논문 2004-41TC-4-5

네트워크 모델을 이용한 전송제어 프로토콜(TCP)

(TCP Congestion and Flow Control Algorithm using a Network Model)

유 영 일*, 이 채 우**

(Young-Il Yu and Chae-Woo Lee)

요 약

최근에 제안된 TCP Vegas는 네트워크의 상황을 예측하고, 상황에 맞는 전송율의 변화를 통해 흐름제어와 혼잡제어를 함으로써 기존의 TCP Reno에 비해 많은 성능향상을 이루었다. 그러나 TCP Vegas는 네트워크에 대한 모델이 없기 때문에 네트워크의 가용대역폭을 충분히 활용할 정확한 전송율을 계산하지 못하고, 제한된 윈도우의 변화만을 적용시킴으로 인해, 급변하는 가용 대역폭의 변화에 적응을 하지 못하고 손실이 발생할 가능성이 여전히 존재한다. 본 논문에서는 이러한 단점을 극복하고자 TCP의 단대단 연결을 큐잉 시스템으로 모델링하여 적절한 전송율을 결정함으로써, 안정되고 빠르게 가용 대역폭에 수렴하는 알고리듬을 제안한다. 시뮬레이션을 통한 TCP Vegas와의 성능 비교 결과는 제안된 알고리듬이 가용 대역폭이 급변하는 네트워크 환경에서 TCP Vegas 보다 안정적이고 빠르게 반응하기 때문에 외부 트래픽의 변화에 더 잘 적응하고 처리율을 향상시키는 결과를 보여준다.

Abstract

Recently announced TCP Vegas predicts the degree of congestion in the network and then control the congestion window size. Thus it shows better performance than TCP Reno. However, TCP Vegas does not assume any network model, its congestion window control is very limited. Because of this limitation, TCP Vegas still can not adapt to fast changing available bandwidth. In this paper, we introduce a new TCP algorithm which adapts to fast changing available bandwidth well. To devise such a TCP, we model the end to end network of TCP connection as a queueing system and finds congestion window size which can utilize the available bandwidth sufficiently but not make the network congested. The simulation results show that our algorithm adapts to the available bandwidth faster than TCP Vegas and as a results, when the available bandwidth is changing rapidly, our algorithm not only operates more stably than TCP Vegas, but also it shows higher thruput than TCP Vegas.

Keyword : TCP, Congestion Control, Flow Control, Queueing System, Queue Model

I. 서 론

1981년 TCP(Transport Control Protocol)가 RFC(Request For Comment) 표준화 문서로 발표된 이후 지금까지 TCP는 지속적인 발전을 거듭해오고 있다[1-2]. TCP는 여러 변종이 있지만 기본적으로 전송되는 세그먼트에 대한 ACK 정보를 바탕으로 네트워크의 혼잡상황 및 세그먼트의 오류 등을 제어한다. 현재 여러 TCP 변종 중 Reno가 가장 많이 사용되며, Reno의 성능을 개선하기 위한 많은 제안이 이루어지고 있다. TCP Reno는 패킷이 손실되었다고 판단될 경우에만 혼잡 상황으로 인식하고 혼잡윈도우(*CWND: Congestion Window*)를 일정 비율 줄이는 방법을 사용하는데, 가용한 대역폭이 큰 폭으로 변할 경우 제대로 적응을 하지 못하는 단점을 지니고 있다[3][10]. 이는 최근 인터넷 트래픽의 폭발적인 증가와 다양한 멀티미디어의 혼재로 인해 가용 대역폭의 변화가 심한 고속의 링크, 그리고 단순 에러에 의한 패킷 손실이 야기되는 무선 링크에서는 치명적인 TCP의 성능 저하 요인이 될 수 있다[4][7].

TCP Reno의 이러한 단점은 혼잡이 발생했는지의 여부를 오직 패킷 손실만으로 판단하기 때문에 비롯된다.

* 학생회원, **정회원, 아주대학교
(Ajou University)

※ 본 논문은 한국과학재단에서 지원하는 연구지원사업(R01-2003-000-10724-0)의 연구 결과입니다.
접수일자: 2003년11월20일, 수정완료일: 2004년4월6일

가장 최근에 등장한 TCP 버전인 Vegas는 종단간 경로상에서 전송중인 세그먼트의 수를 파악하여, 이로부터 혼잡상황을 예측한다[4][6]. 경로상에 전송중인 패킷의 수, 즉 송신측을 떠났지만 아직 수신측까지 도달하지 못한 패킷이 많다면 이는 네트워크의 버퍼에 전송을 대기중인 패킷의 수가 많다는 것을 의미하므로 혼잡 상황으로 해석한다. TCP Vegas에서는 송신측에서 *RTT* (*Round Trip Time*)마다 전송중인 세그먼트의 수를 계산하여 이 값이 기준값보다 작은 경우 *CWND*를 한 세그먼트만큼 증가시키고, 작은 경우에는 한 세그먼트만큼 감소시켜, 네트워크의 가용 대역폭을 효율적으로 사용한다. 즉, TCP Vegas는 네트워크의 혼잡 정도를 예측하고, 이 정보를 바탕으로 *CWND*의 증감 여부를 결정한다. Vegas는 현재 많이 사용되고 있는 Reno에 비해 많은 성능 향상을 이루었지만, 매 *RTT*마다 변화하는 *CWND*의 크기가 세그먼트 한 개로 제한되므로, 가용한 대역폭의 변화가 클 경우 잘 적응하지 못하는 단점이 있다[4].

이와 같이 가용한 대역폭의 변화가 클 경우 잘 적응하지 못하는 것은 TCP의 전형적인 단점이며, 이는 대부분의 TCP가 네트워크에 대해 적절한 모델을 가지고 있지 않기 때문이다. 따라서 종단간 패킷 전송과 이에 대한 응답인 ACK(Acknowledgment)로 커넥션에 대한 간접 정보를 얻더라도, 네트워크에 대한 모델이 존재하지 않기 때문에 이 정보를 잘 활용할 수 없다. 그 결과 *CWND* 크기의 변화도 커넥션의 상황에 따라 가변적인 것이 아니라 단지 일정량만 증감시킬 뿐이다.

만약 TCP가 종단간 경로에 대한 네트워크 모델을 알고 있다면, 윈도우를 증가시켰을 때 네트워크에 미치는 영향을 예측할 수 있다. 따라서 네트워크를 혼잡 상태로 만들지 않으면서도 가용한 대역폭을 사용할 수 있는 윈도우의 크기를 계산할 수 있다. 그러나 커넥션의 가용 대역폭은 접속하는 호스트의 수, 다른 호스트의 데이터 전송량 등에 따라 변하므로 하나의 TCP 관점에서 본 커넥션 모델을 만드는 것은 쉽지 않다.

