

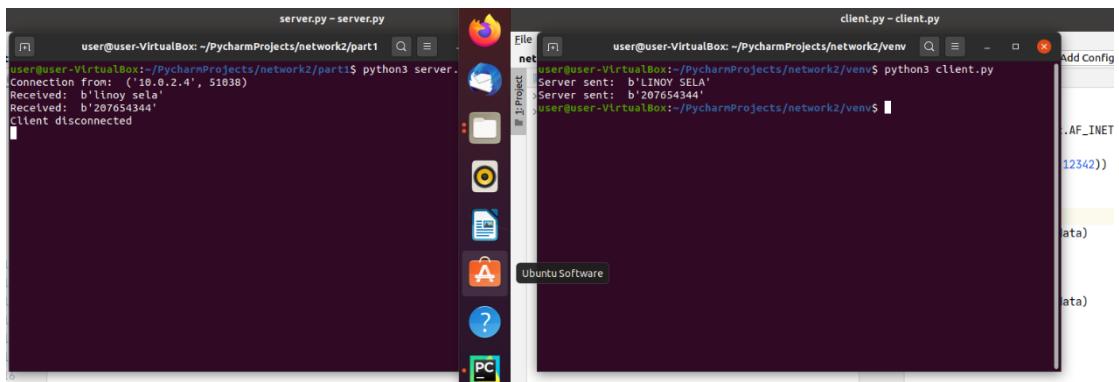
## רשתות תרגיל 2 – חלק 1

### חלק א'

ראשית נריץ את השרת והלקוח על מנת לקבל את השם באותיות גדולות ואת התיעודת הזרחות, בלקוח.

הפעלנו 2 מכונות כאשר מצאנו את הקו של השרת עם הפקודה ifconfig, וגילנו שכטובותינו היא 10.0.2.15, אך בלקוח כאשר ניצור socket, נרצה להגיא ל ko של השרת. כתובות הקו של הלוקוח היא 10.0.2.4 ונינתן לאלוות זאת גם ע"י הדפסת השרת שמדפיס Maipegaagi העיגן הבקשה. הport שהלקוח מעוניין להאזין לו הוא 12342.

ניתן לראות כי התוכנית אכן עבדת כראוי:

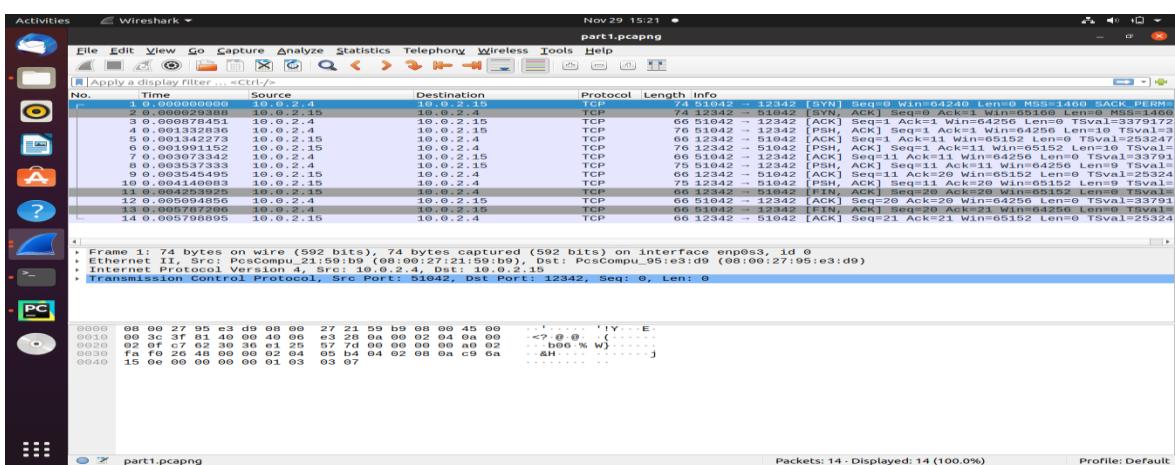


נירץ את התוכנית שוב וננטח את הנתונים:

נפעיל whreshark ונסנן את החיפוש ע"י כתובות הקו של המקור ושל היעד ע"ז:

((ip.dst == 10.0.2.4) && (ip.src==10.0.2.15)) || ((ip.src == 10.0.2.4) && (ip.dst==10.0.2.15))

סינון זה שמור בpart1.

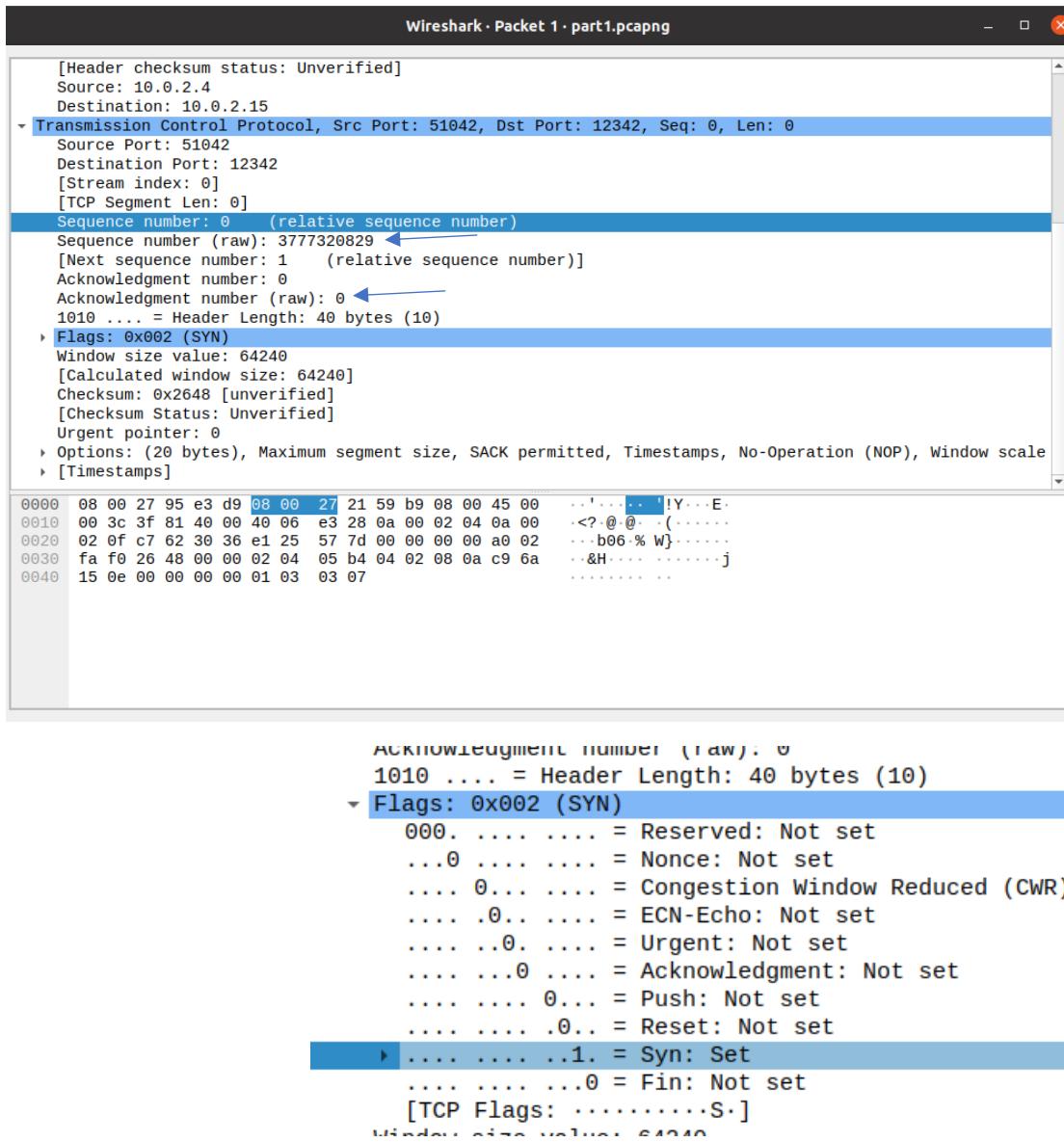


מכיוון שאנחנו עובדים עם פרוטוקול TCP, ראשית יש לבצע חיבור בין 2 המכונות, השלב הראשון הוא שלב לחיצת ידיים המתבטא בהקמת החיבור שנראה זאת ע"י הדגל SYN, יחד

עם מספר סידורי, היעד ישלח גם SYN יחד עם ACK , כלומר אישר את הקמת החיבור מבחינתו – לצורך סyncron, ושלח גם את ack number יחד עם מספר סידורי.

נראת זאת בwireshark:

בחבילה הראשונה של התקשרות, port המקור הוא 51042, כלומר הלוקה, ווק השרת הוא 12342, כלומר השרת. תחת שכבת התעבורה ניתן לראות כי דגל SYN דלוק, המספר הסידורי (נסתכל על האמיטיים ולא היחס) הוא 3777320829 והא ack number גם 0.



בחבילה ההבא, ניתן לראות לדוגמה על פי שכבת הרשת כי זו המקור הוא של השרת 10.0.2.15, כלומר הוא מшиб כעת לביקשת הלוקה ליצור חיבור, נראת זאת בפירוט בשכבת התעבורה. כעת ניתן לראות כי הדגלים הדלוקים הם SYN על הרצון לסינכרון הקמת החיבור מצד השרת, ודגל ACK על מנת לאשר את החבילה הקודמת. ניתן לראות כי המספר הסידורי של השרת הוא 3972403749, בחר אותו אקראי, אך ACK הוא 3777320830 מכיוון 3777320830 + 1 עבור ACK – יחד שמספר הסידורי של הלוקה היה 3777320829.

נראת זאת:

Wireshark - Packet 2 - part1.pcapng

Frame 2: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface enp0s3, id 0
   
Ethernet II, Src: PcsCompu\_95:e3:d9 (08:00:27:95:e3:d9), Dst: PcsCompu\_21:59:b9 (08:00:27:21:59:b9)
   
Internet Protocol Version 4, Src: 10.0.2.15, Dst: 10.0.2.4
   
... 0100 .... = Version: 4
   
.... 0101 = Header Length: 20 bytes (5)
   
Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
   
Total Length: 60
   
Identification: 0x0000 (0)
   
Flags: 0x4000, Don't fragment
   
Fragment offset: 0
   
Time to live: 64
   
Protocol: TCP (6)
   
Header checksum: 0x22aa [validation disabled]
   
[Header checksum status: Unverified]
   
Source: 10.0.2.15
   
Destination: 10.0.2.4

0000 08 00 27 21 59 b9 08 00 27 95 e3 d9 08 00 45 00 ... !Y... '....E.
 0010 00 3c 00 00 40 00 40 06 22 aa 0a 00 02 0f 0a 00 <.. @@.. ".....
 0020 02 04 30 36 c7 62 ec c6 12 25 e1 25 57 7e a0 12 ..06.b.. %W...
 0030 fe 88 18 41 00 00 02 04 05 b4 04 02 08 0a 96 f2 ...A.... .....
 0040 6f 32 c9 6a 15 0e 01 03 03 07 02.j.... .

Wireshark - Packet 2 - part1.pcapng

Internet Protocol Version 4, Src: 10.0.2.15, Dst: 10.0.2.4
   
Transmission Control Protocol, Src Port: 12342, Dst Port: 51042, Seq: 0, Ack: 1, Len: 0
   
Source Port: 12342
   
Destination Port: 51042
   
[Stream index: 0]
   
[TCP Segment Len: 0]
   
Sequence number: 0 (relative sequence number)
   
Sequence number (raw): 3972403749
   
[Next sequence number: 1 (relative sequence number)]
   
Acknowledgment number: 1 (relative ack number)
   
Acknowledgment number (raw): 3777320830
   
1010 .... = Header Length: 40 bytes (10)
   
Flags: 0x012 (SYN, ACK)
   
Window size value: 65160
   
[Calculated window size: 65160]

0000 08 00 27 21 59 b9 08 00 27 95 e3 d9 08 00 45 00 ... !Y... '....E.
 0010 00 3c 00 00 40 00 40 06 22 aa 0a 00 02 0f 0a 00 <.. @@.. ".....
 0020 02 04 30 36 c7 62 ec c6 12 25 e1 25 57 7e a0 12 ..06.b.. %W...
 0030 fe 88 18 41 00 00 02 04 05 b4 04 02 08 0a 96 f2 ...A.... .....
 0040 6f 32 c9 6a 15 0e 01 03 03 07 02.j.... .

החבילה הhabה היא שליחת ack על ידי הלקוט, מאשר את זאת שהחיבור קם וקיים. לכן רואים כי ack הוא 3972403750 (מכיון שהמספר הסידורי היה 1 + 3972403749 ל-ACK), והמספר הסידורי הפק להיות 3777320830 , בהתאם ל-ACK הקודם שנשלח והיה זהה לכך, הוא לא גדול כי לא נסוף מידע.

Wireshark - Packet 3 - part1.pcapng

Frame 3: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface enp0s3, id 0
   
Ethernet II, Src: PcsCompu\_21:59:b9 (08:00:27:21:59:b9), Dst: PcsCompu\_95:e3:d9 (08:00:27:95:e3:d9)
   
Internet Protocol Version 4, Src: 10.0.2.4, Dst: 10.0.2.15
   
Transmission Control Protocol, Src Port: 51042, Dst Port: 12342, Seq: 1, Ack: 1, Len: 0
   
Source Port: 51042
   
Destination Port: 12342
   
[Stream index: 0]
   
[TCP Segment Len: 0]
   
Sequence number: 1 (relative sequence number)
   
Sequence number (raw): 3777320830
   
[Next sequence number: 1 (relative sequence number)]
   
Acknowledgment number: 1 (relative ack number)
   
Acknowledgment number (raw): 3972403750
   
1000 .... = Header Length: 32 bytes (8)
   
Flags: 0x010 (ACK)

0000 08 00 27 95 e3 d9 08 00 27 21 59 b9 08 00 45 00 ... !Y... '....E.
 0010 00 34 3f 82 40 00 40 06 e3 2f 0a 00 02 04 0a 00 .4? @@.. /.....
 0020 02 0f c7 62 30 36 e1 25 57 7e ec c6 12 26 80 10 ..b06.%W...&..
 0030 01 f6 48 ec 00 00 01 01 08 0a c9 6a 15 0f 96 f2 ...H.... j.....
 0040 6f 32 02.j.... .

עד כאן עברו שלב הקמת החיבור, כעת נעברו לשלב מימוש החיבור, כלומר שליחת מידע. לפי התוכנית, ראשית הלקוח שולח לשרת את השם `sela` וכן ניתן לראות זאת ב-data:

Wireshark - Packet 4 · part1.pcapng

```

1000 .... = Header Length: 32 bytes (8)
  Flags: 0x018 (PSH, ACK)
  Window size value: 502
  [Calculated window size: 64256]
  [Window size scaling factor: 128]
  Checksum: 0x1519 [unverified]
  [Checksum Status: Unverified]
  Urgent pointer: 0
  Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
  [SEQ/ACK analysis]
  [Timestamps]
  TCP payload (10 bytes)
    Data (10 bytes)
      Data: 0c696e6f792073656c61
      [Length: 10]

0000  08 00 27 95 e3 d9 08 00  27 21 59 b9 08 00 45 00  .'. . . !. E.
0010  00 3e 3f 83 40 00 40 06  e3 24 0a 00 02 04 0a 00  .>? @ @. $.....
0020  02 0f c7 62 30 36 e1 25  57 7e ec c6 12 26 80 18  ...b06.% W---&...
0030  01 f6 15 19 00 00 01 01  08 0a c9 6a 15 10 96 f2  ..... j.....
0040  6f 32 6c 69 6e 6f 79 20  73 65 6c 61  o2linoy sela

```

בשכבה התעבורת של חיבור זה ניתן לראות כי ack שווה ל-3972403750, לא התקבל מידע חדש لكن נשאר כר, וכן גם המספר הסידורי שווה ל-3777320830, מכיוון שהוא שולח מידע "מהמקום" זהה והלאה.

(אם נסתכל על ACK היחס, ניתן לראות כי next seq number הוא 11, וזאת מכיוון שהוספנו את כמות הביטים של השם לינוי סלה למספר הסידורי שאנו נמצא עליו (1), לעומת דיווח על כר שבפעם הבאיה אני פניו מ-11. ניתן גם לראות בחז הכלול כי גודל המידע שנשלח הוא 10, וכן ככמות האותיות שבסמ"ט ייחד עם רוח.)

