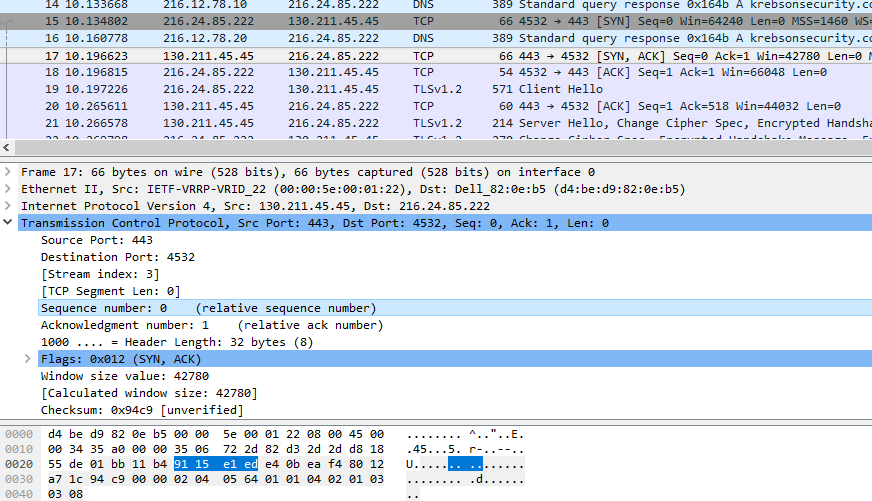
# Lab 6 TCP Handshake, Sequence and Acknowledgement Numbers

TCP uses sequence and acknowledgement numbers to keep track of which bytes and packets have been sent, received, or lost, and to put packets received out of order into the original order.

Sequence and acknowledgement numbers are 4 bytes (32 bits) long, running from 0 – 0xFFFFFFFF in hex or 4,294,967,295 in decimal. They start at a random number to make spoofing attacks more difficult. Wireshark displays the numbers in the Packet Details panel as “relative” numbers that always start at 0. The actual number is in the Packet Bytes panel. In the screenshot below, the relative sequence number for this SYN packet is 0, but the actual number is 0x9115e1ed (decimal

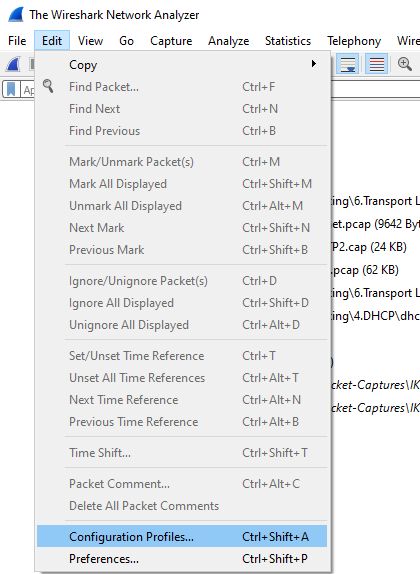
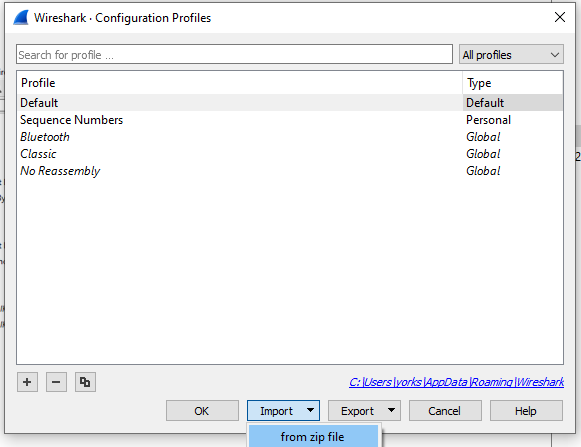


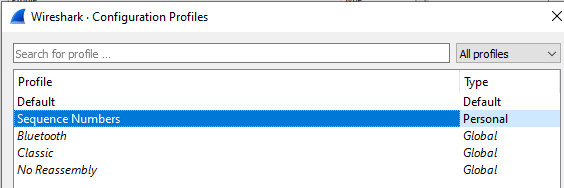
For the rest of this lab we will use the relative numbers from Wireshark, keeping in mind that the actual numbers are different.

