임의 패킷 손실에 대한 TCP의 손실 복구 과정 모델링 및 분석

정회원 김 범 준*, 김 동 연**, 이 재 용*

Modeling TCP Loss Recovery for Random Packet Losses

Beom-Joon Kim*, Dong-Yeon Kim**, Jai-Yong Lee* *Regular Members*

요 약

TCP Reno는 하나의 윈도우 내에서 다수 개의 패킷 손실이 발생하는 경우 손실된 패킷들을 효율적으로 복구하는 것이 불가능한 문제점을 가지고 있다. 이 문제점을 개선하기 위해서 설계된 TCP New-Reno는 부분 승인 패킷(partial acknowledgement)을 통해 fast recovery를 연장함으로써 다수 개의 패킷 손실이 발생하더라도 이들을 재전송에 의해 복구하는 것이 가능하다. 그러나 TCP New-Reno 역시 재전송 패킷들이 다시 손실되는 경우 불가피한 RTO (Retransmission Timeout)가 발생한다는 문제점을 가지고 있다. 이런 문제점을 개선하기 위해서 중복 승인 패킷 수를 근거로 재전송 패킷 손실을 감지할 수 있는 DAC (Duplicate Acknowledgement Counting) 알고리듬을 제안한다. TCP Reno, TCP New-Reno 그리고 DAC를 사용하는 경우에 대해 손실 복구 과정을 정확하게 모델링하고 손실된 패킷이 복구되기 위한 조건들을 유도한다. 임의 패킷 손실 확률에 대한 손실 복구 확률을 수학적으로 계산하고 이를 통해 DAC가 TCP New-Reno의 손실 복구 기능을 향상시킬 수 있다는 것을 보인다.

키워드: TCP 혼잡 제어, TCP 손실 복구, TCP 모델링, DAC

ABSTRACT

The fast retransmit and fast recovery algorithm of TCP Reno, when multiple packets in the same window are lost, cannot recover them without RTO (Retransmission Timeout). TCP New-Reno can recover multiple lost packets by extending fast recovery using partial acknowledgement. If the retransmitted packet is lost again during fast recovery, however, RTO cannot be avoided. In this paper, we propose an algorithm called "Duplicate Acknowledgement Counting (DAC)" to alleviate this problem. DAC can detect the retransmitted packet loss by counting duplicate ACKs. Conditions that a lost packet can be recovered by loss recovery of TCP Reno, TCP New-Reno and TCP New-Reno using DAC are derived by modeling loss recovery behavior of each TCP. We calculate the loss recovery probability for random packet loss probability numerically, and show that DAC can improve loss recovery behavior of TCP New-Reno.

I. 서 론

현재 인터넷의 수송계층 프로토콜로써 널리 사용되고 있는 TCP는 신뢰성이 높은 전송 매체 링크에서 동작한다는 가정 하에 설계되었으므로 모든 패킷 손실을 혼잡 (congestion)이 발생한 것으로 간주한다. 따라서 무선 링크와 같이 혼잡뿐만 아니라 다른 원인에 의한 패킷 손실이 발생하는 경우 TCP의 성능이 크게 저하되는 문제점이 있다[1,12]. 이러한 문제점의 원인으로는 잦은 혼잡 제어

* 연세대학교 전기전자공학과 네크워크 연구실({bjkim, jyl}@nasla.yonsei.ac.kr),
** 국립한경대학교 전자공학과(dykim@hnu.hankyong.ac.kr), 논문번호 : 020510-1129, 접수일자 : 2002년 12월 8일
※본 연구는 한국과학재단 특정기초연구(R01-2002-000-00531-0)지원으로 수행되었음.

로 인한 윈도우 크기의 감소와 손실된 패킷을 재전송에 의해 복구할 수 없는 경우 빈번하게 발생하는 RTO (Retransmission Timeout)로 설명할 수 있다. 특히, RTO가 발생하면 일정 시간 동안 송신원은 패킷을 전송할 수 없을 뿐만 아니라 이후에는 다시 slow start상태에서 윈도우를 증가시켜야 하므로 RTO의 발생 여부는 TCP의 성능에 큰 영향을 준다.

지금까지의 TCP 분석 모델들[11-18]은 손실된 패킷의 복구 과정 (loss recovery)보다는 주로 TCP의 처리율 (throughput)에 초점이 맞추어져 있다. 이미 TCP의 처리율은 손실된 패킷의 복구 확률과 밀접한 관계가 있다는 점이 입증되었다[13]. 따라서 본 논문에서 TCP의 손실 복구 과정에 중점을 두고 TCP Reno와 TCP New-Reno의 패킷 손실 복구 과정을 분석한다. 추가적으로 TCP New-Reno의 손실 복구 과정이 재전송 패킷이 다시 손실되는 경우 RTO를 피할 수 없는 문제점이 있다는 점을 보이고 이 문제를 개선하기 위한 알고리듬인 Duplicate Acknowledgement Counting (DAC)을 제안한다. DAC가 TCP New-Reno의 fast recovery 알고리듬에 추가될 경우 손실된 재전송 패킷을 일부 복구하는 것이 가능하므로 TCP New-Reno보다는 성능 향상을 가져올 수 있다.

II. TCP 손실 복구 과정

이번 장에서는 TCP Reno와 TCP New-Reno의 패킷 손실 복구과정을 간략하게 설명한다. TCP 프로토콜의 자세한 동작에 대해서는 다음과 같은 참조 문헌[2-10]들을 통해서 알 수 있다.

