라우터에서의 동적인 혼잡 제어를 위한 새로운 큐 관리 알고리즘

(Modified Random Early Defection Algorithm for the Dynamic Congestion Control in Routers)

구 자 헌 † 송 병 훈 † 정 광 수 †† 오 승 준 ††

(Jahon Koo) (Byunghun Song) (Kwangsue Chung) (Seoungjun Oh)

요 약 현재의 인터넷 라우터는 Drop tail 방법으로 패킷을 처리한다. 따라서 네트워크 트래픽의 지수적인 증가로 인한 혼잡상황 때문에 많은 패킷이 손실 될 수 있다. 이러한 문제를 해결하기 위해 IETF(Internet Engineering Task Force)에서는 RED(Random Early Detection)와 같은 능동적인 큐 관리 알고리즘을 제시하였다. 이러한 알고리즘은 라우터나 게이트웨이에서 발생할 수 있는 네트워크 혼잡상황을 개선하여 전체적인 패킷 손실을 줄여 줄 수 있다. 그러나 RED 알고리즘의 경우 큐 크기의 변화 정보로 혼잡상황을 판단하고 제어하기 때문에 동적으로 변화하는 현재의 인터넷 트래픽 대한 혼잡상황 제어에 있어서 개선의 필요성이 있다.

본 논문에서는 이러한 RED 알고리즘의 문제를 보완하기 위해서 기존의 RED 알고리즘을 개선한 MRED(Modified Random Early Detection) 알고리즘을 제안했다. 제안한 알고리즘은 간단하게 폐기(drop) 확률을 구하는 RED에 비하여 휴리스틱(heuristic)한 방법을 적용하여 보다 동적으로 폐기 확률 값을 계산한다. MRED 알고리즘의 성능을 검증하기 위해서 실험을 통하여 기존의 큐 관리 방법과 성능을 비교하였고 리눅스 커널에 MRED를 구현하여 성능을 분석하였다.

Abstract In order to reduce the increasing packet loss rates caused by an exponential increase in network traffic, the IETF(Internet Engineering Task Force) is considering the deployment of active queue management techniques such as RED(Random Early Detection). While active queue management in routers and gateways can potentially reduce total packet loss rates in the Internet, this paper has demonstrated the inherent weakness of current techniques and shows that they are ineffective in preventing high loss rates. The inherent problem with these queue management algorithms is that they all use queue lengths as the indicator of the severity of congestion.

In this paper, in order to solve this problem, a new active queue management algorithm called MRED(Modified Random Early Detection) is proposed. MRED computes the packet drop probability based on our heuristic method rather than the simple method used in RED. Using simulation, MRED is shown to perform better than existing queue management schemes. To analyze the performance, we also measure throughput of traffics under the FIFO control, and compared the performance with that of this MRED system.

· 본 연구는 한국과학재단 장기기초연구(97-01-00-12-01-5)와 광운대학교 교내학술연구비의 지원에 의해 수행되었음.

† 학생회원 : 광운대학교 전자통신공학과
jhkoo@adams.kwangwoon.ac.kr
byungh@adams.kwangwoon.ac.kr
†† 종신회원 : 광운대학교 전자공학부 교수
kchung@daisy.kwangwoon.ac.kr
sjoh@media.kwangwoon.ac.kr

논문접수 : 2000년 9월 8일
심사완료 : 2001년 8월 2일

1. 서 론

현재의 인터넷 프로토콜 구조는 비연결 지향적인 (connectionless) 특성을 지닌 IP 프로토콜을 기반으로 하는 종단간 패킷 전달 서비스로 구성되어져 있다. 이러한 비연결 지향적인 네트워크 환경의 이점은 유연성(flexibility)과 견고성(robustness)에 있는 반면 트래픽 제어가 어렵다는 단점이 있다. 그러므로 네트워크의 과도

한 혼잡 상황에서도 좋은 서비스를 제공하기 위해서는 특별히 주위 깊은 네트워크 설계가 요구된다. 만약 부주위하게 설계할 경우 동적인 패킷 전달에 대하여 서비스 악화를 초래 할 수 있기 때문이다. 기술적으로 이런 현상을 혼잡 악화(congestion collapse)라 하며 인터넷이 처음 성장하게된 1980년대 중반부터 이를 해결하기 위한 연구가 진행되고 있다[1].

최근 혼잡상황을 개선하려는 연구들은 크게 TCP와 같은 프로토콜 레벨과 라우터와 같은 네트워크 레벨로 구분되어 연구되어지고 있다. TCP와 같은 전송 프로토콜을 이용한 혼잡제어 방법의 경우 네트워크 혼잡상황을 해결하기 위해서 윈도우 기반의 flow 제어를 사용하게된다. 하지만, 이 방법으로는 의도적이거나 동적인 네트워크 혼잡 상황을 근본적으로 해결하지는 못하는 한계가 있다[2,3].

그러나, 네트워크 레벨의 flow 기반 혼잡제어 방법은 현재 인터넷의 근본적인 혼잡상황 문제를 명시적으로 해결 할 수 있게 해준다. 즉 이러한 방법은 기존의 Drop tail 방법과 같은 flow 처리 방법을 개선하여 지수적인 트래픽 증가 상황에서도 혼잡제어 가능한 망을 설계 할 수 있게 한다. 일반적으로 flow 처리 기법은 트래픽이 집중되는 라우터나 게이트웨이에 구현하게 된다.

IETF의 RFC 2309에서는 네트워크 레벨의 혼잡상황을 제어하기 위해서 라우터 알고리즘을 크게 "스케줄링 알고리즘(scheduling algorithms)"과 "큐 관리 알고리즘(queue management algorithms)"로 분류하고 있다. 일반적으로 라우터 알고리즘은 모든 flow에 대하여 공정성을 제공하여야 하며 모든 네트워크 환경에 대하여 구현이 용이해야 한다. 하지만 두 조건을 동시에 만족하기 매우 어렵다. 왜냐하면 스케줄링 알고리즘의 경우 모든 flow에 대하여 공정성을 제공하는 이점이 있지만 이를 제공하기 위해서는 복잡한 알고리즘이 필요하며 각 flow 단위로 큐를 관리해야하는 어려움 때문에 구현이 어렵기 때문이다. 반면 큐 관리 알고리즘의 경우 큐 관리에 있어서 하나의 큐를 관리하기 때문에 구현에 있어 용이 하지만 flow 간의 공정성을 완전히 제공하지는 못한다. 즉, 위에서 분류한 두 가지 타입의 알고리즘은 서로 추구하는 목적이 다소 다르기 때문이다[1].

