Using Wireshark For Analyzing CIFS Traffic

Ronnie Sahlberg, Samba Team

WHO AM I?

Me

- Ronnie Sahlberg : OpenSource Developer.
  CTDB hacker
  Samba hacker
  SCSI/iSCSI hacker
  Wireshark hacker
  ...

INTRO

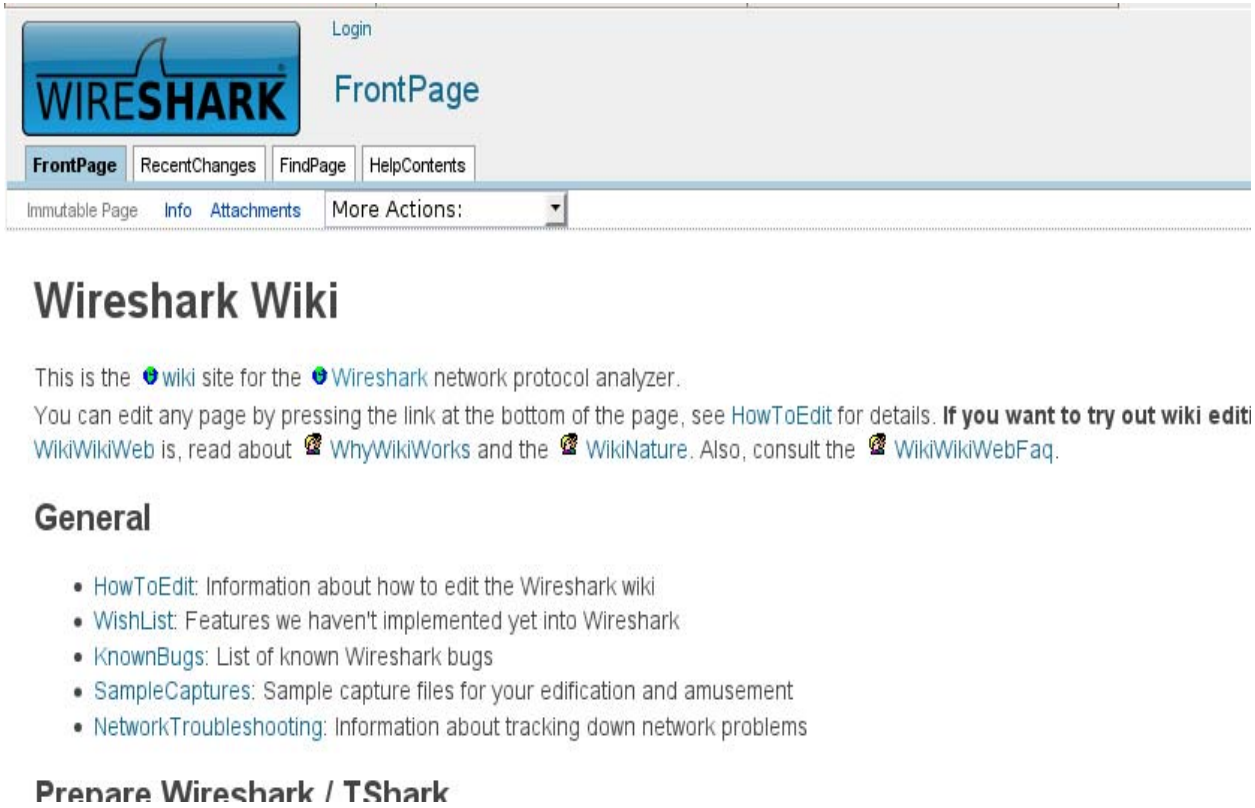
WIRESHARK

- Wireshark is the worlds most popular and complete protocol analyzer.
- Wireshark is free software and is available for almost all types of Unix and Unix-like systems and Windows.
- Wireshark has many unique features with will help you analyze CIFS and other protocols.

WIRESHARK

- WWW.WIRESHARK.ORG

- WIKI.WIRESHARK.ORG



Intro :Wireshark vs the world

What is Wireshark?

WIRESHARK

- Wireshark is a **protocol analyzer**.
- Wireshark is one of very very few protocol analyzers available.
- This means Wireshark is designed to decode not only packet bits and bytes but also the relations between packets and protocols.
- Wireshark understands protocol sequences.

WIRESHARK

- Wireshark uses a lot of memory and cpu to do its analysis and state tracking.

- Wireshark is NOT a real-time application and never will be!

WIRESHARK

- Do not confuse this with a network monitoring tool.
- Network monitoring tools are usually IS type of applications and not engineering tools.
- Network monitoring tools are usually semi-realtime.

- Network monitoring tools often trade detailed analysis for higher speed.

WIRESHARK

- Almost all the tools you might think of as being "similar" to Wireshark are in fact network monitoring tools and not similar at all.

- Different tools for different purposes.

WIRESHARK

- Yes. I know that Wireshark has “less than perfect” support for some foreign proprietary file any formats.
- This will not be fixed anytime soon unless the vendors contribute and document their file formats.

WIRESHARK

- Many commercial network capture, monitoring and analysis software and hardware vendors contribute and maintain code in Wireshark to interoperate with their own file formats.
- Others do not and that means that you, as the end-user suffers.

Filter languages, why have one when you can have two?

FILTERS

- Wireshark provides separate filter languages for different purposes:
  CAPTURE FILTERS
  and
  DISPLAY FILTERS

- Using Wireshark you will normally only use DISPLAY FILTERS but it is useful to know the distinction between them.

FILTERS

- Filtering in Wireshark is VERY important. It is the Alpha-and-Omega of powerful traffic analyzis.
- If you don't practice and learn the filtering languages your Wireshark experience will be sub-optimal.

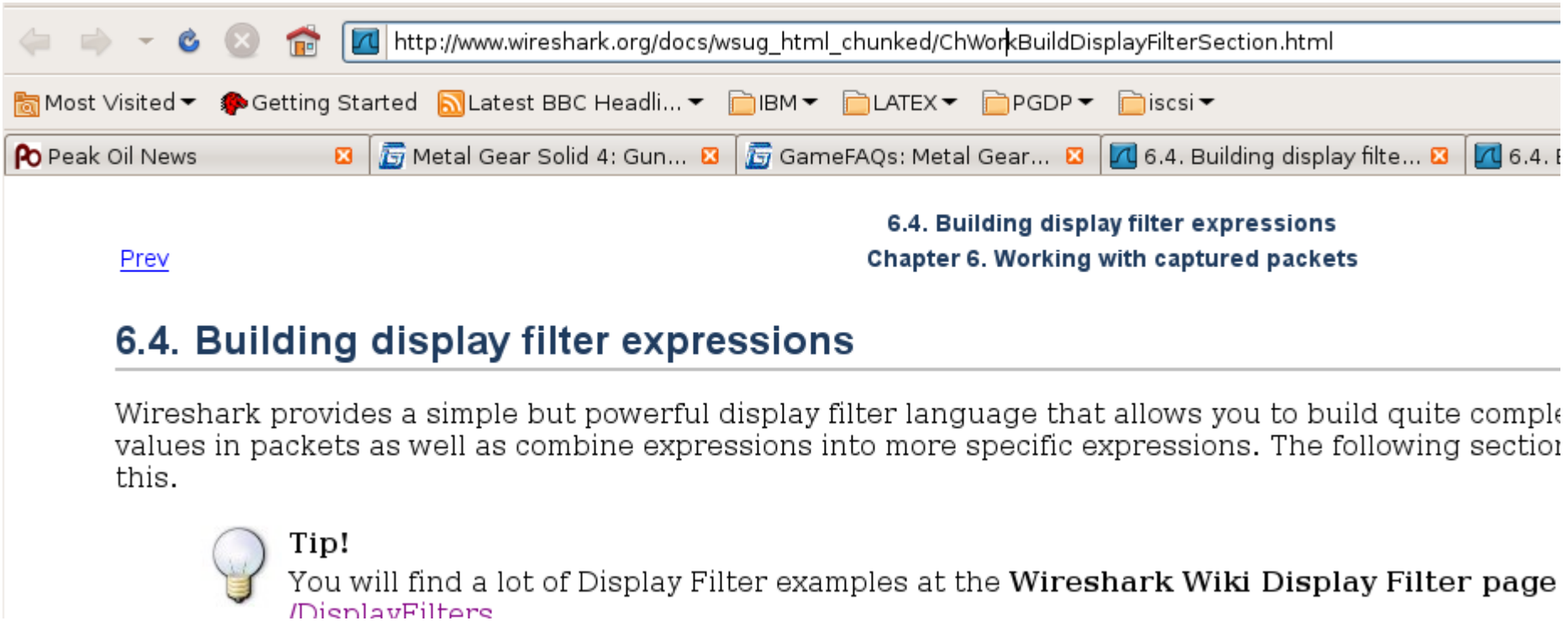
CAPTURE Filters

CAPTURE FILTERS

- Capture filters are only used when capturing traffic. It uses the standard "TCPDUMP/LIBPCAP" language.
- This is a very simple language designed to be provable correct, loop-free and fast.
  The only time this is used is when you capture traffic.
- See www.tcpdump.org for more information.
  - You can use Wireshark very efficiently and never use this language.

DISPLAY Filters

DISPLAY FILTERS



DISPLAY FILTERS

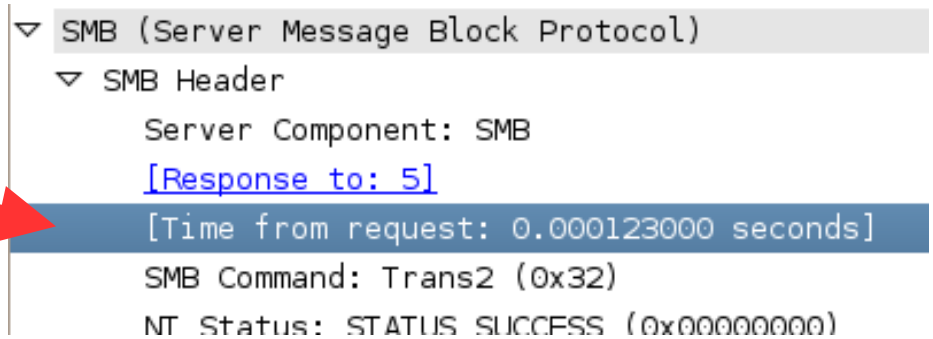
- Display filters are very very slow but powerful.
- During filtering Wireshark drills down into each packet and assigns a filter fieldvariable to each individual bit/byte.
- There is no silly filters of the type “byte at offset X has valye Y”. (That would be a way to do FAST filtering but it doesn't work)

DISPLAY FILTERS

- There are > 70.000 different filter variables right now. And increasing by a lot every week.
- There is no “list” of the fields.
- How to find out which fields to use?

DISPLAY FILTERS

Select a field in the decode pane




```
0050   00 00 01 00 24 08 64 00   81 e8 0a 02 00 08 00 00    ....$.d. .
0060   00 02 00 38 00 00 00 08   00 3c 00 00 00 00 00 0d    ...8.... .
0070   00 00 00 00 00 00 4f 22   04 00 16 00 00 00          ......O" .
```

Time between Request and Response for SMB cmds (smb.time)

And Wireshark will tell you the filter variable name of the selected object

DISPLAY FILTERS

- Noticed that the field clicked was inside brackets?
- This means it was an artificial field.
- This field does not represent any particular bit or byte in the packet that was captured it represents a relation to another packet.

- Wireshark generates a lot of artificial fields. Thats what makes Wireshark good!

