Protocol Layers

Objective

1. To learn how to use Wireshark to capture and analyze network traffic.
2. To learn how protocols and layering are represented in packets. They are key concepts for structuring networks that are covered in **Chapter 1.5 and Chapter 1.6** of the textbook.

Requirements

You need to install following tools on your computer beforehand:

(1) **Wireshark:** Wireshark is a software tool to capture and examine a packet trace You can download it from www.wireshark.org if it is not already installed on your computer. We highly recommend that you watch the short 5-minute video "Introduction to Wireshark" that is on the site.

(2) **wget / curl:** *wget* (Linux and Windows) and *curl* (Mac) are command-line programs that let you fetch a URL. Unlike a web browser, which fetches and executes entire pages, *wget* and *curl* give you control over exactly which URLs you fetch and when you fetch them. Both have many options (try "*wget --help*" or "*curl --help*" to see) but a URL can be fetched simply with *"wget URL"* or "*curl URL*". In the following labs, we will use *wget* as example.

- For Linux, *wget* can be installed via your package manager.
- For Windows, *wget* is available as a binary; look for download information on http://www.gnu.org/software/wget/.
- For Mac, *curl* comes installed with the OS.

Exercise

Task 1: Capture a Trace

Proceed as follows to capture a trace of network traffic:

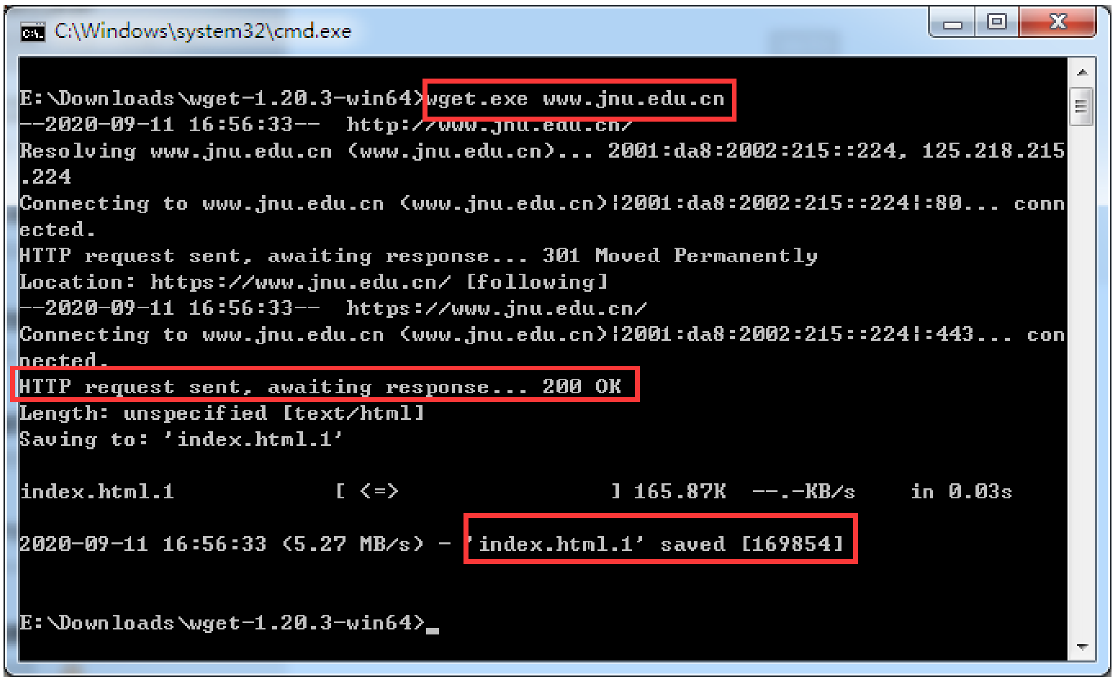
1. Close unnecessary browser tabs and windows on your computer. By minimizing browser activity, you will stop your computer from fetching unnecessary web content, and avoid incidental traffic in the trace.
2. Launch Wireshark and start a capture with a filter of "*tcp.port==80*" and check "enable network name resolution".

