논문 19-44-11-17 The Journal of Korean Institute of Communications and Information Sciences '19-11 Vol.44 No.11
https://doi.org/10.7840/kics.2019.44.11.2102

TCP 혼잡제어 알고리즘 간의 성능 평가

송 영 준◆, 김 건 환*, 조 유 제°

Performance Evaluation between TCP Congestion Control Algorithms

Yeong-Jun Song◆, Geon-Hwan Kim*, You-Ze Cho°

요 약

현재 인터넷 상에서 이용되는 웹 응용들의 트래픽 중 90% 이상이 전송계층 프로토콜로 TCP를 사용하고 있다. TCP의 송신측은 신뢰성 보장 및 과도한 혼잡을 막기 위해 혼잡제어를 수행한다. 오늘날 전송기술과 네트워크 장비의 발전으로 인해 빠른 속도의 네트워크에서 Reno, NewReno와 같은 기존의 손실기반 혼잡제어 알고리즘은 이용 가능한 자원을 완전히 이용하지 못하는 문제점이 발생하였으며, 구글은 빠른 속도와 긴 지연시간을 가진 오늘날의 네트워크에서 최적의 동작을 위해 기존과 다른 개념의 BBR (Bottleneck Bandwidth Round-trip propagation time) 혼잡제어 알고리즘을 발표했다. 본 논문에서는 BBR을 포함한 다양한 혼잡제어 알고리즘을 동작 특성에 따라 분류하고, 테스트베드를 구성하여 각 알고리즘을 사용하는 다수의 플로우 간의 상호작용 및 성능을 평가하였다.

Key Words : TCP congestion control, Reno, CUBIC, Vegas, NewVegas, BBR, Veno

ABSTRACT

More than 90% of the traffic of web applications on the Internet is currently using the TCP as the transport layer protocol. The TCP sender performs congestion control to ensure reliability and prevent excessive congestion. Today, due to the development of transmission technology and network equipment, the existing loss-based congestion control algorithms such as Reno, NewReno cannot fully utilize the available resources in the high-speed networks. Google introduced BBR (Bottleneck Bandwidth Round-trip propagation time), a new concept of congestion control algorithm for optimal operation in today's network with high speed and long latency. In this paper, we classify various congestion control algorithms including BBR according to thier operation characteristics and evaluate the interaction and performance between multiple flows using each algorithm on a testbed.

※ This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2018R1A6A1A03025109).
※ This research was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (NRF-2019R1A2C1006249).
◆ First Author : School of Electronics Engineering, Kyungpook National University, syj5385@knu.ac.kr, 학생회원
° Corresponding Author : School of Electronics Engineering, Kyungpook National University, yzcho@ee.knu.ac.kr, 종신회원
* School of Electronics Engineering, Kyungpook National University, kgh76@ee.knu.ac.kr, 학생회원
논문번호 : 201908-155-B-RN, Received August 12, 2019; Revised September 5, 2019; Accepted September 5, 2019

Ⅰ. 서 론

현재 인터넷 상에서 90% 이상의 트래픽들이 신뢰성 있는 데이터 전송을 위해 전송계층 프로토콜로 TCP를 사용하고 있다[1]. TCP의 주요 기능 중 하나인 혼잡제어는 과도한 네트워크 트래픽으로 부터 인터넷을 보호하고 다수의 호스트가 제한된 네트워크 자원인 대역폭을 공평하게 사용하게 한다[2,3]. TCP가 1980년대 부터 사용해오고 있는 Tahoe, Reno 및 NewReno 등의 혼잡제어 알고리즘은 기본적으로 전송량을 지속적으로 증가시킨 후 발생하는 패킷 손실을 혼잡의 신호로 이용한다. 이전의 낮은 대역폭 및 작은 병목링크 버퍼의 환경에서 기존 손실기반 혼잡제어 알고리즘은 비교적 성능상의 문제없이 잘 동작하여왔다. 그러나, 오늘날 스위치와 라우터 등의 네트워크 장비와 전송기술의 지속적인 발전으로 1990년대에 네트워크 전송용량이 1000배 이상 향상된 현재의 인터넷에서 기존의 손실기반 혼잡제어 알고리즘은 버퍼에 가득한 패킷에 의한 대기 지연과 패킷 손실에 의한 재전송으로 인해 시간당 어플리케이션 영역 데이터 처리율 (goodput) 저하의 심각한 문제점이 발생한다. 기존 손실기반 알고리즘과 달리 1991년에 제안된 Vegas는 초과 대기열에 의한 대기 지연을 혼잡의 신호로 인식하는 대표적 지연기반 혼잡제어 알고리즘으로 초과 대기열의 생성을 억제하며 동작하여 패킷 손실이 발생하지 않고 지연시간이 낮아 goodput이 높은 장점이 있다[4]. 하지만 크로스 트래픽, 지연응답 (Delayed ACK), 일정 이상의 긴 지연시간 등을 원인으로 하는 심각한 성능 저하 문제점이 발생하였다[5].

현재의 빠른 속도와 긴 지연시간을 갖는 네트워크 환경에서 최적의 동작을 위해 2016년 말 구글은 새로운 개념의 혼잡제어 알고리즘인 BBR을 발표했다. TCP 초기의 개발진의 한 명이었으며 현재는 BBR의 개발을 이끄는 구글의 수석 엔지니어인 Van Jacobson은 “TCP가 패킷 손실이 발생할 때만 트래픽의 속도를 낮춘다면 대응하기에 이미 너무 늦은 것이다. BBR은 손실 같은 혼잡이 발생하기를 기다리지 않고 전송 파이프를 모델링한다. 마치 파이프의 길이나 지름을 재는 것처럼 얼마나 많은 데이터를 담을 수 있는지를 미리 결정하는 것이다.” 라고 말했다[6-7]. BBR은 주기적으로 이용 가능한 최대 대역폭의 크기와 전파 지연시간을 검색하여 최대 전송 속도 및 낮은 지연시간을 보장하며 데이터를 전송하는 특성으로 인해 많은 주목을 받고 있으며 BBR과 관련된 다양한 연구가 진행되고 있다[8-10].

본 논문에서는 BBR을 포함하여 다양한 혼잡제어 알고리즘을 동작 특성에 따라 분류하고 물리적 테스트베드를 구현하여 각 알고리즘을 사용하는 다수의 플로우 간의 상호작용 및 성능을 평가하였다. 본 논문의 실험 및 결과는 다양한 네트워크 환경에서 알고리즘 간 상호 성능이 중요한 요소로 작용함을 나타내며, 새로운 혼잡제어 알고리즘 설계 시 알고리즘 간의 상호 성능을 평가하기 위한 실험 환경 구성 및 다양한 환경에서의 평가 요소 및 방법을 확인할 수 있다.

본 논문의 구성은 다음과 같다. II장에서 혼잡제어 알고리즘에 대해 간략히 설명 후 동작 특성에 따라 혼잡제어 알고리즘을 분류한다. III장에서는 분류된 각 알고리즘 그룹의 대표적 혼잡제어 알고리즘의 성능을 평가하고 다수의 플로우 간의 상호작용에 대한 실험 결과를 소개한다.

Ⅱ. TCP 혼잡제어 알고리즘

네트워크 혼잡은 송신 호스트로 부터 전달된 데이터의 양이 네트워크 링크가 수용할 수 있는 양을 초과하여 대기열이 생성되거나 패킷 손실이 발생하는 상태를 의미한다. 혼잡 상태는 병목링크 버퍼에 존재하는 패킷들의 대기 지연으로 인한 지연발생, 손실된 패킷의 재전송으로 인한 goodput 저하 등 성능상의 문제점을 일으킨다. 그러므로 각각의 송신 호스트는 네트워크 혼잡을 피하기 위해 과도한 양의 데이터를 전달하지 않아야 하며 다수의 호스트가 공유하는 대역폭을 특정 호스트가 과도하게 점유하지 않아야 한다. TCP 혼잡제어는 위의 동작 목표를 달성하기 위해 링크의 혼잡상태를 인식하여 특정 시점에 내보낼 데이터의 양을 결정한다. TCP 혼잡 윈도우를 계산하는 방법에 따라 그림 1과 같이 손실기반, 지연기반, 하이브리드 및 모델기반 혼잡제어 알고리즘으로 분류된다.

