

Q.1 — Configuration of the device group can best be understood in terms of what is happening “under the hood” (*i.e.*, beneath the surface, or behind the interface which the GUI provides). Describe the activities “under the hood” in terms which you have learnt in your study of the OSI 7-layer model.

Answer

So, firstly, it is critical to notice that this completely depends on what Ostinato does “under the hood”. Specifically, I would like to point out that Ostinato itself is an application running in a Linux virtual machine which is virtually connected to a docker container on the same machine. This of course introduces another level of complexity which I do not believe is feasible to consider here.

Also, I believe it is also not feasible to describe what Ostinato precisely does internally as it requires an in depth knowledge of the Ostinato codebase.

Having said this, I will restrict myself to a very shallow explanation.

The two main OSI layers which concern the creation of device groups are the Data Link (2nd) Layer and the Network (3rd) Layer.

When we create a device group we are forced to give that device group a base MAC (Media Access Control) address. We have to pick also the number of devices which the device group will have. In the case that there are more than one devices, Ostinato will give the rest of the devices MAC addresses based on an address offset which we specify. The MAC addresses are then used by Ostinato to distinguish between these virtual devices at layer 2.

Note: The uniqueness of the MAC addresses used is responsibility of the individual using Ostinato, although Ostinato seems to provide random MAC address which are unlikely to be already in use.

After we provide MAC addresses to the devices in the device group we also have the option of choosing an Internet Protocol (IP) version and provide a base IP address and an address offset to give each device a different IP address. This essentially sets up the virtual devices for layer 3 functionality.

I think the above is a sufficient answer as delving into the actual details of how these things are implemented is not trivial and would require an incredibly large amount of work.

Q.2.1 — Count the number of packets that you have captured and compare this with the setting in the stream configured on the packet generator.

Answer

Table 1: Wireshark ICMP capture for Question 2.1.

No.	Time	Source	Destination	Protocol	Length	Info
486	4617.638757	192.168.0.10	192.168.0.1	ICMP	60	Echo (ping) request id=0x04d2, seq=0/0, ttl=127 (reply in 487)
487	4617.638893	192.168.0.1	192.168.0.10	ICMP	60	Echo (ping) reply id=0x04d2, seq=0/0, ttl=64 (request in 486)
488	4618.638764	192.168.0.10	192.168.0.1	ICMP	60	Echo (ping) request id=0x04d2, seq=0/0, ttl=127 (reply in 489)
489	4618.638848	192.168.0.1	192.168.0.10	ICMP	60	Echo (ping) reply id=0x04d2, seq=0/0, ttl=64 (request in 488)
490	4619.638761	192.168.0.10	192.168.0.1	ICMP	60	Echo (ping) request id=0x04d2, seq=0/0, ttl=127 (reply in 491)
491	4619.638880	192.168.0.1	192.168.0.10	ICMP	60	Echo (ping) reply id=0x04d2, seq=0/0, ttl=64 (request in 490)
492	4620.638772	192.168.0.10	192.168.0.1	ICMP	60	Echo (ping) request id=0x04d2, seq=0/0, ttl=127 (reply in 493)
493	4620.638890	192.168.0.1	192.168.0.10	ICMP	60	Echo (ping) reply id=0x04d2, seq=0/0, ttl=64 (request in 492)
494	4621.638731	192.168.0.10	192.168.0.1	ICMP	60	Echo (ping) request id=0x04d2, seq=0/0, ttl=127 (reply in 495)
495	4621.638861	192.168.0.1	192.168.0.10	ICMP	60	Echo (ping) reply id=0x04d2, seq=0/0, ttl=64 (request in 494)
496	4622.638714	192.168.0.10	192.168.0.1	ICMP	60	Echo (ping) request id=0x04d2, seq=0/0, ttl=127 (reply in 497)
497	4622.638914	192.168.0.1	192.168.0.10	ICMP	60	Echo (ping) reply id=0x04d2, seq=0/0, ttl=64 (request in 496)
continued on next page						

