韓國 OA學會 論文誌
第4卷 第4號, 1999.12.

99-4-4-3-6

TCP 알고리즘의 성능 비교

김 노 환*, 박 준 식**

Comparing the Performance of TCP Algorithms

No-whan Kim*, Jun-sik Park**

요 약

현재 인터넷에서 사용되고 있는 TCP 방식은 많은 논문들에 의해 제안과 수정을 통해 그 성능이 개선되어 있다. 본 논문에서는 현재 사용중이거나 제안된 Tahoe, Reno, New-Reno, Vegas 및 SACK(Selective Ack) 알고리즘의 성능을 비교하였다. Tahoe는 최초에 제안되어 현재 광범위하게 사용되고 있는 알고리즘이며, Reno는 한 윈도우 내에 하나의 패킷 손실이 발생한 경우에는 최적의 성능을 발휘하지만, 한 윈도우 내에 다수의 패킷 손실이 발생한 경우에는 재전송 타이머의 타임아웃으로만 패킷 복구가 가능하여 그 성능이 다른 알고리즘에 비해 저하된다. New-Reno는 Reno의 이러한 문제점을 개선한 것으로 한 RTT(Round-Trip-Time)내에 다수의 패킷 복구가 가능하지만 불필요한 재전송의 문제가 존재한다. SACK는 위와 같은 모든 문제점을 해결하였으며 bandwidth delay product가 큰 위성망에서 사용이 제안되고 있다. TCP Vegas는 송신기가 경로상의 대역폭을 판단하여 윈도우의 크기를 알맞게 조절하는 방식으로 망 혼잡이 큰 경우를 제외하고는 성능면에서 Reno의 비해 40~70%가 개선된다.

Abstract

TCP has improved by many papers which suggest the new algorithms and modify the previous algorithms. This paper compares Tahoe, Reno, New-Reno, Vegas, and SACK. The first version is Tahoe and is globally used. Reno has optimal performance during occurring one packet loss within a window of data, but can suffer from performance when multiple packets are dropped from a window of data. New-Reno avoids some of the performance problems of Reno TCP when multiple packets are dropped from a window of data, but is occurring the problem of the necessary retransmission. SACK resolves the all above problems and is used in bandwidth delay product environment. Vegas uses network bandwidth more efficiently and is a new implementation of TCP that achieves between 40 and 70 better throughput, with one-fifth to one-half the losses, as compared to the implementation of Reno TCP.

* 동우대학 사무자동화과 조교수
** 영동전문대학 전자과 조교수
논문접수:1999.10.4. 심사완료:1999.11.30

Ⅰ. 소 개

현재 인터넷에서 사용되는 TCP는 그 버전에 따라서 혼잡 제어 방식이 다르며 많은 시간동안 수정이 거듭되어 TCP 성능이 개선되어 왔다[1, 2]. 대표적인 TCP의 버전은 Tahoe, Reno이며 한 윈도우내의 다수 패킷 손실시, Reno의 성능을 개선한 것이 New-Reno이다[3]. 이러한 TCP에 송·수신기의 수정하여 SACK option을 추가하여 송신기가 수신기의 패킷 수신 상태를 정확하게 판단하도록 제안된 것이 SACK이다[3, 4]. 이외에 경로상의 대역폭을 판단하여 윈도우의 크기를 조절하고 재전송 방식과 Slow Start 단계를 수정한 것이 Vegas이다[5]. 본 논문에서는 한 윈도우 내에 다수의 패킷 손실이 발생한 경우에 위의 각 TCP 알고리즘을 비교하여 그 성능의 차이를 검토해 보았다.

Ⅱ. 각 TCP의 비교

본 장에서는 각 TCP 버전이 사용하고 있는 알고리즘을 간단히 소개하였다.