본 논문에서는 네트워크 레벨에서 구현이 간단하면서도 flow 간의 공정성 및 성능을 개선할 수 있는 새로운 큐 관리 알고리즘을 제안하였다. 새로 제안한 알고리즘은 기존의 큐 관리 알고리즘 보다 효율적으로 큐를 관리한다.

본 논문의 2장에서는 혼잡상황 제어 알고리즘에 대한 개념과 RED에 대해 기술하였고 3장에서는 제안한 MRED 알고리즘에 대하여 기술하였다. 4장에서는 시뮬레이터를 이용한 실험 결과에 대하여 기술하였고 실제로 구현하여 간단히 성능을 분석하여 보았다. 5장에는 결론을 맺었다.

2. 관련 연구

2.1 네트워크 레벨의 혼잡상황 제어

Per-hop 패킷 처리를 하는 네트워크 레벨에서의 혼잡제어 방법은 트래픽이 집중되는 라우터나 게이트웨이의 출력 링크에서 링크 자원을 효율적으로 공유하는 라우터 알고리즘을 이용하는 방법이다. 일반적으로 라우터는 유입되는 패킷을 라우팅 하기 위하여 라우터로 들어오는 패킷들을 큐에 임시적으로 저장한다. 이때 가장 간단하게 저장하는 방법으로 FIFO 큐가 사용된다. 만약 라우터로 들어오는 패킷의 입력율이 라우터를 통하여 나가는 출력링크의 처리율을 넘는 경우 임시로 저장되는 큐에 저장이 안된 패킷들은 모두 버려지게 된다. 이러한 기본적인 FIFO 처리 방식을 Drop tail이라 한다. 이 방법은 큐 관리가 가장 간단하여 구현하기는 쉽지만 flow간의 불공정한 출력링크 할당, 전역 동기화(global synchronization)에 따른 평균지연 시간의 증가 및 성능 감소 등의 문제점을 가지고 있어 혼잡상황 제어에 적합하지 않다[5,6].

일반적으로 이러한 문제를 해결 할 수 있는 라우터나 게이트웨이에서의 알고리즘은 크게 두 가지로 분류하여 논의되고 있다. 첫 번째는 스케줄링 알고리즘으로 모든 flow들이 출력 링크를 공정하게 공유하는 방법이다. 대표적인 스케줄링 알고리즘으로 FQ(Fair Queueing) 알고리즘이 있다. 이 알고리즘은 각 flow에 대하여 개별적인 큐를 분리하여 관리하는 per-flow방식을 사용한다. Per-flow방식은 각 flow별로 큐를 유지 관리하므로 공정한 대역할당을 할 수 있다. 그러나 flow들에 관한 정보를 유지하고 처리하기 위해서 복잡한 알고리즘을 필요로 하기 때문에 고속의 라우터 처리 능력을 갖는 프로세스를 요구한다. 그리고 각 flow의 상태 정보를 가지고 있어야 하기 때문에 많은 flow를 갖는 네트워크 환경에서 오버헤드(overhead)가 상대적으로 커진다[7].

최근에는 이러한 정보를 유지 관리하는 오버헤드를 줄이기 위해서 CSFQ(Core Stateless Fair Queueing)와 SFQ(Stochastic Fair Queueing) 같은 알고리즘이 제안되었다. CSFQ 알고리즘은 라우터의 경계를 두 부분으로 나누어 관리하는데, 하나는 경계(edge) 라우터이고 다른 하나는 망의 중앙(core) 라우터이다. 여기서 경

게 라우터는 도착하는 모든 flow에 대하여 각 flow을 per-flow방식으로 처리하여 flow의 상태 정보를 패킷의 헤더에 표시한다. 그러면, 간단한 FIFO 방식으로 구성되어 있는 중앙 라우터는 각 패킷의 헤더에 추가된 상태 정보를 이용하여 패킷을 처리한다. 이 방법은 중앙 라우터에서의 큐를 분류 및 관리하는 복잡도를 줄일 수 있다. 그러나 이 방법 역시 여전히 경계 라우터 설계의 복잡도를 가지고 있으며 flow 상태 정보에 대한 오버헤드가 크다. 반면, SFQ 알고리즘은 해쉬(hash)함수를 이용하여 flow를 FQ 알고리즘에 비해 적은 수의 큐로 관리한다. 그러나 이 방법 역시 FQ 알고리즘 보다 복잡도는 감소하였지만 일반 라우터에서 FQ 알고리즘에 근접한 성능을 얻기 위해서는 적어도 1000개에서 2000개의 큐가 필요하다. 일반적으로 스케줄링 알고리즘의 경우 출력 링크의 모든 flow에게 공정한 대역 할당을 목표로 하여 구현이 되기 때문에 구현이 어렵다[8,9,10].

처음부터 간단한 원칙으로 설계된 큐 관리 알고리즘은 각 flow들에게 어느 정도의 공정성을 유지 제공하면서도 flow 상태에 대한 오버헤드 또한 적게 할 수 있다. 큐 관리 알고리즘이 처리해야 하는 혼잡상황을 분류하면 다음과 같다.

- 일시적인 혼잡상황(transient congestion) : 전송 프로토콜로 인해 반응하여 혼잡상황을 해결하는 시간 보다 짧은 주기를 가지는 혼잡상황을 말하며 이는 짧은 주기로 버스트(bursts)한 특징을 가지는 소수의 flow 때문에 발생한다.
- 지속적인 혼잡상황(long-term congestion) : 큐를 통과하는 모든 flow들의 전송률을 거의 고정적인 상태로 유지하여 생기는 혼잡상황을 말하며 이는 일반적으로 트래픽의 폭주로 발생한다.

