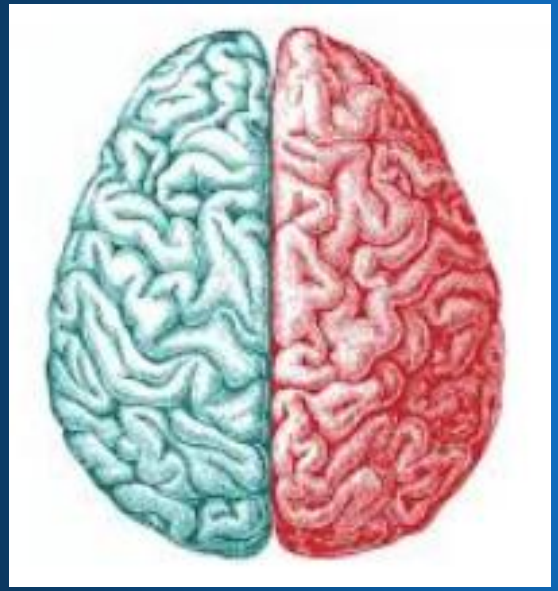
SharkFest '19 Europe

Session #14 - TCP Split Brain - Part II



Using Wireshark to Compare & Contrast behavior and TCP state of client vs. server

John Pittle

Riverbed Technologies
Performance Management Strategist
jpittle@riverbed.com

Premise: TCP Split Brain

- When troubleshooting TCP, you often have to consider both the sender's unique perspective and the receiver's unique perspective
- Both endpoints are independent, but at the same time, they do react to packets from the other end
- The joint behavior gets even more interesting when there's "high" latency in the path

Welcome Back!

- Continuation of Part I … after the break

Session Goals

- Compare and contrast TCP end point behavior
- Drill down into the “what is it doing?” and “why is it doing that?”
- Promote Wireshark Profiles Feature
- Share experience and ideas
- Expose you to visualizations that help reinforce the end point behavior we will be discussing

Part I Comparisons Recap

- 3-way handshake
- Latency
- Expert Info
- Fragment Overlaps / OOS / Retransmissions

Summary Slide from Part I

TCP Split Brain Comparison Summary (Sender vs. Receiver)

Item	Topic	Summary	Sender	Receiver
1	SYN Options	MSS, Scaling, TimeStamps, SACK	Negotiation & Adapt, MSS=1460, WS=8, SACK	Negotiation & Adapt, MSS=1360, WS=8, SACK
2	Latency	Can be different in each direction	Client iRTT == 121ms	Server iRTT == 129ms
3	TCP State	Different States at startup and shutdown; state change only occurs when packet sent / received	Closed, SYN-Sent, Established	Listen, SYN-Received, Established
4	Expert Info			
4.1	Duplicate ACK	Should be close to the same on both captures		
4.2	OOS		Not expected on sender; but retrans could be interpretted as OOS	Will be higher than sender
4.3	Retransmissions		Will be higher than receiver; could also be flagged as OOS	Might be flagged as OOS
4.4	Previous Segement Not Captured	Can be different	Not expected on sender unless there's a capture integrity issue	Expected on receiver when there's packet loss or OOS
4.5	SSL Errors	Could be side effect of reassembly in presence of OOS	Not expected	Likely to be reassembly issue
4.6	Fragment Overlap	Most likely caused by Segmentation Offoad	Most likely to show up on Sender's capture	Receiver could flag overlap using SACK field
5	Frame Sizes	Can be different	Effects of LSO / IP Fragmentation	Effects of LRO
6	Display Time Delta	Very unique to each endpoint	ACKs will usually apply to segments sent much earlier in time	Interpretation can seem confusing, especially when lots of packets are in flight
7	Bytes In-Flight	OOS and retransmissions may impact calculation	Should be pretty accurate	Generally not very interesting from receiver's capture
8	RTT2ACK		Can include receiver's Delayed ACK Timer; never less than 1 RTT	Generally Super Fast! When less than Delayed ACK Timer, could reflect degraded condition for TCP stack
9	Congestion	Requires advanced analytics		
10	Service Response Time		Client: May include latency + protocol delay + congestion	Server: should be most accurate

Part II Agenda

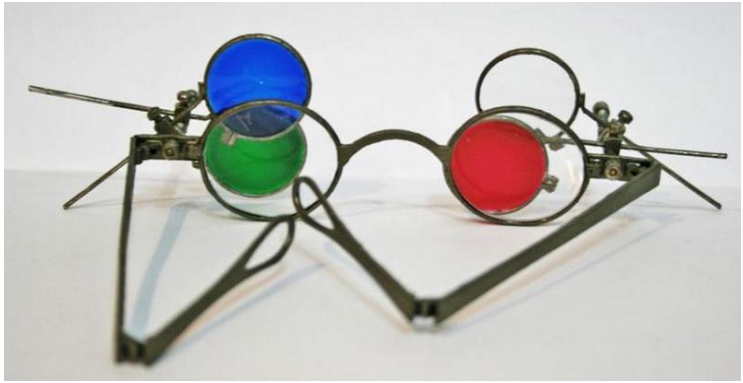
- Finish Fragmentation Topics
- Bytes in Flight Comparison
- Bonus – If Time Available

About me?

- Performance Engineering since 1980
- Protocol Analysis since 1991
- Professional Services with OPNET / Riverbed since 2005
- Love the mystery of a complicated performance issue
- Shaved off beard in 2003…

My Ask of You

- Engage
- Participate
- We have a lot of detailed material
- We will explore conflicting, contradictory, and possibly confusing details
- Ask Questions
- Question Answers



Application Scenario

- HTTPS Web Application
- Private key is not available
- Host based captures on web server and my laptop

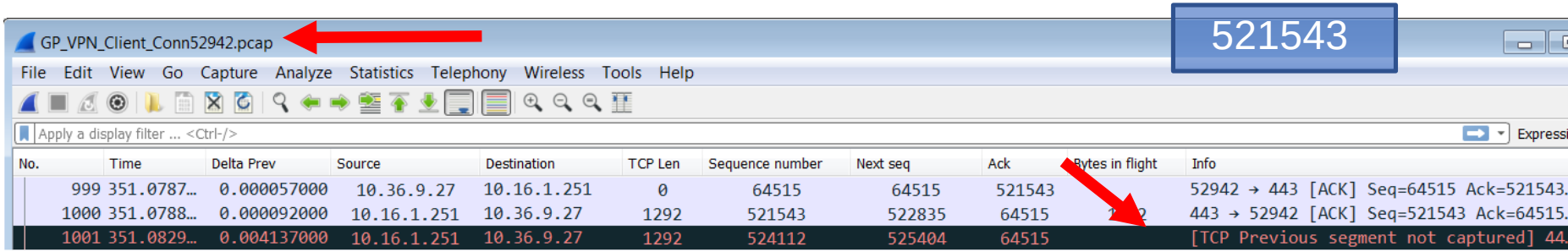
Symptoms to Analyze

- Downloading files take *forever*
- 16 seconds to download a 1.4MB file
- One TCP connection has been isolated as the connection of interest – TCP/52942-443


IT'S GO TIME

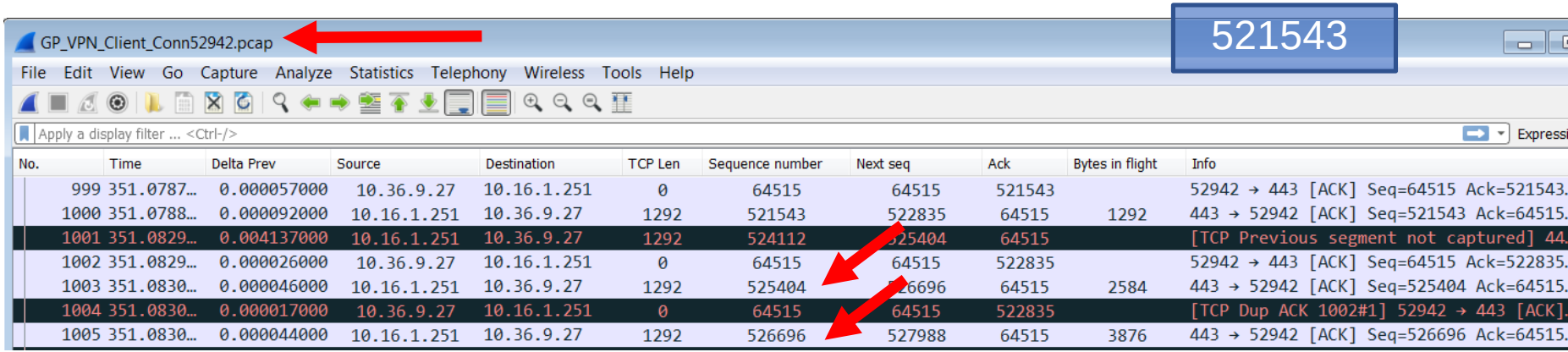
Where we left off in Part I

Dropped or OOS?



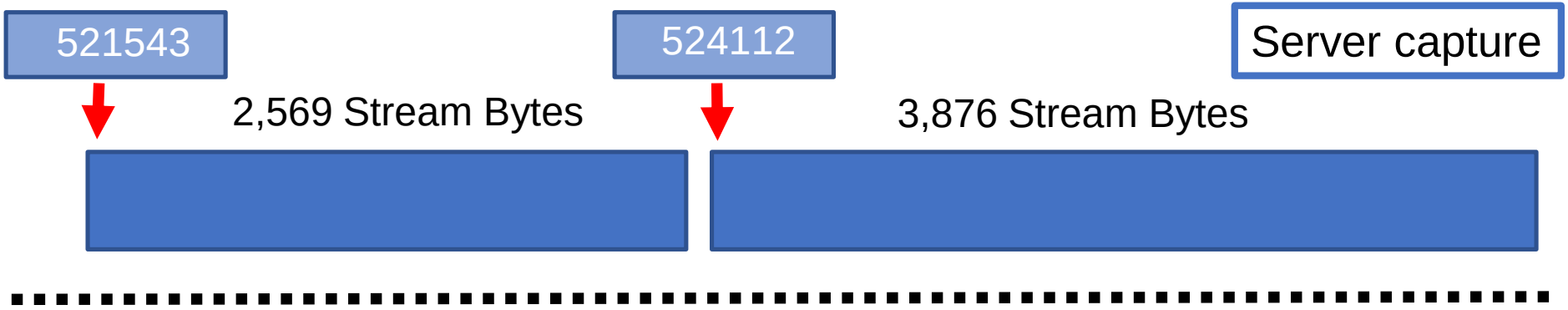
- If we ***freeze time*** right here, we can't be sure if it's just OOS or really a dropped packet
- We have to examine what comes next…

Dropped or OOS?

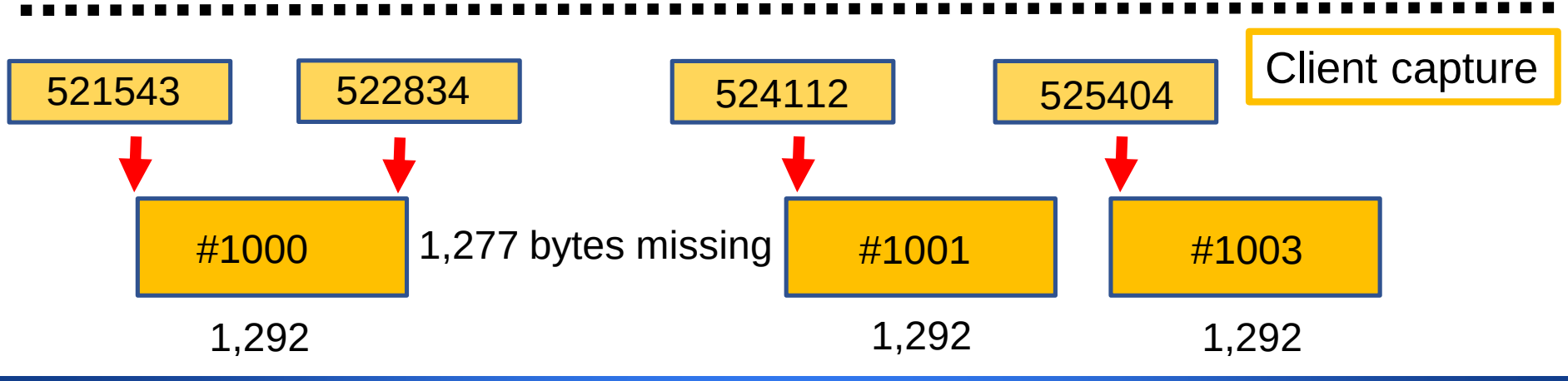


- Our missing segment is not showing up yet…

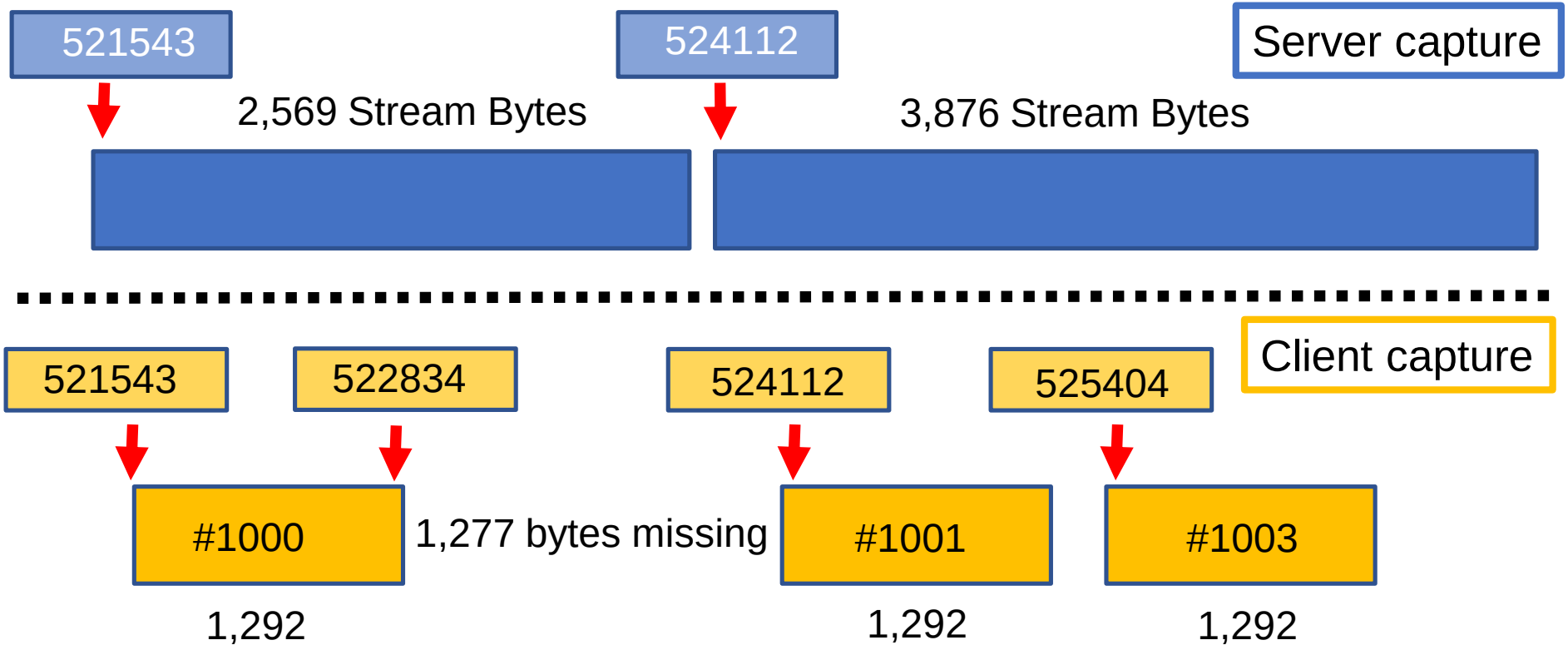
Anchor on SEQ==521543



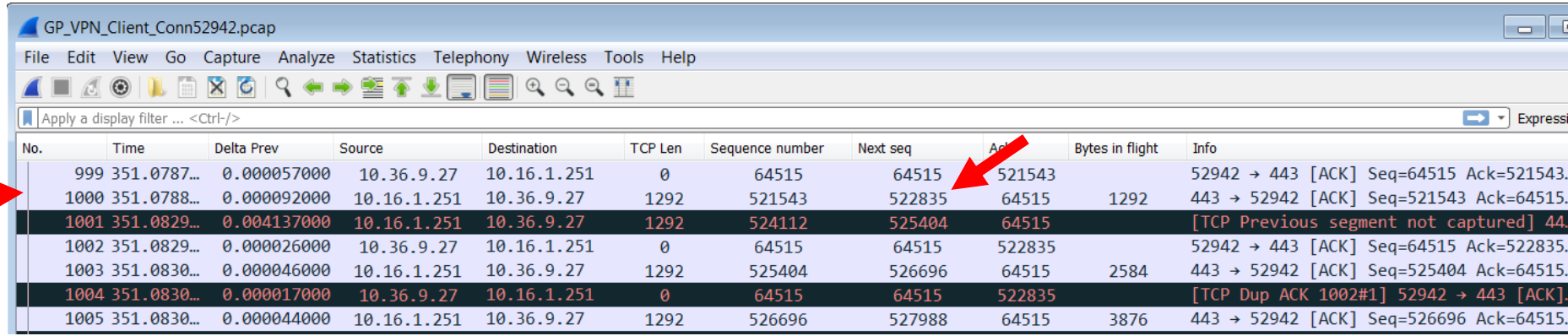
Anchor on SEQ==521543



Anchor on SEQ==521543



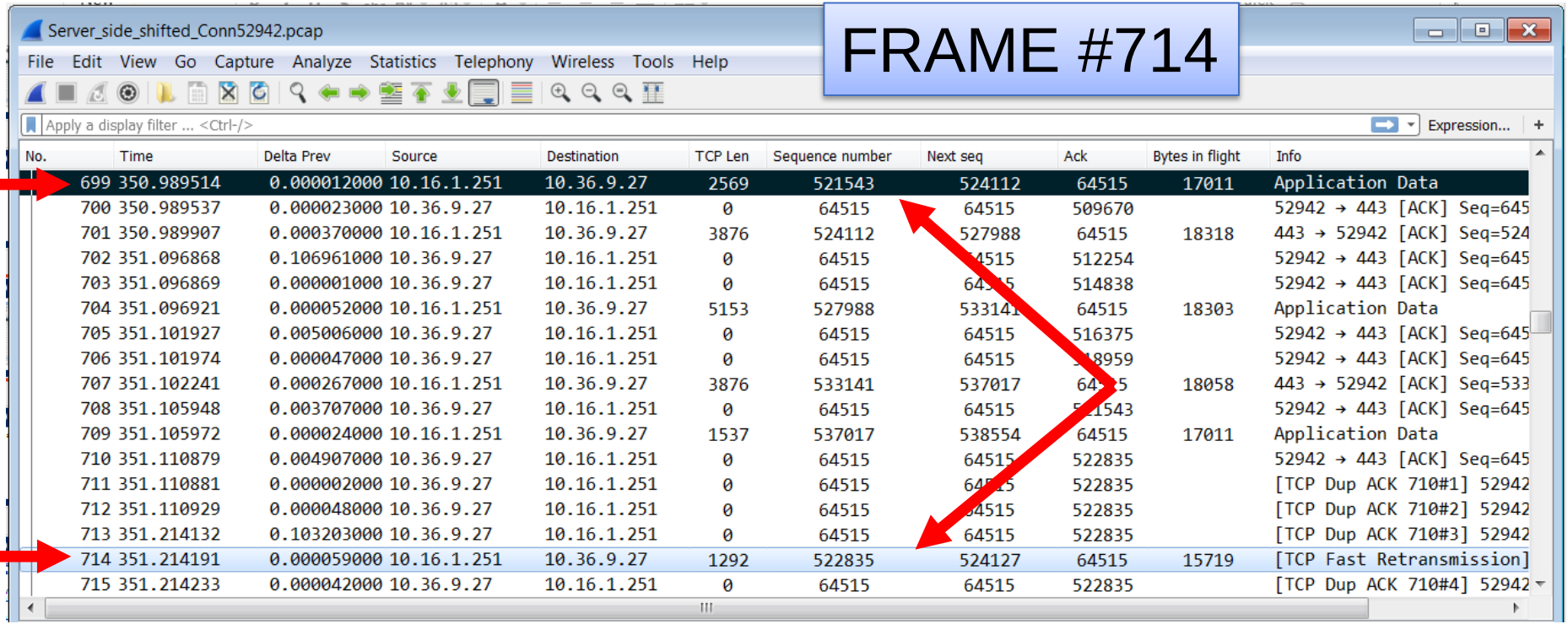
Other Clues



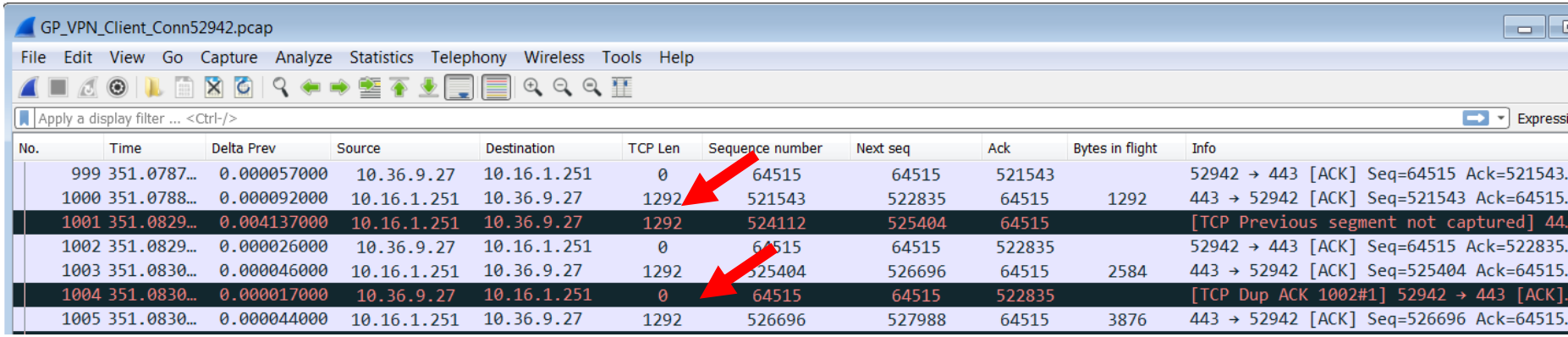
No.	Time	Delta Prev	Source	Destination	TCP Len	Sequence number	Next seq	Ack	Bytes in flight	Info
999	351.0787…	0.000057000	10.36.9.27	10.16.1.251	0	64515	64515	521543		52942 → 443 [ACK] Seq=64515 Ack=521543.
1000	351.0788…	0.000092000	10.16.1.251	10.36.9.27	1292	521543	522835	64515	1292	443 → 52942 [ACK] Seq=521543 Ack=64515.
1001	351.0829…	0.004137000	10.16.1.251	10.36.9.27	1292	524112	525404	64515		[TCP Previous segment not captured] 44.
1002	351.0829…	0.000026000	10.36.9.27	10.16.1.251	0	64515	64515	522835		52942 → 443 [ACK] Seq=64515 Ack=522835.
1003	351.0830…	0.000046000	10.16.1.251	10.36.9.27	1292	525404	526696	64515	2584	443 → 52942 [ACK] Seq=525404 Ack=64515.
1004	351.0830…	0.000017000	10.36.9.27	10.16.1.251	0	64515	64515	522835		[TCP Dup ACK 1002#1] 52942 → 443 [ACK].
1005	351.0830…	0.000044000	10.16.1.251	10.36.9.27	1292	526696	527988	64515	3876	443 → 52942 [ACK] Seq=526696 Ack=64515.

- Next seq after #1000 should have been 522835
- Wait! Isn't this the segment that was retransmitted by server? (yes)

(Slide From Part I)



Other Clues

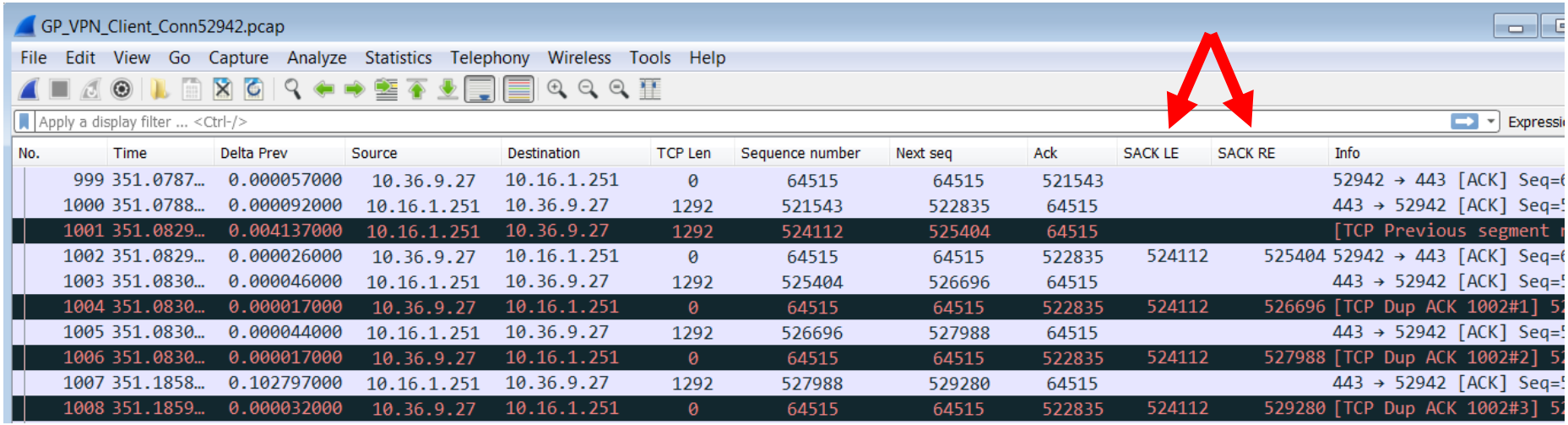


- Notice the change in ACK behavior
- Client ACKs every other packet then starts to ACK every packet
- Why is this?

Profile Power

- Let’s flip the view a little so we can quickly see SACK fields in the decode summary

Profile: SB-SACK



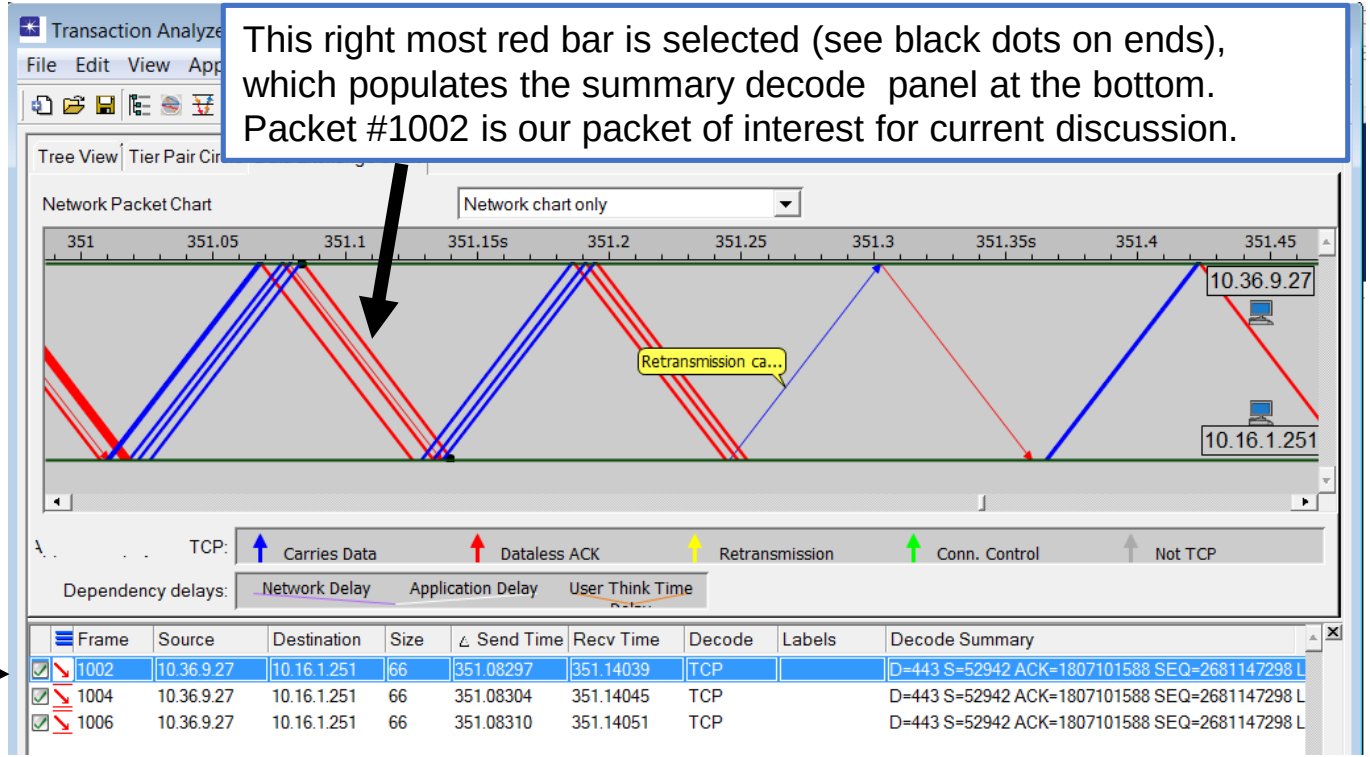
No.	Time	Delta Prev	Source	Destination	TCP Len	Sequence number	Next seq	Ack	SACK LE	SACK RE	Info
999	351.0787…	0.000057000	10.36.9.27	10.16.1.251	0	64515	64515	521543			52942 → 443 [ACK] Seq=6
1000	351.0788…	0.000092000	10.16.1.251	10.36.9.27	1292	521543	522835	64515			443 → 52942 [ACK] Seq=5
1001	351.0829…	0.004137000	10.16.1.251	10.36.9.27	1292	524112	525404	64515			[TCP Previous segment n
1002	351.0829…	0.000026000	10.36.9.27	10.16.1.251	0	64515	64515	522835	524112	525404	52942 → 443 [ACK] Seq=6
1003	351.0830…	0.000046000	10.16.1.251	10.36.9.27	1292	525404	526696	64515			443 → 52942 [ACK] Seq=5
1004	351.0830…	0.000017000	10.36.9.27	10.16.1.251	0	64515	64515	522835	524112	526696	[TCP Dup ACK 1002#1] 52
1005	351.0830…	0.000044000	10.16.1.251	10.36.9.27	1292	526696	527988	64515			443 → 52942 [ACK] Seq=5
1006	351.0830…	0.000017000	10.36.9.27	10.16.1.251	0	64515	64515	522835	524112	527988	[TCP Dup ACK 1002#2] 52
1007	351.1858…	0.102797000	10.16.1.251	10.36.9.27	1292	527988	529280	64515			443 → 52942 [ACK] Seq=5
1008	351.1859…	0.000032000	10.36.9.27	10.16.1.251	0	64515	64515	522835	524112	529280	[TCP Dup ACK 1002#3] 52

- Client “SACKs” the new segments, but continues to report - I’m missing 522835

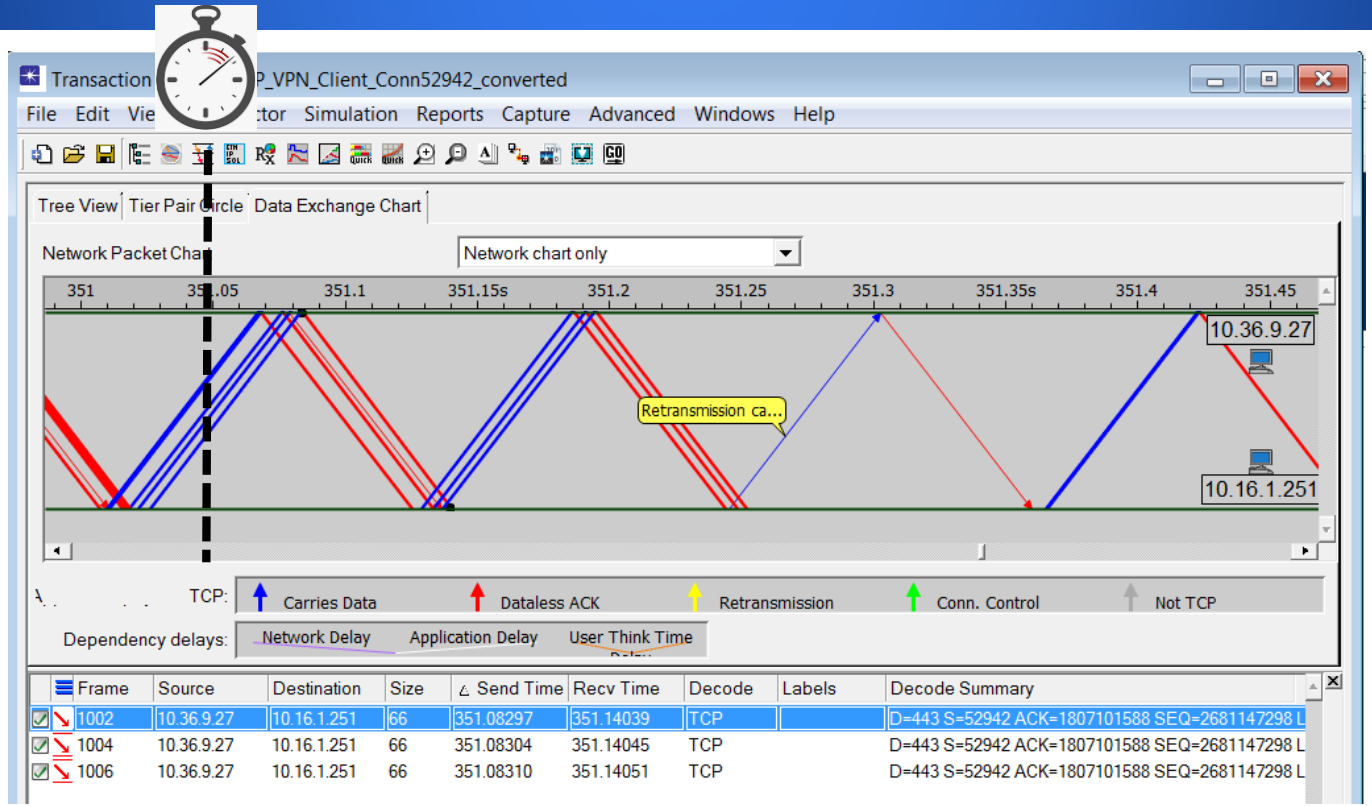
Discussion

- We can see Client is reporting a missing segment
- Yet, why does server continue to send segments other than the one requested?
- Visualization really helps with this…