RESPONSE TIMES

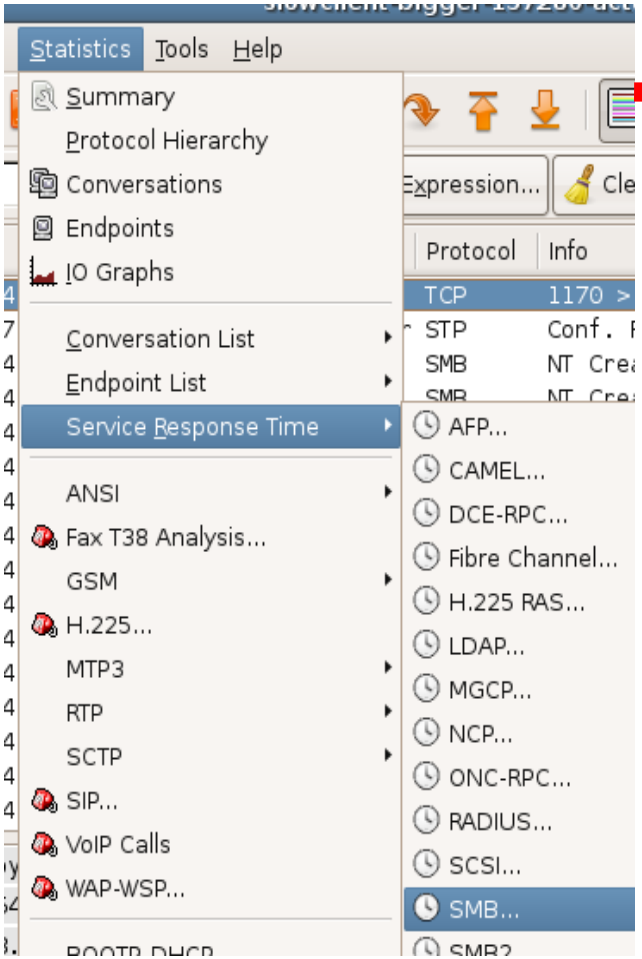
RESPONSE TIMES

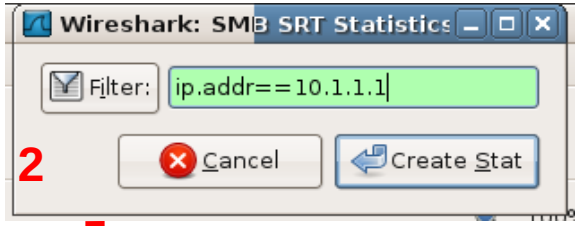
- Wireshark will match all SMB and SMB2 requests and responses and calculate the response time based on the timestamps.

- These are then reported as an artificial filter field for each individual request/response pair.

RESPONSE TIMES

1





3

SMB Service Response Time statistics

Filter:

SMB Commands

Index	Procedure	Calls	Min SRT	Max SRT	Avg SRT
47	Write AndX	2610	0.00024	0.09993	0.00070
162	NT Create AndX	21	0.00060	0.00080	0.00073
50	Trans2	21	0.00011	0.00012	0.00011
4	Close	21	0.00023	0.00067	0.00039

Transaction2 Sub-Commands

Index	Procedure	Calls	Min SRT	Max SRT	Avg SRT
7	QUERY_FILE_INFO	21	0.00011	0.00012	0.00011

RESPONSE TIMES

- Most SMB operations can be divided into 4 types :
  DATA READ
  DATA WRITE
  METADATA READ
  METADATA WRITE
- Identifying which type the “fast” and the “slow” operations belong to can sometimes identify performance bottlenecks.

RESPONSE TIMES


```
tshark -n -r smb.cap -q -z smb,rtt

================================================================================
SMB RTT Statistics:
Filter:
Commands                     Calls   Min RTT   Max RTT   Avg RTT
Close                        21   0.00023   0.00067   0.00039
Write AndX                   2610   0.00024   0.09993   0.00070
NT Create AndX               21   0.00060   0.00080   0.00073

Transaction2 Commands        Calls   Min RTT   Max RTT   Avg RTT
QUERY_FILE_INFO              21   0.00011   0.00012   0.00011

NT Transaction Commands      Calls   Min RTT   Max RTT   Avg RTT
================================================================================
```

Filehandles

Filehandles

- Being a network filesystem, most interesting operations in SMB refer to filehandles, or FIDs as they are called in SMB.
- Normally in an SMB[2] Transaction the FID will only occur once. Either in the request or in the response but not in both.
- This makes filtering on “files” almost useless!

Filehandles

- Fortunately Wireshark is a bit smarter than than and understands that filehandles are special and that when you specify a filter for them, you want the filter to operate on BOTH the request and the response packet!

Filehandles

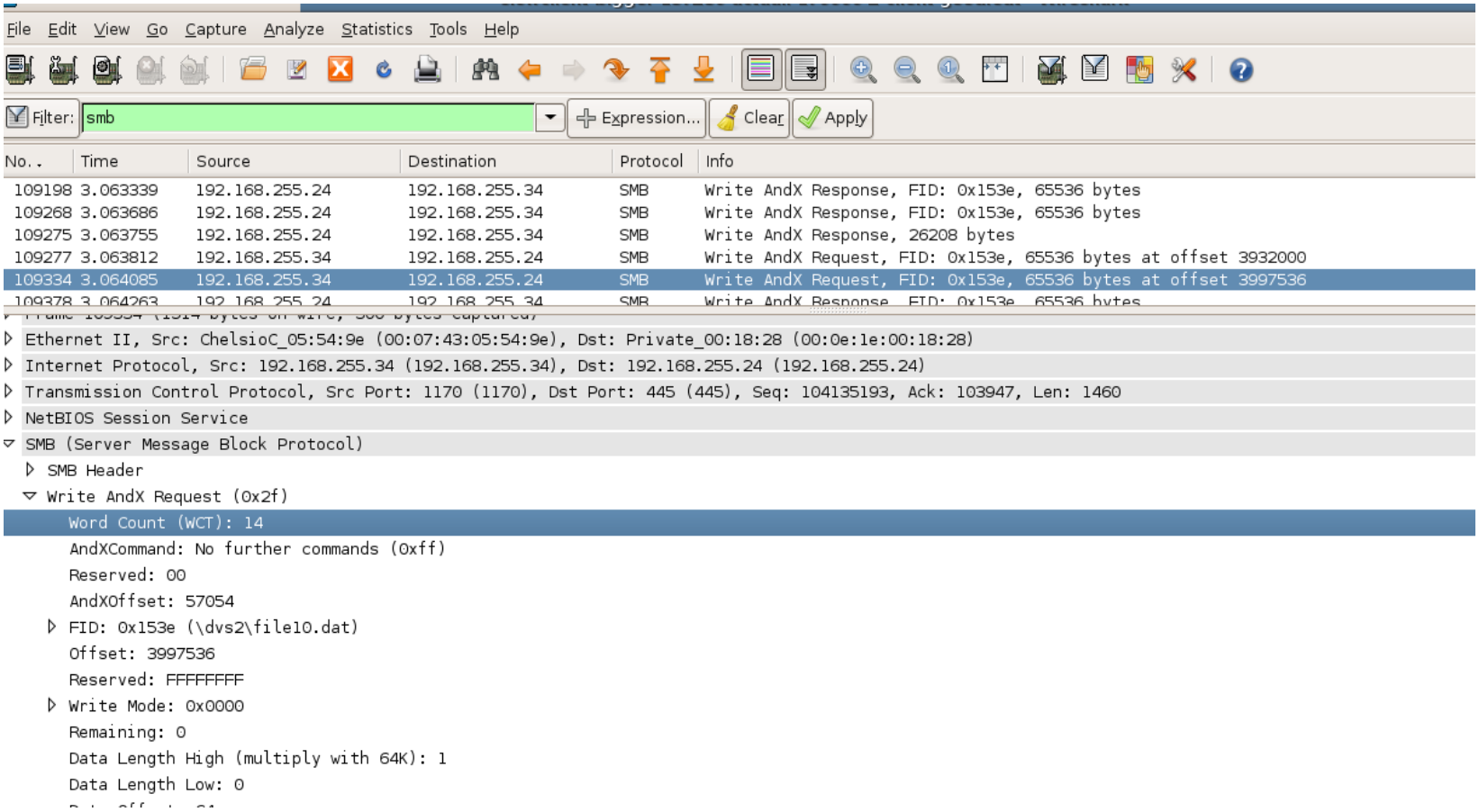

```
Client Request
==============
UCHAR WordCount;
UCHAR AndXCommand;
UCHAR AndXReserved;
USHORT AndXOffset;
USHORT Fid;
ULONG Offset;
ULONG Reserved;
USHORT WriteMode;

USHORT Remaining;
USHORT DataLengthHigh;

USHORT DataLength;
USHORT DataOffset;
ULONG OffsetHigh;

USHORT ByteCount;

UCHAR Pad[];
UCHAR Data[DataLength];
```

```
Server Response
===============
UCHAR WordCount;
UCHAR AndXCommand;
UCHAR AndXReserved;
USHORT AndXOffset;
USHORT Count;
USHORT Remaining;
ULONG Reserved;
USHORT ByteCount;
```

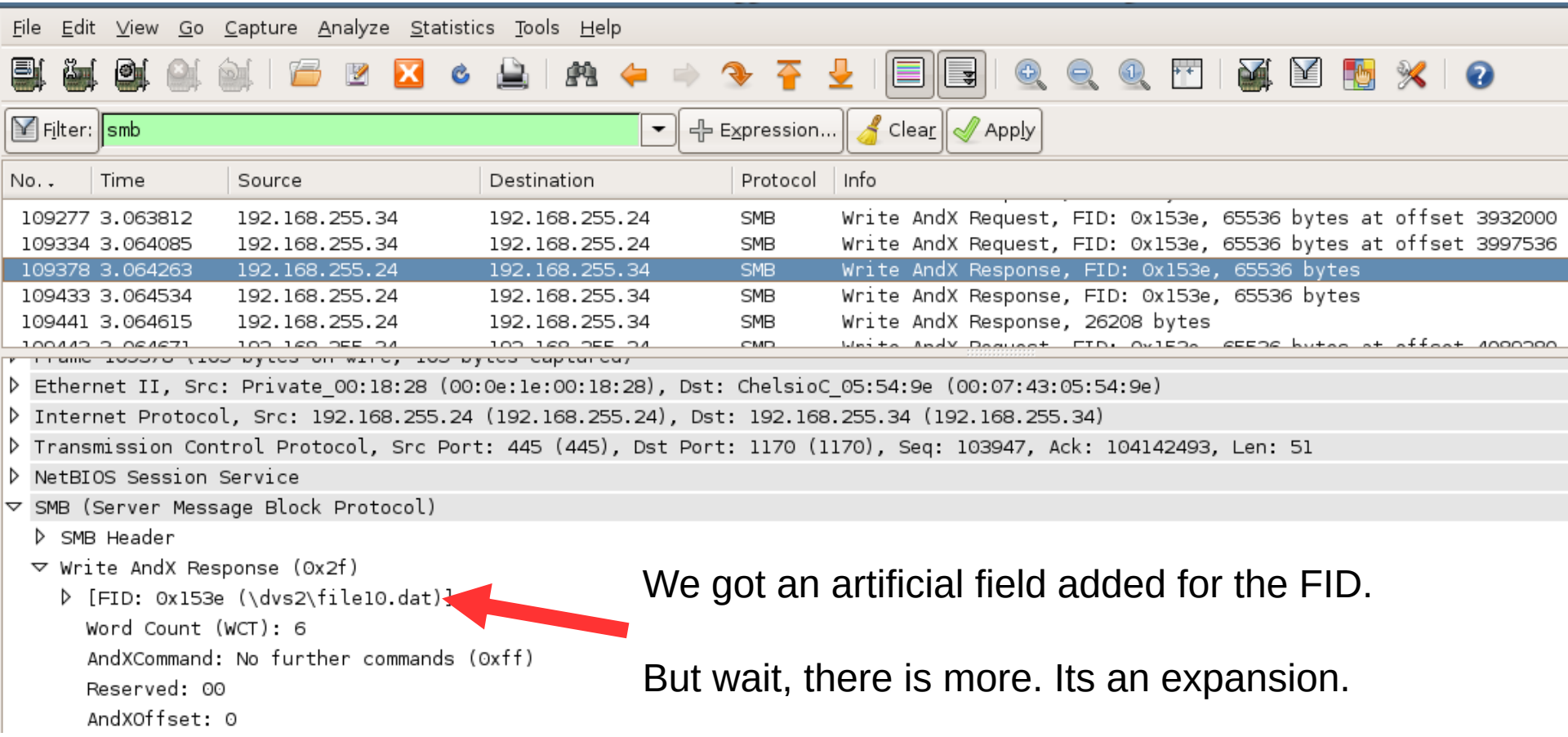
Even worse. Neither the request nor the response contains the FILE-NAME which is infinitely much nicer to use when analyzing traces than some random 16-bit integer FID :-(

Filehandles

Wireshark to the rescue...



