Universidad de León
School of Industrial, Computer and Aerospace Engineering
Course on Distributed Systems and Networks

Reference Solution to Homework #2. Basic TCP.

Submit a single .zip file containing your solutions to the HW exercises. Only .pdf and files are accepted. Submit via agora by 21:00 on Monday 14th-October-2024.

Introduction

The trace included below represents the lifecycle of a TCP connection between a client host (C) and a server (S). The C and the S processes run in their respective hosts (See fig. 1). Solve the questions included after the trace (The trace is composed of 11 packets which are numbered `P<n>` for convenience).

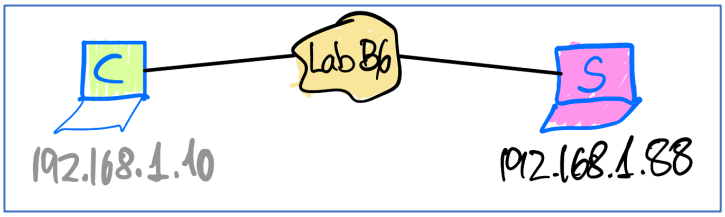

Figure 1. Client and Server hosts in Lab B6 network with their IP addresses

tcpdump trace for TCP Analysis

```
$ /usr/sbin/tcpdump -i eno1 -vvv -n -X tcp port 50001
tcpdump: listening on eno1, link-type EN10MB (Ethernet), capture size 262144 bytes
```

P1

```
14:49:25.354585 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto TCP (6), length 64)
    192.168.1.10.49699 > 192.168.1.88.50001: Flags [SEW], cksum 0x3f42 (correct), seq 1959256602, win 65535,
options [mss 1460,nop,wscale 6,nop,nop,TS val 1470971714 ecr 0,sackOK,eol], length 0
         0x0000:  4500 0040 0000 4000 4006 b705 c0a8 010a  E..@..@.@.......
         0x0010:  c0a8 0158 c223 c351 74c7 e21a 0000 0000  ...X.#.Qt.......
         0x0020:  b0c2 ffff   3f42 0000 0204 05b4 0103 0306  ....?B..........
         0x0030:  0101 080a 57ad 3f42 0000 0000 0402 0000  ....W.?B........
```

P2

```
14:49:25.354631 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto TCP (6), length 60)
    192.168.1.88.50001 > 192.168.1.10.49699: Flags [S.E], cksum 0x83e1 (incorrect -> 0x00f0), seq 3404803869, ack
1959256603, win 65160, options [mss 1460,sackOK,TS val 4113457662 ecr 1470971714,nop,wscale 7], length 0
         0x0000:  4500 003c 0000 4000 4006 b709 c0a8 0158  E..<..@.@......X
         0x0010:  c0a8 010a c351 c223 caf1 2f1d 74c7 e21b  .....Q.#../.t...
         0x0020:  a052 fe88 83e1 0000 0204 05b4 0402 080a  .R..............
         0x0030:  f52e 61fe 57ad 3f42 0103 0307            ..a.W.?B....
```

P3

```
14:49:25.354906 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto TCP (6), length 52)
    192.168.1.10.49699 > 192.168.1.88.50001: Flags [.], cksum 0x267b (correct), seq 1, ack 1, win 2058, options
[nop,nop,TS val 1470971714 ecr 4113457662], length 0
         0x0000:  4500 0034 0000 4000 4006 b711 c0a8 010a  E..4..@.@.......
         0x0010:  c0a8 0158 c223 c351 74c7 e21b caf1 2f1e  ...X.#.Qt...../.
         0x0020:  8010 080a 267b 0000 0101 080a 57ad 3f42  ....&{......W.?B
         0x0030:  f52e 61fe                                ..a.
```

