# 인터넷 혼잡제어에서 공정성 향상을 위한 새로운 큐 관리 알고리즘

## (A New Queue Management Algorithm Improving Fairness of the Internet Congestion Control)

구 자 헌 † 최 웅 철 †† 정 광 수 †††
(Ja-Hon Koo) (Woong-Chul Choi) (Kwang-Sue Chung)

**요 약** 현재 인터넷 라우터는 네트워크 트래픽의 지수적인 증가로 인해 발생하는 혼잡 상황을 명시적으로 해결 할 수 없다. 이러한 문제를 해결하기 위해 IETF(Internet Engineering Task Force)에서는 RED(Random Early Detection) 알고리즘과 같은 능동적인 큐 관리(Active Queue Management) 알고리즘을 제시하였다. 하지만 RED 알고리즘은 UDP 플로우와 같이 양 종단간 혼잡제어를 하지 않는 플로우(flow)나 RTT(Round Trip Time)값이 작은 TCP 플로우에 대한 불공정성 문제를 가지고 있으며 이로 인하여 혼잡상황이 악화되는 문제를 가지고 있다.

본 논문에서는 이러한 문제를 해결하기 위해, 새로운 공정성 향상 큐 관리 알고리즘인 AFQM(Approximate Fair Queue Management) 알고리즘을 제안하였다. AFQM 알고리즘은 공정분배율(fair share rate) 이상의 대역폭을 차지하는 플로우를 판별하여 공정성을 보장한다. AFQM 알고리즘은 적은 오버헤드(overhead)와 구현의 용이성을 가지고 있기 때문에 비반응 플로우(unresponsive flow)와 불량 사용자 플로우(misbehaving flow)를 큰 오버헤드 없이 관리하여 플로우간의 공정성을 보장하는 알고리즘 이다.

**키워드** : 혼잡제어, RED, 능동적 큐 관리, 공정성

***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) algorithm. However, RED algorithm simple but does not protect traffic from high-bandwidth flows, which include not only flows that fail to use end-to-end congestion control such as UDP flow, but also short round-trip time TCP flows.

In this paper, in order to solve this problem, we propose a simple fairness queue management scheme, called AFQM(Approximate Fair Queue Management) algorithm, that discriminate against the flows which submit more packets/sec than is allowed by their fair share. By doing this, the scheme aims to approximate the fair queueing policy. Since it is a small overhead and easy to implement, AFQM algorithm controls unresponsive or misbehaving flows with a minimum overhead.

**Key words** : congestion control, RED, active queue management, fairness

## 1. 서 론

현재의 인터넷 프로토콜 구조는 비연결 지향적인(connectionless) 특성을 지닌 IP 프로토콜을 기반으로 하는 양 종단간(End-to-End) 패킷 전달 서비스로 구성되어져 있다. 이러한 비연결 지향적인 네트워크 환경의 이점은 유연성(flexibility)과 견고성(robustness)에 있는 반면 트래픽 제어가 어렵다는 단점을 가지고 있다. 따라서 네트워크의 과도한 혼잡 상황에서도 좋은 서비스를 제공하기 위해서는 특별히 주위 깊은 네트워크 설계가 요구된다. 만약 부주의하게 네트워크를 설계할 경우 동

· 본 연구는 한국과학재단 특정기초연구[R01-2002-000-00179-0(2002)]의 지원에 의해 수행되었음
† 학생회원 : 광운대학교 전자통신공학과
jhkoo@cclab.kw.ac.kr
†† 비 회 원 : 광운대학교 컴퓨터공학부 교수
wchoi@daisy.kw.ac.kr
††† 종신회원 : 광운대학교 전자공학부 교수
kchung@daisy.kwangwoon.ac.kr
논문접수 : 2002년 12월 17일
심사완료 : 2003년 3월 26일

적인 패킷 전달에 대하여 서비스 악화를 초래 할 수 있기 때문이다. 기술적으로 이런 현상을 혼잡 악화(congestion collapse)라 하며 인터넷이 처음 성장하게 된 1980년대 중반부터 이를 해결하기 위한 연구가 진행되고 있다[1].

최근 혼잡상황을 개선하려는 연구들은 크게 TCP와 같은 전송 프로토콜 레벨과 라우터와 같은 네트워크 레벨로 구분되어 연구되어지고 있다. TCP와 같은 전송 프로토콜을 이용한 혼잡제어 방법의 경우 네트워크 혼잡상황을 해결하기 위해서 윈도우 기반의 플로우(flow) 제어를 종단(end host)에서 하게 된다. 하지만, 이 방법으로는 의도적이거나 동적인 네트워크 혼잡 상황을 근본적으로 해결하지는 못하는 한계가 있다[2,3]. 이 문제를 좀 더 쉽게 해결하기 위한 방법인 네트워크 레벨의 플로우 기반 혼잡제어 방법은 현재 인터넷의 근본적인 혼잡상황 문제를 명시적으로 해결할 수 있으며 이 방법은 보통 트래픽이 집중되는 라우터나 게이트웨이의 알고리즘으로 구현하게 된다[4].

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

본 논문에서는 라우터에서 발생하는 불공정성 문제와 혼잡상황 문제를 해결하기 위해 새로운 공정성 향상 기법을 가지고 있는 큐 관리 알고리즘인 AFQM(Approximate Fair Queue Management) 알고리즘을 제안하였다. 제안한 AFQM 알고리즘은 공정분배율 이상의 대역폭을 차지하는 플로우를 판별하여 공정성을 보장하며, 적은 오버헤드와 구현의 용이성을 가지고 있다. 본 논문의 2장에서는 관련 연구와 RED 알고리즘에서의 불공정성의 문제에 대해서 기술하였고 3장에서는 새로 제안한 AFQM 알고리즘에 대하여 기술하였다. 4장에서는 시뮬레이터를 이용한 AFQM 알고리즘의 공정성 실험 결과에 대하여 기술하였고 5장에서는 결론을 맺었다.

## 2. 관계 이론

### 2.1 공정성 개선을 위한 연구

네트워크에서 발생되는 혼잡상황의 악화를 초래하는 불공정성 문제를 해결할 수 있는 라우터나 게이트웨이에서의 알고리즘을 개선하는 연구는 현재 활발히 진행중에 있다. 이와 관련된 방법은 크게 스케줄링 알고리즘과 큐 관리 알고리즘으로 분류하여 논의할 수 있다. 먼저 스케줄링 알고리즘으로 모든 플로우들이 출력 링크를 공정하게 공유하는 방법이며 대표적인 스케줄링 알고리즘으로 FQ(Fair Queueing) 알고리즘이 있다. 이 알고리즘은 각 플로우에 대하여 개별적인 큐를 분리하여 관리하는 플로우별 관리(per-flow)방식을 사용한다. 플로우별 관리방식은 각 플로우별로 구분하여 독립적인 큐를 유지 관리하므로 공정한 대역할당을 할 수 있다. 그러나 플로우들에 관한 정보를 유지하고 관리하기 위해서 복잡한 알고리즘을 필요로 하기 때문에 고속의 라우터 처리 능력을 갖는 프로세스를 요구한다. 그리고 각 플로우의 상태 정보를 가지고 있어야 하기 때문에 많은 플로우를 갖는 네트워크 환경에서 오버헤드(overhead)가 상대적으로 커진다[5].