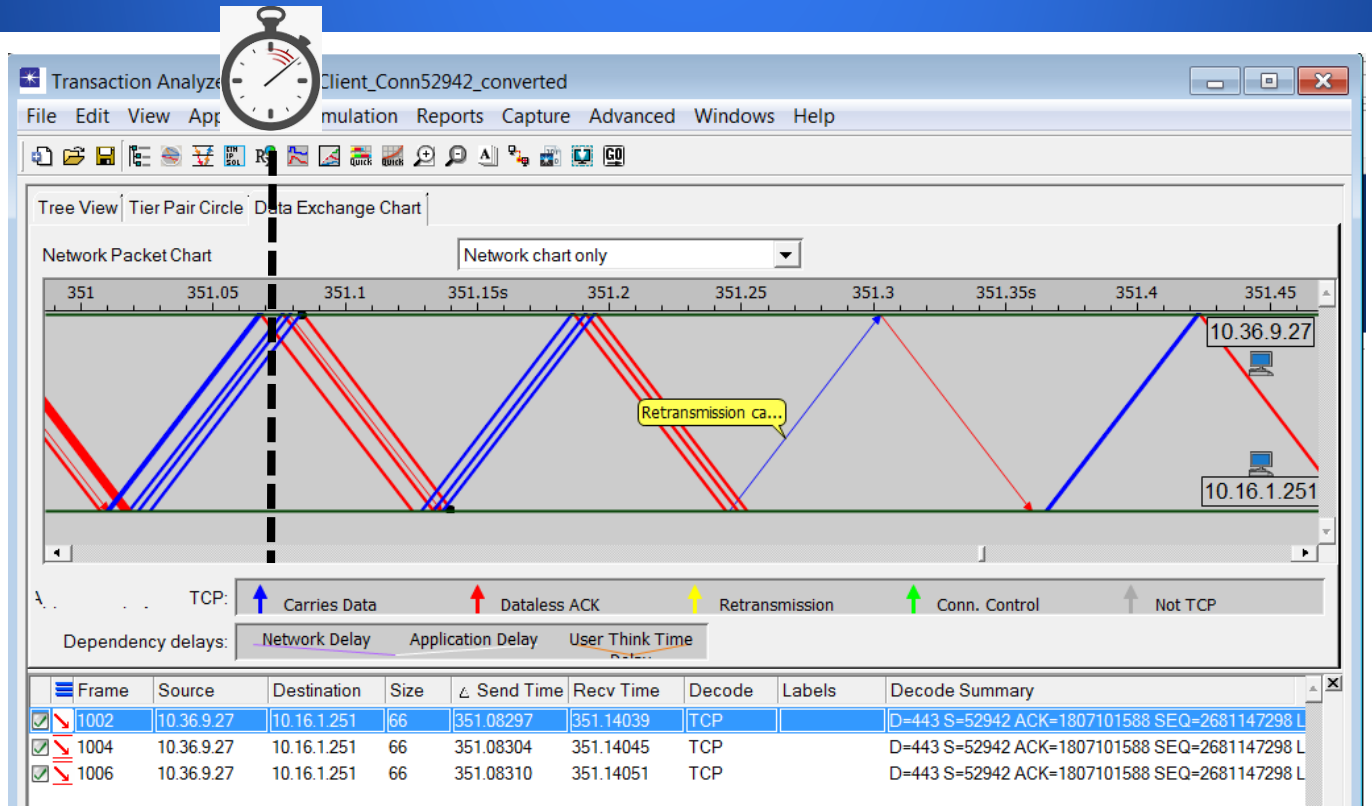
End Point State Review



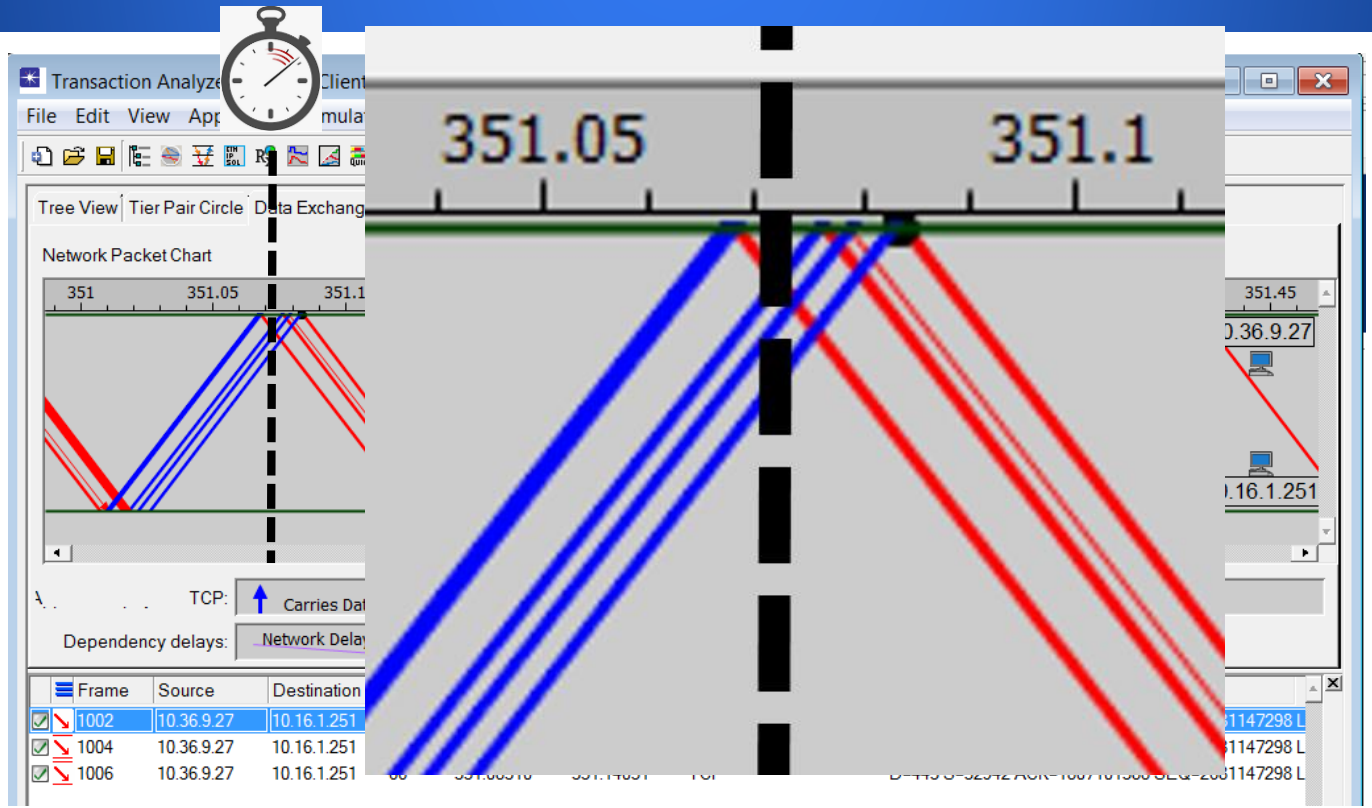
Discuss state of each end point



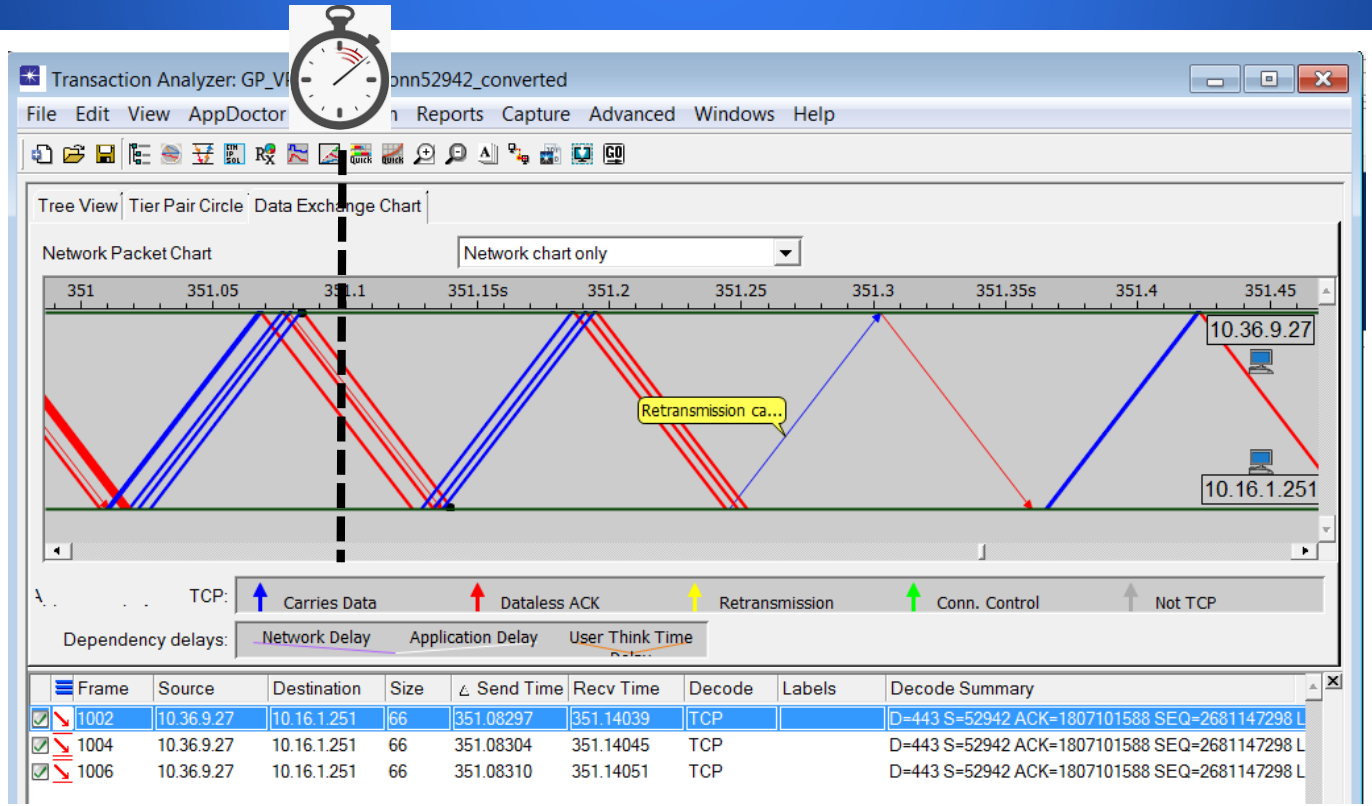
Discuss state of each end point



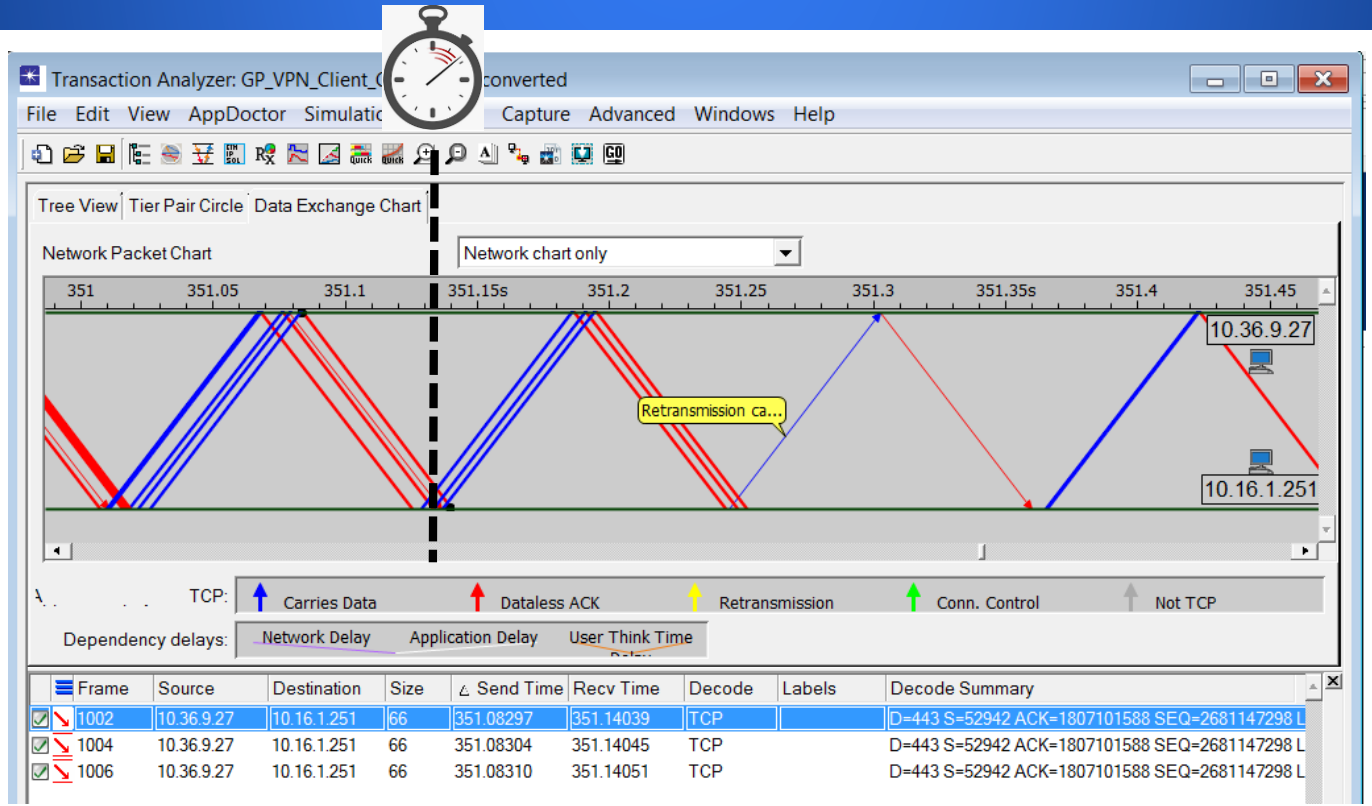
Zoom-in a little…



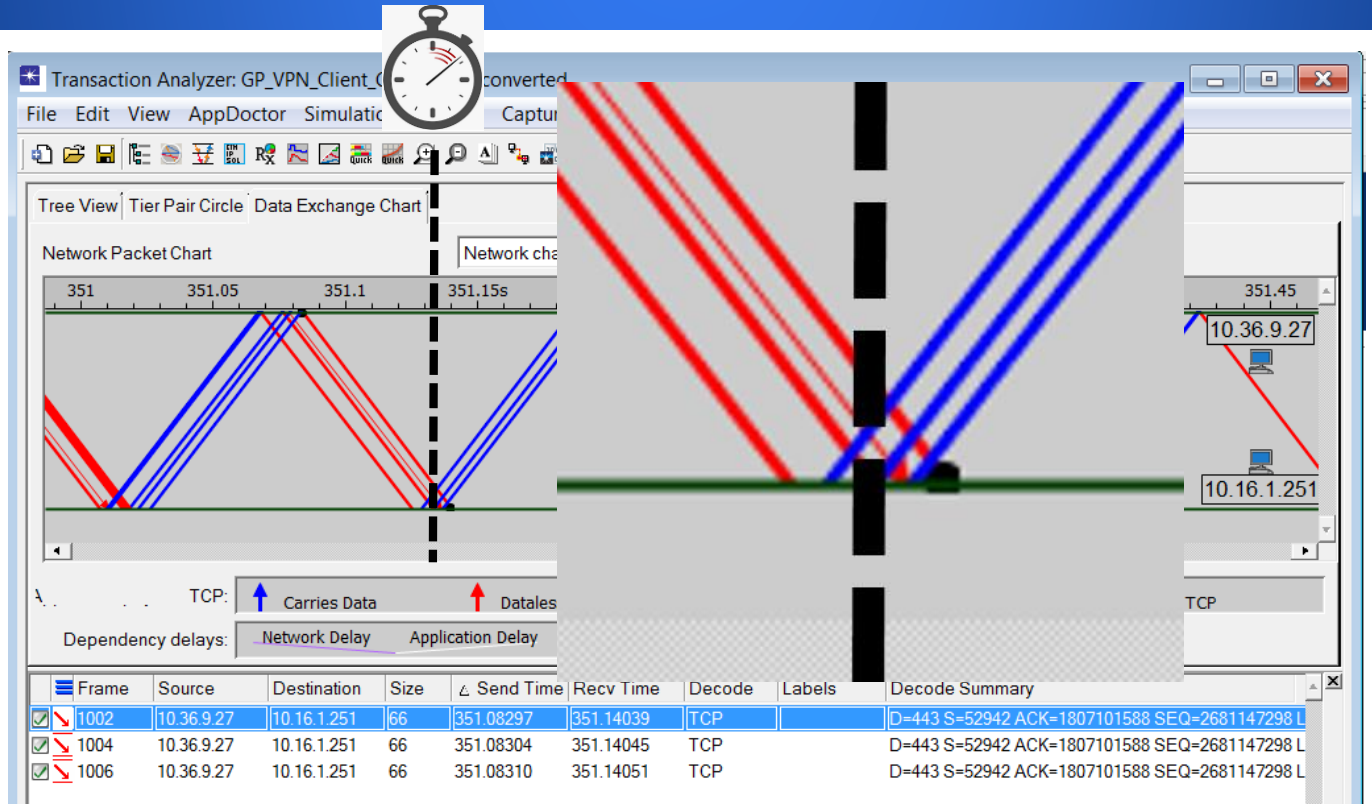
Discuss state of each end point



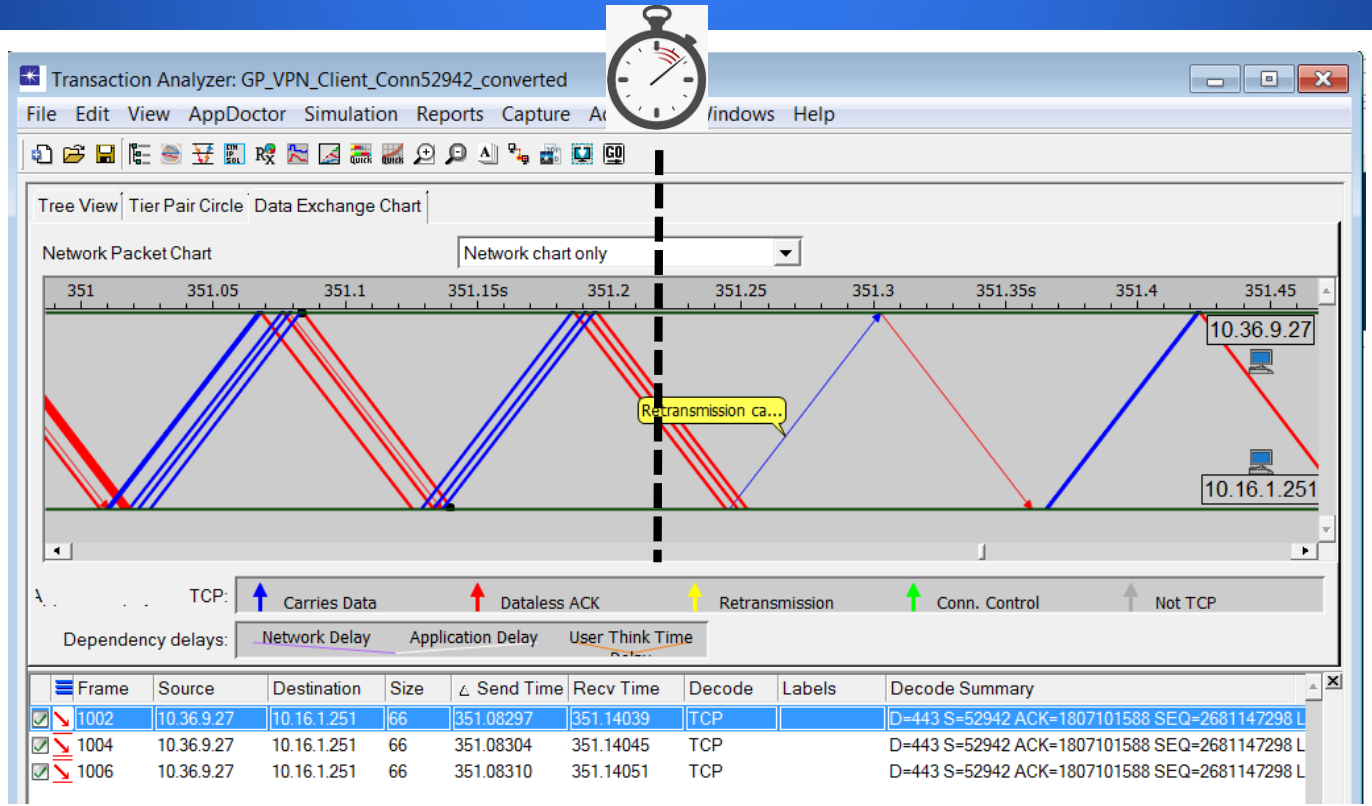
Discuss state of each end point



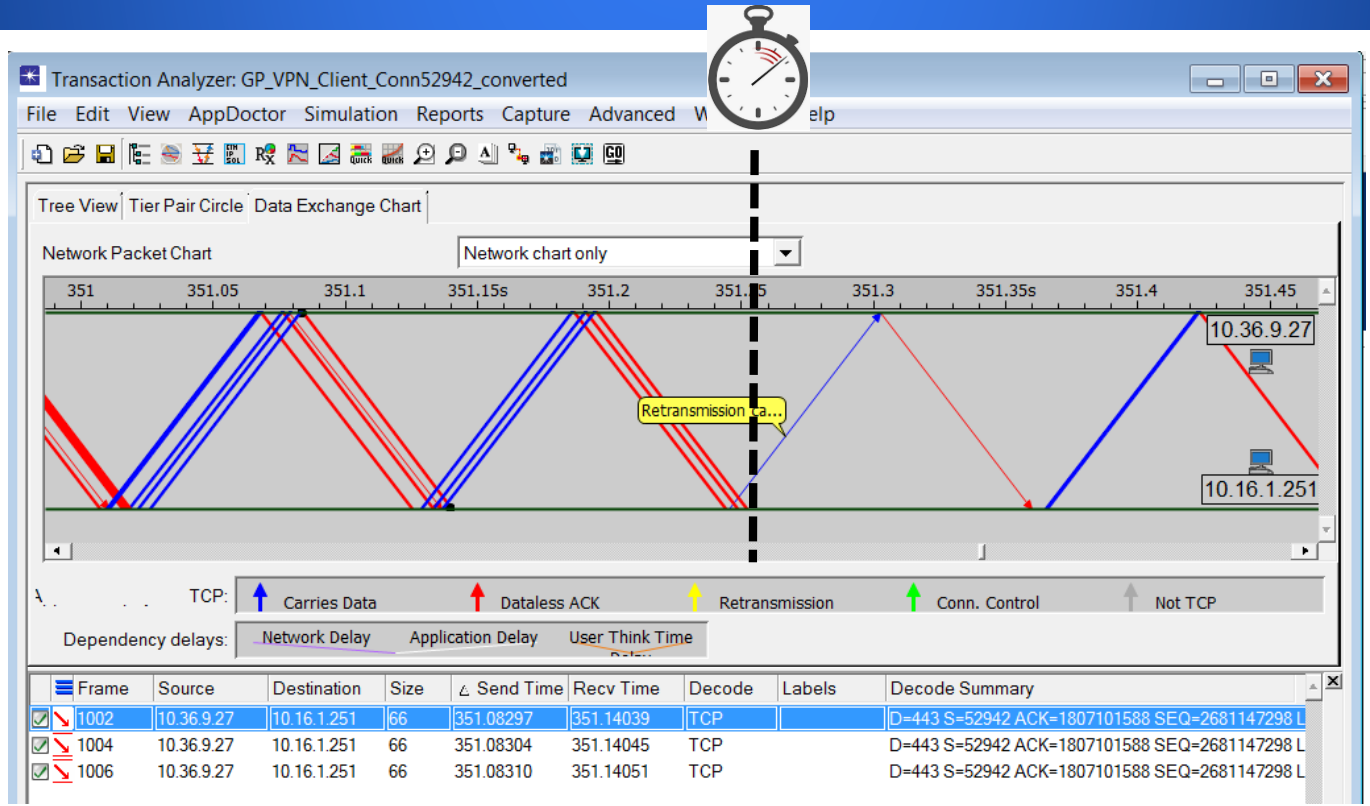
Zoom-in


Transaction Analyzer: GP_VPN_Client_
File Edit View AppDoctor Simulatic
Tree View
Tier Pair Circle
Data Exchange Chart
Network Packet Chart
351
351.05
351.1
351.15s
351.45
10.36.9.27
10.16.1.251
TCP:
Carries Data
Dependency delays:
Network Delay
Application Delay
Frame
Source
Destination
Size
Send Time
Recv Time
Decode
Labels
Decode Summary
1002 10.36.9.27 10.16.1.251 66 351.08297 351.14039 TCP D=443 S=52942 ACK=1807101588 SEQ=2681147298 L
1004 10.36.9.27 10.16.1.251 66 351.08304 351.14045 TCP D=443 S=52942 ACK=1807101588 SEQ=2681147298 L
1006 10.36.9.27 10.16.1.251 66 351.08310 351.14051 TCP D=443 S=52942 ACK=1807101588 SEQ=2681147298 L

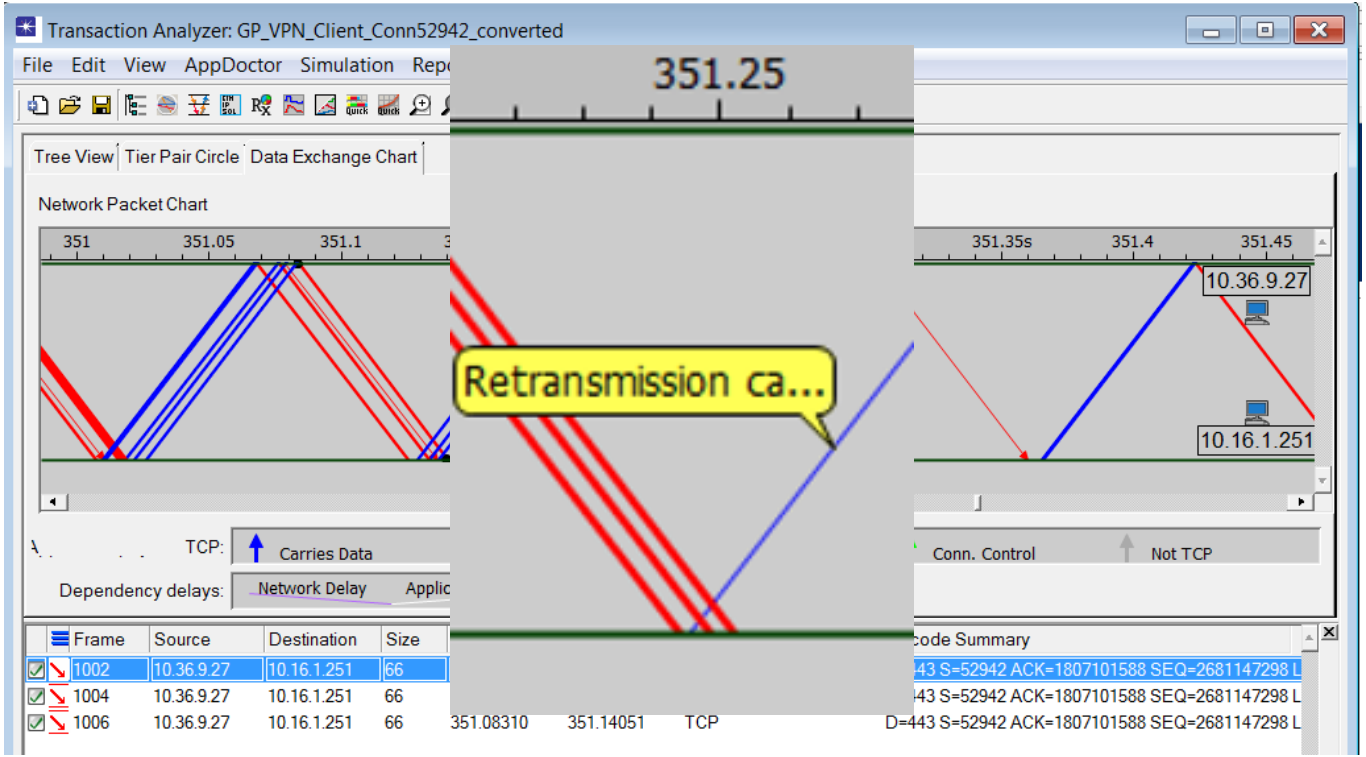
Discuss state of each end point



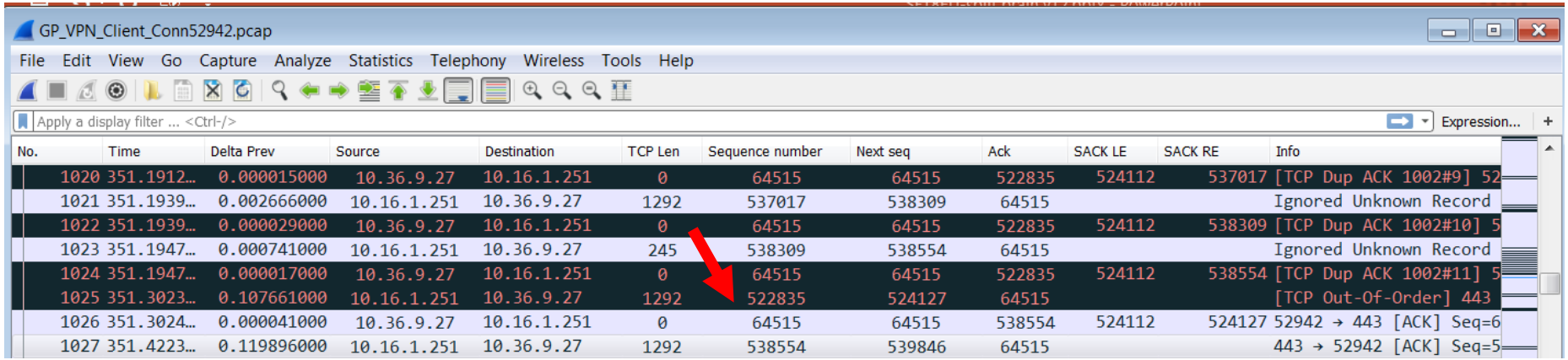
Discuss state of each end point



Discuss state of each end point

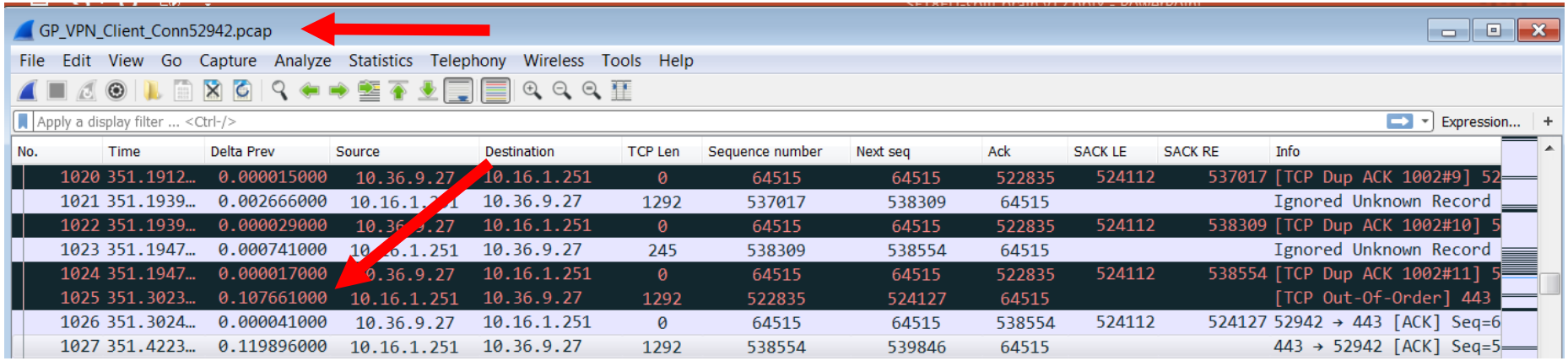


Missing segment finally arrives



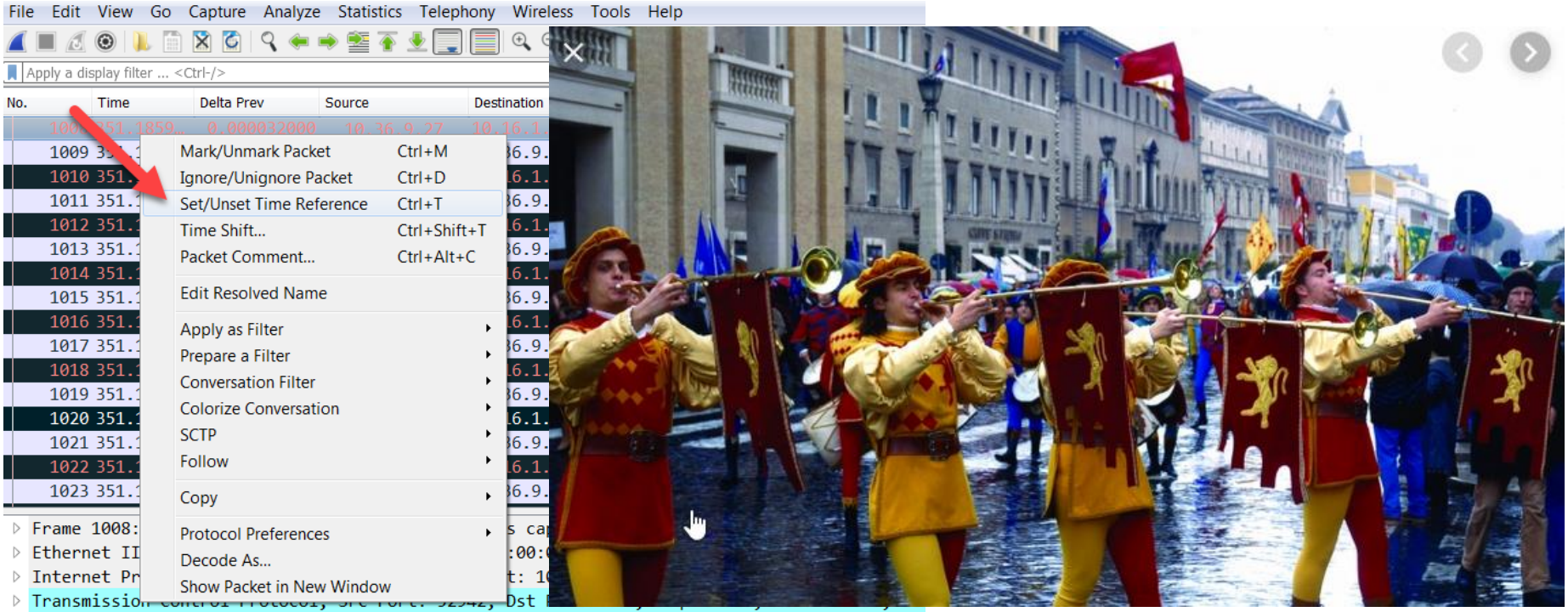
- Let's examine a few more details…

107ms Time Delta

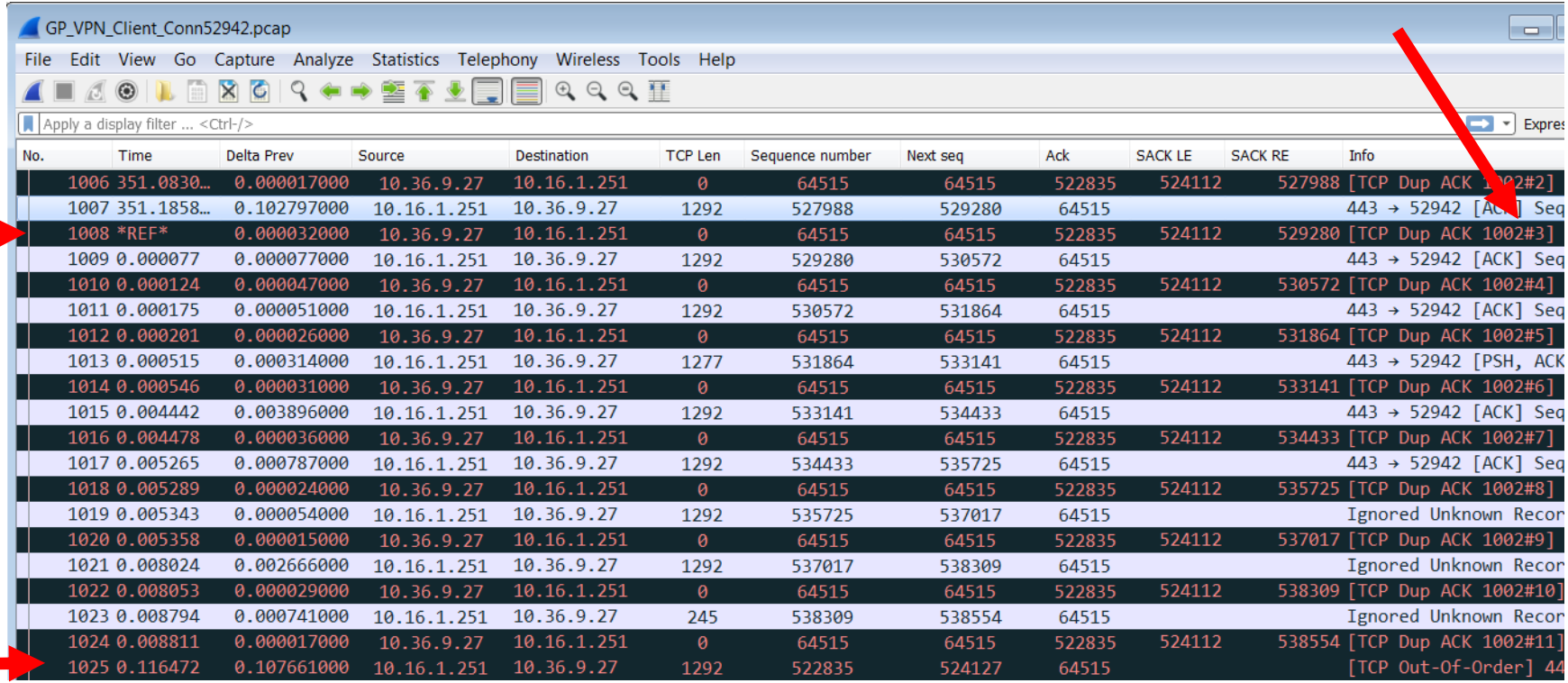


- Why 107ms from previous packet?

Wireshark Feature Parade

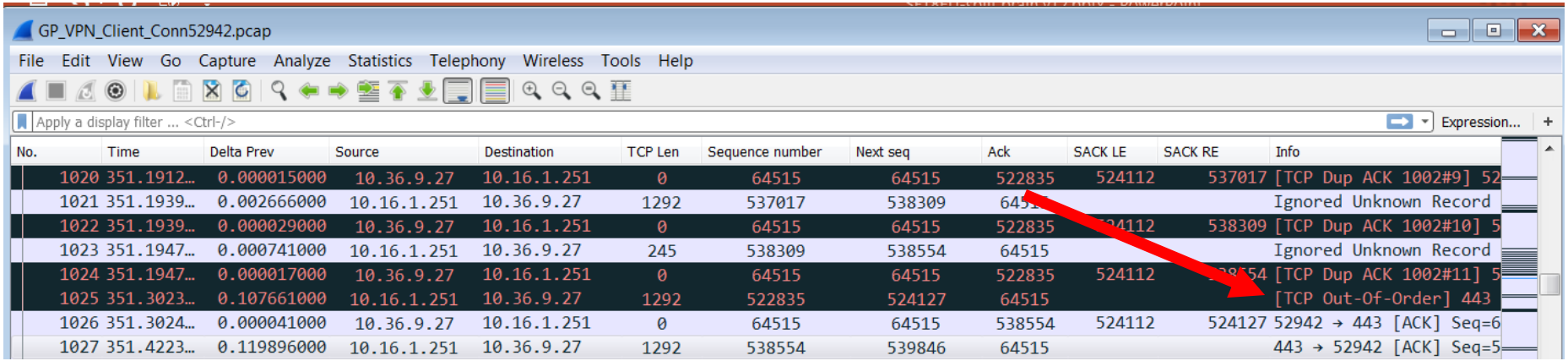


With time ref set to DupACK #3



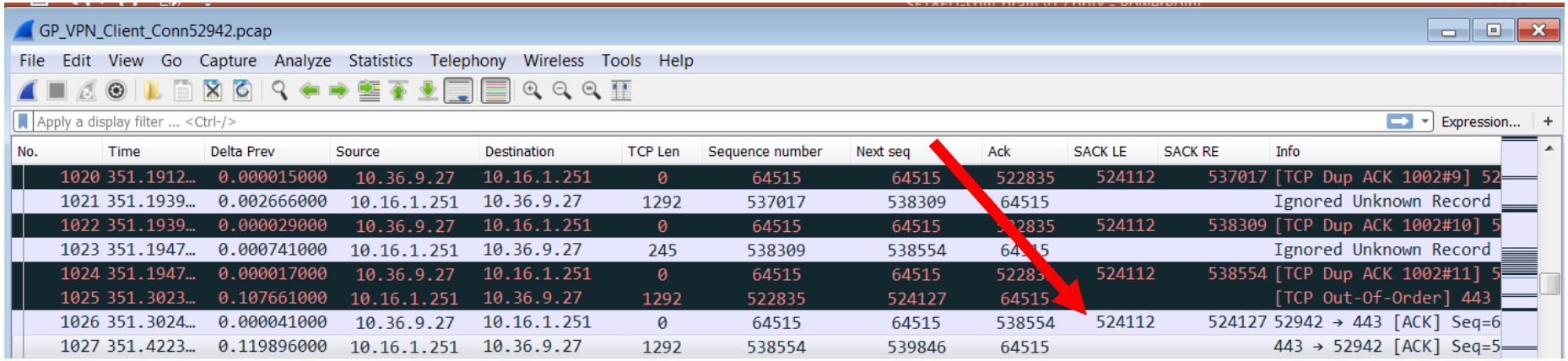
No.	Time	Delta Prev	Source	Destination	TCP Len	Sequence number	Next seq	Ack	SACK LE	SACK RE	Info
1006	351.0830…	0.000017000	10.36.9.27	10.16.1.251	0	64515	64515	522835	524112	527988	[TCP Dup ACK 1002#2]
1007	351.1858…	0.102797000	10.16.1.251	10.36.9.27	1292	527988	529280	64515			443 → 52942 [ACK] Seq
1008	*REF*	0.000032000	10.36.9.27	10.16.1.251	0	64515	64515	522835	524112	529280	[TCP Dup ACK 1002#3]
1009	0.000077	0.000077000	10.16.1.251	10.36.9.27	1292	529280	530572	64515			443 → 52942 [ACK] Seq
1010	0.000124	0.000047000	10.36.9.27	10.16.1.251	0	64515	64515	522835	524112	530572	[TCP Dup ACK 1002#4]
1011	0.000175	0.000051000	10.16.1.251	10.36.9.27	1292	530572	531864	64515			443 → 52942 [ACK] Seq
1012	0.000201	0.000026000	10.36.9.27	10.16.1.251	0	64515	64515	522835	524112	531864	[TCP Dup ACK 1002#5]
1013	0.000515	0.000314000	10.16.1.251	10.36.9.27	1277	531864	533141	64515			443 → 52942 [PSH, ACK
1014	0.000546	0.000031000	10.36.9.27	10.16.1.251	0	64515	64515	522835	524112	533141	[TCP Dup ACK 1002#6]
1015	0.004442	0.003896000	10.16.1.251	10.36.9.27	1292	533141	534433	64515			443 → 52942 [ACK] Seq
1016	0.004478	0.000036000	10.36.9.27	10.16.1.251	0	64515	64515	522835	524112	534433	[TCP Dup ACK 1002#7]
1017	0.005265	0.000787000	10.16.1.251	10.36.9.27	1292	534433	535725	64515			443 → 52942 [ACK] Seq
1018	0.005289	0.000024000	10.36.9.27	10.16.1.251	0	64515	64515	522835	524112	535725	[TCP Dup ACK 1002#8]
1019	0.005343	0.000054000	10.16.1.251	10.36.9.27	1292	535725	537017	64515			Ignored Unknown Recor
1020	0.005358	0.000015000	10.36.9.27	10.16.1.251	0	64515	64515	522835	524112	537017	[TCP Dup ACK 1002#9]
1021	0.008024	0.002666000	10.16.1.251	10.36.9.27	1292	537017	538309	64515			Ignored Unknown Recor
1022	0.008053	0.000029000	10.36.9.27	10.16.1.251	0	64515	64515	522835	524112	538309	[TCP Dup ACK 1002#10]
1023	0.008794	0.000741000	10.16.1.251	10.36.9.27	245	538309	538554	64515			Ignored Unknown Recor
1024	0.008811	0.000017000	10.36.9.27	10.16.1.251	0	64515	64515	522835	524112	538554	[TCP Dup ACK 1002#11]
1025	0.116472	0.107661000	10.16.1.251	10.36.9.27	1292	522835	524127	64515			[TCP Out-Of-Order] 44

Why “Out of Order”?