1. TCP Reno[3,4,7,8]

TCP Reno에서 손실된 패킷을 fast retransmit에 의해 복구하기 위해서는 송신원은 손실된 패킷에 대한 적어도 K개의 (일반적으로 $K=3$이다.) 중복(duplicate) Acknowledgement(ACK)를 수신해야 한다. K번째 중복 ACK를 수신한 송신원은 손실된 패킷을 fast retransmit에 의해 재전송 한 후 윈도우의 크기를 이전 혼잡 윈도우의 크기의 반으로 줄인다. 이어지는 fast recovery 동안 중복 ACK가 수신될 때마다 송신원은 윈도우의 크기를 하나 씩 증가시키고 만약 이때 새로이 윈도우에 포함되는 패킷들은 전송된다. 재전송된 패킷에 의해 정상적인 ACK가 도착하면 fast recovery는 종료되고 송신원은 congestion avoidance상태에서 전송을 계속 진행한다. 동일한 윈도우 내에서 여러 개의 패킷이 손실된 경우 손실된 패킷 각각을 fast retransmit에 의해서만 재전송할 수 있다. 즉, 각각의 손실된 패킷에 대해서 적어도 개의 중복 ACK를 수신해야 하기 때문에 이런 경우 손실된 패킷들이 모두 fast retransmit에 의해 복구되기가 힘들어지게 된다.

2. TCP New-Reno[5-8]

한 윈도우 내에서 다수 개의 패킷 손실이 발생했을 때 자주 RTO가 발생하는 TCP Reno의 손실 복구 과정을 개선하기 위해 TCP New-Reno의 패킷 손실 복구 과정은 부분 ACK를 도입하여 TCP Reno의 fast retransmit과 fast recovery 알고리듬을 수정 보완한다. 한 윈도우 내에서 다수 개의 패킷이 손실된 경우 송신원은 첫 번째 재전송한 패킷에 의해 두 번째 손실된 패킷에 대한 ACK를 수신하게 된다. 이는 패킷 손실이 발생하기 이전에 전송한 모든 패킷들의 전송을 승인하지 못하므로 부분 (partial) ACK 라고 한다. TCP Reno와는 달리 TCP New-Reno의 송신원은 부분 ACK를 수신했을 때 fast recovery를 종료하지 않고 해당 패킷을 즉시 재전송하고, 이 때 윈도우의 크기를 패킷 손실이 발생하기 전 혼잡 윈도우 크기의 반으로 설정한다. Fast recovery가 종료될 때까지 중복 ACK가 수신될 때마다 윈도우의 크기를 하나씩 증가시키고 이 과정에서 가용 윈도우 (usable window) 내에 새로이 포함되는 패킷들을 전송하는 점은 TCP Reno와 동일하다. 이와 같은 과정을 통해서 하나의 윈도우 내에서 여러 개의 패킷 손실이 발생했더라도 부분 ACK를 수신하는 한 매 Round Trip Time (RTT)마다 한 패킷씩 복구하는 것이 가능하다. 손실된 마지막 패킷을 재전송한 후 송신원은 정상 ACK를 수신하게 되므로 패킷 손실이 발생하기 전 혼잡 윈도우 크기에서 반으로 줄어든 윈도우로 congestion avoidance 상태에서 패킷 전송을 계속한다. 이와 같이 TCP New-Reno는 여러 개의 패킷 손실이 발생한 경우 부분 ACK를 통해서 fast recovery를 연장함으로써 이들을 복구하는 것이 가능하다.

TCP New-Reno의 패킷 손실 복구 과정에서 재전송되는 패킷들 가운데 패킷 손실이 발생하는 경우 송신원은 재전송 중 손실된 패킷에 대한 중복 ACK만을 계속 수신하게 되므로 fast recovery를

연장하는 것이 불가능하여 불가피하게 RTO가 발생하게 된다. 패킷 손실 확률이 높을수록 하나의 윈도우 내에서 손실되는 패킷의 수가 증가하고 이 패킷들을 재전송하는 과정에서 다시 손실될 확률 또한 증

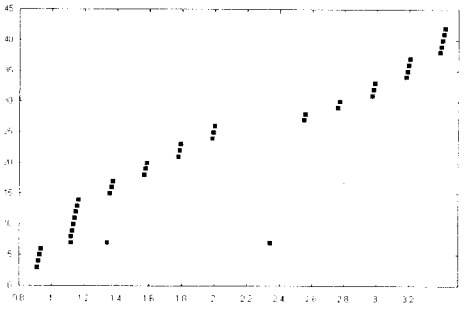


그림 1-(a). 하나의 재전송 패킷이 다시 손실된 경우 TCP New-Reno의 손실 복구 과정

1의 약 1.1초에서 $cwnd=8$인 상태에서 패킷 7~14를 전송한다. 만약 패킷 7이 손실된 경우 약 1.35초에 패킷 7을 fast retransmit을 통해서 재전송한다. 패킷 7에 대한 모든 중복 ACK를 수신했을 때 가용 윈도우의 크기는 11(=[8/2]+7)이 되어 패킷

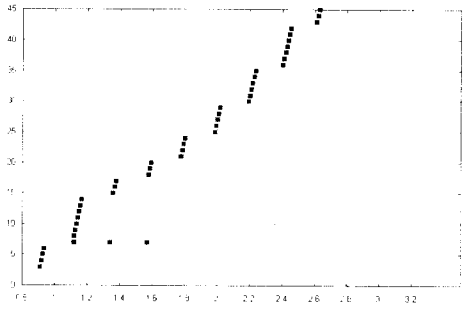


그림 1-(b). 하나의 재전송 패킷이 다시 손실된 경우 DAC의 손실 복구 과정

가하기 때문에 TCP New-Reno의 성능을 상당히 저하시킬 수 있다.

2.3 Duplicate Acknowledgement Counting Algorithm (DAC)

DAC가 동작하기 위해서 TCP 송신원에 몇 가지 변수가 추가되어야 한다. 이 변수들에는 재전송 패킷이 손실되지 않을 경우 이에 대해 발생하게 될 중복 ACK의 수가 저장된다. 만약 어떤 손실된 패킷에 대해 수신된 중복 ACK의 수가 이 값보다 더 큰 경우 재전송한 패킷은 다시 손실되었음을 알 수 있게 된다. 동일한 윈도우 내에서 손실된 패킷의 수가 n 개 인 경우 첫 번째 손실된 패킷에 대한 K 째 중복 ACK를 수신했을 때 송신원은 패킷 손실이 발생하기 전 혼잡 윈도우의 크기 (s_cwnd)와 손실된 패킷에 대해 발생하게 될 중복 ACK의 수 (DAC_1)를 계산해서 저장한다. 손실 복구 과정에서 재전송한 i번째 패킷에 대해 발생이 예상되는 중복 ACK의 수를 DAC_i로 정의할 때 DAC_1의 값은 s_cwnd -1로 설정된다. 왜냐하면 이 때 송신원은 s_cwnd 개의 패킷들 가운데 적어도 하나의 패킷 손실되었다는 사실만을 확실하게 알 수 있기 때문이다.

