TCP Vegas의 공정성 향상을 위한 혼잡 제어 알고리즘

(A Congestion Control Algorithm for the Fairness Improvement of TCP Vegas)

오 민 철 † 송 병 훈 † 정 광 수 ††
(Mincheol Oh) (Byunghoon Song) (Kwangsue Chung)

요 약 인터넷의 안정성에 가장 큰 영향을 미치는 요소는 종단간에 이루어지는 TCP 혼잡제어이다. 현재 인터넷의 주요 TCP 버전인 Reno가 사용하는 수동적인 혼잡제어 방법은 네트워크의 혼잡을 심화시키는 원인이 된다. 이러한 Reno의 문제점을 개선하기 위해 제안된 Vegas는 Reno에 비해 우수한 성능을 가짐이 증명되었음에도 불구하고 두 가지 심각한 불공정성 문제를 가지고 있기 때문에 범용적으로 사용되지 못하고 있다.

본 논문에서는 이러한 Vegas의 문제점을 보완하기 위해서 기존의 Vegas 혼잡제어 알고리즘을 개선한 새로운 TCP PowerVegas 혼잡제어 알고리즘을 제안한다. rtt(round trip time)만을 기반으로 네트워크의 혼잡을 제어하는 기존의 Vegas에 비해서 제안한 PowerVegas는 rtt와 패킷 손실 정보를 유기적으로 결합시킨 새로운 기법으로 경쟁력 있는 혼잡제어를 수행한다. 그러므로 기존의 Vegas에서 발생했던 불공정성 문제를 모두 효과적으로 개선할 수 있다. 제안한 알고리즘의 성능을 검증하기 위해 동일한 시뮬레이션 환경에서 PowerVegas와 Reno 및 Vegas를 비교하는 실험을 수행하였다. 실험 결과를 통해서 제안한 PowerVegas가 기존 Reno의 혼잡제어 방법에 비해 우수한 성능을 보일 뿐만 아니라, Vegas의 불공정성 문제도 크게 개선되었음을 확인할 수 있었다.

키워드 : TCP Vegas, 혼잡제어, 공정성

Abstract The most important factor influencing the robustness of the Internet is the end-to-end TCP congestion control. However, the congestion control scheme of TCP Reno, the most popular TCP version on the Internet, employs passive congestion indication. It makes worse the network congestion. Recently, Brakmo and Peterson have proposed a new version of TCP, which is named TCP Vegas, with a fundamentally different congestion control scheme from that of the Reno. Many studies indicate that the Vegas is able to achieve better throughput and higher stability than the Reno. But there are two unfairness problems in Vegas. These problems hinder the spread of the Vegas in current Internet.

In this paper, in order to solve these unfairness problems, we propose a new congestion control algorithm called TCP PowerVegas. The existing Vegas depends mainly only on the rtt(round trip time), but the proposed PowerVegas use the new congestion control scheme combined the information on the rtt with the information on the packet loss. Therefore the PowerVegas performs the congestion control more competitively than the Vegas. Thus, the PowerVegas is able to solve effectively these unfairness problems which the Vegas has experienced. To evaluate the proposed approach, we compare the performance among PowerVegas, Reno and Vegas under same network environment. Using simulation, the PowerVegas is able to achieve better throughput and higher stability than the Reno and is shown to achieve much better fairness than the existing Vegas.

Key words : TCP Vegas, Congestion Control, Fairness

· 본 연구는 한국 과학재단 목적기초연구(R01-2002-000-00179-0(2002)) 지원으로 수행되었음

† 비 회 원 : 광운대학교 전자공학부
mcoh@adams.kwangwoon.ac.kr
byungh@adams.kwangwoon.ac.kr

†† 종신회원 : 광운대학교 전자공학부
kchung@daisy.kwangwoon.ac.kr

논문접수 : 2003년 1월 2일
심사완료 : 2004년 1월 30일

1. 서 론

TCP(Transmission Control Protocol)의 혼잡제어 알고리즘(Congestion Control Algorithm)은 1988년의 TCP Tahoe 이래로 1990년 TCP Reno, 1995년 TCP SACK에 이르기까지 다양하게 연구되어왔다[1]. 이러한 TCP 혼잡제어 알고리즘의 주요 목적은 송신단의 전송률을 직접 제어하여 혼잡상황으로 인해 발생되는 데이타의 무분별한 손실을 막기 위함이다.

현재 인터넷의 주요 전송 프로토콜로 사용되고 있는 Reno의 혼잡제어 방법인 AIMD(Additive Increase Multiplicative Decrease) 알고리즘은 비 혼잡상황에서는 순차적으로 전송률을 증가시키다가 패킷 손실이 발생하면 전송률을 급격히 감소시킨다[2,3]. 이러한 Reno의 혼잡제어 방법은 패킷 손실에 대한 정보에만 입각해서 네트워크의 상태를 유추하기 때문에 급격한 전송률의 변화가 빈번히 발생하게 되는 문제점을 지니고 있다. 또한, 네트워크에 주기적인 혼잡상황을 야기시킬 수 있으며 공존하는 다른 TCP 플로우들의 전송률에 민감하게 영향을 주는 문제점도 가지고 있다[3]. 이러한 Reno의 혼잡제어 방법의 근본적인 문제를 해결하게 위한 새로운 TCP 혼잡제어 알고리즘에 관한 연구가 많이 진행되고 있다. 특히 Brakmo와 Peterson에 의해서 제안된 TCP Vegas 혼잡제어 알고리즘은 Reno의 혼잡제어 방법과 다르게 rtt(round trip time)의 변화정도를 기반으로 네트워크의 상태를 예측하여 전송률을 조절하기 때문에 전송률의 급격한 변화가 발생하지 않으며 패킷 손실이 적어서 전송효율도 매우 좋다. 또한 Reno의 문제점이었던 다른 TCP 플로우의 전송률에 미치는 영향도 상대적으로 적다[4,5].

많은 관련 연구들을 통해서 Vegas는 현재 인터넷에서의 주요 혼잡제어 방법인 Reno에 비해 전송량, 네트워크의 안정성, 패킷 손실률, 지연시간 등 많은 부분에서 더 우수한 혼잡제어 기법임이 증명되었다[6-9].

그러나 실제로 Vegas와 Reno를 동일한 네트워크 환경에서 경쟁시키면, Reno의 전송률의 변화에 따른 영향으로 경쟁하는 Vegas의 전송률이 현저하게 감소하는 심각한 불공정성 문제를 보이고 있다. 또한 *rtt*가 서로 다른 Vegas 플로우의 경쟁에서도 *rtt*가 작은 플로우가 *rtt*가 큰 플로우에 비해서 더 낮은 전송률을 갖게 되는 불공정성 문제가 나타난다. 그러므로 Vegas가 신뢰성 있는 전송 프로토콜로 자리 잡기 위해서는 이러한 두 가지 문제점을 보완하는 것이 매우 중요한 기술적 과제로 인식되어지고 있다[10-13].

본 논문에서는 Reno와 공존하는 네트워크 환경에서 Vegas의 근본적인 혼잡제어 방법을 보완한 새로운 TCP PowerVegas 혼잡제어 알고리즘을 제안한다. 제안한 PowerVegas는 Vegas의 장점인 전송률의 안정성을 그대로 유지하면서 경쟁력을 새롭게 보강하여 Vegas의 가장 큰 약점인 Reno와의 불공정성 문제를 해결하였다. 또한 제안한 PowerVegas는 기존의 Vegas에서 나타났던 Vegas와 Vegas 플로우 간의 불공정성 문제도 해결하였다.