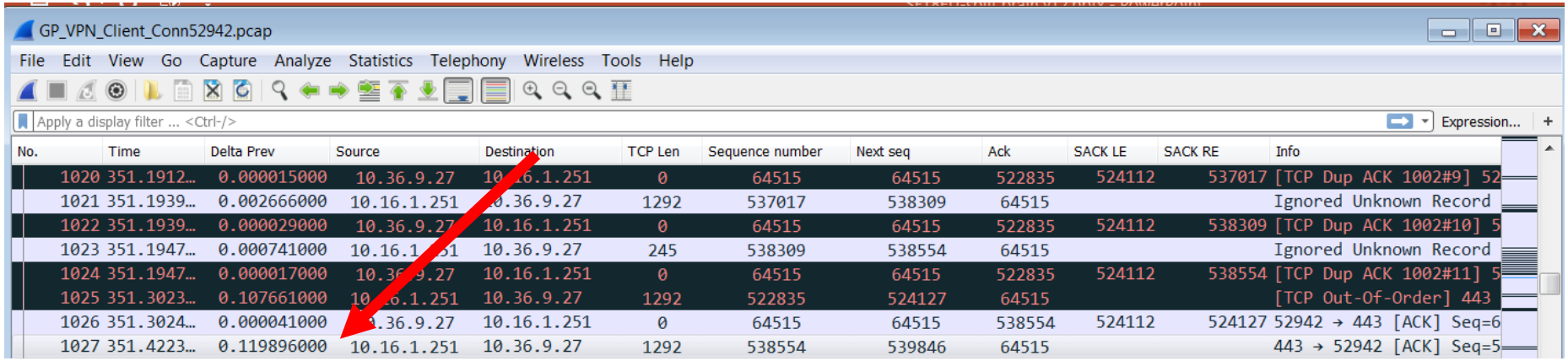
- Wouldn’t “Retrans” or “Fast Retrans” be more accurate?

Why ACK with SACK Fields?



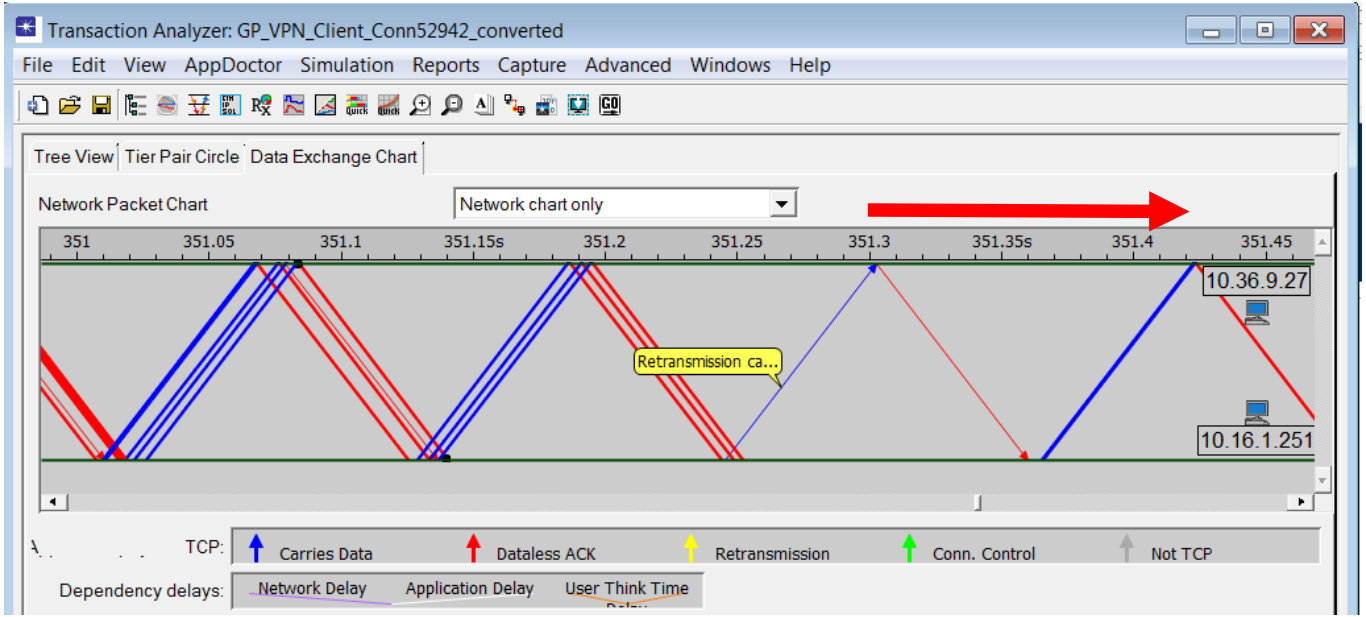
- We're all caught up, nothing is missing…so why send SACK?
- "Oh, btw Sender – I received these 15 bytes of TCP stream again unexpectedly. Just letting you know"

Why 119ms delay here?



- What does this time delta suggest about bytes in flight and the congestion window?

Here's the delay – 1 x RTT



Comparison Summary

TCP Split Brain Comparison Summary (Sender vs. Receiver)				
Item	Topic	Summary	Sender	Receiver
4.6	Fragment Overlap	Most likely caused by Segmentation Offoad	Most likely to show up on Sender's capture	Receiver could flag overlap using SACK field
5	Frame Sizes	Can be different	Effects of LSO / IP Fragmentation	Effects of LRO
6	Display Time Delta	Very unique to each endpoint	ACKs will usually apply to segments sent much earlier in time	Interpretation can seem confusing, especially when lots of packets are in flight

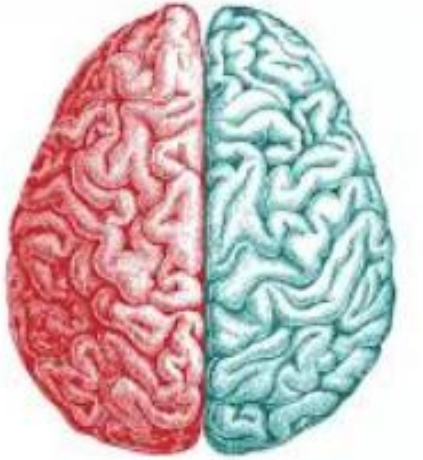
5th Leg Completed

Discussion

- Did you find this deep dive interesting?
- Anything new you've not seen before?
- Do you find visualization helpful?
- Other Comments?

Split Brain Comparisons

- Bytes in Flight

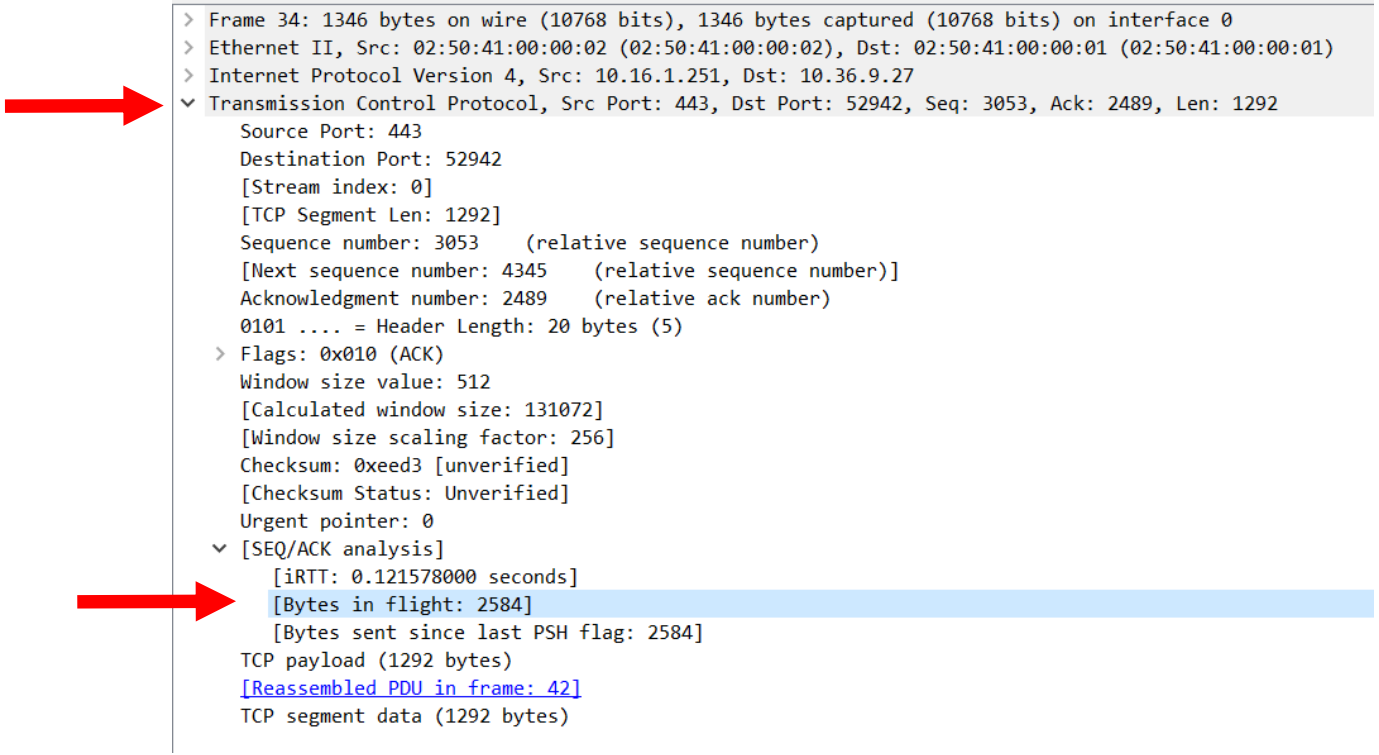


Review

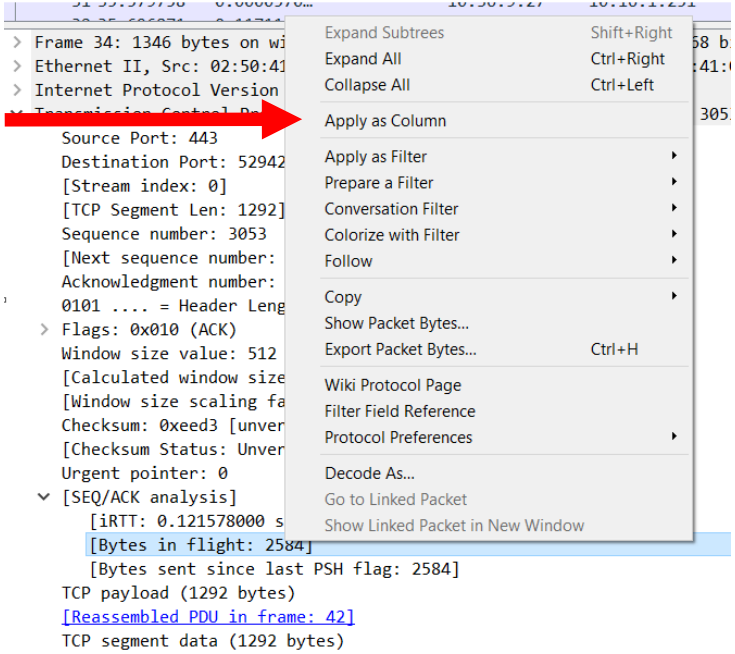
- What is the meaning of Bytes in Flight?
- How is this metric related to performance?
- BIF for sender and receiver captures look very different, let’s compare a few packet exchanges

Dry Cleaners Conveyor Belt

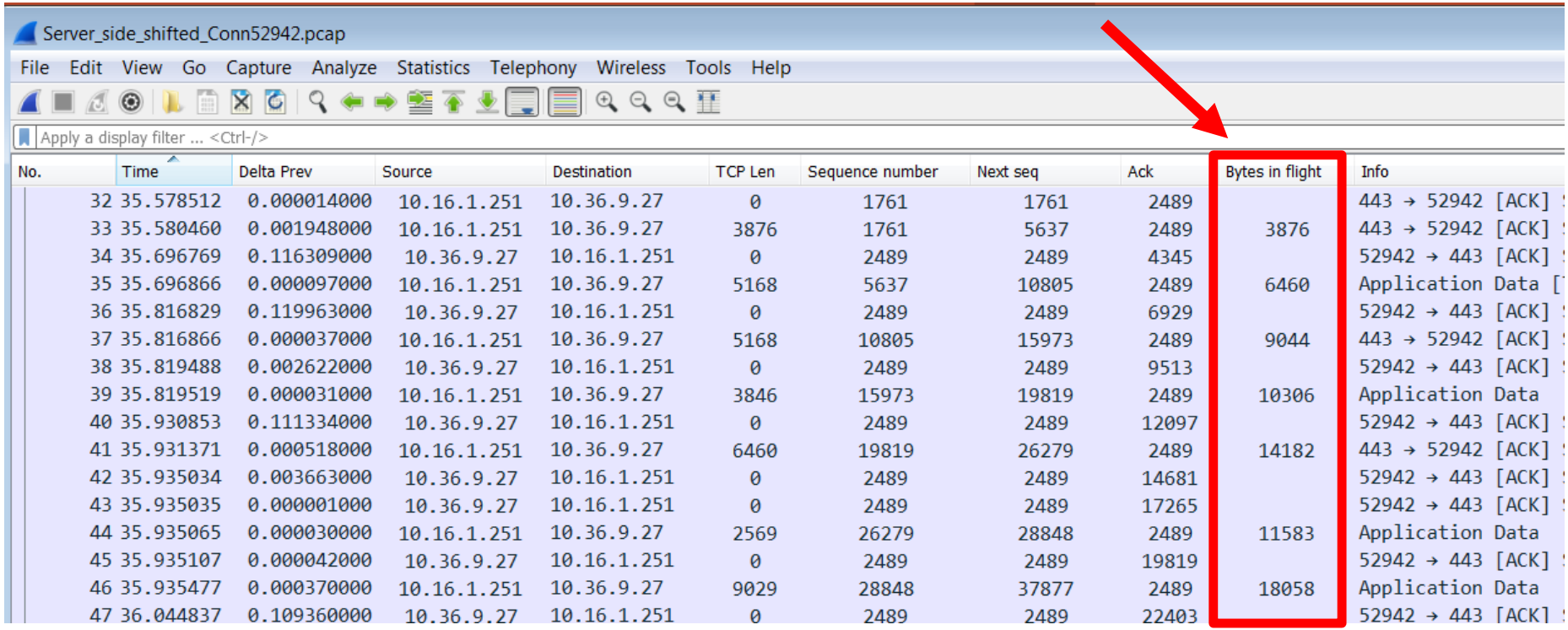
Bytes in Flight - Decode



Right Click, Apply as Column



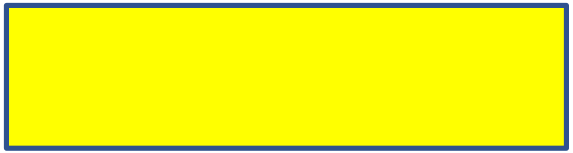
Server Side - Incrementing



No.	Time	Delta Prev	Source	Destination	TCP Len	Sequence number	Next seq	Ack	Bytes in flight	Info
32	35.578512	0.000014000	10.16.1.251	10.36.9.27	0	1761	1761	2489		443 → 52942 [ACK]
33	35.580460	0.001948000	10.16.1.251	10.36.9.27	3876	1761	5637	2489	3876	443 → 52942 [ACK]
34	35.696769	0.116309000	10.36.9.27	10.16.1.251	0	2489	2489	4345		52942 → 443 [ACK]
35	35.696866	0.000097000	10.16.1.251	10.36.9.27	5168	5637	10805	2489	6460	Application Data [
36	35.816829	0.119963000	10.36.9.27	10.16.1.251	0	2489	2489	6929		52942 → 443 [ACK]
37	35.816866	0.000037000	10.16.1.251	10.36.9.27	5168	10805	15973	2489	9044	443 → 52942 [ACK]
38	35.819488	0.002622000	10.36.9.27	10.16.1.251	0	2489	2489	9513		52942 → 443 [ACK]
39	35.819519	0.000031000	10.16.1.251	10.36.9.27	3846	15973	19819	2489	10306	Application Data
40	35.930853	0.111334000	10.36.9.27	10.16.1.251	0	2489	2489	12097		52942 → 443 [ACK]
41	35.931371	0.000518000	10.16.1.251	10.36.9.27	6460	19819	26279	2489	14182	443 → 52942 [ACK]
42	35.935034	0.003663000	10.36.9.27	10.16.1.251	0	2489	2489	14681		52942 → 443 [ACK]
43	35.935035	0.000001000	10.36.9.27	10.16.1.251	0	2489	2489	17265		52942 → 443 [ACK]
44	35.935065	0.000030000	10.16.1.251	10.36.9.27	2569	26279	28848	2489	11583	Application Data
45	35.935107	0.000042000	10.36.9.27	10.16.1.251	0	2489	2489	19819		52942 → 443 [ACK]
46	35.935477	0.000370000	10.16.1.251	10.36.9.27	9029	28848	37877	2489	18058	Application Data
47	36.044837	0.109360000	10.36.9.27	10.16.1.251	0	2489	2489	22403		52942 → 443 [ACK]

Sample - Math Drill Down

1761 5636



BIF == 3876

1761 4344 5636

BIF == 1292 (5636-4344)

ACK 4345

Send Segment #2

1761 4344 5636

BIF == 1292 5636-4344

ACK 4345

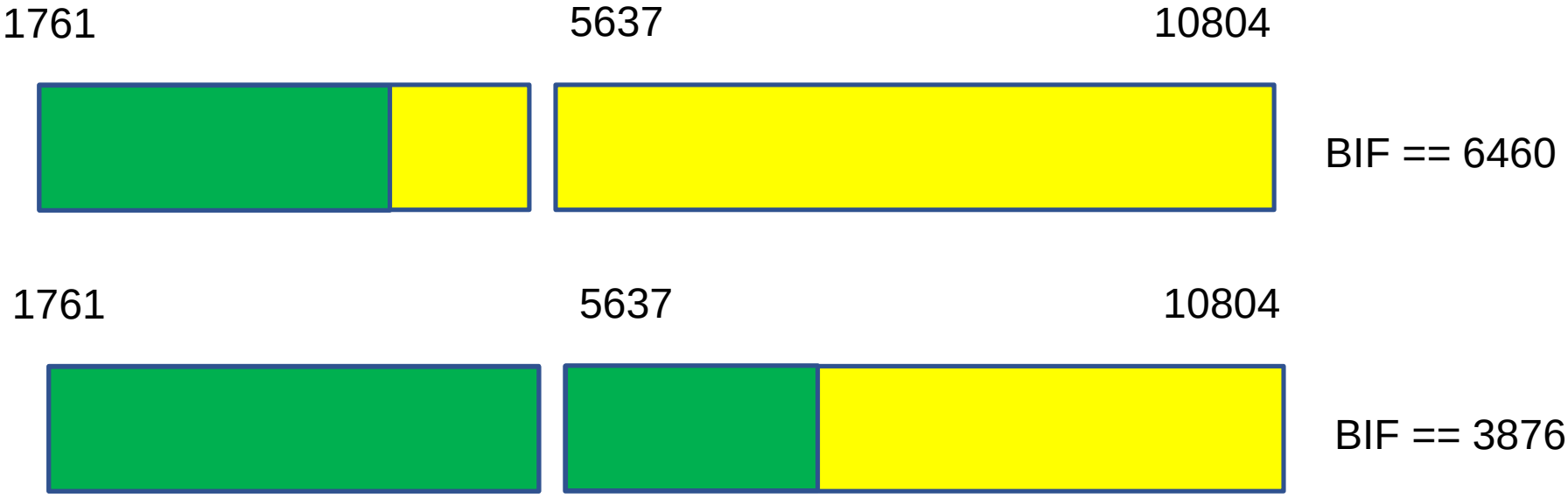
1761 5637 10804

BIF == 6460

(10804-4344)

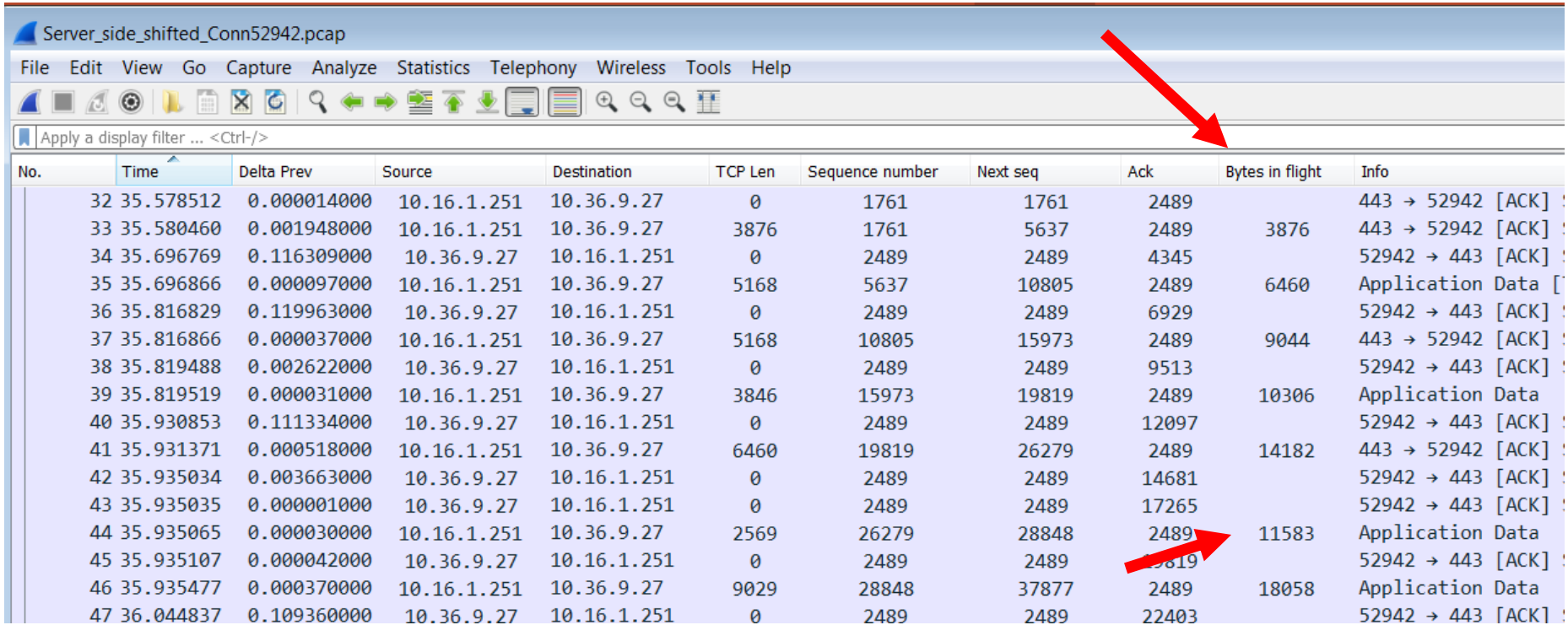
ACK 4345

Receive ACK



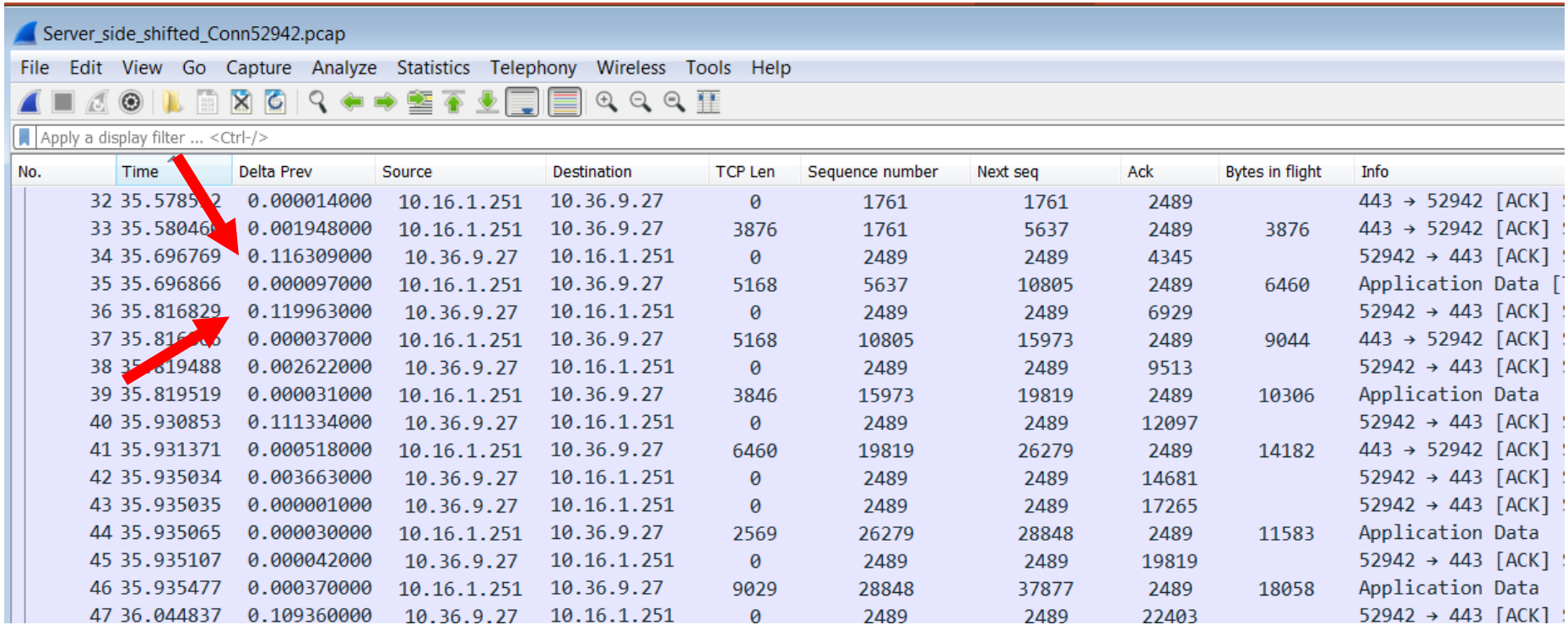
ACK 6929

Server Side – Decreasing?



No.	Time	Delta Prev	Source	Destination	TCP Len	Sequence number	Next seq	Ack	Bytes in flight	Info
32	35.578512	0.000014000	10.16.1.251	10.36.9.27	0	1761	1761	2489		443 → 52942 [ACK]
33	35.580460	0.001948000	10.16.1.251	10.36.9.27	3876	1761	5637	2489	3876	443 → 52942 [ACK]
34	35.696769	0.116309000	10.36.9.27	10.16.1.251	0	2489	2489	4345		52942 → 443 [ACK]
35	35.696866	0.000097000	10.16.1.251	10.36.9.27	5168	5637	10805	2489	6460	Application Data [
36	35.816829	0.119963000	10.36.9.27	10.16.1.251	0	2489	2489	6929		52942 → 443 [ACK]
37	35.816866	0.000037000	10.16.1.251	10.36.9.27	5168	10805	15973	2489	9044	443 → 52942 [ACK]
38	35.819488	0.002622000	10.36.9.27	10.16.1.251	0	2489	2489	9513		52942 → 443 [ACK]
39	35.819519	0.000031000	10.16.1.251	10.36.9.27	3846	15973	19819	2489	10306	Application Data
40	35.930853	0.111334000	10.36.9.27	10.16.1.251	0	2489	2489	12097		52942 → 443 [ACK]
41	35.931371	0.000518000	10.16.1.251	10.36.9.27	6460	19819	26279	2489	14182	443 → 52942 [ACK]
42	35.935034	0.003663000	10.36.9.27	10.16.1.251	0	2489	2489	14681		52942 → 443 [ACK]
43	35.935035	0.000001000	10.36.9.27	10.16.1.251	0	2489	2489	17265		52942 → 443 [ACK]
44	35.935065	0.000030000	10.16.1.251	10.36.9.27	2569	26279	28848	2489	11583	Application Data
45	35.935107	0.000042000	10.36.9.27	10.16.1.251	0	2489	2489	[illegible]		52942 → 443 [ACK]
46	35.935477	0.000370000	10.16.1.251	10.36.9.27	9029	28848	37877	2489	18058	Application Data
47	36.044837	0.109360000	10.36.9.27	10.16.1.251	0	2489	2489	22403		52942 → 443 [ACK]

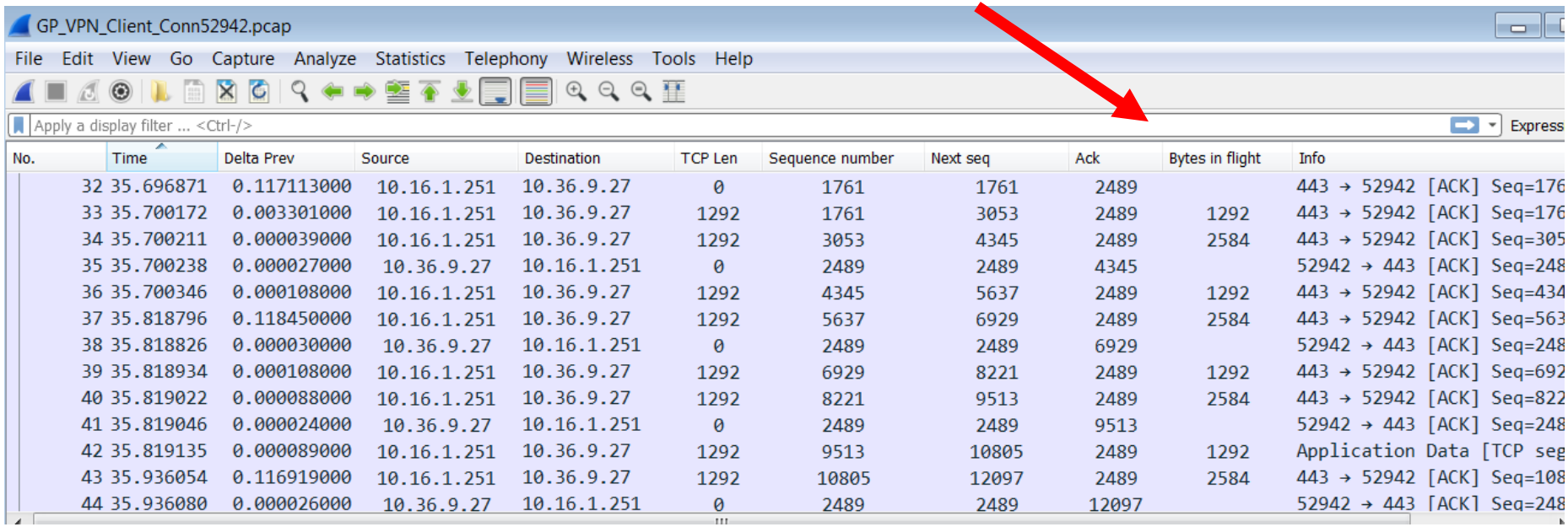
Server Side - Timing



No.	Time	Delta Prev	Source	Destination	TCP Len	Sequence number	Next seq	Ack	Bytes in flight	Info
32	35.5785[illegible]2	0.000014000	10.16.1.251	10.36.9.27	0	1761	1761	2489		443 → 52942 [ACK]
33	35.5804[illegible]	0.001948000	10.16.1.251	10.36.9.27	3876	1761	5637	2489	3876	443 → 52942 [ACK]
34	35.696769	0.116309000	10.36.9.27	10.16.1.251	0	2489	2489	4345		52942 → 443 [ACK]
35	35.696866	0.000097000	10.16.1.251	10.36.9.27	5168	5637	10805	2489	6460	Application Data [
36	35.816829	0.119963000	10.36.9.27	10.16.1.251	0	2489	2489	6929		52942 → 443 [ACK]
37	35.816[illegible]6	0.000037000	10.16.1.251	10.36.9.27	5168	10805	15973	2489	9044	443 → 52942 [ACK]
38	35.819488	0.002622000	10.36.9.27	10.16.1.251	0	2489	2489	9513		52942 → 443 [ACK]
39	35.819519	0.000031000	10.16.1.251	10.36.9.27	3846	15973	19819	2489	10306	Application Data
40	35.930853	0.111334000	10.36.9.27	10.16.1.251	0	2489	2489	12097		52942 → 443 [ACK]
41	35.931371	0.000518000	10.16.1.251	10.36.9.27	6460	19819	26279	2489	14182	443 → 52942 [ACK]
42	35.935034	0.003663000	10.36.9.27	10.16.1.251	0	2489	2489	14681		52942 → 443 [ACK]
43	35.935035	0.000001000	10.36.9.27	10.16.1.251	0	2489	2489	17265		52942 → 443 [ACK]
44	35.935065	0.000030000	10.16.1.251	10.36.9.27	2569	26279	28848	2489	11583	Application Data
45	35.935107	0.000042000	10.36.9.27	10.16.1.251	0	2489	2489	19819		52942 → 443 [ACK]
46	35.935477	0.000370000	10.16.1.251	10.36.9.27	9029	28848	37877	2489	18058	Application Data
47	36.044837	0.109360000	10.36.9.27	10.16.1.251	0	2489	2489	22403		52942 → 443 [ACK]

Let’s jump to Receiver now...

Client Side



No.	Time	Delta Prev	Source	Destination	TCP Len	Sequence number	Next seq	Ack	Bytes in flight	Info
32	35.696871	0.117113000	10.16.1.251	10.36.9.27	0	1761	1761	2489		443 → 52942 [ACK] Seq=176
33	35.700172	0.003301000	10.16.1.251	10.36.9.27	1292	1761	3053	2489	1292	443 → 52942 [ACK] Seq=176
34	35.700211	0.000039000	10.16.1.251	10.36.9.27	1292	3053	4345	2489	2584	443 → 52942 [ACK] Seq=305
35	35.700238	0.000027000	10.36.9.27	10.16.1.251	0	2489	2489	4345		52942 → 443 [ACK] Seq=248
36	35.700346	0.000108000	10.16.1.251	10.36.9.27	1292	4345	5637	2489	1292	443 → 52942 [ACK] Seq=434
37	35.818796	0.118450000	10.16.1.251	10.36.9.27	1292	5637	6929	2489	2584	443 → 52942 [ACK] Seq=563
38	35.818826	0.000030000	10.36.9.27	10.16.1.251	0	2489	2489	6929		52942 → 443 [ACK] Seq=248
39	35.818934	0.000108000	10.16.1.251	10.36.9.27	1292	6929	8221	2489	1292	443 → 52942 [ACK] Seq=692
40	35.819022	0.000088000	10.16.1.251	10.36.9.27	1292	8221	9513	2489	2584	443 → 52942 [ACK] Seq=822
41	35.819046	0.000024000	10.36.9.27	10.16.1.251	0	2489	2489	9513		52942 → 443 [ACK] Seq=248
42	35.819135	0.000089000	10.16.1.251	10.36.9.27	1292	9513	10805	2489	1292	Application Data [TCP seg
43	35.936054	0.116919000	10.16.1.251	10.36.9.27	1292	10805	12097	2489	2584	443 → 52942 [ACK] Seq=108
44	35.936080	0.000026000	10.36.9.27	10.16.1.251	0	2489	2489	12097		52942 → 443 [ACK] Seq=248

What else can we learn from BIF?

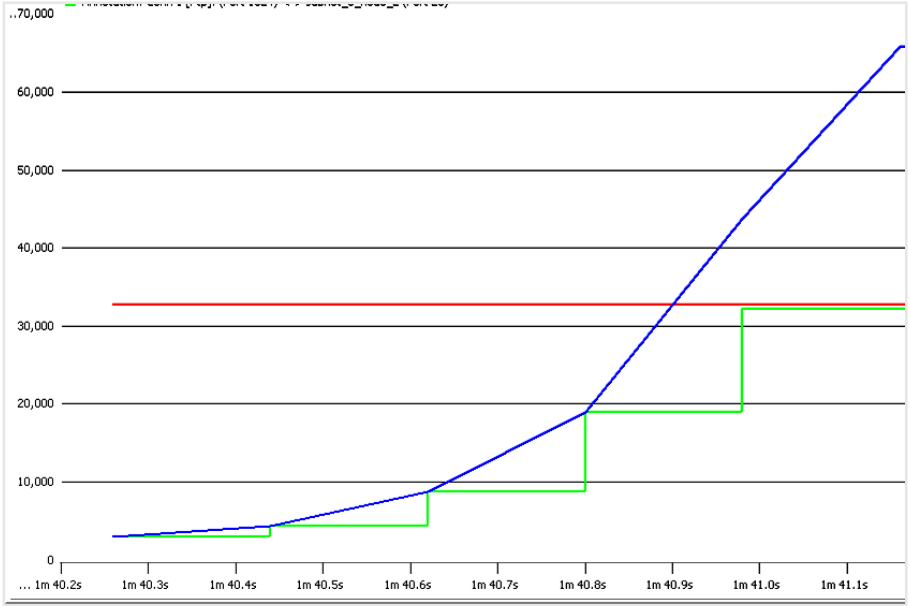
- So far, we've seen how BIF can inform about network health & congestion window
- We can also "get a sense" of Send Buffer sizing and the application's TCP API options

Inferring Send Buffer Size

- Bytes In-flight can give you some insight into how the Send Buffer size might be limiting the Congestion Window

- How can we find the maximum observed Bytes In-Flight?

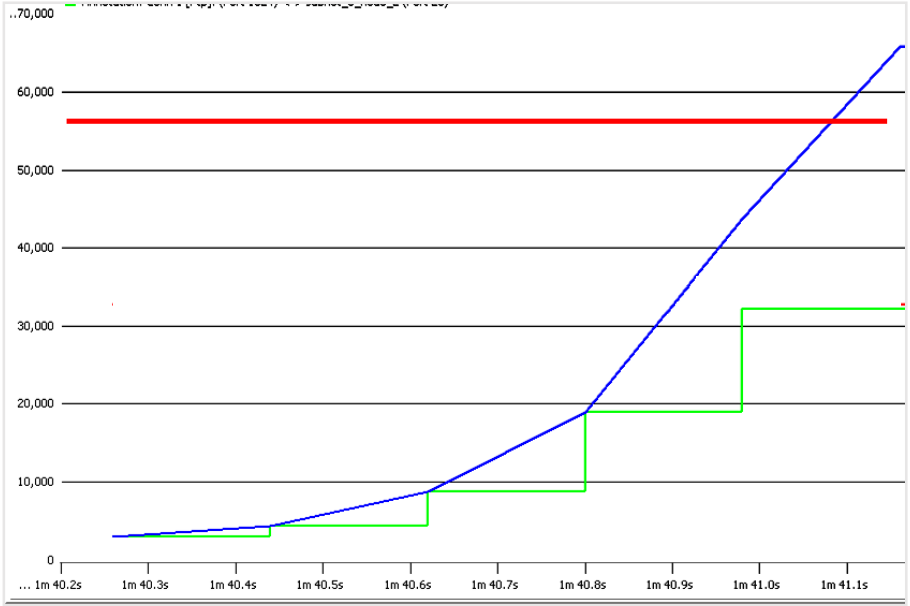
Throughput Throttling



- Congestion Control Window
- Receive Buffer
- In-Flight Data

Notice how green line never gets above red line

Throughput Throttling



- Congestion Control Window
- Receive Buffer
- In-Flight Data

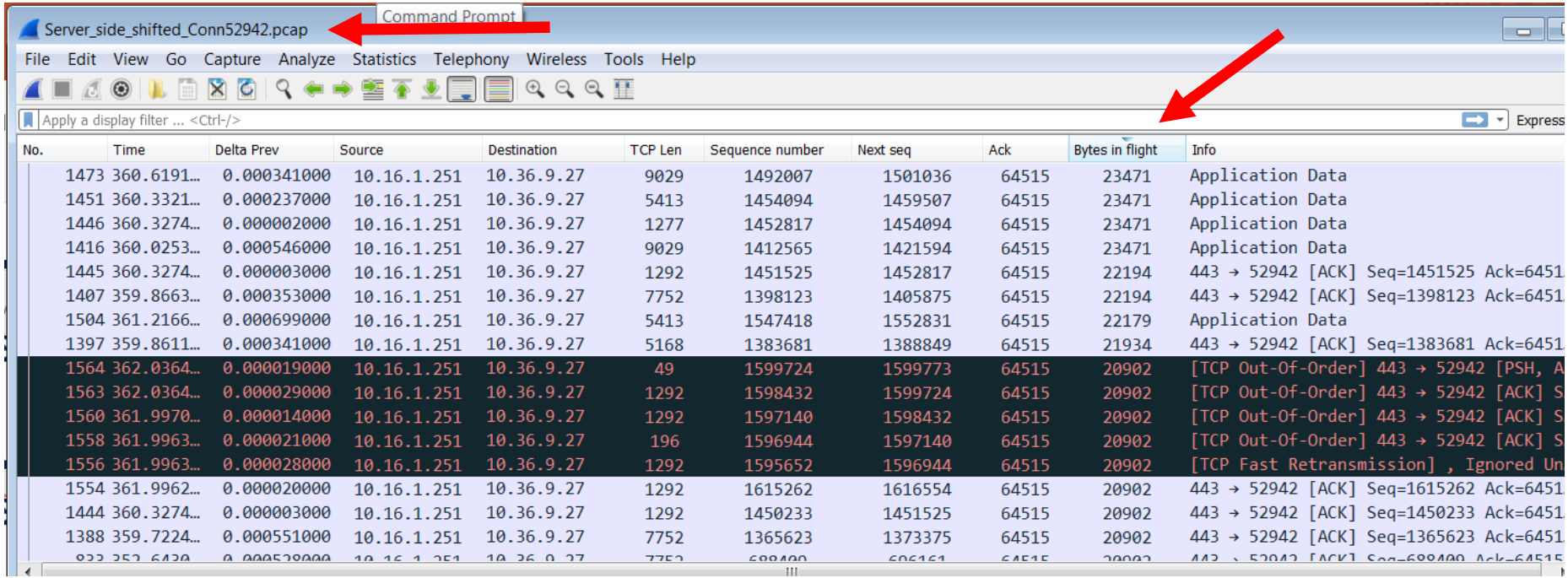
What's holding us back now?

