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:

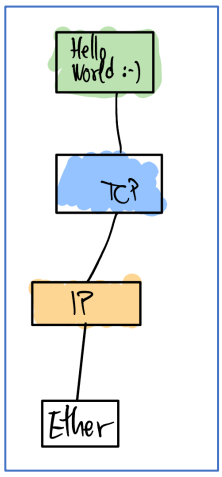

Figure 1. Protocol stack to P4 from hw#2 trace.

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 highlited 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).

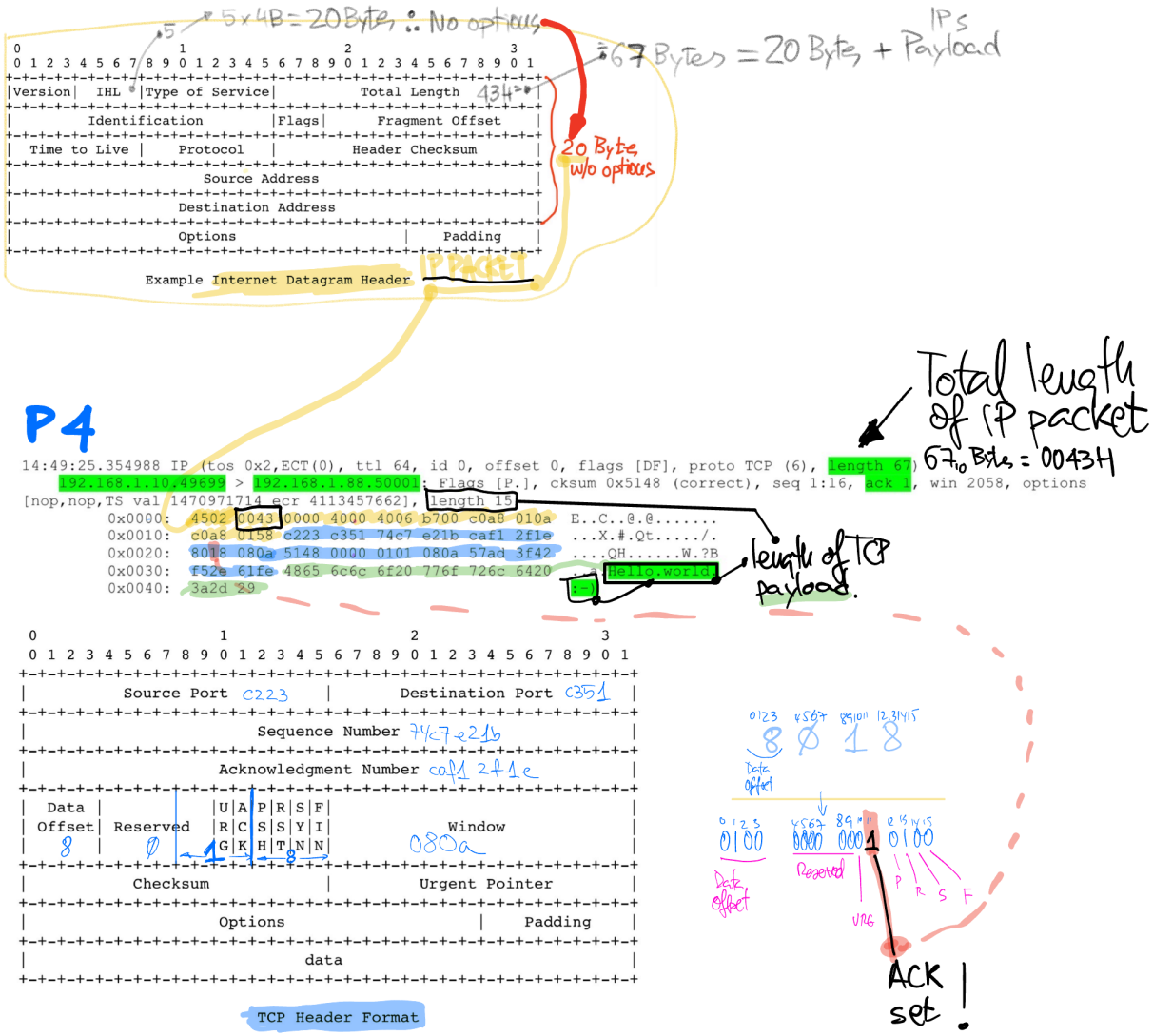

Figure 2. Packet **P4** from trace alongside the IP and TCP headers from RFCs.

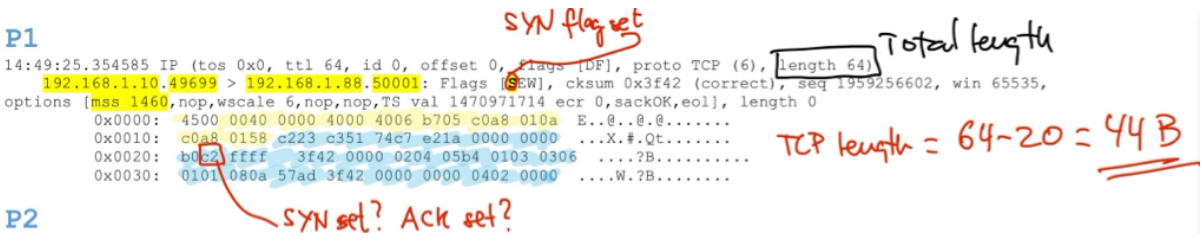

Figure 3. Analysis of **P1** (The conn request TCP segment)