논문 07-32-8-09 한국통신학회논문지 '07-8 Vol. 32 No. 8

상대전송지연시간을 이용한 TCP 세그먼트의 혼잡 손실과 무선 손실 구분 알고리즘

준회원 신 광 식*, 이 보 람*, 김 기 원*, 장 문 석*, 윤 완 오*, 종신회원 최 상 방*

The Classification of Congestion and Wireless Losses for TCP Segments Using ROTT

Kwang-Sik Shin*, Bo-Ram Lee*, Ki-Won Kim*, Mun-Suck Jang*,
Wan-Oh Yoon* ***Associate Memebers***, Sang-Bang Choi* ***Lifelong Member***

요 약

기존의 유선 네트워크 환경에서 쓰이던 TCP 프로토콜을 무선 네트워크에 사용할 경우에, 무선 손실이 발생할 경우 혼잡 제어 알고리즘을 실행하여 성능 저하 현상이 발생한다. 본 논문에서는 네트워크 상태변화를 민감하게 반영하는 손실 구분 알고리즘을 제안한다. TCP는 패킷의 이동 경로를 설정하지 않기 때문에 수신자와 송신자 사이의 병목링크(라우터)의 수를 알 수 없고, 고정된 패킷의 양을 기준으로 패킷의 손실 원인을 구분할 경우 정확성이 떨어진다. 따라서 패킷의 상대적인 전송지연시간인 ROTT(Relative One-way Trip Time)을 이용하여 네트워크 상태변화를 민감하게 반영하는 손실 구분 알고리즘을 수식적으로 전개하였다. 본 논문에서는 NS2를 이용하여 기존의 TCP Veno, Spike 알고리즘과 성능을 비교, 분석하였다. 실험 결과를 통해 제안된 알고리즘이 기존의 알고리즘에 비해 패킷의 손실 구분 오류율을 최고 45% 낮춘다는 것을 알 수 있었다. 그리고 낮은 패킷의 손실 구분 오류율로 인해 공평성(fairness)을 해치지 않으면서 패킷의 전송량이 증가한다는 것을 증명하였다.

Key Word : Wireless TCP, Classification of loss, Congestion Control, ROTT, Fairness

ABSTRACT

TCP is popular protocol for reliable data delivery in the Internet. In recent years, wireless environments with transmission errors are becoming more common. Therefore, there is significant interest in using TCP over wireless links. Previous works have shown that, unless the protocol is modified, TCP may perform poorly on paths that include a wireless link subject to transmission errors. The reason for this is the implicit assumption in TCP that all packet losses are due to congestion which causes unnecessary reduction of transmission rate when the cause of packet losses are wireless transmission errors. In this paper, we propose a new LDA that monitors the network congestion level using ROTT. And we evaluate the performance of our scheme and compare with TCP Veno, Spike scheme with NS2(Network Simulator 2). In the result of our experiment, our scheme reduces the packet loss misclassification to maximum 55% of other schemes. And the results of another simulation show that our scheme raise its transmission rate with the fairness preserved.

* 인하대학교 전자공학과 컴퓨터 구조 및 네트워크 연구실 (sangbang@inha.ac.kr)
논문번호 : KICS2007-02-056, 접수일자 : 2007년 2월 12일, 최종논문접수일자 : 2007년 7월 25일

Ⅰ. 서 론

무선 네트워크 기술의 눈부신 발달로 그 사용의 범위가 점점 더 다양해지면서 광범위한 분야에서 활용되고 있다. 이렇게 발전된 무선 네트워크는 무선망으로만 이루어지기도 하지만 보통 기존의 유선망과 연동하여 복합적인 구조를 갖는다. 하지만 무선 네트워크의 급속한 발전과 그에 따른 환경 변화에도 불구하고 기존의 유선망에서 사용하던 프로토콜들을 그대로 적용하여 사용함으로써 많은 문제점들이 드러나고 있다. 그 중 대표적인 예로 무선망에서 발생한 패킷의 손실을 네트워크의 혼잡에 의해 발생된 패킷 손실로 단정하여 생기는 문제점을 들 수 있다.

기존 유선망에서의 TCP는 패킷 손실이 발생하면 혼잡에 의한 손실로 간주하고 혼잡 제어 알고리즘에 의해 신속 재전송 및 복구(Fast Retransmit and Recovery) 방법을 사용하여 패킷의 전송량을 줄인다. 하지만 무선상에서 발생하는 패킷 손실은 네트워크의 혼잡에 의해서 발생되는 것이 아니라, 일시적인 전파의 간섭이나 핸드오프에 의해 발생하는 경우가 대부분이기 때문에 이러한 경우에 혼잡 제어 알고리즘을 사용하는 것은 불필요하게 데이터 전송량을 줄이는 결과를 초래하게 된다. 이러한 문제점은 혼잡에 의한 패킷 손실과 무선상에서의 패킷 손실을 구분함으로써 해결할 수 있다.

혼잡 손실과 무선 손실을 구분하기 위해 여러 가지 방법들이 제안되었다. 그 예로 Indirect TCP나 Snoop TCP [1][2]와 같이 TCP 계층에서 손실의 유형을 구분하는 방법과 TCP Veno[3]처럼 네트워크의 혼잡 상태를 모니터해서 손실의 유형을 구분하거나 Spike[4]처럼 수신측에서 패킷의 지연시간을 이용하여 손실의 유형을 구분하는 End-to-End 해결 방법이 있다.

본 논문에서는 TCP Veno처럼 수신측에서 네트워크의 혼잡 상태를 판단하여 손실의 유형을 구분하는 알고리즘을 제안한다. 네트워크의 상대적인 상태를 알 수 있는 Spike의 ROTT[4] 값을 이용하며, 현재 병목링크(라우터)의 버퍼의 용량과 버퍼에 쌓인 패킷의 양을 측정한다. 버퍼에 패킷이 쌓이는 속도와 가속도를 측정하여 네트워크 민감한 상태변화를 반영하는 손실 구분 알고리즘을 제안한다. 버퍼가 오버플로우 되는 시간을 측정하여 손실 발생시 버퍼가 오버플로우 되는 시간보다 현재 패킷의 ROTT가 크면 혼잡 손실로 구분하고 그렇지 않으면 무선 손실로 구분한다. 그리고 시뮬레이션을 통해 기존에 제안되었던 알고리즘들과 비교 분석한다.

본 논문의 목표는 실제 네트워크 환경에 부합하는 다중흐름에서 네트워크의 상황을 정확하게 판단하여 손실의 유형을 구분하고, 이를 통해 패킷 손실이 발생하였을 경우 적절하게 대응하도록 하는 것이다. 제안된 알고리즘의 전송률 계산은 Reno TCP[5]의 패킷 전송률 계산 방법을 사용한다. Reno TCP는 패킷 손실이 발생할 경우 현재 네트워크의 패킷 손실률에 따라 패킷 전송량을 좀 더 유연하게 변화시키기 위해 제안된 프로토콜이다. 제안된 알고리즘에 의해 구분된 무선 손실은 수신자가 송신자에게 알려주게 되고, 송신자는 무선 손실 메시지에 따라 혼잡 손실 알고리즘을 수행하지 않고, 바로 재전송한다. 이 때, 송신자는 발생된 무선 손실을 패킷 손실률 계산에 반영하지 않는다.

