד הלקוח (Client)

הלקוח בנוי בצורה מודולרית המפרידה בין הלוגיקה לבין התצוגה. הלקוח פועל באמצעות שני תהליכונים (Threads) הפועלים במקביל:

1. **Thread האזנה (start_receiving):** תהליכון הרץ ברקע ומאזין באופן רציף להודעות נכנסות מהשרת. ברגע שמתקבלת הודעה, היא מועברת לתצוגה. מנגנון זה קריטי כדי לאפשר קבלת הודעות בזמן אמת גם כאשר המשתמש מקליד הודעה חדשה.
2. **Thread ראשי (ממשק משתמש - Main UI):** אחראי על קבלת הקלט מהמשתמש ושליחתו לשרת.
   - **במצב CLI:** הלולאה הראשית משתמשת ב-input() כדי לקרוא פקודות מהמקלדת.
   - **במצב GUI:** הלולאה הראשית מנוהלת על ידי tkinter, המגיבה לאירועי לחיצה והקלדה.

2. הוראות התקנה והרצה

המערכת פותחה בשפת **Python** וניתנת להרצה בשתי תצורות: ממשק גרפי (GUI) וממשק שורת פקודה (CLI) בסיסי.

א. דרישות קדם והתקנה

1. יש לוודא שמותקן **Python 3.6** ומעלה על המחשב.
2. יש לשמור את כל קבצי הפרויקט (server.py, client.py, server_gui.py, client_gui.py, requirements.txt) באותה תיקייה.
3. **התקנת ספריות חיצוניות (עבור ה-GUI):** הממשק הגרפי עושה שימוש בספרייה python-bidi לצורך תמיכה תקינה בהצגת עברית. כדי להתקין את התלויות, יש לפתוח מסוף (Terminal/CMD) בתיקיית הפרויקט ולהריץ:
   - pip install -r requirements.txt

ב. הרצת המערכת במצב גרפי (GUI - מומלץ)

מצב זה מפעיל את המערכת עם ממשק משתמש ויזואלי הכולל חלונות הגדרה, צבעים ותמיכה באימוג'י.

1. הפעלת השרת:

- פתח טרמינל והרץ את הפקודה: python server_gui.py
- ייפתח חלון הגדרות ("Server Config"). ודא שהכתובת היא 127.0.0.1 והפורט 55555, ולחץ על **"START SERVER"**.

- ייפתח חלון הניהול ("System Logs") המציג את המשתמשים המחוברים בזמן אמת.

2. הפעלת לקוח (Clients):

- פתח טרמינל חדש (נפרד מהשרת) והרץ: python client_gui.py
- בחלון ההתחברות ("Login"), הזן שם משתמש (Nickname) ולחץ על **"ENTER"**.
- חזור על פעולה זו בחלונות נוספים כדי לדמות משתמשים נוספים.

ג. פתרון תקלות ומצב CLI (גיבוי)

במידה והממשק הגרפי אינו עולה כראוי (למשל עקב בעיות בספריית tkinter במערכות הפעלה מסוימות או חוסר בהתקנת ספריות), ניתן להריץ את המערכת במצב **שורת פקודה (CLI)** מלא. מצב זה משתמש באותה לוגיקה בדיוק אך ללא הגרפיקה:

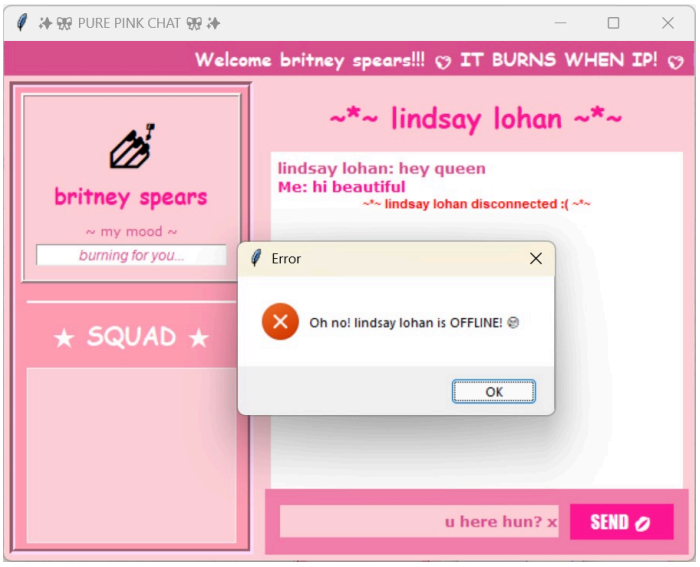
1. **הפעלת שרת CLI:** הריצו את python server.py.
2. **הפעלת לקוח CLI:** הריצו את python client.py ועקבו אחר ההוראות בטקסט.

