**연구 논문 제안서**

2020년 제 1 학기

|  |  |
| --- | --- |
| 제목 | BBR Congestion Control Algorithm 성능 개선 |
| GGitHub URL | <https://github.com/gatherheart/NS3-BBR> |

**작성자: 김현우**

**2020 년 03월 25일**

## **1. 과제의 필요성**

##### **1.1 Abstract**

기존까지 CUBIC, Reno, Vegas와 같은 많은 혼잡 제어 알고리즘은 패킷 손실과 지연을 통해 네트워크에 혼잡(Congestion)이 있음을 판단했다. 해당 알고리즘들은 패킷 손실과 지연(delay)이라는 두 가지 요소를 인식하고 혼잡 윈도우(Congestion Window)의 크기를 줄여 패킷 송신 속도를 제어한다. 이와 달리 구글이 2016년 제시한 BBR 혼잡 제어 알고리즘은 패킷 손실과 지연이 아닌 병목 대역폭(Bottleneck Bandwidth)과 왕복 전파 시간(Round-trip propagation time)을 통해 이용가능 한 BDP의 크기를 추정하고 송신 속도를 제어해 네트워크상의 혼잡을 방지한다. 구글은 이를 통해 이전의 알고리즘보다 전 세계적으로 약 4%정도 네트워크의 속도를 향상시킬 수 있다고 주장했다[1].

하지만 구글이 제시한 알고리즘은 네트워크의 속도 향상에 크게 기여할 것이라는 예상과 다르게 기존의 손실 기반 혼잡 제어 알고리즘(Loss based congestion control algorithm)과 같이 사용될 때 문제점이 발생했다. 기존의 알고리즘들은 대다수가 패킷 손실에 대해 민감하게 반응함으로써 네트워크의 혼잡성을 낮추는 반면, BBR은 패킷 손실을 네트워크의 혼잡 때문에 발생하는 것이라 여기지 않고 최대 대역폭과 최소 RTT(Round Trip Time)을 찾아내는 것에만 초점을 둔다. 이를 이용하여 네트워크의 사용 가능한 BDP를 추정하여 패킷 송신 속도를 유지하는 방식으로 네트워크의 성능 향상을 이끌어 낸다. 하지만 이런 방식은, 이용할 수 있는 BDP를 추정하고 이용하는 데에는 크게 효과적이지만 이전의 알고리즘들과 공정성 면에서 타협되지 않은 방법이기 때문에 문제가 발생한다. 따라서 본 논문에서는 BBR로 인해 네트워크상에서 발생하는 TCP 공정성(TCP Fairness) 문제를 파악하고 이를 해결하는 수정된 알고리즘을 제시하고자 한다.

새로운 알고리즘은 BBR의 방식 그대로 병목 대역폭과 왕복 전파 시간을 이용해 네트워크 대역폭 이용 효율을 높이며, 손실 기반(Loss based) 알고리즘과도 타협할 수 있는 방식으로 보완하여 네트워크의 안정성을 높이도록 한다. 이를 통해 BBR이 손실 기반 혼잡 제어 알고리즘과 같이 사용되더라도 네트워크상에서 서로 공정하게 작동할 수 있을 것으로 기대한다.

##### **1.2 Introduction**

1.2.1 연구 배경 및 목적

1980년 개발된 TCP 혼잡 제어 알고리즘들은 패킷 손실이 곧 네트워크 혼잡(Congesion)이라 가정하였다. 네트워크 대역폭이 전체적으로 낮았던 당시 상황으로는 대다수의 패킷 손실은 네트워크의 낮은 대역폭으로 발생하였고 해당 가정은 실제 네트워크 상황과 일치하였다. 하지만 기술이 발전하여 네트워크 대역폭은 매년 약 50%씩 증가하여 하이엔드 유저(High-end user)의 300bps 대역폭은 325Mbps까지 증가했으며 NIC(Network Interface Controller)는 Mbps 단위를 넘어 Gbps를 지원하게 되었다. 이렇게 발전한 현 네트워크 상황을 비추어볼 때, 패킷 손실이 낮은 대역폭 때문에 발생한다는 가정은 더 이상 성립하지 않게 되었다. 즉, 현재 네트워크 상황에서는 패킷 손실과 네트워크 혼잡의 관련성은 많이 떨어졌다고 볼 수 있다[1-2].

