

- **B-2 Analyzing TCP/IP Networks with Wireshark**

- June 15, 2010

Ray Tompkins

Founder of Gearbit | www.Gearbit.com

SHARKFEST '10

Stanford University

June 14-17, 2010



TCP

In this session we will examine the details of TCP,

How it Works:

how it sets up the connection,

how it keeps track of the data,

how it manages and controls the throughput,

how it recovers lost data,

TCP Overview

Connection Oriented:

Before data can be transferred, a TCP connection must be established.

Full Duplex:

Every TCP conversation has two logical pipes; an outgoing and incoming pipe.

Reliable:

All data is sequenced and lost packets are detected and retransmitted.

Byte Stream:

TCP views data transmitted over a pipe as a continuous stream of Bytes.

Sender and Receiver Flow Control:

A TCP Window is used to avoid sending too much data. This will be discussed in more detail in a later slide.

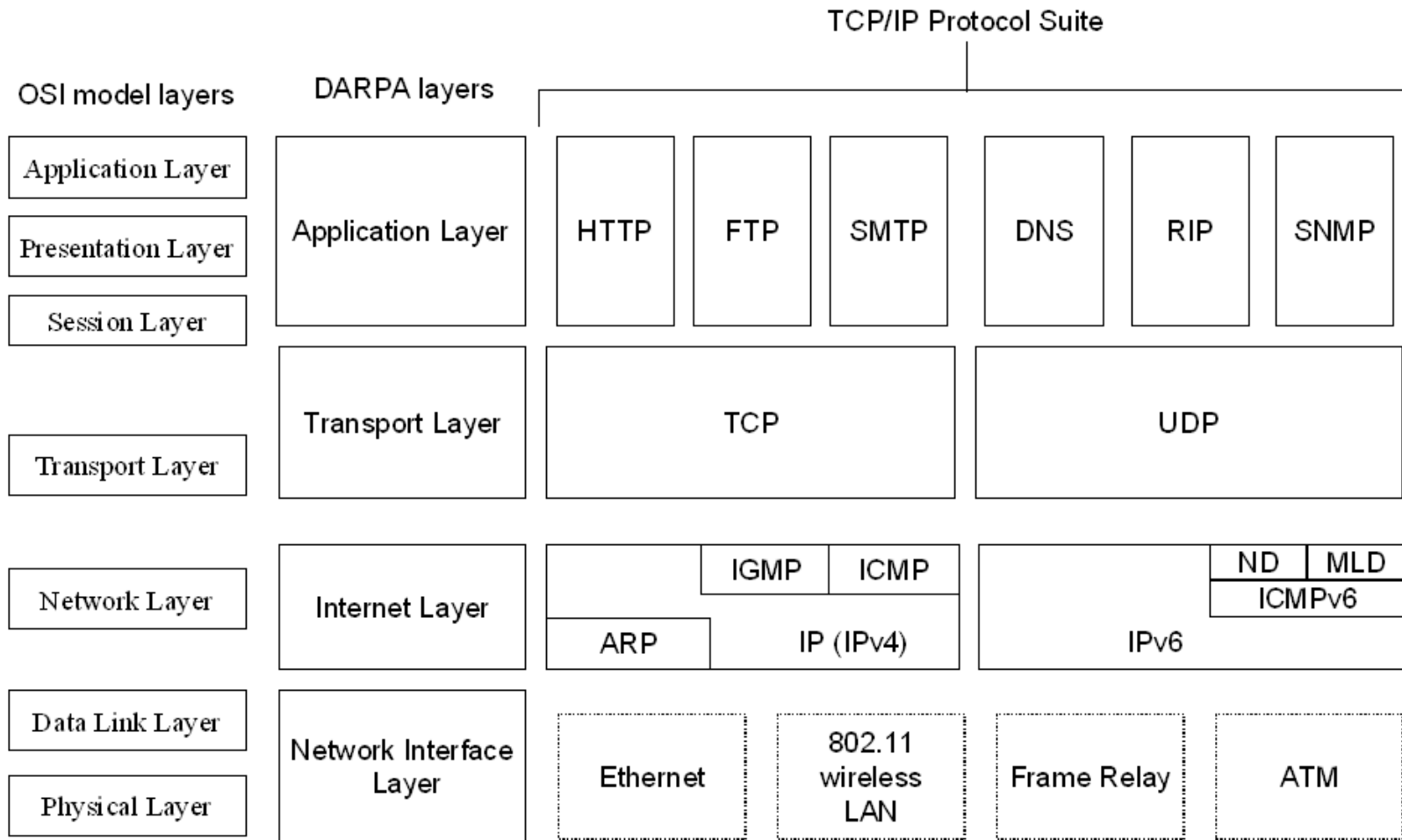
Segmentation:

TCP will segment any application data so that it will fit within the IP MTU.

TCP Header

- Source Port:** 2 Bytes to identify the source application layer protocol.
- Destination Port:** 2 Bytes to identify the destination application layer protocol.
- Sequence Number:** 4 Bytes. Indicates the outgoing bytes stream sequence number. When no data is to be sent the sequence number will be set to the next octet.
- Acknowledgement Number:** 4 Bytes. Provides a positive acknowledgement of all octets in the incoming byte stream.
- Data Offset:** 4 bits. Indicates where the TCP segment data begins.
- Reserved:** 6 bits. For future use.
- Flags:** 6 bits. Indicates one of six different flags.
- Window:** 2 Bytes. The number of Bytes available space in the receive buffer of the sender.
- Checksum:** 2 Bytes. 2 Byte field to provide a bit level integrity check.
- Urgent Pointer:** 2 Bytes. Indicates the location of urgent data in the segment.
- Options:** Indicates additional TCP Options.

TCP/IP Protocol Suite



TCP Three-way Handshake

The image shows a Wireshark capture of a TCP three-way handshake. The interface includes a menu bar (File, Edit, View, Go, Capture, Analyze, Statistics, Telephony, Tools, Help), a toolbar with various icons, and a filter field. The main display area shows a list of captured packets with columns for No., Time, Source, Destination, Protocol, and Info. The packets are as follows:

No.	Time	Source	Destination	Protocol	Info
1	0.000000	192.168.1.101	www.gearbit.com	TCP	trnsprntproxy > http [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=1
2	0.044776	www.gearbit.com	192.168.1.101	TCP	http > trnsprntproxy [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460 WS=1
3	0.044823	192.168.1.101	www.gearbit.com	TCP	trnsprntproxy > http [ACK] Seq=1 Ack=1 Win=65536 Len=0
4	0.045135	192.168.1.101	www.gearbit.com	HTTP	GET /index.shtml HTTP/1.1
5	0.093055	www.gearbit.com	192.168.1.101	TCP	http > trnsprntproxy [ACK] Seq=1 Ack=469 Win=6912 Len=0
6	0.096547	www.gearbit.com	192.168.1.101	TCP	[TCP segment of a reassembled PDU]
7	0.097701	www.gearbit.com	192.168.1.101	TCP	[TCP segment of a reassembled PDU]

The delta value between frames 1 and 2 can be used as a TCP transport connect baseline value.