P4


```
14:49:25.354988 IP (tos 0x2,ECT(0), ttl 64, id 0, offset 0, flags [DF], proto TCP (6), length 67)
    192.168.1.10.49699 > 192.168.1.88.50001: Flags [P.], cksum 0x5148 (correct), seq 1:16, ack 1, win 2058, options
[nop,nop,TS val 1470971714 ecr 4113457662], length 15
        0x0000:  4502 0043 0000 4000 4006 b700 c0a8 010a  E..C..@.@.......
        0x0010:  c0a8 0158 c223 c351 74c7 e21b caf1 2f1e  ...X.#.Qt...../.
        0x0020:  8018 080a 5148 0000 0101 080a 57ad 3f42  ....QH......W.?B
        0x0030:  f52e 61fe 4865 6c6c 6f20 776f 726c 6420  ..a.Hello.world.
        0x0040:  3a2d 29                                  :-)
```

P5

```
14:49:25.355008 IP (tos 0x0, ttl 64, id 12405, offset 0, flags [DF], proto TCP (6), length 52)
    192.168.1.88.50001 > 192.168.1.10.49699: Flags [.], cksum 0x83d9 (incorrect -> 0x2c79), seq 1, ack 16, win 509,
options [nop,nop,TS val 4113457662 ecr 1470971714], length 0
        0x0000:  4500 0034 3075 4000 4006 869c c0a8 0158  E..40u@.@......X
        0x0010:  c0a8 010a c351 c223 caf1 2f1e 74c7 e22a  .....Q.#../.t..*
        0x0020:  8010 01fd 83d9 0000 0101 080a f52e 61fe  ..............a.
        0x0030:  57ad 3f42                                W.?B
```

P6

```
14:49:25.355097 IP (tos 0x2,ECT(0), ttl 64, id 12406, offset 0, flags [DF], proto TCP (6), length 92)
    192.168.1.88.50001 > 192.168.1.10.49699: Flags [P.], cksum 0x8401 (incorrect -> 0xaa70), seq 1:41, ack 16, win
509, options [nop,nop,TS val 4113457662 ecr 1470971714], length 40
        0x0000:  4502 005c 3076 4000 4006 8671 c0a8 0158  E..\0v@.@..q...X
        0x0010:  c0a8 010a c351 c223 caf1 2f1e 74c7 e22a  .....Q.#../.t..*
        0x0020:  8018 01fd 8401 0000 0101 080a f52e 61fe  ..............a.
        0x0030:  57ad 3f42 536f 6c63 6974 7564 206e 6f20  W.?BSolcitud.no.
        0x0040:  7265 636f 6e6f 6369 6461 2070 6f72 2065  reconocida.por.e
        0x0050:  7374 6520 7365 7276 6964 6f72            ste.servidor
```

P7

```
14:49:25.355131 IP (tos 0x0, ttl 64, id 12407, offset 0, flags [DF], proto TCP (6), length 52)
    192.168.1.88.50001 > 192.168.1.10.49699: Flags [F.], cksum 0x83d9 (incorrect -> 0x2c50), seq 41, ack 16, win
509, options [nop,nop,TS val 4113457662 ecr 1470971714], length 0
        0x0000:  4500 0034 3077 4000 4006 869a c0a8 0158  E..40w@.@......X
        0x0010:  c0a8 010a c351 c223 caf1 2f46 74c7 e22a  .....Q.#../Ft..*
        0x0020:  8011 01fd 83d9 0000 0101 080a f52e 61fe  ..............a.
        0x0030:  57ad 3f42                                W.?B
```

P8

```
14:49:25.355447 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto TCP (6), length 52)
    192.168.1.10.49699 > 192.168.1.88.50001: Flags [.], cksum 0x2644 (correct), seq 16, ack 41, win 2058, options
[nop,nop,TS val 1470971714 ecr 4113457662], length 0
        0x0000:  4500 0034 0000 4000 4006 b711 c0a8 010a  E..4..@.@.......
        0x0010:  c0a8 0158 c223 c351 74c7 e22a caf1 2f46  ...X.#.Qt..*../F
        0x0020:  8010 080a 2644 0000 0101 080a 57ad 3f42  ....&D......W.?B
        0x0030:  f52e 61fe                                ..a.
```

