ns-2 시뮬레이터를 이용한 TCP 재전송 손실 복구 알고리듬의 구현

김범준*

Implementation of TCP Retransmitted Packet Loss Recovery using ns-2 Simulator

Beom-Joon Kim*

요 약

인터넷에서 널리 사용되고 있는 수송 계층 프로토콜인 TCP(transmission control protocol)의 혼잡제어(congestion control) 기능은 손실된 패킷을 감지하고 복구하기 위한 손실 복구(loss recovery) 과정을 포함한다. 손실 복구 과정은 fast retransmit와 fast recovery 두 개의 알고리듬으로 이루어지는데 불필요한 재전송 타임아웃을 방지하기 위한 많은 연구가 이루어져 왔다. 그 결과로 최근에는 선택 승인(selective acknowledgement) 옵션과 제한 전송(limited transmit) 기법이 제안되어 IETF (Internet Engineering Task Force)의 표준 문서로 채택되었다. 최근에는 재전송된 패킷이 다시 손실되는 경우 발생하는 타임아웃을 방지하기 위한 재전송 손실 복구(lost retransmission detection)를 위한 방법이 제시되었다. 그러나 아직 재전송 손실 복구 기능의 TCP 혼잡 윈도우의 가장 기본적인 동작 원칙인 AIMD (additive increase multiplicative decrease) 측면에서의 분석이 되어 있지 않은 상태이다. 따라서 본 논문에서는 이를 고려한 재전송 손실 복구 알고리듬의 동작을 시뮬레이션을 통해 평가한다.

ABSTRACT

Transmission control protocol(TCP) widely used as a transport protocol in the Internet includes a loss recovery function that detects and recovers packet losses by retransmissions. The loss recovery function consists of the two algorithms; fast retransmit and fast recovery. There have been researches to avoid nonnecessary retransmission timeouts (RTOs), which leads to selective acknowledgement (SACK) option and limited transmit scheme that are standardized by IETF (Internet Engineering Task Force). Recently, a method that covers the case in which a retransmitted packet is lost again has been propsed. The method, however, is not proved in terms of the additive increase multiplicative decrease (AIMD) principle of TCP congestion control. In this paper, therefore, we analyzed the method in terms of the principle by ns-simulations.

키워드

transmission control protocol (TCP), loss recovery, retransmitted packet loss, ns-2 simulation
TCP, 손실 복구, 재전송 패킷 손실, ns-2 시뮬레이션

* 계명대학교 전자공학과(bkim@kmu.ac.kr)
접수일자 : 2012. 05. 31 심사(수정)일자 : 2012. 07. 26 게재확정일자 : 2012. 08. 09

Ⅰ. 서 론

TCP(transmission control protocol)는 패킷 손실이 발생하지 않는 한 slow start와 congestion avoidance를 통해서 윈도우의 크기를 지속적으로 증가시키므로 언젠가는 혼잡으로 인한 패킷 손실이 발생하게 된다[1][2][5]. 따라서 TCP에는 패킷 손실을 감지하고 복구하기 위한 손실 복구(loss recovery) 기능이 포함되어 있다. TCP 손실 복구 과정은 두 개의 기본 알고리듬인 fast retransmit과 fast recovery로 이루어져 동작하는데[5] 손실 복구 과정에서 감지되지 못한 패킷 손실은 재전송 타임아웃 후에 다시 전송된다. 재전송 타임아웃은 상당히 긴 시간일 뿐만 아니라 이후 송신원은 혼잡 윈도우(congestion window)의 크기를 초기 상태에서 다시 증가시켜야 하기 때문에 결과적으로 잦은 재전송 타임아웃의 발생은 TCP의 전반적인 성능을 크게 저하시키는 문제점이 있다.

그 동안 불필요한 재전송 타임아웃 발생 빈도를 낮추기 위한 많은 연구가 진행되어 왔다. 동일한 윈도우에서 다수 개의 패킷 손실이 발생했을 때 이를 모두 재전송에 의해 복구하지 못함으로 인해서 발생하는 문제를 해결하기 위한 대표적인 방안으로서 TCP NewReno[6]와 선택 승인 (selective acknowledgement) 옵션[3],[6]이 제안되었다. 윈도우의 크기가 작을 때에 발생한 패킷 손실은 중복 승인(duplicate acknowledgement)의 부족으로 인해서 fast transmit에 의해서 복구되지 못함에 따라 발생하는 재전송 타임아웃을 방지하기 위해서 제한 전송(limited transmit) 기법이 제안되어 IETF(Internet Engineering Task Force)에서 표준으로 인정받았다[4]. 마지막으로 이미 감지되어 재전송된 패킷이 다시 손실됨에 따라 발생하는 타임아웃을 방지하기 위한 재전송 손실 복구 알고리듬[7]이 제안되었다.