2.1 Tahoe

Tahoe는 초기에 slow-start를 시작하여 패킷을 송수신하다가, 망에서 패킷 손실이 발생할 경우 즉 time-out 또는 duplicate ack가 3개 수신되면 윈도우 임계치(ssthreshold)를 패킷 손실전의 윈도우 크기의 반(ssthresh=cwnd/2)으로 설정한 후, cwnd=1로 하여 slow-start를 실행한 다음 congestion avoidance(혼잡회피) 단계를 거쳐서 정상적인 TCP 동작을 실행한다.

현재의 윈도우 크기 : W,
윈도우 임계치(Threshold) : Wt

1) 정상적인 Ack의 경우 :
W 〈 Wt, W = W + 1 : slow start 단계
그 밖의 경우, W = W + 1/W : 혼잡회피 단계

2) 타임아웃 또는 duplicate Ack가 3개 수신된 경우 :
Wt = W/2
W = 1 : slow start 실행

여기서, duplicate Ack가 3개 수신된 경우에는 fast retransmission을 실행한 후 slow start를 실행한다.[6]

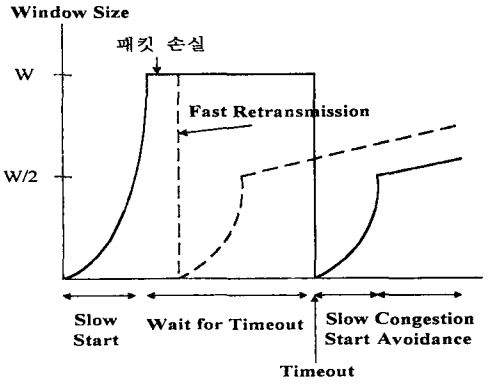

그림 1. 다수의 패킷 손실에 대한 Tahoe의 윈도우 크기 변화

Tahoe의 경우에는 한 윈도우 내에 처음 발생한 손실된 패킷을 재전송한 후 차례로 slow start를 실행하면서 이후의 모든 패킷들을 재 전송하므로 수신기가 이미 수신한 패킷을 다시 수신하게 되는 문제점은 있으나 종국적으로 패킷손실이 발생하지 않아 안전성은 높다. 결국, Taeho는 Slow Start, Congestion Avoidance 및 Fast Retransmission을 지원하며 fast recovery과정은 실행하지 않는다.[1, 2].

2.2 Reno

Reno는 Tahoe와 마찬가지로 초기에slow-start를 시작하여 패킷을 송수신하다가, 망에서 패킷 손실이 발생할 경우 즉 duplicate ack가 3개 수신되면, ssthresh=cwnd/2로 설정한 후, cwnd=ssthresh로 하고 congestion avoidance 단계를 실행한다. tahoe에서는 cwnd=1로 하고 slow Start를 실행했었다. time-out 시에는 tahoe와 마찬가지로 ssthresh=cwnd/2로 설정

하고 cwnd=1로 한 후, slow-start와 congestion avoidance 단계를 거친다.

1) 정상적인 Ack의 경우는 Taeho와 같다.
W < Wt, W = W + 1 : slow start 단계
그 밖의 경우, W = W + 1/W : 혼잡회피 단계

2) duplicate Ack가 3개 수신된 경우 해당 패킷을 재전송하고 fast recovery 과정을 거친다.
Wt = W/2, W = Wt
재전송 패킷이 Ack될 때, 이전 윈도우 크기의 반으로 윈도우 크기를 설정 혼잡회피 단계를 실행한다.

