

---

פרויקט גמר

# ניתוח תעבורה בפרוטוקול

## TCP/IP

מוגש על ידי: לירן דגן

תעודת זהות: 215609397

שם מרצה: ד"ר קוזובוב אנדריי

קבוצת הרצאה: 61305-3

---

---

# תוכן עניינים

---

## חלק ראשון..... 3

- 1. מבוא ..... 3
- 2. יצירת קובץ ה-CSV ..... 3
- 3. תיאור והסבר אריזת הפאקטות ..... 6
- 4. ניתוח התעבורה באמצעות WIRESHARK ..... 10

## חלק שני ..... 15

- 5. מבוא ..... 15
- 6. ארכיטקטורת הפרויקט ..... 16
  - 6.1 אפליקציית השרת ..... 17
  - 6.2 אפליקציית הלקוח ..... 21
- 7. הוראות התקנה והרצה ..... 24
- 8. דוגמאות קלט ופלט ..... 30

# חלק ראשון

## אריזת נתונים ולכידת מנות ב-Wireshark

### 1. מבוא

בחלק הראשון של הפרויקט נדרשים אנו להכין קובץ CSV ובו עמודות שונות לניתוח בסיסי של מידע ברמת היישום. מטרת העל של הקובץ היא להציג הודעות שמקבל המחשב מאפליקציות ואתרים שונים בהתאם לפרוטוקול יישומי נבחר. (כדוגמת HTTP, DNS, SMTP) הודעות אלו הן כאמור מתקבלות על גבי פרוטוקול בשכבת האפליקציה/יישום, ולשם הפרויקט בחרתי לייבא הודעות שהועברו בפרוטוקול DNS (Domain Name System).

### 2. יצירת קובץ ה-CSV

לקובץ ה-CSV שמצורף לדו"ח הוספתי שדות שמלמדים על הפאקטות, בין היתר סוג החיבור (UDP), Request/Reply, גודל הפאקטה וכו'.

כמובן שהקובץ מכיל גם את השדות להם נדרשנו בתיאור הפרויקט והם מכילים:

- msg\_id - מספר סידורי המייחד כל הודעה
- app\_protocol - פרוטוקול ברמת האפליקציה אשר בחרנו הלא הוא DNS
- src\_port - מספר הפורט ממנו יצאה הפאקטה בצד השולח
- dst\_port - מספר הפורט אליו הגיעה הפאקטה בצד המקבל
- time\_stamp - הרגע שבו הגיעה הפאקטה
- message - הודעה שנרצה להפיק מהפאקטה

הקובץ נוצר בשלמותו באמצעות לכידת תעבורה בתוכנת Wireshark והנתונים שבו הם למעשה פאקטות שהוקלטו מאתרים ואפליקציות שהיו פועלים על המחשב שלי בעת ההקלטה. (אפשר למשל למצוא בקובץ התעבורה שהתקבלה הודעות משרתי ה-DNS של Google ושל Discord, שרצו על המחשב שלי בעת ההקלטה).

כדי לקבל את התעבורה שרציתי עבור פרוטוקול ה-DNS השתמשתי בפילטר 'dns' כדי לסווג את הפאקטות הרלוונטיות. עבור כל שדה בפאקטה שרציתי להוסיף כעמודה ביצעתי לחיצה ימנית על העכבר ולחצתי על "apply as column". כך קיבלתי את כל השדות שרציתי לייצא עבור כל פאקטה. לאחר שאספתי מספיק תעבורה עצרתי את ההקלטה, לחצתי על תפריט File, לאחר מכן על Export packet dissections ולאחר מכן על CSV as .as. כך ייצאתי את הטבלה שנוצרה ב-Wireshark אל תוכנת Excel ומשם שמרתי את הטבלה בתור קובץ CSV שנשמר מקומית על המחשב.

עבור השדה message – הרכבתי אותו על ידי שרשור שדות:

( domain name || is Response || Transaction ID)

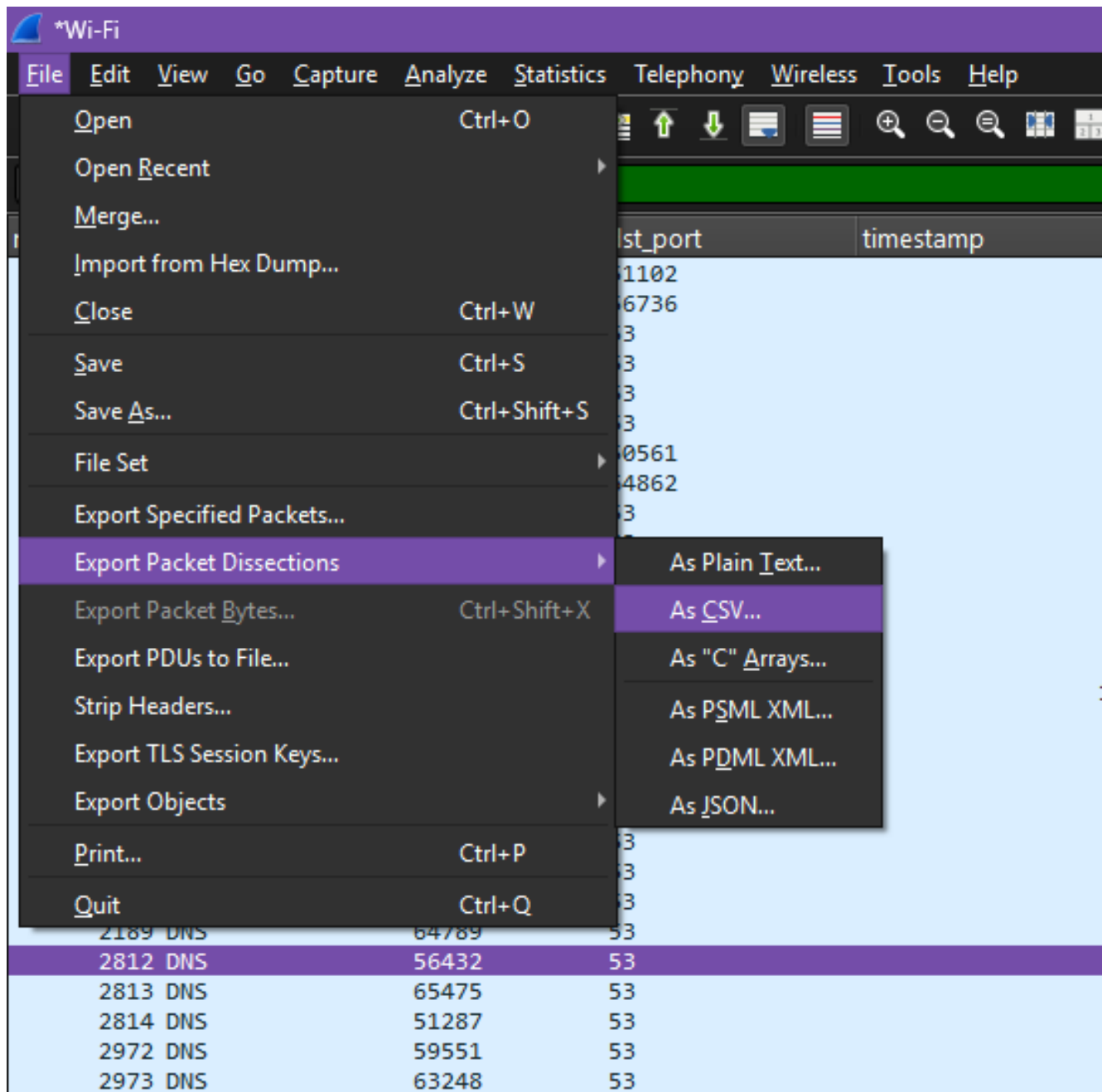
שאלו כולם שדות שמאפיינים את פרוטוקול DNS. Domain name משמעו שם הכתובת של האפליקציה, is Response משמעו האם הפאקטה היא בקשה לשרת (אפס) או תגובה של השרת (אחת).

Transaction ID משמעו מספר סידורי של פרוטוקול ה-DNS עבור השאילתה.

יצירת השדה message

Fields: dns.qry.name    dns.flags.response    dns.id		
Destination Address	timestamp	message
1.1.1.1	0.000108	dns.google,0,0x03d8
1.1.1.1	0.000140	dns.google,0,0x0957
1.1.1.1	0.000336	dns.google,0,0x0b42
1.1.1.1	0.000071	dns.google,0,0x162f
1.1.1.1	16.693004	dns.google,0,0x18d5
1.1.1.1	0.000361	dns.google,0,0x2100
1.1.1.1	0.895141	dns.google,0,0x3072
1.1.1.1	6.888908	dns.google,0,0x368f
1.1.1.1	29.761779	dns.google,0,0x3abc
1.1.1.1	1.014845	dns.google,0,0x41f8
1.1.1.1	2.948365	dns.google,0,0x4261
1.1.1.1	0.000235	dns.google,0,0x5954
1.1.1.1	0.000210	dns.google,0,0x60af
1.1.1.1	0.000120	dns.google,0,0x6d94
1.1.1.1	0.000145	dns.google,0,0x6f9b
1.1.1.1	0.000129	dns.google,0,0x86f9
1.1.1.1	0.000216	dns.google,0,0x940d
1.1.1.1	0.000334	dns.google,0,0x9473
1.1.1.1	0.000388	dns.google,0,0x9cb5
1.1.1.1	0.000256	dns.google,0,0xa3fa
1.1.1.1	1.012652	dns.google,0,0xa7a7
1.1.1.1	0.000329	dns.google,0,0xe57d
1.1.1.1	0.882902	dns.google,0,0xf405
2.168.1.18	0.008372	dns.google,1,0x0957
2.168.1.18	0.000000	dns.google,1,0x162f
2.168.1.18	0.006692	dns.google,1,0x2100
2.168.1.18	0.001023	dns.google,1,0x368f
2.168.1.18	0.029216	dns.google,1,0x3abc
2.168.1.18	0.001168	dns.google,1,0x41f8
2.168.1.18	0.026362	dns.google,1,0x60af
2.168.1.18	0.066347	dns.google,1,0x6d94
2.168.1.18	0.006016	dns.google,1,0x6f9b
2.168.1.18	0.011840	dns.google,1,0x940d
2.168.1.18	0.001820	dns.google,1,0xe57d
2.168.1.18	0.020594	dns.google,1,0xf405
2.168.1.18	14.580840	dns.msftncsi.com,0,0x3fc4
2.168.1.18	0.100803	dns.msftncsi.com,0,0x3fc4
1.1.1.1	1.011470	dns.msftncsi.com,0,0x3fc4
1.1.1.1	8.962245	

## ייצוא הפאקטות אל קובץ CSV



### 3. תיאור והסבר אריזת הפאקטות

השלב השני בחלק זה הוא טעינת קובץ ה-CSV אל מחברת הג'ופיטר המצורפת שבה קוד פייתון לייצור תעבורה מקומית של הודעות שנלקחו מה-CSV.

רעיונית אנחנו משתמשים בבימוס (Encapsulation) במודל OSI כדי לארוז את הפאקטות ולהעבירן. האריזה מרכיבה את הפאקטה מהשכבה העליונה (האפליקציה) עד לשכבות התחתונות. נתאר באופן כללי את תהליך האריזה ולאחר מכן נפרט.

תחילה ההודעה נוצרת בשכבת האפליקציה (שולחים הודעה, טקסט שנבחר למסור) ולאחר מכן מועברת אל שכבת התעבורה. במחברת, האפליקציה שולחת את הודעות ה-CSV שלנו (או הודעת דמה). כלומר המחברת שולחת כל הודעה משורות ה-CSV אל שכבת התעבורה.

שכבת התעבורה עוטפת את ההודעה בתוך SEGMENT לפענוח על ידי שכבת התעבורה בצד המקבל. פרוקטול TCP משייך לפאקטה כותרת TCP Header שבה הוא מכיל:

source port, destination port, flags ושלוש ערכים נוספים. בין הערכים נוסף שדה Checksum שמשמש את הצד המקבל לפענוח שגיאות. שכבת התעבורה מעבירה את הפאקטות לשכבת הרשת. עד כה הודעת ה-CSV שלנו נעטפה בסגמנט של שכבת התעבורה.

שכבת הרשת עוטפת את הסגמנט בתוך IP Datagram. כלומר הוא מוסיף כותרת IP Header שמכילה: source IP, destination IP, version, TTL וערכים נוספים. שכבת הרשת מעבירה את הפאקטה לשכבת הקו. עד כה הודעת ה-CSV שלנו נעטפה בסגמנט, שנעטף בכותרת IP.

רעיונית שכבת הקו מוסיפה מסגרת לפאקטה והיא השכבה האחרונה שעוטפת אותה לפני שהיא משודרת הלאה. היא מוסיפה כתובות MAC (מקור ויעד), פרוטוקול רשת (ipv4, ipv6) ומעבירה את הפאקטה לשכבה הפיזית, כלומר משדרת אותה מהמכשיר אל החוץ. בפועל, במקרה שלנו התהליכים שתעשה המחברת הם לוקאליים ולכן שכבת הקו לא תוסיף כתובות MAC או מידע חדש אבל כן יש לה תפקיד בכך שהיא תשדר את הפאקטות אל לולאת ה-loopback, כלומר מקומית בתוך המחשב.

אלו הצעדים הרעיוניים, מכאן והילך נסביר במפורט איך נוצרים כותרות ה-TCP\_Header ו-IP\_Header באופן רחב על השדות של כל כותרת וכדי להבין לעומק את המתרחש בתהליכים.

#### TCP Header Format

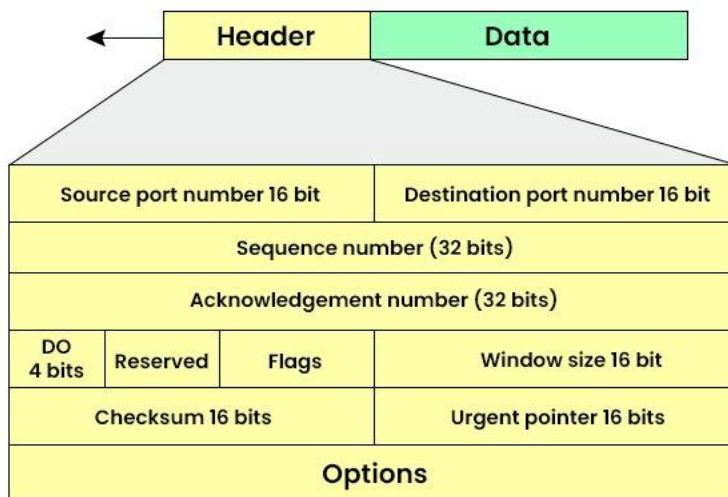


Image from [pynetlabs.com](http://pynetlabs.com)

למעשה עד כאן פירטנו את הרעיון שבאריזת המנות ומעכשיו נפרט מה בפועל כל שכבה עושה בהתאם לקוד שכתוב במחברת. כלומר ננתח את הקוד של הפונקציות build\_tcp\_header ו-build\_ip\_header. מדובר במעין העמקה רחבה הרבה יותר... נתחיל משכבת האפליקציה ונרד לשכבת הרשת.

בשכבת האפליקציה תחילה אנחנו טוענים את ה-CSV המוכן שלנו למחברת עם שימוש בספריית pandas.

