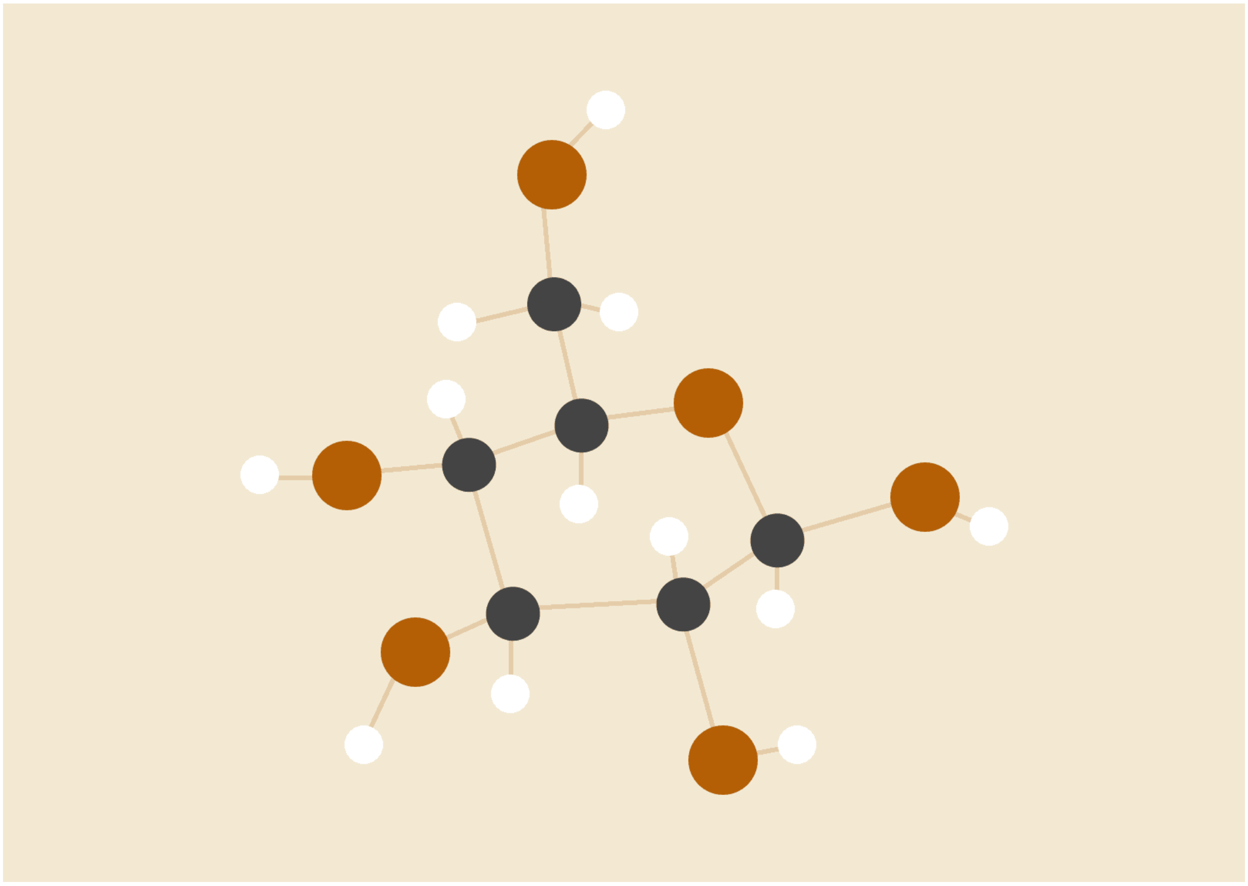
# Computer Networks Lab 9



**Alisetti Sai Vamsi**

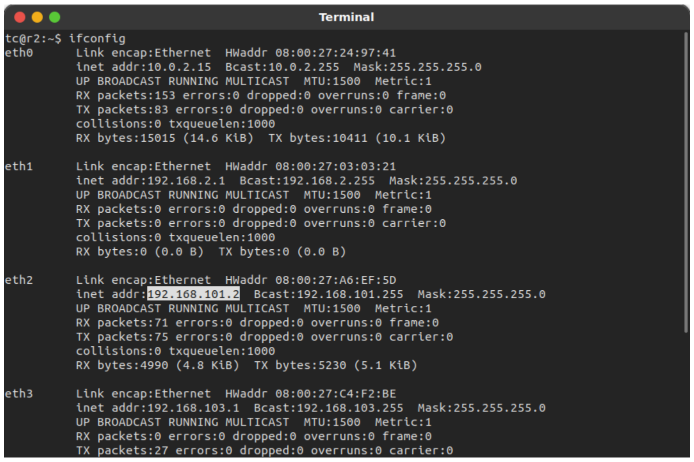
12/11/2021

111801002

## 1. Finding MTU between r1 & r2.

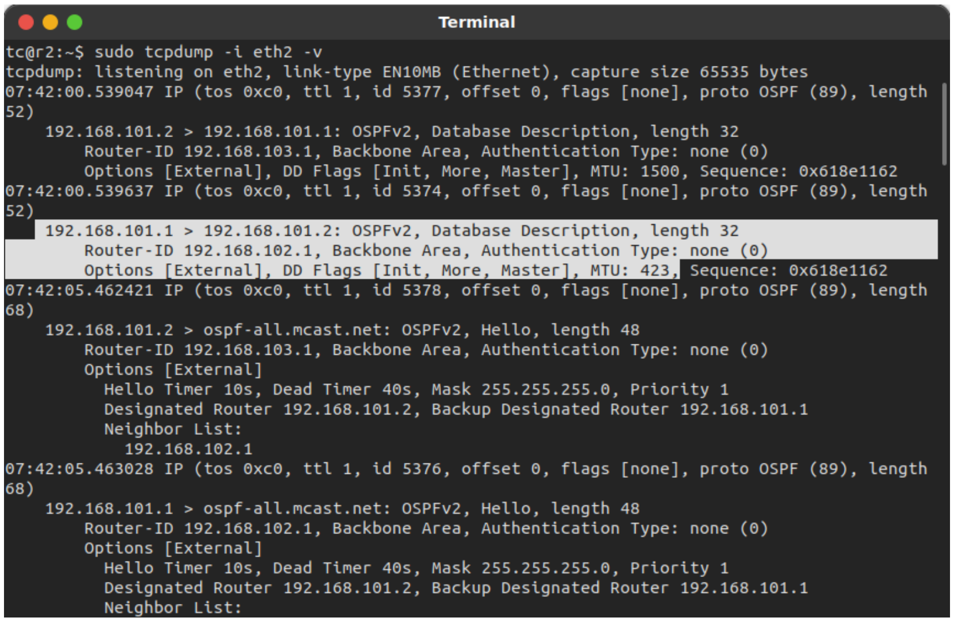
**Definition (MTU):**

The maximum transmission unit (MTU) is the size of the largest protocol data unit (PDU) that can be communicated in a single network layer transaction.



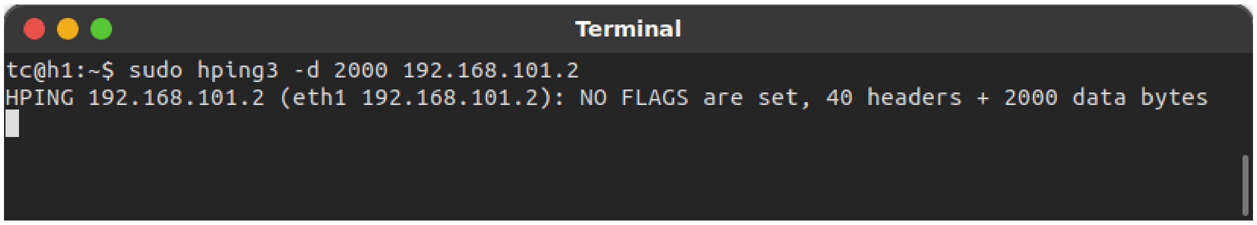
```
tc@r2:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:24:97:41
          inet addr:10.0.2.15  Bcast:10.0.2.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:153 errors:0 dropped:0 overruns:0 frame:0
          TX packets:83 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:15015 (14.6 KiB)  TX bytes:10411 (10.1 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:71 errors:0 dropped:0 overruns:0 frame:0
          TX packets:75 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:4990 (4.8 KiB)  TX bytes:5230 (5.1 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:27 errors:0 dropped:0 overruns:0 carrier:0
```

