TCP Vegas에서 공정성 향상을 위한 혼잡제어 알고리즘

(A New Congestion Control Algorithm for Improving Fairness in TCP Vegas)

이 선 헌 † 송 병 훈 †† 정 광 수 †††
(Sunhun Lee) (Byunghoon Song) (Kwangsue Chung)

요 약 인터넷의 안정성에 영향을 미치는 요소로 종단간에 이루어지는 TCP 혼잡제어가 있다. 현재 인터넷의 주요 TCP 버전인 Reno가 사용하는 수동적인 혼잡제어 방법은 네트워크의 혼잡을 심화시키는 원인이 된다. 그러나 이러한 Reno의 문제점을 개선하기 위해 제안된 Vegas는 Reno에 비해 우수한 성능을 가짐이 이전의 관련 연구에서 증명되었음에도 불구하고 세 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 NewVegas 혼잡제어 알고리즘을 제안한다. 제안한 NewVegas는 병목구간 라우터에서 큐잉되는 패킷의 편차를 사용하여 기존 Vegas의 불공정성 문제를 효과적으로 해결한다. 제안한 알고리즘의 성능을 검증하기 위해 NewVegas와 Reno 및 기존의 Vegas를 비교하는 실험을 수행하였다. 실험 결과를 통해서 제안한 NewVegas가 기존 Vegas의 혼잡제어 방법에 비해 우수한 성능을 보일 뿐만 아니라, Vegas의 불공정성 문제도 크게 개선되었음을 확인할 수 있었다.

키워드 : TCP Vegas, 혼잡제어, 공정성

Abstract An 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 the network congestion worse. Brakmo and Peterson proposed a congestion control algorithm, TCP Vegas, by modifying the congestion avoidance scheme of TCP Reno. Many studies indicate that Vegas is able to achieve better throughput and higher stability than Reno. But there are three unfairness problems in Vegas. These problems hinder the spread of Vegas in the current Internet. In this paper, in order to solve these unfairness problems, we propose a new congestion control algorithm called TCP NewVegas. The proposed NewVegas is able to solve these unfairness problems effectively by using the variation of the number of queued packets in a bottleneck router. To evaluate the proposed approach, we compare the performance among NewVegas, Reno and Vegas. Through the simulation, NewVegas is shown to be able to achieve throughput and better fairness than Vegas.

Key words : TCP Vegas, Congestion control, Fairness

1. 서 론

TCP(Transmission Control Protocol)의 혼잡제어 알고리즘(Congestion control algorithm)은 1988년의 TCP Tahoe 이래로 1990년 TCP Reno, 1995년 TCP SACK에 이르기 까지 다양하게 연구되어 왔다[1]. 이러한 TCP 혼잡제어 알고리즘의 주요 목적은 송신단의 전송률을 제어하여 혼잡상황으로 인해 발생되는 데이터의 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]. 이러한 Reno의 혼잡제어 방

· 본 연구는 정보통신부 및 정보통신연구진흥원의 대학 IT연구센터 육성·지원사업의 연구결과로 수행되었음.
† 비 회 원 : 광운대학교 전자통신공학과
sunlee@cclab.kw.ac.kr
†† 비 회 원 : 전자부품연구원 지능형정보시스템 연구센터 연구원
bhsong@keti.re.kr
††† 종신회원 : 광운대학교 전자공학부 교수
kchung@kw.ac.kr
논문접수 : 2004년 5월 24일
심사완료 : 2005년 4월 8일

법은 패킷 손실에 대한 정보에만 입각해서 네트워크의 상태를 유추하기 때문에 급격한 전송률의 변화가 빈번히 발생하게 되는 문제점을 지니고 있다. 또한, 네트워크에 주기적인 혼잡상황을 야기시키며 공존하는 다른 TCP 플로우들의 전송률에 민감하게 영향을 주는 문제점도 가지고 있다[3]. 최근에 이러한 Reno의 혼잡제어 방법의 근본적인 문제를 해결하기 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 Vegas는 Reno의 문제점이었던 다른 TCP 플로우의 전송률에 미치는 영향도 상대적으로 매우 작다[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 문제는 실제 Vegas와 Reno를 동일한 네트워크 환경에서 경쟁시키면, Reno의 전송률의 변화에 따른 영향으로 경쟁하는 Vegas의 전송률이 현저하게 감소하는 현상이 나타난다는 것이다. 이러한 문제를 불공정성 문제라 한다. 또한 Vegas 플로우 간의 경쟁에서도 두 가지의 심각한 문제점이 발생한다. 첫 번째는 RTT가 서로 다른 Vegas 플로우의 경쟁에서 RTT가 작은 플로우가 RTT가 큰 플로우에 비해서 더 낮은 전송률을 갖게 되는 불공정성 문제이며 두 번째는 연결 시점이 다른 Vegas 플로우 간의 경쟁에서 이전에 연결을 맺은 플로우가 이후에 연결을 맺은 플로우에 비해서 더 낮은 전송률을 갖게 되는 불공정성 문제이다. 그러므로 Vegas가 신뢰성 있는 전송 프로토콜로 자리잡기 위해서는 이러한 세 가지 문제점을 보완하는 것이 매우 중요한 기술적 과제로 인식되어지고 있다[10-13].

본 논문에서는 이러한 Vegas의 문제점을 해결하기 위해서 새로운 TCP NewVegas 혼잡제어 알고리즘을 제안한다. 제안한 NewVegas는 Vegas의 장점인 전송률의 안정성을 그대로 유지하면서도 Reno와의 경쟁에 매우 적극적으로 대응하면서 불공정성 문제를 해결한다. 또한 Vegas 플로우 간의 불공정성 문제도 효과적으로 해결하였다.

본 논문의 2장에서는 TCP Vegas 혼잡제어 알고리즘과 공정성에 대하여 분석하였고, 3장에서는 새로 제안한 NewVegas 혼잡제어 알고리즘을 기술하였다. 4장에서는 시뮬레이터를 이용하여 제안한 알고리즘을 검증하고 NewVegas의 공정성을 분석하였으며, 마지막으로 6장에서는 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 RTT를 측정하여 이를 기반으로 윈도우 크기를 조절하는 새로운 혼잡제어 알고리즘을 사용한다. 본 장에서는 Vegas 혼잡제어 알고리즘의 기본적인 원리와 현재 지적되고 있는 Vegas 혼잡제어 알고리즘의 문제점에 대해서 기술하고자 한다.

