TFRC 프로토콜의 평균 손실 구간 계산방식의 비교평가

(A Comparative Estimation of Performance of Average Loss Interval Calculation Method in TCP-Friendly Congestion Control Protocol)

이 상 철 † 장 주 욱 ††
(Sang-Chul Lee) (Ju-Wook Jang)

요 약 TCP-Friendly 알고리즘중 하나인 TFRC 혼잡제어 프로토콜의 패킷손실 평가를 개선한 새로운 평가 알고리즘을 제시한다. 기존 TCP-Friendly 알고리즘은 단일 패킷 손실에 지나치게 민감하게 반응하여, 전송률의 변화가 심하다. 따라서 일정한 대역폭이 요구되는 멀티미디어 서비스에 적합하지 않다. 제안된 TFRC에서는 다수의 패킷 손실을 취합 계산하여 보다 완만한 전송률과 TCP 프로토콜과의 공정성(Fairness)을 제공한다. 본 논문에서 제안한 지수평활법이 기존의 가중평활법 보다 평활성(smoothness)과 공정성(fairness)의 측면에서 향상됨을 보인다.

키워드 : 정보통신, 인터넷, 프로토콜, TFRC, UDP, Fairness

Abstract We propose a new estimation method for rate adjustment in the face of a packet loss in the TFRC protocol, a TCP-Friendly congestion control protocol for UDP flows. Previous methods respond in a sensitive way to a single packet loss, resulting in oscillatory transmission behavior. This is an undesirable for multimedia services demanding constant bandwidth. The proposed TFRC provides more smooth and fair (against TCP flows) transmission through collective response based on multiple packets loss events. We show our "Exponential smoothing method" performs better than known "Weight smoothing method" in terms of smoothness and fairness.

Key words : Information Technology, Internet, Protocol, TFRC, UDP, Fairness

1. 서 론

온라인 게임이나 멀티미디어 전송, 멀티캐스트 등은 UDP기반으로 혼잡제어 기능 없이 FTP나 SMTP 같은 TCP 기반의 어플리케이션과 더불어 인터넷 대역폭을 공유하고 있다. UDP 기반의 멀티미디어 응용들의 전송률이 높아지면서 TCP기반의 어플리케이션과의 공정성이 크게 문제가 되고 있다. 이제까지 제안된 UDP 기반의 기존 TCP Friendly 알고리즘은 하나의 패킷에 민감하게 반응하여 그 변화가 심하기 때문에, 연속적인 재생이 요구되는 응용프로그램에는 적합하지 않았다. 이에 TCP Friendly Rate Control, 일명 TFRC 혼잡제어 프로토콜이 제안되었다[1]. TFRC는 Reno TCP 모델링 방정식을 혼잡제어 식에 이용하여, 다수의 패킷을 취합 계산한다. 그 결과, 전송률 변동을 적게 하며 TCP와 공정성을 높인다. 이 혼잡제어 방정식에서 패킷손실률(packet loss rate) 평가 및 결정이 가장 중요하다. 가중평활법은 특정시간 구간에서 절반에는 같은 가중치를 나머지 절반에는 더 작은 가중치를 부여하는 방법인데, 전송률의 변동은 적으나 갑작스런 네트워의 변화에 적응이 어렵고 패킷 손실률의 급격한 변화를 주기 어렵다. 지수평활법은 현 시간에 가까운 데이타일수록 가치가 있는 것으로 간주하고 가중치를 부여한 방법이다. 본 연구에서는 지수평활법을 이용하여 급격하게 변하는 네트워크에서 현재 상황에 더 적합한 전송률을 구하는 것을 보인다.

· 본 연구는 한국전산원의 지원을 받았음.
† 비 회 원 : 한국통신 서비스개발연구소 연구원
ideal@kt.co.kr
†† 종신회원 : 서강대학교 전자공학과 교수
jjang@sogang.ac.kr
논문접수 : 2001년 1월 8일
심사완료 : 2002년 6월 12일