Inferring Send Buffer Size

- Bytes In-flight can give you some insight into how the Send Buffer size might be limiting Throughput

- How can we find the maximum observed Bytes In-Flight?

Server Capture – Sort by BIF



No.	Time	Delta Prev	Source	Destination	TCP Len	Sequence number	Next seq	Ack	Bytes in flight	Info
1473	360.6191…	0.000341000	10.16.1.251	10.36.9.27	9029	1492007	1501036	64515	23471	Application Data
1451	360.3321…	0.000237000	10.16.1.251	10.36.9.27	5413	1454094	1459507	64515	23471	Application Data
1446	360.3274…	0.000002000	10.16.1.251	10.36.9.27	1277	1452817	1454094	64515	23471	Application Data
1416	360.0253…	0.000546000	10.16.1.251	10.36.9.27	9029	1412565	1421594	64515	23471	Application Data
1445	360.3274…	0.000003000	10.16.1.251	10.36.9.27	1292	1451525	1452817	64515	22194	443 → 52942 [ACK] Seq=1451525 Ack=6451
1407	359.8663…	0.000353000	10.16.1.251	10.36.9.27	7752	1398123	1405875	64515	22194	443 → 52942 [ACK] Seq=1398123 Ack=6451
1504	361.2166…	0.000699000	10.16.1.251	10.36.9.27	5413	1547418	1552831	64515	22179	Application Data
1397	359.8611…	0.000341000	10.16.1.251	10.36.9.27	5168	1383681	1388849	64515	21934	443 → 52942 [ACK] Seq=1383681 Ack=6451
1564	362.0364…	0.000019000	10.16.1.251	10.36.9.27	49	1599724	1599773	64515	20902	[TCP Out-Of-Order] 443 → 52942 [PSH, A
1563	362.0364…	0.000029000	10.16.1.251	10.36.9.27	1292	1598432	1599724	64515	20902	[TCP Out-Of-Order] 443 → 52942 [ACK] S
1560	361.9970…	0.000014000	10.16.1.251	10.36.9.27	1292	1597140	1598432	64515	20902	[TCP Out-Of-Order] 443 → 52942 [ACK] S
1558	361.9963…	0.000021000	10.16.1.251	10.36.9.27	196	1596944	1597140	64515	20902	[TCP Out-Of-Order] 443 → 52942 [ACK] S
1556	361.9963…	0.000028000	10.16.1.251	10.36.9.27	1292	1595652	1596944	64515	20902	[TCP Fast Retransmission] , Ignored Un
1554	361.9962…	0.000020000	10.16.1.251	10.36.9.27	1292	1615262	1616554	64515	20902	443 → 52942 [ACK] Seq=1615262 Ack=6451
1444	360.3274…	0.000003000	10.16.1.251	10.36.9.27	1292	1450233	1451525	64515	20902	443 → 52942 [ACK] Seq=1450233 Ack=6451
1388	359.7224…	0.000551000	10.16.1.251	10.36.9.27	7752	1365623	1373375	64515	20902	443 → 52942 [ACK] Seq=1365623 Ack=6451

Discussion

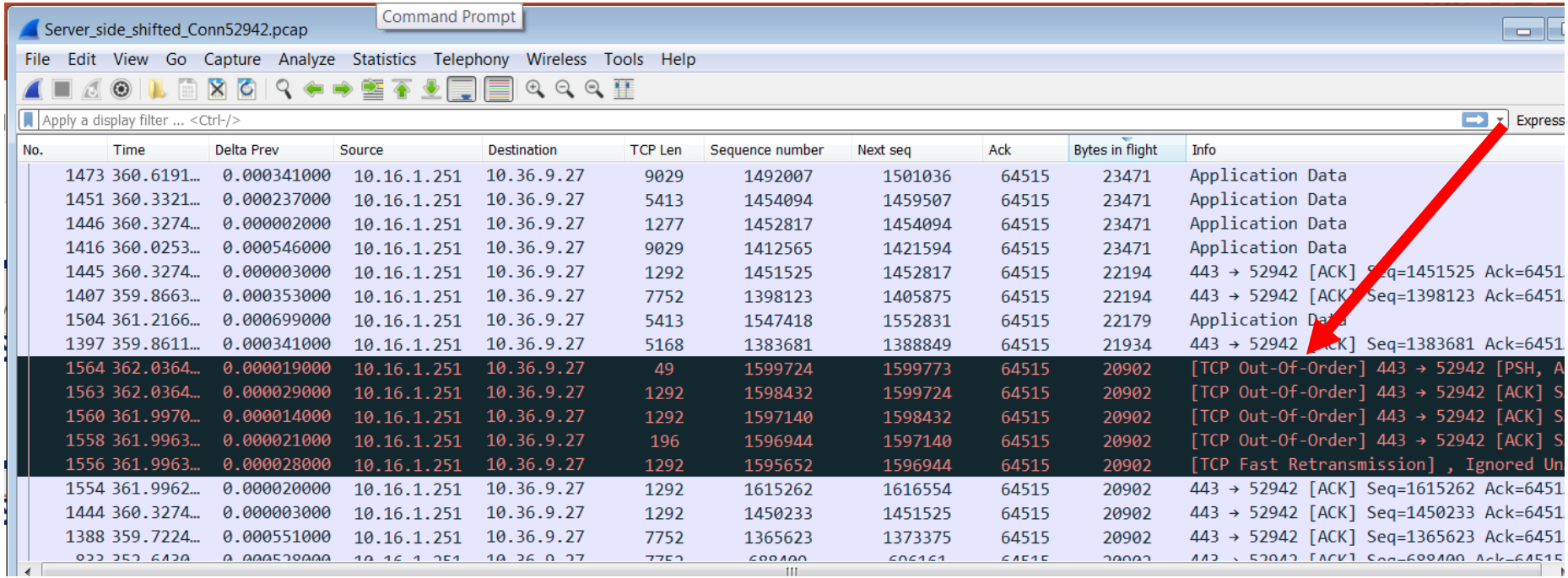
- Assume that network is healthy + TCP Congestion Window is “happy” + receive window is higher than BIF

- What does this infer about Send Buffer size and implementation?

Btw, something was odd…

- Did you notice anything odd in the last decode summary screen?
- Let’s look again…

Server Capture – Out of order?

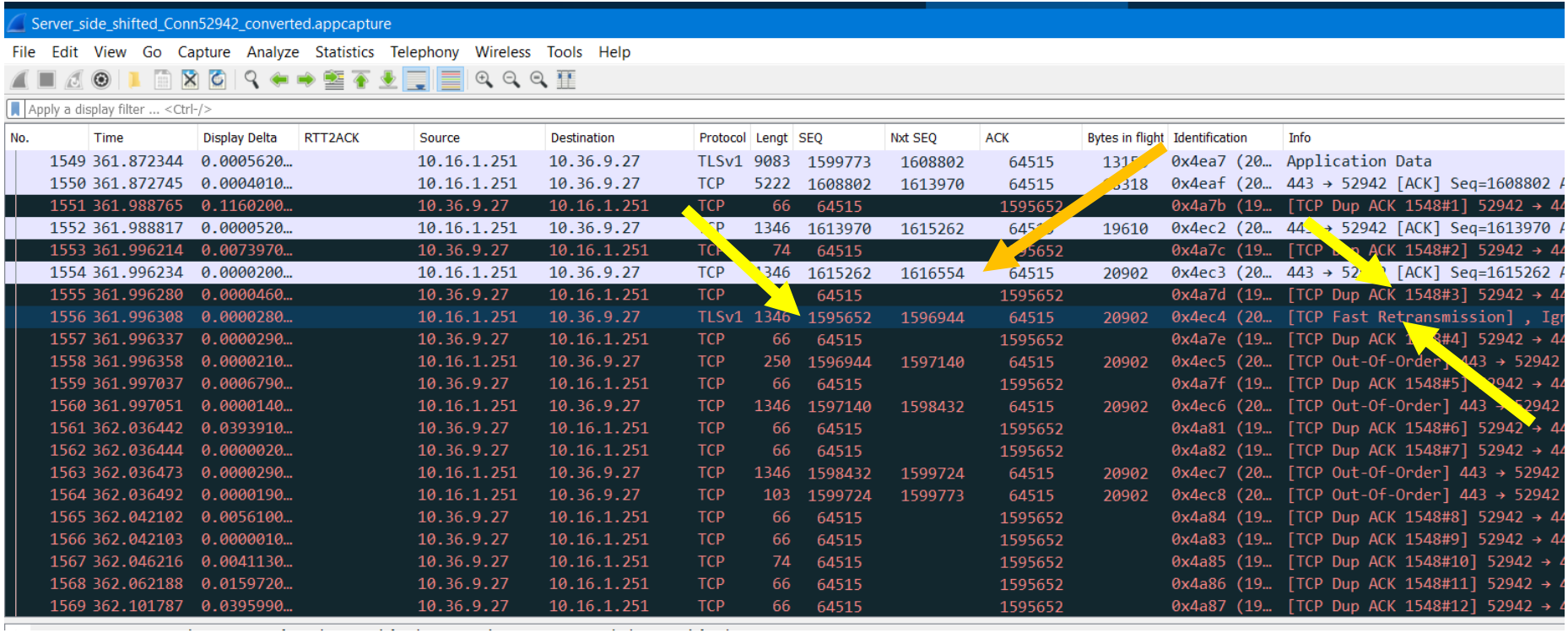


Quick Detour via OOS...

How can sender segments be OOO?

- Is this even possible?
- Why would this happen?
- Let’s re-sort and have another look…

Zoom in...sort by frame



Interpretation vs. Decode

TCP Out-Of-Order

Set when all of the following are true:

- This is not a keepalive packet.
- In the forward direction, the segment length is greater than zero or the SYN or FIN is set.
- The next expected sequence number is greater than the current sequence number.
- The next expected sequence number and the next sequence number differ.
- The last segment arrived within the calculated RTT (3ms by default).

Supersedes “Spurious Retransmission” and “Retransmission”.

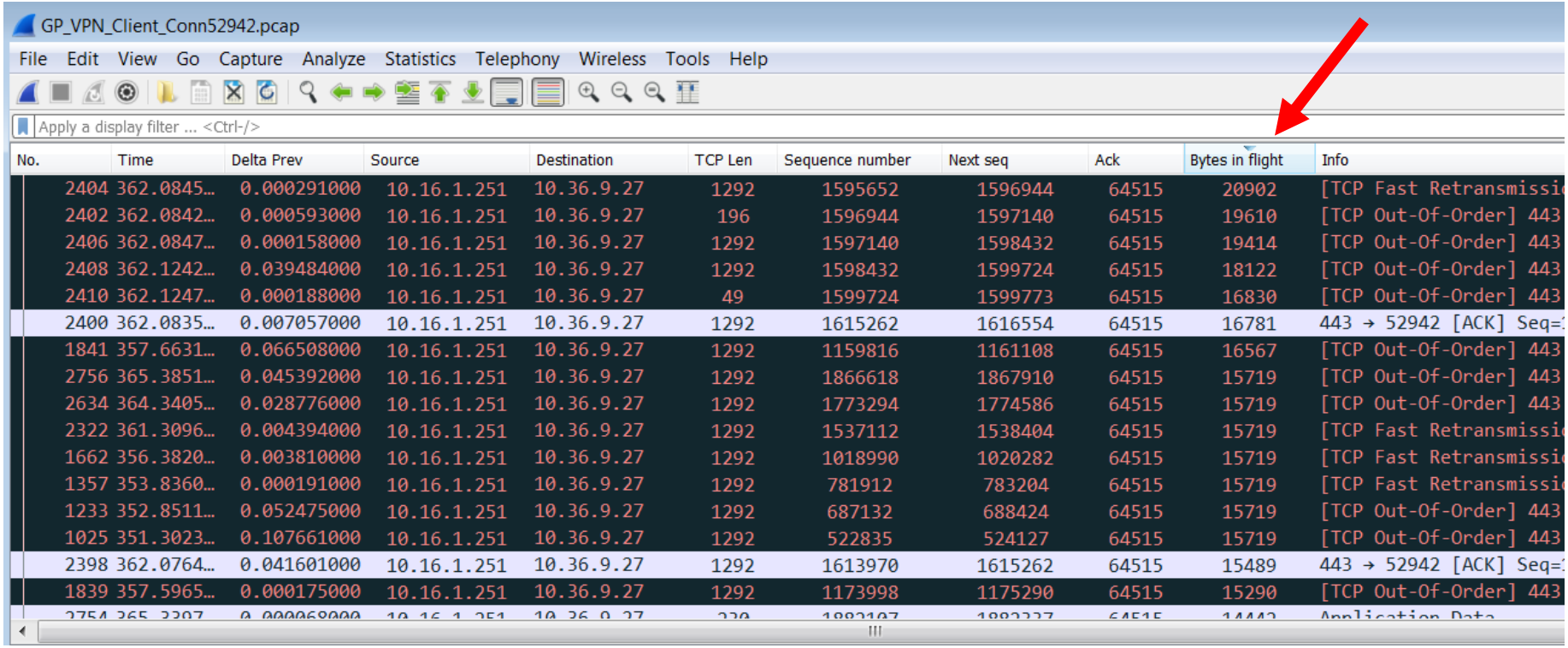
Why this is important...

- Heuristics, though generally pretty good, but they aren't perfect
- Corner cases within corner cases…
- If you relied solely on the Expert Info summary you might incorrectly conclude the network is messing with packet order (more than it really is…)
- Human interpretation is key

Discussion

OK, back to BIF...

BIF - Receiver Side

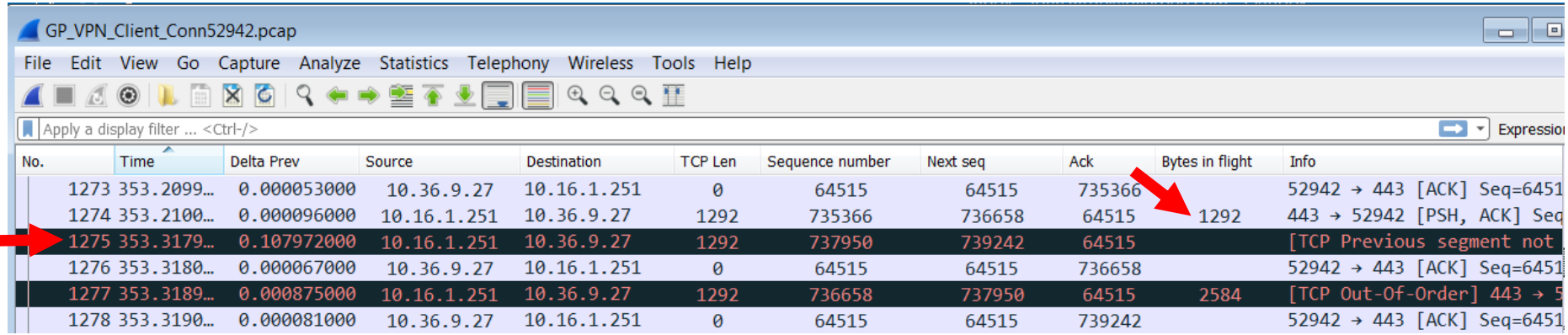


No.	Time	Delta Prev	Source	Destination	TCP Len	Sequence number	Next seq	Ack	Bytes in flight	Info
2404	362.0845…	0.000291000	10.16.1.251	10.36.9.27	1292	1595652	1596944	64515	20902	[TCP Fast Retransmissi
2402	362.0842…	0.000593000	10.16.1.251	10.36.9.27	196	1596944	1597140	64515	19610	[TCP Out-Of-Order] 443
2406	362.0847…	0.000158000	10.16.1.251	10.36.9.27	1292	1597140	1598432	64515	19414	[TCP Out-Of-Order] 443
2408	362.1242…	0.039484000	10.16.1.251	10.36.9.27	1292	1598432	1599724	64515	18122	[TCP Out-Of-Order] 443
2410	362.1247…	0.000188000	10.16.1.251	10.36.9.27	49	1599724	1599773	64515	16830	[TCP Out-Of-Order] 443
2400	362.0835…	0.007057000	10.16.1.251	10.36.9.27	1292	1615262	1616554	64515	16781	443 → 52942 [ACK] Seq=
1841	357.6631…	0.066508000	10.16.1.251	10.36.9.27	1292	1159816	1161108	64515	16567	[TCP Out-Of-Order] 443
2756	365.3851…	0.045392000	10.16.1.251	10.36.9.27	1292	1866618	1867910	64515	15719	[TCP Out-Of-Order] 443
2634	364.3405…	0.028776000	10.16.1.251	10.36.9.27	1292	1773294	1774586	64515	15719	[TCP Out-Of-Order] 443
2322	361.3096…	0.004394000	10.16.1.251	10.36.9.27	1292	1537112	1538404	64515	15719	[TCP Fast Retransmissi
1662	356.3820…	0.003810000	10.16.1.251	10.36.9.27	1292	1018990	1020282	64515	15719	[TCP Fast Retransmissi
1357	353.8360…	0.000191000	10.16.1.251	10.36.9.27	1292	781912	783204	64515	15719	[TCP Fast Retransmissi
1233	352.8511…	0.052475000	10.16.1.251	10.36.9.27	1292	687132	688424	64515	15719	[TCP Out-Of-Order] 443
1025	351.3023…	0.107661000	10.16.1.251	10.36.9.27	1292	522835	524127	64515	15719	[TCP Out-Of-Order] 443
2398	362.0764…	0.041601000	10.16.1.251	10.36.9.27	1292	1613970	1615262	64515	15489	443 → 52942 [ACK] Seq=
1839	357.5965…	0.000175000	10.16.1.251	10.36.9.27	1292	1173998	1175290	64515	15290	[TCP Out-Of-Order] 443

In-Flight & OOS / Loss

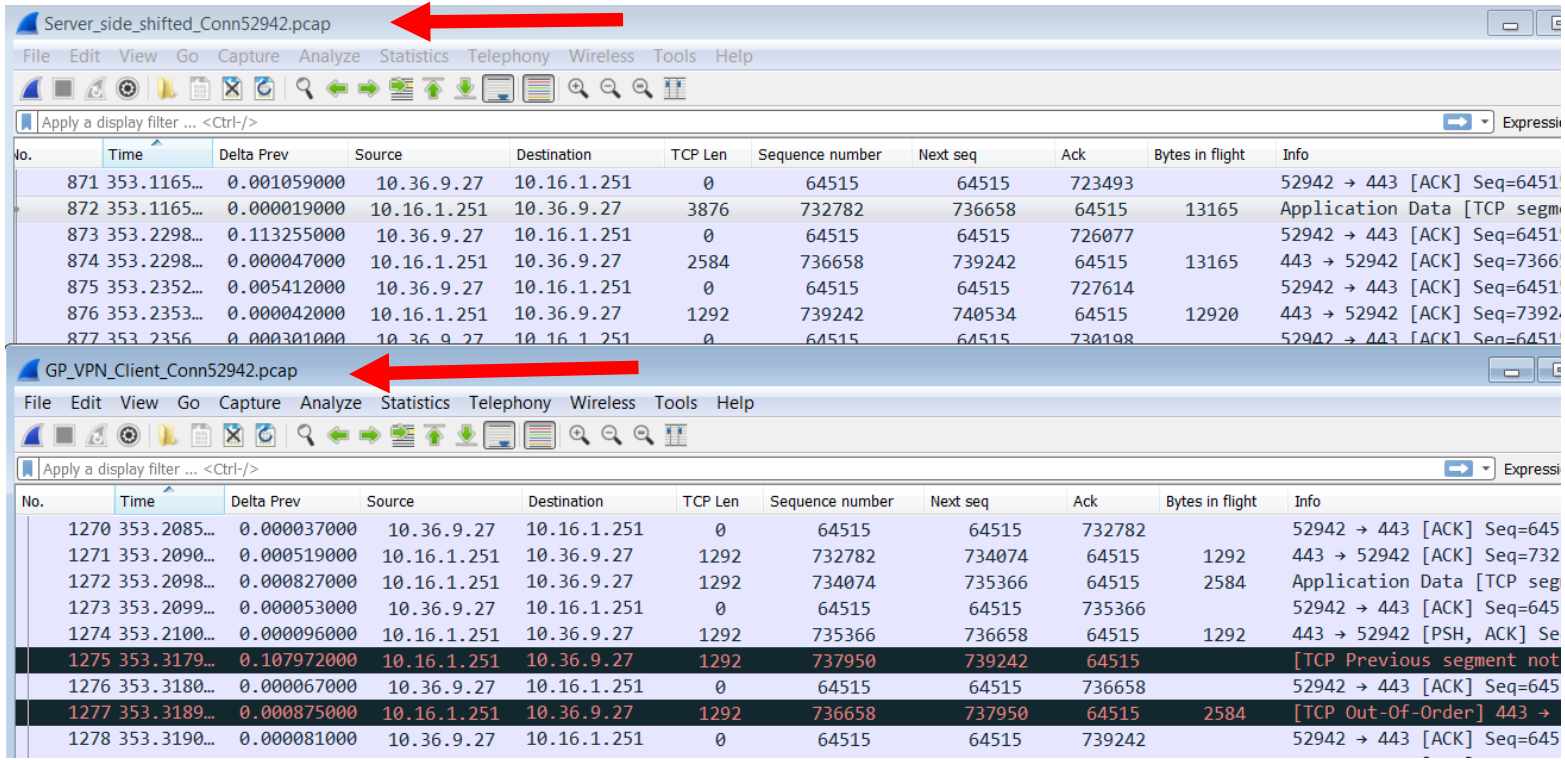
- Let’s look at how Wireshark calculates Bytes in-flight when there is packet loss or OOS

Client – Receive #1275 OOS

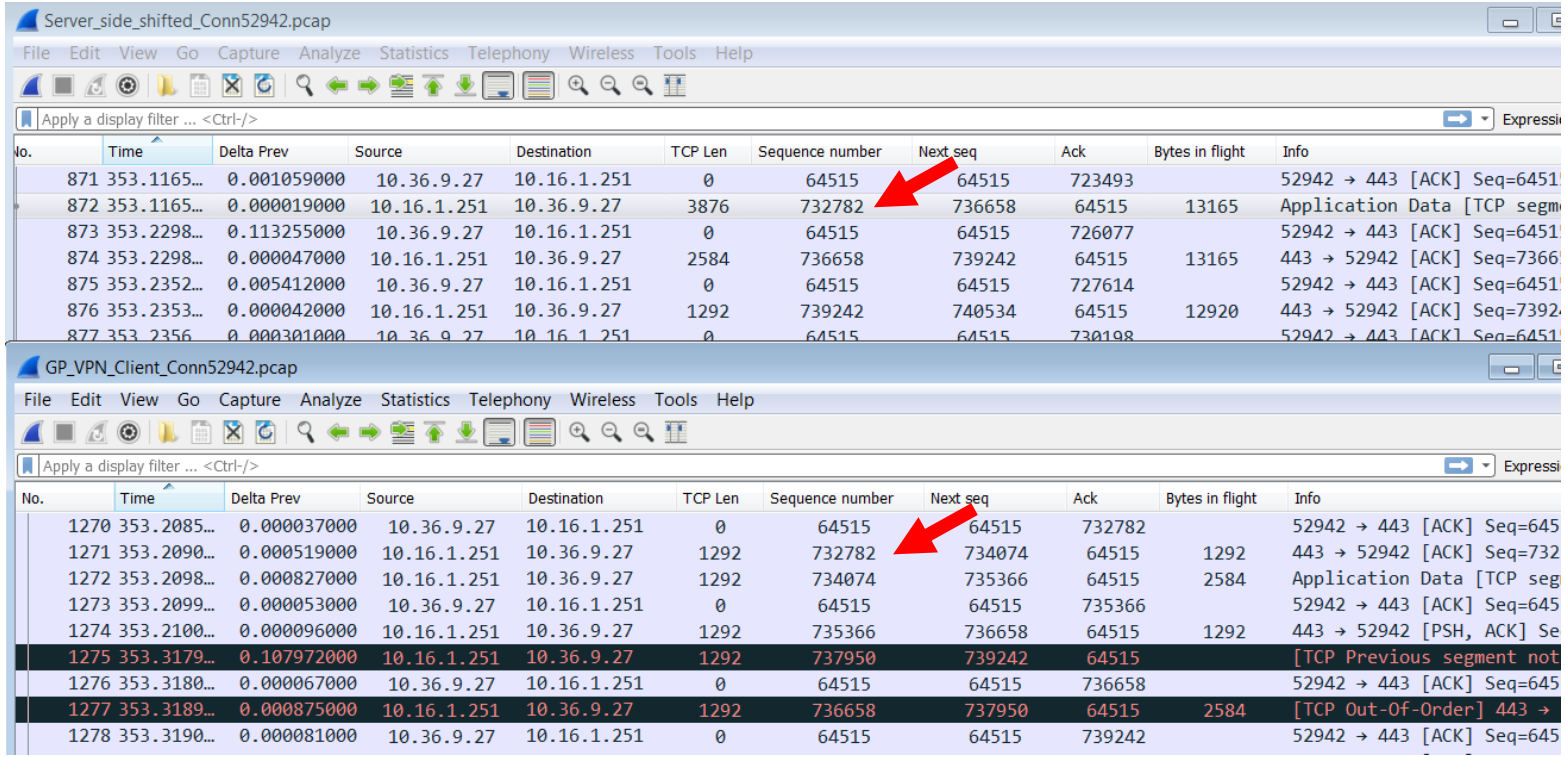


- BIF did not increment for #1275

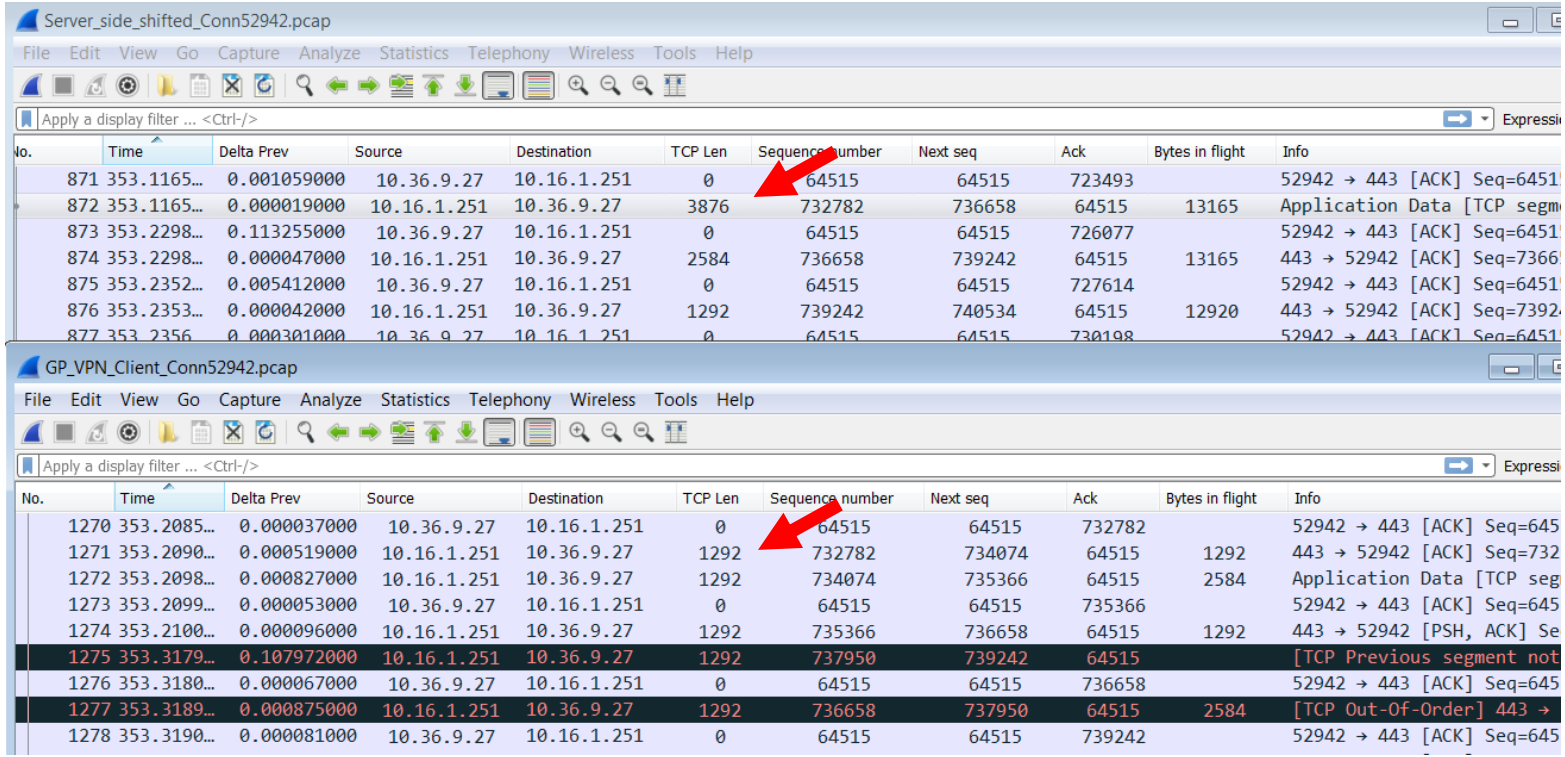
Comparison Storyboard #1



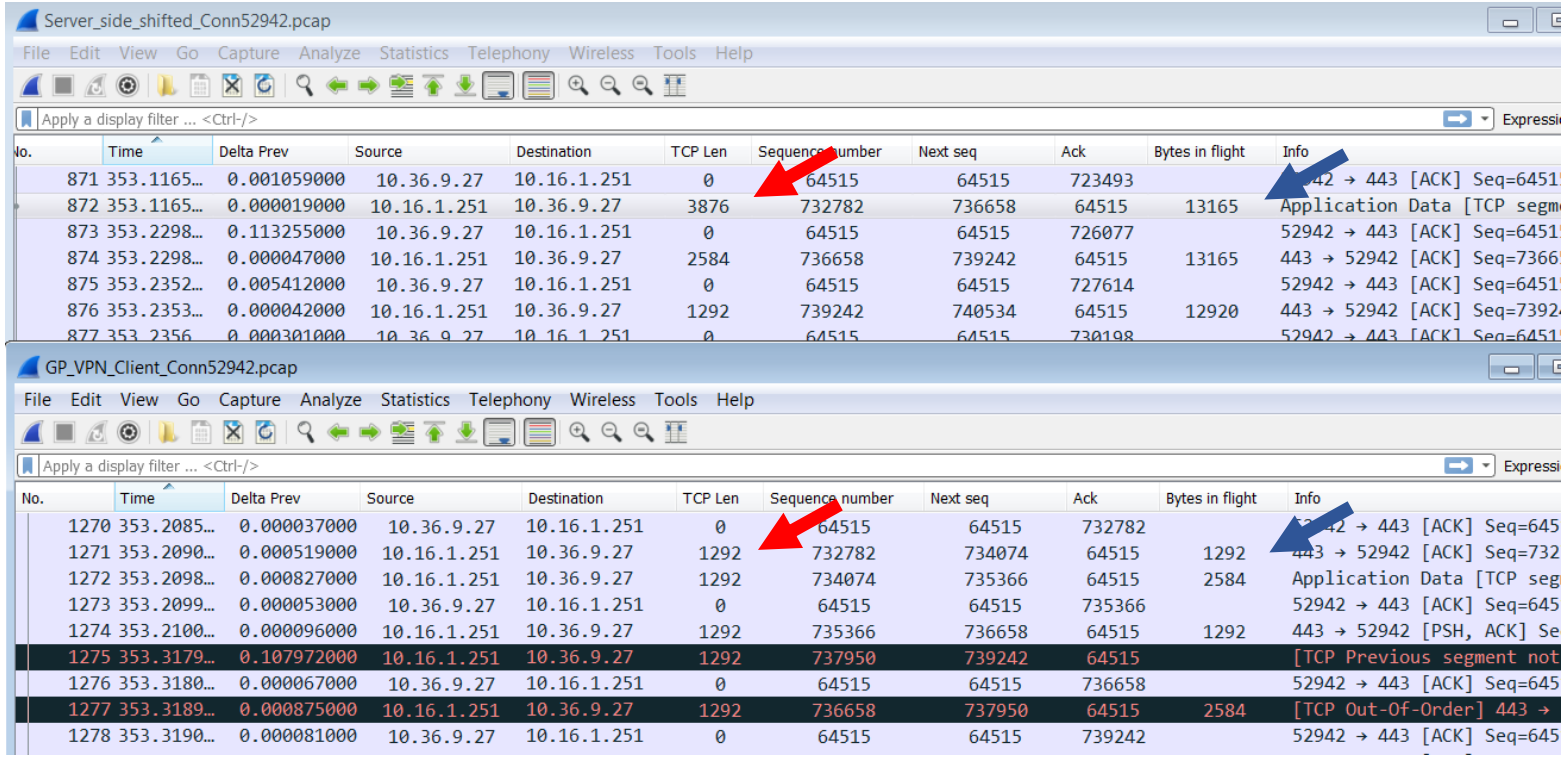
Comparison Storyboard #2



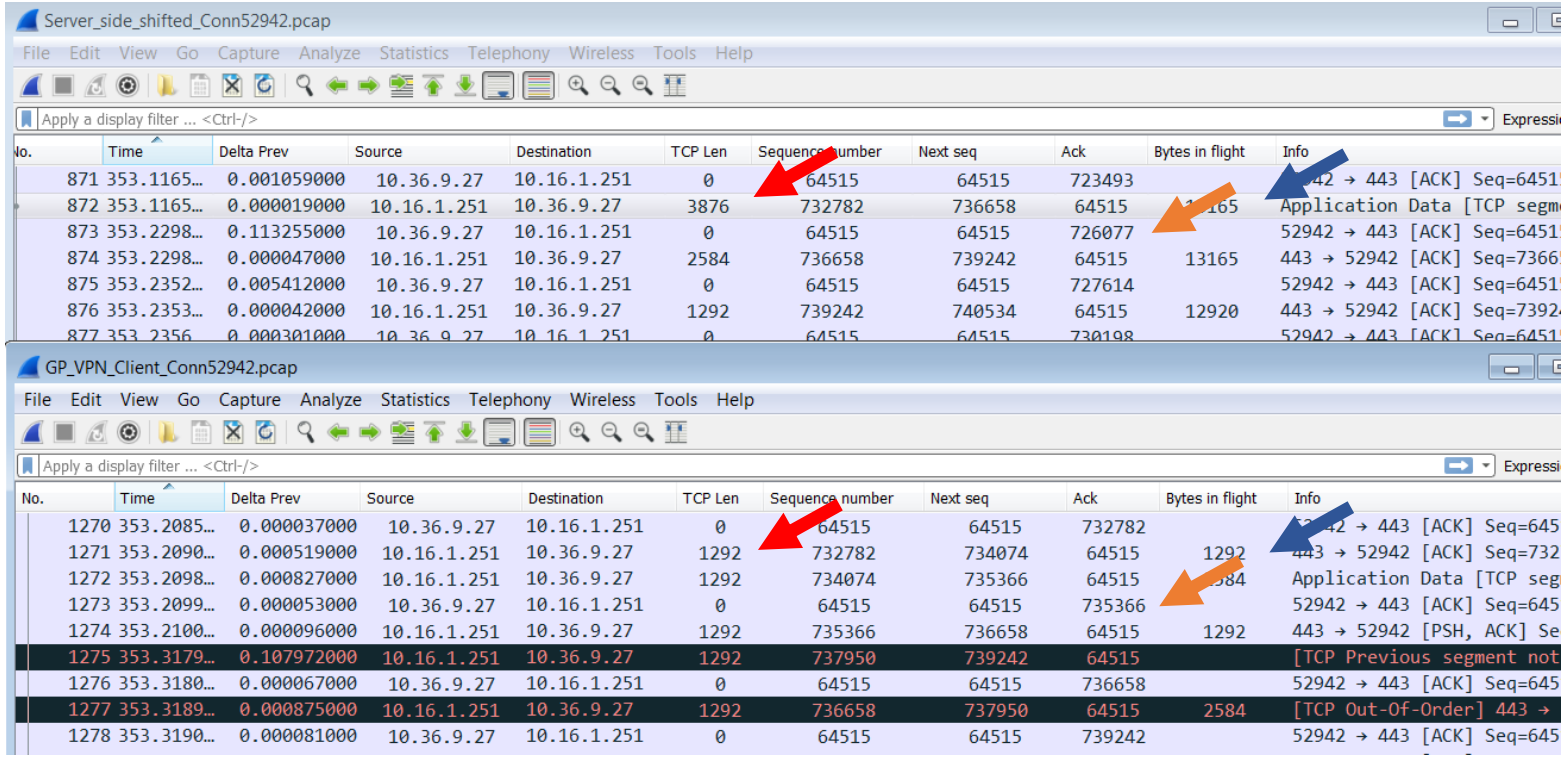
Comparison Storyboard #3



Comparison Storyboard #4



Comparison Storyboard #5



Why doesn’t BIF Increment

Discussion

- We've compared based the stream seq #
- We've looked at LEN, ACK, BIF, and packet order differences
- We tried to compare the "when", but this is a real Brain Bender

We need to visualize this…

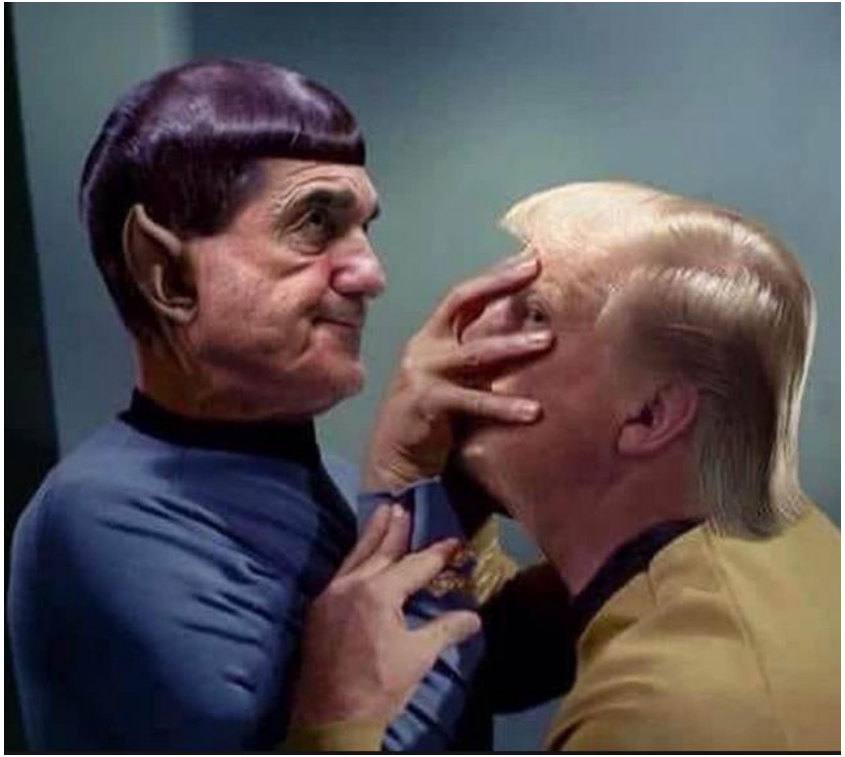
- Now that we've seen how Wireshark counts bytes in flight…
- …and the challenges of side by side comparisons…
- …let's look at how we can gain better insight from using visualization

