

Assignment 3 - 208748368

הקדמה :

כמה מילים שלי. ניסיתי 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 - בדיקת 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.

החיבור נפתח פעם אחת ומשמש להמשך כל התקשורת בין הלקוח לשרת.

שלב זה מאפשר העברת נתונים בצורה אמינה ומסודרת, ומהווה את הבסיס לכל ההודעות שנשלחות בהמשך.

Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM [SYN] 12345 + 65412 56	TCP	127.0.0.1	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	127.0.0.1	131.697903 4445
Seq=1 Ack=1 Win=65280 Len=0 [ACK] 12345 + 65412 44	TCP	127.0.0.1	127.0.0.1	131.698016 4446

בצילום ניתן לראות שהלקוח שולח חבילת SYN לצורך יזום החיבור.

לאחר מכן השרת מגיב בחבילת SYN, ACK שהיא מאשרת את קבלת הבקשה.

לבסוף הלקוח שולח חבילת ACK ובכך החיבור בין הלקוח לשרת הוקם בהצלחה.

* הצילום לקוח מהקלטה 1

2. Handshake ברמת שכבת היישום

לאחר פתיחת חיבור ה TCP מתבצע Handshake ברמת שכבת היישום.

בשלב זה הלקוח והשרת מחליפים הודעות JSON ייעודיות שמטרתן לוודא ששני הצדדים מסונכרנים ומוכנים להתחיל בהעברת נתונים.

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

Time	Source	Destination	Protocol	Length	Info
0.000000	127.0.0.1	127.0.0.1	TCP	56	Seq=1 Ack=1 Win=65280 Len=0 [PSH, ACK] 65412 + 12345 56

Time	Source	Destination	Protocol	Length	Info
0.000000	127.0.0.1	127.0.0.1	TCP	56	Seq=1 Ack=1 Win=65280 Len=0 [PSH, ACK] 65412 + 12345 56

א. בצילום מעלה ניתן לראות את תחילת התהליך בשכבת היישום.

ב Frame 4447 נשלחת חבילת PSH, ACK מהלקוח לשרת (Len=15).

בחלונית ה Data-למטה, ניתן לראות בבירור שהלקוח שולח את

הפקודה {"type": "SIN"}.

וזוהי בקשת ההתחברות הראשונית שמוגדרת בפרוטוקול שלנו, והיא מסמנת לשרת

שהלקוח מעוניין להתחיל בתקשורת.

Time	Source	Destination	Protocol	Length	Info
0.000000	127.0.0.1	127.0.0.1	TCP	63	Seq=1 Ack=16 Win=65280 Len=15 [PSH, ACK] 65412 + 12345 63

Time	Source	Destination	Protocol	Length	Info
0.000000	127.0.0.1	127.0.0.1	TCP	63	Seq=1 Ack=16 Win=65280 Len=15 [PSH, ACK] 65412 + 12345 63

ב. בצילום מעלה ניתן לראות את תגובת השרת.

ב Frame 4449 נשלחת חבילת PSH, ACK מהשרת בחזרה ללקוח (Len=19).

השרת מאשר את הבקשה ושולח {"type": "SIN/ACK"}.

קבלת ההודעה זו בצד הלקוח מאשרת שהשרת פעיל, מאזין, ומוכן לעבור לשלב הבא של

תיאום גודל ההודעה.

No.	Time	Source	Destination	Protocol	Length	Info
4451	0.000000	127.0.0.1	127.0.0.1	TCP	59	Seq=10 Ack=28 Win=65536 Len=15 (PSH, ACK) 12345 -> 65432 59
4452	0.000000	127.0.0.1	127.0.0.1	TCP	59	Seq=10 Ack=28 Win=65536 Len=15 (PSH, ACK) 12345 -> 65432 59
4453	0.000000	127.0.0.1	127.0.0.1	TCP	59	Seq=10 Ack=28 Win=65536 Len=15 (PSH, ACK) 12345 -> 65432 59
4454	0.000000	127.0.0.1	127.0.0.1	TCP	59	Seq=10 Ack=28 Win=65536 Len=15 (PSH, ACK) 12345 -> 65432 59
4455	0.000000	127.0.0.1	127.0.0.1	TCP	59	Seq=10 Ack=28 Win=65536 Len=15 (PSH, ACK) 12345 -> 65432 59

ג. בצילום לעיל ניתן לראות את השלב האחרון בתהליך הקמת הקשר. ב Frame 4451 הלקוח שולח הודעת (Len=15) {"type": "ACK"}.

