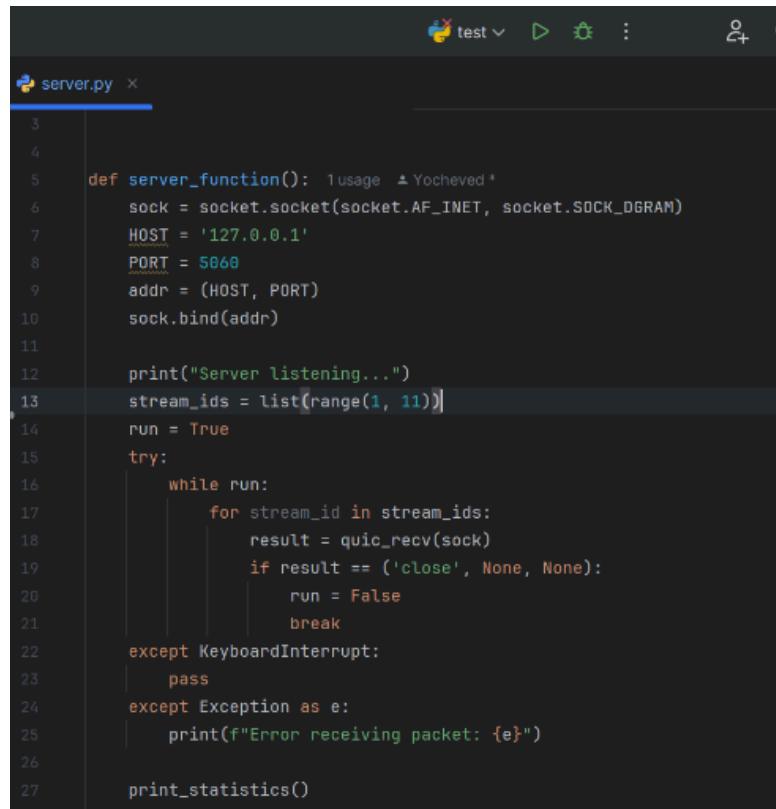


חלק רטוֹב

או נשותח בקוד ב**4V7P**

הסביר על מעבר מידע בין הלקוח לשרת:
שרות:

הקוד מגדיר שרת שמאזין לחבילות נתונים. הוא מקבל נתונים, ואם מתאפשר אות סגירה, הוא מפסיק לפעול
ומძפיא סטטיסטיקות.

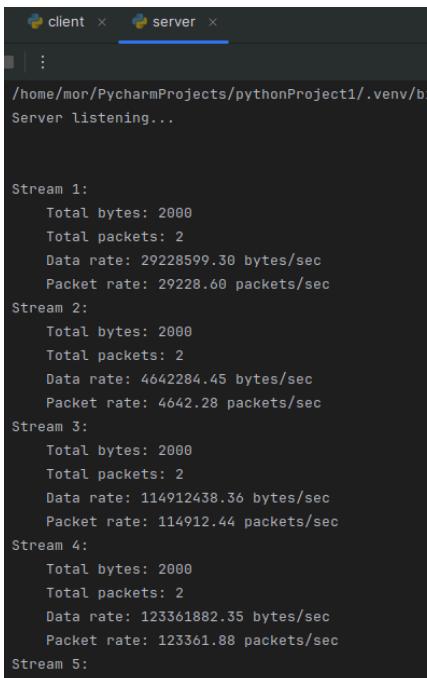


```
server.py
```

```
3
4
5     def server_function():  usage ▲ Yocheved
6         sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
7         HOST = '127.0.0.1'
8         PORT = 5060
9         addr = (HOST, PORT)
10        sock.bind(addr)
11
12        print("Server listening...")
13        stream_ids = list(range(1, 11))
14        run = True
15        try:
16            while run:
17                for stream_id in stream_ids:
18                    result = quic_recv(sock)
19                    if result == ('close', None, None):
20                        run = False
21                        break
22                except KeyboardInterrupt:
23                    pass
24                except Exception as e:
25                    print(f"Error receiving packet: {e}")
26
27        print_statistics()
```

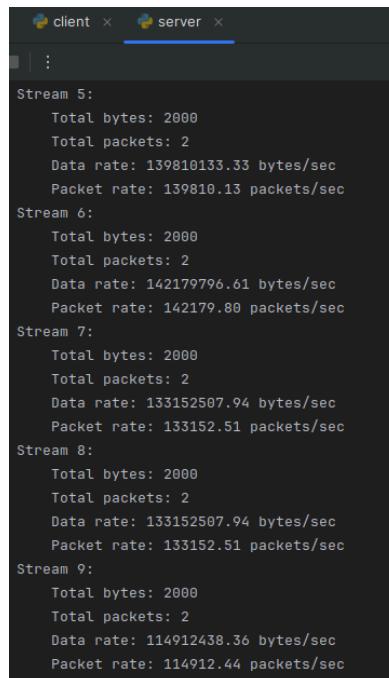
פלט של השירות:

לכל חבילה שמגיעה לשרת נוון מידע על כל חלק, ובסוף ממחזר את הסטטיסטיקה הכלולת על כל החלקים.



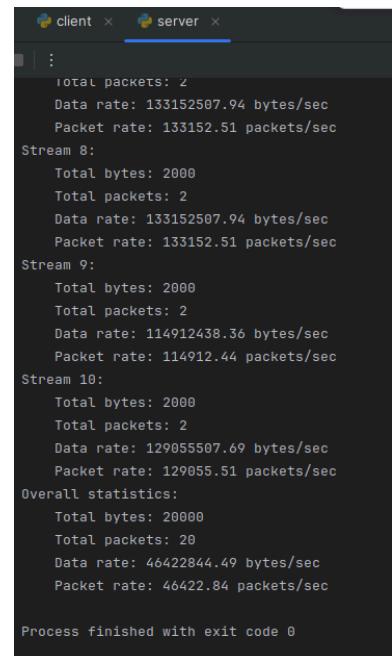
```
client x server x
```

```
/home/mor/PycharmProjects/pythonProject1/.venv/bin
Server listening...
Stream 1:
Total bytes: 2000
Total packets: 2
Data rate: 29228599.30 bytes/sec
Packet rate: 29228.60 packets/sec
Stream 2:
Total bytes: 2000
Total packets: 2
Data rate: 4642284.45 bytes/sec
Packet rate: 4642.28 packets/sec
Stream 3:
Total bytes: 2000
Total packets: 2
Data rate: 114912438.36 bytes/sec
Packet rate: 114912.44 packets/sec
Stream 4:
Total bytes: 2000
Total packets: 2
Data rate: 123361882.35 bytes/sec
Packet rate: 123361.88 packets/sec
Stream 5:
Total bytes: 2000
Total packets: 2
Data rate: 114912438.36 bytes/sec
Packet rate: 114912.44 packets/sec
```



```
client x server x
```

```
Stream 5:
Total bytes: 2000
Total packets: 2
Data rate: 139810133.33 bytes/sec
Packet rate: 139810.13 packets/sec
Stream 6:
Total bytes: 2000
Total packets: 2
Data rate: 142179796.61 bytes/sec
Packet rate: 142179.80 packets/sec
Stream 7:
Total bytes: 2000
Total packets: 2
Data rate: 133152507.94 bytes/sec
Packet rate: 133152.51 packets/sec
Stream 8:
Total bytes: 2000
Total packets: 2
Data rate: 133152507.94 bytes/sec
Packet rate: 133152.51 packets/sec
Stream 9:
Total bytes: 2000
Total packets: 2
Data rate: 114912438.36 bytes/sec
Packet rate: 114912.44 packets/sec
Stream 10:
Total bytes: 2000
Total packets: 2
Data rate: 129055507.69 bytes/sec
Packet rate: 129055.51 packets/sec
Overall statistics:
Total bytes: 20000
Total packets: 20
Data rate: 46422844.49 bytes/sec
Packet rate: 46422.84 packets/sec
Process finished with exit code 0
```



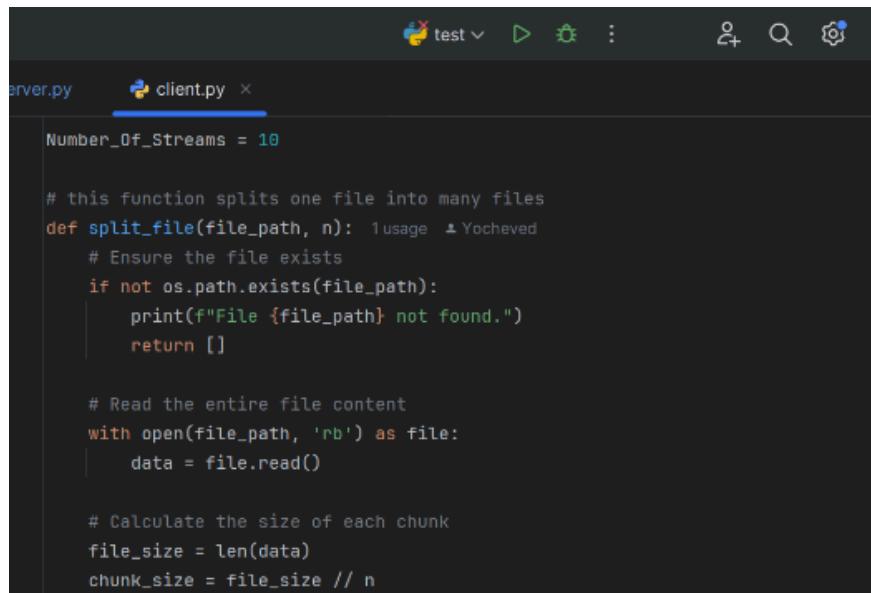
```
client x server x
```

```
:
```

```
total packets: 2
Data rate: 133152507.94 bytes/sec
Packet rate: 133152.51 packets/sec
Stream 8:
Total bytes: 2000
Total packets: 2
Data rate: 133152507.94 bytes/sec
Packet rate: 133152.51 packets/sec
Stream 9:
Total bytes: 2000
Total packets: 2
Data rate: 114912438.36 bytes/sec
Packet rate: 114912.44 packets/sec
Stream 10:
Total bytes: 2000
Total packets: 2
Data rate: 129055507.69 bytes/sec
Packet rate: 129055.51 packets/sec
Overall statistics:
Total bytes: 20000
Total packets: 20
Data rate: 46422844.49 bytes/sec
Packet rate: 46422.84 packets/sec
Process finished with exit code 0
```