P9

```
14:49:25.355464 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto TCP (6), length 52)
    192.168.1.10.49699 > 192.168.1.88.50001: Flags [.], cksum 0x2643 (correct), seq 16, ack 42, win 2058, options
[nop,nop,TS val 1470971714 ecr 4113457662], length 0
        0x0000:  4500 0034 0000 4000 4006 b711 c0a8 010a  E..4..@.@.......
        0x0010:  c0a8 0158 c223 c351 74c7 e22a caf1 2f47  ...X.#.Qt..*../G
        0x0020:  8010 080a 2643 0000 0101 080a 57ad 3f42  ....&C......W.?B
        0x0030:  f52e 61fe                                ..a.
```

P10

```
14:49:25.355493 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto TCP (6), length 52)
    192.168.1.10.49699 > 192.168.1.88.50001: Flags [F.], cksum 0x2641 (correct), seq 16, ack 42, win 2058, options
[nop,nop,TS val 1470971715 ecr 4113457662], length 0
        0x0000:  4500 0034 0000 4000 4006 b711 c0a8 010a  E..4..@.@.......
        0x0010:  c0a8 0158 c223 c351 74c7 e22a caf1 2f47  ...X.#.Qt..*../G
        0x0020:  8011 080a 2641 0000 0101 080a 57ad 3f43  ....&A......W.?C
        0x0030:  f52e 61fe                                ..a.
```

P11

```
14:49:25.355509 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto TCP (6), length 52)
    192.168.1.88.50001 > 192.168.1.10.49699: Flags [.], cksum 0x2c4d (correct), seq 42, ack 17, win 509, options
[nop,nop,TS val 4113457663 ecr 1470971715], length 0
        0x0000:  4500 0034 0000 4000 4006 b711 c0a8 0158  E..4..@.@......X
        0x0010:  c0a8 010a c351 c223 caf1 2f47 74c7 e22b  .....Q.#../Gt..+
        0x0020:  8010 01fd 2c4d 0000 0101 080a f52e 61ff  ....,M........a.
        0x0030:  57ad 3f43                                W.?C
```

Questions about the trace above

1. Which packets make up the 3-way handshake phase of the connection?

   P1, P2 and P3.

2. What is the client's IP address and TCP port?

   Look at P1, where the client sending a TCP that has the S flag set, meaning that a new connection is being requested. The client IP address is the source IP address contained in the IP packet that encapsulates this segment.

3. Likewise, what is the server's IP address and TCP port?

   Look at P2, the segment sent back by the server, its encapsulating IP packet's source IP address is the server IP address.

4. What is the ISN (Initial Sequence Number) specified by the client? In which packet do you identify it?

   The ISN is carried by P1. Look at its SN field, which at this time, when the flag S has been set means that the SN field contains the ISN, the initial sequence number randomly chosen by the client. The tcpdump utility prints out the ISN in a text identified with the label "seq":

   seq 1959256602

5. What is the ISN (Initial Sequence Number) specified by the server? In which packet do you identify it?

   Look at P2. The server's ISN is: `seq 3404803869`. *Note that the SN (sequence numbers) printed out by tcpdump after P2 are logical, meaning that instead of starting at the ISN, they start at 0, which helps the user's perception by avoiding the clutter produced by such large, 32-bit integers.*

6. When the server receives the ISN from the client, it should respond with a TCP segment that has the SYN flag set and an ACK SN that should one plus the received ISN. Check that this holds in the relevant packets. Which packets are those?

   P1 and P2:
   P1 (seq 1959256602)
   P2 (ack 1959256603)

7. What is the purpose of packet P3?

   In P3 the client acknowledges the ISN randomly and independently generated by the server: `ack 1`. *The ACK SN field contains the actual 32-bit value, which can be read out from the hex dump, by interpreting it*

according to the TCP header structure published in the RFC to the TCP protocol, as explained above, however, tcpdump prints out logical SN. In this case the logical value contained in the ACK SN is 1.

