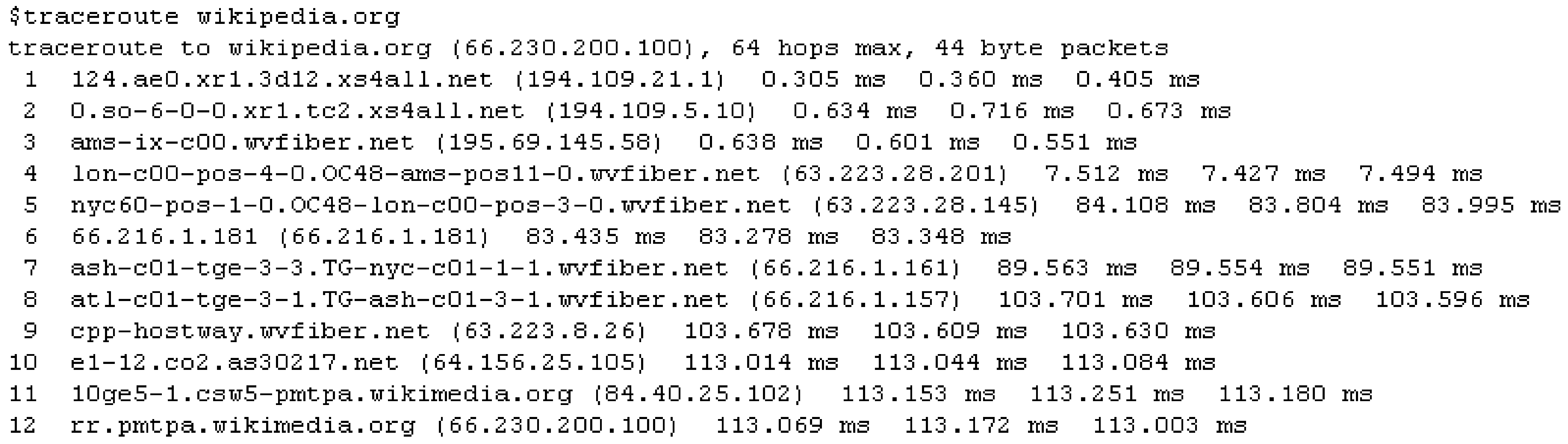
# Tutorial 7: Analysis of IP Protocol

**Objective: Write a python program to analyze a trace of IP datagrams**

# Background of Traceroute

- A tool for displaying possible routes (paths) and measuring transit delays of packets across an Internet Protocol (IP) network



