

SHARKFEST '12

Wireshark Developer and User Conference

Rolf Leutert

Network Expert & Trainer | Leutert NetServices | Switzerland

Tuning Win7 Using Wireshark's TCP Stream Graph

Case Study

- Customer is **distributing Software** over night to remote office in Asia
- But the process **does not finish** before local business hours starts
- Customer is paying for a WAN bandwidth of **45 Mbps**
- He calculates an available throughput of only around **2 Mbps**

- Does the bandwidth **provider** limit the rate?
- Is the **server** or the **client** not performing?

- **Analyze the performance of a TCP session using TCP Stream graph**

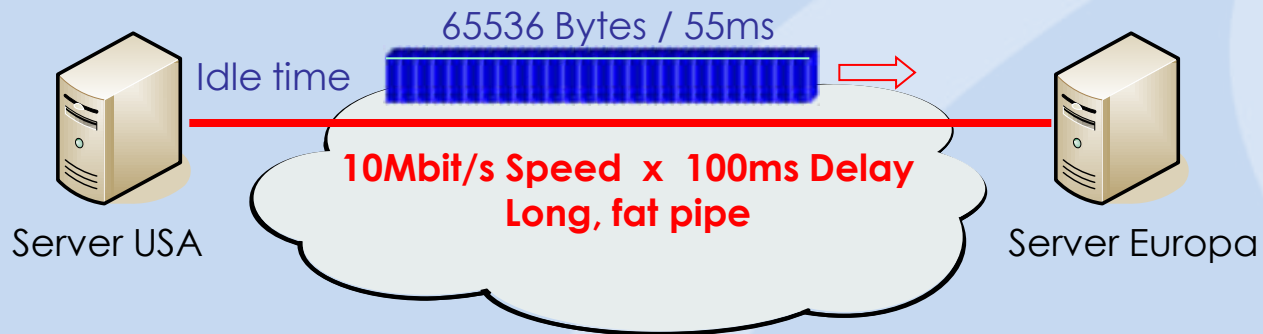
TCP Extension for High performance

- TCP was designed to operate in the range **100bps to 10Mbps** and delays of **1ms to 100sec**.
- The introduction of **fiber optics** is resulting in ever higher transmission speeds paths and are moving out of the domain for which TCP was originally engineered.
- TCP performance depends not upon the transfer rate itself, but rather upon the product of the transfer rate and the round-trip delay. If the **bandwidth x delay product** is large, TCP throughput will be limited.
- Internet path operating in this region are called **"long, fat pipe"**, and a network containing this path as an "LFN" (pronounced "elephan(t)").



‘Long - Fat - Pipe’ Problems

- Maximum standard TCP window size is 65536 Bytes ($=2^{16}$)



„Long - Fat - Pipe‘ Problems

- High-capacity packet **satellite channels** are LFN's. Delay $4 \times 35'800 \text{ km} = 470\text{ms}$ Round Trip Time
- **Terrestrial** fiber-optical paths will also fall into the LFN class
- There are three fundamental performance problems with the current TCP over LFN paths:
 - Window Size Limit (max 65k bytes) → Remedy: **TCP option „Window scale“**
 - Recovery from Segment Losses → Remedy: **TCP option „Selective acknowledges“**
 - Round-Trip Measurement → Remedy: **TCP option „Time stamp“**



TCP ,Window Scaling' Option

- TCP Window Size of 65'535 Bytes is **too small**.
- A multiplier **Scaling Factor** resolves this limitation.
- Scaling Factor **S is negotiated** at TCP setup.
- Each end can offer an **individual** Scaling Factor.
- The value for the Scaling Factors can vary from **0 to 14**.
- Calculation for the scaled Window Size is as follows:

$$\text{Scaled Window Size} = \text{Window Bytes} \times 2^S$$

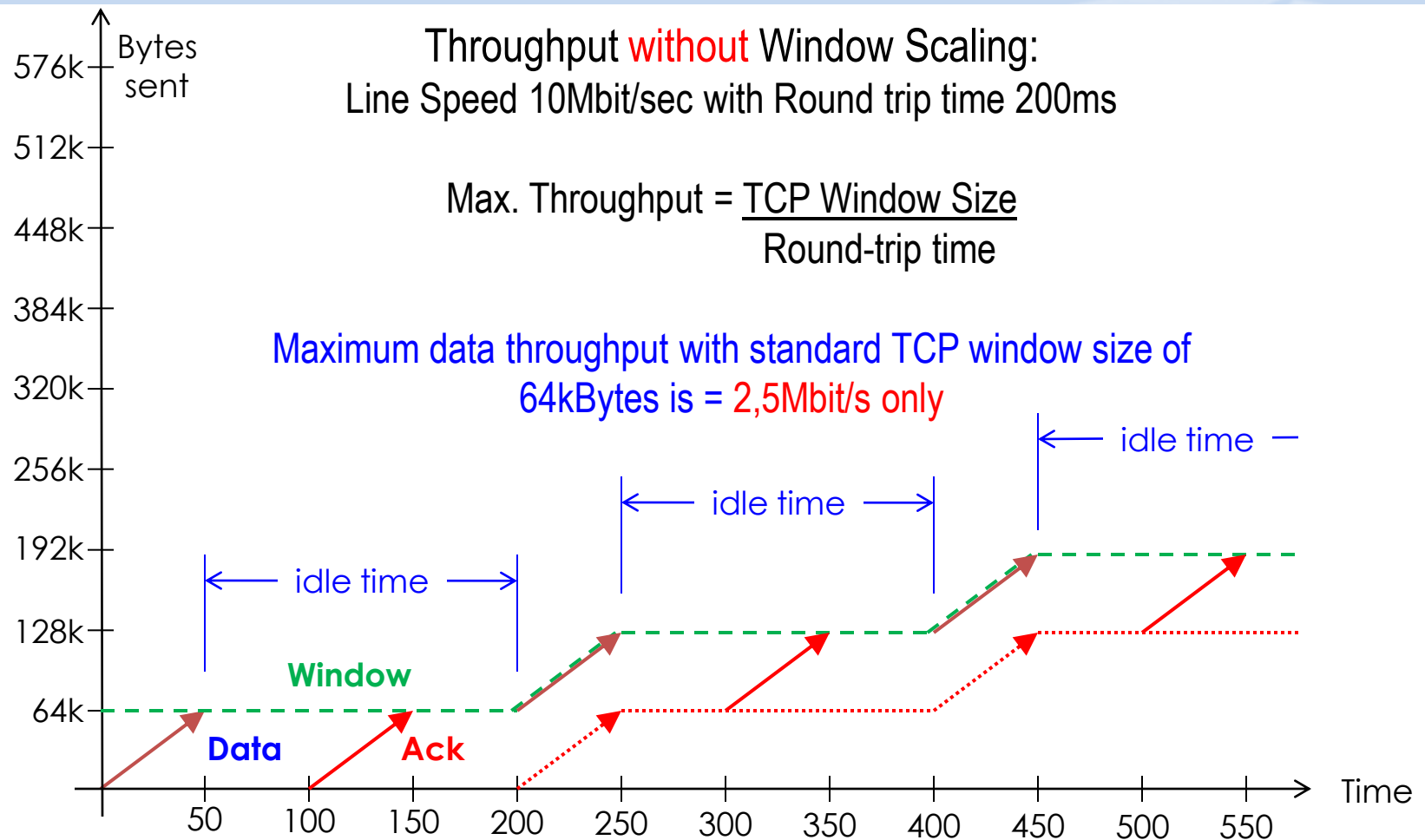
- Example: Window Size 46 Bytes, Scaling Factor $S=7 \rightarrow 2^7 = 128$

