BBR WITH L4S SUPPORT
A FEW EXPERIMENTS AND FINDINGS

Ingemar Johansson, Ericsson Research
INTRO

› BBR = Bottleneck Bandwidth and RTT (BBR) congestion control
  - Developed by Google
  - Estimated bottleneck bandwidth and min RTT, based on heuristics derived from normal TCP ACKs.

› Scope of this work:
  - Modify BBR with L4S support, (BBR evo)
  - No claims that this is the final, there is room for improvement
  - ...But the changes so far are very minimalistic

› Note… these are simulations!
BBR EVO MODIFICATIONS

1. L4S support added (ECN echo code from tcp_dctcp.c)

2. Function `bbr_update_bw(…)`
   Bandwidth estimates take amount of CE marked packets into account
   
   - \[ bw = \frac{rs->delivered}{rs->interval}; \]
     changed to
     \[ bw = \frac{rs->delivered - rs->delivered_ce}{K} / rs->interval; \]
     where `delivered_ce` are the amount of delivered and CE marked packets in the given interval.
     \( K = 4 \) seems to be OK
   - Additional state variable(s) `delivered_ce` need where ‘delivered’ is specified
3. Gain cycle changed to 3 RTTs (from 8 RTTs)
   - Reduced gain variation [9/8,7/8,1.0] instead of [5/4,3/4,1.0] → less jitter but (sometimes) slightly slower rate increase

4. Min RTT probing is removed
   - L4S gives very short (or zero) queue delay, but min RTT probing may still be needed in reality

5. BW window reduced to 2 RTTs (was 8 RTTs)
   - Warning.. Too short window can reduce performance for app limited traffic

6. BBR mode forced to BBR_PROBE_BW if more than 1 RTT with CE marked packets and in BBR_STARTUP
Update in function
bbr_update_bw(...)

Additional code in tcp_input.c
- Added delivered_ce counters
- Similar to delivered counter used with rate sample but only counting CE marked packets

/* Estimate the bandwidth based on how fast packets are delivered */
static void bbr_update_bw(struct sock *sk, const struct rate_sample *rs)
{
... 

bbr->round_start = 0;
if (rs->delivered < 0 || rs->interval_us <= 0)
    return; /* Not a valid observation */

/* See if we've reached the next RTT */
if (!before(rs->prior_delivered, bbr->next_rtt_delivered)) {

}

bbr_lt_bw_sampling(sk, rs);

/* Divide delivered by the interval to find a (lower bound) bottleneck bandwidth sample. Delivered is in packets and interval_us in uS and * ratio will be <<1 for most connections. So delivered is first scaled. */
bw = (u64)rs->delivered * BW_UNIT;
bw -= (u64)(rs->delivered_ce >> 2) * BW_UNIT;
do_div(bw, rs->interval_us);
BBR VS BBR EVO COMPARISON

- RTT = 20ms
- L4S mark threshold = 2ms
- BBR evo manages to keep standing queue < 5ms
- BBR has more problems
- BBR evo is slightly slower in the rate increase

Queue delay = Network queue delay

RTT probing

Throughput [Mbps]

Queue delay [s]
BBR VS BBR EVO COMPARISON
PHANTOM QUEUE

› RTT = 20ms
› Phantom queue
  – L4S mark threshold 95% of BW
  – Measurement period 5ms
› BBR evo manages to keep standing queue < 1ms
› ~10% peak bandwidth sacrificed
BBR MULTIPLE FLOWS

› Large file transfers
› BBR does not keep standing queue small
   - Mainly an $RTT_{\text{min}}$ estimation issue.
› Flow rates converge.. eventually
BBR evo MULTIPLE FLOWS

› L4S mark threshold = 2ms
› Quite low queue delay
  – But higher than 2ms threshold
› Reasonably good convergence when new flows arrive
› Newly arrived flows ramp up more slowly
BBR evo MULTIPLE FLOWS
PHANTOM QUEUE (95%)

› Phantom queue
  – L4S mark threshold 95% of BW
  – Measurement period 5ms

› Very low queue delay
  – But not zero
Phantom queue
- L4S mark threshold 90% of BW
- Measurement period 5ms

Very low queue delay, but not zero
BITRATE RAMP

› RTT = 10ms
› Channel bandwidth reduced from 100 to 10 Mbps in 400ms
› BBR evo reacts better but there is room for improvement
› General problem that throughput is overestimated
BITRATE RAMP, ZOOM IN

› Max BW is slightly overestimated
› More conservative bandwidth probing may help
  – But that can harm flow fairness
RTT FAIRNESS

- BW = 100Mbps
- 100s simulation
- TCP flow #1: RTT = 10ms
- TCP flow #2: RTT = 10, 12, 20, 30, 50ms
- BBR evo: L4S mark threshold = 2ms

<table>
<thead>
<tr>
<th>RTT flow #2</th>
<th>BBR</th>
<th>BBR evo</th>
</tr>
</thead>
<tbody>
<tr>
<td>10ms</td>
<td>46/50</td>
<td>50/46</td>
</tr>
<tr>
<td>12ms</td>
<td>30/65</td>
<td>41/53</td>
</tr>
<tr>
<td>20ms</td>
<td>12/82</td>
<td>33/62</td>
</tr>
<tr>
<td>30ms</td>
<td>10/90</td>
<td>66/30</td>
</tr>
<tr>
<td>50ms</td>
<td>7/87</td>
<td>71/22</td>
</tr>
</tbody>
</table>
RTT FAIRNESS CONT..

- BW = 100Mbps
- 100s simulation
- TCP flow #1: RTT = 2ms
- TCP flow #2: RTT = 2, 5, 10, 15, 20ms
- BBR evo: L4S mark threshold = 2ms

<table>
<thead>
<tr>
<th>RTT flow #2</th>
<th>BBR</th>
<th>BBR.evo</th>
</tr>
</thead>
<tbody>
<tr>
<td>2ms</td>
<td>41/55</td>
<td>46/51</td>
</tr>
<tr>
<td>5ms</td>
<td>21/75</td>
<td>39/57</td>
</tr>
<tr>
<td>10ms</td>
<td>12/84</td>
<td>92/3</td>
</tr>
<tr>
<td>15ms</td>
<td>6/88</td>
<td>59/37</td>
</tr>
<tr>
<td>20ms</td>
<td>7/88</td>
<td>55/40</td>
</tr>
</tbody>
</table>
CONCLUSION

› BBR is quite easy to modify for L4S support
  - But there are probably better ways to do this

› The evolved BBR with L4S support
  - Converges quite well when multiple flows compete for the same bottleneck
  - Keeps standing queue small (or very small with phantom queues)
  - A certain degree of jitter, a result of the necessity to be a bit aggressive in order to achieve convergence for multiple flows

Comments are welcome
ingemar.s.Johansson@ericsson.com
BUZZ WORDS

› BBR = Bottleneck-Bandwidth-RTT
› L4S = Low Loss Low Latency Scalable throughput
› Phantom queue = Link bitrate is measured at e.g. 5ms intervals. Packets are marked when the link bitrate exceeds a given fraction (e.g 95%) of the maximum rate.