그러나 제안된 재전송 손실 복구 알고리듬의 연구[7]에서는 TCP 혼잡 윈도우의 가장 중요한 동작 원칙인 AIMD (Additive Increase Multiplicative Decrease)[5]측면에서의 평가가 이루어지지 않았다. 이 원칙에 따르면 재전송 손실이 발생한 경우에는 망의 혼잡이 그만큼 심각하다는 것을 반영하기 위해서 윈도우의 크기를 두 번 감소시킬 것을 권고하고 있다. 그러나 [7]에 제안된 알고리듬은 이를 고려하지 않기 때문에 경우에 따라서 손실된 재전송을 감지함으로써 오히려 망의 혼잡을 더 가중시키는 결과를 유발할 가능성이 있다. 따라서 본 논문에서는 이 원칙을 고려하여 재전송 알고리듬을 수정하고 이의 동작을 시뮬레이션을 통하여 검증한다.

Ⅱ. TCP 손실 복구

2.1 Fast Retransmit

TCP의 fast retransmit를 이해하기 위해서는 우선 중복 승인(duplicate acknowledgement)과 누적 승인(cumulative acknowledgement)방식에 대한 이해가 필요하다. TCP는 패킷을 수신할 때마다 수신원이 지금까지 정상적으로 수신된 가장 나중의 패킷을 송신원에게 알려주는데 이를 누적 승인 (cumulative acknowledgement)방식이라 한다. 윈도우 크기가 8인 상태에서 패킷 1-8까지의 패킷을 전송하고 이중 패킷 1이 손실된 경우를 생각해 보자. 패킷 1이 전송도중 손실되었으므로 수신원은 패킷 1이 수신되지 못한 상태에서 패킷 2를 수신하게 되고 패킷 1을 수신하지 못했으므로 송신원에 패킷 1을 받을 차례라는 승인 패킷을 전달하게 된다. 다른 패킷 3에서 8까지 수신하게 된 수신원은 역시 패킷 1을 받을 차례라는 같은 승인 패킷을 송신원에 전달하게 되는데 이를 중복 승인(duplicate acknowledgement)이라 한다[1],[5]. TCP는 세 개의 연속적인 중복 승인을 수신했을 때 해당 패킷은 손실되었다고 판단하고 이를 즉시 재전송하게 되는데 이를 fast retransmit이라 한다[5]. 이 후 윈도우의 크기와 slow start threshold (ssthresh)는 혼잡이 발생하기 전 윈도우의 크기의 반으로 설정된다.

2.2 Fast Recovery

Fast retransmit가 적용된 TCP 구현인 TCP Tahoe[6]는 패킷 손실을 fast retransmit에 의해 재전송한 후 항상 slow start부터 전송을 시작하게 된다. 그러나 인터넷의 전송 속도가 증가함에 따라 설정된 연결의 지연대역폭(bandwidth delay product)의 곱이 증가하여 fast retransmit 후에도 현재의 동기를 유지하기에는 충분한 패킷들이 전송됨에 따라 fast recovery 알고리듬이 등장하게 되었다[2].

