BBR Startup Gain: a Derivation

June 2018

The Google BBR team

This document first presents an analytic derivation of the BBR Startup pacing gain constant, and then discusses a validation of that constant using Linux netem emulation and a simple discrete event simulation.

Analytic derivation

(1) For simplicity, let RTT=1 unit of time.

(2) For smooth traffic to avoid queue pressure, we want the sending rate to be driven by smooth pacing, so that the sending rate is equal to the pacing rate. To find the smoothest, gentlest pacing approach that roughly matches the growth dynamics of unpaced Reno/CUBIC, we want the pacing rate (sending rate) to double each RTT (each unit time interval). So let the pacing rate (sending rate) at a given time be given by some curve that doubles each unit of time:
$PacingRate(t) = SendingRate(t) = k\,2^t$
We use the constant scaling factor of k because different transport protocol implementations may choose different initial sending rates, depending on the environment or applicable standards.

(3) Based on the BBR architecture, we also want the pacing rate to be some pacing gain g times the estimated bandwidth:
$PacingRate(t) = g \bullet EstimatedBandwidth(t)$

(4) In Startup in an ideal pipe (uniform RTT and rate) with one flow, the estimated bandwidth will be the latest bandwidth sample, which will

be the amount of data delivered over the most recent round trip (because the RTT is 1 time unit):

$$\begin{aligned} EstimatedBandwidth(t) &= DataDeliveredInRoundTripUpTo(t)/RTT \\ &= DataDeliveredInRoundTripUpTo(t) \end{aligned}$$

(5) In Startup in an ideal pipe (uniform RTT and rate) with one flow that has not filled the pipe yet, the amount of data delivered over the most recent round trip will be the amount of data sent in the previous round trip:

$$DataDeliveredInRoundTripUpTo(t) = DataSentInRoundTripUpTo(t-1)$$

(6) In Startup, BBR sets the cwnd high enough that sending is governed by the pacing rate. This means that the sending rate is the pacing rate, so we know the rate of sending from (2). This means that we can calculate the amount of data sent over an interval by integrating the pacing rate over the time interval in question:

$$DataSentInRoundTripUpTo(t-1) = \int_{t-2}^{t-1} PacingRate(t)\, dt$$

(7) Note that (3)-(6) reflect our decision to compute the pacing rate as a multiple of g times the estimated bandwidth, which is in turn derived from delivery rate history, which is in turn derived from the integral of the pacing rate. And together (3)-(6) combined yield:

$$\begin{aligned} PacingRate(t) &= g \bullet EstimatedBandwidth(t) \\ &= g \bullet DataDeliveredInRoundTripUpTo(t) \\ &= g \bullet DataSentInRoundTripUpTo(t-1) \\ PacingRate(t) &= g \bullet \int_{t-2}^{t-1} PacingRate(t)\, dt \qquad (7) \end{aligned}$$

(8) Thus we have two different ways of defining or constraining the pacing rate: (2) and (7). Thus to derive the gain we can set them equal and solve for g. This yields:

$$PacingRate(t) = k\,2^t = g \bullet \int_{t-2}^{t-1} PacingRate(t)\, dt = g \bullet \int_{t-2}^{t-1} k\,2^t\, dt$$

(9) Symbolically evaluating that integral on the RHS, and then solving for g, we get:

$$PacingRate(t) = k\,2^t = g \bullet \int_{t-2}^{t-1} k\,2^t\,dt$$

$$k\,2^t = g \bullet \frac{k\,2^{t-2}}{ln(2)}$$

$$g = \frac{k\,2^t\,ln(2)}{k\,2^{t-2}}$$

$$g = \frac{2^t\,ln(2)}{2^{t-2}}$$

$$g = 2^2\,ln(2)$$

$$g = 4\,ln(2)$$

$$g = 2.7725887...$$

(10) The Startup gain that BBR used in its original release was $2 / ln(2) = 2.89$. This derivation of $4\,ln(2) = 2.77$ is about $(2.89 - 2.77)/2.89 = 4\%$ lower than the original value.

Validation Via Emulation

We have run quick tests using netem, comparing high_gain=2.77 (710 / 256 = 2.773) instead of 2.89 (739). They look visually the same, and the Linux TCP BBR throughputs for 2-second transfers over a netem-emulated path with bw=100Mbps, rtt=100ms are quite similar: 78.35M (g=2.89) vs 77.98 (g=2.77), showing less than 0.5% difference in throughput. The delay from the first ACK until a bandwidth sample >= 99Mbps was .864 secs with high_gain=2.89, vs .891 secs with high_gain=2.77 (3% longer).