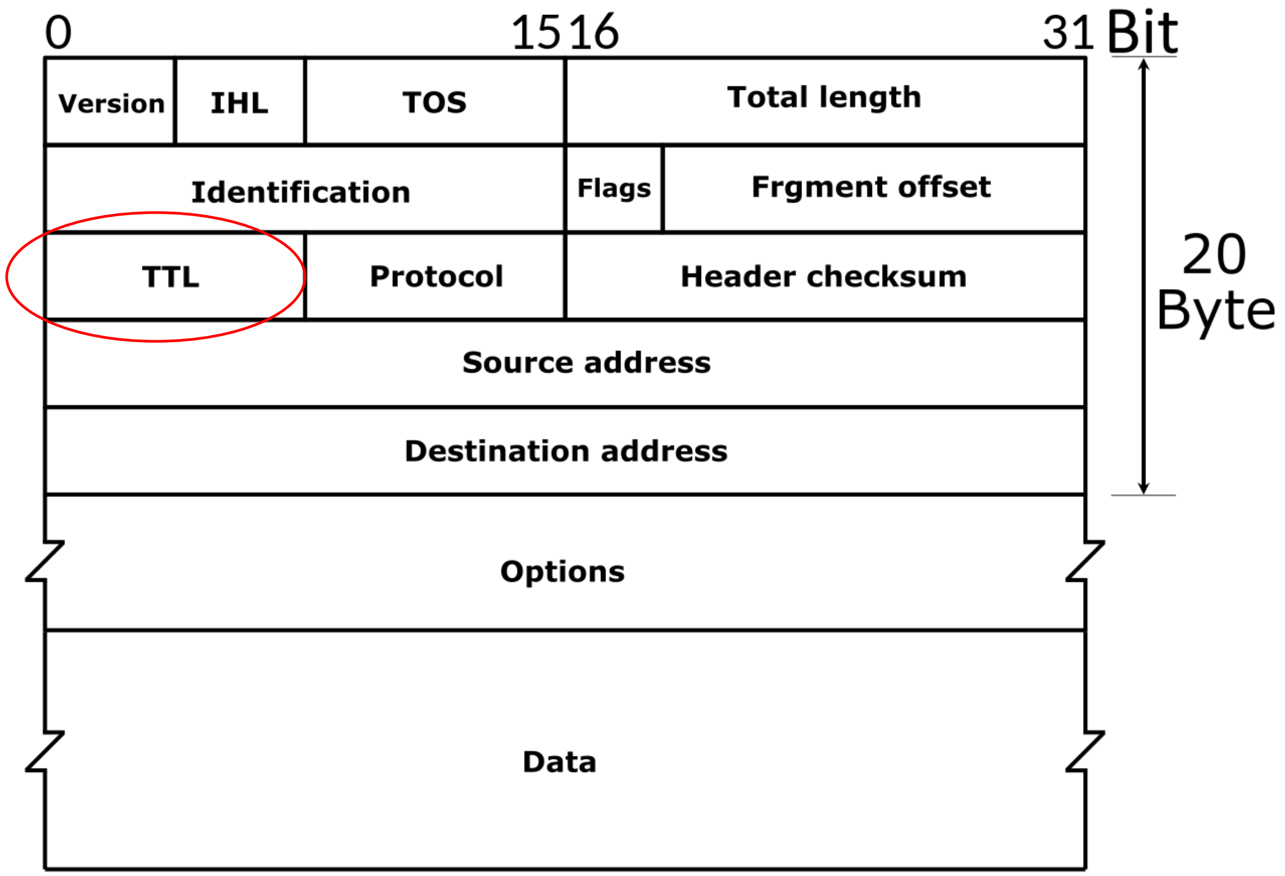
```
$traceroute wikipedia.org
traceroute to wikipedia.org (66.230.200.100), 64 hops max, 44 byte packets
 1  124.ae0.xr1.3d12.xs4all.net (194.109.21.1)  0.305 ms  0.360 ms  0.405 ms
 2  0.so-6-0-0.xr1.tc2.xs4all.net (194.109.5.10)  0.634 ms  0.716 ms  0.673 ms
 3  ams-ix-c00.wvfiber.net (195.69.145.58)  0.638 ms  0.601 ms  0.551 ms
 4  lon-c00-pos-4-0.OC48-ams-pos11-0.wvfiber.net (63.223.28.201)  7.512 ms  7.427 ms  7.494 ms
 5  nyc60-pos-1-0.OC48-lon-c00-pos-3-0.wvfiber.net (63.223.28.145)  84.108 ms  83.804 ms  83.995 ms
 6  66.216.1.181 (66.216.1.181)  83.435 ms  83.278 ms  83.348 ms
 7  ash-c01-tge-3-3.TG-nyc-c01-1-1.wvfiber.net (66.216.1.161)  89.563 ms  89.554 ms  89.551 ms
 8  atl-c01-tge-3-1.TG-ash-c01-3-1.wvfiber.net (66.216.1.157)  103.701 ms  103.606 ms  103.596 ms
 9  cpp-hostway.wvfiber.net (63.223.8.26)  103.678 ms  103.609 ms  103.630 ms
10  e1-12.co2.as30217.net (64.156.25.105)  113.014 ms  113.044 ms  113.084 ms
11  10ge5-1.csw5-pmtpa.wikimedia.org (84.40.25.102)  113.153 ms  113.251 ms  113.180 ms
12  rr.pmtpa.wikimedia.org (66.230.200.100)  113.069 ms  113.172 ms  113.003 ms
```