2.1 TCP Vegas 혼잡제어 알고리즘

Vegas는 패킷 손실에 대한 정보만으로 네트워크의 혼잡제어를 수행하는 Reno와 다르게 측정된 RTT를 기반으로 네트워크의 혼잡제어를 수행한다. 즉, RTT가 커지면 네트워크의 혼잡정도가 증가되었다고 판단하여 윈도우 크기를 감소시키고, 반대로 RTT가 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.

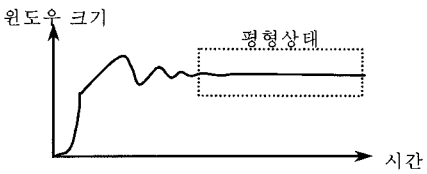


그림 1 Vegas의 윈도우 크기 변화

이와 같은 Vegas의 윈도우 크기 변화는 Reno에 비해 진동폭이 작고 전체적으로 안정된 상태를 유지시킬 수 있는 특징이 있다. 이러한 Vegas의 혼잡제어 알고리즘

의 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 첫번째 TCP 세그먼트에 대한 *RTT* 측정값이 *baseRTT*가 된다.

ii) 기대전송률인 *Expected*를 구한다.

i)에서 구한 *baseRTT* 동안에 전송할 수 있는 윈도우의 크기를 *WindowSize*라고 하고, 이 플로우에 대해 기대할 수 있는 기대전송률인 *Expected*를 다음의 식 (1)과 같이 구한다.

$$Expected = WindowSize / baseRTT \quad (1)$$

iii) 실제전송률인 *Actual*을 구한다.

측정된 *RTT*동안 실제로 전송한 이 플로우의 실제전송률인 *Actual*은 *WindowSize*를 현재의 *RTT*로 나누어 구한다. $baseRTT \leq RTT$ 이므로 $Actual \leq Expected$의 관계가 항상 성립한다.

$$Actual = WindowSize / RTT \quad (2)$$

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.

$$Diff = \left(\frac{WindowSize}{baseRTT} - \frac{WindowSize}{RTT}\right) \times baseRTT \quad (3)$$

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 데이터, 즉 라우터의 큐를 차지하는 패킷의 양을 α와 β의 값 사이로 제한하려고 노력한다. 일반적으로 α, β의 기본값은 α=1, β=3이다.

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

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

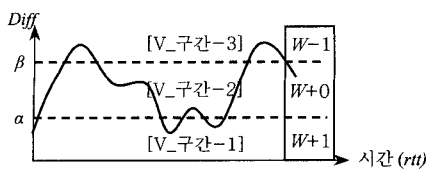


그림 2 Vegas에 의해 정의된 [V_구간]

그림 2의 각 구간은 iv)에서 구한 *Diff*의 변화 정도를 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 혼잡 정도를 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 RTT기반 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 예방하는 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 플로우가 공존하는 경우와 RTT가 서로 다른 Vegas 플로우들이 공존하는 경우 그리고 이전에 연결된 Vegas 플로우가 있는 상태에서 새로운 Vegas 플로우가 네트워크에 유입되었을 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 플로우

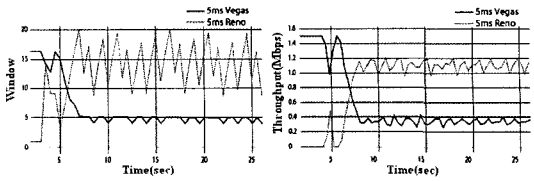


그림 3 Vegas와 Reno 플로우 간의 공정성 비교

의 윈도우 크기와 전송량의 변화를 나타낸 것이다.

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

우선 Reno 플로우가 패킷을 전송하기 시작하면서 Vegas 플로우의 RTT는 점차 증가하게 된다. RTT가 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 늘려나간다. 그러므로 Reno 플로우가 패킷 손실로 인해서 윈도우 크기를 일시적으로 감소시킬 때까지 Vegas 플로우의 윈도우 크기는 계속해서 감소할 수밖에 없게 된다. 이러한 상황에서 Reno 플로우가 감소한 윈도우 크기를 다시 증가시키는 동안 Vegas 플로우의 RTT도 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 서로 다른 RTT를 가지는 Vegas 플로우 간의 불공정성 문제

Reno에서는 RTT가 큰 플로우가 RTT가 짧은 플로우에 비해 불리하게 동작한다. 그러나 Vegas의 경우에는 RTT가 작은 플로우가 더 불리한 모습을 나타낸다. 그림 4에서 RTT가 더 큰 Vegas 플로우가 상대적으로 더 큰 윈도우 크기를 유지하는 모습을 확인할 수 있다. 이러한 현상은 Vegas의 전송량이 [*W/RTT*]에 비례하기 때문이다. 그러므로 RTT가 서로 다른 두 플로우가 경쟁하는 경우, Vegas는 Reno와 반대로 RTT가 짧은 플로우의 전송률이 상대적으로 더 낮게 되는 불공정성 문제가 발생하게 된다[12]. 이와 같이 서로 다른 RTT를 가지는 Vegas 플로우 간의 경쟁에서 나타나는 불공정성 문제의 근본적인 원인은 가능한 일정한 전송률을 유지하려하는 Vegas의 특징으로 인해서 나타나는 경쟁력의 약화에 있다.

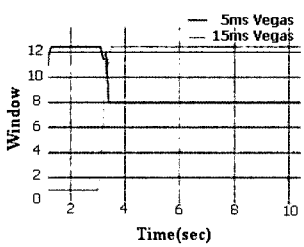


그림 4 RTT가 서로 다른 두 Vegas 플로우 간의 공정성 비교

2.2.3 연결 시점이 다른 Vegas 플로우 간의 불공정성 문제

Reno에서는 두 플로우의 연결 시점이 다르더라도 윈도우의 변화가 주기적인 특성으로 인해 일정 시간이 지난 후에는 두 플로우의 전송률이 공정하게 된다. 그러나 Vegas의 경우에는 이전에 연결된 플로우의 전송률이 더 낮아지는 불공정성 문제를 나타내게 된다[14].