3) 재전송 타이머가 만기된 경우도 Taeho와 같다.
Wt = W/2
W = 1 : slow start 실행

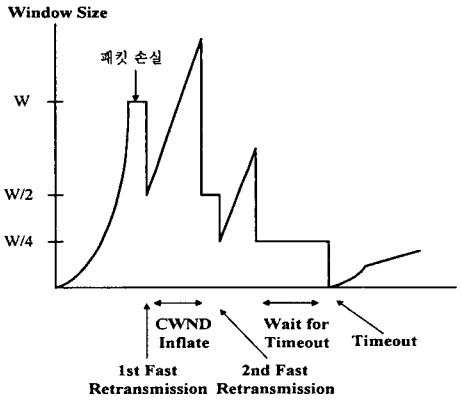

그림 2. 다수의 패킷 손실에 대한 Reno의 윈도우 크기 변화

Reno는 Tahoe의 fast retransmission을 수정하여 3개의 duplicate Ack가 수신되면 fast retransmission을 실행함과 동시에 윈도우 크기를 반으로 줄이고 재전송 이후의 duplicate Ack의 수에 따라 윈도우 크기를 조절하는 fast recovery단계를 실행한다. 또한, Reno 알고리즘은 한 윈도우 내에 하나의 패킷 손실이 발생한 경우에 가장 좋은 성능을 발휘하게 되는데, 두 개 이상의 패킷 손실이 발생하면 첫 번째 패킷을 재전송하고 두 번째 패킷에 대한 duplicate ack 3개의 수신 가능성이 적어짐으로 타임아웃이 발생할 확률이 높아 Tahoe보다 성능이 좋지 않다. 결국, Reno는 Tahoe에 Fast Recovery 과정을 추가한 알고리즘이다[6, 7].

2.3 New-Reno

Reno에서는 첫번째 패킷 손실에 대한 duplicate Ack를 수신하여 복구할 수는 있지만 연속되는 두·세번째 패킷 손실에 대한 duplicate Ack 3개를 수신할 수 있는 확률이 작아서 송신기의 타이머가 종료될 때까지 타임아웃 기간동안 어떤 데이터도 전송하지 못하는 문제점을 갖고 있다.

New-Reno는 Reno에서 문제가 된 다수의 패킷손실에 대한 성능을 개선한 알고리즘으로, 첫번째 패킷 손실은 duplicate Ack 3개를 수신하여 전송하지만 두·세번째 패킷 손실은 duplicate Ack 3개를 수신하지 않고 첫번째 Ack가 수신되어도 패킷이 손실되었다고 가정하고 재전송을 수행함으로써 한 윈도우내에 서 발생된 다수의 패킷 손실에 대해 타임아웃이 발생하는 경우없이 패킷을 재전송한다. RTT당 하나의 손실된 패킷을 재전송하며 Partial Ack(Partial Ack은 송신기가 손실된 패킷을 재전송하여 이 재전송된 패킷이 수신기에 수신되었다는 Ack을 의미하며 송신기의 TCP Sequence Number를 증가시킨다.)가 수신될 경우, Reno는 재전송한 패킷에 해당하는 Ack가 수신되면 fast recovery를 곧 바로 종료하고 Congestion avoidance 단계를 수행하지만 New-Reno는 이러한 Ack가 수신되어도 혼잡 윈도우의 크기를 변화시키지 않으며 바로 fast Recovery를 종료하지 않고, 두·세번째 패킷 손실에 대한 복구를 계속 실행한다. 결국, New-Reno는 다수의 패킷 손실이 한 윈도우 내에서 발생할 때, 재전송 타임아웃이 발생되지 않고 한 윈도우 내에서 손실된 패킷 모두가 재전송될 때까지 RTT(Round Trip Time)당 손실된 패킷을 하나씩 재전송하며 이외의 동작원리는 Reno와 같다[1].