Wireshark - Packet 4 · part1.pcapng

```

Frame 4: 76 bytes on wire (608 bits), 76 bytes captured (608 bits) on interface enp0s3, id 0
  Ethernet II, Src: PcsCompu_21:59:b9 (08:00:27:21:59:b9), Dst: PcsCompu_95:e3:d9 (08:00:27:95:e3:d9)
  Internet Protocol Version 4, Src: 10.0.2.4, Dst: 10.0.2.15
  Transmission Control Protocol, Src Port: 51042, Dst Port: 12342, Seq: 1, Ack: 1, Len: 10
    Source Port: 51042
    Destination Port: 12342
    [Stream index: 0]
    [TCP Segment Len: 10]
    Sequence number: 1 (relative sequence number)
    Sequence number (raw): 3777320830
    [Next sequence number: 11 (relative sequence number)]
    Acknowledgment number: 1 (relative ack number)
    Acknowledgment number (raw): 3972403750
    1000 .... = Header Length: 32 bytes (8)
    Flags: 0x018 (PSH, ACK)
    Window size value: 502

0000  08 00 27 95 e3 d9 08 00  27 21 59 b9 08 00 45 00  .'. . . !. E.
0010  00 3e 3f 83 40 00 40 06  e3 24 0a 00 02 04 0a 00  .>? @ @. $.....
0020  02 0f c7 62 30 36 e1 25  57 7e ec c6 12 26 80 18  ...b06.% W---&...
0030  01 f6 15 19 00 00 01 01  08 0a c9 6a 15 10 96 f2  ..... j.....
0040  6f 32 6c 69 6e 6f 79 20  73 65 6c 61  o2linoy sela

```

בחבילת הבאיה, נצפה שהשרת יתן על אישור על כר שקיבל את החבילת הקודמת עם השם, ככלומר ישלח עליה ACK, לפי חישובנו עליו להיות המספר הסידורי הקודם + 3777320830 גודל המידע - פלאו 10 תווים של השם. סה"כ 3777320840, נראה אם זה מתקיים:

Wireshark - Packet 5 · part1.pcapng

```

Frame 5: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface enp0s3, id 0
Ethernet II, Src: PcsCompu_95:e3:d9 (08:00:27:95:e3:d9), Dst: PcsCompu_21:59:b9 (08:00:27:21:59:b9)
Internet Protocol Version 4, Src: 10.0.2.15, Dst: 10.0.2.4
Transmission Control Protocol, Src Port: 12342, Dst Port: 51042, Seq: 1, Ack: 11, Len: 0
    Source Port: 12342
    Destination Port: 51042
    [Stream index: 0]
    [TCP Segment Len: 0]
        Sequence number: 1 (relative sequence number)
        Sequence number (raw): 3972403750
        [Next sequence number: 1 (relative sequence number)]
        Acknowledgment number: 11 (relative ack number)
        Acknowledgment number (raw): 3777320840
        1000 .... = Header Length: 32 bytes (8)
        > Flags: 0x010 (ACK)
        Window size value: 500
0000  08 00 27 21 59 b9 08 00 27 95 e3 d9 08 00 45 00  .. !Y... '....E.
0010  00 34 c2 7c 40 00 40 06 60 35 0a 00 02 0f 0a 00  .4.|@. `5.....
0020  02 04 30 36 c7 62 ec c6 12 26 e1 25 57 88 80 10  ..06.b... &%W...
0030  01 fd 18 39 00 00 01 01 08 0a 96 f2 6f 34 c9 6a  ..9..... o4.j
0040  15 10

```

אכן ניתן לראות כי חבילת זו מגיעה מהשרת ע"פ port השולח שהוא 12342, אכן דגל ACK דלוק והוא שולח ACK הקודם. זהה ל-ACK הקיים. בעוד, לאחר שהשרת אישר את והמספר הסידורי הוא 3972403750, ישלוח פעולה והיא להזכיר את השם באותיות גדולות. לכן נצפה לקבלת השם, לפי הקוד עליון בצע פעולה והוא LINOY SELA. נראה זאת:

Wireshark - Packet 6 · part1.pcapng

```

Frame 6: 76 bytes on wire (608 bits), 76 bytes captured (608 bits) on interface enp0s3, id 0
Ethernet II, Src: PcsCompu_95:e3:d9 (08:00:27:95:e3:d9), Dst: PcsCompu_21:59:b9 (08:00:27:21:59:b9)
Internet Protocol Version 4, Src: 10.0.2.15, Dst: 10.0.2.4
Transmission Control Protocol, Src Port: 12342, Dst Port: 51042, Seq: 1, Ack: 11, Len: 10
    Source Port: 12342
    Destination Port: 51042
    [Stream index: 0]
    [TCP Segment Len: 10]
        Sequence number: 1 (relative sequence number)
        Sequence number (raw): 3972403750
        [Next sequence number: 11 (relative sequence number)]
        Acknowledgment number: 11 (relative ack number)
        Acknowledgment number (raw): 3777320840
        1000 .... = Header Length: 32 bytes (8)
        > Flags: 0x018 (PSH, ACK)
        Window size value: 500
0000  08 00 27 21 59 b9 08 00 27 95 e3 d9 08 00 45 00  .. !Y... '....E.
0010  00 3e c2 7d 40 00 40 06 60 2a 0a 00 02 0f 0a 00  .>.)@. `*.....
0020  02 04 30 36 c7 62 ec c6 12 26 e1 25 57 88 80 18  ..06.b... &%W...
0030  01 fd 18 43 00 00 01 01 08 0a 96 f2 6f 34 c9 6a  ..C..... o4.j
0040  15 10 4c 49 4e 4f 59 20 53 45 4c 41  ..LINOY SELA

```

Wireshark - Packet 6 · part1.pcapng

```

1000 .... = Header Length: 32 bytes (8)
  ▶ Flags: 0x018 (PSH, ACK)
  Window size value: 509
  [Calculated window size: 65152]
  [Window size scaling factor: 128]
  Checksum: 0x1843 [unverified]
  [Checksum Status: Unverified]
  Urgent pointer: 0
  ▶ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
  ▶ [SEQ/ACK analysis]
  ▶ [Timestamps]
  TCP payload (10 bytes)
  - Data (10 bytes)
    Data: 4c494e4f592053454c41
    [Length: 10]

0000  08 00 27 21 59 b9 08 00  27 95 e3 d9 08 00 45 00  ..'Y.. '....E.
0010  00 3e c2 7d 40 00 40 06  60 2a 0a 00 02 0f 0a 00  .-> @. @. *.....
0020  02 04 30 36 c7 62 ec c6  12 26 e1 25 57 88 80 18  ..06.b... &%W...
0030  01 fd 18 43 00 00 01 01  08 0a 96 f2 6f 34 c9 6a  ..C.....o4.j
0040  15 10 4c 49 4e 4f 59 20  53 45 4c 41  ..LINOY SELA

```

אכן השרת שולח מידע ללקוח, שהוא SELA, המיר את השם לאותיות גדולות, מידע באורך 10 תוים=בתים. המספר הסידורי שלו הוא 3972403750 והאCK נושא 3777320840 כי הוא לא קיבל מידע חדש.

לאחר מכן נצפה שתגייע חביבה מהלקוח, עליו לקבל מידע זה ולאשרו, וכן רואים כי הוא שולח ACK 3972403750 כי המספר הסידורי היה 0 + 10 על המידע החדש, את השם באותיות גדולות, סהכ 3972403760, והמספר הסידורי שלו 3777320840 בדומה לACK הקודם(לא שולח מידע חדש- נשאר זהה).

Wireshark - Packet 7 · part1.pcapng

```

Frame 7: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface enp0s3, id 0
  ▶ Ethernet II, Src: PcsCompu_21:59:b9 (08:00:27:21:59:b9), Dst: PcsCompu_95:e3:d9 (08:00:27:95:e3:d9)
  ▶ Internet Protocol Version 4, Src: 10.0.2.4, Dst: 10.0.2.15
  ▶ Transmission Control Protocol, Src Port: 51042, Dst Port: 12342, Seq: 11, Ack: 11, Len: 0
    Source Port: 51042
    Destination Port: 12342
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 11 (relative sequence number)
    Sequence number (raw): 3777320840
    [Next sequence number: 11 (relative sequence number)]
    Acknowledgment number: 11 (relative ack number)
    Acknowledgment number (raw): 3972403760
    1000 .... = Header Length: 32 bytes (8)
    ▶ Flags: 0x010 (ACK)
    Window size value: 509

0000  08 00 27 95 e3 d9 08 00  27 21 59 b9 08 00 45 00  ..'.... 'Y.. E.
0010  00 34 3f 84 40 00 40 06  e3 2d 0a 00 02 04 0a 00  .?@. @. #.....
0020  02 0f c7 62 30 36 e1 25  57 88 ec c6 12 30 80 18  ..b06.%W...0...
0030  01 f6 48 d4 00 00 01 01  08 0a c9 6a 15 11 96 f2  ..B.....j....
0040  6f 34 32 30 37 36 35 34  33 34 34 04207654 344

```

כעת, יתבצע אותו תהליך שתיארנו בעבר שליחת שם ללקוח וקבלתו באותיות גדולות, הפעם הלוקוח ישלח ת"ז ויקבלו בחזרה.

Wireshark - Packet 8 · part1.pcapng

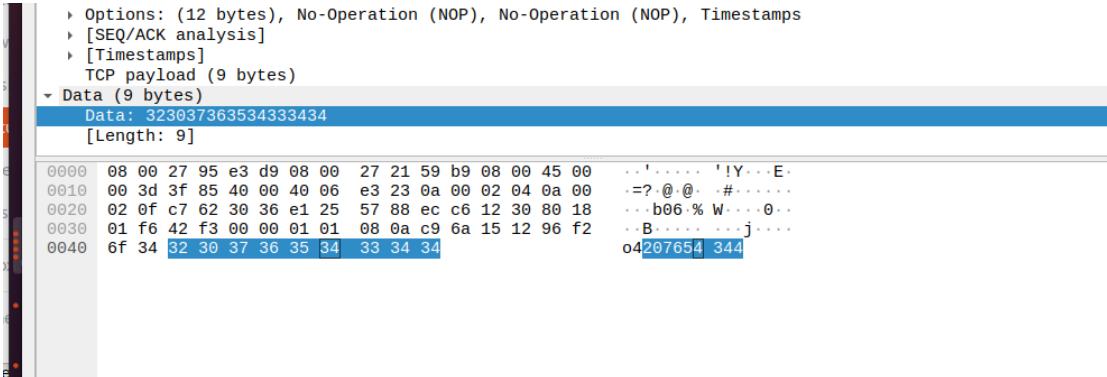
```

Frame 8: 75 bytes on wire (600 bits), 75 bytes captured (600 bits) on interface enp0s3, id 0
  ▶ Ethernet II, Src: PcsCompu_21:59:b9 (08:00:27:21:59:b9), Dst: PcsCompu_95:e3:d9 (08:00:27:95:e3:d9)
  ▶ Internet Protocol Version 4, Src: 10.0.2.4, Dst: 10.0.2.15
  ▶ Transmission Control Protocol, Src Port: 51042, Dst Port: 12342, Seq: 11, Ack: 11, Len: 9
    Source Port: 51042
    Destination Port: 12342
    [Stream index: 0]
    [TCP Segment Len: 9]
    Sequence number: 11 (relative sequence number)
    Sequence number (raw): 3777320840
    [Next sequence number: 20 (relative sequence number)]
    Acknowledgment number: 11 (relative ack number)
    Acknowledgment number (raw): 3972403760
    1000 .... = Header Length: 32 bytes (8)
    ▶ Flags: 0x018 (PSH, ACK)
    Window size value: 509

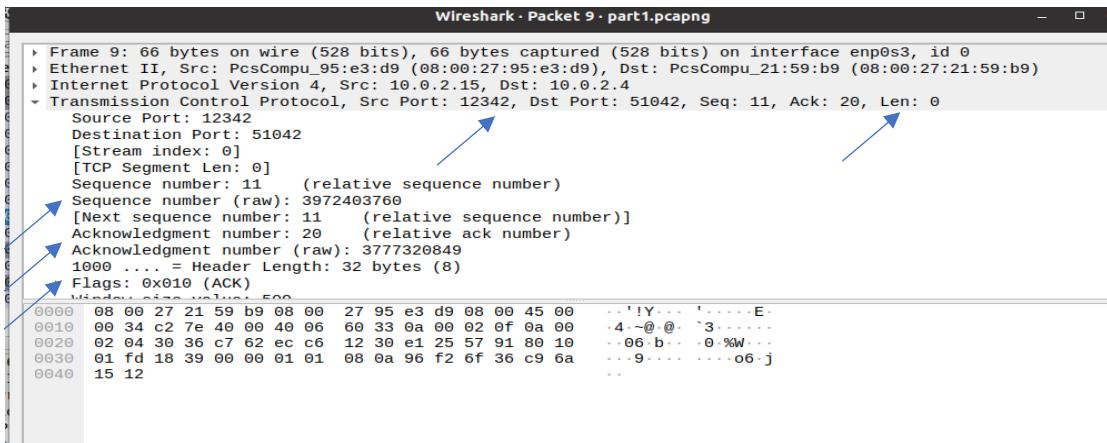
0000  08 00 27 95 e3 d9 08 00  27 21 59 b9 08 00 45 00  ..'.... '!Y.. E.
0010  00 3d 3f 85 40 00 40 06  e3 23 0a 00 02 04 0a 00  .=?@. @. #.....
0020  02 0f c7 62 30 36 e1 25  57 88 ec c6 12 30 80 18  ..b06.%W...0...
0030  01 f6 42 f3 00 00 01 01  08 0a c9 6a 15 12 96 f2  ..B.....j....
0040  6f 34 32 30 37 36 35 34  33 34 34 04207654 344

```

וآن השרת שלוח את הת"ז, מידע באורך 9 בתים, יחד עם ACK עם מס' 3972403760 סידורי 3777320840, המספרים האחוריים עליהם היה, ממשיר מהם והלאה. נראה את המידע בשכבות האפליקציה:

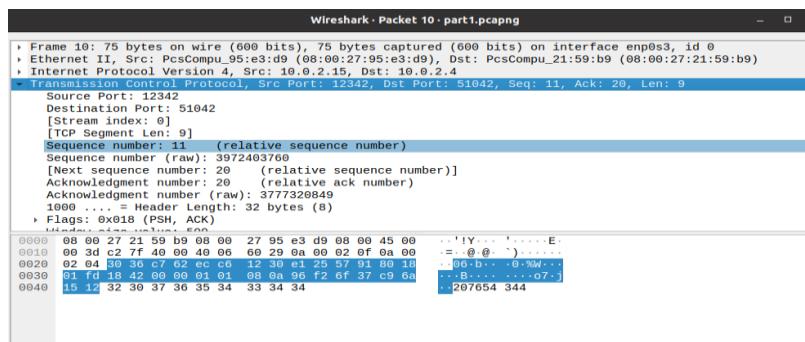


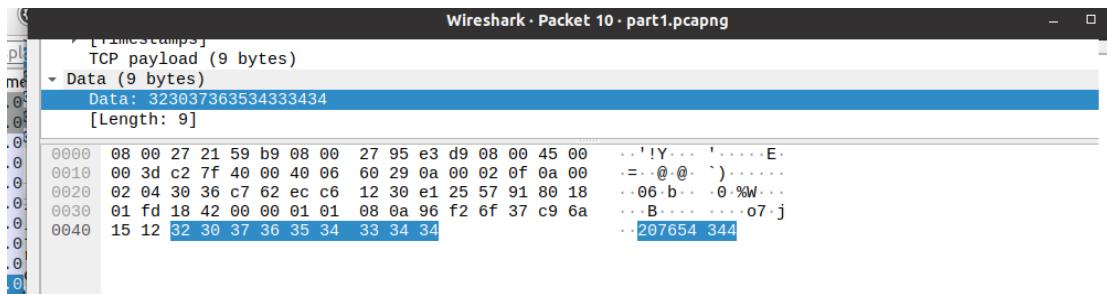
cut נצפה לקלט ACK על חבילת זו מצד הלוקה:



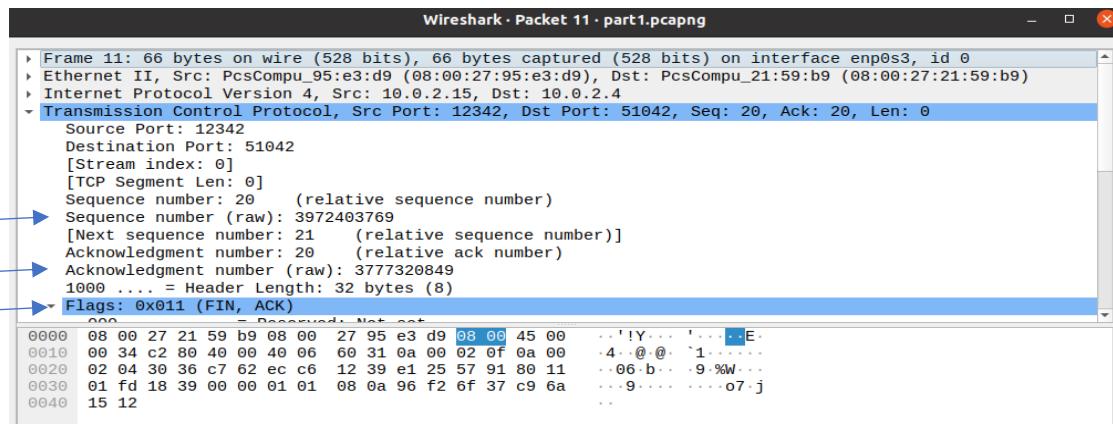
וآن, השרת מאשר את קבלת המידע על ידי ACK, 3777320849 סידורי, מס' 3972403760, בקודם+9בתים על המידע החדש. ב חבילה זו הוא לא מקבל מידע لكن האורך הוא 0, והמספר הסידורי הוא 3972403760, בדומה לACK האחרון, לשלוח מיד חדש لكن נשאר זהה.

cut, עליו להחזיר את המידע ללוקה, להחזיר את הת"ז, נראה שכן הוא שלוח מידע בגודל 9 בתים, עם ACK מס' 3777320849 סידורי 3972403760 - כמו ב막ה הآخر בו הוא רק שלח ACK.