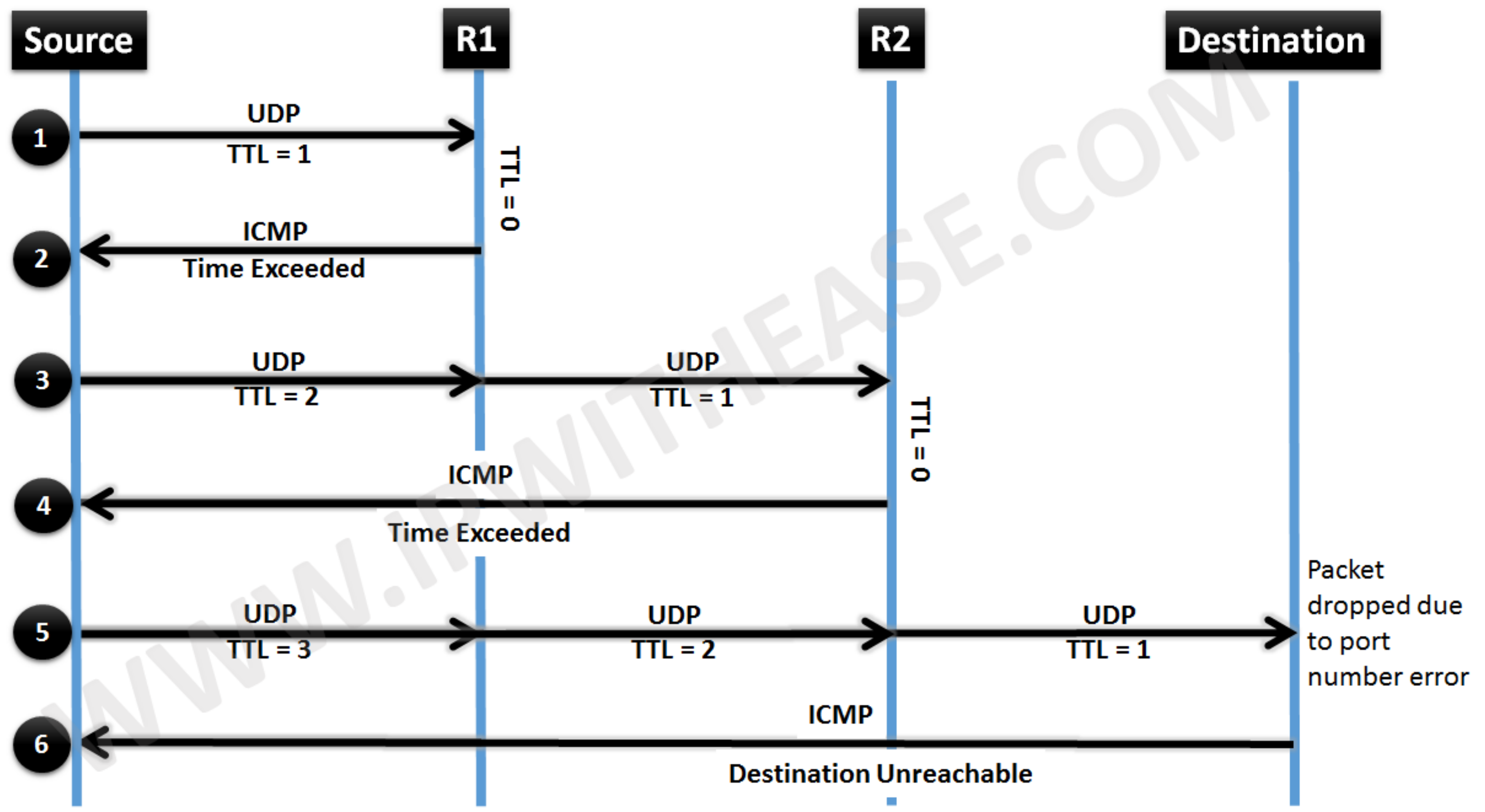
# How Traceroute Works?

- Time-to-live field

A router must decrement the TTL in each received datagram by 1

If TTL = 0, the router returns an ICMP message to the sending host





*** Each set of communication happens 3 times i.e. Set 1&2 , Set 3 &4 and Set 5&6

## P3 Requirement 1 (R1)

- Write a python program to analyze the trace of IP datagrams created by Traceroute:
  - List the IP addresses of the source node, the ultimate destination node, and the intermediate destination nodes (ordered by hop count to source node in the increasing order).

  - Check the IP header of all datagrams in the trace file, and list the set of values in the protocol field of the IP headers. Note that only different values should be listed in a set.

