פרויקט גמר רשתות תקשורת

אפליקציית HTTP המבצעת redirect לשרת אחר שמשם יורד הקובץ

מגישים: שלומי זכריה- 315141242
יסמין כהן- 212733836

תשובות לשאלות מה-pdf

1. **<u>מנה לפחות ארבע הבדלים עיקריים בין פרוטוקול TCP ל-QUIC</u>**

TCP ו-QUIC הם שניהם פרוטוקולים שהתפקיד שלהם הוא להעביר נתונים באינטרנט, נמנה 4 הבדלים עיקריים ביניהם .

1. שחזור נתונים שאבדו במהלך העברת נתונים:
   TCP משתמש במנגנון שנקרא אישור סלקטיבי (SACK) כדי להתגבר על אובדן נתונים.
   QUIC משתמש באלגוריתם אחר , שהוא אלגוריתם שנוצר דווקא עבור הפרוטוקול הזה לבקרת עומס, ותפקיד האלגוריתם הזה הוא להתגבר על אובדן מנות ולשחזר אותם מחדש , האלגוריתם עובד יותר מהיר ויעיל מהמנגנון של.TCP
   נשים לב שההבדל פה הוא בשימוש של כל פרוטוקול באלגוריתמים שונים לשחזר נתונים שאבדו בדרך.

2. ביצירת חיבור: כאשר רוצים להתחבר לשרת בחיבור TCP צריך ליצור לחיצת יד 3 פעמים, 1 לכל כיוון.
   QUIC יוצרת חיבור בשיטה מהירה ומאובטחת יותר שנקראת
   0 (Zero Round Trip Time) RTT-שזו שיטה שמאפשרת לclient לשלוח נתונים מוצפנים לשרת כבר בחבילה הראשונה שהוא שולח.

3. העברת חיבורים: ב QUIC אפשר לשלוח הרבה זרמים של נתונים בחיבור יחיד, **אבל** ב TCP-ישנו חיבור נפרד לכל זרם.

4. הצפנת נתונים: הצפנת הנתונים ב TCPוב QUICנעשית בדרכים שונות, מכוון שהם משתמשים בפרוטוקולים שונים,
   TCP משתמש בפרוטוקול שנקרא Transport Layer Security (TLS) בשביל להצפין את הנתונים שהועברו אליו.
   QUIC מצפין נתונים באמצעות היישום של גוגל של פרוטוקול TLS הנקרא."QUIC Crypto"

2. **מנה לפחות שני הבדלים עיקריים בין Cubic לVegas-**

Cubic ו Vegas-הן שתי דרכים להתמודד עם בכמות הנתונים שנשלחת ברשת. כל אחד עושה את זה בדרך אחרת.

הבדל ראשון-

cubic מתמודד עם זרם הנתונים על ידי כך שהוא מגדיל באיטיות את כמות הנתונים הנשלחת עד שהיא מגיעה לרמה המקסימלית אבל אם הוא מזהה עומס הוא מקטין במהירות את הכמות נתונים שיכולה להישלח.
Vegas לעומת זאת, שולטת בקצב הנתונים על ידי מעקב של עיכובים , בודק כמה זמן לוקח לנתונים לעבור דרך הרשת. אם זמן ההמתנה גבוהה מדי, אז בשיטת vegas פשוט מאטים את קצב זרימת הנתונים.

ההבדל השני-

cubic מאפשרת גדילה איטית אך יציבה של זרימת הנתונים. vegas, לעומת זאת, מגדילה אחרת את הזרם של הנתונים העוברים ברשת, בצורה שעלולה להוביל לשינויים אגרסיביים יותר בזרם.

3. **הסבר מהו פרוטוקול, BGP במה הוא שונה מ OSPF-והאם הוא עובד על פי מסלולים קצרים-**
   BGP (Border Gateway Protocol) זו מערכת שעוזרת לנתבים לנווט באינטרנט.
   זוהי דרך לתקשר בין רשתות שונות באינטרנט וזוהי גם דרך להחליט איזה דרך הכי טובה לשלוח בה נתונים, בין הרשתות האלו.

OSPF (Open Shortest Path First) היא דרך לתקשור בין נתבים שנמצאים באותה רשת, היא פשוט מוצאת את את הנתיב הקצר ביותר לנתונים לעבור בין הנתבים האלו.

וזה בעצם השוני העיקרי בין BGP ל , OSPF-ש BGP-משמש לתקשורת שמתרחשת בין רשתות שונות, בעוד ש OSPF-משמש לתקשורת של נתבים שנמצאים בתוך אותה רשת.

BGP לא, הוא לא תמיד עובד על פי מסלולים קצרים, מכיוון שהוא מתמקד יותר במציאת המסלול הטוב ביותר בין רשתות שונות, טוב זה אומר , בדגש על מסלולים שהחיבורים שלהם מהירים ואמינים יותר, ולא דווקא על המסלול הקצר ביותר.

4) בהינתן הקוד שפיתחתם בפרויקט זה, אנא הוסיפו את הנתונים לטבלה הזו על בסיס תהליך ההודעות של הפרויקט שלכם. הסבירו איך ההודעות ישתנו אם יהיה **NAT** בין המשתמש לשרתים והאם תשתמשו בפרוטוקול **QUIC**

Application	Port Src	Port Des	IP Src	IP Des	Mac Src	Mac Des
num : 3	20836	30242	127.0.0.1	127.0.0.1	comp_mac	comp_mac

5. **הסבירו את ההבדלים בין פרוטוקול ARP לDNS-**

פרוטוקולי ARP ו DNS-פועלים שניהם ברשתות של מחשבים, אך פועלים למטרות שונות וגם פועלים בשכבות שונות של חבילת הרשת כך:

ARP משמש לגילוי כתובת MAC של מכשיר על פי כתובת ה IP-שלו. בנוסף ARP פועל בשכבת קישור הנתונים.

מצד שני DNS, משמש כדי לחפש את כתובת ה IP-השייכת ל "הדומיין" , כך שהמכשיר יוכל ליצור חיבור למרות שיש לו רק דומיין של אתר מסוים. ונשים לב ש DNS פועל בשכבת האפליקציה.

הסבר על שרת ולקוח באפליקציה שבנינו-

נסביר לפני הכל על הרעיון הכללי, ישנם AppServer ,Client, HttpServer .
יצירת אפליקציה HTTP redirect המאפשרת הורדת קבצים תחת פרוטוקול RUDP.

מה שעשינו זה כך, כדי לאבטח את הורדת הקבצים, יצרנו חיבור UDP בין שרת ללקוח ועטפנו אותו בהגנות כך שיגיע לרמה גבוהה של אמינות, בקרת גודש ובקרת זרימה.
מכוון שעשינו אפליקציה http אז התהליך עובד כך:
יצרנו לקוח שפונה עם בקשה להורדת קובץ מסוים לשרת , http שהשרת http הזה עושה redirect לשרת ,AppServer שמהשרת AppServer יורד הקובץ, הקובץ אצלנו יכול להיות איזה סוג של קובץ שרוצים, ללא הגבלה.
כל התהליך של הורדת הקובץ מהבקשה של הclient עד היישום זה בעצם קורה בקריאה רקורסיבית של http לשרת אחר שהשרת האחר הוא המבצע.

החיבור rudp שבנינו מיישם:

אמינות: מבטיח שהנתונים הנשלחים דרך החיבור יגיעו ליעד ללא שגיאות או אובדן, בדומה ל.TCP-

בקרת גודש: מיישם בקרת גודש כדי למנוע עומס ברשת, בדומה ל.TCP-

בקרת זרימה: מיישם בקרת זרימה כדי להבטיח שהקצה המקבל של החיבור יוכל להתמודד עם כמות הנתונים הנשלחים, בדומה לTCP-

נראה בהמשך איפה רואים את המימוש שיוצר כל אחד מ3 התכונות שהזכרנו לעיל.

AppServer

נתחיל עם הסבר על שרת האפלקציה אשר אחראי על הורדת הקבצים, נדגיש שבהסבר הזה כאשר נתייחס לuser אנחנו מתכוונים לשרתי http שפה הם הלקוחות של ה.AppServer.

```python
from socket import AF_INET, socket, SOCK_STREAM, SOCK_DGRAM
from threading import Thread
import time
import os
import pickle


# user class
class User:

# Online_user, holds name, socket client and IP address


    def __init__(self, addr, client):
        self.addr = addr
        self.client = client

    def __repr__(self):
        return f"User({self.addr})"


# GLOBAL CONSTANTS
HOST = ''
PORT = 30242
MAX_CONNETIONS = 10
ADDR = (HOST, PORT)
BUFSIZ = 1024
FILES_FOLDER = 'Files'

# GLOBAL VARIABLES
users = []
SERVER = socket(AF_INET, SOCK_STREAM)
SERVER.bind(ADDR)  # set up server
# dict of all ports 55000-55015
# (if port is caught his value = False otherwise True)
ports = {}
for i in range(16):
    ports[30242 + i] = True
```

הסבר על החלק קוד הזה:

דבר ראשון פתחנו class של user שייצג לנו משתמשים באפלקציה שהם בעצם שרתי.http
השרת שלנו יכול לענות לבקשות בו זמנית עד 10 משתמשים.
נסביר קצת על המשתנים שהגדרנו:
הגדרנו שה-host יכול לקבל אליו **כל** שרת ,http הPort הינו 30242 או במילים אחרות של 30 YYY כמו שהתבקש כאשר YYY הוא ת"ז של שלומי.
יש לנו את ה MAX_CONNECTIONS שזה המקסימום של ה usersשיכולים לגשת לשרת הזה במקביל.
BUFSIZ מגדיר כמה בתים במקסימום יכולים להגיע בפעם אחת.

FILES_FOLDER זה מגדיר את השם לתקיה שיצרנו שהיא מכילה בתוכה את כל הקבצים שאפשרי לשלוח לclient.

בנינו מערך של users נגדיר את ה serverשלנו להיות מסוג חיבור tcp ובנוסף פתחנו האזנה המוגדרת עם ה- ip ה- port המתאימים.
לאחר מכן הגדרנו מילון של ports מהמספר 304242 עד 30257 - 16 פורטים , אלו פורטים שנועדו כל פעם לייצג את השרת עם לקוחות שונים שרוצים להתחבר לשרת הזה להוריד קבצים ממנו.
לאחר מכן עשינו לולאה של פור שמגדירה את כל מספרי הפורטים בתוך מערך שבעתיד ייצין לנו איזה פורט תפוס ואיזה לא ואפשר להשתמש בו.

```
def download(client, filename):
    SIZE = 1000
    LOST =100
    LATENCY= 0.1
    filepath = f"{os.getcwd()}/Files/{filename}"
    filesize = os.path.getsize(filepath)
    prt = 0
    for port, flag in ports.items():
        if flag:
            prt = port
            ports[port] = False
            break
    if prt == 0:
        print("No port find")
        return

    SERVERUDP = socket(AF_INET, SOCK_DGRAM)
    # Bind to address and ip

    SERVERUDP.bind(("", prt))
    print("[UDP STARTED] Waiting for connections...")

    msg = bytes(f"UDP {prt} {filesize} {filename}", "utf8")
    client.send(msg)

    with open(filepath, "rb") as file:
        msg, addr = SERVERUDP.recvfrom(BUFSIZ)
        msg = msg.decode('utf-8')
        print(f'[SEND] sending {filename} to the Client')
        print('started', end='')
        c = 7
        while (msg != "finished"):
            msg = msg.split()
            if msg[0] == "part":
                numpart = int(msg[1])
                file.seek(numpart * SIZE)
                data = file.read(SIZE)
                data_size = len(data)
                c += 1
                if c % LOST == 0:
                    data = file.read(SIZE-10)
                    print("lost packet sent")
                    time.sleep(LATENCY)

                data = (data, data_size)
                data = pickle.dumps(data)
                SERVERUDP.sendto(data, addr)
                print('.', end='')

            msg, addr = SERVERUDP.recvfrom(BUFSIZ)
            msg = msg.decode('utf-8')
        print('finished')
        SERVERUDP.close()
        ports[addr[1]] = True
```

נסביר מה קורה בפונקציה שפה מעלינו:

יצרנו פונקציה download שמקבלת את ה client ומקבלת , filename .
filename זה שם הקובץ שה client רצה להוריד , ה client יהיה מיוצג על ידי אוביקט user מהרשימה של users שיצרנו (שהם בעצם הלקוחות של ה- HTTP Server)
נגדיר את size להיות גודל הבתים המקסימלית עבור כל שליחת פאקטה (כמה בתים מהקובץ ישלחו במקסימום כל פעם), במקרה שלנו זה 1000 בתים.
filepath מגדירה לנו string שה string הזה מייצג לנו את ה-path לתוך תקיית Files שמכילה לנו את הקבצים שיש בתוך האפלקציה שלנו אשר נתנים לשליחה.