Vegas 플로우가 혼잡 상황이 발생하지 않은 네트워크에 유입될 경우, *baseRTT*는 네트워크 최소 *RTT*의 근사값으로 결정된다. 윈도우의 증가로 네트워크의 혼잡 상태가 커질수록 *RTT*는 증가하게 되며 따라서 *RTT*에 대한 *baseRTT*의 비율, [*baseRTT/RTT*]는 감소하게 된다. 혼잡 상황이 커진 상태에서 새로운 Vegas 플로우가 유입될 경우, 새로운 플로우의 *baseRTT*는 이전 플로우의 *baseRTT*에 비해 높은 값을 가지게 된다. 두 플로우의 RTT는 근사한 값을 가지므로 새로 유입된 플로우의 [*baseRTT/RTT*]의 비는 이전 플로우의 [*baseRTT/RTT*]에 비해 높은 값으로 결정된다.

현재의 윈도우를 W라고 하면 식 (3)은 식 (5)의 형태로 치환할 수가 있다. 식 (5)를 통해 새로운 플로우는 이전 플로우에 비해 높은 [*baseRTT/RTT*] 비율을 가지며 상대적으로 낮은 *Diff*값을 갖게 되는 것을 확인할 수 있다. 따라서 새로운 플로우는 식 (4)의 혼잡 제어 알고리즘에 의해 윈도우를 증가시킬 수 있는 기회를 이전 플로우에 비해 상대적으로 많이 갖게 되어 높은 전송률을 가지게 된다.

$$Diff = W \times \left(\frac{RTT - baseRTT}{RTT}\right) \quad (5)$$

2.1절에서 *baseRTT*는 혼잡이 없을 때의 *RTT*라고 정의하였다. 네트워크의 혼잡 상황이 크지 않은 경우, 연결 시점이 다른 플로우들 각각의 *baseRTT*는 비슷하지만 혼잡 상황이 커질수록 플로우들의 *baseRTT*의 차이는 커지게 되고 이러한 차이가 두 플로우의 불공정성을 유발하는 원인이 된다.

연결 시점이 다른 Vegas 플로우 간의 불공정성과 관련한 문제로 "Persistent congestion"이 있다. 새로 유입된 플로우는 부정확한 *baseRTT* 값을 가지므로 네트워크의 대역폭을 실제 대역폭보다 더 큰 것으로 간주하게 되어 전송률을 높이게 된다. 새로운 플로우들이 계속 유입될 경우, 이러한 경향이 심화되어 결국 네트워크의 혼잡 상태가 지속되는 문제점을 가지게 된다.

2.3 기존의 TCP Vegas 불공정성 문제에 관한 연구

기존의 Vegas의 불공정성 문제 해결에 대한 접근은 크게 AQM(Active Queue Management)과 Vegas의 파라미터의 조절 그리고 Vegas의 혼잡제어 알고리즘을 수정하는 방법들로 구분할 수 있다. 최근에는 Vegas의 불공정 문제를 해결하기 위한 연구들이 대부분 Vegas 파

라미터의 튜닝과 혼잡제어 알고리즘을 수정하는 기법을 동시에 고려하고 있다. 가장 최근에 연구가 진행된 Vegas-A와 Vegas+에서는 이러한 특징을 잘 보여 주고 있다[10,15-18].

2.3.1 Vegas-A 연구

Vegas의 조심스런(conservative) 혼잡제어 방법을 공격적으로 수정하여 Reno와의 경쟁에서 공정성을 향상시키기 위해 제안된 방법으로 2003년에 K. Srijith에 의해 제안된 Vegas-A가 있다. Vegas-A는 2.1절에서 설명한 실제 전송률인 *Actual*의 차이를 이용해서 Vegas의 성능을 제한하는 고정된 파라미터인 α, β를 가변적으로 변화시키는 혼잡제어 알고리즘을 사용한다. 즉, 기존 Vegas에서 너무 낮은 값으로 고정된 파라미터인 α, β를 가변적으로 변경하는 방법을 통해 불공정성 문제를 해결하고자 하였다. Vegas-A는 기존 Vegas에 비해 Reno와의 불공정성이나 Vegas 플로우 간의 불공정성 문제를 개선하였으나 근본적인 문제를 해결하기엔 미흡하다고 할 수 있다. 또한 α, β를 가변적으로 변화시킴에 의해 Vegas의 장점인 전송률의 안정성을 잃게 되는 단점을 가진다[17].

2.3.2 Vegas+ 연구

앞 절에서 설명한 Vegas-A와 함께 공정성 향상을 위해 제안된 방법으로 Hasegawa의 Vegas+가 있다. Vegas+는 Reno와의 경쟁을 위해 두 가지 모드를 설정하게 된다. RTT가 커질때마다 증가시키는 새로운 파라미터인 *count*를 추가하여 *count*가 임계값인 $count_{max}$에 이르면 Vegas의 혼잡제어 알고리즘 적용모드에서 Reno의 혼잡제어 알고리즘 적용모드로 변경하여 불공정성 문제를 해결하도록 하였다. 하지만 이러한 방법은 임계값인 $count_{max}$에 따라 알고리즘의 성능이 결정된다는 문제점과 함께 Reno의 혼잡제어 알고리즘을 사용하는 구간에서 전송률의 변화가 심하게 일어나게 되는 문제점을 갖게 된다[18].

3. TCP NewVegas 혼잡제어 알고리즘

본 절에서는 2.2절에서 지적한 기존의 Vegas 혼잡제어 알고리즘의 불공정성 문제들을 개선하기 위해서 새롭게 제안한 NewVegas 혼잡제어 알고리즘의 특징을 다음과 같이 3가지로 분류하여 기술한다.

i) 새로운 파라미터 설정.

ii) 새로운 파라미터, $Q_{variation}$.

iii) NewVegas의 혼잡제어 알고리즘.