2.2 RED 알고리즘

Sally Floyd가 제안한 RED(Random Early Detection) 알고리즘은 모든 flow를 하나의 큐를 통하여 처리하며 네트워크 혼잡상태를 큐 크기의 변화로 제어한다. 이처럼 RED는 기존의 혼잡제어 방법과는 다르게 혼잡상황을 예상하여 도착하는 임의의 flow의 패킷을 폐기하기 때문에 Drop tail 방법이 가지고 있는 "락 아웃(lock out)"과 "풀 큐(full queue)" 문제를 해결 할 수 있다. 또한 미리 혼잡상황을 발견하여 혼잡제어를 수행하기 때문에 네트워크 자원의 낭비를 Drop tail 방법 보다 줄여 좋은 성능을 나타낼 수 있다. 그리고 효과적인 큐 관리로 평균 큐 크기를 작게 유지할 수 있어 큐잉으로 생기는 전송지연 시간을 줄일 수 있다[11,12].

RED가 혼잡 상황을 미리 예측하여 이 상황을 제어하기 위해서는 TCP와 같은 전송 프로토콜이 재동작(react)하기까지의 시간 동안 링크용량을 수용할 수 있는 충분한 크기의 큐 공간이 필요하다. 만약 큐 공간의 크기가 작아 혼잡상황을 알린 후 네트워크의 전송률을 줄이기 이전에 모든 패킷이 폐기되기 시작한다면 오히려 RED 알고리즘의 혼잡제어 수행으로 인한 초기 자원 낭비가 커지게 된다. RED 알고리즘의 경우 혼잡상황을 현재 점유되어 있는 큐 크기의 변화만으로 예측한다. 하지만, 이 정보는 순간적인 트래픽의 변화 및 flow수의 증가에 매우 민감하여 혼잡상황을 제어하기에는 불충분하다. 따라서 효율적으로 혼잡상황을 제어하기 위해서는 추가적인 정보가 요구된다. 또한, RED 알고리즘의 경우 큐 공간 크기가 작으면 최소 경계 값과 최대 경계 값 사이의 크기가 작아져 오히려 전역 동기화의 원인이 될 수 있다. 따라서 RED 알고리즘은 평균 큐 크기 정보를 이용하여 효율적인 혼잡제어를 하기 위해서는 들어오는 패킷을 모두 수용 할 수 있는 충분한 크기의 큐 공간을 확보해야만 한다. 하지만 큐의 크기가 커질수록 큐잉으로 인한 전송시간 지연으로 실시간 트래픽에 대해서는 좋은 서비스가 현실적으로 어려움으로, 적은 큐 공간 크기를 유지하면서 좋은 성능을 제공하기 위해서는 RED 알고리즘을 개선하여야 한다[13,14].

2.3 RED의 개선 방안

RED 알고리즘과 같이 패킷 폐기 방법으로 혼잡상황을 제어하는 알고리즘에서 가장 중요한 요소는 패킷 폐기 확률을 계산함에 있어 어떠한 정보를 기반으로 하는가에 있다. RED 알고리즘의 경우 단순히 전체 큐 크기에서 현재의 패킷들이 점유되어 있는 평균 큐 크기만을 이용하여 계산한다. 이렇게 계산된 패킷 폐기 확률 값은 평균 큐 크기에 선형적인 함수로 계산된다. 하지만 네트워크 트래픽이 동적으로 변화하여 평균 큐 크기의 변화가 심하게 생길 경우 큐 크기 폐기 확률을 계산하기에는 어려움이 있다[11].

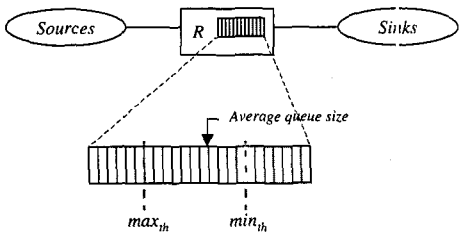
RED 알고리즘의 경우 패킷 폐기 확률을 평균 큐의 변화 정도를 가지고 계산한다. 따라서 RED 알고리즘의 성능을 결정하는 여러 파라미터 값들 중 평균 큐 크기를 결정하는 w_q(Queue Weight) 값의 영향을 받는다. 이 값이 너무 크거나 작을 경우 큐 변화량에 따른 평균 큐 크기의 변화의 정도가 불명확하여 혼잡상황 정도의 예측이 어려워진다. 따라서 평균 큐 크기의 변화만을 이용하여 혼잡상황을 제어하는 패킷 폐기 확률을 계산하는 것은 보완되어야 한다.

본 논문에서는 이러한 RED 알고리즘의 단점을 보완하기 위하여 추가적인 정보인 패킷 손실 정보와 링크

이용 히스토리(link utilization history) 정보를 이용하는 새로운 알고리즘을 제안하였다. 패킷 손실 정보와 링크 이용 히스토리 정보는 평균 큐 크기의 변화뿐 아니라 이전에 혼잡상황을 제어하기 위해서 사용한 변화율의 정도를 기억하고 있어 작은 큐 크기에서도 네트워크 상태에 적합한 패킷 폐기 확률을 구할 수 있게 한다. 제안한 적응적인 큐 관리 알고리즘을 MRED(Modified Random Early Detection)라 부른다. MRED 알고리즘은 네트워크 혼잡 상황 시 RED 알고리즘이 제공하는 패킷 폐기 확률 값을 휴리스틱한 방법을 통하여 적응적으로 계산하는 알고리즘이다. 즉, 이 알고리즘은 혼잡상황의 변화의 증감을 이용하여 효과적으로 패킷 폐기 확률을 계산한다. 이 알고리즘을 이용하면 효율적인 폐기 확률 값을 계산하여 패킷 손실로 인해 발생할 수 있는 자원의 낭비를 막을 수 있다.

3. MRED 알고리즘

본 장에서는 RED 알고리즘에서 제공하는 이점을 그대로 이용하면서 작은 큐 공간 크기에서도 효율적인 큐 관리를 제공 할 수 있는 새로운 적응적 큐 관리 알고리즘인 MRED 알고리즘에 대해서 기술한다. 라우터나 게이트웨이에서 혼잡제어를 하는 큐 관리 알고리즘은 네트워크에서 발생하는 혼잡상황에 정도를 감지하기 위해서 각 출력(output) 인터페이스 큐의 평균 큐 크기를 이용한다. 혼잡상황의 정도에 따라서 큐 관리 알고리즘은 패킷 폐기 확률을 계산하여 패킷을 적절히 폐기시킨다. MRED 알고리즘는 효과적인 혼잡제어를 위해서 평균 큐 크기 이외에도 패킷 손실 정보와 링크 이용 히스토리를 이용하여 각 큐의 혼잡상황을 제어한다. 혼잡상황을 감지하기 위해서 MRED 알고리즘의 평균 큐 크기는 RED 알고리즘과 같이 EWMA(Exponential weighted moving average)방법으로 계산한다[11]. 그림 1은 라우터의 큐에서의 MRED 알고리즘의 최소 경계 값(min_{th}), 최대 경계 값(max_{th})과 평균 큐 크기의 관계를 보여준다.