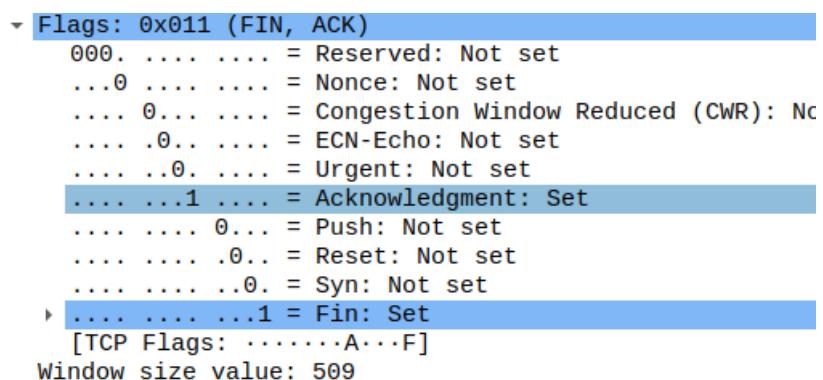


לאחר השרת שלח את המידע האחרון שהייתה לו לשלוח ללקוח, כעת, הוא יכריז על כך שאין יותר מידע להעביר ולכן מבeginתו ניתן לסייע קשר זה. لكن השרת ישלח ללקוח FIN, כמו שນיתן לראות בתצלומים יחד עם ACK:



נשים לב כי כעת המספר הסידורי עלה ל-**3972403769** מכיוון שהשרת שלח חבילה קודמת בגודל 9 בתים, המספר הסידורי שלו יהיה **3972403760**, המספר הקודם, ועוד 9 – הבתים שקיבל – ככלומר סה"כ **3972403769**. בנוספ', הักษת שלו עדין **3777320849** כי הוא לא שלוח מידע חדש AGAIN.

החינוך כאן הוא דגל FIN (שמצהיר על כוונת סיום הקשר) שמודלך יחד עם ACK:



כעת, הלוקוח יסכים עם השרת וישלח על בקשה השרת ACK :

Wireshark - Packet 12 · part1.pcapng

```

    ▶ Transmission Control Protocol, Src Port: 51042, Dst Port: 12342, Seq: 20, Ack: 20, Len: 0
      Source Port: 51042
      Destination Port: 12342
      [Stream index: 0]
      [TCP Segment Len: 0]
      Sequence number: 20 (relative sequence number)
      Sequence number (raw): 3777320849
      [Next sequence number: 20 (relative sequence number)]
      Acknowledgment number: 20 (relative ack number)
      Acknowledgment number (raw): 3972403769
      1000 .... = Header Length: 32 bytes (8)
    ▶ Flags: 0x010 (ACK)
      0000 .... .... = Reserved: Not set
      ...0 .... .... = Nonce: Not set
      .... 0.... .... = Congestion Window Reduced (CWR): Not set
      .... 0.... .... = ECN Echo: Not set
      0000 08 00 27 95 e3 d9 08 00 27 21 59 b9 08 00 45 00 ...'.... !Y...E.
      0010 00 34 3f 86 40 00 40 06 e3 2b 0a 00 02 04 0a 00 ..4? @@... +.....
      0020 02 0f c7 62 30 36 e1 25 57 91 ec c6 12 39 80 10 ...b06 % W....9...
      0030 01 f6 48 bd 00 00 01 01 08 0a c9 6a 15 13 96 f2 ..H..... j....
      0040 6f 37                                     o7
  
```

נראה כי המספר הסידורי של הלקוח הוא 3777320849 זהה לACK הקודם  
מידע לנו נשאר כך. ACK הוא 3972403769, בדומה למספר הסידורי הקודם.

ועכשיו, הלקוח יצהיר גם על כוונתו לניתוק הקשר, וישלח גם-CN ACK עם FIN ללקוח, כאשר  
המספר הסידורי שלו נשאר 3777320849, אך הפעם ה-ACK הוא 3972403770  
. (ACK 1 + 3972403769)

Wireshark - Packet 13 · part1.pcapng

```

    ▶ Frame 13: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface enp0s3, id 0
    ▶ Ethernet II, Src: PcsCompu_21:59:b9 (08:00:27:21:59:b9), Dst: PcsCompu_95:e3:d9 (08:00:27:95:e3:d9)
    ▶ Internet Protocol Version 4, Src: 10.0.2.4, Dst: 10.0.2.15
    ▶ Transmission Control Protocol, Src Port: 51042, Dst Port: 12342, Seq: 20, Ack: 21, Len: 0
      Source Port: 51042
      Destination Port: 12342
      [Stream index: 0]
      [TCP Segment Len: 0]
      Sequence number: 20 (relative sequence number)
      Sequence number (raw): 3777320849
      [Next sequence number: 21 (relative sequence number)]
      Acknowledgment number: 21 (relative ack number)
      Acknowledgment number (raw): 3972403770
      1000 .... = Header Length: 32 bytes (8)
    ▶ Flags: 0x011 (FIN, ACK)
      0000 08 00 27 95 e3 d9 08 00 27 21 59 b9 08 00 45 00 ...'.... !Y...E.
      0010 00 34 3f 87 40 00 40 06 e3 2a 0a 00 02 04 0a 00 ..4? @@... *.....
      0020 02 0f c7 62 30 36 e1 25 57 91 ec c6 12 3a 80 11 ...b06 % W....;
      0030 01 f6 48 ba 00 00 01 01 08 0a c9 6a 15 14 96 f2 ..H..... j....
      0040 6f 37                                     o7
  
```

לבסוף, בחבילה האחורונה, השרת מאשר ושולח ACK. נשים לב שהשרות מנתק את הקשר  
עם הלקוח הזה ספציפי, אך פניו לקבלת לקוחות חדשים, השרת לא נסגר. מספרו הסידורי  
יהי ה-ACK האחרון שבוצע-0 ושליח ACK עם המספר הסידורי האחרון  
. 3777320850, סהכ 1 + 777320849

Wireshark - Packet 14 · part1.pcapng

```

    ▶ Frame 14: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface enp0s3, id 0
    ▶ Ethernet II, Src: PcsCompu_95:e3:d9 (08:00:27:95:e3:d9), Dst: PcsCompu_21:59:b9 (08:00:27:21:59:b9)
    ▶ Internet Protocol Version 4, Src: 10.0.2.15, Dst: 10.0.2.4
    ▶ Transmission Control Protocol, Src Port: 12342, Dst Port: 51042, Seq: 21, Ack: 21, Len: 0
      Source Port: 12342
      Destination Port: 51042
      [Stream index: 0]
      [TCP Segment Len: 0]
      Sequence number: 21 (relative sequence number)
      Sequence number (raw): 3972403770
      [Next sequence number: 21 (relative sequence number)]
      Acknowledgment number: 21 (relative ack number)
      Acknowledgment number (raw): 3777320850
      1000 .... = Header Length: 32 bytes (8)
    ▶ Flags: 0x010 (ACK)
      0000 08 00 27 21 59 b9 08 00 27 95 e3 d9 08 00 45 00 ...'.... !Y...E.
      0010 00 34 c2 81 40 00 40 06 60 30 0a 00 02 0f 0a 00 ..4-@@... @.....
      0020 02 04 30 36 c7 62 ec c6 12 3a e1 25 57 92 80 10 ..06.b... :%W...
      0030 01 fd 18 39 00 00 01 01 08 0a 96 f2 6f 38 c9 6a ...9.....o8.j
      0040 15 14
  
```

## חלק ב'

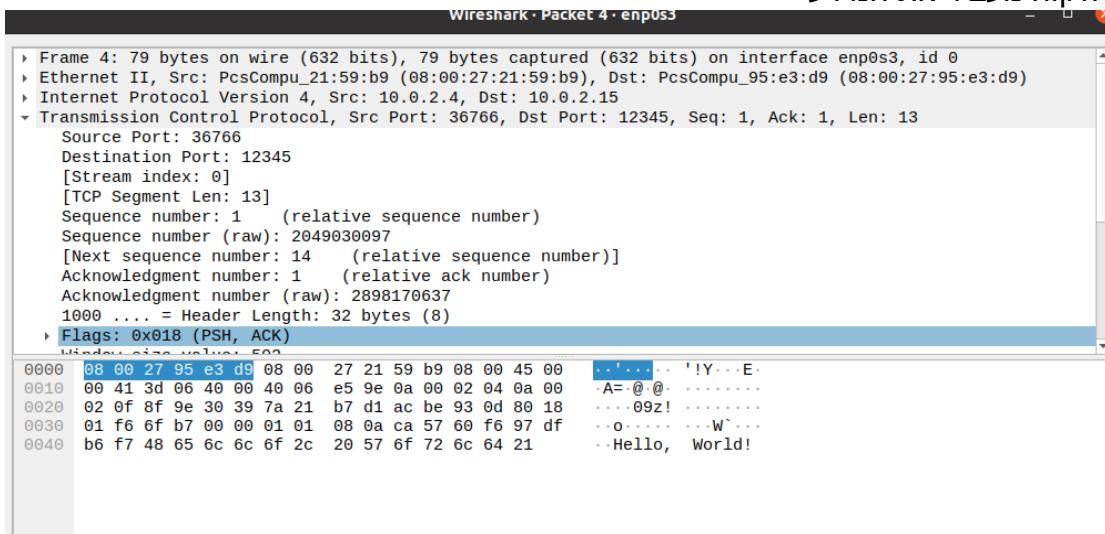
נ裏ץ את 1, נפתח זאת בwireshark ונסון לפ'י

```
((ip.dst == 10.0.2.4) && (ip.src==10.0.2.15)) || ((ip.dst == 10.0.2.4) && (ip.src==10.0.2.15))
```

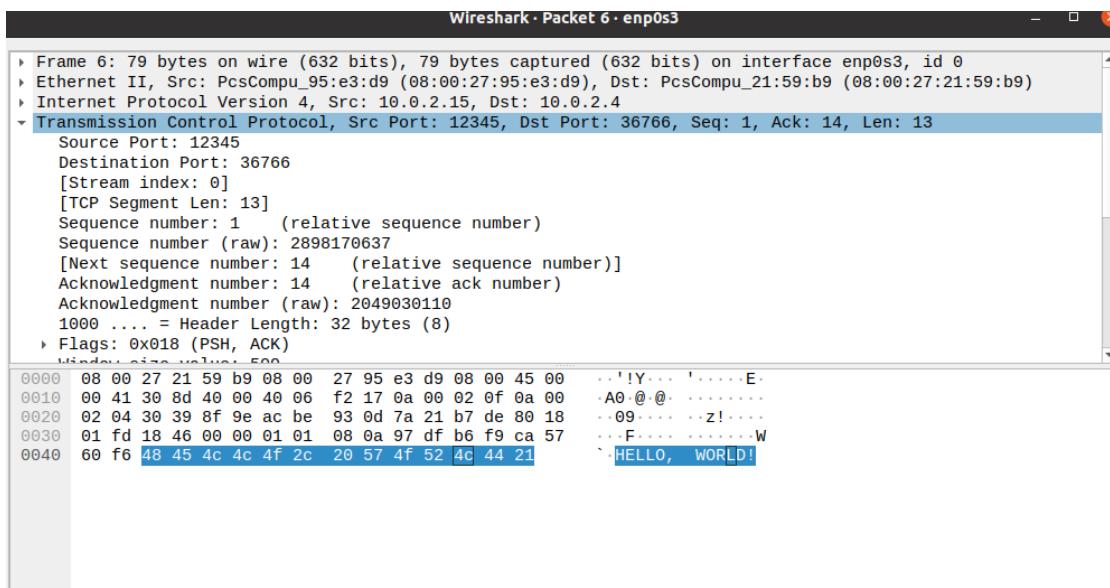
בחלק זה, Port השרת נקבע ע"י ארגומנט שאנו מכנים ים – נבחר 12345, וכשאר נ裏ץ את הלקוח ניתן את הארכומגנטים 12345 וגם 10.0.2.15 שזהו ip של השרת.

הלקוח שולח Hello, World לשרת והשרת יחזיר הודעה זו באותיות גדולות – WORLD. כלומר זה די זהה לכל הקודים שראינו עד בה, אנו עובדים עם TCP, لكن החיבור יבוצע ע"י 3 שלבים-הקמה (דגל SYN), שליחת מידע ויתוק החיבור (דגל FIN).

הלקוח מעביר את המידע -



והשרת מחזיר זאת באותיות גדולות-



מבחןת ניתוק, הלקוח שולח בקשה צד ראשונה, כאשר FIN והACK דלוקים.

```

> INTERNET PROTOCOL VERSION 4, Src: 10.0.2.4, Dst: 10.0.2.15
  Transmission Control Protocol, Src Port: 36766, Dst Port: 12345, Seq: 14, Ack: 14, Len: 0
    Source Port: 36766
    Destination Port: 12345
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 14      (relative sequence number)
    Sequence number (raw): 2049030110
    [Next sequence number: 15      (relative sequence number)]
    Acknowledgment number: 14      (relative ack number)
    Acknowledgment number (raw): 2898170650
    1000 .... = Header Length: 32 bytes (8)
  Flags: 0x011 (FIN, ACK)
    000. .... = Reserved: Not set
    ...0 .... = Nonce: Not set
    ...0.... = Congestion Window Reduced (CWR): Not set
    ACKNOWLEDGMENT NUMBER (raw): 2898170650
    1000 .... = Header Length: 32 bytes (8)
  Flags: 0x011 (FIN, ACK)
    000. .... = Reserved: Not set
    ...0 .... = Nonce: Not set
    ....0.... = Congestion Window Reduced (CWR): Not set
    ....0.. = ECN-Echo: Not set
    ....0.... = Urgent: Not set
    ....1.... = Acknowledgment: Set
    ....0... = Push: Not set
    ....0.. = Reset: Not set
    ....0.... = Syn: Not set
    ....1.... = Fin: Set
  [TCP Flags: .....A---F]
  window size value: 500
000 08 00 27 95 e3 d9 08 00 27 21 59 b9 08 00 45 00 ..!.....'!Y...E.
010 00 34 3d 08 40 00 40 06 e5 a9 0a 00 02 04 0a 00 ..4= @.0.....'.
020 02 0f 8f 9e 30 39 7a 21 b7 de ac be 93 1a 80 11 ....09z!.....
030 01 f6 b0 d4 00 00 01 01 08 0a ca 57 60 fd 97 df .....W.....
040 h6 f9 ...

```

אך השרת אינו מעוניין בסגירת הניתוק, הוא פניו לקבלת ליקוחות נוספים.

אכן מתקיים:

```

user@user-VirtualBox:~/PycharmProjects/network2/versions/v1$ python3 server.py
12345
New connection from: ('10.0.2.4', 36776)
received: b'Hello, World!'

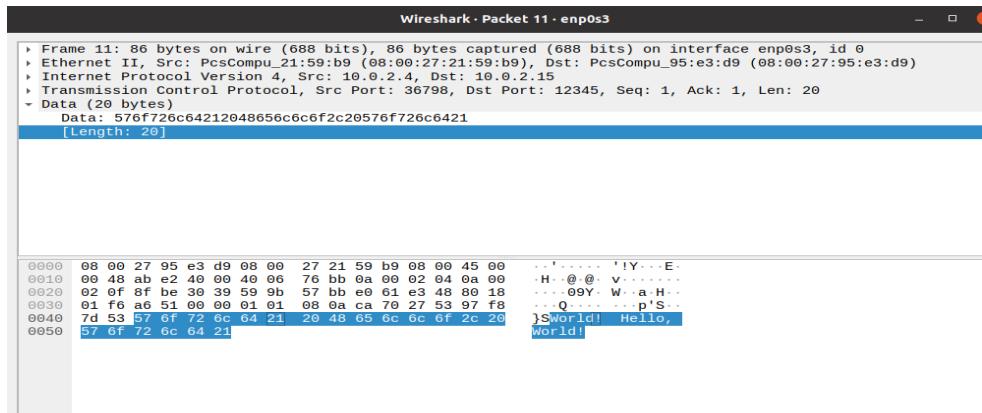
```

```

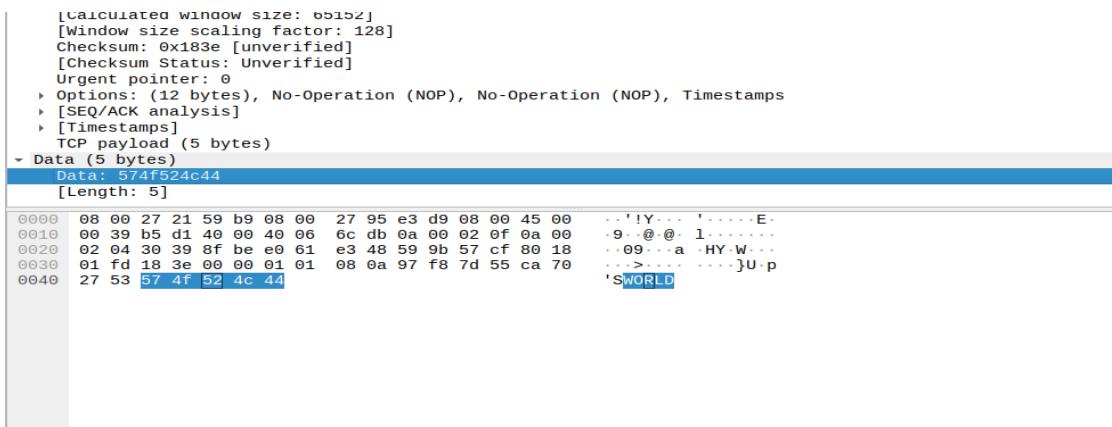
user@user-VirtualBox:~/PycharmProjects/network2/versions/v1$ python3 client.py 10.0.2.15
12345
received data: b'HELLO, WORLD!'
user@user-VirtualBox:~/PycharmProjects/network2/versions/v1$ 

```

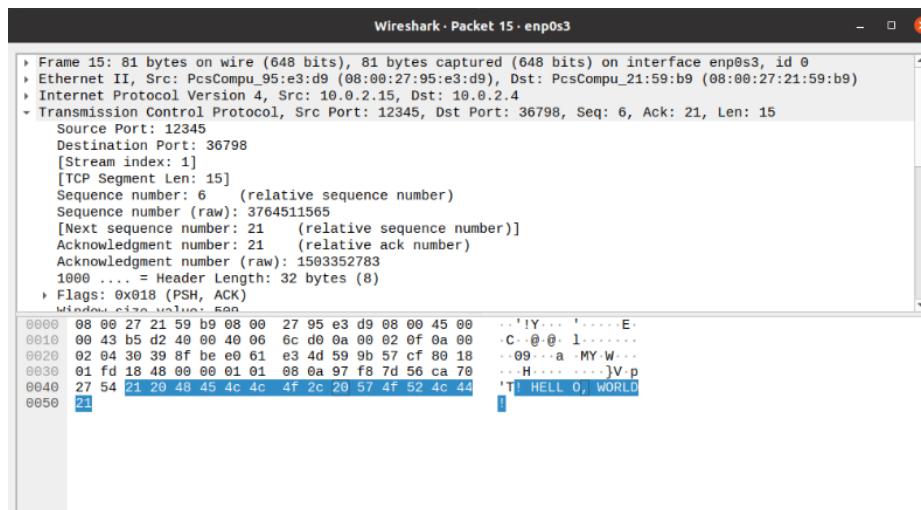
נубור **ל27**, כבר מהקוד עצמו ניתן לראות כי גודל BUFFER הוקטן משמעותית בשרת וכרגע הוא 5. הפעם, הלקוח שלוח **Hello, World!** ומצפה לקבל צאת מהשרת באזויות גדולות. שוב אנו בTCP لكن השלבים הכלליים זהים. המידע שהלקוח שלוח בגודל 20 בתים – במספר התווים בBYTES.