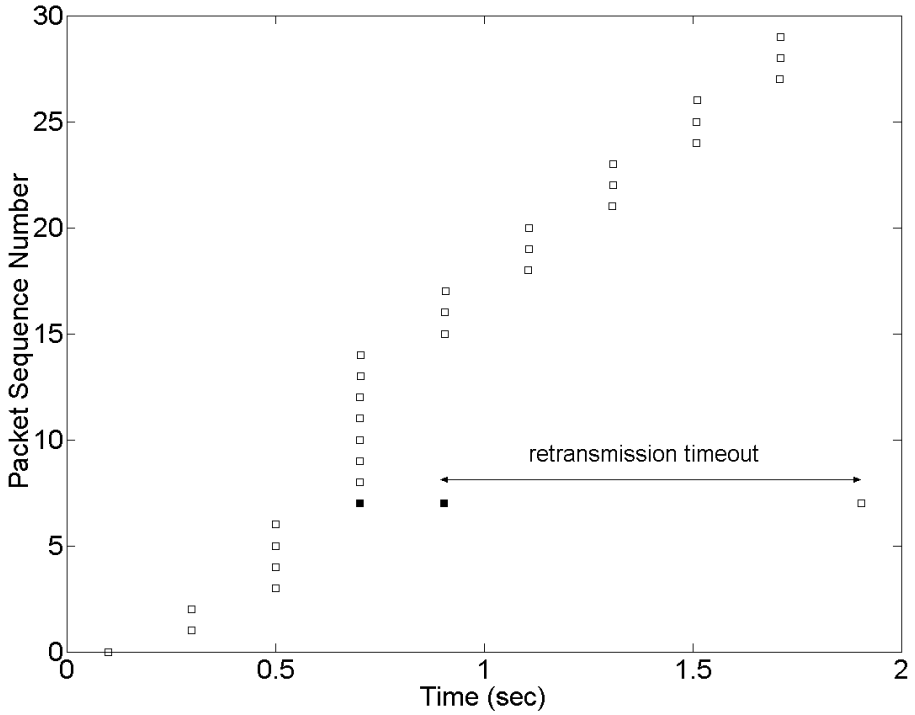

그림 1. 재전송 패킷 손실에 의한 TCP Reno의 손실 복구 과정
Fig. 1 Loss recovery behaviors for a lost retransmitted packet of TCP Reno

Fast recovery 알고리듬은 fast retransmit 후에 중복 승인 패킷이 수신될 때 마다 윈도우 크기를 1씩 증가시킨다. 이 과정에서 새로이 윈도우에 포함되는 패킷들은 전송할 수 있고 재전송이 성공적인 경우 이후 전송은 혼잡이 발생하기 전의 반으로 줄어든 윈도우로 congestion avoidance상태에서 지속한다. 초기 fast recovery 알고리즘이 포함된 TCP 구현이 가장 최근까지 널리 사용되고 있는 TCP Reno이다[6].

III. TCP 재전송 손실 감지 알고리듬

3.1 TCP 재전송 손실 감지 알고리듬의 동작

기존의 TCP에서는 재전송된 패킷이 손실될 경우 재전송 타임아웃이 만료될 때까지 이에 대한 연속적인 중복 승인만이 계속 발생할 뿐 이를 감지할 수 없다. 이 문제를 해결하기 위해서 제안된 재전송 손실 감지 알고리듬은 중복 승인의 개수에 기초하여 동작한다[7].

송신원은 fast retransmit에 의해서 패킷을 재전송할 때마다, 이 패킷이 다시 손실되지 않는 경우 이에 대해서 정상적으로 수신이 예상되는 중복 승인의 개수를 저장한다. 이 후 fast recovery동안 송신원은 손실된 패킷에 대해 발생하는 중복 승인의 개수를 세는데 만약 저장된 값보다 큰 개수의 중복승인이 수신될 때 해당 재전송의 손실을 감지하는 것이 가능하다. 특히 지연 승인 (delayed acknowledgement)를 사용하더라도 승인이 중복인 경우에는 즉시 승인을 전달하도록 되어 있기 때문에[5] 이의 사용이 중복 승인의 개수에 전혀 영향을 주지 않는다는 점을 주의해야 한다.

이의 구현을 위해서 손실된 패킷을 fast retransmit에 의해 재전송하기 전에 혼잡 윈도우의 크기와 재전송된 패킷에 대해서 예상되는 중복 승인의 개수를 저장하는 변수가 필요한데 이들을 각각 DAC_i와 CW_d라고 표현하기로 한다. 첫 번째 손실된 패킷을 재전송하는 순간 송신원은 현재 윈도우에서 몇 개의 패킷이