R : router
min_{th} : minimum threshold
max_{th} : maximum threshold

그림 1 두 임계값과 평균 큐 크기

MRED 알고리즘은 그림 2와 같이 네트워크의 혼잡상황을 감지하기 위해서 평균 큐 크기를 최소 경계 값과 최대 경계 값과 비교하여 혼잡상황의 정도(level)를 구분한다.

```
For each packet arrival
    calculate the average queue size avg
    if min_th ≤ avg ≤ max_th
            calculate probability P_a
            with probability P_a:
                    drop the arriving packet
    else if max_th ≤ avg
            drop the arriving packet

avg : average queue size
min_th : minimum threshold for queue
max_th : maximum threshold for queue
P_a : packet drop probability
```

그림 2 MRED 알고리즘

비교한 평균 큐 크기가 최소 경계 값보다 작다면 링크의 대역이 충분한 비혼잡상황이므로 패킷 폐기 없이 처리한다. 반면, 평균 큐 크기가 최대 경계 값 보다 크다면 링크의 대역폭이 부족하여 풀 큐 현상이 발생하는 혼잡상황이므로 도착하는 모든 패킷은 폐기한다. 만약, 평균 큐 크기가 최소 경계 값과 최대 경계 값 사이에 존재한다면 도착하는 패킷은 평균 큐 크기의 변화에 따라 혼잡상황의 정도를 결정하여, 패킷 폐기 확률(P_a)을 계산하여 큐 안에 저장된 임의의 패킷을 폐기한다. 여기서 폐기 확률 P_a는 RED 알고리즘과는 다르게 평균 큐 크기의 변화 정보 이외에도 이전 변화율의 정도를 기억하고 있는 패킷 손실 정보와 링크 이용 히스토리를 바탕으로 계산한다.

본 논문에서 제안한 MRED 알고리즘은 두개의 알고리즘으로 구분할 수 있다. 하나는 라우터의 큐에서 어느 정도의 버스트한 특성(burstiness)을 허락하는지를 결정하기 위해서 평균 큐 크기를 계산하는 것이다. 평균 큐 크기의 계산 방법은 EWMA을 이용하였다. 다른 하나는 혼잡상황 정도에 따라 패킷을 폐기하는 패킷 폐기 확률 P_a를 계산하는 것이다. MRED 알고리즘은 앞에서 기술한 것과 같이 기존의 RED 알고리즘의 패킷 폐기 확률 방법을 개선하였다.

RED 알고리즘은 혼잡상황 정도에 따라 평균 큐 크기

의 선형적인 함수 분포로 패킷 폐기 확률 P_a 계산한다. 혼잡상황에 대하여 선형적인 분포로 패킷 폐기 확률이 계산되므로 혼잡상황의 정도의 범위를 구하는 두 경계 값의 차가 충분히 커야한다. 결국, 두 경계 값의 차를 충분히 크게 하기 위해서는 큐의 크기가 커져야 한다. 반면 MRED 알고리즘은 이러한 선형 함수가 가지는 문제점을 해결하기 위해서 패킷 손실 정보와 링크 이용 히스토리 정보를 이용하여 계단(step) 형식의 패킷 폐기 확률 값을 계산한다. 그림 3은 MRED 알고리즘의 패킷 폐기 확률 계산 알고리즘을 보여 준다. 그림 3에서 보는 것과 같이 혼잡상황 동안에 평균 큐 크기가 두 경계 값 사이에 존재하면서 평균 큐 크기가 증가하거나 감소 할 경우 현재의 네트워크 혼잡상황에 적합한 폐기 확률을 계산한다. 즉, 특정 주기 동안 평균 큐 크기가 증가하였을 경우 패킷 폐기 확률 P_a를 d_1만큼 증가시킨다. 반면, 특정 주기 동안 평균 큐 크기가 감소하였을 경우 패킷 폐기 확률 P_a는 d_2만큼 감소시킨다.

```
if min_th ≤ avg ≤ max_th
    if avg increase
        if current_time – last_up_time > up_time
            P_a + d_1

    if avg decrease
        if current_time – last_up_time > up_time )
            P_a – d_2
```

그림 3 MRED의 패킷 폐기 확률 계산 알고리즘

제안한 알고리즘은 기존의 RED 알고리즘과 비교하여 보면, 평균 큐 크기가 최소 경계 값 아래에 있는 대기(idle)상태 이후 트래픽의 증가로 혼잡상황이 발생하면, RED 알고리즘의 경우 단지 평균 큐 크기의 변화에 따라 선형적인 패킷 폐기 확률을 계산한다. 반면, MRED 알고리즘은 이전 네트워크의 히스토리를 가지고 있는 이전 패킷 폐기 확률 P_a를 기초로 다시 패킷 폐기 확률을 계산한다. 이를 통하여 폐기된 패킷 때문에 발생하는 재전송으로 다시 발생 할 수 있는 혼잡 상황에 대하여 적응 할 수 있다. 따라서 MRED 알고리즘은 RED 알고리즘보다 혼잡상황을 더 정확히 개선시킬 수가 있다. 네트워크 상황에 대하여 보다 적응적으로 패킷 폐기 확률을 계산하기 위해서 그림 4와 같이 MRED 알고리즘의 패킷 폐기 확률을 조절한다.

