

SHARKFEST '12

Wireshark Developer and User Conference

Mike Canney

Application Performance Analysis

Tekktivitiy™

Welcome to Sharkfest '12

Mike Canney,
Principal Network Analyst, Tektivity, Inc.

canney@getpackets.com

319-365-3336

www.getpackets.com

contact

Agenda

agenda

- So why focus on the application?
- Creating a CDA (Capture to Disk Appliance)
- Using Pilot for “back in time” troubleshooting with your CDA and Wireshark
- Application QA Lifecycle
- Top Causes for Application Performance issues
 - Application Turns
 - TCP
 - Layer 7 Issues
 - TCP Retransmissions
- Using Wireshark to create custom profiles to troubleshoot CIFS/SMB

So why focus on the Application?

- In many cases it is the Network Engineers that have the tool set to help pinpoint where the problem exists.
- “It’s not the Network!” - The Network is guilty until proven innocent.
- Application performance issues can impact your business/customers ability to make money.
- User Response time is “Relative”.
- Intermittent performance issues (moving target).

focus

The “moving target”

- Analyzer placement - Two options
 - Move the analyzers as needed
 - Capture anywhere and everywhere
- To defend the Network multiple capture points of the problem is the best solution.

target

Commercial vs. Free Capture

capture

- Define your capture strategy
 - Data Rates
 - What are my goals? Troubleshooting vs. Statistical information.
 - Do I need to capture every packet?

Capture to Disk Appliance (on a budget)

- What is needed?
 - dumpcap is a command line utility included with the Wireshark download to enable ring buffer captures.
 - Use an inexpensive PC or laptop (best to have 2 NICs or more).
 - Basic batch file to initiate capture.
 - Cascade Pilot (optional but recommended)

budget

Dumpcap Example

```
cd \program files (x86)\wireshark
dumpcap -i 1 -s 128 -b files:100 -b filesize:
2000000 -w c:\traces\internet
\headersonly1.pcap
```

This is a basic batch file that will capture off of interface 1, slice the packets to 128 bytes, write 100 trace files of ~2 Gigabytes, and write the trace file out to a pcap file.

So why did I write multiple 2 Gig trace files?

trace file

- Pilot!
- Pilot can easily read HUGE trace files.
- This allows us to utilize our CDA in ways no other analyzer can.
- I personally have sliced and diced 50 GB trace files in Pilot in a matter of seconds.

So how does this all work together?

practice

- Directory full of 2GB trace files, all time stamped based on when they were written to disk.
- User calls in and complains that “the network” is slow.
- Locate that trace file based on time and date and launch Pilot.

Instructor Demo

demo

**Troubleshooting user
“Network Issue”**

Think about the possibilities...

- From a multix GB trace file we were able to:
 - Look at the total Network throughput.
 - See what applications were consuming the bandwidth.
 - Identify the user that was responsible for consuming the bandwidth.
 - Identify the URI's the user was hitting and what the response times were.
 - Drill down to the packets involved in the slow web response time in Wireshark.
- All in a matter of a few seconds.

hmm

Why are there so many application issues?

- Applications are typically developed in a “golden” environment
 - Fastest PCs
 - High Bandwidth/low latency
- When applications move from test (LAN) to production (WAN) the phone starts ringing with complaints coming in.

help

The Application QA Lifecycle

- In most organizations, applications go through a QA process
- Typical QA/App developers test the following:
 - Functional tests
 - Regression tests
 - Stress tests (server)
 - Rinse and Repeat
- What is often missing is “Networkability” testing
- All QA Lifecycles should include Networkability testing

qa cycle

Application Networkability Testing

- Identify key business transactions, number of users and network conditions the application will be deployed in.
- Simulation vs. Emulation
 - Simulation is very quick, often gives you rough numbers of how an application will perform over different network conditions.
 - Emulation is the only way to determine when an application will “fail” under those conditions.
- A Combination of both is recommended.

testing

Top Causes for Poor Application Performance

- Application Turns
- TCP
- Layer 7 Bottlenecks
- Congestion (network)
- Processing Delay

top 5

Causes for Slow Application Performance

Application Turns

turns

Application Turns

- An Application Turn is a request/response pair
- For each “turn” the application must wait the full round trip delay.
- The greater the number of turns, the worse the application will perform over a WAN (latency bound).

turns

App Turn

Begin

```
GET /assets/images/riverbed_logo.png HTTP/1.1  
[TCP segment of a reassembled PDU]  
[TCP segment of a reassembled PDU]  
49222 > http [ACK] Seq=573 Ack=2921 win=16060 Len=0  
[TCP Window Update] 49222 > http [ACK] Seq=573 Ack=2921 win=1  
HTTP/1.1 200 OK (PNG)  
GET /assets/photos/Riverbed_Cascade_home_010211.png HTTP/1.1
```

End

Example in Wireshark

Display Filter:

Filter:

No.	Time	From	To	Length	Protocol	Info
10.008388	0.096607	237 192.168.151.108	183 192.168.151.122		SMB	Read AndX Response, FID: 0xc00d, 61440 bytes
10.008935	0.000547	117 192.168.151.122	63 192.168.151.108		SMB	Read AndX Request, FID: 0xc00d, 61440 bytes at offset 61440
10.105423	0.096488	237 192.168.151.108	183 192.168.151.122		SMB	Read AndX Response, FID: 0xc00d, 61440 bytes
10.105972	0.000549	117 192.168.151.122	63 192.168.151.108		SMB	Read AndX Request, FID: 0xc00d, 61440 bytes at offset 122880
10.202297	0.096325	237 192.168.151.108	183 192.168.151.122		SMB	Read AndX Response, FID: 0xc00d, 61440 bytes
10.202691	0.000394	117 192.168.151.122	63 192.168.151.108		SMB	Read AndX Request, FID: 0xc00d, 61440 bytes at offset 184320
10.299228	0.096537	237 192.168.151.108	183 192.168.151.122		SMB	Read AndX Response, FID: 0xc00d, 61440 bytes
10.299569	0.000341	117 192.168.151.122	63 192.168.151.108		SMB	Read AndX Request, FID: 0xc00d, 16384 bytes at offset 245760
10.358427	0.058858	441 192.168.151.108	387 192.168.151.122		SMB	Read AndX Response, FID: 0xc00d, 16384 bytes
10.358662	0.000235	117 192.168.151.122	63 192.168.151.108		SMB	Read AndX Request, FID: 0xc00d, 45056 bytes at offset 262144
10.441337	0.082675	1373 192.168.151.108	1319 192.168.151.122		SMB	Read AndX Response, FID: 0xc00d, 45056 bytes
10.445493	0.004156	117 192.168.151.122	63 192.168.151.108		SMB	Read AndX Request, FID: 0xc00d, 61440 bytes at offset 307200
10.541826	0.096333	237 192.168.151.108	183 192.168.151.122		SMB	Read AndX Response, FID: 0xc00d, 61440 bytes
10.542163	0.000337	117 192.168.151.122	63 192.168.151.108		SMB	Read AndX Request, FID: 0xc00d, 61440 bytes at offset 368640
10.638633	0.096470	237 192.168.151.108	183 192.168.151.122		SMB	Read AndX Response, FID: 0xc00d, 61440 bytes
10.639010	0.000377	117 192.168.151.122	63 192.168.151.108		SMB	Read AndX Request, FID: 0xc00d, 61440 bytes at offset 430080
10.735377	0.096367	237 192.168.151.108	183 192.168.151.122		SMB	Read AndX Response, FID: 0xc00d, 61440 bytes