Filesize יחזיק לנו את גודל ה- file שנבחר.
עכשיו נגדיר את ה-port להיות 0 (prt), ורצים בתוך לולאה שרצה על כל הפורטים שנמצאים בתוך ה-dictionary, נגדיר את flag להיות true אם באמת יהיה פורט פנוי שאפשר להשתמש בו, ואם כן ישר נגדיר את הפורט הפנוי להיות הפורט של השרת, שמייצג את השרת העכשווי מול הclient.
לאחר מכן נכניס בתוך רשימת הפורטים את port הזה בתור false כדי שלא יוכלו להשתמש בו אחר כך, (כמובן לא ניתן להעביר שני יוזרים על אותו פורט, כי אם נניח שני לקוחות היו על אותו פורט הם יכולים לקבל הודעה אחת של השני, והורדת הקבצים ביניהם יכולה ל"התערבב").
במידה ו- prt יהיה שווה שווה ל0 בסוף הלולאה, זה אומר שלא מצאנו פורט פנוי ולכן לא נוכל להוריד את הקובץ לclient.
עכשיו נפתח סוקט, udp שהסרבר יאזין לבקשות מאותו פורט שנמצא.
לאחר מכן נגדיר את הmsg שלנו להעביר את אותו srting בבתים.
ה string מופרד ברווחים שבכל חלק אנחנו מייצגים משהו, filename, filesize, port, udp ולאחר מכן אנחנו שולחים את ה msg ל.client

נגיד זאת, וזה ידבר על כל המשך הקוד שאנחנו משתמשים בקידוד מסוג (utf-8).
מכאן נקבל מהלקוח את הבקשה להורדת הקובץ.
נבצע decoded להודעה שלו ונפענח את הstring
כל עוד לא יודפס לנו finshed הלולאה תמשיך לרוץ,ו היא תבצע את הפקודה: msg= msg.split() .

מכאן אנחנו עושים תנאי שאם msg[0] יהיה שווה ל "part", שה "part" מייצג אצלנו איזה חלק מהקובץ אנחנו שולחים, במילים אחרות כל פאקטה מיוצגת בתור "part".
עכשיו נגדיר את המשתנה numpart שיהיה שווה לנו לאיזה חלק אנחנו נמצאים.
ונשתמש ב fileseek כדי להזיז את הזמן בקובץ למקום המתאים.
נשלח את ה(data שהוא מכיל את הפאקטה ואת הגודל שלה), לאחר מכן נשתמש דרך ספריה שנקראת pickle ונשתמש בפונקציית dumps שהיא שולחת את ה data בתור tuple ויודעת לפענח אותו לאחר מכן נשלח את הודעה בתור ,Udp ועד שלא נקבל recve מהלקוח שאכן הקובץ הגיע בשלמותו , נראה זאת שעבר כל כמות הביטים עברו אל הclient איך שהוא מקבל הודעה finshed מהלקוח יוצא מהלולאה וסוגר את הסוקט, וכותב.finshed .

```python
def get_files():
    path = f"{os.getcwd()}/{FILES_FOLDER}"
    return os.listdir(path)


def create_str_files(files: list):
    str_files = ''
    count = 1
    for f in files:
        str_files += f'{count}:{f} '
        count += 1
    return str_files


def check_files(client, files):
    tmp = get_files()
    if files == tmp:
        return True
    else:
        files = tmp
        client.send(bytes(f'UPDATE {create_str_files(files)}', "utf8"))
        print('[UPDATE] sending update of the list of the files.')


def client_communication(user):

# Thread to handle all messages from client


    client = user.client
    # List for the names of the files
    files = get_files()
    while True:  # wait for any messages from person
        try:
            msg = client.recv(BUFSIZ)
        except:
            print('HTTP SERVER CRASHED!')
            users.remove(user)
            break
        if msg == bytes("{quit}", "utf8"):  # if message is qut - so disconnect the client
            print(f"[HTTP DISCONNECTED] {user} disconnected from the server at {time.time()}\n")
            users.remove(user)
            print(f'connected users: {users}')
            break

        elif msg.split()[0].decode('utf-8') == "download_file":
            filename = msg.split()[1].decode('utf-8')
            try:
                download(client, filename)
            except:
                check_files(client, files)
        elif msg.split()[0].decode('utf-8') == "get_files":
            files = get_files()
            client.send(bytes(f'FILES {create_str_files(files)}', "utf8"))
            print('[GET] sending the list of the files.')
        elif msg.decode('utf-8') == "UPDATE":
            check_files(client, files)
        else:  # otherwise send message to all other clients
            print(msg.decode('utf-8'))
    print("[STARTED] Waiting for HTTP connections...")
```

נסביר על הפונקציות הבאות :

()Getfile- אנחנו יוצרים path (נתיב) לתוך התיקייה files , אשר נמצאת בתוך ה serverApp .
ונחזיר את הpath .
בפונקציה ()creatstrfiles - פשוט יוצר רשימה מהקבצים הממוספרים לפי סדר, בצורה אשר נראית כמו dictionary, ומחזירה string של הרשימה של הקבצים הממוספרת.
הפונקציה ()cheak_file בודקת אם הרשימת הקבצים שנשלחה ללקוח האחרונה מעודכנת, אם איננה מעודכנת הפונקציה שולחת מחדש את הרשימה לקבצים ללקוח.
מתי משתמשים בה, כשהלקוח מכניס inpute לא תקין.
()Client_commencion - הפונקציה הזאת עובדת כתהליכון נפרד, כלומר אנחנו קוראים לה ב thread. התפקיד שלה הוא להתמודד עם כל הודעה מהלקוח.
תחילה נגדיר client שיהיה user חדש , הוא מקבל את הקבצים בעזרת הפונקציה cheak_file, אם ההודעה היא getfile זה אומר שהלקוח רוצה את רשימת הקבצים .
כאשר הוא כותב show נפתח לולאה(while(True שבה השרת מחכה להודעה כלשהי מכל user

כלשהו.
במידה וההודעה מהלקוח לא הצליחה להגיע לשרת, כלומר החזירה שגיאה כלשהי, נדפיס שהסרבר קרס ונמחק את היוזר מרשימת היוזרים.
הלקוח של appserver הוא השרת httpserver אז במידה וה appserver לא קיבל "hak" משרת http , או שהייתה תקלה בבקשת ההודעה, נדפיס שה http סרבר קרס, ונמחק אותו מרשימת היוזרים.
אם נקבל בהודעה string שבו רשום quit, אז במילים אחרות הלקוח ירצה להתנתק ונבצע התנתקות בכך שנמחוק אותו מהרשימת היוזרים ונסגור את הסוקט עם הקלינט,
אחרת אם נראה שה msg.split[0] אינו שווה ל "downloadfile" , אז נגדיר את ה filename שלנו להיות msg.split[1] שזה בעצם השם של הקובץ אותו נרצה להוריד.
לאחר מכן ננסה להשתמש בפונקציית download על ידי Try שמקבל את הלקוח הרלוונטי ואת שם הקובץ שהגדרנו ב filename ב except ננסה לתפוס שגיאה תחת הפונקציה שהגדרנו מקודם cheak_files שעליה פירטנו מקודם.
במידה msg.split[0] יהיה שווה ל "getfile" אז הלקוח ירצה את רשימת הקבצים ונעביר לו אותה בעזרת הפונקציה get_files שפירטנו עליה מקודם.
אם הmsg יהיה שווה ל string "update" נשתמש בפונקציה cheak_file כדי לעדכן את הקבצים.
אחרת נשלח את הרשימה לכל היוזרים.

```python
def wait_for_connection():

#  Wait for connecton from new clients, start new thread once connected

    while True:
        try:
            client, addr = SERVER.accept()  # wait for any new connections
            user = User(addr, client)  # create new person for connection
            print(f"[HTTP CONNECTION] {addr} connected to the server at {time.time()}")
            users.append(user)
            print(f'Connected Users: {users}')
            Thread(target=client_communication, args=(user,)).start()
        except Exception as e:
            print("[EXCEPTION]", e)
            break

    print("SERVER CRASHED")


def start_server():
    SERVER.listen(MAX_CONNETIONS)  # open server to listen for connections
    print("[STARTED] Waiting for HTTP connections...")
    ACCEPT_THREAD = Thread(target=wait_for_connection)
    ACCEPT_THREAD.start()
    ACCEPT_THREAD.join()
    SERVER.close()


def stop_server():
    SERVER.close()


if __name__ == "__main__":
    start_server()
```

עכשיו נסביר את הקוד למעלה.
ניצור פונקציה הנקראת wait_for_connection() - זו פונקציה שבה השרת מחכה ליצירת קשר מ clientys חדשים, משתמשת ב srver.accept כדי לאשר את הלקוחות, נגדיר את אותו משתמש אשר מתחבר בתור user ונוסיף אותו לרשימת היוזרים.
נשתמש ב thread שיפעיל לנו את הפונקציה הזאת כדי שנוכל לתקשר עם מספר לקוחות בו זמנית, ושתמיד נהיה בהאזנה לא משנה אם הוא מחובר או לא.
נשתמש בפונקציה start_server() כדי להפעיל את השרת, נשתמש ב Listen() כדי להגדיר את כמות

המקסימלית של הלקוחות שיכולים להתחבר לשרת הזה, לאחר מכן נגדיר את המשתנה ACCEPT_THREAD להיות ה thread אשר מפעיל לנו את הפונקציה ()wait_for_connection שהיא מקבלת משתמשים חדשים.
ויצרנו את הפונקציה ()srop_server שפשוט משתמשת בפונקציה ()close וסוגרת את ה server.
ולסיום נקרא לפונקציה ()start_server ב main כדי להפעיל את השרת.

נסביר עכשיו על הClient

```
class Client:

#    for communication with server

    if len(sys.argv) > 1:
        HOST = sys.argv[1]
    else:
        HOST = '127.0.0.1'
    PORT = 20896
    ADDR = (HOST, PORT)
    BUFSIZE = 1024

    def __init__(self):

#    constractor and send name to server

        self.isconnected = False
        self.reconnect = True
        self.exited=False
        # set a dictionary for the files to download
        self.files = {}
        self.messages = []

        self.connect_thread = Thread(target=self.connect_server)
        self.connect_thread.start()
        self.receive_thread = Thread(target=self.receive_messages)
        self.receive_thread.start()

        self.lock = Lock()
```

נגדיר מחלקה הנקראת Client, מחלקה זו נועדה כדי ליצור תקשורת עם הserverHttp.
במידה וקיבלנו ארגומנטים בטרמינל אז נוכל להגדיר את ה ip לאיזה כתובת ip שנרצה אם היא חיצונית או אם היא פנימית.
נגדיר את כתובת ה ip הראשונית שלנו להיות 127.0.0.1 את הפורט 20836 שזה בעצם 20 XXX

כמתבקש.
ונגדיר בנאי שחשוב לציין שהוא משתמש ב threads כדי להתמודד עם מצב שבו הלקוח התנתק מהשרת, וכמו שהוצג בסרטון הסברה בעזרת האינפוט "reconnect" נוכל להתחבר מחדש אל הסרבר.
בבנאי מוגדר dictionary המכונה Files ומערך בשם.massage .

```
def connect_server(self):
    flag = True
    while not self.exited:
        if self.reconnect and not self.isconnected:
            try:
                # open tcp socket for client
                self.client_socket = socket(AF_INET, SOCK_STREAM)
                self.client_socket.connect(self.ADDR)

                # open UDP with rdt for file transfer
                self.client_socketUDP = socket(AF_INET, SOCK_DGRAM)
                self.client_socketUDP.connect(self.ADDR)
                self.isconnected = True
                print("Client Connected\n")
                print(f"---------------welcome!---------------\n")
                print(f"You can download files from the list below:\n")
                print(f"For get the list of the files, input <show>\n")
                print(f"For discconnect the client, input <quit>\n")
                print(f"Input the number of the file that you want to download.\n")
                self.send_message("UPDATE")

                break
            except:
                if flag:
                    print('Waiting for HTTP Server Connection...')
                    flag = False
        else:
            flag = True
        if not self.isconnected and self.exited:
            break
```

```
def print_files(self, sp):
    print("The list of the files:")
    self.files = {}
    for i in range(1, len(sp)):
        key = sp[i].split(':')[0]
        value = sp[i].split(':')[1]
        self.files[key] = value
        print(f"{key}: {value}")
    print()
```

בפונקציה connect_server פותחים סוקט tcp ליצירת קשר עם השרת, חשוב לציין שזה סוקט מפורטוקול tcp, כמובן נעשה זאת במידה והלקוח במצב reconnect ,כלומר במצב התחברות ולא במצב שהוא כבר מחובר אל ה
sever. נפתח סוקט udp לשם העברת הקבצים ונעטוף אותה במעטפת tcp או שנוכל להגדיר במילים אחרות – RUDP.

