**Lab Exercise – UDP & TCP**

**Objective**

UDP, or User Datagram Protocol, serves as an alternate communication method to TCP,

prioritizing swift connections with the ability to withstand data loss. It's commonly

employed for applications needing real-time responsiveness over 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 offers lower bandwidth overhead and latency compared to TCP by simply sending

packets without establishing a connection. However, this can result in packet loss or out

-of-order delivery due to varied network paths. It suits applications like gaming and real

-time communications where minimal latency is crucial, tolerating some data loss without

significant quality impact. Techniques like forward error correction can mitigate quality

issues. UDP is also used for lossless data transmission when applications manage packet

retransmission and ordering, enhancing data transfer rates for large files compared to TCP.

**Step 1: Capture a UDP Trace**

There are several ways to initiate UDP traffic on your computer:

1. Many background system protocols like DHCP (IP address assignment) and NTP

(time synchronization) generate UDP traffic without user intervention.

1. Browsing the web triggers UDP traffic through DNS queries that resolve domain

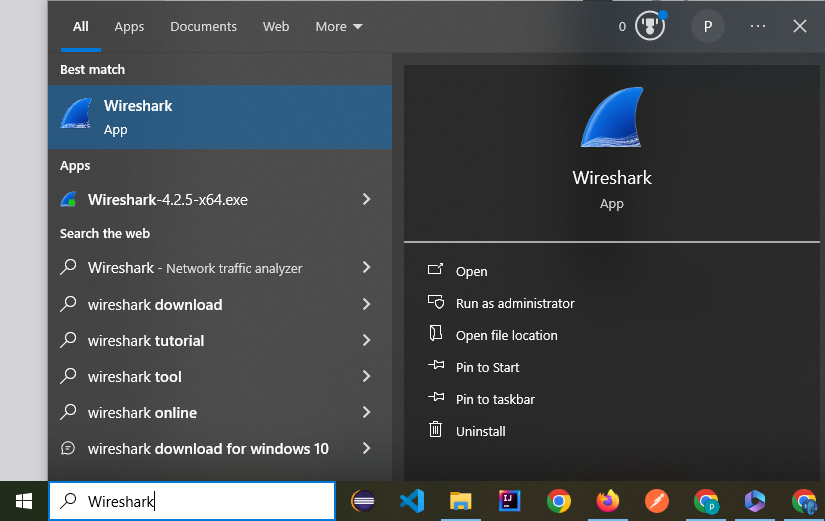
names to IP addresses. It's essential to visit trusted sites to avoid potential security

risks.

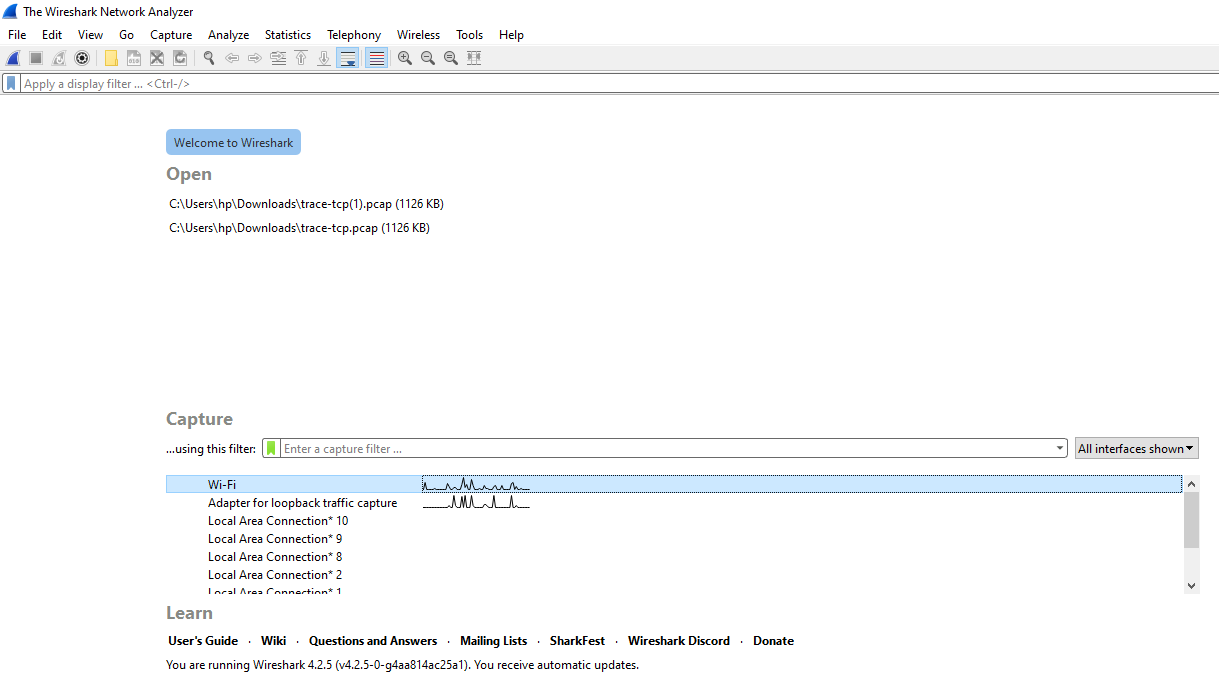
1. Using voice-over-IP applications for calls utilizes UDP via RTP, which transports

