엔터프라이즈 네트워크에서 패킷 지연시간을 최소화하는 공정 큐잉 알고리즘

윤 여 훈[†] · 김 태 윤[††]

요 약

오늘날 네트워크에서 다양한 애플리케이션의 서비스 성능을 저하시키는 불공정 큐잉 문제를 해결하기 위해 공정 큐잉 분야가 활발히 연구중이다. 그 중에서 DRR(Deficit Round Robin)은 작업 복잡도가 낮고 구현이 간단한 기법으로 매 라운드마다 각 큐에 대해 이전 라운드에서의 서비스 결손량을 포함한 서비스 할당량 SQ(Service Quantum)만큼을 서비스하도록 하여 다양한 트래픽들에 대해 정확한 공정성을 보장하는 기법이다. 그러나 엔터프라이즈 네트워크 환경에서 최대 수 kbyte 이상의 패킷 사이즈를 가지는 서비스에 대해 불필요한 SQ 재설정 횟수 및 라운드 순회 횟수로 인한 지연시간 증가를 일으킨다. 본 논문에서는 이러한 DRR의 문제를 최소화하기 위해 SQ를 동적으로 설정하는 기법 및 패킷을 처리하는데 있어서의 작업 복잡도를 최소화하는 기법을 제안한다. 제안한 기법은 SQ의 동적 설정과 보다 단순한 패킷 처리로 인해 엔터프라이즈 네트워크 환경에서 다양한 애플리케이션들에 대한 지연시간을 최소화한다.

Fair Queuing Algorithm Minimizing Packet Delay in Enterprise Network

Yeo Hoon Youn[†] · Tai Yun Kim[††]

ABSTRACT

Nowadays the fair queuing field is studied actively for solving the unfair queuing problem which degrades the service performance of various applications on network. Above all DRR is a scheme that has lower work complexity and can be implemented easily. It guarantees the fair service by serving each queue every round Service Quantum(SQ) that includes the service deficit of the previous round. But it increases the delay by the numbers of unnecessary resetting of SQ or round circulation for the service that have the packet size over maximum several kbyte. In this paper, We propose the method that sets SQ dynamically to minimize this problem and the method that minimizes the work complexity on processing of packet. The proposed scheme minimizes the delay on various applications in enterprise environment by setting SQ dynamically and processing the packet simply.

키워드 : Fair Queuing, DRR, Packet Scheduling, Enterprise Environment

1. 서 론

엔터프라이즈 네트워크 환경이란 수십, 수백 대의 서버와 수천, 수만 대의 퍼스널 컴퓨터가 전국 또는 전세계의 여러 사업장에 산재되어 있는 기업 정보처리 네트워크이다. 이러한 환경에서의 서비스는 광대역 통신과 실시간 처리 및 실시간 전송 메커니즘이 필요하다. 특히 최대 수 기가비트(gigabit)의 대역폭을 갖는 고속 전용선을 사용하는 엔터프라이즈 네트워크 환경에서 패킷 스케줄링 알고리즘은 특정 애플리케이션 트래픽의 대역폭 점령과 그 외의 트래픽의 병목 및 기아(starvation)로 인한 데이터 상실을 막기 위해 중요한 메커니즘이다. 대표적인 예로, 대부분의 네트워크에서 사용되는 최선 노력(best-effort) 기반의 FCFS기법은 스케줄러로 입력되는 패킷의 사이즈와 입력되는 속도가 상대적으로 큰 트래픽이 큐를 점령하는 경우가 발생하여 다른 트래픽들에 대한 병목 및 기아를 일으킨다[2, 7]. 따라서 엔터프라이즈 네트워크 환경에서 최소 수십 bytes에서 최대 수 kbyte 이상의 패킷 사이즈를 갖는 애플리케이션들을 각각 독립적으로 관리하면서 공정한 서비스를 보장하는 패킷 스케줄링 기법이 필요하다.

최근에 공정한 서비스 제공을 위한 많은 패킷 스케줄링 기

† 준 회 원 : 고려대학교 대학원 컴퓨터학과
†† 종신회원 : 고려대학교 컴퓨터학과 교수
논문접수 : 2001년 8월 8일, 심사완료 : 2001년 10월 22일

법들이 개발되고 있고, 그 중에서 DRR(Deficit Round Robin)은 다른 것들에 비해 구현이 쉽고 낮은 작업 복잡도를 가지는 기법으로 정확한 공정성을 보이는 패킷 스케줄링 알고리즘이다[2, 7, 9]. 스케줄러가 첫 번째 큐부터 마지막 큐까지 한번씩 방문하는 것을 라운드라고 정의할 때, DRR은 현재 라운드에서 각 큐에 대한 상대적인 서비스 결손량과 모든 큐에 고정적으로 설정된 기본적인 할당량의 합을 나타내는 SQ(Service Quantum)만큼을 다음 라운드에서 서비스하여 정확한 공정성을 보장하는 기법이다. 이것은 일반적인 패킷 스위칭 네트워크에서는 애플리케이션의 종류에 따라 패킷 사이즈가 대략 50byte에서 1,500byte 정도에서 최적의 성능을 보인다. 그러나 전용 고속망을 이용하는 엔터프라이즈 네트워크 환경에서 애플리케이션간의 패킷 사이즈 차가 최대 수 kbyte 이상인 상황에서는 적합하지 않다. 만약 패킷 사이즈가 수 kbyte 이상인 멀티미디어 애플리케이션의 패킷들이 패킷 스케줄러로 입력될 경우 SQ가 패킷 사이즈 이상의 값을 가질 때까지 SQ를 재설정하면서 패킷을 큐에 담아 둔다. 불필요한 SQ 재설정 횟수 및 라운드 횟수로 인해 시간이 지연되고, 이러한 지연 시간 증가는 경성 실시간(hard real time) 처리를 요하는 멀티미디어 애플리케이션 서비스의 품질을 저하시키는 원인이 되기 때문에 기존의 DRR과 같은 알고리즘으로 최적의 성능을 보장할 수 없다. 따라서 패킷 사이즈에 따라 SQ를 동적으로 재설정하는 기법이 요구된다.

본 논문에서는 DRR의 문제를 해결하기 위해, 매 라운드마다 전송을 앞둔 패킷의 사이즈를 고려하여 SQ를 동적으로 설정하는 기법 및 패킷을 처리하는데 있어서의 작업 복잡도를 줄이는 기법을 제안하여 서비스 지연시간을 최소화하는 패킷 스케줄링 기법을 제안한다.

본 논문의 구성은 다음과 같다. 2장에서는 엔터프라이즈 시스템구성과 엔터프라이즈 환경에서의 다양한 애플리케이션의 종류를 알아본 다음 기존의 다양한 알고리즘들을 분석한다. 3장에서는 엔터프라이즈 환경의 다양한 애플리케이션의 특성을 고려한 공정한 서비스를 제공하는 패킷 스케줄링 알고리즘을 제안한다. 4장에서는 제안한 알고리즘에 대한 이론적인 분석을 하고, 5장에서는 스케줄링 적용 환경 및 기존의 DRR 알고리즘과 제안된 알고리즘의 성능을 분석하고, 6장에서는 결론 및 향후 과제를 제시한다.

