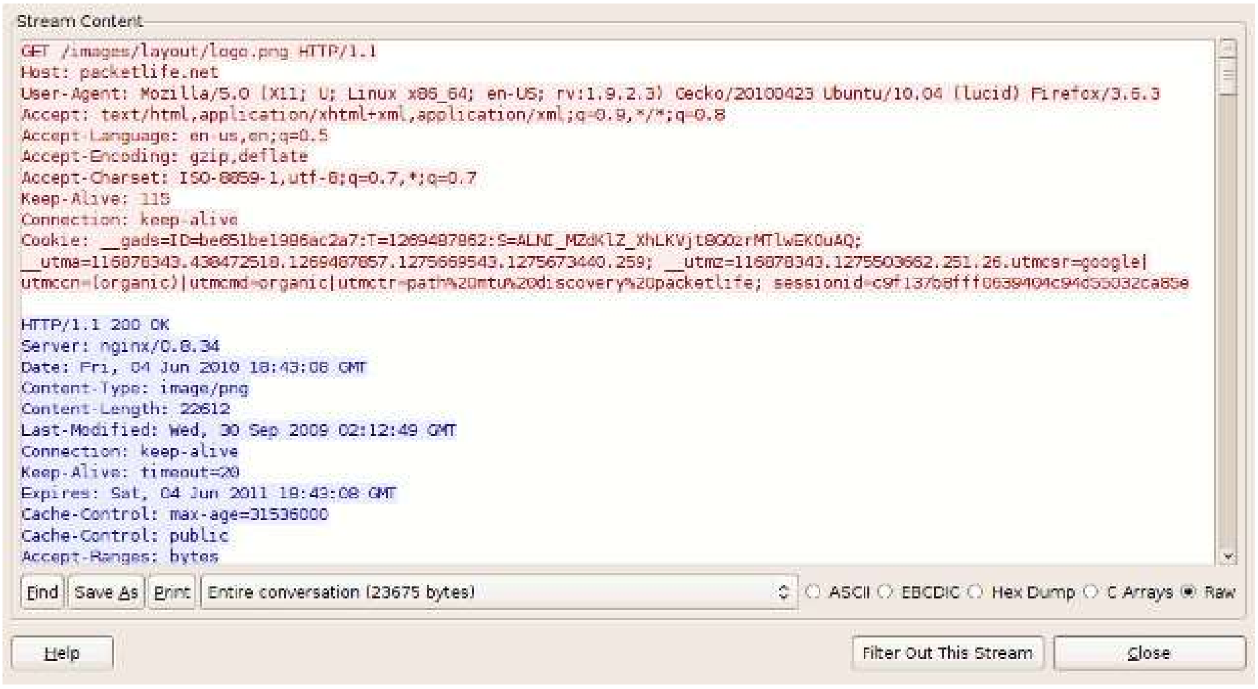
Understanding TCP Sequence and Acknowledgment Numbers

By stretch () | Monday, June 7, 2010 at 2:15 a.m. UTC

If you're reading this, odds are that you're already familiar with TCP's infamous "three-way handshake," or "SYN, SYN/ACK, ACK." Unfortunately, that's where TCP education ends for many networkers. Despite its age, TCP is a relatively complex protocol and well worth knowing intimately. This article aims to help you become more comfortable examining TCP sequence and acknowledgement numbers in the Wireshark (http://www.wireshark.org/) packet analyzer.

Before we start, be sure to open the example capture (/media/blog/attachments/424/TCP_example.cap) in Wireshark and play along.

