

---

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 ..... 5
- 4. ניתוח התעבורה באמצעות WIRESHARK ..... 8

## חלק שני ..... 13

- 5. מבוא ..... 13
- 6. ארכיטקטורת הפרויקט ..... 14
  - 6.1 Server\_interface – אפליקציית השרת ..... 15
  - 6.2 Client\_interface – אפליקציית הלקוח ..... 20
- 7. הוראות התקנה והרצה ..... 24
- 8. דוגמאות קלט ופלט ..... 31

## ביבליוגרפיה ומקורות ..... 47

# חלק ראשון

## אריזת נתונים ולכידת מנות ב-Wireshark

### מבוא

בחלק הראשון של הפרויקט נדרשנו להכין קובץ CSV עם עמודות שונות לניתוח בסיסי של מידע. נשתמש בקובץ בתוך מחברת ג'ופיטר כדי לייצר תעבורה. את התעבורה נלכוד באמצעות Wireshark כדי לנתח את התעבורה. בחרתי לייבא הודעות של פרוטוקול DNS (Domain Name System).

### יצירת קובץ ה-CSV

בקובץ ה-CSV שמצורף לדו"ח (liran\_dns\_input.csv) מופיעים שדות (עמודות) שמלמדות על הפאקטות, בין היתר כללתי: Request/Reply, גודל הפאקטה (פריים) וכו'.

כמובן שהקובץ מכיל גם את השדות להם נדרשנו בתיאור הפרויקט:

- msg\_id - מספר סידורי המייחד כל הודעה
- app\_protocol - פרוטוקול ברמת האפליקציה אשר בחרנו, במקרה שלנו 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 באמצעות לכידת תעבורה בתוכנת Wireshark (אפשר למשל למצוא בקובץ dns\_input\_capture.pcap או בקובץ ה-CSV הודעות משרתי ה-DNS של Google ושל Discord, תוכנות שרצו על המחשב שלי בעת ההקלטה).

את השדה message – הרכבתי על ידי שרשור שדות:

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

כל אלו הם מאפיינים של 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

ייצוא הפאקטות אל קובץ CSV



## תיאור והסבר אריזת הפאקטות

השלב השני בחלק זה הוא טעינת קובץ ה-CSV אל מחברת הג'ופיטר שמצורפת.

המטרה: להשתמש בכימוס (Encapsulation) במודל OSI כדי לארוז את ההודעות לפי השכבות ולהעבירן. האריזות מרכיבות את הפאקטה מהשכבה העליונה (האפליקציה) עד לשכבות התחתונות. נתאר באופן כללי את תהליך האריזה ולאחר מכן נפרט.

תחילה ההודעה נוצרת בשכבת האפליקציה (מחברת הג'ופיטר טוענת את ההודעה שתישלח - payload) ולאחר מכן מועברת אל שכבת התעבורה דרך socket. במחברת - שולחים את הודעות ה-CSV שלנו או הודעות דמה. כלומר המחברת שולחת כל payload משורות ה-CSV אל שכבת התעבורה.

שכבת התעבורה (פרוטוקול TCP) עוטפת את ה-payload כ-Segment שיפוענח על ידי שכבת התעבורה בצד המקבל. בשכבת התעבורה משייכים לפאקטה כותרת TCP Header שמכילה:

source port, destination port, flags ושלוש שדות נוספים שמשמשים לשמירה על חיבור רציף (מספרי seq, ack), בקרת זרימה (window) ועוד. TCP הוא פרוטוקול תעבורה שמעביר מידע בצורה אמינה וקפדנית. בין הערכים נוסף שדה Checksum שמשמש את הצד המקבל לפענוח שגיאות. לאחר מכן שכבת התעבורה מעבירה את הסגמנט לשכבת הרשת.

שכבת הרשת עוטפת את הסגמנט כ-IP Datagram. כלומר היא מוסיפה כותרת IP Header שמכילה: source IP, destination IP, version, TTL ושדות נוספים. ה-Datagram מסוגל לעבור בין רשתות שונות ולהיות מנותבת דרך רכיבי רשת (נתבים) עד ליעד. שכבת הרשת מעבירה את ה-Datagram לשכבת הקו.

רעיונית - שכבת הקו (הלינק) מוסיפה מסגרת Frame והיא השכבה האחרונה שעוטפת את ה-Datagram כדי להפוך אותה לפאקטה שתשודר פיזית (למשל דרך Ethernet, גלי רדיו, חוטי נחושת וכו'). היא מציינת כתובות MAC (מקור ויעד) ומעבירה את הפאקטה לשכבה הפיזית, כלומר משדרת אותה מהמכשיר החוצה.

בפועל, במקרה שלנו התהליכים שהמחברת עושה הם לוקאליים ולכן כשננתח את הפאקטות לא נראה כתובות MAC אמיתיות.

תפקידה העיקרי של שכבה זו בהקשר של המחברת הוא לשדר את הפאקטות אל לולאת ה-loopback, כלומר מקומית בתוך המחשב.

ניתן לראות איך המחברת מממשת את האריזה לסגמנט בפרוטוקול TCP בהתאם לפורמט:

## TCP Header Format

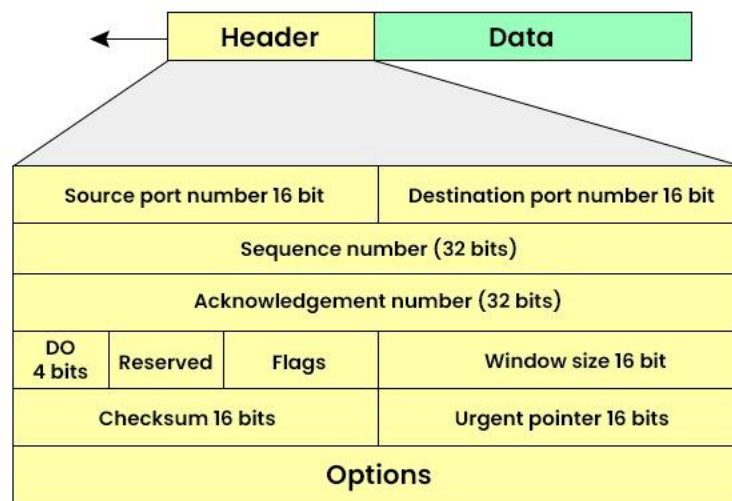


Image from [pynetlabs.com](https://pynetlabs.com)

```
def build_tcp_header(src_ip: str, dst_ip: str, src_port: int, dst_port: int, payload: bytes=b'',
                    seq: Optional[int]=None, ack_seq: int=0, flags: int=0x02, window: int=65535) -> bytes:
    if seq is None: # if sequence number is not initialized
        seq = random.randint(0, 0xFFFFFFFF)
    doff_reserved = (5 << 4)
    checksum_tcp = 0 # initialize checksum
    urg_ptr = 0 # We don't have urgent data
    tcp_header = struct.pack('!HHLLBBHH', # pack an initial header before evaluating checksum
                             src_port, dst_port, seq, ack_seq,
                             doff_reserved, flags, window,
                             checksum_tcp, urg_ptr)

    placeholder = 0
    protocol = socket.IPPROTO_TCP
    tcp_length = len(tcp_header) + len(payload)
    pseudo_header = struct.pack('!4s4sBBH', # pseudo_header is only used to calculate checksum, using some fields included in IP header
                                socket.inet_aton(src_ip), socket.inet_aton(dst_ip),
                                placeholder, protocol, tcp_length)
    chksum = checksum(pseudo_header + tcp_header + payload) # calculate checksum
    tcp_header = struct.pack('!HHLLBBH H H', # build the complete TCP header containing all fields
                             src_port, dst_port, seq, ack_seq,
                             doff_reserved, flags, window,
                             chksum, urg_ptr)

    return tcp_header
```

ניתן לראות איך המחברת מממשת את אריזת שכבת הרשת בפורמט IPv4 בהתאם לשדות:

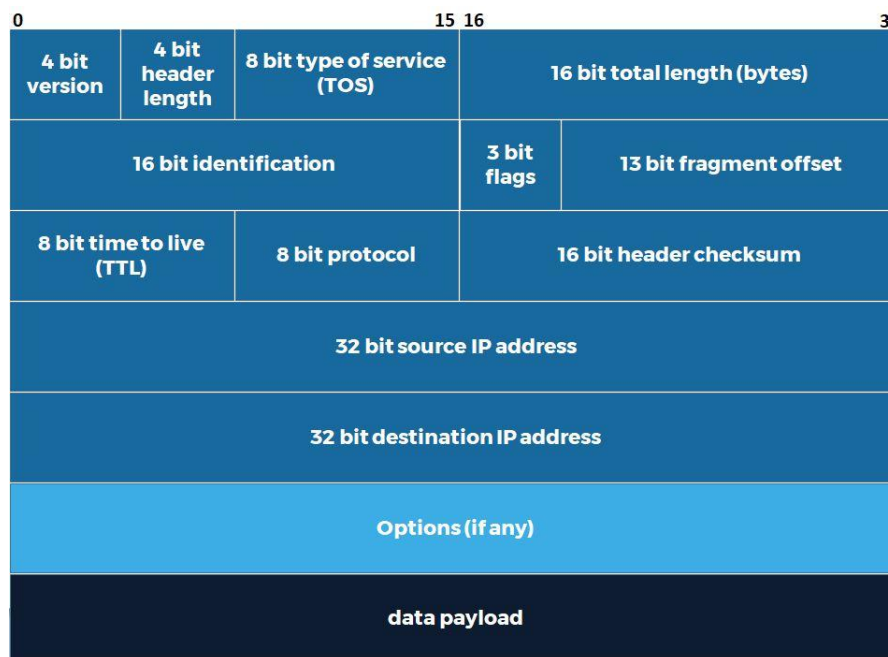


Image from [uninets.com](http://uninets.com)

```
def build_ip_header(src_ip: str, dst_ip: str, payload_len: int, proto: int=socket.IPPROTO_TCP) -> bytes:
    version_ihl = (4 << 4) + 5 # Version = 4, Header length = 5 words (= 20 bytes)
    tos = 0 # Type of service, not used
    total_length = 20 + payload_len # Total length = IP header (20 bytes) + payload length
    identification = random.randint(0, 65535) # This field is used for fragmentation
    flags_fragment = 0 # don't use fragmentation
    ttl = 64 # The Maximum amount of hops the packet can traverse
    header_checksum = 0 # initialize checksum
    src = socket.inet_aton(src_ip)
    dst = socket.inet_aton(dst_ip)
    ip_header = struct.pack('!BBHHBHH4s4s', # pack an initial header before evaluating checksum
                             version_ihl, tos, total_length, identification,
                             flags_fragment, ttl, proto, header_checksum,
                             src, dst)

    chksum = checksum(ip_header) # calculate checksum
    ip_header = struct.pack('!BBHHBHH4s4s', # build the complete IP header containing all fields
                             version_ihl, tos, total_length, identification,
                             flags_fragment, ttl, proto, chksum,
                             src, dst)

    return ip_header
```

בכך השלמנו את האריזה של שלושת השכבות, אפליקציה (הודעות ה-CSV שהמחברת טוענת ושולחת), התעבורה (הסגמנט, tcp\_header) והרשת (ה-IP Datagram). כל הודעה שנשלח מהמחברת תיעטף בשכבות ותועבר.