במידה ונראה שמודפס לנו שה httpsever מחכה ל connection אנחנו נצא מהלולאה, במידה ולא נקבל את ההדפס הזה נחזור על אותה לולאה שוב ושוב כדי לקבל.connection
עכשיו נדבר על הפונקציה printfile הפונקציה הזאת מדפיסה את רשימת הקבצים בצורה יפה ומסודרת.

```
def download(self, port, size, path):
    try:
        with open(path, "wb") as file:
            size = (size // 1000) + 1
            ADDRESS_SERVER = (self.HOST, port)
            CLIENTUDP = socket(AF_INET, SOCK_DGRAM)
            for i in range(size):
                done = False
                while not done:
                    CLIENTUDP.sendto(bytes(f"part {i}", "utf-8"), ADDRESS_SERVER)
                    msg, addr = CLIENTUDP.recvfrom(self.BUFSIZE)
                    # filter messages that are not from our server
                    while (addr != ADDRESS_SERVER):
                        msg, addr = CLIENTUDP.recvfrom(self.BUFSIZE)
                    msg, msg_size = pickle.loads(msg)
                    if type(msg_size) == int and len(msg) == msg_size:
                        done = True
                    else:
                        print("lost a packet, send again.")
                file.write(msg)
                percent = round(i / size * 100, 2)
                print(f"{percent}% downloaded")
        print("File was downloaded 100%\n")

    except:
        self.send_message("UPDATE")


    finally:
        CLIENTUDP.sendto(bytes("finished", "utf-8"), ADDRESS_SERVER)
        CLIENTUDP.close()
```

נסביר על הפונקציה download הפונקציה מקבלת . port,size,path .

Path מייצג את הנתיב שממנו נוריד את הקובץ, נגדיר את Size להיות שווה לכמות החלקים שנחלק את הקובץ (נזכור שהגדרנו שגודל של שליחה של כל פאקטה להיות הגודל 1000).
הלקוח יוצר קשר udp עם ה serverונכנס ללולאה, הלולאה רצה size פעמים ובלולאה זו נוודא שהקובץ יגיע בשלמותו כמו שמתקיים ב.tcp
נשלח בלולאה איזה פאקטה אנחנו רוצים (נזכור שהפאקטות מחולקות בצורה ממוספרת לפי ,(part נקבל את ההודעה מהשרת ונכנס ללולאה שכל עוד ההודעה שנשלחה מהשרת מהכתובת הרצויה הלקוח ימשיך לקבל הודעות.
נצא מהלולאה ונקרא msg ונגדיר msg_size בעזרת כך שנפענח את ה msg בעזרת הספריה pikle בקריאה לפונקציה loads שהיא מפענחת את ה msgומחזירה tupl של ההודעה עם הגודל שלה, במידה והפאקטה שהתקבלה בגודל של ה msgSIze שפוענח(כלומר הגודל המקורי של הפאקטה שנשלחה- הם אכן גדלים שווים) נצא מהלולאה ונרשום את אותה הודעה.
במידה והגדלים אינם שווים אז כנראה פאקטות נאבדו בדרך ונשלח את אותה פאקטה שוב ושוב ושוב עד שהפאקטה התקבלה בשלמותה.

לאחר שהפאקטה התקבלה בשלמותה אז הקלינט ישלח אל הסרבר את ה stringשנקרא "finshed" אז הסרבר ידע לפענח זאת כשההורדה נשלחה בשלמותה ונסגור את הסוקט של.udp

```python
def receive_messages(self):

    receive messages from server

    while True:
        if self.isconnected:
            try:
                msg = self.client_socket.recv(self.BUFSIZE).decode('utf-8')
                sp = msg.split()
                if sp[0] == "FILES":
                    self.print_files(sp)
                elif sp[0] == "UPDATE":
                    self.files = {}
                    print("FILES UPDATE")
                    self.print_files(sp)
                elif msg.split()[0] == "UDP":
                    port = int(msg.split()[1])
                    filesize = int(msg.split()[2])
                    path = os.path.join(os.getcwd(), 'Downloads')
                    try:
                        os.mkdir(path)
                    except:
                        pass
                    download_thread = threading.Thread(target=self.download, args=(
                    port, filesize, f"{os.getcwd()}/Downloads/{msg.split()[3]}"))
                    download_thread.start()
                else:
                    print(f'client received message:{msg}')
                # make sure memory is safe to access
                self.lock.acquire()
                self.messages.append(msg)
                self.lock.release()
            except Exception as e:
                if self.isconnected:
                    self.disconnect()
        else:
            if not self.connect_thread.is_alive():
                self.connect_thread = Thread(target=self.connect_server)
                self.connect_thread.start()
```

הפונקציה הזאת – receive_massage יודעת להתמודד עם קבלת הודעות מן httpServer הפונקציה היא בלולאה אינסופית, קודם כל היא בודקת אם הקלינט מחובר, במידה והוא מחובר פותחים Try ומנסים לקבל הודעה מהשרת(לעשות recv לשרת) ולפענח אותה, במידה והפעולה נכשלה אם הלקוח עדיין מחובר מנתקים אותו.
אחרי שקיבלנו הודעה עושים לה split על ידי רווחים, אם המילה הראשונה היא Files מפעילים את הפונקציה printfile, כי זה אומר שקיבלנו את רשימת הקבצים מהשרת ואנחנו רוצים להדפיס אותה ללקוח.
אם כתוב update זה אומר שקיבלנו עידכון לרשימת קבצים מהשרת, ומפעילים את הפונקציה printfile ומדפיסים ללקוח.
אם המילה הראשונה היא UDP זה אומר שקיבלנו הודעה המורה על תחילה של העברת קובץ היא תכיל פורט, גודל קובץ ושם של קובץ.

ניצור משתנה של פורט שיבצע cast ל int של המילה השנייה בהודעה.
משתנה filesize יהיה משתנה cast של int של המילה השלישית במחרוזת אשר מגדירה לנו את הגודל של הקובץ.
נייצר path שיקח את המיקום הנוכחי ויוסיף לו עוד תקייה של download, ננסה לייצר את התקייה הזאת, במידה ולא הצלחנו זה אומר שהיא קימת נעשה pass .
ונפתח thread חדש שיפעיל את פונקציית download עם הארגומנטים פוראט פייל סייז וסטרינג שהוא שירשור של המיקום של תקיית download שהיא המילה הרביעית בהודעה שהיא המילה הרביעית בהודעה.
נדגיש כי קראנו לפונקציה download ב thread נפרד על מנת שהלקוח יוכל לבצע פעולות נוספות בזמן ההורדה ובפרט אם ההורדה לוקחת זמן רב מידי זה לא ימנע מהלקוח את המשך פעילתו הסדירה, ובכך התגברנו על בעיית leatncy .
נמשיך ב if :
אם המילה הראשונה אינה תואמת לאף אחת מהתבניות נדפיס את תוכן ההודעה ישר ללקוח.
לאחר מכן נוסיף את ההודעה שהתקבלה לרשימת ההודעות ונעבור לאיטרציה הבאה.
אם הלקוח אינו מחובר אז נבדוק אם תהליך ההתחברות עדיין קיים אם לא נפעיל מחדש.

```
    def send_message(self, msg):

#         send messages to server

        try:
            self.client_socket.send(bytes(msg, "utf8"))
        except Exception as e:
            if self.isconnected:
                print('HTTP Server Crashed!')
                self.isconnected = False

    def get_messages(self):

#         returns a list of str messages

        try:
            messages_copy = self.messages[:]

            # make sure memory is safe to access
            self.lock.acquire()
            self.messages = []
            self.lock.release()
            return messages_copy
        except:
            return self.messages[:]

    def disconnect(self):
        if not self.isconnected:
            return
        print("Client Disconnected.")
        self.send_message("{quit}")
        self.client_socket.close()
        self.isconnected = False
    def exit(self):
        self.exited = True
```

()send_massage פונקציה שמקבלת string ושולחת אותו לשרת, אם לא הצליחה לשלוח לשרת

למרות שהשרת מחובר אנו נדפיס שהשרת http קרס ונעבור למצב מנותק.
Get_massage בפונקציה זאת אנחנו נחזיר רשימה של string של massage בעצם נרצה לוודא שהזיכרון שמור לגישה, נבצע זאת בעזרת הפונקציה lock.acquire ובעזרת lock.release ונחזיר את מערך ההודעות.
הפונקציה ()disconnect אומרת שאם הלקוח לא מחובר אל תבצע כלום, אחרת נשלח msg עם ה string "quit"ונסגור את הסוקט ונגדיר אותו ב Isconnected בתור false.

כעת סיימנו לכתוב את המחלקה client וכעת נתחיל לכתוב מה שקורה כאילו ב"main"
אנחנו ניצור מופע של המחלקה Client שנקרא client.
עם הרצת פעולת הבנאי אז יפתח ה thread של ההתחברות וה thread של קבלת ההודעות.

```
def run():
    # Create a client object
    client = Client()

    while True:
        msg = input()
        if not client.isconnected:
            if msg == 'reconnect':
                client.reconnect = True
            else:
                print("CLIENT DISCONNECTED")
        elif msg == 'show':
            client.send_message("get_files")
        elif msg == 'quit':
            client.reconnect = False
            client.disconnect()
            print("for reconnect input <reconnect>")
        elif msg in client.files.keys():
            client.send_message(f'download_file {client.files[msg]}')
        else:
            print(f"<{msg}> not exist")
            client.send_message("UPDATE")
if __name__ == '__main__':
    run()
```

הקוד הזה יפעל לאחר שהסתיימה פעולת הבנאי, אנחנו נכנס ללולאה אינסופית.
נגדיר משתנה msg שיכיל בתוכו את הקלט מהמשתמש, אם הקליינט מנותק וה msg הוא reconnect נחבר את ה client על ידי client.reconnect= true.
לכל הודעה אחרת במצב כזה נדפיס שה client מנותק.
אחרת אם ההודעה היא show נשלח דרך ה client הודעה get_files על מנת לבקש את רשימת הקבצים.
אחרת אם ההודעה היא quit נעביר את ה client למצב מנותק, ונקרא לפונקציית disconnect של client שתנתק אותו.
לאחר מכן נדפיס ללקוח שהוא יכול לכתוב את המילה reconnect ולהתחבר מחדש.

אחרת אם ההודעה היא מפתח בdictionary הקבצים של הלקוח כלומר קיימת רשימת קבצים שמוצגים ללקוח כמפתח וערך.
על מנת להוריד את הקובץ, המשתמש יזין את המפתח של הקובץ, ולכן נשלח לשרת במקרה כזה, את ההודעה download_file ושם הקובץ, כלומר ה value במילון הקבצים **באמצעות** המפתח שהוא ה inpute שהתקבל- הוא Msg, אחרת אם ההודעה לא מהתבניות אנחנו נדפיס שההודעה לא קיימת msg not exist, ונשלח לשרת את ההודעה update על מנת לקבל עידכון אם השתנה משהו ברשימת הקבצים.
סיימנו פה את הclient.

ועכשיו נספר על הhttpServer:

מכוון שהסברנו על כל הפונקציות שהשתמשנו בהם ב Client וחלקן מאוד דומות לאלו שבשרת הזה אנו נסביר באופן כללי מה עושים בפונקציות האחרות.

```
class AppServer:

#   for communication with server

    if len(sys.argv) > 1:
        HOST = sys.argv[1]
    else:
        HOST = '127.0.0.1'
    PORT = 30242
    ADDR = (HOST, PORT)
    BUFSIZE = 1024

    def __init__(self, client):

#        constractor send name to server
        self.isconnected = False
        self.reconnect = True
        self.messages = []
        self.client = client
        self.connect_thread = Thread(target=self.connect_server)
        self.connect_thread.start()
        self.receive_thread = Thread(target=self.receive_messages)
        self.receive_thread.start()
        self.lock = Lock()

    def connect_server(self):
        flag = True
        while True:
            if self.reconnect and not self.isconnected:
                try:
                    # open tcp socket for client
                    self.client_socket = socket(AF_INET, SOCK_STREAM)
                    self.client_socket.connect(self.ADDR)

                    # open UDP with rdt for file transfer
                    self.client_socketUDP = socket(AF_INET, SOCK_DGRAM)
                    self.client_socketUDP.connect(self.ADDR)
                    self.isconnected = True
                    print("Connected to the App Server\n")
                    self.send_message(bytes('App Server is ready',"utf8"))
                    time.sleep(0.1)
                    self.send_to_server(bytes('get_files',"utf8"))
                    break
                except:
                    if flag:
                        print('Waiting for App Server Connection...')
                        flag = False
            else:
                flag = True
```

C

נסביר בכללי על ה class הזה, זה מחלקה שדומה לclass שעשינו ב client מכוון ששרת http הוא בעצם לקוח בעיני המסתכל על הקשר בינו לבין הAppServer , הוא מקבל הודעות מהשרת הזה ולאחר מכן שולח אותם ל.client המקורי.