Other important information gathered from this handshake:

- Window Size
- SACK
- Maximum Segment Size
- Window Scale Option value

TCP Close, FIN, ACK, FIN

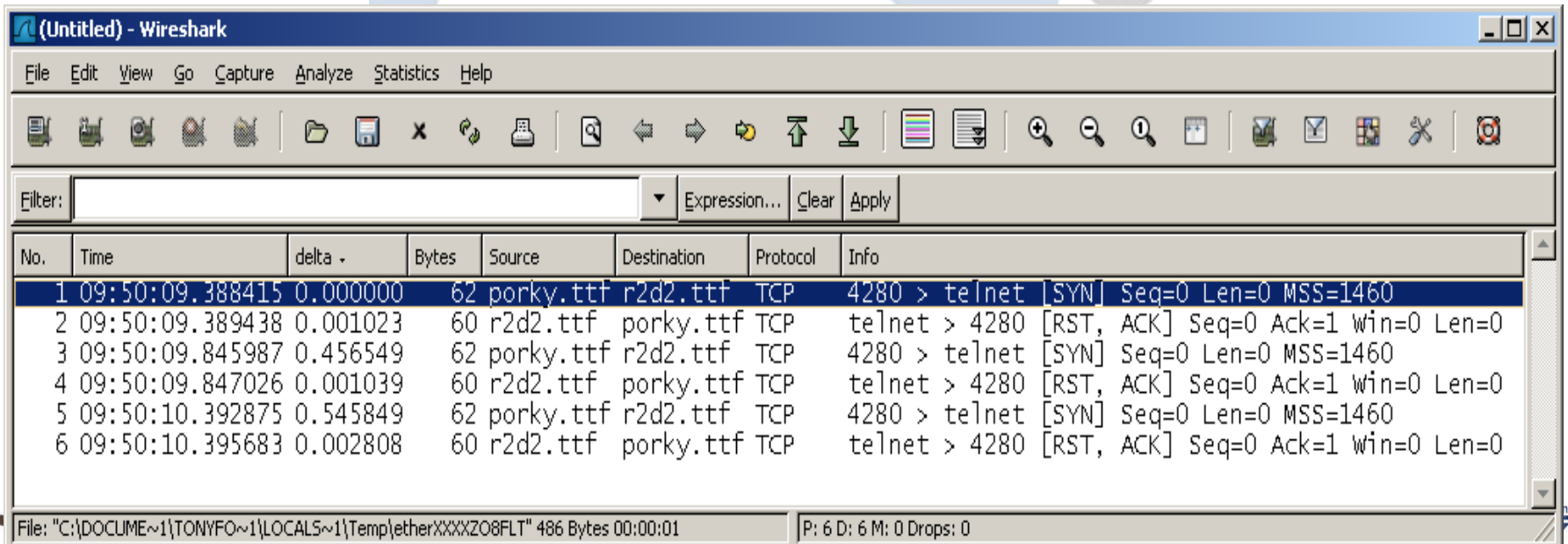
Packet #28 through #31 reveals the FIN, ACK, FIN that closes the TCP session

Filter: ip.addr == www.gearbit.com

No.	Time	Source	Destination	Protocol	Info
3	0.001209	192.168.1.101	www.gearbit.com	TCP	phoenix-rpc > http [SYN] Seq=0 win=65535 Len=0 MSS=1460 WS=1
4	0.046717	www.gearbit.com	192.168.1.101	TCP	http > phoenix-rpc [SYN, ACK] Seq=0 Ack=1 win=5840 Len=0 MSS=1460
5	0.046763	192.168.1.101	www.gearbit.com	TCP	phoenix-rpc > http [ACK] Seq=1 Ack=1 win=65536 Len=0
6	0.046936	192.168.1.101	www.gearbit.com	HTTP	GET /index.shtml HTTP/1.1
7	0.090948	www.gearbit.com	192.168.1.101	TCP	http > phoenix-rpc [ACK] Seq=1 Ack=469 win=6912 Len=0
8	0.095374	www.gearbit.com	192.168.1.101	TCP	[TCP segment of a reassembled PDU]
9	0.096596	www.gearbit.com	192.168.1.101	TCP	[TCP segment of a reassembled PDU]
10	0.096624	192.168.1.101	www.gearbit.com	TCP	phoenix-rpc > http [ACK] Seq=469 Ack=2921 win=65536 Len=0
11	0.143455	www.gearbit.com	192.168.1.101	HTTP	Continuation or non-HTTP traffic
12	0.143497	192.168.1.101	www.gearbit.com	TCP	phoenix-rpc > http [ACK] Seq=469 Ack=4381 win=65536 Len=0
13	0.144779	www.gearbit.com	192.168.1.101	HTTP	Continuation or non-HTTP traffic
14	0.146087	www.gearbit.com	192.168.1.101	HTTP	Continuation or non-HTTP traffic
15	0.146125	192.168.1.101	www.gearbit.com	TCP	phoenix-rpc > http [ACK] Seq=469 Ack=7301 win=65536 Len=0
16	0.191009	www.gearbit.com	192.168.1.101	HTTP	Continuation or non-HTTP traffic
17	0.192234	www.gearbit.com	192.168.1.101	HTTP	Continuation or non-HTTP traffic
18	0.192264	192.168.1.101	www.gearbit.com	TCP	phoenix-rpc > http [ACK] Seq=469 Ack=10221 win=65536 Len=0
19	0.194026	www.gearbit.com	192.168.1.101	HTTP	Continuation or non-HTTP traffic
20	0.273561	192.168.1.101	www.gearbit.com	HTTP	GET /favicon.ico HTTP/1.1
22	0.322570	www.gearbit.com	192.168.1.101	TCP	[TCP segment of a reassembled PDU]
23	0.323127	www.gearbit.com	192.168.1.101	HTTP	HTTP/1.1 404 Not Found (text/html)
24	0.323163	192.168.1.101	www.gearbit.com	TCP	phoenix-rpc > http [ACK] Seq=805 Ack=13197 win=65536 Len=0
28	3.984526	www.gearbit.com	192.168.1.101	TCP	http > phoenix-rpc [FIN, ACK] Seq=13197 Ack=805 win=8064 Len=0
29	3.984567	192.168.1.101	www.gearbit.com	TCP	phoenix-rpc > http [ACK] Seq=805 Ack=13198 win=65536 Len=0
30	10.324895	192.168.1.101	www.gearbit.com	TCP	phoenix-rpc > http [FIN, ACK] Seq=805 Ack=13198 win=65536 Len=0
31	10.368744	www.gearbit.com	192.168.1.101	TCP	http > phoenix-rpc [ACK] Seq=13198 Ack=806 win=8064 Len=0

SYN, RST

- This capture shows that the server is responding with a reset (RST).
- This indicates that the destination server is receiving the packet but there is no application bound to that port.
- Make sure that your application is bound to the correct port on the correct IP address.



Wireshark network capture showing a SYN flood attack. The capture shows alternating SYN and RST/ACK packets between porky.ttf and r2d2.ttf.