최근에는 이러한 정보를 유지 관리하는 오버헤드를 줄이기 위해서 CSFQ(Core Stateless Fair Queueing)[6]와 SFQ(Stochastic Fair Queueing)[7] 같은 알고리즘이 제안되었다. CSFQ 알고리즘은 라우터의 경계를 두 부분으로 나누어 관리하는데, 하나는 경계(edge) 라우터이고 다른 하나는 망의 중앙(core) 라우터이다. 여기서 경계 라우터는 도착하는 모든 플로우에 대하여 각 플로우를 플로우별 관리방식으로 처리하여 플로우의 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 플로우 상태 정보에 대한 오버헤드가 크다. 반면, SFQ 알고리즘은 해쉬(hash)함수를 이용하여 플로우를 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 독립된 큐가 필요하다. 일반적으

로 스케줄링 알고리즘의 경우 출력 링크의 모든 플로우에게 공정한 대역 할당을 목표로 하여 구현이 되기 때문에 일반적으로 구현이 어려운 문제점을 가지고 있다. 현재 라우터 구현 기술로 충분히 FQ과 유사한 라우터를 구현하는 것은 기술적으로 어렵지 않다. 하지만 아직까지 이러한 라우터의 대중적인 사용이 보편화 되지 않았으며 아직도 구현의 복잡도와 비용이 상대적으로 높은 문제점을 가지고 있다[8].

처음부터 간단한 원칙으로 설계된 큐 관리 알고리즘은 각 플로우들에게 적은 오버헤드로 부분적인 공정성을 제공하는 방법이다. 하지만, 특성이 상이한 플로우간 불공정성 문제를 가지고 있어 이 문제를 해결하기 위한 연구가 활발히 진행 중에 있다. 공정성을 보장하기 위한 방법으로 비반응 플로우와 불량 사용자에 대한 불공정성 문제를 해결하기 위해 FRED(Flow Random Early Drop)[10]가 제안되었다. FRED 알고리즘은 RED 알고리즘과 같이 하나의 FIFO 큐 구조를 이용하는 큐 관리 알고리즘을 기반으로 하고 있어 다수개의 큐를 가지고 있는 스케줄링 알고리즘보다 적은 오버헤드를 가지고 있다. 그러나 모든 플로우에 대한 상태 정보를 유지 관리해야 하는 오버헤드를 가지고 있어 구현의 어려움과 설계의 복잡도를 가지고 있다[9]. 이러한 문제를 해결하고자 큐 관리 알고리즘의 간단함을 그대로 유지하면서 적은 관리비용으로 FQ에 근접한 공정성을 보장하는 CHOKe(CHOose and Keep for responsive flows, CHOose and Kill for unresponsive flows)[11] 알고리즘이 제안되었다. CHOKe 알고리즘은 상태 정보 관리 없이 공정성을 보장하지만 네트워크 상태에 따른 매개변수 설정이 어렵고 특정한 조건에서만 공정성을 보장하는 문제를 가지고 있다. 최근 Floyd가 제안한 RED-PD(RED Preferential Dropping)[12] 알고리즘은 적은 오버헤드로 다른 플로우보다 높은 대역폭을 차지하는 플로우를 먼저 폐기하여 부분적인 FQ에 근접한 공정성을 제공하는 알고리즘이 제안되었다. 그러나 이 방법의 중요 매개 변수인 목표 RTT(Target RTT)의 설정이 어려워 일반적인 네트워크 환경에 사용할 수 없는 단점을 가지고 있다. 기존의 큐 관리 알고리즘에서 공정성을 향상시킨 다양한 기법의 큐 관리 알고리즘 모두 스케줄링 알고리즘보다 적은 오버헤드를 가지고 있지만 특정 네트워크 환경에서만 공정성을 보장하는 문제를 가지고 있다. 따라서 적은 오버헤드와 구현의 용이성을 가지고 있으며 라우터의 플로우간의 공정성을 보장하는 새로운 큐 관리 알고리즘에 대한 연구가 필요하다.

### 2.2 RED 알고리즘에서의 불공정성 문제

1992년도 Floyd가 제안한 RED 알고리즘은 라우터와 같은 네트워크 중심(core)에서 네트워크 상황에 맞게 큐 안의 패킷을 관리하여 네트워크 혼잡상황을 해결하는 알고리즘이다. RED 알고리즘은 평균 큐 크기를 기반으로 혼잡상황의 정도를 인지하여 혼잡제어를 하며, 기존의 Drop-tail 방식의 라우터 알고리즘에서 발생되던 "풀 큐(full queue)"와 "락 아웃(lock out)"과 같은 현상을 해결한 알고리즘이다[9].

일반적으로 RED 알고리즘은 패킷 손실을 인지하여 재전송시 윈도우 기반의 혼잡제어를 하는 TCP의 back-off 알고리즘 특성을 이용하여 네트워크 라우터에서 혼잡제어를 한다. 따라서 패킷 손실에 혼잡제어 반응을 하지 않는 UDP 플로우와 같은 비반응 플로우(unresponsive flow)에 대해서는 라우터 기반의 혼잡제어를 적용하지 못하는 문제를 가지고 있다. 따라서 비반응 플로우는 혼잡상황으로 인하여 라우터에서 패킷이 폐기되었을 경우, 혼잡상황에 대해 전송 호스트는 인지하지 못한다. 그리고 프로토콜 레벨의 혼잡제어 알고리즘을 가지고 있지 않기 때문에 네트워크 혼잡상황에 대하여 back-off 알고리즘을 가지고 있는 TCP와 같은 반응 플로우(responsive flow)와 경쟁을 할 경우 불공정성 문제를 발생시킨다. 이뿐만 아니라, 네트워크 기반의 혼잡제어 방법으로 RED 알고리즘을 사용할 경우 비반응 플로우의 과도한 트래픽 증가는 RED 알고리즘이 혼잡상황을 더욱 심각하게 인지하게 만들며 결국 RED 알고리즘의 혼잡제어 방법에 따라 패킷 폐기 확률 값 $P$를 증가시키게 된다. 일반적으로 혼잡상황이 발생하는 라우터를 지나는 모든 플로우의 패킷에 대해 RED 알고리즘은 현재 라우터가 인지한 혼잡상황에 비례한 패킷 폐기 확률 값 $P$를 적용한다. 폐기 확률 값 적용 시 각 플로우의 특성은 전혀 고려가 되지 않으며 높은 대역폭을 가진 플로우나 낮은 대역폭을 가진 플로우나 동일한 조건으로 처리된다. 결국 그림 1과 같이 동일한 패킷 폐기 확률 값 $P$로 모든 플로우의 패킷을 폐기하기 때문에 플로우간

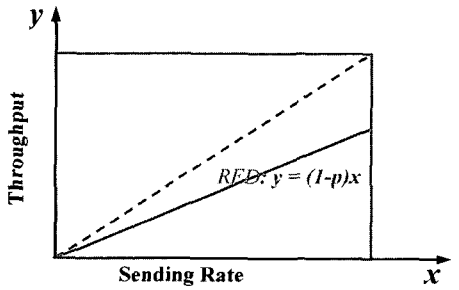


그림 1 RED 알고리즘의 혼잡제어

불공정성은 더욱 악화된다.