לקיים:

הfonקציה `split_file` מחלקת קובץ נתון למספר קבצים קטנים יותר. כל קובץ פלט מכיל חלק מהנתונים של הקובץ המקורי. הfonקציה מקבלת שני פרמטרים:
`file_path`: נתיב הקובץ המקורי שברצונך לחלק.
ח: מספר החלקים שברצונך לחלק את הקובץ אליו.
הfonקציה תחילת בודקת אם הקובץ קיים בנתיב שניתן. אם הקובץ לא קיים, היא מדפסה הודעה מתאימה ומחזירה רשימה ריקה. (הקובץ נפתח במצב ביןארי ('rb'), ותוכן הקובץ נקרא ומואחסן במשתנה `data`).
הfonקציה מחשבת את גודל הקובץ הכלול (בBITS) ומחשבת את גודל כל חלק (`chunk`) על ידי חלוקת גודל הקובץ במספר החלקים ח.



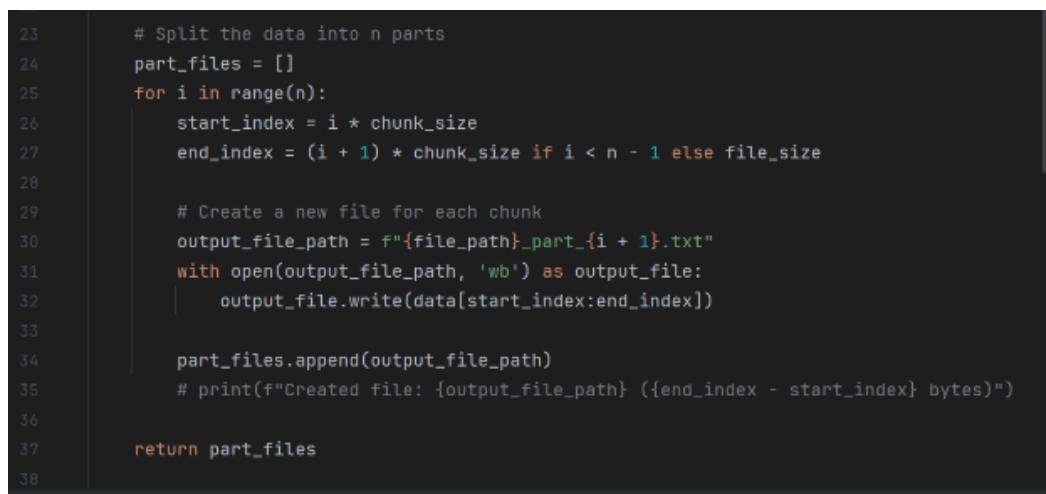
```
Number_Of_Streams = 10

# this function splits one file into many files
def split_file(file_path, n): 1usage ʌ Yocheved
    # Ensure the file exists
    if not os.path.exists(file_path):
        print(f"File {file_path} not found.")
        return []

    # Read the entire file content
    with open(file_path, 'rb') as file:
        data = file.read()

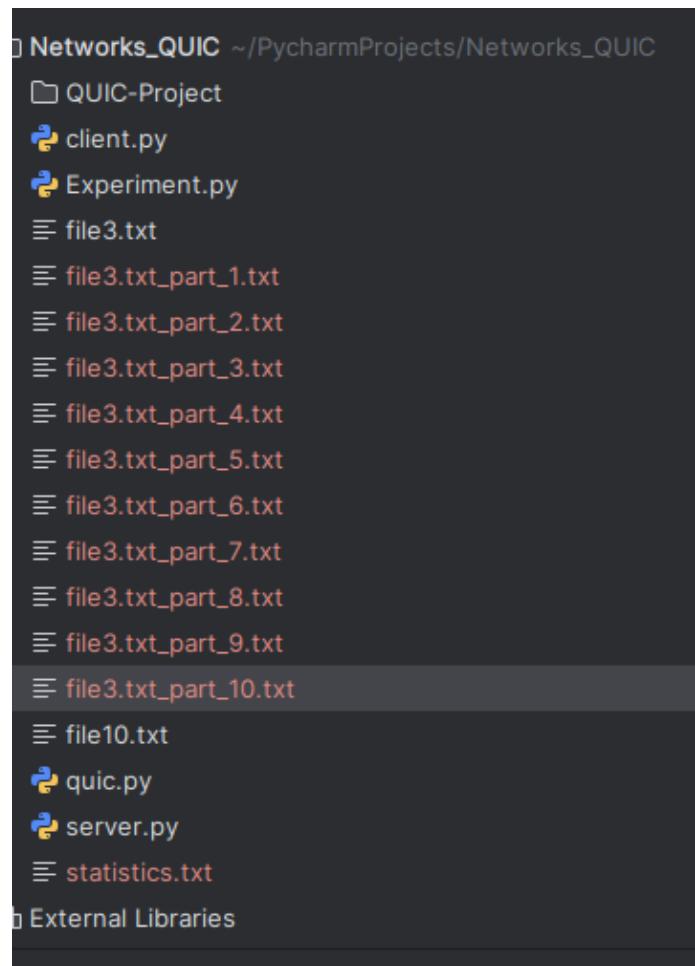
    # Calculate the size of each chunk
    file_size = len(data)
    chunk_size = file_size // n
```

בלויאת FOR הfonקציה עוברת ח פעמים על החלקים ומחשבת את ההתחלה והוסף של כל חלק (החלק האחרון עשוי להיות גדול יותר אם גודל הקובץ לא מתחלק בDIVOK ב-ח).
 לכל חלק נוצר קובץ חדש עם נתיב שמכיל את שם הקובץ המקורי וסיימת המספר של החלק (file_path_part_X.txt), ותוכן החלק נכתב לתוך הקובץ החדש.



```
23     # Split the data into n parts
24     part_files = []
25     for i in range(n):
26         start_index = i * chunk_size
27         end_index = (i + 1) * chunk_size if i < n - 1 else file_size
28
29         # Create a new file for each chunk
30         output_file_path = f"{file_path}_part_{i + 1}.txt"
31         with open(output_file_path, 'wb') as output_file:
32             output_file.write(data[start_index:end_index])
33
34         part_files.append(output_file_path)
35         # print(f"Created file: {output_file_path} ({end_index - start_index} bytes)")
36
37     return part_files
38
```

בסוף הfonקציה מוחזירה את רשימת נתיבי הקבצים שנוצרו.