그러나 큐 이론[14]의 패킷 도착율이 서비스율에 접근할수록 지연이 길어진다는 일반적인 사실과 지연시간은 패킷을 보낸 시점에서 그 패킷에 대한 ACK가 오는 시간을 통해 알 수 있다는 점을 이용하면, TCP는 송신측의 관점에서 커넥션을 대한 모델을 만들 수 있다. 즉, 송신측에서 계산된 가용 대역폭을 큐에서의 서비스율로, 그리고 일정 시간 동안 송신측에서 전송되었지만 아직 ACK되지 않은 패킷의 수를 큐에서 대기중인 패킷의 수로 볼 경우, TCP의 커넥션은 하나의 큐로 모델링될 수 있다. TCP 커넥션의 혼잡 정도가 심해질수록 일정 시간 동안 송신측에서 전송되었지만 아직 ACK되지 않은 패킷의 수가 많아진다는 사실은 큐로 모델링된 TCP 송신측의 관점에서 보면 커넥션의 가용 대역폭 이용율(utilization)이 증가한 것이다. 큐로 모델링된 TCP 커넥션의 가용 대역폭 이용율은 커넥션의 혼잡 정도를 표현하고 TCP 송신측의 알맞은 전송율은 가용 대역폭의 이용율에 따라 결정된다. 종단간 정보만을 이용하는 TCP 송신측은 커넥션의 가용 대역폭 이용율을 알기 위해 송신측에서 전송되었지만 아직 ACK되지 않은 패킷 개수를 예측하는 것이 필요하며 이는 TCP Vegas에서 제시하고 있다[6]. 따라서 큐 이론을 적용하여 TCP 커넥션의 가용 대역폭 이용율을 예측하고 네트워크의 혼잡 상황에 맞는 *CWND*의 크기를 매 *RTT*마다 결정함으로써 TCP 송신측이 목표로 하는 가용 대역폭의 이용율에 안전하게 그리고 빨리 도달할 수 있는 알고리듬을 만들 수 있다.

본 논문에서는 TCP Vegas의 네트워크 혼잡 정도를 계산하는 방법을 바탕으로 큐 이론을 이용하여 이용율과 대기중인 패킷과의 관계를 유도한 후, 매 *RTT*마다 알맞은 전송율로 데이터를 보낼 수 있도록 *CWND* 크기를 결정하는 새로운 TCP 알고리듬을 제안하며 이를 알고리듬을 QM-TCP(Queue Model-based TCP) 라 한다. 논문의 구성은 다음과 같다. Ⅱ장에서 TCP Vegas의 혼잡제어 알고리듬에 대한 간략한 설명을 하고, Ⅲ장에서 본 논문에서 제안하는 알고리듬에 대해 설명한다. 그리고 Ⅳ장에서는 시뮬레이션을 통해 TCP Vegas와 제안한 알고리듬의 성능을 비교 분석하며 Ⅴ장에서 논문을 정리한다.

Ⅱ. TCP Vegas의 혼잡 제어 메커니즘

TCP Vegas는 송신측에서 커넥션의 가용 대역폭 이상에 해당되는 전송율로 보낼 경우에 일정 시간 동안 커넥션 내의 큐에 쌓이는 데이터의 양을 예측하여 혼잡 정도를 판단하고 이를 제어한다[6][13]. 이 데이터의 양을 잉여 데이터(extra data)[5]라고 정의를 하며 TCP가 커넥션의 가용 대역폭을 모두 사용하지 않으면 잉여 데이터는 존재하지 않는다. TCP Vegas의 송신측은 커넥션의 상황을 직접 알 수 없기 때문에 *CWND*값과 계산

된 *RTT*를 통해 간접적으로 잉여 데이터를 예측하고 미리 정해진 기준값과 비교함으로써 *CWND*값의 단위 증감여부를 결정한다. 즉, 잉여 데이터가 기준값보다 클 경우에는 혼잡 상황이 발생할 가능성이 높은 것으로 해석하여 *CWND*를 줄이고, 기준값보다 작을 경우에는 대역에 여유가 있다고 판단하여 *CWND*를 늘린다[13]. 커넥션 링크의 상황을 예측하기 위해 커넥션 전체를 하나의 가상 버퍼로 볼 경우 잉여 데이터는 그 가상의 버퍼에 쌓여 있는 패킷의 양으로 생각할 수 있다. 또는 이를 TCP의 송신측에서는 위치나 상태를 파악할 수 없을 것으로 생각되는 정체된 링크의 버퍼에 쌓인 데이터의 양으로 받아들여도 무방하다. 그 이유는 커넥션 내 정체된 링크 외의 링크에 가용 대역폭의 여유가 있더라도 정체된 링크는 가용 대역폭을 모두 사용하여 버퍼에 데이터가 쌓이기 때문이다. TCP Vegas는 이와 같이 커넥션의 상황을 예측하는 잉여 데이터 산출을 통해 전송율을 조절함으로써 혼잡 상황을 피하고 가용 대역폭을 효율적으로 사용할 수 있다. TCP Vegas에서 사용하는 알고리듬에 대해 자세히 설명하면 다음과 같다.

TCP Vegas는 단위 시간 동안 커넥션을 관찰했을 경우 커넥션 링크에 존재하는 세그먼트의 양에서 커넥션의 가용 대역폭이 처리할 수 있는 세그먼트의 양을 뺌으로써 잉여 데이터를 구하는데 아래 식 1, 2과 같이 정의되는 *기대 처리율(expected throughput)*과 *실제 처리율(actual throughput)*을 이용한다.

커넥션의 중간 노드 버퍼에 데이터가 쌓여 있지 않을 경우 보내진 세그먼트에 의한 *RTT*를 *BaseRTT*라 정의하며, 실제 구현에서는 측정된 *RTT*의 최소 값이다. 이 때 커넥션에 잉여 데이터가 발생하지 않고 최소 *RTT* 동안 *CWND* 크기만큼 전송할 경우, 기대되어지는 전송율을 다음과 같이 *기대 처리율*로 정의한다.

$$\text{기대 처리율} = CWND / BaseRTT. \quad (1)$$

그리고 *RTT*마다 얼마만큼의 데이터가 전송되었는지를 다음과 같이 실제 처리율로 정의한다.

$$\text{실제 처리율} = CWND / BaseRTT. \quad (2)$$

두 값을 이용하면 하나의 *RTT*동안 실제 송신측에서 네트워크로 보낸 데이터는 (*기대 처리율*× *BaseRTT*)이지만 *BaseRTT* 동안 커넥션의 정체된 링크에서 처리할 수 있는 양은 (*실제 처리율*× *BaseRTT*)이기 때문에 송신측에서 추측할 수 있는 잉여 데이터는 Little의 법칙에 의해 다음 식으로 표현된다[14].

$$\text{잉여 데이터}(\delta) = (\text{기대 처리율} - \text{실제 처리율}) \cdot BaseRTT. \quad (3)$$

실제 TCP에서는 세그먼트를 *CWND*의 기본단위로 사용한다. 그러므로 위 식의 잉여 데이터에 대한 기본 단위 역시 세그먼트이다. 그리고 만약 *실제 처리율*이 *기대 처리율*보다 클 경우에는 측정된 *RTT*가 *BaseRTT*보다 작음을 의미함으로 *BaseRTT*는 갱신된다[5-6][13].