The example capture contains a single HTTP request to a web server, in which the client web browser requests a single image file, and the server returns an HTTP/1.1 200 (OK) response which includes the file requested. You can right-click on any of the TCP packets within this capture and select **Follow TCP Stream** to open the raw contents of the TCP stream in a separate window for inspection. Traffic from the client is shown in red, and traffic from the server in blue.



```
GET /images/layout/logo.png HTTP/1.1
Host: packetlife.net
User-Agent: Mozilla/5.0 (X11; U; Linux x86_64; en-US; rv:1.9.2.3) Gecko/20100423 Ubuntu/10.04 (lucid) Firefox/3.6.3
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 115
Connection: keep-alive
Cookie: __gads=ID=be651be1986ac2a7:T=1269487862:S=ALNI_MZdKlZ_XhLKVjt8G0zrMTlwEK0uAQ;
__utma=116878343.438472518.1269487857.1275669543.1275673440.259; __utmz=116878343.1275503662.251.26.utmcsr=google|
utmccn=(organic)|utmcmd=organic|utmctr=path%20mtu%20discovery%20packetlife; sessionid=c9f137b8fff0639404c94d55032ca85e

HTTP/1.1 200 OK
Server: nginx/0.8.34
Date: Fri, 04 Jun 2010 18:43:08 GMT
Content-Type: image/png
Content-Length: 22612
Last-Modified: Wed, 30 Sep 2009 02:12:49 GMT
Connection: keep-alive
Keep-Alive: timeout=20
Expires: Sat, 04 Jun 2011 18:43:08 GMT
Cache-Control: max-age=31536000
Cache-Control: public
Accept-Ranges: bytes
```