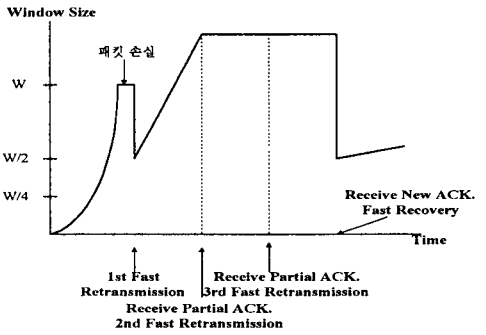

그림 3. 다수의 패킷 손실에 대한 New-Reno의 윈도우 크기 변화

2.4 SACK(Selective Ack)

SACK은 아직까지는 표준화가 되지 않은 상태지만, 이미 최대 4개의 블록을 포함하는 SACK option의 사용이 제안된 바 있으며 이러한 SACK option을 사용하면 Tahoe나 Reno에서처럼 수신기에서 이미 수신된 패킷을 불필요하게 재전송하는 문제가 발생하지 않게 되어[3, 4] 한 윈도우 내에 발생하는 다수의 패킷 손실에 잘 대처할 수 있으며 순간적인 성능 저하가 없다.

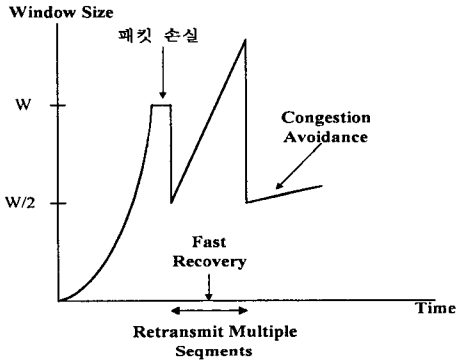

그림 4. 다수의 패킷손실에 대한 SACK의 윈도우 크기 변화

SACK은 기본적으로 현재의 TCP에 기초한 go-back-N 오류제어 방식과는 달리 개별적인 패킷에 대한 오류를 각각의 송신측에 피드백 하여 오류제어의 성능을 높이는 방식으로, 수신기는 성공적으로 수신한 모든 패킷상태를 송신기에게 알리며 송신기는 이 SACK을 바탕으로 실제로 손실된 패킷만을 재전송한다. 이 알고리즘은 두 개의 TCP option을 사용하고 있는데 연결이 시작될 때, 송신기가 수신기에게 전송하는 SACK option의 사용 여부를 결정하는 SYN내의 "SACK-permitted"와 SACK가 허락되었을 때 수신기가 송신기에게 전송하는 SACK option이 있다.[7]

1). SACK-Permitted Option

연결 설정시, 송신 TCP가 지정된 2 Byte의 option을 전송하면 수신 TCP가 SACK option을 사용할 수 있으며, 전송되지 않을 경우에는 SACK option이 사용되지 않는다.

2). SACK Option 형식

SACK option은 설정된 TCP 연결 경로로 수신기에서 송신기로 확장된 Ack 정보를 전송하기 위해 사용된다.

SACK Option :

Length : Variable

		Kind = 5	Length
Left Edge of 1st Block			
Right Edge of 1st Block			
.....			
Left Edge of nst Block			
Right Edge of nst Block			

그림 5. SACK Option

2.5 Vegas

Vegas는 기존의 Reno 버전에서 재전송(retransmit), congestion avoidance, slow start, 타임아웃(time-out)을 수정한 알고리즘으로[5], 손실이 예상되면 윈도우 크기를 줄여 손실발생 확률을 줄임으로써 Reno에 비해 40~70%의 throughput과 1/5~1/2의 loss 향상을 보이고 있다.

1) 재전송 mechanism

첫번째 duplicate Ack가 수신되면, IP header option에 있는 패킷이 전송된 시간을 알 수 있는 타임스탬프(timestamp)를 이용하여, 패킷의 RTT 값이 재전송 타이머의 타임아웃 값보다 클 경우에는 3개의 duplicate Ack을 기다리지 않고 곧바로 해당 패킷을 재전송하며 이후의 duplicate Ack도 타임스탬프를 이용하여 재전송을 실행한다.

