논문 2012-49TC-3-4

NetFPGA 플랫폼 기반 RED스케줄러 구현 및 TCP 성능평가

(Implementation and TCP Performance Measurement of RED scheduler using NetFPGA platform)

오 민 경*, 민 석 홍*, 김 병 철**, 이 재 용**

(Minkyung Oh, Seokhong Min, Byungchul Kim, and Jaeyong Lee)

요 약

최근 다양한 사용자 요구사항의 증가로 인하여 인터넷을 이용한 다양한 응용 제공에 대한 필요성이 증가하고 있다. 그러나, 단순하게 망에서 최소한의 기능만 제공하여 노드 간의 연결성만을 제공하기 위한 연구망으로 탄생한 인터넷의 근본적인 문제로 인하여 오늘날 우리가 필요로 하는 여러 가지의 다양한 응용 제공에 여러 제약들이 존재한다. 이러한 제약들은 라우팅 확장성, 이동성, 보안 및 QoS 제공 등 여러 가지 측면에서 새로운 접근을 필요로 하며, 국내외 여러 분야에서 이러한 제약들을 해결하기 위한 많은 연구들이 진행되고 있다. 본 논문에서는 이러한 제약들 중 하나인 망에서의 QoS 제공에 대한 연구의 일환으로 NetFPGA 플랫폼 기반의 라우터를 이용하여 특정 확률에 따라 패킷을 폐기하는 RED(Random Early Detection) 스케줄러를 구현하고 테스트베드를 구축하였다. 이를 통해 네트워크가 혼잡하여 라우터에서 발생하는 TCP 트래픽의 패킷 손실로 인한 망의 전역동기화(Global Synchronization) 현상을 방지하게 된다. 일반 라우터의 Drop Tail 방식과의 TCP 성능 비교 실험을 통하여 RED 스케줄러가 네트워크가 혼잡한 상황에서 TCP 트래픽의 전송 성능을 향상시켜 망의 품질을 효율적으로 향상시킬 수 있음을 확인하였다.

Abstract

With the increase of various user's requirements, lots of interesting applications on the Internet have been emerging recently. However, Internet has many limitations for providing upcoming new services because it was only designed to provide basic connectivity between research networks and simplified forwarding functions at the first time. Internet has many problems in the aspects of routing scalability, mobility, security and QoS, so lots of researches are being actively performed in many countries to solve these problems. In this paper, we implement RED(Random Early Detection) scheduler using NetFPGA platform and local testbed to provide active queue management. Using the implemented RED scheduler, packets are dropped according to the specified drop probability, so Global Synchronization coming from simultaneous TCP segment losses in a congestion condition can be prevented. With the comparison to the Drop-Tail scheme in the basic router, we show TCP performance can be enhanced in the congestion situation using the NetFPGA-based RED scheduler.

Keywords : RED, TCP, Congestion Avoidance, Scheduler, NetFPGA

Ⅰ. 서 론

현재의 인터넷은 1962년 패킷스위칭과 1974년 네트워크와 네트워크를 엮는 "inter-net"의 기본 개념이 정립된 이후 1978년 TCP/IP 프로토콜이 인터넷의 표준이 되었다. 글로벌 네트워크의 대표인 인터넷 성공의 이면에는 현 인터넷의 시초가 된 미 국방성 주도의 ARPANet이란 대규모 시험 네트워크를 1969년 선행도입하여 1990년대에 NSF 주도로 NSFNet 학술연구망으로 발전시킨 노력이 있었다[1]. 단계별로 각종 프로토콜 및 응용기술을 시험 운영 발전시킴으로서 현재의 글로벌 인터넷 구조를 가능하게 했다.

그러나 2000년대에 들어서면서부터 통신환경의 급격한 변화 및 다양한 사용자 요구사항의 증대로 인해 현

* 학생회원, ** 평생회원, 충남대학교 정보통신공학과
(Chungnam National University)

※ This study was financially supported by research fund of Chungnam National University in 2011
접수일자: 2011년11월18일, 수정완료일: 2012년3월19일

재의 인터넷이 갖는 근본적인 문제에 대해 심각한 고민을 하기 시작하였으며, 최근에는 그 연장선 중 하나로 Clean Slate 설계 방법에 기반을 둔 미래인터넷 연구가 국내외로 활발히 진행되고 있다[2]. 차세대네트워크는 이러한 근본적인 구조를 바꾸려는 시도가 아니라, 다양한 망에서 제공되는 서비스들을 통합하려는 개념이라고 볼 수 있다. 현재의 통신망의 발전을 위해선 연구자들이 통신망의 구조를 깊이 이해하고 직접적인 실험과 연구가 필요하다. 그러나 현실적으로 연구자가 실제 통신망을 운용하는 것은 불가능하다. 이를 극복하기 위해 학계에서 논의되고 있는 것이 하드웨어 또는 소프트웨어를 통한 네트워크 가상화이다. 이런 네트워크 가상화는 일반 연구자라도 자신만의 통신망을 구성해 새로운 알고리즘과 기능을 자유롭게 구현할 수 있게 하면서도, 다른 사용자들에게는 전혀 영향을 미치지 않는 장점을 지닌다[3].

본 논문에서는 미래인터넷 연구에서 실험 및 개발을 위하여 자주 사용되고 있는 프로그래머블 플랫폼인 NetFPGA 플랫폼[4]을 이용하여 NetFPGA 플랫폼 기반 라우터에 RED 스케쥴러[5]를 구현하고 로컬 테스트베드를 구축하여 성능을 실험하였다. 현재, 일반적으로 사용되는 라우터는 네트워크상에 혼잡이 발생하면 주기적으로 플로우 구분없이 모든 플로우의 패킷을 폐기하여 전역동기화(Global Synchronization) 현상이 반복적으로 발생되어 단대단 전송을 제어하는 전송계층 프로토콜인 TCP의 성능을 크게 저하시킨다. 본 논문에서 구현한 RED(Random Early Detection) 스케줄러는 이를 방지하기 위한 기법으로 체증회피와 체증제어를 수행하는데, 이 기법은 라우터 버퍼의 평균 큐사이즈를 계산하여 네트워크의 혼잡을 초기에 감지하고 라우터에 들어오는 패킷을 모니터링하여 특정 확률로 조기에 패킷을 폐기하거나 마킹한다. 이때, 확률의 계산은 moving average를 계산하여 이용하는 방식으로 평균 큐사이즈의 선형함수로 구해지며 항상 낮은 큐길이를 유지하고 버스트한 트래픽을 수용할 수 있다. RED스케줄러는 알고리즘이 간단하여 패킷을 폐기하거나 마킹함으로서 혼잡상황을 송신측에 알려주기 때문에 Transport 프로토콜 수정없이 TCP/IP 네트워크에 쉽게 적용할 수 있다.