2. 관련 연구

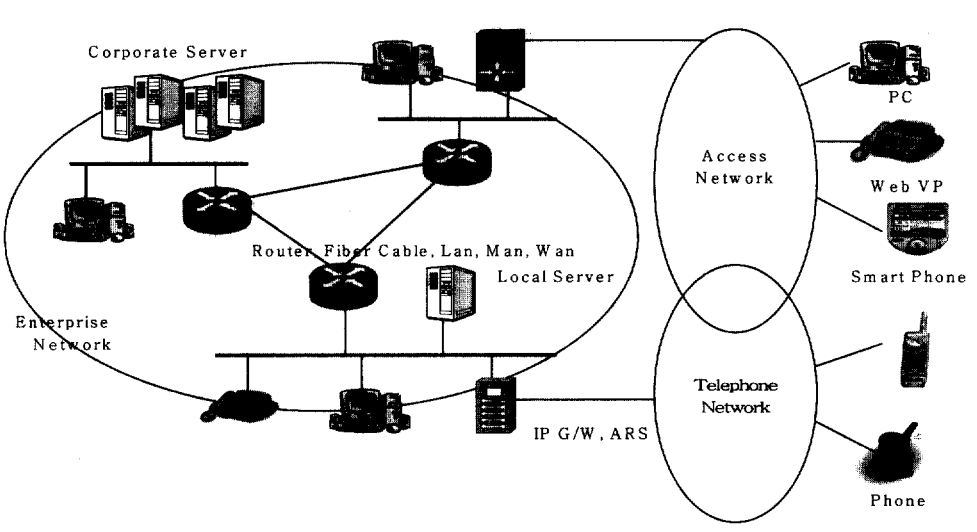
2.1 엔터프라이즈 시스템 구성

엔터프라이즈 시스템 구성 개념도는 (그림 1)과 같다[11].

본 논문에서 제안한 패킷 스케줄링 알고리즘이 적용되는 환경은 (그림 1)에서 엔터프라이즈 네트워크이다. 오늘날 기가비트 이상의 대역폭으로 지역과 지역을 잇는 광케이블 등의 도입으로 Wan, Man의 개념이 Lan의 개념으로 흡수되고 있다. 예를 들어, 수도권의 본사와 지방의 지사까지 최대 수 기가비트 이상의 전용선으로 연결되어 있어 마치 같은 건물 내에서의 쌍방간 통신 효과를 준다. 또한 업무의 효율성뿐만 아니라 기업의 중요한 정보 유출을 방지하기 위해 각 사업장에서 또는 사업장과 사업장 사이에 최소 100M에서 최대 수 기가비트 이더넷의 기업망을 고속 전용선으로 구축하거나 상호 연결하는 것은 필수적이며 이에 대한 비용 지불은 불가피하다. 또한 해외의 지사까지도 전용선으로 연결하기도 한다.

대역폭의 증가는 패킷의 최대 전송 단위인 MTU(Maximum Transfer Unit)의 변화에도 많은 영향을 미친다. 즉, 대역폭이 커질수록 MTU가 증가한다(10Mbps 이더넷에서의 MTU는 1536bytes, 100Mbps 이더넷에서의 MTU는 4096bytes, 기가비트 이더넷에서는 더욱 큰 사이즈의 MTU를 갖는다)[14].

고속 네트워크에서 MTU가 클수록 전송 지연 면에서 효과적이다. MTU가 작다면 상대적으로 사이즈가 큰 패킷들에 대해서 여러 개의 패킷으로 단편화(fragmentaion)를 해야 하



(그림 1) 엔터프라이즈 시스템 구성

는데, 이것은 각각의 패킷들에 대하여 헤더를 붙이는 작업과 패킷이 스케줄러로 삽입되었을 때 늘어난 패킷을 추가로 구분하여 각각의 큐로 삽입하는데 있어서의 오버헤드가 발생하기 때문에 지연시간을 증가시키는 결과를 가져온다[14, 15]. 따라서 고속 네트워크에서는 전송되는 패킷이 MTU 가 제한하는 범위 내에서 단편화 없이 전송되는 것도 패킷을 전송시 고려해야할 사항이다. 본 논문에서 제안한 패킷 스케줄링 기법은 SQ의 재설정 횟수를 단 한번으로 줄일 뿐만 아니라 MTU의 범위 내에서 최초 전송되는 패킷 사이즈를 보존하는 기법으로 전송 지연을 최소화하는 기법이다.

2.2 엔터프라이즈 환경에서의 애플리케이션 서비스

엔터프라이즈 네트워크 환경에서 컴퓨터 및 통신 네트워크를 활용하는 애플리케이션을 속성에 따라 네 가지 클래스로 분류한다[11, 12].

클래스 1 : 임계 작업
- 금융 시스템, 항공 예약 시스템, 전자 상거래의 전자 결제 시스템 등.
- 연성 실시간(soft real time) 또는 경성 실시간(hard real time)처리를 요구한다.
- 패킷 사이즈는 100byte 정도로 크지 않다.

클래스 2 : 대화형 작업
- 인터넷 전화, 웹 비디오 전화, 주문형 뉴스, 인터넷 비디오 광고 등.
- 주기적으로 데이터를 보내는 경성 실시간(hard real time) 처리를 요구하며, 재전송 메 커니즘이 의미가 없다.
- 음성의 경우 패킷 사이즈가 100byte 정도이며, 비디오의 경우는 수 kbyte이다.

클래스 3 : 고용량 고품질 작업
- 높은 정밀도의 오디오/비디오 스트림 처리를 요구하는 것으로 디지털 방송, 주문 형 비디오 등.
- 경성 실시간 처리를 요구하며, 재전송 메커니즘이 의미가 없다.
- 패킷 사이즈는 매우 크다.

클래스 4 : 일반 작업
- 보통의 파일, 텍스트, 그래픽 이미지와 MP3 데이터 등.
- 실시간 처리를 요구하지 않으며, 재전송 메커니즘이 요구된다.
- 패킷 사이즈는 50~1,500bytes 정도이다.

클래스 1과 관련된 서비스는 데이터 량은 작지만 분실이나 지연이 발생해서는 시스템 전체의 신뢰성이 크게 손상되는 중요한 데이터이고, 클래스 2와 관련된 서비스는 최근 인터넷/인트라넷에서 강력히 요구되는 서비스에서 발생되는 데이터 유형으로 사람과 사람의 통신이나, 웹서버와 사람간의 짧은 AV 정보를 제공하는데 사용된다. 클래스 3과 관련된 서비스는 고품질의 MPEG2 수준의 데이터를 실시간으로 전송하는 경우에 발생되는 데이터 유형으로 패킷 사이즈가 수 kbyte 이상이다. 클래스 4와 관련된 서비스는 현재의 인터넷 홈페이지에서 서비스되는 HTML 문서와 같은 종류의 데이터와 인터넷 자료실에서 검색한 수 kbytes에서 수 Mbytes의 데이터를 사용자가 FTP 프로토콜을 이용하여 다운로드 받는 형태로 처리되는 데이터, 그리고 메일 애플리케이션의 데이터 등이다. 클래스 2, 3과 같은 멀티미디어 애플리케이션 서비스일수록 패킷 사이즈가 크고 경성 실시간 처리를 요구한다.