2. TCP-Friendly Rate Control(TFRC) 프로토콜 소개 및 제안

TFRC는 UDP에 기반 하여 손실률을 최소화하면서 실시간 데이타를 전송하기 위해 제안되었다[1][2]. UDP에는 없는 혼잡제어 기능을 구현하기 위해 네트워크 상태를 측정하는 방정식을 이용한다. 방정식 기반의 혼잡제어의 목적은 이용 가능한 대역폭을 발견 이용하는 것이 아니라, 상대적으로 안정된 상태를 유지하면서 혼잡상태에 적응하는 것이다. 이러한 목적을 위해 현재 다수를 차지하는 Reno TCP의 모델링 방정식을 혼잡제어알고리즘으로 이용한다[2].

$$T=\frac{s}{RTT\sqrt{\frac{2p}{3}}+RTO*\min\left(1,3\sqrt{\frac{3p}{8}}\right)p(1+32p^2)} \quad (1)$$

s : 패킷 크기, RTT : 왕복 시간, RTO : 재전송 timeout, p : 안정상태의 손실사건발생률(steady-state loss event rate)

TCP와 공정성을 유지하며 안정된 전송을 위해서는 다음과 같은 특성이 요구된다.

① 공격적으로 이용 가능한 대역폭을 찾지 않는다. 대신 손실 이벤트 율(loss event rate)에 맞춰 전송률을 서서히 증가, 감소시킨다.

② 하나의 손실 이벤트(loss event)에 전송률을 반으로 줄이지 않는다. 그러나 몇 개의 연속적인 손실 이벤트에는 전송률을 반으로 줄인다.

③ 수신단은 임의 구간에서 패킷을 하나라도 받으면 RTT마다 적어도 한번의 피드백(feedback)을 전송하여야 한다.

④ 송신단은 몇 번의 RTT이후에도 피드백을 받지 않으면 전송률을 감소시키고 궁극적으로는 전송을 멈춘다.

2.1 프로토콜 구성

TCP 모델링 방정식을 혼잡제어로 사용하기 위해서는 다음과 같은 사항이 요구된다.

① 파라미터 p가 결정되어야 한다. 손실 이벤트 율, p는 수신단에서 결정하고 RTT은 송신단과 수신단에서 측정되어야 한다.

② 수신단은 가장 최근에 받은 패킷의 순서번호(sequence number), 받은 시각, 파라미터 p나 전송률, T를 송신단에 피드백 정보를 전송하여야 한다.

③ 송신단은 이 피드백정보로 RTT와 타임아웃 값, T_0와 손실 이벤트 율, p를 얻어 혼잡제어 방정식에 적용하여 TCP- Friendly 전송률, T을 계산한다. 현재 전송률이 T보다 낮으면 증가시키고 높으면 감소시킨다.

2.1.1 송신단 기능

송신단이 보내는 패킷에는 순서번호(sequence number)와 보낸 시각이 담겨져 있다. 이 순서번호(sequence number)는 연속적으로 증가한다. 순서번호 i 패킷의 시간도장(time stamp)을 t_i^s 라 하면 송신단은 수신단에 하나의 패킷을 보내기 시작한다. 순서번호를 1이라 하면 수신단은 이 패킷을 받자마자 송신단에 피드백 정보를 보낸다. 만약에 최초의 패킷에 대한 피드백을 받지 못하면 패킷이 성공적으로 ack가 될 때까지 혹은 일정 횟수만큼 매 초마다 패킷을 하나씩 보낸다. 이후 송신단은 송신을 멈춘다. 만약 송신단이 시각 t에 피드백을 받으면 RTT를 $t-t_1^s$로 세팅한다. 이것은 초기 RTT이다. 초기 RTT가 결정된 후 송신단은 RTT당 한 패킷으로 전송률을 맞추고 슬로우 스타트로 들어간다. 순서번호(sequence number)가 1 이상인 모든 패킷에 대해서 현재 RTT정보를 담는다. RTT_i를 i 번째 패킷의 RTT로 가정하기로 하자. 수신단은 송신단에 피드백을 보내는데, 이 피드백 패킷은 피드백 타이머가 조절한다. 수신단이 i 번째 패킷을 받고 피드백 타이머가 정지해 있으면 RTT_i후에 피드백 타이머는 리셋(reset)되고 피드백이 송신단에 전달된다. 슬로우 스타트 기간동안 매 피드백 패킷에 대해 두 배씩 전송률을 증가시킨다. 이 슬로우 스타트는 손실 이벤트 율, p 가 "0" 보다 커지면 멈춘다.

