SelectiveAcknowledgements

TCP Selective Acknowledgements (SACK)

Selective Acknowledgements are a refinement of TCP's traditional "cumulative" acknowledgements.

SACKs allow a receiver to acknowledge non-consecutive data, so that the sender can retransmit only what is missing at the receivers end. This is particularly helpful on paths with a large bandwidth-delay product (BDP).

TCP may experience poor performance when multiple packets are lost from one window of data. With the limited information available from cumulative acknowledgments, a TCP sender can only learn about a single lost packet per round trip time. An aggressive sender could choose to retransmit packets early, but such retransmitted segments may have already been successfully received.

A Selective Acknowledgment (SACK) mechanism, combined with a selective repeat retransmission policy, can help to overcome these limitations. The receiving TCP sends back SACK packets to the sender informing the sender of data that has been received. The sender can then retransmit only the missing data segments.

Multiple packet losses from a window of data can have a catastrophic effect on TCP throughput. TCP uses a cumulative acknowledgment scheme in which received segments that are not at the left edge of the receive window are not acknowledged. This forces the sender to either wait a roundtrip time to find out about each lost packet, or to unnecessarily retransmit segments which have been correctly received. With the cumulative acknowledgment scheme, multiple dropped segments generally cause TCP to lose its ACK-based clock, reducing overall throughput. Selective Acknowledgment (SACK) is a strategy which corrects this behavior in the face of multiple dropped segments. With selective acknowledgments, the data receiver can inform the sender about all segments that have arrived successfully, so the sender need retransmit only the segments that have actually been lost.

The selective acknowledgment extension uses two TCP options. The first is an enabling option, "SACK-permitted", which may be sent in a SYN segment to indicate that the SACK option can be used once the connection is established. The other is the SACK option itself, which may be sent over an established connection once permission has been given by SACK-permitted.

Blackholing Issues

Enabling SACK globally used to be somewhat risky, because in some parts of the Internet, TCP SYN packets offering/requesting the SACK capability were filtered, causing connection attempts to fail. By now, it seems that the increased deployment of SACK has caused most of these filters to disappear.

Performance Issues

Sometimes it is not recommended to enable SACK feature, e.g. for the Linux 2.4.x TCP SACK implementation suffers from significant performance degradation in case of a burst of packet loss. People from CERN observed that a burst of packet loss considerably affects TCP connections with large bandwidth delay product (several MBytes), because the TCP connection doesn't recover as it should. After a burst of loss, the throughput measured in their testbed is close to zero during 70 seconds. This behavior is not compliant with TCP's RFC. A timeout should occur after a few RTTs because one of the loss couldn't be repaired quickly enough and the sender should go back into slow start.

For more information take a look at http://sravot.home.cern.ch/sravot/Networking/TCP_performance/tcp_bug.htm

Additionally, work done at Hamilton Institute found that SACK processing in the Linux kernel is inefficient even for later 2.6 kernels, where at 1Gbps networks performance for a long file transfer can lose about 100Mbps of potential. Most of these issues should have been fixed in Linux kernel 2.6.16.

Detailed Explanation

The following is closely based on a mail that Baruch Even sent to the pert-discuss mailing list on 25 Jan '07:

The Linux TCP SACK handling code has in the past been extremely inefficient - there were multiple passes on a linked list which holds all the sent packets. This linked list on a large BDP link can span 20,000 packets. This meant multiple traversals of this list take longer than it takes for another packet to come. Pretty quickly after a loss with SACK the sender's incoming queue fills up and ACKs start getting dropped. There used to be an anti-DoS mechanism that would drop all packets until the queue had emptied - with a default of 1000 packets that took a long time and could easily drop all the outstanding ACKs resulting in a great degradation of performance. This value is set with `proc/sys/net/core/netdev_max_backlog`

This situation has been slightly improved in that though there is still a 1000 packet limit for the network queue, it acts as a normal buffer i.e. packets are accepted as soon as the queue dips below 1000 again.

Kernel 2.6.19 should be the preferred kernel now for high speed networks. It is believed it has the fixes for all former major issues (at least those fixes that were accepted to the kernel), but it should be noted that some other (more minor) bugs have appeared, and will need to be fixed in future releases.

Historical Note

Selective acknowledgement schemes were known long before they were added to TCP. Noel Chiappa mentioned that Xerox' PUP protocols had them in a posting to the `tcp-ip` mailing list in August 1986. Vint Cerf responds with a few notes about the thinking that lead to the cumulative form of the original TCP acknowledgements.

References

- *TCP Selective Acknowledgment Options*, RFC 2018, M. Mathis, J. Mahdavi, S. Floyd, A. Romanow, October 1996
- *An Extension to the Selective Acknowledgement (SACK) Option for TCP*, RFC 2883. S. Floyd, J. Mahdavi, M. Mathis, M. Podolsky. July 2000.

- *An Experimental Investigation of TCP Performance in High Bandwidth-Delay Product Paths*, B. Even, February 2007 - This thesis contains an analysis of performance problems with the SACK code in Linux 2.6 as well as suggested improvements.

-- Main.UlrichSchmid & Main.SimonLeinen - 02 Jun 2005 - 14 Jan 2007

– Main.BartoszBelter - 16 Dec 2005

-- Main.BaruchEven - 05 Jan 2006