

# Using Wireshark For Analyzing CIFS Traffic

Ronnie Sahlberg, Samba Team

# WHO AM I?

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

# INTRO

- ❑ 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.

## □ WWW.WIRESHARK.ORG

The screenshot shows the Wireshark website homepage. At the top is a blue banner with the Wireshark logo (a shark fin) and the word "WIRESHARK" in white. Below the banner is a navigation menu with links: "Wireshark", "Get Help", "Develop", "Tools", and "Buy". A dropdown menu is open under "Develop", listing "Overview", "Developer's Guide", "Browse the Code", and "Latest Builds". To the right of the navigation is a search box with a "Google Search" button. Below the navigation is a "Get Wireshark" section with a download icon and text: "Get it for OS X, Linux, Solaris, and others". To the right of this is a "PILOT" banner with the text "Taking Wireshark Into Uncharted Waters" and a list of features: "Analysis", "Charting", "Reporting", and "Integrated with Wireshark". Below the PILOT banner is a "CACE TECHNOLOGIES" logo. At the bottom right of the screenshot is a "Sharkfest Recap" section with the text: "Sharkfest was great! If you missed out, don't despair — you can catch up below: Videos from Sharkfest '08 are available at LoveMyTool.com, an online community for network monitoring and management tools. Presentations from each session are available on the official Sharkfest '08 page at CACE Technologies."

## □ WIKI.WIRESHARK.ORG



### Wireshark Wiki

This is the [wiki](#) site for the [Wireshark](#) network protocol analyzer.

You can edit any page by pressing the link at the bottom of the page, see [HowToEdit](#) for details. **If you want to try out wiki editing** [WikiWikiWeb](#) is, read about [WhyWikiWorks](#) and the [WikiNature](#). Also, consult the [WikiWikiWebFaq](#).

### General

- [HowToEdit](#): Information about how to edit the Wireshark wiki
- [WishList](#): Features we haven't implemented yet into Wireshark
- [KnownBugs](#): List of known Wireshark bugs
- [SampleCaptures](#): Sample capture files for your edification and amusement
- [NetworkTroubleshooting](#): Information about tracking down network problems

### Prepare Wireshark / TShark

# Intro : Wireshark vs the world

# What is 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 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!

- ❑ 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.

- ❑ 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.

- ❑ 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.

- ❑ 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?

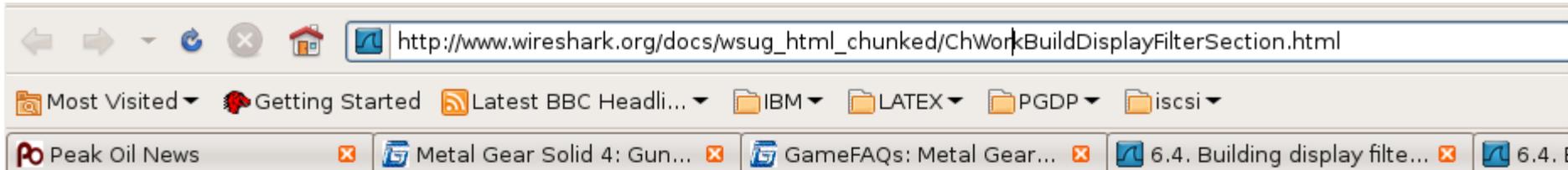
- ❑ Wireshark provides separate filter languages for different purposes:  
**CAPTURE FILTERS**  
and  
**DISPLAY FILTERS**
- ❑ Using Wireshark you will normally only use **DISPLAY FILTERS** but it is useful to know the distinction between them.

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

# 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](http://www.tcpdump.org) for more information.
- ❑ You can use Wireshark very efficiently and never use this language.

# DISPLAY Filters



## 6.4. Building display filter expressions

### Chapter 6. Working with captured packets

[Prev](#)

## 6.4. Building display filter expressions

Wireshark provides a simple but powerful display filter language that allows you to build quite complex values in packets as well as combine expressions into more specific expressions. The following section covers this.



### Tip!

You will find a lot of Display Filter examples at the [Wireshark Wiki Display Filter page](#)

- ❑ 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)

- ❑ 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

```
▼ SMB (Server Message Block Protocol)
  ▼ SMB Header
    Server Component: SMB
    [Response to: 5]
    [Time from request: 0.000123000 seconds]
    SMB Command: Trans2 (0x32)
    NT Status: STATUS_SUCCESS (0x00000000)
```

```
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 00  . . . . .0" .
```

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

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

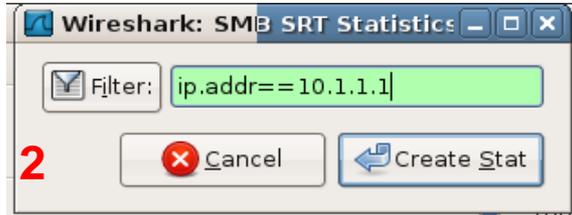
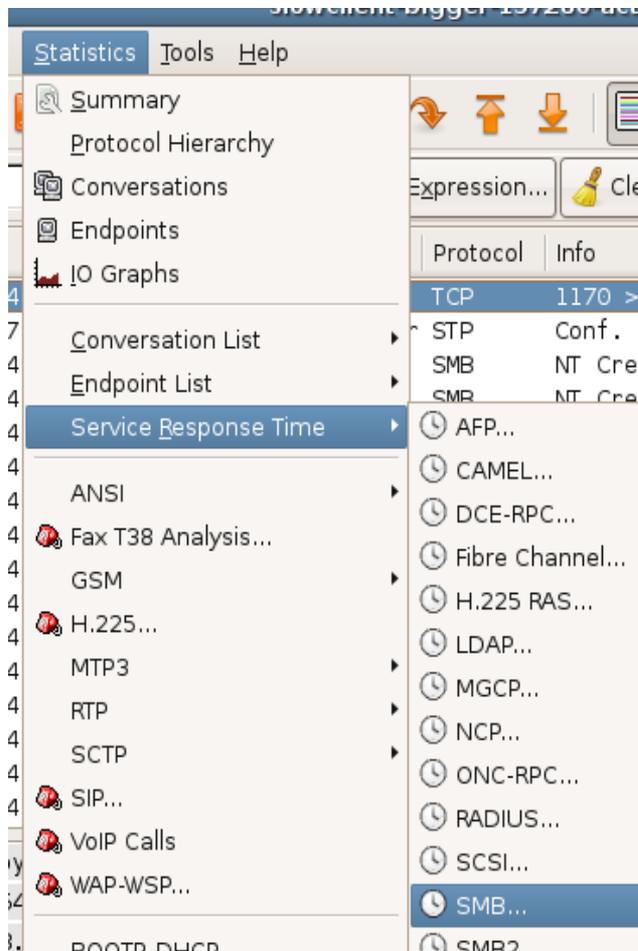
- ❑ Noticed that the field clicked was inside brackets?
- ❑ This means it was an artificial field.
- ❑ This field does not represent any particular bit or byte in the packet that was captured it represents a relation to another packet.
  
- ❑ Wireshark generates a lot of artificial fields. That's what makes Wireshark good!

# 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

The screenshot shows the 'SMB Service Response Time statistics' window. It contains two tables: 'SMB Commands' and 'Transaction2 Sub-Commands'. A red arrow points from the 'Create Stat' button in the previous image to this window.

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

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

- 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

- ❑ 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!