media samples (audio, video) during calls over the Internet.

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

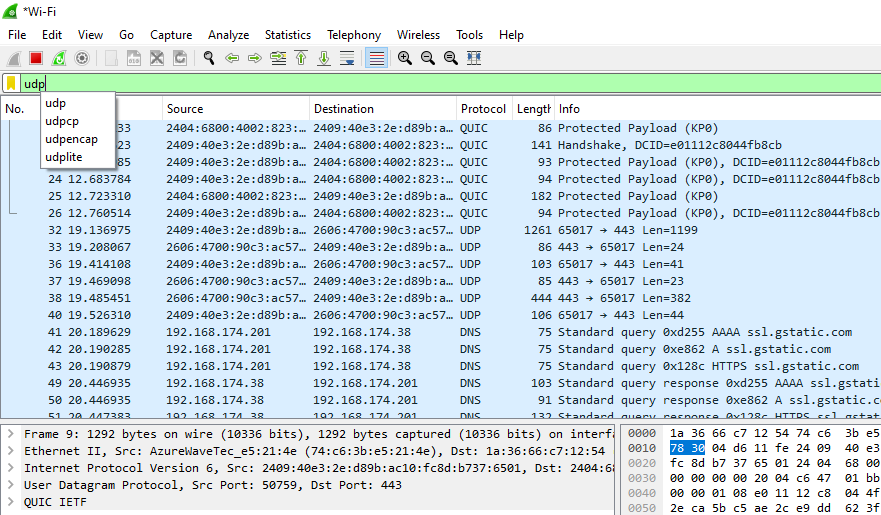


**2. Once Wireshark starts, select the Wifi interface.**



**3. Wireshark will automatically start capturing packets on the network.**

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



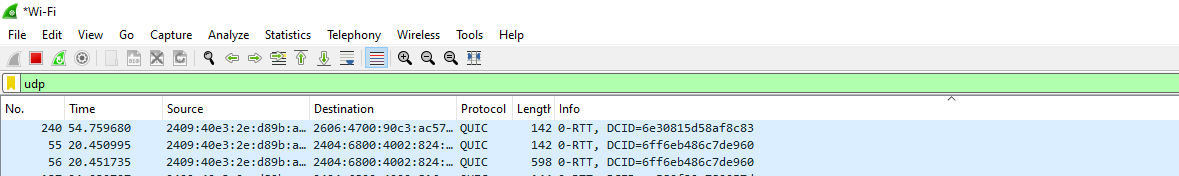
4. When the capture is started, it will collect UDP traffic automatically.

5. 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.

