V1:

The command to run server.py is python2 server.py 12345

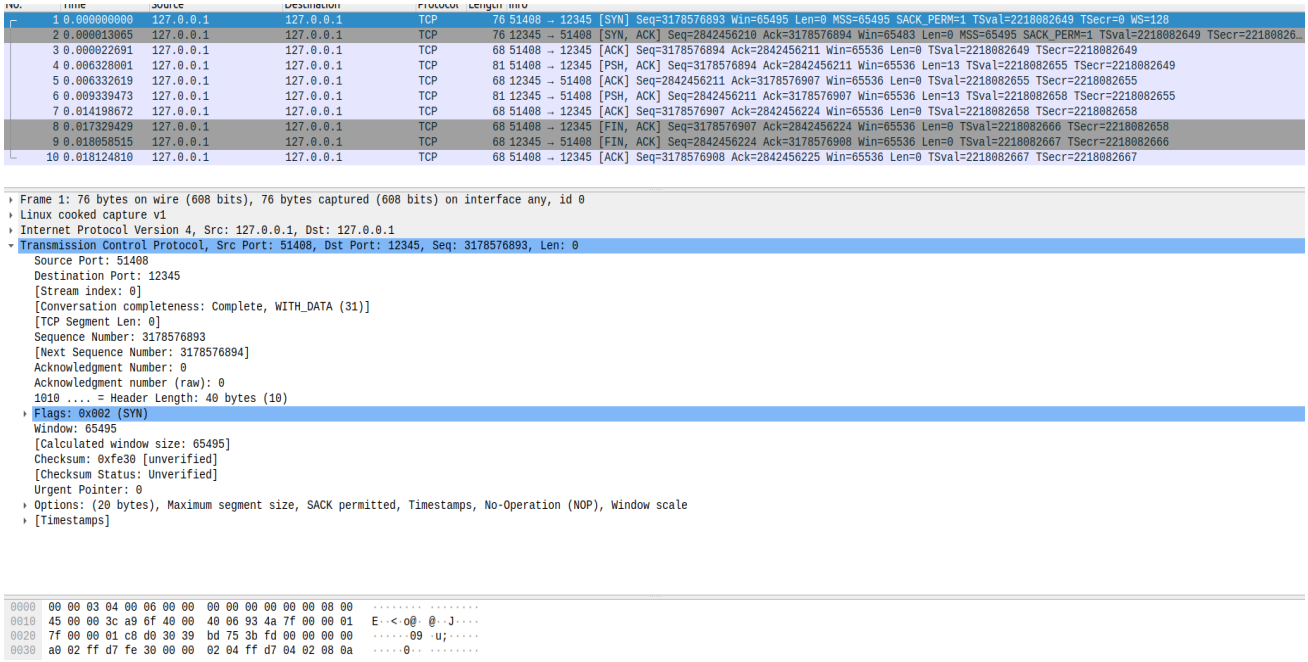
The command to run client.py is python2 client.py 0.0.0.0 12345

```
import socket,sys

TCP_IP = sys.argv[1]
TCP_PORT = int(sys.argv[2])
BUFFER_SIZE = 1024
MESSAGE = "Hello, World!"

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((TCP_IP, TCP_PORT))
s.send(MESSAGE)
data = s.recv(BUFFER_SIZE)
s.close()

print "received data:", data

```

This code creates a socket and connects to a server at the specified IP address and port number passed as arguments to the script. It then sends the message "Hello, World!" to the server and receives a response with a buffer size of 1024 bytes. The response data is then printed to the screen and the socket is closed.

```
import socket,sys

TCP_IP = '0.0.0.0'
TCP_PORT = int(sys.argv[1])
BUFFER_SIZE = 1024

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((TCP_IP, TCP_PORT))
s.listen(1)

while True:
    conn, addr = s.accept()
    print 'New connection from:', addr
    while True:
        data = conn.recv(BUFFER_SIZE)
        if not data: break
        print "received:", data
        conn.send(data.upper())
    conn.close()

```

The code creates a simple TCP server that listens for incoming connections on the IP address 0.0.0.0 and a port number specified as a command-line argument. When a new connection is accepted, the server receives data from the client in chunks of size BUFFER_SIZE, prints the received data, and then echoes the data back to the client in uppercase. The server continues to listen for new connections and handle them in the same way.



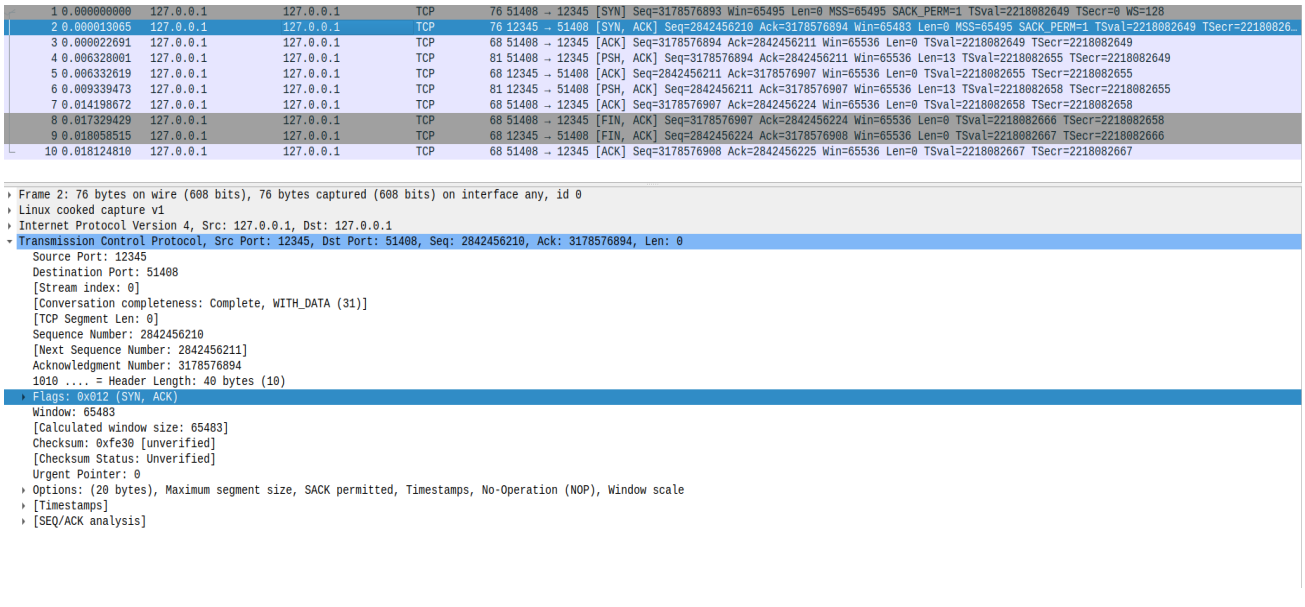
1.This is a SYN packet (SYN flag) and it's the first packet in the handshake and is used to initiate the connection. It contains a sequence number which is: 3178576893 that identifies the starting point for the sequence of data packets that will be exchanged over the connection.

2.The Source Port field which is the client port: 51408 that the OS chose it. indicates the port number of the host that sent the packet which as we said above that the client is asking for a SYN packet which is explained above.

3. The Destination Port field indicates the port number of the host that the packet is being sent to, in this case it's the server port which is: 12345 (chosen by me in the command line) and here the server should be receiving the SYN packet

4. here we should keep in mind that the sequence number is used by the receiver to determine which packets have been received and which ones are missing, based on the ACK number in the received packets (we will see in the next picture)

**sequence number here is : 3178576893.



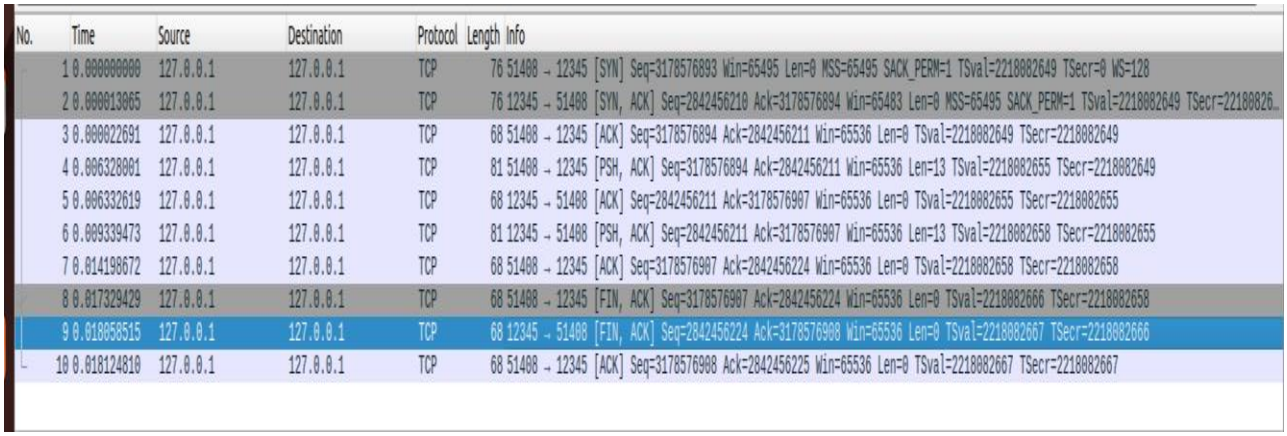
The SYN-ACK packet is the second packet in the handshake and is sent by the server with port: 12345 in response to the SYN packet (we saw above in the first page) from client with port : 51408

Old sequence number is: 3178576893

Ack number: 3178576894

We should pay attention that the ack number is the old sequence number plus 1 and that's because the ACK number is used to acknowledge the receipt of data packets in a TCP connection, and the ACK number is always set to the sequence number of the last received packet plus 1.