따라서 적은 전송률을 가지는 플로우는 상대적으로 큰 전송률을 가지는 플로우로 인한 더욱 강경한(aggressive) 혼잡제어를 받게 된다. 다시 말해 높은 대역폭을 가지는 플로우와 적은 대역폭을 가지는 플로우가 동일한 혼잡상황으로 처리되어 패킷 폐기율의 불공정한 처리가 발생한다. 이러한 라우터의 불공정성 현상은 라우터 알고리즘의 혼잡제어를 무의미하게 만들며 네트워크의 혼잡상황을 더욱 악화시키는 문제를 가져온다.

TCP와 UDP 플로우와 같이 서로 다른 특성을 가지는 플로우 사이의 불공정성 현상이외에도 라우터에서는 또 다른 불공정성 현상이 발생한다. 라우터에서 발생하는 또 다른 불공정성 현상은 동일한 플로우 특성을 가지는 TCP와 같은 반응 플로우 사이에서도 발생된다. TCP 플로우 경우 식 (1)과 같이 TCP의 전송률을 계산 할 수 있다. 이 식에서 $S$는 패킷의 크기이며 $R$은 RTT 값, $p$는 평균 패킷 폐기율 값이다. TCP의 전송률을 결정하는 파라메터 값 중 RTT 값의 차이에 따라 불공정성 문제가 발생하게 된다. 식 (1)의 TCP 전송률 공식에서와 같이 RTT 값이 크면 전송률이 낮아지고 RTT 값이 작으면 전송률이 커지는 것을 알 수 있다[13].

$$T \le \frac{1.5\sqrt{2/3} * S}{R * \sqrt{p}} \quad (1)$$

일반적으로 RTT 값이 작은 플로우와 RTT 값이 큰 플로우가 경쟁을 하면 거의 모든 대역폭을 RTT 값이 작은 플로우가 차지하는 현상이 발생한다. 이로 인한 네트워크의 불공정성 문제는 네트워크 전체의 혼잡상황을 악화 시키고 불필요한 패킷 폐기를 발생시켜 네트워크의 유용성이 떨어뜨린다. 따라서 이러한 불공정성 문제를 해결하여 네트워크 혼잡상황을 효과적으로 제어할 수 있는 새로운 큐 관리 알고리즘이 필요하다.

## 3. 공정성 향상을 위한 새로운 AFQM 알고리즘

본 논문에서는 인터넷의 혼잡상황의 주요 원인이 되는 플로우간의 불공정성 문제를 해결한 새로운 큐 관리 알고리즘인 AFQM(Approximate Fair Queue Management) 알고리즘을 제안하였다. AFQM 알고리즘은 임의의 주기 동안 라우터 큐에 저장되는 각 플로우의 패킷 수를 카운트하여 공정성을 위반하는 플로우에게 추가적인 패킷 폐기 확률을 적용하여 벌하는 것이다. 그림 2는 라우터에서 AFQM 알고리즘과 AQM 알고리즘이 상호 동작하는 라우터의 구조를 보여주고 있다.

AFQM 알고리즘은 RED와 같은 AQM 알고리즘에게

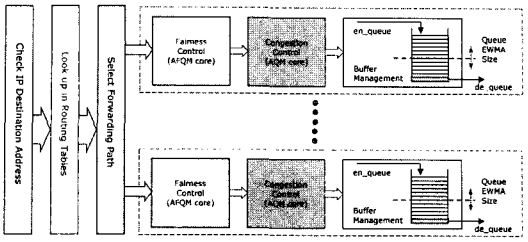


그림 2 AFQM 알고리즘을 이용한 라우터의 구조

공정성 향상을 위하여 공정성 조절 매개변수(fairness adjustment parameter)인 $P_{fa}$값을 제공하여 공정성을 향상시킨다. AFQM 알고리즘의 기본 동작은 라우터에 도착한 패킷이 라우터의 물리적인 큐에 들어가기 전에 그림 2와 같이 혼잡제어 동작을 하는 AQM 알고리즘 앞쪽에서 동작을 하며 공정성 향상을 위해서 계산된 공정성 조절 매개변수 $P_{fa}$정보를 가지고 AQM 알고리즘과 상호 동작하여 공정성을 보장한다.

### 3.1 공정성 조절 매개변수

AFQM 알고리즘은 현재 혼잡상황에 대하여 플로우 $i$가 공정하게 전송해야 하는 최적의 전송률인 공정분배율(fair share rate) $rate_{fair}$를 플로우 $i$의 현재 전송률인 $rate_i$와 공정성 조절 매개변수 $P_{fa}$로 식 (2)와 같은 수식으로 표현할 수 있다.

$$rate_{fair} = rate_i \cdot (1 - P_{fa}) \quad (2)$$

$rate_i$는 $P_{fa}$를 이용하여 현재 모든 플로우에 공정한 $rate_{fair}$로 조절할 수 있다. 식 (2)를 이용하여 $P_{fa}$는 $rate_{fair}$와 $rate_i$의 관계식인 식 (3)으로 표현할 수 있다.

$$P_{fa} = 1 - \frac{rate_{fair}}{rate_i} \quad (3)$$

현재 혼잡상황에 대하여 공정분배율 $rate_{fair}$는 라우터 출력 링크의 대역폭인 $BW$와 현재 활성화(active) 상태로 동작하고 있는 플로우 수 $n$으로 표현하면 식 (4)와 같다.

$$rate_{fair} = BW/n \quad (4)$$

플로우 $i$의 전송률인 $rate_i$를 패킷 당 크기 $S$와 임의의 주기 $\Delta T$ 그리고 $\Delta T$동안 라우터 큐에 들어온 플로우 $i$의 패킷 수 $N_i$를 이용하여 식 (5)와 같이 표현할 수 있다.

$$rate_i = \frac{S \cdot N_i}{\Delta T} \quad (5)$$

임의의 주기 $\Delta T$는 $\Delta T$동안 라우터 큐에 들어온 전체 패킷 수 $N_T$와 라우터 출력 링크의 대역폭인 $BW$로 식 (6)과 같이 표현할 수 있다.

$$\Delta T = \frac{S \cdot N_T}{BW} \quad (6)$$

공정성 조절 매개변수 $P_{fa}$을 표현하는 식 (3)은 식 (4), (5), (6)를 이용하여 식 (7)으로 표현할 수 있다.

$$P_{fa} = 1 - \frac{N_T}{N_i \cdot n} \quad (7)$$

여기서 공정성 조절 매개변수 $P_{fa}$을 표현하는 식 (7)은 기준이 되는 공정분배율 $rate_{fair}$에서 얼마 이상 불공정하게 벗어난 지를 나타내며 이 값이 공정성 지수(fairness index) 비율하고는 비례하는 관계가 아니다. AFQM 알고리즘의 기본 동작은 공정성 정도의 기준이 되는 공정분배율 $rate_{fair}$보다 더 높은 대역을 차지하는 플로우의 $P_{fa}$ 값이 상대적으로 커져 $P_{fa}$ 값이 작은 플로우보다 패킷 폐기 확률을 높이는 방법이다.