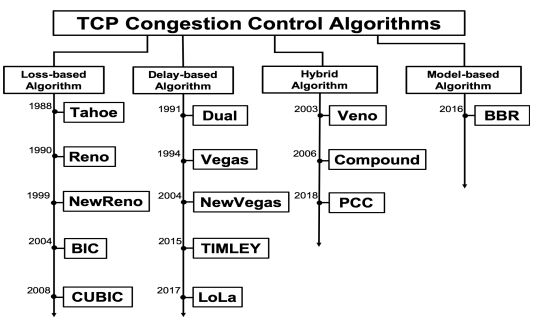


그림 1. TCP 혼잡제어 알고리즘 분류
Fig. 1. TCP congestion control algorithm classification

손실기반 알고리즘은 패킷손실을 혼잡을 판단하기 위한 지표로 사용한다. 송신측은 병목링크에서 패킷 손실이 발생할 때까지 지속적으로 전송률을 증가시켜 나가고, 패킷 손실이 발생하면 전송률을 낮추는 과정을 반복한다. 그림 2의(C)지점이 병목링크의 버퍼의 과부하로 인한 패킷 손실이 발생하는 지점으로 손실기반 알고리즘의 동작점이 된다. 지연기반 알고리즘은 버퍼의 생성으로 인한 대기 지연을 혼잡의 신호로 인식하는 알고리즘으로 각 알고리즘이 설정한 최대 경계값보다 긴 대기열이 생성되면 전송률을 낮춘다. 그림 2의 (A)지점이 대기열이 생성되기 시작하는 지점으로 각 혼잡제어 알고리즘의 최대 경계값에 따라 그림 2의 (B)구간 내에 지연기반 알고리즘의 동작점이 설정된다. 하이브리드 알고리즘은 패킷 손실과 대기지연을 모두 혼잡의 신호로 인식하는 혼잡제어 알고리즘으로 그림 2의 (C)지점에서 패킷 손실이 발생하기 전 대기열의 크기에 따라 혼잡 윈도우를 계산하는 메커니즘을 다르게 수행하는 알고리즘으로 그림 2의 (B)구간 내에서 동작한다. 마지막으로 모델기반 알고리즘인 BBR은 최대 데이터 처리율 및 최소 지연시간을 갖는 Kleinrock의 최적 동작점 (A)에서 동작하기 위해 이용 가능한 최대 대역폭의 크기와 종단 간 최소 지연시간을 측정하여 자신이 이용하고 있는 링크를 모델링함으로써 동적으로 내보내는 데이터의 양을 조절한다.

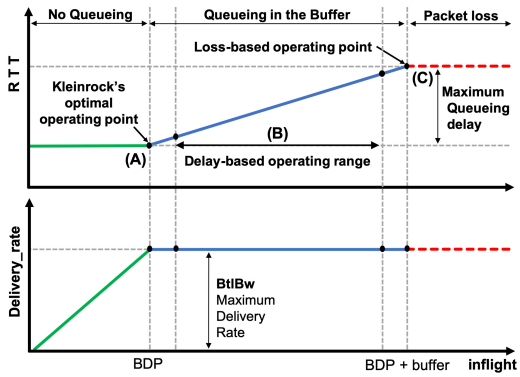


그림 2. 전달중인 데이터 양에 따른 RTT 및 전송속도
Fig. 2. RTT and throughput according to the inflight data

2.1 손실기반 알고리즘

손실기반 혼잡제어 알고리즘은 가장 처음 고안된 방식이며 대표적으로 Reno, Westwood[11], BIC[12] 및 CUBIC[13] 등이 있다. 각 세그먼트에 대한 ACK를 수신할 때마다 혼잡 윈도우 (cwnd) 크기를 지속적으로 증가시켜 대기열을 생성한다. 이때, 대기열의 크기가 버퍼 크기를 초과하면 버퍼 범람 (buffer overflow)으로 인한 패킷 손실이 발생하며, 이러한 패킷 손실을 혼잡 상황으로 판단하고 혼잡 윈도우의 크기를 줄여서 혼잡 상황을 회피한다.

2.1.1 Reno

Reno는 AIMD (Additive Increase Multiplicative Decrease)를 기반으로 동작한다. ACK를 수신할 때마다 혼잡 윈도우 크기를 2배씩 증가시켜 가용 대역폭 크기를 빠르게 검색하며, ssthresh (slow start threshold)의 상한선을 두어 혼잡 윈도우의 크기가 ssthresh 값에 도달하면 패킷 손실이 발생할 때까지 천천히 전송량을 늘리기 위해 혼잡 윈도우를 1씩 증가시키는 혼잡 회피를 수행한다. 이후 패킷 손실이 발생하면 아래 알고리즘 1과 같이 혼잡 윈도우를 줄여 혼잡 상황에서 벗어나며 자신이 이용할 수 있는 대역폭을 찾기 위해 빠른 회복 단계를 수행한다.

최근 네트워크 장비 발전에 의해 이용 가능한 대역폭의 크기가 증가한 반면, Reno의 느린 혼잡 회피 메커니즘은 이용 가능한 자원을 완전히 이용하지 못하게 하였으며, 이러한 문제점을 해결하기 위해 기존의 Reno에 비해 공격적인 혼잡 윈도우 증가 메커니즘을 가진 NewReno, HS-TCP[14], Westwood, BIC, CUBIC 등의 다양한 손실기반 혼잡제어 알고리즘이 제안되었다.

Algorithm 1. Reno 혼잡 윈도우 메커니즘

3-duplicated ACK

$$ssthresh = cwnd$$

$$cwnd = \frac{cwnd}{2}$$

Timeout

$$ssthresh = \frac{cwnd}{2}$$

$$cwnd = \text{ init cwnd}$$

Fast Recovery

$$cwnd = cwnd \times 2, \quad \text{if } cwnd < ssthresh$$

$$cwnd = cwnd + 1MSS, \quad \text{if } cwnd \geq ssthresh$$

2.1.2 BIC & CUBIC

기존 Reno의 느린 혼잡 윈도우 증가 메커니즘을 개선하기 위해 BIC (Binary Increase Congestion control) 알고리즘이 제안되었다. BIC는 이진 증가 접

근을 통해 도달하고자 하는 혼잡 윈도우 목표 지점으로부터 떨어진 정도에 따라 혼잡 윈도우의 증가량을 조절함으로써 패킷 손실을 줄인다. 하지만 BIC는 구현이 복잡할 뿐만 아니라 서로 다른 전파지연을 가진 다수의 플로우가 공존하는 경우 짧은 지연 시간을 갖는 플로우가 더 우세한 성능을 보이는 RTT 공평성 문제가 발생한다. 위의 문제점을 해결하기 위해 CUBIC 혼잡제어 알고리즘이 제안되었다.

CUBIC은 패킷 손실이 발생한 시점으로부터 경과된 시간에 따라 아래 식 (1)에서 표현된 3차 함수를 통해 혼잡 윈도우의 크기를 계산하는 RTT와 독립적인 알고리즘으로 BIC에 비해 RTT 공평성이 우수하다.

$$cwnd = C(t-k)^2 + W_{\max} \quad (1)$$

$$K = \sqrt[3]{\frac{W_{\max}\beta}{c}} \quad (2)$$

위의 수식에서 $cwnd$는 CUBIC의 혼잡 윈도우의 크기, t는 패킷 손실이 발생한 시점으로부터 경과 시간, $W_{\max}$는 패킷 손실이 발생하기 직전의 혼잡 윈도우의 크기를 의미하며, β와 c값은 패킷 손실이 발생하였을 때, 혼잡 윈도우의 증가 속도를 조절하는 역할을 한다.