Table 1 – continued from previous page

No.	Time	Source	Destination	Protocol	Length	Info
500	4623.638729	192.168.0.10	192.168.0.1	ICMP	60	Echo (ping) request id=0x04d2, seq=0/0, ttl=127 (reply in 501)
501	4623.638837	192.168.0.1	192.168.0.10	ICMP	60	Echo (ping) reply id=0x04d2, seq=0/0, ttl=64 (request in 500)
502	4624.638727	192.168.0.10	192.168.0.1	ICMP	60	Echo (ping) request id=0x04d2, seq=0/0, ttl=127 (reply in 503)
503	4624.638831	192.168.0.1	192.168.0.10	ICMP	60	Echo (ping) reply id=0x04d2, seq=0/0, ttl=64 (request in 502)
504	4625.638735	192.168.0.10	192.168.0.1	ICMP	60	Echo (ping) request id=0x04d2, seq=0/0, ttl=127 (reply in 505)
505	4625.638858	192.168.0.1	192.168.0.10	ICMP	60	Echo (ping) reply id=0x04d2, seq=0/0, ttl=64 (request in 504)
506	4626.638725	192.168.0.10	192.168.0.1	ICMP	60	Echo (ping) request id=0x04d2, seq=0/0, ttl=127 (reply in 507)
507	4626.638833	192.168.0.1	192.168.0.10	ICMP	60	Echo (ping) reply id=0x04d2, seq=0/0, ttl=64 (request in 506)

There are 20 ICMP packets. 10 requests and 10 replies.

Protocol Selection

Protocol Data

Variable Fields

Stream Control

Packet View

Send

☒ Packets
☐ Bursts

Mode

☒ Fixed
☐ Continuous

Numbers

Number of Packets

10

Number of Bursts

1

Packets per Burst

10

Rate

☒ Packets/Sec

1.0000

☐ Bursts/Sec

1.0000

☐ Bits/Sec

672

After this stream

☐ Stop
☒ Goto Next Stream
☐ Goto First

Gaps (in seconds)

ISG

0.0

IBG

0.0

IPG

1.000000000

Figure 1: The stream settings in Ostinato for Question 2.1.

As can be seen in Figure 1 the number of packets, specifically requests, was precisely set to 10.

Q.2.2 — Write down the numbers (leftmost column) of the packets that have been generated by:

- the packet generator
- the DUT

Answer

The required values can be read off Table 1.

486, 488, 490, 492, 494, 496, 500, 502, 504 and 506 are the numbers associated with the request packets generated by Ostinato.

487, 489, 491, 493, 495, 497, 501, 503, 505 and 507 are the numbers associated with the reply packets sent by the DUT.

Q.2.3 — Use your answer to 2.2 to find the average inter-packet delay for the packets generated by the packet generator.

Answer

$$\frac{(4618.638764 - 4617.638757) + (4619.638761 - 4618.638764) + (4620.638772 - 4619.638761) + (4621.638731 - 4620.638772) + (4622.638714 - 4621.638731) + (4623.638729 - 4622.638714) + (4624.638727 - 4623.638729) + (4625.638735 - 4624.638727) + (4626.638725 - 4625.638735)}{10 - 1}$$

= 0.999996444444535 seconds
 ≈ 1 seconds

The above computed value is the average inter-packet delay (in seconds). The approximate value, 1, is precisely what was set in the stream's settings, see Figure 1.

Listing 1: Python expression for calculating the inter-packet delay for Question 2.3.

```
1 ((4618.638764 - 4617.638757)
2 + (4619.638761 - 4618.638764)
3 + (4620.638772 - 4619.638761)
4 + (4621.638731 - 4620.638772)
5 + (4622.638714 - 4621.638731)
6 + (4623.638729 - 4622.638714)
7 + (4624.638727 - 4623.638729)
8 + (4625.638735 - 4624.638727)
9 + (4626.638725 - 4625.638735)) / 9
```

Q.3 — Repeat the exercises of question 2 with the new packet rate (2 pkt/s).

