

Assignment 3 - 208748368

הקדמה :

כמה מילימ' של'. ניסתי לפרק את דרישות המטלה, כדי לענות על הכל, כולל הסברים דוגמאות וצילומי מסך מההקלטה מתוך Wireshark.

בנוסף חילקתי את המטלה לשני חלקים :
חלק ראשון - צילומי מסך (Wireshark)
חלק שני - בדיקות הרצה

צילומי ההקלטה של סעיפים 1-7 שייכים להקלטה בשם "1", כל הקוד נכתב ונערך ב- Visual Studio.

צילומי מסך (Wireshark) :

1. הקמת חיבור TCP

בשלב הראשון הלקוח יוצר חיבור TCP לשרת באמצעות TCP socket. הבחירה נפתחת פעם אחת ומשמש להמשך כל התקשרות בין הלקוח לשרת. בשלב זה מאפשר העברת נתונים בצורה אמינה ומוסדרת, ומהווה את הבסיס לכל הודעה שנסלחות בהמשך.

Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM [SYN] 12345 → 65412 56	TCP	127.0.0.1	131.697907 4444
Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM [SYN, ACK] 65412 → 12345 56	TCP	127.0.0.1	131.697983 4445
Seq=1 Ack=1 Win=65280 Len=0 [ACK] 12345 → 65412 44	TCP	127.0.0.1	131.698016 4446

בצלום ניתן לראות שהלקוח שולח חבילה SYN לצורך ייזום החיבור. לאחר מכן השרת מגיב בחביבת SYN, ACK שהוא מאשרת את קבלת הבקשה. לבסוף הלקוח שולח חבילת ACK ובכך החיבור בין הלקוח לשרת הוקם בהצלחה.
*הצלום לקוח מהקלטה 1

2. ברמת שכבת היישום Handshake

לאחר פתיחת חיבור ה TCP מתבצע Handshake ברמת שכבת היישום. בשלב זה הלקוח והשרת מחליפים הודעות JSON "עודיות שמטרתן לוודא שני הצדדים מסונכרים ומוכנים להתחיל בהעברת נתונים".

חשוב לציין שה Handshake זהה אינו חלק מ TCP עצמו, אלא ממומש כחלק מהפrotocol של האפליקציה.

Info	Length	Protocol	Destination	Source	Time	No.
Frame 4447: Packet, 59 bytes on wire (472 bits), 59 bytes captured (472 bits) on interface 'UdeviceWPF_Loopback', id 0	59	TCP	127.0.0.1	131.698276 4447		
Seq=1 Ack=1 Win=65280 Len=15 [PSH, ACK] 12345 → 65412 59	59					
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 131.698276, Seq: 1, Len: 15						
Transmission Control Protocol, Src Port: 65412, Dst Port: 12345, Seq: 1, Len: 15						
Data (15 bytes)						

א. בצלום מעלה ניתן לראות את תחילת התהיליך בשכבה היישום. ב Frame 4447 נשלחת חבילה PSH, ACK מהלקוח לשרת (Len=15). בחולונית ה-Data-Lמטה, ניתן לראות בבירור שהלקוח שולח את הפקודה { "type": "SIN" }.

וזהו בבקשת ההתחברות הראשונית שמוגדרת בפrotocol שלנו, והוא מסמן לשרת שהלקוח מעוניין להתחיל בתקשרות.

Info	Length	Protocol	Destination	Source	Time	No.
Frame 4449: Packet, 65 bytes on wire (504 bits), 65 bytes captured (504 bits) on interface 'UdeviceWPF_Loopback', id 0	65	TCP	127.0.0.1	131.698400 4449		
Seq=1 Ack=16 Win=65280 Len=19 [PSH, ACK] 65412 → 12345 63	65					
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 131.698400, Seq: 1, Len: 19						
Transmission Control Protocol, Src Port: 65412, Dst Port: 12345, Seq: 1, Len: 19						
Data (19 bytes)						

ב. בצלום מעלה ניתן לראות את תגובה השרת. ב Frame 4449 נשלחת חבילה PSH, ACK מהשרת בחזרה ללקוח (Len=19). השרת מאשר את הבקשה ושולח { "type": "SIN/ACK" }.