ההבדלים בינו למחלקת client ב client הם:

ב reseve_massage פה, במקום לבדוק את תוכן ההודעה הוא ישר שולח את ההודעה כמו שהיא-

מכוון שהוא מעביר ל client המקורי את הקבצים מהשרת כמו שהם .
בפונקצית send_massage במקום לשלוח את ההודעה לשרת אליו הוא מחובר, הוא מורה על שליחה דרך המשתמשים ברשימה הגלובלית users ,שמכילה את המשתמשים שמחוברים לשרת http ובעצם הם שולחים הודעות ללקוחות של השרת.
אם הוא לא הצליח לשלוח למרות שהוא מחובר נודיע כי ה appserver קרס וננתק את הקשר איתו.

נוסיף את הפונקציה sendtoserver ששולחת הודעה ל appserver ובמידה ולא הצליח למרות שהוא מחובר נודיע כי הסרבר קרס ונתנתק.

לאחר כתיבת המחלקה, נעבור לתהליך הראשי, הפעם התהליך הראשי דומה מאוד לתהליך הראשי ב appserver , נוסיף משתנה global שנקרא app_server שיכיל בתוכו מופע של מחלקת appserver **כלומר** תופעל פעולת הבנאי של appserver וכך תהיה תקשורת עם appserver במקביל לפעולת השרת http יפתחו thred ים נפרדים של התחברות וקבלת הודעות מה appserver.
ההבדל בין הפונקציה client_communction שנמצאת ב AppServer לפה היא, שבמקום לבדוק את תוכן ההודעה שהתקבלה, נבדוק אם ה appserver מחובר , אם כן נשלח הודעה דרך האוביקט appserver לשרת עם תוכן ההודעה שהתקבלה, באמצעות קריאה לפונקציה senttoserver של appserver.
אחרת אם ה appserver מנותק נשלח הודעה ללקוח שה appserver מנותק.
כל שאר הפעולות זהות לפעולות שהסברנו עליהם ב.appserver, כי הhttpServer מתפקד בתור שרת לClient.

```python
def connect_server(self):
    flag = True
    while True:
        if self.reconnect and not self.isconnected:
            try:
                # open tcp socket for client
                self.client_socket = socket(AF_INET, SOCK_STREAM)
                self.client_socket.connect(self.ADDR)

                # open UDP with rdt for file transfer
                self.client_socketUDP = socket(AF_INET, SOCK_DGRAM)
                self.client_socketUDP.connect(self.ADDR)
                self.isconnected = True
                print("Connected to the App Server\n")
                self.send_message(bytes('App Server is ready',"utf8"))
                time.sleep(0.1)
                self.send_to_server(bytes('get_files',"utf8"))
                break
            except:
                if flag:
                    print('Waiting for App Server Connection...')
                    flag = False
        else:
            flag = True


def receive_messages(self):

    receive messages from server

    while True:
        if self.isconnected:
            try:
                msg = self.client_socket.recv(self.BUFSIZE)
                print('[RECEIVE] received a message from the App Server')
                self.send_message(msg)
                # make sure memory is safe to access
                self.lock.acquire()
                self.messages.append(msg)
                self.lock.release()
            except Exception as e:
                if self.isconnected:
                    self.disconnect()
        else:
            if not self.connect_thread.is_alive():
                self.connect_thread = Thread(target=self.connect_server)
                self.connect_thread.start()

def send_message(self, msg):
```

```python
def send_message(self, msg):

    send messages to server

    try:
        self.client.send(msg)
        print('[SENDING] from HTTP to the Client')
    except Exception as e:
        if self.isconnected:
            print('App Server Crashed!')
            self.isconnected = False


def get_messages(self):

     returns a list of str messages

    try:
        messages_copy = self.messages[:]

        # make sure memory is safe to access
        self.lock.acquire()
        self.messages = []
        self.lock.release()
        return messages_copy
    except:
        return self.messages[:]

def disconnect(self):
    if not self.isconnected:
        return
    print(f"{self} Disconnected.")
    self.send_to_server(bytes("{quit}", "utf8"))
    self.isconnected = False

def send_to_server(self, msg):

   send messages to server

    try:
        self.client_socket.send(msg)
        print(f'[SENDING] from HTTP to {self}')
    except Exception as e:
        if self.isconnected:
            print('Server Crashed!')
            self.isconnected = False

def __repr__(self):
    return f"APP Server({self.ADDR})"
```

```python
# GLOBAL CONSTANTS
HOST = ''
PORT = 20896
MAX_CONNETIONS = 10
ADDR = (HOST, PORT)
BUFSIZ = 1024
FILES_FOLDER = 'Files'

# GLOBAL VARIABLES

users = []
SERVER = socket(AF_INET, SOCK_STREAM)
SERVER.bind(ADDR)  # set up server



def client_communication(user):

#    Thread to handle all messages from client

    client = user.client
    while True:  # wait for any messages from person
        try:
            msg = client.recv(BUFSIZ)
        except:
            users.remove(user)
            break
        if msg == bytes("{quit}", "utf8"):  # if message is qut - so disconnect the client
            print(f"[DISCONNECTED] {user} disconnected from the server at {time.time()}\n")
            user.disconnect()
            users.remove(user)
            print(f'connected users: {users}')
            client.close()
            break
        else:  # otherwise send message to all other clients
            if user.app_server.isconnected:
                print(f'[RECEIVE] received a messa4ge from {user}')
                user.app_server.send_to_server(msg)
            else:
                client.send(bytes(f'NO APP_SERVER CONNECTION', "utf8"))
    print('[DISCONNECTION] Client Disconnected')
```

```python
def wait_for_connection():

#    Wait for connecton from new clients, start new thread once connected

    while True:
        try:
            client, addr = SERVER.accept()  # wait for any new connections
            user = User(addr, client)  # create new person for connection
            print(f"[CONNECTION] {addr} connected to the server at {time.time()}")
            users.append(user)
            print(f'Connected Users: {users}')
            Thread(target=client_communication, args=(user,)).start()
        except Exception as e:
            print("[EXCEPTION]", e)
            break

    print("SERVER CRASHED")


def start_server():
    SERVER.listen(MAX_CONNETIONS)  # open server to listen for connections
    print("[STARTED] Waiting for connections...")
    ACCEPT_THREAD = Thread(target=wait_for_connection)
    ACCEPT_THREAD.start()
    ACCEPT_THREAD.join()
    SERVER.close()


def stop_server():
    SERVER.close()


if __name__ == "__main__":
    start_server()
```

סוף קודים של האפליקציה.

כיצד המערכת מתגברת על איבוד חבילות

לפני פתיחת socket של ה UDP הclient מקבל את גודל הקובץ שאותו הוא ביקש להוריד, והclient "מחלק" את גודל הקובץ לחלקים של 1000 בתים והאחרון הוא השארית שהיא קטנה שווה ל1000 . כמות החלקים זאת הכמות של packets שהוא מצפה לקבל, וכל packet תהיה בגודל 1000 חוץ מהאחרון, והclient לא ינוח עד שכל פעם הוא יקבל את כל הגודל של packet שאמורה להגיע.

כל העברה של כל packet (חלק מהקובץ הרצוי) מתבצעת כך: הclient מבקש את החלק בקובץ בגודל מסוים והוא מצפה לקבל את כל גודל החלק שביקש.
העברה של כל חלק בקובץ מתבצעת שהclient עובר בלולאה על כמות החלקים של הקובץ (כמות packets שמרכיבות את הקובץ עד גודל 1000), ולפי הסדר **בכל איטרציה i** מבקש את **החלק ה i**

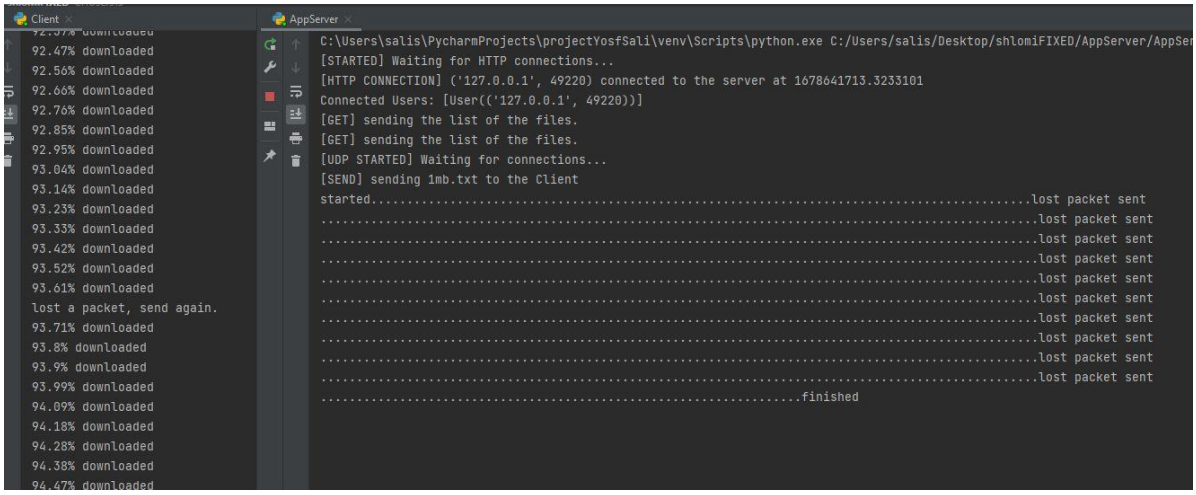
מהשרת .
מצד השרת- כאשר השרת מקבל הודעה של איזה חלק לשלוח הוא מבצע seek למיקום שממנו מתחיל החלק, (1000 , i)*ומבצע קריאה של 1000 בתים מהנקודה שבה הוא נמצא.
לאחר שהוא קורא את החלק הנדרש השרת שולח הודעת UDP המורכבת מ tuple שמכיל את המחרוזת שנקראה ואת הגודל שלה.
מהצד של הלקוח- לאחר שהלקוח שולח את הבקשה לחלק, הוא נכנס ללולאה שנעצרת כאשר הוא מקבל הודעת UDP משרת (בכתובת שאליו הוא מחובר), המכילה tuple המורכב משתי מחרוזות, המחרוזת הראשונה זה מייצג תחלק המקורי בקובץ, והמחרוזת השנייה היא מספר המיצג את הגודל המדויק של המחרוזת הראשונה.
במידה והתקבלה packet שלא עומדת בתנאי של ה) tupleשזה אומר שהגודל של ה stringשנשלח ב tuple שזה בעצם גודל הpacket שנשלחה, לא תואמת את הגודל של מה שציפנו לקבל), נדפיס שנאבדה packe, ונשלח שוב את הבקשה לאותו חלק, וברגע שנגמרת הלולאה, זה אומר שקיבלנו את החלק בשלמותו ונוכל לעבור לבקש את החלק הבא.
בסיום לאחר שה הclient מקבל את כל חלקי הקובץ הוא שולח לסרבר הודעה finished וסוגר את UDP socket. וכאשר הסרבר מקבל את ההודעה הזאת, גם הוא סוגר את הsocket וכך מסתיימת העברת הקבצים בשלמות ולא נשארים sockets פתוחים.

הנה דוגמא לזיוף חבילה-

על כל מאה חבילות ביקשנו לקרוא 10 ביטים פחות ושלחנו פאקטה אבודה ורואים שגם למרות שהוא קיבל פאקטה אבודה עדיין המערכת מצליחה להתגבר על זה ולהוריד את כל הקובץ ולקבל .finished

```python
def download(self, port, size, path):
    try:
        with open(path, "wb") as file:
            size = (size // 1000) + 1
            ADDRESS_SERVER = (self.HOST, port)
            CLIENTUDP = socket(AF_INET, SOCK_DGRAM)
            for i in range(size):
                done = False
                while not done:
                    CLIENTUDP.sendto(bytes(f"part {i}", "utf-8"), ADDRESS_SERVER)
                    msg, addr = CLIENTUDP.recvfrom(self.BUFSIZE)
                    # filter messages that are not from our server
                    while (addr != ADDRESS_SERVER):
                        msg, addr = CLIENTUDP.recvfrom(self.BUFSIZE)
                    msg, msg_size = pickle.loads(msg)
                    if type(msg_size) == int and len(msg) == msg_size:
                        done = True
                    else:
                        print("lost a packet, send again.")
                file.write(msg)
                percent = round(i / size * 100, 2)
                print(f"{percent}% downloaded")
        print("File was downloaded 100%\n")

    except:
        client.send_message("UPDATE")


    finally:
        CLIENTUDP.sendto(bytes("finished", "utf-8"), ADDRESS_SERVER)
        CLIENTUDP.close()
```

```python
    client.send(msg)

    with open(filepath, "rb") as file:
        msg, addr = SERVERUDP.recvfrom(BUFSIZ)
        msg = msg.decode('utf-8')
        print(f'[SEND] sending {filename} to the Client')
        print('started', end='')
        c = 7
        while (msg != "finished"):
            msg = msg.split()
            if msg[0] == "part":
                numpart = int(msg[1])
                file.seek(numpart * SIZE)
                data = file.read(SIZE)
                data_size = len(data)
                c += 1
                if c % 100 == 0:
                    data = file.read(SIZE-10)
                    print("lost packet sent")
                    time.sleep(5)

                data = (data, data_size)
                data = pickle.dumps(data)
                SERVERUDP.sendto(data, addr)
                print('.', end='')

            msg, addr = SERVERUDP.recvfrom(BUFSIZ)
            msg = msg.decode('utf-8')
        print('finished')
        SERVERUDP.close()
        ports[addr[1]] = True
```



כיצד המערכת מתגברת על בעיות latency :

ראינו במהלך ההסברים על האפליקציה שלנו שבפונקציית download השתמשנו ב thread כדי להצליח להתמודד עם מצבים של ריבוי משימות ועם מצבים של מצבי שהיה.

