



SHARKFEST'14

WIRESHARK DEVELOPER AND USER CONFERENCE

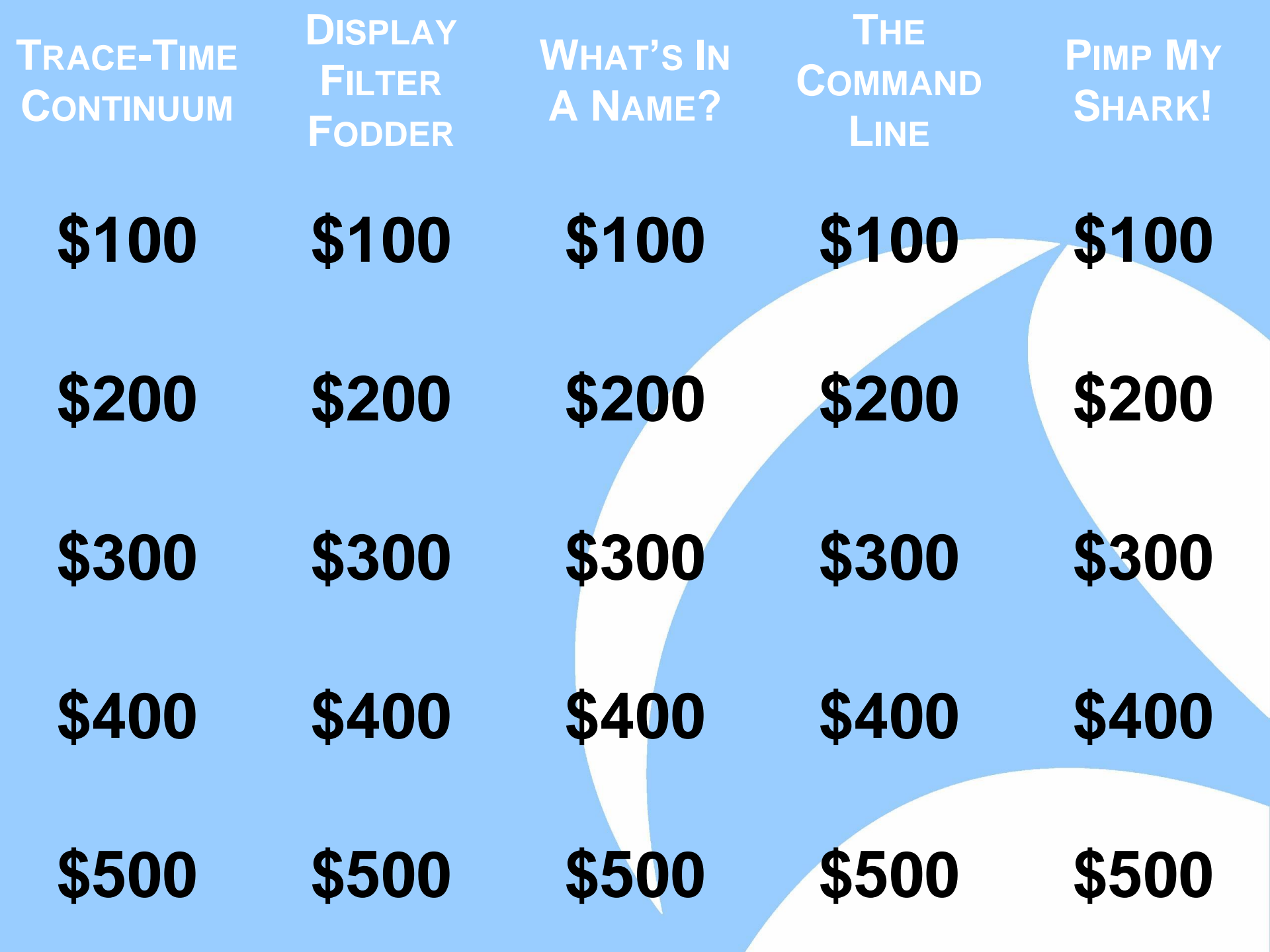
JUNE 16-20 2014 · DOMINICAN UNIVERSITY

Sharkfest Jeopardy!

Robert Bullen

Application Performance Engineer
Blue Cross Blue Shield of Minnesota
robert_d_bullen@bluecrossmn.com

SHARKFEST JEOPARDY!



