TCP Loses its Marbles

SATURDAY EVENING

Here we can see Simba and Minnie happily exchanging traffic. I would like to focus your attention on the TCP SEQ and TCP ACK columns.

==========

<Educational Aside>

SEQuence and ACKnowledge numbers are the scheme whereby TCP provides *reliable delivery* i.e. verifying that both sides have received all the bytes transmitted in each direction. When Simba sets its TCP SEQ field to 62849, as it does in Packet #560, it is saying "Hey Minnie, I've sent you bytes 0-62849 thus far." And when Simba sets TCP ACK to 3811, as it does in Packet #560, it is saying "Hey Minnie, I have received bytes 0-3811 from you thus far."

Notice that in Packet #561, Simba sends Minnie a 723 byte packet ... of which the TCP Payload Length is 669 bytes (the rest of the frame goes to Ethernet/IP/TCP overhead). Thus, we would expect Minnie to acknowledge receipt of these 669 bytes by setting TCP ACK to 62849 + 669 = 63518.

Sure enough, in Packet #562, Minnie responds by setting TCP ACK to 63518, thus saying to Minnie "Hey Simba, I have received bytes 0-63518 from you." The process goes both ways: in this same Packet, Minnie sends 69 bytes of TCP data. Simba responds, in Packet #563, by setting TCP ACK to 3811 + 69 = 3880, effectively saying to Minnie "Hey Minnie, I have received bytes 0-3880 from you."

In this way, the TCP stacks on both sides keep track of how many bytes they have sent, how many bytes they have received, as well as how many bytes the other side claims to have sent, and how many bytes the other side claims to have received. If these tallies ever get out of whack, then TCP contains algorithms for recovering: requesting retransmits, for example.

BTW: if we glance at the ASCII dump of Packet #563, we can see what I interpret as the application acknowledging receipt of the previous Frame. Now, I'm guessing here ... Wireshark doesn't decode HL7 (it incorrectly interprets it as 'UMA') ... but if I'm right, then HL7 is behaving in a typical application-layer way. TCP is responsible for delivering a byte stream; applications are responsible for delivering transactions of some sort ... in this case something like "Hi Minnie, yup, I acknowledge receipt of record xyz, which you just sent me." Normal behavior.

</Educational Aside>

==========

Simba and Minnie continue happily exchanging traffic. Jumping down a bit, in Packet #594, Minnie sends Simba a 69 byte TCP payload, setting its TCP SEQ number to 4087. We would expect Simba to respond with a TCP ACK of 4087 + 69 = 4156, and in fact in Packet #595 Simba does precisely this. [Packet #595 contains no TCP payload ... notice the '0' in the

TCP Len column ... it is purely an ACK ... a waste of bandwidth, to send an acknowledgement without piggy-backing any data into the frame, but hey, the 'waste' here is utterly trivial, given the hugely fat pipes we have, so let's ignore this.] In Packet #603, Simba sends 475 bytes of data (notice that it repeats the TCP ACK of 4156, repeating what it told Minnie in Packet #595, that it has received bytes 0-4156 thus far. Normal behavior. We would expect Minnie to respond by setting TCP ACK to 66160 + 475 = 66635.

But

[Ominous roll of drums, please.]

Minnie does not.

Instead, Minnie sends a TCP ACK of 1,367,156,627, in Packet #604. Whoa. This suggests that Simba has sent 1,367,156,627 - 66635 = 1,367,089,992 bytes of data, during the last millisecond or two (none of which we've seen in this trace), and that Minnie has received all those bytes. [I am deeply skeptical that Simba could have sent ~1.2GB over the last millisecond or two ... that would require ~10,000 GB Ethernet ... which doesn't exist yet (the fastest thing around these days is 100GB Ethernet ... I believe that Minnie and Simba are both running at 100Mb ... Besides, we didn't see any of these putative bytes.] We can see Wireshark struggling to interpret this frame "This frame ACKs a segment we have not seen (lost?)]" at the bottom of the screen shot.

Also, Minnie, by setting TCP SEQ to 4155 is saying that it has sent Simba bytes 0-4155. But, we know that in Packet #594, Minnie sent 69 + 4087 = 4156 bytes, i.e. through byte 4156

Now, in Packet #605, Simba resends its view of the byte stream, claiming that it has sent Minnie bytes all the way through 66635 and has received from Minnie bytes through 4156. What Simba is saying here is: "Hey Minnie, I don't need bytes 0-4155; I already have bytes 0-4156. Please send me whatever you have */after/* byte 4156."

But Minnie has a different view of the byte stream, which it repeats: "Hey Simba, I've sent you bytes 0-4155 and I've received from you bytes 0-1367156627."

And the two go on repeating this back and forth for quite a while. Recognize that? I know I've done that, in all sorts of relationships, in which I'm so focused on my needs that I'm not listening to what the other person is saying/wanting. Sigh.

In fact, Minnie and Simba go back and forth like this from 19:39:53 to Packet #231946 at 19:40:22 (~230,000 packets, i.e. ~115,000 exchanges). Until Minnie finally quits for ~an hour and a quarter ... until at 20:55:57 (Packet #231946), Minnie coughs up yet another ""Hey Simba, I've sent you bytes 0-4155 and I've received from you bytes 0-1367156627." message. Simba responds with a TCP RST, meaning (my interpretation) "Minnie, I've given up on you; go away." [I suspect that Simba has deleted this conversation from its TCP connection table.]