본 논문은 다음과 같이 구성된다. Ⅱ장에서는 본 논문의 이해를 돕기 위하여 Reno TCP에 대해 간략하게 소개하고, 패킷 손실의 유형을 구분하는 방법 및 기존의 손실 구분을 위해 행해졌던 연구들에 대해 소개한다. Ⅲ장에서는 본 논문에서 제안된 실제 네트워크 환경을 고려하여, 네트워크 상태를 측정하여 좀 더 정확하게 패킷 손실의 유형을 구분할 수 있는 방법에 대해 설명하고, 구분을 하기 위한 동작 절차에 대해 자세히 설명한다. Ⅳ장에서는 본 논문에서 제안한 손실 구분 알고리즘을 시뮬레이션을 통해 분석하고, 같은 환경에서의 다른 알고리즘들과의 성능 비교를 통해 본 알고리즘의 유용성에 대해 논의한다. 마지막으로 Ⅴ장에서는 본 논문의 결론 및 향후 연구 과제를 제시하며 본 논문을 마친다.

Ⅱ. 손실 구분과 관련된 연구

패킷 손실을 구분하는 방법은 크게 세 가지 방법으로 나눌 수 있다. 링크 계층에서의 해결 방법과 TCP 계층에서의 해결 방법 그리고 End-to-End의 해결 방법이 있다. 다음 절에서 각각의 세 가지 방법에 대해서 자세히 설명한다.

2.1 링크 계층에서의 해결 방안

링크 계층에서의 해결 방안은 TCP-aware,ARQ, FEC 등을 이용해서 기본적으로 링크의 품질을 높임으로 해서 가능해진다. 이러한 방법은 무선 링크 계층에서 지역적인 재전송을 통한 신뢰성을 제공한다.

2.1.1 FEC를 이용한 방법

FEC(Forward Error Correction)는 패킷의 손실을 복구하기 위해 전송 패킷에 부가적인 정보를 추가하여 수신자에서 판독율을 높여 BER(Bit Error Rate)을 낮추는 방법이다. 이 방법은 에러가 랜덤하게 발생하고, 채널의 특성을 잘 파악하고 있는 상황에서는 유용하지만 추가되는 에러 정정 코드로 인해 CPU의 프로세싱 타임으로 인해 전송시간이 지연되고, 링크의 상태가 양호할 때는 대역폭을 낭비하게 되는 단점을 가지고 있다. 하지만 전송 에러로 인해 손실된 패킷이 재전송 없이 복구될 수 있고, TCP의 재전송 기법과의 간섭이 없기 때문에 지연이 큰 특성을 가지는 망에서는 중요한 의미를 갖는다[6][7].

2.1.2 ARQ를 이용한 방법

ARQ는 전송된 각각의 데이터 블록들에 대한 ACK를 참조하여 ACK가 미리 정해진 일정 시간 안에 도착하지 않으면 패킷의 손실로 판단하고 재전송하는 방법이다. 이때 링크 계층의 타임아웃 시간이 너무 짧으면, ACK가 전송되는 중에 타임아웃이 발생하게 되어 전송된 데이터가 자칫 재전송되는 경우가 생길 수 있다. 이런 경우를 대비해서 각각의 ACK에는 시퀀스를 부여해 중복 수신된 데이터를 처리할 수 있도록 하였다. ARQ는 손실이 적고 전송 지연에 민감하지 않은 경우에 유용하다. 하지만 FEC와 달리 TCP의 재전송 기법과 간섭이 발생할 수 있다. 만약 링크 계층에 순서에 맞지 않는 패킷이 전송되게 되면 TCP는 중복 ACK를 전송하게 된다. TCP가 중복 ACK를 수신한 송신자는 불필요하게 혼잡 윈도우를 줄이게 되어 성능의 저하를 가져온다. 따라서 ARQ가 설계될 때 링크 계층에서의 재전송과 종단간 TCP 계층에서의 재전송과의 간섭과 같은 상호작용을 고려해야 하기 때문에 복잡도가 증가할 수밖에 없고, ARQ 사용이 모든 전송 에러를 송신자로부터 차단할 수 없다는 단점이 있다[7][8].

2.2 TCP 계층 해결 방안

TCP 계층 해결 방안은 무선망에서 발생된 패킷 손실이 유선망에 영향을 주지 않기 위해서 망을 유선과 무선으로 구분한다. 무선과 유선의 경계가 되는 곳에 위치한 기지국에서 송신자로부터 수신자로 전달되는 모든 패킷을 복사해서 저장하고 수신자에게 전송한다. 그리고 저장된 패킷은 수신자로부터 ACK를 수신하면 폐기한다. 기지국에서 손실을 감지하면 송신자를 대신해서 손실된 패킷을 재전송한다. 대표적으로 I-TCP(Indirect TCP)와 Snoop TCP가 이러한 방법을 사용한다. 이 두 방법은 패킷의 손실 유형을 정확하게 구분하지만, 기존의 TCP 구조에 변화가 필요하고 추가적인 시설의 투자가 필요하다는 단점이 있다.

2.2.1 I-TCP

그림 1에서와 같이 유선망에 고정된 송신자와 무선망에 접속된 수신자 사이의 End-to-End TCP 연결을 기지국을 중심으로 송신자에서 기지국으로, 그리고 기지국에서 수신자로 이어지는 연결로 나눈다. 송신자가 보낸 패킷은 기지국에서 먼저 수신한 후, 수신된 패킷에 대한 ACK를 송신자에게 전송하고 수신자로 패킷을 전달한다. 기지국과 수신자 사이에서는 반드시 TCP를 사용할 필요는 없고, 무선 연결에 적합한 별도의 프로토콜을 사용하기도 한다.

이 방법의 장점은 둘로 나누어진 연결이 각자 자신의 환경에 잘 맞는 프로토콜을 사용하기 때문에 무선 손실로 인한 패킷 손실에 효율적으로 대처할 수 있다. 또 기지국에서 모든 오버헤드를 처리하기 때문에 모바일 수신자는 단순화될 수 있고 송신자의 TCP/IP에는 어떤 수정도 필요하지 않다. 그리고 만약 무선 수신자가 다른 셀로 이동하게 되더라도 현재 연결에 관한 모든 정보를 기지국에서 새로운 셀의 기지국으로 전달하기 때문에 일련의 스위칭 과정들도 송신자에게는 영향을 미치지 않는다.

이 방법의 단점은 End-to-End 구조를 유지하지 않는다는데 있다. 이것은 송신자와 수신자가 하나의 연결로 직접 연결되지 않았기 때문에 수신자에게 제대로 전달이 되지 않은 패킷에 대한 ACK가 중간 노드인 기지국에서 송신자에게 전달될 수 있다.

또 기지국의 TCP를 경유하기 때문에 프로세싱 시간이 지연되고, 수신된 모든 패킷을 복사해서 가지고 있어야 하기 때문에 트래픽이 많은 경우에는 상당히 많은 양의 버퍼가 필요하다[1].

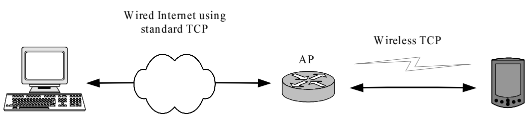


그림 1. I-TCP의 기본구조
Figure 1. The basic structure of I-TCP.

2.2.2 Snoop TCP

Snoop TCP는 기지국에 Snoop Agent 모듈을 구현하여 TCP 연결의 양방향으로 전달되는 모든 패킷을 모니터링 한다. Snoop Agent는 수신자로부터 ACK를 수신 받지 못한 패킷들을 캐쉬하는 방법을 사용한다. Snoop Agent는 손실된 패킷이 버퍼에 저장되어 있으면 이를 재전송하고 송신자로 중복 ACK를 전송하는 것을 막는다. 하지만 버퍼에 없을 경우에는 중복 ACK를 송신자에게 보내고 송신자는 이를 혼잡 손실로 판단하여 혼잡 제어 알고리즘을 시작하게 된다.