본 논문은 2장에서는 TCP Vegas 혼잡제어 알고리즘과 공정성을 분석하였고, 3장에서는 새로 제안한 PowerVegas 혼잡제어 알고리즘으로 기술하였다. 4장에서는 시뮬레이터를 이용하여 제안한 알고리즘을 검증하고 PowerVegas의 공정성을 분석하였으며, 마지막으로 6장에는 결론을 맺었다.

2. TCP Vegas

현재 인터넷에서 널리 사용되고 있는 Reno는 기본적으로 윈도우를 기반으로 전송률을 제어하는 AIMD 혼잡제어 알고리즘을 사용한다. 즉, 윈도우 크기를 증가시키다가 패킷 손실이 발생하면 윈도우 크기를 줄여서 전송률을 감소시키는 주기를 반복적으로 수행한다. 이와 같은 Reno의 혼잡제어 방법은 혼잡상황이 발생했을 때 전송률을 급격히 감소시켜 혼잡상황에 빨리 적응할 수 있는 장점이 있으나, 전송률의 전체적인 진동폭이 크다는 단점이 있다. 또한 주기적으로 반복되는 윈도우 크기의 증가와 감소로 인해서 네트워크의 자원을 충분히 사용하지 못하는 주기와 과도하게 사용하는 주기를 빈번히 왕복하게 되어 전체적인 네트워크 효율이 나빠지게 되는 원인이 된다. 이러한 Reno의 문제점은 오직 패킷 손실에 의해서만 네트워크의 혼잡을 인지하는 알고리즘의 근본적인 특성으로 인해서 발생한다.

Reno의 문제점을 개선하기 위해서 Brakmo와 Peterson에 의해서 제안된 Vegas는 송신측에서 전송했던 패킷의 *rtt*를 측정하여 이를 기반으로 윈도우 크기를 조절하는 새로운 혼잡제어 알고리즘을 사용한다. 본 장에서는 Vegas 혼잡제어 알고리즘의 기본적인 원리와 현재 지적되고 Vegas 혼잡제어 알고리즘의 문제점에 대해서 기술하고자 한다.

2.1 TCP Vegas 혼잡제어 알고리즘

Vegas는 패킷 손실에 대한 정보만으로 네트워크의 혼잡제어를 수행하는 Reno와 다르게 측정된 *rtt*를 기반으로 네트워크의 혼잡제어 수행한다. 즉, *rtt*가 커지면 네트워크의 혼잡상황이 증가되었다고 판단하여 윈도우 크기를 감소시키고, 반대로 *rtt*가 작아지면 네트워크의 혼잡정도가 감소되었다고 판단하여 윈도우 크기를 증가시킨다. 이러한 Vegas의 혼잡제어 방법은 윈도우 크기를 이상적인 상태로 관리하는데 효과적이며, 결과적으로

그림 1에서 보는 것과 같은 평형상태에 도달하여 적절한 전송률을 유지할 수 있게 된다.

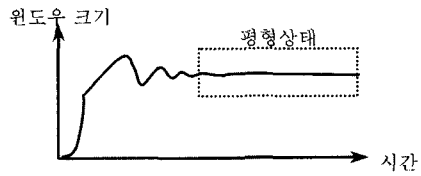


그림 1 Vegas의 윈도우 크기 변화

이와 같은 Vegas의 윈도우 크기의 변화는 Reno에 비해 진동폭이 작고 전체적으로 안정된 상태를 유지시킬 수 있는 특징이 있다. 이러한 Vegas의 혼잡제어 알고리즘의 동작 과정을 기능별로 서술하면 다음과 같다 [3,4].

i) *BaseRTT*를 구한다.

*BaseRTT*는 혼잡이 없을 때의 *rtt*라고 정의한다. 그러므로 측정한 *rtt*들 중에서 최소 *rtt*를 *BaseRTT*로 지정한다. 일반적으로 라우터의 큐가 길어지기 전인 첫 번째 TCP 세그먼트에 대한 *rtt* 측정값이 *BaseRTT*가 된다.

ii) 기대전송률인 *Expected*를 구한다.

i)에서 구한 *BaseRTT* 동안에 전송할 수 있을 것으로 기대되는 데이타의 양을 윈도우 크기의 기대값 W_{exp}라고 하고, 이 플로우에 대해 기대할 수 있는 기대전송률인 *Expected*를 다음의 식 (1)과 같이 구한다.

$$Expected = W_{exp} / BaseRTT \quad (1)$$

iii) 실제전송률인 *Actual*을 구한다.

측정된 *rtt* 동안 실제로 전송한 데이타의 양을 윈도우 크기의 실제값 W_{act}라고 하고, 이 플로우의 실제전송률인 *Actual*은 W_{act}를 현재의 *rtt*로 나누어 구한다. 그리고 $BaseRTT \le rtt$이므로 $Actual \le Expected$가 항상 성립한다.

$$Actual = W_{act} / rtt \quad (2)$$

iv) 두 전송률 간의 차이인 *Diff*를 구한다.

*Actual*과 *Expected*를 비교하여 두 전송률 간의 차이인 *Diff*를 구한다. *Diff*는 네트워크의 상태를 정량적으로 표현한 값이다.

$$Diff = \left(\frac{W_{exp}}{BaseRTT} - \frac{W_{act}}{rtt} \right) \times BaseRTT \quad (3)$$

v) *Diff*에 따라 윈도우 크기를 조절한다.

Vegas는 *Diff*와 두 개의 임계값 α와 β를 이용하여 식 (4)와 같이 세 개의 구간을 정의하고 있다. iv)에서 구한 *Diff*가 [V_구간-1], [V_구간-2] 또는 [V_구간-3]일 때 각각의 윈도우 크기(W)를 한 개 증가, 고정 또는 한 개 감소시킨다. 임계값 α와 β는 네트워크 상태를 판단하기 위한 기준값이다. 그러므로 Vegas는 플로우가 네트워크 내에서 가지는 여분의 데이타, 즉 라우터의 큐를 차지하는 패킷의 양을 α와 β의 값 사이로 제한하려고 노력한다. 일반적으로 α, β의 기본값은 α=1, β=3이다.

$$W = \begin{cases} W-1, & if\ Diff > \beta \quad \cdots [V\_구간-3] \\ W, & if\ \alpha \le Diff \le \beta \cdots [V\_구간-2] \\ W+1, & if\ Diff < \alpha \quad \cdots [V\_구간-1] \end{cases} \quad (4)$$

*Diff*와 임계값 α와 β를 이용하여 식 (4)의 구간을 도시하면 그림 2와 같다.