현재 시간을 t라하고, 패킷 i를 수신단에서 받은 시각을 t_i^r 라 하면 t_i^d는 다음과 같이 정의된다.

$$t_i^d = t - t_i^r \quad (2)$$

수신단은 피드백 정보에 t_i^s 와 t_i^d를 담는다. 이 피드백이 전송된 이후 RTT_s에 소멸되도록 피드백 타이머를 세팅한다.

t'를 송신단에 도착한 시각이라 가정하면 송신단은 RTT_s를 다음과 같이 계산한다.

$$RTT_s = t' - t_i^s - t_i^d \quad (3)$$

그리고 송신단은 다음 식으로 RTT를 갱신한다.

$$RTT = \alpha*RTT + (1-\alpha)*RTT_s \quad (4)$$

α가 작으면 RTT가 패킷지연에 빠르게 응답하고 반대로 크면 점진적으로 변화할 것이다. 송신단은 또한 TCP와 같이 RTT_s를 이용하여 T_0를 갱신한다. 매번 피드백이 돌아올 때마다 p, RTT, T_0를 이용하여 최근 전송률 T_n를 계산하여 현재 전송률 T가 T_n보다 작으면 다음 식으로 전송률을 높인다.

$$T = \min(T + 1/RTT, T_n) \quad (5)$$

만약 T가 T_n보다 작으면 T_n으로 전송률을 감소시킨다[4].

2.1.2 슬로우 스타트(Slowstart)

초기 전송률 기반 슬로우 스타트(rate based slow-start) 는 윈도우 기반 슬로우 스타트(window based slowstart)와 같이 각 RTT에 전송률을 두 배씩 증가시킨다. TCP는 매 ack을 받을 때마다 윈도우 크기를 하나씩 증가시킨다. 그러므로 한 ack에 두 패킷이 전송되지 않는다. 따라서 TCP는 병목(bottleneck) 대역폭의 두 배 이상 증가할 수 없다. 그러나 전송률 기반 프로토콜은 이러한 자체 특성이 없기 때문에 병목 대역폭의 두 배 이상 증가할 위험이 있다. 이러한 오버슈트(overshoot)를 방지하기 위한 기본 방법으로 수신단이 지난 측정된 RTT동안 받은 패킷 수신률을 송신단에 피드백하는 것이다. 만약 송신단이 슬로우 스타트 모드에 있고 수신단으로부터 피드백정보를 받았는데, 패킷손실률이 "0"보다 크면 슬로우 스타트 모드를 벗어난다. 슬로우 스타트 모드에서는 송신단은 송신률을 다음과 같이 유지한다.

$$T = 2*\min(T, R_{rcvd}) \quad (6)$$

(T : 현재의 전송률, R_{rcvd} : 수신단이 피드백한 수신률.)

2.1.3 수신단 기능

수신단은 송신단이 RTT를 계산하기 위한 피드백정보를 보낸다. 또한 손실 이벤트 율, p를 계산하고 송신단에 보낸다. 손실 이벤트 율은 TFRC에서 가장 핵심이 되는 부분이다. 먼저 손실 이벤트(loss event)를 정의하자. TCP Reno의 경우 송신단이 w 혼잡 윈도우 크기로 전송하고 있으면 아직 ack를 받지 않은 w개의 패킷이 네트웍 상에 있다. i를 네트웍 상에 있는 최초의 패킷이라면 $i+w$-1번째가 마지막 패킷일 것이다. 수신단이 3 개의 duplicate ack를 패킷 j, $i \le j < i+w-1$ 에 대해 받았다면 송신단은 j 패킷이 잃어버렸다고 생각하여 혼잡 윈도우를 반으로 줄인다. 또 다른 패킷 k, $j < k \le i+w-1$ 가 분실되었다면 송신단에서는 이에 반응하지 않는다. 이러한 무반응은 똑같은 혼잡상태로 여기기 때문이다. 이렇게 한 혼잡 윈도우 내 다수의 패킷 분실을 하나의 손실 이벤트로 여긴다.

