



# SHARKFEST '14

WIRESHARK DEVELOPER AND USER CONFERENCE

JUNE 16-20 2014 · DOMINICAN UNIVERSITY

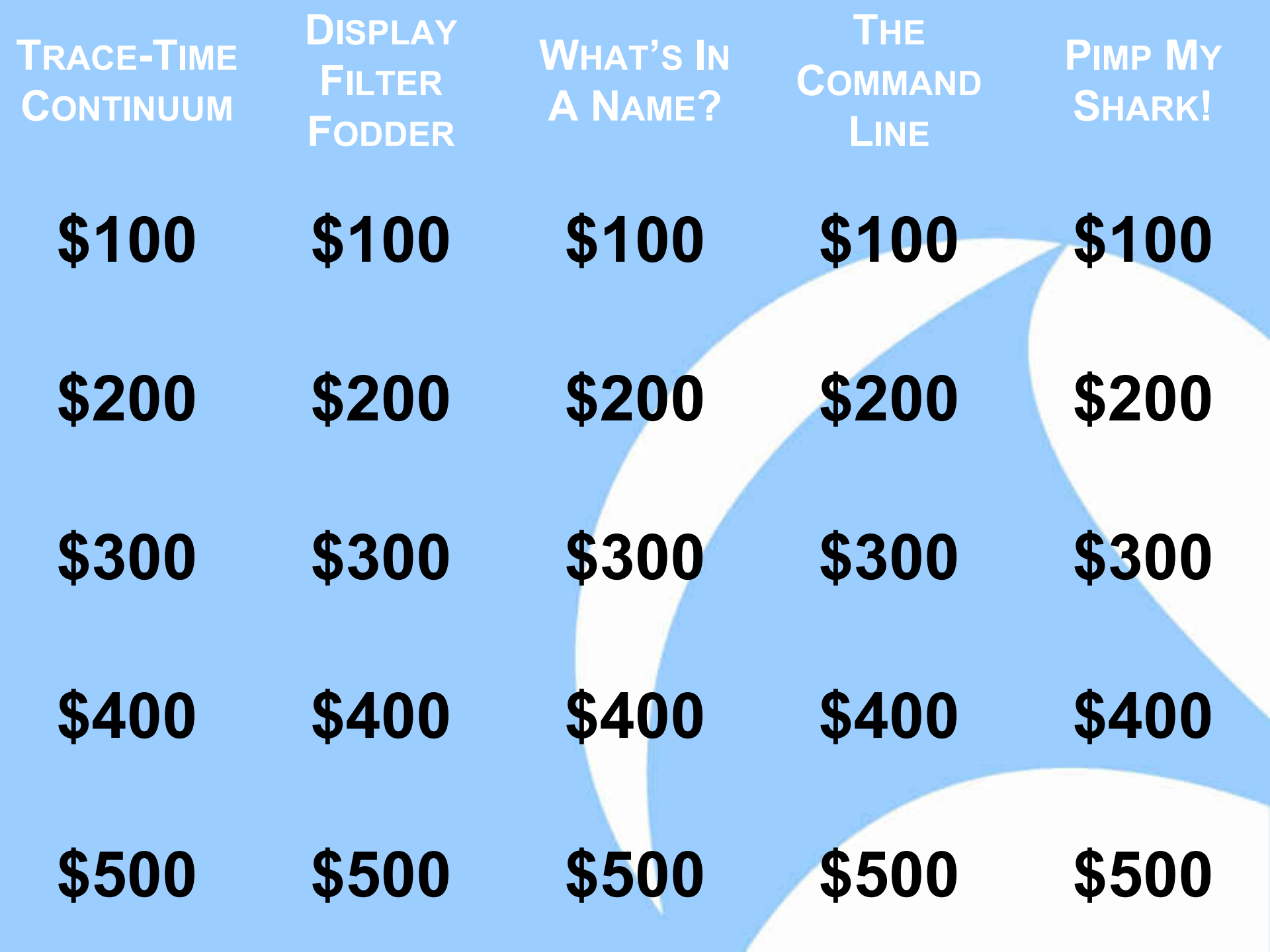
## Sharkfest Jeopardy!

Robert Bullen

Application Performance Engineer  
Blue Cross Blue Shield of Minnesota  
[robert\\_d\\_bullen@bluecrossmn.com](mailto:robert_d_bullen@bluecrossmn.com)

A large, stylized orange shark fin graphic is positioned on the left side of the text. The fin is curved upwards and to the right, with a small hook-like detail at its base. The background features a light blue sky with white, curved shapes representing waves or clouds.

**J SHARK FEST  
JEOPARDY!**



TRACE-TIME CONTINUUM	DISPLAY FILTER FODDER	WHAT'S IN A NAME?	THE COMMAND LINE	PIMP MY SHARK!
<b>\$100</b>	<b>\$100</b>	<b>\$100</b>	<b>\$100</b>	<b>\$100</b>
<b>\$200</b>	<b>\$200</b>	<b>\$200</b>	<b>\$200</b>	<b>\$200</b>
<b>\$300</b>	<b>\$300</b>	<b>\$300</b>	<b>\$300</b>	<b>\$300</b>
<b>\$400</b>	<b>\$400</b>	<b>\$400</b>	<b>\$400</b>	<b>\$400</b>
<b>\$500</b>	<b>\$500</b>	<b>\$500</b>	<b>\$500</b>	<b>\$500</b>



# Trace-Time Continuum \$100

## Question

By calculating the delta time between the first and third packets of the TCP handshake (the SYN and ACK, respectively), which time-based metric of a conversation are you estimating?

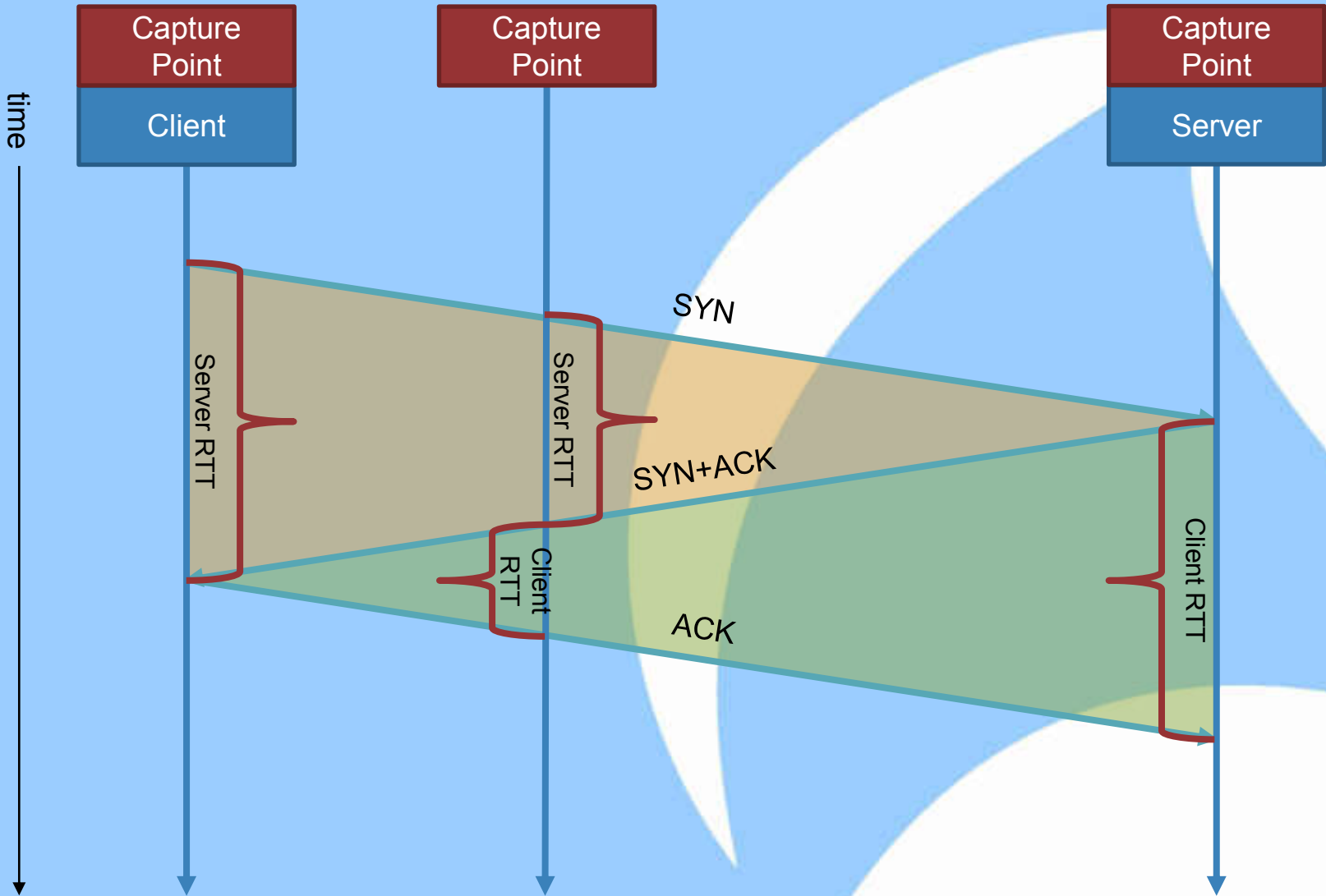
# Trace-Time Continuum \$100

## Answer

The network round-trip time (RTT).