Packets: 22388 Displayed: 882

882 Application Turns in this trace

App Turns and Latency

latency

- It is fairly easy to determine App Turns impact on end user response time
 - Multiply the number of App Turns by the round trip delay:
 - $10,000 \text{ turns} * .050 \text{ ms delay} = 500 \text{ seconds due to latency}$
- Note, this has nothing to do with Bandwidth or the Size of the WAN Circuit

So what causes all these App Turns?

- Size of a fetch in a Data Base call
- Number of files that are being accessed
- Loading single images in a Web Page instead of using an image map
- Number of bytes being retrieved and how they are being retrieved (block size)

cause

Causes for Slow Application Performance

TCP

tcp

TCP Window Size

- The TCP Window Size defines the host's receive buffer.
- Large Window Sizes can sometimes help overcome the impact of latency.
- Depending on how the application was written, advertised TCP Window Size may not have an impact at all (more on this later).

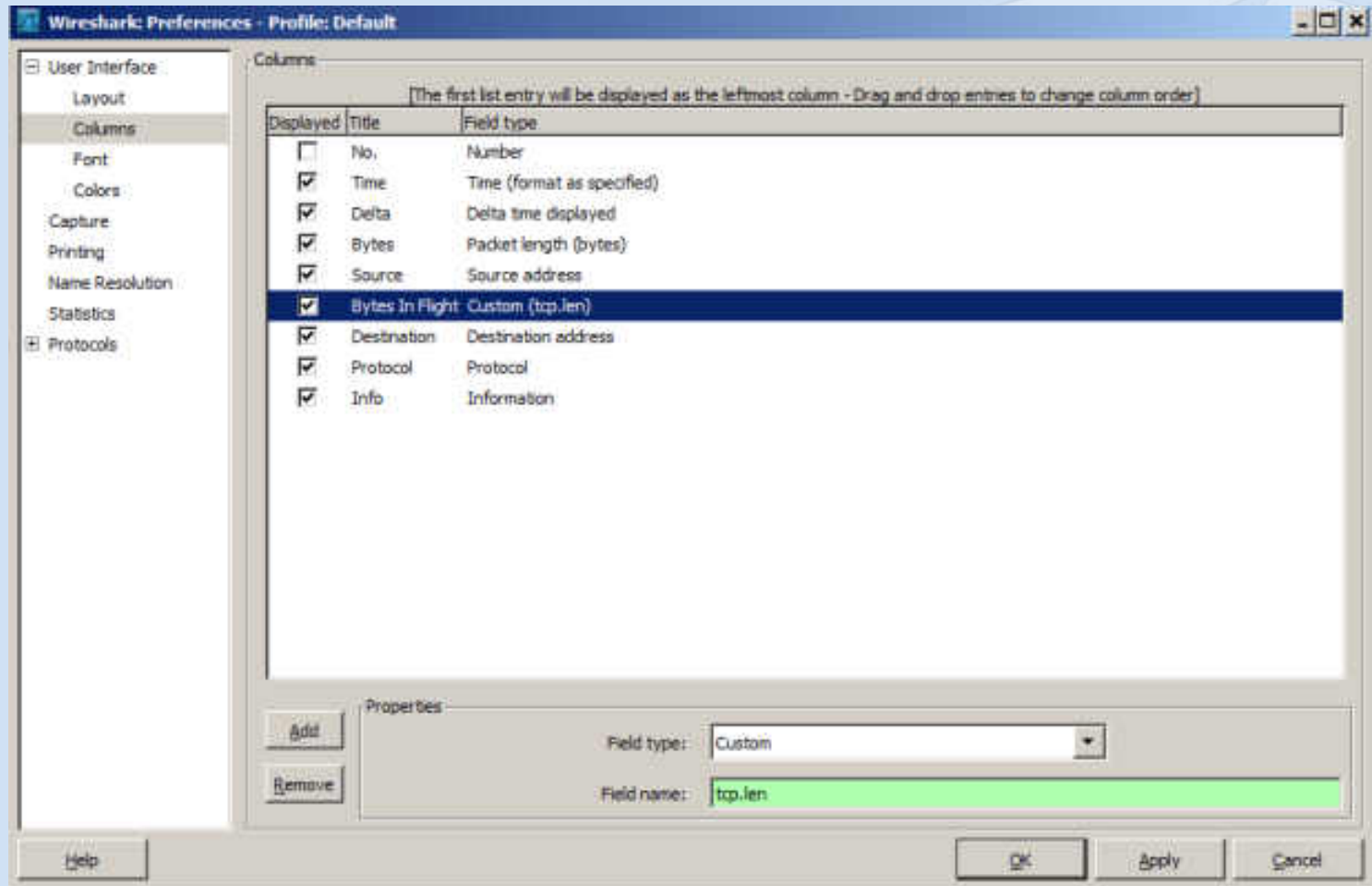
size

TCP Inflight Data

- The amount of unacknowledged TCP data that is on the wire at any given time.
- TCP inflight data is limited by the following:
 - TCP Retransmissions
 - TCP Window Size
 - Application block size
- The amount of TCP inflight data will never exceed the receiving device's advertised TCP Window Size.