מצבי שהיה זה אומר שאנחנו בעצם לא מקבלים מהשרת את כל הpacket במלואה, אלא אנחנו עדיין ממתינים לה, במצבים כאלה יכול להיות מצב שהתור שמעביר לנו את הבתים דרך הsocket- מלא, ולכן כאשר קרה מקרה של השהיה אנחנו השתמשנו בלולאה שתבקש מהשרת שימשיך מאותו נקודה בדיוק שהפסקנו , עד שנקבל את הpacket במלואה.
בנוסף, השתמשנו בפונקציית sleep כדי לדמות עיכוב(השהיה) של שליחת הpacket .

לאחר שהרצנו את שרת האפליקציה כמו שמוצג בסרטון המצורף, נכנס אל - Wireshark ונבחר את ממשק "LoopBack" (ממשק המאפשר לנו להסניף תעבורה אשר מתבצעת באותו מכשיר) .

לאחר מכן נשתמש בפילטרים הבאים כדי לסנן רק את התוצאות הלוונטיות בשבילנו :

Ip.dst == 127.0.0.1 and (udp || tcp)

מכאן נקבל: (נפרט על כל הדגלים באופן מפורט בהמשך)

נשים לב שהפורט שיוצר קשר הוא הפורט 30242 כמו שהגדרנו בקוד (30YYY כמתבקש) שהוא בעצם הפורט שמאזין אליו שרת האפלקציה , בנוסף הדגל SYN דלוק בזמן שיש פנייה לפורט 30242 , או במילים אחרות כאשר יש פנייה ליצירת קשר משרת הHTTP שיצרנו אל שרת האפליקציה.

39	63.784302071	127.0.0.1	127.0.0.1	TCP	74	48934 → 30242 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=3861270995 TSecr=0 WS=128	Set	Not set	N
40	63.784331977	127.0.0.1	127.0.0.1	TCP	74	30242 → 48934 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=3861270995 TSecr=3861270995 WS=1…	Set	Set	N
41	63.784354117	127.0.0.1	127.0.0.1	TCP	66	48934 → 30242 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=3861270995 TSecr=3861270995	Not set	Set	N
42	63.785058936	127.0.0.1	127.0.0.1	TCP	75	48934 → 30242 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=9 TSval=3861270996 TSecr=3861270995	Not set	Set	S
43	63.785242058	127.0.0.1	127.0.0.1	TCP	66	30242 → 48934 [ACK] Seq=1 Ack=10 Win=65536 Len=0 TSval=3861270996 TSecr=3861270996	Not set	Set	N
44	63.787116516	127.0.0.1	127.0.0.1	TCP	122	30242 → 48934 [PSH, ACK] Seq=1 Ack=10 Win=65536 Len=56 TSval=3861270998 TSecr=3861270996	Not set	Set	S
45	63.787142839	127.0.0.1	127.0.0.1	TCP	66	48934 → 30242 [ACK] Seq=10 Ack=57 Win=65536 Len=0 TSval=3861270998 TSecr=3861270998	Not set	Set	N
46	63.844842870	127.0.0.1	127.0.0.1	TCP	66	48934 → 30242 [RST, ACK] Seq=10 Ack=57 Win=65536 Len=0 TSval=3861271055 TSecr=3861270998	Not set	Set	N
47	63.844918178	127.0.0.1	127.0.0.1	TCP	74	48944 → 30242 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=3861271056 TSecr=0 WS=128	Set	Not set	N
48	63.844930088	127.0.0.1	127.0.0.1	TCP	74	30242 → 48944 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=3861271056 TSecr=3861271056 WS=1…	Set	Set	N
49	63.844943506	127.0.0.1	127.0.0.1	TCP	66	48944 → 30242 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=3861271056 TSecr=3861271056	Not set	Set	N
50	64.018639515	127.0.0.1	127.0.0.1	TCP	75	48944 → 30242 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=9 TSval=3861271229 TSecr=3861271056	Not set	Set	S
51	64.018665811	127.0.0.1	127.0.0.1	TCP	66	30242 → 48944 [ACK] Seq=1 Ack=10 Win=65536 Len=0 TSval=3861271229 TSecr=3861271229	Not set	Set	N
52	64.018888015	127.0.0.1	127.0.0.1	TCP	122	30242 → 48944 [PSH, ACK] Seq=1 Ack=10 Win=65536 Len=56 TSval=3861271229 TSecr=3861271229	Not set	Set	S
53	64.018896154	127.0.0.1	127.0.0.1	TCP	66	48944 → 30242 [ACK] Seq=10 Ack=57 Win=65536 Len=0 TSval=3861271230 TSecr=3861271229	Not set	Set	N

עכשיו נראה שהפורט שיוצר קשר הוא הפורט 20836 כמו שהגדרנו בקוד (20XXX כמתבקש) שהוא בעצם הפורט שמאזין אליו שרת הHTTP שיצרנו , וזה בעצם תחילת הקשר בין הלקוח לבין שרת הHTTP, ונוכל לראות ששרת הHTTP מפנה את בקשותיו של הלקוח אל שרת האפליקציה.

נוכל לדעת זאת מכיוון שאחרי כל ack ששולח שרת הHTTP אל הלקוח דרך פורט 20836, נראה לאחר מכן שנשלחת פאקטה [psh,ack] משרת הHTTP אל שרת האפליקציה דרך פורט 30242.

68	87.416779704	127.0.0.1	127.0.0.1	TCP	74	60470 → 20836 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=3861294627 TSecr=0 WS=128
69	87.416803422	127.0.0.1	127.0.0.1	TCP	74	20836 → 60470 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=3861294627 TSecr=3861294627
70	87.416818033	127.0.0.1	127.0.0.1	TCP	66	60470 → 20836 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=3861294627 TSecr=3861294627
71	87.441179889	127.0.0.1	127.0.0.1	TCP	75	60470 → 20836 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=9 TSval=3861294652 TSecr=3861294627
72	87.441338782	127.0.0.1	127.0.0.1	TCP	66	20836 → 60470 [ACK] Seq=1 Ack=10 Win=65536 Len=0 TSval=3861294652 TSecr=3861294652
73	87.447719498	127.0.0.1	127.0.0.1	TCP	75	48944 → 30242 [PSH, ACK] Seq=10 Ack=57 Win=65536 Len=9 TSval=3861294658 TSecr=3861271229
74	87.448034939	127.0.0.1	127.0.0.1	TCP	122	30242 → 48944 [PSH, ACK] Seq=57 Ack=19 Win=65536 Len=56 TSval=3861294659 TSecr=3861294658
75	87.448054049	127.0.0.1	127.0.0.1	TCP	66	48944 → 30242 [ACK] Seq=19 Ack=113 Win=65536 Len=0 TSval=3861294659 TSecr=3861294659
76	87.454784494	127.0.0.1	127.0.0.1	TCP	122	20836 → 60470 [PSH, ACK] Seq=1 Ack=10 Win=65536 Len=56 TSval=3861294665 TSecr=3861294652
77	87.454803208	127.0.0.1	127.0.0.1	TCP	66	60470 → 20836 [ACK] Seq=10 Ack=57 Win=65536 Len=0 TSval=3861294665 TSecr=3861294665
78	89.591540116	127.0.0.1	127.0.0.1	TCP	87	60470 → 20836 [PSH, ACK] Seq=10 Ack=57 Win=65536 Len=21 TSval=3861296802 TSecr=3861294665
79	89.599750714	127.0.0.1	127.0.0.1	TCP	87	48944 → 30242 [PSH, ACK] Seq=19 Ack=113 Win=65536 Len=21 TSval=3861296810 TSecr=3861294659
80	89.600715310	127.0.0.1	127.0.0.1	TCP	91	30242 → 48944 [PSH, ACK] Seq=113 Ack=40 Win=65536 Len=25 TSval=3861296811 TSecr=3861296810
81	89.600731351	127.0.0.1	127.0.0.1	TCP	66	48944 → 30242 [ACK] Seq=40 Ack=138 Win=65536 Len=0 TSval=3861296811 TSecr=3861296811
82	89.600902242	127.0.0.1	127.0.0.1	TCP	91	20836 → 60470 [PSH, ACK] Seq=57 Ack=31 Win=65536 Len=25 TSval=3861296812 TSecr=3861296802
83	89.600914108	127.0.0.1	127.0.0.1	TCP	66	60470 → 20836 [ACK] Seq=31 Ack=82 Win=65536 Len=0 TSval=3861296812 TSecr=3861296812

כאן נוכל לראות את הורדת הקובץ של הלקוח מהשרת תחת פרוטוקול UDP

*Loopback: lo

Statistics Telephony Wireless Tools Help

Destination	Protocol	Length	Info	SYN	ACK	PSH
127.0.0.1	TCP	87	60470 → 20836 [PSH, ACK] Seq=10 Ack=57 Win=65536 Len=21 TSval=3861296802 TSecr=3861294665	Not set	Set	Set
127.0.0.1	TCP	87	48944 → 30242 [PSH, ACK] Seq=19 Ack=113 Win=65536 Len=21 TSval=3861296810 TSecr=3861294659	Not set	Set	Set
127.0.0.1	TCP	91	30242 → 48944 [PSH, ACK] Seq=113 Ack=40 Win=65536 Len=25 TSval=3861296811 TSecr=3861296810	Not set	Set	Set
127.0.0.1	TCP	66	48944 → 30242 [ACK] Seq=40 Ack=138 Win=65536 Len=0 TSval=3861296811 TSecr=3861296811	Not set	Set	Not s
127.0.0.1	TCP	91	20836 → 60470 [PSH, ACK] Seq=57 Ack=31 Win=65536 Len=25 TSval=3861296812 TSecr=3861296802	Not set	Set	Set
127.0.0.1	TCP	66	60470 → 20836 [ACK] Seq=31 Ack=82 Win=65536 Len=0 TSval=3861296812 TSecr=3861296812	Not set	Set	Not s
127.0.0.1	UDP	48	52362 → 30242 Len=6			
127.0.0.1	UDP	1065	30242 → 52362 Len=1023			
127.0.0.1	UDP	48	52362 → 30242 Len=6			
127.0.0.1	UDP	1065	30242 → 52362 Len=1023			
127.0.0.1	UDP	48	52362 → 30242 Len=6			
127.0.0.1	UDP	1065	30242 → 52362 Len=1023			
127.0.0.1	UDP	48	52362 → 30242 Len=6			
127.0.0.1	UDP	1065	30242 → 52362 Len=1023			
127.0.0.1	UDP	48	52362 → 30242 Len=6			
127.0.0.1	UDP	1065	30242 → 52362 Len=1023			
127.0.0.1	UDP	48	52362 → 30242 Len=6			
127.0.0.1	UDP	1065	30242 → 52362 Len=1023			
127.0.0.1	UDP	48	52362 → 30242 Len=6			
127.0.0.1	UDP	1065	30242 → 52362 Len=1023			
127.0.0.1	UDP	48	52362 → 30242 Len=6			
127.0.0.1	UDP	1065	30242 → 52362 Len=1023			
127.0.0.1	UDP	49	52362 → 30242 Len=7			
127.0.0.1	UDP	1065	30242 → 52362 Len=1023			
127.0.0.1	UDP	49	52362 → 30242 Len=7			
127.0.0.1	UDP	1065	30242 → 52362 Len=1023			
127.0.0.1	UDP	49	52362 → 30242 Len=7			
127.0.0.1	UDP	1065	30242 → 52362 Len=1023			
127.0.0.1	UDP	49	52362 → 30242 Len=7			
127.0.0.1	UDP	1065	30242 → 52362 Len=1023			
127.0.0.1	UDP	49	52362 → 30242 Len=7			
127.0.0.1	UDP	1065	30242 → 52362 Len=1023			
127.0.0.1	UDP	49	52362 → 30242 Len=7			
127.0.0.1	UDP	1065	30242 → 52362 Len=1023			
127.0.0.1	UDP	49	52362 → 30242 Len=7			

בצילום מסך מתחת נראה את בקשה לסגירת קשר ושאכן היא מתבצעת כמו שצריך

127.0.0.53	127.0.0.1	DNS	128	Standard query response 0x8e0c AAAA media-mrs2-2.cdn.whatsapp.net AAAA 2a03:2880:
127.0.0.1	127.0.0.1	TCP	72	60470 → 20836 [PSH, ACK] Seq=31 Ack=82 Win=65536 Len=6 TSval=3861302809 TSecr=386
127.0.0.1	127.0.0.1	TCP	66	20836 → 60470 [FIN, ACK] Seq=82 Ack=37 Win=65536 Len=0 TSval=3861302822 TSecr=386
127.0.0.1	127.0.0.1	TCP	66	60470 → 20836 [FIN, ACK] Seq=37 Ack=83 Win=65536 Len=0 TSval=3861302822 TSecr=386
127.0.0.1	127.0.0.1	TCP	66	20836 → 60470 [ACK] Seq=83 Ack=38 Win=65536 Len=0 TSval=3861302822 TSecr=38613028
127.0.0.1	127.0.0.1	TCP	66	48944 → 30242 [FIN, ACK] Seq=40 Ack=138 Win=65536 Len=0 TSval=3861311580 TSecr=38
127.0.0.1	127.0.0.1	TCP	66	30242 → 48944 [ACK] Seq=138 Ack=41 Win=65536 Len=0 TSval=3861311628 TSecr=3861311
127.0.0.1	127.0.0.1	TCP	66	30242 → 48944 [FIN, ACK] Seq=138 Ack=41 Win=65536 Len=0 TSval=3861313326 TSecr=38
127.0.0.1	127.0.0.1	TCP	66	48944 → 30242 [ACK] Seq=41 Ack=139 Win=65536 Len=0 TSval=3861313326 TSecr=3861313

מכאן נסביר על כל הדגלים :

- "Sequence Number" (בקיצור- seq) :

אנחנו מצפים מפרוטוקול TCP שהמידע יגיע בסדר הנכון,הרי התפקיד של שכבת התעבורה הוא לפרק את המידע לסיגמנטים(בבתים) משכבת האפליקציה ולהעביר את המידע אל השכבות הבאות ואז המידע נשלח.