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

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

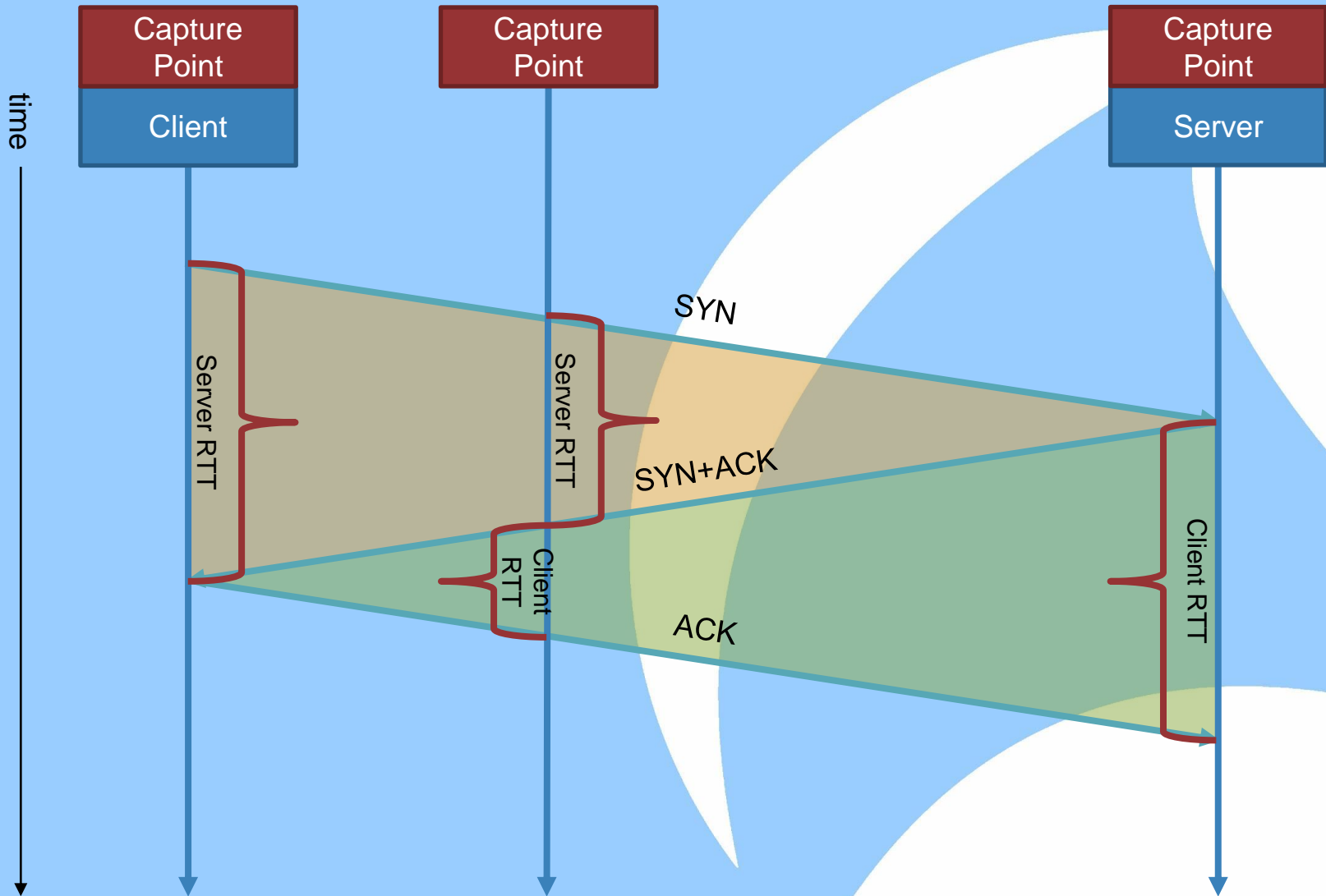
Answer

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.

