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 packet analyzer.

Before we start, be sure to open the example capture 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 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.

The Three-Way Handshake

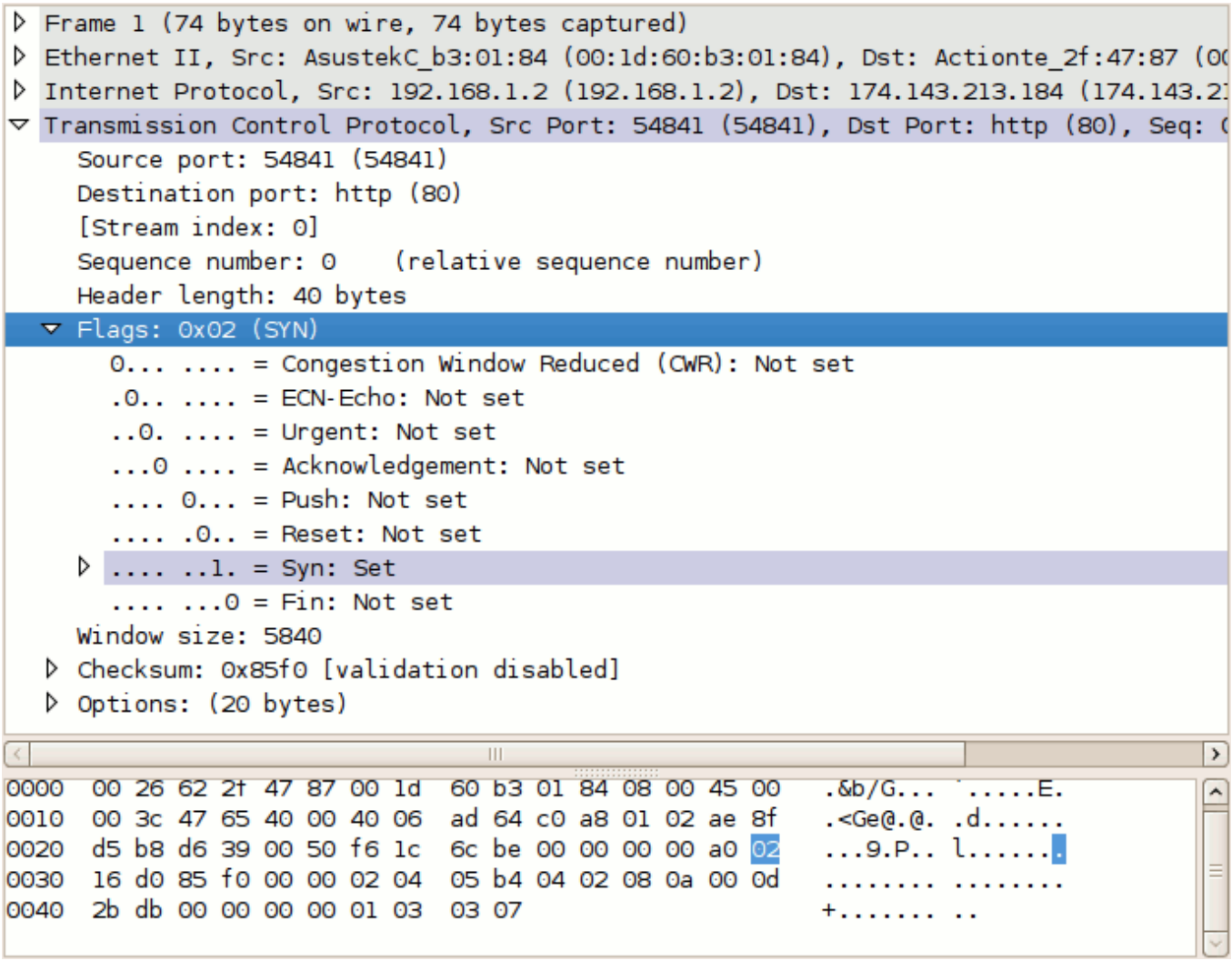
TCP utilizes a number of flags, or 1-bit boolean fields, in its header to control the state of a connection. The three we're most interested in here are:

- **SYN** - (Synchronize) Initiates a connection

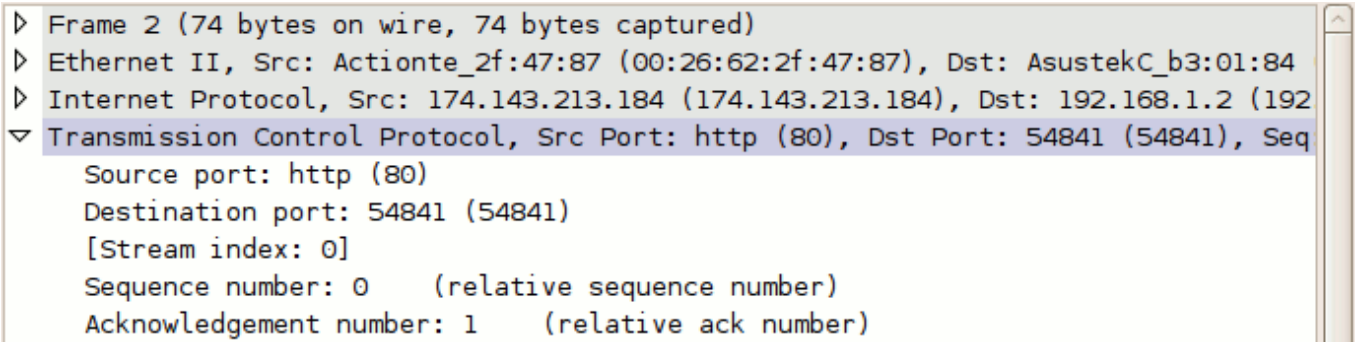
- **FIN** - (Final) Cleanly terminates a connection
- **ACK** - Acknowledges received data

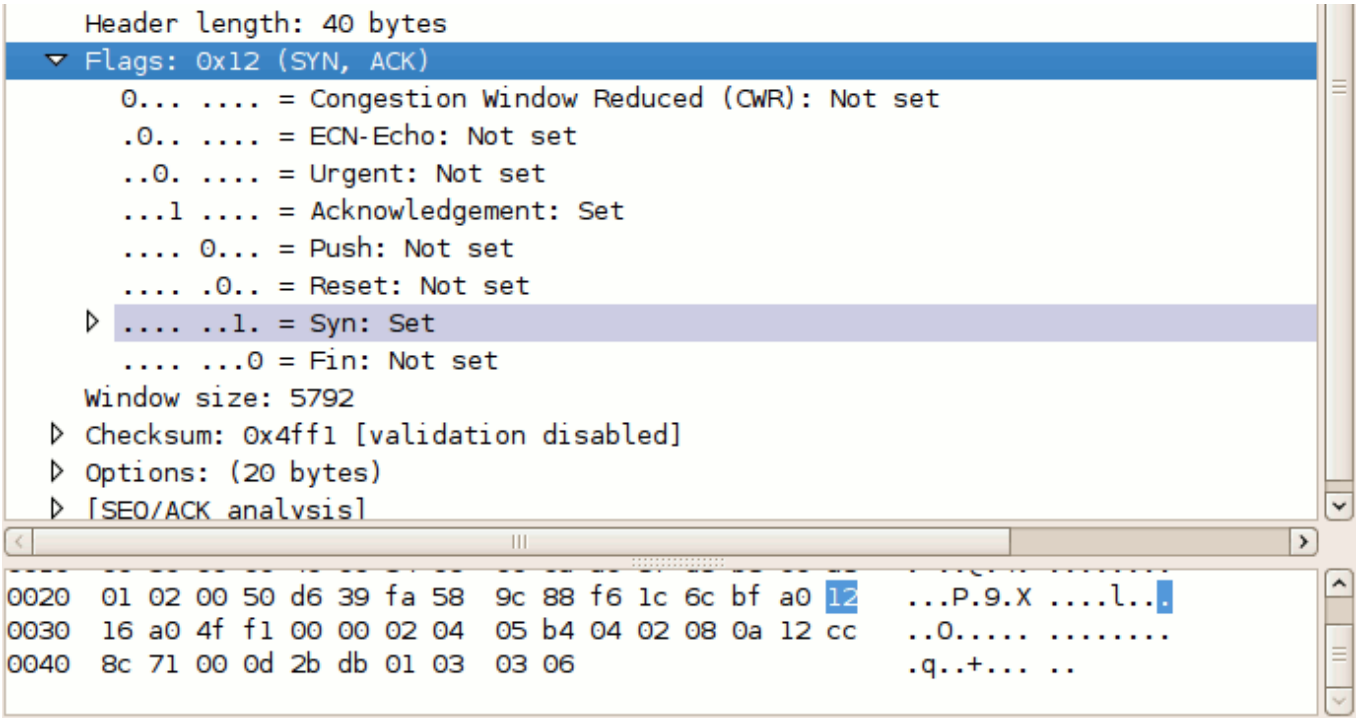
As we'll see, a packet can have multiple flags set.