Here I'm going to be explaining the transportation in Wireshark :



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	TCP	76	51408 → 12345 [SYN] Seq=3178576893 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=2218882649 TSecr=0 WS=128
2	0.000013065	127.0.0.1	127.0.0.1	TCP	76	12345 → 51408 [SYN, ACK] Seq=2842456210 Ack=3178576894 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=2218882649 TSecr=22188826…
3	0.000022691	127.0.0.1	127.0.0.1	TCP	68	51408 → 12345 [ACK] Seq=3178576894 Ack=2842456211 Win=65536 Len=0 TSval=2218882649 TSecr=2218882649
4	0.006328881	127.0.0.1	127.0.0.1	TCP	81	51408 → 12345 [PSH, ACK] Seq=3178576894 Ack=2842456211 Win=65536 Len=13 TSval=2218882655 TSecr=2218882649
5	0.006332619	127.0.0.1	127.0.0.1	TCP	68	12345 → 51408 [ACK] Seq=2842456211 Ack=3178576907 Win=65536 Len=0 TSval=2218882655 TSecr=2218882655
6	0.009339473	127.0.0.1	127.0.0.1	TCP	81	12345 → 51408 [PSH, ACK] Seq=2842456211 Ack=3178576907 Win=65536 Len=13 TSval=2218882658 TSecr=2218882655
7	0.014198672	127.0.0.1	127.0.0.1	TCP	68	51408 → 12345 [ACK] Seq=3178576907 Ack=2842456224 Win=65536 Len=0 TSval=2218882658 TSecr=2218882658
8	0.017329429	127.0.0.1	127.0.0.1	TCP	68	51408 → 12345 [FIN, ACK] Seq=3178576907 Ack=2842456224 Win=65536 Len=0 TSval=2218882666 TSecr=2218882658
9	0.018058515	127.0.0.1	127.0.0.1	TCP	68	12345 → 51408 [FIN, ACK] Seq=2842456224 Ack=3178576908 Win=65536 Len=0 TSval=2218882667 TSecr=2218882666
10	0.018124810	127.0.0.1	127.0.0.1	TCP	68	51408 → 12345 [ACK] Seq=3178576908 Ack=2842456225 Win=65536 Len=0 TSval=2218882667 TSecr=2218882667

The first two rows I have explained above but now I'm going to explain what in fact happens during the termination process

In rows 9 and 10 we see that the client and the server both sends [FIN, ACK] flag which indicates that the connection is being closed gracefully by both sides and that's what indeed happening with code it self (both client and server asking for a connection termination)

V2:

The command to run server.py is python2 server.py 8080

The command to run client.py is python2 client.py 0.0.0.0 8080

Code explanation :

```
import socket,sys

TCP_IP = '0.0.0.0'
TCP_PORT = int(sys.argv[1])
BUFFER_SIZE = 5

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((TCP_IP, TCP_PORT))
s.listen(1)

while True:
      conn, addr = s.accept()
      print 'New connection from:', addr
      while True:
          data = conn.recv(BUFFER_SIZE)
          if not data: break
          print "received:", data
          conn.send(data.upper())
      conn.close()
```

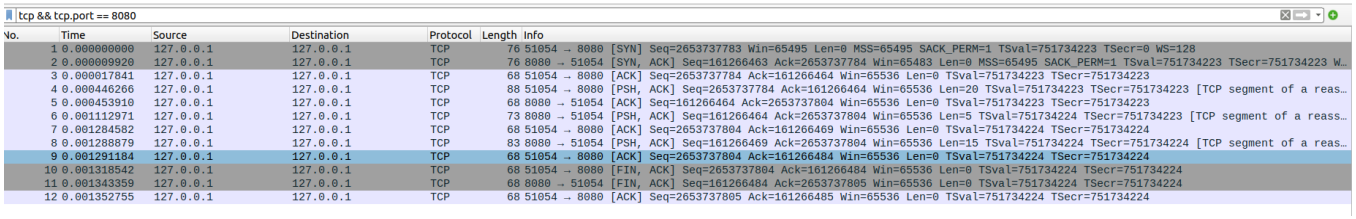
This server listens on a specified port (given as an argument) for incoming connections. When a connection is received, the server will receive data from the client in small chunks of 5 bytes at a time. The server will then print the received data to the console and send it back to the client in upper case.

```
import socket,sys

TCP_IP = sys.argv[1]
TCP_PORT = int(sys.argv[2])
BUFFER_SIZE = 1024
MESSAGE = "World! Hello, World!"

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((TCP_IP, TCP_PORT))
s.send(MESSAGE)
data = s.recv(BUFFER_SIZE)
s.close()

print "received data:", data
```

This client that connects to a server at a specified IP address and port, sends a message to the server, and then receives a response from the server. The client then prints the received data to the console and closes the connection.

Wireshark transportation explanation:

Note: we are now focusing on the differences between V2 to V1 here


tcp && tcp.port == 8080

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	TCP	76	51054 → 8080 [SYN] Seq=2653737783 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=751734223 TSecr=0 WS=128
2	0.000009920	127.0.0.1	127.0.0.1	TCP	76	8080 → 51054 [SYN, ACK] Seq=161266463 Ack=2653737784 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=751734223 TSecr=751734223 W…
3	0.000017841	127.0.0.1	127.0.0.1	TCP	68	51054 → 8080 [ACK] Seq=2653737784 Ack=161266464 Win=65536 Len=0 TSval=751734223 TSecr=751734223
4	0.000446266	127.0.0.1	127.0.0.1	TCP	88	51054 → 8080 [PSH, ACK] Seq=2653737784 Ack=161266464 Win=65536 Len=20 TSval=751734223 TSecr=751734223 [TCP segment of a reas…
5	0.000453910	127.0.0.1	127.0.0.1	TCP	68	8080 → 51054 [ACK] Seq=161266464 Ack=2653737804 Win=65536 Len=0 TSval=751734223 TSecr=751734223
6	0.001112971	127.0.0.1	127.0.0.1	TCP	73	8080 → 51054 [PSH, ACK] Seq=161266464 Ack=2653737804 Win=65536 Len=5 TSval=751734224 TSecr=751734223 [TCP segment of a reass…
7	0.001284582	127.0.0.1	127.0.0.1	TCP	68	51054 → 8080 [ACK] Seq=2653737804 Ack=161266469 Win=65536 Len=0 TSval=751734224 TSecr=751734224
8	0.001288879	127.0.0.1	127.0.0.1	TCP	83	8080 → 51054 [PSH, ACK] Seq=161266469 Ack=2653737804 Win=65536 Len=15 TSval=751734224 TSecr=751734224 [TCP segment of a reas…
9	0.001291184	127.0.0.1	127.0.0.1	TCP	68	51054 → 8080 [ACK] Seq=2653737804 Ack=161266484 Win=65536 Len=0 TSval=751734224 TSecr=751734224
10	0.001318542	127.0.0.1	127.0.0.1	TCP	68	51054 → 8080 [FIN, ACK] Seq=2653737804 Ack=161266484 Win=65536 Len=0 TSval=751734224 TSecr=751734224
11	0.001343359	127.0.0.1	127.0.0.1	TCP	68	8080 → 51054 [FIN, ACK] Seq=161266484 Ack=2653737805 Win=65536 Len=0 TSval=751734224 TSecr=751734224
12	0.001352755	127.0.0.1	127.0.0.1	TCP	68	51054 → 8080 [ACK] Seq=2653737805 Ack=161266485 Win=65536 Len=0 TSval=751734224 TSecr=751734224

Lines: 1, 2 like V1 as I explained above, and we know the connection was established successfully

We will focus on the rest of the lines starting from 4:

1.Once the connection is established, the client will send a PSH-ACK (Push-Acknowledgment) packet (we know that the client did that due to the src port which is: 51054) to the server, which includes the PSH and ACK flags, and the sequence number for the data that is being sent. This packet indicates that the client is ready to transmit the data in the packet, and that the server has acknowledged the receipt of the data.

2.The server will then receive the data in the PSH-ACK packet and process it. In this case, the server will print the received data to the console and then send it back to the client in upper case.

3.The server will then send a PSH-ACK packet back to the client, which includes the PSH and ACK flags, and the sequence number for the data that is being sent. This packet indicates that the server is ready to transmit the data in the packet, and that the client has acknowledged the receipt of the data.

4.The client will then receive the data in the PSH-ACK packet and process it. In this case, the client will print the received data to the console and then close the connection.

5.The server will then send a FIN-ACK (Finish-Acknowledgment) packet to the client

Keep in mind that the main difference between V1 and V2 is here where the server send a [PSH,ACK] twice the first one contains data with length 5 because this is the size of the buffer (BUFF_SIZE=5) and because this was the first package we weren't waiting for a previous ACK and the second one with length 15 and that's because the server while waiting to first ack saved the data in a buffer and when the ACK was received from the client the server sent all the data in a buffer

V3:

The command to run server.py is python2 server.py 9090

The command to run client.py is python2 client.py 0.0.0.0 9090

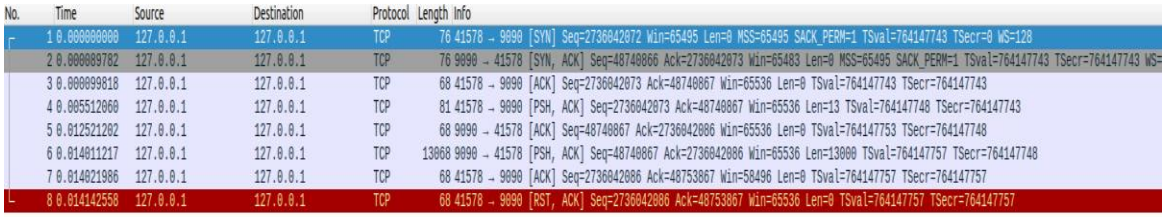
```
import socket,sys

TCP_IP = '0.0.0.0'
TCP_PORT = int(sys.argv[1])
BUFFER_SIZE = 1024

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((TCP_IP, TCP_PORT))
s.listen(1)

while True:
      conn, addr = s.accept()
      print 'New connection from:', addr
      while True:
          data = conn.recv(BUFFER_SIZE)
          if not data: break
          print "received:", data
          conn.send(data.upper()*1000)
      conn.close()
```

This code listens for incoming connections on a specified port. It uses the socket module to create a socket and bind it to the specified IP address and port (that is taken as an argument) . It then listens for incoming connections, and when a new connection is made, it prints the address of the client and receives data from the client. It then prints the received data and sends it back to the client in upper case, 1000 times. The server continues to listen for incoming connections and repeats this process forever (outer while never stops) .

```
import socket,sys

TCP_IP = sys.argv[1]
TCP_PORT = int(sys.argv[2])
BUFFER_SIZE = 1024
MESSAGE = "Hello, World!"

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((TCP_IP, TCP_PORT))
s.send(MESSAGE)
data = s.recv(BUFFER_SIZE)
print "received data:", data
data = s.recv(BUFFER_SIZE)
print "received data:", data
s.close()
```

This code create a client that connects to a server using the IP address and port number specified as command line arguments. The client then sends a message to the server and receives two responses, which are printed to the console. The client then closes the connection.