inflight

TCP Inflight Data in Wireshark



TCP Inflight Data in Wireshark

No.	Time	Source	Destination	Protocol	Length	Info
4.888131	0.000170	130 192.168.151.108	76 192.168.151.122	SMB		NT TRANS Response, NT NOTIFY
4.888215	0.000064	54 192.168.151.122	0 192.168.151.108	TCP	1041 > 445 [ACK] Seq=682 Ack=2514 win=63969 Len=0	
4.888499	0.000284	142 192.168.151.122	88 192.168.151.108	SMB		Trans2 Request, SET_FILE_INFO, FID: 0xc000
4.888828	0.000329	142 192.168.151.122	88 192.168.151.108	SMB		NT Trans Request, NT NOTIFY, FID: 0x4005
4.933909	0.045081	118 192.168.151.108	64 192.168.151.122	SMB		Trans2 Response, FID: 0xc000, SET_FILE_INFO
4.934090	0.000181	130 192.168.151.108	76 192.168.151.122	SMB		NT Trans Response, NT NOTIFY
4.934149	0.000059	54 192.168.151.122	0 192.168.151.108	TCP	1041 > 445 [ACK] Seq=858 Ack=2654 win=63829 Len=0	
4.939393	0.005244	142 192.168.151.122	88 192.168.151.108	SMB		NT Trans Request, NT NOTIFY, FID: 0x4005
5.197543	0.258150	60 192.168.151.108	0 192.168.151.122	TCP	445 > 1041 [ACK] Seq=2654 Ack=946 win=63663 Len=0	
5.197735	0.000192	142 192.168.151.122	88 192.168.151.108	SMB		NT Trans Request, NT NOTIFY, FID: 0x4006
5.243042	0.045307	130 192.168.151.108	76 192.168.151.122	SMB		NT Trans Response, NT NOTIFY
5.243344	0.000302	142 192.168.151.122	88 192.168.151.108	SMB		NT Trans Request, NT NOTIFY, FID: 0x4006
5.506404	0.263060	60 192.168.151.108	0 192.168.151.122	TCP	445 > 1041 [ACK] Seq=2730 Ack=1122 win=63487 Len=0	
6.736559	1.230155	144 192.168.151.122	90 192.168.151.108	SMB		Trans2 Request, FIND_FIRST2, Pattern: *
6.784039	0.047480	1514 192.168.151.108	1460 192.168.151.122	TCP		[TCP segment of a reassembled PDU]
6.784098	0.000059	314 192.168.151.108	260 192.168.151.122	SMB		Trans2 Response, FIND_FIRST2, Files:test.pci
6.784142	0.000044	54 192.168.151.122	0 192.168.151.108	TCP	1041 > 445 [ACK] Seq=1212 Ack=4450 win=64240 Len=0	
9.690081	2.905939	1514 192.168.151.122	1460 192.168.151.108	TCP		[TCP segment of a reassembled PDU]
9.690373	0.000292	1514 192.168.151.122	1460 192.168.151.108	TCP		[TCP segment of a reassembled PDU]
9.690457	0.000084	1514 192.168.151.122	1460 192.168.151.108	TCP		[TCP segment of a reassembled PDU]
9.690536	0.000079	1514 192.168.151.122	1460 192.168.151.108	TCP		[TCP segment of a reassembled PDU]
9.690620	0.000084	1514 192.168.151.122	1460 192.168.151.108	TCP		[TCP segment of a reassembled PDU]
9.690698	0.000078	1514 192.168.151.122	1460 192.168.151.108	TCP		[TCP segment of a reassembled PDU]
9.690777	0.000079	1514 192.168.151.122	1460 192.168.151.108	TCP		[TCP segment of a reassembled PDU]
9.690854	0.000077	1514 192.168.151.122	1460 192.168.151.108	TCP		[TCP segment of a reassembled PDU]
9.690932	0.000078	1514 192.168.151.122	1460 192.168.151.108	TCP		[TCP segment of a reassembled PDU]
9.691010	0.000078	1514 192.168.151.122	1460 192.168.151.108	TCP		[TCP segment of a reassembled PDU]
9.691089	0.000079	1514 192.168.151.122	1460 192.168.151.108	TCP		[TCP segment of a reassembled PDU]
9.691170	0.000081	1514 192.168.151.122	1460 192.168.151.108	TCP		[TCP segment of a reassembled PDU]

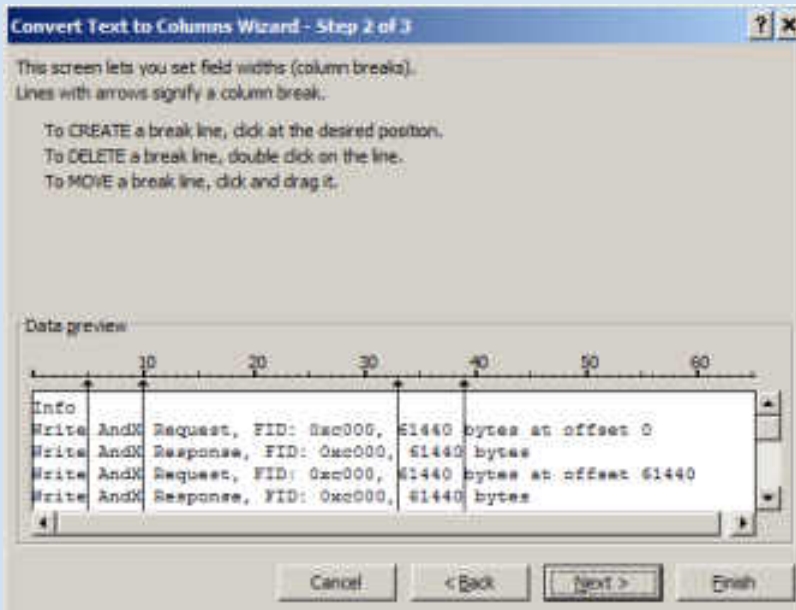
The Bytes in Flight Column shows us how much payload is in each packet.