The nice thing about using the handshake is that TCP implementations try to conduct the handshake as quickly as possible (many NICs implement it in hardware these days), so there is typically negligible endpoint processing delay factored into the resulting RTT estimation.

# Trace-Time Continuum \$100 Diagram



# Trace-Time Continuum \$200

## Question

If the TCP handshake is not captured, how can you use Wireshark to estimate the network round-trip time?

# Trace-Time Continuum \$200

## Answer Part 1

One way is to display two of Wireshark's built-in RTT graphs (Statistics | TCP StreamGraph | Round Trip Time Graph).

Two graphs are needed because each graph plots the sub-RTT for one direction of the conversation. To get the full network RTT you'll need to add the values of both graphs.



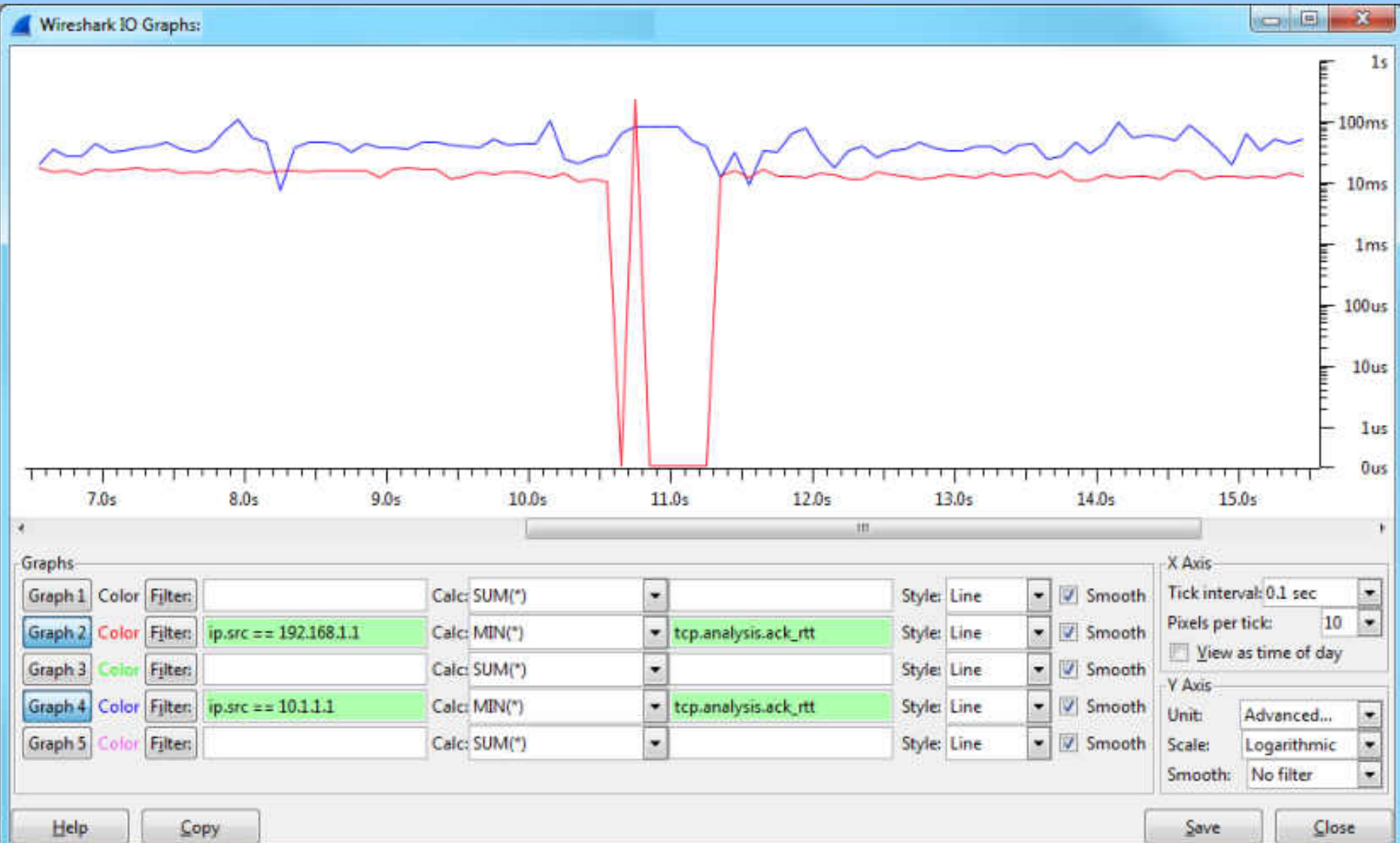
# Trace-Time Continuum \$200

## Answer Part 2

Another way is to use Wireshark's TCP SEQ/ACK analysis capabilities along with an advanced I/O Graph.

Plot the minimum value of `tcp.analysis.ack_rtt` for both the client and server over the course of the conversation.

# Trace-Time Continuum \$200 Example



# Trace-Time Continuum \$200

## Caveat

If layer 4 proxies are present in the path, they may give misleading RTTs no matter which method you employ.

This is one benefit that PINGs may have over the TCP handshake—they aren't proxied.

# Trace-Time Continuum \$300

## Question

Name two TCP features that may intentionally delay sending packets.