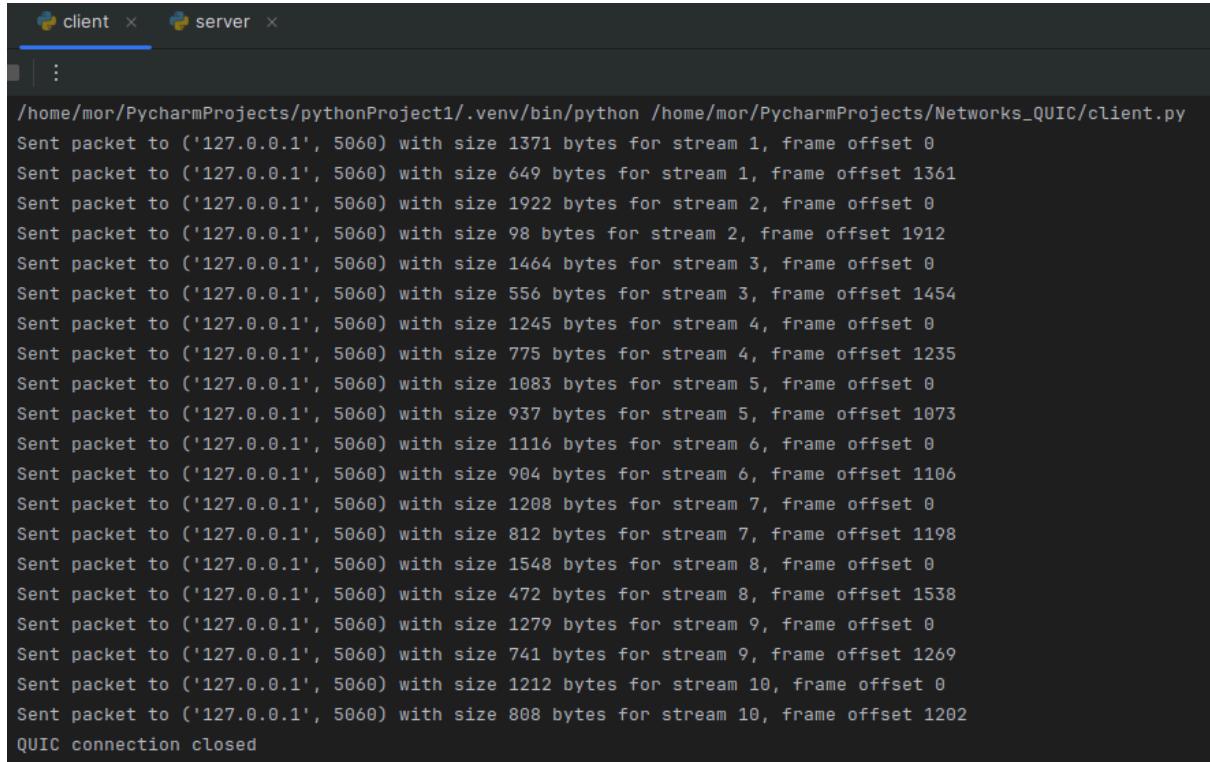


בפונקציית הלוקוח, הלוקוח פותח סוקט, קורא לפונקציה שמחלקת את הקובץ, ושולח כל חלק אל השרת.

```
server.py client.py ×
37
38
39
40 def client_function(): 1 usage ▲ Yocheved
41     HOST = '127.0.0.1'
42     PORT = 5060
43     sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
44     destination = (HOST, PORT)
45
46     # Split the file into parts for each stream
47     file_paths = split_file(file_path='file3.txt', Number_of_Streams)
48
49     for i in range(Number_of_Streams):
50         file_path = file_paths[i]
51         with open(file_path, 'rb') as file:
52             data = file.read()
53             quic_send(sock, destination, data, i+1)
54
55         quic_close(sock, destination)
56
57
58 if __name__ == '__main__':
59     client_function()
```

פלט לפקוח:

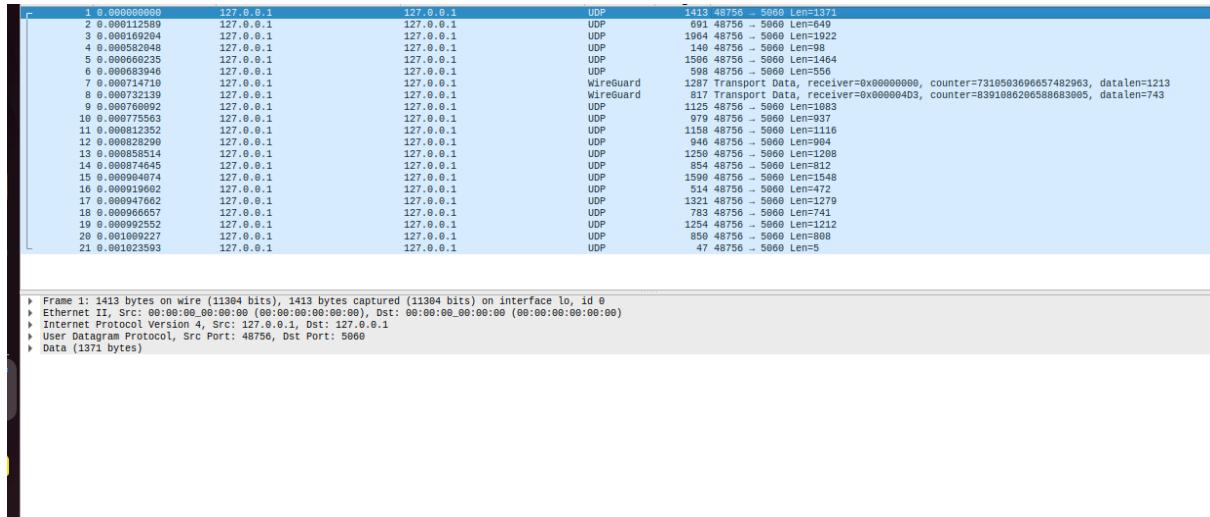
כל חבילה הנשלחת מפורטת בפקוח, ניתן לראות את גודל החבילה (LEN) וכן גם לראות תיעוד של המעבר שלו בהקלטה WIRESHARK



```
/home/mor/PycharmProjects/pythonProject1/.venv/bin/python /home/mor/PycharmProjects/Networks_QUIC/client.py
Sent packet to ('127.0.0.1', 5060) with size 1371 bytes for stream 1, frame offset 0
Sent packet to ('127.0.0.1', 5060) with size 649 bytes for stream 1, frame offset 1361
Sent packet to ('127.0.0.1', 5060) with size 1922 bytes for stream 2, frame offset 0
Sent packet to ('127.0.0.1', 5060) with size 98 bytes for stream 2, frame offset 1912
Sent packet to ('127.0.0.1', 5060) with size 1464 bytes for stream 3, frame offset 0
Sent packet to ('127.0.0.1', 5060) with size 556 bytes for stream 3, frame offset 1454
Sent packet to ('127.0.0.1', 5060) with size 1245 bytes for stream 4, frame offset 0
Sent packet to ('127.0.0.1', 5060) with size 775 bytes for stream 4, frame offset 1235
Sent packet to ('127.0.0.1', 5060) with size 1083 bytes for stream 5, frame offset 0
Sent packet to ('127.0.0.1', 5060) with size 937 bytes for stream 5, frame offset 1073
Sent packet to ('127.0.0.1', 5060) with size 1116 bytes for stream 6, frame offset 0
Sent packet to ('127.0.0.1', 5060) with size 904 bytes for stream 6, frame offset 1106
Sent packet to ('127.0.0.1', 5060) with size 1208 bytes for stream 7, frame offset 0
Sent packet to ('127.0.0.1', 5060) with size 812 bytes for stream 7, frame offset 1198
Sent packet to ('127.0.0.1', 5060) with size 1548 bytes for stream 8, frame offset 0
Sent packet to ('127.0.0.1', 5060) with size 472 bytes for stream 8, frame offset 1538
Sent packet to ('127.0.0.1', 5060) with size 1279 bytes for stream 9, frame offset 0
Sent packet to ('127.0.0.1', 5060) with size 741 bytes for stream 9, frame offset 1269
Sent packet to ('127.0.0.1', 5060) with size 1212 bytes for stream 10, frame offset 0
Sent packet to ('127.0.0.1', 5060) with size 808 bytes for stream 10, frame offset 1202
QUIC connection closed
```

בהקלטה WIRESHARK ניתן לראות שהמגיד עבר בקשר UDP, אפשר לראות את אורך החבילה בהתאם לפולט של הפקוח) החבילה הראשונה עברה בגודל של 1371 ביטים, שכן, ניתן לראות בהקלטה שזה האורך של החבילה בשורה הראשונה (LEN=1371)).

החבילה עברה לכתובות SOCKET, 127.0.0.1, שזו הכתובת שהגדנו בהקלטה



Frame	Source	Destination	Protocol	Length	Time
1 0.000000000	127.0.0.1	127.0.0.1	UDP	1413	48756 - 5060 Len=1371
2 0.000112589	127.0.0.1	127.0.0.1	UDP	601	48756 - 5060 Len=649
3 0.000169264	127.0.0.1	127.0.0.1	UDP	1064	48756 - 5060 Len=1922
4 0.000582948	127.0.0.1	127.0.0.1	UDP	148	48756 - 5060 Len=98
5 0.000889325	127.0.0.1	127.0.0.1	UDP	1508	48756 - 5060 Len=1464
6 0.000905016	127.0.0.1	127.0.0.1	UDP	509	48756 - 5060 Len=1083
7 0.000974718	127.0.0.1	127.0.0.1	WireGuard	1287	Transport Data, receiver=0x00000000, counter=7310503696657482963, datalen=1213
8 0.0009732139	127.0.0.1	127.0.0.1	WireGuard	817	Transport Data, receiver=0x000000403, counter=8391886206588683005, datalen=743
9 0.0008760992	127.0.0.1	127.0.0.1	UDP	1125	48756 - 5060 Len=1803
10 0.000877556	127.0.0.1	127.0.0.1	UDP	979	48756 - 5060 Len=937
11 0.000882352	127.0.0.1	127.0.0.1	UDP	1158	48756 - 5060 Len=116
12 0.000882600	127.0.0.1	127.0.0.1	UDP	946	48756 - 5060 Len=904
13 0.0008858534	127.0.0.1	127.0.0.1	UDP	1250	48756 - 5060 Len=1208
14 0.000974645	127.0.0.1	127.0.0.1	UDP	854	48756 - 5060 Len=812
15 0.000904974	127.0.0.1	127.0.0.1	UDP	1500	48756 - 5060 Len=1548
16 0.000919662	127.0.0.1	127.0.0.1	UDP	514	48756 - 5060 Len=472
17 0.000977602	127.0.0.1	127.0.0.1	UDP	1321	48756 - 5060 Len=1779
18 0.00090677	127.0.0.1	127.0.0.1	UDP	783	48756 - 5060 Len=741
19 0.000992552	127.0.0.1	127.0.0.1	UDP	1254	48756 - 5060 Len=1212
20 0.001009227	127.0.0.1	127.0.0.1	UDP	859	48756 - 5060 Len=868
21 0.001023593	127.0.0.1	127.0.0.1	UDP	47	48756 - 5060 Len=5

Frame 1: 1413 bytes on wire (11304 bits), 1413 bytes captured (11304 bits) on interface lo, id 0
Ethernet II, Src: 00:00:00:00:00:00 (00:00:00:00:00:00), Dst: 127.0.0.1 (127.0.0.1)
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
User Datagram Protocol, Src Port: 48756, Dst Port: 5060
Data (1371 bytes)

:QUIC

בחלק התיאורתי הרחכנו על פרוטוקול QUIC, פרוטוקול המודיע למעבר מידע ברשות. במתלה מימושו את ריבוי הזרימות.

