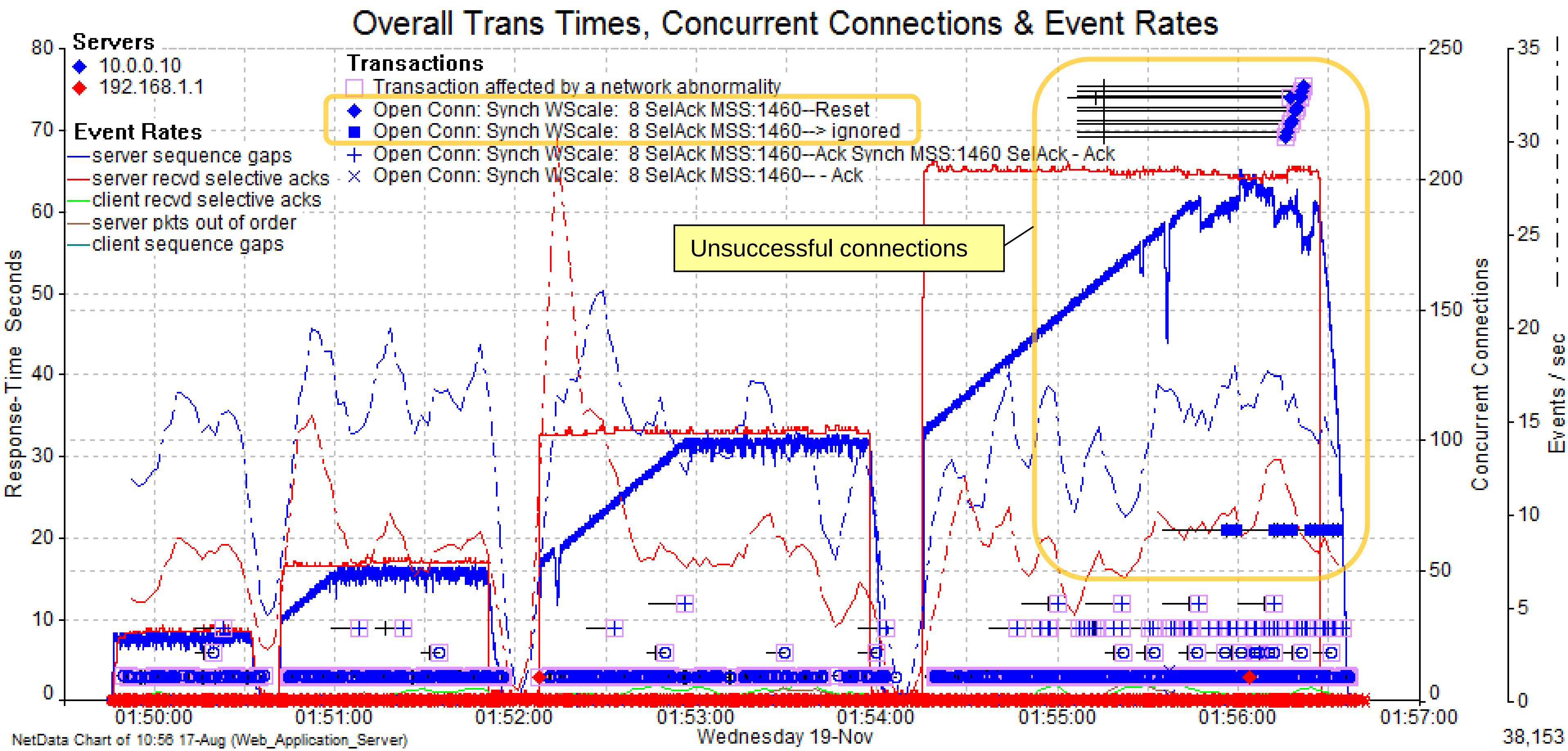
My Solution

to the SharkFest 2015 “Megalodon Challenge”



Megalodon Challenge

This is an analysis of the network and application activity involved in the "Megalodon Challenge", distributed at SharkFest 2015 by Jasper Bongertz.
https://blog.packet-foo.com/2015/07/the-megalodon-challenge/

The observations, commentary and recommendations are presented first.

The supporting evidence is provided in subsequent slides - for readers who would like to follow along with the detailed analysis.

First the answer to the question posed in the Challenge.
"At a certain point in time during the test there would be unanswered page requests, but it was unclear if it was a network problem, an application framework problem, something in the application logic itself or maybe even something else entirely."

After that, an extended analysis to examine the server, application and network bottlenecks in the two flows.
There are numerous behaviours that defy explanation. For now, all we can do is report the observations and suggest potential alternative reasons for the behaviours.

This analysis was performed by Philip Storey, a freelance network & application performance analyst and troubleshooter living in Sydney, Australia. Contact Phil at: Phil@NetworkDetective.com.au

Megalodon Challenge

This is a very interesting case study, containing many elements and application behaviours that are difficult to explain.

The tool used to perform the analysis is "NetData-Pro". Its graphical visualisation capabilities – combining many different attributes superimposed on one chart - make interesting behaviours "eye-catching". This allows problems to be identified much more quickly.

All the data can be visualised at once, even with multi-GB capture files. Each of the capture files here are about 2 GB each.

The author has been a user of NetData-Pro for around 6 years.

The capture files had been modified by "TraceWrangler" – an excellent free utility that can perform many "sanitisation" functions. Here it had substituted IP addresses, MAC addresses and replaced the payload content with the repeated string, "Payload Removed!".

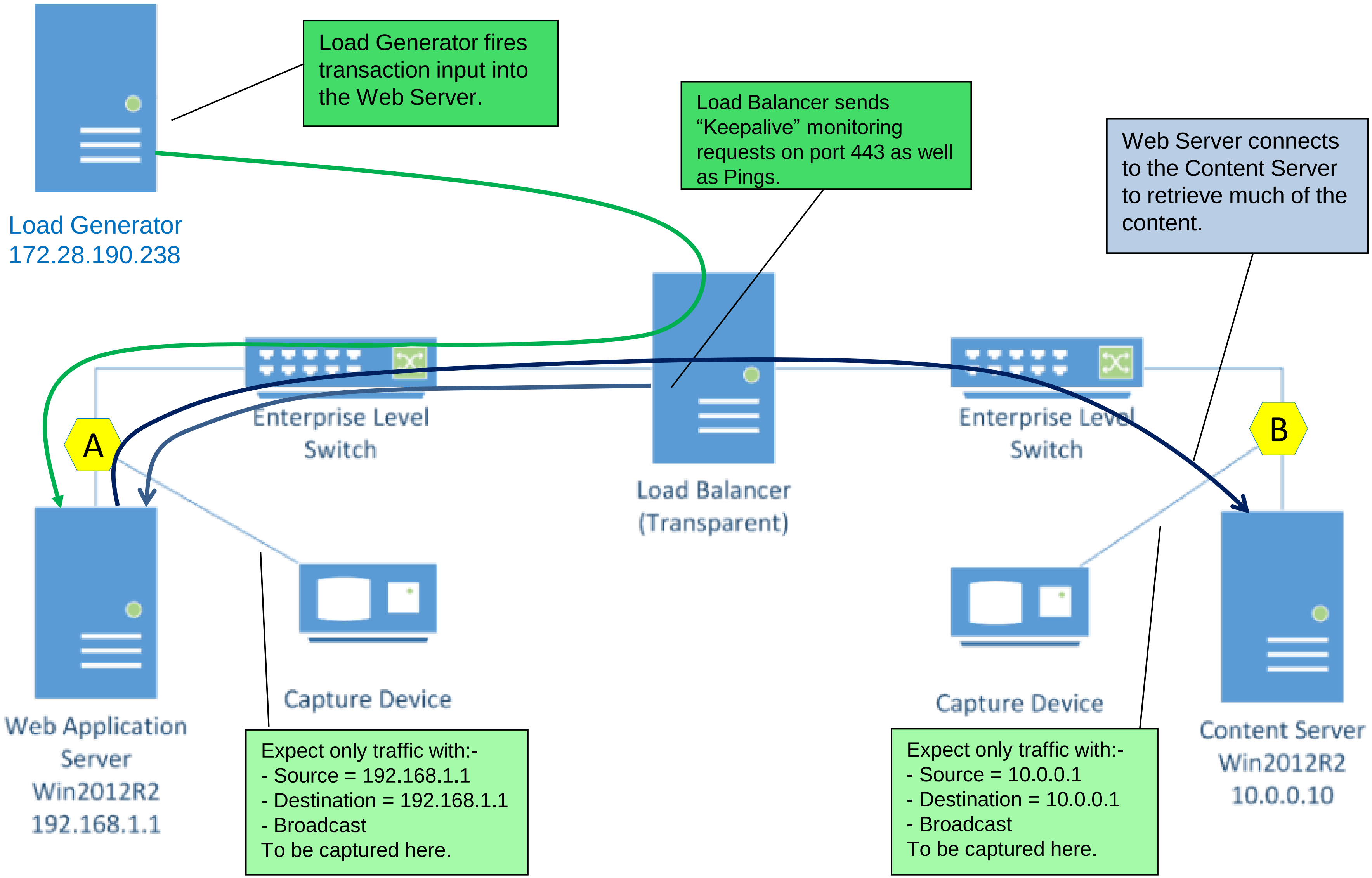
Find it at - https://www.tracewrangler.com

Even with no real application payload data, we can still observe several application behaviours that are quite intriguing.

"NetData-Pro" is a commercial product, from Measure IT Pty Ltd in Sydney (Australia).
The Principal is Bob Brownell: Bob@NetData-Pro.com

Diagram (As Provided)

Added the Load Generator and flows based on the observations. As we'll see later, it is very unlikely that Tap-B is actually where it is drawn here.



Observations

1) TraceWrangler had been used to sanitise the packets and substitute the payloads.
(This means that we couldn't use the powerful layer-7 analysis functions of NetData. We nevertheless are able to discover details about server performance by using the generic "request/response" decoder to identify individual transaction timings and categorise them by request/response sizes).

2) It appeared that there were 4 load test runs, each time "doubling" the load.
3) There are many packet losses consistently throughout the capture period (in regular time periods).
4) There are regular instances of Syn and/or Syn-Ack packets going missing.
5) All the losses (in both directions) occurred between the capture tap(s) and the Content Server.
6) There are two different types of packet loss behaviour, one consistently throughout the four test runs and another that appears only under heavier loads.
7) There are more frequent instances of Syn/Syn-Ack losses during the fourth test run.

8) The answer to the Challenge is that failed connections occurred during the fourth, biggest volume, test run.
- The Content Server terminated 11 connection requests by issuing a Reset after a minute.
- The server also "ignored" 47 connection requests, i.e., there was no response to the repeated Syns.

9) The Content Server appears to begin to suffer "stress" during the third test run and then more so during the fourth (heavier) test run.
This stress appears to be related to a limit on the server application availability to take requests off the incoming TCP stack.

Performance Observations

10) There are constant, regular periods of 5 seconds where 2.5 secs have packet losses, then 2.5 of no losses. This repeats for the whole 10 minute capture.
    - All connection setups during the "lossy" 2.5 sec periods take longer than those in the "non-lossy" 2.5 sec periods - even those that are not affected by lost packets.
    - So the same device may be responsible for both behaviours.
    - The lost data packets are of all payload or message sizes.

11) There is a different type of packet loss period during the fourth test run. Here, there are instances of Syn/Syn-Ack losses that aren't within the other "lossy" 2.5 sec periods.
    - These are the instances that cause the failed connections and transactions.
    - Their frequency of occurrence ramps up "exponentially" at the end of the fourth test run.
    - There is not an equivalent increase in data packet losses.

12) The first transaction in every connection is a 127-byte request with 3031-byte response (which has the flavour of an SSL handshake).
    - These take progressively longer during test runs (with a minimum of 0.5 secs in the fourth test run).
    - Probably because the Content Server application can't take the requests off the inbound TCP queue quickly enough - due to some form of load/stress.
    - Improving these transactions will have the biggest effect on the overall performance.

13) The second transaction in every connection is a 158-byte request with 51-byte response (which has the flavour of an SSL cypher exchange).
    - These are usually very fast, but many always take ~300 ms and ~500 ms of server "thinking" time.
    - These times are not load related – as they occur during all four tests.
    - These should be examined because a third or half a second is significant.
    - Only 128 out of 36472 return the 51 bytes in one packet. 36344 deliver 2 packets of 6+45, often with large times between those 2 packets.

Performance Observations (Cont.)

14) There are regular "gap" periods of 0.3 seconds, every 4.5 seconds or so (unrelated to the 2.5/5 sec "lossy" periods) where BOTH the Web Server and Content Server applications seem to stop communicating or responding.

- The only packets in either direction during these 0.3 sec "gaps" are TCP (Acks or retransmissions) - not application data or new connection requests.
- If it was within just one server, we might call it "garbage collection" or similar. I have no explanation for these periods to be synchronised between two different servers.

15) The failed connection requests do not appear to be related to client ephemeral port "recycling" – even though, due the relatively high rate of new connections, they are reused every 170 seconds or so. The Web Server restarts at 49155 – which is interesting because the MS Windows "standard" has been 49152 since 2008, implying that it has been slightly modified in this environment.

16) The responses from the Content Server are not delivered with maximum efficiency (i.e., as a stream of full sized packets). Rather, the flows consist of some "blocks" of good flows, but interspersed with many and various smaller packets. This occurs even in the lighter test runs.

This could be due to the application being unable to keep up in delivering data to the outgoing TCP stack. Or perhaps there is another device in the path that is causing this? Packets of size 29 bytes are very common. Because of this, NetData's generic "request/response" decoder has categorised transactions based on the packets flowing each way. The large 52 KB responses look like this for example:
Request: blk[154]; blk[1284] – (Response) blk[4xx]; blk[2x] (5); blk[1xxxx]; blk[1xxxx]; blk[2x] (5); blk[1xxxx]; blk[1xxx]; blk[2x]; blk[3x]; blk[2x]
Which means that the request was in 2 packets of [154]+[1284] and, in this case, the response totalling 52590 bytes came back in packets of [445]; 5 x [29]; a large block of [17128] {11 x [1460] + [1068]}; another [15708]; 5 x [29]; [17520]; [1404]; [29]; [37]; [29].
Thus, the thousands of common 52 KB and 983 byte responses have been categorised separately, even though they are likely to be the same. The colour groupings in the charts were created by colour coding the transactions only by their request signatures.

Performance Observations (Cont.)

17) The TCP connections from the Load Generator to the Web Server ramp up to the respective maximums very quickly (forming an almost vertical line for the "concurrent connections" chart).

However, the connections from the Web Server to the Content Server step up more slowly, about 1 extra connection per second.

The effect of this is most apparent in test run 4, where the 200 connections from the Load Generator are initiated all at once (with the respective 200 transaction requests). The 200 requests are queued up and it takes the Web Server 12-18 seconds to work its way through them all. This is because the Web Server begins with only 100 or so connections to the Content Server – hence can only handle 100 back-end transactions in parallel. It takes ~100 seconds before the connections have ramped up to match.

Could this be due the Web Server making a decision to increase its available application threads only once every second? Would the performance be improved if the Web Server initiated more connections sooner?

18) The vast majority of Load Generator transactions (into the Web Server) are a 325+ byte request with a response of 286,650 bytes.
These keep coming as fast as they can. There are no randomised gaps that might simulate real users.
The queuing effect of these transactions is much more significant in the fourth test run.

19) The packet losses in these flows all occur between the tap(s) and the Load Generator. The regular 5 second "loss" + "no loss" periods are not apparent in these flows. The losses are far fewer in number and seem more random.

Final Observation: Tap Locations

20) The location of the tap for the capture file named “Content Server” does not appear to be in the location described in the Megalodon Challenge information.

Based on the client-server flows, MAC address (even though they were “Wrangled”), TTL values and flows, the Tap-B “Content Server” location appears to have been on the same TCP segment as the “Web Server” (which I’ve called Tap-A). Tap-B captured all the same client-server flows as Tap-A.

The two capture files are very similar – and could almost be mistaken as being identical – but they did contain minor differences.

- Some packets common to both captures had ever-so-slightly timing differences.
- There are some packets that are in one capture but not the other (mostly at the beginning and end – so likely due to Wireshark start-stop timing differences).
- Where these packets are missing in either capture file, they appear to have been dropped by Wireshark rather than actually lost in the network.

To save this PPT file becoming overly large, the evidence for this is included in a separate report, “Megalodon-Challenge-Comparing-The-Two-Captures”.

The diagram on the next slide is my guess as to where Tap-B might have been located.
This is also assuming that the diagram as originally provided was accurate. It is possible that the Load Balancer is not in that exact location.

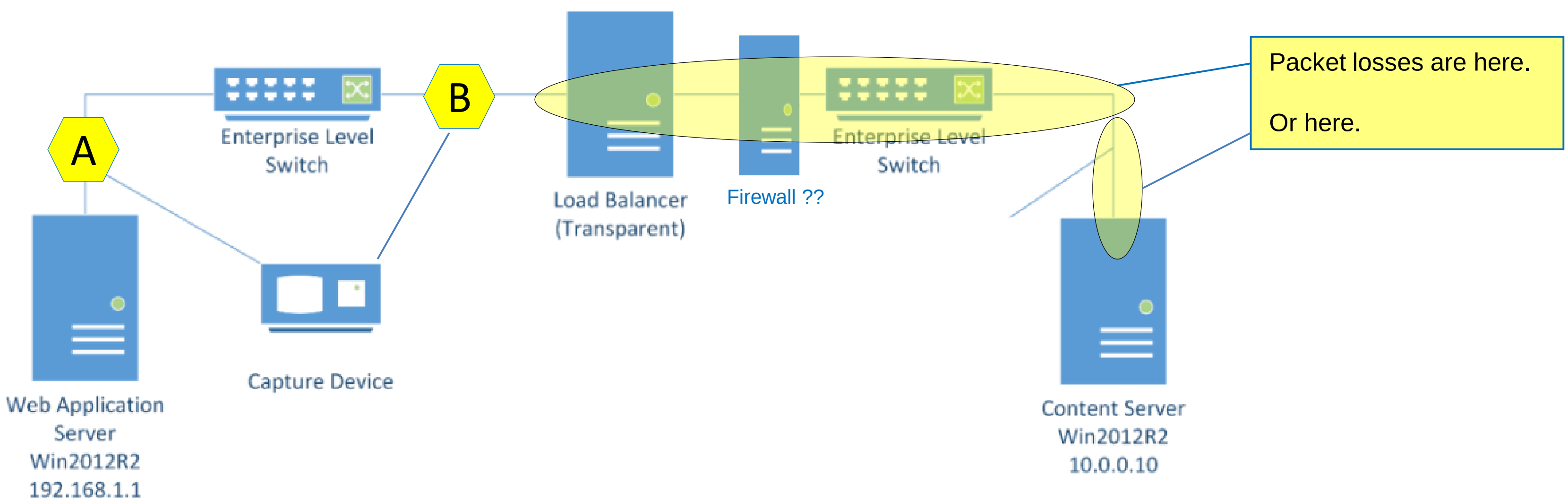
Challenge: The Answer

The 47 “Ignored” connections and 11 “Reset” connections would result in failed transactions from the Web Server. These are the cause of the observed problem as stated in the “Challenge”:

“At a certain point in time, the web application server would not get a reply for some of its requests anymore.”

Those connections terminate abnormally because the client Syn packets, server Syn-Ack packets and client Acks get lost somewhere between the capture points and the Content Server.

The evidence for this conclusion is presented in the “Evidence Trail” in the following slides.



Recommendations

1) The source of the packet losses be investigated and fixed.

Both kinds of losses need to be identified. The reported problem though, is due to the Syn/Syn-Ack losses which are load related.

Candidates for the cause of the losses could be:

- Load Balancer (if in the path from Web to Content Server).
- A firewall (if there is one in the Web to Content Server path).
- Switches (e.g., duplex mismatches).
- Faulty TCP settings in the Content Server (e.g., duplex mismatch).
- Faulty NIC card(s) in the Content Server.
- Faulty Ethernet Cables.
- Internal Content Server resource limits (since load plays a part).

2) The exact network topology be determined.

All devices in the Web Server to Content Server path need to be identified so that they can be examined so see if they play a part in the observed behaviours. Simultaneous packet captures in more locations then need to be performed in order to narrow down the “suspects” and eliminate each component as the cause of the problem behaviours.

In a standard multi-tier architecture, load balancers and firewalls are likely to be in this path.

Recommendations (Cont.)

3) That a capture be taken at the Content Server.

The capture that was supposedly taken by Tap-B at the Content Server does not appear to be correctly located. It seems to have been taken on the same VLAN/segment as the Web Server capture.

A capture here would prove whether or not the observed packet losses are due to the Content Server.

4) Increase the range of ephemeral ports used by the Web Server to connect to the Content Server.

The ports begin at 49155 and are being “recycled” in just over 3 minutes due to the large number of short running connections. If the starting number was reduced to nearer to 20000, the duration between “recycles” would stretch out to 6 minute

5) Investigate the Content Server’s application behaviour of breaking up the responses into very small packets.

For example, why would the vast majority of the Content Server’s second transaction type return a 51 byte response as two packets of [6] + [45] rather than a single packet of [51]? Finding and fixing this behaviour could have a significant impact of the overall performance of the system.

The starting point for this investigation would be to look at the application’s TCP output buffering settings.

Evidence Trail

The following slides provide the evidence of the observed behaviours.

1) First we look at all the client-server connection pairs within the packet capture files. We can get a high level view of flow volumes, retransmission rates and other statistics for both directions of each flow.

2) Then we look at all the actual connection setups and other transactions, all at once on a single chart. Here the "problem" behaviours stand out. We also get a visualisation of the whole 7 minutes where the four different test runs are clearly visible and separated. Visual correlations can be observed between the Load Generator traffic and the Web-to-Content Server traffic.

3) We then "zoom-in" to investigate the troublesome connections. We can use "Packet Timing" views to see the packet behaviours in a time-of-day setting.

4) By choosing the right chart overlays, we can then observe the packet loss behaviours and how they correlate (or don't) with other behaviours.
This is where some surprising behaviours are observed!

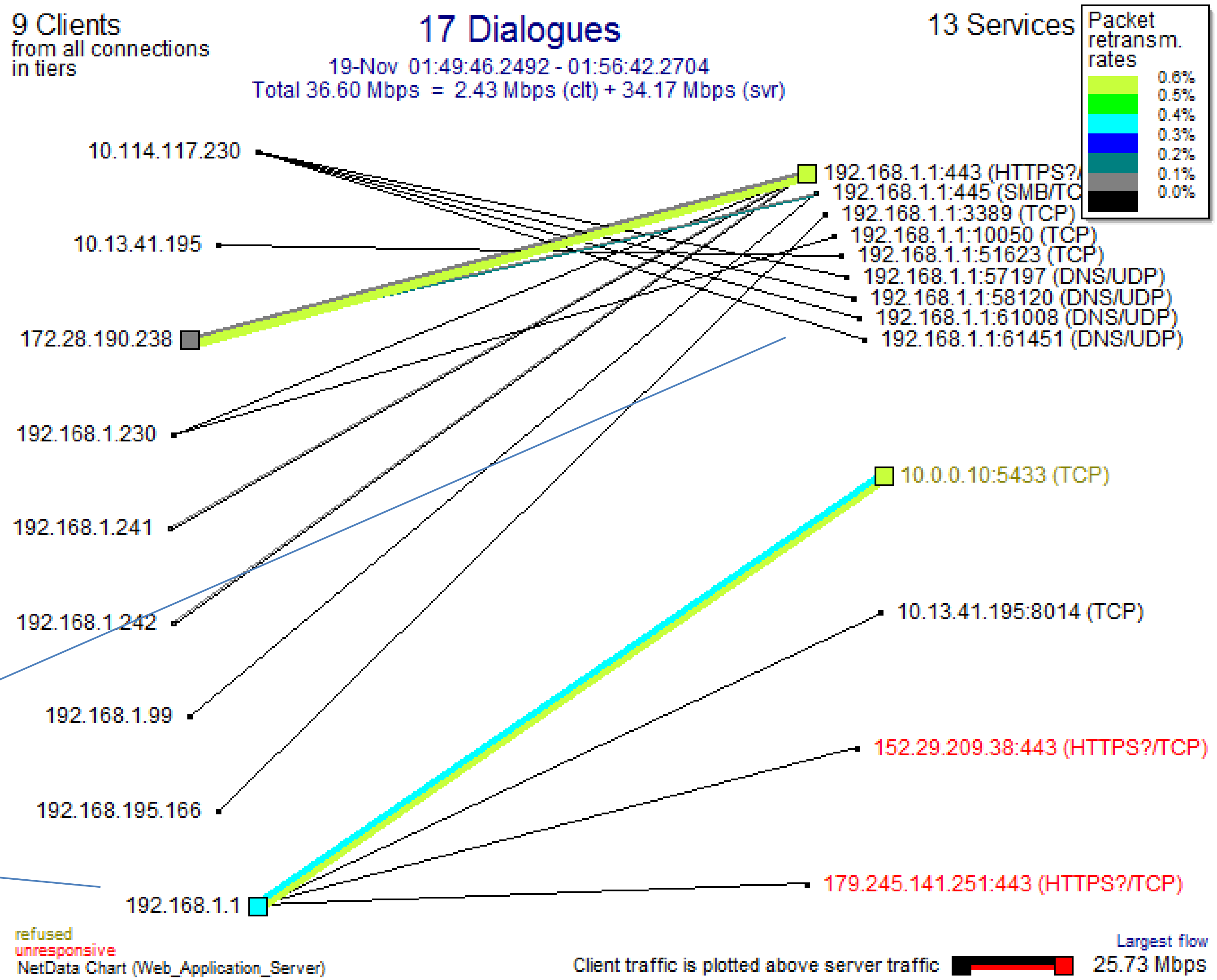
Overview - Dialogues (Web Server Capture)

This is a high level view of the clients (on the left) and servers (on the right).

Line thickness relates to traffic volumes.

Line colour indicates Packet Retransmission rates in each direction.

We clearly see the two pairs of three servers with the traffic that we are interested in.



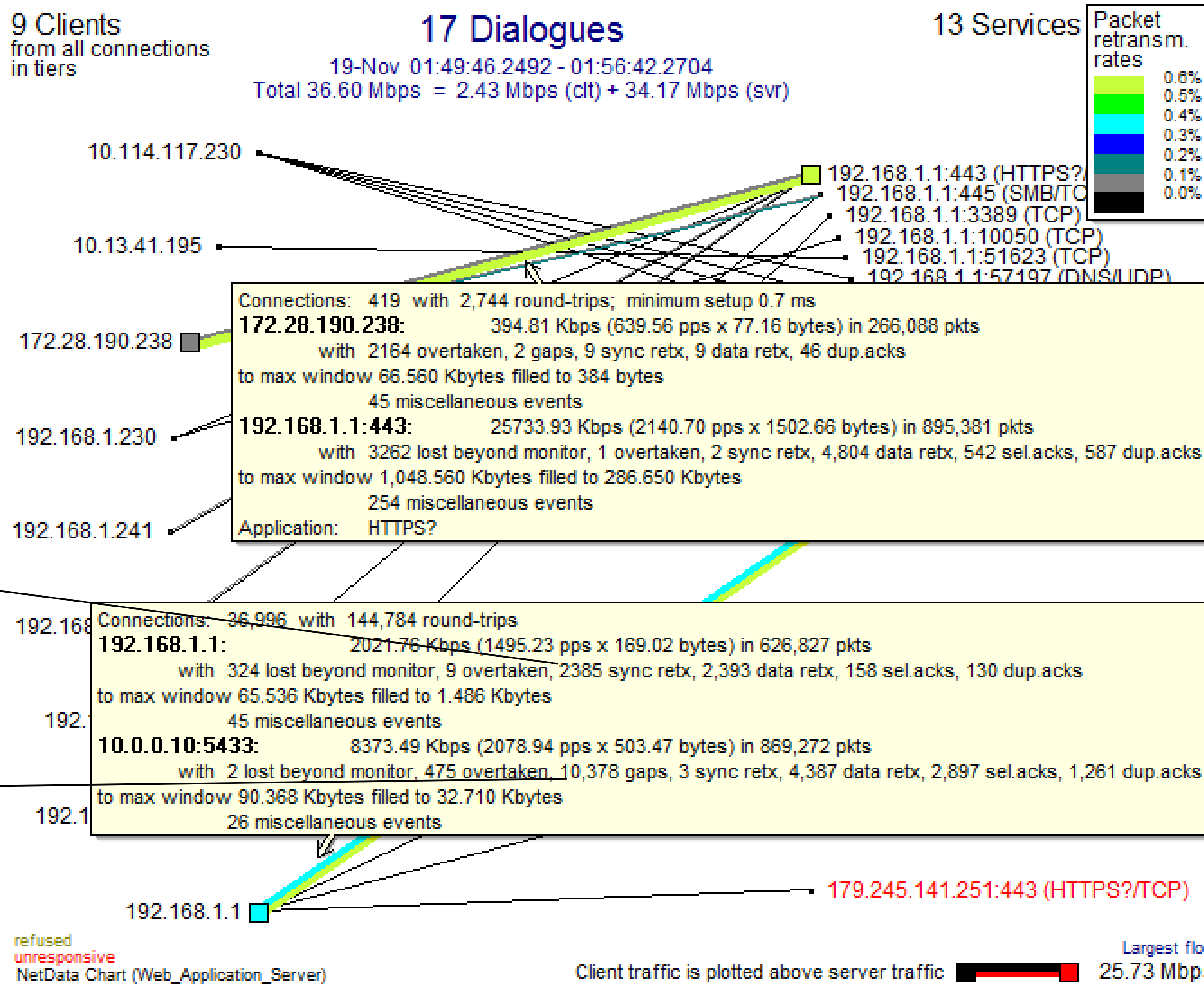
This is the Web Server acting as a server, receiving connections.

This is the Web Server acting as a client, initiating connections.

Overview - Dialogues (Web Server Capture)

Hovering on the lines produces these popups, providing more details of the traffic flows.

From this we get the idea that lost packets are likely to play a part in our “troublesome” behaviours.



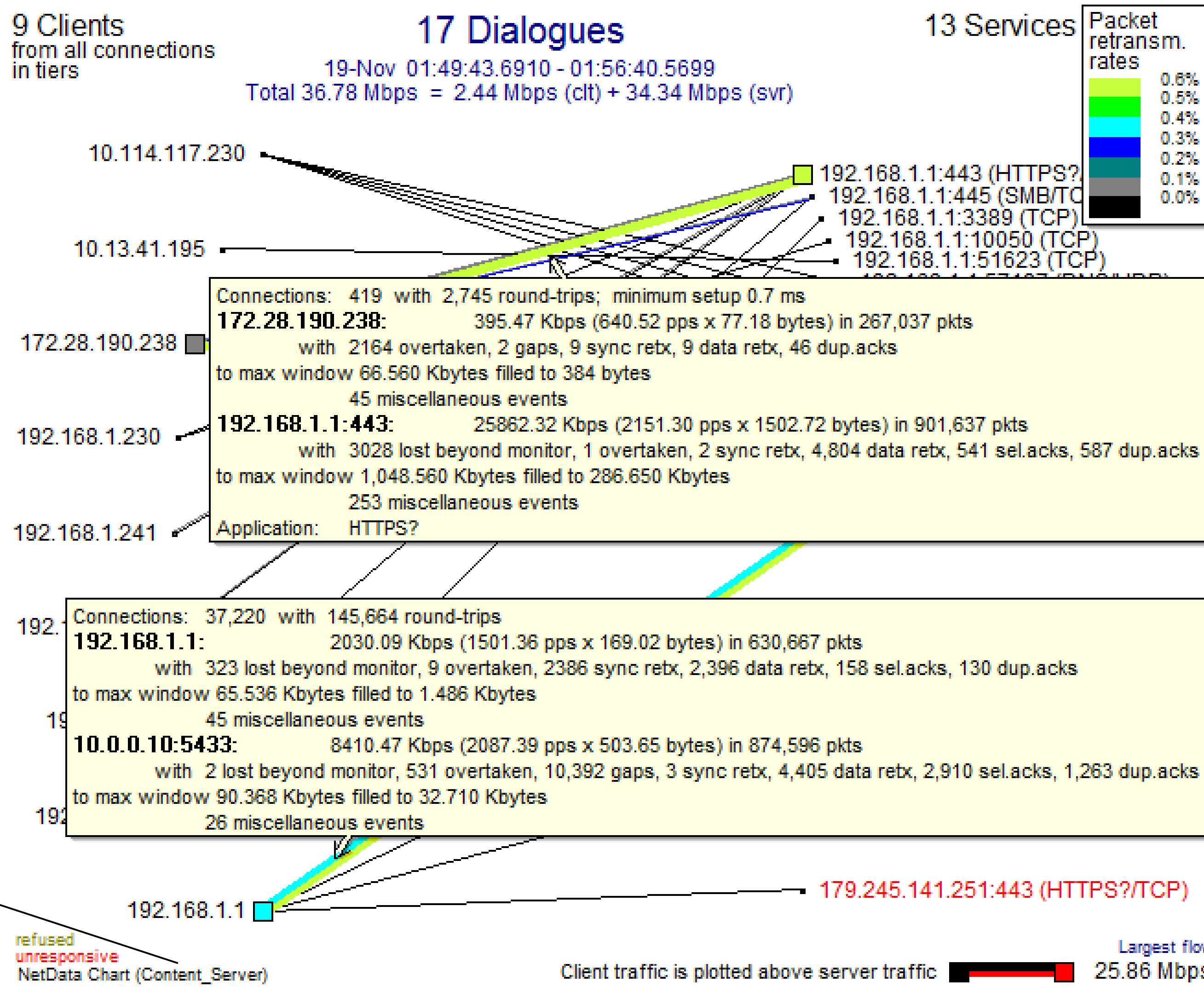
Lots of Syn & data retransmissions from the Web Server to Content Server.