2.3 기존의 패킷 스케줄링 알고리즘 및 DRR 알고리즘 분석

기존의 공정한 서비스를 제공하는 방식에는 DRR(Deficit Round Robin), ORR(Ordinary Round Robin), WRR(Weighted Round Robin), WFQ(Weight Fair Queuing) 방식이 있다.

DRR은 매 라운드마다 각 큐의 결손량을 다음 라운드에서 보상하여 정확한 공정성을 보장하는 알고리즘으로, DRR에서 DC(Deficit Counter), Q(Quantum), SQ(Service Quantum)은 각 큐간의 공정한 서비스를 제공하기 위해 사용되는 것들이다. DC는 큐의 상대적인 결손량을 나타내고, Q는 모든 큐에 고정적으로 부여된 동일한 할당량을 나타내며, SQ는 이전 라운드에서의 DC에 Q를 더한 값이다.

DRR은 매 라운드마다 각 큐에 SQ만큼을 서비스하여 정확한 공정성을 보장한다. 또한 현재 패킷을 포함하는 큐들의 인덱스 리스트인 활성리스트를 운영하여 스케줄러가 큐들을 순회하는데 있어서 지연을 최소화한다[2, 7, 9].

그러나 DRR은 기본적인 서비스 할당량 Q를 고정적으로 설정해 놓는 기법이기 때문에 엔터프라이즈 환경에서 데이터 량이 많고 패킷 사이즈가 상당히 큰 클래스 2, 3에 해당하는 애플리케이션의 서비스를 효율적으로 지원하지 못한다. 패킷 사이즈가 매우 큰 인터넷 방송, 주문형 비디오 등 클래스 2, 3의 멀티미디어 서비스 트래픽들이 큐로 삽입될 경우 스케줄러는 SQ가 패킷 사이즈보다 클 때까지 SQ를 계속 재설정하면서 최대 수십 번의 라운드를 순회한다. 불필요한 SQ 재설정 횟수와 라운드 순회 횟수로 인한 시간이 지연되어 경성 실시간 처리를 요하는 클래스 2, 3에 해당하는 애플리케이션의 서비스 품질을 저하시킨다. 따라서 애플리케이션간의 패킷 사이즈의 차가 매우 큰 엔터프라이즈 환경에서 최적의 성능을 위해 패킷 사이즈에 따라 서비스 할당량 SQ를 동적으로 설정하는 기법이 요구된다.

ORR[7, 9]은 스케줄러가 각 큐를 방문할 때마다 패킷 하나씩을 전송하는 기법으로 패킷 사이즈가 동일한 네트워크에서는 아주 정확한 공정성을 보장하지만, 패킷 사이즈가 다른 상황에서는 패킷 하나를 전송할 때마다 패킷의 크기가 상대적으로 큰 것에 대한 작은 것의 손실은 계속 누적되는 불공정 큐잉을 일으켜 공정성 저하를 일으킨다. WRR[3]은 각

큐에 대하여 전송할 패킷의 수를 나타내는 가중치(weight)를 패킷의 평균 사이즈를 기반으로 설정하기 때문에 정확한 공정성을 보장하지 못할 뿐만 아니라 가중치를 예측하기가 매우 어렵다. 또한 예측된 가중치에 조금의 오차가 발생할 경우 공정성에 상당한 악영향을 미친다. WFQ[1, 5, 7] 계열의 알고리즘은 들어오는 패킷에 대해 마감시간을 나타내는 인덱스를 부여하여 패킷이 들어올 때마다 패킷들을 인덱스에 따라 큐 안에서 정렬시켜 마감시간이 가장 임박한 패킷부터 먼저 서비스하는 기법이다. 이것은 공정성에서 우수한 성능을 보이지만 패킷의 인덱스를 계산하고 그것에 따라 큐 안에서 정렬하는데 높은 작업 복잡도가 요구되어 근원지와 목적지 사이에 노드 수가 많을 때 패킷을 전송하는데 많은 시간이 지연된다.

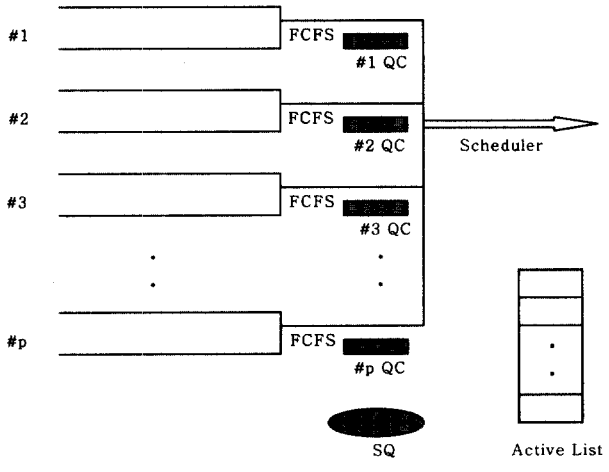
3. 다양한 애플리케이션을 지원하는 공정 큐잉 알고리즘

제안한 알고리즘은 현재 라운드를 마친 후 각 큐의 가장 앞쪽에 있는 패킷과 각 큐가 현재 라운드까지 전송한 총전송량을 더한 값을 다음 라운드의 SQ로 설정하는 기법이다. 이것은 각 큐의 가장 앞쪽에서 전송을 앞둔 패킷들 중 사이즈가 가장 큰 패킷도 현재 라운드에서 서비스될 수 있도록 하여 불필요한 SQ 재설정 횟수 및 라운드 순회 횟수를 줄여 지연시간을 최소화하는 기법이다.

3.1 제안된 스케줄러의 개요

제안한 알고리즘은 엔터프라이즈 환경의 다양한 애플리케이션의 패킷 사이즈와 그 특성을 고려하여 기존의 DRR을 수정한 것으로 매 라운드를 시작하기 전 전송을 앞둔 패킷들의 사이즈를 고려하여 SQ를 동적으로 설정하는 기법을 제안한다.

스케줄러의 전체적인 개요는 (그림 2)와 같다.



(그림 2) 패킷 스케줄러 구조

기존의 DRR에서의 DC 대신 지금까지 전송한 총데이터 량의 계측자인 QC(Quantum Counter)를 사용하고, 기존의 DRR과 같이 스케줄러는 현재 패킷을 하나 이상 포함하는 큐의 인덱스 노드들의 링크드 리스트인 활성리스트의 순서에 따라 큐들을 방문한다.