이 때문에 패킷 손실이 곧 네트워크의 혼잡의 결과라고 판단하는 손실 기반(Loss based) 알고리즘의 경우 네트워크의 큰 대역폭을 제대로 활용하지 못하고 속도를 낮게 제어하는 경향이 크다. 더군다나 손실 기반 알고리즘들은 네트워크가 이미 혼잡해져 패킷 손실이 발생한 후에야 속도 제어를 하기 때문에 네트워크에 무리를 주는 방법이라 할 수 있다[3]. 이를 해결하기 위해 BBR은 패킷 손실을 무시하면서, 정해진 패턴에 따라 송신 속도를 높이고 줄이며 최적의 BDP(Bandwidth delay product)를 찾는 데 초점을 둔다. 이런 방식을 통해 BBR은 패킷 손실이 곧 네트워크의 혼잡성을 나타내지 않는 현 네트워크 상황에 적용되어 이용가능한 BDP를 찾는데 효과적이었고, 그 결과 네트워크를 향상시킬 수 있었다.

하지만 3,4 년이 지난 이후 현재 네트워크에서는 BBR의 공격적인 알고리즘 때문에 크게 다섯 가지 문제가 나타나기 시작했다. 첫번째 문제점은 버퍼가 작은 라우터에서는 BBR이 적합하지 않다는 점이다. BBR은 startup 단계에서 pacing\_gain과 cwnd를 배 만큼 설정하여 RTTs 시간으로 BtlBw(Bottleneck Bandwidth)를 찾는다. 그에 따른 결과로 약 2BDP의 추가적인 버퍼를 형성하게 된다[1]. 하지만 버퍼가 작아서 추가적인 1.5 BDP를 유지할 수 없을 경우에는 패킷 손실이 증가하고 결국 BBR의 Goodput이 저하된다[5]. 즉, BBR은 작은 버퍼를 고려하지 않은 문제가 있다.

두번째, RTT가 송신 과정 중 증가하는 경우 문제가 발생한다. BBR은 ProbeBW 단계에서 이전 ProbeRTT 단계에서 찾은 RTprop (minimum RTT)을 이용한다. 즉, ProbeBW 단계에 진입한 이후 RTporp가 변경되는 경우 ProbeBW 단계가 진행되는 10초간 BDP를 과대평가(Overestimate)하는 경향이 있다[5]. 사실 RTT가 급격하게 변하는 네트워크 상황은 일반적이지 않기 때문에 무시할 수도 있다. 하지만 문제의 초점은 BBR이 네트워크가 변화하는 상황에 유연하지 못하다는 점에 있다.

세번째 문제도 이러한 BBR이 변화에 유연하지 못한 점과 관련되어 있다. BBR은 ProbeRTT 단계에서 이전 단계에서 형성된 버퍼를 비우며 RTprop(minimum RTT)를 찾는다. 이 때, 다른 BBR flow들과 동기화(Synchronization)가 이루어져 네트워크 대역폭을 공정(Fair)하게 이용해야 한다. 하지만 새로운 flow가 참여하는 시기에 따라 그 동기화가 어려워질 수 있다. 동기화 직후에 새로운 flow가 진입하는 경우, 다른 flow가 ProbeBW 단계를 거치는 동안 새로운 flow는 Startup 단계에 진입하게 된다. 이 경우 새로운 flow의 Startup 단계의 공격적인 탐색 방법으로 네트워크의 혼잡이 발생하고 이전의 flow들은 10개의 표본 패킷을 통해 BtlBwFilter를 새롭게 업데이트 하기 전까지, 네트워크 혼잡을 해결하지 못한 채 잘못 예측된 BDP를 계속 사용하게 된다. 이후 샘플 패킷을 통해 BtlBwFilter를 업데이트 한 이후에도 기존의 flow들은 이용가능한 BDP를 낮게 설정하고 동기화를 위한 ProbeRTT 단계까지 낮은 송신 속도를 사용한다. 즉, BBR은 새로운 flow가 진입하는 시점에 따라 유동적으로 대처하지 못하는 문제점이 있다[5, Fig. 14].

네번째, 동기화가 이루어지더라도 ProbeRTT 단계에 진입하고 나가는 시점이 미세하게 달라 생기는 문제점이 있다. ProbeRTT 단계는 RTprop이 10초 동안 갱신되지 않을 경우 시작된다. 즉, BBR은 RTprop이 갱신된 이후 10초 뒤에 ProbeRTT에 진입하게 되는데 이 RTprop이 만료되는 시점이 완벽하게 일치할 수는 없다. 한 flow가 조금이라도 ProbeRTT 단계에 늦게 진입하게 되면, 이미 ProbeRTT 단계에 진입한 flow들은 버퍼에 존재하는 패킷때문에 RTprop를 정상적으로 얻지 못하게 된다. 이 경우 BDP를 과대평가하게 되어 네트워크 혼잡을 유발할 수 있는 문제점이 발생한다[5].