Threads הם ייחודת ביצוע בתוכנית, אשר מאפשרות לשלווח מספר חבילות בו-זמנית ובכך מייעלות את הביצועים ומתקשרות את זמן המתנה. נדרש ניהול נכון של קוווי השילוח על מנת למנוע התנגשויות או בעיות גישה בין ייחודת הקצה.

```
1  > import ...
5
7  # Global dictionaries for packet sizes and statistics
8  stream_packet_sizes = {}
9  stream_statistics = {}
10 data_rates = []
11 packet_rates = []
12 num_flows_list = []
13
14
15 def set_stream_packet_size(stream_id):  2 usages  ▲ Yocheved
16     if stream_id not in stream_packet_sizes:
17         packet_size = random.randint(1000, 2000)
18         stream_packet_sizes[stream_id] = packet_size
19         stream_statistics[stream_id] = {'bytes': 0, 'packets': 0, 'start_time': None, 'end_time': None}
20     return stream_packet_sizes[stream_id]
21
```

פונקציה זו קובעת את גודל החבילה עבור זרם נתון וכותבת את הסטטיסטיות שלו. הפונקציהבודקת אם מזהה הזרם כבר קיים ב-stream_packet_sizes (מכיל את גודל החבילות לכל זרם). אם הזרם לא קיים, הפונקציה תגריל מספר אקראי בין 1000 ל-2000 אשר ישמש כגודל החבילה לזרם זה. בנוסף, היא מגדרה ערכי התחלת עבור הסטטיסטיות של הזרם בתוך stream_statistics, כמו מספר הביטים שנשלחו, מספר החבילות, זמן התחלת וזמן סיום.

```
22
23 def create_quic_packet(stream_id, sequence_number, data):  1 usage  ▲ Yocheved
24     packet_size = set_stream_packet_size(stream_id)
25     data = data[:packet_size]  # Ensure data is within packet size
26     packet_format = f'I I H {len(data)}s'
27     packet = struct.pack(packet_format, stream_id, sequence_number, len(data), data)
28     return packet
```

הfonקציה create_quic_packet יוצרת חבילה נתונים (packet) עבור פרוטוקול QUIC. החבילה מכילה מידע על הזרם, מספר הסידור של החבילה, ואת הנתונים עצמם, כשהם מוגבלים לגודל המוגדר מראש עבור הזרם.

```
30
31     def parse_quic_packet(packet):  1 usage  ▲ Yocheved
32         header_format = 'I I H'
33         header_size = struct.calcsize(header_format)
34         try:
35             stream_id, frame_offset, payload_length = struct.unpack(header_format, packet[:header_size])
36             data_format = f'{payload_length}s'
37             data = struct.unpack(data_format, packet[header_size:header_size + payload_length])
38             return stream_id, frame_offset, data
39         except struct.error as e:
40             print(f"Error parsing packet: {e}")
41         return None, None, None
42
```

הfonקציה parse_quic_packet מנתחת חבילה QUIC ומפענחת את המידע שהתקבל, כולל מזהה הזרם, מספר הסידור של החבילה, והנתונים שנשלחו.

הfonקציה משתמשת ב-`struct.unpack` כדי לפענח את הcontent של החבילה מהבתים הראשונים של הנתונים, על פי הפורמט שהוגדר. זה מחלץ את המידע על מזהה הזרם, מספר הסידור, ואת אורך הנתונים.

```
def quic_send(sock, destination, data, stream_id):  4 usages  ▲ Yocheved*
    frame_offset = 0
    packet_size = set_stream_packet_size(stream_id)
    threads = []

    while frame_offset < len(data):
        data_chunk = data[frame_offset:frame_offset + packet_size]
        thread = threading.Thread(target=send_packet, args=(sock, destination, stream_id, frame_offset, data_chunk))
        threads.append(thread)
        thread.start()
        frame_offset += packet_size

    # Wait for all threads to finish
    for thread in threads:
        thread.join()

    print(f"Finished sending packets to {destination} for stream {stream_id}")
```

הfonקציה `quic_send` אחראית לשילוח נתונים בצורה פרגמנטרית לכתובת יעד בראשת, תוך שימוש ב-`threading` ליצירת פעולות מקבילות. היא מחלקת את הנתונים לחunks (chunks) בגודל קבוע, שולחת כל חלק, וمعدכנת את הסטטיסטיקות להזרם הנתונים. בסיום, הfonקציה ממתינה לסיום כל הקווים שליחת לפניה שהיא מודיעה על סיום שליחת הנתונים.

כל חזרה בולולאה חותכת חלק מהנתונים (packet_data) בגודל `packet_size` (packet_size).
ויצרת חבילה מהנתונים (packet), ושולחת אותה לכתובת יעד (sock.sendto).
معدכנת סטטיסטיקות של הזרם (bytes, packets, end_time).
יצרת ומתחילה קוו שליחת חדש כדי לשלוח את החבילה (thread.start).

```
def quic_recv(sock):  2 usages  ▲ Yocheved
    packet, _ = sock.recvfrom(2048)
    if packet == b"close":
        sock.close()
        # print("Received close packet")
        return "close", None, None

    stream_id, frame_offset, data = parse_quic_packet(packet)
    if stream_id is None:
        print("Received invalid packet: %s" % packet)
        return

    if stream_id not in stream_statistics:
        stream_statistics[stream_id] = {'bytes': 0, 'packets': 0, 'start_time': None, 'end_time': None}

    if stream_statistics[stream_id]['start_time'] is None:
        stream_statistics[stream_id]['start_time'] = time.time()
    stream_statistics[stream_id]['bytes'] += len(data)
    stream_statistics[stream_id]['packets'] += 1
    stream_statistics[stream_id]['end_time'] = time.time()
    # print("Received packet from stream %s, frame offset %s" % (stream_id, frame_offset))

    return stream_id, frame_offset, data
```

הfonקציה `quic_recv` אחראית לקבל חבילות נתונים בפרוטוקול QUIC דרך סוקט (socket), לפענח אותן, ולעדכן את הסטטיסטיקות של הזרם שמננו התקבלו החבילות. הfonקציה מטפלת גם במקרה שבו מקבלת חבילת "close" שגורמת לסגירת החיבור.

הfonקציה `parse_quic_packet` כדי לפענח את החבילה ולהוציא את מזהה הזרם (id), מספר הסידור (stream_id), (frame_offset), ואת הנתונים עצם (data). אם הפענוח נכשל,

יהי `None`, והפונקציה תדפס הودעת שגיאה וטסימ ללא החזרת ערכיהם. אם מדובר בזרם חדש שטרם התקבלו ממנה חבילות, הפונקציה מוסיפה ערך חדש למלון `stream_statistics` מספר הבעיות, מספר החבילות, זמן התחלת וזמן סיום.

```
def quic_close(sock, destination): 3 usages  ↳ Yocheved
    try:
        sock.sendto(b"close", destination)
        sock.close()
        print("QUIC connection closed")
    except socket.error as e:
        print(f"Socket error: {e}")
```

הפונקציה שולחת הודעת סגירה ("מחוזת close" בפורמט ביןארי) לעד שנמסר, דרך הסקוט. הודעת הסגירה מסמנת לכך שהחיבור עומד להיסגר. לאחר מכן, סוגרת את הסקוט.

```
def print_statistics(): 2 usages  ↳ Yocheved
    with open('statistics.txt', 'w') as file:
        for stream_id, stats in stream_statistics.items():
            total_bytes = stats['bytes']
            total_packets = stats['packets']
            start_time = stats['start_time']
            end_time = stats['end_time']
            duration = end_time - start_time if end_time and start_time else 0
            data_rate = total_bytes / duration if duration > 0 else 0
            packet_rate = total_packets / duration if duration > 0 else 0
            file.write(f"Stream {stream_id}:\n")
            file.write(f"\tTotal bytes: {total_bytes}\n")
            file.write(f"\tTotal packets: {total_packets}\n")
            file.write(f"\tData rate: {data_rate:.2f} bytes/sec\n")
            file.write(f"\tPacket rate: {packet_rate:.2f} packets/sec\n")

            total_bytes = sum(stats['bytes'] for stats in stream_statistics.values())
            total_packets = sum(stats['packets'] for stats in stream_statistics.values())
            total_duration = max((stats['end_time'] - stats['start_time']) for stats in stream_statistics.values() if stats['end_time'] and stats['start_time'])
            total_data_rate = total_bytes / total_duration if total_duration > 0 else 0
            total_packet_rate = total_packets / total_duration if total_duration > 0 else 0
```

הפונקציה `print_statistics` אוספת ומיצגנה נתונים סטטיסטיים על זרמי נתונים שנשלחו וקיבלו במהלך חיבור QUIC, ומחשבת מדדים כמו קצב העברת הנתונים וקצב החבילות. הנתונים נשמרים גם בקובץ וגם מוצגים על המסך.