3.1 새로운 파라미터 설정

Vegas 혼잡제어 알고리즘은 병목 현상이 발생하는 라우터 큐에서 Vegas 플로우에 의해 점유되는 큐의 크기를 일정한 값으로 제한하여 네트워크의 안정한 상태를 유지하려는 특성을 가진다. 이러한 특성은 전송률의 안정을 유지할 수 있다는 이점이 있는 반면 Reno와의 경쟁시 Vegas 플로우의 전송률을 낮게 제한하는 단점도 가지고 있다.

이전의 선행 연구를 통해 기존 Vegas의 고정된 파라미터 α, β, γ가 너무 낮은 값인 α=1, β=3, γ=1로 설정되어 있어서 네트워크의 가용한 대역폭을 효과적으로 사용하지 못한다는 것이 증명되었다. 그리고 이러한 기본 설정은 Reno와 경쟁하는 상황에서도 Vegas의 성능을 저하시키는 요인으로 작용한다. 따라서 NewVegas에서는 이러한 제한을 극복하고 Vegas의 성능 향상을 위해 실험을 통해 새로운 파라미터 값인 α=5, β=6, γ=5를 사용하게 된다[10].

3.2 파라미터 $Q_{variation}$

식 (1)과 식 (2)를 통해 Vegas는 기대전송률인 *Expected*와 실제전송률인 *Actual*을 계산한다. 그리고 *Expected*와 *Actual*의 차이인 *Diff*를 식 (3)이나 식 (5)를 이용하여 구하게 된다. 두 전송률의 차이인 *Diff*는 플로우가 지나는 전체 경로에서, 병목 구간에 있는 라우터에서의 큐잉된 플로우의 패킷 수를 의미한다.

기존 Vegas의 혼잡제어 알고리즘에서는 매 RTT주기마다 *Diff*를 구하고 이를 이용해서 식 (2.4)와 같은 혼잡제어 알고리즘을 적용하게 된다. 하지만 Vegas는 *Diff*를 구하는데 있어서 중요한 변수가 되는 *baseRTT*를 부정확하게 판단하는 근본적인 문제점을 가지고 있다. 2.2.3절에서 설명하였듯이 이론적으로 *baseRTT*는 네트워크에 혼잡이 없을 때의 *RTT*가 된다. 혼잡하지 않은 네트워크에 새로운 플로우가 유입될 경우, *baseRTT*는 이전 플로우들의 *baseRTT*와 근사값을 가진다. 하지만 혼잡 상황이 심해진 네트워크에서 새로운 플로우의 *baseRTT*는 이전 플로우의 *baseRTT*에 비해 상대적으로 큰 값을 가지게 되기 때문에 연결 시점이 다른 플로우간에 불공정성 문제를 발생시키는 원인이 된다.

제안한 NewVegas에서는 기존 Vegas의 혼잡제어 알고리즘에서 *Diff*를 구하는데 있어 중요 변수이지만 불공정성을 유발하는 변수인 *baseRTT*에 대한 의존도를 낮추기 위해 새로운 파라미터인 $Q_{variation}$을 사용한다. $Q_{variation}$을 설명하기 전에 $Q_{variation}$을 구하기 위한 변수인 *Target*을 식 (6)과 같이 구할 수 있다. 변수 *Target*은 식 (1)의 기대전송률과는 달리 현재 네트워크 상태에 따른 기대전송률을 의미하며 W는 윈도우의 크기를, *previousRTT*는 이전 RTT주기에서의 *RTT*를 의미한다.

$$Target = \frac{W}{previousRTT} \tag{6}$$

식 (6)을 이용하여 현재 네트워크 상태에서의 기대전송률을 구한 후, 실질적으로 NewVegas의 혼잡제어 알고리즘에 사용될 파라미터인 $Q_{variation}$을 식 (7)과 같이 구한다. 이때 *Actual*은 식 (2)와 마찬가지로 실제전송률

을 의미한다.

$$Q_{variation} = (Target - Actual) \times previous\,RTT$$
$$= (\frac{W}{previous\,RTT} - \frac{W}{RTT}) \times previous\,RTT$$
$$= W \times (\frac{RTT - previous\,RTT}{RTT}) \quad (7)$$

NewVegas에서 혼잡제어 알고리즘에 사용되는 새로운 파라미터, $Q_{variation}$은 하나의 RTT주기 동안에 병목구간의 라우터에서 큐잉되는 패킷의 편차를 의미하게 된다. 식 (7)은 현재의 RTT와 이전 RTT주기에서 구해진 RTT의 차이를 이용하여 $Q_{variation}$을 구하는 방법을 설명하고 있다. 그러므로 $Q_{variation}$이 양의 값을 가지는 경우는 네트워크의 혼잡 상황이 커져서 라우터에 큐잉되는 패킷이 증가했음을 의미하며 음의 값을 가지는 경우는 혼잡 상황이 줄어들어 라우터에 큐잉되는 패킷이 줄어들었음을 의미하게 된다. 그리고 $Q_{variation}$이 0의 값을 가지는 경우는 네트워크가 안정한 상태로서 라우터에 큐잉되는 패킷이 같은 크기로 유지되고 있음을 의미한다.

NewVegas에서 사용되는 새로운 파라미터인 $Q_{variation}$은 $Diff$와는 다른 의미를 가진다. NewVegas는 네트워크 혼잡을 판단하는 중요 기준으로 $baseRTT$에 크게 의존하는 $Diff$값 대신에 새로운 $Q_{variation}$을 사용하므로써 보다 정확한 혼잡 상황 결정을 가능하게 한다. 이를 통해 기존의 Vegas에서 중요하게 사용되지만 부정확한 값을 가지는 변수인 $baseRTT$에 대한 의존성을 줄임으로써 NewVegas는 Vegas 플로우간에 생기는 불공정성 문제를 해결할 수 있게 된다. 또한 $Q_{variation}$에 따라 기존 Vegas의 혼잡제어 알고리즘을 보완함으로써 Reno와의 경쟁에 있어서도 보다 좋은 성능을 가질 수 있게 된다. NewVegas의 새로운 혼잡제어 알고리즘은 다음의 3.3절에서 자세하게 설명하게 된다.