Wireshark transportation explanation



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	TCP	76	41578 → 9090 [SYN] Seq=2736042872 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=764147743 TSecr=0 WS=128
2	0.000089782	127.0.0.1	127.0.0.1	TCP	76	9090 → 41578 [SYN, ACK] Seq=48740866 Ack=2736042873 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=764147743 TSecr=764147743 WS=
3	0.000099818	127.0.0.1	127.0.0.1	TCP	68	41578 → 9090 [ACK] Seq=2736042873 Ack=48740867 Win=65536 Len=0 TSval=764147743 TSecr=764147743
4	0.005512060	127.0.0.1	127.0.0.1	TCP	81	41578 → 9090 [PSH, ACK] Seq=2736042873 Ack=48740867 Win=65536 Len=13 TSval=764147748 TSecr=764147743
5	0.012521202	127.0.0.1	127.0.0.1	TCP	68	9090 → 41578 [ACK] Seq=48740867 Ack=2736042886 Win=65536 Len=0 TSval=764147753 TSecr=764147748
6	0.014011217	127.0.0.1	127.0.0.1	TCP	13068	9090 → 41578 [PSH, ACK] Seq=48740867 Ack=2736042886 Win=65536 Len=13000 TSval=764147757 TSecr=764147748
7	0.014021986	127.0.0.1	127.0.0.1	TCP	68	41578 → 9090 [ACK] Seq=2736042886 Ack=48753867 Win=58496 Len=0 TSval=764147757 TSecr=764147757
8	0.014142558	127.0.0.1	127.0.0.1	TCP	68	41578 → 9090 [RST, ACK] Seq=2736042886 Ack=48753867 Win=65536 Len=0 TSval=764147757 TSecr=764147757

In this capture of wire shark after running the client and server in V3 we notice a huge difference which is the client sends a RST ACK packet to the server, it is essentially telling the server to reset the connection. This happened because the client is unable to process the large packet (13,000 byte) from the server and needs to terminate the connection. Other than that it is similar to V1 and V2 in terms of initiating connection (3-way handshake)

V4:

The command to run server.py is python2 server.py 12345

The command to run client.py is python2 client.py 0.0.0.0 12345

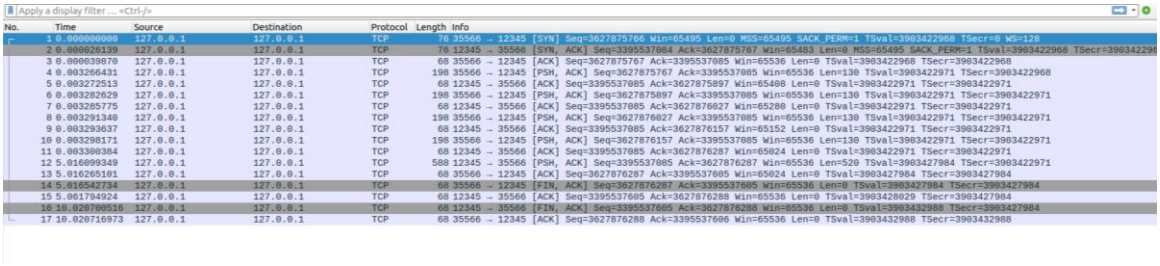
```
import socket,sys

TCP_IP = sys.argv[1]
TCP_PORT = int(sys.argv[2])
BUFFER_SIZE = 1024
MESSAGE = "Hello, World!"

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((TCP_IP, TCP_PORT))
s.send(MESSAGE*10)
s.send(MESSAGE*10)
s.send(MESSAGE*10)
s.send(MESSAGE*10)
data = s.recv(BUFFER_SIZE)
s.close()

print "received data:", data
```

This is a TCP client that connects to a specified IP address and port. Once connected, the client sends the message "Hello, World!" multiple times (each time he sends is 10 times) to the server and then receives data back from the server. The client then prints the data it receives from the server to the console.

```
import socket,sys,time

TCP_IP = '0.0.0.0'
TCP_PORT = int(sys.argv[1])
BUFFER_SIZE = 1024

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((TCP_IP, TCP_PORT))
s.listen(1)

while True:
      conn, addr = s.accept()
      print 'New connection from:', addr
      while True:
          time.sleep(5)
          data = conn.recv(BUFFER_SIZE)
          if not data: break
          print "received:", data
          conn.send(data.upper())
      conn.close()
```

This is a TCP server that listens for incoming connections on a specified port. When a new connection is established, the server receives data from the client in 1024-byte chunks and sends the data back to the client in all upper case. The server also prints the incoming data to the console, and it introduces a delay (for 5 seconds will explain this in depth in the Wireshark capture) between receiving data from the client.

Explanation of the transport layer we see in Wireshark:



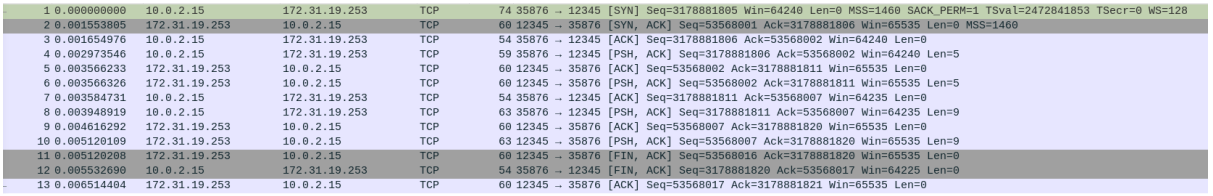
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	TCP	76	35566 → 12345 [SYN] Seq=3627875766 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=3903422968 TSecr=0 WS=128
2	0.000026139	127.0.0.1	127.0.0.1	TCP	76	12345 → 35566 [SYN, ACK] Seq=3395537084 Ack=3627875767 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=3903422968 TSecr=3903422296
3	0.000039870	127.0.0.1	127.0.0.1	TCP	68	35566 → 12345 [ACK] Seq=3627875767 Ack=3395537085 Win=65536 Len=0 TSval=3903422968 TSecr=3903422968
4	0.003266431	127.0.0.1	127.0.0.1	TCP	198	35566 → 12345 [PSH, ACK] Seq=3627875767 Ack=3395537085 Win=65536 Len=130 TSval=3903422971 TSecr=3903422968
5	0.003272513	127.0.0.1	127.0.0.1	TCP	68	12345 → 35566 [ACK] Seq=3395537085 Ack=3627875897 Win=65408 Len=0 TSval=3903422971 TSecr=3903422971
6	0.003282629	127.0.0.1	127.0.0.1	TCP	198	35566 → 12345 [PSH, ACK] Seq=3627875897 Ack=3395537085 Win=65536 Len=130 TSval=3903422971 TSecr=3903422971
7	0.003285775	127.0.0.1	127.0.0.1	TCP	68	12345 → 35566 [ACK] Seq=3395537085 Ack=3627876027 Win=65280 Len=0 TSval=3903422971 TSecr=3903422971
8	0.003291340	127.0.0.1	127.0.0.1	TCP	198	35566 → 12345 [PSH, ACK] Seq=3627876027 Ack=3395537085 Win=65536 Len=130 TSval=3903422971 TSecr=3903422971
9	0.003293637	127.0.0.1	127.0.0.1	TCP	68	12345 → 35566 [ACK] Seq=3395537085 Ack=3627876157 Win=65152 Len=0 TSval=3903422971 TSecr=3903422971
10	0.003298171	127.0.0.1	127.0.0.1	TCP	198	35566 → 12345 [PSH, ACK] Seq=3627876157 Ack=3395537085 Win=65536 Len=130 TSval=3903422971 TSecr=3903422971
11	0.003300384	127.0.0.1	127.0.0.1	TCP	68	12345 → 35566 [ACK] Seq=3395537085 Ack=3627876287 Win=65024 Len=0 TSval=3903422971 TSecr=3903422971
12	5.016099349	127.0.0.1	127.0.0.1	TCP	588	12345 → 35566 [PSH, ACK] Seq=3395537085 Ack=3627876287 Win=65536 Len=520 TSval=3903427984 TSecr=3903422971
13	5.016265101	127.0.0.1	127.0.0.1	TCP	68	35566 → 12345 [ACK] Seq=3627876287 Ack=3395537605 Win=65024 Len=0 TSval=3903427984 TSecr=3903427984
14	5.016542734	127.0.0.1	127.0.0.1	TCP	68	35566 → 12345 [FIN, ACK] Seq=3627876287 Ack=3395537605 Win=65536 Len=0 TSval=3903427984 TSecr=3903427984
15	5.061794924	127.0.0.1	127.0.0.1	TCP	68	12345 → 35566 [ACK] Seq=3395537605 Ack=3627876288 Win=65536 Len=0 TSval=3903428029 TSecr=3903427984
16	10.020700516	127.0.0.1	127.0.0.1	TCP	68	12345 → 35566 [FIN, ACK] Seq=3395537605 Ack=3627876288 Win=65536 Len=0 TSval=3903432988 TSecr=3903427984
17	10.020716973	127.0.0.1	127.0.0.1	TCP	68	35566 → 12345 [ACK] Seq=3627876288 Ack=3395537606 Win=65536 Len=0 TSval=3903432988 TSecr=3903432988

Here we see the client [PSH,ACK] multiple times and that's due to the client is sending a message 4 time and each time its getting ACKed from the server and then the server sends all the data he received with length 520 which is the sum lengths that the client sends and the main difference between what we see here and any other capture before is that in the Time section we see a delay of 5 seconds from the moment the server ACKed the last PSH,ACK from the client to when the server sent the PSH,ACK to the client further more the client asked to terminate the connection by sending the flag FIN,ACK and from the moment the server ACKed it another delay of 5 seconds took place (look at lines 16 ,17) and that's because the client sent an empty data when he asked for termination then the server has to delay for 5 seconds and then he sends a FIN,ACK flag to ask the client to end the communication between them and then the client ACKs the the FIN,ACK

Note: I did not explain the what flags indications here because I explained earlier .