본 논문에서는 특정 확률에 따라 패킷을 폐기하는 방식의 스케줄러를 구현하였으며 NetFPGA 플랫폼 기반의 로컬테스트베드를 구축하고 일반 라우터의 Drop Tail 방식과 비교하여 TCP 트래픽의 전송 성능에 대하여 비교 분석하였으며 본 논문의 RED 스케쥴러는 네트워크가 혼잡한 상황에서 TCP 트래픽의 전송 성능을 향상시켜 망의 품질을 효율적으로 향상시킬 수 있음을 확인하였다.

본 논문의 II장 에서는 동적 큐 관리기법과 RED스케줄러에 대해 상세히 설명하였고 III장에서 NetFPGA 플랫폼에 대한 소개와 NetFPGA 기반의 RED스케줄러 구현부분을 설명하였다. IV장에서는 실험을 통한 TCP 성능평가에 대해 설명하고 마지막으로 V장에서 실험내용에 대한 결론을 기술하며 본 논문을 마무리 하였다.

Ⅱ. 관련 연구

1. 동적 큐 관리 기법

현재 인터넷 구성의 대부분 Drop Tail방식을 이용하는 라우터로 구성되어 있으며 IP 프로토콜을 기반으로 하는 최선형 서비스를 하고 있다. 지수적으로 증가하는 인터넷 트래픽으로 인하여 발생하는 패킷 손실률의 증가는 네트워크의 성능을 저하시키며 인터넷 서비스의 열화를 초래한다. 현재 라우터에서의 버퍼는 FIFO 패킷 스케쥴링에 의해 관리된다. 만약에 버퍼가 꽉 차게 된다면, 도착하는 패킷들은 모두 폐기된다. 이러한 FIFO 스케줄링 기법은 Drop Tail이라고 불리는 큐 관리 알고리즘에 기반한다. Drop Tail 알고리즘은 알고리즘 자체가 쉬워 구현이 간단하기 때문에 현재 대부분의 인터넷 라우터에서 사용되는 기법이다. 그러나 Drop Tail에는 두 가지 중요한 문제점이 존재하는데, lock-out과 full-queue 현상이다. Lock-out은 Drop Tail에서 몇 개의 연결들이 큐의 공간을 독점하는 현상이다. TCP는 버퍼가 꽉 차게 될 때, 패킷 손실에 의하여 데이터를 전송하는 소스에게 혼잡 신호를 되돌려 보내게 된다. full queue 현상은 Drop Tail 큐 관리 기법 하에서 오랜 시간 지속될 경우 발생하게 된다. 그러므로 라우터에는 버스트한 데이터 트래픽을 받아들일 수 있도록 충분한 버퍼 용량을 갖으면서도, 큐 길이는 작게 유지하는것이 필요하다. 큐가 가득차 생기는 문제의 해결책은 큐가 가득 찰 때 까지 패킷의 폐기를 기다리지 않고 오히려 큐가 가득차기 전에 패킷을 폐기함으로써 송신자가 혼잡을 인지하고 전송량을 줄여 큐가 오버플로우되는 것을 방지하는 것이다. 또한 네트워크 혼잡을 효율적으로 조절하고 네트워크 상태를 안정화하기 위하여 버퍼가 오버플로우가 되기전에 혼잡을 발견하는 것

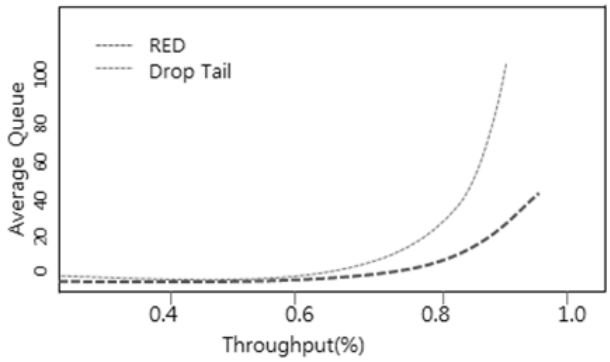

그림 1. Drop Tail과 RED 성능 비교
Fig. 1. Comparison of Drop Tail and RED.

이 필요하다[6].

Drop Tail 알고리즘이 갖고 있는 단점을 극복하기 위한 해결책은 버퍼들이 오버플로우가 발생하기 전에 소스가 혼잡에 대응할 수 있도록 하기 위하여 큐가 꽉 차기 전에 패킷을 폐기하는 것이다. 이러한 메커니즘을 AQM(Active Queue Management)이라고 한다. 그리고 RED알고리즘은 AQM 메커니즘중에 가장 대표적인 방법이라고 할 수 있다[7]. 그림 1은 Drop Tail과 RED의 성능을 비교한 그림이다.

AQM 알고리즘은 보다 정확한 정보들을 이용하여 혼잡이 발생한 라우터에서의 트래픽을 모니터링하고 조절할 수 있다는 장점이 있다. AQM의 또 다른 장점은 혼잡 제어를 TCP 소스에 독립적으로 할 수 있다는 것이다. 그렇기 때문에 AQM 알고리즘은 라우터에서 혼잡을 발견하고 제어하는 수락 제어기와 같이 동작한다고 할 수 있다.

2. RED 스케줄링

Floyd와 Jacobson에 의해 처음 소개된 동적 큐관리 알고리즘인 RED는 IETF에 의해 차세대 네트워크를 위한 기본 동적 큐관리 알고리즘으로 추천받았다. 버퍼가 가득 찼을때 패킷을 폐기하는 드롭 테일과 같은 전통적인 큐 방식과 다르게 RED는 버퍼 오버플로우 이전에 패킷을 폐기한다. 비록 하나의 큐에서 동작하도록 설계되었지만 RED는 능동적으로 큐에 적용되기 때문에 공평한 큐잉과 같은 다중 큐 시스템에서 쉽게 동작될 수 있게 할 수 있다.