3.3 NewVegas의 혼잡제어 알고리즘

앞선 3.2절에서 NewVegas의 혼잡제어에 사용될 새로운 파라미터 $Q_{variation}$에 대해서 자세하게 설명하였다. 이번 절에서는 $Q_{variation}$을 이용하여 기존의 Vegas의 혼잡제어 알고리즘을 보완한 새로운 혼잡제어 알고리즘에 대해 설명할 것이다.

기존 Vegas의 혼잡제어 알고리즘은 근본적으로 플로우의 패킷이 전달되는 경로상에서 라우터에 큐잉되는 패킷의 수를 일정하게 유지하려는 경향을 지닌다. 하지만 이러한 조심스런 혼잡제어 알고리즘에 의해 Reno와의 경쟁시에 네트워크의 가용한 대역폭을 공평하게 할당받지 못하게 된다. 그림 2의 Vegas가 네트워크의 안정상태를 유지하기 위해 혼잡 윈도우를 증가시키지 못하는 구간인 [V_구간-2]와 [V_구간-3]에서도 Reno 플로우는 패킷 손실이 발생하기 전까지 혼잡 윈도우를 계속적으로 증가시키므로 Vegas가 할당받아야 할 대역폭을 점유하는 결과를 발생하게 된다.

NewVegas에서는 Reno와의 경쟁력 향상을 위해 $Q_{variation}$에 따라 혼잡제어 알고리즘을 차등적으로 적용하게 된다. 병목구간의 라우터에 큐잉되는 패킷의 편차를 의미하는 $Q_{variation}$을 이용하여 네트워크의 상태를 판단하며 그림 5와 같은 수정된 혼잡제어 알고리즘의 동작에 따라 전송할 윈도우 크기를 조절한다.

```
If Q_variation > 0
    If Diff > β
        let W = W - 1
    Else if Diff < α
        let W = W + 1
    Else if α < Diff < β
        let W = W
If Q_variation == 0
    If Diff > β
        let W = W - 1/2
    Else if Diff < α
        let W = W + 1
    Else if α < Diff < β
        let W = W + 1/2
If Q_variation < 0
    If Diff > β
        let W = W
    Else if Diff < α
        let W = W + 1
    Else if α < Diff < β
        let W = W + 1
```

그림 5 Qvariation를 이용한 혼잡제어 알고리즘

첫번째로 $Q_{variation}$이 양의 값을 가지는 경우는 현재 네트워크 상태가 혼잡 상황이 커져서 라우터에 큐잉되는 패킷의 수가 증가했음을 의미한다. 이 경우에는 기존의 Vegas 알고리즘과 같은 혼잡제어 수행하게 된다. 즉, 2.1절에서의 식 (4)와 같은 혼잡제어 알고리즘에 따라 윈도우의 크기를 조절하게 된다.

두번째로 $Q_{variation}$이 0의 값을 가지는 경우는 현재 네트워크 상태가 안정하여 라우터에 큐잉되는 패킷의 수가 일정함을 의미한다. 하지만 이러한 경우에도 Reno 플로우는 윈도우를 계속 증가시키므로 Reno와의 경쟁력 향상을 위해 기존 Vegas의 혼잡제어 알고리즘을 좀 더 공격적으로 수정한다. 윈도우를 감소시키는 $Diff>\beta$인 구간에서는 윈도우를 1을 줄이는 대신 1/2을 줄이며 윈도우를 유지하는 $\alpha<Diff<\beta$인 구간에서는 윈도우를 유지하는 대신 1/2을 증가시킴으로써 경쟁력 향상을 추구하게 된다.

세번째로 $Q_{variation}$이 음의 값을 가지는 경우는 현재 네트워크 상태가 혼잡 상황이 줄어들어 라우터에 큐잉되는 패킷의 수가 감소했음을 의미한다. 즉, 네트워크의 사용가능한 대역폭이 남아있는 것으로 이 경우에는 Reno와

의 대역폭 경쟁을 위해 $Q_{variation}==0$인 경우보다 더욱 공격적으로 혼잡제어 알고리즘을 적용하게 된다. $Diff>\beta$인 구간에서는 윈도우를 줄이지 않고 현재의 윈도우를 유지하며, 윈도우를 유지했던 $\alpha<Diff<\beta$인 구간에서는 윈도우를 1만큼 증가시킴으로써 사용가능한 네트워크의 대역폭을 공격적으로 점유할 수 있도록 한다.

이상의 세 가지 기법을 추가하여 NewVegas는 기존 Vegas의 낮은 성능을 개선함과 동시에 Reno 플로우와의 경쟁, 그리고 Vegas 플로우 간의 경쟁에 있어서 공정성을 제공하게 된다. 이것은 기존 Vegas에서 불공정성 문제를 유발하는 *baseRTT* 변수에 대한 의존성을 제거하고, 새로운 파라미터를 사용하여 혼잡제어 알고리즘을 차등적으로 적용한 결과가 된다.

4. 실험 및 성능 평가

새로 제안한 TCP NewVegas의 성능 평가를 위해서 본 장에서는 LBNL(Lawrence Berkely National Laboratory)의 ns-2(network simulator)를 사용하여 시뮬레이션 하였다[19].

4.1 실험 환경

제안한 NewVegas의 성능을 평가하기 위해서 먼저 그림 6과 같은 환경을 구성하여 실험을 하였다. 먼저 NewVegas기반의 FTP 트래픽을 발생시키고 10초후에 경쟁하는 다른 트래픽을 발생시키는 시나리오를 구성하였다. 경쟁하는 다른 트래픽으로, Reno와의 공정성 실험에서는 TCP Reno 플로우를, NewVegas간의 공정성 실험에서는 NewVegas 플로우를, 기존 연구와의 공정성 향상 비교에서는 Vegas-A 플로우를 사용하였다. 노드의 수는 각 실험조건에 따라 송신단, 수신단 노드에서 각각 최대 20개의 노드를 이용하여 실험하였다. NewVegas의 공정성에 대한 성능 실험결과는 송신단에서의 윈도우 크기 변화와 전송률의 변화에 대한 결과를 모니터링 하였다.

