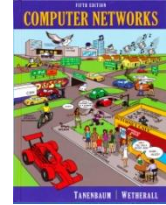


# Lab Exercise – UDP & TCP

---



## Objective

UDP (User Datagram Protocol) is an alternative communications protocol to Transmission Control Protocol (TCP) used primarily for establishing low-latency and loss tolerating connections between applications on the Internet. Both UDP and TCP run on top of the Internet Protocol (IP) and are sometimes referred to as UDP/IP or TCP/IP. Both protocols send short packets of data, called datagrams. To look at the details of UDP (User Datagram Protocol). UDP is a transport protocol used throughout the Internet as an alternative to TCP when reliability is not required. UDP provides two services not provided by the IP layer. It provides port numbers to help distinguish different user requests and, optionally, a checksum capability to verify that the data arrived intact. TCP has emerged as the dominant protocol used for the bulk of Internet connectivity owing to services for breaking large data sets into individual packets, checking for and resending lost packets and reassembling packets into the correct sequence. But these additional services come at a cost in terms of additional data overhead, and delays called latency.

In contrast, UDP just sends the packets, which means that it has much lower bandwidth overhead and latency. But packets can be lost or received out of order as a result, owing to the different paths individual packets traverse between sender and receiver. UDP is an ideal protocol for network applications in which perceived latency is critical such as gaming, voice and video communications, which can suffer some data loss without adversely affecting perceived quality. In some cases, forward error correction techniques are used to improve audio and video quality in spite of some loss. UDP can also be used in applications that require lossless data transmission when the application is configured to manage the process of retransmitting lost packets and correctly arranging received packets. This approach can help to improve the data transfer rate of large files compared with TCP. We first examine UDP.

## Step 1: Capture a UDP Trace

There are many ways to cause your computer to send and receive UDP messages since UDP is widely used as a transport protocol. The easiest options are to:

- Do nothing but wait for a while. UDP is used for many “system protocols” that typically run in the background and produce small amounts of traffic, e.g., DHCP for IP address assignment and NTP for time synchronization.
- Use your browser to visit sites. UDP is used by DNS for resolving domain names to IP addresses, so visiting fresh sites will cause DNS traffic to be sent. Be careful not to visit unsafe sites; pick recommended sites or sites you know about but have not visited recently. Simply browsing the web is likely to cause a steady stream of DNS traffic.
- Start up a voice-over-IP call with your favorite client. UDP is used by RTP, which is the protocol commonly used to carry media samples in a voice or video call over the Internet.

1. Launch Wireshark by entering *Wireshark* in the “ask my anything” search box in Windows.

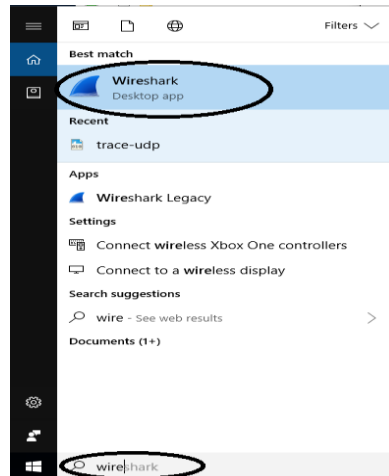


Figure 1: Starting Wireshark

2. Once Wireshark starts, select the *Ethernet interface*.

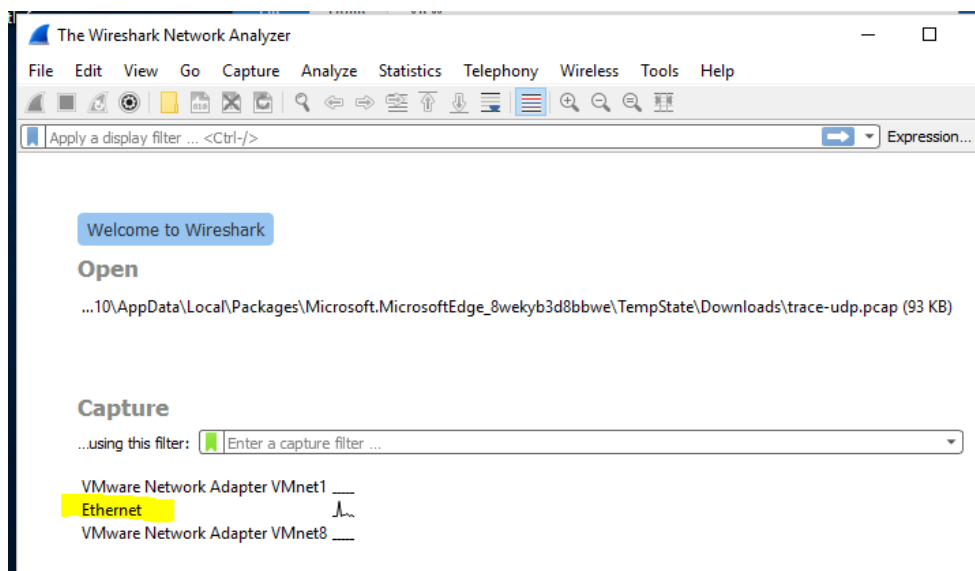


Figure 2: Selecting the Ethernet Interface

- Wireshark will automatically start capturing packets on the network.

Now, enter a filter of **udp**. (This is shown below).