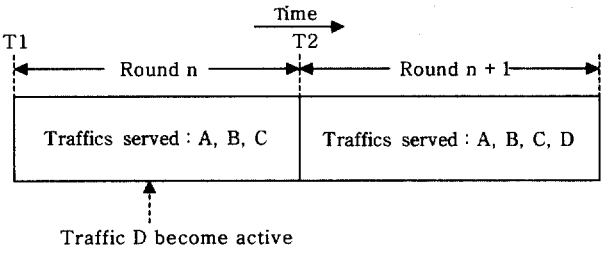
제안된 패킷 스케줄러는 enqueue와 dequeue 모듈로 구성되며, enqueue와 dequeue 모듈은 서로 비동기적으로 동작한다. (그림 3)은 enqueue 모듈의 의사 코드이다.

```
1.  Initialization
2.    for (i = 1 ; i < P ; I++)
3.      QC = 0 ;
4.  Enqueuing module : on arrival of packet p :
5.      while(Traffic(p))
6.        for all queue i(i = 1, 2, 3, ···, P)
7.        classify packets ;
8.        if(NonExistsActiveList(i)) then
9.            i = ExtractFlow(p) ;
10.           InsertActiveList(i) ;
11.       else
12.           i = ExtractFlow(p) ;
```

(그림 3) enqueue 모듈

최초에 모든 큐들의 QC를 0으로 초기화시킨다(라인 2). 패킷들이 입력됨에 따라 2.1절에서와 같이 엔터프라이즈 환경에서의 4가지 클래스로 패킷을 구분하여 클래스의 번호와 실시간 처리 요구도에 따라 해당 클래스 큐에 삽입시킨다(라인 6-7). 동일한 클래스의 다수개의 큐에 대해 실시간 요구도가 높은 애플리케이션 패킷을 우선 순위가 높은 큐에 입력한다.

패킷이 계속 입력됨에 따라 그것이 현재 활성 리스트에 속하는 큐로 분류되는지 아닌지를 판단한다. 활성리스트에 속하지 않는다면 큐에 삽입시킨 다음 활성리스트의 끝부분에 큐를 등록시키고, 활성리스트에 속해있다면 활성리스트에 등록할 필요 없이 큐에만 삽입시킨다(라인 8-12).



(그림 4) 패킷 처리 시점

(그림 4)와 같이 n번째 라운드에서 데이터 A, B, C가 서비스되고 있을 때, 활성 리스트에 등록되지 않은 데이터 D가 들어온다면 데이터 D는 n+1번째 라운드부터 서비스된다. 이러한 과정을 데이터가 들어오는 동안 계속된다. (그림 5)는 dequeue 모듈의 의사코드이다. dequeue 모듈은 각 큐에

삽입된 패킷들을 SQ에 따라 실제 출력 링크로 전송하는 부분이다.

```
1.  Dequeuing module :
2.    calculate SQ of first round ;
3.    while(ActiveList(i))
4.     if(!RQ(p))
5.       service according to ActiveList
6.        with calculation QCi
7.        if (Emptied(i))
8.          QCi = 0 ;
9.          DeleteActiveList(i) ;
10.    else
11.       service from RQ ;
12.       DeleteActiveList(RQ) ;
13.    calculate SQ
```

(그림 5) dequeue 모듈

(그림 5)와 같이 SQ(m)(m = 1, 2, 3, …)를 m번째 라운드에서의 서비스 할당량, $QC(m)_i$(i = 1, 2, 3, …, P)를 i번째 큐가 m번째 라운드를 끝낸 후 지금까지 전송한 총데이터 량, $A(0)_i$을 첫 라운드가 시작될 때 각 큐의 가장 앞쪽에 있는 패킷의 사이즈라고 할 때, 첫 라운드에서의 SQ(1)는 식 (1)과 같다(라인 1). 즉, 각 큐의 $A(0)_i$ 중에 가장 큰 값이 SQ (1)가 된다.

$$SQ(1) = \mathrm{Max}(A(0)_i) \tag{1}$$

각 큐마다 enqueue 모듈에서 0으로 설정된 $QC(1)_i$에서 시작하여 최대 SQ(1)만큼 패킷을 서비스한 후 $QC(1)_i$에 전송한 데이터 량을 더한다(라인 4-6). 활성리스트의 순서에 따라 큐들의 서비스가 끝나고 두 번째 라운드가 시작되기 전에 SQ(2)값이 다시 설정된다. $A(1)_i$를 첫 번째 라운드에서 i번째 큐를 서비스한 다음 i번째 큐의 가장 앞쪽에 위치한 패킷의 사이즈라고 할 때, 두 번째 라운드에서의 SQ(2)는 식 (2)와 같다. 즉, 모든 큐에 대하여 $QC(1)_i$와 $A(1)_i$를 합한 값들 중 가장 큰 값으로 두 번째 라운드의 SQ(2)를 재설정한다(라인 13).

$$SQ(2) = \mathrm{Max}(QC(1)_i + A(1)_i) \tag{2}$$

각 큐마다 이전 라운드의 $QC(1)_i$에서 시작하여 최대 SQ(2)만큼 패킷을 서비스한 후 $QC(2)_i$에는 이전 라운드의 $QC(1)_i$ 값과 현 라운드에서 전송한 데이터 량의 더한 값을 설정한다(라인 4-6). 활성리스트의 순서에 따라 큐들의 서비스가 끝나고 세 번째 라운드부터 두 번째 라운드에서와 동일한 방법으로 SQ(m)와 $QC(m)_i$가 재설정되어 서비스된다.

$bytes_k(i)$(k = 1, 2, 3, …)를 스케줄러가 k번째 라운드에서 i번째 큐에 대하여 실제 전송한 데이터 값이라고 할 때, $bytes_k(i)$에 대해서 식 (3)의 관계가 성립하고, m번째 라운드에서 전송한 실제 데이터 값 $bytes_m(i)$에 대하여 스케줄링이 가능하기 위해서 식 (4)를 만족해야 한다.

