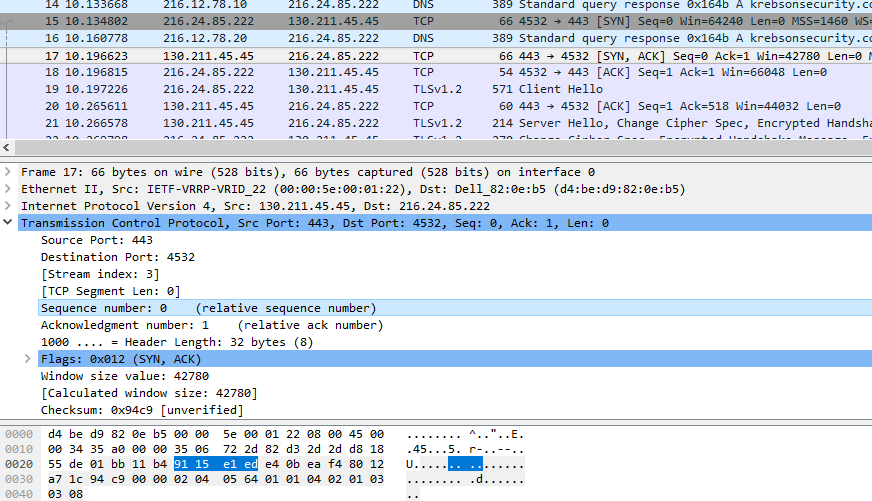
# Lab 6 TCP Handshake, Sequence and Acknowledgement Numbers

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, and start at a random number to make spoofing attacks more difficult. Wireshark displays them 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.

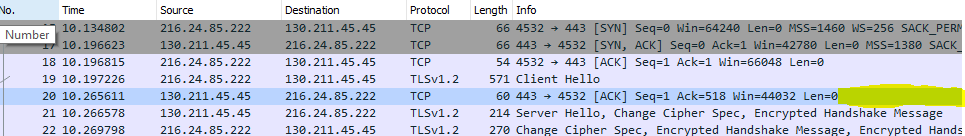
TCP sequence numbers increase by the number of bytes in the TCP data section of the packet; in a way, you can think of them as serial numbers for the bytes sent. SYN and FIN packets are exceptions; they contain no data, but increase the sequence numbers by one. Note that acknowledgement numbers are one 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.”

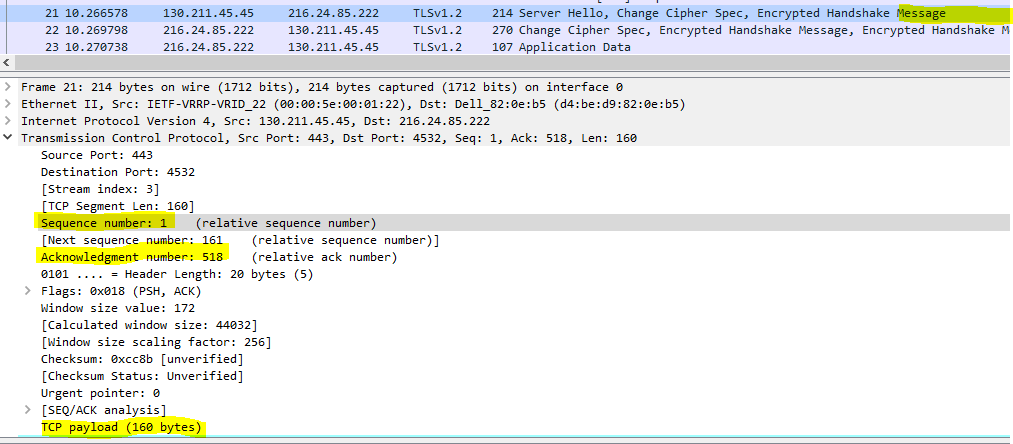
## Packet Capture

Use Wireshark to make a packet capture of your host visiting a web site. If the site is http://, you will be able to see the commands in plain (but sometimes compressed) text. If the site is https://, you will see plain text as the hosts negotiate encryption, but the rest of the packets will be unreadable. Either will work for this exercise.

Select a packet in the capture, and then right-click Follow > TCP Stream. Make sure your stream starts with the handshake (SYN, SYN/ACK, ACK) and includes several packets.

## Worksheet

Fill out the attached worksheet based on the direction of the packet (from your computer to the web server, or vice versa), Flags (SYN, SYN/ACK, or ACK, and FIN or RST at the end), relative sequence number, relative acknowledgment number, and packet length. For some packets, you can get the information you need from the packet list panel. The highlighted packet below is an example that shows the flag (ACK), sequence number (1), acknowledgement number (1) and length (0).  
 

For other packets, you will have to get your information from the packet details panel. 