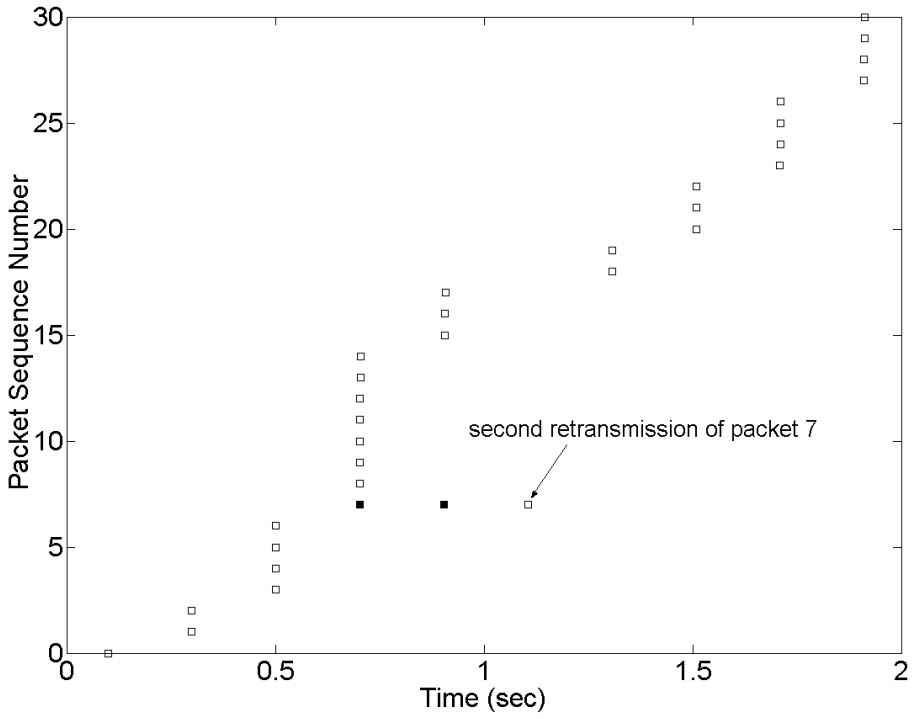

그림 2. 재전송 패킷 손실 복구 기능이 추가된 TCP Reno의 손실 복구 과정
Fig. 2 Loss recovery behaviors for a lost retransmitted packet of TCP Reno with lost retransmission detection

손실되었는지에 대한 정보가 전혀 없으므로 DAC_1은 항상 CW_{d-1}로 설정된다. 이의 동작을 설명하기 위해서 그림 1에 나타난 예를 생각해보자.

그림 1에는 TCP 송신원의 혼잡 윈도우의 상태가 8인 경우 첫 번째 패킷인 7번 패킷이 전송 중에 손실되고 이의 재전송 역시 손실된 경우 제안된 알고리듬의 동작을 나타내었다. 세 개의 중복 승인을 수신한 송신원은 패킷 7을 재전송하고 이에 대한 정상적인 수신이 예상되는 중복 승인의 개수를 7(=8-1)로 설정한다. 패킷 7을 재전송하고 이에 대한 모든 중복 승인을 수신했을 때 가용 윈도우는 현재 혼잡 윈도우의 값의 절반에 수신한 중복 승인의 개수인 7을 더한 11(=[8/2]+7)이 된다[6]. 이 때 새로이 윈도우에 포함되는 세 개의 패킷인 패킷 15, 16, 17이 전송된다.

그런데 패킷 7의 재전송이 다시 손실되었으므로 이들 세 개의 패킷들에 의해서도 역시 패킷 7에 대한 중복 승인이 발생하게 되는데 결과적으로 송신원은 총 10개의 중복 승인을 수신하게 되는 결과를 가져온다. 제안된 알고리듬을 사용하는 경우 패킷 15에 의한 8번째 중복 승인을 수신했을 때 이는 DAC_1에 저장된 값인 7보다 큰 값을 수신하므로 송신원은 즉시 패킷 7을 다시 재전송한다.

이 후 윈도우의 크기는 [5]의 권고에 따라 다시 반으로 감소하므로 2(=[4/2])로 설정된다. 이 후에 추가적인 두개의 중복 승인은 윈도우의 크기를 2만큼 증가시키므로 이 때 윈도우는 크기가 4인 패킷 7부터 11까지 포함하는 상태가 된다. 물론 이 윈도우에 포함되는 네 개의 패킷은 이미 전송이 완료되었으므로 이 시점에서 전송 가능한 패킷은 없다. 이 후 두 번째 재전송한 패킷 7이 정상적으로 전송된다면 패킷 17까지 전송이 완료되었다는 승인이 수신되고 송신원은 크기가 2인 윈도우로 congestion avoidance상태에서 전송을 지속할 수 있다.