$$QC(m)_i = \sum_{k=1}^{m} bytes(k)_i \tag{3}$$

$$\mathrm{SQ(m)} - QC(m-1)_i \geq bytes(m)_i \tag{4}$$

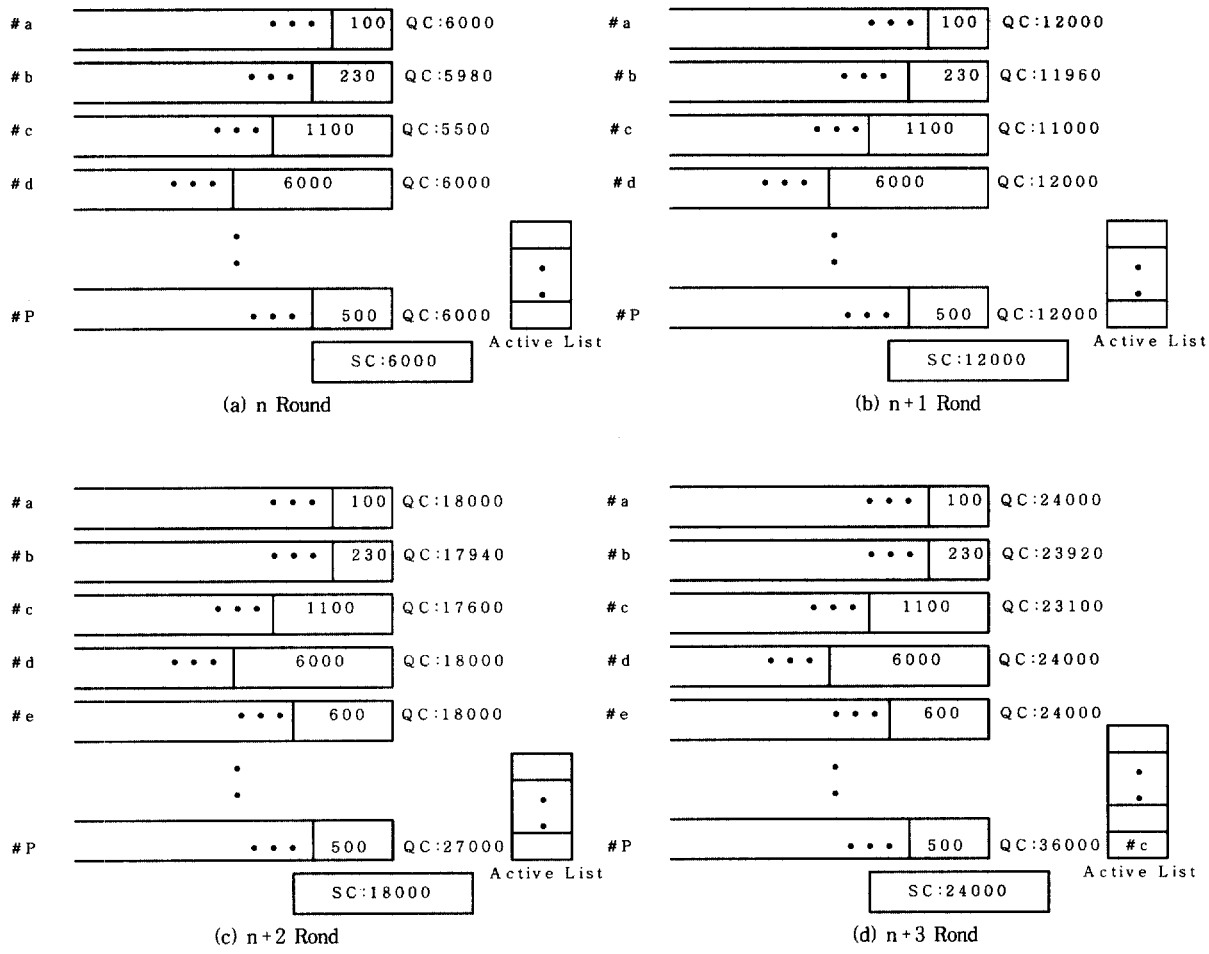
스케줄러가 $bytes(m)_i$에 대해 식 (4)을 만족하면서 큐들을 순회하기 때문에 큐를 방문할 때마다 최소한 하나 이상의 패킷 전송을 보장한다. 그러므로 큐들의 가장 앞쪽에 위치한 패킷들 중 패킷 사이즈가 아무리 큰 것이라도 현재 라운드에서 전송되어 불필요한 SQ 재설정 횟수 및 라운드 순회 횟수를 줄인다.

매 라운드마다 스케줄러가 각 큐를 방문하는 동안 또는 방문한 후 큐에 패킷이 더 이상 존재하지 않는다면 해당 큐의 인덱스 노드는 활성 리스트에서 삭제되고 $QC(m)_i$는 0으로 설정된다(라인 7-9). 활성 리스트에서 삭제되었던 큐에 패킷이 다시 들어왔을 때 또는 새로운 트래픽의 패킷이 들어왔을 때 SQ(m)는 앞에서 설명한 방식으로 설정되고 $QC(m)_i$은 SQ(m)와 현재 0으로 설정되어 있는 $QC(m)_i$의 합한 값으로 설정한다.

3.2 제안된 스케줄러의 수행 과정

(그림 6)은 엔터프라이즈 환경에서 클래스 1에서부터 클래스 4까지의 다양한 애플리케이션 서비스 패킷들이 해당 큐에 삽입되고 있는 상황에서 매 라운드마다 SQ(n) 및 $QC(n)_i$이 재설정 되어 패킷이 서비스되는 과정을 나타내고 있다. 현재 #a, #b, #c, #d, …, #P 큐에는 평균 패킷 사이즈가 각각 100byte, 230byte, 1100byte, 6000byte, …, 500byte인 패킷들이 들어오고 있고 n−1번째 라운드까지 각 큐의 $QC(n-1)_i$에 측정된 값이 0이라고 가정한다.

(그림 6 (a))는 n번째 라운드에서 서비스 직전의 큐와 SQ(n) 및 라운드를 마친 후의 $QC(n)_i$ 상태를 보여주고 있다. n번째 라운드를 시작하기 전, 이전 라운드까지의 각 큐의 $QC(n-1)_i$와 현재 각 큐의 가장 앞쪽에 위치한 패킷의 사이즈를 합한 값들 중 가장 큰 값으로 SQ(n)를 설정한다. 스케줄러가 n번째 라운드를 순회하면서 각 큐에 대하여 $QC(n-1)_i$에서 시작하여 최대 SQ(n)만큼 패킷들을 서비스할 수 있는데, 각 큐에 대하여 실제 전송한 데이터 량을 $QC(n)_i$에 더한다. (그림 6 (b))에서도 (그림 6 (a))의 n번째 라운드와 같은 방법으로 SQ(n+1)를 설정하여 서비스한 다음 같은 방법으로 $QC(n+1)_i$를 측정한다. (그림 6 (c))는 n+2번째 라운드에서 #c 큐에 새로운 데이터가 들어왔을 때의 상황이다. $QC(n+2)_i$를 현재 라운드의 SQ(n+2)와 동일한 값으로 설정하고 활성리스트의 제일 끝부분에 #c 큐의 인덱스 노드를 삽입시킨 다음 n+3번



(그림 6) 스케줄러의 수행 과정

째 라운드에서 서비스된다. (그림 6(d))는 (그림 6(c))에서 #c 큐의 인덱스 노드가 삽입된 상태를 보여준다.

4. 알고리즘 분석

4.1 제안한 알고리즘 분석 및 증명

4.1절에서는 제안한 알고리즘의 타당성을 입증하기 위한 3가지 lemma 및 각각에 대한 증명을 나타낸다.

m번째 라운드에서 i번째 큐의 QC를 $QC(m)_i$라고 하고, 모든 큐 안에 삽입되는 패킷의 사이즈가 Max라고 가정한다.

lemma 1 : $0 \le SQ(m) - QC(m)_i < Max$

proof : 라운드마다 큐 i에 대한 서비스가 끝났을 때 두 가지 가능성이 있다.