2.2 지연기반 알고리즘

지연기반 혼잡제어 알고리즘은 초과 대기열 생성에 의한 대기 지연을 혼잡의 신호로 사용한다. 대기열의 생성 없이 동작하여 버퍼 범람에 의한 패킷 손실을 발생 시키지 않으므로 우수한 goodput성능을 보이며, Vegas, VegasA[15] 및 NewVegas 등이 대표적인 알고리즘이다. 지연기반 알고리즘은 각 세그먼트의 RTT를 통해 버퍼가 채워지는 시점을 예측하여 대기열 생성을 피하며 버퍼 범람에 의한 패킷 손실을 발생시키지 않으므로 손실 회피 혼잡제어 알고리즘으로도 불린다.

2.2.1 Vegas

Vegas는 대표적인 지연기반 혼잡제어 알고리즘으로 대기 지연을 혼잡의 신호로 인식한다. 아래 식 (3)을 이용하여 지속적으로 병목 링크 버퍼에 채워진 대기열의 실제 크기를 계산하여 미리 정의된 경계 값보다 작게 혼잡 윈도우의 크기를 조절한다.

$$N = \frac{cwnd}{RTT_c} \times (RTT_c - \text{baseRTT}) \quad (3)$$

N은 병목 링크 버퍼에 채워진 대기열의 크기, baseRTT는 측정된 RTT의 최소값, RTT_c는 현재의 RTT값을 의미한다. 위 식을 통해 계산된 대기열의 크기가 미리 정의된 α보다 작으면 혼잡하지 않다고 판단하여 혼잡 윈도우의 크기를 증가시키며, β보다 크면 네트워크가 혼잡한 것으로 판단하여 혼잡 윈도우의 크기를 감소시킨다.

하지만, Vegas의 RTT 추정 방식은 delayed-ACK, 크로스 트래픽, 다른 플로우에 의해 생성된 버퍼, 라우팅 변경 등에 의한 실제와 다른 RTT측정은 큰 성능 저하를 발생시킨다. 특히, 다른 혼잡제어 알고리즘과 병목 링크에서 경쟁할 때, 서로 다른 동작 특성에 의해 심각한 성능 불균형을 보일 수 있으며, 이에 대한 자세한 내용은 실험 결과에서 다룬다.

2.2.2 NewVegas

Vegas 혼잡제어 알고리즘은 매우 긴 지연시간을 갖는 링크에서 심각한 성능 저하가 나타날 뿐만 아니라 수신 호스트의 delayed-ACK 구현 및 크로스 트래픽이 존재할 때 실제와 다른 RTT 측정으로 인해 성능 저하가 나타난다. 이를 해결하기 위해 아래 세 가지의 동작이 추가된 NewVegas 혼잡제어 알고리즘이 제안되었다.

[Packet Pacing]: Slow start 구간에서 과도한 데이터전송으로 인해 발생하는 부정확한 RTT 추정을 피하기 위해 Slow start 구간 동안 전달되는 각 세그먼트 사이에 $\frac{RTT}{cwnd}$에 해당하는 지연을 추가한다.

[Packet Pairing]: 수신 호스트의 delayed-ACK 구현에 의해 발생하는 심각한 성능 변동을 피하기 위해 Slow start 구간에서 하나의 ACK당 1MSS씩 혼잡 윈도우가 증가하는 기존 방법이 아닌 두 개의 ACK를 받을 때 혼잡 윈도우의 크기를 2MSS씩 증가시킨다.

[Rapid Window Convergence]: 긴 RTT를 갖는 네트워크 환경에서 성능 저하가 발생하는 Vegas를 개선하기 위해 Slow start 동작이 종료 된 이후 Slow start 동작 보다는 느리고 congestion avoidance 보다는 빠른 혼잡 윈도우 증가 메커니즘을 추가되었다.

2.3 하이브리드 알고리즘

1) Veno

Veno[16]는 대표적인 하이브리드 혼잡제어 알고리즘으로 무선 네트워크에서 동작하기 위해 설계된 혼잡제어 알고리즘이다. 무선 네트워크에서는 유선 네트워크와 달리 버퍼 범람에 의한 패킷 손실뿐만 아니라 비트 에러에 의한 패킷 손실이 자주 발생하며, 이는 데이터 처리율을 저하 시키는 주요한 요인이 된다. Veno는 Vegas 알고리즘에서 사용된 버퍼 내의 대기열 크기를 측정하기 위한 식 (3)을 이용해 혼잡에 의한 패킷 손실과 비트 에러에 의한 패킷 손실을 구분하며 알고리즘 2와 같이 동작한다.

2.4 모델기반 알고리즘

2) BBR

BBR은 2016년 구글이 제안한 모델기반 혼잡제어 알고리즘으로 주기적으로 이용 가능한 최대 대역폭의 크기와 최소 지연 시간 ($RTT_{\min}$)으로 이루어진 네트워크 경로 모델을 통해 자신이 이용하고 있는 링크를 모델링하여 과도한 대기열의 생성 없이 데이터를 전송한다. BBR은 빠르게 이용 가능한 대역폭을 검색하기 위한 StartUp 동작과 네트워크 경로 모델을 구성하는 병목 링크 대역폭의 크기와 최소 지연 시간을 주기적으로 측정하는 steady-state 동작으로 구분된다.

Algorithm 2. Veno 혼잡 윈도우 메커니즘

<Increase Mechanism>

if $N < \beta$:

$$cwnd = cwnd + 1MSS, \text{ for each } RTT$$

else if $N > \beta$:

$$cwnd = cwnd + 1MSS, \text{ for each } 2RTT$$

<Decrease Mechanism>

if retransmission for timeout:

$$cwnd = \frac{cwnd}{2}$$

$$ssthresh = cwnd$$

else if retransmission for 3-duplicated ACK:

$$ssthresh = \begin{cases} cwnd \times \frac{4}{5}, & (N < \beta) \\ cwnd = \frac{cwnd}{2}, & (otherwise) \end{cases}$$

$$cwnd = \frac{cwnd}{2}$$

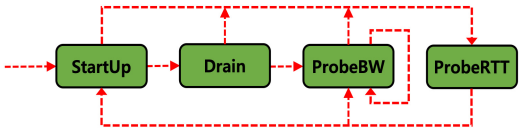


그림 3. BBR의 4가지 동작모드
Fig. 3. 4-operation mode of BBR

[StartUp Operation]: BBR은 ACK를 수신할 때마다 내보내는 데이터의 양을 약 2.89배 증가시켜 데이터 전송률을 빠르게 증가시킨다. 파이프가 가득 찬 이 후 대기열이 생성되며 더 이상 데이터 처리율이 증가하지 않으면 BBR은 StartUp 모드를 종료하고 생성된 버퍼를 제거하기 위해 Drain 모드를 수행한다.

[Steady-State Operation]: BBR은 대부분의 시간을 Probe BW 모드에서 동작하며 pacing_cycle을 통해 이용 가능한 대역폭의 크기를 주기적으로 검색한다.

또한 $RTT_{\min}$이 일정 시간 (기본 : 10초) 동안 갱신 되지 않은 경우 새로운 $RTT_{\min}$을 측정하기 위해 혼잡 윈도우의 크기를 4로 제한하여 일정시간 (기본 : 200ms + RTT) 동안 유지하는 Probe RTT 모드를 수행한다.