- 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 :-)

## Wireshark to the rescue...

The screenshot shows the Wireshark interface with a filter set to 'smb'. The packet list pane shows several SMB packets. Packet 109334 is selected, and the packet details pane shows the following information:

No.	Time	Source	Destination	Protocol	Info
109198	3.063339	192.168.255.24	192.168.255.34	SMB	Write AndX Response, FID: 0x153e, 65536 bytes
109268	3.063686	192.168.255.24	192.168.255.34	SMB	Write AndX Response, FID: 0x153e, 65536 bytes
109275	3.063755	192.168.255.24	192.168.255.34	SMB	Write AndX Response, 26208 bytes
109277	3.063812	192.168.255.34	192.168.255.24	SMB	Write AndX Request, FID: 0x153e, 65536 bytes at offset 3932000
109334	3.064085	192.168.255.34	192.168.255.24	SMB	Write AndX Request, FID: 0x153e, 65536 bytes at offset 3997536
109378	3.064263	192.168.255.24	192.168.255.34	SMB	Write AndX Response, FID: 0x153e, 65536 bytes

Packet details for the selected packet (109334):

- Word Count (WCT): 14
- AndXCommand: No further commands (0xff)
- Reserved: 00
- AndXOffset: 57054
- FID: 0x153e (\dvs2\file10.dat)
- Offset: 3997536
- Reserved: FFFFFFFF
- Write Mode: 0x0000
- Remaining: 0
- Data Length High (multiply with 64K): 1
- Data Length Low: 0

# Filehandles

File Edit View Go Capture Analyze Statistics Tools Help

Filter: smb + Expression... Clear Apply

No.	Time	Source	Destination	Protocol	Info
109277	3.063812	192.168.255.34	192.168.255.24	SMB	Write AndX Request, FID: 0x153e, 65536 bytes at offset 3932000
109334	3.064085	192.168.255.34	192.168.255.24	SMB	Write AndX Request, FID: 0x153e, 65536 bytes at offset 3997536
109378	3.064263	192.168.255.24	192.168.255.34	SMB	Write AndX Response, FID: 0x153e, 65536 bytes
109433	3.064534	192.168.255.24	192.168.255.34	SMB	Write AndX Response, FID: 0x153e, 65536 bytes
109441	3.064615	192.168.255.24	192.168.255.34	SMB	Write AndX Response, 26208 bytes
109442	3.064671	192.168.255.24	192.168.255.34	SMB	Write AndX Request, FID: 0x153e, 65536 bytes at offset 4000000

Frame 109378 (105 bytes on wire, 105 bytes captured)

- ▶ Ethernet II, Src: Private\_00:18:28 (00:0e:1e:00:18:28), Dst: ChelsioC\_05:54:9e (00:07:43:05:54:9e)
- ▶ Internet Protocol, Src: 192.168.255.24 (192.168.255.24), Dst: 192.168.255.34 (192.168.255.34)
- ▶ Transmission Control Protocol, Src Port: 445 (445), Dst Port: 1170 (1170), Seq: 103947, Ack: 104142493, Len: 51
- ▶ NetBIOS Session Service
- ▼ SMB (Server Message Block Protocol)
  - ▶ SMB Header
  - ▼ Write AndX Response (0x2f)
    - ▶ [FID: 0x153e (\dvs2\file10.dat)]
    - Word Count (WCT): 6
    - AndXCommand: No further commands (0xff)
    - Reserved: 00
    - AndXOffset: 0

We got an artificial field added for the FID.  
But wait, there is more. Its an expansion.

```
▼ SMB (Server Message Block Protocol)
  ▶ SMB Header
  ▼ Write AndX Response (0x2f)
    ▼ [FID: 0x153e (\dvs2\file10.dat)]
      [Opened in: 105140]
      [Closed in: 115661]
      [File Name: \dvs2\file10.dat]
    ▶ Create Flags: 0x00000016
    ▶ Access Mask: 0x00020196
```