Lots of “gaps” in the flow from the server (meaning packets were lost on the way back).

Overview - Dialogues (Content Server Capture)

This is the same chart but from the capture file named “Content Server”).

Given that it contains traffic from the Load Generator to the Web Server (top flow) and also given that the underlying details are very close to the previous slide, the conclusion is that this capture was not taken at the Content Server, i.e. Tap-B was located elsewhere and on the same VLAN/segment as Tap-A.



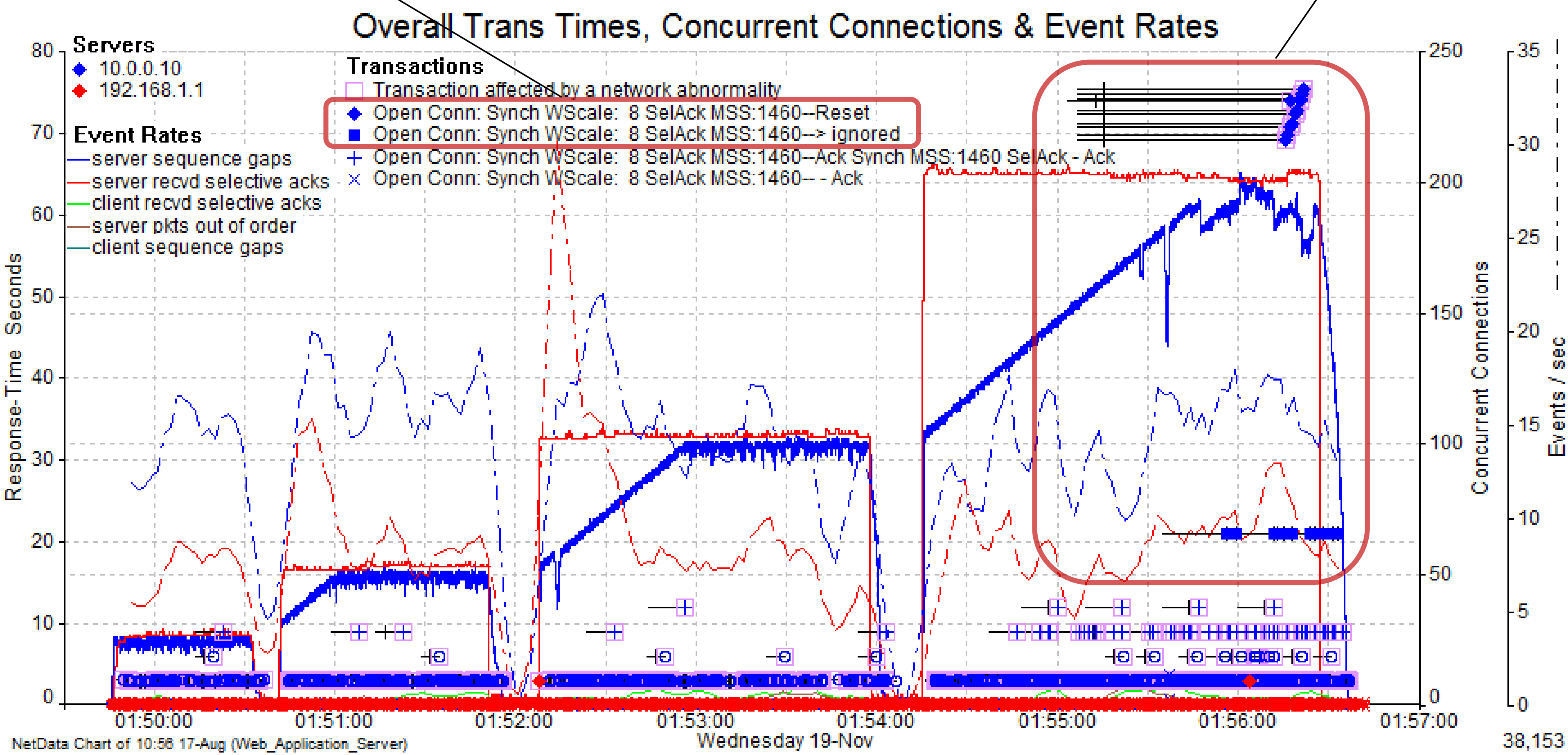
The capture file name appears here in all charts.

All Connection Setups

This chart shows the connection details for the full 7 minutes. The solid red & blue lines display the "concurrent connections" into each server. The "rectangular" red shapes indicate that the Load Generator quickly ramped up connections to the Web Server (first ~25, then ~50, ~100 & ~200 - RHS scale). These appear to be 4 test runs, each time doubling the number of concurrent connections. The blue Content Server connections ramp up at a slower rate. The red and blue markers represent connection setups (with height being setup time - LHS scale). The horizontal black lines in each setup also show duration.

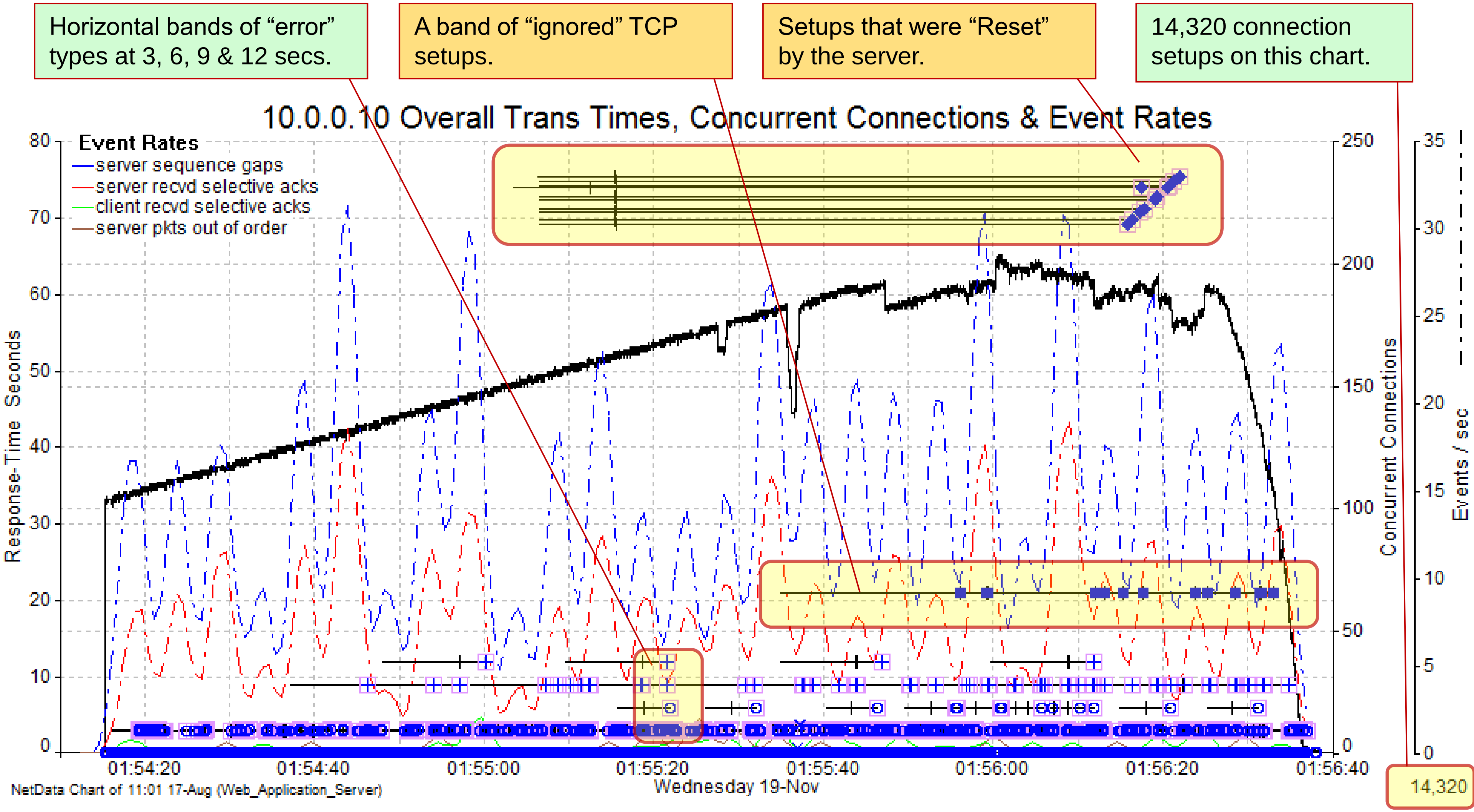
Some connections are not successful.

The most "interesting" activity seems to be here, during the heaviest test.



Setups Zoomed to Test Run Four

This is now zoomed-in to only the 4th test. Also, only Web Server to Content Server traffic is shown (Load Gen traffic removed). The connections were ignored or rejected towards the end when the concurrent connection count worked its way towards the 200 mark. The dashed lines plot the packet loss rates (which have a clearly visible regularity about them).
NetData makes problem behaviour “eye catching”. Those Resets and “ignored” connections certainly stand out here.



Connection Statistics (Test 4)

This table shows the statistics of the items that were plotted on the previous slide. That is, just test run four.

The 14,320 setups were mostly normal – but 11 were “Reset” and 47 “Ignored”.

The 258 with no Window Scaling are those where the first 2 client Syn packets were lost but the 3rd one made it through (and the server’s Syn-Ack also made it through).

As we will see, if the first 2 Syn packets don’t make it, the client’s third attempt does not specify a Window Scale factor. The Microsoft TCP Stack developer should get brownie points for that. If the first 2 Syns failed, perhaps the developer thought a modified “dumbed down” 3rd Syn might work?

ID	Transaction Description	Plot	Count	Req Bytes	SecsMin	Average	Maximum	Rsp Min	Rsp Bytes	End Avg	End Max
1	Open Conn: Synch WScale: 8 SelAck MSS:1460--Ack Synch MSS:1460 SelAck WScale: 8 - Ack	Yes	14000	70.0	0.0000	0.021	3.025	70	70.0	0.287	6.026
8	Open Conn: Synch WScale: 8 SelAck MSS:1460--Ack Synch MSS:1460 SelAck - Ack	Yes	258	70.0	0.0005	0.145	3.015	66	66.0	9.154	12.029
15	Open Conn: Synch WScale: 8 SelAck MSS:1460--> ignored	Yes	47	70.0					0.0	21.026	21.064
14	Open Conn: Synch WScale: 8 SelAck MSS:1460--Reset	Yes	11	70.0	60.3274	63.637	66.467		0.0	72.652	75.484
0	Open Conn: Synch WScale: 8 SelAck MSS:1460-- - Ack	Yes	4	70.0	0.7790	3.028	3.792	2920	2920.0	3.028	3.792

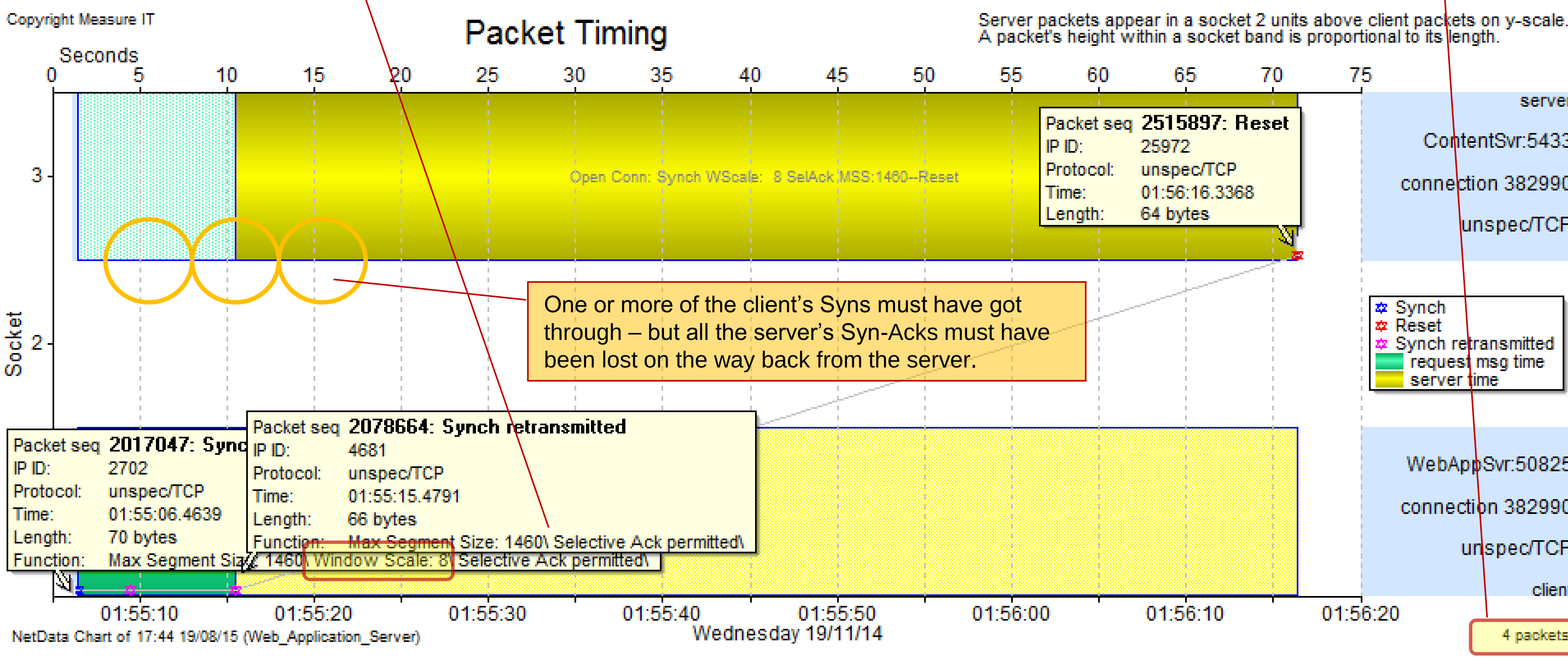
Reset “Rejected” Connections

The “rejected” connections look like this. [Client packets along the bottom row, server packets along the top row.]
The client sends a Syn that gets no response, so retransmits another after 3 seconds, no response, so a third Syn after a further 6 seconds. 60 seconds after that, the server responds with a Reset.

If the server “rejected” the request due to some known limitation, we would expect to see the Reset much sooner.
The inference here is that the server actually sent a Syn-Ack in response to one or more of the client Syn packets (because the server is aware of the connection). These Syn-Acks were all lost on the way back (so no corresponding client Ack). This forced the server to hold the connection open until it timed-out. Packet losses in both directions are responsible for this error.

A further observation is that Syn 3, unlike 1 & 2, does not specify a Window Scale factor.

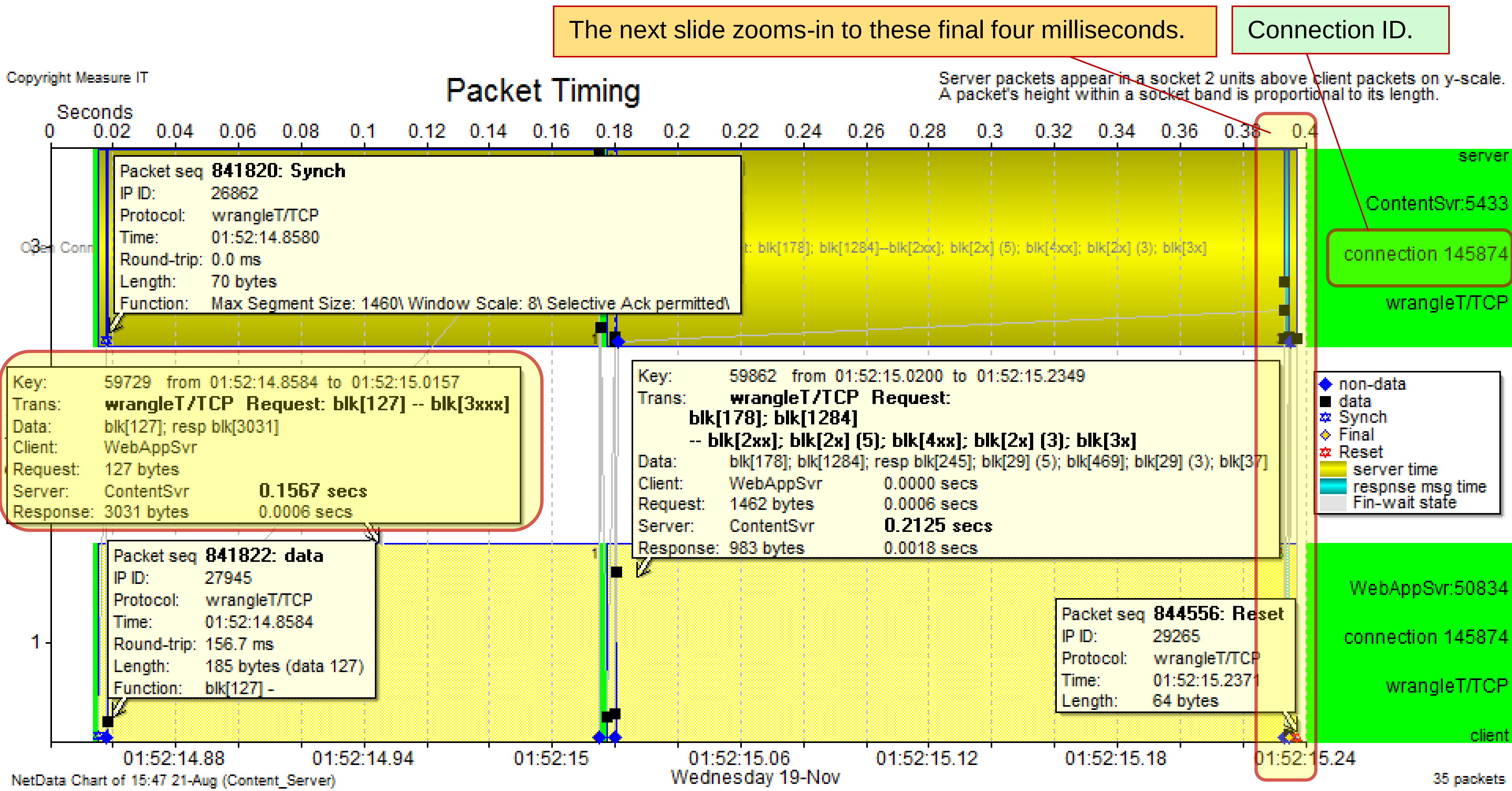
4 packets on this whole chart..



Packet Timing (Normal Transactions)

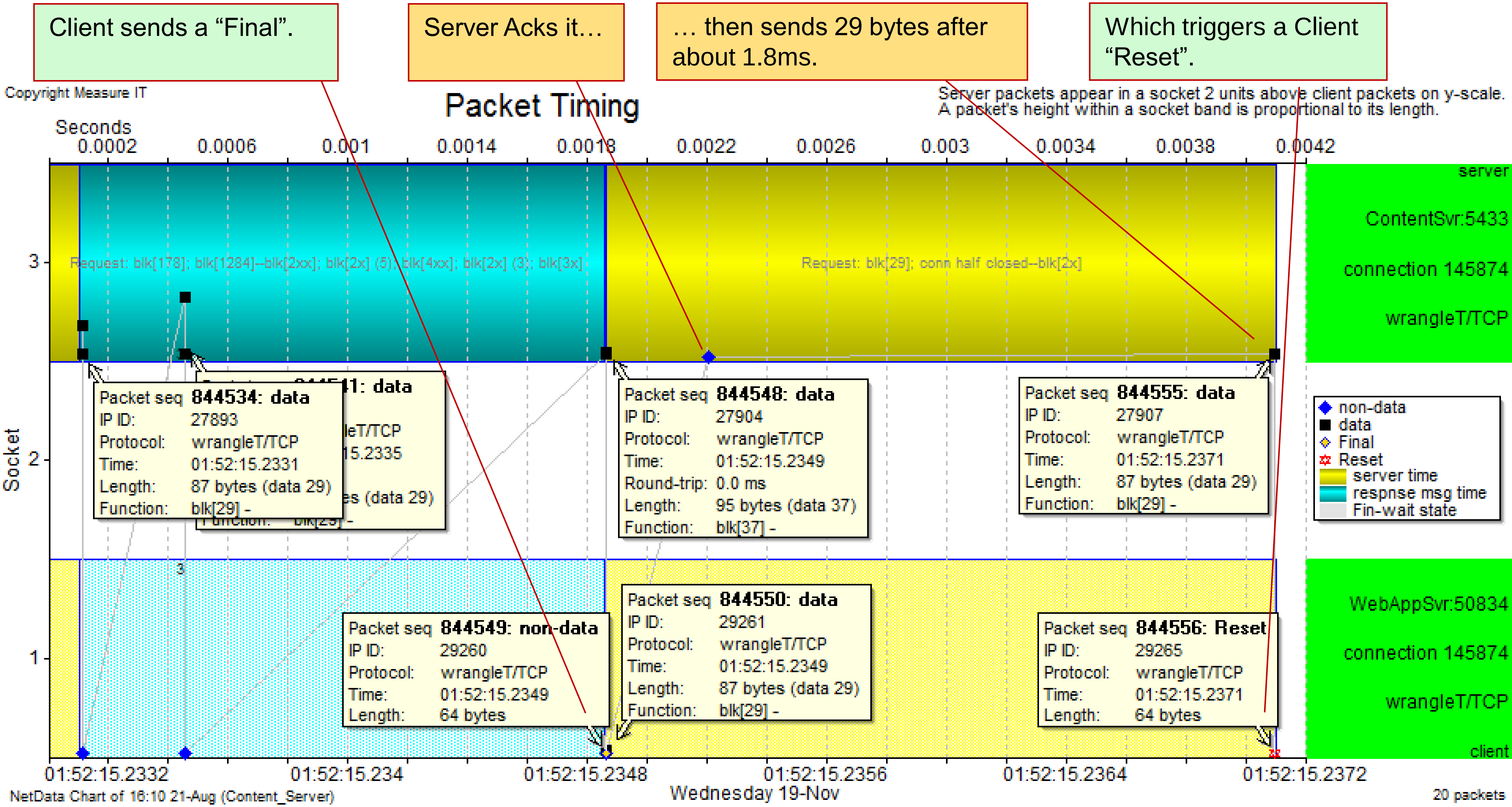
These chart types provide a visual representation of the packet times. Shapes & colours highlight different packet types (see legend). Blue diamonds are Acks. Vertical position of black squares indicates size of data packets .

The 3-way handshake is usually quick. The first transaction is always a 127 byte request with a 3031 byte response. Then more request/response transactions, followed by a client “Final”, server Ack, server data and client “Reset”.



Packet Flow (Normal Transactions)

This is the final part of the connection in the previous slide. The tail-end of the second transaction is in the blue area. Note the small packets (29 bytes is common). The client sends a 29 byte data packet – but with a Final packet (which the server Acks). The server responds with a 29 byte packet – and the client immediately replies with a Reset. All the connections look similar to this.



Packet Table (Normal Transactions)

Some readers may be more comfortable with this table of the packets from the last two slides.

	Time Of Day	Delta	Seq	Source	Destination	Len	Hdr	Net	TOS/TTL	IP ID	Tspt	Flags	Data Seq	Data Ack	ConnID	AppType	Data	Blks	Function
	01:52:14.8553		841794	WebAppSvr:50834	ContentSvr: 5433	70	70	IP4 DF	TTL 128	27934	TCP	S	568490814	0	145874	wrangleT			
	01:52:14.858	0.0028	841820	ContentSvr: 5433	WebAppSvr:50834	70	70	IP4 DF	TTL 126	26862	TCP	A S	3819784676	568490815	145874	wrangleT			
	01:52:14.858		841821	WebAppSvr:50834	ContentSvr: 5433	64	58	IP4 DF	TTL 128	27944	TCP	A	568490815	3819784677	145874	wrangleT			
	01:52:14.8584	0.0004	841822	WebAppSvr:50834	ContentSvr: 5433	185	58	IP4 DF	TTL 128	27945	TCP	AP	568490815	3819784677	145874	wrangleT	127	1	blk[127]
	01:52:15.0151	0.1567	842996	ContentSvr: 5433	WebAppSvr:50834	1518	58	IP4 DF	TTL 126	27397	TCP	A	3819784677	568490942	145874	wrangleT	1460	1	
	01:52:15.0151		842997	ContentSvr: 5433	WebAppSvr:50834	1518	58	IP4 DF	TTL 126	27398	TCP	A	3819786137	568490942	145874	wrangleT	1460	1	
	01:52:15.0151		842998	WebAppSvr:50834	ContentSvr: 5433	64	58	IP4 DF	TTL 128	28481	TCP	A	568490942	3819787597	145874	wrangleT			
	01:52:15.0157	0.0005	843000	ContentSvr: 5433	WebAppSvr:50834	169	58	IP4 DF	TTL 126	27401	TCP	AP	3819787597	568490942	145874	wrangleT	111	1	blk[3031]
	01:52:15.0175	0.0019	843008	WebAppSvr:50834	ContentSvr: 5433	216	58	IP4 DF	TTL 128	28486	TCP	AP	568490942	3819787708	145874	wrangleT	158	1	blk[158]
	01:52:15.02	0.0025	843025	ContentSvr: 5433	WebAppSvr:50834	64	58	IP4 DF	TTL 126	27415	TCP	AP	3819787708	568491100	145874	wrangleT	6	1	blk[6]
	01:52:15.02		843026	ContentSvr: 5433	WebAppSvr:50834	103	58	IP4 DF	TTL 126	27416	TCP	AP	3819787714	568491100	145874	wrangleT	45	1	blk[45]
	01:52:15.02		843027	WebAppSvr:50834	ContentSvr: 5433	64	58	IP4 DF	TTL 128	28491	TCP	A	568491100	3819787759	145874	wrangleT			
	01:52:15.02		843031	WebAppSvr:50834	ContentSvr: 5433	236	58	IP4 DF	TTL 128	28493	TCP	AP	568491100	3819787759	145874	wrangleT	178	1	blk[178]
	01:52:15.0206	0.0006	843034	WebAppSvr:50834	ContentSvr: 5433	1342	58	IP4 DF	TTL 128	28496	TCP	AP	568491278	3819787759	145874	wrangleT	1284	1	blk[1284]
	01:52:15.0209	0.0004	843043	ContentSvr: 5433	WebAppSvr:50834	64	58	IP4 DF	TTL 126	27422	TCP	A	3819787759	568492562	145874	wrangleT			
■	01:52:15.2331	0.2122	844531	ContentSvr: 5433	WebAppSvr:50834	303	58	IP4 DF	TTL 126	27891	TCP	AP	3819787759	568492562	145874	wrangleT	245	1	blk[245]
■	01:52:15.2331		844532	ContentSvr: 5433	WebAppSvr:50834	87	58	IP4 DF	TTL 126	27892	TCP	AP	3819788004	568492562	145874	wrangleT	29	1	blk[29]
◆	01:52:15.2331		844533	WebAppSvr:50834	ContentSvr: 5433	64	58	IP4 DF	TTL 128	29256	TCP	A	568492562	3819788033	145874	wrangleT			
■	01:52:15.2331		844534	ContentSvr: 5433	WebAppSvr:50834	87	58	IP4 DF	TTL 126	27893	TCP	AP	3819788033	568492562	145874	wrangleT	29	1	blk[29]
■	01:52:15.2331		844535	ContentSvr: 5433	WebAppSvr:50834	87	58	IP4 DF	TTL 126	27894	TCP	AP	3819788062	568492562	145874	wrangleT	29	1	blk[29]
■	01:52:15.2331		844536	ContentSvr: 5433	WebAppSvr:50834	87	58	IP4 DF	TTL 126	27895	TCP	AP	3819788091	568492562	145874	wrangleT	29	1	blk[29]
■	01:52:15.2331		844537	ContentSvr: 5433	WebAppSvr:50834	87	58	IP4 DF	TTL 126	27896	TCP	AP	3819788120	568492562	145874	wrangleT	29	1	blk[29]
◆	01:52:15.2331		844538	WebAppSvr:50834	ContentSvr: 5433	64	58	IP4 DF	TTL 128	29257	TCP	A	568492562	3819788149	145874	wrangleT			
■	01:52:15.2335	0.0003	844539	ContentSvr: 5433	WebAppSvr:50834	527	58	IP4 DF	TTL 126	27897	TCP	AP	3819788149	568492562	145874	wrangleT	469	1	blk[469]
■	01:52:15.2335		844540	ContentSvr: 5433	WebAppSvr:50834	87	58	IP4 DF	TTL 126	27898	TCP	AP	3819788618	568492562	145874	wrangleT	29	1	blk[29]
■	01:52:15.2335		844541	ContentSvr: 5433	WebAppSvr:50834	87	58	IP4 DF	TTL 126	27899	TCP	AP	3819788647	568492562	145874	wrangleT	29	1	blk[29]
◆	01:52:15.2335		844542	WebAppSvr:50834	ContentSvr: 5433	64	58	IP4 DF	TTL 128	29258	TCP	A	568492562	3819788676	145874	wrangleT			
■	01:52:15.2349	0.0014	844547	ContentSvr: 5433	WebAppSvr:50834	87	58	IP4 DF	TTL 126	27903	TCP	AP	3819788676	568492562	145874	wrangleT	29	1	blk[29]
■	01:52:15.2349		844548	ContentSvr: 5433	WebAppSvr:50834	95	58	IP4 DF	TTL 126	27904	TCP	AP	3819788705	568492562	145874	wrangleT	37	1	blk[37]
◆	01:52:15.2349		844549	WebAppSvr:50834	ContentSvr: 5433	64	58	IP4 DF	TTL 128	29260	TCP	A	568492562	3819788742	145874	wrangleT			
■	01:52:15.2349		844550	WebAppSvr:50834	ContentSvr: 5433	87	58	IP4 DF	TTL 128	29261	TCP	AP	568492562	3819788742	145874	wrangleT	29	1	blk[29]
◆	01:52:15.2349		844551	WebAppSvr:50834	ContentSvr: 5433	64	58	IP4 DF	TTL 128	29262	TCP	A F	568492591	3819788742	145874	wrangleT			
◆	01:52:15.2352	0.0003	844553	ContentSvr: 5433	WebAppSvr:50834	64	58	IP4 DF	TTL 126	27905	TCP	A	3819788742	568492592	145874	wrangleT			
■	01:52:15.2371	0.0019	844555	ContentSvr: 5433	WebAppSvr:50834	87	58	IP4 DF	TTL 126	27907	TCP	AP	3819788742	568492592	145874	wrangleT	29	1	blk[29]
△	01:52:15.2371		844556	WebAppSvr:50834	ContentSvr: 5433	64	58	IP4 DF	TTL 128	29265	TCP	A R	568492592	3819788771	145874	wrangleT			

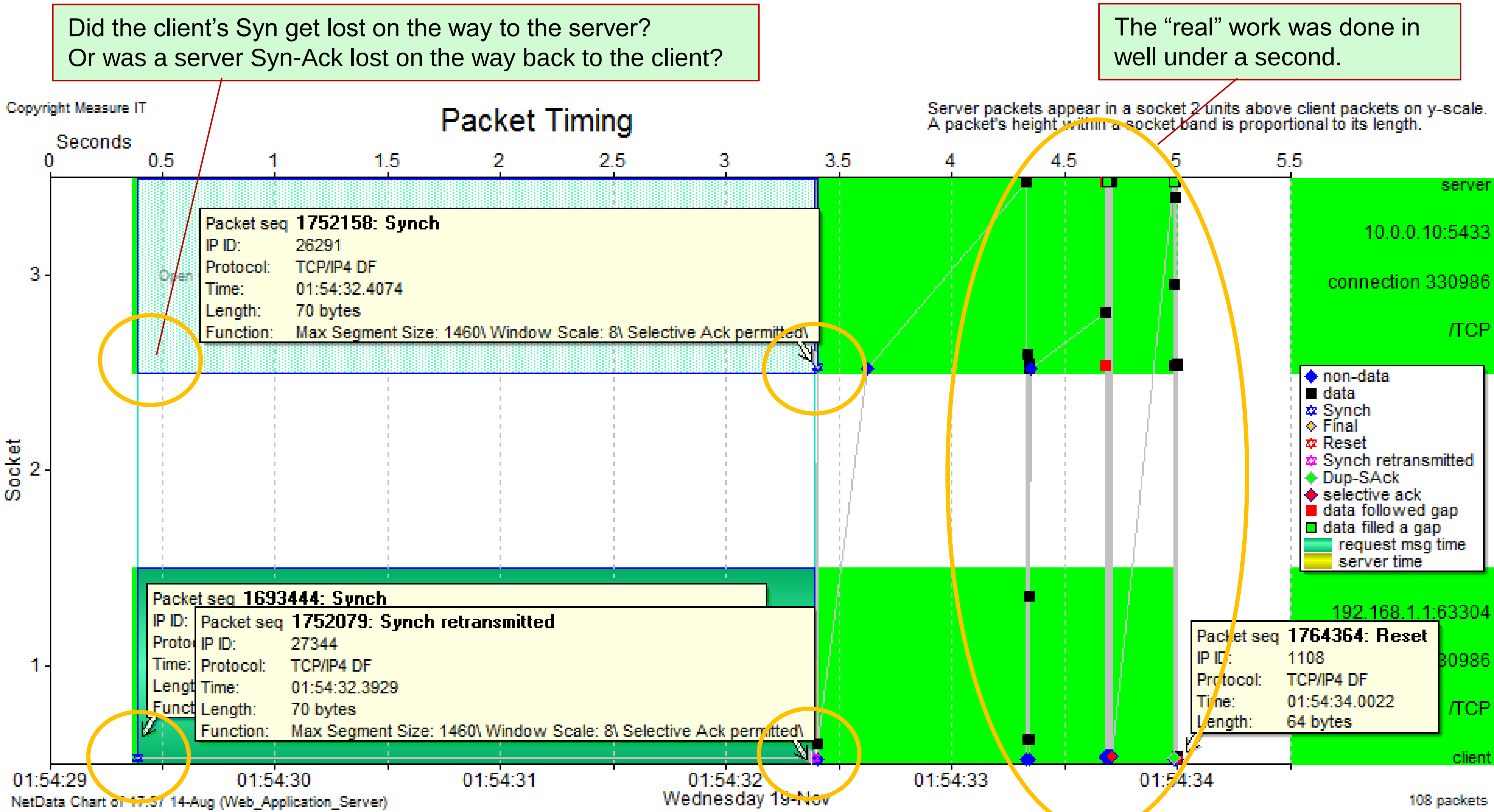
The “Lost Syn” Connections

We’ve already seen what the “rejected” connections look like (Multiple lost Syn or Syn-Ack packets then a server Reset after 60-70 seconds).