예를 들어 10Mbps의 대역을 2개의 플로우가 점유하여 전송하고 있는 네트워크에서 플로우 1이 8Mbps를 차지하고 플로우 2가 2Mbps를 차지하는 경우 플로우 1은 공정분배율 5Mbps 보다 약 3Mbps를 더 차지한 것이 되고 플로우 2의 경우 3Mbps를 덜 차지한 경우가 된다. 따라서 라우터의 큐 관리 알고리즘은 혼잡상황 시 우선적으로 플로우 1의 패킷을 폐기해야 한다. 하지만 현재 큐 관리 알고리즘은 모든 패킷에 대하여 동일한 확률로 패킷을 폐기하기 때문에 공정성을 제공하지 못한다. 이러한 큐 관리 알고리즘에 공정성 조절 매개변수 $P_{fa}$를 적용하여 패킷 폐기 확률 값을 계산하면 간단히 공정성을 보장 할 수 있다. 식 (7)을 이용하여 각 플로우의 $P_{fa}$를 계산하면 플로우 1의 $P_{fa}$값은 0.375, 플로우 2의 $P_{fa}$ 값은 -1.5를 가진다. 간단히 $P_{fa}$를 RED와 같은 큐 관리 알고리즘의 패킷 폐기 확률에 더하면 플로우 2의 패킷 폐기 확률은 0이 되어 큐 관리 알고리즘에서 패킷이 폐기되지 않는다. 반대로 플로우 1의 경우 0.375 만큼의 확률이 추가되어 패킷이 폐기된다.

### 3.2 AFQM 알고리즘

공정성 조절 매개변수 $P_{fa}$는 라우터에서 공정성을 제공하기 위해서 현재 혼잡상황에 적합한 각 플로우의 전송률을 조절하는 매개변수이다. AFQM 알고리즘에서 $P_{fa}$를 이용하여 동작하는 공정한 혼잡제어 알고리즘은 그림 3과 같다.

```
In  Congestion Situation(Calculation P_fa)
  If(P_fa ≤ 0)
     No Packet Drop
  else
     Packet Drop with P_fa
```

그림 3 AFQM 알고리즘

그림 3과 같이 임의의 주기 $\Delta T$동안 전송된 플로우 $i$의 전송률 $rate_i$가 공정분배율 $rate_{fair}$보다 낮은 전송률을 가질 경우에서는 음수(-)값을 가지며 $rate_{fair}$보다 초과된 값을 가질 경우 초과된 크기에 비례한 1보다 작은 양수(+)값을 가질 것이다. AFQM 알고리즘에 의해서 계산된 $P_{fa}$값은 라우터 혼잡제어 알고리즘에서 공정성을 제공하기 위해서 사용되어 지며, RED와 같은 AQM 알고리즘과는 그림 4와 같이 상호동작한다.

```
For Each Packet Arrival
     Calculate the Fairness Adjustment Parameter P_fa
     Calculate the Average Queue Size(EWMA) avg
  If(avg < min_th)
        No Packet Drop
  elseIf(min_th ≤ avg ≤ max_th)
        Calculate the Packet Drop Probability P
        P_tw = P + P_fa
  elseIf(max_th ≤ avg)
        Drop the Arriving Packet
```

그림 4 AFQM 알고리즘과 RED 알고리즘의 상호동작

그림 4와 같이 현재 라우터에 들어온 플로우 $i$의 평균 큐 크기가 최소임계 값($min_{th}$)보다 작을 경우, 즉 혼잡상황이 없는 구간에서는 패킷 폐기 없이 처리된다. 평균 큐 크기가 최소임계 값($min_{th}$)과 최대임계 값($max_{th}$) 사이에 위치한 혼잡제어 구간에서는 높은 대역폭을 차지하는 플로우나 혼잡제어 동작을 하지 않는 비반응 플로우에 대해서 AFQM 알고리즘에서 계산한 $P_{fa}$을 이용하여 공정성을 보장하는 혼잡제어를 한다. 즉, 플로우 $i$의 패킷이 높은 대역폭을 차지하거나 비반응 플로우일 경우 $P_{fa}$값은 양수 값을 가지므로 현재 혼잡상황에 대한 RED 알고리즘의 패킷 폐기 확률 $P$에 추가적인 벌칙으로 $P_{fa}$ 만큼 패킷 폐기율을 증가 시킨다. 반대로 플로우 $i$의 패킷이 다른 플로우에 비하여 상대적으로 낮은 대역폭을 차지하는 경우 $P_{fa}$ 값은 음수 값을 가지며 불공정한 정도에 비례한 값을 패킷 폐기 확률 $P$ 값에서 줄여 패킷 폐기율을 감소시키는 동작을 한다. 즉, AFQM 알고리즘은 공정분배율 보다 높은 대역을 차지하는 플로우의 패킷에게는 높은 패킷 폐기율을, 낮은 대역을 차지하는 플로우의 패킷에게는 낮은 패킷 폐기율을 제공하여 공정한 혼잡제어 동작을 하게 한다. AFQM 알고리즘에서 $P_{fa}$의 동작은 그림 5와 같이 높은 대역폭을 차지하는 플로우(high-bandwidth flow)만을 관리하여 낮은 대역폭을 가지는 플로우에게보다 높은 전송률을 가질 수 있도록 패킷 폐기 확률을 조절한다.

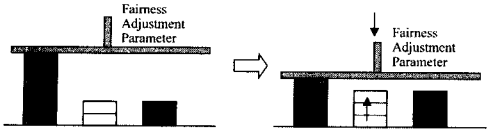


그림 5 Fairness Adjustment Parameter의 동작

### 3.3 AFQM 알고리즘의 $\Delta T$

본 논문에서 설계한 AFQM 알고리즘의 $P_{fa}$을 결정하는데 사용되는 각 플로우들의 갱신(update) 정보의 유용성은 주기 $\Delta T$에 따라 결정된다. 다시 말해 AFQM 알고리즘에서의 $\Delta T$는 AFQM 알고리즘의 동작 반응 속도를 결정하는 주요 매개변수 값에 해당된다. AFQM 알고리즘의 동작 특성상 큐에 저장되었던 각 플로우의 패킷 수로 공정성을 비교한다. 따라서 공정성 비교 정보의 정확성을 위해서 적어도 플로우 $i$의 첫 번째 패킷이 큐에 들어온 후 큐를 빠져나갈 때까지의 플로우 $i$의 패킷에 대한 큐 점유 개수를 그 이후에 도착하는 플로우 $i$의 $P_{fa}$를 계산하는데 이용한다. 이와 같은 조건을 만족시키는 AFQM 알고리즘의 $\Delta T$는 식 (8)을 만족해야 한다. 여기서 $B$는 라우터의 큐 크기이다.

$$\Delta T \geq B/BW \tag{8}$$

일반적인 라우터 설계 시 라우터의 큐 크기는 Delay Bandwidth Product를 기반으로 식 (9)와 같은 관계식으로 설정이 된다.

$$B = BW \times \nabla ay(RTT_a) \tag{9}$$

식 (8)을 AFQM 알고리즘의 주기 $\Delta T$를 식 (7)을 이용하여 다시 표현하면 식 (10)과 같다.

$$\Delta T = \nabla ay(RTT_a) \tag{10}$$

일반적으로 Delay Bandwidth Product에서의 $\nabla ay(RTT_a)$는 망 관리자가 망 특성을 고려해 라우터에 설정한 최대 큐잉 지연(queueing delay) 값이 된다. 따라서 AFQM 알고리즘의 주기 $\Delta T$는 망 관리자의 설정에 따라 망 특성에 의존적으로 설정된다.