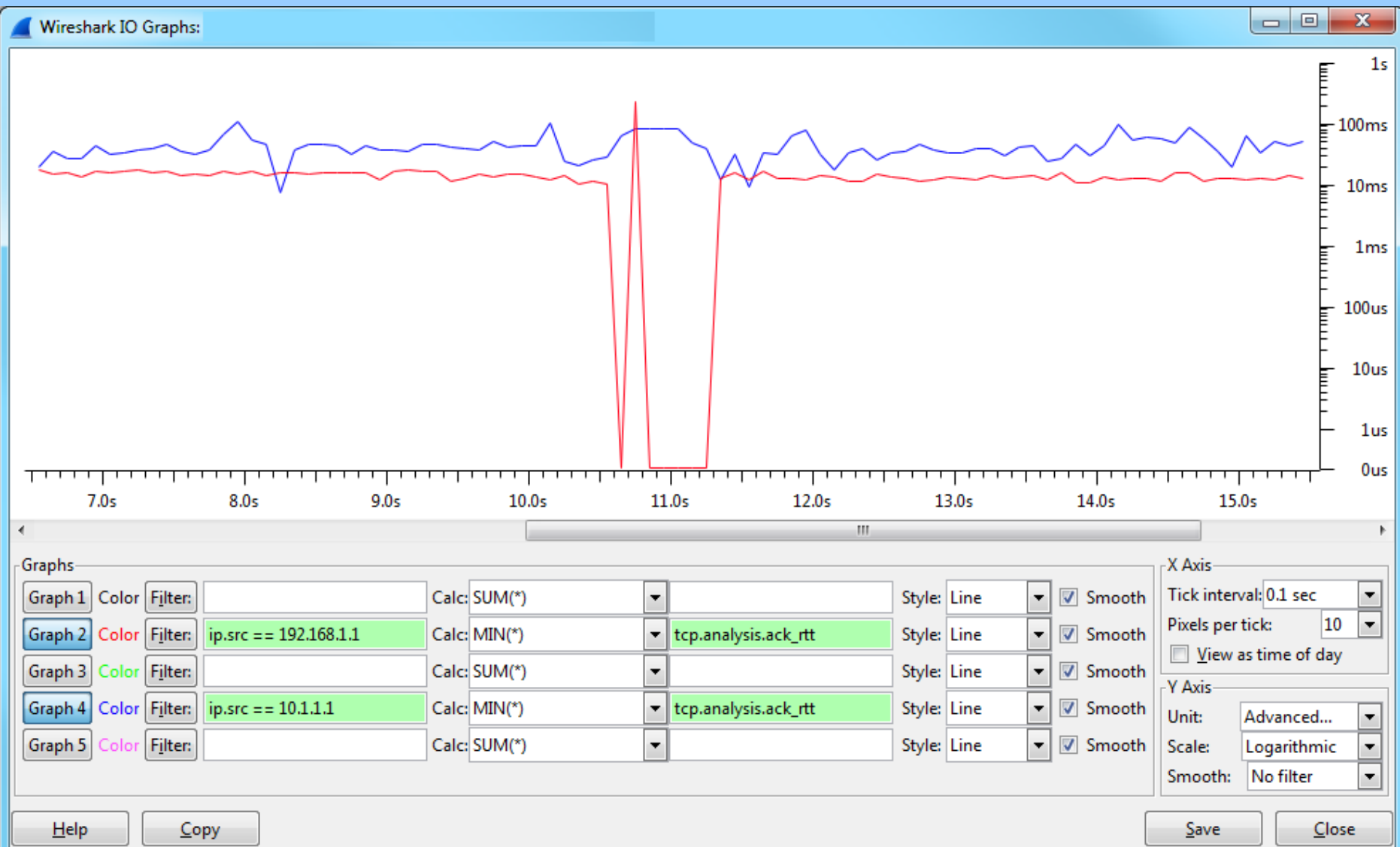
Trace-Time Continuum \$200

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

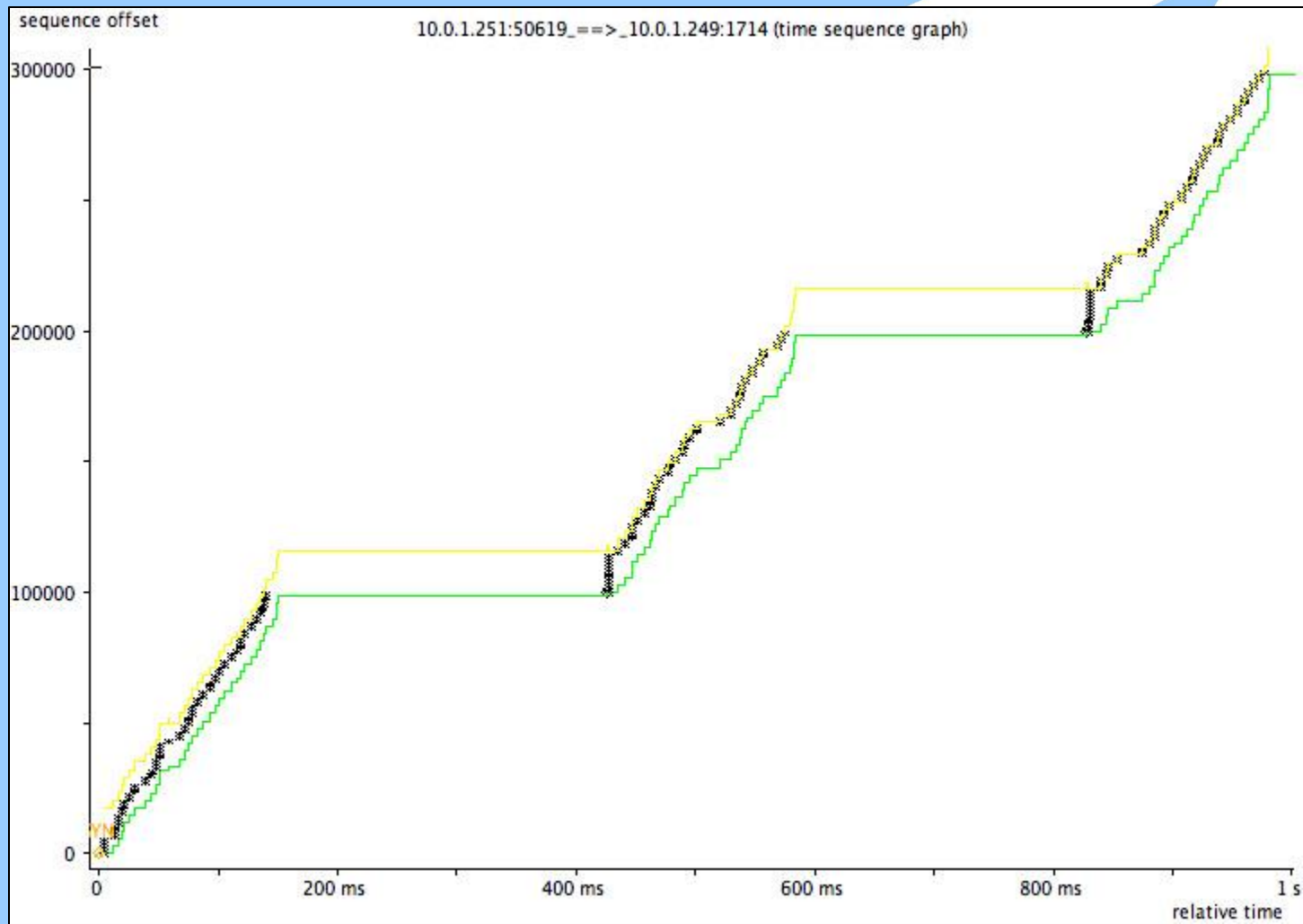
Tiny Packet Avoiders

- ***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
 - ≥ 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`

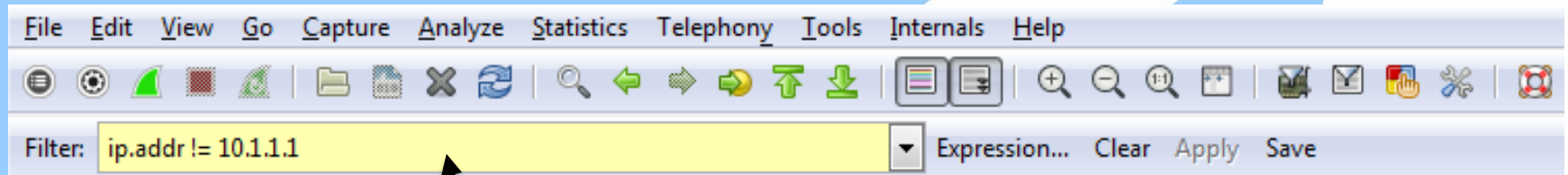
Display filter: `ip.addr != 10.1.1.1`

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 0

- 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 4: 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 everything under
And all that you have,
And all that is now,
And all that's yet to come,
And all that is done,
And all that's to be,
And all that's to come,
And all that's to be.

Angry ones too

m for Coloring Rules

Coloring Rules

The image shows a Wireshark packet capture of a TCP connection establishment. The packets are color-coded: red for SYN, yellow for ACK, and green for the final ACK. The packet list on the left shows the following details:

No.	Time	Source	Destination	Protocol	Length	Info
10	0.000000	10.13.10.25	108.162.204.234	TCP	62	61500 → 80 [SYN] Seq=0
11	0.000000	10.13.10.25	74.125.239.102	TCP	62	61501 → 80 [SYN] Seq=0
12	0.000000	74.125.239.102	10.13.10.25	TCP	66	80 → 61501 [SYN, ACK] Seq=1
13	0.000000	10.13.10.25	74.125.239.102	TCP	54	61501 → 80 [ACK] Seq=1
14	0.000000	108.162.204.234	10.13.10.25	TCP	66	80 → 61500 [SYN, ACK] Seq=1
15	0.000000	10.13.10.25	108.162.204.234	TCP	54	61500 → 80 [ACK] Seq=1

The packet details pane on the right shows the structure of the selected packet (Frame 7):

- 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 packet bytes pane shows the raw data in hexadecimal and ASCII.

The image features a large, detailed full moon in the center. The background is black, with four horizontal stripes of rainbow colors (red, orange, yellow, and purple) running across it. Overlaid on these stripes are green, jagged lines that resemble an ECG or heartbeat, with peaks and valleys. The title text is centered over the moon.

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?