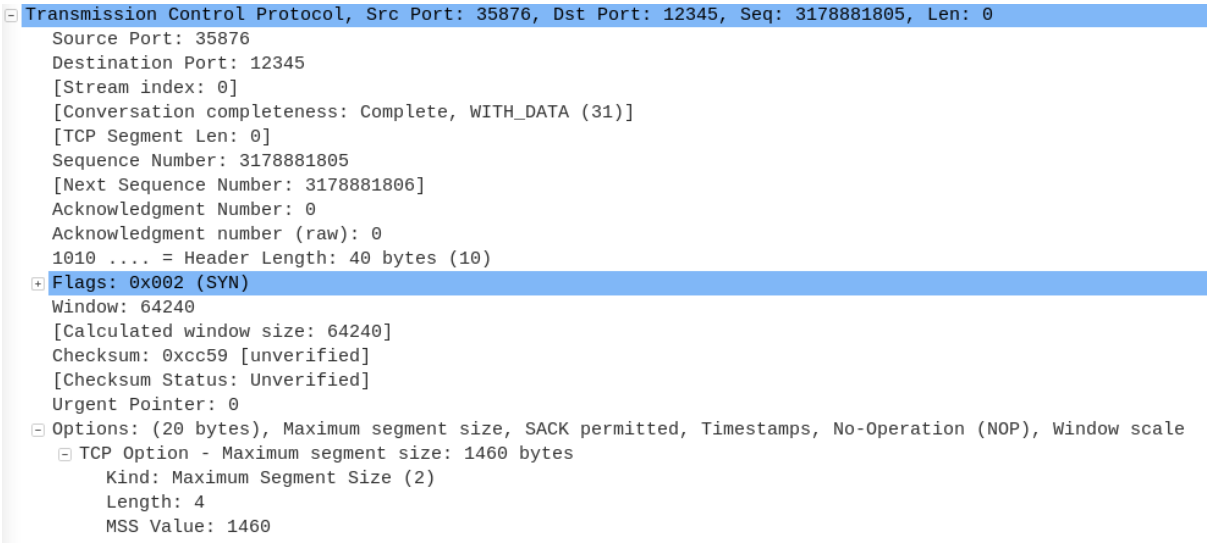
חלק א:

הרצת קטעי הקוד tcp_client.py ו- tcp_server.py על מחשבים שונים, ופירוט על התעבורה



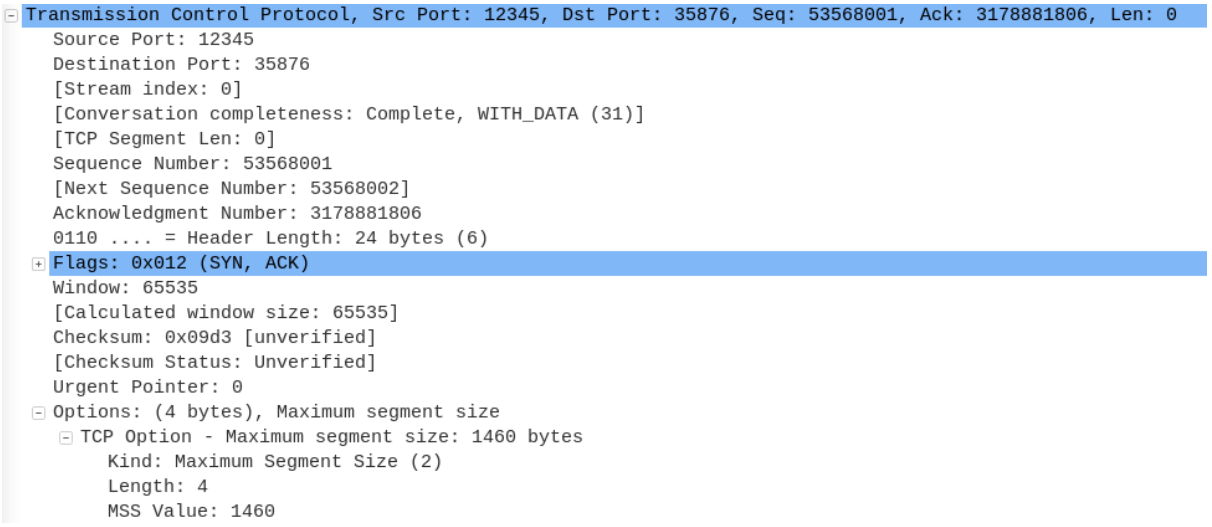
1	0.000000000	10.0.2.15	172.31.19.253	TCP	74	35876 → 12345 [SYN] Seq=3178881805 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=2472841853 TSecr=0 WS=128
2	0.001553805	172.31.19.253	10.0.2.15	TCP	60	12345 → 35876 [SYN, ACK] Seq=53568001 Ack=3178881806 Win=65535 Len=0 MSS=1460
3	0.001654976	10.0.2.15	172.31.19.253	TCP	54	35876 → 12345 [ACK] Seq=3178881806 Ack=53568002 Win=64240 Len=0
4	0.002973546	10.0.2.15	172.31.19.253	TCP	59	35876 → 12345 [PSH, ACK] Seq=3178881806 Ack=53568002 Win=64240 Len=5
5	0.003566233	172.31.19.253	10.0.2.15	TCP	60	12345 → 35876 [ACK] Seq=53568002 Ack=3178881811 Win=65535 Len=0
6	0.003566326	172.31.19.253	10.0.2.15	TCP	60	12345 → 35876 [PSH, ACK] Seq=53568002 Ack=3178881811 Win=65535 Len=5
7	0.003584731	10.0.2.15	172.31.19.253	TCP	54	35876 → 12345 [ACK] Seq=3178881811 Ack=53568007 Win=64235 Len=0
8	0.003948919	10.0.2.15	172.31.19.253	TCP	63	35876 → 12345 [PSH, ACK] Seq=3178881811 Ack=53568007 Win=64235 Len=9
9	0.004616292	172.31.19.253	10.0.2.15	TCP	60	12345 → 35876 [ACK] Seq=53568007 Ack=3178881820 Win=65535 Len=0
10	0.005120109	172.31.19.253	10.0.2.15	TCP	63	12345 → 35876 [PSH, ACK] Seq=53568007 Ack=3178881820 Win=65535 Len=9
11	0.005120208	172.31.19.253	10.0.2.15	TCP	60	12345 → 35876 [FIN, ACK] Seq=53568016 Ack=3178881820 Win=65535 Len=0
12	0.005532690	10.0.2.15	172.31.19.253	TCP	54	35876 → 12345 [FIN, ACK] Seq=3178881820 Ack=53568017 Win=64225 Len=0
13	0.006514404	172.31.19.253	10.0.2.15	TCP	60	12345 → 35876 [ACK] Seq=53568017 Ack=3178881821 Win=65535 Len=0

שלבים 1 - 3 הם שלב "לחיצת הידיים"
שלבים 4 - 11 מתארים את העברת המידע בין השרת ללקוח וכנל בין הלקוח לשרת
שלבים 11 - 13 הם הודעות סיום התקשורת

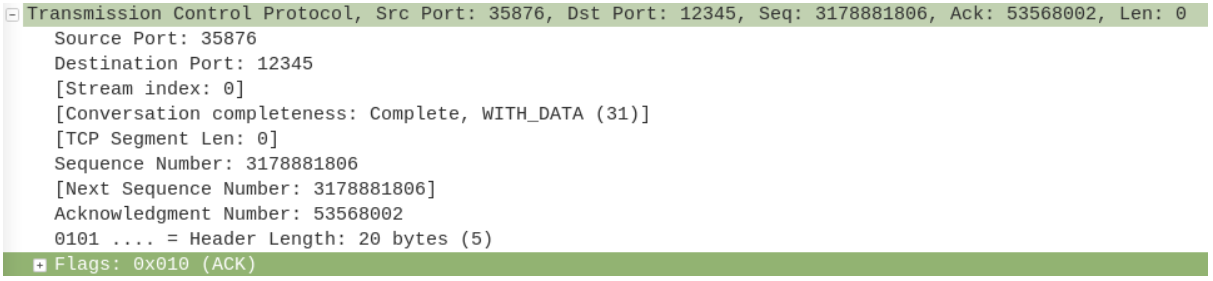


```
Transmission Control Protocol, Src Port: 35876, Dst Port: 12345, Seq: 3178881805, Len: 0
    Source Port: 35876
    Destination Port: 12345
    [Stream index: 0]
    [Conversation completeness: Complete, WITH_DATA (31)]
    [TCP Segment Len: 0]
    Sequence Number: 3178881805
    [Next Sequence Number: 3178881806]
    Acknowledgment Number: 0
    Acknowledgment number (raw): 0
    1010 .... = Header Length: 40 bytes (10)
  + Flags: 0x002 (SYN)
    Window: 64240
    [Calculated window size: 64240]
    Checksum: 0xcc59 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  - Options: (20 bytes), Maximum segment size, SACK permitted, Timestamps, No-Operation (NOP), Window scale
    - TCP Option - Maximum segment size: 1460 bytes
          Kind: Maximum Segment Size (2)
          Length: 4
          MSS Value: 1460
```

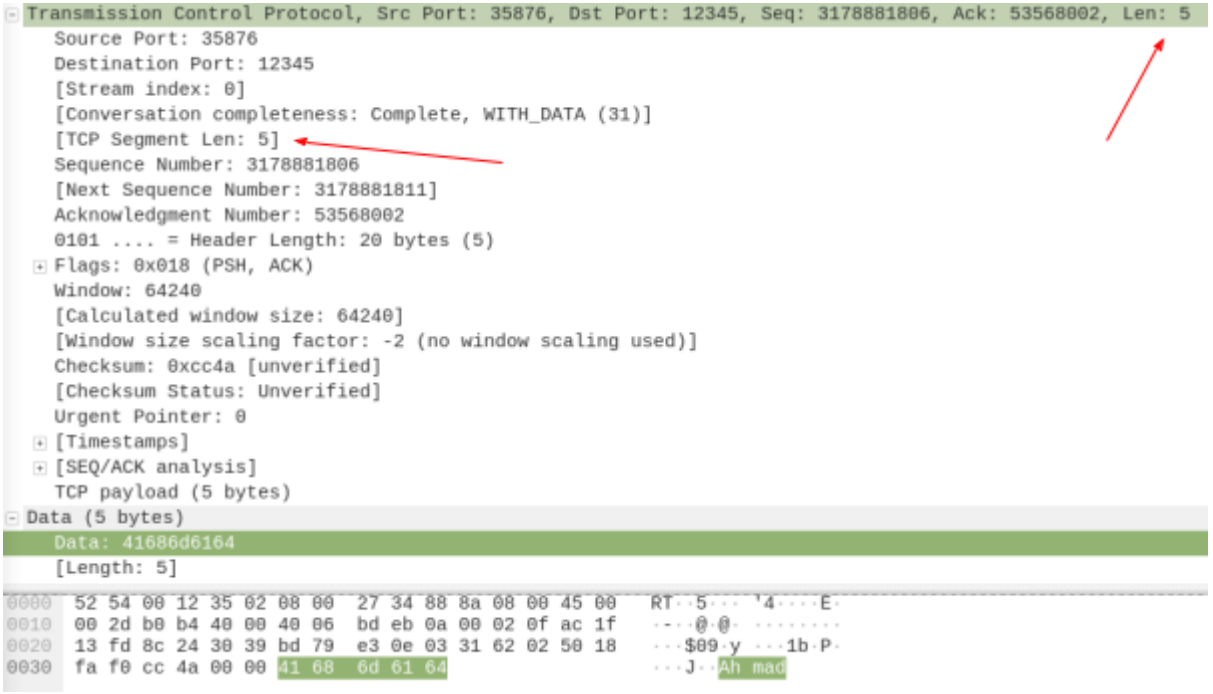
נתבונן על ההודעה הראשונה אשר נשלחת מהלקוח אל השרת.
ניתן לראות כי בתחילית הTCP תחת לשונית הFlags מופיע דגל הSYN.
הלקוח מבקש לעשות סינכרוניזציה (להסתנכרן) עם השרת.
פורט הלקוח הוא: 35876
פורט השרת הוא: 12345.
מבחינת מספרים סידוריים, נתבונן ב - Sequence number וב- Acknowledgment number:
Sequence number: 3178881805
Acknowledgment number: 0
כלומר הלקוח מתחיל את התקשורת ממספר זה (seq) ומאשר שקיבל עד מספר זה (ack).
נראה כיצד מספרים אלו משתנים לאורך התקשורת.