In the following few slides we’ll examine the packet timings and behaviours for the 3, 6, 9 and 12 second connection setups as well as the “ignored” connections.

Packet Flow (3 Second)

This is an example of one of the 3 second connection setups. The initial client Syn is ignored but the 2rd Syn (after 3 seconds) is answered. After that the first [127-3031] transaction takes under 1 second then 3 more transactions in 0.7 sec before the usual termination, Fin-Ack-Data-Reset.



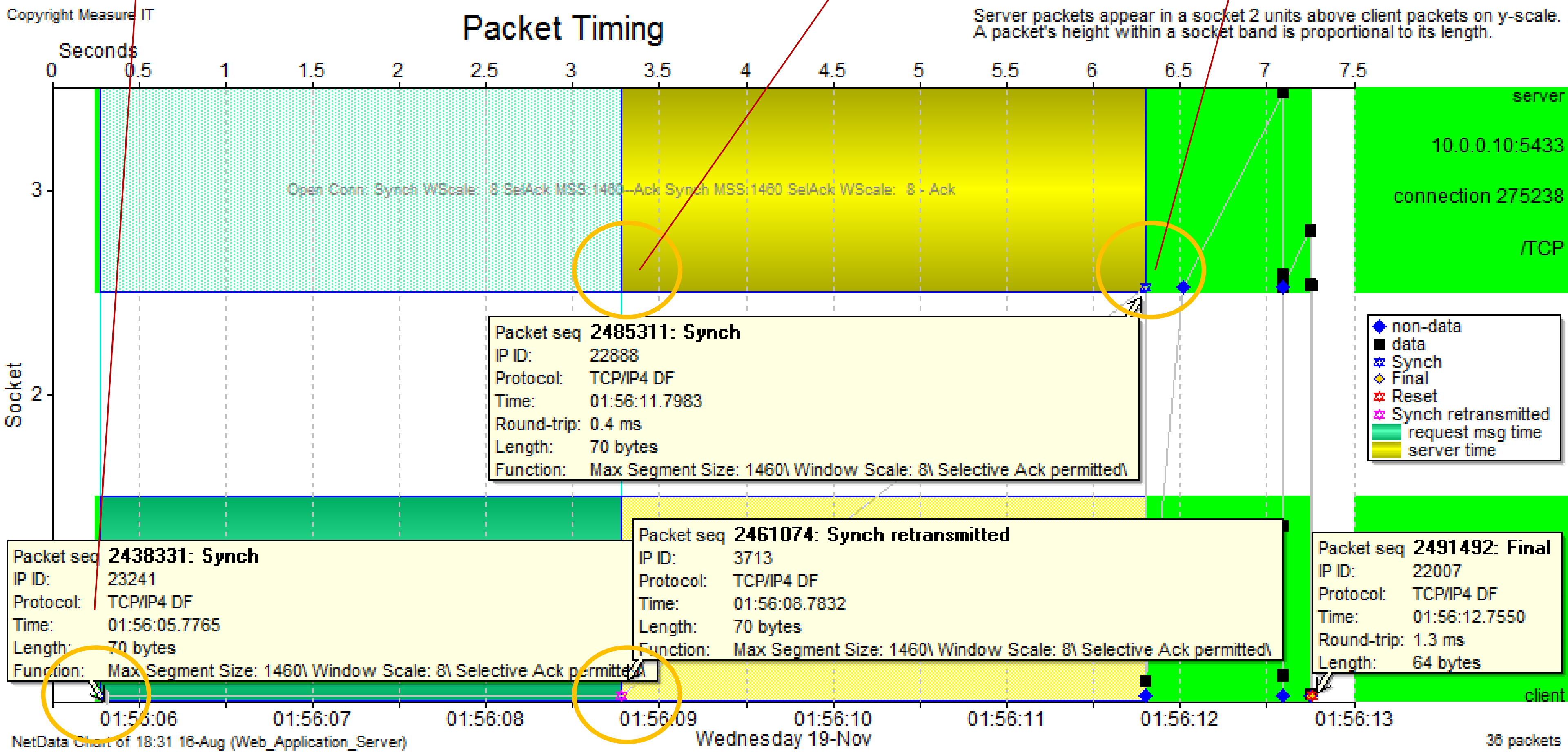
Packet Flow (6 Second)

This is an example of the 6 second connection setups. The initial client Syn is ignored but the 2[rd] Syn (after 3 seconds) is answered. Note the matching Window Scale factors.
After that the normal transactions take a total 1 second before the usual termination, Fin-Ack-Data-Reset.

Did the client's Syn get lost on the way to the server?
Or was a server Syn-Ack lost on the way back to the client?

A server Syn-Ack was lost on the way back to the client?

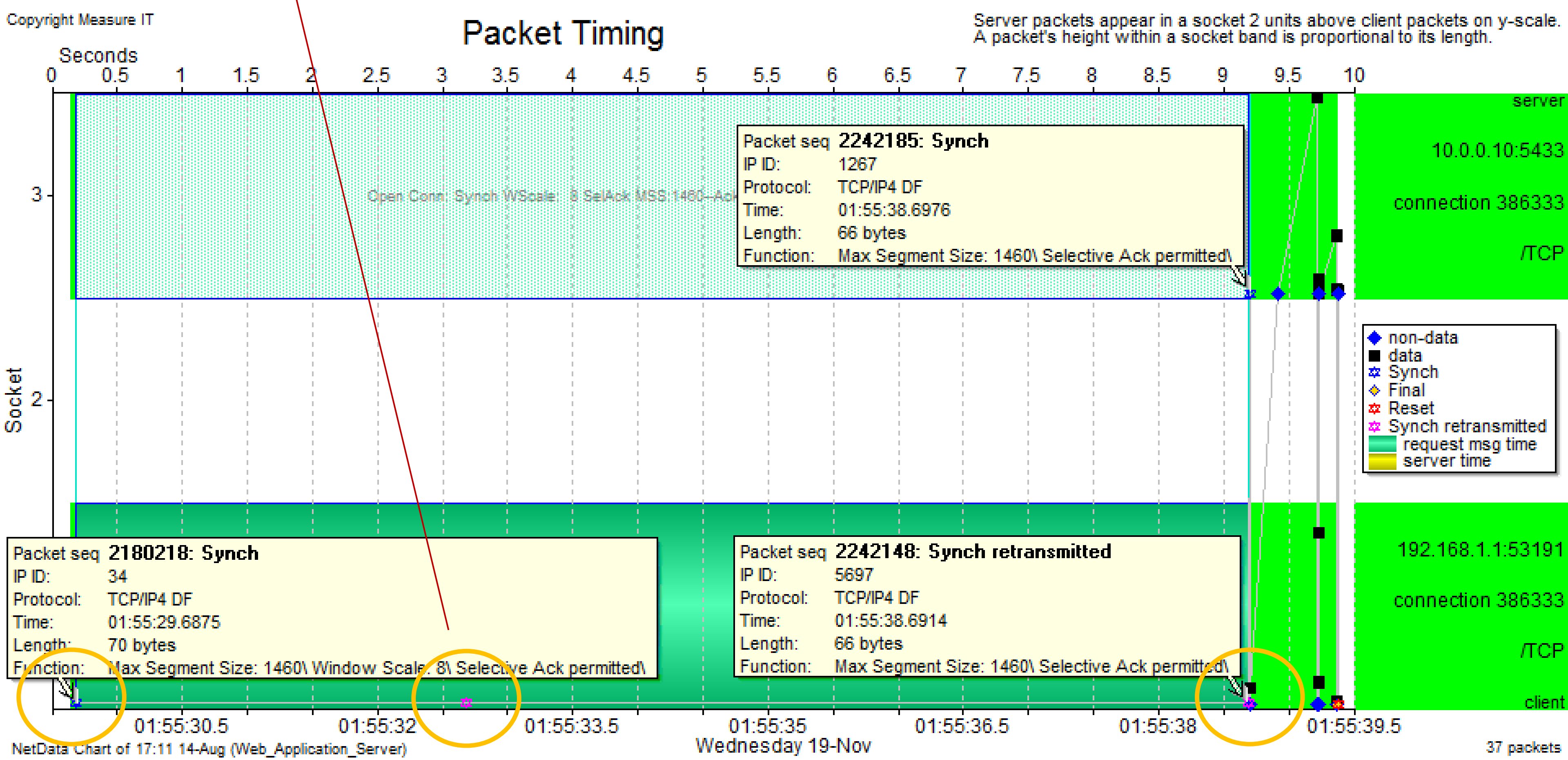
Prompting a 2[nd] Syn-Ack three secs later.



Packet Flow (9 Second)

This is an example of one of the 9 second connection setups. Both the initial client Syn and the retransmitted Syn are ignored but the 3rd Syn (after a further 6 seconds) is answered.
The fact that the server's Syn-Ack has no Window Scaling may be a hint that the first two client Syn packets were indeed lost on the way (rather than the server's Syn-Acks being lost on the way back).

Did the client's two Syns get lost on the way to the server?
Or were the server's Syn-Acks lost on the way back to the client?

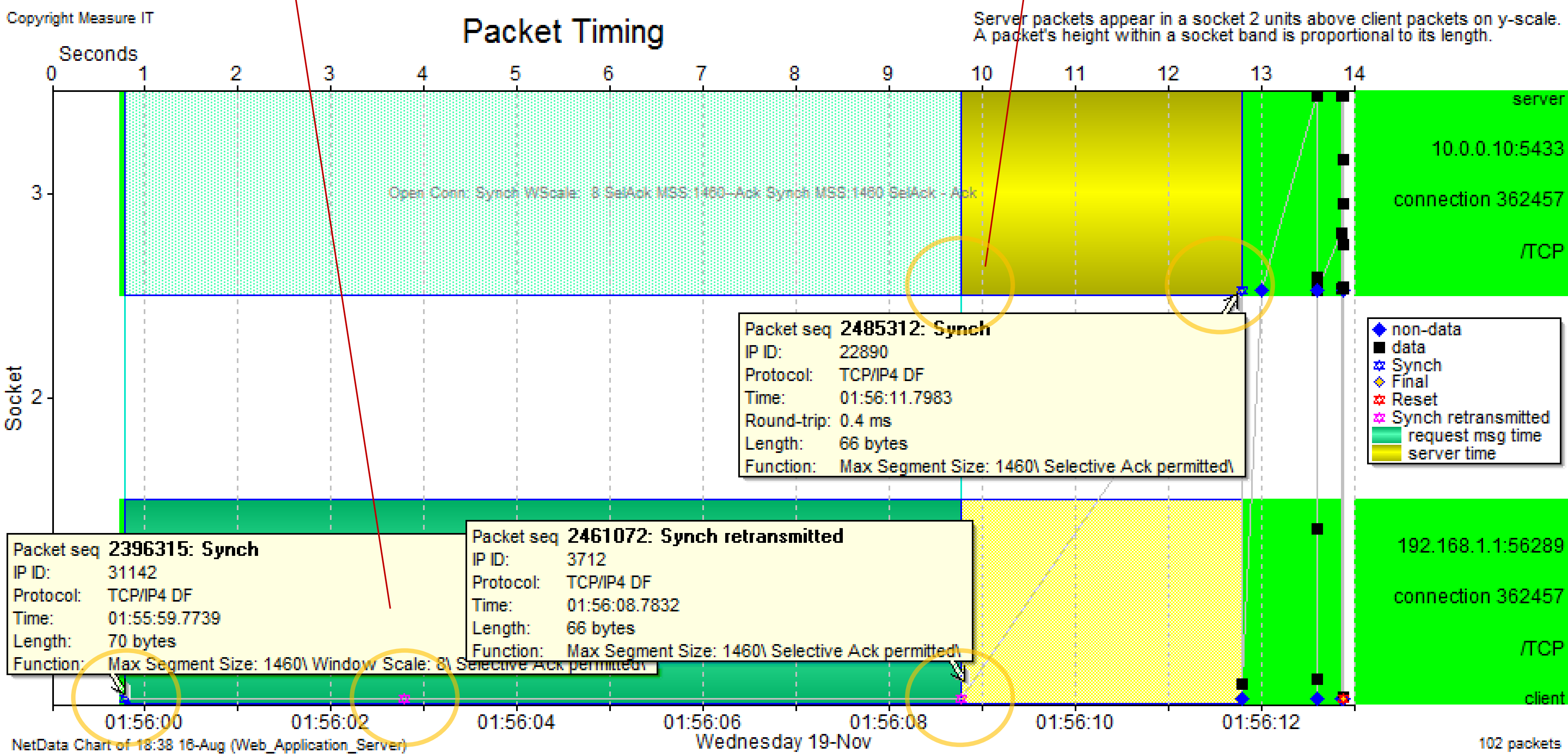


Packet Flow (12 Second)

This is an example of the 12 second connection setups. It is like a 9 second one, but with the extra 3 second delay from the server. Note the matching “No Window Scaling”.

These two client Syns were likely lost on the way to the server?

Was a server Syn-Ack (expected here) lost on the way back to the client?

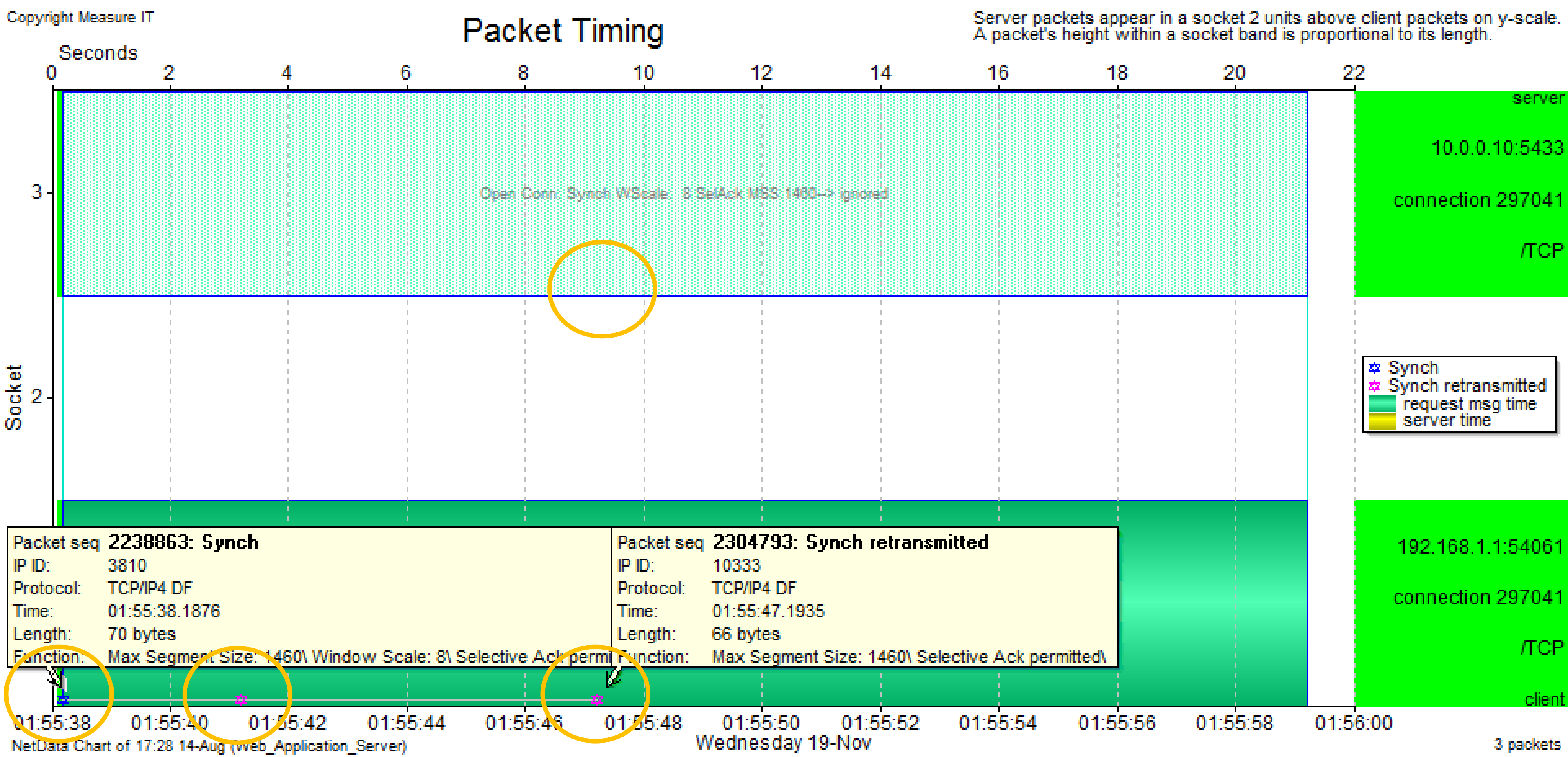


Packet Flow (“Ignored” Connection)

This is an example of the “21 second” connection setups. All of the client’s 3 Syn packets are ignored (or lost). There is no response from the server at all. This would have resulted in failed transactions from the client’s viewpoint.

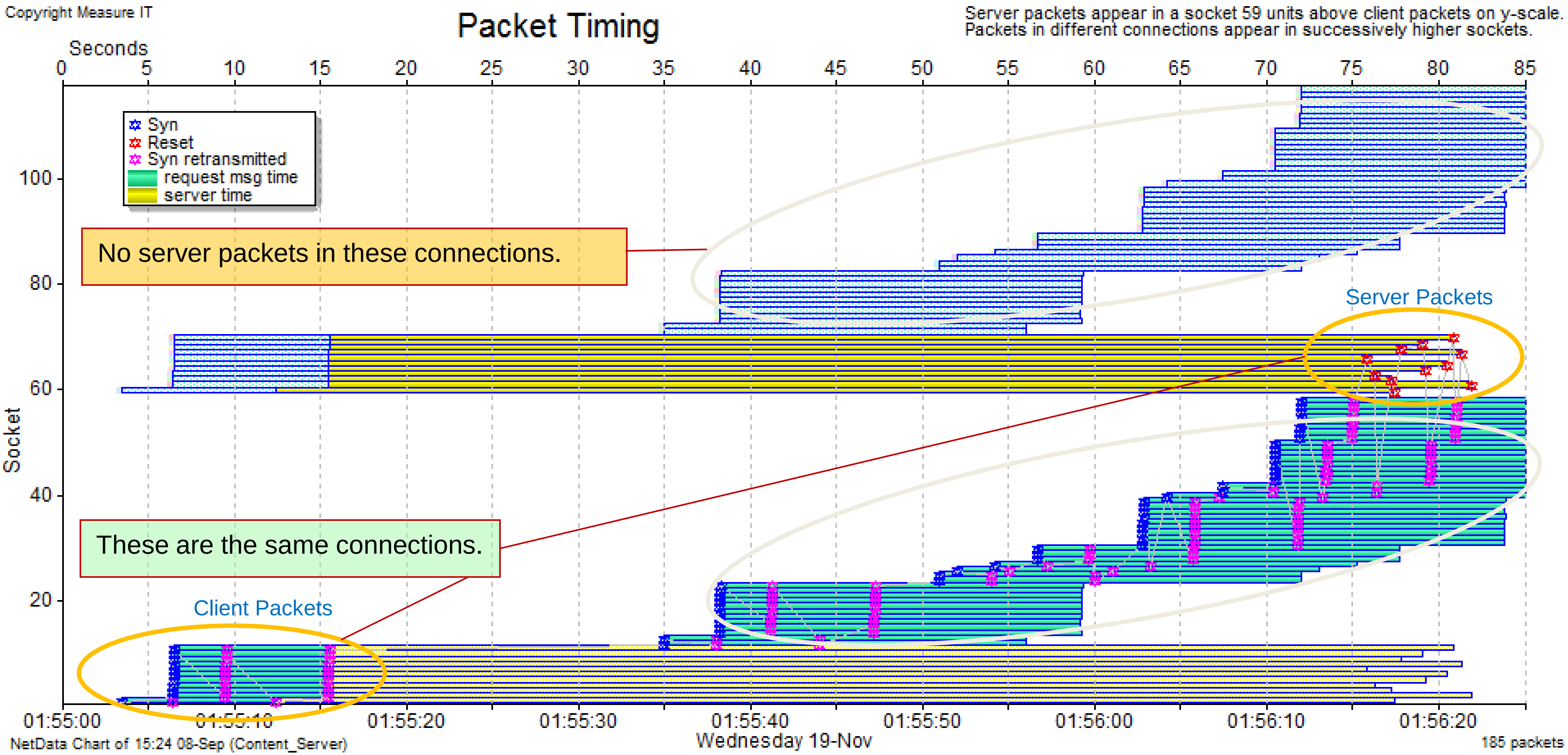
Alternatively, the server’s Syn-Acks could have been lost on the way back (but we have no evidence that the server knew about this connection).

Another possibility is that the capture terminated before we saw the server’s 60-second Reset.



Lost Syn Behaviour

This is a similar chart as the above – but showing about 60 connections all at once. Socket labels 1-59 are the client rows for the corresponding server rows 60-119. These are the “ignored” and “rejected” connections – so only Syn, Syn-Ack and Reset packets appear.
The first batch of yellow circled Syns and Retransmitted Syns end up with (yellow circled) server Resets after around 70 seconds.
Subsequent “batches” of unanswered Syns/Retrans (light blue circled) occur with exponentially increasing frequency – but have no responses. Could it be that the capture was stopped before we got to see the server’s Resets? Making “ignored” the same as “rejected”?



Packet Loss Behaviour

Now that we know that packet losses are the reason for the failed transactions, we now need to examine the form the losses took in order to narrow down the possible causes.

There are two different loss types - regular data packet losses and Syn losses under heavier loads. Both types occur between the taps and the Content Server (not between the two taps).

The following slides present various different views of the behaviour.

Data Packet Losses.

- The packet losses are not random. They occur for 2.5 seconds in every 5 second time period.
- They are not related to load (they occur regularly across all 4 test runs).
- They are not related to transaction payload size.

Syn Packet Losses.

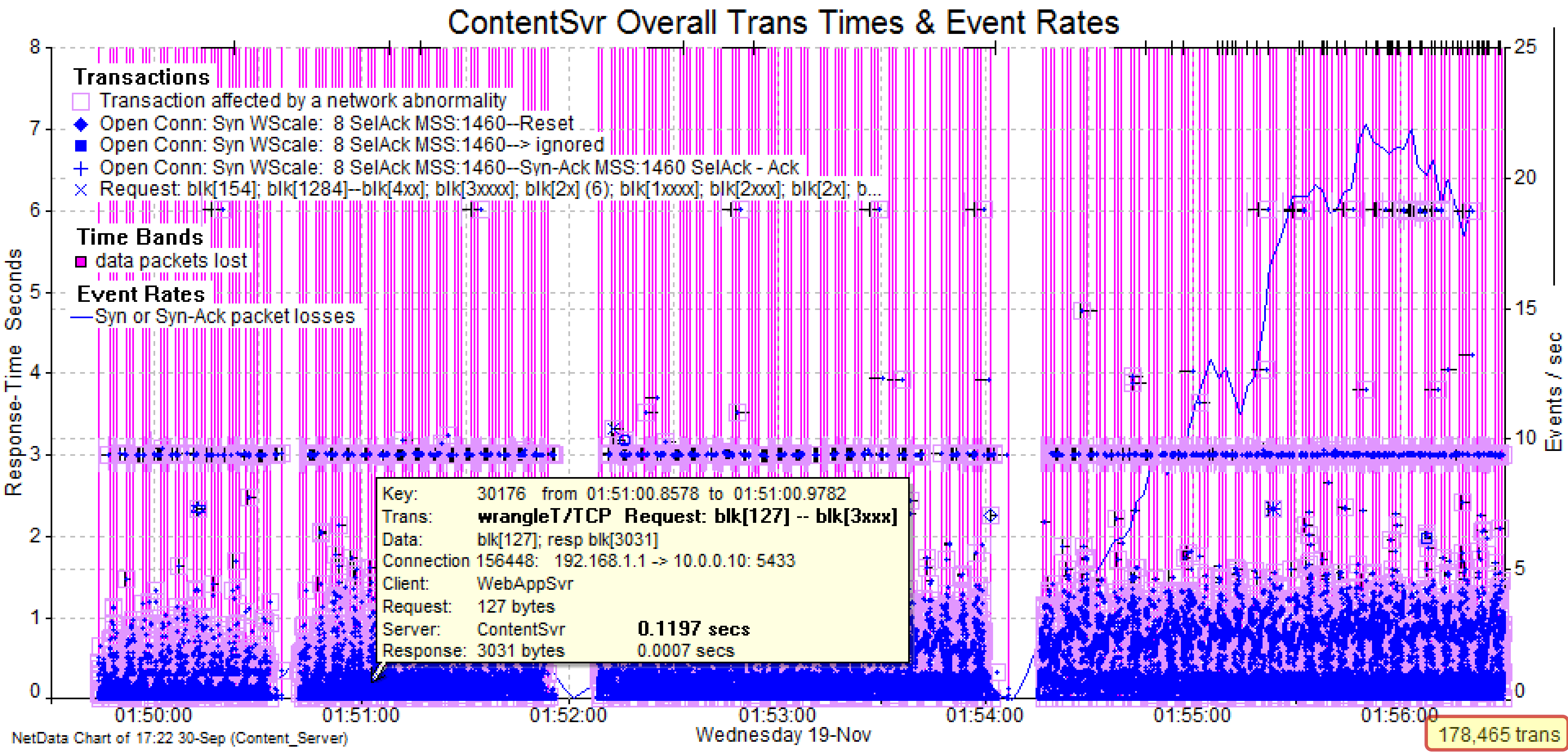
- Occur mostly towards the end of the test period – increasing as load increases.
- Some Syn losses do occur during the 2.5 second “lossy” periods
- Can occur in contiguous groups (as already seen in the previous slide).
- They are not related to traffic load (i.e., they can occur when there is no other simultaneous network activity).
- They are related to server stress (successful connections at the same time have slower responses).
- If not server stress, then some stress in an intermediate device such as a firewall or load balancer.

Two Kinds of Packet Loss Behaviour

The packet losses are not random. They occur for 2.5 seconds (or half) of every 5 second period.

As the legend says, the pink background represents times where lost data packets were observed. These form fairly consistent bands across the whole 7 minute time period. The blue dashed line shows that Syn/Syn-Ack losses ramped up during the fourth test run. So general losses are not load related, but Syn losses do increase up to 20+ per sec when under load (apparently due to a different loss mechanism).

Many of the 178,465 transactions are affected by the losses (pink squares) and times are generally longer in test run four.



Packet Losses Correlate with Slow Setups

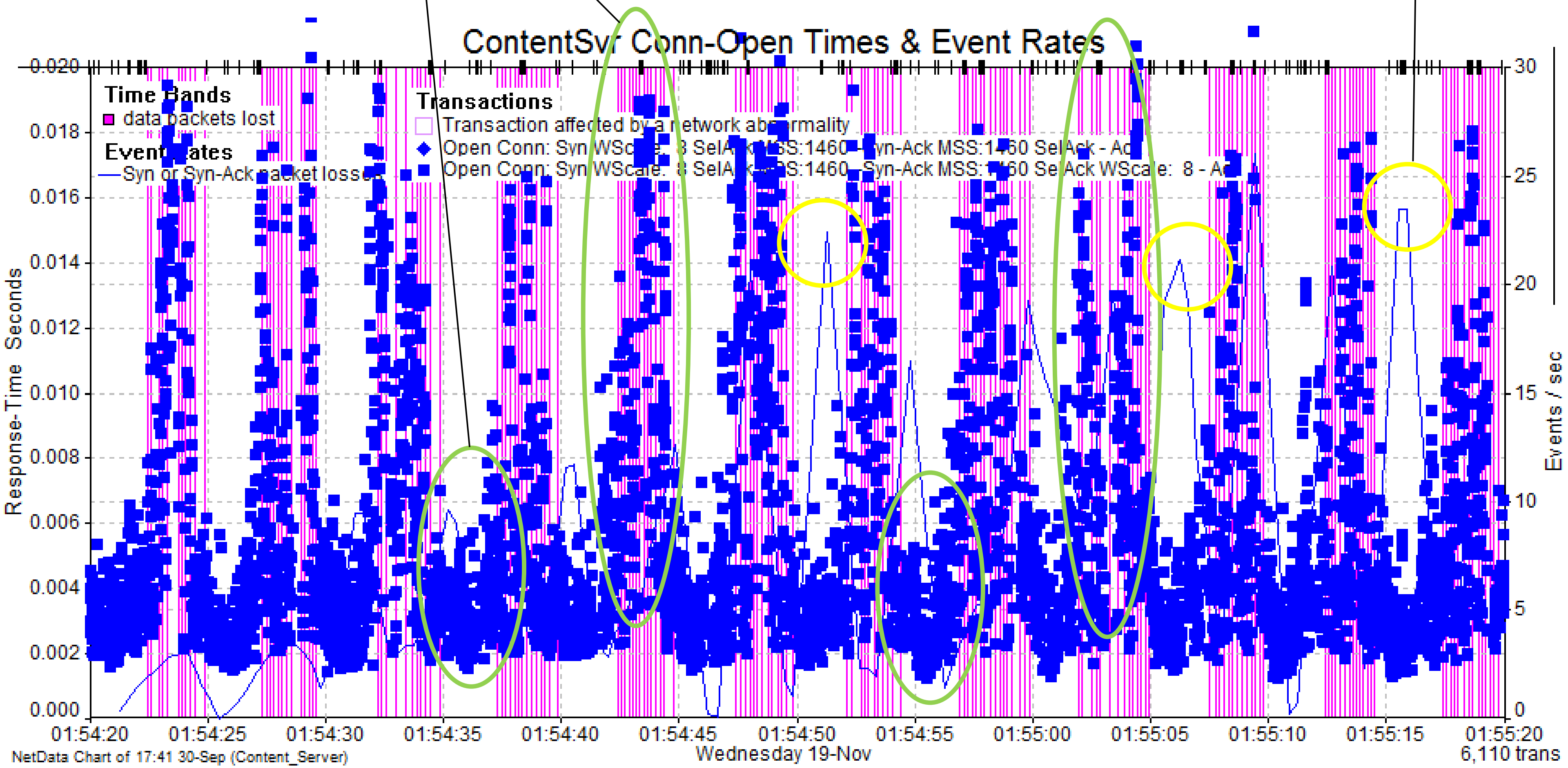
The blue markers here represent TCP 3-way handshakes (other transaction types are not shown). The pink background is where there are packet losses in general (in transactions not plotted here). Connection setups are all milliseconds quicker in periods without packet loss.