TCP Inflight Data in Wireshark

No.	Time	Delta	Bytes	Source	Bytes in Flight	Destination	Protocol	Info
2	96	9.79559	0	242 192.168.1.1	188	192.168.1.5	MB	Write AndX Request, FID: 0xc000, 61440 bytes at offset 0
3	109	9.84588	0.05029	105 192.168.1.1	51	192.168.1.5	MB	Write AndX Response, FID: 0xc000, 61440 bytes
4	162	9.91585	0.06998	242 192.168.1.1	188	192.168.1.5	MB	Write AndX Request, FID: 0xc000, 61440 bytes at offset 61440
5	174	9.96369	0.04784	105 192.168.1.1	51	192.168.1.5	MB	Write AndX Response, FID: 0xc000, 61440 bytes
6	227	10.0333	0.06963	242 192.168.1.1	188	192.168.1.5	MB	Write AndX Request, FID: 0xc000, 61440 bytes at offset 122880
7	239	10.0822	0.04891	105 192.168.1.1	51	192.168.1.5	MB	Write AndX Response, FID: 0xc000, 61440 bytes
8	291	10.1495	0.0673	242 192.168.1.1	188	192.168.1.5	MB	Write AndX Request, FID: 0xc000, 61440 bytes at offset 184320
9	304	10.1974	0.04788	105 192.168.1.1	51	192.168.1.5	MB	Write AndX Response, FID: 0xc000, 61440 bytes
10	356	10.2653	0.06785	242 192.168.1.1	188	192.168.1.5	MB	Write AndX Request, FID: 0xc000, 61440 bytes at offset 245760
11	369	10.3201	0.0548	105 192.168.1.1	51	192.168.1.5	MB	Write AndX Response, FID: 0xc000, 61440 bytes
12	420	10.3872	0.06709	242 192.168.1.1	188	192.168.1.5	MB	Write AndX Request, FID: 0xc000, 61440 bytes at offset 307200
13	434	10.436	0.0489	105 192.168.1.1	51	192.168.1.5	MB	Write AndX Response, FID: 0xc000, 61440 bytes
14	485	10.5012	0.0652	242 192.168.1.1	188	192.168.1.5	MB	Write AndX Request, FID: 0xc000, 61440 bytes at offset 368640
15	499	10.5491	0.04783	105 192.168.1.1	51	192.168.1.5	MB	Write AndX Response, FID: 0xc000, 61440 bytes
16	550	10.6142	0.0651	242 192.168.1.1	188	192.168.1.5	MB	Write AndX Request, FID: 0xc000, 61440 bytes at offset 430080
17	564	10.6608	0.04659	105 192.168.1.1	51	192.168.1.5	MB	Write AndX Response, FID: 0xc000, 61440 bytes
18	614	10.7235	0.06269	242 192.168.1.1	188	192.168.1.5	MB	Write AndX Request, FID: 0xc000, 61440 bytes at offset 491520
19	629	10.7714	0.04797	105 192.168.1.1	51	192.168.1.5	MB	Write AndX Response, FID: 0xc000, 61440 bytes
20	679	10.8337	0.06232	242 192.168.1.1	188	192.168.1.5	MB	Write AndX Request, FID: 0xc000, 61440 bytes at offset 552960
21	694	10.8818	0.04804	105 192.168.1.1	51	192.168.1.5	MB	Write AndX Response, FID: 0xc000, 61440 bytes
22	744	10.9457	0.06391	242 192.168.1.1	188	192.168.1.5	MB	Write AndX Request, FID: 0xc000, 61440 bytes at offset 614400
23	759	10.9937	0.04797	105 192.168.1.1	51	192.168.1.5	MB	Write AndX Response, FID: 0xc000, 61440 bytes
24	808	11.054	0.06031	242 192.168.1.1	188	192.168.1.5	MB	Write AndX Request, FID: 0xc000, 61440 bytes at offset 675840
25	824	11.102	0.04803	105 192.168.1.1	51	192.168.1.5	MB	Write AndX Response, FID: 0xc000, 61440 bytes
26	873	11.162	0.06005	242 192.168.1.1	188	192.168.1.5	MB	Write AndX Request, FID: 0xc000, 61440 bytes at offset 737280
27	889	11.2088	0.04671	105 192.168.1.1	51	192.168.1.5	MB	Write AndX Response, FID: 0xc000, 61440 bytes

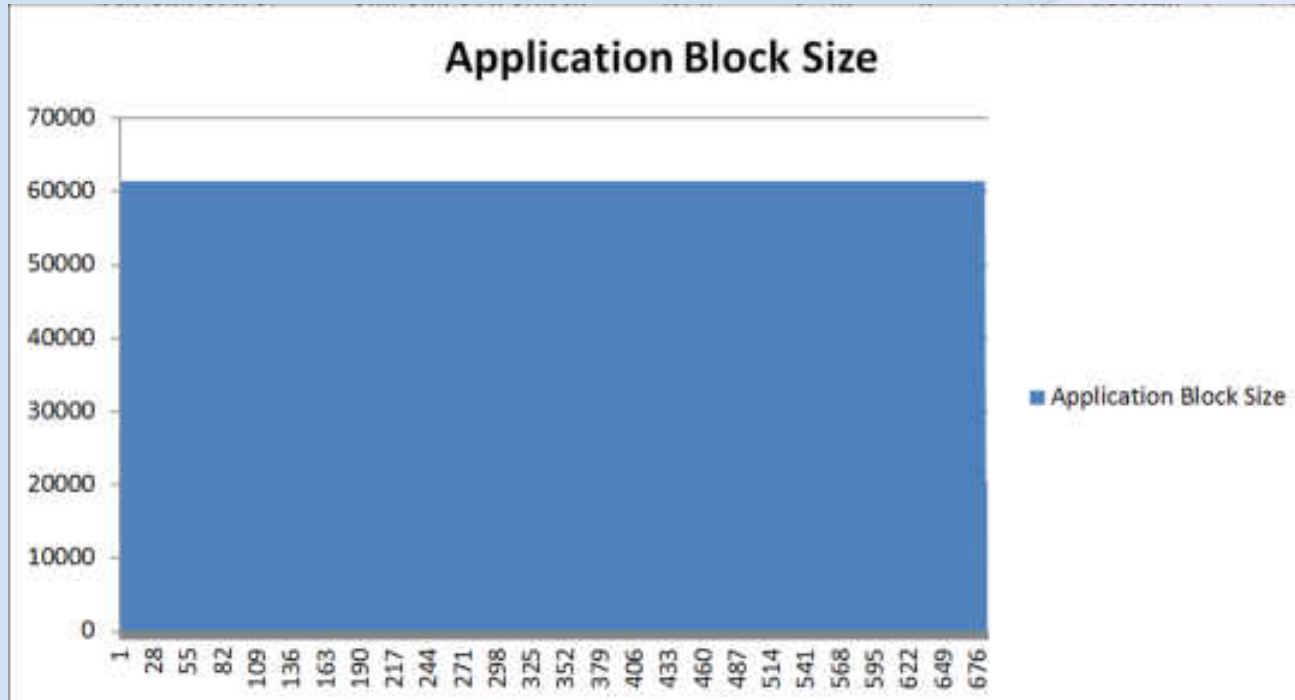
The Bytes in Flight Column shows us how much payload is in each packet.