AFQM 알고리즘은 $\Delta T$ 주기 동안 현재 공정분배율에서 벗어난 높은 대역폭을 차지하는 플로우를 검출해야 한다. 그러나 플로우 수가 많아질수록 식 (10)에서 설정한 $\Delta T$동안 높은 대역폭을 가지는 플로우가 검출될 확률이 상대적으로 낮아진다. 즉, 플로우의 수가 많아짐에 따라 주기 $\Delta T$동안에 높은 대역폭을 차지하는 플로우를 검출할 비교 표본의 개수가 적어지기 때문이며 이로 인하여 높은 대역폭을 차지하는 플로우에 대한 판단 조건의 정확도가 떨어진다. 따라서 현재 활성화된 플로우 수가 증가할 경우 보다 정확한 판단을 위해서 $\Delta T$를 활성화된 플로우의 수에 비례적으로 증가시킬 필요가 있다. AFQM 알고리즘에서는 플로우 수의 증가 시 높은 대역폭을 차지하는 플로우의 판단의 정확도를 높이기 위해서 식 (11)과 같은 주기 $\Delta T$의 조건을 개선한 새로운 주기 $\Delta T$ 계산 방법을 이용하였다.

$$\Delta T = RTT_a \cdot (1 + \frac{n}{B}) = RTT_a + \frac{n}{BW} \tag{11}$$

$\Delta T$는 현재 활성화된 플로우의 수 $n$에 비례하고 라우터의 출력 링크의 대역폭에 반비례한 크기를 가진다.

### 3.4 AFQM 알고리즘의 오버헤드

AFQM 알고리즘은 $\Delta T$동안 활성화된 플로우의 상태 정보를 저장하고 관리해야 하는 오버헤드를 가지고 있다. 가장 이상적인 공정성을 제공하는 FQ(Fair Queueing) 알고리즘의 경우 모든 플로우의 상태 정보를 저장하고 관리해야 하는 오버헤드로 인하여 확장성 및 실용성이 떨어지는 문제와 구현의 어려움을 가지고 있다. 본 논문에서 제안한 AFQM 알고리즘은 $\Delta T$ 동안 활성화된 플로우의 상태정보를 관리하기 때문에 기존의 FQ 알고리즘이 가지는 오버헤드 보다 상당히 적은 오버헤드를 가지고 있다. 즉 $\Delta T$ 동안 활성화된 플로우의 정보만을 이용하여 다음 주기 $\Delta T$까지 공정성 조절 매개변수를 결정하는 정보로 이용한다. 일반적으로 $\Delta T$ 동안 계산된 활성화된 플로우 수는 현재 라우터를 통해 지나가는 전체 플로우의 수 보다 적다. 이를 알아보기 위해 1000초 동안 180개의 TCP 플로우를 순차적으로 전송하였을 때 AFQM 알고리즘의 활성화 플로우 판별 기법으로 주기 $\Delta T$ 동안 활성화된 플로우으로 판별되는 플로우의 수를 계산하면 그림 6과 같다.

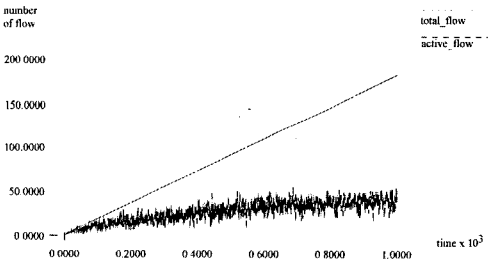


그림 6 전체 플로우 수와 활성화된 플로우 수의 비교

그림 6의 결과와 같이 AFQM 알고리즘의 활성화된 플로우를 관리하는 오버헤드의 복잡도는 전체 플로우 수 $N$에 비례한 $O(N)$이 아닌 $O(log\ N)$의 오버헤드를 가진다. 따라서 기존의 FQ 방법의 오버헤드와 비교하여 상당히 작은 오버헤드를 가짐을 알 수 있다.

## 4. 성능 시험 및 평가

본 장에서는 라우터에서 발생하는 불공정성 문제를 해결하여 네트워크 혼잡상황을 효과적으로 제어할 수 있는 새로운 큐 관리 알고리즘인 AFQM 알고리즘의 성능을 평가하기 위해 아래와 같은 사항의 성능을 ns (Network Simulator)를 이용하여 시험 및 평가하였다 [14].

i) 반응 플로우와 비반응 플로우 사이의 공정성

ii) 서로 다른 RTT 값을 가지는 호스트사이의 공정성

iii) $\Delta T$ 변화에 따른 AFQM 알고리즘 성능변화

### 4.1 반응 플로우와 비반응 플로우 사이의 공정성

본 논문에서 제안한 AFQM 알고리즘의 반응 플로우와 비반응 플로우 사이의 공정성 시험을 위해 그림 7과 같은 네트워크 시뮬레이션 환경을 설정하였다. 혼잡상황이 발생하는 라우터 R1에 AFQM 알고리즘과 RED 알고리즘을 적용하여 시험하였다. 대표적인 반응 플로우로는 TCP를 비반응 플로우로는 UDP를 설정하여 아래와 같은 2가지 시나리오에 대하여 실험하였다.

i) 10개의 TCP 플로우와 1Mbps를 가지는 UDP 플로우 수를 1개에서 10개까지 증가

ii) 10개의 TCP 플로우와 1개의 UDP 플로우 전송률을 1Mbps에서 10Mbps까지 증가

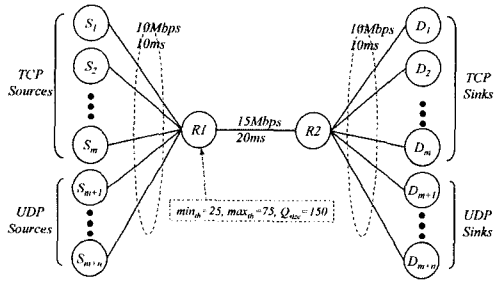


그림 7 네트워크 시뮬레이션 환경

본 논문의 시뮬레이션에서 이용한 UDP 플로우의 설정은 CBR(Constant Bit Rate) 트래픽 타입으로 burst 시간은 2초, idle 시간은 1초로 설정하였다. TCP 플로우의 설정은 FTP 서비스를 응용서비스로 이용하여 윈도우 크기를 무한대로 설정하였다. 각 트래픽의 패킷 크기는 1024byte로 설정하여 시험하였다.

첫 번째 시나리오인 10개의 TCP 플로우와 1개에서 10개까지 플로우 수가 증가하는 UDP 플로우의 공정성 실험은 200초 동안 TCP 플로우와 UDP 플로우를 순차적으로 전송하여 전체 대역폭에서 TCP 플로우가 차지하는 비율과 UDP 플로우가 차지하는 비율을 계산하였다. RED 알고리즘을 사용할 때와 본 논문에서 제안한 AFQM 알고리즘을 사용할 때의 두 가지 특성을 가지는 플로우의 전체 대역폭 비율은 그림 8과 같다.