마지막으로, RTT가 다른 flow끼리 경쟁할 경우 문제가 발생한다. RTT가 작은 경우 RTprop도 작을 수밖에 없고 이에 따라 BBR은 BDP를 상대적으로 작게 예측한다. 작은 BDP를 예측 값으로 가질 경우 네트워크 리소스를 공유할 때 상대적으로 적은 리소스를 이용할 수밖에 없는데 이는 두 가지 원인에서 비롯된다. 첫 번째로 BDP의 작은 예측 값이 그 원인이다. RTT가 다른 flow가 동기화 되어 ProbeBW 단계에 동시에 진입하였을 때, RTT가 작은 flow는 작은 BDP 예측 값을 갖기 때문에 BBR의 알고리즘에 따라 패킷 송신양이 RTT가 큰 flow에 비해 상대적으로 적다. 큰 BDP 예측값을 가진 flow에 의해 병목현상이 발생하는 라우터의 버퍼가 빠르게 차게 되고 작은 BDP 예측값을 가진 flow는 이용 가능한 버퍼 크기가 줄어들어 BtlBwFilter(Available Maximum Bandwidth)를 더 높은 값으로 갱신하기 힘들어지는 것이다. 두 번째로는 Probe 상태의 지속 시간 차이 때문이다. BBR은 최소 200ms 이상의 RTT 시간 동안 ProbeRTT 단계를 진행하는데, 큰 RTT를 가진 flow의 경우 ProbeRTT 단계가 길 수 밖에 없다. 작은 RTT를 가진 flow는 먼저 ProbeRTT를 종료하게 되고 다시 BDP를 채우게 된다. 이 경우 버퍼가 제대로 비워지지 못해 큰 RTT를 가진 flow는 정확한 RTprop(Minimum RTT)를 발견하지 못하고 더 긴 RTT를 갖게 되어 문제가 악화되는 것이다. ProbeBW 단계에서도 ProbeRTT 단계와 마찬가지로 RTT의 차이 때문에 문제가 발생한다. ProbeBW에서는 pacing rate를 증가시켰다 줄이며 BtlBW을 찾는데 이 찾는 시간은 각 flow의 RTT만큼 이루어진다. 즉, RTT가 길수록 ProbeBW에서 pacing rate를 증가시키는 시간이 길어지므로 더 네트워크 리소스를 더 독점적으로 이용하게 되는 것이다[7].

이러한 문제들은 모두 BBR의 초기의 정적인 설계때문에 나타나는 문제들이다. 이를 해결하기 위해 많은 연구가 이루어지고 있고 상황에 따라 유동적으로 대처할 수 있는 BBR v2, BBRPlus 등 다양한 알고리즘으로 더 발전되어가고 있다. 본 논문에서는 ns-3 시뮬레이터를 통해 언급된 BBR의 문제점을 파악하고 BBR을 네트워크 상황에 좀 더 적합한 알고리즘으로 발전시켜 BBR의 성능을 유지하며 다른 TCP 흐름(TCP flows)들과의 공정성을 유지하는 결과를 도출하도록 한다.

## **2. 선행연구 및 기술현황**

##### **2.1 BBRPlus**

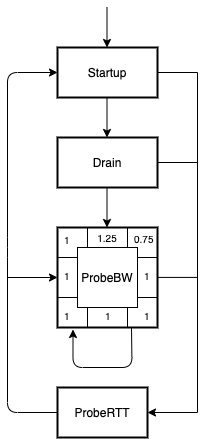


Figure 1: A set of states in BBR

기존 BBR의 경우 ProbeBW를 진행할 때, BtlBw를 찾기 위해 [1.25, 0.75, 1, 1, 1, 1, 1, 1]배 만큼씩 pacing rate를 조절하는 8개 단계를 반복한다. 이렇게 고정적인 방법으로는 변동성이 많은 네트워크 상황에 유동적으로 대처할 수 없다. 이를 해결하기 위해 BBRPlus는 반복되는 사이클의 주기를 랜덤하게 설정한다. 사이클의 주기가 랜덤하게 적용됨으로써 flow마다 pacing rate를 1.25배 만큼 높이는 시점은 서로 달라지고 RTT가 긴 flow라 해서 네트워크 리소스를 독점적으로 사용하는 것을 어느정도 방지할 수 있게 되는 것이다. 이를 통해 flow간 공정성을 높이고 패킷 손실을 줄일 수 있게 된다[4].