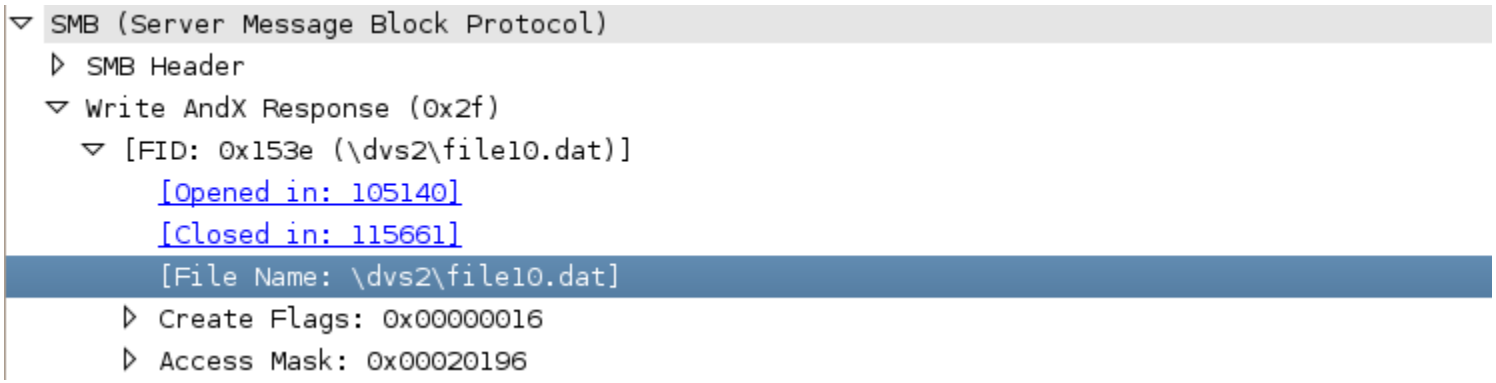
Filehandles



We got an artificial field added for the FID.

But wait, there is more. Its an expansion.

Filehandles



Everytime that FID is encountered in the trace Wireshark will add information about how and when the FID was opened as well as the actual filename. This is added to both the request and the response.

Thus if we apply a filter such as smb.file == "\\dvs2\\file10.dat" it will actually work and do what we want it to do !

This is pretty unique to wireshark. And VERY useful.

Filehandles

SMB Service Response Time statistics
Filter: smb.file == "\\dvs2\\file10.dat"

SMB Commands

Index	Procedure	Calls	Min SRT	Max SRT	Avg SRT
47	Write AndX	130	0.00024	0.00145	0.00048
162	NT Create AndX	1	0.00075	0.00075	0.00075
50	Trans2	1	0.00011	0.00011	0.00011
4	Close	1	0.00037	0.00037	0.00037

Transaction2 Sub-Commands

Index	Procedure	Calls	Min SRT	Max SRT	Avg SRT
7	QUERY_FILE_INFO	1	0.00011	0.00011	0.00011

Response times for SMB for one specific file!

Filehandles

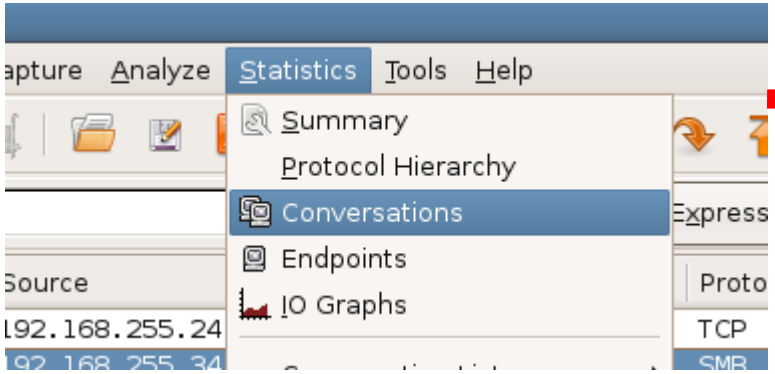

```
tshark -n -r smb.cap -q -z 'smb,rtt,smb.file=="\\dvs2\\file10.dat"'

================================================================
SMB RTT Statistics:
Filter: smb.file=="\\dvs2\\file10.dat"
Commands                 Calls   Min RTT   Max RTT   Avg RTT
Close                        1   0.00037   0.00037   0.00037
Write AndX                 130   0.00024   0.00145   0.00048
NT Create AndX               1   0.00075   0.00075   0.00075

Transaction2 Commands     Calls   Min RTT   Max RTT   Avg RTT
QUERY_FILE_INFO               1   0.00011   0.00011   0.00011

NT Transaction Commands    Calls   Min RTT   Max RTT   Avg RTT
```

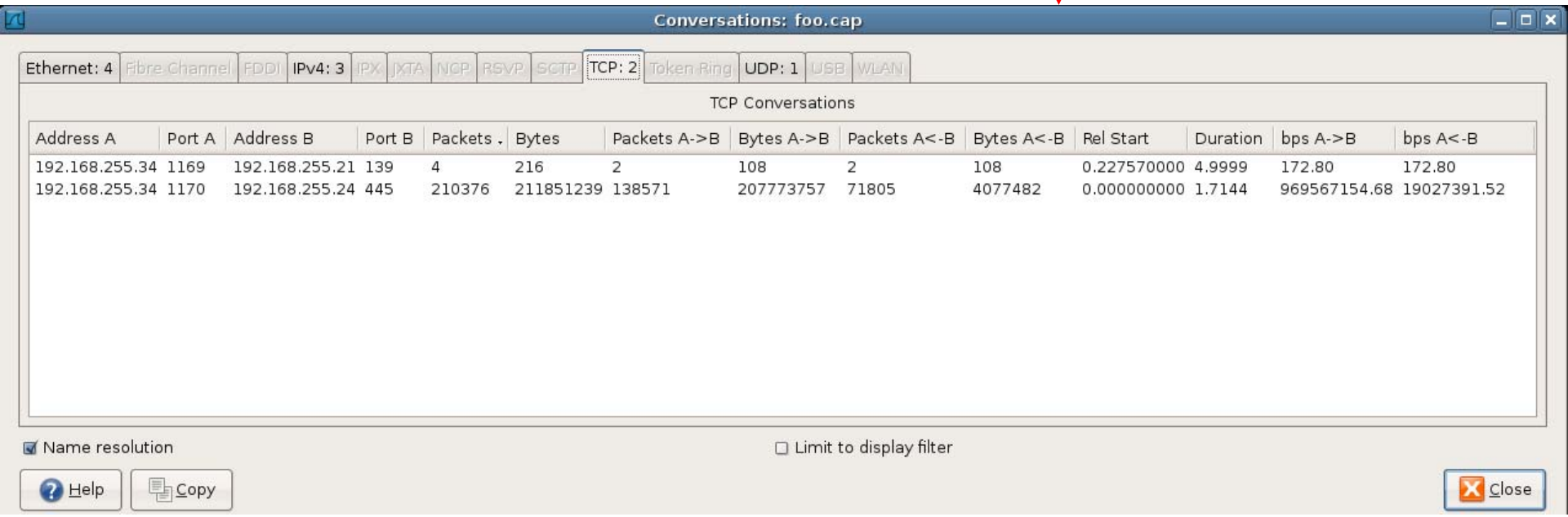
CONVERSATIONS

CONVERSATIONS

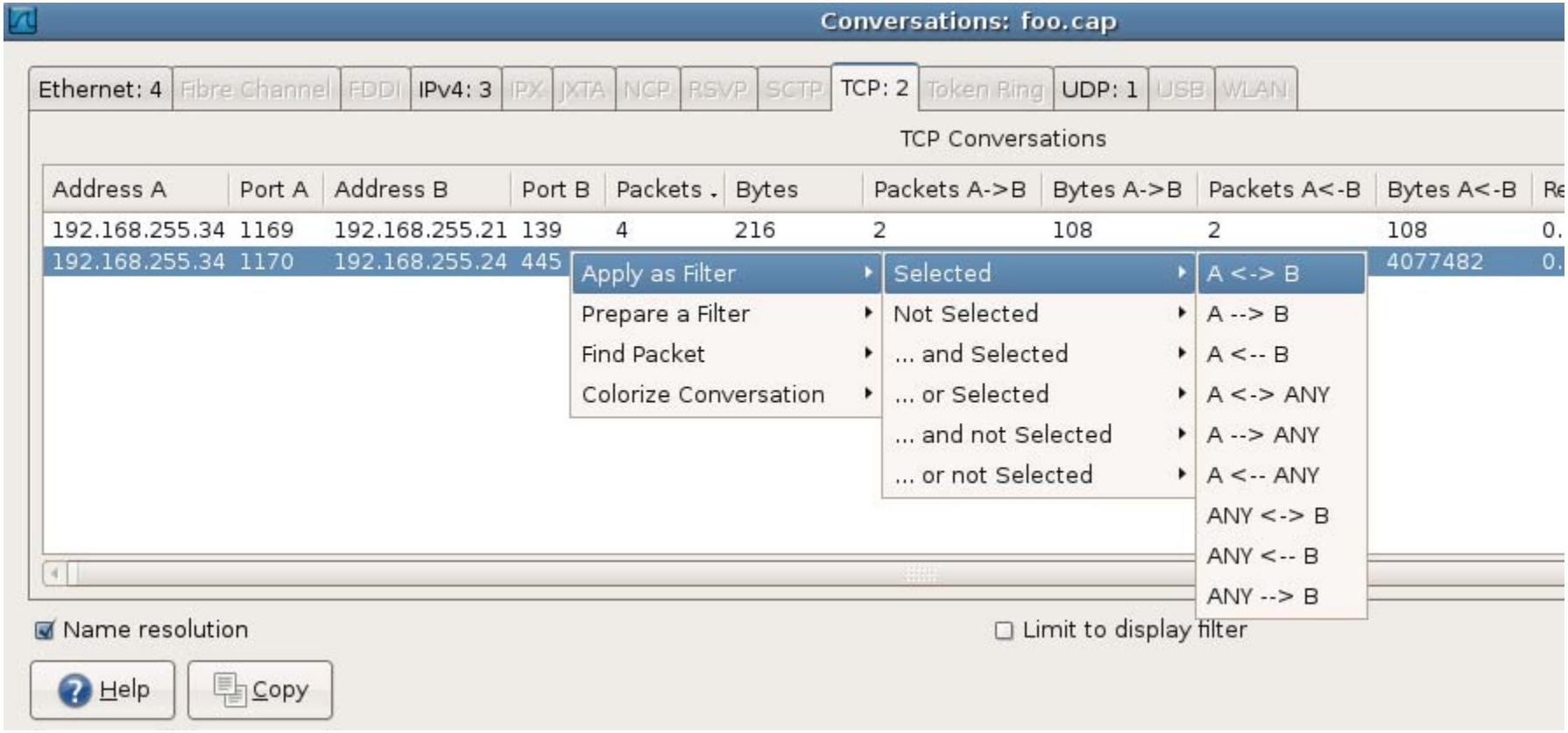
- Conversations is an easy way to get a list of all the TCP connections in a trace that are used for CIFS.
- This provides a quick overview of all CIFS clients and which clients are doing a lot of I/O.