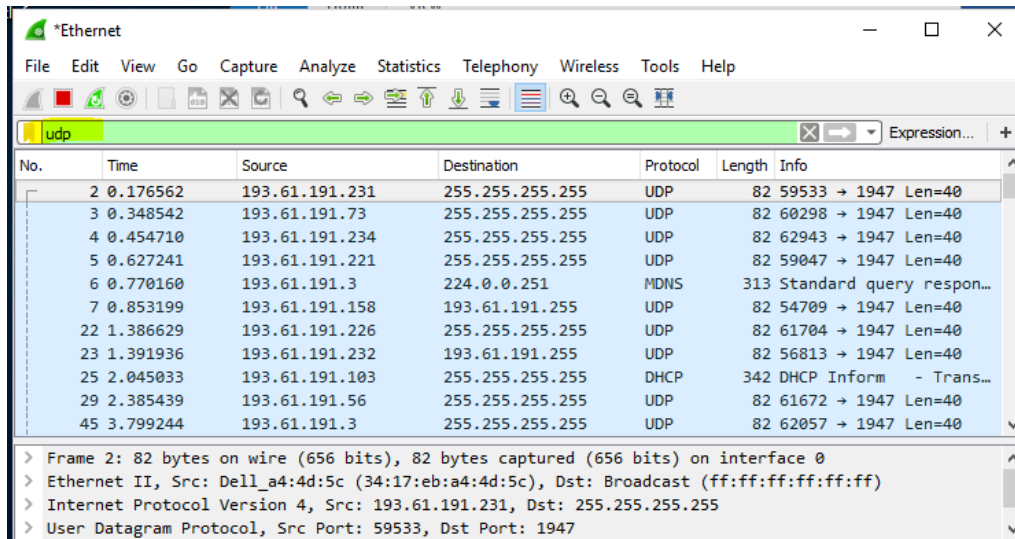


Figure 3: Setting up the capture options

- When the capture is started, it will collect UDP traffic automatically.
- Wait a little while (say 60 seconds) after you have stopped your activity to also observe any background UDP traffic. It is likely that you will observe a trickle of UDP traffic because system activity often uses UDP to communicate. We want to see some of this activity.
- Use the Wireshark menus or buttons to stop the capture.

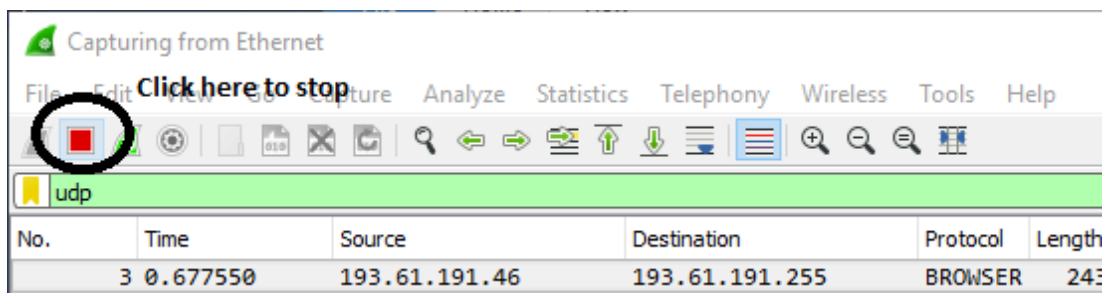


Figure 4: Stopping the capture

- You should now have a trace with many UDP packets.

## Step 2: Inspect the Trace

Different computers are likely to capture different kinds of UDP traffic depending on the network setup and local activity. Observe that the protocol column is likely to show multiple protocols, none of which is UDP. This is because the listed protocol is an application protocol layered on top of UDP. Wireshark gives the name of the application protocol, not the (UDP) transport protocol unless Wireshark cannot determine the application protocol. However, even if the packets are listed as an application protocol, they will have a UDP protocol header for us to study, following the IP and lower-layer protocol headers.

*Select different packets in the trace (in the top panel) and browse the expanded UDP header (in the middle panel). You will see that it contains the following fields:*

- Source Port, the port from which the UDP message is sent. It is given as a number and possibly a text name; names are given to port values that are registered for use with a specific application.
- Destination Port. This is the port number and possibly name to which the UDP message is destined. Ports are the only form of addressing in UDP. The computer is identified using the IP address in the lower IP layer.
- Length. The length of the UDP message.
- Checksum. A checksum over the message that is used to validate its contents. Is your checksum carrying 0 and flagged as incorrect for UDP messages sent from your computer? On some computers, the operating system software leaves the checksum blank (zero) for the NIC to compute and fill in as the packet is sent. This is called protocol offloading. It happens after Wireshark sees the packet, which causes Wireshark to believe that the checksum is wrong and flag it with a different color to signal a problem. You can remove these false errors if they are occurring by telling Wireshark not to validate the checksums. Select “Preferences” from the Wireshark menus and expand the “Protocols” area. Look under the list until you come to UDP. Uncheck “Validate checksum if possible”.

That is it. The UDP header has different values for different messages, but as you can see, it is short and sweet. The remainder of the message is the UDP payload that is normally identified the higher-layer protocol that it carries, e.g., DNS, or RTP.

### Step 3: UDP Message Structure

The figure below shows the UDP message structure as you observed. It shows the position of the IP header, UDP header, and UDP payload. Within the UDP header, it shows the position and size of each UDP field. Note how the Length field gives the length of the UDP payload plus the UDP header. The checksum is 16 bits long and the UDP header is 8 bytes long.

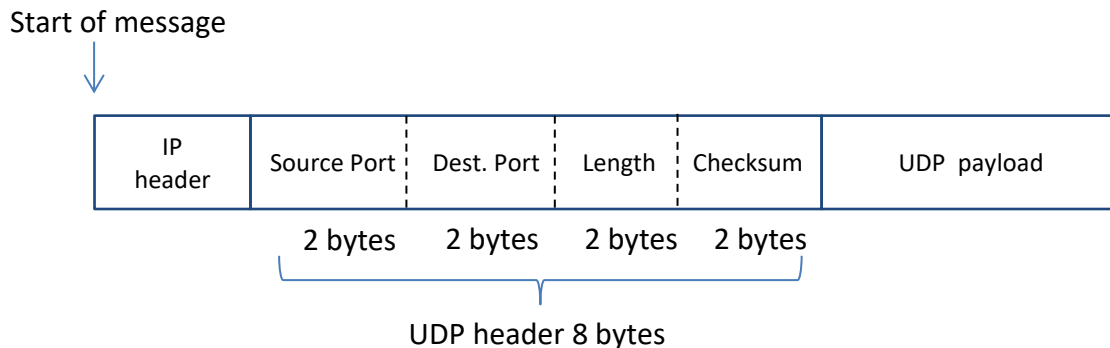


Figure 5: Structure of a UDP message

### Step 4: UDP Usage

The Protocol field in the IP header is how IP knows that the next higher protocol layer is UDP. The IP Protocol field value of 17 indicates UDP.