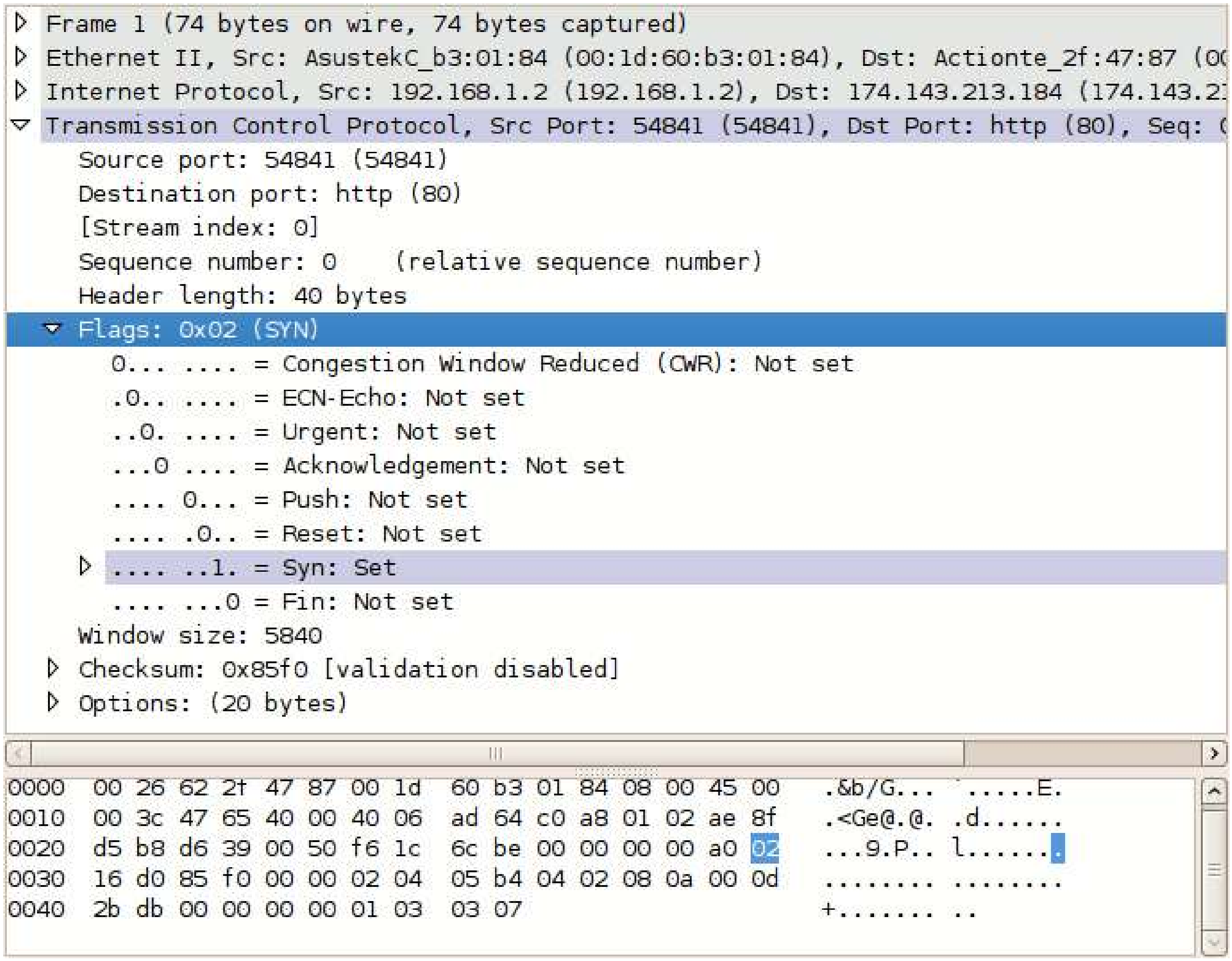
The Three-Way Handshake

TCP utilizes a number of flags, or 1-bit boolean fields, in its header to control the state of a connection. The three we're most interested in here are:

- **SYN -** (Synchronize) Initiates a connection
- **FIN -** (Final) Cleanly terminates a connection
- **ACK -** Acknowledges received data

As we'll see, a packet can have multiple flags set.