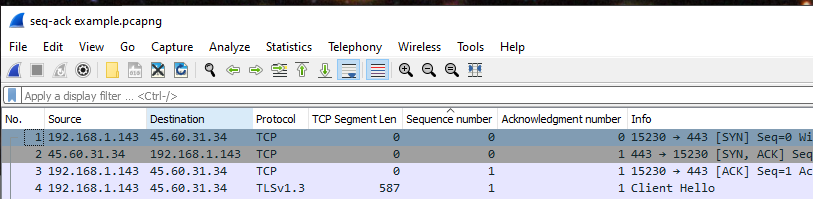
TCP sequence numbers increase by the number of bytes in the TCP data section of the packet; in a way, you can think of them as serial numbers for the bytes sent. SYN and FIN packets are exceptions; they contain no data but increase the sequence numbers by one. Note that acknowledgement numbers are higher than the sequence number of the packet the host just received. The host is saying, “The next sequence number I expect to receive is xxx.”

## Modify Wireshark to see Sequence and Acknowledgment Numbers in Action.

For this part of the lab we will use a special configuration to help us understand sequence (seq) and acknowledgement (ack) number. Download the file seq-ack-profile.zip from Canvas. In Wireshark select Configuration Profiles, then Import > from Zip file.

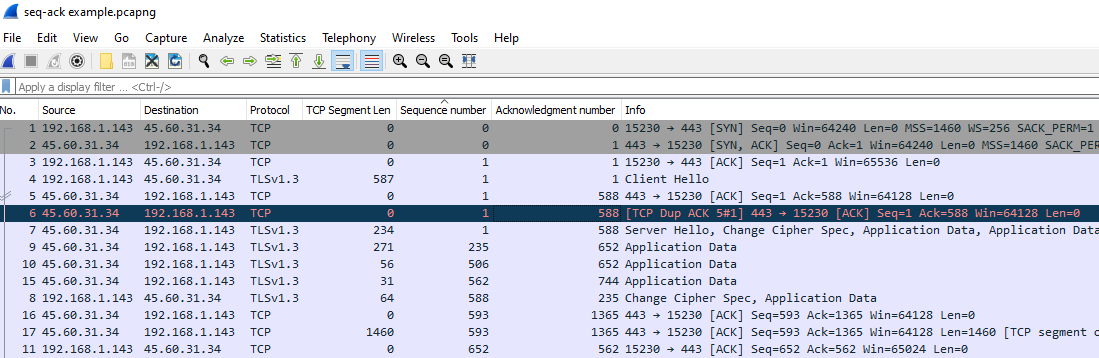
 

Find the seq-ack-profile.zip file and let Wireshark open it. In the same panel (Edit > Configuration Profiles) select the Sequence Numbers profile. Note: To get your Wireshark back to normal after this lab is done, you will need to change the profile back to Default.  


You should see Wireshark columns that look like this.  


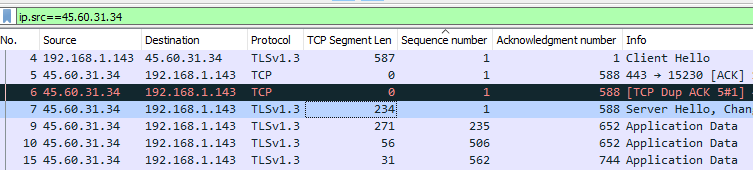
If the profile does not import properly, you can create it using the procedure in the appendix to this lab.

## Packet Capture File

There should be a packet capture file associated with this lab called seq-ack-example.pcapng. Open that file with Wireshark. It is a single stream between 192.168.1.143 and 45.60.31.34, where most of the stream has 45.60.31.34 sending data to 192.168.1.143.  


Note: When Wireshark detects an error in the SEQ or ACK numbers, it highlights the packet in black. Packet 6, above, is a repeat of an ACK packet that 45.60.31.34 in packet 5. Often a duplicate ACK means that the host has missed a packet and wants the packet re-sent. That did not happen in this case, so I am not sure why we received the duplicate packet 6.

### Packets sent by 45.60.31.34



Put ip.src==45.60.31.34 in the display filter so that we only see packets sent by 45.60.31.34 and look at packet 7. The Sequence Number is 1 and the length of the data in the TCP portion (TCP Segment Length) is 234. Then look at the next packet sent by 45.60.31.34, packet 7. The Sequence Number is 235 (234 plus 1), and the TCP Segment Length is 271. In packet 8 the Sequence Number is 506, 235 plus 271.  
A picture containing graphical user interface

Description automatically generated

You can always compute the sequence number of the next packet by adding the sequence number and length of the current packet. The sequence number is basically the number of bytes that the source has sent so far in the connection.