8. Packet P4 is used by the client to send a block of bytes which SN covers the range 1:15. Is that true?

It is true. The field seq indicates 1:16, which means that in this segment the range of SN covered by the payload is from 1 to (16 – 1) and that the ACK SN that should acknowledge this block of data should have an ACK SN of 16 whenever it is sent by the server.

*seq 1:**16***
*length **15***

9. What packets sent by the server acknowledge the block of data mentioned in the preceding question? List them all.

Packets P5, P6, P7 and P11, all of them cumulatively acknowledge the mentioned block of data

10. What's the purpose of sending segments with no payload, if any? Set an example of one of such segments?

All of the packets which length is 0, carry no payload data. Their purpose consists of informing the other side about their TCP's state, somehow. For example: packet P5 carries no payload; it simply acknowledges the correct, in-order reception of all of the bytes from logical sn 1 to sn 15 and that the next in-order byte expected is 16. In this segment the server's TCP also informs the client's that its offered window size (Adevrtised Window Size) is 509 bytes.

11. Calculate the MTU of the underlying Ethernet, by hand.

That calculation can only be done by using the fields in packets P1 and P2, for the client and for the server, respectively. I'll do the MTU for the client by using P1:

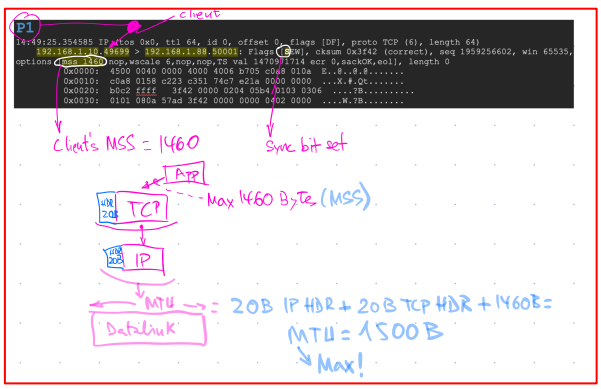

Figure 1. MTU calculation from MSS offered by client in the 3-way handshake

12. Identify the first *small segment* that appears on the data transfer phase included in the trace (A small segment is one that has a size less than MSS?

P4 is the first such segment appearing on the given trace. Its payload weighs 15 bytes which SNs span 1 to 15.

13. Check that packet P9 has its ACK flag set. Use the hex dump for identifying the position of the ACK bit. You will want to consult the TCP header from RFC 9293.

