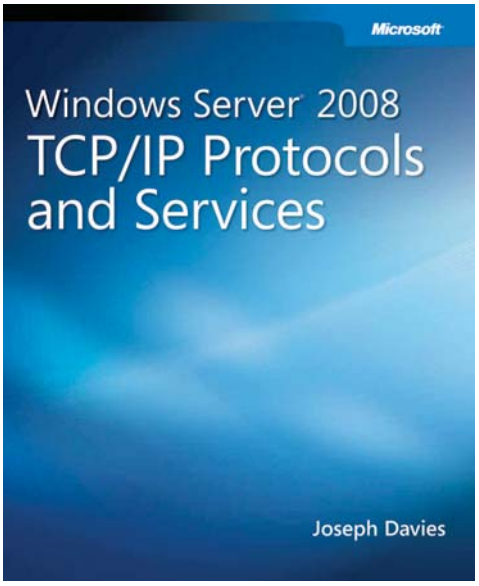

Windows Server® 2008 TCP/IP Protocols and Services

Joseph Davies

To learn more about this book, visit Microsoft Learning at
http://www.microsoft.com/MSPress/books/11630.aspx

Microsoft® Press

978-0-7356-2447-4

Table of Contents

Chapter 10

Transmission Control Protocol (TCP) Basics

There are two protocols at the Transport Layer that TCP/IP applications typically use for transporting data: Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). This chapter describes the characteristics of TCP and the fields in the TCP header.

Introduction to TCP

TCP, defined in RFC 793, is the Transport Layer protocol that provides a reliable data-transfer service and a method to pass TCP-encapsulated data to an Application Layer protocol. TCP has the following characteristics:

Connection-oriented

- Before data can be transferred, two Application Layer processes must formally negotiate a TCP connection using the TCP connection establishment process. TCP connections are formally closed using the TCP connection termination process. For more information about TCP connection processes, see Chapter 11, "Transmission Control Protocol (TCP) Connections."

Full duplex

- For each TCP peer, the TCP connection consists of two logical pipes: an outgoing pipe and an incoming pipe. With the appropriate Network Interface Layer technology, data can be flowing out of the outgoing pipe and into the incoming pipe simultaneously. The TCP header contains both the sequence number of the outgoing data and an acknowledgment of the incoming data.

Reliable

- Data sent on a TCP connection is sequenced and a positive acknowledgment is expected from the receiver. If no acknowledgment is received, the segment is retransmitted. At the receiver, duplicate segments are discarded and segments arriving out of sequence are placed back in the proper sequence. A TCP checksum is always used to verify the bit-level integrity of the TCP segment.

Byte stream

- TCP views the data sent over the incoming and outgoing logical pipes as a continuous stream of bytes. The sequence number and acknowledgment number in each TCP header are defined along byte boundaries. TCP is not aware of record or message boundaries within the byte stream. The Application Layer protocol must provide the proper parsing of the incoming byte stream.

Sender- and receiver-side flow control

- To avoid sending too much data at one time and congesting the routers of the network, TCP implements sender-side flow control that gradually scales the amount of data sent at one time. To avoid having the sender send data that the receiver cannot buffer, TCP implements receiver-side flow control that indicates the number of bytes that the receiver can receive. For more information on how TCP implements sender- and receiver-side flow control, see Chapter 12, "Transmission Control Protocol (TCP) Data Flow."

Segmentation of Application Layer data

- TCP segments data obtained from the Application Layer process so that it will fit within an IP datagram sent on the Network Interface Layer link. TCP peers inform each other of the maximum-sized segment that they can receive and adjust the maximum size using Path Maximum Transmission Unit (PMTU) discovery.

One-to-one delivery

- TCP connections are a logical point-to-point circuit between two Application Layer protocols. TCP does not provide a one-to-many delivery service.

TCP is typically used when an Application Layer protocol requires a reliable data transfer service.

> **MoreInfo** All of the RFCs referenced in this chapter can be found in the \Standards\Chap10_TCP folder on the companion CD-ROM.

The TCP Segment

A TCP segment, consisting of a TCP header and its optional payload (a segment), is identified in the IP header with IP Protocol number 6. The segment can be a maximum size of 65,495 bytes: 65,535 minus the minimum-size IP header (20 bytes) and the minimum-size TCP header (20 bytes). The resulting IP datagram is then encapsulated with the appropriate Network Interface Layer header and trailer. Figure 10-1 displays the resulting frame.

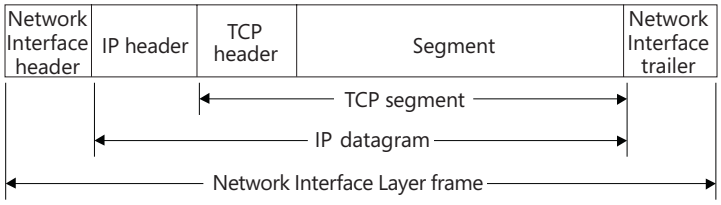

Figure 10-1 TCP segment encapsulation showing the IP header and Network Interface Layer header and trailer.

In the IP header of TCP segments, the Source IP Address field indicates the unicast address of the host interface that sent the TCP segment. The Destination IP Address field indicates the unicast address of the destination host (or intermediate router if the packet is source routed).

The TCP Header

The TCP header is of variable length, consisting of the fields shown in Figure 10-2. When TCP options are not present, the TCP header is 20 bytes long.

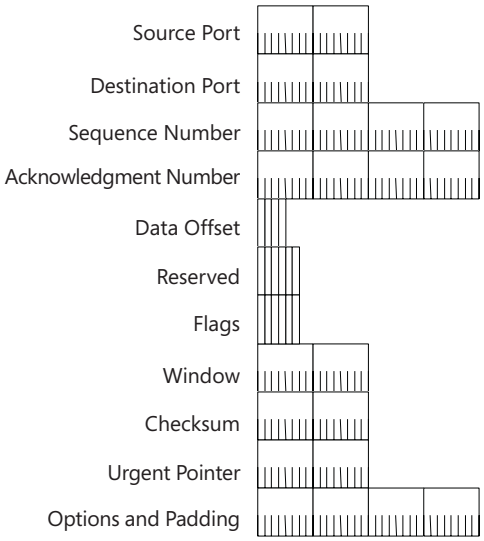

Figure 10-2 The structure of the TCP header.

The fields in the TCP header are defined as follows:

Source Port

- A 2-byte field that indicates the source Application Layer protocol sending the TCP segment. The combination of the source IP address in the IP header and the source port in the TCP header indicates a *source socket*—a unique, globally significant address from which the segment was sent.