가. RED 스케줄러의 목적

RED는 두 가지 중요한 목적을 가진다. 첫째는 혼잡의 지표로 몇가지 파라미터를 이용해 혼잡상황의 큐 관리를 효율적으로 수행한다. 이를 위해서는 혼잡을 일찍 탐지하는 것이 중요하다. 둘째는 평균 큐 크기가 증가하는 것에 따라 확률 함수를 이용하여 랜덤하게 패킷을 폐기시킨다. 이것은 RED가 임의의 연결이나 임의의 버스트한 트래픽에 대해 편향되지 않고 몇 개의 플로우에 독점되는 것을 방지한다.

더욱이 순간적인 큐 크기 대신에 평균 큐 크기에 따라 랜덤하게 패킷을 폐기함으로서 혼잡상황을 더 잘 이해할 수 있다. 만약 평균 큐 크기가 최근 높게 유지된다면 내재된 혼잡이 높을 확률이다. 또한 RED는 평균 큐 길이를 제어함으로써 전통적인 큐 관리가 직면하는 전역동기화(Global Synchronization)와 높은 지연을 미연에 방지한다.

(1) RED 상세 알고리즘

RED 알고리즘은 평균 큐 크기(avg)를 계산하는 단계와 폐기 확률(Pa)을 계산하여 들어오는 패킷의 폐기 여부를 결정하는 단계로 구성된다. RED의 성능을 조절하는 주요 파라미터는 [표 1]과 같이 5가지로 나타낼 수 있다.

평균 큐 길이(avg)는 최소 임계치(minth)와 최대 임계치(maxth)에 의하여 결정이 된다. 평균 큐 길이(avg)

표 1. RED 알고리즘의 파라미터
Table 1. The parameters of the RED algorithm.

파라미터	설명	권고
qlen	최대 버퍼 크기나 큐제한	
min_{th}	패킷이 드롭되는 최소 임계치	5패킷
max_{th}	패킷이 드롭되는 최대 임계치	3 x minth
w_q	평균 큐 크기를 계산하기 위한 가중치	0.002
max_p	최대 드롭 확률	0.1

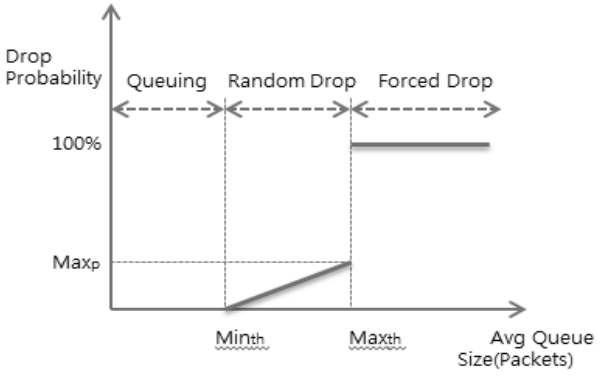

그림 2. RED 패킷 드롭 확률
Fig. 2. RED Packet Drop Probability.

```
# 초기화
Avg ← 0
Count ← -1
# 패킷이 도달할 때 마다 평균 큐사이즈 avg 계산,
        If the queue is nonempty
                Avg ← (1-wq) (m)avg + wq
        Else
                M ← f(time-q_time)
                Avg ←(1-wq)mavg
        If minth ≤ avg < maxth
#       Count 증가시키고, Pa계산
        Pb ← maxp(avg- minth) / (maxth – minth)
        Pa ← Pb /(1-count· Pb)
#       도착확률에 마킹
        Count ← 0
Else count ← -1
#큐가 비어있을 때
 q_time ← time

Saved Variables:
        avg: 평균 큐 사이즈
        q_time: 큐 idle time 시작
        count: 마지막 mark된 패킷 이후의 unmarked된 패킷 수
Fixed parameters:
        Wq : 큐 가중치 (weight)          0.002
        minth: 큐의 최소 임계값          5
        maxth: 큐의 최대 임계값          15
        maxp: Pa의 최대치                0.002(1/50)
Other:
        Pb: 현재 패킷 마킹 확률 (Pa : uniform value)
        q: 현재 큐 사이즈
        time: 현재 시간
        f(t): time t의 선형함수
```

그림 3. 효율적인 RED 알고리즘
Fig. 3. The Efficient RED Algorithm.

가 최소임계치보다 작으면 도착하는 패킷들을 폐기하지 않지만, 만약 평균 큐 길이(avg)가 최대임계치보다 크게되면 새로 들어오는 모든 패킷을 폐기하게 된다.

이를 도식적으로 나타내면 그림 2와 같다. 버퍼의 큐 길이의 최소 및 최대 임계치값에 따라 패킷을 폐기할지 큐에 쌓을지 혹은 확률에 의해 폐기할지를 결정한다.

RED는 지수적 가중 이동 평균치 즉, EWMA (Exponential Weighted Moving Average)방식을 사용하여 평균 큐 길이(avg)를 계산한다. 이때 평균 큐 길이는 식 (1)과 같이 구해진다.

$$avg(t) = (1-w_q)avg(t-1)+w_q q(t) \quad (1)$$

avg(t)는 현재의 평균 큐길이를 말하는데 이전 시간의 큐 길이가 현재 시간의 큐 길이에 반영됨을 알 수 있다. w_q는 가중치를 나타내며, 이 값은 $0 \le w_q \le 1$ 의 범위를 갖는데 Floyd실험에 의하여 가장 최적화된 값은 0.002이다.