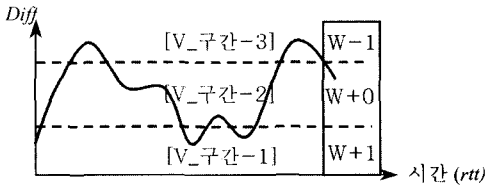


그림 2 Vegas에 의해 정의된 [V_구간]

그림 2의 각 구간은 iv)에서 구한 *Diff*의 변화정도를 근거로 판단한 네트워크의 상태를 의미한다. 먼저, [V_구간-1]은 네트워크가 혼잡하지 않은 상태를 의미하며, 이때는 전송률을 높이기 위해서 윈도우 크기를 하나 증가시킨다. 반대로, [V_구간-3]은 네트워크가 혼잡한 상태를 의미하며, 이때는 전송률을 줄여서 네트워크의 혼잡정도를 낮추기 위해서 윈도우 크기를 하나 감소시킨다. 그리고 [V_구간-2]는 네트워크의 안정상태를 의미하며, 이때는 전송률을 그대로 유지시키기 위해서 윈도우 크기에 변화를 주지 않는다. Vegas는 α와 β 사이의 [V_구간-2]를 유지하여 윈도우 크기가 특정값에 수렴하도록 함으로써 평형상태를 이루게 된다. 이와 같은 Vegas의 선형증가와 선형감소 방법은 기존의 Reno에서 채택한 선형증가와 지수형감소에 비해서 윈도우 크기의 변화가 급격하지 않기 때문에 상대적으로 전송상태가 안정적이며, 결과적으로 전체적인 네트워크의 상태를 안정적으로 유지시킨다.

Vegas의 *rtt* 기반의 혼잡제어 방법은 혼잡의 초기상황을 빠르게 인지할 수 있어서 패킷 손실에 따른 혼잡상황 발생 이후에 대처하는 기존의 Reno에 비해 패킷 손실이 적다. 또한 Vegas는 혼잡이 발생할 때까지 윈도우 크기를 늘리는 Reno와는 다르게 네트워크 내에 큐잉되는 패킷의 수를 특정 범위 이내로 제한하여 윈도우 크기가 과도하게 커지는 현상을 지양함으로써 높은 처리율을 유지하면서도 혼잡상황을 억제하는 효과가 뛰어나다.

2.2 TCP Vegas 혼잡제어 알고리즘의 문제점

본 절에서는, Vegas의 문제점으로 지적되고 있는 불공정성 문제에 대해서 기술한다. Vegas 혼잡제어 알고리즘의 불공정성은 주로 Vegas와 Reno 플로우와 공존하는 경우와 *rtt*가 서로 다른 Vegas 플로우들이 공존하는 경우에 나타난다.

2.2.1 Vegas와 Reno 플로우 간의 불공정성 문제

Vegas의 가장 큰 문제점은 현재 인터넷의 주요 혼잡제어 방법인 Reno와 공존하는 경우에 발생하는 불공정성이다. 이러한 Reno와의 불공정성 문제는 Vegas가 널리 사용되는 것을 가로막는 치명적인 문제로 지적되고 있다. 그림 3은 Reno 플로우와 경쟁하는 Vegas 플로우의 윈도우 크기와 전송량의 변화를 나타낸 것이다.

앞절에서 설명했듯이 Vegas는 자신의 윈도우 크기를 특정값으로 수렴시키기 때문에 Reno에 비해서 가용 대역폭을 차지하는 경쟁에는 전송률이 너무 낮게 되는 문제점이 있다. 이러한 문제점의 원인은 다음과 같이 설명할 수 있다.

우선 Reno 플로우가 패킷을 전송하기 시작하면서 Vegas 플로우의 *rtt*는 점차 증가하게 된다. *rtt*가 증가하면, Vegas의 혼잡제어 알고리즘은 네트워크의 혼잡정도가 심화된 것으로 판단하여 패킷 손실이 발생하기 전에 윈도우 크기를 감소시키게 된다. 반면 Reno 플로우는 패킷 손실이 발생할 때까지 윈도우 크기를 계속 늘려 나간다. 그러므로 Reno 플로우가 패킷 손실로 인해서 윈도우 크기를 일시적으로 감소시킬 때까지 Vegas 플로우의 윈도우 크기는 계속해서 감소할 수 밖에 없게 된다. 이러한 상황에서 Reno 플로우가 감소한 윈도우 크기를 다시 증가시키는 동안 Vegas 플로우의 *rtt*도 역시 커지기 때문에 Vegas 플로우의 윈도우 크기는 더 이상 증가하지 않게 된다. 이와 같이 Reno 플로우에 비해 상대적으로 경쟁력이 약한 Vegas 플로우는 윈도우 크기를 회복할 기회가 없기 때문에 불공정성 문제가 발생하게 된다.

2.2.2 Vegas와 Vegas 플로우 간의 불공정성 문제

Reno에서는 *rtt*가 큰 플로우가 *rtt*가 짧은 *rtt*의 플로우에 비해 불리하게 동작한다. 그러나 Vegas의 경우에는 *rtt*가 작은 플로우가 더 불리한 모습을 나타낸다. 그림 4에서 *rtt*가 더 큰 Vegas 플로우가 상대적으로 더 큰 윈도우 크기를 유지하는 모습을 확인할 수 있다. 이러한 현상은 Vegas의 전송량이 $[W/rtt]$에 비례하기 때문이다. 그러므로 *rtt*가 서로 다른 두 플로우가 경쟁하는 경우, Vegas는 Reno와 반대로 *rtt*가 짧은 플로우의 전송률이 상대적으로 더 낮게 되는 불공정성 문제가 발생하게 된다.

*rtt*가 서로 다른 두 Vegas 플로우 간의 경쟁에서 나타나는 불공정성 문제를 분석하기 위해서 시간의 변화에 따른 윈도우 크기의 변화, 즉 $W_1(t)$와 $W_2(t)$의 변화