$$46 \text{ Bytes} \times 128 = 5'888 \text{ Bytes}$$

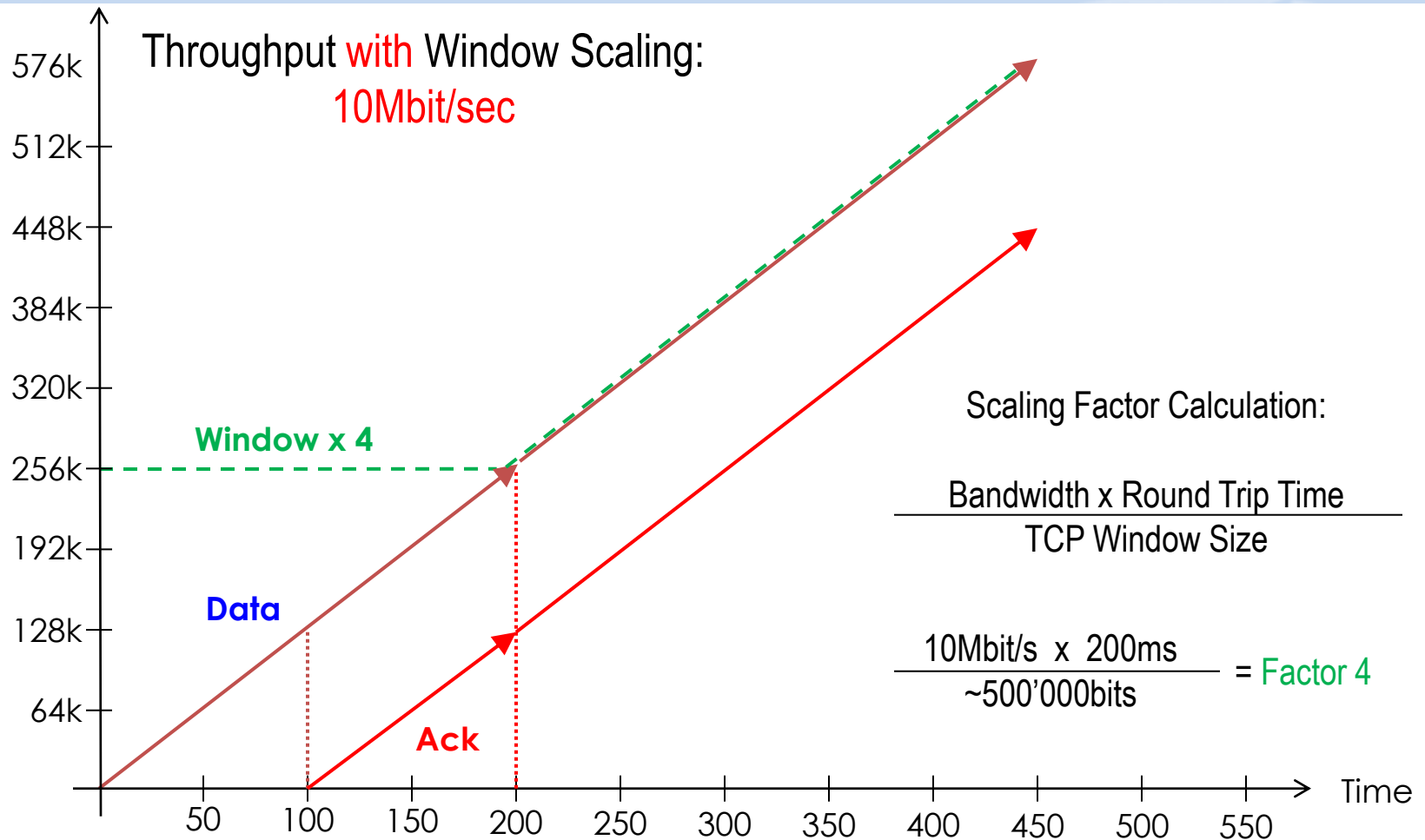
- The maximum Window Size can be 1'073'741'824 Bytes = **1 Gigabyte**



TCP 'Window Scaling' Option



TCP 'Window Scaling' Option



TCP 'Window Scaling' Option

The image shows a Wireshark packet capture of a TCP connection. The main table displays several packets. Packet 3367 is a SYN from the client (192.168.0.203) to the server (69.4.231.52) with a window size of 8192 and a scaling factor of 4. Packet 3368 is a SYN-ACK from the server with a window size of 5840 and a scaling factor of 512. Packet 3369 is an ACK from the client with a window size of 66640. Packet 3370 is an HTTP GET request. Packet 3371 is an ACK from the client with a window size of 7680. Packet 3372 is a reassembly PDU. Packet 3373 is a reassembly PDU. Packet 3374 is an ACK from the client with a window size of 66640. Packet 3375 is a reassembly PDU.

Callouts from the image:

- ,Window Scaling' factor from Client** (points to the WS=4 in packet 3367)
- ,Window Size' unscaled** (points to the win=8192 in packet 3367)
- ,Window Size' scaled** (points to the win=66640 in packet 3369)
- ,Window Scaling' factor from Server** (points to the WS=512 in packet 3368)

Detailed view of packet 3370 (HTTP GET):

```
Frame 3370: 878 bytes on wire (7024 bits) captured on interface eth0 (00:11:95:b7:e0:3e)
Ethernet II, Src: Intel E1000 (08:00:27:00:00:03), D-Link: 69.4.231.52 (69.4.231.52)
Internet Protocol Version 4, Src: 192.168.0.203, Dst: 69.4.231.52 (80), Seq: 1, Ack: 1
Transmission Control Protocol, Src Port: 54889 (54889), Dst Port: http (80), Seq: 1, Ack: 1
[Stream index: 99]
Sequence number: 1 (relative sequence number)
[Next sequence number: 825 (relative sequence number)]
Acknowledgement number: 1 (relative sequence number)
Header length: 20 bytes
Flags: 0x18 (PSH, ACK)
Window size value: 16660
[Calculated window size: 66640]
[window size scaling factor: 4]
Checksum: 0xf0fe [validation disabled]
```

After the two TCP SYN frames, the window size is announced in the scaled format and Wireshark displays the scaled value.

TCP Extensions for High Performance

- The following TCP options are defined in RFC1323:
 - 01 No operation (for padding)
 - 02 Max. Window size (SYN)
 - 03 **Window scale** (SYN)
 - 04 **SACK permitted** (SYN)
 - 05 **SACK option** (Acknowledges)
 - 08 **Time stamp** (SYN and Acknowledges)

```
options: (24 bytes)
  Maximum segment size: 1460 bytes
  NOP
  Window scale: 2 (multiply by 4)
  NOP
  NOP
  Timestamps: Tsval 0, Tsecr 0
  NOP
  NOP
  SACK permitted
0000  00 90 27 96 a9 2e 00 00 e8 20 20 58 08 00 45 00  ..'.....X..E.
0010  00 40 00 79 40 00 80 06 77 fe c0 a8 00 69 c0 a8  .@.y@...w....i..
0020  00 87 04 10 00 8b aa 5a 20 00 00 00 00 00 00 b0 02  .....Z.....
0030  eb c0 f9 07 00 00 02 04 05 b4 01 03 03 02 01 01  .....[.....
0040  08 0a 00 00 00 00 00 00 00 00 01 01 04 02  .....