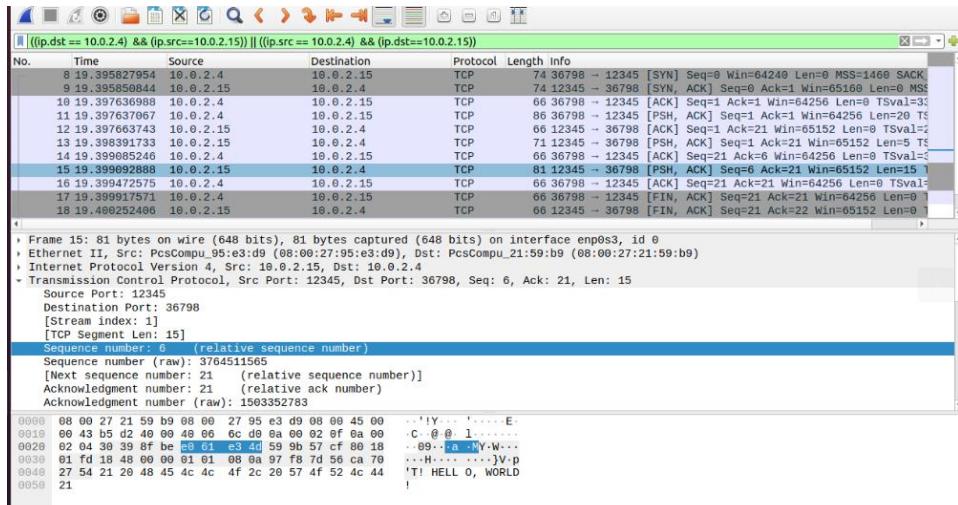
כעת, מכיוון שהשרת יכול לקבל עד 5 בתים, לפי הב哀ר, הוא יצליח לקרואครגע רק את 5 הבתים הראשונים, כלומר World.



כעת, מכיוון שיש עוד מידע לקרוא והשרת לא נסגר, הוא ימשיך לקבל 5 בתים הבאים, אך במקרה הבא, הוא עוד לא קיבל ACK על החבילה הנוכחית ולכן ימתין ובינתיים ימשיך לקרוא עוד 5 בתים ועוד 5 בתים, לאחר מכן יקבל ACK על כולם יחד, וכך נראה זאת:

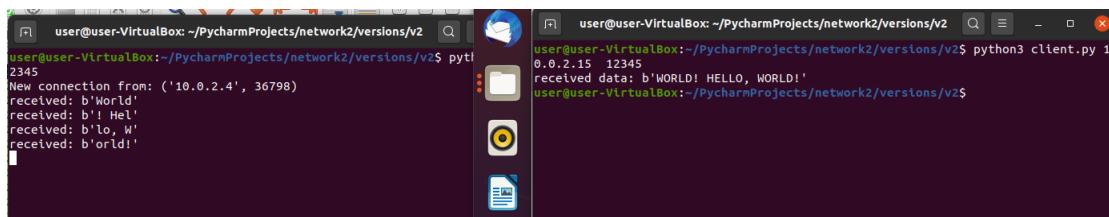


החvíלה המסומנת בבחול היא החvíלה עם שאר המÍדיע:



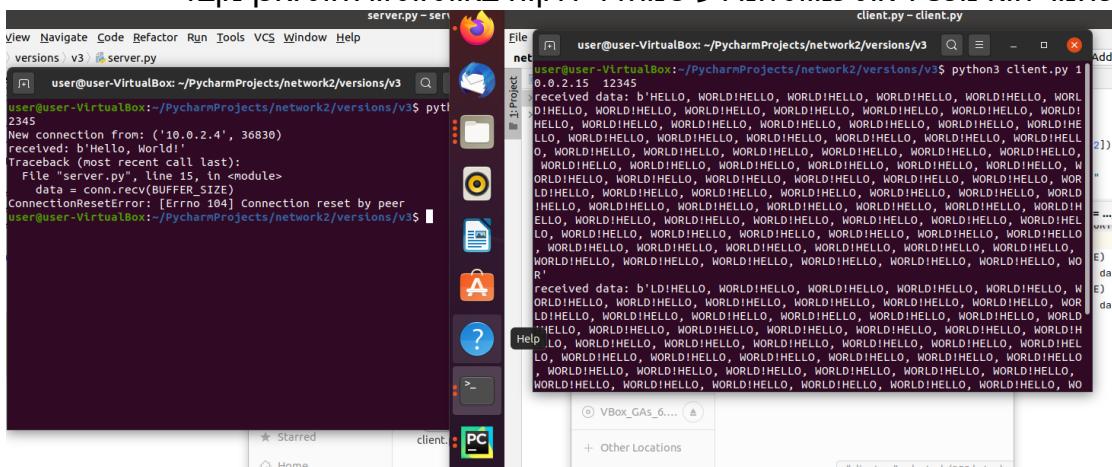
כֹּךְ שָׁנִיתָן לְרֹאֹת שַׁהֲיָא מִקְבָּלָת אֶת שָׁאֵר הַבִּיטּוֹ וְאַיְן עַלְיָהּ ACK בֵּין לְבֵין, רַק אַחַד סּוֹפִי, אַחֲרֵי חַבִּילָה זו. לְבִסּוֹפִי. מִתְבָּצֵעַ הַנִּتּוֹק עַבְורַ הַלְּקֹוח, הַשְׁרָת עַדְיִן מְחוּבָר.

סּוֹהֵג, נִקְבָּל:



נִעְבּוּר **לִזְבָּח**, כַּעַט גּוֹדֵל הַבָּאָפָר הוּא **1024**.  
הַלְּקֹוח בְּפִעְמֵם הַרְאָסְוָה שָׁולֵח "Hello, World!" לְשְׁרָת וּמַצְפָּה לְקַבֵּל זָאת בָּאוּתִוִּות גְּדוּלָות.  
נִשְׁימָנָה לְבָבִי בְּקָרוֹד הַשְׁרָת, בְּשׂוֹנוֹ מִמָּה שְׁקִיבָּלָנוּ עַד כֵּה, הוּא מִבְּצָעַ  
`conn.send(data.upper() * 1000)`

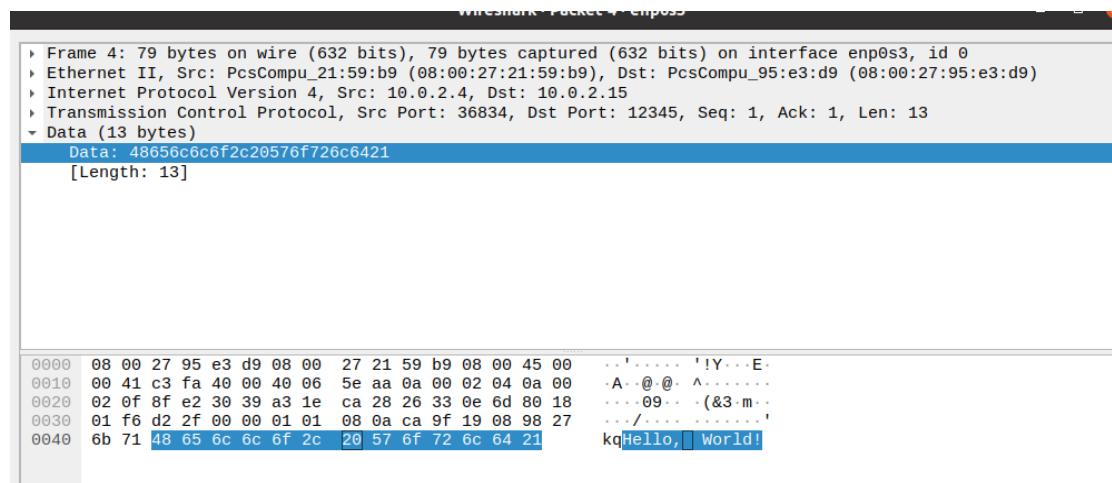
כְּלָוְמָר הָא מַכְפִּיל אֶת כִּמוֹת הַמִּידָע שְׁמַחְצֵיר לְקֹחַ בָּאוּתִוִּות גְּדוּלָות וְאַنְכָן נִקְבָּל



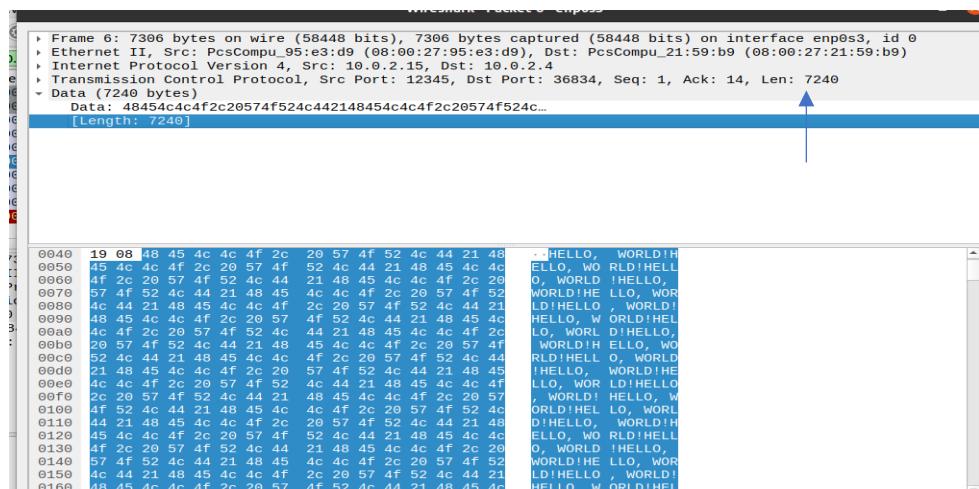
הַשְׁגַּיאָה נִבְעָת מִכְּרָ שָׁאָנוּ חֹזִים אֶת הַגְּבוּלּוֹת הַמוֹתְרֹות לְבָפָאָר. נִסְתַּכֵּל עַל כֵּךְ בְּwhireskark

((ip.dst == 10.0.2.4) && ((ip.src == 10.0.2.15))    ((ip.src == 10.0.2.4) && (ip.dst == 10.0.2.15)) )						
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.2.4	10.0.2.15	TCP	74	36834 → 12345 [SYN] Seq=0 Win=64240 Len=0 MSS=1466 SACK_PERM
2	0.000030653	10.0.2.15	10.0.2.4	TCP	74	12345 → 36834 [SYN, ACK] Seq=1 Win=65160 Len=0 MSS=146
3	0.000684143	10.0.2.4	10.0.2.15	TCP	66	36834 → 12345 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=339942
4	0.005628146	10.0.2.4	10.0.2.15	TCP	79	36834 → 12345 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=13 TSval=
5	0.005639183	10.0.2.15	10.0.2.4	TCP	66	12345 → 36834 [ACK] Seq=1 Ack=14 Win=65152 Len=0 TSval=25527
6	0.006103716	10.0.2.15	10.0.2.4	TCP	7306	12345 → 36834 [PSH, ACK] Seq=1 Ack=14 Win=65152 Len=7240 TS
7	0.006504118	10.0.2.15	10.0.2.4	TCP	5826	12345 → 36834 [PSH, ACK] Seq=7241 Ack=14 Win=65152 Len=5760
8	0.006712435	10.0.2.4	10.0.2.15	TCP	66	36834 → 12345 [ACK] Seq=14 Ack=7241 Win=60672 Len=0 TSval=33
9	0.007296552	10.0.2.4	10.0.2.15	TCP	66	36834 → 12345 [ACK] Seq=14 Ack=13001 Win=56832 Len=0 TSval=33
10	0.008753711	10.0.2.4	10.0.2.15	TCP	66	36834 → 12345 [RST, ACK] Seq=14 Ack=13001 Win=64128 Len=0 TS

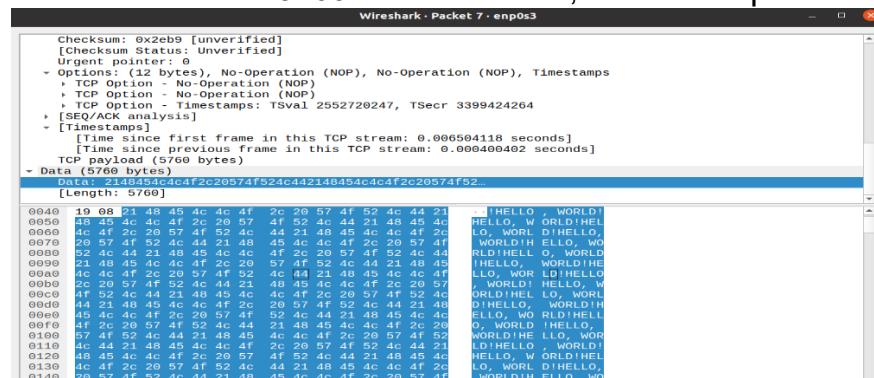
הקמת החיבור מתבצעת כרגע עבורי TCP, לאחר מכן, נראה כי הלווי שולח



לאחר קבלת ACK עבורי חביבה זו, קיבל חביבה שמקורה מהשרת-שלוחת את ה Hello World באותיות גדולות ברצף. גודל המידע הוא 7240.

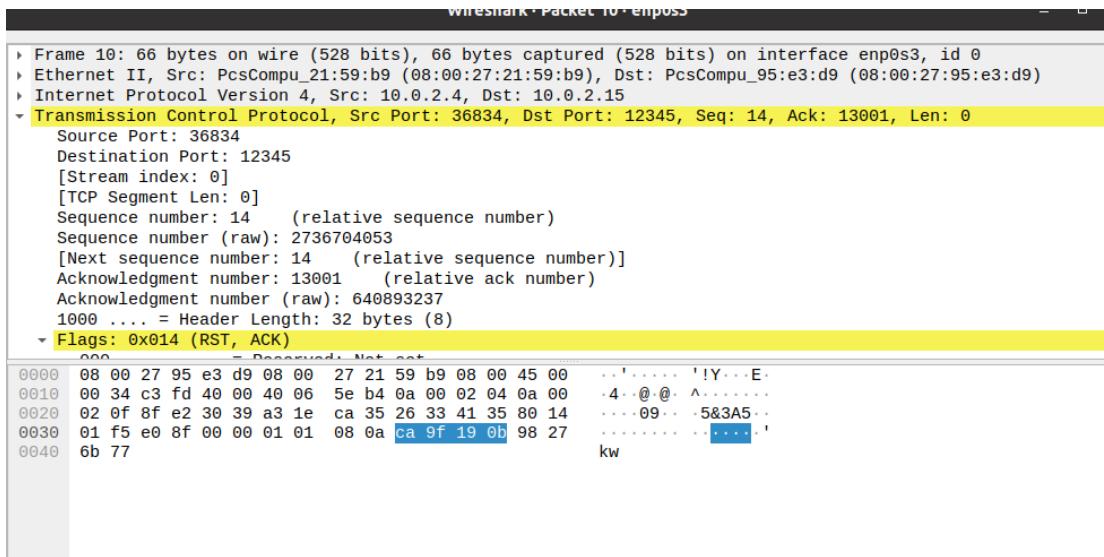


ולאחר מכן חביבה נוספת, נשים לב שאורך -5760



סה"כ גודל המידע הנשלח הוא 13000 בתים.