   Tips:

   - This filter will record only standard web traffic and not other kinds of packets that your computer may send. You can also set other kinds of filters, e.g., based on the IP addresses of the website.
   - You need to select the interface from which to capture as the main wired or wireless interface used by your computer to connect to the Internet. If unsure, guess and revisit this step later if your capture is not successful.
   - Capture→Options:

   i. Uncheck “Enable promiscuous mode…”. This mode is useful to overhear packets sent to/from other computers on broadcast networks. We only want to record packets sent to/from your computer.
   ii. Check “Resolve network names”.
   iii. Leave other options at their default values.

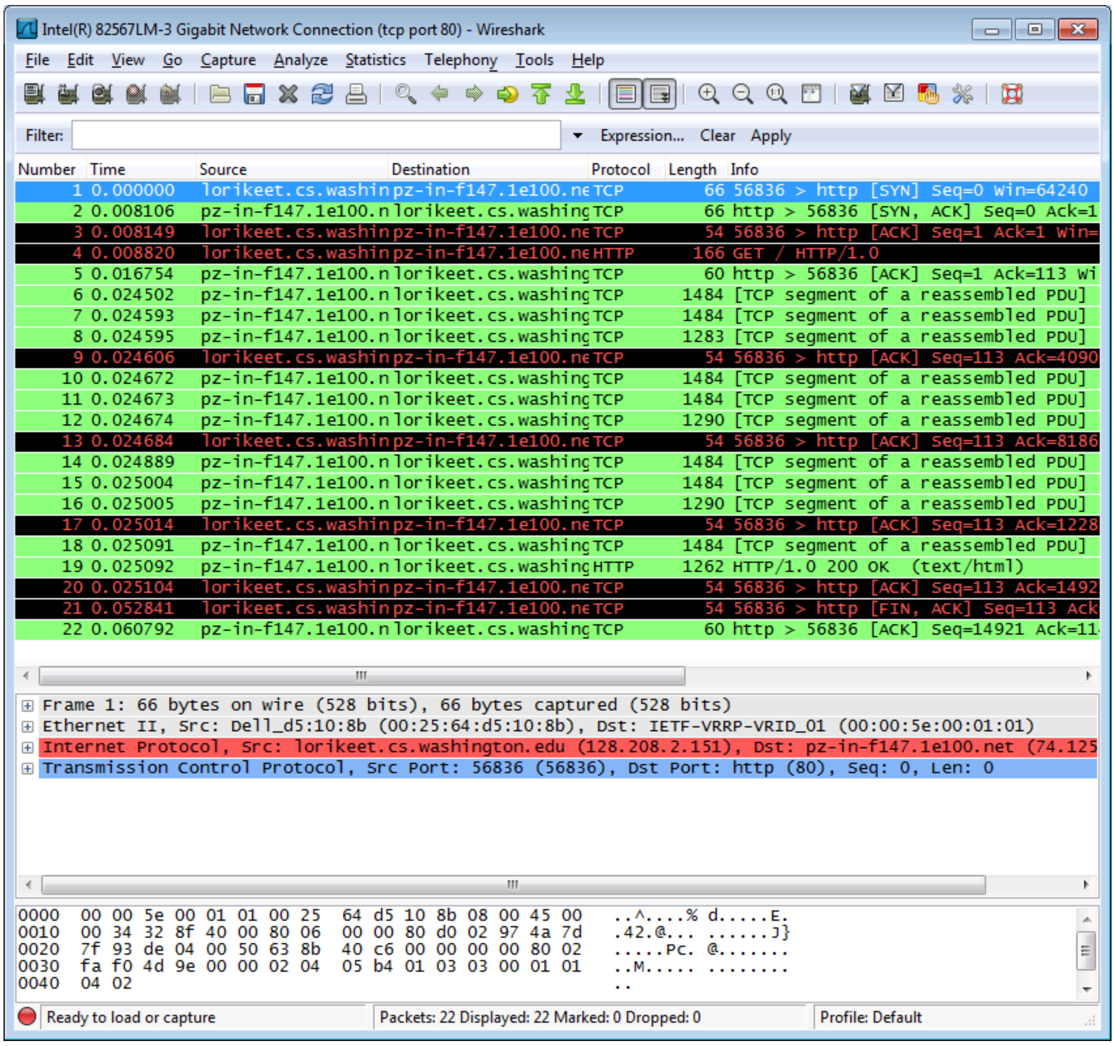
3. Pick a URL and fetch it with *wget*. For example, “wget www.jnu.edu.cn” or “curl www.jnu.edu.cn”. This will fetch the resource and either write it to a file (*wget*) or to the screen (*curl*). A successful example is shown below for *wget*.
   Tips:
   - You want a single response with status code “200 OK”.
   - If the fetch does not work, then try a different URL; if no URLs seem to work, then debug your use of *wget/curl* or your Internet connectivity.



```
E:\Downloads\wget-1.20.3-win64>wget.exe www.jnu.edu.cn
--2020-09-11 16:56:33--  http://www.jnu.edu.cn/
Resolving www.jnu.edu.cn (www.jnu.edu.cn)... 2001:da8:2002:215::224, 125.218.215
.224
Connecting to www.jnu.edu.cn (www.jnu.edu.cn)|2001:da8:2002:215::224|:80... conn
ected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://www.jnu.edu.cn/ [following]
--2020-09-11 16:56:33--  https://www.jnu.edu.cn/
Connecting to www.jnu.edu.cn (www.jnu.edu.cn)|2001:da8:2002:215::224|:443... con
nected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/html]
Saving to: 'index.html.1'

index.html.1            [ <=>                ] 165.87K  --.-KB/s    in 0.03s

2020-09-11 16:56:33 (5.27 MB/s) - 'index.html.1' saved [169854]


E:\Downloads\wget-1.20.3-win64>_
```

