Academic Year 2023/2024

Internet Performance and Troubleshooting Lab

Politecnico di Torino

Report I

IP Fragmentation

Group 3

Brendon Mendicino (s317639)

Alessandro Ciullo (s310023)

Davide Colaiacomo (s313372)

1 Network Configuration

Host name	IP address
R1/H1	10.0.3.1/25
H2	10.0.3.2/25
R1/H3	10.1.3.1/25
H4	10.1.3.2/25

Table 1.1: Net hosts

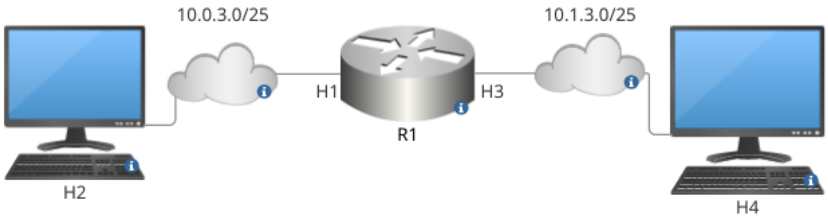


Figure 1.1: Network Topology

2 IP Fragmentation Overview

2.1 Fragmentation header fields

-Which information is used to manage the fragmentation?

Identification : The `Identification` field is used, together with *Source Address*, *Destination Address* and *Protocol* field, to uniquely identify a datagram and its fragments.

Fragment Offset and Total Length : The `Fragment Offset` field tells the receiver the position of a fragment in the original datagram (measured in units of 8 bytes), while the `Total Length` is used to derive the length of the data contained in the fragment (total length - IP header length).

Flags : The `Flags` field is composed of three bits: a `reserved` one, the `DF` (Don't Fragment), set to prohibit fragmentation on the path, and the `MF` (More Fragment), set if the fragment is not the last one.

2.2 The Datagram reassembly problem

When the receiver gets a packet with the `Fragment Offset` set to 0 and the `More Fragment` flag set to 0, it means that a whole datagram has been received and it is sent to the next processing step, otherwise reassembly resources are allocated. These resources consist of a **data buffer**, a **header buffer**, a **fragment block bit table**, and a **timer**. The `buffer identifier` is computed as the concatenation of the *Source Address*, *Destination Address*, *Protocol* field, and *Identification* field. Only after instantiating these data structures the reassembly process can proceed.

-How can the receiver reassemble the packet from the received fragments?

The receiver can reassemble the datagram by concatenating the fragments inside the data buffer until the last packet, which is the one with the `More Fragment` flag set to 0, is received.

-How can it eventually reorder them?

The payload received is put in the data buffer according to its fragment offset and data length, and the bits in the `fragment block bit table` corresponding to the fragment blocks received are set. In this way, the receiver has fragments always ordered and tracked.

-How can it detect that all fragments have been received?

When the last fragment, which is the one with `More Fragment` flag set to 0, is received, the receiver can compute the total data length of the datagram; by using this information and the bits set in the `fragment block bit table`, the host can ensure that the entire datagram has been received and no fragments have been lost.

3 Fragments Order

The fragments are sent in sequential order, which means that when the payload of the IP packet is split into many pieces the bytes are sent in order; we can notice that by analyzing the payload of the packet via Wireshark. If our host will start fragmenting our packets because our MTU is too large we will see in the first packet the header of the ICMP protocol (i.e. if we use `ping`), and the remaining data in the following packets. This is further confirmed by the `fragment_offset` field, that will contain monotonic increasing numbers with each packet of the sequence. Of course, at the receiver, the ordered reception of fragments is not guaranteed.

4 Fragment Reassembling Problem

-Describe the possible algorithm the receiver could implement for this.

An easy implementation of the reassemble algorithm might be to store all the received IP packets in an array of bytes (dynamically re-sizable), where the position is given by the `fragment_offset` and the size of the contiguous cells to fill is given by `length`, an array is created for each tuple `(id, src, dest, prot)`, and until the receiver receives the last packet with the flag `MF` set to 0 the array will be kept in memory; furthermore, it needs to be checked that the whole array is full because some packets might have been lost or still on their way to arrive.

