קורס: רשתות תקשורת מחשבים

פרויקט: ניתוח תעבורה בפרוטוקול TCP/IP

מגישים:

אניטה פקר – 31872274

שיר ירמיהו – 322281114

אדיר בייטבאובו – 212766489

מנחת הקורס: ד"ר מרבל רויטל

תאריך הגשה: 24/01/2025

תוכן עניינים

חלק 1 – אריזה ולכידת מנות

1.1 מטרת המטלה

2.1 יצירת קובץ CSV עם הודעות בשכבת היישום

3.1 תיאור והסבר של תהליך הלכידה

4.1 תיאור והסבר של תהליך אריזת מנות

5.1 תיאור והסבר של תעבורה שנלכדה ב-Wireshark

חלק 2 – יצירת יישום וניתוח תעבורה

1.2 הסבר כללי על המערכת

2.2 הוראות התקנה והרצה

3.2 דוגמאות קלט ופלט

4.2 ניתוח תעבורה של היישום ב-Wireshark

חלק 3 – שימוש בבינה מלאכותית

1.3 מטרות השימוש

2.3 דוגמאות פרומפטים

חלק ראשון: אריזת נתונים ולכידת מנות בעזרת Wireshark

1.1. מטרת המטלה:

במסגרת חלק זה בחנו את תהליך אריזת נתונים ולכידת מנות רשת באמצעות Wireshark ומטרתו בעבודה היא לנתח את תעבורת הרשת שנוצרת במהלך הפעלת יישום רשת תוך שימוש בWireshark, בניתוח זה נבחן כיצד הודעות שנשלחות על ידי היישום מועברות ברשת, ואילו פרוטוקולים ושכבות משתתפים בתהליך העברת המידע.
מטרת הניתוח היא להמחיש את הקשר בין המימוש של היישום לבין אופן פעולתו ברמת התקשורת, וכן להעמיק את ההבנה של תפקידי שכבות הרשת השונות בתהליך העברת המידע.

תיאור התהליך שנבדק:

שלב ראשון- הוכן קובץ CSV שנעזרנו בבינה מלאכותית על מנת להכין אותו, המכיל הודעות בשכבת היישום כאשר כל שורה מייצגת הודעת יישום נפרדת הכוללת מזהה הודעה, מקור, יעד, תוכן ההודעה וחותמת זמן.
שלב שני- הכנסנו את קובץ ה־ CSV כקלט למחברת Jupyter שפתחנו בVisual Studio Code .
שלב שלישי- המחברת מדמה את תהליך הקפסולציה ,שבו הודעות משכבת היישום נארזות בכותרות של שכבת התעבורה ולאחר מכן בכותרות של שכבת הרשת.
שלב רביעי- כחלק מההרצה נוצרה תעבורת רשת שלכדנו בזמן אמת באמצעות Wireshark .
התקשורת המדומה מבוססת על פרוטוקול ,TCP והמידע הנשלח הוא טקסטואלי. מטרת התהליך היא להמחיש את מעבר הנתונים בין שכבות המודל, ולהבין כיצד הודעת יישום מופיעה בפועל כחבילת רשת.

תיאור סביבת העבודה:

הלכידה בוצעה על מחשב אחד עם המערכת הפעלה של Windows השתמשנו במחברת גופיט'ר בעזרת ה- Visual Studio Code שנכתבה בשפת פייתון. בWireshark השתמשנו בתפיסה לפי Adapter for loopback traffic capture.

2.1. שלב ראשון- קובץ נתונים- CSV:

הקובץ מכיל הודעות בשכבת היישום , כאשר כל שורה מייצגת הודעת יישום נפרדת שמאופיינת עם מספר מזהה, איזה פרוטוקול מזמים, מי השולח בשכבת האפליקציה, מי המקבל בשכבת האפליקציה, מה ההודעה עצמה וחותמת זמן (בשניות) שמייצגת מתי ההודעה נשלחה יחסית לתחילת התהליך.

השתמשנו בבינה מלאכותית (chat GPT) על מנת להפיק את הקובץ, ואנחנו משתמשים בו כקלט למחברת גופיט'ר כדי שהיא תשתמש בהודעות אלו לבצע את הפירוק לשכבות.

קובץ רלוונטי: group02_input.csv

שלב שני- שימוש במחברת Jupyter:

השתמשנו במחברת על מנת להריץ בה את הקובץ CSV עם ההודעות, להדגים פירוק הודעות משכבת היישום לשכבות התעבורה והרשת.

ניתוח קלט בשכבת האפליקציה- המחברת קוראת את הקובץ, מפרידה כל שורה לשדות ומתייחסת לכל שורה כהודעה נפרדת.
עיבוד נתונים לצורך שליחה ברשת- מומר לביטים ולהופך רק לנתוני ההודעה עצמה
חישוב ברמת שכבות הרשת- חישוב אורך החבילה, קביעת פורמטים, קביעת פרוטוקול.
ניתוח מבנה של חבילת הרשת- המחברת מדמה את תהליך הקפסולציה על ידי יצירה ידנית של כותרות IP וTCP, חיבורן לנתוני היישום והצגת החבילה כרצף בתים בפורמט הקסדצימלי.

קובץ רלוונטי: project_notebook

3.1. תהליך הלכידה ב־Wireshark:

בחירת ממשק רשת- השתמשנו ב- Adapter for loopback traffic capture, הוא מתאים בגלל שהחבילות לא יוצאות לרשת חיצונית ככה שצריך לבדוק את התקשורת הפנימית בתוך אותו מחשב. בWindows ה Wireshark לא יכול לתפוס לבד את התעבורה הפנימית ולכן צריך להשתמש בתפיסה זו.

הגדרת סינונים- השתמשנו בסינון ip.addr == 127.0.0.1 && tcp.port == 12345, בקוד השתמשנו כמספר פורט יעד ב- 12345 כי הוא נמצא בטווח החופשי (לא דורש הרשאות ולא מתנגש עם שירותים מוכרים).
ולכתובות IP היעד והמקור השתמשנו ב- 127.0.0.1 כי הן מייצגות תקשורת מקומית בין הלקוח לשרת הפועלים על אותו מחשב. שימוש בכתובות אלו מאפשר ניתוח של התהליך מבלי לערב תעבורת רשת חיצונית.

שלבי הלכידה:

1. שמירת קובץ ה- CSV בתיקיה שהכתובת שלה רק באנגלית.
2. פתחנו את תוכנת הWireshark ובחרנו את תהליך התפיסה כAdapter for loopback traffic capture .
3. הפעלנו סינון על פי ip.addr == 127.0.0.1 && tcp.port == 12345.
4. פתחנו והחלפנו במחברת ב-step 1 את כתובת התיקייה שבה הקובץ נמצא.
5. הרצנו את המחברת שטענה את הקובץ CSV והדגימה את תהליך הקפסולציה של הודעות יישום לחבילות רשת.
6. עצרנו את הלכידה ב.Wireshark
7. ואת הקובץ שמרנו כ-pcap

קובץ רלוונטי: wireshark_part1.