Whatever causes the regular losses might also slow down TCP setups – OR – whatever slows down setups might also cause packet losses?

All the setups within the “non-lossy” periods (white background) are faster…

… than setups during the “lossy” (pink background) periods.

Also observe the Syn/Syn-Ack losses in white areas (i.e., not correlated with the other losses).

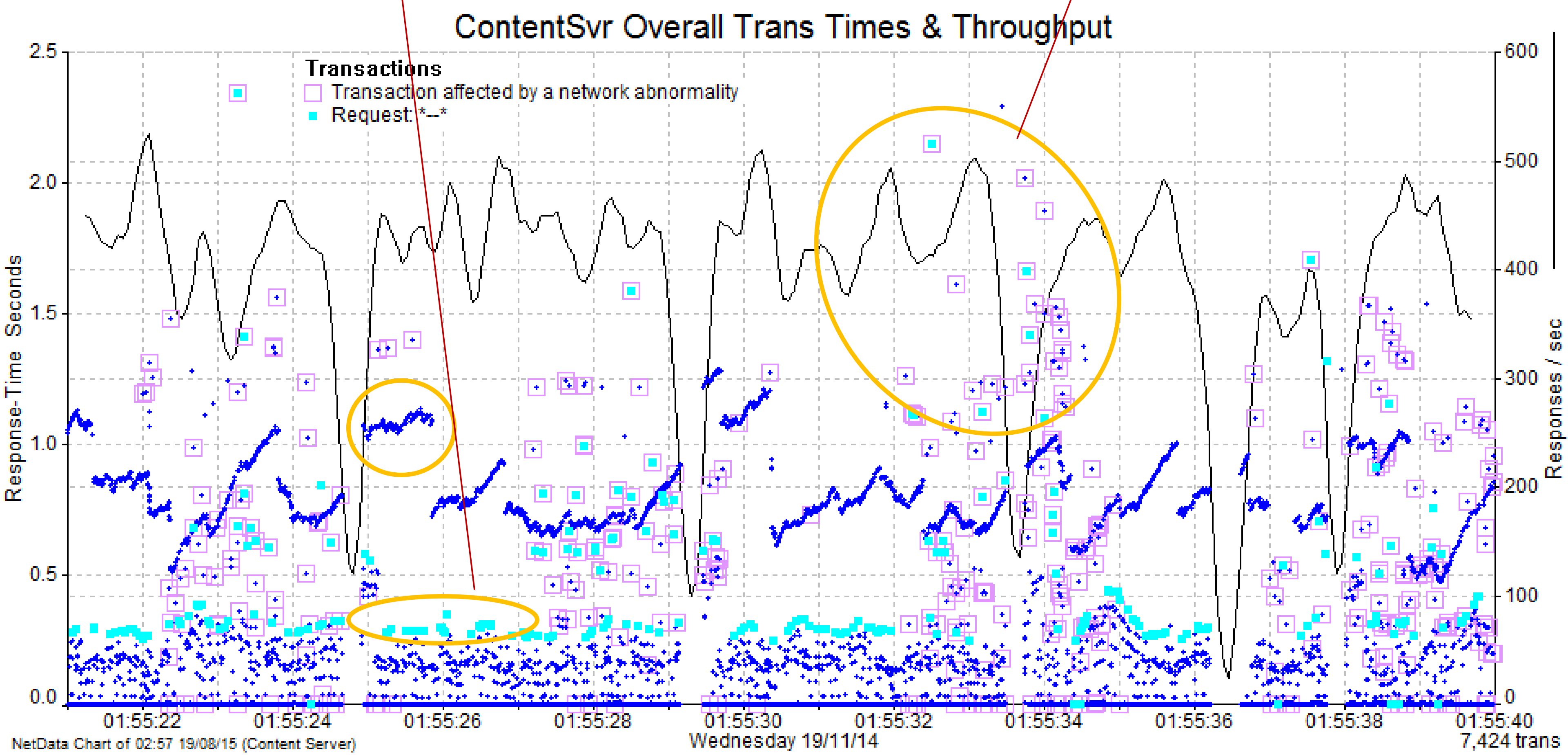


Packet Loss Behaviour (Response Size Doesn't Matter)

The transactions with light blue markers have response messages longer than 52 KBytes, and all other transactions have messages shorter than 5 KB. Transactions with pink squares are those containing packet losses and retransmissions.
There doesn't appear to be any correlation of packet loss with response message length.

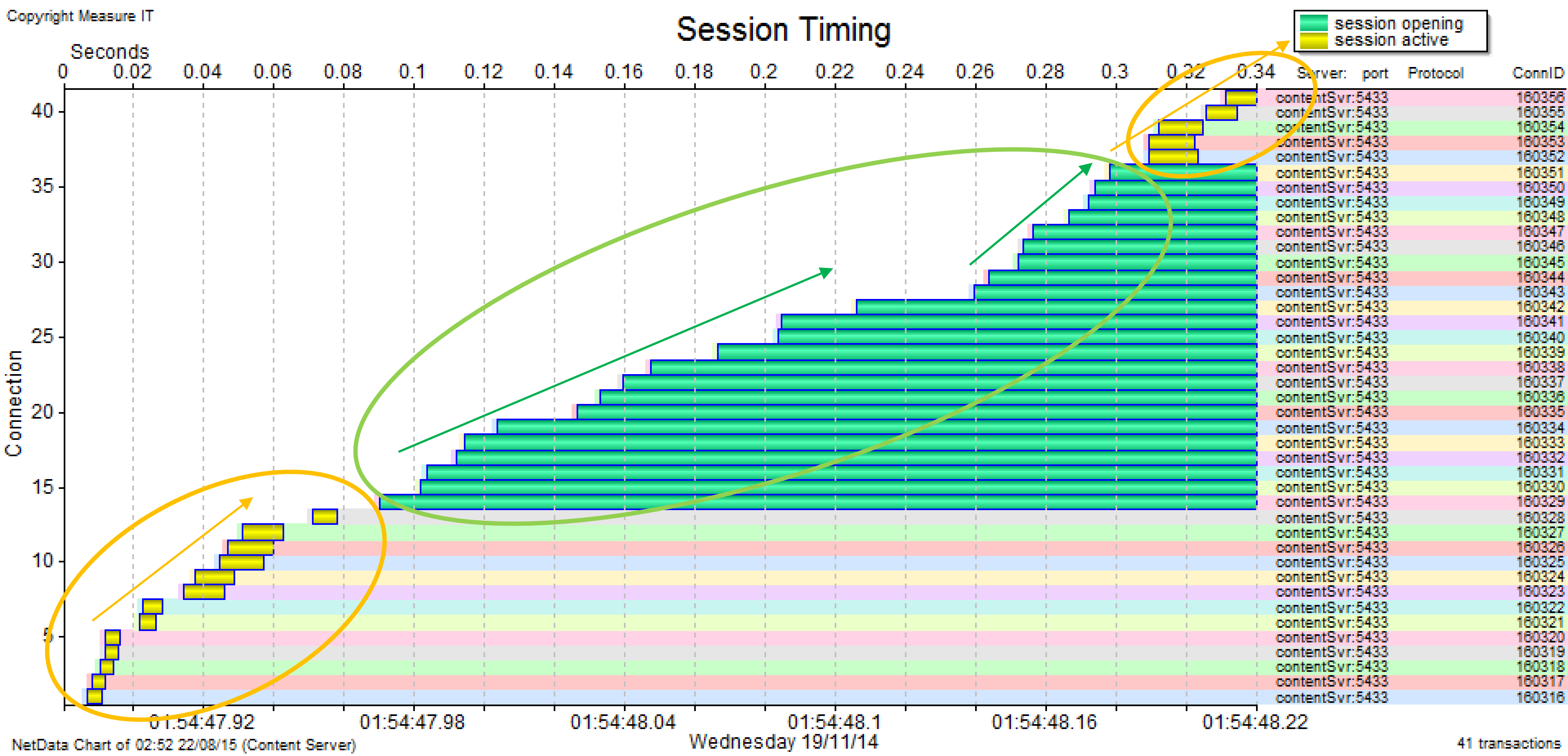
These large and small sized transactions contain no losses.

Large and small both contain losses.



Syn Losses Occur in “Batches”

Yellow bars are successful connection setups, green are setups with lost Syn/Syn-Acks. Horizontal length of bars represents duration (x-axis is time-of-day). Entire chart is only a third of a second wide. Concurrent setups had been handled successfully – but with slightly increasing delays – and then a large batch of Syn/Syn-Ack packets were lost (no successful setups in that time). Then good connections again. The connection arrival rate (slope of left edges) was not unusually high.



Port Numbers are "Recycled"

There are so many TCP connection requests that the same client ephemeral port numbers get reused two or three times each during the 4 test runs.

The Web Server (acting as client) starts its port numbers at 49155 and it takes just over 3 minutes to cycle through the ports 49155-65534.

This doesn't appear to be the cause of the failed connection attempts.

It can be a problem when new connections are attempted on a port that was used recently and is still in the server's Time-Wait state.

If workloads are expected to become as high as the load test, then to eliminate this as a potential future problem, we could suggest to the customer that they configure the Web Server to begin its port number sequence with a lower starting value than 49155. This would extend the time period between reuse of the same client port numbers.

Connections Table

This is a small portion of the Connections table, sorted by client port number. It shows the 10 “Server Refused” connection requests that began very close together. Observe that other requests around the same time were successful.

A secondary observation is that all the port numbers appear twice, showing that they were “recycled” after 171 seconds (2:51 min). The earlier connections that used these port numbers were successful.

ConnID	Type	Client (Call...	cPort	Server (Call...	sPort	First Packet	Closing	Closure	Total sec	Trips	Clt Pkts	ReTx	Kbps	Svr Pkts	ReTx	Kbps
381490	TCP	192.168.1.1	50822	10.0.0.10	5433	opn01:52:14.7728	Fclt01:52:14.9677	cltF cR	0.1964	4	12		19.840	12		33.136
381490	TCP	192.168.1.1	50822	10.0.0.10	5433	opn01:55:06.4223	Fclt01:55:10.6360	cltF cR	4.2171	4	16	1	5.282	21		10.142
381990	TCP	192.168.1.1	50823	10.0.0.10	5433	opn01:52:14.7777	Fclt01:52:14.9826	cltF cR	0.2059	4	12		19.840	12		33.136
381990	TCP	192.168.1.1	50823	10.0.0.10	5433	opn01:55:06.4464	Refu01:55:27.4465	svrRefusedR	75.4838	0	3	2	0.078	1		0.024
382490	TCP	192.168.1.1	50824	10.0.0.10	5433	opn01:52:14.7803	Fclt01:52:14.9906	cltF cR	0.2112	4	12		19.840	12		33.136
382490	TCP	192.168.1.1	50824	10.0.0.10	5433	opn01:55:06.4593	Refu01:55:27.4592	svrRefusedR	70.8089	0	3	2	0.078	1		0.024
382990	TCP	192.168.1.1	50825	10.0.0.10	5433	opn01:52:14.7971	Fclt01:52:14.9953	cltF cR	0.1990	4	12		19.840	12		33.136
382990	TCP	192.168.1.1	50825	10.0.0.10	5433	opn01:55:06.4639	Refu01:55:27.4640	svrRefusedR	69.8729	0	3	2	0.078	1		0.024
383490	TCP	192.168.1.1	50826	10.0.0.10	5433	opn01:52:14.8030	Fclt01:52:15.0093	cltF cR	0.2078	4	12		19.840	12		33.136
383490	TCP	192.168.1.1	50826	10.0.0.10	5433	opn01:55:06.4733	Refu01:55:27.4733	svrRefusedR	72.7751	0	3	2	0.078	1		0.024
383990	TCP	192.168.1.1	50827	10.0.0.10	5433	opn01:52:14.8049	Fclt01:52:15.0125	cltF cR	0.2084	4	12		19.840	12		33.136
383990	TCP	192.168.1.1	50827	10.0.0.10	5433	opn01:55:06.4906	Refu01:55:27.4906	svrRefusedR	74.0059	0	3	2	0.078	1		0.024
384490	TCP	192.168.1.1	50828	10.0.0.10	5433	opn01:52:14.8209	Fclt01:52:15.0239	cltF cR	0.2044	4	12		19.840	12		33.136
384490	TCP	192.168.1.1	50828	10.0.0.10	5433	opn01:55:06.4965	Refu01:55:27.4965	svrRefusedR	69.3411	0	3	2	0.078	1		0.024
384990	TCP	192.168.1.1	50829	10.0.0.10	5433	opn01:52:14.8231	Fclt01:52:15.0232	cltF cR	0.2021	4	12		19.840	12		33.136
384990	TCP	192.168.1.1	50829	10.0.0.10	5433	opn01:55:06.4993	Refu01:55:27.4993	svrRefusedR	74.8189	0	3	2	0.078	1		0.024
385490	TCP	192.168.1.1	50830	10.0.0.10	5433	opn01:52:14.8306	Fclt01:52:15.0301	cltF cR	0.2021	4	12		19.840	12		33.136
385490	TCP	192.168.1.1	50830	10.0.0.10	5433	opn01:55:06.5219	Refu01:55:27.5219	svrRefusedR	71.2969	0	3	2	0.078	1		0.024
385990	TCP	192.168.1.1	50831	10.0.0.10	5433	opn01:52:14.8348	Fclt01:52:15.0353	cltF cR	0.2015	4	12		19.840	12		33.136
385990	TCP	192.168.1.1	50831	10.0.0.10	5433	opn01:55:06.5530	Refu01:55:27.5530	svrRefusedR	72.5156	0	3	2	0.078	1		0.024
386490	TCP	192.168.1.1	50832	10.0.0.10	5433	opn01:52:14.8383	Fclt01:52:15.2115	cltF cR	0.3751	4	15		21.696	20		42.224
386490	TCP	192.168.1.1	50832	10.0.0.10	5433	opn01:55:06.5577	Refu01:55:27.5577	svrRefusedR	74.2594	0	3	2	0.078	1		0.024
386990	TCP	192.168.1.1	50833	10.0.0.10	5433	opn01:52:14.8397	Fclt01:52:15.2214	cltF cR	0.3862	4	15		21.696	20		42.224
386990	TCP	192.168.1.1	50833	10.0.0.10	5433	opn01:55:06.5999	Fclt01:55:07.7602	cltF cR	1.1640	4	15		18.700	20		36.392
387490	TCP	192.168.1.1	50834	10.0.0.10	5433	opn01:52:14.8552	Fclt01:52:15.2349	cltF cR	0.3819	4	15		21.696	20		42.224
387490	TCP	192.168.1.1	50834	10.0.0.10	5433	opn01:55:06.5999	Fclt01:55:07.7461	cltF cR	1.1489	4	14		18.482	21		37.285
387990	TCP	192.168.1.1	50835	10.0.0.10	5433	opn01:52:14.8576	Fclt01:52:15.1515	cltF cR	0.2978	4	15		21.632	20		43.504
387990	TCP	192.168.1.1	50835	10.0.0.10	5433	opn01:55:06.6176	Fclt01:55:07.8253	cltF cR	1.2099	4	15		17.965	21		35.387

Connections – Client Port Numbers

This is the Connections Table, sorted by client port number. We can see that the Web Server begins at 49155 when it “recycles” its ephemeral port numbers.

Also that new connections are being made very rapidly (just a few milliseconds apart).

As we observed in the previous slide, it takes just over 3 minutes to cycle through the ports 49155-65534.

Here’s where the port numbers “recycle”.

ConnID	Type	Client (Call...	cPort	Server (Call...	sP...	First Packet	Closing	Clos...	Total sec	Tri...	Clt Pkts	ReTx	SelAck	Kbps	Svr Pkts	ReTx	SelAck	Kbps
160571	wrangleT/TCP	WebAppSvr	65531	ContentSvr	5433	opn01:51:43.2922	Fclt01:51:43.3674	cltF cR	0.0760	4	12			19.840	11			32.624
160572	wrangleT/TCP	WebAppSvr	65532	ContentSvr	5433	opn01:51:43.2944	Fclt01:51:43.4836	cltF cR	0.1921	4	15			21.632	20			43.504
160573	wrangleT/TCP	WebAppSvr	65533	ContentSvr	5433	opn01:51:43.2983	Fclt01:51:43.5137	cltF cR	0.2169	4	16			22.208	19			41.712
160574	wrangleT/TCP	WebAppSvr	65534	ContentSvr	5433	opn01:51:43.3040	Fclt01:51:43.5148	cltF cR	0.2128	4	15			21.696	19			41.712
144195	wrangleT/TCP	WebAppSvr	49155	ContentSvr	5433	opn01:51:43.3049	Fclt01:51:43.4370	cltF cR	0.1342	4	12			20.352	12			33.136
144196	wrangleT/TCP	WebAppSvr	49156	ContentSvr	5433	opn01:51:43.3084	Fclt01:51:43.5945	cltF cR	0.2889	4	15			21.696	19			41.712
144197	wrangleT/TCP	WebAppSvr	49157	ContentSvr	5433	opn01:51:43.3146	Fclt01:51:43.5057	cltF cR	0.1937	4	16			22.144	19			42.992
144198	wrangleT/TCP	WebAppSvr	49158	ContentSvr	5433	opn01:51:43.3229	Fclt01:51:43.5553	cltF cR	0.2337	4	16			22.208	19			41.712

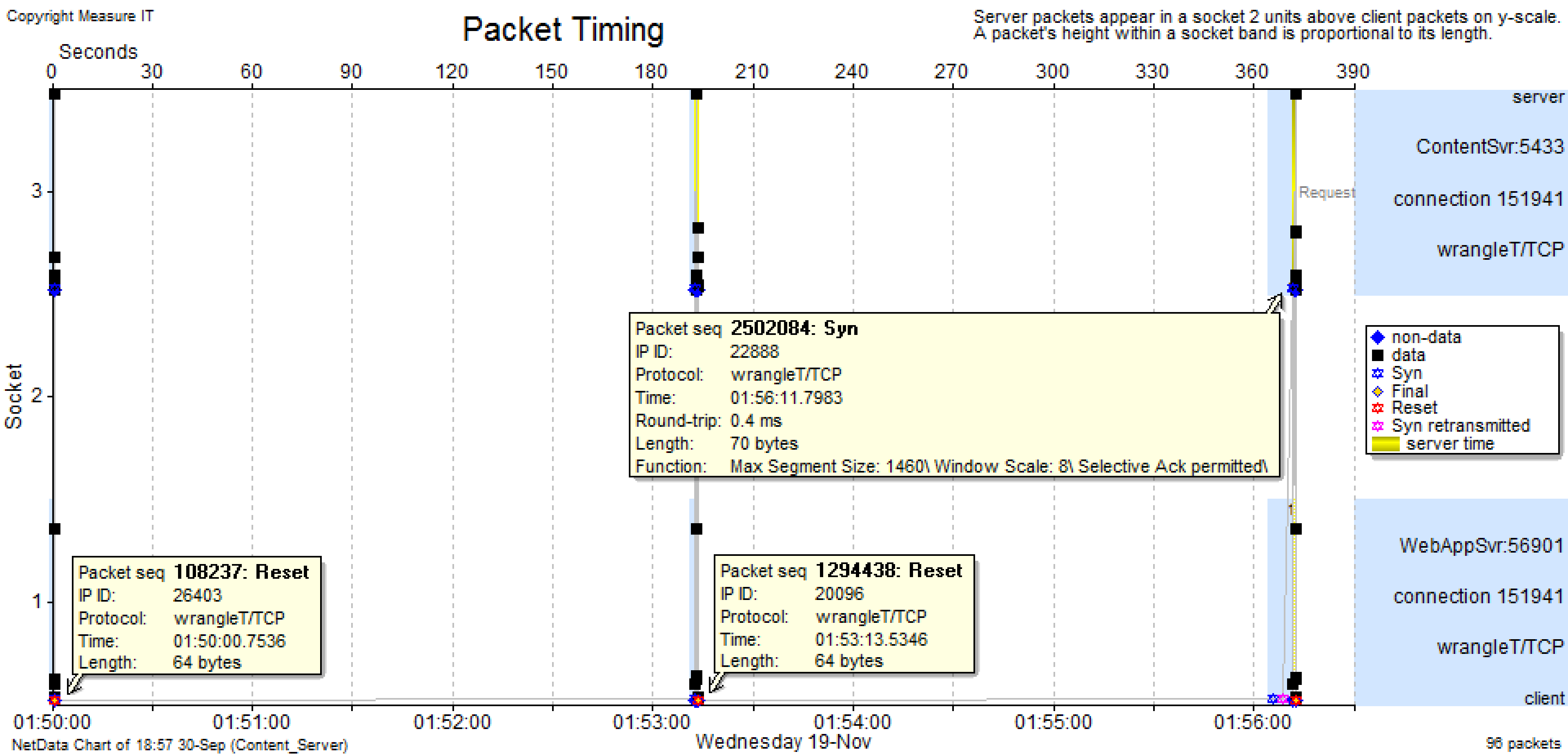
ConnID	Type	Client (Call...	cPort	Server (Call...	sP...	First Packet	Closing	Clos...	Total sec	Tri...	Clt Pkts	ReTx	SelAck	Kbps	Svr Pkts	ReTx	SelAck	Kbps
160571	wrangleT/TCP	WebAppSvr	65531	ContentSvr	5433	opn01:54:50.3093	Rclt01:54:51.1668	cltR	0.8574	3	11			19.840	12			33.136
160572	wrangleT/TCP	WebAppSvr	65532	ContentSvr	5433	opn01:54:50.3207	Fclt01:54:51.2942	cltF cR	0.9769	4	15			21.696	21			42.736
160573	wrangleT/TCP	WebAppSvr	65533	ContentSvr	5433	opn01:54:50.3278	Fclt01:54:51.1818	cltF cR	0.8558	4	12			20.352	13			33.648
160574	wrangleT/TCP	WebAppSvr	65534	ContentSvr	5433	opn01:54:50.3353	Fclt01:54:51.1846	cltF cR	0.8510	4	12			20.352	13			33.648
144195	wrangleT/TCP	WebAppSvr	49155	ContentSvr	5433	opn01:54:50.3389	Fclt01:54:51.1459	cltF cR	0.8095	4	12			19.840	13			33.648
144196	wrangleT/TCP	WebAppSvr	49156	ContentSvr	5433	opn01:54:50.3676	Fclt01:54:51.2021	cltF cR	0.8363	4	12			20.352	13			33.648
144197	wrangleT/TCP	WebAppSvr	49157	ContentSvr	5433	opn01:54:50.3676	Fclt01:54:51.3422	cltF cR	0.9766	4	16			22.208	21			42.736
144198	wrangleT/TCP	WebAppSvr	49158	ContentSvr	5433	opn01:54:50.3708	Fclt01:54:51.2204	cltF cR	0.8520	4	12			20.352	13			33.648

Client Port Recycling

Here is an example of a client port number (56901) that was used 3 times during the 4 test runs.

This chart is 6 minutes wide, making each individual connection “compressed” into a very narrow column. I’ve popped-up just one packet in each one in order to see the exact times.

We can see that the recycle time is just over 180 secs.



Additional Application Performance Observations

So far we've covered the cause of the transaction failures – which was needed to answer the question(s) posed in the original "Megalodon Challenge".

Some other interesting application behaviours became apparent in that analysis.

Next, we'll examine other interesting performance characteristics of the applications and network components.

I've never come across some of these behaviours before – and for some I can't even hypothesise the mechanism(s) that might cause them.

I hope you also find these interesting.

If you have ideas as to the potential causes – or if you have experienced similar behaviours in your packet analysis career, please let me know!

Phil@NetworkDetective.com.au.

Commentary

A typical connection request to the Content Server involves:

- TCP 3-way handshake.
- First data exchange: 127 byte request – 3031 byte response (possibly an SSL key exchange).
- Second data exchange: 158 – 51 (possibly SSL cypher or user authentication?).
- Third data exchange: 14xx – 52K/983/~1K (likely the main HTTP GET/POST or similar request).
- Termination by: Client 29-bytes data - Final – server Ack – server 29-bytes data – client Reset. (which has the flavour of an SSL "Alert" session termination).

There usually seems to be just one "real" request in each connection (HTTP GET/POST?). This allows us to use the "total connection duration" as a proxy for server application response time.

Further, comparing the time components of the "SSL handshake" versus the "HTTP request/response". The "SSL key exchange" takes significantly longer during the heavier load tests.

The Content Server appears to struggle during the heaviest load test.
This could be due to:

- Server limit on application threads available to process these functions (most likely).
- Server limit on TCP connections (not apparent in capture).
- A difficulty with client port recycle times (not apparent in capture).
- Load balancer limitations

The packet losses could be caused by the load balancer, other network equipment or the Content Server itself.

Performance Recommendations

1) The Content Server configuration be investigated and rectified for any limitations on:
   - Concurrent threads for HTTP/HTTPS processing.
   - TCP connections (e.g. a maximum concurrent connection limit).
   - Port "recycling" ability.

   Although port "recycling" doesn't appear to be an issue, our observed 170 seconds is less than the TCP "TIME_WAIT" maximum of 240 seconds. The Web Server could be reconfigured to use a larger range of ephemeral ports for its outgoing connections. That is, begin the cycles with a client port number lower than 49155.

   This would protect against the possibility of running into a future server "TIME_WAIT" connection problem if real life loads ever increase above the levels of the fourth test run.

2) The application behaviour be investigated, with a view to improving performance.
   - The first & second transaction in every connection are the same.
   - They appear to be SLL related.
   - If so, then the 3031-byte certificate is delivered in full for every connection.
   - Investigate the possibility of utilising any of the SSL "session reuse" options so that subsequent connections don't always incur this setup overhead.

3) Investigate reducing the number of connections.
   - After the same first & second transactions, there is usually just one other transaction in every connection.
   - If the Web Server application could be modified to perform more than one request per connection, then the "per connection" overheads (transactions 1 & 2 - which can be significant) would be significantly reduced.

Performance Recommendations (Cont.)

4) Investigate the mechanism that causes the regular "gap" periods of 0.3 seconds, every 4.5 seconds or so. BOTH the Web Server and Content Server applications seem to stop communicating or responding – but the TCP stacks still operate properly.
This represents approximately 6% of the total time.

- Why would this mechanism occur simultaneously in both servers?

5) Investigate the reason for the Content Server responses not being delivered with maximum efficiency (i.e., as a stream of full sized packets).
This could be related to internal server resource limitations, meaning that the only "fix" would be to move to a more powerful server.

6) Investigate the reason for the Web Server ramping up its outgoing connection count only at one second intervals.
It is apparent at the beginning of all test runs – but more visible in test run 4 - that the Load Generator's connections are all initiated at once (with the corresponding number of transaction requests). However, the Web Server begins with around half the number of connections to the Content Server. This means that the initial Load Generator transactions in each run are always slower than the subsequent transactions.

If more back-end connections were initiated up front, this queuing effect would be reduced.

Performance Evidence Trail

We'll first look at the Web Server to Content Server traffic flows, connections and transactions.

These have been broken down and examined for each of the four test runs, showing the differing behaviours under the different loads.

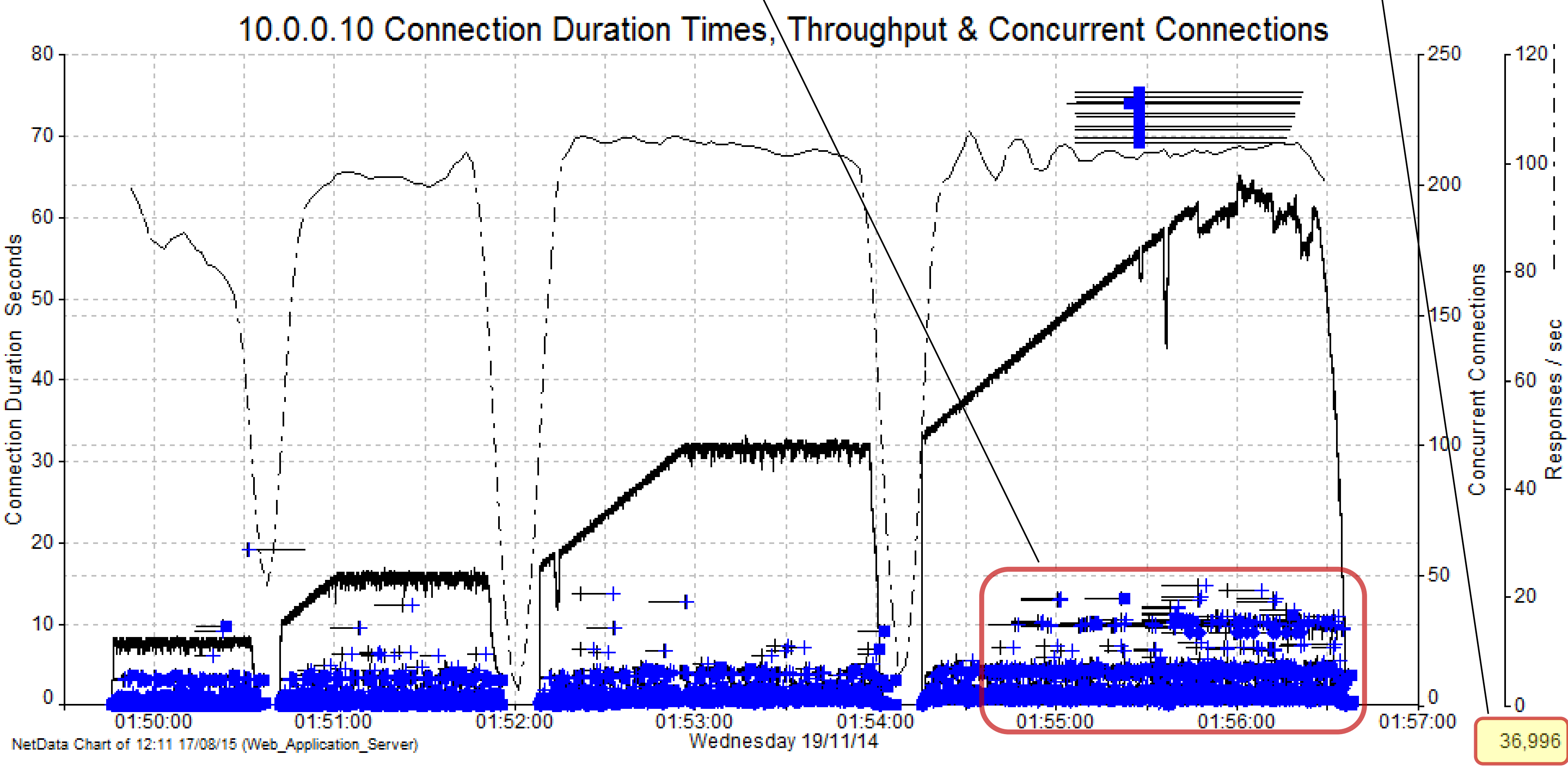
Later on, there is an investigation into some interesting application "sleeping" behaviour that seems synchronised between these two servers.

Content Server Connection Total Durations

This chart shows the connection duration details (to the Content Server) for the full 7 minutes. The blue markers now represent total durations of TCP connections (not just the setup times). Longer durations are higher up on the chart (LHS scale). The dashed line now represents “connections per second” (RHS scale). Connections are longer after 1:54:30, suggesting that the server takes longer to respond to requests.
We need to zoom down to see the data for most connections more clearly.