Figure 2: Using *wget* to fetch a URL

4. After the fetch is successful, return to Wireshark and use the menus or buttons to stop the trace. If you have succeeded, the upper Wireshark window will show multiple packets, and most likely it will be full.
   Tips:
   - How many packets are captured will depend on the size of the web page, but there should be at least 8 packets in the trace, and typically 20~100, and many of these packets will be colored green. An example is shown below.


Intel(R) 82567LM-3 Gigabit Network Connection (tcp port 80) - Wireshark
File Edit View Go Capture Analyze Statistics Telephony Tools Help
Filter:
Expression... Clear Apply

```
Number Time     Source            Destination       Protocol Length Info
     1 0.000000 lorikeet.cs.washin pz-in-f147.1e100.ne TCP    66 56836 > http [SYN] Seq=0 Win=64240
     2 0.008106 pz-in-f147.1e100.n lorikeet.cs.washing TCP    66 http > 56836 [SYN, ACK] Seq=0 Ack=1
     3 0.008149 lorikeet.cs.washin pz-in-f147.1e100.ne TCP    54 56836 > http [ACK] Seq=1 Ack=1 Win=
     4 0.008820 lorikeet.cs.washin pz-in-f147.1e100.ne HTTP  166 GET / HTTP/1.0
     5 0.016754 pz-in-f147.1e100.n lorikeet.cs.washing TCP    60 http > 56836 [ACK] Seq=1 Ack=113 Wi
     6 0.024502 pz-in-f147.1e100.n lorikeet.cs.washing TCP  1484 [TCP segment of a reassembled PDU]
     7 0.024593 pz-in-f147.1e100.n lorikeet.cs.washing TCP  1484 [TCP segment of a reassembled PDU]
     8 0.024595 pz-in-f147.1e100.n lorikeet.cs.washing TCP  1283 [TCP segment of a reassembled PDU]
     9 0.024606 lorikeet.cs.washin pz-in-f147.1e100.ne TCP    54 56836 > http [ACK] Seq=113 Ack=4090
    10 0.024672 pz-in-f147.1e100.n lorikeet.cs.washing TCP  1484 [TCP segment of a reassembled PDU]
    11 0.024673 pz-in-f147.1e100.n lorikeet.cs.washing TCP  1484 [TCP segment of a reassembled PDU]
    12 0.024674 pz-in-f147.1e100.n lorikeet.cs.washing TCP  1290 [TCP segment of a reassembled PDU]
    13 0.024684 lorikeet.cs.washin pz-in-f147.1e100.ne TCP    54 56836 > http [ACK] Seq=113 Ack=8186
    14 0.024889 pz-in-f147.1e100.n lorikeet.cs.washing TCP  1484 [TCP segment of a reassembled PDU]
    15 0.025004 pz-in-f147.1e100.n lorikeet.cs.washing TCP  1484 [TCP segment of a reassembled PDU]
    16 0.025005 pz-in-f147.1e100.n lorikeet.cs.washing TCP  1290 [TCP segment of a reassembled PDU]
    17 0.025014 lorikeet.cs.washin pz-in-f147.1e100.ne TCP    54 56836 > http [ACK] Seq=113 Ack=1228
    18 0.025091 pz-in-f147.1e100.n lorikeet.cs.washing TCP  1484 [TCP segment of a reassembled PDU]