- How many fragments were created from the original datagram? Note that 0 means no fragmentation. Print out the offset (in terms of bytes) of the last fragment of the fragmented IP datagram. Note that if the datagram is not fragmented, the offset is 0.

- Calculate the average and standard deviation of round trip time (RTT) between the source node and the intermediate destination node (s) and the average round trip time between the source node and the ultimate destination node. The average and standard deviation are calculated over all fragments sent/received between the source nodes and the (intermediate/ultimate) destination node.

# Output format for R1

```
The IP address of the source node: 192.168.1.12
The IP address of ultimate destination node: 12.216.216.2
The IP addresses of the intermediate destination nodes:
      router 1: 24.218.01.102,
      router 2: 24.221.10.103,
      router 3: 12.216.118.1.

The values in the protocol field of IP headers:
      1: ICMP
      17: UDP


The number of fragments created from the original datagram is: 3

The offset of the last fragment is: 3680

The avg RTT between 192.168.1.12 and 24.218.01.102 is: 50 ms, the s.d. is: 5 ms
The avg RTT between 192.168.1.12 and 24.221.10.103 is: 100 ms, the s.d. is: 6 ms
The avg RTT between 192.168.1.12 and 12.216.118.1 is: 150 ms, the s.d. is: 5 ms
The avg RTT between 192.168.1.12 and 12.216.216.2 is: 200 ms, the s.d. is: 15 ms
```

## P3 Requirement 2 (R2)

- You can finish this part either with a python program or by manually collecting/analyzing data
- From a given set of five traceroute trace files, all with the same destination address,
  - determine the number of probes per “ttl” used in each trace file,
  - determine whether or not the sequence of intermediate routers is the same in different trace files,
  - if the sequence of intermediate routers is different in the five trace files, list the difference and explain why,

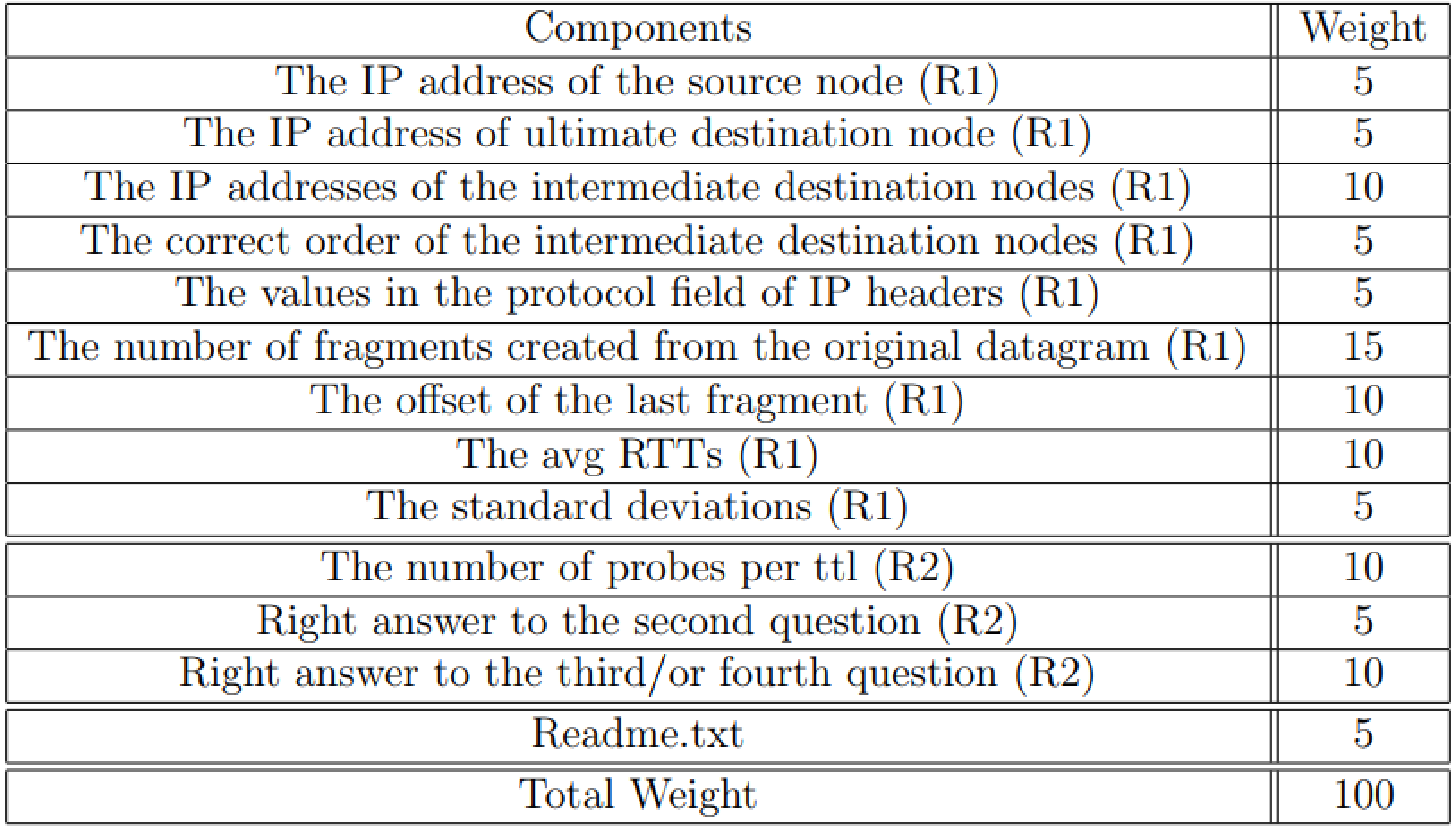
- if the sequence of intermediate routers is the same in the five trace files, draw a table as shown below (warning: the values in the table do not correspond to any trace files) to compare the RTTs of different traceroute attempts. From the result, which hop is likely to incur the maximum delay? Explain your conclusion.