## ניתוח התעבורה באמצעות WIRESHARK

בשלב הראשון ננתח תעבורה 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 הפקודות:

```
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)
```

בתמונות שבהמשך, מספר הפורט 25215 הוא פורט שהוקצה למשתמש (אני, שהריץ את Wireshark) ומספר הפורט 12345 הוא מספר הפורט של האפליקציה (מחברת ה-Jupyter)

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

זו הגדרה ל-TCP segment שבו רק הדגל SYN פועל. במילים אחרות זו הפאקטה הראשונה בתהליך 3-way-handshake.

אבל נשים לב שלא נוצר חיבור TCP מלא, כלומר לא חזרה פאקטת syn+ack, כי במחברת שולחים פאקטות גולמיות, listen, bind, accept, לא כלולים בקוד. במילים אחרות, ברמת האפליקציה אין ביסוס של תקשורת TCP אלא שליחה של פאקטות גולמיות בין כתובת פורט כלשהי במחשב ל-loopback.

לכן נצפה שלאחר הפאקטה הזו, תישלח פאקטה שתבשר על Reset, כי אין אפשרות להתחיל חיבור. ובאמת ניתן לראות שלאחר כל פאקטה כזו שנשלחת למחברת (אלו שמכילות TCP payload שהן Hello Packet 0 או Hello Packet 1 או Hello Packet 2 בשדה ה-TCP payload) מקבלים 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	

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



• demo\_send(num\_packets=3, flags=0x18)

זו הגדרה לפאקטת TCP שבה רק הדגלים ack+push פועלים. כמו מקודם אין באמת תקשורת TCP בין הצדדים ולכן נצפה לקבל חזרה Reset=1. הפעם נשים לב שנקבל 0.ack. 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	1	0	0x0004	
20325	25215	12345	2117.214729	TCP	0	1	1	0	0	0x0018	48656c6c6f205061636b65742031
20326	12345	25215	2117.214837	TCP	0	0	0	1	0	0x0004	
20331	25215	12345	2118.217473	TCP	0	1	1	0	0	0x0018	48656c6c6f205061636b65742032
20332	12345	25215	2118.217552	TCP	0	0	0	1	0	0x0004	

שאר הפקודות מתנהגות באופן דומה למעט האחרונה:

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

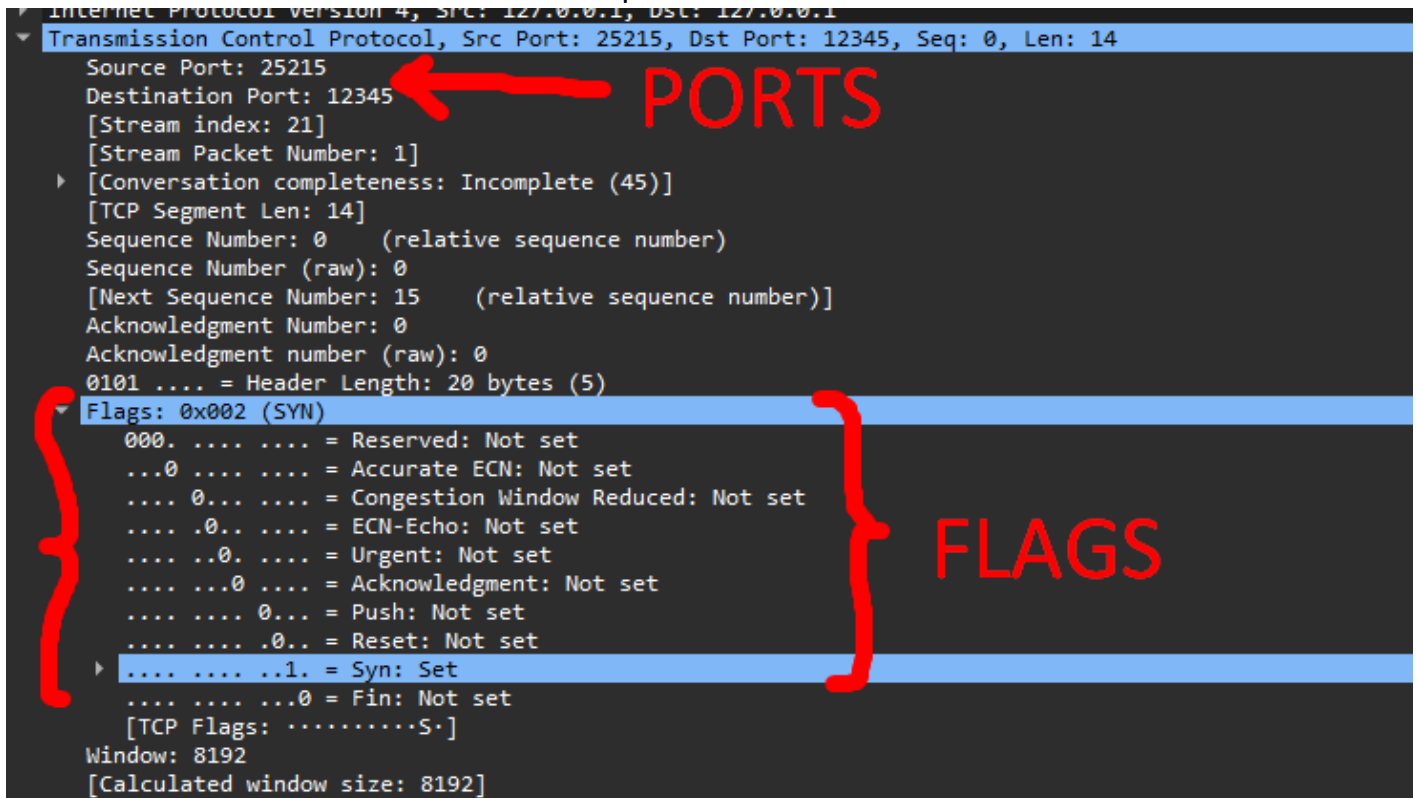
זו פאקטה שבה רק הדגל Reset פועל. כשפאקטה זו נשלחת אפילו לא נקבל פאקטות כתגובה מהמחברת. הסיבה היא שפאקטה מהסוג הזה לא מצפה לתגובה בחזרה אלא מבקשת מהמחברת לסגור את החיבור הקיים. מכיוון שלמעשה אין חיבור קיים בכל מקרה, לא נקבל תגובה מהמחברת.

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 (rela	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 ה-TCP סגמנט של הפאקטות כפי שמיוצגות ב-WIRESHARK.



ניתן לראות גם את ה-TCP Payload (ההודעה) של הפאקטות (בתמונה – "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 שמכילות את הודעות ה-CSV עם הדגלים ack=1,push=1.

אכן ניתן לראות שכל הפאקטות שמתקבלות בעלות TCP סגמנט עם אותם דגלים (ack+push):

```

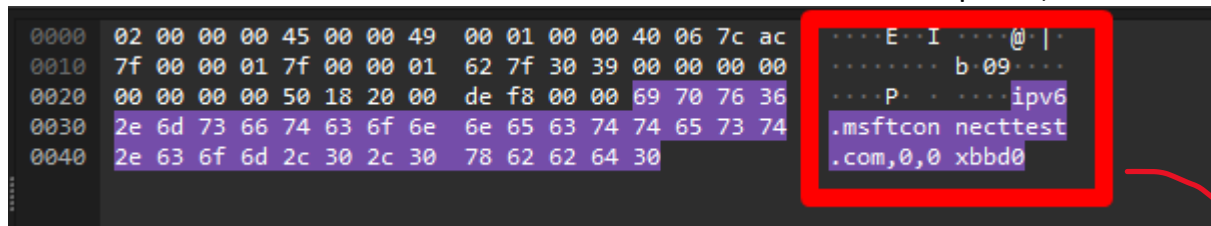
[Stream index: 0]
Transmission Control Protocol, Src Port: 25215, Dst Port: 12345, Seq: 1, Ack: 1, Len: 32
Source Port: 25215
Destination Port: 12345
[Stream index: 0]
[Stream Packet Number: 17]
[Conversation completeness: Incomplete (40)]
[TCP Segment Len: 32]
Sequence Number: 1 (relative sequence number)
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 הליכידה (מצורף בגיטהאב: jupyter\_send\_csv\_capture.pcap) 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

וגם ניתן לראות שכל פאקטה ששולחים נושאת את שדה ה-message של השורה המתאימה בקובץ ה-CSV. למשל, הפאקטה הראשונה נושאת את ההודעה:



נחזור לקובץ ה-CSV (מצורף בגיטהאב: liran\_dns\_input.csv) ונשים לב שגם השורה הראשונה בקובץ נושאת את שדה ה-message:

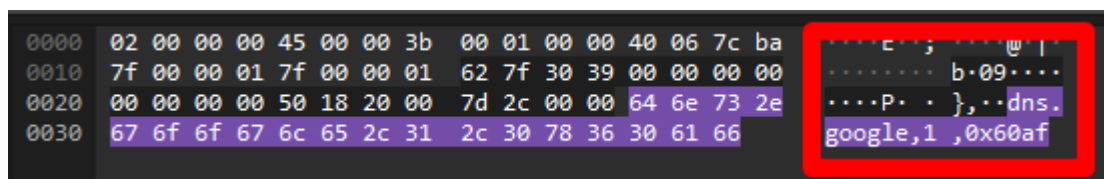
message
ipv6.msftconnecttest.com,0,0xbbd0

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 (jupyter\_send\_csv\_capture.pcap), פאקטה מספר 63 נושאת את ה-Payload:



נחזור לקובץ ה-CSV ונראה ששורה מספר 32 נושאת את שדה ה-message:

AAAA	70	dns.google,0,0x60af	0.00021	1.1.1.1	192.168.1.	53	59365	DNS	30
A	70	dns.google,0,0x0957	0.00014	1.1.1.1	192.168.1.	53	64789	DNS	31
AAAA	130	dns.google,1,0x60af	0.026362	192.168.1.	1.1.1.1	59365	53	DNS	32
HTTPS	150	dns.google,1,0x41f8	0.001168	192.168.1.	1.1.1.1	58331	53	DNS	33
A	106	dns.google,1,0x0957	0.008372	192.168.1.	1.1.1.1	64789	53	DNS	34
HTTPS	90	dns.google,0,0x4261	2.948365			53	56432	DNS	35

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 ולשייך כל הודעה בקובץ ה-CSV גם בקובץ הלכידה.

## חלק שני

# פיתוח יישום רשת לקוח/שרת בפרוטוקול TCP

### מבוא

נדרשנו לכתוב פרויקט המיישם צ'אט רב משתמשים שבו קיימים לקוחות שמתקשרים אחד עם השני דרך שרת שמנהל את החלפת המסרים ביניהם.

**יש לציין:** דרישות הפרויקט מבקשות מהמשתמש לציין לקוח אליו רוצה לפנות כדי לשלוח אליו טקסט: "כאשר השרת יקבל פניה לקוח השרת צריך לפתוח צ'אט בין הלקוח ללקוח אחר. הפניה צריכה להכיל את שם הלקוח המבוקש. ניתן להניח שלכל לקוח יש שם ייחודי."

כלומר, משתמש יכול לשלוח הודעה רק למשתמש אחד שהוא מציין במפורש את שמו. ניתן היה לשנות את השרת בדרכים אחרות כך שיעביר למשל הודעות לכל המשתמשים, או יעביר רק ללקוחות מסוימים. ניתן להפוך את הצ'אט למשל לקבוצתי, "כולם עם כולם". הנקודה היא שהצ'אט עלול להיראות שונה בהתאם להעדפה. (צ'אט "אחד על אחד", צ'אט "כולם על כולם", וכו'). כאן הדרישה באה לידי ביטוי - הפרויקט שיתואר מכאן והילך, השרת והאפליקציות שיתוארו בהמשך עוקבות בדיוק אחרי הדרישה - להעביר הודעות מלקוח אחד רק ללקוח אחר עם שם שיינתן מראש.

לדו"ח זה מצורפים שתי אפליקציות שמרכיבות את הפרויקט, האחת היא עבור הלקוח ששולח בקשות לשרת ומקבל מסרים מהשרת. השנייה היא עבור השרת. השרת מקבל קלטים ממספר לקוחות ויודע לנהל אותם, יודע להעביר הודעות מלקוח פונה א' אל לקוח נמען ב' ומתקשר הוא עצמו עם הלקוחות.

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

בנוסף לדו"ח מצורפים האפליקציות בתיקיות:

- Server\_interface
- Client\_interface

server\_interface היא אפליקציית השרת, client\_interface היא אפליקציית הלקוח. כדי להריץ שיחה בין לקוחות נצטרך להפעיל מספר מופעים של client\_interface (נפתח את האפליקציה כמה פעמים בחלונות שונים של ויזואל סטודיו). כל קבצי הקוד בשתי האפליקציות server\_interface, client\_interface מלווים בתיעודים ובהסברים.

## ארכיטקטורת הפרויקט

כדי ליצור תקשורת בין יישומים בשפת C++ אנחנו משתמשים בספריית WinSock2 שמאפשרת לנו גישה לכל פעולות ה-Sockets להן נידרש כדי ליצור את התקשורת. כאשר מדובר בתקשורת מבוססת sockets אנחנו נדרשים לביצוע תהליכים טכניים באפליקציות לפני שנתחיל להעביר הודעות. התהליכים משתנים במעט בין הלקוח לשרת, ואלו הם:

עבור הלקוח נבצע:

- אתחול ספריית Winsock
- יצירת socket של הלקוח (פונקציית יצירת socket)
- ציון כתובת השרת ויצירת חיבור TCP 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 השרת שיאזין ויתקשר עם לקוחות (פונקציית socket)
- קשירת ה-socket המאזין אל הכתובת הלוגית 0.0.0.0, 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 ברשימה שתפורט בהמשך (פונקציית accept)
- לאחר קבלת לקוח נתקשר איתו באמצעות קבלה ושליחת הודעות (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. לכן גם השרת מטפל בכל לקוח 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

נפרט על אפליקציית השרת ולאחר מכן על אפליקציית הלקוח.

## Server interface – אפליקציית השרת

תיקיה זו מורכבת מ-4 קבצי header שם מרוכזים הפונקציות עם תיעודים כלליים ומ-4 קבצי cpp גם הם עם תיעודים מפורטים. פירוט רחב נמצא בתיעוד שבפרויקט, כאן נתאר את ה-headers:

### – client\_info.h

בקובץ זה מוגדר class עם פרטים על הלקוח. אנחנו שומרים את השם של הלקוח (השם ייחודי לכל לקוח – דרישות הפרויקט אומרות 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.

נגדיר גם פונקציה למחיקת הלקוח מהרשימה (במקרה של יציאה מהצ'אט).

ונגדיר פונקציה לקבלת הודעות מהשרת, כלומר השרת יפנה ל-Socket השמור של הלקוח באמצעות הפונקציה הזו.