마지막으로 TCP Vegas는 잉여 데이터로 커넥션의 상황을 예측하여 *CWND* 크기의 증가, 감소, 그리고 고정 이 세 가지에 대한 기준을 정하는데 *CWND*의 크기를 고정시키는 제어는 목표로 하는 잉여 데이터를 적절한 영역으로 설정함으로써 *CWND* 크기의 불필요한 진동을 막고 안정된 제어가 가능하도록 한다. 그 영역의 경계로 $\alpha < \beta$의 관계인 α와 β 두 값을 설정한다면 $\alpha \le \delta \le \beta$인 경우는 잉여 데이터가 적당한 때, $\delta < \alpha$는 잉여데이터가 적을 때, $\delta > \beta$는 잉여 데이터가 많을 때를 뜻한다. 일반적으로 TCP Vegas에서는 $\alpha = 1$, $\beta = 3$을 사용하며, *CWND*는 매 *RTT*마다 아래와 같은 방식으로 갱신된다[15].

$$CWND_{new} = \begin{cases} CWND_{current} + 1 & (\delta < \alpha) \\ CWND_{current} & (\alpha \le \delta \le \beta) \\ CWND_{current} - 1 & (\beta < \delta) \end{cases} . \quad (4)$$

여기서 $CWND_{current}$는 현재 측정된 *CWND*의 크기이며, $CWND_{new}$는 다음 *RTT* 동안 적용될 *CWND*의 크기이다. TCP Vegas는 그림 1과 같이 송신측에서 계산된 잉여 데이터를 안정 영역인 와 β 사이에 놓기 위해 *CWND*를 단위 세그먼트만큼 늘이거나 줄여준다. 여기서 $\Delta CWND$는 매 *RTT*마다 변화시키는 *CWND*의 크기로, 다음 식과 같이 $CWND_{new}$에서 $CWND_{current}$를 뺀 값이다.

$$\Delta CWND = CWND_{new} - CWND_{current}. \quad (5)$$

이 알고리듬은 변화하는 커넥션의 혼잡 정도에 따라 알맞은 잉여 데이터를 유지하고 커넥션의 가용 대역폭을 충분히 활용하게 한다[6]. 즉, 혼잡 상황을 제어할 때 미리 네트워크 상황을 예측하고 *CWND*의 크기를 변화시키는 것으로써 Reno에 비해 버퍼 오버플로우(Overflow)나 타임아웃(Timeout)을 감소시켜 보다 나은 성능 향상을 유도한다.

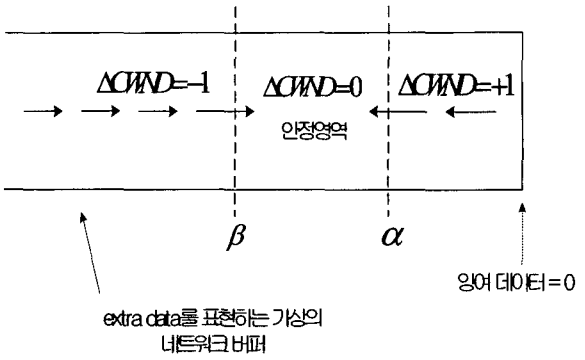


그림 1. 커넥션의 경로를 버퍼로 모델링한 TCP Vegas의 혼잡 윈도우(*CWND*) 제어

Fig. 1. Control of Congestion Window in TCP Vegas which models TCP connection as a buffer

그러나 TCP Vegas는 매 *RTT*마다 잉여 데이터의 크기를 안정 영역인 α와 β의 사이에 놓기 위해 *CWND*의 크기를 증감시키지만 변화하는 크기가 세그먼트 한 개의 크기로 제한되므로, 고속의 링크에서 가용한 대역폭의 변화가 클 경우 잘 적응하지 못한다. 따라서 *CWND*의 크기가 증가할 때는 비효율적인 대역폭의 사용, 그리고 감소할 때에는 버퍼 오버플로우나 타임아웃의 발생과 같은 치명적인 약점을 지닌다[4].

Ⅲ. 큐 모델을 사용한 TCP 알고리즘(QM-TCP)

TCP Vegas는 매 *RTT*마다 하나의 세그먼트만을 증감시키기 때문에 급격하게 변하는 가용 대역폭에 빨리 수렴하지 못한다. 이러한 단점은 TCP 송신측에서 네트워크의 혼잡 상황에 맞게 *CWND*의 크기를 결정하고 적용하면 해결된다. 그러나 계층화된 네트워크 구조에서 TCP가 현재처럼 종단간 방식의 제어를 유지한다면 네트워크 정보를 예측하기 힘든 TCP의 송신측은 정확하게 이 전송율을 결정할 수 없다[6]. 물론 ECN(Explicit Congestion Notification) 또는 ELN(Expl-icit Loss Notification)처럼 커넥션의 중간 노드에서 패킷 손실에 대한 정보나 링크의 가용 대역폭 및 버퍼 상태 정보를 송신측으로 보내준다면 네트워크 정보의 부족으로 인한 문제는 발생하지 않는다[8]. 그러나 이는 다양하고 넓게 퍼져 있는 전 세계 모든 망의 라우터들 위에 기능이 추가되고 중간 노드 정보를 전송할 수 있는 패킷 구조가 포함될 때 가능한 시나리오이므로 현실적인 접근 방법이 되지 않는다. 그렇기 때문에 종단간 프로토콜을 유지하는 흐름 및 혼잡제어 알고리듬의 개발은 상당히 어려운 과제이다. 이번 절에서는 이러한 TCP의 문제 해결을 위해 큐잉 시스템으로 TCP 커넥션을 모델링하고 전송율을 결정하는 방법을 제안한다.