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).



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.





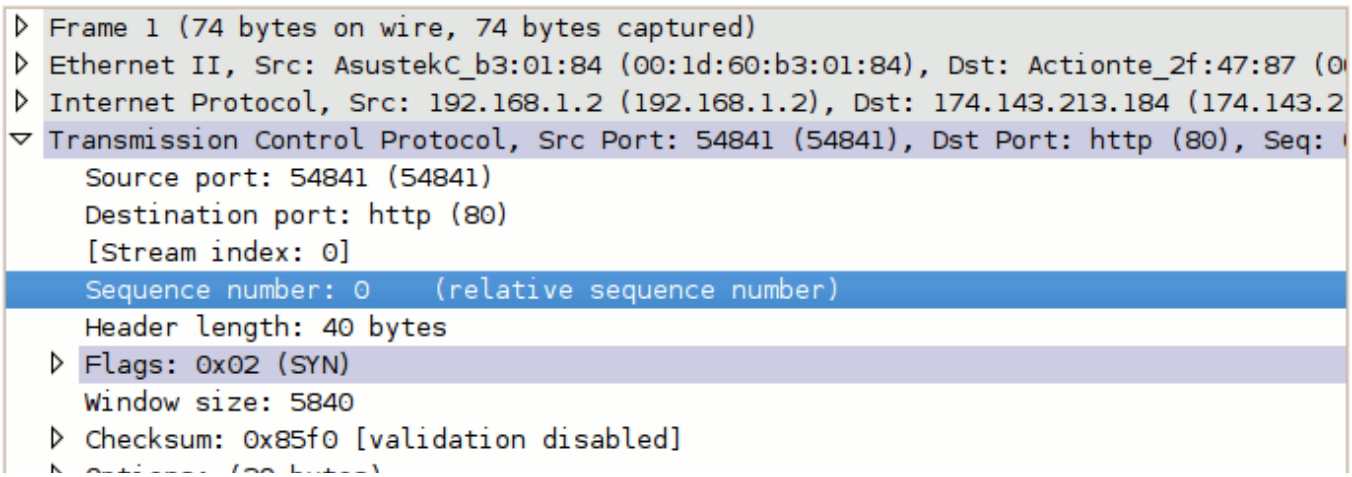
Packet #3, from the client, has only the ACK flag set. These three packets complete the initial TCP three-way handshake.

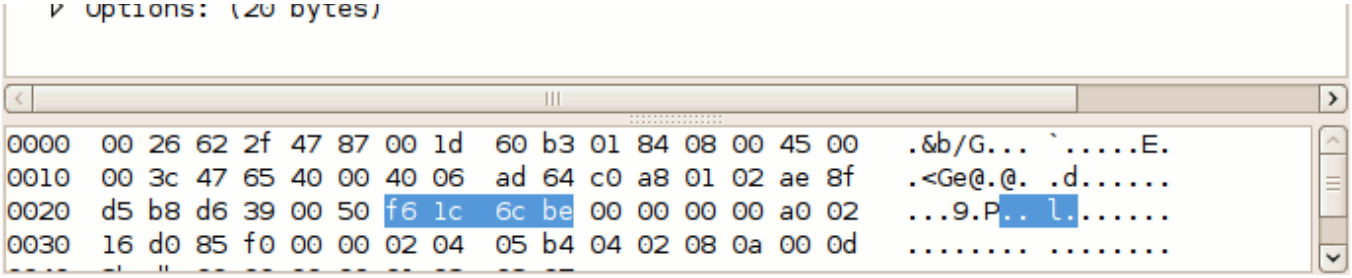
Sequence and Acknowledgment Numbers

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 number in place of the field's actual value. These values 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.