Ⅲ. 실험 및 평가

3.1 테스트베드 환경 구성

TCP 혼잡제어 알고리즘의 성능 및 서로 간의 상호작용을 평가하기 위하여 그림 4와 같이 테스트베드 실험 환경을 구성하였다. 총 3개의 송수신 호스트 (6-core Intel Core i5 3.6GHz CPU, 16GB 메모리)는 리눅스 커널 4.14 버전의 우분투에서 동작하며 라우터 (2-core Intel core Pentium and 4GB 메모리)는 리눅스 커널 4.9 버전의 우분투에서 동작한다. 또한 "ethtool"과 "netem"을 이용하여 100Mbps의 병목링

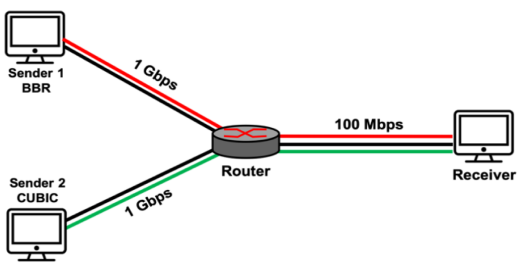


그림 4. 테스트베드 토폴로지
Fig. 4. Testbed Topoilogy

크, 병목링크 버퍼의 크기, 전파지연 시간을 구현하였으며, 두 개의 송신 호스트는 “iperf“[17] 을 통해 TCP 트래픽을 생성하여 일정 시간동안 수신 호스트에게 데이터를 전송한다.

각 실험에서 성능을 평가하기 위해 파이썬 기반의 오픈 소스 성능 측정 프로그램인 “TCPinfo“, “TCPlog“[18]를 이용하였으며, 리눅스 커널의 기본 TCP 파라미터 로깅 모듈인 “tcpprobe“를 필요에 따라 수정하여 사용하였다.

3.1.1 단일 플로우 성능 평가

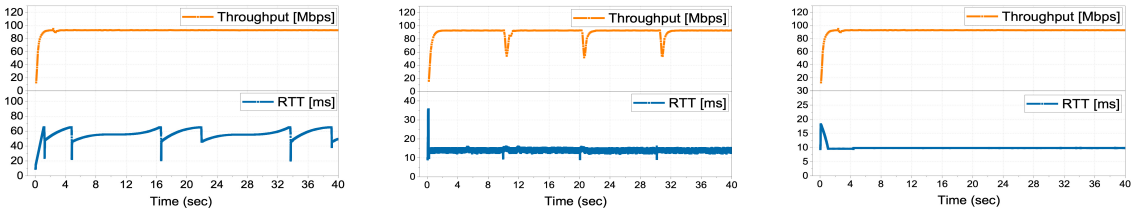
각 혼잡제어 알고리즘의 단일 플로우 성능을 평가하기 위해 그림 4의 호스트 1은 호스트 3에게 100초간 데이터를 전송한다. 송수신 호스트 간 RTT는 10ms로 설정하며, 버퍼의 크기가 큰 경우는 5BDP, 작은 경우는 0.8BDP로 설정하여 실험하였다. 각 실험에 대한 데이터 처리량, 재전송 패킷 수, 평균 혼잡 윈도우 크기 및 RTT 결과를 표 1에 나타내었으며, CUBIC, Vegas, BBR을 이용한 플로우의 0~40 초 구간 동안 시간에 따른 데이터 처리율과 RTT를 그림 5에 나타내었다.

BBR을 제외한 혼잡제어 알고리즘은 모두 92Mbps 이상의 데이터 처리율을 보였으며 주기적으로 수행되는 Probe RTT에 의해 BBR은 91.2Mbps로 약간 낮은 데이터 처리율을 보였다.

버퍼의 크기가 큰 경우, 손실기반 알고리즘을 제외한 모든 알고리즘은 혼잡이 발생하기 전에 전송하는 데이터의 양을 조절하거나 혼잡 윈도우의 크기를 매우 천천히 증가시키기 때문에 패킷 재전송이 일어나지 않았지만, 패킷 손실이 발생할 때까지 전송률을 증가시킨 후 손실이 발생한 후 전송률을 감소시키는 손실기반 알고리즘은 지연기반 및 하이브리드 알고리즘에 비해 높은 혼잡 윈도우 및 RTT를 보였으며, 버퍼 범람으로 인한 패킷 재전송이 발생하였다.

표 1. 단일 플로우 실험 결과
Table 1. Result of single flow experiment

	Group	algorithm	throughput [Mbps]	retransmission [packet]	Average cwnd [MSS]	average RTT [ms]
Large Buffer (5BDP)	loss-based	Reno	92.9	13	407	50.33
		CUBIC	92.9	50	455	56.15
	delay-based	Vegas	92.8	0	78	9.78
		NewVegas	92.8	0	116	14.39
	model-based	BBR	91.2	0	175	13.27
	Hybrid	Veno	92.9	0	376	46.37
Small Buffer (0.8BDP)	loss-based	Reno	92.5	176	124	15.49
		CUBIC	92.8	132	138	17.17
	delay-based	Vegas	92.8	0	78	9.72
		NewVegas	92.8	24	104	12.9
	model-based	BBR	91.2	253	174	13.16
	Hybrid	Veno	92.8	20	125	15.53



(a) CUBIC (b) Vegas (c) BBR

그림 5. 단일 플로우 처리량 및 RTT (Buffer size : 5BDP)
Fig. 5. Throughout and RTT of Single flow (Buffer size : 5BDP)

버퍼의 크기가 작은 경우, 손실기반 알고리즘의 버퍼 범람에 의한 패킷 손실 발생 주기가 짧아져 버퍼의 크기가 큰 경우에 비해 많은 재전송이 일어났다. 또한 모델기반 알고리즘은 BBR은 다른 알고리즘에 비해 재전송된 패킷 수가 많은 것을 확인할 수 있다. BBR은 계산된 BDP를 기반으로 최대 2BDP의 데이터를 내보내며, 이 때 생성될 수 있는 대기열의 최대 크기인 1BDP는 버퍼의 크기를 초과하여 과도한 패킷 손실이 발생한다. 뿐만 아니라 BBR은 패킷 손실이 발생한 이후 내보내는 데이터의 양을 줄이지 않아 지속적인 패킷 손실이 발생한다.

3.2 다중 플로우 성능 평가

3.2.1 알고리즘 간 성능 평가

TCP 혼잡제어 알고리즘 간 상호작용을 평가하기 위하여 호스트 1과 2는 동시에 호스트 3에게 150 초간 데이터를 전송한다. 호스트 1, 2와 호스트 3 간 RTT는 10 ms로 동일하게 설정하며, 버퍼의 크기는 단일 플로우 실험에서와 동일하게 설정한다. 두 플로우 간의 성능 측정 결과는 부록 1에 나타냈다. 두 플로우 간의 공평성을 평가하기 위한 Jain's Index[19] 는 아래의 식 (4)와 같이 계산된다.

$$F=\frac{1}{n}\times[\sum_{i=0}^{n}x_i]^2/\sum_{i=0}^{n}x_i^2 \quad (4)$$

위 식에서 n은 링크를 공유하는 플로우의 수, x_i는 i번째 플로우의 데이터 처리량을 의미한다. F 값은 1.0에 가까울수록 다수의 플로우가 대역폭을 공평하게 사용하고 있음을 나타낸다.

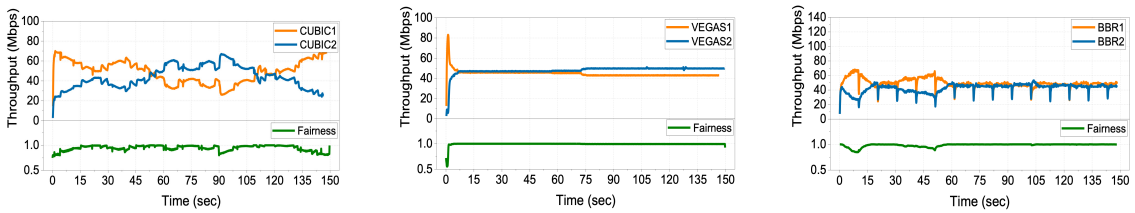
(1) 동일 그룹 내 알고리즘 간 성능 분석

[손실기반 알고리즘]: 부록의 표를 통해 손실기반 알고리즘을 이용하는 서로 다른 두 플로우는 0.9 이상의 높은 공평성을 보이는 것을 확인할 수 있으며, 특히 버퍼의 크기가 작은 경우 공평성이 더 높게 나타났다. 이는 버퍼의 크기가 클수록 각 알고리즘이 가지는 서로 다른 혼잡 윈도우 증감 메커니즘의 영향을 크게 받기 때문이다.