##### **2.2 BBR v2**

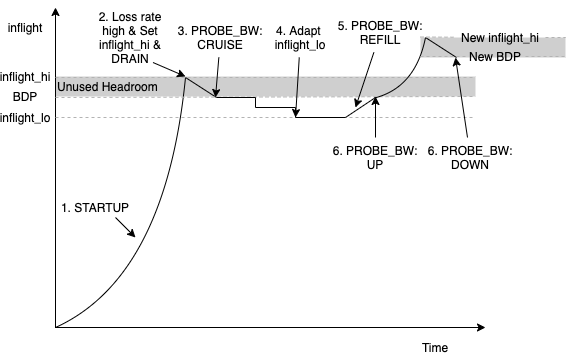


Figure 2: BBR v2 flow cycle, from [6]

기존 BBR의 경우 버퍼가 작은 경우에 패킷 손실이 많이 발생하는 문제, 그리고 Reno/CUBIC flow들과 경쟁하는 경우에 독점적으로 네트워크 리소스를 독점하는 문제가 발생했다. 이 문제를 해결하기 위해 수정된 BBR v2는 ProbeBW 단계에서 BBR v1과 다른 방법을 사용한다. BBR v2에서는 inflight data의 범위인 inflight\_hi와 inflight\_lo를 설정하여 BDP를 동적으로 예측하게 된다. 이를 통해 Reno/CUBIC flow와 경쟁할 때에도 공정성을 유지하며, 패킷 손실과 ECN 헤더를 이용해 이전 보다 더 유동적으로 네트워크 혼잡에 대처 하게 된다. 또한 ProbeRTT 단계에서 cwnd를 4개의 패킷으로 설정하지 않고 BDP의 0.5배만큼 설정함으로써 ProbeRTT 단계에서 처리량이 매우 줄어드는 문제를 해결하였다[6].

## **3. 작품/논문 전체 진행계획 및 구성**

3.1 ns3 실험 환경 구축

논문에서 나타난 BBR의 문제점을 파악하고 새로운 해결책을 찾기 위해 ns3 기반으로 BBR 알고리즘의 테스트 환경을 갖출 생각입니다. 현재까지는 구글이 제시한 BBR의 결과를 재생산 할 수 있는 ns3 BBR 코드를 찾았으며 해당 코드를 리뷰하여 전체적인 알고리즘을 이해하였습니다. 해당 코드를 기반으로 수정하여 BBR의 문제점을 파악 및 해결책을 적용해볼 생각입니다.

3. 2 결과물 재생산을 통한 증명

ns3 환경에서 사용할 M. Claypool의 ns3 BBR 코드[[1]](#footnote-0)가 제대로 구글의 결과를 재생산하는 지 여부를 정확히 기록해 둘 예정입니다. 해당 실험을 진행하기 이전에 해당 코드가 제대로 구글이 제시한 BBR의 성능을 낼 수 있는 지 여부를 확인하기 위해 구글이 진행했던 BBR의 실험 환경과 동일한 시뮬레이션 환경을 구축하였습니다. 현재는 구글이 진행했던 실험과 같은 조건으로 시뮬레이션한 결과, M. Claypool의 ns3 BBR 코드는 동일한 성능을 보였습니다.

3.3 문제점 발견

해당 ns3 BBR 코드 기반으로 이제 문제점이 나타나는 시뮬레이션 환경을 구축할 생각입니다. 해당 시뮬레이션을 통해 BBR의 문제점을 지적했던 논문[[2]](#footnote-1)의 결과를 재생산하여 확인할 생각입니다.

3.4 문제점 해결 실험

해당 문제점을 해결하는 방법을 고안하여 BBR 코드를 수정하고 문제점이 나타나는 시뮬레이션 환경에 적용해볼 생각입니다. BBRPlus와 BBR v2 모두 BBR v1이 가진 정적인 문제를 해결하는 방향이었고 저 역시 BBR v1이 급변하는 네트워크에 유동적으로 대처하지 못하는 점에 초점을 두고 BBR v1의 문제를 해결하는 방안을 찾아볼 예정입니다.

