Mike Canney

Application Performance Analysis

Tektivity™
Welcome to Sharkfest ‘12

Mike Canney,
Principal Network Analyst, Tektivity, Inc.

canney@getpackets.com
319-365-3336

www.getpackets.com
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).
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.
Commercial vs. Free 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)
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?

- 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?

- 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.
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.
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.
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
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.
Top Causes for Poor Application Performance

- Application Turns
- TCP
- Layer 7 Bottlenecks
- Congestion (network)
- Processing Delay
Causes for Slow Application Performance

Application 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).
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: \texttt{smb.cmd}

<table>
<thead>
<tr>
<th>Time</th>
<th>Diffs</th>
<th>Bytes</th>
<th>Source</th>
<th>Destination</th>
<th>Protocol</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.008388</td>
<td>0.096607</td>
<td>237 192.168.151.108</td>
<td>183 192.168.151.122</td>
<td>SMB Read AndX Response, FID: 0xc00d, 61440 bytes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10.008935</td>
<td>0.000547</td>
<td>117 192.168.151.122</td>
<td>63 192.168.151.108</td>
<td>SMB Read AndX Request, FID: 0xc00d, 61440 bytes at offset 61440</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10.105423</td>
<td>0.096488</td>
<td>237 192.168.151.108</td>
<td>183 192.168.151.122</td>
<td>SMB Read AndX Request, FID: 0xc00d, 61440 bytes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10.365972</td>
<td>0.000549</td>
<td>117 192.168.151.122</td>
<td>63 192.168.151.108</td>
<td>SMB Read AndX Response, FID: 0xc00d, 61440 bytes at offset 122880</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10.702297</td>
<td>0.096325</td>
<td>237 192.168.151.108</td>
<td>183 192.168.151.122</td>
<td>SMB Read AndX Request, FID: 0xc00d, 61440 bytes at offset 122880</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10.20691</td>
<td>0.000394</td>
<td>117 192.168.151.122</td>
<td>63 192.168.151.108</td>
<td>SMB Read AndX Response, FID: 0xc00d, 61440 bytes at offset 184320</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10.299228</td>
<td>0.096337</td>
<td>237 192.168.151.108</td>
<td>183 192.168.151.122</td>
<td>SMB Read AndX Response, FID: 0xc00d, 61440 bytes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10.299569</td>
<td>0.000341</td>
<td>117 192.168.151.122</td>
<td>63 192.168.151.108</td>
<td>SMB Read AndX Request, FID: 0xc00d, 16384 bytes at offset 245760</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10.358427</td>
<td>0.058858</td>
<td>441 192.168.151.108</td>
<td>387 192.168.151.122</td>
<td>SMB Read AndX Response, FID: 0xc00d, 16384 bytes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10.358662</td>
<td>0.000235</td>
<td>117 192.168.151.122</td>
<td>63 192.168.151.108</td>
<td>SMB Read AndX Request, FID: 0xc00d, 45056 bytes at offset 262144</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10.444337</td>
<td>0.082675</td>
<td>1373 192.168.151.108</td>
<td>1319 192.168.151.122</td>
<td>SMB Read AndX Response, FID: 0xc00d, 45056 bytes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10.445493</td>
<td>0.004156</td>
<td>117 192.168.151.122</td>
<td>63 192.168.151.108</td>
<td>SMB Read AndX Request, FID: 0xc00d, 61440 bytes at offset 307200</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10.541826</td>
<td>0.096333</td>
<td>237 192.168.151.108</td>
<td>183 192.168.151.122</td>
<td>SMB Read AndX Response, FID: 0xc00d, 61440 bytes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10.542163</td>
<td>0.000337</td>
<td>117 192.168.151.122</td>
<td>63 192.168.151.108</td>
<td>SMB Read AndX Request, FID: 0xc00d, 61440 bytes at offset 368640</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10.638633</td>
<td>0.096470</td>
<td>237 192.168.151.108</td>
<td>183 192.168.151.122</td>
<td>SMB Read AndX Response, FID: 0xc00d, 61440 bytes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10.639010</td>
<td>0.000377</td>
<td>117 192.168.151.122</td>
<td>63 192.168.151.108</td>
<td>SMB Read AndX Request, FID: 0xc00d, 61440 bytes at offset 430080</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10.735377</td>
<td>0.096367</td>
<td>237 192.168.151.108</td>
<td>183 192.168.151.122</td>
<td>SMB Read AndX Response, FID: 0xc00d, 61440 bytes</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

882 Application Turns in this trace
App Turns and 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 turns * .050 ms delay = 500 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)
Causes for Slow Application Performance