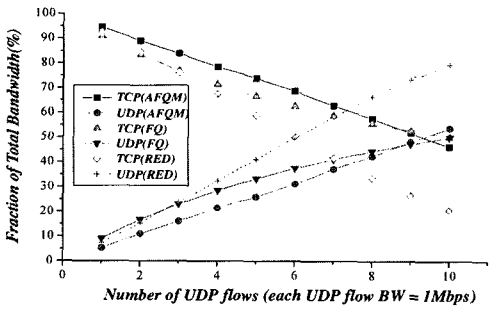


그림 8 TCP 플로우와 UDP 플로우의 공정성 시험 i)

RED 알고리즘을 사용할 경우 그림 8과 같이 UDP 플로우가 6개 이상 전송되면서부터 전체 대역폭에서 차지하는 비율이 50% 이상을 초과하고 있음을 확인할 수 있다. 이 원인은 UDP 플로우의 경우 비반응 플로우이기 때문에 RED 알고리즘의 혼잡제어 동작에도 불구하고 UDP 플로우의 전송률로 계속 전송하기 때문이다. 그리고 TCP 플로우의 전송률이 상대적으로 적어지는 원인은 UDP 플로우의 과도한 트래픽 유입으로 인한 RED 알고리즘의 평균 큐 크기가 상대적으로 커져 TCP 플로우와 같은 반응 플로우에게 높은 패킷 폐기 확률 값이 적용되기 때문이다. AFQM 알고리즘의 경우 그림 8에서와 같이 플로우의 수에 비례적으로 계산한 공정 분배율(fair share)과 근사한 성능을 보이고 있다. 즉 공정 분배율 이상으로 패킷 수가 많아지는 플로우에 대하여 AFQM 알고리즘은 더 높은 패킷 폐기율을 적용한다. 그리고 TCP 플로우중 비례적으로 전송률이 낮은 플로우에 대해서는 불공정한 전송률 차이에 비례하여 패킷 폐기율을 낮추어서 폐기하기 때문에 상대적인 폐기율이 감소한다. 따라서 AFQM 알고리즘을 이용할 경우 공정 분배율로 서로 다른 특성을 가지는 플로우 사이에 불공정성 문제없이 공정한 혼잡제어를 한다. 첫 번째 시나리오에서 실험 TCP와 UDP 플로우의 실제 전송률은 표 1과 같다.

두 번째 시나리오는 불량 사용자(misbehaving user)에 대한 공정성 보장을 알아보기 위해서 10개의 TCP 플로우와 1Mbps에서 10Mbps의 다양한 대역을 가지는 1개의 UDP 플로우 사이의 공정성에 관한 실험을 하였다. 첫 번째 시나리오와 같은 실험 조건에서 UDP 플로

우의 전송률을 1Mbps에서 10Mbps로 증가시키면서 전송률을 비교해본 결과 그림 9와 같은 결과를 확인할 수 있다.

표 1 TCP 플로우와 UDP 플로우의 공정성 시험 i) 결과

| | AFQM | | RED | | AFQM | RED |
|---|---|---|---|---|---|---|
| # of UDP | TCP | UDP | TCP | UDP | Total | Total |
| 1 | 12.74 | 0.72 | 12.54 | 0.99 | 13.46 | 13.53 |
| 2 | 11.48 | 1.41 | 10393 | 1.96 | 12.89 | 12.90 |
| 3 | 10.43 | 2.00 | 9.31 | 2.92 | 12.43 | 12.23 |
| 4 | 9.42 | 2.59 | 8.03 | 3.85 | 12.00 | 11.88 |
| 5 | 8.79 | 3.10 | 6.81 | 4.76 | 11.89 | 11.57 |
| 6 | 8.04 | 3.65 | 5.69 | 5.67 | 11.69 | 11.35 |
| 7 | 7.24 | 4.29 | 4.68 | 6.54 | 11.53 | 11.23 |
| 8 | 6.55 | 4.82 | 3.76 | 7.41 | 11.37 | 11.17 |
| 9 | 5.87 | 5.46 | 2.97 | 8.18 | 11.33 | 11.15 |
| 10 | 5.20 | 6.04 | 2.30 | 8.85 | 11.25 | 11.14 |

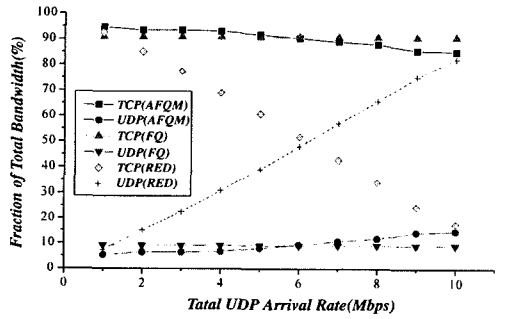


그림 9 TCP 플로우와 UDP 플로우의 공정성 시험 ii)

그림 9와 같이 RED 알고리즘을 사용하는 경우 불량 사용자인 UDP 플로우의 전송률을 1Mbps 이상으로 전송 시 다른 플로우들은 상대적으로 불공정한 서비스를 받게 된다. 그림 7의 라우터 R1에서 처리되는 트래픽의 비율이 한 개의 UDP 플로우에 의해서 모두 점유되고 있음을 확인할 수 있다. 높은 전송률을 보내는 한 개의 플로우로 인하여 라우터 R1에서의 RED 알고리즘의 혼잡상황을 판별하는 평균 큐 크기는 상대적으로 높게 계산되고 결국 높은 패킷 폐기율은 정상 사용자인 10개의 TCP 플로우에 적용되어 불공정성을 더욱 악화시키는 결과를 가져온다. 본 논문에서 제안한 AFQM 알고리즘의 경우 큐 안에 들어 있는 전체 패킷 수에서 특정 플로우의 패킷 수를 비례적으로 계산하여 불공정성 여부를 판별한다. 따라서 높은 전송률을 가지는 플로우의 경우 다른 플로우보다 높은 비율로 큐를 차지하고 있기 때문에 패킷이 폐기 될 확률이 상대적으로 커진다. 반대로 나머지 플로우의 경우 큐 안의 전체 분포에서 낮은 비율로 큐를 점유하고 있기 때문에 혼잡제어 모듈에서 계산된 패킷 폐기 확률이 상대적으로 낮아진다. 그림 9의 AFQM 알고리즘은 공정분배율을 가지는 FQ과 비슷한 분포로 공정성을 보장하고 있음을 알 수 있다. 두 번째 시나리오에서 실험 TCP와 UDP 플로우의 실제 전송률은 표 2와 같다.

표 2 TCP 플로우와 UDP 플로우의 공정성 시험 ii) 결과

| | AFQM | | RED | | AFQM | RED |
|---|---|---|---|---|---|---|
| UDP rate | TCP | UDP | TCP | UDP | Total | Total |
| 1 Mbps | 12.74 | 0.72 | 12.54 | 0.99 | 13.46 | 13.53 |
| 2 Mbps | 12.72 | 0.88 | 11.12 | 1.97 | 13.60 | 13.09 |
| 3 Mbps | 12.88 | 0.89 | 10.03 | 2.93 | 13.76 | 12.96 |
| 4 Mbps | 12.64 | 0.92 | 8.66 | 3.88 | 13.56 | 12.54 |
| 5 Mbps | 12.57 | 1.10 | 7.50 | 4.81 | 13.67 | 12.31 |
| 6 Mbps | 12.29 | 1.28 | 6.16 | 5.72 | 13.57 | 11.89 |
| 7 Mbps | 11.81 | 1.43 | 4.95 | 6.61 | 13.23 | 11.56 |
| 8 Mbps | 11.87 | 1.59 | 3.86 | 7.46 | 13.46 | 11.32 |
| 9 Mbps | 11.52 | 1.91 | 2.69 | 8.23 | 13.43 | 10.92 |
| 10 Mbps | 11.72 | 2.01 | 1.90 | 8.96 | 13.73 | 10.87 |

표 2에서 AFQM 알고리즘과 RED 알고리즘 사용 시 전체 전송률의 13.73Mbps와 10.87Mbps로 차이가 나는 것을 알 수 있다. 그 이유는 일반적인 RED 알고리즘의 경우 하나의 불량 사용자로 인한 불공정하게 불필요한 높은 패킷 폐기 확률을 가져 AFQM 알고리즘을 이용할 때 보다 10개의 TCP 플로우에 높은 패킷 폐기 확률이 적용되어 네트워크 활용도(Utilization)가 감소되기 때문이다.