-How could an attacker flood the memory of the destination host by abusing the IP fragmentation?

By looking at this algorithm, it's easy to see a flaw in it: an attacker could exploit this caching mechanism by preventing the last packet to be sent (he plays the role of the `Man In The Middle`) by the sender, so the receiver will never send the array to the upper protocol, thus causing the memory to start filling. If this is done with many connections, the attacker could potentially be able to completely fill the RAM of the receiver.

Another possible attack could be to create many connections with a host and, for each connection, send only one packet with the `MF` flag set to 1 without never sending any more packets; this attack aims at filling the host's RAM as well.

5 Laboratory experiment and analysis

In this section we implement a few experiments to analyse the behavior of the ping exchanges when different MTU sizes, Network topologies and ping hints are used.

5.1 Experiments between two host: Default configuration

The followings tests, conducted between two hosts, aim to analyze the network's behavior in a default configuration when different ping `hints` are used .

5.1.1 Test 1 : H1 MTU 1500 - H2 MTU 1500

```
H2$ ping H1 -s 2000 -c 1 -M want
```

In this first test, **H2** sends an ICMP request of size equal to 2000 bytes and option `-M want` activated; it is to be noted that **H2** fragments the packet before sending it through the network due to its own MTU; then **H1** receives the fragmented packet and sends an ICMP reply consequentially.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.3.2	10.0.3.1	ICMP	1514	Echo (ping) request id=0x002a, seq=1/256, ttl=64 (reply in 3)
2	0.000000196	10.0.3.2	10.0.3.1	IPv4	562	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=333f)
3	0.000076325	10.0.3.1	10.0.3.2	ICMP	1514	Echo (ping) reply id=0x002a, seq=1/256, ttl=64 (request in 1)
4	0.000113679	10.0.3.1	10.0.3.2	IPv4	562	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=8965)

Figure 5.1: Wireshark Test1

If the size of the IP packet sent is less than the MTU, adopting this option would set the `DF` bit to 1.

5.1.2 Test 2 : H1 MTU 1500 - H2 MTU 1500

```
H2$ ping H1 -s 2000 -c 1 -M do
```

In this test, the previous situation is recreated, but **H2** activates the `DF` flag through the `-M do` option, specifying that the packet is not to be fragmented. As a consequence, the packet is directly discarded at IP level without being even sent through the network card, since its MTU would require the fragmentation. The error message received in the terminal is:

```
H2$ ping: local error: message too long, mtu=1500
```

If the size of the IP packet sent is less than the MTU, adopting this option would set the `DF` bit to 1 and the request would go through.

5.1.3 Test 3 : H1 MTU 1500 - H2 MTU 1500

```
H2$ ping H1 -s 2000 -c 1 -M dont
```

This test maintains the same MTU configuration as before, but with the `-M dont` option activated. The result obtained is the same as Figure 5.1. If the size of the IP packet sent is less than the MTU, adopting this option would set the `DF` flag to 0.

5.2 Experiments between two host: non-default MTU

The followings tests, conducted between two hosts, aim to analyze the network's behavior in a non-default configuration when different ping hints are used.

5.2.1 Test 4 : H1 MTU 800 - H2 MTU 1500

```
H2$ ping H1 -s 2000 -c 1 -M want
```

In this test, **H2** sends an ICMP packet which is fragmented at first by **H2** itself, according to its own MTU; after receiving and reassembling the entire Echo Request, **H1** answers back, independently fragmenting again according to its own MTU.

Source	Destination	Protocol	Length	Info
10.0.3.2	10.0.3.1	ICMP	1514	Echo (ping) request id=0x002d, seq=1/256, ttl=64 (reply in 3)
10.0.3.2	10.0.3.1	IPv4	562	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=f876)
10.0.3.1	10.0.3.2	ICMP	810	Echo (ping) reply id=0x002d, seq=1/256, ttl=64 (request in 1)
10.0.3.1	10.0.3.2	IPv4	810	Fragmented IP protocol (proto=ICMP 1, off=776, ID=f338)
10.0.3.1	10.0.3.2	IPv4	490	Fragmented IP protocol (proto=ICMP 1, off=1552, ID=f338)

Figure 5.2: Wireshark Test 4

5.2.2 Test 5 : H1 MTU 800 - H2 MTU 1500

```
H2$ ping H1 -s 2000 -c 1 -M do
```

In this test, in a similar way to `Test 2`, the `-M do` option is adopted, so that the `DF` flag is activated and the packet is immediately discarded at IP level.