Ⅲ. NetFPGA기반의 RED 스케줄러 구현

1. NetFPGA 플랫폼

NetFPGA 플랫폼은 네트워크 관련 학생 및 연구자들

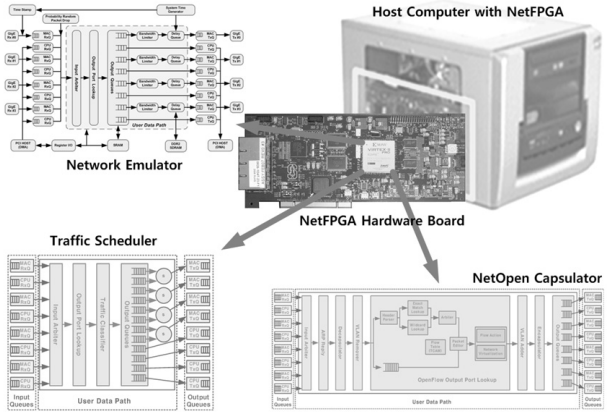

그림 4. NetFPGA 플랫폼
Fig. 4. NetFPGA platform.

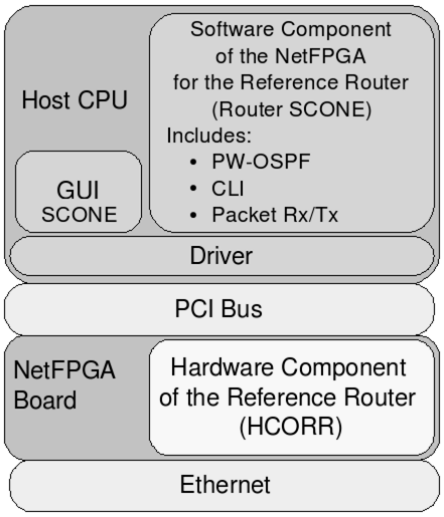

그림 5. NetFPGA 플랫폼의 구조[4]
Fig. 5. Overall Architecture of NetFPGA platform[4].

에게 필요한 교육 및 연구 개발의 환경을 제공하기 위하여 미국의 스탠포드 대학에서 개발되어 네트워크와 관련된 여러 분야에서 세계적으로 이용률이 증가하고 있는 오픈 프로그래머블 플fot폼으로써 그림 4와 같이 일반적으로 사용하는 PC에 NetFPGA 하드웨어가 추가된 형상을 하고 있으며, 그림 5와 같은 구조를 갖는다.

현재, 전 세계적으로 여러 기관에서 1G 이더넷 인터페이스를 지원하는 NetFPGA 플랫폼을 연구개발에 이용하고 있으며, 10G 이더넷 인터페이스를 지원하는 NetFPGA 플랫폼도 출시되었다. NetFPGA 플랫폼은 패킷 처리를 전담하는 전용의 하드웨어와 기타 필요한 어플리케이션을 구현할 수 있는 소프트웨어 패키지로 구성되어 있으며, 일반 PC를 이용하여 어렵지 않게 개발환경을 구축하여 사용할 수 있도록 되어 있다.

NetFPGA 플랫폼의 가장 큰 매력은 전용의 하드웨어를 이용하여 CPU의 사용률 점유없이 line-rate로 고속의 패킷 처리가 가능하다는 점에 있으며, 패킷 처리를 전담하는 NetFPGA 하드웨어는 4개의 기가 비트 이더넷 인터페이스와 PCI 인터페이스가 제공되며, 제공되는 PCI 인터페이스를 통하여 PC에서 NetFPGA 하드웨어를 제어할 수 있도록 설계되어 있다. NetFPGA 플랫폼을 이용하는 개발자는 PC의 어플리케이션과 NetFPGA 하드웨어의 FPGA 로직 구현에만 전념하면 되도록 설계되어 있어 NetFPGA 플랫폼을 이용한 고속의 패킷 처리 시스템 구축에 용이하다는 점 또한 매력이 아닐 수 없다. NetFPGA 플랫폼을 이용하는 개발자는 NetFPGA 플랫폼 하드웨어의 FPGA(Field Programmable Gate Array)에 HDL(Hardware Description Language)이라는 디지털 하드웨어의 논리 로직을 기술하는 언어를 이용하여 고속의 패킷 처리 로직을 구현하고, C, C++, Java, Perl 또는 Python 과 같은 프로그래밍 언어를 이용하여 필요한 어플리케이션을 구현하여 필요한 시스템을 개발하여 이용할 수 있다. 또한, NetFPGA 플랫폼을 이용하여 구현된 오픈 코드들이 점점 증가 하고 있기 때문에 향후 이를 이용한 교육 및 연구개발에 NetFPGA 플랫폼의 활용도가 높아질 것으로 기대된다[8].

2. NetFPGA 플랫폼 기반 RED 스케쥴러 구현

RED 스케줄러는 라우터 출력 포트의 큐의 상태를 실시간으로 모니터링 하면서 큐에 버퍼링 되는 패킷의 수가 특정 임계치를 초과할 경우 특정 폐기 확률에 따라 임의적으로 패킷을 폐기하여 동적으로 라우터 큐의 오버플로우를 방지함으로써 전역동기화 현상으로 인한 end-to-end 데이터 전송에 대한 성능 저하 방지를 가장 큰 목적으로 두고 있으며 그림 6에 RED 스케쥴러가 적용되어 NetFPGA 플랫폼 기반 라우터의 패킷 프로세싱을 담당하는 FPGA 내부 파이프라인 구조를 나타내었다.

그림 6의 RED 모듈은 라우터 출력포트 큐 관리를 위하여 크게 프리프로세서 모듈과 프로세서 모듈로 구성된다. 프리프로세서 모듈의 경우 라우터 출력 포트의 큐 상태를 모니터링하는 것과 동시에 패킷의 폐기확률 계산을 위한 일련의 전처리 기능을 담당하고 프로세서 모듈의 경우에는 프리프르세서 모듈에서 계산된 확률에 따라 라우터 출력포트의 큐에 패킷의 큐잉 여부 결정을

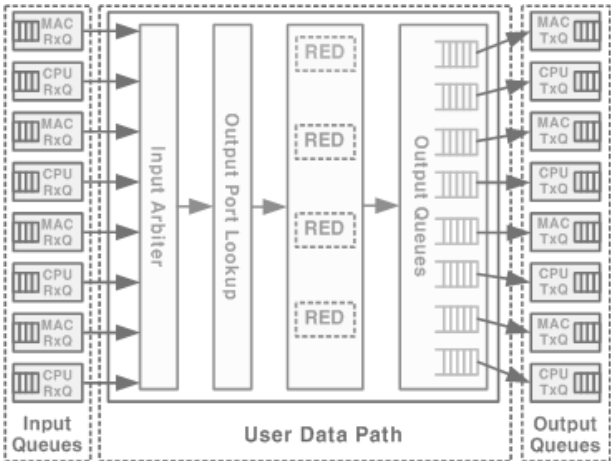

그림 6. RED Router의 NetFPGA 파프라인 구조
Fig. 6. NetFPGA Pipeline Architecture of the RED Router.