קבלת הودעה זו בצד הלקוח מאשרת שהשרת פעיל, מאמין, ומוכן לעبور לשלב הבא של תיאום גודל ההודעה.

Info	Length	Protocol	Destination	Source	Time	No.
Frame 4449: Packet, 65 bytes on wire (504 bits), 65 bytes captured (504 bits) on interface 'UdeviceWPF_Loopback', id 0	65	TCP	127.0.0.1	131.698400 4449		
Seq=1 Ack=16 Win=65280 Len=19 [PSH, ACK] 65412 → 12345 63	65					
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 131.698400, Seq: 1, Len: 19						
Transmission Control Protocol, Src Port: 65412, Dst Port: 12345, Seq: 1, Len: 19						
Data (19 bytes)						

Seq#	Ack#	Wlan	Lens#	[PSH, ACK]	Info Length	Protocol	Destination	Source	Time	No.
82	80 00 00 00 45 00 00 37	c3	ee 40 00 00 00 00 00	... E 7 @ ...	59	TCP	127.0.0.1	131.698481 4451		

ג. בצלום לעיל ניתן לראות את השלב האחרון בתהליך הקמת הקשר. ב Frame 4451 הלקוק שולח הודעה (Len=15) {"type": "ACK"} (Len=15). הודהה זו מהווע אישור סופי מצד הלקוק שהוא קיבל את תשובה שלשתה. כעת, אחרי שהשתיים תחליר ה握手 המשולש ברמת האפליקציה, החיבור נחשב מבוסס ושני הצדדים עוברים מידית לשלב הבא, שהוא שליחת בקשה לקבלת גודל ההודהה המקסימלי.

3. קביעת גודל הודעה מקסימלי

א. בשלב הבא מתבצע תהליכי תיאום הפרמטרים בין הלקוק לשרת. הלקוק שולח בקשה לקבלת גודל ההודהה המקסימלי, והשרת משביע עם הערכים המוגדרים עצמם.

Seq#	Ack#	Wlan	Lens#	[PSH, ACK]	Info Length	Protocol	Destination	Source	Time	No.
82	80 00 00 00 45 00 00 61	c3	30 40 00 00 00 00 00	... E a @ ...	180	TCP	127.0.0.1	131.698481 4451		

בצלום מעלה ניתן לראות את הקמת הבקשה על ידי הלקוק. ב Frame 4453 הלקוק שולח הודעה (Len=101) {"type": "GET_MAX_MSG_SIZE"} (Len=101). בשלב ה-Data-Request ניתן לראות את הפקודה שנשלחה: {"type": "GET_MAX_MSG_SIZE"} (Len=101). הודהה זו מבקשת מהשרת לשלוח את הגדרות התעבורה שלו לעבוד בקצב דינמי (MSS) והאם עבודות בקצב דינמי).

Seq#	Ack#	Wlan	Lens#	[PSH, ACK]	Info Length	Protocol	Destination	Source	Time	No.
82	80 00 00 00 45 00 00 78	c3	b2 40 00 00 00 00 00	... E x @ ...	992	TCP	127.0.0.1	131.698481 4451		

ב. בצלום מעלה ניתן לראות את תשובה השרת לבקשה: ב Frame 4455 הלקוק שולח הודעה (Len=80) {"type": "MAX_MSG_SIZE": 400, "maximum message size": 400, "dynamic message size": false} (Len=80). בשלב ה-Data-Response ניתן לראות את ה-JSON שחרז: {"type": "MAX_MSG_SIZE", "maximum message size": 400, "dynamic message size": false}. השרת הגידר שהגודל המקסימלי לכל הודעה יהיה 400 בתים. הלקוק קולט את המידע זהה, שומר אותו, ומשתמש בו כעט כדי לחלק את הקובץ שהוא רוצה לשלוח למקטעים שלא גדולים מ-400 בתים.