No.	Time	delta -	Bytes	Source	Destination	Protocol	Info
1	09:50:09.388415	0.000000	62	porky.ttf	r2d2.ttf	TCP	4280 > telnet [SYN] Seq=0 Len=0 MSS=1460
2	09:50:09.389438	0.001023	60	r2d2.ttf	porky.ttf	TCP	telnet > 4280 [RST, ACK] Seq=0 Ack=1 Win=0 Len=0
3	09:50:09.845987	0.456549	62	porky.ttf	r2d2.ttf	TCP	4280 > telnet [SYN] Seq=0 Len=0 MSS=1460
4	09:50:09.847026	0.001039	60	r2d2.ttf	porky.ttf	TCP	telnet > 4280 [RST, ACK] Seq=0 Ack=1 Win=0 Len=0
5	09:50:10.392875	0.545849	62	porky.ttf	r2d2.ttf	TCP	4280 > telnet [SYN] Seq=0 Len=0 MSS=1460
6	09:50:10.395683	0.002808	60	r2d2.ttf	porky.ttf	TCP	telnet > 4280 [RST, ACK] Seq=0 Ack=1 Win=0 Len=0

File: "C:\DOCUME~1\TONYFO~1\LOCAL5~1\Temp\etherXXXXZO8FLT" 486 Bytes 00:00:01 P: 6 D: 6 M: 0 Drops: 0

TCP Flag Information

Urgent (URG):

Indicates that the segment portion contains urgent data. The urgent data is found using the Urgent Pointer Field.

Acknowledgment (ACK):

The ACK flag is always set, except during the first phase of a TCP connection.

Push Function (PSH):

Indicates that the contents of the receive buffer should be immediately passed to the application layer.

Reset The Connection (RST):

Indicates the connection is to be aborted. (abnormal session connection).

Synchronize Sequence Number (SYN):

Indicates that this segment contains an Initial Sequence Number. This flag counts as **1 Byte**.

Finish Sending Data (FIN):

Indicates that the sender is finished sending data. When a FIN is transmitted, the receiver will respond with a FIN to close down the connection. This flag counts as **1 Byte**.

TCP Performance



TCP Algorithms:

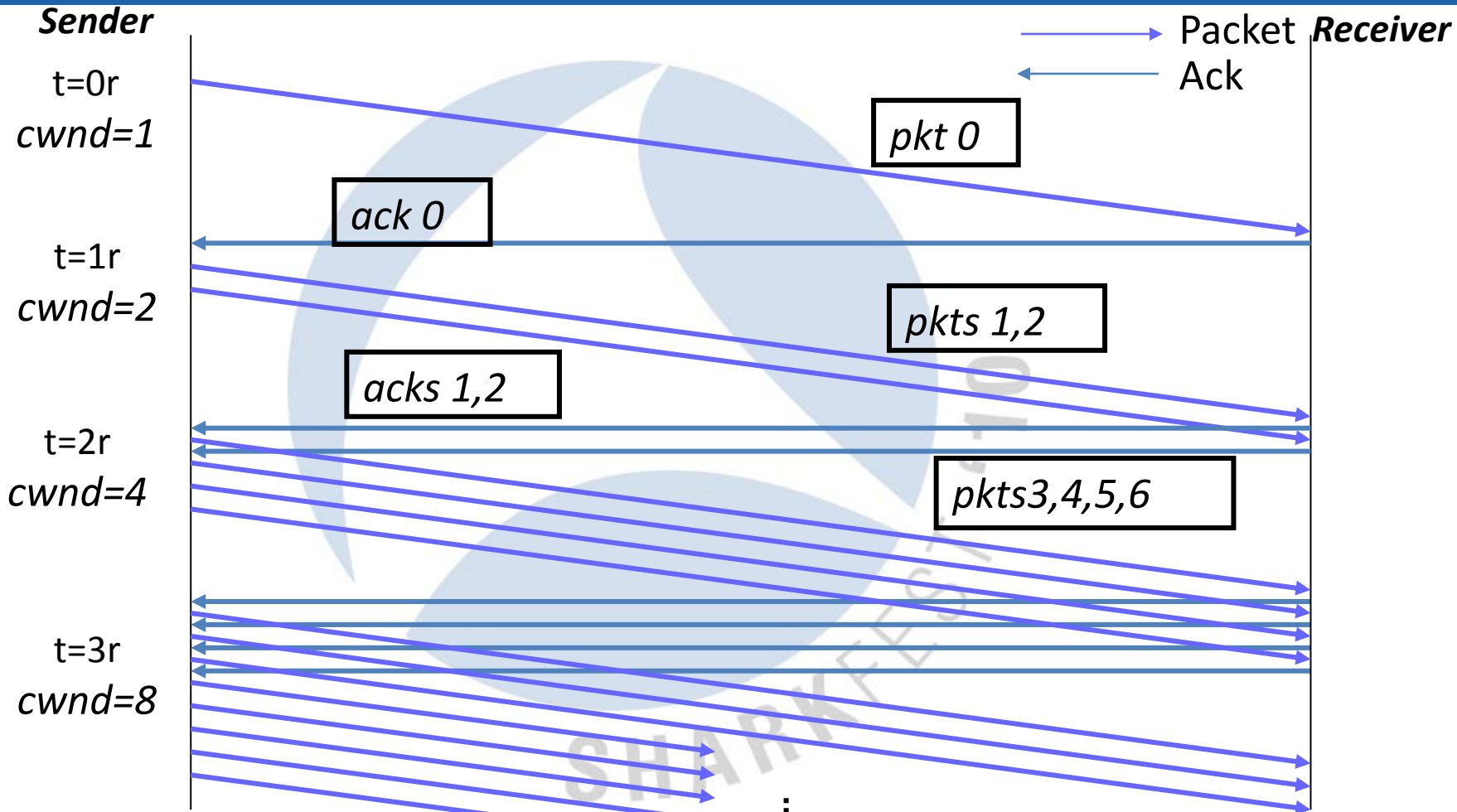
- **Slow Start** - Every *ack* increases the sender's window (*congestion window*) size by 1
- **Congestion Avoidance** - Reducing sender's window size by half at experience of loss, and increase the sender's window at the rate of about *one packet per RTT*
- **Fast Retransmit** - Don't wait for retransmit timer to go off, retransmit packet if *3 duplicate acks* received
- **Fast Recovery** - Since duplicate *ack* came through, one packet *has left the wire*. Perform *congestion avoidance*, don't jump down to *slow start*

TCP Slow Start Algorithm & Congestion Avoidance

- When a connection is established, TCP 'tests the waters' at first to assess the bandwidth of the connection and to avoid overflowing the receiving host, any other devices or links in the path.
- When a connection is successfully opened, only one packet is sent until an ACK is received.
- This continues until the amount of data being sent per burst reaches the size of the receive window on the remote host.
- At that point, the slow start algorithm is no longer in use and flow control is governed by the receive window. However, at any time during transmission, congestion could still occur on a connection. If this happens (evidenced by the need to retransmit), a congestion avoidance algorithm is used to reduce the send window size temporarily, and to grow it back towards the receive window size.
- Slow start and congestion avoidance are discussed in RFC 2001.