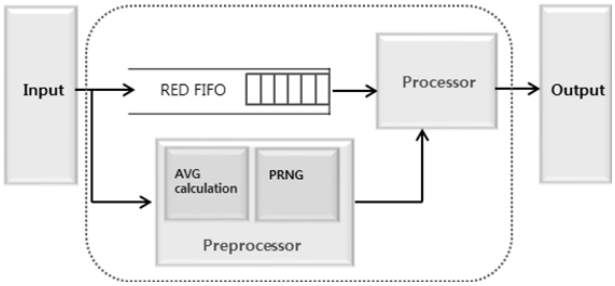

그림 7. RED 모듈의 구조
Fig. 7. Architecture of the RED Module.

담당하게 되는데 그림 7에 RED 스케쥴러 모듈의 구조를 나타내었다.

가. Preprocessor 모듈

프리프로세서 모듈은 매 패킷이 들어올 때 마다 EWMA(Exponent Weighted Moving Average) 방식으로 라우터 출력포트의 평균 큐 크기를 계산하는 Average Calculation 모듈과 21-bits LFSR(Linear Feedback Shift Register)을 사용하여 Uniform Random Variable의 특성을 갖고 난수를 발생시키는 PRNG (Pseudo Random Number Generation) 모듈로 구성되어있으며, 패킷이 들어올 때 마다 계산된 평균 큐의 크기가 그림 2에서와 같이 사전에 미리 결정되어지는 세 가지 구간에 어디에 위치하는지에 따라 패킷의 폐기 여부 결정을 달리하여 프로세서에 전달하는 기능을 갖고 있다. 계산된 평균 큐의 크기가 Queueing 구간에 위치할 때에는 패킷의 폐기가 없으며 Force Drop 구간에 위치할 경우에는 패킷을 무조건 폐기한다. 그러나, Random Drop 구간에 위치할 경우에는 발

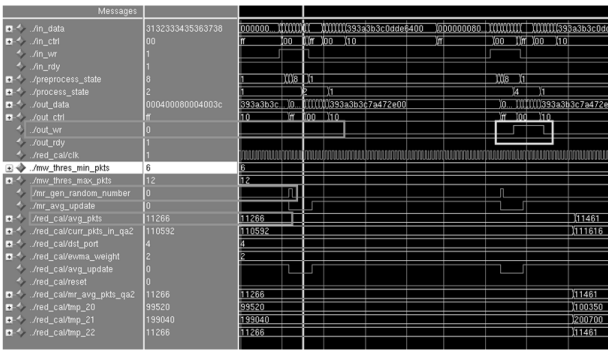

그림 8. RED 스케줄러의 시뮬레이션 결과
Fig. 8. The result of the RED scheduler simulation

생 된 난수와 패킷의 폐기 여부를 결정하기 위하여 미리 정해지는 폐기 확률에 따라 설정되는 임계치와의 비교를 통하여 패킷의 폐기 여부를 결정하게 되고 결정 된 패킷의 폐기 여부를 프로세서에 알리게 된다. 이때 모든 패킷은 일단 RED 모듈의 큐에 저장된다. 그림 8은 출력 포트의 큐가 36개의 패킷을 저장할 수 있는 경우에 대한 시뮬레이션 결과로써 첫 번째 패킷이 RED 모듈에 도착했을 때의 'avg_pkts' 의 값은 11266을 나타내는데 이는 연산을 위하여 2^{10}으로 스케일링 된 값으로 11266/1024, 즉 해당 패킷의 출력 포트에 11개의 패킷이 출력 대기중인 상태를 의미하게 되고 이는 표 2의 임계값을 참고하여 'Random Drop' 구간에 위치하고 있음을 알 수 있다. 이 경우 'avg_pkts' 의 값이 'Random Drop' 구간에 위치하므로 PRNG가 동작하여 난수를 발생시키고 발생된 난수와 패킷의 폐기 여부를 결정하기 위하여 미리 정해지는 폐기 확률에 따라 설정되는 임계치와의 비교를 통하여 패킷의 폐기 여부가 결정되어 프로세서 모듈로 전달이 된다. 시뮬레이션 결과에서 첫 번째 패킷은 폐기가 결정되고 두 번째 패킷은 출력이 결정되는 상태이다.

나. Processor 모듈

프로세서 모듈은 프리프로세서 모듈에서 결정된 패킷의 폐기 여부에 따라 동작하게 되는데 프리프로세서 모듈에서 패킷 폐기 결정이 내려지면 해당 패킷을 즉시 폐기하고 반대의 경우에는 해당 출펵 포트의 큐에 저장하게 된다. 이때, 패킷의 폐기는 RED 모듈의 큐에 임시적으로 저장되어 있는 패킷을 출력시키지만 출력 포트의 큐에 저장되지 않도록 NetFPGA 내부 모듈 간의 데이터 전송에 대한 컨트롤을 담당하는 컨트롤 버스의 신호 중 다음 모듈에게 데이터의 유효성을 알리는 신호인 'out_wr' 신호를 동작시키지 않아 다음 모듈에서 입력으로 받아드리지 않게 하는 방법을 이용하게 된다. 그림 8의 'out_data' 신호는 데이터 버스를 의미하며 데이터가 버스에 존재할 때 'out_wr' 신호의 확인으로 가능하다. 시뮬레이션의 첫 번째 경우는, 그림 8의 노란색 세로줄 부분, 프로세서 모듈에서의 패킷 폐기 결정에 따라 'out_wr' 신호가 변하지 않게되고 이는 해당 패킷의 출력 포트의 큐에 패킷이 저장되지 않도록 유효하지 않은 패킷임을 알리게 된다. 그리고, 두 번째 패킷의 경우는, 그림 8의 노란색 상자 부분, 프로세서 모듈에서의 출력 결정에 따라 'out_wr' 신호가 변하게 되는데 이는 해당 패킷의 출력 포트의 큐에 패킷이 저장되도록 유효한 패킷임을 알리게 되어, 출력 포트의 큐에 저장되어 다음 목적지로 패킷이 전송될 수 있도록 한다.