# Trace-Time Continuum \$300

## Answer—Part 1

### Tiny Packet Avoiders

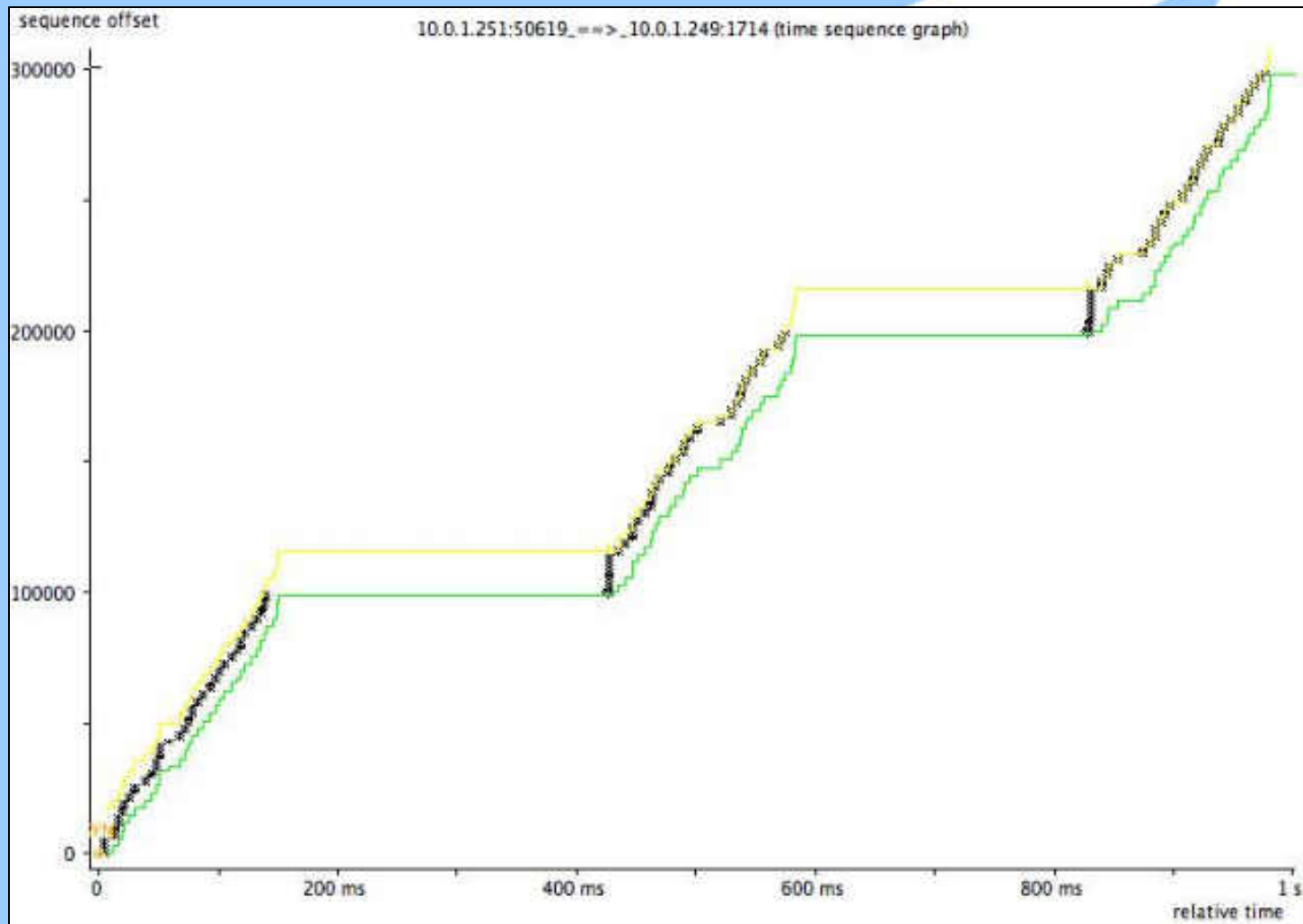
- ***Nagle's Algorithm***—The sending TCP buffers data until one of the following conditions is met:
  - All outstanding data have been ACKed
  - The Nagle timer expires
  - $\geq 1$  MSS worth of data is ready to send
    - A more recent softening of this rule is if there already is one small outstanding segment, don't send another; i.e. one small, odd packet is allowed at the end of a block
- ***Delayed Acknowledgement***—The receiving TCP waits 100–200ms to aggregate ACKs for 2 (or more) received segments into a single ACK.



# Trace-Time Continuum \$300

## More Info on Tiny Packet Avoiders

<http://www.stuartcheshire.org/papers/NagleDelayedAck/>



# Trace-Time Continuum \$300

## Answer—Part 2

**Sliding Window**—The receiving TCP's window size limits the amount of unacknowledged in-flight data.

### Congestion Controllers

- **Slow Start**—At the beginning of a connection or after times of severe packet loss, the sending TCP limits its congestion window to just two or three segments. The congestion window is increased (up to a threshold) by the number of ACKed segments per round trip.
  - This results in exponential growth of the congestion window.
- **Congestion Avoidance**—After packet loss occurs, the sending TCP halves its congestion window and increases it by 1 MSS per round trip.
  - This results in linear growth of the congestion window.

# Trace-Time Continuum \$400

## Question

In TCP, *slow retransmissions* are retransmissions that result from the sender waiting for full retransmission timeout (RTO) interval.

Even when triple-duplicate ACKs are in effect to reduce the likelihood of slow retransmissions, in what scenario are they still encountered?

# Trace-Time Continuum \$400

## Answer

When the last  $n$  packets of a packet burst/block are lost.

When packets arrive out of order, which will occur when a packet goes missing, the receiver goes into verbose ACKing mode, generating an ACK for every segment it receives with the ACK number set to the expected sequence number of the missing segment.

If three such *duplicate* ACKs are observed by the sender, it knows to retransmit that segment immediately.

Notice that generating triple-duplicate ACKs depends upon *trailer segments*—segments arriving at the receiver following the lost segment. If all segments after the first lost segment are also lost, then the receiver generates no ACKs.

Then it is up to the sender to resend those lost segments after the RTO timer expires.

# Trace-Time Continuum \$500

## Question

Assume the bandwidth-delay product between two TCP endpoints is  $x$  bytes. Ignoring packetization/encapsulation overhead, what is the recommended minimum TCP receive window size to ensure that the sender can continuously stream packets even in the face of occasional packet drops?



# Trace-Time Continuum \$500

## Answer

$2x$

It takes 1 RTT to recover a lost segment via a fast retransmission (signaled by a triple-duplicate ACK).

Meanwhile the lower edge of the sliding TCP window is pinned at the sequence number of the lost segment.

Therefore, in order for the sender to continue sending while a segment is recovered (costing 1 RTT), there must be an additional RTT worth of receive window space (2 RTTs worth), and therefore it should be at least  $2x$  the BDP.

# Display Filter Fodder \$100

## Question

Why are display filters like the following misleading?

```
ip.addr != 10.1.1.1  
tcp.port != 80
```

(I call filters like these *chitty chitty bang bangs*—pronounced with a soft *ch*.)

# Display Filter Fodder \$100

## Answer

A single Wireshark dissector field may contain multiple values—one value per occurrence in a packet. Virtual fields like `ip.addr` and `tcp.port` are great examples because there are two (`src` and `dst`) in every TCP packet.

Display filter conditions are satisfied when ANY of the multiple values contained by a field evaluates to true. This can be both handy and misleading, depending on the operator.

# Display Filter Fodder \$100

## Stepped Evaluation of ==

Example packet: `ip.dst=172.16.2.2` ← `ip.src=10.1.1.1`

Display filter: `ip.addr == 10.1.1.1`

1. `ip.addr == 10.1.1.1`
2. `[ip.dst, ip.src] == 10.1.1.1`
3. `[172.16.2.2, 10.1.1.1] == 10.1.1.1`
4. `(172.16.2.2 == 10.1.1.1) || (10.1.1.1 == 10.1.1.1)`
5. `FALSE || TRUE`
6. `TRUE`

Conclusion: Handy!

# Display Filter Fodder \$100

## Stepped Evaluation of !=

Example packet: `ip.dst=172.16.2.2` ← `ip.src=10.1.1.1`

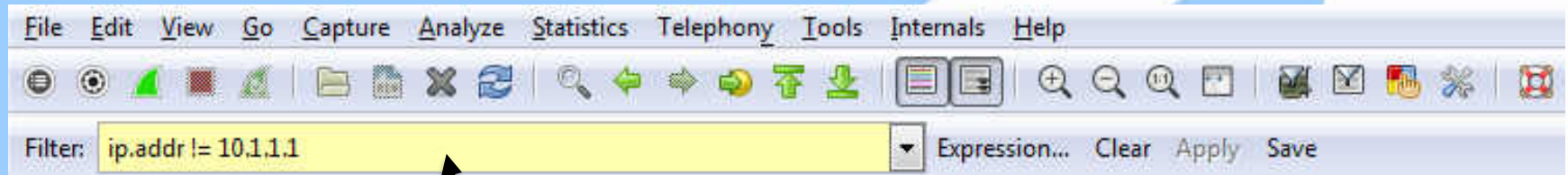
Display filter: `ip.addr != 10.1.1.1`

1. `ip.addr != 10.1.1.1`
2. `[ip.dst, ip.src] != 10.1.1.1`
3. `[172.16.2.2, 10.1.1.1] != 10.1.1.1`
4. `(172.16.2.2 != 10.1.1.1) || (10.1.1.1 != 10.1.1.1)`
5. `TRUE || FALSE`
6. `TRUE`

Conclusion: Misleading!



# Display Filter Fodder \$100 Fortunately Wireshark Warns You



Wireshark: “Yellow may have unexpected results.”

Robert: “If it’s yellow, don’t let it mellow!”