4. חלוקת הודעה למקטעים

לאחר קבלת גודל ההודהה המקסימלי (בשלב הקודם), הלקוק מחילק את ההודהה למקטעים קטנים יותר - בהתאם לגודל שהוגדר (למשל 400 בתים לפחות). כל מקטע מקבל מספר רצף ועקב ומועבר כהודהה נפרדת. החלוקה מתבצעת ברמת שכבת היישום.

Seq#	Ack#	Wlan	Lens#	[PSH, ACK]	Info Length	Protocol	Destination	Source	Time	No.
82	80 00 00 00 45 00 00 80	c3	24 40 00 00 00 00 00	... E @ ...	3992	TCP	127.0.0.1	131.698481 4451		

בצלום מעלה ניתן לראות את הקמת הבקשה על ידי הלקות. ב-frame 4457 נשלחת חביתה PSH, ACK מהלקות לשרת (Len=455). בחולונית ה-Data ניתן לראות את תוכן ה-JSON.

```
{"...type": "DATA", "seq": 0, "payload": "This is a test message"}
```

ניתן לראות שהלקות הקצה למקטע זה את מספר הרצף 0 (0) וצירוף חלק מהטקסט המקורי. החלוקת והמספור מתרכזים על ידי הקוד בשכבה היישום, לפני שהמידע מועבר לTCP לשילוחה.

5. שליחה באמצעות Sliding Window הלקות משתמש במנגנון Sliding Window, המאפשר שליחת מקטעים ברצף על בסיס גודל החלון המוגדר.

מנגנון זה מנהל את קצב השילחה ומבודד שהלקות לא מציף את השירות, תוך כדי מעקב אחריו אישוריהם (ACKs) הגיעו מהשרת.

Seq=88 Ack=100 Win=65280 Len=455 [PSH, ACK] 12345 → 65412 499	TCP	127.0.0.1	127.0.0.1	131.698902 4457
Seq=100 Ack=543 Win=64768 Len=23 [PSH, ACK] 65412 → 12345 67	TCP	127.0.0.1	127.0.0.1	131.699005 4459
Seq=543 Ack=123 Win=65280 Len=449 [PSH, ACK] 12345 → 65412 493	TCP	127.0.0.1	127.0.0.1	131.699097 4461

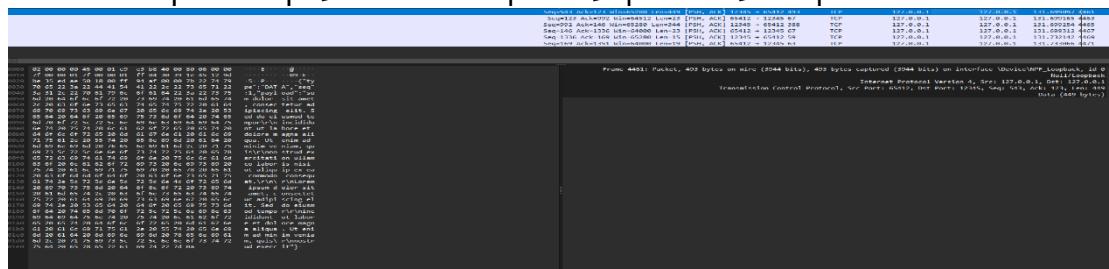
בצלום לעיל ניתן לראות את רצף העברת הנתונים והאישורים: Frame 4457: הלקות שולח את המקטע הראשון (M0) עם מידע (Len=455). Frame 4459: השירות שולח אישור (ACK) מיידי (Len=23)._frame 4461: אישור Ack=543 מעיד על כך שהשרת קיבל את המידע ותוכן-h-JSON שלו תקין. Frame 4461: הלקות שולח את המקטע השני (M1) עם מידע (Len=449). ניתן לראות מספרי הרצף עליים (השורה הראשונה מסומנת כ-Seq=88 והשלישית כ-Seq=543) והתעבורה זורמת בצורה של שליחת וקבלת. מכיוון שההרכזה מתבצעת על מחשב מקומי (localhost), האישורים מגיעים מהר מאוד, אך הפורטוקול בלקות בניו להמשיך ולשלוח את המקטעים הבאים בחלון גם ללא המתנה, כל עוד גודל החalon מאפשר זאת.