패킷 폐기 확률의 변화는 TCP의 혼잡제어 방법에서 사용하는 AIMD(Additive Increase Multiplicative Decrease)방법과 유사하게 동작한다. 즉, 순간적으로 발생되는 트래픽으로 인한 혼잡 상황에 대하여 빨리 반응

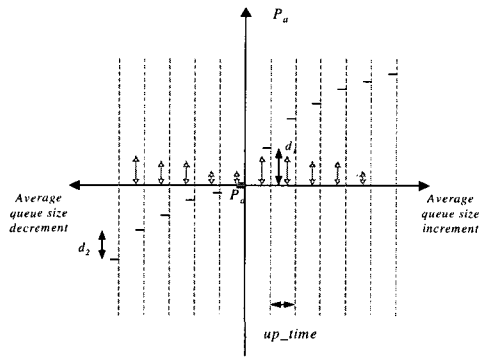


그림 4 MRED의 패킷 폐기 확률 변화

하기 위하여 초기의 패킷 폐기 확률의 증가 폭(d_1)을 크게 하였다. 만약, 계속해서 임의의 횟수 이상 패킷 폐기 확률이 증가 할 경우 네트워크 이용률을 높이기 위해서 최대 패킷 폐기 확률(P_{max})까지 증가폭(d_1)을 점점 감소시킨다. 그림 4에서 보는 것처럼 패킷 폐기 확률 P_a는 초기의 급격한 증가 이후 완만한 증가 분포를 보여 준다. 이와는 반대로, 네트워크 상황이 좋아져서 연속적으로 평균 큐 크기가 감소 할 경우 효과적인 큐 관리를 위해 패킷 폐기 확률의 감소 폭(d_2)을 점점 증가시킨다. 혼잡상황은 비교적 빨리 극복하고, 비혼잡상황은 네트워크 상황에 유의하면서 천천히 적응하기 위해서 d_1을 d_2보다 크게 설정하였다. RED 알고리즘과 MRED 알고리즘의 패킷 폐기 확률의 특성은 그림 5와 같이 나타낼 수 있다.

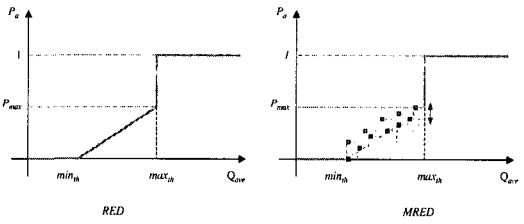


그림 5 패킷 폐기 확률 값

여기서 계산한 패킷 폐기 확률(P_a)을 기초로 패킷을 폐기 할 경우 그 폐기 확률 분포는 geometric한 분포를 보인다. 만약 패킷 폐기가 이러한 geometric한 분포를 가질 경우 폐기되는 패킷이 특정 flow에 편중되어 발생할 수 있다. 패킷 폐기가 편중 될 경우 특정 트래픽에 대해서 불공정한 서비스 할 수 있으며 동시에 많은 수의 flow 패킷을 폐기 시켜 전역 동기화 문제를 발생시

킬 수 있다. 따라서 이 문제를 해결하기 위해서는 모든 구간에서 균일한 분포를 가지는 uniform한 분포로 변환하였다[15]. 그림 6은 그림 7의 첫 번째 수식을 이용하여 확률 분포를 변환하였을 때의 패킷 폐기 분포이다. 수식의 변환없이 패킷을 폐기 할 경우 그림 6의 (a)처럼 한쪽 패킷에 편중되어 폐기되는 geometric한 분포를 보이는 반면 수식을 이용하여 재계산한 그림 6에서 (b)의 유형을 보면 모든 구간에서 균일한 uniform 한 분포를 보인다.

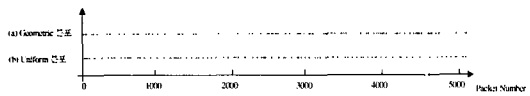


그림 6 패킷 폐기 확률 분포

$P_b \leftarrow P_b / (1 - count \cdot P_b)$ — *Geometric* to *Uniform*

$P[X = n] = (1 - P_b)^{n-1} \cdot P_b$ — *Geometric* random variables

$P[X = n] = P_b$ for $1 \le n \le 1/P_b$

$= 0$ for $n > 1/P_b$ — *Uniform* random variables

n: 최근에 packet drop 이후 도착 packet 수
count : 마지막 패킷 drop이후 drop되지 않은 packet수

그림 7 확률 분포 변환 수식

4. 실험 및 성능 평가

4.1 트래픽 패턴에 대한 실험

새로 제안한 MRED 알고리즘의 성능 평가를 위하여 본 논문에서는 LBNL(Lawrence Berkely National Laboratory)의 ns(Network Simulator) 네트워크 시뮬레이터를 이용하였다[16]. MRED 알고리즘의 성능을 평가하기 위해서 먼저 그림 8과 같은 시뮬레이션 네트워크 망을 구성하여 간단한 실험을 하였다.

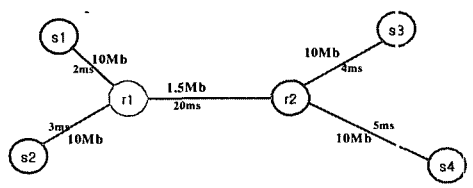


그림 8 시뮬레이션 네트워크 구성

실험을 위하여 그림 8과 같은 네트워크 망에서 먼저 s1 s3 사이에 TCP를 이용한 파일 전송(FTP)을 하였고 3초 후 s2에서 s3로 TCP를 이용한 파일을 전송하였을 때 r1과 r2 링크 사이에서 일어나는 패킷의 흐름을 모니터링 하였다. 실험에 사용한 각 큐 관리 알고리즘의 설정은 표 1과 같다.

표 1 RED와 MRED 설정

RED	MRED
min_{th}=5	Update time=0.2
max_{th}=15	d_1=0.002
Queue weight=0.002	d_2=0.0002

그림 9에서 평균 큐 크기의 변화량을 비교해 보면 MRED 알고리즘이 RED 알고리즘과 유사한 트래픽 특성을 가지고 있음을 확인 할 수 있다. 그리고 그림 10에서 보는 것과 같이 MRED 알고리즘이 RED 알고리즘보다 서로 다른 두 flow의 출력 링크 대역 할당에 대하여 좋은 공정성을 제공하는 것을 확인 할 수 있다.

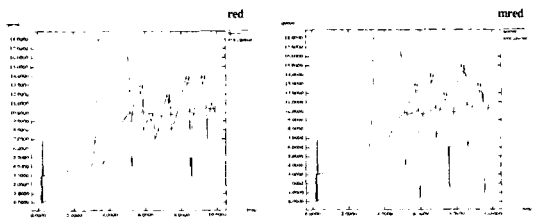


그림 9 큐 크기의 변화

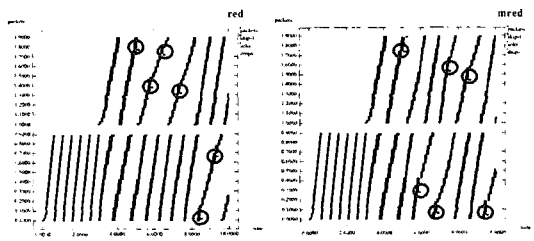


그림 10 flow별 출력링크 분포

4.2 공정성에 대한 실험

또한 MRED 알고리즘이 서로 다른 rate의 flow들에게 어느 정도의 공정성을 제공하는지 알아보기 위해서 그림 11과 같은 시뮬레이션 네트워크 망을 구성하여 실험을 하였다.