| TTL | Average RTT in trace 1 | Average RTT in trace 2 | Average RTT in trace 3 | Average RTT in trace 4 | Average RTT in trace 5 |
|---|---|---|---|---|---|
| 1 | 0.5 | 0.7 | 0.8 | 0.7 | 0.9 |
| 2 | 0.9 | 1 | 1.2 | 1.2 | 1.3 |
| 3 | 1.5 | 1.5 | 1.5 | 1.5 | 2.5 |
| 4 | 2.5 | 2 | 2 | 2.5 | 3 |
| 5 | 3 | 2.5 | 3 | 3.5 | 3.5 |
| 6 | 5 | 4 | 5 | 4.5 | 4 |

## Deliverable

- Source code,
- a readme file,
- a pdf file for your solution of R2.
- Use %tar -czvf command in linux.csc.uvic.ca to generate a .tar file and submit the .tar file

# Marking Scheme



| Components | Weight |
|---|---|
| The IP address of the source node (R1) | 5 |
| The IP address of ultimate destination node (R1) | 5 |
| The IP addresses of the intermediate destination nodes (R1) | 10 |
| The correct order of the intermediate destination nodes (R1) | 5 |
| The values in the protocol field of IP headers (R1) | 5 |
| The number of fragments created from the original datagram (R1) | 15 |
| The offset of the last fragment (R1) | 10 |
| The avg RTTs (R1) | 10 |
| The standard deviations (R1) | 5 |
| The number of probes per ttl (R2) | 10 |
| Right answer to the second question (R2) | 5 |
| Right answer to the third/or fourth question (R2) | 10 |
| Readme.txt | 5 |
| Total Weight | 100 |

## Additional Information

- You can reuse your code in P2 to extract pcap files, but you are not allowed to use python packages that can automatically extract each packet from the pcap files.
- Some intermediate router may only send back one “ICMP TTL exceeded” message for multiple fragments of the same datagram. In this case, please use this ICMP message to calculate RTT for all fragments.
  - For example, Assume that the source sends Frag 1, Frag2, Frag 3 (of the same datagram, ID: 3000). The timestamps for Frag1, Frag2, Frag3 are t1; t2; t3, respectively.
  - Later, the source receives one “ICMP TTL exceeded” message (ID: 3000). The timestamp is T. Then the RTTs are calculated as: $T - t1$, $T - t2$, $T - t3$.

