

FINAL PROJECT

יועד תמר
213451818

ליאור וינמן
213081763

14 במרץ 2023

תוכן עניינים

3	מבוא	1
3	קבצים	1.1
4	הוראות הרצה	1.2
4	דיאגרמת היררכיות	1.3
4	דרישות	1.4
5	שרת הגדרת מארחים דינאמיים - DHCP_SERVER	2
5	צד השרת	2.1
7	צד הלקוח	2.2
8	פלט	2.3
8	שרת	2.3.1
8	לקוח	2.3.2
9	תעבורה	2.4
9	תרשים זרימה	2.5
10	מערכת שמות תחומים - DNS_SERVER	3
10	צד השרת	3.1
12	צד הלקוח	3.2
12	פלט	3.3
12	שרת	3.3.1
13	לקוח	3.3.2
13	תעבורה	3.4
14	תרשים זרימה	3.5
15	אפליקציה בחיבור TCP - APPLICATION_SERVER_TCP	4
15	צד השרת	4.1
18	צד הלקוח	4.2
22	פלט	4.3
22	שרת	4.3.1
22	לקוח	4.3.2
23	תעבורה	4.4
23	תרשים זרימה	4.5
24	אפליקציה בחיבור RUDP - APPLICATION_SERVER_RUDP	5
24	צד השרת	5.1
27	צד הלקוח	5.2
30	פלט	5.3
30	שרת	5.3.1

30 לקוח	5.3.2	
31 תעבורה	5.4	
31 ללא איבוד חבילות	5.4.1	
31 איבוד חבילות 5%	5.4.2	
31 איבוד חבילות 10%	5.4.3	
32 תרשים זרימה	5.5	
33 הסבר נוסף	5.6	
33 RELIABILITY - אמינות	5.6.1	
33 CONGESTION CONTROL - בקרת עומס	5.6.2	
33 FLOW CONTROL - בקרת זרימה	5.6.3	
34 בסיס נתונים ושאלות	6	
34 FIREBASE - בסיס הנתונים	6.1	
34 FIREBASE_SDK - קובץ המפתח	6.2	
35 מחלקת השאלות	6.3	
40 תרשים זרימה	6.4	
41 APPLICATION_GUI - גרפי	7	
41 צילומי הממשק	7.1	
46 מעבר חלקי על הקוד	7.2	
48 שאלות תיאורטיות	8	
48 שאלה 1	8.1	
48 שאלה 2	8.2	
48 שאלה 3	8.3	
49 שאלה 4	8.4	
49 שאלה 5	8.5	
50 ביבליוגרפיה	9	

1 מבוא

בפרק זה נדבר על המבוא לפרויקט שלנו ועל הלוגיस्टיקה של הקבצים וההרצה. אנו בחרנו לבנות אפליקציית SQL, אשר תהווה מערכת לניהול סטודנטים במוסד אקדמי כלשהו. המערכת שלנו תומכת בעשר השאלות הבאות:

1. הוספת סטודנט חדש למערכת.
2. מחיקת סטודנט קיים מהמערכת.
3. עדכון פרטי סטודנט הנמצא במערכת.
4. הדפסת כל הסטודנטים.
5. הדפסת סטודנט ספציפי.
6. הדפסת הסטודנטים בעלי הממוצע הגבוה ביותר.
7. הדפסת הסטודנטים בעלי הממוצע הנמוך ביותר.
8. מתן פקטור (תוספת לציון) לכלל הסטודנטים.
9. הדפסת כל הסטודנטים שנמצאים על תנאי.
10. העלאת כל הסטודנטים לשנת הלימודים הבאה.

1.1 קבצים

לפרויקט זה בסך הכל מצורפים 15 קבצים. ישנם 7 קבצי קוד (הכתוב בשפת PYTHON).

1. *DHCP_Server.py* - שרת DHCP.
2. *DNS_Server.py* - שרת DNS.
3. *Application_GUI.py* - ממשק גרפי של האפליקציה.
4. *Application_Queries.py* - מחלקת שאלות מול מסד הנתונים.
5. *Application_Server_TCP.py* - שרת אפליקציה על בסיס TCP.
6. *Application_Server_RUDP.py* - שרת אפליקציה על בסיס RUDP.
7. *FireBase\$DK.json* - מפתח עבור שימוש במסד הנתונים.

ישנם 6 קבצי הקלטות של תעבורת הרשת (הקלטות WIRESHARK).

1. *10%loss.pcapng* - הקלטה של עשר אחוז איבודי חבילות, של שרת RUDP.
2. *5%loss.pcapng* - הקלטה של חמש אחוז איבודי חבילות, של שרת RUDP.
3. *dhcp.ec.pcapng* - הקלטה של שרת DHCP.
4. *dns.ec.pcapng* - הקלטה של שרת DNS.
5. *rudp.ec.pcapng* - הקלטה של שרת RUDP, ללא איבודי חבילות.
6. *tcp.ec.pcapng* - הקלטה של שרת TCP.

ישנם 2 קבצי הסבר על כלל הפרויקט.

1. *readme.pdf* - קובץ הסברים מפורט.
2. *vid.mp4* - סרטון המדגים את הרצת הפרויקט והסבר נוסף קצר.

1.2 הוראות הרצה

את הפרויקט יש להריץ על הגרסה האחרונה של LINUX UBUNTU LTS בלבד. יש להריץ את הפרויקט על גבי מחשב עם גישה לאינטרנט. כדי להפעיל את ממשק האפליקציה יש לפתוח טרמינל המכוון לתוך התקיה עם כלל הקבצים ולהקליד את הפקודה הבאה:

```
sudo python3 Application_GUI.py
```

כעת, יש באפשרותנו להריץ את כלל התוכניות שמימשנו במטלה. אם ברצוננו להריץ את שרת ה-DHCP, נפתח טרמינל נוסף המכוון לתוך התקיה עם הקבצים ונכתוב:

```
sudo python3 DHCP_Server.py
```

ולאחר מכן נוכל לבחור במסך הקודם שנפתח בממשק, שימוש בשרת ה-DHCP. אם ברצוננו להריץ את שרת ה-DNS, נפתח טרמינל המכוון לתוך התקיה ונכתוב:

```
sudo python3 DNS_Server.py
```

כעת, אם ברצוננו להריץ את אחד משרתי האפליקציות, נריץ בהתאמה את השורה הנדרשת ונוכל להשתמש באפליקציה דרך הממשק.

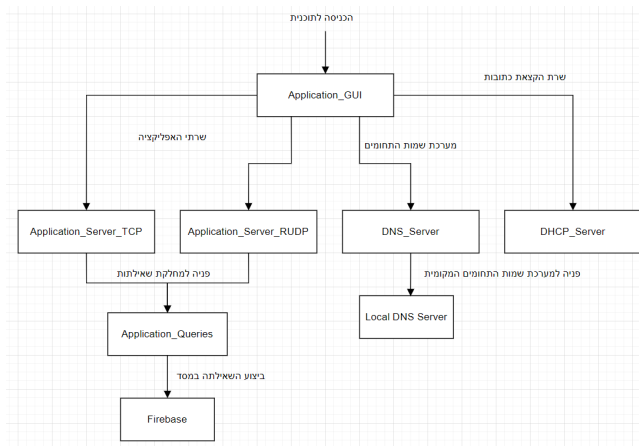
```
sudo python3 Application_Server_TCP.py
```

```
sudo python3 Application_Server_RUDP.py
```

בהרצה, חשוב לתת הרצאות מנהל (SUDO - SUPERUSER) כיוון שאנו בחלק מהשרתים משתמשים בפורטים שכבר תפוסים על ידי מערכת ההפעלה ונדרש אישור להשתמש בהם.

1.3 דיאגרמת היררכיות

כאן נראה דיאגרמה המתארת את היררכיית הקבצים בפרויקט (לא UML).



1.4 דרישות

לפני הרצת התוכנית, יש להתקין את הספריות הבאות:

`socket, time, random, json, pickle, tkinter, re, firebase - admin, future, datetime, validators, urllib3`

2 שרת הגדרת מארחים דינאמיים - DHCP_SERVER

בפרק זה נעסוק בשרת הגדרת המארחים הדינאמיים (DHCP - DYNAMIC HOST CONFIGURATION PROTOCOL). מטרתו של שרת זה הינה להקצות כתובת IP (INTERNET PROTOCOL) למחשבים חדשים אשר מתחברים לנת השרת המקומית (LAN - LOCAL AREA NETWORK). נראה כיצד מימשנו את הקצאת הכתובת, את החיבור ואת השימוש בשרת.

2.1 צד השרת

כאן נעבור על הקוד בצד השרת:

כאן נמצאים הקבועים אשר הגדרנו, הראשון - גודל הבאפר המקסימלי, השני - כתובת הלקוח, השלישי - כתובת השרת.

```
# constants
BUFFER_SIZE = 1024
DHCP_DEST = ('255.255.255.255', 9989)
ADDR = ('', 67)
```

כאן אנו פותחים שקע תקשורת עם השרת, השקע על פרוטוקול UDP.

```
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

print(f"[{datetime.now().strftime('%d-%m-%Y %H:%M:%S')}] socket is created!")

sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)

print(f"[{datetime.now().strftime('%d-%m-%Y %H:%M:%S')}] socket options set!")

sock.bind(ADDR)

print(f"[{datetime.now().strftime('%d-%m-%Y %H:%M:%S')}] socket bound on: {ADDR}")
```

כאן קורה תהליך הפעולה של שרתי DHCP, ראשית כאשר השרת מקבל תשובה מהלקוח זה נקרא גילוי השרת (DISCOVERY), לאחר מכן השרת שולח ללקוח חבילה עם נתונים (OFFER), לאחר מכן, הלקוח שולח לשרת חבילה המבוססת על הנתונים שהתקבלו קודם לכן (REQUEST) ולבסוף השרת שולח חבילה המאשרת את התהליך ולאחר חבילה זו המחשב מתחיל להשתמש בנתונים שקיבל (ACKNOWLEDGE).

```
while True:
    try:
        print(f"[{datetime.now().strftime('%d-%m-%Y %H:%M:%S')}] waiting for DHCP discovery...")
        data, address = sock.recvfrom(BUFFER_SIZE)
        print(f"[{datetime.now().strftime('%d-%m-%Y %H:%M:%S')}] received DHCP discovery!")

        print(f"[{datetime.now().strftime('%d-%m-%Y %H:%M:%S')}] DHCP offer has been sent!")
        data = self.offer_get()
        sock.sendto(data, DHCP_DEST)

        while True:
            try:
                print(f"[{datetime.now().strftime('%d-%m-%Y %H:%M:%S')}] waiting for DHCP request")

                data, address = sock.recvfrom(BUFFER_SIZE)
                print(f"[{datetime.now().strftime('%d-%m-%Y %H:%M:%S')}] received DHCP request!")

                print(f"[{datetime.now().strftime('%d-%m-%Y %H:%M:%S')}] sent DHCP packet")
                data = self.pack_get()
                sock.sendto(data, DHCP_DEST)
                break
            except (Exception, socket.error) as e:
                raise e
        except (Exception, socket.error) as e:
            raise e
```

כאן ניתן לראות את הפונקציה אשר מייצרת את החבילה של השלב השני - OFFER. תחילה ישנו פירוק של החבילה לתאים הרלוונטיים ולבסוף אנו מרכיבים את החבילה בעזרת כל התאים ושולחים אותה.

```
def offer_get(self):
    """
    This function creates a DHCP offer package to send to a DHCP client.
    returns: package (bytes): a byte string representing the DHCP offer package to be sent to the client
    """
    OP = bytes([0x01])
    HWLEN = bytes([0x01])
    HTYPE = bytes([0x01])
    HLEN = bytes([0x01])
    HOPS = bytes([0x00])
    XID = bytes([0x01, 0x02, 0x03, 0x04])
    SECS = bytes([0x00, 0x00])
    FLAGS = bytes([0x00, 0x00])
    CIADDR = bytes([0x00, 0x00, 0x00, 0x00])
    YIADDR = bytes([0x01, 0x02, 0x03, 0x04]) # 192.168.1.100
    SIADDR = bytes([0x01, 0x02, 0x03, 0x04]) # 192.168.1.1
    GIADDR = bytes([0x00, 0x00, 0x00, 0x00])
    CHADDR1 = bytes([0x00, 0x00, 0x00, 0x00])
    CHADDR2 = bytes([0x00, 0x00, 0x00, 0x00])
    CHADDR3 = bytes([0x00, 0x00, 0x00, 0x00])
    CHADDR4 = bytes([0x00, 0x00, 0x00, 0x00])
    CHADDR5 = bytes([0x00, 0x00, 0x00, 0x00])
    CHADDR6 = bytes([0x00, 0x00, 0x00, 0x00])
    MagicCookie = bytes([0x63, 0x62, 0x61, 0x62])
    DHCPOption1 = bytes([55, 1, 2]) # DHCP Offer
    DHCPOption2 = bytes([55, 4, 0x01, 0x02, 0x03, 0x04]) # 192.168.1.100 subnet mask
    DHCPOption3 = bytes([55, 4, 0x01, 0x02, 0x03, 0x04]) # 192.168.1.1 router
    DHCPOption4 = bytes([55, 4, 0x00, 0x00, 0x01, 0x01]) # 86400(1 day) IP address lease time
    DHCPOption5 = bytes([55, 4, 0x01, 0x02, 0x03, 0x04]) # DHCP server

    package = OP + HTYPE + HLEN + HOPS + XID + SECS + FLAGS + CIADDR + YIADDR + SIADDR + GIADDR + CHADDR1 + CHADDR2 + CHADDR3 + CHADDR4 + CHADDR5 + MagicCookie + DHCPOption1 + DHCPOption2 + DHCPOption3 + DHCPOption4 + DHCPOption5

    return package
```