CONVERSATIONS





CONVERSATIONS



I/O Graphs
(except for LOAD graphs)

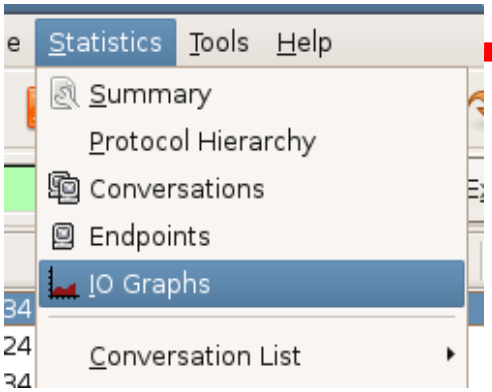
I/O Graphs

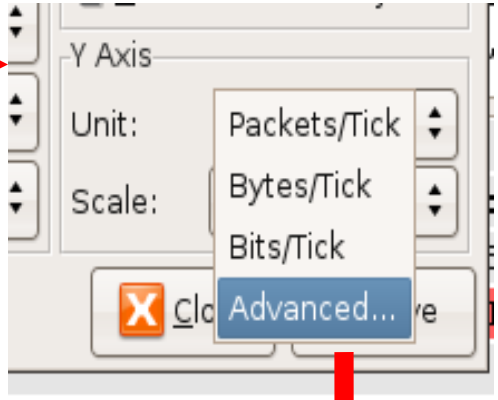
- I/O Graphs provides a mechanism to calculate many different types of statistics and plot them over time.
- I/O Graphs works with almost all of the ~70k display filter variables but only makes sense for a smaller subset of them.

I/O Graphs

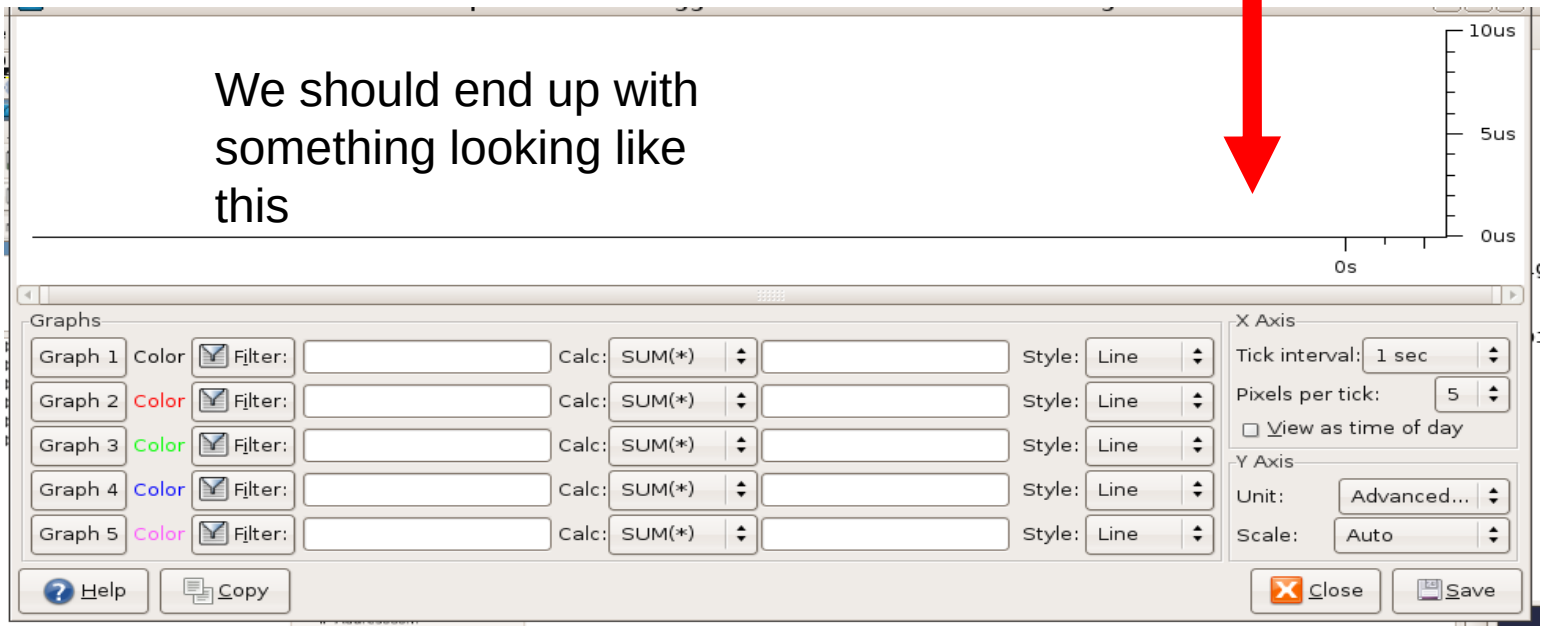
- The supported operations are :
  MAMIMUM
  MINUMUM
  AVERAGE
  SUM
  COUNT
  LOAD (more about this one later)

I/O Graphs

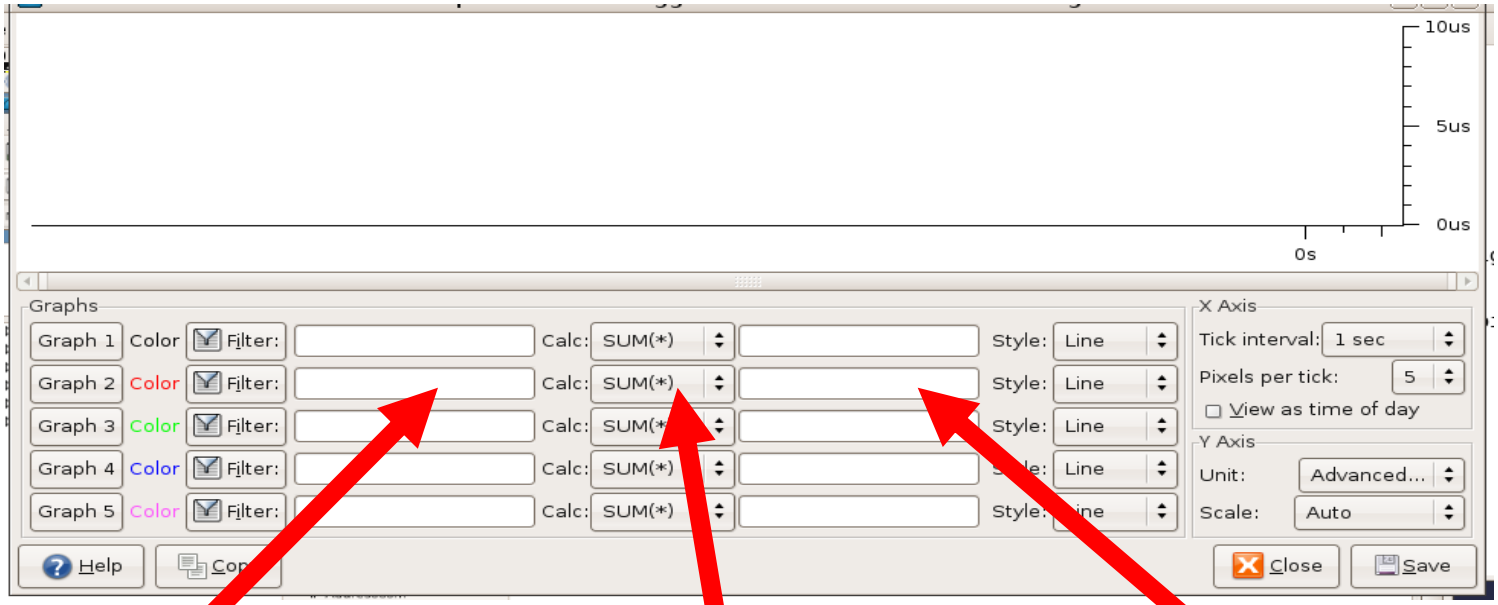




We should end up with something looking like this



I/O Graphs



Here goes a full blown display filter that control which subset of the trace we want to calculate over.
No filter means all packets.

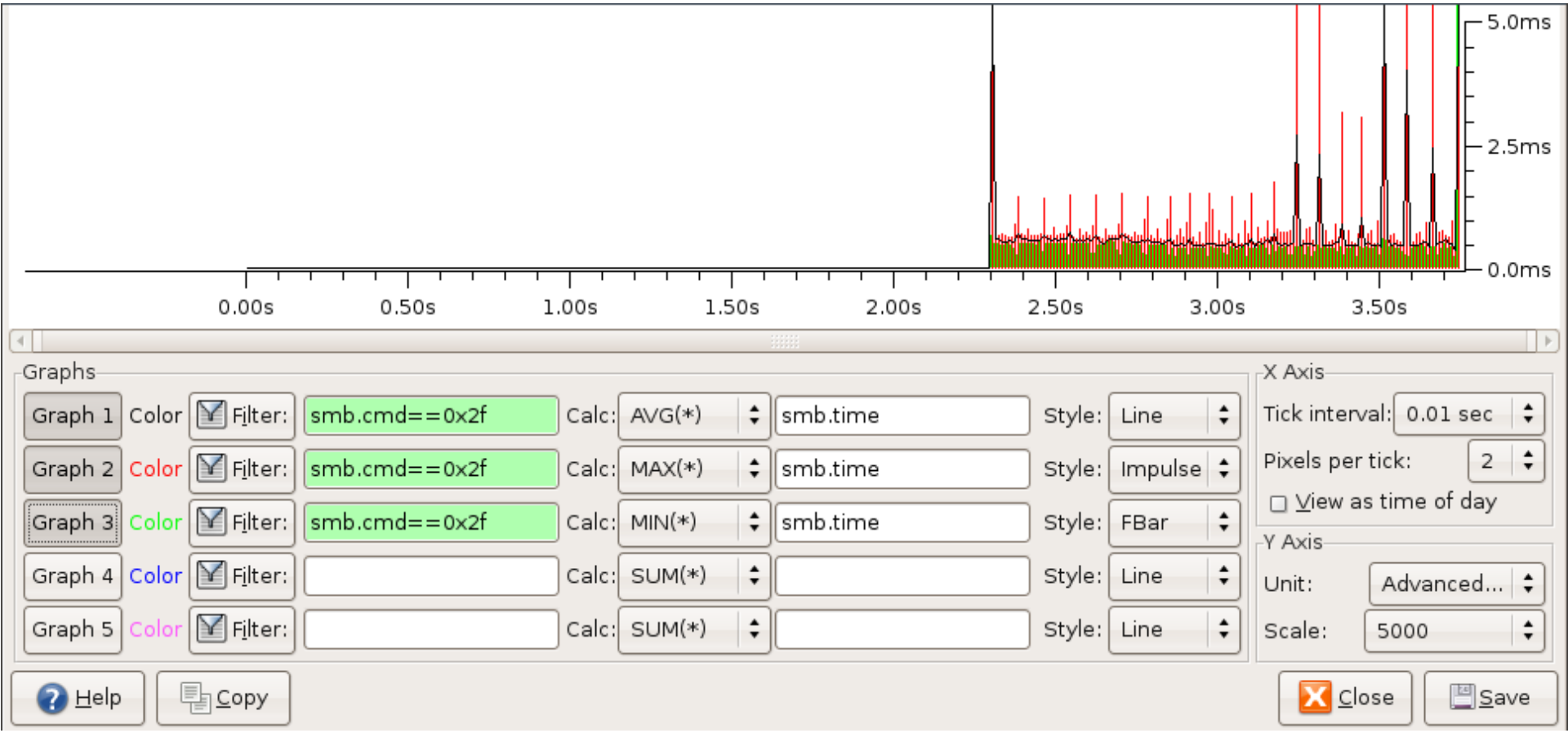
Here goes the name of the display filter variable to use.

Here is the type of operation to graph.