***r1 is connected to r2 on the interface eth2***

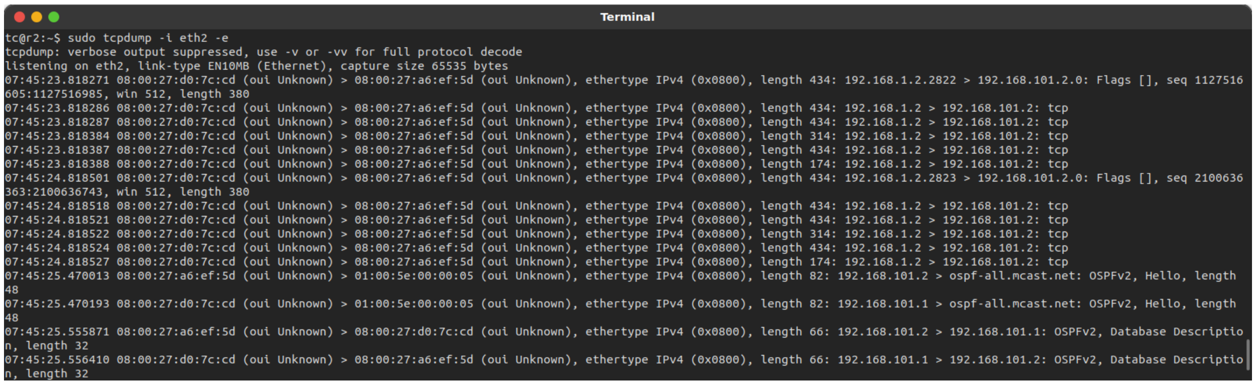


```
tc@r2:~$ sudo tcpdump -i eth2 -v
tcpdump: listening on eth2, link-type EN10MB (Ethernet), capture size 65535 bytes
07:42:00.539047 IP (tos 0xc0, ttl 1, id 5377, offset 0, flags [none], proto OSPF (89), length
52)
    192.168.101.2 > 192.168.101.1: OSPFv2, Database Description, length 32
        Router-ID 192.168.103.1, Backbone Area, Authentication Type: none (0)
        Options [External], DD Flags [Init, More, Master], MTU: 1500, Sequence: 0x618e1162
07:42:00.539637 IP (tos 0xc0, ttl 1, id 5374, 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: 0x618e1162
07:42:05.462421 IP (tos 0xc0, ttl 1, id 5378, offset 0, flags [none], proto OSPF (89), length
68)
    192.168.101.2 > ospf-all.mcast.net: 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
07:42:05.463028 IP (tos 0xc0, ttl 1, id 5376, offset 0, flags [none], proto OSPF (89), length
68)
    192.168.101.1 > ospf-all.mcast.net: 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:
```

***MTU of r1 is 423***

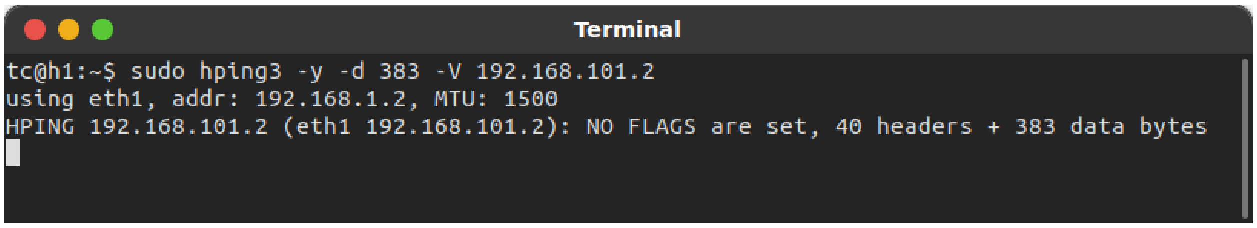


```
tc@h1:~$ sudo hping3 -d 2000 192.168.101.2
HPING 192.168.101.2 (eth1 192.168.101.2): NO FLAGS are set, 40 headers + 2000 data bytes
```

