<u>פרויקט סיום– רשתות תקשורת</u>

מגישות:

- **אופק כץ**
- **מוריה אסתר אוחיון**

<u>תוכן עניינים:</u>

- מטרת המטלה
- הסבר כללי על הDHCP server, DNS server וAPP
- הסבר על הרצת התוכנית
- דיאגרמת מצבים של המערכת
- פירוט DHCP server
- פירוט DNS server
- פירוט APP
- מערכת באיבוד פקטות
- תשובות לשאלות
- ביבליוגרפיה

סרטון הרצה מצורף בשם "network_final.mp4"

מטרת המטלה:

במטלה זו הקמנו לקוח ושרתים שונים אליהם הוא ניגש.

תחילה הלקוח ניגש לDHCP לקבל את נתוני הרשת.

לאחר מכן הלקוח ניגש לDNS לקבלת הIP של האפליקציה.

לבסוף הלקוח ניגש אל האפליקציה.

האפליקציה שאנו בחרנו הינה http redirect- הקמנו שרת APP שמהווה כ"proxy" - הלקוח ניגש לAPP לבקשת תמונה, הAPP ניגש לSERVER לבקשת נתיב התמונה, הAPP מקבל את הנתיב ושולח את התמונה ללקוח.

הסבר כללי

DHCP server:

שרת DHCP אחראי על הקצאת כתובות IP והעברת נתוני הרשת ללקוחות.

השרת שאנו יצרנו עובד מעל UDP כאשר טווח הIPים אותם הוא יכול לספק הינה:
10.0.0.17-10.0.0.254 (במידה והגענו לסוף הטווח הDHCP יחזיר IP "0.0.0.0")
השרת מחזיק רשימת IPים תפוסה ובהגעת לקוח חדש בודק IP שאינו ברשימה, שולח ללקוח ומכניס את אותו הIP לרשימה בכדי להימנע משליחת אותו IP ללקוחות שונים, בנוסף שולח ללקוח את הIP של הDNS.

סה"כ מועברות 4 פקטות: לקוח לDHCP - DISCOVER
DHCP ללקוח – OFFER
לקוח לDHCP - REQUEST
DHCP ללקוח – ACK

DNS server:

שרת DNS אחראי על התאמת דומיין לIP.

השרת שאנו יצרנו עובד מעל UDP , השרת מכיל מילון עם key- דומיין, value- IP.

לאחר בקשת לקוח, הDNS בודק האם הדומיין נמצא במילון -
במידה וכן- יחזיר ללקוח את המידע
במידה ולא- הDNS יחזיר "0.0.0.0"

סה"כ מועברות 2 פקטות: לקוח לDNS – DNS query
DNS ללקוח – DNS response

APP:

האפליקציה שאנחנו בחרנו הינה http redirect.

המימוש שלנו- האפליקציה היא "proxy" – הלקוח ניגש לAPP עם בקשה לתמונה, הAPP ניגש לSERVER עם בקשה לנתיב התמונה, הSERVER מעביר לAPP את הנתיב, והAPP מעביר ללקוח את התמונה.

החיבור בין הלקוח לAPP ממומש ע"י rudp או TCP לפי בחירת המשתמש. החיבור בין הAPP לSERVER ממומש ע"י TCP.

<u>הסבר על הרצת התוכנית:</u>

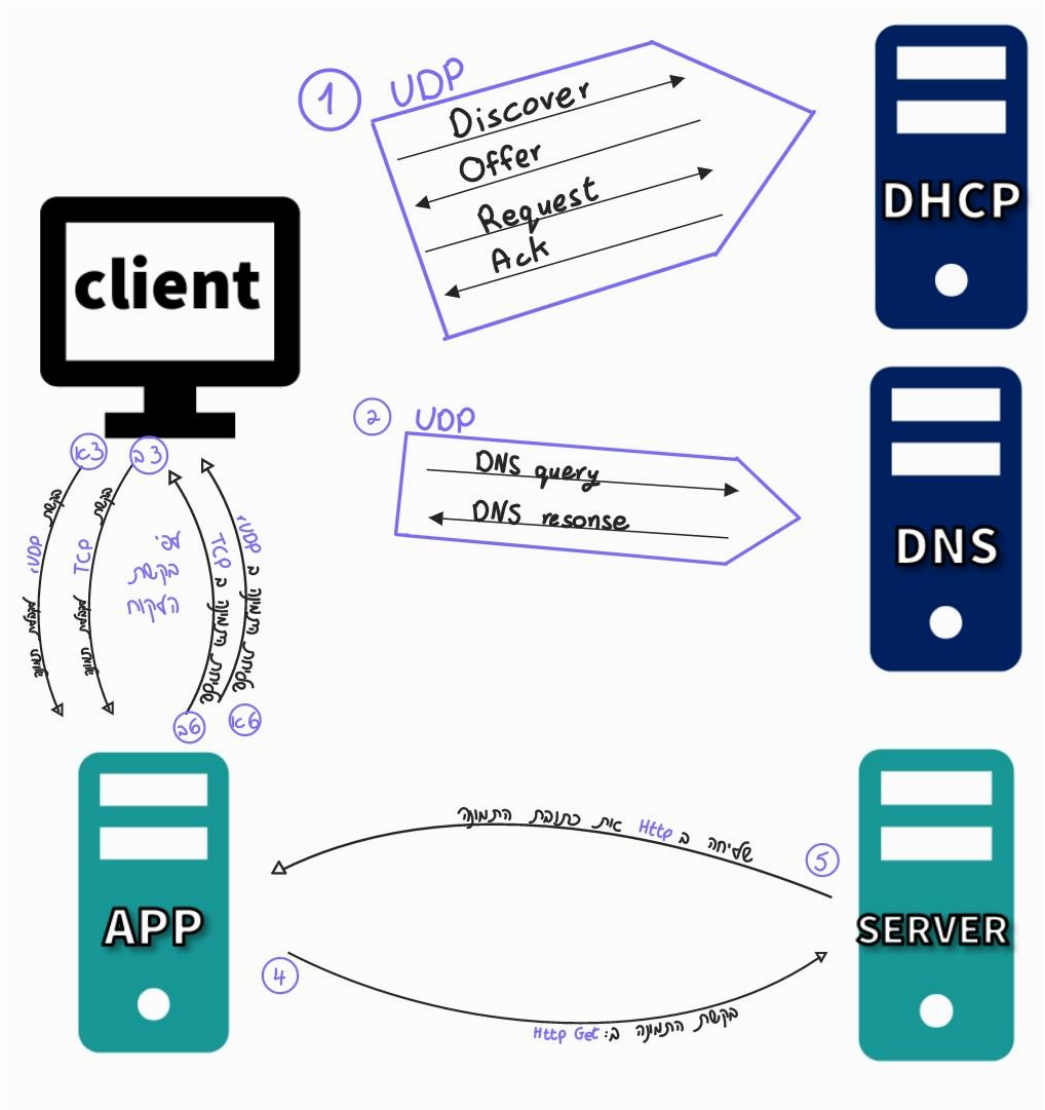
<u>חלק א'- נדרש:</u>

1. קובץ DHCP.py
2. קובץ DNS.py
3. קובץ APP.py
4. קובץ SERVER.py
5. קובץ client.py
6. image_TCP.jpg
7. image_UDP.jpg

<u>הרצת התוכנית:</u>

1. הקוד רץ על סביבת עבודה ubuntu
2. פתיחת 5 טרמינלים שונים (בתיקייה בה נמצאים הקבצים הדרושים)
3. בכל טרמינל נריץ קובץ אחר: **sudo python3 "****"**
(כאשר "****" מייצג שם הקבצים מ1-5)
4. **את הלקוח יש להריץ אחרון (client.py)!**
5. לאחר שהלקוח קיבל את הip של הAPP הוא יבקש מהמשתמש להכניס T בשביל tcp וI בשביל rudp
6. לאחר קבלת התמונה, הלקוח יבקש מהמשתמש להכניס C בשביל לחזור לפעולה 5 או E ליציאה
7. לאחר שהלקוח נסגר ניתן לסגור את השרתים ב ctrl c

