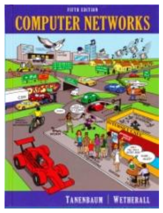
Lab Exercise – TCP



Objective

To see the details of TCP (Transmission Control Protocol). TCP is the main transport layer protocol used in the Internet.

The trace file is here: http://scisweb.ulster.ac.uk/~kevin/com320/labs/wireshark/trace-tcp.pcap

Requirements

Wireshark: This lab uses Wireshark to capture or examine a packet trace. A packet trace is a record of traffic at some 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 low-layer headers to the higher-layer contents. Wireshark runs on most operating systems, including Windows, 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. The packets are color-coded to convey their meaning, and Wireshark includes various ways to filter and analyze them to let you investigate different aspects of behavior. It is widely used to troubleshoot networks. You can download Wireshark 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.

wget / curl: This lab uses `wget` (Linux and Windows) and `curl` (Mac) to fetch web resources. `wget` and `curl` 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. Under Linux, `wget` can be installed via your package manager. Under Windows, `wget` is available as a binary at my site http://scisweb.ulster.ac.uk/~kevin/com320/labs/wget.exe or look for download information on http://www.gnu.org/software/wget/. 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`*".

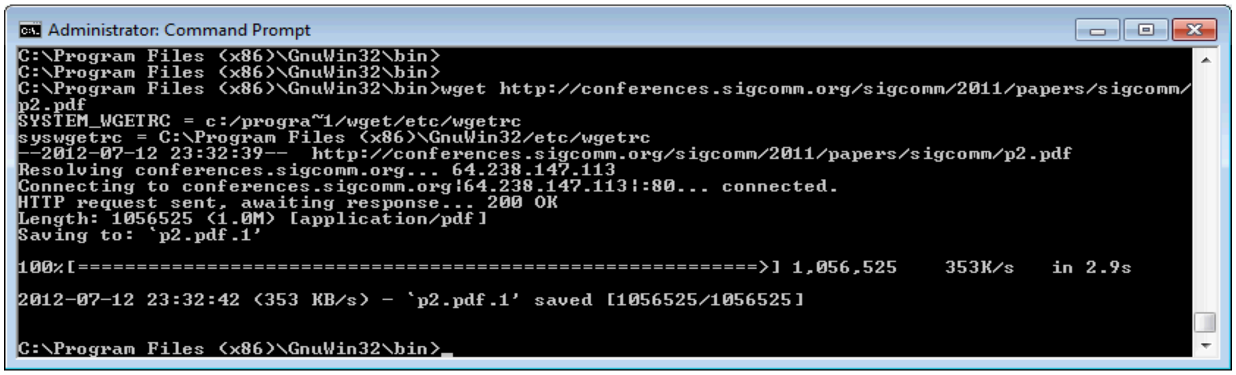
Browser: This lab uses a web browser to find or fetch pages as a workload. Any web browser will do.

Step 1: Capture a Trace

Proceed as follows to capture a trace of a single TCP connection that sends a moderate amount of data; alternatively, you may use a supplied trace. Many applications use TCP as a transport, including web browsers. So we will simply perform a web download to exercise a TCP connection. However, note that TCP is able to transfer data in both directions at the same time, but with a download content is only sent from the remote server the local computer (after the initial request).

1. *Find a URL of a single moderately-sized resource, and that you can download using HTTP (rather than HTTPS).* You may use your browser to search, perhaps looking for a picture (.jpg) or PDF document (.pdf). You want to ensure that it is a single resource and not a web page (e.g., a .html) with many inlined resources.

2. *Fetch the URL with `wget` or `curl` to check that you are able to retrieve at least 500 KB of content over at least several of network time seconds.* For example, use the command "`wget` http://scisweb.ulster.ac.uk/~kevin/com320/papers/macpaper.pdf" or go to http://conferences.sigcomm.org/sigcomm/2011/conf-program.php in your browser and pick a PDF to download. Successful examples of fetching are shown in the figure below.