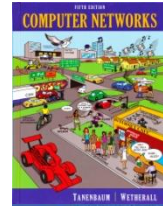
You might be surprised to find UDP messages in your trace that neither come from your computer or are sent only to your computer. You can see this by sorting on the Source and Destination columns. The source and destinations will be domain names, if Network layer name resolution is turned on, and otherwise IP addresses. *(You can toggle this setting using the View menu and selecting Name resolution.)* You can find out the IP address of your computer using the "**ipconfig**" command (Windows).

The reason you may find UDP messages without your computer's IP address as either the source or destination IP address is that UDP is widely used as part of system protocols. These protocols often send messages to all local computers who are interested in them using broadcast and multicast addresses. In our traces, we find DNS (the domain name system), MDNS (DNS traffic that uses IP multicast), NTP (for time synchronization), NBNS (NetBIOS traffic), DHCP (for IP address assignment), SSDP (a service discovery protocol), STUN (a NAT traversal protocol), RTP (for carrying audio and video samples), and more.

A variety of broadcast and multicast addresses may be found. These include the Internet broadcast address of 255.255.255.255, subnet broadcast addresses such as 192.168.255.255 and multicast IP addresses such as 224.0.xx.xx for multicast DNS.

Note also that UDP messages can be as large as roughly 64Kbytes but most often they are a few hundred bytes or less, typically around 100 bytes.

# Lab Exercise – TCP



## Objective

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

## Step 1: Open the Trace

Open the trace file here: <https://kevincurran.org/com320/labs/wireshark/trace-tcp.pcap>

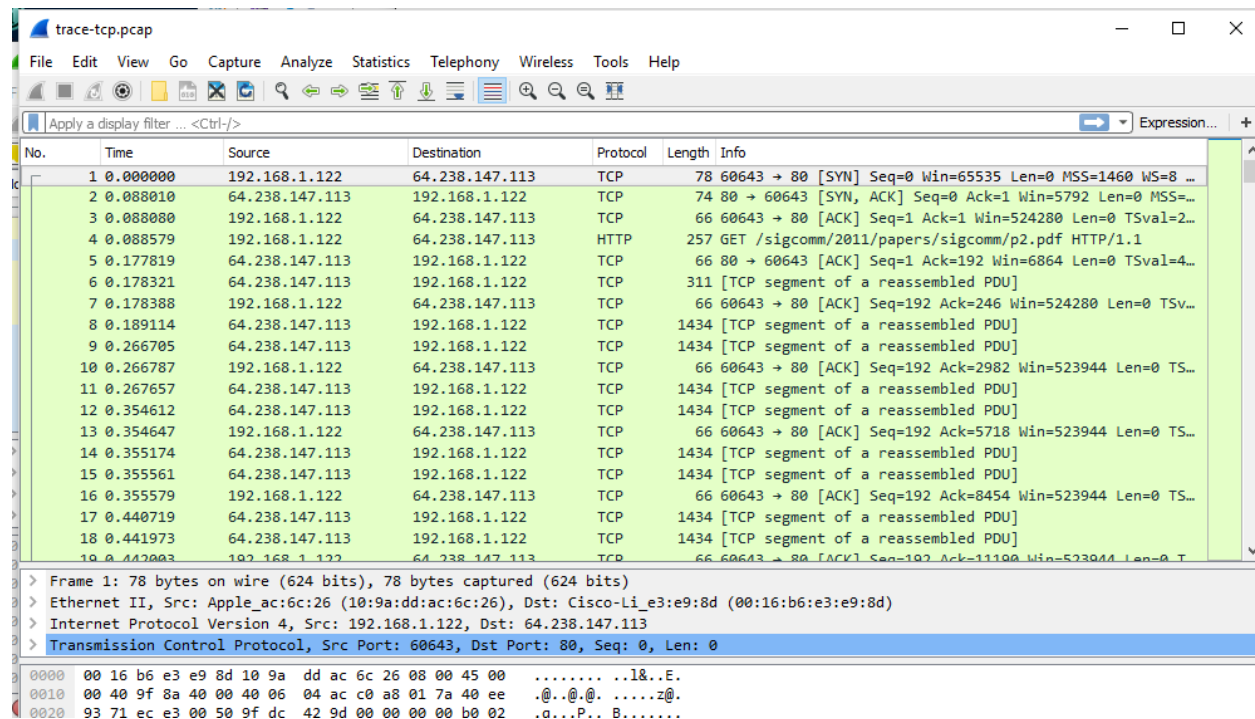


Figure 6: Selecting the Ethernet Interface

## 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 byte stream 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

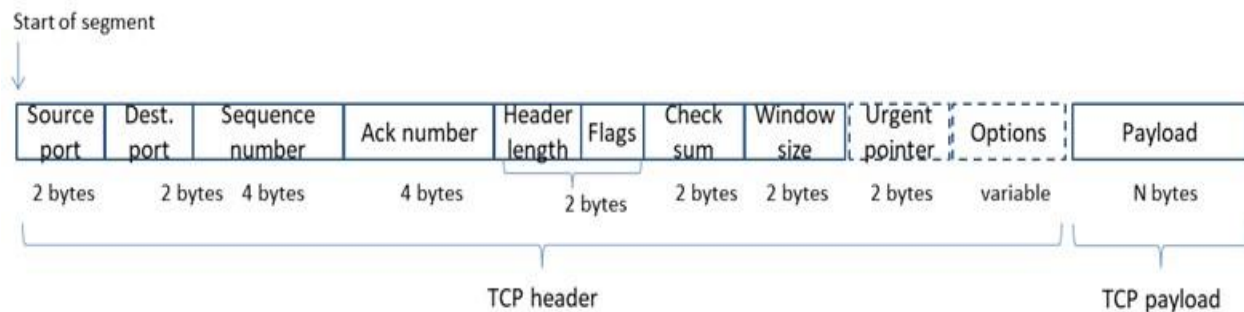


Figure 7: Structure of a TCP segment

This drawing differs from the text drawing in the book in only minor respects:

- The Header length and Flags fields are combined into a 2-byte quantity. It is not easy to determine their bit lengths with Wireshark.
- The Urgent Pointer field is shown as dotted. This field is typically not used, and so does not show up in Wireshark and we do not expect you to have it in your drawing. You can notice its existence in Wireshark, however, by observing the zero bytes in the segment that are skipped over as you select the different fields.
- The Options field is shown dotted, as it may or may not be present for the segments in your trace. Most often it will be present, and when it is then its length will be a multiple of four bytes.
- The Payload is optional. It is present for the segment you viewed, but not present on an Ack-only segment, for example.
- *Note, you can work out sizes yourself by clicking 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. See below where a TCP packet of length 66 is highlighted.*

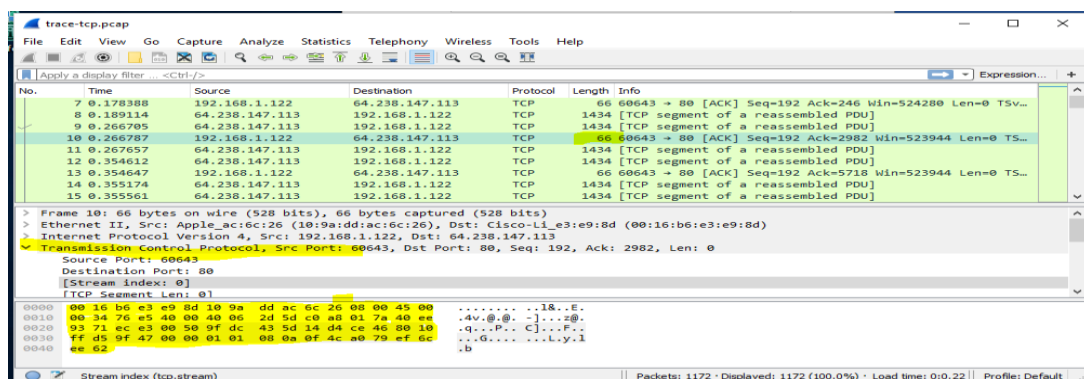


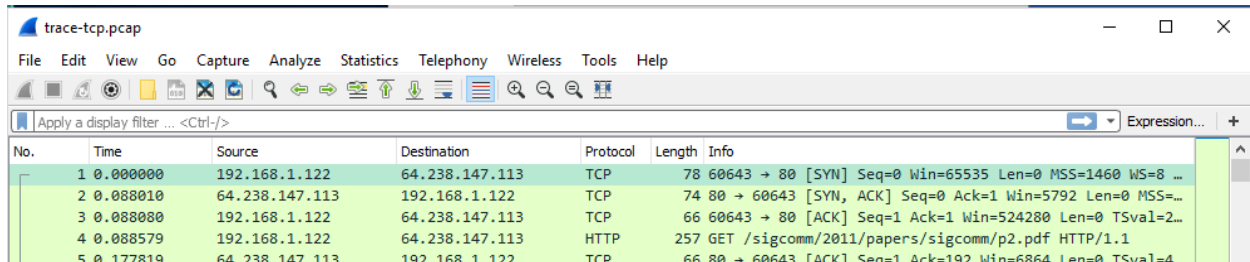
Figure 8: Examining the size of segments



## 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. These are up at the beginning of your trace, and the packets that follow it (see below).

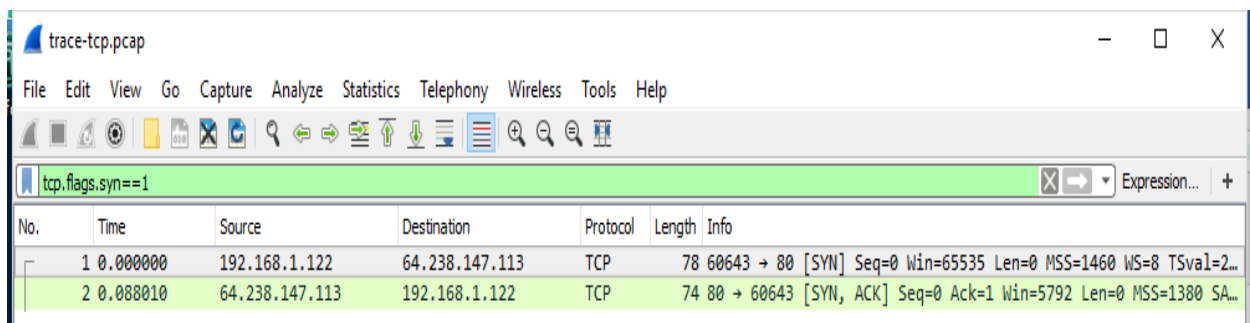


The screenshot shows the Wireshark interface with a packet capture named 'trace-tcp.pcap'. The packet list pane shows five packets. Packet 1 is a SYN packet from 192.168.1.122 to 64.238.147.113. Packet 2 is a SYN, ACK packet from 64.238.147.113 to 192.168.1.122. Packet 3 is an ACK packet from 192.168.1.122 to 64.238.147.113. Packet 4 is an HTTP GET request. Packet 5 is an ACK packet from 64.238.147.113 to 192.168.1.122.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.1.122	64.238.147.113	TCP	78	60643 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=8 ...
2	0.088010	64.238.147.113	192.168.1.122	TCP	74	80 → 60643 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=...
3	0.088080	192.168.1.122	64.238.147.113	TCP	66	60643 → 80 [ACK] Seq=1 Ack=1 Win=524280 Len=0 TSval=2...
4	0.088579	192.168.1.122	64.238.147.113	HTTP	257	GET /sigcomm/2011/papers/sigcomm/p2.pdf HTTP/1.1
5	0.177819	64.238.147.113	192.168.1.122	TCP	66	80 → 60643 [ACK] Seq=1 Ack=192 Win=6864 Len=0 TSval=4...