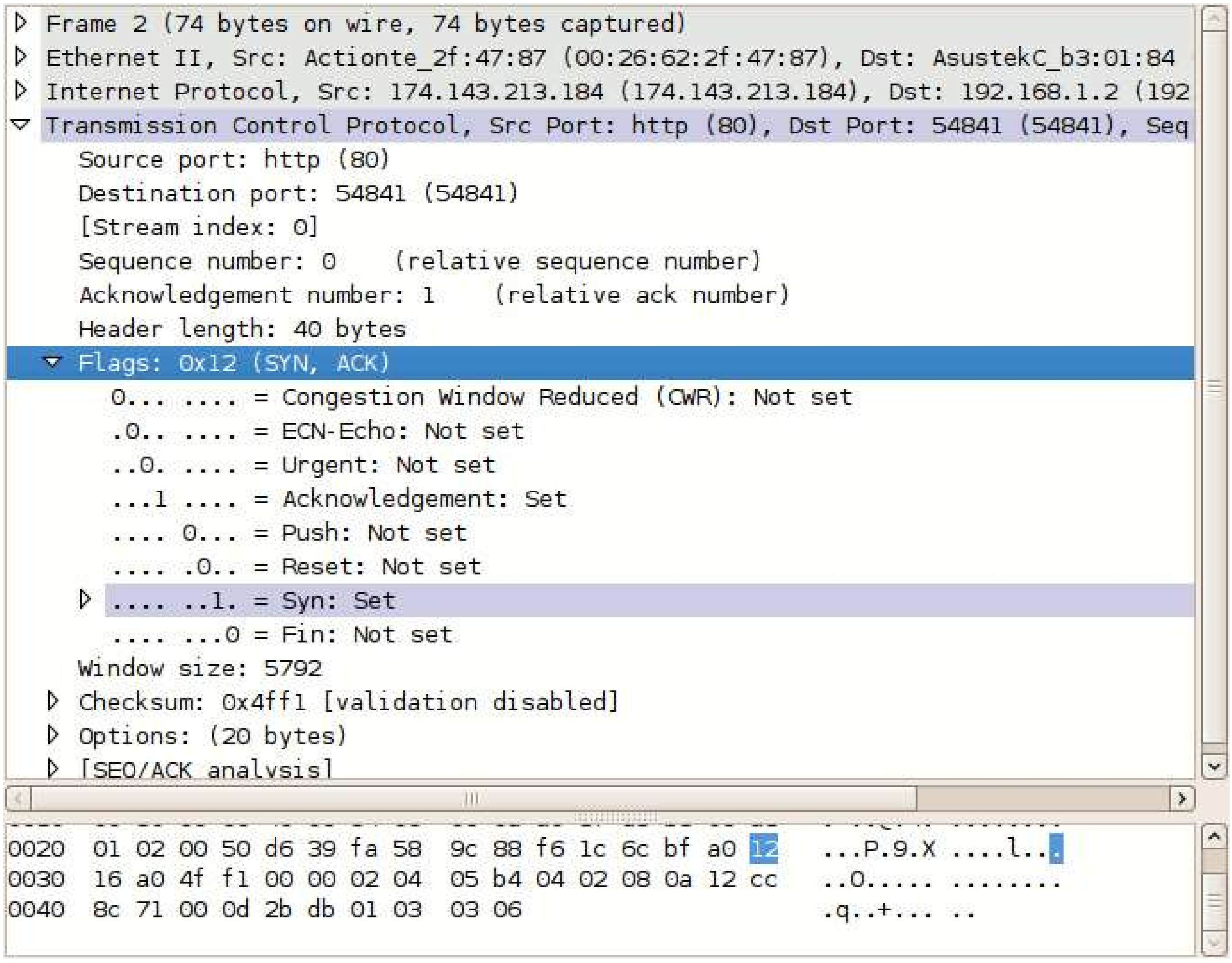
Select packet #1 in Wireshark and expand the TCP layer analysis in the middle pane, and further expand the "Flags" field within the TCP header. Here we can see all of the TCP flags broken down. Note that the SYN flag is on (set to 1).



```
▷ Frame 1 (74 bytes on wire, 74 bytes captured)
▷ Ethernet II, Src: AsustekC_b3:01:84 (00:1d:60:b3:01:84), Dst: Actionte_2f:47:87 (0
▷ Internet Protocol, Src: 192.168.1.2 (192.168.1.2), Dst: 174.143.213.184 (174.143.2
▽ Transmission Control Protocol, Src Port: 54841 (54841), Dst Port: http (80), Seq: 
    Source port: 54841 (54841)
    Destination port: http (80)
    [Stream index: 0]
    Sequence number: 0    (relative sequence number)
    Header length: 40 bytes
  ▽ Flags: 0x02 (SYN)
      0... .... = Congestion Window Reduced (CWR): Not set
      .0.. .... = ECN-Echo: Not set
      ..0. .... = Urgent: Not set
      ...0 .... = Acknowledgement: Not set
      .... 0... = Push: Not set
      .... .0.. = Reset: Not set
    ▷ .... ..1. = Syn: Set
      .... ...0 = Fin: Not set
    Window size: 5840
  ▷ Checksum: 0x85f0 [validation disabled]
  ▷ Options: (20 bytes)

0000  00 26 62 2f 47 87 00 1d  60 b3 01 84 08 00 45 00   .&b/G... `.....E.
0010  00 3c 47 65 40 00 40 06  ad 64 c0 a8 01 02 ae 8f   .<Ge@.@. .d......
0020  d5 b8 d6 39 00 50 f6 1c  6c be 00 00 00 00 a0 02   ...9.P.. l.......
0030  16 d0 85 f0 00 00 02 04  05 b4 04 02 08 0a 00 0d   ........ ........
0040  2b db 00 00 00 00 01 03  03 07                     +....... ..
```