Destination Port

- A 2-byte field that indicates the destination Application Layer protocol. The combination of the destination IP address in the IP header and the destination port in the TCP header indicates a *destination socket*—a unique, globally significant address to which the segment is sent.

Sequence Number

- A 4-byte field that indicates the outgoing byte-stream-based sequence number of the segment's first byte. The Sequence Number field is always set, even when there is no data in the segment. In this case, the Sequence Number field is set to the number of the outgoing byte stream's next byte. When establishing a TCP connection, TCP segments with a SYN (Synchronization) flag value of 1 set the Sequence Number field to the Initial Sequence Number (ISN). This indicates that the first byte in the outgoing byte stream sent on the connection is ISN + 1.

Acknowledgment Number

- A 4-byte field that indicates the sequence number of the next byte in the incoming byte stream that the receiver of the incoming byte stream expects to receive. The acknowledgment number provides a positive acknowledgment that all bytes in the incoming byte stream up to, but not including, the acknowledgment number were received. The acknowledgment number is significant in all TCP segments with the ACK (Acknowledgment) flag set.

Data Offset

- A 4-bit field that indicates where the TCP segment data begins. The Data Offset field is also the TCP header's size. Just as in the IP header's Header Length field, the Data Offset field is the number of 32-bit words (4-byte blocks) in the TCP header. For the smallest TCP header (no options), the Data Offset field is set to 5 (0x5), indicating that the segment data begins in the twentieth byte offset starting from the beginning of the TCP segment (the offset starts its count at 0). With a Data Offset field set to its maximum value of 15 (0xF), the largest TCP header, including TCP options, can be 60 bytes long.

Reserved

- A 4-bit field that is reserved for future use. The sender sets these bits to 0.

Flags

- An 8-bit field that indicates the eight TCP flags defined in RFCs 793 and 3168. The eight TCP flags, known as CWR (Congestion Window Reduced), ECE (Explicit Congestion Notification [ECN]-Echo), URG (Urgent), ACK, PSH (Push), RST (Reset), SYN, and FIN (Finish), are discussed in greater detail in the "TCP Flags" section of this chapter.

Window

- A 2-byte field that indicates the number of bytes that the receiver of the incoming byte stream allows the other TCP peer to send. By advertising the window size with each segment, a TCP receiver is telling the sender how much data can be sent and successfully received and stored. The sender should not be sending more data than the receiver can receive. If the receiver cannot receive any more data, it advertises a window size of 0 bytes. With a window size of 0, the sender cannot send any more data until the window size is a nonzero value. The advertisement of the window size is an implementation of receiver-side flow control. The use of this field is extended to larger window sizes with the TCP Window Scale option, discussed in the "TCP Options" section of this chapter.

Checksum

- A 2-byte field that provides a bit-level integrity check for the TCP segment (TCP header and segment). The Checksum field's value is calculated in the same way as the IP header checksum, over all the 16-bit words in a TCP pseudo header, the TCP header, the segment, and, if needed, a padding byte of 0x00. The padding byte is used only if the segment length is an odd number of bytes. The value of the Checksum field is set to 0 during the checksum calculation. For more information, see "The TCP Pseudo Header" section in this chapter.

Urgent Pointer

- A 2-byte field that indicates the location of urgent data in the segment. The Urgent Pointer field and urgent data are discussed in the "TCP Urgent Data" section of this chapter.

Options

- One or more TCP options can be added to the TCP header but must be done in 4-byte increments so that the TCP header size can be indicated with the Data Offset field. TCP options are discussed in the "TCP Options" section of this chapter.

An example of a TCP segment is Capture 10-01, a Network Monitor trace that is included in the \Captures folder on the companion CD-ROM. The following is frame 1 from Capture 10-01, as displayed with Network Monitor 3.1:

```
  Frame:
+ Ethernet: Etype = Internet IP (IPv4)
+ Ipv4: Next Protocol = TCP, Packet ID = 57288, Total IP Length = 1500
- Tcp: Flags=....A..., SrcPort=FTP data(20), DstPort=1163, Len=1460,
Seq=1038577021 - 1038578481, Ack=3930983524, Win=17520 (scale factor not
found)
    SrcPort: FTP data(20)
    DstPort: 1163
    SequenceNumber: 1038577021 (0x3DE76D7D)
    AcknowledgementNumber: 3930983524 (0xEA4E0C64)
  - DataOffset: 80 (0x50)
     DataOffset: (0101....) (20 bytes)
     Reserved:   (....000.)
     NS:         (.......0) Nonce Sum not significant
  - Flags: ....A...
     CWR:    (0.......) CWR not significant
     ECE:    (.0......) ECN-Echo not significant
     Urgent: (..0.....) Not Urgent Data
     Ack:    (...1....) Acknowledgement field significant
     Push:   (....0...) No Push Function
     Reset:  (.....0..) No Reset
     Syn:    (......0.) Not Synchronize sequence numbers
     Fin:    (.......0) Not End of data
    Window: 17520 (scale factor not found)
    Checksum: 46217 (0xB489)
```

```
    UrgentPointer: 0 (0x0)
    TCPPayload:
+ Ftp: Data Transfer To Client,DstPort = 1163,size = 1460 bytes
```

> **Note** Network Monitor 3.1 parses the last bit of the Reserved field of the TCP header as the Nonce Sum field, which is defined in RFC 3540. TCP/IP in Windows Server 2008 and Windows Vista does not support RFC 3540.

TCP Ports

A TCP port defines a location for the delivery of TCP connection data. Included in each TCP segment is the source port that indicates the Application Layer process from which the segment was sent, and a destination port that indicates the Application Layer process to which the segment was sent. There are port numbers that are assigned by the Internet Assigned Numbers Authority (IANA) to specific Application Layer protocols.

Table 10-1 shows assigned TCP port numbers used by components of Windows Server 2008 and Windows Vista.

Table 10-1. Well-Known TCP Port Numbers

Port Number	Application Layer Protocol
20	FTP Server (data channel)
21	FTP Server (control channel)
23	Telnet Server
25	Simple Mail Transfer Protocol (SMTP)
69	Trivial File Transfer Protocol (TFTP)
80	Hypertext Transfer Protocol (HTTP; Web server)
139	NetBIOS Session Service
443	HTTP protocol over Transport Layer Security (TLS)
445	Direct-Hosted Server Message Block (SMB)

See *http://www.iana.org/assignments/port-numbers* for the most current list of IANA-assigned TCP port numbers.