3.5 해결책 제시

여러 방안 중 제일 좋은 결과를 내는 모델을 이용하여 논문을 작성할 생각입니다. 수정된 BBR이 Reno/CUBIC 알고리즘과도 공정하게 작동할 수 있으며 네트워크 상황에 유동적으로 대처할 수 있음을 보이고 싶습니다. 현재는 룰베이스의 알고리즘을 생각하고 있습니다. 만약 가능하다면 머신러닝을 적용하여 문제를 해결해보고도 싶습니다.

## 

## **4. 기대효과 및 개선방향**

4.1 개선방향: BBR의 공정성 문제 해결

현재 구글은 BBR을 적용하여 전체적인 네트워크 성능 향상을 가져왔습니다. 하지만 다른 알고리즘들과 같이 사용될 때 공정성과 관련하여 큰 문제가 발생했습니다. 이 밖에도 알고리즘에서 pacing\_gain이나 probe duration 등 고정된 값을 사용하여 유동적인 네트워크 상황에 적합하지 않은 문제도 존재했습니다. 이러한 BBR의 고정적인 부분을 조금 더 유동적인 값을 사용하여 특정한 상황에 따라 다르게 반응하는 알고리즘으로 개선하고 싶습니다.

4.2 기대효과: 네트워크 성능 향상

구글은 BBR을 표준으로 지정하기 위해 많은 노력을 기울이고 있습니다. 하지만 BBR의 문제점도 여럿 나타난 상황에서 표준으로 받아들여지기에는 무리가 있다고 생각합니다. 구글은 BBR v2를 제시하였지만 아직 관심을 많이 받지 못하고 있고 그 성능이 아직 제대로 증명되지 않았습니다. 제가 BBR v2보다 나은 성능의 새로운 알고리즘을 제시하는 것은 불가능하다고 생각합니다. 다만 BBRPlus처럼 간단한 아이디어이지만 상황에 유동적으로 대처할 수 있는 방법을 제시하여 BBR이 표준으로 받아들여지는 데 도움이 되는 방향으로 이끌고 싶습니다. 이를 통해서 수정된 BBR이 혼잡 제어 알고리즘의 표준으로 지정되어 네트워크의 성능 향상을 유지하면서 유동적으로 네트워크 상황에 적합하게 작동해 네트워크의 전체적인 성능 향상을 기대합니다.

## **5. 기타**

5.1 비용 분석

서버를 이용해 ns3를 설치해 실험을 할 예정입니다. 서버는 주로 교수님이 제공해주신 연구실 서버와 AWS의 Elsatic Computing Instance를 필요할 때만 이용할 예정이라 비용지출은 크지 않을 것으로 예상합니다.

## **6. 참고문헌**

[1] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, “BBR: Congestion-based Congestion Control,” ACM Queue, vol. 14, no. 5, 2016.

[2] Nielsen , J. (2019, September 27). Nielsen's Law of Internet Bandwidth. Retrieved March 16, 2020, from https://www.nngroup.com/articles/law-of-bandwidth/

[3] K. Thompson, G. J. Miler, and R. Wilder, “Wide-area Internet traffic patterns and characteristics ( extended version),” IEEENetworkMagazine, November 1997.

[4] Songyang Zhang. An Evaluation of BBR and its variants. arXiv preprint arXiv:1909.03673, 2019.

[5] Dominik Scholz, Benedikt Jaeger, Lukas Schwaighofer, Daniel Raumer, Fabien Geyer, and Georg Carle. 2018. Towards a Deeper Understanding of TCP BBR Congestion Control. In IFIP Networking 2018. Zurich, Switzerland

[6] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, V. Jacobson, I. Swett, J. Iyengar, and V. Vasiliev, “BBR Congestion Control: IETF 100 Update: BBR in shallow buffers,” IETF 100, 2017, Presentation Slides. [Online]. Available: https://datatracker.ietf.org/meeting/100/materials/slides-100-iccrg-a-quick-bbr-update-bbr-in-shallow-buffers/

[7] S. Ma, J. Jiang, W. Wang, and B. Li, “Towards RTT Fairness of Congestion-Based Congestion Control,” CoRR, vol. abs/1706.09115, 2017. [Online]. Available: <http://arxiv.org/abs/1706.09115>

1. <http://web.cs.wpi.edu/~claypool/papers/bbr-prime/> [↑](#footnote-ref-0)
2. [4], [5]번 논문에 해당합니다. [↑](#footnote-ref-1)