Experiment 3

Question 1

Please illustrate the process of a packet being delivered from h1 through r1 to h2 with regard to these aspects:

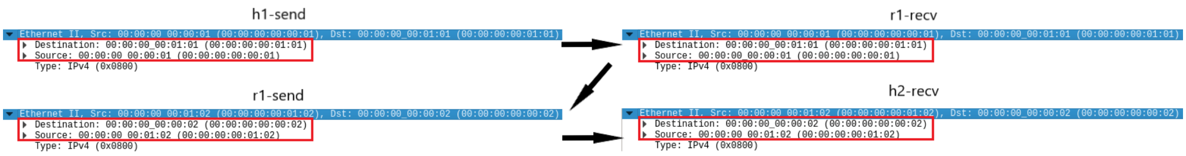
- How MAC addresses change
- How IP addresses change

Hint: View Linux cooked as encryption

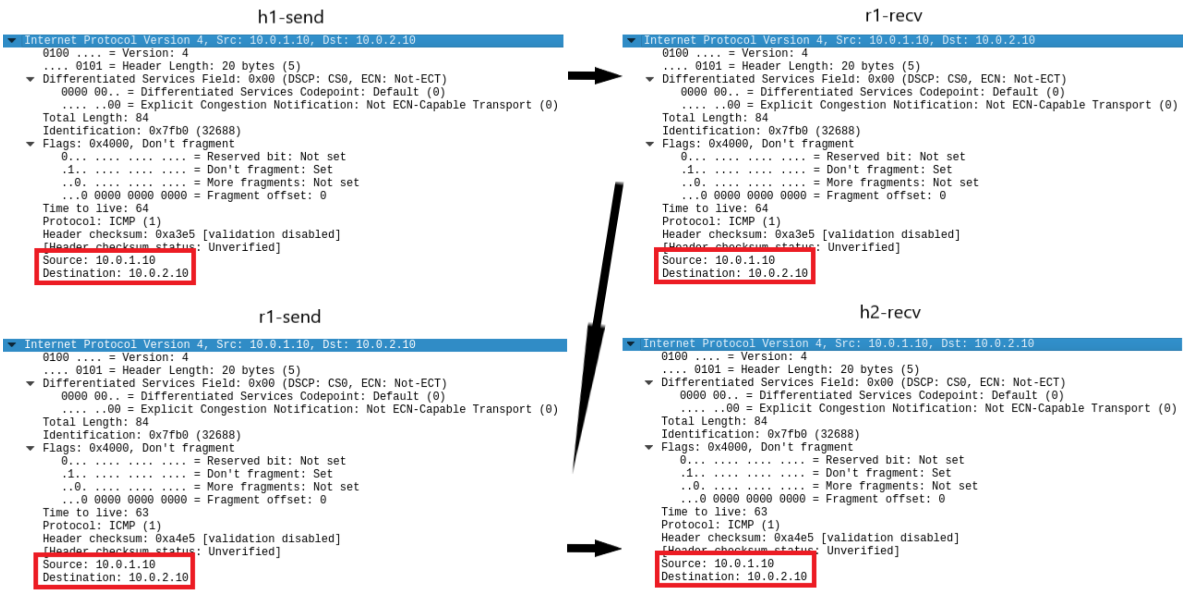
Answer 1

Non-Linux cooked capture mode

How MAC changes:



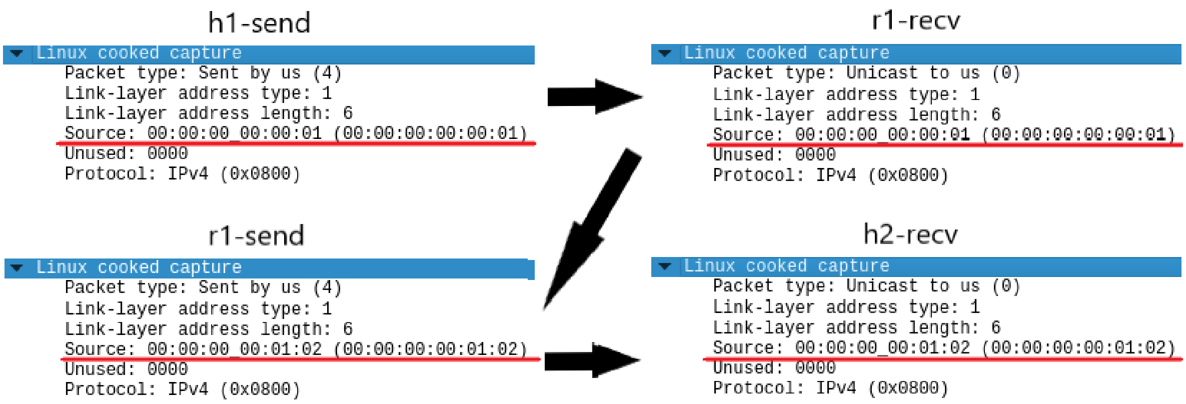
How IP changes:



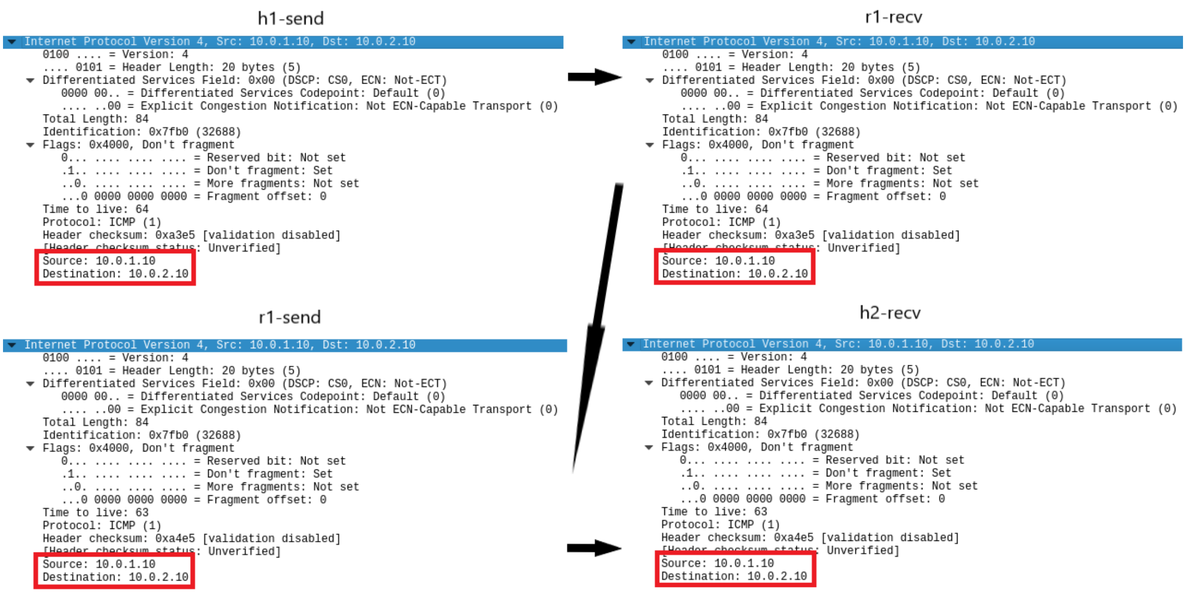
	h1	r1-recv	r1-send	h2
Source MAC	00:00:00:00:00:01	00:00:00:00:00:01	00:00:00:00:01:02	00:00:00:00:01:02
Destination MAC	00:00:00:00:01:01	00:00:00:00:01:01	00:00:00:00:00:02	00:00:00:00:00:02
Source IP	10.0.1.10	10.0.1.10	10.0.1.10	10.0.1.10
Destination IP	10.0.2.10	10.0.2.10	10.0.2.10	10.0.2.10

Linux cooked capture mode

How MAC changes:



How IP changes:



	h1	r1-recv	r1-send	h2
Source MAC	00:00:00:00:00:01	00:00:00:00:00:01	00:00:00:00:01:02	00:00:00:00:01:02
Source IP	10.0.1.10	10.0.1.10	10.0.1.10	10.0.1.10
Destination IP	10.0.2.10	10.0.2.10	10.0.2.10	10.0.2.10

Question 2

Have IP addresses in the packet changed during h1 -> r1 -> h2?

Please describe how the IP header has changed during h1 -> r1 -> h2 with regard to:

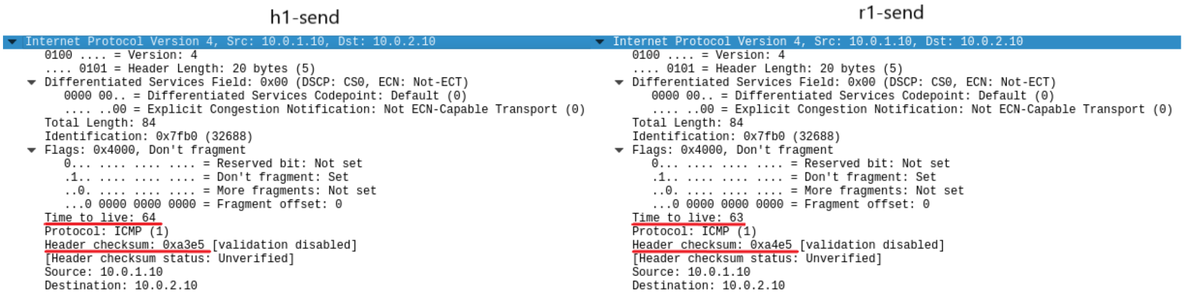
- TTL
- ID
- Fragmentation information

Answer 2

IP addresses did not change during the transmission of h1 -> r1 -> h2, as is displayed in the table above.