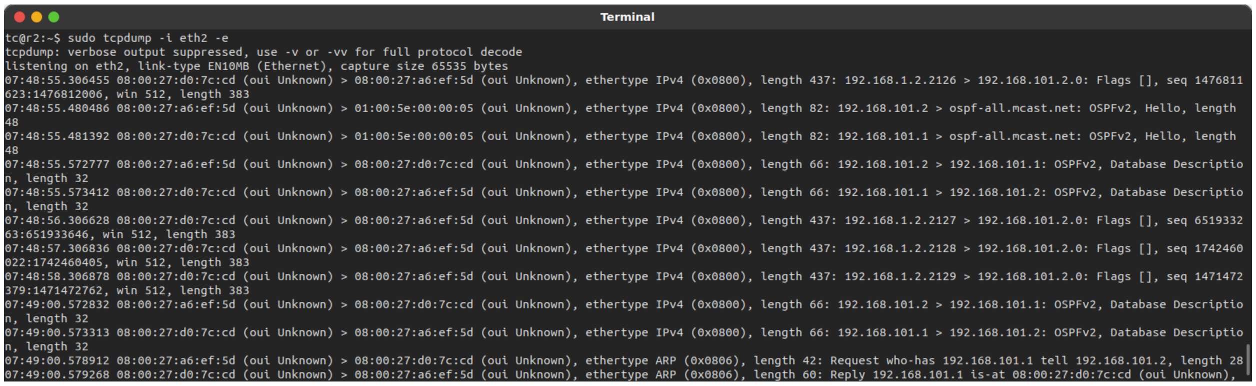


```
tc@r2:~$ sudo tcpdump -i eth2 -e
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth2, link-type EN10MB (Ethernet), capture size 65535 bytes
07:45:23.818271 08:00:27:d0:7c:cd (oui Unknown) > 08:00:27:a6:ef:5d (oui Unknown), ethertype IPv4 (0x0800), length 434: 192.168.1.2.2822 > 192.168.101.2.0: Flags [], seq 1127516
605:1127516985, win 512, length 380
07:45:23.818286 08:00:27:d0:7c:cd (oui Unknown) > 08:00:27:a6:ef:5d (oui Unknown), ethertype IPv4 (0x0800), length 434: 192.168.1.2 > 192.168.101.2: tcp
07:45:23.818287 08:00:27:d0:7c:cd (oui Unknown) > 08:00:27:a6:ef:5d (oui Unknown), ethertype IPv4 (0x0800), length 434: 192.168.1.2 > 192.168.101.2: tcp
07:45:23.818384 08:00:27:d0:7c:cd (oui Unknown) > 08:00:27:a6:ef:5d (oui Unknown), ethertype IPv4 (0x0800), length 314: 192.168.1.2 > 192.168.101.2: tcp
07:45:23.818387 08:00:27:d0:7c:cd (oui Unknown) > 08:00:27:a6:ef:5d (oui Unknown), ethertype IPv4 (0x0800), length 434: 192.168.1.2 > 192.168.101.2: tcp
07:45:23.818388 08:00:27:d0:7c:cd (oui Unknown) > 08:00:27:a6:ef:5d (oui Unknown), ethertype IPv4 (0x0800), length 174: 192.168.1.2 > 192.168.101.2: tcp
07:45:24.818501 08:00:27:d0:7c:cd (oui Unknown) > 08:00:27:a6:ef:5d (oui Unknown), ethertype IPv4 (0x0800), length 434: 192.168.1.2.2823 > 192.168.101.2.0: Flags [], seq 2100636
363:2100636743, win 512, length 380
07:45:24.818518 08:00:27:d0:7c:cd (oui Unknown) > 08:00:27:a6:ef:5d (oui Unknown), ethertype IPv4 (0x0800), length 434: 192.168.1.2 > 192.168.101.2: tcp
07:45:24.818521 08:00:27:d0:7c:cd (oui Unknown) > 08:00:27:a6:ef:5d (oui Unknown), ethertype IPv4 (0x0800), length 434: 192.168.1.2 > 192.168.101.2: tcp
07:45:24.818522 08:00:27:d0:7c:cd (oui Unknown) > 08:00:27:a6:ef:5d (oui Unknown), ethertype IPv4 (0x0800), length 314: 192.168.1.2 > 192.168.101.2: tcp
07:45:24.818524 08:00:27:d0:7c:cd (oui Unknown) > 08:00:27:a6:ef:5d (oui Unknown), ethertype IPv4 (0x0800), length 434: 192.168.1.2 > 192.168.101.2: tcp
07:45:24.818527 08:00:27:d0:7c:cd (oui Unknown) > 08:00:27:a6:ef:5d (oui Unknown), ethertype IPv4 (0x0800), length 174: 192.168.1.2 > 192.168.101.2: tcp
07:45:25.470013 08:00:27:a6:ef:5d (oui Unknown) > 01:00:5e:00:00:05 (oui Unknown), ethertype IPv4 (0x0800), length 82: 192.168.101.2 > ospf-all.mcast.net: OSPFv2, Hello, length
48
07:45:25.470193 08:00:27:d0:7c:cd (oui Unknown) > 01:00:5e:00:00:05 (oui Unknown), ethertype IPv4 (0x0800), length 82: 192.168.101.1 > ospf-all.mcast.net: OSPFv2, Hello, length
48
07:45:25.555871 08:00:27:a6:ef:5d (oui Unknown) > 08:00:27:d0:7c:cd (oui Unknown), ethertype IPv4 (0x0800), length 66: 192.168.101.2 > 192.168.101.1: OSPFv2, Database Descriptio
n, length 32
07:45:25.556410 08:00:27:d0:7c:cd (oui Unknown) > 08:00:27:a6:ef:5d (oui Unknown), ethertype IPv4 (0x0800), length 66: 192.168.101.1 > 192.168.101.2: OSPFv2, Database Descriptio
n, length 32
```

