

CS4150: Computer Networks Lab

Lab9

111901030

Mayank Singla

Q1. You are given a virtual network with three hosts **h1** (192.168.1.2), **r1**, and **r2** (192.168.101.2). **r1** has two interfaces with IPs 192.168.1.1 and 192.168.101.1, and acts as a router between **h1** and **r2**. Let x be the MTU between **r1** and **r2**. Find the value of x and report the approach you used to discover it. *Hint: use the tool `hping3` installed on **h1**. Make sure always to use the `-v` flag with `hping3`.*

The host **h1** is connected to the router **r1** via the interface **eth1**

```
tc@h1:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:5C:20:74
          inet addr:10.0.2.15  Bcast:10.0.2.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:820 errors:0 dropped:0 overruns:0 frame:0
          TX packets:492 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:64750  TX bytes:57214

eth1      Link encap:Ethernet  HWaddr 08:00:27:63:A5:D5
          inet addr:192.168.1.2  Bcast:192.168.1.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:41 errors:0 dropped:0 overruns:0 frame:0
          TX packets:508 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:15670  TX bytes:481888

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0  TX bytes:0

tc@h1:~$
```

The router **r2** is connected to the router **r1** via the interface **eth2**

```
tc@r2:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:24:97:41
          inet addr:10.0.2.15  Bcast:10.0.2.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1726 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1408 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:120728 (117.8 KiB)  TX bytes:204764 (199.9 KiB)

eth1      Link encap:Ethernet  HWaddr 08:00:27:03:03:21
          inet addr:192.168.2.1  Bcast:192.168.2.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

eth2      Link encap:Ethernet  HWaddr 08:00:27:A6:EF:5D
          inet addr:192.168.101.2  Bcast:192.168.101.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1897 errors:0 dropped:0 overruns:0 frame:0
          TX packets:555 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:534836 (522.3 KiB)  TX bytes:38444 (37.5 KiB)

eth3      Link encap:Ethernet  HWaddr 08:00:27:C4:F2:BE
          inet addr:192.168.103.1  Bcast:192.168.103.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:157 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:12174 (11.8 KiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

tc@r2:~$
```

Executing the **tcpdump** command on the router **r2**, we notice a packet coming from the router **r1** which says the **MTU** between **r1** and **r2** is **423**

```
tc@r2:~$ sudo tcpdump -n -i eth2 -v
tcpdump: listening on eth2, link-type EN10MB (Ethernet), capture size 65535 bytes
09:37:59.393925 IP (tos 0xc0, ttl 1, id 14120, offset 0, flags [none], proto OSPF (89), length 68)
  192.168.101.2 > 224.0.0.5: OSPFv2, Hello, length 48
    Router-ID 192.168.103.1, Backbone Area, Authentication Type: none (0)
    Options [External]
      Hello Timer 10s, Dead Timer 40s, Mask 255.255.255.0, Priority 1
      Designated Router 192.168.101.2, Backup Designated Router 192.168.101.1
    Neighbor List:
      192.168.102.1
09:37:59.483350 IP (tos 0xc0, ttl 1, id 14122, offset 0, flags [none], proto OSPF (89), length 52)
  192.168.101.2 > 192.168.101.1: OSPFv2, Database Description, length 32
    Router-ID 192.168.103.1, Backbone Area, Authentication Type: none (0)
    Options [External], DD Flags [Init, More, Master], MTU: 1500, Sequence: 0x6362352e
09:38:02.642774 IP (tos 0xc0, ttl 1, id 14119, offset 0, flags [none], proto OSPF (89), length 52)
  192.168.101.1 > 192.168.101.2: OSPFv2, Database Description, length 32
    Router-ID 192.168.102.1, Backbone Area, Authentication Type: none (0)
    Options [External], DD Flags [Init, More, Master], MTU: 423, Sequence: 0x6362352b
09:38:04.484446 IP (tos 0xc0, ttl 1, id 14123, offset 0, flags [none], proto OSPF (89), length 52)
  192.168.101.2 > 192.168.101.1: OSPFv2, Database Description, length 32
    Router-ID 192.168.103.1, Backbone Area, Authentication Type: none (0)
    Options [External], DD Flags [Init, More, Master], MTU: 1500, Sequence: 0x6362352e
09:38:04.624808 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 192.168.101.1 tell 192.168.101.2, length 28
09:38:04.625690 ARP, Ethernet (len 6), IPv4 (len 4), Reply 192.168.101.1 is-at 08:00:27:d0:7c:cd, length 46
09:38:07.593206 IP (tos 0xc0, ttl 1, id 14121, offset 0, flags [none], proto OSPF (89), length 68)
  192.168.101.1 > 224.0.0.5: OSPFv2, Hello, length 48
    Router-ID 192.168.102.1, Backbone Area, Authentication Type: none (0)
    Options [External]
      Hello Timer 10s, Dead Timer 40s, Mask 255.255.255.0, Priority 1
      Designated Router 192.168.101.2, Backup Designated Router 192.168.101.1
    Neighbor List:
      192.168.103.1
09:38:07.643147 IP (tos 0xc0, ttl 1, id 14122, offset 0, flags [none], proto OSPF (89), length 52)
  192.168.101.1 > 192.168.101.2: OSPFv2, Database Description, length 32
    Router-ID 192.168.102.1, Backbone Area, Authentication Type: none (0)
    Options [External], DD Flags [Init, More, Master], MTU: 423, Sequence: 0x6362352b
^C
8 packets captured
8 packets received by filter
0 packets dropped by kernel
tc@r2:~$
```