그림 1은 한 패킷이 손실된 경우 재전송한 패킷이 다시 손실되었을 때 TCP New-Reno와 DAC를 사용하는 경우 손실 복구 과정을 보여준다. 그림

15~17을 전송한다. ($[a]$는 a의 정수 부분을 의미한다.) 만약 재전송한 패킷 7이 다시 손실된 경우 패킷 15~17에 의해서 다시 패킷 7에 대한 중복 ACK를 수신하게 된다. 패킷 15~17에 의해서 가용 윈도우의 크기는 다시 세 개의 패킷 크기만큼 증가하기 때문에 다시 패킷 18~20을 전송한다. 그러나 정상적인 ACK를 수신하는 것은 불가능하기 때문에 fast recovery를 정상적으로 종료되지 못하고 결국은 RTO가 발생한다. 반면 DAC를 사용하는 경우 패킷 7을 재전송하고 $DAC_1=7(=8-1)$로 설정한다. 패킷 15에 의해서 패킷 7에 대한 8번째 중복 ACK를 수신하기 되는데 이는 DAC_1의 값보다 크다. 따라서 송신원은 패킷 7이 다시 손실되었다는 것을 알 수 있고 이를 다신 재전송한다. 다수 개의 패킷들이 동일한 윈도우 내에서 손실된 경우 DAC_i (for $i\geq 2$)의 값은 $(i-1)$번째 손실된 패킷을 재전송한 이 후 새로 전송하는 패킷의 수와 동일하다. 재전송한 $(i-1)$번째 손실된 패킷에 의해서 i번째 손실된 패킷에 대한 부분 ACK가 수신되고 이 후에 전송된 패킷들에 의해서는 i번째 손실된 패킷에 대한 중복 ACK가 수신되기 때문이다. 그림 2는 윈도우의 크기가 8인 상태에서 두 개의 패킷, 패킷 7과 14가 손실되고 두 번째 재전송한 패킷 14가 다시 손실되었을 때 TCP New-Reno와 DAC를 사용하는 경우의 손실 복구 과정을 보여준다. New Reno의 경우 패킷 7을 fast

retransmit한 이 후 패킷 7에 대한 모든 중복 ACK를 수신했을 때 가용 윈도우의 크기는 10(=[8/2]+6)이 되고 새로이 포함된 패킷 15~16이 전송된다. 재전송한 패킷 7은 손실되지 않으므로 이에 의해 패킷 14에 대한 부분 ACK가 발생하고 패킷 14는 재전송된다.

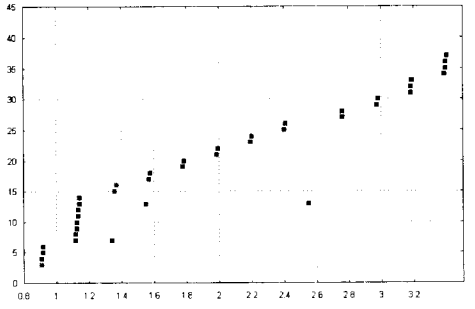


그림 2-(a). 재전송한 두 번째 패킷이 다시 손실된 경우 TCP New-Reno 손실 복구 과정

만약 재전송한 패킷 14가 다시 손실되는 경우 패킷 15와 16에 의해서는 패킷 14에 대한 중복 ACK가 발생한다. 이 후의 매 라운드마다 가용 윈도우의 팽창에 의해 두 개씩의 패킷들을 전송하지만 결국 패킷 14에 의한 RTO를 막을 수는 없다. DAC를 사용하는 경우에는 패킷 1을 재전송한 이 후 전송하는 새로운 패킷의 수는 패킷 15~16, 두 개이므로 $DAC_2=2$로 설정한다. 패킷 14가 다시 손실된 경우 패킷 17에 의해서도 패킷 14에 대한 중복 ACK를 수신하게 되고 이 때 수신된 패킷 14에 대한 중복 ACK의 수는 3 ($>DAC_2$)이므로 패킷 14가 다시 손실된 것을 알 수 있다. 그러므로 DAC를 사용하는 경우 약 1.8초에서 패킷 14에 대한 두 번째 재전송이 이루어지고 다시 손실되지 않는다면 RTO는 발생하지 않는다.

손실된 패킷이 다수 개인 경우에 DAC는 단 한 개의 손실된 패킷에 한 번만 적용이 가능한 것으로 가정한다. 즉, 앞의 예에서 패킷 7과 패킷 14가 모두 재전송 과정에서 손실되는 경우 혹은 DAC에 의해 재전송한 패킷이 다시 손실되는 경우 DAC를 사용하는 경우라고 하더라도 RTO가 발생한다. DAC가 성공적으로 손실된 패킷을 복구할 수 있는가는 그 패킷이 몇 번째 손실된 패킷인지 그리고 윈도우 내 몇 번째 패킷인지에 의해 결정되는데 이에 대해서는 다음 장의 모델링 과정에서 자세하게 설명한다.

Ⅲ. TCP 패킷 손실 복구 과정 모델링

패킷 손실 복구 과정을 분석하기 위해서 몇 가지 가정과 정의가 필요하다. 우선 각 패킷은 임의의

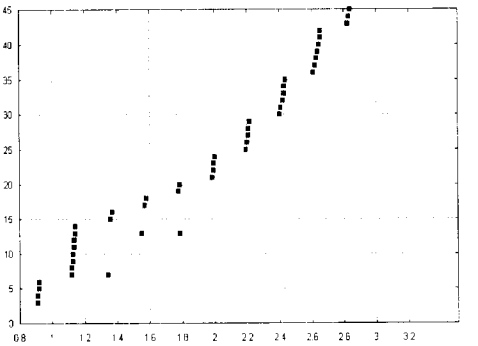


그림 2-(b). 재전송한 두 번째 패킷이 다시 손실된 경우 DAC 손실 복구 과정

(random) 확률 p로 손실되고 패킷 손실들은 서로 독립적인 것으로 가정한다. 송신원은 전송해야 할 패킷들을 충분히 가지고 있어서 항상 윈도우가 허용하는 최대 개수의 패킷들을 전송하는 것으로 가정한다. 수신원은 지연 (delayed) ACK를 사용하지 않으므로 패킷을 수신할 때마다 ACK를 송신원에 전달하는 것으로 가정한다.

