

<https://dropbox.tech/infrastructure/evaluating-bbrv2-on-the-dropbox-edge-network>

<https://www.ietf.org/proceedings/104/slides/slides-104-iccrg-an-update-on-bbr-00>

<https://translate.google.com/translate?hl=en&sl=zh-CN&u=https://blog.csdn.net/u010643777/article/details/97375310&prev=search&pto=aue>

All of these new design principles need an explicit, independent, tight bound on in-flight data

BBR v2 model:

- Mostly cruise at an operating point that maintains flow balance and leaves headroom

- inflight\_lo: conservative in-flight bound based on recent loss/ECN signals

- Periodically probe beyond flow balance to probe robustly for higher volume, bandwidth

- inflight\_hi: max volume flow had in flight before signals of congestion (loss, ECN)

- If probing higher inflight doesn't trigger loss/ECN signals , grow probing rapidly

- inflight\_probe: incremental probe data beyond inflight\_hi (during probing)

이러한 모든 새로운 설계 원칙은 명시적이고 독립적이며 엄격한 비행 중 데이터가 필요하다.

BBR v2 모델:

- 대부분 유량 균형을 유지하고 헤드룸을 이탈하는 운용지점에서 순항

- inflight\_lo: 최근 손실/ECN 신호에 기반한 보수적인 기내 바운드

- 유동 밸런스를 넘어 주기적으로 프로브하여 더 큰 볼륨, 대역폭을 위한 강력한 프로브

- inflight\_hi: 정체 신호(손실, ECN) 전 최대 체적 흐름이 비행 중이었습니다.

- 고경량 프로빙이 손실/ECN 신호를 유발하지 않는 경우 프로빙을 빠르게 증가시키십시오.

- inflight\_probe: inflight\_hi를 넘어서는 증분 프로브 데이터(프로빙 중)

BBRv2 is a model-based congestion control algorithm that aims for low queues, low loss, and (bounded) Reno/CUBIC coexistence. To maintain a model of the network path, it uses measurements of bandwidth and RTT, as well as (if they occur) packet loss and/or DCTCP/L4S-style ECN signals. Note that although it can use ECN or loss signals explicitly, it does not require either; it can bound its in-flight data based on its estimate of the BDP.

BBRv2는 모델 기반의 정체 제어 알고리즘으로, 낮은 대기열, 낮은 손실, (경계) Reno/CUB 공존을 목표로 한다. 네트워크 경로 모델을 유지하기 위해 대역폭과 RTT의 측정뿐만 아니라 (발생하는 경우) 패킷 손실 및/또는 DCTCP/L4S 스타일의 ECN 신호도 사용한다. ECN 또는 손실 신호를 명시적으로 사용할 수 있지만 둘 중 하나를 요구하지는 않는다. BDP의 추정치에 근거하여 비행 중 데이터를 바인딩할 수 있다.

The model has both higher and lower bounds for the operating range:

lo: bw\_lo, inflight\_lo: conservative short-term lower bound

hi: bw\_hi, inflight\_hi: robust long-term upper bound

The bandwidth-probing time scale is (a) extended dynamically based on estimated BDP to improve coexistence with Reno/CUBIC; (b) bounded by an interactive wall-clock time-scale to be more scalable and responsive than Reno and CUBIC.

이 모델은 운용 범위의 상한과 하한 모두를 가지고 있다:

lo: bw\_lo, inflight\_lo: 보수적인 단기 하한

hi: bw\_hi, inflight\_hi: 강력한 장기 상한

대역폭 확장 시간 척도는 (a) Reno/CUBIC와의 공존을 개선하기 위해 추정된 BDP를 기반으로 동적으로 확장되며, (b) Reno 및 CUBIC보다 확장성과 응답성이 더 높은 대화형 벽시계 시간 척도로 제한된다.

**Elapsed real time**, **real time**, **wall-clock time**, or **wall time** is the actual time taken from the start of a [computer program](https://en.wikipedia.org/wiki/Computer_program) to the end.