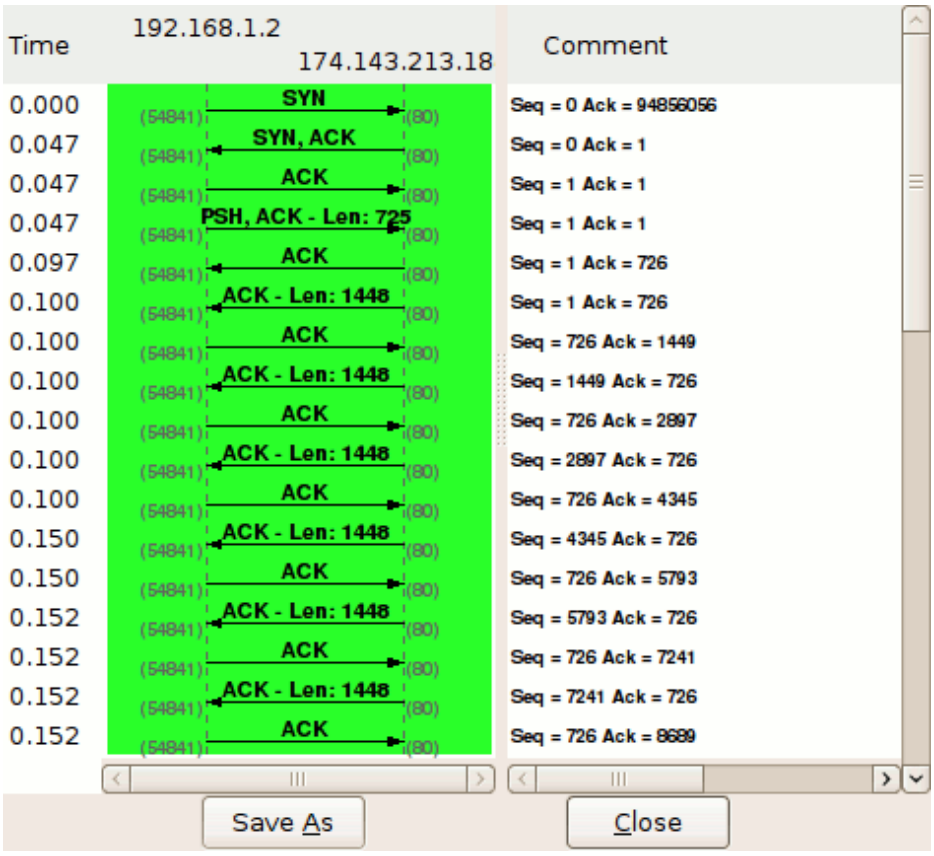
For example, the initial relative sequence number shown in packet #1 is 0 (naturally), while the ASCII decode in the third pane shows that the actual sequence number is 0xf61c6cbe, or 4129057982 decimal.





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.



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).

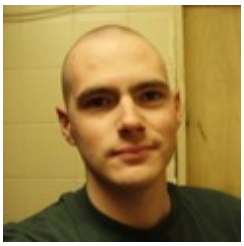
Packet #39

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.



About the Author

Jeremy Stretch is a freelance networking engineer, instructor, and the maintainer of PacketLife.net. He currently lives in Fairfax, Virginia, on the edge of the Washington, DC metro area. Although primarily an R&S guy, he likes to get into everything, and runs a free network training lab out of his basement for fun. You can contact him by email or follow him on Twitter.

Posted in Packet Analysis

Comments

Cd-MaN (guest) commented on Monday, 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/

eaadams commented on Monday, 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

Mimo (guest) commented on Monday, June 7, 2010 at 3:27 p.m. UTC

Hi Jeremy,

Great article.