4.1. שלב שלישי- תהליך אריזת המנות (קפסולציה):

תהליך זה הוא תהליך שבו נתונים הנוצרים בשכבת היישום נארזים בהדרגה בכותרות של שכבות התקשורת השונות, עד להפיכתם לחבילת רשת הניתנת להעברה. תהליך זה מאפשר העברת מידע לוגי של יישומים על גבי תשתית רשת פיזית.

בשלב הראשון, הנתונים נוצרים בשכבת היישום ומייצגים את המידע שהיישום מעוניין להעביר, כגון טקסט או בקשה לוגית. בשלב זה הנתונים עדיין אינם כוללים מידע הקשור לניתוב או לניהול התקשורת, והם מיועדים אך ורק לשימוש היישום עצמו.

בשלב השני, בשכבת התעבורה מתווספת כותרת TCP לנתוני היישום. כותרת זו כוללת מידע כגון פורטי מקור ויעד, המאפשרים זיהוי של היישומים בקצוות התקשורת, וכן מנגנונים התומכים בתקשורת אמינה, כגון מספור מקטעים, אישורי קבלה (ACK) ובקרת זרימה. הוספת כותרת זו הופכת את נתוני היישום למקטע TCP.

בשלב השלישי, בשכבת הרשת מתווספת כותרת IP למקטע ה-TCP. כותרת זו כוללת את כתובות ה- IP של המקור והיעד, וכן מידע נוסף הדרוש לניתוב החבילה ברשת. בשלב זה נקבעת זהות המחשבים המשתתפים בתקשורת, והחבילה מוכנה לעבור בין רכיבי הרשת השונים.

בסיום התהליך מתקבלת חבילת רשת מלאה, הכוללת את כותרות ה- IP וה- TCP לצד נתוני היישום עצמם. חבילה זו מיוצגת כרצף רציף של ביטים והיא מועברת בפועל ברשת בהתאם למנגנוני התקשורת של שכבות הרשת.

5.1. שלב רביעי- ניתוח תעבורה לפי שכבות:

שכבת האפליקציה:

המידע שנשלח בפועל בשכבת היישום הוא מידע טקסטואלי, המייצג את תוכן ההודעות הלוגיות של היישום שהוא עמודת ה message. מידע זה נוצר בשכבת היישום לפני הוספת כותרות התקשורת, והוא מהווה את הנתונים עצמם שאותם היישום מבקש להעביר.

פורמט ההודעות הוא טקסט וניתן לקרוא את תוכן ההודעות בצורה ישירה. ההודעות אינן מקודדות או מוצפנות, אלא מועברות כמחרוזות טקסט פשוטות, המאפשרות צפייה בתוכן ההודעה בחלון הנתונים של Wireshark .

```
Frame 1291: Packet, 83 bytes on wire (664 bits), 83 bytes captured (664 bits) on interface \Device\NPF_Loopback, id 0
                                                                                                        Null/Loopback
                                                                                                     Family: IP (2)
                                                                  Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
                                                                                                  Version: 4 = .... 0100
                                                                                     Header Length: 20 bytes (5) = 0101 ....
                                                                 Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
                                                                                                    Total Length: 79
                                                                                          Identification: 0x0001 (1)
                                                                                              Flags: 0x0 = .... .000
                                                                                  Fragment Offset: 0 = 0000 0000 0000 0...
                                                                                                  Time to Live: 64
                                                                                                   Protocol: TCP (6)
                                                                                     Header Checksum: 0x7ca6 [validation disabled]
                                                                                          [Header checksum status: Unverified]

02 00 00 00 45 00 00 4f  00 01 00 00 40 06 7c a6   ····E··O ····@·|·
7f 00 00 01 7f 00 00 01  48 9d 30 39 00 00 00 00   ........ H·09····
00 00 00 00 50 18 20 00  68 86 00 00 3c 68 74 6d   ····P··· h···<htm
6c 3e 3c 62 6f 64 79 3e  3c 68 31 3e 48 6f 6d 65   l><body> <h1>Home
3c 2f 68 31 3e 3c 2f 62  6f 64 79 3e 3c 2f 68 74   </h1></b ody></ht
6d 6c 3e                                           ml>
```

בצילום מסך זה ניתן לראות את נתוני היישום (payload) של החבילה. בחלון ה־ ASCII מוצג טקסט קריא בפורמט HTML המהווה את תוכן ההודעה שנשלחה בשכבת היישום:

<html><body><h1>Home</h1></body></html>

הודעה זאת הייתה אחת מההודעות שהיו בקובץ CSV שהעלנו למחברת.

שכבת התעבורה:

פרוטוקול TCP משתמש בתהליך 3 way handshake לצורך יצירת חיבור אמין בין לקוח לשרת. עם זאת, בחלק זה של העבודה התעבורה נוצרה באמצעות הדמיית חבילות TCP ולא באמצעות חיבור TCP אמיתי בין יישומים. ולכן, לא נצפה תהליך handshake מלא בלכידת התעבורה.

לכל מקטע TCP מצורפים פורט מקור Src Port ופורט יעד Dst Port, המשמשים לזיהוי היישומים המשתתפים בתקשורת. פורט המקור מזוהה עם היישום השולח, בעוד שפורט היעד מייצג את היישום המקבל. שימוש בפורטים מאפשר למחשב לנהל מספר חיבורים במקביל.

```
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
                                  Version: 4 = .... 0100
                     Header Length: 20 bytes (5) = 0101 ....
Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
                                  Total Length: 59
                        Identification: 0x0001 (1)
                            Flags: 0x0 = .... .000
                Fragment Offset: 0 = 0000 0000 0000 0...
                                  Time to Live: 64
                                   Protocol: TCP (6)
         Header Checksum: 0x7cba [validation disabled]
              [Header checksum status: Unverified]
                       Source Address: 127.0.0.1
                  Destination Address: 127.0.0.1
                                  [Stream index: 0]
Transmission Control Protocol, Src Port: 18589, Dst Port: 12345, Seq: 0, Ack: 1, Len: 19
```

TCP משתמש במספרי רצף (Sequence Number) ומספרי אישור (Acknowledgment Number) לצורך מעקב אחר סדר המנות שהועברו. מספר הרצף מציין את מיקום הנתונים בזרם התקשורת, בעוד שמספר האישור מציין אילו נתונים התקבלו בהצלחה. מנגנון זה מאפשר זיהוי אובדן מנות ושמירה על סדר תקין של הנתונים.

