# 고속 네트워크 환경에서 멀티미디어 데이터를 지원하는 리눅스 기반 QoServer 개발

윤여훈, 김태윤
고려대학교 컴퓨터학과
e-mail : joy1223@netlab.korea.ac.kr

# Linux Based QoServer Development Supporting Mutimedia Data In High Speed Network Environment

Yeo-Hoon Youn, Tai-Yun Kim
Dept. of Computer Science & Engineering, Korea University

요 약

오늘날 네트워크의 대역폭이 커지고 동시에 실시간 처리를 요하는 다양한 멀티미디어 애플리케이션들이 생성되고 있다. 그러나 문제는 고속 LAN 환경에서 많은 사용자들이 멀티미디어 애플리케이션들을 비롯한 다양한 네트워크 서비스들을 사용하고 있지만, WAN 환경으로의 선로로 전송하는데 있어서의 차별화가 없다는 것이다. 따라서 경성 실시간(hard real time) 처리를 요하는 멀티미디어 데이터들의 시간 제한을 지켜줄 수 없고, 비교적 지연시간의 제약을 받지 않는 HTML, FTP, e-Mail, 등의 연성 실시간(soft real time) 처리를 요하는 애플리케이션들에 대해 불필요한 대역폭 낭비를 일으킨다.

이러한 문제를 최소화하기 위해 본 논문에서는 엔터프라이즈 네트워크 등과 같은 고속 네트워크 망을 사용하는 환경에서 다양한 멀티미디어 데이터 패킷들을 고정적으로 할당된 대역폭에 따라 우선적으로 서비스되도록 하여 지연시간 제한을 최대한 보장해 주기 위한 리눅스 상에서 구현된 QoServer 개발 기술을 소개한다.

## 1. 서 론

오늘날 네트워크에서의 음성 데이터를 비롯한 멀티미디어 데이터의 서비스를 보장하는데 있어서의 문제점은 LAN 환경에서의 다양한 패킷 요구 사항이 존재한다는 것이다. 즉, LAN 환경에서의 많은 사용자들은 다양한 네트워크 서비스를 사용하고 있는데, 이것을 WAN 환경으로의 선로로 전송하는데 있어서의 차별화가 없다는 것이다. 그래서 지연시간의 제한을 많이 받는 멀티미디어 데이터들의 시간 제한을 지켜줄 수 없고, 비교적 지연시간의 제약을 받지 않는 HTML, FTP, e-Mail 등의 애플리케이션들에 대해 불필요한 대역폭 낭비를 일으킨다.

본 논문에서는 이러한 문제를 최소화하기 위해 경성 실시간 처리를 요하는 멀티미디어 애플리케이션들을 우선적으로 서비스하는 기술을 제안한다. 특히, 엔터프라이즈 네트워크와 같이 고속 네트워크 망을 사용하는 환경에서 패킷 사이즈가 최대 수 kbytes에 이르는 멀티미디어 데이터 패킷들을 우선적으로 서비스하기 위해 개발된 QoServer 기술을 소개한다. 이것은 호스트나 서버에서 전송한 패킷들이 라우터를 통과하기 전에 리눅스로 구현된 QoServer를 통과하도록 하여 멀티미디어 패킷들에 대해 고정적으로 설정된 대역폭을 갖고 우선적으로 서비스하도록 함으로써 실시간 요구도를 최대한 보장해 주는 기술이다.

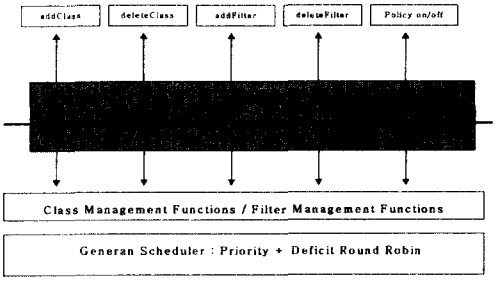
제시한 기술은 기존의 리눅스 시스템에서 적용하였는데, 본 논문에서는 이것이 리눅스 시스템에서 어떻게 적용되고 구현되는지를 나타내고 있다.