6. Use the Wireshark menus or buttons to stop the capture.



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

**Step 2: Inspect the Trace**

Inspecting UDP traffic using Wireshark reveals application protocols layered atop UDP, not

directly showing UDP in the protocol column. Each UDP packet includes source and

destination ports, message length, and a checksum for validation. Wireshark may flag

checksum errors due to protocol offloading, where NIC computes checksums after

Wireshark captures packets. Adjusting preferences can prevent false checksum errors.

The UDP header is concise, with the payload revealing higher-layer protocols like DNS or

RTP, illustrating the simplicity and functional focus of UDP in network communications.

**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.

**Step 4: UDP Usage**

The structure of a UDP message includes an 8-byte header with fields like Source Port,

Destination Port, Length (UDP header + payload), and a 16-bit Checksum. IP headers use

a Protocol field value of 17 to indicate UDP. UDP messages in traces can originate from or

be sent to various IP addresses due to their widespread use in system protocols like DNS,

NTP, and DHCP, which utilize broadcast and multicast addresses for communication

across networks. Understanding these addresses helps in identifying UDP traffic sources

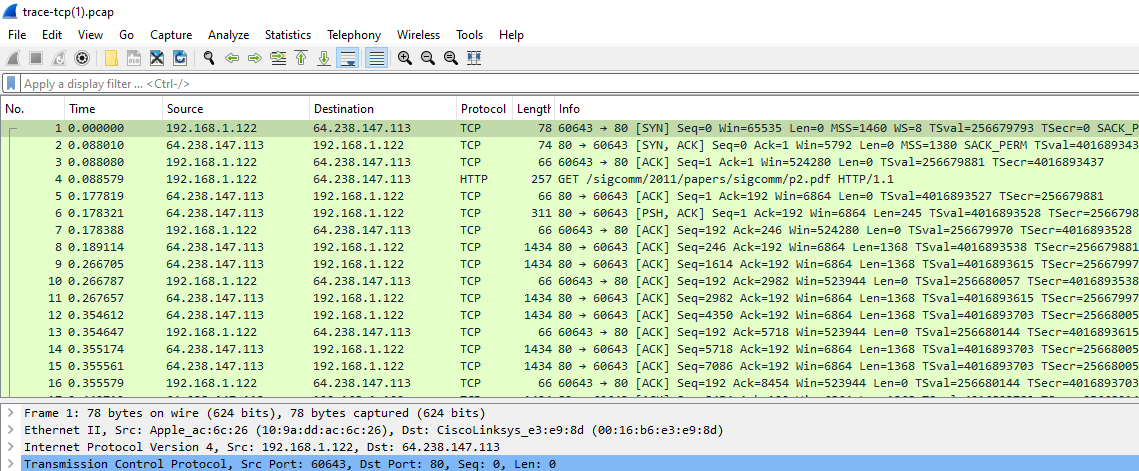
and destinations during network analysis.

**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**

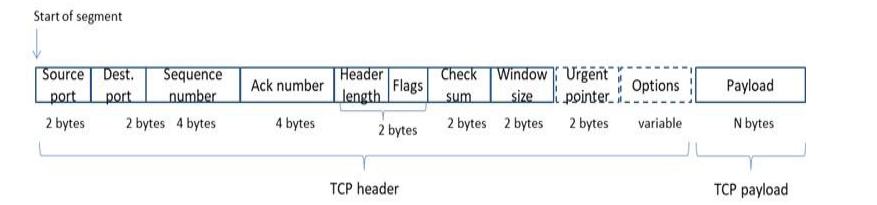


**Step 2: Inspect the Trace**

In a TCP packet captured by Wireshark, the protocol layers reveal key details:

1. Source and destination ports are identified, likely port 80 for web server communication.
2. The sequence number indicates the position of the first byte in the payload stream.
3. An acknowledgement field shows the last received byte in reverse transmission.
4. Header length specifies the size of the TCP header.
5. Flags denote segment type, including ACK and SYN.
6. Checksum detects transmission errors for reliability.
7. Optional fields like TCP options can be expanded for additional settings.
8. TCP payload carries transported bytes. Wireshark offers further contextual information for packet analysis.

**Step 3: TCP Segment Structure**



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.