```
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 class ClientInfo {
12 public:
13     string name;
14     SOCKET socket;
15
16 public:
17     ClientInfo();
18     ClientInfo(string& name, SOCKET socket);
19     bool operator==(const ClientInfo&);
20
21     // A method to find a specific client in a list, We are given that all client names are unique
22     // This is used when a client specifies the name of the target they wish to message.
23     static ClientInfo* findByName(const string&, list<ClientInfo>&);
24
25     // Given a message, deliver it to this client via the socket with "send" function. true for a successful send, false for a failed send
26     bool getMessage(const string& message);
27
28     // Given a list containing this client, we remove the client from the list.
29     void deleteFromList(list<ClientInfo>& all_clients);
30 };
31
```

## – socket\_setups.h

בקובץ זה מוגדר class עם פעולות מקדימות שהשרת מבצע לפני שהוא מתחיל לקבל לקוחות. כלומר נשתמש בפונקציות שבו כדי לאתחל את ספריית Winsock, לאתחל socket עבור השרת ולקשור אותו לכתובת והפורט שלו.

```
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 to 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  We would close the socket and clean up too, but the server never stops running anyway so it's not crucial to mention
22  ...
23  The goal: Use 'recv' to get inputs from the users, Use 'send' to instruct users and
24  forward their messages onwards
25  */
26 class SocketSetups {
27 public:
28     // step 1: Initialize WinSock version 2.2
29     static bool initialize();
30
31     // step 2: Create a listening TCP socket for the server to accpet clients
32     static SOCKET createTCPsocket();
33
34     // step 3.1: Create the server's Address details, ip and port
35     static 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     static bool assignAddress(const SOCKET& listenSocket, sockaddr_in& serverAddr);
39 };
```



## – interact\_with\_client.h

בקובץ זה הפונקציה `interact`. נפעיל את הפונקציה הזו באמצעות `thread` כדי לנהל את התקשורת עם כל הלקוחות א-סינכרונית. הפונקציה נעזרת בפונקציות אחרות ומממשת לוגיקה והעברת הודעות בין השרת ללקוח.

פונקציות העזר הן:

`equalStrings`: משווה מחרוזות בלי חשיבות לאותיות קטנות וגדולות. היא לא מהותית בלוגיקה, היא נועדה לאפשר למשתמש להקליט קלט נוח יותר (למשל "QUIT" במקום "QUIT")

`isQuit`: בודקת אם ההודעה של הלקוח היא "Quit" ואם כן מנתקת אותו

`validateTarget`: בודקת אם המשתמש אליו רוצים לשלוח חוקי ומחזירה פידבק למשתמש.

`getTarget`: אם המשתמש הרצוי מזוהה על ידי השרת תחזיר אותו

```
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  enum Session {
10     REGISTER,
11     SEARCH_TARGET,
12     DELIVER_TO_TARGET
13 };
14 /* Given 2 strings, compare them case isensitively (for example "LiRaN" == "LIRaN") */
15 bool equalStrings(const string& str1, const string& str2);
16
17 /* Given any client's message to the server we first have to check if he entered 'quit'
18    If he did, then we notify the client on his disconnection and return true
19    If he didn't, we return false */
20 bool isQuit(string& message, ClientInfo& client);
21
22 /* Given a client's input of a target he wants to message, the server checks if the target
23    is valid and sends feedback to the client. Errors to be detected are:
24    1. target is the client himself (logical error, client is trying to message himself).
25    2. target doesn't exist (technical error, the target can not be found)
26    If there is no error we notify the client that we found the target, else we
27    send the error to the client. */
28 bool validateTarget(string& targetName, ClientInfo& client, list<ClientInfo>& all_clients);
29
30 /* Once the target is validated, we need to make sure they're still active on send.
31    If the target has disconnected mid conversation we notify the client that the connection is closed
32    and ask him to pick a different target */
33 ClientInfo* getTarget(string& targetName, ClientInfo& client, list<ClientInfo>& all_clients);
34
35 /* 'Interact' is the main function that manages the interactions between the server and the client.
36    It handles the server's actions with a single client and forwards messages from the client
37    to a target client.
38    If needed, the server sends feedback messages regarding errors.
39    Interact utilizes all the functions from above.
40
41    In short, we do the following:
42    0. We got a message from the client
43    1. Check for 'quit' (isQuit), if the client didn't send quit we proceed to 2., else we stop running
44       and remove him (deleteClient).
45    2. Use a 'mode' variable that gets 0, 1 or 2 to distinguish between actions as followed:
46       mode = REGISTER -> We receive client's name and save his details (createClient)
47       mode = SEARCH_TARGET -> we receive the target client and check for errors (isError)
48       mode = DELIVER_TO_TARGET -> We got a message to deliver and we send it to the target client by finding his socket in the clients list
49    3. return to 0.
50    */
51
52 void Interact(SOCKET clientSocket, list<ClientInfo>& all_clients);
53
```

main.h – בקובץ זה פונקציית ה-main, בה אנחנו מקימים את השרת עם כל פעולות ה-socket המקדימות. כלומר אנחנו משתמשים בקלאס socketSetups כדי להקים את השרת ובפונקציות listen, accept כדי לקבל לקוחות. כל לקוח שמתקבל ינוהל בנפרד באמצעות thread ובפונקציית interact.

כדי לנהל את כל הלקוחות במקביל אנחנו צריכים לשמור את כולם. כלומר אנחנו צריכים מבנה נתונים כלשהו כדי לנהל את הלקוחות, ובאמת כפי שנאמר אנחנו משתמשים ברשימה. בפרויקט נזדקק למבנה נתונים שיאפשר להוסיף, למחוק ולחפש לקוחות בתוך מאגר.

נוכל לבצע זאת עם המון שיטות וסוגים של מבנים. למשל אפשר לשמור עץ AVL מאוזן שמסדר את הלקוחות לפי סדר האלף-בית, או להשתמש בדרכים מתוחכמות יותר ויעילות יותר. עם זאת, ההתמקדות היא בתקשורת בין השרת ללקוח לכן נפשט מעט את המימוש ואת ההסברים. נשמור את הלקוחות בתוך רשימה מקושרת (list) מסוג ClientInfo.

נאתחל את הרשימה כריקה, משם כל לקוח שמתחבר לשרת יתווסף לרשימה, כל לקוח שיתנתק מהשרת יימחק מהרשימה ואם נרצה להעביר הודעה אל לקוח מסוים נוכל פשוט לחפש אותו ברשימה ולקבל פוינטר לאובייקט שבו מאוחסנים הפרטים שלו.

נזכיר שכל הפעולות קורות במקביל בין השרת לכל הלקוחות באמצעות פיצולים של thread.

```
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();
21
```

מתוך main.cpp - קבלת הלקוחות ופיצול הקשב של השרת לכולם במקביל:

```
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 manageClient(Interact, clientSocket, ref(clients)); // Each client gets his own treatment simultaneously with threads
        manageClient.detach();
    }
}
closesocket(listenSocket);
WSACleanup();
return 0;
```

## Client interface – אפליקציית הלקוח

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

### – socket\_setups.h

בדומה לאפליקציית השרת, גם באפליקציית הלקוח נגדיר class עם פעולות מקדימות שאנו צריכים לקיים כדי לאפשר תקשורת מבוססת sockets. בקובץ מופיע מימוש בסיסי של אתחול ספריית Winsock, יצירת socket הלקוח 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  class SocketSetups
24  {
25  public:
26      // step 1: Initialize WinSock version 2.2
27      static bool initialize();
28
29      // step 2: Create a TCP socket to contact the server with
30      static SOCKET createTCPsocket();
31
32      // step 3.1: Create the address details of the server the client connects - ip and port
33      static bool createAddress(sockaddr_in& serverAddr, const string& ip, const int portNumber);
34
35      // step 3.2: Connect to the server using the 'connect' function
36      static bool connectToServer(SOCKET& serverSocket, sockaddr_in& serverAddr);
37  };

```

## – interact\_with\_server.h

בקובץ זה הפונקציות `sendMessage`, `receiveMessage`.