```

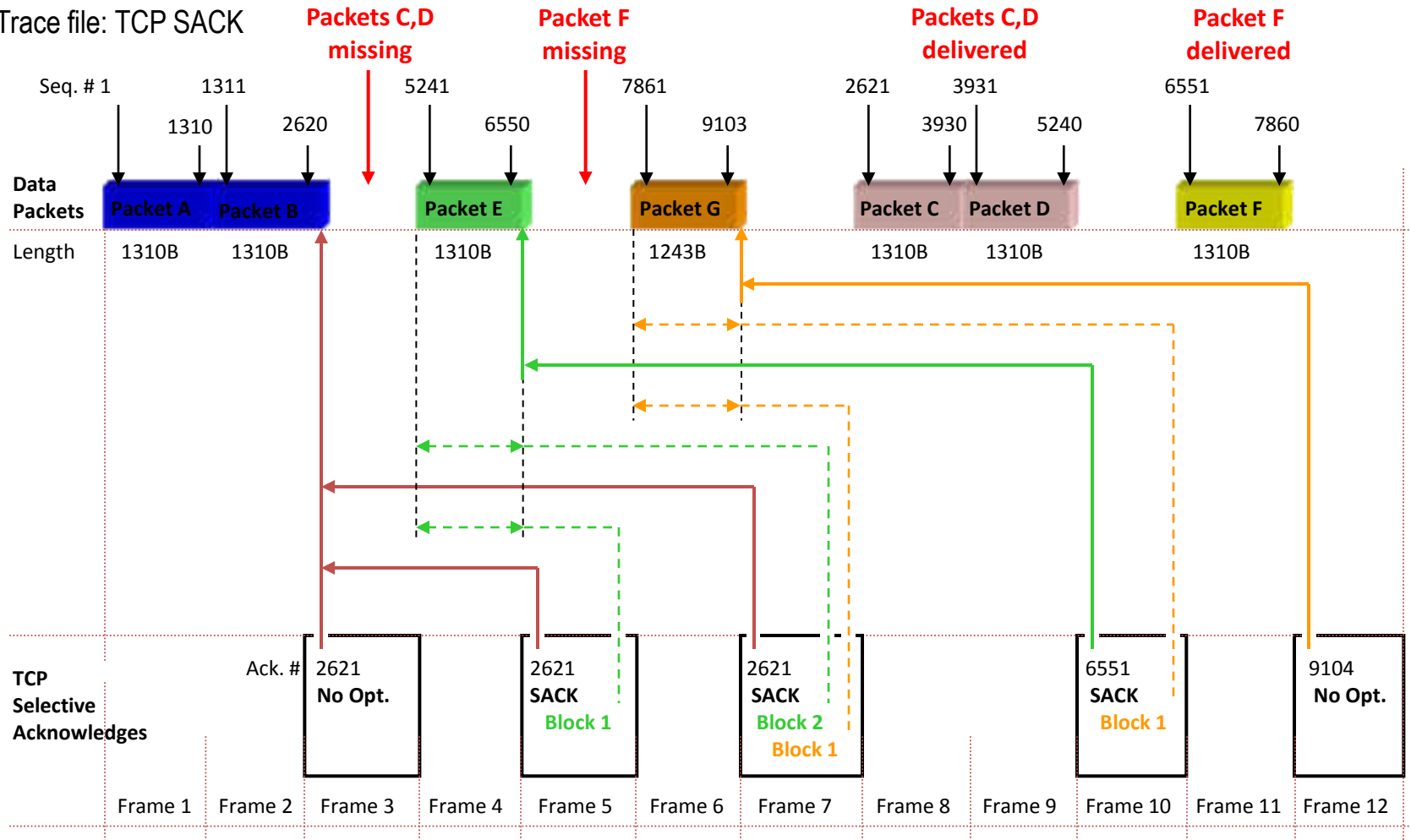
TCP 'Selective Acknowledge' Option

- The usage of the **TCP SACK option** is negotiated during the 3-Way hand shake.
- The SACK option can be activated from **one or both sides**.
- Without SACK option, only the **last** received segment of a contiguous series can be acknowledged.
- The SACK Option allows to **acknowledge non-contiguous** segments of a series and can request for specific segments.
- The SACK Option can **improve** the throughput of LFN's significantly.



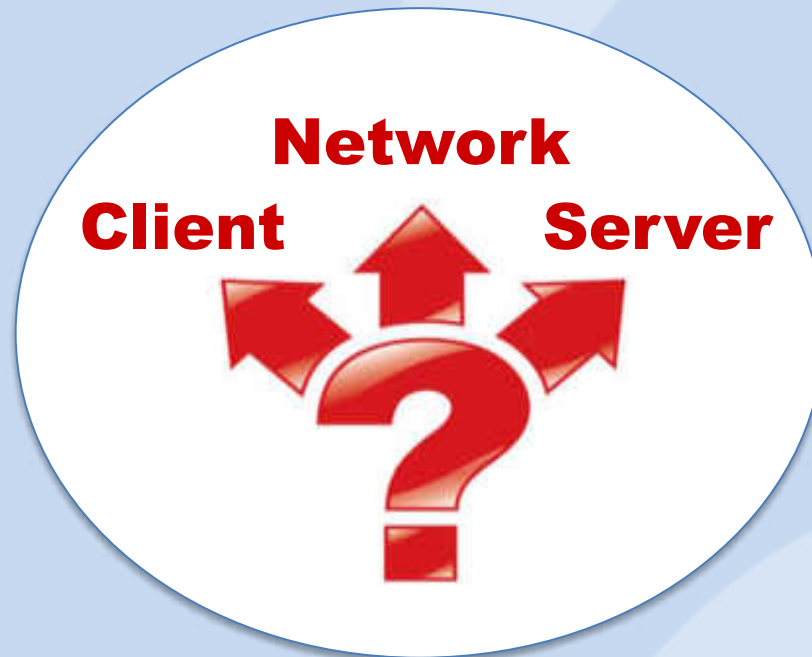
TCP 'Selective Acknowledge' Option

Trace file: TCP SACK



TCP Analysis with Wireshark Expert

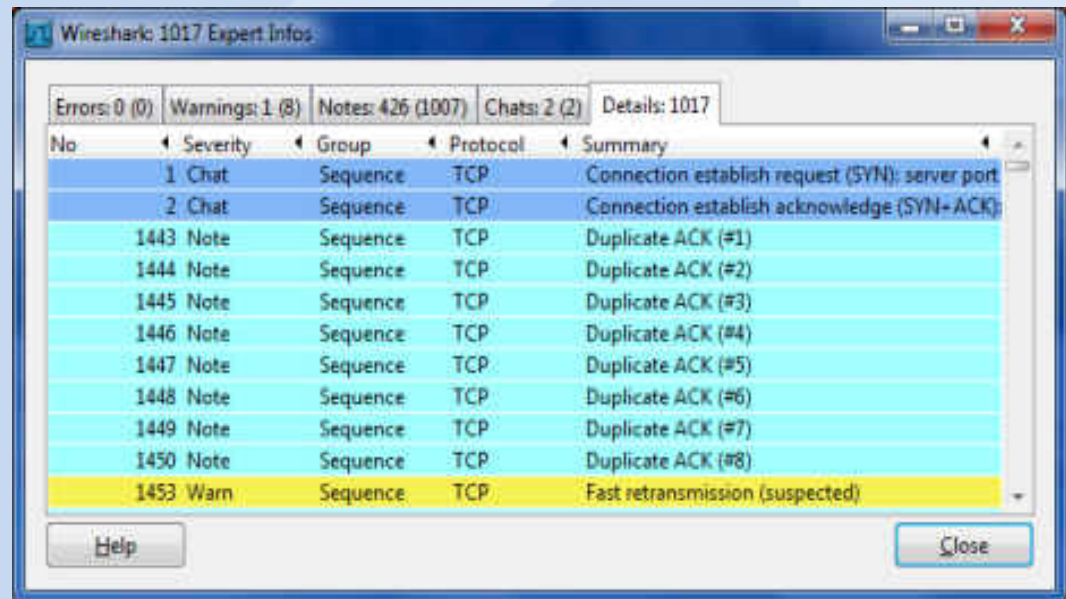
- TCP performance can be influenced by these **three main components**
- The **Wireshark Expert** is offering great support in analyzing TCP sessions
- Understanding **TCP and Expert Messages** helps isolating problems