Typically, the server side of an Application Layer protocol listens on the well-known port number. The client side of an Application Layer protocol uses either the well-known port number or, more commonly, a dynamically allocated port number. These dynamically allocated port numbers are used for the duration of the process and are known also as *ephemeral* or *short-lived ports*.

A Windows Sockets application using the *GetServByName()* function can refer to a TCP port number by name. The name is resolved to a TCP port number through the Services file stored in the %*SystemRoot*%\System32\Drivers\Etc folder.

A sending node determines the destination port (using either a specified value or the *GetServByName()* function) and the source port (using either a specified value, or by obtaining a dynamically allocated port through Windows Sockets). The sending node then

passes the source IP address, destination IP address, source port, destination port, and the data to be sent to TCP/IP. The TCP component segments the data as needed. The TCP component calculates the Checksum field and indicates the TCP segment with the appropriate source IP address and destination IP address to the IP component.

When receiving a TCP segment at the destination, IP verifies the IP header. Then, based on the value of 6 in the Protocol field, IP passes the TCP segment, the source IP address, and the destination IP address to the TCP component. After verifying the TCP Checksum field, the TCP component verifies the destination port. If a process is listening on the port, the TCP segment is passed to the application. If no process is listening on the port, TCP sends a TCP Connection Reset segment to the sender. For more information about the TCP Connection Reset segment, see Chapter 11, "Transmission Control Protocol (TCP) Connections."

Figure 10-3 shows the demultiplexing of received TCP connection data based on the TCP destination port.

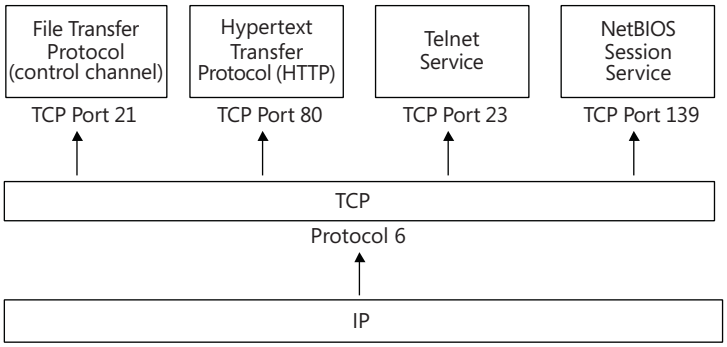

Figure 10-3 The demultiplexing of a TCP segment to the appropriate Application Layer protocol using the IP Protocol field and the TCP Destination Port field.

BestPractices TCP ports are separate from UDP ports, even for the same port number. A TCP port represents one side of a TCP connection for an Application Layer protocol. A UDP port represents a UDP message queue for an Application Layer protocol. The Application Layer protocol using the TCP port is not necessarily the same Application Layer protocol using the UDP port. For example, the Extended Filename Server (EFS) protocol uses TCP port 520, and the Routing Information Protocol (RIP) uses UDP port 520. Clearly these are separate Application Layer protocols. Therefore, it is not good practice to refer to a port by just its port number, which is ambiguous. Always refer to either a TCP port number or a UDP port number.

TCP Flags

Figure 10-4 shows the eight TCP flags in the Flags field of the TCP header that are defined in RFCs 793 and 3168.

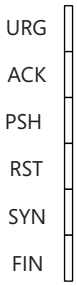

Figure 10-4 The eight TCP flags in the Flags field of the TCP header.

The TCP flags are defined as follows:

CWR (congestion window has been reduced)

- Indicates that the sending host has received a TCP segment with the ECE flag set. The congestion window is an internal variable maintained by TCP to manage the size of the send window. For more information, see Chapter 12, "Transmission Control Protocol (TCP) Data Flow."

ECE (TCP peer is ECN-capable)

- Indicates that a TCP peer is ECN-capable during the TCP 3-way handshake and to indicate that a TCP segment was received on the connection with the ECN field in the IP header set to 11. For more information about ECN, see Chapter 12, "Transmission Control Protocol (TCP) Data Flow."

URG (Urgent Pointer field is significant)

- Indicates that the segment portion of the TCP segment contains urgent data and the Urgent Pointer field should be used to determine the location of the urgent data in the segment. Urgent data is discussed in more detail in the section "TCP Urgent Data," later in this chapter.

ACK (Acknowledgment field is significant)

- Indicates that the Acknowledgment field contains the next byte expected on the connection. The ACK flag is always set, except for the first segment of a TCP connection establishment.

PSH (the Push function)

- Indicates that the contents of the TCP receive buffer should be passed to the Application Layer protocol. The data in the receive buffer must consist of a contiguous block of data from the left edge of the buffer. In other words, there cannot be any missing segments of the byte stream up to the segment containing the PSH flag; the data cannot be passed to the Application Layer protocol until missing segments arrive. Normally, the TCP receive buffer is flushed (the contents are passed to the Application Layer protocol) when the receive buffer fills with contiguous data or during normal TCP connection maintenance processes. The PSH flag overrides this default behavior and immediately flushes the TCP receive buffer. The PSH flag is used also for interactive Application Layer protocols such as Telnet, in which each keystroke in the virtual terminal session is sent with the PSH flag set. Another example is the setting of the PSH flag on the last segment of a file transferred with FTP. Data sent with the PSH flag does not have to be immediately acknowledged.

RST (Reset the connection)

- Indicates that the connection is being aborted. For active connections, a node sends a TCP segment with the RST flag in response to a TCP segment received on the connection that is incorrect, causing the connection to fail. The sending of an RST segment for an active connection forcibly terminates the connection, causing data stored in send and receive buffers or in transit to be lost. For TCP connections being established, a node sends an RST segment in response to a connection establishment request to deny the connection attempt.

SYN (Synchronize sequence number)

- Indicates that the segment contains an ISN. During the TCP connection establishment process, TCP sends a TCP segment with the SYN flag set. Each TCP peer acknowledges the receipt of the SYN flag by treating the SYN flag as if it were a single byte of data. The Acknowledgment Number field for the acknowledgment of the SYN segment is set to ISN + 1.

FIN (Finish sending data)

- Indicates that the TCP segment sender is finished sending data on the connection. When a TCP connection is gracefully terminated, each TCP peer sends a TCP segment with the FIN flag set. A TCP peer does not send a TCP segment with the FIN flag set until all outstanding data to the other TCP peer has been sent and acknowledged. Each peer acknowledges receipt of the FIN flag by treating it as if it were a single byte of data. When both TCP peers have sent segments with the FIN flag set and received acknowledgment of their receipt, the TCP connection is terminated.