הודעה זו מהווה אישור סופי מצד הלקוח שהוא קיבל את תשובת השרת. כעת, אחרי שהסתיים תהליך ה Handshake המשולש ברמת האפליקציה, החיבור נחשב מבוסס ושני הצדדים עוברים מיידית לשלב הבא, שהוא שליחת בקשה לקבלת גודל ההודעה המקסימלי.

3. קביעת גודל ההודעה מקסימלי

א. בשלב הבא מתבצע תהליך תיאום הפרמטרים בין הלקוח לשרת. הלקוח שולח בקשה לקבלת גודל ההודעה המקסימלי, והשרת משיב עם הערכים המוגדרים אצלו.

No.	Time	Source	Destination	Protocol	Length	Info
4453	0.000000	127.0.0.1	127.0.0.1	TCP	101	Seq=10 Ack=28 Win=65536 Len=101 (PSH, ACK) 65432 -> 12345 101
4454	0.000000	127.0.0.1	127.0.0.1	TCP	101	Seq=10 Ack=28 Win=65536 Len=101 (PSH, ACK) 65432 -> 12345 101
4455	0.000000	127.0.0.1	127.0.0.1	TCP	101	Seq=10 Ack=28 Win=65536 Len=101 (PSH, ACK) 65432 -> 12345 101
4456	0.000000	127.0.0.1	127.0.0.1	TCP	101	Seq=10 Ack=28 Win=65536 Len=101 (PSH, ACK) 65432 -> 12345 101
4457	0.000000	127.0.0.1	127.0.0.1	TCP	101	Seq=10 Ack=28 Win=65536 Len=101 (PSH, ACK) 65432 -> 12345 101

בצילום מעלה ניתן לראות את הקמת הבקשה על ידי הלקוח. ב Frame 4453 הלקוח שולח הודעת (Len=101) PSH, ACK.

בחלונית ה-Data ניתן לראות את הפקודה שנשלחת: {"type": "GET_MAX_MSG_SIZE", "...".}

הודעה זו מבקשת מהשרת לשלוח את הגדרות התעבורה שלו לעבוד במצב דינמי (MSS והאם לעבוד במצב דינמי).

No.	Time	Source	Destination	Protocol	Length	Info
4455	0.000000	127.0.0.1	127.0.0.1	TCP	80	Seq=10 Ack=28 Win=65536 Len=80 (PSH, ACK) 65432 -> 12345 80
4456	0.000000	127.0.0.1	127.0.0.1	TCP	80	Seq=10 Ack=28 Win=65536 Len=80 (PSH, ACK) 65432 -> 12345 80
4457	0.000000	127.0.0.1	127.0.0.1	TCP	80	Seq=10 Ack=28 Win=65536 Len=80 (PSH, ACK) 65432 -> 12345 80
4458	0.000000	127.0.0.1	127.0.0.1	TCP	80	Seq=10 Ack=28 Win=65536 Len=80 (PSH, ACK) 65432 -> 12345 80
4459	0.000000	127.0.0.1	127.0.0.1	TCP	80	Seq=10 Ack=28 Win=65536 Len=80 (PSH, ACK) 65432 -> 12345 80

ב. בצילום מעלה ניתן לראות את תשובת השרת לבקשה: ב Frame 4455 הלקוח שולח הודעת (Len=80) PSH, ACK בחזרה ללקוח.