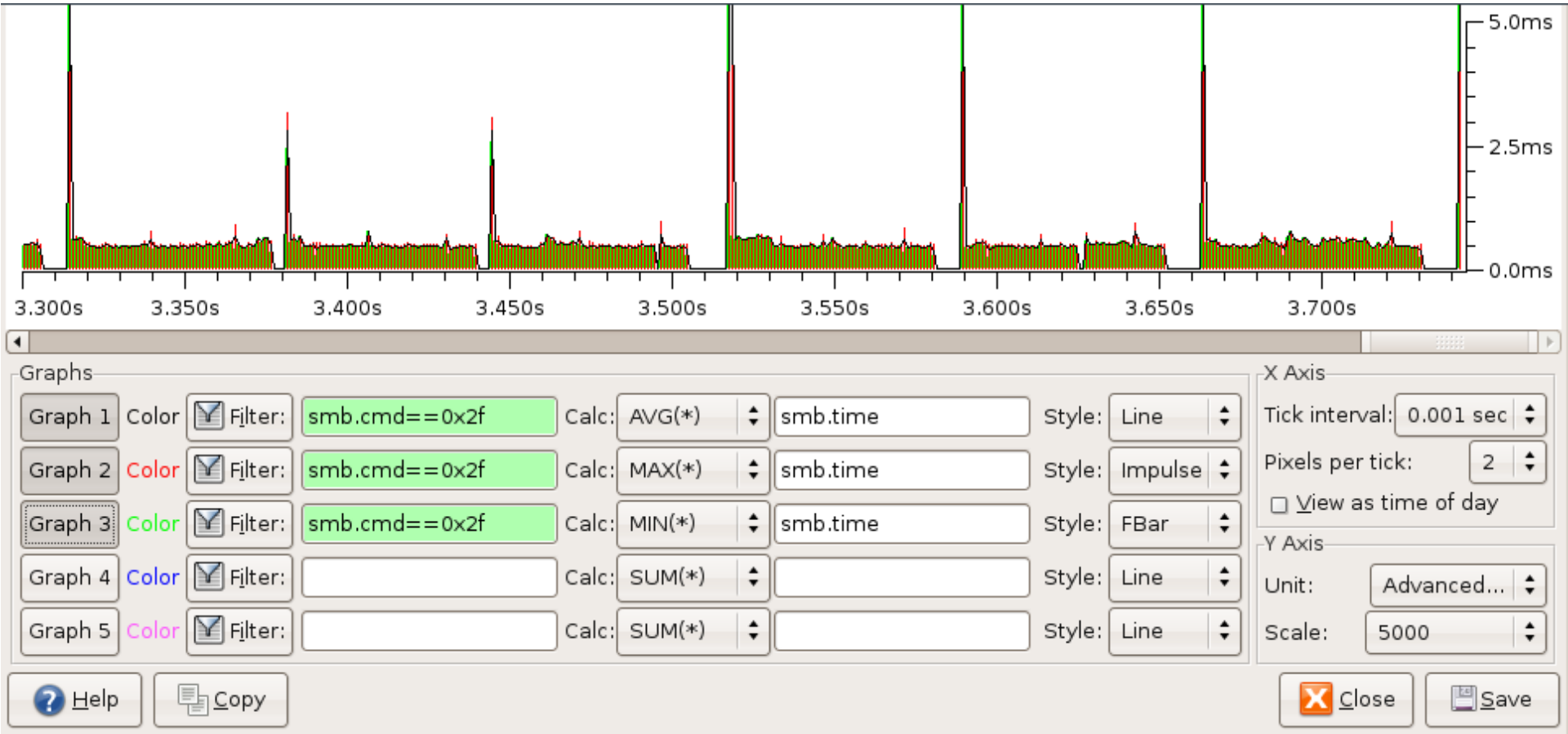
I/O Graphs

- As an example. Lets plot the MIN/MAX/AVG SMB response times for SMB WRITES to the file \svd2\file10.dat from the client with ip address 10.1.1.1
- This would require a display filter string such as : ip.addr==10.1.1.1 && smb.file==”\\dvs2\\file10.dat” && smb.cmd==0x2f

I/O Graphs



I/O Graphs



I/O Graphs

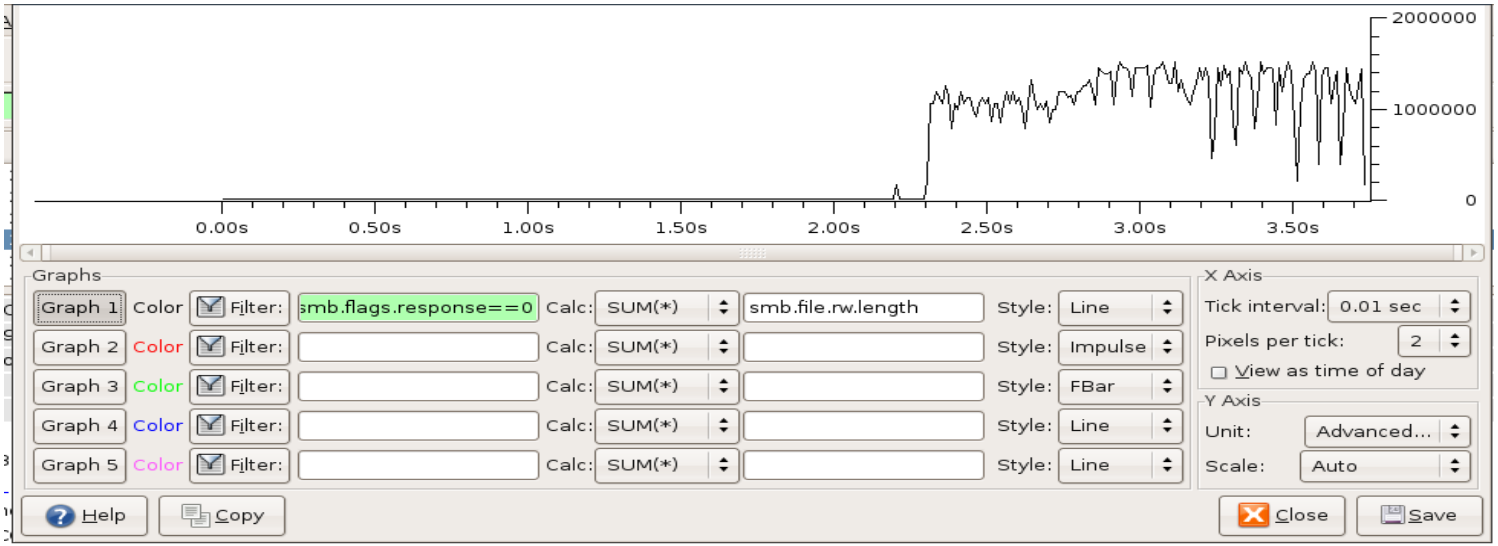
- -z "io,stat,0.01,MIN(smb.time)smb.time,MAX(smb.time)smb.time,AVG(smb.time)smb.time"

  To generate a nice table for SMB in tshark:

```
================================================================================
IO Statistics
Interval: 0.010 secs
Column #0: MIN(smb.time)smb.time
Column #1: MAX(smb.time)smb.time
Column #2: AVG(smb.time)smb.time
                |   Column #0    |   Column #1    |   Column #2
Time            |           MIN |           MAX |           AVG
000.000-000.010             0.000           0.000           0.000

...
```

I/O Graphs

Filtering on smb.cmd==0x2f && smb.flags.response==0 i.e. SMB WRITES



The client has serious problems keeping a WRITE rate of 150MByte/sec. That is very very very slow. There are also frequent spikes when the I/O rate drops dramatically.
Something is seriously wrong here!

TCP

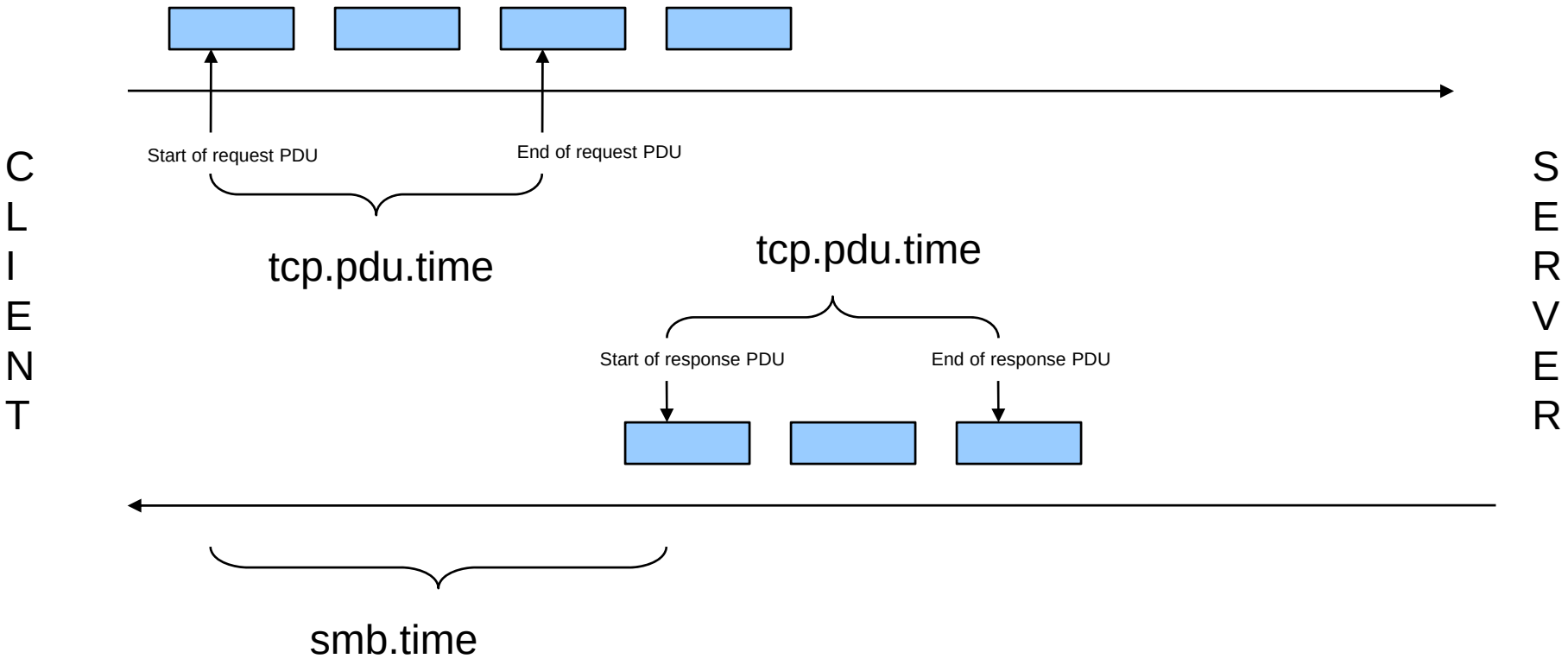
TCP

- If a performance issue is caused by the network, this can often be found by looking at the TCP layer.
- A list of useful things to look for in the TCP layer.