QoServer를 구현하기 위해 기존의 리눅스 시스템에서 제공되는 단일(single) 큐를 확장하여 사용자가 필요에 따라 여러 개의 큐를 생성할 수 있도록 했으며, 생성된 큐들에 대하여 제안한 패킷 스케줄링 기법을 적용하여 서비스하도록 하였다.

## 2. QoServer의 구조

QoServer는 클래스(class)와 필터(filter)라는 두 가지 중요 개념으로 구성되어 진다. 클래스라 함은 대역폭을 보장하는 기본적인 단위이며 각각의 큐는 각기 다른 클래스에 속해있다 [3], [5]. 필터 역시 클래스의 구성 요소로써 삽입되어 들어오는 패킷들을 각 클래스별로 분류하는 기준이 된다[3], [5]. 이러한 클래스 기반 연산을 지원하기 위하여 addclass, deleteclass 등의 오퍼레이션 등이 존재하고 이들은

시스템 콜을 사용하여 커널 내부와 통신한다.




<그림 1> QoServer 구조

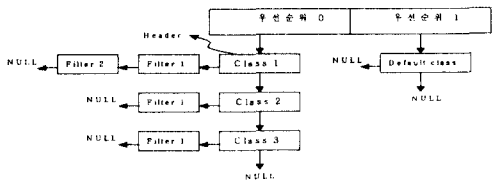
## 3. 스케줄러 구조

### 3.1 기존의 리눅스 패킷 스케줄러 구조

리눅스에서 기본적으로 제공되는 스케줄링 기법은 하나의 큐를 기반으로 한 FCFS(First Come First Serve) 방식이다. FCFS [9] 방식은 enqueue()와 dequeue() 함수의 단순한 동작에 의해서 수행된다. enqueue()는 들어오는 모든 패킷들에 대해서 필터링(filtering)을 수행함 없이 기본적으로 제공되는 하나의 큐에 무조건 삽입시키는 역할을 하고, dequeue()는 큐에 들어오는 순서대로 패킷을 출력링크(output link)로 전송하는 역할을 한다. 이러한 특성으로 인해 트래픽들의 서비스 목적과는 상관없이 모든 패킷에게 동일한 수준의 서비스를 제공하기 때문에 알고리즘 자체는 단순하고 낮은 작업 복잡도(work complexity)를 가지지만 효율적인 스케줄링 기법은 아니다.

### 3.2 QoServer 패킷 스케줄러 구조

먼저 QoServer 패킷 스케줄링 기법에서 클래스들과 필터들의 상관관계에 대한 대략적인 내용은 <그림 2>와 같다. 여기서 클래스라 함은 큐의 확장된 개념이고, 필터는 패킷을 분류하는 기준이 된다.




<그림 2> QoServer 스케줄러 구조

클래스 우선 순위는 배열을 이용해서 나열 시켰으며, 우선 순위 0은 우선 순위 0에 해당하는 클래스들이 링크드 리스트로 연결된 클래스 리스트의 처음(header)을 가리키게 한다. 우선 순위 1의 default 클래스는 우선 순위 0 클래스들 중 어느 곳에도 속하지 않은 패킷들이 삽입되는 곳이다.

우선 순위 0에 해당하는 클래스들은 우선 순위 0 아래에 기능함수(add_class(), delete_class())를 사용하여 생성 및 제거할 수 있다. 각 클래스에는 각 클래스의 특성(IP주소, port 번호, 프로토콜…)을 담고 있는 곳이라고 할 수 있는 필터들이 서로 연결되어 있다. 우선 순위 0에 해당하는 클래스들에는 각 클래스 필터의 특성에 해당하는 멀티미디어 패킷들이 삽입되고, 클래스 1에 해당하는 default 큐에는 멀티미디어 패킷 외의 패킷들이 삽입된다.

각 클래스에 삽입된 패킷들을 제안한 패킷 스케줄링 기법을 사용하여 외부 출력 링크로 서비스된다.