דיאגרמת מצבים של המערכת



DHCP

שימוש בספריית scapy וחיבור UDP

<u>צד לקוח:</u>

א. בונה פקטת discover
- נשלח בברודקאסט מ0.0.0.0 ל255.255.255.255.
- יוצא מפורט 68 ונשלח לפורט 67 (הפורטים של בקשות DHCP)
- סוג הפקטה היא discover (1)

ב. שולח בעזרת פונקציית ()sendp

ג. ממתין לקבלת פקטת offer משרת הDHCP בעזרת פונקציית ()sniff

ד. שולח את הפקטה שקיבל לפונקציה ()dhcp_request

ה. מוציא מתוך הפקטה את הip של הclient ואת הip של שרת הDNS

ו. אם הip שהקליינט קיבל הוא '0.0.0.0' נגמרו לDHCP הIP הפנויים והוא יוציא את הclient מהתוכנית.

ז. בונה פקטת request
- נשלח בברודקאסט מ0.0.0.0 ל255.255.255.255.
- יוצא מפורט 68 ונשלח לפורט 67 (הפורטים של בקשות DHCP)
- מאשר את כתובת הip שהDHCP סיפק בyiadder
- סוג הפקטה היא request (3)

ח. שולח בעזרת פונקציית ()sendp

ט. ממתין לקבלת פקטת ack משרת הDHCP בעזרת פונקציית ()sniff

<u>צד שרת:</u>

א. מסניף פקטות שמגיעות לפורט 67, הפורט של שרת DHCP

ב. מכין רשימת ip תפוסים

ג. בודק שהפקטה שהגיעה מהלקוח היא אכן discover

ד. מכין את הip הפנוי הבא בפורמט 10.0.0.17-254

ה. בונה פקטת offer
- נשלח בברודקאסט מ0.0.0.0 ל255.255.255.255.
- יוצא מפורט 67 ונשלח לפורט 68 (הפורטים של DHCP)
- מספק את הכתובת הip ללקוח תחת yiaddr ואת כתובת הip של שרת הDNS תחת name-server
- הip ינתן ל86400 שניות (ליום שלם)
- סוג הפקטה היא offer (2)

ו. שולח בעזרת פונקציית ()sendp

ז. ממתין לקבלת פקטת request מהלקוח בעזרת פונקציית sniff()

ח. שולח את הפקטה שקיבל לפונקציה dhcp_ack()

ט. מוסיף את הIp שהלקוח מבקש לרשימת הip התפוסים

י. בונה פקטת ack לאישור סופי

- נשלח בברודקאסט מ0.0.0.0 ל255.255.255.255.
- יוצא מפורט 67 ונשלח לפורט 68 (הפורטים של DHCP)
- סוג הפקטה היא ack (3)

יא. חוזר לסעיף א

צילומי מסך

Wireshark: (מהקלטה DHCP.pcap)

1	0.000000000	0.0.0.0	255.255.255.255	DHCP	288	DHCP Discover - Transaction ID 0x11111111
2	1.023492861	10.0.0.17	255.255.255.255	DHCP	312	DHCP Offer - Transaction ID 0x11111111
3	2.047691528	0.0.0.0	255.255.255.255	DHCP	288	DHCP Request - Transaction ID 0x11111111
4	3.083016116	10.0.0.17	255.255.255.255	DHCP	306	DHCP ACK - Transaction ID 0x11111111

צד לקוח:

```
to DHCP
.
Sent 1 packets.
sent DISCOVER
got OFFER
dns: 10.0.0.18
client: 10.0.0.20
.
Sent 1 packets.
sent REQUEST
got ack
done with DHCP
```

צד שרת:

```
DHCP on
got DISCOVER
ip:  10.0.0.17 is already in use
ip:  10.0.0.18 is already in use
.
Sent 1 packets.
send OFFER
got REQUEST
IP appreved, DHCP append the IP to list
['10.0.0.17', '10.0.0.18', '10.0.0.10', '10.0.0.19']
.
Sent 1 packets.
send ACK
done with client
```

DNS

שימוש בספריית scapy וחיבור UDP

צד לקוח:

1. בונה פקטת query לבקשת הip של www.myApp.com
   - שולח מהIP שקיבל מהDHCP לIP של הDNS (גם כן קיבל מהDHCP) מפורט 20308 (308-ספרות אחרונות בתז של מוריה אסתר) לפורט 53
   - דגל rd דלוק (במידה ולDNS אין את המידע שיבדוק בצורה רקורסיבית ויחזיר תשובה)
   - qd מכיל את הדומיין (www.myApp.com) ואת סוג הבקשה – A (בקשת IP לdomain)
2. שולח בעזרת פונקציית sendp()
3. ממתין לקבלת פקטת response משרת הDNS בעזרת פונקציית sniff()
4. מוציא מתוך הפקטה את הip של האפליקציה

צד שרת:

1. ממתין לפקטות בפורט 53 שנשלחות לIP של הDNS ("10.0.0.18") בעזרת פונקציית sniff()
2. אם התקבלה פקטה הDNS עובר לפונקציה handle_dns_request
   2.1. מוציא מהפקטה את הdomain שמחפשים לו את הIP
   2.2. קורא לפונקציה check_domain_to_ip להחזרת הIP
      2.2.1. פונקציה זו בודקת האם הdomain קיים במילון של הDNS
      2.2.2. במידה וכן- תחזיר את הIP
      2.2.3. במידה ולא- ע"י שימוש בפונקציה socket.gethostbyname(domain) בודקת את הIP, אם הוחזרה תשובה- מחזירה את הIP אם לא- מחזירה '0.0.0.0'
   2.3. בניית פקטת התגובה-
      Id זהה לבקשה
      qr דולק – מציין תגובת DNS
      aa דולק- סמכות להחזיר את הIP
      rd דולק כי המשתמש ביקש רקורסיבי במידה ואין את התשובה
      qdcount – שאלה אחת
      ancount – תשובה אחת
      nscount – ללא רשומות סמכות
      arcount – ללא רשומות נוספות
   2.4. שליחת הפקטה בעזרת הפונקציה sendp()
3. חזרה לסעיף 1

צילומי מסך

Wireshark: (מהקלטה DNS.pcap)

1	0.000000000	10.0.0.23	10.0.0.18	DNS	75	Standard query 0x0000 A www.myApp.com
2	2.021211741	10.0.0.18	10.0.0.23	DNS	104	Standard query response 0x0000 A www.myApp.com A 10.0.2.50

צד לקוח:

```
to DNS
sending DNS query to 'www.myApp.com'
.
Sent 1 packets.
got DNS response
ip of the app:  10.0.2.50
done with DNS
```

צד שרת:

```
listening to DNS request...
domain: www.myApp.com
DNS dict=  {'www.myApp.com': '10.0.2.50'}
domain in dict!, IP-  10.0.2.50
Received DNS query for  www.myApp.com
10.0.2.50
.
Sent 1 packets.
send DNS response
done with query
```

APP:

שימוש בספריות scapy וsocket, חיבורים rUDP וTCP

הסבר כללי על מימוש חיבור rUDP:

1. גודל פקטת מידע מינימלית: 1000 ביטים (גודל חלון השליחה)
2. שליחת מידע עם seq_num
3. ניסיון לקבל ack במשך 5 שניות (timeout=5)
4. אם התקבל ack בseq_num שנשלח- החלון יגדל פי 2 (או יהיה גודל החלון המקסימלי של הלקוח פחות headers)
5. אם לא התקבל ack או התקבל ack על seq_num קודם-
   a. פעם ראשונה עד שלישית: נחלק את החלון ל2 (אם קטן מ2000 נחזיר לגודל המינימלי- 1000)
   b. פעם שלישית: גודל החלון יקטן לגודל המינימלי- 1000

צד לקוח:

1. מחכה לקבל מידע במשך 5 שניות (timeout=5)
2. במידה והגיעה פקטה - שמירת הseq_num של הפקטה
3. במידה והseq_num שווה לseq שאנו מצפים לו- תגדיל את הseq של הack שנשלח
4. אחרת- תשלח שוב ack עם הseq האחרון שהגיע

חיבור בין האפליקציה לשרת:

שימוש בספריית socket וחיבור TCP

צד אפליקציה (פונקציית get_image()):

1. נפתח חיבור TCP ipV4 בין האפליקציה לשרת ע"י הפונקציה socket.socket(socket.AF_INET, socket.SOCK_STREAM)
2. נחבר את האפליקציה לip והפורט הספציפיים ע"י connect()
3. נשלח בקשת http get לקבלת הכתובת של התמונה המבוקשת (rUDP/TCP – לפי בחירת המשתמש, לכל בחירה נתיב תמונה שונה) ע"י sendall()
4. נמתין לקבלת התשובה ע"י recv()
5. נשמור את הכתובת במשתנה addr_image
6. נסגור את החיבור

צד שרת:

1. נפתח חיבור TCP ipV4 בין השרת לאפליקציה ע"י הפונקציה socket.socket(socket.AF_INET, socket.SOCK_STREAM)
2. נעשה שיוך בין הפורט לכתובת שנרצה באמצעות פונקציית bind() (פורט 40693, 693 - ספרות אחרונות בתז של אופק)
3. נקבל את בקשת הhttp get מהאפליקציה ע"י פונקציית recv()
4. נבדוק איזו תמונה התבקשה (rUDP/TCP) ונשלח את כתובת התמונה המבוקשת ע"י sendall()

לקוח UDP:

חיבור ראשוני לאפליקציה: שימוש בספריית scapy

המשך חיבור בUDP socket

צד לקוח:

1. גודל החלון של הלקוח הינו 16384 ביטים
2. בלולאה עד קבלת ack :
   a. יצירת פקטה של שליחת גודל החלון לאפליקציה
   b. שליחת הפקטה בעזרת פונקציה ()send
   c. קבלת הack בעזרת פונקציה ()sniff מוגדר timeout של 5 שניות- במידה ולא קיבלנו בזמן זה תשלח פקטת גודל החלון פעם נוספת
3. קריאה לפונקציית ()get_image
   a. פתיחת socket UDP
   b. bind לפורט 40693 (693 – סוף תז של אופק)
   c. הגדרת הסוקט ללא בלוקינג (בשביל מימוש timeout אמין)- sock.setblocking(False)
   d. שינה למשך כ-4 שניות (לאפשר לאפליקציה לסיים תקשורת עם השרת)
   e. יצירת קובץ חדש בשם "received_image.jpg"
   f. קבלת התמונה ב rUDP:
      i. שמירת משתנה wanted_seq ששומר את מספר הseq_num אותו הלקוח רוצה לקבל כרגע (הבא ממה שהוא כבר קיבל, מאותחל ב0)
      ii. לולאת קבלת תמונה:
         1. הגדרנו משתנה num מאותחל ב5 (למימוש timout)
         2. לולאת קבלת מידע:
            a. קבלת מידע בעזרת פונקציה recvfrom (עם גודל החלון המקסימלי של הלקוח)
            b. שנייה של שינה
            c. אם התקבל מידע:
               i. אם התקבל "final" נגמר המידע ונצא מהלולאה
               ii. שמירת הseq_num
               iii. אם הseq_num שקיבלנו שווה לwanted_seq (מה שאנחנו מצפים לקבל) אז נכתוב את המידע שקיבלנו לקובץ ("received_image.jpg") ונגדיל את wanted_seq
               iv. נצא מהלולאה בסעיף c
            d. אם לא התקבל- נכנס ללולאה 5 פעמים ואז נצא (timout=5)
         3. אם התקבל "final" נגמר המידע ונצא מהלולאה
         4. נשלח ack עם הwanted_seq פחות 1 (מספר הseq האחרון שקיבלנו וכתבנו את הdata שלו לקובץ)
   g. סגירת הסוקט עם האפליקציה
   h. פתיחת התמונה שהתקבלה

צד אפליקציה:

1. ממתינה לבקשה בעזרת פונקציה ()sniff
2. במידה והבקשה שהגיעה הינה מעל UDP קראיה לפונקציה get_request_rUDP(pkt)
   a. הוצאת גודל חלון הלקוח מהפקטה שהגיעה
   b. בניית פקטת ack
   c. שליחה של הack בעזרת פונקציית send

d. קריאה לget_image – **תקשורת אפליקציה שרת מפורטת בעמ 10**

e. קריאה לimage_to_client ()

i. הגדרת שליחת חלון מינימלי ל1000 ביטים בשם buffer_size

ii. הגדרת החלון המקסימלי של הלקוח פחות headers

iii. פתיחת נתיב התמונה שהתקבלה מהשרת

iv. פתיחת socket UDP

v. bind לפורט 40308 (308 ספרות אחרונות בתז של מוריה אסתר)

vi. הגדרת הסוקט ללא בלוקינג (בשביל מימוש timeout אמין)- sock.setblocking(False)

vii. שמירת count (יאותחל ב0) לבדיקת של 3 ackים זהים

viii. שמירת file_pos – המציין את המיקום סמן הקריאה בתמונה

ix. שמירת seq_num – המספר שישלח בפקטת המידע (יאותחל ב0)

x. שמירת seq_ack – המספר שהגיע בack (יאותחל ב1-)

xi. לולאת שליחת התמונה:

1. אם ה seq_num שווה לseq_ack פחות 1:

a. נעביר את סמן הקריאה בקובץ לfile_pos בעזרת הפונקציה seek

b. נשמור chunk של קריאת buffer_size ביטים מהקובץ

c. אם אין יותר מידע בקובץ נצא מהלולאה

d. נשלח את chunk המידע ללקוח בעזרת הפונקציה sendto

e. לולאת קבלה: נחכה במשך 5 שניות לקבלת ack בעזרת פונקציית recvfrom

f. אם קיבלנו מידע-

i. אם הack היה על השליחה האחרונה – נגדיל את הbuffer_size פי 2 (מקסימום גודל החלון של הלקוח פחות headers)

ii. אם הack אינו עבור הפקטה האחרונה שנשלחה:

1. פחות מ3 פעמים: נחלק ב2 את הbuffer_size ונגדיל את הcount למספר הtimeout)