כעת גם כאן ניתן לראות את הפונקציה אשר מייצרת את החבילה של השלב השלישי - REQUEST. כמובן, גם כאן אנו מפרקים לשדות הרלוונטיים אשר נמאים בחבילה ומרכיבים אותה.

```
def pack_get(self):
    """
    Generates a DHCP request packet.
    returns: bytes: a byte string representing the DHCP request packet.
    """
    OP = bytes([0x01])
    HWLEN = bytes([0x01])
    HTYPE = bytes([0x01])
    HLEN = bytes([0x01])
    HOPS = bytes([0x00])
    XID = bytes([0x01, 0x02, 0x03, 0x04])
    SECS = bytes([0x00, 0x00])
    FLAGS = bytes([0x00, 0x00])
    CIADDR = bytes([0x00, 0x00, 0x00, 0x00])
    YIADDR = bytes([0x01, 0x02, 0x03, 0x04])
    SIADDR = bytes([0x01, 0x02, 0x03, 0x04])
    GIADDR = bytes([0x00, 0x00, 0x00, 0x00])
    CHADDR1 = bytes([0x00, 0x00, 0x00, 0x00])
    CHADDR2 = bytes([0x00, 0x00, 0x00, 0x00])
    CHADDR3 = bytes([0x00, 0x00, 0x00, 0x00])
    CHADDR4 = bytes([0x00, 0x00, 0x00, 0x00])
    CHADDR5 = bytes([0x00, 0x00, 0x00, 0x00])
    CHADDR6 = bytes([0x00, 0x00, 0x00, 0x00])
    MagicCookie = bytes([0x63, 0x62, 0x61, 0x62])
    DHCPOption1 = bytes([55, 1, 3]) # DHCP ACK(value = 3)
    DHCPOption2 = bytes([55, 4, 0x01, 0x02, 0x03, 0x04]) # 192.168.1.100 subnet mask
    DHCPOption3 = bytes([55, 4, 0x01, 0x02, 0x03, 0x04]) # 192.168.1.1 router
    DHCPOption4 = bytes([55, 4, 0x00, 0x00, 0x01, 0x01]) # 86400(1 day) IP address lease time
    DHCPOption5 = bytes([55, 4, 0x01, 0x02, 0x03, 0x04]) # DHCP server

    package = OP + HTYPE + HLEN + HOPS + XID + SECS + FLAGS + CIADDR + YIADDR + SIADDR + GIADDR + CHADDR1 + CHADDR2 + CHADDR3 + CHADDR4 + CHADDR5 + MagicCookie + DHCPOption1 + DHCPOption2 + DHCPOption3 + DHCPOption4 + DHCPOption5

    return package
```

ההרצה של הקובץ הזה מתבצעת על ידי קריאה לפונקציה הראשית במחלקה המייצגת את השרת.

```
if __name__ == "__main__":
    dhcp_server = DHCP()
    dhcp_server.main()
```

2.2 צד הלקוח

כאן נעבור על הקוד בצד הלקוח:

כאן ניתן לראות את הקבועים שהגדרנו לשימוש הלקוח, הראשון - הפורט של השרת, השני - כתובת הלקוח, השלישי - כתובת השרת.

```
DHCP_SERVER_PORT = 67
DHCP_CLIENT_ADDR = ("0.0.0.0", 9989)
DHCP_DEST = ("<broadcast>", DHCP_SERVER_PORT)
```

כאן אנו יוצרים שקע לחיבור עם השרת. נאפשר בו את האופציות לשימוש חוזר ושליחת תפוצה ברשת המקומית.

```
def conn_dhcp(self):
    exp_flag = False

    try:
        sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        sock.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)
        sock.bind(DHCP_CLIENT_ADDR)
    except (Exception, socket.error) as e:
        messagebox.showerror("Error-Occurred", f"{e}")
        exp_flag = True
```

כאן קורה שוב תהליך ה-DHCP שהזכרנו מלעיל. לאחריו אנו מדפיסים הודעת הצלחה למשתמש, או שמחזירים הודעת שגיאה על אי הצלחה.

```
data = self.discover_get()

try:
    sock.sendto(data, DHCP_DEST)
    data, address = sock.recvfrom(BUFFER_SIZE)
except (Exception, socket.error) as e:
    messagebox.showerror("Error-Occurred", f"{e}")
    exp_flag = True

data = self.request_get()

try:
    sock.sendto(data, DHCP_DEST)
    data, address = sock.recvfrom(BUFFER_SIZE)
except (Exception, socket.error) as e:
    messagebox.showerror("Error-Occurred", f"{e}")
    exp_flag = True

if not exp_flag:
    messagebox.showinfo("Success", "DHCP configuration success!")
```

2.3 פלט

כאן נראה את הפלט מהתוכנית, הן בלקוח והן בשרת.

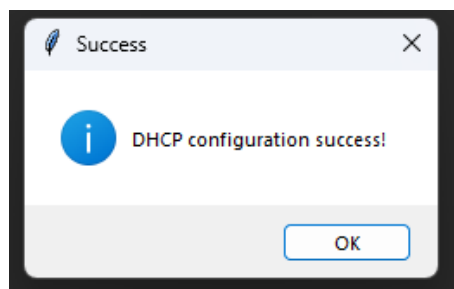
2.3.1 שרת

בצד השרת הפלט היחידי הוא הפלט הסטנדרטי בטרמינל, הפלט הוא שורות של תיעוד זמן ותיאור הפעולה שהתקיימה באותו הזמן (כך אפשר לעקוב במסודר אחרי הפעילות בשרת). נציג פלט לדוגמה: (בריבוע האדום והירוק ניתן לראות שתי פניות לשרת ה-DHCP).

```
[12-03-2023 20:09:18] socket is created!  
[12-03-2023 20:09:18] socket options set!  
[12-03-2023 20:09:18] socket bound on: ('', 67)  
[12-03-2023 20:09:18] waiting for DHCP discovery...  
[12-03-2023 20:09:25] received DHCP discovery!  
[12-03-2023 20:09:25] DHCP offer has been sent!  
[12-03-2023 20:09:25] waiting for DHCP request  
[12-03-2023 20:09:25] received DHCP request!  
[12-03-2023 20:09:25] sent DHCP packet  
[12-03-2023 20:09:25] waiting for DHCP discovery...  
[12-03-2023 20:09:27] received DHCP discovery!  
[12-03-2023 20:09:27] DHCP offer has been sent!  
[12-03-2023 20:09:27] waiting for DHCP request  
[12-03-2023 20:09:27] received DHCP request!  
[12-03-2023 20:09:27] sent DHCP packet  
[12-03-2023 20:09:27] waiting for DHCP discovery...
```

2.3.2 לקוח

בצד הלקוח אנו נקבל הודעה שאכן ההקצאה בוצעה בהצלחה.



2.4 תעבורה

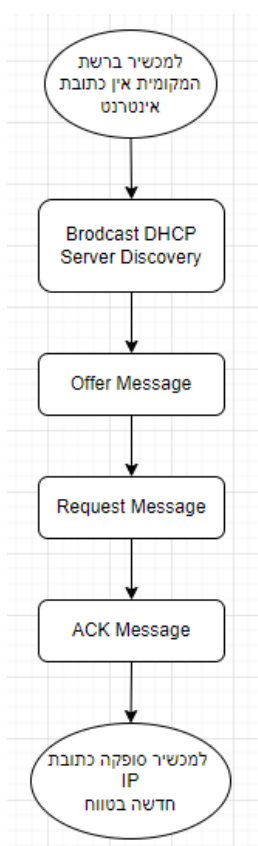
כאן נתבונן בתעבורה של שרת ה-DHCP:

ניתן לראות כי קורה תהליך ה-DHCP הסטנדרטי שתיארנו, ישנן חבילות גילוי, הצעה, בקשה ואישור כנדרש

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.2.15	255.255.255.255	DHCP	293	DHCP Discover - Transaction ID 0x3903f326
2	0.000403008	10.0.2.15	255.255.255.255	DHCP	311	DHCP Offer - Transaction ID 0x3903f326
3	0.000593935	10.0.2.15	255.255.255.255	DHCP	299	DHCP Request - Transaction ID 0x3903f326
4	0.000896612	192.168.56.1	255.255.255.255	UDP	311	63336 → 9989 Len=267
5	0.001000996	10.0.2.15	255.255.255.255	DHCP	311	DHCP ACK - Transaction ID 0x3903f326
6	0.001395064	192.168.56.1	255.255.255.255	UDP	311	63336 → 9989 Len=267

2.5 תרשים זרימה

כאן נראה כתרשים זרימה את התהליך הקורה בעת השימוש בשרת.



תהליך זה ניתן לראות גם בהקלטת התעבורה של השרת שלנו וגם בקוד עצמו, כאשר ישנה שליחה וקבלה של מידע בשרת (ושליחה וקבלה הפוכה בלקוח).

3 מערכת שמות תחומים - DNS_SERVER

בפרק זה נעסוק במערכת שמות התחומים (DNS - DOMAIN NAME SYSTEM). מטרתו של השרת היא לבצע המרות של כתובות אתרי אינטרנט (URL) מהכתובת עצמה לכתובת IP המתארת אותה.

3.1 צד השרת

כאן נעבור על הקוד בצד השרת:

כאן הגדרנו את הקבועים איתם נשתמש - הראשון כתובת השרת, השני - גודל הבאפר המקסימלי, השלישי - הודעת שגיאה גלובלית, הרביעי - מספר חיבורים מקסימליים בו זמנית.

```
# constants
SERVER_ADDR = ("127.0.0.1", 53) # server address and port
BUFFER_SIZE = 1024 # maximal size of received message
ERR = -1 # global err code
NUM_CONN = 300
```

כאן כתבנו פונקציה אשר מקבלת קישור ומחלצת את שם התחום, הריי כל קישור בנוי בצורה הבאה:

`< protocol >: // < subDom > . < Dom > . < TLD > / < path >`

כדי לגלות את כתובת ה-IP, ברצוננו לחלץ את החלק האמצע עד כמה שניתן, או להחזיר שגיאה אם זו כתובת לא חוקית.

```
def get_domain(self, url):
    """
    this function gets an url and checks if it valid
    :param url: the url to check validation
    :return: url's domain if the url valid, -1 else
    """

    if not urlparse(url).scheme: # checking if url is in legal format
        url = "http://" + url
    try:
        validation = validators.url(url) # validating url
    except TypeError:
        return ERR
    if validation:
        return urlparse(url).netloc # returning the domain name
    return ERR
```

כאן אנו בונים מילון אשר יחזיק זוגות של שם תחום והכתובת המיועדת לו ומחזיקים גם דגל המצביע על שגיאה. לאחר מכן פותחים שקע בתקשורת TCP (הבהרה: לא הייתה דרישה עם איזה סוג תקשורת להשתמש, אז אנו בחרנו בתקשורת TCP. הנימוק לכך הוא שזוהי חשיבה על אמינות בפניה לשרת, ברצוננו שכל מי שפונה לשרת תמיד יקבל מידע, לכן זוהי הבחירה).

```
dns_cache = {} # dictionary that holding pairs: (domain, ip)
exp_flag = False

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server_sock: # creating UDP socket
    try:
        server_sock.bind(SERVER_ADDR) # socket binding
    except Exception as e:
        print(f"[{datetime.now().strftime('%d-%m-%Y %H:%M:%S')}] {e}")
        print(f"[{datetime.now().strftime('%d-%m-%Y %H:%M:%S')}] server bound on: {SERVER_ADDR}")

    try:
        server_sock.listen(NUM_CONN) # socket listening
    except Exception as e:
        print(f"[{datetime.now().strftime('%d-%m-%Y %H:%M:%S')}] {e}")

    print(f"[{datetime.now().strftime('%d-%m-%Y %H:%M:%S')}] server listening for connection")
```

כאן אנו מאשרים התחברות של לקוח בפועל ומקבלים ממנו את הקישור, לאחר מכן פונים לפונקציה לחילוץ שם התחום אם יש שגיאה נחזיר זאת.

```
while True:

    client_sock, client_addr = server_sock.accept()
    print(f"[{datetime.now().strftime('%d-%m-%Y %H:%M:%S')}]] accepted connection from: {client_addr}")

    with client_sock:

        try:
            client_msg = client_sock.recv(BUFFER_SIZE) # receiving url
        except Exception as e:
            print(f"[{datetime.now().strftime('%d-%m-%Y %H:%M:%S')}]] {e}")

        print(f"[{datetime.now().strftime('%d-%m-%Y %H:%M:%S')}]] received message from: {client_addr}")

        dom = DNS.get_domain(self, client_msg.decode()) # getting domain name

        if dom == ERR: # checking if there is an error
            try:
                client_sock.sendall("Non-Existent Domain".encode()) # sending error message
            except Exception as e:
                print(f"[{datetime.now().strftime('%d-%m-%Y %H:%M:%S')}]] {e}")

            print(f"[{datetime.now().strftime('%d-%m-%Y %H:%M:%S')}]] message has been sent to: {client_addr}")

            continue
```

כאן אנו מוסיפים את שם התחום למילון (כדי שנדע לפעם הבאה להחזיר תשובה מידית) ואז מחזירים ללקוח תשובה בהתאם.

```
elif dom not in dns_cache: # checking if current domain is in cache
    try:
        dns_cache[dom] = socket.gethostbyname(dom)
    except Exception as e:
        print(f"[{datetime.now().strftime('%d-%m-%Y %H:%M:%S')}]] {e}")
        exp_flag = True

if not exp_flag:
    rep = dns_cache[dom]

    try:
        client_sock.sendall(rep.encode()) # sending the domain's ip
    except Exception as e:
        print(f"[{datetime.now().strftime('%d-%m-%Y %H:%M:%S')}]] {e}")

    print(f"[{datetime.now().strftime('%d-%m-%Y %H:%M:%S')}]] message has been sent to: {client_addr}")

else:
    try:
        client_sock.sendall("Non-Existent Domain".encode()) # sending error message
    except Exception as e:
        print(f"[{datetime.now().strftime('%d-%m-%Y %H:%M:%S')}]] {e}")

    print(f"[{datetime.now().strftime('%d-%m-%Y %H:%M:%S')}]] message has been sent to: {client_addr}")

    exp_flag = False
```

הרצת השרת הינה על ידי קריאה לפונקציה הראשית.

```
if __name__ == "__main__":
    DNS.main(__name__)
```

3.2 צד הלקוח

כאן נעבור על הקוד שכתבנו בצד הלקוח:

כאן כתבנו פונקציה אשר נקראת כאשר מבקשים לחשב כתובת, הפונקציה שולחת לשרת את הכתובת ומחזירה הודעה בהתאם ללקוח.

```
def dns_submit(self):
    data = self.dns_box.get()

    if data == "":
        messagebox.showerror("Error-Occured", "Enter non empty url!")
    else:
        try:
            sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            sock.connect(DNS_ADDR)
            sock.sendall(data.encode())
            data = sock.recv(BUFFER_SIZE).decode()
            sock.shutdown(socket.SHUT_RDWR)
            sock.close()
        except Exception as e:
            messagebox.showerror("Error-Occured", "Cannot connect to DNS server!")

    text = data if data == "Non-Existent Domain" else f"Domain's IP Address = {data}"

    self.dns_out.config(text=text)
```

3.3 פלט

כאן נציג את הפלט המתקבל בשרת ובלקוח ה-DNS.

