Data Center Fundamentals

Understand Data Center network design and infrastructure architecture, including load balancing, SSL, and security

Mauricio Arregoces, CCIE® No. 3285
Maurizio Portolani

ciscopress.com

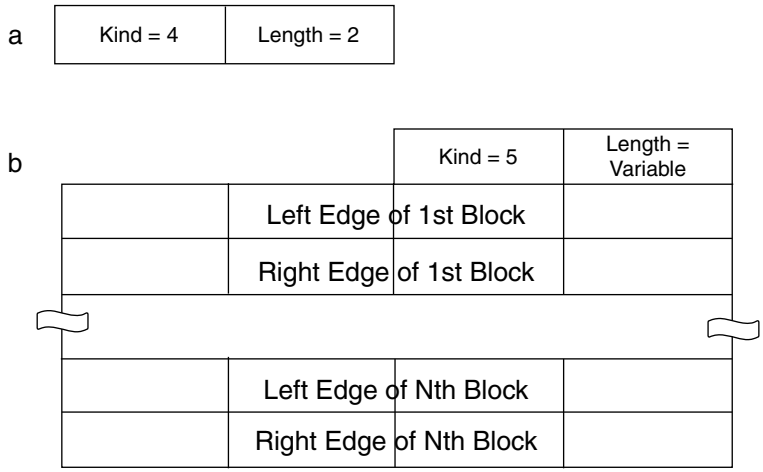
Data Center Fundamentals

Mauricio Arregoces, CCIE No. 3285
Maurizio Portolani

Cisco Press

Cisco Press
800 East 96th Street
Indianapolis, IN 46240 USA

Figure 7-23 *SACK-Permitted and SACK Options*



The left-edge block is the first SEQ of the block, and the right edge is the SEQ that follows the last SEQ of the block. This setup implies that between blocks (to the right of block x and the left of block x+1), there are segments that have not been received.

Given that the option field is up to 40 bytes, a SACK option is able to carry up to 4 blocks (4 blocks * 8 bytes each + 2 bytes for the kind and length). The timestamp option is typically used concurrently with SACK, and it reduces the overall number of blocks to 3. The next section explains the details.

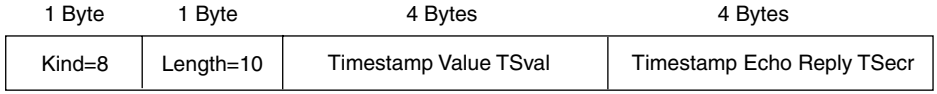
NOTE Microsoft knowledge base article 224829 provides a description of Microsoft Windows 2000 TCP features that include SACK as well as timestamp and window scale (http://support.microsoft.com/default.aspx?scid=kb;en-us;224829). This article contains explanations of the operation of the option and the mechanisms to change the values through the Registry.

In Linux, TCP SACK is enabled by the **tcp_sack** sysctls, which can be accessed by the **/proc/sys/net/ipv4/*** files or with the interface. (See http://www.europe.redhat.com/documentation/man-pages/man7/tcp.7.php3.)

Timestamp Option

The TCP timestamp, a 10-byte option, was also introduced in RFC 1072 (along with SACK) to allow a sender to place a timestamp on every segment. It also allows the receiver to return the timestamp in an acknowledgement so that the receiver can accurately measure the RTT. Figure 7-24 introduces the timestamp option fields.

Figure 7-24 *Timestamp Option Fields*



Using timestamps lets you avoid a number of issues resulting from changing traffic conditions. The most obvious problem concerns the average RTT being used in the calculation for retransmission: if the actual RTT increases, the number of retransmissions increases. TCP would have to continuously calculate an accurate average RTT, which, as referenced in the RFC, would be computationally heavy and under certain error conditions, impossible to calculate. RFC 1323 made RFC 1072 obsolete, clarified some issues, and corrected some ambiguity. (Refer to Appendix C, "Video Encoding Mechanisms" for more details on RFC 1323.) Two important changes are

- Timestamp values could be shared by both Round Trip Time Measurement (RTTM) and Protection Against Wrapped Sequence Numbers (PAWS).
- Timestamps can be placed on ACKs as well as data segments.

As pointed out in RFC 1323, "TCP implements reliable data delivery by retransmitting segments that are not acknowledged within some retransmission timeout (RTO) interval . . . RTO is determined by estimating the mean and variance of the measured round-trip time (RTT)"—thus the importance of an accurate RTT.

Because timestamps can occur in either simplex flow (sender to receiver or receiver to sender), they should be supported for each flow, which is why the timestamp value and timestamp reply are present in a single timestamp option. The timestamp value field contains the current value of the TCP clock of the sender. The timestamp reply field contains the timestamp value sent by the peer in the TCP connection (its timestamp value field,) and it is valid if the ACK bit is set.