5.2.3 Test 6 : H1 MTU 800 - H2 MTU 1500

```
H2$ ping H1 -s 2000 -c 1 -M dont
```

Using the `-M dont` flag gives us the same results as Figure 5.2.

5.3 Experiments with a router on the path

The followings tests, conducted between two hosts, aim to analyze the network's behavior in a non-default configuration, when different ping hints are used and a router is present on the path between sender and receiver.

5.3.1 Test 7 : H1 MTU 800 - H2 MTU 1500 - H3 MTU 1200 - H4 MTU 900

```
H2$ ping H4 -s 1300 -c 2 -M want
```

In this test, when the first packet is sent by **H2**, it is received by the router, which will need to forward it to **H4**; when this happens, since the size of the packet is too big to be forwarded on the interface connected to **H4**, it sends back to **H2** an ICMP `Destination Unreachable` packet with the relative MTU specified. **H2** will then fragment its packets accordingly and will start sending them again to **H2**.

After receiving the entire Echo request, **H4** generates his reply and fragments it according to its MTU. The first fragment of the Echo reply is too big for **H1**'s MTU to **H2**, so the router further fragments the fragment as shown in 5.3.

Source	Destination	Protocol	Length	Info
10.0.3.2	10.1.3.2	ICMP	1342	Echo (ping) request id=0x002f, seq=1/256, ttl=64 (no response found!)
10.0.3.1	10.0.3.2	ICMP	590	Destination unreachable (Fragmentation needed)
10.0.3.2	10.1.3.2	ICMP	1210	Echo (ping) request id=0x002f, seq=2/512, ttl=64 (reply in 5)
10.0.3.2	10.1.3.2	IPv4	166	Fragmented IP protocol (proto=ICMP 1, off=1176, ID=2d98)
10.1.3.2	10.0.3.2	ICMP	810	Echo (ping) reply id=0x002f, seq=2/512, ttl=63 (request in 3)
10.1.3.2	10.0.3.2	IPv4	138	Fragmented IP protocol (proto=ICMP 1, off=776, ID=dbf3)
10.1.3.2	10.0.3.2	IPv4	462	Fragmented IP protocol (proto=ICMP 1, off=880, ID=dbf3)

Figure 5.3: Wireshark Test7 - R1/H1

Source	Destination	Protocol	Length	Info
10.0.3.2	10.1.3.2	ICMP	1210	Echo (ping) request id=0x002f, seq=2/512, ttl=63 (reply in 3)
10.0.3.2	10.1.3.2	IPv4	166	Fragmented IP protocol (proto=ICMP 1, off=1176, ID=2d98)
10.1.3.2	10.0.3.2	ICMP	914	Echo (ping) reply id=0x002f, seq=2/512, ttl=64 (request in 1)
10.1.3.2	10.0.3.2	IPv4	462	Fragmented IP protocol (proto=ICMP 1, off=880, ID=dbf3)

Figure 5.4: Wireshark Test7 - R1/H3

5.3.2 Test 8 : H1 MTU 800 - H2 MTU 1500 - H3 MTU 1200 - H4 MTU 900

```
H2$ ping H4 -s 1300 -c 2 -M do
```

When using the `-M do` option, **H2** sends a packet to the router, which will need to fragment and forward it to **H4**; since the `Dont Fragment` flag is set on the packet, the router will send back to the sender a message with `Destination unreachable (Fragmentation Needed)` and due to the `hint` provided to `ping`, the host terminates the communication. The error message received in the terminal is:

```
H2$ from 10.0.3.1 icmp_seq=1 Frag needed and DF set (mtu = 1200)
```