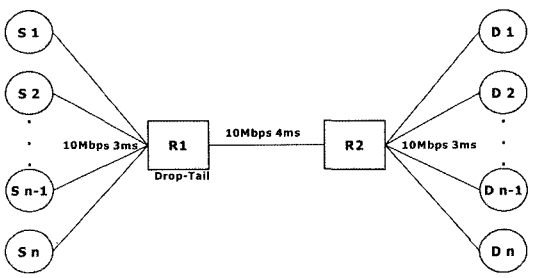
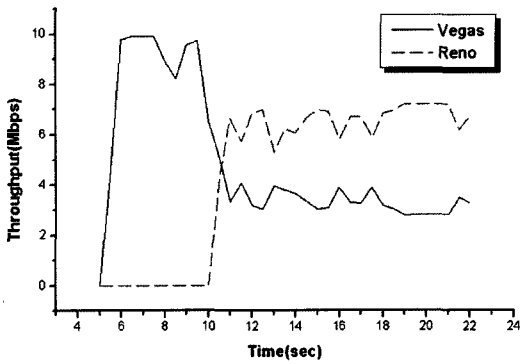


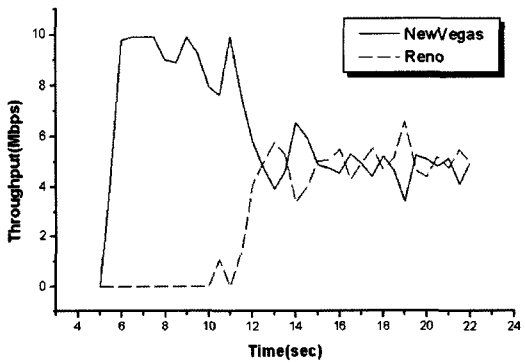
그림 6 실험 환경

4.2 NewVegas와 Reno 플로우 간의 공정성 실험

그림 7은 NewVegas와 Reno 플로우의 공정성에 대한 기본적인 실험 결과이다. 기존 Vegas와 제안하는 NewVegas



(a) Vegas vs. Reno



(b) NewVegas vs. Reno

전송률 (Mbps)	Vegas vs. Reno		NewVegas vs. Reno	
	Vegas	Reno	NewVegas	Reno
	3.48	6.41	5.26	4.53

(c) 전송률 비교

그림 7 NewVegas와 Reno, Vegas 플로우 간의 공정성 실험

의 Reno와의 공정성을 비교하기 위해 그림 6의 실험 환경에서 Vegas 플로우와 Reno 플로우의 경쟁과 New Vegas 플로우와 Reno 플로우의 경쟁을 실험하였다. 실험을 통해 기존의 Vegas는 Reno와의 경쟁에서 네트워크의 대역폭을 Reno 플로우에게 빼앗기므로 낮은 전송률을 가지지만 NewVegas는 Reno와의 경쟁에서 기존의 Vegas에 비해 좋은 성능을 가지는 것을 확인할 수 있다. 그러므로 Vegas의 문제점인 Reno와의 불공정성이 크게 개선되었음을 확인할 수 있다.

그림 8은 그림 7(b)의 실험 결과에서 NewVegas의 주요 파라미터터인 $Q_{variation}$의 변화를 나타낸 것이다. $Q_{variation}$이 양의 값을 가지는 경우, NewVegas는 기존 Vegas 알고리즘과 같이 동작하게 된다. 하지만 그림 8에서 볼수있듯이 $Q_{variation}$은 대부분 0이나 음의 값을 가지며, 따라서 기존 Vegas보다 공격적으로 동작하게 된다. 결과적으로 그림 7의 실험결과처럼 Reno와의 경쟁에

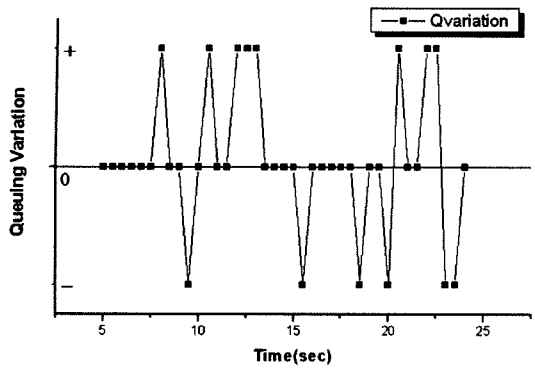
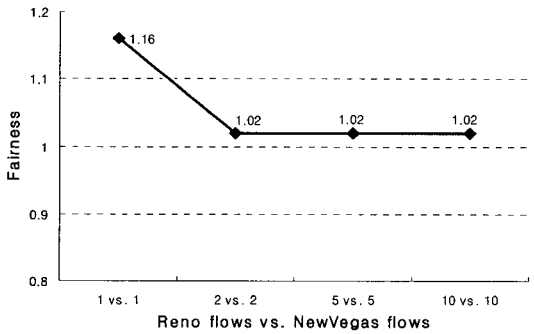
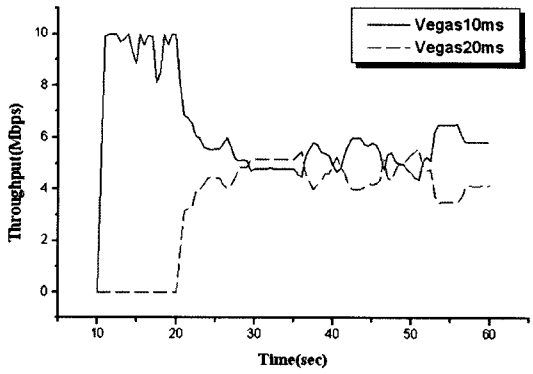