[지연기반 알고리즘]: 부록의 표를 통해 지연기반 알고리즘 그룹에서 동일한 알고리즘을 사용하는 경우 0.99 이상의 매우 높은 공평성을 보이지만, Vegas, NewVegas를 이용하는 서로 다른 두 플로우가 경쟁하는 경우 0.56의 매우 낮은 공평성을 보이는 것을 확인할 수 있다.

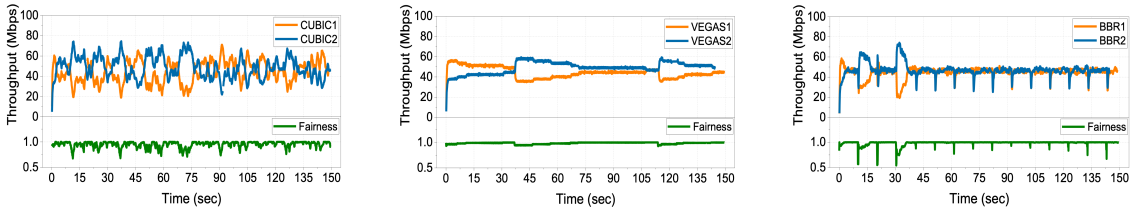
[하이브리드 알고리즘]: 부록의 표를 통해 두 플로우가 모두 하이브리드 알고리즘인 Veno를 사용하여 데이터를 전송하였을 때, 버퍼의 크기와 관계없이 0.99 이상의 매우 높은 공평성을 보였다.

[모델기반 알고리즘]: 그림 6-(c)와 그림 7-(c)를 통해 모델기반 알고리즘인 BBR을 이용하는 두 플로우가 공통의 병목링크에서 경쟁하는 경우 버퍼의 크



(a) CUBIC vs CUBIC (b) Vegas vs Vegas (c) BBR vs BBR

그림 6. 동일 그룹 내의 혼잡제어 알고리즘 간 상호작용 (Large Buffer: 5BDP)
Fig. 6. Interaction between congestion algorithm in the same group (Large Buffer: 5BDP)



(a) CUBIC vs CUBIC (b) Vegas vs Vegas (c) BBR vs BBR

그림 7. 동일 그룹 내의 혼잡제어 알고리즘 간 상호작용 (Small Buffer: 0.8BDP)
Fig. 7. Interaction between congestion algorithm in the same group (Small Buffer: 0.8BDP)

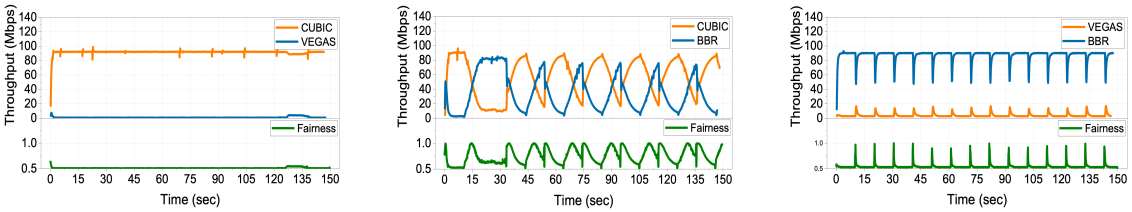
기와 관계없이 매우 우수한 공평성 성능을 보이는 것을 확인할 수 있다. 하지만, 버퍼의 크기가 작은 경우 두 플로우에 의해 생성되는 대기열의 크기가 버퍼의 크기를 초과하여 각각 32,794, 34,392개의 매우 과도한 패킷 재전송이 발생한 것을 부록의 표를 통해 확인할 수 있다.

(2) 서로 다른 그룹 내 알고리즘 간 성능 분석

[CUBIC vs Vegas]: CUBIC과 Vegas가 공통의 병목 링크에서 공존할 때, 버퍼의 크기와 관계없이 대부분의 대역폭을 CUBIC이 차지하는 것을 그림 8-(a)와 그림 9-(a)를 통해 확인할 수 있다. 손실기반 알고리즘인 CUBIC은 혼잡 윈도우의 크기를 지속적으로 증가시켜 버퍼를 채우는 반면에, 지연기반 알고리즘인 Vegas는 CUBIC에 의한 대기 지연을 네트워크 혼잡 상황으로 인식하여 혼잡 윈도우의 크기를 계속해서 감소시키기 때문이다.

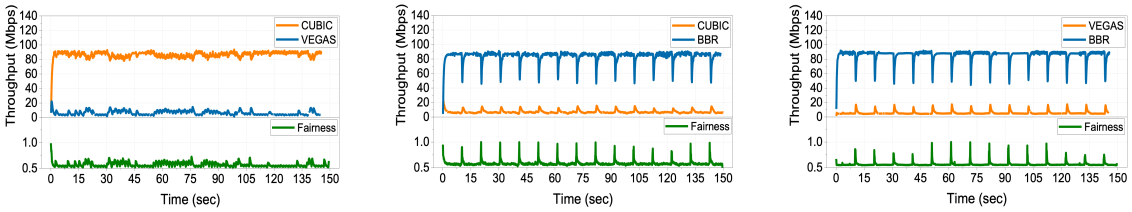
[CUBIC vs BBR]: CUBIC과 BBR의 상호작용 성능은 버퍼의 크기에 의존함을 그림 8-(b)와 그림 9-(b)를 통해 확인할 수 있다. 버퍼의 크기가 큰 경우, 두 플로우의 데이터 처리율은 주기적으로 심각한 성능 변동이 발생한다. CUBIC의 버퍼를 채우는 동작으로 인해 BBR은 Probe RTT 모드에서 실제 전파지연 시간보다 높은 $RTT_{\min}$을 측정하게 되고, 그 결과 매우 긴 대기열을 생성하여 CUBIC을 압도한다. BBR은 $RTT_{\min}$이 만료되어 다시 Probe RTT 모드에 진입하면, 자신이 채우고 있던 대부분의 버퍼를 비우며 실제 전파지연 시간에 가까운 $RTT_{\min}$를 측정되며, 이어서 CUBIC의 동작 특성에 의해 대부분의 버퍼는 CUBIC에 의해 채워지고 대부분의 대역폭은 CUBIC에 의해 점유된다. 뿐만 아니라 실제와 다른 BDP 계산으로 인해 BBR은 690, CUBIC은 327개의 상대적으로 많은 패킷 재전송이 발생한 것을 부록의 표를 통해 확인할 수 있다. 버퍼의 크기가 작은 경우, BBR이 대부분의 대역폭을 차지하는 확인할 수 있다. BBR은 이용 가능한 대역폭의 크기와 최소 지연 시간의 곱으로 이루어진 BDP를 기반으로 최대 2BDP의 데이터를 내보낸다. 이 때, 생성될 수 있는 최대 대기열의 크기인 1BDP는 버퍼의 크기를 초과하므로 과도한 패킷 손실이 지속적으로 발생하며 CUBIC은 BBR에 의한 손실로 인해 혼잡 윈도우의 크기를 전혀 증가시키지 못하고 낮은 상태를 유지한다.