First, a word about Merging

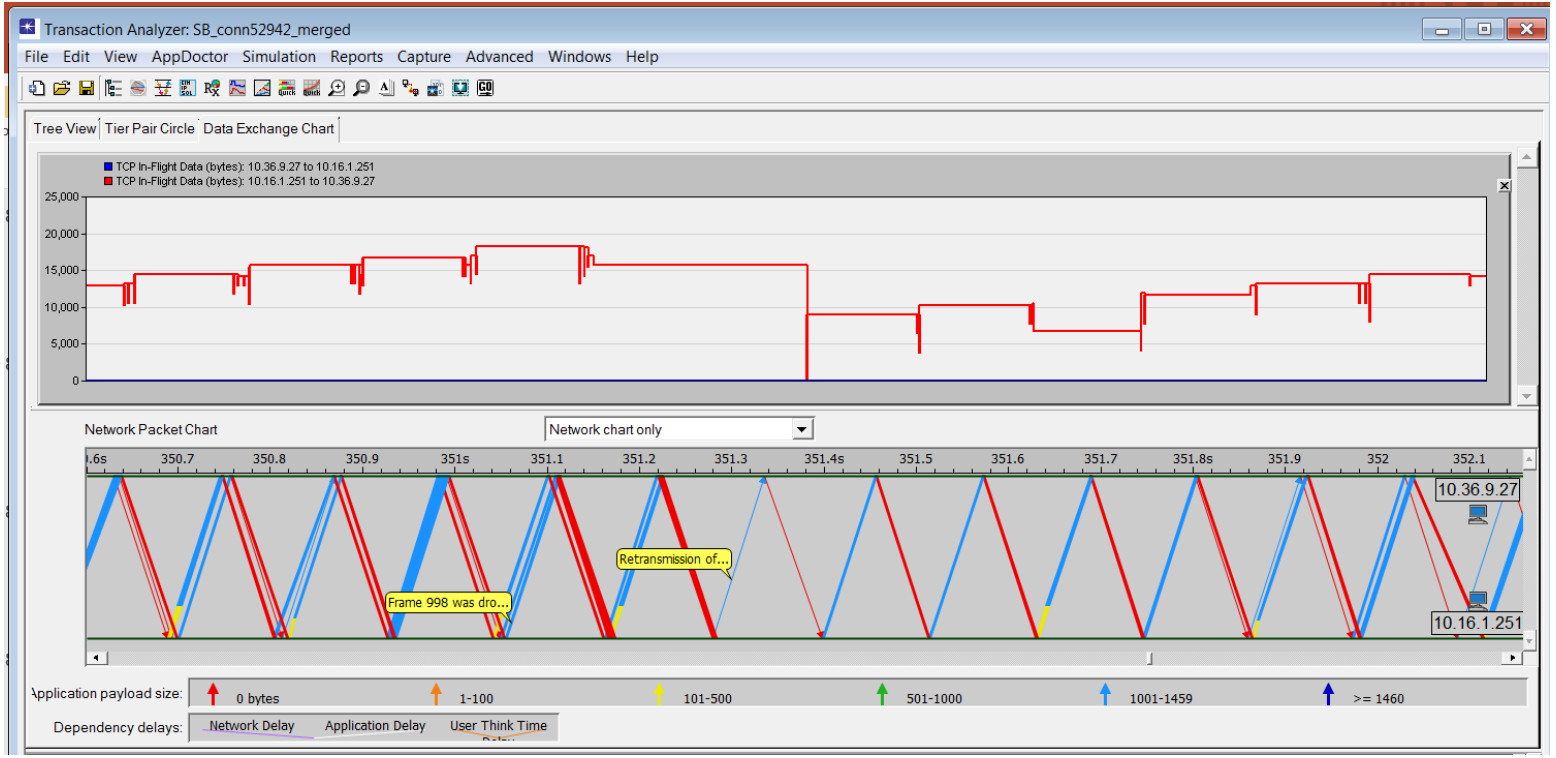
- What if we could merge captures?
- High Fidelity merge client and service side captures
- Fine tune “Send Time” and “Recv Time”
- Explicitly identify drops
- Accurately measure congestion
- Accurately measure Server Response Time
- Increases Accuracy of Advanced Analytics

Merge Melds Split Brain

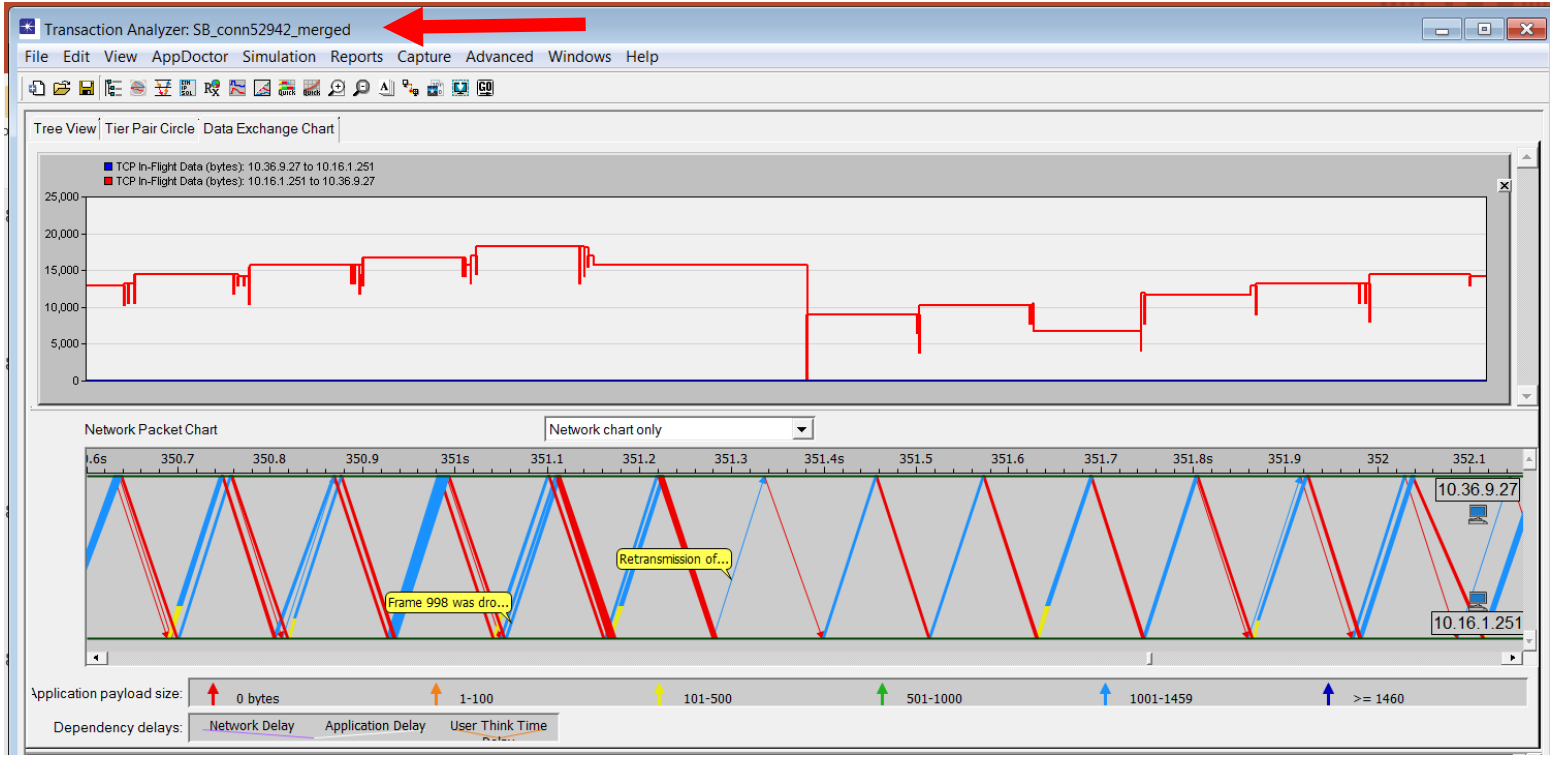
Circa 2019...



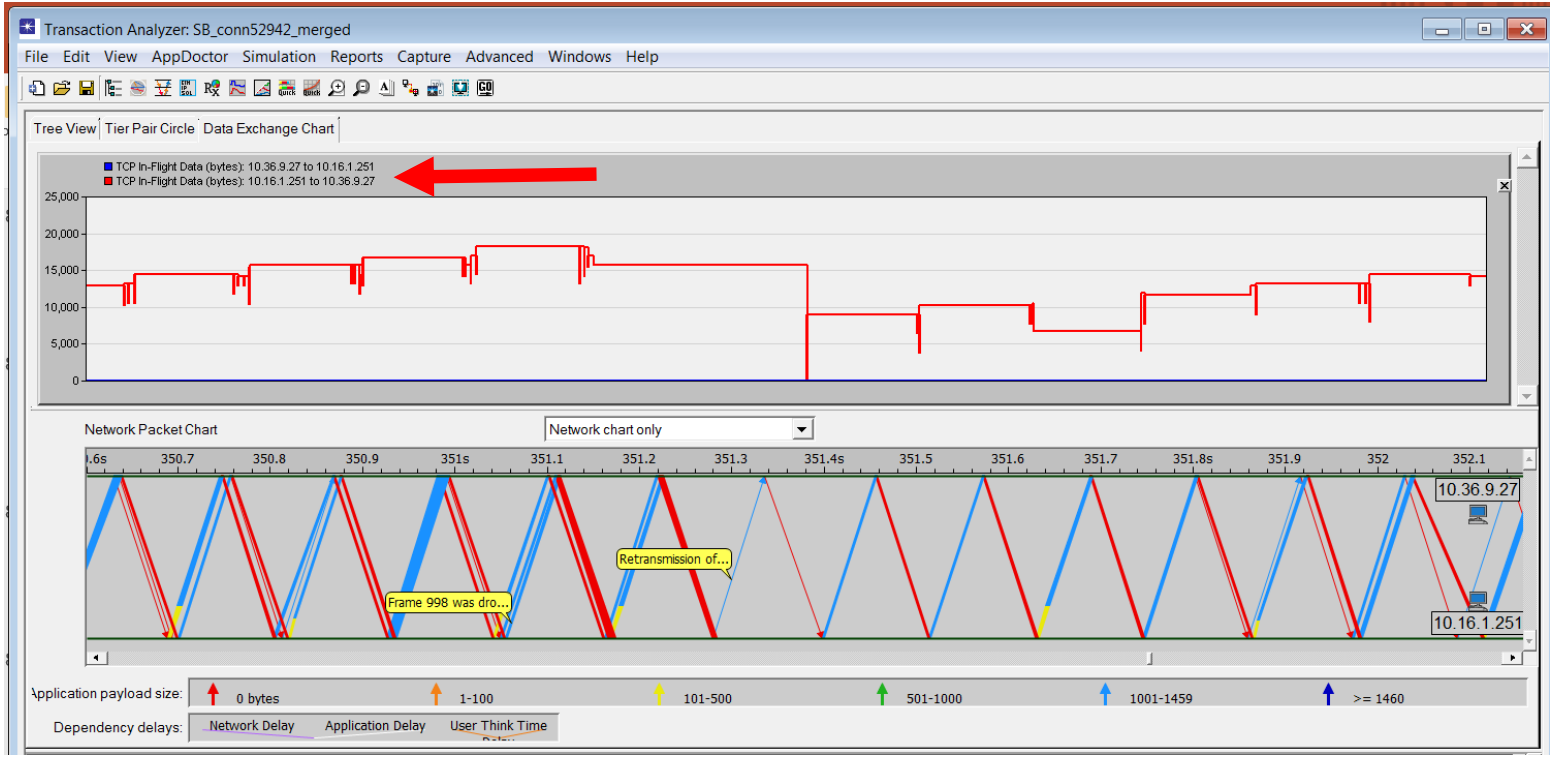
Merged Packet Exchange with BIF Overlay



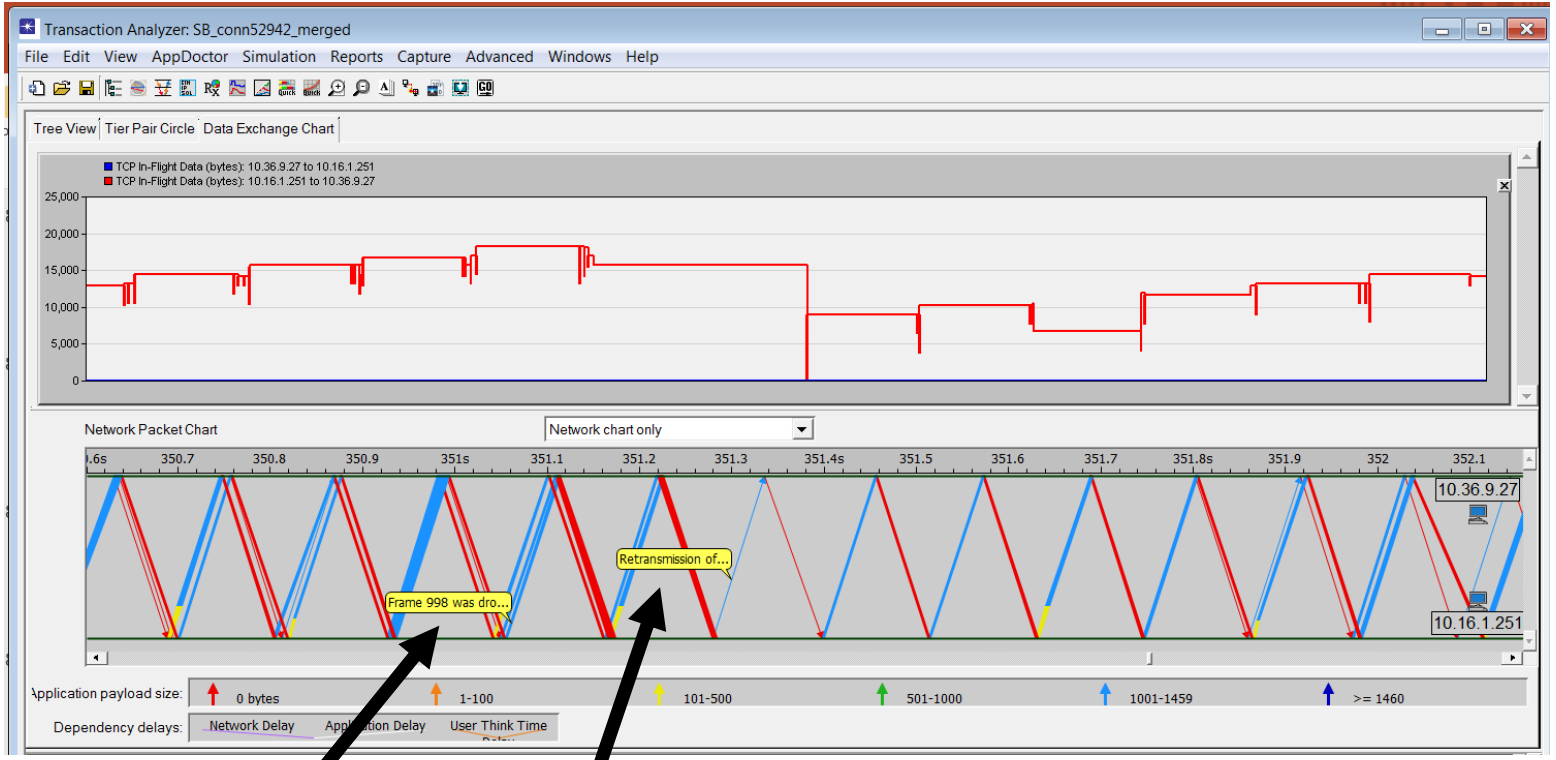
Quick Orientation



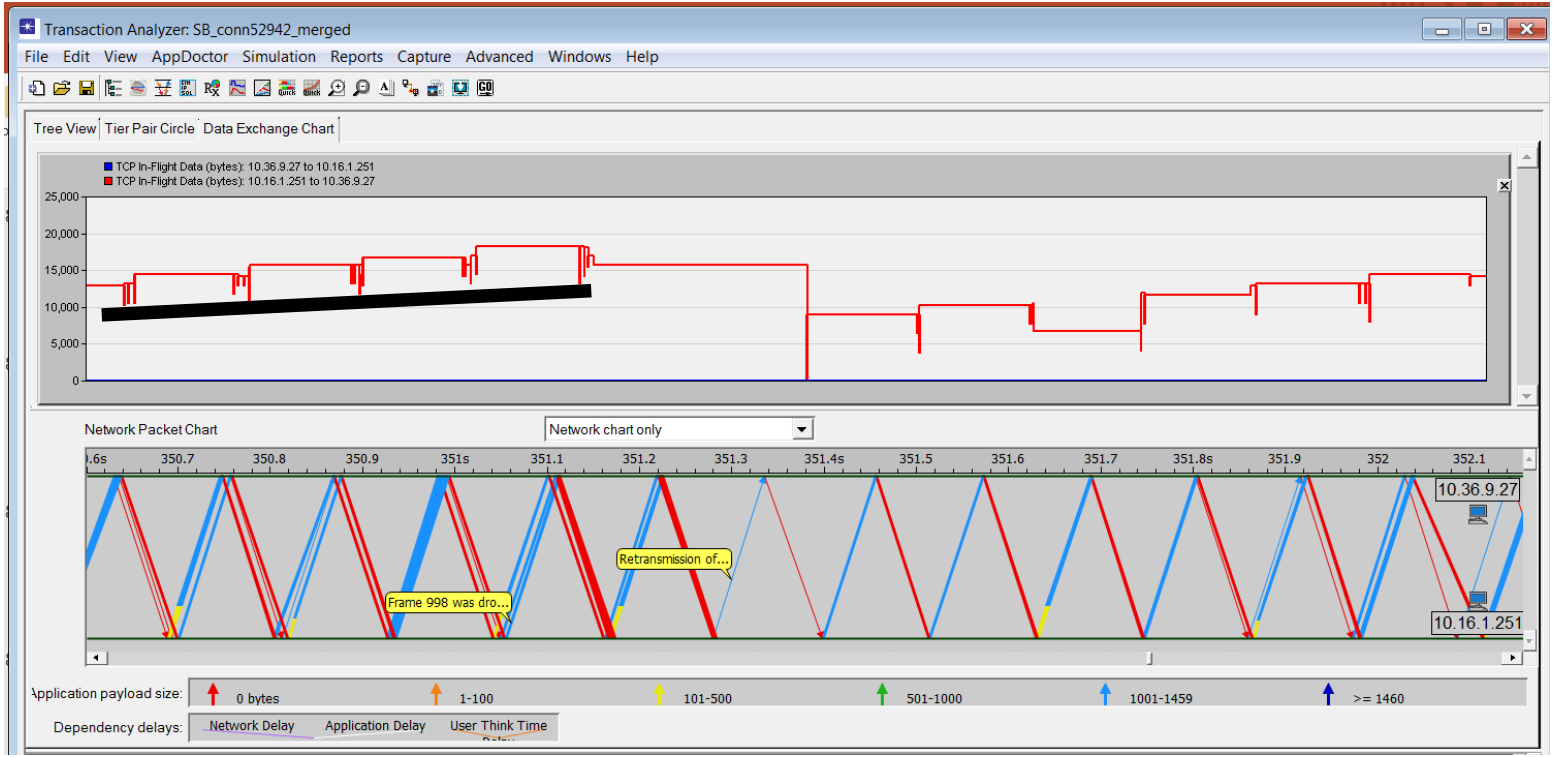
In-flight Bytes



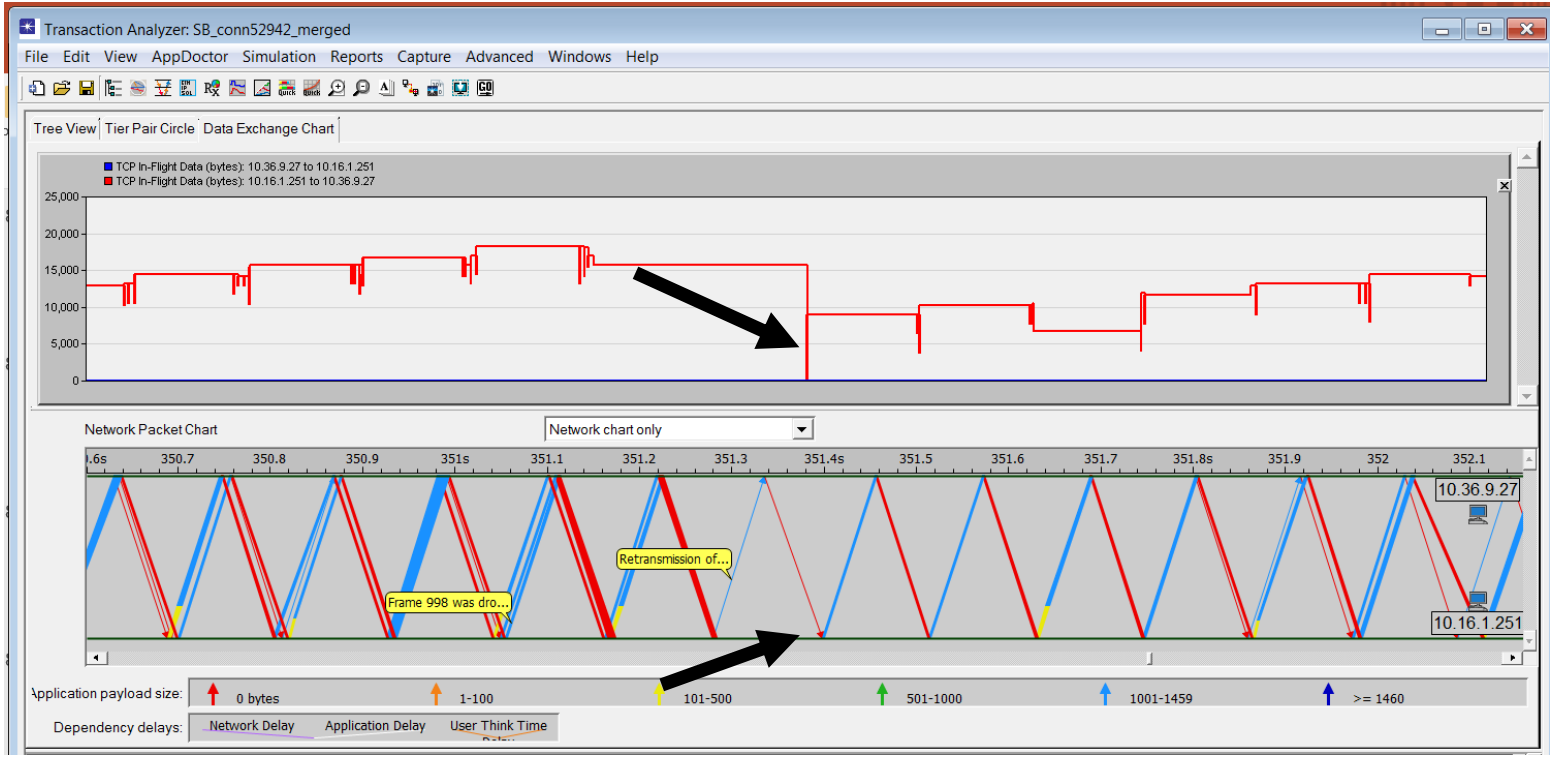
Symptom Pop-up Labels



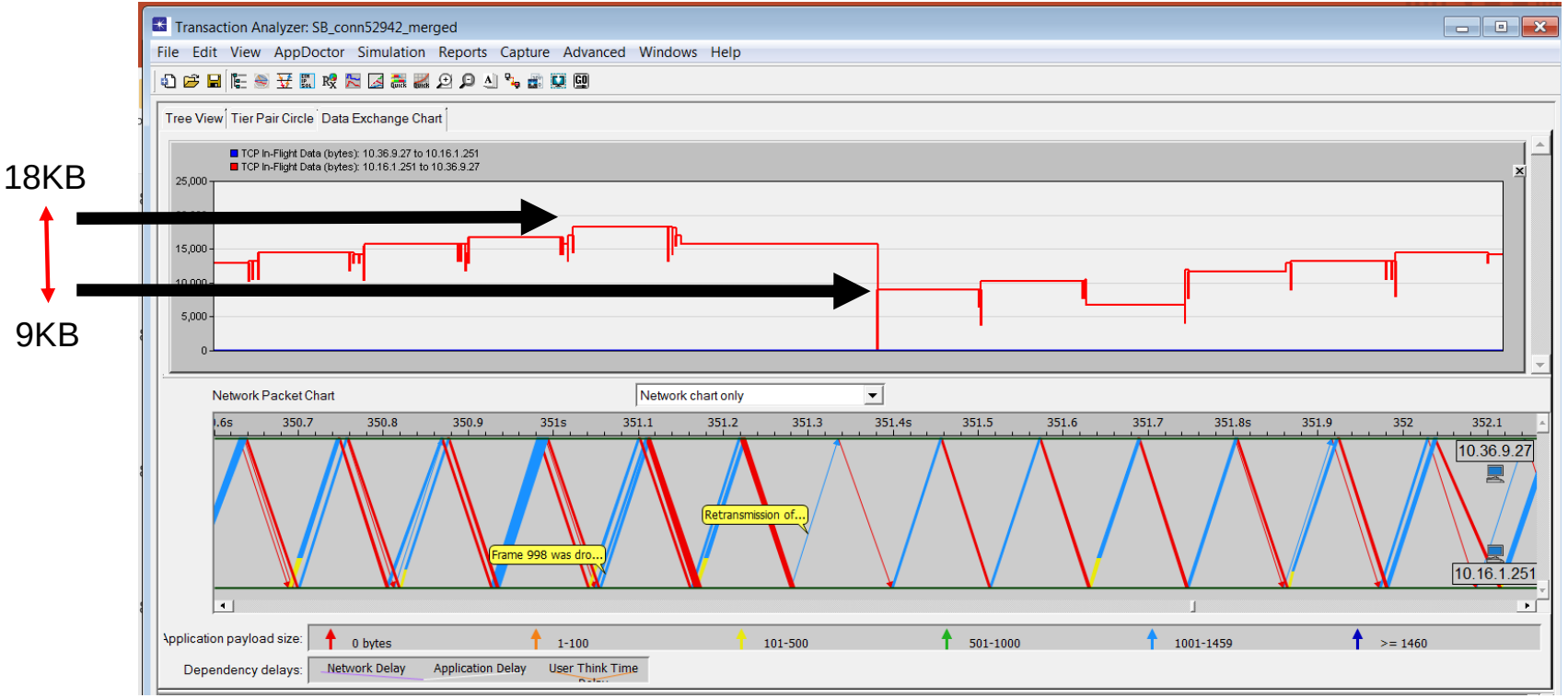
What does this upward progression tell us about CWND?



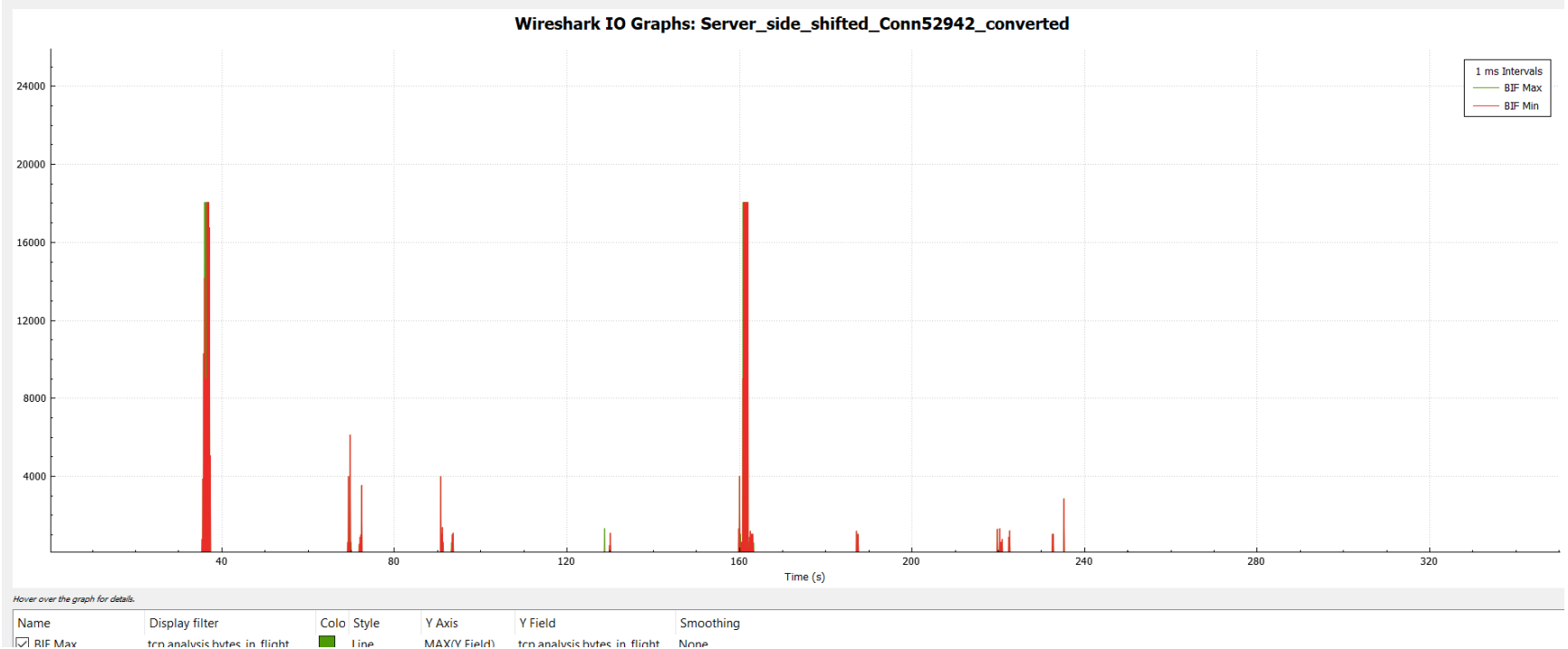
Why In-flight drops to 0?



What does this delta tell us?

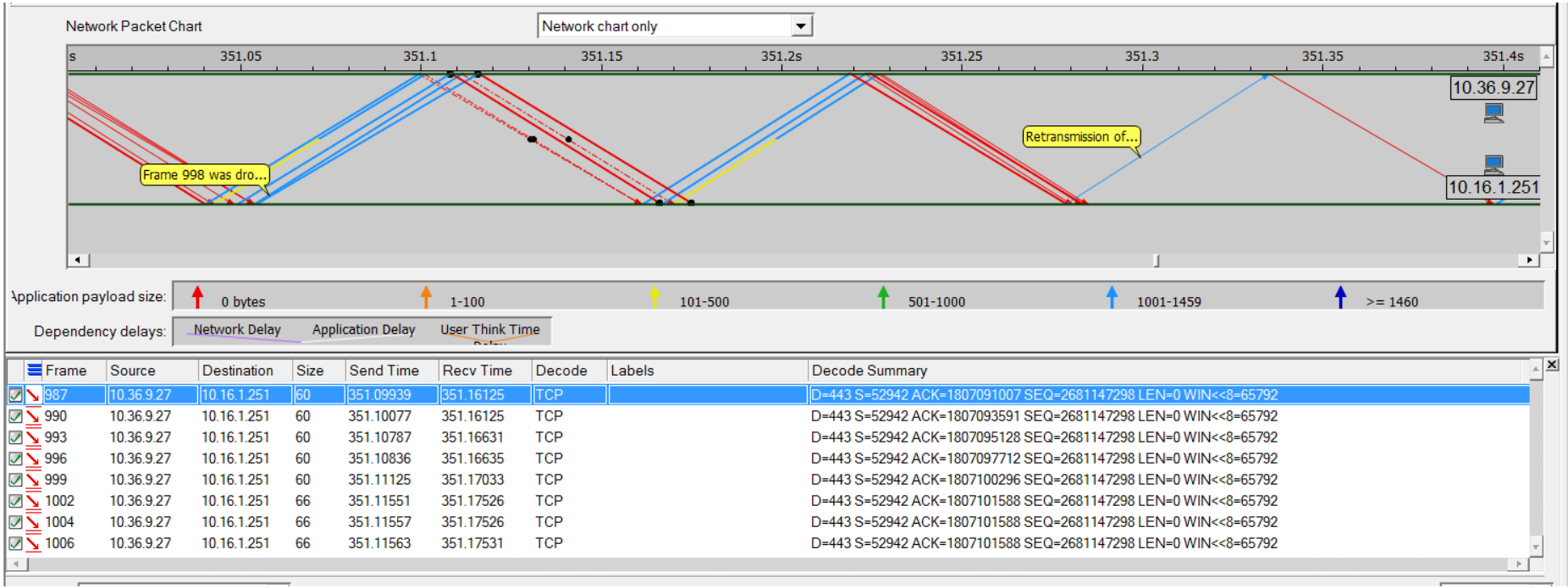


Wireshark BIF Chart



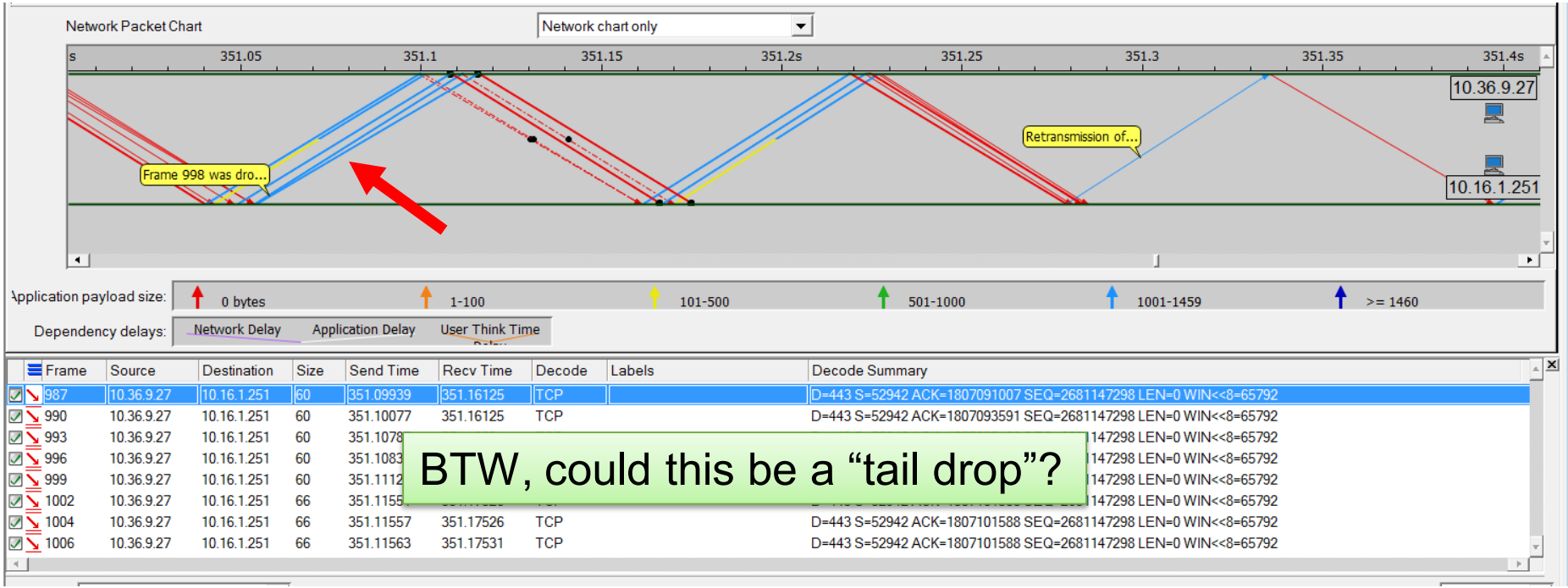
Looking closer @ timing...

When does Server hear the news?



