TCP 플로우 간의 공정성 개선을 위한 새로운 큐잉 알고리즘

채 현 석[†] · 최 명 렬[††]

요 약

TCP Vegas 버전은 현재 인터넷에서 가장 많이 사용되고 있는 TCP Tahoe나 Reno 버전에 비하여 높은 성능과 안정된 서비스를 제공한다. 그러나 TCP Vegas는 다른 버전의 TCP와 병목 링크를 공유하여 사용하게 되면 TCP Tahoe나 Reno 버전에 비하여 상대적으로 낮은 성능을 나타낸다. 이는 각 버전의 TCP가 수행하는 혼잡제어 알고리즘의 차이에서 기인하는 것으로 이를 해결하기 위한 많은 연구가 진행되고 있다. 본 논문에서는 링크의 대역폭에 따른 TCP의 최대 성능을 유지할 수 있는 최소한의 윈도우 크기를 분석하고 중간 라우터의 큐에서 각 TCP 연결에 적절한 윈도우 크기를 할당하기 위한 패킷 폐기 정책을 수행함으로써 TCP의 성능을 유지하면서 공정성을 개선할 수 있는 알고리즘을 제안한다. 제안한 알고리즘의 성능평가는 각 TCP 연결에 의하여 종단 시스템간에 전송된 데이터 바이트 수를 기준으로 측정하였으며, 그 결과 제안한 알고리즘은 높은 TCP 성능을 유지하면서 공정성이 개선됨을 확인하였다.

A New Queueing Algorithm for Improving Fairness between TCP Flows

Hyun-Seok Chae[†] · Myung-Ryul Choi[††]

ABSTRACT

TCP Vegas version provides better performance and more stable services than TCP Tahoe and Reno versions, which are widely used in the current Internet. However, in the situation where TCP Vegas and Reno share the bottleneck link, the performance of TCP Vegas is much smaller than that of TCP Reno. This unfairness is due to the difference of congestion control mechanisms of each TCP use. Several studies have been executed in order to solve this unfairness problem. In this paper, we analyze the minimum window size to maintain the maximum TCP performance of link bandwidth. In addition, we propose an algorithm which maintains the TCP performance and improves fairness by selective packet drops in order to allocate proper window size of each TCP connections. To evaluate the performance of the proposed algorithm, we have measured the number of data bytes transmitted between end-to-end systems by each TCP connections. The simulation results show that the proposed algorithm maintains the maximum TCP performance and improves the fairness.

키워드 : TCP 서비스 공정성(TCP Service Fairness), 혼잡제어(Congestion Control), 혼잡 창(Cwnd : Congestion Window), 서비스 품질(QoS : Quality of Service)

1. 서 론

TCP(Transmission Control Protocol)는 HTTP, FTP, TELNET 등과 같은 인터넷에서 가장 널리 사용되고 있는 서비스의 신뢰성 있는 데이터 전송을 담당하는 전송계층 프로토콜이다. RFC793에서 첫 번째 버전의 TCP가 제안된 이후 그 성능이나 안정성이 개선된 버전의 TCP가 개발되고 사용되어 왔다. 현재 인터넷에서 가장 널리 사용되고 있는 버전으로는 TCP Tahoe와 TCP Reno[10]가 있으며, 최근에는 새로운 혼잡제어(Congestion Control) 알고리즘을 적용한 TCP Vegas[1]가 제안되었다. TCP Tahoe나 Reno에서는 패킷의 손실이 감지될 때까지 윈도우 크기를 증가시키는 것에 비하여 TCP Vegas는 전송 패킷에 대한 RTT(Round Trip Time) 값을 기준으로 윈도우 크기를 적절하게 조절한다. 여러 연구결과에서 보면 이러한 방식의 혼잡제어를 사용하는 TCP Vegas는 인터넷에서의 성능이 TCP Reno에 비하여 40% 이상 높게 나타난다[2]. 또한 하나의 링크를 여러개의 TCP 연결이 공유하여 사용하는 경우에도 TCP Vegas는 전송지연이 클수록 성능이 저하되는 TCP Reno보다 안정되고 공정한 성능을 보이고 있다[3].

그러나 인터넷의 성능을 향상시키기 위해서 TCP의 버전을 Vegas로 업그레이드하는 경우 동시에 모든 컴퓨터를 업그레이드하는 것은 불가능하며, 따라서 인터넷에 접속되어 있는 컴퓨터의 TCP Vegas 버전으로 업그레이드할 때 일

† 정 회 원 : 동원대학 모바일컨텐츠과 교수
†† 정 회 원 : 한양대학교 전자컴퓨터공학부 교수
논문접수 : 2003년 7월 25일, 심사완료 : 2004년 2월 26일

정 기간 TCP Vegas와 Reno가 공존하게 된다. 이러한 경우에는 TCP Vegas와 Reno가 하나의 링크를 공유하여 동시에 데이터를 전송하는 경우가 발생하게 되는데, 이때 각 TCP 연결에 대한 공정성(Fairness)에 심각한 문제가 발생한다. TCP Vegas는 망의 상태에 따라 윈도우 크기를 적절하게 증가 혹은 감소시키지만 TCP Reno는 패킷이 손실될 때까지 윈도우 크기를 지속적으로 증가시키므로 병목링크에서는 TCP Reno가 상대적으로 데이터를 전송할 기회를 많이 갖는다. 한 연구결과에 따르면 TCP Vegas와 Reno가 공존하는 다중(Heterogeneous) 환경에서 두 개 이상의 TCP 연결이 하나의 병목 링크를 공유하여 동시에 데이터를 전송하게 되면, TCP Vegas는 Reno에 비하여 매우 낮은 성능을 나타낸다[4].

TCP Vegas를 인터넷에 적용하기 위해서는 이러한 공정성의 개선이 필수적이며, 이를 해결하기 위하여 많은 연구가 진행중이다. 현재 진행중인 연구의 방향은 크게 두 가지 방법으로 구분되는데, 하나는 TCP Vegas의 혼잡제어 메커니즘을 수정[5-7]하는 것이고, 또 하나는 라우터와 같은 중간 노드의 큐에서 트래픽을 차등 처리하는 방법이다[6, 7]. 큐에서의 차등 처리 방법은 주로 RED(Random Early Detection) 알고리즘[8]을 수정하는 방향으로 진행되고 있다. 그러나 TCP Vegas 알고리즘을 수정하는 방법은 인터넷에 접속되는 컴퓨터의 소프트웨어에 수정이 필요하므로 인터넷 트래픽의 성능을 개인 사용자의 능력에 의존해야 하는 단점이 있다. 또한 라우터의 큐잉 메커니즘을 수정하여 공정성을 개선하는 방법은 TCP Reno와 Vegas 간의 공정성은 개선되나 전체적인 트래픽 성능이 저하되는 단점이 있다[6, 7].