Answer

The screenshot shows the 'Edit Stream [Echo Request]' dialog box with the following settings:

- Send:** ☒ Packets, ☐ Bursts
- Mode:** ☒ Fixed, ☐ Continuous
- Numbers:** Number of Packets: 10, Number of Bursts: 1, Packets per Burst: 10
- Rate:** ☒ Packets/Sec (2.0000), ☐ Bursts/Sec (1.0000), ☐ Bits/Sec (1,344)
- After this stream:** ☐ Stop, ☒ Goto Next Stream, ☐ Goto First
- Gaps (in seconds):** ISG: 0.0, IBG: 0.0, IPG: 0.500000000

Figure 2: The stream settings in Ostinato for Question 3

Table 2: Wireshark ICMP capture for Question 3.

No.	Time	Source	Destination	Protocol	Length	Info
16	140.926179	192.168.0.10	192.168.0.1	ICMP	60	Echo (ping) request id=0x04d2, seq=0/0, ttl=127 (reply in 17)
17	140.926306	192.168.0.1	192.168.0.10	ICMP	60	Echo (ping) reply id=0x04d2, seq=0/0, ttl=64 (request in 16)
18	141.426171	192.168.0.10	192.168.0.1	ICMP	60	Echo (ping) request id=0x04d2, seq=0/0, ttl=127 (reply in 19)
19	141.426320	192.168.0.1	192.168.0.10	ICMP	60	Echo (ping) reply id=0x04d2, seq=0/0, ttl=64 (request in 18)
20	141.926165	192.168.0.10	192.168.0.1	ICMP	60	Echo (ping) request id=0x04d2, seq=0/0, ttl=127 (reply in 21)
21	141.926287	192.168.0.1	192.168.0.10	ICMP	60	Echo (ping) reply id=0x04d2, seq=0/0, ttl=64 (request in 20)
continued on next page						

Table 2 – continued from previous page

No.	Time	Source	Destination	Protocol	Length	Info
22	142.426145	192.168.0.10	192.168.0.1	ICMP	60	Echo (ping) request id=0x04d2, seq=0/0, ttl=127 (reply in 23)
23	142.426254	192.168.0.1	192.168.0.10	ICMP	60	Echo (ping) reply id=0x04d2, seq=0/0, ttl=64 (request in 22)
24	142.926106	192.168.0.10	192.168.0.1	ICMP	60	Echo (ping) request id=0x04d2, seq=0/0, ttl=127 (reply in 25)
25	142.926275	192.168.0.1	192.168.0.10	ICMP	60	Echo (ping) reply id=0x04d2, seq=0/0, ttl=64 (request in 24)
26	143.426150	192.168.0.10	192.168.0.1	ICMP	60	Echo (ping) request id=0x04d2, seq=0/0, ttl=127 (reply in 27)
27	143.426249	192.168.0.1	192.168.0.10	ICMP	60	Echo (ping) reply id=0x04d2, seq=0/0, ttl=64 (request in 26)
28	143.926104	192.168.0.10	192.168.0.1	ICMP	60	Echo (ping) request id=0x04d2, seq=0/0, ttl=127 (reply in 29)
29	143.926259	192.168.0.1	192.168.0.10	ICMP	60	Echo (ping) reply id=0x04d2, seq=0/0, ttl=64 (request in 28)
30	144.426138	192.168.0.10	192.168.0.1	ICMP	60	Echo (ping) request id=0x04d2, seq=0/0, ttl=127 (reply in 31)
31	144.426250	192.168.0.1	192.168.0.10	ICMP	60	Echo (ping) reply id=0x04d2, seq=0/0, ttl=64 (request in 30)
32	144.926075	192.168.0.10	192.168.0.1	ICMP	60	Echo (ping) request id=0x04d2, seq=0/0, ttl=127 (reply in 33)
33	144.926259	192.168.0.1	192.168.0.10	ICMP	60	Echo (ping) reply id=0x04d2, seq=0/0, ttl=64 (request in 32)
34	145.426128	192.168.0.10	192.168.0.1	ICMP	60	Echo (ping) request id=0x04d2, seq=0/0, ttl=127 (reply in 35)
35	145.426236	192.168.0.1	192.168.0.10	ICMP	60	Echo (ping) reply id=0x04d2, seq=0/0, ttl=64 (request in 34)

Again the number of packets in total is 20. 10 requests and 10 replies as specified in the stream's settings, see Figure 2.