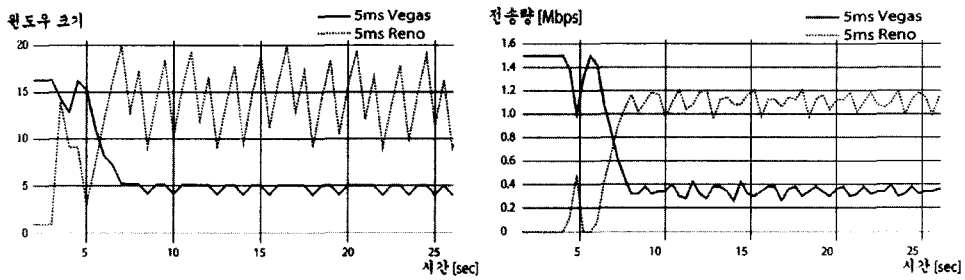


그림 3 Vegas와 Reno 플로우 간의 공정성 비교

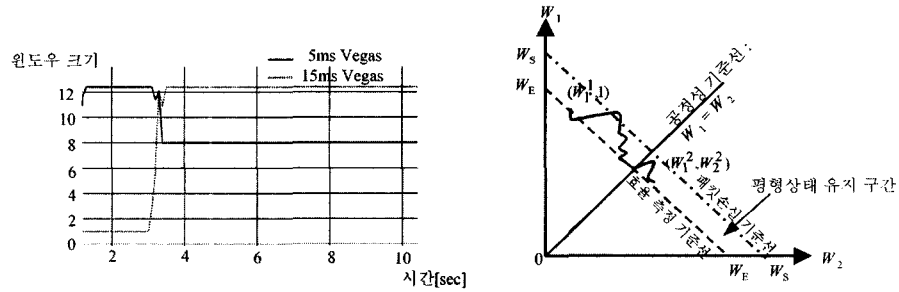


그림 4 *rtt*가 서로 다른 두 Vegas 플로우 간의 공정성 비교와 W_1-W_2 그래프

를 추적하는 그래프를 이용하였다[12]. Vegas 플로우들 간의 공정성 분석을 위해서, 플로우-1이 데이타를 전송하기 시작한 상황에서 플로우-2가 네트워크에 유입된 후 각 송신측은 Vegas 알고리즘에 따라 데이타를 전송하게 된다. 그리고 Vegas 플로우들 간의 공정성은 $W_1(t)$과 $W_2(t)$를 분석하여 그림 4의 W_1-W_2 그래프를 통해 나타내었다. 그래프에서 x축과 y축은 플로우-1과 플로우-2의 윈도우 크기를 나타내고, 네트워크 상태점 $(W_1(t), W_2(t))$은 시간 t에서 두 플로우가 공존하고 있는 네트워크의 이용 상태를 나타낸다. 공정성 기준선은 W_1=W_2인 경우를 말하는 것으로, 만일 네트워크 상태점이 이 선상에 위치해 있다면, 두 플로우의 윈도우 크기가 동일하다는 것을 의미한다. 또한 효율 측정 기준선은 링크가 사용되는 정도를 나타내는 기준이 된다. 이때, 두 플로우의 윈도우 크기의 합이 효율 측정 기준선 아래에 위치한다는 것은 링크의 대역폭을 완전하게 사용하지 않는다는 것을 의미하며, 패킷 손실 기준선보다 높은 곳에 있다는 것은 패킷의 손실을 의미한다. 그러므로 네트워크 상태점이 효율 측정 기준선과 패킷 손실 기준선사이의 평형상태 유지구간에 위치할 때 링크를 완전하게 사용하면서 패킷 손실이 없는 것을 의미하며, 이때 네트워크 상태점$(W_1(t), W_2(t))$이 공정성 기준선상에 위치하는 것이 가장 이상적인 상태이다.

2.1절에서 설명한대로 Vegas는 평형상태에서 윈도우 크기를 변화시키지 않고 유지하면서 임의의 고정된 값으로 수렴한다. 이러한 현상을 그림 4의 W_1-W_2 그래프에서 살펴보면 네트워크 상태점이 시작점인 (W_1^1, 1)에서 (W_1^2, W_2^2)로 이동한 후, 그 점으로 계속 수렴하는 것을 확인할 수 있다. 여기서, W_1^1은 플로우-2가 활성화되는 시점에서의 플로우-1의 윈도우 크기이고, W_1^2과 W_2^2는 플로우-1과 플로우-2가 각각 수렴하는 윈도우 크기이다. 즉, Vegas 플로우 간의 경쟁에서 나타나는 불공정성 문제의 근본적인 원인은 가능한 일정한 전송률을 유지하려고 노력하는 Vegas의 특징으로 인해서 나타나는 경쟁력의 약화에 있다.

2.3 TCP Vegas 불공정성 문제의 해결 방법

기존의 Vegas의 불공정성 문제점에 대한 해결책은 크게 AQM(Active Queue Management)과 Vegas의 파라메타의 튜닝을 통한 방법으로 구분할 수가 있다. 각각의 방법들은 모두 RTT에 따른 Reno와의 불공정성 문제를 해결하는 방안들을 제시하고 있으나 근본적인 문제 해결에는 한계가 있다[13,14].

2.3.1 AQM 파라메타 튜닝

AQM은 라우터 단의 혼잡 제어를 위한 모델이다. 대표적인 AQM으로는 RED(Random Early Drop)가 있다. 일반적으로 Vegas는 낮은 AQM 큐의 길이에서 최적의 성능을 나타낸다. 그러므로 AQM을 통한 해결 방법은 RED의 maxth 파라메타의 튜닝을 이용하여 낮은 큐의 길이를 갖도록 유도 한다. 그러나 낮은 큐의 길이는 Vegas의 성능 및 불균형성 문제에는 좋은 효과를 나타내지만, 상대적으로 낮은 큐 길이에 따른 빈번한 패킷 손실로 인해 Reno에는 좋지 않은 성능을 낼 수 있으므로 최적의 효과를 기대하기가 매우 어렵다.

2.3.2 Vegas 파라메타 튜닝

Vegas의 불공정성 문제를 특정 파라메타의 튜닝을 통해 해결하려는 방법으로 기존의 대표적인 연구로는 Lai의 Vegas 파라메타 튜닝이 있다[13]. 이 연구는 Vegas의 성능 및 불균형 문제를 Vegas의 핵심 파라메타인 α와 β의 튜닝을 통해 개선하였다. 특히, Lai는 실험을 통해서 α가 β에 비해서 Vegas 성능에 더욱 중요한 영향을 미침을 보였으며 α, β의 관계를 $\beta=\alpha+2$로 설정함으로서 불공정성 문제를 어느 정도 해결할 수 있음을 보였다. 그러나 이러한 방법도 근본적인 불공정 문제를 해결하기에는 미흡하며 네트워크 환경에 맞는 최적의 α, β를 매번 설정해줘야 하는 문제점이 남아 있다. 이것은 다양한 네트워크 환경에 따른 유연성에 큰 단점으로 지적된다.

3. TCP PowerVegas 혼잡제어 알고리즘

본 절에서는 2.2절에서 지적한 기존의 Vegas 혼잡제어 알고리즘의 불공정성 문제들을 개선하기 위해서 새롭게 제안한 PowerVegas 혼잡제어 알고리즘의 특징을 기술한다. 제안한 PowerVegas는 기본적으로 Vegas 파마메타 튜닝을 이용한 방법과는 구별되게 혼잡 제어 기법 자체를 수정한 새로운 방법으로 기본적으로 Vegas의 동작 기법에 다음의 세 가지 기법을 추가하여 Vegas의 불공정성 문제들을 해결하였다.

(1) 기존의 Vegas에서 정의한 [V_구간]을 확장하여 [P_구간]으로 재 정의하고, *Diff* 계산 알고리즘을 수정
(2) *rtt* 정보를 이용한 혼잡제어 알고리즘을 수정
(3) 패킷 손실 정보를 이용한 혼잡제어 알고리즘을 추가

3.1 P_구간 정의 및 *Diff* 계산 알고리즘

PowerVegas는 기존의 Vegas에서 정의한 세 개의 [V_구간]을 그림 5와 같이 여섯 개의 [P_구간]으로 재 정의하였다.

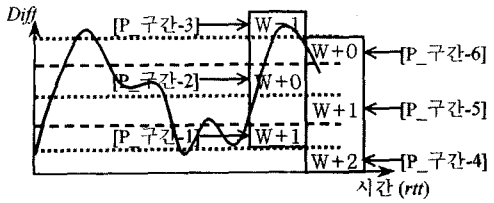


그림 5 PowerVegas에 의해 정의된 [P_구간]

기존의 Vegas의 문제점은 각 [V_구간]의 유효범위가 너무 크기 때문에 윈도우 크기의 변화가 발생하는 빈도가 적다는 것이다. 그러므로 Vegas 플로우의 전송률을 안정적으로 유지시킬 수는 있었으나 경쟁하는 Reno 플로우에 비해서는 상대적으로 경쟁력이 약할 수 밖에 없었다. 이러한 문제를 해결하기 위해서 PowerVegas는 기존의 Vegas의 *rtt* 정보를 이용한 혼잡제어 알고리즘을 수정하고 패킷 손실 정보를 이용한 혼잡제어 알고리즘을 새롭게 추가하여, 기존의 Vegas에서 사용하던 세 개의 [V_구간] 내에서 윈도우 크기의 변화가 생길 수 있는 새로운 구간을 재 정의하여 여섯 개의 [P_구간]으로 확장하였다. 수정한 *rtt*에 따른 혼잡제어 알고리즘과 새롭게 추가한 패킷 손실 정보를 이용한 혼잡제어 알고리즘에 대한 자세한 설명은 다음의 3.2절과 3.3절에서 다루어질 것이다.