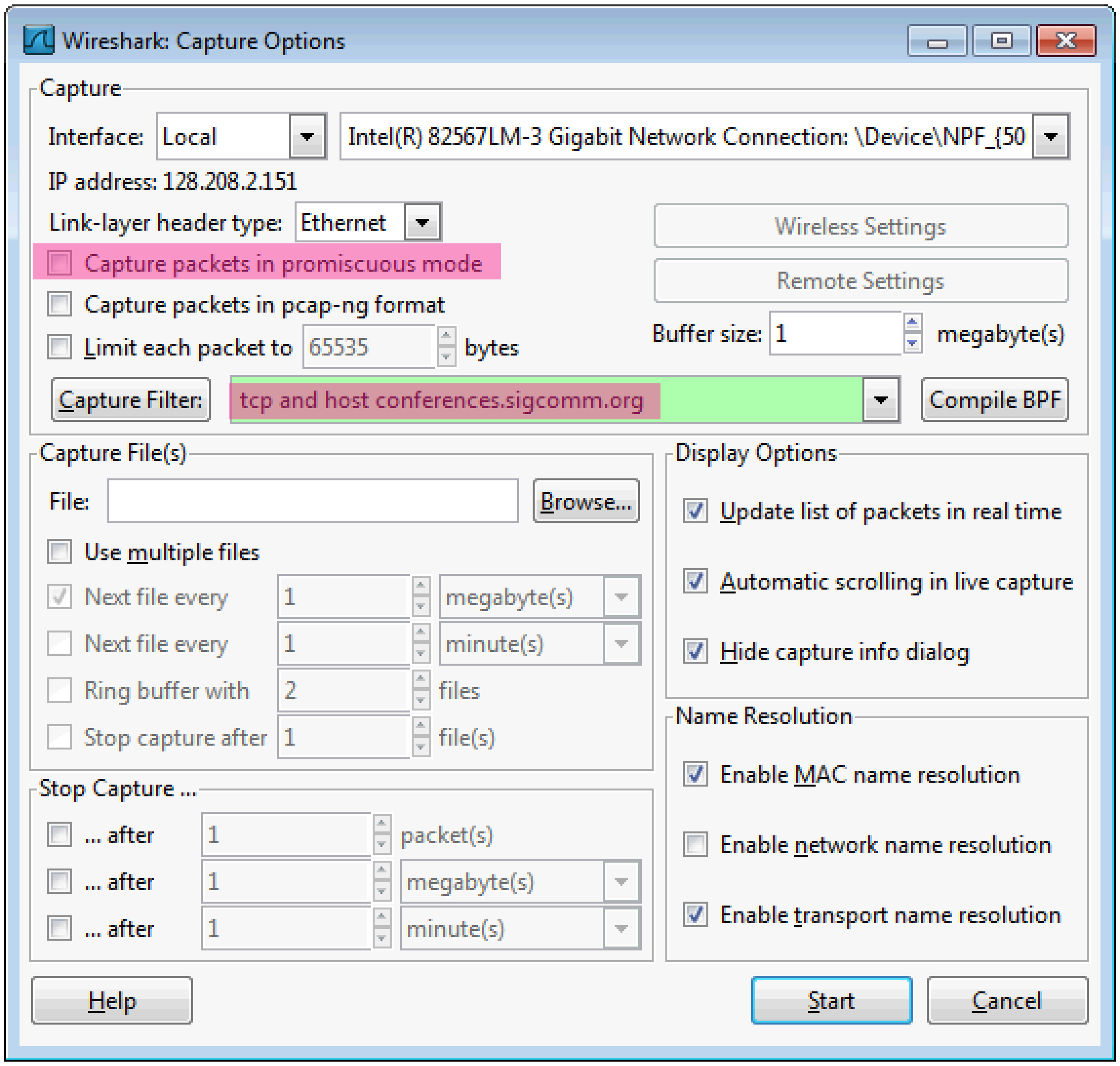


Administrator: Command Prompt

```
C:\Program Files (x86)\GnuWin32\bin>
C:\Program Files (x86)\GnuWin32\bin>
C:\Program Files (x86)\GnuWin32\bin>wget http://conferences.sigcomm.org/sigcomm/2011/papers/sigcomm/
p2.pdf
SYSTEM_WGETRC = c:/progra~1/wget/etc/wgetrc
syswgetrc = C:\Program Files (x86)\GnuWin32/etc/wgetrc
--2012-07-12 23:32:39--  http://conferences.sigcomm.org/sigcomm/2011/papers/sigcomm/p2.pdf
Resolving conferences.sigcomm.org... 64.238.147.113
Connecting to conferences.sigcomm.org|64.238.147.113|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1056525 (1.0M) [application/pdf]
Saving to: `p2.pdf.1'

100%[=====================================================>] 1,056,525    353K/s   in 2.9s

2012-07-12 23:32:42 (353 KB/s) - `p2.pdf.1' saved [1056525/1056525]


C:\Program Files (x86)\GnuWin32\bin>_
```