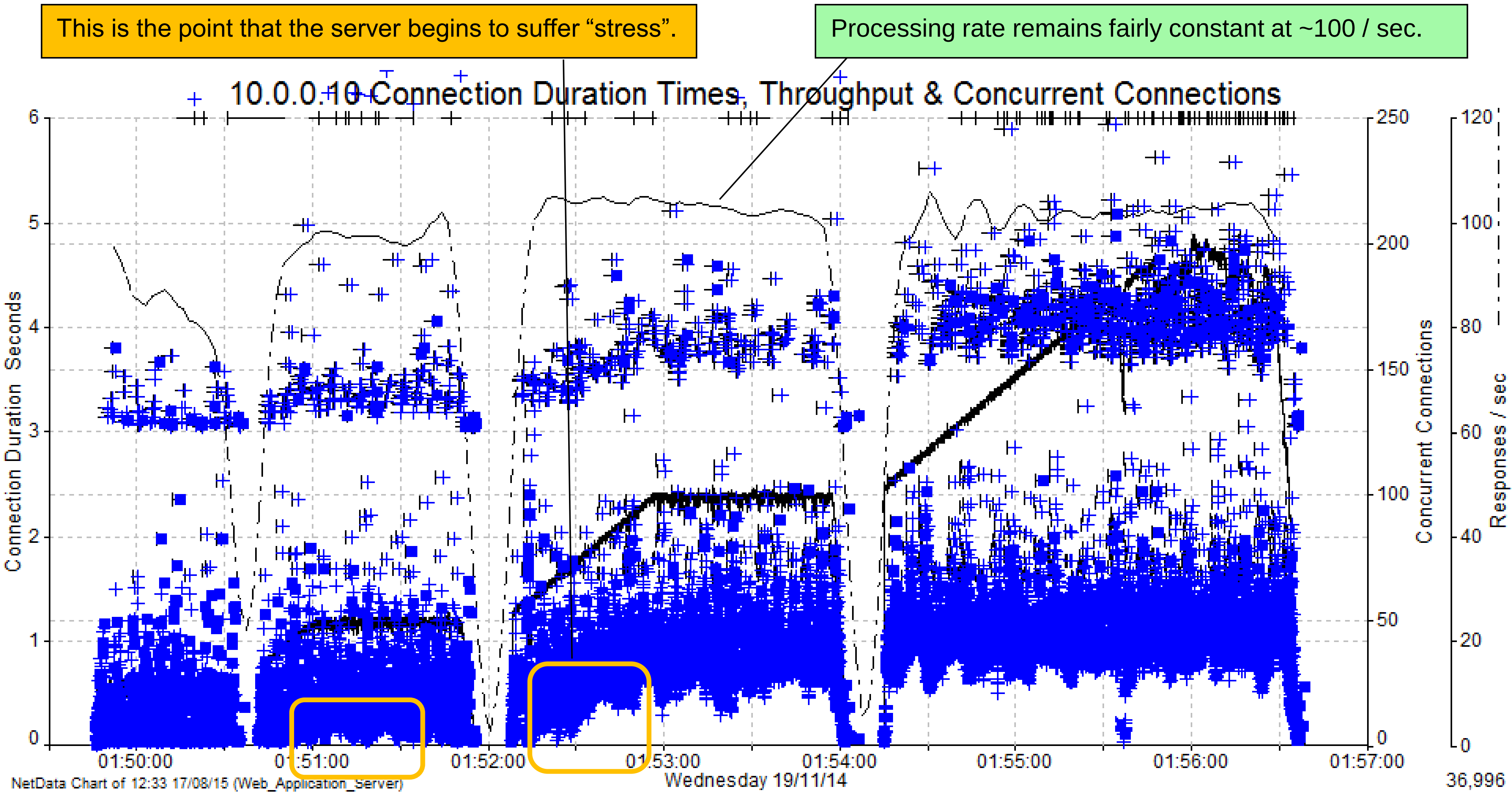
The most “interesting” activity seems to be here, during the heaviest test.

No. of connections on the chart.



Content Server Connection Total Durations

The LHS scale is now just 6 seconds high. We can see that there are very short (near zero) connections during tests 1. However, during test 2, there is some "white space" indicating longer times.
In tests 3 & 4, very few connections are under half a second (forming a "cloud base" on the chart) – and there are many more above 3-4 seconds.
Note that the server seems incapable of processing more than ~100 connections per second (RHS scale).



Every Connection Has Same First & Second Transaction

This snippet from the top of the Statistics table shows the transactions in order of frequency. It also shows the colours (LHS) that will be used on the following slides. It is sorted by the "Count" column.
The pink transaction in the top row (127 byte request – 3031 byte response) is the first one in every successful connection. The orange one (158 – 51) is always the second transaction. The transactions with 52 KB (pale blue) and 983 byte responses (red) are very common (but are spread throughout this table due to their varying packet size characteristics).

	ID	Transaction Description	Plot	Clt Avg	Count	Req Bytes	SecsMin	Average	Maximum	Rsp Bytes	End Avg	End Max
	3	Request: blk[127]--blk[3xxx]	Yes		36829	127.0	0.0088	0.498	4.056	3031.0	0.518	4.241
	2	Open Conn: Syn WScale: 8 SelAck MSS:1460--Syn-Ack MSS:1460 SelAck WScale: 8 - Ack	Yes		36826	70.0	0.0000	0.019	3.026	70.0	0.160	6.026
	4	Request: blk[158]--blk[6]; blk[4x]	Yes	0.002	34584	158.0	0.0000	0.008	1.185	51.0	0.011	1.432
	1	Request: blk[29]; conn half closed--blk[2x]	Yes	0.000	32000	29.0	0.0000	0.004	3.063	29.0	0.004	3.063
	5	Request: blk[178]; blk[1284]--blk[2xx]; blk[2x] (5); blk[4xx]; blk[2x] (3); blk[3x]	Yes	0.000	10281	1462.0	0.0322	0.217	3.064	983.0	0.220	3.066
	10	Request: blk[170]; blk[1284]--blk[4xx]; blk[2x] (5); blk[4xx]; blk[2x] (3); blk[3x]	Yes	0.000	6687	1454.0	0.0290	0.163	1.184	1143.0	0.165	1.187
	11	Request: blk[138]; blk[1284]--blk[2xx]; blk[2x]; blk[3x]	Yes	0.000	3734	1422.0	0.0107	0.046	10.496	311.0	0.048	10.497
	9	Request: blk[202]; blk[1284]--blk[2xx]; blk[2x]; blk[3x]	Yes	0.000	3706	1486.0	0.0227	0.105	1.199	311.0	0.107	1.201
	20	Request: blk[158]--blk[5x]	Yes	0.002	1888	158.0	0.0012	0.040	9.980	51.0	0.048	9.980
	18	Request: blk[29]; conn half closed; conn already half closed by client--blk[2x]	Yes	0.000	1661	29.0	0.0000	0.003	0.303	29.0	0.003	0.303
	35	Request: blk[154]; blk[1284]--blk[4xx]; blk[2x] (5); blk[3xxxx]; blk[2x] (6); blk[1xxxx]; blk[2x] (2); blk[3x]	Yes	0.000	1273	1438.0	0.0942	0.304	1.003	52557.0	0.328	2.489
	17	Request: blk[178]; blk[1284]--blk[2xx]; blk[2x] (4); blk[4xx]; blk[2x] (2); blk[3x]	Yes	0.000	863	1462.0	0.0345	0.220	1.197	983.0	0.223	1.200
	37	Request: blk[170]; blk[1284]--blk[4xx]; blk[2x] (4); blk[4xx]; blk[2x] (3); blk[3x]	Yes	0.000	858	1454.0	0.0333	0.170	1.113	1143.0	0.173	1.114
	36	Request: blk[154]; blk[1284]--blk[4xx]; blk[2x] (5); blk[3xxxx]; blk[2x] (7); blk[1xxxx]; blk[2x] (2); blk[3x]	Yes	0.000	630	1438.0	0.1086	0.290	1.175	52557.0	0.314	1.190
	21	Request: blk[178]; blk[1284]--blk[2xx]; blk[2x] (5); blk[4xx]; blk[2x] (2); blk[3x]	Yes	0.000	490	1462.0	0.0402	0.229	1.326	983.0	0.232	1.328
	26	Request: blk[202]; blk[1284]--blk[2xx]; blk[3x]	Yes	0.000	406	1486.0	0.0247	0.112	1.024	311.0	0.114	1.026
	57	Request: blk[138]; blk[1284]--blk[2xx]; blk[3x]	Yes	0.000	365	1422.0	0.0122	0.049	0.662	311.0	0.052	0.663
	32	Request: blk[170]; blk[1284]--blk[4xx]; blk[2x] (5); blk[4xx]; blk[2x] (3); blk[3x]; blk[2x]	Yes	0.000	289	1454.0	0.0286	0.094	0.690	1172.0	0.097	0.696
	22	Request: blk[178]; blk[1284]--blk[2xx]; blk[2x] (5); blk[4xx]; blk[2x] (3); blk[3x]; blk[2x]	Yes	0.000	268	1462.0	0.0333	0.142	1.013	1012.0	0.144	1.013
+	310	Open Conn: Syn WScale: 8 SelAck MSS:1460--Syn-Ack MSS:1460 SelAck - Ack	Yes		265	70.0	0.0005	0.175	6.005	66.0	9.162	12.029
	60	Request: blk[178]; blk[1284]--blk[2xx]; blk[2x] (5); blk[4xx]; blk[2x] (2); blk[6x]	Yes	0.000	254	1462.0	0.0422	0.225	1.324	983.2	0.229	1.328
	6	Request: blk[29]--blk[2x]	Yes	0.000	241	29.0		0.000	0.002	29.0	0.000	0.002
	67	Request: blk[127]; blk[158]--blk[3xxx]; blk[6]; blk[4x]	Yes		200	285.0	0.0006	0.025	0.920	3082.0	0.025	0.920
	121	Request: blk[170]; blk[1284]--blk[4xx]; blk[2x] (5); blk[4xx]; blk[2x] (2); blk[6x]	Yes	0.000	189	1454.0	0.0431	0.181	0.928	1143.0	0.184	0.931
	66	Request: blk[154]; blk[1284]--blk[4xx]; blk[2x] (5); blk[3xxxx]; blk[2x] (5); blk[1xxxx]; blk[2x] (2); blk[3x]	Yes	0.000	151	1438.0	0.1120	0.301	0.949	52557.1	0.475	1.660
	42	Request: blk[138]; blk[1284]--blk[2xx]; blk[2x]; blk[3x]; blk[2x]	Yes	0.000	149	1422.0	0.0113	0.039	0.928	340.0	0.042	0.929
	206	Request: blk[138]; blk[1284]--blk[2xx]; blk[6x]	Yes	0.000	139	1422.0	0.0136	0.038	0.348	312.0	0.049	0.361
	108	Request: blk[202]; blk[1284]--blk[2xx]; blk[6x]	Yes	0.000	138	1486.0	0.0290	0.118	0.682	311.4	0.125	0.684
	58	Request: blk[157]--blk[6]; blk[4x]	Yes	0.002	134	157.0	0.0013	0.010	0.314	51.0	0.014	0.530
	7	Request: blk[202]; blk[1284]--blk[2xx]; blk[2x]; blk[3x]; blk[2x]	Yes	0.000	130	1486.0	0.0228	0.062	0.370	340.0	0.063	0.371
	24	Request: blk[154]; blk[1284]--blk[4xx]; blk[2x] (5); blk[3xxxx]; blk[2x] (6); blk[1xxxx]; blk[2xxx]; blk[2x] (2); ...	Yes	0.000	115	1438.0	0.1229	0.312	1.107	52557.2	0.337	1.748

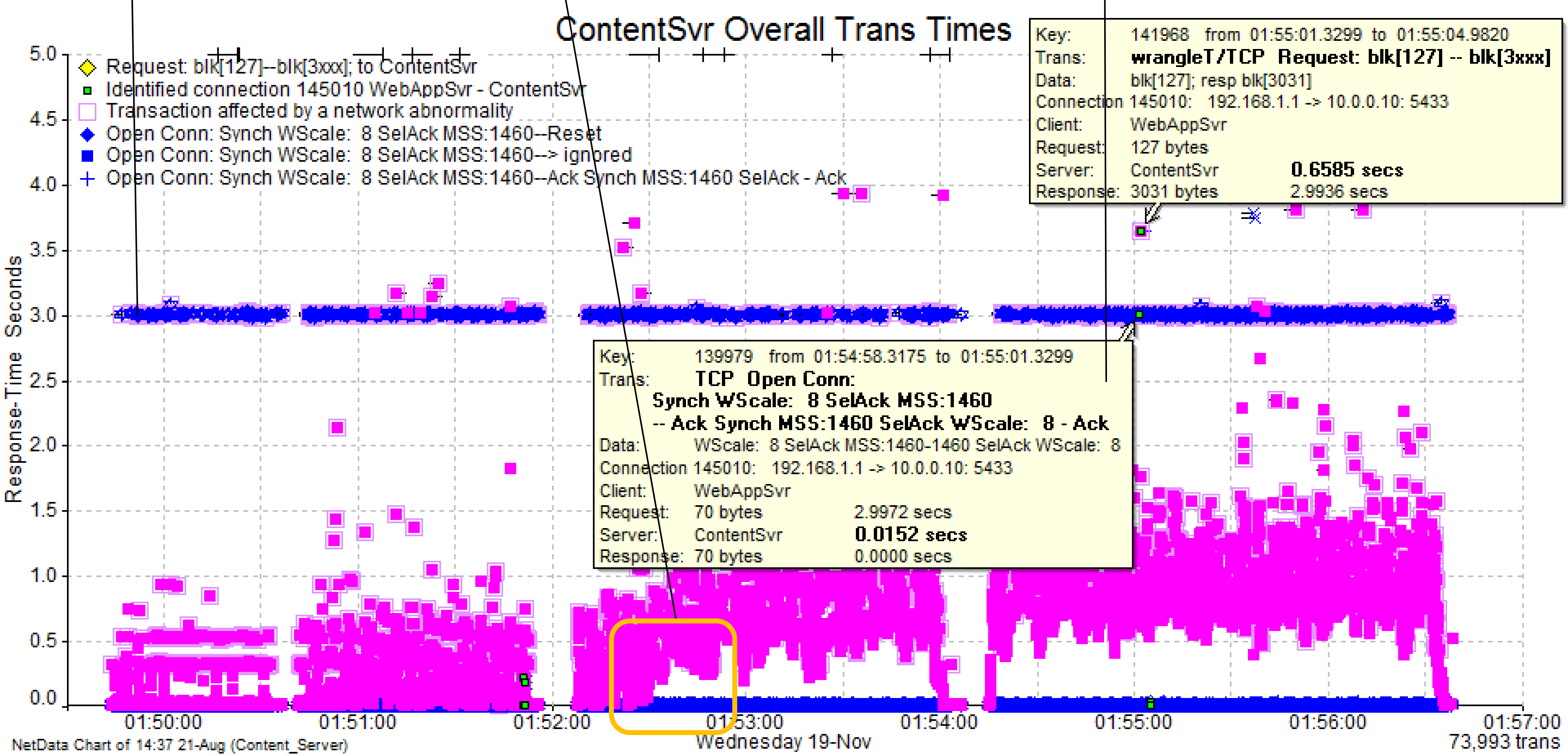
The First Transaction in Every Connection

This chart plots only TCP connection setups (blue) plus just that one transaction type (solid pink square).
The hollow pink squares surround transactions containing packet losses.
The 2 popups show an example of a connection that suffered two 3 second delays, one in the 3-way setup “transaction” and one in the data delivery transaction.

These are all 3 second TCP setups.

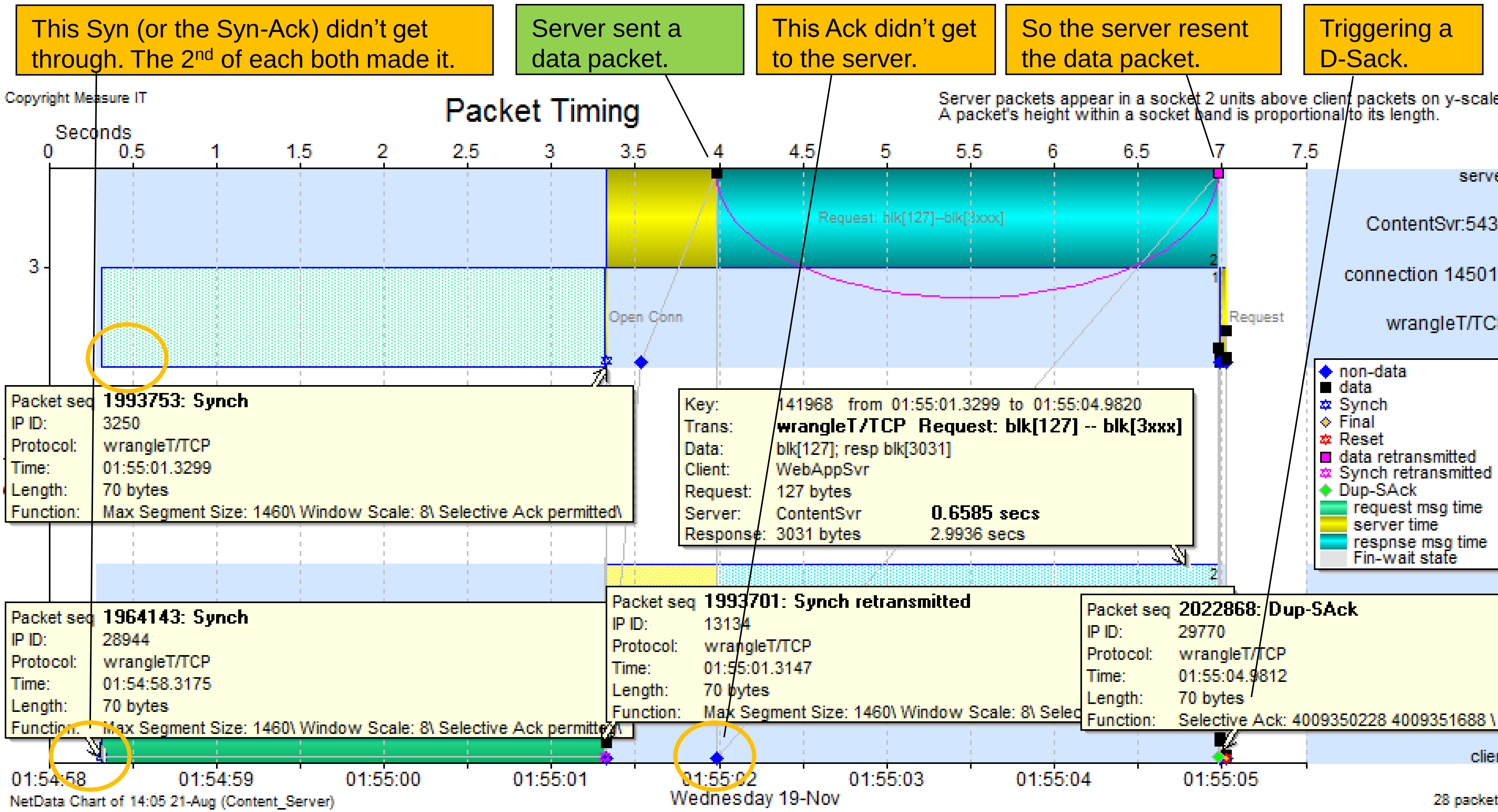
This is the point that the server begins to suffer “stress”. Always with this first transaction type.

On the next slide, we’ll see the packets in this connection + transaction.



3 Sec Setup + 0.5 Sec Trans + 3 Sec Retrans = 6.5 Secs

In this example, we see a lost Syn adding 3 seconds, then a data packet (black square) being retransmitted (pink square) - adding a further 3 seconds to what was a half second response. This is because the client's Ack (blue diamond) was lost on the way back to the server. (The pink loop shows the data->retrans linkage). Note the client's Duplicate Selective Ack (green diamond) informing the server that the retransmission was unnecessary.



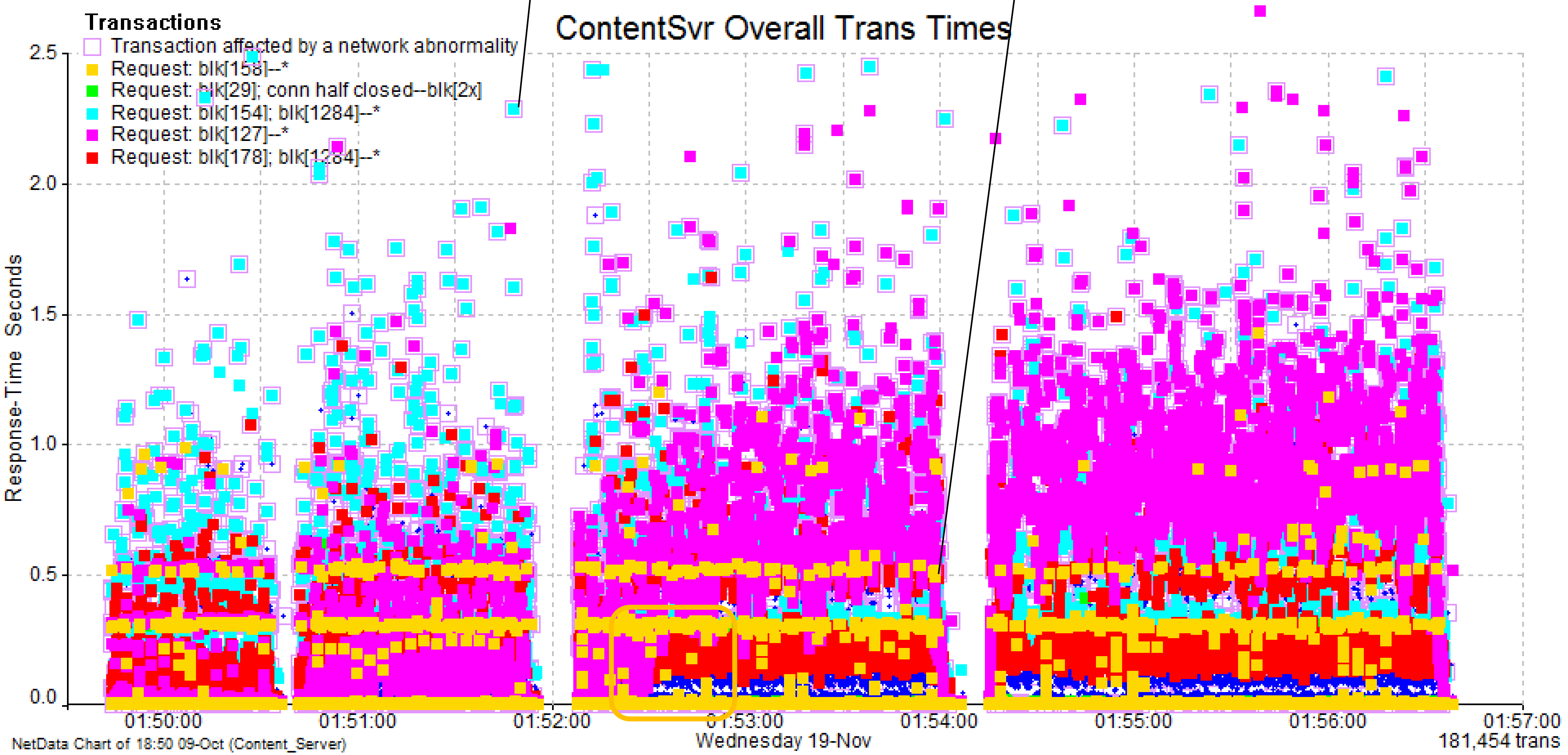
The Second & Other Transactions

This chart plots all the transaction types. The orange ones are always the second transaction in every connection. They make an interesting pattern of horizontal bands across all tests, showing that there is some mechanism causing regular occurrences of 300ms, 500ms and 900ms responses. Most are very fast though. The pale blue transactions are the 52 KB transactions, red are 983 bytes. Only the pink ones seem to be significantly affected by load.

As usual, the hollow pink squares surround transactions containing packet losses.

These are all 52 KB responses, taking longer when affected by packet losses.

The orange [158 – 51] transactions are often 300m, 500ms or more – independently of load.



The Transaction Groupings in Every Connection

This is a snippet of the Transactions table, sorted by connections (ConnID).
We see the regular groupings of:
[TCP Setup], [127—3031], [158—51], [Varying Transaction], [29—29] (closure).

The timing durations are also broken up into request/server/response for each transaction.

Transaction colour as shown in the charts.

Just one connection in this table.

Trn Key	Request Strt	Resp End	Type	Description	Rqst Dur	Strt Rsp	End Rsp	Resp Dur	ConnID	Client	Server	Port	LRqst	LResp	Frame
89291	01:53:10.797667	01:53:10.799795	TCP	Open Conn: Syn WScale: 8 SelAck MSS:1460--Syn-Ack M...		0.0021	0.0021	0.0000	151771	WebAppSvr	ContentSvr	5433	70	70	1275335
89686	01:53:10.799796	01:53:11.501142	wrangleT/TCP	Request: blk[127]--blk[3xxx]		0.7007	0.7013	0.0007	151771	WebAppSvr	ContentSvr	5433	127	3031	1275347
89688	01:53:11.503124	01:53:11.505081	wrangleT/TCP	Request: blk[158]--blk[6]; blk[4x]		0.0020	0.0020	0.0000	151771	WebAppSvr	ContentSvr	5433	158	51	1280855
89764	01:53:11.505084	01:53:11.675965	wrangleT/TCP	Request: blk[170]; blk[1284]--blk[4xx]; blk[2x] (5); blk[4xx]; b...	0.0005	0.1685	0.1704	0.0019	151771	WebAppSvr	ContentSvr	5433	1454	1143	1280889
89769	01:53:11.675967	01:53:11.67891	wrangleT/TCP	Request: blk[29]; conn half closed--blk[2x]		0.0029	0.0029		151771	WebAppSvr	ContentSvr	5433	29	29	1282100
172173	01:56:03.966291	01:56:03.983379	TCP	Open Conn: Syn WScale: 8 SelAck MSS:1460--Syn-Ack M...		0.0171	0.0171	0.0000	151771	WebAppSvr	ContentSvr	5433	70	70	2443574
172669	01:56:03.983383	01:56:05.116693	wrangleT/TCP	Request: blk[127]--blk[3xxx]		1.1321	1.1333	0.0012	151771	WebAppSvr	ContentSvr	5433	127	3031	2443669
172671	01:56:05.119235	01:56:05.120794	wrangleT/TCP	Request: blk[158]--blk[6]; blk[4x]		0.0016	0.0016	0.0000	151771	WebAppSvr	ContentSvr	5433	158	51	2450022
172828	01:56:05.120796	01:56:05.410303	wrangleT/TCP	Request: blk[154]; blk[1284]--blk[4xx]; blk[2x] (5); blk[3xxxx]...	0.0005	0.2807	0.2890	0.0083	151771	WebAppSvr	ContentSvr	5433	1438	52561	2450028
172831	01:56:05.410305	01:56:05.413042	wrangleT/TCP	Request: blk[29]; conn half closed--blk[2x]		0.0027	0.0027		151771	WebAppSvr	ContentSvr	5433	29	29	2452039
6840	01:49:58.229417	01:49:58.243299	TCP	Open Conn: Syn WScale: 8 SelAck MSS:1460--Syn-Ack M...		0.0111	0.0139	0.0027	151772	WebAppSvr	ContentSvr	5433	70	70	95909
6952	01:49:58.2433	01:49:58.556925	wrangleT/TCP	Request: blk[127]--blk[3xxx]		0.0140	0.3136	0.2996	151772	WebAppSvr	ContentSvr	5433	127	3031	96177
6955	01:49:58.558722	01:49:58.561993	wrangleT/TCP	Request: blk[158]--blk[6]; blk[4x]		0.0033	0.0033	0.0000	151772	WebAppSvr	ContentSvr	5433	158	51	98257
7008	01:49:58.561995	01:49:58.662709	wrangleT/TCP	Request: blk[178]; blk[1284]--blk[2xx]; blk[2x] (5); blk[4xx]; b...	0.0006	0.0992	0.1002	0.0010	151772	WebAppSvr	ContentSvr	5433	1462	983	98281
7009	01:49:58.663089	01:49:58.664196	wrangleT/TCP	Request: blk[29]; conn half closed--blk[2x]		0.0011	0.0011		151772	WebAppSvr	ContentSvr	5433	29	29	98592
89301	01:53:10.80831	01:53:10.811514	TCP	Open Conn: Syn WScale: 8 SelAck MSS:1460--Syn-Ack M...		0.0029	0.0032	0.0003	151772	WebAppSvr	ContentSvr	5433	70	70	1275621
89690	01:53:10.811522	01:53:11.511488	wrangleT/TCP	Request: blk[127]--blk[3xxx]		0.6995	0.7000	0.0004	151772	WebAppSvr	ContentSvr	5433	127	3031	1275826
89696	01:53:11.513621	01:53:11.518517	wrangleT/TCP	Request: blk[158]--blk[6]; blk[4x]		0.0023	0.0049	0.0026	151772	WebAppSvr	ContentSvr	5433	158	51	1280942
89789	01:53:11.518821	01:53:11.719206	wrangleT/TCP	Request: blk[178]; blk[1284]--blk[2xx]; blk[2x] (5); blk[4xx]; b...	0.0005	0.1964	0.1999	0.0035	151772	WebAppSvr	ContentSvr	5433	1462	983	1281062
89791	01:53:11.719215	01:53:11.722121	wrangleT/TCP	Request: blk[29]; conn half closed--blk[2x]		0.0029	0.0029		151772	WebAppSvr	ContentSvr	5433	29	29	1282338
172174	01:56:03.966292	01:56:03.983381	TCP	Open Conn: Syn WScale: 8 SelAck MSS:1460--Syn-Ack M...		0.0171	0.0171	0.0000	151772	WebAppSvr	ContentSvr	5433	70	70	2443575
172675	01:56:03.983382	01:56:05.126836	wrangleT/TCP	Request: blk[127]--blk[3xxx]		1.1423	1.1435	0.0011	151772	WebAppSvr	ContentSvr	5433	127	3031	2443668
172677	01:56:05.128655	01:56:05.130815	wrangleT/TCP	Request: blk[158]--blk[6]; blk[4x]		0.0022	0.0022	0.0000	151772	WebAppSvr	ContentSvr	5433	158	51	2450057
172766	01:56:05.130817	01:56:05.323415	wrangleT/TCP	Request: blk[178]; blk[1284]--blk[2xx]; blk[2x] (5); blk[4xx]; b...	0.0006	0.1902	0.1920	0.0019	151772	WebAppSvr	ContentSvr	5433	1462	983	2450064
172770	01:56:05.323418	01:56:05.327576	wrangleT/TCP	Request: blk[29]; conn half closed--blk[2x]		0.0042	0.0042		151772	WebAppSvr	ContentSvr	5433	29	29	2451465
6846	01:49:58.269786	01:49:58.271606	TCP	Open Conn: Syn WScale: 8 SelAck MSS:1460--Syn-Ack M...		0.0018	0.0018	0.0000	151773	WebAppSvr	ContentSvr	5433	70	70	96241
6849	01:49:58.271607	01:49:58.288693	wrangleT/TCP	Request: blk[127]--blk[3xxx]		0.0133	0.0171	0.0037	151773	WebAppSvr	ContentSvr	5433	127	3031	96244
6853	01:49:58.29043	01:49:58.294221	wrangleT/TCP	Request: blk[158]--blk[6]; blk[4x]		0.0038	0.0038		151773	WebAppSvr	ContentSvr	5433	158	51	96279
6861	01:49:58.294224	01:49:58.308973	wrangleT/TCP	Request: blk[138]; blk[1284]--blk[2xx]; blk[2x]; blk[3x]	0.0005	0.0135	0.0143	0.0007	151773	WebAppSvr	ContentSvr	5433	1422	311	96317
6862	01:49:58.308976	01:49:58.30932	wrangleT/TCP	Request: blk[29]; conn half closed--blk[2x]		0.0003	0.0003		151773	WebAppSvr	ContentSvr	5433	29	29	96407
89307	01:53:10.819403	01:53:10.821353	TCP	Open Conn: Syn WScale: 8 SelAck MSS:1460--Syn-Ack M...		0.0019	0.0020	0.0000	151773	WebAppSvr	ContentSvr	5433	70	70	1275916
89692	01:53:10.821354	01:53:11.514653	wrangleT/TCP	Request: blk[127]--blk[3xxx]		0.6929	0.6933	0.0004	151773	WebAppSvr	ContentSvr	5433	127	3031	1275945

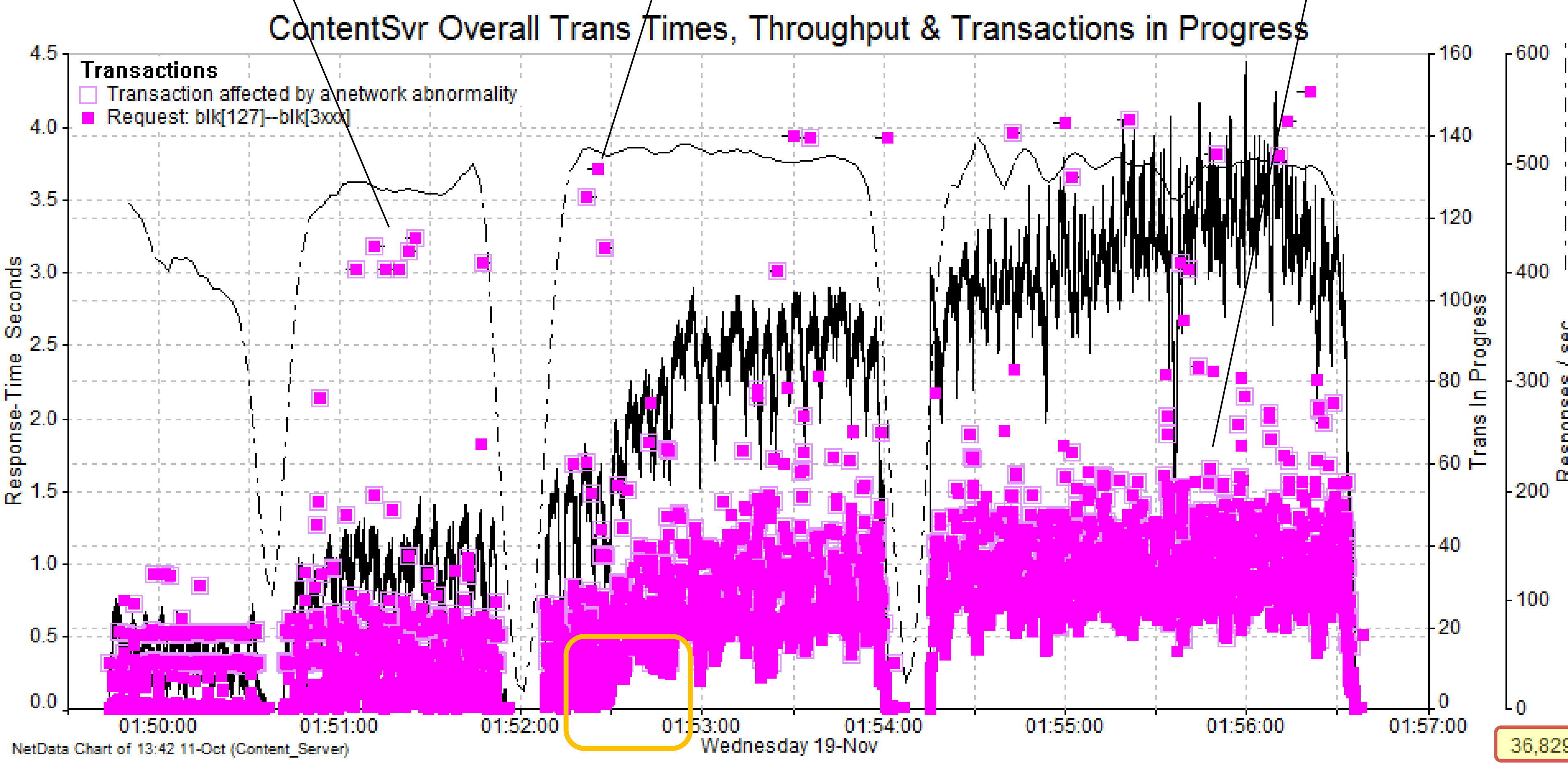
Just the First Transaction

This chart now plots just the 36,829 instances of the first common transaction type. The pink outlines here surround those transactions containing packet losses and retransmissions.
The solid black line is “Transactions in Progress” – which shows how many of these transactions were being processed in parallel at any one time. The dashed black line effectively counts “Transactions per Second”.
We see that around 100 is the maximum for both values. Improving this transaction would improve performance across all tests.