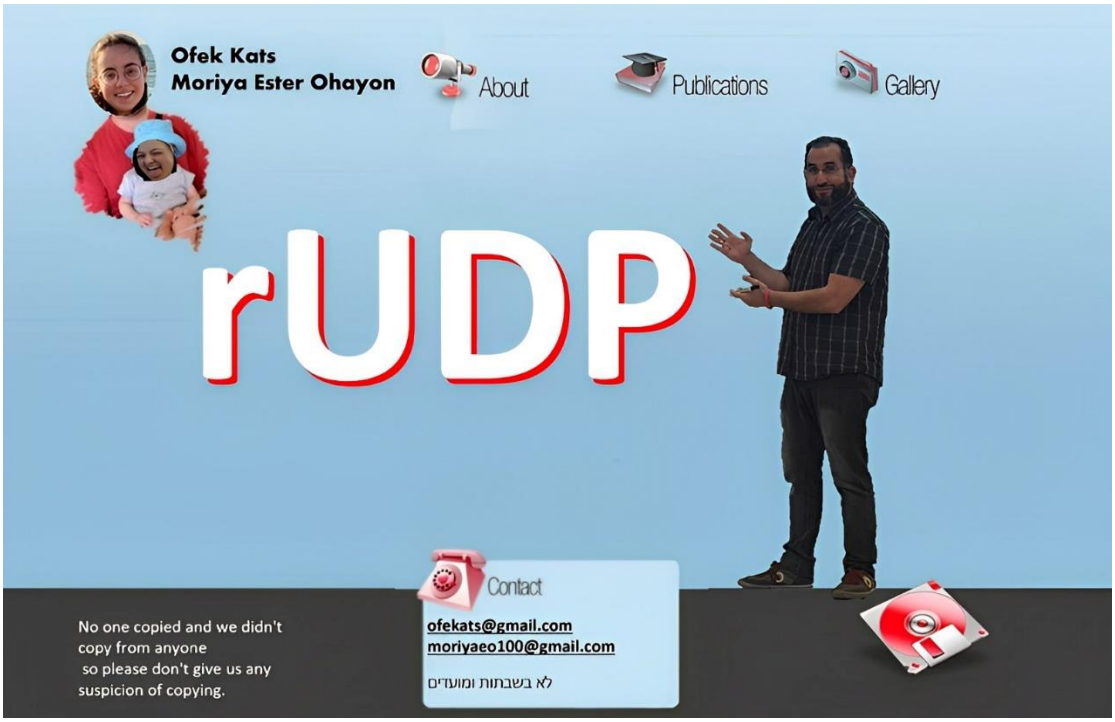
2. 3 פעמים: נחזיר לגודל המינימלי- 1000 (נחזיר את count ל0)

iii. נצא מלולאת הקבלה

xii. נשלח פקטה אחרונה "final" להודעה ללקוח שאין יותר מידע לקבל

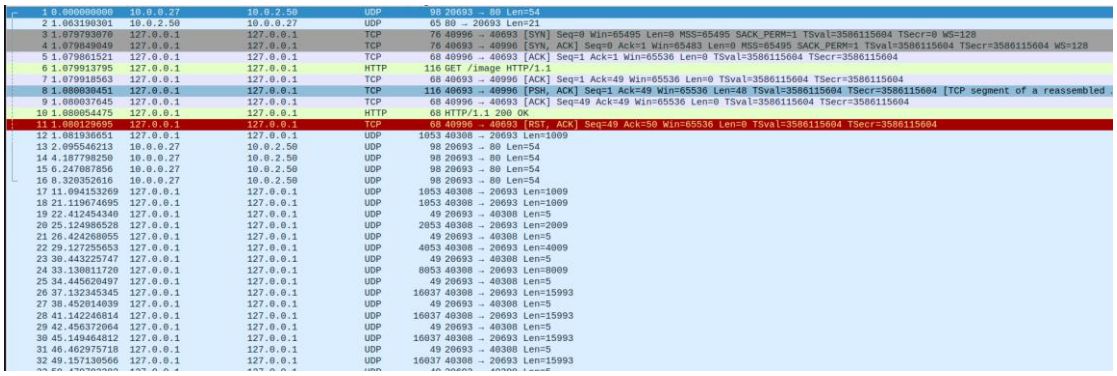
xiii. סגירת הסוקט בעזרת הפונקציה sock.close()

התמונה שנשלחת:



צילומי מסך

Wireshark: (מהקלטה rUDP.pcap)



1	0.000000000	10.0.0.27	10.0.2.50	UDP	98 20693 → 80 Len=54
2	1.063190301	10.0.2.50	10.0.0.27	UDP	65 80 → 20693 Len=21
3	1.079793070	127.0.0.1	127.0.0.1	TCP	76 40996 → 40693 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=3586115604 TSecr=0 WS=128
4	1.079849049	127.0.0.1	127.0.0.1	TCP	76 40693 → 40996 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=3586115604 TSecr=3586115604 WS=128
5	1.079861521	127.0.0.1	127.0.0.1	TCP	68 40996 → 40693 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=3586115604 TSecr=3586115604
6	1.079913795	127.0.0.1	127.0.0.1	HTTP	116 GET /image HTTP/1.1
7	1.079918563	127.0.0.1	127.0.0.1	TCP	68 40693 → 40996 [ACK] Seq=1 Ack=49 Win=65536 Len=0 TSval=3586115604 TSecr=3586115604
8	1.080030451	127.0.0.1	127.0.0.1	TCP	116 40693 → 40996 [PSH, ACK] Seq=1 Ack=49 Win=65536 Len=48 TSval=3586115604 TSecr=3586115604 [TCP segment of a reassembled
9	1.080037645	127.0.0.1	127.0.0.1	TCP	68 40996 → 40693 [ACK] Seq=49 Ack=49 Win=65536 Len=0 TSval=3586115604 TSecr=3586115604
10	1.080054475	127.0.0.1	127.0.0.1	HTTP	68 HTTP/1.1 200 OK
11	1.080129695	127.0.0.1	127.0.0.1	TCP	68 40996 → 40693 [RST, ACK] Seq=49 Ack=50 Win=65536 Len=0 TSval=3586115604 TSecr=3586115604
12	1.081936651	127.0.0.1	127.0.0.1	UDP	1053 40308 → 20693 Len=1009
13	2.095546213	10.0.0.27	10.0.2.50	UDP	98 20693 → 80 Len=54
14	4.187798250	10.0.0.27	10.0.2.50	UDP	98 20693 → 80 Len=54
15	6.247087856	10.0.0.27	10.0.2.50	UDP	98 20693 → 80 Len=54
16	8.320352616	10.0.0.27	10.0.2.50	UDP	98 20693 → 80 Len=54
17	11.094153269	127.0.0.1	127.0.0.1	UDP	1053 40308 → 20693 Len=1009
18	21.119674695	127.0.0.1	127.0.0.1	UDP	1053 40308 → 20693 Len=1009
19	22.412454340	127.0.0.1	127.0.0.1	UDP	49 20693 → 40308 Len=5
20	25.124986528	127.0.0.1	127.0.0.1	UDP	2053 40308 → 20693 Len=2009
21	26.424268055	127.0.0.1	127.0.0.1	UDP	49 20693 → 40308 Len=5
22	29.127255653	127.0.0.1	127.0.0.1	UDP	4053 40308 → 20693 Len=4009
23	30.443225747	127.0.0.1	127.0.0.1	UDP	49 20693 → 40308 Len=5
24	33.130811720	127.0.0.1	127.0.0.1	UDP	8053 40308 → 20693 Len=8009
25	34.445620497	127.0.0.1	127.0.0.1	UDP	49 20693 → 40308 Len=5
26	37.132345345	127.0.0.1	127.0.0.1	UDP	16037 40308 → 20693 Len=15993
27	38.452014039	127.0.0.1	127.0.0.1	UDP	49 20693 → 40308 Len=5
28	41.142246814	127.0.0.1	127.0.0.1	UDP	16037 40308 → 20693 Len=15993
29	42.456372064	127.0.0.1	127.0.0.1	UDP	49 20693 → 40308 Len=5
30	45.149464812	127.0.0.1	127.0.0.1	UDP	16037 40308 → 20693 Len=15993
31	46.462975718	127.0.0.1	127.0.0.1	UDP	49 20693 → 40308 Len=5
32	49.157130566	127.0.0.1	127.0.0.1	UDP	16037 40308 → 20693 Len=15993