This problem can be simplified into comparing the difference of the IP header between h1-send and r1-send. This is because the packet does not change in transmission, meaning that h1-send and r1-recv is the same, and r1-send and h2-recv is the same.

The difference:



The IP header's TTL has been degraded by one and the header checksum was recomputed. Nothing else changed.

Question 3

How does Wireshark know whether the packet is Linux cooked or not?

Answer 3

Citing from https://wiki.wireshark.org/SLL : "When capturing from the 'any' device, or from one of those other devices, in Linux, the libpcap doesn't supply the link-layer header for the real 'hardware protocol' like [Ethernet](), but instead supplies a fake link-layer header for this pseudo-protocol."

From answer 1, we can also see that when setting Wireshark to capture packets on `any`, the link-layer header changes from `Ethernet` to `Linux cooked capture`. This is because capturing in `cooked mode` is done by reading from a PF_PACKET/SOCK_DGRAM socket rather than the more normally used PF_PACKET/SOCK_RAW socket. When using the PF_PACKET/SOCK_DGRAM socket, Linux socket code doesn't supply the packet's link-layer header, so libpcap constructs a synthetic link-layer header from the address supplied.

Finally, in regard to the question asked, if Wireshark is capturing on the `any` device or on any network interface where libpcap uses cooked mode, **all** traffic is SLL traffic.

Question 4

What is the size of the data part for IP? Note that the IP header does not count as data.

Considering that ICMP also has it's header, what is the size of the data part for ICMP?

Answer 4

The size of the data part for IP is 64.

This can be counted in two ways:

1. IP Total Length - IP Header Length
2. ICMP Total Length

IP Aspect of Data Length

```
▼ Internet Protocol Version 4, Src: 10.0.1.10, Dst: 10.0.2.10
      0100 .... = Version: 4
      .... 0101 = Header Length: 20 bytes (5)
    ▼ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
         0000 00.. = Differentiated Services Codepoint: Default (0)
         .... ..00 = Explicit Congestion Notification: Not ECN-Capable Transport (0)
      Total Length: 84
      Identification: 0x7fb0 (32688)
    ▼ Flags: 0x4000, Don't fragment
         0... .... .... .... = Reserved bit: Not set
         .1.. .... .... .... = Don't fragment: Set
         ..0. .... .... .... = More fragments: Not set
         ...0 0000 0000 0000 = Fragment offset: 0
      Time to live: 64
      Protocol: ICMP (1)
      Header checksum: 0xa3e5 [validation disabled]
      [Header checksum status: Unverified]
      Source: 10.0.1.10
      Destination: 10.0.2.10
```

Thus we have IP data length = 84 - 20 = 64.

ICMP Aspect of Data Length

```
▶ Internet Control Message Protocol

0000  00 04 00 01 00 06 00 00  00 00 00 01 00 00 08 00   ........ ........
0010  45 00 00 54 2d fa 40 00  40 01 f5 9b 0a 00 01 0a   E··T-·@· @·······
0020  0a 00 02 0a 08 00 3c af  09 bc 00 01 97 db a2 5d   ······<· ·······]
0030  00 00 00 00 b8 87 00 00  00 00 00 00 10 11 12 13   ········ ········
0040  14 15 16 17 18 19 1a 1b  1c 1d 1e 1f 20 21 22 23   ········ ···· !"#
0050  24 25 26 27 28 29 2a 2b  2c 2d 2e 2f 30 31 32 33   $%&'()*+ ,-./0123
0060  34 35 36 37                                        4567
```

One line contains 16 bytes, and ICMP takes up 4 lines.

Thus we IP data length = ICMP length = 16 * 4 = 64.

ICMP Data Length

The size of the data part for ICMP is 56 bytes, as the two timestamps are actually data too, and they account for 8 bytes of data, resulting in 48 + 8 = 56 bytes.

```
▼ Internet Control Message Protocol
      Type: 8 (Echo (ping) request)
      Code: 0
      Checksum: 0x3caf [correct]
      [Checksum Status: Good]
      Identifier (BE): 2492 (0x09bc)
      Identifier (LE): 48137 (0xbc09)
      Sequence number (BE): 1 (0x0001)
      Sequence number (LE): 256 (0x0100)
      [Response frame: 4]
      Timestamp from icmp data: Oct 13, 2019 01:08:55.000000000 PDT
      [Timestamp from icmp data (relative): 0.034776021 seconds]
    ▶ Data (48 bytes)
```

Question 5

How many packets are sent from h2 to h1?