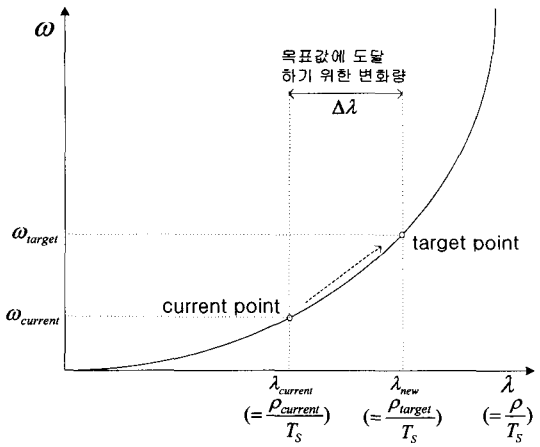


그림 2. 일반적인 큐 모델에서의 패킷 도착율(λ)과 큐에 대기중인 패킷 양(w)의 관계

Fig. 2. Packet arrival rate(λ) versus mean number of packets waiting to be served(w)

TCP Vegas에서와 같이 잉여 데이터가 TCP 송신측에서 전송한 트래픽에 의해 TCP 커넥션에 쌓여 있는 패킷의 양으로 비유된다면 TCP에서 바라본 커넥션의 경로는 하나의 큐로 모델링될 수 있다. 큐는 일반적으로 패킷 하나에 대한 서비스 시간(T_s)이 일정할 경우 이용률(ρ)은 패킷 도착율(λ)과 비례하게 되고, 패킷 도착율(λ)이 증가함에 따라 큐에 쌓이는 패킷의 양(w)은 아래의 그림 2와 같이 지수함수 형태로 증가하게 되는 특성을 지닌다.

그림 2에서 현재의 패킷 도착율($\lambda_{current}$)에 해당하는 큐에 대기중인 패킷의 양은 $w_{current}$이고, 큐에 대기중인 목표로 하는 패킷의 양(w_{target})을 만족시키는 패킷 도착율은 λ_{new}이다. 그리고 $\rho_{current}$와 ρ_{target}은 각각의 이용율을 나타낸다. 일반적인 큐 모델은 다음과 같이 T_s가 일정할 경우 같은 시점의 λ와 ρ는 항상 비례 관계에 놓이게 된다.

$$\rho = \lambda \cdot T_s \tag{6}$$

위 식은 모든 λ와 ρ에 대한 관계를 나타내므로 λ_{new}는 다음과 같다.

$$\lambda_{new} = \frac{\rho_{target}}{\rho_{current}} \cdot \lambda_{current} \tag{7}$$

TCP의 커넥션을 큐로 모델링하면 매 RTT마다 계산되는 TCP의 전송율은 패킷 도착율(λ), 세그먼트를 하나 서비스하는데 걸리는 시간은 패킷 하나에 대한 서비스 시간(T_s), TCP 가용 대역폭의 이용율은 이용율(ρ), 그리고 잉여 데이터는 큐에 쌓이는 패킷의 양(w)으로 표현된다. 목표로 하는 잉여 데이터(w_{target})에 도달하기 위한 윈도우 크기의 변화량($\Delta CWND$)은 커넥션의 혼잡 정도를 가장 정확하게 표현하는 현재의 RTT ($RTT_{current}$)동안 계산된 현재의 전송율($\lambda_{current}$)을 목표로 하는 전송율(λ_{new})로 바꿔주기 위한 값이 된다. TCP의 커넥션을 큐잉 시스템으로 모델링했을 때 윈도우 크기 변화량에 관한 식을 표현하면 다음과 같다.

$$\Delta CWND = (\lambda_{new} - \lambda_{current}) \cdot RTT_{current} \quad (8)$$

매 RTT마다 위의 $\Delta CWND$를 적용하면 커넥션의 상황에 보다 빨리 반응하여 주어진 대역폭을 효율적으로 사용할 수 있으며, TCP 커넥션으로 유입되는 외부 트래픽으로 인해 가용 대역폭이 급변하더라도 목표로 하는 잉여 데이터에 빨리 수렴시킴으로써 혼잡 상황으로 야기되는 성능 저하를 막는다. TCP에 적용할 전송율에 대한 잉여 데이터의 함수식은 몇 가지의 큐 모델들에서 유도할 수 있는데 비교적 구현이 간단하고 TCP의 특성에 따르는 M/D/1 큐 모델[11]을 선택한다. TCP 커넥션을 표현하기 위해 M/D/1 큐 모델을 선택한 이유는 다음과 같다.

첫째, 비록 TCP가 매 RTT마다 전송율의 변화가 심한 특성을 나타내고 있지만 고속의 다양한 트래픽이 존재하는 상황이라면 TCP 커넥션에는 입력 트래픽을 포아송 도착(Poisson arrival)으로 가정할 수 있다.

둘째, TCP는 세그먼트의 크기가 대부분 고정되어 있어서 전송하는 기본 단위가 일정하다. 세그먼트 하나를 서비스하는데 걸리는 시간은 커넥션의 가용 대역폭이 변하더라도 실제 처리율의 계산을 통해 산출된다. 이 시간은 매 RTT마다 적용되므로 전송율을 결정하는 시점에는 단위 RTT 동안 고정되어 있다고 볼 수 있다.

셋째, 정체된 링크의 버퍼를 TCP에서 바라본 네트워크에 해당하는 가상의 버퍼로 해석하기 때문에 서버의 개수는 하나로 놓는다.

정확한 예측이 불확실한 TCP 커넥션을 정확한 M/D/1 큐 모델의 환경에 맞출 수는 없다. 그러나 매 RTT마다 변화시킬 윈도우의 크기를 결정할 수는 있는 전송율(λ)에 대한 잉여 데이터(w)의 함수관계를 제공한다는 사실은 M/D/1 큐 모델의 적용 근거로 충분하다. TCP 커넥션에서 가용 대역폭의 이용율(ρ)은 송신측에서 네트워크의 혼잡 상황을 예측할 수 있는 수단이 되고, M/D/1 큐의 경우 이용율에 대한 큐에 대기중인 패킷의 양은 다음과 같이 주어진다[16].

$$w = \frac{\rho^2}{2(1-\rho)} \quad (9)$$

위식을 ρ에 대해 풀면 다음과 같은 식을 얻는다.

$$\rho = -w + \sqrt{w^2 + 2w} \quad (10)$$

현재의 전송율($\lambda_{current}$)은 다음과 같이 현재의 RTT ($RTT_{current}$)동안 전송되는 윈도우의 크기($CWND_{current}$)로 계산되며,

$$\lambda_{current} = \frac{CWND_{current}}{RTT_{current}} \quad (11)$$

위 식의 현재의 전송율($\lambda_{current}$)이 식(7)에 적용되면 목표로 하는 전송율(λ_{new})은 다음 식과 같다.