3.3.1 שרת

הפלט בשרת הוא פלט סטנדרטי, המתעד את הפעילות שקרתה עם תיעוד הזמנים והפעולות שקרו מול אילו לקוחות.

```
[13-03-2023 11:15:26] server bound on: ('127.0.0.1', 53)
[13-03-2023 11:15:26] server listening for connection
[13-03-2023 11:15:35] accepted connection from: ('127.0.0.1', 60944)
[13-03-2023 11:15:35] received message from: ('127.0.0.1', 60944)
[13-03-2023 11:15:35] [Errno 11001] getaddrinfo failed
[13-03-2023 11:15:35] message has been sent to: ('127.0.0.1', 60944)
[13-03-2023 11:15:41] accepted connection from: ('127.0.0.1', 60947)
[13-03-2023 11:15:41] received message from: ('127.0.0.1', 60947)
[13-03-2023 11:15:41] message has been sent to: ('127.0.0.1', 60947)
```

3.3.2 לקוח

הפלט ללקוח הוא כתובת ה-IP המבוקשת או הודעת שגיאה שאין כזו, נציג את שני המקרים. כאן ניתן לראות כי הכנסנו כתובת שאינה תקינה ולכן אנו מקבלים הודעת שאין IP מתאים.



וכאן הכנסנו כתובת תקינה וקיבלנו את ה-IP המתאים עבורה.



3.4 תעבורה

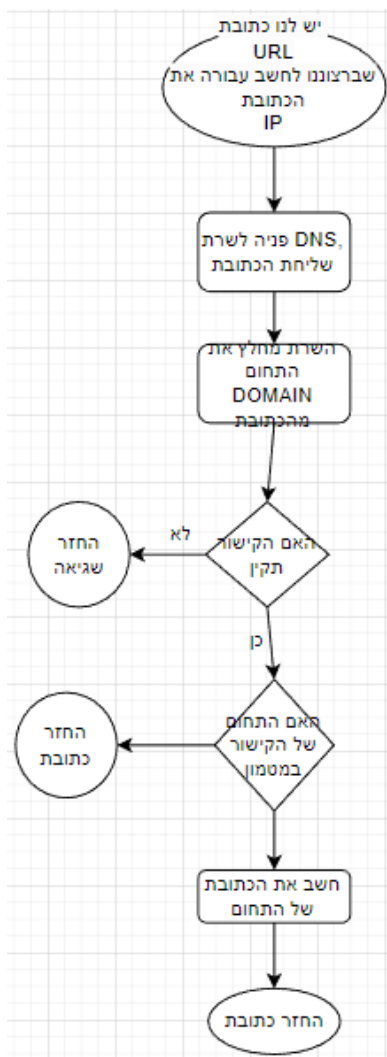
נסתכל על תעבורה לדוגמה.

כאן ביצענו שתי קריאות לשרת, הראשונה לחישוב הכתובת של GOOGLE.COM, והשנייה לחישוב ICECREAM.COM. בכל קריאה ניתן לזהות שלוש חבילות לפתיחת הקשר ועוד שלוש חבילות לסגירת הקשר בעזרת TCP. כמו כן, ניתן לזהות גם עוד ארבע חבילות לצורך חישוב כתובת ה-IP המתאימה.

1	0.00000000	127.0.0.1	127.0.0.1	TCP	76	55822 → 53 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=1340964373 TSecr=0 WS=128
2	0.000026452	127.0.0.1	127.0.0.1	TCP	76	53 → 55822 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=1340964373 TSecr=1340964373 WS=128
3	0.000049441	127.0.0.1	127.0.0.1	TCP	68	55822 → 53 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=1340964373 TSecr=1340964373
4	0.000113151	127.0.0.1	127.0.0.1	TCP	78	55822 → 53 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=10 TSval=1340964373 TSecr=1340964373 [TCP segment of a reassembled PDU]
5	0.000122502	127.0.0.1	127.0.0.1	TCP	68	53 → 55822 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=1340964374 TSecr=1340964373
6	0.005942327	127.0.0.1	127.0.0.53	DNS	83	Standard query 0x7fcc A google.com OPT
7	0.006401359	10.0.2.15	208.67.222.222	DNS	83	Standard query 0x7d88 A google.com OPT
8	0.078455828	208.67.222.222	10.0.2.15	DNS	99	Standard query response 0x7d88 A google.com A 142.251.142.206 OPT
9	0.079021376	127.0.0.53	127.0.0.1	DNS	99	Standard query response 0x7fcc A google.com A 142.251.142.206 OPT
10	0.079299274	127.0.0.1	127.0.0.1	TCP	83	53 → 55822 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=15 TSval=1340964453 TSecr=1340964373 [TCP segment of a reassembled PDU]
11	0.079332296	127.0.0.1	127.0.0.1	TCP	68	55822 → 53 [ACK] Seq=11 Ack=16 Win=65536 Len=0 TSval=1340964453 TSecr=1340964453
12	0.079511252	127.0.0.1	127.0.0.1	TCP	68	55822 → 53 [FIN, ACK] Seq=11 Ack=16 Win=65536 Len=0 TSval=1340964453 TSecr=1340964453
13	0.079648550	127.0.0.1	127.0.0.1	TCP	68	53 → 55822 [FIN, ACK] Seq=16 Ack=12 Win=65536 Len=0 TSval=1340964453 TSecr=1340964453
14	0.079713928	127.0.0.1	127.0.0.1	TCP	68	55822 → 53 [ACK] Seq=12 Ack=17 Win=65536 Len=0 TSval=1340964453 TSecr=1340964453
15	8.169830006	127.0.0.1	127.0.0.1	TCP	76	37454 → 53 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=1340972543 TSecr=0 WS=128
16	8.169851342	127.0.0.1	127.0.0.1	TCP	76	53 → 37454 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=1340972543 TSecr=1340972543 WS=128
17	8.169878652	127.0.0.1	127.0.0.1	TCP	68	37454 → 53 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=1340972543 TSecr=1340972543
18	8.169928726	127.0.0.1	127.0.0.1	TCP	80	37454 → 53 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=12 TSval=1340972543 TSecr=1340972543 [TCP segment of a reassembled PDU]
19	8.169929768	127.0.0.1	127.0.0.1	TCP	68	53 → 37454 [ACK] Seq=1 Ack=13 Win=65536 Len=0 TSval=1340972543 TSecr=1340972543
20	8.170781363	127.0.0.1	127.0.0.53	DNS	85	Standard query 0xba41 A icecream.com OPT
21	8.171174606	10.0.2.15	208.67.222.222	DNS	85	Standard query 0xba42 A icecream.com OPT
22	8.240986174	208.67.222.222	10.0.2.15	DNS	149	Standard query response 0xba42 A icecream.com A 151.101.67.10 A 151.101.131.10 A 151.101.195.10 A 151.101.3.10 OPT
23	8.241631946	127.0.0.53	127.0.0.1	DNS	149	Standard query response 0xba41 A icecream.com A 151.101.67.10 A 151.101.131.10 A 151.101.195.10 A 151.101.3.10 OPT
24	8.242467269	127.0.0.1	127.0.0.1	TCP	81	53 → 37454 [PSH, ACK] Seq=1 Ack=13 Win=65536 Len=13 TSval=1340972616 TSecr=1340972543 [TCP segment of a reassembled PDU]
25	8.242513921	127.0.0.1	127.0.0.1	TCP	68	37454 → 53 [ACK] Seq=13 Ack=14 Win=65536 Len=0 TSval=1340972616 TSecr=1340972616
26	8.242748382	127.0.0.1	127.0.0.1	TCP	68	37454 → 53 [FIN, ACK] Seq=13 Ack=14 Win=65536 Len=0 TSval=1340972616 TSecr=1340972616
27	8.242809462	127.0.0.1	127.0.0.1	TCP	68	53 → 37454 [FIN, ACK] Seq=14 Ack=14 Win=65536 Len=0 TSval=1340972616 TSecr=1340972616
28	8.242853100	127.0.0.1	127.0.0.1	TCP	68	37454 → 53 [ACK] Seq=14 Ack=15 Win=65536 Len=0 TSval=1340972616 TSecr=1340972616

3.5 תרשים זרימה

כאן נראה תרשים זרימה של המערכת. התרשים מתאר את המערכת שלנו, תחילה יש לנו כתובת שברצוננו לחשב, הלקוח שולח לשרת את הכתובת, השרת מקבל אותה ושולח לפונקציה לבדיקה וחילוץ תחום, אם החזיר שגיאה נחזיר שגיאה, אם חילץ את התחום נבדוק האם התחום כבר במטמון שהגדרנו, אם כן נחזיר מידית, אם לא נחשב ממש את הכתובת של התחום ונוסיף למטמון, לבסוף נחזיר כתובת נדרשת.



4 אפליקציה בחיבור TCP - APPLICATION_SERVER_TCP

בפרק זה נעבור על האפליקציה שכתבנו על בסיס חיבור TCP.

4.1 צד השרת

כאן נעבור על הקוד של צד השרת:

כאן הגדרנו את הקבועים שעומדים בשרת, הראשון - כתובת השרת, השני - מספר חיבורים מקסימלי, השלישי - גודל הבאפר.

```
# constants
SERVER_ADDR = ("127.0.0.1", 9090)
NUM_CONN = 300
BUFFER_SIZE = 1024
```

כאן מבוצע חיבור לבסיס הנתונים.

```
cred = credentials.Certificate("Firebase_SDK.json")
print(f"[{datetime.now().strftime('%d-%m-%Y %H:%M:%S')}]] opened database's credentials")

firebase_admin.initialize_app(cred, {"databaseURL": "https://cn-finalproject-default-rtdb.firebaseio.com/"})
print(f"[{datetime.now().strftime('%d-%m-%Y %H:%M:%S')}]] connected to database")

obj = Application_Queries.FirebaseQueries()
```

כאן אנו פותחים שקע לתקשורת עם השרת.

```
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server_sock:

    try:
        server_sock.bind(SERVER_ADDR)
        server_sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    except Exception as e:
        print(f"[{datetime.now().strftime('%d-%m-%Y %H:%M:%S')}]] {e}")

    print(f"[{datetime.now().strftime('%d-%m-%Y %H:%M:%S')}]] server bound on: {SERVER_ADDR}")

    try:
        server_sock.listen(NUM_CONN)
    except Exception as e:
        print(f"[{datetime.now().strftime('%d-%m-%Y %H:%M:%S')}]] {e}")

    print(f"[{datetime.now().strftime('%d-%m-%Y %H:%M:%S')}]] listening for incoming connections")
```

כאן אנו מאשרים התחברות בפועל של לקוח.

```
while True:
    try:
        client_sock, client_addr = server_sock.accept()
    except Exception as e:
        print(f"[{datetime.now().strftime('%d-%m-%Y %H:%M:%S')}] {e}")

    print(f"[{datetime.now().strftime('%d-%m-%Y %H:%M:%S')}] got connection from: {client_addr}")

    num = client_sock.recv(BUFFER_SIZE).decode("iso-8859-1")
    print(f"[{datetime.now().strftime('%d-%m-%Y %H:%M:%S')}] received message from: {client_addr}")
```

כאן נראה כעת את הקריאה לשאילתות, השאילתות מבוצעות במחלקה אחרת. כאן ישנה ארבע שאילתות - הוספה, מחיקה, עדכון, הדפסת כולם.

```
if num == "1": # add new student
    student_data = pickle.loads(client_sock.recv(BUFFER_SIZE))
    Application_Queries.FirebaseQueries.add_new_student(obj, student_data)
    client_sock.sendall(f"Student with id = {student_data[1]} was added to database!".encode())

elif num == "2": # delete existing student
    student_data = pickle.loads(client_sock.recv(BUFFER_SIZE))
    res = Application_Queries.FirebaseQueries.delete_existing_student(obj, student_data)
    client_sock.sendall(f"{res}".encode())
    print("data=", student_data, " res=", res)

elif num == "3": # update existing student
    student_data = pickle.loads(client_sock.recv(BUFFER_SIZE))
    res = Application_Queries.FirebaseQueries.update_existing_student(obj, student_data)
    client_sock.sendall(f"{res}".encode())

elif num == "4": # print all students
    all_students = Application_Queries.FirebaseQueries.print_all_students(obj)
    all_students = json.dumps(all_students)
    client_sock.sendall(all_students.encode())
```

כאן ניתן לראות עוד שלוש שאילתות - הדפסת יחיד, הדפסת ממוצע מקסימלי והדפסת ממוצע מינימלי. (מהתנאי עבור 7 להתעלם בבקשה).

```
elif num == "5": # print student
    student_data = pickle.loads(client_sock.recv(BUFFER_SIZE))
    student = Application_Queries.FirebaseQueries.print_single_student(obj, student_data)
    if student == -1:
        print("Student dont exist!")
        client_sock.sendall(f"{1}".encode())
    else:
        client_sock.sendall(f"{0}".encode())
        time.sleep(0.0001)
        student = json.dumps(student)
        client_sock.sendall(student.encode())

elif num == "6": # print min/max avg of students
    avg = client_sock.recv(BUFFER_SIZE).decode("iso-8859-1") # 1 - max , 0 - min
    data = Application_Queries.FirebaseQueries.print_avg_student(obj, avg)
    print("data=", data)
    data = pickle.dumps(data)
    client_sock.sendall(data)

elif num == "7": # avg of avg
    client_sock.sendall(f"{Application_Queries.FirebaseQueries.print_avg_of_avgs(obj)}".encode())
```


כאן ניתן לראות עוד שלוש שאלות - מתן פקטור, הדפסת סטודנטים על תנאי וקידום סטודנטים בשנה.

```
elif num == "8": # factor
    factor = client_sock.recv(BUFFER_SIZE).decode("iso-8859-1")
    factor = int(factor)
    if Application_Queries.FirebaseQueries.factor_students_avg(obj, factor) == -1:
        client_sock.sendall(f"error occurred!".encode())
    else:
        client_sock.sendall(f"{factor} ".encode())

elif num == "9": # condition
    cond = Application_Queries.FirebaseQueries.print_conditon_students(obj)
    if cond == -1:
        client_sock.sendall(f"error occurred!".encode())
    else:
        data = pickle.dumps(cond)
        client_sock.sendall(data)

elif num == "10":
    ny = Application_Queries.FirebaseQueries.next_year(obj)
    if ny == -1:
        client_sock.sendall(f"{-1}".encode())
    else:
        client_sock.sendall(f"{0}".encode())

client_sock.close()
```

ההרצה של השרת היא על ידי קריאה לפונקציה הראשית שלו.

```
if __name__ == "__main__":
    ServerTCP.main(__name__)
```

4.2 צד הלקוח

צד הלקוח מתוך APPLICATION_GUI (נעבור על חלקים ספציפיים):

כאן זוהי השאילתה להוספת סטודנט חדש.

```
try:
    tcp_sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    tcp_sock.connect(APP_ADDR)
    tcp_sock.sendall(f"{1}".encode())
    time.sleep(0.001)
    tcp_sock.sendall(pickle.dumps(data))
    time.sleep(0.001)
    tcp_sock.shutdown(socket.SHUT_RDWR)
    tcp_sock.close()
    messagebox.showinfo("Success", f"Student with ID: {data[1]} was added successfully!")
    frame.destroy()
except Exception as e:
    print(e)
```