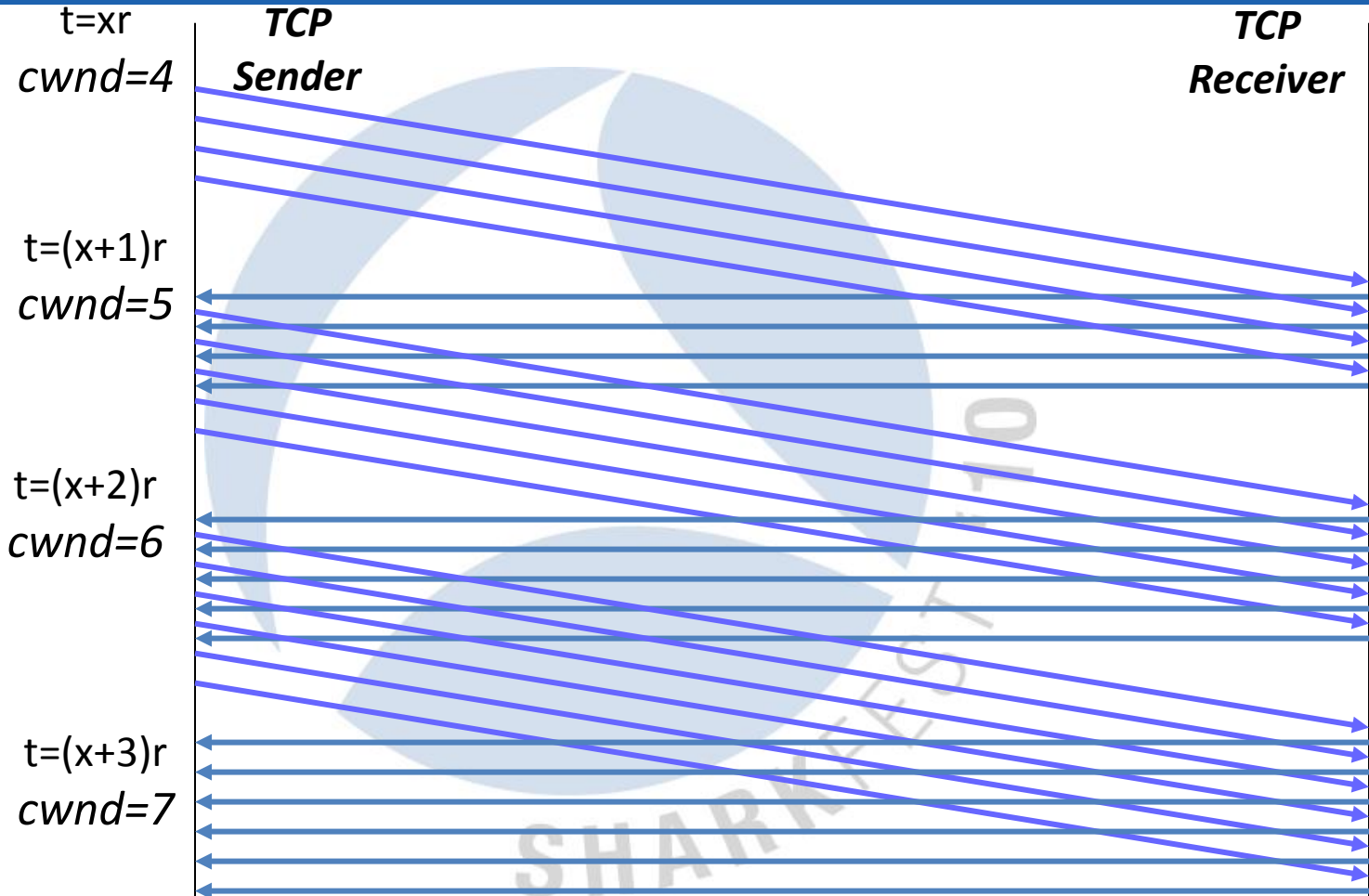
No.	Status	Source Address	Dest Address	Summary	Rel.
1	M	444553540000	Broadcast	ARP: C PA=[216.254.136.5] PRO=IP	0
2		[216.254.166.19]	[216.254.136.5]	DNS: C ID=1 OP=QUERY NAME=www.skiontario.on.ca	0
3		[216.254.136.5]	[216.254.166.19]	DNS: R ID=1 OP=QUERY STAT=OK NAME=www.skiontario.on.ca	0
4		444553540000	Broadcast	ARP: C PA=[206.172.13.28]skiontario.on.ca PRO=IP	0
5		[216.254.166.19]	skiontario.on.ca	TCP: D=80 S=1361 SYN SEQ=9740984 LEN=0 WIN=8192	0
6		skiontario.on.ca	[216.254.166.19]	TCP: D=1361 S=80 SYN ACK=9740985 SEQ=26380995	0
7		[216.254.166.19]	skiontario.on.ca	TCP: D=80 S=1361 ACK=2638099534 WIN=8192	0
8		[216.254.166.19]	skiontario.on.ca	HTTP: C Port=1361 GET /index.htm HTTP/1.1	0
9		skiontario.on.ca	[216.254.166.19]	TCP: D=1361 S=80 ACK=9741347 WIN=32334	0
10		skiontario.on.ca	[216.254.166.19]	HTTP: R Port=1361 HTML Data	0
11		[216.254.166.19]	skiontario.on.ca	TCP: D=80 S=1361 ACK=2638100070 WIN=8192	0
12		skiontario.on.ca	[216.254.166.19]	HTTP: R Port=1361 HTML Data	0
13		skiontario.on.ca	[216.254.166.19]	HTTP: R Port=1361 HTML Data	0
14		[216.254.166.19]	skiontario.on.ca	TCP: D=80 S=1361 ACK=2638101142 WIN=8192	0