When I studied TCP first, I was thought that ack no. is the next byte receiver is expecting. i.e if ack no. is 726 that means, receiver received 725 bytes of data and sender can send data starting from 726th byte.

So to me this sentence doesn't make much sense:

"Notice that the acknowledgment 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."

As it seems to me that is is expecting data stream starting from byte 1. So Ack no. is not increased.

Though the meaning is same.

PS: Thank you for the demo of ability of wireshark. I still don't know much abt this tool.

stretch commented on Monday, June 7, 2010 at 5:02 p.m. UTC

@Mimo: In this instance, as is typical with network protocols in general, our counter is *zero-indexed*; the first byte is byte 0, not byte 1. the SYN flag increases the counter by one to 1, which is actually the second possible value.

Hope that helps.

yohan900 (guest) commented on Tuesday, June 8, 2010 at 12:13 a.m. UTC

Mimo, you are correct. Do a search on google for TCP phantom byte and that will explain why there is an increment of 1 in the setup and

teardown processes.

Also check out Laura Chappells WireShark training, good stuff.

http://support.novell.com/techcenter/articles/nc2001_05e.html

Note. During the TCP startup and teardown sequence, a "phantom byte" causes the sequence number and acknowledgment number fields to increment by 1 even though no data is exchanged. This phantom byte can be confusing when you have just learned that the sequence number field increments only when data is sent.

tormentum (guest) commented on Tuesday, June 8, 2010 at 12:51 a.m. UTC

G'day,

Great post! I've just recently started down the road to CCNA having always been interested in networking. I've used wireshark in the past for very basic troubleshooting, but this is a nice deep (comparitivley speaking!) look at TCP.

One question: packet 36 contains a PSH or push flag. What is this in relation to the HTTP stream?

Cheers,
Adam.

Mimo (guest) commented on Tuesday, June 8, 2010 at 3:00 p.m. UTC

Thank you yohan900 and stretch for the information!!

@ tormentum
http://support.microsoft.com/kb/169292
If the PUSH bit set, data will be delivered up to the application right away, but the ack may still be delayed.

Hope this helps....

Daren (guest) commented on Wednesday, June 9, 2010 at 7:49 a.m. UTC

What a bit of luck!!! I have a second-stage telephone interview today and one of the subjects I will be tested on is TCP. The first interview covered it in a little detail (3-way handshake / sliding window) but this one today will delve further into it. This will help enormously - thank

you very much Jeremy.

Ned (guest) commented on Wednesday, June 9, 2010 at 7:01 p.m. UTC

Hi Jeremy, Nice Article. Thx for posting. If you get a chance can you continue on in this series and dive deeper into TCP with regards to the various algorithms like slow start and other concepts like windowing, congestion control, PAWS, Reno, SACKs etc. Tx

AaronDhiman commented on Wednesday, June 9, 2010 at 7:21 p.m. UTC

Awesome Article!

Dano (guest) commented on Thursday, June 10, 2010 at 6:18 p.m. UTC

It would have been much better to call this a "Three-step handshake" or "Three-phase handshake". To me, "Three-way handshake" implies that here are three different parties shaking hands when in fact it is a method that requires three steps for two parties to communicate.

Just an observation . .

ecbanks commented on Thursday, June 17, 2010 at 1:38 p.m. UTC

Succinct explanation. Helped straighten this out in my head.

raylook (guest) commented on Thursday, July 1, 2010 at 3:21 p.m. UTC

great article , thanks ;)

Angelo (guest) commented on Saturday, November 20, 2010 at 6:39 p.m. UTC

Hi, So if client is sending a data over different TCP segments the acknowledge number on client side will not increase untill all the data

has been sent over..???

I have a client which sends data to server and when I capture it I see the data is not in single IP datagram but I see that the whole data sent to server spans over multiple TCP segments and in Two IP datagrams.

My questions is if I need to collect all the data sent by client to server how do I reassemble them at TCP lever. I am capturing IP packets which encapsulates TCP....