본 논문에서는 트래픽 성능은 유지하면서 TCP Reno와 Vegas 간의 공정성을 개선하기 위한 최적 윈도우를 고려한 패킷 선별 폐기 알고리즘을 제안한다. 제안 알고리즘은 라우터의 큐에서 동작되며, 각 TCP 연결에 해당하는 트래픽을 구분하고, 병목 링크의 대역폭과 통신 지연 등을 고려한 최적의 윈도우 크기를 분석하고 이를 새롭게 유입되는 패킷에 적용, 선별적으로 폐기함으로써 TCP의 성능을 유지하면서 공정성을 개선할 수 있다. 또한 각 TCP 연결에 따르는 트래픽을 제어할 수 있어 향후 인터넷의 QoS를 보장하기 위한 방법으로도 사용할 수 있다. 라우터의 큐에서는 TCP 연결의 윈도우 크기를 알 수 없으므로 큐에 누적되는 각 TCP 연결에 의한 패킷의 수를 사용하여 최적의 윈도우 크기를 계산하였다. 기존에 연구되는 대부분의 큐잉의 성능평가는 큐에서의 데이터 처리율을 기준으로 하였으나, TCP의 성능개선을 위한 성능평가는 종단 시스템간에 전송된 데이터 양을 기준으로 하는 것이 가장 정확하다. 따라서 제안한 알고리즘의 성능평가는 각 TCP 연결에 의하여 종단 시스템간에 전송된 데이터 바이트 수를 기준으로 측정하였으며, 그 결과 TCP의 트래픽 성능은 병목 링크의 대역폭을 모두 사용하는 수준으로 유지하면서 TCP Vegas와 Reno 간의 공정성이 개선됨을 확인하였다. 개선된 성능의 결과는 ns2(Network Simulator 2)[9]를 이용하여 나타내었다.

2. TCP 혼잡제어 메커니즘

TCP는 인터넷에서 종단 시스템간의 신뢰성 있는 데이터 전송을 담당하는 프로토콜로 흐름제어(Flow Control)와 혼잡제어를 수행한다. 데이터를 높은 성능으로 신뢰성 있게 전송하기 위해서는 데이터 처리속도의 차이를 극복하기 위한 흐름제어와 망에서의 혼잡상황을 회피하기 위한 혼잡제어가 동시에 수행되어야 하며 이를 위하여 TCP에서는 혼잡 윈도우(*cwnd* : congestion window)를 사용하고 있는데, 이 혼잡 윈도우를 어떻게 조절하느냐에 따라 TCP의 성능 및 망의 성능이 크게 좌우된다. 이번 절에서는 TCP Reno와 Vegas에서 수행하는 혼잡제어 메커니즘에 대하여 알아보고 모의실험을 통하여 각 버전에 대한 성능과 공정성을 측정한다.

2.1 TCP Reno

기존의 TCP Tahoe 버전에서 사용하던 느린 출발(Slow Start)과 혼잡 회피(Congestion Avoidance), 그리고 빠른 재전송(Fast Retransmit) 알고리즘에 빠른 회복(Fast Recovery) 알고리즘을 추가한 것이 TCP Reno버전의 혼잡제어 알고리즘[11, 12]이다. TCP Reno 버전은 패킷이 손실될 때까지 윈도우 크기를 증가시키는데, 느린 출발 구간에서와 혼잡 회피 구간에서 다르게 증가된다. 혼잡 윈도우의 크기는 ACK 패킷이 수신되면 증가되는데, 현재 시간을 t, ACK 패킷이 수신된 시간을 $t+t_A$, 현재의 임계값(Threshold Value)을 $ssth(t)$, 그리고 현재의 혼잡 윈도우의 크기를 $cwnd(t)$라 하면 ACK 패킷이 수신되어 증가되는 윈도우 크기 $cwnd(t+t_A)$는 다음의 식 (1)과 같이 표현된다.

$$cwnd(t+t_A)=\begin{cases} slow\ start\ phase: & \\ \quad cwnd(t)+1, & \text{if } cwnd(t) < ssth(t) \\ congestion\ avoidance\ phase: & \\ \quad cwnd(t)+\dfrac{1}{cwnd(t)}, & \text{if } cwnd(t) \geq ssth(t) \end{cases} \quad (1)$$

한편 패킷의 손실이 감지되면 혼잡 윈도우의 크기를 감소시키는데, 재전송 시간의 종료에 의한 패킷의 손실을 감지한 경우의 윈도우 크기는 식 (2)에 의하여 감소되며, ACK 패킷을 중복 수신함으로써 패킷의 손실을 감지한 경우의 윈도우 크기는 식 (3)에 의하여 감소된다.

$$cwnd(t)=1\ ;\quad ssth(t)=\frac{cwnd(t)}{2} \tag{2}$$

$$ssth(t)=\frac{cwnd(t)}{2}\ ;\quad cwnd(t)=ssth(t) \tag{3}$$

2.2 TCP Vegas

TCP Vegas는 혼잡 윈도우의 크기를 조정할 때 현재 망에서 사용 가능한 대역폭을 고려하기 위하여 왕복응답시간(RTT : Round-Trip Time)을 측정한다. 또한 송신자가 패킷의 손실을 회피하면서 보낼 수 있는 윈도우 크기를 계산할 때 측정된 왕복응답시간을 적용함으로써 현재 망의 상태를 고려한 가장 적절한 윈도우 크기를 구할 수 있다. 측정된 왕복응답시간을 *rtt*, 측정된 왕복응답시간의 최소 값을 *basertt*, 예상 성능과 실제 성능의 차이를 *diff*, 그리고 두 개의 임계값을 α와 β라 하면 ACK 패킷을 수신했을 때의 윈도우 크기 $cwnd(t+t_A)$는 다음의 식 (4)와 같이 계산된다.

$$cwnd(t+t_A)=\begin{cases} cwnd(t)+1, & \text{if } diff<\dfrac{\alpha}{basertt} \\ cwnd(t), & \text{if } \dfrac{\alpha}{basertt}\le diff\le\dfrac{\beta}{basertt} \\ cwnd(t)-1, & \text{if } diff>\dfrac{\beta}{basertt} \end{cases}$$

$$diff=\frac{cwnd(t)}{basertt}-\frac{cwnd(t)}{rtt} \tag{4}$$

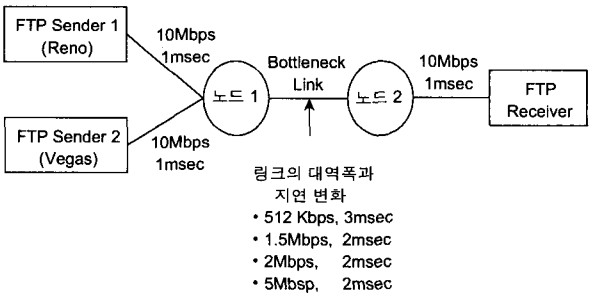
2.3 TCP Reno와 Vegas의 성능 및 공정성

TCP Reno와 Vegas의 성능과 공정성을 측정하기 위하여 ns-2를 사용하여 모의실험을 수행하였다. 모의실험에서 사용한 트래픽은 인터넷에서 가장 많은 양의 데이터를 발생시키는 FTP(File Transfer Protocol)를 사용하였으며, FTP에 의하여 전송된 데이터의 바이트를 기준으로 하여 각각의 전송 성능과 공정성을 측정하였다.