[Vegas vs BBR]: BBR과 Vegas가 링크를 공유할 때, 두 플로우 간의 데이터 처리율 성능은 버퍼의 크기와 관계없이 BBR에 의해 대부분의 대역폭이 사용되는 것을 그림 8-(c)과 그림 9-(c)를 통해 확인할 수 있다. 앞에서 언급한 것과 같이 BBR은 데이터를 전송할 때 최대 1BDP크기의 버퍼를 채운다. Vegas는 BBR에 의한 대기 지연을 네트워크의 혼잡 상태로 판단하여 혼잡 윈도우의 크기를 낮은 값으로 유지한다.



(a) CUBIC vs Vegas (b) CUBIC vs BBR (c) Vegas vs BBR

그림 8. 서로 다른 그룹 내의 혼잡제어 알고리즘 간 상호작용 (Large Buffer: 5BDP)
Fig. 8. Interaction between congestion algorithm in the same group (Large Buffer: 5BDP)



(a) CUBIC vs Vegas (b) CUBIC vs BBR (c) Vegas vs BBR

그림 9. 서로 다른 그룹 내의 혼잡제어 알고리즘 간 상호작용 (Large Buffer: 5BDP)
Fig. 9. Interaction between congestion algorithm in the same group (Large Buffer: 5BDP)

3.2.2 전파지연 시간이 서로 다른 플로우 간 성능 평가

서로 다른 RTT를 가진 두 플로우 간의 성능을 평가하기 위해 우리는 병목링크 버퍼의 크기를 5BDP로 설정하였으며 호스트 1과 3 사이의 전파지연 시간은 10ms으로, 호스트 2와 3 사이의 전파지연 시간을 40ms으로 설정하였다. 호스트 1과 2는 동시에 150초 동안 호스트 3에게 동시에 데이터를 전송하였으며. 두 플로우 간의 데이터 처리율 공평성을 평가하기 위해 Jain's Index를 사용하였다. 시간에 따른 데이터 처리율 및 공평성 지수는 그림 10에 나타내었다.

[손실기반 알고리즘]: ACK를 수신할 때마다 혼잡 윈도우의 크기를 조절하는 Reno는 RTT가 낮은 플로우가 높은 RTT 플로우에 비해 높은 대역폭을 차지하는 것을 그림 10-(a)를 통해 확인할 수 있다. 반면, 그림 10-(b)를 통해 RTT와 독립적으로 동작하는 CUBIC은 초기에 공평성 지수가 0.75 정도로 낮게 나타났지만 시간이 지날수록 1.0에 가까워지며 Reno에 비해 향상된 RTT 공평성 성능을 보이는 것을 확인할 수 있다.

[지연기반 알고리즘]: 대기 지연을 혼잡의 신호로 인식하여 대기열 생성을 최소화하는 지연기반 알고리즘은 서로 다른 RTT를 가진 플로우 간 공평성 성능이 매우 좋게 나타났으며, 그림 10-(c), (d)를 통해 확인할 수 있다.

[하이브리드 알고리즘]: RTT가 서로 다른 Veno 플로우가 공존할 때, RTT가 작은 플로우가 더 많은 대역폭을 차지하는 것을 그림 10-(e)를 통해 확인할 수 있다. Veno는 ACK를 수신할 때 혼잡 윈도우의 크기를 조절하는 동작 특성으로 인해 RTT가 작은 플로우가 더욱 빠르게 혼잡 윈도우를 증가시킨다.

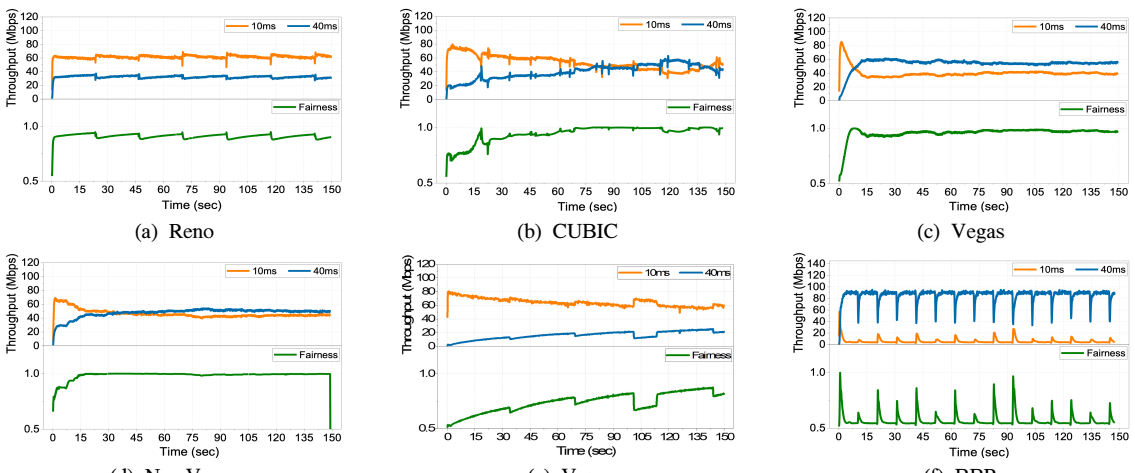
[모델기반 알고리즘]: RTT가 서로 다른 BBR 플로우가 공존할 때, 대부분의 대역폭은 큰 RTT를 가진 플로우에 의해 점유 되는 것을 그림 10-(f)를 통해 확인할 수 있다. 이러한 결과는 $RTT_{\min}$을 이용하여 계산된 BDP를 기반으로 내보낼 데이터의 양을 결정하는 BBR의 동작 특성으로 인해 발생한다.

3.3 새로 진입하는 플로우의 안정화 실험

기존 플로우가 존재하는 네트워크에 새로운 플로우가 진입했을 때, 새로운 플로우 및 기존 플로우의 성능 안정화를 평가하기 위해 호스트 1과 3사이의 병목링크 버퍼의 크기는 5BDP, 전파지연 시간은 10ms으로 설정하였으며, 3개의 플로우가 먼저 250초 동안 데이터를 전송한다. 이 후, 100초가 지났을 때 새로운 플로우가 합류하여 150초 동안 데이터를 전송한다. 실험은 CUBIC, Vegas, BBR을 이용하여 수행되었으며, 시간에 따른 데이터 처리율 및 공평성 지수를 그림 11에 나타내었다.

그림 11-(a)를 통해 새로 합류한 CUBIC 플로우는 기존의 동기화 된 CUBIC 플로우와 거의 유사한 처리율을 보이며 대부분의 시간에서 공평성은 1.0에 가깝게 유지된다. 하지만, 손실기반 알고리즘의 패킷 손실이 발생했을 때 혼잡 윈도우를 줄이는 동작 특성으로 인해 데이터 처리량 및 공평성 지수가 계속해서 변화하는 것을 확인할 수 있다.

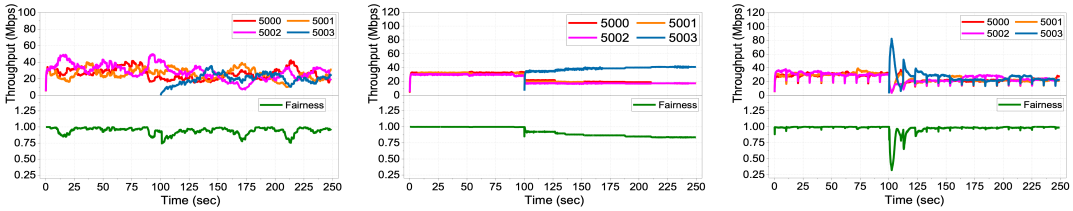
동시에 시작한 기존 3개의 Vegas 데이터 처리율 안정성 성능은 매우 좋은 것을 그림 11-(b)의 0~100



(a) Reno (b) CUBIC (c) Vegas

(d) NewVegas (e) Veno (f) BBR

그림 10. RTT가 서로 다른 플로우 간의 상호 작용
Fig. 10. Interaction between different RTT flow



(a) CUBIC (b) Vegas (c) BBR

그림 11. 새로 진입한 플로우의 안정화 실험 결과
Fig. 11. Stabilization result of newly entered flow

초 구간에서 확인할 수 있다. 하지만 새로운 Vegas 플로우가 진입했을 때, 기존의 플로우에 인해 실제 전파 전파지연 시간보다 높은 최소 RTT를 측정하여 병목 링크의 버퍼에 대기열이 생성되기 시작하는 기준점을 실제보다 높게 예측한다. 그 결과 기존의 3개 플로우보다 더 높은 데이터 처리율을 보이며, 시간이 지남에 따라 공평성 지수가 점점 낮아지는 것을 확인할 수 있다.