    19 0.025092 pz-in-f147.1e100.n lorikeet.cs.washing HTTP 1262 HTTP/1.0 200 OK  (text/html)
    20 0.025104 lorikeet.cs.washin pz-in-f147.1e100.ne TCP    54 56836 > http [ACK] Seq=113 Ack=1492
    21 0.052841 lorikeet.cs.washin pz-in-f147.1e100.ne TCP    54 56836 > http [FIN, ACK] Seq=113 Ack
    22 0.060792 pz-in-f147.1e100.n lorikeet.cs.washing TCP    60 http > 56836 [ACK] Seq=14921 Ack=11
```

```
⊞ Frame 1: 66 bytes on wire (528 bits), 66 bytes captured (528 bits)
⊞ Ethernet II, Src: Dell_d5:10:8b (00:25:64:d5:10:8b), Dst: IETF-VRRP-VRID_01 (00:00:5e:00:01:01)
⊞ Internet Protocol, Src: lorikeet.cs.washington.edu (128.208.2.151), Dst: pz-in-f147.1e100.net (74.125
⊞ Transmission Control Protocol, Src Port: 56836 (56836), Dst Port: http (80), Seq: 0, Len: 0
```

```
0000  00 00 5e 00 01 01 00 25  64 d5 10 8b 08 00 45 00   ..^....% d.....E.
0010  00 34 32 8f 40 00 80 06  00 00 80 d0 02 97 4a 7d   .42.@... ......J}
0020  7f 93 de 04 00 50 63 8b  40 c6 00 00 00 00 80 02   .....Pc. @.......
0030  fa f0 4d 9e 00 00 02 04  05 b4 01 03 03 00 01 01   ..M..... ........
0040  04 02                                              ..
```

```
Ready to load or capture | Packets: 22 Displayed: 22 Marked: 0 Dropped: 0 | Profile: Default
```

Figure 3: Packet trace of *wget* traffic

Task 2: Inspect the Trace

Select a packet for which the Protocol column is "HTTP" and the Info column says it is a GET. It is the packet that carries the web (HTTP) request sent from your computer to the server.

Tips:

- You can click the column headings to sort by that value, though it should not be difficult to find an HTTP packet by inspection.

Since we are fetching a web page, we know that the protocol layers being used are as shown below. That is, HTTP is the application layer web protocol used to fetch URLs. Like many Internet applications, it runs on top of the TCP/IP transport and network layer protocols. The link and physical layer protocols depend on your network, but are typically combined in the form of Ethernet (shown) if your computer is wired, or 802.11 (not shown) if your computer is wireless.

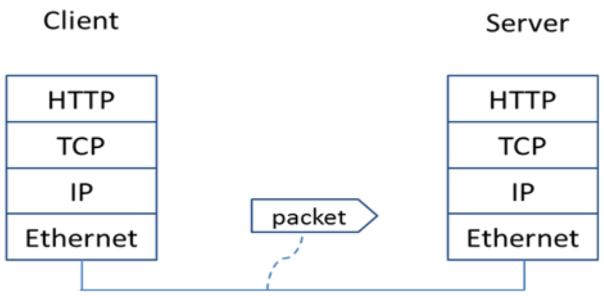


Figure 4: Protocol stack for a web fetch

With the HTTP GET packet selected, look closely to see the similarities and differences between it and our protocol stack as described next. The protocol blocks are listed in the middle panel. You can expand each block (by clicking on the "+" expander or icon) to see its details.