# More explanation about the output format

```
The number of fragments created from the original datagram is:

The offset of the last fragment is:
```

If there are multiple fragmented datagrams, you need to output the above information for each datagram. For example, assume that the source sends two datagrams: $D_1$, $D_2$, where $D_1$ and $D_2$ are the identification of the two datagrams. Assume that $D_1$ has three fragments and $D_2$ has two fragments. Then output should be:

```
The number of fragments created from the original datagram D1 is: 3

The offset of the last fragment is: xxx.


The number of fragments created from the original datagram D2 is: 2

The offset of the last fragment is: yyy.
```

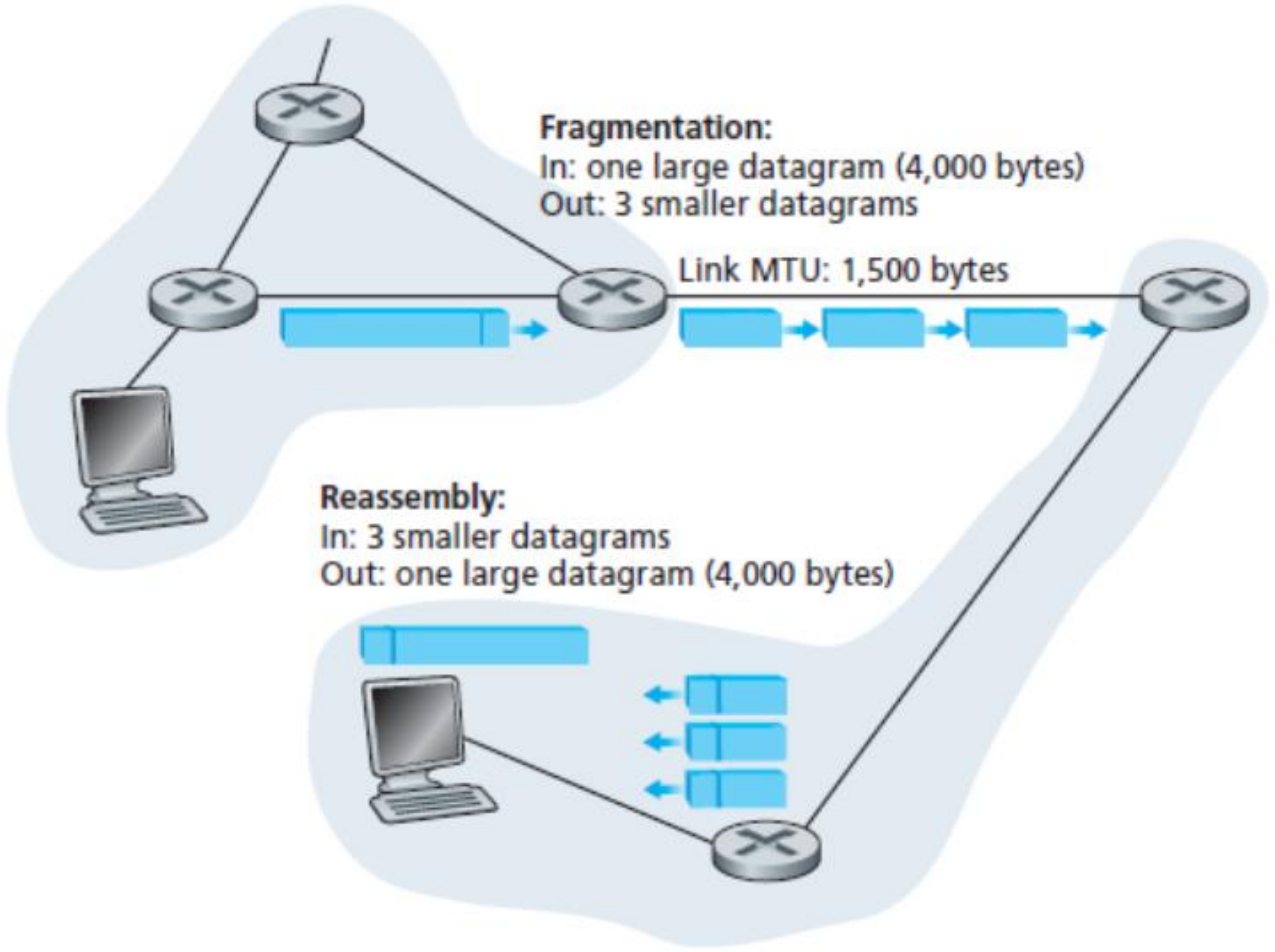
where *xxx* and *yyy* denote the actual number calculated by your program.