Ⅳ. NetFPGA기반의 RED 스케줄러 TCP 성능평가

NetFPGA 플랫폼 기반의 라우터를 이용하여 구현된 RED 알고리즘의 성능 확인 실험을 위하여 NetFPGA 플랫폼기반의 Network Emulator[9]를 이용하여 그림 9와 같은 로컬 테스트베드를 구축하였고 라우터 간 링크는 병목현상(bottleneck)으로 인한 패킷 손실이 잘 발생하도록 하기위하여 100Mbps의 가용 대역폭을 갖는 환경이 되도록 하였다. 또한, 구축된 테스트베드에서 Drop Tail 방식의 라우터와 성능을 비교하기 위하여 전송계층 프로토콜인 TCP(Transport Control Protocol)를 이용하여 패킷 전송률, 큐 점유율 및 전송 대역폭에 대한 실험을 수행하였다.

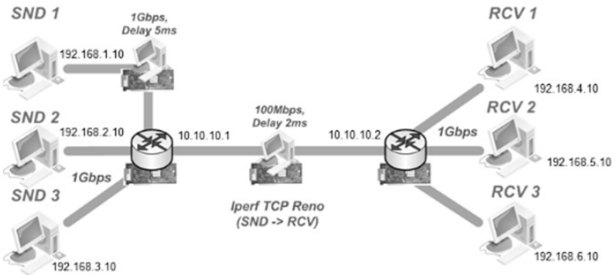

그림 9. 실험 네트워크 토폴로지
Fig. 9. Experiment network topology

1. 패킷 전송률

구축된 테스트베드를 이용하여 Drop Tail 방식의 라우터와 RED 스케줄러 방식 라우터에 대한 평균 패킷 전송률 성능 비교를 위하여 SND 1, 2 및 3에서 각각

표 2. Drop Tail(좌)와 RED(우)의 Sender1 Throughput
Table 2. Sender1's throughput for Drop Tail and RED.

Drop Tail			RED				
Buffer (pkts)	Throughput (Mbps)	Tx Ratio (%)	Min_{th}	Max_{th}	Buffer (pkts)	Throughput (Mbps)	Tx Ratio (%)
24	11	11.23596	4	8	24	17.2	17.30382
30	11.5	11.66329	5	10	30	18.3	18.37349
36	13.2	13.37386	6	12	36	19.2	19.95842
42	13.9	14.0546	7	14	42	20	19.92032
48	14.4	14.53078	8	16	48	18.9	19.05242
54	15.5	15.73604	9	18	54	17.9	17.88212
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

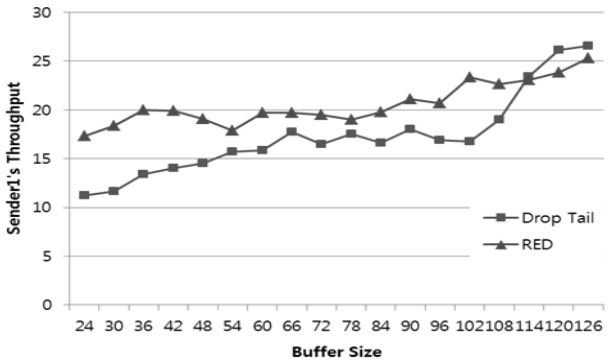

그림 10. Drop Tail과 RED Sender 1의 Throughput 비교
Fig. 10. Comparison of Sender 1's throughput for Drop Tail and RED.

RCV 1, 2 및 3를 향하여 iperf[10]를 이용한 TCP 트래픽을 발생시켜 라우터 간 병목현상 발생을 유발하였다. 표 2와 그림 10은 출력 포트의 버퍼 크기 변화에 따른 SND 1의 평균 패킷 전송률의 결과를 비교하여 나타낸다. 그림 10을 통해 RED 스케쥴러 방식의 라우터가 Drop Tail 방식의 라우터에 비하여 버퍼의 크기가 어느 정도 작을 때 보다 우수한 성능을 보임을 확인 할 수 있다.

2. 큐 점유율

동적 버퍼 관리에 따른 효율성을 확인하기 위하여 Drop Tail 방식과 RED 스케쥴러 방식의 라우터 평균 큐 점유율 비교를 하였다. 표 2의 실험 중 버퍼의 크기가 36인 경우에 대하여 비교하였으며, 표 3은 이때 사용된 RED 스케쥴러의 파라메터를 나타낸다.

그림 11은 Drop Tail 방식과 RED 스케쥴러 방식의 라우터에서 출력 포트의 평균 큐 점유율을 매초 간격으로 나타내는데 그림의 왼쪽 편이 큐의 점유율이 낮은 상태를 의미하며 RED 스케쥴러 방식의 라우터 실험의 결과를 나타내고 오른쪽 편은 Drop Tail 방식 라우터의

표 3. RED 스케줄러의 파라미터
Table 3. The parameters of the RED on NetFPGA

파라미터	설명	값
Min^{th}	평균 큐의 최소임계치	6
Max^{th}	평균 큐의 최대임계치	12
W_q	버퍼의 평균 큐 가중치	0.002
Max_p	평균 큐의 최대 드롭확률	0.02
Buffer(pkts)	6 x Minth	36

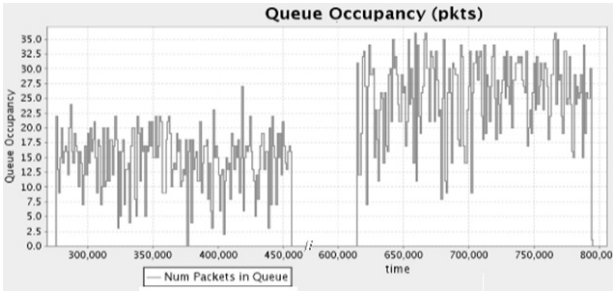

그림 11. RED 게이트웨이(좌)와 Drop Tail 게이트웨이(우)의 큐점유율 비교
Fig. 11. Queue occupancy of the RED(L) and DropTail(R).

