Increase of the Congestion Window when the Sender Is Rate-Limited

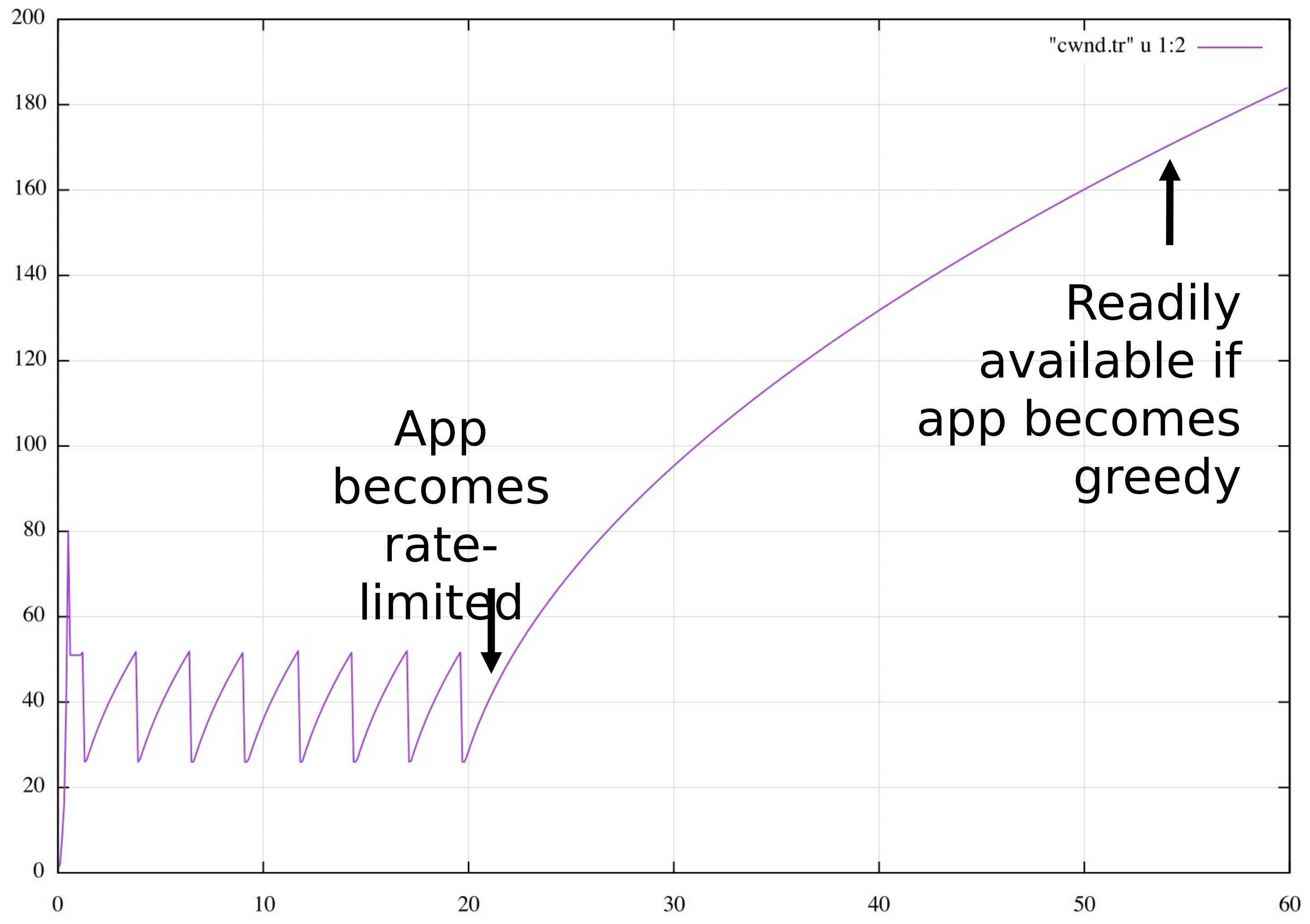
draft-welzl-ccwg-ratelimited-increase-01

Michael Welzl, Tom Henderson, Gorry Fairhurst

CCWG

IETF 119

Why are we doing this, and why PS?



- Current TCP specifications allow this, when app is just not producing more data, or rwnd is limited!
- ns-3 & ns-2 try to follow the specs; and implement this
  - Does anyone else?
  - We hope not.

=> Primary motivation: fix this with a PS.