In Linux, timestamps are enabled by the **tcp_timestamps** sysctls, which can be accessed by the **/proc/sys/net/ipv4/*** files or with the interface. (See http://www.europe.redhat.com/documentation/man-pages/man7/tcp.7.php3.)

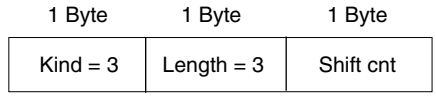
Window Scale Option

The window scale is a 3-byte option that allows TCP to expand the window size limited by the TCP header. The TCP header allocates 16 bits for the window size, which limits the maximum window size to 65 KB (2^{16}). The TCP window scale option permits window sizes beyond 65 KB, expanding the window size to 32 bits and using a scale factor to carry the 32-bits value in the 16-bits window size field (TCP header). The window scale option field carries the scale factor.

This option is only sent on SYN segments, which means the window scale is fixed in either simplex flow until the end of the connection. When the window scale option is used, both the sender and receiver indicate they are ready to use window scaling and communicate the scale factor to use on each receiving window.

Figure 7-25 presents the window scale option fields.

Figure 7-25 *Window Scale Option Fields*



The kind or type field value is 3 and the length is 3. The last field of 1 byte is used to carry the shift count. The sender uses the shift count to right-shift its receive window values the number of bits indicated in the shift.cnt field.

For more information on the window scale operation, consult RFC 1323.

In Linux, TCP SACK is enabled by the **tcp_windows_scaling** sysctls, which can be accessed by the **/proc/sys/net/ipv4/*** files or with the interface. (See http://www.europe.redhat.com/documentation/man-pages/man7/tcp.7.php3.)

PAWS

PAWS lets TCP avoid having the same SEQ used concurrently on a different connection on the same host. In networks that support speeds such as 1 Gigabit per second (Gbps), the SEQ can be wrapped in approximately 34 sec, so at some point there could be two packets with the same SEQ. To avoid this potential confusion, the TCP timestamp is used to timestamp every segment. The timestamp value is sent back to the sender, compared to a duplicate segment, and discarded if the timestamp is less than other timestamps recently received on the connection.

For more details on PAWS, consult section 3 of RFC 1323.

TCP Header Compression

Some of the early work on header compression was done by Van Jacobson in early 1990. This work is documented in RFC 1144, "Compressing TCP/IP Headers for Low-Speed Serial Links," and proves that TCP/IP headers can be compressed to an average in the 3 bytes range. The benefits of header compression mainly focus on improving efficiency and reducing response time on interactive sessions such as Telnet and rlogin, in which the ratio of header bytes to data is high. Bulk data transfers are more efficient, so the benefit of header compression is not significant.

More recent work for IP header compression, including TCP and UDP, is documented in RFC 2507, which is on the standards track. Header compression can be applied to IPv4, IPv6, TCP, and UDP.

The following are the advantages described by RFC 2507, which are mainly applicable to low- and medium-speed links:

- Improve interactive response time
- Allow use of small packets for bulk data with good line efficiency
- Allow use of small packets for delay-sensitive, low data-rate traffic
- Decrease header overhead
- Reduce packet loss rate over lossy links

The principle behind header compression is to avoid sending the header fields that are likely to remain constant through the life of the connection. Figure 7-26 highlights the fields that do not change during the connection (the shaded fields), as stated in RFC 1144.

The headers shown are the minimum size headers (no options in use), which equal 40 bytes, 20 for the IP header and 20 for the TCP header. The number of unchanged bytes is equivalent to 50 percent of the original header size. The context used by the headers can be occasionally communicated, through a full packet header, so that the subsequent compressed headers can be received in reference to the pre-established context.

The compression process occurs only after a connection is established and on packets that belong to an active connection. Figure 7-27 presents the compressed header.

Note that Figure 7-27 displays the compressed packet formats defined in RFC 1144 (part a in Figure 7-27) and later in RFC 2507 (part b in Figure 7-27). Byte 0 in part a in Figure 7-27, which corresponds to byte 1 in part b, is referred to as the *change mask* that determines how the remaining fields are interpreted. The change mask determines which of the fields that are expected to change has changed. A value of 1 in the specific change-mask bit implies the related field has changed.

The connection number (byte 1 in part a of Figure 7-27), which was flagged by the C bit, has been eliminated in part b because the connection identifier (CID) is always present (byte 0 in part b).

