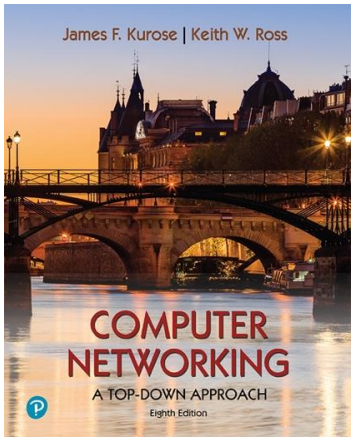
Wireshark Lab: IP v8.1

Supplement to *Computer Networking: A Top-Down Approach, 8th ed.*, J.F. Kurose and K.W. Ross

"Tell me and I forget. Show me and I remember. Involve me and I understand." Chinese proverb



In this lab, we'll investigate the celebrated IP protocol, focusing on the IPv4 and IPv6 datagram. This lab has three parts. In the first part, we'll analyze packets in a trace of IPv4 datagrams sent and received by the `traceroute` program (the `traceroute` program itself is explored in more detail in the Wireshark ICMP lab). We'll study IP fragmentation in Part 2 of this lab, and take a quick look at IPv6 in Part 3 of this lab.

Before getting started, you'll probably want to review sections 1.4.3 in the text[1] and section 3.4 of RFC 2151 to update yourself on the operation of the `traceroute` program. You'll also want to read Section 4.3 in the text, and probably also have RFC 791 on hand as well, for a discussion of the IP protocol. If you answer the questions on IP fragmentation, you'll definitely also want to review material on IP fragmentation. Although we've removed the topic of IP fragmentation from the 8th edition of our textbook (to make room for new material), you can find material on IP fragmentation from the 7th edition of our textbook (and earlier) at http://gaia.cs.umass.edu/kurose_ross/Kurose_Ross_7th_edition_section_4.3.2.pdf. RFC 8200 is the full RFC for IPv6, but reading that is a bit of overkill for this lab; you can review IPv6 by consulting Section 4.3.4 in the textbook.

Capturing packets from an execution of traceroute

In order to generate a trace of IPv4 datagrams for the first two parts of this lab, we'll use the `traceroute` program to send datagrams of two different sizes to gaia.cs.umass.edu. Recall that `traceroute` 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

---

[1] References to figures and sections are for the 8th edition of our text, *Computer Networks, A Top-down Approach, 8th ed., J.F. Kurose and K.W. Ross, Addison-Wesley/Pearson, 2020.* Our website for this book is http://gaia.cs.umass.edu/kurose_ross You'll find lots of interesting open material there.

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 IP addresses of the routers between itself and the destination by looking at the source IP addresses in the datagrams containing the ICMP TTL-exceeded messages.

Let's run `traceroute` and have it send datagrams of two different sizes. The larger of the two datagram lengths will require `traceroute` messages to be fragmented across multiple IPv4 datagrams.

- **Linux/MacOS.** With the Linux/MacOS `traceroute` command, the size of the UDP datagram sent towards the final destination can be explicitly set by indicating the number of bytes in the datagram; this value is entered in the `traceroute` command line immediately after the name or address of the destination. For example, to send `traceroute` datagrams of 2000 bytes towards gaia.cs.umass.edu, the command would be:

  ```
  %traceroute gaia.cs.umass.edu 2000
  ```

- **Windows.** The `tracert` program provided with Windows does not allow one to change the size of the ICMP message sent by `tracert`. So it won't be possible to use a Windows machine to generate ICMP messages that are large enough to force IP fragmentation. However, you can use `tracert` to generate small, fixed length packets to perform Part 1 of this lab. At the DOS command prompt enter:

  ```
  >tracert gaia.cs.umass.edu
  ```

  If you want to do the second part of this lab, you can download a packet trace file that was captured on one of the author's computers[2].

Do the following:

- Start up Wireshark and begin packet capture. *(Capture->Start* or click on the blue shark fin button in the top left of the Wireshark window).
- Enter two `traceroute` commands, using gaia.cs.umass.edu as the destination, the first with a length of 56 bytes. Once that command has finished executing, enter a second `traceroute` command for the same destination, but with a length of 3000 bytes.
- Stop Wireshark tracing.

---

[2] You can download the zip file http://gaia.cs.umass.edu/wireshark-labs/wireshark-traces-8.1.zip and extract the trace file *ip-wireshark-trace1-1.pcapng*. This trace file can be used to answer these Wireshark lab questions without actually capturing packets on your own. The trace was made using Wireshark running on one of the author's computers, while performing the steps in this Wireshark lab. Once you've downloaded a trace file, you can load it into Wireshark and view the trace using the *File* pull down menu, choosing *Open*, and then selecting the trace file name.