אמינות התקשורת ב- TCP מושגת באמצעות שימוש באישורי קבלה, מיספור מנות ובמקרה הצורך שידור חוזר של מנות שלא אושרו. מנגנונים אלו מבטיחים שהנתונים יועברו בשלמותם, בסדר הנכון וללא אובדן.

שכבת הרשת:

בכל חבילת רשת מופיעות כתובת IP של המקור וכתובת IP של היעד, המשמשות לזיהוי נקודות הקצה ברמת הרשת. בלכידה ניתן לראות שימוש בכתובת 127.0.0.1 הן ככתובת מקור והן ככתובת יעד, דבר המעיד על תקשורת מקומית המתבצעת על גבי ממשק loopback .
IPv4 אחראי על מספור וניתוב חבילות ברשת ומייצג את הגרסה הנפוצה ביותר של פרוטוקול האינטרנט.

תפקידה של שכבת הרשת הוא ניתוב חבילות ממקור ליעד על בסיס כתובות IP . השכבה אחראית לקביעת נתיב ההעברה של החבילה ברשת, ללא הבטחה לאמינות או לסדר ההגעה. שכבה זו פועלת באופן בלתי תלוי בתוכן הנתונים, ומתמקדת בהעברת החבילה בין נקודות הקצה.

```
Frame 1293: Packet, 67 bytes on wire (536 bits), 67 bytes captured (536 bits) on interface \Device\NPF_Loopback, id 0
                                                                                                                Null/Loopback
                                                                                  Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
                                                                  Transmission Control Protocol, Src Port: 18589, Dst Port: 12345, Seq: 0, Ack: 1, Len: 23

02 00 00 00 45 00 00 3f  00 01 00 00 40 06 7c b6   ····E··? ····@·|·
7f 00 00 01 7f 00 00 01  48 9d 30 39 00 00 00 00   ........ H·09····
00 00 00 00 50 18 20 00  a4 54 00 00 47 45 54 20   ····P··· ·T··GET·
2f 73 74 79 6c 65 2e 63  73 73 20 48 54 54 50 2f   /style.c ss HTTP/
31 2e 31                                           1.1
```

סיכום:

ניתוח התעבורה שבוצע בעבודה זו אפשר הבנה מעמיקה של אופן העברת המידע ברשת, החל משכבת היישום ועד לשכבת הרשת. באמצעות בחינת החבילות ב־ Wireshark ניתן היה לראות כיצד נתוני היישום מועברים כ- payload בתוך מקטעי TCP ונעטפים בכותרות של שכבות התקשורת השונות כחלק מתהליך הקפסולציה.

הניתוח הדגים את חלוקת האחריות בין שכבות הרשת: שכבת היישום אחראית ליצירת המידע הלוגי, שכבת התעבורה (TCP) מספקת מנגנונים של אמינות, סדר ואישור קבלה, ואילו שכבת הרשת (IP) אחראית על ניתוב החבילות בין נקודות הקצה באמצעות כתובות IP.

בנוסף, נצפה כי גם כאשר התעבורה נוצרה באופן מדומה וללא חיבור TCP מלא בין יישומים, ניתן לזהות בבירור את מבנה החבילות ואת תפקידי השכבות השונות. עבודה זו חיזקה את ההבנה לגבי אופן פעולתו של פרוטוקול TCP ותפקידו המרכזי בהבטחת תקשורת אמינה מעל פרוטוקול

חלק שני- כתיבת יישום רשת וניתוח תעבודה של אותו יישום:

2.1. הסבר כללי על המערכת:

מטרת המערכת היא מימוש יישום צ'אט מבוסס Client-Server המאפשר למספר לקוחות להתחבר לשרת בו־זמנית, להזדהות בשם, ולשלוח הודעות ללקוח אחר באמצעות פורמט הודעה מוסכם.

מבנה כללי:

Server:

- מאזין לחיבורים נכנסים על כתובת IP ופורט קבועים.
- מקבל חיבור מלקוח, מבקש שם משתמש, ושומר את הלקוח במילון.
- מנתב הודעות מהשולח לנמען לפי השם שהוזן אליו רוצים לשלוח.

Clients:

- מתחבר לשרת באמצעות TCP socket.
- לאחר ההתחברות מקבל הודעה מהשרת על מנת לקבל את השם שלו.
- הלקוח יכול לשלוח הודעות.
- הלקוח מקבל הודעות נכנסות מהשרת ומציג אותן.

שפת תכנות וספריות:

הקוד נכתב בשפת **Python** ומשתמש בספריות:

- socket- לתקשורת רשת
- threading- לריבוי תהליכונים שנועד לאפשר לשרת לטפל במספר חיבורי לקוחות במקביל מבלי חסימה או עיכוב בין לקוחות.
- Tkinter- ספרייה מובנת בפייתון שמאפשרת ליצור ממשק גרפי GUI.

שימוש ב- TCP וב- Sockets:

המערכת משתמשת בפרוטוקול TCP כדי להבטיח תקשורת אמינה, מסודרת וללא איבוד הודעות. התקשורת מתבצעת באמצעות sockets:

- Server socket: bind, listen, accept
- Client socket: connect
- העברת מידע: sendall, recv

תמיכה בלפחות 5 לקוחות במקביל:

השרת תומך בריבוי לקוחות באמצעות **Thread לכל לקוח**:

- לכל חיבור חדש מהלקוח נפתח thread שמטפל בו.
- שימוש ב- Lock על מבנה הנתונים של הלקוחות כדי למנוע בעיות תחרות בזמן עדכון המילון.

מבנה הקוד:

פירוט קבצי הקוד:

- server.py - מימוש השרת: חיבור לקוחות, ניהול רשימת לקוחות, ניתוב הודעות, טיפול בניתוקים.
- client.py- מימוש הלקוח: התחברות לשרת, שליחת הודעות, קבלת הודעות והצגה.
- clientGui.py- מימוש הממשק המשתמש הגרפי- יצירת חלון הצאט, הצגת הודעות בצורה גרפית, קלט מהמשתמש, הצגת רשימת הלקוחות הפעילים וניהול האינטרקציה בין המשתמש ולבין מחלקת הלקוח.

תפקידי הקבצים:

server.py

- אתחול socket שרת והאזנה
- accept() בלולאה לחיבורים נכנסים
- יצירת thread לכל לקוח.
- אחסון לקוחות לפי שם במילון.
- ניתוב הודעות לפי יעד.

client.py

- התחברות לשרת
- שליחת שם המשתמש
- לולאה לקבלת הודעות (recv)
- קלט מהמשתמש ושליחה לשרת (sendall)

clientGui.py

- מייבא מחלקה מקובץ הלקוח (import)
- מריצים את קובץ הלקוח דרכו.

ריבוי תהליכונים:

- בשרת: לכל לקוח thread משלו.
- בלקוח: לקבלת ושליחת הודעות .
- שימוש ב- threading.Lock() כדי להגן על מבנה הלקוחות.