2.3.1 모의실험 환경

모의실험 환경은 다음의 (그림 1)과 같다. 모의 실험에서는 두 개의 FTP 송신자와 한 개의 FTP 수신자를 사용하였으며, 각 FTP 연결은 하나의 병목 링크를 공유하게 된다. 병목 링크에 연결되는 노드 1과 노드 2에서 사용하는 큐는 Drop-tail 큐를 사용한다. FTP가 사용하는 TCP의 버전은 하나는 TCP Reno를 사용하였으며, 다른 하나는 TCP Vegas를 사용하였다. 또한 병목 링크는 대역폭과 링크의 지연(Delay)을 512Kbps, 3msec에서부터 5Mbps, 2msec까지 다양한 형태로 변화시키면서 각각에 대한 모의실험을 진행하였다.

모의실험에의 결과 측정은 성능과 공정성(Fairness)의 두 가지 항목에 대하여 측정하였다. 성능은 TCP Reno 버전을 사용하는 FTP Sender 1과 TCP Vegas 버전을 사용하는 FTP Sender 2가 단위시간당 전송한 데이터 바이트 수의 합을 시간으로 나누어 계산하였으며 최대 값은 병목 링크의 대역폭과 같다. 공정성은 Reno와 Vegas를 사용한 각 FTP 연결이 점유한 최대 전송속도에 대한 최소 전송속도의 비율로, (*최소 성능/최대 성능*)×100의 식으로 계산하였으며, 공정성이 100%이면 두 FTP 연결의 최대 전송속도와 최소 전송속도가 같은, 즉 병목 링크를 공평하게 사용하였음을 의미한다.



(그림 1) TCP 성능 모의실험 환경

2.3.2 모의실험 결과

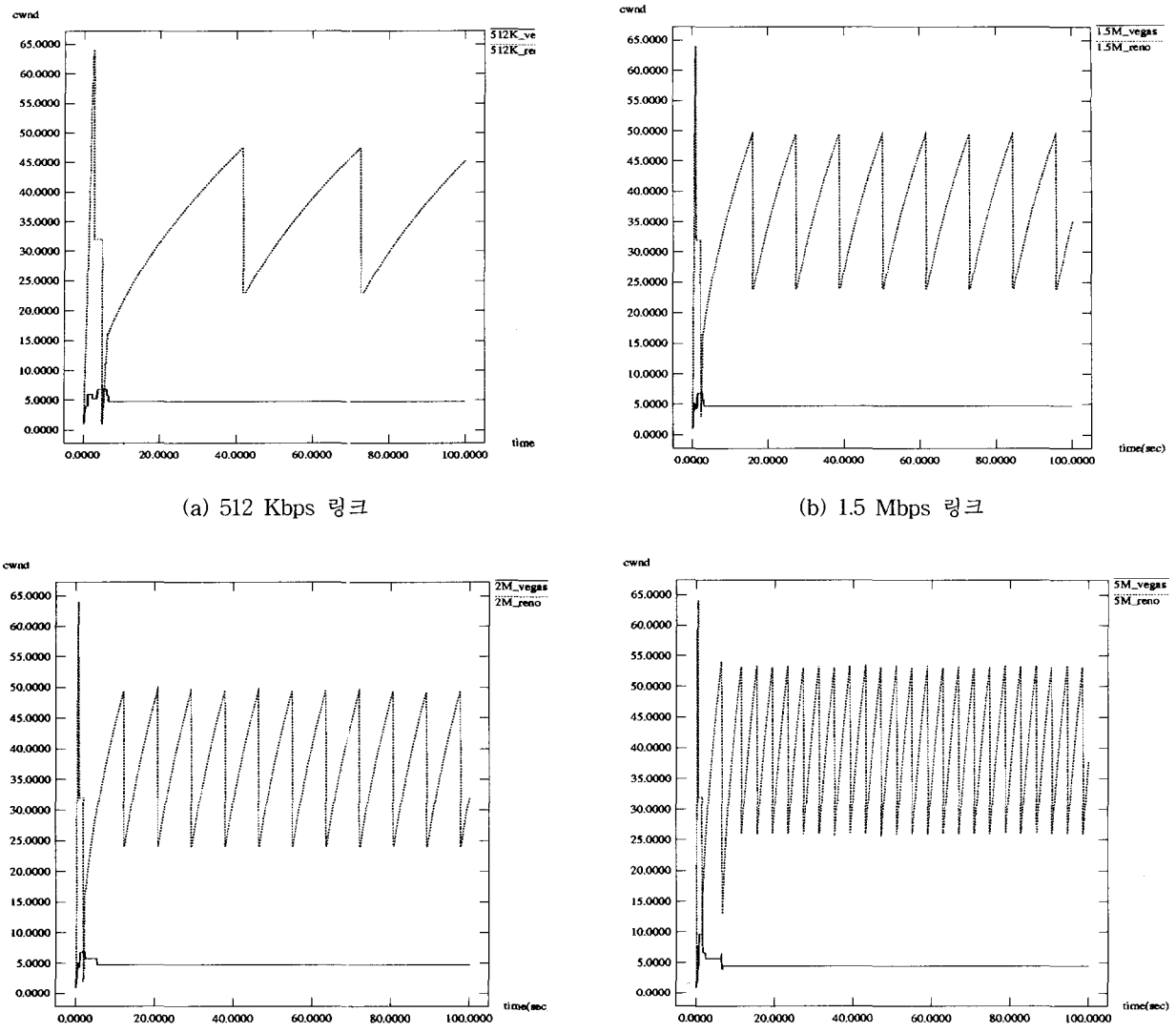
TCP Reno 버전과 TCP Vegas 버전에 대한 모의실험의 측정치와 성능, 그리고 공정성에 대한 결과는 <표 1>에 나타내었다.

〈표 1〉 모의실험 결과

대역폭(지연)	측정치(Kbps)		성능 (Kbps)	공정성(Reno/Vegas) (%)
	TCP Reno	TCP Vegas		
512Kbps(5msec)	457.16	54.84	512	12
1.5Mbps(2msec)	1348.18	151.82	1500	11.26
2Mbps(2msec)	1797.69	202.31	2000	11.25
5Mbps(2msec)	4528.53	471.47	5000	10.41

표에서 나타난 결과를 살펴보면 두 개의 FTP에 의한 데이터 전송 성능은 병목 링크에서 제공하는 대역폭을 나누어 모두 사용한 것으로 나타난다. 그러나 공정성의 경우에는 Vegas를 사용하는 FTP는 Reno를 사용한 FTP 에 비하여 약 10~12% 정도의 성능밖에 사용하지 못하고 있다. 즉, TCP Reno와 Vegas 버전이 병목 링크를 동시에 사용하는 경우에는 Vegas 버전의 성능이 Reno 버전의 성능에 비하여 매우 낮게 나타난다.