בחלונית ה-Data ניתן לראות את ה JSON שחזר:

```
{
  "type": "MAX_MSG_SIZE",
  "maximum message size": 400,
  "dynamic message size": false
}
```

השרת הגדיר שהגודל המקסימלי לכל הודעה יהיה 400 בתים. הלקוח קולט את המידע הזה, שומר אותו, ומשתמש בו כעת כדי לחלק את הקובץ שהוא רוצה לשלוח למקטעים שלא גדולים מ400 בתים.

4. חלוקת ההודעה למקטעים

לאחר קבלת גודל ההודעה המקסימלי (בשלב הקודם), הלקוח מחלק את ההודעה למקטעים קטנים יותר - בהתאם לגודל שהוגדר (למשל 400 בתים ל Payload).

כל מקטע מקבל מספר רצף עוקב ומועבר כהודעה נפרדת. החלוקה מתבצעת ברמת שכבת היישום.

No.	Time	Source	Destination	Protocol	Length	Info
4457	0.000000	127.0.0.1	127.0.0.1	TCP	400	Seq=10 Ack=28 Win=65536 Len=400 (PSH, ACK) 65432 -> 12345 400
4458	0.000000	127.0.0.1	127.0.0.1	TCP	400	Seq=10 Ack=28 Win=65536 Len=400 (PSH, ACK) 65432 -> 12345 400
4459	0.000000	127.0.0.1	127.0.0.1	TCP	400	Seq=10 Ack=28 Win=65536 Len=400 (PSH, ACK) 65432 -> 12345 400
4460	0.000000	127.0.0.1	127.0.0.1	TCP	400	Seq=10 Ack=28 Win=65536 Len=400 (PSH, ACK) 65432 -> 12345 400
4461	0.000000	127.0.0.1	127.0.0.1	TCP	400	Seq=10 Ack=28 Win=65536 Len=400 (PSH, ACK) 65432 -> 12345 400

ב 4457 Frame נשלחת חבילת ACK, PSH, מהלקוח לשרת (Len=455).

בחלונית ה-Data ניתן לראות את תוכן ה-JSON .

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

ניתן לראות שהלקוח הקצה למקטע זה את מספר הרצף 0 ("seq": 0) וצירף חלק מהטקסט המקורי.

החלוקה והמספור מתבצעים על ידי הקוד בשכבת היישום, לפני שהמידע מועבר ל TCP לשליחה.

5. שליחה באמצעות Sliding Window

הלקוח משתמש במנגנון Sliding Window, המאפשר שליחת מקטעים ברצף על בסיס גודל החלון המוגדר.

מנגנון זה מנהל את קצב השליחה ומוודא שהלקוח לא מציף את השרת, תוך כדי מעקב אחרי אישורים (ACKs) המגיעים 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

בצילום לעיל ניתן לראות את רצף העברת הנתונים והאישורים:

Frame 4457: הלקוח שולח את המקטע הראשון (M0) עם מידע (Len=455).

Frame 4459: השרת שולח אישור (ACK) מייד (Len=23).

האישור Ack=543 מעיד על כך שהשרת קיבל את המידע ותוכן ה-JSON שלו תקין.

Frame 4461: הלקוח שולח את המקטע השני (M1) עם מידע (Len=449).

ניתן לראות שמספרי הרצף עולים (השורה הראשונה מסומנת כ-Seq=88 והשלישית כ-Seq=543) והתעבורה זורמת בצורה של שליחה וקבלה.

מכיוון שההרצה מתבצעת על מחשב מקומי (Localhost), האישורים מגיעים מהר מאוד, אך הפרוטוקול בלקוח בנוי להמשיך ולשלוח את המקטעים הבאים בחלון גם ללא המתנה, כל עוד גודל החלון מאפשר זאת.

6. ACK מצטבר מהשרת

השרת שולח ACK מצטבר, המציין את מספר הרצף הגבוה ביותר שהתקבל בצורה רציפה.