The TCP Pseudo Header

The TCP pseudo header is used to associate the TCP segment with the IP header. The TCP pseudo header is added to the beginning of the TCP segment only during the checksum calculation and is not sent as part of the TCP segment. The use of the TCP pseudo header assures the receiver that a routing or fragmentation process did not improperly modify key fields in the IP header.

Figure 10-5 illustrates the TCP pseudo header.

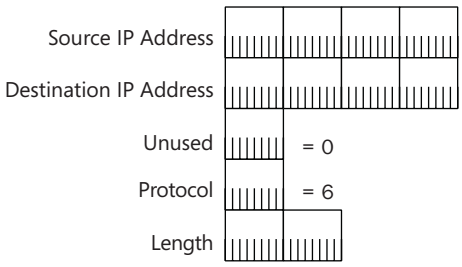

Figure 10-5 The structure of the TCP pseudo header.

The TCP pseudo header consists of the Source IP Address field, the Destination IP Address field, an Unused field set to 0x00, the Protocol field for TCP (set to 6), and the length of the TCP segment. When sending a TCP segment, TCP knows all of these values. When receiving a TCP segment, IP indicates all of these values to TCP.

TCP calculates the TCP checksum over the combination of the TCP pseudo header, the TCP segment, and, if needed, a 0x00 padding byte. The checksum calculation relies on summing 16-bit words. Therefore, the quantity over which the checksum is calculated must be an even number of bytes. The padding byte is used only if the segment length is an odd number of bytes. The padding byte is not included in the IP length and is not sent as part of the TCP segment. Figure 10-6 shows the resulting quantity for the calculation of the TCP Checksum field.

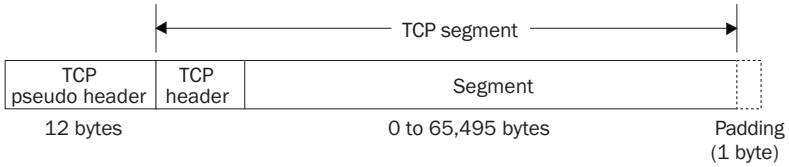

Figure 10-6 The resulting quantity used for the TCP checksum calculation.

Note Unlike the IP security (IPsec) Authentication header, the TCP pseudo header and Checksum field are not providing data authentication or data integrity for the fields in the IP header and the TCP segment. IP header and TCP port number fields can be modified as long as the TCP checksum is updated. This is how a Network Address Translator (NAT) works. A NAT is a router that translates public and private addresses during the forwarding process. For example, when translating a source IP address from a private address to a public address, the NAT also recalculates the TCP Checksum field.

TCP Urgent Data

Normal data sent on a TCP connection is data corresponding to the incoming and outgoing byte stream data. In some data-transfer situations, there must be a method of sending control data to interrupt a process or inform the Application Layer protocol of asynchronous events. This control data is known as *out of band data*—data that is not part of the TCP byte stream but is needed to control the data flow. Out of band data for TCP connections can be implemented in the following ways:

- **Use a separate TCP connection for the out of band data.** The separate TCP connection sends control commands and status information without being combined on the data stream of the data connection. This is the method used by FTP. FTP uses a TCP connection on port 21 for control commands such as logins, gets (downloading files to the FTP client), and puts (uploading files to the FTP server), and a separate TCP connection on port 20 for the sending or receiving of file data.
- **Use TCP urgent data.** TCP urgent data is sent on the same TCP connection as the data. TCP urgent data is indicated by setting the URG flag, and the urgent data is distinguished from the nonurgent data using the Urgent Pointer field. Urgent data within the TCP segment must be processed before the nonurgent data. Urgent data is used by the Telnet protocol to send control commands, even though the advertised receive window of the Telnet server is 0.

The interpretation of the Urgent Pointer value depends on the TCP implementation's adherence to either RFC 793, the original TCP RFC, or RFC 1122, which defines

requirements for Internet hosts. The difference between the two interpretations is the following:

- RFC 793 defines the value of the Urgent Pointer field as the positive offset from the beginning of the TCP segment to the first byte of nonurgent data.
- RFC 1122 defines the value of the Urgent Pointer field as the positive offset from the beginning of the TCP segment to the last byte of urgent data.

These two definitions of the Urgent Pointer field differ by one byte. Both hosts on a TCP connection must use the same interpretation, otherwise data corruption could occur. There is no interoperability of these two interpretations, nor is there a mechanism to negotiate the interpretation during the TCP connection establishment process.

The definition of the Urgent Pointer field in RFC 793 was made in error (the correct interpretation is actually given later in the RFC during the discussion of event processing in Section 3.9). The correct use of the Urgent Pointer field is the RFC 1122 version, but numerous implementations of TCP use the RFC 793 definition.

Figure 10-7 shows the placement of urgent data within the TCP segment and the RFC 793 and RFC 1122 interpretation of the Urgent Pointer field.

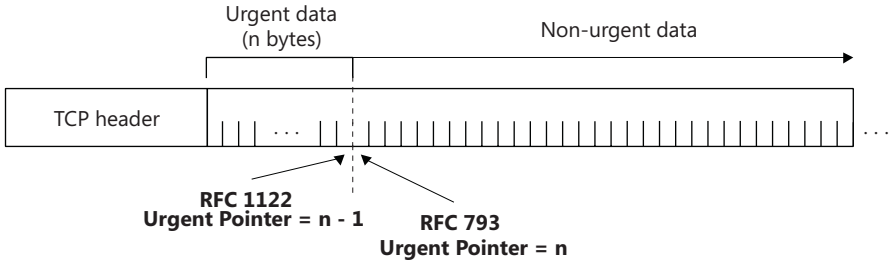

Figure 10-7 The location of TCP urgent data within a TCP segment.

To configure the interpretation of the TCP Urgent Pointer field for TCP in Windows Server 2008 and Windows Vista, use the following registry value:

TcpUseRFC1122UrgentPointer

```
Key: HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters
Value type: REG_DWORD
Valid range: 0-1
Default: 0
Present by default: No
```

Set this registry value to 1 to use the RFC 1122 interpretation of the Urgent Pointer field or to 0 to use the RFC 793 interpretation (the default).