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.3.2	10.1.3.2	ICMP	1342	Echo (ping) request id=0x0032, seq=1/256, ttl=64 (no response found!)
2	0.000041359	10.0.3.1	10.0.3.2	ICMP	590	Destination unreachable (Fragmentation needed)

Figure 5.5: Wireshark Test 8

5.3.3 Test 9 : H1 MTU 800 - H2 MTU 1500 - H3 MTU 1200 - H4 MTU 900

```
H2$ ping H4 -s 1300 -c 2 -M dont
```

In this test **H2** sends a packet to **H4**, the router receives it but cannot forward it to **H4** because the MTU on the corresponding interface is too small; despite this, since the `Dont Fragment` flag is set to 0, the router can perform the fragmentation on its own. When **H4** receives all the fragments, it will reply according to its own MTU, leading to the router fragmenting the received packets again due to the MTU of the interface leading back to **H2**.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.3.2	10.1.3.2	ICMP	1342	Echo (ping) request id=0x0034, seq=1/256, ttl=64 (reply in 2)
2	0.000513438	10.1.3.2	10.0.3.2	ICMP	810	Echo (ping) reply id=0x0034, seq=1/256, ttl=63 (request in 1)
3	0.000522979	10.1.3.2	10.0.3.2	IPv4	138	Fragmented IP protocol (proto=ICMP 1, off=776, ID=5e30)
4	0.000527616	10.1.3.2	10.0.3.2	IPv4	462	Fragmented IP protocol (proto=ICMP 1, off=880, ID=5e30)
5	1.025071700	10.0.3.2	10.1.3.2	ICMP	1342	Echo (ping) request id=0x0034, seq=2/512, ttl=64 (reply in 6)
6	1.025574609	10.1.3.2	10.0.3.2	ICMP	810	Echo (ping) reply id=0x0034, seq=2/512, ttl=63 (request in 5)
7	1.025584705	10.1.3.2	10.0.3.2	IPv4	138	Fragmented IP protocol (proto=ICMP 1, off=776, ID=5ea8)
8	1.025589180	10.1.3.2	10.0.3.2	IPv4	462	Fragmented IP protocol (proto=ICMP 1, off=880, ID=5ea8)

Figure 5.6: Wireshark Test 9 - R1/H1

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.3.2	10.1.3.2	ICMP	1210	Echo (ping) request id=0x0034, seq=1/256, ttl=63 (reply in 3)
2	0.000008543	10.0.3.2	10.1.3.2	IPv4	166	Fragmented IP protocol (proto=ICMP 1, off=1176, ID=f73e)
3	0.000453995	10.1.3.2	10.0.3.2	ICMP	914	Echo (ping) reply id=0x0034, seq=1/256, ttl=64 (request in 1)
4	0.000454557	10.1.3.2	10.0.3.2	IPv4	462	Fragmented IP protocol (proto=ICMP 1, off=880, ID=5e30)
5	1.025067403	10.0.3.2	10.1.3.2	ICMP	1210	Echo (ping) request id=0x0034, seq=2/512, ttl=63 (reply in 7)
6	1.025076040	10.0.3.2	10.1.3.2	IPv4	166	Fragmented IP protocol (proto=ICMP 1, off=1176, ID=f81b)
7	1.025515582	10.1.3.2	10.0.3.2	ICMP	914	Echo (ping) reply id=0x0034, seq=2/512, ttl=64 (request in 5)
8	1.025516136	10.1.3.2	10.0.3.2	IPv4	462	Fragmented IP protocol (proto=ICMP 1, off=880, ID=5ea8)

Figure 5.7: Wireshark Test 9 - R1/H3

5.4 Experiments with Jumbo Frames

In the following tests we analyse the exchange of Jumbo Frames (Ethernet frames with more than 1500 bytes of payload) between two hosts, in three different scenarios.

5.4.1 Test 10 : H1 MTU 1500 - H2 MTU 9000

```
H2$ ping H1 -s 65000 -c 1
```

The configuration adopted is as follow: the transmitter has enabled Jumbo Frames by setting his MTU on the interface to 9000 (9114 is the maximum allowed size on the laboratory computers) while the receiver used the default configuration.

