우선순위 정책과 피드백 리소스 제어를 이용하여 적응적 QoS를 제공하는 네트워크 프로토콜 처리기법

이승만° 김지민 유민수
한양대학교 전자컴퓨터통신공학과
{smlee°, jmkim, msryu}@rtcc.hanyang.ac.kr

Adaptive QoS-aware Network Protocol Processing via Priority-based Resource Allocation and Feedback Resource Control

Seungman Lee°, Jimin Kim, Minsoo Ryu
Department of Electronics & Computer Engineering, Hanyang University

요 약

본 논문에서는 네트워크 응용프로그램의 최적의 성능(QoS)을 보장하기 위한 우선순위 기반의 프로토콜 처리기법을 제안한다. 시스템 자원이 제한된 상황에서 다수의 응용프로그램들이 원활히 수행되기 위해서는 우선순위를 기반으로 자원을 배분하는 것이 바람직하다. 특히 우선순위 기반의 시스템에서 네트워크 응용프로그램의 성능은 운영체제의 프로토콜 처리방식에 따라서 좌우된다. 제안하는 기법은 프로토콜 처리율과 응용프로그램의 성능과의 관계를 추적하여 최적의 성능을 나타내는 지점을 검출하고, 우선순위에 따라서 프로토콜 처리율을 차별화함으로써 각 응용프로그램들의 성능을 조절한다. 본 논문에서 제안하는 프로토콜 처리기법을 검증하기 위해서 리눅스 커널에 이를 구현하였으며 실험을 통하여 우선순위에 따라서 네트워크 응용프로그램의 성능이 보장됨을 확인하였다.

1. 서론

컴퓨터 시스템의 발전으로 범용 컴퓨터 시스템에서부터 임베디드 시스템에 이르기까지 운영체제의 네트워킹과 멀티태스킹에 대한 지원은 이제 필수요소가 되었다. 더욱이 최근 등장하는 네트워크 기반의 응용프로그램들은 고속의 대용량 데이터 전송을 요구하고 있다. 시스템의 자원의 양이 충만하다면 응용프로그램은 더 나은 성능과 서비스를 제공할 수 있다. 그러나 시스템 자원은 유한하며, 동시에 실행되는 응용프로그램들의 자원요구량이 많을수록 모든 응용프로그램의 자원에 대한 요구사항을 만족시키기는 어려워진다. 따라서 다수의 네트워크 응용프로그램들의 QoS을 만족시키기 위해서는 효과적으로 시스템 자원을 분배하는 것이 대단히 중요하다.

컴퓨터 네트워크 분야에서 QoS(Quality of Service)는 일반적으로 유저들 또는 응용프로그램들에 따라서 차등의 우선순위를 제공하여 응용프로그램의 성능을 차별화하는 것을 말한다. 네트워크 응용프로그램의 QoS를 만족시키기 위해서는 네트워크상의 end-to-end 패킷 전송에 관여하는 모든 컴포넌트(component)들이 협력적(cooperative)으로 지원해야 한다. 즉, 네트워크를 구성하는 라우터나 스위치 등의 하드웨어 장치뿐만 아니라 네트워크 프로토콜 소프트웨어가 QoS를 지원해야하며, 각 end-host들 내부의 운영체제 및 자원분배(resource allocation) 메커니즘 역시 네트워크 QoS를 지원하기 위한 프레임워크가 갖추어져야 한다. QoS 지원에 대한 연구는 주로 네트워크 데이터전송 분야에서 진행되어 온 반면에 운영체제 측면에서의 연구는 미흡한 실정이다. 그러나 end-host에서 수행되는 운영체제가 QoS를 효과적으로 지원하지 못하면 사용자 또는 어플리케이션의 QoS를 만족시키기 어렵다.

본 논문에서는 운영체제 수준에서 효과적으로 QoS를 지원할 수 있는 네트워크 프로토콜 처리 방법 및 시스템 자원할당 기법에 대하여 다룬다. 제안하는 기법은 프로토콜 처리율과 응용프로그램의 성능과의 관계를 추적하여 최적의 성능을 나타내는 지점을 검출하고, 우선순위에 따라서 프로토콜 처리율을 차별화함으로써 각 응용프로그램들의 성능을 조절한다.

본 논문의 2절에서는 QoS지원 네트워크 프로토콜 처리와 관련되어 수행되었던 연구내용을 소개하고, 3절에서는 제안하는 패킷처리율 제한을 통한 자원할당 기법에 대해서 논의한다. 4절에서는 제안하는 기법의 성능에 관해서 기술하고, 마지막으로 5절에서는 결론 및 향후 연구과제에 대해서 논의한다.

