Large TCP Windows

In order to achieve high data rates with TCP over "long fat networks", i.e. network paths with a large bandwidth-delay product, TCP sinks (that is, hosts receiving data transported by TCP) must advertise a large TCP receive window (referred to as just 'the window', since there is not an equivalent advertised 'send window').

The window is a 16 bit value (bytes 15 and 16 in the TCP header) and so, in TCP as originally specified, it is limited to a value of 65535 (64K). The receive window sets an upper limit on the sustained throughput achieveable over a TCP connection since it represents the maximum amount of unacknowledged data (in bytes) there can be on the TCP path. Mathematically, achieveable throughput can never be more than WINDOW_SIZE/RTT, so for a trans-Atlantic link, with say an RTT (Round trip Time) of 150ms, the throughput is limited to a maximum of 3.4Mbps. With the emergence of "long fat networks", the limit of 64K bytes (on some systems even just 32K bytes!) was clearly insufficient and so RFC 7323 laid down (amongst other things) a way of scaling the advertised window, such that the 16-bit window value can represent numbers larger than 64K.

RFC 7323 extensions

RFC 7323, *TCP Extensions for High Performance,* (and formerly RFC 1323) defines several mechanisms to enable high-speed transfers over LFNs: *Windo w Scaling, TCP Timestamps,* and *Protection Against Wrapped Sequence numbers (PAWS)*.

The TCP window scaling option increases the maximum window size fom 64KB to 1Gbyte, by shifting the window field left by up to 14. The window scale option is used only during the TCP 3-way handshake (both sides **must** set the window scale option in their SYN segments if scaling is to be used in either direction).

It is important to use *TCP timestamps* option with large TCP windows. With the TCP timestamps option, each segment contains a timestamp. The receiver returns that timestamp in each ACK and this allows the sender to estimate the RTT. On the other hand with the TCP timestamps option the problem of wrapped sequence number could be solved (PAWS - Protection Against Wrapped Sequences) which could occur with large windows.

(Auto) Configuration

In the past, most operating systems required manual tuning to use large TCP windows. The OS-specific tuning section contains information on how to to this for a set of operating systems.

Since around 2008-2010, many popular operating systems will use large windows and the necessary protocol options by default, thanks to TCP Buffer Auto-Tuning.

Can TCP Windows ever be *too* large?

There are several potential issues when TCP Windows are larger than necessary:

1. When there are many active TCP connection endpoints (sockets) on a system - such as a popular Web or file server - then a large TCP window size will lead to high consumption of system (kernel) memory. This can have a number of negative consequences: The system may run out of buffer space so that no new connections can be opened, or the high occupation of kernel memory (which typically must reside in actual RAM and cannot be "paged out" to disk) can "starve" other processes of access to fast memory (cache and RAM)
2. Large windows can cause large "bursts" of consecutive segments/packets. When there is a bottleneck in the path - because of a slower link or because of cross-traffic - these bursts will fill up buffers in the network device (router or switch) in front of that bottleneck. The larger these bursts, the higher are the risks that this buffer overflows and causes multiple segments to be dropped. So a large window can lead to "sawtooth" behavior and worse link utilisation than with a just-big-enough window where TCP could operate at a steady rate.

Several methods for automatic TCP buffer tuning have been developed to resolve these issues, some of which have been implemented in (at least) recent Linux versions.

References

- RFC 7323, *TCP Extensions for High Performance*, D. Borman, B. Braden, V. Jacobson, B. Scheffenegger, September 2014 (obsoletes RFC 1323 from May 1993)
- *Achieving Reliable High Performance in LFNs*, S. Ubik and P. Cimbal, TNC 2003, May 2003, http://staff.cesnet.cz/~ubik/publications/2003/terena2003.pdf
- *Debugging end-to-end performance in commodity operating systems*, S. Ubik and P. Cimbal, PFLDnet 2003, February 2003, http://staff.cesnet.cz/~ubik/publications/2003/pfldnet2003.pdf

– Main.SimonLeinen - 27 Oct 2004 - 27 Sep 2014