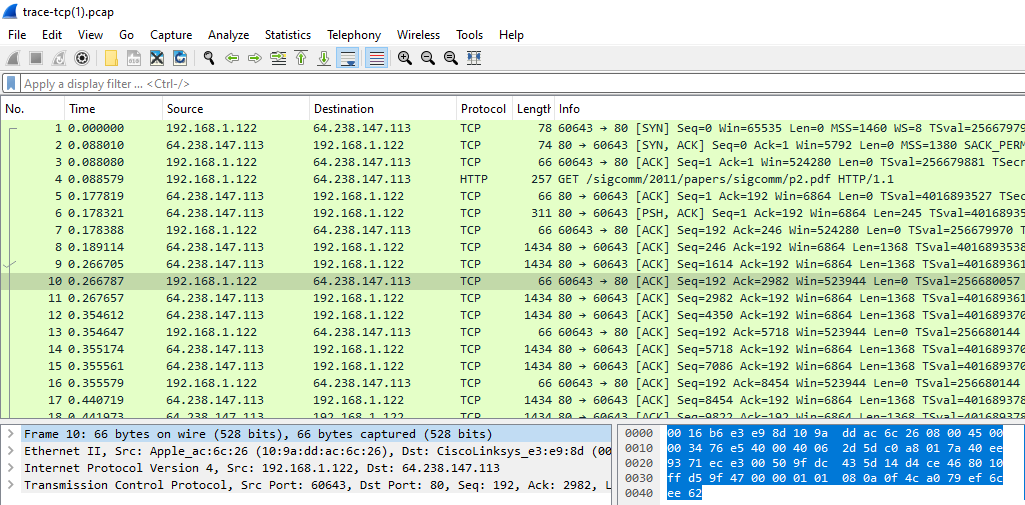
• The Options field is shown dotted, as it may or may not be present for the segments in

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.

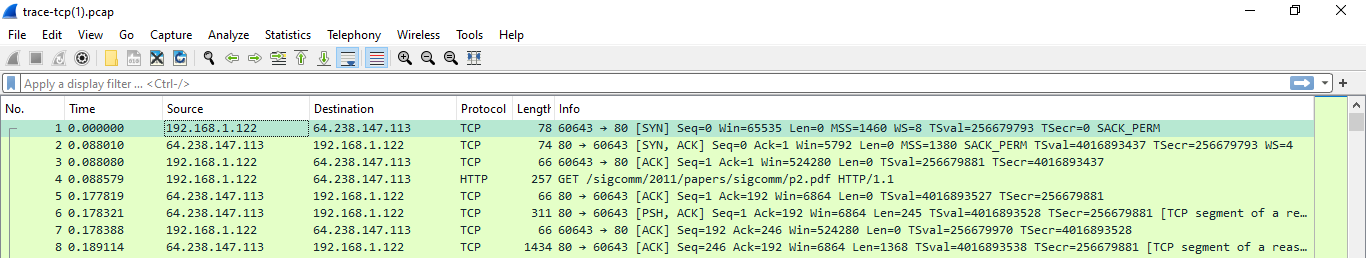


**Step 4: TCP Connection Setup/Teardown**

Three-Way Handshake

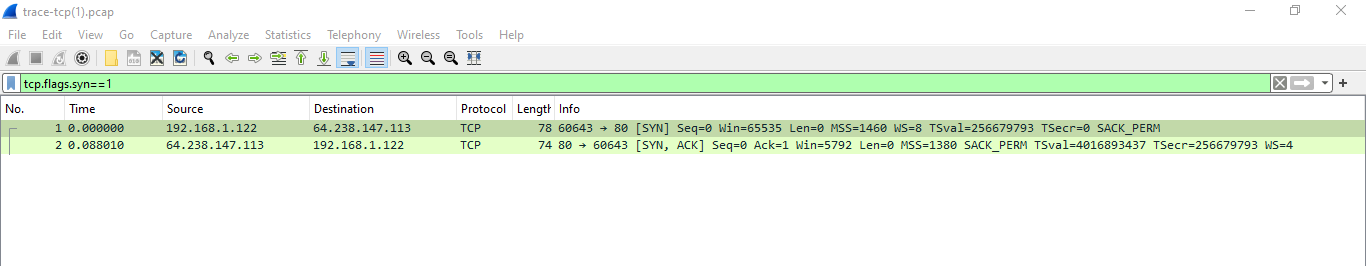
To see the “three way handshake” in action, look for a TCP segment with the SYN flag on.