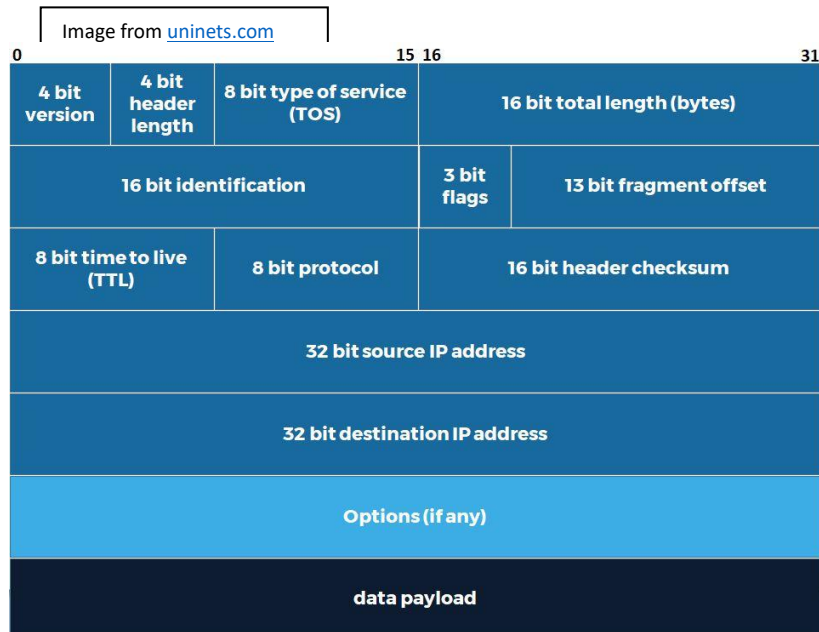
במחברת מוגדרות פונקציות לבנייה "ידנית" של כותרת ה-TCP header של שכבת

התעבורה ולבניית כותרת ה-IP header של שכבת הרשת, לשימוש במידת הצורך בהתאם למערכת ההפעלה. לאחר מכן המחברת טוענת את ההודעה עבור כל שורה ב-CSV ושולחת אותה לוקאלית עם פונקציית send.

כדי לבנות את כותרת ה-TCP אנחנו קודם כל בונים כותרת tcp\_header ראשונית שמכילה רק חלק מהשדות. הסיבה היא ששדה ה-checksum משמש לבקרת המידע נשען על שאר השדות, וכדי לחשב אותו נצטרך להרכיב את שאר השדות לפניו.

אז, בהינתן:

- מספרי הפורט של המקור ושל היעד
  - מספר סידורי (seq number) שסופר כל בייט שעובר (ומאותחל רנדומלית אם לא אותחל עדיין)
  - מספר סידורי שמאשר את קבלת הנתונים עד הבייט הנוכחי (ack number)
  - סוג הפאקטה (flags), כלומר רצף ביטים התואמים לדגלי Syn, Ack, Fin ודגלים נוספים המתארים את הפעולות שהפאקטה אחראית להשלים.
  - גודל החלון (Window) שמשמש לדעת כמה בייטים ניתן לשלוח ברצף פאקטות אחד ברגע זה למקבל.
  - Data Offset, Reserved, שדות שמשמשים להגדרת אורך הפאקטה ולשמירת 4 ביטים נוספים.
  - urgentPointer = 0, שדה שמשמש לציון בייטים מסוימים שדחופים לעיבוד והוא מצביע על סוף רצף הבייטים האלו. לנו אין נתונים שדחופים לעיבוד ולכן השדה יישאר אפס.
- לאחר שאספנו את כל הנתונים אנחנו אורזים אותם לתוך כותרת tcp\_header ראשונית.
- לאחר מכן אנחנו בונים pseudo\_header עם חלק מהשדות שנכנסים לכותרת ה-IP. (כמו כתובות ה-IP). רק לאחר מכן אנחנו מחשבים את שדה ה-checksum.
- Checksum, שדה זה משמש את המקבל כדי לאמת את המידע שבפאקטה. הוא עושה זאת על ידי חיבור כל השדות בפאקטה ביחד עם שדות של pseudo\_header. החיבור הוא של כל 16 ביטים. לאחר קבלת וצאת החיבור הופכים את הבייטים וזהו ה-checksum. אם המקבל מבצע את אותו תהליך ומקבל תוצאה שונה ממה ששמור בשדה זה הוא יגלה שהמידע שבפאקטה השתבש במהלך ההעברה, וידע לזרוק את הפאקטה או לבקש שליחה מחודשת.
- רק לאחר שחישבנו את שדה ה-checksum אנחנו אורזים מחדש את ה-tcp\_header שלנו, הפעם עם שדה ה-checksum ובכך השלמנו את אריזת ה-segment של שכבת התעבורה.
- כדי להסביר את שכבת הרשת ובניית כותרת ה-IPv4, נשים לב למבנה של ה-Header:



בניית כותרת ה-IP של שכבת הרשת. גם כאן נבנה header התחלתי כדי לבצע את חישוב ה-checksum מאוחר יותר.

בהינתן:

- כתובות ה-IP של המקור ושל היעד
- גודל הדאטה שהפאקטה נושאת (כלומר – ה-payload שקיבלנו בשכבת האפליקציה וגודל הסגמנט שנוסף לה בשכבת התעבורה).
- פרוטוקול התעבורה (TCP)

אנחנו בונים את ה-header כך:

- אצלנו הגרסה היא 4 (ipv4) ולכן version=4.
- header\_length הוא גודל ה-IP\_Header שאנחנו בונים עכשיו. הגודל נמדד במילים ולא בבייטים. לכן אם הגודל הוא למשל 5 (מילים), הגודל בפועל הוא  $5 \cdot 4 = 20$  בייטים.
- TOS הוא שדה שנועד לציין לצד המקבל כיצד לטפל בפאקטה עם רצפי ביטים שמיועדים לכך. (בדרך כלל כדי לציין דרישות דילוי, תפוקה, אמינות מידע וכו'). לנו אין דרישות מיוחדות עבור הפאקטות ולכן השדה יישאר אפס.
- total\_length הוא הגודל של הפאקטה כולה, כלומר הגודל header\_length בבייטים פלוס הגודל של ההודעה והסגמנט שקיבלנו משכבות האפליקציה והתעבורה.
- identification. רעיונית השדה הזה משמש את הדאטה להתפצל למקטעים. כלומר, במידה והפאקטה גדולה מדי עבור הקישור שצריך לשדר אותה החוצה, שכבת הרשת תפצל את הדאטה למקטעים נפרדים, עם IP\_headers שונים אך עם שדות identification זהים. המקטעים יישלחו בנפרד לקישור, וביעד (שכבת הרשת של הצד המקבל) יתאספו חזרה אל הפאקטה המקורית שהם מרכיבים ביחד. הם ידעו להתאסף חזרה אל הפאקטה המקורית לפי השדה identification המשותף שהם מחזיקים.
- השדות flags, fragment offset שב-header הם ביטים שמציינים האם יש לפצל את הדאטה למקטעים ואם כן, מה המיקום של כל מקטע ביחס לפאקטה המקורית.



- השדה ttl (Time To Live) מגדיר את מספר רכיבי הרשת המקסימלי שהפאקטה יכולה לעבור דרכם לפני שתיזרק. כך השדה מגדיר את תוחלת החיים של הפאקטה.

לאחר שהגדרנו את כל השדות הללו אורזים IP\_header ראשוני. לאחר מכן אנחנו מחשבים את שדה ה-checksum על ידי חיבור כל השדות של ה-IP\_header (ב-16 ביטים) והפיכת הביטים של הסכום.

לאחר מכן אנחנו אורזים מחדש את ה-IP\_header ביחד עם שדה ה-checksum ובך השלמנו את האריזה של שכבת הרשת.

בכך השלמנו את האריזה של שלושת השכבות, אפליקציה (הודעות ה-CSV שהמחברת טוענת ושולחת), התעבורה (הסגמנט שנבנה, tcp\_header) והרשת (ה-ip datagram שנבנה, ip\_header). כך כל תעבורה שנבצע מהמחברת תקרה לאחר Encapsulation של שלושת השכבות האלו. התחלנו מרמת האפליקציה וארזנו את ההודעות בשכבות עד ליצירת פאקטות שנושאות את ההודעות.

#### 4. ניתוח התעבורה באמצעות WIRESHARK

בשלב הראשון ננתח תעבורה ראשונית שהתקבלה מהפעלת הפונקציה:

```
[30]: def demo_send(num_packets: int=3, delay_sec: float=1.0, flags: int=0x02):
    for i in range(num_packets):
        payload = f'Hello Packet {i}'.encode()
        transport.send(payload, flags=flags)
        time.sleep(delay_sec)
```

עבור כל אחת מהפקודות:

```
demo_send(num_packets=3, delay_sec=1.0, flags=0x02)
demo_send(num_packets=3, flags=0x18)
demo_send(num_packets=3, flags=0x10)
demo_send(num_packets=3, flags=0x01)
demo_send(num_packets=3, flags=0x04)
```

אציין שמספר הפורט 25215 הוא פורט שהוקצה למשתמש (אני, שהריץ את Wireshark) על ידי מערכת ההפעלה ומספר הפורט 12345 הוא מספר הפורט של האפליקציה (מחברת ה-Jupyter)

• `demo_send(num_packets = 3, delay_sec=1.0, flags=0x02)`

זו הגדרה לפאקט TCP שבה רק הדגל SYN פועל. במילים אחרות זו פאקטה שמהווה את השלב הראשון בתהליך 3-way-handshake. אבל לפאקטה אין כתובת מאזינה, במילים אחרות אין כאן באמת חיבור TCP בין שני צדדים, אין תקשורת sockets אלא העברה סתמית של פאקטות בין כתובת פורט כלשהי שהוקצתה על ידי מערכת ההפעלה לבין המחברת. ולכן נצפה שלאחר הפאקטה הראשונה, תישלח פאקטה שתבשר על Reset, כי אין אפשרות להתחיל חיבור. ובאמת ניתן לראות שלאחר כל פאקטה כזו שנשלחת למחברת (ומזהות על ידי Hello Packet 0 או Hello Packet 1 או Hello Packet 2 בשדה ה-TCP payload) מקבלים מהמחברת בתגובה פאקטת TCP שבה .Ack=1, Reset=1

No.	Source Port	Destination Port	Time	Protocol	Syn	Acknowledgment	Push	Reset	Fin	Flags	TCP payload
349	25215	12345	11.084745	TCP	1	0	0	0	0	0x0002	48656c6c6f205061636b65742030
350	12345	25215	11.084795	TCP	0	1	0	1	0	0x0014	
355	25215	12345	12.088255	TCP	1	0	0	0	0	0x0002	48656c6c6f205061636b65742031
356	12345	25215	12.088322	TCP	0	1	0	1	0	0x0014	
374	25215	12345	13.092120	TCP	1	0	0	0	0	0x0002	48656c6c6f205061636b65742032
375	12345	25215	13.092247	TCP	0	1	0	1	0	0x0014	

• `demo_send(num_packets=3, flags=0x18)`

זו הגדרה לפאקטת TCP שבה רק הדגלים ack, push פועלים. כמו מקודם אין באמת תקשורת בין הצדדים ולכן נצפה לקבל חזרה Reset=1. הפעם נשים לב שנקבל ack=0. הסיבה היא שהמחברת לא קיבלה לפני כן פאקטת SYN מהמשתמש ולכן היא לא מודעת או לא חושדת בקיום קשר.

20313	25215	12345	2116.209752	TCP	0	1	1	0	0	0x0018	48656c6c6f205061636b65742030
20314	12345	25215	2116.209836	TCP	0	0	0	0	1	0x0004	
20325	25215	12345	2117.214729	TCP	0	0	1	1	0	0x0018	48656c6c6f205061636b65742031
20326	12345	25215	2117.214837	TCP	0	0	0	0	1	0x0004	
20331	25215	12345	2118.217473	TCP	0	1	1	0	0	0x0018	48656c6c6f205061636b65742032
20332	12345	25215	2118.217552	TCP	0	0	0	0	1	0x0004	

שאר הפקודות מתנהגות באופן דומה למעט האחרונה:

• demo\_send(num\_packets=3, flags=0x04)

זו פאקטה שבה רק הדגל Reset פועל. כשפאקטה זו נשלחת אפילו לא נקבל פאקטות כתגובה מהמחברת. הסיבה היא שפאקטה מהסוג הזה לא מצפה לתגובה בחזרה אלא מבקשת מהמחברת לסגור את החיבור הקיים. אבל למעשה אין חיבור אמיתי ולכן לא נקבל תגובה מהמחברת.

33219	25215	12345	3169.293865	TCP	0	0	0	1	0	0x0004
33432	25215	12345	3170.296178	TCP	0	0	0	1	0	0x0004
33440	25215	12345	3171.299444	TCP	0	0	0	1	0	0x0004

Sequence Number: 0	(relative sequence number)	0000	02 00 00 00 45 00 00 36	00 01 00 00 40 06 7c bf	....E..6....@ .
Sequence Number (raw): 0		0010	7f 00 00 01 7f 00 00 01	62 7f 30 39 00 00 00 00	.....b.09....
[Next Sequence Number: 14		0020	00 00 00 00 50 04 20 00	a1 b4 00 00 48 65 6c 6c	....P. ....Hell
Acknowledgment Number: 0		0030	6f 20 50 61 63 6b 65 74	20 30	o Packet 0
Acknowledgment number (raw)					

התמונה הבאה מדגימה כיצד נראה הסגמנט של הפאקטות כפי שמוצגות ב-WIRESHARK.

Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1

Transmission Control Protocol, Src Port: 25215, Dst Port: 12345, Seq: 0, Len: 14

Source Port: 25215

Destination Port: 12345 ← **PORTS**

[Stream index: 21]

[Stream Packet Number: 1]

[Conversation completeness: Incomplete (45)]

[TCP Segment Len: 14]

Sequence Number: 0 (relative sequence number)

Sequence Number (raw): 0

[Next Sequence Number: 15 (relative sequence number)]

Acknowledgment Number: 0

Acknowledgment number (raw): 0

0101 .... = Header Length: 20 bytes (5)

Flags: 0x002 (SYN)

000. .... = Reserved: Not set

...0 .... = Accurate ECN: Not set

.... 0... = Congestion Window Reduced: Not set

.... .0.. = ECN-Echo: Not set

.... ..0. = Urgent: Not set

.... ...0 = Acknowledgment: Not set

.... .... 0... = Push: Not set

.... .... .0.. = Reset: Not set

.... .... .1. = Syn: Set

.... .... ...0 = Fin: Not set

[TCP Flags: .....S.]