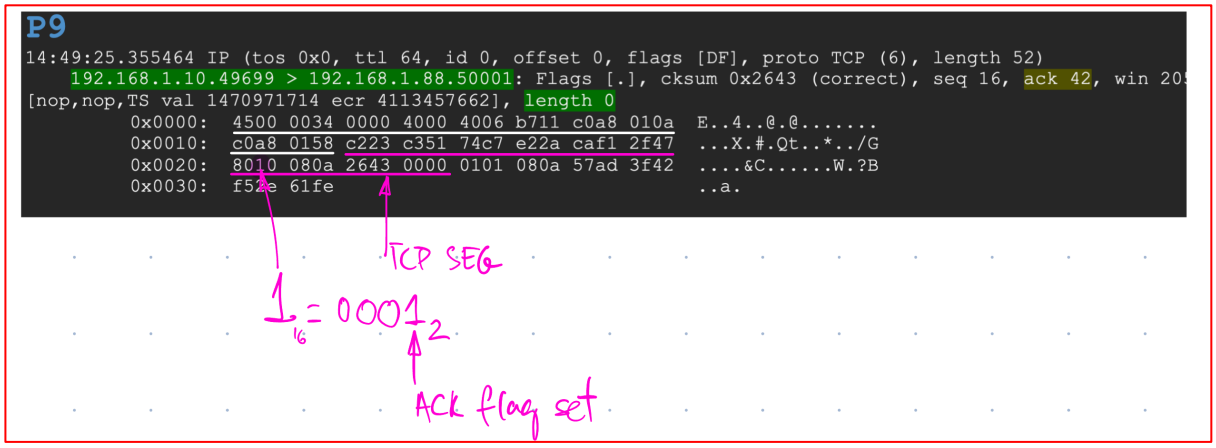

Figure 2. ACK bit set in TCP segment

14. Check whether all of the segments that belong to the data transfer phase have their ACK flag set.

All of packets P3, P5, etc. do have their ACK bit set.

15. Identify some data transfer segments that have their SYN flag set

None of the packets comprising the data transfer phase should have their SYN flag set, and it can be readily checked by reading the tcpdump informational text where, in none of them do we see that bit set, in other words, in none of them will we see the letter S in the informational text

16. Which flags are set in the TCP segment which IP packet was sent in P11.

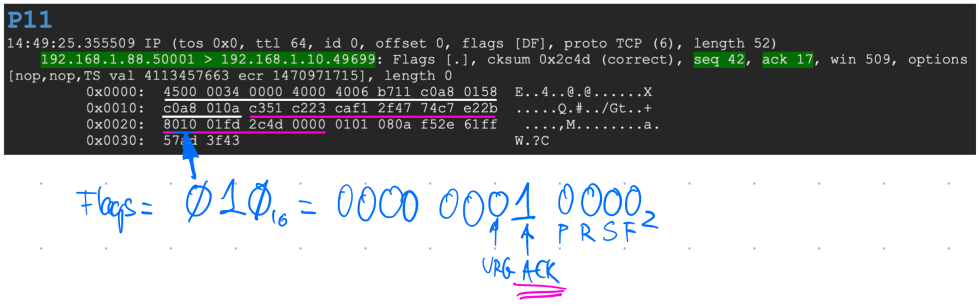

Figure 3. Flags of segment in P11

Comments and overall guidance to DS homework #2 (2024)

To gain some insight into how the protocol stack processes IP packets that carry TCP payloads, we'll analyze packet P4 from homework #2 (2024). This brief analysis is meant to improve your confidence as you progress with hw#2.

Analysis of trace packet number P4

For convenience, I'm including a sketch of packet P4 along with the two standard protocol headers encapsulated. Notice that the options included in the tcpdump command line preclude the Ethernet header, which indeed was sent along with the IP and TCP headers. The protocol stack is this:

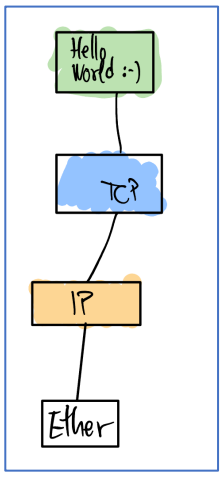

Figure 1. Protocol stack to P4 from hw#2 trace.

The protocol stack can be deduced from the tcpdump trace. Again, it is necessary that we recall that the Ethernet header has been omitted from the tcpdump parameters such that the output not be cluttered. Consequently, the first group of bytes constitute the IP packet header, followed by the TCP header. The last bytes correspond to the TCP payload submitted by the application protocol "Hello World :-)".

Explanation of the IP packet header. In fig. 2 you can find the tcpdump trace that we explain here. The header structures to IP and TCP PDUs from the relevant RFCs are included here for convenience.

Look at the P4 tcpdump trace, now. You'll identify the IP packet first there, which is highlighted in yellow. Let's see how long the IP header is by looking at its IHL field. This field spans bits 4-7 at from the *Internet Datagram Header* (On the upper area of fig. 2). The IHL value is 5, according to the hexadecimal value 45 that appears on the trace. This means that the IHL (Internet Header Length) is:

```
5 x 4 Bytes = 20 Bytes
```

Consequently, after our simple calculation, the IP header weights 20 Bytes spanning byte indexes 0x0000 to 0x0013 on the trace. This area has been highlighted in light yellow. 20 Bytes is the IP header size that will show up most frequently in the practicals and in the homework in DS and in CN.