Listing all the services using **netstat** to find the port for the machine **r1**. The ports for **h1(14501)** and **r2(14602)** were already listed in **connect.sh** script. We find the port of **r1** as **14601**

```
cs4150@aha-acdgfl-058l:~/Downloads/lab9_network$ sudo netstat -plnt | grep VBox
[sudo] password for cs4150:
cs4150 is not in the sudoers file. This incident will be reported.
cs4150@aha-acdgfl-058l:~/Downloads/lab9_network$ su -
Password:
root@aha-acdgfl-058l:~# netstat -plnt | grep VBox
tcp        0      0 0.0.0.0:14501          0.0.0.0:*              LISTEN     9122/VBoxHeadless
tcp        0      0 0.0.0.0:14601          0.0.0.0:*              LISTEN     9019/VBoxHeadless
tcp        0      0 0.0.0.0:14602          0.0.0.0:*              LISTEN     9070/VBoxHeadless
root@aha-acdgfl-058l:~# logout
cs4150@aha-acdgfl-058l:~/Downloads/lab9_network$
```

(sudo password was filled by the IT staff in the lab)

Trying to connect to **r1** with the found MTU value of **423**

```
cs4150@aha-acdgfl-0581:~/Downloads/lab9_network$ ssh -p 14601 -o StrictHostKeyChecking=no tc@localhost
tc@localhost's password:
Permission denied, please try again.
tc@localhost's password: █
```

It should not fail as it should be the expected MTU between **r1** and **r2**. But it fails which means, we need to try something different.

First, I find the maximum data length that can be sent from **h1** to **r2** without any fragmentation.

Using the **hping3** command from **h1** to send data of length 423 to **r2**

```
tc@h1:~$ sudo hping3 192.168.101.2 -V --data 423
using eth1, addr: 192.168.1.2, MTU: 1500
HPING 192.168.101.2 (eth1 192.168.101.2): NO FLAGS are set, 40 headers + 423 data bytes
█
```

We can see packets from **h1** came in lengths of **420** and **380**, which means the data of length 423 was fragmented.

```
tc@r2:~$ sudo tcpdump -n -i eth2 -v
tcpdump: listening on eth2, link-type EN10MB (Ethernet), capture size 65535 bytes
09:55:57.665831 IP (tos 0xc0, ttl 1, id 14549, offset 0, flags [none], proto OSPF (89), length 68)
  192.168.101.1 > 224.0.0.5: OSPFv2, Hello, length 48
    Router-ID 192.168.102.1, Backbone Area, Authentication Type: none (0)
    Options [External]
      Hello Timer 10s, Dead Timer 40s, Mask 255.255.255.0, Priority 1
      Designated Router 192.168.101.2, Backup Designated Router 192.168.101.1
    Neighbor List:
      192.168.103.1
09:55:57.767598 IP (tos 0x0, ttl 63, id 53812, offset 0, flags [+], proto TCP (6), length 420)
  192.168.1.2.1484 > 192.168.101.2.0: Flags [+], seq 432345859:432346239, win 512, length 380
09:55:57.767629 IP (tos 0x0, ttl 63, id 53812, offset 400, flags [none], proto TCP (6), length 63)
  192.168.1.2 > 192.168.101.2: ip-proto-6
09:55:57.784789 IP (tos 0xc0, ttl 1, id 14550, offset 0, flags [none], proto OSPF (89), length 52)
  192.168.101.1 > 192.168.101.2: OSPFv2, Database Description, length 32
    Router-ID 192.168.102.1, Backbone Area, Authentication Type: none (0)
    Options [External], DD Flags [Init, More, Master], MTU: 423, Sequence: 0x6362352b
09:55:57.913278 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 192.168.101.2 tell 192.168.101.1, length 46
09:55:57.913315 ARP, Ethernet (len 6), IPv4 (len 4), Reply 192.168.101.2 is-at 08:00:27:a6:ef:5d, length 28
09:55:58.768055 IP (tos 0x0, ttl 63, id 20961, offset 0, flags [+], proto TCP (6), length 420)
  192.168.1.2.1485 > 192.168.101.2.0: Flags [+], seq 451574981:451575361, win 512, length 380
09:55:58.768086 IP (tos 0x0, ttl 63, id 20961, offset 400, flags [none], proto TCP (6), length 63)
  192.168.1.2 > 192.168.101.2: ip-proto-6
09:55:59.455181 IP (tos 0xc0, ttl 1, id 14552, offset 0, flags [none], proto OSPF (89), length 68)
  192.168.101.2 > 224.0.0.5: OSPFv2, Hello, length 48
    Router-ID 192.168.103.1, Backbone Area, Authentication Type: none (0)
    Options [External]
      Hello Timer 10s, Dead Timer 40s, Mask 255.255.255.0, Priority 1
      Designated Router 192.168.101.2, Backup Designated Router 192.168.101.1
    Neighbor List:
      192.168.102.1
09:55:59.611182 IP (tos 0xc0, ttl 1, id 14554, offset 0, flags [none], proto OSPF (89), length 52)
  192.168.101.2 > 192.168.101.1: OSPFv2, Database Description, length 32
    Router-ID 192.168.103.1, Backbone Area, Authentication Type: none (0)
    Options [External], DD Flags [Init, More, Master], MTU: 1500, Sequence: 0x6362352e
09:55:59.768120 IP (tos 0x0, ttl 63, id 64387, offset 0, flags [+], proto TCP (6), length 420)
  192.168.1.2.1486 > 192.168.101.2.0: Flags [+], seq 206070908:206071288, win 512, length 380
09:55:59.768150 IP (tos 0x0, ttl 63, id 64387, offset 400, flags [none], proto TCP (6), length 63)
  192.168.1.2 > 192.168.101.2: ip-proto-6
^C
12 packets captured
12 packets received by filter
0 packets dropped by kernel
tc@r2:~$ █
```