Snoop TCP의 장점은 손실된 패킷을 복구하기 위해서 기지국을 거쳐 무선 송출된 패킷에 대한 중복 ACK를 송신자에게 전송하는 것을 금지함으로써 불필요한 혼잡 제어를 방지하는 점이다. 이 방법은 기지국에 있는 라우팅 코드의 수정으로 가능하며, 송신자나 수신자의 네트워크 계층의 수정은 필요치 않다. 그리고 End-to-End 구조를 유지하고, 기존 응용프로그램과의 완전 호환성을 제공하기 때문에 응용 프로그램의 수정 없이도 TCP 연결의 성능을 상당히 향상시킬 수 있다.

단점은 기지국에서 이루어지는 재전송으로 인해 중복 ACK가 송신자에게 전송되지 않더라도 송신자가 ACK를 받지 못한 패킷에 대해 타임아웃을 발생시키는 현상은 막을 수 없다. 따라서 기지국에 의해 재전송이 성공하더라도 송신자가 타임아웃에 의해 혼잡 제어 알고리즘을 실행하게 되어 불필요한 재전송을 할 수 있다. 또한 TCP 헤더가 암호화 되거나 TCP 패킷과 ACK가 서로 다른 경로를 통해 전달될 경우에는 사용할 수 없다[2].

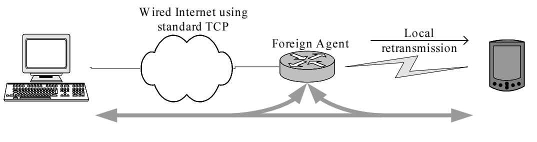


그림 2. Snoop TCP의 기본구조
Figure 2. The basic structure of Snoop TCP.

2.3 End-to-End 해결 방안

앞서 설명한 I-TCP와 Snoop TCP는 비교적 정확한 손실 구분을 하지만 여러 가지 단점에 의해 실제 네트워크에 적용하기에는 무리가 따를 수밖에 없다. 그리하여 제안된 방법이 End-to-End 해결 방법이다. 이 방법은 기존의 TCP 구조에 전혀 변화를 주지 않고, 수신측에서 패킷의 정보만을 이용하여 패킷 손실을 구분한다. 실제 I-TCP나 Snoop TCP만큼의 성능을 보여주진 않지만 비교적 정확한 예측으로 패킷 손실을 구분하여 불필요한 혼잡 제어 알고리즘이 발생하는 것을 방지한다. 이와 관련된 대표적인 알고리즘으로 Veno와 Spike가 있다. 본 논문에서는 Veno와 Spike 알고리즘을 Reno TCP에 적용하여 그 성능을 시험 하였다. Reno TCP 프로토콜은 혼잡이 발생하였을 경우 현재 네트워크의 상황에 적합한 전송률을 계산하여 패킷 전송량을 조절한다. 먼저 Reno TCP에 대해 설명하고, 다음으로 Veno와 Spike 알고리즘에 대해서 설명한다.

2.3.1 Reno TCP

Reno TCP는 혼잡제어를 위해 네트워크 상태를 측정하는 특별한 방정식을 이용한다. 방정식 기반의 혼잡제어의 목적은 이용 가능한 대역폭을 발견 이용하는 것이 아니라, 상대적으로 안정된 상태를 유지하면서 혼잡 상태에 적응하는 것에 있다.

Reno TCP는 손실 발생률(loss event rate)에 맞춰 전송률을 서서히 증가, 감소시킨다. 이 때, 손실 발생률은 최근의 패킷 손실의 상황을 반영한다. 다음으로 하나의 손실 발생에 패킷 전송량을 반으로 줄이지 않는다. 그러나 몇 개의 연속적인 손실 이벤트에는 전송률을 반으로 줄인다. 마지막으로 수신자는 임의 구간에서 패킷을 하나라도 받으면 RTT마다 적어도 한 번의 피드백을 전송하는데, 송신자가 몇 번의 RTT 이후에도 피드백을 받지 않을 경우에는 전송률을 점차 감소시키고 결국 전송을 멈춘다.

Reno TCP의 패킷 전송률은 아래의 식 1에 의해 계산되어진다. 식에서 s는 패킷의 크기이고, RTT는 패킷의 왕복시간, RTO는 재전송 타임아웃, p는 안정 상태에서의 패킷 손실 발생률이다[5].

$$T=\frac{s}{RTT\sqrt{\frac{2p}{3}}+RTO*\min\left(1.3\sqrt{\frac{3p}{8}}\right)p(1+32p^2)} \qquad (1)$$

RTT는 다음 식 2를 이용하여 구하며, RTO는 현재의 RTT의 평균값의 4 배의 수가 된다(식 3 참조). 식 2에서 RTT_{avg}는 현재의 RTT 평균, $RTT_{prev-avg}$는 이전의 RTT 평균, $RTT_{current}$는 현재 측정한 RTT이다.

$$RTT_{avg}=0.9\times RTT_{prev-avg}+0.1\times RTT_{current} \qquad (2)$$

$$RTO = 4 \times RTT_{avg} \quad (3)$$

패킷 손실 발생률 p는 지수가중이동평균(EWMA)을 이용하여 구한다. 최근 n 개의 패킷 손실을 고려하여 p를 계산한다고 가정하자. 식 4의 i_{weight}는 패킷 손실의 시간 간격 i_i에 각각의 가중치 w_i를 곱한 것의 합이다. 이 때, w_i는 보통 최근 $n/2$개의 손실에서는 1.0으로 하고 그 이후의 가중치는 0.2씩 줄어드는 값을 갖는다. 식 5의 W는 각각의 가중치 w_i의 합이다. p는 식 4, 5에 의해 구해진 W에서 i_{weight}를 나눈 값이 된다.

$$i_{weight} = \sum_{i=0}^{n-1} (i_i \times w_i) \quad (4)$$

$$W = \sum_{i=0}^{n-1} w_i \quad (5)$$

$$p = W / i_{weight} \quad (6)$$

2.3.2 Veno 알고리즘

Veno 알고리즘은 혼잡 손실은 네트워크 상태의 혼잡으로 인해 패킷 전송 시간의 지연을 동반하지만, 무선 손실은 무선상에서 없어지는 패킷이므로 전송 시간의 지연이 없다는 것에 착안하여 세워진 알고리즘이다. Veno는 TCP Vegas와 TCP Reno의 특성을 결합한 알고리즘이다. TCP Vegas는 패킷 손실이 일어나는 것을 사전에 방지하기위해 네트워크 상태를 측정하는데, 이를 이용하여 Veno는 네트워크의 상태에 따라 패킷 손실의 원인이 네트워크 혼잡 때문인지 아니면 무선링크의 랜덤 에러로 인한 손실인지를 판단한다. Veno는 아래 식의 N을 이용하여 네트워크의 혼잡여부를 판단한다.

$$Expected = cwnd/\text{BaseRTT} \quad (7)$$

$$Actual = cwnd/RTT \quad (8)$$

$$RTT = \text{BaseRTT} + N/Actual \quad (9)$$

$$N = Actual * (RTT - \text{BaseRTT}) \quad (10)$$

$cwnd$는 현재 TCP window size를 의미한다. BaseRTT는 최소 RTT(round-trip-time)이며, RTT는 현재 측정된 RTT이다. RTT>BaseRTT 일때, 병목링크(라우터)에 패킷이 축적되어있다. 큐에 축적된 양을 N이라 표기한다.

Veno는 측정된 N을 현재 연결된 네트워크가 혼잡상태인지를 결정하는데 사용한다. 패킷 손실발생시 만약 $N < \beta$라면, 패킷 손실을 무선 손실로 간주하고 Reno와는 다른 window-adjustment scheme을 사용한다. 만약 N이 β보다 클 경우, 패킷 손실은 네트워크 혼잡으로 인한 손실로 간주하고 기존의 TCP Reno의 window-adjustment scheme을 그대로 사용한다. 실험결과 $\beta = 3$일 때 가장 좋은 성능을 나타낸다.