כאן זוהי השאילתה למחיקת סטודנט.

```
try:
    tcp_sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    tcp_sock.connect(APP_ADDR)
    tcp_sock.sendall(f"{1}".encode())
    time.sleep(0.001)
    tcp_sock.sendall(pickle.dumps(data))
    time.sleep(0.001)
    tcp_sock.shutdown(socket.SHUT_RDWR)
    tcp_sock.close()
    messagebox.showinfo("Success", f"Student with ID: {data[1]} was added successfully!")
    frame.destroy()
except Exception as e:
    print(e)
```

כאן זוהי השאילתה לעדכון סטודנט.

```
try:
    tcp_sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    tcp_sock.connect(APP_ADDR)
    tcp_sock.sendall(f"{1}".encode())
    time.sleep(0.001)
    tcp_sock.sendall(pickle.dumps(data))
    time.sleep(0.001)
    tcp_sock.shutdown(socket.SHUT_RDWR)
    tcp_sock.close()
    messagebox.showinfo("Success", f"Student with ID: {data[1]} was added successfully!")
    frame.destroy()
except Exception as e:
    print(e)
```

כאן זוהי שאילתה להדפסת כל הסטודנטים.

```
try:
    tcp_sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    tcp_sock.connect(APP_ADDR)
    tcp_sock.sendall(f"{4}".encode())
    time.sleep(0.005)
    data = b""
    while True:
        segment = tcp_sock.recv(1024)
        if not segment:
            break
        data += segment
    time.sleep(0.01)
    tcp_sock.shutdown(socket.SHUT_RDWR)
    tcp_sock.close()
    data = json.loads(data)
    if data is None:
        res = 1
    except Exception as e:
        print(e)
if not res == 1:
    self.display_table(data)
else:
    messagebox.showerror("Error-Occurred", "there is no student in data")
```

כאן זוהי שאילתה להדפסת סטודנט ספציפי.

```
try:
    tcp_sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    tcp_sock.connect(APP_ADDR)
    tcp_sock.sendall(f"{5}".encode())
    time.sleep(0.005)
    tcp_sock.sendall(pickle.dumps(data))
    time.sleep(0.001)
    res = int(tcp_sock.recv(BUFFER_SIZE).decode("iso-8859-1"))
    if res == 1:
        messagebox.showerror("Error", "ID not found!")
    else:
        saveData = b""
        while True:
            segment = tcp_sock.recv(1024)
            if not segment:
                break
            saveData += segment
        time.sleep(0.01)
        tcp_sock.shutdown(socket.SHUT_RDWR)
        tcp_sock.close()
    except Exception as e:
        print(e)
```

כאן זוהי שאילתה להדפסת הסטודנטים עם הממוצע הגבוהה ביותר או הנמוך ביותר (אותה שאילתה עם פרמטר מועבר אחר).

```
try:
    tcp_sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    tcp_sock.connect(APP_ADDR)
    tcp_sock.sendall(f"{4}".encode())
    time.sleep(0.005)
    data = b""
    while True:
        segment = tcp_sock.recv(1024)
        if not segment:
            break
        data += segment
        time.sleep(0.01)
        tcp_sock.shutdown(socket.SHUT_RDWR)
        tcp_sock.close()
        data = json.loads(data)
        if data is None:
            res = 1
    except Exception as e:
        print(e)
if not res == 1:
    self.display_table(data)
else:
    messagebox.showerror("Error-Occurred", "there is no student in data")
```

כאן זוהי שאילתה למתן פקטור לסטודנטים.

```
try:
    tcp_sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    tcp_sock.connect(APP_ADDR)
    tcp_sock.sendall(f"{4}".encode())
    time.sleep(0.005)
    data = b""
    while True:
        segment = tcp_sock.recv(1024)
        if not segment:
            break
        data += segment
        time.sleep(0.01)
        tcp_sock.shutdown(socket.SHUT_RDWR)
        tcp_sock.close()
        data = json.loads(data)
        if data is None:
            res = 1
    except Exception as e:
        print(e)
if not res == 1:
    self.display_table(data)
else:
    messagebox.showerror("Error-Occurred", "there is no student in data")
```

כאן זוהי שאילתה להדפסת סטודנטים על תנאי.

```
try:
    tcp_sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    tcp_sock.connect(APP_ADDR)
    tcp_sock.sendall(f"{4}".encode())
    time.sleep(0.005)
    data = b""
    while True:
        segment = tcp_sock.recv(1024)
        if not segment:
            break
        data += segment
    time.sleep(0.01)
    tcp_sock.shutdown(socket.SHUT_RDWR)
    tcp_sock.close()
    data = json.loads(data)
    if data is None:
        res = 1
    except Exception as e:
        print(e)
if not res == 1:
    self.display_table(data)
else:
    messagebox.showerror("Error-Occurred", "there is no student in data")
```

כאן זוהי שאילתה להעלאת סטודנטים בשנה.

```
try:
    tcp_sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    tcp_sock.connect(APP_ADDR)
    tcp_sock.sendall(f"{4}".encode())
    time.sleep(0.005)
    data = b""
    while True:
        segment = tcp_sock.recv(1024)
        if not segment:
            break
        data += segment
    time.sleep(0.01)
    tcp_sock.shutdown(socket.SHUT_RDWR)
    tcp_sock.close()
    data = json.loads(data)
    if data is None:
        res = 1
    except Exception as e:
        print(e)
if not res == 1:
    self.display_table(data)
else:
    messagebox.showerror("Error-Occurred", "there is no student in data")
```

4.3 פלט

כאן נעבור על הפלט של האפליקציה בתקשורת TCP.

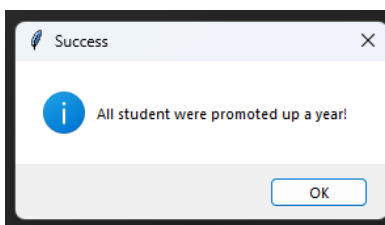
4.3.1 שרת

הפלט בשרת הוא פלט סטנדרטי בטרמינל המתעד פעילות, נראה פלט לדוגמה.

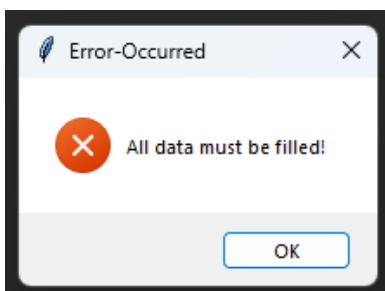
```
[14-03-2023 00:49:33] opened database's credentials  
[14-03-2023 00:49:33] connected to database  
[14-03-2023 00:49:33] server bound on: ('127.0.0.1', 9090)  
[14-03-2023 00:49:33] listening for incoming connections  
[14-03-2023 00:49:45] got connection from: ('127.0.0.1', 60865)  
[14-03-2023 00:49:45] received message from: ('127.0.0.1', 60865)  
[14-03-2023 00:49:51] got connection from: ('127.0.0.1', 60871)  
[14-03-2023 00:49:51] received message from: ('127.0.0.1', 60871)
```

4.3.2 לקוח

בלקוח הפלט המתקבל הוא אישור על הצלחה או אי-הצלחה בביצוע הפעולה הרצויה, נראה שני פלטים לדוגמה. פלט של הצלחה:



פלט של כשלון:



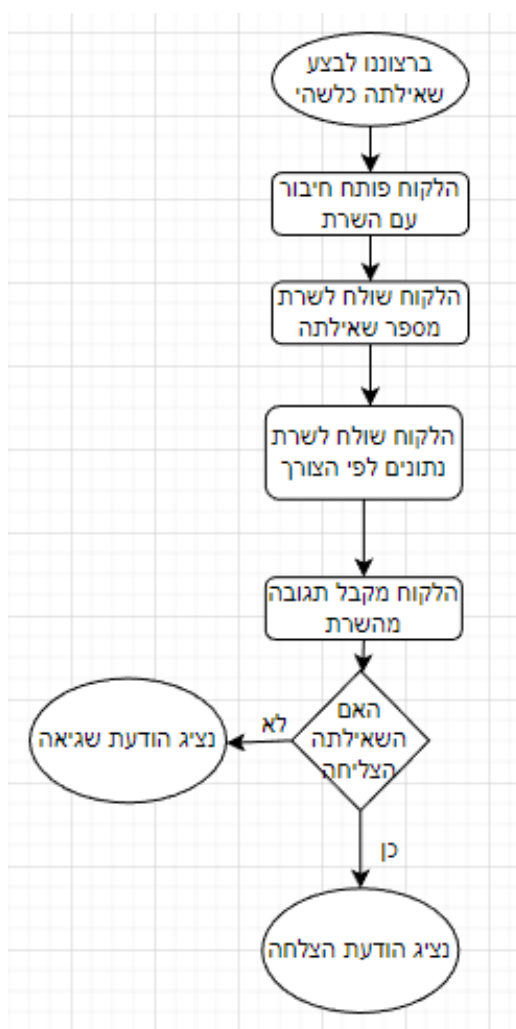
4.4 תעבורה

נתבונן על תעבורה לדוגמה, כאן ניתן לראות תקשורת שלמה עם השרת - עבור שאילתה כלשהי. יש את פתיחת הקשר, שליחת מספר שאילתה לשרת, קבלת תשובה מהשרת, סגירת הקשר. כמובן אישורי קבלה בין לבין.

1	0.000000000	127.0.0.1	127.0.0.1	TCP	74 56208 → 9090	[SYN] Seq=0 Win=65536 Len=0 MSS=65495 SACK_PERM TSval=1340050612 TSecr=0 WS=128
2	0.000016290	127.0.0.1	127.0.0.1	TCP	74 9090 → 56208	[SYN, ACK] Seq=0 Ack=1 Win=65536 Len=0 MSS=65495 SACK_PERM TSval=1340050612 TSecr=1340050612 WS=128
3	0.000030572	127.0.0.1	127.0.0.1	TCP	66 56208 → 9090	[ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=1340050612 TSecr=1340050612
4	0.000076254	127.0.0.1	127.0.0.1	TCP	67 56208 → 9090	[PSH, ACK] Seq=1 Ack=1 Win=65536 Len=1 TSval=1340050612 TSecr=1340050612
5	0.000082879	127.0.0.1	127.0.0.1	TCP	66 9090 → 56208	[ACK] Seq=1 Ack=2 Win=65536 Len=0 TSval=1340050612 TSecr=1340050612
6	1.312503117	127.0.0.1	127.0.0.1	TCP	527 9090 → 56208	[PSH, ACK] Seq=1 Ack=2 Win=65536 Len=461 TSval=1340051924 TSecr=1340050612
7	1.312528386	127.0.0.1	127.0.0.1	TCP	66 56208 → 9090	[ACK] Seq=2 Ack=462 Win=65152 Len=0 TSval=1340051924 TSecr=1340051924
8	1.312554030	127.0.0.1	127.0.0.1	TCP	66 9090 → 56208	[FIN, ACK] Seq=462 Ack=2 Win=65536 Len=0 TSval=1340051924 TSecr=1340051924
9	1.322823300	127.0.0.1	127.0.0.1	TCP	66 56208 → 9090	[FIN, ACK] Seq=2 Ack=463 Win=65536 Len=0 TSval=1340051935 TSecr=1340051924
10	1.322868547	127.0.0.1	127.0.0.1	TCP	66 9090 → 56208	[ACK] Seq=463 Ack=3 Win=65536 Len=0 TSval=1340051935 TSecr=1340051935

4.5 תרשים זרימה

נציג תרשים זרימה של המערכת, אצלו במערכת - הלקוח לוחץ על כפתור של שאילתה, לאחר מכן, ממלא נתונים אם יש צורך ושולח אותם לשרת, בסוף מקבל פלט בהתאם - הודעת שגיאה או הודעת הצלחה (בשאילתות בהן אין הודעת הצלחה מפורשת, הדפסת הטבלה מהווה הודעת הצלחה).



5 אפליקציה בחיבור RUDP - APPLICATION_SERVER_RUDP

בפרק זה נראה את האפליקציה בחיבור בפרוטוקול RUDP.

5.1 צד השרת

כאן נעבור על החלק של צד השרת:

כאן אנו מגדירים את הקבועים לשימושנו, יעזרו לנו להגדיר את כל מה שנזדקק לו בבניית הפרוטוקול.

```
# Constants
SERVER_IP = '127.0.0.1'
SERVER_PORT = 1235
BUFFER_SIZE = 1024
WINDOW_SIZE = 4
PACKET_COUNT = 20
INITIAL_CWND = 1
THRESHOLD = 8
TIMEOUT = 3
```

כאן אנו פותחים שקע לתקשורת ומתחברים למסד הנתונים.

```
cred = credentials.Certificate("Firebase_SDK.json")
print(f"[{datetime.now().strftime('%d-%m-%Y %H:%M:%S')}]] opened database's credentials")

firebase_admin.initialize_app(cred, {"databaseURL": "https://cn-finalproject-default-rtdb.firebaseio.com/"})
print(f"[{datetime.now().strftime('%d-%m-%Y %H:%M:%S')}]] connected to database")

obj = Application_Queries.FirebaseQueries()

# Create UDP socket
server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
server_socket.bind((SERVER_IP, SERVER_PORT))

# Initialize variables for sequence numbers, window size, and packets buffer
seq_num = 1
window_size = INITIAL_CWND
packets_buffer = {}
received = {}
last_seq = 0
```


כאן אנו מתחילים לקבל את המידע, החבילות שאנו מקבלים הן חבילות מהצורה: $SeqNum : Data$ לכן אנו גם מפצלים לפי הנקודתיים. כמו כן, מיד אנו גם שולחים ack .

```
# Receive packets
while True:
    # Receive packet
    # print(packets_buffer)
    packet, client_address = server_socket.recvfrom(BUFFER_SIZE)
    packet_data = packet.decode()
    print(packet_data)
    packet_seq_num = int(packet_data.split(':')[0])
    data = packet_data.split(':')[1]
    print(f'Received packet {packet_seq_num}')
    print(f'data: {data}')

    # Add packet to buffer
    packets_buffer[packet_seq_num] = packet

    # Send ACK for received packet
    send_ack(packet_seq_num, data)

    # Update received dictionary
    received[packet_seq_num] = True
```