Now do the same for packet #2. Notice that it has two flags set: ACK to acknowledge the receipt of the client's SYN packet, and SYN to indicate that the server also wishes to establish a TCP connection.



```
▷ Frame 2 (74 bytes on wire, 74 bytes captured)
▷ Ethernet II, Src: Actionte_2f:47:87 (00:26:62:2f:47:87), Dst: AsustekC_b3:01:84
▷ Internet Protocol, Src: 174.143.213.184 (174.143.213.184), Dst: 192.168.1.2 (192
▽ Transmission Control Protocol, Src Port: http (80), Dst Port: 54841 (54841), Seq
    Source port: http (80)
    Destination port: 54841 (54841)
    [Stream index: 0]
    Sequence number: 0    (relative sequence number)
    Acknowledgement number: 1    (relative ack number)
    Header length: 40 bytes
  ▽ Flags: 0x12 (SYN, ACK)
      0... .... = Congestion Window Reduced (CWR): Not set
      .0.. .... = ECN-Echo: Not set
      ..0. .... = Urgent: Not set
      ...1 .... = Acknowledgement: Set
      .... 0... = Push: Not set
      .... .0.. = Reset: Not set
    ▷ .... ..1. = Syn: Set
      .... ...0 = Fin: Not set
    Window size: 5792
  ▷ Checksum: 0x4ff1 [validation disabled]
  ▷ Options: (20 bytes)
  ▷ [SEQ/ACK analysis]

0020  01 02 00 50 d6 39 fa 58  9c 88 f6 1c 6c bf a0 12   ...P.9.X ....l...
0030  16 a0 4f f1 00 00 02 04  05 b4 04 02 08 0a 12 cc   ..O..... ........
0040  8c 71 00 0d 2b db 01 03  03 06                     .q..+... ..
```

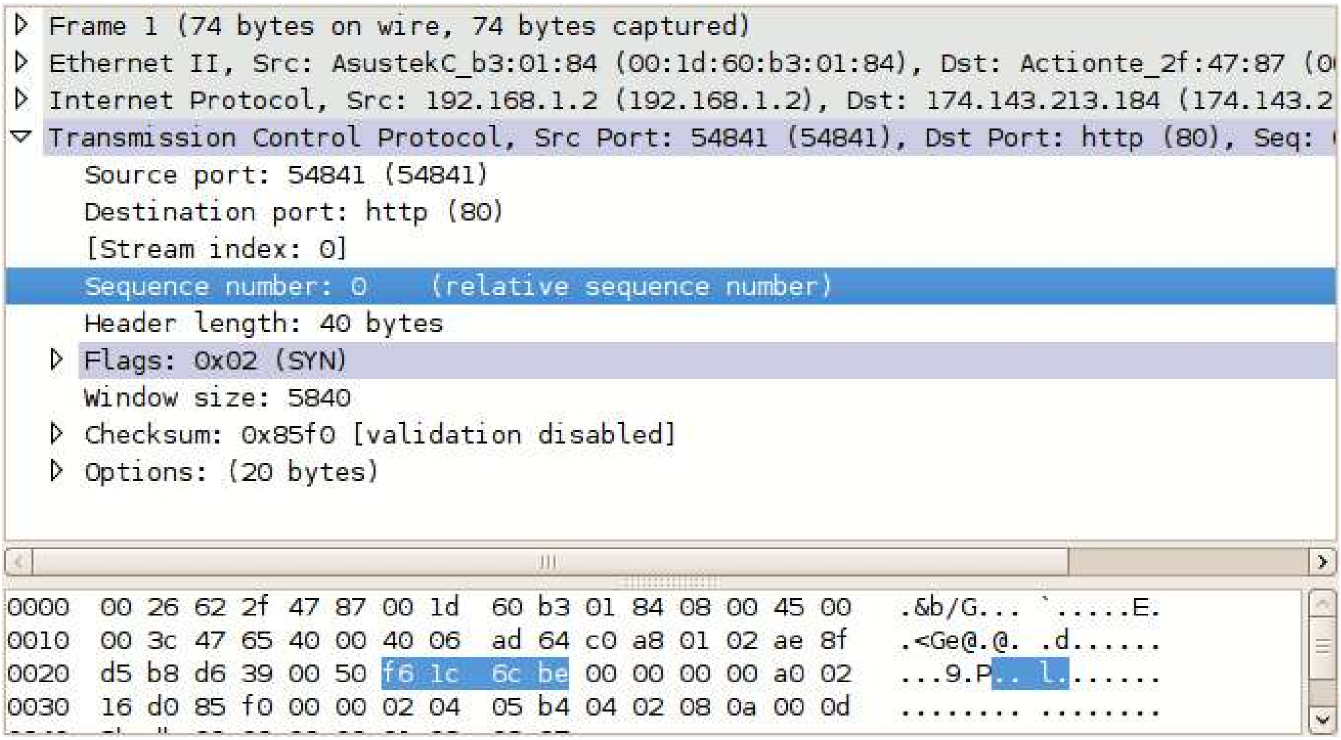
Packet #3, from the client, has only the ACK flag set. These three packets complete the initial TCP three-way handshake.