첫 번째, 만약, 큐 i에 패킷이 남아있을 경우 $QC(m)_i$은 최대 SQ(m)까지 서비스가 가능하므로 $SQ(m)-QC(m)_i < MaX$를 보장하고, $QC(m)_i$가 SQ(m)를 초과할 수 없으므로 $0 \le SQ(m)-QC(m)_i$가 성립한다.

두 번째, 만약, 큐 i 안에 패킷이 남아있지 않을 경우 제안된 알고리즘은 3.1절에서 설명한 것과 같이 $QC(m)_i$에 0이 설정된다. 따라서 $0 < SQ(m) - QC(m)_i$을 보장한다.

$sent_i(t_1, t_2)$를 i번째 큐가 시간 t_1에서 t_2 사이에 전송한 총 bytes이고, m을 t_1과 t_2 사이에 i번째 큐가 서비스된 라운드 횟수라고 가정한다(여기서 t_1은 0번째 라운드가 끝나는 시점이다).

lemma 2 : $SQ(m)-Max < sent_i(t_1, t_2) \le SQ(m)$

proof : 먼저, $sent_i(m)$을 m번째 라운드까지 큐 i에서 전송된 총 bytes이고, $bytes(k)_i$를 k번째 라운드에서 큐 i가 전송한 bytes라고 가정한다. 따라서 $sent_i(t_1, t_2) = sent_i(m)$이다.

$sent_i(m) = \sum_{k=1}^{m} bytes(k)_i$이고,

$bytes(k)_i + SQ(k)-QC(k)_i = SQ(k)-QC(k-1)_i$이 성립한다.

$bytes(k)_i = QC(k)_i - QC(k-1)_i$이고, $\sum_{k=1}^{m}(QC(k)_i - QC(k-1)_i)$ $= QC(m)_i$가 성립한다(최초 라운드에서 $QC(0)_i$의 값은 0이다). 따라서, $sent_i(m) = QC(m)_i$가 성립하는데, $QC(m)_i$는 m

번째 라운드까지 큐 i가 전송한 총전송량이므로 lemma 2가 성립한다.

결과적으로 lemma 1과 2에 의해 매 라운드마다 (4)의 $SQ(k)-QC(k-1)_i \geq bytes(k)_i$를 만족하여 스케줄러가 큐들을 방문할 때마다 아무리 사이즈가 큰 패킷도 최소한 하나 이상의 전송이 보장되므로 SQ 재설정으로 인한 지연시간을 최소화한다.

lemma 3 : 각 큐에 대하여 k라운드까지의 총전송량은 서로 같거나 거의 비슷한 값을 갖는다.

proof : $D(k)_i$를 SQ(k)와 $QC(k)_i$의 차라고 할 때, 큐들 중 현재 $QC(k)_i$ 가장 큰 값인 MaX($QC(k)_i$)에 대하여 SQ(k) = Max($QC(k)_i$)가 성립한다. 즉, $D(k)_i$는 Max($QC(k)_i$)에 대한 상대적인 결손량을 나타내고 $0 \leq D(k)_i \leq A(k)_i$를 만족한다.

따라서, $A(k)_i$가 각 큐에 대한 패킷 하나 사이즈라는 것을 고려할 때, 각 큐가 지금까지 전송한 총데이터 량은 같거나 거의 비슷한 값을 가지므로, 공정성에 있어서도 정확성을 보장한다.

4.2 작업 복잡도 비교

<표 1>은 제안한 알고리즘의 작업 복잡도를 분석하기 위해 DRR과 제안한 알고리즘을 각 단계적으로 연산처리 과정을 비교한 것이다.

〈표 1〉 DRR과 제안한 알고리즘의 단계적 비교

	DRR	제안된 스케줄링 알고리즘
1단계	패킷을 분류	패킷을 분류
2단계	각 큐에 대한 서비스가 끝난 후 각 큐에 대해 DC_i를 계산($DC_i = SQ_i - bytes_i$)	매 라운드가 끝난 후 각 큐에 대해 QC_i와 각 큐의 가장 앞쪽에 있는 패킷 사이즈(A_i)의 합을 계산
3단계	새로운 라운드가 시작되기 전에 각 큐마다 SQ_i를 계산 ($SQ_i = Q + DC_i$)	새로운 라운드가 시작되기 전에 2단계에서 구한 값들 중 가장 큰 값을 추출하여 SQ로 설정

1단계의 패킷 분류에서 DRR과 제안한 알고리즘은 단순히 패킷 헤더의 정보를 보고 패킷을 분류하므로 작업 복잡도는 같다. 2단계에서도 큐의 개수가 n개일 경우 DRR과 제안한 알고리즘은 각각 DC와 QC+A를 동일하게 n번씩 계산해야 함으로 작업 복잡도가 같다. 3단계에서도 DRR은 큐의 개수가 n개일 경우 SQ를 n번 계산하는 작업과 제안한 알고리즘의 2단계에서 계산된 각 큐의 QC+A 값들 중 가장 큰 값을 추출하여 SQ로 설정하는 작업의 복잡도가 동일하다. 패킷 사이즈를 누적시킨 값인 $bytes_i$와 QC_i를 계산하는데 있어서의 작업 복잡도가 동일하다는 것을 감안할 때 두 알고리즘의 작업 복잡도는 식 (5)의 관계를 만족한다.

DRR의 작업 복잡도 ≥ 제안한 알고리즘의 작업 복잡도 (5)

DRR의 작업 복잡도가 O(1)이므로[2, 9] 제안한 알고리즘의 작업 복잡도 또한 최대 O(1)이다.

5. 성능 분석 및 실험 결과

5.1 스케줄링 적용 환경

<표 2>는 제안한 스케줄링 기법을 모의 실험한 시스템 환경이다.

〈표 2〉 시스템 환경

	리눅스 시스템
CPU	인텔 펜티엄 Ⅲ 700MHZ
RAM	256M
HDD	20G
배포판	Power Linux 6.0
Kernel version	2.2.5-15
GCC version	2.91.66
Libc6 version	2.1.1
language	C

5.2 스케줄링 기법 비교 분석

<표 3>은 제안된 스케줄링 알고리즘과 DRR 및 WRR의 특성을 비교 분석한 것이다. 세 스케줄러 모두 작업 복잡도가 O(1)로 동일하고 필요한 큐의 개수에 있어서도 동일하다. 제안된 스케줄링 기법은 엔터프라이즈 환경에서 실시간 처리 요구 및 재전송 메커니즘을 요구하는 애플리케이션 특성을 고려한 반면 ORR, WRR, DRR은 고려하지 않았다.