제안한 패킷 스케줄링 기법은 엔터프라이즈 네트워크 환경과 같은 고속 네트워크 망에서의 다양한 애플리케이션들의 패킷 사이즈와 그 특성 [10]을 고려한 것으로, 기존의 DRR(Deficit Round Robin) [8], [9]을 수정한 것이다. 이 기법은 매 라운드를 시작하기 전에 각 큐에서 전송을 앞둔 패킷들 중 가장 앞쪽에 있는 패킷과 현재까지 전송한 총데이터 량(bytes)의 합들 중 가장 큰 값을 SQ(Service Quantum)로 설정하는 기법이다. 이것은 각 큐의 가장 앞쪽에서 전송을 앞둔 패킷들 중 사이즈가 아무리 큰 패킷도 현재 라운드에서 서비스될 수 있도록 하여 불필요한 SQ 재설정 횟수 및 라운드 순회 횟수를 줄여 기존의 DRR에 비하여 지연시간을 최소화할 수 있다.

QoServer로 패킷이 들어왔을 때, enqueue 모듈에서는 <그림 2>에서 우선 순위 0에 해당하는 클래스들의 필터들을 차례로 검색하여 필터의 정보와 패킷 헤더의 정보가 일치하면 해당 클래스에 패킷을 삽입한다. dequeue 모듈에서는 enqueue 모듈에서 삽입된 패킷들을 제안된 스케줄링 기법에 따라 우선 순위 0에 해당하는 클래스들과 우선 순위 1에 해당하는 default 클래스를 외부 출력 링크로 차례로 서비스한다.

## 4. 스케줄링 적용 시점

리눅스 시스템으로 패킷 하나가 들어올 때마다 패킷에 대한 정보를 포함하는 소켓 버퍼가 형성된다. 커널 내에서 소켓 버퍼는 include/linux/sk_buff.h/ struct skb_buff와 같은 구조체 형태를 지닌다. 패킷이 삽일 될 때마다 형성된 소켓 버퍼는 skb 형태로 각 계층으로 전달되고 데이터 링크 계층에 속하는 /usr/src/linux/net/core/dev.c/dev_queue_xmit() 함수까지 skb가 전달된다. 그리고 이 함수에 의해 enqueue(), dequeue() 함수가 비동기적으로 호출되는데 [7], enqueue() 함수는 skb를 큐에 삽입하는 역할을 하고 dequeue()는 큐에서 skb를 외부 출력 링크로 전송하는 역할을 한다.

dev_queue_xmit()이라는 함수 내부에 있는 q->enqueue() 함수에 의해 실질적인 enqueue() 함수인 /usr/src/linux/net/sched/generic.c 파일 내에 있는 pfifo_fast_enqueue() 함수가 호출된다. 또한 dev_queue_xmit() 함수 내부에 있는 qdisk_wakeup() 함수에 의해 /usr/src/linux/net/sched/generic.c 파일 내부에 있는 qdisc_restart() 함수가 호출되고, 이 함수 내의 q->dequeue() 함수에 의해 동일 파일 내에 있는 실질적인 dequeue() 함수인 pfifo_fast_dequeue() 함수가 호출된다.

커널 내부에서 enqueue와 dequeue를 담당하는 최종 함수들이 바로 /usr/src/linux/net/sched/generic.c

파일 내에 있는 pfifo_fast_enqueue()와 pfifo_fast_dequeue() 함수이다. 이 두 함수는 각각 비동기적으로 호출되어 패킷을 큐에 삽입시키면서 외부 출력 링크로 전송한다.

본 논문에서 QoServer에 적용한 스케줄링 정책을 위해 pfifo_fast_enqueue() 함수 내부에 있는 단일 큐를 확장하여 여러 개의 큐를 생성할 수 있도록 하였고, pfifo_fast_dequeue() 함수에서 QoServer 패킷 스케줄링 정책에 따라 각각의 큐를 서비스하도록 하였다.