- If the tracefile is captured in Linux, the source port number included in the original UDP can be used to match against the ICMP error message

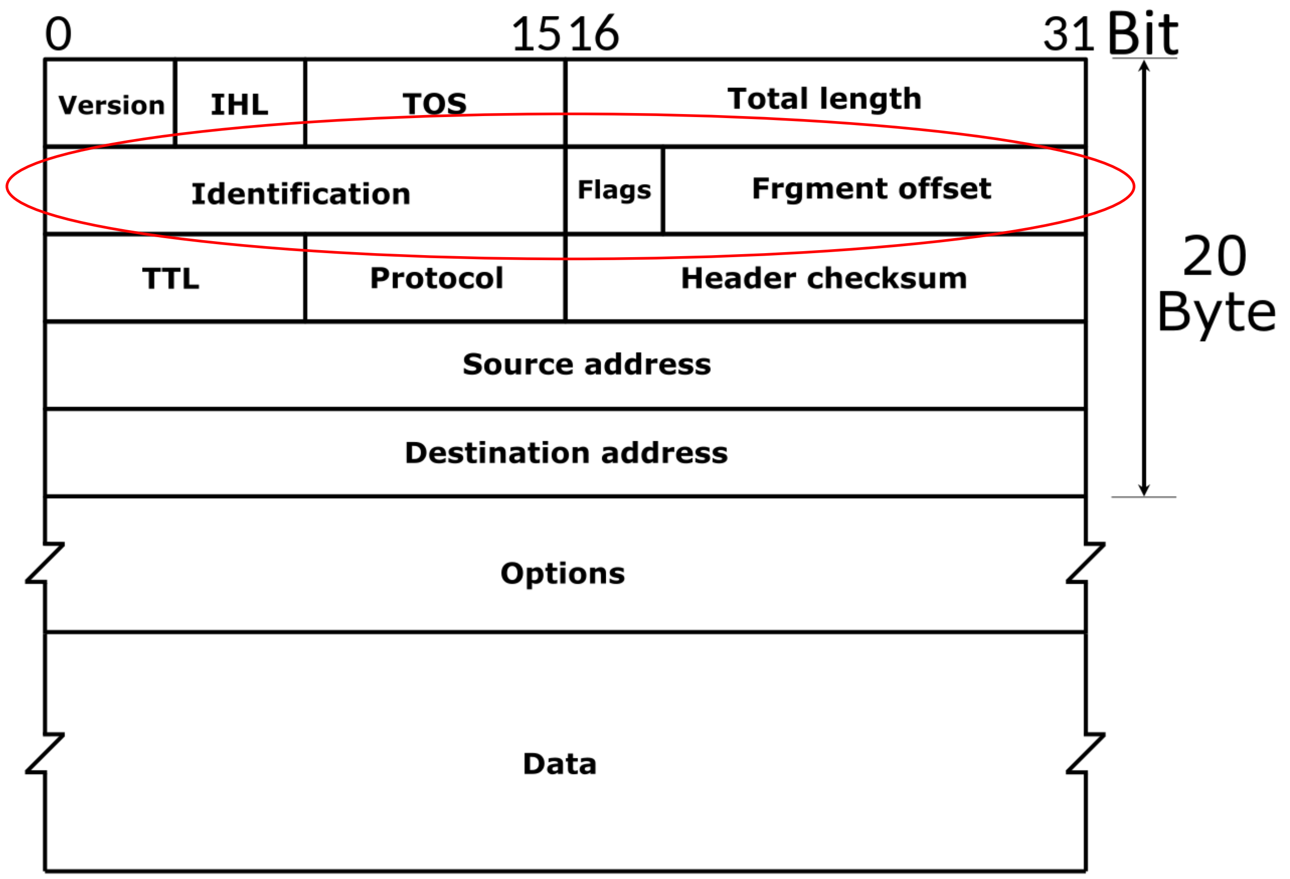
- If the tracefile is captured in Windows, we should use the sequence number in the returned ICMP error message to match the sequence number in the ICMP echo (ping) message from the source node.