הפונקציה פותחת קובץ בשם `statistics.txt` כתיבה, בו היא תשמור את הסטטיסטיות של הזרמים. הפונקציה עוברת על כל זרם `stream_statistics`, מחלצת את כמות הביטים והחבילות שהועברו, את זמן ההתחלה והסיום של הזרם, ומחשבת את משך הזמן של הזרם ואת קצב העברת הנתונים והחבילות. לאחר מכן, היא כותבת את הסטטיסטיות אלו לקובץ.

```

def print_statistics():
    total_bytes = sum(stats['bytes'] for stats in stream_statistics.values())
    total_packets = sum(stats['packets'] for stats in stream_statistics.values())
    total_duration = max((stats['end_time'] - stats['start_time']) for stats in stream_statistics.values())
    total_data_rate = total_bytes / total_duration if total_duration > 0 else 0
    total_packet_rate = total_packets / total_duration if total_duration > 0 else 0

    file.write("Overall statistics:\n")
    file.write(f"\tData rate: {total_data_rate:.2f} bytes/sec\n")
    file.write(f"\tPacket rate: {total_packet_rate:.2f} packets/sec\n")

    print("\n")
    for stream_id, stats in stream_statistics.items():
        total_bytes = stats['bytes']
        total_packets = stats['packets']
        start_time = stats['start_time']
        end_time = stats['end_time']
        duration = end_time - start_time if end_time and start_time else 0
        data_rate = total_bytes / duration if duration > 0 else 0
        packet_rate = total_packets / duration if duration > 0 else 0
        print(f"Stream {stream_id}:")
        print(f"\tTotal bytes: {total_bytes}")

```

לאחר חישוב הסטטיסטיות לכל זרם, הפקציה מחשבת את הנתונים הכלולים עבור כל הזרמים יחד: כמות הביטים והחbillות הכלולות, משך הזמן הכלול (זרם הארוך ביותר), וקצב העברת הנתונים והחbillות הכלול. בסופו, הפקציה מדפיה את הסטטיסטיות של כל זרם ושל כל הזרמים יחד על המסך.

```

def print_statistics():
    print("\n")
    for stream_id, stats in stream_statistics.items():
        total_bytes = stats['bytes']
        total_packets = stats['packets']
        start_time = stats['start_time']
        end_time = stats['end_time']
        duration = end_time - start_time if end_time and start_time else 0
        data_rate = total_bytes / duration if duration > 0 else 0
        packet_rate = total_packets / duration if duration > 0 else 0
        print(f"Stream {stream_id}:")
        print(f"\tTotal bytes: {total_bytes}")
        print(f"\tTotal packets: {total_packets}")
        print(f"\tData rate: {data_rate:.2f} bytes/sec")
        print(f"\tPacket rate: {packet_rate:.2f} packets/sec")

    total_bytes = sum(stats['bytes'] for stats in stream_statistics.values())
    total_packets = sum(stats['packets'] for stats in stream_statistics.values())
    total_duration = max((stats['end_time'] - stats['start_time']) for stats in stream_statistics.values())
    total_data_rate = total_bytes / total_duration if total_duration > 0 else 0
    total_packet_rate = total_packets / total_duration if total_duration > 0 else 0

    data_rates.append(total_data_rate)
    packet_rates.append(total_packet_rate)
    num_flows_list.append(len(stream_statistics))

```

```

def print_statistics():
    print("\n")
    for stream_id, stats in stream_statistics.items():
        total_bytes = stats['bytes']
        total_packets = stats['packets']
        start_time = stats['start_time']
        end_time = stats['end_time']
        duration = end_time - start_time if end_time and start_time else 0
        data_rate = total_bytes / duration if duration > 0 else 0
        packet_rate = total_packets / duration if duration > 0 else 0
        print(f"Stream {stream_id}:")
        print(f"\tTotal bytes: {total_bytes}")
        print(f"\tTotal packets: {total_packets}")
        print(f"\tData rate: {data_rate:.2f} bytes/sec")
        print(f"\tPacket rate: {packet_rate:.2f} packets/sec")

    total_bytes = sum(stats['bytes'] for stats in stream_statistics.values())
    total_packets = sum(stats['packets'] for stats in stream_statistics.values())
    total_duration = max((stats['end_time'] - stats['start_time']) for stats in stream_statistics.values())
    total_data_rate = total_bytes / total_duration if total_duration > 0 else 0
    total_packet_rate = total_packets / total_duration if total_duration > 0 else 0

    data_rates.append(total_data_rate)
    packet_rates.append(total_packet_rate)
    num_flows_list.append(len(stream_statistics))

    print("Overall statistics:")
    print(f"\tTotal bytes: {total_bytes}")
    print(f"\tTotal packets: {total_packets}")
    print(f"\tData rate: {total_data_rate:.2f} bytes/sec")
    print(f"\tPacket rate: {total_packet_rate:.2f} packets/sec")

```

```
def send_packet(sock, destination, stream_id, frame_offset, data_chunk): 1 usage  new *
    packet = create_quic_packet(stream_id, frame_offset, data_chunk)
    sock.sendto(packet, destination)
    stream_statistics[stream_id]['bytes'] += len(data_chunk)
    stream_statistics[stream_id]['packets'] += 1
    stream_statistics[stream_id]['end_time'] = time.time()
    print(
        f"Sent packet to {destination} with size {len(packet)} bytes for stream {stream_id}, frame offset {frame_offset}")
```

הfonקציה `send_packet` אחראית לשילוח חבילה (packet) של נתונים ליעד מסוים ברשות באמצעות סוקט (socket). היא גם מעדכנת את הסטטיסטיות הנוגעות לזרם (stream) של הנתונים שנשלחו. ייצרת החבילה: קרייה לפונקציה `create_quic_packet` לייצור חבילה (packet) עם הנתונים. ה Fonקציה מחזירה את החבילה בפורמט המתאים לשילוח ברשת. לאחר מכן שולחת את החבילה שנוצרה לכתובה היעד באמצעות ה Fonקציה `sendto` של הסוקט. ומעדכנת את הסטטיסטיות.

Experiment.py

הקובץ Experiment מכיל בתוכו קוד שמבצע רצף של ניסויים לבדיקת ביצועים של מערכת תקשורת, כמספר הזרמים הפעילים משתנה, עבור מספר זרמים בין 1 ל-10. הוא מתחילה בעדכון של הקליינט עם מספר הזרמים הנוכחי, ואז הסרבר מופעל בתהילך נפרד והקוד ימתין 2 שניות כדי שהగדרנו על מנת לתת לסרבר זמן להתחיל לפעול. לאחר מכן המכין הקליינט מריץ את הסקריפט שלו, והקוד ימתין 2 שניות נוספת כדי לאפשר לסרבר לסייע לרשום את הסטטיסטיקות. לבסוף, לאחר שהсер버 נסגר, המידע שנאסף מקובץ הסטטיסטיקות, כולל את קצב הנתונים וקצב החבילות. התוצאות נכתבות לטרמינל, ובסיום מוצגים בגרפים המראים את הקשר בין מספר הזרמים לבין ביצוע המערכת.

בתמונה שכאן ניתן לראות את התהילך ההרצה של הקובץ Experiment.py. נראה לדוגמה את התוצאות של ניסוי 4.

תחילה יודפס לטרמינל שהсер버 מќשיב והתחיל לבצע את הפעולות שלו וכל חבילה שנשלחת נכתבת בטרמינל, עם המידע שלה.

לאחר מכן יודפס בטרמינל התוצאות של כל אחד מ-10 הזרמים עם המידע של כל זרם.