### 4.2 서로 다른 RTT 값을 가지는 호스트사이의 공정성

반응 플로우 안에서도 각 플로우별로 가지는 플로우 특성에 따라 불공정성 문제는 발생할 수 있다. TCP 플로우 경우 TCP의 전송률을 결정하는 파라메터 값 중 RTT 값의 차이에 따라 불공정성 문제는 발생한다. 일반적으로 RTT 값이 작은 플로우와 RTT 값이 큰 플로우가 경쟁을 하면 거의 모든 대역폭을 RTT 값이 작은 플로우가 차지하는 현상이 발생한다. 본 논문에서 제안한 AFQM 알고리즘의 공정성 실험을 위해서 그림 10과 같은 네트워크 환경을 설정하여 실험 하였다. 총 30개의 플로우를 전송하여 RTT 값이 작은 1개의 플로우와 동일한 RTT값을 가지는 29개의 플로우를 전송하였을 때 각 플로우의 실효전송률(goodput)을 비교하였다. 실험 설정은 S(1)과 D(1)을 지나가는 Flow 1의 플로우를 RTT 값이 작을 플로우로 선택하여 RTT = 34ms를 가지게 설정 하였고 나머지 29개의 플로우는 RTT = 1024ms를 가지게 설정 하였으며 200/30(실험시간/플로우 수)초 간격으로 플로우를 순차적으로 전송하였다.

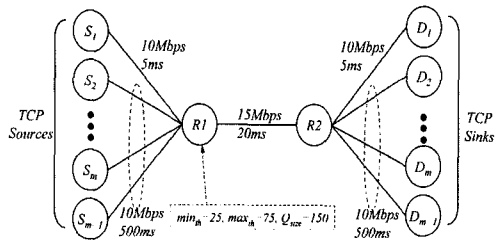


그림 10 네트워크 시뮬레이션 환경

200초 동안 실험한 각 플로우의 실효 전송률은 표 3과 같다. 표 3에서와 같이 RED 알고리즘을 사용하는 경우 RTT값을 34ms를 가지는 TCP 플로우가 전체 대역폭의 2.31Mbps의 대역을 차지하고 있다. RTT 값이 1024ms를 가지는 29개의 TCP 플로우 경우 전체 대역폭에서 약 0.23Mbps 대역폭을 차지하고 있다.

표 3 RTT 값에 따른 불공정성 시험 결과

| Flow ID | AFQM | RED | Flow ID | AFQM | RED |
|---|---|---|---|---|---|
| 1 | 1.1092 | 2.3107 | 17 | 0.3068 | 0.2559 |
| 2 | 0.3283 | 0.2433 | 18 | 0.3172 | 0.2637 |
| 3 | 0.3222 | 0.2389 | 19 | 0.3287 | 0.2423 |
| 4 | 0.3253 | 0.2328 | 20 | 0.3328 | 0.2600 |
| 5 | 0.3274 | 0.2184 | 21 | 0.3206 | 0.2186 |
| 6 | 0.3232 | 0.2485 | 22 | 0.3147 | 0.2480 |
| 7 | 0.3261 | 0.2654 | 23 | 0.3106 | 0.2405 |
| 8 | 0.3171 | 0.2241 | 24 | 0.3307 | 0.2569 |
| 9 | 0.3330 | 0.2553 | 25 | 0.3280 | 0.2161 |
| 10 | 0.3152 | 0.2669 | 26 | 0.3292 | 0.2512 |
| 11 | 0.3290 | 0.2099 | 27 | 0.3315 | 0.2386 |
| 12 | 0.3285 | 0.2547 | 28 | 0.3328 | 0.2455 |
| 13 | 0.3244 | 0.2436 | 29 | 0.3327 | 0.2442 |
| 14 | 0.3198 | 0.2694 | 30 | 0.3163 | 0.2736 |
| 15 | 0.3299 | 0.2353 | | | |
| 16 | 0.3207 | 0.2303 | **Total** | **10.5119** | **9.4026** |

RTT 값에 따른 불공정성 문제를 확인하기 위해 전체 실효 전송률에서 각 플로우가 차지하는 전송률의 비율은 그림 11과 같이 표현할 수 있다. 그림 11과 같이 RTT 값이 작은 플로우가 전체 실효 전송률 비율에서 25%의 대역을 차지하고 있다. 여기서 3.3%(100%/30개)의 공정 분배율과 비교해보면 RTT 값이 작은 플로우가 상당히 큰 비율을 차지하고 있음을 알 수 있다. AFQM 알고리즘의 경우 RED 알고리즘보다 RTT 값이 작은 플로우의 실효 전송률 비율을 약 10% 정도로 낮추었다. 다른 29개의 플로우도 RED 알고리즘의 경우보다 약 0.5%의 실효 전송률 비율이 증가했음을 확인할 수 있다. 그림 11을 통해 본 논문에서 제안한 AFQM 알고리즘이 RTT 값의 차이에 따른 불공정성 현상을 개선했음을 확인할 수 있다.