TCP Veno의 장점은 수신측이나 중간 노드의 변화 없이 송신측의 TCP Reno만을 변경하므로 현재 인터넷의 서버측에 즉시 적용할 수 있다는 것이다. 단점은 RTT를 사용하여 네트워크 상태를 측정하였기 때문에 데이터 송신되어 거쳐가는 병목링크(라우터)의 버퍼의 쌓인 패킷 양뿐 아니라 ack가 송신측에 전달되며 거쳐가는 병목링크(라우터)의 버퍼에 쌓인 패킷의 양까지 고려되어 정확한 네트워크의 상태를 판단할 수 없다는 점이다. 그리고 TCP는 패킷의 이동 경로를 설정하지 않기 때문에 수신자와 송신자 사이의 병목링크(라우터)의 수를 알 수 없다. 따라서 Veno처럼 고정된 패킷의 양을 기준으로 패킷 손실의 원인을 구분할 경우 정확성이 떨어지게 된다.

2.3.3 Spike 알고리즘

앞서 설명한 Biaz 알고리즘은 패킷의 전송간격 시간을 측정하여 패킷 손실의 유형을 추측했던 것에 비해서, Spike 알고리즘은 상대적인 패킷 이동시간을 이용하여 패킷 손실의 유형을 추측한다. 송신자와 수신자의 시간 동기화는 사실상 거의 불가능하다. 그래서 송신자에서 수신자까지 패킷이 이동한 시간을 정확하게 측정하기란 힘든 일이다. 상대적인 이동시간을 측정하기 위해 송신자는 패킷을 보낼 때 패킷의 헤더에 타임스탬프를 같이 보내고 수신자는 이 패킷을 받은 시간을 기록하여 두 시간의 차를 이용하게 된다. 그리고 이 시간차를 ROTT (Relative One-way Trip Time)라 정의한다. 이렇게 구한 ROTT의 변화는 실제 네트워크에서의 전송시간 지연을 좀 더 정확하게 반영한다.

Spike 알고리즘은 아래 식의 $B_{spikestart}$와

$B_{spikeend}$를 이용하여 그림 3과 같이 Spike 상태를 설정하게 된다. ROTT 값이 $B_{spikestart}$보다 커지게 되면 Spike 상태로 전환이 되고, ROTT 값이 $B_{spikeend}$보다 작아질 때까지 발생되는 패킷 손실은 모두 혼잡 손실로 구분되어진다. 반면, ROTT가 $B_{spikeend}$보다 작아지면 Nonspike 상태가 되고 다시 $B_{spikestart}$보다 커질 때까지 발생되는 패킷 손실은 모두 무선 손실로 구분된다.

$$B_{spikestart} = ROTT_{\min} + \alpha \times (ROTT_{\max} - ROTT_{\min}) \tag{11}$$

$$B_{spikeend} = ROTT_{\min} + \beta \times (ROTT_{\max} - ROTT_{\min}) \tag{12}$$

$D = \alpha - \beta$라고 할 때, d의 값이 너무 작을 경우에는 Spike 상태와 Nonspike 상태를 지속적으로 오고가는 진동현상이 발생하여 좋지 않은 결과를 보여준다. 반면 D의 값이 너무 크게 되면 혼잡 손실과 무선 손실을 잘 못 추측하는 경우가 많아지게 된다. 따라서 D의 값을 결정하는 것은 이 알고리즘의 성능을 결정하는데 있어서 가장 중요한 요소로 작용한다. 마지막 노드만이 무선으로 연결되어 있을 경우에 적당한 α와 β 값을 변화시켜 그 특성에 적합한 손실 추측 알고리즘을 만들어야 하는 단점이 있다.

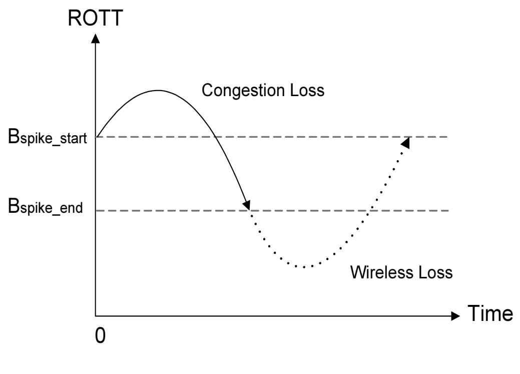


그림 3. Spike 알고리즘의 손실 구분 구간
Figure 3. Loss differentiation section of Spike.

Ⅲ. 제안된 손실 구분 알고리즘

앞장에서는 링크 계층, TCP 계층, End-to-End 패킷 손실을 구분하는 방법의 장·단점에 대해서 살펴보았다. 이번 장에서는 본 논문에서 제안한 손실 구분 알고리즘에 대하여 설명한다.

3.1 손실 구분 알고리즘

본 논문에서 제안하는 손실 구분 알고리즘의 주된 목표는 네트워크 상태 변화에 민감하게 반응하여 패킷의 손실 발생시, 손실의 원인을 정확하게 구분하는 것이다. 그리고 제안된 알고리즘은 기본적으로 End-to-End 패킷 손실 구분을 기초로 한다.

3.1.1 네트워크 상황 판단

Veno 알고리즘은 RTT를 사용하여 네트워크에 쌓인 패킷의 양을 측성한다. 하지만 네트워크의 혼잡으로 인한 패킷 손실은 송신측에서 보낸 패킷이 수신측에 도착할 때까지 거쳐간 병목링크(라우터)의 버퍼가 오버플로우 될 때 발생하기 때문에, 수신측이 송신측에 보낸 ACK가 거쳐간 병목링크(라우터)의 버퍼 상태를 반영할 경우 정확한 네트워크 상태를 측정할 수 없다. 따라서 본 논문은 패킷이 송신되어 수신될 때까지 거쳐간 병목링크(라우터)의 버퍼의 상태만을 고려하기 위해 ROTT를 사용하여 네트워크 상태를 측정하였다.

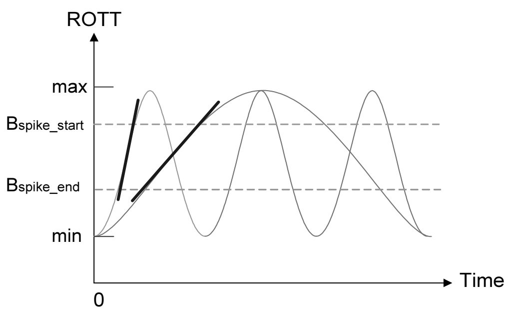


그림 4. 시간에 따른 ROTT변화량
Figure 4. ROTT variation versus time

앞에서 설명했듯이 Spike 알고리즘은 특정 ROTT($B_{spikestart}$) 보다 긴 시간으로 패킷이 전송될 경우, 패킷손실 발생시 손실의 원인을 네트워크 혼잡 때문으로 간주한다. 그러나 동일한 ROTT라도 ROTT가 급격하게 커지는 추세라면 혼잡이 발생할 확률이 높아진다. 그림 4는 시간에 따른 ROTT의 변화량을 나타낸다. 두 그래프는 동일한 ROTT에서 네트워크의 순간 변화율을 직선으로 나타낸다. 안정상태의 ROTT라도 ROTT가 급격하게 커지는 추세라면 혼잡이 발생할 확률이 높아진다. 그러나 ROTT

가 완만하게 변화하면 손실 발생시 무선으로 인한 패킷 손실로 판단할 수 있다. 본 논문에선 ROTT의 변동을 이용하여 버퍼의 상태변화를 수식으로 산출하여 급변하는 네트워크 상황을 정확히 반영하는 손실 구분 알고리즘을 제안한다.