פונקציות אלה מרכזות את הלוגיקה של הלקוח בתקשורת עם השרת. פונקציה אחת אחראית על קבלת הודעות מהשרת והשנייה על שליחת הודעות לשרת. הפעולות של הפונקציות משפיעות זו על זו והן מתואמות ביניהן. הן משתמשות בפונקציות אחרות כדי לנהל את התהליכים מול השרת.

פונקציות העזר הן:

`equalStrings`: כמו מקודם, משווה מחרוזות בלי חשיבות לסוג האות. כדי להקל על קלט המשתמש.

`registerClient`: בשלב ההתחברות לשרת, הפונקציה שולחת לשרת את שם הלקוח.

`reachTarget`: כזכור הלקוח צריך לציין את שם הלקוח אליו רוצה לשלוח הודעות, פונקציה זו שולחת לשרת שם זה.

```
4  #include <iostream>
5  #include <thread>
6  #include <string>
7
8  using namespace std;
9
10 enum Session {
11     REGISTER,
12     SEARCH_TARGET,
13     CHAT
14 };
15 /* Given 2 strings, compare them case isensitively(for example "liRaN" == "LIRan") */
16 bool equalStrings(const string&, const string&);
17
18 // Send client's name to the server to be known. Return success (true) or fail (false)
19 bool registerClient(SOCKET& server);
20
21
22 /* Send name of another client we want to message. Server will respond with a feedback message.
23    Return success (true) or fail (false) */
24 bool reachTarget(SOCKET& server);
25
```

```

22  v  /* 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  v  /* 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

```

## - main.h

בקובץ זה פונקציית ה-main, בה אנחנו מבצעים את כל פעולות ה-socket המקדימות. כלומר אנחנו משתמשים בפונקציות של קלאס SocketSetups. חשוב לציין – אנחנו יוצרים שני thread כאשר פונקציות ה-thread הן: sendMessage, receiveMessage - מהקובץ interact\_with\_server.h. 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
```

מתוך main.cpp – חיבור שני הפונקציות כדי לתקשר עם השרת:

```
// Connection is created, we're ready to communicate with the server
cout << "Connection to server established" << endl;

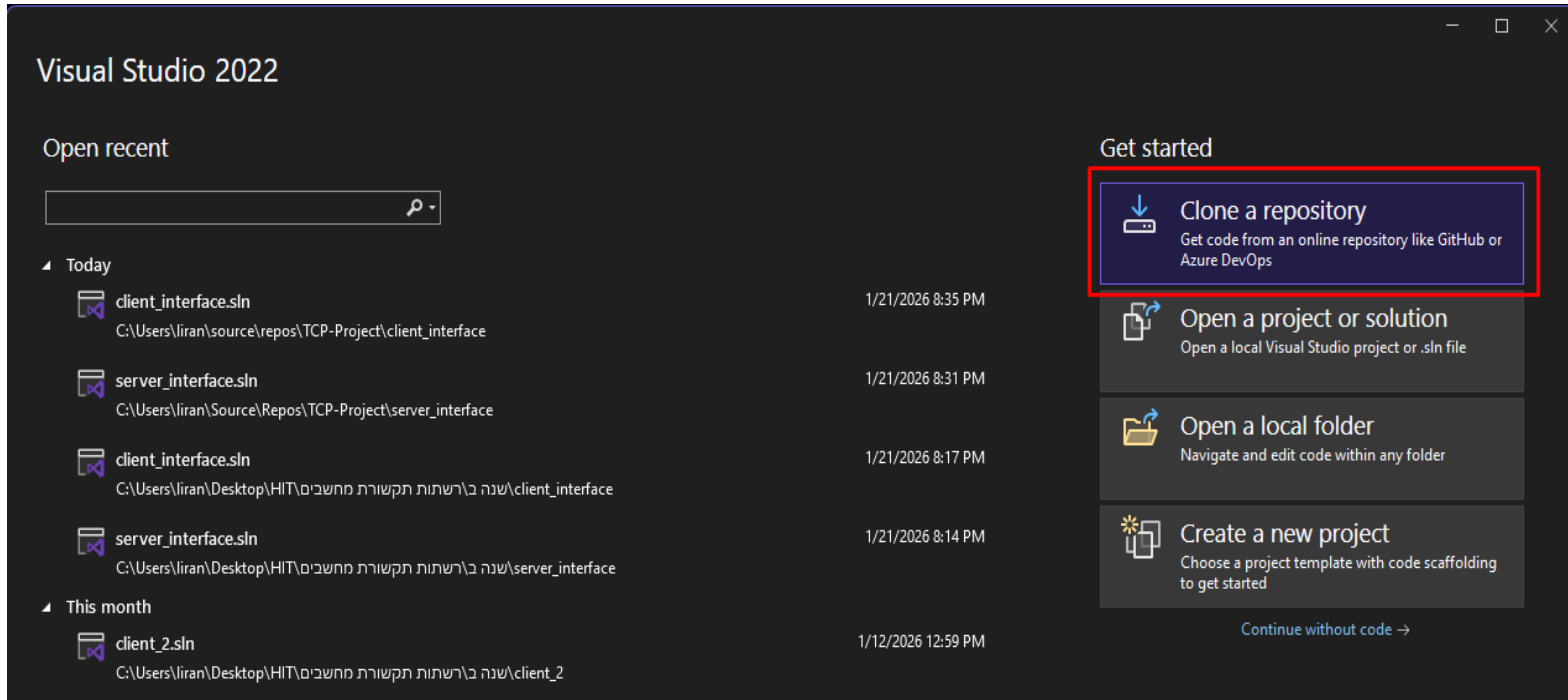
// We split the application's attention to both send messages to the server and receive messages from the server, simultaneously
thread senderthread(sendMessage, clientSocket);
thread receiverthread(receiveMessage, clientSocket);
senderthread.join();
receiverthread.join();