Trying to send the data of length **420** from **h1** to **r2** with **DontFrag** bit enabled so as not to fragment the data

```
tc@h1:~$ sudo hping3 192.168.101.2 -V --data 420 --dontfrag
using eth1, addr: 192.168.1.2, MTU: 1500
HPING 192.168.101.2 (eth1 192.168.101.2): NO FLAGS are set, 40 headers + 420 data bytes
ICMP Fragmentation Needed/DF set from ip=192.168.1.1 get hostname...
```

We received an error that IP fragmentation is needed, hence we can't find data of length **420** from **h1** without fragmentation.

Trying to send the data of length **380** from **h1** to **r2** with **DontFrag** bit enabled so as not to fragment the data

```
tc@h1:~$ sudo hping3 192.168.101.2 -V --data 380 --dontfrag
using eth1, addr: 192.168.1.2, MTU: 1500
HPING 192.168.101.2 (eth1 192.168.101.2): NO FLAGS are set, 40 headers + 380 data bytes
```

We were successfully able to send the data of length **380**.

To find the maximum data length that can be sent without fragmentation, I did a binary search on the data length between the data length values **380** and **420**.

```
tc@h1:~$ sudo hping3 192.168.101.2 -V --data 400 --dontfrag
using eth1, addr: 192.168.1.2, MTU: 1500
HPING 192.168.101.2 (eth1 192.168.101.2): NO FLAGS are set, 40 headers + 400 data bytes
ICMP Fragmentation Needed/DF set from ip=192.168.1.1 get hostname...
```

```
tc@h1:~$ sudo hping3 192.168.101.2 -V --data 390 --dontfrag
using eth1, addr: 192.168.1.2, MTU: 1500
HPING 192.168.101.2 (eth1 192.168.101.2): NO FLAGS are set, 40 headers + 390 data bytes
ICMP Fragmentation Needed/DF set from ip=192.168.1.1 get hostname...
```

```
tc@h1:~$ sudo hping3 192.168.101.2 -V --data 385 --dontfrag
using eth1, addr: 192.168.1.2, MTU: 1500
HPING 192.168.101.2 (eth1 192.168.101.2): NO FLAGS are set, 40 headers + 385 data bytes
ICMP Fragmentation Needed/DF set from ip=192.168.1.1 get hostname...
```

```
tc@h1:~$ sudo hping3 192.168.101.2 -V --data 383 --dontfrag
using eth1, addr: 192.168.1.2, MTU: 1500
HPING 192.168.101.2 (eth1 192.168.101.2): NO FLAGS are set, 40 headers + 383 data bytes
```

```
tc@h1:~$ sudo hping3 192.168.101.2 -V --data 384 --dontfrag
using eth1, addr: 192.168.1.2, MTU: 1500
HPING 192.168.101.2 (eth1 192.168.101.2): NO FLAGS are set, 40 headers + 384 data bytes
ICMP Fragmentation Needed/DF set from ip=192.168.1.1 get hostname...
```

From the result of the binary search, the maximum data length that can be sent from **h1** to **r2** is **383**. As the MTU observed previously was **423**, that means the header length in it was **40 (423 - 383)**

As the MTU value of 423 was not working, I tried enabling the link layer header information to be displayed on **r2** and noticed the total length of the packets including the link layer header.