מצורף צילום מסך מההארק שמראה שההארק שומרת שעborות בכל פעם. לבסוף מוצגת תמונה של הגרף של הניסויים שערכנו.

```
quit connection closed
-----
EXPERIMENT Number 4:

Server listening...
Sent packet to ('127.0.0.1', 5060) with size 1966 bytes for stream 1, frame offset 0
Sent packet to ('127.0.0.1', 5060) with size 1966 bytes for stream 1, frame offset 1956
Sent packet to ('127.0.0.1', 5060) with size 1098 bytes for stream 1, frame offset 3912
Sent packet to ('127.0.0.1', 5060) with size 1094 bytes for stream 2, frame offset 0
Sent packet to ('127.0.0.1', 5060) with size 1094 bytes for stream 2, frame offset 1084
Sent packet to ('127.0.0.1', 5060) with size 1094 bytes for stream 2, frame offset 2168
Sent packet to ('127.0.0.1', 5060) with size 1094 bytes for stream 2, frame offset 3252
Sent packet to ('127.0.0.1', 5060) with size 674 bytes for stream 2, frame offset 4336
Sent packet to ('127.0.0.1', 5060) with size 1203 bytes for stream 3, frame offset 0
Sent packet to ('127.0.0.1', 5060) with size 1203 bytes for stream 3, frame offset 1193
Sent packet to ('127.0.0.1', 5060) with size 1203 bytes for stream 3, frame offset 2386
Sent packet to ('127.0.0.1', 5060) with size 1203 bytes for stream 3, frame offset 3579
Sent packet to ('127.0.0.1', 5060) with size 238 bytes for stream 3, frame offset 4772
Sent packet to ('127.0.0.1', 5060) with size 1564 bytes for stream 4, frame offset 0
Sent packet to ('127.0.0.1', 5060) with size 1564 bytes for stream 4, frame offset 1554
Sent packet to ('127.0.0.1', 5060) with size 1564 bytes for stream 4, frame offset 3108
Sent packet to ('127.0.0.1', 5060) with size 348 bytes for stream 4, frame offset 4662
```

```

Stream 1:
    Total bytes: 5000
    Total packets: 3
    Data rate: 25700392.16 bytes/sec
    Packet rate: 15420.24 packets/sec
Stream 2:
    Total bytes: 5000
    Total packets: 5
    Data rate: 29371876.75 bytes/sec
    Packet rate: 29371.88 packets/sec
Stream 3:
    Total bytes: 5000
    Total packets: 5
    Data rate: 28728109.59 bytes/sec
    Packet rate: 28728.11 packets/sec
Stream 4:
    Total bytes: 5000
    Total packets: 4
    Data rate: 38550588.24 bytes/sec
    Packet rate: 30840.47 packets/sec
Overall statistics:
    Total bytes: 20000
    Total packets: 17
    Data rate: 102801568.63 bytes/sec
    Packet rate: 87381.33 packets/sec
QUIC connection closed

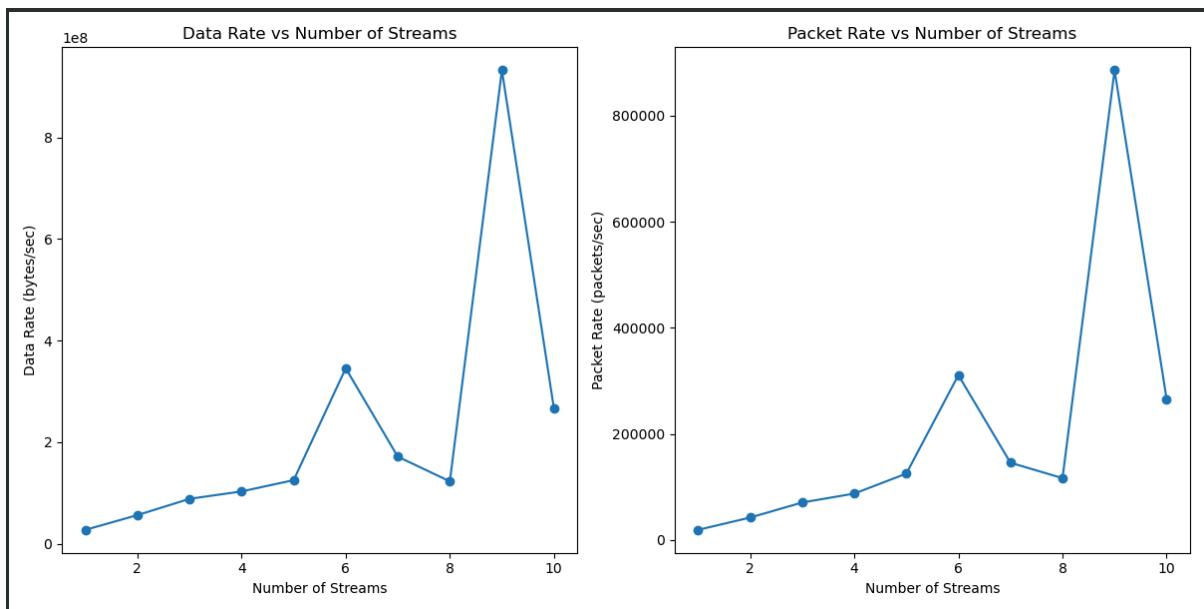
```

הרכבת הניסוי:

בניסוי שלנו אנו שולחים קובץ בגודל 20000 בתים, ובכל ניסוי אנו מקובעים כמהות אחרת של זרימות.

בכל זרימה גודל הקבצים אחיד (מוגדר בין 1000 ל- 2000 בתים) מה שמשפיע על חלוקת המידע בעת השילחה.

מטרת הניסוי הוא להבין איך כמהות הזרימות משפיע על הקצב שליחת הדטה.



בגרף של התוצאות ניתן לראות שכל שmagdilim את מספר הזרימות, קצב שליחת הנתונים הכלול עולה. זאת מכיוון שהדטה מתחולק בין הזרימות השונות וככל שיש יותר זרים, כך אפשר לחלק את הדטה יותר, ולשלוח פחות דטה במקביל בכל זרימה. וכך הדטה מתפרק בקצב מהיר יותר.

ניתן לראות שקיבלנו קפיצות בניסויים 9 ו-6. זה קורה מכיוון שגם מגדים את גודל הפקטה שאפשר לשלוות בכל זרימה. וכנראה שבמקרים אלו הטעינה חלוקה יותר טובה. כאמור, היו יותר זרים מאשר גודל הפקטות קטנה יותר וזה יצר מצב שהזרימה שלחה יותר פקודות בזמן קצר יותר. לדוגמה בניסוי 9 זרימה מס' 5 הייתה קטנה והיא היחידה ששלה 3 פקודות, לעומת ניסוי 8 שבו יש כמה זרים שצריכות לשלוות פקודות גדולות יותר גם 3 פעמים, וניסוי 10 שבו כל הפקודות שנשלחות הן גדולות ומתחולקות בצורה פחות יעילה.

מצורף צילום מסך מההקלטת Wireshark ובאמת ניתן לראות שככל הפקודות נשלוות בגדים שהגדרנו.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	UDP	1525	32970 - 5069 Len=1483
2	0.000171254	127.0.0.1	127.0.0.1	UDP	1525	32970 - 5069 Len=1483
3	0.000256283	127.0.0.1	127.0.0.1	UDP	1525	32970 - 5069 Len=1483
4	0.000302376	127.0.0.1	127.0.0.1	UDP	1525	32970 - 5069 Len=1483
5	0.000341547	127.0.0.1	127.0.0.1	UDP	1525	32970 - 5069 Len=1483
6	0.000386997	127.0.0.1	127.0.0.1	UDP	1525	32970 - 5069 Len=1483
7	0.000426354	127.0.0.1	127.0.0.1	UDP	1525	32970 - 5069 Len=1483
8	0.000468365	127.0.0.1	127.0.0.1	UDP	1525	32970 - 5069 Len=1483
9	0.000507498	127.0.0.1	127.0.0.1	UDP	1525	32970 - 5069 Len=1483
10	0.000546161	127.0.0.1	127.0.0.1	UDP	1525	32970 - 5069 Len=1483
11	0.000584762	127.0.0.1	127.0.0.1	UDP	1525	32970 - 5069 Len=1483
12	0.000660757	127.0.0.1	127.0.0.1	UDP	1525	32970 - 5069 Len=1483
13	0.0006739513	127.0.0.1	127.0.0.1	UDP	1525	32970 - 5069 Len=1483
14	0.0006782353	127.0.0.1	127.0.0.1	UDP	903	32970 - 5069 Len=861
15	0.0006827042	127.0.0.1	127.0.0.1	UDP	47	32970 - 5069 Len=3
16	4.368316048	127.0.0.1	127.0.0.1	UDP	1907	59040 - 5069 Len=1065
17	4.368331480	127.0.0.1	127.0.0.1	UDP	1907	59040 - 5069 Len=1065
18	4.368511134	127.0.0.1	127.0.0.1	UDP	1907	59040 - 5069 Len=1065
19	4.368555761	127.0.0.1	127.0.0.1	UDP	1907	59040 - 5069 Len=1065
20	4.368662798	127.0.0.1	127.0.0.1	UDP	1907	59040 - 5069 Len=1065
21	4.368644544	127.0.0.1	127.0.0.1	UDP	777	59040 - 5069 Len=735
22	4.368731262	127.0.0.1	127.0.0.1	UDP	1258	59040 - 5069 Len=1216
23	4.368780746	127.0.0.1	127.0.0.1	UDP	1258	59040 - 5069 Len=1216
24	4.368825858	127.0.0.1	127.0.0.1	UDP	1258	59040 - 5069 Len=1216
25	4.368865955	127.0.0.1	127.0.0.1	UDP	1258	59040 - 5069 Len=1216
26	4.368904827	127.0.0.1	127.0.0.1	UDP	1258	59040 - 5069 Len=1216
27	4.368942697	127.0.0.1	127.0.0.1	UDP	1258	59040 - 5069 Len=1216
28	4.368980647	127.0.0.1	127.0.0.1	UDP	1258	59040 - 5069 Len=1216
29	4.369830793	127.0.0.1	127.0.0.1	UDP	1258	59040 - 5069 Len=1216
30	4.369885256	127.0.0.1	127.0.0.1	UDP	404	59040 - 5069 Len=362
31	4.369145613	127.0.0.1	127.0.0.1	UDP	47	59040 - 5069 Len=5
32	8.718180372	127.0.0.1	127.0.0.1	UDP	1617	44073 - 5069 Len=1575
33	8.718324245	127.0.0.1	127.0.0.1	UDP	1617	44073 - 5069 Len=1575

תיאור הקוד של הקובץ [Experiment.py](#):

בתחילת הקובץ יש יבוא של ספירות שונות ששמשו בהם בכתיבת הקוד:

subprocess - משמש להפעלת סקRYPTים של קליינט וסרבר

time - ספרייה שמאפשרת לטפל באירועות זמן, לדוגמה בקובץ הנוכחי זה משמש להוספת עיכובים כדי שנוכל להבטיח שהתהליכים יתחלו או יסתיימו.

matplotlib.pyplot - משמש לציר גרפים.

לאחר מכן נגדיר פרמטרים למספר הניסויים וטוווח הזרמים המינימלים והמקסימליים `min_streams` ו-`max_streams`, ונגדיר את `results` שהוא מילון שמאחסן את תוצאות כל ניסוי (מספר זרים, שיעורי נתונים, שיעורי חבילות).

עכשו, נגדיר את הפונקציה `run_experiment` שתציג לנו את הנתונים. היא תכלול בתוכה שינוי של הסקריפט של הקליינט כדי להגדיר את מספר הזרים לניסוי, הפעלת הסקריפט של הסרבר והפעלת הסקריפט של הקליינט עם מספר הזרים הנוכחי.