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!

```
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

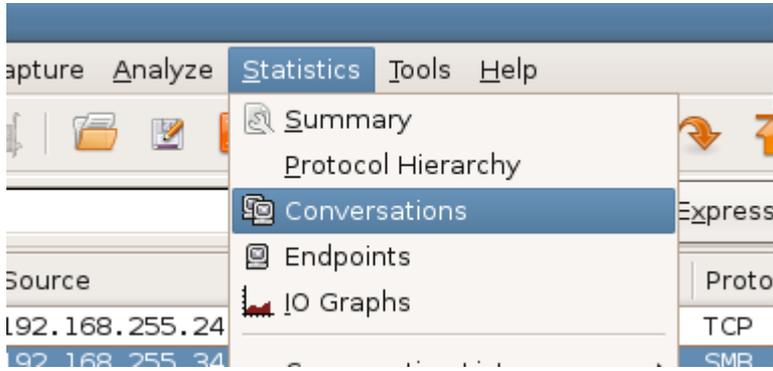
Transaction2 Commands	Calls	Min RTT	Max RTT	Avg RTT
QUERY_FILE_INFO	1	0.00011	0.00011	0.00011

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

# CONVERSATIONS

- ❑ Conversations 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: foo.cap

Ethernet: 4 | Fibre Channel | FDDI | IPv4: 3 | IPX | JXTA | NCP | RSVP | SCTP | **TCP: 2** | Token Ring | UDP: 1 | USB | WLAN

TCP Conversations

Address A	Port A	Address B	Port B	Packets	Bytes	Packets A->B	Bytes A->B	Packets A<-B	Bytes A<-B	Rel Start	Duration	bps A->B	bps A<-B
192.168.255.34	1169	192.168.255.21	139	4	216	2	108	2	108	0.227570000	4.9999	172.80	172.80
192.168.255.34	1170	192.168.255.24	445	210376	211851239	138571	207773757	71805	4077482	0.000000000	1.7144	969567154.68	19027391.52

Name resolution       Limit to display filter

Help   Copy   Close

# CONVERSATIONS

The screenshot shows a software interface titled "Conversations: foo.cap". At the top, there are several protocol filters: Ethernet: 4, Fibre Channel, FDDI, IPv4: 3, IPX, JXTA, NCP, RSVP, SCTP, TCP: 2, Token Ring, UDP: 1, USB, and WLAN. Below this is a section for "TCP Conversations" with a table of data. A context menu is open over the second row of the table, showing options like "Apply as Filter", "Prepare a Filter", "Find Packet", and "Colorize Conversation". The menu is further expanded to show filter types such as "Selected", "Not Selected", and "A <-> B".

Address A	Port A	Address B	Port B	Packets .	Bytes	Packets A->B	Bytes A->B	Packets A<-B	Bytes A<-B	Re
192.168.255.34	1169	192.168.255.21	139	4	216	2	108	2	108	0.
192.168.255.34	1170	192.168.255.24	445						4077482	0.

Name resolution  Limit to display filter

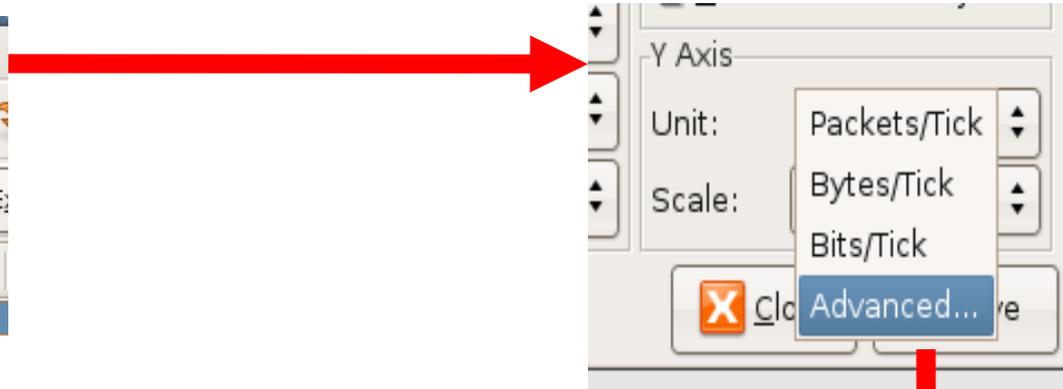
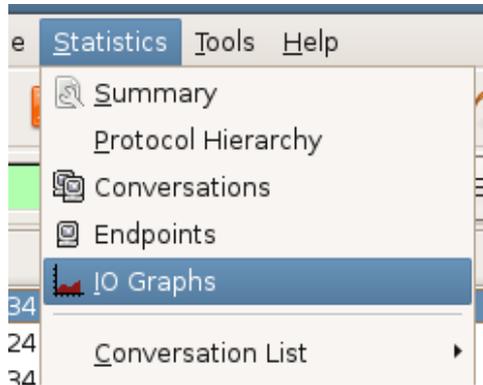
Help Copy

# I/O Graphs (except for LOAD graphs)

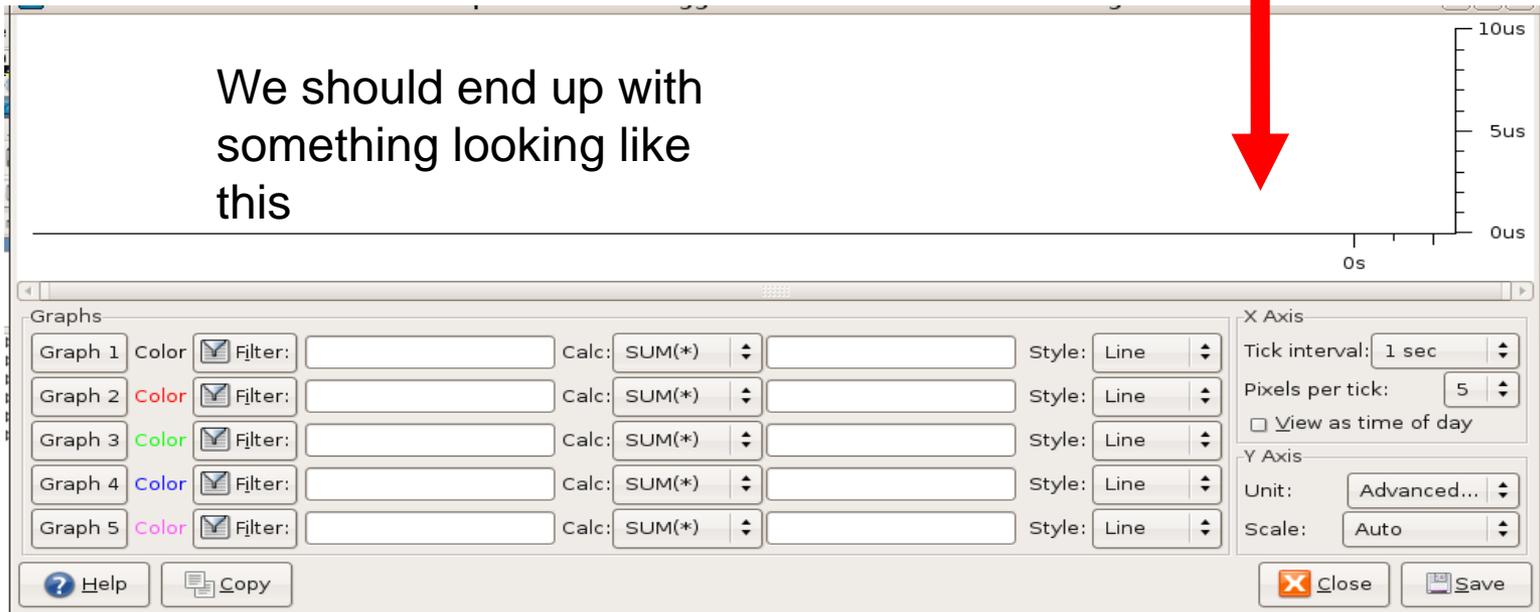
- ❑ I/O Graphs provides a mechanism to calculate many different types of statistics and plot them over time.