What are the sizes of each of these packets?

Answer 5

```
[4 IPv4 Fragments (5008 bytes): #10(1480), #11(1480), #12(1480), #13(568)]
    [Frame: 10, payload: 0-1479 (1480 bytes)]
    [Frame: 11, payload: 1480-2959 (1480 bytes)]
    [Frame: 12, payload: 2960-4439 (1480 bytes)]
    [Frame: 13, payload: 4440-5007 (568 bytes)]
    [Fragment count: 4]
    [Reassembled IPv4 length: 5008]
    [Reassembled IPv4 data: 000018e710fe0001d829a35d000000006ef5010000000000...]
```

4 packets are sent from h2 to h1.

The sizes of these packets are 1480, 1480, 1480 and 568.

Question 6

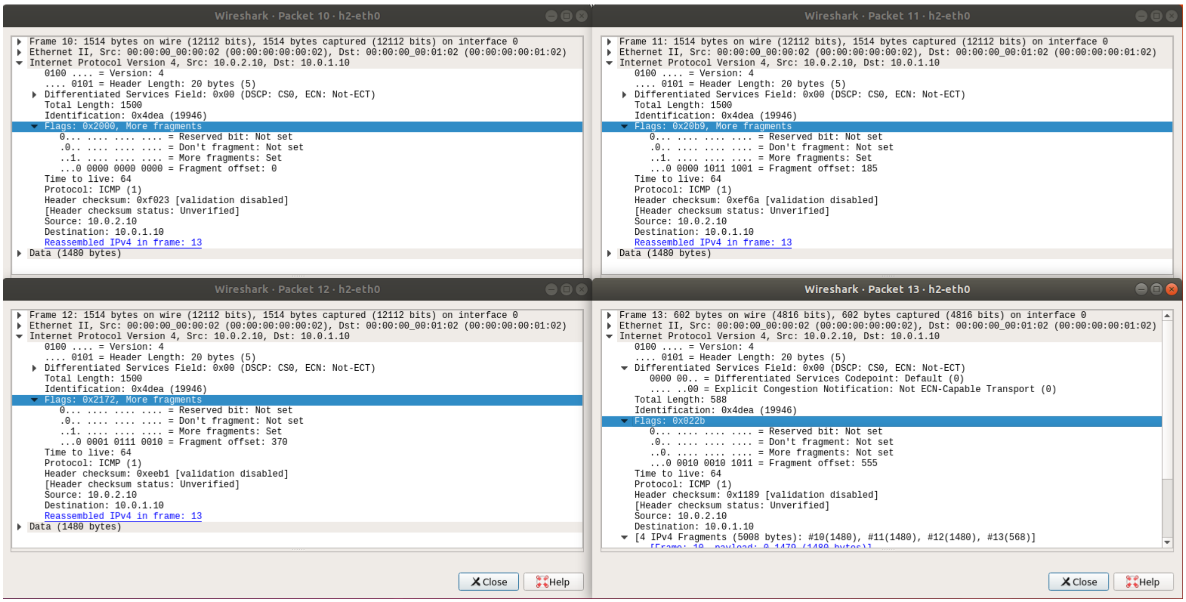
What is each fragment's flags?

What is each fragment's offset and its meaning?

What is the total length of these fragments?

Why is their total length bigger than 5000?

Answer 6



Flags

	Packet 10	Packet 11	Packet 12	Packet 13
Reserved bit	Not set	Not set	Not set	Not set
Don't fragment	Not set	Not set	Not set	Not set
More fragments	Set	Set	Set	Not set

Offset

	Packet 10	Packet 11	Packet 12	Packet 13
Offset	0	185	370	555

Offset stands for the starting point of a fragment in regard to the original over-sized packet. To be more specific, offset `0` for `Packet 10` means that this fragment starts at the 0 * 8 = 0 byte in the original packet; offset `185` for `Packet 11` means that this fragment starts at the 185 * 8 = 1480 byte in the original packet, which lines up with the `1480` data payload delivered by `Packet 10`; offset `370` for `Packet 12` means that this fragment starts at the 370 * 8 = 2960 byte in the original packet, which lines up with the `2960` data payload delivered by `Packet 10` and `Packet 11`, and so on.

Total length

Total length of these fragments is 1500 + 1500 + 1500 + 588 = 5088 bytes.

Why

Since each fragment has its own IP header, 4 * 20 = 80 bytes is used by these headers.

The other 8 bytes come from the ICMP header.