לאחר מכן היא תסגור את הסרבר ותמתין שהתהליך יסתיים.

```
import subprocess
import time
import matplotlib.pyplot as plt

# Define the number of experiments and the range of stream numbers
min_streams = 1
max_streams = 10
results = {'num_streams': [], 'data_rates': [], 'packet_rates': []}

def run_experiment(streams):
    # Update the number of streams in the client script
    with open('client.py', 'r') as file:
        client_script = file.read()
    client_script = client_script.replace('Number_of_Streams = 10', f'Number_of_Streams = {streams}')
    client_script = client_script.replace('Number_of_Streams = {streams-1}', f'Number_of_Streams = {streams}')
    with open('client.py', 'w') as file:
        file.write(client_script)

    # Start the server
    server_process = subprocess.Popen(['python3', 'server.py'])

    # Give the server time to start
    time.sleep(2)

    # Run the client
    subprocess.run(['python3', 'client.py'])

    # Give the server time to finish writing statistics
    time.sleep(2)

    # Close the server
    server_process.terminate()

    # Wait for the server process to terminate
    server_process.wait()
```

עכשו לאחר שהסרבר נסגר, היא תאסוף סטטיסטיות על ידי קריית קובץ הסטטיסטיות, וכל עוד יש לפחות שורה אחת בקובץ, ככלומר יש לו מידע, היא תאסוף מהקובץ את המידע על שיעורי הנתונים ושיעורי החבילות. במידע שאין יודע הודהה שבקובץ הסטטיסטיות אין מידע.

```
# Collect statistics
try:
    with open('statistics.txt', 'r') as file:
        lines = file.readlines()

    # # Debug: Print the content of the statistics file
    # print("Content of statistics.txt:")
    # for line in lines:
    #     print(line.strip())

    # Extract the data rates and packet rates from the file
    if len(lines) > 1:
        data_rate_line = lines[-2].split(':')
        packet_rate_line = lines[-1].split(':')

        data_rate = float(data_rate_line[1].strip().split()[0]) if len(data_rate_line) > 1 else 0
        packet_rate = float(packet_rate_line[1].strip().split()[0]) if len(packet_rate_line) > 1 else 0

        results['num_streams'].append(num_streams)
        results['data_rates'].append(data_rate)
        results['packet_rates'].append(packet_rate)
    else:
        print("Statistics file does not contain the expected data.")

except Exception as e:
    print(f"Error reading statistics: {e}")
```

עכשו הקוד יריץ את הפונקציה `run_experiment` עבור כל מספר זרים מ-1 עד 10. לאחר שהקוד סיים להריץ את הפונקציה הוא יעבור ליצירת הגрафים. יוצרו שני גרפים, הgraf הראשון מראה את הקשר בין מספר הזרים לשיעור הנתונים והgraf השני מראה את הקשר בין מספר הזרים לשיעור החבילות. לכל גراف יועברו הנתונים המתאימים אליו של מספר הזרים, שיעור החבילות/שיעור הנתונים (בהתאם לגרף), שם לכל ציר בgraf וכותרת לגרף. לבסוף הוא יציג את הגרף על המסך.

```
if __name__ == '__main__':
    for num_streams in range(1, 11): # Adjust range as needed
        print(f"-----\n\nEXPERIMENT Number {num_streams}:\n")
        run_experiment(num_streams)

    plt.figure(figsize=(12, 6))

    plt.subplot(1, 2, 1) Yocheved, 8 hours ago • file size changed and prints sorted
    plt.plot(results['num_streams'], results['data_rates'], marker='o')
    plt.xlabel('Number of Streams')
    plt.ylabel('Data Rate (bytes/sec)')
    plt.title('Data Rate vs Number of Streams')

    plt.subplot(1, 2, 2)
    plt.plot(results['num_streams'], results['packet_rates'], marker='o')
    plt.xlabel('Number of Streams')
    plt.ylabel('Packet Rate (packets/sec)')
    plt.title('Packet Rate vs Number of Streams')

    plt.tight_layout()
    plt.show()
```

test.py

הקובץ משמש להרצת טיטים לפונקציות השונות על מנת לוודא שהן עובדות כראוי. הטיטים כוללים פונקציות לשילחה (quic_send), קבלה (quic_recv), ופירוק חבילות (parse_quic_packet). בנוסף, יש גם בדיקה של הפונקציות המרכזיות בסרבר (server_function) ובקלינט (client_function). הבדיקות כוללות וידוא שהחבילות נשלחות ומתקבלות כראוי, שהסתטיסטיקות מתעדכנות בהתאם, ושפונקציות הסרבר והקלינט פועלות כפי שצריך, כולל טיפול בשגיאות כמו קבצים חסרים או חבילות שאינן שגויות. בקובץ יש שימוש ב-`unittest.mock` כדי לדמות אובייקטים כמו `socket` ופעולות על קבצים כדי שנוכל לבצע בדיקות אמינות.

בתמונה המצורפת ניתן לראות את אופן ההרצה של הטיטים וההדף בטרמינל.

```
Ran 10 tests in 0.005s
OK
```

תיאור הקוד של הקובץ [test.py](#):

בתחילת הקובץ יש יבוא של ספירות שונות ששמשות בהם בכתיבת הקוד לטסטים:
unittest - ספירה לביצוע טסטים.
.mock - ספירה שמאפשרת ליצור אובייקטים מודולים לצורך טסטים.
time - ספירה לטיפול בזמן.
fonctions - שמות שמיובאות מתוך .client, .quic, .server ו-.

```
You, 18 hours ago | 1 author (You)
import unittest
from unittest.mock import patch, mock_open, call
import time
from quic import (
    quic_send,
    quic_recv,
    parse_quic_packet,
    print_statistics,
    create_quic_packet,
    stream_statistics,
)
from server import server_function
from client import client_function
```

class Test

יצירת המחלקה של הטסטים של unittest שכוללת פונקציית הגדרה ראשונית setUp. הפונקציה מכילה בתוכה את המידע של כתובת ה-IP וה포רט של היעד מוגדרים עבורי הטסטים. הפונקציה הזאת תבצע לפני כל פונקציית בדיקה חדשה. בתוך המחלקה יש את כל פונקציות הטסטים שייצרנו.

```
You, 19 hours ago | 1 author (You)
class Test(unittest.TestCase):
    def setUp(self):
        # Address to send messages to
        self.destination = ('127.0.0.1', 5060)
```

test_quic_recv_valid_packet

טוט שמודדא שהפונקציה `quic_recv` מקבלת ומפרשת חבילות תקיןות כראוי. תחיליה ניצור מידע מודמה, ולאחר מכן ניצור חבילה תקינה. לאחר מכן ננקה את הסטטיסטיות על מנת לוודא שהסטטיסטיות שנקלע עכשו הן אכן תקיןות ואז נשלח את המידע המודמה שייצרנו לפונקציה `quic_recv`.
לסיום נבדוק שהחביבה נשלחה בכוונה וshall אחד מהנתונים בסטטיסטיות (ביטים, חבילות, זמן התחלת וזמן סוף) תואם למה שאנוינו צריכים לקבל על מנת לוודא שהפונקציה עובדת כראוי.

```
@patch('socket.socket')
def test_quic_recv_valid_packet(self, mock_socket):
    """Test quic_recv correctly receives and parses a valid packet."""
    mock_sock_instance = mock_socket.return_value

    # Create a valid packet
    stream_id = 1
    frame_offset = 0
    payload = b'Test payload'
    packet = create_quic_packet(stream_id, frame_offset, payload)

    # Mock recvfrom to return our packet
    mock_sock_instance.recvfrom.return_value = (packet, self.destination)

    # Clear before the test
    stream_statistics.clear()

    result = quic_recv(mock_sock_instance)

    self.assertEqual(result, (stream_id, frame_offset, payload))

    # Check statistics
    self.assertIn(stream_id, stream_statistics)
    self.assertEqual(stream_statistics[stream_id]['bytes'], len(payload))
    self.assertEqual(stream_statistics[stream_id]['packets'], 1)
    self.assertIsNotNone(stream_statistics[stream_id]['start_time'])
    self.assertIsNotNone(stream_statistics[stream_id]['end_time'])
```

בתמונה המצורפת ניתן לראות 2 פונקציות טסטים.

test_quic_recv_invalid_packet (1)

הטסט בודק האם quic_recv יודעת להתמודד עם שלילה לא תקינה. במקרה זה ניצור קודם כל את המידע המדומה וניצור שלילה לא תקינה. לאחר מכן נשלח את השלילה בפונקציה .Nonerecv ונוודא שההתוצאה שנקלט היא אכן .None

test_quic_recv_close_packet (2)

הטסט השני מיועד לבדוק שrecv_quic יודעת להתמודד עם חבילות שמסמנות סגירה של החיבור בצורה תקינה. תחילה ניצור קודם כל את המידע המדומה ונשלח את השלילה לפונקציה קר שתחזיר .close לבסוף נבודק שהסוקט נסגר וההתוצאה היא אכן .close

```
@patch('socket.socket')
def test_quic_recv_invalid_packet(self, mock_socket):
    """Test quic_recv handles invalid packet gracefully."""
    mock_sock_instance = mock_socket.return_value

    # Create an invalid packet (insufficient data)
    invalid_packet = b'\x00\x00'

    # Mock recvfrom to return invalid packet
    mock_sock_instance.recvfrom.return_value = (invalid_packet, self.destination)

    result = quic_recv(mock_sock_instance)

    self.assertIsNone(result)

@patch('socket.socket')
def test_quic_recv_close_packet(self, mock_socket):
    """Test quic_recv handles 'close' packet correctly."""
    mock_sock_instance = mock_socket.return_value

    # Mock recvfrom to return 'close' packet
    mock_sock_instance.recvfrom.return_value = (b'close', self.destination)

    result = quic_recv(mock_sock_instance)

    # Check socket is closed
    mock_sock_instance.close.assert_called_once()

    self.assertEqual(result, ('close', None, None))
```