2) modifed congestion avoidance mechansm

가장 짧은 RTT를 BaseRTT로 설정하고

$$\text{Expected rate} = \frac{WindowSize}{BaseRTT}$$

여기서, Expected rate란 패킷을 전송 할 때 송신기가 어느 정도의 전송속도를 가질 수 있는지의 예측치

실제 전송율을 송·수신되는 패킷과 Ack를 이용하여 측정한 후, Diff 값에 따라서 윈도우의 크기를 다음과 같이 조절한다.(α, β : 상수)

Diff = Expected Rate -Actual Rate
Diff ⟨ α : 윈도우 증가
Diff ⟩ β : 윈도우 감소
α ⟨ Diff ⟨ β : 윈도우 변화 없음

Vegas는 예측한 전송율을 기준으로, 실제 측정한 전송율이 크면 윈도우를 증가시키고 실제 전송율이 작으면 윈도우를 감소시키므로써 망의 혼잡을 피해서 loss를 줄이는 알고리즘이다.

3) modified slow start mechanism

slow start시에 RTT마다 일반적인 slow start를 실행하되 실제 전송률이 Expected Rate에 비해 특정 임계치(γ)보다 낮을 경우에는 ssthrehold와는 상관없이 혼잡회피 단계를 실행한다.

4) modified time-out mechanism

타이머의 타밍아웃이 발생한 경우에는 Reno와 마찬가지로 slow-start를 실행하게 되는데 Vegas에서는 Reno보다 좋은 TCP tick을 사용하므로 timeout 측정이 훨씬 정확하다.

Ⅲ. Simulation

본 모의실험은 Network Simulator ns v1.4를 사용하였으며 송신기와 라우터간의 전송 속도와 지연은 8Mbps, 0.1ms로, 라우터와 수신기간의 전송 속도와 지연은 0.8Mbps, 100ms로 각각 설정하여 한 윈도우 내에 다수의 패킷 손실이 발생하게 하였고 귀환경로의 Ack가 손실되지 않는 조건하에서, TCP vegas의 α값은 1로, β값은 3으로, γ값은 1로 각각 설정하였다. 다음의 시험결과는 각 TCP 알고리즘에 따른 시간과 Sequence Number의 관계를 그림으로 표현한 것이다.

그림 6의 Tahoe는 2와 4초 사이의 패킷 손실에 대하여 Slow Start를 수행하는 송신기의 재전송 과정을 보여주고 있다. 이 과정에서 수신기에 이미 저장되어 있는 패킷들을 송신기가 다시 재전송함으로써 불필요한 재전송 문제가 발생한다.

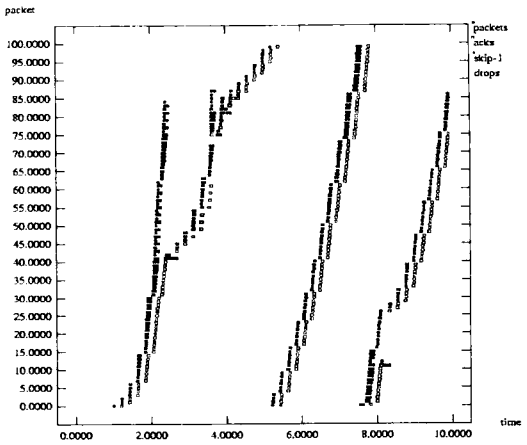

그림 6. Tahoe

그림 7의 Reno는 패킷 손실마다 윈도우 크기를 감소하기 때문에 재전송 타이머의 타임아웃만이 패킷 복구를 가능하게 함으로써 2초와 4초 사이의 긴 타임아웃 기간에 송신기가 데이터를 전송하지 않는 기간 때문에 그 성능이 크게 저하됨을 보이고 있다.