Figure 1: A successful fetch of a web resource with `wget` (Windows)

3. *Launch Wireshark and start a capture with a filter of* "`tcp and host xx.xx.xx`", *where* `xx.xx.xx` *is the name of the remote server from which you will fetch content, e.g.,* "`conferences.sigcomm.org`" *in the figure showing our example below.* The idea of the filter is to only capture TCP traffic between your computer and the server. Your capture window should 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. On Wireshark 1.8, the capture filter box is present directly on the options screen, but on Wireshark 1.9, you set a capture filter by double-clicking on the interface.



Capture

Interface: Local | Intel(R) 82567LM-3 Gigabit Network Connection: \Device\NPF_{50

IP address: 128.208.2.151

Link-layer header type: Ethernet

☐ Capture packets in promiscuous mode

☐ Capture packets in pcap-ng format

☐ Limit each packet to 65535 bytes

Buffer size: 1 megabyte(s)

Capture Filter: tcp and host conferences.sigcomm.org

Capture File(s)

File:

☐ Use multiple files

✓ Next file every 1 megabyte(s)

☐ Next file every 1 minute(s)

☐ Ring buffer with 2 files

☐ Stop capture after 1 file(s)

Stop Capture ...

☐ ... after 1 packet(s)

☐ ... after 1 megabyte(s)

☐ ... after 1 minute(s)

