CSc 361: Computer Communications and Networks

Assignment 3: Analysis of IP Protocol

Due: 11:59 pm Nov 30, 2025

1 Goal

The purpose of this assignment is to learn about the IP protocol. You are required to write a python program to analyze a trace of IP datagrams.

2 Introduction

In this assignment, we will investigate the IP protocol, focusing on the IP datagram. We'll do so by analyzing a trace of IP datagrams sent and received by an execution of the *traceroute* program. We will investigate the various fields in the IP datagram, and study IP fragmentation in detail.

A background of the *traceroute* program is summarized as follows. The *traceroute* program operates by first sending one or more datagrams with the time-to-live (TTL) field in the IP header set to 1; it then sends a series of one or more datagrams towards the same destination with a TTL value of 2; it then sends a series of datagrams towards the same destination with a TTL value of 3; and so on. Recall that a router must decrement the TTL in each received datagram by 1 (actually, RFC 791 says that the router must decrement the TTL by at least one). If the TTL reaches 0, the router returns an ICMP message (type 11 – TTL-exceeded) to the sending host. As a result of this behavior, a datagram with a TTL of 1 (sent by the host executing traceroute) will cause the router one hop away from the sender to send an ICMP TTL-exceeded message back to the sender; the datagram sent with a TTL of 2 will cause the router two hops away to send an ICMP message back to the sender; the datagram sent with a TTL of 3 will cause the router three hops away to send an ICMP message back to the sender; and so on. In this manner, the host executing *traceroute* can learn the identities of the routers between itself and a chosen destination by looking at the source IP addresses in the datagrams containing the ICMP TTL-exceeded messages. You will be provided with a trace file created by *traceroute*.

Of course, you can create a trace file by yourself. Note that when you create the trace file, you need to use different datagram sizes (e.g., 2500 bytes) so that the captured trace file includes information on fragmentation.

3 Requirements

There are two requirements for this assignment:

3.1 Requirement 1 (R1)

You are required to write a python program to analyze the trace of IP datagrams created by *traceroute*. To make terminologies consistent, in this assignment we call the *source node* as the computer that executes *traceroute*. The *ultimate destination node* refers to the host that is the ultimate destination defined when running *traceroute*. For example, the ultimate destination node is “mit.edu” when you run

```
%traceroute mit.edu 2000
```

In addition, an *intermediate destination node* refers to the router that is not the ultimate destination node but sends back a ICMP message to the source node.

As another example, you can set “don’t fragment”’ bit and set the number of probes per “ttl” to 5 queries using the following command:

```
%traceroute -F -q 5 mit.edu 200
```

Your program needs to output the following information:

- List the IP address of the source node, the IP address of ultimate destination node, the IP address(es) of the intermediate destination node(s). If multiple intermediate destination nodes exist, they should be ordered by their hop count to the 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.

The output format is as follows: (Note that the values do not correspond to any trace file).

```
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
```

3.2 Requirement 2 (R2)

Note: You can finish this part either with a python program or by manually collecting/analyzing data. In other words, coding is optional for the tasks listed in this section.

You need to test your code on the two groups of trace files provided. Each group includes five *traceroute* trace files, all with the same destination address. For each group,

- 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

4 Miscellaneous

Important! Please read!

- Same as in Assignment 2, you are not allowed in this assignment to use python packages that can automatically extract each packet from the pcap files. That means, you should re-use your code in Assignment 2 to extract packets.
- 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 t_1, t_2, t_3, respectively. Later, the source receives one "ICMP TTL exceeded" message (ID: 3000). The timestamp is T. Then the RTTs are calculated as: $T - t_1$, $T - t_2$, $T - t_3$.

- 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. This is due to the special traceroute implementation in linux, which uses UDP and ICMP. 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. Note that this ICMP error message (type 11) includes the content of the ICMP echo message (type 8) from the source. This is due to the special traceroute implementation in Windows, which uses ICMP only (mainly message type 8 and message type 11). It is also possible that traceroute may be implemented in another different way. For instance, we have found that some traceroute implementation allows users to select protocol among ICMP, TCP, UDP and GRE. To avoid the unnecessary complexity of your program, **you only need to handle the two scenarios in finding a match between the original datagram and the returned ICMP error message: either (1) use the source port number in the original UDP, or (2) use the sequence number in the original ICMP echo message.** You code should **automatically** find out the right case for matching datagrams in the trace file. We will not test your code with a trace file not falling in the above cases.

5 Deliverables and Marking Scheme

For your final submission of your assignment, you are required to submit your source code to brightspace. You should include a readme file to tell TA how to compile and run your code. In addition, you are required to submit 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. Make sure that you use %tar -xzvf command to double-check if you have included all the files before submitting the tar file. Note that your code will be tested over linux.csc.uvic.ca.

The marking scheme is as follows:

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

6 Plagiarism

This assignment is to be done individually. You are encouraged to discuss the design of your solution with your classmates, but each person must implement their own assignment.

The End