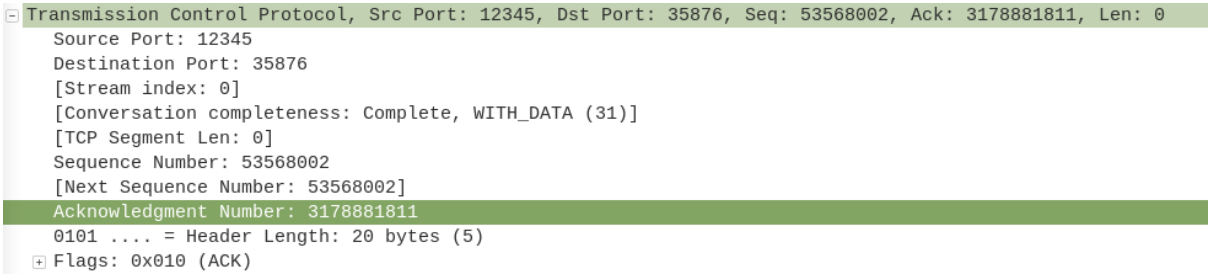
כעת השרת מחזיר ללקוח הודעת SYN,ACK. כלומר הוא מאשר את בקשת הסינכרוניזציה (ACK) של השרת ומבקש להסתנכרן עם הלקוח (SYN).
נשים לב שכעת הseq num של השרת הוא 53568001 (השרת שולח הודעות החלק ממספר זה), בעוד שה-ack num הוא 3178881806 שזה בדיוק 1 מעל הseq num של הלקוח (השרת קיבל עד מספר זה). כלומר, השרת שולח חזרה ללקוח 1 נוסף שמסמן את הack לחיבור.



בשלב האחרון בחלק שנקרא לו "לחיצת הידיים" הלקוח שולח חזרה אל השרת הודעת ack המאשרת את הסינכרון שביקש השרת.
ניתן לראות כי הseq num כעת זהה לack num של השרת: 3178881806. כלומר הלקוח קיבל את האישור של השרת לבקשת הסינכרון שהוא (הלקוח) שלח אליו (השרת).
מצד שני, הack num הוא כעת 53568002, כלומר 1 מעל הack num של השרת. משמעות הדבר היא שהלקוח מאשר את הסינכרון בכך ששולח חזרה לשרת 1 נוסף שמסמן את הack לחיבור.



כעת הלקוח שולח אל השרת את השם "Ahmad". ניתן לראות זאת הפירוט הדאטא.
בנוסף, ניתן לראות כי אורך ההודעה הוא 5 (מסומן באדום).
מספרי הseq/ack לא השתנו כצפוי.



כאן ניתן לראות שהשרת מחזיר ללקוח הודעת ack המאשרת את קבלת המידע ("Ahmad") ומספר הack הוא בהתאם גדול ב5 ממספר הseq של הלקוח (מספר הack הקודם של השרת): 3178881811. כלומר השרת מאשר שקרא עד מספר זה.

```
Transmission Control Protocol, Src Port: 12345, Dst Port: 35876, Seq: 53568002, Ack: 3178881811, Len: 5
    Source Port: 12345
    Destination Port: 35876
    [Stream index: 0]
    [Conversation completeness: Complete, WITH_DATA (31)]
    [TCP Segment Len: 5]
    Sequence Number: 53568002
    [Next Sequence Number: 53568007]
    Acknowledgment Number: 3178881811
    0101 .... = Header Length: 20 bytes (5)
  + Flags: 0x018 (PSH, ACK)
    Window: 65535
    [Calculated window size: 65535]
    [Window size scaling factor: -2 (no window scaling used)]
    Checksum: 0x0eb4 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  + [Timestamps]
  + [SEQ/ACK analysis]
    TCP payload (5 bytes)
Data (5 bytes)
    Data: 41686d6164
    [Length: 5]
```

```
0000  08 00 27 34 88 8a 52 54  00 12 35 02 08 00 45 00   ··'4··RT ··5···E·
0010  00 2d 10 79 00 00 40 06  9e 27 ac 1f 13 fd 0a 00   ·-·y··@· ·'·····
0020  02 0f 30 39 8c 24 03 31  62 02 bd 79 e3 13 50 18   ··09·$·1 b··y··P·
0030  ff ff 0e b4 00 00 41 68  6d 61 64 00               ·····Ah mad·
```

כעת השרת שולח חזרה ללקוח את המידע שהוא קיבל. ניתן לראות זאת בפירוט הדאטא.
בשלב זה מספרי הseq/ack נשארים ללא שינוי כצפוי.

```
Transmission Control Protocol, Src Port: 35876, Dst Port: 12345, Seq: 3178881811, Ack: 53568007, Len: 0
    Source Port: 35876
    Destination Port: 12345
    [Stream index: 0]
    [Conversation completeness: Complete, WITH_DATA (31)]
    [TCP Segment Len: 0]
    Sequence Number: 3178881811
    [Next Sequence Number: 3178881811]
    Acknowledgment Number: 53568007
    0101 .... = Header Length: 20 bytes (5)
  + Flags: 0x010 (ACK)
```

הלקוח מקבל את המידע מהשרת ומחזיר ack. ניתן לראות שמספר הack גדל ב5 וכעת הוא 53568007.

```
Transmission Control Protocol, Src Port: 35876, Dst Port: 12345, Seq: 3178881811, Ack: 53568007, Len: 9
    Source Port: 35876
    Destination Port: 12345
    [Stream index: 0]
    [Conversation completeness: Complete, WITH_DATA (31)]
    [TCP Segment Len: 9]
    Sequence Number: 3178881811
    [Next Sequence Number: 3178881820]
    Acknowledgment Number: 53568007
    0101 .... = Header Length: 20 bytes (5)
  + Flags: 0x018 (PSH, ACK)
    Window: 64235
    [Calculated window size: 64235]
    [Window size scaling factor: -2 (no window scaling used)]
    Checksum: 0xcc4e [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  + [Timestamps]
  + [SEQ/ACK analysis]
    TCP payload (9 bytes)
Data (9 bytes)
    Data: 323131343337323233
    [Length: 9]
```

```
0000  52 54 00 12 35 02 08 00  27 34 88 8a 08 00 45 00   RT··5··· '4····E·
0010  00 31 b0 b6 40 00 40 06  bd e5 0a 00 02 0f ac 1f   ·1··@·@· ········
0020  13 fd 8c 24 30 39 bd 79  e3 13 03 31 62 07 50 18   ···$09·y ···1b·P·
0030  fa eb cc 4e 00 00 32 31  31 34 33 37 32 32 33      ···N··21 1437223
```

הלקוח שולח אל השרת מספר ת.ז. אורך ההודעה הוא 9 (מסומן באדום). מספרי seq/ack נותרים ללא שינוי.

```
Transmission Control Protocol, Src Port: 12345, Dst Port: 35876, Seq: 53568007, Ack: 3178881820, Len: 0
    Source Port: 12345
    Destination Port: 35876
    [Stream index: 0]
    [Conversation completeness: Complete, WITH_DATA (31)]
    [TCP Segment Len: 0]
    Sequence Number: 53568007
    [Next Sequence Number: 53568007]
    Acknowledgment Number: 3178881820
    0101 .... = Header Length: 20 bytes (5)
  + Flags: 0x010 (ACK)
```

השרת מאשר את קבלת המידע (מספר ת.ז) מהלקוח. מספר ack גדל ב9 וכעת הוא 3178881820 בעוד שמספר seq התעדכן ל- 53568007.

```
Transmission Control Protocol, Src Port: 12345, Dst Port: 35876, Seq: 53568007, Ack: 3178881820, Len: 9
    Source Port: 12345
    Destination Port: 35876
    [Stream index: 0]
    [Conversation completeness: Complete, WITH_DATA (31)]
    [TCP Segment Len: 9]
    Sequence Number: 53568007
    [Next Sequence Number: 53568016]
    Acknowledgment Number: 3178881820
    0101 .... = Header Length: 20 bytes (5)
  + Flags: 0x018 (PSH, ACK)
    Window: 65535
    [Calculated window size: 65535]
    [Window size scaling factor: -2 (no window scaling used)]
    Checksum: 0x259d [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  + [Timestamps]
  + [SEQ/ACK analysis]
    TCP payload (9 bytes)
Data (9 bytes)
    Data: 323131343337323233
```

```
0000  08 00 27 34 88 8a 52 54  00 12 35 02 08 00 45 00    ··'4··RT ··5···E·
0010  00 31 10 7b 00 00 40 06  9e 21 ac 1f 13 fd 0a 00    ·1·{··@· ·!······
0020  02 0f 30 39 8c 24 03 31  62 07 bd 79 e3 1c 50 18    ··09·$·1 b··y··P·
0030  ff ff 25 9d 00 00 32 31  31 34 33 37 32 32 33       ··%···21 1437223
```

השרת מחזיר ללקוח הודעה עם מספר ת.ז. מספרי ack/seq ללא שינוי.

```
Transmission Control Protocol, Src Port: 12345, Dst Port: 35876, Seq: 53568016, Ack: 3178881820, Len: 0
    Source Port: 12345
    Destination Port: 35876
    [Stream index: 0]
    [Conversation completeness: Complete, WITH_DATA (31)]
    [TCP Segment Len: 0]
    Sequence Number: 53568016
    [Next Sequence Number: 53568017]
    Acknowledgment Number: 3178881820
    0101 .... = Header Length: 20 bytes (5)
  + Flags: 0x011 (FIN, ACK)
    Window: 65535
    [Calculated window size: 65535]
    [Window size scaling factor: -2 (no window scaling used)]
    Checksum: 0x2173 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  + [Timestamps]
```

הגענו לשלב סיום התקשורת. השרת שולח ללקוח הודעת FIN,ACK. כלומר, השרת מודיע שהוא מעוניין לסיים את התקשורת עם הלקוח.
מספר הseq גדל ב9 שכן השרת שלח את הת.ז בשלב קודם לכן. (כעת עומד על 53568016).