And from Table 2 we can get the respective No. of the request and reply packets

The request packets are precisely numbers: 16, 18, 20, 22, 24, 26, 28, 30, 32, 34.

And the reply packets are precisely numbers: 17, 19, 21, 23, 25, 27, 29, 31, 33, 35.

Finally, the below computed value is the average inter-packet delay (in seconds).

$$\begin{aligned} & \frac{(141.426171 - 140.926179) + (141.926165 - 141.426171) + (142.426145 - 141.926165) \\ & + (142.926106 - 142.426145) + (143.426150 - 142.926106) + (143.926104 - 143.426150) \\ & + (144.426138 - 143.926104) + (144.926075 - 144.426138) + (145.426128 - 144.926075)}{10 - 1} \\ & = 0.499994333333335 \text{ seconds} \\ & \approx 0.5 \text{ seconds} \end{aligned}$$

The approximate value, 0.5, is precisely what should be expected since if two request are being sent per second, see Figure 2, that would equate to a packet every half a second which is precisely 0.5 seconds.

Listing 2: Python expression for calculating the inter-packet delay for Question 3.

```

1 ((141.426171 - 140.926179)
2 +(141.926165 - 141.426171)
3 +(142.426145 - 141.926165)
4 +(142.926106 - 142.426145)
5 +(143.426150 - 142.926106)
6 +(143.926104 - 143.426150)
7 +(144.426138 - 143.926104)
8 +(144.926075 - 144.426138)
9 +(145.426128 - 144.926075))/9

```

Q.4.1 — Expand the frame section and inspect it to find out the number of bytes captured by Wireshark from the link (“bytes on wire”).

Answer

→	1 0.000000	192.168.0.10	192.168.0.1	ICMP	60 Echo (ping) request	id=0x04d2, seq=0/0, ttl=127 (reply in 2)
→	2 0.000299	192.168.0.1	192.168.0.10	ICMP	60 Echo (ping) reply	id=0x04d2, seq=0/0, ttl=64 (request in 1)

Figure 3: The packets under inspection for Question 4.1.

The info related to the request from the generator to the DUT is provided below. Specifically, the information related to the Ethernet II frame.

```

▼ Frame 1: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface -, id 0
  Section number: 1
  ▶ Interface id: 0 (-)
    Encapsulation type: Ethernet (1)
    Arrival Time: Dec  3, 2023 11:51:36.046898000 CET
    UTC Arrival Time: Dec  3, 2023 10:51:36.046898000 UTC
    Epoch Arrival Time: 1701600696.046898000
    [Time shift for this packet: 0.000000000 seconds]
    [Time delta from previous captured frame: 0.000000000 seconds]
    [Time delta from previous displayed frame: 0.000000000 seconds]
    [Time since reference or first frame: 0.000000000 seconds]
    Frame Number: 1
    Frame Length: 60 bytes (480 bits)
    Capture Length: 60 bytes (480 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:ethertype:ip:icmp:data]
    [Coloring Rule Name: ICMP]
    [Coloring Rule String: icmp || icmpv6]
  ▶ Ethernet II, Src: 90:00:01:a9:41:f1 (90:00:01:a9:41:f1), Dst: be:98:b5:5f:fb:a8 (be:98:b5:5f:fb:a8)
  ▶ Internet Protocol Version 4, Src: 192.168.0.10, Dst: 192.168.0.1
  ▶ Internet Control Message Protocol

```

Figure 4: The frame information of the request packet under inspection for Question 4.1.