Slow Start



New Congestion Window = Old Congestion Window + Number ACKs / Congestion Window

Congestion Avoidance



New Congestion Window = Old Congestion Window + Number ACKs / Congestion Window

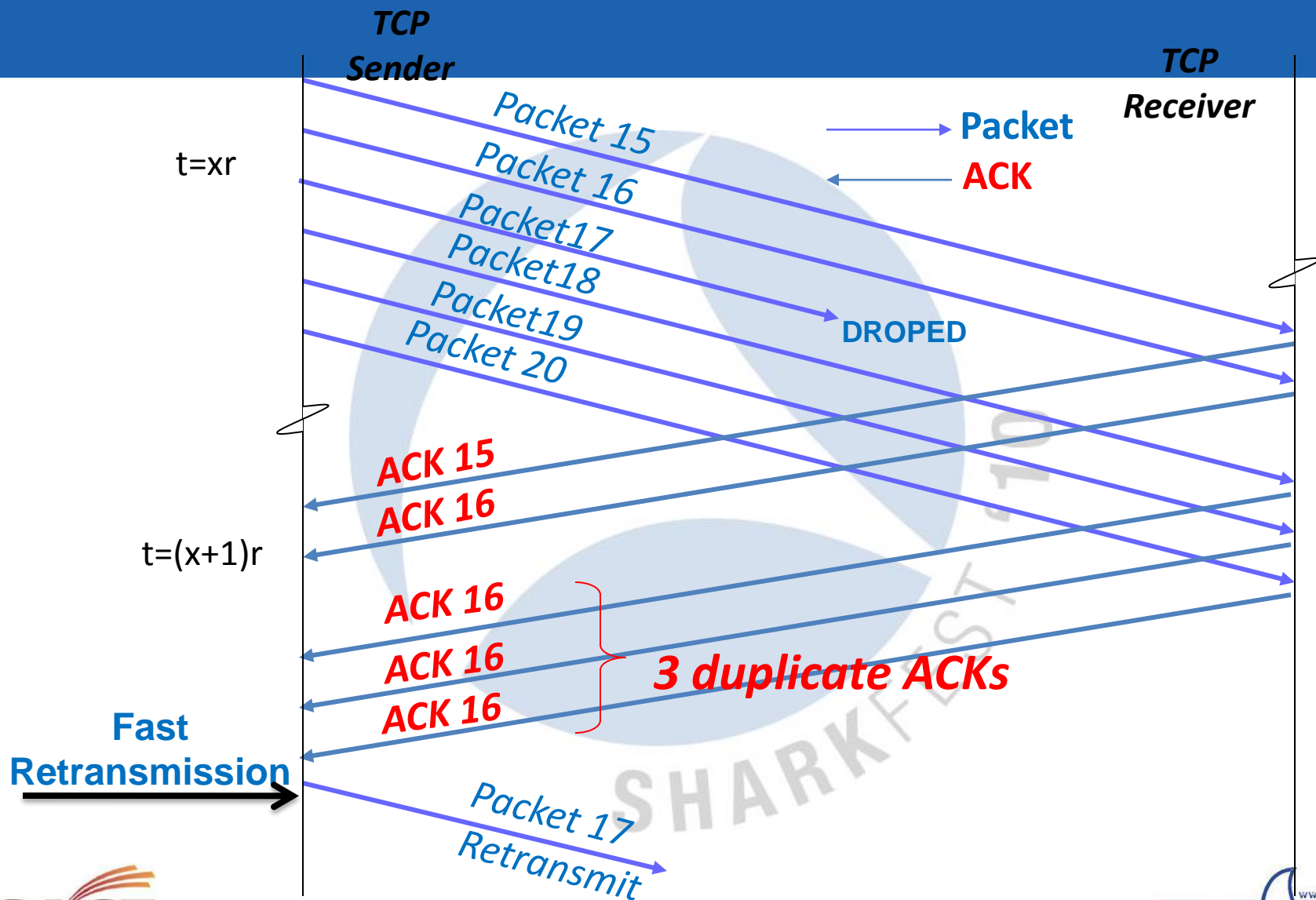
TCP Retransmission Behavior

- TCP starts a retransmission timer when each outbound segment is handed down to IP.
- If no acknowledgment has been received for the data in a given segment before the timer expires, then the segment is retransmitted.
- For new connection requests, the retransmission timer is initialized to 3 seconds, and the request (SYN) is resent up to `TcpMaxConnectRetransmissions` times.
 - (the default for Windows 2000 & XP is 2 times, Windows NT 4.0 is 3 times).
- On existing connections, the number of retransmissions is controlled by the `TcpMaxDataRetransmissions` registry parameter (5 by default).
- The retransmission time-out is adjusted "on the fly" to match the characteristics of the connection using Smoothed Round Trip Time (SRTT) calculations as described in RFC 793. The timer for a given segment is doubled after each retransmission of that segment. Using this algorithm, TCP tunes itself to the "normal" delay of a connection. TCP connections over high-delay links will take much longer to time out than those over low-delay links.
- By default, Windows XP resends a segment if it receives three ACKs for the same sequence number and that sequence number lags the current one. This is controllable with the `TcpMaxDupAcks` registry parameter.

TCP Fast Retransmit Behavior

- There are some circumstances under which TCP will retransmit data prior to the retransmission timer expiring. The most common of these occurs due to a feature known as fast retransmit.
- When a receiver that supports fast retransmit receives data with a sequence number beyond the current expected one, then it is likely that some data was dropped. To help make the sender aware of this event, the receiver immediately sends an ACK, with the ACK number set to the sequence number that it was expecting.
- It will continue to do this for each additional TCP segment that arrives containing data subsequent to the missing data in the incoming stream. When the sender starts to receive a stream of ACKs that are acknowledging the same sequence number, and that sequence number is earlier than the current sequence number being sent, it can infer that a segment (or more) must have been dropped. Senders that support the fast retransmit algorithm will immediately resend the segment that the receiver is expecting to fill in the gap in the data, without waiting for the retransmission timer to expire for that segment. This optimization greatly improves performance in a lousy network environment. By default, Windows XP resends a segment if it receives three ACKs for the same sequence number, and that sequence number lags the current one. This is controllable with the TcpMaxDupAcks registry parameter.

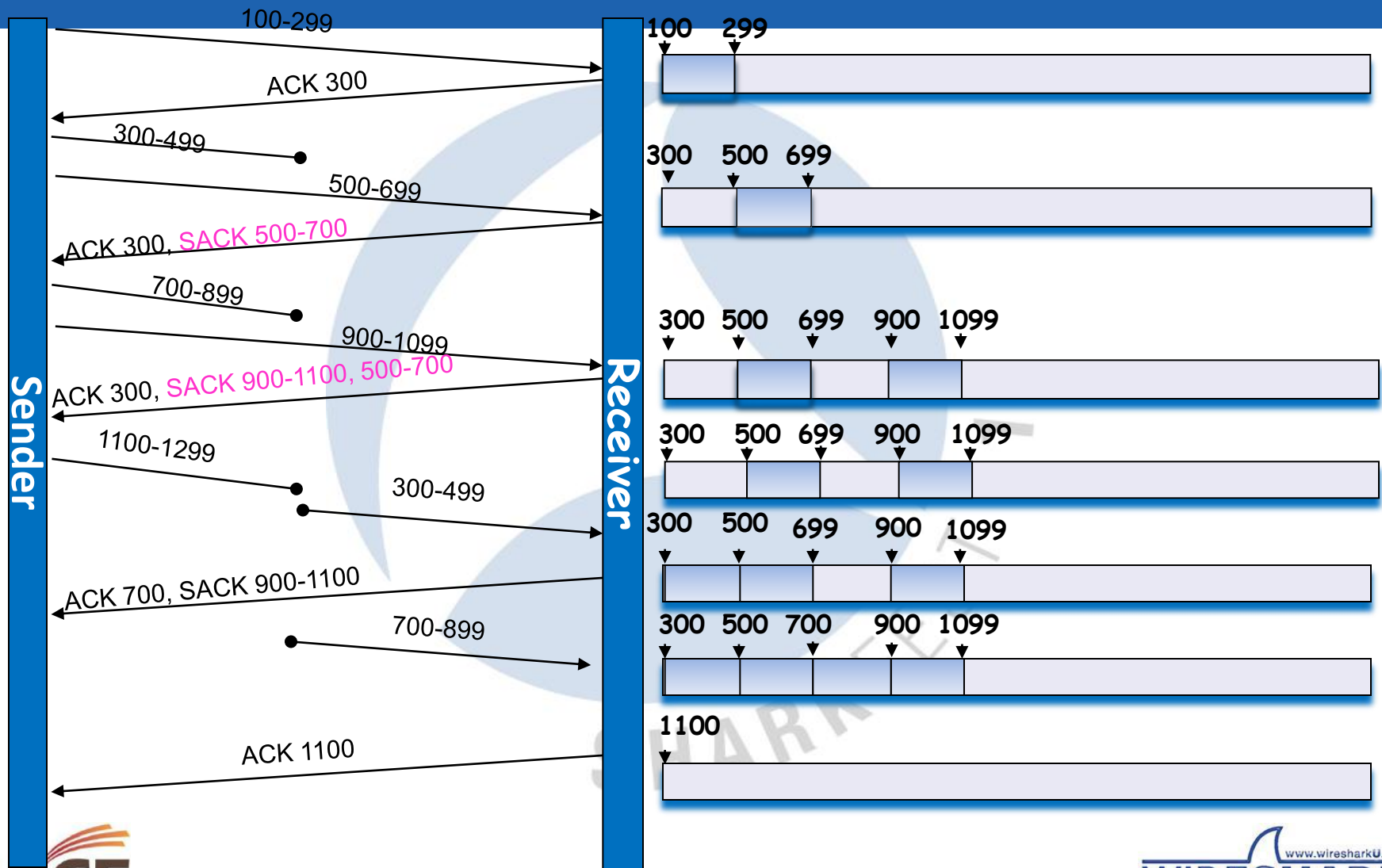
Fast Retransmit



Selective Acknowledgements (SACKS)