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).
TCP Inflight Data

- The amount of unacknowledged TCP data that is on the wire at any given time.
- TCP inflight data in limited by the following:
  - TCP Retransmissions
  - TCP Window Size
  - Application block size
- The amount of TCP inflight data will never exceed the receiving devices advertised TCP Window Size.
TCP Inflight Data in Wireshark
## TCP Inflight Data in Wireshark

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

<table>
<thead>
<tr>
<th>Seq</th>
<th>D porta</th>
<th>Bytes</th>
<th>Source</th>
<th>Destination</th>
<th>Protocol</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.88815</td>
<td>0.000170</td>
<td>120 192.168.151.106</td>
<td>192.168.151.108</td>
<td>70 192.168.151.108</td>
<td>SMB</td>
<td>NT TRANS RESPONSE, NT NOTIFY</td>
</tr>
<tr>
<td>4.888215</td>
<td>0.000064</td>
<td>54 192.168.151.108</td>
<td>0 192.168.151.108</td>
<td>TCP</td>
<td>1041 &gt; 445 [ACK] Seq=682 Ack=2514 Win=63969 Len=0</td>
<td></td>
</tr>
<tr>
<td>4.888499</td>
<td>0.000284</td>
<td>142 192.168.151.108</td>
<td>88 192.168.151.108</td>
<td>SMB</td>
<td>Trans2 Request, SET_FILE_INFO, FID: 0xc000</td>
<td></td>
</tr>
<tr>
<td>4.888828</td>
<td>0.000329</td>
<td>142 192.168.151.108</td>
<td>88 192.168.151.108</td>
<td>SMB</td>
<td>NT Trans Request, NT NOTIFY, FID: 0x4005</td>
<td></td>
</tr>
<tr>
<td>4.93909</td>
<td>0.045081</td>
<td>118 192.168.151.108</td>
<td>64 192.168.151.122</td>
<td>SMB</td>
<td>Trans2 Response, FID: 0xc000, SET_FILE_INFO</td>
<td></td>
</tr>
<tr>
<td>4.934090</td>
<td>0.000181</td>
<td>130 192.168.151.108</td>
<td>76 192.168.151.122</td>
<td>SMB</td>
<td>NT Trans Response, NT NOTIFY</td>
<td></td>
</tr>
<tr>
<td>4.934149</td>
<td>0.000059</td>
<td>54 192.168.151.108</td>
<td>0 192.168.151.108</td>
<td>TCP</td>
<td>1041 &gt; 445 [ACK] Seq=858 Ack=2654 Win=63829 Len=0</td>
<td></td>
</tr>
<tr>
<td>4.939393</td>
<td>0.005244</td>
<td>142 192.168.151.108</td>
<td>88 192.168.151.108</td>
<td>SMB</td>
<td>NT Trans Request, NT NOTIFY, FID: 0x4005</td>
<td></td>
</tr>
<tr>
<td>5.197543</td>
<td>0.258150</td>
<td>60 192.168.151.108</td>
<td>0 192.168.151.108</td>
<td>TCP</td>
<td>445 &gt; 1041 [ACK] Seq=2654 Ack=946 Win=63663 Len=0</td>
<td></td>
</tr>
<tr>
<td>5.197735</td>
<td>0.000192</td>
<td>142 192.168.151.108</td>
<td>88 192.168.151.108</td>
<td>SMB</td>
<td>NT Trans Request, NT NOTIFY, FID: 0x4006</td>
<td></td>
</tr>
<tr>
<td>5.243042</td>
<td>0.045307</td>
<td>130 192.168.151.108</td>
<td>76 192.168.151.122</td>
<td>SMB</td>
<td>NT Trans Response, NT NOTIFY</td>
<td></td>
</tr>
<tr>
<td>5.243344</td>
<td>0.000302</td>
<td>142 192.168.151.108</td>
<td>88 192.168.151.108</td>
<td>SMB</td>
<td>NT Trans Request, NT NOTIFY, FID: 0x4006</td>
<td></td>
</tr>
<tr>
<td>5.506404</td>
<td>0.263050</td>
<td>60 192.168.151.108</td>
<td>0 192.168.151.108</td>
<td>TCP</td>
<td>445 &gt; 1041 [ACK] Seq=2730 Ack=1122 Win=63487 Len=0</td>
<td></td>
</tr>
<tr>
<td>6.784039</td>
<td>0.047480</td>
<td>1514 192.168.151.108</td>
<td>1460 192.168.151.122</td>
<td>TCP</td>
<td>[TCP segment of a reassembled PDU]</td>
<td></td>
</tr>
<tr>
<td>6.784142</td>
<td>0.000044</td>
<td>54 192.168.151.108</td>
<td>0 192.168.151.108</td>
<td>TCP</td>
<td>1041 &gt; 445 [ACK] Seq=1212 Ack=4450 Win=64240 Len=0</td>
<td></td>
</tr>
<tr>
<td>6.690373</td>
<td>0.000292</td>
<td>1514 192.168.151.108</td>
<td>1460 192.168.151.108</td>
<td>TCP</td>
<td>[TCP segment of a reassembled PDU]</td>
<td></td>
</tr>
<tr>
<td>6.690457</td>
<td>0.000084</td>
<td>1514 192.168.151.108</td>
<td>1460 192.168.151.108</td>
<td>TCP</td>
<td>[TCP segment of a reassembled PDU]</td>
<td></td>
</tr>
<tr>
<td>6.690536</td>
<td>0.000079</td>
<td>1514 192.168.151.108</td>
<td>1460 192.168.151.108</td>
<td>TCP</td>
<td>[TCP segment of a reassembled PDU]</td>
<td></td>
</tr>
<tr>
<td>6.690620</td>
<td>0.000084</td>
<td>1514 192.168.151.108</td>
<td>1460 192.168.151.108</td>
<td>TCP</td>
<td>[TCP segment of a reassembled PDU]</td>
<td></td>
</tr>
<tr>
<td>6.690698</td>
<td>0.000078</td>
<td>1514 192.168.151.108</td>
<td>1460 192.168.151.108</td>
<td>TCP</td>
<td>[TCP segment of a reassembled PDU]</td>
<td></td>
</tr>
<tr>
<td>6.690777</td>
<td>0.000079</td>
<td>1514 192.168.151.108</td>
<td>1460 192.168.151.108</td>
<td>TCP</td>
<td>[TCP segment of a reassembled PDU]</td>
<td></td>
</tr>
<tr>
<td>6.690854</td>
<td>0.000077</td>
<td>1514 192.168.151.108</td>
<td>1460 192.168.151.108</td>
<td>TCP</td>
<td>[TCP segment of a reassembled PDU]</td>
<td></td>
</tr>
<tr>
<td>6.690932</td>
<td>0.000078</td>
<td>1514 192.168.151.108</td>
<td>1460 192.168.151.108</td>
<td>TCP</td>
<td>[TCP segment of a reassembled PDU]</td>
<td></td>
</tr>
<tr>
<td>6.691010</td>
<td>0.000078</td>
<td>1514 192.168.151.108</td>
<td>1460 192.168.151.108</td>
<td>TCP</td>
<td>[TCP segment of a reassembled PDU]</td>
<td></td>
</tr>
<tr>
<td>6.691089</td>
<td>0.000079</td>
<td>1514 192.168.151.108</td>
<td>1460 192.168.151.108</td>
<td>TCP</td>
<td>[TCP segment of a reassembled PDU]</td>
<td></td>
</tr>
<tr>
<td>6.691170</td>
<td>0.000081</td>
<td>1514 192.168.151.108</td>
<td>1460 192.168.151.108</td>
<td>TCP</td>
<td>[TCP segment of a reassembled PDU]</td>
<td></td>
</tr>
</tbody>
</table>
The Bytes in Flight Column shows us how much payload is in each packet.
TCP Inflight Data in Wireshark
TCP Inflight Data in Wireshark