TCP Analysis with Wireshark Expert

- The **Wireshark Expert System** recognizes many abnormalities or errors and creates a list sorted by severities:

- Segment Lost
- Duplicate ACK
- Retransmissions
- Fast Retransmissions
- Zero Window
- Window Full
- and many more...



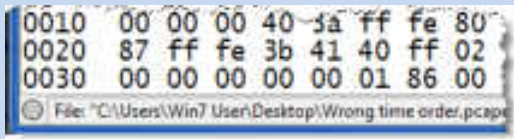
The screenshot shows the 'Wireshark: 1017 Expert Infos' window. It displays a summary of expert messages: Errors: 0 (0), Warnings: 1 (8), Notes: 426 (1007), Chats: 2 (2), and Details: 1017. Below this is a table with columns for No., Severity, Group, Protocol, and Summary. The table lists several events, with the last one highlighted in yellow.

No.	Severity	Group	Protocol	Summary
1	Chat	Sequence	TCP	Connection establish request (SYN): server port
2	Chat	Sequence	TCP	Connection establish acknowledge (SYN+ACK)
1443	Note	Sequence	TCP	Duplicate ACK (#1)
1444	Note	Sequence	TCP	Duplicate ACK (#2)
1445	Note	Sequence	TCP	Duplicate ACK (#3)
1446	Note	Sequence	TCP	Duplicate ACK (#4)
1447	Note	Sequence	TCP	Duplicate ACK (#5)
1448	Note	Sequence	TCP	Duplicate ACK (#6)
1449	Note	Sequence	TCP	Duplicate ACK (#7)
1450	Note	Sequence	TCP	Duplicate ACK (#8)
1453	Warn	Sequence	TCP	Fast retransmission (suspected)

- You still need well-founded TCP knowledge to understand the error messages and to draw the right conclusions.

TCP Analysis with Wireshark Expert

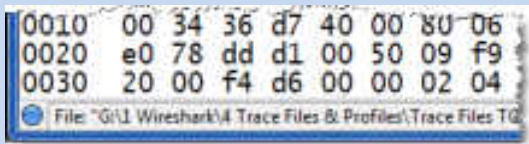
- Click on the colored **Expert Button** to open the **Expert Infos** window



```
0010 00 00 00 40 ja ff fe 80
0020 87 ff fe 3b 41 40 ff 02
0030 00 00 00 00 00 01 86 00
```

File: "C:\Users\Win7 User\Desktop\Wrong time order.pcap"

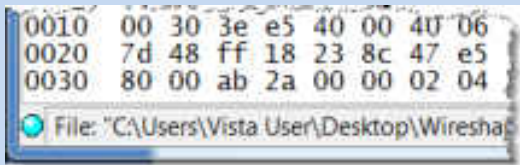
Level 0 = No Expert info available for protocols present in trace file (i.e. for protocols using UDP)



```
0010 00 34 36 d7 40 00 80 06
0020 e0 78 dd d1 00 50 09 f9
0030 20 00 f4 d6 00 00 02 04
```

File: "G:\1 Wireshark\4 Trace Files & Profiles\Trace Files TG"

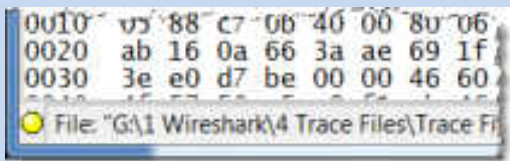
Level 1 = Chats: Information about normal data flow, e.g. TCP session establishment and closing. HTTP Get/OK/404 etc.



```
0010 00 30 3e e5 40 00 40 06
0020 7d 48 ff 18 23 8c 47 e5
0030 80 00 ab 2a 00 00 02 04
```

File: "C:\Users\Vista User\Desktop\Wiresha"

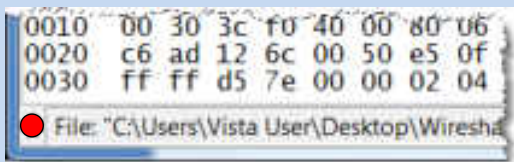
Level 2 = Notes: Reference to slight abnormalities like **Duplicate ACK**, **Retransmissions** etc.



```
0010 05 88 c7 0b 40 00 80 06
0020 ab 16 0a 66 3a ae 69 1f
0030 3e e0 d7 be 00 00 46 60
```

File: "G:\1 Wireshark\4 Trace Files\Trace Fi"

Level 3 = Warnings: Informs about abnormalities like **Segment lost**, **Segments out of order** etc.



```
0010 00 30 3c f0 40 00 80 06
0020 c6 ad 12 6c 00 50 e5 0f
0030 ff ff d5 7e 00 00 02 04
```