So far, we have successfully delimited the IP header. Now, it's time for us to interpret the information carried in it. Simply, read the tcpdump *explaining* text included in the trace:

- Protocol encapsulated: TCP
  `( proto TCP (6) )`
- Let's check that that is true by interpreting the HEX dump:
  `According to IP header structure from the IP RFC, the Protocol field spans Byte number 9. The value contained in that byte number is: 06.`
- Yes, the protocol number is 06, which according to the official protocol number database is: TCP (In Linux /etc/services). That is what we expected.

Explanation of the TCP segment header. Now we turn our attention to the TCP segment header where we want to prove that **P4**'s TCP segment does have its ACK flag set. This fact cannot be derived from the explaining text produced by tcpdump. The relevant area in the tcpdump trace is highlighted in light blue, however, we won't provide a full demonstration that that is the area spanned by the segment. Nevertheless, you can rely on the relevant calculations which have been carefully made, though not provided here. For our immediate purpose of checking the ACK flag, none of those calculations are necessary since the TCP segment will span at least the initial 20 Bytes in the TCP header and the ACK flag is always to be found therein.

Turn your attention to the *TCP Header Format,* at the lower area of fig. 2. Identify the ACK flag in the row that starts at byte number 12. The relevant bits are expanded on the handwritten diagram that is located at the right side of *TCP Header Format.* You identify the ACK flag by counting bits starting at byte number 12. Recall that each row from the TCP header spans 32 bits (4 bytes) and the ACK is on the fourth row (Rows numbering starts at 0), thereby starting at Byte 12 = (4 – 1) * 4 = 3 * 4 = 12.

Not only should ACK be set in this segment, but it should be set in all the segments comprising the trace, except the initial segment, P0, in which the client is sending a connection request to the server. That specific segment should never have its ACK set. Since it is the initial segment, it cannot acknowledge any segment received in the past, because there were no past segments; consequently, ACK cannot not significant and it is not set. As a simple exercise, check that ACK is actually not set in segment encapsulated into **P0** (See fig. 3).

Finally, observe, as well in P4, that the payload encapsulated into TCP is 15 Bytes long (length 15). And that those 15 bytes are indexed with sequence numbers 1 to 15 (Look for tcpdump text: `seq 1:16`). The meaning of that text is that the SN that applies to the first byte is 1, the last one being (16 - 1). Index 16 means that byte index 16 is the next *in-order* byte that will be sent by this sender. 16, will, in the future have to the the ACK SN value that the receiver will have to send back as the accumulated, positive acknowledgment to the told byte block (1 to 15).