패킷 손실이 감지되기 전에 전송된 패킷들에 의한 모든 정상 ACK를 수신했을 때의 윈도우의 크기를 Ω로 나타내고 이를 손실윈도우(loss window)라고 한다[13]. 손실윈도우가 포함하는 첫 번째 패킷은 항상 손실된 패킷이 된다. 패킷 손실에 따른 손실윈도우의 정상 상태 분포를 얻고 패킷 손실 복구 확률을 구하는 과정은 Markov 프로세스를 통한 분석 과정을 도입한다[13]. 패킷 손실 과정은 '라운드' 단위로 모델링 되고[16] Φ_i는 손실된 패킷과 재전송 패킷을 제외한 각 i번째 라운드에서 전송되는 패킷의 수를 나타낸다.

1. TCP Reno

$\Omega=u$ 일 때 손실된 한 개의 패킷이 재전송에 의해 복구될 확률 $R_K^{(1)}$은 $\Phi_1=u-1\geq K$ 을 만족하는 경우에 대해 다음의 식 (1) 로 주어진다.

$$R_R^{(1)} = (1-p)^{\Phi_1}(1-p) = (1-p)^{u} \quad (1)$$

마찬가지로 두 개의 손실된 패킷들이 재전송에 의해 복구될 확률 $R_R^{(2)}$은 $\Phi_2=[u/2]-2\geq K$을 만족하는 경우에 대해 다음의 식 (2) 로 주어진다.

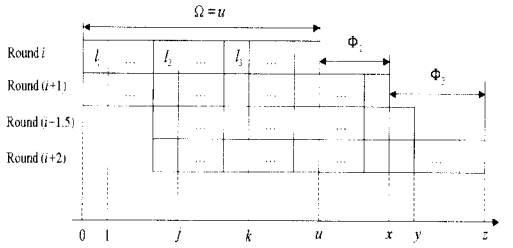


그림 3. 세 개의 패킷 손실에 대한 TCP Reno의 손실 복구 과정 모델

$$R_R^{(2)} = \binom{u-1}{1} p(1-p)^{u-2}(1-p)^{\Phi_2}(1-p)^2 = (u-1)p(1-p)^{u+\Phi_2} \quad (2)$$

손실된 세 개의 패킷들은 $\Phi_3\geq K$ 인 경우 모두 재전송에 의해 복구되는 것이 가능하다. 다음 그림 3은 Φ_3의 값을 구하기 위해서 손실 패킷 복구 과정을 보여준다.

- k 라운드에서 손실 복구 과정이 시작한다.
- $(k+1)$ 라운드에서 첫 번째 손실된 패킷 l_1에 대한 마지막 중복 ACK를 수신한다.
- $(k+1.5)$ 라운드에서 재전송한 l_1에 의해 두 번째 손실된 패킷 l_2에 대한 첫 번째 ACK를 수신한다.
- $(k+2)$ 라운드에서 l_2에 대한 마지막 중복 ACK를 수신한다.

이 때, 각 라운드의 윈도우의 오른쪽 경계인 x, y 그리고 z는 다음의 식 (3)으로 주어진다.

$$\begin{aligned} x &= [u/2]+(u-3) \\ y &= (j-1)+[u/2] \\ z &= (j-1)+[u/4]+\Phi_2,\ \Phi_2=[u/2]-3 \end{aligned} \quad (3)$$

$(k+1.5)$ 라운드에서는 l_2가 재전송되기 전이므로, $y\geq x$ 인 경우 전송되는 $y-x$개의 패킷들에 의해서는 l_2에 대한 중복 ACK가 발생하게 된다. 따라서 Φ_3는 다음의 식 (4)로 주어진다.

$$\Phi_3 = z-\max(x,y) = \begin{cases} (j-1)+[u/4]-u & 2\leq j\leq u-2 \\ [u/4]-3 & j=u-1 \end{cases} \quad (4)$$

$j=u-1$ 인 경우 $(k+2)$ 라운드의 윈도우는 최대 이동하게 되므로, $K=3$이라면 세 개의 손실된 패킷이 모두 재전송에 의해 복구 가능한 윈도우의 최소 값은 $[u/4]-3=3$에 의해 24 라는 것을 알 수 있다. $2\leq j\leq u-2$인 경우 $(j-1)+[u/4]-u\geq K$을 만족하는 경우에 한해 세 개의 손실된 패킷은 모두 재전송에 의해 복구하는 것이 가능하다. 즉, 첫 번째 손실된 패킷과 두 번째 손실된 패킷 사이에 적어도 $u-[u/4]+(K-1)$개의 패킷이 손실되지 않는 경우이므로, 이 때의 복구 확률 $R_R^{(3)}$는 다음의 식 (5)로 주어진다.

$$R_R^{(3)} = \binom{[u/4]-K}{2} p^2(1-p)^{u+\Phi_2+\Phi_3} \quad (5)$$

동일한 윈도우 내에서 네 개 이상의 패킷 손실이 발생하는 경우, 어떤 경우에도 $\Phi_4\geq K$ 를 만족할 수 없기 때문에 항상 RTO가 발생하게 된다. 결과적으로 TCP Reno의 손실 복구 확률은 다음의 식 (6) 로 주어진다.

$$\begin{aligned} R_R &= R_R^{(1)}+R_R^{(2)}+R_R^{(3)} \\ &= (1-p)^{u}\left\{\begin{matrix} 1+(u-1)p(1-p)^{\Phi_2} \\ +\binom{[u/4]-K}{2}p^2(1-p)^{\Phi_2+\Phi_3} \end{matrix}\right\} \end{aligned} \quad (6)$$