6. ACK מצטבר מהשרת

השרות שולח ACK מצטבר, המציג את מספר הרצף הגבוה ביותר שהתקבל בצורה רציפה. אם מתקבל מקטע מחוץ לסדר, השירות אינו מקדם את האישור עד להשלמת הרצף החסר. טיפול במקרה קצה (לפני קבלת M0):

בתחילת העברת הנתונים, לפני שהתקבל המקטע הראשון (M0), מצב השירות הוא שעדין לא התקבל אף מקטע ברצף. לכן, במקרה שבו מתקבל מקטע מחוץ לסדר או מקטע לא תקין לפני קבלת M0, השירות מחזיר ACK מצטבר בתור ערך -1.

ערך זה מייצג שטרם התקבל אף מקטע רציף החל מ-0. לאחר קבלת המקטעים הסדריים, ערך ה-ACK מתקדם בהתאם למספר הרצף שהתקבלו. התננוגות זו מאפשרת ללקות לדעת אילו מקטעים התקבלו בהצלחה ולעדכן את החalon שלו בהתאם.



בצלום מעלה ניתן לראות את הלקות שולח הודעה מידע (DATA).

הודעת מידע עם מספר רצף 1: seq. ניתן לראות בתוך חלונית המידע שהפרוטוקול שלנו שולח את הנתונים בפורמט JSON, בנפרד מכותרות ה-TCP הרגילות.

```

Sep-123: Ack=992 Win=65432 Len=31 [PSH, ACK] Seq=1345 Len=31 TCP 127.0.0.1 127.0.0.1 131.699140 4463
Sep-992: Ack=1346 Win=65208 Len=344 [PSH, ACK] Seq=1345 + 65432 - 188 TCP 127.0.0.1 127.0.0.1 131.699254 4465
Seq=140: Ack=1346 Win=64000 Len=23 [PSH, ACK] Seq=1342 + 12345 - 67 TCP 127.0.0.1 127.0.0.1 131.699312 4467
Seq=140: Ack=1346 Win=65208 Len=23 [PSH, ACK] Seq=1342 + 12345 - 69 TCP 127.0.0.1 127.0.0.1 131.699312 4468
Seq=169: Ack=1346 Win=65208 Len=23 [PSH, ACK] Seq=1342 + 12345 - 71 TCP 127.0.0.1 127.0.0.1 131.699312 4469
Seq=169: Ack=1346 Win=65208 Len=23 [PSH, ACK] Seq=1342 + 12345 - 63 TCP 127.0.0.1 127.0.0.1 131.737300 4473

Frame 4463: Packet, 67 bytes on wire (536 bits), 67 bytes captured (536 bits) on interface \Device\PFN_Lightning, id 0x0
[Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1, Internet Control Message Protocol, Src Port: 65432, Dst Port: 65432, Seq: 1345, Len: 23]
Data (23 bytes)

```

בצילום זה ניתן לראות את ה ACK המצביע על השרת שולח בחרזה .
בתוך ה Data-של החבילה (שורה 4463).
ניתן לראות בברור את ה { 1: {"type": "ACK", "ack": 1 } }.
הערך ack: 1 מאשר ללקוח שהודעה מסטר 1 התקבלה ועובדת בהצלחה על ידי האפליקציה, והרצף
תקין עד לנקודה זו.
וכעת הליקון יודיע לנו את ה צהיר על המשיך לשולח את ההודעות הבאות.

7. שימוש בחיבור TCP עקבי

לאורך כל התקשרות נעשה שימוש בחיבור TCP יחיד שנשאר פתוח. החיבור נפתח בתחילת הריצה בשלב ה Handshake , ונסגר רק לאחר סיום העברת כל הנתונים, ללא פתיחה מחדש של חיבורים נוספים עבור כל הודעה.

גישה זו מאפשרת תקשורת עיליה יותר וחוסכת פטיחה ופתיחת חזרת של חיבור TCP.