TCP Options

Just like options in the IP header extend IP functionality, TCP options extend TCP functionality. There are a variety of defined TCP options that are used for negotiating maximum segment sizes, window scaling factors, performing selective acknowledgments, recording timestamps, and providing padding for 4-byte boundaries. A node is not required to support all TCP options; however, the support for processing TCP options is

required. The presence of TCP options is indicated by a Data Offset field with a value greater than 5 (0x5) (the TCP header is longer than 20 bytes).

A TCP option is either a single byte or multiple bytes. For multiple-byte options, the TCP option is in type-length-value format, where the length is the length in bytes of the entire option. Figure 10-8 shows the structure of multiple-byte TCP options. A TCP option type is known as an *option kind*.

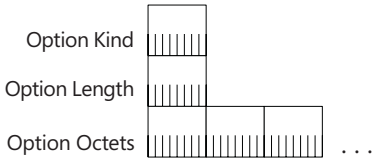

Figure 10-8 The structure of multiple-byte TCP options.

End Of Option List and No Operation

To implement 4-byte boundary support for TCP options, RFC 793 defines the following single-byte TCP options:

The End Of Option List

- The option kind set to 0 (0x00), which indicates that no other options follow. The End Of Option List option is not used to delimit TCP options. If the set of TCP options falls along a 4-byte boundary, this option is not needed.

The No Operation

- The option kind set to 1 (0x01), which is used between TCP options for 4-byte alignment. The No Operation option is not required, so TCP implementations must be able to correctly interpret TCP options that are not on 4-byte boundaries.

Maximum Segment Size Option

The TCP maximum segment size (MSS) is the largest segment that can be sent on the connection. To obtain the MSS value, take the IP Maximum Transmission Unit (MTU) and subtract the IP header size and the TCP header size. Figure 10-9 shows the relationship between the IP MTU and the TCP MSS. For a typical IP header (without options) and a typical TCP header (without options), the MSS is 40 bytes less than the IP MTU.

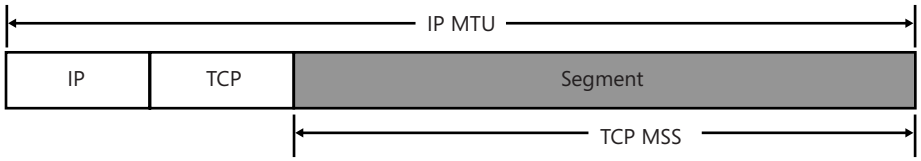

Figure 10-9 The TCP MSS defined in terms of the IP MTU and the TCP and IP header sizes.

A TCP peer uses the TCP MSS option to indicate the MSS that it can receive. The TCP MSS option is included only in TCP segments with the SYN flag set during the TCP connection establishment process. Figure 10-10 shows the TCP MSS option structure.

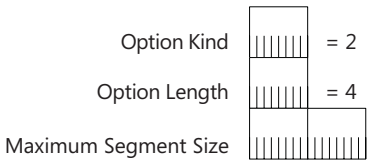

Figure 10-10 The structure of the TCP MSS option.

The fields in the TCP MSS option are defined as follows:

Option Kind

- Set to 2 (0x02) to indicate the MSS option kind.

Option Length

- Set to 4 (0x04) to indicate that the size of the entire MSS option is 4 bytes.

Maximum Segment Size

- Two bytes that indicate the MSS of received segments. For IP datagrams sent on an Ethernet network segment using Ethernet II encapsulation, the MSS is 1460 (an IP MTU of 1500 minus 40 bytes for minimum-sized IP and TCP headers).

An example of the TCP MSS option is Capture 10-02, a Network Monitor trace that is included in the \Captures folder on the companion CD-ROM. The following is the TCP SYN segment from Capture 10-02 (frame 1), as displayed with Network Monitor 3.1:

```
  Frame:
+ Ethernet: Etype = Internet IP (IPv4)
+ Ipv4: Next Protocol = TCP, Packet ID = 10474, Total IP Length = 48
- Tcp: Flags=.S......, SrcPort=1162, DstPort=FTP control(21), Len=0,
Seq=3928116524, Ack=0, Win=16384 (scale factor not found)
    SrcPort: 1162
    DstPort: FTP control(21)
    SequenceNumber: 3928116524 (0xEA224D2C)
    AcknowledgementNumber: 0 (0x0)
  - DataOffset: 112 (0x70)
     DataOffset: (0111....) (28 bytes)
     Reserved:   (....000.)
     NS:         (.......0) Nonce Sum not significant
  - Flags: .S......
     CWR:    (0.......) CWR not significant
     ECE:    (.0......) ECN-Echo not significant
     Urgent: (..0.....) Not Urgent Data
     Ack:    (...0....) Acknowledgement field not significant
     Push:   (....0...) No Push Function
     Reset:  (.....0..) No Reset
```

```
     Syn:     (......1.) Synchronize sequence numbers
     Fin:     (.......0) Not End of data
    Window: 16384 (scale factor not found)
    Checksum: 34126 (0x854E)
    UrgentPointer: 0 (0x0)
  - TCPOptions:
   - MaxSegmentSize:
       type: Maximum Segment Size. 2(0x2)
       OptionLength: 4 (0x4)
       MaxSegmentSize: 1460 (0x5B4)
   + NoOption:
   + NoOption:
   + SACKPermitted:
```

When two TCP peers exchange their MSS during the connection establishment process, both peers adjust their initial MSS to the minimum value reported by both. For example, when an Ethernet node sends an MSS of 1460 and an 802.11 wireless node sends an MSS of 2272 (the 802.11 IP MTU of 2312, minus 40 bytes), both nodes agree to send maximum-sized TCP segments of 1460 bytes. The initial MSS is adjusted on an ongoing basis through PMTU discovery. For example, two 802.11 wireless nodes on two separate network segments—connected by routers over Ethernet network segments—exchange a TCP MSS of 2272. However, the wireless nodes begin sending 2272-byte TCP segments, and PMTU discovery messages adjust the MSS for the connection to 1460. For more information about PMTU, see Chapter 6, "Internet Control Message Protocol (ICMP)."

The TCP MSS option does not prevent problems that could occur between two hosts on the same network segment (subnet) that are separated by a Network Interface Layer technology with a lower IP MTU size. For example, Host A and Host B in Figure 10-11 are 802.11 wireless nodes connected to separate wireless access points (APs) that are connected by an Ethernet backbone.