Figure 9: Selecting a TCP segment with SYN flag

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`”. (See below)



The screenshot shows the same Wireshark interface, but with a display filter applied: 'tcp.flags.syn==1'. This filter has filtered the packet list to show only the two SYN packets (packets 1 and 2). The packet list pane shows two packets. Packet 1 is a SYN packet from 192.168.1.122 to 64.238.147.113. Packet 2 is a SYN, ACK packet from 64.238.147.113 to 192.168.1.122.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.1.122	64.238.147.113	TCP	78	60643 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=8 TSval=2...
2	0.088010	64.238.147.113	192.168.1.122	TCP	74	80 → 60643 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1380 SA...

Figure 10: Selecting a TCP segment with SYN flag on

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.

## Step 5: TCP Connection Setup/Teardown

Next, we wish to clear the display filter `tcp.flags.syn==1` so that we can once again see all the packets in our original trace. Do this by clearing the display filter as shown below.

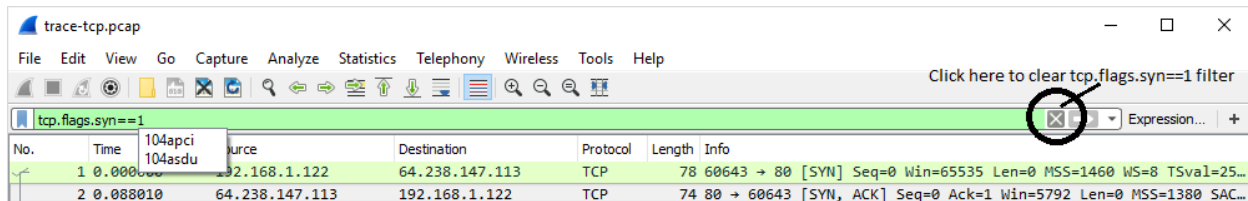


Figure 11: Clearing the display filter TCP segment with SYN flag on

If you do this correctly, you should see the full trace. We are most interested in the first three packets.

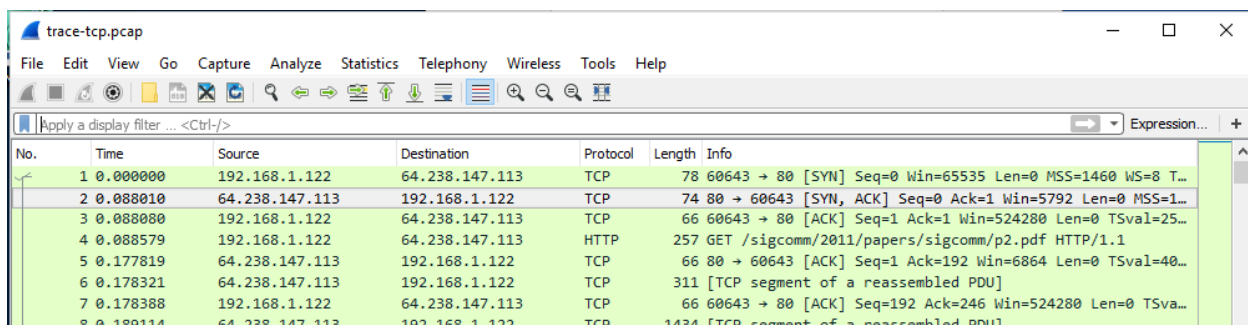


Figure 12: Viewing the complete trace

Below is 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. As usual, time runs down the page, and lines across the page indicate segments.

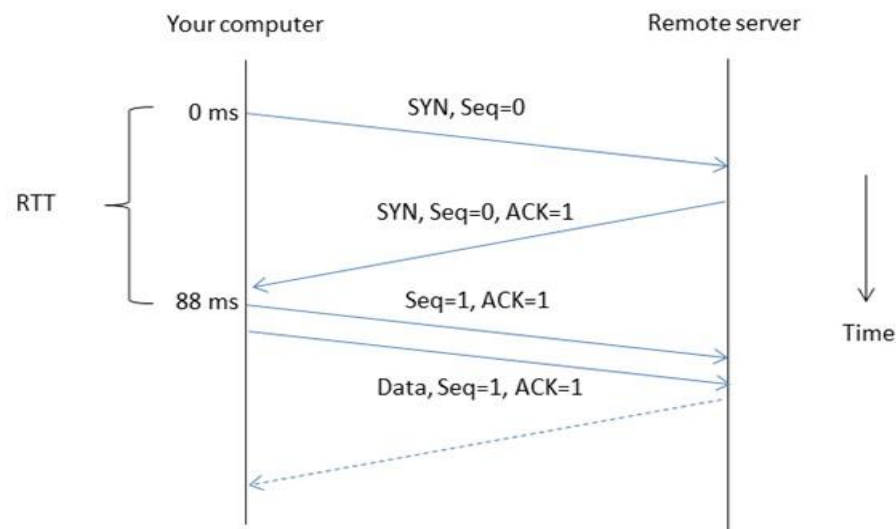


Figure 13: Time sequence diagram for the TCP three-way handshake

There are several features to note:

- The initial SYN has no ACK number, only a sequence number. All subsequent packets have ACK numbers.
- The initial sequence numbers are shown as zero in each direction. This is because our Wireshark is configured to show relative sequence numbers. The actual sequence number is some large 32-bit number, and it is different for each end.
- The ACK number is the corresponding sequence number plus 1.
- Our computer sends the third part of the handshake (the ACK) and then sends data right away in a different packet. It would be possible to combine these packets, but they are typically separate (because one is triggered by the OS and one by the application).
- For the Data segment, the sequence number and ACK stay with the previous values. The sequence number will advance as the sender sends more data. The ACK number will advance as the sender receives more data from the remote server.
- The three packets received and sent around 88ms happen very close together compared to the gap between the first and second packet. This is because they are local operations; there is no network delay involved.
- The RTT is 88ms in our trace. If you use a local web server, the RTT will be very small, likely a few milliseconds. If you use a major web server that may be provided by a content distribution network, the RTT will likely be tens of milliseconds. If you use a geographically remote server, the RTT will likely be hundreds of milliseconds.

