Lab 1 – An Introduction to Wireshark

# Objective

To introduce Wireshark and to show how protocols and layering are represented in packets.

# Requirements

**Wireshark**: This lab uses the Wireshark software tool to capture and examine a packet trace. A packet trace is a record of traffic at a location on the network, as if a snapshot was taken of all the bits that passed across a particular wire. The packet trace records a timestamp for each packet, along with the bits that make up the packet, from the lower-layer headers to the higher-layer contents.

#### Wireshark runs on most operating systems, including Windows (we are going to focus on Windows as this is the OS available in the school’s labs), Mac and Linux. It provides a graphical UI that shows the sequence of packets and the meaning of the bits when interpreted as protocol headers and data. It color-codes packets by their type, and has various ways to filter and analyze packets to let you investigate the behavior of network protocols. Wireshark is widely used to troubleshoot networks. You can download it from [www.wireshark.org](http://www.wireshark.org) if it is not already installed on your computer (ver. 1.12.7 should be installed). Wireshark.org recommends **the free “Introduction to Wireshark” Tutorial series with Chris (click** [here](https://www.youtube.com/channel/UCHN1aYRP473xX6Z13H_mxMQ) **for the channel) and the** [SharkFest](https://www.youtube.com/channel/UCHBY7sUVdWK4bOSe7khG0UA) channel on youtube.

# Step 1: Capture a Trace (this step can be skipped – go to step 2 if you intend to use the provided PCAP trace file)

*Proceed as follows to capture a trace of network traffic; alternatively, you may use a supplied trace.* We want this trace to look at the protocol structure of packets. A simple Web fetch of a URL from a server of your choice to your computer, which is the client, will serve as traffic.

1. *Pick a URL and fetch it with* curl*.* For example, “curl <http://www.google.com>” in CMD. This will fetch the resource and you will be checking to see that the fetch works and retrieves some content (if the fetch does not work then try a different URL; if no URLs seem to work then debug your use of curl or your Internet connectivity). Alternatively, you may wish to start a new capture on Wireshark, however, a live capture can include a huge stream of data packets.
2. *Close unnecessary browser tabs and windows*. By minimizing browser activity, you will stop your computer from fetching unnecessary web content, and avoid incidental traffic in the trace.
3. *Launch Wireshark and start a capture with a filter of* “tcp port 80 (as we progress, we’ll be seeing display vs capture filters).This filter will record only standard web traffic and not other kinds of packets that your computer may send. The checking will translate the addresses of the computers sending and receiving packets into names, which should help you to recognize whether the packets are going to or from your computer. **Depending on Wireshark versions**, your capture window may be similar to the one pictured below, other than our highlighting. 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. Uncheck “capture packets in 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. Leave other options at their default values. The capture filter, if present, is used to prevent the capture of other traffic your computer may send or receive.

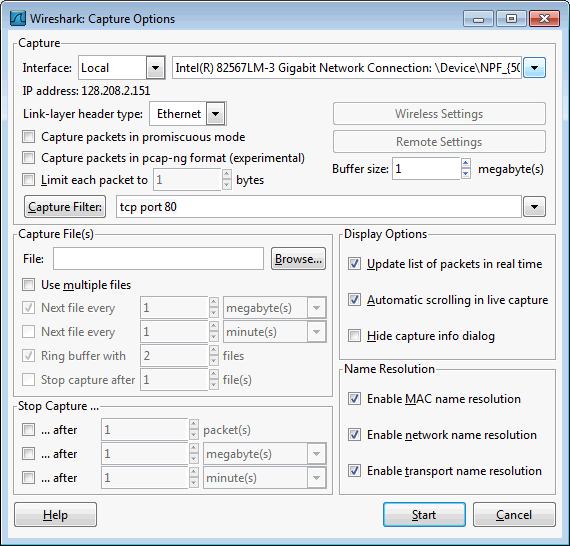


Figure 1: Setting up the capture options (depending on version)

1. *When the capture is started, repeat the web fetch using* curl*.* This time, the packets will be recorded by Wireshark as the content is transferred.
2. *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. 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. Congratulations, you have captured a trace!