צד לקוח:

```
Enter U for rUdp or T for TCP: U
you chose rUDP!
client max window=  16384
.
Sent 1 packets.
send max window
wait for ack
wait for ack
wait to recv
got packet seq: 0
send seq:  0
wait to recv
got packet seq: 1
send seq:  1
wait to recv
got packet seq: 2
send seq:  2
wait to recv
got packet seq: 3
send seq:  3
wait to recv
got packet seq: 4
send seq:  4
wait to recv
got packet seq: 5
send seq:  5
wait to recv
got packet seq: 6
send seq:  6
wait to recv
got packet seq: 7
send seq:  7
wait to recv
got packet seq: 8
send seq:  8
wait to recv
wait to recv
got all the image!
opening image...
back to main
```

צד שרת:

```
seq_num 2
wait for ack
ack seq: 2
got ack! double the size of buffer!
send new packet, buffer size=  8000
seq_num 3
wait for ack
ack seq: 3
got ack! buffer = max client window!
send new packet, buffer size=  15984
seq_num 4
wait for ack
ack seq: 4
got ack! buffer = max client window!
send new packet, buffer size=  15984
seq_num 5
wait for ack
ack seq: 5
got ack! buffer = max client window!
send new packet, buffer size=  15984
seq_num 6
wait for ack
ack seq: 6
got ack! buffer = max client window!
send new packet, buffer size=  15984
seq_num 7
wait for ack
ack seq: 7
got ack! buffer = max client window!
send new packet, buffer size=  15984
seq_num 8
wait for ack
ack seq: 8
got ack! buffer = max client window!
send new packet, buffer size=  15984
```

```
got pkt
got pkt UDP
max window of client =  16384
.
Sent 1 packets.

connect to sever
connecting to  ('127.0.0.1', 40693)
send get path of image to server:  b'GET /image HTTP/1.1\r\nHost: ww
w.myApp.com\r\n\r\nrUDP'
wait to recv from server
addr image =  image_UDP.jpg
done with server

send new packet, buffer size=  1000
seq_num 0
wait for ack
didn't recv
send new packet, buffer size=  1000
seq_num 0
wait for ack
didn't recv
send new packet, buffer size=  1000
seq_num 0
wait for ack
ack seq: 0
got ack! double the size of buffer!
send new packet, buffer size=  2000
seq_num 1
wait for ack
ack seq: 1
got ack! double the size of buffer!
send new packet, buffer size=  4000
```

```
no more data to send
done with client
```

לקוח TCP:

שימוש בספריית socket וחיבור TCP

צד לקוח:

1. בלולאה עד קבלת ack בעזרת פונקציה ()sniff
2. פתיחת socket TCP בipV4
3. bind לפורט 30693 (693 – סוף תז של אופק)
4. שליחת בקשת כתובת התמונה ע"י http get
5. קבלת התמונה ב TCP:
    a. קבלת תשובת 200 ok
    b. שמירת משתנה response שיאסוף לתוכו את כל חלקי התמונה
    c. קבלת התמונה בצ'אנקים של 1024 (כולל הדרים)
6. פתיחת קובץ לתמונה בשם received_image.jpg וכתיבת התמונה
7. פתיחת התמונה שהתקבלה
8. סגירת הסוקט עם האפליקציה

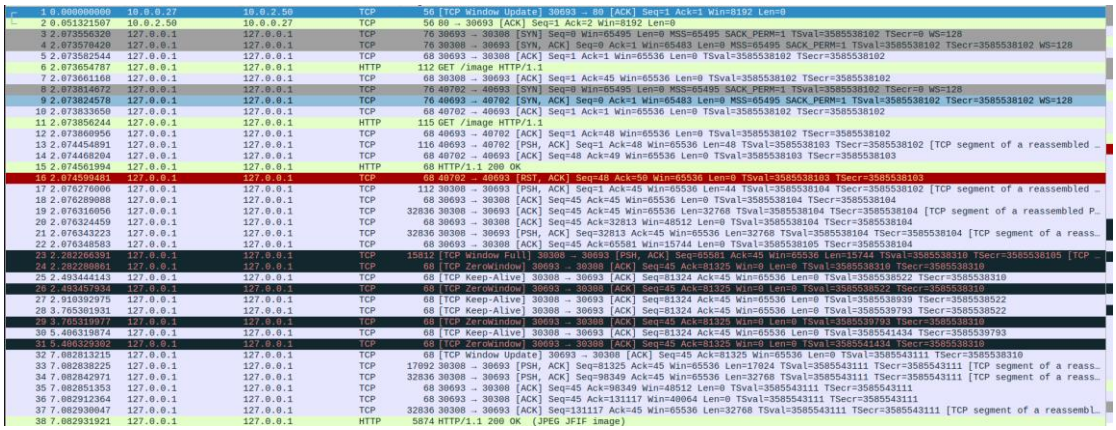
צד אפליקציה:

3. ממתינה לבקשה בעזרת פונקציה ()sniff
4. בניית פקטת ack ושליחתה בעזרת פונקציית ()send
5. קריאה לפונקציה get_request_TCP
    a. פתיחת קשר TCP עם הלקוח בipV4
    b. קישור כתובת ופורט ע"י הפונקציה ()bind (פורט 30308 , 308 ספרות אחרונות בתז של מוריה אסתר)
    c. קבלת בקשת http get לכתובת התמונה
    d. קריאה לפונקציה ()get image – מפורט בעמ 10
    e. פתיחת התמונה לקריאה בינארית ע"י: open(addr_image, 'rb') as f כך ש addr_image זה כתובת התמונה המתקבלת מפונקציית ()get_image
    f. שליחת תשובת 200 ok ללקוח על בקשת התמונה
    g. שליחת כל התמונה בפעם אחת (היא תתקבל בצ'אנקים של 1024 כולל headers)

התמונה שנשלחת:

צילומי מסך

Wireshark: (מהקלטה TCP.pcap)