Window: 8192

[Calculated window size: 8192]

Checksum: 0xa1b6 [unverified]

[Checksum Status: Unverified]

Urgent Pointer: 0

[Timestamps]

[Client Contiguous Streams: 0]

[Server Contiguous Streams: 1]

TCP payload (14 bytes)

Data (14 bytes)

Data: 48656c6c6f205061636b65742030

[Length: 14]

**FLAGS**

ניתן לראות גם את ה-TCP Payload ("Hello World 0"):

0000	02 00 00 00 45 00 00 36	00 01 00 00 40 06 7c bf	....E..6....@ .
0010	7f 00 00 01 7f 00 00 01	62 7f 30 39 00 00 00 00	.....b.09....
0020	00 00 00 00 50 02 20 00	a1 b6 00 00 48 65 6c 6c	....P. ....Hell
0030	6f 20 50 61 63 6b 65 74	20 30	o Packet 0

בשלב השני ננתח תעבורה ממחברת הג'ופיטר שהתקבלה כתוצאה מהעברת הודעות ה-CSV:

## Send Messages from CSV file

Iterate over the rows and send message by message

```
[43]: #Send messages from CSV file
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
    # Send the message using the RawTcpTransport class
    # (You may need to adjust flags and other parameters as needed)

    #TODO: uncomment the line below to send the messages
    transport.send(message.encode(), flags=0x18) # Example with PSH+ACK flags

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

למעשה שולחים פאקטות TCP עם הדגלים `ack=1, push=1`. ניתחנו מקודם את התנהגות התעבורה של פאקטות אלו. ההבדל הוא שכאן אנחנו מחלצים מקובץ ה-CSV שצירפנו בכל פעם את שדה ה-message ושולחים אותו. אכן ניתן לראות שכל הפאקטות שמתקבלות בעלות כותרת TCP מהצורה:

```
Transmission Control Protocol, Src Port: 25215, Dst Port: 12345, Seq: 1, Ack: 1, Len: 32
Source Port: 25215
Destination Port: 12345
[Stream index: 0]
[Stream Packet Number: 17]
  [Conversation completeness: Incomplete (40)]
  [TCP Segment Len: 32]
    Sequence Number: 1 (relative sequence number)
    Sequence Number (raw): 0
    [Next Sequence Number: 33 (relative sequence number)]
    Acknowledgment Number: 1 (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: 8192
```

## קובץ הלכידה לאחר שליחת הודעות ה-CSV בתוך המחברת נראה כך:

No.	Source Port	Destination Port	Time	Protocol	Syn	Acknowledgment	Push	Reset	Fin	Flags	TCP payload
1	25215	12345	0.000000	TCP	0	1	1	0	0	0x0018	697076362e6d736674636f6e6e656374746573742e636f6d2c302c307862626430
2	12345	25215	0.000051	TCP	0	0	0	1	0	0x0004	
3	25215	12345	0.104457	TCP	0	1	1	0	0	0x0018	697076362e6d736674636f6e6e656374746573742e636f6d2c302c307865346462
4	12345	25215	0.104562	TCP	0	0	0	1	0	0x0004	
5	25215	12345	0.209886	TCP	0	1	1	0	0	0x0018	697076362e6d736674636f6e6e656374746573742e636f6d2c302c307862626430
6	12345	25215	0.209951	TCP	0	0	0	1	0	0x0004	
7	25215	12345	0.315308	TCP	0	1	1	0	0	0x0018	697076362e6d736674636f6e6e656374746573742e636f6d2c302c307865346462
8	12345	25215	0.315410	TCP	0	0	0	1	0	0x0004	
9	25215	12345	0.419121	TCP	0	1	1	0	0	0x0018	697076362e6d736674636f6e6e656374746573742e636f6d2c302c307862626430
10	12345	25215	0.419220	TCP	0	0	0	1	0	0x0004	
11	25215	12345	0.524244	TCP	0	1	1	0	0	0x0018	697076362e6d736674636f6e6e656374746573742e636f6d2c302c307865346462
12	12345	25215	0.524342	TCP	0	0	0	1	0	0x0004	
13	25215	12345	0.631054	TCP	0	1	1	0	0	0x0018	697076362e6d736674636f6e6e656374746573742e636f6d2c312c307865346462
14	12345	25215	0.631209	TCP	0	0	0	1	0	0x0004	
15	25215	12345	0.734618	TCP	0	1	1	0	0	0x0018	697076362e6d736674636f6e6e656374746573742e636f6d2c312c307862626430
16	12345	25215	0.734695	TCP	0	0	0	1	0	0x0004	
17	25215	12345	0.840747	TCP	0	1	1	0	0	0x0018	7777772e6d736674636f6e6e656374746573742e636f6d2c302c307835633234
18	12345	25215	0.840873	TCP	0	0	0	1	0	0x0004	
19	25215	12345	0.945010	TCP	0	1	1	0	0	0x0018	7777772e6d736674636f6e6e656374746573742e636f6d2c302c307866636531
20	12345	25215	0.945091	TCP	0	0	0	1	0	0x0004	
21	25215	12345	1.048312	TCP	0	1	1	0	0	0x0018	7777772e6d736674636f6e6e656374746573742e636f6d2c302c307835633234
22	12345	25215	1.048476	TCP	0	0	0	1	0	0x0004	
23	25215	12345	1.154639	TCP	0	1	1	0	0	0x0018	7777772e6d736674636f6e6e656374746573742e636f6d2c302c307866636531
24	12345	25215	1.154743	TCP	0	0	0	1	0	0x0004	
25	25215	12345	1.260281	TCP	0	1	1	0	0	0x0018	7777772e6d736674636f6e6e656374746573742e636f6d2c302c307866636531
26	12345	25215	1.260364	TCP	0	0	0	1	0	0x0004	
27	25215	12345	1.365673	TCP	0	1	1	0	0	0x0018	7777772e6d736674636f6e6e656374746573742e636f6d2c302c307835633234
28	12345	25215	1.365783	TCP	0	0	0	1	0	0x0004	
29	25215	12345	1.470111	TCP	0	1	1	0	0	0x0018	7777772e6d736674636f6e6e656374746573742e636f6d2c312c307866636531
30	12345	25215	1.470203	TCP	0	0	0	1	0	0x0004	
31	25215	12345	1.575103	TCP	0	1	1	0	0	0x0018	7777772e6d736674636f6e6e656374746573742e636f6d2c312c307835633234
32	12345	25215	1.575235	TCP	0	0	0	1	0	0x0004	
33	25215	12345	1.680408	TCP	0	1	1	0	0	0x0018	646973636f72642e636f6d2c302c307838306634

וביתן לראות שכל פאקטה בעלת TCP Payload שאינו ריק נושאת את שדה ה-message של השורה המתאימה בקובץ ה-CSV. למשל, הפאקטה הראשונה נושאת את ההודעה:

0000	02 00 00 00 45 00 00 49	00 01 00 00 40 06 7c ac	E I @
0010	7f 00 00 01 7f 00 00 01	62 7f 30 39 00 00 00 00	b 09
0020	00 00 00 00 50 18 20 00	de f8 00 00 69 70 76 36	P . . . . . ipv6
0030	2e 6d 73 66 74 63 6f 6e	6e 65 63 74 74 65 73 74	.msftconnecttest
0040	2e 63 6f 6d 2c 30 2c 30	78 62 62 64 30	.com,0,0 xbbd0

ושורת ה-CSV הראשונה נושאת את שדה ה-message:

L	K	J	I	H	G	F	E	D	C	B	A
Connection Protocol	Response	Type	Length	message	timestamp	Destination Address	Source Address	dst_port	src_port	app_protocol	msg_id
	Message is a query	A	104	ipv6.msftconnecttest.com,0,0xbbd0				53	64155	DNS	1

הפאקטה ושורת ה-CSV שתיהן נושאות את אותה ההודעה:

Ipv6.msftconnecttest.com,0,0xbbd0

ולכן שורה מספר 1 בקובץ ה-CSV מתאימה לפאקטה מספר 1 בקובץ הלכידה.

דוגמה נוספת, בקובץ הלכידה (שמצורף לדו"ח), פאקטה מספר 63 נושאת את ה-Payload:

```

0000  02 00 00 00 45 00 00 3b 00 01 00 00 40 06 7c ba
0010  7f 00 00 01 7f 00 00 01 62 7f 30 39 00 00 00 00
0020  00 00 00 00 50 18 20 00 7d 2c 00 00 64 6e 73 2e
0030  67 6f 6f 67 6c 65 2c 31 2c 30 78 36 30 61 66

      .....;.....
      .....b-09....
      ....P.. },..dns.
      google,1 ,0x60af
  
```

ושורת ה-CSV מספר 32 נושאת את שדה ה-message:

שדה	תוכן	סוג	מספר	תוכן	מספר	מספר	מספר	מספר	מספר	מספר	מספר
UDP	Message is a response	AAAA	130	dns.google,1,0x60af	0.026362	192.168.1.18	1.1.1.1	59365	53	DNS	32

הפאקטה ושורת ה-CSV שתיהן נושאות את ההודעה:

dns.google,1,0x60af

ולכן שורה מספר 32 בקובץ ה-CSV מתאימה לפאקטה מספר 63 בקובץ הלכידה.

באופן דומה ניתן לזהות כל הודעה בקובץ ה-CSV גם בקובץ הלכידה.

## חלק שני

# פיתוח יישום רשת לקוח/שרת בפרוטוקול TCP

### 5. מבוא

נדרשנו לכתוב פרויקט המיישם צ'אט רב משתמשים שבו קיימים לקוחות שמתקשרים אחד עם השני ושרת שמנהל את החלפת המסרים ביניהם. כלומר, בהינתן השרת, לקוח א' יציין את שם הלקוח ב' אליו ירצה לפנות, והשרת ידאג להעביר את המסרים של לקוח א' אל לקוח ב'. בנוסף השרת יעביר ללקוח הודעות שקיבל מלקוחות אחרים שכן התקשורת בין הלקוחות אמורה להיות רב-כיוונית; ללקוח אמורה להיות היכולת גם לשלוח הודעות לכל לקוח אחר וגם לקבל הודעות מכל לקוח אחר.

לדו"ח זה מצורפים שתי אפליקציות שמרכיבות את היישום, האחת היא עבור הלקוח ששולח בקשות לשרת ומקבל מסרים מהשרת. השנייה היא עבור השרת שמנהל את רוב העבודה בתהליכים. הוא זה שמקבל קלטים ממספר לקוחות ויודע להתנהל לפיהם, יודע להעביר הודעות מלקוח פונה א' אל לקוח נמען ב' ומתקשר הוא עצמו עם לקוחות.

את היישום בחרתי לכתוב בשפת C++ ובסביבת Visual Studio 2022.

כאמור בנוסף לדו"ח מצורפים האפליקציות בתיקיות:

- Server\_interface
- Client\_interface
- Client\_2
- Client\_3
- Client\_4
- Client\_5

כאשר server\_interface היא אפליקציית השרת, client\_interface היא אפליקציית הלקוח, וכל השאר הם עותקים של אפליקציית הלקוח client\_interface שנועדו לרוץ במקביל כדי לקיים שיח. אין הבדל בין התיקיות האלו.

ניתן להריץ את הפרויקט גם עם שניים, שלושה או ארבעה עותקים מתוך החמישה שצורפו. ניתן גם לשכפל את התיקיות ולהריץ הרבה יותר לקוחות מהחמישה שצורפו. הקורא יכול להריץ כמה עותקים שירצה, לכן לשם הנוחות מצורפים חמישה עותקים, ניתן להשתמש בכולם או בחלקם לפי מידת הצורך.

כל קבצי הקוד בשתי האפליקציות server\_interface, client\_interface מלווים בתיעודים ובהסברים לנוחיות הקורא/ת.

## 6. ארכיטקטורת הפרויקט

כדי ליצור תקשורת בין יישומים בשפת C++ אנחנו משתמשים בספריית WinSock2 שמאפשרת לנו גישה לכל פעולות ה-Sockets להן נידרש כדי לאפשר תקשורת. כאשר מדובר בתקשורת מבוססת sockets אנחנו נדרשים לביצוע תהליכים טכניים לפני שנוכל לתקשר עם אפליקציות אחרות ולאחר שנסיים לתקשר עימן. התהליכים משתנים במעט בין הלקוח לשרת, ואלו הם:

עבור הלקוח נבצע:

- אתחול ספריית Winsock
- יצירת socket שימש את הלקוח להתחבר לשרת (פונקציית יצירת socket)
- ציון כתובת השרת וחיבור בין socket הלקוח לבין השרת (פונקציית connect)
- ניהול תקשורת עם השרת באמצעות שליחה וקבלת הודעות (פונקציות send, recv)
- בסוף השימוש, קרי התנתקות של הלקוח מהשרת או יציאה מהצ'אט, נסגור את ה-socket (פונקציית closesocket)
- קריאה לפונקציית cleanup של ספריית Winsock

עבור השרת נבצע:

- אתחול ספריית Winsock
- יצירת socket שיאזין ללקוחות נכנסים (פונקציית socket)
- קשירת ה-socket המאזין אל הכתובת שבה נציב את השרת, כלומר אל כתובת ה-ip ואל מספר הפורט שנקצה לו (פונקציית Bind)
- רק לאחר שיצרנו את ה-socket וקשרנו אותו לכתובת שלו, נתחיל בהאזנה ללקוחות נכנסים. (פונקציית listen)
- בעת האזנה, נקבל לקוחות חדשים ונשמור אותם בבסיס נתונים שיפורט בהמשך (פונקציית accept)
- לאחר קבלת הלקוח ננהל תקשורת באמצעות קבלה ושליחת הודעות בין הלקוח לשרת (פונקציות send, recv). חשוב לציין שכל לקוח פועל בצורה א-סינכרונית. כלומר הפעולות של לקוח אחד אינן תלויות בזמני הפעולה של לקוח שני. התהליכים של כל לקוח קורים במקביל.
- באופן עקרוני השרת אמור לעבוד תמיד (ריצה אינסופית) כי בלעדיו הצ'אט לא קיים, בשונה מהלקוח שכן אמור להתנתק מהצ'אט בשלב מסוים (לא רץ אינסופית). עם זאת, אנחנו מתעסקים עם sockets ולכן למען הסדר הטוב נציין שיש לסגור את ה-socket בסוף השימוש. (פונקציית closesocket).
- בהמשך לסעיף הקודם, קריאה לפונקציית cleanup של ספריית Winsock

כאמור הפרויקט מורכב משתי אפליקציות, לקוח ושרת ולכן נקדיש פירוט לכל אחת.



## 6.1. Server\_interface – אפליקציית השרת

תיקייה זו מורכבת מ-4 קבצי header שם מרוכזים הפונקציות עם תיעודים כלליים ומ-4 קבצי cpp גם הם עם תיעודים מפורטים ובהם בפועל מיושמים התהליכים. פירוט רחב נמצא בתיעוד שבפרויקט, כאן נציג את קבצי ה-header

Client\_info.h – בקובץ זה מוגדר struct עם פרטים על הלקוח, שבו אנחנו שומרים את השם של הלקוח (השם ייחודי לכל לקוח – דרישות הפרויקט מאפשרות לנו להניח זאת), ואת ה-socket של הלקוח. נגדיר גם פונקציה למציאת לקוח בודד בתוך רשימת לקוחות בהתבסס על השם שלו. כפי שניתן להבין, בסיס הנתונים שבו נשמור את הלקוחות יהיה רשימה.

```
1  /* Made by Liran Dagan 215609397 */
2  #pragma once
3  #include <iostream>
4  #include <WinSock2.h>
5  #include <string>
6  #include <list>
7
8  using namespace std;
9
10 // Each client has a name and a socket to which the server sends info to and receives info from
11 struct ClientInfo {
12     string name;
13     SOCKET socket;
14
15     bool operator==(const ClientInfo&);
16 };
17
18 ClientInfo* findByName(const string&, list<ClientInfo>&);
19
```

socket\_setups.h – בהמשך לפרוצדורות שצוינו בעמודים הקודמים, קובץ זה מכיל פעולות מקדימות שאנו צריכים לקיים כדי לאפשר תקשורת מבוססת sockets. בקובץ מופיע מימוש בסיסי של אתחול ספריית Winsock, יצירת ה-socket המאזין של השרת וקשירתו לכתובת השרת. ההאזנה וקבלת הלקוחות מתרחשות בנפרד בפונקציית ה-main שבקובץ server.h שיוצג מיד.

```
1  /* Made by Liran Dagan 215609397 */
2  #pragma once
3  #include <WinSock2.h>
4  #include <WS2tcpip.h>
5  #include <string>
6  #pragma comment(lib, "ws2_32.lib")
7
8  using namespace std;
9
10 /*
11  These are the steps the server handles to perform socket interactions:
12  1. Initialize winsock lib
13  2. Create the server socket
14  3. Bind ip (Which is 0.0.0.0) and port (which is 12345) to the socket
15  -----
16  4. Listen on the socket
17  5. Accept clients
18  -----
19  6. 'recv' for receiving messages from clients and 'send' for forwarding them to other clients
20  -----
21  7. Close the server socket when finished
22  8. Cleanup
23  ...
24  The goal: Use 'recv' to get inputs from the users, Use 'send' to instruct users and
25  forward their messages onwards
26  */
27
28 // step 1: Initialize WinSock version 2.2
29 bool initialize();
30
31 // step 2: Create a listening TCP socket for the server to accpet clients
32 bool createTCPsocket(SOCKET& listener);
33
34 // step 3.1: Create the server's Address details, ip and port
35 bool createAddress(sockaddr_in& serverAddr, const string& ip, const int portNumber);
36
37 // step 3.2: Bind our TCP listening socket to the address details we created
38 bool assignAddress(SOCKET& listenSocket, sockaddr_in& serverAddr);
39
```

interact\_with\_client.h – בקובץ זה מרוכזות הפונקציות שמהוות את הלוגיקה שבתקשורת עם כל לקוח כאשר אנחנו משתמשים בפונקציית על interact שנעזרת בפונקציות נוספות כדי לנהל את התהליכים מול הלקוחות.