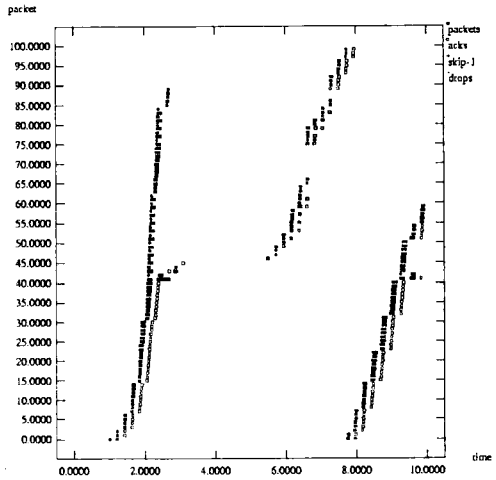

그림 7. Reno

그림 8의 New-Reno는 Reno를 수정한 것으로 2초 6초 사이의 다수의 패킷 손실에 대해서 RTT당 하나의 패킷만을 복구하여 패킷 복구 기간이 길어진 반면에, 그림 9의 SACK은 SACK option을 이용하여 Fast

Recovery 단계에서 손실된 모든 패킷을 정확히 복구하여 이전 알고리즘들에 비해 패킷 복구 기간이 짧게 나타나고 있다.

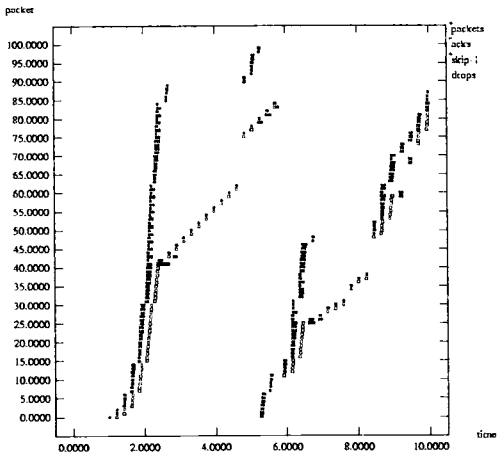

그림 8. New-Reno

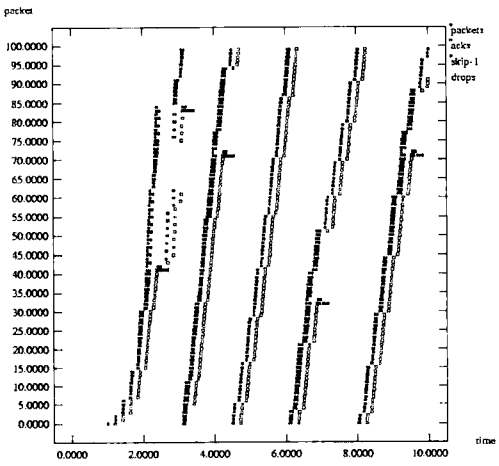

그림 9. SACK

그림 10의 TCP Vegas는 송신기가 경로상의 대역폭을 정확히 판단하여 윈도우 크기를 조절함으로써 다른 알고리즘에 나타났던 패킷 손실이 발생하지 않는다.

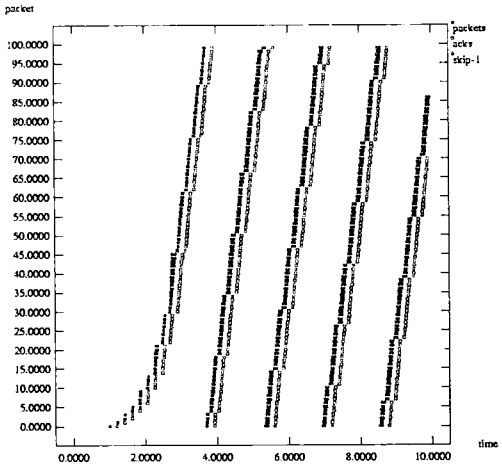

그림 10. TCP Vegas

그림 11은 이전 그림들의 TCP 성능을 같이 비교한 것이며 Tahoe의 성능은 242Kbps, Reno는 139Kbps, NewReno는 149Kbps, SACK은 434Kbps, Vegas는 417Kbps 이다. 위의 다수의 패킷 손실이 발생하는 실험 조건에서는 SACK이 가장 좋은 성능을 발휘하고 있고 그 다음이 Vegas이다.