TCP Inflight Data in Wireshark



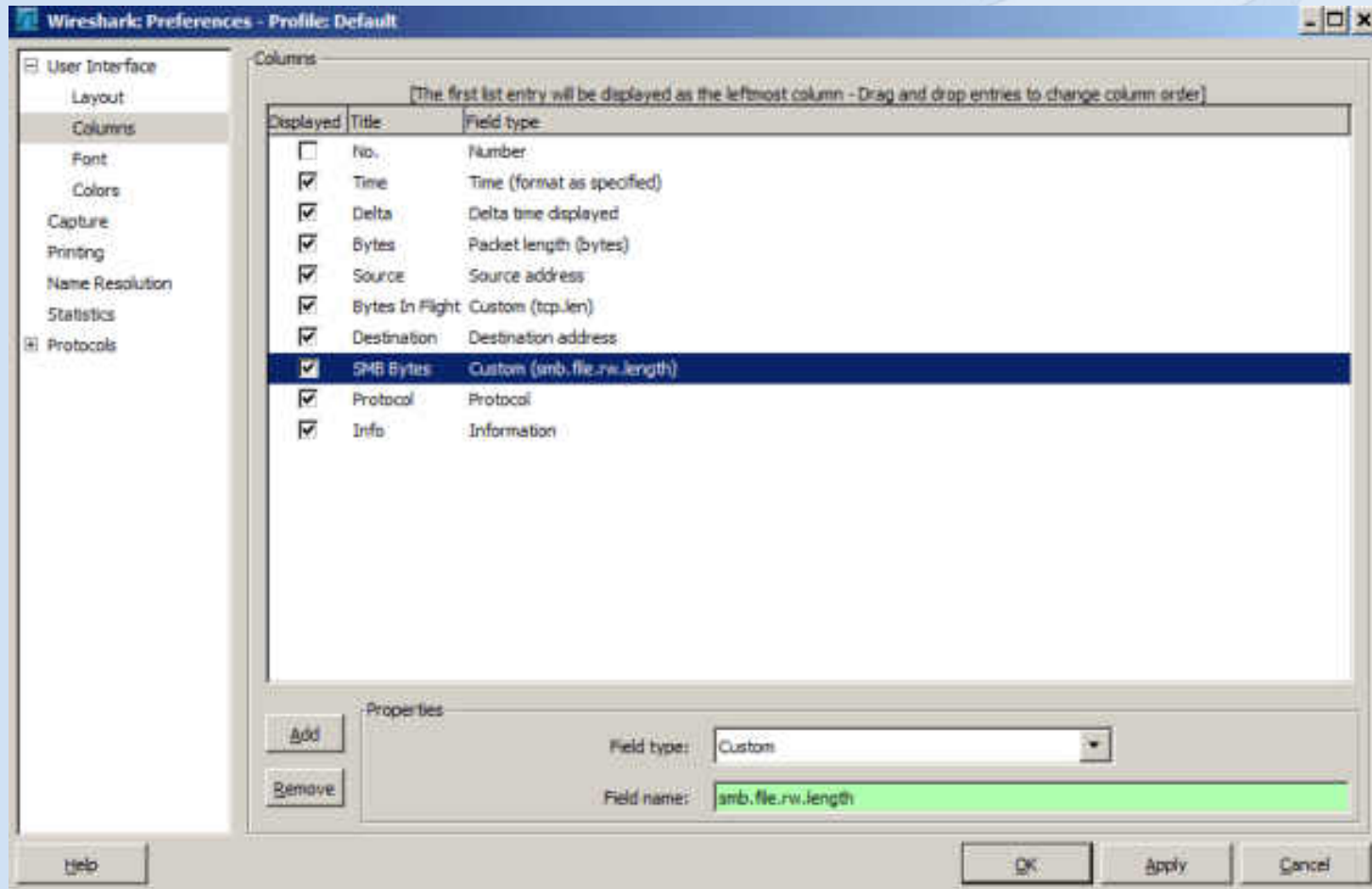
No.	Time	Delta	Bytes	Source	Bytes in FID	Destination	Protocol	Info
1	98.979550	0	242	192.168.1.1	188	192.168.1.100	SMB	Write AndX Request, FID: 0xc000, 61440 bytes at offset 0
2	109.984588	0.01029	105	192.168.1.1	51	192.168.1.100	SMB	Write AndX Response, FID: 0xc000, 61440 bytes
3	162.9.91385	0.04998	242	192.168.1.1	188	192.168.1.100	SMB	Write AndX Request, FID: 0xc000, 61440 bytes at offset 61440
4	174.9.96369	0.04784	105	192.168.1.1	51	192.168.1.100	SMB	Write AndX Response, FID: 0xc000, 61440 bytes
5	227.10.0353	0.06963	242	192.168.1.1	188	192.168.1.100	SMB	Write AndX Request, FID: 0xc000, 61440 bytes at offset 122880
6	239.10.0422	0.04891	105	192.168.1.1	51	192.168.1.100	SMB	Write AndX Response, FID: 0xc000, 61440 bytes
7	291.10.1485	0.0673	242	192.168.1.1	188	192.168.1.100	SMB	Write AndX Request, FID: 0xc000, 61440 bytes at offset 184320
8	304.10.1974	0.04788	105	192.168.1.1	51	192.168.1.100	SMB	Write AndX Response, FID: 0xc000, 61440 bytes
9	356.10.2653	0.06785	242	192.168.1.1	188	192.168.1.100	SMB	Write AndX Request, FID: 0xc000, 61440 bytes at offset 245760
10	369.10.3201	0.0548	105	192.168.1.1	51	192.168.1.100	SMB	Write AndX Response, FID: 0xc000, 61440 bytes
11	420.10.3872	0.06709	242	192.168.1.1	188	192.168.1.100	SMB	Write AndX Request, FID: 0xc000, 61440 bytes at offset 307200
12	434.10.436	0.0489	105	192.168.1.1	51	192.168.1.100	SMB	Write AndX Response, FID: 0xc000, 61440 bytes
13	485.10.5012	0.0652	242	192.168.1.1	188	192.168.1.100	SMB	Write AndX Request, FID: 0xc000, 61440 bytes at offset 368640

TCP Inflight Data in Wireshark



Graphed in Excel

Easier way for SMB/CIFS



TCP Retransmissions

- Every time a TCP segment is sent, a retransmission timer is started.
- When the Acknowledgement for that segment is received the timer is stopped.
- If the retransmission timer expires before the Acknowledgement is received, the TCP segment is retransmitted.

tcp

TCP Retransmissions

tcp flow

- Excessive TCP Retransmissions can have a huge impact on application performance.
- Not only does the data have to get resent, but TCP flow control (Slow Start) kicks into action.

Application Performance

demo

Layer 7 Bottlenecks

ULPs (upper layer protocols)

- TCP often gets blamed for the ULPs problem.
 - The application hands down to TCP amount of data to go retrieve (application block size)
 - TCP then is responsible for reliably getting that data back to the application layer
 - TCP has certain parameters in which to work with and can usually be tuned based on bandwidth and latency
 - Many times too much focus is put on “tuning” TCP as the fix for poor performance in the network
- If the TCP advertised receive window is set to 64K and the application is only handing down to TCP requests for 16K, where is the bottleneck?

ulp

ULPs (upper layer protocols)