כעת אנו בודקים האם יש חבילות שהגיעו לא במקומן או חבילות נוספות שאנו יכולים לשלוח.

```
# Check if there are packets in the buffer that can be delivered
while packets_buffer.get(seq_num) and seq_num <= window_size:
    # Deliver packet
    delivered_packet = packets_buffer.pop(seq_num)
    print(f'Delivered packet {seq_num}')
    # Update sequence number
    seq_num += 1

# Check for out-of-order packets and send cumulative ACKs
if last_seq + 1 != seq_num - 1:
    # Find the next sequence number that has not been received
    print(f'miss: ')
    missing_seq_num = seq_num
    while missing_seq_num in received:
        missing_seq_num += 1
    # Send a cumulative ACK for all packets up to the missing sequence number
    num_acks = missing_seq_num - window_size
    print(f'seq_num{seq_num}, num_acks {num_acks}')
    send_ack(seq_num, data, num_acks)

last_seq += 1
```

כעת, לבסוף אנו מעדכנים את גודל החלון.

```
# Update window size and threshold
if len(received) % WINDOW_SIZE == 1 or len(received) % WINDOW_SIZE == 0:
    # Update window size and threshold based on ACKs
    if window_size < THRESHOLD:
        # Slow start phase
        window_size *= 2
    else:
        # Congestion avoidance phase
        window_size += 1 / window_size
    # Reset received dictionary
    received = {}
elif len(packets_buffer) == WINDOW_SIZE:
    # Timeout, retransmit unacknowledged packets
    print('Timeout, retransmitting packets')
    window_size = max(window_size / 2, 1)
    for seq_num, packet in packets_buffer.items():
        server_socket.sendto(packet, client_address)
        print(f'Retransmitted packet {seq_num}')
    # Reset received dictionary
    received = {}
```

כאן אנו בונים את הפונקציה לשליחת *ack*, ברגע שקיבלנו חבילות. חבילת *ack* שאנו שולחים היא מהצורה: *SeqNum : ACK*. כמובן, כאן אנו גם בודקים האם כבר קיבלנו את החבילה הזו ולפי המנגנון של TCP - אם קיבלנו אותה כבר 3 פעמים אנו נבצע RETRANSMITION. נציין כי השליחה חזרה של המידע, מתבצעת בפרוטוקול TCP. למה? השרת כולו מבוצע בפרוטוקול RUDP, כל המידע הנכנס הוא בפרוטוקול הזה. כדי גם לשלוח בחזרה ללקוח את המידע ב-RUDP, היינו צריכים להפוך גם אותו לשרת אשר יודע לטפל בקבלת חבילות כאלו, מכיוון שהלקוח שלנו הוא אינו שרת, וכדי שהמידע המעובד יישלח בצורה אמינה וזאת לאחר שלפני העיבוד - כולו נשלח בפרוטוקול RUDP, אנו שולחים אותו ב-TCP בפרויקט זה ניתנה לנו יד חופשית לצורך ביצוע המימוש, כך אנו החלטנו לבצע זאת - שכן, כל המידע אכן מתקבל ב-RUDP.

```
# Define a function to send ACKs
def send_ack(seq_num, data, num_acks=1):
    # Check if the ACK is a duplicate
    if seq_num in received and received[seq_num] == num_acks:
        # Increment the counter for the sequence number
        received[seq_num] += 1
        # If the counter reaches a threshold, assume packet loss and retransmit the packet
        if received[seq_num] == 3:
            # Retransmit the packet
            server_socket.sendto(packets_buffer[seq_num], client_address)
            print(f'Retransmitted packet {seq_num}')
            # Reset the counter for the sequence number
            received[seq_num] = 0
    else:
        # Send the ACK
        ack_packet = f'{seq_num}:ACK'.encode()
        server_socket.sendto(ack_packet, client_address)
        # Update the received dictionary with the number of ACKs received for the sequence number
        received[seq_num] = num_acks
        # Create TCP socket
        data_sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        # ""
        # Connect to server
        data_sock.connect((SERVER_IP, SERVER_PORT + 1))

        # Send message to server
        return_data = handle_data(data)
        message = f'{return_data}'
        data_sock.send(message.encode())
```

כמו כן, כעת יש לנו את הפונקציה של הטיפול בחבילות. החבילות האלה נשלחות בצורה הבאה: $SeqNum\$Data1\$Data2\$...$ חוץ מהפורמט הזה, הפונקציה זהה לחלוטין כמו ל-TCP.

```
def handle_data(data):
    obj = Application_Queries.FirebaseQueries()
    num = int(data.split('$')[0])

    if num == 1: # add new student
        #student_data = pickle.loads(client_sock.recv(BUFFER_SIZE))
        #Application_Queries.FirebaseQueries.add_new_student(obj, student_data)
        #client_sock.sendall(f"Student with id = {student_data[1]} was added to database!".encode())
        string_after_first_dollar = data.split('$')[1:]
        Application_Queries.FirebaseQueries.add_new_student(obj, string_after_first_dollar)
        return f"{0}".encode("iso-8859-1")

    elif num == 2: # delete existing student
        # student_data = pickle.loads(stud_data)
        string_after_first_dollar = data.split('$')[1:]
        res = Application_Queries.FirebaseQueries.delete_existing_student(obj, string_after_first_dollar)
        print("data=", string_after_first_dollar, " res=", res)
        return f"{res}".encode("iso-8859-1")

    elif num == 3: # update existing student
        string_after_first_dollar = data.split('$')[1:]
        res = Application_Queries.FirebaseQueries.update_existing_student(obj, string_after_first_dollar)
        print("data=", string_after_first_dollar, " res=", res)
        return f"{res}".encode("iso-8859-1")
```

5.2 צד הלקוח

בצד הלקוח ישנן שתי פונקציות אשר נציג אותן, כל שאר הדברים זהים כמעט לגמריי למימוש ב-TCP כדי לשלוח לשרת חבילות בפורמט אותו הצגנו קודם, בנינו פונקציה אשר מקבלת מידע ומייצרת את החבילה מהפורמט הרצוי.

```
def send_data(data):
    global seq_num
    # Divide the message into chunks
    packet = f"{seq_num}:{data}".encode()
    print(packet)
    return send_packet(packet)
```

כעת, בנינו פונקציה נוספת אשר בפועל שולחת לשרת את החבילות בפרוטוקול RUDP. כאן אנחנו מטפלים במספר הרצף של החבילות ומטפלים בזמני השליחה.

```
def send_packet(packet):
    global seq_num
    global window_size
    global rudp_server_address
    global client_socket
    client_socket.sendto(packet, rudp_server_address)
    print(f"Sent packet {seq_num}")
    # Add packet to buffer
    packets_buffer[seq_num] = packet

    # Update sequence number and window size
    seq_num += 1
    if seq_num <= window_size:
        window_size = min(window_size * 2, WINDOW_SIZE)
    else:
        window_size += 1 / WINDOW_SIZE

    # Set timeout based on estimated RTT
    start_time = time.time()
    client_socket.settimeout(TIMEOUT)
```

כאן אנו מקבלים את ה־ACK, עובר החבילות ובודקים האם עלינו לבצע RETRANSMISSION, כדי לשמור על האמינות.

```
while True:
    # Receive ACK
    try:
        ack_packet, _ = client_socket.recvfrom(BUFFER_SIZE)
        end_time = time.time()
        rtt = end_time - start_time
        client_socket.settimeout(max(TIMEOUT - rtt, 0))
    except socket.timeout:
        # Timeout, retransmit unacknowledged packets
        print('Timeout, retransmitting packets')
        window_size = max(window_size / 2, 1)
        for seq, pkt in packets_buffer.items():
            client_socket.sendto(pkt, rudp_server_address)
            print(f'Retransmitted packet {seq}')
        break
```

כאן אנחנו בודקים את חבילות ה-ACK ומבצעים עדכון נדרש.

```
# Parse ACK
ack_data = ack_packet.decode()
print(f'ack_data: {ack_data}')
ack_seq_num = int(ack_data.split(':')[0])
print(f'Received ACK {ack_seq_num}')

# Update packets buffer and window size
if ack_seq_num in packets_buffer:
    del packets_buffer[ack_seq_num]
    if ack_seq_num <= window_size:
        window_size = min(window_size + 1, WINDOW_SIZE)
else:
    # Duplicate ACK, ignore
    continue
```

כאן אנו מבצעים טיפול בחבילות שהגיעו לא בסדר שלהן, כמובן גם מבצעים בדיקה האם הגיע ACK לכל החבילות ומקבלים מידע נוסף מהלקוח.

```
# Handle out-of-order ACKs
while len(packets_buffer) > 0:
    if min(packets_buffer.keys()) == ack_seq_num + 1:
        break
    seq = min(packets_buffer.keys())
    pkt = packets_buffer[seq]
    client_socket.sendto(pkt, rudp_server_address)
    print(f'Retransmitted packet {seq}')
    del packets_buffer[seq]

# Check if all packets have been acknowledged
if len(packets_buffer) == 0:
    # Accept connection from client
    client_data_socket, client_data_address = data_sock.accept()

    # Receive data from client
    data = b""
    while True:
        segment = client_data_socket.recv(1024)
        if not segment:
            break
        data += segment
    print(f'Received: {data}')
    return data
```


5.4 תעבורה

5.4.1 ללא איבוד חבילות

כאן ניתן לראות את החיבור של השרת RUDP, יש את חיבורי TCP כי שלחנו חזרה את המידע בפרוטוקול זה, למטה ניתן לראות שני בקשות UDP שזהו למעשה ה־RUDP שמימשנו. ניתן לראות שם חבילה שהיא ACK מספר 2.

No.	Time	Source	Destination	Protocol	Length	Info
4	0.000258352	127.0.0.1	127.0.0.1	TCP	74	1236 → 59326 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=1340321907 TSecr=1
5	0.000273678	127.0.0.1	127.0.0.1	TCP	66	59326 → 1236 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=1340321907 TSecr=1340321907
6	0.992397527	127.0.0.1	127.0.0.1	TCP	527	59326 → 1236 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=461 TSval=1340322099 TSecr=1340321907
7	0.992457285	127.0.0.1	127.0.0.1	TCP	66	1236 → 59326 [ACK] Seq=1 Ack=462 Win=65024 Len=0 TSval=1340322099 TSecr=1340322099
8	0.992926354	127.0.0.1	127.0.0.1	TCP	66	59326 → 1236 [FIN, ACK] Seq=462 Ack=1 Win=65536 Len=0 TSval=1340322099 TSecr=1340322099
9	0.993000719	127.0.0.1	127.0.0.1	TCP	66	1236 → 59326 [FIN, ACK] Seq=1 Ack=463 Win=65536 Len=0 TSval=1340322900 TSecr=1340322099
10	0.993070456	127.0.0.1	127.0.0.1	TCP	66	59326 → 1236 [ACK] Seq=463 Ack=2 Win=65536 Len=0 TSval=1340322900 TSecr=1340322900
11	3.478284455	127.0.0.1	127.0.0.1	UDP	51	58740 → 1235 Len=0
12	3.478584920	127.0.0.1	127.0.0.1	UDP	47	1235 → 58740 Len=5
13	3.47884634	127.0.0.1	127.0.0.1	TCP	74	59334 → 1236 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=1340325385 TSecr=0 WS=128
14	3.478900329	127.0.0.1	127.0.0.1	TCP	74	1236 → 59334 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=1340325385 TSecr=1
15	3.47952022	127.0.0.1	127.0.0.1	TCP	66	59334 → 1236 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=1340325385 TSecr=1340325385
• Frame 12: 47 bytes on wire (376 bits), 47 bytes captured (376 bits) on interface lo, id 0						
• Ethernet II, Src: 00:00:00:00:00:00 (00:00:00:00:00:00), Dst: 00:00:00:00:00:00 (00:00:00:00:00:00)						
• Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1						
• User Datagram Protocol, Src Port: 1235, Dst Port: 58740						
• Data (5 bytes)						
0000	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00E
0010	00 21 f3 a4 40 00 00 11	49 25 7f 00 00 01 7f 00
0020	00 01 04 d3 e5 7a 41 43 40	fe 20 32 3a 41 43 40 4b

5.4.2 איבוד חבילות 5%

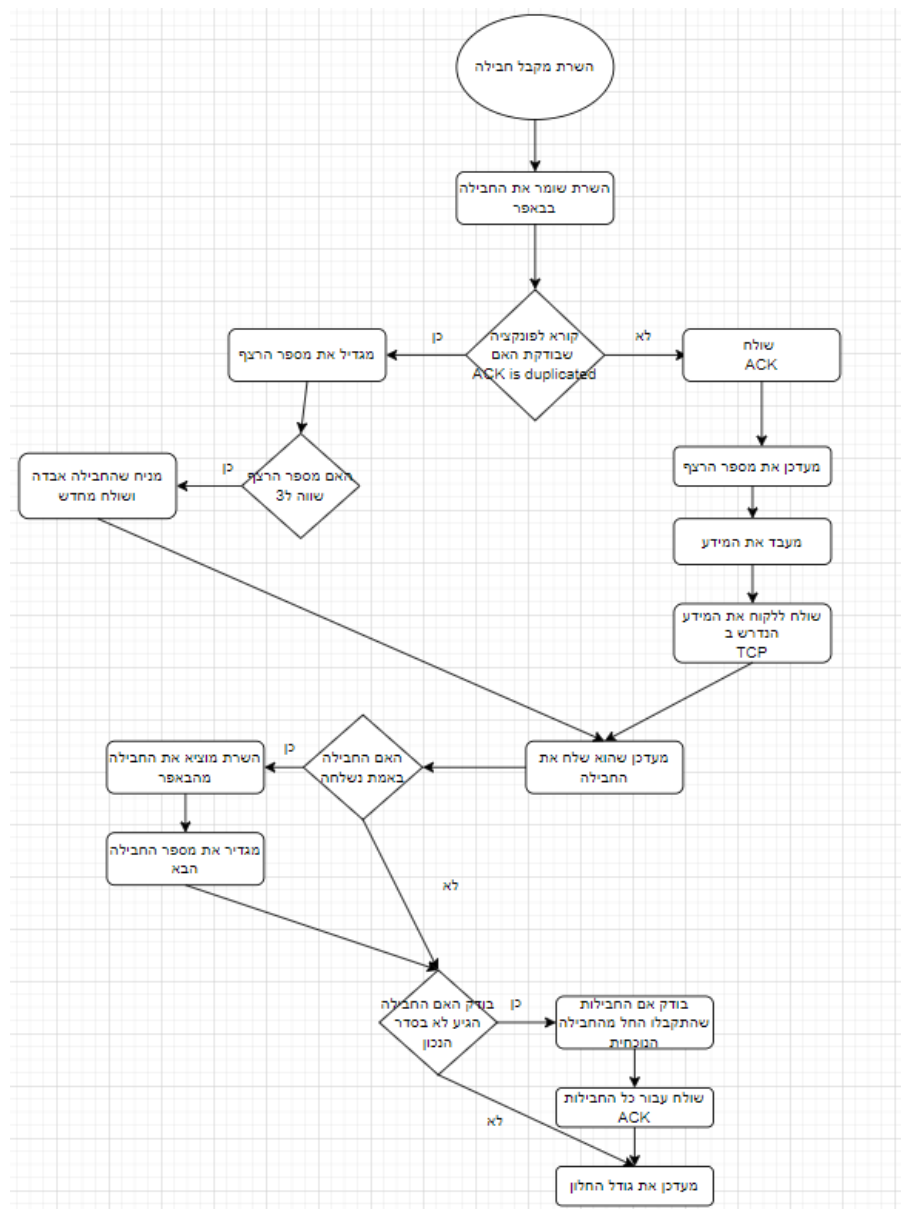
כאן ניתן לראות שוב אותם פעולות, על כל שתי חבילות ב־RUDP יש פתיחות קשר ב־TCP.