I tried sending the data of length **383** from **h1** to **r2** with and without fragmentation

```
tc@h1:~$ sudo hping3 192.168.101.2 -V --data 383 --dontfrag
using eth1, addr: 192.168.1.2, MTU: 1500
HPING 192.168.101.2 (eth1 192.168.101.2): NO FLAGS are set, 40 headers + 383 data bytes
```

```
tc@r2:~$ sudo tcpdump -n -i eth2 -v -e
tcpdump: listening on eth2, link-type EN10MB (Ethernet), capture size 65535 bytes
10:10:09.499793 08:00:27:a6:ef:5d > 01:00:5e:00:00:05, ethertype IPv4 (0x0800), length 82: (tos 0xc0, ttl 1, id 14892, offset 0, flags [none], proto OSPF (89), length 68)
192.168.101.2 > 224.0.0.5: OSPFV2, Hello, length 48
Router-ID 192.168.103.1, Backbone Area, Authentication Type: none (0)
Options [External]
Hello Timer 10s, Dead Timer 40s, Mask 255.255.255.0, Priority 1
Designated Router 192.168.101.2, Backup Designated Router 192.168.101.1
Neighbor List:
192.168.102.1
10:10:09.719207 08:00:27:a6:ef:5d > 08:00:27:d0:7c:cd, ethertype IPv4 (0x0800), length 66: (tos 0xc0, ttl 1, id 14894, offset 0, flags [none], proto OSPF (89), length 52)
192.168.101.2 > 192.168.101.1: OSPFV2, Database Description, length 32
Router-ID 192.168.103.1, Backbone Area, Authentication Type: none (0)
Options [External], DD Flags [Init, More, Master], MTU: 1500, Sequence: 0x6362352e
10:10:10.111329 08:00:27:d0:7c:cd > 08:00:27:a6:ef:5d, ethertype IPv4 (0x0800), length 437: (tos 0xc0, ttl 63, id 61350, offset 0, flags [DF], proto TCP (6), length 423)
192.168.1.2.2766 > 192.168.101.2.0: Flags [DF], cksum 0xaf9b (correct), seq 308883087:308883470, win 512, length 383
10:10:11.111854 08:00:27:d0:7c:cd > 08:00:27:a6:ef:5d, ethertype IPv4 (0x0800), length 437: (tos 0xc0, ttl 63, id 63461, offset 0, flags [DF], proto TCP (6), length 423)
192.168.1.2.2767 > 192.168.101.2.0: Flags [DF], cksum 0x50ef (correct), seq 1084958684:1084959067, win 512, length 383
10:10:12.112070 08:00:27:d0:7c:cd > 08:00:27:a6:ef:5d, ethertype IPv4 (0x0800), length 437: (tos 0xc0, ttl 63, id 19713, offset 0, flags [DF], proto TCP (6), length 423)
192.168.1.2.2768 > 192.168.101.2.0: Flags [DF], cksum 0x2447 (correct), seq 2060737822:2060738205, win 512, length 383
10:10:12.889985 08:00:27:d0:7c:cd > 08:00:27:a6:ef:5d, ethertype IPv4 (0x0800), length 66: (tos 0xc0, ttl 1, id 14891, offset 0, flags [none], proto OSPF (89), length 52)
192.168.101.1 > 192.168.101.2: OSPFV2, Database Description, length 32
Router-ID 192.168.102.1, Backbone Area, Authentication Type: none (0)
Options [External], DD Flags [Init, More, Master], MTU: 423, Sequence: 0x6362352b
10:10:13.112207 08:00:27:d0:7c:cd > 08:00:27:a6:ef:5d, ethertype IPv4 (0x0800), length 437: (tos 0xc0, ttl 63, id 24122, offset 0, flags [DF], proto TCP (6), length 423)
192.168.1.2.2769 > 192.168.101.2.0: Flags [DF], cksum 0x5c2d (correct), seq 843025515:843025898, win 512, length 383
10:10:14.112392 08:00:27:d0:7c:cd > 08:00:27:a6:ef:5d, ethertype IPv4 (0x0800), length 437: (tos 0xc0, ttl 63, id 8339, offset 0, flags [DF], proto TCP (6), length 423)
192.168.1.2.2770 > 192.168.101.2.0: Flags [DF], cksum 0xcf81 (correct), seq 1977239825:1977240208, win 512, length 383
^C
8 packets captured
8 packets received by filter
0 packets dropped by kernel
tc@r2:~$
```

```
tc@h1:~$ sudo hping3 192.168.101.2 -V --data 383
using eth1, addr: 192.168.1.2, MTU: 1500