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



The 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)



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

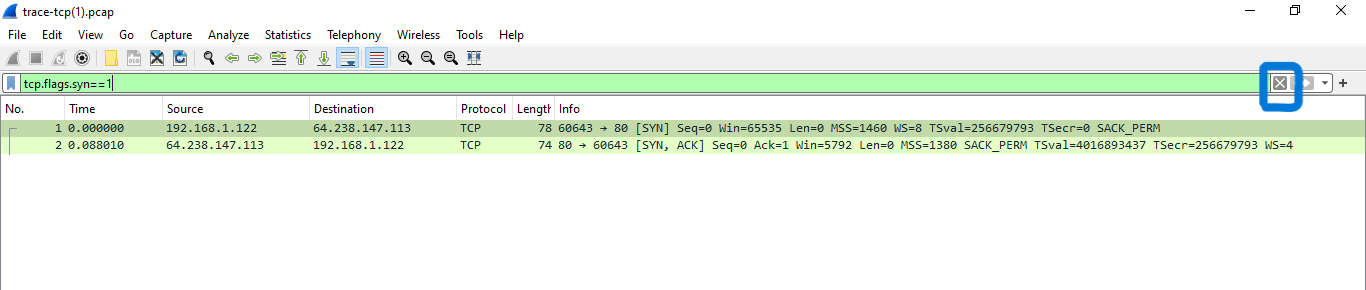
computer will ACK it, consider the connection set up, and begin sending data, which in

this case will be the HTTP request.10

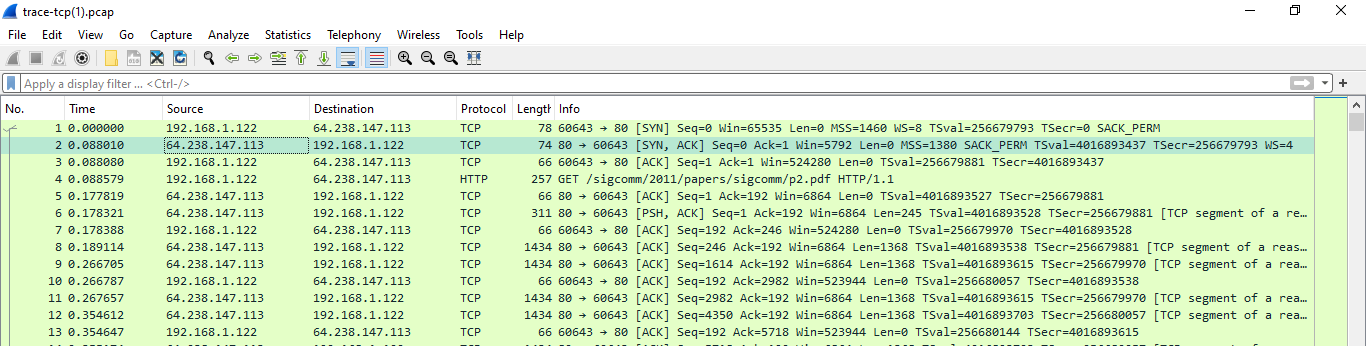
**Step 5: TCP Connection Setup/Teardown**

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.



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



The three-way handshake establishes a TCP connection between a client and a server. It

involves sending three packets: SYN, SYN-ACK, and ACK.

1. The client sends a SYN packet with its initial sequence number.
2. The server responds with a SYN-ACK packet containing its own sequence number

and acknowledging the client's SYN.