이러한 현상은 각 TCP가 사용하는 혼잡제어 메커니즘의 차이에서 비롯된다. Vegas 버전은 링크에서의 상황을 혼잡 윈도우(*cwnd*)에 바로 적용하는 반면 Reno 버전에서는 링크에서의 패킷의 손실이 발생할 때까지 계속하여 혼잡 윈도우의 크기를 증가시키기 때문이다. (그림 2)에는 모의실험에서 측정한 TCP Reno와 Vegas의 *cwnd* 값을 나타내었다.



(a) 512 Kbps 링크

(b) 1.5 Mbps 링크

(c) 2 Mbps 링크

(d) 5 Mbps 링크

(그림 2) 혼잡 윈도우(Congestion Window) 크기 변화

(그림 2)에서 나타난 *cwnd*의 값을 살펴보면 TCP Reno의 경우에는 병목 링크의 속도에 따라 약간씩 인한 것이다. 따라서 Vegas 버전을 인터넷에 적용하기 위해서는 바로 이러한 공정성 문제를 해결하는 것이 반드시 필요하다.

본 실험 결과에서 알 수 있는 것은 TCP Reno 버전과 TCP Vegas 버전을 같이 사용하는 경우에 Vegas 버전이 가장 효율적인 혼잡제어 메커니즘을 제공함에도 불구하고 Reno 버전에 비하여 매우 낮은 성능을 보인다. 이는 Reno와 Vegas의 혼잡제어 메커니즘의 차이에 기인한 것이다. 따라서 Vegas 버전을 인터넷에 적용하기 위해서는 바로 이러한 공정성 문제를 해결하는 것이 반드시 필요하다.

3. 윈도우 크기와 공정성과의 관계

TCP는 흐름제어를 위하여 슬라이딩 윈도우 기법을 사용한다. 그런데 실제 인터넷에서는 두 종단 시스템간의 데이터 처리속도의 차이보다는 망에서의 처리 속도가 훨씬 느리며, 이로 인한 망에서의 혼잡상황을 극복하기 위한 흐름제어와 더불어 혼잡제어를 수행한다. 이를 위하여 TCP에서는 고정된 값을 갖는 기본 윈도우와 데이터 전송 상태에 의하여 그 값이 변하는 혼잡 윈도우등의 두 개의 윈도우를 사용하여 흐름제어를 수행한다. 슬라이딩 윈도우 알고리즘에 적용되는 윈도우 크기는 기본 윈도우와 혼잡 윈도우 중 작은 값을 적용하는데, 기본 윈도우의 크기는 혼잡 윈도우에 비하여 상대적으로 큰 값으로 설정되어 있으므로 실질적인 흐름제어는 혼잡 윈도우에 의하여 수행된다.

3.1 기본 윈도우 값에 의한 TCP 성능 분석

TCP Reno 버전의 경우에는 망에서 패킷이 손실되기 전까지 혼잡 윈도우의 크기가 지속적으로 증가하게 되며, 이로

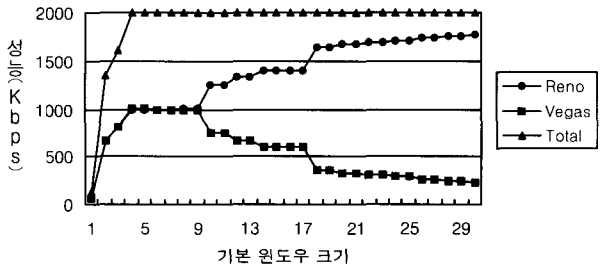
인하여 TCP Vegas 버전이 사용할 수 있는 대역폭이 감소함으로써 공정성이 저하된다. 그런데 작은 크기의 기본 윈도우를 사용하면 혼잡 윈도우에 의한 영향이 적어지므로 TCP Reno와 Vegas의 성능이 달라지게 된다. 본 절에서는 기본 윈도우의 값에 따른 TCP Reno와 Vegas의 성능 및 공정성에 대하여 알아보았다.

3.1.1 모의실험 환경

모의실험의 환경은 (그림 1)과 같이 두 개의 FTP 송신자가 병목 링크를 공유하여 데이터를 전송하도록 노드를 구성하였다. FTP가 사용하는 TCP의 버전은 하나는 TCP Reno를, 다른 하나는 TCP Vegas를 사용하였다. 또한 병목 링크는 2 Mbps의 대역폭과 5 msec의 전송 지연을 갖도록 구성하였으며, 각 TCP가 사용하는 기본 윈도우의 크기는 1에서부터 30까지 증가하면서 각 TCP 버전에 의한 성능과 공정성을 측정하였다.

3.1.2 모의실험 결과

기본 윈도우 크기에 대한 TCP Reno 버전과 TCP Vegas 버전에 대한 모의실험의 측정치와 성능, 그리고 공정성에 대한 결과는 (그림 3)과 같이 나타났다.

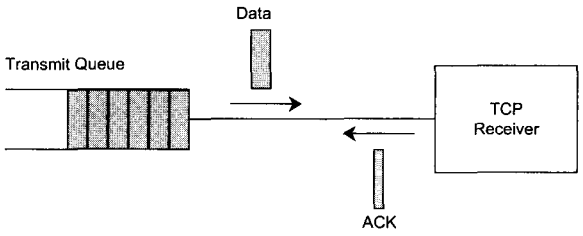


(그림 3) 기본 윈도우 크기에 따른 TCP 성능

실험 결과에서 기본 윈도우의 크기가 4 이하인 경우에는 두 개의 FTP에 의한 전체 성능(Total)이 링크에서 제공하는 대역폭을 모두 사용하지 못한다. 즉, TCP의 윈도우 크기가 작으면 링크의 대역폭을 충분히 사용하지 못하여 전체 성능이 저하된다는 것을 알 수 있다. 반면에 윈도우의 크기가 10 이상인 경우에는 전체 성능은 링크의 대역폭을 모두 사용하지만 TCP Reno와 Vegas 간의 성능 차이가 점점 심해진다. 즉, TCP의 기본 윈도우의 크기가 커지면 혼잡 윈도우 크기에 의하여 데이터가 전송되므로 TCP Reno가 데이터를 전송할 수 있는 기회를 많이 갖는다는 것을 알 수 있다. 그러나 기본 윈도우의 크기가 5에서 9 사이인 경우에는 전체 성능과 공정성이 모두 우수한 것으로 나타났다. 즉, TCP의 성능은 유지하면서 동시에 공정성이 개선되는 기본 윈도우의 크기가 존재한다는 것을 알 수 있었다.