Both wireless APs and their connected wireless clients and the Ethernet backbone are on the same network segment as the router. Therefore, when Hosts A and B exchange their MSSs, both agree to send maximum-sized TCP segments with a size of 2272 bytes. However, when they begin to send bulk data with maximum-sized segments, the wireless APs, acting as Layer 2 translating bridges, have no facilities for translating 2272-byte 802.11 payloads to 1500-byte Ethernet payloads. Therefore, the wireless APs silently discard the maximum-sized TCP segments. The wireless AP is not an IP router and does not send PMTU discovery messages to the TCP peers to lower their MSS. Maximum-sized TCP segments cannot be sent between the two TCP peers.

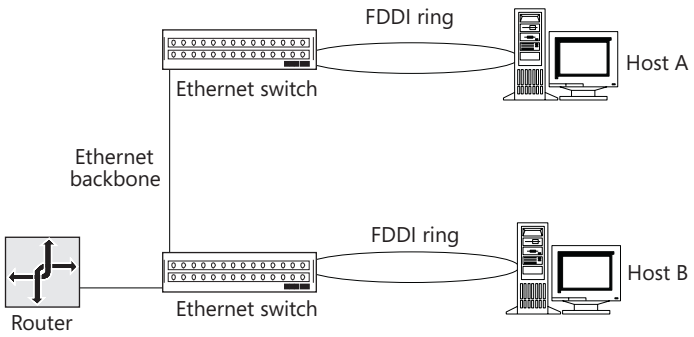

Figure 10-11 Hosts connected to two wireless APs that are connected by an Ethernet backbone.

If Host A were an FTP server and Host B were an FTP client, the user at Host B would be able to connect and log in to the FTP server. However, when the user issued a get or put instruction to send a file, TCP segments at the maximum size would be dropped by the wireless APs.

The only solution to this problem is to adjust the IP MTU on the wireless nodes to the lowest value supported by all the Network Interface Layer technologies on the network segment. For example, you could use the **netsh interface ipv4 set interface mtu** command or the MTU registry value described in Chapter 5, "Internet Protocol (IP)," to lower the IP MTU of the two wireless adapters to 1500.

TCP Window Scale Option

The TCP window size defined in RFC 793 is a 16-bit field for a maximum receive window size of 65,535 bytes. This means that a sender can have only 65,535 bytes of data in transit before having to wait for an acknowledgment. This is not an issue on typical local area network (LAN) and wide area network (WAN) links, but it is possible on newer LAN and WAN technologies operating at gigabit-per-second speeds with a sizable transit delay to have more than 65,535 bytes in transit. If TCP cannot fill the logical pipe between the sender and receiver and keep it filled, it is operating at lower efficiency.

The TCP Window Scale option described in RFC 1323 allows the receiver to advertise a larger window size than 65,535 bytes. The Window Scale option includes a window scaling factor that, when exponentially combined with the 16-bit window size in the TCP header, increases the receive window size to a maximum of 1,073,741,824 bytes, or 1 gigabyte (GB). The Window Size option is sent only in a SYN segment during the connection establishment process. TCP peers can indicate different window scaling factors used for their receive window sizes. The receiver of the TCP connection establishment request (the SYN segment) cannot send a Window Scale option unless the initial SYN segment contains it.

Figure 10-12 illustrates the TCP Window Scale option structure.

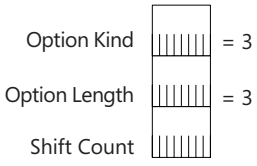

Figure 10-12 The structure of the TCP Window Scale option.

The fields in the TCP Window Scale option are defined as follows:

Option Kind

- Set to 3 (0x03) to indicate the Window Scale option kind.

Option Length

- Set to 3 (0x03) to indicate that the size of the entire TCP option is three bytes.

Shift Count

- One byte that indicates the scaling factor as the exponent of 2. For example, for a Shift Count of 5, the scaling factor is 2^5, or 32. The exponent is used rather than a whole number so that implementations can take advantage of binary shift programming techniques to quickly calculate the actual window size. For example, for a Shift Count of 5, the actual window size is the binary value of the Window field with five zeros added (the Window field is left-shifted by 5). The maximum value of the Shift Count is 14 for a window scaling factor of 2^{14}, or 16,384. Combined with the original window size of 2^{16}, the maximum window size is $2^{16} \times 2^{14} = 2^{30}$, or 1,073,741,824 bytes.

An example of a TCP Window Scale option is Capture 10-03, a Network Monitor trace that is included in the \Captures folder on the companion CD-ROM. The following is the TCP SYN segment from Capture 10-03 (frame 1), as displayed with Network Monitor 3.1:

```
  Frame:
+ Ethernet: Etype = Internet IP (IPv4)
+ Ipv4: Next Protocol = TCP, Packet ID = 594, Total IP Length = 52
- Tcp: Flags=.S......, SrcPort=49786, DstPort=NETBIOS Session
Service(139), Len=0, Seq=2626199192, Ack=0, Win=8192 (scale factor not
found)
    SrcPort: 49786
    DstPort: NETBIOS Session Service(139)
    SequenceNumber: 2626199192 (0x9C889E98)
    AcknowledgementNumber: 0 (0x0)
  + DataOffset: 128 (0x80)
  + Flags: .S......
    Window: 8192 (scale factor not found)
    Checksum: 15591 (0x3CE7)
    UrgentPointer: 0 (0x0)
  - TCPOptions:
   + MaxSegmentSize:
   + NoOption:
   - WindowsScaleFactor:
      type: Window scale factor. 3(0x3)
      Length: 3 (0x3)
```

```
      ShiftCount: 8 (0x8)
   + NoOption:
   + NoOption:
   + SACKPermitted:
```

Notice the use of the TCP No Operation option (NoOption) preceding the Window Scale option to align the Window Scale option on 4-byte boundaries.

When the Window Scale option is used, the window size advertised in each TCP segment for the connection is scaled by the factor indicated in the peer's SYN segment. Therefore, the TCP header's Window field is no longer a byte counter of the amount of space left in the receive buffer. Rather, the Window field is a block counter in which the block size in bytes is the scaling factor. For example, for a TCP peer using a Shift Count of 3, the Window field in outgoing TCP segments is actually indicating the number of 8-byte blocks remaining in the receive buffer.

By default, TCP for Windows Server 2008 and Windows Vista always uses window scaling with a scaling factor of 8, for a 16-megabyte (MB) receive window. To disable window scaling, use the **netsh interface tcp set global autotuninglevel=disabled** command. When window scaling is disabled, TCP uses a window size based on the link speed of the sending interface. For more information about how TCP for Windows Server 2008 and Windows Vista uses the receive window to maximize incoming data, see Chapter 12, "Transmission Control Protocol (TCP) Data Flow."