טיפול בשגיאות וניתוקים:

תיאור מה המערכת עושה במצבים כמו:

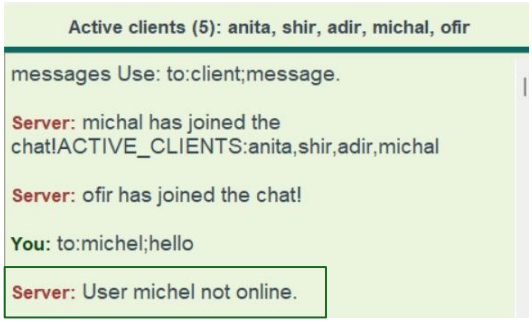
- כשלקוח מתנתק- בשרת מתקבלת הודעה שם המשתמש שהתנתק, מאיפה כתובת IP ומאיזה PORT ובצאטים ללקוחות ישלח לכל לקוח את שם המשתמש שעזב את הצאט ובנוסף יעודכן הactive clients לכמות הלקוחות הנוכחים ושמות אותם הלקוחות.
- שם משתמש קיים כבר- נתון לנו שהשם של כל לקוח יחודי
- שם משתמש לא הוזן- לא יתן להתחבר לצאט ויקבל הודעת שגיאה
- לקוח לא קיים - הודעת שגיאה לשולח שהשם משתמש שהזין לא קיים.

- הודעה בפורמט לא תקין -אם ההודעה לא בפורמט to:<name>;<message> תתקבל הודעה שהפורמט לא נכון ולא תשלח ההודעה למשתמש הרצוי.

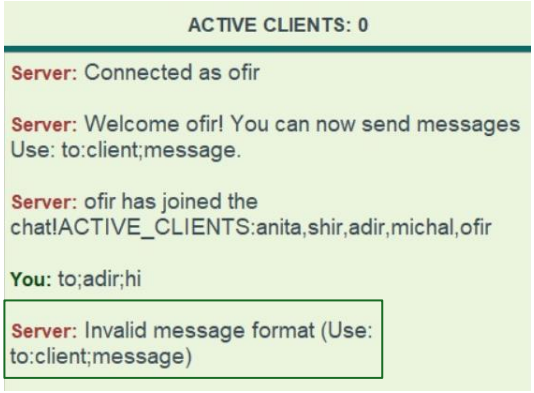
- בזכות try/except נוכל לטפל בשגיאות תקשורת וניתוקי לקוחות.
במקרה של שגיאה או ניתוק, השרת מסיר את הלקוח מהמילון, סוגר את חיבור ה- socket ומשחרר משאבים, ובכך מונע קריסת המערכת ופגיעה בלקוחות אחרים.

<u>דוגמאות לטיפול בשגיאות:</u>

שימוש בשם לקוח לא קיים:



שימוש בפורמט לא נכון:



אי הזנת שם לקוח:

2.2. הוראות התקנה והרצה:

דרישות מקדימות

- התקנת Visual Studio Code, ו- python.
- שימוש בספריות: socket, Tkinter, threading.
- רשת: שני מכשירים באותה רשת או מחשב יחיד (אם כך אז צריך להריץ על ידי פתיחת New Terminal, להזין python clientGui.py)

הרצת השרת

1. לפתוח את הקוד server.py ב- Visual Studio Code.
2. שינוי הHOST לכתובת הIP של המחשב עליו רץ השרת.
3. להריץ את הקוד.
4. השרת יתחיל להאזין וידפיס הודעה שהוא מקשיב.

הרצת לקוח

1. לפתוח את הקוד של clientGui.py במחשב אחר או לפתוח טרמינל נוסף באותו מחשב.
2. שינוי הHOST לכתובת הIP שנרשמה בקובץ השרת (server.py) בקובץ client.py.
3. לשים לב ששני הקבצים clientGui.py וclient.py נמצאים באותה תיקייה באותו מחשב.
4. להריץ את הקובץ clientGui.py (אם זה על מחשב אחד ורוצים להריץ כמה לקוחות צריך לפתוח New Terminal, להזין python clientGui.py).
5. להזין שם משתמש כאשר מתבקשים.
6. לשלוח הודעות לפי הפורמט:
   to:<name>;<message>

סדר פעולות ברור

1. מפעילים שרת.
2. מפעילים 2+ לקוחות מהקובץ של clientGui.py.
3. מזינים שמות.
4. שולחים הודעות בין לקוחות לפי הפורמט.

3.2. דוגמאות להתחברות לקוח:

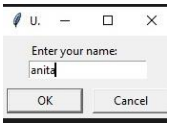
- הפעלת קובץ קוד השרת- אחרי הרצת קובץ השרת זו ההודעה שמתקבלת אצלו IP:PORT.

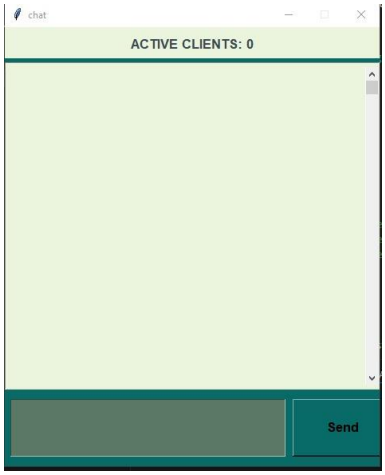
```
Server listening on 172.20.10.6:10000
```

צד השרת אחרי שכל הלקוחות מתנתקים:

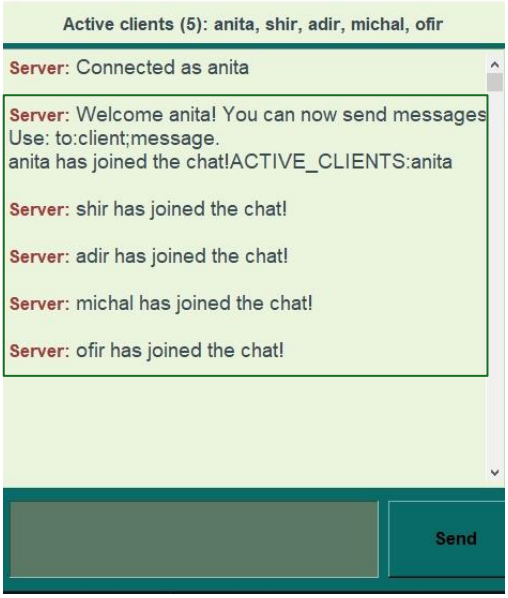
```
Server listening on 172.20.10.6:10000
Active clients: 1
anita connected from ('172.20.10.6', 51004)
Active clients: 2
shir connected from ('172.20.10.7', 57468)
Active clients: 3
adir connected from ('172.20.10.8', 60436)
Active clients: 4
michal connected from ('172.20.10.7', 57470)
Active clients: 5
ofir connected from ('172.20.10.8', 65322)
michal disconnected from ('172.20.10.7', 57470)
shir disconnected from ('172.20.10.7', 57468)
adir disconnected from ('172.20.10.8', 60436)
ofir disconnected from ('172.20.10.8', 65322)
anita disconnected from ('172.20.10.6', 51004)
```