The number of bytes capture by Wireshark is exactly 60 bytes.

Q.4.2 — Search for the minimum length of an Ethernet packet, and state it.

Answer

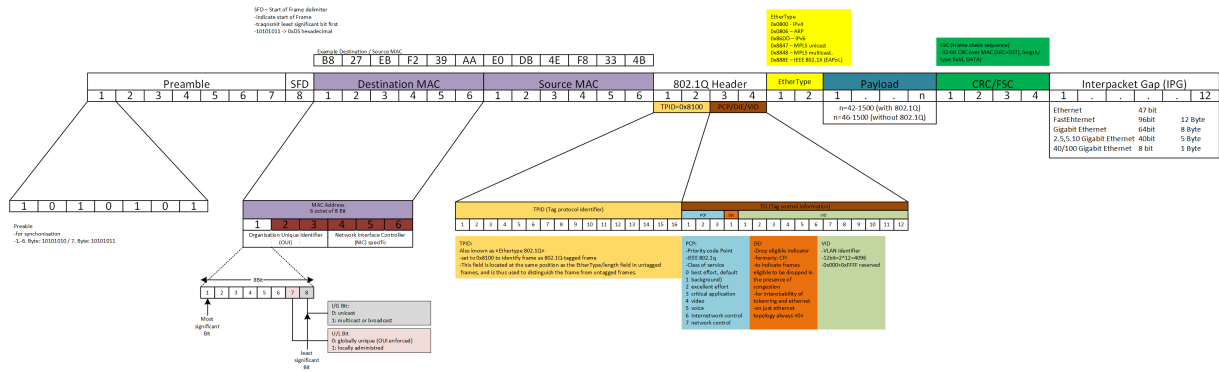


Figure 5: A diagram of the components of an Ethernet II frame.

Reference: https://upload.wikimedia.org/wikipedia/commons/7/72/Ethernet_Frame.png

Note: There is a mistake in the graphic. “CRC/FSC” should be “CRC/FCS” and “FSC (Frame check sequence)” should be “FCS (Frame check sequence)”.

The minimum and maximum lengths can be derived from Figure 5. Importantly, the preamble, SFD and inter-pack gap are almost never exposed beyond layer 1. Therefore, we shall not factor these into our calculation.

Hence, the Ethernet II frame at layer 2 consists of the following segments:

- Destination MAC (6 octets);
- Source MAC (6 octets);
- 802.1Q Header (optional & 4 octets);
- EtherType (2 octets);
- Payload (42 — 1500 octets with 802.1Q & 46 — 1500 octets without 802.1Q); and
- CRC/FCS (4 octets).

Following this, the minimum length can be calculated follows:

$$\text{Frame}_{\min} = 6 + 6 + 2 + 46 + 4 = 64 \text{ octets}$$

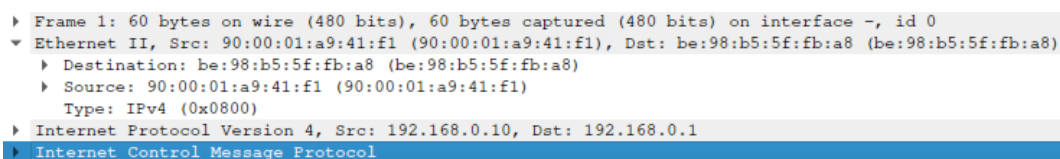
Note: In the case when there is no 802.1Q header, the total header and payload lengths add up to 46. This is because the header length is 0 and the payload has a minimum length of 46. In the other case *i.e.* when there is a 802.1Q header, the total header and payload lengths also add up to 46. This is because the header length is 4 and the payload length has a new minimum of 42, since 4 have already been used by 802.1Q header. Hence, both cases the sum of their lengths is identical.