**BBR v2 takes packet loss into its control logic**

We bound the BBR-native inter-bw-probe wall clock time to be:

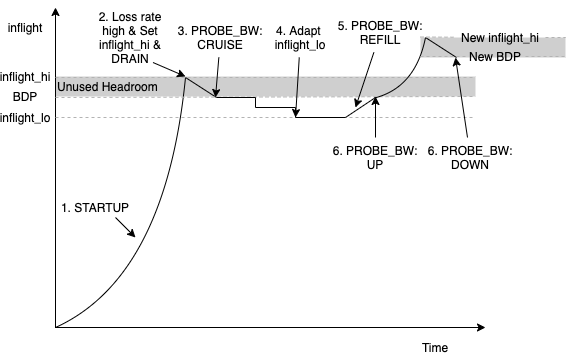
(a) higher than 2 sec: to try to avoid causing loss for a long enough time to allow Reno at 30ms to get 4K video bw, the inter-bw-probe time must be at least: 25Mbps .030sec / (1514bytes) 0.030sec = 1.9secs

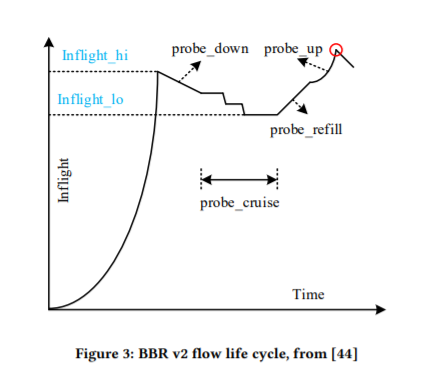
(b) lower than 3 sec: to ensure flows can start probing in a reasonable amount of time to discover unutilized bw on human-scale interactive time-scales (e.g. perhaps traffic from a web page download that we were competing with is now complete).

BBR-native inter-bw-probe wall clock time to be:

(a) 2초 이상: 30ms에 Reno가 4K 비디오 bw를 얻을 수 있을 정도로 장시간 손실을 방지하려면 bw-probe 간 시간이 최소 25Mbps .030sec/(1514bytes) 0.030s = 1.9s여야 함

(b) 3초 미만: 흐름이 인간 규모의 대화형 시간 추적에서 미활용 bw를 발견하기 위한 합리적인 시간 내에 탐색을 시작할 수 있도록 보장하기 위해(예: 우리가 경쟁하고 있던 웹 페이지 다운로드의 트래픽이 현재 완료되었을 수 있음)



  
BBR flow will send packets at the full estimated bandwidth and packet loss is not exploited to indicate link congestion. If the queue length at the bottleneck is smaller than 1.5 ∗ *BDP* [44], multiple BBR flows will cause high packet loss.

Reno/CUBIC flows gain low throughput when share bottleneck with BBR flows. BBR v2 [44] is proposed to solve these issues in BBR v1. BBR v2 is claimed that it can make better coexistence with Reno/CUBIC flows and achieves low queue delay.

BBR 흐름은 완전한 예상 대역폭에서 패킷을 전송하며 패킷 손실은 링크 정체를 나타내기 위해 이용되지 않는다. 병목현상의 대기열 길이가 1.5 BDP[44]보다 작을 경우, 복수의 BBR 흐름이 높은 패킷 손실을 유발한다. Reno/CUBIC 흐름은 BBR 흐름의 공유 병목 현상으로 인해 처리량이 낮아진다. BBR v2[44]는 BBR v1에서 이러한 문제를 해결하기 위해 제안되었다. BBR v2는 Reno/CUBIC 흐름과 더 잘 공존하고 낮은 대기열 지연을 달성할 수 있다고 주장되었다.

BBR v2 takes **packet loss** into its control logic. The life cycle of BBR v2 flow is shown in Figure 3. When the estimated bandwidth does not exceed the target for 3 times, the sender assumes reaching the full bandwidth in BBR v1. Besides that, condition on packet loss is added to exist from StartUp to Drain: **the number of loss packet in a round exceeds 8 and packet loss rate exceeds *loss threshold* (0.02)**. Such condition is applied to avoid excessive packet loss.

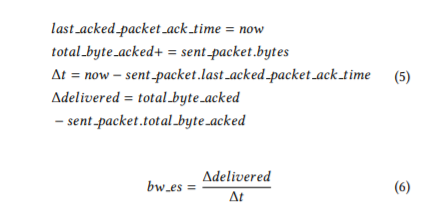
BBR v2는 패킷 손실을 제어 논리에 반영한다. BBR v2 흐름의 수명주기는 그림 3과 같다. 예상 대역폭이 3회 동안 목표를 초과하지 않을 때, 송신자는 BBR v1의 전체 대역폭에 도달한다고 가정한다. 그 외에도, 패킷 손실의 조건은 StartUp에서 Drain까지 존재하도록 추가된다: 라운드에서 손실 패킷의 수는 8을 초과하고 패킷 손실률은 손실 임계값(0.02)을 초과한다. 그러한 조건은 과도한 패킷 손실을 피하기 위해 적용된다.