- Selective Acknowledgment (SACK) is a mechanism that includes a retransmission algorithm which helps overcome weak links on the TCP/IP stack. The selective acknowledgment extension uses two TCP options.
 - The first is a two-byte enabling option, SACK-permitted, which can be sent in a SYN segment to indicate that the SACK option can be used once the connection is established.
 - The second option is the SACK option itself, which can be sent over an established connection once both the sender and the receiver have successfully negotiated the SACK-permitted option. Whenever there is loss of data, the data receiver can send the SACK option to acknowledge the out-of-order segments. The data blocks are identified using the sequence number at the start and at the end of that block of data. This is also known as the left and right edge of the block of data.
- The Novell and Microsoft TCP/IP stack supports SACK per RFC 1323. The use of SACK is helpful in a scenario where there is a heavy flow of traffic and some packets are getting lost. With SACK, the sender doesn't have to resend all the packets that were sent after one lost packet. He can selectively resend only the packets that were lost. SACK has no impact on a LAN connection's performance.
- SACK is especially important for connections that use large TCP window sizes. Prior to SACK, a receiver could only acknowledge the latest sequence number of a contiguous data stream that had been received, or the "left edge" of the receive window. With SACK enabled, the receiver continues to use the ACK number to acknowledge the left edge of the receive window, but it can also acknowledge other blocks of received data individually.

Selective Acknowledgments (SACK)



TCP Keep-Alive Messages

- A TCP keep-alive packet is simply an ACK with the sequence number set to one less than the current sequence number for the connection.
- A host receiving one of these ACKs will respond with an ACK for the current sequence number.
- Keep-Alives can be used to verify that the computer at the remote end of a connection is still available.
- TCP keep-Alives can be sent once every KeepAliveTime (defaults to 7,200,000 milliseconds or two hours), if no other data or higher level keep-alives have been carried over the TCP connection. If there is no response to a keep-alive, it is repeated once every KeepAliveInterval seconds. KeepAliveInterval defaults to 1 second.
- NetBT connections, such as those used by many Microsoft networking components, send NetBIOS keep-alives more frequently, so normally no TCP keep-alives will be sent on a NetBIOS connection.
- TCP keep-alives are disabled by default, but Windows Sockets applications can use the SetSockOpt function to enable them.

TCP Window And Timers

- Because TCP guarantees delivery and reliability of traffic flow, the window cannot slide past any data that has not been acknowledged. If the window cannot slide beyond a packet of data, no more data beyond the window is transmitted, TCP eventually has to shut down the session, and the communication fails.
- Each machine is therefore instructed to wait a certain amount of time before either retransmitting data or sending acknowledgments for packets that arrive out of sequence. Each window is given a timer:
 - the send window has the Retransmit Timer
 - the receive window has the Delayed Acknowledgment Timer
- These timers help define what to do when communication isn't flowing very smoothly

TCP Sending Window Timer

- In the sending window, a Retransmit Timer is set for each packet, specifying how long to wait for an acknowledgment before making the assumption that the packet did not get to its destination. After this timer has expired, the send window is instructed to resend the packet and wait twice as long as the time set on the preceding timer. The default starting point for this timer is approximately 3 seconds but is usually reduced to less than a second almost immediately. Each time an acknowledgment is not received, the Retransmit Timer doubles. For instance, if the Retransmit Timer started at approximately 1 second, the second Retransmit Timer is set for 2 seconds, the third for 4 seconds, the fourth, 8 seconds, up to a fifth attempt that waits 16 seconds. The number of attempts can be altered in the Registry, but if after these attempts an acknowledgment still cannot be received, the TCP session is closed and errors are reported to the application.
- Illustrates the resending of data after the first Retransmit Timer has expired.
- The Registry location for changing the number of times to retry a transmission is in the following subkey:
 - HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters
- The Registry parameter and value is:
 - TcpMaxDataRetransmissions (REG_DWORD)
- The default value is 5.

TCP Receive Window Timer

- In the receiving window, a Delayed Acknowledgment Timer is set for those packets that arrive out of order. Remember, by default an acknowledgment is sent for every two sequenced packets, starting from the left-hand side of the window. If packets arrive out of order (if, for instance, 1 and 3 arrive but 2 is missing), an acknowledgment for two sequenced packets is not possible. When packets arrive out of order, a Delayed Acknowledgment Timer is set on the first packet in the pair. In the parenthetical example, a Timer is set on packet number 1.
- The Delayed Acknowledgment Timer is hard-coded for 200 milliseconds, or $1/5$ the Retransmit Timer. If packet 2 does not show up before the Delayed Acknowledgment Timer expires, an acknowledgment for packet 1, and only packet 1, is sent. No other acknowledgments are sent, including those for packets 3 through 8 that might have appeared. Until packet 2 arrives, the other packets are considered interesting, but useless. As data is acknowledged and passed to the Application layer, the receive window slides to the right, enabling more data to be received. Again though, if a packet doesn't show up, the window is not enabled to slide past it.

Configuring Delayed Acknowledgments

- A problem may occur with message block (SMB) write operations to a Windows XP - based domain controller and may experience a delay of up to 200 milliseconds between file copies.
- If you review a trace of the problem, you notice that the delay occurs after the client sends the server an SMB Notify Change command with the FID entry that matches the FID entry of the target folder.
- Windows Explorer posts a Notify Change request on the network share, which asks to be notified if something changes in the folder that appears in the right pane of Windows Explorer. If a domain controller receives the Notify Change request, it does not respond to it immediately; it does not send packets for up to 200 milliseconds. At that point, a simple Transmission Control Protocol (TCP) acknowledgement (ACK) packet is sent and the file operation resumes as usual.
- This behavior is a result of the interaction between two core networking components of Windows XP, TCP delayed ACKs, and thread prioritization on domain controllers.
- The Windows XP-based domain controller, you can edit the TcpDelAckTicks registry value to adjust the TCP delayed ACK timer. If you change the TCP delayed ACK timer to a lower value, the server sends an ACK packet more frequently but at shorter intervals.
 1. Start Registry Editor (Regedt32.exe).
 2. Locate and click the following key in the registry, where Adapter GUID is the globally unique identifier (GUID) for the network adapter that connects to the clients:
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters\Interfaces\Adapter GUID
 3. On the Edit menu, click Add Value, and then add the following registry value:
Value name: TcpDelAckTicks
Data type: REG_DWORD
Value data: You can set this value to a range from 0 to 6. The default setting is 2 (200 milliseconds).
 4. Quit Registry Editor.
 5. Restart Windows for this change to take effect.