3.2 최적의 TCP 기본 윈도우 크기 분석

TCP는 전송한 데이터에 해당하는 ACK 패킷을 수신하기 전에 기본 윈도우 크기만큼의 데이터를 계속해서 전송할 수 있다. 따라서 기본 윈도우의 크기가 작아지면 ACK 패킷의 수신을 기다리는 확률이 높아지고, 링크에 전송되는 데이터가 없는 휴지(Idle)상태가 발생함으로써 전송성능이 저하된다. 따라서 윈도우의 크기는 링크에서 제공하는 대역폭을 모두 사용할 수 있는 값을 유지해야 한다. 그런데 일정 수준 이상의 값을 사용하면 공정성에 문제가 발생하므로 기본 윈도우 중에서 최소 값의 윈도우 크기를 사용하면 성능은 최대로 유지하면서 TCP간의 공정성을 개선할 수 있다. 이번 절에서는 TCP의 성능을 유지하면서 공정성을 개선할 수 있는 최적의 윈도우 크기를 분석하였다.



(그림 4) 송신 큐에 누적되는 데이터와 ACK 패킷

TCP 송신자에 의하여 윈도우 크기만큼 한꺼번에 전송된 패킷 데이터는 (그림 4)와 같이 병목링크에 연결되어있는 중간 노드의 송신 큐에 쌓이게 된다. 링크의 대역폭을 BW, 병목 링크의 전송지연을 τ, TCP 수신자가 데이터를 수신하고 이에 해당하는 ACK 데이터를 생성하는 시간을 T_{ACK}, 그리고 패킷 데이터의 길이를 L_D라 정의하면 큐의 맨 앞에 있는 한 개의 데이터가 큐에서 출력되는데 소요되는 시간 T_D는 다음의 식 (5)와 같이 표현된다.

$$T_D=(L_D\times 8)/BW \tag{5}$$

큐에 누적되어 있는 다음 데이터가 계속하여 출력되는 동안 첫 번째 데이터는 병목링크를 통하여 목적지로 전송되고, 이를 수신한 TCP는 ACK 패킷을 생성하여 원래의 송신자로 응답한다. 병목 링크의 전송지연이 크다고 가정하면 큐에서 출력된 첫 번째 데이터에 해당하는 ACK 패킷이 원래의 송신자로 도착하는 시간 T_{res}는 다음의 식 (6)과 같이 표현된다.

$$T_{res}\approx 2\tau + T_{ACK} \tag{6}$$

TCP 송신자가 사용하는 기본 윈도우 크기를 W라 놓으면 첫 번째 전송된 데이터에 대한 ACK 패킷이 도착하는 시간, 즉 T_{res}보다 큐에 누적되어 있는 나머지 $W-1$개의 데이터가 큐에서 출력되는 시간이 작게되면 중간 노드의 송신 큐가 비어있게 된다. 송신 큐가 비어있는 동안에는 링크에 새롭게 출력되는 데이터가 없으므로 링크의 대역폭만큼

의 성능을 사용하지 못한다. 즉, 데이터 전송이 중단되지 않고 링크에서 제공하는 대역폭을 모두 사용하기 위해서는 T_{res}보다 $T_D \times (W-1)$가 커야 하며, 다음의 식 (7)을 만족하는 W 중 최소 값이 TCP의 성능은 유지하면서 공정성을 개선할 수 있는 최적의 윈도우 크기가 된다.

$$T_D \times (W-1) > T_{res}$$

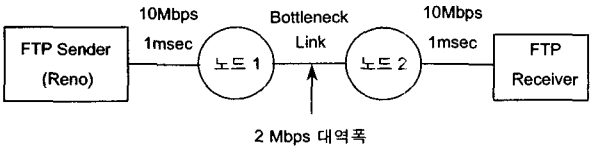
$$\text{즉, } W > \frac{(2\tau + T_{ACK}) \times BW}{8 \times L_D} + 1 \quad (7)$$

N_C개의 TCP 연결이 해당 링크를 사용하여 동시에 데이터를 전송하는 경우에는 각각의 TCP가 링크의 대역폭을 나누어 사용하게 되므로 각 TCP 연결이 사용할 수 있는 최적의 윈도우 크기 W_{best} 다음의 식 (8)과 같이 표현할 수 있다.

$$W_{best} = \frac{(2\tau + T_{ACK}) \times BW}{8 \times L_D \times N_C} + 2 \quad (8)$$

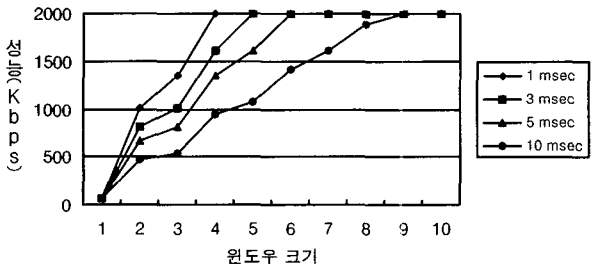
위의 식 (7)과 (8)에서 TCP 성능을 최대로 유지하면서 공정성을 유지하기 위한 윈도우 크기는 병목 링크의 지연이 커질수록, 그리고 대역폭이 커질수록 큰 값의 윈도우를 사용하여야 하며, TCP 연결의 개수가 커질수록, 그리고 데이터의 길이가 클수록 작은 값의 윈도우를 사용하여야 함을 알 수 있다.

링크의 대역폭과 지연에 대한 최적의 기본 윈도우 크기를 확인하기 위하여 다음의 (그림 5)와 같은 환경에서 모의실험을 수행하였다.



(그림 5) 최적 윈도우 모의실험 환경

모의실험에서는 병목링크의 대역폭을 2Mbps로 설정하였으며, 1msec, 3msec, 5msec, 그리고 10msec의 전송지연에 대한 최적의 윈도우 크기를 확인하였다. 모의실험의 결과는 다음의 (그림 6)에 나타내었다.



(그림 6) 2Mbps 대역폭에서의 최적 윈도우 크기

모의실험 결과에서 전송 지연이 1msec, 3msec, 5msec, 10msec인 링크에서의 최적 윈도우 값은 각각 4, 5, 6, 그리고 8임을 보이고 있다. 따라서 링크의 상태에 따른 최적의 윈도우가 존재한다는 사실과 전송지연이 증가할수록 최적 윈도우의 크기가 커진다는 것을 확인하였다.

4. 최적 윈도우를 고려한 패킷 선별 폐기 알고리즘

링크에서 제공하는 대역폭과 전송 지연, TCP 연결의 개수, 그리고 컴퓨터에서 ACK 패킷을 처리하는 시간에 맞는 최적 윈도우를 사용하면 TCP의 최대 성능을 유지하면서 공정성을 개선할 수 있다는 것을 확인하였다. 그러나 윈도우 크기의 조절을 하기 위해서는 TCP에서 수행하는 혼잡제어 알고리즘을 수정해야하므로, 다른 버전의 TCP와 동시에 사용하면 공정성이 저하된다. 이번 절에서는 TCP의 성능을 유지하면서 공정성을 개선하기 위한 방법으로 최적 윈도우를 고려한 패킷 선별 폐기 알고리즘을 제안한다.