3. דוגמאות קלט ופלט

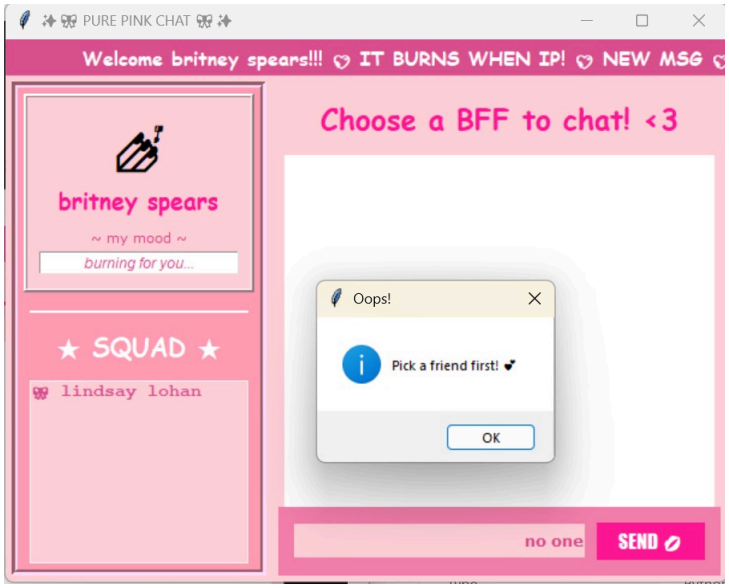
להלן תרחישים במערכת המדגימים את הלוגיקה שמומשה:

תרחיש א': שליחת הודעה תקינה

תרחיש ב': טיפול בשגיאת "משתמש לא מחובר"



תרחיש ג': טיפול בשגיאת "שליחה ללא יעד"



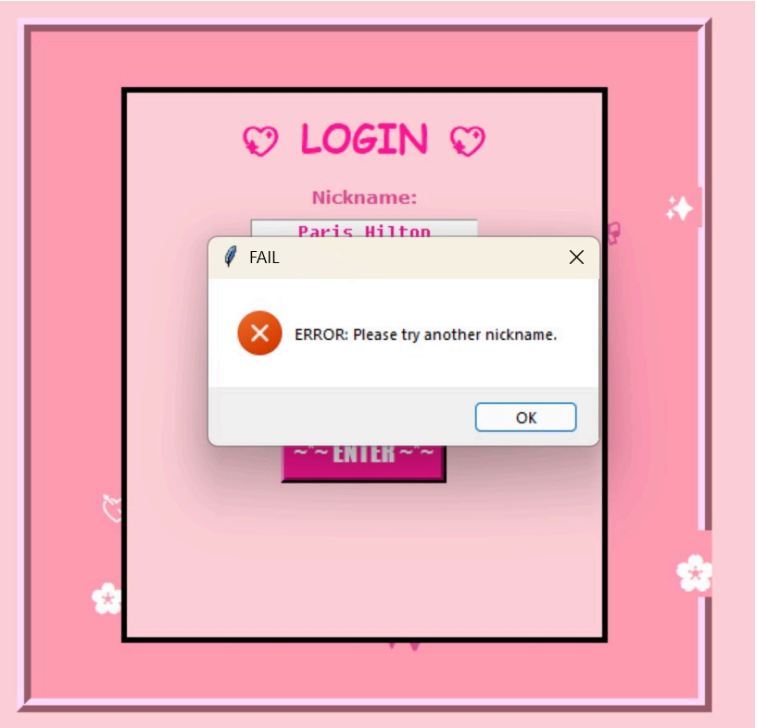
תרחיש ד': טיפול בשגיאת שליחת הודעה ריקה (ניתנת להצגה באפליקציית cli בלבד, מאחר שבאפליקציית gui בוצע טיפול מונע)

```
Host IP (default 127.0.0.1):
Port (default 55555):
Choose a Nickname: NoGui_User
Connected! Usage: 'TargetName: Your Message'
Type 'exit' to quit.
>
ONLINE_USERS:Britney Spears,Paris Hilton,NoGui_User
>
Invalid format! Use: TargetName: Message
> Britney Spears:

System: Name or message cannot be empty.
> > :Hello!

System: Name or message cannot be empty.
> >
```