Case in point: CIFS/SMB

ulps

Troubleshooting CIFS/SMB

cifs/smb

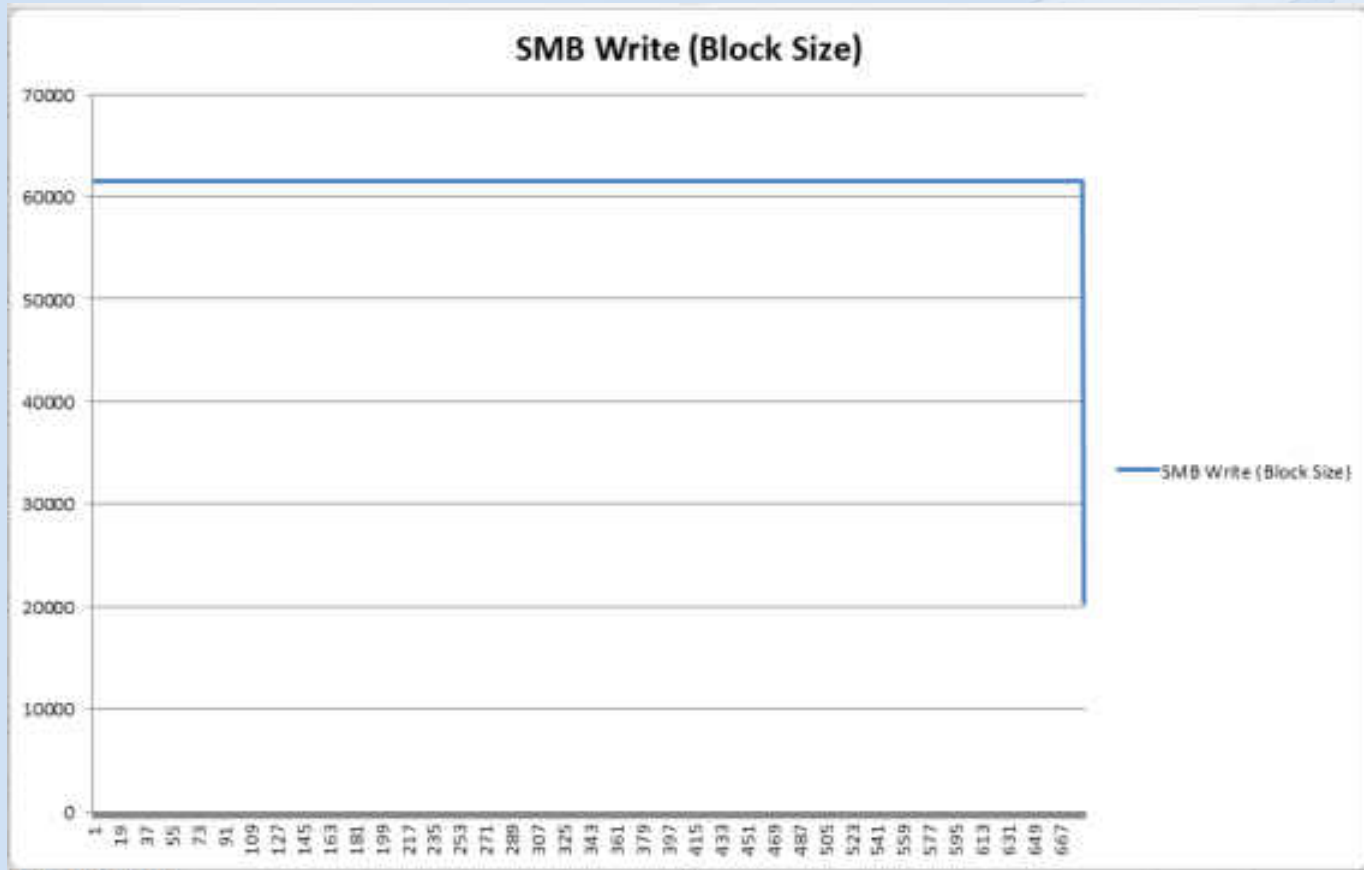
- Arguably the most common File Transfer method used in businesses today.
- SMB was NOT developed with the WAN in mind.
- One of the most “chatty” protocols/ applications I run into (with the exception of poorly written SQL).

CIFS/SMB Quiz

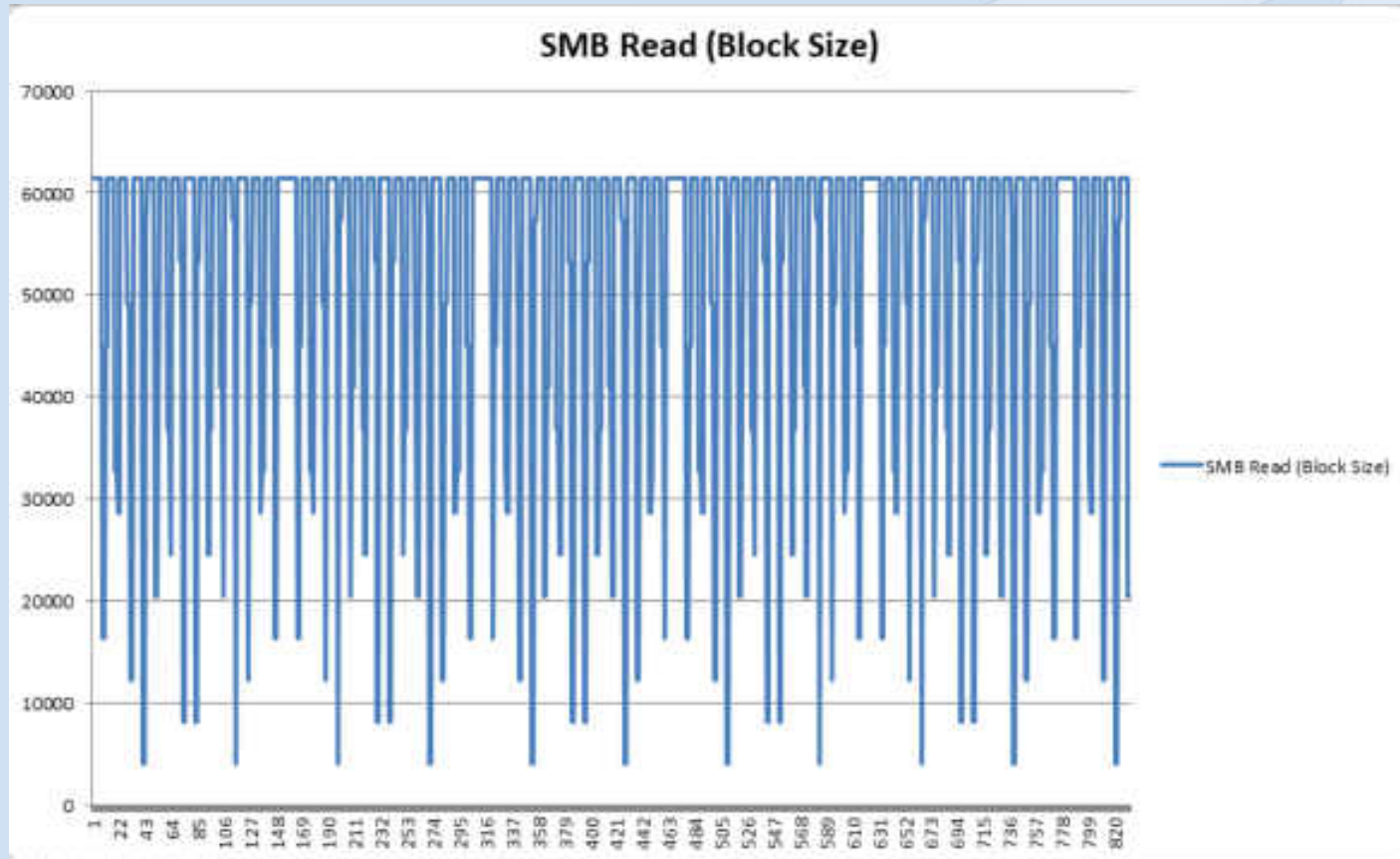
- What is faster using MS File Sharing?
 - Pushing a file to a file server?
 - Pulling a file from a file server?