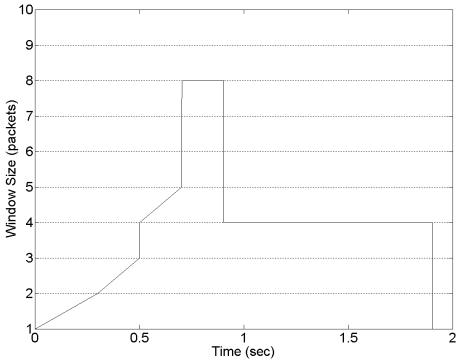

그림 3. TCP Reno의 손실 복구 과정동안의 윈도우 크기 변화
Fig. 3 Window size variations during the loss recovery process of TCP Reno

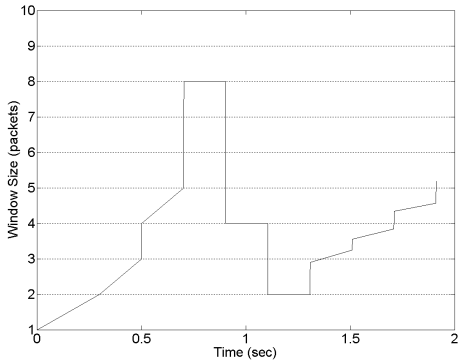

그림 4. 재전송 패킷 손실 복구 기능이 추가된 TCP Reno의 손실 복구 과정동안의 윈도우 크기 변화
Fig. 4 Window size variations during the loss recovery process of TCP Reno lost retransmission detection

3.2 제안하는 재전송 손실 복구 알고리듬의 수정

기존의 재전송 손실 복구 알고리듬은 재전송한 패킷보다 나중에 전송된 패킷들에 의해서 재전송 손실을 감지한다. 따라서 망의 혼잡 상황이 계속되어 타임아웃동안 전송을 중단해야 하는 경우에는 추가적인 패킷을 전송하지 않는다. 왜냐하면 지속적인 망의 혼잡 상태에서는 재전송된 패킷뿐만 아니라 후에 전송된 패킷들 역시 손실될 가능성이 많기 때문에 재전송 손실의 감지 자체가 불가능하기 때문이다.

그러나 본 논문에서 수정되기 이 전의 재전송 손실 복구 알고리듬은 손실된 재전송을 다시 재전송한 후 윈도우의 크기를 줄이지 않기 때문에 몇 개의 패킷을 추가적으로 전송할 수 있다. 네트워크의 혼잡이 재전송 손실이 발생 후 바로 약화된 경우에는 이 추가적인 패킷들의 전송은 크게 문제가 되지 않는다. 그러나 재전송 손실이 발생했다는 것은 망이 그만큼 혼잡 상태에 있다는 것을 의미하기 때문에 이 추가적인 패킷들로 인해서 오히려 망의 혼잡을 더욱 가중시킬 수 있는 우려가 있다. 반면 본 논문에서 제안된 수정은 윈도우의 크기를 충분히 감소시키므로 그만큼 신뢰성을 높일 수 있고 기존 알고리듬의 과다한 전송(aggressiveness)을 미연에 방지하는 것이 가능하다. 이는 [6]의 maxburst와도 같은 맥락에서 이해될 수 있다.

Ⅳ. 시뮬레이션

시뮬레이션을 위해서 앞에서 설명한 재전송 손실 복구 알고리듬과 이의 수정을 ns-2 시뮬레이터에 구현하였다[8][9]. 그림 2에는 윈도우의 크기가 8인 상태에서 전송한 8개의 패킷들 가운데 패킷 7이 손실되고 이의 재전송이 손실된 경우의 TCP Reno의 동작을 나타내었다. 패킷 7에 대한 세 개의 중복승인을 수신한 송신원은 약 1초경에 이를 fast retransmit에 의해 재전송한다. 이 후 모든 중복 승인들을 수신했을 때 윈도우의 크기는 11이 되므로 fast recovery동안 세 개의 새로운 패킷인 패킷 15, 16, 17이 전송된다. 재전송한 패킷 7이 다시 손실되었으므로 이들 세 개의 패킷들에 의해서도 패킷 7에 대한 중복 승인이 전달되고 이들에 의해 윈도우의 크기는 다시 3만큼 증가하므로 새로운 패킷 18, 19, 20이 전송된다. 이 후 송신원은 정상적인 승인이 아닌 패킷 7에 대한 중복 승인만을 연속적으로 수신하기 때문에 재전송 타임아웃이 발생할 때까지 매 라운드마다 세 개의 추가적인 패킷을 전송한다. 약 1.9초경 타임아웃이 만료되어 패킷 7이 slow start상태에서 전달되는 것을 볼 수 있다.