- ❑ I/O Graphs works with almost all of the ~70k display filter variables but only makes sense for a smaller subset of them.

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

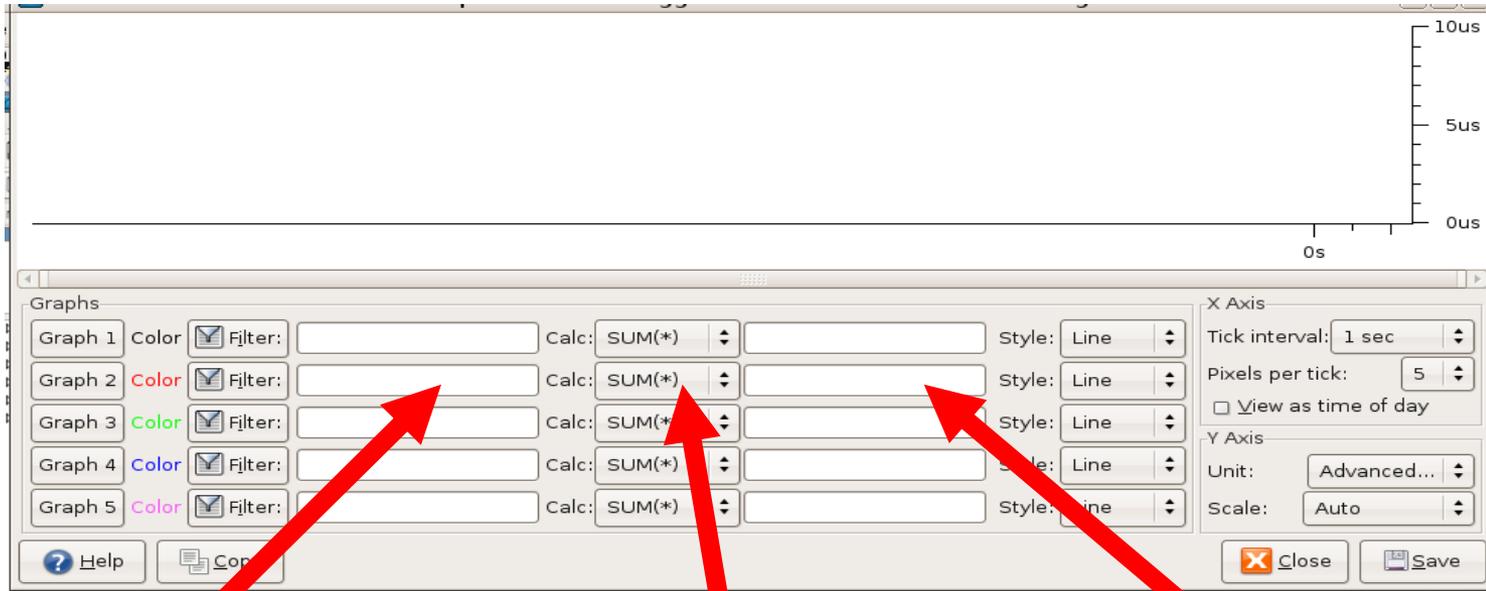
# I/O Graphs



We should end up with something looking like this



# I/O Graphs



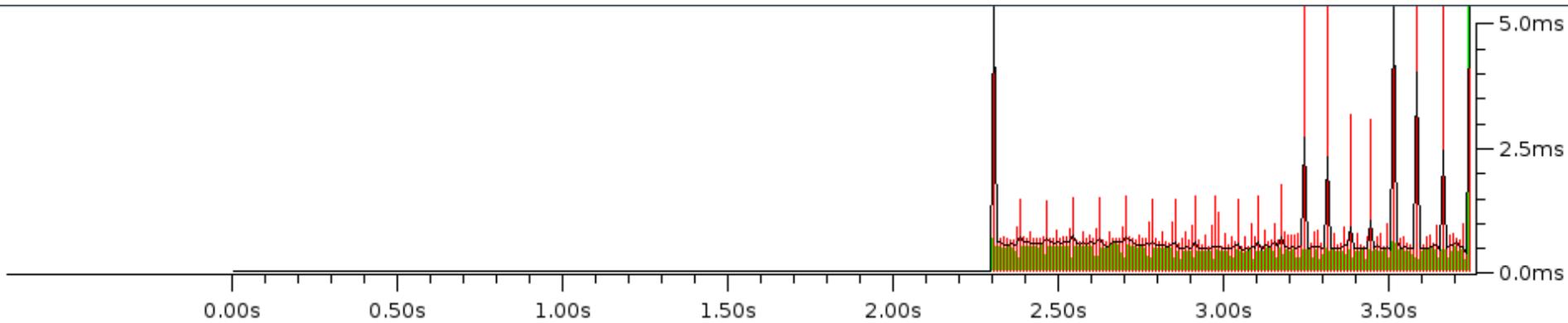
Here goes a full blown display filter that control which subset of the trace we want to calculate over.  
No filter means all packets.

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

Here is the type of operation to graph.

- ❑ 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



Graphs

Graph 1	Color	<input checked="" type="checkbox"/> Filter: smb.cmd==0x2f	Calc: AVG(*)	smb.time	Style: Line
Graph 2	Color	<input checked="" type="checkbox"/> Filter: smb.cmd==0x2f	Calc: MAX(*)	smb.time	Style: Impulse
Graph 3	Color	<input checked="" type="checkbox"/> Filter: smb.cmd==0x2f	Calc: MIN(*)	smb.time	Style: FBar
Graph 4	Color	<input type="checkbox"/> Filter:	Calc: SUM(*)		Style: Line
Graph 5	Color	<input type="checkbox"/> Filter:	Calc: SUM(*)		Style: Line

X Axis

Tick interval: 0.01 sec

Pixels per tick: 2

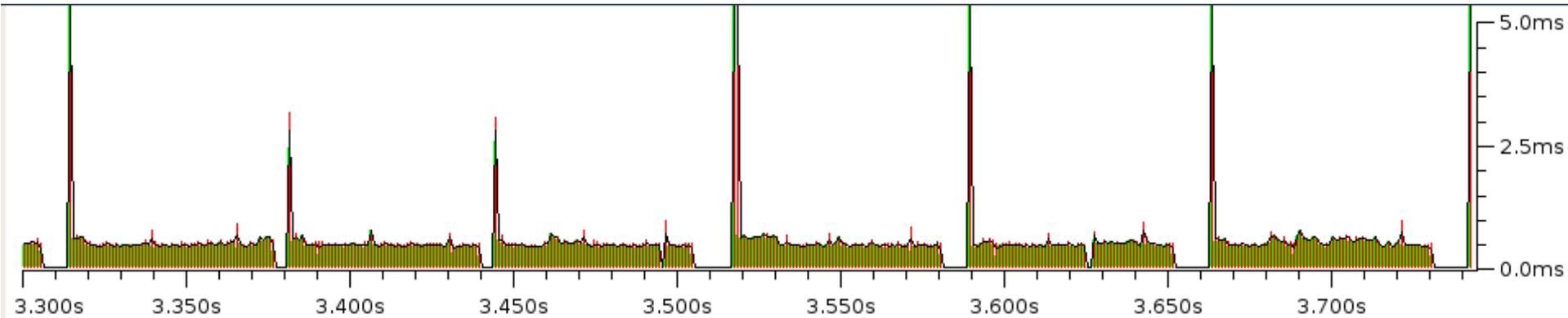
View as time of day

Y Axis

Unit: Advanced...

Scale: 5000

# I/O Graphs



Graphs

Graph	Color	Filter	Calc	Field	Style
Graph 1	Color	<input checked="" type="checkbox"/> Filter: smb.cmd==0x2f	Calc: AVG(*)	smb.time	Style: Line
Graph 2	Color	<input checked="" type="checkbox"/> Filter: smb.cmd==0x2f	Calc: MAX(*)	smb.time	Style: Impulse
Graph 3	Color	<input checked="" type="checkbox"/> Filter: smb.cmd==0x2f	Calc: MIN(*)	smb.time	Style: FBar
Graph 4	Color	<input type="checkbox"/> Filter:	Calc: SUM(*)		Style: Line
Graph 5	Color	<input type="checkbox"/> Filter:	Calc: SUM(*)		Style: Line

X Axis

Tick interval: 0.001 sec

Pixels per tick: 2

View as time of day

Y Axis

Unit: Advanced...

Scale: 5000

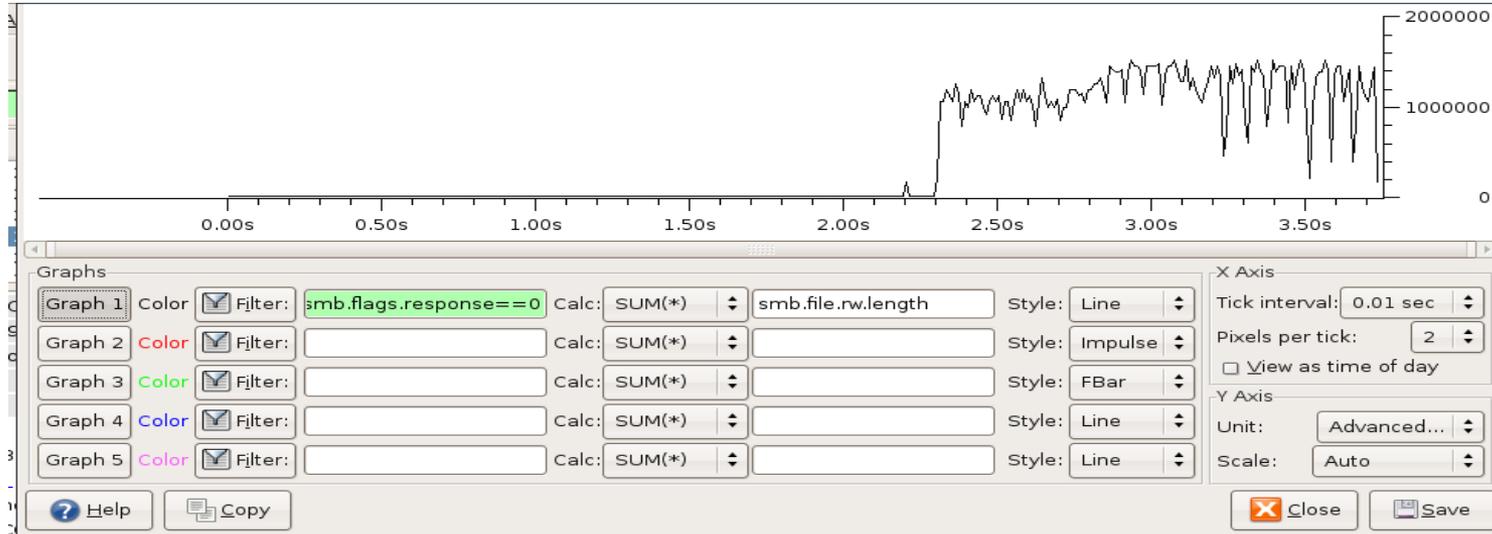
- `-z "io,stat,0.01,MIN(smb.time)smb.time,MAX(smb.time)smb.time,AVG(smb.time)smb.time"`

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
...