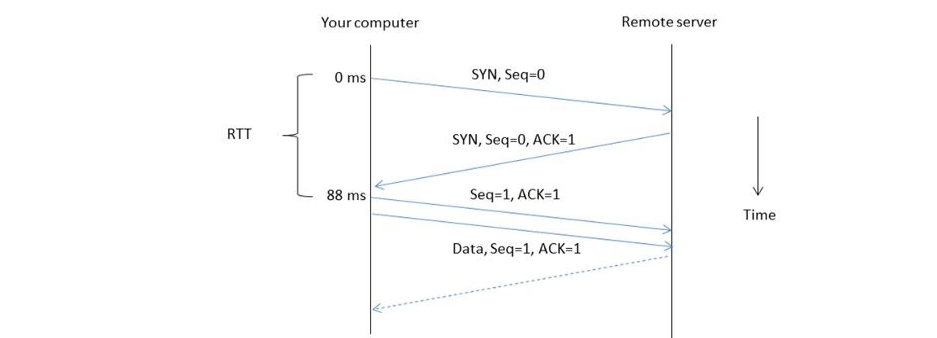
1. The client sends an ACK packet acknowledging the server's SYN-ACK.

After the handshake, the client can send data to the server.

1. The initial sequence numbers are typically zero.
2. The ACK number is the corresponding sequence number plus 1.
3. The client can send data immediately after sending the ACK packet.
4. The RTT is the time it takes for a packet to travel from the client to the server and back.

**Step 6: Connection Options**

TCP SYN packets negotiate parameters like MSS (max segment size) and timestamps during connection setup. Options like NOP and End of Option List are for formatting. Options can be used in regular segments too, depending on their purpose. Our example uses MSS, Window Scale, SACK, and Timestamps options.



The FIN teardown handshake is similar to SYN handshake but uses FIN flag instead. Sequence and ACK numbers are incremented by 1 for FIN flag. Teardown can be initiated by either client or server.

**Step 7: FIN/RST Teardown**

TCP connection teardown happens after data transfer. Two common methods are FIN and

RST. FIN uses a three-way handshake similar to SYN handshake, while RST is abrupt

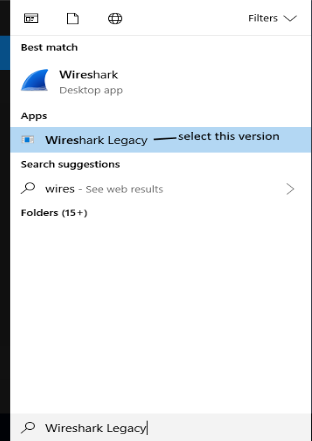
termination. Our example shows RST teardown initiated by the client. Sequence and ACK

numbers are not important for abrupt teardown and there is no RTT estimation.

**Step 8: TCP Data Transfer**

For this part, we are going to launch an older version of Wireshark called Wireshark legacy.

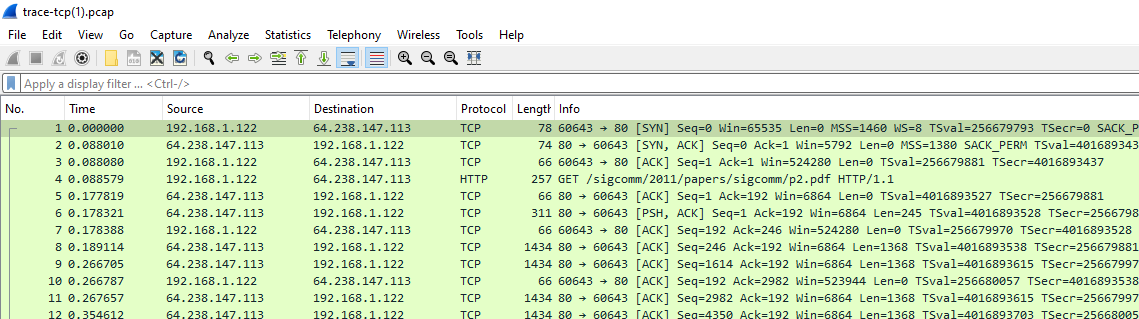
You do this by selecting the Wireshark legacy application as follows.



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.