- הפעלת קובץ קוד הלקוח- הכנסת שם המשתמש:

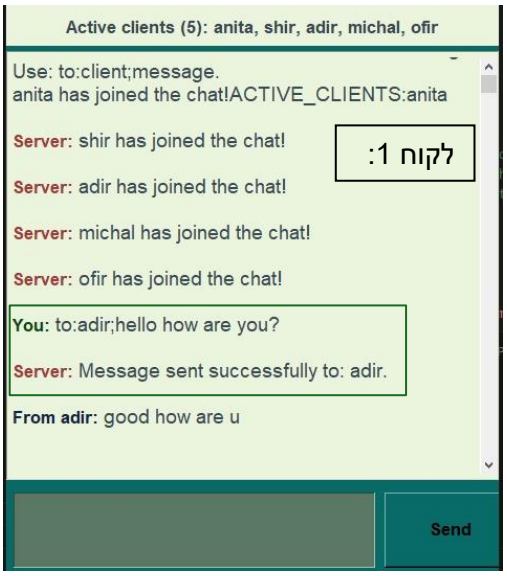


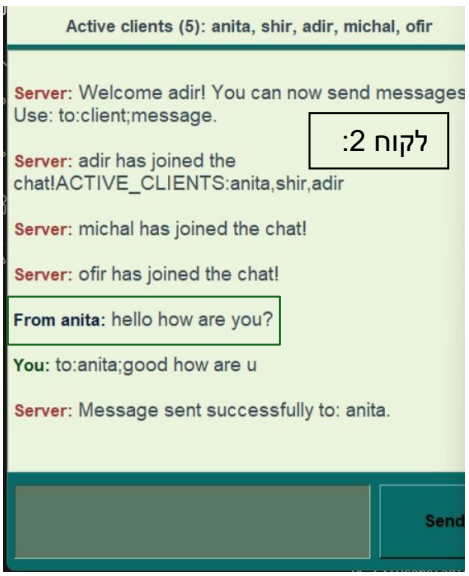


התחברות של אניטה כלקוח ראשון ושל עוד 4 לקוחות:

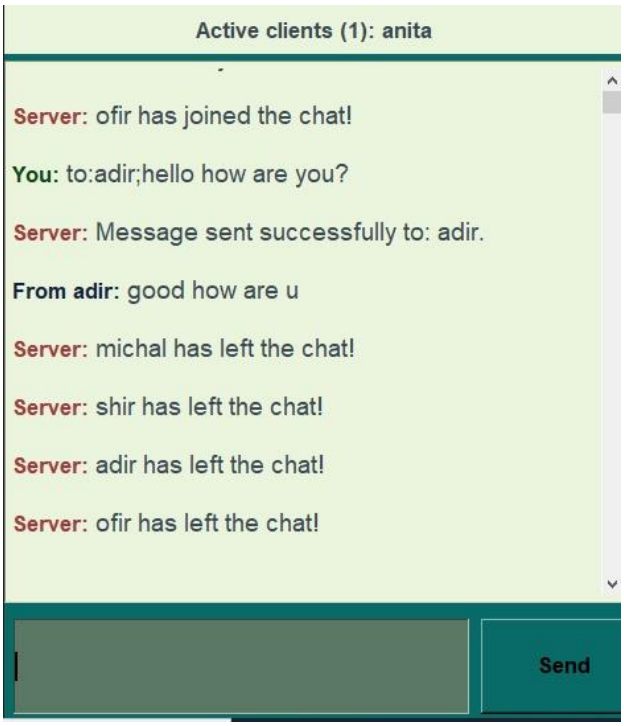


לקוח 1 (אניטה) שולח הודעה
לפי הפורמט ללקוח 2 (אדיר)
ומקבל אישור מהשרת
שההודעה נשלחה וגם רואים
איך לקוח 2 (אדיר) מקבל את
ההודעה.
ואותו הדבר קורה כשלקוח 2
(אדיר) עונה ללקוח 1 (אניטה):





הצ'אט של לקוח 1 (אניטה) אחרי התנתקות של כולם:



4.2. ניתוח תעבורה:

תיאור הלכידה- השתמשנו ב-Wireshark הפעלנו כממשק רשת את ה-Wi-Fi, עם סינון של tcp.port==10000 .

שלבי הלכידה:

1. התחלנו תפיסה בתוכנה.
2. התחברות של חמישה לקוחות.
3. שליחת הודעה בפורמט.
4. נראות של שגיאות (לקוח לא קיים, הודעה לא בפורמט).
5. התנתקויות של הלקוחות .
6. עצירת התפיסה.
7. שמירת הקובץ כ-pcap.

ניתוח לפי שכבות:

שכבת האפליקציה:

בשכבת האפליקציה ניתן לראות את פרוטוקול האפליקציה כפי שהוגדר. ההודעות מועברות לתוך שדה הנתונים של מקטעי TCP וכוללות מחרוזות טקסט קריאות. שכבת התעבורה אחראית רק להעברת המידע בצורה אמינה, בעוד שתוכן ההודעות והמשמעות שלהן שייכים לשכבת האפליקציה.

הפרוטוקול של הצ'אט שלנו:

- בקשת שם: "Enter your name:"
- פורמט הודעה: to:<name>;<message> משמש לניתוב הודעות בין לקוחות

השרת מבקש להזין שם

```
Frame 194: Packet, 71 bytes on wire (568 bits), 71 bytes captured (568 bits)
Ethernet II, Src: Intel_20:56:1b (34:e1:2d:20:56:1b), Dst: AzureWaveTec_03:2a:25 (50:fe:0c:03:2a:25)
Internet Protocol Version 4, Src: 172.20.10.6, Dst: 172.20.10.8
Transmission Control Protocol, Src Port: 10000, Dst Port: 58172, Seq: 1, Ack: 1, Len: 17
Data (17 bytes)

0000  50 fe 0c 03 2a 25 34 e1  2d 20 56 1b 08 00 45 00   P···*%4· - V···E·
0010  00 39 6e 4f 40 00 80 06  20 39 ac 14 0a 06 ac 14   ·9nO@··· 9·····
0020  0a 08 27 10 e3 3c 40 48  fa eb cf dc 0c fc 50 18   ··'··<@H ######P·
0030  02 01 fd 3d 00 00 45 6e  74 65 72 20 79 6f 75 72   ···=··En ter your
0040  20 6e 61 6d 65 3a 20                                name:
```