המידע שנשלח בסיגמנטים יכול להגיע לא טוב, כלומר הבתים שנשלחים יכולים להשלח לא לפי הסדר, ולכן החבילה לא תגיע בשלמותה.

לכן בפרוטוקול TCP כאשר המידע נשלח נוכל לדעת מה המספר הסידורי של הבית(המיקום שלו),ואז נוכל לדאוג שהמידע ישלח לפי סדר של מספור הבתים.

- "ACKnowledgement" (בקיצור- ACK) :

הרי המידע מגיע ברצף מסודר של בתים,אז ACK מציין את המספר הסידורי של הבית הבא שמצופה להתקבל ולכן המידע התקבל באופן תקין,המקבל (receiver) מקבל את הseq מהשולח (sender) וכדי לדעת מאיפה להתחיל את קבלת הבתים הבאים אז המקבל שולח את הseq הנוכחי של השולח + גודל המידע בבתים שזה בעצם המספר של ה-ACK (נסיק מכך שיש שני stream של מידע,רצף של בתים משני הצדדים).

- "SYNchronize" (בקיצור- SYN) :

SYN היא החבילה הראשונה ביצירת הקשר, הלקוח שולח לשרת חבילה שבה הוא מבקש ליצור איתו קשר, המכילה את המספר הסדורי ההתחלתי(seq) ליצירת קשר.

ולפי מה שראינו ב – WireShark :

[syn]-בקשת תחילת קשר(הסברנו ופירטנו על כך לפני תחילת התרגיל)

[syn,ack]-השרת מחזיר ללקוח שהקשר הוקדם בהצלחה(syn), ושולח בחזרה את המספר הסידורי ההתחלתי של הבתים שנשלחו מהשרת ללקוח(ack)

[ack]-הלקוח מאשר את החבילה [syn,ack] ועכשיו גם הלקוח וגם השרת מסוכרנים על המספרים הסידוריים הראשונים (הלקוח שולח את המספר הסידורי הבא שהוא מצפה לקבל)

[push,ack]-הלקוח מבקש מהשרת שיעבד את המידע ושולח לו את הack שממנו הוא מצפה שהמידע יתחיל להתקבל.

[ack]-השרת שולח ללקוח שהוא מאשר את הבקשה ואת הack הנוכחי

[push,ack]השרת מתחיל לעבד את המידע ל — ack,נוכל לראות שקיבלנו את המידע לפי ארוך המידע "len"

[ack]-הלקוח מאשר שהוא קיבל את החבילה

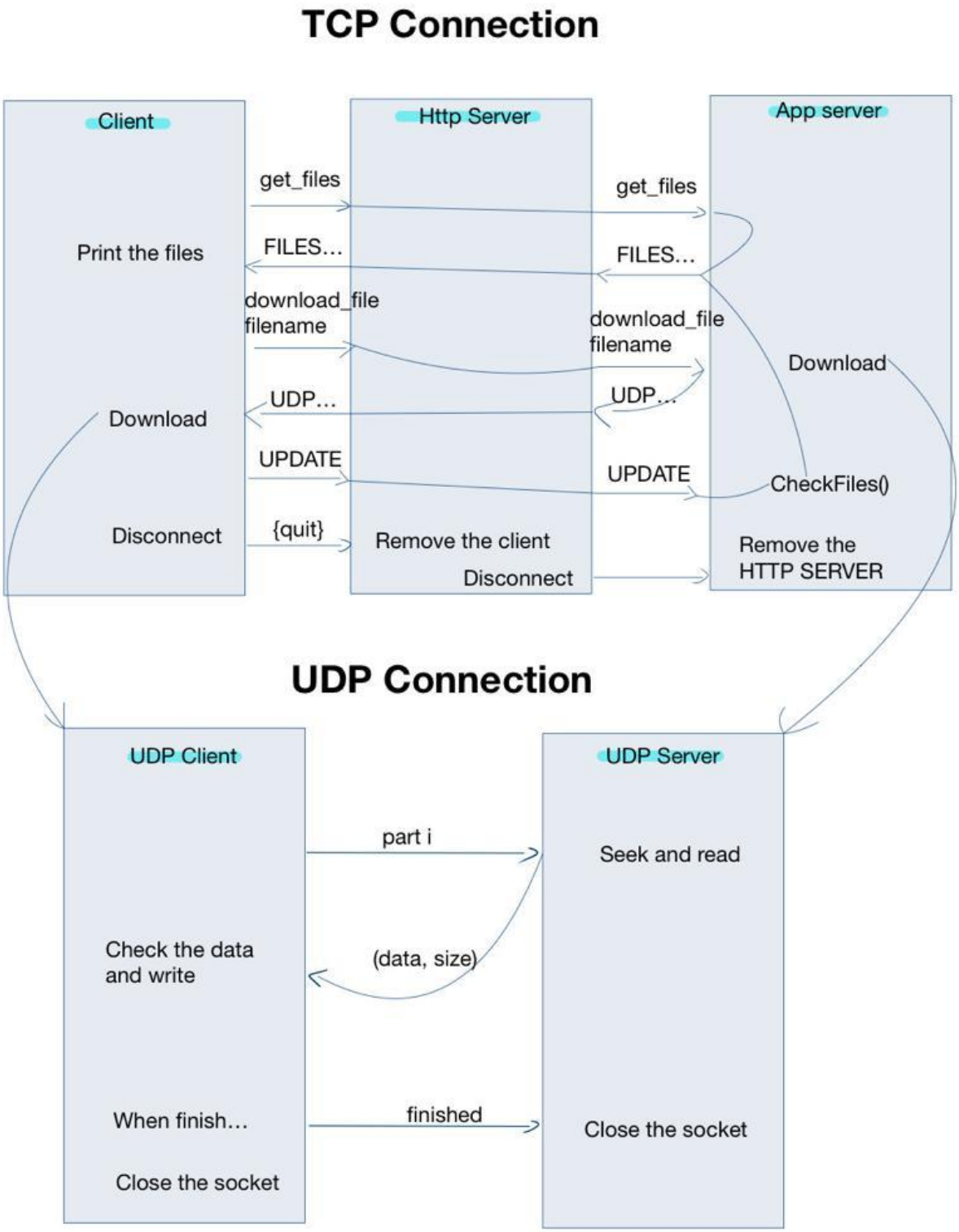
[fin,ack]-שהדגל של ה-fin דלוק אז הלקוח מבקש לסגור את הקשר ושהכל התנהל כראוי,כמובן הוא שולח את זה עם המספר הסדורי (ack) שהוא מצפה לקבל את סגירת הקשר.

[fin,ack]-השרת מאשר את סגירת בקשר

[ack]-הלקוח שולח שהוא קיבל את האישור לסגירת הקשר.

הקשר נסגר

תרשים זרימה
-AppServer-



DHCP

DHCP : (Dynamic Host Configuration Protocol)

DHCP הינו פרוטוקול רשת המאפשר למכשירים שונים לקבל כתובת IP באופן אוטומטי.

אז איך עובד פרוטוקול DHCP ?

- עבור מכשיר A שרוצה לקבל כתובת IP , על המכשיר להיות מחובר לרשת שבה DHCP משתמש.

- מכשיר A שולח בקשה Broadcast (לכל המחשבים מאותה רשת מקומית) מסוג "Discover" תחת פרוטוקול DHCP, שבקשה זו מפורשת כבקשה לקבלת כתובת IP.

-שרתי DHCP מאותה רשת מקבלים את פאקטת "Discover" שנשלחה על ידי מכשיר A, ואותם שרתים שולחים לו פאקטה מפרוטוקול DHCP מסוג "Offer", כאשר אותה פאקטה מכילה את הכתובת IP החדשה שאותו שרת DHCP מציע למכשיר A .

- נניח ומכשיר A קיבל את ההצעה של שרת B ואכן רוצה את ה- IP שהשרת B הציע לו, מכשיר A ישלח פאקטה Broadcast עם הכתובת IP החדשה ופאקטה זו תהיה מסוג "request", אשר מפורשת כבקשה של מכשיר A אל שרת B לשימוש בכתובת ה- IP שהציע שרת B למכשיר A בפאקטת ה- "Offer" .

- לסיום, שרת B מאשר למכשיר A להשתמש באותו IP על ידי פאקטה מסוג "Ack" .

נעבור אל הקודים שכתבנו בפייטון, ליצירת שרת ולקוח DHCP : (שימוש בספריית Scapy לצורך יצירת הפאקטות ושליחתם)

1. שרת DHCP : (name of the file : DHCP_SERVER)

Open DHCP_SERVER.py ~/Desktop/project/DHCP Save

DHCP_CLIENT.py | DHCP_SERVER.py

```python
from scapy.all import *


def dhcp_server(pkt):
# check if the packet protocol is DHCP
    if pkt[DHCP]:
        if pkt[DHCP].options[0][1] == 1: # options[0][1] = discover
            offer = Ether(dst='ff:ff:ff:ff:ff:ff')/IP(src='192.168.1.1', dst='192.168.1.100')/UDP(sport=67,dport=68)/BOOTP(op=2, yiaddr='192.168.1.100', siaddr='192.168.1.1', chaddr=pkt[Ether].src)/DHCP(options=[('message-type', 'offer'), ('subnet_mask', '255.255.255.0'), ('router', '192.168.1.1'), 'end'])
# send offer for the discover packet
            sendp(offer)
        elif pkt[DHCP].options[0][1] == 3: # oprions[0][3] = request
            ack = Ether(dst='ff:ff:ff:ff:ff:ff')/IP(src='192.168.1.1', dst='192.168.1.100')/UDP(sport=67,dport=68)/BOOTP(op=2, yiaddr='192.168.1.100', siaddr='192.168.1.1', chaddr=pkt[Ether].src)/DHCP(options=[('message-type', 'ack'), ('subnet_mask', '255.255.255.0'), ('router', '192.168.1.1'), 'end'])
# send ack for the request packet(server accept the request massage)
            sendp(ack)

# sniff packets udp with port 67 or 68
sniff(filter='udp and (port 67 or 68)', prn=dhcp_server)
```

יצרנו פונקציה הנקראת dhcp_server אשר תקבל פאקטה "pkt" (נסביר איך בהמשך) ותשלח את הפאקטה המתאימה לאותה פאקטה שהיא קיבלה.

לדוגמא :

אם השרת יקבל פאקטה מסוג "Discover" על השרת לשלוח פאקטה חדשה מסוג "Offer" כמו שהסברנו למעלה.

לאחר שהפונקציה מקבלת פאקטה שהיא מפרוטוקול DHCP (בקוד זה נראה כך : pkt[DHCP]) נבדוק איזה סוג הפאקטה, אם היא מסוג "Discover" או מסוג "Request".
בקוד זה יהיה רשום כך :

pkt[DHCP].options[0][1] = “Discover”

pkt[DHCP].options[0][2] = “Offer”

pkt[DHCP].options[0][3] = “Request”

pkt[DHCP].options[0][5] = “Ack”

עכשיו נסביר את תהליך בניית הפאקטה :

בספריית Scapy נוכל להשתמש בפרוטוקולים הבאים על מנת ליצור את הפאקטה שברצוננו לשלוח :

Ether - כתובת ה-MAC שאליה נרצה לשלוח את הפאקטה ("ff:ff:ff:ff:ff:ff" - לכל כתובת MAC)

IP - נגדיר את כתובת ה - IP של המקור ושל היעד

UDP - העברת המידע מתבצעת תחת Ports של פרוטוקול UDP.