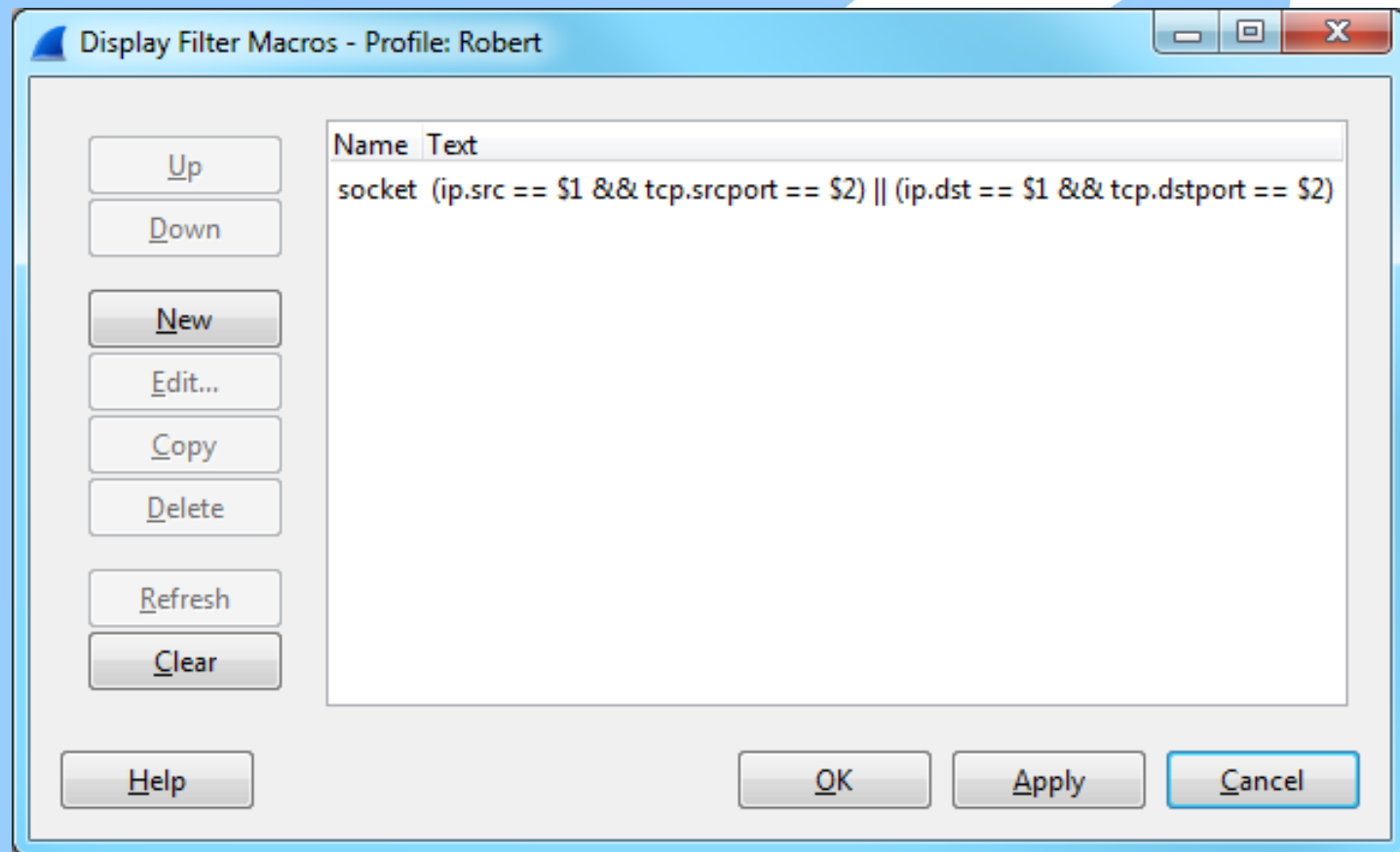
A. `ip.addr == 10.1.1.1 && tcp.port == 80`