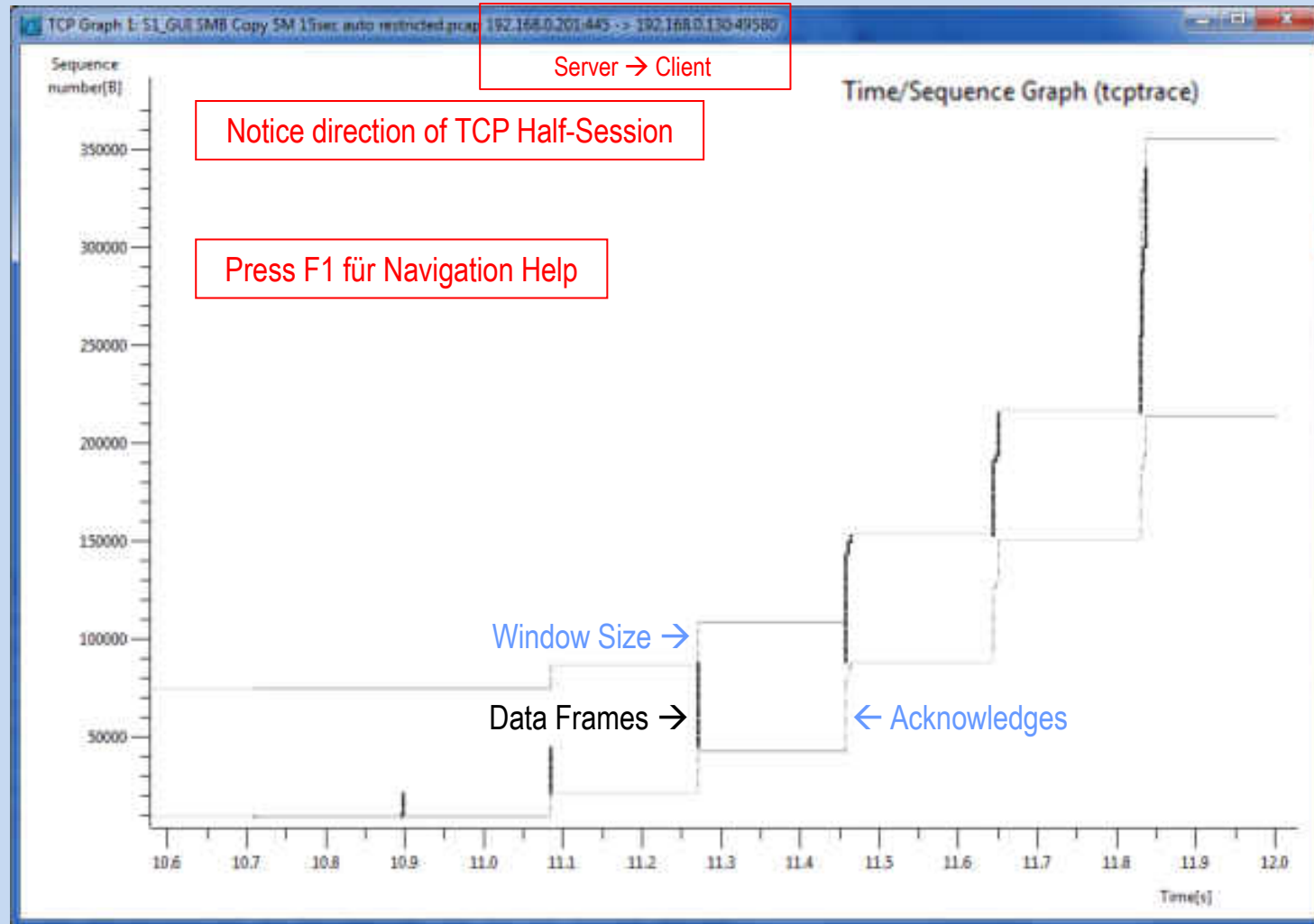
File: "C:\Users\Vista User\Desktop\Wiresha"

Level 4 = Errors: Messages on serious problems like **deformed segments** (i.e. missing fields)

TCP Analysis with TCP Stream Graph

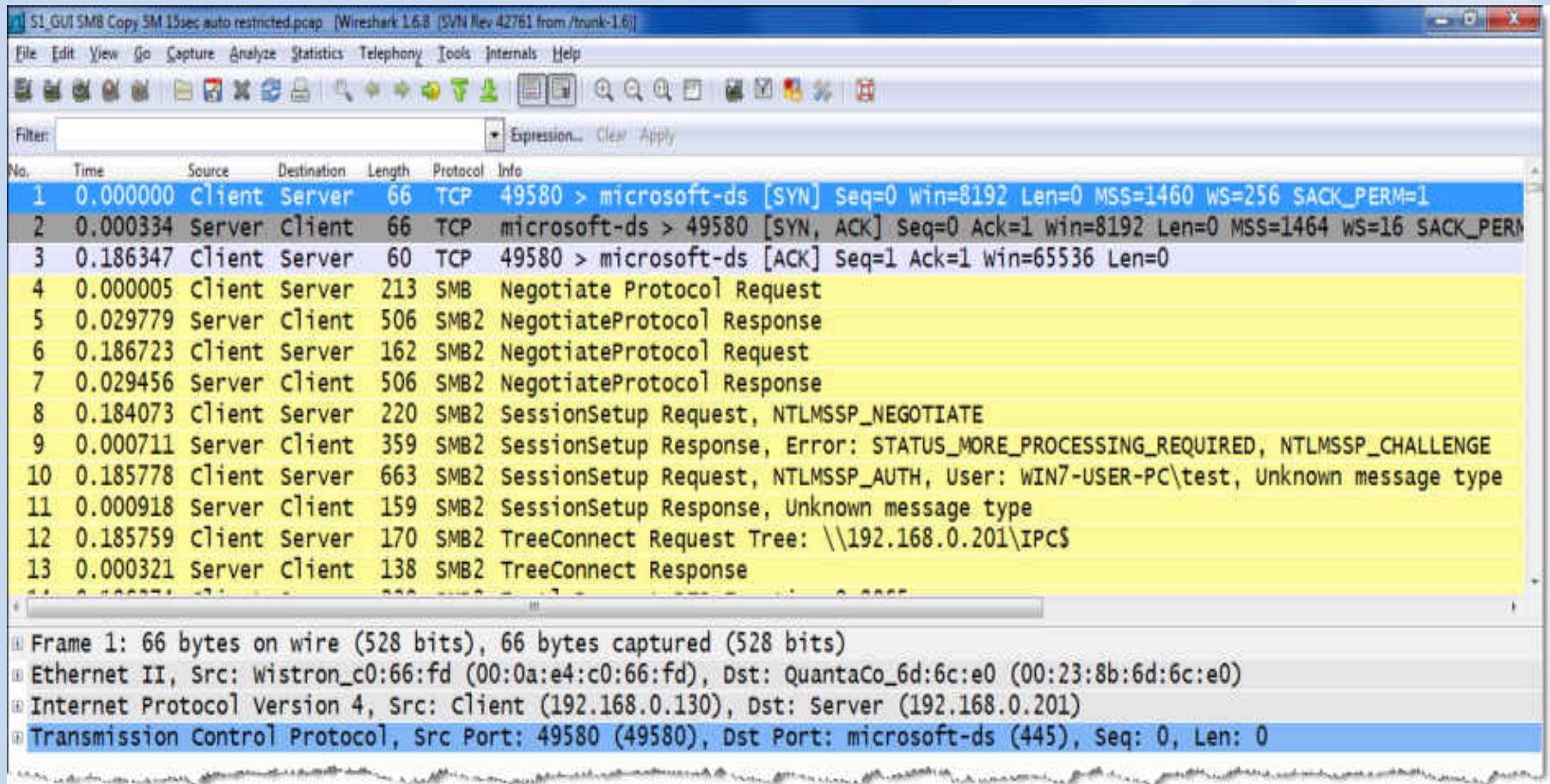
- Sometimes, a graphic tells us more than a thousand frames
- Wireshark offers excellent **graphical TCP session** presentations
- **TCP Stream Graph** allows to recognize all the following abnormalities:
 - Lost Frames
 - Duplicate Frames
 - Out of order Frames
 - TCP Sequence number and Segment Sizes
 - Acknowledges, Delayed Acknowledges
 - Duplicate and Selective Acknowledges
 - Retransmissions and Fast Retransmissions
 - Windows Sizes, sliding Window, exceeded und frozen Windows Size
 - Window Scaling, Zero Window and Window Full Situation
 - Slow Start, full Flow rate and Flow throttling