Specifying the increase

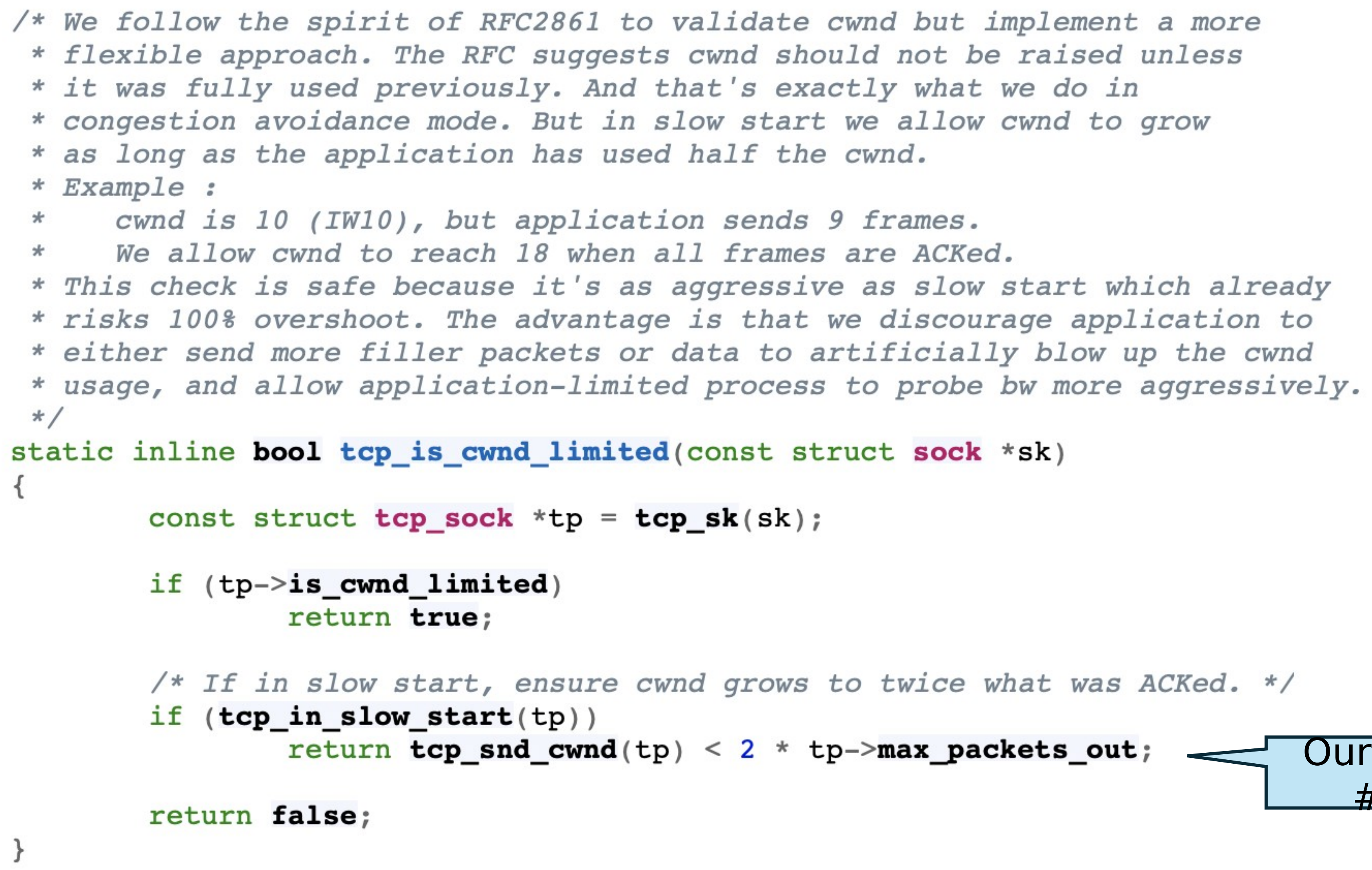
- Why not just say "stop", when cwnd is not fully utilized?
  - There are complications related to transients (pacing, ..)
  - Also, sending X < cwnd packets in the previous RTT *did* probe for capacity

    => Should allow growing cwnd to 2*X in Slow Start (and similar in general, e.g. in Congestion Avoidance or HyStart++)

- Our proposed rules: senders (..)
  1. **MUST** include a limit to the growth of cwnd when FlightSize < cwnd.
  2. **SHOULD** limit the growth of cwnd when FlightSize < cwnd with inc(maxFS).

- maxFS: max. FlightSize so far, since last increase (or beginning)
- inc: the CC's increase if the sender were not rate-limited

Linux, since kernel 3.16, August 2014



```c
/* We follow the spirit of RFC2861 to validate cwnd but implement a more
 * flexible approach. The RFC suggests cwnd should not be raised unless
 * it was fully used previously. And that's exactly what we do in
 * congestion avoidance mode. But in slow start we allow cwnd to grow
 * as long as the application has used half the cwnd.
 * Example :
 *    cwnd is 10 (IW10), but application sends 9 frames.
 *    We allow cwnd to reach 18 when all frames are ACKed.
 * This check is safe because it's as aggressive as slow start which already
 * risks 100% overshoot. The advantage is that we discourage application to
 * either send more filler packets or data to artificially blow up the cwnd
 * usage, and allow application-limited process to probe bw more aggressively.
 */
static inline bool tcp_is_cwnd_limited(const struct sock *sk)
{
        const struct tcp_sock *tp = tcp_sk(sk);

        if (tp->is_cwnd_limited)
                return true;

        /* If in slow start, ensure cwnd grows to twice what was ACKed. */
        if (tcp_in_slow_start(tp))
                return tcp_snd_cwnd(tp) < 2 * tp->max_packets_out;

        return false;
}
```

Our rule #2

RFCs are a bit messy here... our draft proposes to clean this up.

- E.g., RFC 9438
  - "Cubic doesn't increase cwnd when it is limited by the sending application or rwnd."
  - But *Linux: tcp_is_cwnd_limited* also applies to Cubic.

More examples in the appendix

- E.g., RFC 9260
  - "When cwnd is less than or equal to ssthresh, an SCTP endpoint MUST use the slow-start algorithm to increase cwnd only if the current congestion window is being fully utilized and the data sender is not in Fast Recovery. Only when these two conditions are met can the cwnd be increased; otherwise, the cwnd MUST NOT be increased."
  - This just says "stop", and is only written for Slow Start.

Open questions

- How ought we to deal with transients?
  - Could re-formulate how a sender knows that the app is rate-limited
  - Or maybe: just turn our rule #2 into MUST?
    MUST limit the growth of cwnd when FlightSize < cwnd with inc(maxFS).

- How many (and which) RFCs to update?
  - There's also (MP-)DCCP, we have the RMCAT CCs, even multicast...

Thank you !