בתמונה כאן ניתן לראות 2 פונקציות טסטים.

test_parse_quic_packet_valid (1)

הטסט בודק האם parse_quic_packet יודעת לנתח נתונים של חבילה בצורה תקינה. תחילת ניצור מספר משתנים שנשתמש בהם ליצירת הפקטה ולביקורת התוצאות הסופיות שהמשתנים יהיו .stream_id, frame_offset, data ניצור את הפקטה על ידי שימוש בפונקציה create_quic_packet ושליפה של הנתונים המתאים parse_quic_packet ולאחר מכן ניצור משתנים חדשים שיקבלו את המידע מהנתונים שהפונקציהparse_quic_packet תחזיר.

test_parse_quic_packet_invalid (2)

הטסט השני מיועד לבדוק שparse_quic_packet יודעת להתמודד עם ניתוח של חבילה לא תקינה. כאן לא יוצרים נתונים לביליה ופשוט יוצרים מראש חבילה לא תקינה. נשלח את החבילה לפונקציה parse_quic_packet ונודע שאכן קיבל None בכל הנתונים.

```
def test_parse_quic_packet_valid(self):
    """Test parse_quic_packet with valid packet."""
    stream_id = 1
    frame_offset = 0
    data = b'Test data'

    packet = create_quic_packet(stream_id, frame_offset, data)

    parsed_stream_id, parsed_frame_offset, parsed_data = parse_quic_packet(packet)

    self.assertEqual(parsed_stream_id, stream_id)
    self.assertEqual(parsed_frame_offset, frame_offset)
    self.assertEqual(parsed_data, data)

def test_parse_quic_packet_invalid(self):
    """Test parse_quic_packet with invalid packet."""
    # Invalid packet - there are missing parts
    invalid_packet = b'\x00\x01'

    result = parse_quic_packet(invalid_packet)

    self.assertEqual(result, (None, None, None))
```

test_print_statistics_with_data

בטעט זה נבדוק האם הֆונקציה `print_statistics` שצריכה לבצע רק את ההפצת של הסטטיסטיות מתבצעת בצורה נכונה. בהתחלה ניצור שני משתנים עם מידע של הסטטיסטיות ולאחר מכן נקרא לפונקציה `print_statistics` ונזוזן שהיא אכן נקראת לפחות פעם אחת.

```
def test_print_statistics_with_data(self):
    """Test print_statistics with populated statistics."""
    # Populate statistics
    stream_statistics[1] = {
        'bytes': 1000,
        'packets': 10,
        'start_time': time.time(),
        'end_time': time.time() + 2 # 2 seconds duration
    }
    stream_statistics[2] = {
        'bytes': 2000,
        'packets': 20,
        'start_time': time.time(),
        'end_time': time.time() + 4 # 4 seconds duration
    }

    with patch('builtins.print') as mock_print:
        print_statistics()

    self.assertTrue(mock_print.call_count > 0)
```

test_server_function

הfonקציה בודקת את התהיליך של השרת מהfonקציה `server_function`. היא בודקת קבלת חבילות על ידי השרת, כולל קבלת חבילה שמסמנת סגירת החיבור. הבדיקה מזדאת שהשרת מזמן כראוי, שהfonקציה `quic_recv` מתבצעת לפי הציפיות, ושהסטטיסטיקות מודפסות. הטוט מתייחס על ידי יצירה של דאטה מדומה ואז קוראת לfonקציית השרת. לאחר מכן בודק את הסגירה וידפים שהשרת מזמן ולסויום יבדוק שההדפסת הסטטיסטיקות אכן נקבעו פעם אחת.

```
@patch('server.quic_recv')
@patch('server.print_statistics')
def test_server_function(self, mock_print_statistics, mock_quic_recv):
    """Test server_function behavior."""
    # Setup mock for quic_recv
    mock_quic_recv.side_effect = [
        (1, 0, b'Data1'),
        (2, 0, b'Data2'),
        ('close', None, None)
    ]

    with patch('socket.socket') as mock_socket_class:
        mock_socket_instance = mock_socket_class.return_value

        with patch('builtins.print') as mock_print:
            server_function()

            # Check socket binding
            mock_socket_instance.bind.assert_called_once_with(('127.0.0.1', 5060))

            # Check 'Server listening...' was printed
            mock_print.assert_any_call("Server listening...")

            # Check print_statistics was called
            mock_print_statistics.assert_called_once()
```

test_client_function

הפונקציה בודקת את התהיליך של הקלינט מהפונקציה client_function שבודקת שליחת קבצים מוחלקים באמצעות QUIC. תחיליה ניצור קובץ מדומה ואז נפתח את הקובץ וניקח ממנו את המידע. לאחר מכן נקרא לפונקציית הקלינט ובודק את מספר הקריאה בקובץ שצריך להיות בין 1 ל 10. לסירוגין נבדוק האם quic_send נשלחה בדיק 10 פעמים, כפי שציריך, והחיבור בquic_close נקרא בבדיקה פעם אחת.

```
@patch('client.quic_send')
@patch('client.quic_close')
def test_client_function(self, mock_quic_close, mock_quic_send):
    """Test client_function behavior."""
    # Mock file data
    mock_file_data = b'Test file data'

    # Mock open to return file data
    m = mock_open(read_data=mock_file_data)

    with patch('builtins.open', m):
        with patch('socket.socket') as mock_socket_class:
            mock_socket_instance = mock_socket_class.return_value

            client_function()

            expected_file_calls = [call(f'file3.txt_part_{i}.txt', 'wb') for i in range(1, 11)]
            m.assert_has_calls(expected_file_calls, any_order=True)

            # Check quic_send was called the expected number of times
            self.assertEqual(mock_quic_send.call_count, 10)

            # Check quic_close was called once
            mock_quic_close.assert_called_once_with(mock_socket_instance, self.destination)
```

Test_quic_send

הבדיקה זו מודדת שפעולות `send_quic` שולחת את הנתונים בצורה נכונה ומעדכנת את הסטטיסטיות של הזרם (.stream).

`expected_packets = (len(data) + packet_size - 1) // packet_size`: מבצע חלוקה כדי לחשב את מספר החבילות שצפויות להישלח, בהתחשב באורך הנתונים ובגודל החבילה.

הבדיקה מודדת שהפונקציה `sendto` של אובייקט הsocket נקראה מספר פעמים כפי שצפוי לפי מספר החבילות שחושב.

הפונקציה מודדת שהערך של הביטים שנשלחו בסטטיסטיות מתאים לכמות הנתונים שנשלחה, עם התאמה של הפחתת 12 ביטים, והמספר של החבילות שנשלחו תואם למספר החבילות שצפוי להישלח, עם התאמה של הפחתת 1 חבילה.

12 הביטים מייצגים את המידע הנוסף שנשלח המציג את ניהול הפרוטוקול (כמו כותרת או ניהול זרם). החבילה אחת מינימלית הניהול. لكن מפתחיתים אותם כדי לבדוק את הנתונים נתו שנשלחו.

```
class Test(unittest.TestCase):
    # Noa Vered Shalom
    @patch('socket.socket')
    # Mock the packet size
    @patch(target='quic.set_stream_packet_size', return_value=10)
    def test_quic_send(self, mock_set_stream_packet_size, mock_socket):
        """Test that quic_send sends data correct and updates the statistics."""
        mock_sock_instance = mock_socket.return_value

        # Use mock data to simulate file content
        data = b'Test'
        stream_id = 1

        quic_send(mock_sock_instance, self.destination, data, stream_id)

        # Calculate expected number of packets
        packet_size = 10
        expected_packets = (len(data) + packet_size - 1) // packet_size # Ceiling division

        # Check sendto was called the expected number of times
        self.assertEqual(mock_sock_instance.sendto.call_count, expected_packets)

        # Check statistics
        # TODO
        self.assertIn(stream_id, stream_statistics)
        self.assertEqual(stream_statistics[stream_id]['bytes'] - 12, len(data)) # Corrected byte count check
        self.assertEqual(stream_statistics[stream_id]['packets'] - 1, expected_packets)
```

test_client_function_file_not_found

הפונקציה בודקת את המקרה שבו אין הקובץ לкриאה לא נמצא בפונקציית הקלינט client_function. היא מדממת מצב שבו הקובץ לא נמצא, על ידי קרייה לפונקציה פתיחת הקובץ ללא קובץ ויצירה side_effect של משתנה הקובץ שייצרנו עם שגיאת FileNotFoundError. לאחר מכן נקרה לפונקציית הקלינט ונודע שבגלל השגיאה שהפונקציות quic_close ו-quic_send לא נקרו.

main

לסיום יש בлок שמבצע את הטעיטים בהפעלת הקובץ שרצה קקובץ הראשי

```
@patch('client.quic_send')
@patch('client.quic_close')
def test_client_function_file_not_found(self, mock_quic_close, mock_quic_send):
    """Test client_function handles file not found errors."""
    # Mock open to raise FileNotFoundError
    m = mock_open()
    m.side_effect = FileNotFoundError

    with patch('builtins.open', m):
        with patch('socket.socket') as mock_socket_class:
            with self.assertRaises(FileNotFoundError):
                client_function()

            # Check quic_send was never called
            mock_quic_send.assert_not_called()

            # Check quic_close was not called
            mock_quic_close.assert_not_called()

if __name__ == '__main__':
    unittest.main()
```