Frame	Source	Destination	Size	Send Time	Recv Time	Decode	Labels	Decode Summary
987	10.36.9.27	10.16.1.251	60	351.09939	351.16125	TCP		D=443 S=52942 ACK=1807091007 SEQ=2681147298 LEN=0 WIN<<8=65792
990	10.36.9.27	10.16.1.251	60	351.10077	351.16125	TCP		D=443 S=52942 ACK=1807093591 SEQ=2681147298 LEN=0 WIN<<8=65792
993	10.36.9.27	10.16.1.251	60	351.10787	351.16631	TCP		D=443 S=52942 ACK=1807095128 SEQ=2681147298 LEN=0 WIN<<8=65792
996	10.36.9.27	10.16.1.251	60	351.10836	351.16635	TCP		D=443 S=52942 ACK=1807097712 SEQ=2681147298 LEN=0 WIN<<8=65792
999	10.36.9.27	10.16.1.251	60	351.11125	351.17033	TCP		D=443 S=52942 ACK=1807100296 SEQ=2681147298 LEN=0 WIN<<8=65792
1002	10.36.9.27	10.16.1.251	66	351.11551	351.17526	TCP		D=443 S=52942 ACK=1807101588 SEQ=2681147298 LEN=0 WIN<<8=65792
1004	10.36.9.27	10.16.1.251	66	351.11557	351.17526	TCP		D=443 S=52942 ACK=1807101588 SEQ=2681147298 LEN=0 WIN<<8=65792
1006	10.36.9.27	10.16.1.251	66	351.11563	351.17531	TCP		D=443 S=52942 ACK=1807101588 SEQ=2681147298 LEN=0 WIN<<8=65792

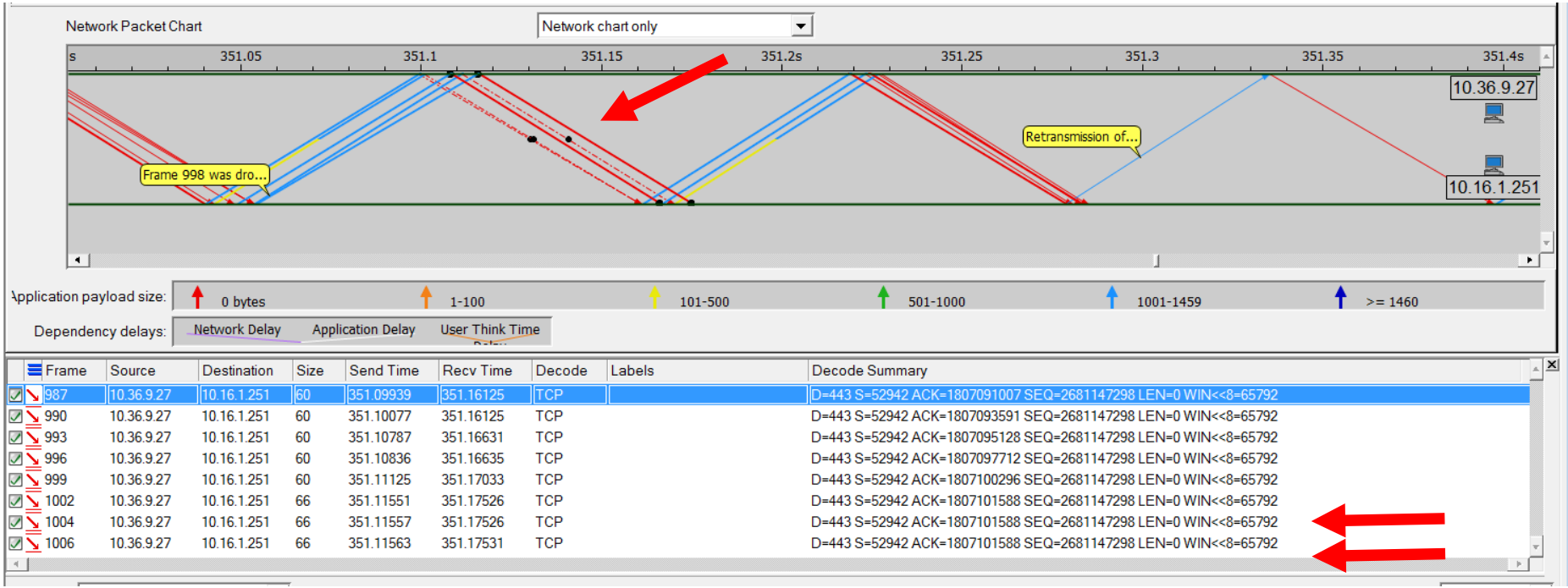
#998 is dropped



Frame	Source	Destination	Size	Send Time	Recv Time	Decode	Labels	Decode Summary
987	10.36.9.27	10.16.1.251	60	351.09939	351.16125	TCP		D=443 S=52942 ACK=1807091007 SEQ=2681147298 LEN=0 WIN<<8=65792
990	10.36.9.27	10.16.1.251	60	351.10077	351.16125	TCP		D=443 S=52942 ACK=1807093591 SEQ=2681147298 LEN=0 WIN<<8=65792
993	10.36.9.27	10.16.1.251	60	351.1078				147298 LEN=0 WIN<<8=65792
996	10.36.9.27	10.16.1.251	60	351.1083				147298 LEN=0 WIN<<8=65792
999	10.36.9.27	10.16.1.251	60	351.1112				147298 LEN=0 WIN<<8=65792
1002	10.36.9.27	10.16.1.251	66	351.1155				147298 LEN=0 WIN<<8=65792
1004	10.36.9.27	10.16.1.251	66	351.11557	351.17526	TCP		D=443 S=52942 ACK=1807101588 SEQ=2681147298 LEN=0 WIN<<8=65792
1006	10.36.9.27	10.16.1.251	66	351.11563	351.17531	TCP		D=443 S=52942 ACK=1807101588 SEQ=2681147298 LEN=0 WIN<<8=65792

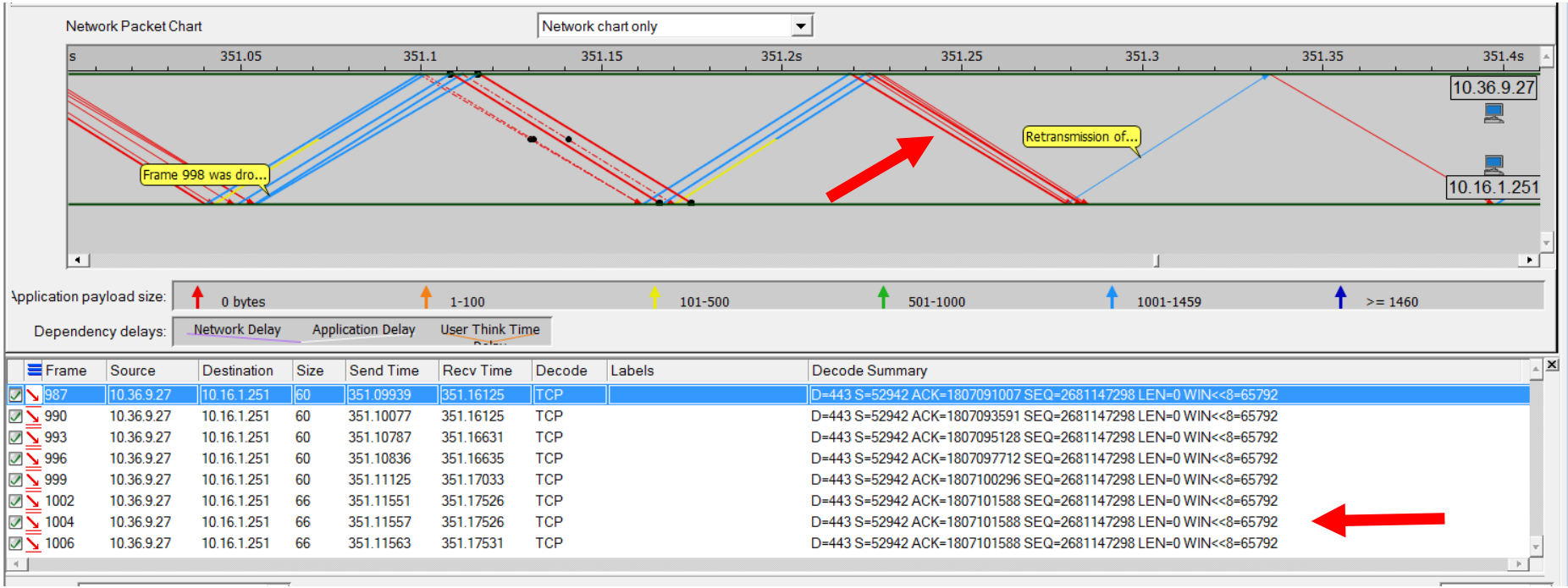
BTW, could this be a “tail drop”?

2 of the DupACKs



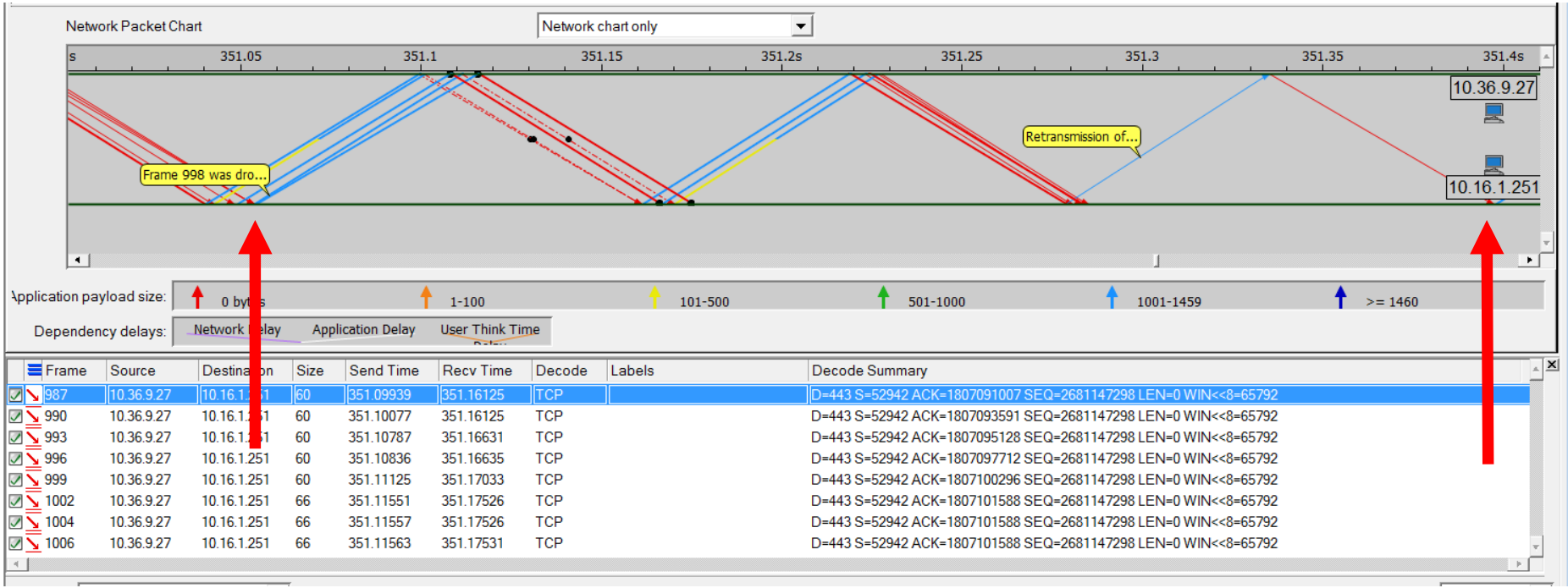
Frame	Source	Destination	Size	Send Time	Recv Time	Decode	Labels	Decode Summary
987	10.36.9.27	10.16.1.251	60	351.09939	351.16125	TCP		D=443 S=52942 ACK=1807091007 SEQ=2681147298 LEN=0 WIN<<8=65792
990	10.36.9.27	10.16.1.251	60	351.10077	351.16125	TCP		D=443 S=52942 ACK=1807093591 SEQ=2681147298 LEN=0 WIN<<8=65792
993	10.36.9.27	10.16.1.251	60	351.10787	351.16631	TCP		D=443 S=52942 ACK=1807095128 SEQ=2681147298 LEN=0 WIN<<8=65792
996	10.36.9.27	10.16.1.251	60	351.10836	351.16635	TCP		D=443 S=52942 ACK=1807097712 SEQ=2681147298 LEN=0 WIN<<8=65792
999	10.36.9.27	10.16.1.251	60	351.11125	351.17033	TCP		D=443 S=52942 ACK=1807100296 SEQ=2681147298 LEN=0 WIN<<8=65792
1002	10.36.9.27	10.16.1.251	66	351.11551	351.17526	TCP		D=443 S=52942 ACK=1807101588 SEQ=2681147298 LEN=0 WIN<<8=65792
1004	10.36.9.27	10.16.1.251	66	351.11557	351.17526	TCP		D=443 S=52942 ACK=1807101588 SEQ=2681147298 LEN=0 WIN<<8=65792
1006	10.36.9.27	10.16.1.251	66	351.11563	351.17531	TCP		D=443 S=52942 ACK=1807101588 SEQ=2681147298 LEN=0 WIN<<8=65792

3 of 3 DupACKs



Frame	Source	Destination	Size	Send Time	Recv Time	Decode	Labels	Decode Summary
987	10.36.9.27	10.16.1.251	60	351.09939	351.16125	TCP		D=443 S=52942 ACK=1807091007 SEQ=2681147298 LEN=0 WIN<<8=65792
990	10.36.9.27	10.16.1.251	60	351.10077	351.16125	TCP		D=443 S=52942 ACK=1807093591 SEQ=2681147298 LEN=0 WIN<<8=65792
993	10.36.9.27	10.16.1.251	60	351.10787	351.16631	TCP		D=443 S=52942 ACK=1807095128 SEQ=2681147298 LEN=0 WIN<<8=65792
996	10.36.9.27	10.16.1.251	60	351.10836	351.16635	TCP		D=443 S=52942 ACK=1807097712 SEQ=2681147298 LEN=0 WIN<<8=65792
999	10.36.9.27	10.16.1.251	60	351.11125	351.17033	TCP		D=443 S=52942 ACK=1807100296 SEQ=2681147298 LEN=0 WIN<<8=65792
1002	10.36.9.27	10.16.1.251	66	351.11551	351.17526	TCP		D=443 S=52942 ACK=1807101588 SEQ=2681147298 LEN=0 WIN<<8=65792
1004	10.36.9.27	10.16.1.251	66	351.11557	351.17526	TCP		D=443 S=52942 ACK=1807101588 SEQ=2681147298 LEN=0 WIN<<8=65792
1006	10.36.9.27	10.16.1.251	66	351.11563	351.17531	TCP		D=443 S=52942 ACK=1807101588 SEQ=2681147298 LEN=0 WIN<<8=65792

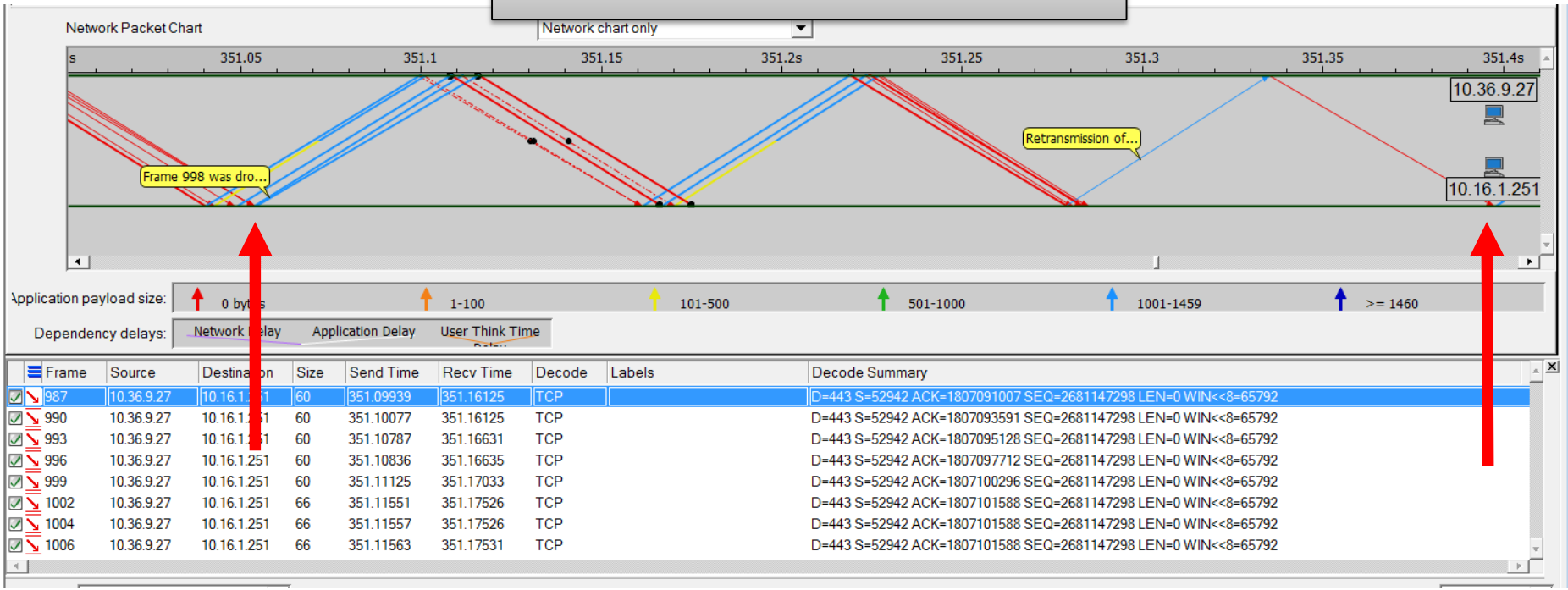
Delay Cost of Drop: # of RTTs?



Frame	Source	Destination	Size	Send Time	Recv Time	Decode	Labels	Decode Summary
987	10.36.9.27	10.16.1.251	60	351.09939	351.16125	TCP		D=443 S=52942 ACK=1807091007 SEQ=2681147298 LEN=0 WIN<<8=65792
990	10.36.9.27	10.16.1.251	60	351.10077	351.16125	TCP		D=443 S=52942 ACK=1807093591 SEQ=2681147298 LEN=0 WIN<<8=65792
993	10.36.9.27	10.16.1.251	60	351.10787	351.16631	TCP		D=443 S=52942 ACK=1807095128 SEQ=2681147298 LEN=0 WIN<<8=65792
996	10.36.9.27	10.16.1.251	60	351.10836	351.16635	TCP		D=443 S=52942 ACK=1807097712 SEQ=2681147298 LEN=0 WIN<<8=65792
999	10.36.9.27	10.16.1.251	60	351.11125	351.17033	TCP		D=443 S=52942 ACK=1807100296 SEQ=2681147298 LEN=0 WIN<<8=65792
1002	10.36.9.27	10.16.1.251	66	351.11551	351.17526	TCP		D=443 S=52942 ACK=1807101588 SEQ=2681147298 LEN=0 WIN<<8=65792
1004	10.36.9.27	10.16.1.251	66	351.11557	351.17526	TCP		D=443 S=52942 ACK=1807101588 SEQ=2681147298 LEN=0 WIN<<8=65792
1006	10.36.9.27	10.16.1.251	66	351.11563	351.17531	TCP		D=443 S=52942 ACK=1807101588 SEQ=2681147298 LEN=0 WIN<<8=65792

Delay Cost of Drop: # of RTTs?

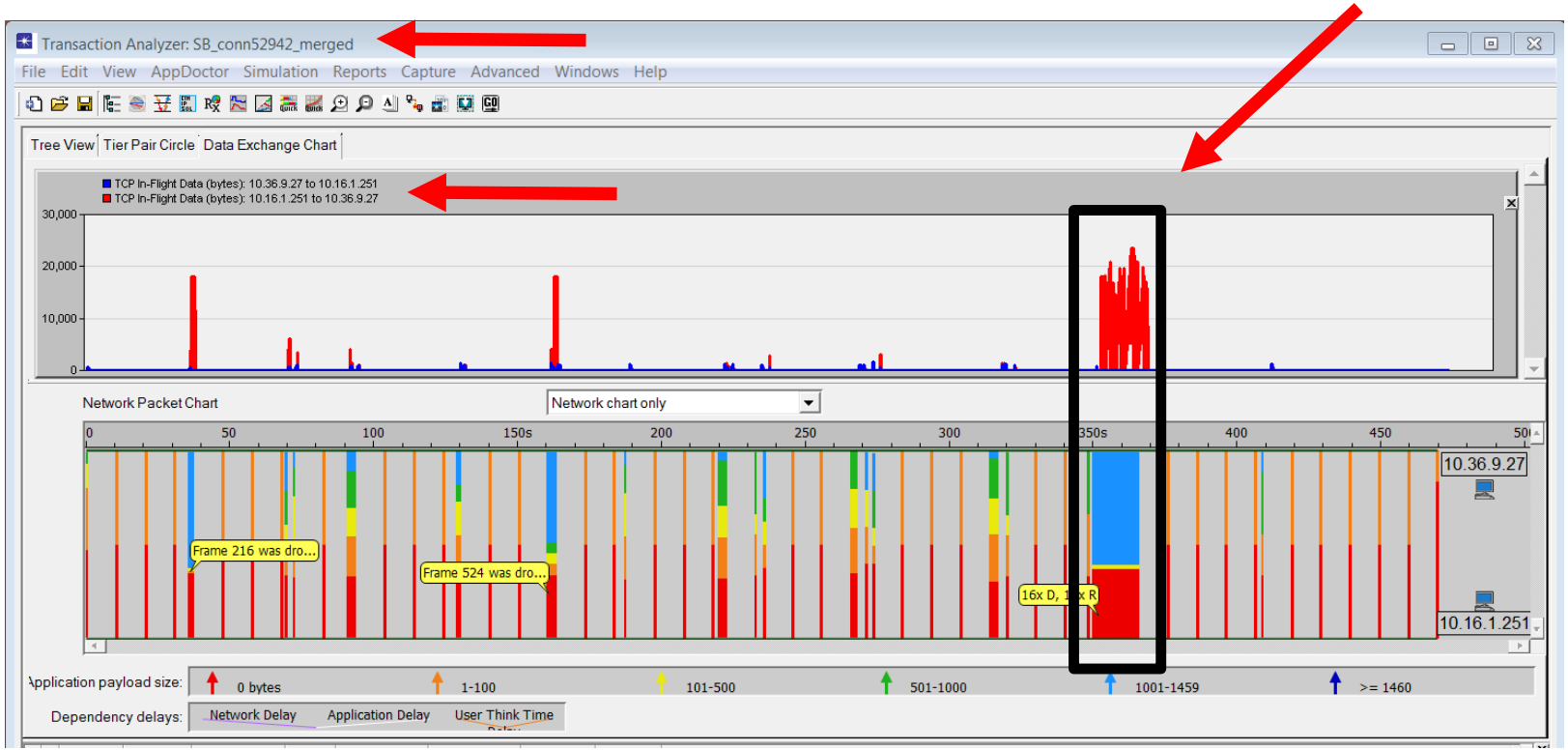
3 RTT x iRTT == 387 ms



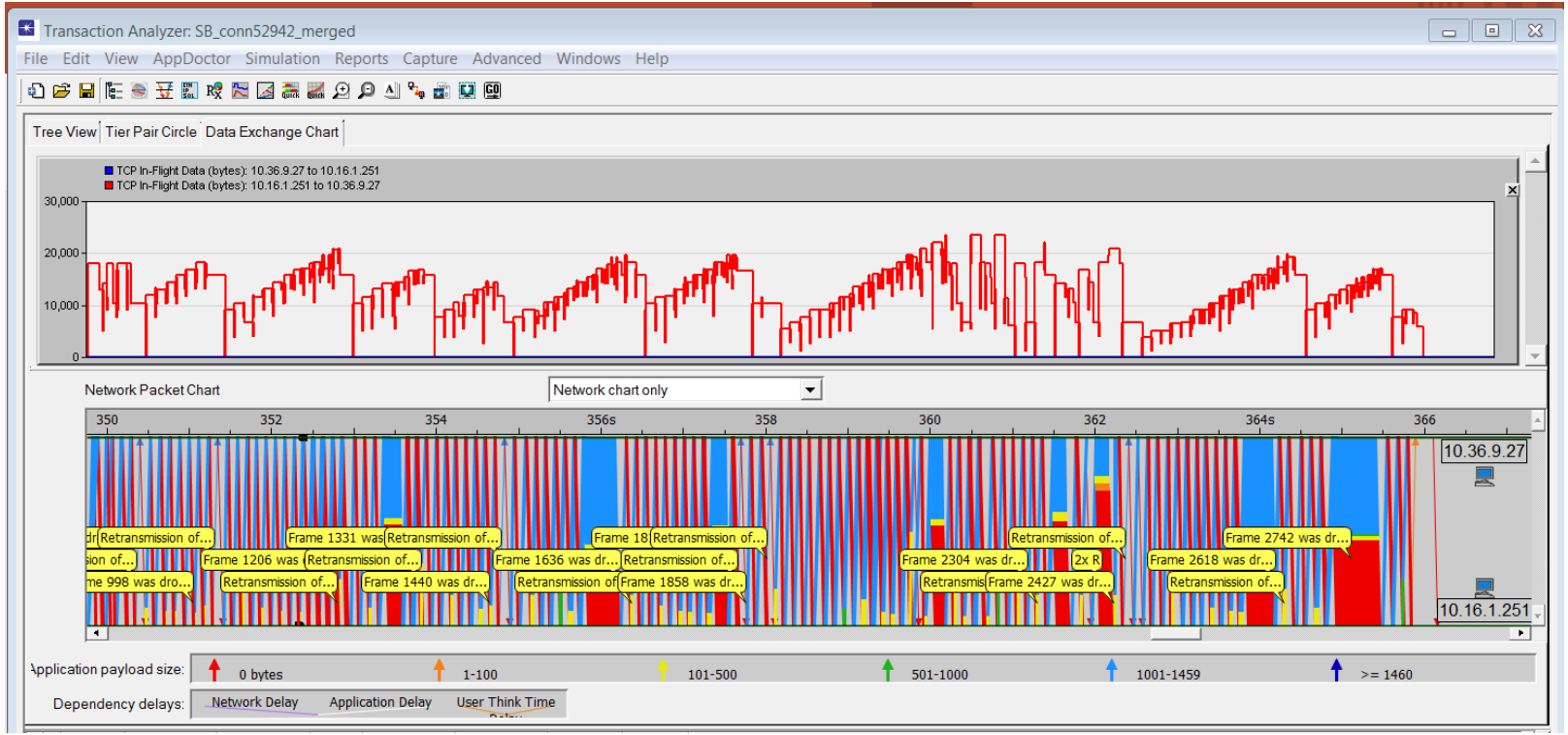
Frame	Source	Destination	Size	Send Time	Recv Time	Decode	Labels	Decode Summary
987	10.36.9.27	10.16.1.251	60	351.09939	351.16125	TCP		D=443 S=52942 ACK=1807091007 SEQ=2681147298 LEN=0 WIN<<8=65792
990	10.36.9.27	10.16.1.251	60	351.10077	351.16125	TCP		D=443 S=52942 ACK=1807093591 SEQ=2681147298 LEN=0 WIN<<8=65792
993	10.36.9.27	10.16.1.251	60	351.10787	351.16631	TCP		D=443 S=52942 ACK=1807095128 SEQ=2681147298 LEN=0 WIN<<8=65792
996	10.36.9.27	10.16.1.251	60	351.10836	351.16635	TCP		D=443 S=52942 ACK=1807097712 SEQ=2681147298 LEN=0 WIN<<8=65792
999	10.36.9.27	10.16.1.251	60	351.11125	351.17033	TCP		D=443 S=52942 ACK=1807100296 SEQ=2681147298 LEN=0 WIN<<8=65792
1002	10.36.9.27	10.16.1.251	66	351.11551	351.17526	TCP		D=443 S=52942 ACK=1807101588 SEQ=2681147298 LEN=0 WIN<<8=65792
1004	10.36.9.27	10.16.1.251	66	351.11557	351.17526	TCP		D=443 S=52942 ACK=1807101588 SEQ=2681147298 LEN=0 WIN<<8=65792
1006	10.36.9.27	10.16.1.251	66	351.11563	351.17531	TCP		D=443 S=52942 ACK=1807101588 SEQ=2681147298 LEN=0 WIN<<8=65792

Discussion

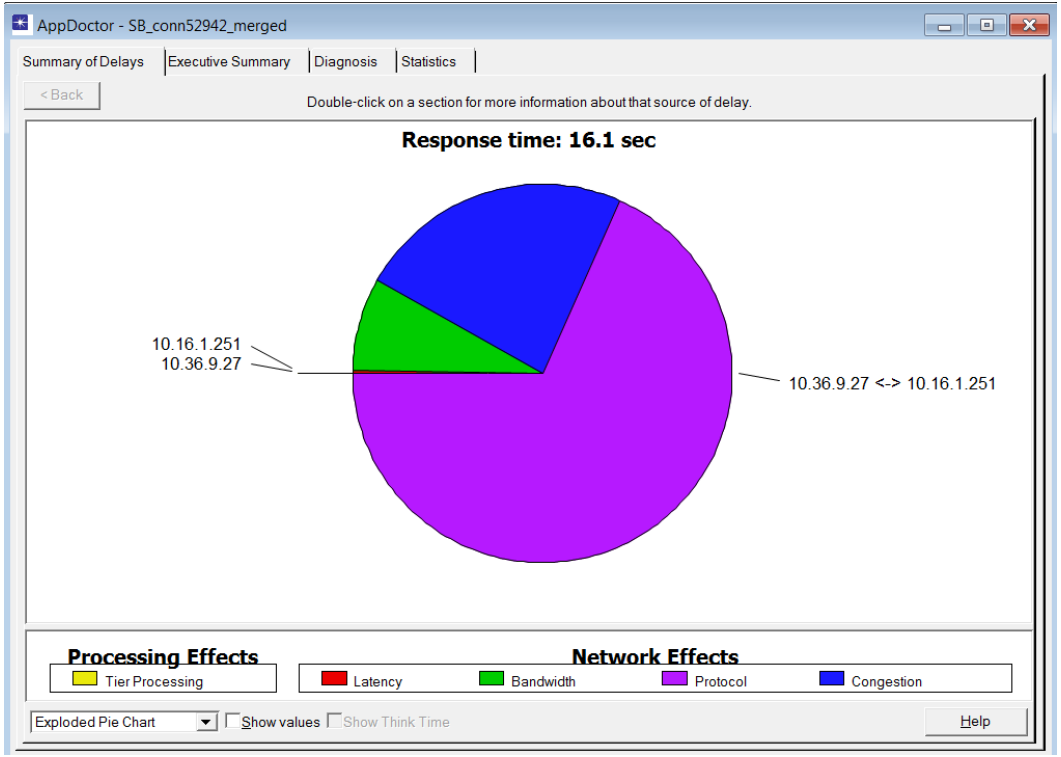
Let's Visualize BIF for the Entire TCP Connection



...drill into the 1.4MB Xfer



Delay Analysis during Xfer



Review

- BIF for receiver side traffic doesn’t tell you much about actual congestion window
- Segmentation offload can create misleading conclusions on sender capture
- Packet loss on high latency path can have severe impact on performance

Comparison Summary

TCP Split Brain Comparison Summary (Sender vs. Receiver)				
Item	Topic	Summary	Sender	Receiver
7	Bytes In-Flight	OOS and retransmissions may impact calculation	Should be pretty accurate	Generally not very interesting from receiver's capture

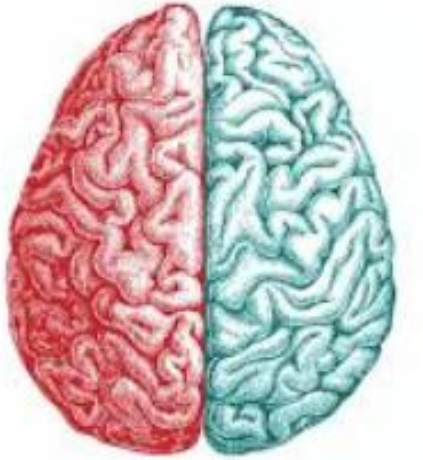
6th Leg Completed

Bonus Section

- Time Permitting

Split Brain Comparisons

- Congestion



Remember this from Part I

- Client Capture – 121ms

GP_VPN_Client_Conn52942.pcap

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-/>

No.	Time	Time delta from previous c	Time delta from previous	Source	Destination	Identification	Lengtl	Sequence numbe	Acknowledgmer	Bytes in flight	Info
1	09:11:42.223468	0.000000000	0.000000000	10.36.9.27	10.16.1.251	0x0085 (1…	66	0	0		52942 → 443 [SYN] S
2	09:11:42.344959	0.121491000	0.121491000	10.16.1.251	10.36.9.27	0x77bc (3…	66	0	1		443 → 52942 [SYN, A
3	09:11:42.345046	0.000087000	0.000087000	10.36.9.27	10.16.1.251	0x0091 (1…	60	1	1		52942 → 443 [ACK] S

- Server Capture – 129ms

Server_side_shifted_Conn52942.pcap

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-/>

No.	Time	Time delta from previ	Time delta from previo	Source	Destination	Identification	Lengtl	Sequence numbe	Acknowledgmer	Bytes in flight	Info
1	09:11:42.262470	0.000000…	0.0000000…	10.36.9.27	10.16.1.2…	0x0085 (1…	82	0	0		S+, 52942 → 443 [SYN]
2	09:11:42.262524	0.000054…	0.0000540…	10.16.1.2…	10.36.9.27	0x77bc (3…	66	0	1		443 → 52942 [SYN, ACK]
3	09:11:42.391722	0.129198…	0.1291980…	10.36.9.27	10.16.1.2…	0x0091 (1…	60	1	1		52942 → 443 [ACK] Seq=

Group Discussion

- What's the definition of latency?
- What's the definition of congestion?
- How can you tell latency from congestion?

Congestion is not...

- ...protocol delay
- ...serialization (bandwidth delay)
- ...server delay (well maybe...)
- ...others?

Introducing…1W-TTT

- One Way - Total Transfer Time

1-Way Total Transfer Time

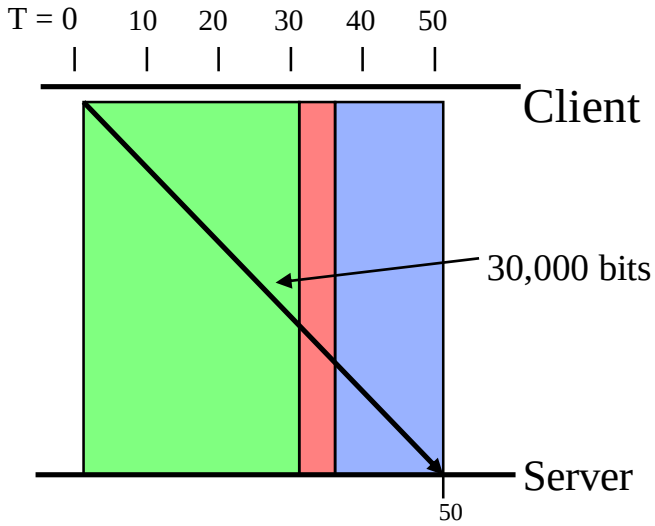
- Probably other names for this…
- Time required for bits to leave sender and arrive at receiver…
- TTT == Total Transfer Time

One Possible Formula

- 1-Way Total Transfer time (TTT) .....minus
- (Bandwidth Delay + Latency Delay)
- TTT – (BD + LD) == Congestion

Exaggerated TTT Example

- Bandwidth = 1,000 bps
- Latency = 5 seconds



Bandwidth delay

30 seconds

Latency delay

5 seconds

Congestion delay

15 seconds

RTT vs. 1-Way TTT

- RTT is not the same as 1-Way TTT
- What can Wireshark tell us about 1-Way TTT?
- Answer: not a lot…but we can infer a few things

WireShark RTT include PD

- 1-Way Total Transfer time (TTT) ……..minus
- (Bandwidth Delay + Latency Delay + Protocol Delay)
- TTT – (BD + LD + PD) == Congestion

Finding congestion with Wireshark

- How can we find examples of congestion in a capture with Wireshark?
- For TCP we can use RTT2ACK, and infer congestion based on our knowledge about how Wireshark calculates RTT2ACK

It’s not Perfect…

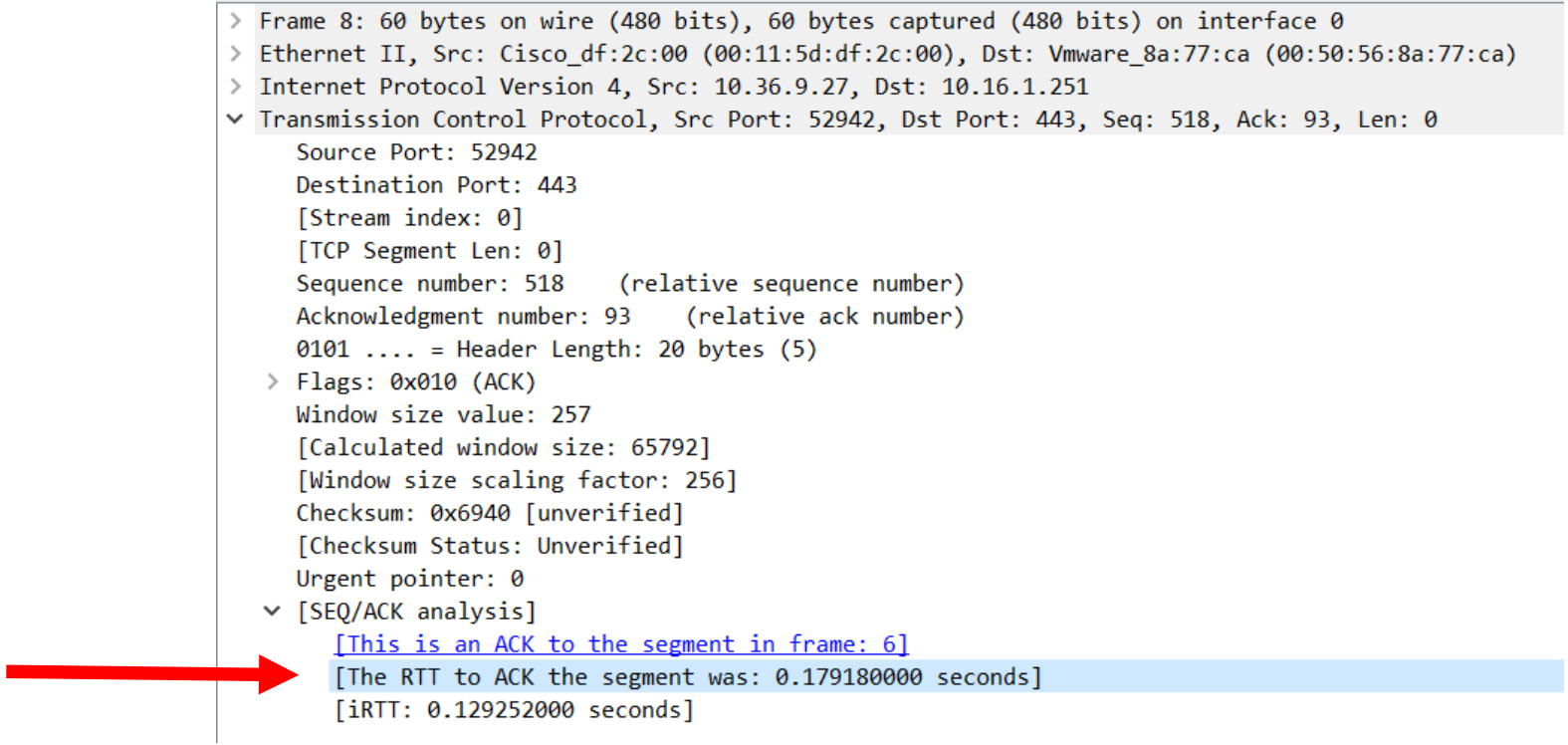
- …but you’ll get a decent approximation
- …however, be aware…
- … it could be overstated due to protocol delay

Sender or Receiver Capture

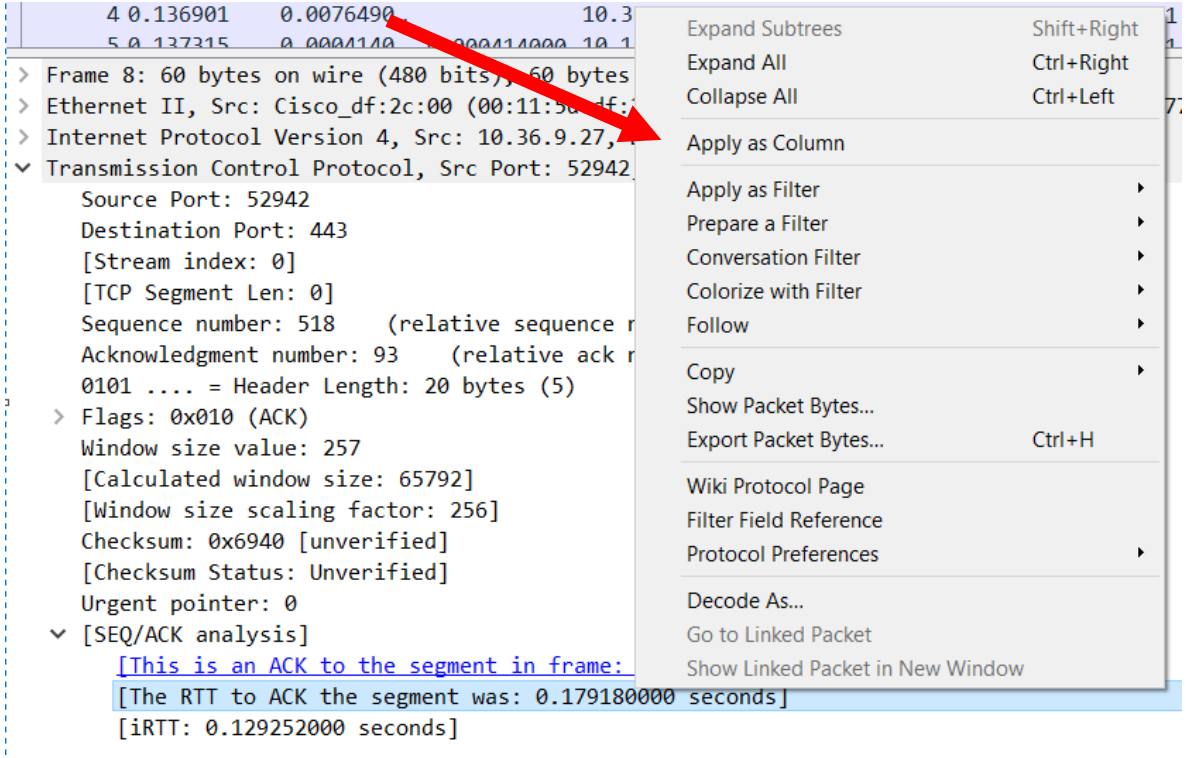
- Which capture would be most interesting to understand congestion?