2. TCP New-Reno

$\Omega=u$ 일 때 손실된 n개의 패킷들이 모두 재전송에 의해 복구되기 위해서는 첫 번째 손실된 패킷을 fast retransmit에 의해 재전송이 가능해야 하고 fast recovery 동안 재전송되는 n개의 패킷들은 모두 손실되지 않아야 한다. 그러므로 TCP New-Reno의 손실 복구 확률 R_{NR}은 다음의 식 (7)로 주어진다.

$$\begin{aligned} R_{NR} &= \sum_{n=1}^{u-K}\binom{u-1}{n-1}p^{n-1}(1-p)^{u-n}(1-p)^{n} \\ &= (1-p)^{u}\sum_{n=1}^{u-K}\binom{u-1}{n-1}p^{n-1} \end{aligned} \quad (7)$$

3. TCP New-Reno using DAC

DAC를 사용하는 경우 재전송되는 n개의 패킷들

가운데 한 패킷이 다시 손실되더라도 다시 복구 재전송에 의해 복구하는 것이 가능하다. 그러므로 이 경우 손실 복구 확률 R_{DAC}는 다음의 식 (8)으로 주어진다.

$$R_{DAC}=R_{NR}+p\cdot\Delta \tag{8}$$

재전송하는 n개의 패킷들 가운데 몇 번째 패킷이

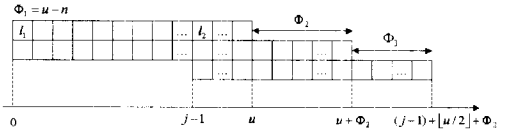


그림 4. 두 번째 손실된 패킷의 재전송 패킷이 다시 손실된 경우 DAC의 손실 복구 과정

다시 손실되었느냐에 따라서 복구되는 조건이 다르기 때문에 정확한 Δ값을 구하기 위해서 다음과 같은 과정이 필요하다. 재전송하는 n개의 패킷들 가운데 첫 번째 패킷이 다시 손실된 경우 이를 다시 재전송하기 위해서는 적어도 u개의 중복 ACK가 발생해야 하므로 $\Phi_1\geq K$ 를 만족하는 u와 n에 대해서 다음의 식 (9)로 주어진다.

$$\begin{aligned}&\Phi_1+\Phi_2\geq u\\&(u-n)+([u/4]-n)\geq u\\&1\leq n\leq[u/4]\end{aligned} \tag{9}$$

즉, 첫 번째 손실된 패킷이 재전송 과정에서 다시 손실된 경우 이를 DAC에 의해서 다시 재전송할 수 있기 위해서는 손실된 패킷들의 수가 윈도우 크기의 1/4을 넘지 않아야 한다. 그러므로 다시 손실된 첫 번째 재전송 패킷의 DAC에 의한 복구 확률을 다음의 식 (10)과 같이 표현된다.

$$\begin{aligned}\Delta_1&=\sum_{n=1}^{[u/4]}\binom{u-1}{n-1}p^{n-1}(1-p)^{u-n}(1-p)^n\\&=\sum_{n=1}^{[u/4]}\binom{u-1}{n-1}p^{n-1}(1-p)^u\end{aligned} \tag{10}$$

그림 4는 u개의 패킷들 중 발생한 n개의 패킷 손실 가운데 재전송 과정에서 다시 손실된 패킷이 두 번째 손실된 패킷인 경우 DAC의 동작을 보여준다. 두 번째 라운드에서 전송한 Φ_2개의 패킷들에 의해서 두 번째 손실된 패킷에 대한 중복 ACK가 발생하게 되고 이 패킷이 재전송 과정에서 다시 손실되는 경우 이 보다 많은 중복 ACK를 받기 위해서는 $\Phi_3\geq 1$ 이어야 한다. $\Phi_3=(j-1)+[u/2]-u$ 이므로 이를 위한 조건은 다음의 식 (11)로 주어진다.

$$j-2\geq u-[u/2] \tag{11}$$

Φ_2개의 패킷들에 의해 발생하는 중복 ACK들은 다음 라운드의 윈도우 크기를 Φ_2 만큼 증가시키므로 Φ_2의 값은 실제적으로 이 조건에 영향을 미치

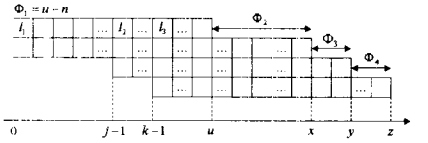


그림 5. 세 번째 손실된 패킷의 재전송 패킷이 다시 손실된 경우 DAC의 손실 복구 과정

지 않는다. 식 (11)에 의해서 손실된 두 번째 재전송 패킷이 DAC에 의해 복구되기 위해서는 적어도 첫 번째 손실된 패킷과 두 번째 손실된 패킷 사이에는 적어도 $u-[u/2]$개의 패킷이 있는 경우로 한정된다는 것을 알 수 있다. 그러므로 손실된 두 번째 재전송 패킷에 대한 DAC의 복구 확률은 다음의 식 (12)로 주어진다.

$$\Delta_2=\sum_{n=2}^{[u/2]}\binom{[u/2]-1}{n-1}p^{n-1}(1-p)^u \tag{12}$$

그림 5는 u개의 패킷들 중 발생한 n개의 패킷 손실 가운데 재전송 과정에서 다시 손실된 패킷이 세 번째 패킷인 경우 DAC의 동작을 보여준다. 만약 $\Phi_3\geq 1$라면 항상 $\Phi_4\geq 1$ 이기 때문에 $\Phi_3\leq 0$ 인 경우에 대해서만 고려하면 된다. 각 라운드의 윈도우의 오른쪽 경계 값들인 x, y 그리고 z의 값은 다음의 식 (13)과 같이 주어진다.

$$\begin{aligned}x&=u+\Phi_2\\y&=(j-1)+[u/2]+\Phi_2\\z&=(k-1)+[u/2]+\Phi_3\end{aligned} \tag{13}$$