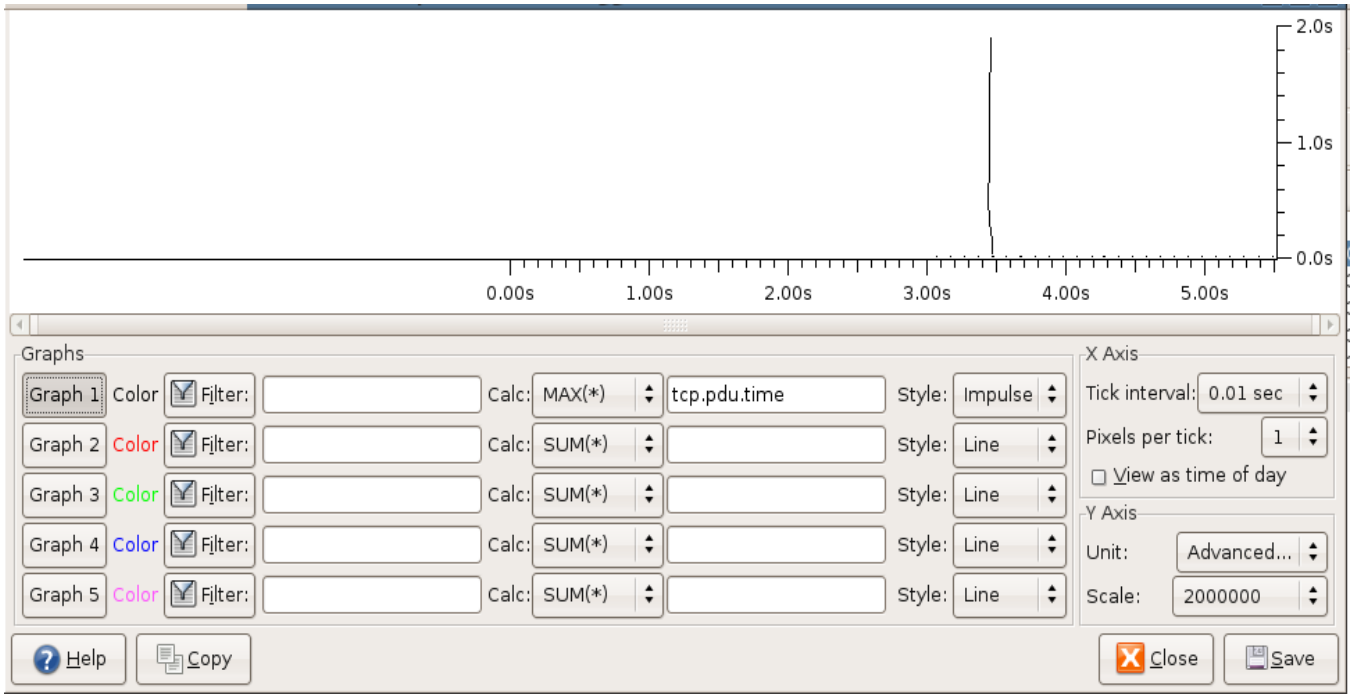
TCP

- Wireshark tracks the sequence numbers where a higher layer PDU starts and stops and can measure how long it took to transfer a PDU !
- The filter field is called tcp.pdu.time

Two trains of ethernet frames between the client and the server

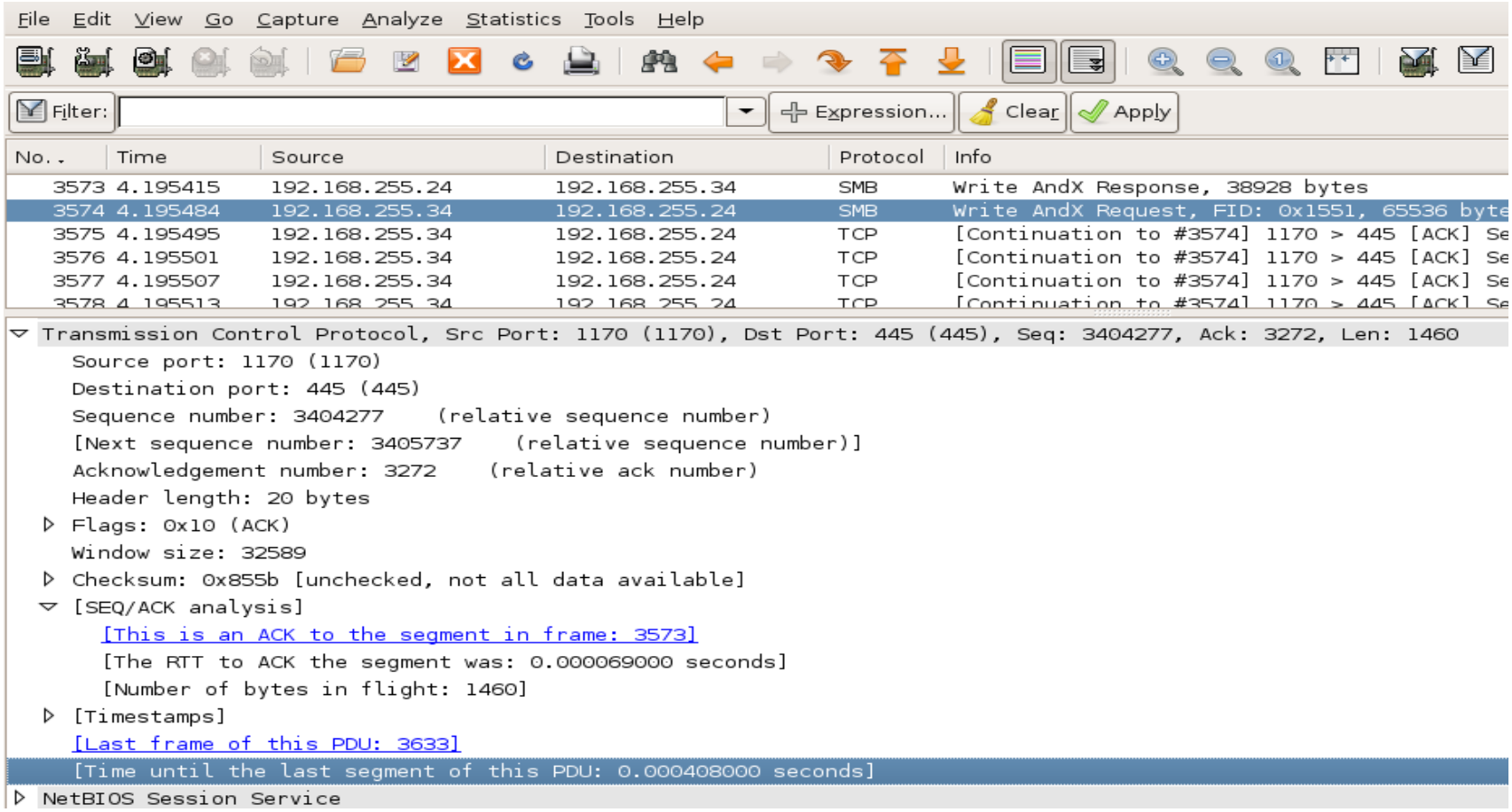


TCP



Not good. One PDU took almost 2 seconds to transfer across the network.

TCP

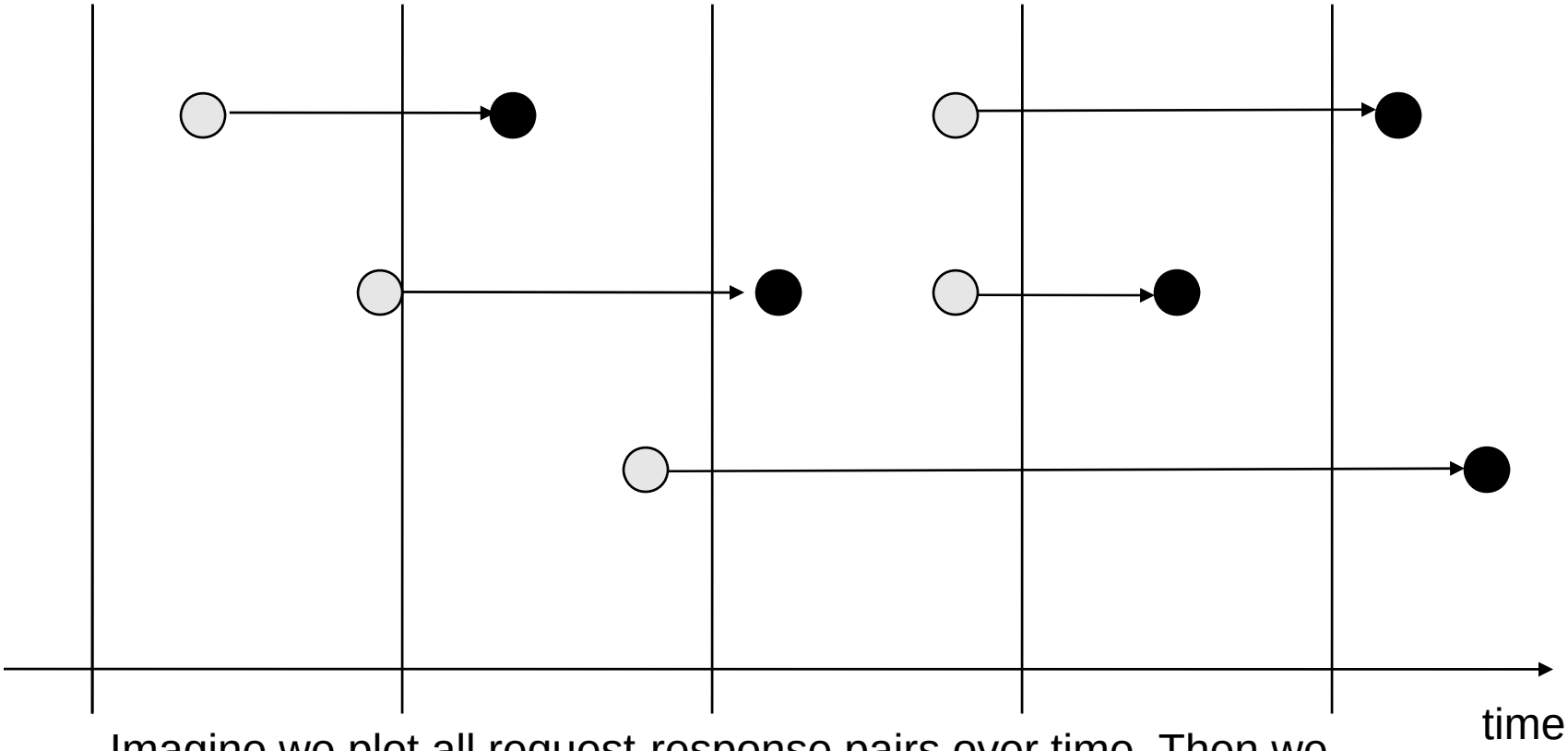


I/O LOAD Graphs

LOAD graphs

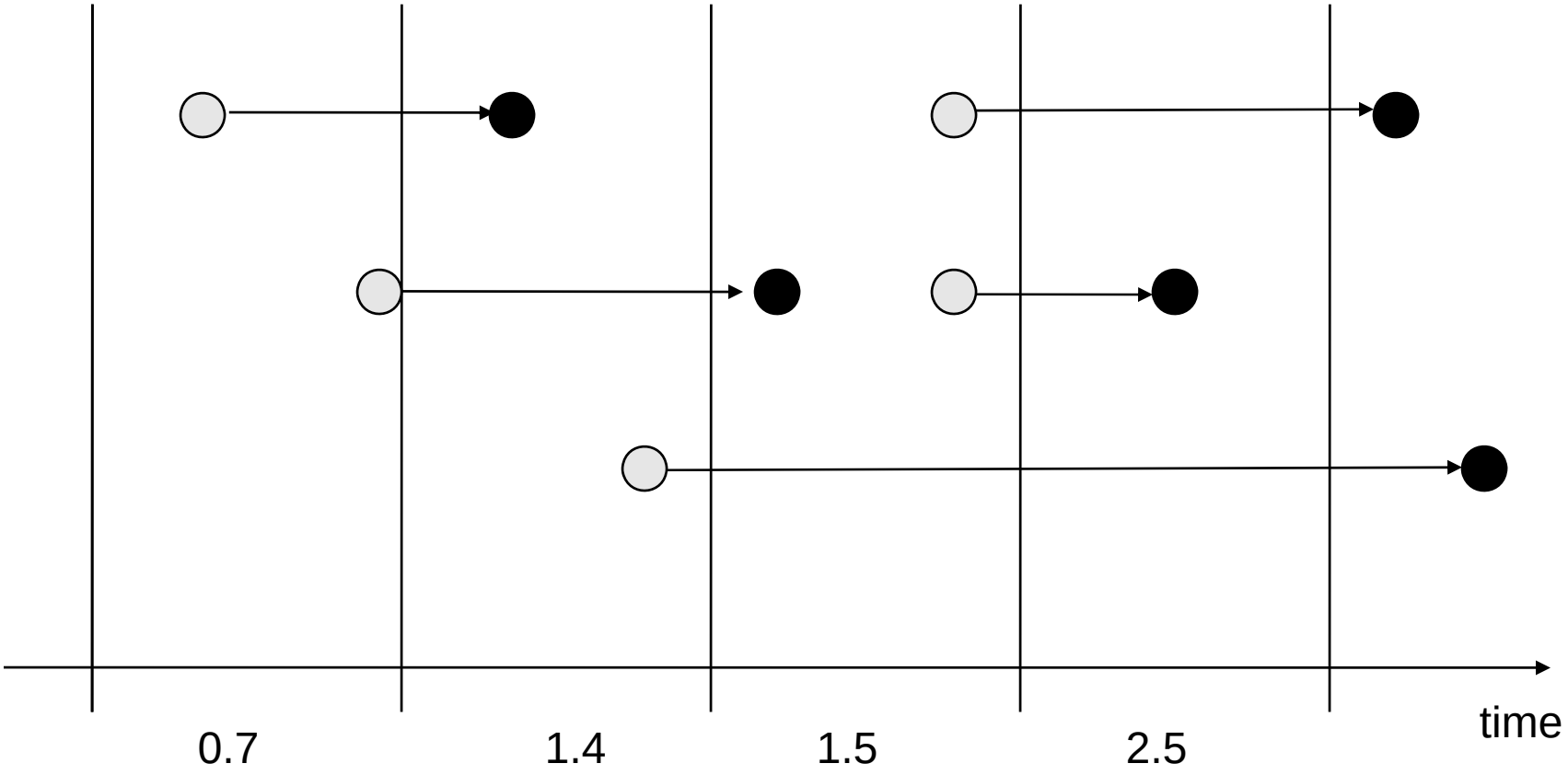
- LOAD graphs are a special kind of I/O graphs.
- These graphs operate only on fields that are response time fields such as smb.time, rpc.time, scsi.time etc.
- What they then plot is the QUEUE-depth of the connection over time.