- Answer: each one is unique and will tell part of the story (split brain)

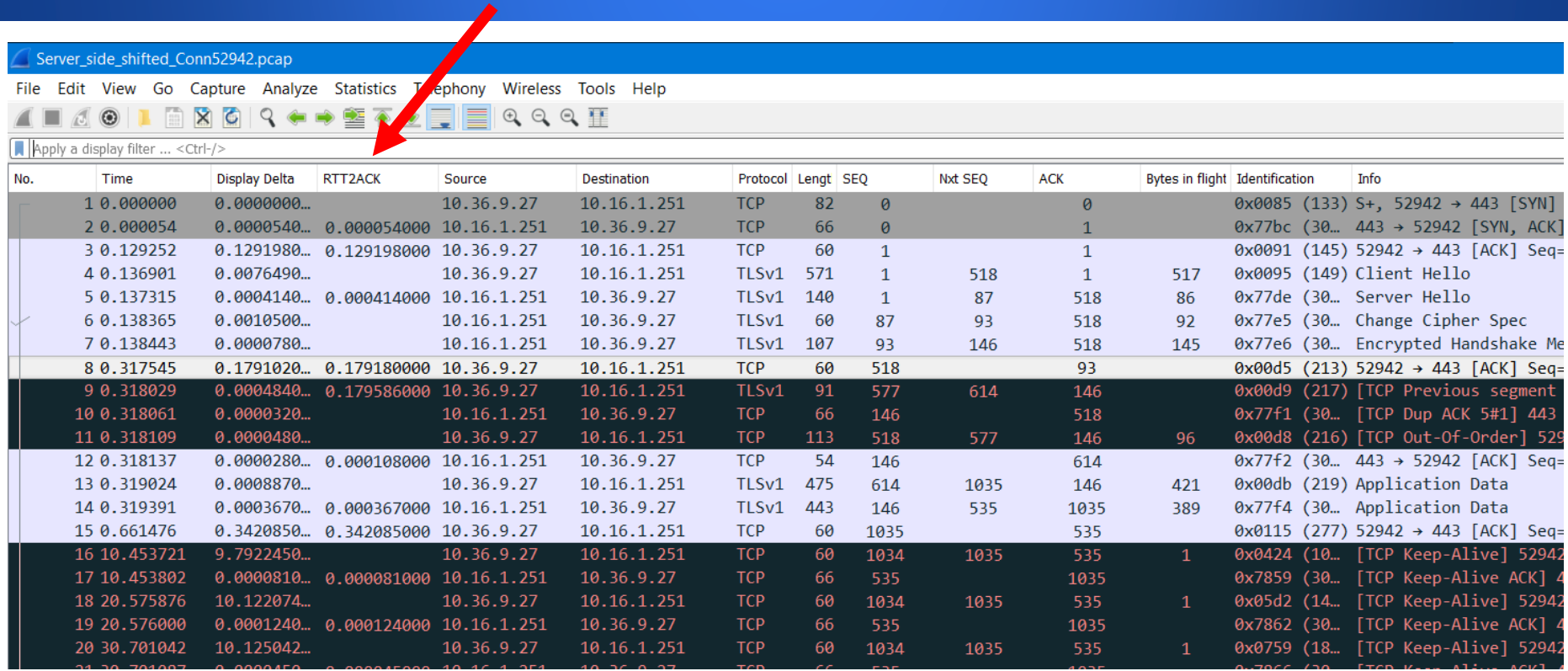
RTT2ACK Decode



Apply as Column

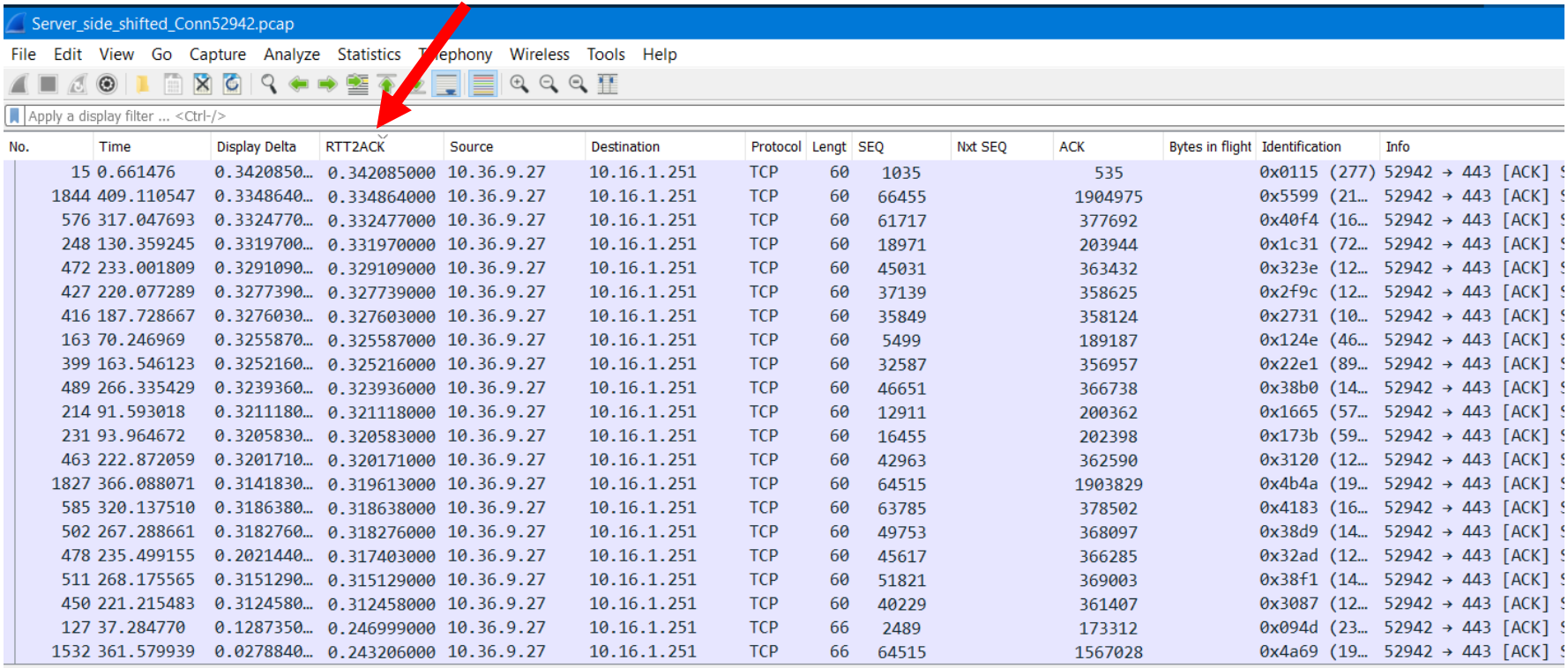


Column added to our view...



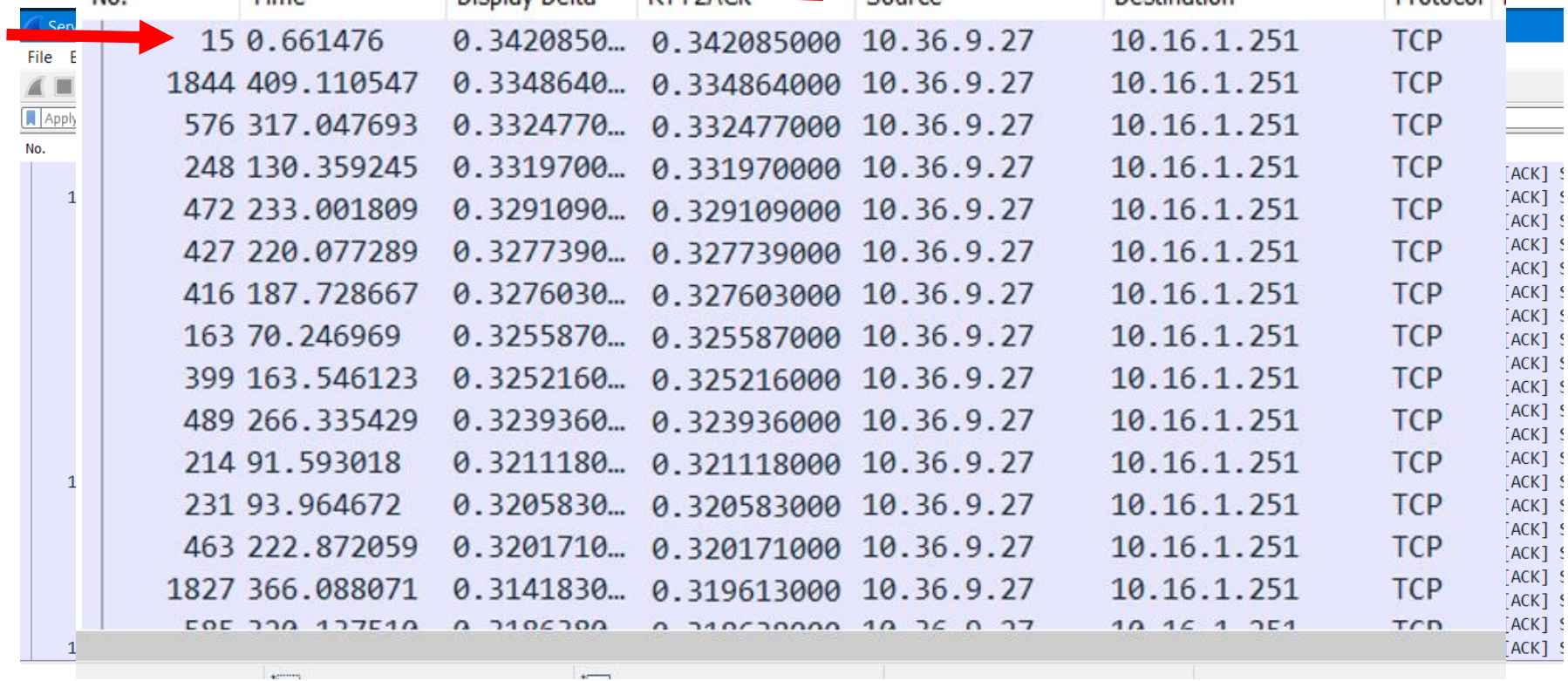
No.	Time	Display Delta	RTT2ACK	Source	Destination	Protocol	Lengt	SEQ	Nxt SEQ	ACK	Bytes in flight	Identification	Info
1	0.000000	0.0000000…		10.36.9.27	10.16.1.251	TCP	82	0		0		0x0085 (133)	S+, 52942 → 443 [SYN]
2	0.000054	0.0000540…	0.000054000	10.16.1.251	10.36.9.27	TCP	66	0		1		0x77bc (30…	443 → 52942 [SYN, ACK]
3	0.129252	0.1291980…	0.129198000	10.36.9.27	10.16.1.251	TCP	60	1		1		0x0091 (145)	52942 → 443 [ACK] Seq=
4	0.136901	0.0076490…		10.36.9.27	10.16.1.251	TLSv1	571	1	518	1	517	0x0095 (149)	Client Hello
5	0.137315	0.0004140…	0.000414000	10.16.1.251	10.36.9.27	TLSv1	140	1	87	518	86	0x77de (30…	Server Hello
6	0.138365	0.0010500…		10.16.1.251	10.36.9.27	TLSv1	60	87	93	518	92	0x77e5 (30…	Change Cipher Spec
7	0.138443	0.0000780…		10.16.1.251	10.36.9.27	TLSv1	107	93	146	518	145	0x77e6 (30…	Encrypted Handshake Me
8	0.317545	0.1791020…	0.179180000	10.36.9.27	10.16.1.251	TCP	60	518		93		0x00d5 (213)	52942 → 443 [ACK] Seq=
9	0.318029	0.0004840…	0.179586000	10.36.9.27	10.16.1.251	TLSv1	91	577	614	146		0x00d9 (217)	[TCP Previous segment
10	0.318061	0.0000320…		10.16.1.251	10.36.9.27	TCP	66	146		518		0x77f1 (30…	[TCP Dup ACK 5#1] 443
11	0.318109	0.0000480…		10.36.9.27	10.16.1.251	TCP	113	518	577	146	96	0x00d8 (216)	[TCP Out-Of-Order] 529
12	0.318137	0.0000280…	0.000108000	10.16.1.251	10.36.9.27	TCP	54	146		614		0x77f2 (30…	443 → 52942 [ACK] Seq=
13	0.319024	0.0008870…		10.36.9.27	10.16.1.251	TLSv1	475	614	1035	146	421	0x00db (219)	Application Data
14	0.319391	0.0003670…	0.000367000	10.16.1.251	10.36.9.27	TLSv1	443	146	535	1035	389	0x77f4 (30…	Application Data
15	0.661476	0.3420850…	0.342085000	10.36.9.27	10.16.1.251	TCP	60	1035		535		0x0115 (277)	52942 → 443 [ACK] Seq=
16	10.453721	9.7922450…		10.36.9.27	10.16.1.251	TCP	60	1034	1035	535	1	0x0424 (10…	[TCP Keep-Alive] 52942
17	10.453802	0.0000810…	0.000081000	10.16.1.251	10.36.9.27	TCP	66	535		1035		0x7859 (30…	[TCP Keep-Alive ACK] 4
18	20.575876	10.122074…		10.36.9.27	10.16.1.251	TCP	60	1034	1035	535	1	0x05d2 (14…	[TCP Keep-Alive] 52942
19	20.576000	0.0001240…	0.000124000	10.16.1.251	10.36.9.27	TCP	66	535		1035		0x7862 (30…	[TCP Keep-Alive ACK] 4
20	30.701042	10.125042…		10.36.9.27	10.16.1.251	TCP	60	1034	1035	535	1	0x0759 (18…	[TCP Keep-Alive] 52942

Sort by RTT2ACK



No.	Time	Display Delta	RTT2ACK	Source	Destination	Protocol	Lengt	SEQ	Nxt SEQ	ACK	Bytes in flight	Identification	Info
15	0.661476	0.3420850…	0.342085000	10.36.9.27	10.16.1.251	TCP	60	1035		535		0x0115 (277)	52942 → 443 [ACK] S
1844	409.110547	0.3348640…	0.334864000	10.36.9.27	10.16.1.251	TCP	60	66455		1904975		0x5599 (21…	52942 → 443 [ACK] S
576	317.047693	0.3324770…	0.332477000	10.36.9.27	10.16.1.251	TCP	60	61717		377692		0x40f4 (16…	52942 → 443 [ACK] S
248	130.359245	0.3319700…	0.331970000	10.36.9.27	10.16.1.251	TCP	60	18971		203944		0x1c31 (72…	52942 → 443 [ACK] S
472	233.001809	0.3291090…	0.329109000	10.36.9.27	10.16.1.251	TCP	60	45031		363432		0x323e (12…	52942 → 443 [ACK] S
427	220.077289	0.3277390…	0.327739000	10.36.9.27	10.16.1.251	TCP	60	37139		358625		0x2f9c (12…	52942 → 443 [ACK] S
416	187.728667	0.3276030…	0.327603000	10.36.9.27	10.16.1.251	TCP	60	35849		358124		0x2731 (10…	52942 → 443 [ACK] S
163	70.246969	0.3255870…	0.325587000	10.36.9.27	10.16.1.251	TCP	60	5499		189187		0x124e (46…	52942 → 443 [ACK] S
399	163.546123	0.3252160…	0.325216000	10.36.9.27	10.16.1.251	TCP	60	32587		356957		0x22e1 (89…	52942 → 443 [ACK] S
489	266.335429	0.3239360…	0.323936000	10.36.9.27	10.16.1.251	TCP	60	46651		366738		0x38b0 (14…	52942 → 443 [ACK] S
214	91.593018	0.3211180…	0.321118000	10.36.9.27	10.16.1.251	TCP	60	12911		200362		0x1665 (57…	52942 → 443 [ACK] S
231	93.964672	0.3205830…	0.320583000	10.36.9.27	10.16.1.251	TCP	60	16455		202398		0x173b (59…	52942 → 443 [ACK] S
463	222.872059	0.3201710…	0.320171000	10.36.9.27	10.16.1.251	TCP	60	42963		362590		0x3120 (12…	52942 → 443 [ACK] S
1827	366.088071	0.3141830…	0.319613000	10.36.9.27	10.16.1.251	TCP	60	64515		1903829		0x4b4a (19…	52942 → 443 [ACK] S
585	320.137510	0.3186380…	0.318638000	10.36.9.27	10.16.1.251	TCP	60	63785		378502		0x4183 (16…	52942 → 443 [ACK] S
502	267.288661	0.3182760…	0.318276000	10.36.9.27	10.16.1.251	TCP	60	49753		368097		0x38d9 (14…	52942 → 443 [ACK] S
478	235.499155	0.2021440…	0.317403000	10.36.9.27	10.16.1.251	TCP	60	45617		366285		0x32ad (12…	52942 → 443 [ACK] S
511	268.175565	0.3151290…	0.315129000	10.36.9.27	10.16.1.251	TCP	60	51821		369003		0x38f1 (14…	52942 → 443 [ACK] S
450	221.215483	0.3124580…	0.312458000	10.36.9.27	10.16.1.251	TCP	60	40229		361407		0x3087 (12…	52942 → 443 [ACK] S
127	37.284770	0.1287350…	0.246999000	10.36.9.27	10.16.1.251	TCP	66	2489		173312		0x094d (23…	52942 → 443 [ACK] S
1532	361.579939	0.0278840…	0.243206000	10.36.9.27	10.16.1.251	TCP	66	64515		1567028		0x4a69 (19…	52942 → 443 [ACK] S

Zoom in a little...



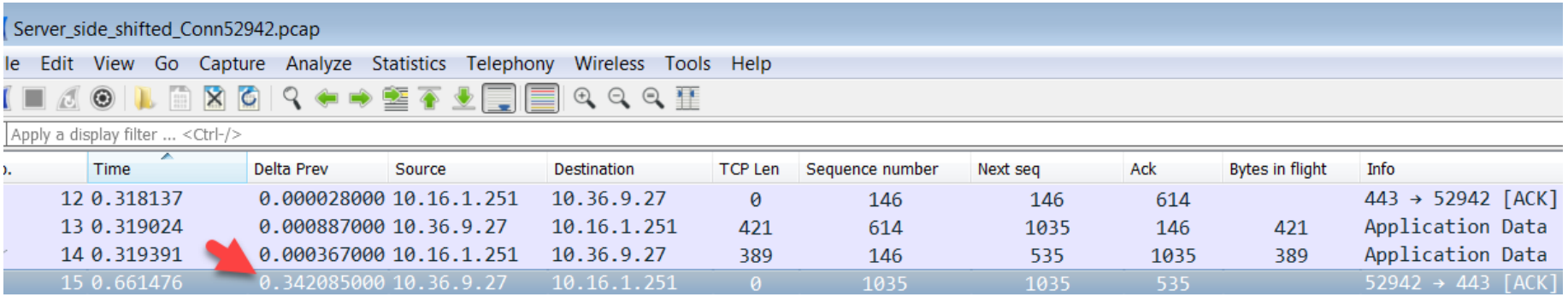
No.	Time	Display Delta	RTT2ACK	Source	Destination	Protocol
15	0.661476	0.3420850…	0.342085000	10.36.9.27	10.16.1.251	TCP
1844	409.110547	0.3348640…	0.334864000	10.36.9.27	10.16.1.251	TCP
576	317.047693	0.3324770…	0.332477000	10.36.9.27	10.16.1.251	TCP
248	130.359245	0.3319700…	0.331970000	10.36.9.27	10.16.1.251	TCP
472	233.001809	0.3291090…	0.329109000	10.36.9.27	10.16.1.251	TCP
427	220.077289	0.3277390…	0.327739000	10.36.9.27	10.16.1.251	TCP
416	187.728667	0.3276030…	0.327603000	10.36.9.27	10.16.1.251	TCP
163	70.246969	0.3255870…	0.325587000	10.36.9.27	10.16.1.251	TCP
399	163.546123	0.3252160…	0.325216000	10.36.9.27	10.16.1.251	TCP
489	266.335429	0.3239360…	0.323936000	10.36.9.27	10.16.1.251	TCP
214	91.593018	0.3211180…	0.321118000	10.36.9.27	10.16.1.251	TCP
231	93.964672	0.3205830…	0.320583000	10.36.9.27	10.16.1.251	TCP
463	222.872059	0.3201710…	0.320171000	10.36.9.27	10.16.1.251	TCP
1827	366.088071	0.3141830…	0.319613000	10.36.9.27	10.16.1.251	TCP

How else could we do this?

How else could we do this?

- Display filter…

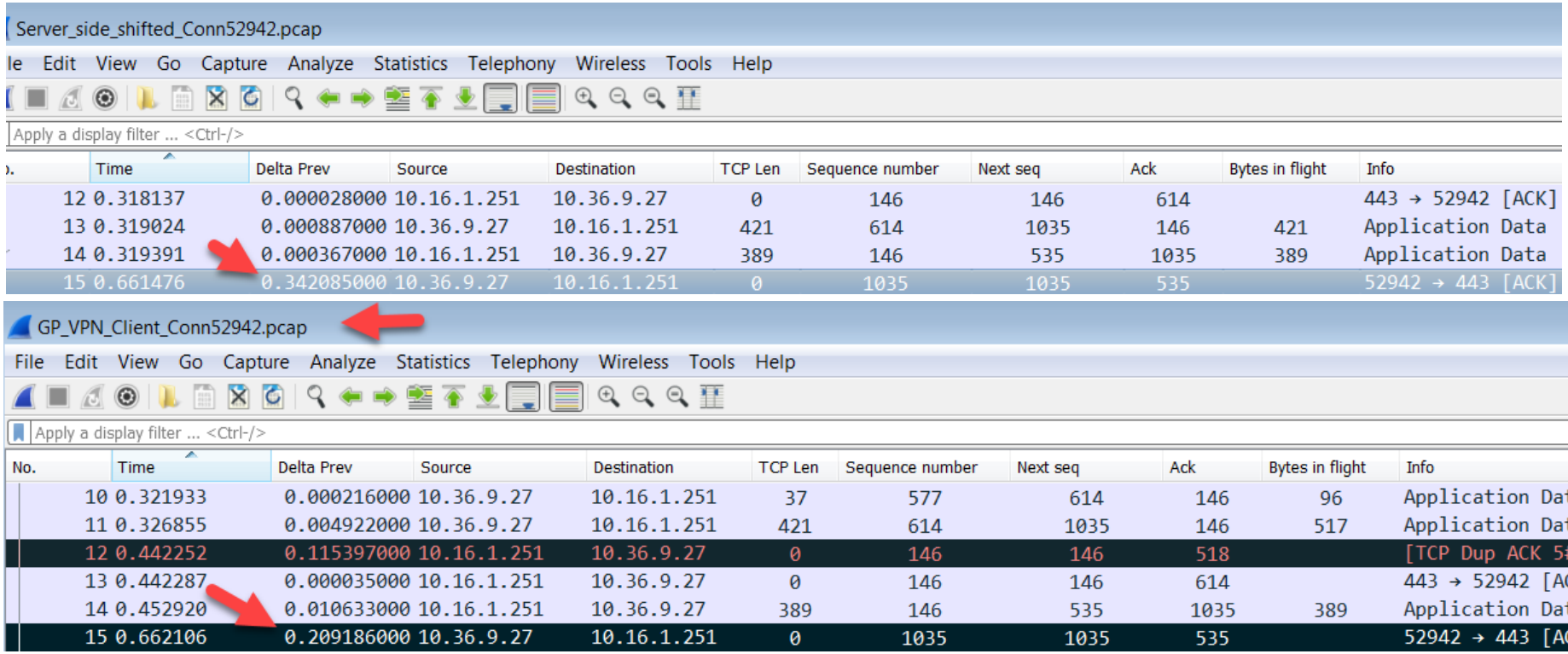
Server Cap Shows 342ms RTT



.	Time	Delta Prev	Source	Destination	TCP Len	Sequence number	Next seq	Ack	Bytes in flight	Info
12	0.318137	0.000028000	10.16.1.251	10.36.9.27	0	146	146	614		443 → 52942 [ACK]
13	0.319024	0.000887000	10.36.9.27	10.16.1.251	421	614	1035	146	421	Application Data
14	0.319391	0.000367000	10.16.1.251	10.36.9.27	389	146	535	1035	389	Application Data
15	0.661476	0.342085000	10.36.9.27	10.16.1.251	0	1035	1035	535		52942 → 443 [ACK]

- Could this be right? We just established RTT is 120-ish ms
- What is interesting about this ACK?

Client shows 209ms RTT?



Server_side_shifted_Conn52942.pcap

Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-/>

	Time	Delta Prev	Source	Destination	TCP Len	Sequence number	Next seq	Ack	Bytes in flight	Info
12	0.318137	0.000028000	10.16.1.251	10.36.9.27	0	146	146	614		443 → 52942 [ACK]
13	0.319024	0.000887000	10.36.9.27	10.16.1.251	421	614	1035	146	421	Application Data
14	0.319391	0.000367000	10.16.1.251	10.36.9.27	389	146	535	1035	389	Application Data
15	0.661476	0.342085000	10.36.9.27	10.16.1.251	0	1035	1035	535		52942 → 443 [ACK]

GP_VPN_Client_Conn52942.pcap

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

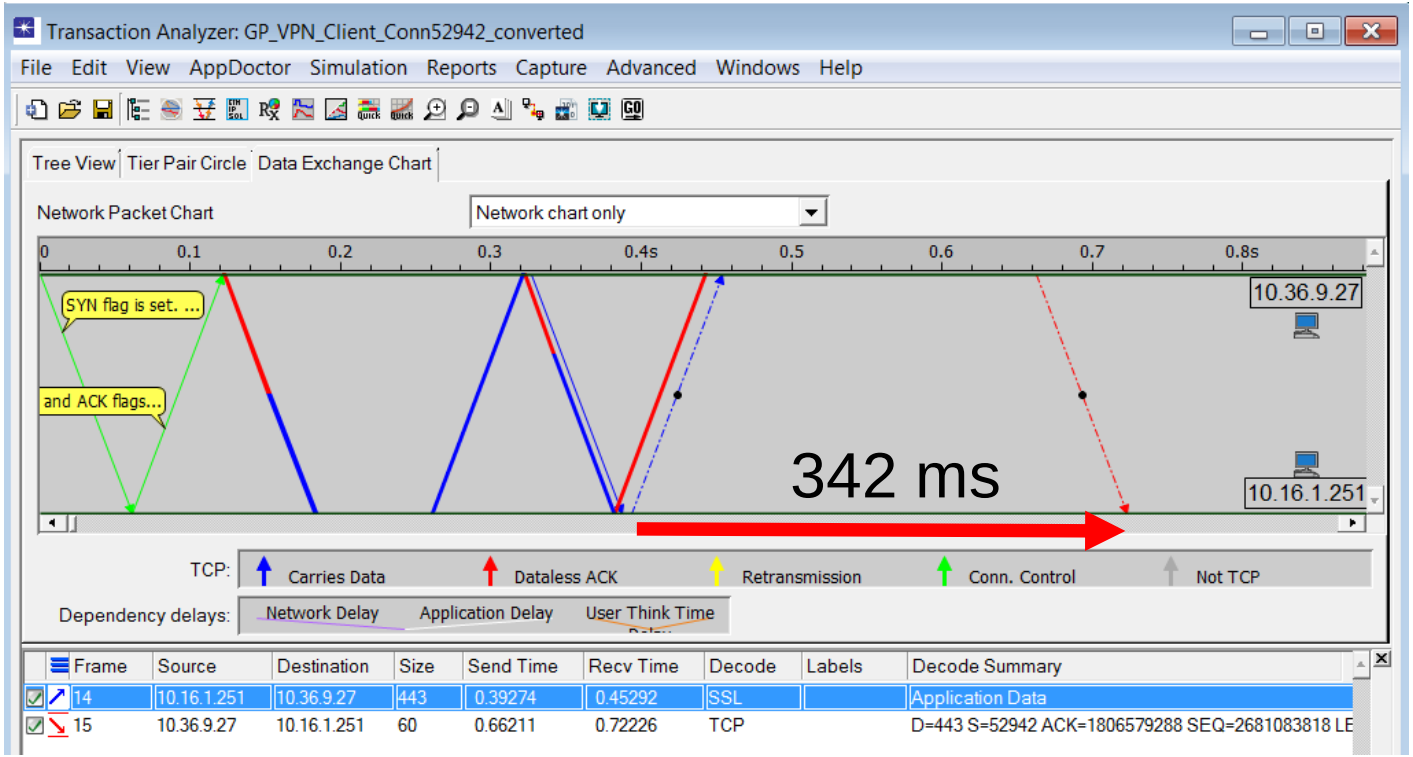
Apply a display filter ... <Ctrl-/>

No.	Time	Delta Prev	Source	Destination	TCP Len	Sequence number	Next seq	Ack	Bytes in flight	Info
10	0.321933	0.000216000	10.36.9.27	10.16.1.251	37	577	614	146	96	Application Data
11	0.326855	0.004922000	10.36.9.27	10.16.1.251	421	614	1035	146	517	Application Data
12	0.442252	0.115397000	10.16.1.251	10.36.9.27	0	146	146	518		[TCP Dup ACK 5#1]
13	0.442287	0.000035000	10.16.1.251	10.36.9.27	0	146	146	614		443 → 52942 [ACK]
14	0.452920	0.010633000	10.16.1.251	10.36.9.27	389	146	535	1035	389	Application Data
15	0.662106	0.209186000	10.36.9.27	10.16.1.251	0	1035	1035	535		52942 → 443 [ACK]

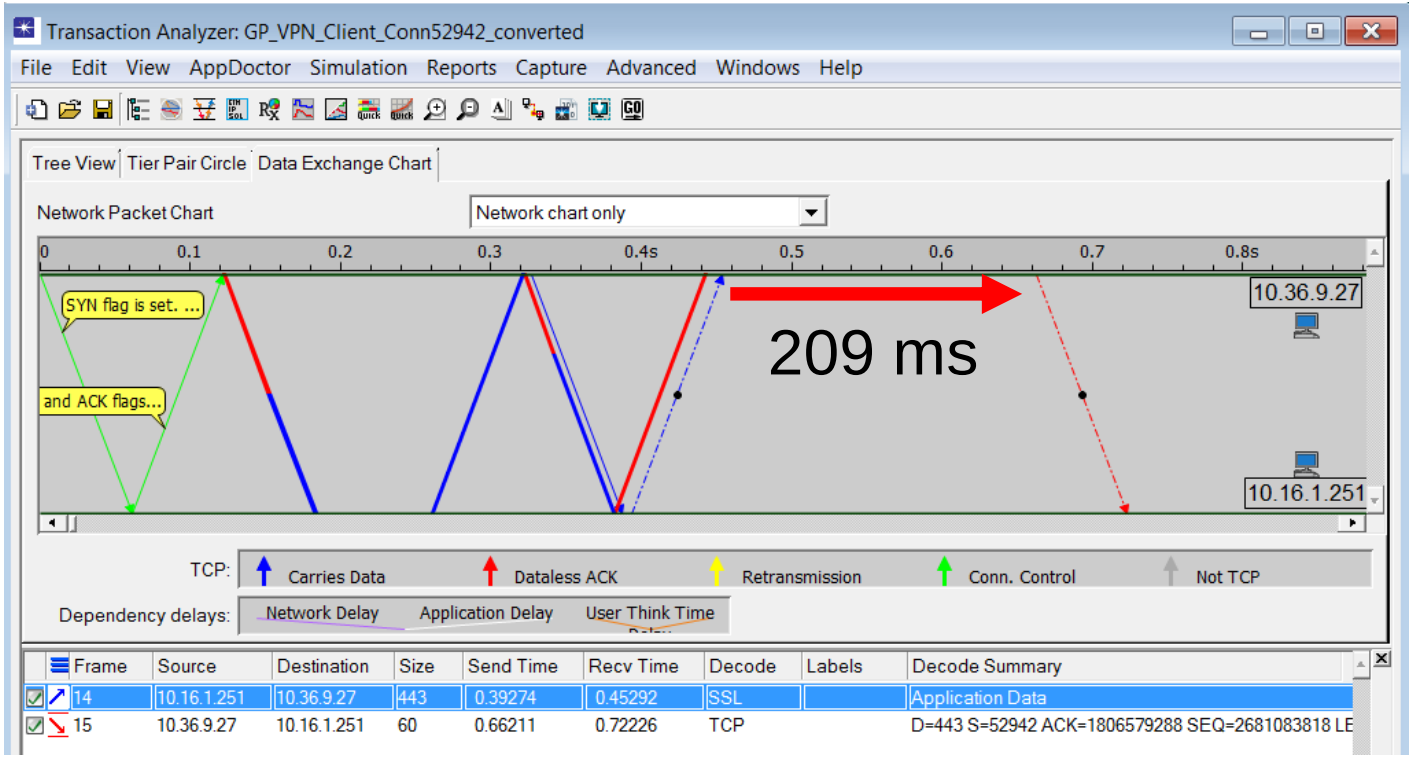
Recap

- Server shows 342ms RTT2ACK
- 1TTT + Delayed ACK Timer + 1TTT
- Delayed ACK Timer 209ms (from client capture)
- RTT 342ms – 209ms == 133ms (2 x 1TTT)
- Is this in the ball park?
- What's the smallest latency we've seen so far in these captures?

Let's Visualize – Server



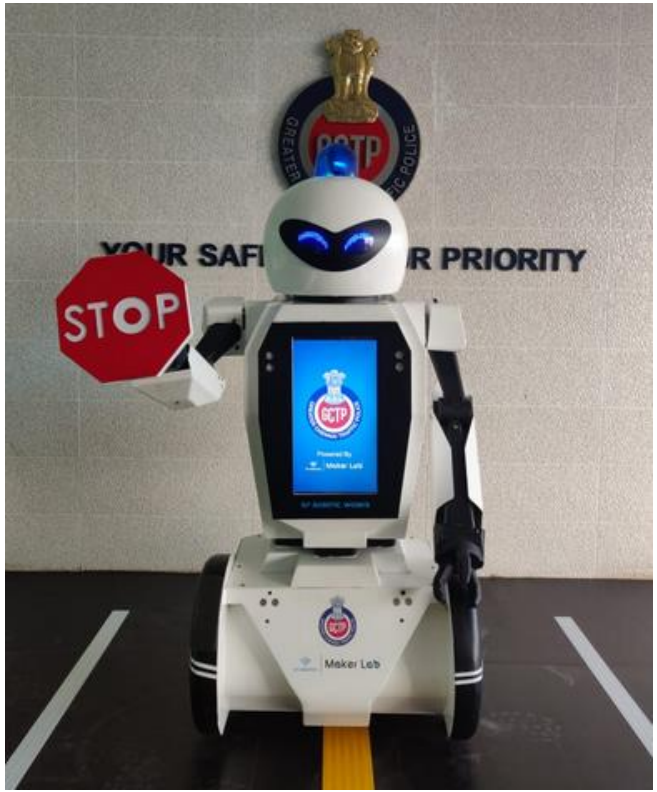
Let's Visualize – Client



Discussion

But Wait!

I thought we were here to talk about congestion?



We are, but it turns out we really need to understand latency first…

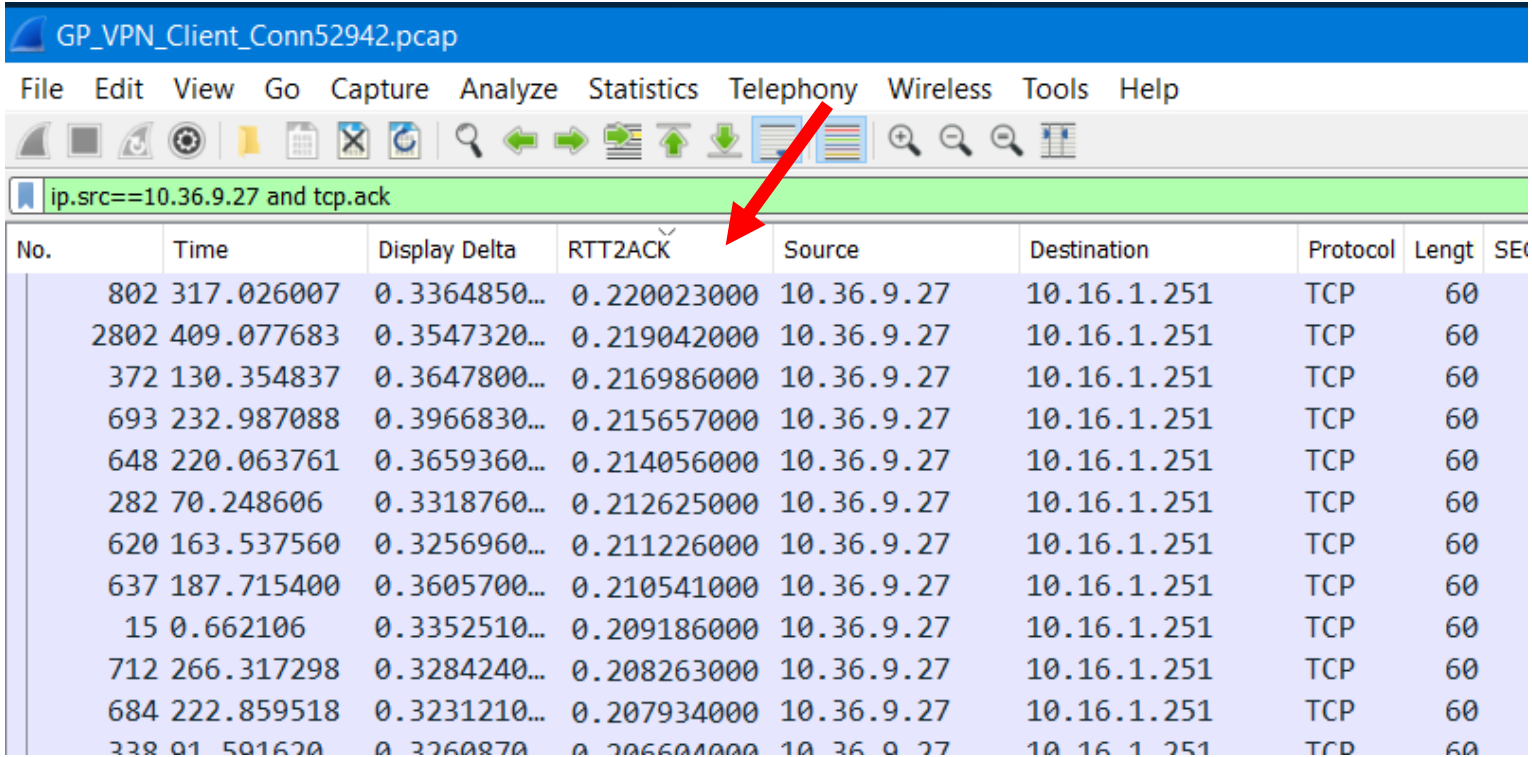
Can we Predict RTT?