```

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.

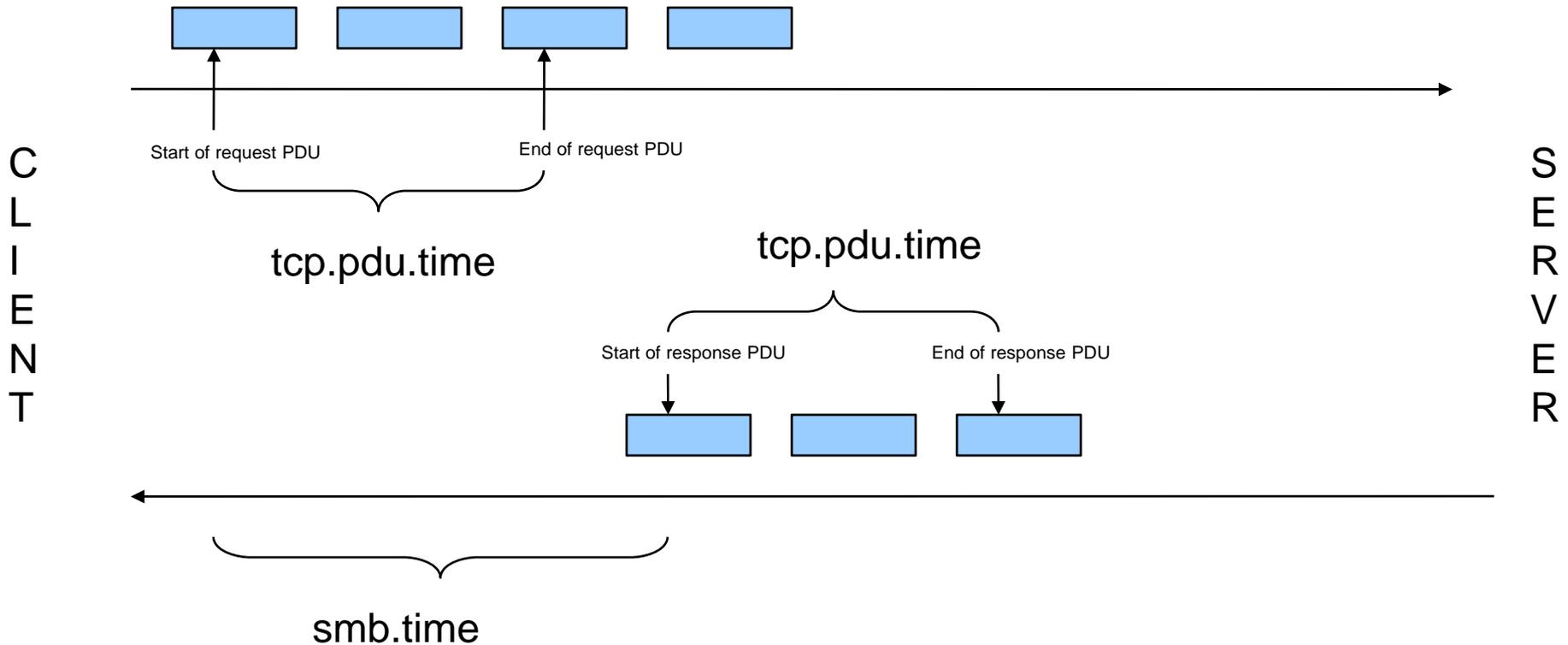
Something is seriously wrong here!

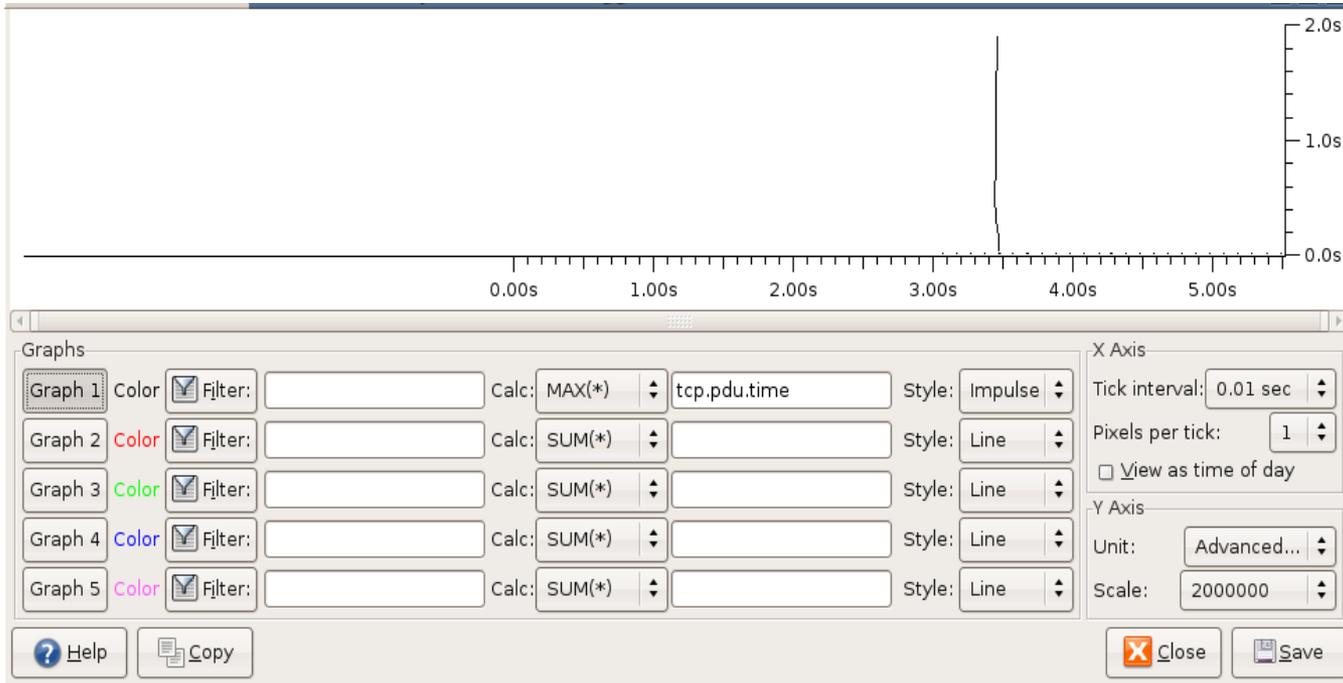
# TCP

- 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.

- ❑ 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





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

File Edit View Go Capture Analyze Statistics Tools Help

Filter:  + Expression... Clear Apply

No. .	Time	Source	Destination	Protocol	Info
3573	4.195415	192.168.255.24	192.168.255.34	SMB	Write AndX Response, 38928 bytes
3574	4.195484	192.168.255.34	192.168.255.24	SMB	Write AndX Request, FID: 0x1551, 65536 bytes
3575	4.195495	192.168.255.34	192.168.255.24	TCP	[Continuation to #3574] 1170 > 445 [ACK] Seq
3576	4.195501	192.168.255.34	192.168.255.24	TCP	[Continuation to #3574] 1170 > 445 [ACK] Seq
3577	4.195507	192.168.255.34	192.168.255.24	TCP	[Continuation to #3574] 1170 > 445 [ACK] Seq
3578	4.195513	192.168.255.34	192.168.255.24	TCP	[Continuation to #3574] 1170 > 445 [ACK] Seq

Transmission Control Protocol, Src Port: 1170 (1170), Dst Port: 445 (445), Seq: 3404277, Ack: 3272, Len: 1460