실험에 사용한 트래픽 특성은 지수적인(exponential) 트래픽으로 on time은 2초 off time은 1초로 설정하였

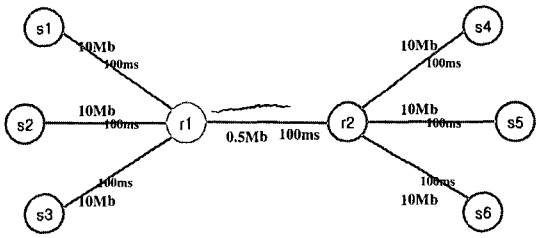


그림 11 시뮬레이션 네트워크 구성

으며 패킷의 크기는 200 바이트로 설정하였다. 실험 방법은 실험 시작 10초 후에 s1과 s4 사이에 요구 전송률이 100Kbps인 지수적인 트래픽을 전송하고 s2와 s5사이에 요구 전송률이 200Kbps인 지수적인 트래픽을 전송하면서 동시에 s3와 s6사이에 요구 전송률이 300 Kbps인 지수적인 트래픽을 같이 전송하였다. 실험은 60초 동안 하였으며 혼잡상황이 발생하는 r1과 r2사이에서 Drop tail, RED, MRED 알고리즘을 이용하여 요구 전송률이 서로 다른 3개의 flow를 모니터링 하였다.

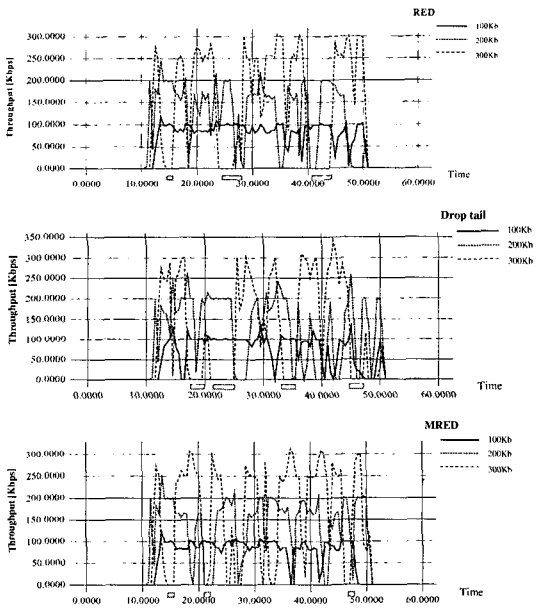


그림 12 Drop tail, RED, MRED 공정성 비교

결과 그래프인 그림 12와 같이 100Kb의 전송률을 가지는 flow의 경우 모든 알고리즘에 대하여 적절한 대역폭을 이용하는 것을 확인 할 수 있다. 그러나 300Kb와 같이 전송률이 높은 flow가 Drop tail을 사용하는 라우터를 지나는 경우 전혀 대역폭을 이용하지 못하는 것을 확인 할 수 있다. 네트워크에 혼잡상황이 발생할 경우 Drop tail 알고리즘은 overflow로 인하여 패킷 폐기할 때 전송률이 높은 300Kb의 flow들이 많이 폐기되어 거의 대역폭을 할 당 받지 못한다. 그러나 RED나 MRED와 같은 큐 관리 알고리즘의 경우 패킷 폐기 시 폐기 분포도 고려하여 폐기하기 때문에 Drop tail보다는 모든 flow에 대하여 개선된 공정성을 얻을 수 있다. RED 알고리즘의 결과 그래프의 경우도 Drop tail 방법보다는 좋은 성능을 제공하지만 전송률이 높은 flow에 대해서는 여전히 낮은 성능을 제공하고 있다. 본 논문에서 제안한 MRED 알고리즘의 결과 그래프의 경우 모든 flow에 대하여 공정하게 서비스하므로 전체적으로 다른 방법보다 개선된 전송 성능을 보여준다.

4.3 전송 성능과 지연 시간 대한 실험

MRED 알고리즘이 큐 공간의 크기에 따라 어떠한 성능을 내는지 알아보기 위해서 다음과 같은 실험을 하였다. 실험을 위하여 그림 13과 같은 시뮬레이션 네트워크 망을 구성 한 후 먼저 s1과 s3사이에 TCP를 이용한 파일 전송을 하였고 임의의 시간 이후에 s2와 s4사이에 TCP를 이용한 파일 전송을 하였다. Drop tail, RED, MRED 알고리즘을 이용하여 큐 공간의 크기를 변화시켰을 때의 r1과 r2사이의 전송 성능 및 지연 시간을 모니터링 하였다. 그리고 목적지까지 도착하는 패킷들의 평균 지연 시간을 계산하였다. 참고로 본 실험에서의 TCP flow에는 100개의 패킷을 처리하는데 0.017 초의 오버헤드를 부과하였다.

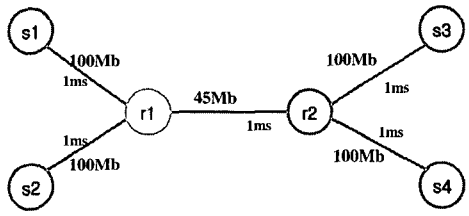


그림 13 시뮬레이션 네트워크 구성

그림 14는 전송 성능을 실험 결과로 그림에서와 같이 큐 공간의 크기가 증가함에 따라 전송 성능이 증가함을 알 수 있다. 이미 기술한바와 같이 RED 알고리즘의 경우 큐 공간 크기가 커질수록 전송 성능 증가함을 보여준다. 그림 14에서 MRED 알고리즘은 다른 큐 관리 알고리즘 보다 동일 큐 공간 크기에서 좋은 성능을 나타냄을 보여 주고 있다.