1	0.000000000	10.0.0.27	10.0.2.50	TCP	56	[TCP Window Update] 30693 → 80 [ACK] Seq=1 Ack=1 Win=8192 Len=0
2	0.051321507	10.0.2.50	10.0.0.27	TCP	56	80 → 30693 [ACK] Seq=1 Ack=2 Win=8192 Len=0
3	2.073556320	127.0.0.1	127.0.0.1	TCP	76	30693 → 30308 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=3585538102 TSecr=0 WS=128
4	2.073578420	127.0.0.1	127.0.0.1	TCP	76	30308 → 30693 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=3585538102 TSecr=3585538102 WS=128
5	2.073582544	127.0.0.1	127.0.0.1	TCP	68	30693 → 30308 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=3585538102 TSecr=3585538102
6	2.073654787	127.0.0.1	127.0.0.1	HTTP	112	GET /image HTTP/1.1
7	2.073661168	127.0.0.1	127.0.0.1	TCP	68	30308 → 30693 [ACK] Seq=1 Ack=45 Win=65536 Len=0 TSval=3585538102 TSecr=3585538102
8	2.073814672	127.0.0.1	127.0.0.1	TCP	76	40702 → 40693 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=3585538102 TSecr=0 WS=128
9	2.073824578	127.0.0.1	127.0.0.1	TCP	76	40693 → 40702 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=3585538102 TSecr=3585538102 WS=128
10	2.073833650	127.0.0.1	127.0.0.1	TCP	68	40702 → 40693 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=3585538102 TSecr=3585538102
11	2.073856244	127.0.0.1	127.0.0.1	HTTP	115	GET /image HTTP/1.1
12	2.073860956	127.0.0.1	127.0.0.1	TCP	68	40693 → 40702 [ACK] Seq=1 Ack=48 Win=65536 Len=0 TSval=3585538102 TSecr=3585538102
13	2.074454891	127.0.0.1	127.0.0.1	TCP	116	40693 → 40702 [PSH, ACK] Seq=1 Ack=48 Win=65536 Len=48 TSval=3585538103 TSecr=3585538102 [TCP segment of a reassembled …
14	2.074468204	127.0.0.1	127.0.0.1	TCP	68	40702 → 40693 [ACK] Seq=48 Ack=49 Win=65536 Len=0 TSval=3585538103 TSecr=3585538103
15	2.074561994	127.0.0.1	127.0.0.1	HTTP	68	HTTP/1.1 200 OK
16	2.074599481	127.0.0.1	127.0.0.1	TCP	68	40702 → 40693 [RST, ACK] Seq=48 Ack=50 Win=65536 Len=0 TSval=3585538103 TSecr=3585538103
17	2.076276006	127.0.0.1	127.0.0.1	TCP	112	30308 → 30693 [PSH, ACK] Seq=1 Ack=45 Win=65536 Len=44 TSval=3585538104 TSecr=3585538102 [TCP segment of a reassembled …
18	2.076289088	127.0.0.1	127.0.0.1	TCP	68	30693 → 30308 [ACK] Seq=45 Ack=45 Win=65536 Len=0 TSval=3585538104 TSecr=3585538104
19	2.076316056	127.0.0.1	127.0.0.1	TCP	32836	30308 → 30693 [ACK] Seq=45 Ack=45 Win=65536 Len=32768 TSval=3585538104 TSecr=3585538104 [TCP segment of a reassembled P…
20	2.076324459	127.0.0.1	127.0.0.1	TCP	68	30693 → 30308 [ACK] Seq=45 Ack=32813 Win=48512 Len=0 TSval=3585538104 TSecr=3585538104
21	2.076343223	127.0.0.1	127.0.0.1	TCP	32836	30308 → 30693 [PSH, ACK] Seq=32813 Ack=45 Win=65536 Len=32768 TSval=3585538104 TSecr=3585538104 [TCP segment of a reass…
22	2.076348583	127.0.0.1	127.0.0.1	TCP	68	30693 → 30308 [ACK] Seq=45 Ack=65581 Win=15744 Len=0 TSval=3585538105 TSecr=3585538104
23	2.282206391	127.0.0.1	127.0.0.1	TCP	15812	[TCP Window Full] 30308 → 30693 [PSH, ACK] Seq=65581 Ack=45 Win=65536 Len=15744 TSval=3585538310 TSecr=3585538105 [TCP …
24	2.282280861	127.0.0.1	127.0.0.1	TCP	68	[TCP ZeroWindow] 30693 → 30308 [ACK] Seq=45 Ack=81325 Win=0 Len=0 TSval=3585538310 TSecr=3585538310
25	2.493444143	127.0.0.1	127.0.0.1	TCP	68	[TCP Keep-Alive] 30308 → 30693 [ACK] Seq=81324 Ack=45 Win=65536 Len=0 TSval=3585538522 TSecr=3585538310
26	2.493457934	127.0.0.1	127.0.0.1	TCP	68	[TCP ZeroWindow] 30693 → 30308 [ACK] Seq=45 Ack=81325 Win=0 Len=0 TSval=3585538522 TSecr=3585538310
27	2.910392975	127.0.0.1	127.0.0.1	TCP	68	[TCP Keep-Alive] 30308 → 30693 [ACK] Seq=81324 Ack=45 Win=65536 Len=0 TSval=3585538939 TSecr=3585538522
28	3.765301931	127.0.0.1	127.0.0.1	TCP	68	[TCP Keep-Alive] 30308 → 30693 [ACK] Seq=81324 Ack=45 Win=65536 Len=0 TSval=3585539793 TSecr=3585538522
29	3.765319977	127.0.0.1	127.0.0.1	TCP	68	[TCP ZeroWindow] 30693 → 30308 [ACK] Seq=45 Ack=81325 Win=0 Len=0 TSval=3585539793 TSecr=3585538310
30	5.406319874	127.0.0.1	127.0.0.1	TCP	68	[TCP Keep-Alive] 30308 → 30693 [ACK] Seq=81324 Ack=45 Win=65536 Len=0 TSval=3585541434 TSecr=3585539793
31	5.406329302	127.0.0.1	127.0.0.1	TCP	68	[TCP ZeroWindow] 30693 → 30308 [ACK] Seq=45 Ack=81325 Win=0 Len=0 TSval=3585541434 TSecr=3585538310
32	7.082813215	127.0.0.1	127.0.0.1	TCP	68	[TCP Window Update] 30693 → 30308 [ACK] Seq=45 Ack=81325 Win=65536 Len=0 TSval=3585543111 TSecr=3585538310
33	7.082838225	127.0.0.1	127.0.0.1	TCP	17092	30308 → 30693 [PSH, ACK] Seq=81325 Ack=45 Win=65536 Len=17024 TSval=3585543111 TSecr=3585543111 [TCP segment of a reass…
34	7.082842971	127.0.0.1	127.0.0.1	TCP	32836	30308 → 30693 [PSH, ACK] Seq=98349 Ack=45 Win=65536 Len=32768 TSval=3585543111 TSecr=3585543111 [TCP segment of a reass…
35	7.082851353	127.0.0.1	127.0.0.1	TCP	68	30693 → 30308 [ACK] Seq=45 Ack=98349 Win=48512 Len=0 TSval=3585543111 TSecr=3585543111
36	7.082912364	127.0.0.1	127.0.0.1	TCP	68	30693 → 30308 [ACK] Seq=45 Ack=131117 Win=40064 Len=0 TSval=3585543111 TSecr=3585543111
37	7.082930047	127.0.0.1	127.0.0.1	TCP	32836	30308 → 30693 [ACK] Seq=131117 Ack=45 Win=65536 Len=32768 TSval=3585543111 TSecr=3585543111 [TCP segment of a reassembl…
38	7.082931921	127.0.0.1	127.0.0.1	HTTP	5874	HTTP/1.1 200 OK (JPEG JFIF image)

צד לקוח:

```
you chose TCP!
.
Sent 1 packets.
send get image
recv ack
connecting to  ('127.0.0.1', 30308)
bind
wait to recv
got packet
got packet
got packet
got packet
got packet
got packet
got packet
got packet
got packet
got packet
got packet
got packet
got packet
got packet
got all the image!
opening image...
done
back to main
enter E to exit or C to chose protocol again:
```

צד שרת:

```
starting up on 127.0.0.1 port 30308
waiting for a connection
connection from ('127.0.0.1', 30693)
got from client =  b'GET /image HTTP/1.1\r\nHost: www.myApp.com\r\n\
r\n'

connect to sever
connecting to  ('127.0.0.1', 40693)
send get path of image to server:  b'GET /image HTTP/1.1\r\nHost: ww
w.myApp.com\r\n\r\nTCP'
wait to recv from server
addr image =  image_TCP.jpg
done with server

send all the image to client
finally
```

מערכת באיבוד פקטות: (20% איבוד)

TCP:

Wireshark: (מהקלטה TCP_20_LOSS.pcap)