תשובת השם של הלקוח

```
Frame 195: Packet, 59 bytes on wire (472 bits), 59 bytes captured (472 bits)
Ethernet II, Src: AzureWaveTec_03:2a:25 (50:fe:0c:03:2a:25), Dst: Intel_20:56:1b (34:e1:2d:20:56:1b)
Internet Protocol Version 4, Src: 172.20.10.8, Dst: 172.20.10.6
Transmission Control Protocol, Src Port: 58172, Dst Port: 10000, Seq: 1, Ack: 18, Len: 5
Data (5 bytes)

0000  34 e1 2d 20 56 1b 50 fe  0c 03 2a 25 08 00 45 00   4·- V·P· ··*%··E·
0010  00 2d e7 83 40 00 80 06  a7 10 ac 14 0a 08 ac 14   ·-··@··· ########
0020  0a 06 e3 3c 27 10 cf dc  0c fc 40 48 fa fc 50 18   ···<'··· ··@H··P·
0030  00 ff f4 43 00 00 61 6e  69 74 61                  ···C··an ita
```

הודעת התחברות בהצלחה
עם שם הלקוח.

```
Frame 196: Packet, 119 bytes on wire (952 bits), 119 bytes captured (952 bits)
Ethernet II, Src: Intel_20:56:1b (34:e1:2d:20:56:1b), Dst: AzureWaveTec_03:2a:25 (50:fe:0c:03:2a:25)
Internet Protocol Version 4, Src: 172.20.10.6, Dst: 172.20.10.8
Transmission Control Protocol, Src Port: 10000, Dst Port: 58172, Seq: 18, Ack: 6, Len: 65
Data (65 bytes)
```

עדכון השרת ללקוחות
שלקוח נוסף התחבר

```
Frame 335: Packet, 104 bytes on wire (832 bits), 104 bytes captured (832 bits)
Ethernet II, Src: Intel_20:56:1b (34:e1:2d:20:56:1b), Dst: Intel_93:24:a2 (98:43:fa:93:24:a2)
Internet Protocol Version 4, Src: 172.20.10.6, Dst: 172.20.10.7
Transmission Control Protocol, Src Port: 10000, Dst Port: 49718, Seq: 82, Ack: 5, Len: 50
Data (50 bytes)
```

```
0000  98 43 fa 93 24 a2 34 e1  2d 20 56 1b 08 00 45 00   ·C··$·4· - V···E·
0010  00 5a a8 18 40 00 80 06  e6 4f ac 14 0a 06 ac 14   ·Z··@··· ·O......
0020  0a 07 27 10 c2 36 20 4a  c5 73 11 18 4f 19 50 18   ··'··6 J ·s··O·P·
0030  02 01 88 63 00 00 73 68  69 72 20 68 61 73 20 6a   ···c··sh ir has j
0040  6f 69 6e 65 64 20 74 68  65 20 63 68 61 74 21 41   oined th e chat!A
0050  43 54 49 56 45 5f 43 4c  49 45 4e 54 53 3a 61 6e   CTIVE_CL IENTS:an
0060  69 74 61 2c 73 68 69 72                            ita,shir
```

*כלל החיבורים של הלקוחות הנוספים נראים אותו הדבר

לקוח אחד שולח הודעה ללקוח אחר

```
Frame 2232: Packet, 80 bytes on wire (640 bits), 80 bytes captured (640 bits)
Ethernet II, Src: AzureWaveTec_03:2a:25 (50:fe:0c:03:2a:25), Dst: Intel_20:56:1b (34:e1:2d:20:56:1b)
Internet Protocol Version 4, Src: 172.20.10.8, Dst: 172.20.10.6
Transmission Control Protocol, Src Port: 58172, Dst Port: 10000, Seq: 6, Ack: 365, Len: 26
Data (26 bytes)
```

```
0000  34 e1 2d 20 56 1b 50 fe  0c 03 2a 25 08 00 45 00   4·- V·P· ··*%··E·
0010  00 42 e7 9c 40 00 80 06  a6 e2 ac 14 0a 08 ac 14   ·B··@··· ........
0020  0a 06 e3 3c 27 10 cf dc  0d 01 40 48 fc 57 50 18   ···<'··· ··@H·WP·
0030  00 fe c0 79 00 00 74 6f  3a 61 64 69 72 3b 68 65   ···y··to :adir;he
0040  6c 6c 6f 20 68 6f 77 20  61 72 65 20 79 6f 75 3f   llo how  are you?
```

הלקוח הראשון קיבל אישור שההודעה נשלחה

```
0000  50 fe 0c 03 2a 25 34 e1  2d 20 56 1b 08 00 45 00   P···*%4· - V···E·
0010  00 4b 6e 67 40 00 80 06  20 0f ac 14 0a 06 ac 14   ·Kng@··· ........
0020  0a 08 27 10 e3 3c 40 48  fc 57 cf dc 0d 1b 50 18   ··'··<@H ·W····P·
0030  02 01 54 3a 00 00 4d 65  73 73 61 67 65 20 73 65   ··T:··Me ssage se
0040  6e 74 20 73 75 63 63 65  73 73 66 75 6c 6c 79 20   nt succe ssfully
0050  74 6f 3a 20 61 64 69 72  2e                        to: adir .
```

שליחת הודעה ללקוח שלא מחובר

```
Frame 3261: Packet, 69 bytes on wire (552 bits), 69 bytes captured (552 bits)
Ethernet II, Src: Intel_93:24:a2 (98:43:fa:93:24:a2), Dst: Intel_20:56:1b (34:e1:2d:20:56:1b)
Internet Protocol Version 4, Src: 172.20.10.7, Dst: 172.20.10.6
Transmission Control Protocol, Src Port: 49727, Dst Port: 10000, Seq: 7, Ack: 215, Len: 15
Data (15 bytes)
```

```
0000  34 e1 2d 20 56 1b 98 43  fa 93 24 a2 08 00 45 00   4·- V··C ··$···E·
0010  00 37 17 aa 40 00 80 06  76 e1 ac 14 0a 07 ac 14   ·7··@··· v·......
0020  0a 06 c2 3f 27 10 f0 09  98 30 f7 63 ae 02 50 18   ···?'··· ·0·c··P·
0030  00 ff fa e4 00 00 74 6f  3a 6d 69 63 68 65 6c 3b   ......to :michel;
0040  68 65 6c 6c 6f                                     hello
```