기존 리눅스 시스템 내부에 있는 pfifo_fast_enqueue() 함수 내의 단일 큐는 <그림 3>과 같다.

```
static int
pfifo_fast_enqueue(struct sk_buff skb, struct Qdisc qdisc)
 {
   struct sk_buff_head *list;
    list = (struct skb_buff_head*)qdisc->data;
            :
           __skb_queue_tail(list, skb);
            :
  }
```

<그림 3> enqueue() 함수

<그림 3>에서 구조체 변수 list가 리눅스 시스템에서의 단일 큐이다. skb 형태의 모든 패킷을 어떠한 처리도 없이 __skb_queue_tail(list, skb) 함수를 이용하여 단일 큐 list로 삽입한다.

본 논문에서는 기존의 단일 큐를 <그림 4> 같이 클래스라는 구조체 내에 다른 요소들과 함께 포함시켜 여러 개의 클래스를 생성한 다음 <그림 2>와 같은 구조로 기존의 FCFS 스케줄링 정책을 개선하였다.

```
   struct Class_queue {
          :
     struct Class_queue *next;
          :
     strcut filter * filter_list;
          :
     struct sk_buff_head skt;
          :
   }
```

<그림 4> 클래스 구조체

<그림 4>에서 struct Class_queue *next 는 다음 클래스를 가리키는 포인터이고, <그림 4>와 유사한 형태로 패킷을 분류하기 위한 정보를 설정할 수 있는 필터 구조체가 있는데 struct filter *filter_list가 해당 클래스의 필터를 나타내는 필터 구조체 포인터이다. 필터 구조체 내부에도 클래스와 같이 다음 필터를 가리키는 struct filter *next 부분이 있다. struct sk_buff_head skt 는 해당 클래스의 큐를 나타내고 있다.

이러한 형태의 클래스 또는 필터 구조체를 만든 것은 <그림 2>와 같이 기존 리눅스 시스템의 단일 큐를 확장함에 있어서 유연성을 제공하기 위해서이다.

## 5. 결론 및 향후 과제

오늘날 네트워크의 대역폭이 커지고 동시에 실시간 처리를 요하는 수많은 멀티미디어 애플리케이션들이 생성되고 있다. 그러나 LAN 환경에서의 많은 사용자들은 다양한 네트워크 서비스를 사용하고 있는데, 이것을 WAN 환경으로의 선로로 전송하는데 있어서의 차별화가 없다는 것이다. 따라서 경성 실시간 처리를 요하는 멀티미디어 데이터들의 시간 제한을 지켜줄 수 없고, 비교적 지연시간의 제약을 받지 않는 HTML, FTP, e-Mail, 등의 애플리케이션에 대해 불필요한 대역폭 낭비를 일으킨다. 따라서 본 논문에서는 엔터프라이즈 네트워크 환경과 같이 고속 네트워크 망을 사용하는 환경에서 다양한 애플리케이션들의 특성을 고려하여 서비스를 제공하는 QoServer 기술을 소개하였다.

이것은 호스트나 서버에서 전송된 데이터들이 라우터를 통과하기 전에 리눅스로 구현된 QoServer를 통과하도록 해서 멀티미디어 데이터들이 고정적으로 설정된 대역폭을 갖고 우선적으로 서비스되도록 하여 실시간 요구도를 최대한 보장해 주는 기술이다.

## 참고문헌


[1] Armitage, "Quality of Service in IP Networks: Foundations for a Multi-Service Internet", MTP, 2000.

[2] Floyd, S., and Jacobson, V., Link-sharing and Resource Management Models for Packet Networks", IEEE/ACM Transactions on Networking, Vol, 3 Vol. 4, pp. 365-386, August 1995.

[3] Floyd, S., "Notes on CBQ and Guaranteed Service", Draft document, July 1995.

[4] F. Fisso and B. Gevros, "Operational and *performance* Issues of a CBQ Router, CCR, October 1999.

[5] Floyd, S., Notes of Class-Based Queueing: Setting Parameters", Informal notes, September 1995.

[6] Kenjiro Cho, "Managing Traffic with ALTO." Proceedings of USENIX 1999 Annual Technical Conference: FREENIX Trak, Monterey CA, June 1999.

[7] Saravanan Radhakrishnan, "Linux-Advanced Networking Overview: Version1", Information and Telecommunications Technology Center, August 22, 1999.

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

[9] 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.

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