```
1  /* Made by Liran Dagan 215609397 */
2  #pragma once
3  #include <iostream>
4  #include <string>
5  #include <list>
6  #include "client_info.h"
7  using namespace std;
8
9  /* Given 2 strings, compare them case insensitively(for example "liRaN" == "LIRaN") */
10 bool equalStrings(const string& str1, const string& str2);
11
12 /* Given a client's name and socket we create and save his info in a 'ClientInfo' struct and add him to
13  | a list containing all active chatters */
14 ClientInfo createClient(string& name, SOCKET& socket, list<ClientInfo>& all_clients);
15
16 /* Given any client's message to the server we first have to check if he entered 'quit'
17  | If he did, then we notify the client on his disconnection and return true
18  | If he didn't, we return false */
19 bool isQuit(string& message, ClientInfo& client);
20
21 /* Given a client's input of a target he wants to message, the server checks if the target
22  | is valid and sends feedback to the client. Errors to be detected are:
23  | 1. target is the client himself (logical error, client is trying to message himself).
24  | 2. target doesn't exist (technical error, the target can not be found)
25  | If there is no error we notify the client that we found the target, else we
26  | send the error to the client. */
27 bool isError(string& targetName, ClientInfo& client, list<ClientInfo>& all_clients);
28
29 /* If the client disconnected (due to 'quit' message or due to connection error)
30  | Then we discard his socket and remove him from the list of active chatters */
31 void deleteClient(ClientInfo& client, list<ClientInfo>& all_clients);
32
33
34 /* 'Interact' is the main function that manages the interactions between the server and the client.
35  | It Handles the server's actions with a single client and forwards messages from the client
36  | to a target client.
37  | If needed, the server sends feedback messages regarding errors.
38  | Interact utilizes all the functions from above.
39
40  | In short, we do the following:
41  | 0. We got a message from the client
42  | 1. Check for 'quit' (isQuit), if the client didn't send quit we proceed to 2., else we stop running
43  | and remove him (deleteClient).
44  | 2. Use a 'mode' variable that gets 0, 1 or 2 to distinguish between actions as followed:
45  | mode = 0 -> We receive client's name and save his details (createClient)
46  | mode = 1 -> we receive the target client and check for errors (isError)
47  | mode = 2 -> We got a message to deliver and we send it to the target client by finding his socket in the clients list
48  | 3. return to 0.
49  | */
50
51 void Interact(SOCKET clientSocket, list<ClientInfo>& all_clients);
```

Server.h – בקובץ זה פונקציית ה-main, בה אנחנו מבצעים את כל פעולות ה-socket המקדימות כדי לאפשר לשרת לתקשר עם הלקוחות. כלומר, אנחנו מבצעים את כל הפעולות הקיימות בקובץ socket\_setups.h ומוסיפים האזנה וקבלת לקוחות.

כדי לנהל את כל הלקוחות במקביל אנחנו צריכים לשמור מידע על כולם. כלומר אנחנו צריכים מבנה נתונים כלשהו כדי לנהל את הלקוחות, ובאמת כפי שנאמר אנחנו משתמשים ברשימה. בפרויקט שלנו נזדקק למבנה נתונים שיאפשר להוסיף, למחוק ולחפש לקוחות בתוך המאגר. נוכל לבצע זאת עם המון סוגים של מבנים. אפשר למשל לשמור עץ AVL מאוזן שמסדר את הלקוחות לפי סדר האלף-בית של שמותיהם, או להשתמש בדרכים מתוחכמות יותר לניהול המאגר. עם זאת, זו לא גולת הכותרת של הפרויקט הזה, לכן נפשט מעט את המימוש ואת ההסברים. כאמור נשמור את הלקוחות בתוך רשימה מקושרת (list) מסוג ClientInfo אותה נאתחל בהתחלה כריקה. כל לקוח שמתחבר לשרת יתווסף לרשימה, כל לקוח שיתנתק מהשרת יימחק מהרשימה ואם נרצה להעביר הודעה אל לקוח נממן נוכל פשוט לחפש אותו ברשימה ולקבל פוינטר אל הפרטים שלו.

חשוב לציין – עבור כל לקוח שמבוצע עליו accept אנחנו יוצרים thread שמטרתו לפצל את הקשב של השרת לכמה לקוחות במקביל. פונקציית ה-thread היא Interact מהקובץ interact\_with\_client.h לה אנחנו נותנים את ה-socket של הלקוח הנכנס ורפרנס למבנה הנתונים, הרשימה, ששומרת את כל הלקוחות. בדרך הזו נוצרת א-סינכרוניות ומשתמשים אינם תלויים זה בזה!

```
1  /* Made by Liran Dagan 215609397 */
2  #pragma once
3  #include <iostream>
4  #include <tchar.h>
5  #include <thread>
6  #include <list>
7  #include "client_info.h"
8  #include "socket_setups.h"
9  #define MAX_USERS 100
10 using namespace std;
11
12 /* In main we do all socket procedures from socket_setups.h along with listening and
13    accepting clients. We check for errors in each procedure. We then create an empty
14    list of type ClientInfo in which we will store all the active clients.
15    Once a client comes in and we accept him, we send his socket to the 'Interact'
16    function from interact_with_client.h by a thread to isolate the server and him, along with a
17    reference to the list of clients we created.
18    By using threads we split the server's attention to each client in particular creating desynchronization
19    which is essential for chatting between one and another */
20 int main();
```

מתוך server.cpp:

```
list<ClientInfo> clients; // Stores the names & sockets of all clients participating in chat
while (true) {
    // step 5: Accept and start communicating with clients
    SOCKET clientSocket = accept(listenSocket, NULL, NULL); // A client tries to access the server
    if (clientSocket == INVALID_SOCKET)
    {
        cout << "Client socket is invalid" << endl;
    }
    else // We split the server's attention to each reaching client, so we get a multi client system
    {
        thread T(Interact, clientSocket, ref(clients)); // Each client gets his own treatment simultaneously with threads
        T.detach();
    }
}
closesocket(listenSocket);
WSACleanup();
return 0;
```

## 6.2. Client\_interface – אפליקציית הלקוח

תיקייה זו מורכבת מ-3 קבצי header שם מרוכזים הפונקציות עם תיעודים כלליים ומ-3 קבצי cpp גם הם עם תיעודים מפורטים ובהם בפועל מיושמים התהליכים. פירוט רחב נמצא בתיעוד שבפרויקט, באן נציג את קבצי ה-header.

socket\_setups.h – בדומה לאפליקציית השרת, גם באפליקציית הלקוח נגדיר קובץ עם פעולות מקדימות שאנו צריכים לקיים כדי לאפשר תקשורת מבוססת sockets. בקובץ מופיע מימוש בסיסי של אתחול ספריית Winsock, יצירת socket והתקשרות לשרת וההתחברות לשרת.

```
1  /* Made by Liran Dagan 215609397 */
2  #pragma once
3  #include <WinSock2.h>
4  #include <WS2tcpip.h>
5  #include <string>
6  #pragma comment(lib, "ws2_32.lib")
7  using namespace std;
8
9  /*
10  These are the steps the client handles to perform socket interactions:
11  1. Initialize winsock
12  2. Create communicating socket
13  3. Connect to server
14  4. 'send' for delivering messages to server and 'recv' for receiving messages from server
15  5. Close the socket when finished
16  6. Cleanup
17  ...
18  The goal: Use 'send' and 'recv' methods with the server in order to
19  1. forward messages to a specified client
20  2. get messages from other clients
21  ...
22  */
23
24  // step 1: Initialize WinSock version 2.2
25  bool initialize();
26
27  // step 2: Create a TCP socket to contact the server with
28  bool createTCPsocket(SOCKET& serverSocket);
29
30  // step 3.1: Create the address details of the server the client connects - ip and port
31  bool createAddress(sockaddr_in& serverAddr, const string& ip, const int portNumber);
32
33  // step 3.2: Connect to the server using the 'connect' function
34  bool connectToServer(SOCKET& serverSocket, sockaddr_in& serverAddr);
35
```

interact\_with\_server.h – בקובץ זה מרוכזת כל הלוגיקה של אפליקציית הלקוח משום שבקובץ זה מפורטות כל הפעולות בין הלקוח לשרת מצד הלקוח. הפונקציות שבקובץ הזה מהוות את הלוגיקה שבתקשורת כאשר אנחנו משתמשים בפונקציות על sendMessage, receiveMessage שנעזרות בפונקציות נוספות כדי לנהל תהליכים מול השרת.

```

1  /* Made by Liran Dagan 215609397 */
2  #pragma once
3  #include <WinSock2.h>
4  #include <iostream>
5  #include <thread>
6  #include <string>
7
8  using namespace std;
9
10 // return str1==str2 case isensitive ('a'=='A' for example)
11 bool equalStrings(const string&, const string&);
12
13 // Send client's name to the server to be known. Return success (true) or fail (false)
14 bool registerClient(SOCKET& server);
15
16
17 /* Send name of another client we want to message. Server will respond with a feedback message.
18    Return success (true) or fail (false) */
19 bool searchTarget(SOCKET& server);
20

```