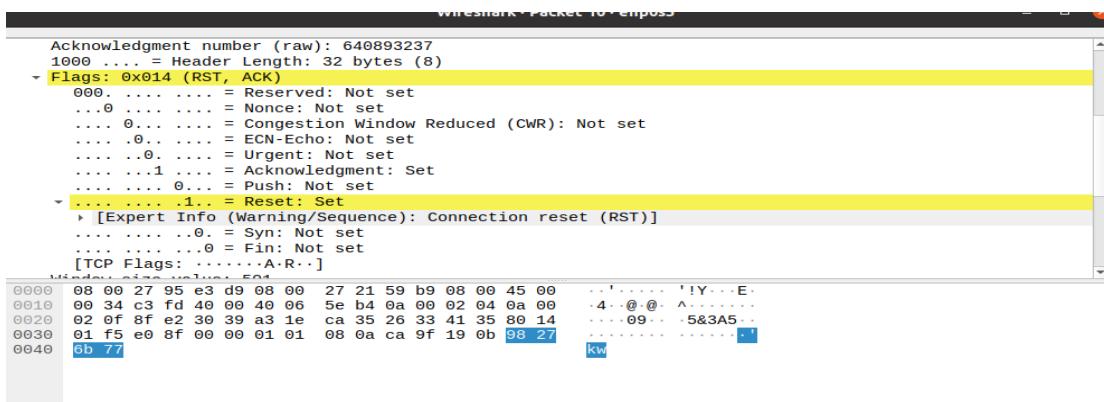
לבסוף, בחבילה האخונה המסומנת באדום:



Wireshark screenshot showing a single TCP packet (Frame 10) in hex and ASCII formats. The packet is a RST-ACK (Flags: 0x014). The sequence number is 14 and the acknowledgment number is 13001. The payload contains the string 'Hello World' followed by a carriage return and a line feed. The packet is captured on interface enp0s3.

```
Frame 10: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface enp0s3, id 0
Ethernet II, Src: PcsCompu_21:59:b9 (08:00:27:21:59:b9), Dst: PcsCompu_95:e3:d9 (08:00:27:95:e3:d9)
Internet Protocol Version 4, Src: 10.0.2.4, Dst: 10.0.2.15
Transmission Control Protocol, Src Port: 36834, Dst Port: 12345, Seq: 14, Ack: 13001, Len: 0
    Source Port: 36834
    Destination Port: 12345
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 14      (relative sequence number)
    Sequence number (raw): 2736704053
    [Next sequence number: 14      (relative sequence number)]
    Acknowledgment number: 13001      (relative ack number)
    Acknowledgment number (raw): 640893237
    1000 .... = Header Length: 32 bytes (8)
    Flags: 0x014 (RST, ACK)
        000 .... = Reserved: Not set
        000 00 27 95 e3 d9 08 00 27 21 59 b9 08 00 45 00 ...'. . . !Y..E.
        0010 00 34 c3 fd 40 00 40 06 5e b4 0a 00 02 04 0a 00 ..4. @. ^.....
        0020 02 0f 8f e2 30 39 a3 1e ca 35 26 33 41 35 80 14 ...09.. 5&3A5..
        0030 01 f5 e0 8f 00 00 01 01 08 0a ca 9f 19 0b 98 27 ..... .
        0040 6b 77                               kw
```

שנה מודולק הדגל של RST – ACK – קלומר הלקוח אומר שהוא לא יכול לקבל יותר מידע, יש לו עוד תווים בבאפר שהוא לא קרא מכיוון שהוא לא מבצע מספיק קריאות, ומקש לחיש-לרטט את הקשר.



Wireshark screenshot showing the same TCP RST-ACK packet from the previous screenshot, but with more detailed flag information. The 'Flags' field is expanded to show individual flags: Reserved (0), Nonce (0), Congestion Window Reduced (CWR) (0), ECN-Echo (0), Urgent (0), Acknowledgment (1), Push (0), and Reset (1). The payload shows the string 'Hello World' followed by a carriage return and a line feed.

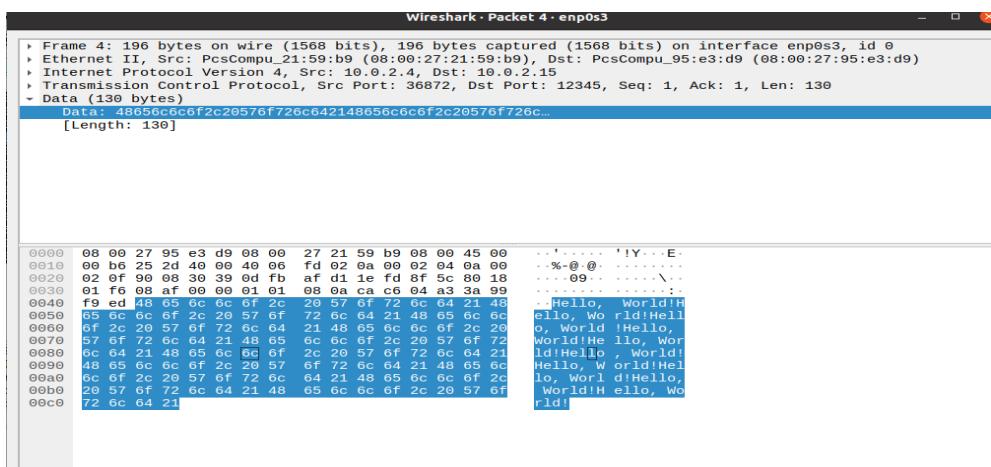
```
Acknowledgment number (raw): 640893237
1000 .... = Header Length: 32 bytes (8)
Flags: 0x014 (RST, ACK)
    000 .... = Reserved: Not set
    000 0.... = Nonce: Not set
    .... 0.... = Congestion Window Reduced (CWR): Not set
    .... 0.... = ECN-Echo: Not set
    .... 0.... = Urgent: Not set
    .... 1.... = Acknowledgment: Set
    .... 0.... = Push: Not set
    .... 1.... = Reset: Set
    > [Expert info (Warning/Sequence): Connection reset (RST)]
    .... 0.... = Syn: Not set
    .... 0.... = Fin: Not set
    [TCP Flags: .R...]
    Window size: 54
```

0000	08 00 27 95 e3 d9 08 00 27 21 59 b9 08 00 45 00	...'. . . !Y..E.
0010	00 34 c3 fd 40 00 40 06 5e b4 0a 00 02 04 0a 00	..4. @. ^.....
0020	02 0f 8f e2 30 39 a3 1e ca 35 26 33 41 35 80 14	...09.. 5&3A5..
0030	01 f5 e0 8f 00 00 01 01 08 0a ca 9f 19 0b 98 27	..... .
0040	6b 77	kw

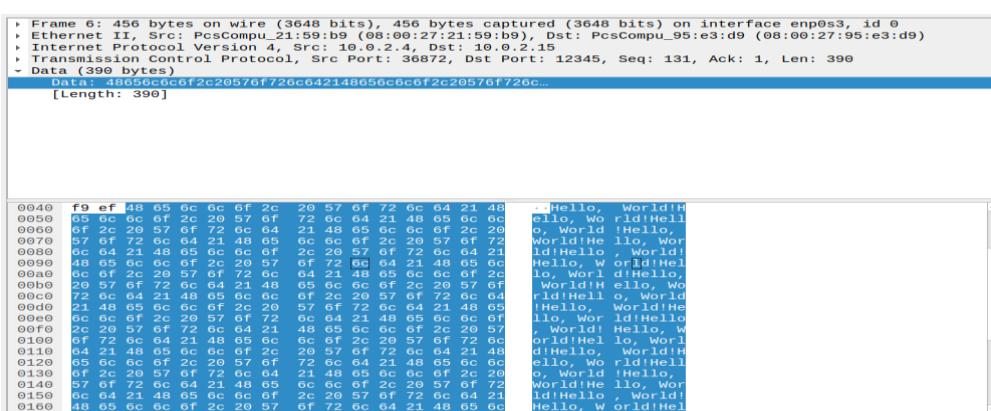
מעבר ל4א, בחלק זה ניתן לראות כי הלקוח שולח 4 פעמים את המידע שלו שזה Hello World כאשר כל פעם שהוא שולח את המידע הוא מכפיל זאת ב10:  
MESSAGE\*10)).send(s.)

בנוסף, נשים לב כי בקוד השרת נוספה שורה `time.sleep(5)`, קלומר עליו להמתין 5 שניות לפני שהוא מקבל מידע, אך השרת ישאר על מצב זה, בדומה לתמונה, לפחות 5 שניות:

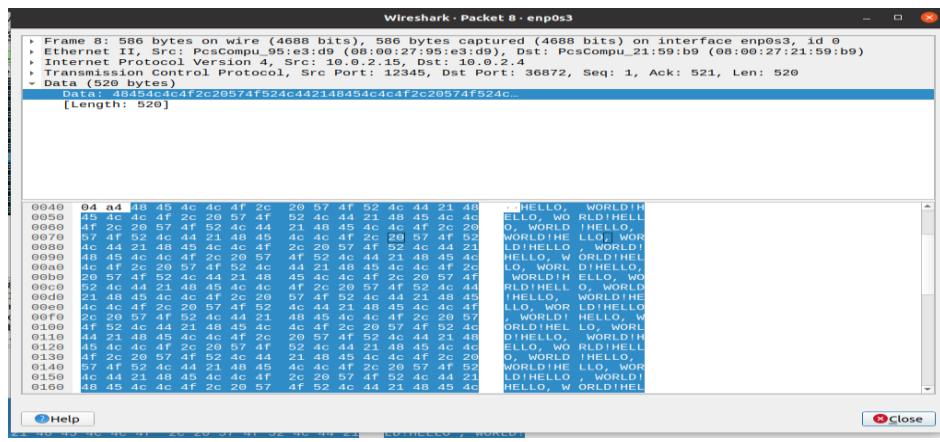
icutennetzaat. Me'cioon shano be'protokol TCP, ha'shelb shel hachibor matk'im carigil. Ha'pum, ha'lekhot sholoh rachz shel houdotot shel World Hello , bennigod le'3a, bagadol 130:



לאחר מכן יתקבל על כר ACK והשרת ימשיר לשולח, הפעם בגודל 390 בתים.



ואז השרת מתחילה להחזיר הודעות אלו באותיות גדולות, בגודל 520:



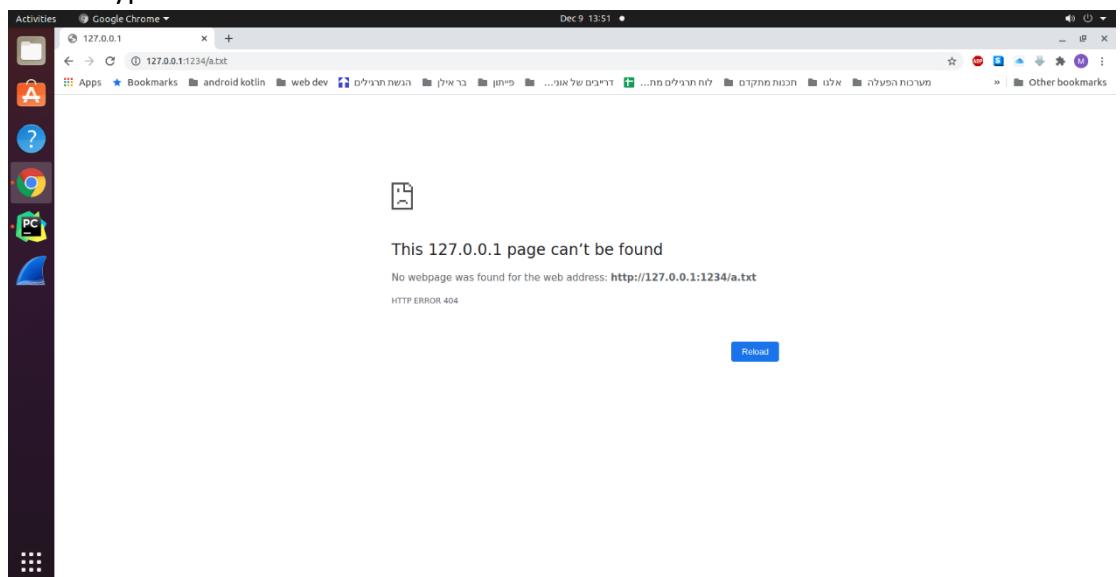
הלקוח מאשר את קבלת החבילות האחרונה בACK ואז מתחילה שלב ניתוק החיבור. סהכ', המידע שMOVVER CAN קטון יותר מהמידע שהועבר בעט וגורם להגעת גבול הבאפר. لكن CAN החיבור יסתיים כהרגלן.

## Part 2 of the assignment – implement TCP server:

Because you didn't ask to run the client and server in different machines, I ran them on the same virtual machine – Ubuntu OS with chrome browser and now for the conclusions :

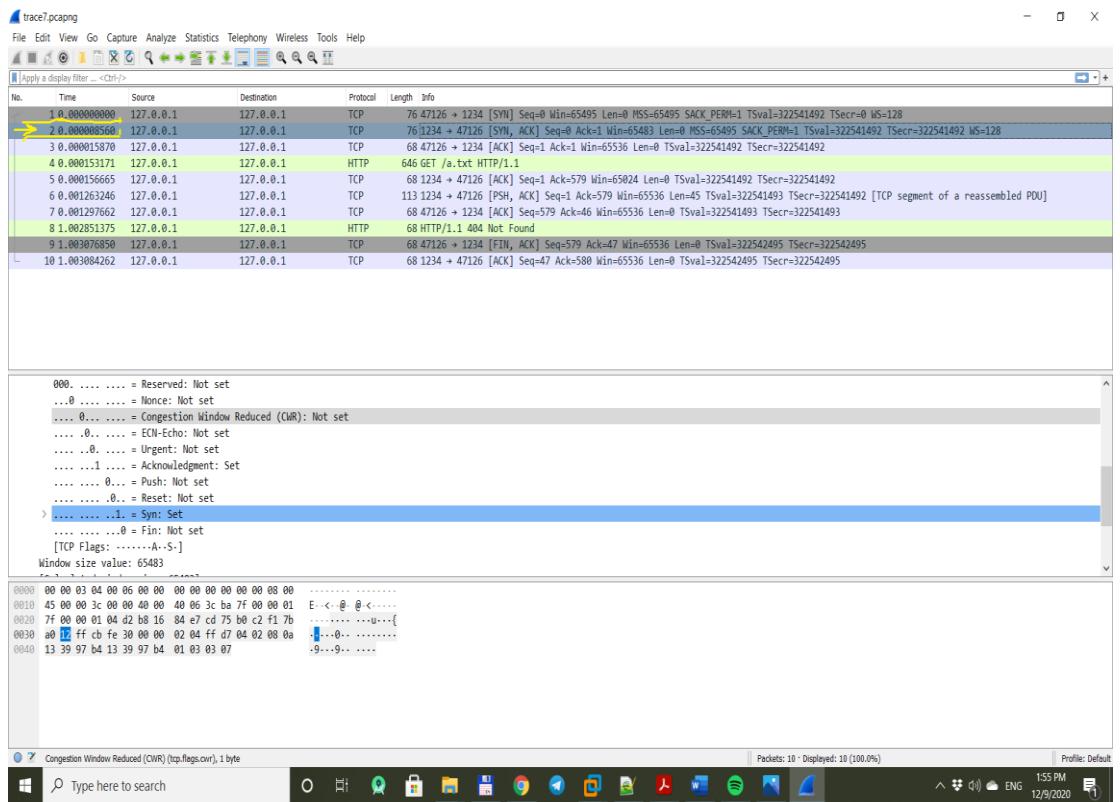
- a. Firstly I started by performing a simple sanity test for my code- I entered an illegal URI in order to get "404 Not Found" error message in the browser and ran wireshark in the background to sniff the data

What I typed in the browser :

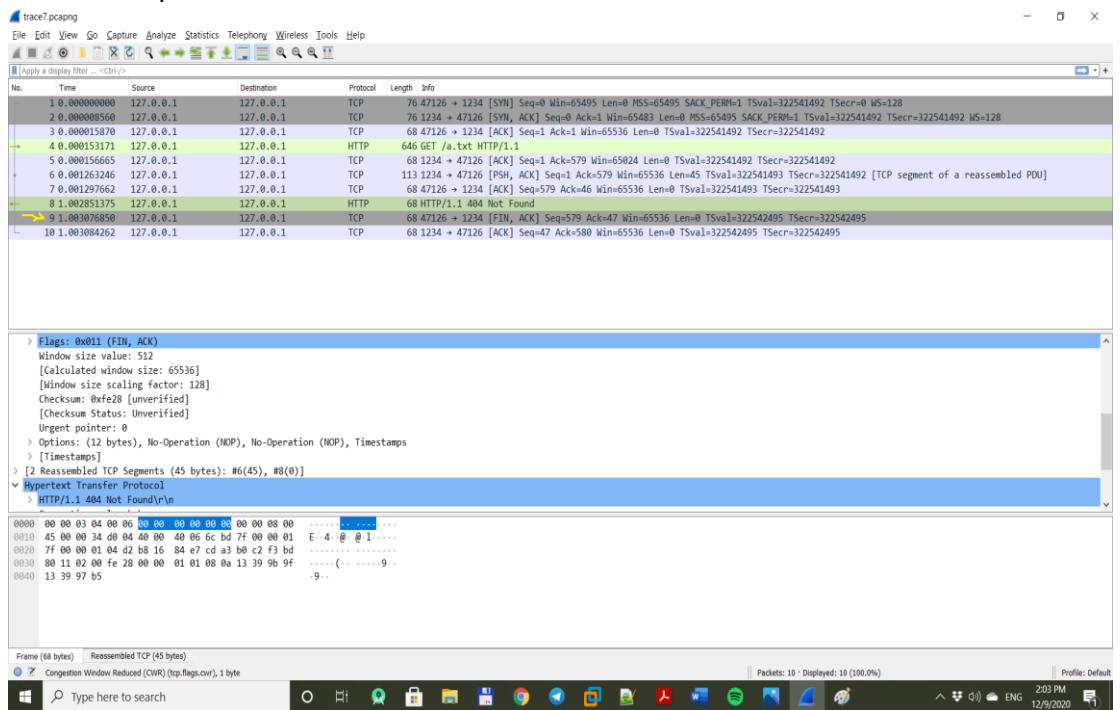


Now lets take a look at the sniff from wireshark:

First we see that there was a hands shaking between the server and the browser(the client), only one socket opened and the client turned on the SYN flag and then ACK,SYN flags were turned in the message from server



Afterwards the client sent another message with data – the requested URI but then we see that the URI isn't found in the server hence the server sent to the client a classic "404 Not Found" error message . we can see it in the wireshark packets – the server searched for the URI and because it didn't find it , it send a packet to the client with FIN,ACK flags turned on – saying that the server has ended the current session and it has nothing else to send regarding the previous request and also the server closed the socket due to exercise requirement



Afterwards, the client sent ACK to the server with no data in packet 10 in the sniffing so we conclude that the client didn't requested for another URI.

b. Now lets check what happens when the client type in the browser the address "127.0.0.1:1234/redirect"

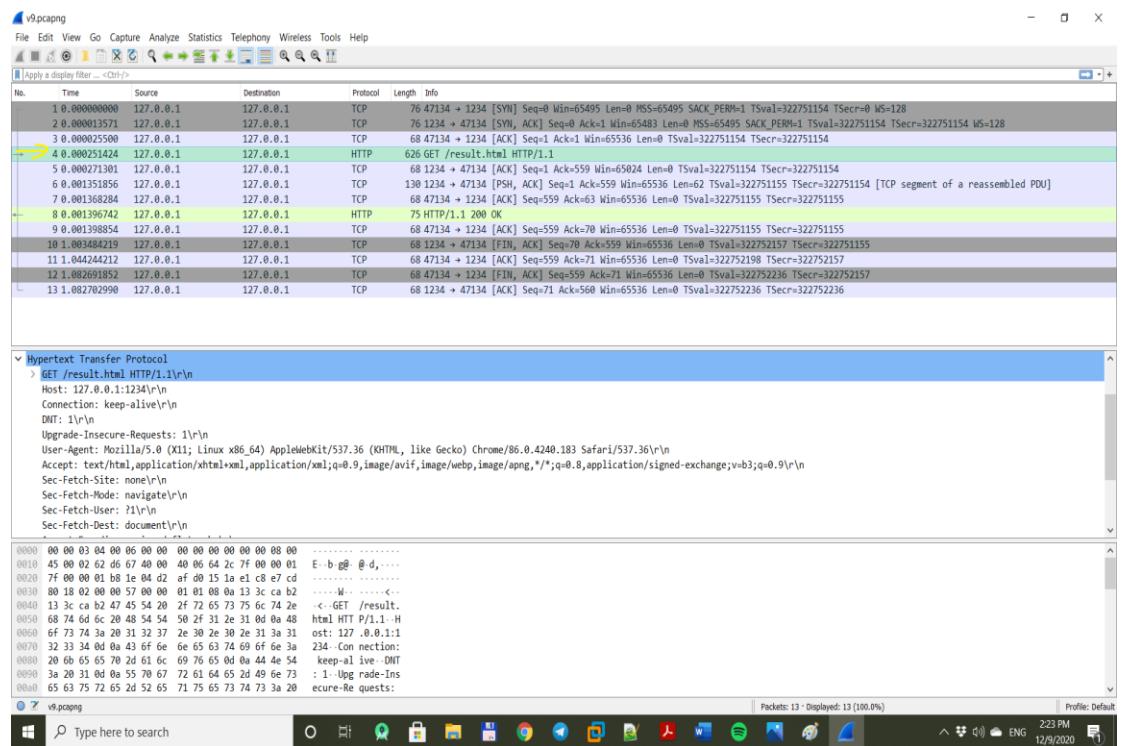
ran wireshark in the background to sniff the data

What I typed in the browser : "127.0.0.1:1234/redirect" and pressed "Enter"

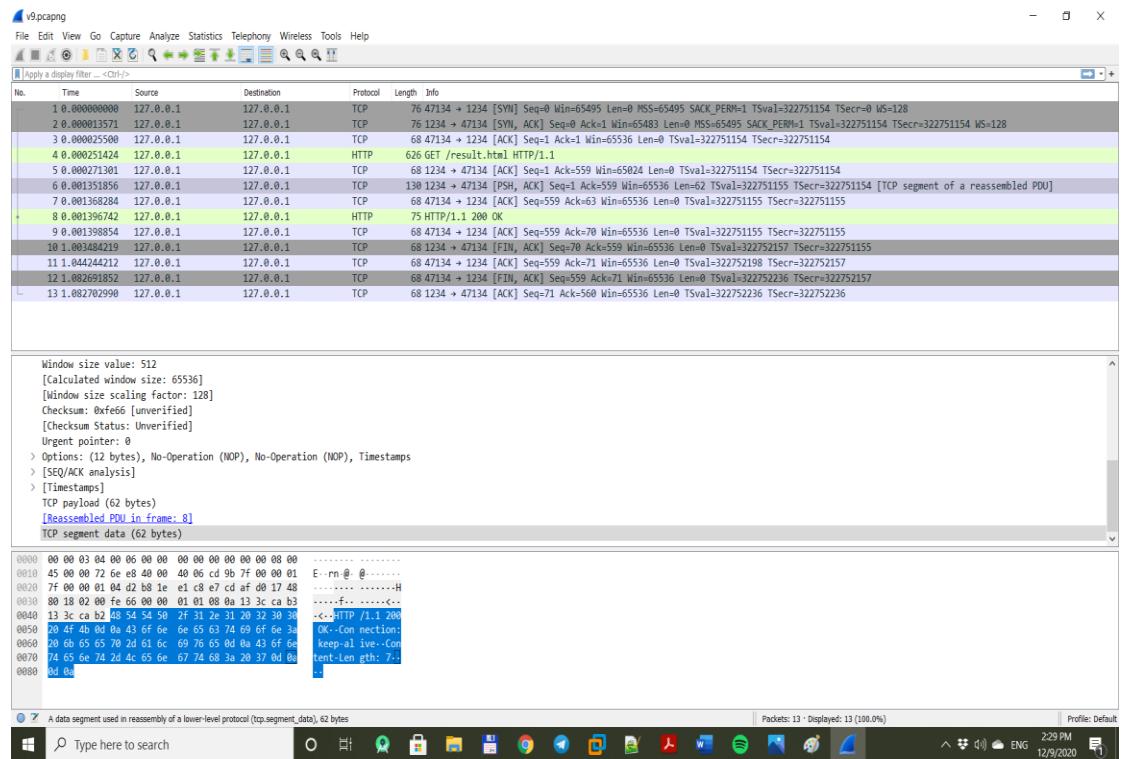
Then I got the data from the file "result.html" which is in "files" folder and the URI in the search box of the Chrome browser changed to "result.html"

Now lets take a look at the sniff from wireshark(trace9.pcap):

First we see that there was a hands shaking between the server and the browser(the client), only one socket opened and the client turned on the SYN flag and then ACK,SYN flags were turned on in the message from server  
Afterwards we see that the data from the client searching for "redirect" – in HTTP and the request pattern is in GET message - it means that the client wants the result.html file



hence the server sent to the client the content of results.html file along with status code of 200 which means the server found what the client was looking for and also wrote the content length:



In the requirement of the exercise the tutor ordered to close the socket after the response was sent to client so we see in the follow-up that the server sent a message to client with FIN,ACK flags turned on meaning that the server wants to close the socket and the client also sent ACK for the previous message from the server with FIN flag tuned on.

- c. Now lets see what happens when the client sends a request for same URI of "index.html" file and leaves the connection in "keep-alive" mode and after he got the response from the server and afterwards it requested for the same URI 2 consecutive times and leaves the connection in "keep-alive" mode and the server responded 2 more times with the same data in the same socket]/ The client code is attached in the zip by the name "clientwithkeepalivecon.py".

First , lets take a look at the response from the server in the terminal from where we executed the client.py script:

## First response:

```
Activities Terminal Dec 9 12:35 meni@ubuntu: ~/Desktop
meni@ubuntu:~/Desktop$ python "client (copy).py" 1
meni@ubuntu:~/Desktop$ python "client (copy).py" 1
meni@ubuntu:~/Desktop$ python client.py 1234
HTTP/1.1 200 OK
Connection: keep-alive
Content-Length: 558
<HTML>
<HEAD>
<TITLE>Your Title Here</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<CENTER><IMG SRC="clouds.jpg" ALIGN="BOTTOM"> </CENTER>
<HR>
<a href="http://somegreatsite.com">Link Name</a>
is a link to another nifty site
<H1>This is a Header</H1>
<H2>This is a Medium Header</H2>
Send me mail at <a href="mailto:support@yourcompany.com">
support@yourcompany.com</a>.
<P> This is a new paragraph!
<P> <B>This is a new paragraph!</B>
<BR> <B><I>This is a new sentence without a paragraph break, in bold italics.</I></B>
<HR>
</BODY>
</HTML>
```

## Second response :

```
Activities Terminal Dec 9 12:35 meni@ubuntu: ~/Desktop
meni@ubuntu:~/Desktop$ python "client (copy).py" 1
meni@ubuntu:~/Desktop$ python "client (copy).py" 1
meni@ubuntu:~/Desktop$ python client.py 1234
HTTP/1.1 200 OK
Connection: keep-alive
Content-Length: 558
<HTML>
<HEAD>
<TITLE>Your Title Here</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<CENTER><IMG SRC="clouds.jpg" ALIGN="BOTTOM"> </CENTER>
<HR>
<a href="http://somegreatsite.com">Link Name</a>
is a link to another nifty site
<H1>This is a Header</H1>
<H2>This is a Medium Header</H2>
Send me mail at <a href="mailto:support@yourcompany.com">
support@yourcompany.com</a>.
<P> This is a new paragraph!
<P> <B>This is a new paragraph!</B>
<BR> <B><I>This is a new sentence without a paragraph break, in bold italics.</I></B>
<HR>
</BODY>
</HTML>
```

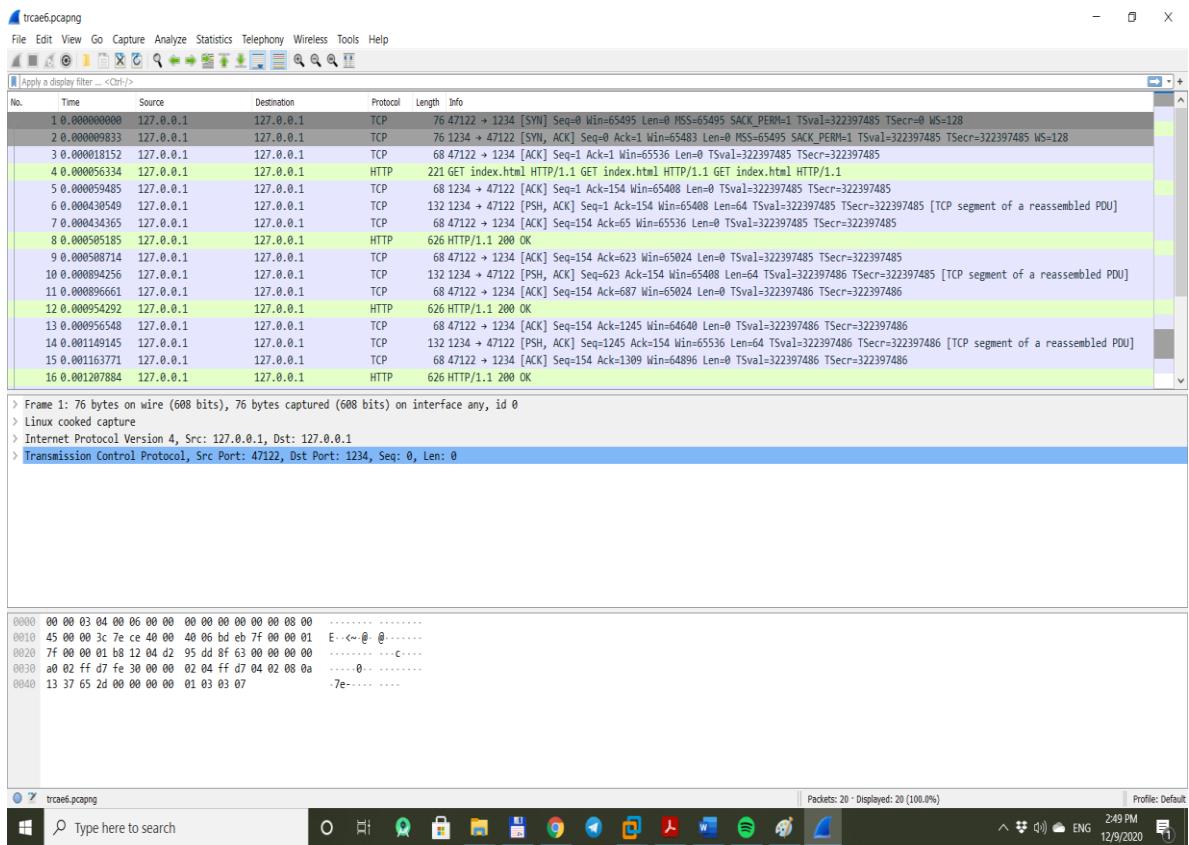
## Third response:

```
Activities Terminal Dec 9 12:35 meni@ubuntu: ~/Desktop

HTTP/1.1 200 OK
Connection: keep-alive
Content-Length: 558

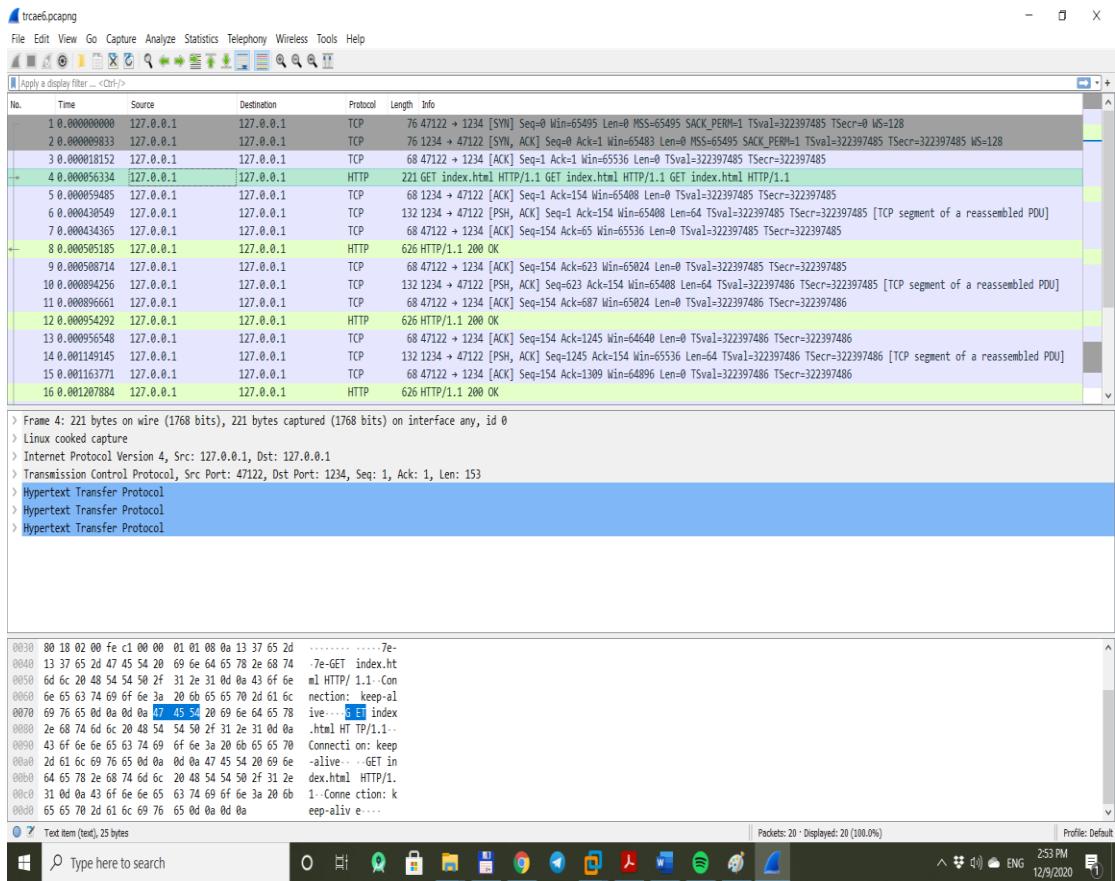
<HTML>
<HEAD>
<TITLE>Your Title Here</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<CENTER><IMG SRC="clouds.jpg" ALIGN="BOTTOM"> </CENTER>
<HR>
<a href="http://somegreatsite.com">Link Name</a>
is a link to another nifty site
<H1>This is a Header</H1>
<H2>This is a Medium Header</H2>
Send me mail at <a href="mailto:support@yourcompany.com">
support@yourcompany.com</a>.
<P> This is a new paragraph!
<P> <B>This is a new paragraph!</B>
<BR> <B><I>This is a new sentence without a paragraph break, in bold italiccs.</I></B>
<HR>
</BODY>
</HTML>
```