또한 PowerVegas 혼잡제어 알고리즘은 *Diff* 계산에 사용되는 네 개의 인자, 즉 *BaseRTT*, *rtt*, W_{exp}, 그리고 W_{act} 중에서 네트워크의 상태에 따라 상대적으로 더 많은 영향을 미치게 되는 인자의 비중을 크게 부여함으로써 Vegas의 경쟁력을 강화시킬 수 있도록 식 (3)의 *Diff* 계산 알고리즘을 다음의 식 (5)와 같이 수정하였다.

$$Diff = \left(\frac{W_{exp}}{*w\_BaseRTT} - \frac{W_{act}}{rtt}\right) \times [(1-\mu)*w\_BaseRTT + \mu \cdot rtt] \quad (5)$$

$$New\_BaseRTT = \frac{BaseRTT}{2} + \frac{rtt}{2}, \ \mu = 0.92 \quad (6)$$

PowerVegas의 *Diff* 계산 알고리즘은 현재의 네트워크 상태를 반영하는 인자인 *rtt*의 비중을 높임으로써 평형상태에서 *BaseRTT*에 지나치게 의존하는 기존 Vegas의 방법을 개선하였고, RTO(Retransmission Time Out)가 발생하면 *BaseRTT*를 새로운 *New_BaseRTT*로 갱신하도록 동작하게 된다.

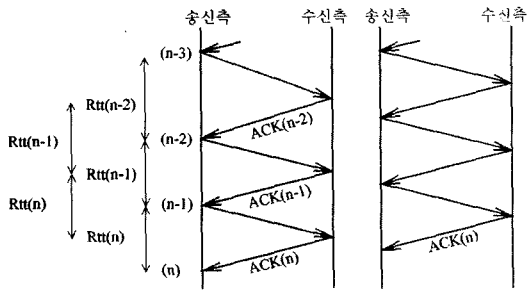
3.2 *rtt* 정보를 이용한 혼잡제어 알고리즘

PowerVegas는 기존의 Vegas가 *Diff* 계산에만 이용했던 *rtt*를 직접적으로 이용하여 두 가지 정보를 얻어낸다. 먼저 가장 최근의 *rtt*와 바로 이전 *rtt*의 차이를 이용해서 *rtt*의 변화정도에 대한 정보를 얻어내고, 다음으로 그림 6과 같은 방법으로 네트워크 혼잡정도의 변화 경향에 대한 정보를 얻어낸다. 차례대로 얻어낸 두 가지 정보를 네트워크의 상태를 판단하는 기준으로 활용해서 기존 Vegas의 세 개의 [V_구간]을 다섯 개의 [P_구간]으로 재 정의한다. 그림에서 $\tau(n)$은 가장 최근에 수신된 n번째 Ack의 시간이고, Rtt(n)은 수신된 두 Ack 간의 시간의 차이, 즉 *rtt*를 의미한다. 그러므로 식 (7)이 성립한다.

$$Rtt(n) = \tau(n) - \tau(n-1) \quad (7)$$

그리고 ΔRtt(n)은 두 *rtt* 간의 변화량을 나타내며 식 (8)과 같이 표현할 수 있다.

$$\Delta Rtt(n) = Rtt(n) - Rtt(n-1) \quad (8)$$



(a) 혼잡정도가 감소하는 변화 경향 (b) 혼잡정도가 일정한 변화 경향

그림 6 네트워크 혼잡정도의 변화 경향

송신측은 Ack를 수신하면, ΔRtt(n)의 크기 그리고 ΔRtt(n)과 ΔRtt(n-1) 간의 차이를 비교해서 얻어낸 두 개의 정보를 기반으로 네트워크의 상태를 판단하여 그림 7의 *rtt* 정보를 이용한 혼잡제어 알고리즘의 동작에 따라 전송할 윈도우 크기를 조절하게 된다.

```
While   < Diff ≤
  if ( Rtt(n) < 0)
     let W = W + 1/W
  else if ( Rtt(n) = 0)
          if ( Rtt(n) ≤  Rtt(n-1)
             let W = W + 1/W
          else ( Rtt(n) >  Rtt(n-1)
             let W = W
  else ( Rtt(n) > 0)
             let W = W
```

```
While Diff >
  if ( Rtt(n) < 0)
     let W = W
  else if ( Rtt(n) = 0)
          if ( Rtt(n) ≤  Rtt(n-1)
             let W = W
          else ( Rtt(n) >  Rtt(n-1)
             let W = W  1
  else ( Rtt(n) > 0)
             let W = W  1
```

그림 7 *rtt* 정보를 이용한 혼잡제어 알고리즘

ΔRtt(n)<0이면, 즉 현재의 *rtt*가 이전의 *rtt*보다 작은 경우에는 네트워크의 혼잡정도가 경감되고 있음을 의미한다. 이러한 경우, 기존의 Vegas는 *Diff* 값에 영향을 미치지 않는 허용범위 내에서의 *rtt* 변화는 무시한다. 그러나 PowerVegas는 그림 5와 같이 $\alpha < Diff \leq \beta$ 구간

에서는 윈도우 크기를 한 개 더 증가시키는 [P_구간-5]를 추가하고, $Diff > \beta$ 구간에서는 윈도우 크기를 감소시키지 않고 그대로 유지하는 [P_구간-6]을 추가시킨다. $Diff \leq \alpha$ 구간은 rtt가 항상 줄어드는 구간이므로 여기서는 [P_구간]을 고려하지 않았다.

ΔRtt(n)=0이면, 즉 현재의 rtt와 이전의 rtt가 같은 경우는 네트워크의 혼잡정도가 이전과 동일함을 의미한다. 이러한 경우, 기존의 Vegas는 네트워크가 안정된 상태로 판단하고 윈도우 크기를 변화 없이 그대로 유지하지만, PowerVegas는 rtt의 변화를 검사하여 네트워크의 혼잡정도가 변화하는 경향을 판단한다. 만약 그림 6(a)와 같이 ΔRtt(n)이 ΔRtt(n-1)보다 작으면 네트워크가 혼잡이 감소하는 방향으로 흐르고 있는 것이고, 그림 6(b)와 같이 ΔRtt(n)과 ΔRtt(n-1)이 같으면 네트워크의 혼잡정도가 더 이상 악화되지 않는 경우라고 판단한다. 그러므로 PowerVegas는 이런 상황에서 $\alpha < Diff \leq \beta$ 구간에서는 윈도우 크기를 한 개 증가시키는 [P_구간-5]를 추가하고, $Diff > \beta$ 구간에서는 윈도우 크기를 감소시키지 않고 그대로 유지하는 [P_구간-6]을 추가시킨다.

ΔRtt(n)>0이면, 즉 Rtt(n)이 Rtt(n-1)보다 큰 경우에는 네트워크의 혼잡정도가 심화되고 있다는 것을 의미한다. 이 때는 [P_구간-1, 2, 3]만을 이용하여 기존의 Vegas와 동일하게 동작하게 된다.