Sequence and Acknowledgment Numbers

The client on either side of a TCP session maintains a 32-bit *sequence number* it uses to keep track of how much data it has sent. This sequence number is included on each transmitted packet, and acknowledged by the opposite host as an *acknowledgement number* to inform the sending host that the transmitted data was received successfully.

When a host initiates a TCP session, its initial sequence number is effectively random; it may be any value between 0 and 4,294,967,295, inclusive. However, protocol analyzers like Wireshark will typically display *relative* sequence and acknowledgement numbers in place of the actual values. These numbers are relative to the initial sequence number of that stream. This is handy, as it is much easier to keep track of relatively small, predictable numbers rather than the actual numbers sent on the wire.

For example, the initial relative sequence number shown in packet #1 is 0 (naturally), while the ASCII decode in the third pane shows that the actual sequence number is 0xf61c6cbe, or 4129057982 decimal.

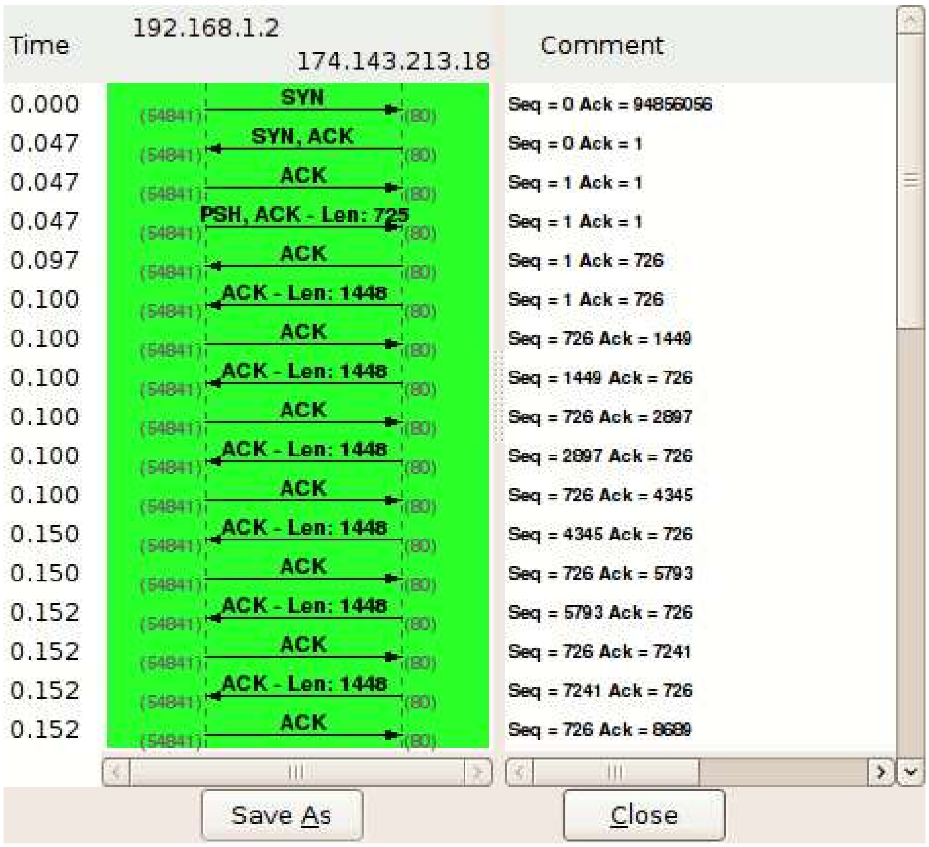


```
▷ Frame 1 (74 bytes on wire, 74 bytes captured)
▷ Ethernet II, Src: AsustekC_b3:01:84 (00:1d:60:b3:01:84), Dst: Actionte_2f:47:87 (0
▷ Internet Protocol, Src: 192.168.1.2 (192.168.1.2), Dst: 174.143.213.184 (174.143.2
▽ Transmission Control Protocol, Src Port: 54841 (54841), Dst Port: http (80), Seq: 
    Source port: 54841 (54841)
    Destination port: http (80)
    [Stream index: 0]
    Sequence number: 0    (relative sequence number)
    Header length: 40 bytes
  ▷ Flags: 0x02 (SYN)
    Window size: 5840
  ▷ Checksum: 0x85f0 [validation disabled]
  ▷ Options: (20 bytes)

0000  00 26 62 2f 47 87 00 1d  60 b3 01 84 08 00 45 00   .&b/G... `.....E.
0010  00 3c 47 65 40 00 40 06  ad 64 c0 a8 01 02 ae 8f   .<Ge@.@. .d......
0020  d5 b8 d6 39 00 50 f6 1c  6c be 00 00 00 00 a0 02   ...9.P.. l.......
0030  16 d0 85 f0 00 00 02 04  05 b4 04 02 08 0a 00 0d   ........ ........
```