표 1. 매개변수의 의미
Table 1. meaning of variables

Symbol	Meaning
$ROTT$	Relative One-Way Trip Time
N_i	큐에 축적된 양
S_i	큐의 용량
t_i	패킷의 도착 시간
V_i	큐에 패킷이 축적되는 속도
a_i	큐에 패킷이 축적되는 가속도

송신측에서 패킷의 전송시간 간격이 일정하다고 가정한다. 수신측의 패킷 도착시간 간격이 송신측의 패킷 전송시간 간격과 일치한다고 가정한다. 버퍼의 용량이 S이고 버퍼에 축적된 패킷의 양이 N일 경우, i번째 패킷이 수신측에 도착한다면 현재 큐에 패킷이 축적되는 속도 V와 가속도 a는 다음과 같이 계산할 수 있다.

$$V_i = (N_i - N_{i-1})/(t_i - t_{i-1}) \tag{12}$$

$$a_i = (V_i - V_{i-1})/(t_i - t_{i-1}) \tag{13}$$

여기서 N_i은 식 (7)~(10)에서 RTT 대신 ROTT를 사용하여 다음의 식으로 유도할 수 있다. BaseROTT 는 버퍼가 비었을 때의 Relative One-Way Trip Time이다.

$$N_i = \frac{cwnd}{ROTT_i} * (ROTT_i - \text{Base}\,ROTT) \tag{14}$$

버퍼의 용량이 S라고 할 때, 버퍼가 overflow될 때까지 걸리는 시간 T는 다음과 같다. 버퍼는 FIFO 정책을 따른다.

$$T = \frac{-V_i + \sqrt{V_i^2 + 2a_i(S - N_i)}}{a_i} \tag{15}$$

3.1.2 패킷 손실 구분

앞에서 얻은 T는 ROTT의 순간 변화율을 고려하여 네트워크의 상태를 측정한 것이다. 따라서 T에 따라 손실의 원인을 판단하여 보다 정확한 패킷 손실 구분을 할 수 있다. 손실 발생시 네트워크의 버퍼가 오버플로우에 이르는 시간(T)을 이용하여 보다 정확한 손실 구분을 하기위해, 우선 수신자는 패킷을 받을 때마다 앞서 설명한 Spike 알고리즘의 ROTT 값을 측정한다. 이때 측정된 ROTT를 식 (14)에 대입해 현재 버퍼에 쌓인 패킷의 양 N을 구한다. 여기서 패킷이 거쳐간 네트워크의 버퍼의 용량을 알 수 없기 때문에 현재까지 측정한 N중 가장 큰 값을 버퍼의 용량 S로 삼는다. 또한 송신자는 패킷 손실이 일어날 경우 T를 이용해 현재 네트워크의 버퍼에 패킷이 쌓이는 정도를 판단하여 손실의 원인을 구분한다.

$$T_i \geq \alpha ROTT_i \tag{16}$$

위 식 (16)과 같이 Ti가 α ROTTi보다 크거나 같을 경우, 버퍼가 오버플로우에 이르기까진 시간이 길다고 판단한다. 즉, 네트워크상태가 혼잡하지 않으므로 손실 발생시 ACK에 무선 손실로 표기하여 송신자에게 보낸다. Ti가 αROTTi보다 작을 경우, 네트워크 상태가 급격히 나빠지고 있다고 판단하여 손실 발생시 유선 손실로 간주하여 송신자에게 유선 손실을 표기한 ACK를 보낸다.

패킷 손실 발생시 혼잡 손실일 경우 TCP Reno의 혼잡 손실 알고리즘을 따른다. 무선 손실일 경우에는 혼잡 손실 알고리즘을 실행하지 않고 손실된 패킷만을 재전송하게 된다. α는 네트워크 구성에 따라 달라진다. Ti가 허수일 경우, 시간에 관계없이 버퍼가 overflow에 이르지 못하므로 패킷 손실 발생시 무선 손실로 간주한다.

3.2 패킷의 손실의 처리

제안된 알고리즘에 의해 수신자는 네트워크의 상태를 측정한다. 손실 발생시 최근에 들어온 패킷의 ROTT와 네트워크의 버퍼가 오버플로우에 이르는 시간을 비교하여 패킷 손실의 유형을 구분한다. ACK의 TCP 헤더에 해당 패킷 손실이 혼잡 손실인지, 무선 손실인지를 기록하여 송신자에게 보낸다. 그리고 해당 ACK를 받은 송신자는 패킷 손실 유형에 따라 서로 다른 작업을 수행하게 된다.

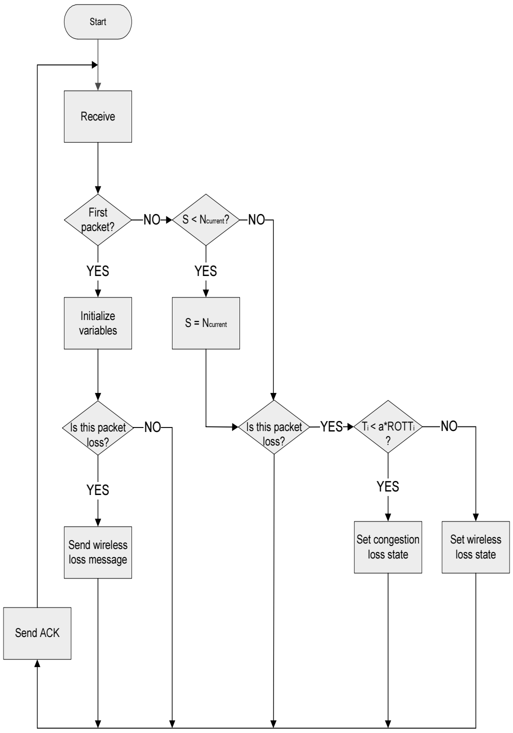


그림 5. 패킷 손실 구분 흐름도
Figure 5. Flow chart of differentiating packet loss.

3.2.1 구분된 패킷 손실에 대한 ACK

무선 패킷의 손실이 발생할 경우에는 무선 손실의 여부를 그림 6의 TCP 헤더에 사용하지 않는 6비트 중 1비트를 이용하여 송신자에게 알려준다. 무선 손실을 알려주는 비트를 1로 설정하여 보내면 해당 ACK를 받은 송신자는 그 패킷이 무선 손실되었다고 간주한다. 반대로 0으로 설정된 ACK를 받게 되면 해당 패킷이 혼잡 손실이 되었다고 간주한다.

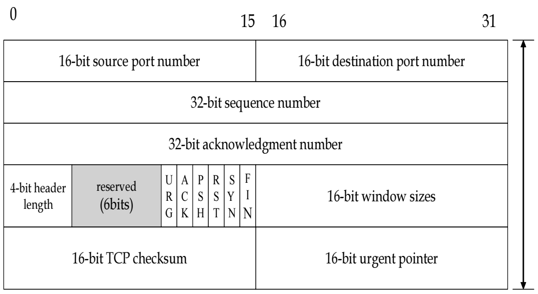


그림 6. TCP 헤더.
Figure 6. TCP header.

3.2.2 혼잡 손실 발생시

발생된 손실이 혼잡 손실로 구분이 되면, 우선적으로 손실된 패킷을 수신자에게 재전송한다. 그리고 식 (6)의 p를 현재 발생한 패킷 손실까지 고려하여 계산한다. 그리고 구해진 p를 이용하여 현재의 패킷 전송률 T를 계산하고, 실제 패킷 전송률을 T 만큼 낮추게 된다.

3.2.3 무선 손실 발생시

발생된 손실이 무선 손실로 구분될 경우 혼잡 손실과 다르게 동작한다. 먼저 손실된 패킷을 수신자에게 재전송한다. 그리고 혼잡 손실과는 달리 현재 발생한 패킷 손실을 패킷 손실률 p를 계산하는 데 고려하지 않는다. 즉, 패킷 손실로 구분하지 않으며, 패킷 손실률을 변화시키지 않음으로써 패킷의 전송률을 낮추지 않고 유지한다.