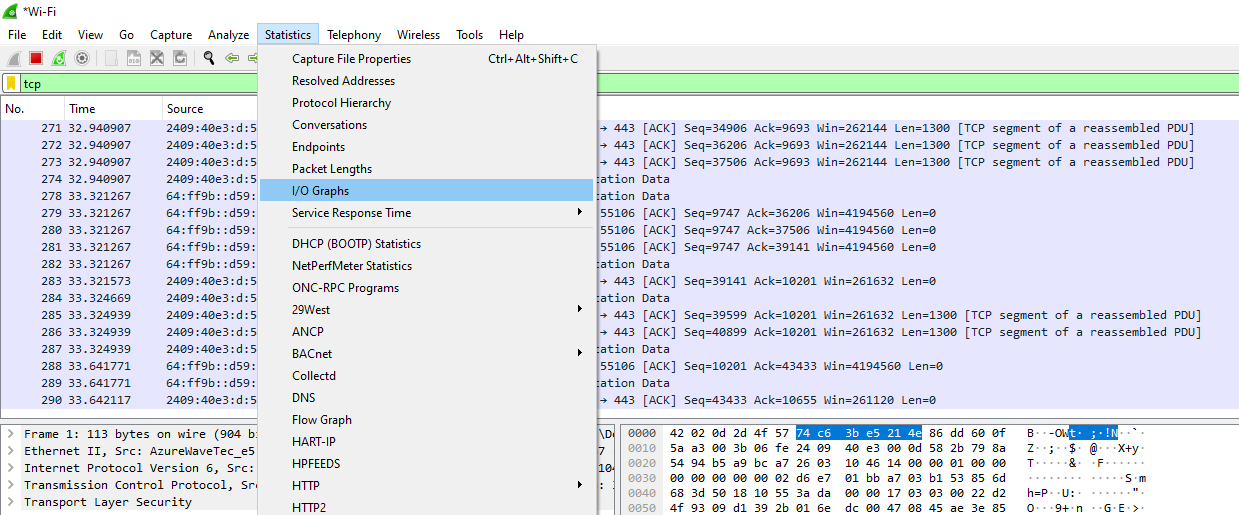


The middle portion of the TCP connection is the data transfer, or download, in our trace.

This is the main event. To get an overall sense of it, we will first look at the download rate

over time.

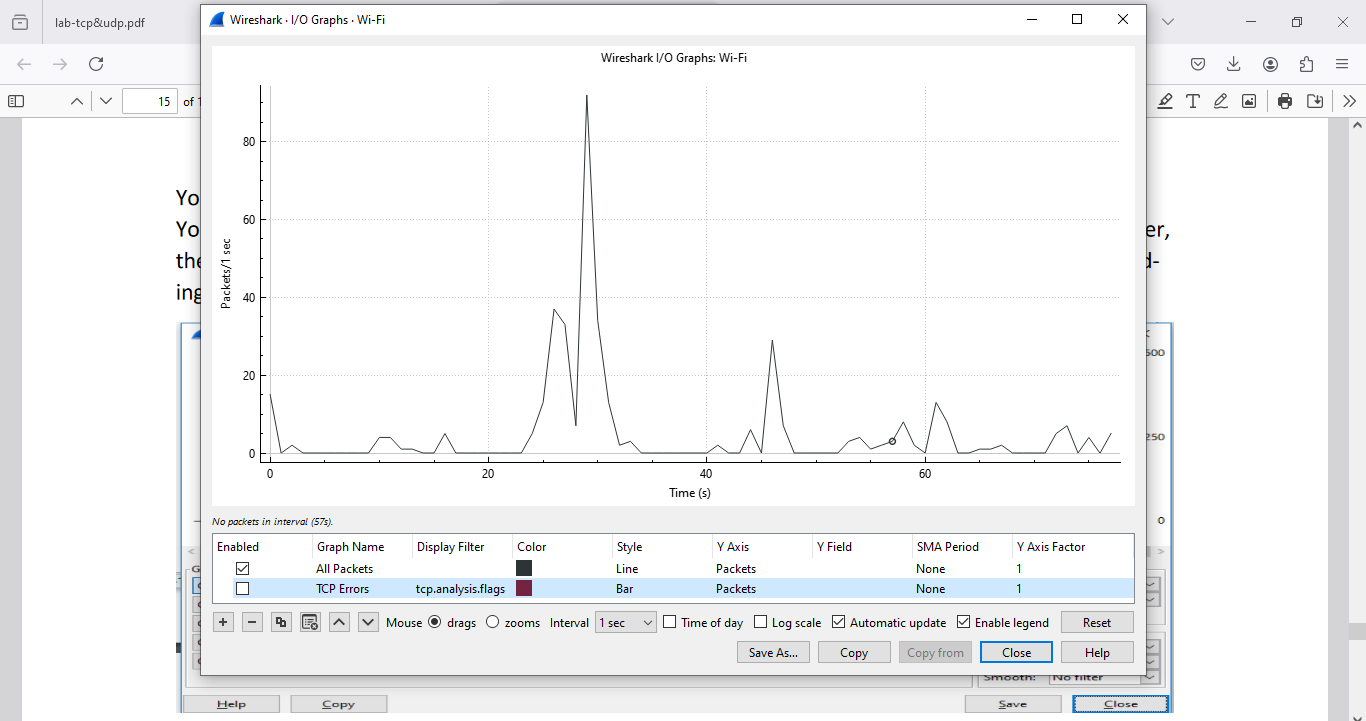
Under the Statistics menu select an “IO Graph” (as shown below).