Lets take a look at the sniff from wireshark(trace6.pcap):

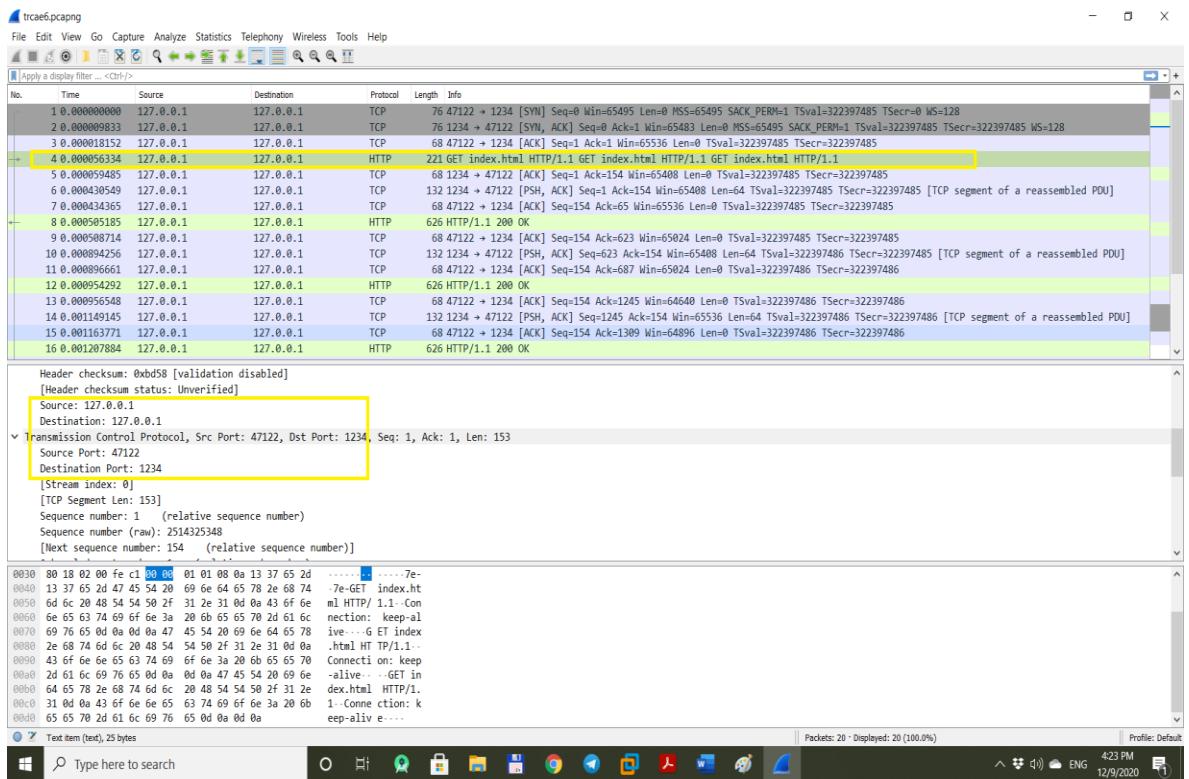


First we see that there was a hands shaking between the server and the browser(the client), only one socket opened and the client turned on the SYN flag and then ACK,SYN flags were turned on in the message from server

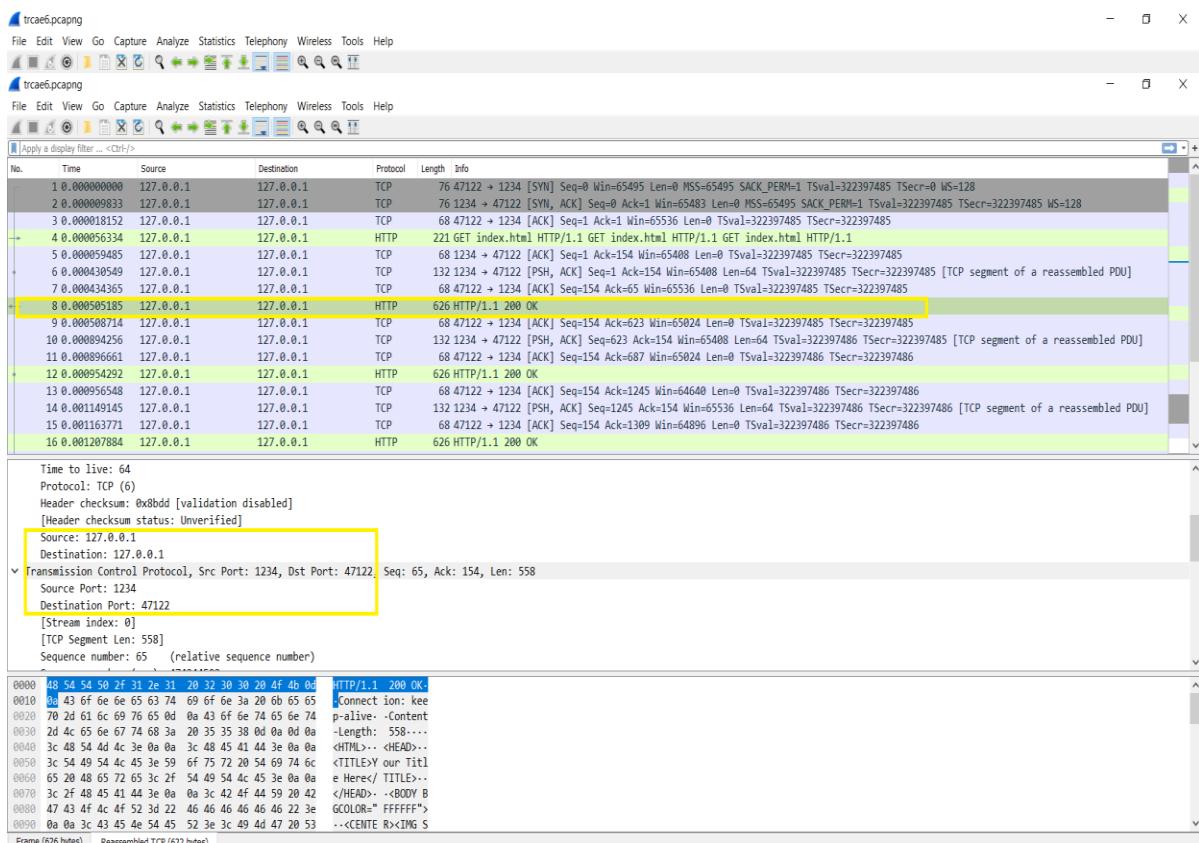
Afterwards, in packet no.4 we see the GET message in HTTP format and in that message we see that the client searched for URI "127.0.0.1:1234/files/index.html" three consecutive times. From packet no.4 :

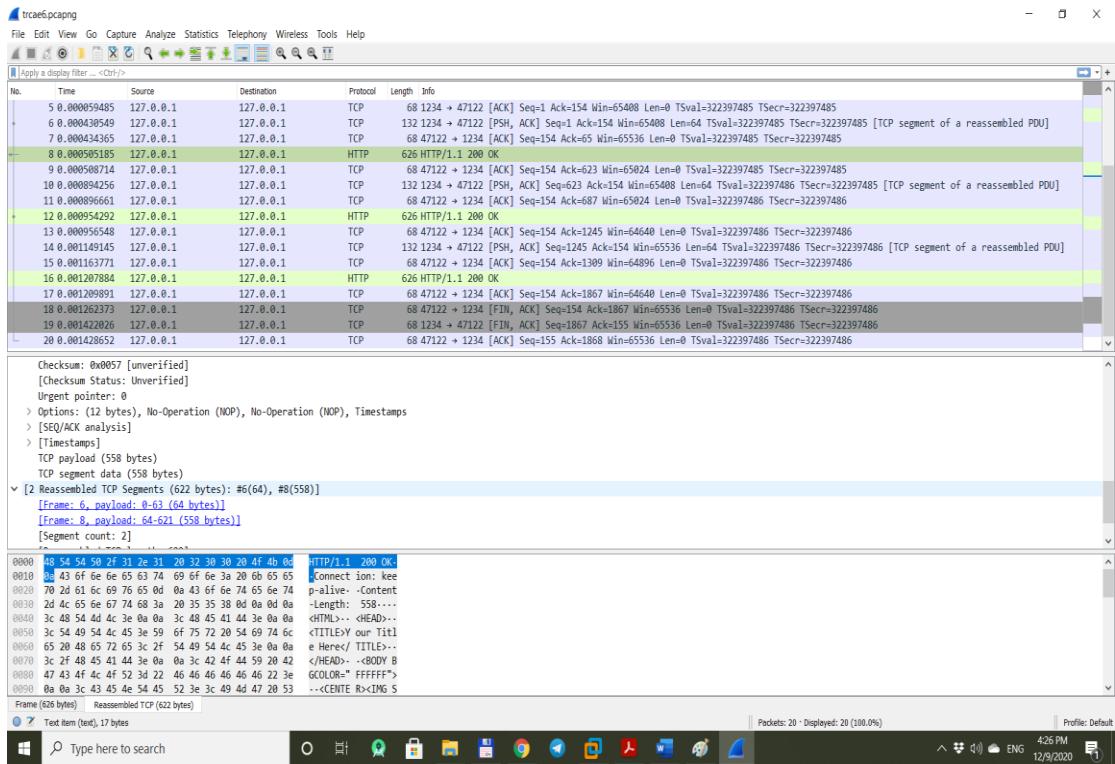


We see three GET requests for the same URI afterwards the server responded with ACK and in packets no.8,12,16 the server responded with HTTP message with status code of 200 which means the server found the file and responded to the client and after every "200 OK" message we see that the client sent ACK to the server. Its important to stress here that only one connection was opened in this session because the client asked for "keep-alive" connection mode and we can see in the wireshark that, just like Hemi showed up – the socket identifier is the combination of source IP+ source PORT + Dest IP + Dest PORT: (server port is 1234)



And in the server response :



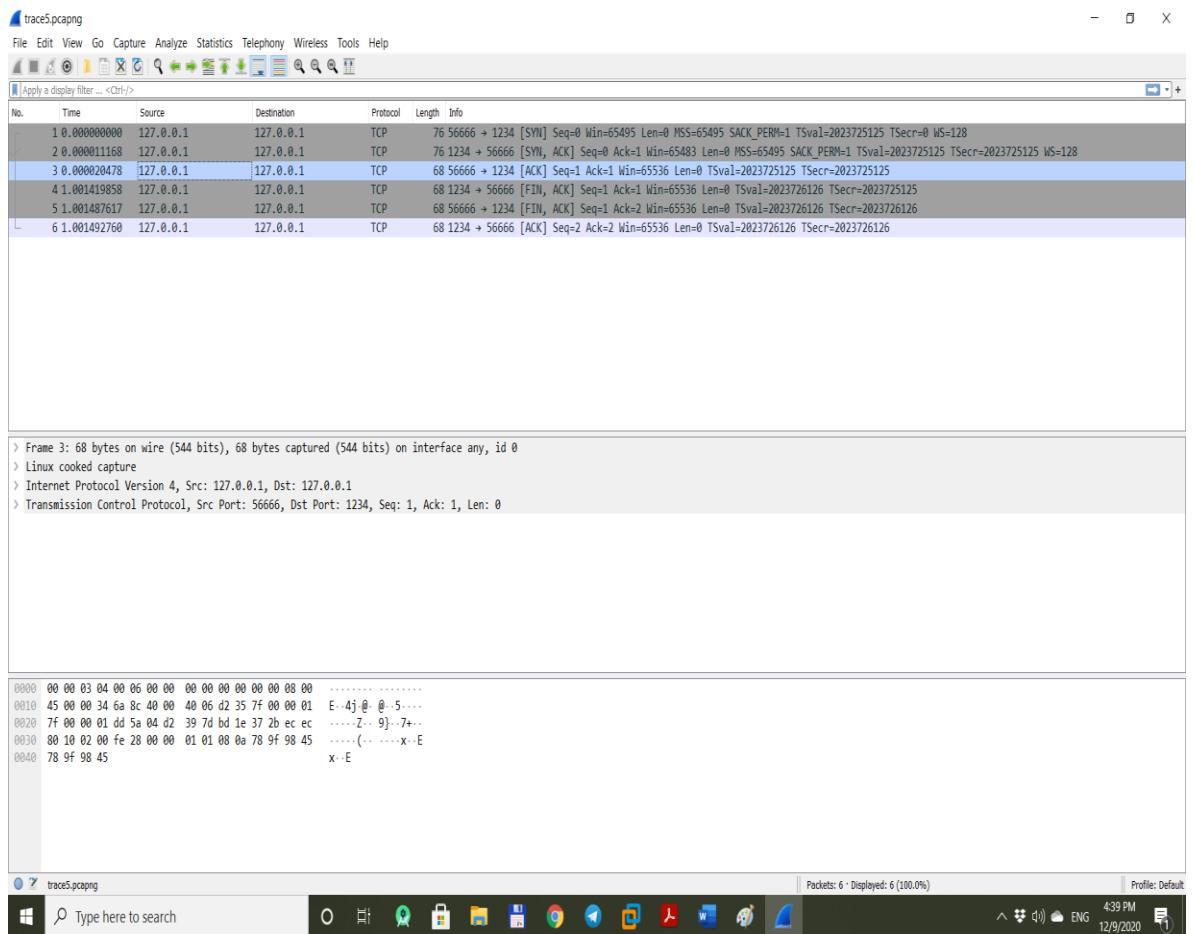


We can see that in each response the server had 622 B of data to send so it sends it in segments of size 558 B and 64 B. each consequent response after the first one will be identical

- d. Lets try to send an empty message to the server and see what happens:  
(the code for client with empty message is attached to zip and called "clientempty.py")

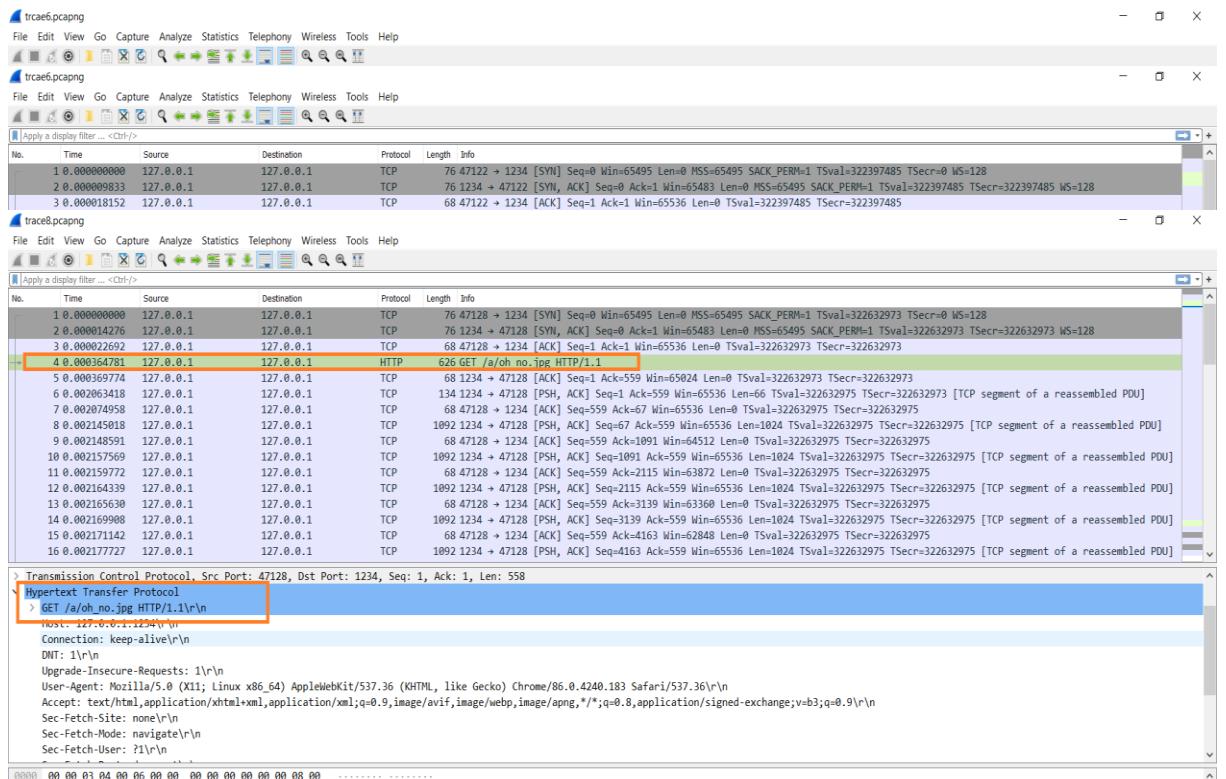
Lets see the sniff in "trace5.pcap"