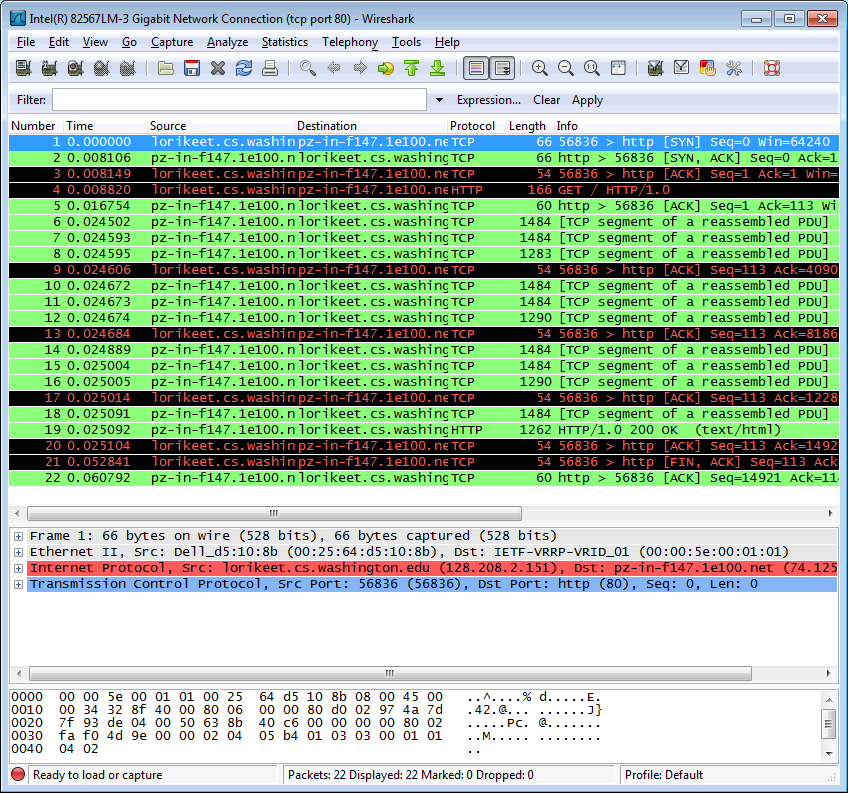


Figure 2: Packet trace of traffic

# Step 2: Inspect the Trace

Wireshark will let us select a packet (from the top panel) and view its protocol layers, in terms of both header fields (in the middle panel) and the bytes that make up the packet (in the bottom panel). In the figure above, the first packet is selected (shown in blue). Note that we are using “packet” as a general term here. Strictly speaking, a unit of information at the link layer is called a frame. At the network layer it is called a packet, at the transport layer a segment, and at the application layer a message. Wireshark is gathering frames and presenting us with the higher-layer packet, segment, and message structures it can recognize that are carried within the frames. We will often use “packet” for convenience, as each frame contains one packet and it is often the packet or higher-layer details that are of interest.

***Let us refer to the supplied 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. (You can click the column headings to sort by that value, though it should not be difficult to find an HTTP packet by inspection.) Let’s have a closer look to see how the packet structure reflects the protocols that are in use.

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.

HTTP

TCP

IP

Ethernet

Client

Server

HTTP

TCP

IP

Ethernet

packet

Figure 3: Remember: 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.

* 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.
* The second block is “Ethernet”. This matches our diagram! Then come IP, TCP, and HTTP, which are just as we wanted.

*Now find another HTTP packet, the response from the server to your computer, and look at the structure of this packet for the differences compared to the HTTP GET packet.* This packet should have “200 OK” in the Info field, denoting a successful fetch. In our trace, there are two extra blocks in the detail panel as seen in the next figure.

* The first extra block says “[11 reassembled TCP segments …]”. Details in your capture will vary, but this block is describing more than the packet itself. Most likely, the web response was sent across the network as a series of packets that were put together after they arrived at the computer. The packet labeled HTTP is the last packet in the web response, and the block lists packets that are joined together to obtain the complete web response. Each of these packets is shown as having protocol TCP even though the packets carry part of an HTTP response. Only the final packet is shown as having protocol HTTP when the complete HTTP message may be understood, and it lists the packets that are joined together to make the HTTP response.
* The second extra block says “Line-based text data …”. Details in your capture will vary, but this block is describing the contents of the web page that was fetched. In our case it is of type text/html, though it could easily have been text/xml, image/jpeg, or many other types. As with the Frame record, this is not a true protocol. Instead, it is a description of packet contents that Wireshark is producing to help us understand the network traffic.

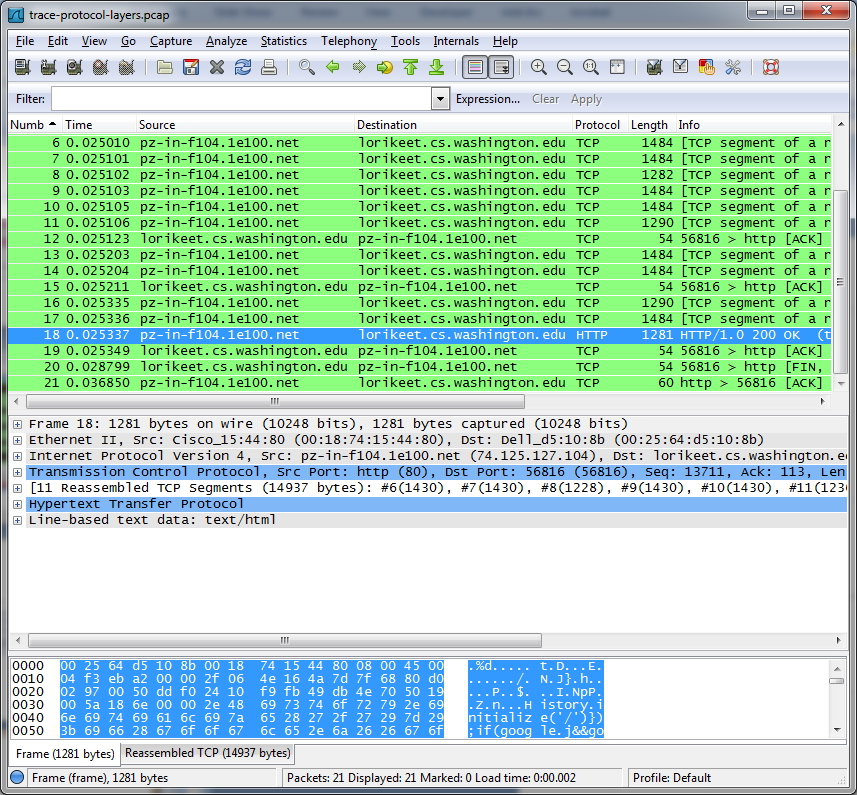


Figure 4: Inspecting a HTTP “200 OK” response

# Step 3: Protocol Overhead

*Estimate the download protocol overhead, or percentage of the download bytes taken up by protocol overhead. 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. [END]