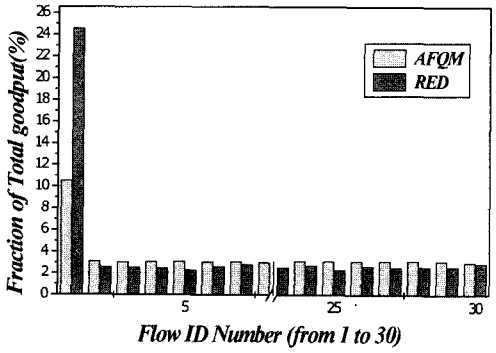


그림 11 RTT 값에 따른 실효 전송률 비율

### 4.3 $\Delta T$변화에 따른 AFQM 알고리즘 성능변화

AFQM 알고리즘의 성능은 공정성 조절 매개변수 $P_{fa}$을 결정하는데 사용되는 각 플로우들의 갱신 정보의 유용성에 따라 결정된다. 공정성 조절 매개변수 $P_{fa}$의 유용성은 얼마나 자주 플로우 정보를 갱신 하는가 그리고 갱신 정보가 전체 플로우의 상황을 정확히 유추 할 수 있는가에 따라 달라진다. 공정성 조절 매개변수의 정확성과 정보의 유용성은 갱신 주기 $\Delta T$에 따라 결정되며 두 특성은 어느 정도 trade-off 관계를 가지고 있다. 정확성을 높이기 위해서는 충분한 시간 동안 플로우의 상태를 저장하여 각 플로우의 특성을 유추해야 하지만 오랜 시간동안 정보를 수집할 경우 즉시 공정성 조절을 하지 못하기 때문에 공정성 조절 매개변수 $P_{fa}$의 유용성은 상대적으로 떨어진다. 이와는 반대로 주기를 짧게 설정 할 경우 즉시 공정성을 조절하기 때문에 공정성 조절 매개변수 $P_{fa}$의 유용성은 커지지만 짧은 시간 동안 플로우의 상태를 판별하기 때문에 정확성은 떨어지는 문제를 가지고 있다. 본 절에서는 AFQM 알고리즘의 성능을 결정하는 중요 매개변수인 갱신 주기 $\Delta T$ 변화에 따른 성능변화를 시험해 본다. 그림 7과 같은 네트워크 환경에서 5개의 TCP 플로우와 5Mbps를 가지는 1개의 UDP 플로우를 순차적으로 전송하였고 주기 $\Delta T$에 0.0001배부터 1000배 까지 10배 단위로 크기를 조절하였을 때 TCP 플로우의 전체 전송률과 UDP 플로우의 전송률을 비교하였다. 본 논문에서 제안한 $\Delta T$의 기본 설정값인 식 (10)의 $\Delta T = \nabla ay(RTT_a)$조건의 기본값 $RTT_a$의 크기를 그림 12의 X 축과 같이 조절하였을 때 AFQM 알고리즘의 성능의 변화는 그림 12와 같다.

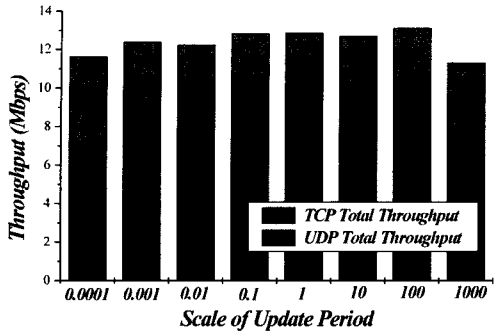


그림 12 $\Delta T$의 변화에 따른 AFQM 알고리즘의 성능 변화

전체 전송률은 $\Delta T$의 변화에 비하여 크게 변화는 없다. 그러나 $\Delta T$가 기본 설정값 $RTT_a$보다 너무 작거나 너무 클 경우 불공정성이 발생하며 전체 전송률이 다소 떨어지는 것을 확인할 수 있다. 즉 $\Delta T$ 크기가 너무 작은 경우 공정성을 판단하기위한 정확성이 떨어져 성능이 감소하였으며 $\Delta T$ 크기가 너무 큰 경우 공정성 조절 매개변수 $P_{fa}$을 결정하는 정보의 유용성이 떨어져 부적절한 공정성 조절 매개변수가 생성되게 된다. 그림 4.6의 결과와 같이 본 논문에서 제안한 AFQM 알고리즘의 성능을 결정하는 주기 $\Delta T$의 값을 초기 기준으로 설정한 $RTT_a$에 근접하게 설정할 경우 전체적으로 안정된 성능을 나타내고 있음을 알 수 있다.

## 5. 결 론

IETF에서는 효율적인 혼잡제어를 위하여 라우터 알고리즘으로 RED 알고리즘과 같은 큐 관리 알고리즘을 권고하고 있다. 하지만 RED 알고리즘은 UDP와 같이 양 종단간 혼잡제어를 하지 않는 플로우나 RTT 값이 작은 TCP 플로우로 인하여 발생하는 불공정성 문제를 가지고 있으며 이로 인하여 혼잡상황이 악화되는 문제를 가지고 있다.

본 논문에서는 이러한 문제를 해결하기 위해 새로운 공정성 향상 큐 관리 알고리즘인 AFQM 알고리즘을 제안하였다. AFQM 알고리즘은 공정분배율 이상의 대역폭을 차지하는 플로우를 판별하여 공정성을 보장하며 적은 오버헤드와 구현의 용이성을 가지고 있다. AFQM 알고리즘은 RED와 같은 AQM 알고리즘에게 공정성 향상을 위하여 공정성 조절 매개변수 $P_{fa}$값을 이용하여 혼잡상황이 발생하면 공정하게 패킷을 폐기하여 라우터에서의 공정성을 향상시킨다. 즉, 비반응 플로우나 불량 사용자 플로우를 적은 오버헤드로 관리하여 라우터의 불공정성으로 인하여 발생되는 혼잡상황의 악화를 방지하는 알고리즘이다. 시뮬레이션 결과를 통하여 AFQM 알고리즘은 기존 방법보다 적은 오버헤드로 기존의 RED 알고리즘의 불공정성을 개선하여 안정되고 공정한 혼잡제어 동작을 수행하는 것을 확인하였다.

향후 연구 과제로는 AFQM 알고리즘과 이상적으로 동작 할 수 있도록 새로운 형태의 인터넷 혼잡제어 알고리즘을 연구하는 것이며, 인터넷 QoS를 보장하는 응용 서비스 기술에 공정성을 보장하는 AFQM 알고리즘을 적용하여 다양한 네트워크 환경에서 시험 및 성능을 평가해야 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 SIGCOMM 88, 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] Demers, A., Keshav, S. and Shenker, S., "Analysis and Simulation of a Fair Queueing Algorithm," Journal of Internetworking Research and Experience, October 1990. Also in Proceedings of ACM SIGCOMM 89.

[6] 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.

[7] McKenny, P., "Stochastic Fairness Queueing," Proceedings of INFOCOM 90, pp733-740, June, 1990.

[8] Legout, A. and Biersack, E. W., "Revisiting the Fair Queueing Paradigm for End-to-End Congestion Control," IEEE Network Magazine, 2002, September.

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

[10] Lin, D. and Morris, R., "Dynamics of random

early detection," ACM SIGCOMM 97, pp 127-137, October 1997.
[11] 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.
[12] Mahajan, R. and Floyd, S. "Controlling High-Bandwidth Flows at the Congested Router," ACIRI, Berkeley, California, November 2000.
[13] Padhy, J., Firoiu, V., Towsley, D., and Kurose, J., "A Stchastic Model of TCP Reno Congestion Avoidance and Control," Technical Report CMPSCI TR 99-02, Univ. of Massachusetts, Amherst, 1999.
[14] UCB/LBNL/VINT, "Network Simulator-ns (Version 2.1b9a)," http://www-mash.cs.berkeley.edu/ns/.

구 자 헌

1999년 광운대학교 전자통신공학과 학사 2001년 광운대학교 전자통신공학과 석사 2001년~현재 광운대학교 전자통신공학과 박사 과정. 2003년~현재 Inno Wireless 정보통신연구소 연구원. 관심분야는 컴퓨터 통신, 인터넷 QoS, 멀티미디어통신

최 웅 철

1989년 서울대학교 컴퓨터공학과 학사 1991년 서울대학교 컴퓨터공학과 석사 2001년 Ph.D., Computer Science, University of Illinois, Urbana-Champaign 2001년 Research Scientist, Telcordia Technologies(formerly Bellcore), Morristown, NJ. 2002년~현재 광운대학교 컴퓨터공학부 전임강사. 관심분야는 Wireless/Mobile computing/network infrastructure, QoS, Network Management

정 광 수

정보과학회논문지 : 정보통신
제 30 권 제 2 호 참조