# Display Filter Fodder \$100

## Workaround for !=

Example packet: ip.dst=172.16.2.2 ← ip.src=10.1.1.1

Display filter: !(ip.addr == 10.1.1.1)

1. !(ip.addr == 10.1.1.1)
2. !([ip.dst, ip.src] == 10.1.1.1)
3. !([172.16.2.2, 10.1.1.1] == 10.1.1.1)
4. !((172.16.2.2 == 10.1.1.1) || (10.1.1.1 == 10.1.1.1))
5. !(FALSE || TRUE)
6. !(TRUE)
7. FALSE

Conclusion: Correct

# Display Filter Fodder \$200

## Question

Is it possible for there be more than two `ip.src` fields in a packet? If so, how?

# Display Filter Fodder \$200

## Answer

Yes.

This situation occurs when multiple IP headers are present in a packet. For example, ICMP messages, tunneling protocols, etc.

# Display Filter Fodder \$200 Example

ICMPOnly.pcap [Wireshark 1.10.3\_rvbd (svn revision 5680 from svn://svn/tools/trunk/wireshark)]

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

Filter:  Apply Save

No.	Delta	Time	Source	Destination	Length	Info
1	0.000000	0.000000	192.168.239.58	192.168.121.231	70	Destination unreachable (Fragmentation needed)

Frame 1: 70 bytes on wire (560 bits), 70 bytes captured (560 bits) on interface

- Ethernet II, Src: Cisco\_3f:5b:67 (00:d0:58:3f:5b:67), Dst: DellComp\_0f:e0:35 (00:06:5b:0f:e0:35)
- Internet Protocol Version 4, Src: 192.168.239.58 (192.168.239.58), Dst: 192.168.121.231 (192.168.121.231)
- Internet Control Message Protocol
  - Type: 3 (Destination unreachable)
  - Code: 4 (Fragmentation needed)
  - Checksum: 0x08f3 [correct]
  - MTU of next hop: 1430
- Internet Protocol Version 4, Src: 192.168.121.231 (192.168.121.231), Dst: 192.168.101.152 (192.168.101.152)
  - Version: 4
  - Header length: 20 bytes
  - Differentiated Services Field: 0x80 (DSCP 0x20: Class Selective Service CS4; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
  - Total Length: 1500
  - Identification: 0x91a5 (37285)
  - Flags: 0x02 (Don't Fragment)
  - Fragment offset: 0
  - Time to live: 123
  - Protocol: TCP (6)
  - Header checksum: 0x0726 [in ICMP error packet]
  - Source: 192.168.121.231 (192.168.121.231)
  - Destination: 192.168.101.152 (192.168.101.152)
  - [Source GeoIP: Unknown]

Original IP Header

IP header inside of ICMP message

# Display Filter Fodder \$300

## Question

Display filters are the key mechanism behind Wireshark's *coloring rules* feature, which is touted frequently by Laura Chappell. She likes to create coloring rules for problematic packets using unattractive color combinations.

What is her terminology for this technique?

# Display Filter Fodder \$300

## Answer

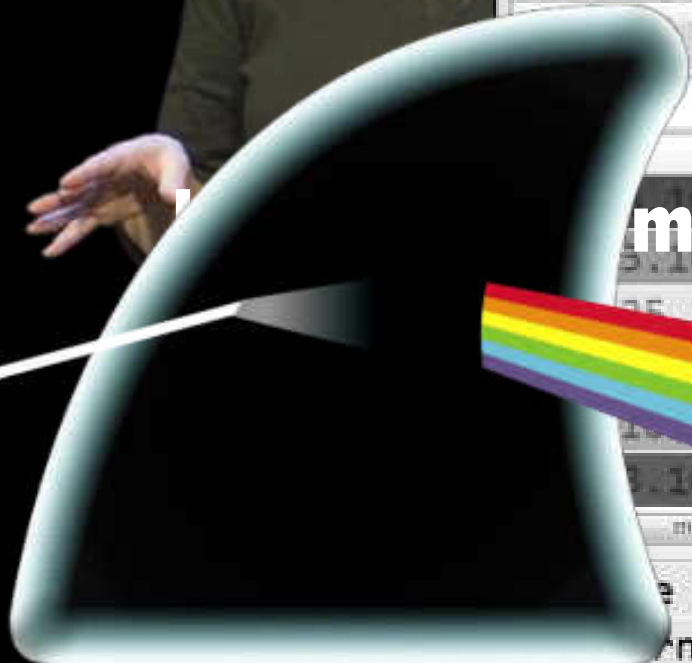
Butt Uglies!







And all that is now,  
 And all that is gone,  
 And all that's to come,  
 All are before me in a fight.



# Funnel for Coloring Rules

Shark Side of the Net  
File Edit View G

Destination	Protocol	Length	Info
10.13.10.25	TCP	62	61500 > 80 [SYN] Seq=8...
74.125.239.102	TCP	66	80 > 61501 [SYN, ACK] Se...
10.13.10.25	TCP	54	61501 > 80 [ACK] Seq=1...
108.162.204.234	TCP	66	80 > 61500 [SYN, ACK] S...
10.13.10.25	TCP	54	61500 > 80 [ACK] Seq=1...

0000	d4 be d9 f8 42 ea 24 77
0010	02 40 07 96 40 00 80 06
0020	cc ea f0 3c 00 50 d9 4c
0030	00 40 92 ae 00 00 47 45
0040	2f 31 2e 31 0d 0a 48 6f
0050	77 69 72 65 73 68 61 72
0060	6f 6e 6e 65 63 74 69 6f
0070	61 6c 69 76 65 0d 0a 41
0080	65 78 74 2f 68 74 6d 6c
0090	74 69 6f 6e 2f 78 68 74
00a0	70 70 6c 69 63 61 74 69

Frame 7: 590 bytes on wire (4720 b) ...  
 Ethernet II, Src: IntelCor\_d9:36:9  
 Internet Protocol Version 4, Src:  
 Transmission Control Protocol, Src:



# **The Shark Side of the Moon**

# Display Filter Fodder \$400

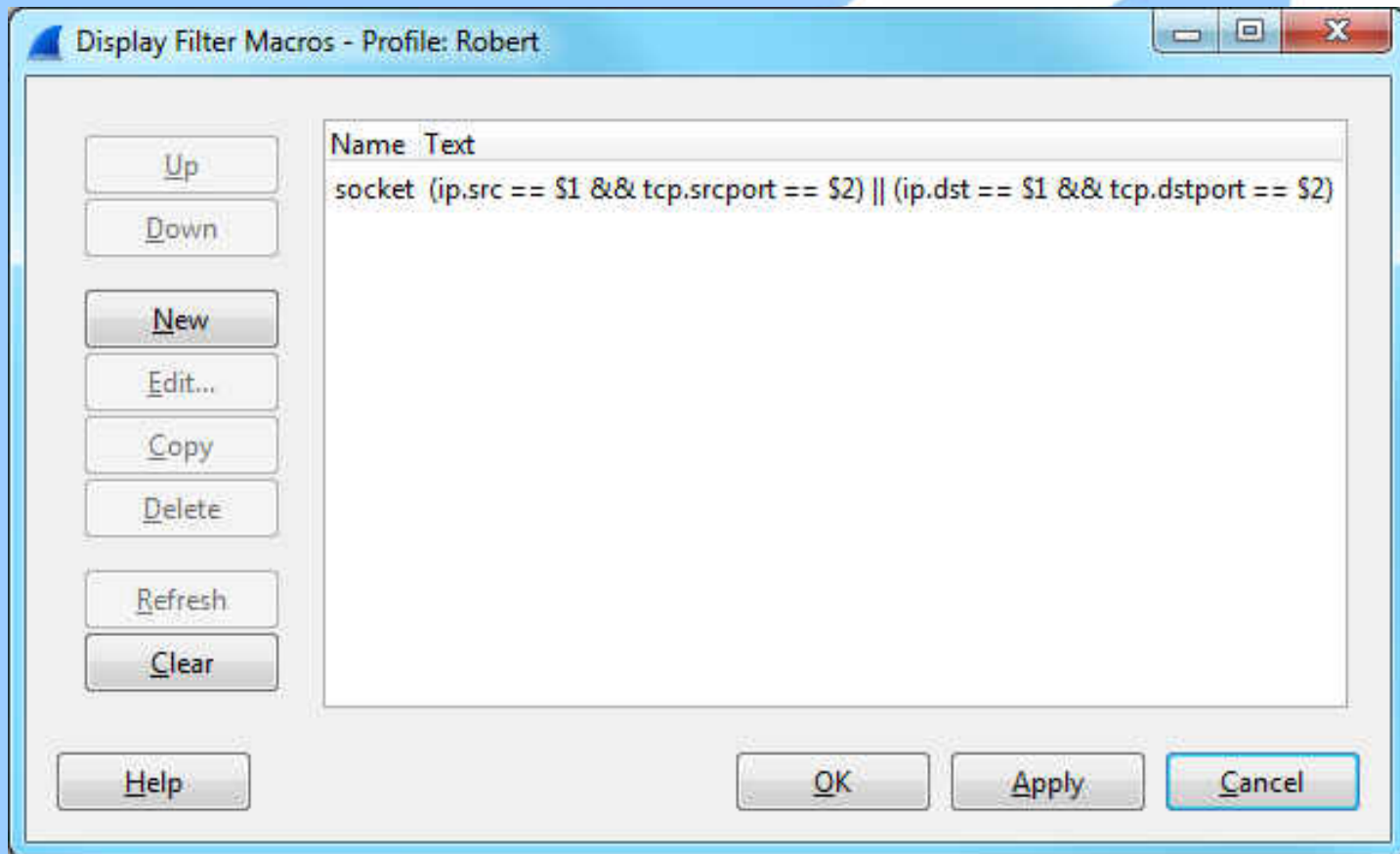
## Question

Which of the following two display filters will accurately allow packets only to or from 10.1.1.1:80 and deny all others?

- A. `ip.addr == 10.1.1.1 && tcp.port == 80`
- B. `${socket: 10.1.1.1;80}`

# Display Filter Fodder \$400 Answer

B. `${socket: 10.1.1.1;80}`



# Display Filter Fodder \$400

## Macros

`{socket: 10.1.1.1;80}` is a display filter macro defined as:

```
(ip.src == $1 && tcp.srcport == $2) ||  
(ip.dst == $1 && tcp.dstport == $2)
```

# Display Filter Fodder \$400

## B vs A

A. `ip.addr == 10.1.1.1 && tcp.port == 80`

B. `${socket: 10.1.1.1;80}`

B is more accurate than A because A will also allow packets like:

`172.16.2.2:80 ← 10.1.1.1:40298`

# Display Filter Fodder \$500

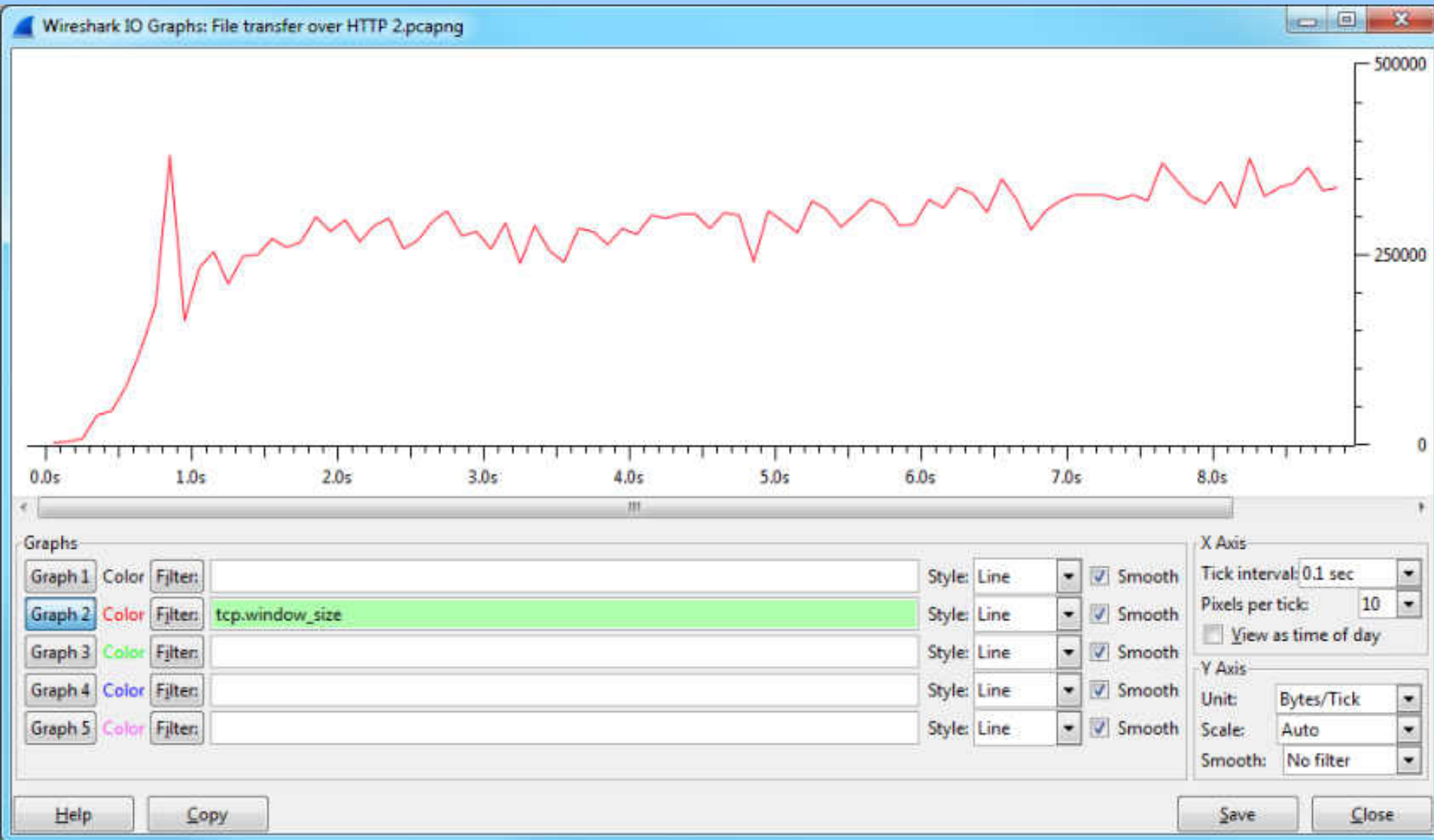
## Question

In addition to isolating traffic in the packet list, display filters can also be used in Wireshark's I/O Graph to plot a subset of packet activity over time.

What is plotted in red on the following graph?