(1) The first Wireshark block is "Frame". This is not a protocol, it is a record that describes overall information about the packet, including when it was captured and how many bits long it is.

(2) The second block is "Ethernet". This matches our diagram!

Tips: Note that you may have taken a trace on a computer using 802.11 yet still see an Ethernet block instead of an 802.11 block. It happens because we asked Wireshark to capture traffic in Ethernet format on the capture options, so it converted the real 802.11 header into a pseudo-Ethernet header.

(3) Then come IP, TCP, and HTTP, which are just as we wanted.

Tips: Note that the order is from the bottom of the protocol stack upwards. This is because as packets are passed down the stack, the header information of the lower layer protocol is added to the front of the information from the higher layer protocol. That is, the lower layer protocols come first in the packet "on the wire".

Task 3: Packet Structure

To show your understanding of packet structure, draw a figure of an HTTP GET packet that shows the position and size in bytes of the TCP, IP and Ethernet protocol headers.

Tips:

- Your figure can simply show the overall packet as a long, thin rectangle. Leftmost elements are the first sent on the wire. On this drawing, show the range of the Ethernet header and the Ethernet payload that IP passed to Ethernet to send over the network. To show the nesting structure of protocol layers, note the range of the IP header and the IP payload. You may have questions about the fields in each protocol as you look at them.
- To work out sizes, observe that when you click on a protocol block in the middle panel (the block itself, not the "+" expander) then Wireshark will highlight the bytes it corresponds to in the packet in the lower panel and display the length at the bottom of the window. For instance, clicking on the IP version 4 header of a packet in our trace shows us that the length is 20

bytes. (Your trace will be different if it is IPv6, and may be different even with IPv4 depending on various options.) You may also use the overall packet size shown in the Length column or Frame detail block.

Task 4: Protocol Overhead

Estimate the download protocol overhead, or percentage of the download bytes taken up by protocol overhead.

Tips:

- To do this, consider HTTP data (headers and message) to be useful data for the network to carry, and lower layer headers (TCP, IP, and Ethernet) to be the overhead. We would like this overhead to be small, so that most bits are used to carry content that applications care about.
- To work this out, first look at only the packets in the download direction for a single web fetch. You might sort on the Destination column to find them.
- The packets should start with a short TCP packet described as a SYN ACK, which is the beginning of a connection.
- They will be followed by mostly longer packets in the middle (of roughly 1 to 1.5KB), of which the last one is an HTTP packet. This is the main portion of the download. And they will likely end with a short TCP packet that is part of ending the connection.

For each packet, you can inspect how much overhead it has in the form of Ethernet / IP / TCP headers, and how much useful HTTP data it carries in the TCP payload. You may also look at the HTTP packet in Wireshark to learn how much data is in the TCP payloads over all download packets.

Task 5: Demultiplexing Keys

When an Ethernet frame arrives at a computer, the Ethernet layer must hand the packet that it contains to the next higher layer to be processed. The act of finding the right higher layer to process received packets is called demultiplexing. We know that in our case the higher layer is IP. But how does the Ethernet protocol know this? After all, the higher-layer could have been another protocol entirely (such as ARP). We have the same issue at the IP layer, i.e., IP must be able to determine that the contents of IP message is a TCP packet so that it can hand it to the TCP protocol to process. The answer is that protocols use information in their header known as a "demultiplexing key" to determine the higher layer.

Look at the Ethernet and IP headers of a download packet in detail to answer the following questions:

1. Which Ethernet header field is the demultiplexing key that tells it the next higher layer is IP? What value is used in this field to indicate "IP"?
2. Which IP header field is the demultiplexing key that tells it the next higher layer is TCP? What value is used in this field to indicate "TCP"?

Task 6: Explore on your own

We encourage you to explore protocols and layering once you have completed this lab. Some ideas:

1. Look at a short TCP packet that carries no higher-layer data. To what entity is this packet destined? After all, if it carries no higher-layer data, then it does not seem very useful to a higher layer protocol such as HTTP!
2. In the classic layered model described above, lower layers append headers to the messages passed down from higher layers. How will this model change if a lower layer adds encryption?
3. In the classic layered model described above, lower layers append headers to the messages passed down from higher layers. How will this model change if a lower layer adds compression?