HPING 192.168.101.2 (eth1 192.168.101.2): NO FLAGS are set, 40 headers + 383 data bytes
```

```

tc@r2:~$ sudo tcpdump -n -i eth2 -v -e
tcpdump: listening on eth2, link-type EN10MB (Ethernet), capture size 65535 bytes
10:24:27.791858 08:00:27:d0:7c:cd > 01:00:5e:00:00:05, ethertype IPv4 (0x0800), length 82: (tos 0xc0, ttl 1, id 15233, offset 0, flags [none], proto OSPF (89), length 68)
  192.168.101.1 > 224.0.0.5: OSPFv2, Hello, length 48
    Router-ID 192.168.102.1, Backbone Area, Authentication Type: none (0)
    Options [External]
      Hello Timer 10s, Dead Timer 40s, Mask 255.255.255.0, Priority 1
      Designated Router 192.168.101.2, Backup Designated Router 192.168.101.1
    Neighbor List:
      192.168.103.1
10:24:27.937115 08:00:27:d0:7c:cd > 08:00:27:a6:ef:5d, ethertype IPv4 (0x0800), length 437: (tos 0x0, ttl 63, id 54450, offset 0, flags [none], proto TCP (6), length 423)
  192.168.1.2.2141 > 192.168.101.2.0: Flags [none], cksum 0xa2ea (correct), seq 126543954:126544337, win 512, length 383
10:24:27.983623 08:00:27:d0:7c:cd > 08:00:27:a6:ef:5d, ethertype IPv4 (0x0800), length 66: (tos 0xc0, ttl 1, id 15234, offset 0, flags [none], proto OSPF (89), length 52)
  192.168.101.1 > 192.168.101.2: OSPFv2, Database Description, length 32
    Router-ID 192.168.102.1, Backbone Area, Authentication Type: none (0)
    Options [External], DD Flags [Init, More, Master], MTU: 423, Sequence: 0x6362352b
10:24:27.993112 08:00:27:d0:7c:cd > 08:00:27:a6:ef:5d, ethertype ARP (0x0806), length 60: Ethernet (len 6), IPv4 (len 4), Request who-has 192.168.101.2 tell 192.168.101.1, length 46
10:24:27.993146 08:00:27:a6:ef:5d > 08:00:27:d0:7c:cd, ethertype ARP (0x0806), length 42: Ethernet (len 6), IPv4 (len 4), Reply 192.168.101.2 is-at 08:00:27:a6:ef:5d, length 28
10:24:28.937448 08:00:27:d0:7c:cd > 08:00:27:a6:ef:5d, ethertype IPv4 (0x0800), length 437: (tos 0x0, ttl 63, id 43368, offset 0, flags [none], proto TCP (6), length 423)
  192.168.1.2.2142 > 192.168.101.2.0: Flags [none], cksum 0x9814 (correct), seq 1927203017:1927203400, win 512, length 383
10:24:29.556744 08:00:27:a6:ef:5d > 01:00:5e:00:00:05, ethertype IPv4 (0x0800), length 82: (tos 0xc0, ttl 1, id 15236, offset 0, flags [none], proto OSPF (89), length 68)
  192.168.101.2 > 224.0.0.5: OSPFv2, Hello, length 48
    Router-ID 192.168.103.1, Backbone Area, Authentication Type: none (0)
    Options [External]
      Hello Timer 10s, Dead Timer 40s, Mask 255.255.255.0, Priority 1
      Designated Router 192.168.101.2, Backup Designated Router 192.168.101.1
    Neighbor List:
      192.168.102.1
10:24:29.827068 08:00:27:a6:ef:5d > 08:00:27:d0:7c:cd, ethertype IPv4 (0x0800), length 66: (tos 0xc0, ttl 1, id 15238, offset 0, flags [none], proto OSPF (89), length 52)
  192.168.101.2 > 192.168.101.1: OSPFv2, Database Description, length 32
    Router-ID 192.168.103.1, Backbone Area, Authentication Type: none (0)
    Options [External], DD Flags [Init, More, Master], MTU: 1500, Sequence: 0x6362352e
10:24:29.937934 08:00:27:d0:7c:cd > 08:00:27:a6:ef:5d, ethertype IPv4 (0x0800), length 437: (tos 0x0, ttl 63, id 29972, offset 0, flags [none], proto TCP (6), length 423)
  192.168.1.2.2143 > 192.168.101.2.0: Flags [none], cksum 0xfc06 (correct), seq 1256756360:1256756743, win 512, length 383
^C
9 packets captured
9 packets received by filter
0 packets dropped by kernel
tc@r2:~$