# Display Filter Fodder \$500 Question—I/O Graph



# Display Filter Fodder \$500

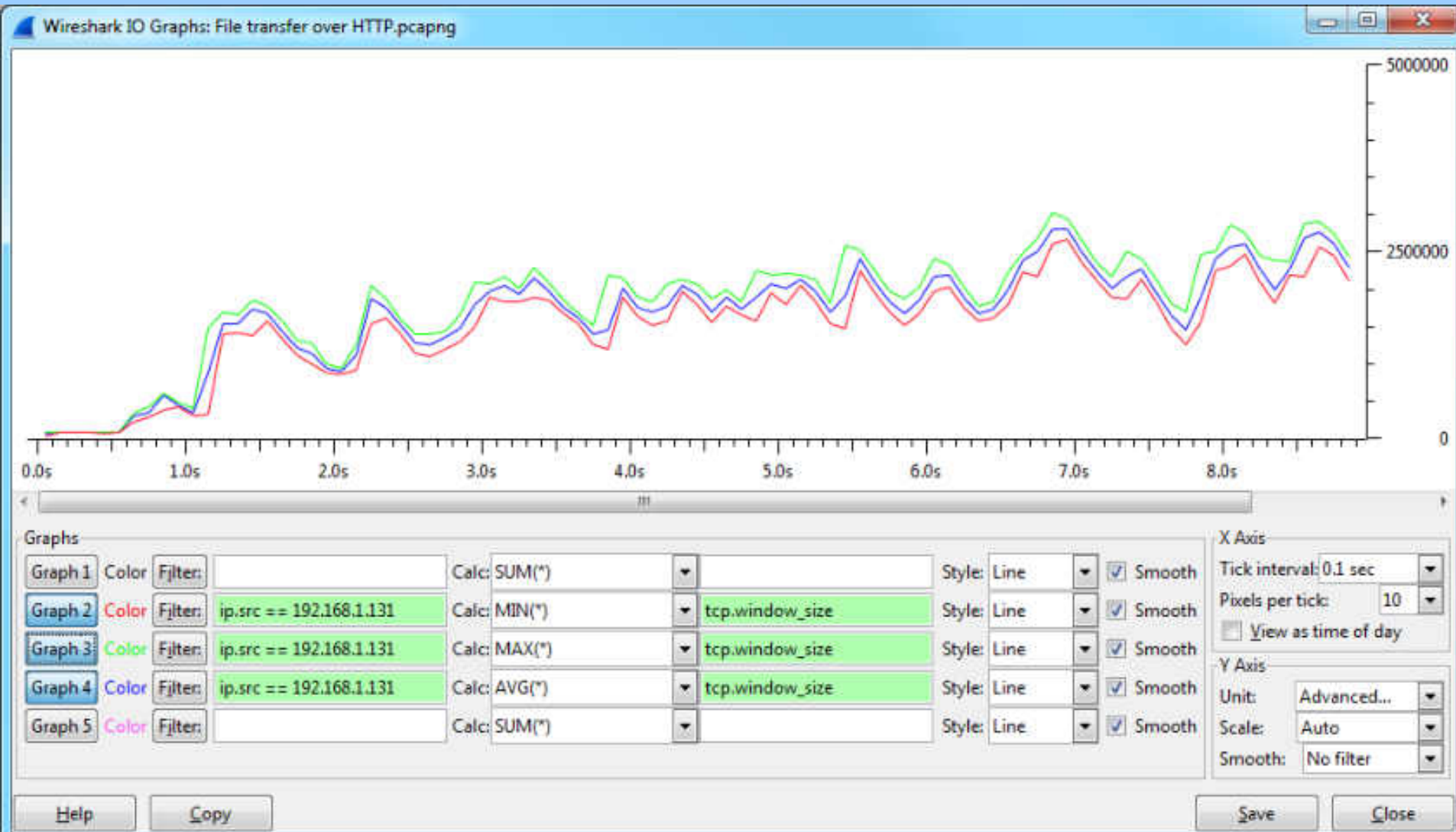
## Answer

The red line plots the packet bytes per tick (1/10<sup>th</sup> of a second) that satisfy the display filter `tcp.window_size`.

In other words, about 250 kilobytes worth of packets per 100ms have the field `tcp.window_size`.

It is NOT the value of TCP window size over time!

# Display Filter Fodder \$500 Answer—Advanced I/O Graph



# What's In A Name? \$100

## Question

If the creator of Wireshark covered up male pattern baldness by brushing his hair from one side of his head to the other, he might also be known as...

# What's In A Name? \$100 Answer



Gerald Combover

# What's In A Name? \$200 Question



Why is Jasper  
Bonkers?

# What's In A Name? \$200

## Answer

For many, many reasons, but mostly because he is coding Trace Wrangler—a packet anonymizing tool. And packet anonymization is *really hard*.

<http://www.tracewrangler.com/>



# What's In A Name? \$300

## Question

User interface development in Wireshark has historically been done with the GTK framework. Current and future UI development will be with the Qt framework, unofficially pronounced “cue-tee.” What is its official pronunciation?

**What's In A Name? \$300**

**Answer**

“Cute”



# What's In A Name? \$400 Question

Whose name is represented by the following puzzle?

Jan	Feb	Mar
Apr	May	Jun
Jul	Aug	Sep
Oct	Nov	Dec

+



+



+



# What's In A Name? \$400

## Answer



Janice Spampinato

# What's In A Name? \$500

## Question

What is the correct pronunciation of Ethereal?

- A. Ether-**real**
- B. **Ate**-her-eel
- C. Eh-thuh-**ray**-al
- D. Eee-**thear**-real

**What's In A Name? \$500**

**Answer**

**Wireshark**

A large, stylized graphic of a shark's dorsal fin is positioned on the right side of the slide. The fin is white with a blue outline and is set against a light blue background. The fin's shape is curved and tapers to a point at the bottom.

# The Command Line \$100

## Question

Which of the following are NOT command line utilities in the Wireshark tool suite? (Pick all that apply.)

- A. dumpcap
- B. mergecap
- C. ettercap
- D. editcap
- E. andycapp
- F. All of the above