The display of relative sequence numbers can optionally be disabled by navigating to **Edit > Preferences...** and un-checking **Relative sequence numbers and window scaling** under TCP protocol preferences. However, be aware that the remainder of this article will reference relative sequence and acknowledgement numbers only.

To better understand how sequence and acknowledgement numbers are used throughout the duration of a TCP session, we can utilize Wireshark's built-in flow graphing ability. Navigate to **Statistics > Flow Graph...**, select **TCP flow** and click **OK**. Wireshark automatically builds a graphical summary of the TCP flow.



```
Time    192.168.1.2                            Comment
                   174.143.213.18
0.000  (54841) ---- SYN ----------------> (80)  Seq = 0 Ack = 94856056
0.047  (54841) <--- SYN, ACK ------------ (80)  Seq = 0 Ack = 1
0.047  (54841) ---- ACK ----------------> (80)  Seq = 1 Ack = 1
0.047  (54841) ---- PSH, ACK - Len: 725 > (80)  Seq = 1 Ack = 1
0.097  (54841) <--- ACK ----------------- (80)  Seq = 1 Ack = 726
0.100  (54841) <--- ACK - Len: 1448 ----- (80)  Seq = 1 Ack = 726
0.100  (54841) ---- ACK ----------------> (80)  Seq = 726 Ack = 1449
0.100  (54841) <--- ACK - Len: 1448 ----- (80)  Seq = 1449 Ack = 726
0.100  (54841) ---- ACK ----------------> (80)  Seq = 726 Ack = 2897
0.100  (54841) <--- ACK - Len: 1448 ----- (80)  Seq = 2897 Ack = 726
0.100  (54841) ---- ACK ----------------> (80)  Seq = 726 Ack = 4345
0.150  (54841) <--- ACK - Len: 1448 ----- (80)  Seq = 4345 Ack = 726
0.150  (54841) ---- ACK ----------------> (80)  Seq = 726 Ack = 5793
0.152  (54841) <--- ACK - Len: 1448 ----- (80)  Seq = 5793 Ack = 726
0.152  (54841) ---- ACK ----------------> (80)  Seq = 726 Ack = 7241
0.152  (54841) <--- ACK - Len: 1448 ----- (80)  Seq = 7241 Ack = 726
0.152  (54841) ---- ACK ----------------> (80)  Seq = 726 Ack = 8689
```

Each row represents a single TCP packet. The left column indicates the direction of the packet, TCP ports, segment length, and the flag(s) set. The column at right lists the relative sequence and acknowledgement numbers in decimal. Selecting a row in this column also highlights the corresponding packet in the main window.