BOOTP - תחת פרוטוקול זה נגדיר את סוג הפאקטה (בקשה - op=1 / תגובה - op=2) , נשלח את ה- IP שהשרת מציע ללקוח (yiaddr) ואת אותו ה- IP של אותו שרת(siaddr), ונגדיר ב - את היעד של שכבת הקו (chaddr = כתובת ה- MAC של הלקוח).

DHCP - נגדיר את סוג הבקשה ב - options : (‘message-type’, ’(discover/offer/request/ack)’)

- אם נקבל פאקטה מסוג "Discover" , נשלח פאקטת "Offer" :

Ether - כתובת ה-MAC שאליה נרצה לשלוח את הפאקטה ("ff:ff:ff:ff:ff:ff" - לכל כתובת MAC)

IP - מקור : IP של השרת, יעד : IP שהשרת מציע

UDP - העברת המידע מתבצעת תחת Ports (67/68)של פרוטוקול UDP.

BOOTP - siaddr : כתובת ה-IP של השרת, yiaddr : IP שהשרת מציע

DHCP - נגדיר את סוג הבקשה ב - options : ('message-type', 'offer')

נשלח את הפאקטה תחת הפונקציה sendp מספריית scapy.

- אם נקבל פאקטה מסוג "Request" , נשלח פאקטת "Ack" :

Ether - כתובת ה-MAC שאליה נרצה לשלוח את הפאקטה ("ff:ff:ff:ff:ff:ff" - לכל כתובת MAC)

IP - מקור : IP של השרת, יעד : IP שהשרת מציע

UDP - העברת המידע מתבצעת תחת Ports (67/68)של פרוטוקול UDP.

BOOTP - siaddr : כתובת ה-IP של השרת, yiaddr : IP שהשרת מציע

DHCP - נגדיר את סוג הבקשה ב - options : ('message-type', 'ack')

את קבלת הפאקטות נקבל מהפונקציה sniff מספריית scapy אשר מסניפה פאקטות מהתעבורה של הרשת,

הפונקציה תסניף רק פאקטות udp עם פורט : (67 או 68) , וכל פאקטה שהסנפנו תשלח אל הפונקציה dhcp_server.

2. לקוח DHCP : (name of the file : DHCP_CLIENT)

```python
from scapy.all import *
import sys


conf.checkIPaddr = False
fam, hw = get_if_raw_hwaddr(conf.iface)
# create a DHCP packet with type 'discover'(Brodcast)
discover = Ether(dst = 'ff:ff:ff:ff:ff:ff')/IP(src='0.0.0.0', dst='255.255.255.255')/UDP(sport=68, dport=67)/BOOTP(chaddr=hw)/DHCP(options=[('message-type', 'discover'), 'end'])
# send discover packet and define ansDIS be the answer with type 'offer'
ansDis = srp1(discover)
if True:
# create a DHCP packet with type 'request'
    request = Ether(dst = 'ff:ff:ff:ff:ff:ff')/IP(src=ansD[BOOTP].yiaddr, dst='255.255.255.255')/UDP(sport=68, dport=67)/BOOTP(chaddr=hw,yiaddr = ansD[BOOTP].yiaddr)/DHCP(options=[('message-type', 'request'), 'end'])
# send request packet
    ansReq = sendp(request)
```

תחילה ניצור פאקטה מסוג "Discover" חסרת כתובת IP (ip.src = 0.0.0.0) ונשלח את הפאקטה broadcast

(ip.dst = 255.255.255.255) נגדיר את כתובת הMAC להיות של הכרטיס רשת שלנו ונשלח את הפאקטה תחת שימוש בפונקציה : srp1 מספריית scapy אשר שולחת פאקטה בתור בקשה ומחכה לקבלת תשובה.

נגדיר את ansDis להיות התשובה לאותה פאקטה וניצור פאקטה חדשה מסוג "Request" , רק שהפעם ניקח את כתובת ה- IP מפאקטת ה- "Offer" שקיבלנו ככתובת המקור(ip.src = ansD[BOOTB].yiaddr).

ובפרוטוקול BOOTB נמלא בשדה yiaddr את yiaddr של ansDis.

לאחר שנסיים ליצור את פאקטת ה - "Request" נשלח אותה Broadcast תחת הפונקציה sendp שהסברנו עלייה מקודם.

לאחר שהסברנו את הקוד נראה בטרמינל הרצה של הקוד :

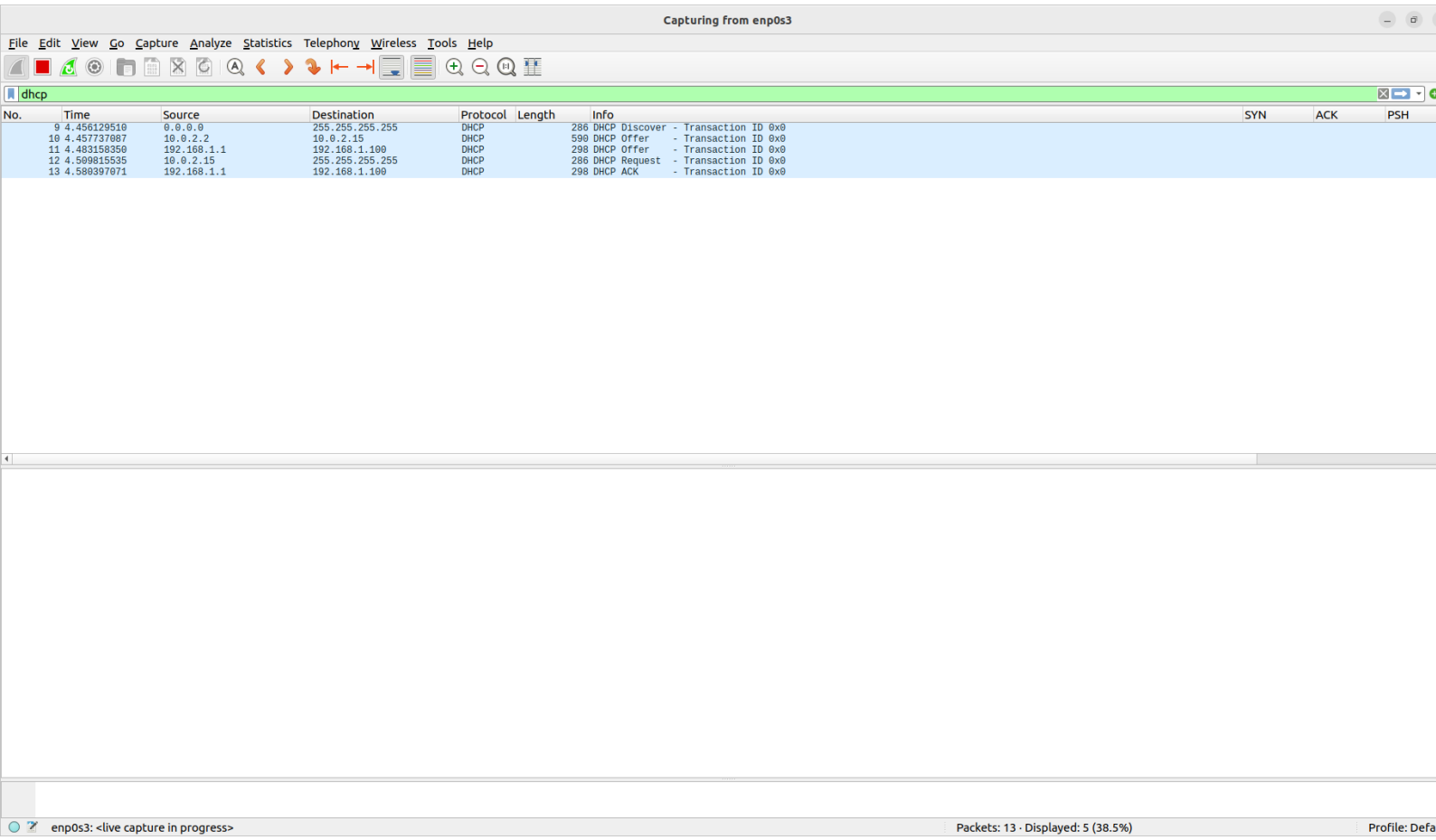
```
shlomi@2shlomi9:~/Desktop/project/DHCP$ sudo python3 DHCP_CLIENT.py
Begin emission:
Finished sending 1 packets.
*
Received 1 packets, got 1 answers, remaining 0 packets
.
Sent 1 packets.
shlomi@2shlomi9:~/Desktop/project/DHCP$
```

```
shlomi@2shlomi9:~/Desktop/project/DHCP$ sudo python3 DHCP_SERVER.py
.
Sent 1 packets.
.
Sent 1 packets.
```

בצד שמאל – הרצנו את הקוד של הלקוח ונוכל לראות שנשלחו 2 פאקטות, הראשונה נשלחה וקיבלה תשובה ולאחר מכן שלחה בקשה חדשה.

בצד ימין – הרצנו אל הקוד של השרת ונוכל לראות שנשלחו 2 פאקטות, אבל הפאקטות האלה יוכלו להשלח רק אחרי שהם הסניפו פאקטה מסוג "Discover" או מסוג "Request" כמו שהסברנו מקודם.

נעבור ל-Wireshark :



נראה שנשלחה בקשת "Discover" מהלקוח ולאחר מכן התקבלו הצעות "Offer" כמו שניתן לראות.

הלקוח קיבל כתובת IP ושלח בקשה חדשה לשימוש באותה כתובת ספציפית, ולאחר מכן נשלחה פאקטת ה– ACK שבה השרת מאשר את הבקשה.

DNS

DNS – Domain Name System :

DNS הינה מערכת אשר מתרגמת כתובת URL כמו :www.google.com , לכתובת- IP שהמחשב ידע לאן לפנות כמו : 127.0.0.1

למדנו במהלך הקורס שהכתובות הללו רשומות בתור URL כדי שלבן אדם יהיה יותר נוח לזכור את הכתובת (יותר קשה לזכור כתובות IP עם הרבה מספרים), והמחשב צריך כתובת IP כדי לדעת לאן לפנות.

אז איך עובד שרת ה-DNS?

כאשר לקוח מסויים רוצה לגשת לתוך אתר מסויים, הלקוח מקיש כתובת URL של האתר שהוא רוצה לגשת, ולאחר מכן הדפדפן שולח אל שרת ה-DNS בקשה לכתובת ה-IP של אותו אתר לפי אותה כתובת URL.

לאחר שהדפדפן מקבל את כתובת ה-IP של האתר, הדפדפן מעביר את אותה כתובת אל המכשיר של הלקוח, ואותו מכשיר מפנה אל האתר המבוקש.

נעבור אל הקודים שכתבנו בפייטון, ליצירת שרת ולקוח DNS : (שימוש בספריית Scapy לצורך יצירת הפאקטות ושליחתם)

1.DNS Server : (Name of the file : dns_server.py)

Open dns_server.py ~/Desktop/project/DNS Save

dns_client.py | dns_server.py

```python
from scapy.all import DNS, DNSQR, IP, sr1, UDP
from scapy.all import *
import sys


def dns_server(pkt):
# check if packet protocol is dns
    if pkt[DNS]:
# check if the packet send to the server(check the ip destination)
        if pkt[IP].dst== '127.0.0.1':
            print(f"Received message from {pkt[IP].src}")
# define qname to be the url
            qname = pkt[DNS].qd.qname.decode('utf-8')
            print("DNS query:", qname)
# define newIP to be the url IP address
            newIP = socket.gethostbyname(qname)
            print(f"The domain that needs to be converted to IP is:{newIP} ")
# send the packet with answer the ip of the url address
            sendp(Ether(dst = 'ff:ff:ff:ff:ff:ff')/IP(dst=pkt[IP].src)/UDP(dport=53)/DNS(rd=1,qd=DNSQR(qname=f"{newIP}"), an=DNSRR(rrname= qname, rdata= f"{newIP}")),verbose=0)


# sniff packet with protocol udp and port 53
sniff(filter='udp and (port 53)', prn=dns_server,count =1)
```

*הסברנו בעמודים הקודמים (היכן שהסברנו מהו DHCP) על יצירת פאקטות ושליחתם תחת ספריית Scapy ובנוסף הסברנו את הפונקציות הבאות : srp1, sendp, sniff .

השתמשנו באותה שיטה של הסנפת פאקטות כמו שעשינו ב- DHCP Server, שליחת הפאקטות גם הם בוצעו באותם פונקציות שבהם השתמשנו ב- DHCP Server.

בנוסף הסברנו על הפרוטוקולים : Ether, IP, UDP .

מכאן נסביר על הפרוטוקול DNS :

בפרוטוקול DNS אנו נגדיר שאילתה ותשובה לאותה שאילתה (שאילתה : כתובת IP עבור כתובת URL מסויימת, תשובה : כתובת ה- IP המבוקשת).