TFRC는 TCP와 달리 혼잡 윈도우가 없다. 여기에 위 개념을 도입하면 다음과 같다. TFRC 수신단이 이미 l 손실 이벤트 가 있고, 마지막 손실 이벤트가 i 에서 끝났다고 가정한다. 또한 l 번째 손실 이벤트 후 m개의 패킷을 받았고 현재까지 제일 높은 순서번호(sequence number)가 $i+n$ 이하로 가정하면 송신단은 적어도 n 개의 패킷을 전송했고 수신단은 m 개의 패킷을 받은 것이다. j 를 다음을 만족하는 가장 작은 순서번호라 하고

$$i < j < i+n \ , \ j \in n-m \quad (7)$$

k 를 다음을 만족하는 가장 작은 순서번호라 하자.

$$j < k \le i+n \ , \ k \in m \ , \ t_k^r \ge t_{j-1}^r + RTT_{j-1} \quad (8)$$

여기서 $j-1$ 과 k 사이의 모든 분실 패킷은 하나의 손실 이벤트로 본다[4].

손실 이벤트 횟수의 계산에는 다음 같은 원칙이 있다.

① 손실률 계산치는 하나의 패킷손실을 측정하는 것이 아니라 손실 이벤트 율을 측정하는 것으로, 손실 이벤트 율은 한 RTT내 몇 개의 패킷 손실로 구성된다.

② 손실률 계산치는 새로운 손실 이벤트에만 반영되어야 한다.

③ 이전의 계산평균보다 큰 새로운 손실구간(loss interval) 혹은 지난 손실 이벤트보다 충분히 큰 구간에 손실률 계산치는 감소하여야 한다.

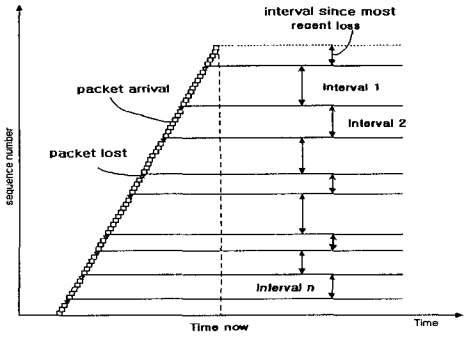


그림 1 손실구간

그림 1은 손실구간을 나타내고 있다. s_i를 i번째 최근의 손실구간의 패킷 개수라 하고, s_0는 지난 손실이후 도착하는 패킷이라 하자. 하나의 손실이 발생하면 s_0는 s_1이 되고, 그 이후의 해당 손실구간은 하나씩 이동하게 되며 s_0는 0이 된다. s_0가 아직 손실이 발생되지 않은 점에서 다른 손실구간과 구분이 된다. 평균 손실구간 계산을 위해 다음의 알고리즘을 적용한다..

i) 가중평활법(Weight smoothing method)

n개의 구간중 가장 최근 $\frac{n}{2}$개 구간에는 똑같이 "1"씩 가중치를 부여하고 그 이전의 $\frac{n}{2}$개 구간에는 더 작게 가중치를 부여한다. 평균 손실구간, s^A는 다음과 같이 계산한다.

$$s^A = \frac{\sum_{i=1}^{n} w_i s_i}{\sum_{i=1}^{n} w_i} \quad (9)$$

여기서 $w_i = 1, 1 \le i \le \frac{n}{2}$ 이고 $w_i = 1 - \frac{i - \frac{n}{2}}{\frac{n}{2}+1}$, $\frac{n}{2} < i \le n$ 이다. 예를 들어 $n=8$이면 $w_1, w_2, w_3, w_4 = 1; w_5 = 0.8, w_6 = 0.6, w_7 = 0.4, w_8 = 0.2$이다.

s_0부터 s_{n-1}까지 평균 손실구간은 다음 (10)과 같다.

$$s_{new}^{A} = \frac{\sum_{i=0}^{n-1} w_{i+1} s_i}{\sum_{i=1}^{n} w_i} \quad (10)$$

최종적인 평균 손실구간은 $\max(s^A, s_{new}^A)$으로 구한다.