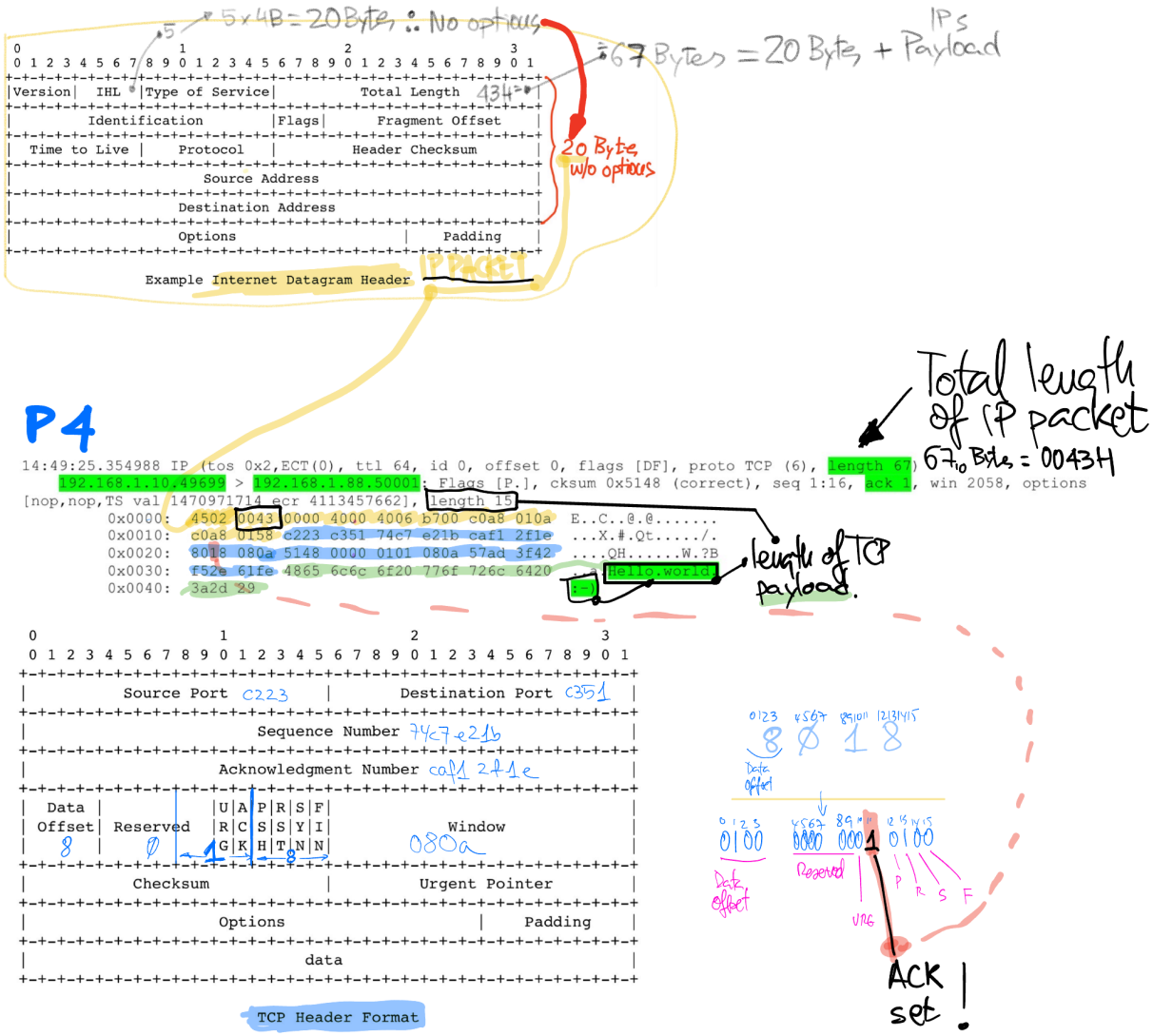

Figure 2. Packet **P4** from trace alongside the IP and TCP headers from RFCs.

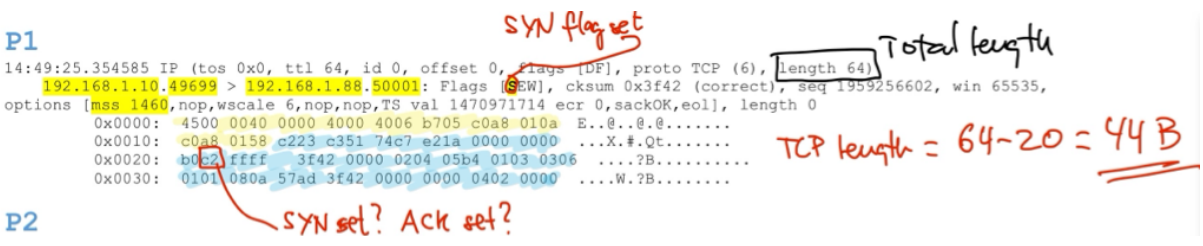

Figure 3. Analysis of **P1** (The conn request TCP segment)