Validation Via Simulation

We wrote a tiny discrete event simulator to run fast, deterministic simulations of BBR Startup behavior with different gain values:

https://github.com/google/bbr/tree/master/Documentation/startup/gain/simulation

From these simulations, it seems that 2.77, a value just *below* 4 * ln(2), is not quite high enough to double the bandwidth estimate and pacing rate in all rounds between rounds 10-15:

```
./startup 2.77 | grep ROUND
ROUND: bw: 0.000x t: 0.100000 round: 1 cwnd:     20 pif:      0 bw: 100.00000
ROUND: bw: 0.000x t: 0.200000 round: 2 cwnd:     21 pif:     19 bw: 100.00000
ROUND: bw: 2.000x t: 0.300000 round: 3 cwnd:     41 pif:     39 bw: 200.00000
ROUND: bw: 2.000x t: 0.400000 round: 4 cwnd:     81 pif:     68 bw: 400.00000
ROUND: bw: 1.725x t: 0.500189 round: 5 cwnd:    150 pif:    143 bw: 690.00000
ROUND: bw: 2.087x t: 0.600354 round: 6 cwnd:    294 pif:    292 bw: 1440.0000
ROUND: bw: 2.035x t: 0.700354 round: 7 cwnd:    587 pif:    570 bw: 2930.0000
ROUND: bw: 1.949x t: 0.800385 round: 8 cwnd:   1158 pif:   1145 bw: 5710.0000
ROUND: bw: 2.007x t: 0.900420 round: 9 cwnd:   2304 pif:   2302 bw: 11460.000
ROUND: bw: 2.010x t: 1.000420 round: 10 cwnd:   4607 pif:   4586 bw: 23030.00
ROUND: bw: 1.992x t: 1.100432 round: 11 cwnd:   9194 pif:   9169 bw: 45870.00
ROUND: bw: 1.999x t: 1.200437 round: 12 cwnd:  18364 pif:  18347 bw: 91700.00
ROUND: bw: 2.001x t: 1.300440 round: 13 cwnd:  36712 pif:  36649 bw: 183480.0
ROUND: bw: 1.997x t: 1.400440 round: 14 cwnd:  73362 pif:  73253 bw: 366500.0
ROUND: bw: 1.999x t: 1.500440 round: 15 cwnd: 146616 pif: 146437 bw: 732540.0
```

But 2.78, a value just *above* 4 * ln(2), *is* high enough to double the bandwidth estimate and pacing rate in all rounds between rounds 10-15:

```
./startup 2.78 | grep ROUND
ROUND: bw: 0.000x t: 0.100000 round: 1 cwnd:     20 pif:      0 bw: 100.00000
ROUND: bw: 0.000x t: 0.200000 round: 2 cwnd:     21 pif:     19 bw: 100.00000
ROUND: bw: 2.000x t: 0.300000 round: 3 cwnd:     41 pif:     39 bw: 200.00000
ROUND: bw: 2.000x t: 0.400000 round: 4 cwnd:     81 pif:     69 bw: 400.00000
ROUND: bw: 1.750x t: 0.500727 round: 5 cwnd:    151 pif:    144 bw: 700.00000
ROUND: bw: 2.071x t: 0.601013 round: 6 cwnd:    296 pif:    294 bw: 1450.0000
ROUND: bw: 2.034x t: 0.701013 round: 7 cwnd:    591 pif:    579 bw: 2950.0000
ROUND: bw: 1.966x t: 0.801112 round: 8 cwnd:   1171 pif:   1158 bw: 5800.0000
ROUND: bw: 1.998x t: 0.901155 round: 9 cwnd:   2330 pif:   2328 bw: 11590.000
ROUND: bw: 2.009x t: 1.001155 round: 10 cwnd:   4659 pif:   4657 bw: 23290.00
ROUND: bw: 2.000x t: 1.101155 round: 11 cwnd:   9317 pif:   9315 bw: 46580.00
ROUND: bw: 2.000x t: 1.201155 round: 12 cwnd:  18633 pif:  18631 bw: 93160.00
ROUND: bw: 2.000x t: 1.301155 round: 13 cwnd:  37265 pif:  37263 bw: 186320.0
ROUND: bw: 2.000x t: 1.401155 round: 14 cwnd:  74529 pif:  74527 bw: 372640.0
ROUND: bw: 2.000x t: 1.501155 round: 15 cwnd: 149058 pif: 149054 bw: 745290.0
```

This seems to support the above analysis showing the required gain to be roughly $g = 2.773$.

Visualizing Slow Start and Startup Alternatives

Here is a superimposed time-sequence diagram illustrating the Slow Start and Startup behavior of several alternative congestion control approaches. These are all netperf Linux TCP transfers over a netem-emulated path with 1Gbps throughput and 100ms RTT. Thick lines are transmits; thin lines are ACKs (the gray lines are receive windows). They are listed in order of increasing latency (for a transfer large enough to fill the pipe):

1. BBR with startup_gain=2.77 [high_gain=710] (IW32; initial cwnd = 32 packets)
2. unpaced CUBIC (IW10)
3. paced CUBIC (IW10)
4. BBR with startup_gain=2.89 [high_gain=739] (IW10)
5. BBR with startup_gain=2.77 [high_gain=710] (IW10)