4.1 알고리즘 개요

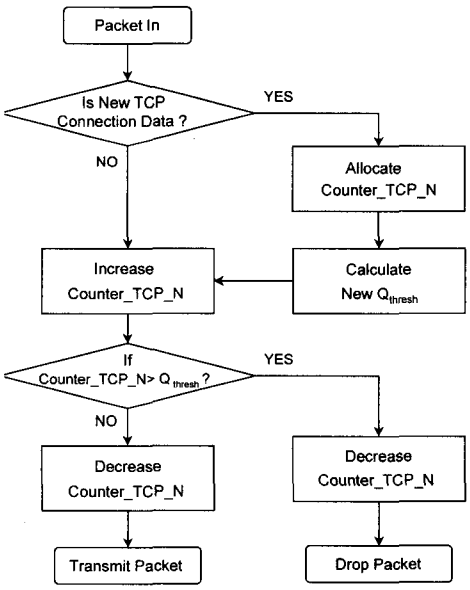
최적 윈도우를 고려한 패킷 선별 폐기 알고리즘은 중간 노드의 송신 큐에서 동작하는 알고리즘으로 최적 윈도우 크기를 초과하여 유입되는 패킷을 폐기함으로써 공정성을 개선한다. 병목링크를 사용하여 통신하는 경우 송신 큐에는 각 TCP 연결에 의하여 전송되어온 패킷이 누적되는데, 각 TCP 송신자는 윈도우 크기만큼의 데이터를 한꺼번에 전송하므로 큐에 누적되는 패킷의 수와 각 TCP 송신자가 사용하는 윈도우 크기는 비례한다. 따라서 중간노드의 송신 큐에서 일정 개수 이상 누적되는 패킷을 폐기하면 이 노드를 통하여 통신하는 각 TCP의 성능은 유지하면서 공정성을 개선할 수 있다.

제안 알고리즘에서는 데이터 전송을 하는 TCP 연결 및 각 TCP 연결에 의하여 전송되는 데이터를 구분하고, 현재 큐에 누적된 각 TCP 연결에 의한 패킷의 수를 계산한다. 또한 현재 큐를 사용하는 TCP 연결의 개수, 링크에서 제공하는 대역폭과 전송지연 등을 고려하여 각 TCP 연결의 성능을 최대로 유지하면서 공정성을 개선할 수 있는 큐의 임계 값 Q_{thresh}를 계산한다. Q_{thresh}는 각 TCP 연결이 최대 성능을 유지할 수 있는 최적의 윈도우 크기에 비례하는 값을 가지며, 다음의 식 (9)와 같이 표현할 수 있으며, 망의 상태와 트래픽 특성에 따라 능동적인 값을 갖는다.

$$Q_{thresh} \propto \frac{(2\tau + T_{ACK}) \times BW}{8 \times L_D \times N_C} + 2 \quad (9)$$

새로운 패킷이 유입되어 송신 큐에 누적되면, 현재 큐에 누적되어 있는 해당 TCP 연결의 패킷의 수, Counter_TCP_N

을 Q_{thresh}과 비교한다. 큐에 누적된 패킷의 수가 Q_{thresh}보다 크면 해당 TCP 연결의 윈도우 크기를 공정성을 개선할 수 있는 최적의 값으로 유지하기 위하여 해당 패킷을 폐기한다. 이러한 최적 윈도우를 고려한 패킷 선별 폐기 알고리즘의 흐름도는 다음의 (그림 7)과 같다.



(그림 7) 제안 알고리즘의 흐름도

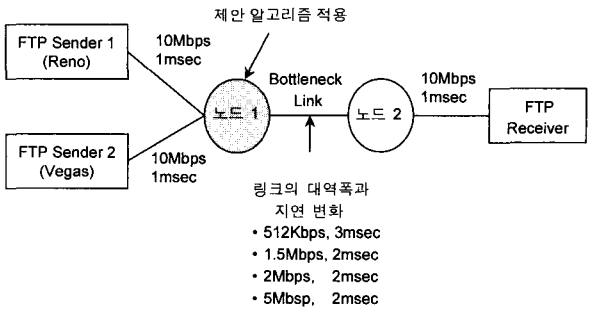
제안 알고리즘에서 사용되는 Q_{thresh}는 각 TCP 연결에서 사용하고 있는 현재의 윈도우 크기로 이 값을 조절하면 해당 TCP 연결에서 발생하는 트래픽의 양을 조절할 수 있어, 모든 TCP 연결에 동일한 값으로 적용하면 각 TCP 연결의 공정성을 개선하여 동일한 성능을 제공할 수 있다. Q_{thresh}의 계산은 식 (9)에 의하여 계산되므로 병목 링크의 대역폭과 전송지연, 트래픽의 특성을 고려하여 능동적으로 변화하게 된다. 또한 특정 TCP 연결에 가중치를 두어 적용하게 되면, 해당 TCP 연결의 성능을 다른 연결의 트래픽에 비하여 높게 부가할 수 있어 QoS를 보장하기 위한 방법으로도 사용 가능하다.

이러한 패킷 선별 폐기 알고리즘은 라우터 등과 같은 중간 노드에 적용할 수 있으므로 모든 종단 시스템의 소프트웨어를 업그레이드 할 필요가 없으며, 망 관리자에 의하여 공정성을 개선하거나 QoS 정책을 수립할 때에 라우터 설정 통하여 이루어 질 수 있다는 장점이 있다.

4.2 제안 알고리즘의 성능평가

4.2.1 모의실험 환경

제안 알고리즘의 성능평가를 위하여 앞의 2.3절에서 수행한 모의실험과 동일한 환경에서 TCP의 성능과 공정성을 측정하였다. 모의실험 환경은 다음의 (그림 8)과 같이 두 개의 FTP 송신자와 한 개의 FTP 수신자를 사용하였으며, 병목 링크에 연결되는 노드 1과 노드 2에서 사용하는 큐에는 본 논문에서 제안한 알고리즘을 적용하였다.



(그림 8) 제안 알고리즘의 모의실험 환경

모의실험에의 결과 측정은 성능과 공정성(Fairness)의 두 가지 항목에 대하여 측정하였다. 성능은 TCP Reno 버전을 사용하는 FTP Sender 1과 TCP Vegas 버전을 사용하는 FTP Sender 2가 단위시간당 전송한 데이터 바이트 수의 합을 시간으로 나누어 계산하였으며 최대 값은 병목 링크의 대역폭과 같다. 공정성은 Reno와 Vegas를 사용한 각 FTP 연결이 점유한 최대 전송속도에 대한 최소 전송속도의 비율로, (*최소 성능/최대 성능*)×100의 식으로 계산하였으며, 공정성이 100%이면 병목 링크를 공평하게 사용하였음을 의미한다.