```

In both cases, I can packet of the length **437** arriving on **r2**

I tried this as the MTU value and it still didn't work

```

cs4150@aha-acdglf1-0581:~/Downloads/lab9_network$ ssh -p 14601 -o StrictHostKeyChecking=no tc@localhost
tc@localhost's password:
Permission denied, please try again.
tc@localhost's password:

```

I tried sending a packet with data of a larger length from **h1** to **r2** with fragmentation

```

tc@h1:~$ sudo hping3 192.168.101.2 -V --data 423
using eth1, addr: 192.168.1.2, MTU: 1500
HPING 192.168.101.2 (eth1 192.168.101.2): NO FLAGS are set, 40 headers + 423 data bytes

```

```

tc@r2:~$ sudo tcpdump -n -i eth2 -v -e
tcpdump: listening on eth2, link-type EN10MB (Ethernet), capture size 65535 bytes
10:26:13.920640 08:00:27:d0:7c:cd > 08:00:27:a6:ef:5d, ethertype IPv4 (0x0800), length 434: (tos 0xc0, ttl 63, id 33519, offset 0, flags [+], proto TCP (6), length 420)
  192.168.1.2.1882 > 192.168.101.2.0: Flags [+], seq 1826304316:1826304696, win 512, length 380
10:26:13.920670 08:00:27:d0:7c:cd > 08:00:27:a6:ef:5d, ethertype IPv4 (0x0800), length 77: (tos 0xc0, ttl 63, id 33519, offset 400, flags [none], proto TCP (6), length 63)
  192.168.1.2 > 192.168.101.2: tp-proto-6
10:26:14.839191 08:00:27:a6:ef:5d > 08:00:27:d0:7c:cd, ethertype IPv4 (0x0800), length 66: (tos 0xc0, ttl 1, id 15279, offset 0, flags [none], proto OSPF (89), length 52)
  192.168.101.2 > 192.168.101.1: OSPFv2, Database Description, length 32
    Router-ID 192.168.103.1, Backbone Area, Authentication Type: none (0)
    Options [External], DD Flags [Init, More, Master], MTU: 1500, Sequence: 0x6362352e
10:26:14.864523 08:00:27:a6:ef:5d > 08:00:27:d0:7c:cd, ethertype ARP (0x0806), length 42: Ethernet (len 6), IPv4 (len 4), Request who-has 192.168.101.1 tell 192.168.101.2, length 28
10:26:14.865329 08:00:27:d0:7c:cd > 08:00:27:a6:ef:5d, ethertype ARP (0x0806), length 60: Ethernet (len 6), IPv4 (len 4), Reply 192.168.101.1 is-at 08:00:27:d0:7c:cd, length 46
10:26:14.921241 08:00:27:d0:7c:cd > 08:00:27:a6:ef:5d, ethertype IPv4 (0x0800), length 434: (tos 0xc0, ttl 63, id 35865, offset 0, flags [+], proto TCP (6), length 420)
  192.168.1.2.1883 > 192.168.101.2.0: Flags [+], seq 882354496:882354876, win 512, length 380
10:26:14.921271 08:00:27:d0:7c:cd > 08:00:27:a6:ef:5d, ethertype IPv4 (0x0800), length 77: (tos 0xc0, ttl 63, id 35865, offset 400, flags [none], proto TCP (6), length 63)
  192.168.1.2 > 192.168.101.2: tp-proto-6
10:26:15.921497 08:00:27:d0:7c:cd > 08:00:27:a6:ef:5d, ethertype IPv4 (0x0800), length 434: (tos 0xc0, ttl 63, id 64497, offset 0, flags [+], proto TCP (6), length 420)
  192.168.1.2.1884 > 192.168.101.2.0: Flags [+], seq 784445250:784445630, win 512, length 380
10:26:15.921528 08:00:27:d0:7c:cd > 08:00:27:a6:ef:5d, ethertype IPv4 (0x0800), length 77: (tos 0xc0, ttl 63, id 64497, offset 400, flags [none], proto TCP (6), length 63)
  192.168.1.2 > 192.168.101.2: tp-proto-6
10:26:16.921864 08:00:27:d0:7c:cd > 08:00:27:a6:ef:5d, ethertype IPv4 (0x0800), length 434: (tos 0xc0, ttl 63, id 34341, offset 0, flags [+], proto TCP (6), length 420)
  192.168.1.2.1885 > 192.168.101.2.0: Flags [+], seq 50891990:50892370, win 512, length 380
10:26:16.921896 08:00:27:d0:7c:cd > 08:00:27:a6:ef:5d, ethertype IPv4 (0x0800), length 77: (tos 0xc0, ttl 63, id 34341, offset 400, flags [none], proto TCP (6), length 63)
  192.168.1.2 > 192.168.101.2: tp-proto-6
^C
11 packets captured
11 packets received by filter
0 packets dropped by kernel
tc@r2:~$

```

This time the total packet length including link layer header was found to be **434**

```

cs4150@aha-acdgfl-058l:~/Downloads/lab9_network$ ssh -p 14601 -o StrictHostKeyChecking=no tc@localhost
tc@localhost's password:
( '>')
/) TC (\   Core is distributed with ABSOLUTELY NO WARRANTY.
(/-__-_)   www.tinycorelinux.net

tc@r1:~$ ls
ipfrags.tar.xz
tc@r1:~$ █

```

And it worked :)

Conclusion: The MTU value printed by the **tcpdump** command was **423** which should be the actual correct value of the MTU between **r1** and **r2**, but this didn't work as the password and the value received by sending packets with fragmentation and also including the link layer header in it, which was **434** works. Hence, the value of **x** is **434**

```

cs4150@aha-acdgfl-058l:~/Downloads/lab9_network$ ls
connect.sh  setupVMs.sh  startVMs.sh  stopVMs.sh  VirtualBox  VMImages
cs4150@aha-acdgfl-058l:~/Downloads/lab9_network$ scp -P 14601 tc@localhost:/home/tc/ipfrags.tar.xz ./
tc@localhost's password:
ipfrags.tar.xz                                     100%  13KB   6.2MB/s   00:00
cs4150@aha-acdgfl-058l:~/Downloads/lab9_network$ ls
connect.sh  ipfrags.tar.xz  setupVMs.sh  startVMs.sh  stopVMs.sh  VirtualBox  VMImages
cs4150@aha-acdgfl-058l:~/Downloads/lab9_network$ █

```

Copying the **ipfrags.tar.xz** file from **r1** to the local machine using the **scp** command and the port found earlier.

Q2. Given the value of **x** from the above questions, the password for **r1** is **user@x**. Using this password, download the file **ipfrags.tar.xz** from **r1** and extract it on your host machine. The extracted folder contains 540 IPv4 fragments, of which only 54 are legitimate. Use the IPv4 header checksum to weed out fragments with errors. Print the header details for all the legitimate packets.

Use the following structure to parse IPv4 packets that are stored as files.

```
typedef struct IPPacket_t {
    unsigned char v_hl;
    unsigned char dscp_ecn;
    unsigned short int totalLen;
    unsigned short int id;
    unsigned short int flags_frag_offset;
    unsigned char ttl;
    unsigned char proto;
    unsigned short int checksum;
    unsigned char sAddr[4];
    unsigned char dAddr[4];
    unsigned int o1;
    unsigned int o2;
    unsigned char data[1024];
} IPPacket;
```

I created the given struct and defined **getters** for all the different fields which are present in the IPv4 datagram packet to use them later for printing the packet details.