השרת שולח שהלקוח
הזה לא קיים

```
Frame 3262: Packet, 77 bytes on wire (616 bits), 77 bytes captured (616 bits)
Ethernet II, Src: Intel_20:56:1b (34:e1:2d:20:56:1b), Dst: Intel_93:24:a2 (98:43:fa:93:24:a2)
Internet Protocol Version 4, Src: 172.20.10.6, Dst: 172.20.10.7
Transmission Control Protocol, Src Port: 10000, Dst Port: 49727, Seq: 215, Ack: 22, Len: 23
Data (23 bytes)
```

```
0000  98 43 fa 93 24 a2 34 e1  2d 20 56 1b 08 00 45 00   ·C··$·4· - V···E·
0010  00 3f a8 25 40 00 80 06  e6 5d ac 14 0a 06 ac 14   ·?·%@··· ·]******
0020  0a 07 27 10 c2 3f f7 63  ae 02 f0 09 98 3f 50 18   ··'··?·c ·····?P·
0030  02 01 b6 75 00 00 55 73  65 72 20 6d 69 63 68 65   ···u··Us er miche
0040  6c 20 6e 6f 74 20 6f 6e  6c 69 6e 65 2e            l not on line.
```

שליחת הודעה בפורמט לא נכון

```
Frame 3896: Packet, 64 bytes on wire (512 bits), 64 bytes captured (512 bits)
Ethernet II, Src: AzureWaveTec_03:2a:25 (50:fe:0c:03:2a:25), Dst: Intel_20:56:1b (34:e1:2d:20:56:1b)
Internet Protocol Version 4, Src: 172.20.10.8, Dst: 172.20.10.6
Transmission Control Protocol, Src Port: 56176, Dst Port: 10000, Seq: 5, Ack: 149, Len: 10
Data (10 bytes)
```

```
0000  34 e1 2d 20 56 1b 50 fe  0c 03 2a 25 08 00 45 00   4·- V·P· ··*%··E·
0010  00 32 e7 a2 40 00 80 06  a6 ec ac 14 0a 08 ac 14   ·2··@··· ********
0020  0a 06 db 70 27 10 24 af  74 5c e0 b3 56 cf 50 18   ···p'·$· t\··V·P·
0030  00 ff 80 9e 00 00 74 6f  3b 61 64 69 72 3b 68 69   ******to ;adir;hi
```

תשובת השרת שהפורמט לא נכון

```
Frame 3897: Packet, 101 bytes on wire (808 bits), 101 bytes captured (808 bits)
Ethernet II, Src: Intel_20:56:1b (34:e1:2d:20:56:1b), Dst: AzureWaveTec_03:2a:25 (50:fe:0c:03:2a:25)
Internet Protocol Version 4, Src: 172.20.10.6, Dst: 172.20.10.8
Transmission Control Protocol, Src Port: 10000, Dst Port: 56176, Seq: 149, Ack: 15, Len: 47
Data (47 bytes)
```

```
0000  50 fe 0c 03 2a 25 34 e1  2d 20 56 1b 08 00 45 00   P···*%4· - V···E·
0010  00 57 6e 6a 40 00 80 06  20 00 ac 14 0a 06 ac 14   ·Wnj@··· ********
0020  0a 08 27 10 db 70 e0 b3  56 cf 24 af 74 66 50 18   ··'··p·· V·$·tfP·
0030  02 01 b1 ce 00 00 49 6e  76 61 6c 69 64 20 6d 65   ******In valid me
0040  73 73 61 67 65 20 66 6f  72 6d 61 74 20 28 55 73   ssage fo rmat (Us
0050  65 3a 20 74 6f 3a 63 6c  69 65 6e 74 3b 6d 65 73   e: to:cl ient;mes
0060  73 61 67 65 29                                      sage)
```

השרת מעדכן את שאר
הלקוחות בלקוחות שנשארו

```
Frame 4074: Packet, 89 bytes on wire (712 bits), 89 bytes captured (712 bits)
Ethernet II, Src: Intel_20:56:1b (34:e1:2d:20:56:1b), Dst: Intel_93:24:a2 (98:43:fa:93:24:a2)
Internet Protocol Version 4, Src: 172.20.10.6, Dst: 172.20.10.7
Transmission Control Protocol, Src Port: 10000, Dst Port: 49718, Seq: 343, Ack: 5, Len: 35
Data (35 bytes)
```

```
0000  98 43 fa 93 24 a2 34 e1  2d 20 56 1b 08 00 45 00   ·C··$·4· - V···E·
0010  00 4b a8 29 40 00 80 06  e6 4d ac 14 0a 06 ac 14   ·K·)@··· ·M******
0020  0a 07 27 10 c2 36 20 4a  c6 78 11 18 4f 19 50 18   ··'··6 J ·x··O·P·
0030  02 01 ec 52 00 00 41 43  54 49 56 45 5f 43 4c 49   ···R··AC TIVE_CLI
0040  45 4e 54 53 3a 61 6e 69  74 61 2c 73 68 69 72 2c   ENTS:ani ta,shir,
0050  61 64 69 72 2c 6f 66 69  72                        adir,ofi r
```

שכבת התעבורה:

שלושת השורות הראשונות (191-193) מתארות את תהליך ה־3 -way handshake שבו הלקוח מבקש לפתוח חיבור לשרת, השרת מאשר את קבלת הבקשה ומוכן לחיבור ואז שורה שהחיבור נוצר בהצלחה.

SYN- הלקוח שולח לשרת בקשת פתיחת חיבור ומציע מספר רצף התחלתי.(Sequence Number)

SYN-ACK- השרת מאשר את הבקשה (ACK=1) ושולח גם SYN משלו עם מספר רצף משלו.

ACK- הלקוח מאשר את ה SYN-של השרת, ובכך החיבור נוצר

לאחר שלב זה החיבור TCP נחשב פעיל, וניתן להעביר נתוני אפליקציה.

tcp.port == 10000

.No	Time	Source	Destination	Protocol	ength	Info
191	26.609404	172.20.10.8	172.20.10.6	TCP	66	Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM [SYN] 10000 → 58172
192	26.609508	172.20.10.6	172.20.10.8	TCP	66	Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM [SYN, ACK] 58172 → 10000
193	26.618602	172.20.10.8	172.20.10.6	TCP	54	Seq=1 Ack=1 Win=65280 Len=0 [ACK] 10000 → 58172

משתמש ראשון התנתק מהצ'אט

4068	561.472036	172.20.10.7	172.20.10.6	TCP	Seq=45 Ack=301 Win=0 Len=0 [RST, ACK] 10000 → 49727 54
4069	561.472602	172.20.10.6	172.20.10.8	TCP	Seq=425 Ack=32 Win=131328 Len=25 [PSH, ACK] 58172 → 10000 79
4070	561.472686	172.20.10.6	172.20.10.7	TCP	Seq=318 Ack=5 Win=131328 Len=25 [PSH, ACK] 49718 → 10000 79
4071	561.472726	172.20.10.6	172.20.10.8	TCP	Seq=334 Ack=28 Win=131328 Len=25 [PSH, ACK] 58176 → 10000 79
4072	561.472761	172.20.10.6	172.20.10.8	TCP	Seq=196 Ack=15 Win=131328 Len=25 [PSH, ACK] 56176 → 10000 79