No.	Time	Source	Destination	Protocol	Length	Info
26	4.368811806	127.0.0.1	127.0.0.1	TCP	66	1236 → 49688 [FIN, ACK] Seq=1 Ack=463 Win=65536 Len=0 TSval=1341719786 TSecr=1341716447
27	4.368829683	127.0.0.1	127.0.0.1	TCP	66	49688 → 1236 [ACK] Seq=463 Ack=2 Win=65536 Len=0 TSval=1341719786 TSecr=1341719786
28	4.663456775	127.0.0.1	127.0.0.1	TCP	259	49704 → 1236 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=193 TSval=1341720080 TSecr=1341719785
29	4.663490256	127.0.0.1	127.0.0.1	TCP	66	1236 → 49704 [ACK] Seq=1 Ack=194 Win=65408 Len=0 TSval=1341720080 TSecr=1341720080
30	4.663899805	127.0.0.1	127.0.0.1	TCP	66	49704 → 1236 [FIN, ACK] Seq=194 Ack=1 Win=65536 Len=0 TSval=1341720081 TSecr=1341720080
31	4.708571914	127.0.0.1	127.0.0.1	TCP	66	1236 → 49704 [ACK] Seq=1 Ack=195 Win=65408 Len=0 TSval=1341720125 TSecr=1341720081
32	8.500090512	127.0.0.1	127.0.0.1	UDP	48	37320 → 1235 Len=0
33	8.501177212	127.0.0.1	127.0.0.1	UDP	47	1235 → 37320 Len=5
34	8.501284693	127.0.0.1	127.0.0.1	TCP	74	59480 → 1236 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=1341723918 TSecr=0 WS=128
35	8.501303559	127.0.0.1	127.0.0.1	TCP	74	1236 → 59480 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=1341723918 TSecr=1
36	8.501322967	127.0.0.1	127.0.0.1	TCP	66	59480 → 1236 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=1341723918 TSecr=1341723918
37	8.502990329	127.0.0.1	127.0.0.1	TCP	66	1236 → 49704 [FIN, ACK] Seq=1 Ack=195 Win=65536 Len=0 TSval=1341723920 TSecr=1341720081
• Frame 33: 47 bytes on wire (376 bits), 47 bytes captured (376 bits) on interface lo, id 0						
• Ethernet II, Src: 00:00:00:00:00:00 (00:00:00:00:00:00), Dst: 00:00:00:00:00:00 (00:00:00:00:00:00)						
• Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1						
• User Datagram Protocol, Src Port: 1235, Dst Port: 37320						
• Data (5 bytes)						
0000	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00E
0010	00 21 aa f8 40 00 00 11	91 d1 7f 00 00 01 7f 00
0020	00 01 04 d3 91 c8 00 0d	fe 20 33 3a 41 43 4b

5.4.3 איבוד חבילות 10%

כאן מבחינת איבוד החבילות ב־TCP הפרוטוקול עצמו מטפל בזה אוטומטית אך באיבוד החבילות ב־RUDP ניתן לראות טיפול ידני בהמשך ההקלטה.

No.	Time	Source	Destination	Protocol	Length	Info
57	19.9834649	127.0.0.1	127.0.0.1	TCP	74	45184 → 1236 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=1341894142 TSecr=0 WS=128
58	19.9835188	127.0.0.1	127.0.0.1	TCP	74	1236 → 45184 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=1341894142 TSecr=1
59	19.9835783	127.0.0.1	127.0.0.1	TCP	66	45184 → 1236 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=1341894142 TSecr=1341894142
60	0.000000000	127.0.0.1	127.0.0.1	TCP	66	1236 → 45184 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=1341894355 TSecr=1341894142
61	20.1909714	127.0.0.1	127.0.0.1	TCP	78	[TCP Window Update] 1236 → 45184 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=1341894355 TSecr=1341894142
62	20.1970064	127.0.0.1	127.0.0.1	TCP	260	[TCP Out-Of-Order] 45184 → 1236 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=194 TSval=1341894355 TSecr=1341894142
63	20.1970242	127.0.0.1	127.0.0.1	TCP	66	1236 → 45184 [ACK] Seq=1 Ack=199 Win=65408 Len=0 TSval=1341894355 TSecr=1341894355
64	21.4077430	127.0.0.1	127.0.0.1	UDP	51	40240 → 1235 Len=0
65	21.4080684	127.0.0.1	127.0.0.1	UDP	47	1235 → 40240 Len=5
66	21.4081850	127.0.0.1	127.0.0.1	TCP	74	45196 → 1236 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=1341895620 TSecr=0 WS=128
67	21.4082036	127.0.0.1	127.0.0.1	TCP	74	1236 → 45196 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=1341895620 TSecr=1
68	21.4082228	127.0.0.1	127.0.0.1	TCP	66	45196 → 1236 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=1341895627 TSecr=1341895620
• Frame 65: 47 bytes on wire (376 bits), 47 bytes captured (376 bits) on interface lo, id 0						
• Ethernet II, Src: 00:00:00:00:00:00 (00:00:00:00:00:00), Dst: 00:00:00:00:00:00 (00:00:00:00:00:00)						
• Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1						
• User Datagram Protocol, Src Port: 1235, Dst Port: 40240						
• Data (5 bytes)						
0000	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00E
0010	00 21 ce 6e 40 00 00 11	6e 5b 7f 00 00 01 7f 00
0020	00 01 04 d3 9d 36 00 0d	fe 20 37 3a 41 43 4b



5.6 הסבר נוסף

5.6.1 אמינות - RELIABILITY

השרת שומר על אמינות במספר אופנים:

1. שמירה על סדר הגעת החבילות - השרת עוקב אחר מספר הרצף של כל חבילה שהתקבלה ושומר זאת במילון, אם חבילה מתקבלת לא בסדר שלה אז השרת שולח ACK מצטבר עד שהחבילה האחרונה מתקבלת. השרת ממשיך לשלוח ACK עבור כל חבילה חסרה עד לקבלת החבילה הבאה ברצף וכך הוא מקבל את כל החבילות בסדר הנכון ושומר על כך.
2. שמירה על כפל חבילות - השרת עוקב אחר מספר אישורי ההגעה ACK שהגיעו עבור כל חבילה במילון ואם הוא מקבל מספר אישורי קבלה כפולים עבור אותה חבילה הוא מניח שהחבילה אבדה ומבקש אותה מחדש.

5.6.2 בקרת עומס - CONGESTION CONTROL

בקרת העומס באה לידי ביטוי כך - השרת משתמש בהזאת החלון כדי לשלוט בקצת שליחת החבילות ללקוח. הוא משתמש בחלון הראשוני בגודל 1 ומגדיל את החלון בהתבסס על מספר אישורי הקבלה ACK, שהגיעו מהלקוח וזה תוך שימוש בשלה התחלה איטי והמנעות מעומס. אם גודל החלון הגיע למקסימום, הוא מתחיל לטפל בעומס ומגדיל את החלון בהדרגה.

5.6.3 בקרת זרימה - FLOW CONTROL

בקרת הזרימה באה לידי ביטוי כך - בשרת משתמש בגודל החלון כדי לשלוט במספר החבילות שנמצאות בדרך ליעד בכל זמן נתון. הוא שולח רק עד מקסימום חבילות בגודל החלון עד ההמתנה ל-ACK וזה לפני שליחת עוד חבילות. כלומר, רוחב הפס אינו קבוע ותלוי בגודל החלון.

6 בסיס נתונים ושאלות

בפרק זה נעבור על בסיס הנתונים בו השתמשנו ונסביר כיצד השתמשנו בו וכיצד השאלות בנויות ואיך מתבצעות הפניות (שליחות ושליפות נתונים) לבסיס הנתונים.

6.1 בסיס הנתונים - FIREBASE

אני השתמשנו בבסיס נתונים הנקרא FIREBASE שהוא בסיס נתונים של גוגל הנמצא על ענן. רעיון האפליקציה שלנו היה מאגר נתונים לניהול סטודנטים. אנו מחזיקים מחלקות, כאשר בכל מחלקה יש שלוש שנות לימוד. בכל שנת לימוד, נמצאים סטודנטים, כאשר לכל סטודנט יש מזהה יחודי (תעודת זהות) ובנוסף לכך, יש עוד שתי קטגוריות מידע. הראשונה היא מידע אקדמי - שם נמצאים פרטים כמו ממוצע, תואר, מסלול, האם נמצא על תנאי. ובנוסף, מידע אישי שם נמצא מידע כמו - שם פרטי, שם משפחה, כתובת מייל, מספר טלפון ותעודת זהות שלו. נציג את מאגר הלימודים עם סטודנט כלשהו לדוגמה:

כאן ניתן לראות שכרגע במאגר יש מחלקה אחת שהיא CS (COMPUTER SCIENCE) בתוך המחלקה נמצאים רק שנה ג' ובשנה ג' ישנו רק סטודנט יחיד, ניתן לראות עליו את כל הפרטים שהזכרנו.



בסיס הנתונים מבוסס על NoSQL ושמירת הנתונים היא על ידי שימוש בפורמט JSON, של KEY : VALUE.

6.2 קובץ המפתח - FIREBASE_SDK

להגשה מצורף גם הקובץ - "FIREBASE_SDK.JSON", זהו הקובץ המאפשר להתחבר דרך הקוד שלנו למערכת של בסיס הנתונים. למעשה כאן נמצאות הסיסמאות להתחברות למסד.

```
{
  "type": "service_account",
  "project_id": "cn-finalproject",
  "private_key_id": "b80e456cf412bd08dc4d91daa6f31630d7d71ee",
  "private_key": "-----BEGIN PRIVATE KEY-----\nMIIIEVAIBADANBgkqhkiG9w0BAQEFAASCBKYYggSiAgEAAoIBAQCbPYKuyXA4eZwr\neYI+wonPQ6nsYZPaXrJm8pI00H5qok705KpuIVJ07ku+KvoS2d38miHbCHXc3",
  "client_email": "firebase-adminsdk-7146z@cn-finalproject.iam.gserviceaccount.com",
  "client_id": "109906168026165669640",
  "auth_uri": "https://accounts.google.com/o/oauth2/auth",
  "token_uri": "https://oauth2.googleapis.com/token",
  "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
  "client_x509_cert_url": "https://www.googleapis.com/robot/v1/metadata/x509/firebase-adminsdk-7146z@cn-finalproject.iam.gserviceaccount.com"
}
```

6.3 מחלקת השאילות

לצורך תקשורת עם מסד הנתונים, בנינו מחלקה ארש תטפל לנו בשאילות מול השרתים והמסד. כלומר, כאשר הלקוחות מבצעים פניה לשרת בבקשה למידע מבסיס הנתונים, השרת פונה למחלקה זו והיא מבצעת את הפניה לבסיס הנתונים ומחזירה את המידע לשרתים שהם שולחים את המידע ללקוחות. נעבור על המחלקה: כאן זוהי השאילתה להוספת סטודנט חדש.

```
def add_new_student(self, student_data):
    """
    this function add new student to the database
    :param: student_data: a list that represent a student:
            [department, id, first name, last name, email, phone number, degree, track, avg, condition]
    """
    dep = db.reference(student_data[0])
    year = dep.child('year1')
    year.update({
        student_data[1]: {
            'info': {
                'id': student_data[1],
                'firstName': student_data[2],
                'lastName': student_data[3],
                'email': student_data[4],
                'phoneNumber': student_data[5]
            },
            'academic': {
                'degree': student_data[6],
                'track': student_data[7],
                'avg': int(student_data[8]),
                'condition': student_data[9]
            }
        }
    })
```

כאן זוהי השאילתה למחיקת סטודנט קיים.

```
def update_existing_student(self, student_data):
    """
    this function update student from the database
    :param: student_data: a list that represent a student: [department , year , id , info/academic , update sub. , update]
    """
    dep = db.reference(student_data[0])
    year = dep.child(student_data[1])
    _id = year.child(student_data[2])
    if _id.get() is None:
        return 1
    else:
        choose = _id.child(student_data[3])
        if student_data[4] == "avg":
            choose.update((student_data[4]:int(student_data[5])))
        else:
            choose.update((student_data[4]:student_data[5]))
    return 0
```

כאן זוהי השאילתה לעדכון סטודנט.

```
def delete_existing_student(self, student_data):
    """
    this function delete student from the database
    :param: student_data: a list that represent a student: [department, year, id]
    """
    dep = db.reference(student_data[0])
    year = dep.child(student_data[1])
    student = year.child(student_data[2])
    if student.get() is None:
        return 1
    else:
        year.child(student_data[2]).delete()
    return 0
```

כאן זוהי השאילתה להדפסת כל הסטודנטים.

```
def print_all_students(self):
    """
    :return:
    """
    all_stud = db.reference().get()
    return all_stud # change
```

כאן זוהי השאילתה להדפסת סטודנט ספציפי.

```
def print_single_student(self, student_data):
    """
    this function print all the students from the database
    """
    dep = db.reference(student_data[0])
    year = dep.child(student_data[1])
    student = year.child(student_data[2])
    if student.get() is None:
        return -1
    else:
        return student.get()
```

כאן זוהי השאילתה להדפסת סטודנטים בעלי ממוצע מקסימלי או מינימלי (אותה שאילתה עם פרמטר שונה).

```
def add_new_student(self, student_data):
    """
    this function add new student to the database
    :param: student_data: a list that represent a student:
            [department, id, first name, last name, email, phone number, degree, track, avg, condition]
    """
    dep = db.reference(student_data[0])
    year = dep.child('year1')
    year.update({
        student_data[1]: {
            'info': {
                'id': student_data[1],
                'firstName': student_data[2],
                'lastName': student_data[3],
                'email': student_data[4],
                'phoneNumber': student_data[5]
            },
            'academic': {
                'degree': student_data[6],
                'track': student_data[7],
                'avg': int(student_data[8]),
                'condition': student_data[9]
            }
        }
    })
    })
```

כאן זוהי השאילתה למתן פקטור לכל הסטודנטים.

```
def factor_students_avg(self, x):
    """
    this function add all the students a factor to avg
    :param x: the factor to add
    """
    data = db.reference().get()
    parsed_data = json.loads(json.dumps(data))
    if data is not None:
        add_to_avgs(parsed_data, x)
        db.reference().set(parsed_data)
        return 0
    else:
        return -1
```

כאן זוהי השאילתה להדפסת הסטודנטים שנמצאים על תנאי.

```
def print_conditon_students(self):
    """
    this function print all the student that have a false condition
    :return:
    """
    data = db.reference().get()
    if data is not None:
        parsed_data = json.loads(json.dumps(data))
        return get_students_by_ids(parsed_data, get_false_condition_students_ids(parsed_data))
    else:
        return -1
```

כאן זוהי השאילתה להעלאה בשנה של כלל הסטודנטים.

```
def next_year(self):
    """
    this function change every student to his next year
    if he is on third year - he finishes the degree and delete from the database
    :return:
    """
    data = db.reference().get()
    json_obj = json.loads(json.dumps(data))
    if data is not None:
        for department, years in json_obj.items():
            for year in sorted(years.keys()):
                if year == 'year3': # delete year3
                    del json_obj[department][year]
                else: # move students to the next year
                    next_year = 'year' + str(int(year[4:]) + 1)
                    json_obj[department][next_year] = json_obj[department].pop(year)
        db.reference().set(json_obj)
        return 0
    else:
        return -1
```

בסך הכל עד כה ראינו 01 שאילתות כנדרש. כאשר הלקוח מבקש לבצע שאילתה מול המסד, השרת פונה למחלקה ומבצע כאן את השאילתות ולאחר מכן שולח את התגובה ללקוח בפרוטוקול הנבחר בכל פעם.