$$\lambda_{new} = \left(\frac{\rho_{target}}{\rho_{current}}\right) \cdot \left(\frac{CWND_{current}}{RTT_{current}}\right) \quad (12)$$

위 식의 $\rho_{current}$와 ρ_{target}는 각각 식 10을 이용하여 현재의 잉여 데이터($w_{current}$)와 목표로 하는 잉여 데이터(w_{target})로부터 구해진다. 식 8에서 구한 윈도우 크기의 변화량($\Delta CWND$)을 식 11, 12를 이용하여 구하면 다음과 같다.

$$\Delta CWND = \left(\frac{\rho_{target}}{\rho_{current}} - 1\right) \cdot CWND_{current}. \quad (13)$$

그러나 사실상 예측이 힘든 커넥션의 상황을 정확하게 판단하기 위한 척도로 M/D/1 큐 모델을 적용하는 것은 정확하지 않기 때문에 그림 2의 ρ에 대한 w의 그래프를 적용할 때 몇 가지 기본적인 파라미터들의 미세한 설정 및 수정이 요구된다.

첫째, 매 RTT마다 적용되는 TCP 커넥션에 대한 큐 모델은 완전하지 않기 때문에 현재의 이용율은 하나의 정해진 목표 이용율에 수렴하지 않고 그 부근에서 진동한다. 매 RTT마다 네트워크의 상황이 변하고, TCP에서 네트워크의 혼잡 정도를 나타내는 측정된 RTT의 지연이 커지면 이러한 진동은 더욱 불안정하게 커지게 된다. 본 논문에서는 TCP Vegas와 유사한 방식으로

측정된 이용률이 일정 구간에 위치할 경우에 $CWND$의 크기에 변화를 주지 않음으로써 진동을 없애는 방법을 적용한다. 이 때 적용되는 안정영역의 경계는 TCP Vegas와의 비교를 위해 일반적인 TCP Vegas($\alpha=1$, $\beta=3$)의 α와 β에 해당하는 이용률을 사용하며 목표로 하는 이용률은 이 두 값의 중간값인 $(\alpha+\beta)/2$를 사용한다. 식 10을 이용하여 α에 해당하는 이용률(ρ_α), β에 해당하는 이용률(ρ_β), $(\alpha+\beta)/2$에 해당하는 이용률(ρ_{target})을 각각 구한다.

$$\begin{cases} \rho_\alpha = 0.732, & w = \alpha(=1) \\ \rho_{target} = 0.828, & w = target(=2) \\ \rho_\beta = 0.873, & w = \beta(=3) \end{cases} \quad (14)$$

둘째, 그림 2를 보면 잉여 데이터(w)가 작을 경우에는 잉여 데이터가 조금 변하더라도 ρ가 크게 변하는 특징을 볼 수 있다. 잉여 데이터(w)가 작을 경우에 잉여 데이터를 계산에 약간의 오차가 생겨도, 그것에 의해 계산된 가용 대역폭 이용율(ρ)은 네트워크 모델이 정확히 M/D/1인 경우라도 큰 오차가 발생된다. 따라서 계산된 잉여 데이터가 작을 경우에 발생하는 불안정한 전송율 제어를 막기 위해 $\rho_{current}$의 한계 최소값을 설정한다. 본 논문에서는 여러 번의 시뮬레이션을 통해 목표로 하는 잉여 데이터($w_{target}=2$)의 30%에 해당되는 이용율(=0.65)을 $\rho_{current}$의 한계 최소값으로 정한다. 즉, $w_{current}$가 0.6보다 작으면 이용율의 한계 최소값인 0.65를 $\rho_{current}$로 사용한다.

셋째, 식(13)을 통해 구해진 $\Delta CWND$가 큰 값을 갖고 단위 RTT동안 적용이 된다면 부정확한 네트워크 모델 때문에 순간적으로 오버플로우나 타임아웃이 발생할 수 있다. 이는 M/D/1 큐 모델이 실제 TCP의 커넥션과 정확히 일치하지 않기 때문이다. 그리고 TCP의 송신측이 $CWND_{new}$를 네트워크에 전송하면 그것은 1 RTT 후에 네트워크에 영향을 미치고 송신측은 2 RTT 후에 측정된 RTT를 통해 네트워크의 혼잡 정도를 알 수 있다. 그렇기 때문에 송신측에서 최적의 전송율 제어 주기는 1 RTT가 아니라 2 RTT가 되어야 한다[2]. 그러나 실제 TCP를 2 RTT마다 제어한다면 커넥션의 가용 대역폭이 변화하는 상황에 빨리 전송율을 적응시킬 수 없으므로 기본적으로 TCP는 1 RTT마다 제어를 한다. 따라서 식(13)에서 구한 $\Delta CWND$를 매 RTT마다 적용하기 위해 이 값을 2로 나누어줄 필요가 있으며, 본 알고리듬에서 사용한 M/D/1 큐 모델의 부정확도를 고려한다면 안전한 $CWND$제어를 위해 이 값을 더 큰 비율로 줄여야 한다. 본 논문에서는 아래와 같이 식 13에서 구해진 $\Delta CWND$를 다양한 네트워크 시뮬레이션을 통해 선택된 값인 3.5로 나눠준다.

$$\Delta CWND = \frac{\Delta CWND}{3.5} = \frac{(\lambda_{new} - \lambda_{current})}{3.5} \cdot RTT_{current} \quad (15)$$

그리고 전송율이 급변하는 것을 방지함으로써 보다 안전하게 제어가 가능하도록 하기 위해 위의 식에서 구한 $\Delta CWND$을 아래와 같이 ±4로 제한한다. 이 값 역시 다양한 시뮬레이션을 통해 산출된 값이다. 매 RTT마다 적용되는 $\Delta CWND$이 최대 ±4로 제한되더라도 기존의 TCP Vegas에서 제시한 ±1보다는 크기 때문에 훨씬 빨리 목표로 하는 잉여 데이터의 값에 도달할 수 있다.

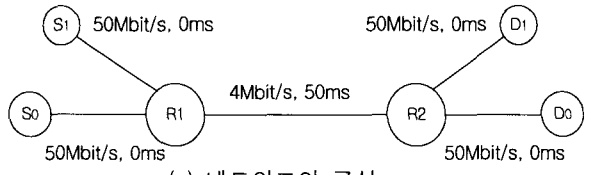
$$|\Delta CWND| \leq 4 \quad (16)$$

위와 같은 세 가지 수정을 포함한 QM-TCP는 매우 안정적으로 작동하고 TCP Vegas보다 빠르게 가용 대역폭에 수렴한다. 다음 절에서는 시뮬레이션을 통해 QM-TCP와 TCP Vegas의 성능을 비교하고 QM-TCP의 알고리듬이 TCP Vegas보다 개선된 성능을 보여줌을 확인한다.