אם מתקבל מקטע מחוץ לסדר, השרת אינו מקדם את האישור עד להשלמת הרצף החסר.
טיפול במקרי קצה (לפני קבלת M0):

בתחילת העברת הנתונים, לפני שהתקבל המקטע הראשון (M_0), מצב השרת הוא שעדיין לא התקבל אף מקטע ברצף.

לכן, במקרה שבו מתקבל מקטע מחוץ לסדר או מקטע לא תקין לפני קבלת M0, השרת מחזיר ACK מצטבר בתור ערך 1-.

ערך זה מייצג שטרם התקבל אף מקטע רציף החל מ-0.

לאחר קבלת המקטעים הסדירים, ערך ה-ACK מתקדם בהתאם למספרי הרצף שהתקבלו.

התנהגות זו מאפשרת ללקוח לדעת אילו מקטעים התקבלו בהצלחה ולעדכן את החלון שלו בהתאם.

[illegible]

בצילום מעלה ניתן לראות את הלקוח שולח הודעת 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.

Seq=123	Ack=992	Win=65536	Len=13	[PSH, ACK]	65412	→	12345	63	TCP	127.0.0.1	127.0.0.1	131.093162 4463
Seq=992	Ack=146	Win=65536	Len=344	[PSH, ACK]	12345	→	65412	388	TCP	127.0.0.1	127.0.0.1	131.093164 4465
Seq=146	Ack=1336	Win=64000	Len=23	[PSH, ACK]	65412	→	12345	67	TCP	127.0.0.1	127.0.0.1	131.093162 4467
Seq=1336	Ack=169	Win=65536	Len=23	[PSH, ACK]	12345	→	65412	59	TCP	127.0.0.1	127.0.0.1	131.732162 4469
Seq=169	Ack=1351	Win=64000	Len=19	[PSH, ACK]	65412	→	12345	63	TCP	127.0.0.1	127.0.0.1	131.733066 4471

0000	02 00 00 00 45 00 00 3f	a3 c8 40 00 00 00 00 00	...	E 7	0	...
0008	7f 00 00 01 7f 00 00 01	30 39 ff 64 ba 35 ad dc	...	09	5	...
0010	3e 45 15 45 20 18 00 ff	64 f9 00 00 7b 22 74 79	E p
0014	70 65 22 3a 22 41 43 4b	22 2c 22 03 63 6b 22 3a	pm	"ACK ", "ack"
0018	33 7d 0a		

Frame 4463: Packet, 67 bytes on wire (536 bits), 67 bytes captured (536 bits) on interface \Device\NPF_{...} id 0
Null/Loopback
Internet Protocol Version 4, Src: 127.0.0.1, Dest: 127.0.0.1
Transmission Control Protocol, Src Port: 12345, Dst Port: 65412, Seq: 12345, Ack: 992, Len: 13
Data (13 bytes):

בצילום זה ניתן לראות את ה ACK המצטבר שהשרת שולח בחזרה .
בתוך ה Data-של החבילה (שורה 4463).
ניתן לראות בבירור את ה `JSON: {"type": "ACK", "ack": 1}` .
הערך 1: ack מאשר ללקוח שהודעה מספר 1 התקבלה ועובדה בהצלחה על ידי האפליקציה, והרצף תקין עד לנקודה זו.
וכעת הלקוח יודע שניתן להזיז את החלון ולהמשיך לשלוח את ההודעות הבאות.

7. שימוש בחיבור TCP עקבי

לאורך כל התקשורת 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 יעילה יותר וחוסכת פתיחה וסגירה חוזרת של חיבורי TCP.