3.3 패킷 손실 정보를 이용한 혼잡제어 알고리즘

PowerVegas는 기존의 Vegas가 고려하지 않는 패킷 손실 정보를 이용한 새로운 알고리즘을 추가하였다. 패킷 손실 정보를 이용한 혼잡제어 알고리즘은 3.2절의 rtt 정보를 이용한 혼잡제어 알고리즘에서 정의한 [P_구간]들 중에서 [P_구간-5]와 [P_구간-6]을 확장하면서 새로운 [P_구간-4]를 추가시킨다. 패킷 손실 정보를 이용한 혼잡제어 알고리즘은 그림 8에서 정의한 인자들을 사용하여 네트워크의 상태를 판단한다. 그림 8에서 $W_{exp}(n)$은 $BaseRTT$ 동안 전송될 것으로 기대되는 윈도우 크기의 기대값이고, $W_{act}(n)$은 rtt 동안 실제로 전송된 윈도우 크기의 실제값이다. 그리고 $\Delta W_{exp}(n)$은 윈도우 크기의 기대값의 기대변화량을 나타내고 식 (9)와 같이 계산된다.

$$\Delta W_{exp} = W_{exp}(n) - W_{exp}(n-1) \quad (9)$$

그리고 $\Delta W_{act}(n)$은 윈도우 크기의 실제값의 실제변화량을 나타내며 식 (10)과 같이 계산된다.

$$\Delta W_{act} = W_{act}(n) - W_{act}(n-1) \quad (10)$$

송신측은 Ack를 수신하면, 윈도우 크기의 기대변화량 $\Delta W_{exp}(n)$을 검사해서 얻어낸 정보를 기반으로 네트워크의 상태를 판단하여 그림 9 패킷 손실 정보를 이용한 혼잡제어 알고리즘의 동작에 따라 전송할 윈도우 크기

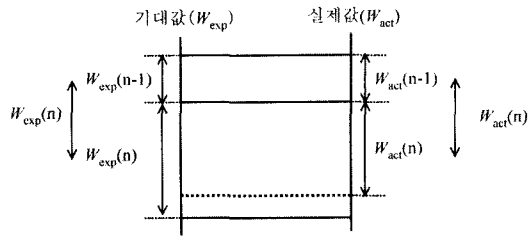


그림 8 패킷 손실 정보를 이용한 혼잡제어 알고리즘을 위한 인자들의 정의

```
While Diff ≤
  if ( W_act(n) ≤ 0)
    let W = W + 2/W
  else
    let W = W + 1/W

While  < Diff ≤
  if ( W_exp(n) ≥ 0)
    if ( W_act(n) ≤ 0)
      let W = W + 1/W
    else
      let W = W
  else
    let W = W

While Diff >
  if ( W_exp(n) = 0)
    if ( W_act(n) < 0)
      let W = W
    else
      let W = W - 1
  else
    let W = W - 1
```

그림 9 패킷 손실 정보를 이용한 혼잡제어 알고리즘

를 조절한다.

첫 번째 구간인 $Diff \leq \alpha$인 구간은 윈도우 크기를 한 개 증가시키는 구간이다. 이 구간에서는 현재의 윈도우 크기의 기대값이 이전의 윈도우 크기의 기대값보다 항상 더 크므로 기대변화량은 검사할 필요가 없다. 이때 실제로 전송된 윈도우 크기의 차이인 실제변화량이 0보다 작거나 같다면, 전송 중에 혼잡과 관계가 없는 패킷 손실이 발생한 것이므로 이에 대한 보상을 해주어야 한다. 이를 위해 그림 5와 같이 전송할 윈도우 크기를 한 개 더 증가시키는 [P_구간-4]를 추가하였다.

두 번째 구간인 $\alpha < Diff \leq \beta$인 구간은 윈도우 크기를 변화시키지 않고 유지하거나 3.2절의 rtt 정보를 이용한 혼잡제어 알고리즘에 의해서 윈도우 크기를 한 개 증가시키는 구간이다. 이 구간에서는 먼저 기대변화량이 1이거나 0인지를 검사한 후에 실제변화량을 검사하여 두 값을 비교한다. 즉 기대값은 증가했는데 실제값은 증가하지 않은 경우이거나, 기대값은 같은데 실제값은 감소한 경우에 혼잡과 관계가 없는 패킷 손실이 발생한 것으로 판단한다. 그러므로 $\Delta W_{exp}(n)=1$이고 $\Delta W_{act}(n)=0$인 경우, 또는 $\Delta W_{exp}(n)=0$이고 $W_{act}(n)<0$이면, 전송할 윈도우 크기를 한 개 증가시키는 [P_구간-5]의 유효 범위를 확장하였다.

마지막 구간인 $Diff > \beta$인 구간은 윈도우 크기를 한 개 감소하거나 앞서 기술한 rtt 정보를 이용한 혼잡제어 알고리즘에 의해서 윈도우 크기를 유지하는 구간이다. 이 구간에서는 먼저 기대변화량이 0인지를 검사한 후에 실제변화량을 검사하여 두 값을 비교한다. 즉 기대값은

동일한데 실제값은 감소한 경우에 혼잡과 관계가 없는 패킷 손실이 발생했다고 판단한다. 그러므로 $\Delta W_{exp}(n)=0$ 일 때 $\Delta W_{act}(n)<0$이면, 전송할 윈도우 크기를 그대로 유지하는 [P_구간-6]의 유효 범위를 확장하였다.

이상의 세 가지 기법을 추가하여 기존 Vegas의 불공정성 문제를 개선한 PowerVegas의 윈도우 크기의 변화는 그림 10과 같은 모습을 나타내게 된다.

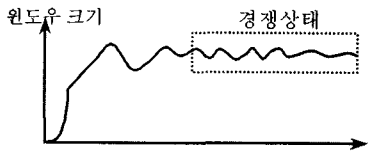


그림 10 PowerVegas의 윈도우 크기의 변화

제안한 PowerVegas는 기존의 Vegas가 네트워크의 상태에 따라 윈도우 크기를 조절하다가 특정값으로 수렴하는 것과 다르게 윈도우 크기를 특정값으로 수렴하지 않고 가용대역폭을 찾아 경쟁을 계속하게 된다. 이와 같이 경쟁력이 강화된 PowerVegas는 기존 Vegas의 가장 큰 문제점으로 지적되고 있는 Reno와의 경쟁에서 공정성을 향상시킬 수 있었다. 게다가 기존의 Vegas와 다르게 PowerVegas와 PowerVega 플로우 간에도 공정성을 이룰 수 있었다.

4. 실험 및 성능 평가

새로 제안한 TCP PowerVegas의 성능 평가를 위해서 본 장에서는 LBNL(Lawrence Berkely National Laboratory)의 ns(network simulator)를 사용하여 시뮬레이션하였다[15].

4.1 실험 환경

제안한 PowerVegas의 성능을 평가하기 위해서 먼저 그림 11과 같은 환경을 구성하여 실험을 하였다.

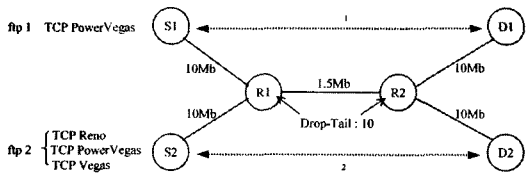


그림 11 실험 환경

그림 11의 S1과 D1 사이에 PowerVegas 기반의 FTP를 이용한 실험 트래픽을 우선 전송하고 나서, 3에서 7초 사이에 S2에서 D2로 다양한 백그라운드 트래픽을 발생시키는 시나리오를 구성하였다. 그리고 R1과 R2 링크 사이에서 일어나는 패킷의 흐름을 모니터링하였다.