Instead of using "TCP Stream Graph" for analyzing packet rate over time, use the default

graph that shows packet rate. This is because our trace captures data received and "TCP

Stream Graph" assumes data is sent from the machine where trace is captured.



To analyze download rate over time, adjust the graph settings:

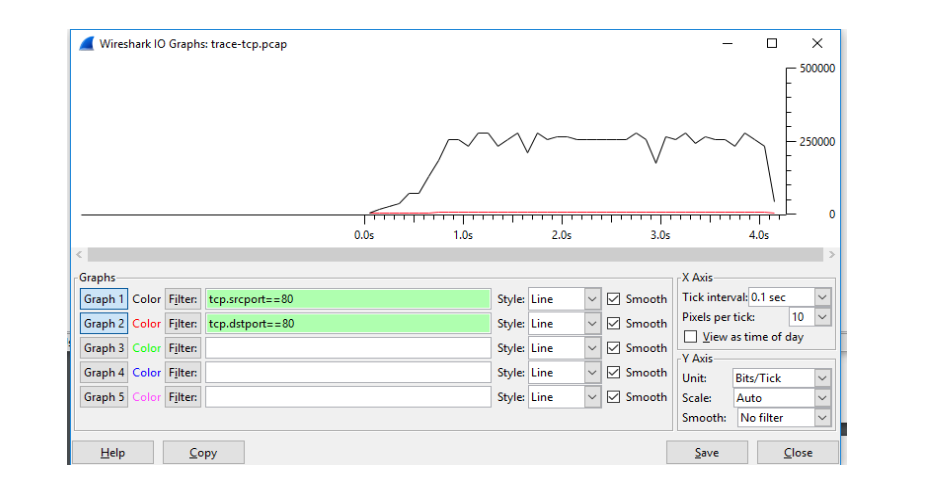
X-axis: Tick interval: 0.1 seconds, Pixels per tick: 10 (shows details without cluttering)

Y-axis: Change unit to Bits/Tick (easier for throughput calculation)

Filter: "tcp.srcport==80" to see only downloaded data from port 80 (web server)

This will show download rate (increasing to steady state) and a small upload rate (ACKs).

Real downloads may vary due to network conditions or server control.



Analyze download packets in the middle of the trace:

1. See pattern of data segments with corresponding ACKs sent back (Delayed ACKs).

2. Download sequence number increases, while upload sequence number stays the same.

3. Window info in each segment tells available buffer space.

4. Download rate is 250 packets/sec (2.5 Mbps), with 95% data payload.

5. Upload rate is 120 packets/sec (60 kbps) due to ACKs (around half of download rate).

6. ACK number is the expected sequence number + data payload bytes in the previous data segment.