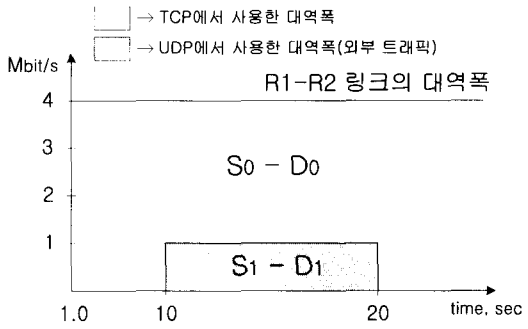
IV. 제안된 알고리듬(QM-TCP)과 TCP Vegas의 성능 비교

본 절에서는 커넥션이 급격히 변하는 유동적인 가용 대역폭을 가지는 경우에 본 논문에서 제안하는 QM-TCP와 기존의 TCP Vegas에 대한 성능 분석을 한다. 성능분석을 두 종류의 일반적인 시나리오를 설정했는데, 첫 번째 시나리오는 커넥션에 하나의 외부 트래픽이 유입되어 중간에 한 번 가용 대역폭을 바꾼 경우이고, 두 번째 시나리오는 다수의 외부 트래픽이 가용 대역폭을 여러 번에 걸쳐 지속적으로 변화시킨 경우이다. 본 논문의 시뮬레이션은 NS-2[17]를 사용하여 수행했다.

시나리오 A. TCP 커넥션 링크에서 가용 대역폭이 급격하게 한 번 바뀌는 경우



(a) 네트워크의 구성
(a) Network configuration



(b) R1-R2 링크의 입력 트래픽
(b) Input traffic of R1-R2 link

그림 3. 시나리오 A의 시뮬레이션 환경
Fig. 3. Simulation scenario A

그림 3의 (a)와 (b)는 각각 시나리오 A에 대한 네트워크의 구성과 R1-R2의 대역폭을 사용하는 입력 트래픽을 보여주고 있다. 그림 3의 (a)에서 노드 S0는 D0에게 0~30s(초)동안 FTP를 이용하여 세그먼트의 크기가 1KB인 TCP로 큰 파일을 전송한다. 그리고 노드 S1은 D1에게 10~20s동안 UDP로 1Mbps의 데이터를 전송한다. R1-R2 구간은 4Mbps의 대역폭에 50ms의 전송 지연 시간을 가지며 S0, S1-R1 구간과 R2-D0, D1 구간은 50Mbps에 0ms의 전송지연 시간을 가진다. 그리고 R1, R2의 버퍼 크기는 각각 20KB의 크기를 갖는다.

시뮬레이션은 외부 트래픽으로 인해 가용 대역폭이 급격히 변할 경우에 빨리 새로운 가용 대역폭에 수렴하는 신속성과 커넥션의 안정성에 초점을 맞춘다. 외부 트래픽이 TCP 커넥션의 중간에 한 번 유입되었다가 나갈 경우 QM-TCP와 기존의 TCP Vegas간의 성능 비교를 위해 S0의 *CWND* 크기와 보내는 방향의 정체된 링크에 위치한 R1의 버퍼 크기를 확인한다. TCP는 신뢰할 수 있는 서비스를 제공하므로 *CWND*의 크기 자체가 데이터 전송율, 수신율과 같은 성능을 표현한다고 볼 수 있으며, R1 버퍼에 쌓여 있는 패킷의 양은 커넥션의 혼잡 정도를 나타낸다.

TCP의 세그먼트 크기가 1KB이고 *BaseRTT*가 약 100ms이므로 송신측이 외부 트래픽없이 4Mbps의 R1-R2 링크 대역폭을 정확하게 모두 사용하면 *CWND*

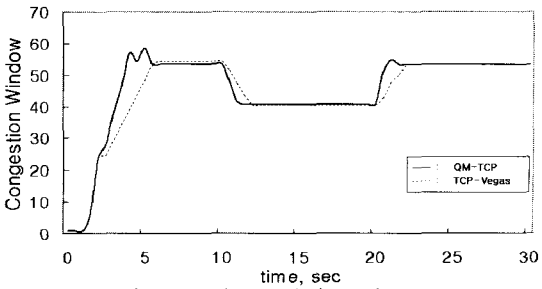


그림 4. 혼잡 윈도우(CWND)의 크기
Fig. 4. Congestion Window Size

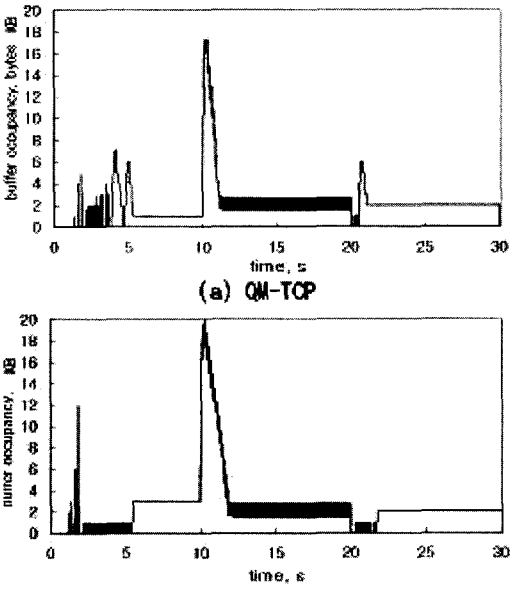


그림 5. R1 버퍼에 쌓여 있는 패킷의 양
Fig. 5. Buffer Occupancy of R1

크기는 아래 식에 의해 약 50개의 세그먼트가 된다.