quiz

ULPs (upper layer protocols)



ULPs (upper layer protocols)



CIFS/SMB

cifs/smb

- What is faster using MS File Sharing?
 - Pushing a file to a file server?
 - Pulling a file from a file server?
 - SMB Write (Pushing the file) can almost be 2X as fast as pulling (SMB Read)
 - Depends on the Latency

CIFS/SMB Tuning

- SMB Maximum Transmit Buffer Size
 - Negotiated MaxBufferSize in the Negotiate Protocol response
 - Default for Windows servers is typically 16644 (dependent upon physical memory)
 - Client default typically 4356

tuning

CIF/SMB Tuning

The screenshot shows the Windows Registry Editor window. The left pane displays a tree view of the registry, with 'Parameters' expanded under 'Lanman\Workstation'. The right pane shows a list of registry values for 'Parameters'. A dialog box titled 'Edit DWORD (32-bit) Value' is open, showing the 'SizeRegBuf' value being edited to '0x' in hexadecimal format.

Name	Type	Data
(Default)	REG_SZ	(value not set)
AdjustFullSessionPipes	REG_DWORD	0x00000003 (3)
autoconnect	REG_DWORD	0x0000000F (15)
EnableAuthenticateUserSharing	REG_DWORD	0x00000000 (0)
enableforcedlogoff	REG_DWORD	0x00000001 (1)
enablesecuritysignature	REG_DWORD	0x00000000 (0)
Guid	REG_BINARY	09 a2 a6 47 5c e4 3d 4a 98 58 0b 7b 35 c9 c4 04
Unannounce	REG_DWORD	0x00000000 (0)
FullSessionPipes	REG_MULTI_SZ	
requiresecuritysignature	REG_DWORD	0x00000000 (0)
restrictfullaccess	REG_DWORD	0x00000001 (1)
ServiceDll	REG_EXPAND_SZ	%SystemRoot%\system32\svchost.dll
ServiceDll,InloadOrStop	REG_DWORD	0x00000001 (1)
Size	REG_DWORD	0x00000001 (1)
SizeRegBuf	REG_DWORD	0x00000000 (0)

Dialog Box: Edit DWORD (32-bit) Value

Value name: SizeRegBuf

Value data: 0x

Base: Hexadecimal Decimal

Buttons: OK, Cancel

CIFS/SMB Tuning

- Caveat:
 - SMB is extremely dependent upon the API
 - Even though you set the max buffer size to 64K, windows “share” data will always get truncated to 60K (61440) even though the server can support 64K

tuning

CIFS/SMB Tuning

- Custom SMB APIs
 - The Windows limitation can be exceeded by programs written to use SMB as they file transfer protocol

tuning

CIFS/SMB Tuning

Time	Length	Offset	Operation	SMB Type	Task
5.072671	0.047837	454 192.168.151.108	500 192.168.151.122	SMB	TRANS response, FINU_FINAL, FYIES: ... test.pcap.index 2000ops.pcap 000ops.pcap and
6.231794	0.359123	1418 192.168.151.122	1364 192.168.151.108	65536 SMB	write AndX Request, FID: 0x400b, 65536 bytes at offset 0
6.279478	0.047884	105 192.168.151.108	51 192.168.151.122	65536 SMB	write AndX Response, FID: 0x400b, 65536 bytes
6.373990	0.094512	1418 192.168.151.122	1364 192.168.151.108	65536 SMB	write AndX Request, FID: 0x400b, 65536 bytes at offset 65536
6.420427	0.046437	105 192.168.151.108	51 192.168.151.122	65536 SMB	write AndX Response, FID: 0x400b, 65536 bytes
6.492078	0.071651	1418 192.168.151.122	1364 192.168.151.108	65536 SMB	write AndX Request, FID: 0x400b, 65536 bytes at offset 131072
6.540968	0.048890	105 192.168.151.108	51 192.168.151.122	65536 SMB	write AndX Response, FID: 0x400b, 65536 bytes
6.612075	0.071107	1418 192.168.151.122	1364 192.168.151.108	65536 SMB	write AndX Request, FID: 0x400b, 65536 bytes at offset 196608
6.659702	0.047627	105 192.168.151.108	51 192.168.151.122	65536 SMB	write AndX Response, FID: 0x400b, 65536 bytes
6.730775	0.071073	1418 192.168.151.122	1364 192.168.151.108	65536 SMB	write AndX Request, FID: 0x400b, 65536 bytes at offset 262144
6.777258	0.046483	105 192.168.151.108	51 192.168.151.122	65536 SMB	write AndX Response, FID: 0x400b, 65536 bytes
6.859852	0.082594	1418 192.168.151.122	1364 192.168.151.108	65536 SMB	write AndX Request, FID: 0x400b, 65536 bytes at offset 327680
6.909832	0.049980	105 192.168.151.108	51 192.168.151.122	65536 SMB	write AndX Response, FID: 0x400b, 65536 bytes
6.981768	0.071936	1418 192.168.151.122	1364 192.168.151.108	65536 SMB	write AndX Request, FID: 0x400b, 65536 bytes at offset 393216
7.028324	0.046556	105 192.168.151.108	51 192.168.151.122	65536 SMB	write AndX Response, FID: 0x400b, 65536 bytes
7.094564	0.066240	1418 192.168.151.122	1364 192.168.151.108	65536 SMB	write AndX Request, FID: 0x400b, 65536 bytes at offset 458752
7.142103	0.047539	105 192.168.151.108	51 192.168.151.122	65536 SMB	write AndX Response, FID: 0x400b, 65536 bytes
7.208186	0.066083	1418 192.168.151.122	1364 192.168.151.108	65536 SMB	write AndX Request, FID: 0x400b, 65536 bytes at offset 524288
7.255740	0.047554	105 192.168.151.108	51 192.168.151.122	65536 SMB	write AndX Response, FID: 0x400b, 65536 bytes
7.327783	0.072043	1418 192.168.151.122	1364 192.168.151.108	65536 SMB	write AndX Request, FID: 0x400b, 65536 bytes at offset 589824
7.375325	0.047542	105 192.168.151.108	51 192.168.151.122	65536 SMB	write AndX Response, FID: 0x400b, 65536 bytes
7.439171	0.063846	1418 192.168.151.122	1364 192.168.151.108	65536 SMB	write AndX Request, FID: 0x400b, 65536 bytes at offset 655360
7.487444	0.048273	105 192.168.151.108	51 192.168.151.122	65536 SMB	write AndX Response, FID: 0x400b, 65536 bytes
7.551070	0.063826	1418 192.168.151.122	1364 192.168.151.108	65536 SMB	write AndX Request, FID: 0x400b, 65536 bytes at offset 720896
7.597565	0.046495	105 192.168.151.108	51 192.168.151.122	65536 SMB	write AndX Response, FID: 0x400b, 65536 bytes
7.659311	0.061746	1418 192.168.151.122	1364 192.168.151.108	65536 SMB	write AndX Request, FID: 0x400b, 65536 bytes at offset 786432
7.708128	0.048837	105 192.168.151.108	51 192.168.151.122	65536 SMB	write AndX Response, FID: 0x400b, 65536 bytes