〈표 3〉 제안된 스케줄러와 DRR의 비교

구분 \ 알고리즘	ORR	WRR	DRR	제안된 스케줄링 알고리즘
알고리즘 작업 복잡도	O(1)	O(1)	O(1)	O(1)
필요한 큐	P	P	P	P
패킷 사이즈 고려	×	○	○	○
애플리케이션 특성 고려	×	×	×	○

(○ : 고려함, × : 고려하지 않음)

5.3 스케줄링 기법의 실험 결과

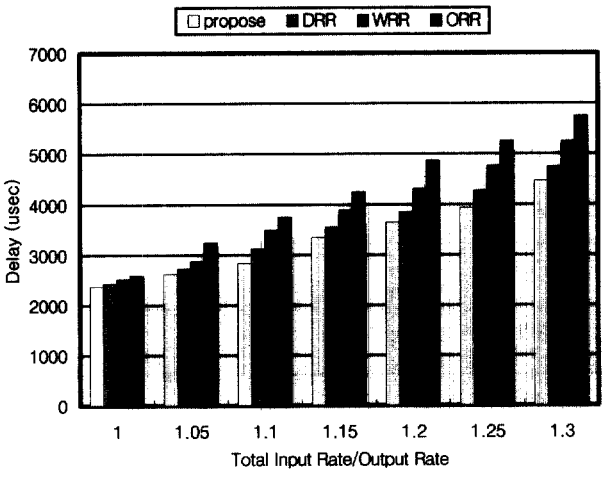
스케줄러의 지연 시간을 실험하기 위한 모델은 20개의 근

원지(source)에서 각각 100Mb/s 링크를 통해 스케줄러가 있는 라우터와 연결되어 있고, 라우터에서 목적지(destination)까지는 100Mb/s의 링크로 연결되어 있다. 실험을 위해 20개의 근원지에서 전송되는 트래픽들을 <표 4>와 같이 엔터프라이즈 환경에서의 애플리케이션 특성에 따라 4가지 클래스로 구분하였다.

(그림 7)은 각 큐에 패킷들이 입력되는 속도의 합에 대한 출력 링크로 전송되는 속도의 비율에 따라 큐 안에서 마지막 패킷이 서비스될 때까지 패킷 하나가 큐 안에 머물러 있는 평균 지연시간을 측정한 것을 왼쪽부터 제안한 스케줄링 기법, DRR, WRR, ORR 순으로 나타낸 것이다. 제안된 알고리즘이 매 라운드마다 전송을 앞둔 패킷을 포함한 총 전송량이 가장 큰 값으로 SQ를 재설정하는 기법을 사용하기 때문에 패킷 사이즈가 아무리 큰 패킷이라도 현재 라운드에서 서비스된다. 따라서 불필요한 SQ의 재설정 횟수 및 라운드 순회 횟수를 줄이기 때문에 (그림 7)과 같이 기존의 DRR 보다 지연시간이 감소하는 결과를 얻는다.

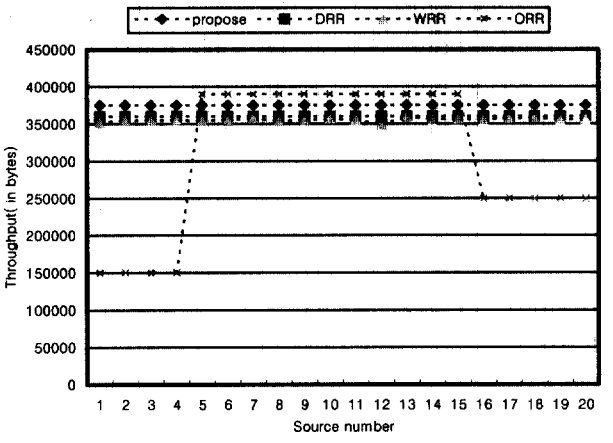
〈표 4〉 실험 조건

구 분	특 성
1~5 source (class 1)	· 패킷 사이즈 : 대략 100byte · 연결당 10kbps의 적은 대역폭 요구 · 재전송 메커니즘 요구
6~11 source (class 2)	· 패킷 사이즈 : 대략 100 byte 또는 수 kbyte · 연결당 100kbyte~1Mbps의 대역폭 요구 · 재전송 메커니즘 요구하지 않음
12~16 source (class 3)	· 패킷 사이즈 : 수 kbyte · 연결당 1~ 5Mbps 정도의 대역폭 요구 · 재전송 메커니즘 요구하지 않음
17~20 source (class 4)	· 패킷 사이즈 : 대부분 1500byte 이하 · 애플리케이션 종류에 따라 다양하나 대략 수십에서 수백 kbps의 대역폭 요구 · 재전송 메커니즘 요구



(그림 7) 지연시간 비교

(그림 8)는 각 근원지에 따른 처리량(Throughput)을 비교한 것이다. 제안된 스케줄러가 SQ의 설정 횟수 및 라운드 순회 횟수 감소로 DRR에 비하여 향상된 성능을 나타내고 있고, ORR은 패킷 사이즈와 전송 대역폭이 가장 큰 클래스 2, 3 이 가장 많은 처리량을 가지는 것으로 나타나 불공정성이 상당히 심한 것을 보이고 있다. WRR은 DRR에 비하여 상대적으로 고른 처리량을 가지는 것으로 나타났지만 각 큐에 할당된 가중치가 패킷의 평균 사이즈를 기반으로 설정되기 때문에 정확한 공정성을 보여주지는 못하고 있다.



(그림 8) 처리량 비교

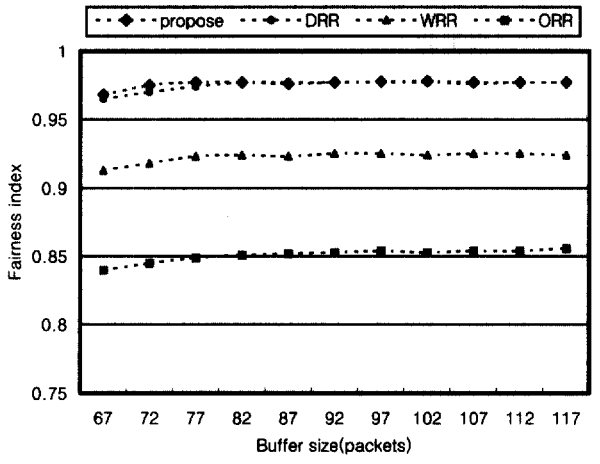
스케줄러의 전체적인 공정성을 측정하기 위한 모델은 ftp 서버/클라이언트의 5개의 쌍이 두 개의 라우터(스케줄러)를 통해 연결되어 있다. 연속된 두 라우터들은 전송 지연(propagation delay)이 20msec인 2Mb/s의 병목링크로 연결되어 있다. 서버와 라우터 그리고 라우터와 클라이언트 사이는 전송 지연이 0인 10Mb/s의 링크로 연결되어 있다. 공정 인덱스를 계산하기 위한 수식은 식 (5)과 같다(공정 인덱스의 최고 수치는 1이고 인덱스 값이 1에 가까울수록 공정성이 높다) [4, 16].

$$\text{Fairness index} = \frac{\left[\sum_{i=1}^{n} T_i\right]^2}{n\sum_{i=1}^{n} T_i^{\,2}} \qquad (6)$$

(T_i는 클라이언트 i가 받은 전송량, n은 클라이언트의 수)