4.2 PowerVegas와 Reno 플로우 간의 공정성 실험

Vegas는 네트워크의 상태를 세 가지로 구분하고, 해당 구간에 따라 전송률을 조절한다. 그러나 각 구간의 유효범위가 크기 때문에 네트워크의 변화에 민감하지 못한 단점이 있었다. 이러한 Vegas의 근본적인 문제점은 Reno와의 경쟁에서 불공정성 문제를 유발하였다.

PowerVegas는 Vegas의 [V_구간]을 새로운 기준에 따라 [P_구간]으로 세분화하여 네트워크의 변화에 보다 빨리 대처할 수 있다. 또한 한 구간에 오랫동안 머물 가능성을 적게 함으로써 윈도우 크기가 항상 일정한 값으로 수렴하는 Vegas의 특성을 보완하였다. 그러므로 평형상태에서의 PowerVegas의 윈도우 크기는 기존의 Vegas와 달리 작은 유효범위 내에서 계속 진동하면서 가용대역폭의 발생을 감시하게 된다. 결과적으로 가용대역폭에 대한 경쟁력이 기존의 Vegas에 비해서 크게 강화되었다.

그림 12는 PowerVegas와 Reno 플로우의 공정성에 대한 실험 결과이다. 실험 결과 PowerVegas 플로우는 Reno 플로우가 유입되면 전송량을 반으로 줄인 후, Reno와 거의 균등하게 네트워크의 대역폭을 양분하면서 경쟁함을 확인할 수 있다. 그러므로 Vegas의 불공정성 문제가 크게 개선되었음을 확인할 수 있다.

그림 13은 S2와 D2 사이에 Vegas 기반의 FTP 응용을 새로 추가하여 PowerVegas, Reno 그리고 Vegas 간의 공정성을 비교한 실험이다. 실험 결과 Vegas 플로우는 상대적으로 경쟁력이 약하기 때문에 윈도우 크기와 전송량이 계속 감소함을 확인할 수 있다. 반면에, PowerVegas 플로우는 초기에 Vegas와 경쟁할 때는 기존의 Vegas의 기본적인 특징을 가지고 있으므로 경쟁하는 Vegas 플로우의 특성에 따라 어느 정도 일정한 윈도우 크기를 유지하지만 Reno 플로우가 유입되어 경쟁하는 상황이 되면 Reno의 특성에 따라 윈도우 크기가 경쟁적으로 변화하게 된다. 즉, PowerVegas 플로우는 Reno 플로우의 주기와 거의 유사한 주기를 가지고 윈도우 크기를 조절하는 . 진동을 갖게 된다. 그러나 Reno 플로우에 비해서 진동폭은 매우 작으며 상대적으로 더 안정된 상태를 유지하게 된다. 결과적으로 Reno 플로우가 유입된 후의 기존 Vegas 플로우는 경쟁에서 뒤쳐져 네트워크의 자원을 거의 사용하지 못하게 되는 반면에 PowerVegas 플로우는 Reno 플로우와 대등하게 경쟁하는 모습을 확인할 수 있다.

4.3 PowerVegas와 PowerVegas 플로우 간의 공정성 실험

본 실험은 *rtt*가 서로 다른 두 Vegas 플로우 간의 경쟁에서 *rtt*가 작은 플로우에 나타났던 불공정성 문제가 PowerVegas 혼잡제어 알고리즘에 의해서 개선된 것을

보이기 위한 것이다. 그림 14의 W_1-W_2 그래프에서 제안한 PowerVegas 알고리즘에 의해서 이러한 Vegas의 불공정성 문제가 크게 개선되었음을 확인할 수 있다.

기존의 Vegas에서 나타났던 *rtt*가 작은 플로우가 *rtt*가 큰 플로우에 비해서 상대적으로 더 낮은 전송량을 갖게 되는 불공정성 문제는 2.2.2절에서 설명했듯이 *BaseRTT*에 지나치게 의존하는 Vegas의 근본적인 문제 때문이다. 그러나 PowerVegas는 현재의 네트워크 상태를 반영하는 *rtt*의 비중을 높이고 기존의 구간을 세분화함으로써 이러한 문제를 해결하였다. 그림 14에서 볼 수 있듯이, 네트워크 상태점 ($W_1(t)$, $W_2(t)$)이 (W_1^1, 1)에서 시작해서 공정성 기준선상으로 이동한 후 특정점으로 수렴하지 않고 경쟁하게 함으로써 *rtt*가 작은 플로우에 나타났던 불공정성 문제를 개선할 수 있었다.

4.4 PowerVegas와 Lai's Vegas의 공정성 비교

본 실험은 같은 *rtt* 환경에서 2.3절에서 언급한 Lai의 Vegas 튜닝과 제안한 PowerVegas의 공정성을 비교한 것이다. 그림 15(a)는 Reno와 Lai의 튜닝 Vegas의 결과이다. Lai의 Vegas는 최적의 α, β로 (5, 7)를 선택하여 기존의 Vegas에 비해 어느 정도는 전송률 및 공정성을 향상 시켰다. 그러나 그림 (b)의 PowerVegas와 Reno의 결과와 비교하여 보면 전체적인 전송률과 공정성이 상대적으로 낮음을 확인할 수 있다.

그러므로 PowerVegas는 Lai의 Vegas보다 공정성이 향상된 보다 경쟁력 있는 혼잡 제어 알고리즘임을 확인할 수 있다.