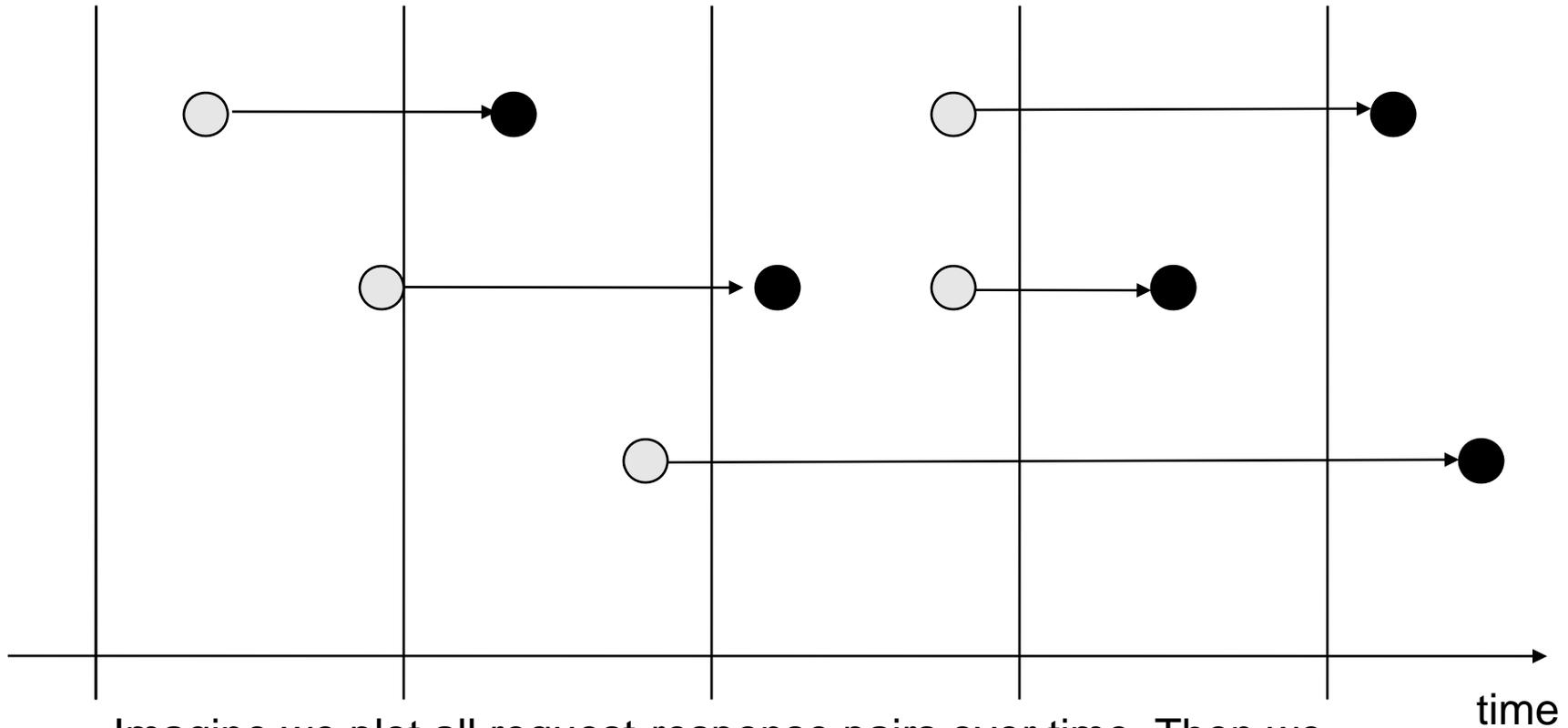
- Source port: 1170 (1170)
- Destination port: 445 (445)
- Sequence number: 3404277 (relative sequence number)
- [Next sequence number: 3405737 (relative sequence number)]
- Acknowledgement number: 3272 (relative ack number)
- Header length: 20 bytes
- Flags: 0x10 (ACK)
- Window size: 32589
- Checksum: 0x855b [unchecked, not all data available]
- [SEQ/ACK analysis]
  - [This is an ACK to the segment in frame: 3573]
  - [The RTT to ACK the segment was: 0.000069000 seconds]
  - [Number of bytes in flight: 1460]
- [Timestamps]
  - [Last frame of this PDU: 3633]
  - [Time until the last segment of this PDU: 0.000408000 seconds]

NetBIOS Session Service

# I/O LOAD Graphs

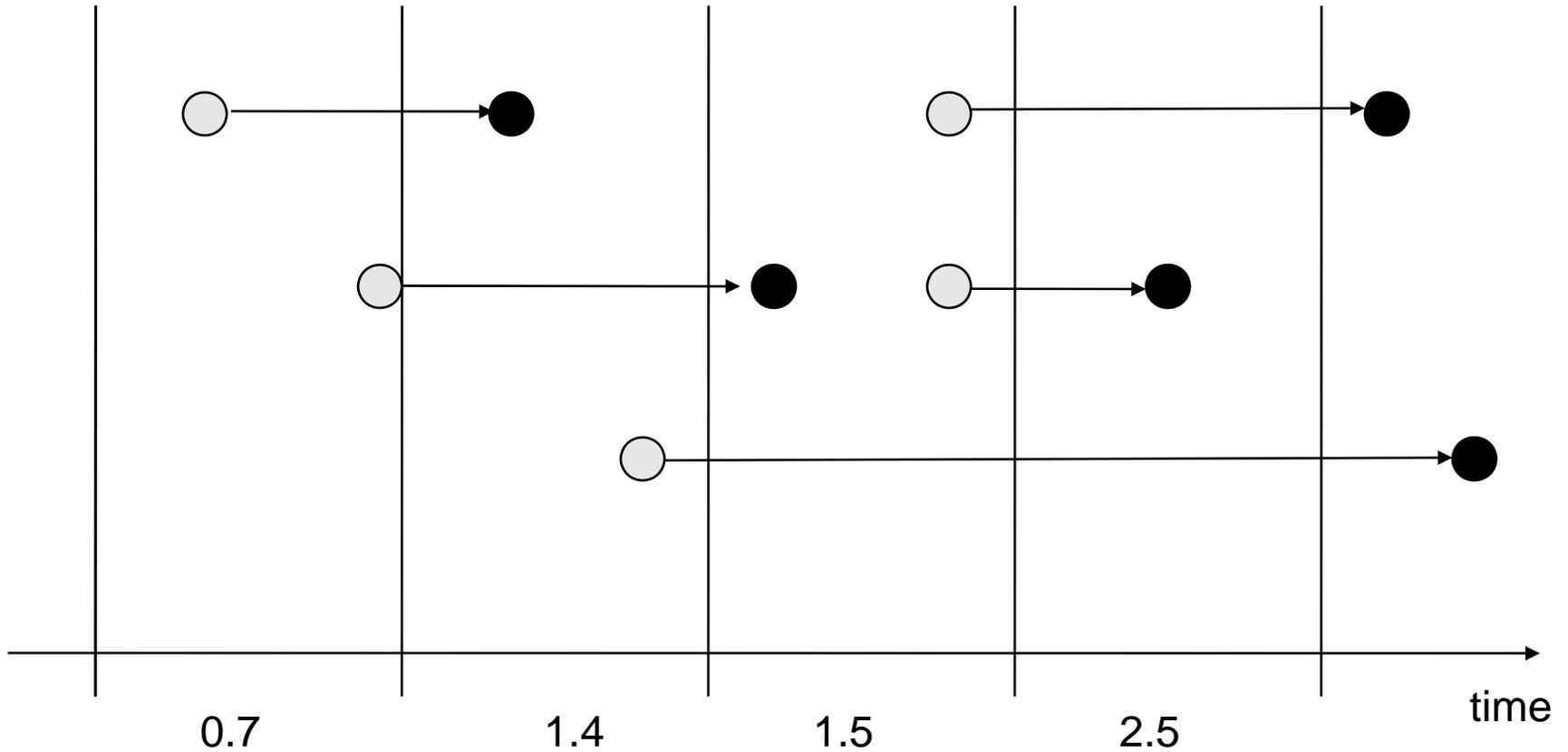
- ❑ LOAD graphs are a special kind of I/O graphs.
- ❑ These graphs operate only on fields that are response time fields such as smb.time, rpc.time, scsi.time etc.
- ❑ What they then plot is the QUEUE-depth of the connection over time.