그러나 가중평활법은 최근 4개의 손실구간에 똑같은 가중치를 부여함으로서 전송률의 변동은 적으나 갑작스런 네트워의 변화에 적응이 어렵고 패킷 손실률의 급격한 변화를 주기 어렵다. 또한 최근의 네트워 상황은 s_0, s_1 에 가장 잘 반영되기 때문에 s_0, s_1의 직접적인 변화를 주는 것이 능동적이고, 네트워 상황에 적절히 가변적인 전송률을 주는 것이 용이하다. 이에 통계학의 시계열분석에서 사용하는 지수평활이론을 변형 적용한 다음의 지수평활법을 제안한다[3].

ii) 지수평활법(Exponential smoothing method)

지수평활법이란 예측수법의 하나이며, 이 방법은 현시점에 가까운 데이타일수록 가치가 있는 것으로 지수형에 무게를 가중하는 것이 특징이고 지수형 가중이동 평균법이라고도 불린다

n개의 구간중 가장 최근 $n-1$개 구간에는 산술평균를 계산하여 평활값, $(1-\alpha)$값을 곱하고 s_1에는 α 평활값을 곱한다. 평균 손실구간, s^A는 다음과 같이 계산한다.

$$s^A = \alpha * s_1 + (1-\alpha)\frac{\sum_{i=2}^{n} s_i}{n-1}, \quad 0 \le \alpha \le 1 \quad (11)$$

s_0부터 s_{n-1}까지 평균 손실구간은 다음 (12)과 같다.

$$s_{new}^{A} = \alpha * s_0 + (1-\alpha)\frac{\sum_{i=1}^{n-1} s_i}{n-1}, \quad 0 \le \alpha \le 1 \quad (12)$$

최종적인 평균 손실구간 $s^{\wedge}$은 $\max(s^A, s_{new}^A)$으로 구한다.

그리고 $s^{\wedge}$는 최종적으로 손실사건률 p는 $1/s^{\wedge}$로 한다.

이렇게 지수평활법으로 구한 평균 손실구간을 손실사건률에 반영하면 가중평활법을 사용했을 때보다 급격하게 변하는 네트워크에서 현재 상황에 더 적합한 전송률을 구할 수 있다.

3. 시뮬레이션 및 시뮬레이션 결과

3.1 시뮬레이션 환경

simulator는 버클리 대학의 ns(network simulator)로

그림 2 시뮬레이션 구성

하였고 TCP 64개, TFRC 64개씩 송수신 노드를 배치하였여 링크 대역폭(link bandwidth) 15M/BPS RED Queue로 구성하였고, 손실구간의 개수 n=8로 하였다. 시뮬레이션에서 사용한 사양은 Redhat Linux 6.0, CPU pentium 350 MHz, Memory 128 Mbytes로 하였다[4].

TFRC와 혼잡구간에서 경쟁하는 트래픽으로 Reno TCP를 사용하여, 혼잡구간의 네트워크 대역폭을 공정하게 나누는가에 대한 실험을 하였다. 그리고 이 Reno TCP의 알고리즘을 사용한 TFRC와, 가까운 미래에 많이 사용될 것으로 예상되는 Sack TCP[5]와의 공정성도 비교하여, 이 알고리즘이 병목구간에서 비교적 다양한 조건에 잘 적응할 수 있음을 보였다.

3.2 시뮬레이션 결과

TCP와 TFRC사이의 공정성을 보기 위해 Reno TCP와 Sack TCP로 나누어, 지수평활지수(Exponential smoothing factor), α의 변화에 따른 링크 노드간 TCP와 TFRC 전송속도(throughput)를 나타내 보았다. 지수평활법의 α를 결정하기 위해 15초 이후의 각 α의 변화에 따른 TCP 전송속도 평균값과 TFRC의 전송속도 평균값 차이를 계산하여 링크 대역폭과 백분율 하여 그림 3과 같은 결과를 얻었다.

Reno TCP 경우 α=0.3 부근에서 Sack TCP 경우 α=0.37 부근에서 최저를 보이고 있다.

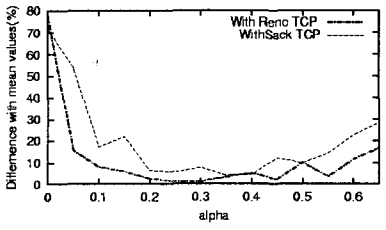
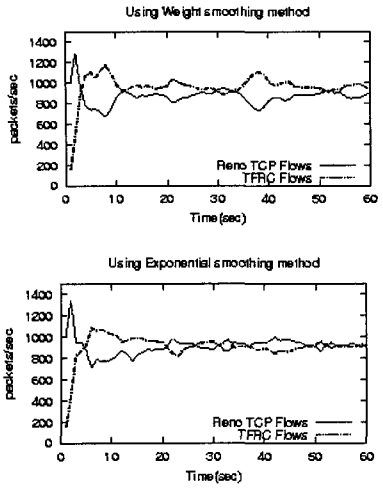