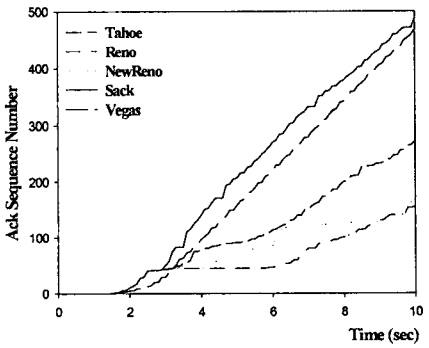

그림 11. 각 TCP의 성능 비교

물론, 망 배치에 따라서 각 TCP 버전의 성능 차이가 다르게 나타날 수 있으며 Vegas는 망의 대역폭을 적절히 판단하지 못하는 경우에는 패킷 손실이 발생할 수도 있다. 앞으로는 여러 망 배치를 구성하여 각 상황에 최적의 성능을 발휘할 수 있는 TCP 버전을 찾을 계획이다.

Ⅳ. 결론

현재 인터넷에서 사용되는 TCP 방식은 경로 상에 혼잡이 증가하여 한 윈도우 내에 다수의 패킷 손실이 발생한 경우에 TCP의 성능이 크게 저하되며 Slow Start의 지수적인 윈도우 증가가 망의 혼잡을 가중시킨다. 이러한 문제점을 해결하기 위해 많은 방식들이 제안되고 있으며 본 논문에서는 그 방식들 중에서 대표적인 방식들인 Tahoe, Reno, New-Reno, SACK 및 TCP Vegas를 동일한 상황에서 모의실험을 하였으며 다수의 패킷 손실이 연속해서 발생하는 망이 구성될 경우에는 SACK과 Vegas가 좋은 성능을 보이고 있다.

참고문헌

[1] Sally Floyd. TCP and successive fast retransmits.
ftp://ftp.ee.lbl.gov/papers/fastretrans.ps, February 24, 1995.

[2] Van Jacobson. Congestion avoidance and control. ACM SIGCOMM 88, pages 273-288, 1988.

[3] K. Fall and S. Floyd, "Comparisons of Tahoe, Reno,and Sack TCP".
ftp://ftp.ee.lbl.gov/papers/sack.ps.Z,Dec. 1995.

[4] M. Mathis, J.Mahdavi, S. Floyd, and A. Romanow, "TCP Selective Acknowledgmen Options", Internet draft, 2018.txt, October 1996.

[5] Lawrence S. Brakmo, San W. O'Malley, and Larry. L. Peterson. TCP Vegas: New techniques for congestion detection and avoidance. In Proceedings of ACM SIGCOMM '94, pages 24-35, May 1994.

[6] W. Stevens, "TCP slow start, congestion avoidance, fast retransmit, and fast recovery algoritms", RFC 2001.

[7] Sally Floyd. "Issues of TCP With SACK". ftp://ftp.ee.lbl.gov/papers/issues_sa.ps.Z, January 1996.

저 자 소 개

김 노 환

1978년 숭실대학교 전자공학과 (공학사)
1985년 연세대학교 대학원 (공학석사)
1997년~현. 강원대학교 대학원 박사과정
1980년 금성전기(주) 기술연구소
1983년 현대전자산업(주) 시스템 연구소
1993년~현재 : 동우대학 사무자동화과 조교수
관심분야 : TCP/IP network, ATM network

박 준 식

1988년 동아대학교 전자공학과 (공학사)
1991년 경남대학교 대학원 (공학석사)
1998년~현. 강원대학교 대학원 박사과정
1978~1991 한국통신 부산사업본부
1991~1996 한국통신기술(주) 교환감리실
1997~현재 : 영동전문대학 전자과 조교수
관심분야 : 초고속통신, signal processing, CALS