LOAD graphs



Imagine we plot all request-response pairs over time. Then we "estimate" what the average amount of coverage is per "interval". This then becomes out "average" queue-depth for that interval.

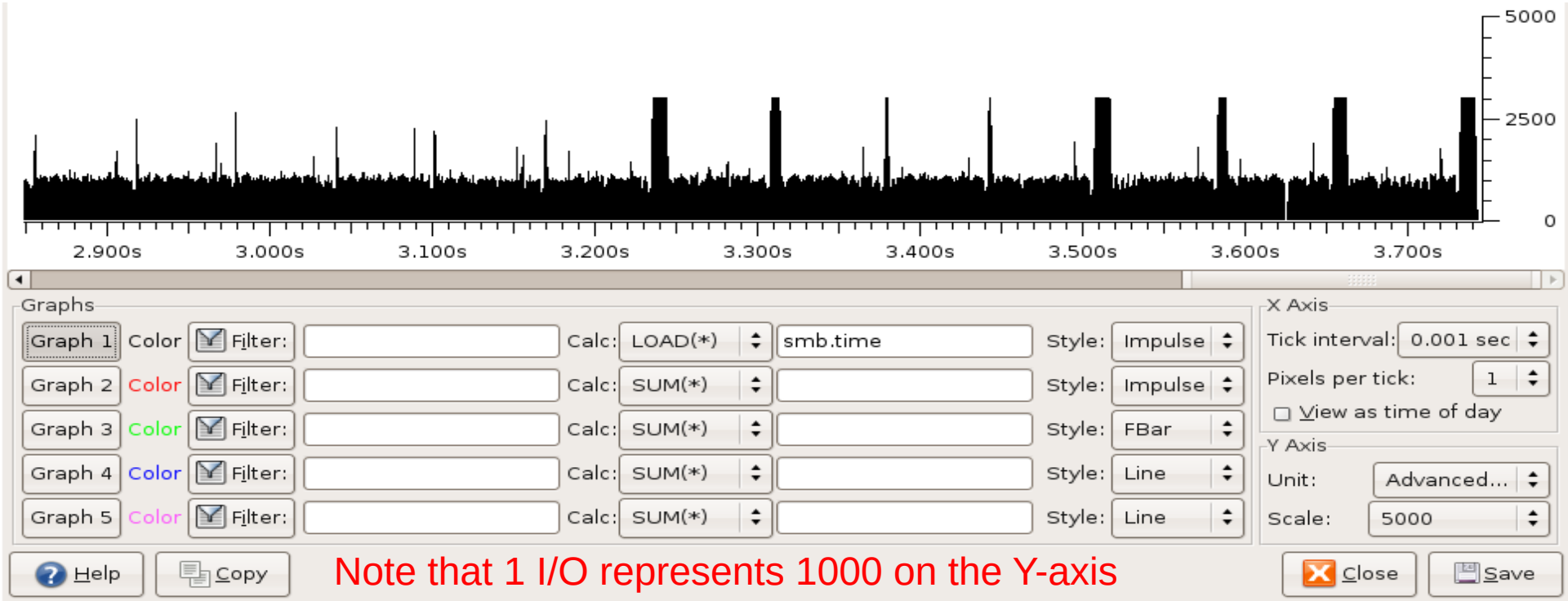
LOAD graphs



LOAD graphs

- This is very useful since this is the only means we have of viewing client-performance.
- The fluctuations in queue-depth are caused by the client issuing commands (queue increases) and the server completing them (queue shrinks).
- This allows us a way to see the relative speeds between the client and the server.

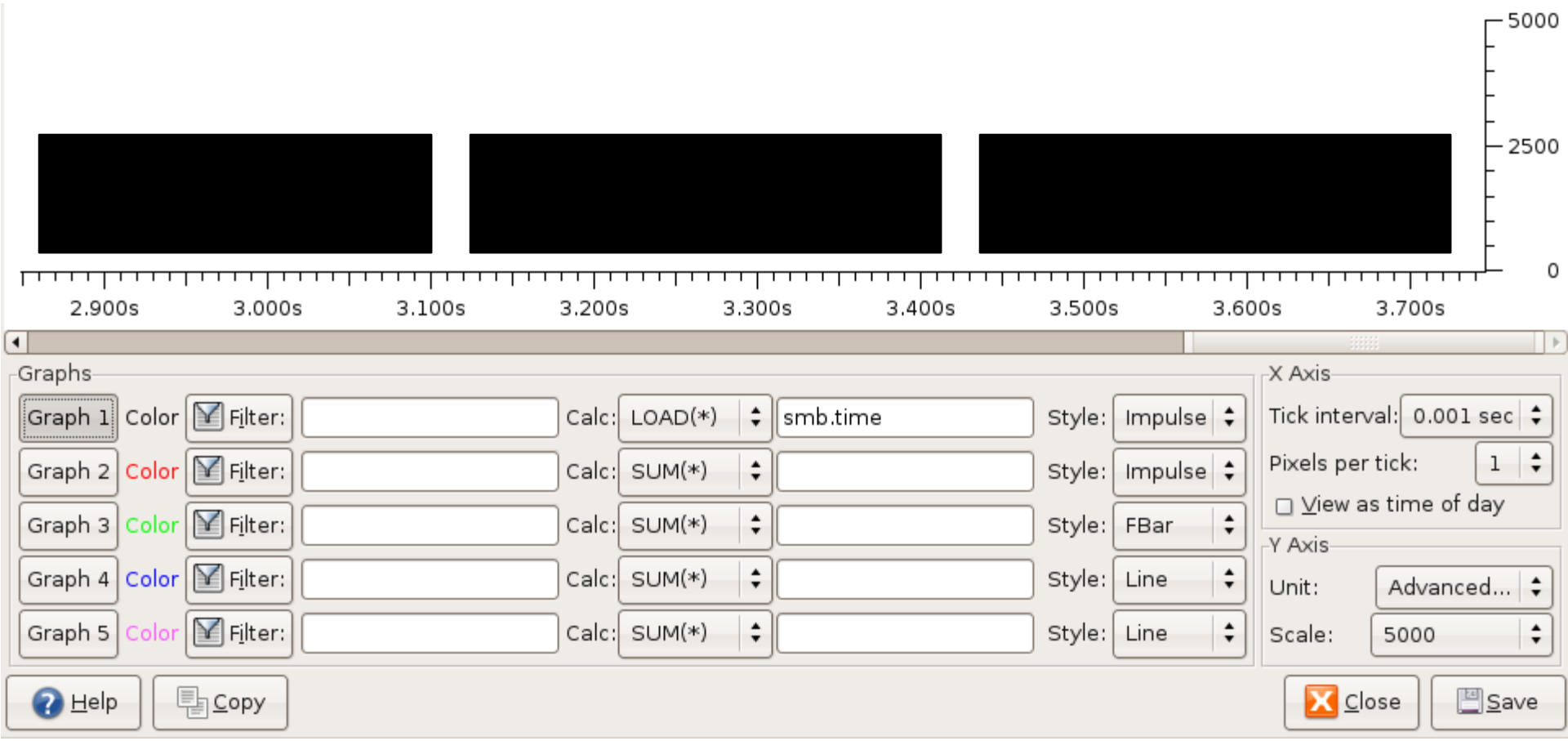
LOAD graphs



Note that 1 I/O represents 1000 on the Y-axis

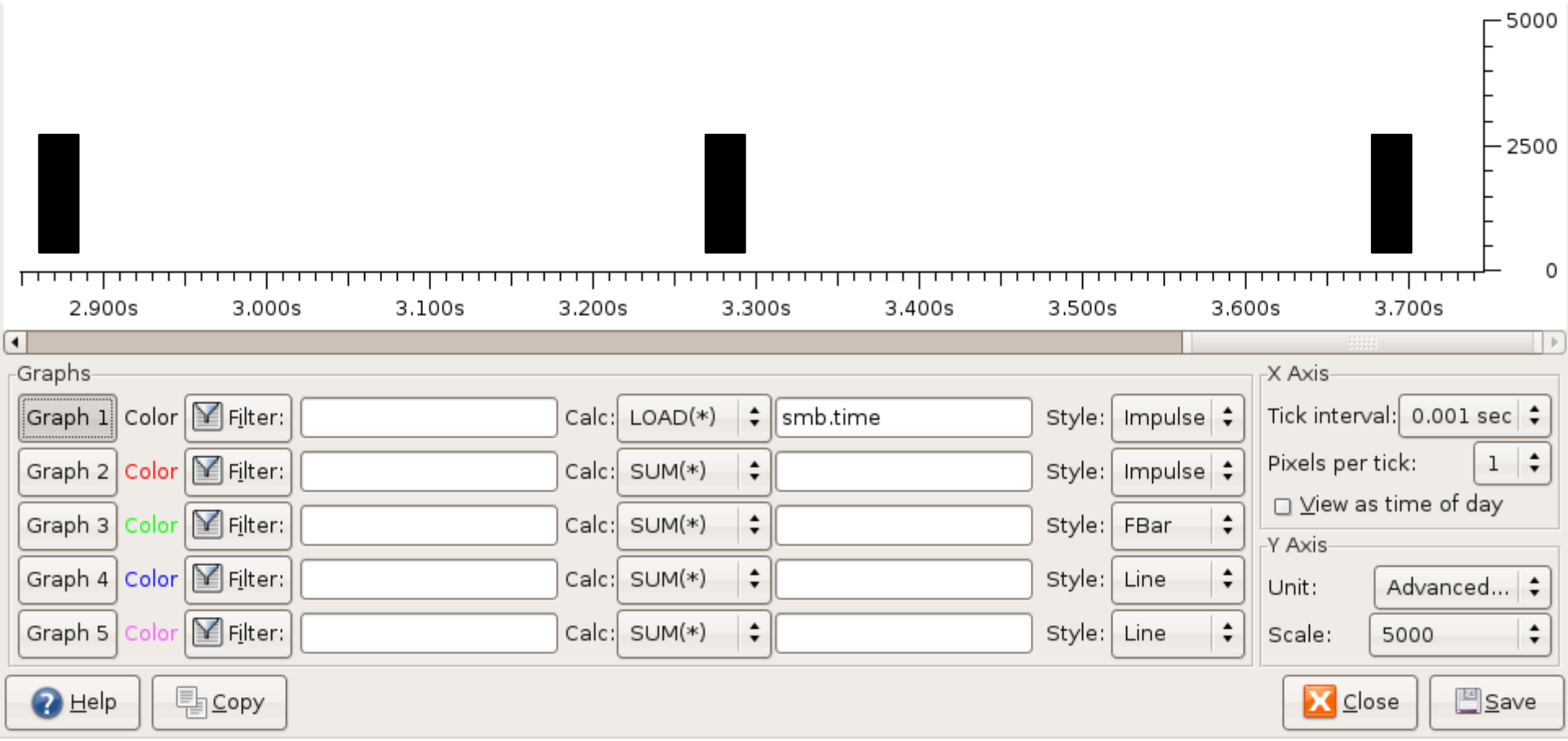
We see the client uses 3 threads. The client can on average barely keep 1 I/O in flight. Every ~80ms the server pauses for a few ms.