3.3 알고리즘에 대한 의사코드

이번 절에서는 패킷 손실 구분 알고리즘과 관련된 수신자와 송신자에서의 의사코드를 기술한다. 표 2는 의사코드에 사용된 기호의 의미를 기술한 것이다.

표 2. 제안된 알고리즘의 의사코드에 사용된 기호의 의미
Table 2. The mean of notations using in pseudocode at the suggested algorithm

Symbol	Meaning
tsbefore	이전 패킷의 도착시간
tsnow	현재 패킷의 도착시간
snd_tsbefore	이전 패킷의 타임스탬프
snd_tsnow	현재 패킷의 타임스탬프
base_rott	최소 ROTT
rottcurrent	현재 ROTT
s	버퍼의 크기
ncurrent	현재 버퍼에 쌓인 패킷의 양
t	버퍼가 오퍼플로어 될 때까지 걸리는 시간

그림 7은 수신측에서 패킷 손실을 구분하는 일련의 동작, 그리고 구분된 손실에 따른 동작에 관한 의사코드이다.

```
In Receiver
 Received packet from the sender
  //initialize variables at first received packet
  if(seq_num = 0)
   s ← 0;
   ts_before ← current time;
   snd_ts_before ← timestamp of current packet;
   base_rott ← ts_before - snd_ts_before;
  //initialize base_rott
  //compute variables
  snd_ts_now ← timestamp of current packet;
  rott_current ← ts_now - snd_ts_now;
 if(base_rott > rott_current)
  base_rott ← rott_current;    //renew base rott
  n_current ← (rott_current - base_rott)*cwnd /
  base_rott;
 if(s < n_current)
  s ← n_current;    //renew base s
 //v_current : the variation rate of the packet's
             amount in buffer as the time
 //t : the time that until the buffer has
       overflowing
 v_current ← (n_current - n_before)/(ts_now - ts_before);
 a_current ← (v_current - v_before)/(ts_now - ts_before);
 t ← (-v_current + (v_current^2 + 2*a_current*(s -
 n_current))^(1/2))/a_current;
 //update
 n_before ← n_current;
 v_before ← v_current;
 a_before ← a_current;
 snd_ts_before ← snd_ts_now;
 ts_now ← ts_before;
Occur packet loss
if(t < alpha*rott_current)
 send Congestion_ACK message to sender;
else
 send WirelessLoss_ACK message to sender;
```

그림 7. 수신자의 의사 코드.
Figure 7. Pseudocode of Receiver.

그림 8은 수신자로부터 패킷 손실 메시지를 받았을 경우에 대한 송신자의 의사코드이다.

```
In sender
 Receive Congestion_ACK message
 from MH
  start congestion avoidance algorithm;
 Receive WirelessLoss_ACK message
 from MH
  send packet_lost;
```

그림 8. 송신자의 의사 코드.
Figure 8. Pseudocode of sender.

Ⅳ. 시뮬레이션 및 성능분석

4.1 시뮬레이션 환경

제안된 손실 구분 알고리즘의 성능을 측정하기 위해서 NS2(Network Simulator 2)를 이용하여 다양한 환경을 설정하여 시뮬레이션을 실행하였다. 다음은 NS2를 사용하여 구현한 네트워크 구조 및 이에 적용한 환경 변수들에 대한 구체적인 내용이다.

4.1.1 네트워크 구조

본 논문에서는 제안된 패킷 손실 구분 알고리즘의 성능을 측정하기 위해 그림 9와 같이 마지막 링크를 제외한 연결은 유선으로 하고, 마지막 링크만을 무선망으로 구성하였으며, 이를 NS2(Network Simulator 2)를 이용하여 구현하였다. 송신자와 수신자는 각각 Reno TCP를 사용한다. 제안된 알고리즘에 의해 패킷의 손실이 무선 손실로 구분되어질 경우, 패킷 손실률에 영향을 주지 않도록 설계하였다. 그림 9에서 각각의 유·무선 노드수인 N을 변화시키며 네트워크에 연결된 흐름의 수를 변화시켰다. 그리고 각각의 송신자가 하나의 공통된 링크를 통해서 서로 경쟁하며 데이터를 보내는 구조로써 혼잡에 의한 손실과 무선에 의한 손실이 발생할 수 있는 환경을 구성하였다.

4.1.2 네트워크 파라미터

기본적으로 무선 링크는 802.11 무선랜을 사용한다. 802.11 무선랜을 사용하는 CDMA-2000 표준의 패킷 크기는 382byte이다. 따라서 시뮬레이션에서의 패킷 크기는 382byte로 정하였다. 송신자와 수신자의 수는 각각 N으로 하였다. 그리고 노드의 수 N을 2, 4, 6, 8로 변화시키며 시뮬레이션 하였다. 각 유선노드와 라우터는 5메가바이트 밴드위스(bandwidth)와 2ms의 지연(delay)을 가진 이중통신 링크(duplex link)방식으로 연결하였다. 무선 링크의 큐잉 정책은

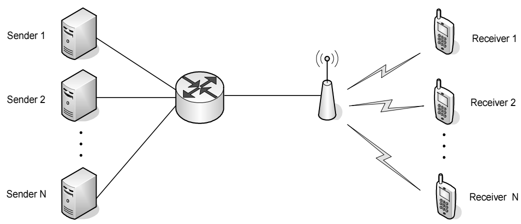


그림 9. 시뮬레이션을 위한 네트워크 구조
Figure 9. the architecture of network for the simulation.

표 3. 네트워크 요소
Table 3. Network parameter

Parameter		Value
Packet Size	Wired	381
	Wireless	381
Number of flows (N)		2, 4, 6, 8
Wireless LAN		802.11 wireless LAN
Queueing Policy		DropTail
Type		FTP
Period		250s
Trial		10

DropTail 방법을 사용하였다. 데이터 전송은 FTP를 사용하여 지속적으로 데이터를 보내는 구조로 설정하였다. 한 번의 시뮬레이션은 50초에 FTP 데이터 전송을 시작하여 250초에 완료하도록 만들었다. 약 200초 시간동안 각각의 송신자가 수신자에게로 FTP를 이용한 파일 전송을 한다.

4.2 시뮬레이션 결과 및 분석

시뮬레이션은 제안된 알고리즘과 기존에 사용되던 Reno TCP와 Veno, Spike 알고리즘을 사용하였을 경우를 비교 설명한다. 패킷 손실 모형은 NS2의 Uniform error model을 사용하였고, 에러율을 변화시키며 데이터를 구하였다. 서로 다른 난수를 생성하여 시뮬레이션을 총 10회 수행한 후, 얻은 데이터의 평균을 사용하여 비교 분석한다. 그리고 송신측과 수신측의 노드수를 2, 4, 6, 8로 각각 바꾸면서 하나의 링크에 경쟁하는 흐름의 수를 증가시키면서 각각의 프로토콜에 대한 성능을 측정하였다.

4.2.1 데이터 전송량 비교

그림 10은 무선 손실율이 0.01인 네트워크에서 TCP Reno 알고리즘의 데이터 전송량을 100이라 했을 때, 다른 알고리즘들의 상대적인 전송량을 시뮬레이션을 통해 구한 것이다. 흐름의 수가 증가 할수록 제안된 알고리즘의 성능이 다른 알고리즘에 비해서 좋아진다는 것을 알 수 있다. 이처럼 전송된 데이터양이 증가한 것은 제안된 알고리즘이 다른 알고리즘들에 비해서 비교적 패킷 손실 구분을 정확하게 하였기 때문이다. 손실 구분이 정확해지면 앞서 말했던 불필요한 패킷 전송률의 제한을 막게 되고, 이를 통해 네트워크의 대역폭을 충분히 사용하여 패킷을 전송할 수 있게 된다. 시뮬레이션을 이용한 데이터 전송량의 측정을 통해 제안된 알고리즘의 다중 흐름에서의 패킷 손실 구분에 대한 성능이 향상되었다는 것을 알 수 있다.