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

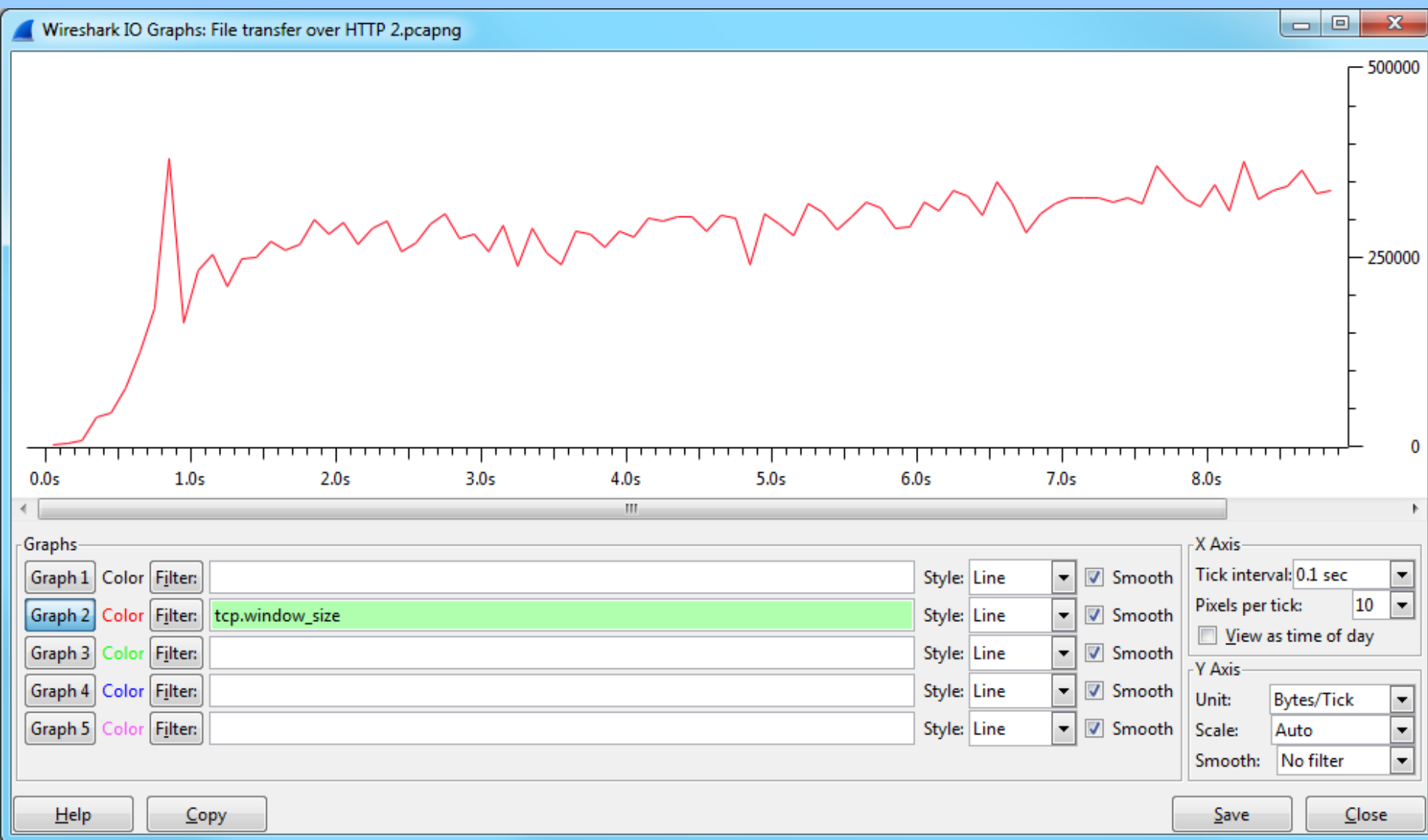
Question

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

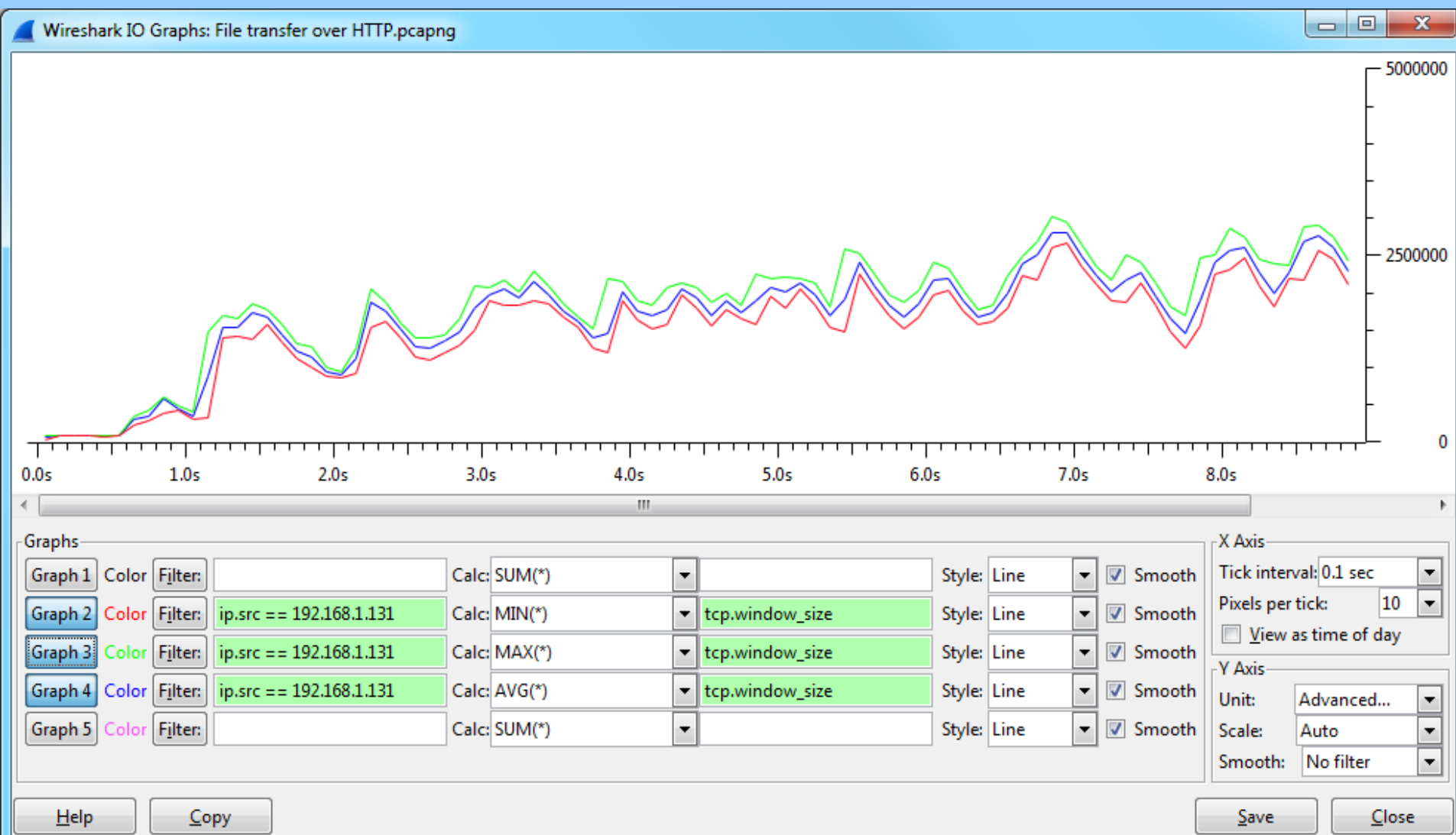
The red line plots the packet bytes per tick ($1/10^{\text{th}}$ of a second) that satisfy the display filter `tcp.window_size`.

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”

A decorative graphic on the right side of the slide, featuring a portion of a beach ball with alternating light blue and white curved segments.

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 Ethereum?

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

What's In A Name? \$500

Answer

Wireshark



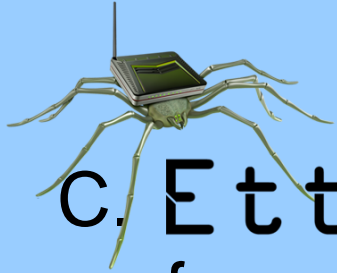
The Command Line \$100

Question

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

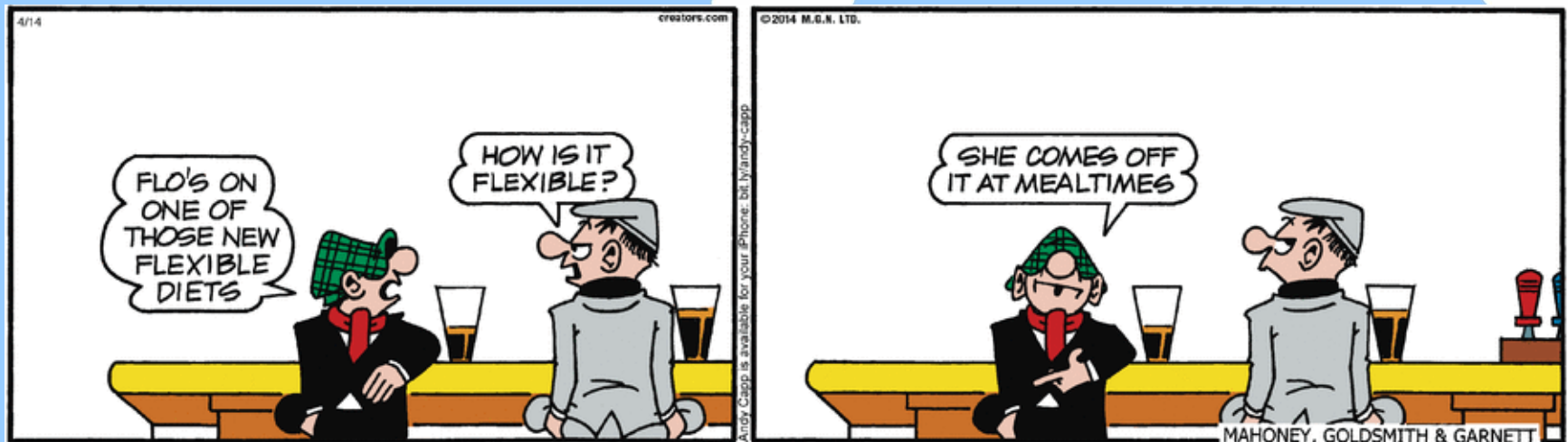
- A. dumpcap
- B. mergecap
- C. ettercap
- D. editcap
- E. andycapp
- F. All of the above