***If we consider link layer header and fragmentation then MTU is 434***



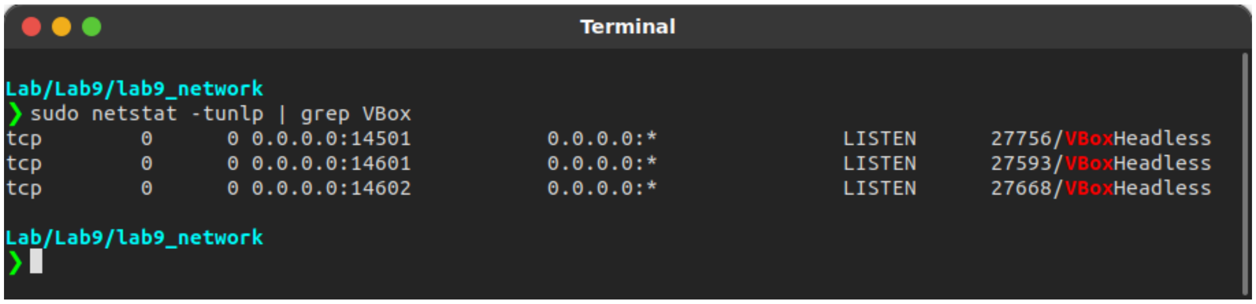
```
tc@h1:~$ sudo hping3 -y -d 383 -V 192.168.101.2
using eth1, addr: 192.168.1.2, MTU: 1500
HPING 192.168.101.2 (eth1 192.168.101.2): NO FLAGS are set, 40 headers + 383 data bytes
```



```
tc@r2:~$ sudo tcpdump -i eth2 -e
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth2, link-type EN10MB (Ethernet), capture size 65535 bytes
07:48:55.306455 08:00:27:d0:7c:cd (oui Unknown) > 08:00:27:a6:ef:5d (oui Unknown), ethertype IPv4 (0x0800), length 437: 192.168.1.2.2126 > 192.168.101.2.0: Flags [], seq 1476811
623:1476812006, win 512, length 383
07:48:55.480486 08:00:27:a6:ef:5d (oui Unknown) > 01:00:5e:00:00:05 (oui Unknown), ethertype IPv4 (0x0800), length 82: 192.168.101.2 > ospf-all.mcast.net: OSPFv2, Hello, length
48
07:48:55.481392 08:00:27:d0:7c:cd (oui Unknown) > 01:00:5e:00:00:05 (oui Unknown), ethertype IPv4 (0x0800), length 82: 192.168.101.1 > ospf-all.mcast.net: OSPFv2, Hello, length
48
07:48:55.572777 08:00:27:a6:ef:5d (oui Unknown) > 08:00:27:d0:7c:cd (oui Unknown), ethertype IPv4 (0x0800), length 66: 192.168.101.2 > 192.168.101.1: OSPFv2, Database Descriptio
n, length 32
07:48:55.573412 08:00:27:d0:7c:cd (oui Unknown) > 08:00:27:a6:ef:5d (oui Unknown), ethertype IPv4 (0x0800), length 66: 192.168.101.1 > 192.168.101.2: OSPFv2, Database Descriptio
n, length 32
07:48:56.306628 08:00:27:d0:7c:cd (oui Unknown) > 08:00:27:a6:ef:5d (oui Unknown), ethertype IPv4 (0x0800), length 437: 192.168.1.2.2127 > 192.168.101.2.0: Flags [], seq 6519332
63:651933646, win 512, length 383
07:48:57.306836 08:00:27:d0:7c:cd (oui Unknown) > 08:00:27:a6:ef:5d (oui Unknown), ethertype IPv4 (0x0800), length 437: 192.168.1.2.2128 > 192.168.101.2.0: Flags [], seq 1742460
022:1742460405, win 512, length 383
07:48:58.306878 08:00:27:d0:7c:cd (oui Unknown) > 08:00:27:a6:ef:5d (oui Unknown), ethertype IPv4 (0x0800), length 437: 192.168.1.2.2129 > 192.168.101.2.0: Flags [], seq 1471472
379:1471472762, win 512, length 383
07:49:00.572832 08:00:27:a6:ef:5d (oui Unknown) > 08:00:27:d0:7c:cd (oui Unknown), ethertype IPv4 (0x0800), length 66: 192.168.101.2 > 192.168.101.1: OSPFv2, Database Descriptio
n, length 32
07:49:00.573313 08:00:27:d0:7c:cd (oui Unknown) > 08:00:27:a6:ef:5d (oui Unknown), ethertype IPv4 (0x0800), length 66: 192.168.101.1 > 192.168.101.2: OSPFv2, Database Descriptio
n, length 32
07:49:00.578912 08:00:27:a6:ef:5d (oui Unknown) > 08:00:27:d0:7c:cd (oui Unknown), ethertype ARP (0x0806), length 42: Request who-has 192.168.101.1 tell 192.168.101.2, length 28
07:49:00.579268 08:00:27:d0:7c:cd (oui Unknown) > 08:00:27:a6:ef:5d (oui Unknown), ethertype ARP (0x0806), length 60: Reply 192.168.101.1 is-at 08:00:27:d0:7c:cd (oui Unknown),
```