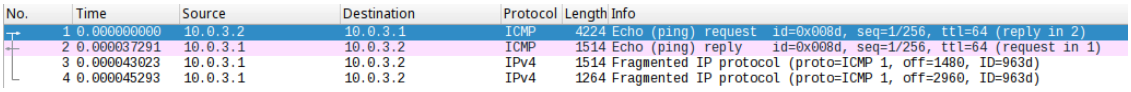
The experiment's result shows that the receiver's Network card drops the jumbo frames sent and gives no answer or errors back to the transmitter.

5.4.2 Test 11 : H1 MTU 1501 - H2 MTU 9000

```
H2$ ping H4 -s 4182 -c 1
```

Let's delve into the issue mentioned earlier in this second experiment. In this experience the MTU of the receiver has been changed from the Default size to 1501 while the transmitter's MTU is still 9000.

The Wireshark's capture. in figure 5.8, shows a different result this time. The receiver doesn't drop the Jumbo Frame even if is bigger than its MTU and by using fragmentation generates his Echo reply. We can deduce that by setting the MTU above 1500 bytes, jumbo frames are enabled.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.3.2	10.0.3.1	ICMP	4224	Echo (ping) request id=0x008d, seq=1/256, ttl=64 (reply in 2)
2	0.000037291	10.0.3.1	10.0.3.2	ICMP	1514	Echo (ping) reply id=0x008d, seq=1/256, ttl=64 (request in 1)
3	0.000043023	10.0.3.1	10.0.3.2	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=963d)
4	0.000045293	10.0.3.1	10.0.3.2	IPv4	1264	Fragmented IP protocol (proto=ICMP 1, off=2960, ID=963d)

Figure 5.8: Wireshark Test 11 - Receiver Side

5.4.3 Test 12 : H1 MTU 1501 - H2 MTU 9000

```
H2$ ping H4 -s 4183 -c 1
```

In this last experiment, we tested the maximum frame size that can be received by enabling jumbo frames at the receiver with an MTU of 1501.

This experience shows that the receiver starts dropping packets beyond a ICMP data length of 4182 bytes (an Ethernet payload of 4210 bytes). By increasing the MTU at the receiver we found a non-linear increment in the maximum Jumbo Frame size that can be accepted by the Network card.

6 Path MTU discovery options

Many of the previous tests have been conducted specifying the strategy that had to be taken in the process of the path MTU discovery, which is the understanding of the maximum size of the packets that a host can send through the network in order to reach a certain destination, while exploiting the network resources and avoiding a possibly sub-optimal throughput. This has been thanks to the `-M` option, that allows 3 possible techniques to be used when attempting to `ping` a host.

6.1 Option 1 : want

```
ping {ip address} -M want
```

In this scenario, a hypothetical sender **H** sets the `Don't Fragment` flag to 1. When **H** forwards in the network a packet that is over the Path MTU, a host with an MTU smaller than the packet size (which might be **H** itself) will discard it and will answer back with an `ICMP Destination Unreachable` packet, specifying the MTU that caused the discard in a specific 16 bit field contained in the ICMP header. As a consequence, **H** reduces its assumed Path MTU according to that field and locally fragments the datagram.

6.2 Option 2 : do

```
ping {ip address} -M do
```

In this scenario, the sender **H** sets the `Don't Fragment` flag to 1 and the fragmentation of packets is not allowed under any circumstances. This means that, when **H** sends a packet, this will reach its destination only if the Path MTU is big enough not to require fragmentation; if that is not the case, a host with local MTU smaller than the packet size will discard it and will answer back with an `ICMP Destination Unreachable` packet, just like in the previous case, but then **H** will not fragment the packets accordingly and they are discarded without further do.

6.3 Option 3 : dont

```
ping {ip address} -M dont
```

In this scenario, the sender **H** sets the `Don't Fragment` flag to 0, so that packet fragmentation is allowed at any node along the path. In practice, when **H** sends a datagram that is bigger than the Path minimum MTU, a router with an MTU smaller than the datagram size will autonomously fragment it and forward it, without letting **H** know directly anything about this.