Display Options

✓ Update list of packets in real time

✓ Automatic scrolling in live capture

✓ Hide capture info dialog

Name Resolution

✓ Enable MAC name resolution

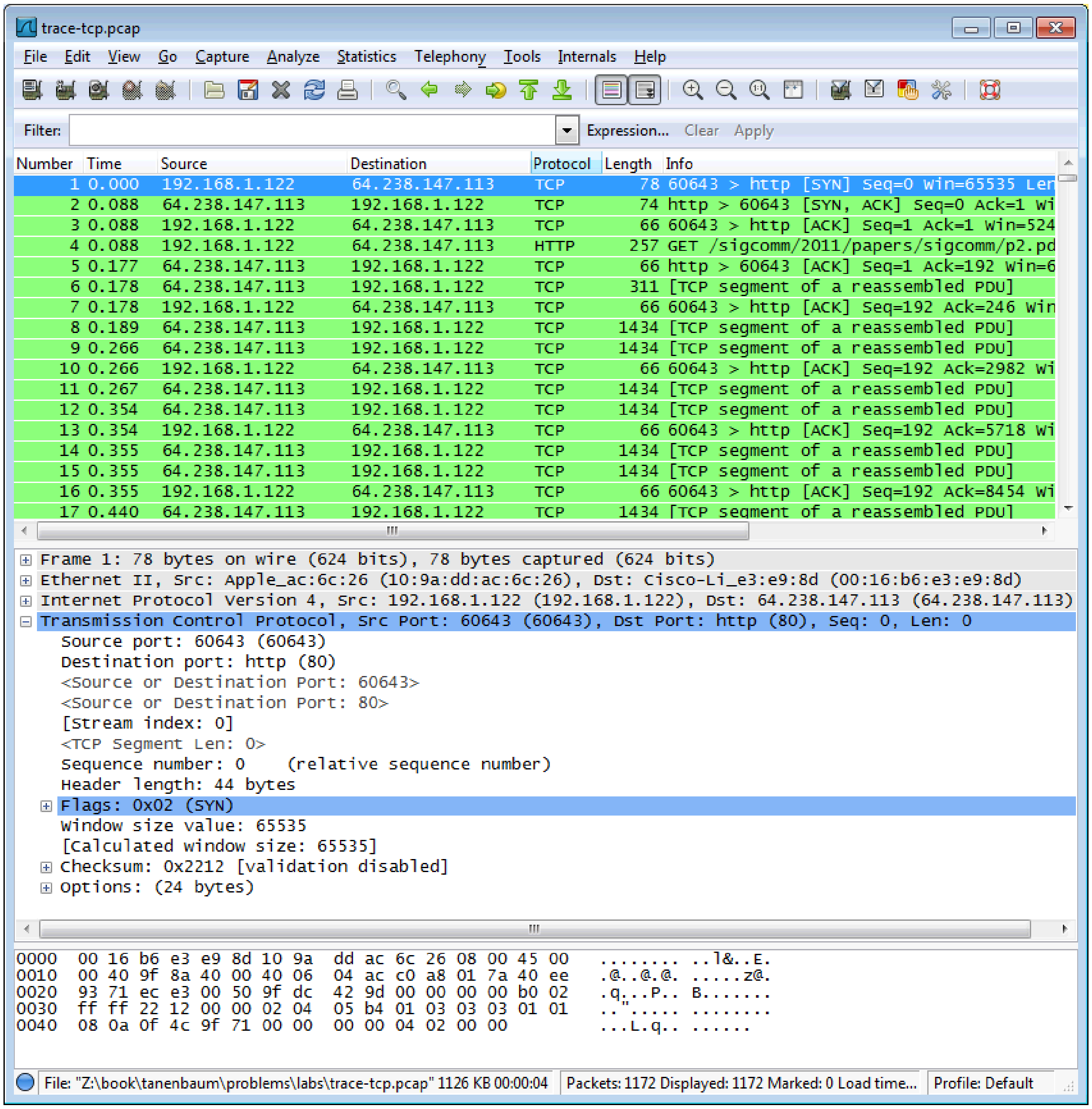
☐ Enable network name resolution

✓ Enable transport name resolution

Figure 2: Setting up the capture options

4. *After the capture is started, repeat the* `wget/curl` *command above.* This time, the packets will also be recorded by Wireshark.

5. *When the command is complete, return to Wireshark and use the menus or buttons to stop the trace.* You should now have a trace similar to that shown in the figure below. We have expanded the detail of the TCP header in our view, since it is our focus for this lab.


trace-tcp.pcap
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression... Clear Apply

```
Number Time   Source          Destination     Protocol Length Info
     1 0.000  192.168.1.122   64.238.147.113  TCP        78 60643 > http [SYN] Seq=0 Win=65535 Len
     2 0.088  64.238.147.113  192.168.1.122   TCP        74 http > 60643 [SYN, ACK] Seq=0 Ack=1 Wi
     3 0.088  192.168.1.122   64.238.147.113  TCP        66 60643 > http [ACK] Seq=1 Ack=1 Win=524
     4 0.088  192.168.1.122   64.238.147.113  HTTP      257 GET /sigcomm/2011/papers/sigcomm/p2.pd
     5 0.177  64.238.147.113  192.168.1.122   TCP        66 http > 60643 [ACK] Seq=1 Ack=192 Win=6
     6 0.178  64.238.147.113  192.168.1.122   TCP       311 [TCP segment of a reassembled PDU]
     7 0.178  192.168.1.122   64.238.147.113  TCP        66 60643 > http [ACK] Seq=192 Ack=246 Win
     8 0.189  64.238.147.113  192.168.1.122   TCP      1434 [TCP segment of a reassembled PDU]
     9 0.266  64.238.147.113  192.168.1.122   TCP      1434 [TCP segment of a reassembled PDU]
    10 0.266  192.168.1.122   64.238.147.113  TCP        66 60643 > http [ACK] Seq=192 Ack=2982 Wi
    11 0.267  64.238.147.113  192.168.1.122   TCP      1434 [TCP segment of a reassembled PDU]
    12 0.354  64.238.147.113  192.168.1.122   TCP      1434 [TCP segment of a reassembled PDU]
    13 0.354  192.168.1.122   64.238.147.113  TCP        66 60643 > http [ACK] Seq=192 Ack=5718 Wi
    14 0.355  64.238.147.113  192.168.1.122   TCP      1434 [TCP segment of a reassembled PDU]
    15 0.355  64.238.147.113  192.168.1.122   TCP      1434 [TCP segment of a reassembled PDU]
    16 0.355  192.168.1.122   64.238.147.113  TCP        66 60643 > http [ACK] Seq=192 Ack=8454 Wi
    17 0.440  64.238.147.113  192.168.1.122   TCP      1434 [TCP segment of a reassembled PDU]
```

```
⊞ Frame 1: 78 bytes on wire (624 bits), 78 bytes captured (624 bits)
⊞ Ethernet II, Src: Apple_ac:6c:26 (10:9a:dd:ac:6c:26), Dst: Cisco-Li_e3:e9:8d (00:16:b6:e3:e9:8d)
⊞ Internet Protocol Version 4, Src: 192.168.1.122 (192.168.1.122), Dst: 64.238.147.113 (64.238.147.113)
⊟ Transmission Control Protocol, Src Port: 60643 (60643), Dst Port: http (80), Seq: 0, Len: 0
    Source port: 60643 (60643)
    Destination port: http (80)
    <Source or Destination Port: 60643>
    <Source or Destination Port: 80>
    [Stream index: 0]
    <TCP Segment Len: 0>
    Sequence number: 0    (relative sequence number)
    Header length: 44 bytes
  ⊞ Flags: 0x02 (SYN)
    Window size value: 65535
    [Calculated window size: 65535]
  ⊞ Checksum: 0x2212 [validation disabled]
  ⊞ Options: (24 bytes)
```

```
0000  00 16 b6 e3 e9 8d 10 9a  dd ac 6c 26 08 00 45 00   ........ ..l&..E.
0010  00 40 9f 8a 40 00 40 06  04 ac c0 a8 01 7a 40 ee   .@..@.@. .....z@.
0020  93 71 ec e3 00 50 9f dc  42 9d 00 00 00 00 b0 02   .q...P.. B.......
0030  ff ff 22 12 00 00 02 04  05 b4 01 03 03 03 01 01   .."..... ........
0040  08 0a 0f 4c 9f 71 00 00  00 00 04 02 00 00         ...L.q.. ......
```