Now change the display filter to be ip.addr==45.60.31.34 (ip.addr instead of ip.src.) This will show both sides of the conversation, instead of just the side where 45.60.31.34 is sending. With both sides going, it is harder to follow the numbers, so concentrate.

Both sides are constantly calculating the bytes they have sent and received, and both sides use acknowledgement numbers to verify that both sides agree on how many bytes have been sent and received.

Graphical user interface, text, application, chat or text message

Description automatically generated

In the screenshot above, 45.60.31.34 (1) is sending a packet with a TCP Segment Length of 234 bytes (2), and the current sequence number is 1. When 192.168.1.143 receives (4) the packet, it does the same calculation the sender does. It adds 234 (length) and 1 (seq number) to get 235. Therefore, it sets its acknowledgement number to 235 (5), which means “The next sequence number I expect to see from you is 235, and I have received all your packets before that.” Sure enough, the next packet from 45.60.31.34 has a sequence number of 235 (6) and everything is going according to plan; no packets have been lost.

### Packets sent by 192.168.1.143

The same process is going on it the other direction as well, when 192.168.1.143 sends data to 45.60.31.34.

Graphical user interface

Description automatically generated

In the screenshot above, the orange ellipses are the ones we looked at before. The red ellipses with numbers mark data going in the opposite direction, from 192.168.1.143 to 45.60.31.34. Now, 192.168.1.143 (1) is sending a packet with a TCP Segment Length of 92 bytes (2), and the current sequence number is 652. When 45.60.31.34 receives (4) the packet, it does the same calculation the sender does. It adds 92 (length) and 1 (seq number) and gets 744. Therefore, it sets its acknowledgement number to 744 (5), which means, “The next sequence number I expect to see from you is 744, and I have received all your packets before that.”

Note that 192.168.1.143 did not wait for acknowledgement 744 from 45.60.31.34 before it sent more data. This is allowed because it speeds the connection. There is another part of the TCP header called Window Size that says how far ahead a sender can get before it has to wait for an acknowledgement. This keeps the receiver from getting data faster than it can handle it.

To review, when the sender transmits a packet, it adds the TCP segment length to the previous sequence number to calculate the value for the current sequence number. The receiver adds the TCP length and the sequence number and sends that back to the sender as an acknowledgement number. The acknowledgement number means, “The next sequence number I expect to see from you is 744, and I have received all your packets before that.”

## Missing Packets

When a packet is lost in transit, it looks like the screenshot below.

A picture containing text

Description automatically generated

The sender, 45.60.31.34 (1), sent a packet to 192.168.1.143 with a TCP length of 1460 (2) and a sequence number of 445064 (3). The next sequence number should be 1460 + 445064 = 446524. However, the next packet we see from 45.60.31.34 has sequence number 458204. We lost the packet with sequence number 446524! Wireshark realizes this immediately and shows us there is a problem by coloring the packet black and adding the message, “TCP Previous segment not captured.” (5)

Meanwhile the receiver 192.168.1.143 (6) has done the same calculation and realized that sequence number 446524 is missing. It stops acknowledging packets after that and just sends acknowledgement number 446524. It is saying, “I’m missing 446524! I’m going to keep sending the acknowledgement for 446524 until you resend it!”

Timeline

Description automatically generated

It takes a long time (about a tenth of a second) for 45.60.31.34 (1) to realize it needs to resend the packet with sequence number 446524 (2). Wireshark notices the resent packet and shows a TCP Fast Retransmission message (3).

After a few packets, 192.168.1.143 (4) acknowledges that it received 446524 by asking for packets with higher sequence numbers (5). The sender retransmits packets that were lost while the receiver was waiting for 446524 (6), and the receiver acknowledges them (7).

Finally, the sender and receiver are back in sync again (8)!

## The Point

The point here is not that you should be able to trace any lost packet in Wireshark and show exactly how TCP fixed the problem (a good packet monkey can, though ;-). The point is to know that the TCP stacks on both the sender and receiver are keeping track of all the data and will take action to repair any losses.

Note that it took about 0.1 seconds (100 msec) for TCP to fix the lost packet in this example. People notice this gap if it is greater than 50 msec for audio and perhaps 100 msec for video. Imagine a channel that had more loss than the example we just looked at and was continuously losing packets and repairing the loss. It would be unpleasant to listen to or watch. That is why most audio and video streams are sent using UDP; instead of the problems caused by repairing loss, UDP just drops the packets. Also, try to imagine downloading a large program or application; an occasional delay will not bother you, but losing code in the middle of the program could cause huge problems. That is why most downloads use TCP.