그러므로 Φ_4의 값은

$$\begin{aligned}\Phi_4&=z-\max x,u\\&=(k-1)+[u/2]-\begin{cases}u+\Phi_2\\u\end{cases}\\&=\begin{cases}(k-1)-(u-n) & \Phi_2\geq 0\\(k-1)-(u-[u/2] & else\end{cases}\end{aligned} \tag{14}$$

결과적으로 손실된 세 번째 재전송 패킷은 $\Phi_4\geq 1$

인 경우 DAC에 의해 복구 가능하기 때문에 이를 위한 조건은 다음의 식 (15)로 주어진다.

$$k-2 \geq u-\min(n,[u/2]) \tag{15}$$

식 (15)를 통해 첫 번째 손실된 패킷과 세 번째 손실된 패킷 사이에 적어도 $u-\min(n,[u/2])$ 개의 패킷이 있는 경우에 한해 손실된 세 번째 재전송 패킷은 복구가 가능하다는 사실을 알 수 있다. 따라서 복구될 확률은 $m=\min(n,[u/2])$라고 했을 때 식 (16)와 같이 주어진다.

$$\Delta_3 = \sum_{u=3}^{K} \binom{u-m}{1} \binom{m-1}{n-2} p^{u-1}(1-p)^{u} \tag{16}$$

이 후 TCP New-Reno의 손실 복구 과정은 계속 같은 방식으로 반복되기 때문에, $h \geq 4$인 경우 재전송 과정에서 다시 손실된 l_h가 DAC에 의해 복구되는 과정은 l_1과 l_h사이에 있는 손실된 패킷의 개수만 차이가 있을 뿐 l_3가 복구되기 위한 조건과 동일하다. 즉, DAC에 의해 복구되는 패킷의 손실 윈도우 내 위치만이 이 패킷의 복구 여부를 결정한다. 그러므로 이 때의 손실 복구 확률은 다음 식 (17)로 주어진다.

$$\sum_{h=3}^{n} \Delta_h = \sum_{n=3}^{K} \sum_{h=3}^{n} \binom{u-m}{h-2} \binom{m-1}{n-h+1} p^{u-1}(1-p)^{u} \tag{17}$$

결과적으로 DAC를 사용하는 경우 손실 복구 확률은 식 (8), (10), (12) 그리고 (17)에 의해 다음의 식 (18)으로 주어진다.

$$\begin{aligned} R_{DAC} &= R_{NR} + p \cdot \Delta \\ &= R_{NR} + p\left\{\Delta_1 + \Delta_2 + \sum_{h=3}^{n} \Delta_h\right\} \end{aligned} \tag{18}$$

Ⅳ. 결과 및 분석

1. 손실 복구 확률

각 TCP의 손실 복구 기능을 비교 분석하기 위해서 평균 손실 복구 확률을 다음의 식 (19)과 같이 정의한다. 이 값은 손실된 패킷들에 RTO가 발생하지 않을 확률을 의미한다.

$$\overline{R} = \sum_{u=1}^{W_{max}} R_{TCP} \cdot \pi(u) \tag{19}$$

R_{TCP}는 각 TCP의 손실 복구 확률이고 W_{max}는 수신원이 허용하는 최대 윈도우의 크기 (advertised window)값이다. $\pi(u)$는 정상 상태에서의 윈도우 크기의 확률 분포를 의미하는데 Markov chain을 이용한 송신원의 윈도우의 주기적인 변화 특성을 이를 분석함으로써 얻어진다.

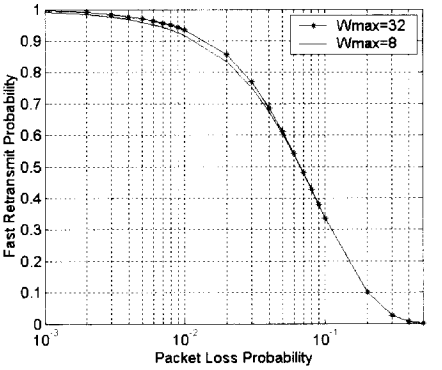


그림 6. 임의 패킷 손실 확률에 대한 TCP Reno의 손실 복구 확률

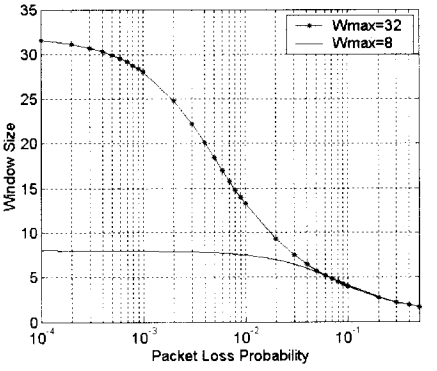


그림 7. 임의 패킷 손실 확률에 대한 TCP Reno의 윈도우 평균 크기 분포

다음의 참조 문헌들[13,14,16]을 통해 이에 대한 자세한 과정에 대해 알 수 있다.

2. Numerical Results

그림 6은 W_{max}의 값이 8과 32일 때, 임의 패킷 손실 확률에 따른 TCP Reno의 손실 복구 확률을

보여준다. 분석 과정의 모든 경우에 대해 $K=3$으로 설정한다. 패킷 손실 확률이 1%(=0.01)보다 커짐에 따라 손실 복구 확률이 급격하게 감소하는 것을 볼 수 있다. 패킷 손실 확률이 10%보다 커지면 손실된 패킷들이 복구되는 것은 거의 불가능하다. $W_{max}=8$인 경우 하나의 윈도우 내에서 두 개 이상의 패킷 손실이 발생하는 경우 무조건 RTO가 발생하는 반면($R_R=R_R^{(1)}$), $W_{max}=32$인 경우

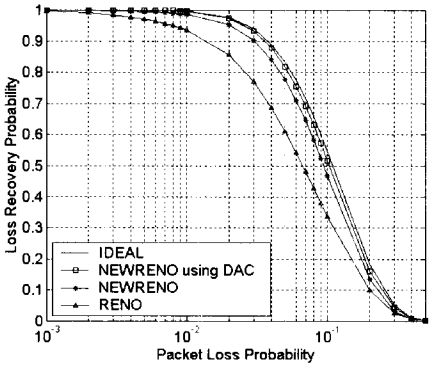


그림 8. 임의 패킷 손실에 대한 각 TCP의 손실 복구 확률의 비교 (Wmax=8)

$R_R=R_R^{(1)}+R_R^{(2)}+R_R^{(3)}$이다. 그러나 그림에서 볼 수 있듯 W_{max} 값은 패킷 손실 확률에 그다지 큰 영향을 주지 못한다. 이 사실에 대한 이유로 첫 째 임의 패킷 손실 모델에서는 하나의 윈도우 내에서 두 개 이상의 패킷 손실이 발생할 확률이 낮기 때문에 $R_R^{(2)}$, $R_R^{(3)}$의 값은 매우 작은 값을 가진다는 점을 들 수 있다.