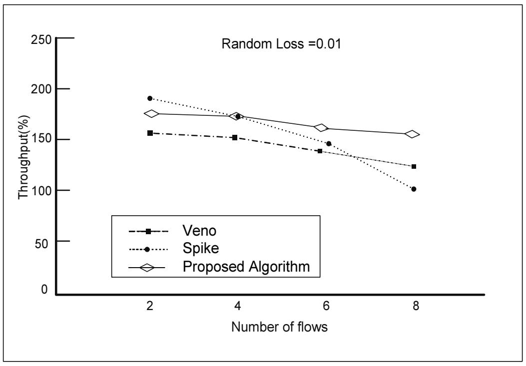


그림 10. TCP Reno 알고리즘의 전송량이 100% 라고 할 때, 다른 알고리즘의 데이터 전송량.
Figure 10. If throughput of TCP Reno algorithm is 100%, show the throughput of the other algorithms.

4.2.2 다중 흐름에서 Reno와 제안된 알고리즘의 성능 비교

그림 11과 그림 12는 다중 흐름에서 제안된 알고리즘의 성능을 알아보고자 시행하였다. 각각 5개인 송신자와 수신자가 모두 통신을 하는 네트워크 환경에서 무선 링크의 에러율을 0.0, 0.01, 0.1로 변화시켰을 때 시간에 따른 cwnd의 변화를 관찰하였다. 혼잡손실과 무선 손실이 함께 나타나는 상황을 설정하기위해 라우터와 기지국(base station)간의 밴드위스(bandwidth) 2메가바이트(Mb)로 제한하여 병목링크가 형성되도록 하였다.

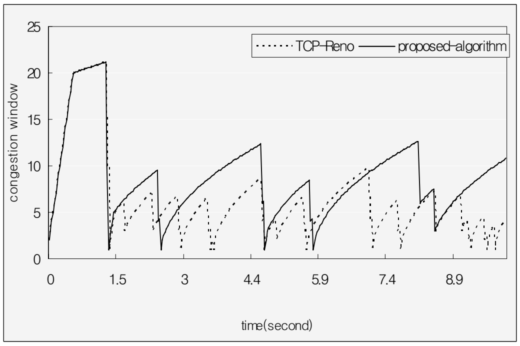


그림 11. TCP Reno와 제안된 알고리즘의 혼잡 윈도우 크기 변화 비교(에러율 0.1%).
Figure 11. the congestion window variation of TCP-Reno and the proposed algorithm.(error rate 0.1%).

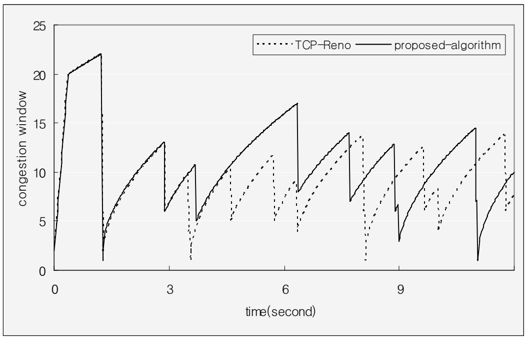


그림 12. TCP Reno와 제안된 알고리즘의 혼잡 윈도우 크기 변화 비교(에러율 0.1%).
Figure 11. the congestion window variation of TCP-Reno and the proposed algorithm.(error rate 0.1%).

4.2.3 α 값에 따른 데이터 전송량

α 값은 제안된 알고리즘에서 무선 손실과 유선 손실을 구분하는 T의 임계값을 결정하는 중요한 요소이다. α는 ROTT의 값을 반영하는 정도를 결정하여 T가 나타내는 네트워크 상황이 혼잡인지, 아니면 안정인지를 판단하는 기준이 된다. α 값은 네트워크 구조에 따라 달라진다. 버퍼에 크기와 ROTT의 관계를 고려하여 실험을 통해 α값을 구한다. α 너무 크다면 혼잡 손실이 아닌 경우에 혼잡 손실로 구분하게 되는 오류가 발생할 수 있다. 반면에 α 값이 너무 작다면 무선 손실이 아닌 경우에 무선 손실로 구분하게 된다.

α 값을 0.7, 0.8, 0.9로 바꿔가며 시뮬레이션을 하였다. 그림 13에서 볼 수 있듯이 α가 0.9일 때 가장 좋은 성능을 보여주었다. 물론 네트워크의 상황에 따라 그 성능은 다소 차이가 날 것이다.

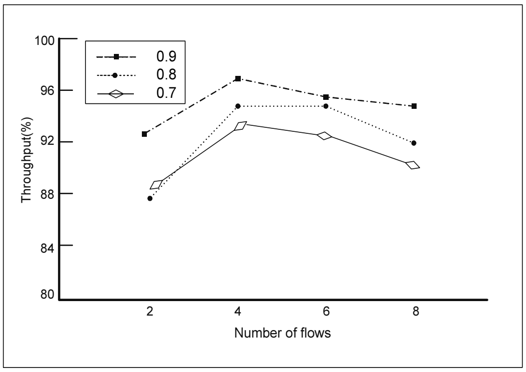


그림 13. 플로우 수에 따른 데이터 전송량.
Figure 13. Throughput by number of flows

4.2.4 혼잡 손실의 구분 오류

혼잡 손실이 발생하였으나 이를 무선 손실로 구분하는 오류가 발생한 확률을 시뮬레이션을 통해 측정하였다. 이러한 오류는 혼잡 제어 알고리즘이 실행되지 않고 혼잡 상황임에도 불구하고 지속적으로 패킷을 보내기 때문에 패킷 손실률이 높아지고, 수신자가 받는 데이터의 양도 줄어들게 된다. 하지만 시뮬레이션 결과를 나타내는 그림 14를 보면, 혼잡 손실의 구분 오류가 전체적으로 상당히 낮다는 것을 알 수 있다. 대부분이 1% 미만의 확률을 갖기 때문에 실질적으로 데이터 전송량에 큰 영향을 미치지 않다는 것을 알 수 있다.

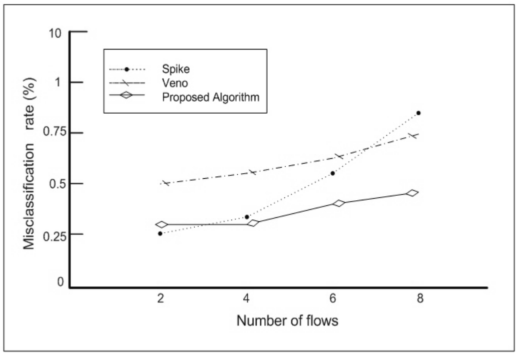


그림 14. 혼잡 손실 구분 오류율.
Figure 14. Misclassification rate for congestion loss.

4.2.5 무선 손실의 구분 오류

무선 손실의 구분 오류는 혼잡 상황이 아닌 상태에서의 패킷 손실을 혼잡 손실로 간주하여 패킷의 전송량을 불필요하게 줄이는 결과를 야기 시킨다. 그림 15에서 볼 수 있듯이, 시뮬레이션 결과 혼잡 손실 구분 오류가 전체적으로 1% 미만으로 작았다는 점에 비해 무선 손실의 구분 오류를 측정한 이번 시뮬레이션에서는 상당히 높은 손실 구분 오류율을 보여준다. 특히 Veno와 Spik 알고리즘의 경우에는 링크에 연결되어 있는 흐름의 수가 증가함에 따라 무선 손실 구분 오류도 큰 폭으로 증가하는 비슷한 양상을 나타낸다. 반면에 제안된 알고리즘은 혼잡 손실의 구분 오류보단 높은 손실 구분 오류를 보여주긴 하였지만 기존의 알고리즘에 비해 비교적 낮은 오류율을 보여준다. 제안된 알고리즘은 Veno에 비해 최고 45%, Spike에 비해 최고 40% 손실 구분 오류율을 감소시키는 결과를 얻었다.