**INFLIGHT HI**

**INFLIGHT LO**

**If the condition on packet loss holds true, the calculated BDP is assigned to inflight hi**. When a new acknowledge packet arrives, **inflight lo is updated as Equation (8)**. Here, ∆delivered is 4 calculated in the same way in Equation (5) and kBeta is 0.3

패킷 손실에 대한 조건이 참이면 계산된 BDP는 inflight hi에 할당된다. 새로운 승인 패킷이 도착하면, inflight lo는 방정식 (8)로 업데이트된다. 여기서 ∆delivered는 방정식 (5)에서 같은 방법으로 계산하고 kBeta는 0.3이다.





In ProbeBW state, the working mechanism of BBR v2 is quite different from BBR v1. The phases (probe up, probe down, probe cruise) switching is no longer depended on the time interval *RTT\_min*

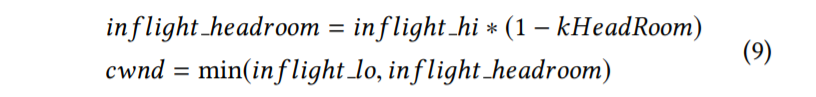
*cwnd* is also not set as 2 ∗ *BDP* and is related with *inflight lo and inflight hi*.

1. *inflight hi* is updated if the inflight packets are too high (inflight too high), in which the loss packet rate exceeds *loss threshold* in last round.
2. In the probe down phase, the pacing gain is 0.75, the phase will be switched to probe cruise if the inflight packets are drained to BDP or the condition inflight too high holds true.

Does loss/ECN rate for this sample say inflight is "too high"? This is used by both the bbr\_check\_loss\_too\_high\_in\_startup() function, which can be used in either v1 or v2, and the PROBE\_UP phase of v2, which uses it to notice when loss/ECN rates suggest inflight is too high. /

/ 이 샘플의 손실/ECN 비율이 인플리트가 "너무 높음"이라고 하는가? 이는 v1 또는 v2에서 사용할 수 있는 bbr\_check\_loss\_too\_high\_in\_startup() 함수와 PROBR\_에서 모두 사용된다.v2의 UP 페이즈(UP phase)를 사용하여 손실/ECN 비율이 너무 높음을 시사하는 경우를 알 수 있다. /

1. The *cwnd* in probe cruise is calculated by Equation (9). *kHeadRoom* is 0.15. *inflight hi* indicates the channel is in dangerous area. To leave headroom to cwndis to alleviate link congestion to some extent. The interval for **probe cruise is randomized from 2 seconds to 3 seconds**. If duration in probe cruise phase exceeds the *interval*, a probe refill phase is applied as shown in Figure 3. *cwnd* is set as *inflight hi* to increase the inflight packets in a round. The probe refill is to make preparation for probe up.



**PROBE UP**

In probe up phase, cwnd is increases exponentially per round:1, 2 ,4, 8. It makes a fast probe to if extra bandwidth available. Once lost bytes are too much, probe down phase is applied to get rid of excess queue, as the red ring shows in Figure 3. Algorithm 2 is to exponentially increase in flight hi per round. In ProbeRTT, cwnd is reduced by half in v2 to remedy the throughput variation.

프로브 업 단계에서 cwnd는 라운드당 기하급수적으로 증가한다. 1, 2,4,8. 추가 대역폭을 사용할 수 있는 경우 빠른 프로브를 만든다. 일단 손실 바이트가 너무 많으면, 빨간색 링이 그림 3에 나타낸 것처럼 초과 대기열을 제거하기 위해 프로브 다운 단계가 적용된다. 알고리즘 2는 라운드당 비행 hi를 기하급수적으로 증가시키는 것이다. ProbeRTT에서는 처리량 변화를 개선하기 위해 v2에서 cwnd를 절반으로 줄인다.