```

22 /* A thread function. Here we handle all of client's messages to the server in correlation to 'mode'
23    'sendMessage' utilizes all the functions above.
24    In short we do the following:
25    1. Use the 'mode' variable that gets 0, 1 or 2 to distinguish between actions as followed:
26        mode = 0 -> Send client's name to server (registerClient)
27        mode = 1 -> Specify target (searchTarget)
28        mode = 2 -> Send message to server to deliver to target
29    2. If connected==true meaning client didn't quit and no connection error occurred then go to 1.
30    else quit function */
31 void sendMessage(SOCKET server);
32
33
34 /* A thread function. Here we handle all of server's messages to the client including error feedbacks
35    and messages from other clients.
36    In short we do the following:
37    0. Get message/prompt from server
38    1. if got sameUser or UserNotFound prompts then stay in mode = 1 for sendMessage
39    2. if got userFound prompt then proceed to mode = 2 for sendMessage
40    3. if got Quit prompt or a disconnection error then connected=false and 6.
41    4. if none of the above then we actually got a message from another client and not a prompt, so we display it
42    5. Go to 0.
43    6. quit both thread functions sendMessage, receiveMessage and return to main to finish program */
44 void receiveMessage(SOCKET server);
45

```



client.h – בקובץ זה פונקציית ה-main, בה אנחנו מבצעים את כל פעולות ה-socket המקדימות כדי לאפשר ללקוח לתקשר עם השרת. אנחנו מבצעים את כל הפעולות הקיימות בקובץ socket\_setups.h. חשוב לציין – אנחנו יוצרים שני thread כאשר פונקציות ה-thread הן:

sendMessage receiveMessage, מהקובץ interact\_with\_server.h להן אנחנו נותנים את ה-socket שאיתו מתקשרים עם השרת. שילוב שני ה-threads מאפשר לנו לשלוח הודעות ולקבל הודעות בו זמנית.

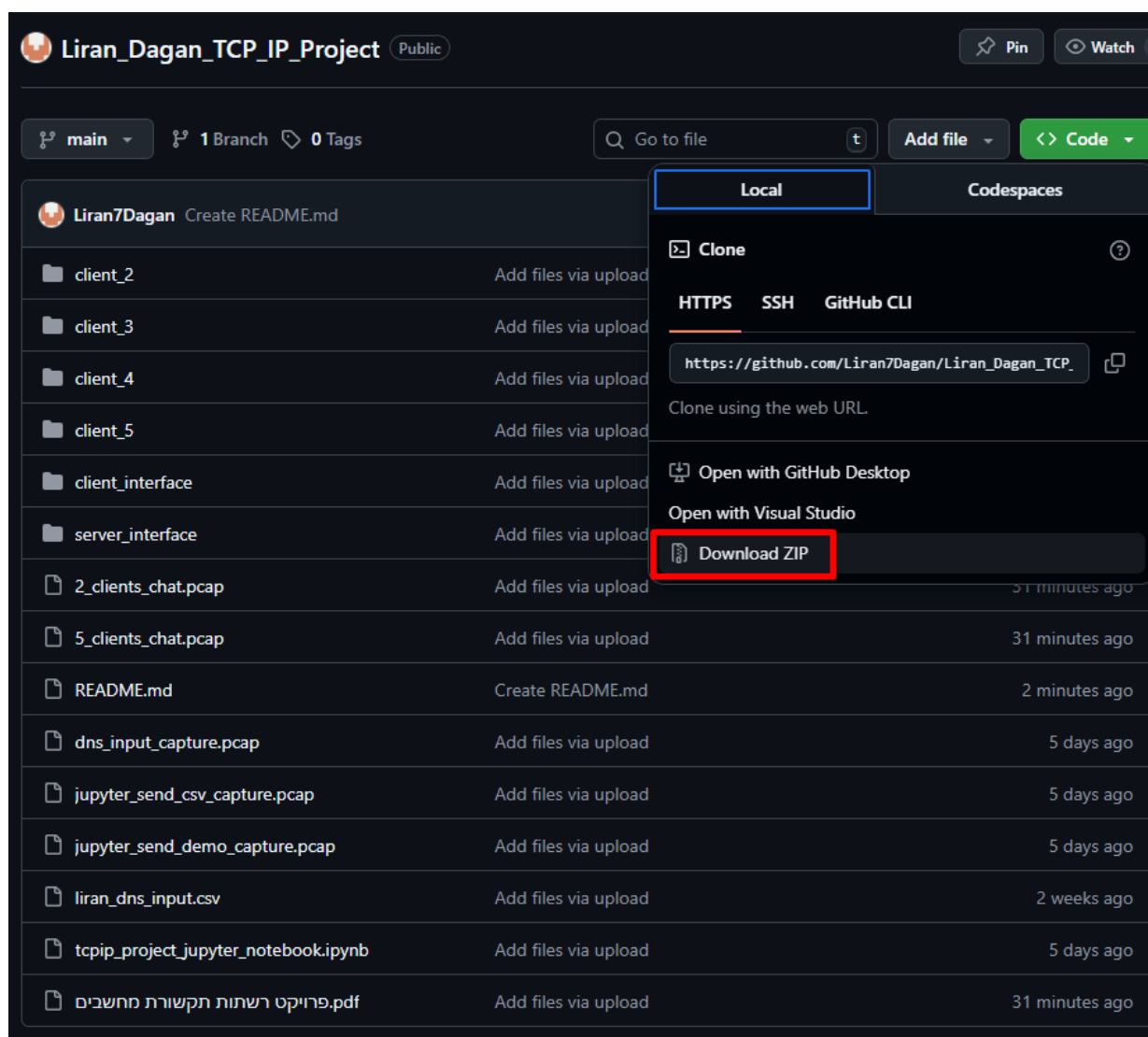
```
1  /* Made by Liran Dagan 215609397 */
2  #include "socket_setups.h"
3  #include "interact_with_server.h"
4  using namespace std;
5
6  /* In main we do all socket procedures from socket_setups.h.
7   We check for errors in each procedure.
8   We then create 2 threads, one for each function: sendMessage, receiveMessage
9   and we give both of them the socket with which we communicate with the server.
10  By joining the threads we communicate with the server while having the two functions enable
11  each other. receiveMessage reacts to the server's feedback to sendMessage, and
12  sendMessage adjusts it's actions by reacting to receiveMessage's response to the server's
13  feedback. As a result we get one coherent application that sends and receives messages
14  at the same time. */
15  int main();
16
```

## 7. הוראות התקנה והרצה

לגיטהב מצורפים התיקיות client\_interface, server\_interface, client\_2. כאמור שלושת האפליקציות נכתבו בשפת C++ ולכן כדי להריץ אותן על המחשב יש לפתוח את קבצי ה-solution שלהן בויזואל סטודיו.

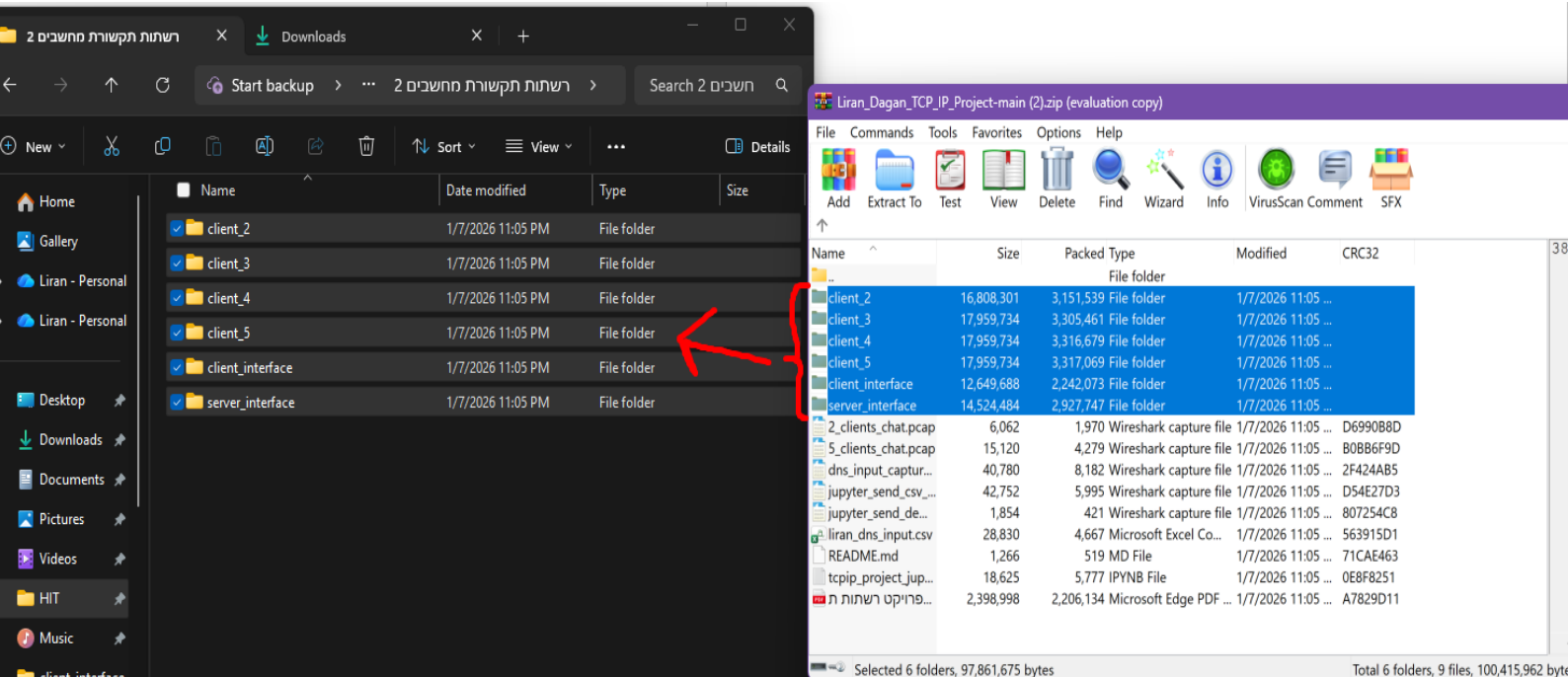
צריך לפתוח חלון ויזואל סטודיו אחד שיריץ את השרת, חלון אחד שיריץ את לקוח 1 וחלון אחד שיריץ את לקוח 2. במידת הרצון ניתן לשכפל את תיקיית הלקוח ולפתוח עוד חלונות שיריצו עוד לקוחות.

לכן אני מציע להוריד בקובץ ZIP את תוכן הגיטהב ולחלץ / לגרור את שתי התיקיות אל נתיב מועדף במחשב.

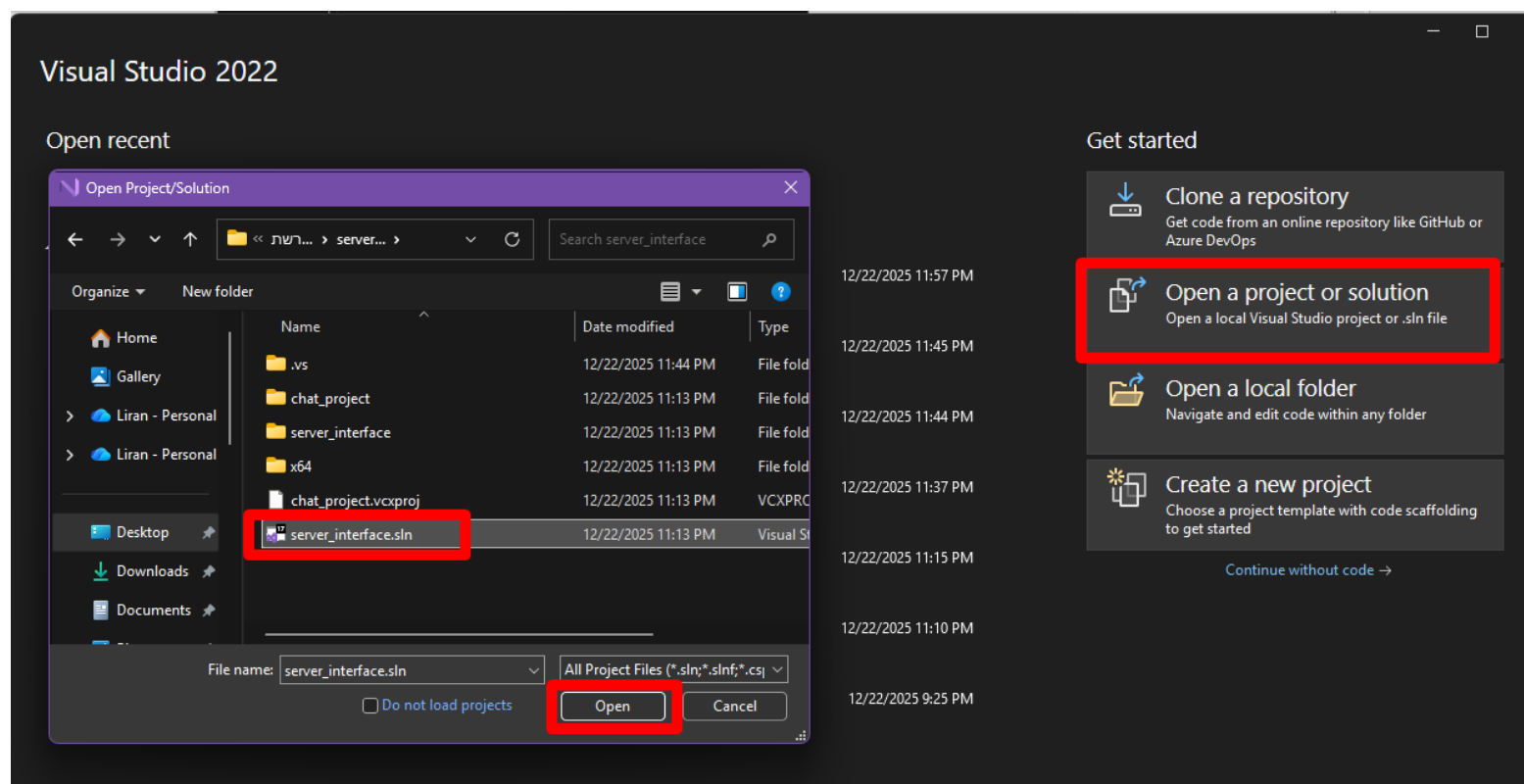




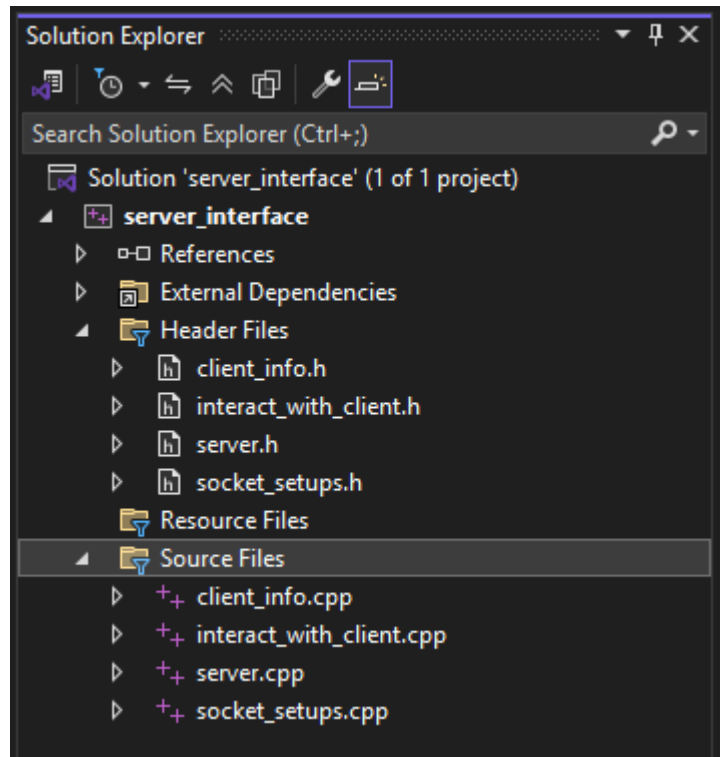
לאחר הורדת ה-ZIP, גרירת התיקיות:



לאחר מכן, בחלונות של ויזואל סטודיו יש לפתוח את server\_interface.sln שנמצא בתיקייה שנגררה.



בשלב זה ניתן לפתוח ולראות את כל קבצי הקוד - ה-header וה-cpp של השרת בתוך ויזואל סטודיו.

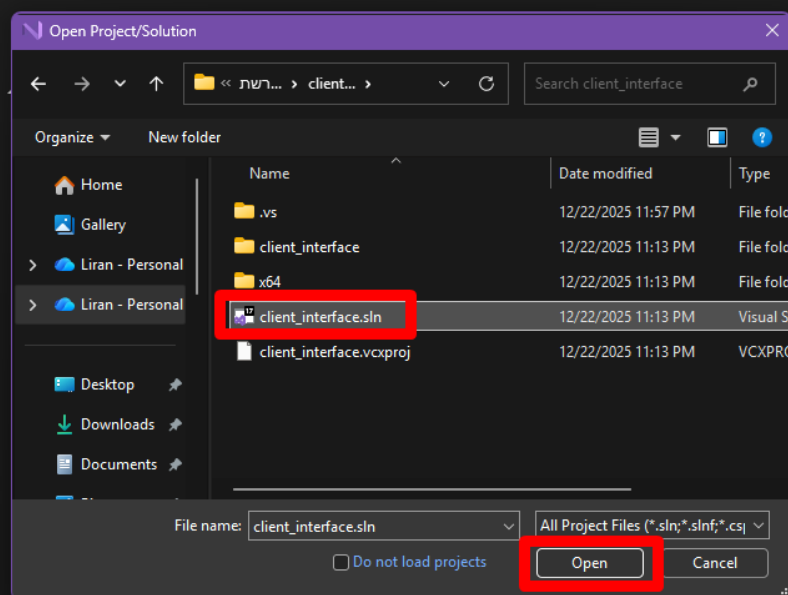


עכשיו צריך לפתוח את אפליקציית הלקוח, בשביל כך צריך לפתוח חלון נוסף של ויזואל סטודיו שיעבוד במקביל לחלון שכבר נפתח עבור השרת.

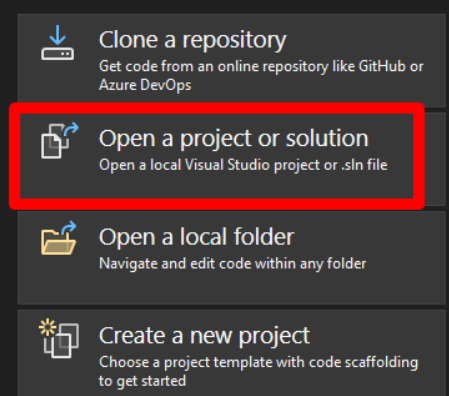
לאחר שנפתח חלון ויזואל סטודיו חדש נפתח את אפליקציית הלקוח באותו אופן, נפתח את client\_interface.sln שנמצא בתיקייה שנגררה:

## Visual Studio 2022

### Open recent

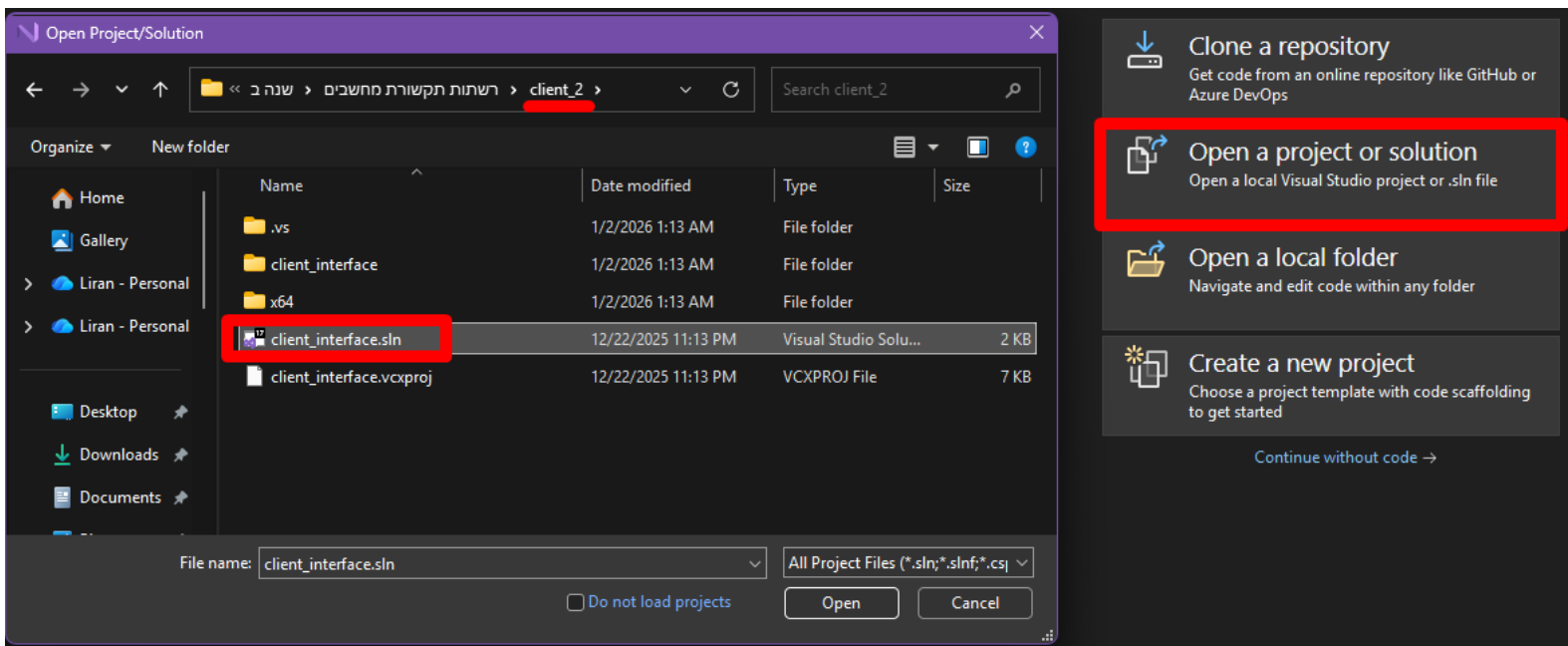


### Get started

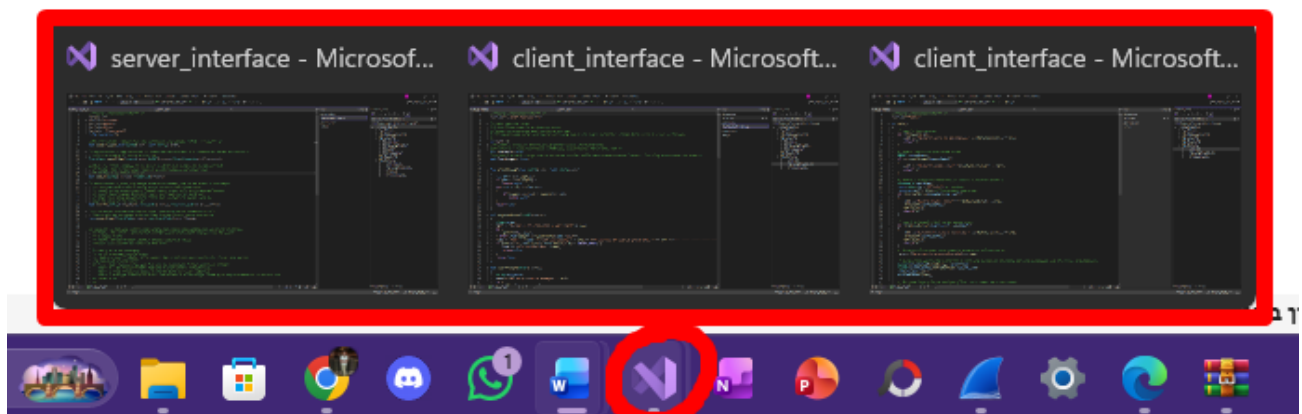


בשלב זה ניתן לפתוח ולראות את כל קבצי הקוד – ה-header וה-cpp של הלקוח בתוך ויזואל סטודיו.

נעת נצטרך לפתוח גם את הלקוח השני אם נרצה להריץ שני משתמשים.  
ובאופן זה נפתח בחלון נפרד את client\_2.sln שנמצא בתיקייה שנגררה:

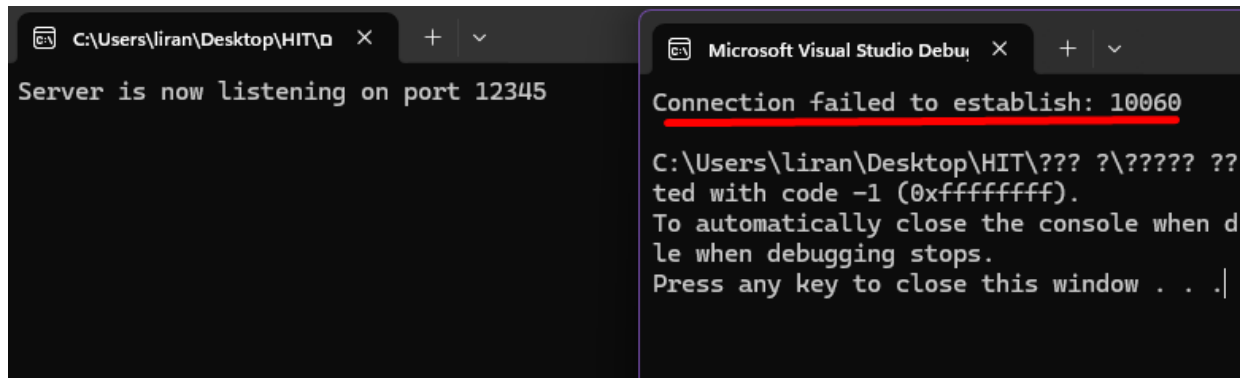


אם נרצה לפתוח עוד לקוחות, כאמור נשכפל את תיקיית client\_interface ונפתח בחלונות ויזואל סטודיו נפרדים את קובץ ה-sln. של כל אחד.  
עד לקבלה של לפחות שלושה חלונות ויזואל סטודיו שפועלים וניתנים להרצה במקביל. אחד בלבד עבור השרת ושניים או יותר עבור הלקוחות.



בכך מסתכם שלב ההתקנה, נעבור לשלב ההרצה.

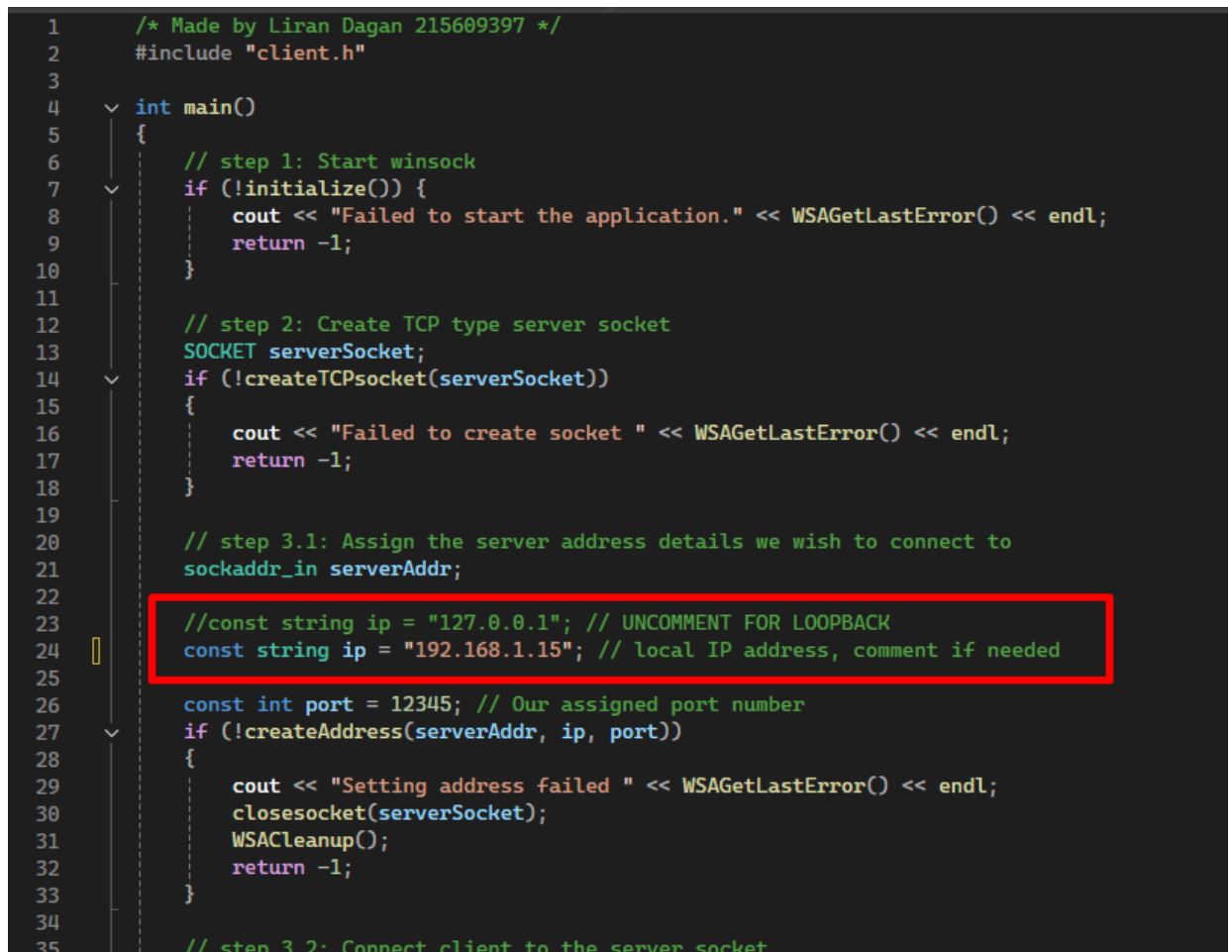
שגיאה צפויה היא שגיאה מהסוג:



```
C:\Users\liran\Desktop\HIT\ > Server is now listening on port 12345

Microsoft Visual Studio Debug Console:
Connection failed to establish: 10060
C:\Users\liran\Desktop\HIT\??? \????? ??
ted with code -1 (0xffffffff).
To automatically close the console when d
le when debugging stops.
Press any key to close this window . . .|
```

הסיבה: באפליקציית הלקוח, בקובץ client.cpp, שורה 23:



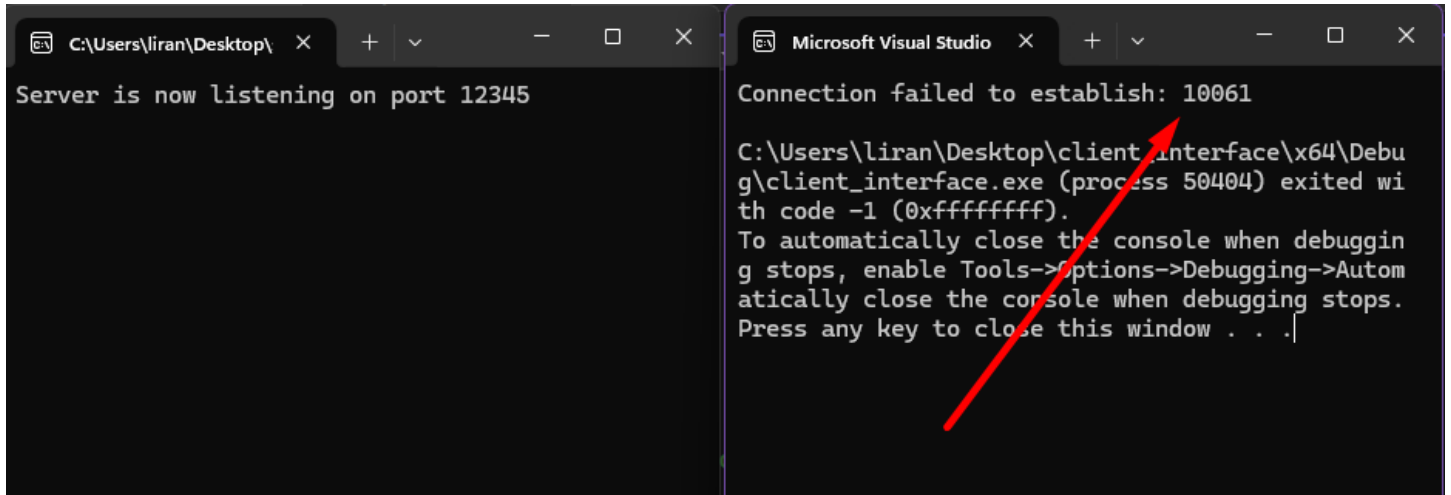
```
1  /* Made by Liran Dagan 215609397 */
2  #include "client.h"
3
4  int main()
5  {
6      // step 1: Start winsock
7      if (!initialize()) {
8          cout << "Failed to start the application." << WSAGetLastError() << endl;
9          return -1;
10     }
11
12     // step 2: Create TCP type server socket
13     SOCKET serverSocket;
14     if (!createTCPsocket(serverSocket))
15     {
16         cout << "Failed to create socket " << WSAGetLastError() << endl;
17         return -1;
18     }
19
20     // step 3.1: Assign the server address details we wish to connect to
21     sockaddr_in serverAddr;
22
23     //const string ip = "127.0.0.1"; // UNCOMMENT FOR LOOPBACK
24     const string ip = "192.168.1.15"; // local IP address, comment if needed
25
26     const int port = 12345; // Our assigned port number
27     if (!createAddress(serverAddr, ip, port))
28     {
29         cout << "Setting address failed " << WSAGetLastError() << endl;
30         closesocket(serverSocket);
31         WSACleanup();
32         return -1;
33     }
34
35     // step 3.2: Connect client to the server socket
```

לפני ההרצה יש להתאים את כתובת ה-IP. ניתן להשתמש בכתובת loopback או להשתמש בכתובת ה-IP המקומית של מחשב הבודק. מומלץ להשתמש בכתובת IP אמיתית אם כי האפליקציות יעבדו כך או כך.

יש להריץ את אפליקציית השרת ראשונה, ורק לאחר מכן את אפליקציות הלקוח.

אם ההרצה תקרה בסדר הפוך (קודם לקוח ואז שרת), השרת יעבוד כרגיל אבל הלקוח יקבל את ההדפסה (שהיא חלק מהקוד): Connection failed to establish: 10061, כלומר הודעה לכך שהשרת לא נמצא ולכן לא ניתן להתחבר אליו. (השרת הורץ רק אחרי הלקוח ולכן הוא עדיין לא קיים בזמן הרצת הלקוח).

## דוגמה לשגיאת הרצת הלקוח לפני השרת:



```
C:\Users\liran\Desktop\ X + - □ X
Server is now listening on port 12345

Microsoft Visual Studio X + - □ X
Connection failed to establish: 10061

C:\Users\liran\Desktop\client_interface\x64\Debug\client_interface.exe (process 50404) exited with code -1 (0xffffffff).
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .|
```

בנוסף, האפליקציות מטפלות בשגיאות (כמו זו שבדוגמה) ולכן לא אמורות לקרות קריסות בלתי צפויות. אם קורות שגיאות בלשהן האפליקציות לא יקראו אלא ידפסו על המסך הודעות מתאימות. עוד יש לציין כי המסך שמקבלים מאפליקציית השרת אינו אינטראקטיבי אלא אינפורמטיבי, הוא עוזר להמחיש התהליכים שקורים בזמן אמת בין לקוחות לשרת ובין לקוחות ללקוחות ולכן לא מקבל בעצמו קלט מהמשתמש אלא בסך הכל מציג מידע שעובר דרכו. המסך שמקבלים מאפליקציית הלקוח לעומת זאת הוא כמובן אינטראקטיבי ואינפורמטיבי.

## 8. דוגמאות קלט ופלט

דוגמה לשיחה בין שני משתמשים:

Microsoft Visual Studio Del

Connection to server established  
Welcome to the chat! What is your name?  
Liran  
Hello Liran! Have fun chatting.  
You can quit any time by typing 'QUIT'.  
-----  
Who do you want to message?  
itzik  
User Was Not Found.  
-----  
Who do you want to message?  
yaron  
User Was Not Found.  
-----  
Who do you want to message?  
Maor  
Connection created  
-----  
Hey Maor!!!  
  
Maor: Whats upppppppppp Liranannnn  
  
I'm doing great  
  
There's a big football game today  
  
Maor: I know today is Real Madrid vs Barcelona  
  
Right it will be a good game for sure!!!  
  
Do you wanna come over and watch it together? We c  
an order pizza ;)  
  
Maor: YEAH For sure  
  
Great so be at my place in about 15 minutes  
  
Maor: I'm on my way, see you then!  
  
quit

C:\Users\liran\Desktop\HIT\??? ?\????? ?????? ????  
??\client\_interface\x64\Debug\client\_interface.exe

Microsoft Visual Studio Debu

Connection to server established  
Welcome to the chat! What is your name?  
Maor  
Hello Maor! Have fun chatting.  
You can quit any time by typing 'QUIT'.  
-----  
Who do you want to message?  
Maor  
Please Message a Different User.  
-----  
Who do you want to message?  
Liran  
Connection created  
-----  
Liran: Hey Maor!!!  
  
Whats upppppppppp Liranannnn  
  
Liran: I'm doing great  
  
Liran: There's a big football game today  
  
I know today is Real Madrid vs Barcelona  
  
Liran: Right it will be a good game for sure!!!  
  
Liran: Do you wanna come over and watch it together?  
We can order pizza ;)  
  
YEAH For sure  
  
Liran: Great so be at my place in about 15 minutes  
  
I'm on my way, see you then!  
  
quit

C:\Users\liran\Desktop\HIT\??? ?\????? ?????? ??????\n  
client\_2\x64\Debug\client\_interface.exe (process 5257

```
C:\Users\liran\Desktop\HIT\m X + v
Server is now listening on port 12345
New client joined: 'Liran'
New client joined: 'Maor'
'Liran' Tried to reach a non existent client: 'itzik'
Maor Tried to message himself
'Liran' Tried to reach a non existent client: 'yaron'
'Liran' Reaches 'Maor'. Server is ready to transmit messages!
'Maor' Reaches 'Liran'. Server is ready to transmit messages!
'Liran' TO 'Maor': Hey Maor!!!
'Maor' TO 'Liran': Whats uppppppppppp Liranannnn
'Liran' TO 'Maor': I'm doing great
'Liran' TO 'Maor': There's a big football game today
'Maor' TO 'Liran': I know today is Real Madrid vs Barcelona
'Liran' TO 'Maor': Right it will be a good game for sure!!!
'Liran' TO 'Maor': Do you wanna come over and watch it together? We can order pizza ;)
'Maor' TO 'Liran': YEAH For sure
'Liran' TO 'Maor': Great so be at my place in about 15 minutes
'Maor' TO 'Liran': I'm on my way, see you then!
'Liran' Disconnected
'Maor' Disconnected
|
```

קצת על התקשורת בין השרת ללקוחות בשיחה זו:

המשתמשים שולחים לשרת את שמותיהם "לירן", "מאור", השרת זוכר ושומר אותם. בכל פעם שהמשתמשים מנסים לפנות ללקוח שהשרת אינו מכיר, כמו "איציק" ו"ירון" הוא שולח להם פידבק "המשתמש לא נמצא". באופן דומה אם המשתמש פונה אל עצמו כמו שעשה מאור, השרת משיב: "בבקשה פנה למשתמש אחר". בשלירן פונה אל מאור (שהשרת מכיר) השרת מודיע לו שנוצר חיבור וההודעות מועברות מכאן והילך אל מאור. באופן זה השרת עושה עבור מאור כשהוא פונה אל לירן. מכאן המשתמשים משוחחים ביניהם. משתמש מתנתק מהשרת על ידי הקלט "quit", כמו שעושים מאור ולירן בהודעות האחרונות שלהם. שאלה שאולי נשאלת היא: מה היה קורה אם לירן היה שולח הודעה למאור אחרי שהוא התנתק? כלומר, מאור לא מזוהה יותר על ידי השרת ולירן בכל זאת מנסה לשלוח לו הודעות.

במקרה כזה השרת לא יזהה את מאור ולכן יודיע ללירן על סיום החיבור ויחזיר אותו למצב ההתחלתי, כלומר יבקש ממנו לציין לקוח חדש שהוא רוצה לתקשר איתו.

בתמונה ניתן לראות תהליך שבו לירן מנסה לשלוח הודעה למאור לאחר שכבר התנתק. השרת מספר ללירן שאי אפשר לשלוח את ההודעה כי השני התנתק. מיד לאחר מכן הוא מחזיר אותו לנקודת ההתחלה ומבקש ממנו שם של משתמש אחר.