LOAD graphs



Typical slow server. The client quickly reaches its maximum queuedepth, then it takes long before the server responds and a new cycle can start.

LOAD graphs



Slow client. The server quickly completes any I/O and then sits idle for long periods waiting for the client to issue the next I/O.

LOAD graphs

- QUEUE DEPTH analysis can also be done by tshark.

Kerberos

Kerberos

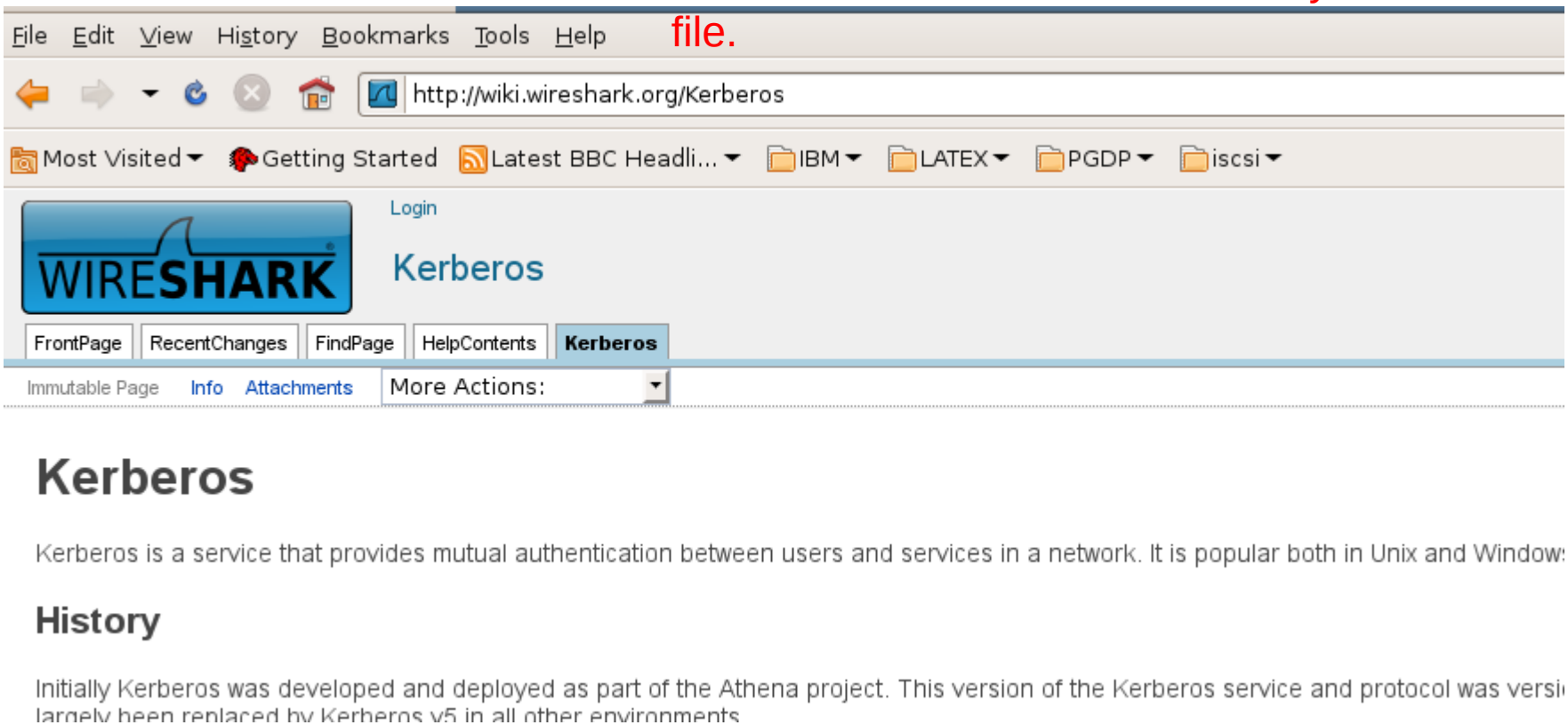
- Virtually all CIFS today uses Kerberos which makes life difficult for us when troubleshooting.
- We can see that someone got “ACCESS DENIED” but dont really know which user credentials this happened for.

Kerberos

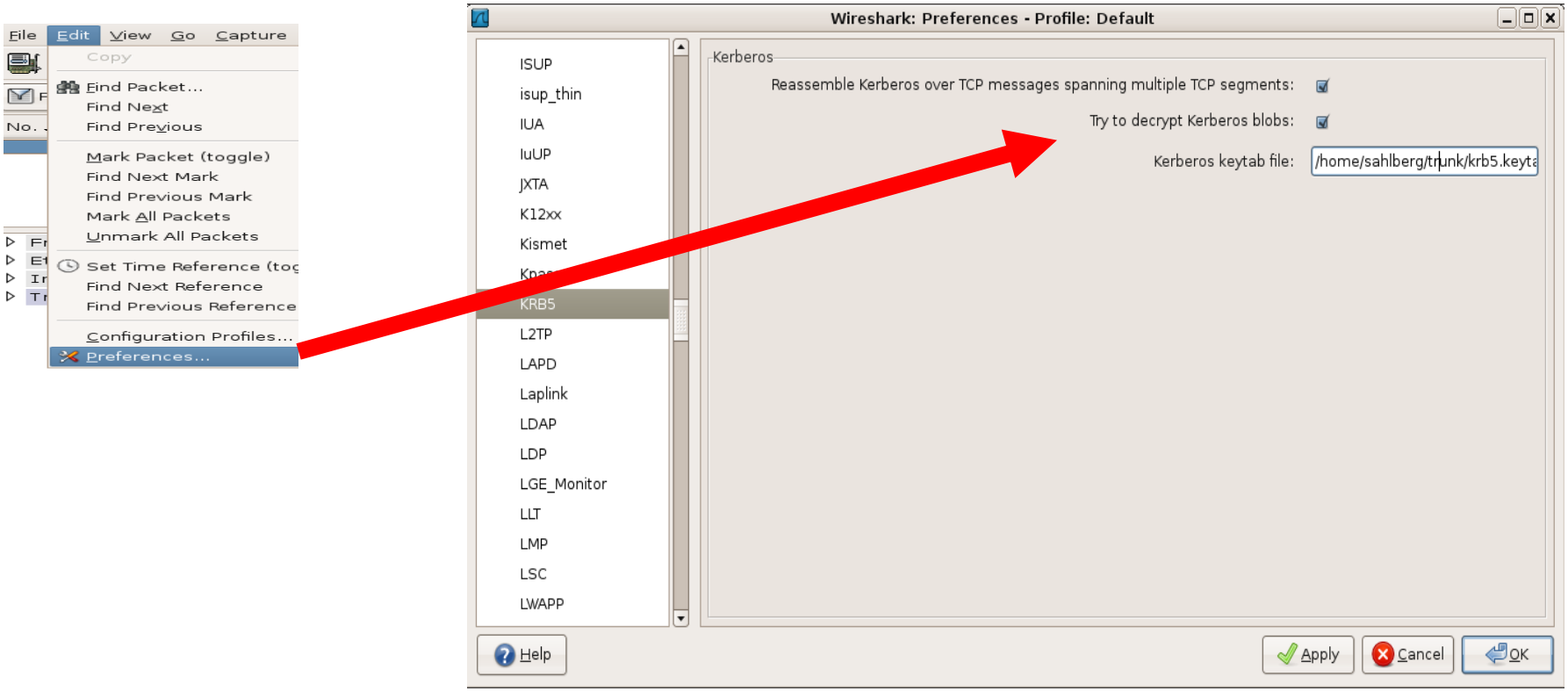
- If you can provide a keytab file containig the account secret, Wireshark can decrypt Kerberos and decode the username that is used in the initial SessionSetup handshake.
- This is very useful.
- It also works for SecureLDAP and DCE/RPC interfaces.

Kerberos

Instructions on how to create a keytab file.

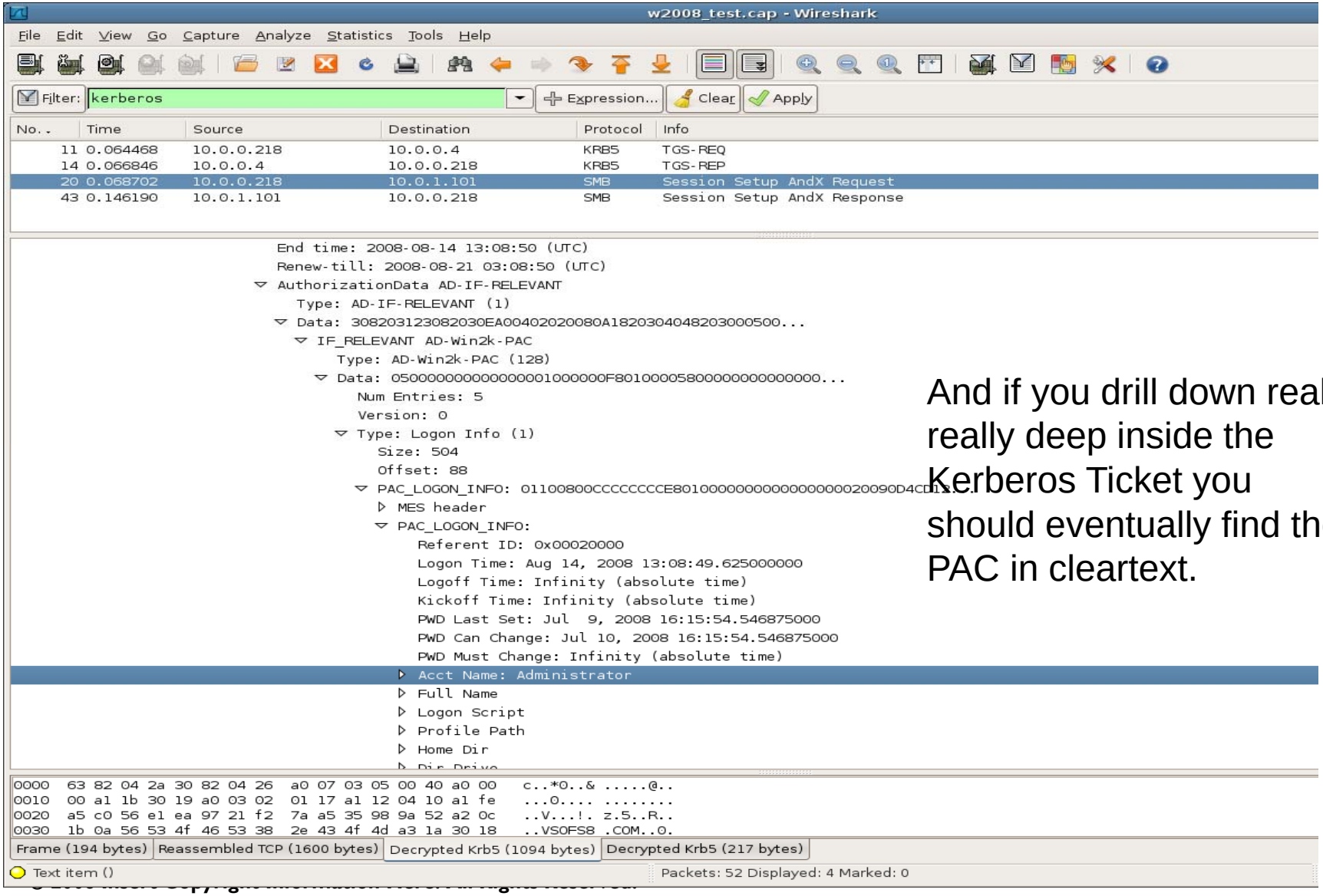


Kerberos



And then Apply and OK and restart wireshark.

Kerberos



And if you drill down really really deep inside the Kerberos Ticket you should eventually find the PAC in cleartext.

QnA
(if time permits :-)