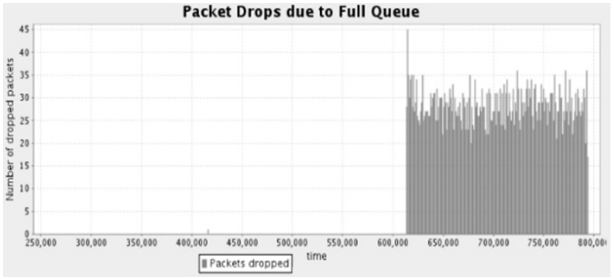

그림 12. RED와 Drop Tail패킷의 tail drop측정
Fig. 12. Measurement of Tail drop on RED and Drop Tail gateway.

실험 결과로써 큐의 점유율이 높은 상태를 의미하며 이로인하여 버퍼 오버플로우 현상이 빈번하게 발생함을 짐작할 수 있다. RED 스케쥴러의 실험 결과를 통하여 라우터 출력 포트 버퍼의 최대 수용 능력을 초과하는 버퍼 오버플로우로 인하여 발생하는 패킷 유실 현상에 따른 전역 동기화 문제를 해결하고, 또한 버퍼의 크기를 항상 낮게 유지하여 버스트 트래픽을 수용할 수 있는 특성을 확인 할 수 있다. 반면 Drop Tail 방식의 라우터는 버퍼의 오버플로우가 자주 발생되므로 전역 동기화 문제가 발생 되어 전송률 저하를 일으키고 있음을 있음을 확인 할 수 있으며, 그림 12에 발생된 Tail drop을 측정하여 나타내었다.

3. 전송대역폭(throughput)

Drop Tail 방식과 RED 스케줄러 방식 라우터의 전송 대역폭 비교 실험을 통하여 RED 스케줄러의 능동적인 버퍼관리를 통하여 TCP를 이용한 데이터 전송 시 전역동기화 현상으로 인한 순간적인 링크 사용률의 저하를 방지하여 전송률이 개선됨을 확인하였다. 이는 TCP 송신측의 혼잡윈도우(Congestion Window)의 크기를 비교함으로써 가능하다. 그림 13은 Drop Tail 방식의 라우터를 이용한 실험 결과로써 모든 송신측의 혼잡윈도우의 변화를 나타낸다. 순간적으로 모든 TCP 송신측의 혼잡윈도우의 크기가 동시에 줄어드는 전역동기화 현상이 자주 나타나는 것을 확인할 수 있으며, 이는 링크사용률을 감소시켜 전체적으로 전송률을 감소시키게 된다. 또한, 라우터의 버퍼 크기가 증가하게 되더라도 전역동기화 이후 회복률에 영향을 주기 때문에 전역동기화 현상으로 인한 전송률 감소에 대한 문제를 해소하는데 도움이 되지 않는다. 이에 반하여 그림 14는 RED 스케줄러 방식의 라우터를 이용한 실험 결과로써

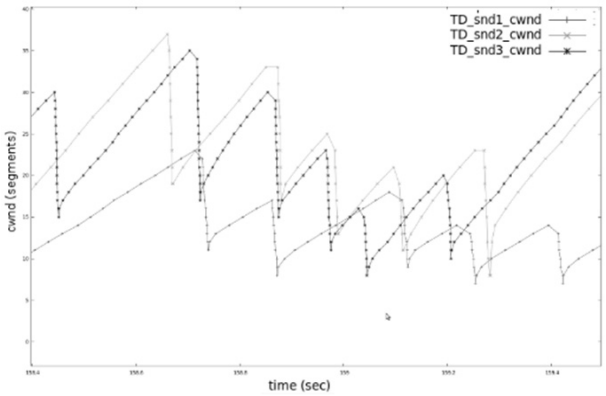

그림 13. DropTail Gateway의 각 노드의 혼잡윈도우
Fig. 13. Each node's cwnd on the Drop tail gateway.

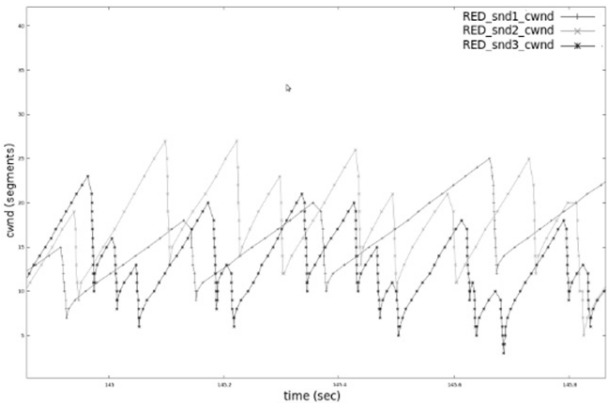

그림 14. RED Gateway의 각 노드의 혼잡윈도우
Fig. 14. Each node's cwnd on the RED gateway.

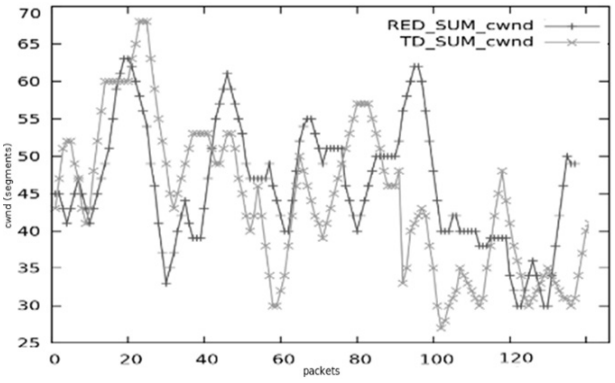

그림 15 Drop Tail과 RED의각 노드들의 합의 비교
Fig. 15. Comparison of sum of the each node on Drop Tail and RED gateways.

혼잡윈도우의 변화를 나타내는데 그림 13에서 확인되는 것과 같은 전역동기화현상이 나타나지 않음을 확인할 수 있다.