// When both threads finish their run (by typing 'Quit' or by server connection error)
// in other words when the connection to the server stops, the program ends.
closesocket(clientSocket);
WSACleanup();
return 0;
```

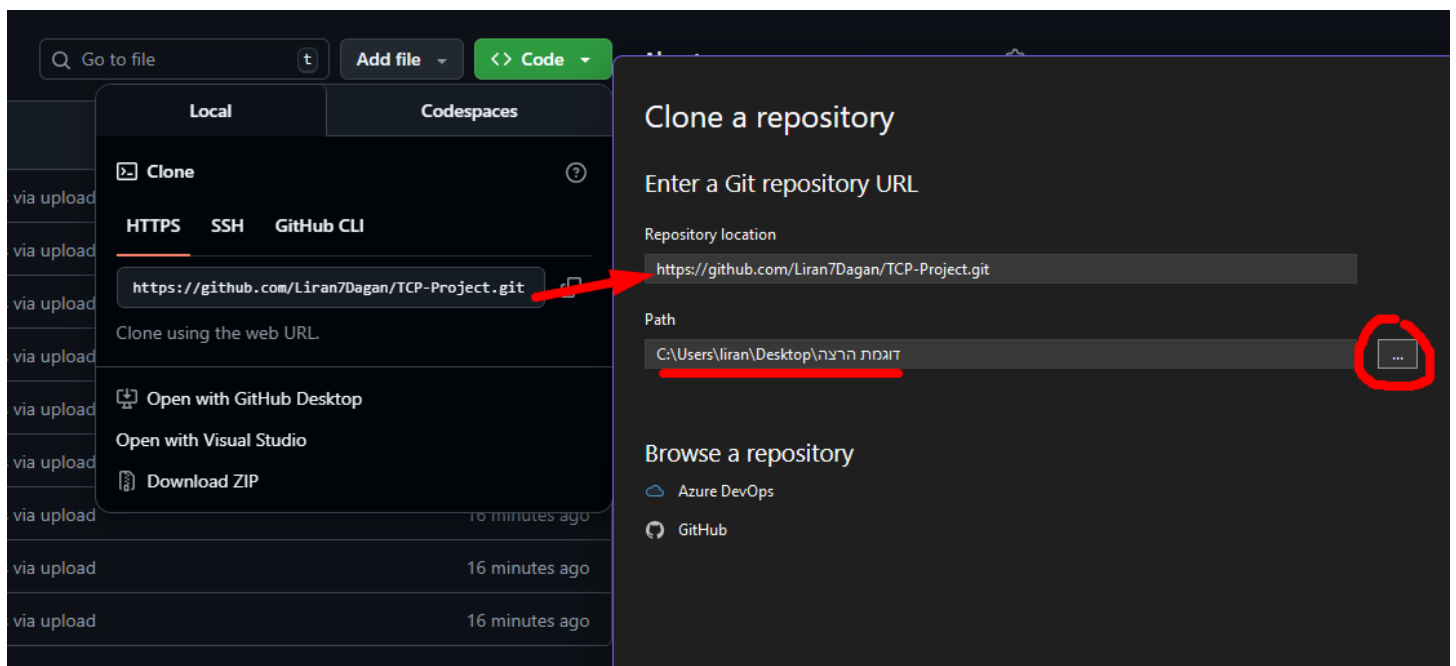
## הוראות התקנה והרצה

לגיטהאב מצורפים התיקיות client\_interface, server\_interface שצוינו קודם לכן. כאמור שתי האפליקציות נכתבו בשפת C++ ולכן כדי להריץ אותן על המחשב יש לפתוח את קבצי ה-solution שלהן בויזואל סטודיו.

פותחים את ויזואל סטודיו ולוחצים clone a repository:

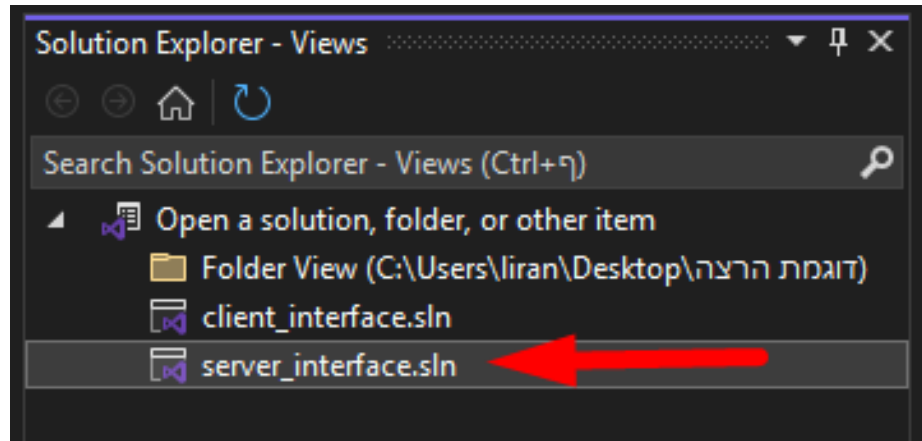


מעתיקים את הלינק של ספריית הגיטהאב, חשוב לשמור בנתיב משמעותי על המחשב כי נשתמש בתיקייה הזאת שוב

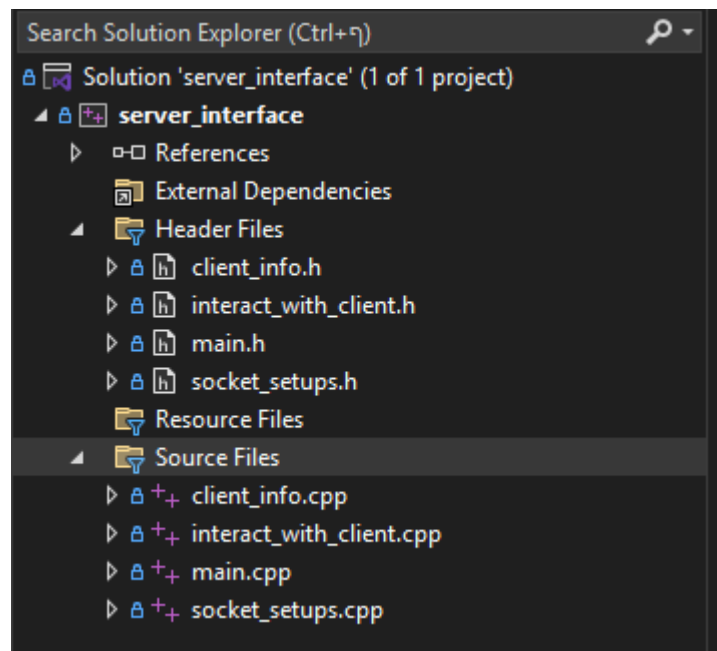




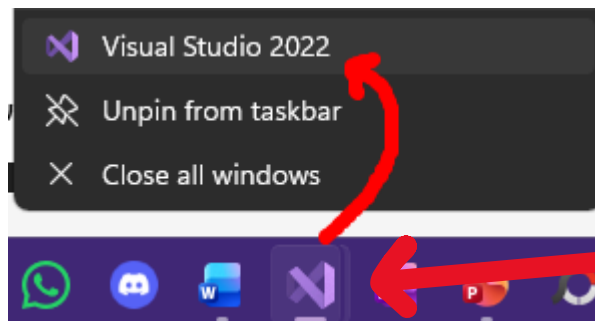
נלחץ clone ומבאן אפשר לפתוח את אפליקציית השרת:



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

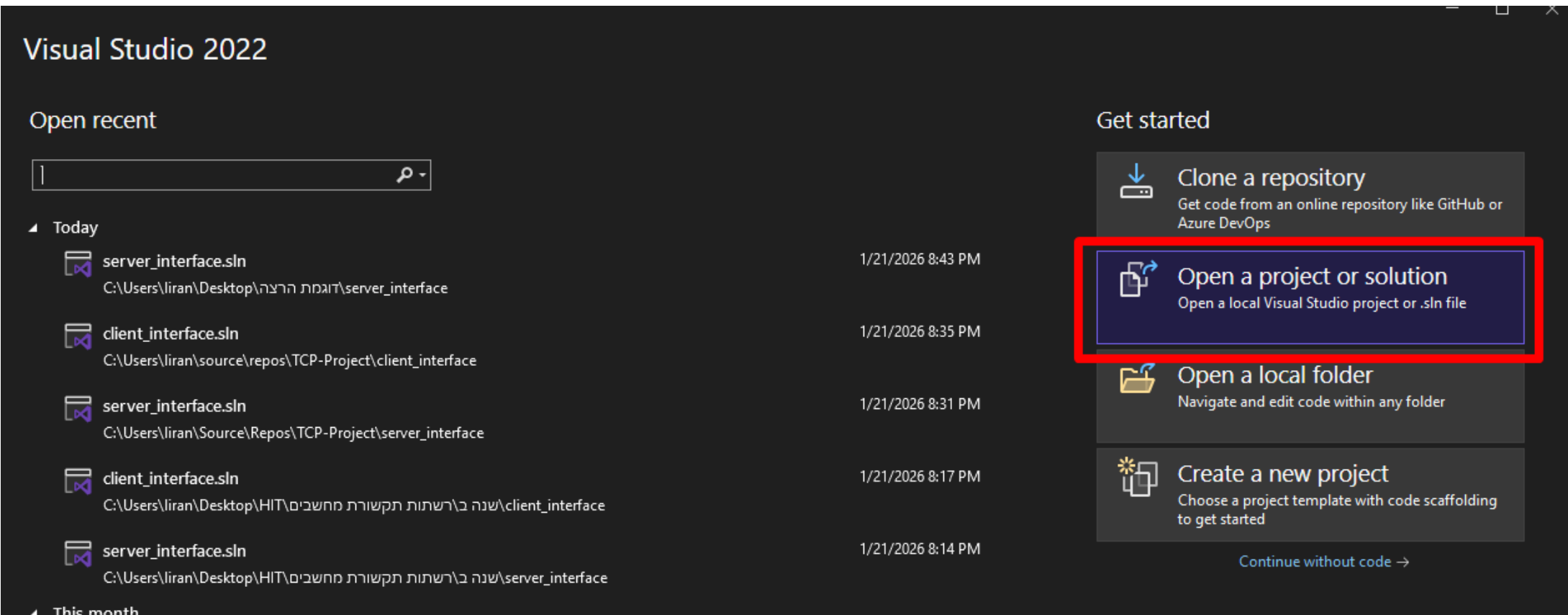


עכשיו צריך לפתוח את אפליקציית הלקוח, בשביל כך צריך לפתוח חלון נוסף של ויזואל סטודיו:

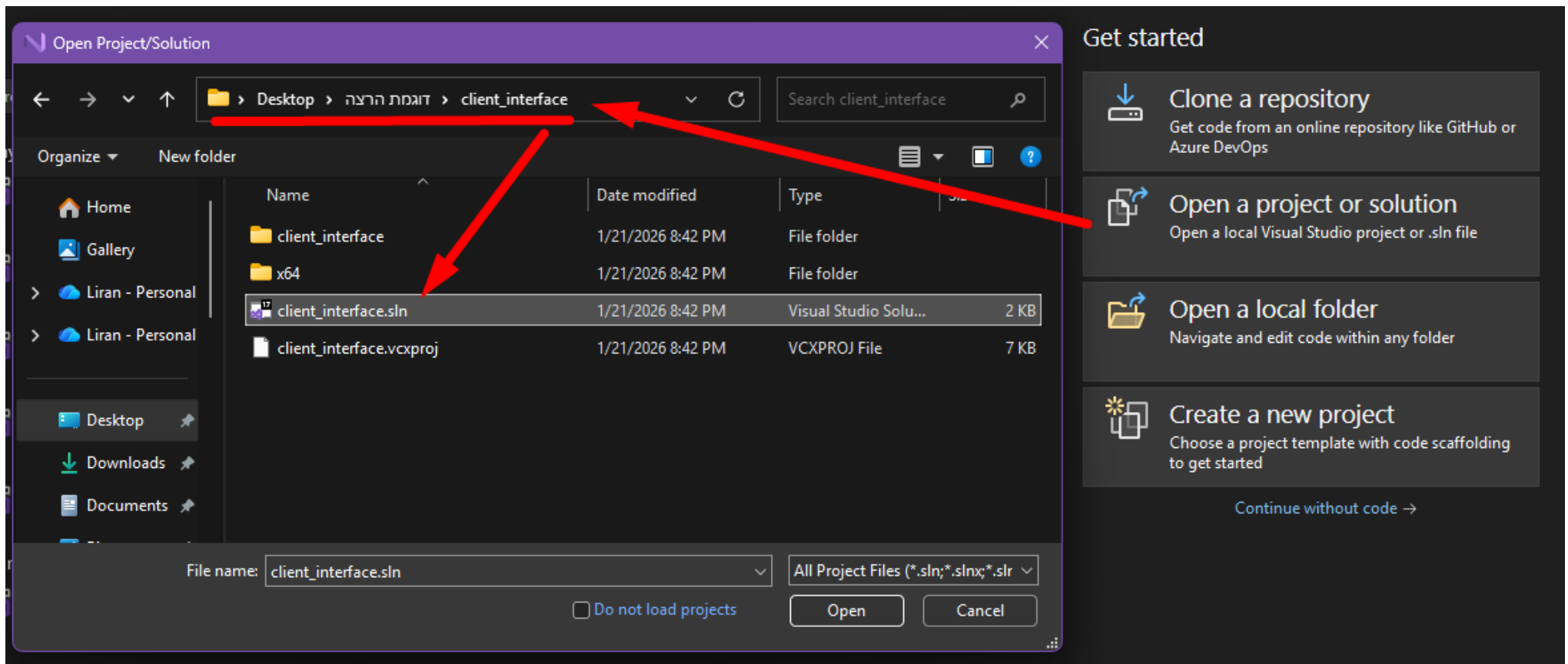


קליק ימני על האייקון של ויזואל סטודיו

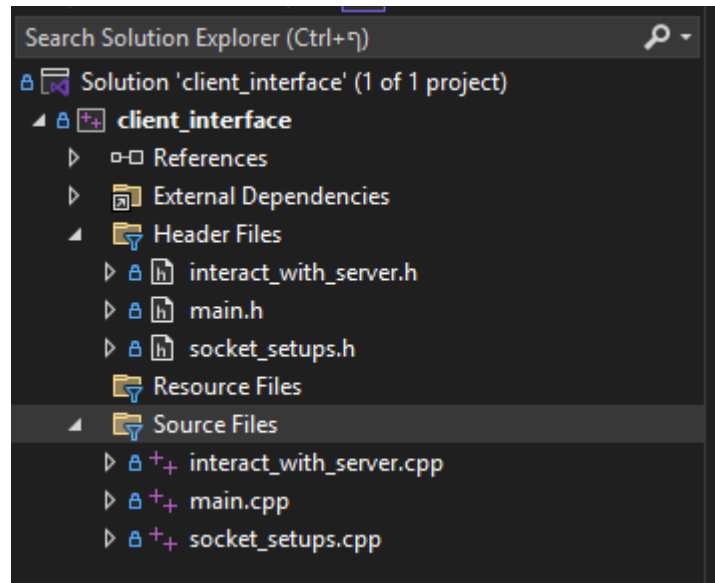
יש לנו חלון חדש של ויזואל סטודיו - נפתח את קובץ ה-solution של אפליקציית הלקוח בנתיב ששמרנו:



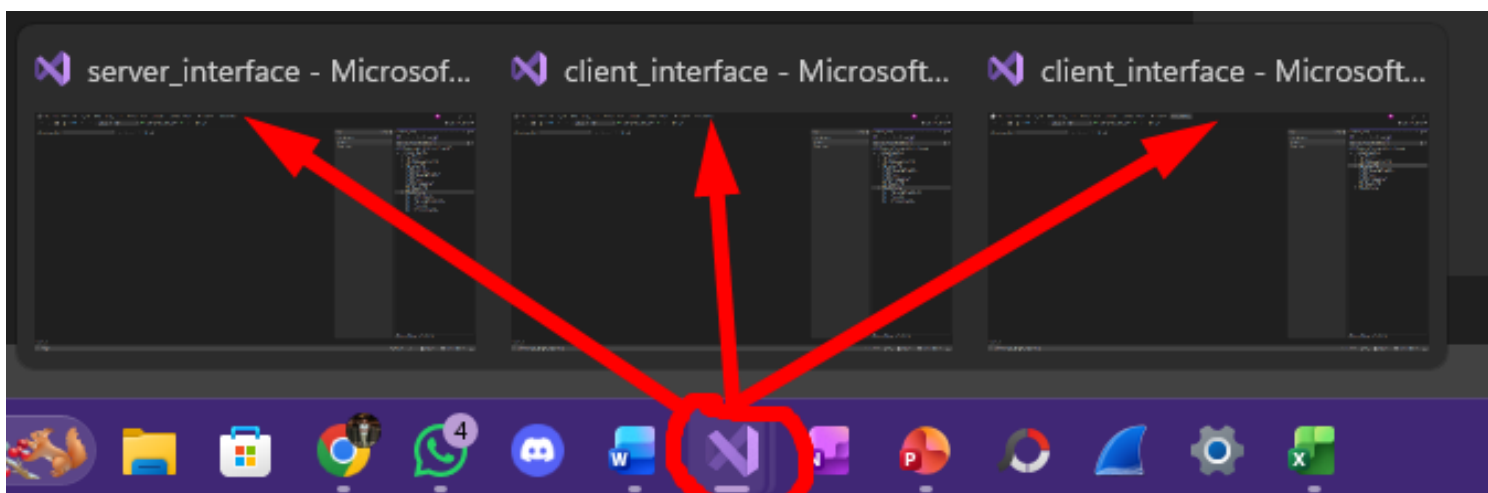
פתיחת אפליקציית הלקוח:



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

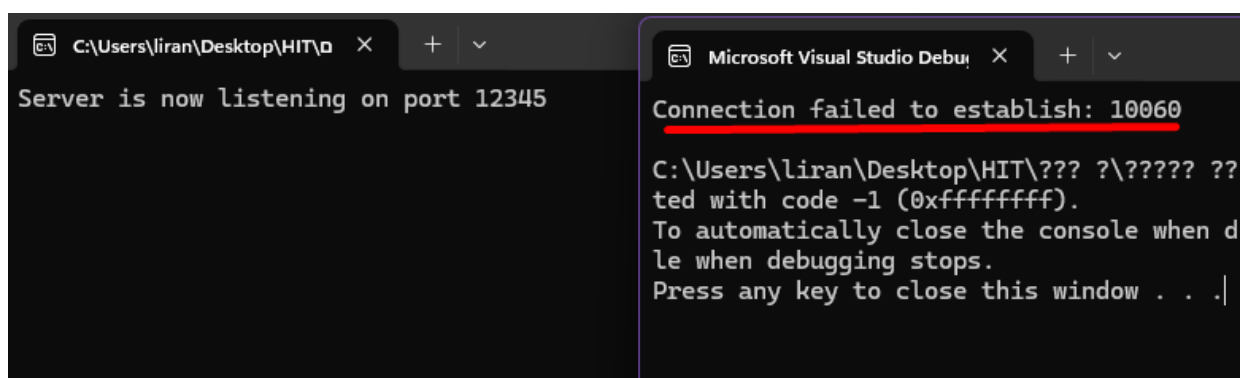


אם נרצה להריץ צ'אט יש לפתוח לפחות עוד לקוח אחד באותו אופן שתואר בשני העמודים האחרונים. עד לקבלה של שרת אחד ומספר כלשהו לבחירתנו של לקוחות, בכה:



באן מסתיים שלב ההתקנה, יש לנו שרת אחד ולפחות שני לקוחות שמובנים להרצה.

שגיאה צפויה היא שגיאה מהסוג:



השגיאה תופיע לאחר זמן מה שהלקוח לא מצליח להתחבר לשרת.  
הסיבה לשגיאה: באפליקציית הלקוח, בקובץ main.cpp, שורה 23:

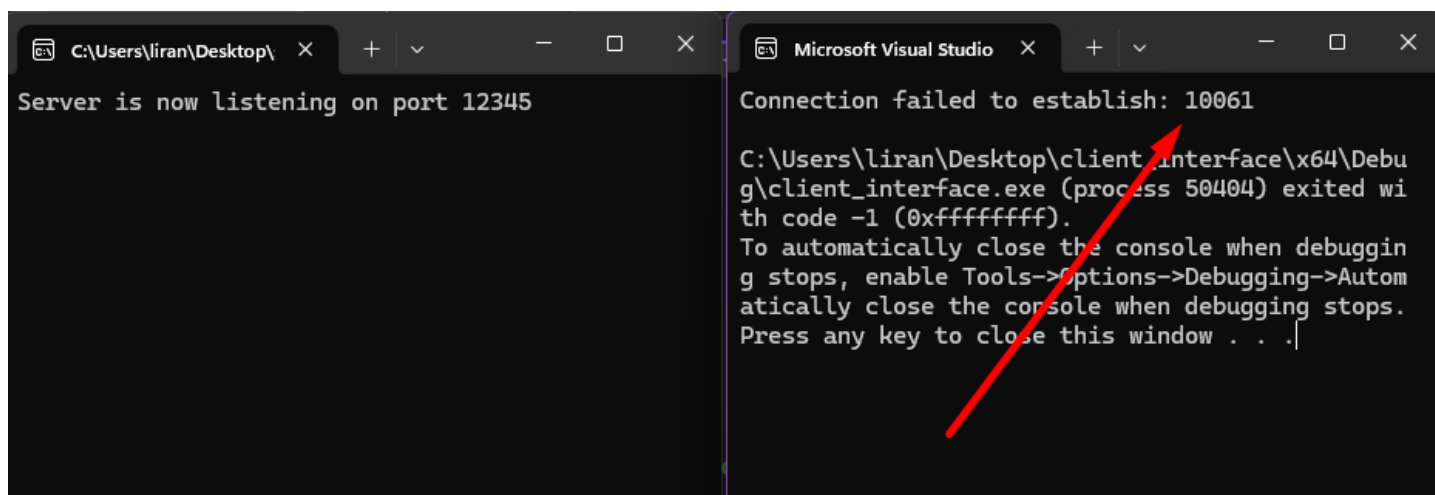
```
1  /* Made by Liran Dagan 215609397 */
2  #include "main.h"
3
4  int main()
5  {
6      // step 1: Start winsock
7      if (!SocketSetups::initialize()) {
8          cout << "Failed to start the application." << WSAGetLastError() << endl;
9          return -1;
10     }
11
12     // step 2: Create TCP type server socket
13     SOCKET clientSocket = SocketSetups::createTCPsocket();
14     if (clientSocket == INVALID_SOCKET)
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 or change if needed
25
26     const int port = 12345; // Our assigned port number
27     if (!SocketSetups::createAddress(serverAddr, ip, port))
28     {
29         cout << "Setting address failed " << WSAGetLastError() << endl;
30         closesocket(clientSocket);
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 באפליקציית הלקוח לזו של מחשב הבודק (או כתובת ה-IP של המחשב עליו רץ השרת).