Thanks,
Angelo

natraj (guest) commented on Thursday, December 9, 2010 at 12:34 p.m. UTC

Hi,
This is good stuff, but why didn't u talk about packet#36 and packet#37? If those are also added to this article, then this will be more useful for beginners.

Thanks,

Natraj.

Wajih (guest) commented on Monday, May 30, 2011 at 5:05 p.m. UTC

Thanks indeed. God bless you!

Taki (guest) commented on Saturday, June 4, 2011 at 6:19 p.m. UTC

Good tutorial, many thanks for it.

Why does wireshark keep displaying len=0 for each tcp segment ?

Himanshu (guest) commented on Wednesday, June 8, 2011 at 8:10 p.m. UTC

Itz really a nice work .
Good help for beginners.

matti (guest) commented on Friday, July 29, 2011 at 8:20 a.m. UTC

A good explanation on TCP basics. Your decision to use relative numbers is clear, but I always tend to stick with actual numbers in the 3-way handshake part. This is to emphasize my students that both ends use their own, independent seq numbers that gets increased while data flow goes on. It's easy to mix them when both start from zero, as is the case with wireshark using relative seq numbers.

Keep up the good work!

Vijay (guest) commented on Tuesday, October 11, 2011 at 5:05 p.m. UTC

Great Explanation !!! It cleared most of the doubts that I had.

Earlier my assumption was the sequence number would be sequential and I did not know that Acknowledgement was related to the amount of bytes transferred. But you made it clear. Thanks a lot.

mkriesten (guest) commented on Thursday, October 27, 2011 at 11:18 a.m. UTC

Excellent explanation. This article is a wonderful piece for educational purposes. Stumbled upon it yesterday while talking about TCP with one of my younger colleagues.

jrreyes (guest) commented on Tuesday, November 29, 2011 at 11:37 p.m. UTC

Very clearly explained. Thank you !!!

leffe (guest) commented on Friday, December 9, 2011 at 9:06 p.m. UTC

Thank you for this great explanation!

A guest commented on Tuesday, December 20, 2011 at 11:20 a.m. UTC

Very good explained. Thanks a lot. Guys like you makes life easier for us.

Amit Gupta (guest) commented on Friday, January 27, 2012 at 4:27 a.m. UTC

Excellent explanation. I was just hunting for this type of explanation on the complete cycle of TCP packet. Got more interest after reading this article.

Thank you very much Jeremy.

kr105 commented on Saturday, March 24, 2012 at 8:12 p.m. UTC

Awesome, thank you!!

mani (guest) commented on Friday, April 27, 2012 at 8:00 a.m. UTC

its awesome man its great effort i really appreciate it good work it is very helpful for beginners

Zeo (guest) commented on Wednesday, May 16, 2012 at 5:31 p.m. UTC

Thank you Jeremy it is a great article!

Looking at the connection termination:

"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."

Please share your thoughts on this because I thought that the ACK and FIN can not be combined in the same segment because the the server might still be processing a transaction and not ready to close yet so it waits until it finishes processing those transactions before setting FIN flag. Thus it is a way hand shake when tearing down a connection.

Joe (guest) commented on Monday, May 21, 2012 at 7:02 p.m. UTC

Wow, I've been looking for a simple explanation like this for the last week, thank you so much!!!!

Mitchell (guest) commented on Wednesday, May 23, 2012 at 3:17 p.m. UTC

Hi, thanks for this great tutorial, have an exam in a few days and this had made things a lot clearer although realy, I didn't need to go this much is depth.

Thank you.

James Goulding (guest) commented on Tuesday, May 29, 2012 at 12:01 a.m. UTC

After a zillion sites I finally understand the concept of ACK numbers. Your step by step break down of the diagram was key for me. Note: I'm going to a very well-known University and none of their material explains this concept like you have done.

The following was what did it for me:
"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."

I realized that the server could send more than the client requested. We were asked to break down 8 packets into a diagram and explain their movements, in Wireshark, with several criteria. In ALL of their examples the server NEVER sent more data. After 3 hours I was ready to blow.

Thanks for your posting.

Leave a Comment