이 과정에서의 동작을 보다 쉽게 설명하기 위해서 혼잡 윈도우의 크기의 변화를 그림 3에 나타내었다. 초기상태에서 slow start로 윈도우가 8까지 증가한 후 fast retransmit 후에 다시 4로 설정되는 것을 볼 수

있다. 이 후 fast recovery동안 혼잡 윈도우는 계속 4를 유지하다가 타임아웃 후에는 다시 1로 설정된다.

그림 4에는 수정된 재전송 손실 복구 알고리듬이 TCP Reno에 적용된 경우 같은 예에 대한 동작을 나타내었다. 약 1.1초경 손실된 재전송이 감지되어 재전송되는 것을 볼 수 있다. 두 번째 재전송된 패킷 7에 의해서 패킷 17까지의 승인이 완료되면 이 후 송신원은 타임아웃을 기다리지 않고 congestion avoidance상태에서 전송을 지속할 수 있으므로 같은 시간 동안 그림 2에 비해서 많은 패킷들 전송할 수 있음을 알 수 있다.

그림 5는 이 동안의 윈도우의 변화를 보여주는데 패킷 7을 두 번째 재전송하고 나서 윈도우의 크기는 다시 한 번 감소하여 2가 되고 이 후 congestion avoidance 상태에서 선형적으로 증가하는 것을 볼 수 있다. 결과적으로 재전송 손실로 인한 타임아웃 후의 slow start보다 윈도우가 훨씬 빨리 증가하게 된다.

V. 결 론

본 논문에서는 재전송 손실 복구 알고리듬을 수정하고 이의 동작을 시뮬레이션을 통해서 검증하였다. 제안된 수정을 통해서 재전송 손실 복구 알고리듬의 과다한 패킷 전송을 방지할 수 있고 TCP 윈도우의 기본 동작 원리인 AIMD를 완벽하게 준수함으로써 공평성을 제공할 수 있다. 이 후에는 수정된 재전송 손실 복구 알고리듬의 구현을 통해서 실제 인터넷상에서 이의 성능을 분석하는 것이 필요하다.

참고 문헌

[1] V. Jacobson, “Congestion Avoidance and Control,” ACM SIGCOMM’88, pp. 314-329, 1988.

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

[3] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, “TCP Selective Acknowledgement Options,” RFC 2018, 1997.

[4] M. Allman, H. Balakrishnan, and S. Floyd, “Enhancing TCP’s Loss Recovery using Limited Transmit,” RFC 3042, 2001.

[5] M. Allman, V. Paxon, and W. Stevens, “TCP Congestion Control,” RFC 2581, 1999.

[6] K. Fall and S. Floyd, “Simulation-based Comparisons of Tahoe, Reno, and SACK TCP,” ACM Computer Communication Review, Vol. 26, No. 3, pp. 5-21, Jul. 1996.

[7] Beomjoon Kim and Jaiyong Lee, “Retransmission Loss Recovery by Duplicate Acknowledgement Counting,” IEEE Communications Letters Vol. 8. No. 1, pp. 69-71, Jan. 2004.

[8] 김성열, “RCR 네트워크에서 최단 경로를 위한 탐색 알고리즘,” 한국전자통신학회논문지, 5권, 5호, pp. 444-448, 2010.

[9] 서희종, “실시간 네트워크에서 개선된 분산 QoS 알고리듬,” 한국전자통신학회논문지, 7권, 1호, pp. 53-60, 2012..

감사의 글

본 연구는 지식경제부・한국산업기술진흥원 지정 계명대학교 전자화자동차부품지역혁신센터의 지원에 의한 것입니다.

저자 소개

김범준(Beom-Joon Kim)

1996년 2월 연세대학교 전자공학과 졸업 (공학사)

1998년 8월 연세대학교 대학원 전자공학과 졸업(공학석사)

2003년 8월 연세대학교 대학원 전자공학과 졸업(공학박사)

계명대학교 전자공학과 교수

※ 관심분야 : 모바일 IPTV/VoIP 서비스