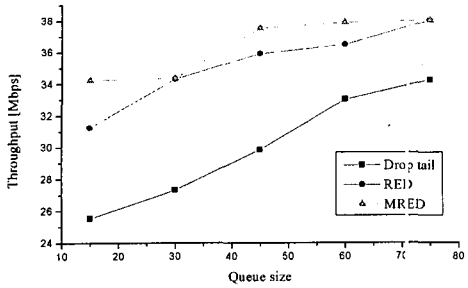


그림 14 패킷의 전송 성능 실험

실험 결과 그림 15에서 보는 것과 같이 패킷의 평균 전송 지연 시간은 큐 공간 크기가 증가함에 따라 증가하고 있음을 알 수 있다. 일반적으로 전송 지연 시간은 큐 공간 크기에 비례하여 큐 공간 크기가 커질수록 전송 지연시간이 길어진다. 이 실험에서도 역시 MRED 알고리즘의 경우 RED 알고리즘 보다 전반적으로 좋은 성능을 보여 주고 있다. 본 논문에서 제안한 MRED 알고리즘은 기존의 RED 알고리즘의 동작 방식을 개선하여 불필요한 패킷 폐기 과정이나 부적절한 평균 큐 크기의 유지하는 과정을 개선하여 네트워크 혼잡 상황에 맞게 패킷을 폐기하며 가능하면 적은 평균 큐 크기를 유지하도록 동작하여 기존의 RED보다 개선된 성능을 보여준다. 실험 결과와 같이 큐 관리 알고리즘의 경우 버퍼의 크기가 클 경우 많은 수의 패킷을 폐기 없이 큐에 관리 및 유지 할 수 있어 패킷 폐기 및 재 전송으로 인한 발생하는 네트워크 자원의 낭비를 줄여 전송 성능이 개선이 된다. 하지만 큐 공간의 크기가 증가 할 경우 큐에 대기하는 패킷의 수가 증가하여 평균 지연 시간이 증가하는 결과를 보여주고 있다.

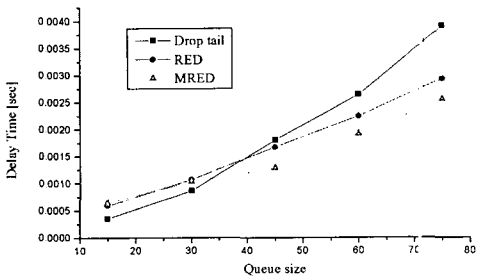


그림 15 패킷의 평균 전송 지연시간 실험

4.4 리눅스 커널의 MRED 알고리즘의 구현 및 실험

본 논문에서는 MRED 알고리즘을 실제 네트워크 환경에서 평가하기 위해서 PC를 기반으로 한 리눅스(Linux) 환경에서 라우터 기능을 구현하고, 라우터의 출력 인터페이스의 트래픽 제어부에 MRED 알고리즘을 구현하였다.

리눅스로 구현된 시스템을 이용하여 MRED 알고리즘을 사용하는 라우터 트래픽 제어부와 기존의 RED 알고리즘과 Drop tail 방법을 사용하는 라우터 트래픽 제어부의 전송률을 비교 평가하였다[18]. 본 논문에서의 실험은 그림 16과 같은 네트워크 환경에서 진행하였으며, 왼쪽의 호스트와 라우터 사이는 100Mbps 링크를 사용하였고 오른쪽의 호스트와 라우터 사이는 10Mbps 링크를 사용하였다.

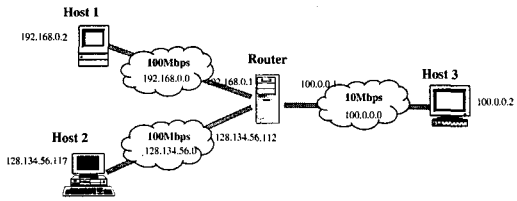


그림 16 실험 환경

실험방법은 호스트 2에서 호스트 3로 FTP 프로그램을 이용하여 파일을 전송하였고 약 20초 후 HP의 Netperf (Network Performance Benchmark)[19] 툴(tool)를 이용하여 호스트 1과 호스트 3사이의 TCP flow의 전송률을 측정하였다. 이 실험에서 사용한 각 알고리즘의 설정은 표 2와 같다.

표 2 실험을 위한 알고리즘의 설정

RED	MRED
min_{th}=15Kb	Update time=0.001초
max_{th}=45Kb	d_1=0.002
Queue weight=0.002	d_2=0.0002

총 3회 에 걸쳐 실험을 하였고 10초 동안 Netperf로 생성한 TCP flow의 평균 전송률을 측정하였다. 실질적인 실험을 통하여 측정된 결과와 시뮬레이터를 이용하여 실험한 결과를 비교하기 위해서 ns 시뮬레이터 환경에 리눅스 환경에서 실험한 환경을 구성하여 실험하였다. 두 실험 환경의 결과인 그림 17에서 보는 것과 같이 MRED 알고리즘은 기존의 큐 관리 알고리즘 보다 우수한 성능을 보임을 알 수 있다. 그리고 MRED 알고리즘은 두 실험 환경에서 모두 유사한 성능 개선을 보였다.

두 실험환경에서 각 알고리즘 측정값들의 차이는 호스트와 라우터사이 링크의 실효 속도가 다르기 때문이다. PC를 이용하여 구성한 라우터와 호스트 사이의 실효 속도는 약 7.5Mbps였고 시뮬레이터로 구성한 라우터와 호스트 사이의 실효 속도는 약 9.5Mbps였다.

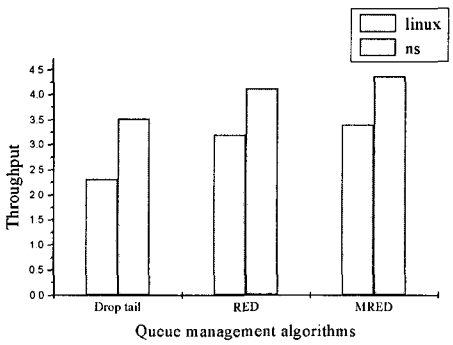


그림 17 리눅스와 ns 실험 결과 비교

5. 결론 및 향후 연구 과제

본 논문에서는 동적으로 변화하는 인터넷 트래픽을 단지 평균 큐 크기 정보만을 기초로 혼잡상황을 제어하려는 기존의 RED 알고리즘을 개선한 MRED 알고리즘을 제안하였다. 이 알고리즘은 기존의 RED 알고리즘이 가지는 이점을 그대로 이용하여 효과적으로 혼잡상황을 제어하는 알고리즘이다. MRED 알고리즘은 기존의 네트워크 혼잡상황시 RED 알고리즘이 계산하는 패킷 폐기 확률 방법을 개선하여 휴리스틱한 방법을 통하여 패킷 폐기 확률을 계산한다. 시뮬레이터를 이용한 실험을 통하여 공정성과 전송 성능 및 전송 지연 시간에 대한 성능을 비교 분석하였다. 추가로 PC를 이용한 라우터에 이 알고리즘을 구현하여 실제 네트워크 환경에서의 성능을 검증하였다.