We can use this flow graph to better understand how sequence and acknowledgement numbers work.

Packet #1

Each side of a TCP session starts out with a (relative) sequence number of zero. Likewise, the acknowledgement number is also zero, as there is not yet a complementary side of the conversation to acknowledge.

(Note: The version of Wireshark used for this demonstration, 1.2.7, shows the acknowledgement number as an apparently random number. This believed to be a software bug; the initial acknowledgement number of a session should always be zero, as you can see from inspecting the hex dump of the packet.)

Packet #2

The server responds to the client with a sequence number of zero, as this is its first packet in this TCP session, and a relative acknowledgement number of 1. The acknowledgement number is set to 1 to indicate the receipt of the client's SYN flag in packet #1.

Notice that the acknowledgement number has been increased by 1 although no payload data has yet been sent by the client. This is because the presence of the SYN or FIN flag in a received packet triggers an increase of 1 in the sequence. (This does not interfere with the accounting of payload data, because packets with the SYN or FIN flag set do not carry a payload.)

Packet #3

Like in packet #2, the client responds to the server's sequence number of zero with an acknowledgement number of 1. The client includes its own sequence number of 1 (incremented from zero because of the SYN).

At this point, the sequence number for both hosts is 1. This initial increment of 1 on both hosts' sequence numbers occurs during the establishment of all TCP sessions.

Packet #4

This is the first packet in the stream which carries an actual payload (specifically, the client's HTTP request). The sequence number is left at 1, since no data has been transmitted since the last packet in this stream. The acknowledgement number is also left at 1, since no data has been received from the server, either.

Note that this packet's payload is 725 bytes in length.

Packet #5

This packet is sent by the server solely to acknowledge the data sent by the client in packet #4 while upper layers process the HTTP request. Notice that the acknowledgement number has increased by 725 (the length of the payload in packet #4) to 726; e.g., "I have received 726 bytes so far." The server's sequence number remains at 1.

Packet #6

This packet marks the beginning of the server's HTTP response. Its sequence number is still 1, since none of its packets prior to this one have carried a payload. This packet carries a payload of 1448 bytes.

Packet #7

The sequence number of the client has been increased to 726 because of the last packet it sent. Having received 1448 bytes of data from the server, the client increases its acknowledgement number from 1 to 1449.

For the majority of the capture, we will see this cycle repeat. The client's sequence number will remain steady at 726, because it has no data to transmit beyond the initial 725 byte request. The server's sequence number, in contrast, continues to grow as it sends more segments of the HTTP response.

Tear-down

Packet #38

After acknowledging the last segment of data from the server, the client processes the HTTP response as a whole and decides no further communication is needed. Packet #38 is sent by the client with the FIN flag set. Its acknowledgement number remains the same as in the prior packet (#37).

Packet #39

The server acknowledges the client's desire to terminate the connection by increasing the acknowledgement number by one (similar to what was done in packet #2 to acknowledge the SYN flag) and setting the FIN flag as well.

Packet #40

The client sends its final sequence number of 727, and acknowledges the server's FIN packet by incrementing the acknowledgement number by 1 to 22952.

At this point, both hosts have terminated the session and can release the software resources dedicated to its maintenance.

Posted in Packet Analysis (/blog/category/packet-analysis/)

Comments

Cd-MaN (http://hype-free.blogspot.com/)

June 7, 2010 at 4:39 a.m. UTC

Fun fact: the 3-way hanshake can be in fact a four way handshake - see here (although I don't believe that it is implemented like that anywhere):

http://www.breakingpointsystems.com/community/blog/tcp-portals-the-handshakes-a-lie/ (http://www.breakingpointsystems.com/community/blog/tcp-portals-the-handshakes-a-lie/)

eaadams

June 7, 2010 at 8:38 a.m. UTC

Great stuff Jeremy! Excellent explanation of TCP operation PLUS how to use Wireshark features to examine and learn about how a protocol works. Expect a flood of hits from my students!

Aubrey