두 번째 이유는 패킷 손실에 따른 윈도우의 크기 분포에서 찾을 수 있다. 다음 그림 7은 평균 윈도우를 다음의 식 (20)과 같이 정의할 때 두 가지 경우의 평균 윈도우의 크기 분포를 보여준다.

$$\overline{W}=\sum_{u=1}^{W_{max}} u\cdot\pi(u) \tag{20}$$

패킷 손실 확률이 증가함에 따라 평균 윈도우의 크기는 점차 감소하고 약 패킷 손실 확률이 0.05 보다 커지게 되면 두 경우의 윈도우의 분포는 일치한다. 이는 패킷 손실 확률이 높을수록 잦은 혼잡 제어로 인해서 윈도우의 감소가 빈번하게 발생하고 결과적으로 큰 W_{max}로 인한 이득은 거의 없다고 할 수 있다.

다음 그림 8과 9는 W_{max}가 8과 32일 때 임의 패킷 손실 확률에 대한 각 TCP의 손실 복구 확률들을 비교한 것이다. 이상적인 경우의 손실 복구 확률은 손실 복구 과정에서 전송하는 재전송 패킷의 손실 여부에 관계없이 단지 첫 번째 손실된 패킷에 대한 fast retransmit이 가능한지 만을 고려한 것으로 R_{Ideal}은 다음의 식 (21)로 주어진다.

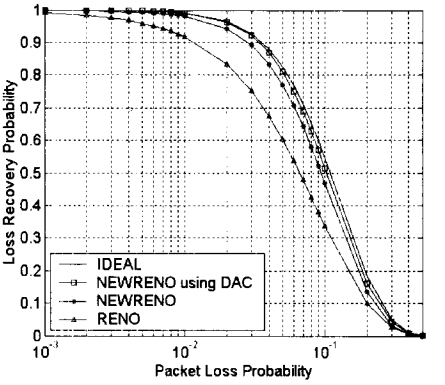


그림 9. 임의 패킷 손실에 대한 각 TCP의 손실 복구 확률의 비교 (Wmax=32)

$$R_{Ideal}=\sum_{n=1}^{u-K}\binom{u-1}{n-1}p^{n-1}(1-p)^{u-n} \tag{21}$$

즉, 첫 번째 손실된 패킷이 fast retransmit에 의해 재전송 될 수 있다면 무조건 RTO는 발생하지 않는 경우를 의미하며, 이는 SACK option[4,10]을 추가 보완함으로써 가능하다. W_{max}의 값에 관계없이 거의 비슷한 분포를 보이는 점은 앞에서 설명한 TCP Reno의 경우와 동일한 이유에서이다. 패킷 손실 확률이 0.01보다 커짐에 따라 TCP New-Reno의 손실 복구 확률은 TCP Reno에 비해 월등하게 높지만 이상적인 경우에 비해서는 다소 작은 값을 보인다.

TCP New-Reno의 손실 복구 확률과 이상적인 경우의 손실 복구 확률은 손실된 재전송 패킷이 복구될 수 있느냐 없느냐의 차이이다. 패킷 손실 확률이 높아짐에 따라 두 경우의 손실 복구 확률의 차이가 생기는 것을 볼 수 있다. 물론 평균적으로 봤을 때 재전송 패킷 손실이 발생하는 빈도는 그리 빈번하지 않기 때문에 TCP Reno의 손실 복구 확률과 TCP New-Reno의 손실 복구 확률의 차이보다는 크지 않다.

DAC를 사용하는 경우에는 하나의 재전송 패킷 손실을 복구하는 것이 가능하기 때문에 TCP New Reno의 손실 복구 확률보다는 높은 손실 확률을 가지게 되며 거의 이상적인 경우에 근접하는 것을 볼 수 있다. 특히, 기존의 SACK option을 사용하는 경우에도 재전송 패킷 손실이 발생하는 경우 RTO는 무조건 발생하며 이의 보완이 이루어진다고 하더라도 SACK option은 송신원과 수신원이 모두 이를 지원하는 경우에만 사용할 수 있다는 점을 고려할 때 DAC는 TCP의 손실 복구 차원에서 상당한 성능 향상을 가져오는 것으로 평가할 수 있다.

V. 결 론

비 혼잡 패킷 손실이 존재할 때 TCP의 전반적인 성능은 손실 패킷 복구 과정의 효율성에 의해 결정된다. TCP Reno의 손실 복구 과정은 하나의 윈도우 내에서 하나의 패킷이 손실되는 경우에 효율적으로 동작하지만, 다수 개의 패킷들이 손실되는 경우 빈번한 RTO의 발생으로 인한 성능 저하가 발생한다. TCP New-Reno의 손실 복구는 부분 ACK를 도입하여 fast recovery를 연장함으로써 다수 개의 패킷 손실을 재전송을 통해 복구하는 것이 가능하다. 본 논문에서는 TCP Reno와 New-Reno의 손실 복구 과정을 정확하게 모델링하고 이의 효율성을 확률적인 분석을 통해 비교하였다.