File: "Z:\book\tanenbaum\problems\labs\trace-tcp.pcap" 1126 KB 00:00:04 | Packets: 1172 Displayed: 1172 Marked: 0 Load time... | Profile: Default

Figure 3: Trace of TCP traffic showing the details of the TCP header

Step 2: Inspect the Trace

Select a long packet anywhere in the middle of your trace whose protocol is listed as TCP. Expand the TCP protocol section in the middle panel (by using the "+" expander or icon). All packets except the initial HTTP GET and last packet of the HTTP response should be listed as TCP. Picking a long packet ensures that we are looking at a download packet from the server to your computer. Looking at the protocol layers, you should see an IP block before the TCP block. This is because the TCP segment is carried in an IP. We have shown the TCP block expanded in our figure.

You will see roughly the following fields:

- First comes the source port, then the destination port. This is the addressing that TCP adds beyond the IP address. The source port is likely to be 80 since the packet was sent by a web server and the standard web server port is 80.
- Then there is the sequence number field. It gives the position in the bytestream of the first payload byte.
- Next is the acknowledgement field. It tells the last received position in the reverse byte stream.
- The header length giving the length of the TCP header.
- The flags field has multiple flag bits to indicate the type of TCP segment. You can expand it and look at the possible flags.
- Next is a checksum, to detect transmission errors.
- There may be an Options field with various options. You can expand this field and explore if you would like, but we will look at the options in more detail later.
- Finally, there may be a TCP payload, carrying the bytes that are being transported.

As well as the above fields, there may be other informational lines that Wireshark provides to help you interpret the packet. We have covered only the fields that are carried across the network.

Step 3: TCP Segment Structure

To show your understanding of TCP, sketch a figure of the TCP segment you studied. It should show the position and size in bytes of the TCP header fields you can observe using Wireshark. Do not break down the Flags field, or any Options field, and if you find that some TCP fields share a byte then group them. As usual, your figure can simply show the frame as a long, thin rectangle. Try not to look at the figure of a TCP segment in your text; check it afterwards to note and investigate any differences.

To work out sizes, observe that when you click on a protocol block in the middle panel (the block itself, not the "+" expander) Wireshark will highlight the corresponding bytes in the packet in the lower panel, and display the length at the bottom of the window. You may also use the overall packet size shown in the Length column or Frame detail block. Note that this method will not tell you sub-byte positions.

Step 4: TCP Connection Setup/Teardown

Three-Way Handshake

To see the "three way handshake" in action, look for a TCP segment with the SYN flag on, most likely at the beginning of your trace, and the packets that follow it. The SYN flag is noted in the Info column. You can also search for packets with the SYN flag on using the filter expression "`tcp.flags.syn==1`". A "SYN packet" is the start of the three-way handshake. In this case it will be sent from your computer to the remote server. The remote server should reply with a TCP segment with the SYN and ACK flags set, or a "SYN ACK packet". On receiving this segment, your computer will ACK it, consider the connection set up, and begin sending data, which in this case will be the HTTP request. Your exchange should follow this pattern, though it is possible that it differs slightly if a packet was lost and must be retransmitted.