Drop Tail 방식과 RED 스케줄러 방식의 라우터를 이용한 실험에 대하여 각 전송측의 모든 혼잡윈도우의 합을 비교하여 보면 RED 스케줄러 방식의 라우터가 이용된 실험에서의 값이 Drop Tail 방식의 라우터가 이용된 실험의 값보다 대체적으로 크게 나타남을 확인할 수 있으며 이를 통하여 RED 스케줄러 방식의 라우터가 보다 혼잡 제어에 효과적이라는 것을 알 수 있다. 그림 15는 두 방식에 대한 혼잡윈도우 합의 비교를 나타낸다.

V. 결 론

본 논문에서는 동시에 여러 개의 TCP Flow들이 경쟁적으로 유입되어 라우터에서 발생되는 병목현상으로 인한 라우터 버퍼 오버플로우로 인하여 발생되는 패킷 손실문제를 해결하기 위하여 능동적으로 혼잡제어와 혼잡회피를 수행하는 알고리즘의 대표적인 RED 스케줄러를 NetFPGA 플랫폼 기반의 라우터에 구현하였고 테스트베드를 구축하여 Drop Tail 방식의 라우터와 TCP를 이용한 성능 비교 실험을 수행하였다.

RED 스케줄러 방식의 라우터는 능동적으로 버퍼의 평균 큐 길이를 낮게 유지함으로써 네트워크에서 비효율적인 링크 사용률 원인의 하나인 전역 동기화 현상을 방지할 수 있음을 확인하였고, 이로 인하여 TCP 송신측 혼잡위도우의 크기가 적절하게 유지되면서 전반적으로 전송률이 향상됨을 확인하였고, 또한 다른 플로우에

비하여 전송 지연이 큰 노드의 TCP 전송률이 향상됨을 확인함으로써 Drop Tail 방식의 라우터에 비하여 망 혼잡제어에 효과적임을 확인하였다.

RED 스케줄러는 동적으로 버퍼를 관리하여 전송률을 향상시키는 알고리즘의 하나로써 네트워크 환경의 영향을 받아 파라미터 설정이 어려운 단점이 존재하나 파라메터를 변경하며 실험한 결과를 토대로 최적의 파라메터 설정이 가능하다. 그리고 NetFPGA 플랫폼을 이용하여 구축된 테스트베드를 통하여 시뮬레이션을 통한 방법에 비하여 보다 실제 환경과 유사한 환경의 실험 결과를 얻을 수 있었으며, 이러한 점은 많은 비용이 필요한 상용 장비 제조사의 라우터로 실험하기 어려운 부분을 저렴한 비용으로 PC 기반 프로그래머블 하드웨어 플랫폼을 활용하여 저렴한 비용으로 높은 효율의 테스트베드 구축이 가능함을 의미한다.

향후, 하드웨어 기반의 고속 패킷처리가 지원되는 NetFPGA 플랫폼을 이용하여 RED의 단점을 보완한 Adaptive RED, Weighted RED 및 REM(Random Exponential Marking)등을 구현할 것이며, 새로운 AQM 기법을 설계하고 구현하여 성능평가를 수행하는 연구를 진행할 것이다.

참 고 문 헌

[1] Soon Jong Gwen, “미래인터넷을 위한 네트워크 가상화 기술”, 정보과학회지, Oct. 2008.
[2] Man Kyu Park, Whoi Jin Jung, Jae Yong Lee, Byung Chul Kim, “A Study of Future Internet Testbed Construction using NetFPGA/OpenFlow Switch on KOREN/KREONET”, Journal of the IEEK, Vol. 47-TC, No. 7, July 2010.
[3] 김대영, 문수복, 박성용, “네트워크 가상화에 대한 고찰”, Oct. 2008.
[4] NetFPGA forum, http://www.netfpga.org
[5] S. Floyd, “Random Early Detection Gateways for Congestion Avoidance”, IEEE/1993.
[6] B. Braden, “Recommendations on queue management and congestion avoidance in the Internet”, IETF RFC2309, April, 1998.
[7] Y. Gao, “On Exploiting Traffic Predictability in Active Queue Management”, IEEE INFOCOM, pp. 1630-1639, June 2002.
[8] Seokhong Min, Whoijin Jung, Byungchul Kim and Jaeyong Lee, “NetFPGA-based Scheduler Implementation and its Performance Evaluation for QoS of Virtualized Network Resources on the Future Internet Testbed”, Journal of the IEEK, Vol. 48-TC, No. 8, Aug. 2011.
[9] Seok Hong Min, Jae Yong Lee, Byung Chul Kim, “A Network Emulator on the NetFPGA Platform”, 1stAsia NetFPGA Developer’s Workshop ,Korea, June2010
[10] Iperf site, http://iperf.sourceforge.net/

저 자 소 개

오 민 경(학생회원)
2003년 동국대학교 정보통신 공학과 학사
2011년 충남대학교 전자전파 정보통신공학과 석사
2003년～한국과학기술원 정보통신팀 근무
<주관심 분야 : 데이터 통신, 미래인터넷, 네트워크 성능분석>

민 석 홍(학생회원)
2005년 공주대학교 전기전자정보 공학과 석사
2010년～현재 충남대학교 전자전파정보통신공학과 박사과정
2004년 한국전자통신연구원 BcN 시험기술팀 위촉연구원
2005년 디지피아㈜ 방송장비팀 연구원
2006년～2009년 ㈜엠티아이 연구2실 전임연구원
<주관심분야 : 무선 메쉬 네트워크, 데이터 통신, 이동통신 네트워크>

김 병 철(평생회원)-교신저자
1988년 서울대학교 전자공학과 학사
1990년 한국과학기술원 전기 및 전자공학과 석사
1996년 한국과학기술원 전기 및 전자공학과 박사
1993년～1999년 삼성전자 CDMA 개발팀
1999년～현재 충남대학교 정보통신공학부 교수
<주관심 분야 : 이동인터넷, 이동통신 네트워크, 데이터 통신>

이 재 용(평생회원)
1988년 서울대학교 전자공학과 학사
1990년 한국과학기술원 전기 및 전자공학과 석사
1995년 한국과학기술원 전기 및 전자공학과 박사
1990년～1995년 디지콤 정보통신연구소 선임연구원
1999년～현재 충남대학교 정보통신공학부 교수
<주관심분야 : 초고속통신, 인터넷, 네트워크 성능분석>