```
Microsoft Visual Studio X + - □ X
Connection to server established
Welcome to the chat! What is your name?
Liran
Hello Liran! Have fun chatting.
You can quit any time by typing 'QUIT'.
-----
Who do you want to message?
Maor
Connection created
hey maor
The other party has disconnected.
-----
Who do you want to message?
maor
User Was Not Found.
-----
Who do you want to message?
Quit

Microsoft Visual Studio Debug Console X + - □ X
Connection to server established
Welcome to the chat! What is your name?
Maor
Hello Maor! Have fun chatting.
You can quit any time by typing 'QUIT'.
-----
Who do you want to message?
quit
C:\Users\liran\Desktop\HIT\??? \????? ??????
code 0 (0x0).
To automatically close the console when debugging stops.
Press any key to close this window . . .
```

**בקשה להתחבר למאור, בשלב זה הוא קיים**

**לירן שולח הודעה למאור, בשלב זה הוא מנותק**

**מאור מתנתק תוך כדי שיחה**

בך השרת מתמודד עם ניתוקים יזומים תוך כדי שיחות.



יש לציין גם כי המשתמשים לא חייבים בהכרח להתחבר זה אל זה בזמנית, כלומר משתמש אחד יכול לקבל הודעות ממשתמש שני ללא תלות בפעולות אחרות. (אסינכרוניות), לדוגמה:

```
C:\Users\liran\Desktop\ X + - □ X
Connection to server established
Welcome to the chat! What is your name?
Liran
Hello Liran! Have fun chatting.
You can quit any time by typing 'QUIT'.
-----
Who do you want to message?
Maor
Connection created
-----
hello
-----
Maor: HI!
|

C:\Users\liran\Desktop\HIT\ X + - □ X
Connection to server established
Welcome to the chat! What is your name?
Maor
Hello Maor! Have fun chatting.
You can quit any time by typing 'QUIT'.
-----
Who do you want to message?
Liran: hello ההודעה של לירן
-----
Who do you want to message?
xxx
User Was Not Found.
-----
Who do you want to message?
la la la
User Was Not Found.
-----
Who do you want to message?
Liran רק בשלב הזה מאור מתחבר
Connection created
-----
HI!
|
```

ננתח את השיחה הראשונה בין לירן למאור ב-Wireshark:

No.	Time	Source Address	Destination Address	Source Port	Destination Port	Protocol	TCP payload	Syn	Acknowledgment	Fin	Push	Sequence Number	Acknowledgment Number	ip version	Time to Live
277	9.771315	192.168.1.15	192.168.1.15	57918	12345	TCP		1	0	0	0	0	0	4	128
278	9.771436	192.168.1.15	192.168.1.15	12345	57918	TCP		1	1	0	0	0	1	4	128
279	9.771475	192.168.1.15	192.168.1.15	57918	12345	TCP		0	1	0	0	1	1	4	128
542	17.205985	192.168.1.15	192.168.1.15	52906	12345	TCP		1	0	0	0	0	0	4	128
543	17.206091	192.168.1.15	192.168.1.15	12345	52906	TCP		1	1	0	0	0	1	4	128
544	17.206126	192.168.1.15	192.168.1.15	52906	12345	TCP		0	1	0	0	1	1	4	128
796	45.658828	192.168.1.15	192.168.1.15	57918	12345	TCP	4c6972616e	0	1	0	1	1	1	4	128
797	45.658868	192.168.1.15	192.168.1.15	12345	57918	TCP		0	1	0	0	1	6	4	128
816	52.423497	192.168.1.15	192.168.1.15	52906	12345	TCP	4d616f72	0	1	0	1	1	1	4	128
817	52.423528	192.168.1.15	192.168.1.15	12345	52906	TCP		0	1	0	0	1	5	4	128
827	63.691099	192.168.1.15	192.168.1.15	57918	12345	TCP	69747a696b	0	1	0	1	6	1	4	128
828	63.691131	192.168.1.15	192.168.1.15	12345	57918	TCP		0	1	0	0	1	11	4	128
829	63.691492	192.168.1.15	192.168.1.15	12345	57918	TCP	533a55736572...	0	1	0	1	1	11	4	128
830	63.691512	192.168.1.15	192.168.1.15	57918	12345	TCP		0	1	0	0	11	15	4	128
831	72.934822	192.168.1.15	192.168.1.15	52906	12345	TCP	4d616f72	0	1	0	1	5	1	4	128
832	72.934855	192.168.1.15	192.168.1.15	12345	52906	TCP		0	1	0	0	1	9	4	128
833	72.935063	192.168.1.15	192.168.1.15	12345	52906	TCP	533a53616d65...	0	1	0	1	1	9	4	128
834	72.935084	192.168.1.15	192.168.1.15	52906	12345	TCP		0	1	0	0	9	11	4	128
874	87.834920	192.168.1.15	192.168.1.15	57918	12345	TCP	7961726f6e	0	1	0	1	11	15	4	128
875	87.834957	192.168.1.15	192.168.1.15	12345	57918	TCP		0	1	0	0	15	16	4	128
876	87.835487	192.168.1.15	192.168.1.15	12345	57918	TCP	533a55736572...	0	1	0	1	15	16	4	128
877	87.835514	192.168.1.15	192.168.1.15	57918	12345	TCP		0	1	0	0	16	29	4	128
894	89.980732	192.168.1.15	192.168.1.15	57918	12345	TCP	4d616f72	0	1	0	1	16	29	4	128
895	89.980763	192.168.1.15	192.168.1.15	12345	57918	TCP		0	1	0	0	29	20	4	128
896	89.981272	192.168.1.15	192.168.1.15	12345	57918	TCP	533a55736572...	0	1	0	1	29	20	4	128
897	89.981302	192.168.1.15	192.168.1.15	57918	12345	TCP		0	1	0	0	20	40	4	128
898	99.156198	192.168.1.15	192.168.1.15	52906	12345	TCP	4c6972616e	0	1	0	1	9	11	4	128
899	99.156240	192.168.1.15	192.168.1.15	12345	52906	TCP		0	1	0	0	11	14	4	128
900	99.156652	192.168.1.15	192.168.1.15	12345	52906	TCP	533a55736572...	0	1	0	1	11	14	4	128
901	99.156682	192.168.1.15	192.168.1.15	52906	12345	TCP		0	1	0	0	14	22	4	128
1014	125.662005	192.168.1.15	192.168.1.15	57918	12345	TCP	486579204d61...	0	1	0	1	20	40	4	128
1015	125.662041	192.168.1.15	192.168.1.15	12345	57918	TCP		0	1	0	0	40	31	4	128
1016	125.662259	192.168.1.15	192.168.1.15	12345	52906	TCP	433a4c697261...	0	1	0	1	22	14	4	128
1017	125.662292	192.168.1.15	192.168.1.15	52906	12345	TCP		0	1	0	0	14	42	4	128
1038	137.925910	192.168.1.15	192.168.1.15	52906	12345	TCP	576861747320...	0	1	0	1	14	42	4	128
1039	137.925961	192.168.1.15	192.168.1.15	12345	52906	TCP		0	1	0	0	42	40	4	128
1040	137.926124	192.168.1.15	192.168.1.15	12345	57918	TCP	433a4d616f72...	0	1	0	1	40	31	4	128
1041	137.926163	192.168.1.15	192.168.1.15	57918	12345	TCP		0	1	0	0	31	74	4	128
1114	145.462912	192.168.1.15	192.168.1.15	57918	12345	TCP	49276d20646f...	0	1	0	1	31	74	4	128
1115	145.463054	192.168.1.15	192.168.1.15	12345	57918	TCP		0	1	0	0	74	46	4	128
1116	145.463245	192.168.1.15	192.168.1.15	12345	52906	TCP	433a4c697261...	0	1	0	1	42	40	4	128
1117	145.463273	192.168.1.15	192.168.1.15	52906	12345	TCP		0	1	0	0	40	66	4	128

1205	166.677188	192.168.1.15	192.168.1.15	57918	12345	TCP	546865726527...	0	1	0	1	46	74	4	128
1206	166.677220	192.168.1.15	192.168.1.15	12345	57918	TCP		0	1	0	0	74	79	4	128
1207	166.677264	192.168.1.15	192.168.1.15	12345	52906	TCP	433a4c697261...	0	1	0	1	66	40	4	128
1208	166.677283	192.168.1.15	192.168.1.15	52906	12345	TCP		0	1	0	0	40	108	4	128
1299	202.731631	192.168.1.15	192.168.1.15	52906	12345	TCP	49206b6e6f77...	0	1	0	1	40	108	4	128
1300	202.731665	192.168.1.15	192.168.1.15	12345	52906	TCP		0	1	0	0	108	80	4	128
1301	202.731747	192.168.1.15	192.168.1.15	12345	57918	TCP	433a4d616f72...	0	1	0	1	74	79	4	128
1302	202.731783	192.168.1.15	192.168.1.15	57918	12345	TCP		0	1	0	0	79	122	4	128
1379	231.741231	192.168.1.15	192.168.1.15	57918	12345	TCP	526967687420...	0	1	0	1	79	122	4	128
1380	231.741277	192.168.1.15	192.168.1.15	12345	57918	TCP		0	1	0	0	122	119	4	128
1381	231.741335	192.168.1.15	192.168.1.15	12345	52906	TCP	433a4c697261...	0	1	0	1	108	80	4	128
1382	231.741363	192.168.1.15	192.168.1.15	52906	12345	TCP		0	1	0	0	80	157	4	128
1464	279.675940	192.168.1.15	192.168.1.15	57918	12345	TCP	446f20796f75...	0	1	0	1	119	122	4	128
1465	279.675974	192.168.1.15	192.168.1.15	12345	57918	TCP		0	1	0	0	122	186	4	128
1466	279.676059	192.168.1.15	192.168.1.15	12345	52906	TCP	433a4c697261...	0	1	0	1	157	80	4	128
1467	279.676083	192.168.1.15	192.168.1.15	52906	12345	TCP		0	1	0	0	80	233	4	128
1822	405.954706	192.168.1.15	192.168.1.15	52906	12345	TCP	594541482046...	0	1	0	1	80	233	4	128
1823	405.954744	192.168.1.15	192.168.1.15	12345	52906	TCP		0	1	0	0	233	93	4	128
1824	405.954825	192.168.1.15	192.168.1.15	12345	57918	TCP	433a4d616f72...	0	1	0	1	122	186	4	128
1825	405.954866	192.168.1.15	192.168.1.15	57918	12345	TCP		0	1	0	0	186	143	4	128
1993	449.486308	192.168.1.15	192.168.1.15	57918	12345	TCP	477265617420...	0	1	0	1	186	143	4	128
1994	449.486336	192.168.1.15	192.168.1.15	12345	57918	TCP		0	1	0	0	143	229	4	128
1995	449.486501	192.168.1.15	192.168.1.15	12345	52906	TCP	433a4c697261...	0	1	0	1	233	93	4	128
1996	449.486526	192.168.1.15	192.168.1.15	52906	12345	TCP		0	1	0	0	93	285	4	128
2094	504.798385	192.168.1.15	192.168.1.15	52906	12345	TCP	49276d206f6e...	0	1	0	1	93	285	4	128
2095	504.798429	192.168.1.15	192.168.1.15	12345	52906	TCP		0	1	0	0	285	121	4	128
2096	504.798503	192.168.1.15	192.168.1.15	12345	57918	TCP	433a4d616f72...	0	1	0	1	143	229	4	128
2097	504.798533	192.168.1.15	192.168.1.15	57918	12345	TCP		0	1	0	0	229	179	4	128
2154	522.033410	192.168.1.15	192.168.1.15	57918	12345	TCP	71756974	0	1	0	1	229	179	4	128
2155	522.033453	192.168.1.15	192.168.1.15	12345	57918	TCP		0	1	0	0	179	233	4	128
2156	522.033573	192.168.1.15	192.168.1.15	12345	57918	TCP	533a71756974	0	1	0	1	179	233	4	128
2157	522.033599	192.168.1.15	192.168.1.15	57918	12345	TCP		0	1	0	0	233	185	4	128
2158	522.033969	192.168.1.15	192.168.1.15	12345	57918	TCP		0	1	1	0	185	233	4	128
2159	522.033992	192.168.1.15	192.168.1.15	57918	12345	TCP		0	1	0	0	233	186	4	128
2160	522.034187	192.168.1.15	192.168.1.15	57918	12345	TCP		0	1	1	0	233	186	4	128
2161	522.034229	192.168.1.15	192.168.1.15	12345	57918	TCP		0	1	0	0	186	234	4	128
2241	533.052152	192.168.1.15	192.168.1.15	52906	12345	TCP	71756974	0	1	0	1	121	285	4	128
2242	533.052188	192.168.1.15	192.168.1.15	12345	52906	TCP		0	1	0	0	285	125	4	128
2243	533.052354	192.168.1.15	192.168.1.15	12345	52906	TCP	533a71756974	0	1	0	1	285	125	4	128
2244	533.052380	192.168.1.15	192.168.1.15	52906	12345	TCP		0	1	0	0	125	291	4	128
2245	533.052682	192.168.1.15	192.168.1.15	12345	52906	TCP		0	1	1	0	291	125	4	128
2246	533.052700	192.168.1.15	192.168.1.15	52906	12345	TCP		0	1	0	0	125	292	4	128
2247	533.052915	192.168.1.15	192.168.1.15	52906	12345	TCP		0	1	1	0	125	292	4	128
2248	533.052956	192.168.1.15	192.168.1.15	12345	52906	TCP		0	1	0	0	292	126	4	128

הדבר הראשון שנראה לעין הוא כמובן סוג התעבורה – TCP בכל הפאקטות. ובאמת זה המצב מכיוון שהגדרנו את השרת ברמת האפליקציה לעבוד עם חיבור TCP.

מזהים גם שכתובת ה-ip זהה בין הלקוחות, במקרה הזה מכיוון ששניהם פועלים על אותו המחשב.

דבר נוסף שרואים הוא בשכבת התעבורה, מספר הפורט 12345 חוזר על עצמו בכל הפאקטות, ולא בכדי מכיוון שזה מספר הפורט שהקצינו לשרת. לכן כל פאקטה שה-source port שלה הוא 12345, משמעות הדבר שהשרת שלח את הפאקטה וכל פאקטה שה-destination port שלה הוא 12345, משמעות הדבר שהשרת קיבל את הפאקטה מלקוח ששלח אותה.

כמו כן מספרי הפורט 57918 וגם 52906 חוזרים על עצמם בכל הפאקטות, לעיתים מספר אחד ולעיתים המספר השני. זה בגלל שאלו מספרי הפורט מהם מתקשרים המשתמשים לירן ומאור ולכן הם קבועים וחוזרים על עצמם.

מספר הפורט 57918 שייך ללירן ומספר הפורט 52906 שייך למאור, נסביר בהמשך מדוע.

נשים לב לחיבור של המשתמשים "לירן" ו"מאור" לשרת. בחבילות הראשונות נוכל בבירור לראות את ה-3 way handshake של כל אחד מהמשתמשים עם השרת. נראה את ההתחברות של לירן מפורט 57918 לשרת:

No.	Time	Source Address	Destination Address	Source Port	Destination Port	Protocol	TCP payload	Syn	Acknowledgment
277	9.771315	192.168.1.15	192.168.1.15	57918	12345	TCP		1	0
278	9.771436	192.168.1.15	192.168.1.15	12345	57918	TCP		1	1
279	9.771475	192.168.1.15	192.168.1.15	57918	12345	TCP		0	1

בחבילה מספר 277 קורה syn=1, ack=0, destination port = 12345, כלומר זו חבילה שהשרת קיבל מהמשתמש "לירן" והיא מהווה בקשה של הלקוח להתחבר לשרת.

בחבילה מספר 278 קורה syn=1, ack=1, source port = 12345, כלומר זו חבילה שהשרת שלח למשתמש "לירן" והיא מהווה אישור לבקשת החיבור של הלקוח

בחבילה מספר 279 קורה syn=0, ack=1, destination port = 12345, כלומר זו חבילה שהשרת קיבל מהמשתמש "לירן" שמהווה את אישור התחברות הלקוח לשרת, תהליך 3 way handshake הסתיים ומכאן והילך כל הפאקטות ששולח המשתמש "לירן" הן מהצורה הזו.

נשים לב שבפאקטות 542,543,544 קורה אותו התהליך בדיוק עבור המשתמש "מאור" מפורט 52906 שגם הוא התחבר לשרת.

542	17.205985	192.168.1.15	192.168.1.15	52906	12345	TCP		1	0
543	17.206091	192.168.1.15	192.168.1.15	12345	52906	TCP		1	1
544	17.206126	192.168.1.15	192.168.1.15	52906	12345	TCP		0	1

שאלה שנשאלת היא כיצד ידעתי לשייך את הפורט 57918 דווקא לחיבור של "לירן" ואת הפורט 52906 דווקא לחיבור של "מאור"? הרי הסדר יכול להיות הפוך? תשובה אחת לשאלה היא שאני, בתור מי שהריץ את הדיאלוג מן הסתם יודע מי התחבר קודם...

אבל זו לא התשובה שאנחנו מחפשים; התשובה האמיתית היא, נשים לב לפאקטה מספר 796:

796	45.658828	192.168.1.15	192.168.1.15	57918	12345	TCP	4c6972616e	0
0000	02 00 00 00 45 00 00 2d	ff 4f 40 00 80 06 00 00	.....E...@.....					
0010	c0 a8 01 0f c0 a8 01 0f	e2 3e 30 39 33 6c b0 94	.....>0931..					
0020	34 3f cd 3d 50 18 00 ff	06 99 00 00 4c 69 72 61	4?..=P.....Lira					
0030	6e		n					

שמים לב שהפאקטה נושאת את ה-payload: "Liran", זו ההודעה הראשונה לגמרי שנשלחת לשרת, וזה כי הדבר הראשון שמשתמש עושה כשהוא פותח את האפליקציה זה להקליד את השם שלו לשרת. נשים לב שפורט השולח הוא 57918 והמקבל הוא כמובן השרת. אם כן כעת אנחנו יכולים בוודאות לשייך את מספר הפורט 57918 למשתמש "לירן" ואילו את מספר הפורט האחר 52906 לשייך למשתמש "מאור". ובאמת לפי פאקטה מספר 816 רואים שהפורט 52906 שייך ל"מאור" לפי אותו ניתוח:

816	52.423497	192.168.1.15	192.168.1.15	52906	12345	TCP	4d616f72	0
0000	02 00 00 00 45 00 00 2c	ff 51 40 00 80 06 00 00	.....E...Q@.....					
0010	c0 a8 01 0f c0 a8 01 0f	ce aa 30 39 3c 9d 6c cc	.....09<.1..					
0020	8d 0d 6b 6d 50 18 00 ff	cd be 00 00 4d 61 6f 72	..kmP.....Maor					

באותו אופן אפשר לבחון את ניתוק המשתמשים מהשרת, נשים לב לפאקטות 2158-2161. נזכיר שבתמונה הבאה, הביט הימני הוא השדה Fin, משמאלו Ack ומשמאלו Syn שמאופס בכל השורות.

2158	522.033969	192.168.1.15	192.168.1.15	12345	57918	TCP	0	1	1
2159	522.033992	192.168.1.15	192.168.1.15	57918	12345	TCP	0	1	0
2160	522.034187	192.168.1.15	192.168.1.15	57918	12345	TCP	0	1	1
2161	522.034229	192.168.1.15	192.168.1.15	12345	57918	TCP	0	1	0

אלו הפאקטות האחרונות שהוקלטו עבור המשתמש "לירן" שבפורט 57918. נשים לב לתהליך 4 way handshake עבור הניתוק של לירן מהשרת: בחבילה מספר 2158 קורה  $fin=1$ ,  $ack=1$  וזו חבילה שהשרת שולח ללירן. כאן השרת שולח בקשה לניתוק החיבור. (כתגובה לפאקטה קודמת שבה לירן כמובן שלח הודעת quit לשרת). בחבילה מספר 2159 קורה  $fin=0$ ,  $ack=1$  וזו חבילה ששולח לירן לשרת. כאן לירן מאשר את בקשת השרת לניתוק החיבור. בחבילה מספר 2160 קורה  $fin=1$ ,  $ack=1$  וזו חבילה ששולח לירן לשרת. כאן לירן שולח בקשה לניתוק החיבור. בחבילה מספר 2161 קורה  $fin=0$ ,  $ack=1$  וזו חבילה ששולח השרת ללירן. כאן השרת מאשר את הבקשה של לירן והחיבור נסגר סופית. זו הפאקטה האחרונה שעברה בין לירן לשרת או להיפך. התהליך חוזר על עצמו עבור המשתמש "מאור" שבפורט 52906 ובאמת 4 הפאקטות האחרונות של ההקלטה הן הניתוק של מאור מהשרת:

1921	244.105287	12345	63992	TCP	0	1	1
1922	244.105308	63992	12345	TCP	0	1	0
1923	244.107007	63992	12345	TCP	0	1	1
1924	244.107064	12345	63992	TCP	0	1	0

בשכבת הרשת, כאמור כתובת המקור וכתובת היעד זהות, מכיוון שהלקוח והשרת רצים על אותו מחשב.

יש לציין שבשונה מהרצה על כתובת ה-loopback, השתמשנו בכתובת IP מקומית אמיתית של המחשב.

כתוצאה מכך הפאקטות נארזות בכותרת IP וניתן לראות את האריזה של הפאקטות ולנתח אותן ב-Wireshark.

את השרת הגדרנו בכתובת 0.0.0.0, מה שמאפשר לו להאזין לכל כתובות ה-IP של המחשב ולקבל לקוחות לפי כתובת ה-IP שלהם.

הכתובת 0.0.0.0 היא כתובת לוגית לצורך binding שאינה נשלחת ברשת, ולכן היא לא מופיעה בתור כתובת ה-IP האמיתית של השרת בקובץ הלכידה.

כותרות ה-IPv4 של כל הפאקטות הן מהצורה:

```

▼ Internet Protocol Version 4, Src: 192.168.1.15, Dst: 192.168.1.15
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 40
    Identification: 0xff54 (65364)
  ▼ 010. .... = Flags: 0x2, Don't fragment
    0... .... = Reserved bit: Not set
    .1.. .... = Don't fragment: Set
    ..0. .... = More fragments: Not set
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 128
    Protocol: TCP (6)
    Header Checksum: 0x0000 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 192.168.1.15
    Destination Address: 192.168.1.15
    [Stream index: 1]

```

עם שינויים מסוימים בשדות ה-Header Length, Total Length בהתאם לגודל הסגמנט וההודעה ובשדה ה-Identification. נשים לב לסוג הפרוטוקול, 6 עבור חיבור TCP. נשים לב גם לשדה

Time to Live, שדה זה תמיד עם ערך 128, ערך שהוא קבוע וגדול יחסית.

הסיבה לכך היא שחיבור הרשת הוא בין המחשב לעצמו (כתובות IP זהות). כלומר אין ראוטרס או רכיבי רשת אחרים בדרך ולכן הפאקטות עם ערך קבוע וגדול שקובעת מערכת ההפעלה.



מקודם צוינה הפאקטה 796 שנשאה את ה-payload: "Liran".  
 כמובן שבקובץ הלכידה שמצורף אפשר לעיין בכל ההודעות ברמת האפליקציה, כמו:  
 לירן שולח לשרת "Itzik" בפאקטה 827 (משתמש אליו רוצה לשלוח הודעה)

No.	Time	Source Address	Destination Address	Source Port	Destination Port	Protocol	TCP payload
277	9.771315	192.168.1.15	192.168.1.15	57918	12345	TCP	
278	9.771436	192.168.1.15	192.168.1.15	12345	57918	TCP	
279	9.771475	192.168.1.15	192.168.1.15	57918	12345	TCP	
542	17.205985	192.168.1.15	192.168.1.15	52906	12345	TCP	
543	17.206091	192.168.1.15	192.168.1.15	12345	52906	TCP	
544	17.206126	192.168.1.15	192.168.1.15	52906	12345	TCP	
796	45.658828	192.168.1.15	192.168.1.15	57918	12345	TCP	4c6972616e
797	45.658868	192.168.1.15	192.168.1.15	12345	57918	TCP	
816	52.423497	192.168.1.15	192.168.1.15	52906	12345	TCP	4d616f72
817	52.423528	192.168.1.15	192.168.1.15	12345	52906	TCP	
827	63.691099	192.168.1.15	192.168.1.15	57918	12345	TCP	69747a696b
828	63.691131	192.168.1.15	192.168.1.15	12345	57918	TCP	

Frame 827: Packet, 49	0000	02 00 00 00 45 00 00 2d	ff 53 40 00 80 06 00 00	.....E...S@.....
Null/Loopback	0010	c0 a8 01 0f c0 a8 01 0f	e2 3e 30 39 33 6c b0 99	.....>0931..
Internet Protocol Version 4	0020	34 3f cd 3d 50 18 00 ff	e4 80 00 00 69 74 7a 69	4?..P...itzi
TCP	0030	6b		k

השרת משיב ללירן: S:UserNotFound בפאקטה 829 כי איציק הוא לא משתמש קיים

No.	Time	Source Address	Destination Address	Source Port	Destination Port	Protocol	TCP payload
542	17.205985	192.168.1.15	192.168.1.15	52906	12345	TCP	
543	17.206091	192.168.1.15	192.168.1.15	12345	52906	TCP	
544	17.206126	192.168.1.15	192.168.1.15	52906	12345	TCP	
796	45.658828	192.168.1.15	192.168.1.15	57918	12345	TCP	4c6972616e
797	45.658868	192.168.1.15	192.168.1.15	12345	57918	TCP	
816	52.423497	192.168.1.15	192.168.1.15	52906	12345	TCP	4d616f72
817	52.423528	192.168.1.15	192.168.1.15	12345	52906	TCP	
827	63.691099	192.168.1.15	192.168.1.15	57918	12345	TCP	69747a696b
828	63.691131	192.168.1.15	192.168.1.15	12345	57918	TCP	
829	63.691492	192.168.1.15	192.168.1.15	12345	57918	TCP	533a55736572...
830	63.691512	192.168.1.15	192.168.1.15	57918	12345	TCP	

Frame 829: Packet, 58	0000	02 00 00 00 45 00 00 36	ff 55 40 00 80 06 00 00	.....E..6..U@.....
Null/Loopback	0010	c0 a8 01 0f c0 a8 01 0f	30 39 e2 3e 34 3f cd 3d	.....09>4?..=
Internet Protocol Version 4	0020	33 6c b0 9e 50 18 00 ff	84 a1 00 00 53 3a 55 73	31..P...S:Us
TCP	0030	65 72 4e 6f 74 46 6f 75	6e 64	erNotFou nd

הודעה שלירן שלח למאור דרך השרת בפאקטה 1464: "Wanna come over? We can order pizza"

No.	Time	Source Address	Destination Address	Source Port	Destination Port	Protocol	TCP payload
1301	202.731747	192.168.1.15	192.168.1.15	12345	57918	TCP	433a4d616f72...
1302	202.731783	192.168.1.15	192.168.1.15	57918	12345	TCP	
1379	231.741231	192.168.1.15	192.168.1.15	57918	12345	TCP	526967687420...
1380	231.741277	192.168.1.15	192.168.1.15	12345	57918	TCP	
1381	231.741335	192.168.1.15	192.168.1.15	12345	52906	TCP	433a4c697261...
1382	231.741363	192.168.1.15	192.168.1.15	52906	12345	TCP	
1464	279.675940	192.168.1.15	192.168.1.15	57918	12345	TCP	446f20796f75...
1465	279.675974	192.168.1.15	192.168.1.15	12345	57918	TCP	

Frame 1464: Packet, 111	0000	02 00 00 00 45 00 00 6b	ff 7f 40 00 80 06 00 00	.....E..K..@.....
Null/Loopback	0010	c0 a8 01 0f c0 a8 01 0f	e2 3e 30 39 33 6c b1 0a	.....>0931..
Internet Protocol Version 4	0020	34 3f cd b6 50 18 00 ff	87 41 00 00 44 6f 20 79	4?..P...A..Do y
TCP	0030	6f 75 20 77 61 6e 6e 61	20 63 6f 6d 65 20 6f 76	ou wanna come ov
	0040	65 72 20 61 6e 64 20 77	61 74 63 68 20 69 74 20	er and w atch it
	0050	74 6f 67 65 74 68 65 72	3f 20 57 65 20 63 61 6e	together ? We can
	0060	20 6f 72 64 65 72 20 70	69 7a 7a 61 20 3b 29	order p izza ;)

מלבד הדוגמאות האלו ניתן למצוא את כל שאר ההודעות והפאקטות בקובץ הלכידה המצורף.

נציג דוגמה לשיחה בין חמישה(!) משתמשים. נציג בהתחלה את פלט השרת שמסכם את השיחה. הודעות בין משתמשים מסומנות בקו אדום, הפלט נראה כך:

[illegible]



```
Connection to server established
Welcome to the chat! What is your name?
Liran
Hello Liran! Have fun chatting.
You can quit any time by typing 'QUIT'.
```

Who do you want to message?

Osher

Connection created

Hey, want to go see a movie tonight?

Osher: I was just talking with Eden about it, join us

ok I can drive us

Osher: ok I'll tell eden

[back](#)

Who do you want to message?

Noa

Connection created

WANNA COME WITH US?

[illegible]

QUIT

הפלט של המשתמש "Osher":

[illegible]

```
Connection to server established
Welcome to the chat! What is your name?
Eden
Hello Eden! Have fun chatting.
You can quit any time by typing 'QUIT'.
```

Who do you want to message?  
Osher: Want to go to the cinema later?

```
Who do you want to message?  
Gal  
Connection created
```

Are you free for a movie tonight?

Gal: Yeah sure

Osher: Liran can pick us up

[back](#)

```
Who do you want to message?  
Osher  
Connection created
```

Ok, also Gal comes too

[back](#)

```
Who do you want to message?
Liran
Connection created
```

[back](#)

```
Who do you want to message?  
Noa  
Connection created
```

WANNA COME WITH US?

[illegible]

quit

הפלט של המשתמש "Gal":

```
Connection to server established
Welcome to the chat! What is your name?
Gal
Hello Gal! Have fun chatting.
You can quit any time by typing 'QUIT'.
```

Who do you want to message?  
Eden: Are you free for a movie tonight?

```
Who do you want to message?  
Eden  
Connection created
```

Yeah sure

[back](#)

```
Who do you want to message?  
Noa  
Connection created
```

WANNA COME WITH US?

[illegible]

quit

[illegible]