***If we consider link layer header and no fragmentation then MTU is 437***

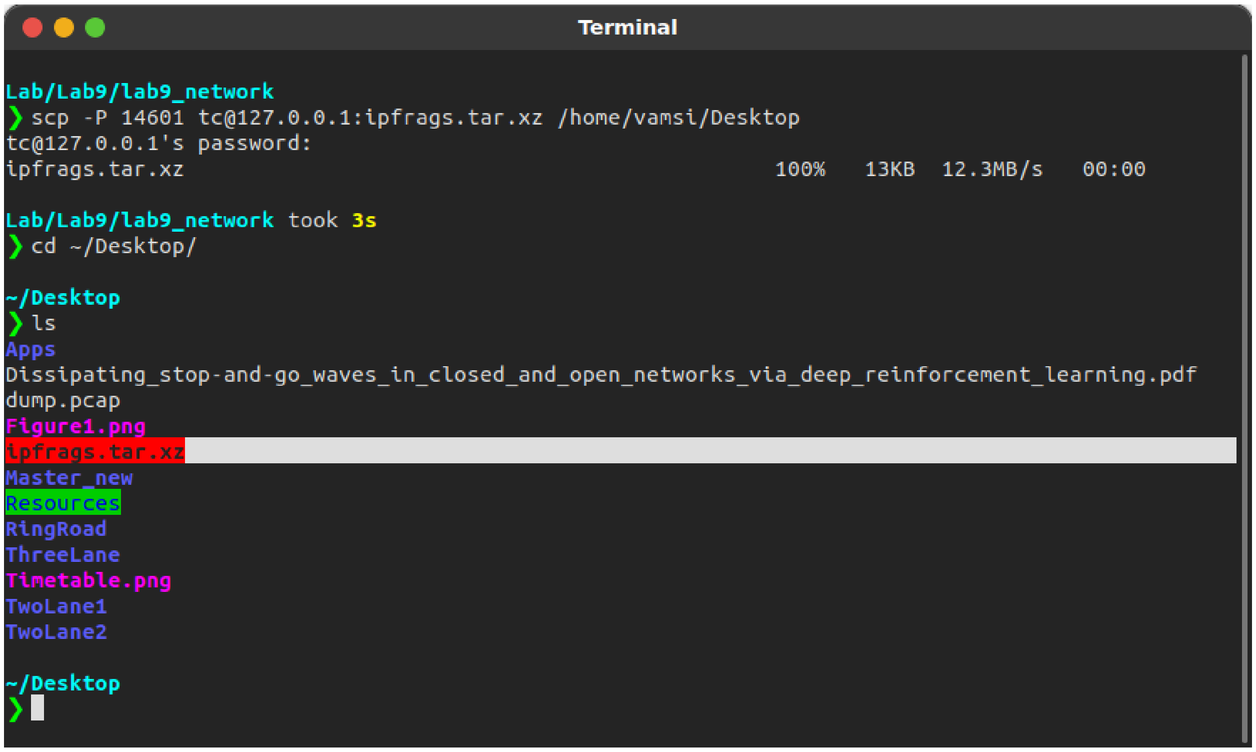
Upon trying the values of 423, 434 and 437 as the passwords, 434 seems to work although the correct answer is 423 since link layer header should not be considered in the calculation of MTU. Therefore password for r1 is **user@434.** But in fact it should've been **user@423.**



```
Terminal

Lab/Lab9/lab9_network
❯ sudo netstat -tunlp | grep VBox
tcp        0      0 0.0.0.0:14501           0.0.0.0:*               LISTEN      27756/VBoxHeadless
tcp        0      0 0.0.0.0:14601           0.0.0.0:*               LISTEN      27593/VBoxHeadless
tcp        0      0 0.0.0.0:14602           0.0.0.0:*               LISTEN      27668/VBoxHeadless

Lab/Lab9/lab9_network
❯
```

***We can see that r2 is running on port 14601***
***(Since in connect.sh we can see that h1 is mapped to 14501 and r2 is mapped to 14602)***