The Command Line \$100 Answer



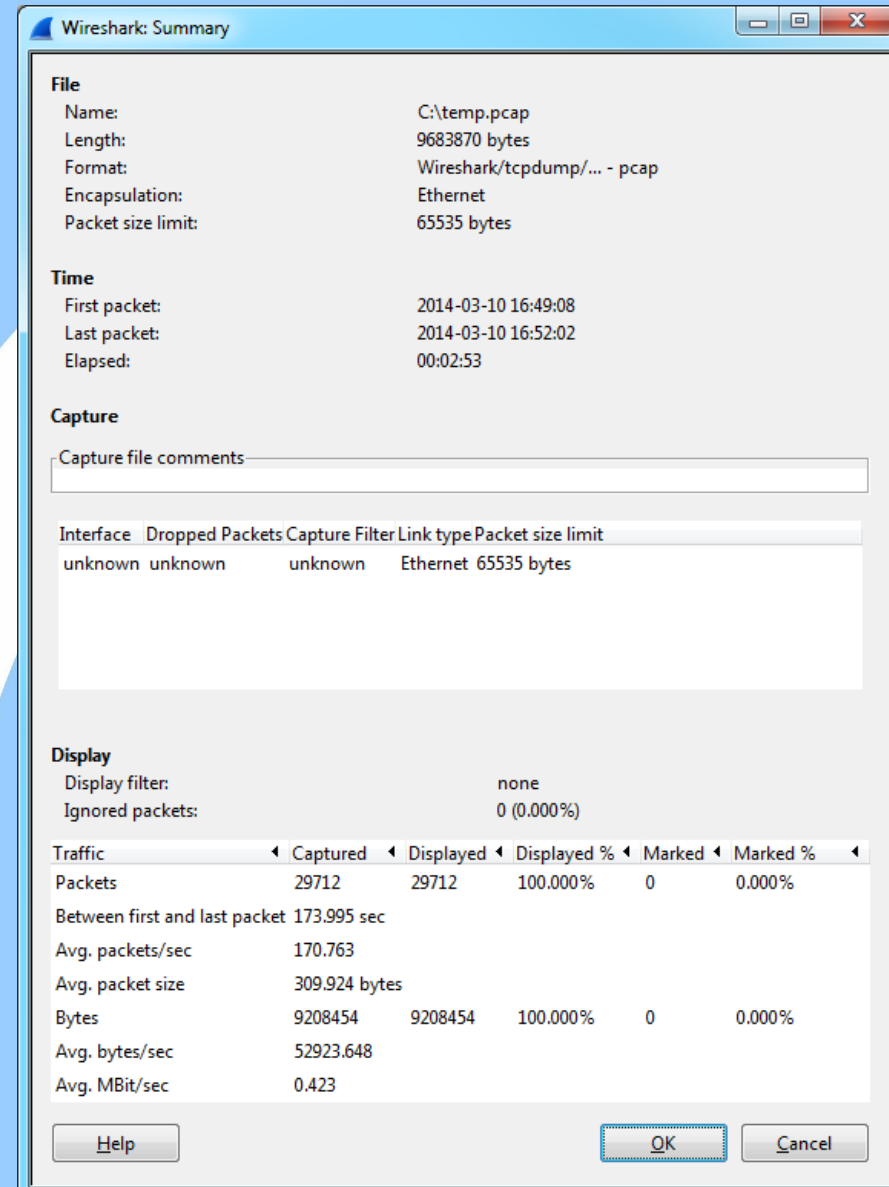
C. **Ettercap** – An open source tool suite for man-in-the-middle attacks.

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 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

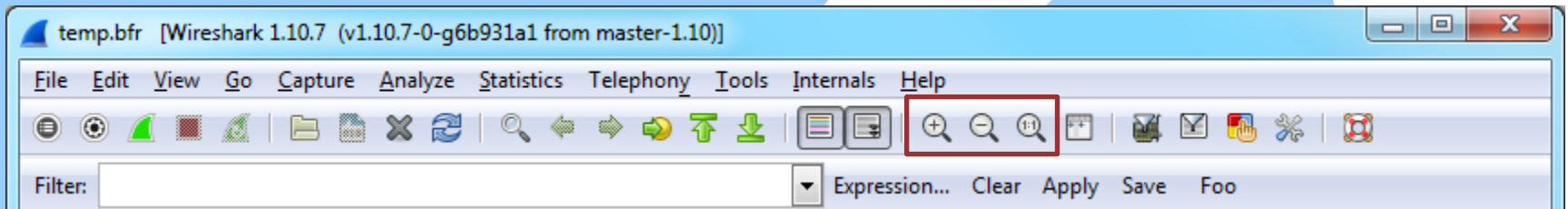
Name one of the ways you can customize the font appearance in Wireshark.