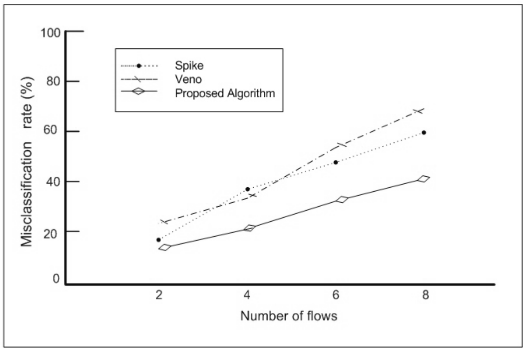


그림 15. 무선 손실 구분 오류율.
Figure 15. Misclassification rate for wireless loss.

V. 결 론

본 논문은 유무선망을 갖는 복합적인 네트워크 환경에서 패킷 손실을 무선 손실과 혼잡 손실로 구분하기 위한 알고리즘을 제안하였고, 시뮬레이션을 통해 기존의 알고리즘들과 성능을 비교하였다. 특히 하나의 링크에 여러 개의 흐름이 있는 다중 흐름의 구조에서 성능을 평가하기 위해 ns-2를 이용해 흐름의 수를 증가시키면서 시뮬레이션을 하였다.

제안된 패킷 손실 구분 알고리즘은 패킷 교환을 수행하며 얻을 수 있는 제한된 정보를 이용하여 정확하게 손실 구분을 하기 위하여, 패킷의 전송시간뿐만 아니라 네트워크의 상태변화도 고려하였다. 이를 통해 기존의 알고리즘과 비교하였을 때, 다중 흐름에도 좋은 성능을 보여준다. 기존에 널리 쓰이고 있는 TCP Reno와 함께 네트워크를 구성하여도 Reno의 패킷 전송도를 해치지 않으면서 정확한 네트워크 상황판단을 통한 손실구분으로 인해 TCP Reno보다 좋은 성능을 나타낸다. 제안된 알고리즘은 특히 무선 손실 구분에서 Veno에 비해 최고 45%, Spike에 비해 최고 40% 손실 구분 오류율을 감소시키는 결과를 얻었다. 손실 구분이 타 알고리즘에 비해 비교적 정확하기 때문에 무선 손실과 혼잡 손실이 함께 나타나는 다중흐름 속에서도 비교적 좋은 성능을 나타내었다.

본 논문에서 제안된 알고리즘은 네트워크의 상태변화를 나타내는 Ti를 패킷의 혼잡 손실과 무선 손실을 구분하기위해 반영하는 정도를 결정하는 α값에 영향을 받는다. 서로 다른 네트워크 환경에서는 α값에 따라 서로 다른 손실 구분의 정확도가 달라진다. 여러 네트워크의 환경에서도 적합한 알고리즘이 되기 위해서는 네트워크 환경에 맞게 α값을 조절하여야 한다.

참 고 문 헌

〔1〕 A. Bakre and B. Badrinath, "I-TCP: Indirect TCP for mobile hosts", *in Proc. 15th Int. Conf. Distributed Computing Systems (ICDCS)*, Vancouver, BC, Canada, May 1995.

〔2〕 H.Balakrishman, V. N. Padmanabhan, S. Seshan, and R. H, Katz. "Improving TCP/IP Performance over Wireless Networks", *in Proc. 1st ACM Int'l Conf. on Mobile Computing and Networking*, Nov, 1995.

〔3〕 C. P. Fu, S. C. Liew, "TCP Veno: TCP Enhancement for Transmission over Wireless Access Networks", *IEEE (JSAC) Journal of Selected Areas in Communications*, pp. 216-228, Feb 2003.

〔4〕 Y.Tobe, Y. Tamura, A. Molano, S. Ghosh, and H. Tokuda, "Achieving moderate fairness for UDP flows by path-status classification", *in Proc. 25th Annu. IEEE Conf. Local Computer (LCN 2000)*, Tampa, FL, pp. 252-261, Nov. 2000.

〔5〕 S. Floyd, M. Handly, T. Padhye, and J. Widmer, "Equation=based congestion control for unicast application", *in Proc. ACM SIGCOMM*, Stockholm, Sweden, pp. 43-56, Aug. 2000.

〔6〕 H. Balakrishman, V. N. Padmanabhan, S. Seshan, and R. H. Katz, "A Comparison of Mechani는 of Improving TCP Performance over Wireless Links", *IEEE/ACM Transactions On Networking, Vol. 5, No 6*, pp. 756-769, Dec. 1997.

〔7〕 C. Barakat, E. Altman, and W. Dabbous, "On TCP Performance in a Heterogeneous Network: A Survey", *IEEE Commun Mag*, pp. 40-46, Jan. 2000.

〔8〕 Nachiket Deshpande, "TCP Extensions for Wireless Networks", http://www.cis.ohio-state.edu/~jain/ cis788-99/tcp_wireless.

신 광 식 (Kwang-Sik Shin) 준회원

2001년 2월 : 인하대학교 전자공학과 졸업

2003년 2월 : 인하대학교 전자공학과 석사

2003년 3월~현재 : 인하대학교 전자공학과 박사 과정

<관심분야> 멀티미디어 데이터 통신, 컴퓨터 네트워크, 무선 통신, 시스템 프로그래밍

이 보 람 (Bo-Ram Lee) 준회원

2003년 2월 : 인하대학교 전자공학과 졸업

2006년 2월 : 인하대학교 전자공학과 석사

2006년~현재 : LG 전자 (주), DA 사업부 주임연구원

<관심분야> 컴퓨터 네트워크, 무선 통신, 네트워크 프로토콜, 홈 네트워크

김 기 원 (Ki-Won Kim) 준회원

2004년 2월 : 인하대학교 전자공학과 졸업

2006년 8월 : 인하대학교 전자공학과 석사

2005년~현재 : 파이오링크(주), 연구원

<관심분야> 컴퓨터 네트워크, wireless TCP, 멀티미디어 데이터 통신

장 문 석 (Mun-Suck Jang) 준회원

1997년 8월 : 건양대학교 컴퓨터공학과 졸업

2000년 2월 : 인하대학교 전자공학과 석사

2005년 3월~현재 : 인하대학교 전자공학과 박사 과정

<관심분야> 센서네트워크, 컴퓨터구조, Fault-tolerant computing, Parallel processing

윤 완 오 (Wan-Oh Yoon) 준회원

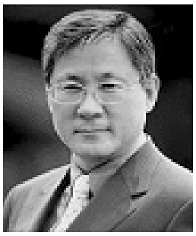
2000년 2월 : 경기대학교 전자공학과 졸업

2002년 2월 : 인하대학교 전자공학과 석사

2002년 3월~현재 : 인하대학교 전자공학과 박사 과정

<관심분야> 분산 처리 시스템, 병렬프로그래밍, 컴퓨터 아키텍처

최 상 방 (Sang-Bang Choi) 종신회원



1981년 2월 : 한양대학교 전자공학과 졸업

1981년~1986년 : LG 정보통신(주)

1988년 3월 : University of Washington 석사

1990년 8월 : University of Washington 박사

1991년~현재 : 인하대학교 전자공학과 교수

<관심분야> 컴퓨터 구조, 컴퓨터 네트워크, 무선 통신, 병렬 및 분산 처리 시스템, antFault-tolerant computing