מספר הack נותר ללא שינוי שכן הלקוח לא שלח עוד הודעות אל השרת.

```
Transmission Control Protocol, Src Port: 35876, Dst Port: 12345, Seq: 3178881820, Ack: 53568017, Len: 0
    Source Port: 35876
    Destination Port: 12345
    [Stream index: 0]
    [Conversation completeness: Complete, WITH_DATA (31)]
    [TCP Segment Len: 0]
    Sequence Number: 3178881820
    [Next Sequence Number: 3178881821]
    Acknowledgment Number: 53568017
    0101 .... = Header Length: 20 bytes (5)
  + Flags: 0x011 (FIN, ACK)
```

הלקוח שולח לשרת הודעת FIN,ACK למטרת סיום התקשורת ביניהם. בנוסף, ניתן לראות כי מספר הack של הלקוח גדל ב-10(!) וכעת הוא 53568017. זה מכיוון שהלקוח מאשר גם את קבלת מספר ת.ז בגודל 9 וגם מוסיף 1 נוסף שמסמן את אישור הFIN.

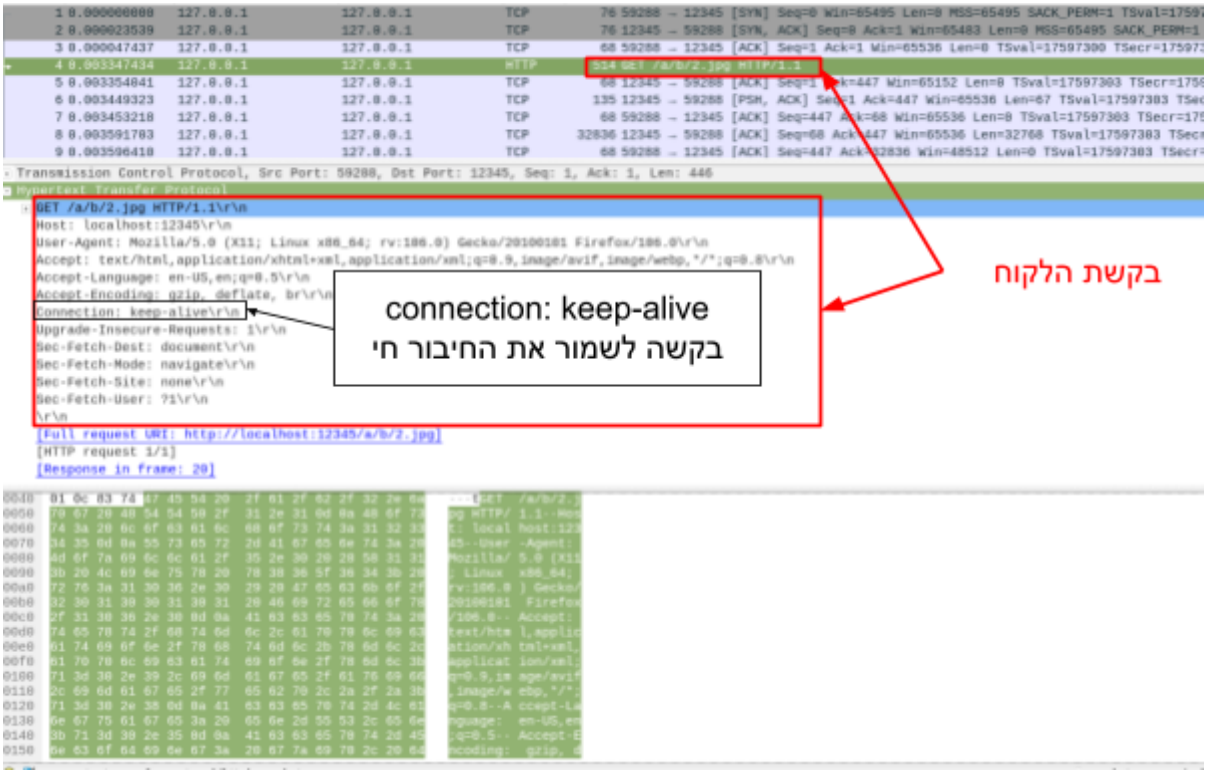
```
Transmission Control Protocol, Src Port: 12345, Dst Port: 35876, Seq: 53568017, Ack: 3178881821, Len: 0
    Source Port: 12345
    Destination Port: 35876
    [Stream index: 0]
    [Conversation completeness: Complete, WITH_DATA (31)]
    [TCP Segment Len: 0]
    Sequence Number: 53568017
    [Next Sequence Number: 53568017]
    Acknowledgment Number: 3178881821
    0101 .... = Header Length: 20 bytes (5)
  + Flags: 0x010 (ACK)
```

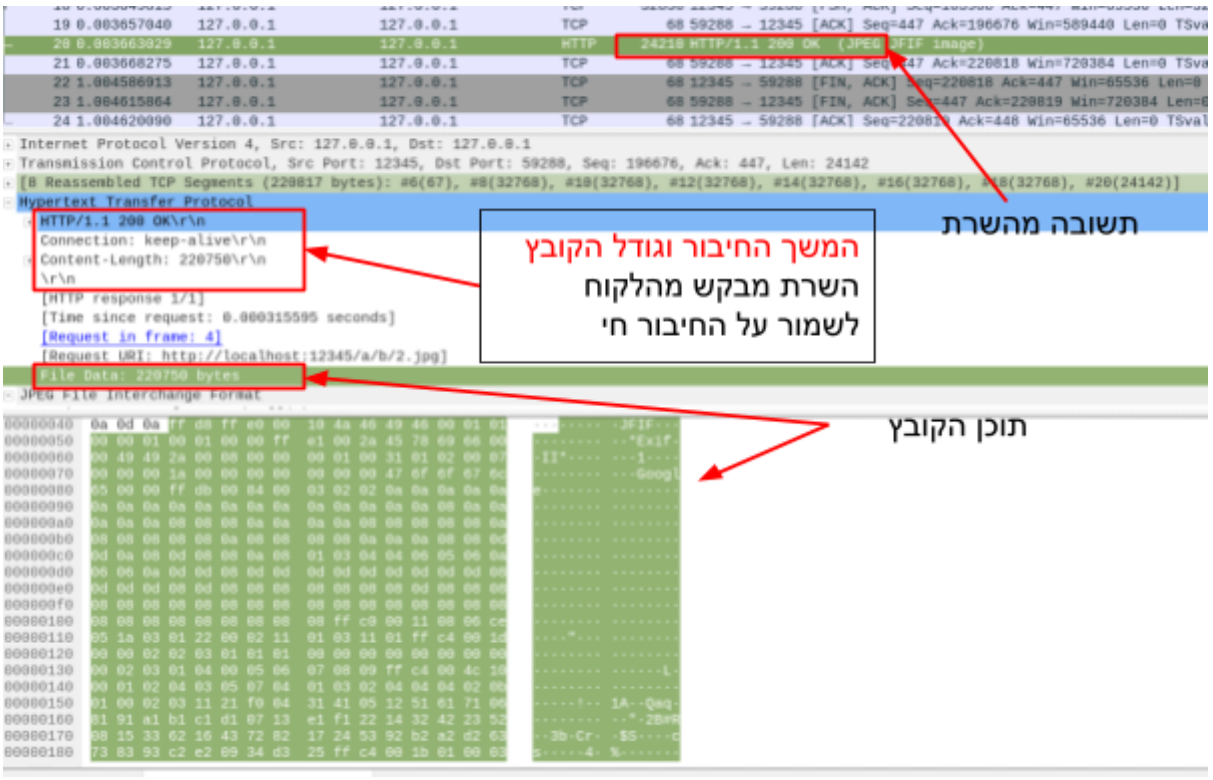
לבסוף, השרת מאשר את בקשת סיום התקשורת מהלקוח עם ACK ועם מספר ack גדול ב-1: 3178881821. מספר הseq הוא 53568017 כצפוי.

<u>חלק ב':</u>
סה"כ היו 10 חיבורים. החיבורים היו מול מספרי הפורט הבאים: 59288, 43892, 34976, 34646(הקלדה), 34662, 34678, 34680, 34686, 34688, 34692.
התעבורה מופיעה בקובץ PartB_withRef.pcapng

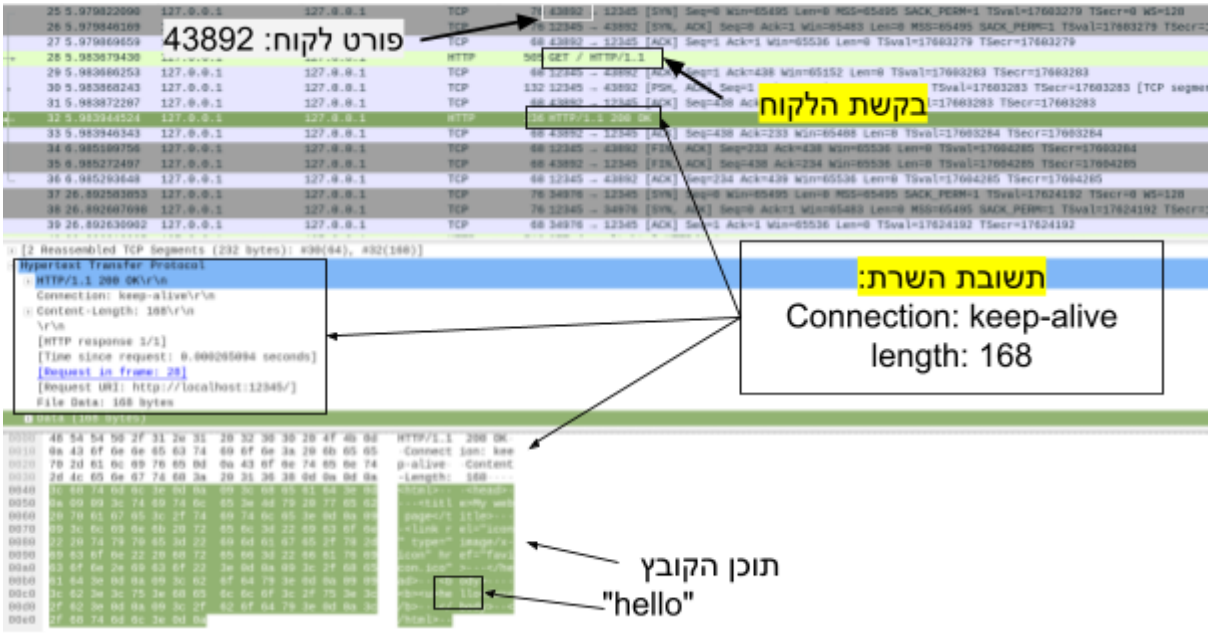
נתבונן בחיבור הראשון (59288): (שורות 1 - 24)

הייתה בקשה אחת מהלקוח לקבל את התמונה jpg.2 בנתיב a/b.
כשהוזן הקישור בדפדפן נשלחה לשרת הודעת HTTP עם תוכן הבקשה (שורה 4) (כמובן לאחר "לחיצת הידיים" בין הלקוח לשרת), והשרת, לאחר ששלח את המידע שביקש הלקוח, החזיר הודעת HTTP עם התוכן (שורה 20).
במקרה זה הדפדפן והשרת שולחים FIN אחד לשני ומסיימים את התקשורת ביניהם.