I created the function **getBinary()** to get the binary representation of a given value of **char** or **int** data type families. I created the function **binaryToInt()** to perform the reverse operation of converting a binary string value to its decimal value. I used the **<bitset>** library provided by C++ to perform these operations.

I created the function **getIP()** that will convert the 32-bit IP address in binary form to its dotted notation form. It will simply take 8-bits at a time and convert it to its decimal.

I created the function **addBinaryBits()** that will add two binary bits and the given input carry based on the following digital logic.

$$\begin{aligned} \text{Sum} &= A \oplus B \oplus C_{in} \\ C_{out} &= AB + C_{in}(A \oplus B) \end{aligned}$$

I created the function **addBinaryNums()** that will add two binary numbers using **1's complement addition**. In this, if after adding the binary numbers, we get a carry at the end, then we add **1** to the obtained sum to get the final resulting sum.

I created the function **computeChecksum()** which will compute the header checksum of a given IP packet. It will take 2 bytes (16 bits) at a time and do the 1's complement addition of those bytes. The final checksum will be 1's complement of the resultant addition.

I created the function **validateChecksum()** which will get the checksum of the IP packet and validate it if it is valid or not. If there is no error in the received IP packet, then the computed checksum should come out to be **0**.

Lastly, I created the function **printPacketDetails()** to print all the fields present in an IP packet.

In the **main** function, I used the **<filesystem>** library provided by **C++17** onwards to read all the files in the **ipfrags** directory. I opened the files in the binary mode using the **ifstream** object and populated the binary data present in it to a pointer to the **IPPacket** struct to automatically populate all the different fields in it. The struct is designed in such a way that it satisfies the IPv4 datagram format. Then, I validate the header checksum in the packet using the **validateChecksum()** function, and if it is valid I printed out its details.

Here are a few screenshots of the legitimate packets received.

(We need to provide **-std=C++2a** flag for C++20 while compiling to use the **<filesystem>** library)

```
codes on master !1 ?1
g++ -std=c++2a q2.cpp -o q2

codes on master !1 ?1
./q2
Packet          : "ip_R5Ei7J"
Version         : 4
Header Length (bytes) : 7
Type of Service : 9
Datagram Length (bytes) : 28
16-bit Identifier : 38906
Flags           : Reserved Bit(0), Don't Fragment Bit(0), More Fragment Bit(1)
13-bit Fragmentation Offset : 4
Time-to-Live    : 4
Upper-Layer Protocol : 8
Header Checksum : 0001010010000000
32-bit Source IP Address : 192.168.10.1
32-bit Destination IP Address : 1.10.168.192
Options         : 32625, 1049595812

Packet          : "ip_56nzse"
Version         : 4
```

```
Packet          : "ip_VcGyWt"
Version         : 4
Header Length (bytes) : 7
Type of Service  : 73
Datagram Length (bytes) : 52
16-bit Identifier : 20026
Flags           : Reserved Bit(0), Don't Fragment Bit(0), More Fragment Bit(1)
13-bit Fragmentation Offset : 2
Time-to-Live    : 4
Upper-Layer Protocol : 8
Header Checksum  : 0111011000110100
32-bit Source IP Address : 192.168.10.1
32-bit Destination IP Address : 1.10.168.192
Options         : 32663, 2791231396
```

```
Packet          : "ip_thaz50"
Version         : 4
Header Length (bytes) : 7
Type of Service  : 73
Datagram Length (bytes) : 47
16-bit Identifier : 20026
Flags           : Reserved Bit(0), Don't Fragment Bit(0), More Fragment Bit(0)
13-bit Fragmentation Offset : 22
Time-to-Live    : 4
Upper-Layer Protocol : 8
Header Checksum  : 1111011000100101
32-bit Source IP Address : 192.168.10.1
32-bit Destination IP Address : 1.10.168.192
Options         : 32663, 2791231396
```

```
32-bit Destination IP Address : 1.10.168.192
Options                       : 32663, 2791231396
```

```
Packet          : "ip_5RpOfy"
Version         : 4
Header Length (bytes) : 7
Type of Service  : 9
Datagram Length (bytes) : 44
16-bit Identifier : 38906
Flags           : Reserved Bit(0), Don't Fragment Bit(0), More Fragment Bit(1)
13-bit Fragmentation Offset : 12
Time-to-Live    : 4
Upper-Layer Protocol : 8
Header Checksum  : 0001010001101000
32-bit Source IP Address : 192.168.10.1
32-bit Destination IP Address : 1.10.168.192
Options         : 32625, 1049595812
```

```
Number of legitimate packets found: 54
```