향후 연구 과제로는 비반응(unresponsive) flow에 대한 공정성을 개선하는 방법에 대한 연구가 수행되어야 하며, 다양한 네트워크 환경에서의 성능 개선 및 구현에 대한 연구가 수행되어야 할 것이다. 그리고 구현한 MRED 알고리즘을 확장하여 네트워크 레벨의 QoS를 지원하는 라우터 알고리즘에 대하여 연구해야 할 것이다.

참 고 문 헌

[1] Braden, B., Clark, D., Crowcroft, J., Davie, B., Deering, S., Estrin, D., Floyd, S., Jacobson, V., Minshall, G., Partridge, C., Peterson, L., Ramakrishnan, K., Shenker, S., Wroclawski, J., Zhang, L., "Recommendations on Queue Management and Congestion Avoidance in the Internet," IETF RFC (Informational) 2309, April 1998.

[2] Jacobson, V., "Congestion Avoidance and Control," Proceeding of SIGCOMM88, August 1988.

[3] Stevens, W., "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms," IETF RFC 2001, January 1997.

[4] Floyd, S., and Fall, K., "Router Mechanisms to Support End-to-End Congestion Control," LBL Technical report, February 1997.

[5] 이원일 외, "고속 라우터를 위한 Drop-tail 방식의 공정한 대역할당 알고리즘", 한국통신학회 논문지, pp. 910-917, 2000, 6.

[6] Reininger, D., Raychaudhuri, D. and Ott, M., "A Dynamic Quality of Service Framework for Video in Broadband Networks," IEEE Network, November 1998.

[7] Demers, A., Keshav, S. and Shenker, S., "Analysis and Simulation of a Fair Queueing Algorithm," Journal of Internetworking Research and Experience, Oct. 1990. Also in Proceedings of ACM SIGCOM M' 89.

[8] Stoica, I., Shemker, S., and Zhang, H., "Core-Stateless Fair Queueing: Achieving Approximately Fair Bandwidth Allocations in High Speed Networks," Proceedings of ACM SIGCOMM' 98, August 1998.

[9] McKenny, P., "Stochastic Fairness Queueing," Proceedings of INFOCOM' 90, pp.733-740, June, 1990.

[10] Manin, A. and Ramakrishnan K., "Gateway Congestion Control Survey," IETF RFC(Informational) 1254, August 1991.

[11] Floyd, S., and Jacobson, V., "Random Early Detection Gateways for Congestion Avoidance," IEEE/ACM Transaction on Networking, August 1993.

[12] Pan, R., Prabhakar, B., and Psounis, K., "CHOKe, A Stateless Active Queue Management Scheme for Approximating Fair Bandwidth Allocation," Proceedings of INFOCOM' 2000, February 2000.

[13] Feng, W., Kandlur, D., Saha, D., Shin, K., "Blue: A New Class of Active Queue Management Algorithms," Univ. of Michigan CSE-TR-387-99, April 1999.

[14] Cisco Technical Report, "Congestion Avoidance Overview," Cisco IOS Release 12.0, Quality of Service Solutions Configuration Guide, June 1999.

[15] Arnold O. Allen, *Probability, Statistics, and Queueing Theory*, Second Edition, Academic Press.

1990.
[16] UCB/ LBNL/ VINT, "Network Simulator ns (Version 2)," http://www-mash.cs.berkeley.edu/ns/.
[17] W. Richard Stevens, *TCP/IP Illustrated, Volume 2*, Addison-Weslesy, 1994.
[18] Almesberger W., "Linux Network Traffic Control-Implementation Overview," Proceeding of the 5th Linux Expo, May 1999.
[19] Hewlett-Packard, "Netperf: A Network performance Benchmark," http://www.netperf.org/, 1995.

구 자 헌
1999년 광운대학교 전자통신공학과 학사. 2001년 광운대학교 전자통신공학과 석사. 2002년 ~ 현재 광운대학교 전자통신공학과 박사 과정. 관심분야는 컴퓨터 통신, 인터넷 QoS

송 병 훈
1998년 광운대학교 컴퓨터과학과 학사. 2000년 광운대학교 전자통신공학과 석사. 2001년 ~ 현재 광운대학교 전자통신공학과 박사 과정. 관심분야는 컴퓨터 통신, 멀티미디어 통신

정 광 수
1981년 한양대학교 전자공학과 학사. 1983년 한국과학기술원 전기 및 전자공학과 석사. 1991년 미국 University of Florida 전기공학과 박사(컴퓨터공학전공). 1983년 ~ 1993년 한국전자통신연구원 선임연구원. 1991년 ~ 1992년 한국과학기술원 대우교수. 1993년 ~ 현재 광운대학교 전자공학부 부교수(정보통신 연구원). 관심분야는 멀티미디어통신, 컴퓨터통신, 분산처리

오 승 준
1980년 2월 서울대학교 전자공학과 졸업(학사). 1982년 2월 서울대학교 전자공학과 대학원 졸업(석사). 1988년 5월 미국 Syracuse University 전기 및 컴퓨터공학과 졸업(박사). 1982년 3월 ~ 1992년 8월 한국전자통신연구원 근무(멀티미디어연구실 실장). 1986년 7월 ~ 1986년 8월 NSF Supercomputer Center 초청 학생연구원. 1987년 5월 ~ 1988년 5월 Northeast Parallel Architecture Center 학생연구원. 1992년 9월 ~ 현재 광운대학교 전자공학부 및 정보통신연구원 부교수(멀티미디어연구실). 2000년 3월 ~ 현재 (주)인티스 정보통신연구소 연구소장. 관심분야는 비디오처리, 비디오 및 영상압축, 멀티미디어시스템