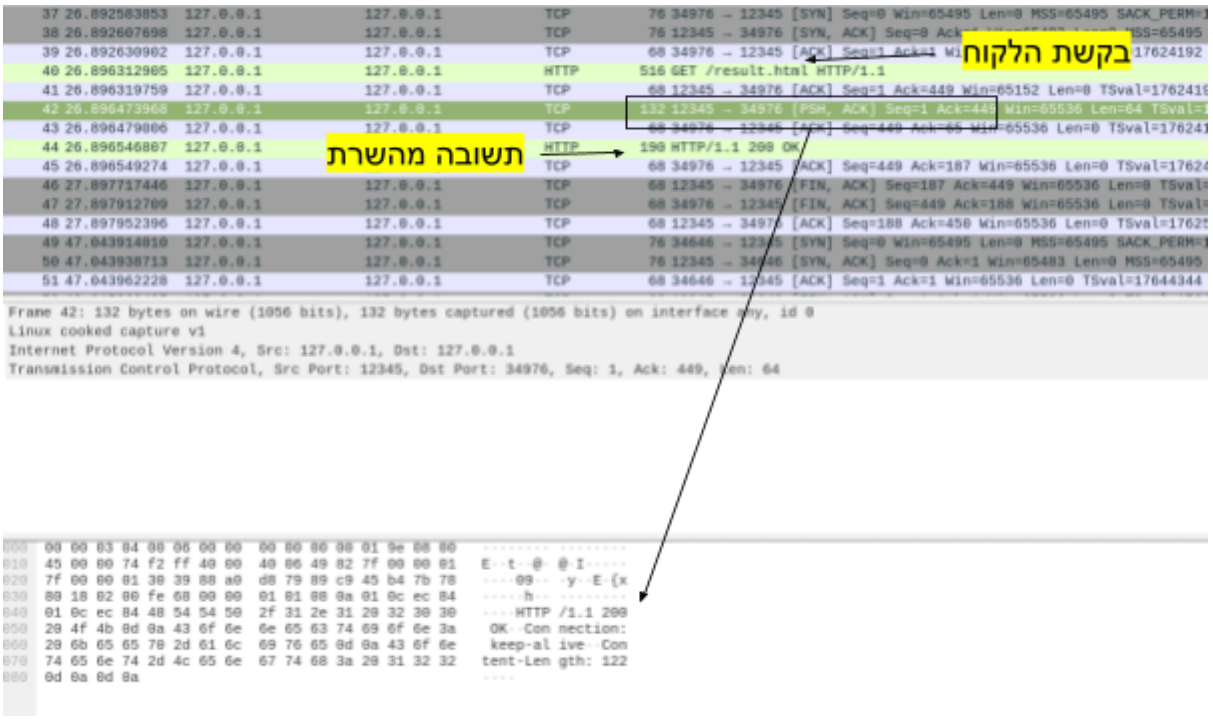



תשובה מהשרת
המשך החיבור וגודל הקובץ
השרת מבקש מהלקוח
לשמור על החיבור חי
תוכן הקובץ

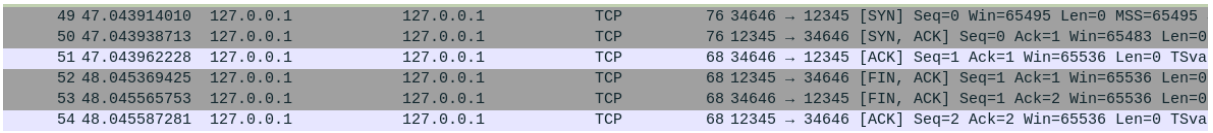
חיבור שני: (43892) (שורות 25 - 36)
בחיבור זה נשלחה הבקשה / (index.html). ניתן לראות את תוכן הקובץ שהשרת מחזיר ללקוח. השרת מבקש לשמור את החיבור חי. תוכן ההודעה הוא מכיל את המחזורת "hello".



חיבור שלישי (34976): (שורות 37 - 48)
בחיבור זה נשלחה הבקשה /result.html. ניתן לראות את פרטי החיבור שהשרת מחזיר ללקוח לאחר יצירת החיבור.



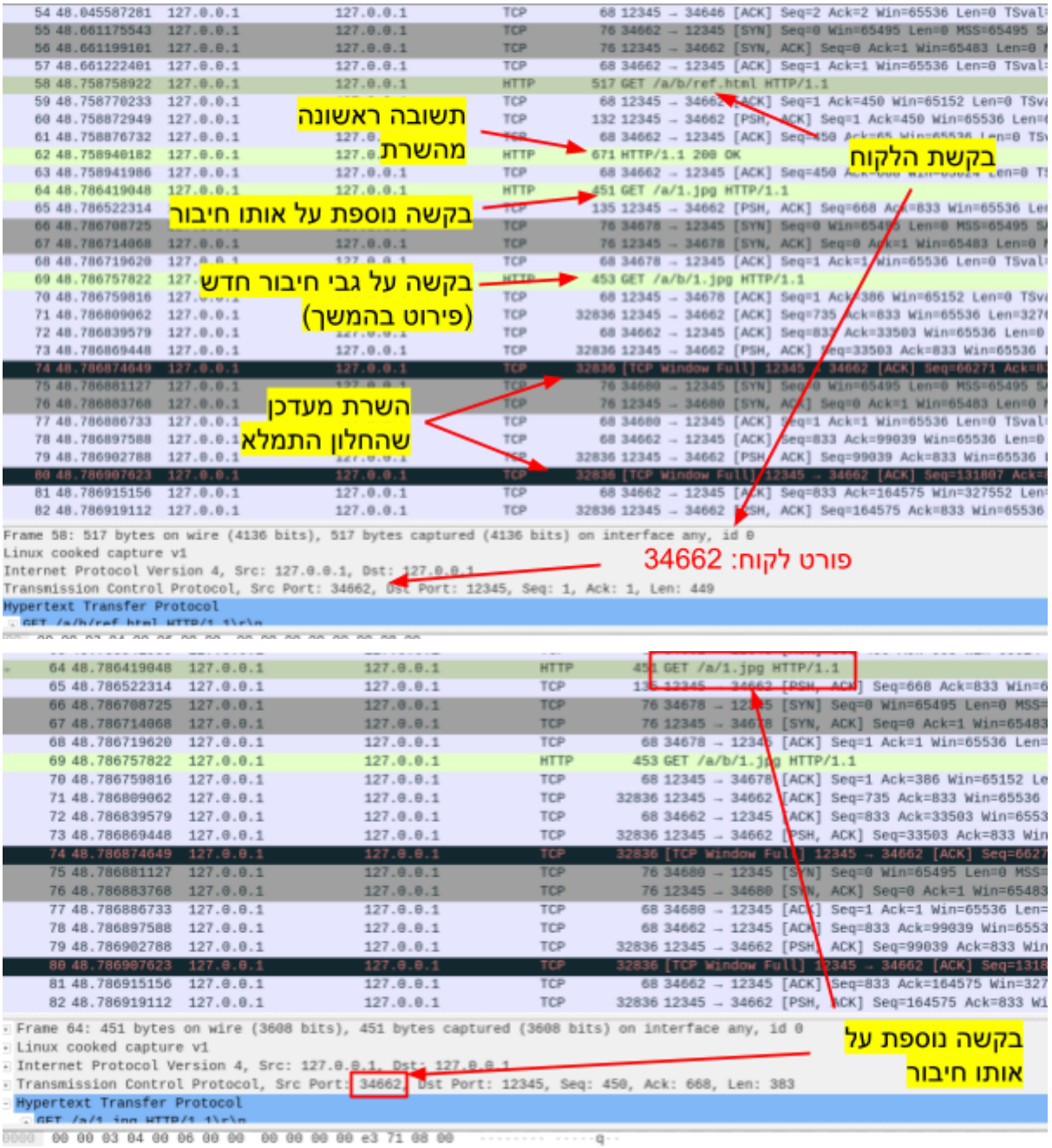
חיבור רביעי (34646(הקלדה)): (שורות 49 - 54)



בfirefox כל הקלדה בשורת הURL יוצרת חיבור עם השרת. כאן באה לידי ביטוי ההקלדה של הבקשה הבאה.

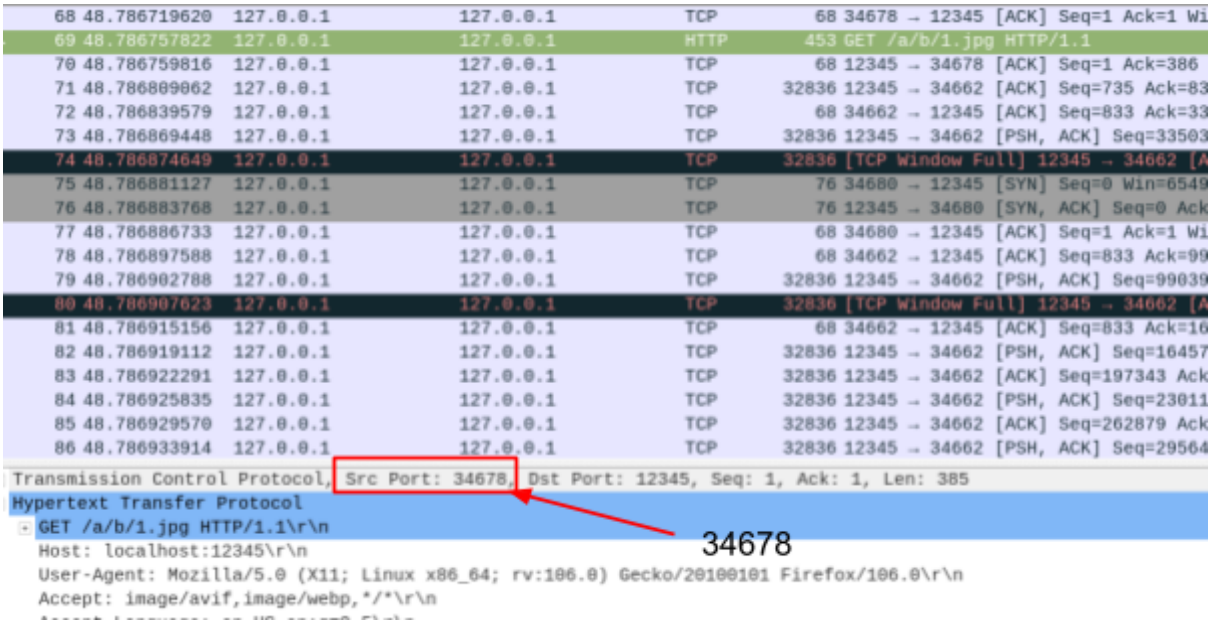
חיבור 5 - 10 (34662, 34678, 34680, 34686, 34688, 34692): משורה 55 עד הסוף. עבור הבקשה a/b/ref.html

הפעם נשלחת יותר מבקשה אחת על אותו חיבור, למשל אחרי ש - a/b/ref.html נשלחת, גם , a/1.jpg a/3.jpg נשלחות מאותו לקוח (הדוגמאות הנ"ל).