TCP Window Information

- The Expert reports “Window Frozen” statistics.
 - Look to see if one particular device is freezing when the window is small. Why is it unable to use a reasonable size window?
- In the “Silly Window Syndrome,” the receiver keeps advertising a small window and sender keeps filling it with small packets.
 - Can you configure it to use a larger size?
- “Zero Window” symptom alerts you to stations that have closed their window.
 - Don’t worry if the window closes briefly at the beginning of a connection, then opens and maintains a reasonable size.
 - Do worry if a host frequently closes the window for long periods of time.
 - Do you see the window gradually growing smaller before it closes?
- A TCP MAY keep its offered receive window closed indefinitely. As long as the receiving TCP continues to send acknowledgments in response to the probe segments, the sending TCP MUST allow the connection to stay open.

TCP Window Scaling Option

- The computer will send a packet offering the Window Scale option, with a scaling factor of up to 5. If the target computer responds, accepting the Window Scale option in the SYN-ACK, then it is understood that any TCP window advertised by this computer needs to be left-shifted 5 bits from this point onward (the SYN itself is not scaled).
- The Large Windows option defines an implicit scale factor, which is used to multiply the window size value found in a TCP header to obtain the true window size. The TCP/IP stack supports a maximum window size of 1 GB. This Large Window option is negotiated when the TCP connection is established.
- The TCP Large Windows size is useful on fast networks (such as Gigabit Ethernet) with large round-trip times. To understand how this works, think of a water hose. To achieve maximum water flow, the hose should be full. As the hose increases in diameter and length, the volume of water necessary to keep it full increases. In networks, diameter equates to bandwidth, length is measured as round-trip time, and the volume of water is analogous to the TCP window size. On fast networks with large round-trip times, the TCP window size must be increased to achieve maximum TCP bandwidth.
- TCP performance depends not upon the transfer rate itself, but rather upon the product of the transfer rate and the round-trip delay. This "bandwidth delay product" measures the amount of data that would fill the pipe. It is the buffer space required at the sender and the receiver to obtain maximum throughput on the TCP connection over the path—in other words, the amount of unacknowledged data that TCP must handle in order to keep the pipeline full. So on fast networks with large round-trip times, having a large TCP Window helps by allowing for a greater amount of unacknowledged data.
- Windows XP uses window scaling automatically if the `TcpWindowSize` is set to a value greater than 64 KB, and the `Tcp1323Opts` registry parameter is set appropriately.

Window Scaling Option Reference Chart

Scale Factor	Scale Value	Initial Window	Window Scaled
0	1	65535 or less	65535 or less
1	2	65535	131,070
2	4	65535	262,140
3	8	65535	524,280
4	16	65535	1,048,560
5	32	65535	2,097,120
6	64	65535	4,194,240
7	128	65535	8,388,480
8	256	65535	16,776,960
9	512	65535	33,553,920
10	1024	65535	67,107,840
11	2048	65535	134,215,680
12	4096	65535	268,431,360
13	8192	65535	536,862,720
14	16384	65535	1,073,725,440

Protection Against Wrapped Sequence Numbers (PAWS)

- The TCP sequence number field is limited to 32 bits, which limits the number of sequence numbers available. With high capacity networks and a large data transfer, it is possible to wrap sequence numbers before a packet traverses the network.
- If sending data on a 1 Gigabit per second (Gbps) network, the sequence numbers could wrap in as little as 34 seconds. If a packet is delayed, a different packet could potentially exist with the same sequence number.
- To avoid confusion in the event of duplicate sequence numbers, the TCP timestamp is used as an extension to the sequence number. Packets have current and progressing time stamps. An old packet has an older time stamp and is discarded.

TCP Zero Window Example

- When you get into a Zero Window situation, it is quite normal to see the transmitting station send 1 byte packets.

No.	Dest Address	Source Address	Summary	Delta Time	F
28	[192.168.1.210]	[192.168.1.216]	HTTP: R Port=1151 Graphics Data	0.003.044	
29	[192.168.1.210]	[192.168.1.216]	HTTP: R Port=1151 Graphics Data	0.001.229	
30	[192.168.1.210]	[192.168.1.216]	HTTP: R Port=1151 Graphics Data	0.001.259	
31	[192.168.1.216]	[192.168.1.210]	TCP: D=80 S=1151 ACK=334277 WIN=2920	0.000.066	
32	[192.168.1.216]	[192.168.1.210]	TCP: D=80 S=1151 ACK=337197 WIN=0	0.000.586	
33	[192.168.1.210]	[192.168.1.216]	TCP: Retransmitted in frame 35; 21 Bytes of data	0.996.467	
34	[192.168.1.216]	[192.168.1.210]	TCP: D=80 S=1151 ACK=337197 WIN=0	0.000.413	
35	[192.168.1.210]	[192.168.1.216]	TCP: Retransmitted in frame 38; 21 Bytes of data	2.002.551	
36	[192.168.1.216]	[192.168.1.210]	TCP: D=80 S=1151 ACK=337197 WIN=0	0.000.397	
37	[192.168.1.216]	[192.168.1.210]	TCP: D=80 S=1151 ACK=337197 WIN=8760	2.574.534	
38	[192.168.1.210]	[192.168.1.216]	HTTP: R Port=1151 Graphics Data	0.001.578	
39	[192.168.1.210]	[192.168.1.216]	HTTP: R Port=1151 Graphics Data	0.001.229	
40	[192.168.1.210]	[192.168.1.216]	HTTP: R Port=1151 Graphics Data	0.001.236	

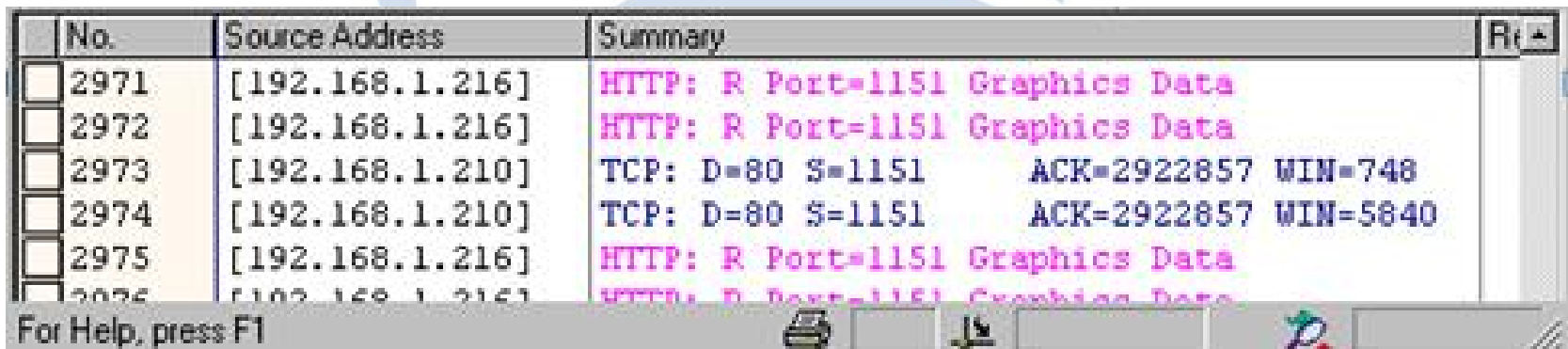
For Help, press F1

TCP Window XP Information

- The Windows XP TCP/IP stack was designed to tune itself in most environments and uses larger default window sizes than earlier versions. Instead of using a hard-coded default receive window size, TCP adjusts to even increments of the maximum segment size (MSS) negotiated during connection setup. Matching the receive window to even increments of the MSS increases the percentage of full-sized TCP segments used during bulk data transmission.
- The receive window size defaults to a value calculated as follows:
 - The first connection request sent to a remote host advertises a receive window size of 16 KB (16,384 bytes).
 - Upon establishing the connection, the receive window size is rounded up to an increment of the maximum TCP segment size (MSS) that was negotiated during connection setup.
 - If that is not at least four times the MSS, it is adjusted to $4 * MSS$, with a maximum size of 64 KB unless a window scaling option (RFC 1323) is in effect.
- For Ethernet, the window is normally set to 17,520 bytes (16 KB rounded up to twelve 1460-byte segments.) There are two methods for setting the receive window size to specific values:
 - The TcpWindowSize registry parameter