# Your Turn

Change your Wireshark Preferences to show the columns we have used in this example. Use the file seq-ack-profile.zip and the procedure shown at the beginning of this assignment or use the procedure in the appendix to set the columns manually. Then load  
seq-ack-example.pcapng into Wireshark and try to reproduce what we did in the first part of this assignment, Packets sent by 45.60.31.34. You can search for the missing packet if you like, but that in not necessary.

Next, use Wireshark to make a packet capture while you download a file or open a website. Use the ip.addr filter to just show one side of the conversation. Try to find packets that show how the TCP segment length and the sequence number of one packet can be used to calculate the sequence number of the next packet. Once you have found an example, print a screenshot of your Wireshark display, and write on it to show the TCP length, sequence number, and sequence number of the following packet.

# Hand in

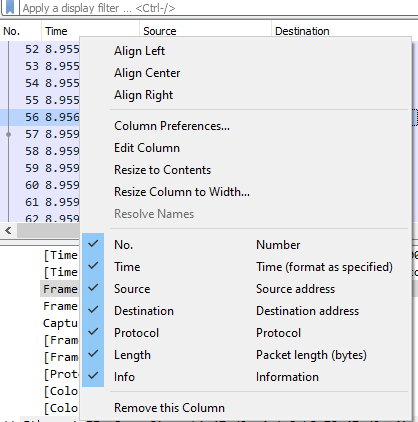
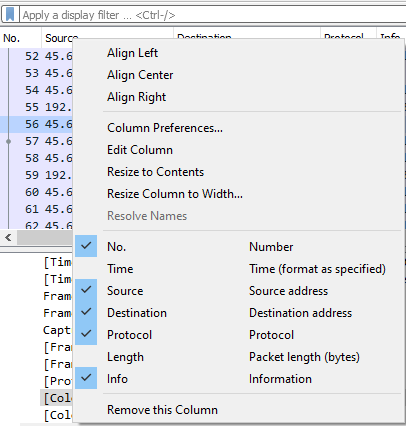
1. Hand in the screenshot of your Wireshark display.
2. Very briefly, describe how TCP recovers from lost packets
3. Normally, downloads use TCP and streaming media uses UDP. Why?

# Appendix—Changing the Wireshark Columns Manually

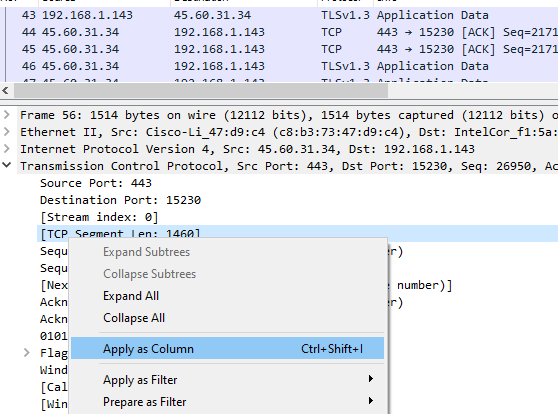
If you do not have access to the file seq-ack-profile.zip, or it does not import correctly, you can change your Wireshark columns this way. This is from <https://unit42.paloaltonetworks.com/unit42-customizing-wireshark-changing-column-display/>.

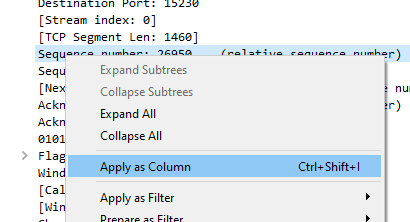
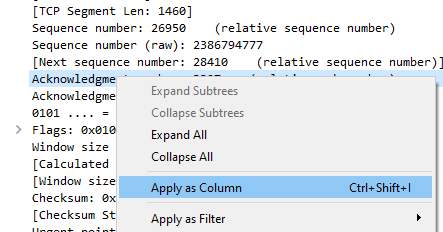
## Hide Columns

To save space, hide the Time and Length columns. The Length column is the total length of the frame (Ethernet, IP, and TCP), but the length we need is the length of only the TCP portion.

Right-click on any column header and remove the check mark from Time and Length.  
 

## Adding the TCP Segment Length, Sequence Number, and Acknowledgement Number columns.

Find a TCP packet in the Packet List (top) panel and click on it. Then click on the Protocol column so Wireshark will know where to put the new column. In the Packet Details Panel (middle) open Transmission Control Protocol, right-click on [TCP Segment Len], and select Apply as Column.  


Repeat the procedure for Sequence number (relative sequence number) and Acknowledgement number (relative ack number).  
 

Your new columns should look like this:  