1	0.000000000	PcsCompu_9b:dc:8c		ARP	44	Who has 10.0.2.50? Tell 10.0.2.15
2	2.024177988	10.0.0.27	10.0.2.50	TCP	56	[TCP Window Update] 30693 → 80 [ACK] Seq=1 Ack=1 Win=8192 Len=0
3	2.090367490	PcsCompu_9b:dc:8c		ARP	44	Who has 10.0.2.2? Tell 10.0.2.15
4	2.090556857	RealtekU_12:35:02		ARP	62	10.0.2.2 is at 52:54:00:12:35:02
5	2.113223267	10.0.2.50	10.0.0.27	TCP	56	80 → 30693 [ACK] Seq=1 Ack=2 Win=8192 Len=0
6	2.113609055	RealtekU_12:35:02		ARP	62	Who has 10.0.2.50? Tell 10.0.2.2
7	4.151838259	127.0.0.1	127.0.0.1	TCP	76	30693 → 30308 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=3587446867 TSecr=0 WS=128
8	4.151860817	127.0.0.1	127.0.0.1	TCP	76	30308 → 30693 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=3587446867 TSecr=3587446867 WS=128
9	4.151880050	127.0.0.1	127.0.0.1	TCP	68	30693 → 30308 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=3587446867 TSecr=3587446867
10	4.152134130	127.0.0.1	127.0.0.1	HTTP	112	GET /image HTTP/1.1
11	4.152148212	127.0.0.1	127.0.0.1	TCP	68	30308 → 30693 [ACK] Seq=1 Ack=45 Win=65536 Len=0 TSval=3587446868 TSecr=3587446868
12	4.152367225	127.0.0.1	127.0.0.1	TCP	76	39164 → 40693 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=3587446868 TSecr=0 WS=128
13	7.199068175	127.0.0.1	127.0.0.1	TCP	76	40693 → 39164 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=3587449915 TSecr=3587446868 WS=128
14	7.199103243	127.0.0.1	127.0.0.1	TCP	76	[TCP Retransmission] [TCP Port numbers reused] 39164 → 40693 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=3587
15	7.199113474	127.0.0.1	127.0.0.1	TCP	68	39164 → 40693 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=3587449915 TSecr=3587449915
16	7.199131575	127.0.0.1	127.0.0.1	TCP	76	[TCP Retransmission] 40693 → 39164 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=3587449915 TSecr=35
17	7.199139928	127.0.0.1	127.0.0.1	TCP	68	[TCP Dup ACK 15#1] 39164 → 40693 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=3587449915 TSecr=3587449915
18	24.402143346	10.0.2.15	192.168.1.1	DNS	102	Standard query 0xe7d2 A connectivity-check.ubuntu.com OPT
19	24.418597696	192.168.1.1	10.0.2.15	DNS	246	Standard query response 0xe7d2 A connectivity-check.ubuntu.com A 91.189.91.48 A 91.189.91.49 A 185.125.190.18 A 35.232.11
20	24.419875181	10.0.2.15	91.189.91.48	TCP	76	52522 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=2820779577 TSecr=0 WS=128
21	24.566047838	91.189.91.48	10.0.2.15	TCP	62	80 → 52522 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
22	24.566131163	10.0.2.15	91.189.91.48	TCP	56	52522 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
23	24.566385853	10.0.2.15	91.189.91.48	HTTP	143	GET / HTTP/1.1
24	24.566832067	91.189.91.48	10.0.2.15	TCP	62	80 → 52522 [ACK] Seq=1 Ack=88 Win=65535 Len=0
25	24.712493956	91.189.91.48	10.0.2.15	HTTP	203	HTTP/1.1 204 No Content
26	24.712539888	10.0.2.15	91.189.91.48	TCP	56	52522 → 80 [ACK] Seq=88 Ack=148 Win=64093 Len=0
27	24.712749923	91.189.91.48	10.0.2.15	TCP	62	80 → 52522 [FIN, ACK] Seq=148 Ack=88 Win=65535 Len=0
28	24.712960729	10.0.2.15	91.189.91.48	TCP	56	52522 → 80 [FIN, ACK] Seq=88 Ack=149 Win=64092 Len=0
29	24.713591076	91.189.91.48	10.0.2.15	TCP	62	80 → 52522 [ACK] Seq=149 Ack=89 Win=65535 Len=0
30	42.271207268	127.0.0.1	127.0.0.1	HTTP	115	GET /image HTTP/1.1
31	42.271230869	127.0.0.1	127.0.0.1	TCP	68	40693 → 39164 [ACK] Seq=1 Ack=48 Win=65536 Len=0 TSval=3587484987 TSecr=3587484987
32	42.271386091	127.0.0.1	127.0.0.1	TCP	116	40693 → 39164 [PSH, ACK] Seq=1 Ack=48 Win=65536 Len=48 TSval=3587484987 TSecr=3587484987 [TCP segment of a reassembled PD
33	42.271396404	127.0.0.1	127.0.0.1	TCP	68	39164 → 40693 [ACK] Seq=48 Ack=49 Win=65536 Len=0 TSval=3587484987 TSecr=3587484987
34	42.271457944	127.0.0.1	127.0.0.1	TCP	68	39164 → 40693 [RST, ACK] Seq=48 Ack=49 Win=65536 Len=0 TSval=3587484987 TSecr=3587484987
35	42.271647983	127.0.0.1	127.0.0.1	TCP	112	30308 → 30693 [PSH, ACK] Seq=1 Ack=45 Win=65536 Len=44 TSval=3587484987 TSecr=3587446868 [TCP segment of a reassembled PD
36	42.271656284	127.0.0.1	127.0.0.1	TCP	68	30693 → 30308 [ACK] Seq=45 Ack=45 Win=65536 Len=0 TSval=3587484987 TSecr=3587484987

UDP:

Wireshark: (מהקלטה UDP_20_LOSS.pcap)