Similarly, the maximum length is given by the following calculation:

$$\text{Frame}_{\max} = 6 + 6 + 2 + 1500 + 4 = 1518 \text{ octets}$$

Q.4.3 — How does the number of bytes captured (which you found in (4.1)) compare with the minimum length of an Ethernet packet (which you looked up in (4.2))? Can you explain the difference?

Answer



```
▶ Frame 1: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface -, id 0
▼ Ethernet II, Src: 90:00:01:a9:41:f1 (90:00:01:a9:41:f1), Dst: be:98:b5:5f:fb:a8 (be:98:b5:5f:fb:a8)
  ▶ Destination: be:98:b5:5f:fb:a8 (be:98:b5:5f:fb:a8)
  ▶ Source: 90:00:01:a9:41:f1 (90:00:01:a9:41:f1)
  Type: IPv4 (0x0800)
▶ Internet Protocol Version 4, Src: 192.168.0.10, Dst: 192.168.0.1
▶ Internet Control Message Protocol
```

Figure 6: The Ethernet II header of the request packet under inspection for Question 4.1.

The number of “bytes on wire” according to Wireshark, see Figure 6, is precisely 60 bytes. However, the minimum number of bytes is at least 64. This of course, is a 4 byte discrepancy.

To account for this discrepancy, notice that the Ethernet II header does not contain an 802.1Q header. Hence, the payload is allowed a minimum of 46 bytes. Additionally, all these 46 bytes are used up by the ICMP request (including all IP overhead).

Hence, the Ethernet header and ICMP request sum up to 60 bytes. This leaves only one option as to what the remaining 4 bytes can they constitute the Cyclic Redundancy Check (CRC) or Frame Check Sequence (FCS). In fact, according to Figure 5, CRC field is exactly 4 bytes.

Furthermore, this conclusion is further supported by the below referenced Wireshark forum discussion. One of the users exclaims that “bytes on wire” is often more like “bytes on wire without CRC”. This is the case because most network drivers do not provide the CRC to user space applications, instead invalid packets are just immediately dropped.

Reference: <https://osqa-ask.wireshark.org/questions/1344/does-frame-length-include-also-crc-bytes>

Q.4.4 — Expand the Ethernet II section and write down the source MAC address and the destination MAC address.

Answer

Source MAC = 90:00:01:a9:41:f1

Destination MAC = be:98:b5:5f:fb:a8

The above were taken from Figure 6.

Q.4.5 — Look up the source MAC address in the device group configured on the packet generator at the port group.

Answer

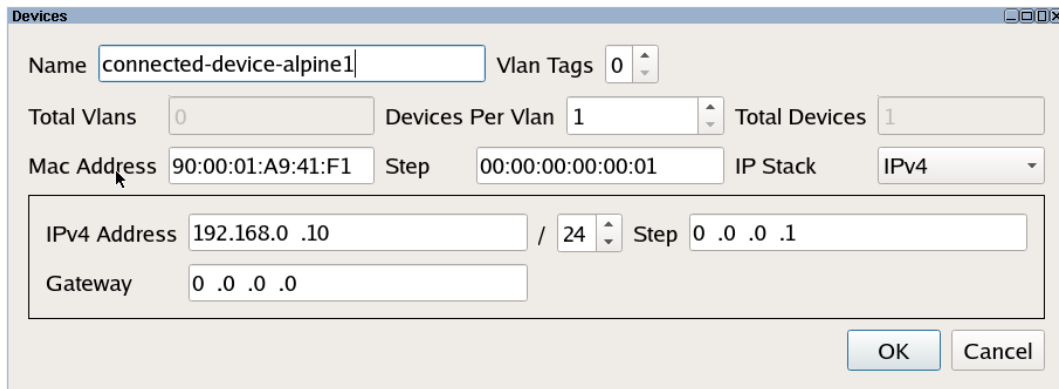


Figure 7: The device group configuration in Ostinato for Question 4.5.

Note: Since the device group contains as single device the base MAC address *i.e.* 90:00:01:a9:41:f1 is used for that device.