TCP Analysis with TCP Stream Graph



TCP Analysis with TCP Stream Graph

- Now, let us analyze our customer case using Frame Analysis



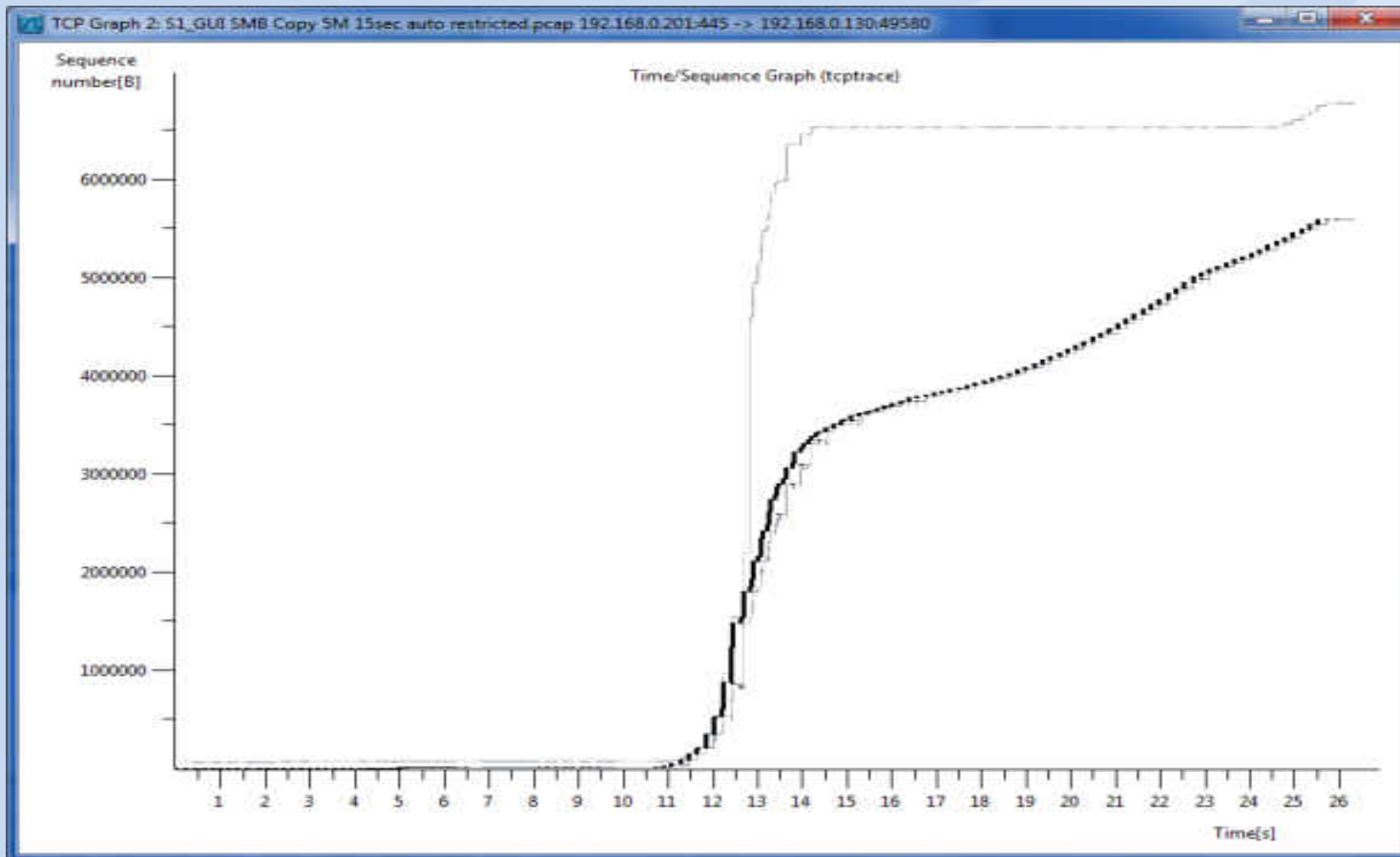
The image shows a Wireshark network traffic capture window. The main pane displays a list of captured packets. The first three packets are TCP SYN and ACK exchanges between a client and a server. The subsequent packets are SMB2 Negotiate Protocol requests and responses, followed by SMB2 Session Setup requests and responses, and finally SMB2 TreeConnect requests and responses.

No.	Time	Source	Destination	Length	Protocol	Info
1	0.000000	Client	Server	66	TCP	49580 > microsoft-ds [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
2	0.000334	Server	Client	66	TCP	microsoft-ds > 49580 [SYN, ACK] Seq=0 Ack=1 win=8192 Len=0 MSS=1464 WS=16 SACK_PERM=1
3	0.186347	Client	Server	60	TCP	49580 > microsoft-ds [ACK] Seq=1 Ack=1 Win=65536 Len=0
4	0.000005	Client	Server	213	SMB	Negotiate Protocol Request
5	0.029779	Server	Client	506	SMB2	NegotiateProtocol Response
6	0.186723	Client	Server	162	SMB2	NegotiateProtocol Request
7	0.029456	Server	Client	506	SMB2	NegotiateProtocol Response
8	0.184073	Client	Server	220	SMB2	SessionSetup Request, NTLMSSP_NEGOTIATE
9	0.000711	Server	Client	359	SMB2	SessionSetup Response, Error: STATUS_MORE_PROCESSING_REQUIRED, NTLMSSP_CHALLENGE
10	0.185778	Client	Server	663	SMB2	SessionSetup Request, NTLMSSP_AUTH, User: WIN7-USER-PC\test, Unknown message type
11	0.000918	Server	Client	159	SMB2	SessionSetup Response, Unknown message type
12	0.185759	Client	Server	170	SMB2	TreeConnect Request Tree: \\192.168.0.201\IPC\$
13	0.000321	Server	Client	138	SMB2	TreeConnect Response

Frame 1: 66 bytes on wire (528 bits), 66 bytes captured (528 bits)
Ethernet II, Src: Wistron_c0:66:fd (00:0a:e4:c0:66:fd), Dst: QuantaCo_6d:6c:e0 (00:23:8b:6d:6c:e0)
Internet Protocol Version 4, Src: Client (192.168.0.130), Dst: Server (192.168.0.201)
Transmission Control Protocol, Src Port: 49580 (49580), Dst Port: microsoft-ds (445), Seq: 0, Len: 0

TCP Analysis with TCP Stream Graph

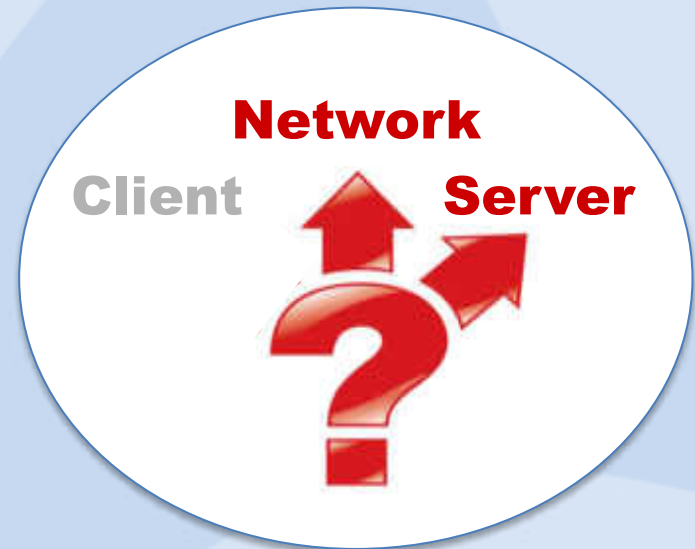
- Now, let us analyze our customer case using TCP Stream Graph



TCP Analysis with TCP Stream Graph

What can be read out of the trace file and the TCP Stream graph:

- Client and Server are both using **Window Scaling and Selective ACKs**
- The trace file has been captured on the **server side**
- The Round-Trip-Time is **186ms**
- The receiver (Client) window is **wide open**
- The network is **dropping** frames
- The server is **retransmitting** frames
- At this stage, we can **exclude the client !**



TCP Analysis with TCP Stream Graph

- TCP ,Three-way Handshake‘

Client SYN

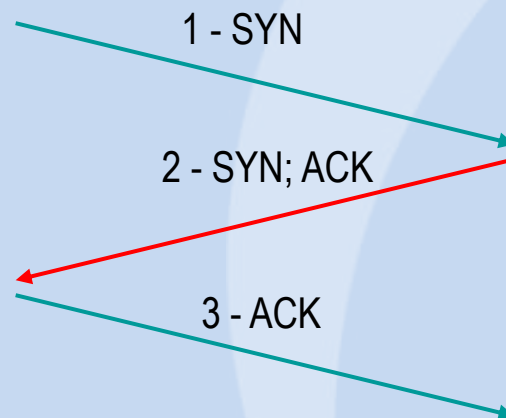
- Start Sequence Number
- Window Size

Options:

- Maximum Segment Size
- Window Scaling
- Selective Acknowledges
- Timestamp
- PAWS (Protection against wrapped sequence #)

Client ACK

- Acknowledge Server Sequence Number



Server SYN; ACK

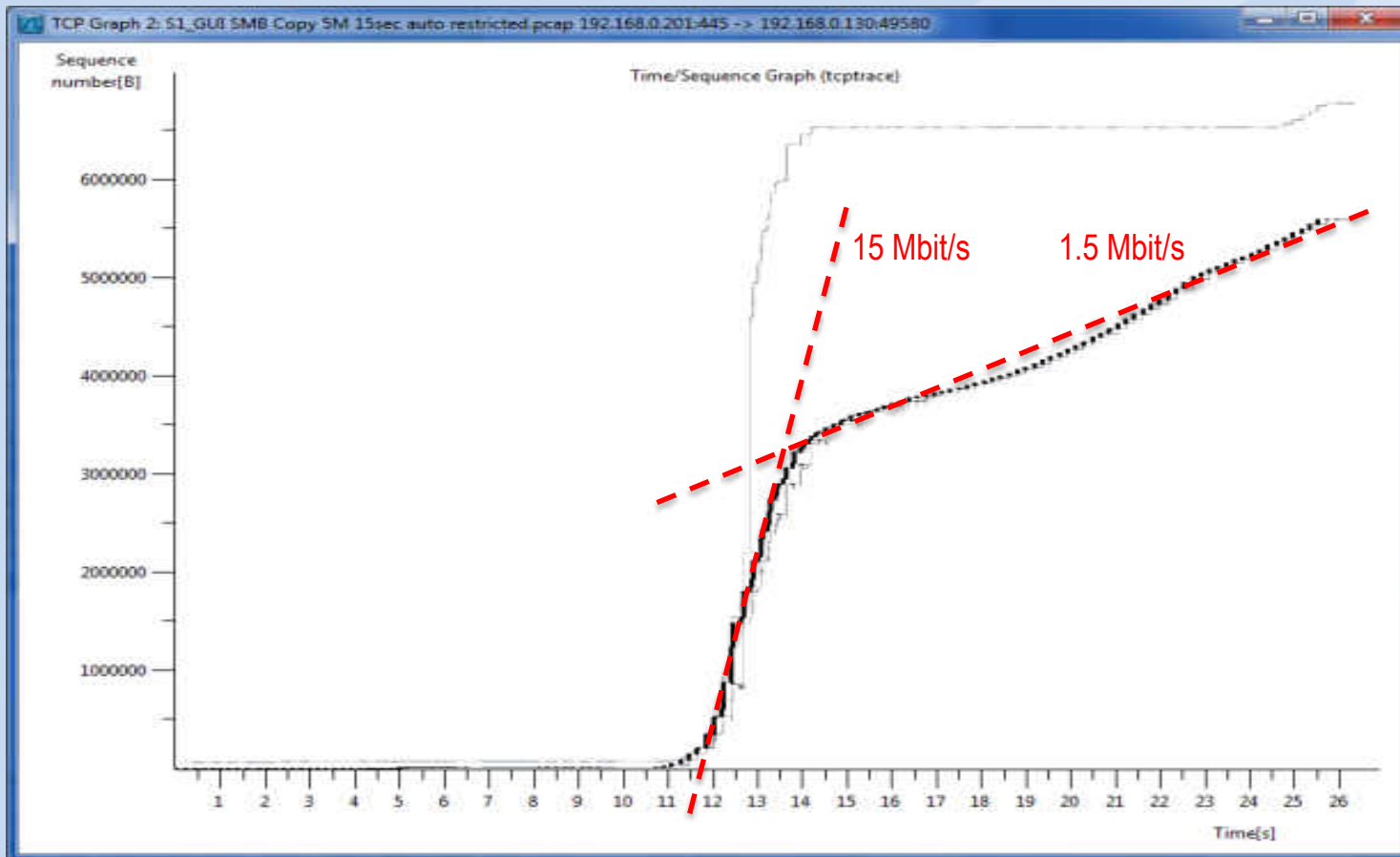
- Start Sequence Number
- Acknowledge Client Sequence Number
- Window Size

Options:

- Maximum Segment Size
- Window Scaling
- Selective Acknowledges
- Timestamp
- PAWS (Protection against wrapped sequence #)

TCP Analysis with TCP Stream Graph

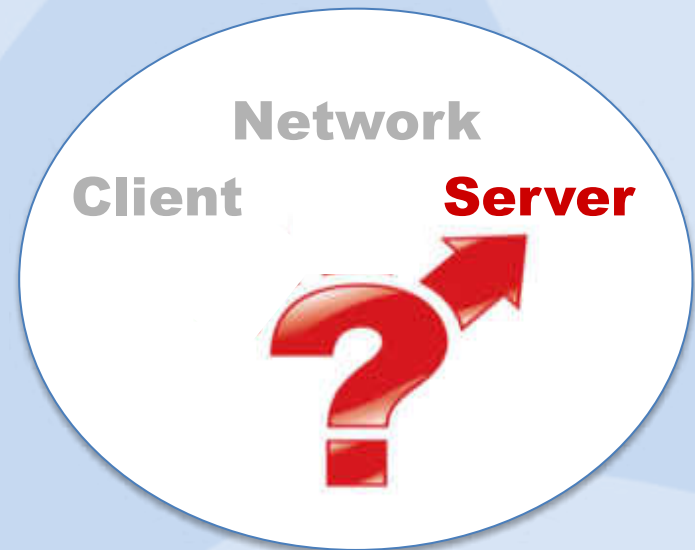
- Let us have a closer look at the servers behavior!