TCP New-Reno의 손실 복구 과정은 TCP Reno의 손실 복구 과정에 비해서 매우 효율적이지만, 재전송된 패킷이 다시 손실되는 경우 RTO를 피할 수 없는 단점을 가지고 있다. 본 논문에서는 손실된 패킷에 대해 발생이 예상되는 중복 ACK의 수를 실제 수신되는 중복 ACK의 수와 비교하여 이 문제점을 완화할 수 있는 DAC 알고리듬을 제안한다. DAC가 New-Reno의 손실 복구 과정에 추가되는 경우 약 0.01에서 0.1의 값을 가지는 패킷 손실 확률에 대해 New-Reno에 비해 최대 5%정도의 손실 복구 확률을 높이는 것이 가능하고, SACK option을 수정, 보완함으로써 가능한 이상적인 TCP의 손실 복구 과정에 거의 근접하는 손실 패킷 능력을 가지게 된다. 특히, DAC는 기존 TCP의 명세 (specification)에 부합하는 동시에 송신원의 간단한 수정만으로써 구현될 수 있는 장점을 가지고 있다.

참 고 문 헌

[1] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. H. Katz, "A Comparison of Mechanisms for Improving TCP Performance over Wireless Links," *IEEE/ACM Trans. Networking*, vol. 5, no. 6, pp. 756-769, Dec. 1997.

[2] V. Jacobson, "Congestion Avoidance and Control," in *Proc. ACM SIGCOMM'88*, Aug. 1988.

[3] V. Jacobson, "Modified TCP congestion avoidance algorithm," note sent to end2end-interest mailing list, 1990.

[4] M. Mathis, S. Floyd, and A. Romanow, "TCP Selective Acknowledgement Options," Oct. 1996.

[5] J. Hoe, "Improving the Start up Behavior of a Congestion Control Scheme for TCP,". in *Proc. ACM SIGCOMM'96*, Aug. 1996.

[6] K. Fall and S. Floyd, "Simulation based Comparisons of Tahoe, Reno, and SACK TCP," *Computer Communication Review*, pp. 521, vol. 26, no. 3, Jul. 1996.

[7] W. Stevens, *TCP/IP Illustrated, vol. 1 The Protocols*, Reading, MA: Addison-Wesley, 1997.

[8] W. Stevens, "TCP slow start, congestion avoidance, fast retransmit, and fast recovery algorithms," RFC2001, Jan. 1997.

[9] M. Allman, V. Paxson, and W. Stevens, "TCP Congestion Control," RFC2581, Apr. 1999.

[10] S. Floyd, J. Madavi, M. Mathis, and M. Podolsky, " An Extention to the Selective Acknowledgement Option for TCP," RFC2883, Jul. 2000.

[11] P. P. Mishira, D. Sanghi, and S. K. Tripathi, "TCP flow control in lossy networks: Analysis and enhancements," in *Computer Networks, Architecture and Applications*, IFIP Transactions C-13, S. V. Raghavan, G. V. Bochman, and G. Pujolle, Eds. Amsterdam, The Netherlands: Elsevier North Holland, pp. 181-193, 1993.

[12] T. V. Lakshman and Upamanyu Madhow, "The Performance of TCP/IP for Networks with High Bandwidth-Delay Products and Random Loss," *IEEE/ACM Trans. Networking*, vol.5, no.3, pp. 336-350, Jun. 1997.

[13] Anurag Kumar, "Comparative Performance Analysis of Versions of TCP in a Local Network with a Lossy Link," *IEEE/ACM Trans. Networking*, vol. 6. no. 4, pp. 485-498, Aug. 1998.

[14] Anurag Kumar and Jack Holtzman, (Feb. 1998). Comparative Performance Analysis of Versions of TCP in a Local Netowork with a Mobile Radio Link [Online] http://ece.iisc.ernet.in/~anurag/

[15] Farooq Anjum and Leandros Tassiulas, "On the Behavior of Different TCP Algorithms over a Wireless Channel with Correlated Packet Losses," in *Proc. ACM SIGMETRICS '99*, pp 155-165, May 1999.

[16] J. Padhye, V. Firoiu, D. F. Towsley, and J. F. Kurose, "Modeling TCP Reno Performance: A Simple Model and Its Empirical Validation," *IEEE/ACM Trans. Networking*, vol.8, no.2, pp. 133-145, Apr. 2000.

[17] Michele Zorzi and A. Chockalingam, "Throughput Analysis of TCP on Channels with Memory," *IEEE J.Select.Areas Commun.*, vol. 18, no. 7, pp. 1289-1300, Jul. 2000.

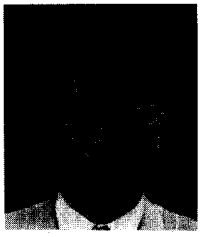
[18] Alhussein A., Abouzeid, S. Roy, and M. Azizoglu, "Stochastic Modeling of TCP over Lossy Links," in *Proc. IEEE Infocom'2000*, pp. 1724-1733.

김 범 준(Beom-Joon Kim) 정회원

1996년 2월 : 연세대학교 전자공학과 졸업
1998년 8월 : 연세대학교 전자공학과 석사
1998년 9월~현재 : 연세대학교 전자공학과 박사과정

<주관심분야> TCP 성능 평가, 무선 인터넷

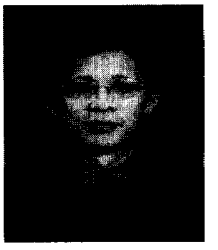
김 동 연(Dong-Yeon Kim) 정회원



1986년 2월 : 연세대학교 전자공학과 졸업
1988년 2월 : 연세대학교 전자공학과 석사
1995년 2월 : 연세대학교 전자공학과 박사
1996년~현재 : 국립 한경대학교 전자공학과 부교수

<주관심분야> 위치/이동성 관리, 위성 ATM

이 재 용(Jai-Yong Lee) 정회원



1977년 2월 : 연세대학교 전자공학과 졸업
1984년 5월 : IOWA State University 공학 석사
1987년 5월 : IOWA State University 공학 박사
1987년 6월~1994년 8월 : 포항공과대학 교수
1994년 9월~현재 : 연세대학교 전자공학과 교수

<주관심분야> Protocol Design for QoS Management, Network Management, High Speed Networks