Draw a time sequence diagram of the three-way handshake in your trace, up to and including the first data packet (the HTTP GET request) sent by your computer when the connection is established Put your computer on the left side and the remote server on the right side. As usual, time runs down the page, and lines across the page indicate segments. The result will be similar to diagrams such as Fig. 6-37.

Include the following features on your diagram:

- *The Sequence and ACK number, if present, on each segment.* The ACK number is only carried if the segment has the ACK flag set.
- *The time in milliseconds, starting at zero, each segment was sent or received at your computer.*
- *The round-trip time to the server estimated as the difference between the SYN and SYN-ACK segments.*

Step 5: Connection Options

As well as setting up a connection, the TCP SYN packets negotiate parameters between the two ends using Options. Each end describes its capabilities, if any, to the other end by including the appropriate Options on its SYN. Often both ends must support the behavior for it to be used during data transfer.

Answer the following question:

1. *What TCP Options are carried on the SYN packets for your trace?*

Step 6: FIN/RST Teardown

Finally, the TCP connection is taken down after the download is complete. This is typically done with FIN (Finalize) segments. Each side sends a FIN to the other and acknowledges the FIN they receive; it is similar to the three-way handshake. Alternatively, the connection may be torn down abruptly when one end sends a RST (Reset). This packet does not need to be acknowledged by the other side.

Draw a picture of the teardown in your trace, starting from when the first FIN or RST is issued until the connection is complete. As before, show the sequence and ACK numbers on each segment. If you have FINs then use the time difference to estimate the round-trip time.

Step 7: TCP Data Transfer

The middle portion of the TCP connection is the data transfer, or download, in our trace. This is the main event. To get an overall sense of it, we will first look at the download rate over time.

Under the Statistics menu select an "IO Graph". By default, this graph shows the rate of packets over time. Tweak it to show the download rate with the changes given below. You might be tempted to use the "TCP Stream Graph" tools under the Statistics menu instead. However, these tools are not useful for our case because they assume the trace is taken near the computer sending the data; our trace is taken near the computer receiving the data.

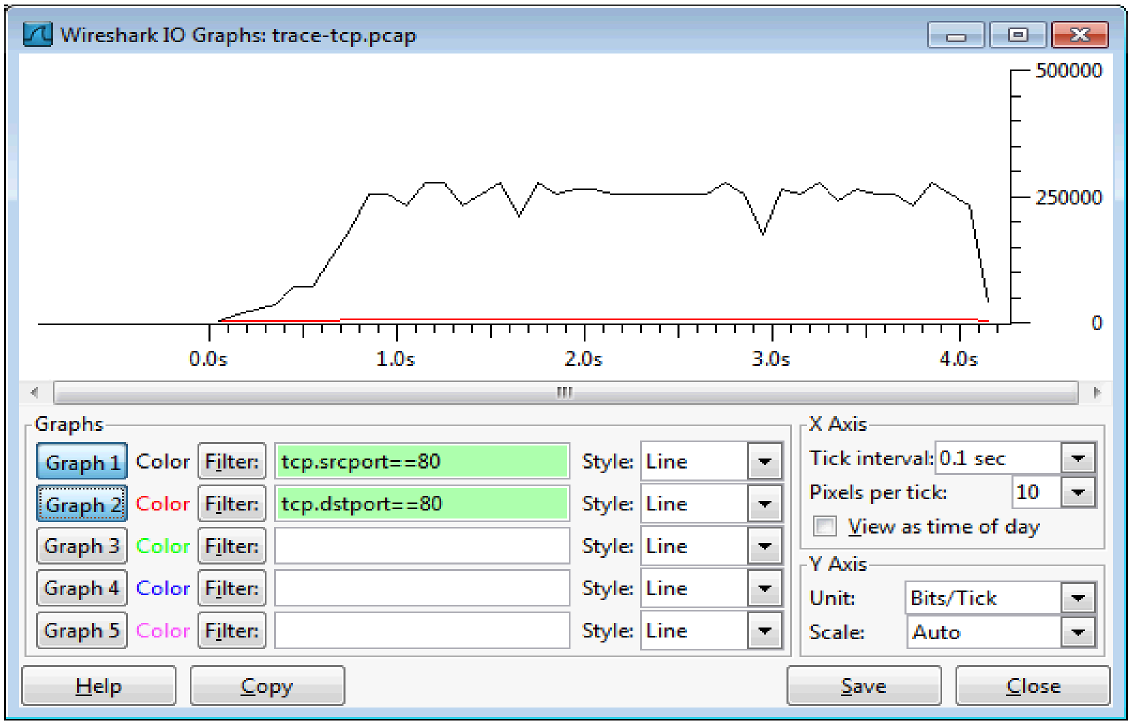
- *On the x-axis, adjust the tick interval and pixels per tick.* The tick interval should be small enough to see into the behavior over the trace, and not so small that there is no averaging. 0.1 seconds is a good choice for a several second trace. The pixels per tick can be adjusted to make the graph wider or narrower to fill the window.