Pimp My Shark! \$100

Answer 1

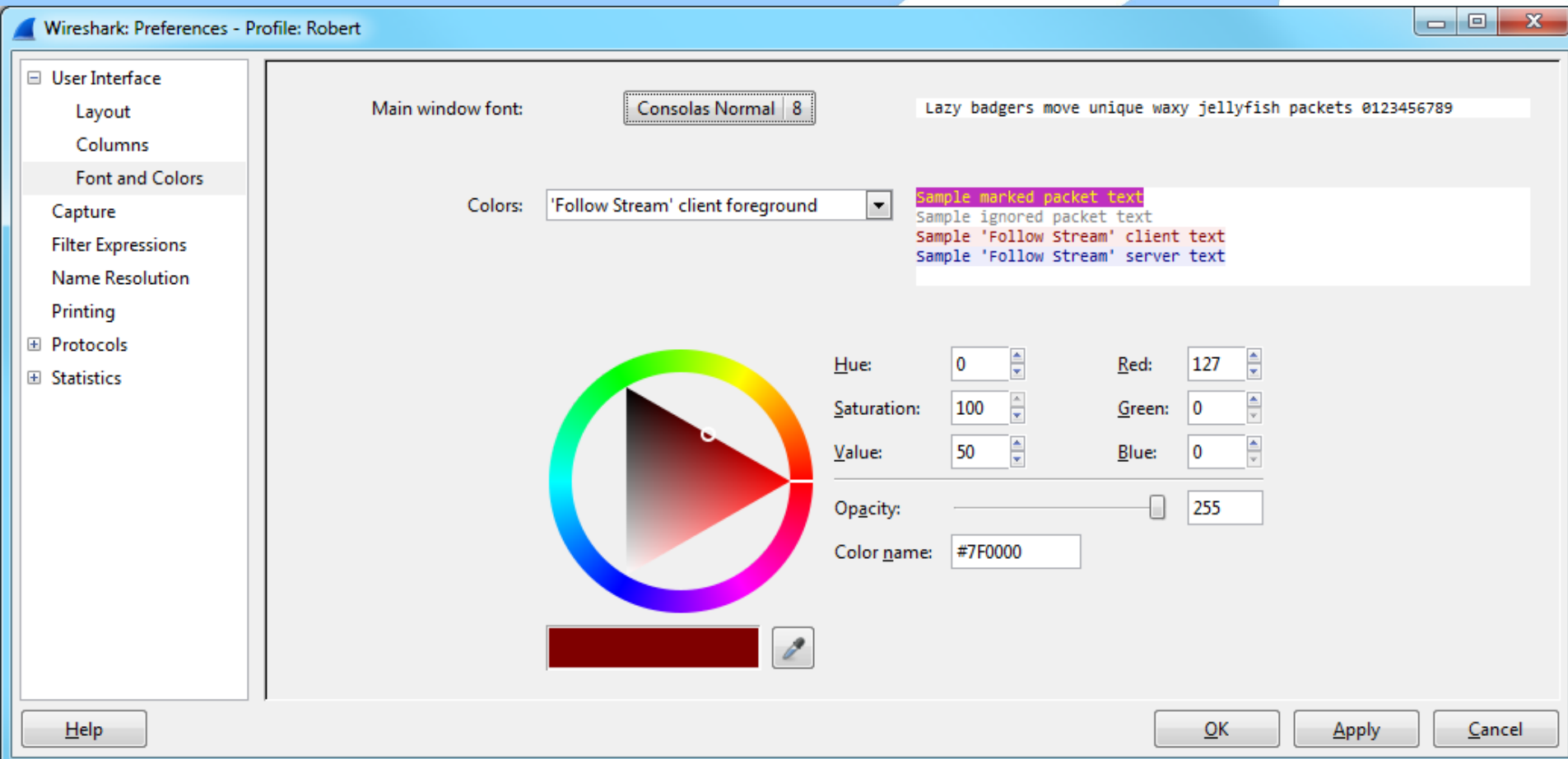
For quick size increases/decreases, use the toolbar “zoom” buttons