-שימוש בשדה qd=DNSQR(qname) בפרוטוקול DNS הינו משמש ליצירת שאילתה,

-שימוש בשדה an=DNSRR(rrname=X, rdata=Y) בפרוטוקול DNS הינו משמש ליצירת תשובה לשאילתה, כך ש- X מייצג את כתובת ה- URL שאלייה הלקוח רוצה לגשת, ו- Y מייצג את התשובה לשאילתה שזאת כתובת ה- IP של אותה כתובת URL המבוקשת.

מכאן נסביר איך עובד הקוד :

כמו שהסברנו הפונקציה dns_server מקבלת פאקטות מהפונקציה sniff ומסננת את הפאקטות בכך שהיא מקבלת רק את הפאקטות שהיעד שלהם הוא : 127.0.0.1, שזאת בעצם הכתובת IP שהגדרנו עבור שרת ה- DNS שלנו, ולפי פרוטוקול DNS.

לאחר שהפונקציה קיבלה פאקטה מפרוטוקול DNS עם יעד 127.0.0.1 היא מפרשת את השאילתה (מקידוד utf-8) לכתובת URL.

נוכל לראות זאת בקוד : qname = pkt[DNS].qd.qname.decode(utf-8)

נגדיר משתנה "newIP" להיות כתובת ה- IP של qname (כתובת הURL המבוקשת), ונעשה זאת באמצעות הפונקציה gethostbyname אשר נמצאת בספריית socket.

נוכל לראות בקוד: newIP = socket.gethostbyname(qname)

לאחר מכן נגדיר rdata להיות newIP (כתובת ה- IP המבוקשת) ולאחר מכן נגדיר את rrname להיות qname (כתובת ה – URL מהשאילתה).

ולבסוף נשלח את הפאקטה.

2.DNS Client : (Name of the file : dns_client.py)

```
from scapy.all import DNS, DNSQR, IP, sr1, UDP
from scapy.all import *
import sys


def dns_request():
#create a DNS packet with query for the ip of "www.google.com"
    request = Ether(dst = 'ff:ff:ff:ff:ff:ff')/IP(src = '0.0.0.0',dst = '127.0.0.1')/UDP(dport=53)/DNS(rd=1,qd=DNSQR(qname='www.google.com'))
#send request
    sendp(request)


dns_request()
```

ניצור פאקטה ששולחת בקשה אל כתובת ה- IP : "127.0.0.1" את השאילתה(הסברנו למעלה איך מגדירים את השדות)

לצורך הדוגמא בחרנו את השאילתה להיות הבקשה לכתובת URL של גוגל : www.google.com.

לאחר מכן שלחנו את השאילתה אל ה- DNS Server.

- נראה הרצה של שרת והלקוח DNS בטרמינל :

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

shlomi@2shlomi9:~/Desktop/project/DNS$ sudo python3 dns_client.py
.
Sent 1 packets.
shlomi@2shlomi9:~/Desktop/project/DNS$

shlomi@2shlomi9:~/Desktop/project/DNS$ sudo python3 dns_server.py
Received message from 0.0.0.0
DNS query: www.google.com.
The domain that needs to be converted to IP is:172.217.22.100
shlomi@2shlomi9:~/Desktop/project/DNS$
```

מצד שמאל – הלקוח שולח פאקטת עם שאילתה לשרת הDNS .

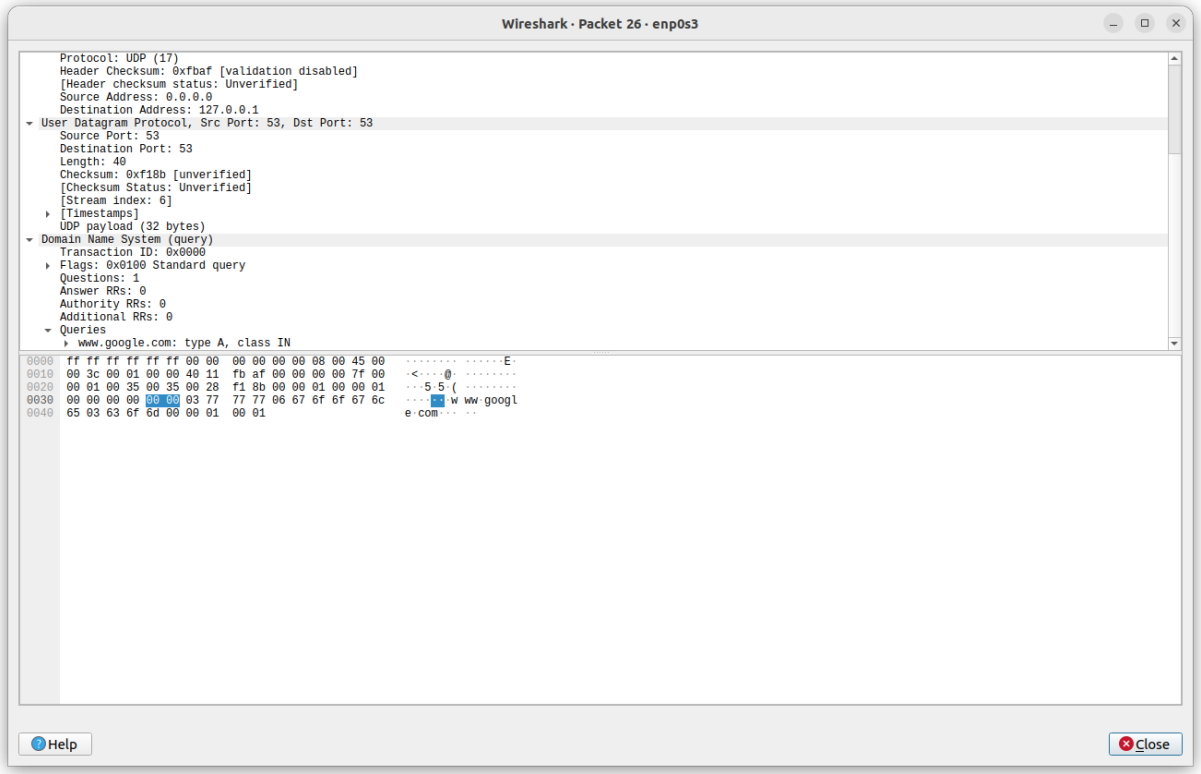
מצד ימין – השרת מקבל את הבקשה מכתובת הIP של הלקוח, ממיר את השאילתה לכתובת IP, ושולח את אותה כתובת IP בחזרה כתשובה.

-לאחר שהרצנו את הלקוח והשרת DNS בטרמינל, נפתח את ווירשארק :

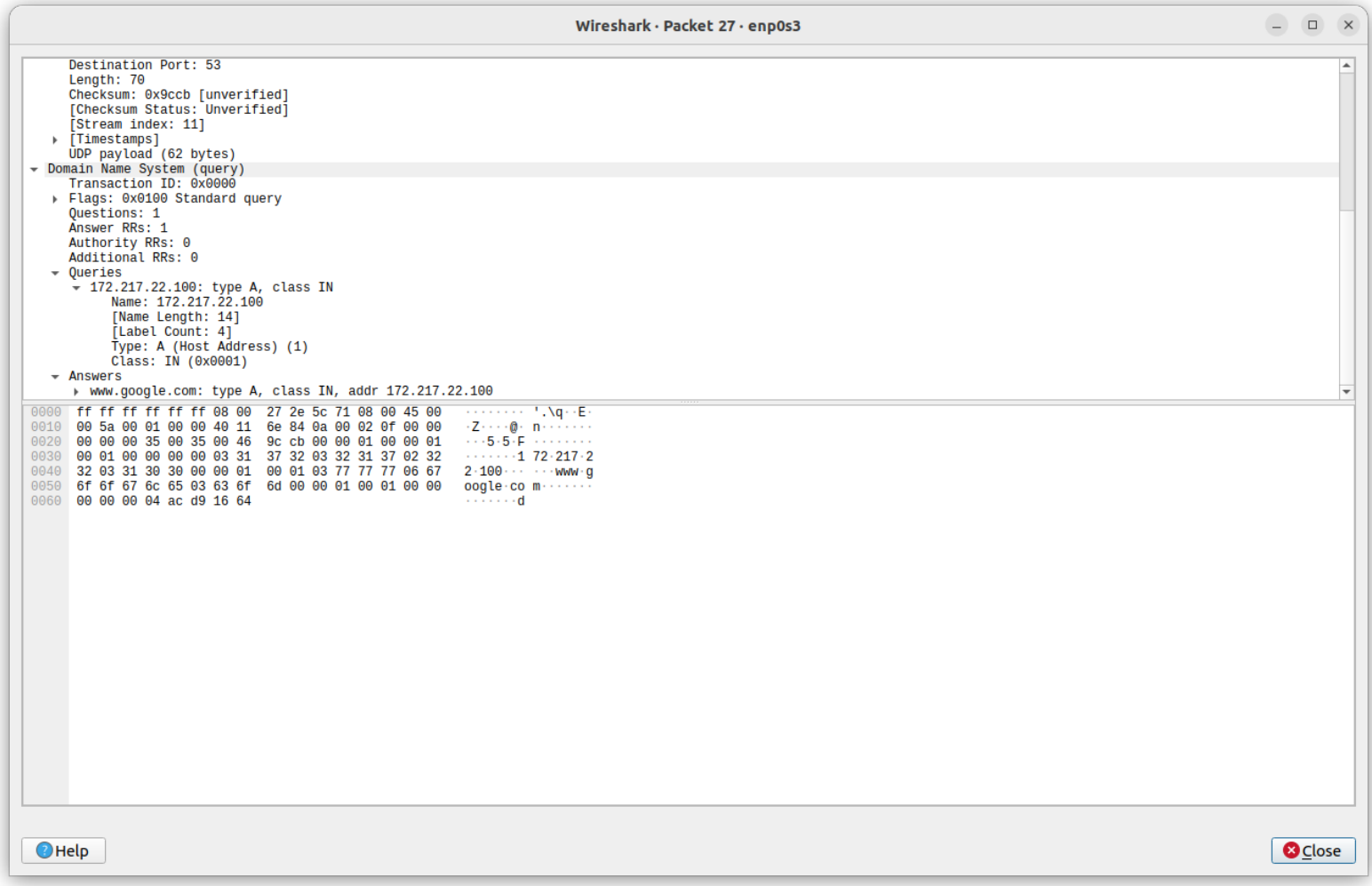
18	13.520909780	10.100.102.1	10.0.2.15	DNS	73 Standard query response 0xb7f7 No such name AAAA 2shlomi9.home
19	13.520910112	10.100.102.1	10.0.2.15	DNS	73 Standard query response 0x30c5 No such name A 2shlomi9.home
20	15.595269461	10.0.2.15	10.100.102.1	DNS	73 Standard query 0x20c8 A 2shlomi9.home
21	15.595688026	10.0.2.15	10.100.102.1	DNS	73 Standard query 0x392c AAAA 2shlomi9.home
22	15.601539842	10.100.102.1	10.0.2.15	DNS	73 Standard query response 0x392c No such name AAAA 2shlomi9.home
23	15.601540378	10.100.102.1	10.0.2.15	DNS	73 Standard query response 0x20c8 No such name A 2shlomi9.home
26	16.987013577	0.0.0.0	127.0.0.1	DNS	74 Standard query 0x0000 A www.google.com
27	17.028151072	10.0.2.15	0.0.0.0	DNS	104 Standard query 0x0000 A 172.217.22.100 A 172.217.22.100

נסתכל על הפאקטות המודגשות בכחול, הפאקטה הראשונה אכן מכילה שאילתה לכניסה לכתובת URL של גוגל עם הכתובת יעד והמקור שהגדרנו, ולאחר מכן קיבלנו תשובה לאותה שאילתה משרת הDNS שיצרנו.

נכנס לכל אחת מהפאקטות:



*זאת הפאקטה של הלקוח, נוכל לראות בתוך שכבת הפרוטוקול Domain Name System(query) שרק הדגל של questions דלוק, ולכן זאת שאילתה.



* זאת הפאקטה של השרת, נוכל לראות בתוך שכבת הפרוטוקול Domain Name System(query) שהדגל של questions דלוק, והדגל של Answer RRs גם הוא דלוק.

נכנס לתוך Queries ונראה את התשובה של השרת, בנוסף נכנס ל- Answers ונוכל לראות את התשובה של השרת עבור הבקשה לכתובת הIP של אתר www.google.com, שהתשובה הינה: 172.217.22.100.

ביבילגרפיה

https://www.sciencedirect.com/science/article/pii/S2405452620305784

https://www.omnisecu.com/cisco-certified-network-associate-ccna/difference-between-bgp-and-ospf-protocols.php

https://blog.cloudflare.com/the-road-to-quic/.

https://www.youtube.com/watch?v=npM9RZH9cLk

https://docs.oracle.com/cd/E19253-01/816-4554/dhcp-overview-3/index.html