Packet losses can account for many of the longer durations.

But sometimes 3 or 4 seconds is due to the server “thinking”.

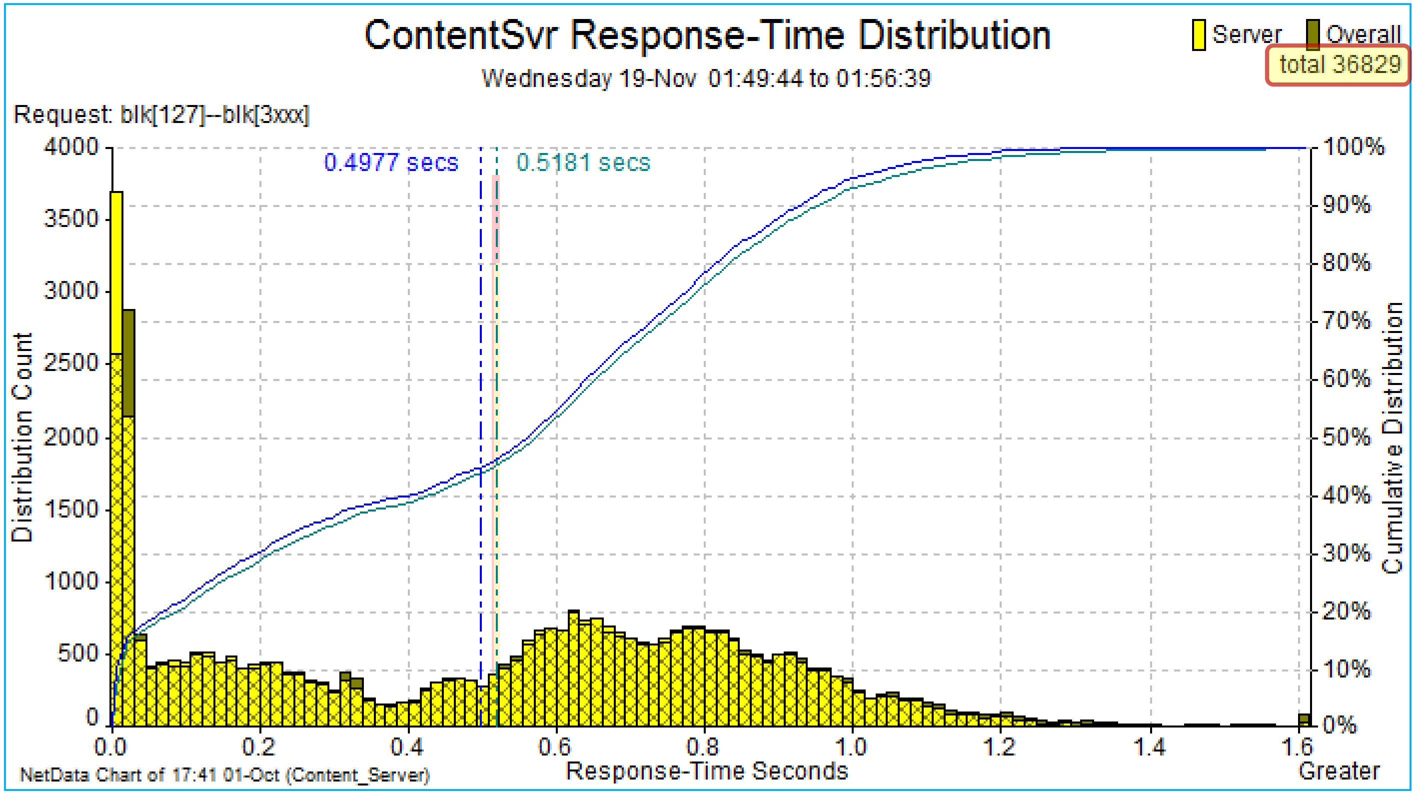
The fact that they all take significantly longer during the heavier tests is readily apparent.



Duration of the First Transactions

This frequency histogram of the [127]—[3031] transactions shows that most of them are under a second – with an average of half a second.

On the next slides are the same transaction charts and histograms – broken down by the individual test runs. There we see the increasing times as the test runs get progressively "heavier". Even better, we also see how the transaction times vary.

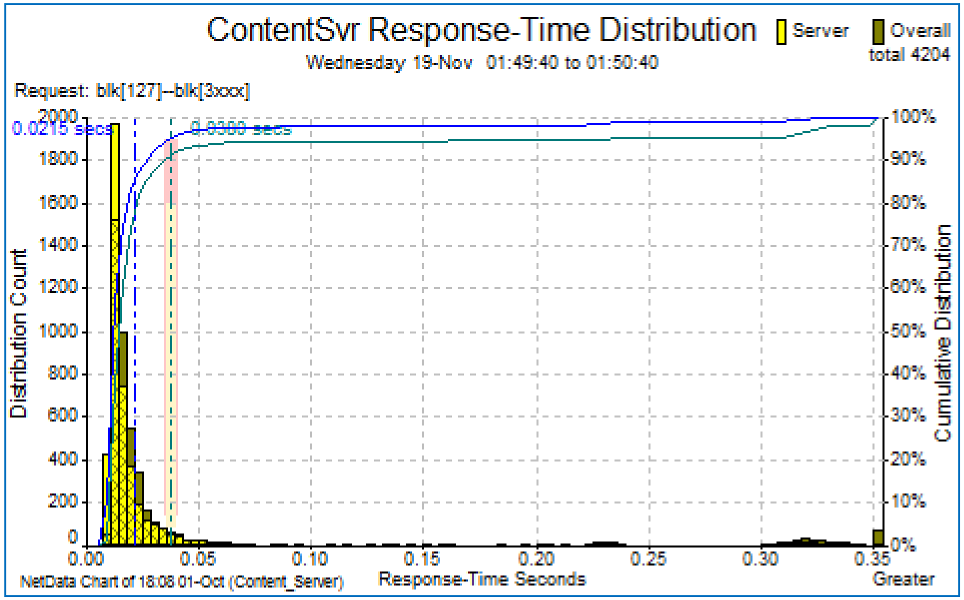


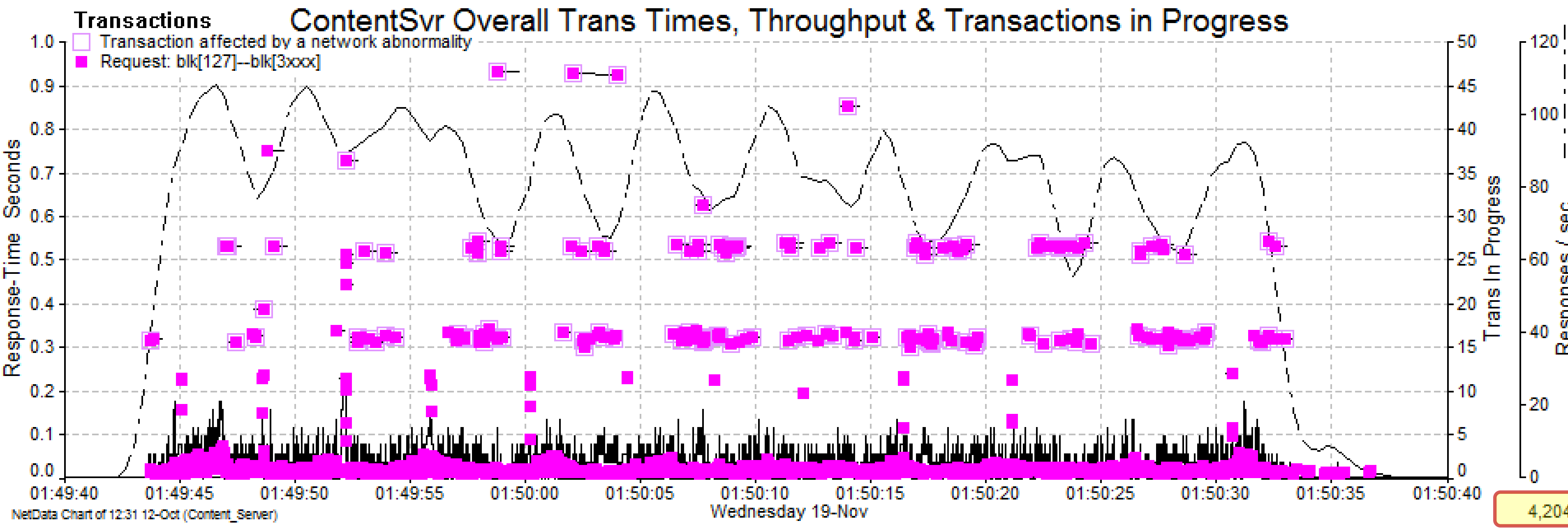
Response Time Histogram (Test Run 1)

In test run 1, we see that most first transactions are very fast but the server takes a little longer to handle some transactions.

90% of the 4204 are under 0.03 secs and the average is just 0.0215 secs.

Packet losses add 0.3, 0.5 and up to 0.9 seconds to each transaction.



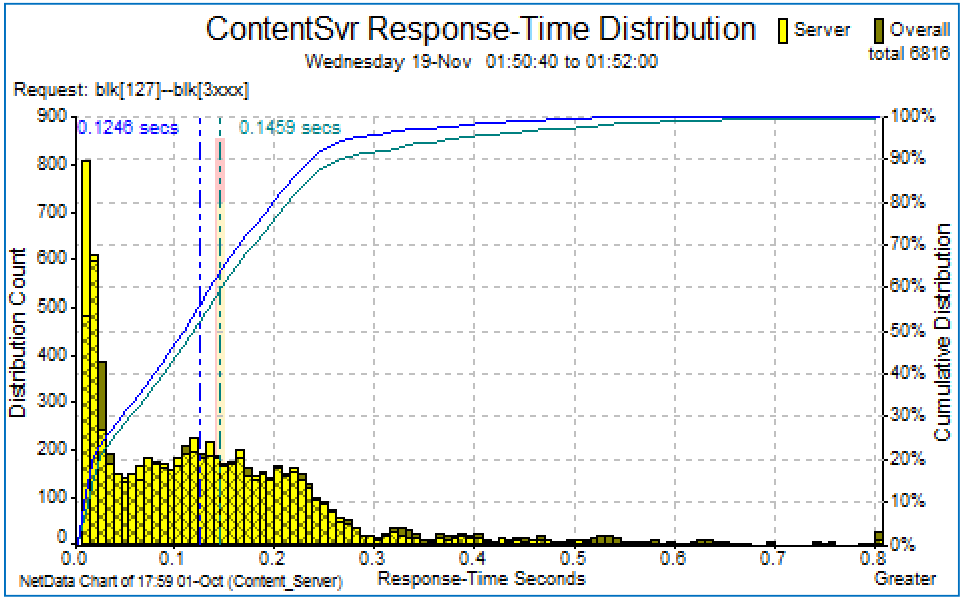


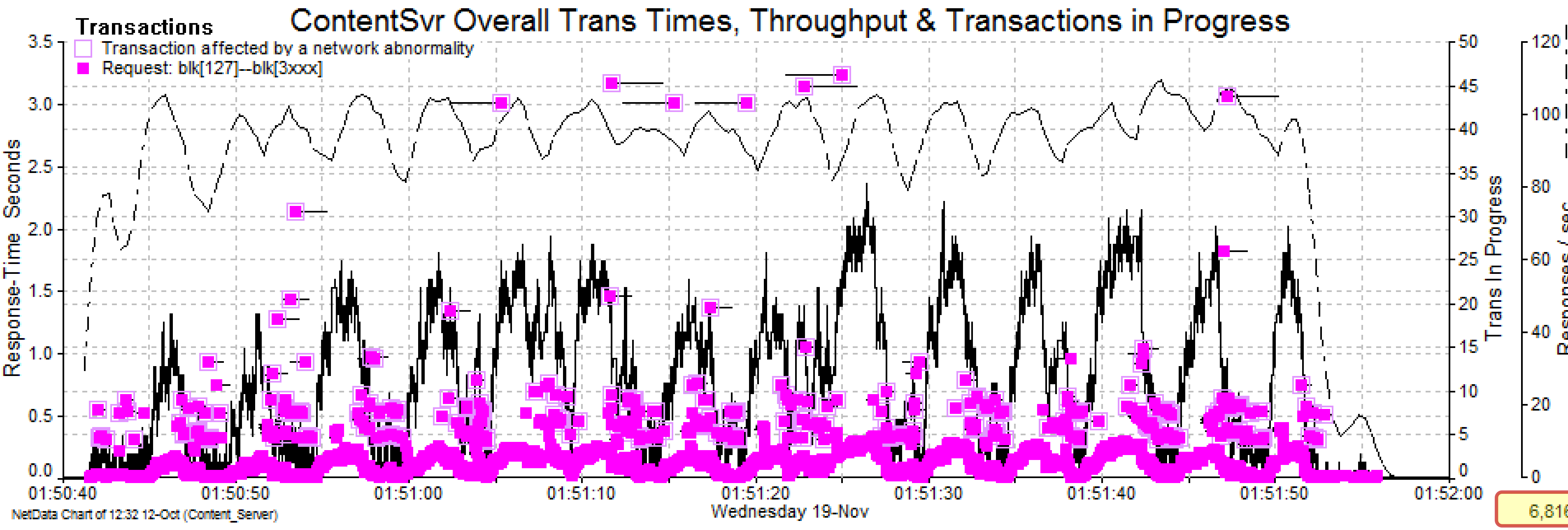
Response Time Histogram (Test Run 2)

Even in test run 2, we now see that the server takes longer to handle each of these transactions when it is already processing more than 10-15 of them. The see the "waves" as the load moves up and down.

Even so, 90% of the 6816 are under 0.3 secs and the average is 0.125 secs.

Packet losses add 1.5 to 3 seconds to each transaction.



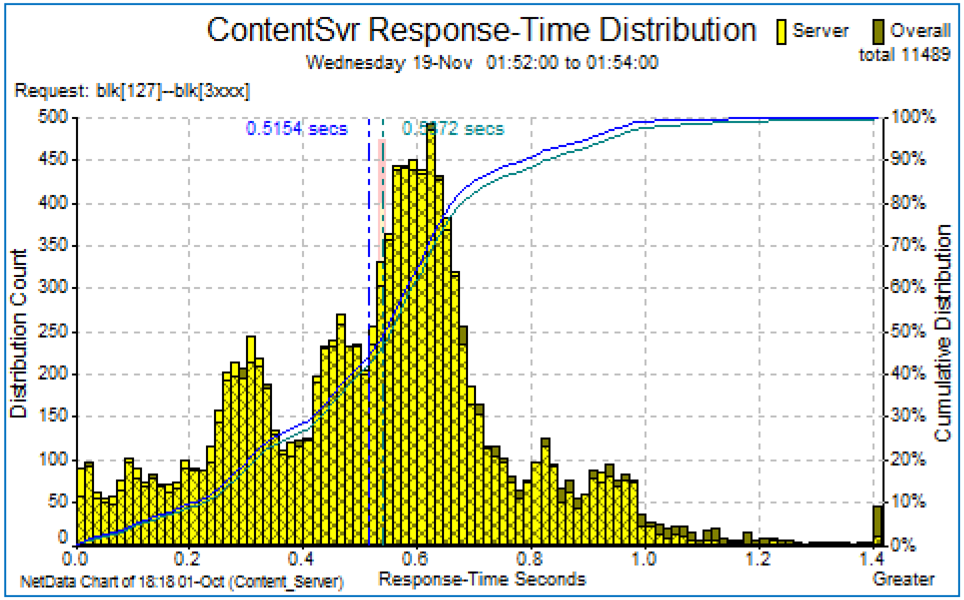


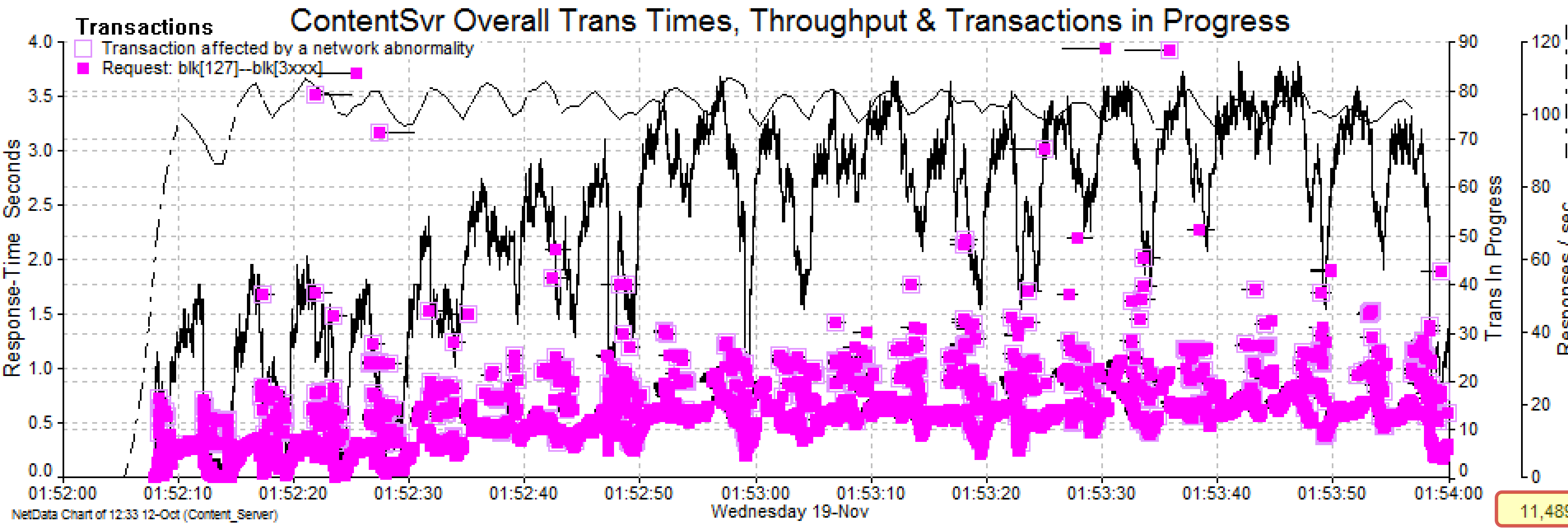
Response Time Histogram (Test Run 3)

In test run 3, we now see the initial "waves" but then the rise up off the bottom as the number of parallel transactions rises above 50. We still have "waves" but with a higher base.

Even so, 90% of the 11,489 are under 0.8 secs and the average is 0.515 secs.

Packet losses still add 1.5 to 3 seconds to each transaction.



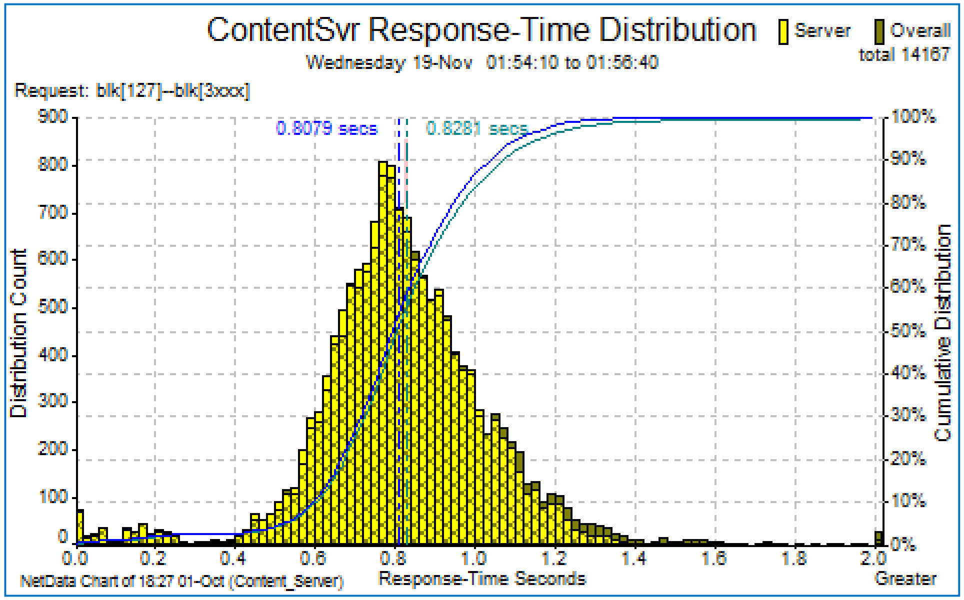


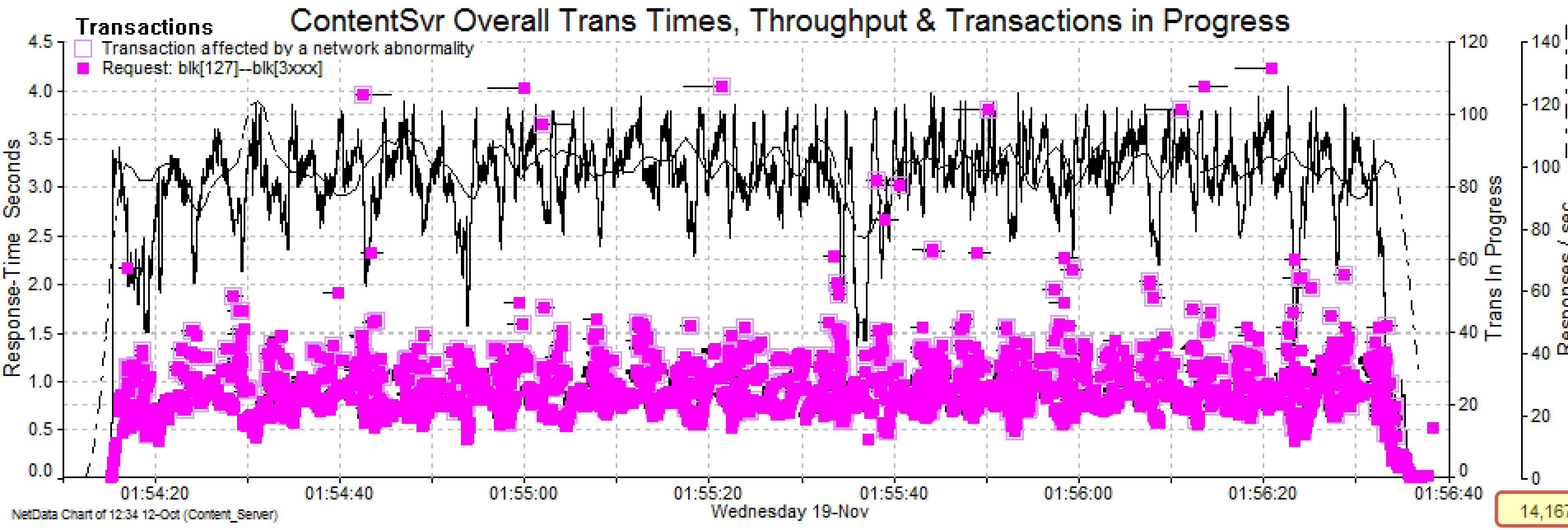
Response Time Histogram (Test Run 4)

In test run 4, we see the rise up off the bottom straight away as the number of parallel transactions rises quickly to 80. We still have the "waves" with a higher base.

Even so, 90% of the 14,167 are under 1.0 sec and the average is 0.8 secs. Interestingly, the shape of the histogram chart is narrower than test 3.

Packet losses still add seconds to each affected transaction.





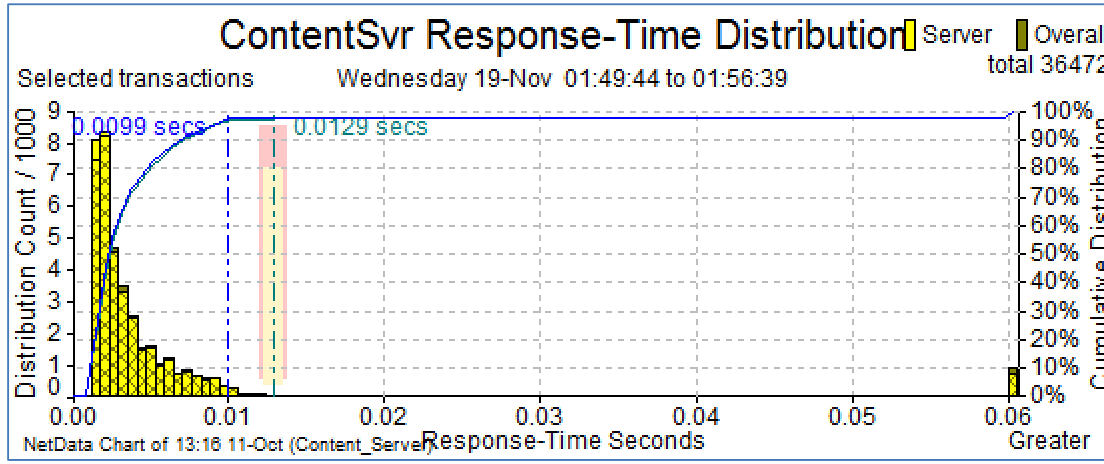
Response Time Histogram (2nd Transaction)

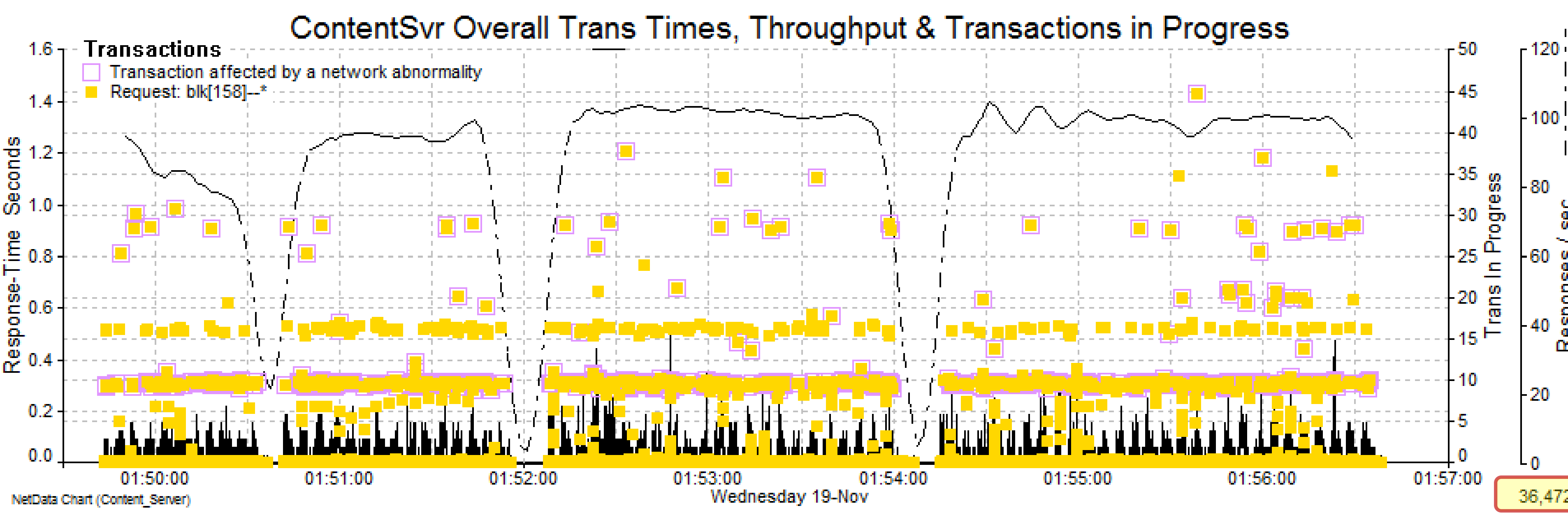
	ID	Transaction Description	Plot	Clt Avg	Count	Req Bytes	SecsMin	Average	Maximum	Rsp Bytes	End Avg	End Max
	4	Request: blk[158]--blk[6]; blk[4x]	Yes	0.002	36344	158.0	0.0000	0.009	9.980	51.0	0.012	9.980
	52	Request: blk[158]--blk[5x]	Yes	0.002	128	158.0	0.2874	0.320	1.205	51.0	0.320	1.205

Above we see that there are 2 “types” of the second transaction. Both types can be slow, many due to losses + retransmissions.

To the right, we see the majority are fast – but about 1000 of them are slow, dragging the mean out to 120ms.