# LOAD graphs



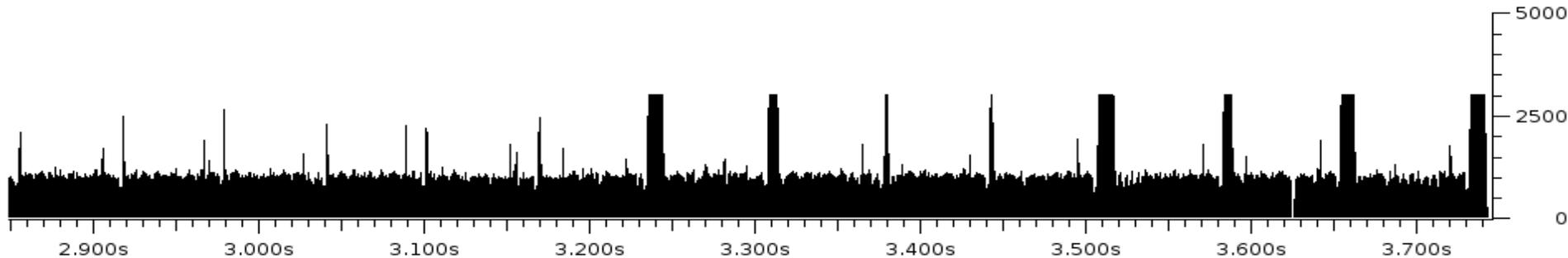
Imagine we plot all request-response pairs over time. Then we “estimate” what the average amount of coverage is per “interval”. This then becomes out “average” queue-depth for that interval.

# LOAD graphs



- ❑ 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



Graphs

Graph 1	Color	<input type="checkbox"/> Filter:	Calc: LOAD(*)	smb.time	Style: Impulse
Graph 2	Color	<input type="checkbox"/> Filter:	Calc: SUM(*)		Style: Impulse
Graph 3	Color	<input type="checkbox"/> Filter:	Calc: SUM(*)		Style: FBar
Graph 4	Color	<input type="checkbox"/> Filter:	Calc: SUM(*)		Style: Line
Graph 5	Color	<input type="checkbox"/> Filter:	Calc: SUM(*)		Style: Line

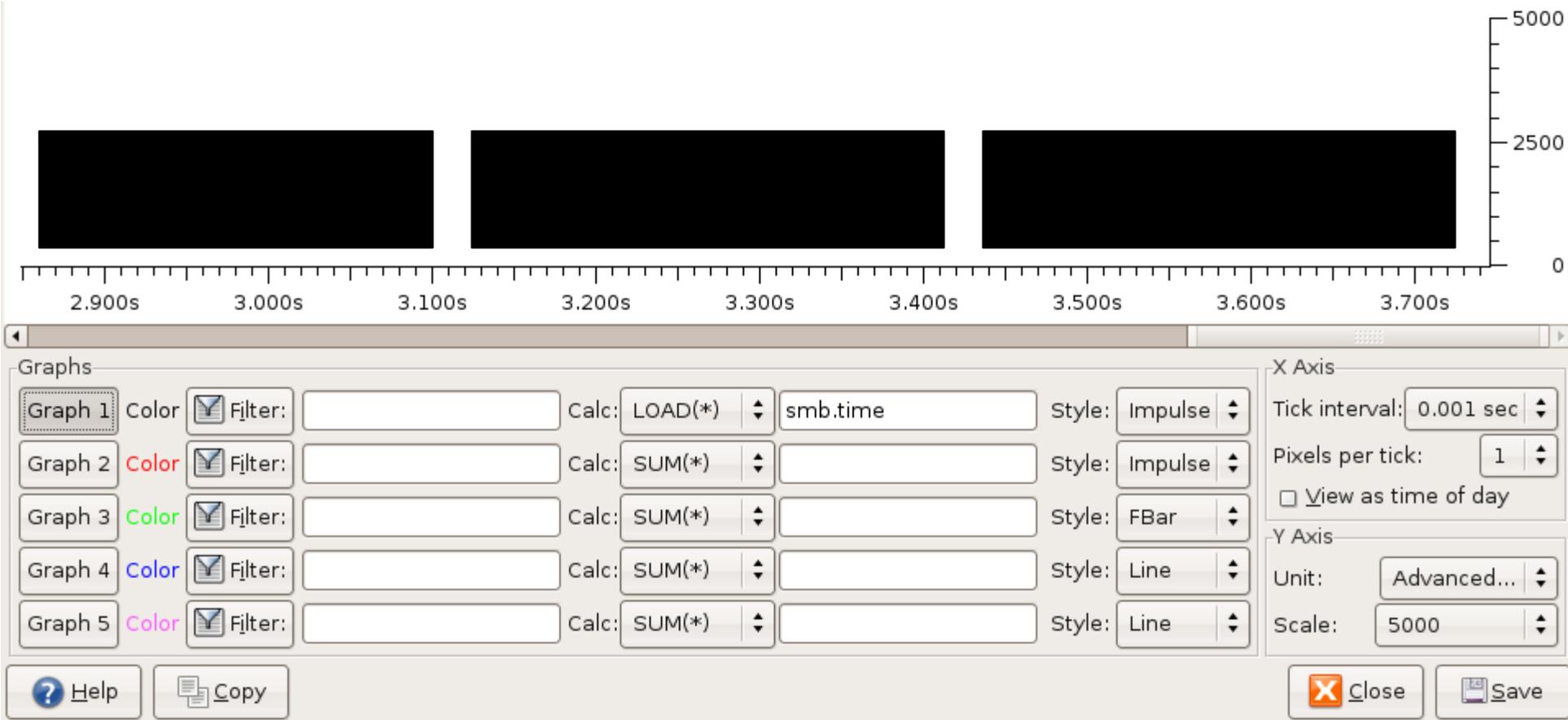
X Axis  
Tick interval: 0.001 sec  
Pixels per tick: 1  
 View as time of day

Y Axis  
Unit: Advanced...  
Scale: 5000

Help Copy **Note that 1 I/O represents 1000 on the Y-axis** Close Save

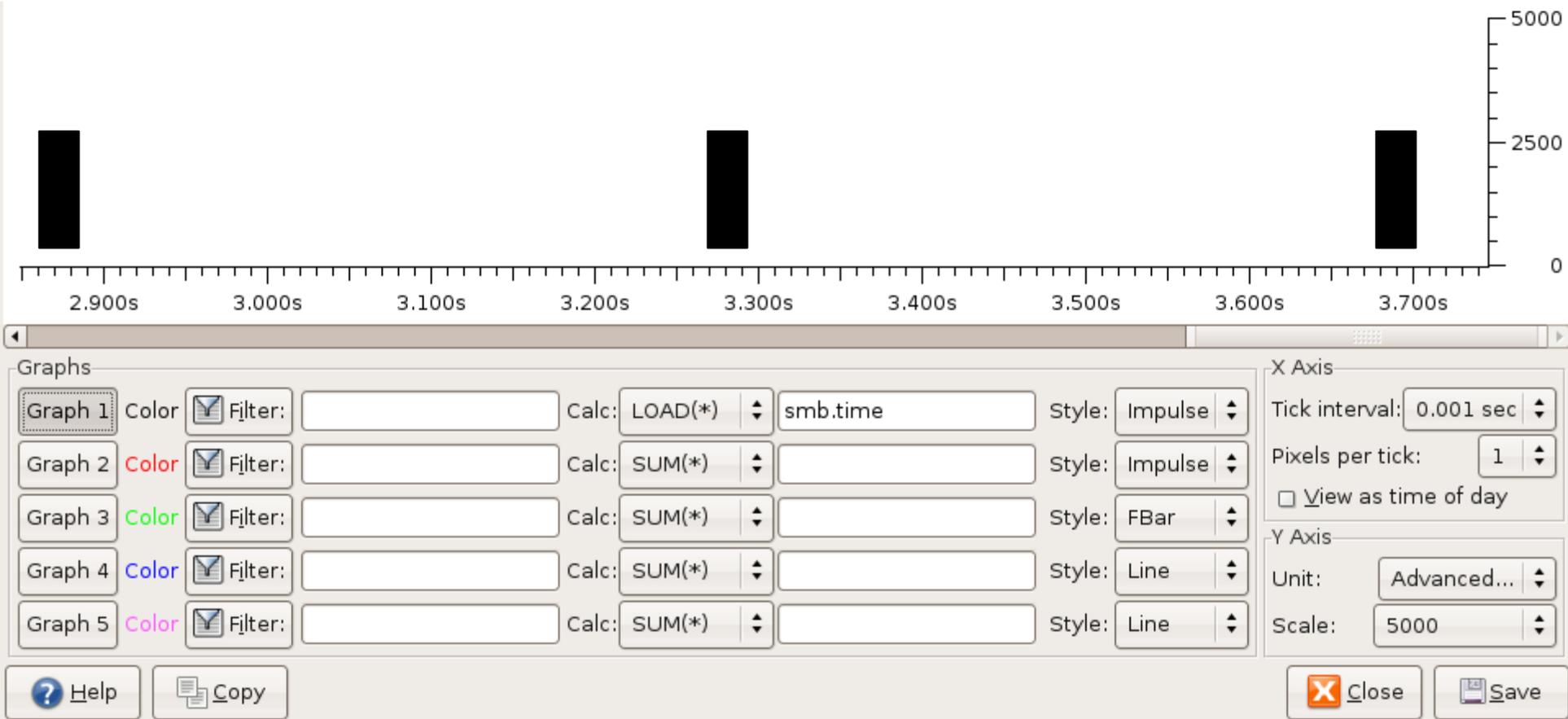
We see the client uses 3 threads. The client can on average barely keep 1 I/O in flight. Every ~80ms the server pauses for a few ms.