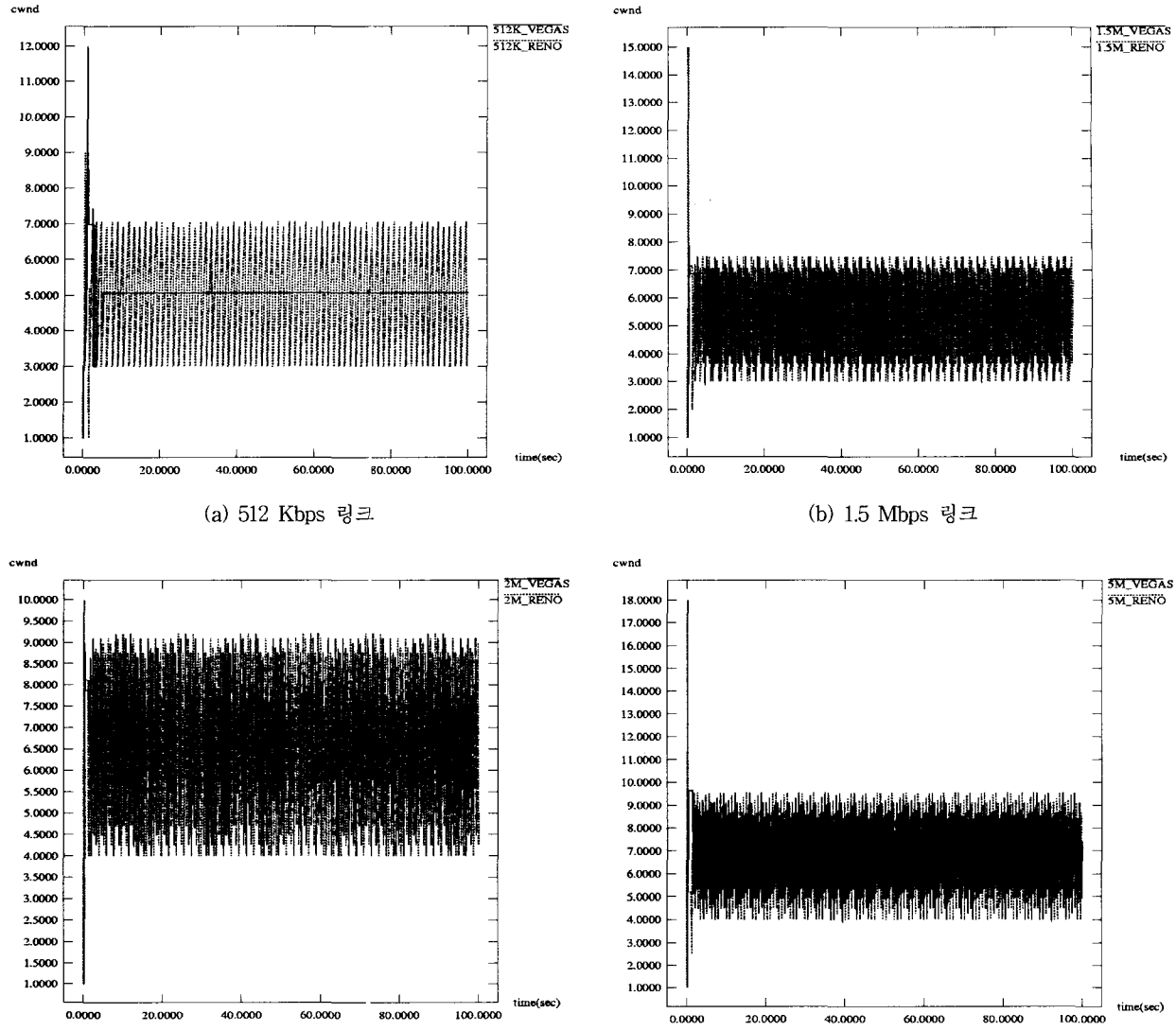
4.2.2 모의실험 결과

TCP Reno 버전과 TCP Vegas 버전에 대한 모의실험의 측정치와 성능, 그리고 공정성에 대한 결과를 <표 2>에 나타내었으며, 모의실험 시 TCP의 성능과 공정성이 가장 좋게 나타난 Q_{thresh} 값을 같이 표시하였다.

〈표 2〉 제안 알고리즘의 모의실험 결과

대역폭(지연)	Q_{thresh} 값	측정치(Kbps)		성능 (Kbps)	공정성(최소/최대) (%)
		TCP Reno	TCP Vegas		
512Kbps(5msec)	4	244.8	267.2	512	91.62
1.5Mbps(2msec)	5	772.89	727.1	1500	94.08
2Mbps(2msec)	6	936.98	1063.02	2000	88.14
5Mbps(2msec)	6	2454.58	2545.42	5000	96.43

모의실험의 결과에서 보면 각 TCP의 성능은 링크에서 제공하는 대역폭을 모두 최대로 사용하고 있으며, TCP Reno와 Vegas간의 공정성이 크게 개선된 것을 알 수 있다. 앞의 2.3절에서 Drop-Tail 큐를 사용하여 실행한 모의실험에서는 TCP Reno와 Vegas간의 공정성이 약 10~12% 정도를 나타내었으나, 최적 윈도우를 고려한 패킷 선별 폐기



(a) 512 Kbps 링크

(b) 1.5 Mbps 링크

(c) 2 Mbps 링크

(d) 5 Mbps 링크

(그림 9) 제안 알고리즘 적용 시의 혼잡 윈도우 크기 변화

알고리즘을 적용한 경우에는 TCP Reno와 Vegas간 공정성이 약 90% 이상으로 개선되었다. 알고리즘의 동작에 사용한 Q_{thresh} 값은 링크의 대역폭이나 전송지연, TCP 연결의 개수 등에 따라 가장 적절한 값을 선택하여야 공정성이 개선되었으며, 각 실험 환경에서 사용한 Q_{thresh} 값을 표에 나타내었다.

TCP Reno 버전의 경우 패킷이 폐기될 때까지 윈도우 크기를 계속 증가시키기 때문에 한꺼번에 많은 수의 패킷이 중간 노드의 송신 큐에 누적된다. 그러나 중간 노드의 송신 큐에 제안 알고리즘을 적용하게 되면 Q_{thresh} 값 보다 많은 수의 패킷이 누적되면 해당 TCP 연결에 의한 데이터를 폐기함으로써 TCP Reno 버전의 혼잡 윈도우 크기의 증가를 억제할 수 있다. 따라서 TCP Reno버전과 TCP Vegas 버전간의 공정성이 개선된다. 다음의 (그림 9)에는 제안 알고리즘을 적용한 모의실험에서 측정한 TCP Reno와 Vegas의 혼잡 윈도우(*cwnd*) 값을 나타내었다.

그림에서 나타난 제안 알고리즘 적용 시의 혼잡 윈도우의 값을 살펴보면 병목 링크의 송신 큐에서 패킷을 선별적으로 폐기함으로써 TCP Reno 버전의 혼잡 윈도우의 증가가 9 이하로 억제됨을 알 수 있다. 또한 이에 따라 평균 윈도우크기가 TCP Vegas 버전과 비슷하게 유지됨으로써 두 버전간의 공정성이 개선되었음을 알 수 있다.