마지막으로, 그림 11-(c)를 통해 새로 들어온 BBR 플로우는 기존 3개의 BBR 플로우와 동기화 되는데 약 30 초의 긴 안정화 시간이 소요되었다. 새로 들어온 플로우는 기존 3개의 플로우가 생성한 대기열로 인해 실제 전파지연 시간 보다 높은 $RTT_{\min}$을 측정하며, 그 결과 기존 플로우 보다 높은 데이터 처리량을 보인다. 하지만 4개의 플로우가 동시에 유사한 $RTT_{\min}$을 측정하여 내보내는 데이터의 양이 비슷해지며 공평성 지수가 1에 가깝게 유지되는 완전한 안정화가 이루어진다.

Ⅳ. 결 론

본 논문에서는 다양한 TCP 혼잡 제어 알고리즘을 동작 특성과 혼잡 인식 방법에 따라 손실기반, 지연기반, 하이브리드 및 모델기반 혼잡제어 알고리즘으로 분류하여 각 그룹의 동작 특성을 파악하고 상호간 성능을 평가하였으며 본 논문의 실험 및 결과를 통해 다양한 네트워크 환경에서 기존 알고리즘과의 상호 성능 평가를 위해 고려할 요소와 시뮬ㅁ 구성 및 방법을 참고할 수 있다. 다수의 플로우가 공통의 병목 링크에서 공존하는 경우 각 혼잡 제어 알고리즘의 동작 특성과 지연 시간 및 버퍼의 크기와 같은 네트워크 환경에 의존하여 서로 상호작용 성능이 다르게 나타나는 것을 확인하였다. 특히, BBR은 버퍼의 크기 및 지연 시간에 따라 매우 큰 성능 변화가 나타나는 것을 확인하였다.

References

[1] J. Postel, “*Transmission control protocol specification*,” RFC 793, IETF, 1981.

[2] A. Afanasyev, N. Tilley, P. Reiher, and L. Kleinrock, “Host-to-host congestion control for TCP,” *IEEE Commun. Surv. Tuts.*, vol. 12, no. 3, pp. 304-342, 2010.

[3] M. Allman, V. Paxson, and E. Blanton, “*TCP Congestion Control*,” RFC 5681, IETF, 2009.

[4] L. Brakmo, S. O’Malley, and L. Peterson, “TCP Vegas: New techniques for congestion detection and avoidance,” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 24, no. 4, pp. 24-35, 1994.

[5] J. Sing and B. Soh, “TCP new vegas: Improving the performance of TCP Vegas over high latency links,” in *Proc. 4th IEEE Int. Symp. Netw. Comput. Appl. NCA 2005*, vol. 2005, pp. 73-80, 2005.

[6] N. Cardwell, Y. Cheong, C. S. Gunn, S. H. Yeganch, and V. Jacobson, “BBR: Congestion-Based congestion control,” *ACM Queue,* vol. 14, no. 5, 2016.

[7] G. H. Kim, Y. J. Song, C. H. Park, Y. Z. Cho, “Standardization and Research Trends of BBR Congestion Control Algorithm,“ *J. KICS*. vol. 44, no. 09, pp. 1713-1722, 2019.

[8] M. Hock, R. Bless, and M. Zitterbart, “Experimental evaluation of BBR congestion control,” in *Proc. Int. Conf. Netw. Protoc. ICNP*, 2017.

[9] K. Miyazawa, K. Sasaki, N. Oda, and S. Yamaguchi, “Cycle and Divergence of Performance on BBR,” in *Proc. 2018 IEEE*

7th Int. Conf. Cloud Networking, CloudNet 2018, pp. 1-6, 2018.

[10] D. Scholz, B. Jaeger, L. Schwaighofer, D. Raumer, F. Geyer, and G. Carle, "Towards a deeper understanding of TCP BBR congestion control," *IFIP Networking,* 2018.

[11] K. Yamada, R. Wang, M. Y. Sanadidi, and M. Gerla, "TCP westwood with agile probing: Dealing with dynamic, large, leaky pipes," *IEEE Int. Conf. Commun.,* vol. 2, pp. 1070-1074, 2004.

[12] L. Xu, K. Harfoush, and I. Rhee, "Binary increase congestion control (BIC) for fast long-distance networks," in *Proc. IEEE INFOCOM,* vol. 4. pp. 2514-2524, 2004.

[13] S. Ha, I. Rhee, and L. Xu, "CUBIC: A new TCP-friendly high speed TCP variant," *ACM SIGOPS Operation Syst. Rev.*, vol. 42, no. 5, 2008.

[14] S. Floyd, "*HighSpeed TCP for large congestion windows*," Tech. Rep., 2003.

[15] K. Srijith, L. Jacob, and A. L. Ananda, "TCP Vegas-A: Improving the performance of TCP Vegas," *Comput. Commun.*, vol. 28, no. 4, pp. 429-440, 2005.

[16] C. P. Fu, S. C. Liew, "TCP Veno: TCP enhancement for transmission over wireless access networks," *IEEE J. Sel. Areas Commun,* vol. 21, no. 2, pp. 216-228, 2003.

[17] iperf, from https://iperf.fr.

[18] TCPinfo, from https://git.scc.kit.edu/CPUnetLOG/TCPlog.

[19] R. Jain, D.-M. Chiu, and W. R. Hawe, *A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Computer System.* Eastern Research Laboratory, Digital Equipment Corporation Hudson, MA, vol. 38, 1984.

송 영 준 (Yeong-Jun Song)

2019년 2월 : 경북대학교 전자공학부 졸업

2019년 3월~현재 : 경북대학교 전자공학부 석사과정

<관심분야> TCP 혼잡제어, 미래 네트워크, 전송계층프로토콜

[ORCID:0000-0001-7586-5795]

김 건 환 (Geon-Hwan Kim)

2015년 2월 : 경북대학교 전자공학부 졸업

2017년 2월 : 경북대학교 전자공학부 석사

2017년 3월~현재 : 경북대학교 전자공학부 박사과정

<관심분야> 드론 ICT 융합 기술, TCP 혼잡제어

[ORCID:0000-0003-2739-8939]

조 유 제 (You-Ze Cho)

1982년 : 서울대학교 전자공학과 학사

1983년 : 한국과학기술원 전기전자공학 석사

1988년 : 한국과학기술원 전기전자공학 박사

1992년~1994년 : Univ. of Toronto in Canada, 객원교수

2002년~2003년 NIST(미국 국립표준연구소) 객원연구원

<관심분야> 차세대이동 네트워크, 무선애드혹 네트워크, 이동성 관리 기술, 차세대 전송계층 프로토콜

[ORCID:0000-0001-9427-4229]

부록 1. 혼잡제어 알고리즘 간 상호 성능 평가
Appendix 1. Interactive performance evaluation between congestion control algorithm