Seq=1336	Ack=169	Win=65536	Len=15	[PSH, ACK]	12345	→	65412	59	TCP	127.0.0.1	127.0.0.1	131.732162 4469
----------	---------	-----------	--------	------------	-------	---	-------	----	-----	-----------	-----------	-----------------

0000	02 00 00 00 45 00 00 3f	a3 c8 40 00 00 00 00 00	...	E 7	0	...
0008	7f 00 00 01 7f 00 00 01	ff 64 30 39 3e 45 15 b6	...	09	5	...
0010	3e 35 ad 61 60 18 00 ff	60 a3 00 00 7b 22 74 79	S p
0014	70 65 22 3a 22 46 49 4e	22 7d 0a	pm	"FIN "

Frame 4469: Packet, 59 bytes on wire (472 bits), 59 bytes captured (472 bits) on interface \Device\NPF_{...} id 0
Null/Loopback
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
Transmission Control Protocol, Src Port: 65412, Dst Port: 12345, Seq: 1336, Ack: 169, Len: 15
Data (15 bytes):

בצילום זה ניתן לראות את סיום התקשורת.
לאחר שכל מקטעי הנתונים הועברו ואושרו, נשלחת הודעת סיום.
בחלונית ה Data ניתן לראות את הפקודה: `{"type": "FIN"}`.
עובדה שהודעה זו נשלחת באותו פורט ובאותו רצף (מיד אחרי המידע האחרון), מוכיחה שהמערכת שמרה על ה Socket פתוח לכל אורך התהליך, וסגרה אותו בצורה מבוקרת, רק כאשר האפליקציה סיימה את תפקידה.
ניתן לראות את זה בצילום מטה - Frame 4471: השרת מגיב בהודעת `{"type": "FIN_ACK"}` ומסיים את ההתקשורת.

Seq=169	Ack=1351	Win=64000	Len=19	[PSH, ACK]	65412	→	12345	63	TCP	127.0.0.1	127.0.0.1	131.733066 4471
---------	----------	-----------	--------	------------	-------	---	-------	----	-----	-----------	-----------	-----------------

0000	02 00 00 00 45 00 00 3f	a3 c2 40 00 00 00 00 00	...	E 7	0	...
0008	7f 00 00 01 7f 00 00 01	30 39 ff 64 ba 35 ad dc	...	09	5	...
0010	3e 45 15 45 20 18 00 ff	a8 00 00 00 7b 22 74 79	E p
0014	70 65 22 3a 22 46 49 4e	5f 41 43 4b 22 7d 0a	pm	"FIN_ACK"

Frame 4471: Packet, 63 bytes on wire (504 bits), 63 bytes captured (504 bits) on interface \Device\NPF_{...} id 0
Null/Loopback
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
Transmission Control Protocol, Src Port: 12345, Dst Port: 65412, Seq: 169, Ack: 1351, Len: 19
Data (19 bytes):

8. Timeout ושליחה מחדש (Retransmission)

מקרים אלו מתרחשים רק אם יש אובדן / עיכוב.
במהלך הקלטה "1" לא היו אירועי Timeout או Retransmission, מאחר שכל המקטעים התקבלו והאישורים חזרו בזמן.
עם זאת, הקוד מממש מנגנון Timeout ושליחה מחדש, שמופעל במקרה שבו ACK אינו מתקבל בזמן שהוגדר.

בדיקת הרצה:

1. בדיקת הרצה - לטובת בדיקה חוזרת ויעילה של התקשורת בין השרת ללקוח, נעזרתי במנוע AI של GPT, **הקישור** מצורף.

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

מצורף צילום מסך של הסקריפט:

```
bat
@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.exe X + -  
Starting transfer. Window Size: 4, Initial MSS: 400  
[SEND] Seq 0 (len: 400)  
[SEND] Seq 1 (len: 400)  
[SEND] Seq 2 (len: 400)  
[ACK] Cumulative up to 0  
[ACK] Cumulative up to 1  
[ACK] Cumulative up to 2  
Transfer Complete. FIN_ACK received.  
  
C:\Users\UserOne> dir C:\Users\UserOne\Documents\Assignment3\3 Assignment3\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
```

בצילום זה, ניתן לראות כי הלקוח מחלק את ההודעה 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 רציפה.

לאחר קבלת כל המקטעים, השרת מבצע איחוד (reassembly) של הנתונים ומדפיס את ההודעה המלאה, תהליך ההעברה הסתיים בהצלחה.