תרחיש ה': טיפול ביצירת משתמש עם שם קיים



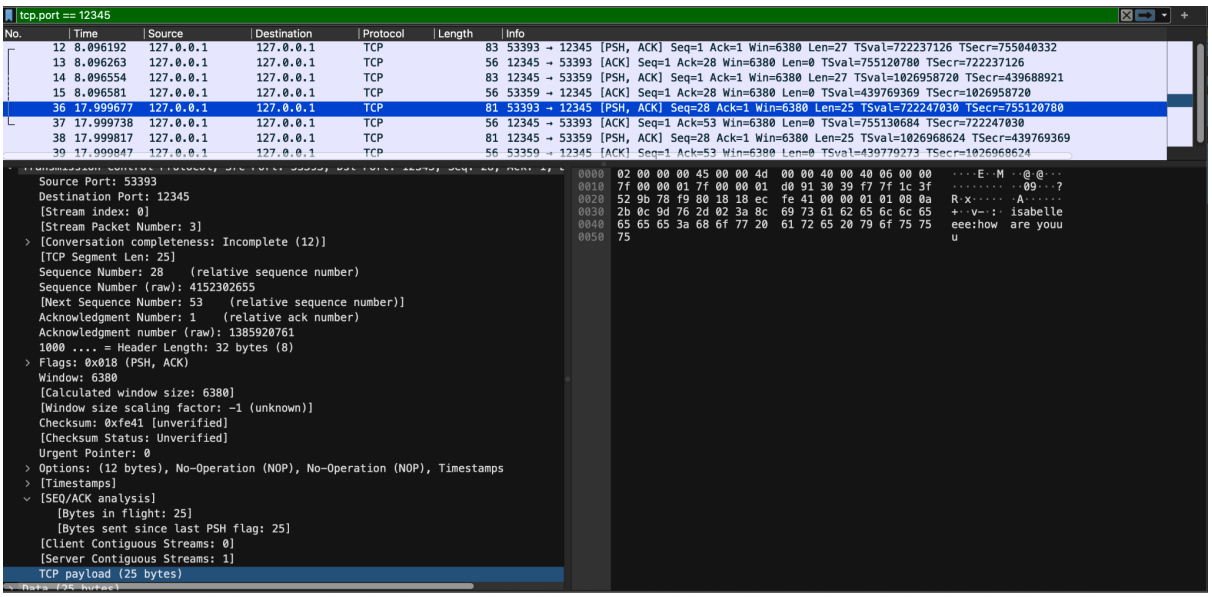
תרחיש ו': טיפול בהתנתקות בלתי צפויה של משתמש

4. ניתוח תעבורת היישום (Wireshark)

במהלך הרצת הצ'אט, לכדנו את התעבורה ב-Wireshark בממשק ה-Loopback תוך שימוש בפילטר tcp.port == 12345.

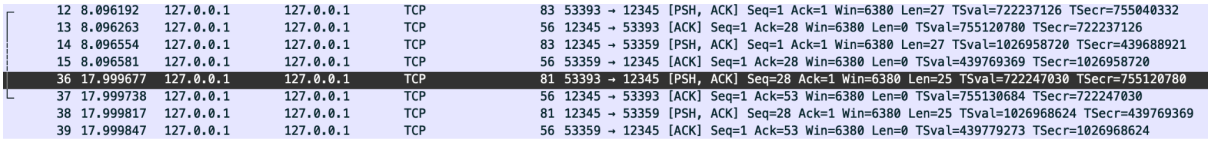
ממצאים וניתוח:

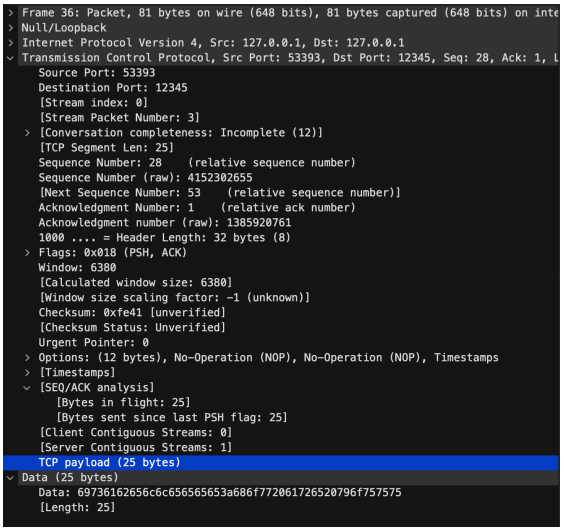
1. **לחיצת היד (Three-Way Handshake):** בניגוד לחלק 1 (בו הזרקנו מנות), כאן רואים תהליך התחברות TCP תקני לחלוטין:
   - [SYN]: הלקוח פונה לשרת (פורט 12345).
   - [SYN, ACK]: השרת מאשר ופותח פורט.
   - [ACK]: הלקוח מאשר חזרה, והחיבור ("Connection") נוצר.
2. **העברת נתונים (PSH, ACK):**
   - כאשר נשלחת הודעה בצ'אט (למשל "Hi"), רואים מנת TCP עם דגל PSH (Push).
   - בחלון ה-Payload (בתחתית המסך ב-Wireshark) ניתן לראות את הטקסט הקריא: UserA: Hi.
   - מכיוון שהשרת מבצע broadcast, נראה מיד לאחר מכן את אותה הודעה נשלחת מהשרת ללקוחות האחרים.
3. **סיום התקשורת (FIN/RST):**
   - כאשר סוגרים את הלקוח, נשלחת מנת FIN (Finish) המסמנת לשרת על רצון לסגור את החיבור, והשרת מגיב ב-ACK וסוגר את המשאבים.



כאן רואים את רשימת המנות (Packets). השורה המודגשת היא המעניינת אותנו:

- **Time:** הזמן שבו המנה נלכדה.
- **Source & Destination:** הכתובת היא 127.0.0.1 בשני הצדדים. זה מוכיח שהלקוח והשרת רצים על אותו מחשב (Localhost), בדיוק כמו בהנחיות הפרויקט.
- **Protocol:** הפרוטוקול הוא **TCP**.
- **Info:** רואים את הפורטים: 53393 -> 12345.
  - 53393: פורט המקור (של הלקוח).
  - 12345: פורט היעד (של השרת) שנבחר באופן ידני בעת פתיחת השרת.
  - **[PSH, ACK]:** הדגלים הללו חשובים. PSH (Push) מסמן שיש מידע שצריך לעבור לאפליקציה, ו-ACK מאשר קבלת מנות קודמות.
  - **Len=25:** אורך המידע (Payload) הוא 25 בתים.





כאן רואים את הפירוט של המנה שנבחרה:

- **Source Port / Destination Port:** רואים שוב את הפורטים (53393 ל-12345).
- **Flags:** הדגל PSH (Push) דולק. זהו הסימן המובהק שבמנה הזו עוברת הודעת טקסט של הצ'אט, ולא רק פקודת בקרה של הרשת.
- **TCP Payload:** מצוין שהגודל הוא 25 bytes.

```
02 00 00 00 45 00 00 4d  00 00 40 00 40 06 00 00   ····E··M ··@·@···
7f 00 00 01 7f 00 00 01  d0 91 30 39 f7 7f 1c 3f   ···· ···· ··09···?
52 9b 78 f9 80 18 18 ec  fe 41 00 00 01 01 08 0a   R·x····· ·A ·····
2b 0c 9d 76 2d 02 3a 8c  69 73 61 62 65 6c 6c 65   +··v-·:· isabelle
65 65 65 3a 68 6f 77 20  61 72 65 20 79 6f 75 75   eee:how  are youu
75                                                 u
```

זה החלק המעניין ביותר לפרויקט, שמראה את **תוכן ההודעה (Layer 7 - Application)**:

- בצד שמאל רואים את הייצוג ההקסדצימלי (מספרים כמו 69 73 61...).
- בצד ימין רואים את התרגום לטקסט קריא (ASCII).
- **הטקסט שנלכד:** isabelleeee:how are youuu
  - זה מוכיח שהפרוטוקול שלך עובד בפורמט שהגדרת: Target:Message.
  - isabelleeee: שם היעד.
  - how are youuu: תוכן ההודעה.

שימוש בבינה מלאכותית (AI)

בפיתוח חלק זה נעשה שימוש ב-AI למטרות הבאות:

1. **מימוש Threading:** נעזרנו במודל כדי להבין את הסינטקס הנכון של ספריית threading בפייתון, וכיצד להעביר ארגומנטים (כמו client_socket) לפונקציית ה-Thread.
   - *דוגמת פרומפט:* "How to handle multiple TCP clients in Python using threads?"
2. **טיפול בשגיאות:** התייעצות כיצד לטפל במצב בו לקוח מתנתק במפתיע מבלי לקרוס את השרת. הפתרון שהוצע ומומש הוא שימוש ב-try...except סביב פקודת ה-recv.