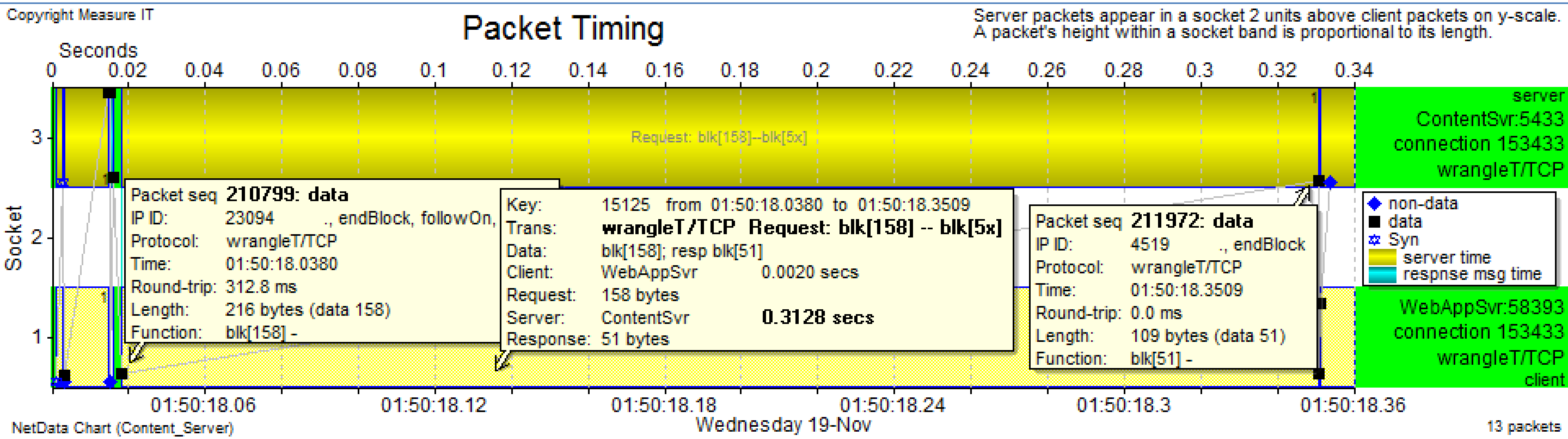
Below we see that they aren’t noticeably affected by load. The slower ones are consistent - not due to a random mechanism.

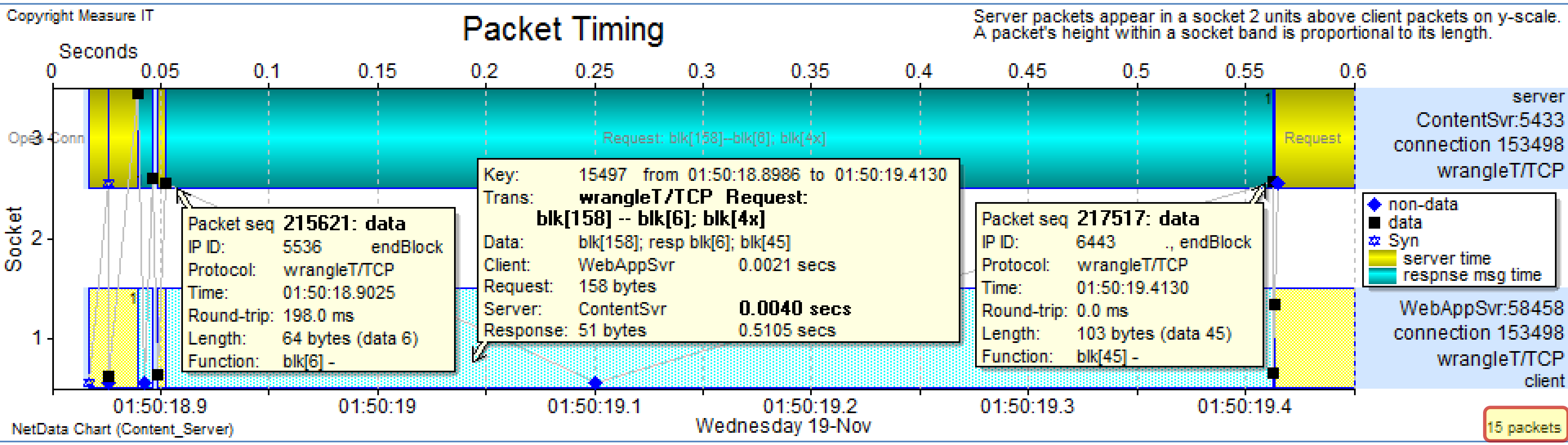




Second Transaction Types

The difference between the two “types” of the second transaction is whether the response is delivered in two packets or just one (which we would expect for just 51 bytes). Out of 36472, only 128 have a single packet response.
Below is one example of each type. First, the [158] request gets the full [51] after 300ms, second, the request gets [6] immediately, but the remaining [45] takes ~500ms, forcing a client delayed Ack at the 200ms mark.



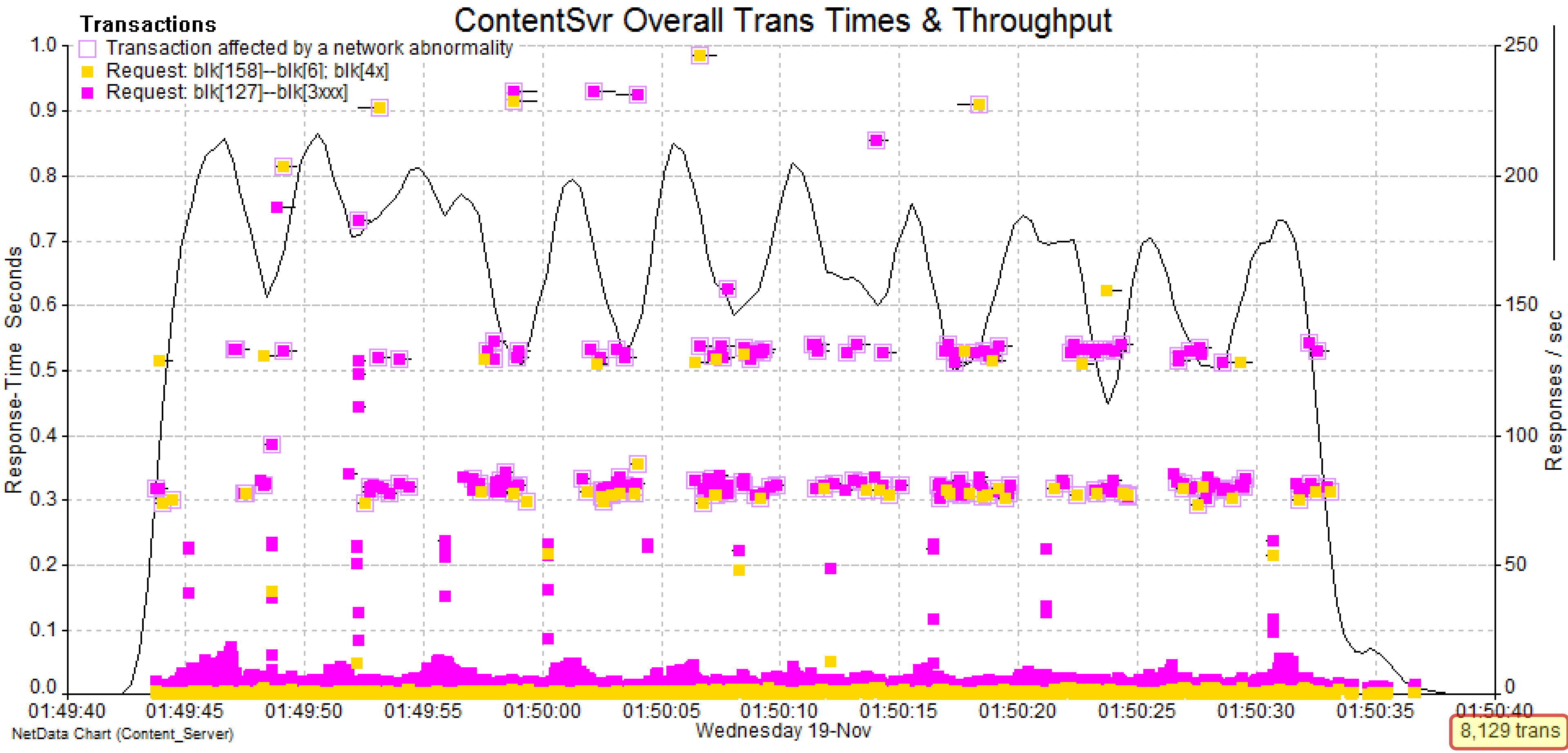


Test Run 1: First & Second Transactions

Now plotting only the first and second transaction types – and only during test run 1.

The mechanism that causes the "slow" responses to form horizontal bands at the ~300ms and ~500ms times clearly affects both of these transaction types in the same way.

If these are the two components of an SSL setup, then perhaps the SSL mechanism should be examined more closely.



Application “Sleep” Behaviour

The client and server (Web Server and Content Server) applications experience regularly occurring “gaps” – where they both seem to “sleep” (i.e., stop processing at the respective application layer for around 300 ms).

The “gaps” happen around every 4.5 seconds but vary between 4 to 6 seconds.

During these “gaps”, the TCP stacks at each end are still working, but the applications are not. Thus, the only packets observed during these gap periods are TCP type packets (Syn, Syn-Ack, Ack).

Any transactions that are in progress when a “gap” begins are carried over until the “gap” ends. Effectively adding 300 ms to all such transactions.

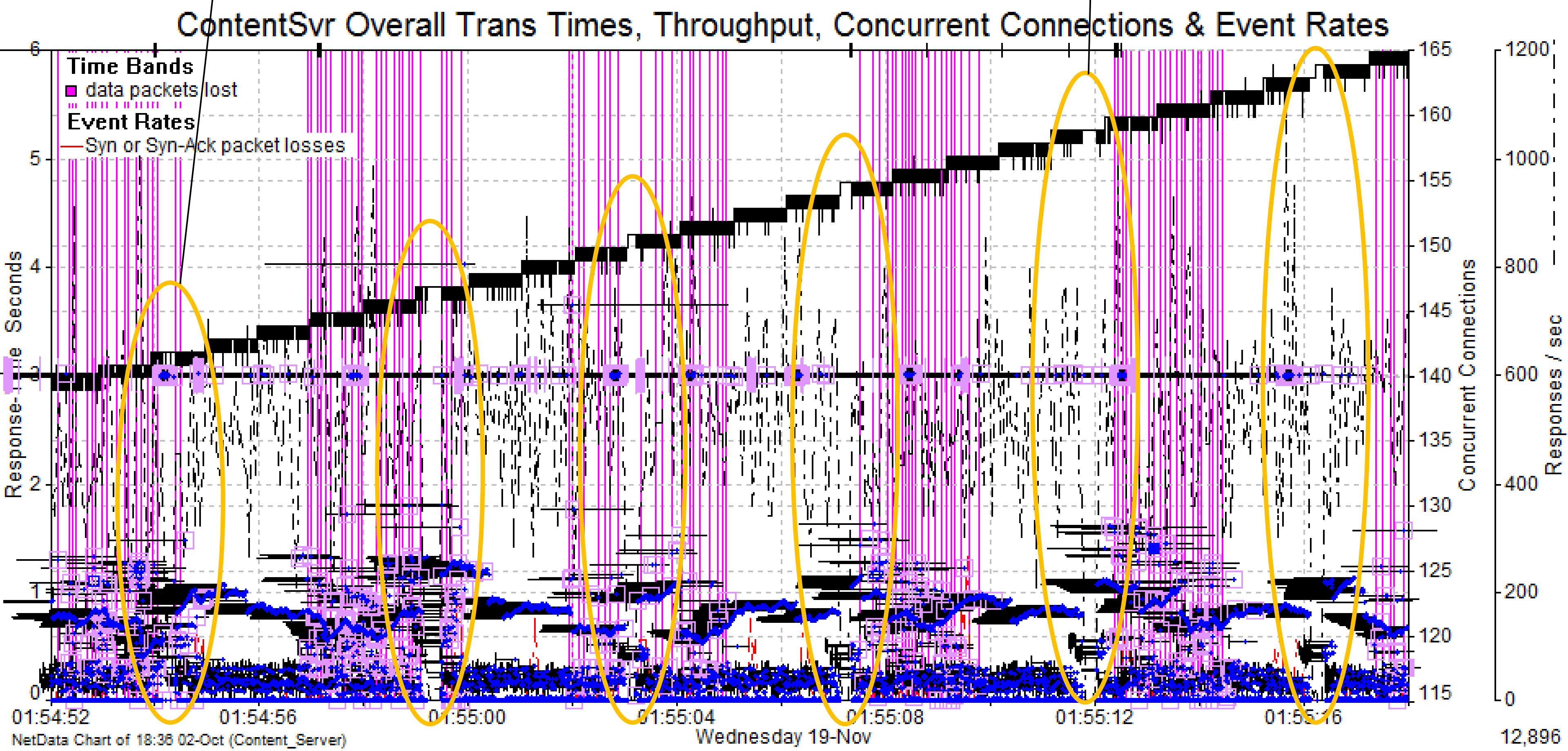
If it was just at one end, we might infer some sort of regular garbage collection routine in a server. However, I have no explanation for a mechanism that synchronises such behaviour across 2 separate servers. Something to do with virtual servers perhaps?

Application "Sleeping" Behaviour

In the orange ovals, we see very regular time periods where application activity stops (for both the Web Server and Content Server at the same time). No connections or transactions begin or end in these "gaps", all existing transactions span across the "gaps". The "gaps" occur at regular intervals, not correlated with the pink-background packet loss periods.
On the next slide we'll zoom-in to the 5th gap here – and see that the only packets within the "gaps" are at the TCP level (Syns, retransmissions or Acks) – not at the application level.

The "gaps" (inside the yellow ovals) occur at very regular intervals – across the whole 10 minute period.

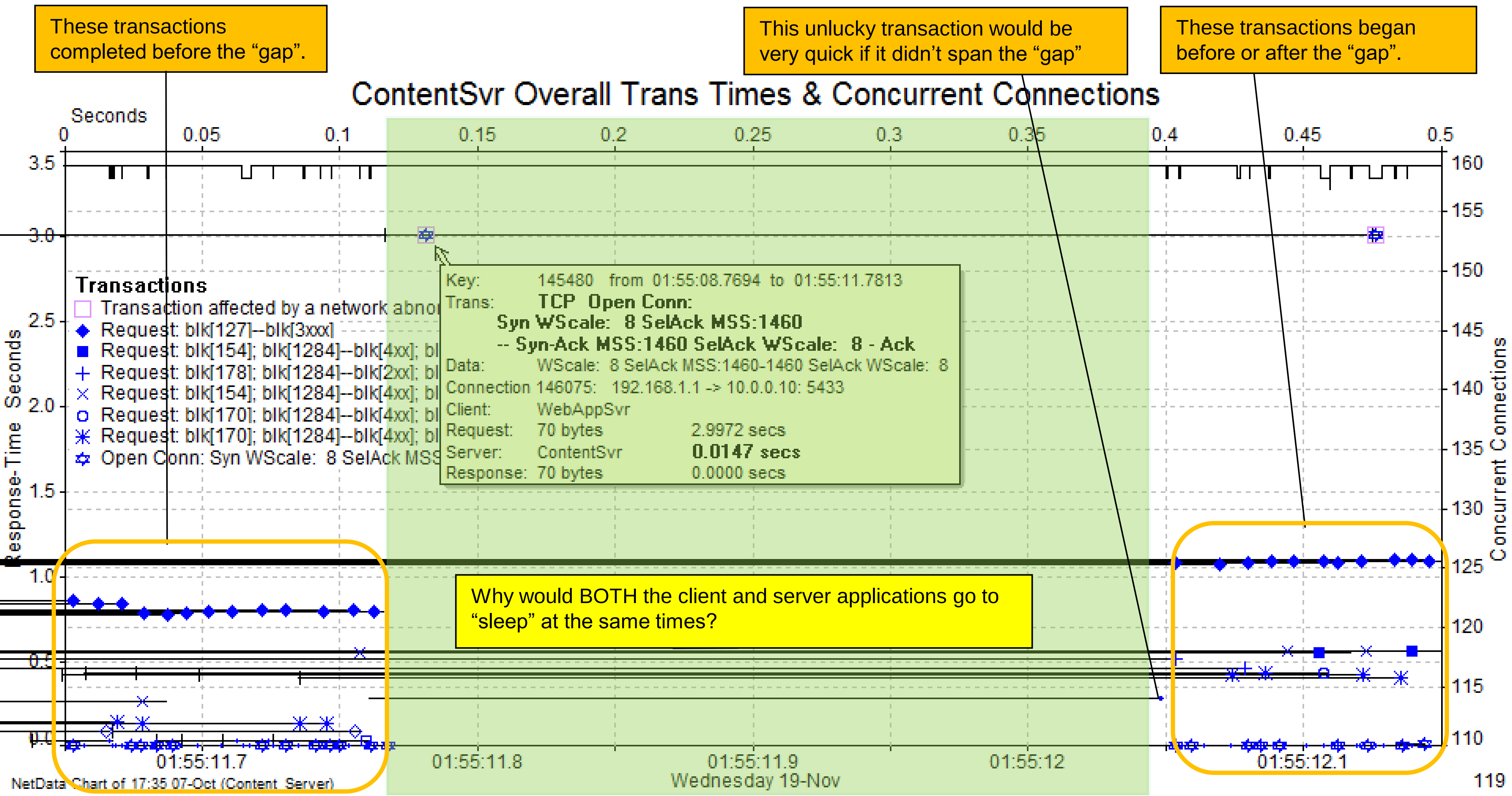
Why would BOTH the client and server applications go to "sleep" at the same times?



Application "Sleeping" Behaviour

Zoomed-in to a total half second period, we see a ~300 ms period (light green box) where no transactions begin or end, all in-progress transactions span across the "gap" (as indicated by the black horizontal lines).

The popped-up "transaction" inside the "gap" is actually a 3-sec TCP connection setup – where we had a retransmitted Syn from 3 seconds ago (this is TCP acting, not an application).



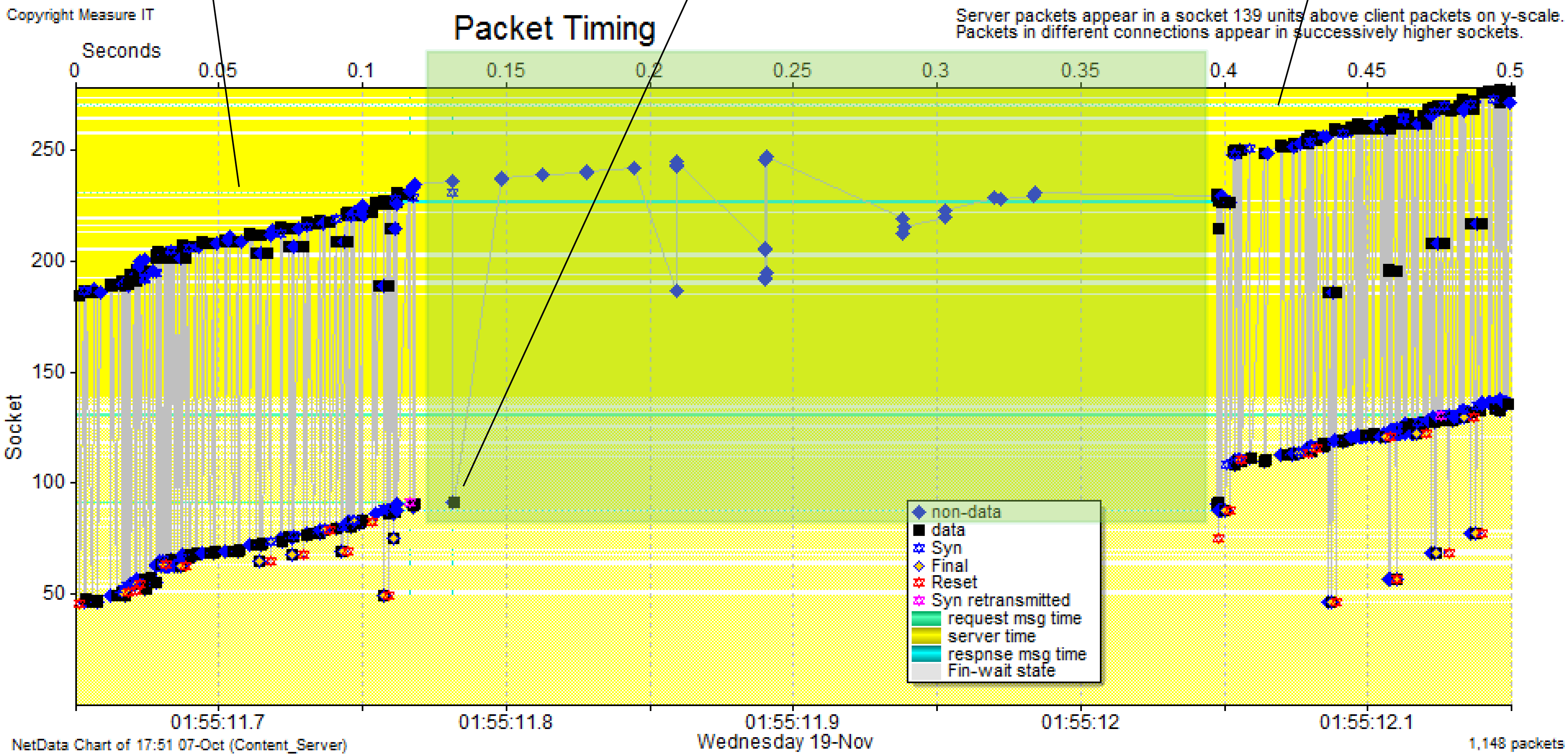
Application “Sleeping” Behaviour

A Packet Timing view (client packets along the bottom, server along the top – 138 connections) of the same time period shows no data packets in the ~300 ms “gap”. Only TCP level Syn-Ack and Ack packets (blue diamonds) occur inside the “gap”. These are server “delayed Acks” for client data packets that were transmitted before the “gap”.
That one data packet in the green area is a 127-byte first request that the client TCP stack must have had ready-to-go in response to the server’s Syn-Ack.

Packets are flowing constantly in both directions.

One 127-byte data request here – associated with that server Syn-Ack (see next slide).

Packet flow continues as normal.



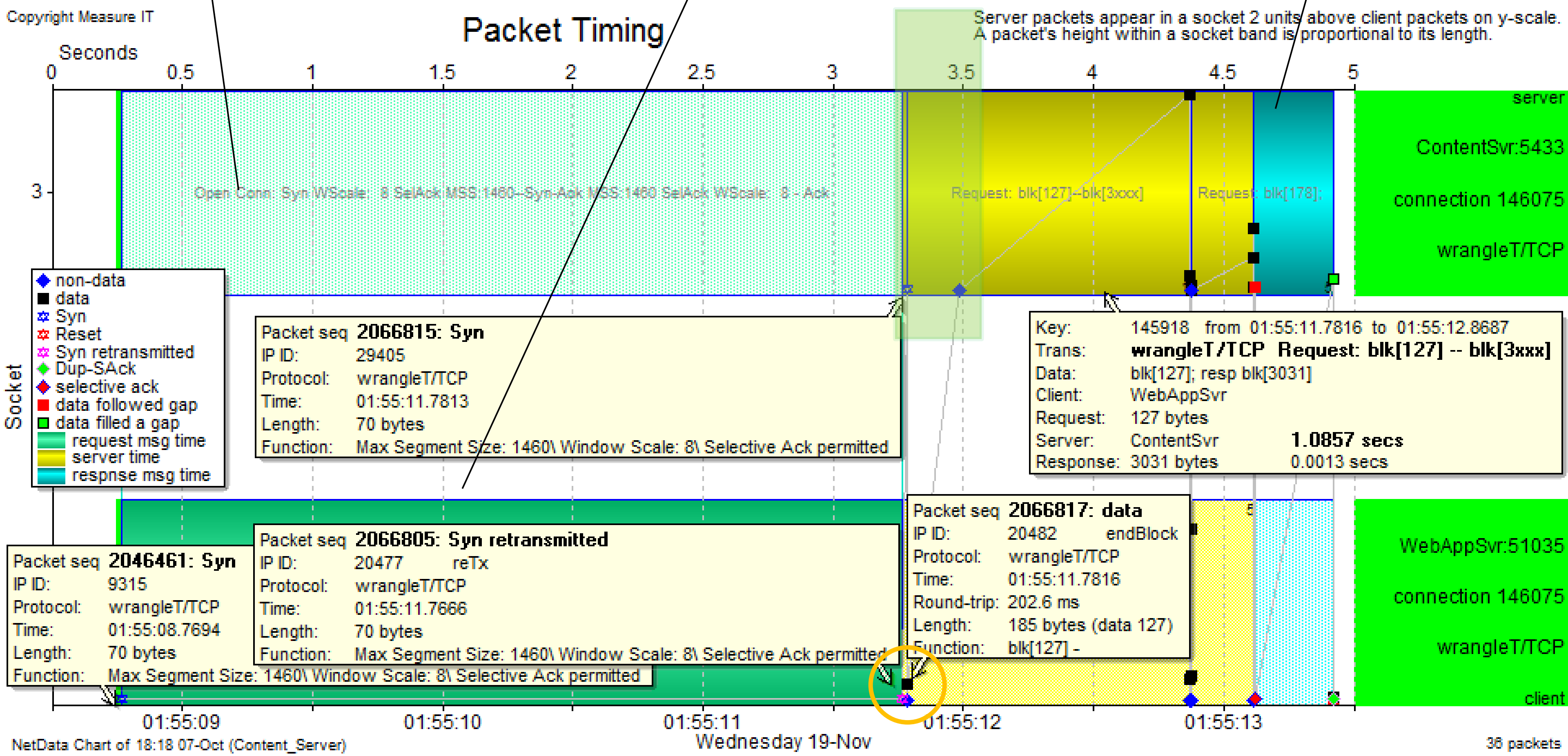
That Single Data Packet

This Packet Timing view of the connection containing that one data request packet (circled). We see that the client began the connection 3 seconds earlier – and the data is the first data packet in the connection.
It is the normal 127-byte request and we see a 200 ms delayed server Ack then the 3031-byte server response over a second later.

Packets are flowing constantly in both directions.

One 127-byte data request here – associated with that server Syn-Ack (see next slide).

Packet flow continues as normal.



Load Generator to Web Server

Next we'll look more deeply into the Load Generator to Web Server traffic. This is the "front-end" where the Web Server to Content Server is the "back-end".

There are some differences in behaviours of the Load Generator to the Web Server, particularly in the connection setups.

There are more concurrent connections here – and they are initiated all at once. However, there are far fewer total connection initiations across each test run because once initiated, each connection here triggers multiple sequential transactions (10 – 20).

Each front-end transaction here must correspond to multiple back-end connections and transactions.

Commentary

A typical connection request from the Load Generator to the Web Server involves:
(Note that these use port 443, so less need to hypothesise about SSL).

- TCP 3-way handshake.
- First data exchange: [61] byte request – [4513] byte response (SSL certificate?). All very fast.
- Second data exchange: [267]+[59] – [59] (SSL cypher?).
- Several (sometimes 20) large transactions: [362] – [286650].
- Termination by: Client 399-bytes data - Final – server Ack – server tens/hundreds KB data – client Reset.

All the connections are initiated at, or near, the very start of each test run.
There are therefore far fewer connections and SSL setups – and they are fast because they occur before the load ramps up.

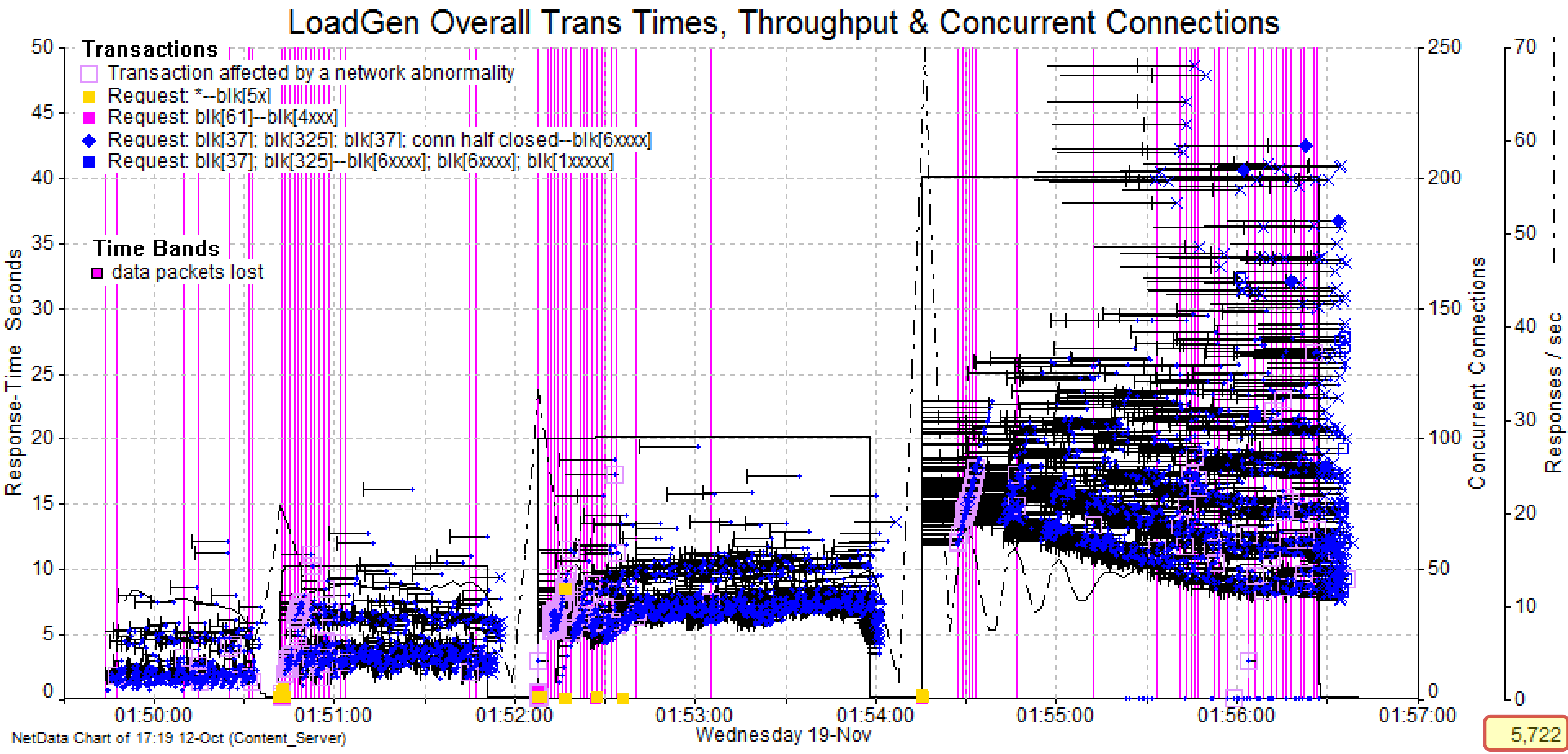
The responses to the Load Balancer's data requests do get progressively slower though, because the back-end requests to the Content Server get slower.

All the observed packet losses are between the tap(s) and the Load Generator. There is no regular pattern to these loses though (unlike the Content Server flows). The losses could be caused by a firewall, load balancer or other network device on the way to the Load Generator.

The transactions at the start of test run 4 take longer than transactions later in the test run because the Web Server begins with only 100 or so connections to the Content Server. The 200 transactions that initially arrive from the Load Generator are queued up. As more Content Server connections are initiated over time, more back-end transactions can be handled in parallel (making the front-end transactions faster due to spending less time in the queue).

All Load Generator Transactions

Here are all 4 test runs, showing only the Load Generator traffic. There are just 5,722 transactions here – and that includes connection setups. We see the connection lines rapidly rising to 25, 50, 100 & 200 respectively for each of the 4 test runs. The packet loss behaviour is more random (not every 5 secs). The transaction/sec figures (RHS scale) are greater at the beginning of each test run (presumably as all the connection setups fire off at once). There are far fewer pink and orange transactions (SSL certificate & cypher).
The transactions are significantly slower in test run 4 where the load is greatest.
Note: The packet losses are all “upstream” (i.e., between the tap and the Load Generator).