As can be seen in Figure 7, the device is given the MAC address 90:00:01:a9:41:f1.

Q.4.6 — Compare what was found in 4.4 with what was found in 4.5.

Answer

Note: Assuming that in the original question, 5.4 and 5.5 where meant to be 4.4 and 4.5 respectively.

As expected, the MAC address of the virtual device is identical to the source MAC Address in the packet. This is because the packet is a request packet *i.e.* it was created by Ostinato.

It is also critical to notice that the request is very much dependent on ARP (Address Resolution Protocol) to establish who has which MAC addresses.

Q.4.7 — Inspect the Ethernet II section and find the field in that section which the receiver (the DUT) uses to identify the layer 3 entity which is to receive the Ethernet frame's payload.

Answer

In Question 4.2 the EtherType /Type field is referenced. I also included a picture of the Ethernet Header of the Request packet.

The Type field is set to 0x0800 which refers to IPv4. Hence, the receiver will understand that the Payload is an IPv4 packet and it will be able to decode it

Q.4.8 — Inspect the section below the Ethernet II section and:

- write down the source address and the destination address that you see in this underlying section;
 - compare the source address and the destination address with what you have set up in the stream named “ICMP Echo Request Stream”.
 - Look up the field named “Total length” in this section and account for the difference between this number and the number you have found in 4.1.
-

Q.5 — For each packet, state source and destination IPv4 address. Compare these latter two addresses with the IPv4 address bound to alpine-1 eth0, and justify the outcome of your comparison.

Q.6 — Compare the packet(s) you capture on Switch1 ↔ alpine-2. State at least one difference between your output and that shown in Fig. 37, and explain it.

Q.7.1 — Present a screenshot that shows the packets you have captured on Switch1 ↔ alpine-4.

Q.7.2 — For each packet, explain how it fits into the sequence that is generated as a result of running the packet stream you have configured on Ostinato.

Q.7.3 — Select the ICMP packet generated by Ostinato. Use a tabular structure, exemplified by Fig. 39, to name **all** the fields, for all the layers, in the ICMP packet. For each field name, write a prefix to indicate which layer the field pertains to, e.g., L1 for layer 1, L2 for layer 2, etc.

Q.8 — Explain why the packets captured on Switch1 ↔ alpine-2 do not change between the point in time just before running the packet stream on Ostinato, and the point in time just after running it.

Q.9 — This concerns the dynamics of transmission using TCP. For each of the four cases (lossless, and packet loss at the rate of 1%, 3% and 5% respectively):

1. Plot a 1-s MA of the throughput, and
 2. calculate the average throughput
-

Q.10 — Consider the packet sequence pertaining to the lossless case.

1. List the flags (within square brackets) pertaining to the first three packets.
2. What part of connection establishment do the first three packets pertain to?
3. List the sequence (Seq=) and acknowledgement (Ack=) numbers pertaining to the first three packets.
4. Identify the maximum segment size which the two parties in the connection state.
5. Identify the range of packets involved in the data transfer phase.
6. Identify the packet(s) involved in the connection teardown phase.

Show screenshots that allow a reader to validate your answers.

Q.11 — Consider the packet sequence pertaining to the 5% packet loss case. List `received_file` on alpine-4 and compare its size with that of `large_file`.

Show screenshots that allow a reader to validate your answers.

Q.12 — Consider the packet sequence pertaining to the lossless case.

1. List `received_file` on alpine-4 and compare its size with that of `large_file`.
2. Go to File->Export Packet Dissections, export the captured packets as CSV and inspect the CSV file in Excel. How does the number of octets sent (as you determine from the CSV file) compare with the size of `received_file`?
3. Inspect the first few packets in the UDP window and identify the length of the UDP datagram (“Len”). How does this compare with the Ethernet frame’s MTU of 1500? Explain any difference you observe.

Show screenshots that allow a reader to validate your answers.

Q.13 — Explain the difference between your observations in 11 and 12.1.