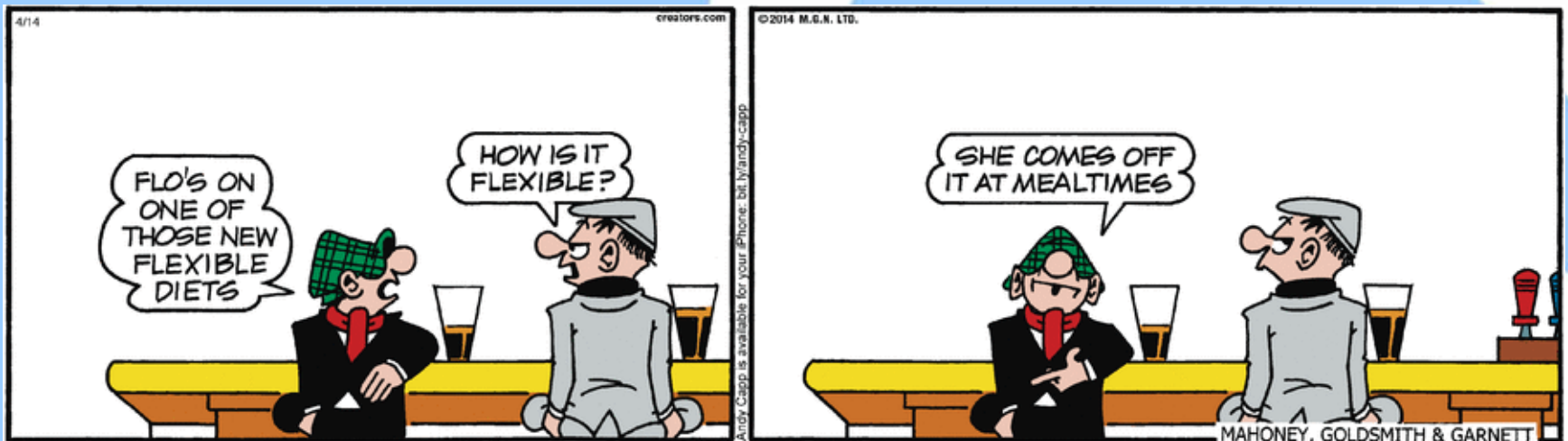


# The Command Line \$100 Answer



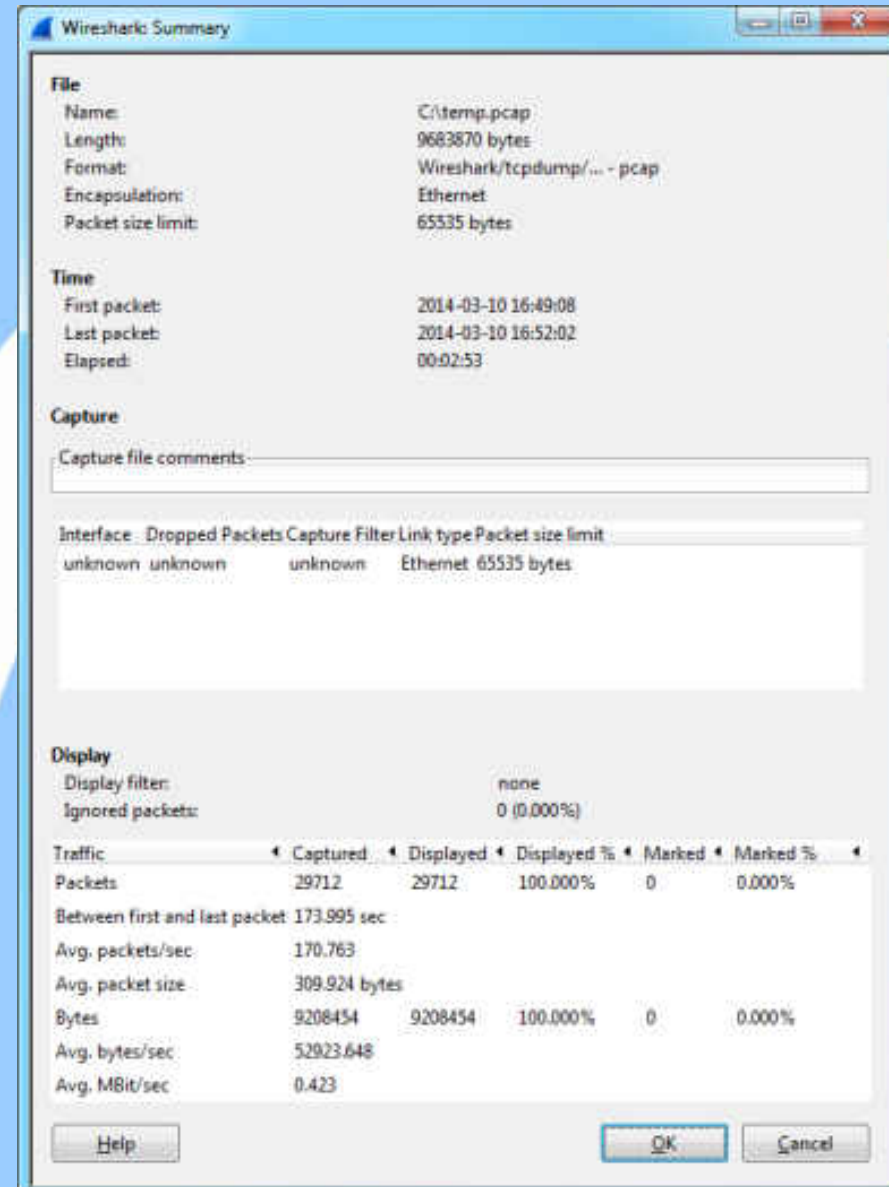
C. **Ettercap** – An open source tool suite for man-in-the-middle attacks.

E. **Andy Capp** – A comic strip about the lifestyle of a dysfunctional British bloke and his wife Flo.



# The Command Line \$200 Question

Which command line utility would you use to gather basic statistics about a packet capture file, akin to those found in Wireshark's Statistics | Summary dialog?



The screenshot shows the 'Wireshark: Summary' dialog box. It is divided into several sections: File, Time, Capture, and Display. The File section shows the file name 'C:/temp.pcap', length '9683870 bytes', format 'Wireshark/tcpdump/... - pcap', encapsulation 'Ethernet', and packet size limit '65535 bytes'. The Time section shows the first packet at '2014-03-10 16:49:08', the last packet at '2014-03-10 16:52:02', and elapsed time '00:02:53'. The Capture section has a text area for 'Capture file comments'. The Display section shows 'Display filter: none' and 'Ignored packets: 0 (0.000%)'. Below this is a table with columns for Traffic, Captured, Displayed, Displayed %, Marked, and Marked %.

Traffic	Captured	Displayed	Displayed %	Marked	Marked %
Packets	29712	29712	100.000%	0	0.000%

Below the table, it shows 'Between first and last packet: 173.995 sec'. Further statistics include 'Avg. packets/sec: 170.763', 'Avg. packet size: 309.924 bytes', 'Bytes: 9208454', 'Avg. bytes/sec: 52923.648', and 'Avg. MBit/sec: 0.423'. At the bottom, there are buttons for 'Help', 'OK', and 'Cancel'.

# The Command Line \$200

## Answer

```
>capinfos temp.pcap
File name:          temp.pcap
File type:          Wireshark/tcpdump/... - pcap
File encapsulation: Ethernet
Packet size limit:  file hdr: 65535 bytes
Number of packets:  29 k
File size:          9683 kB
Data size:          9208 kB
Capture duration:   174 seconds
Start time:         Mon Mar 10 16:49:08 2014
End time:           Mon Mar 10 16:52:02 2014
Data byte rate:     52 kBps
Data bit rate:      423 kbps
Average packet size: 309.92 bytes
Average packet rate: 170 packets/sec
SHA1:               474aebc6827bfc6b2c8273521b434d1f9af55ef8
RIPEMD160:          132e2659895caa4cab92f02f6bfcd8a9afda1b36
MD5:                 5e2ba1d1279ef8956ebcac2662d925b6
Strict time order:  True
```

# The Command Line \$300 Question

If you needed to break an extremely large packet capture file into multiple shark-bite sized subfiles, divided at regular frame counts or time intervals, which utility would be the most efficient at such a task?

# The Command Line \$300

## Answer

editcap is the most efficient tool for this job because it doesn't track conversations, tabulate statistics, or even dissect packets. Therefore it has a fixed memory requirement and is essentially I/O-bound.

editcap has the ability to output multiple files, each with  $x$  packets per file or  $y$  seconds per file. It can also do minimal filtering of frames that fall within or without of a given time slice.

# The Command Line \$400 Question

Which command line tool can be used to remove duplicate frames from a packet capture file?

# The Command Line \$400

## Answer

editcap is able to remove byte-for-byte duplicate frames with one of three switches:

`-d/-D <dedup FIFO capacity>`

`-w <dedup FIFO time duration>`

# The Command Line \$500 Question

tshark, the terminal (command line) version of Wireshark, is able to read a packet capture file and apply a display filter in order to select a subset of the file's packets.

There are two argument switches that specify a display filter. What are they and how do they differ?



# The Command Line \$500

## Answer

-R “<read filter>” -2

This filter is applied when packets are being *read* (the first pass). Forward-looking fields, such as ‘response in frame #’, will not yet have been calculated. This used to be the only display filter switch but has succeeded by...

-Y “<display filter>”

This filter is applied when packets are being *displayed* (the second pass), which could mean being printed to stdout or written to a file. **In most cases this is the switch you want because it is equivalent in function to Wireshark’s display filter tool bar.**

# **Pimp My Shark! \$100**

## **Question**

Name one of the ways you can customize the font appearance in Wireshark.

# Pimp My Shark! \$100

## Answer 1

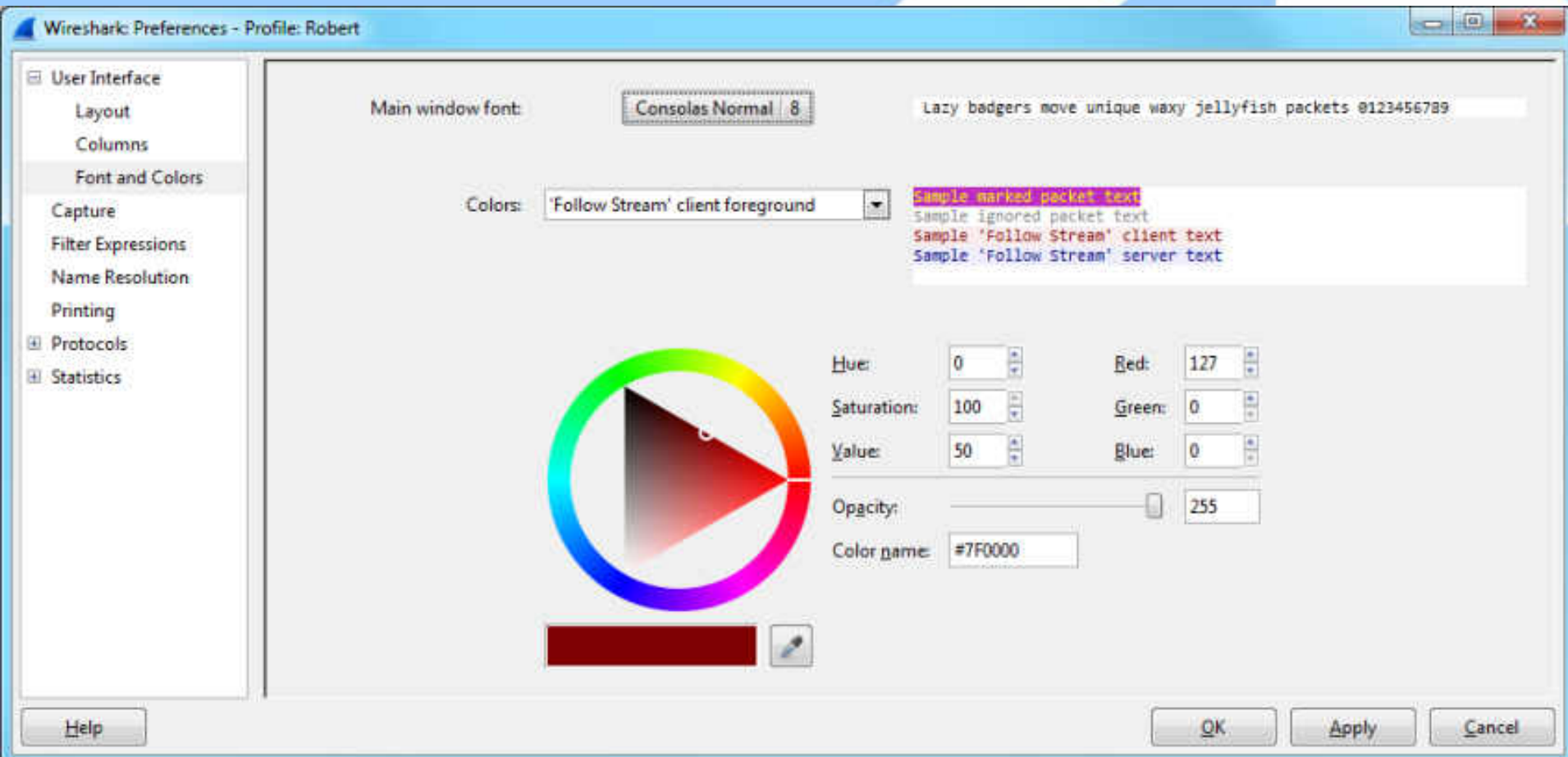
For quick size increases/decreases, use the toolbar “zoom” buttons