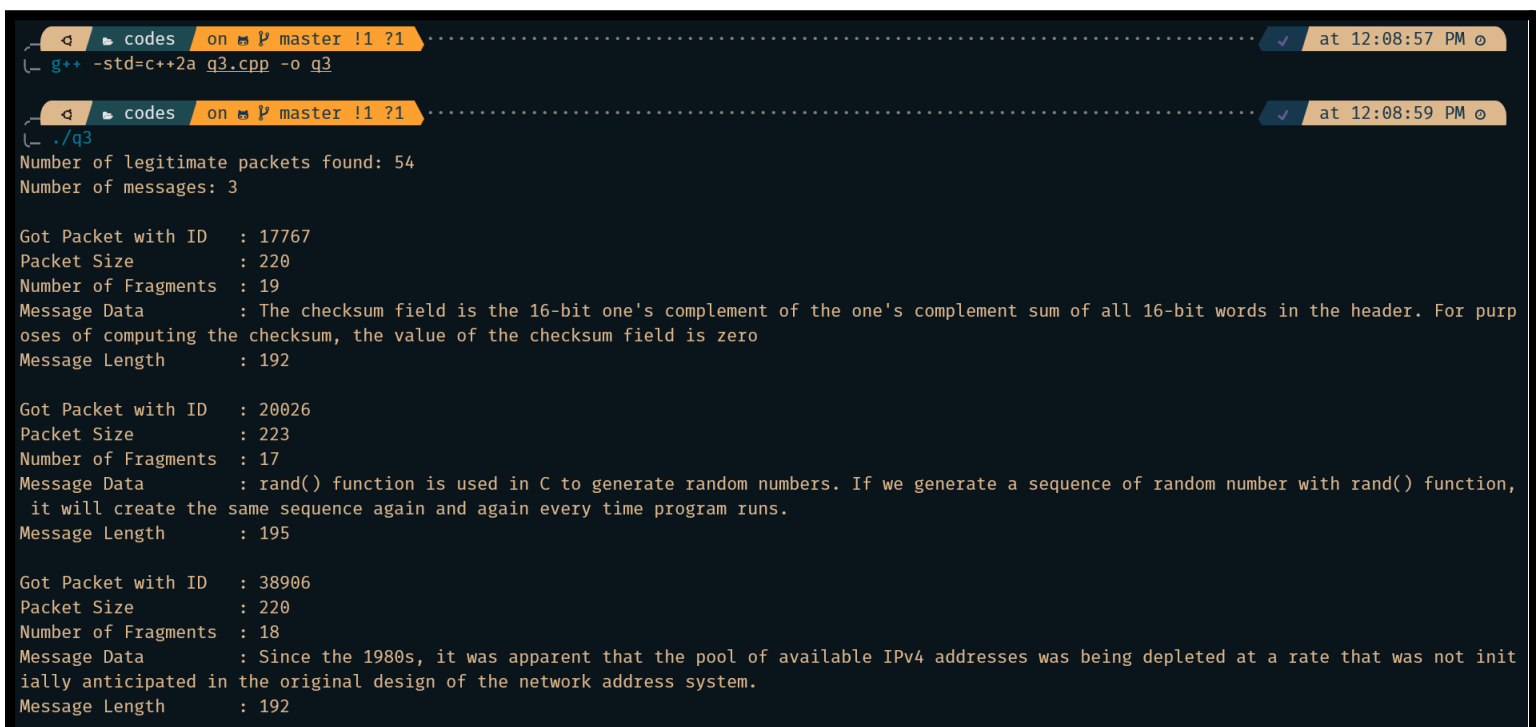
Q3. Assemble the 54 legitimate IPv4 fragments using the fragment flag and fragment offset information contained in the IPv4 header. How many IPv4 packets did you obtain after the assembly? What are the sizes of these packets? What is the message contained within these packets?

I copied the file from the previous question and moved the logic of the **main** function to a new function **getValidPackets()** that will return the vector of all the legitimate packets as a pointer to the **IPPacket** struct.

I created a function **validateMessage()** which will take all the fragments of a particular message received in sorted order of their fragment offset and will check if the message is fully received correctly or not. It checks two things, first that there should not be any missing fragment in between which is checked by computing the expected offset value of each packet. Second, it checks that the More Fragment bit of all except the last fragment should be 1 and that of the last fragment should be 0. I observed that some fragments were having **0** data length and they were having the same offset value as the fragments before them. So, I simply skipped those fragments as they lead to wrong validation.

In the **main** function, I first get the vector of all the valid packets. Then, I created a map of (**packId**, **vector<IPPacket*>**) and looped all the valid packets, and clubbed the fragments having the same identifier value into this map. Then, to construct the final message, I take the clubbed fragments for each packet ID and sort them in increasing order of their fragmentation offset, and validated them. If it was valid, I concatenated the data of the individual fragments to get the final assembled message.

Here is the screenshot of the results obtained.



```
codes on master !1 ?1 at 12:08:57 PM
g++ -std=c++2a q3.cpp -o q3

codes on master !1 ?1 at 12:08:59 PM
./q3
Number of legitimate packets found: 54
Number of messages: 3

Got Packet with ID : 17767
Packet Size : 220
Number of Fragments : 19
Message Data : The checksum field is the 16-bit one's complement sum of all 16-bit words in the header. For purposes of computing the checksum, the value of the checksum field is zero
Message Length : 192

Got Packet with ID : 20026
Packet Size : 223
Number of Fragments : 17
Message Data : rand() function is used in C to generate random numbers. If we generate a sequence of random number with rand() function, it will create the same sequence again and again every time program runs.
Message Length : 195

Got Packet with ID : 38906
Packet Size : 220
Number of Fragments : 18
Message Data : Since the 1980s, it was apparent that the pool of available IPv4 addresses was being depleted at a rate that was not initially anticipated in the original design of the network address system.
Message Length : 192
```