그림 8 $Q_{variation}$의 변화

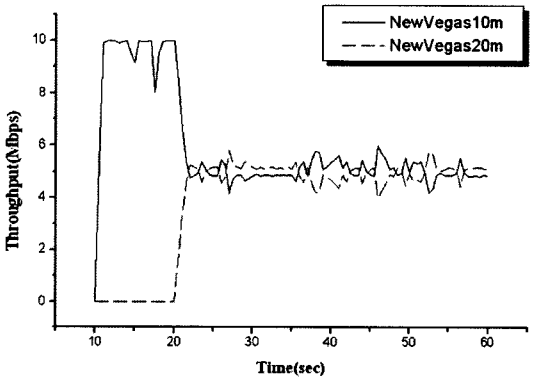


$$Fairness = \frac{Avg.\,NewVegas\ Throughput}{Avg.\,Reno\ Throughput}$$

그림 9 플로우 수에 따른 공정성 비교



(a) Vegas



(b) NewVegas

그림 10 RTT가 서로 다른 두 플로우 간의 공정성 실험

서 기존 Vegas에 비해 공평하게 대역폭을 경쟁하게 된다.

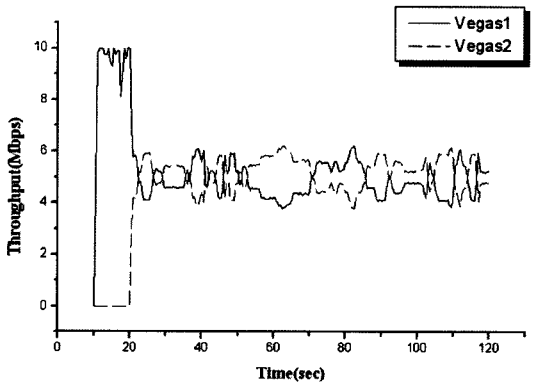
그림 9는 그림 7의 실험결과에서 확인한 NewVegas의 공정성 개선에 대한 실험을 여러 NewVegas 플로우와 Reno 플로우가 경쟁하는 경우로 확장하고, 플로우 수의 증가에 따른 공정성 비교를 나타낸 결과이다. 플로우 수에따른 각각의 실험에서 공정성은 Reno 플로우의 평균 전송률에 대한 NewVegas 플로우의 평균 전송률로 구하였으며 그 결과를 공정성에 대한 지수 *Fairness*로 나타내었다. 결과를 통해 플로우 개수가 극히 작은 경우를 제외하고는 플로우 개수에 독립적으로 공정성이 일정하게 유지됨을 확인할 수 있었으며 여러 플로우가 경쟁하는 경우에도 공정성 문제가 크게 개선되었다는 것을 확인할 수 있다.

4.3 NewVegas 플로우 간의 공정성 실험

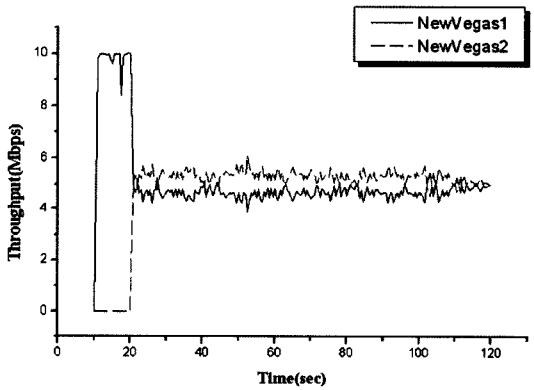
본 실험은 NewVegas 플로우 간의 공정성 실험을 위한 것으로 두 가지 실험을 수행하였다. 첫번째 실험은 RTT가 서로 다른 두 Vegas 플로우 간의 경쟁에서 RTT가 작은 플로우에 나타났던 불공정성 문제를 NewVegas의 혼잡제어 알고리즘에서 사용되는 새로운 파라미터 $Q_{variation}$에 의해 개선된 것을 보이기 위한 것이다. 두번째 실험은 연결 시점이 다른 Vegas 플로우 간의 경쟁에서 이전에 연결된 플로우에 나타났던 불공정성 문제를, 역시 파라미터 $Q_{variation}$에 의해 개선된 것을 보이기 위한 실험이다.

그림 10은 RTT가 서로 다른 두 Vegas 플로우 간의 경쟁에서 발생했던 불공정성 문제가 개선되었음을 보여주는 실험 결과이다. 3.3절에서 설명했듯이 기존의 Vegas는 부정확한 변수인 *baseRTT*를 이용해 혼잡제어를 수행함으로써 두 플로우간에 불공정성 문제를 유발시킨다. 하지만 NewVegas의 경우 *baseRTT*에 독립적인 파라미터인 $Q_{variation}$을 사용하여 이러한 문제를 개선하였다.

그림 11은 연결 시점이 다른 두 Vegas 플로우 간의 경쟁에서 발생했던 불공정성 문제가 개선되었음을 보여주는 실험 결과이다. 기존 Vegas에 비해 NewVegas는 일정 시간이 지난 후, 두 플로우의 전송률이 같은 값으로 수렴하는 것을 결과를 통해 확인할 수 있다. 이것은 *baseRTT*에 의존하지 않는 새로운 파라미터 $Q_{variation}$을 사용함으로써 이전에 연결된 Vegas 플로우에 대한 불공정성 문제가 개선되었음을 확인시켜 주는 결과이다.