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

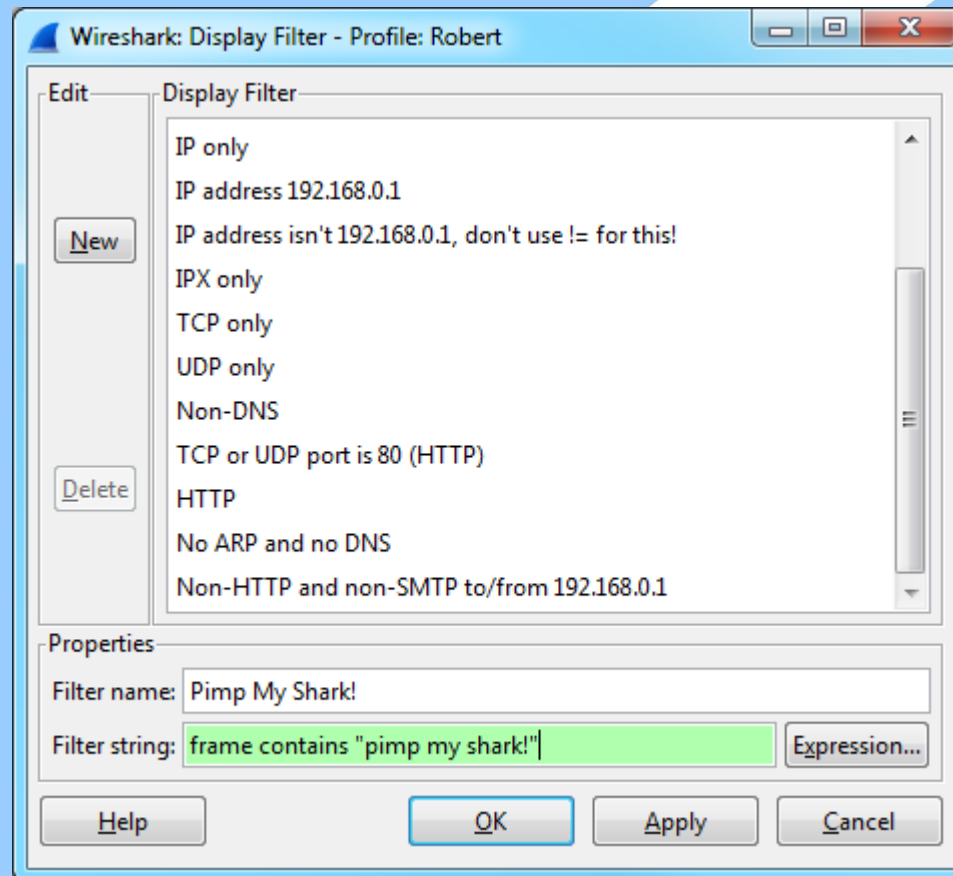
Question

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

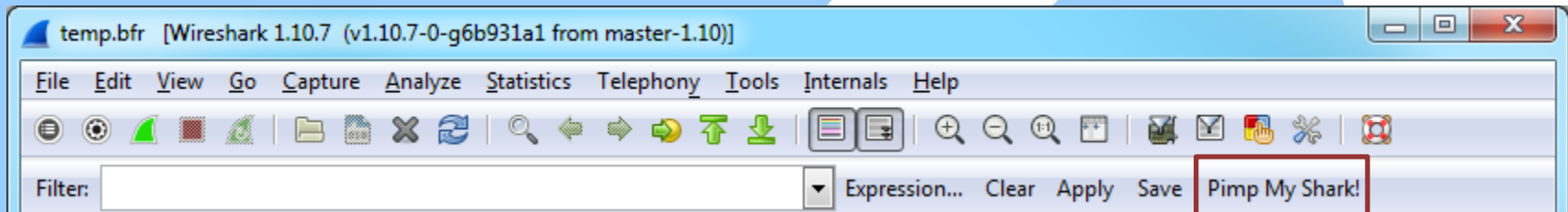
Save them in the Display Filters dialog



Pimp My Shark! \$300

Answer 2

Add them to the Display Filter toolbar as Expression buttons



Pimp My Shark! \$400

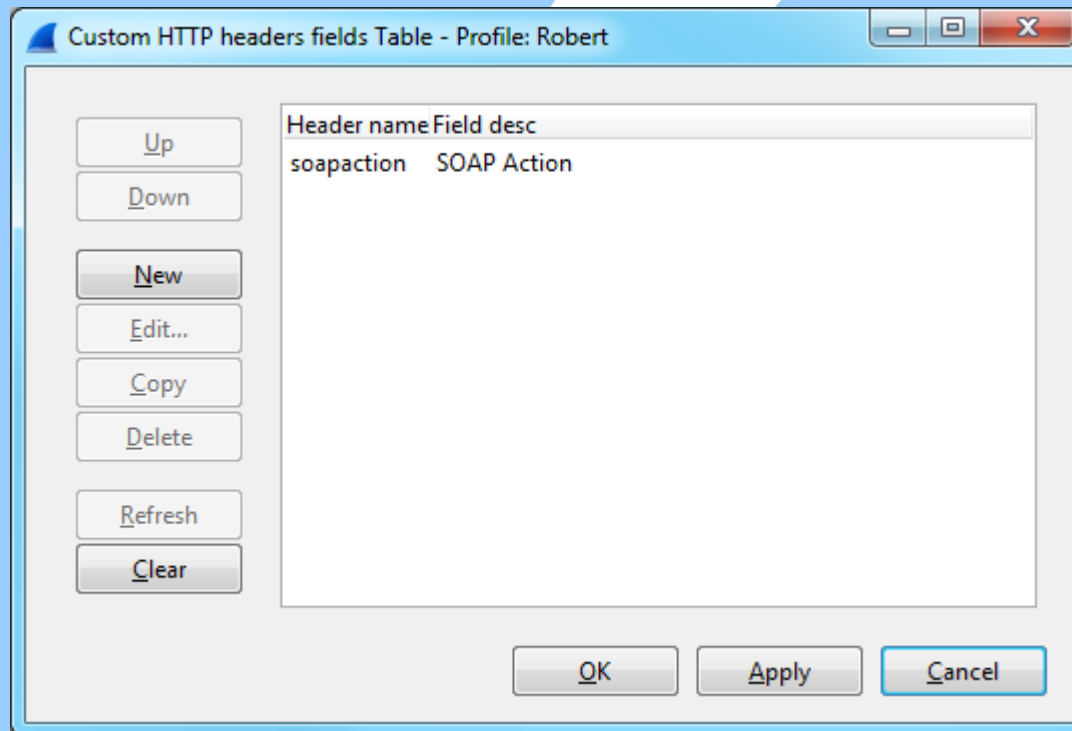
Question

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

Pimp My Shark! \$400

Answer

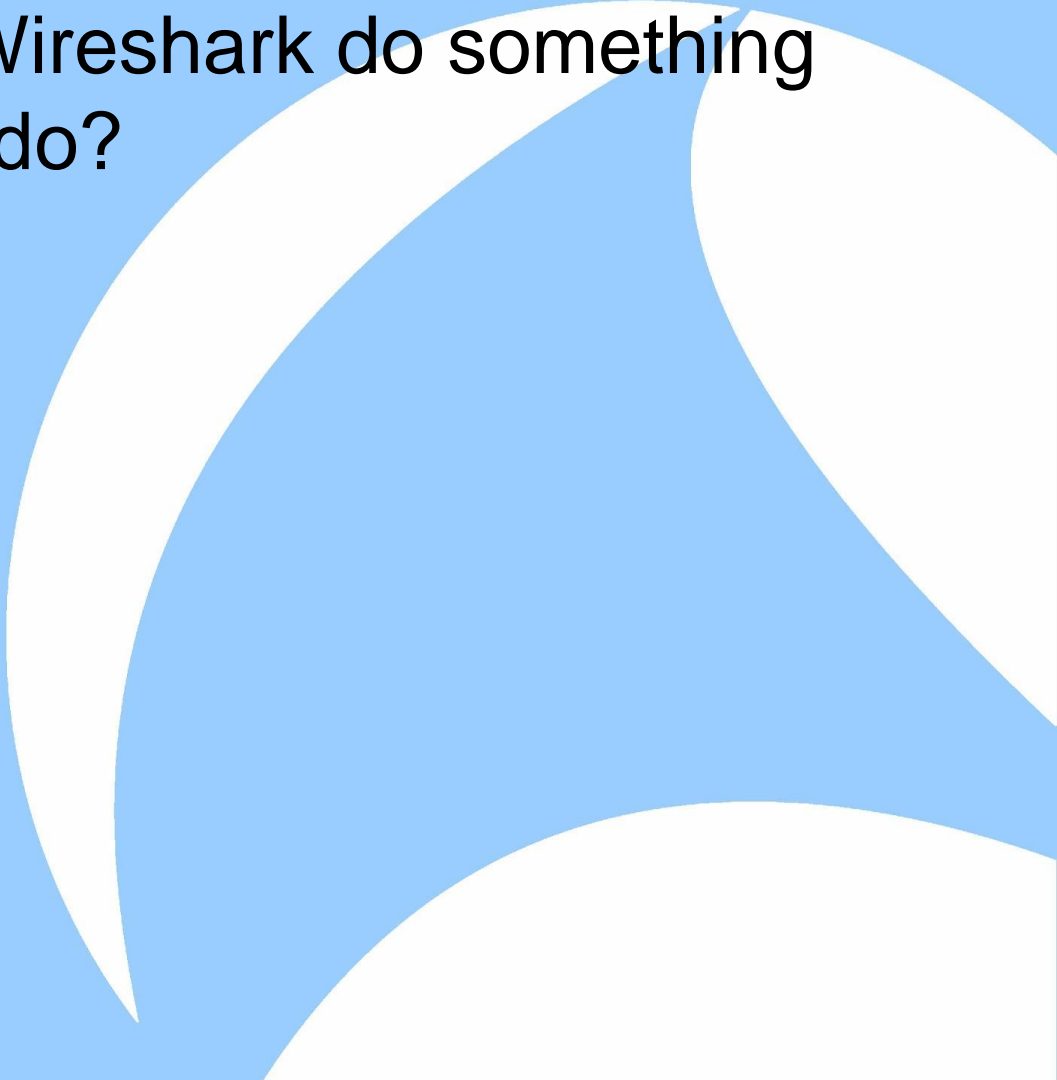
Add it to HTTP's protocol's preferences! It then becomes available as something like `http.header.soapaction`.



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.