ישנה גם אופציה להשתמש בכתובת ה-loopback במקום, אם רוצים. אני השתמשתי בכתובת IP אמיתית, מומלץ להשתמש בכתובת אמיתית אבל הוספתי גם את הלופבאק בשביל הנוחות.

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 בזמן הרצת הלקוח, השרת לא קיים).



```

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 קורות קריסות בלתי צפויות. אם קורות שגיאות כלשהן כמו התנתקויות למשל, האפליקציות ידפיסו על המסך הודעות מתאימות.

המסך שמקבלים מאפליקציית השרת הוא נטו אינפורמטיבי, הוא לא מקבל קלט מהמשתמש.  
המסך שמקבלים מאפליקציית הלקוח לעומת זאת הוא כמובן גם אינטראקטיבי.

## דוגמאות קלט ופלט

דוגמה לשיחה בין שני משתמשים, לירן ומהר:

```
Microsoft Visual Studio Del  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?
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 Liranannn

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

Microsoft Visual Studio Debu  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?
Maor
Please Message a Different User.
-----
Who do you want to message?
Liran
Connection created
-----
Liran: Hey Maor!!!

Whats upppppppppp Liranannn

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
```

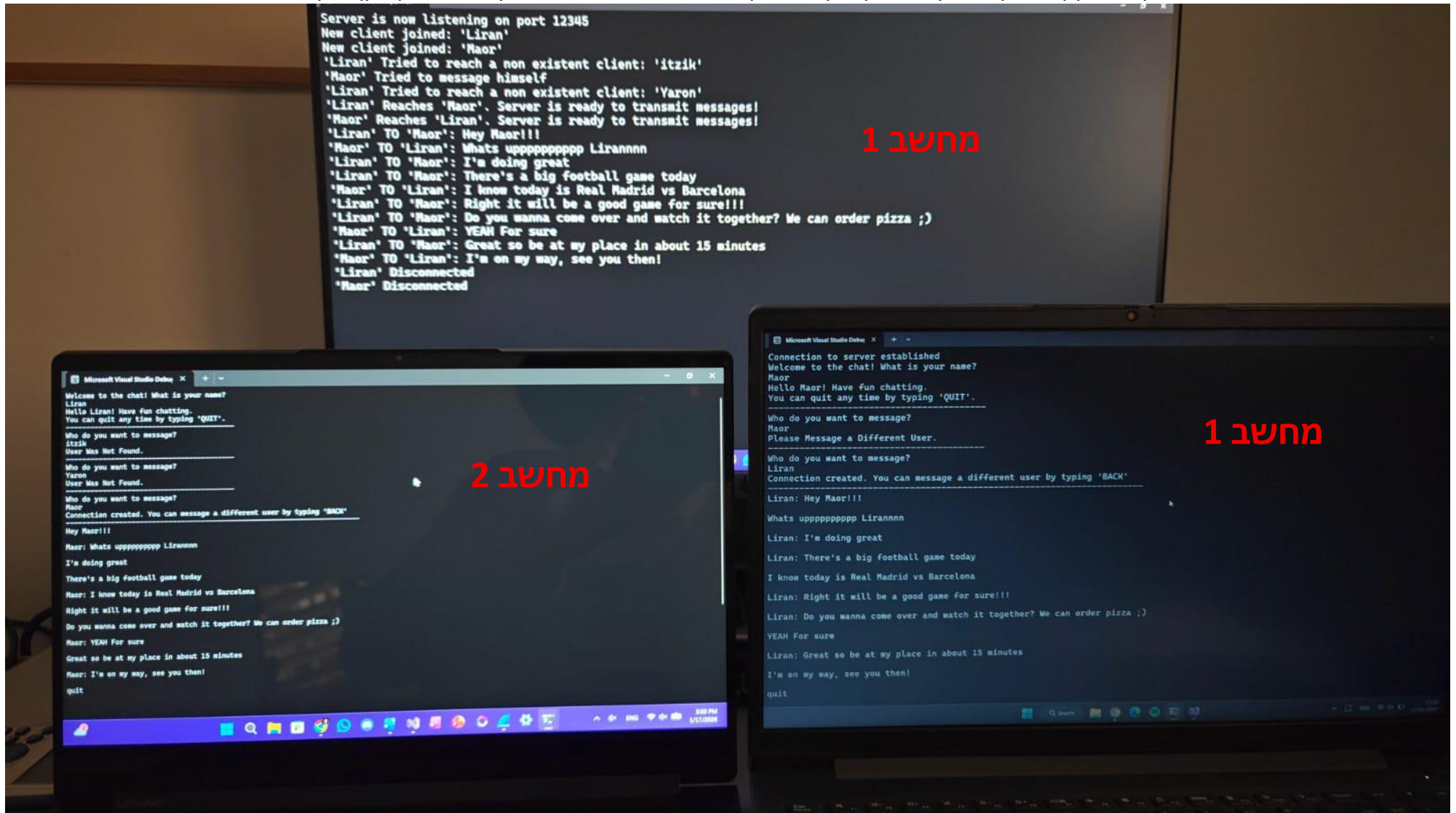
עבור השיחה שלמעלה, פלט השרת הוא:

```
C:\Users\liran\Desktop\HIT\ >
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 upppppppppp Lirannnn
'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
|
```



מנקודת המבט שלי – הכותב :) למעלה פלט השרת, משמאל הקלט/פלט של מאור

בתמונה שני מחשבים – המסך העליון (השרת) והמסך הימני (הלוח מאור) שייכים לאותו מחשב. המסך השמאלי (לירן) שייך למחשב אחר באותה רשת.



קצת על התקשורת בין השרת ללקוחות בשיחה זו (ננתח תעבורה בהמשך, כאן אסביר על האפליקציה): 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". כשלירן 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 הבאה 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**

```

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
בקשה להתחבר למאור, בשלב זה הוא קיים
Connection created
-----
hey maor
לירן שולח הודעה למאור, בשלב זה הוא מנותק
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**

```

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
מאור מתנתק תוך כדי שיחה
C:\Users\liran\Desktop\HIT\??? \????? \?????
code 0 (0x0).
To automatically close the console when debugging stops,
press any key to close this window . . .

```

כך 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 בהתחברות שלו אליו. הוזכר מספר פעמים שהלקוחות מתנהלים אסינכרונית. היתרון שהדבר נותן הוא יכולת גמישה למשתמש לשלוח הודעות רק למי שהוא רוצה בכל רגע. 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
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
Connection created
-----
HI!
|
```

ההודעה של לירן

ההודעה של מאור

רק בשלב הזה מאור מתחבר

ננתח את השיחה הראשונה (עמוד 31) בין לירן למאור ב-Wireshark. ננתח את הצד של לירן (שרץ במחשב אחר ברשת מהשרת ומאור):

קובץ הלכידה גם מצורף בשם 2\_clients\_chat.pcap

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
245	21.223307	192.168.1.15	192.168.1.56	57760	12345	TCP		1	0	0	0	0	0	4	128
249	21.341685	192.168.1.56	192.168.1.15	12345	57760	TCP		1	1	0	0	0	1	4	128
250	21.341881	192.168.1.15	192.168.1.56	57760	12345	TCP		0	1	0	0	1	1	4	128
561	43.966002	192.168.1.15	192.168.1.56	57760	12345	TCP	4c6972616e	0	1	0	1	1	1	4	128
562	44.349885	192.168.1.15	192.168.1.56	57760	12345	TCP	4c6972616e	0	1	0	1	1	1	4	128
563	44.387799	192.168.1.56	192.168.1.15	12345	57760	TCP		0	1	0	0	1	6	4	128
657	63.896643	192.168.1.15	192.168.1.56	57760	12345	TCP	69747a696b	0	1	0	1	6	1	4	128
658	64.278017	192.168.1.15	192.168.1.56	57760	12345	TCP	69747a696b	0	1	0	1	6	1	4	128
659	64.286456	192.168.1.56	192.168.1.15	12345	57760	TCP	533a55736572...	0	1	0	1	1	11	4	128
660	64.339513	192.168.1.15	192.168.1.56	57760	12345	TCP		0	1	0	0	11	15	4	128
661	64.461313	192.168.1.56	192.168.1.15	12345	57760	TCP		0	1	0	0	15	11	4	128
738	78.110474	192.168.1.15	192.168.1.56	57760	12345	TCP	7961726f6e	0	1	0	1	11	15	4	128
744	78.347603	192.168.1.56	192.168.1.15	12345	57760	TCP	533a55736572...	0	1	0	1	15	16	4	128
745	78.390215	192.168.1.15	192.168.1.56	57760	12345	TCP		0	1	0	0	16	29	4	128
746	78.742892	192.168.1.56	192.168.1.15	12345	57760	TCP	533a55736572...	0	1	0	1	15	16	4	128
747	78.742974	192.168.1.15	192.168.1.56	57760	12345	TCP		0	1	0	0	16	29	4	128
844	83.144811	192.168.1.15	192.168.1.56	57760	12345	TCP	4d616f72	0	1	0	1	16	29	4	128
845	83.188005	192.168.1.56	192.168.1.15	12345	57760	TCP	533a55736572...	0	1	0	1	29	20	4	128
846	83.239717	192.168.1.15	192.168.1.56	57760	12345	TCP		0	1	0	0	20	40	4	128
918	108.988693	192.168.1.15	192.168.1.56	57760	12345	TCP	486579204d61...	0	1	0	1	20	40	4	128
919	109.157486	192.168.1.56	192.168.1.15	12345	57760	TCP		0	1	0	0	40	31	4	128
1018	126.671707	192.168.1.56	192.168.1.15	12345	57760	TCP	433a4d616f72...	0	1	0	1	40	31	4	128
1019	126.717885	192.168.1.15	192.168.1.56	57760	12345	TCP		0	1	0	0	31	74	4	128
1161	136.825199	192.168.1.15	192.168.1.56	57760	12345	TCP	49276d20646f...	0	1	0	1	31	74	4	128
1163	137.108872	192.168.1.56	192.168.1.15	12345	57760	TCP		0	1	0	0	74	46	4	128
1266	157.734175	192.168.1.15	192.168.1.56	57760	12345	TCP	546865726527...	0	1	0	1	46	74	4	128
1267	158.000740	192.168.1.56	192.168.1.15	12345	57760	TCP		0	1	0	0	74	79	4	128
1386	181.944050	192.168.1.56	192.168.1.15	12345	57760	TCP	433a4d616f72...	0	1	0	1	74	79	4	128
1387	181.985822	192.168.1.15	192.168.1.56	57760	12345	TCP		0	1	0	0	79	122	4	128
1510	211.550325	192.168.1.15	192.168.1.56	57760	12345	TCP	526967687420...	0	1	0	1	79	122	4	128
1511	211.757499	192.168.1.56	192.168.1.15	12345	57760	TCP		0	1	0	0	122	119	4	128
1674	235.291860	192.168.1.15	192.168.1.56	57760	12345	TCP	446f20796f75...	0	1	0	1	119	122	4	128
1675	235.392300	192.168.1.56	192.168.1.15	12345	57760	TCP		0	1	0	0	122	186	4	128
1706	245.037435	192.168.1.56	192.168.1.15	12345	57760	TCP	433a4d616f72...	0	1	0	1	122	186	4	128
1707	245.079946	192.168.1.15	192.168.1.56	57760	12345	TCP		0	1	0	0	186	143	4	128
1708	245.651650	192.168.1.56	192.168.1.15	12345	57760	TCP	433a4d616f72...	0	1	0	1	122	186	4	128
1709	245.651729	192.168.1.15	192.168.1.56	57760	12345	TCP		0	1	0	0	186	143	4	128
1710	245.652483	192.168.1.56	192.168.1.15	12345	57760	TCP	433a4d616f72...	0	1	0	1	122	186	4	128
1711	245.652526	192.168.1.15	192.168.1.56	57760	12345	TCP		0	1	0	0	186	143	4	128
1825	269.606406	192.168.1.15	192.168.1.56	57760	12345	TCP	477265617420...	0	1	0	1	186	143	4	128
1826	269.819964	192.168.1.56	192.168.1.15	12345	57760	TCP		0	1	0	0	143	229	4	128
1950	287.024120	192.168.1.56	192.168.1.15	12345	57760	TCP	433a4d616f72...	0	1	0	1	143	229	4	128
1951	287.069915	192.168.1.15	192.168.1.56	57760	12345	TCP		0	1	0	0	229	179	4	128
1977	291.217116	192.168.1.15	192.168.1.56	57760	12345	TCP	71756974	0	1	0	1	229	179	4	128
1979	291.260262	192.168.1.56	192.168.1.15	12345	57760	TCP	533a71756974	0	1	0	1	179	233	4	128
1981	291.285594	192.168.1.56	192.168.1.15	12345	57760	TCP		0	1	1	0	185	233	4	128
1982	291.285649	192.168.1.15	192.168.1.56	57760	12345	TCP		0	1	0	0	233	186	4	128
1984	291.727135	192.168.1.15	192.168.1.56	57760	12345	TCP		0	1	1	0	233	186	4	128
1986	291.793995	192.168.1.56	192.168.1.15	12345	57760	TCP		0	1	0	0	186	234	4	128

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

מזהים גם שקיימות שתי כתובות IP:

192.168.1.15 – כתובת המחשב שבו רץ המשתמש לירן

192.168.1.56 – כתובת המחשב שבו רץ השרת

מזהים גם שקיימים שני מספרי פורט:

57760 – מספר הפורט של לירן

12345 – מספר הפורט של השרת (הגדרנו זאת בקוד)

נתייחס מכאן והילך בנייתו למספרי הפורט 57760 ו-12345 כמזהים, כשהראשון מזהה עבור לירן והשני מזהה עבור השרת. כמובן שאפשר להתייחס גם לכתובות ה-IP, כמזהים, 192.168.1.15 עבור לירן ו-192.168.1.56 עבור השרת.

נשים לב לחיבור של המשתמש לירן לשרת. בחבילות הראשונות נוכל בבירור לראות את ה-3 way handshake של TCP בין המשתמש לשרת:

No.	Time	Source Address	Destination Address	Source Port	Destination Port	Protocol	TCP payload	Syn	Acknowledgment
245	21.223307	192.168.1.15	192.168.1.56	57760	12345	TCP		1	0
249	21.341685	192.168.1.56	192.168.1.15	12345	57760	TCP		1	1
250	21.341881	192.168.1.15	192.168.1.56	57760	12345	TCP		0	1

בחבילה מספר 245 קורה  $\text{syn}=1, \text{ack}=0, \text{destination port} = 12345$ , כלומר זו חבילה שהשרת קיבל מהמשתמש "לירן" והיא מהווה בקשה של הלקוח להתחבר לשרת.

בחבילה מספר 249 קורה  $\text{syn}=1, \text{ack}=1, \text{source port} = 12345$ , כלומר זו חבילה שהשרת שלח למשתמש "לירן" והיא מהווה אישור לבקשת החיבור של הלקוח

בחבילה מספר 250 קורה  $\text{syn}=0, \text{ack}=1, \text{destination port} = 12345$ , כלומר זו חבילה שהשרת קיבל מהמשתמש "לירן" שמהווה את אישור ההתחברות של הלקוח, תהליך 3 way handshake הסתיים.

ניתן לראות את כל ההודעות שלירן שולח לשרת ואת כל ההודעות שהשרת שולח ללירן, למשל:

לירן שולח את שמו לשרת כדי להירשם לצ'אט, ההודעה הראשונה:

561	43.966002	192.168.1.15	192.168.1.56	57760	12345	TCP	4c6972616e
562	44.349885	192.168.1.15	192.168.1.56	57760	12345	TCP	4c6972616e

Frame 561: Packet, 59 bytes on wire (47 bytes captured)	0000	ea 94 f6 21 f7 c8 08 5b d6 f4 fe 1b 08 00 45 00	...!...[.....E
Ethernet II, Src: Intel f4:fe:1	0010	00 2d e8 e2 40 00 80 06 00 00 c0 a8 01 0f c0 a8	...@...J...8..
Internet Protocol Version 4, Src: 192.168.1.15, Destination: 192.168.1.56	0020	01 38 e1 a0 30 39 00 b7 64 f2 8a 31 d7 af 50 18	...09...1...P
TCP, Src Port: 57760, Destination Port: 12345	0030	00 ff 83 b7 00 00 4c 69 72 61 6e	...Li ran

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.

השרת מעביר ללירן את ההודעות של מאור, לדוגמה:

1018	126.671707	192.168.1.56	192.168.1.15	12345	57760	TCP	433a4d616f72...
------	------------	--------------	--------------	-------	-------	-----	-----------------

Identification: 0x2cff (111)	0000	08 5b d6 f4 fe 1b ea 94 f6 21 f7 c8 08 00 45 00	...[.....!...E
010. .... = Flags: 0x2, Do not fragment	0010	00 4a 2c ff 40 00 80 06 4a 17 c0 a8 01 38 c0 a8	..J, @...J...8..
...0 0000 0000 0000 = Fragment offset	0020	01 0f 30 39 e1 a0 8a 31 d7 d6 00 b7 65 10 50 18	...09...1...e P
Time to Live: 128	0030	10 04 99 3f 00 00 43 3a 4d 61 6f 72 3a 20 57 68	...?..C: Maor: Wh
Protocol: TCP (6)	0040	61 74 73 20 75 70 70 70 70 70 70 70 70 70 70 70	ats uppp pppppp
Header Checksum: 0x4a17 [valid]	0050	4c 69 72 61 6e 6e 6e 6e	Liranannn

באופן דומה להתחברות, אפשר לבחון את ניתוק המשתמש מהשרת, נשים לב לפאקטות:  
 1981, 1982, 1984, 1986. נזכיר שבתמונה הבאה, הביט הימני הוא השדה Fin, משמאלו Ack  
 ומשמאלו Syn שמאופס בכל השורות.

1981	291.285594	192.168.1.56	192.168.1.15	12345	57760	TCP	0	1	1
1982	291.285649	192.168.1.15	192.168.1.56	57760	12345	TCP	0	1	0
1984	291.727135	192.168.1.15	192.168.1.56	57760	12345	TCP	0	1	1
1986	291.793995	192.168.1.56	192.168.1.15	12345	57760	TCP	0	1	0

אלו הפאקטות האחרונות שהוקלטו עבור המשתמש "לירן" שבפורט 57760.  
 נשים לב לתהליך 4 way handshake עבור הניתוק של לירן מהשרת:  
 בחבילה מספר 1981 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 1982 קורה  $fin=0, ack=1$ , וזו חבילה ששולח לירן לשרת. כאן לירן מאשר את בקשת  
 השרת לניתוק החיבור.  
 בחבילה מספר 1984 קורה  $fin=1, ack=1$ , וזו חבילה ששולח לירן לשרת. כאן לירן שולח בקשה לניתוק  
 החיבור.  
 בחבילה מספר 1986 קורה  $fin=0, ack=1$ , וזו חבילה ששולח השרת ללירן. כאן השרת מאשר את  
 הבקשה של לירן והחיבור נסגר סופית. זו הפאקטה האחרונה שעברה.  
 בשכבת הרשת, כאמור כתובות ה-IP שונות עבור לירן והשרת כי הם רצים במכשירים שונים על אותה  
 רשת.  
 יש לציין שבשונה מהרצה של כל האפליקציות על אותו המחשב ועל כתובת ה-loopback, השתמשנו  
 בכתובות IP אמיתיות של מחשבים שונים כדי לוודא את ההרצה עבור מכשירים שונים.  
 את השרת הגדרנו בכתובת 0.0.0.0, כלומר אפשרנו לו להאזין לכל כתובות ה-IP של הרשת המקומית  
 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. (כזכור כתובת השרת היא 192.168.1.56, הכתובת  
 של המחשב עצמו).



## כותרות ה-IPv4 של כל הפאקטות הן מהצורה:

```

▼ Internet Protocol Version 4, Src: 192.168.1.15, Dst: 192.168.1.56
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  ▼ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    0000 00.. = Differentiated Services Codepoint: Default (0)
    .... ..00 = Explicit Congestion Notification: Not ECN-Capable Transport (0)
  Total Length: 40
  Identification: 0xe8f9 (59641)
  ▼ 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: 192.168.1.56
  [Stream index: 11]
    
```

למעט שינויים מעת לעת בשדות ה-Total Length בהתאם לגודל הסגמנט וההודעה, ובשדה ה-Identification. בנוסף כמובן שמפעם לפעם כתובות ה-IP מקור ויעד מתחלפים ביניהם (לירן והשרת מחליפים תפקידים, אחד שולח והשני מקבל). נשים לב לסוג הפרוטוקול, 6 עבור חיבור TCP שהגדרנו. נשים לב גם לשדה

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

הסיבה ששדה זה לא משתנה היא שחיבור הרשת הוא בין מכשירים באותה רשת מקומית. על אף שהמכשירים נפרדים ובעלי כתובות IP שונות, אין ראוטרם או רכיבי רשת אחרים בדרך, הפאקטות לא עוזבות את הרשת המקומית ולכן תמיד נשארות עם ערך קבוע וגדול.

בהמשך לתמונות האחרונות, כמובן שבקובץ הלכידה שמצורף (2\_clients\_chat.pcap) אפשר לעיין בכל ההודעות ברמת האפליקציה, כמו:

לירן שולח לשרת "Itzik" בפאקטה 658 (משתמש אליו רוצה לשלוח הודעה)

657	63.896643	192.168.1.15	192.168.1.56	57760	12345	TCP	69747a696b
658	64.278017	192.168.1.15	192.168.1.56	57760	12345	TCP	69747a696b

Ethernet II, Src: Intel_	0000	ea 94 f6 21 f7 c8 08 5b d6 f4 fe 1b 08 00 45 00	...
Internet Protocol Versio	0010	00 2d e8 e5 40 00 80 06 00 00 c0 a8 01 0f c0 a8	...
0100 .... = Version:	0020	01 38 e1 a0 30 39 00 b7 64 f7 8a 31 d7 af 50 18	...
.... 0101 = Header Le	0030	00 ff 83 b7 00 00 69 74 7a 69 6b	...
▼ Differentiated Serv			...
0000 00.. = Differ			...

השרת משיב ללירן: S:UserNotFound בפקטה 659 כי איציק הוא לא משתמש קיים:

659	64.286456	192.168.1.56	192.168.1.15	12345	57760	TCP	533a55736572...
▶ Ethernet II, Src: ea:94:4d:00:00:00	0000	08 5b d6 f4 fe 1b ea 94 f6 21 f7 c8 08 00 45 00		[.....!....E.			
▼ Internet Protocol Version 4	0010	00 36 2c f9 40 00 80 06 4a 31 c0 a8 01 38 c0 a8		6,.,@... J1...8..			
0100 .... = Version: 4	0020	01 0f 30 39 e1 a0 8a 31 d7 af 00 b7 64 fc 50 18		..09...1 ....d.P.			
.... 0101 = Header Length: 20 bytes	0030	10 04 94 04 00 00 53 3a 55 73 65 72 4e 6f 74 46		.....S: UserNotF			
▼ Differentiated Services Code Point	0040	6f 75 6e 64		ound			
0000 00.. = Differentiated Services Code Point							

הודעה שלירן שלח למאור דרך השרת בפקטה 1674: "Wanna come over? We can order pizza"

1674	235.291860	192.168.1.15	192.168.1.56	57760	12345	TCP	446f20796f75...					
.... ..00 = Explicitly set	0000	ea 94 f6 21 f7 c8 08 5b d6 f4 fe 1b 08 00 45 00	...!...[.....E.									
Total Length: 107	0010	00 6b e8 f2 40 00 80 06 00 00 c0 a8 01 0f c0 a8	k.,@... .....									
Identification: 0xe8f2	0020	01 38 e1 a0 30 39 00 b7 65 68 8a 31 d8 28 50 18	8. 09.. eh.1.(P.									
▼ 0100 .... = Flags: 0x0000	0030	00 ff 83 f5 00 00 44 6f 20 79 6f 75 20 77 61 6e	.....Do you wan									
0... .... = Reserved bits	0040	6e 61 20 63 6f 6d 65 20 6f 76 65 72 20 61 6e 64	na come over and									
.1... .... = Don't set sequence number if no window	0050	20 77 61 74 63 68 20 69 74 20 74 6f 67 65 74 68	watch i t togeth									
..0. .... = More fragments	0060	65 72 3f 20 57 65 20 63 61 6e 20 6f 72 64 65 72	er? We c an order									
...0 0000 0000 0000 = Offset: 0	0070	20 70 69 7a 7a 61 20 3b 29	pizza ; )									
Time to Live: 128												
Protocol: TCP (6)												

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





```
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

[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]

# ביבליוגרפיה ומקורות

- Geeks For Geeks on TCP analysis using wireshark – [/https://www.geeksforgeeks.org/ethical-hacking/tcp-analysis-using-wireshark](https://www.geeksforgeeks.org/ethical-hacking/tcp-analysis-using-wireshark)
- Jim Kurose on Wireshark – <https://youtu.be/kCwd2YoJcvg?si=Tpu3bIBNL6yWyGfe>
- Nicolas Day on C++ socket programming (part 1) - <https://youtu.be/gntyAFoZp-E?si=59aVk5LLMTQO1X15>
- Nicolas Day on C++ socket programming (part 2) [https://youtu.be/sXW\\_sNGvqcU?si=ZmPkmQ1Q2F5KMAow](https://youtu.be/sXW_sNGvqcU?si=ZmPkmQ1Q2F5KMAow)
- Jim Kurose on socket programming – <https://youtu.be/iHMMo7SDfQ?si=q-hwG4Nf6ODydAAQ>
- stack overflow Q&A's <https://stackoverflow.com/questions>

As well as other public Computer Science and networking forums, Youtube tutorials regarding C++ libraries, wireshark, TCP and IP properties

לא נעשה שימוש בכלי AI בשום שלב של הפרויקט. הוא נכתב לאורך כל הסמסטר.  
כל פסקה ותמונה בדו"ח זה וכל שאר הקבצים שמצורפים:  
כולל קבצי לכידה ב-Wireshark, קבצי קוד בשפת C++, כולל הלוגיקה והתיעודים שלהם נכתבו  
על ידי. הרווחתי מהפרויקט הבנה פרקטית ועמוקה יותר, של המון מושגים ונושאים בסילבוס.

---

# *Fin*

---