- So you need to handle the two scenarios in finding a match between the original datagram and the returned ICMP error message

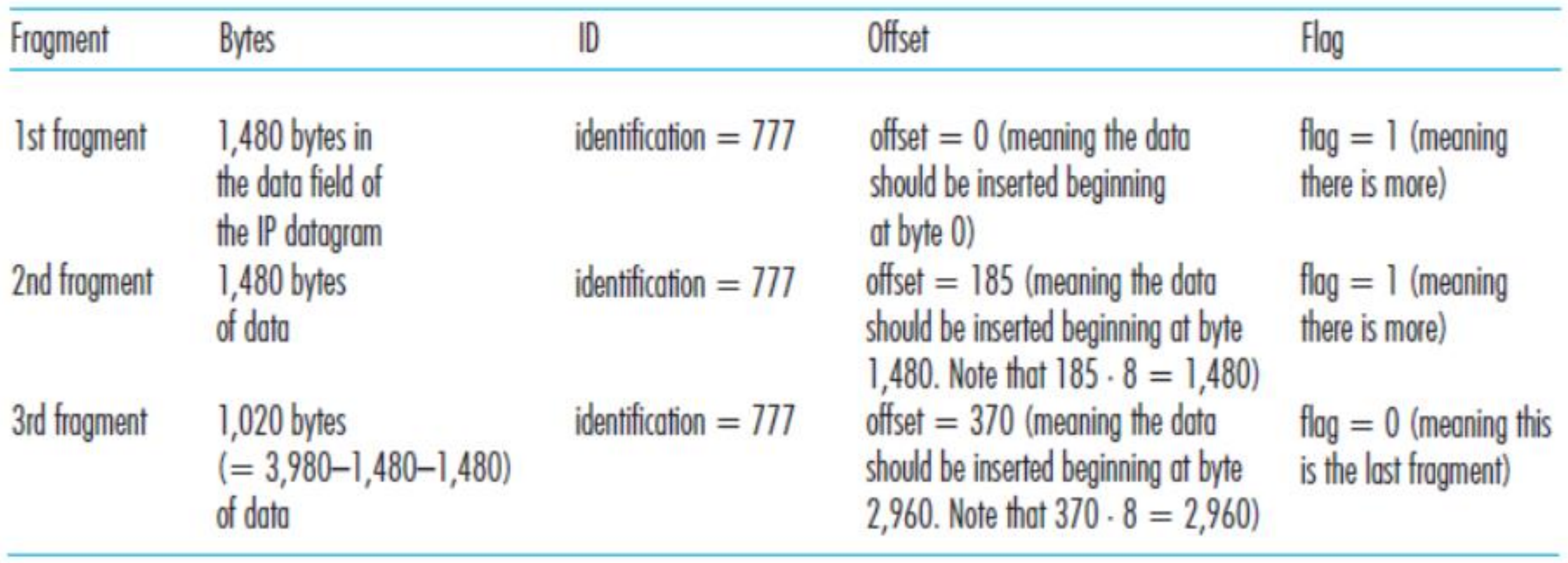
# IP Datagram Fragmentation Example



# IP header


0
15
16
31
Bit
Version
IHL
TOS
Total length
Identification
Flags
Frgment offset
TTL
Protocol
Header checksum
Source address
Destination address
Options
Data
20 Byte

# IP Datagram Fragmentation Example



| Fragment | Bytes | ID | Offset | Flag |
|---|---|---|---|---|
| 1st fragment | 1,480 bytes in the data field of the IP datagram | identification = 777 | offset = 0 (meaning the data should be inserted beginning at byte 0) | flag = 1 (meaning there is more) |
| 2nd fragment | 1,480 bytes of data | identification = 777 | offset = 185 (meaning the data should be inserted beginning at byte 1,480. Note that 185 · 8 = 1,480) | flag = 1 (meaning there is more) |
| 3rd fragment | 1,020 bytes (= 3,980–1,480–1,480) of data | identification = 777 | offset = 370 (meaning the data should be inserted beginning at byte 2,960. Note that 370 · 8 = 2,960) | flag = 0 (meaning this is the last fragment) |