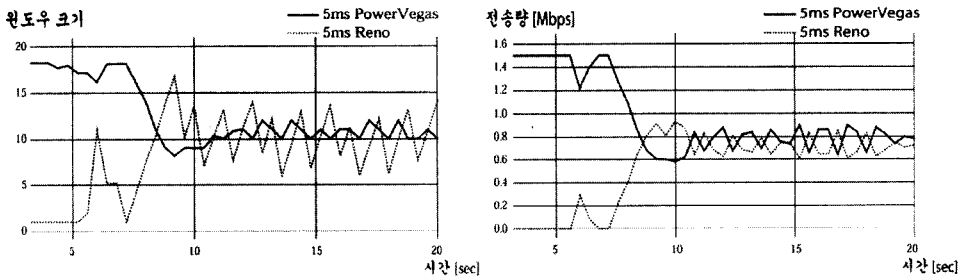


그림 12 PowerVegas와 Reno 플로우 간의 공정성 실험

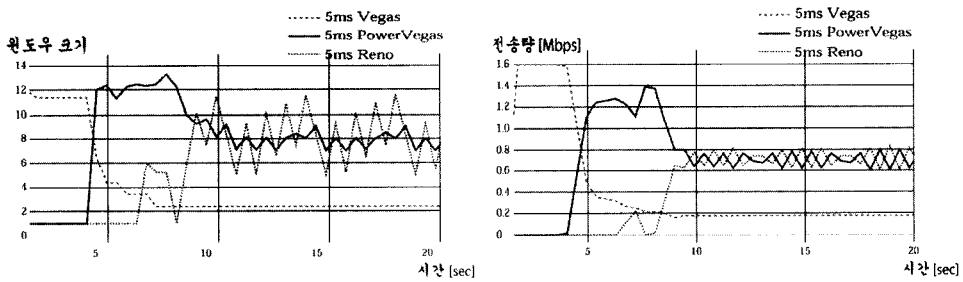


그림 13 Vegas, PowerVegas 그리고 Reno 플로우 간의 공정성 실험

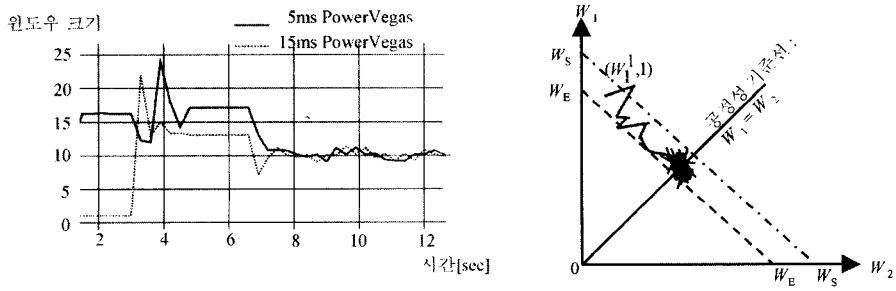
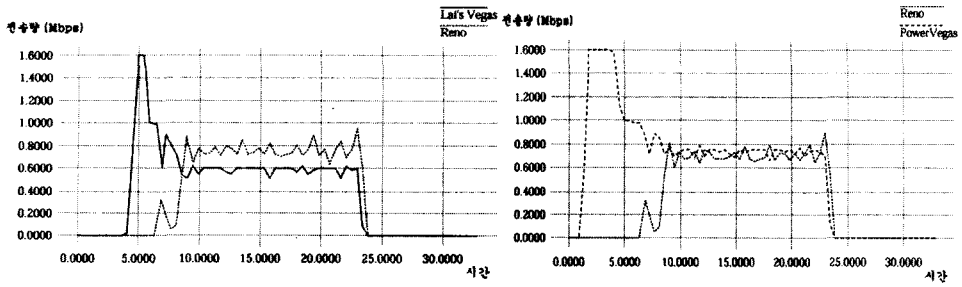


그림 14 *rtt*가 서로 다른 두 PowerVegas 플로우 간의 공정성 실험과 W_1-W_2 그래프



(a) Reno vs Lai's Vegas (b) Reno vs PowerVegas

그림 15 PowerVegas와 Lai's Vegas의 공정성 비교

5. 결론 및 향후 과제

현재 인터넷의 주요 TCP 버전인 Reno는 패킷 손실을 인지한 후 네트워크의 상태를 판단하는 수동적인 혼잡제어 방법을 사용하고 있다. Reno의 수동적인 혼잡제어 방법은 네트워크의 혼잡을 심화시키는 원인이 된다. 이러한 Reno의 문제점을 개선하기 위해 Brakmo와 Peterson에 의해 제안된 새로운 혼잡제어 알고리즘인 TCP Vegas는 Reno에 비해 우수한 성능을 가짐이 증명되었음에도 불구하고 두 가지 심각한 불공정성 문제를 가지고 있기 때문에 범용적으로 사용되지 못하고 있다.

본 논문에서는 기존 Vegas의 불공정성 문제를 개선한 TCP PowerVegas 혼잡제어 알고리즘을 제안하였다. *rtt*만을 기반으로 네트워크의 혼잡을 제어하는 기존의 Vegas에 비해서 제안한 PowerVegas는 *rtt* 정보와 패킷 손실 정보를 유기적으로 결합시킨 새로운 기법으로 혼잡제어를 수행한다. 그러므로 PowerVegas는 기존의 Vegas가 Reno와 경쟁할 때 발생하는 불공정성 문제와 *rtt*가 서로 다른 Vegas 플로우들 간의 경쟁에서 *rtt*가 작은 플로우에 나타났던 불공정성 문제를 모두 효과적으로 개선할 수 있다. 본 논문에서는 ns 시뮬레이터를 이용한 실험을 통해서 PowerVegas와 Vegas 및 Reno 그리고 Lai의 Vegas 간의 안정성과 공정성에 대한 성능을 비교 분석하였다. 실험 결과 제안한 PowerVegas가 기존의 Vegas 및 Lai의 Vegas에 비해서 공정성이 크게 향상됨을 확인할 수 있었다.

향후 연구 과제로는 제안한 PowerVegas와 새로운 TCP-Friendly 혼잡제어 알고리즘인 TFRC나 TEAR와의 경쟁에서도 공정성을 개선할 수 있는 방법에 대한 연구가 수행되어야 하고, RED과 같은 AQM(Active Queue Management) 알고리즘들과의 연동에 대한 연구도 같이 수행되어야 할 것이다.

참 고 문 헌

[1] K. Fall and S. Floyd, "Simulation-based Comparisons of Tahoe, Reno, and SACK TCP," *Proceeding of ACM SIGCOMM '96*, pp. 5-21, July 1996.

[2] V. Jacobson and M. Karels, "Congestion Avoidance and Control," *Proceeding of ACM SIGCOMM '88*, pp. 314-319, August 1988.

[3] V. Jacobson, "Modified TCP Congestion Avoidance Algorithm," *LBNL Technical Report*, April 1990.

[4] L. Brakmo, S. O'Malley and L. Peterson, "TCP Vegas: New Techniques for Congestion Detection and Avoidance," *Proceeding of ACM SIGCOMM '94*, pp. 24-35, August 1994.

[5] L. Brakmo and L. Peterson, "TCP Vegas: End-to-end Congestion Avoidance on a Global Internet," *IEEE Communication*, pp. 1465-1480, October 1995.

[6] J. Mo, R. La and J. Walrand, "Analysis and Comparison of TCP Reno and TCP Vegas," *Proceeding of IEEE INFOCOM '99*, pp. 1556-1563, March 1999.

[7] A. Vendictis and A. Baiocchi, "Modeling a Mixed TCP Reno and TCP Vegas Scenario," *Proceeding of IFIP 2002*, pp. 612-623, May 2002.

[8] T. Bonald, "Comparison of TCP Reno and The Vegas: Efficiency and Fairness," *Proceeding of Elsevier Science Publisher B. V.*, pp. 307-332, August 1999.

[9] O. Hellal and E. Altman, "Analysis of TCP Vegas and TCP Reno," *Proceeding of ICC '97*, pp. 495-499, June 1997.

[10] Y. Lai, "Improving the Performance of TCP Vegas in a Heterogeneous Environment," *Proceeding of ICPADS 2001*, pp. 581-587, June 2001.

[11] T. Henderson and E Sahouria, "On Improving the Fairness of TCP Congestion Avoidance," *Proceeding of IEEE GLOBECOM '98*, pp. 539-544, November 1998.

[12] G. Hasegawa, M. Murata and H. Miyahara, "Fairness and Stability of Congestion Control Mechanisms of TCP," *Proceeding of IEEE INFOCOM '99*, pp. 1329-1336, March 1999.
[13] Y. Lai, "Improving the Performance of TCP Vegas in a Heterogeneous Environment," *Proceeding of IEEE ICPADS 2001*, pp. 581-587, June 2001.
[14] C. Boutremans and J. Boudec, "A Note on Fairness of TCP Vegas," *Proceeding of Broadband Communications*, pp. 163-170, March 2000.
[15] The Network Simulator ns-2, http://www.isi.edu/nanam/ns/

오 민 철
2001년 광운대학교 전자통신공학과 학사
2001년~현재 광운대학교 전자통신공학과 석사 과정

송 병 훈
1998년 광운대학교 컴퓨터 과학과 학사
2000년 광운대학교 전자통신공학과 석사
2000년~현재 광운대학교 전자통신공학과 박사 과정

정 광 수
정보과학회논문지 : 정보통신
제 31 권 제 1 호 참조