TCP Acknowledgements

In TCP's sliding-window scheme, the receiver *acknowledges* the data it receives, so that the sender can advance the window and send new data. As originally specified, TCP's acknowledgements ("ACKs") are *cumulative*: the receiver tells the sender how much *consecutive* data it has received. More recently, selective acknowledgements were introduced to allow more fine-grained acknowledgements of received data.

Delayed Acknowledgements and "ack every other"

RFC 831 first suggested a *delayed acknowledgement* (Delayed ACK) strategy, where a receiver doesn't always immediately acknowledge segments as it receives them. This recommendation was carried forth and specified in more detail in RFC 1122 and RFC 5681 (formerly known as RFC 2581). RFC 5681 mandates that an acknowledgement be sent for at least every other full-size segment, and that no more than 500ms expire before any segment is acknowledged.

The resulting behavior is that, for longer transfers, acknowledgements are only sent for every two segments received ("ack every other"). This is in order to reduce the amount of reverse flow traffic (and in particular the number of small packets). For transactional (request/response) traffic, the delayed acknowledgement strategy often makes it possible to "piggy-back" acknowledgements on response data segments.

A TCP segment which is **only** an acknowledgement i.e. has no payload, is termed a *pure ack*.

Delayed Acknowledgments should be taken into account when doing *RTT estimation*. As an illustration, see this change note for the Linux kernel from Gavin McCullagh.

Critique on Delayed ACKs

John Nagle nicely explains problems with current implementations of delayed ACK in a comment to a thread on Hacker News:

> *Here's how to think about that. A delayed ACK is a bet. You're betting that there will be a reply, upon with an ACK can be piggybacked, before the fixed timer runs out. If the fixed timer runs out, and you have to send an ACK as a separate message, you lost the bet. Current TCP implementations will happily lose that bet forever without turning off the ACK delay. That's just wrong.*
>
> *The right answer is to track wins and losses on delayed and non-delayed ACKs. Don't turn on ACK delay unless you're sending a lot of non-delayed ACKs closely followed by packets on which the ACK could have been piggybacked. Turn it off when a delayed ACK has to be sent.*

Duplicate Acknowledgements

A duplicate acknowledgement (DUPACK) is one with the same acknowledgement number as its predecessor - it signifies that the TCP receiver has received a segment newer than the one it was expecting i.e. it has missed a segment. The missed segment might not be lost, it might just be re-ordered. For this reason the TCP sender will not assume data loss on the first DUPACK but (by default) on the third DUPACK, when it will, as per RFC 5681, perform a "Fast Retransmit", by sending the segment again without waiting for a timeout. DUPACKs are never delayed; they are sent immediately the TCP receiver detects an out-of-order segment.

"Quickack mode" in Linux

The TCP implementation in Linux has a special receive-side feature that temporarily disables delayed acknowledgements/ack-every-other when the receiver thinks that the sender is in slow-start mode. This speeds up the slow-start phase when RTT is high. This "quickack mode" can also be explicitly enabled or disabled with `setsockopt()` using the `TCP_QUICKACK` option. The Linux modification has been criticized because it makes Linux' slow-start more aggressive than other TCPs' (that follow the SHOULDs in RFC 5681), without sufficient validation of its effects.

"Stretch ACKs"

Techniques such as LRO and GRO (and to some level, Delayed ACKs as well as ACKs that are lost or delayed on the path) can cause ACKs to cover many more than the two segments suggested by the historic TCP standards. Although ACKs in TCP have always been defined as "cumulative" (with the exception of SACKs), some congestion control algorithms have trouble with ACKs that are "stretched" in this way. In January 2015, a patch set was submitted to the Linux kernel network development with the goal to improve the behavior of the Reno and CUBIC congestion control algorithms when faced with stretch ACKs.

References

- *TCP Congestion Control*, RFC 5681, M. Allman, V. Paxson, E. Blanton, September 2009
- RFC 813, *Window and Acknowledgement Strategy in TCP*, D. Clark, July 1982
- RFC 1122, *Requirements for Internet Hosts -- Communication Layers*, R. Braden (Ed.), October 1989

– Main.TobyRodwell - 2005-04-05

– Main.SimonLeinen - 2007-01-07 - 2015-01-28