To get the ipfrags.tar.xz file, we can use **scp** command using the loopback address from the host machine.



```
Terminal

Lab/Lab9/lab9_network
❯ scp -P 14601 tc@127.0.0.1:ipfrags.tar.xz /home/vamsi/Desktop
tc@127.0.0.1's password:
ipfrags.tar.xz                                100%   13KB  12.3MB/s   00:00

Lab/Lab9/lab9_network took 3s
❯ cd ~/Desktop/

~/Desktop
❯ ls
Apps
Dissipating_stop-and-go_waves_in_closed_and_open_networks_via_deep_reinforcement_learning.pdf
dump.pcap
Figure1.png
ipfrags.tar.xz
Master_new
Resources
RingRoad
ThreeLane
Timetable.png
TwoLane1
TwoLane2

~/Desktop
❯
```

Methodology: First i used hping3 with fragmentation and sent a data packet of 1000 bytes from h1 to r2, and listened on eth2 interface on r2. I could see that i was receiving a packet of length 420. Now i used hping3 without fragmentation and tried sending a data packet of length 420 upon which i received an ICMP error message from r1 saying DF bit not set. So i kept decrementing the data packet by 10 and kept trying. And upon reaching 380 the transmission was successful. Then I kept incrementing by 1 and checked if the data was being sent or not. On 384 I was getting the same error message. This meant that a data of 383 + 40(TCP + IP header) = 423 made the MTU. But the question assumes the link layer to be a part of MTU and fragmentation to be set, so I used tcpdump with -e option and passed 383 data bytes with fragmentation, and I received 434 bytes on r2. Without fragmentation it comes to 437 bytes.

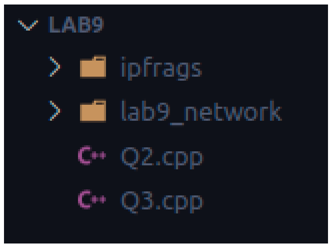
## 2. Validating fragments using checksum

**Program name:** Q2.cpp
**Invocation**:

```
$ g++ Q2.cpp -o Q2
$ ./Q2
```

**Initialization:**

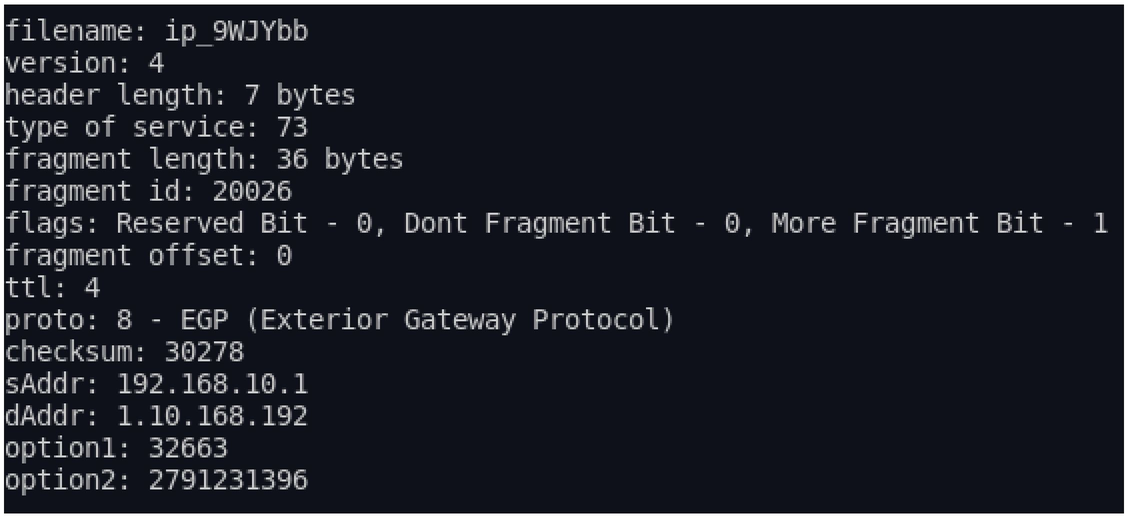
1. Initialize the directory name to be "ipfrags/"
2. Directory structure should be in this format