5. 결 론

TCP는 인터넷에서 종단 시스템간의 신뢰성 있는 전송을 담당하는 매우 중요한 프로토콜이다. 특히 혼잡제어와 흐름제어를 수행하는 메커니즘에 따라 그 성능에서 많은 차이

가 날 수 있어 TCP의 혼잡제어 메커니즘과 성능에 대한 많은 연구가 진행되고 있으며, 이에 따른 새로운 버전의 TCP가 발표되고 있다. 또한 최근 활발한 연구가 이루어지고 있는 IntServ[13]나 DiffServ[14] 등과 같은 인터넷에서의 QoS를 보장할 수 있도록 하기 위한 TCP의 윈도우 크기를 조절할 수 있는 큐잉 기법에 대한 연구가 활발히 이루어지고 있다.

현재 인터넷에서 가장 널리 사용되고 있는 TCP로는 TCP Tahoe와 TCP Reno가 있으며, 아직 인터넷에 적용되고 있지는 않지만 TCP Vegas 버전이 발표되었다. 인터넷에서의 성능은 TCP Vegas 버전이 가장 우수한 것으로 알려지고 있으나, TCP 버전간의 공정성(Fairness) 문제로 인하여 실제 적용을 위해서는 많은 연구가 필요한 실정이다.

본 논문에서는 각 버전의 TCP에 대한 성능과 공정성을 측정하기 위하여 다양한 환경에서 모의실험을 수행하였다. 모의실험 결과에서 각 버전의 성능에 있어서는 TCP Reno와 Vegas 버전이 모두 우수하게 나타났다. 그러나 공정성에 있어서는 Reno 버전과 Vegas 버전이 병목링크를 동시에 사용하는 경우에 Vegas 버전을 사용하는 TCP의 성능이 매우 낮은 성능을 보이고 있다. 이는 두 버전의 TCP에서 수행하는 혼잡제어 메커니즘의 차이에서 기인하는 것으로 Reno 버전에 비하여 약 10% 정도의 성능을 보이는 것으로 나타났다. 이러한 공정성을 개선하기 위한 방법으로 최적 윈도우를 고려한 패킷 선별 폐기 알고리즘을 제안하였으며, 모의실험을 통하여 그 성능을 보였다. 제안 알고리즘은 TCP의 성능을 최대로 유지할 수 있는 최적의 윈도우 크기를 기반으로 한 것으로 중간 노드의 송신 큐에서 각 버전의 TCP가 사용하는 윈도우크기를 조절함으로써 공정성을 개선한다. 모의실험 결과에서 보면, 병목구간을 통하여 Reno와 Vegas 버전이 동시에 데이터를 전송하는 경우에는 Reno 버전에 의하여 발생한 트래픽을 선별해서 폐기함으로써 TCP의 공정성을 개선하는 결과를 나타냈으며, 특히 공정성을 높이면서 전체 TCP의 성능에는 아무런 영향없이 높은 성능을 유지하였다.

본 논문에서 제안한 알고리즘은 다양한 버전의 TCP를 동시에 사용하는 경우에 각 TCP 버전의 성능을 공정하게 높일 수 있다. 또한 중간 노드의 큐에서 TCP 연결의 윈도우크기를 조절함으로써 인터넷에서의 서비스 질(QoS)을 보장하기 위한 방법으로도 이용할 수 있을 것으로 사료된다. 이러한 제안 알고리즘을 실제 인터넷에 적용하기 위해서는 FTP 등과 같은 다량의 트래픽뿐만 아니라 Telnet이나 HTTP 등과 같이 불규칙한 트래픽에 대한 대처 방법의 개발이 필요하다.

참 고 문 헌

[1] Lawrence S. Brakmo, Sean W.O'Malley, and Larry L. Peterson, "TCP Vegas : New techniques for congestion detection and avoidance," in *Proceedings of ACM SIGCOMM '94*, pp.24-35, October, 1994.

[2] Lawrence S.Brakmo and Larry L. Peterson, "TCP Vegas : End to End Congestion Avoidance on a Global Internet," *IEEE JSAC*, Vol.13, pp.465-1480, 1995.

[3] O. Ait-Hellal and E. Altman, "Analysis of TCP Vegas and TCP Reno," *IEEE ICC '97*, pp.95-499, 1997.

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

[5] Jin-Ru Chen and Yaw-Chung Chen, "Vegas Plus : Improving the Service Fairness," *IEEE COMMUNICATION LETTERS*, Vol.4, No.5, May, 2000.

[6] Go Hasegawa, Kenji Kurata and Masayuki Murata, "Analysys and Improvement of Fairness between TCP Reno and Vegas for Deployment of TCP Vegas to the Internet," *Network Protocols '2000 Proceedings of International Conference*, pp.177-186, 2000.

[7] Yuan-Cheng Lai, "Improving the Performance of TCP Vegas in a Heterogeneous Environment," *ICPADS '2001 Parallel and Distributed Systems 8th International Conference*, pp.581-587, 2001.

[8] Sally Floyd and Van Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking*, Vol.1, No.4, pp. 397-413, August, 1993.

[9] The Network Simulatorns-2, Nov., 2000, http://www.isi.edu/nsnam/ns/.

[10] W. Richard Stevens, *TCP/IP Illustrated, Volume I : The Protocols*, Addison-Wesley, 1994.

[11] W. Stevens, "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms," *RFC 2001*, Jan., 1997.

[12] V.Paxson M, Allman and W. Stevens, "TCP Congestion Control," *RFC 2581*, April, 1999.

[13] R. Braden, D. Clark, S. Shenker, "Integrated Services in the Internet Architecture : An Overview," *RFC1633*, June, 1994.

[14] Y.Bernet, J. Binder, S. Blake, M. Carson, et al., "A Framework for Differentiated Services," *Internet Draft*, October, 1998.

채 현 석

e-mail : hschae@tongwon.ac.kr

1987년 한양대학교 공과대학 전자공학과 학사

1990년 한양대학교 대학원 전자공학과 석사

1990년~1999년 LG정보통신 중앙연구소 선임연구원

1998년~현재 한양대학교 전자컴퓨터공학부 박사과정

1999년~현재 동원대학 모바일컨텐츠과 조교수

관심분야 : 큐 관리 알고리즘, TCP 성능제어, QoS 라우팅, 고성능 라우터

최 명 렬

e-mail : choimy@asic.hanyang.ac.kr

1983년 한양대학교 공과대학 전자공학과 학사

1985년 미시간 주립대학 전기공학과 석사

1991년 미시간 주립대학 전기공학과 박사

1991년~1992년 생산기술연구원산하 전자부품종합기술연구소 선임연구원

1992년~현재 한양대학교 전자컴퓨터공학부 교수

관심분야 : 스마트카드, DSP 응용, RF/IR 통신, ITS, LCD, ATM/internet