ההודעה שהשרת שולח לכל הלקוחות כשהוא התנתק

Frame 4069: Packet, 79 bytes on wire (632 bits), 79 bytes captured (632 bits)
Ethernet II, Src: Intel_20:56:1b (34:e1:2d:20:56:1b), Dst: AzureWaveTec_03:2a:25 (50:fe:0c:03:2a:25)
Internet Protocol Version 4, Src: 172.20.10.6, Dst: 172.20.10.8
Transmission Control Protocol, Src Port: 10000, Dst Port: 58172, Seq: 425, Ack: 32, Len: 25
Data (25 bytes)

Src Port - פורט מקור

Dst Port- פורט יעד

```
0000  50 fe 0c 03 2a 25 34 e1  2d 20 56 1b 08 00 45 00   P···*%4· - V···E·
0010  00 41 6e 6b 40 00 80 06  20 15 ac 14 0a 06 ac 14   ·Ank@··· ·******
0020  0a 08 27 10 e3 3c 40 48  fc 93 cf dc 0d 1b 50 18   ··'··<@H ******P·
0030  02 01 af e8 00 00 6d 69  63 68 61 6c 20 68 61 73   ******mi chal has
0040  20 6c 65 66 74 20 74 68  65 20 63 68 61 74 21       left th e chat!
```

2232- לקוח אחד שולח לשרת בקשה לשלוח הודעה ללקוח השני

2233- השרת שולח ללקוח השני את ההודעה שהלקוח הראשון ביקש

1691	177.139757	172.20.10.8	172.20.10.6	TCP	Seq=6 Ack=365 Win=65024 Len=0 [ACK] 10000 → 58172 54
1692	177.148641	172.20.10.8	172.20.10.6	TCP	Seq=5 Ack=149 Win=65280 Len=0 [ACK] 10000 → 56176 54
2232	256.874938	172.20.10.8	172.20.10.6	TCP	Seq=6 Ack=365 Win=65024 Len=26 [PSH, ACK] 10000 → 58172 80
2233	256.875104	172.20.10.6	172.20.10.8	TCP	Seq=268 Ack=5 Win=131328 Len=30 [PSH, ACK] 58176 → 10000 84
2234	256.875164	172.20.10.6	172.20.10.8	TCP	Seq=365 Ack=32 Win=131328 Len=35 [PSH, ACK] 58172 → 10000 89
2235	256.933727	172.20.10.8	172.20.10.6	TCP	Seq=5 Ack=298 Win=65024 Len=0 [ACK] 10000 → 58176 54
2236	256.933727	172.20.10.8	172.20.10.6	TCP	Seq=32 Ack=400 Win=65024 Len=0 [ACK] 10000 → 58172 54

חיבור של לקוח ראשון-

194	26.619510	172.20.10.6	172.20.10.8	TCP	Seq=1 Ack=1 Win=131328 Len=17 [PSH, ACK] 58172 → 10000 71
195	26.627739	172.20.10.8	172.20.10.6	TCP	Seq=1 Ack=18 Win=65280 Len=5 [PSH, ACK] 10000 → 58172 59
196	26.627906	172.20.10.6	172.20.10.8	TCP	Seq=18 Ack=6 Win=131328 Len=65 [PSH, ACK] 58172 → 10000 119
198	26.681817	172.20.10.8	172.20.10.6	TCP	Seq=6 Ack=83 Win=65280 Len=0 [ACK] 10000 → 58172 54
199	26.681857	172.20.10.6	172.20.10.8	TCP	Seq=83 Ack=6 Win=131328 Len=46 [PSH, ACK] 58172 → 10000 100
200	26.743124	172.20.10.8	172.20.10.6	TCP	Seq=6 Ack=129 Win=65280 Len=0 [ACK] 10000 → 58172 54

שכבת הרשת:

הצילום מציג את שכבת הרשת (IPv4). ניתן לראות את כתובת ה- IP של המקור והיעד, את שדה ה- TTL ואת שדה ה- Protocol שמציין כי המטען מועבר לפרוטוקול TCP בשכבת התעבורה.
שכבה זו אחראית על ניתוב החבילה בין מארחים ברשת.

```
Internet Protocol Version 4, Src: 172.20.10.6, Dst: 172.20.10.7
                                        Version: 4 = .... 0100
                         Header Length: 20 bytes (5) = 0101 ....
Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
                                   Total Length: 104
                             Identification: 0xa7e4 (42980)
                       Flags: 0x2, Don't fragment = .... .010
                   Fragment Offset: 0 = 0000 0000 0000 0...
                                   Time to Live: 128
                                   Protocol: TCP (6)
                 Header Checksum: 0xe675 [validation disabled]
                          [Header checksum status: Unverified]
                                Source Address: 172.20.10.6
                           Destination Address: 172.20.10.7
                                   [Stream index: 1]
```

שליחת הודעה בין 2 לקוחות שיושבים על אותו מחשב ולכן הכתובת IP זהה.

130- לקוח שולח ללקוח אחר הודעה

131- הלקוח השני מקבל את ההודעה

132- אישור שההודעה הועברה

133- הלקוח ששלח את ההודעה מקבל אישר מהשרת שההודעה נשלחה

Seq=22 Ack=238 Win=65280 Len=23 [PSH, ACK]	10000 → 57470	77	TCP	172.20.10.6	172.20.10.7	171.709512	130
Seq=238 Ack=45 Win=131328 Len=26 [PSH, ACK]	57470 → 10000	80	TCP	172.20.10.7	172.20.10.6	171.709689	131
Seq=45 Ack=264 Win=65024 Len=0 [ACK]	10000 → 57470	54	TCP	172.20.10.6	172.20.10.7	171.764508	132
Seq=264 Ack=45 Win=131328 Len=37 [PSH, ACK]	57470 → 10000	91	TCP	172.20.10.7	172.20.10.6	171.764544	133

קובץ רלוונטי: wireshark_part2.

1.3. שימוש בבינה מלאכותית:

- לקבל עזרה בבניית הקובץ .CSV
- עזרה בקובץ ה clientGui.py עם העיצוב של הצאט עצמו, אמר לנו איך להראות מי הלקוחות המחוברים.
- הAI הציע לנו שיפור לקוד שאם הלקוח לא הזין שם משתמש זה יראה לו שגיאה.
- עזר לסדר את הדוח שיראה ברמה גבוהה.
- נעזרו בו על מנת לעשות את הקובץ Read me.

2.3. דוגמאות פרומפטים:

- תוכל ליצור לי קובץ CSV לפי שמות העמודות הבאות?
- תוכל ליצור פונקציה שמראה איזה לקוחות מחוברים בזמן אמת?
- איך להפוך את הקוד ליותר מאורגן?
- האם יש שיפור או הערה חשובה שצריך להוסיף לקוד?
- איך נראה קובץ Read me? ומה צריך להיות בו?