TCP Silly Window Syndrome (SWS)

- Silly Window Syndrome is described in RFC 1122.
- In brief, SWS is caused by the receiver advancing the right window edge whenever it has any new buffer space available to receive data and by the sender using any incremental window, no matter how small, to send more data.
- The result can be a stable pattern of sending tiny data segments, even though both sender and receiver have a large total buffer space for the connection.
- Windows XP TCP/IP implements SWS avoidance as specified in RFC 1122 by not sending more data until there is a sufficient window size advertised by the receiving end to send a full TCP segment. It also implements SWS on the receive end of a connection by not opening the receive window in increments of less than a TCP segment.

A screenshot of a Wireshark packet capture showing a sequence of small data segments. The packets are numbered 2971 through 2975. Packets 2971, 2972, and 2975 are HTTP responses from port 1151 containing graphics data. Packets 2973 and 2974 are TCP segments from port 1151 to port 80, both with ACK=2922857. Packet 2973 has a window size of 748, while packet 2974 has a window size of 5840, indicating a significant increase in the receiver's advertised window size between these two packets, which is characteristic of SWS.

No.	Source Address	Summary	Filter
2971	[192.168.1.216]	HTTP: R Port=1151 Graphics Data	
2972	[192.168.1.216]	HTTP: R Port=1151 Graphics Data	
2973	[192.168.1.210]	TCP: D=80 S=1151 ACK=2922857 WIN=748	
2974	[192.168.1.210]	TCP: D=80 S=1151 ACK=2922857 WIN=5840	
2975	[192.168.1.216]	HTTP: R Port=1151 Graphics Data	
2976	[192.168.1.216]	HTTP: R Port=1151 Graphics Data	

For Help, press F1

TCP Nagle Algorithm

- Windows NT and Windows XP TCP/IP implement the Nagle algorithm described in RFC 896.
- The purpose of this algorithm is to reduce the number of very small segments sent, especially on high-delay (remote) links.
- The Nagle algorithm allows only one small segment to be outstanding at a time without acknowledgment. If more small segments are generated while awaiting the ACK for the first one, then these segments are combined into one larger segment.
- Any full-sized segment is always transmitted immediately, assuming there is a sufficient receive window available.
- The Nagle algorithm is effective in reducing the number of packets sent by interactive applications, such as Telnet, especially over slow links.
- Windows Sockets applications can disable the Nagle algorithm for their connections by setting the TCP_NODELAY socket option. However, this practice should be avoided unless absolutely necessary as it increases network utilization. Some network applications may not perform well if their design does not take into account the effects of transmitting large numbers of small packets and the Nagle algorithm.

TCP Delayed ACK

- A TCP SHOULD implement a delayed ACK, but an ACK should not be excessively delayed; in particular, the delay MUST be less than 0.5 seconds, and in a stream of full-sized segments there SHOULD be an ACK for at least every second segment.
- As specified in RFC 1122, TCP uses delayed acknowledgments (ACKs) to reduce the number of packets sent on the media. The Microsoft TCP/IP stack takes a common approach to implementing delayed ACKs. As data is received by TCP on a connection, it only sends an acknowledgment back if one of the following conditions is met:
 - No ACK was sent for the previous segment received.
 - A segment is received, but no other segment arrives within 200 milliseconds for that connection.
- A delayed ACK gives the application an opportunity to update the window and perhaps to send an immediate response. In particular, in the case of character-mode remote login, a delayed ACK can reduce the number of segments sent by the server by a factor of 3 (ACK, window update, and echo character all combined in one segment).
- In summary, normally an ACK is sent for every other TCP segment received on a connection, unless the delayed ACK timer (200 milliseconds) expires. The delayed ACK timer can be adjusted through the TcpDelAckTicks registry parameter, which is new in Windows XP.

TCP TIME-WAIT Delay

- When a TCP connection is closed, the socket-pair is placed into a state known as TIME-WAIT so that a new connection will not use the same protocol, source IP address, destination IP address, source port, and destination port until enough time has passed to ensure that any segments that may have been misrouted or delayed will not be delivered unexpectedly.
- The length of time that the socket-pair should not be re-used is specified by RFC 793 as "2MSL" (two maximum segment lifetimes) or 4 minutes. This is the default setting for Windows NT and Windows XP. However, with this default setting, some network applications that perform many outbound connections in a short time may use up all available ports before the ports can be recycled.

TCP Push Bit Interpretation

- By default, Windows XP TCP/IP completes a `recv()` call when one of the following conditions is met:
 - Data arrives with the PUSH bit set.
 - The user `recv()` buffer is full.
 - 0.5 seconds have elapsed since any data has arrived.
 - If a client application is run on a computer with a TCP/IP implementation that does not set the push bit on send operations, response delays may result. It's best to correct this on the client, however a configuration parameter (`IgnorePushBitOnReceives`) was added to `Afd.sys` to force it to treat all arriving packets as though the push bit were set. This parameter was new in Windows NT 4.0 and is also supported in Windows XP.

TCP Forced ACK

By default, Microsoft Windows TCP/IP will acknowledge every second packet.

No.	Source	Destination	Protocol	Info
777	VAIO	CR584842-A	NBSS	NBSS Continuation Message
778	VAIO	CR584842-A	NBSS	NBSS Continuation Message
779	CR584842-A	VAIO	TCP	1190 > nbssession [ACK] Seq=4655837 Ack=486327956
780	CR584842-A	VAIO	SMB	SMBreadBraw Request
781	VAIO	CR584842-A	NBSS	NBSS Continuation Message
782	VAIO	CR584842-A	NBSS	NBSS Continuation Message
783	CR584842-A	VAIO	TCP	1190 > nbssession [ACK] Seq=4655892 Ack=486330468
784	CR584842-A	VAIO	SMB	SMBreadBraw Request
785	VAIO	CR584842-A	NBSS	NBSS Continuation Message
786	VAIO	CR584842-A	NBSS	NBSS Continuation Message
787	CR584842-A	VAIO	TCP	1190 > nbssession [ACK] Seq=4655947 Ack=486333388
788	VAIO	CR584842-A	NBSS	NBSS Continuation Message
789	VAIO	CR584842-A	NBSS	NBSS Continuation Message
790	CR584842-A	VAIO	TCP	1190 > nbssession [ACK] Seq=4655947 Ack=486336308
791	VAIO	CR584842-A	NBSS	NBSS Continuation Message
792	VAIO	CR584842-A	NBSS	NBSS Continuation Message
793	CR584842-A	VAIO	TCP	1190 > nbssession [ACK] Seq=4655947 Ack=486338664

How to contact us at gearbit

Ray Tompkins

sales@gearbit.com

www.gearbit.com