First we see that there was a hands shaking between the server and the browser(the client), only one socket opened and the client turned on the SYN flag and then ACK,SYN flags were turned on in the message from server  
Afterwards we can see that the client doesn't have data to send (Data layer is absent from sniff packet 3)



And we see that afterwards the server immediately sent Message with ACK and FIN flags turned on which means that the server has nothing to respond and wants to close the socket. At the last packet the client sends ACK and the socket is closed

- e. Now lets run another scenario – download a jpg pic (trace8.pcap)  
First we see that there was a hands shaking between the server and the browser(the client), only one socket opened and the client turned on the SYN flag and then ACK,SYN flags were turned on in the message from server

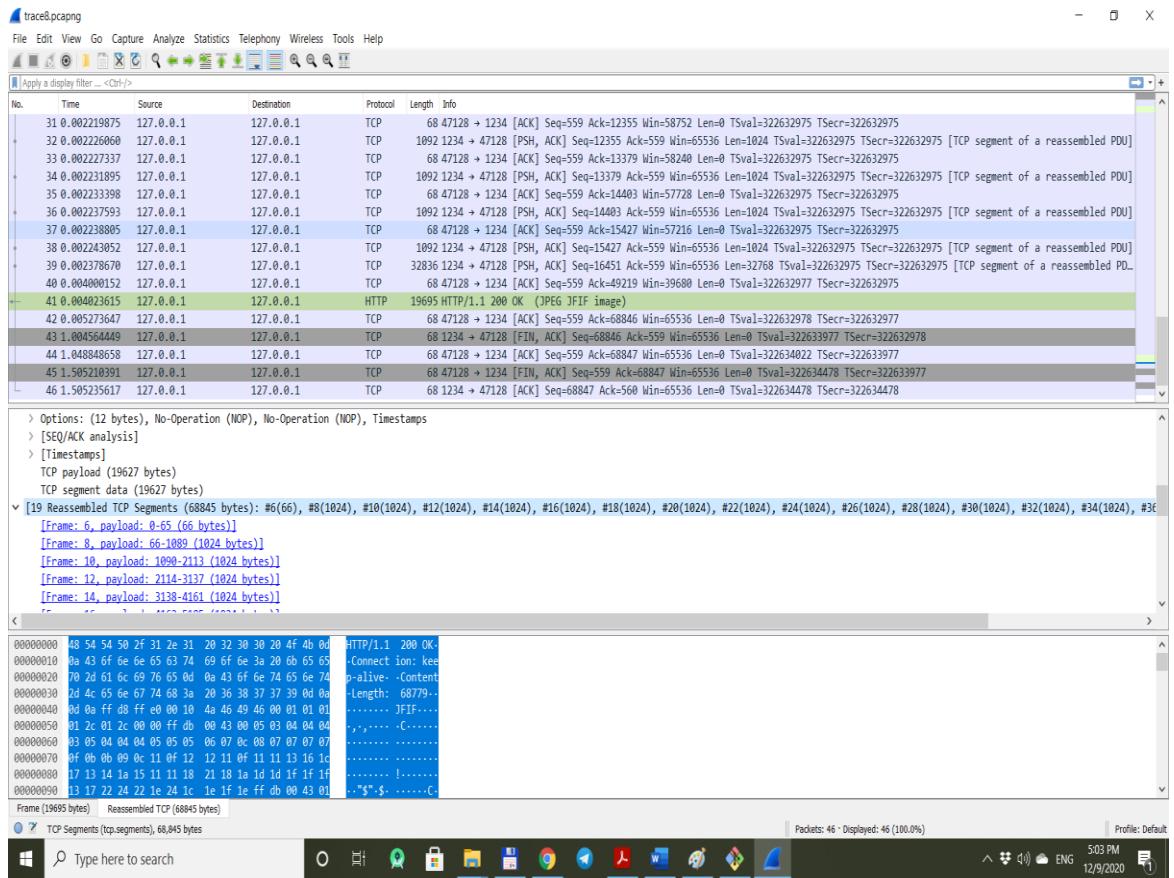


In packet no.4 we see that the client sent a request message in HTTP format – GET and after the name of the file request with its relative path in the server (URI)

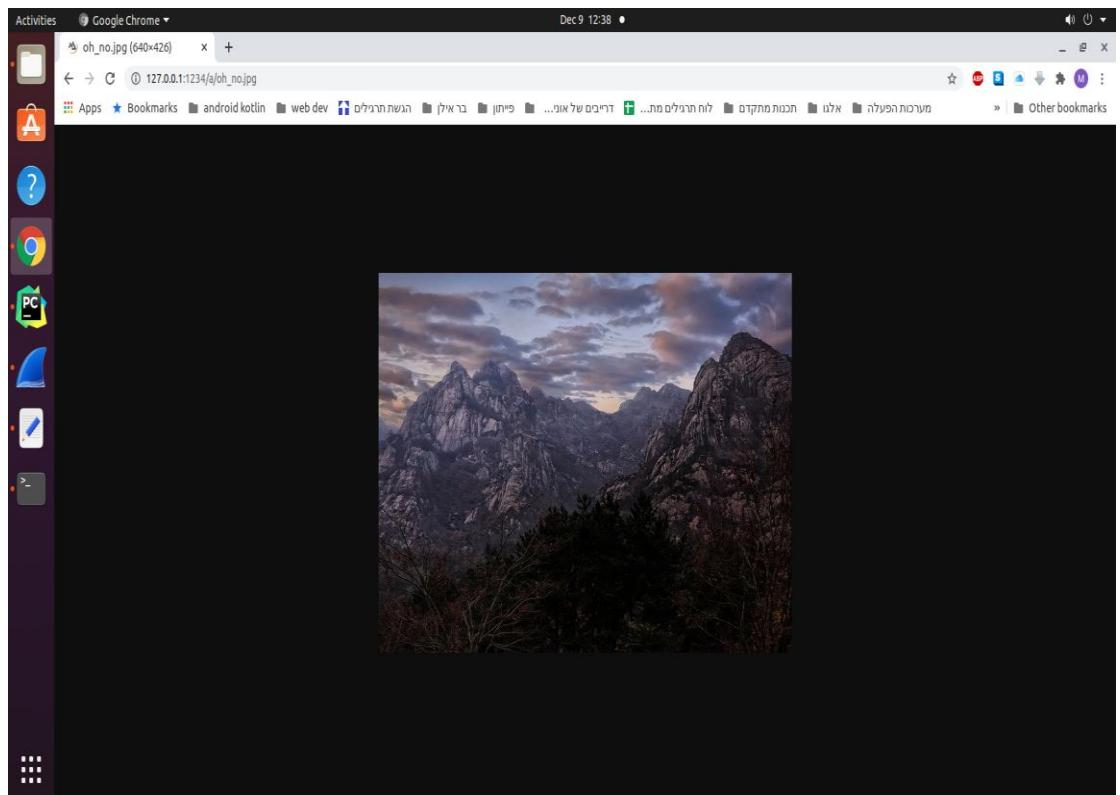
Lets describe what the server returned to the client : we have to understand that the jpg file is big and because TCP is a reliable protocol and data is transferred in bytes, the TCP has to separate the jpg file data to segments which will be delivered in order to the client and the client will form the segments into one big picture at the application layer .

In packet no.5 in the wireshark sniff we see the first segment that is transferred to the client , and in packet no.6 we see that the client sent ACK to the server saying that the segment was received successfully.

In packet no.41 in the wireshark sniff we see that the server finally sent the status code of 200 in HTTP format which means that everything went well and the data has been transferred completely



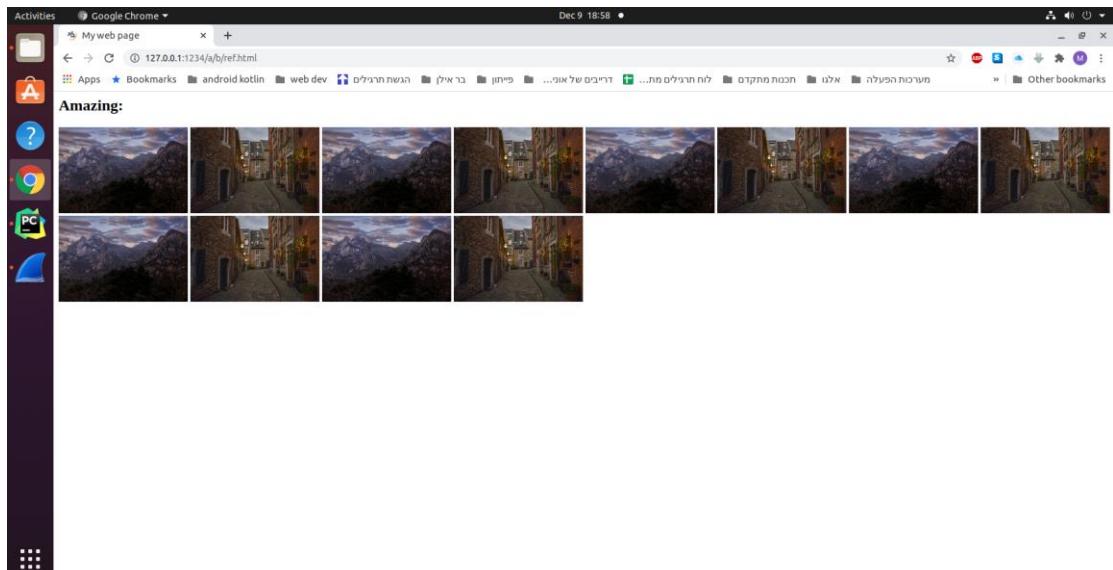
We can see inside packet no.41 the number of reassembled TCP segments that were sent to the client . In packet no.42 we see that the client sent ACK on the status code of 200 and that the server has finished delivering data to the client and in the consequent packets we see that the server sends message with FIN,ACK flags turned on which means the server wants to end to conversation and close the socket, the client sends FIN,ACK afterwards also and the last packet is the server sending ACK to the client that he is ready for the socket to be closed and then we could see the picture in the browser :



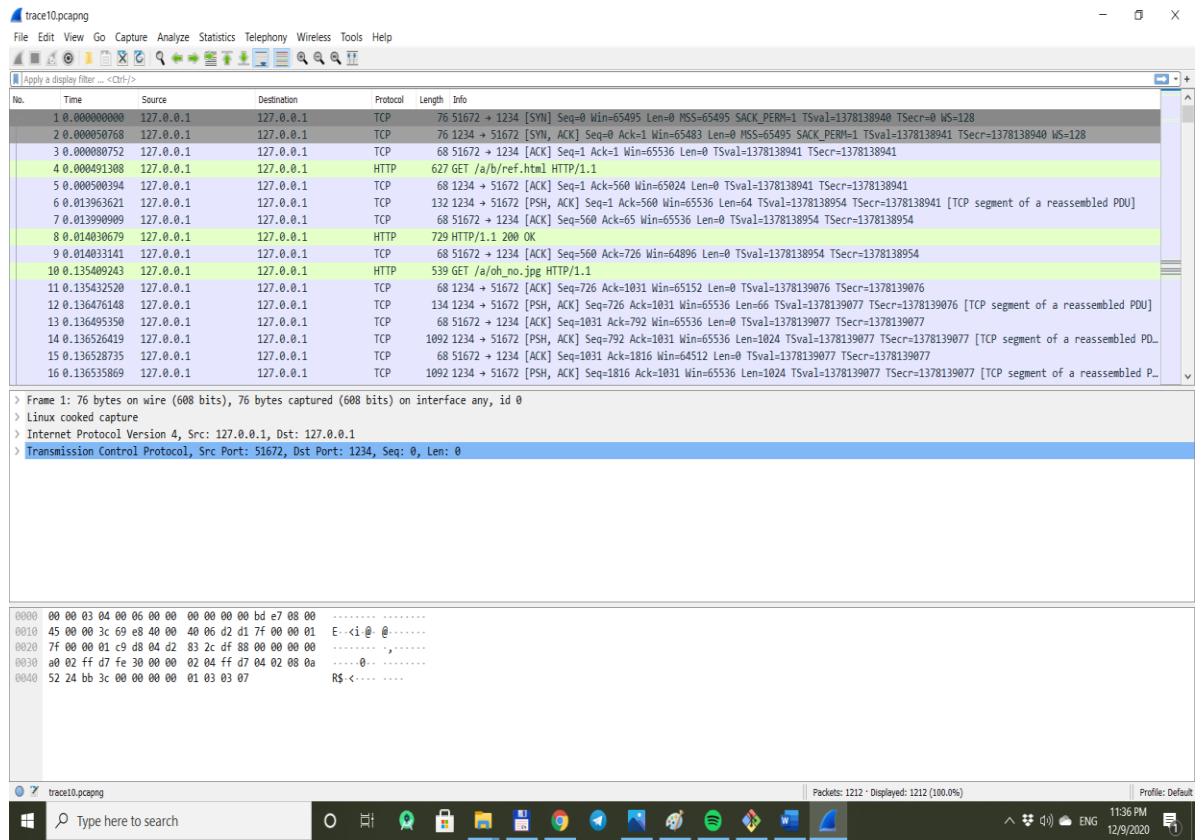
f. Now for the last scenario – type URI "/a/b/ref.html" into chrome browser

First we see that there was a hands shaking between the server and the browser(the client), only one socket opened and the client turned on the SYN flag and then ACK,SYN flags were turned on in the message from server

Now lets type run the server and type the URI in the chrome browser :



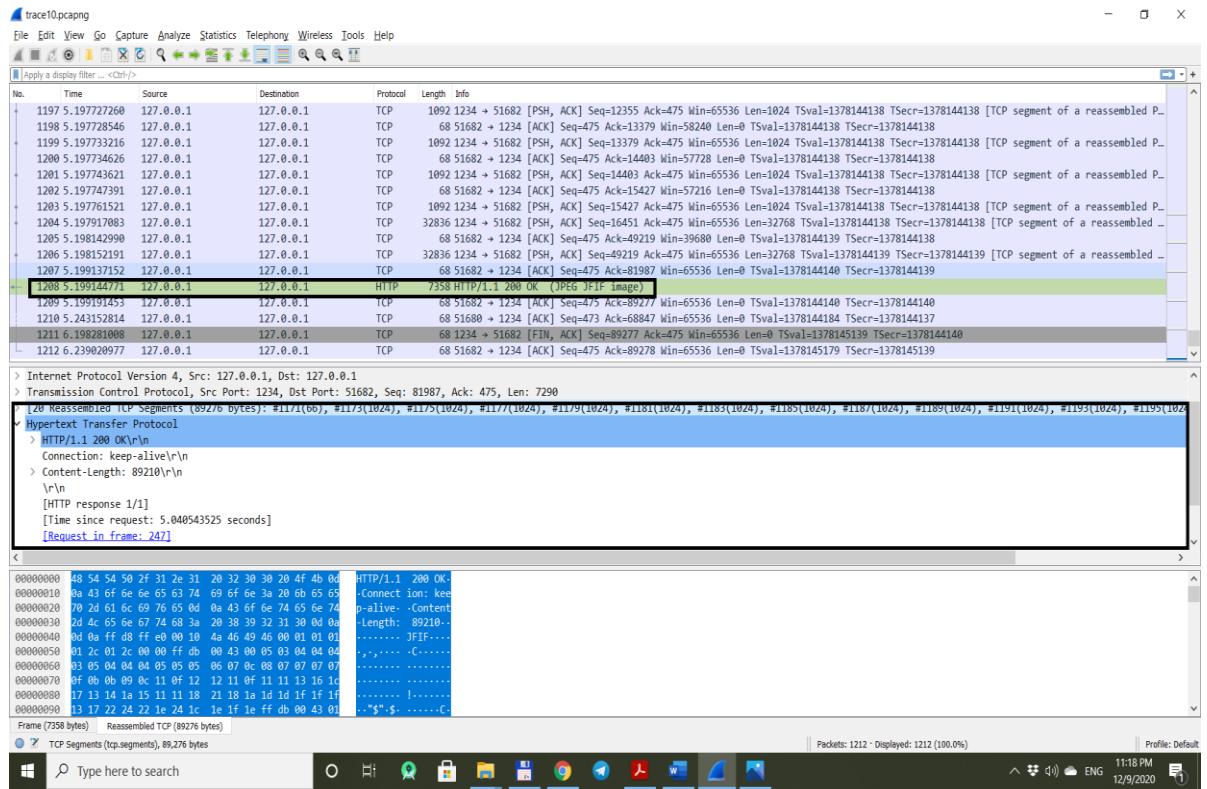
We can see here that all the content that was referenced in the "ref.html" file was shown in the browser. Lets check the wireshark sniff(trace 10.pcap)



We can see that the handshaking was in the first three packets and then the client sent the GET message in HTTP format to the server , requesting the "ref.html" content through the specified URI(packet 4).

We can see in the sniff that during the conversation 1212 packets were sent back and forth between the client and the server and we explain that amount because each picture cant be sent in one segment so the TCP divided the data to multiple segments and the TCP at the client's side has to form the segments back together to one picture.

Lets take a look at packet 1208 for example:



We can see here that after the specific picture was sent completely to the server , the server sent the "200 OK" message in HTTP format to the client and the client responded with ACK message, which means that the server can move on to sent the next picture.

At the two last packets we see that the server sent a message with FIN,ACK flags turned on meaning that the server has finished to send everything and wish to close the socket, after that the client responded with ACK message about closing the socket and then the conversation ended and the socket was closed.

- If we go to statistics tab->conversation we will see that everything here occurred in one conversation (socket)