그림 3 α값의 변화에 따른 전송속도 평균값 차이 변화

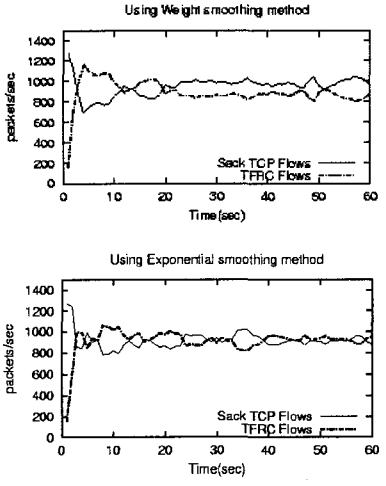
3.2.1 공정성 비교

Reno, Sack TCP와의 가중평활법, 지수평활법의 전송속도 평균값 차이 최저 점에서 전송속도를 비교해 보면 다음과 같다.



when α=0.3

그림 4 Reno TCP와 각 평활법 전송속도 공정성



when α=0.37

그림 5 Sack TCP와 각 평활법 전송속도 공정성

위 그림의 가로축은 시간이고 세로 축은 전송속도이다. 영문 "Using weight smoothing method"가 기존 가중평활법의 경우이고 "Using Exponential smoothing method"가 지수평활법의 경우이다. 그림 4의 가중평활법 보다 지수평활법이 정상상태에서 시간이 지남에 따라 TCP 흐름(flow)와 공평하게 수렴함을 보여주고 있다. 또한 그림 5의 Sack TCP와의 경우도 비슷한 수렴성을 보이고 있다.

3.2.2 등가(Equivalence) 비교

먼저 시간 t에 flow의 전송률을 정의하자.

$$R_{\delta,F}(t)=\frac{s*(t \text{ 와 } t+\delta \text{ 사이의 } F\text{에 의해 보내진 패킷 수})}{\delta}$$

s : 패킷 크기(bytes). δ : 시간크기(time scale).

F : 흐름(flow)

여기서 시간크기(time scale)에 따른 전송률을 비교하기 위해 등가를 정의하면 다음과 같다.

$$e_{\delta,a,b}(t)=\min\left(\frac{R_{\delta,a}(t)}{R_{\delta,b}(t)},\frac{R_{\delta,b}(t)}{R_{\delta,a}(t)}\right), R_{\delta,a}(t)>0 \text{ or } R_{\delta,a}(t)>0$$

a, b flows의 최소 값을 취하면 "0"과 "1"사이 유지한다. 등가가 "1"에 가까울수록 두 흐름(flows)은 비슷한 전송속도 갖는다.

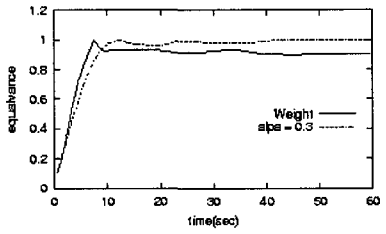


그림 6 Reno TCP와의 등가

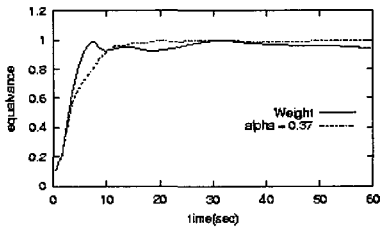


그림 7 Sack TCP와의 등가

위의 그림 6은 시간크기를 1초로 취할 때 각각 Reno TCP와 가중평활법, Reno TCP와 지수평활법의 α값이 0.3일 때 등가를 나타낸 것이고 그림 7은 Sack TCP와 가중평활법, Sack TCP와 지수평활법의 α값이 0.37일 때 등가를 나타낸 것이다.

15초 이후 정상 상태의 각각의 평균 등가를 나타내면 표 1과 같다.

표 1 등가 비교

평활법 \ TCP	Reno TCP	Sack TCP
가중평활법	0.91	0.94
지수평활법 α=0.3/0.37	0.97	0.98

지수평활법의 등가차이를 보면 Reno TCP와 지수평활법의 α=0.3일 때 기존 가중평활법 대비 약 6%정도 향상됨을 알 수 있다. 마찬가지로 Sack TCP와의 경우 α=0.37일 때 기존 가중평활법 대비 약 4%의 향상이 있었다.