Figure 7-26 *IP and TCP Header Fields*

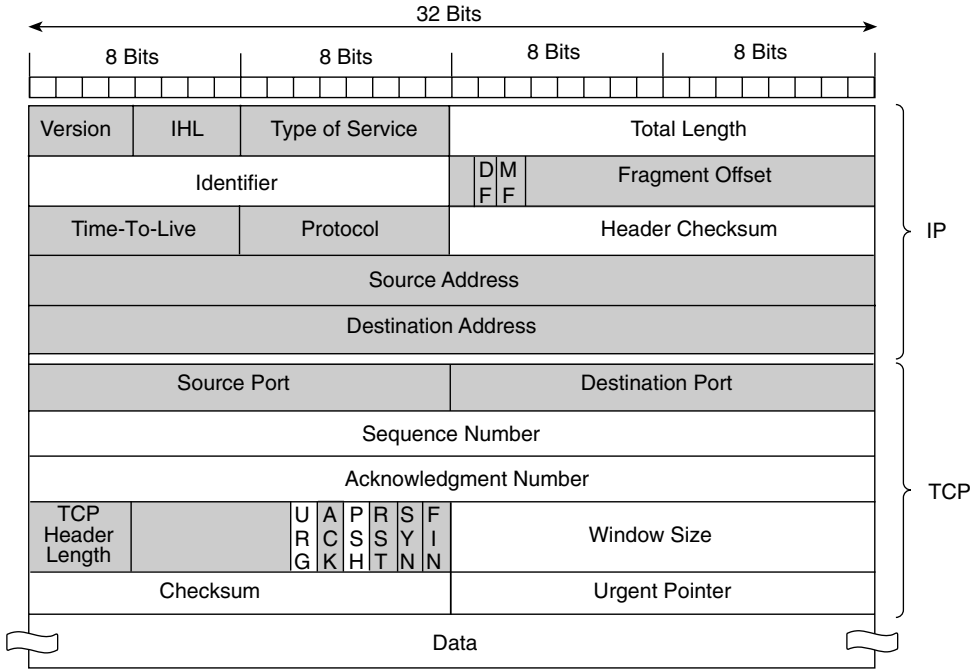
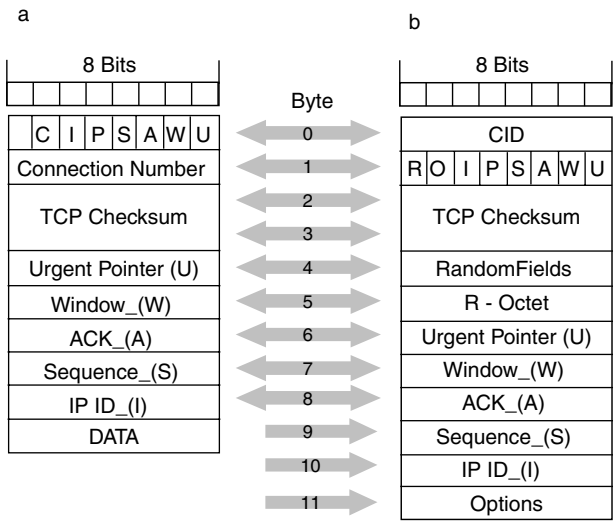


Figure 7-27 *Compressed Packet Format*



All the other bits correspond to the fields in Table 7-9.

Table 7-10 *New Mapping of Header Compression Fields*

Bit	RFC 1144	RFC 2507
C	Connection ID	N/A
R	N/A	R-Octet
O	N/A	Options
I	IP ID delta	IP ID delta
P	Push flag	Push flag
S	Sequence delta	Sequence delta
A	Acknowledgment delta	Acknowledgment delta
W	Window delta	Window delta
U	Urgent pointer	Urgent pointer

The CID is used to send the context, which keeps track of the IP version (IPv4 or IPv6) and the random fields that might be present.

The "delta" (Δ shown in the diagrams) is the difference in value between the field in this header and the field on the previous header. The delta value is expected to be small, which makes it cheaper to send than the complete new value.

The I bit signals whether it is an IPv4 header (set to 1) immediately before the TCP header. The R bit is used for random fields to be included "as is" in a compressed header because they are expected to change unpredictably. The header-compression work is applicable mainly to point-to-point links one hop at a time.

For more information on header compression, refer to RFC 1144 and RFC 2507.

NOTE Microsoft's TCP/IP stack implementation supports RFC 1144. See http://www.microsoft.com/windows2000/docs/tcpip2000.doc for more details.

To disable IP header compression on a Microsoft TCP/IP stack, refer to the knowledge base article at http://support.microsoft.com/default.aspx?scid=kb;en-us;161986.

You enable IP header compression on Cisco routers by using the command **ip tcp header-compression**. Header compression is supported on serial lines using Frame Relay, high-level data link control (HDLC), or Point-to-Point Protocol (PPP) encapsulation.