ישנן מספר פעולות עזר נוספות במחלקה, מעבר לממש ביצוע השאילתות מול המסד, נעבור גם עליהן:
כאן זוהי פונקציה אשר מחזירה רשימה של כל תעודות הזהות בעלות הציון הרצוי.

```
def get_ids_by_avg(database, avg):
    ids = []
    for dept, years in database.items():
        for year, students in years.items():
            for student_id, info in students.items():
                if info['academic']['avg'] == avg:
                    ids.append(info['info']['id'])
    return ids
```

כאן זוהי פונקציה אשר מקבלת פורמט JSON שמתאר את המסד ומחפשת את כל הסטודנטים עם הממוצע.

```
def find_grades(obj):
    """
    this function save all the avg of the students in the database
    :param obj: json that represent the database
    :return: return dict that contains all the avg
    """
    if isinstance(obj, dict):
        for k, v in obj.items():
            if k == 'avg':
                yield v
            else:
                yield from find_grades(v)
    elif isinstance(obj, list):
        for i in obj:
            yield from find_grades(i)
```

כאן זוהי פונקציה אשר מוסיפה את הפקטור לציון של כל הסטודנטים. אם הציון אחרי הפקטור גדול ממאה, הציון הסופי יהיה 100.

```
def add_to_avgs(obj, x):
    """
    this function add factor to all avg of the students in the database
    :param obj: json that represent the database
    :param x: the factor to add
    :return: a json that represent the database after the add
    """
    if isinstance(obj, dict):
        for k, v in obj.items():
            if k == 'avg':
                obj[k] = int(min(int(obj[k]) + int(x), 100)) # limit to max 100
            else:
                add_to_avgs(v, x)
    elif isinstance(obj, list):
        for i in obj:
            add_to_avgs(i, x)
```

כאן זוהי פונקציה אשר מקבלת JSON שמתאר את המסד ומחזירה רשימה של תעודות זהות של סטודנטים על תנאי.

```
def get_false_condition_students_ids(obj):
    """
    This function returns a list of ids of all students that have a false condition in the database
    :param obj: json that represent the database
    :return: list of student ids
    """
    student_ids = []
    if isinstance(obj, dict):
        if 'condition' in obj.get('academic', {}):
            if obj['academic']['condition'] == 'True':
                student_ids.append(obj['info']['id'])
        for k, v in obj.items():
            student_ids += get_false_condition_students_ids(v)
    elif isinstance(obj, list):
        for i in obj:
            student_ids += get_false_condition_students_ids(i)
    return student_ids
```

כאן זוהי פונקציה אשר מקבלת רשימה של תעודות זהות ומחזירה רשימה של אובייקטי הסטודנטים בפורמט JSON.

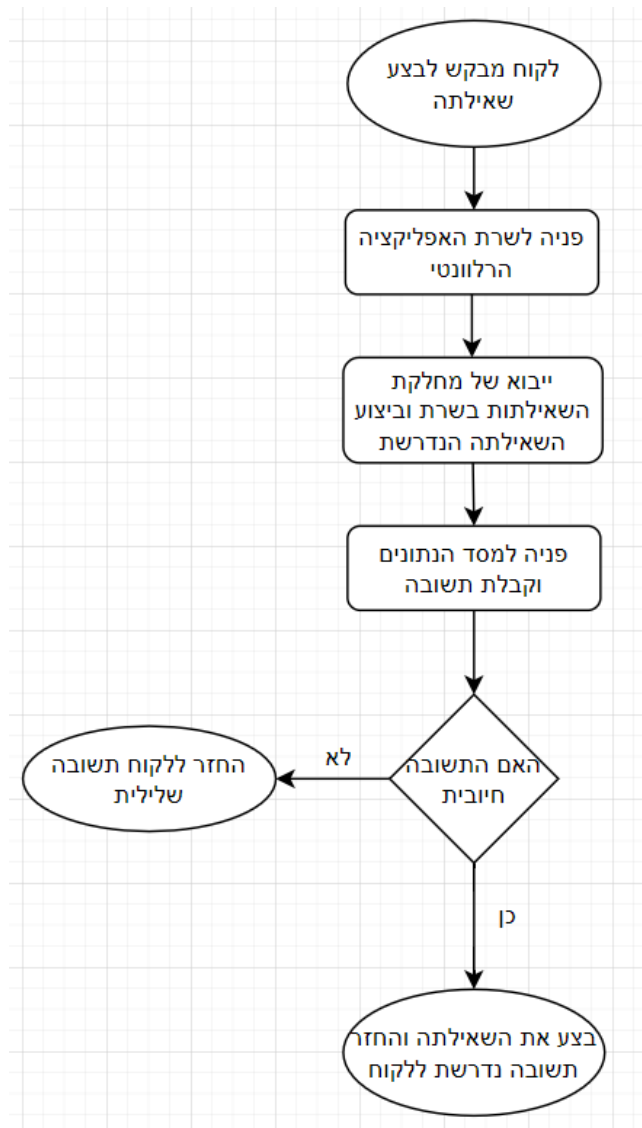
```
def get_students_by_ids(json_obj, id_list):
    """
    This function returns the JSON representation of all the students with the ids in the given list
    :param json_obj: JSON object representing the database
    :param id_list: list of student ids to search for
    :return: list of JSON objects representing the matching students
    """
    matching_students = []
    for id in id_list:
        student = get_student_by_id(json_obj, id)
        if student:
            matching_students.append(student)
    return matching_students
```

כאן זוהי פונקציה שמקבלת מתאר JSON של המסד ותעודת זהות ומחזירה JSON של סטודנט בעלת תעודת הזהות הזו.

```
def get_student_by_id(json_obj, id):
    """
    This function returns the JSON representation of the student with the given id, if it exists in the given JSON object
    :param json_obj: JSON object representing the database
    :param id: student id to search for
    :return: JSON object representing the matching student, or None if no match is found
    """
    if isinstance(json_obj, dict):
        if 'id' in json_obj.get('info', {}):
            if json_obj['info']['id'] == id:
                return json_obj
        for k, v in json_obj.items():
            student = get_student_by_id(v, id)
            if student:
                return student
    elif isinstance(json_obj, list):
        for i in json_obj:
            student = get_student_by_id(i, id)
            if student:
                return student
    return None
```

6.4 תרשים זרימה

כאן נראה את תרשים הזרימה המתאר את פעילות מחלקת השאלות ומסד הנתונים. ראשית, הלקוח מבקש לעשות שאילה - הדבר בא לידי ביטוי בכך שהוא לוחץ על אחד הכפתורים באפליקציה, האפליקציה פונה לשרת האפליקציה המתאים, בין אם הוא TCP או UDP, השרת משתמש במחלקת השאלות (אין בניהם תקשורת של ממש דרך שקע אבל יש שימוש כמו במודול), מחלקת השאלות מבצעת את הפניות לפועל למסד הנתונים, שם הבקשה מתבצעת. כמובן, מחזירים ללקוח תשובה בהתאם. כמובן ישנו עוד שלב ביניים של בדיקת תקינות הקלט בצד הלקוח, אך ההתייחסות פה היא לקלט תקין, ברור שאם הקלט אינו תקין התרשים אינו יתקדם החל מהשלב השני ולכן אין בכך משמעות רבה.

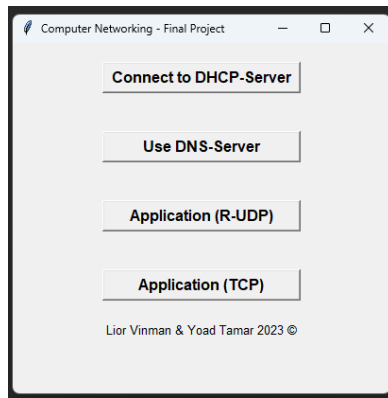


7 ממשק גרפי - APPLICATION_GUI

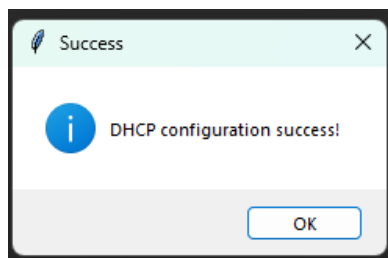
כאן נעבור על כל הקשור לממשק הגרפי של המשתמש, נראה צילומי מסך ונסביר את הבניה שלו. בשביל לבנות את הממשק השתמשנו בספריית TKINTER.

7.1 צילומי הממשק

כך נראה החלון הראשי שדרכו ניתן לבחור את סוג השירות הרצוי.



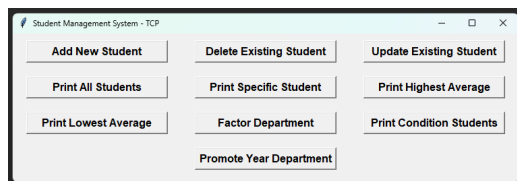
כעת נבחר ב־DHCP, זוהי ההודעה שמוחזרת ללקוח לאחר הלחיצה.



כך נראה החלון של שרת ה־DNS, כאן ניתן להקליד כתובת ולקבל IP.



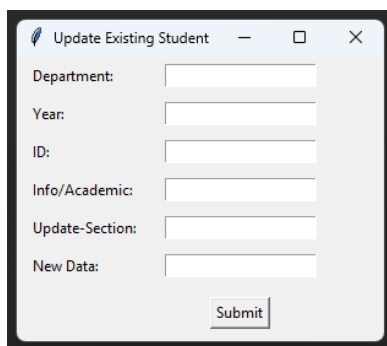
כך נראה החלון הראשי של האפליקציה (זהה עבור TCP ו-RUDP), פה נמצאות כל השאילות.



כאן זהו החלון של שאילתת ההוספה, נדרש למלא את הפרטים לכל סטודנט.

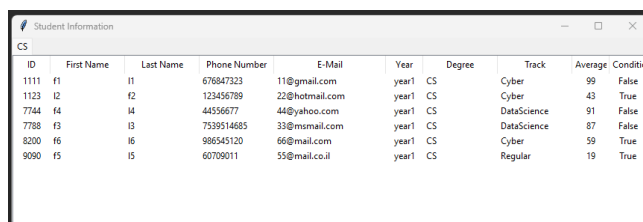
כאן זהו החלון של שאילתת המחיקה.

כאן זהו החלון של שאילתת העדכון.



A screenshot of a software window titled "Update Existing Student". It contains six text input fields labeled "Department:", "Year:", "ID:", "Info/Academic:", "Update-Section:", and "New Data:". A "Submit" button is located at the bottom right of the form.

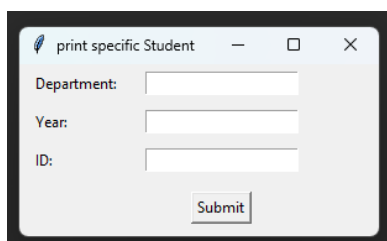
כאן זוהי טבלה לדוגמה בשאילתת הדפסת כל המאגר.



A screenshot of a software window titled "Student Information" displaying a table of student data. The table has 10 columns: ID, First Name, Last Name, Phone Number, E-Mail, Year, Degree, Track, Average, and Condition. The data is as follows:

ID	First Name	Last Name	Phone Number	E-Mail	Year	Degree	Track	Average	Condition
1111	f1	11	678947323	11@gmail.com	year1	CS	Cyber	99	False
1123	12	12	123456789	22@hotmail.com	year1	CS	Cyber	43	True
7744	14	14	445566777	44@yahoo.com	year1	CS	DataScience	91	False
7788	13	13	7539514685	33@gmail.com	year1	CS	DataScience	87	False
8200	16	16	986545120	66@mail.com	year1	CS	Cyber	59	True
9090	15	15	60709011	55@mail.co.il	year1	CS	Regular	19	True

כאן זוהי השאילתה להדפסת סטודנט ספציפי, לאחר מכן תודפס טבלה בעלת שורה אחת עם הסטודנט המתאים.



A screenshot of a software window titled "print specific Student". It contains three text input fields labeled "Department:", "Year:", and "ID:". A "Submit" button is located at the bottom right of the form.

כאן זוהי טבלה לדוגמה להדפסת סטודנטים עם ממוצע הכי גבוה.

Student Information									
ID	First Name	Last Name	Phone Number	E-Mail	Degree	Track	Average	Condition	
1111	f1	l1	676847323	11@gmail.com	CS	Cyber	99	False	

כאן זוהי טבלה לדוגמה להדפסת סטודנטים עם ממוצע הכי נמוך.

Student Information									
ID	First Name	Last Name	Phone Number	E-Mail	Degree	Track	Average	Condition	
1111	f1	l1	676847323	11@gmail.com	CS	Cyber	99	False	

כאן זוהי השאילתה של מתן פקטור לסטודנטים.