- *On the y-axis, change the unit to be Bits/Tick. The default is Packet/Tick*. By changing it, we can easily work out the bits/sec throughput by taking the y-axis value and scaling as appropriate, e.g., 10X for ticks of 0.1 seconds.

- *Add a filter expression to see only the download packets.* So far we are looking at all of the packets. Assuming the download is from the usual web server port of 80, you can filter for it with a filter of "`tcp.srcport==80`". Don't forget to press Enter, and you may need to click the "Graph" button to cause it to redisplay.

- *To see the corresponding graph for the upload traffic, enter a second filter in the next box*. Again assuming the usual web server port, the filter is "`tcp.dstport==80`". After you press Enter and click the Graph button, you should have two lines on the graph.

Our graph for this procedure is shown in the figure below. From it we can see the sample download rate quickly increase from zero to a steady rate, with a bit of an exponential curve. This is slow-start. The download rate when the connection is running is approximately 2.5 Mbps. You can check your rate estimate with the information from `wget`/`curl`. The upload rate is a steady, small trickle of ACK traffic. Our download also proceeds fairly steadily until it is done. This is the ideal, but many downloads may display more variable behavior if, for example, the available bandwidth varies due to competing downloads, the download rate is set by the server rather than the network, or enough packets are lost to disrupt the transfer. You can click on the graph to be taken to the nearest point in the trace if there is a feature you would like to investigate.


Wireshark IO Graphs: trace-tcp.pcap

Graphs

Graph 1	Color	Filter:	tcp.srcport==80	Style:	Line
Graph 2	Color	Filter:	tcp.dstport==80	Style:	Line
Graph 3	Color	Filter:		Style:	Line
Graph 4	Color	Filter:		Style:	Line
Graph 5	Color	Filter:		Style:	Line

X Axis

Tick interval: 0.1 sec

Pixels per tick: 10

View as time of day

Y Axis

Unit: Bits/Tick

Scale: Auto

Figure 4: TCP download rate over time via an IO graph

Answer the following questions to show your understanding of the data transfer:

1. *What is the rough data rate in the download direction in packets/second and bits/second once the TCP connection is running well?*
2. *What percentage of this download rate is content? Show your calculation*. To find out, look at a typical download packet; there should be many similar, large download packets. You can see how long it is, and how many bytes of TCP payload it contains.
3. *What is the rough data rate in the upload direction in packets/second and bits/second due to the ACK packets?*

Inspect the packets in the download in the middle of your trace for these features:

- You should see a pattern of TCP segments received carrying data and ACKs sent back to the server. Typically there will be one ACK every couple of packets. These ACKs are called Delayed ACKs. By delaying for a short while, the number of ACKs is halved.
- Since this is a download, the sequence number of received segments will increase; the ACK number of subsequently transmitted segments will increase correspondingly.
- Since this is a download, the sequence number of transmitted segments will not increase (after the initial get). Thus the ACK number on received segments will not increase either.
- Each segment carries Window information to tell the other end how much space remains in the buffer. The Window must be greater than zero, or the connection will be stalled by flow control.

Answer the following question:

4. *If the most recently received TCP segment from the server has a sequence number of X, then what ACK number does the next transmitted TCP segment carry?*

As well as regular TCP segments carrying data, you may see a variety of other situations. You can sort the trace on the Info column and browse the packets of type "`[TCP xxx ...`". Depending on the download, you may see duplicate acks, out of order data, retransmissions, zero windows, window updates, and more. These segments are not generally distinguished by flags in the TCP header, like SYN or FIN segments. Instead, they are names for situations that may occur and be handled during transport.