TCP Analysis with TCP Stream Graph

What can be read out of the TCP Stream graph:

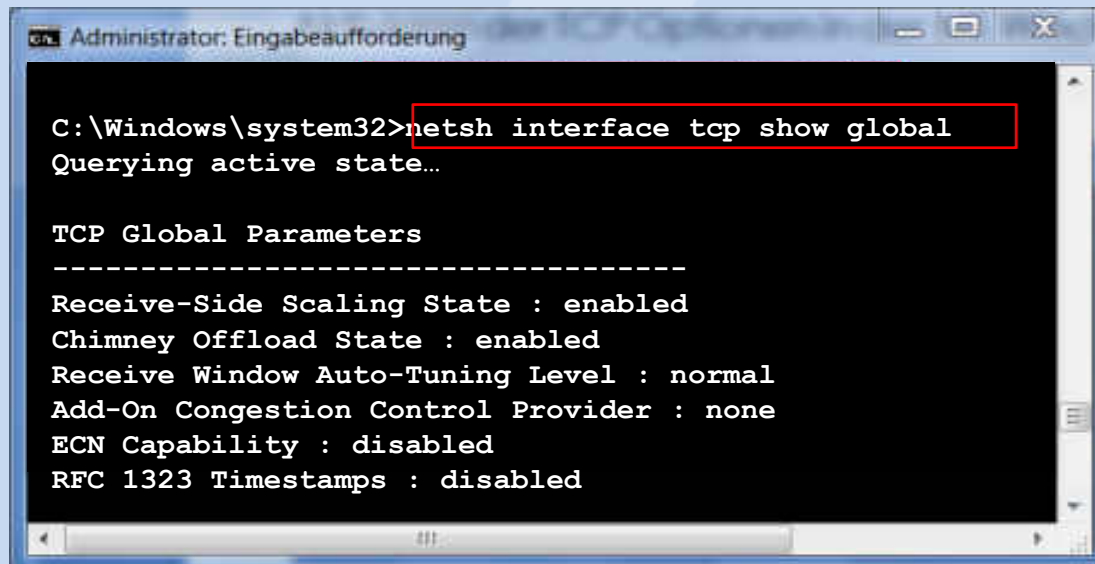
- The server is starting with **15Mbit/s** transmission rate
- The network is **dropping** some frames (pretty normal on WAN)
- Server is throttling down to **1.5 Mbit/s**
- Server is **not** trying to speed up again
- But why?
- At this stage we can **exclude the network**



MS Windows TCP Autotuning Features

Microsoft has implemented new autotuning in Vista, Win7, Server2008

- These features should improve TCP throughput and are ON by default
- However, this is not always the case, and may cause some Internet related issues and problems !



```
Administrator: Eingabeaufforderung
C:\Windows\system32>netsh interface tcp show global
Querying active state...

TCP Global Parameters
-----
Receive-Side Scaling State : enabled
Chimney Offload State : enabled
Receive Window Auto-Tuning Level : normal
Add-On Congestion Control Provider : none
ECN Capability : disabled
RFC 1323 Timestamps : disabled
```


MS Windows TCP Autotuning Features

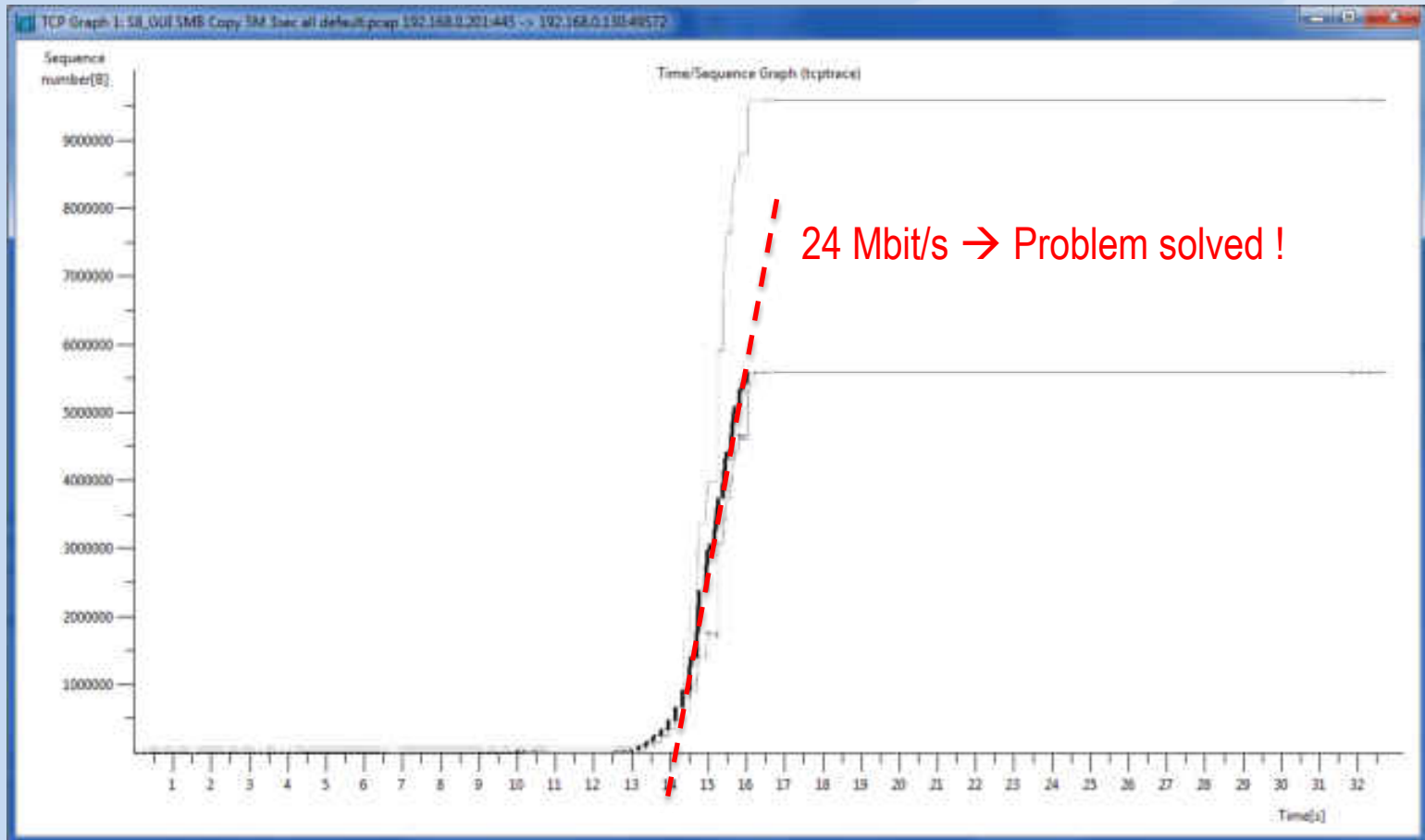
- **Autotuning**
 - Activate: `netsh interface tcp set global autotuning=normal`
 - Deactivate: `netsh interface tcp set global autotuning=disabled`
- **Compound TCP**
 - Activate: `netsh interface tcp set global congestionprovider=ctcp`
 - Deactivate: `netsh interface tcp set global congestionprovider=none`
- **ECN Support**
 - Activate: `netsh interface tcp set global ecncapability=enabled`
 - Deactivate: `netsh interface tcp set global ecncapability=disabled`
- **TCP Chimney offloading**
 - Activate: `netsh interface tcp set global chimney=enabled`
 - Deactivate: `netsh interface tcp set global chimney=disabled`
- **Receive-side Scaling (RSS)**
 - Activate: `netsh interface tcp set global rss=enabled`
 - Deactivate: `netsh interface tcp set global rss=disabled`

This command did **solve the issue** in our case:

- **Windows Scaling heuristics**
 - Deactivate: `netsh int tcp set heuristics disabled`
 - Activate: `netsh int tcp set heuristics enabled`

TCP Analysis with TCP Stream Graph

- Now let us have a closer look at the servers behavior again!



Thanks for visiting



Rolf Leutert, Leutert NetServices, www.wireshark.ch