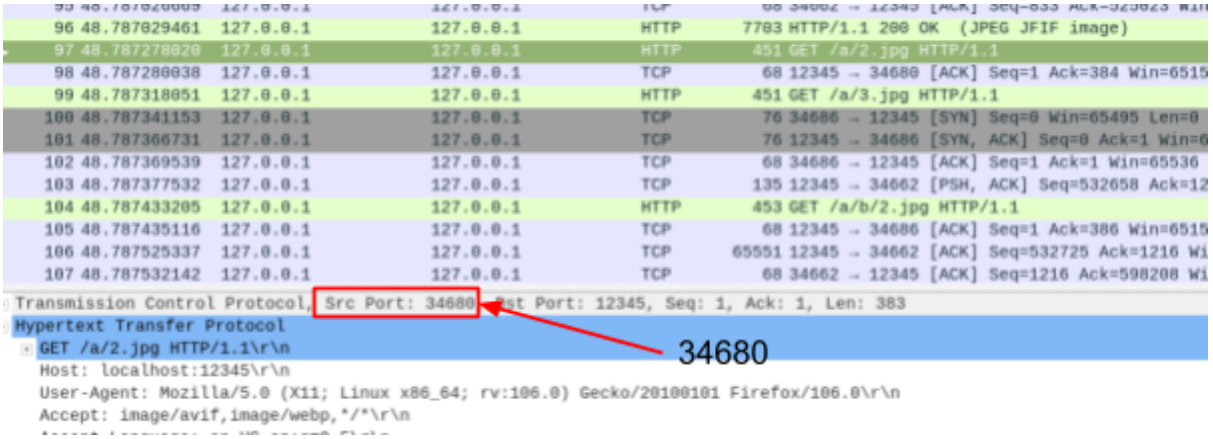


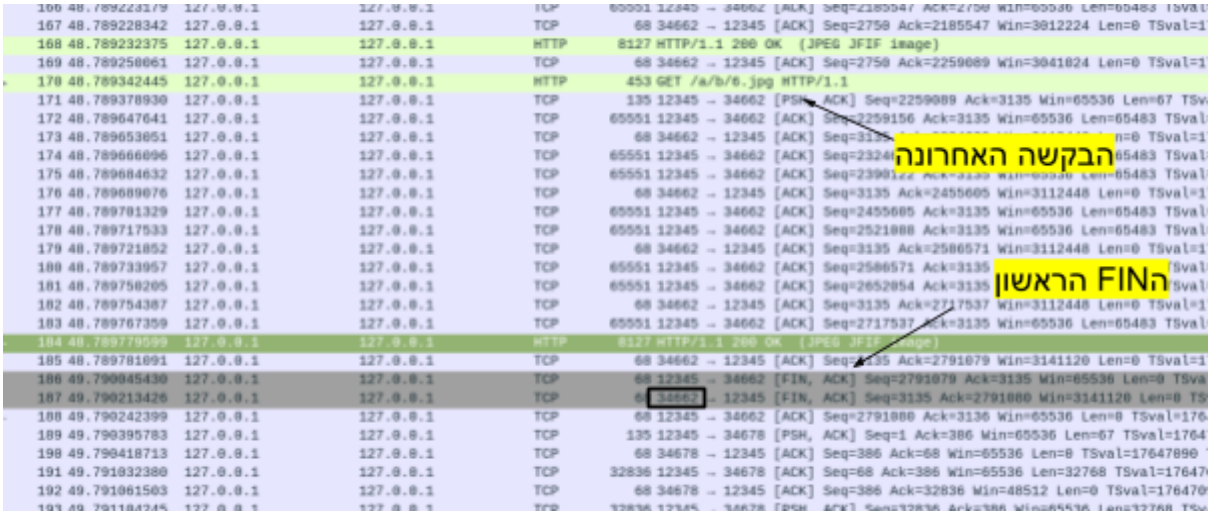
גם הבקשות עבור a/b/6.jpg , a/6.jpg , a/b/5.jpg , a/5.jpg , a/4.jpg נשלחות על גבי אותו החיבור הנ"ל.

לעומת זאת הבקשה עבור /a/b/1.jpg שנשלחת מפורט אחר. כלומר, חיבור חדש.



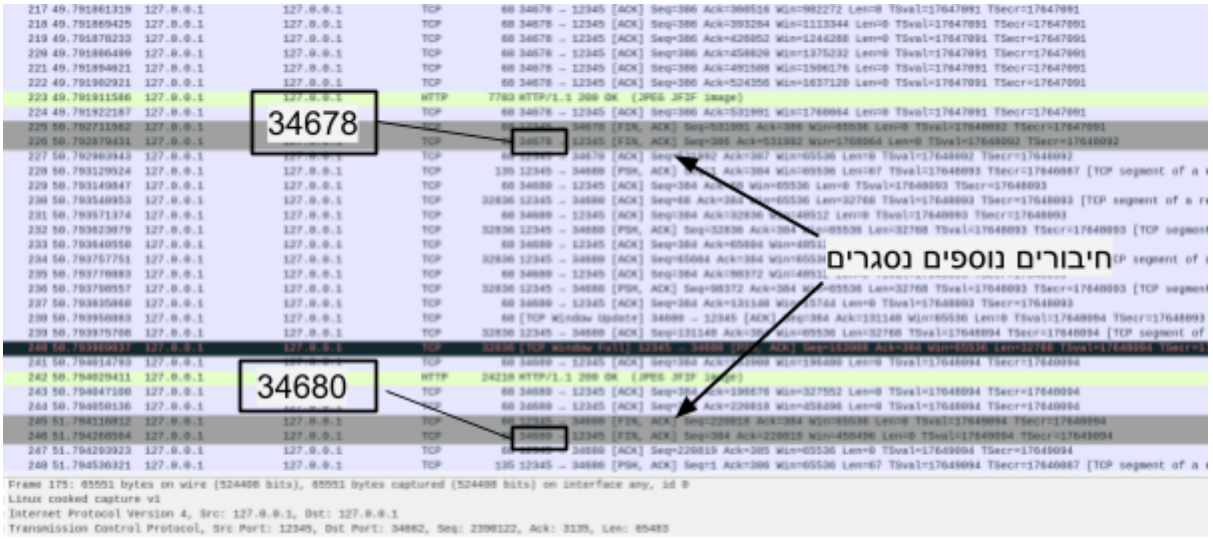
כנל לגבי הבקשה עבור /a/2.jpg. כך גם כל הבקשות: a/b/4.jpg , a/b/3.jpg, a/b/2.jpg נשלחות על פורט נפרד עבור כל בקשה.



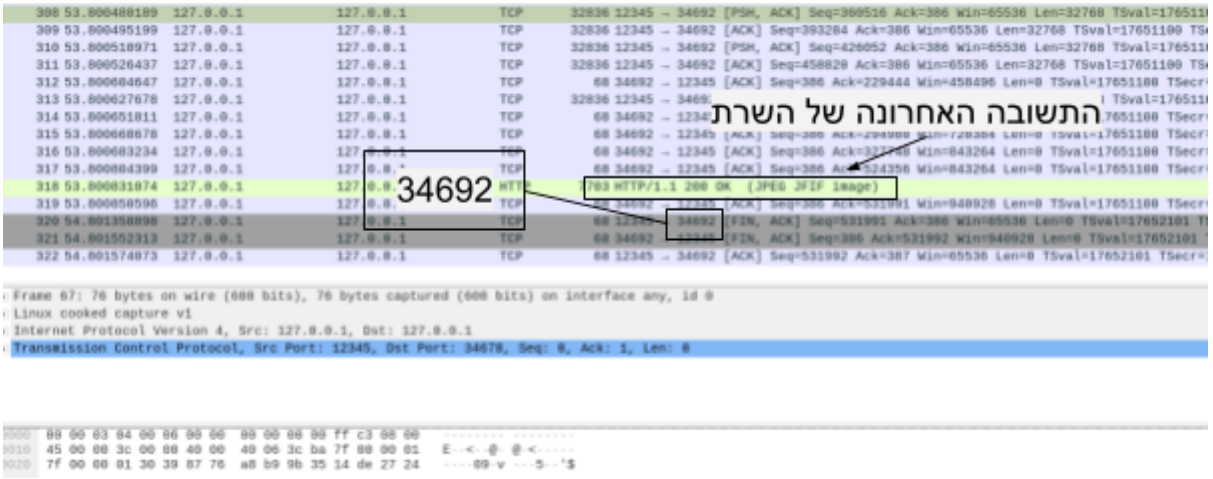


כעת לאחר שהלקוח שלח את כל הבקשות מתחילים להיסגר כל החיבורים. כאן ניתן לראות את החיבור הראשון שנסגר, שהוא גם החיבור הראשון שנפתח עבור הבקשה הראשונית a/b/ref.html. כפי שצוין מקודם, הבקשה a/b/6.jpg יושבת על החיבור עם פורט 34662, אותו חיבור ראשוני שעכשיו נסגר.

בהמשך לאחר שנשלח מידע נוסף (מידע בינארי של התמונות שמוצגות) נסגרים חיבורים נוספים.



עד שלבסוף נסגר החיבור האחרון (34692).



דוגמא נוספת לקובץ שלא קיים (נמצא בקובץ PartB_close_example.pcapng):