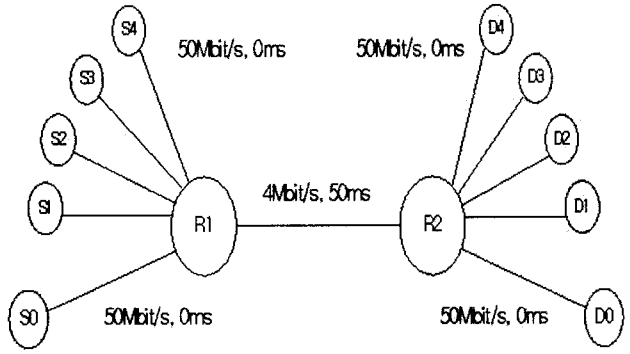
$$50\text{세그먼트} = \frac{R1-R2\ \text{링크의 대역폭}}{\text{단위 세그먼트의 크기}} \times BaseRTT$$

$$= \frac{4Mbps}{1KB} \times 0.1s \qquad (17)$$

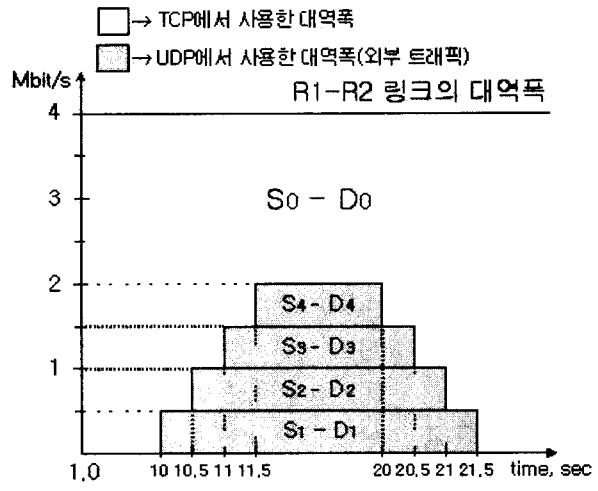
그림 4와 5에서 R1-R2 링크에 외부 트래픽이 없을 경우에는 *CWND*가 52개의 세그먼트이며 R1의 버퍼에는 2개의 세그먼트가 대기 중이다. 그리고 외부 트래픽이 유입되는 10s~20s에는 CWND가 약 40개의 세그먼트이며 R1의 버퍼에는 2개의 세그먼트가 대기중이다. 주목할만한 세 부분은 가용 대역폭이 갑작스레 바뀌는 경우인 1s(TCP 커넥션이 성립되는 순간), 10s(UDP 플로우의 생성이 시작되는 순간), 20s(UDP 플로우의 생성이 중단되는 순간)이다. 우선 커넥션이 성립되는 순간 즉, 1s부터 가용 대역폭까지 이르는데 걸리는 시간을 보면 큐 모델을 이용한 TCP가 2.61s, TCP Vegas가

4.16s로 2.54s정도의 큰 차이를 보여준다. 이는 QM-TCP가 매 *RTT*마다 1보다 큰 *ΔCWND*를 적용하기 때문에 TCP Vegas에 비해 빨리 목표로 하는 잉여 데이터 값으로 수렴함을 나타낸다. 둘째, 10s(UDP 플로우의 생성이 시작되는 순간)에는 QM-TCP가 매 *RTT*마다 *CWND*를 하나 이상으로 줄이기 때문에 약 0.72s 빨리 가용 대역폭에 맞춰가며 수렴함을 볼 수 있는데 이 순간 R1의 버퍼를 나타내는 그림 5를 보면 (a)가 (b)에 비해 훨씬 급한 경사를 이루며 버퍼 크기가 감소함을 알 수 있다. 그렇기 때문에 버퍼의 안정도면에서도 우세함이 확인된다. 셋째, 20s(UDP 플로우의 생성이 시작되는 순간)에도 QM-TCP는 매 *RTT*마다 1보다 큰 *ΔCWND*의 적용으로 Vegas에 비해 약 0.79s 먼저 가용 대역폭에 도달한다.

시나리오 B. TCP 커넥션 링크에서 가용 대역폭이 계단 형태로 짧은 시간동안 수차례 바뀌는 경우



(a) 네트워크의 구성
(a) Network configuration



(b) R1-R2 링크의 입력 트래픽
(b) Background traffic

그림 6. 시나리오 B의 시뮬레이션 환경
Fig. 6. Simulation scenario B

그림 6의 (a)와 (b)는 각각 시나리오 B에 대한 네트워크의 구성과 외부 트래픽에 따른 TCP 커넥션의 가용 대역폭 변화를 보여준다. 이번에는 짧은 시간동안 지속적으로 변화하는 외부 트래픽에 적응하는 TCP의 성능을 비교하기 위해 시나리오 A에서 했던 시뮬레이션 설정을 약간 수정하였다. 외부 트래픽은 S1~S4에서 R1~R4로 아래의 그림과 같이 500Kbps UDP를 네 번에 걸쳐 계단 형태로 10s~21.5s 동안 입력한다. 그림 7과 그림 8을 보면 QM-TCP는 외부 트래픽이 계단 형태로 짧은 시간동안 계속 유입되더라도 꾸준하게 적응을 하지만 TCP Vegas는 타임아웃이 발생하여 결국에는 *CWND*가 초기값인 1까지 감소함을 알 수 있다. 이는 매 *RTT*마다 단위 세그먼트의 *CWND*를 증감시키는 TCP Vegas가 순간적으로 큰 가용 대역폭의 변화에 제대로 적응하지 못하기 때문이다.

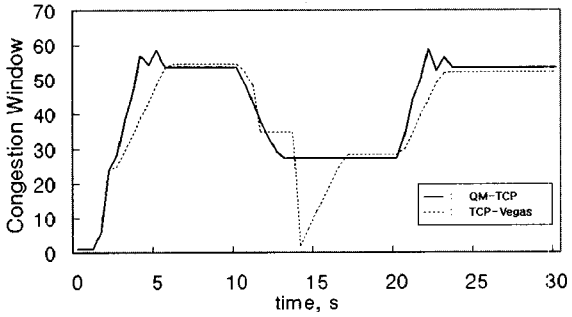


그림 7. 혼잡 윈도우(CWND)의 크기
Fig. 7. Congestion Window Size

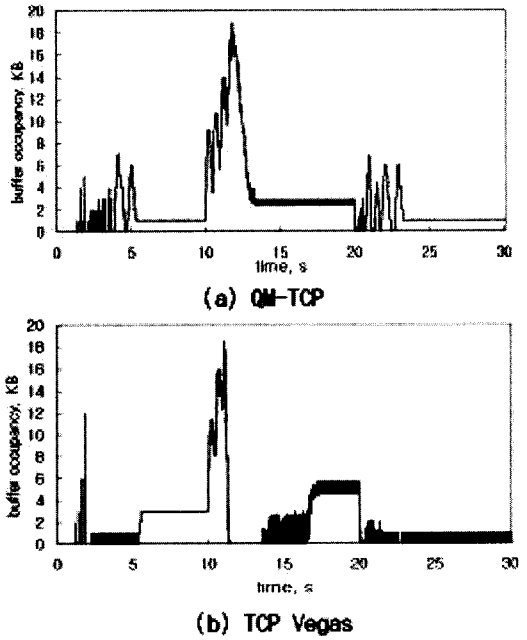


그림 8. R1 버퍼에 쌓여 있는 패킷의 양
Fig. 8. Buffer Occupancy of R1

위에서 분석한 두 가지 시나리오의 *CWND*의 크기와 버퍼의 크기를 전체적으로 비교해 보면 QM-TCP는 TCP Vegas보다 빠른 시간 내에 변화된 가용 대역폭에 수렴하고 네트워크에 쌓여있는 패킷의 양도 역시 TCP Vegas보다 적거나 같도록 유지시킨다. 이는 혼잡 상황을 빨리 벗어나게 함으로써 타임아웃이나 오버플로우의 발생을 예방하는 혼잡 제어를 가능하게 하며, 가용 대역폭이 낭비되는 시간을 줄이고 커넥션의 실제 처리율을 향상시킴으로써 안전하고 효율적인 흐름 제어를 가능하게 한다.