Factor

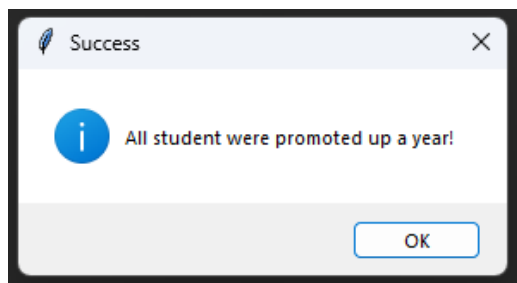
Number of points

Submit

כאן זוהי השאילתה של הדפסת סטודנטים על תנאי, זוהי טבלה לדוגמה.

Student Information									
ID	First Name	Last Name	Phone Number	E-Mail	Degree	Track	Average	Condition	
1123	I2	f2	123456789	22@hotmail.com	CS	Cyber	43	True	
8200	f6	I6	986545120	66@mail.com	CS	Cyber	59	True	
9090	f5	I5	60709011	55@mail.co.il	CS	Regular	19	True	

כאן זוהי הודעת הצלחה על קידום כל הסטודנטים בשנה.



7.2 מעבר חלקי על הקוד

כאן נציג מספר פונקציות המראות לנו כיצד נוצר הממשק (רוב אם לא כולן זהות, נסביר בקצרה את העקרונות):
כאן ניתן לראות את הבנאי של האובייקט, שמיוצר על ידי ה-GUI. ניתן לראות כאן בניה של הכפתורים ויצירת האובייקט.

```
def __init__(self, master):  
    """  
    GUI's constructor  
    :param master: the main GUI window (tk.TK())  
    """  
  
    # main window settings  
    self.master = master  
    master.geometry(f"{MAIN_WIDTH}x{MAIN_HEIGHT}")  
    master.title("Computer Networking - Final Project")  
    master.config()  
  
    # DHCP button settings  
    self.dhcp_button = tk.Button(master, text="Connect to DHCP-Server",  
                                command=self.conn_dhcp, width=20)  
    self.dhcp_button.config(font=('MS Outlook', 12, 'bold'))  
    self.dhcp_button.pack(padx=20, pady=20)  
  
    # DNS button settings  
    self.dns_button = tk.Button(master, text="Use DNS-Server",  
                                command=self.conn_dns, width=20)  
    self.dns_button.config(font=('MS Outlook', 12, 'bold'))  
    self.dns_button.pack(padx=20, pady=20)  
  
    # APP_RUDP button settings  
    self.app_rudp_button = tk.Button(master, text="Application (R-UDP)",  
                                    command=lambda: self.conn_app(RUDP), width=20)  
    self.app_rudp_button.config(font=('MS Outlook', 12, 'bold'))  
    self.app_rudp_button.pack(padx=20, pady=20)  
  
    # APP_TCP button settings  
    self.app_tcp_button = tk.Button(master, text="Application (TCP)",  
                                    command=lambda: self.conn_app(TCP), width=20)  
    self.app_tcp_button.config(font=('MS Outlook', 12, 'bold'))  
    self.app_tcp_button.pack(padx=20, pady=20)  
  
    # credits label settings  
    self.cred = tk.Label(master, text="Lior Vinman & Yoad Tamar 2023 \u00A9")  
    self.cred.config(font=("MS Outlook", 10))  
    self.cred.pack()
```

כאן ניתן לראות פונקציה, אשר מבצעת הדפסה של טבלה בהינתן JSON.

```
def display_table(self, json_data):
    """
    this function prints all the database in table
    :param json_data: json format of the database
    """
    root = tk.Toplevel(self.master)
    root.title("Student Information")
    notebook = ttk.Notebook(root)

    for degree in json_data.keys():
        frame = ttk.Frame(notebook)
        notebook.add(frame, text=degree)

        columns = ("ID", "First Name", "Last Name", "Phone Number", "E-Mail", "Year", "Degree", "Track", "Average",
                  "Condition")
        tree = ttk.Treeview(frame, columns=columns, show="headings")

        tree.column("ID", width=50, anchor="center")
        tree.column("First Name", width=100, anchor="w")
        tree.column("Last Name", width=100, anchor="w")
        tree.column("Phone Number", width=100, anchor="w")
        tree.column("E-Mail", width=150, anchor="w")
        tree.column("Year", width=50, anchor="center") # New column
        tree.column("Degree", width=100, anchor="w")
        tree.column("Track", width=100, anchor="w")
        tree.column("Average", width=50, anchor="center")
        tree.column("Condition", width=50, anchor="center")

        for col in columns:
            tree.heading(col, text=col)

        for year in json_data[degree].keys():
            for id_num in json_data[degree][year].keys():
                student = json_data[degree][year][id_num]
                tree.insert("", "end", values=(student["info"]["id"],
                                              student["info"]["firstName"],
                                              student["info"]["lastName"],
                                              student["info"]["phoneNumber"],
                                              student["info"]["email"],
                                              year, # New value
                                              student["academic"]["degree"],
                                              student["academic"]["track"],
                                              student["academic"]["avg"],
                                              student["academic"]["condition"])))

        tree.pack(fill="both", expand=True)

    notebook.pack(fill="both", expand=True)
```

8 שאלות תיאורטיות

בפרק זה נענה על השאלות הנוספות למטלה.

8.1 שאלה 1

ארבעה הבדלים עקריים בין פרוטוקול TCP לבין פרוטוקול QUIC:

1. התחברות - בפרוטוקול QUIC ההתחברות מתבצעת בשיטה הנקראת 0-RTT, מה שמאפשר ללקוח, ישר להתחיל לשלוח מידע ונתונים ללא שום התחברות ממשית (כלומר, ללא כל לחיצת יד ראשונית), לעומת זאת, בפרוטוקול TCP, כדי להתחבר צריכה להיות לחיצת יד משולשת בין שתי התחנות (הלכוח והשרת) כאשר בלחיצה הזאת נשלחות 3 חבילות (SYN, ACK, SYN), דבר הגורם לTCP להיות איטי יותר מאשר QUIC.
2. אובדן חבילות - TCP משתמש בשיטה הנקראת RETRANSMISSION והיא כאשר יש אובדן של חבילות, הפרוטוקול חוזר אחורה למקום שבו חסרות חבילות ו-"מתעקש" לקבל אותן עד שיגיעו כדי שכל המידע יעבור ובתהליך הזה נשלחות הרבה חבילות ACK, לעומת זאת, QUIC, משתמש בשיטה הרבה יותר יעילה הנקראת FORWARD ERROR CORRECTION אשר מאפשרת להשיג חבילות שנאבדו בזמן ריצה, ללא צורך לחזור אחורה למקום בו התחיל האובדן.
3. ריבוי התחברויות - פרוטוקול TCP, תומך במספר חיבורים על שקע (socket), אבל כל חיבור כזה צריך לעבור תהליך של לחיצת ידיים משולשת ולאחר מכן בקרת עומס (CONGESTION CONTROL), לעומת זאת, פרוטוקול QUIC, עובד על ריבוי זרמים על פני חיבור של שקע יחיד, מה שמאפשר לנצל את משאבי הרשת יותר ביעילות.
4. אבטחת מידע - פרוטוקול QUIC, משלב תכונות של אבטחה, לדוגמה הצפנה ואימות ישיר בתוך הפרוטוקול מה שמבטיח רמות הבטחה גבוהות מאוד בברירת מחדל. לעומת זאת, פרוטוקול TCP אינו מספק אבטחה שמובנת בתוך הפרוטוקול עצמו והוא מסתמך על אבטחה של פרוטוקול אחרים הפועלים בשכבת התעבורה כדי לספק את האבטחה הדרושה (לדוגמה, TLS - TRANSPORT LAYER SECURITY).

8.2 שאלה 2

שני הבדלים עקריים בין CUBIC לבין VEGAS:

1. בקרת העומס - אלגוריתם CUBIC משתמש בפונקציה ממעלה שלישית ומעלה על מנת לשמור על בקרת העומס, ולהתאים את גודל החלון בהתבסס על רמות העומס ברשת. גודל החלון קטן לפי הפונקציה שהאלגוריתם משתמש בה ולאחר שהוא הצטמצם הוא גדל באיטיות. לעומת זאת, VEGAS משתמש בשיטה בשם "TCP VEGAS", כדי לזהות תעבורה ועומס. השיטה פועלת על ידי מדידה של הזמן הממוצע של הגעת המנות ומשתמש בנתון זה כדי להתאים את גודל השליחה והקבלה. כאשר מתגלה עומס, האלגוריתם מפחית את קצב השליחה בכמות כלשהי, אשר תפחית את העומס.
2. תגובות לעומס - אלגוריתם CUBIC מגיב בצורה איטית יותר עבור העומס אבל כאשר מגיב, התגובה יותר אגריסיבית (הכוונה היא שהוא מקטין את החלון בצורה קיצונית יותר ולא מתונה, הקטנת החלון תלויה בגודל הפונקציה שהיא ממעלה שלישית לפחות שהוא משתמש בזה) והדבר מפחית את קצב השליחה ברשת. לעומת זאת, אלגוריתם VEGAS מגיב בצורה מהירה יותר ומידית יותר לעומס והוא מפחית את קצבי השליחה והקבלה בהדגה, ההבדל כמובן הוא בכך שהאלגוריתם הראשון יותר אגריסיבי ויותר איטי מהשני. הדבר משפיע על ביצועי פרוטוקול TCP בתנאי הרשת הנוכחיים.

8.3 שאלה 3

הסבר על פרוטוקול BGP:

הפרוטוקול הוא פרוטוקול ניתוב מידע סטנדרטי המשמש להחלפת מידע וניתוב בין רשתות ותתי רשתות באינטרנט. הפרוטוקול משתמש בנתיבים וקטוריים ומשתמש בעקרון של מערכות אוטונומיות (ASes) כדי לתת ייצוג לכל רשת בודדת, לכל מערכת כזו ניתן מספר יחודי המשמש לניהול מדיניות הניתוב שלו. השימוש בפרוטוקול הוא לצורך ניתוב מידע בין מערכות אוטונומיות כאלה ולצורך קביעת הנתיב הטוב ביותר לתנועה להגיע ליעד. הפרוטוקול הוא מורכב ובעל יכולות רבות לדוגמה - תמיכה במספר מסלולים, סינון מסלולים, בחירת מסלולים על סמך קריטריונים כמו אורך המסלול, וכמובן תמיכה גם ב-IPv4 וגם ב-IPv6.

הבדלים בין פרוטוקול BGP לבין פרוטוקול OSPF:

1. אופן השימוש - פרוטוקול OSPF הוא פרוטוקול המשמש בתוך מערכת אוטונומית אחת ואילו פרוטוקול BGP הוא משמש להעברת מידע בין מערכות אוטונומיות.
 2. סוג הניתוב - BGP בוחר את המסלול בהתחשב במספר תכונות כדי לקבוע את המסלול הטוב ביותר (לא בהכרח הקצר ביותר), פרוטוקול OSPF בוחר תמיד את המסלול הקצר ביותר. בנוסף, OSPF בוחר את המסלול בהתבסס על עלות הקישור ברשת וגם לפי האורך, לעומת זאת BGP בוחר את המסלול לפי העדפה מקומית, אורך המסלול במערכת האוטונומית, קפיצות ועוד.
 3. גודל השימוש - פרוטוקול OSPF משמש לרוב ברשתות קטנות יחסית, לעומת זאת BGP משמש ברשתות גדולות (כמו לדוגמה רשת האינטרנט), מכיוון שהוא יכול להתמודד עם מספר רב של מסלולים ומדיניות מורכבות בו זמנית.
- כפי שכבר אמרנו, פרוטוקול BGP אינו עובד לפי מסלולים קצרים ביותר. הוא תמיד בוחר את המסלול הטוב ביותר - אבל ההגדרה של מסלול טוב ביותר לא בהכרח אומרת שהוא חייב להיות קצר ביותר (אך לפעמים זהו כן המסלול הטוב ביותר), הפרוטוקול גם מתחשב בשיקולים אחרים לבחירת המסלול כגון העדפה מקומית, תלות בלקוח ובשרת ועלות המעבר ברשת.

8.4 שאלה 4

APPLICATION	PORT SRC	PORT DES	IP SRC	IP DES	MAC SRC	MAC DES
DNS	53	53	127.0.0.1	127.0.0.1	LOCAL	LOCAL
DHCP	67	9989	127.0.0.1	BROADCAST	LOCAL	LOCAL
APPLICATION TCP	9090	9090	127.0.0.1	127.0.0.1	LOCAL	LOCAL
APPLICATION RUDP	1235	1235	127.0.0.1	127.0.0.1	LOCAL	LOCAL

בשימוש בפרוטוקול QUIC אין שום שינוי, בשימוש עם NAT יש שינוי והיו יותר תבילות.

8.5 שאלה 5

הבדלים בין ARP לבין DNS:

1. שימוש - הפרוטוקול ARP משמש למיפוי של כתובות פיזיות (MAC), לעומת זאת הפרוטוקול DNS משמש להמרה של כתובות אינטרנט לכתובות IP.
2. שכבות - ARP פועל בשכבת הקישוריות (LINK), לעומת זאת DNS פועל בשכבת האפליקציה (APPLICATION) - הכוונה לשכבות במודל ה-OSI.
3. זכרון - פרוטוקול ARP שומר על מטמון של כתובות פיזיות (MAC) אשר הופיעו לאחרונה ברשת המקומית, לעומת זאת, DNS, שומר על מטמון של דומיינים אשר כבר ביקשו להחליף אותם בכתובת IP.
4. סוגי התקפות - הפרוטוקולים הנל שונים גם בהתקפות עליהם - לדוגמה, פרוטוקול ARP פגיע להתקפות של זיוף ARP (ARP SPOOFING) ולעומת זאת, DNS, לדוגמה פגיע להתקפות של הרעלת המטמון שלו (DNS CACHE POISONING).
5. מהירות - פרוטוקול ARP מהיר יותר מפרוטוקול DNS, מכיוון שהוא מבצע מיפוי ישיר של כתובות אינטרנט עם כתובות פיזיות לעומת DNS אשר צריך באופן רקורסיבי לחשב כתובת של קישור.

9 ביבליוגרפיה

https://he.wikipedia.org/wiki/Dynamic_Host_Configuration_Protocol .1

https://he.wikipedia.org/wiki/Domain_Name_System .2

https://learn.microsoft.com/en-us/sql/relational-databases/tables/tables?view=sql-server-ver16 .3

https://www.freecodecamp.org/news/how-to-get-started-with-firebase-using-python/ .4

https://www.geeksforgeeks.org/flow-control-in-data-link-layer/ .5

https://www.geeksforgeeks.org/difference-between-flow-control-and-congestion-control/?ref=rp .6

https://www.geeksforgeeks.org/slow-start-restart-algorithm-for-congestion-control/?ref=rp .7

https://www.geeksforgeeks.org/reliable-user-datagram-protocol-rudp/ .8