בצילום זה ניתן לראות את סיום התקשרות.
לאחר שכל מקטיע הנתונים הועבר ואושרו, נשלחת הודעה סיום.
בחולונית ה Data ניתן לראות את הפקודה : { "type": "FIN" } .
עובדת שהודעה זו נשלחת באותו פורט ובאותו רצף (מיד אחרי המידע האחרון), מוכיחה שהמערכת
שומרה על Socket פתוח לכל אורך התהילה, וסגרה אותו בצורה מבוקרת, רק כאשר האפליקציה
סיימה את תפקידה.

ניתן לראות את זה בצילום מטה - Frame 4471: השרת מגיב בהודעת {"type": "FIN_ACK"} ומס'ים את התחשרות.

(Retransmission ושליטה מחדש Timeout .8

מקרים אלו מתרחשים רק אם יש אובדן / עיכוב.

במהלך הקלטה "1" לא היו אירועי Timeout או Retransmission, מאחר שככל המקטעים התקבלו והאישורים חזרו בזמן.

עם זאת, הקוד ממש מגונן `Timeout` ושליחה מחדש, שMOVED במקורה שבו ACK אינו מתקבל בזמן שהוגדר.

בדיקות הרצה:

1. בדיקת הרצה - לטובת בדיקה חוזרת ויעילה של התקשרות בין השרת ללקוח, נعزيزת במנוע AI של GPT, **הקיים** מצורף.

<https://chatgpt.com/share/694d499f-ac68-8013-a043-e0267c799c6a>

מצורף צילום מסך של הסקריפט:

```
bat
Copy code

@echo off

REM Start server in a new CMD window
start cmd /k python server.py

REM Wait ~1 second for the server to initialize
timeout /t 1 >nul

REM Start client in a new CMD window
start cmd /k python client.py
```

מצורפת התוצאה לאחר הפעלת הסקריפט:

```
C:\WINDOWS\system32\cmd. x + v
Starting transfer. Window Size: 4, Initial MWS: 400
[SEND] Seq 0 (len: 400)
[SEND] Seq 1 (len: 400)
[SEND] Seq 2 (len: 305)
[ACK] Cumulative up to 0
[ACK] Cumulative up to 1
[ACK] Cumulative up to 2
Transfer Complete. FIN_ACK received.

C:\Users\user\OneDrive - Ariel University\��ססס - תרגום הוחזר\��ססס - תרגום הוחזר\Assignment 3>
The server is ready to receive on localhost:12345
--- FINAL MESSAGE ---
This is a test message for the Transport Layer assignment.

The purpose of this file is to verify that the reliable protocol implementation
correctly performs segmentation, sliding window transmission, cumulative ACKs,
timeout handling, and retransmissions.

This message is intentionally long enough to exceed the maximum message size
and to require multiple transmission windows.

Orel salem test Orel salem test Orel salem test Orel salem test Orel salem test
Orel salem test Orel salem test Orel salem test Orel salem test Orel salem test
Orel salem test Orel salem test Orel salem test Orel salem test Orel salem test
Orel salem test Orel salem test Orel salem test Orel salem test Orel salem test
Orel salem test Orel salem test Orel salem test Orel salem test Orel salem test
Orel salem test Orel salem test Orel salem test Orel salem test Orel salem test
Orel salem test Orel salem test Orel salem test Orel salem test Orel salem test
Orel salem test Orel salem test Orel salem test Orel salem test Orel salem test
Orel salem test Orel salem test Orel salem test Orel salem test Orel salem test
Orel salem test Orel salem test Orel salem test Orel salem test Orel salem test
```

בצלום זה, ניתן לראות כי הלקוח מחלק את הודעה למקטעים בהתאם לגודל המקסימלי שנקבע על ידי השרת (maximum message size), ומשדר אותם באמצעות מנגנון Sliding Window בגודל שהוגדר.

כל מקטע נשלח עם מספר רצף עוקב החל מ-0, והשרת שולח ACK מצטבר המציין את מספר הרצף הגבוה ביותר שהתקבל בצורה ריצפה. לאחר קבלת כל המקטעים, השרת מבצע איזוד (reassembly) של הנתונים ומדפיס את הודעה המלאה, תהליך העברת הסטיים בהצלחה.