Load Generator - Statistics Table

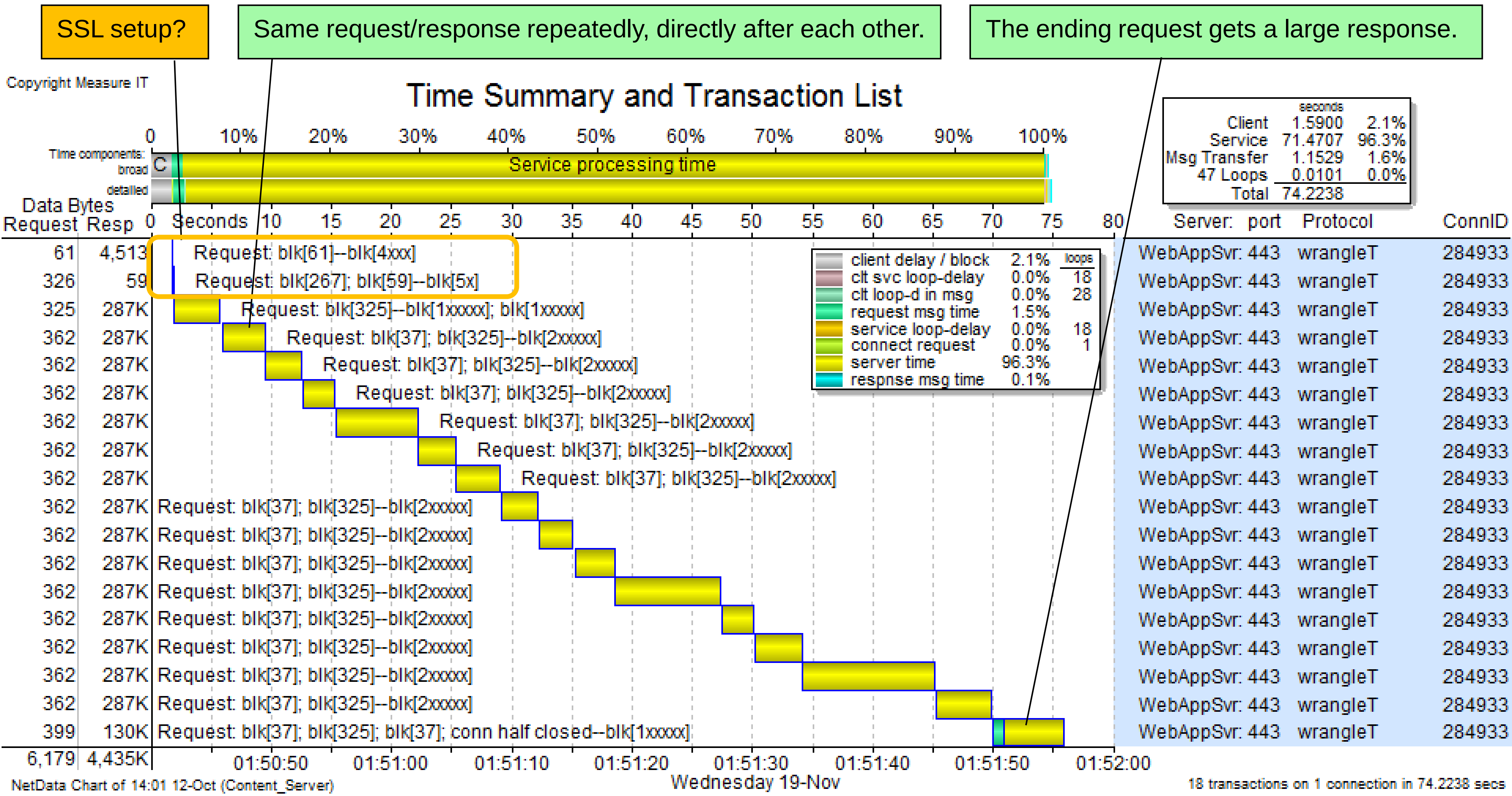
This snippet from the top of the Statistics table shows the transactions in order of response size. It also shows the colours (LHS) that will be used on the following slides. The pink transaction (61 byte request – 4513 byte response) is the first one in every successful connection. The orange one (158 – 51) is always the second transaction. Other transactions are closures or requests with 286 KB responses.

ID	Transaction Description	Plot	Clt Avg	Count	Req Bytes	SecsMin	Average	Maximum	Rsp Bytes	End Avg	End Max
53	Request: blk[326]--blk[5x]	Yes	0.008	14	326.0	0.0000	0.004	0.010	59.0	0.036	0.226
348	Request: blk[267]; blk[59]--blk[5x]	Yes	0.032	336	326.0	0.0000	0.001	0.019	59.0	0.385	8.505
346	Open Conn: Syn WScale: 4 SelAck MSS:1460--Syn-Ack MSS:1460 SelAck WScale: 8...	Yes		394	78.0		0.000	0.001	78.0	0.020	3.012
1007	Request: blk[93]--blk[1xx]	Yes		44	93.0		0.000	0.002	145.0	0.000	0.002
1103	Request: blk[6]; blk[378]; blk[37]; conn half closed--blk[2xxx]	Yes	0.006	14	421.0	0.8604	6.134	9.124	2896.0	15.479	32.351
1182	Request: blk[6]; blk[415]; conn half closed--blk[2xxx]	Yes	0.003	1	421.0	7.3724	7.372	7.372	2896.0	11.050	11.050
347	Request: blk[61]--blk[4xxx]	Yes		349	61.0	0.0000	0.003	0.022	4513.0	0.033	0.514
605	Request: blk[37]; blk[325]; blk[37]; conn half closed--blk[5xxx]	Yes	0.499	1	399.0	2.8648	2.865	2.865	5792.0	6.285	6.285
1179	Request: blk[37]; blk[325]; blk[37]; conn half closed--blk[7xxx]	Yes	0.231	2	399.0	5.9937	6.351	6.708	7240.0	11.608	11.828
1158	Request: blk[37]; blk[325]; blk[37]; conn half closed--blk[1xxxx]	Yes	0.141	4	399.0	0.7664	5.104	8.909	14118.0	14.781	24.589
1184	Request: blk[37]; blk[325]; blk[37]; conn half closed--blk[2xxxx]	Yes	0.096	3	399.0	7.5562	7.992	8.390	24616.0	13.083	16.606
1170	Request: blk[37]; blk[325]; blk[37]; conn half closed--blk[3xxxx]	Yes	0.087	2	399.0	4.8219	6.656	8.490	36924.0	17.577	26.889
1172	Request: blk[37]; blk[325]; blk[37]; conn half closed--blk[4xxxx]	Yes	0.100	5	399.0	6.0054	7.221	8.787	47494.4	14.518	24.204
1104	Request: blk[37]; blk[325]; blk[37]; conn half closed--blk[6xxxx]	Yes	0.126	6	399.0	2.0739	6.791	12.583	65401.3	29.935	42.582
1167	Request: blk[37]; blk[325]; blk[37]; conn half closed--blk[6xxxx]; blk[2xxx]	Yes	0.031	1	399.0	2.9550	2.955	2.955	68580.0	17.817	17.817
1166	Request: blk[37]; blk[325]; blk[37]; conn half closed--blk[6xxxx]; blk[5xxx]	Yes	0.030	1	399.0	2.4434	2.443	2.443	71476.0	12.141	12.141
874	Request: blk[37]; blk[325]; blk[37]; conn half closed--blk[6xxxx]; blk[7xxx]	Yes	0.123	1	399.0	1.8754	1.875	1.875	72924.0	11.945	11.945
604	Request: blk[37]; blk[325]; blk[37]; conn half closed--blk[7xxxx]	Yes	0.077	10	399.0	0.7417	3.592	8.387	75847.6	8.606	17.032
866	Request: blk[37]; blk[325]; blk[37]; conn half closed--blk[6xxxx]; blk[1xxxx]	Yes	0.096	8	399.0	0.5492	1.841	3.377	78897.0	7.203	10.974
593	Request: blk[37]; blk[325]; blk[37]; conn half closed--blk[8xxxx]	Yes	0.099	78	399.0	0.0251	2.111	6.006	85458.9	6.304	15.683
594	Request: blk[37]; blk[325]; blk[37]; conn half closed--blk[9xxxx]	Yes	0.098	20	399.0	0.9713	4.291	8.484	94699.2	9.525	26.291
592	Request: blk[37]; blk[325]; blk[37]; conn half closed--blk[1xxxxx]	Yes	0.077	208	399.0	0.0153	5.248	18.663	133469.1	17.729	48.662
1161	Request: blk[37]; blk[325]; blk[37]; conn half closed--blk[2xxxxx]	Yes	0.087	21	399.0	1.4869	6.847	9.639	231266.3	14.490	26.532
1033	Request: blk[6]; blk[378]--blk[1xxxxx]; blk[1xxxxx]	Yes	0.004	15	384.0	8.2080	13.487	26.583	285758.1	13.632	26.596
727	Request: blk[37]; blk[325]--blk[1xxxxx]; blk[6xxxx]; blk[6xxxx]; blk[2xxxx]	Yes	0.134	1	362.0	12.3218	12.322	12.322	286650.0	12.330	12.330
374	Request: blk[325]--blk[1xxxxx]; blk[7xxxx]; blk[7xxxx]	Yes	0.006	1	325.0	5.4460	5.446	5.446	286650.0	6.421	6.421
1107	Request: blk[6]; blk[378]--blk[1xxxxx]; blk[3xxxx]; blk[1xxxxx]	Yes	0.003	1	384.0	11.9245	11.925	11.925	286650.0	12.643	12.643
710	Request: blk[37]; blk[325]--blk[1xxxxx]; blk[6xxxx] (2); blk[2xxxx]	Yes	0.099	6	362.0	6.0469	7.859	9.999	286650.0	7.868	10.010
1086	Request: blk[6]; blk[378]--blk[6xxxx]; blk[2xxxxx]	Yes	0.006	1	384.0	8.9057	8.906	8.906	286650.0	11.595	11.595
560	Request: blk[37]; blk[325]--blk[1xxxxx]; blk[1xxxxx]; blk[2xxxx]	Yes	0.110	4	362.0	3.5467	5.273	7.038	286650.0	5.279	7.042
625	Request: blk[325]--blk[1xxxxx]; blk[4xxx]; blk[1xxxxx]	Yes	0.002	1	325.0	4.4888	4.489	4.489	286650.0	6.117	6.117
633	Request: blk[325]--blk[2xxxxx]; blk[2xxxx]	Yes	0.004	7	325.0	4.7470	7.852	14.908	286650.0	8.364	14.925

A Load Generator Connection

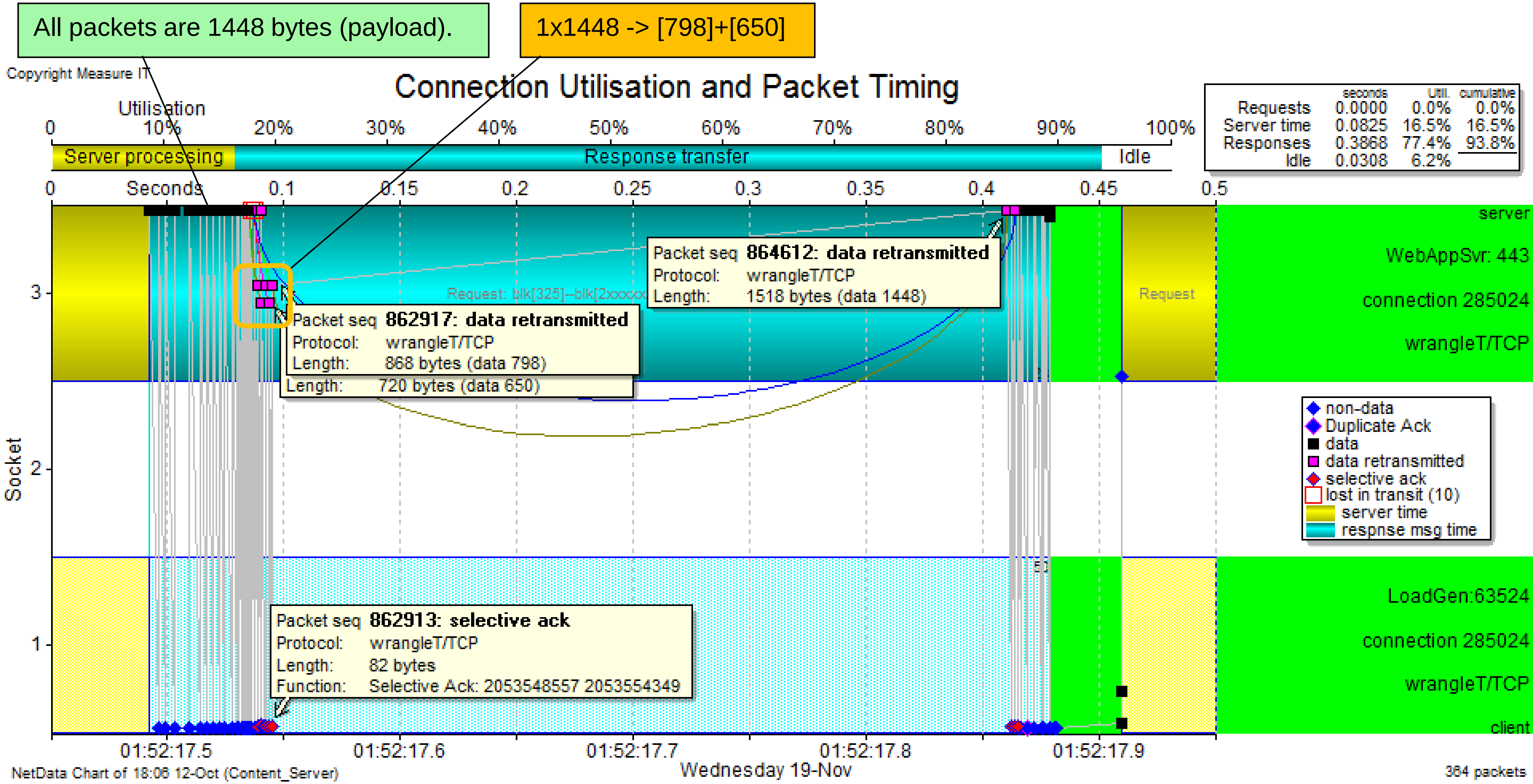
This "Waterfall" (or Gantt style) chart displays all the transactions within one of the connections. Notice the first & second very fast SSL setup transactions, then the repeated same large responses taking from 2.5 to 11 seconds. A new one begins immediately after the previous one completes.

The yellow indicates that this is all server "thinking" time (96.3% of the whole chart). Note the other timing breakdowns too.



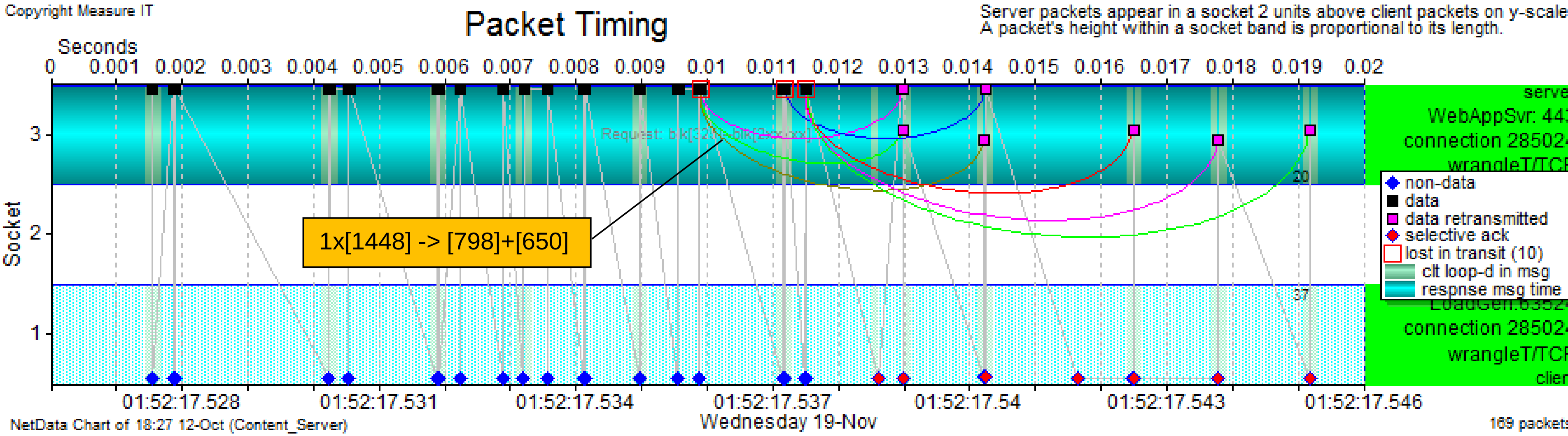
Example of Packet Losses

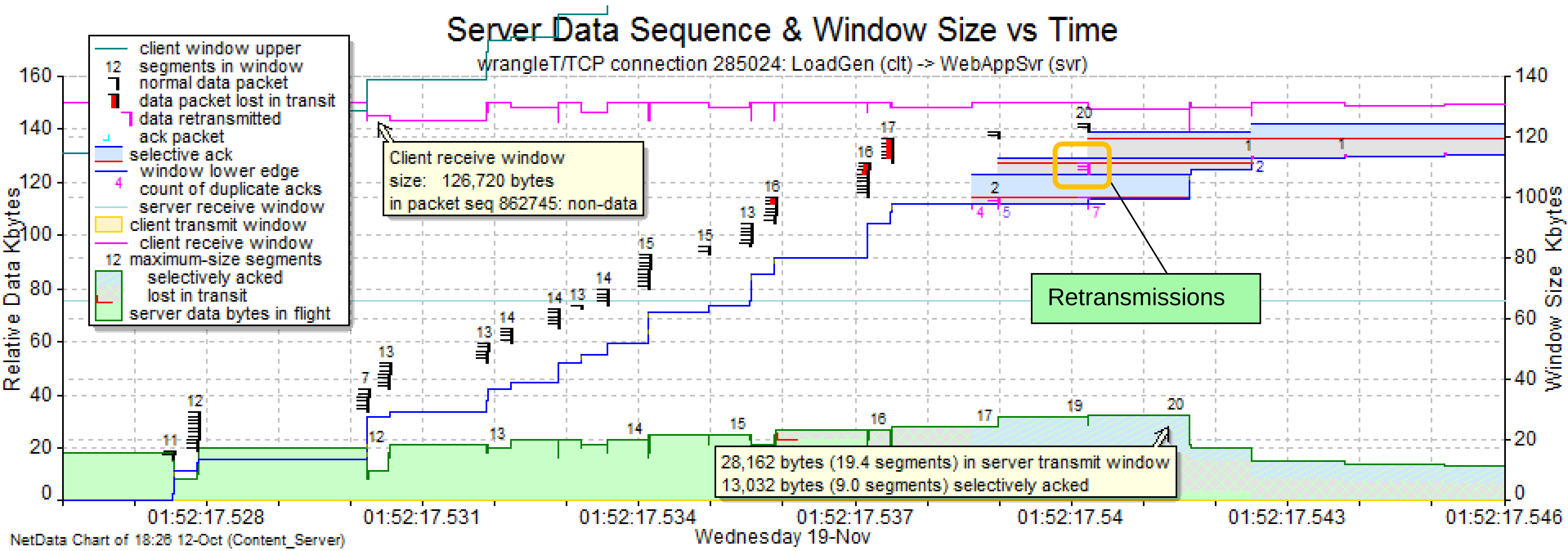
This Packet Timing chart is from the middle of a transaction containing losses and retransmissions. Top row is the Web Server here, bottom row is the Load Generator. The hollow red squares surround packets that were seen in this capture – but we know were not received by the client (because we also see SAcks). This means that the packets were lost between the tap and the Load Generator (through the Load Balancer?). The "loops" connect packets with their retransmissions.
Of interest is the small cluster of retransmitted packets, where the original [1448] byte single packet is re-sent in two packets of [798]+[650]. I've seen this behaviour in F5 load balancers – but here were are supposedly at the Web Server interface.



Example of Packet Losses (Zoomed)

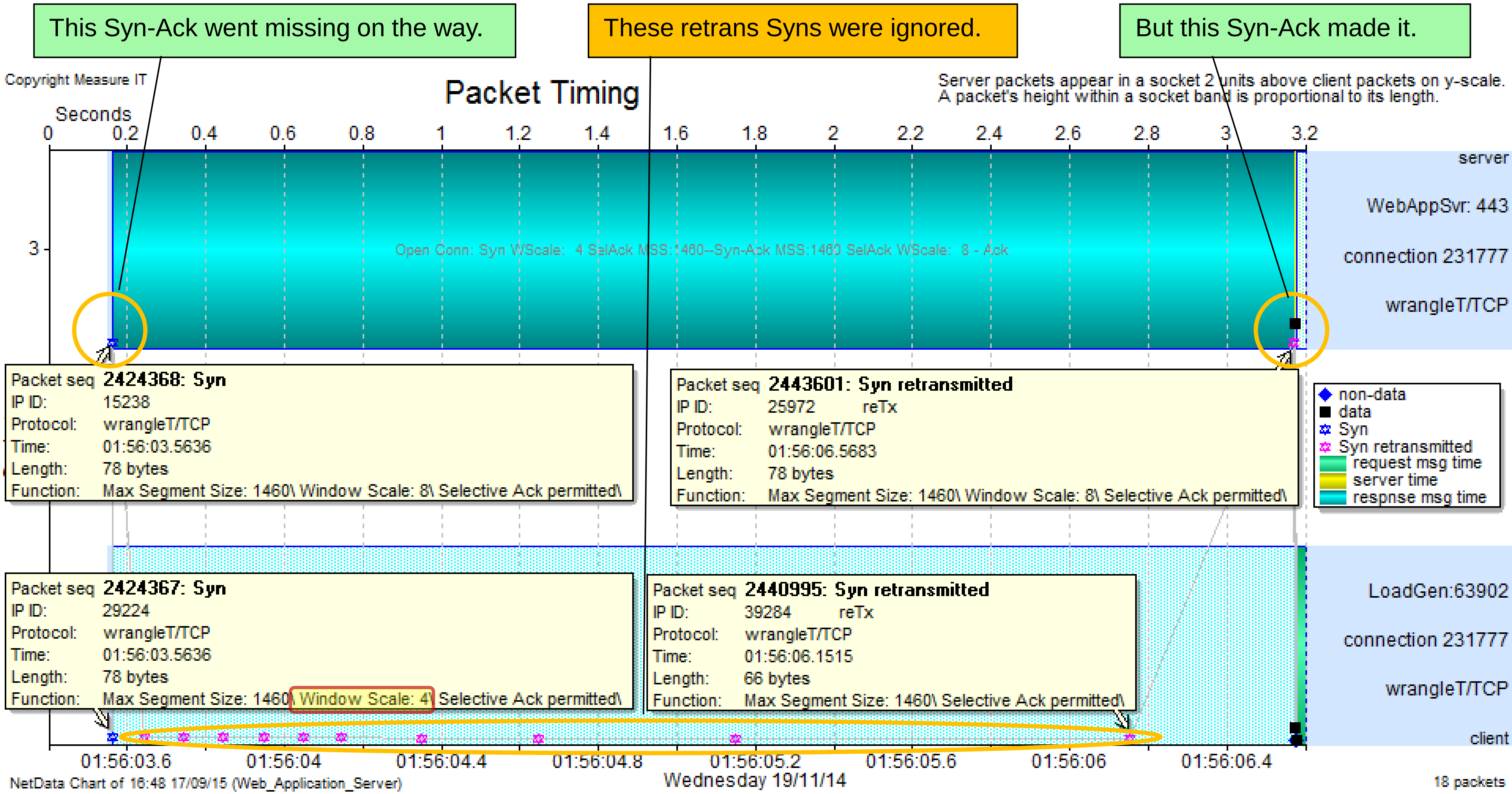
This is two views of the same packet flows – a small portion of the flow from the previous slide. Some retransmissions come very quickly in response to the Selective Acks.





Connection to Web Server from Load Generator

This is the one instance in the Load Generator traffic where an initial Syn was lost. We saw the Syn-Ack from the Web Server (circled) – but the Load Generator did not get it. The Load Generator retried its Syns many time very quickly (but the Web Server was now ignoring them because it already saw the first one). The second Syn-Ack from the Web Server (after a 300ms retransmission timeout) made it through.



Front-End & Back-End Together

Finally we'll look at a few slides showing both front-end and back-end activities all together.

- The first slide shows the whole time period.
- Second slide just test run four.
- Third slide just the start of test run four.

There are approximately 10 times more Web Server to Content Server transactions than Load Generator to Web Server.

Each front-end (red) transaction here must correspond to multiple back-end connections and transactions (blue).

If we had the original payloads, we would perhaps be able to match the reds up exactly with their corresponding blues.

The slow ramp up of blue connections is worth some further examination by the application owner.

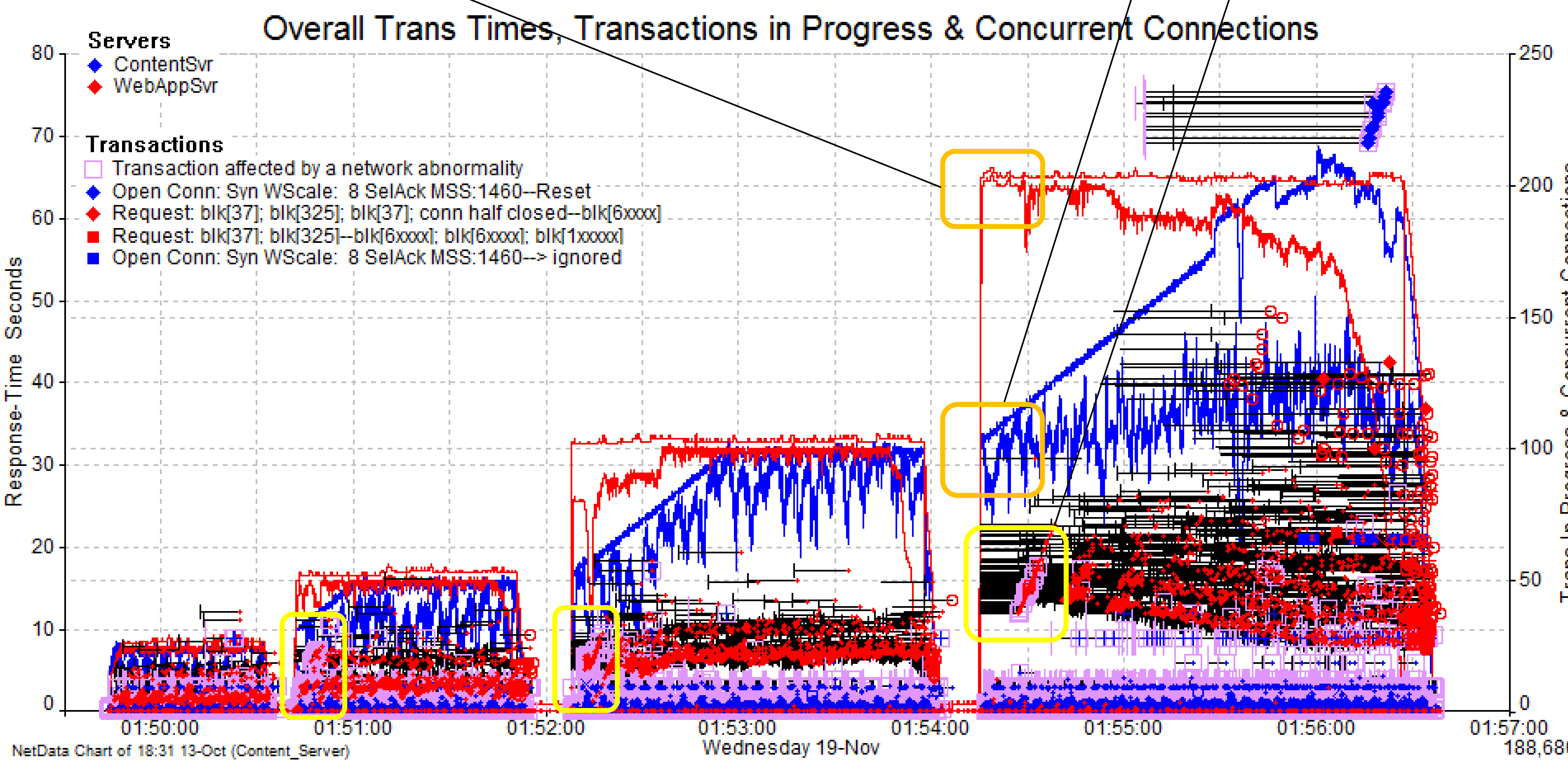
This Chart Tells the Whole Story!

The red transactions, connections & "TIP" lines represent the Load Generator (front-end) activity. The blue is the Content Server (back-end). The four test runs are clearly visible and we see the correlation of red & blue.

Load Generator connections ramp-up very quickly to ~200. There are ~200 "Transactions in Progress".

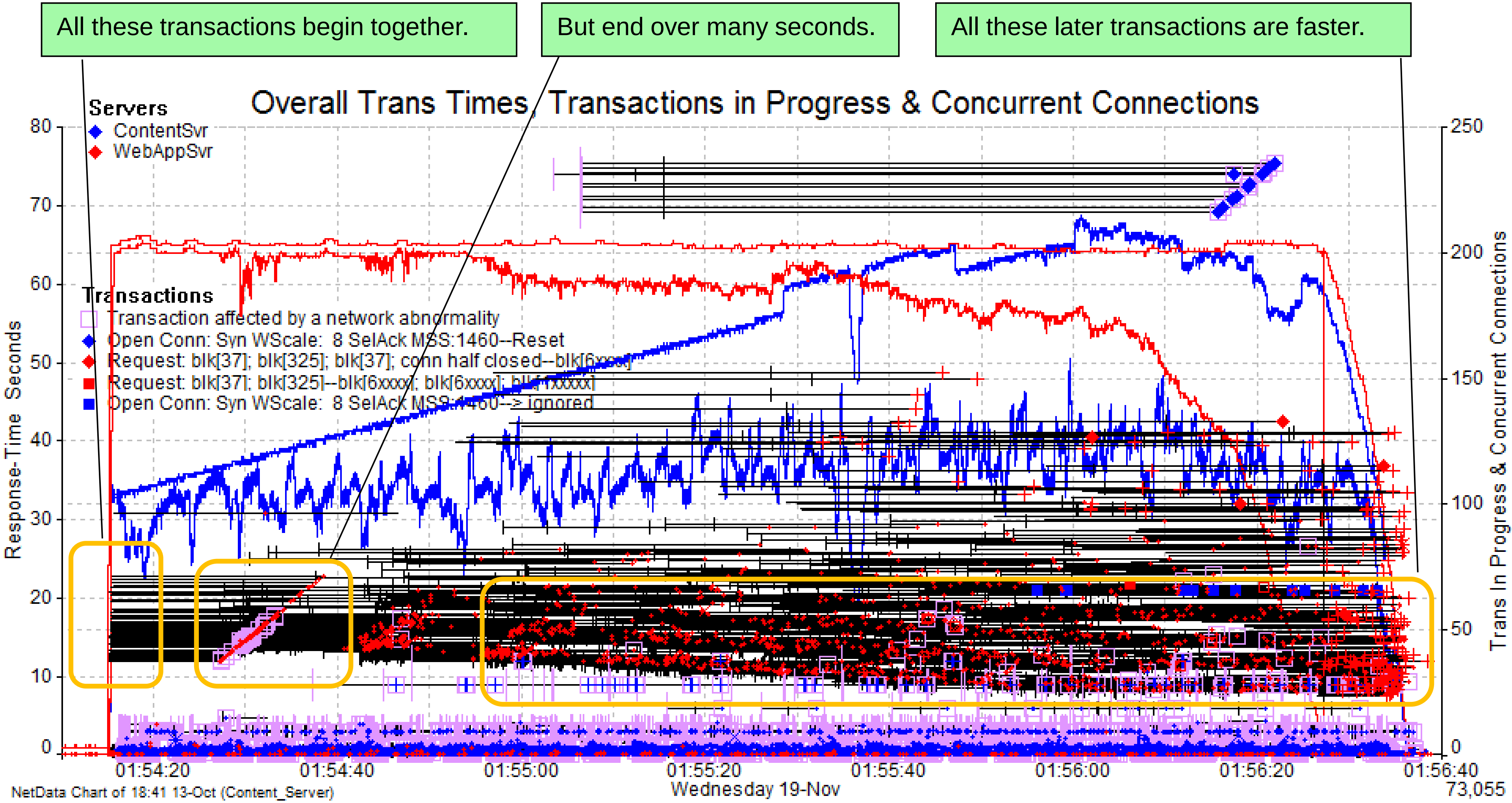
But only ~100 of both to the Content Server. The red transactions must be queued up (since only 100 can be worked on at once).

Therefore, the initial red transactions take longer then the subsequent ones.



Test Run Four - Whole Story!

Zoomed-in to test four. All the initial red requests arrive at once, but the responses take several seconds to get worked through. New requests take their place as they complete. Newer ones are (mostly) processed more quickly (because we have more parallel blue connections).



Start of Test Run Four

Zoomed-in even more - to the start of test four. All the initial red requests arrive at once, but the responses take several seconds to get worked through. New requests take their place as they complete. Each red transaction involves many blue ones – so even though the blues are faster, their times add up to make the reds.