## 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.

Common Options include Maximum Segment Size (MSS) to tell the other side the largest segment that can be received, and Timestamps to include information on segments for estimating the round trip time. There are also Options such as NOP (No-operation) and End of Option list that serve to format the Options but do not advertise capabilities. You do not need to include these formatting options in your answer above. Options can also be carried on regular segments after the connection is set up when they play a role in data transfer. This depends on the Option. For example: the MSS option is not carried on each packet because it does not convey new information; timestamps may be included on each packet to keep a fresh estimate of the RTT; and options such as SACK (Selective Acknowledgments) are used only when data is received out of order.

Our TCP Options are Maximum Segment Size, Window Scale, SACK permitted, and Timestamps. Each of these Options is used in both directions. There are also the NOP & End of Option List formatting options.

Here is an example of a FIN teardown:

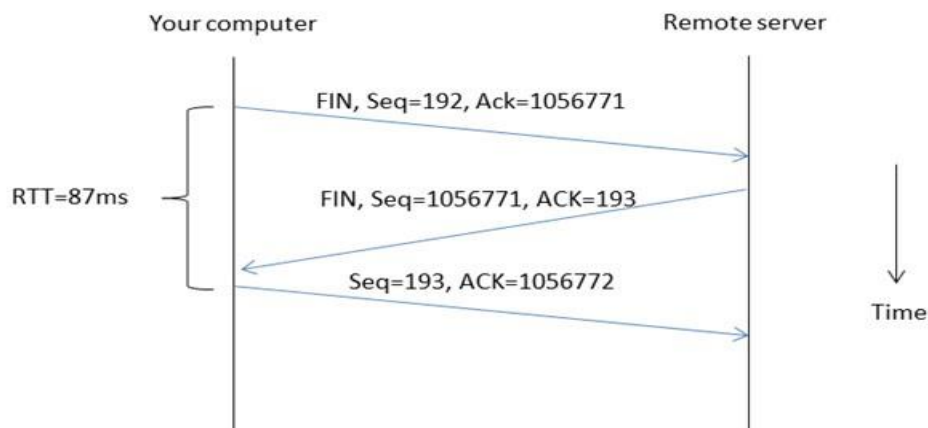


Figure 14: Time sequence diagram for FIN teardown

Points to note:

- The teardown is initiated by the computer; it might also be initiated by the server.
- Like the SYN, the FIN flag occupies one sequence number. Thus, when the sequence number of the FIN is 192, the corresponding Ack number is 193.
- Your sequence numbers will vary. Our numbers are relative (as computed by Wireshark) but clearly depend on the resource that is fetched. You can tell that it is around 1 MB long.
- The RTT in the FIN exchange is like that in the SYN exchange, as it should be. Your RTT will vary depending on the distance between the computer and server as before.

## 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.

Below is a picture of the teardown in your trace, starting from when the first FIN or RST is issued until the connection is complete. It shows the sequence and ACK numbers on each segment.

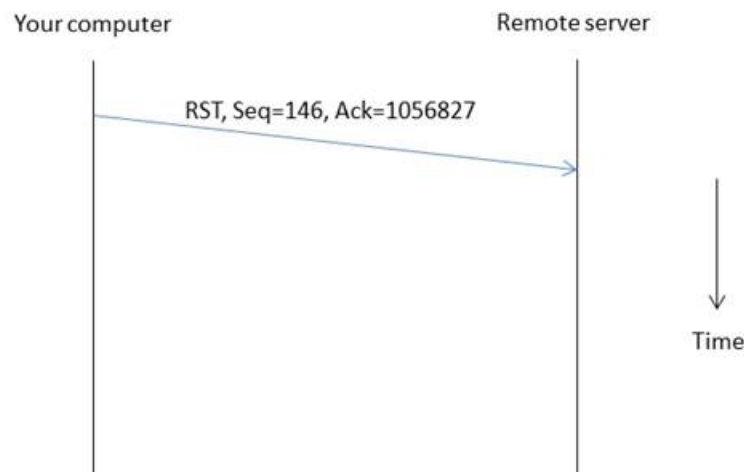


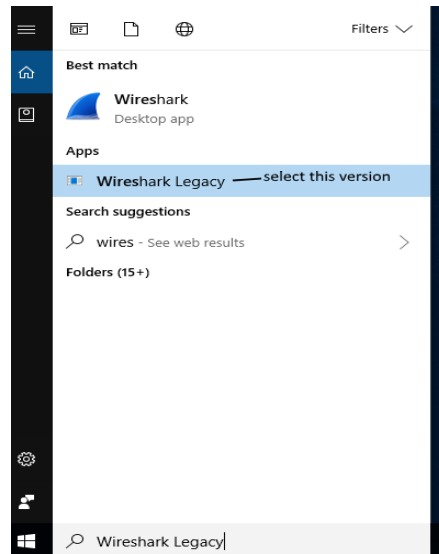
Figure 15: Time sequence diagram for RST teardown

Points to note:

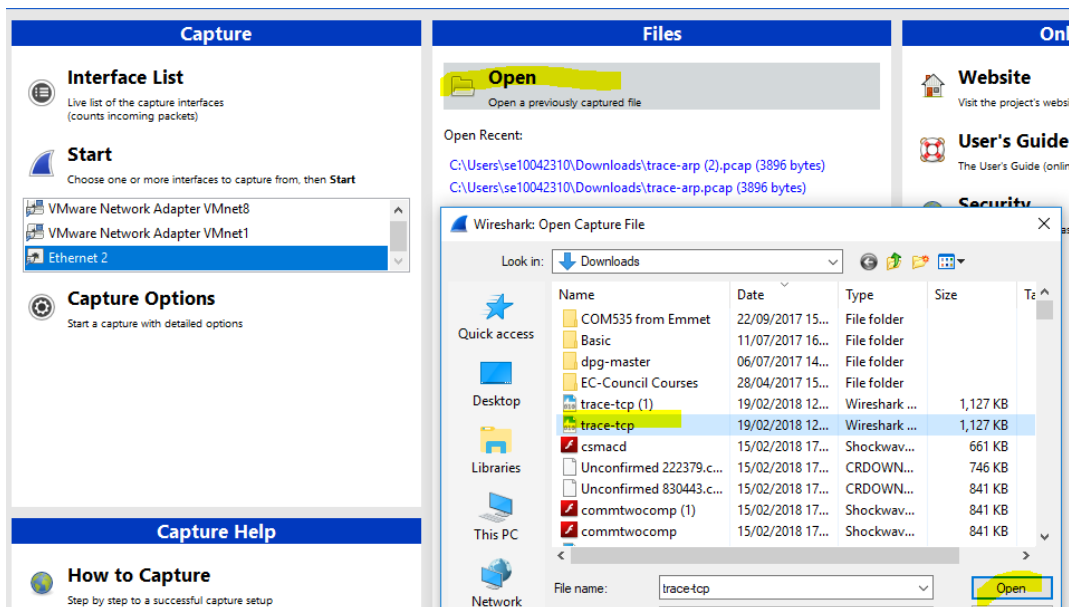
- The teardown is initiated by the computer; it might also be initiated by the server.
- The teardown is abrupt – a single RST in this case, and then it is closed, which the other end must accommodate.
- The sequence and Ack numbers do not really matter here. They are simply the (relative Wireshark) values at the end of the connection.
- Since there is no round trip exchange, no RTT can be estimated.

## Step 7: TCP Data Transfer

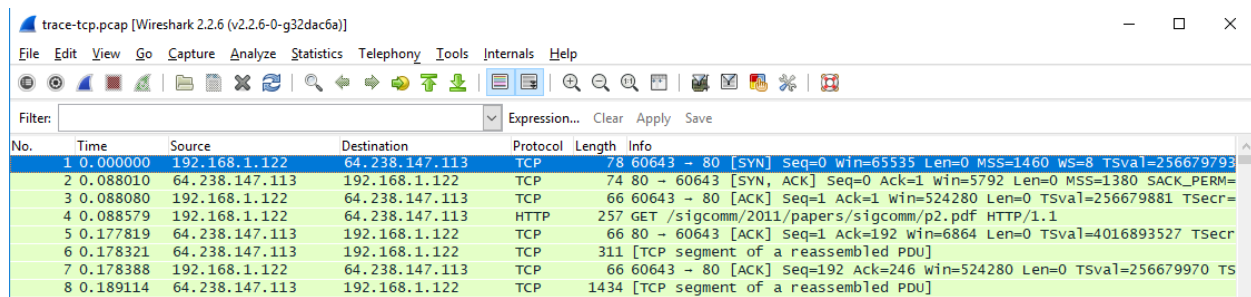
For this part, we are going to launch an older version of Wireshark called Wireshark legacy. You do this by selecting the Wireshark legacy application as follows.



When it launches, you should open the trace-tcp file which is in your downloads folder from earlier.



You should then be presented with the same trace-tcp as used previously in this exercise.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.1.122	64.238.147.113	TCP	78	60643 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=8 TSval=256679793
2	0.088010	64.238.147.113	192.168.1.122	TCP	74	80 → 60643 [SYN, ACK] Seq=0 Ack=1 win=5792 Len=0 MSS=1380 SACK_PERM=
3	0.088080	192.168.1.122	64.238.147.113	TCP	66	60643 → 80 [ACK] Seq=1 Ack=1 win=524280 Len=0 TSval=256679881 TSecr=
4	0.088579	192.168.1.122	64.238.147.113	HTTP	257	GET /sigcomm/2011/papers/sigcomm/p2.pdf HTTP/1.1
5	0.177819	64.238.147.113	192.168.1.122	TCP	66	80 → 60643 [ACK] Seq=1 Ack=192 win=6864 Len=0 TSval=4016893527 TSecr=
6	0.178321	64.238.147.113	192.168.1.122	TCP	311	[TCP segment of a reassembled PDU]
7	0.178388	192.168.1.122	64.238.147.113	TCP	66	60643 → 80 [ACK] Seq=192 Ack=246 win=524280 Len=0 TSval=256679970 TS
8	0.189114	64.238.147.113	192.168.1.122	TCP	1434	[TCP segment of a reassembled PDU]

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” (as shown below).

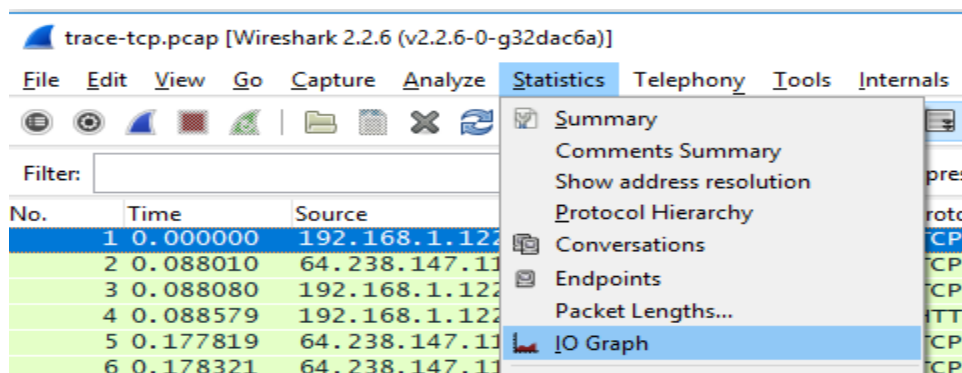


Figure 16: Opening an IO graph

You should end up with a graph like below. By default, this graph shows the rate of packets over time. 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.