As you fill out the worksheet, you should see the handshake progress with the sequence and acknowledgement numbers incrementing by one. Thereafter, they will increment by the length of the TCP payload. You will need to pay carefull attention to detail, as you may get two or three data packets in a row, with the ACK responses afterwards. If the client and server have selected a large window size, the client may only send ACK responses every few packets; the acknowledgement number in this case will be that for the most recent packet and means that the host has received all the previous packets. The sequence and acknowledge numbers are just bookeeping to keep track of all the bytes sent and received; bookeeping is easy for computers but tedious or difficult for many humans. Don’t obsess over the number--the lab is just intended to show that sequence numbers keep track of the bytes.

# Turn-in

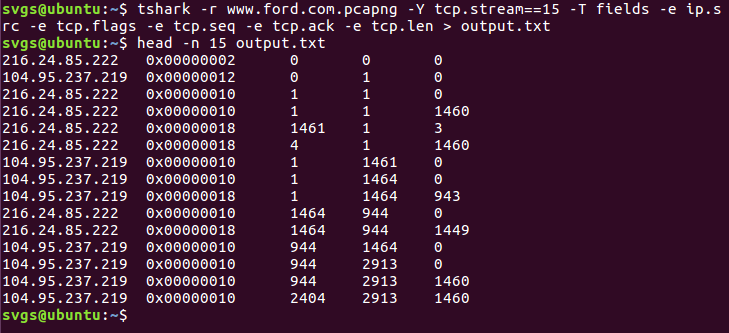
Turn in your worksheet.

## Creating a worksheet automatically (optional)

Wireshark comes with a command line companion called tshark. Wireshark processes large pcap files very slowly; tshark is much faster, although still slow by some standards. We can use tshark and our Linux text editing tools to make a quick, accurate list of sequence and acknowledgement numbers. This example uses a connection to ford.com saved in the file www.ford.com.pcapng. Wireshark was used to examine the file and find that the connection was stream 15 in the file (Follow > TCP Stream).

The command line was:  
tshark -r [www.ford.com.pcap](http://www.ford.com.pcap)ng -Y tcp.stream==15 -T fields -e ip.src -e tcp.flags -e tcp.seq -e tcp.ack -e tcp.len > output.txt

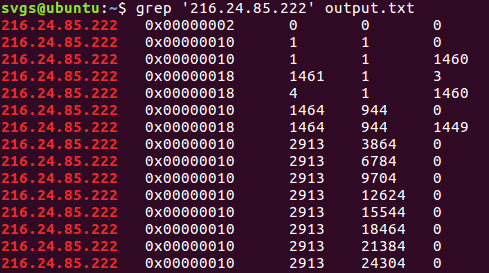
The -r option tells tshark to read from the pcap file. The -Y option, tcp.stream==15, does the same thing as using tcp.stream==15 in the Display Filter window in Wireshark (select stream 15). The -T option tells tshark we only want to extract certain fields, which are listed with the -e options. The final output is redirected to the file output.txt.

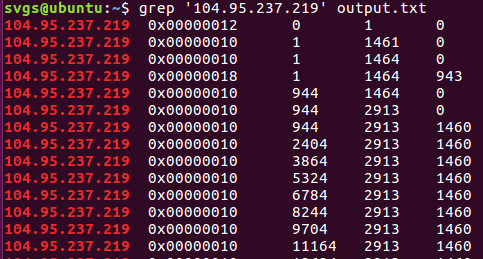
The file output.txt looks like this:  


Headers are:  
ip.src tcp.flags tcp.seq tcp.ack tcp.len

The tcp.flags column can be easily translated. 0x00000002 is SYN, 0x00000012 is SYN/ACK, 0x00000010 is ACK, and 0x00000018 is PSH/ACK. So, 216.24.85.222 starts the handshake with a SYN, 104.95.237.219 replies with a SYN\ACK, and 216.24.85.222 completes the handshake with an ACK.

It is easier to see how the sequence numbers and the packet payload length in bytes are related by looking at the traffic from the two hosts separately. We can do that easily enough with grep.

When we look at traffic from the client 216.24.85.222, we see that once the initial GET request is sent, the sequence number stays the same at 2913 as the client is sending no data to the server, just acknowledgements. The acknowledgement number increments as the client receives traffic from the server. (Note: there is an error in the fourth and fifth packets the client sent. Wireshark interpreted the fifth packet as a TCP Retransmission. The remaining packets are correct.)  
  
Headers are:  
ip.src tcp.flags tcp.seq tcp.ack tcp.len

When we look at the traffic from the server 104.95.237.219, we see that the sequence number begins to increment by 1460 (standard TCP payload for Ethernet is 1460 bytes per packet) for each packet once the file transfer begins. The acknowledgement number stays constant at 2913 once the server has received the GET request from the client.  
  
Headers are:  
ip.src tcp.flags tcp.seq tcp.ack tcp.len

Note: If you’ve gone this far, and read the entire document before you started working, you can submit screenshots for your data instead of filling out a worksheet.

We will use tshark again in later labs that better demonstrate tshark’s power.