# 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.

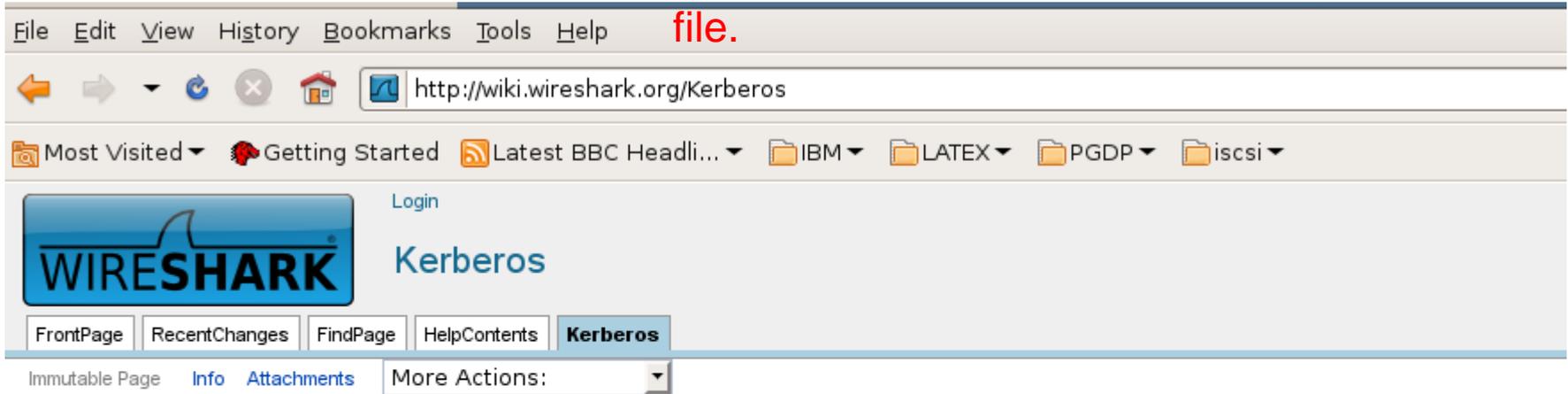
- QUEUE DEPTH analysis can also be done by tshark.

# 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.

- ❑ 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.

## Instructions on how to create a keytab file.



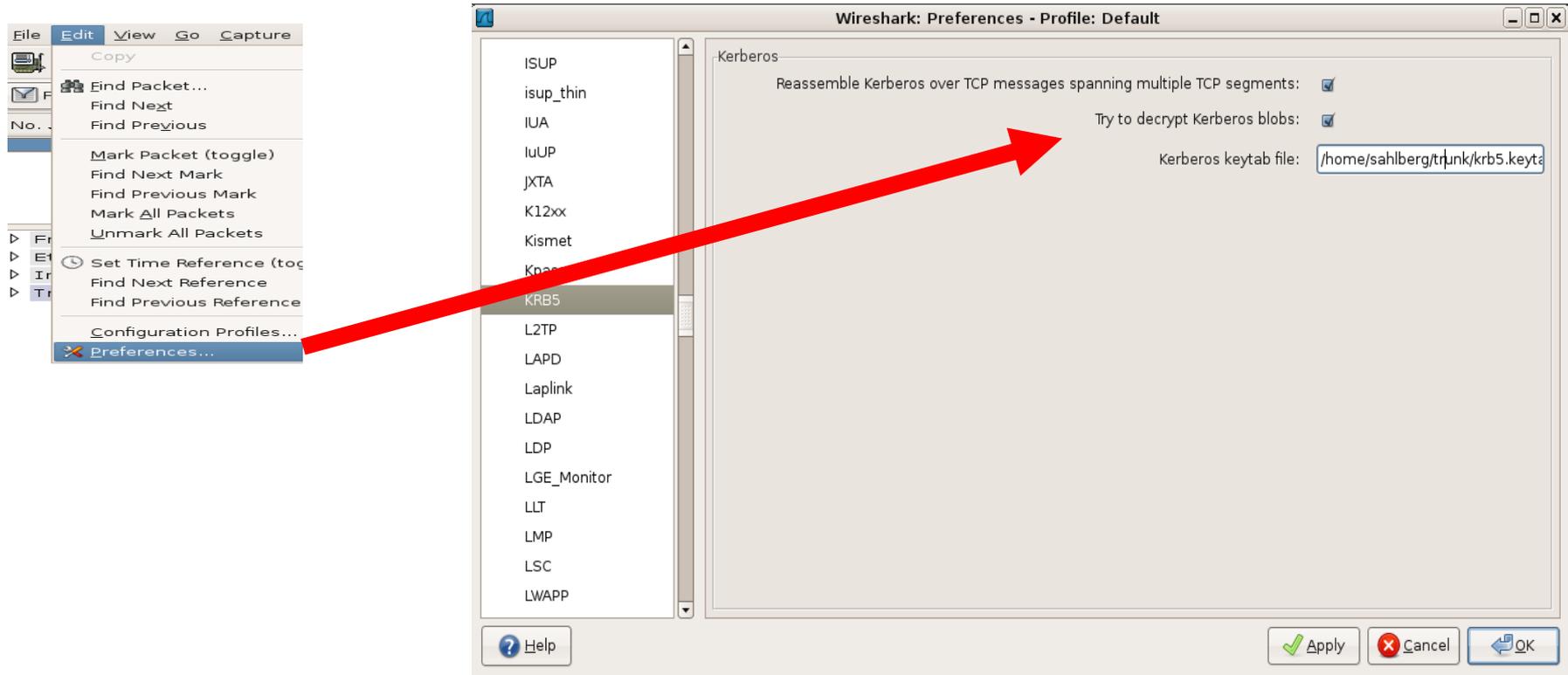
## Kerberos

Kerberos is a service that provides mutual authentication between users and services in a network. It is popular both in Unix and Windows.

## History

Initially Kerberos was developed and deployed as part of the Athena project. This version of the Kerberos service and protocol was eventually replaced by Kerberos v5 in all other environments.

# Kerberos



And then Apply and OK and restart wireshark.



# QnA (if time permits :-)