# Pimp My Shark! \$100

## Answer 2

The Font and Colors preferences dialog



# **Pimp My Shark! \$200**

## **Question**

Name one of the numerous ways to customize the appearance of the packet summary list.

# Pimp My Shark! \$200

## Answer

- Change the time display format
- Enable/disable name resolution for MAC, network, and transport layer
- Add/remove/hide/show/reorder columns
- Use custom coloring rules
- Change field alignment (left, right, or center)

# Pimp My Shark! \$300

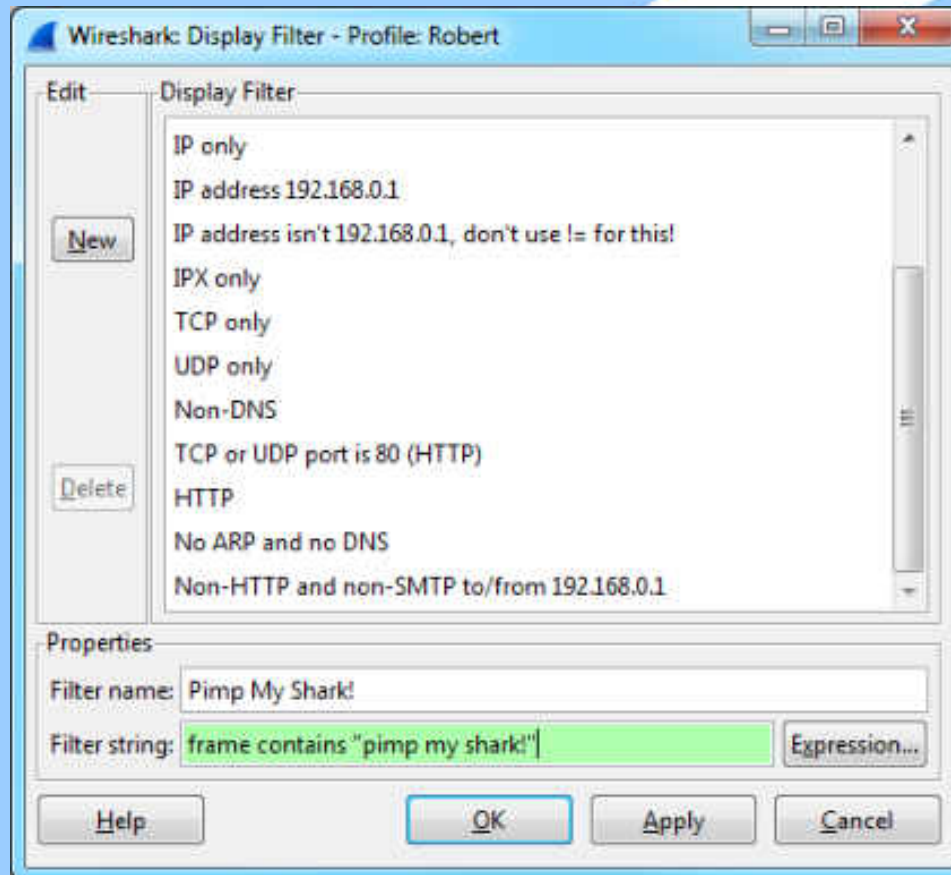
## Question

How can you save time and/or typing when you find yourself using common display filters over and over?

# Pimp My Shark! \$300

## Answer 1

Save them in the Display Filters dialog





# Pimp My Shark! \$300

## Answer 2

Add them to the Display Filter toolbar as Expression buttons



# Pimp My Shark! \$400

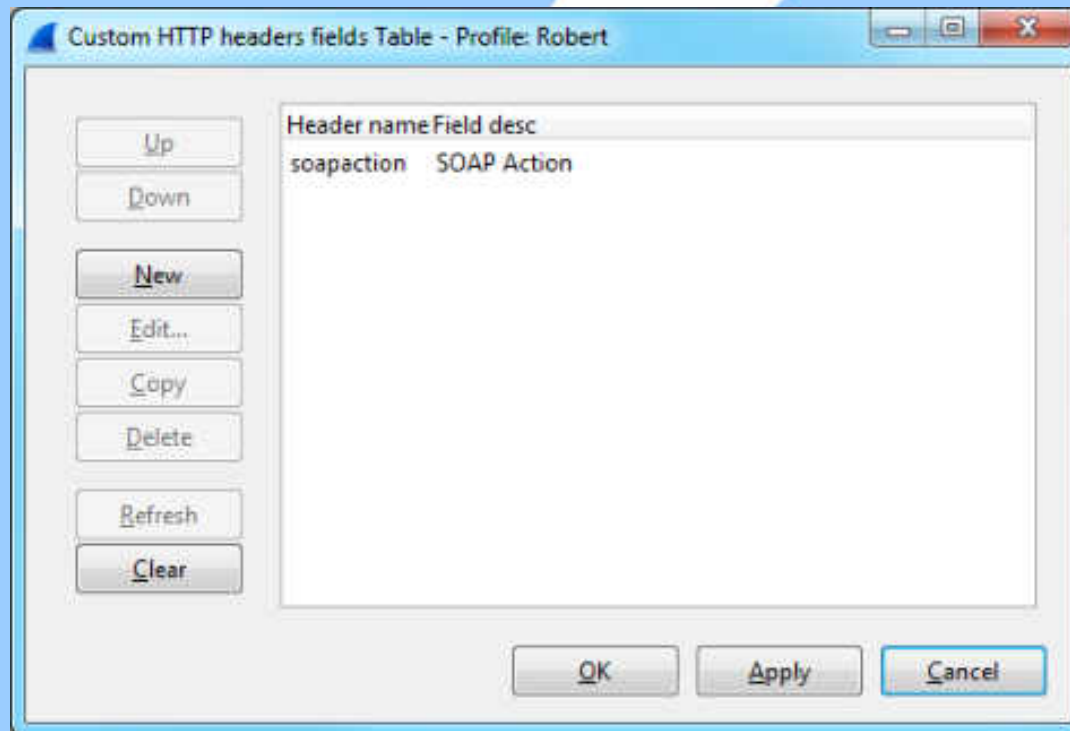
## Question

How can you tell Wireshark to treat a non-standard or custom HTTP header as a filterable/exportable field?

# Pimp My Shark! \$400

## Answer


Add it to HTTP's protocol's preferences! It then becomes available as something like `http.header.soapaction`.



# **Pimp My Shark! \$500**

## **Question**

How can you make Wireshark do something that it can't currently do?



# **Pimp My Shark! \$500**

## **Answer**

Program it yourself!

The beauty of Wireshark being open source is that anyone can enhance, extend, or customize the application.