> **Note** When tracing TCP connection data, make sure that you also look at the connection establishment process to determine whether window scaling is being used. Otherwise, you might misinterpret the Window field value during the connection.

Selective Acknowledgment Option

The acknowledgment scheme for TCP was originally designed as a positive cumulative acknowledgment scheme in which the receiver sends a segment with the ACK flag set and the Acknowledgment field set to the next byte the receiver expects to receive. This use of the Acknowledgment field provides an acknowledgment of all bytes up to, but not including, the sequence number in the Acknowledgment field. This scheme provides reliable byte-stream data transfer, but can result in lower TCP throughput in environments with high packet losses.

If a segment at the beginning of the current send window is not received and all other segments are, the data received cannot be acknowledged until the missing segment arrives. The sender begins to retransmit the segments of the current send window until the acknowledgment for all the segments received has arrived. The sender needlessly retransmits some segments, consequently wasting network bandwidth. This problem is exacerbated in environments such as satellite links, with high bandwidth and high delay, when TCP has a large window size. The more segments in the send window, the more segments can be retransmitted unnecessarily when segments are lost.

RFC 2018 describes a method of selective acknowledgment using TCP options that selectively acknowledges the noncontiguous data blocks that have been received. A

sender that receives a selective acknowledgment can retransmit just the missing blocks, preventing the sender from waiting for the retransmission time-out for the unacknowledged segments and retransmitting segments that have successfully arrived.

The selective acknowledgment scheme defines the following two different TCP options:

- The Selective Acknowledgment (SACK)-Permitted option to negotiate the use of selective acknowledgments during the connection establishment process
- The SACK option to indicate the noncontiguous data blocks that have been received

The SACK-Permitted Option

The SACK-Permitted option is sent in segments with the SYN flag set and indicates that the TCP peer can receive and interpret the TCP SACK option when data is flowing on the connection. The SACK-Permitted option is 2 bytes consisting of an Option Kind set to 4 (0x04) and an Option Length set to 2 (0x02), as shown in Figure 10-13.

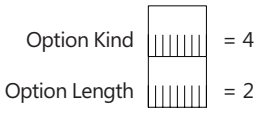

Figure 10-13 The structure of the TCP SACK-Permitted option.

An example of a TCP SACK-Permitted option is Capture 10-04, a Network Monitor trace that is included in the \Captures folder on the companion CD-ROM. The following is the TCP SYN segment from Capture 10-04 (frame 1), as displayed with Network Monitor 3.1:

```
  Frame:
+ Ethernet: Etype = Internet IP (IPv4)
+ Ipv4: Next Protocol = TCP, Packet ID = 10474, Total IP Length = 48
- Tcp: Flags=.S......, SrcPort=1162, DstPort=FTP control(21), Len=0,
Seq=3928116524, Ack=0, Win=16384 (scale factor not found)
    SrcPort: 1162
    DstPort: FTP control(21)
    SequenceNumber: 3928116524 (0xEA224D2C)
    AcknowledgementNumber: 0 (0x0)
  + DataOffset: 112 (0x70)
  + Flags: .S......
    Window: 16384 (scale factor not found)
    Checksum: 34126 (0x854E)
    UrgentPointer: 0 (0x0)
  - TCPOptions:
   + MaxSegmentSize:
   + NoOption:
   + NoOption:
```

```
- SACKPermitted:
    type: SACK permitted. 4(0x4)
    OptionLength: 2 (0x2)
```

Notice the use of the two TCP No Operation option (NoOption) fields preceding the SACK-Permitted option to align the SACK-Permitted option on 4-byte boundaries.

The SACK Option

The SACK option is sent as needed in segments of the open connection with the ACK flag set. As Figure 10-14 shows, the SACK option is a variable-size option, depending on how many contiguous blocks are being acknowledged.

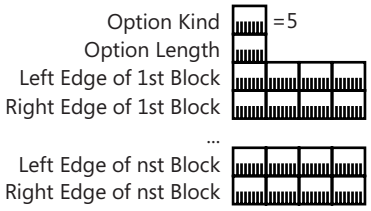

Figure 10-14 The structure of the TCP SACK option.

The fields in the TCP SACK option are defined as follows:

Option Kind

- Set to 5 (0x05) to indicate the SACK option kind.

Option Length

- Set to 10 (a single noncontiguous block), 18 (two noncontiguous blocks), 26 (three noncontiguous blocks), or 34 (four noncontiguous blocks) bytes to indicate the size of the entire TCP option.

Left Edge of Nth Block

- A 4-byte field that indicates the sequence number of this block's first byte.

Right Edge of Nth Block

- A 4-byte field that indicates the next sequence number expected to be received immediately following this block.

An example of a TCP SACK option is Capture 10-05, a Network Monitor trace that is included in the \Captures folder on the companion CD-ROM. The following is the TCP segment from Capture 10-05 (frame 1), as displayed with Network Monitor 3.1:

```
  Frame:
+ Ethernet: Etype = Internet IP (IPv4)
+ Ipv4: Next Protocol = TCP, Packet ID = 64013, Total IP Length = 64
- Tcp: Flags=....A..., SrcPort=1242, DstPort=NETBIOS Session
Service(139), Len=0, Seq=925293, Ack=55053434, Win=32767 (scale factor
not found)
    SrcPort: 1242
    DstPort: NETBIOS Session Service(139)
```

```
    SequenceNumber: 925293 (0xE1E6D)
    AcknowledgementNumber: 55053434 (0x3480C7A)
  + DataOffset: 176 (0xB0)
  + Flags: ....A...
    Window: 32767 (scale factor not found)
    Checksum: 17262 (0x436E)
    UrgentPointer: 0 (0x0)
  - TCPOptions:
   + NoOption:
   + NoOption:
   + TimeStamp:
   + NoOption:
   + NoOption:
   - SACK:
      type: SACK. 5(0x5)
      Length: 10 (0xA)
    - Blocks:
       LeftEdge: 55054882 (0x3481222)
       RightEdge: 55059226 (0x348231A)
```

In the trace, the sender of this segment is acknowledging the receipt of all contiguous bytes in the byte stream up to, but not including, byte 55053434, and the receipt of the block of contiguous data from bytes 55054882 through 55059225. There is a missing segment consisting of the bytes 55053434 through 55054881. Notice the use of the Nop options (NoOption) to align the SACK option on 4-byte boundaries.

TCP in Windows Server 2008 and Windows Vista always uses selective acknowledgments and the SACK options.

For more information on the use of selective acknowledgments to retransmit data, see Chapter 13, "Transmission Control Protocol (TCP) Retransmission and Time-Out."