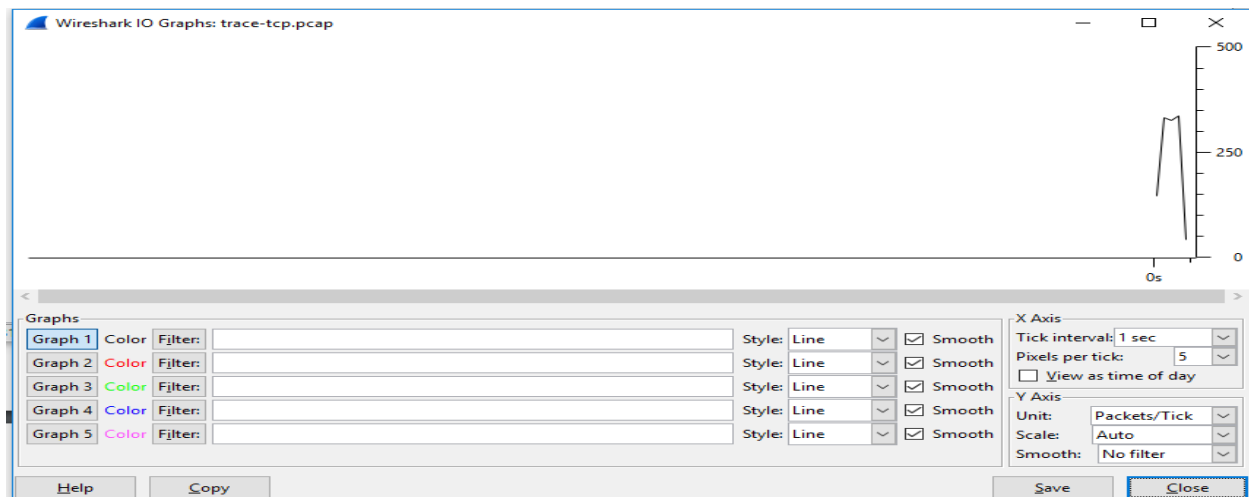
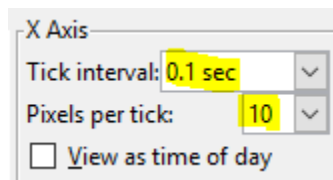


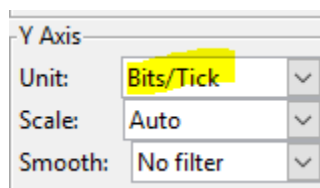
Figure 17: The IO graph

Now we will tweak it to show the download rate with the changes given below

- *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. Make this 10. See below.



- *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 on next page. 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. 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.

Note, 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.

Try clicking on parts of the graph and watch where you are taken in the Wireshark trace window.



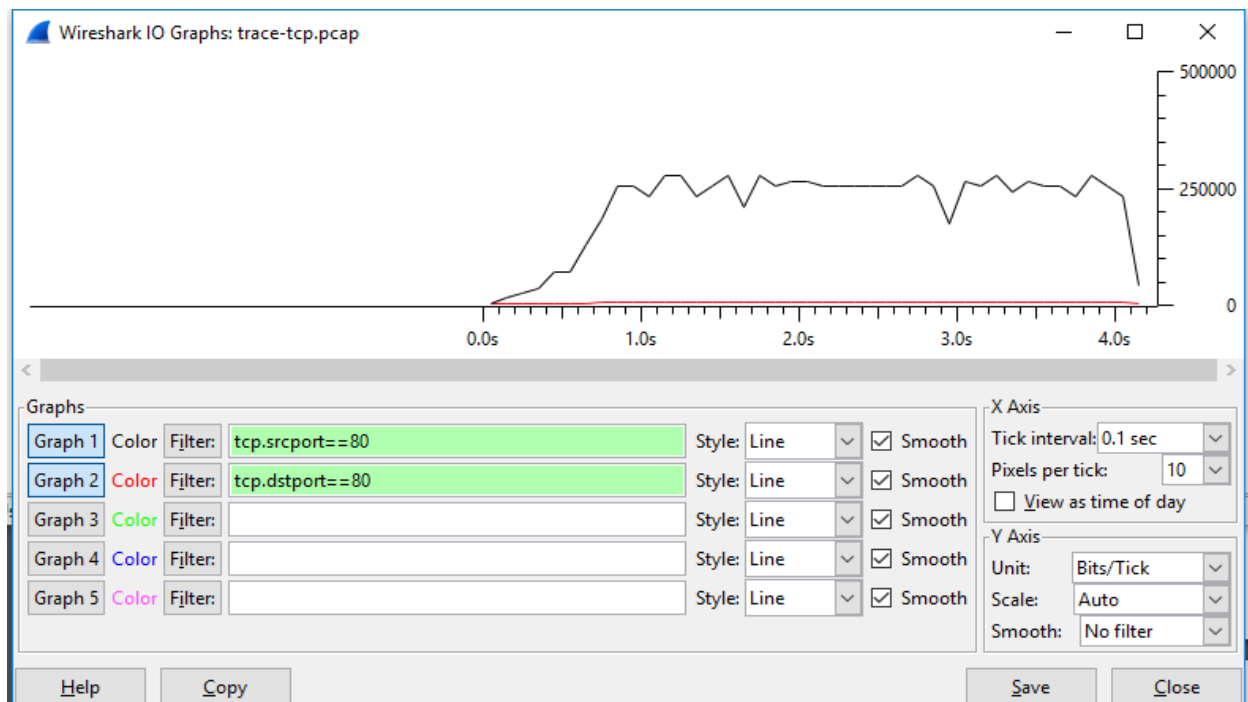


Figure 16: TCP download rate over time via an IO graph

*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.

Note the data rate in the download direction in packets/second and bits/second once the TCP connection is running well is 250 packet/sec and 2.5 Mbps.

Our download packets are 1434 bytes long, of which 1368 bytes are the TCP payload carrying contents. Thus 95% of the download is content.

*The data rate in the upload direction in packets/second and bits/second due to the ACK packets is 120 packets/sec and 60,000 bits/sec. We expect the ACK packet rate to be around half of the data packet rate for the typical pattern of one delayed ACK per two data packets received. The ACK bit rate will be at least an order of magnitude below the data bit rate as the packets are much smaller, around 60 bytes.*

The Ack number tells the next expected sequence number therefore it will be X plus the number of TCP payload bytes in the data segment.