4. 결 론

본 논문에서는 Reno TCP 모델을 기반으로 혼잡제어 방정식으로 이용한 TFRC(TCP Friendly Rate Control) 혼잡제어 프로토콜의 패킷 손실률 평가 알고리즘으로 지수평활법을 제안하여 기존 가중평활법 사이의 공정성, 전송률 증감 등의 성능을 비교 분석하였다.

기존 가중평활법은 최근 4개의 손실구간에 똑같은 가중치를 부여하여 전송률의 변동은 적으나 급격한 네트웍의 변화에 적응이 어렵고 손실 사건률의 적합한 변화를 주기 어렵다. 또한 최근의 네트웍 상황은 손실구간 s_0, s_1에 가장 잘 반영되기 때문에 s_0, s_1의 직접적인 변화를 주는 것이 능동적이고, 네트웍 상황에 적절히 가변적인 전송률이 주는 것이 용이하다.

이에 통계학의 시계열분석에서 사용하는 지수평활 이론을 변형 적용한 지수평활법을 제안, Berkeley ns(network simulator)를 이용 각각 64개 같은 수의 TCP, TFRC 상황에서 시뮬레이션을 하였다.

그 결과 현재 인터넷 데이타 전송의 대부분을 차지하는 TCP와의 공정성 측면에서 Reno TCP와 지수평활법의 α=0.3일 때 가중평활법 대비 등가가 약 6%정도 향상되었다. 마찬가지로 Sack TCP 경우 α=0.37일 때 등가가 약 4%의 향상이 있었다.

α 값은 유동적인 값으로 전송속도 평균값을 매번 산출하여 이에 따라 바꾸어주면 네트웍 상황에 맞는 전송률을 얻을 수 있다. TFRC는 UDP를 이용하여 전송하는 알고리즘이므로, 전송 rate를 응용 프로그램이 정할 수 있기 때문에 매번 보정해 줄 수 있다.

참 고 문 헌

[1] Sally Floyd, Mark Handley, Jorg Widmer, Jitendra Padhye. "Equation-Based Congestion Control for Unicast Application," Proceeding of ACM SIGCOMM'00, pp.43-56, August 2000.

[2] S. Floyd, Jitendra Padhye, Mark Handley "TCP Friendly Rate Control(TFRC) : Protocol Specification," IETF Internet-draft, 20 July 2001. URL http://www.aciri.org/tfrc/draft-ietf-tsvwg-tfrc-02.txt

[3] 김제영, 배현웅, 윤병창. 확률 및 통계, 청문각 1999.

[4] Kevin Fall & Kannan Varadhan. ns Notes and Documentation UC Berkeley. December 1999. URL http://www.isi.edu/nsnam/ns/ns-documentation.html

[5] S. Floyd. "Issues of TCP with SACK," Technical report, Mar. 1996. URL ftp://ftp.ee.lbl.gov/papers/issues_sa.ps.Z.

[6] V.Jacobson. "Modified TCP Congestion Avoidance Algorithm," Technical reprort, end2end-interest mailing list, 30 Apr.1990. URL ftp://ftp.ee.lbl.gov/email/vanj.90apr30.txt.

[7] Kevin Fall and Sally Floyd "Comparisons of Tahoe, Reno, and Sack TCP," http://www.aciri.org/floyd/papers.html 1996.

[8] W. Richard Stevens. TCP/IP Illustrated, Volume I, The Protocols. Addison Wesley, 1994.

[9] Jitendra D.Padhye. "Towards A Comprehensive Congestion Control Framework For Continuous Media Flows in Best Effort Networks," PhD thesis ,University of Massachusetts Amherst 2000.

이 상 철

1995년 숭실대학교 전자공학과 학사. 2001년 서강대학교 대학원 전자공학과 석사. 현재 한국통신 서비스개발연구소 음성인식 및 응용연구. 관심분야는 VoIP, 음성인식

장 주 욱

1983년 서울대학교 전자공학과 학사. 1985년 한국과학기술원 전기 및 전자공학과 석사. 1993년 University of Southern California 박사. 1995년 ~ 현재 서강대학교 전자공학과 부교수. 관심분야는 인터넷 프로토콜, 병렬 처리