(그림 9)에서 DRR은 버퍼 사이즈가 대략 77 packets 이전까지는 버퍼 안에 입력되지 못한 패킷들로 인해 공정 인덱스가 상대적으로 낮은 값을 나타내면서 증가하다가 77 packets 이후부터 동일한 수준을 유지한다. 제안한 패킷 스케줄러는 대략 72 packets부터 공정 인덱스 값이 거의 일정한 결과치를 보이면서 DRR과 비슷한 공정 인덱스 값을 나타났다. 제안한 스케줄러의 공정 인덱스 값이 72 packets 이후부터 일정한 것은 SQ에 대한 연산 횟수의 최소화로 패킷들이 스케

즐러에서 머무는 시간이 감소하여 버퍼 안에 입력되지 못한 패킷들의 수가 줄어들었기 때문이다. WRR와 ORR은 각각 큐의 사이즈가 대략 83 packets, 88 packets부터 거의 동일한 수준을 유지하는 것으로 나타났고 전체적인 공정성에서는 DRR과 제안된 스케줄러 보다 상대적으로 낮은 것으로 나타났다. 따라서 제안된 스케줄링 알고리즘 또한 정확한 공정성을 보장한다.



(그림 9) 공정성 비교

6. 결론 및 향후 과제

대부분의 네트워크에서 사용되는 최선 노력(best-effort) 방식은 스케줄러로 입력되는 패킷의 사이즈와 입력되는 속도가 상대적으로 큰 트래픽이 큐를 점령하는 경우가 발생하여 다른 트래픽들에 대한 병목 및 기아(starvation)가 발생한다. 따라서 다양한 애플리케이션들을 각각 독립적으로 관리하면서 공정한 서비스를 보장하는 패킷 스케줄링 기법이 필요하다.

공정한 서비스 제공을 위한 많은 패킷 스케줄링 기법들이 개발되었고, 그 중에서 DRR은 다른 것들에 비해 구현이 쉽고 낮은 작업 복잡도를 가지면서 정확한 공정성을 보이는 패킷 스케줄링 알고리즘이다[2, 7, 9]. DRR은 매 라운드마다 각 트래픽들의 상대적인 결손량을 보상하여 공정성을 보장하는 것이다. 이것은 모든 큐에게 부여된 동일한 할당량 Q(Quantum)을 고정으로 설정해 놓는 기법이므로 엔터프라이즈 환경에서 패킷 사이즈가 수 kbyte에 이르고 경성 실시간 처리를 요하는 멀티미디어 패킷들에 대해 서비스 할당량 SQ에 대한 불필요한 재설정 횟수 및 라운드 순회 횟수로 인해 전송 시간을 지연시킨다.

본 논문에서는 이러한 문제를 해결하기 위해 기존의 DRR 기법을 변형한 기법으로 정확한 공정성을 보장할 뿐만 아니라 서비스 할당량 SQ를 전송을 준비중인 패킷 사이즈에 따라 동적으로 설정하도록 하여 지연시간을 최소화하였다.

향후 연구 과제로는 서비스 지연을 더욱 최소화하기 위해 활성리스트에 대한 더욱 정교한 운영 방법을 연구하는 것이다.

참 고 문 헌

[1] Hemant M. Chaskar, U. Madhow, "Fair scheduling with tunable latency : a round robin approach," GlOBECOM (Global Telecommunications Conference), Vol.2, pp.1328-1333, 1999.

[2] Salil S. kanhere, Alpa B. Parekh and Harish Sethu, "Fair and Efficient Packet Scheduling in Wormhole Networks," Proc. of IPDPS(International Parallel and Distributed Processing Symposium), pp.623-631, 2000.

[3] O. Altintas, Y. Atsumi, T Yoshida, "Urgency-based round robin : a new scheduling discipline for packet switching networks," ICC(IEEE International Conference Communication), Vol.2, pp.1179-1184, 1998.

[4] Hakyong Kim, Yongtak Lee and Kiseon Kim, "Fairness concept in terms of utilisation," IEEE Electronics Letters, Vol.36, No.4, 17th Feb. 2000.

[6] M. H. MacGregor, W. Shi, "Deficits for bursty latency-critical flows : DRR++," Proc. of ICON(IEEE International Conference), pp.287-293, 2000.

[7] Onur Altintas, Yukio Atsumi, Teruaki Yoshida, "A note on fair queueing and best-effort service in the Internet," IWS(Internet Workshop), pp.145-150, 1999.

[8] G. Hasegawa, T. Murata, H. Miyahara, "Comparisons of packet scheduling algorithms for fair service among connections on the Internet," Proc. of INFOCOM(Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies), Vol.3, pp.1253-1262, 2000.

[9] M. Shreedhar, G. Varghese, "Efficient Fair Queuing Using Deficit Round-Robin," IEEE/ACM Transaction, Vol.4, No. 3, pp.375-385, June, 1996.

[10] Hui Zhang, "Service disciplines for guaranteed performance service in packet-switching networks," Proc. of IEEE, Vol. 83, Issue 10. pp.1374-1396, June, 1996.

[11] 유항재, 김태윤, "엔터프라이즈 환경에서의 멀티미디어 QoS 프레임워크", 고려대학교 석사학위논문, 1999.

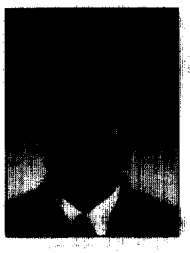
[12] 김병철, 김태윤, "엔터프라이즈 환경에서 서비스 요구 사항을 지원하는 패킷 스케줄링 알고리즘", 정보과학회논문지 : 정보통신, 제27권 제3호, 2000.

[13] Erik. Nordmark, "Contribution of packet sizes to packet and byte volumes," Internet Draft. Jun. 1997.

[14] Christian Huitema, "Routing in the internet," Prentice Hall PTR, pp.27-61, 1995.

[15] G. Fairhurst, "Maximum Transfer Unit (MTU)," Internet Draft, 1998.

[16] Igor B.H. de A. Alves, Jose F. de Rezende, Luis Felipe M. de Moraes, "Evaluating Fairnes in Aggreagated Traffic Marking," GLOBECOM'00, IEEE, Vol.1, pp.445-449, 2000.



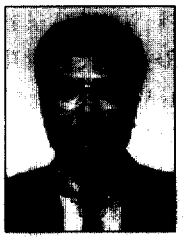
윤 여 훈

e-mail : joy1223@netlab.korea.ac.kr

2000년 동국대학교 전산통계학과 졸업 (학사)

2000년~현재 고려대학교 컴퓨터학과 석사 과정 재학중.

관심분야 : 라우팅 프로토콜, 망관리, wireless



김 태 윤

e-mail : tykim@netlab.korea.ac.kr

1981년 고려대학교 산업공학과 학사

1983년 미국 Wayne State University 전산과학과 석사

1987년 미국 Auburn University 전산과학과 박사

1988년~현재 고려대학교 컴퓨터학과 교수

관심분야 : 전자상거래, 컴퓨터 네트워크, EDI, 이동통신, 멀티미디어 등