- Forget the terms “Client” vs. “Server” momentarily
- Consider instead the terms “Sender” vs. “Receiver”
- What would you expect the possible time components for RTT to be for a Receiver capture?
- …and for a Sender side capture?
- (assume there’s “some” latency between hosts)

Receiver Capture

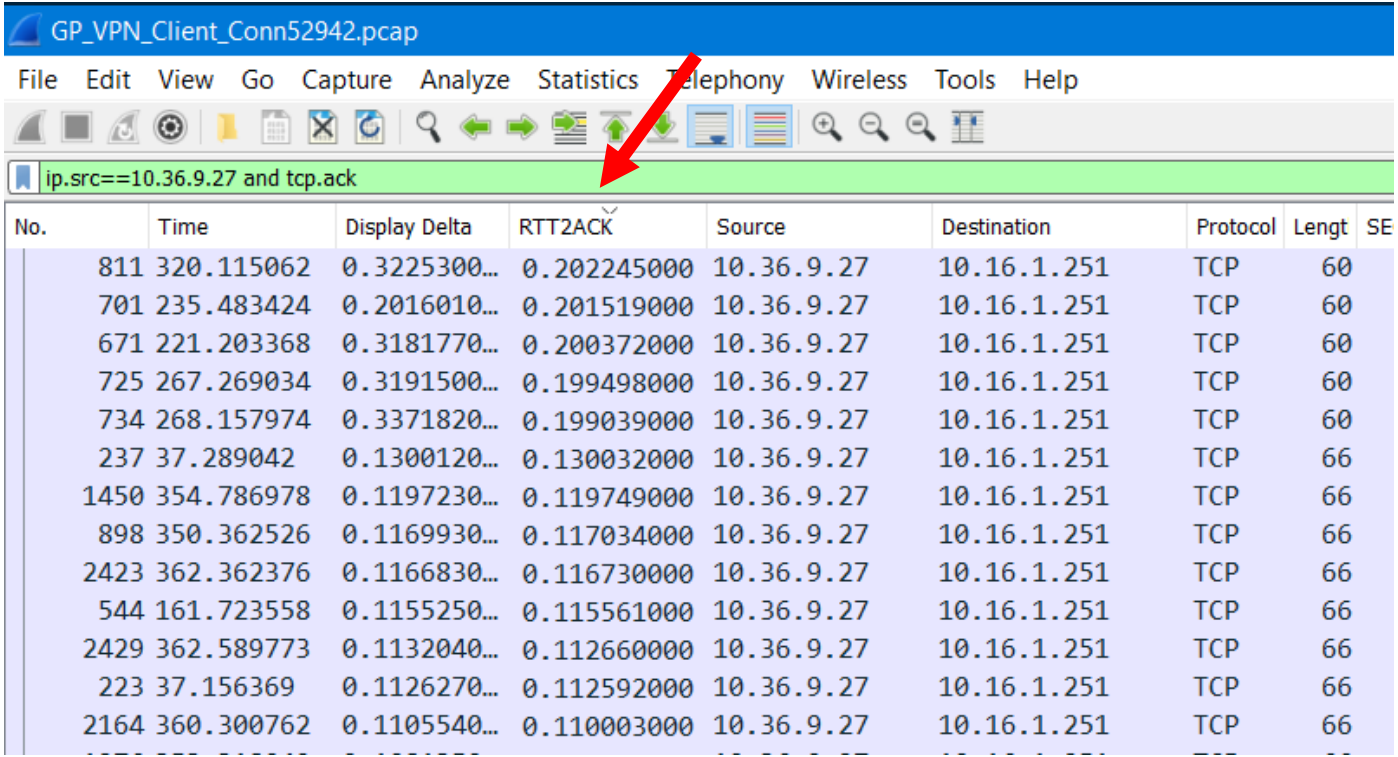
- Should be….
- Either super fast, or….
- Influenced by Delayed ACK Timer, and/or…
- Influenced by OOS and retransmissions

Receiver Capture



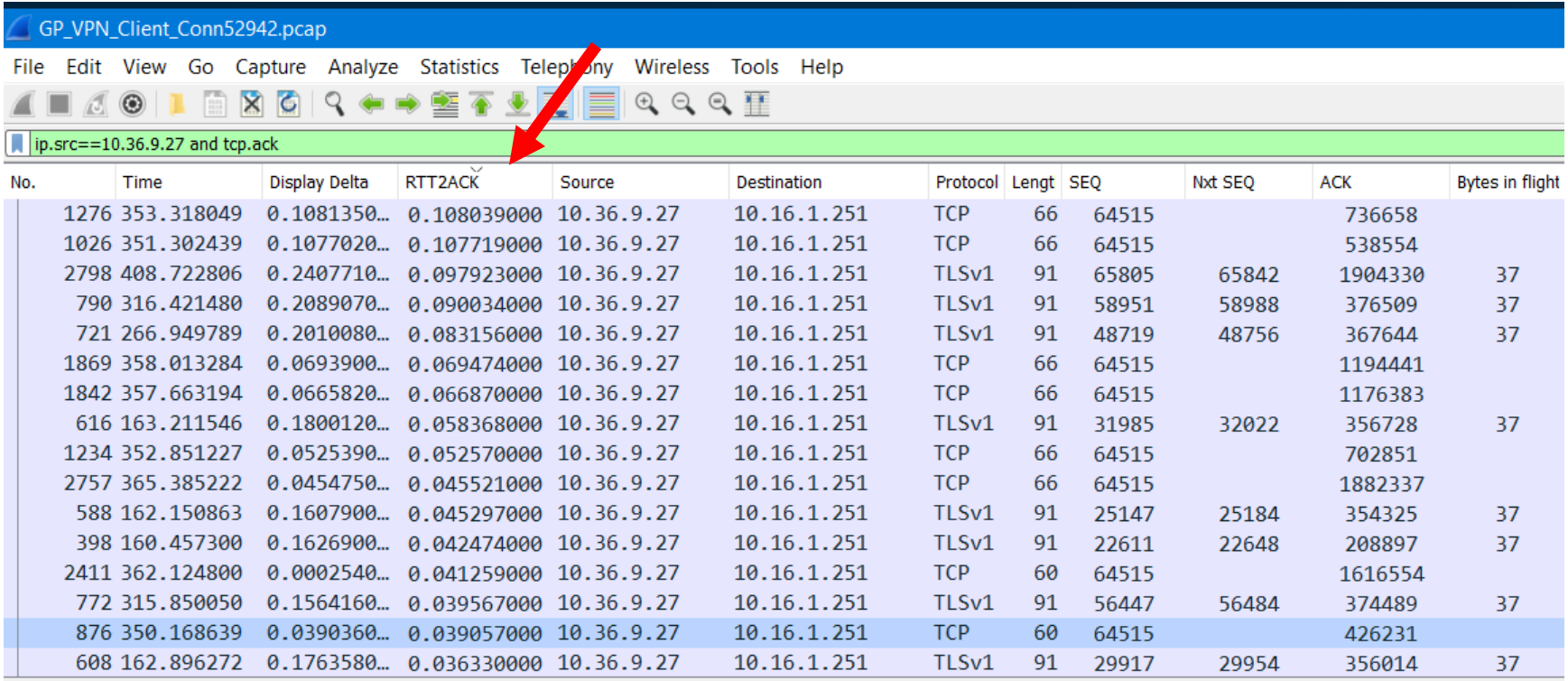
No.	Time	Display Delta	RTT2ACK	Source	Destination	Protocol	Lengt	SE
802	317.026007	0.3364850…	0.220023000	10.36.9.27	10.16.1.251	TCP	60	
2802	409.077683	0.3547320…	0.219042000	10.36.9.27	10.16.1.251	TCP	60	
372	130.354837	0.3647800…	0.216986000	10.36.9.27	10.16.1.251	TCP	60	
693	232.987088	0.3966830…	0.215657000	10.36.9.27	10.16.1.251	TCP	60	
648	220.063761	0.3659360…	0.214056000	10.36.9.27	10.16.1.251	TCP	60	
282	70.248606	0.3318760…	0.212625000	10.36.9.27	10.16.1.251	TCP	60	
620	163.537560	0.3256960…	0.211226000	10.36.9.27	10.16.1.251	TCP	60	
637	187.715400	0.3605700…	0.210541000	10.36.9.27	10.16.1.251	TCP	60	
15	0.662106	0.3352510…	0.209186000	10.36.9.27	10.16.1.251	TCP	60	
712	266.317298	0.3284240…	0.208263000	10.36.9.27	10.16.1.251	TCP	60	
684	222.859518	0.3231210…	0.207934000	10.36.9.27	10.16.1.251	TCP	60	
338	91.591620	0.3260870…	0.206604000	10.36.9.27	10.16.1.251	TCP	60	

Scroll down a little...



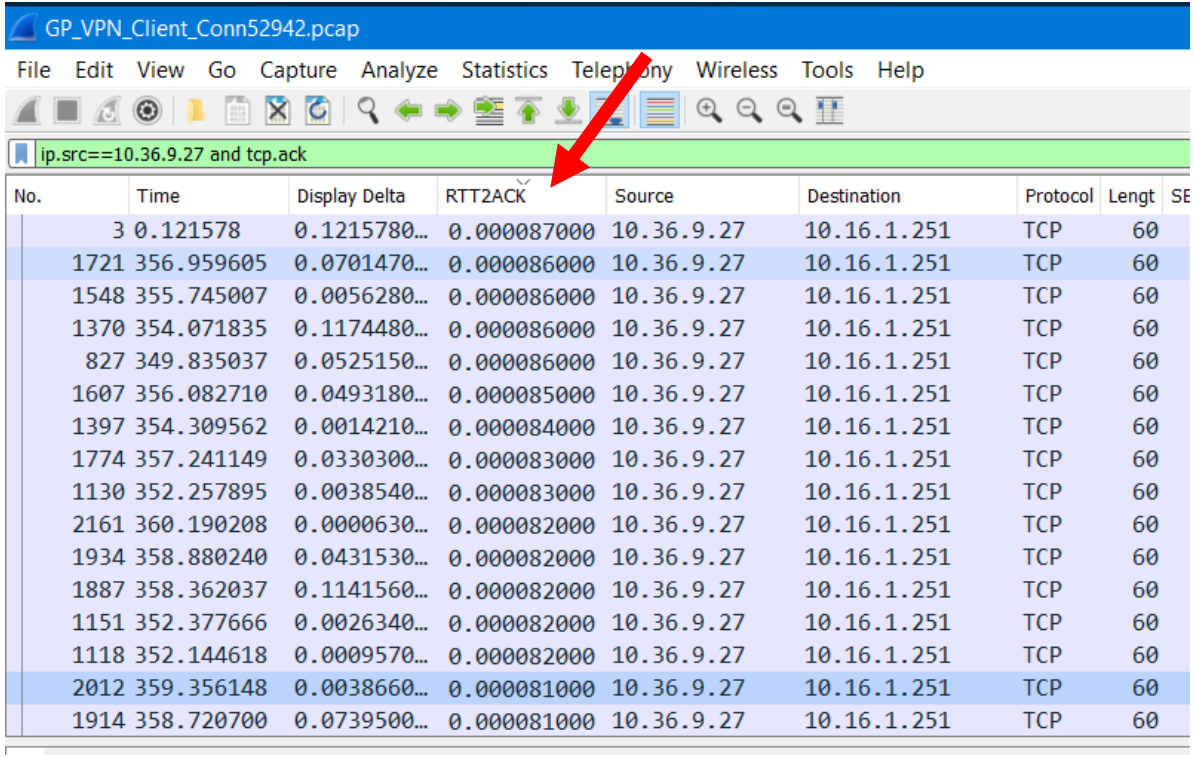
No.	Time	Display Delta	RTT2ACK	Source	Destination	Protocol	Lengt	SE
811	320.115062	0.3225300…	0.202245000	10.36.9.27	10.16.1.251	TCP	60	
701	235.483424	0.2016010…	0.201519000	10.36.9.27	10.16.1.251	TCP	60	
671	221.203368	0.3181770…	0.200372000	10.36.9.27	10.16.1.251	TCP	60	
725	267.269034	0.3191500…	0.199498000	10.36.9.27	10.16.1.251	TCP	60	
734	268.157974	0.3371820…	0.199039000	10.36.9.27	10.16.1.251	TCP	60	
237	37.289042	0.1300120…	0.130032000	10.36.9.27	10.16.1.251	TCP	66	
1450	354.786978	0.1197230…	0.119749000	10.36.9.27	10.16.1.251	TCP	66	
898	350.362526	0.1169930…	0.117034000	10.36.9.27	10.16.1.251	TCP	66	
2423	362.362376	0.1166830…	0.116730000	10.36.9.27	10.16.1.251	TCP	66	
544	161.723558	0.1155250…	0.115561000	10.36.9.27	10.16.1.251	TCP	66	
2429	362.589773	0.1132040…	0.112660000	10.36.9.27	10.16.1.251	TCP	66	
223	37.156369	0.1126270…	0.112592000	10.36.9.27	10.16.1.251	TCP	66	
2164	360.300762	0.1105540…	0.110003000	10.36.9.27	10.16.1.251	TCP	66	

A little more...



No.	Time	Display Delta	RTT2ACK	Source	Destination	Protocol	Lengt	SEQ	Nxt SEQ	ACK	Bytes in flight
1276	353.318049	0.1081350…	0.108039000	10.36.9.27	10.16.1.251	TCP	66	64515		736658	
1026	351.302439	0.1077020…	0.107719000	10.36.9.27	10.16.1.251	TCP	66	64515		538554	
2798	408.722806	0.2407710…	0.097923000	10.36.9.27	10.16.1.251	TLSv1	91	65805	65842	1904330	37
790	316.421480	0.2089070…	0.090034000	10.36.9.27	10.16.1.251	TLSv1	91	58951	58988	376509	37
721	266.949789	0.2010080…	0.083156000	10.36.9.27	10.16.1.251	TLSv1	91	48719	48756	367644	37
1869	358.013284	0.0693900…	0.069474000	10.36.9.27	10.16.1.251	TCP	66	64515		1194441	
1842	357.663194	0.0665820…	0.066870000	10.36.9.27	10.16.1.251	TCP	66	64515		1176383	
616	163.211546	0.1800120…	0.058368000	10.36.9.27	10.16.1.251	TLSv1	91	31985	32022	356728	37
1234	352.851227	0.0525390…	0.052570000	10.36.9.27	10.16.1.251	TCP	66	64515		702851	
2757	365.385222	0.0454750…	0.045521000	10.36.9.27	10.16.1.251	TCP	66	64515		1882337	
588	162.150863	0.1607900…	0.045297000	10.36.9.27	10.16.1.251	TLSv1	91	25147	25184	354325	37
398	160.457300	0.1626900…	0.042474000	10.36.9.27	10.16.1.251	TLSv1	91	22611	22648	208897	37
2411	362.124800	0.0002540…	0.041259000	10.36.9.27	10.16.1.251	TCP	60	64515		1616554	
772	315.850050	0.1564160…	0.039567000	10.36.9.27	10.16.1.251	TLSv1	91	56447	56484	374489	37
876	350.168639	0.0390360…	0.039057000	10.36.9.27	10.16.1.251	TCP	60	64515		426231	
608	162.896272	0.1763580…	0.036330000	10.36.9.27	10.16.1.251	TLSv1	91	29917	29954	356014	37

~Speed of Light RTT2ACK

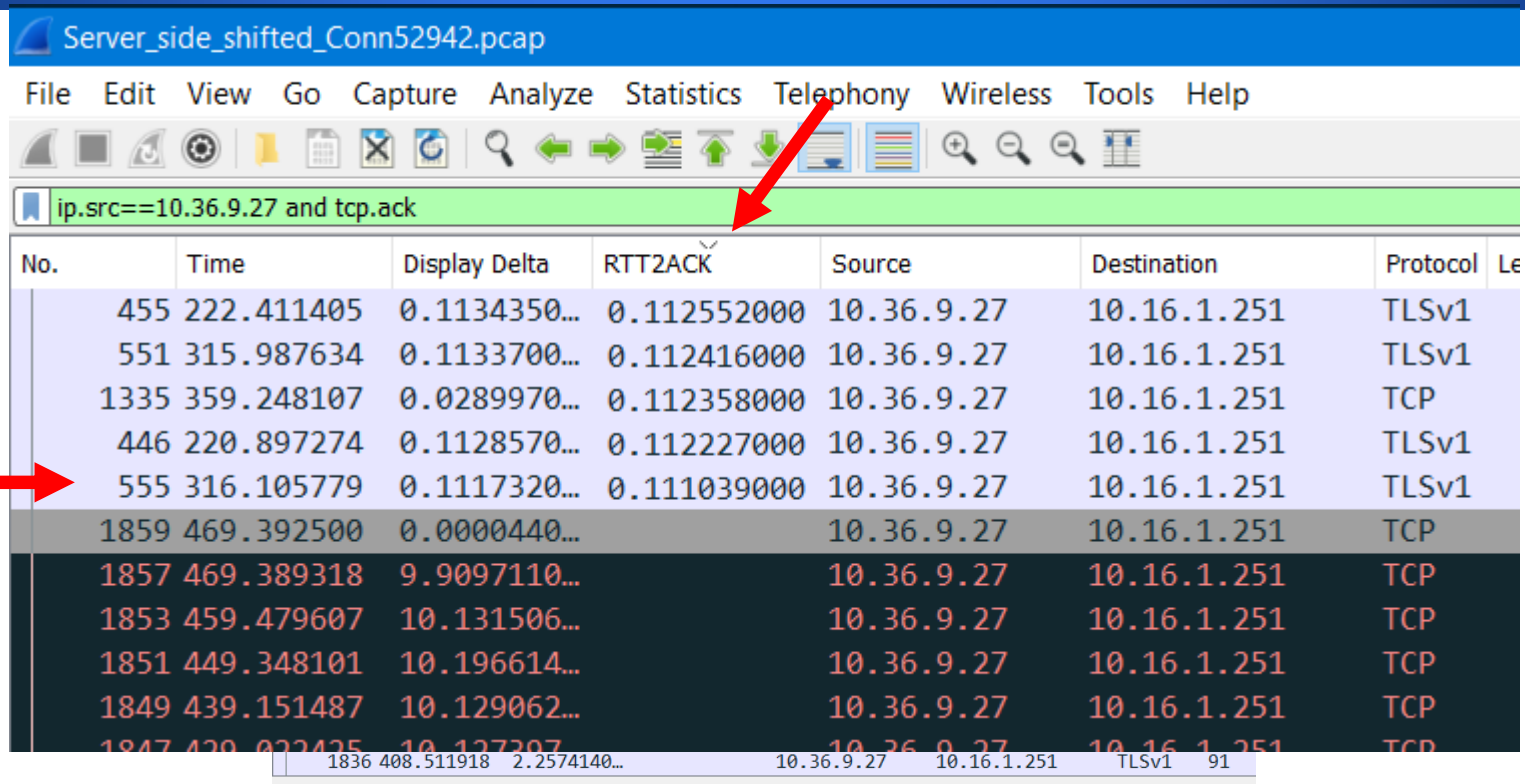


No.	Time	Display Delta	RTT2ACK	Source	Destination	Protocol	Lengt	SE
3	0.121578	0.1215780…	0.000087000	10.36.9.27	10.16.1.251	TCP	60	
1721	356.959605	0.0701470…	0.000086000	10.36.9.27	10.16.1.251	TCP	60	
1548	355.745007	0.0056280…	0.000086000	10.36.9.27	10.16.1.251	TCP	60	
1370	354.071835	0.1174480…	0.000086000	10.36.9.27	10.16.1.251	TCP	60	
827	349.835037	0.0525150…	0.000086000	10.36.9.27	10.16.1.251	TCP	60	
1607	356.082710	0.0493180…	0.000085000	10.36.9.27	10.16.1.251	TCP	60	
1397	354.309562	0.0014210…	0.000084000	10.36.9.27	10.16.1.251	TCP	60	
1774	357.241149	0.0330300…	0.000083000	10.36.9.27	10.16.1.251	TCP	60	
1130	352.257895	0.0038540…	0.000083000	10.36.9.27	10.16.1.251	TCP	60	
2161	360.190208	0.0000630…	0.000082000	10.36.9.27	10.16.1.251	TCP	60	
1934	358.880240	0.0431530…	0.000082000	10.36.9.27	10.16.1.251	TCP	60	
1887	358.362037	0.1141560…	0.000082000	10.36.9.27	10.16.1.251	TCP	60	
1151	352.377666	0.0026340…	0.000082000	10.36.9.27	10.16.1.251	TCP	60	
1118	352.144618	0.0009570…	0.000082000	10.36.9.27	10.16.1.251	TCP	60	
2012	359.356148	0.0038660…	0.000081000	10.36.9.27	10.16.1.251	TCP	60	
1914	358.720700	0.0739500…	0.000081000	10.36.9.27	10.16.1.251	TCP	60	

Sender Capture

- Never less than 2 x TTT (…right?)
- Add in Receiver's Delayed ACK Timer and/or
- Add in Receiver's time to wait for sender to fix OOS and Packet Loss

Sender Capture – Lowest

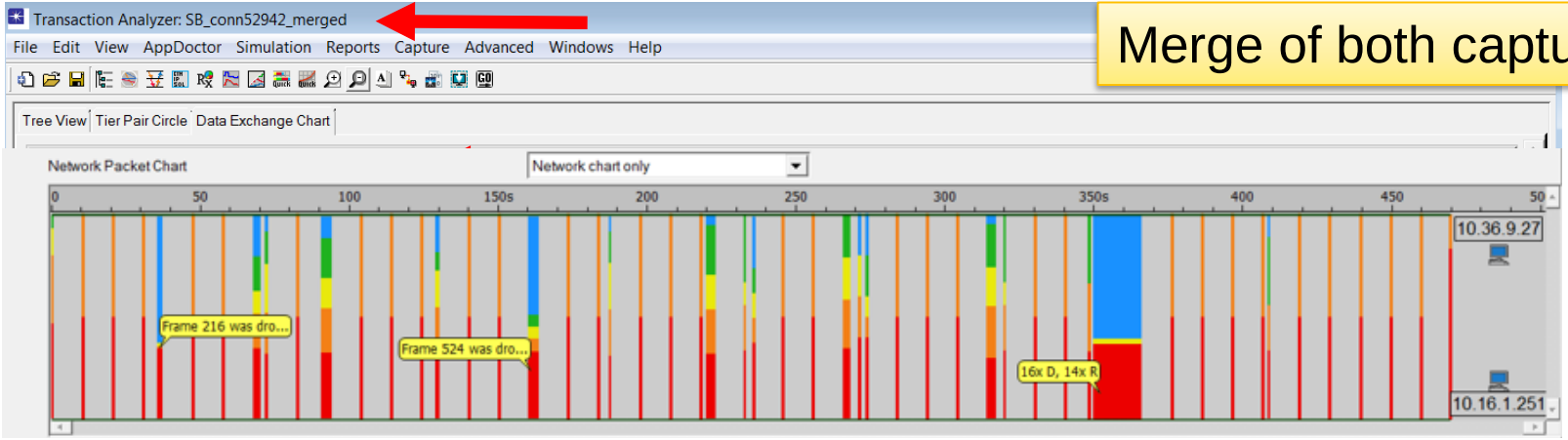


Enough about Latency...

More Visualizations

- Congestion is really hard to analyze with lists of packets and time deltas
- Let’s use visualizations starting at the 10K foot view and then drill down to the details
- We’ll use merged captures…

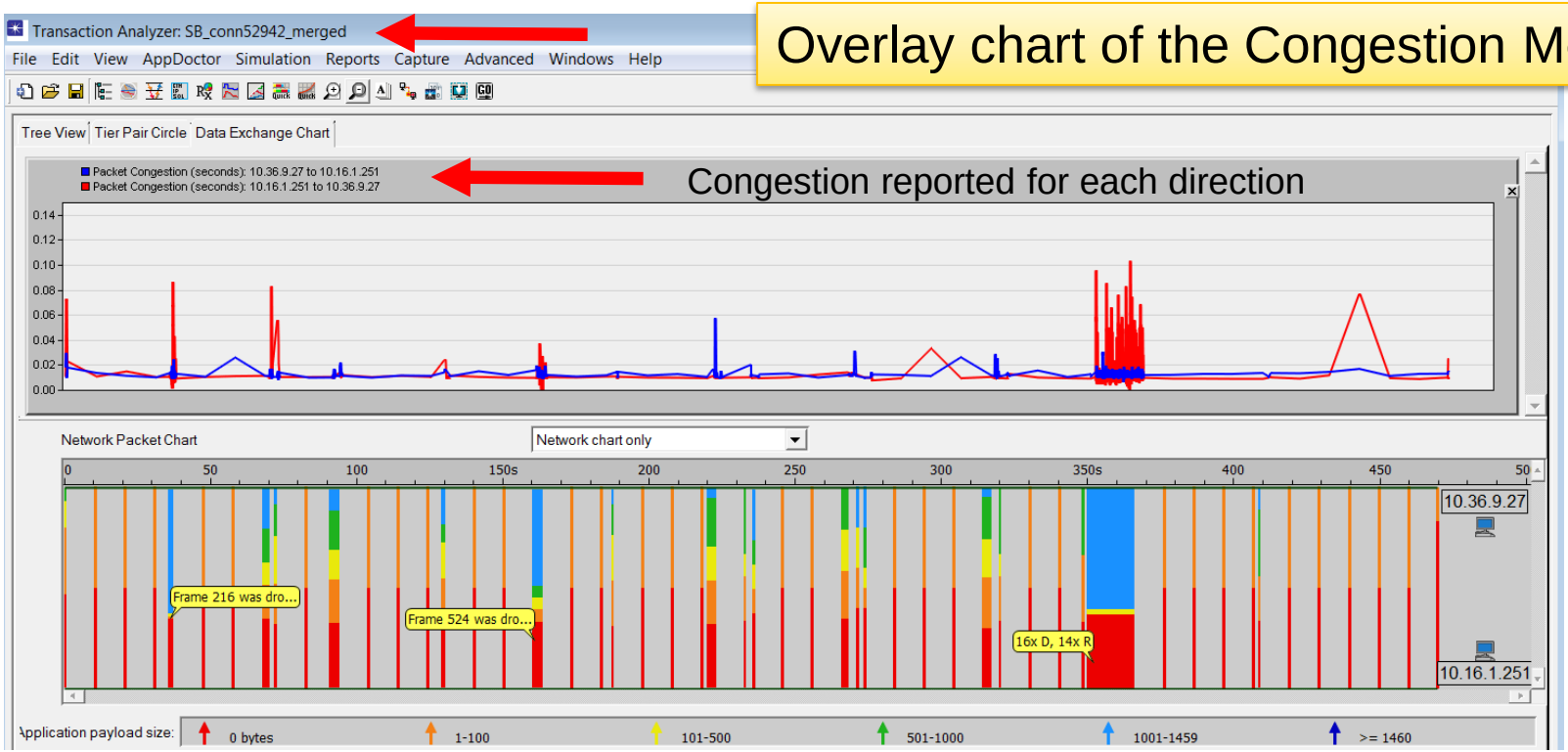
10,000 ft/m View



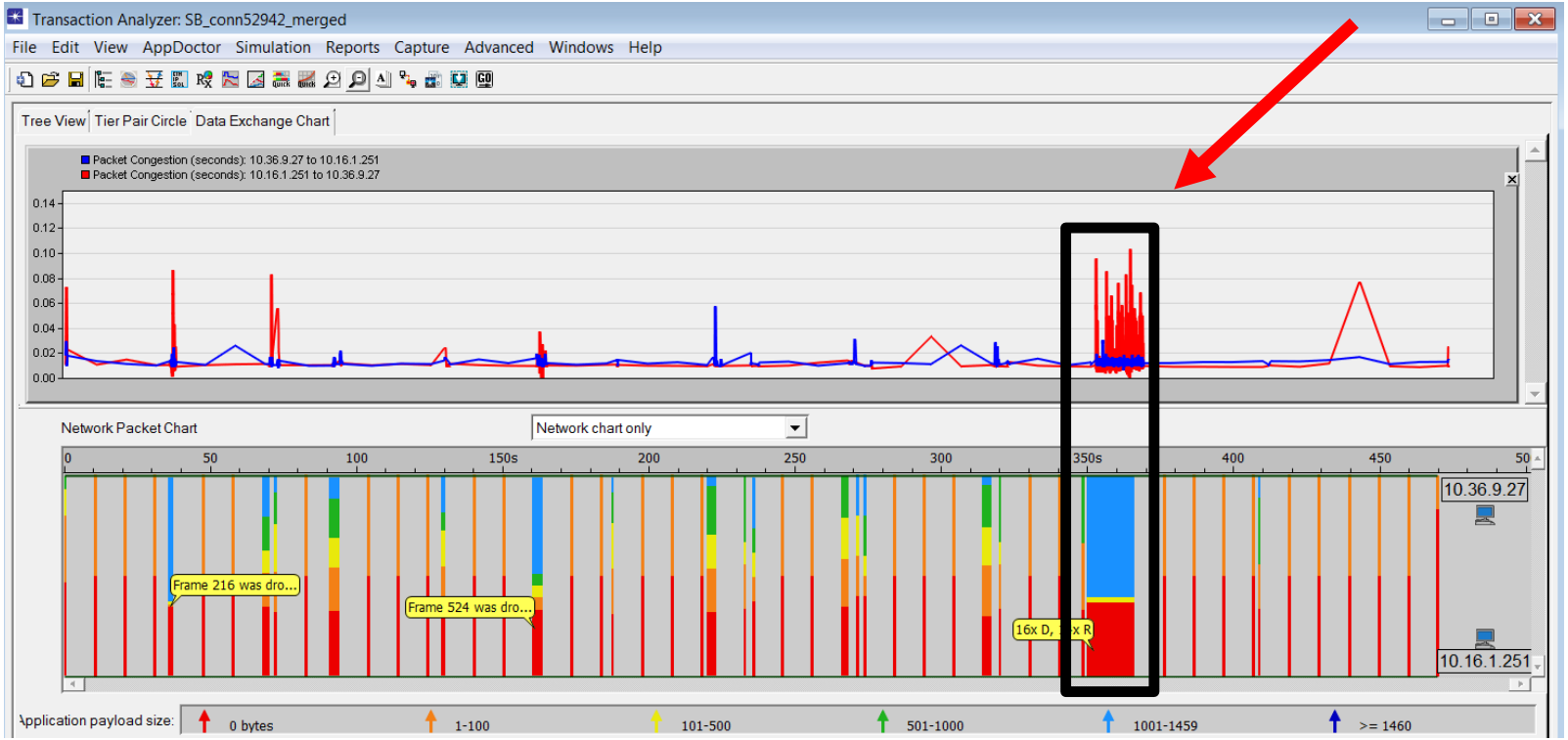
Merge of both captures

- Entire capture visualized
- You can see bursts of packet exchanges
- You can see the 10 sec keep alive pattern
- You can see call outs for packet loss

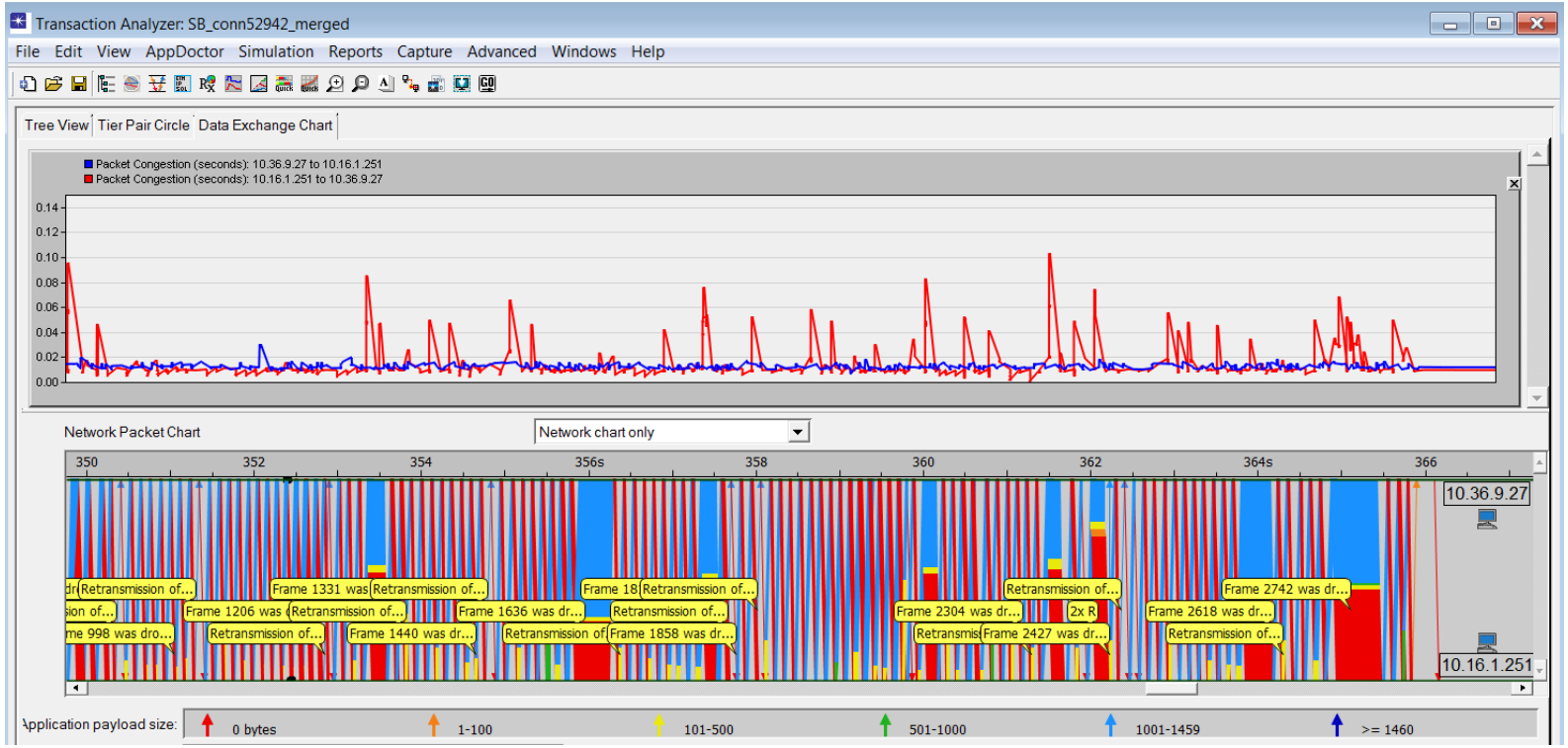
10,000 ft/m View



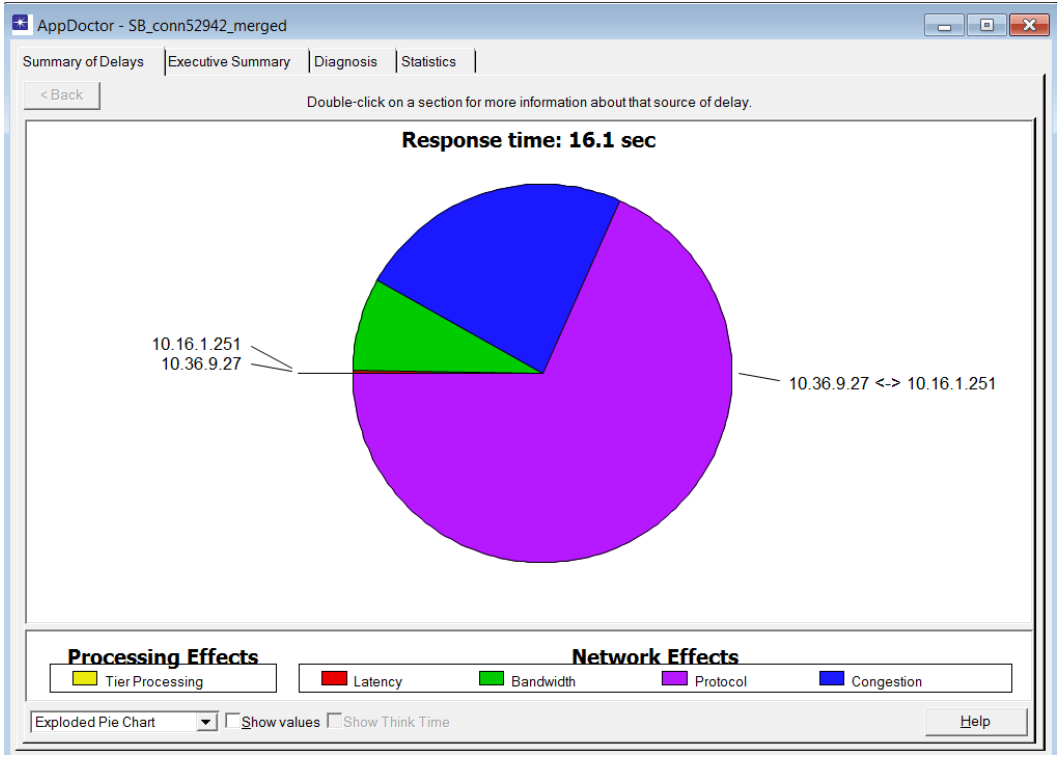
Let's Zoom-In


Transaction Analyzer: SB_conn52942_merged
File Edit View AppDoctor Simulation Reports Capture Advanced Windows Help
Tree View
Tier Pair Circle
Data Exchange Chart
Packet Congestion (seconds): 10.36.9.27 to 10.16.1.251
Packet Congestion (seconds): 10.16.1.251 to 10.36.9.27
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
Network Packet Chart
Network chart only
0
50
100
150s
200
250
300
350s
400
450
10.36.9.27
Frame 216 was dro...
Frame 524 was dro...
16x D,
x R
10.16.1.251
Application payload size:
0 bytes
1-100
101-500
501-1000
1001-1459
>= 1460

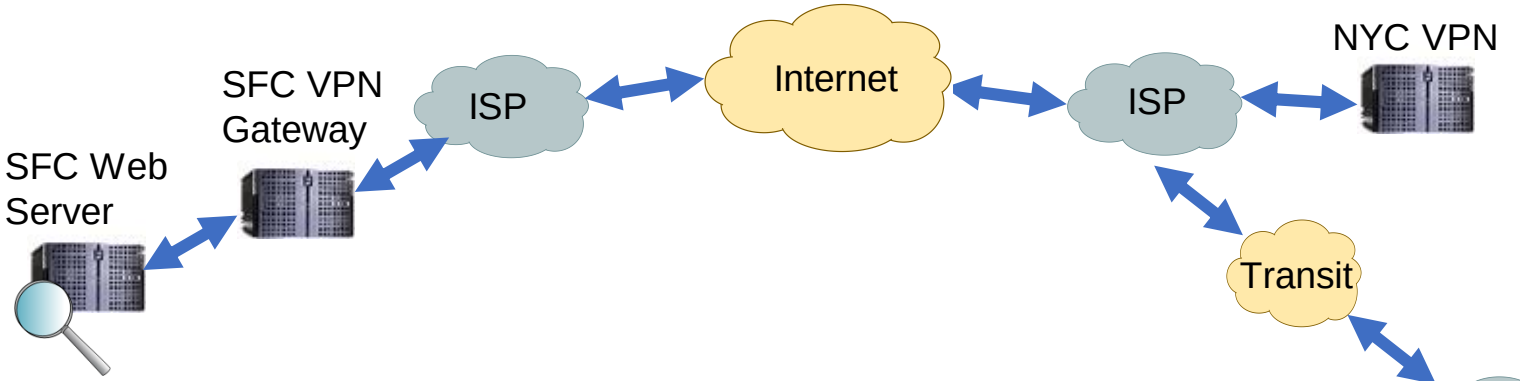
Zoom-in to File Download



Advanced Analytics: ~ 20% of overall delay (blue)



Why so much congestion?



- Multiple Tunnels & Protocols
- Multiple ISPs
- ESX Host Oversubscribed?
- VPN Client Overhead?

Congestion Recap

- Any surprises about the amount of congestion?
- Congestion occurs in both directions independently
- Does this view help to explain how congestion impacts performance?

Summary Results

TCP Split Brain Comparison Summary (Sender vs. Receiver)				
Item	Topic	Summary	Sender	Receiver
8	RTT2ACK		Can include receiver's Delayed ACK Timer; never less than 1 RTT	Generally Super Fast! When less than Delayed ACK Timer, could reflect degraded condition for TCP stack
9	Congestion	Requires advanced analytics		

Marathon Completed !

Summary & Wrap-Up

- Interpreting TCP behavior can be confusing and complicated, especially when there is “high” latency in the path
- Captures from both end points can be beneficial
- You need to split your brain into “server perspective” and “client perspective”

Summary & Wrap-Up

- Wireshark features are extremely helpful, but can only take you so far
- Visualization can help you understand behavior and quickly interpret root cause
- Advanced analytics are icing on the cake…

Recap Part I Topics

- Background on app, topology, and symptoms
- Compare and Contrast (aka Split Brain)
  - 3-way Handshake
  - Latency
  - Expert Info
  - Fragment Overlaps ,OOS, Retransmissions

Recap Part II Topics

- Continue where we left off
- Compare and contrast…
  - Bytes in Flight
  - Bonus Topic
- Session Wrap-Up

Full Summary Results

TCP Split Brain Comparison Summary (Sender vs. Receiver)

Item	Topic	Summary	Sender	Receiver
1	SYN Options	MSS, Scaling, TimeStamps, SACK	Negotiation & Adapt, MSS=1460, WS=8, SACK	Negotiation & Adapt, MSS=1360, WS=8, SACK
2	Latency	Can be different in each direction	Client iRTT == 121ms	Server iRTT == 129ms
3	TCP State	Different States at startup and shutdown; state change only occurs when packet sent / received	Closed, SYN-Sent, Established	Listen, SYN-Received, Established
4	Expert Info			
4.1	Duplicate ACK	Should be close to the same on both captures		
4.2	OOS		Not expected on sender; but retrans could be interpretted as OOS	Will be higher than sender
4.3	Retransmissions		Will be higher than receiver; could also be flagged as OOS	Might be flagged as OOS
4.4	Previous Segement Not Captured	Can be different	Not expected on sender unless there's a capture integrity issue	Expected on receiver when there's packet loss or OOS
4.5	SSL Errors	Could be side effect of reassembly in presence of OOS	Not expected	Likely to be reassembly issue
4.6	Fragment Overlap	Most likely caused by Segmentation Offoad	Most likely to show up on Sender's capture	Receiver could flag overlap using SACK field
5	Frame Sizes	Can be different	Effects of LSO / IP Fragmentation	Effects of LRO
6	Display Time Delta	Very unique to each endpoint	ACKs will usually apply to segments sent much earlier in time	Interpretation can seem confusing, especially when lots of packets are in flight
7	Bytes In-Flight	OOS and retransmissions may impact calculation	Should be pretty accurate	Generally not very interesting from receiver's capture
8	RTT2ACK		Can include receiver's Delayed ACK Timer; never less than 1 RTT	Generally Super Fast! When less than Delayed ACK Timer, could reflect degraded condition for TCP stack
9	Congestion	Requires advanced analytics		
10	Service Response Time		Client: May include latency + protocol delay + congestion	Server: should be most accurate

Final Questions / Comments

End of Session