Application Block Size

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 Retransmissions

• 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.
Layer 7 Bottlenecks
UPLs (upper layer protocols)

- TCP often gets blamed for the UPLs 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?
ULPs (upper layer protocols)

Case in point: CIFS/SMB
Troubleshooting 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).
What is faster using MS File Sharing?
- Pushing a file to a file server?
- Pulling a file from a file server?
ULPs (upper layer protocols)
ULPs (upper layer protocols)
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
CIF/SMB Tuning
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
CIFS/SMB Tuning

• Custom SMB APIs

  - The Windows limitation can be exceeded by programs written to use SMB as they file transfer protocol
CIFS/SMB Tuning

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)

Preallocation sets the file info for SMB Writes and can drastically reduce some of the “ chattyness” of SMB
Demo of SMB Tracefiles
My personal SMB Profile

![Screen shot of Wireshark Preferences - Profile: SMB]

- User Interface
  - Layout
- Columns
  - Font
  - Colors
  - Capture
  - Printing
  - Name Resolution
  - Statistics
  - Protocols
- Properties
  - Field type: Number
  - Field name: 

The first list entry will be displayed as the leftmost column - Drag and drop entries to change column order.
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.
SHARKFEST ‘12
Wireshark Developer and User Conference

Mike Canney
Principal Network Analyst
Tektivity™