TCP protocol

Last updated: 9 February 2008

Contents

Congestion control

The following variables are used in the tcp_sock for congestion control:

snd_cwnd	The size of the congestion window
snd_ssthresh	Slow start threshold. We are in slow start if snd_cwnd is less than this.
snd_cwnd_cnt	A counter used to slow down the rate of increase once we exceed slow start threshold.
snd_cwnd_clamp	This is the maximum size that snd_cwnd can grow to.
snd_cwnd_stamp	Timestamp for when congestion window last validated.
snd_cwnd_used	Used as a highwater mark for how much of the congestion window is in use. It is used to adjust snd_cwnd down when the link is limited by the application rather than the network.

As of 2.6.13, Linux supports pluggable congestion control algorithms.
A congestion control mechanism can be registered through functions in
tcp_cong.c. The functions used by the congestion control mechanism are
registered via passing a tcp_congestion_ops struct to
tcp_register_congestion_control. As a minimum name, ssthresh,
cong_avoid, min_cwnd must be valid.

Private data for a congestion control mechanism is stored in tp->ca_priv.
tcp_ca(tp) returns a pointer to this space. This is preallocated space - it
is important to check the size of your private data will fit this space, or
alternatively space could be allocated elsewhere and a pointer to it could
be stored here.

There are three kinds of congestion control algorithms currently: The
simplest ones are derived from TCP reno (highspeed, scalable) and just
provide an alternative the congestion window calculation. More complex
ones like BIC try to look at other events to provide better
heuristics. There are also round trip time based algorithms like
Vegas and Westwood+.

Good TCP congestion control is a complex problem because the algorithm
needs to maintain fairness and performance. Please review current
research and RFC's before developing new modules.

The method that is used to determine which congestion control mechanism is
determined by the setting of the sysctl net.ipv4.tcp_congestion_control.
The default congestion control will be the last one registered (LIFO);
so if you built everything as modules, the default will be reno. If you
build with the defaults from Kconfig, then CUBIC will be builtin (not a

module) and it will end up the default.

If you really want a particular default value then you will need
to set it with the sysctl. If you use a sysctl, the module will be autoloaded
if needed and you will get the expected protocol. If you ask for an
unknown congestion method, then the sysctl attempt will fail.

If you remove a tcp congestion control module, then you will get the next
available one. Since reno cannot be built as a module, and cannot be
deleted, it will always be available.

How the new TCP output machine [nyi] works.
==========

Data is kept on a single queue. The skb->users flag tells us if the frame is
one that has been queued already. To add a frame we throw it on the end. Ack
walks down the list from the start.

We keep a set of control flags

```
        sk->tcp_pend_event

                TCP_PEND_ACK                    Ack needed
                TCP_ACK_NOW                     Needed now
                TCP_WINDOW                      Window update check
                TCP_WINZERO                     Zero probing


        sk->transmit_queue              The transmission frame begin
        sk->transmit_new                First new frame pointer
        sk->transmit_end                Where to add frames

        sk->tcp_last_tx_ack             Last ack seen
        sk->tcp_dup_ack                 Dup ack count for fast retransmit
```

Frames are queued for output by tcp_write. We do our best to send the frames
off immediately if possible, but otherwise queue and compute the body
checksum in the copy.

When a write is done we try to clear any pending events and piggy back them.
If the window is full we queue full sized frames. On the first timeout in
zero window we split this.

On a timer we walk the retransmit list to send any retransmits, update the
backoff timers etc. A change of route table stamp causes a change of header
and recompute. We add any new tcp level headers and refinish the checksum
before sending.