Note the SMB writes of 65,536

This is a file transfer using a custom API on a Windows XP machine

CIFS/SMB Tuning (Preallocation)

The screenshot shows a Wireshark capture of SMB traffic. The top pane lists 27 packets, including Echo Responses, Trans2 Requests (SET_FILE_INFO, FIND_FIRST2), and Trans2 Responses. The bottom pane provides a detailed view of a SET_FILE_INFO packet (packet 14), showing its structure and a hex dump of the data.

```
Timeout: Return Immediately (0)
Reserved: 0000
Parameter Count: 6
Parameter Offset: 68
Data Count: 8
Data Offset: 76
Setup Count: 1
Reserved: 00
Subcommand: SET_FILE_INFO (0x0006)
Byte Count (BCC): 18
Padding: 000000
= SET_FILE_INFO Parameters
0000 00 1a ad 98 b3 ad 00 0c 29 e7 7a 46 08 00 45 00 .....J.ZF.E.
0010 00 80 3c e7 40 00 80 06 1d 59 c0 a8 97 7a c0 a8 .....W...Y...Z..
0020 97 4c 04 11 01 bd be a1 8f 06 71 0d 5a f6 50 18 .....T...Z.P.
0030 fa bb 26 f1 00 00 00 00 00 54 ff 33 4d 42 32 00 .....A.....T.SMB2.
0040 00 00 00 18 07 c8 00 00 00 00 00 00 00 00 00 00 .....
0050 00 00 01 08 40 0e 01 08 41 1e 0f 06 00 08 00 02 .....@...A.....
0060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 44 .....D
0070 00 08 00 4c 00 01 00 08 00 13 00 00 00 00 08 40 .....L.....W
0080 fc 03 00 00 00 00 00 00 ad 00 00 00 00 00 .....
```

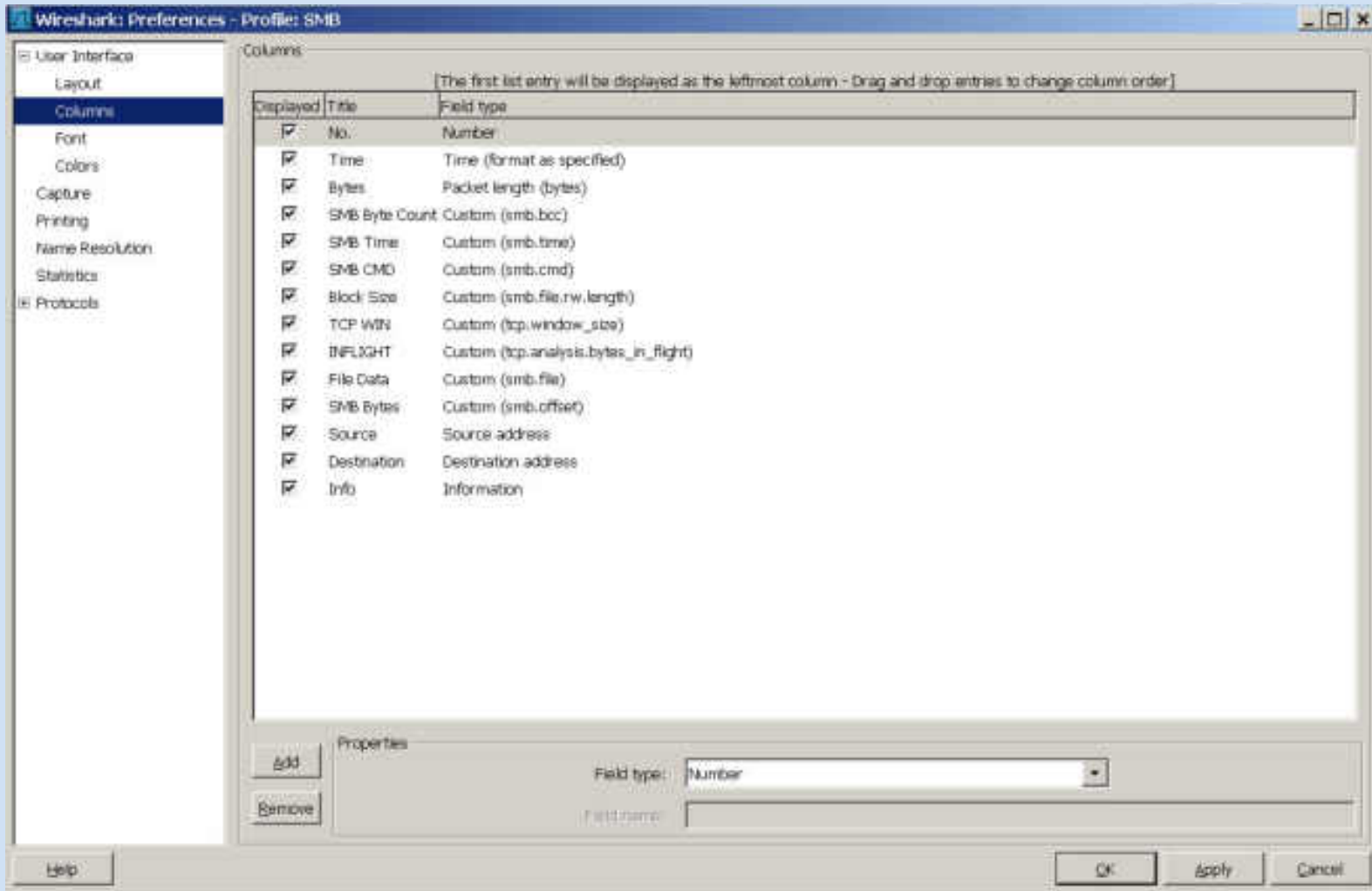
Preallocation sets the file info for SMB Writes and can drastically reduce some of the “chattiness” of SMB

Instructor Demo of SMB Profiles

demo

Demo of SMB Tracefiles

My personal SMB Profile



Take Away Points

- Building your own CDA is easy to do and may fit in a majority of the areas you need to capture from
- Pilot, Pilot, Pilot, it's not just a fancy reporting engine for Wireshark!
- Test your applications “Networkability” before they hit production.
- Use the Wireshark Profiles, they will save you a ton of time.

points

SHARKFEST '12

Wireshark Developer and User Conference

Mike Canney

Principal Network Analyst

Tektivity™