**Algorithm Outline:**

1. Loop through all the files in the ipfrags directory
    a. if the filename is not "." or ".." (these are hidden files in the directory, we have to skip these)
        i. Initialize IPPacket structure
        ii. Open the file using fopen
        iii. read the file content using fread and write it to the IPPacket structure
        iv. Typecast the pointer to the structure with a (short *) pointer (since a short reads 16 bits from the start of the pointer)
        v. Keep extracting 16 bits this way until the end of the header(i.e for 14 times, since header length is 28 bytes = 28 x 8 bits = (28 x 8)/16 = 14 16-bit blocks ) and store a running binary sum - In this binary sum the overflow bits are added back to the sum
        vi. Compare the sum to 16-bit all ones binary value
            1. if they are equal then
                a. the header is valid

        b. print the header details
    2. Else
        a. go to the next file



```
filename: ip_9WJYbb
version: 4
header length: 7 bytes
type of service: 73
fragment length: 36 bytes
fragment id: 20026
flags: Reserved Bit - 0, Dont Fragment Bit - 0, More Fragment Bit - 1
fragment offset: 0
ttl: 4
proto: 8 - EGP (Exterior Gateway Protocol)
checksum: 30278
sAddr: 192.168.10.1
dAddr: 1.10.168.192
option1: 32663
option2: 2791231396
```

***Example of a header detail of one of the fragment***

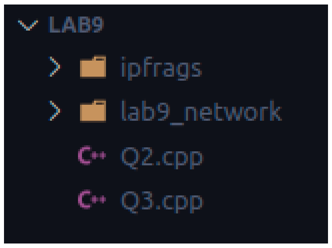
## 3. Reconstructing packets from fragments

**Program name:** Q3.cpp
**Invocation**:

```
$ g++ Q3.cpp -o Q3
$ ./Q3
```

**Initialization:**

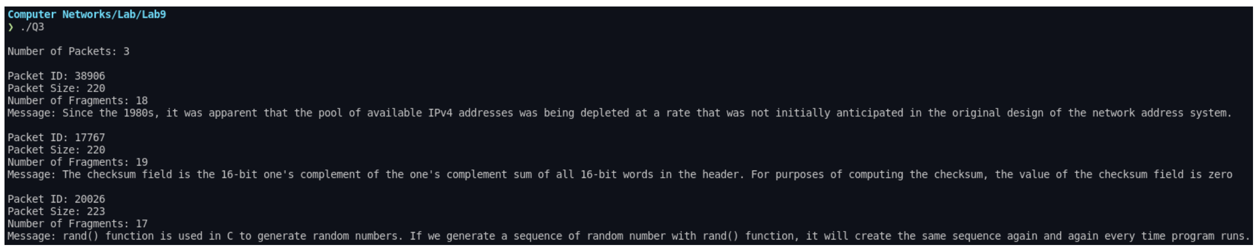
1. Initialize the directory name to be "ipfrags/"
2. Directory structure should be in this format



```
LAB9
  > ipfrags
  > lab9_network
    Q2.cpp
    Q3.cpp
```

**Algorithm Outline:**

1. Loop through all the files in the ipfrags directory
    a. if the filename is not "." or ".." (these are hidden files in the directory, we have to skip these)
        i. Initialize IPPacket structure
        ii. Open the file using fopen
        iii. read the file content using fread and write it to the IPPacket structure
        iv. Typecast the pointer to the structure with a (short *) pointer (since a short reads 16 bits from the start of the pointer)
        v. Keep extracting 16 bits this way until the end of the header(i.e for 14 times, since header length is 28 bytes = 28 x 8 bits = (28 x 8)/16 = 14 16-bit blocks ) and store a running binary sum - In this binary sum the overflow bits are added back to the sum
        vi. Compare the sum to 16-bit all ones binary value
            1. if they are equal then (header is valid)
                a. Push into the fragments vector
            3. Else
                a. go to the next file
2. Create a map of <id, vector<IPPacket*>> which stores the fragment id and the fragments having this fragment id.
    a. Loop through all the stored fragments having valid headers
        i. Insert into the map
3. for each (id, vector<IPPacket*>) pair in the map
    a. sort all the fragments in vector<IPPacket*> according to the offset value of each fragment
    b. for each fragment in the vector<IPPacket*>
        i. calculate the datalen = total packet length - header length
        ii. skip to the end of the header using an int pointer (incrementing the pointer by 7 times skips 7*4 bytes = 28 bytes (header length))
        iii. capture datalen amount of bytes (typecast the pointer into a char pointer since each char is of 1 byte, increment this pointer by datalen times and on each increment save the character)
        iv. concatenate the captured data to a running string
    c. print the running string



```
Computer Networks/Lab/Lab9
❯ ./Q3

Number of Packets: 3

Packet ID: 38906
Packet Size: 220
Number of Fragments: 18
Message: 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.

Packet ID: 17767
Packet Size: 220
Number of Fragments: 19
Message: The checksum field is the 16-bit one's complement of the 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

Packet ID: 20026
Packet Size: 223
Number of Fragments: 17
Message: 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.
```