(a) Vegas



(b) NewVegas

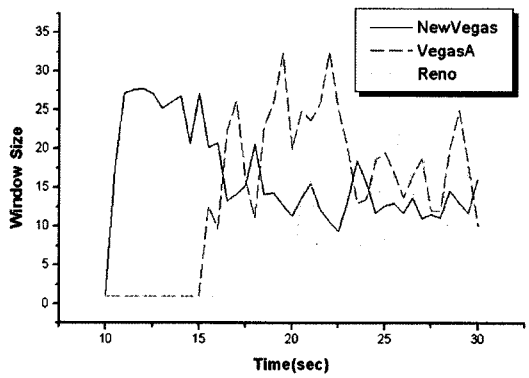
그림 11 연결 시점이 다른 두 플로우 간의 공정성 실험

4.4 NewVegas와 Vegas-A의 성능 비교

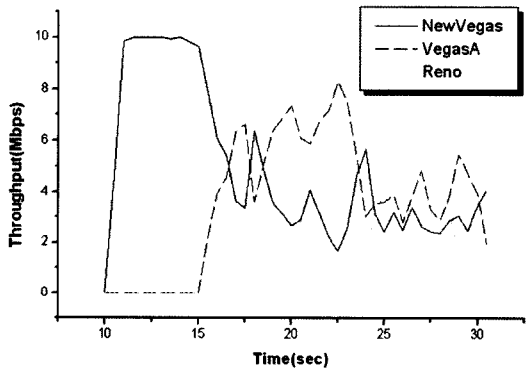
본 실험은 2.3절에서 설명했던 Vegas의 성능 향상을 위한 이전의 관련연구 중의 하나인 Vegas-A와의 성능을 비교한 것으로 Vegas-A와 NewVegas 그리고 Reno와의 경쟁을 통한 성능 및 공정성 비교 결과를 그림 12에서 보여주고 있다. 2.3절에서 설명하였듯이 Vegas-A는 기존의 Vegas에 비해 Reno와의 공정성이 향상되었지만 Vegas의 파라미터인 α, β를 너무 빈번하게 변화시키므로써 Vegas의 장점인 전송률의 안정성을 잃게되는 새로운 문제점을 갖는다. 그러나 NewVegas의 경우 전송률의 안정화를 그대로 유지하면서 Reno와의 공정성에서도 Vegas-A에 비해 좋은 성능을 보여준다. 따라서 New Vegas는 Srijith의 Vegas-A보다 공정성과 성능이 향상된 보다 안정적인 혼잡제어 알고리즘임을 확인할 수 있다.

5. 결론 및 향후 과제

현재 인터넷의 주요 TCP 버전인 Reno는 패킷 손실을 인지한 후 네트워크의 상태를 판단하는 수동적인 혼잡제



(a) 윈도우 비교



(b) 전송률 비교

평균전송률	NewVegas	Vegas-A	Reno
(Mbps)	3.43	3.72	3.19

(c) 평균 전송률 비교

그림 12 NewVegas와 Vegas-A, Reno 플로우 간의 공정성 실험

어 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 세 가지 심각한 불공정성 문제를 가지고 있기 때문에 범용적으로 사용되지 못하고 있다.

본 논문에서는 기존 Vegas의 불공정성 문제를 개선한 TCP NewVegas 혼잡제어 알고리즘을 제안하였다. RTT를 기반으로 라우터의 큐잉되는 패킷의 크기를 판단하고, 그 값에 의해 네트워크의 혼잡을 제어하는 기존의 Vegas에 비해서 제안한 NewVegas는 새로운 파라미터 $Q_{variation}$을 이용하여 기존의 혼잡제어 알고리즘을 보완한다. 즉, 네트워크 상황 판단의 근거를 *baseRTT*의 영향

에 민감한 *Diff*값이 아닌 새로운 파라미터 $Q_{variation}$을 이용하여 보다 정확한 혼잡 유추를 수행한다. 그로 인해 NewVegas는 기존의 Vegas가 Reno와 경쟁할 때 발생하는 불공정성 문제와 RTT가 서로 다른 Vegas 플로우들 간의 경쟁에서 RTT가 작은 플로우에 나타났던 불공정성 문제, 연결 시점이 다른 두 Veegas 플로우들 간의 경쟁에서 이전의 플로우에 나타났던 불공정성 문제를 모두 효과적으로 개선할 수 있다. 논문에서는 ns-2 시뮬레이터를 이용한 실험을 통해서 NewVegas와 Vegas 및 Reno 그리고 이전 관련 연구로서 Srijith의 Vegas-A 플로우 간의 안정성과 공정성에 대한 성능을 비교, 분석하였다. 실험 결과, 제안한 NewVegas가 기존의 Vegas 및 Vegas-A에 비해서 성능 및 공정성이 향상되었음을 확인할 수 있었다.

향후 연구 과제로는 제안한 NewVegas와 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. 또한 점차 확산될 새로운 네트워크 환경인 큰 BDP(Bandwidth Delay Product) 네트워크에서 NewVegas의 성능 향상에 대한 연구가 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] W. Feng and S. Vanichpun, "Enabling Compatibility Between TCP Reno and TCP Vegas," *IEEE SAINT 2003*, pp. 301-308, January 2003.

[14] U. Hengartner, J. Bolliger and Th. Gross, "TCP Vegas Revisited," *Proceeding of IEEE INFOCOM 2000*, pp. 1546-1555, March 2000.

[15] M. Oh, B. Song and K. Chung, "TCP PowerVegas: A Study on the Fairness Improvement of TCP Vegas," *JCCI 2003*, pp. VIII-B-4.1~4, May 2003.

[16] 오민철, 송병훈, 정광수, "TCP Vegas의 공정성 향상을 위한 혼잡 제어 알고리즘", 한국정보과학회 논문지, 제 31 권 3 호, pp. 269-279, 2004.

[17] G. Hasegawa, K. Kurata and M. Murata, "Analysis and Improvement of Fairness between TCP Reno and Vegas for Deployment of TCP Vegas to the Internet," *Proceeding of IEEE ICNP 2000*, November 2000.

[18] K. Srijith, L. Jacob and A. Ananda, "TCP Vegas-A: Solving the Fairness and Rerouting Issues of TCP Vegas," *Proceeding of IEEE IPCCC 2003*, April 2003.

[19] The Network Simulator ns-2, http://www.isi.edu/nanam/ns/

이 선 헌

정보과학회논문지 : 정보통신
제 32 권 제 4 호 참조

송 병 훈

1998년 광운대학교 컴퓨터과학과 학사
2000년 광운대학교 전자통신공학과 석사
2004년 광운대학교 전자통신공학과 박사
2004년 2~2004년 6월 (주)인티스 선임연구원. 2004년 7월~현재 전자부품연구원 지능형정보시스템 연구센터 연구원

정 광 수

정보과학회논문지 : 정보통신
제 32 권 제 2 호 참조