> **Note** TCP in Windows Server 2008 and Windows Vista no longer supports the SackOpts registry value.

TCP Timestamps Option

To set the retransmission time-out (RTO) on TCP segments sent, TCP monitors the round-trip time (RTT) on an ongoing basis. Normally, TCP calculates the RTT of a TCP segment and its acknowledgment once for every full send window of data. Although this works well in many environments, for high-bandwidth and high-delay environments such as satellite links with large window sizes, the sampling rate of one segment for each window size

cannot monitor the RTT to determine the current RTO and prevent unnecessary retransmissions.

To calculate the RTT on any TCP segment, the segment is sent with the TCP Timestamps option described in RFC 1323. This option places a timestamp value based on a local clock on an outgoing TCP segment. The acknowledgment for the data in the TCP segment echoes back the timestamp, and the RTT can be calculated from the segment's echoed timestamp and the time (relative to the local clock) that the segment's acknowledgment arrived.

Including the Timestamps option in the SYN segment during the connection establishment process indicates its use for the connection. Both sides of the TCP connection can selectively use timestamps. Once indicated during connection establishment, the timestamp can be included in TCP segments at the discretion of the sending TCP peer.

Figure 10-15 shows the TCP Timestamps option structure.

F10xx15 (Art Missing)

Figure 10-15. **The structure of the TCP Timestamps option.**

The fields in the TCP Timestamps option are defined as follows:

Option Kind

- Set to 8 (0x08) to indicate the Timestamps option kind.

Option Length

- Set to 10 (0x0A) to indicate that the size of the entire TCP option is 10 bytes.

TS Value

- A 4-byte field that indicates the timestamp value of this TCP segment. The TS Value is calculated from an internal clock that is based on real time. The TS Value increases over time and wraps around when needed.

TS Echo Reply

- A 4-byte field set on a TCP segment that acknowledges data received (with the ACK flag set) that is set to the same value as the TS Value for the received segment being acknowledged. In other words, the TS Echo Reply is an echo of the TS Value of the acknowledged segment.

Figure 10-16 illustrates an example of the values of the TS Value and TS Echo Reply for an exchange of data between two hosts.

F10xx16 (Art Missing)

Figure 10-16. An example of the use of the TCP Timestamps option.

Host A's internal clock starts its TS Value at 100. Host B's internal clock starts its TS Value at 9000. Segments 1 through 4 are for two data blocks sent by Host A. Segments 5 and 6 are for a data block sent by Host B. Notice how the TS Echo Reply value for the acknowledgments is set to the TS Value of the segments they are acknowledging. To prevent gaps in the sending of data from increasing the RTT, the TS Echo Reply is used for RTT measurement only if the segment is an acknowledgment of new data sent.

An example of the use of the TCP Timestamps option is Capture 10-06, a Network Monitor trace that is included in the \Captures folder on the companion CD-ROM. The following is frame 1 containing the TCP Timestamps option and frame 2 containing the corresponding acknowledgment, as displayed with Network Monitor 3.1:

```
  Frame:
+ Ethernet: Etype = Internet IP (IPv4)
+ Ipv4: Next Protocol = TCP, Packet ID = 6677, Total IP Length = 1500
- Tcp: Flags=....A..., SrcPort=NETBIOS Session Service(139),
DstPort=1242, Len=1448, Seq=55050538 - 55051986, Ack=925293, Win=16564
(scale factor not found)
    SrcPort: NETBIOS Session Service(139)
    DstPort: 1242
    SequenceNumber: 55050538 (0x348012A)
    AcknowledgementNumber: 925293 (0xE1E6D)
  + DataOffset: 128 (0x80)
  + Flags: ....A...
    Window: 16564 (scale factor not found)
    Checksum: 48513 (0xBD81)
    UrgentPointer: 0 (0x0)
  - TCPOptions:
   + NoOption:
   + NoOption:
   - TimeStamp:
      type: Timestamp. 8(0x8)
      Length: 10 (0xA)
      TimestampValue: 4677 (0x1245)
      TimestampEchoReply: 7114 (0x1BCA)
    TCPPayload:
+ Nbtss: NbtSS Continue payload, Length = 1448
```

```
  Frame:
+ Ethernet: Etype = Internet IP (IPv4)
+ Ipv4: Next Protocol = TCP, Packet ID = 62989, Total IP Length = 52
- Tcp: Flags=....A..., SrcPort=1242, DstPort=NETBIOS Session
Service(139), Len=0, Seq=925293, Ack=55051986, Win=32722 (scale factor
not found)
```

```
    SrcPort: 1242
    DstPort: NETBIOS Session Service(139)
    SequenceNumber: 925293 (0xE1E6D)
    AcknowledgementNumber: 55051986 (0x34806D2)
  + DataOffset: 128 (0x80)
  + Flags: ....A...
    Window: 32722 (scale factor not found)
    Checksum: 47929 (0xBB39)
    UrgentPointer: 0 (0x0)
  - TCPOptions:
   + NoOption:
   + NoOption:
   - TimeStamp:
      type: Timestamp. 8(0x8)
      Length: 10 (0xA)
      TimestampValue: 7126 (0x1BD6)
      TimestampEchoReply: 4677 (0x1245)
```

Notice that in the second frame the TS Echo Reply field (TimestampEchoReply) is set to 4677, echoing the TS Value field (TimestampValue) of the first frame.

In Windows Server 2008 and Windows Vista, the use of TCP timestamps can be controlled by the **netsh interface tcp set global timestamps=disabled|enabled|default** command. By default, TCP timestamps are disabled.

You can also use the following registry value:

Tcp1323Opts

```
Key:HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters
Value type: REG_DWORD
Valid range: 0 or 2
Default value: 0
Present by default: No
```

Set this value to 0 to disable timestamps. Set this value to 2 to enable timestamps. The default behavior of TCP is to not use timestamps. For more information on RTT, RTO, and retransmission behavior, see Chapter 13, "Transmission Control Protocol (TCP) Retransmission and Time-Out."

Summary

TCP provides connection-oriented and reliable data transfer for applications that require end-to-end guaranteed delivery service. Application Layer protocols use TCP for one-to-one traffic. The TCP header provides sequencing, acknowledgment, a checksum, and the identification of source and destination port numbers to multiplex TCP segment data to the proper Application Layer protocol. TCP options are used to indicate maximum segment sizes and window scaling, indicate and provide selective acknowledgments, and provide timestamps for better RTT determination.