PROBE\_BW "UP" phase:

- Probe for bandwidth and volumetric capacity

- Grow beyond inflight\_hi slowly at first, then rapidly

- inflight\_target = inflight\_hi + inflight\_probe

- inflight\_probe grows exponentially per round trip:

- 1, 2, 4, 8... packets

- Set inflight\_hi ceiling to estimated max safe in-flight volume if:

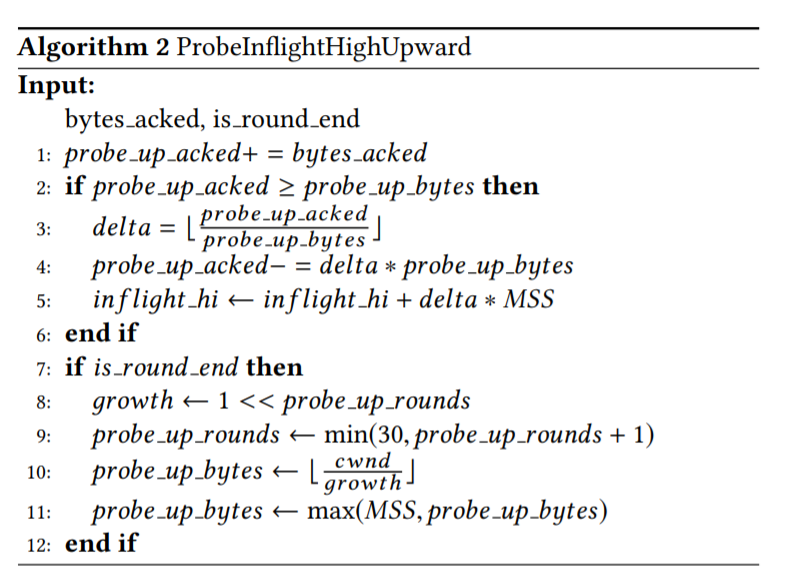
- Loss rate is too high: loss\_rate > loss\_ceiling (1%)

- Filtered ECN rate signal is too high

- Terminate probing upon any of:

- Estimated queue is too high (inflight > 1.25 \* estimated\_bdp)

- Hit inflight\_hi loss or ECN ceiling



/ After probing in PROBE\_UP, we have usually accumulated some data in the bottleneck buffer (if bw probing didn't find more bw). We next enter PROBE\_DOWN to try to drain any excess data from the queue. To do this, we use a pacing\_gain < 1.0. We hold this pacing gain until our inflight is less then that target cruising point, which is the minimum of (a) the amount needed to leave headroom, and (b) the estimated BDP. Once inflight falls to match the target, we estimate the queue is drained; persisting would underutilize the pipe. /

/ PROBE\_UP에서 프로빙을 한 후, 우리는 보통 병목현상 버퍼에 약간의 데이터를 축적해 왔다(bw 프로빙이 더 많은 bw를 찾지 못한 경우). 다음으로 PROBE\_DOWN으로 들어가 여분의 데이터를 대기열에서 빼내려고 한다. 이를 위해 페이싱\_게인 < 1.0>을 사용한다. 우리는 이 페이싱 이득을 (a) 머리 공간을 떠나는 데 필요한 최소 양인 목표 순항 지점과 (b) 추정 BDP보다 더 적을 때까지 보유한다. 일단 불이 목표물과 일치하게 되면, 우리는 대기열이 빠져나간다고 추정한다; 계속하면 파이프를 충분히 활용하지 못할 것이다. /

Start a new PROBE\_BW probing cycle of some wall clock length. Pick a wall clock time at which to probe beyond an inflight that we think to be safe. This will knowingly risk packet loss, so we want to do this rarely, to keep packet loss rates low. Also start a round-trip counter, to probe faster if we estimate a Reno flow at our BDP would probe faster.

일부 벽 클럭 길이의 새 PROBE\_BW 프로빙 사이클을 시작하십시오. 우리가 안전하다고 생각하는 기내 이상으로 탐사할 벽시계 시간을 선택하십시오. 이것은 고의로 패킷 손실을 초래할 것이기 때문에, 우리는 패킷 손실률을 낮게 유지하기 위해 이것을 거의 하지 않기를 원한다. 또한 왕복 카운터를 가동하여 BDP의 리노 흐름이 더 빨리 탐사될 것으로 추정할 경우 더 빨리 탐사하도록 하십시오.

<https://studywithus.tistory.com/archive/20181001>

<https://github.com/SoonyangZhang/DrainQueueCongestion>

BBR 2.0을 테스트할 수 있는 코드

수정해야할 사항

1. g++ version std=c++14
2. ubuntu 16.04
3. Proto\_time 코드 수정
4. traces 폴더 생성
5. src 안에 dqc 복사
6. 현재 버전 1.1