==> Seems to me that Minnie's TCP stack became confused at Packet #604. And then was unable to respond to internalize Simba's claim that it had already received bytes 0-4156 (by sending, say, byte 4157 and beyond).

But of course, this is complex stuff, perhaps there are other clues awaiting us. Let's look at another trace.

SUNDAY MORNING

In Packet #13, Minnie initiates the three-way TCP handshake with Simba -- SYN, SYN, ACK (Packets #13-15), and in Packet #16 Minnie sends the first HL7 frame -- 341 bytes of TCP payload. In Packet #17, Simba responds, ACKing 342 (1 + 341 = 342), as we would predict. Simba sends 85 bytes of TCP payload in Packet #18, and in Packet #19 Minnie correctly acks to 86 (1 + 85 = 86), and reaffirms that it has sent 342 bytes thus far. A few hours pass ...

[Ominous roll of drums, please.]

In Packet #1970, a few hours later, Minnie sends 348 bytes of TCP payload ... but ACKS to 341 (rather than 342, as it did in Packet #19). And, it claims to have sent 2117622181 bytes of data (i.e. it ACKs to 2117622181). Wireshark sees this as a 'TCP Retransmission', because it has already seen bytes 0-342 ... and here we have a frame which claims to be resending byte 342 all over again (In this frame, Minnie is claiming to be sending 348 bytes, starting after byte 341). In this case, Simba ignores Minnie. I'm a little surprised here – I would expect Simba to restate its understanding of the byte stream, as we saw it do Sunday evening. I don't know enough to know whether or not this is normal behavior. Could be -- could be that Simba figures, "Hey, this is a byte sequence I've already seen; I'll throw this away and wait for something new." But back on Sunday, when this happened, it spoke up ... here is it is silent. I don't know which behavior is normal (perhaps both responses are normal ... but why would Simba pick one response on Sunday and a different one on Saturday?)

So, Minnie sends this outrageous frame twelve (12) times across ~10 minutes and finally gives up in Packet #2145, sending a TCP RST (Minnie slamming the phone down). Simba responds as we would predict, by sending its understanding of the byte stream (TCP SEQ 86 and TCP ACK 342). By this time, Minnie has removed this conversation from its TCP connection table, so it responds to Simba's ACK with a TCP RST (essentially, Minnie saying "I don't know what you're talking about, go away".) This section is normal behavior, when one side has given up. What it tell us is that Simba had kept this conversation alive in its TCP connection table (otherwise, Simba would have responded to Minnie's RST with its own RST, rather than with an ACK). Continuing through this trace, we see the same experience repeated in the second half of the screen shot.

TUESDAY AFTERNOON

And on Tuesday afternoon, we see the same pattern we saw on Sunday morning: after a while (a short while), Minnie's TCP stack seems to get confused about where the conversation is, on both byte streams.

<Educational Aside>
This, BTW, is a wonderful illustration of what the TCP weenies mean when they say that TCP provides *reliable* or *guaranteed* delivery. Of course, they aren't claiming that they can *guarantee* delivery of the data -- one cannot promise perfection. What they are saying is that they can guarantee that *either* TCP will deliver the data *or* that TCP will tell the application that it was *unable* to deliver the data ... meaning, you get a deterministic or 'reliable' result ... although perhaps not the result you wanted!
</Educational Aside>

LOOSE ENDS

In these sorts of things, I often look for a device in the middle ... a firewall, say ... fiddling with the traffic. That's why I like to include the 'IP ID' and 'TTL' columns -- typically, if a device in the middle is spoofing, those numbers will be wildly out of whack (firewalls are smart enough to spoof IP addresses and TCP port numbers, but not, in my experience, IP ID numbers or TTL.) I see no such evidence here -- TTL is constant in both directions, and IP ID numbers increment in a predictable fashion. [No surprises here: the Hutch's network inserts no such device in this particular path -- Minnie and Simba are separated by a bunch of switches and routers which do no filtering, just forwarding frames.]

The perspicacious amongst us will have noticed duplicated IP IDs in the Tuesday Afternoon trace. Packets 216857/216858 are identical ... as are Packets 216955/216956, 217229/217230, and 217319/217320. I don't have a good explanation here. I propose that this is a bug in Wireshark's capture function (the WinPcap library), that Minnie and Simba are not really duplicating packets ... this would be a truly gross bug in their IP stacks, if they were doing this. We are running an old version of Winpcap here (in fact, we're running Ethereal on Simba, not even Wireshark, don't know what ancient version of Winpcap is loaded). I recommend uninstalling Ethereal and installing the latest version of Wireshark -- that will get us the latest version of Winpcap as well.

SUMMARY

At the moment, I'm turning my attention to Minnie. Why is its TCP stack intermittently spitting out TCP SEQ and ACK numbers which don't fit the byte stream as we see it (and as Simba sees it)?

POSTSCRIPT

Contacted the vendor of the TCP/IP stack on Minnie, sent them packet traces. Front-line tech support suggested installing a patch which they had released the previous month that fixed several TCP bugs. We installed the patch; problem went away.