1	0.000000000	PcsCompu_9b:dc:8c		ARP	44	Who has 10.0.2.50? Tell 10.0.2.15
2	2.020701078	10.0.0.27	10.0.2.50	UDP	98	20693 → 80 Len=54
3	2.067801384	PcsCompu_9b:dc:8c		ARP	44	Who has 10.0.2.50? Tell 10.0.2.15
4	3.080419920	PcsCompu_9b:dc:8c		ARP	44	Who has 10.0.2.2? Tell 10.0.2.15
5	3.080609451	RealtekU_12:35:02		ARP	62	10.0.2.2 is at 52:54:00:12:35:02
6	3.099636546	10.0.2.50	10.0.0.27	UDP	65	80 → 20693 Len=21
7	3.115399254	127.0.0.1	127.0.0.1	TCP	76	56154 → 40693 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=3587756423 TSecr=0 WS=1
8	3.115413763	127.0.0.1	127.0.0.1	TCP	76	40693 → 56154 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=3587756423 TS
9	3.115439621	127.0.0.1	127.0.0.1	HTTP	116	GET /image HTTP/1.1
10	3.115486619	127.0.0.1	127.0.0.1	TCP	68	40693 → 56154 [ACK] Seq=1 Ack=49 Win=65536 Len=0 TSval=3587756423 TSecr=3587756423
11	3.115662078	127.0.0.1	127.0.0.1	TCP	116	40693 → 56154 [PSH, ACK] Seq=1 Ack=49 Win=65536 Len=48 TSval=3587756423 TSecr=3587756423 [TCP
12	3.115679215	127.0.0.1	127.0.0.1	HTTP	68	HTTP/1.1 200 OK
13	3.115705025	127.0.0.1	127.0.0.1	TCP	68	56154 → 40693 [ACK] Seq=49 Ack=49 Win=65536 Len=0 TSval=3587756423 TSecr=3587756423
14	3.115961702	127.0.0.1	127.0.0.1	TCP	68	56154 → 40693 [RST, ACK] Seq=49 Ack=50 Win=65536 Len=0 TSval=3587756423 TSecr=3587756423
15	3.115978879	127.0.0.1	127.0.0.1	UDP	1053	40308 → 20693 Len=1009
16	3.115990425	127.0.0.1	127.0.0.1	ICMP	592	Destination unreachable (Port unreachable)
17	4.088290970	10.0.0.27	10.0.2.50	UDP	98	20693 → 80 Len=54
18	4.152397141	PcsCompu_9b:dc:8c		ARP	44	Who has 10.0.2.50? Tell 10.0.2.15
19	6.172367993	10.0.0.27	10.0.2.50	UDP	98	20693 → 80 Len=54
20	6.224380630	PcsCompu_9b:dc:8c		ARP	44	Who has 10.0.2.50? Tell 10.0.2.15
21	8.254598605	10.0.0.27	10.0.2.50	UDP	98	20693 → 80 Len=54
22	8.300188817	PcsCompu_9b:dc:8c		ARP	44	Who has 10.0.2.50? Tell 10.0.2.15
23	10.320792861	10.0.0.27	10.0.2.50	UDP	98	20693 → 80 Len=54
24	13.811089061	10.0.2.15	192.168.1.1	DNS	102	Standard query 0xfcc0 A connectivity-check.ubuntu.com OPT
25	13.859984450	192.168.1.1	10.0.2.15	DNS	246	Standard query response 0xfcc0 A connectivity-check.ubuntu.com A 91.189.91.48 A 34.122.121.32
26	13.860656800	10.0.2.15	91.189.91.48	TCP	76	51600 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=2821079610 TSecr=0 WS=128
27	14.010331361	91.189.91.48	10.0.2.15	TCP	62	80 → 51600 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
28	14.010407362	10.0.2.15	91.189.91.48	TCP	56	51600 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
29	14.010750278	10.0.2.15	91.189.91.48	HTTP	143	GET / HTTP/1.1
30	14.011151837	91.189.91.48	10.0.2.15	TCP	62	80 → 51600 [ACK] Seq=1 Ack=88 Win=65535 Len=0
31	14.159754882	91.189.91.48	10.0.2.15	HTTP	203	HTTP/1.1 204 No Content
32	14.159813806	10.0.2.15	91.189.91.48	TCP	56	51600 → 80 [ACK] Seq=88 Ack=148 Win=64093 Len=0
33	14.160003844	10.0.2.15	91.189.91.48	TCP	56	51600 → 80 [FIN, ACK] Seq=88 Ack=148 Win=64093 Len=0
34	14.160208423	91.189.91.48	10.0.2.15	TCP	62	80 → 51600 [ACK] Seq=148 Ack=89 Win=65535 Len=0
35	14.160932557	91.189.91.48	10.0.2.15	TCP	62	80 → 51600 [FIN, ACK] Seq=148 Ack=89 Win=65535 Len=0
36	14.160968339	10.0.2.15	91.189.91.48	TCP	56	51600 → 80 [ACK] Seq=89 Ack=149 Win=64093 Len=0
37	23.156392188	127.0.0.1	127.0.0.1	UDP	1053	40308 → 20693 Len=1009
38	24.434162813	127.0.0.1	127.0.0.1	UDP	49	20693 → 40308 Len=5

דוגמא לtimeout:

```
timeout!!! buffer decreases by half, count- 0
send new packet, buffer size=  4001
seq_num 3
wait for ack
```

תשובות:

1. מנה לפחות ארבעה הבדלים עיקריים בין פרוטוקול TCP לQUIC-

a. **הבדל בדרך החיבור:**

TCP משתמש בדרך של 3 פקטות (SYN, SYN ACK, ACK)- לחיצת יד(handshake) ואילו QUIC משתמש ב2 פקטות בלבד- הלקוח שולח בקשה לחיבור עם מזהה והשרת מאשר.

b. **הבדל בשליחת מנות שאבדו:**

TCP מחכה לtimeout או למספר שכפולי ACK לפני שליחה מחדש של פקטה שאבדה. לעומת QUIC שלא מחכה לTIMEOUT ושולח פקטה נוספת במידה ויש שכפול ACK אחד. בנוסף הוא משתמש בFEC (מוסיף לכל חבילה מידע נוסף שיכול לעזור במידה ויהיה איבוד פקטות לשחזור המידע ללא שליחה מחדש של כל הפקטה).

c. **הבדל מבחינת אבטחה :**

ל-TCP אין אבטחה שמובנת בתוך הפרוטוקול (בדרך כלל מתבסס על TLS - מנגנון חיצוני). לעומת QUIC שכבר יש לו מנגנוני אבטחה מובנים כמו הצפנה ואימות.

d. **הבדל במספר החיבורים:**

בTCP אם לקוח צריך יותר מזרם מידע יחיד הוא יצטרך חיבור TCP נוסף לעומת QUIC שתומך בשליחת כמה זרמי מידע שונים בו זמנית עם חיבור יחיד

2. מנה לפחות שני הבדלים עיקריים בין Cubic ל-Vegas

a. **הבדל בין CUBIC לVEGAS :**

CUBIC שומר על חלוקה שווה של רוחב פס בין זרמים שונים של מידע לעומת VEGAS שאינה מבטיחה חלוקה שווה אלא נותנת עדיפות לזמני עיכוב נמוכים ואובדן פקטות נמוך מה שעלול לגרום לאי אחידות ברוחב הפס.

b. **הבדל נוסף-** VEGAS מגיבה מהר יותר לשינויים ברשת ע"י התאמת קצב השליחה מאשר CUBIC שתהיה איטית יותר בהגבה לשינויים (VEGAS מקטינה את חלון השליחה ביותר אגרסיביות לעומת cubic שמורידה באופן הדרגתי).

3. הסבר מהו פרוטוקול, BGP במה הוא שונה מ OSPF-והאם הוא עובד על פי מסלולים קצרים

a. פרוטוקול BGP: הוא פרוטוקול המאפשר לכל רשת למצוא את הנתיב הטוב ביותר לרשת אחרת (פרוטוקול חיצוני)

b. לעומת זאת OSPF שהוא פרוטוקול פנימי ברשת המוצא לכל ראוטר את הדרך הקצרה ביותר לכל ראוטר אחר

c. BGP לא תמיד עובד על פי המסלול הקצר ביותר אלא לפעמים יקח מסלול ארוך יותר אך זול יותר (או התחשבות בכל מיני גורמים נוספים)

4. אם יהיה NAT בין המשתמש לAPP המשתמש ישלח את אותן הודעות בעוד הAPP ישלח לIP של הנתב החיצוני של הNAT של הלקוח והנתב ינתב אותם פנימה ברשת הפנימית ללקוח.
5. הסבירו את ההבדלים בין פרוטוקול ARP ל-DNS

הבדלים:

a. ARP מקבל כתובת IP ומחזיר כתובת MAC

וDNS מקבל דומיין ומחזיר IP

b. ARP משמש לתקשורת פנימית ברשת בעוד DNS משמש לתקשורת אינטרנט בין מכשירים ברשת שונה

c. ARP בשכבת הלינק , DNS בשכבת האפליקציה

ביבליוגרפיה:

1. https://he.wikipedia.org/wiki/Domain_Name_System
2. https://he.wikipedia.org/wiki/Dynamic_Host_Configuration_Protocol
3. https://he.wikipedia.org/wiki/Address_Resolution_Protocol
4. https://he.wikipedia.org/wiki/BGP
5. https://he.wikipedia.org/wiki/Open_Shortest_Path_First
6. https://www.techtarget.com/searchnetworking/tip/BGP-vs-OSPF-When-to-use-each-protocol
7. https://inapp.com/blog/quic-vs-tcp-the-full-story/
8. https://en.wikipedia.org/wiki/TCP_Vegas
9. https://en.wikipedia.org/wiki/CUBIC_TCP
10. https://www.youtube.com/watch?v=3P2PCCNavCM
11. https://www.youtube.com/watch?v=cu1BVjAPcTU
12. https://serverfault.com/questions/729025/what-are-all-the-flags-in-a-dig-response
13. https://he.wikipedia.org/wiki/Network_Address_Translation
14. https://blog.apnic.net/2019/03/04/a-quick-look-at-quic/
15. https://en.wikipedia.org/wiki/QUIC
16. מצגות הקורס