		그룹		손실기반		지연기반		모델기반	하이브리드
	그룹	알고리즘	항목	Reno	CUBIC	Vegas	NewVegas	BBR	Veno
Large Buffer 5BDP	손실기반	Reno	처리율	45.5 / 49.0	31.4 / 61.5	89.6 / 3.34	81.2 / 11.8	42.0 / 50.9	60.5 / 28.0
			공평성 지수	0.99888	0.90499	0.54722	0.64231	0.99090	0.88116
			평균 cwnd	89 / 103	168 / 309	398 / 56	372 / 55	252 / 404	302 / 135
			평균 RTT	22.5 / 22.5	56.9 / 56.6	50.3 / 33.7	52.3 / 51.6	46.6 / 52.3	52.6 / 52.6
			재전송 패킷	23 / 32	44 / 59	16 / 1	21 / 3	214 / 540	21 / 11
		CUBIC	처리율	31.4 / 61.5	49.5 / 43.5	91.9 / 1.03	83.5 / 9.39	57.5 / 35.5	76.7 / 16.2
			공평성 지수	0.90499	0.99585	0.51120	0.61105	0.94700	0.70200
			평균 cwnd	168 / 309	269 / 240	455 / 10	428 / 56	244 / 318	390 / 92
			평균 RTT	56.9 / 56.6	59.4 / 59.7	56.7 / 76.7	58.6 / 60.1	41.6 / 42.2	57.9 / 58.9
			재전송 패킷	44 / 59	54 / 80	55 / 4	50 / 7	327 / 690	62 / 14
	지연기반	Vegas	처리율	89.6 / 3.34	91.9 / 1.03	44.8 / 48.1	5.15 / 87.8	3.76 / 87.7	2.50 / 90.4
			공평성 지수	0.54722	0.51120	0.99873	0.55845	0.54279	0.52763
			평균 cwnd	398 / 56	455 / 10	41 / 92	12 / 151	9 / 172	17 / 384
			평균 RTT	50.3 / 33.7	56.7 / 76.7	10.5 / 58.9	15.5 / 19.5	21.1 / 19.1	50.8 / 48.5
			재전송 패킷	16 / 1	55 / 4	0 / 0	0 / 0	0 / 0	0 / 2
		New Vegas	처리율	81.2 / 11.8	83.5 / 9.39	5.15 / 87.8	45.0 / 47.9	26.7 / 66.0	18.2 / 74.7
			공평성 지수	0.64231	0.61105	0.55845	0.99026	0.84765	0.72998
			평균 cwnd	372 / 55	428 / 56	12 / 151	138 / 139	96 / 178	88 / 331
			평균 RTT	52.3 / 51.6	58.6 / 60.1	15.5 / 19.5	34.6 / 33.7	28.3 / 26.3	51.5 / 50.5
			재전송 패킷	21 / 3	50 / 7	0 / 0	0 / 0	0 / 0	5 / 3
	모델기반	BBR	처리율	42.0 / 50.9	57.5 / 35.5	3.76 / 87.7	26.7 / 66.0	49.8 / 42.8	52.2 / 40.7
			공평성 지수	0.99090	0.94700	0.54279	0.84765	0.99431	0.98490
			평균 cwnd	252 / 404	244 / 318	9 / 172	96 / 178	89 / 103	316 / 190
			평균 RTT	46.6 / 52.3	41.6 / 42.2	21.1 / 19.1	28.3 / 26.3	22.5 / 22.5	46.6 / 40.7
			재전송 패킷	214 / 540	327 / 690	0 / 0	0 / 0	0 / 0	133 / 85
	하이브리드	Veno	처리율	60.5 / 28.0	76.7 / 16.2	2.50 / 90.4	18.2 / 74.7	52.2 / 40.7	47.9 / 44.9
			공평성 지수	0.88116	0.70200	0.52763	0.72998	0.98490	0.99895
			평균 cwnd	302 / 135	390 / 92	17 / 384	88 / 331	316 / 190	250 / 193
			평균 RTT	52.6 / 52.6	57.9 / 58.9	50.8 / 48.5	51.5 / 50.5	46.6 / 40.7	53.7 / 53.2
			재전송 패킷	21 / 11	62 / 14	0 / 2	5 / 3	133 / 85	5 / 10
Small Buffer 0.8 BDP	손실기반	Reno	처리율	45.6 / 47.2	43.1 / 49.8	86.8 / 5.92	66.8 / 25.6	6.41 / 85.2	64.1 / 28.6
			공평성 지수	0.99970	0.99482	0.56788	0.83415	0.57481	0.87210
			평균 cwnd	66 / 63	65 / 73	116 / 10	93 / 36	13 / 162	93 / 47
			평균 RTT	15.5 / 15.5	16.5 / 16.3	15.4 / 16.1	15.8 / 15.9	18.0 / 17.4	15.9 / 16.1
			재전송 패킷	407 / 373	366 / 274	226 / 160	299 / 104	2939 / 10689	325 / 167
		CUBIC	처리율	43.1 / 49.8	44.2 / 48.7	86.5 / 6.33	67.4 / 25.4	7.04 / 84.3	62.5 / 30.4
			공평성 지수	0.99482	0.99765	0.57278	0.82998	0.58293	0.89334
			평균 cwnd	65 / 73	70 / 76	126 / 12	101 / 41	12 / 158	96 / 54
			평균 RTT	16.5 / 16.3	17.2 / 17.3	16.6 / 17.7	17.0 / 17.5	17.9 / 17.4	16.9 / 17.2
			재전송 패킷	366 / 274	320 / 328	234 / 96	245 / 116	1502 / 6386	288 / 104
	지연기반	Vegas	처리율	86.8 / 5.92	86.5 / 6.33	43.9 / 48.9	7.98 / 84.9	5.57 / 85.4	7.27 / 85.7
			공평성 지수	0.56788	0.57278	0.99710	0.59316	0.56494	0.58422
			평균 cwnd	116 / 10	126 / 12	41 / 46	10 / 109	9 / 172	9 / 115
			평균 RTT	15.4 / 16.1	16.6 / 17.7	10.5 / 10.6	9.9 / 14.7	16.9 / 14.9	17.3 / 15.3
			재전송 패킷	226 / 160	234 / 96	30 / 29	0 / 3	19 / 137	27 / 47
		New Vegas	처리율	66.8 / 25.6	67.4 / 25.4	7.98 / 84.9	46.0 / 46.8	2.92 / 88.3	28.6 / 64.2
			공평성 지수	0.83415	0.82998	0.59316	0.99992	0.53303	0.87171
			평균 cwnd	93 / 36	101 / 41	10 / 109	66 / 67	6 / 169	40 / 91
			평균 RTT	15.8 / 15.9	17.0 / 17.5	9.9 / 14.7	16.6 / 16.6	27.6 / 17.0	16.6 / 16.1
			재전송 패킷	299 / 104	245 / 116	0 / 3	7 / 5	587 / 5661	44 / 56
	모델기반	BBR	처리율	6.41 / 85.2	7.04 / 84.3	5.57 / 85.4	2.92 / 88.3	44.4 / 47.0	86.5 / 4.84
			공평성 지수	0.57481	0.58293	0.56494	0.53303	0.99919	0.55577
			평균 cwnd	13 / 162	12 / 158	9 / 172	6 / 169	90 / 96	88 / 94
			평균 RTT	18.0 / 17.4	17.9 / 17.4	16.9 / 14.9	27.6 / 17.0	18.8 / 18.8	18.6 / 18.5
			재전송 패킷	2939 / 10689	1502 / 6386	19 / 137	587 / 5661	32794 / 34392	6006 / 1094
	하이브리드	Veno	처리율	64.1 / 28.6	62.5 / 30.4	7.27 / 85.7	28.6 / 64.2	86.5 / 4.84	47.1 / 45.7
			공평성 지수	0.87210	0.89334	0.58422	0.87171	0.55577	0.99977
			평균 cwnd	92 / 47	96 / 54	9 / 115	40 / 91	88 / 94	68 / 52
			평균 RTT	15.9 / 16.1	16.9 / 17.2	17.3 / 15.3	16.6 / 16.1	18.6 / 18.5	16.6 / 16.7
			재전송 패킷	325 / 167	288 / 104	27 / 47	44 / 56	6006 / 1094	92 / 89