V. 결 론

본 논문은 빠르게 변화하는 가용 대역폭에 잘 적응하는 새로운 TCP 알고리듬을 제안했다. 제안된 알고리듬은 매 *RTT*마다 단위 세그먼트 이상으로 혼잡 윈도우의 크기를 변화시킴으로써 TCP Vegas보다 나은 성능을 나타낸다. TCP는 주어진 정보가 극히 제한적이기 때문에 TCP 커넥션의 상황에 맞도록 일정한 모델을 제시하고 이를 제어하기가 사실상 어렵다. 그러나 TCP 커넥션에 Little의 법칙을 적용시켰을 때 알 수 있는 커넥션의 이용률과 큐에 쌓인 평균 패킷 개수간의 관계는 TCP 커넥션을 큐잉 시스템으로 모델링할 수 있도록 했다. 비록 큐잉 시스템이 TCP 커넥션을 모델링하기에는 완벽하지 않았지만 다양한 네트워크 환경에서의 시뮬레이션을 통해 부분적으로 수정을 거친 큐로 모델링된 TCP(QM-TCP)는 TCP Vegas에 비해 혼잡 윈도우의 변화폭을 적절히 조절함으로써 혼잡제어 및 흐름제어에서 상대적으로 뛰어난 성능을 보여 주었다. 시뮬레이션에서 볼 수 있듯이 매 *RTT*마다 혼잡 윈도우의 변화량을 커넥션의 상황에 맞춰 증감시키는 제어는 TCP Vegas에 비해 급격하고 끊임없이 변화하는 가용 대역폭에 보다 안정되고 빠르게 수렴했다. 큐 모델을 적용한 TCP 알고리듬은 기존의 TCP 버전들에 비해 외부 트래픽 상황에 영향을 덜 받으면서 처리율을 향상시킬 수 있는 안정된 결과를 보여줬다. 만약 우리가 보다 정확한 큐 모델을 사용하거나 TCP 커넥션에 대한 많은 정보를 알고 있다면 더 적절한 혼잡 윈도우의 크기를 결정할 수 있을 것이다. 우리는 앞으로 다양한 네트워크 환경에서 다양한 알고리듬과 큐 모델을 적용한 TCP의 성능 분석 및 개선에 관해 연구할 필요가 있다.

참 고 문 헌

[1] J. Postel, "*Transport control protocol*," Request for Comments 793, DDN Network Information Center, SRI International, September 1981.

[2] V. Jacobson, "Congestion avoidance and control," in Proceedings of ACM SIGCOMM, pp.134-143, August 1988.

[3] W. Stevens, "TCP/IP Illustrated, Volume 1," Addison-Wesley Publishing Company, pp.223-242, 1994.

[4] W. Xu and A.G. Qureshi, "Novel TCP congestion control scheme and its performance evaluation," IEE Proceedings Communications, vol.149, no.4, pp.217-222, August 2002.

[5] L. Brakmo, and L. Peterson, "TCP Vegas: End to end congestion avoidance on a global Internet," IEEE J. Sel. Area Communications, vol.13, no.8, pp.1465-1480, October 1995.

[6] J. Mo, "Analysis and Comparison of TCP Reno and Vegas,"INFOCOM, pp.1556-1563, March 1999.

[7] J. Aweya and M. Ouellette, "Enhancing network performance with TCP rate control,"GLOBECOM, pp.1712-1718, December 2000.

[8] S. Floyd, "An Extension to the Selective Acknowledgement(SACK) Option for TCP," RFC 2883, July 2000.

[9] K. Ramakrishnan and S. Floyd, "A Proposal to add Explicit Congestion Notification(ECN) to IP," RFC 2481, January 1999.

[10]F. Kevin and S. Floyd, "Simulation-based Comparisons of Tahoe, Reno, and SACK TCP," Computer Communication Review, October 1996.

[11] B. Hajek, "A queue with periodic arrivals and constant service rate," in Probability, Statistics and Optimization a Tribute to Peter Whittle (F.P. Kelly ed., John Wiley and Sons), pp.166-175, 1994.

[12] R. Jain, "A Delayed-Based Congestion Control Scheme for Window Flow-Networks," DEC TR 566, April 1989.

[13] S. Low, L. Peterson, "Understanding TCP Vegas: Theory and Practice," preprint, http://www.- ee.mu.oz.au/staf/slow/research, February 2000.

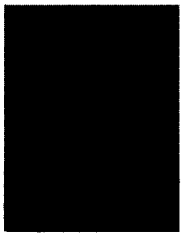
[14] W. Stallings, "High-Speed Networks (TCP/IP and ATM design principles)," Computer and Data CommunicationsTechnology, pp.149-172, January 1998.

[15] 김종권, "TCP Vegas RTT Ambiguity 문제와 그

해결 TCP Vegas RTT Ambiguity Problem and Its Solutions," 한국통신학회지, vol.25, no.7, pp.1260-1269, 2001년 9월.
[16] L. Kleinrock, "Queueing Systems," John Wiley & Sons, pp.312-325, 1975.
[17] http://www.isi.edu/nsnam/ns

저 자 소 개

유 영 일(학생회원)
2002년 아주대학교 전자공학과 학사
2003년~현재 아주대학교 대학원 전자공학과 석사과정
<주관심분야 : TCP, Internet Qos, Traffic Engineering>



이 채 우(정회원)
1985년 서울대학교 제어계측 학사
1988년 한국과학기술원 전자공학과 석사
1995년 University of Iowa 박사
1985년 (주)금성통신 연구원
1988년~1999년 한국통신 선임연구원
1999년~2001년 Lucent Technologies Korea 이사
2001년~2002년 한양대학교 겸임교수
2002년~현재 아주대학교 전자공학과 조교수
<주관심분야 : 광대역 통신망, Ubiquitous networking, Traffic Engineering>