2. 관련 연구

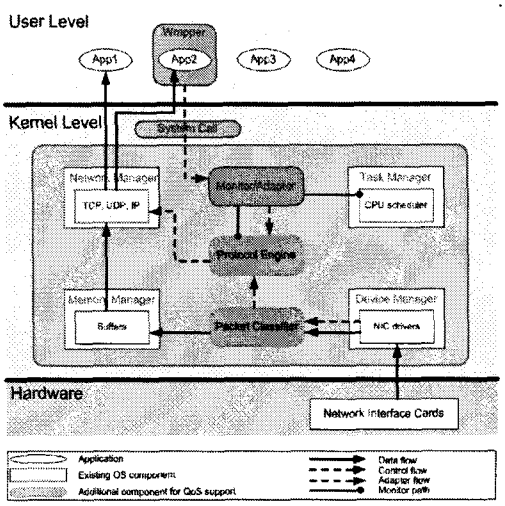
네트워크 응용프로그램의 QoS를 보장하기 위한 운영체제 수준의 서비스에 관한 연구들에 대해서 살펴본다. [1]은 Q-RAM으로 명명된 QoS 기반의 다중 자원할당 모델을 정의하고, 최적의 자원할당을 통해서 전체적인 CPU utilization이 최대가 될 수 있는 알고리즘을 제시한다. [1]에서는 시간적인 적시성, 데이터 전송의 신뢰성과 보안성 그리고 데이터 자체의 품질 등을 보장해 주어야 하는 QoS 요소들로 여기고, 각 QoS 요소들에 대해서 응용프로그램이 요구하는 최소 자원의 양을 검출하고 전체 자원을 적절히 배분함으로써 전반적으로 CPU utilization을 높일 수 있도록 한다. 하지만 이는 다중의

QoS 요소들에 대한 정의가 모호하며, 실제로는 각각의 QoS 요소들 간에 상호연관을 가지므로 구현하기가 매우 까다롭다.

[2]는 네트워크 end-to-end 시스템에서 커널의 자원 관리자들 사이에 통신채널을 형성하고 이벤트를 교환함으로써 응용프로그램의 QoS를 보장하도록 하는 미들웨어를 제시하였다. 각 end-host 내부의 커널은 자원관리자와 자원 감시자를 두어서, 상대방의 가용한 시스템 자원의 상황에 따라서 전송하는 데이터의 품질을 조절함으로써 전반적으로 시스템이 안정적으로 동작하도록 한다. 그러나 이러한 접근방법은 데이터 전송의 병목현상이나 자원 할당량 조절을 통한 성능향상기법이 아니라, 전송하는 데이터의 품질을 저하시킴으로써 간접적으로 시스템을 안정화 시키는 방법이다.

[4]와 [5]는 본 연구의 선행연구로서, 우선순위를 기반으로 네트워크 프로토콜 처리를 수행함으로써 네트워크 응용프로그램들의 QoS를 만족시켜 나가는 방법을 제시한다. [4]에서는 기존의 FIFO 순서로 진행되는 네트워크 프로토콜 처리를 우선순위 기반으로 처리하기 위한 기법과 아키텍처를 제시하였다. [5]에서는 이를 리눅스 운영체제 상에 커널 쓰레드와 bottom-half 형태로 구현하기 위한 방법들을 제시하였으며, 각각의 구현에 대해서 성능을 측정하고 평가하였다.

본 논문에서는 [4]와 [5]에서 제시된 프로토콜 처리기법을 이용하여 시스템 자원을 차등으로 할당함으로써 QoS를 만족시키는 방안에 대해서 논의한다. [4]와 [5]에서 제안하는 QoS지원 프로토콜 처리 아키텍처는 아래 그림과 같이 커널 수준에서의 프로토콜 처리엔진(Protocol Engine), 패킷 분류기(Packet Classifier), QoS 모니터/어댑터(Monitor/Adapter) 그리고 사용자 수준의 응용 프로그램 랩퍼(Wrapper)로 구성된다.



<그림 1 우선순위기반 프로토콜처리 아키텍처>

각 컴포넌트들의 기능은 다음과 같다.

#응용프로그램 랩퍼:

응용 프로그램 랩퍼는 기존에 생성된 응용 프로그램의 수정이나 재컴파일을 수행하지 않고서도 QoS를 만족시키기 위해서 사용되는 랩퍼 코드이다. 랩퍼는 사용자로부터 해당 응용 프로그램의 우선순위를 입력받아서, 커널 수준의 시스템 콜을 통해 QoS 모니터/어댑터에게 태스크의 우선순위를 전달한다.

#모니터/어댑터:

모니터/어댑터는 시스템의 자원 할당량 검출과 제한을 담당한다. 모니터는 랩퍼로부터 네트워크 연결에 대한 정보와 우선순위를 시스템 콜을 통해서 전달받고, 모니터링 주기마다 각 네트워크 연결에 대해서 시스템 자원 사용량을 모니터링한다. 어댑터는 모니터로부터 수집한 정보를 이용해서 네트워크 연결의 우선순위에 따라서 시스템 자원의 주기적인 재분배를 담당한다.

#패킷 분류기:

패킷 분류기는 네트워크 프로토콜 처리에 우선순위를 반영할 수 있도록 각각의 네트워크 연결별로 early packet demultiplexing[6]을 수행한다. 즉 NIC에 패킷이 도착하면 패킷을 네트워크 연결별로 분류하여 커널의 메모리에 존재하는 해당 버퍼에 저장한다.

#프로토콜 처리엔진:

프로토콜 처리엔진은 프로토콜 처리서버와 QoS지원 자료구조 그리고 패킷버퍼로 구성된다. 프로토콜 처리서버는 네트워크 응용 태스크가 설정한 각 네트워크 연결에 대해서 QoS지원 자료구조와 패킷버퍼를 생성한다. 패킷버퍼는 NIC(Network Interface Card)에서 전달받은 네트워크 패킷을 저장하는 FIFO 형태의 버퍼이며, 프로토콜 처리서버는 패킷버퍼에 저장되어진 패킷에 프로토콜 처리를 수행하는 서버이다. 프로토콜 처리서버는 커널 쓰레드로 구현되거나 리눅스의 softirq 메커니즘을 활용하여 구현될 수 있다. 응용프로그램은 래퍼를 통해서 해당 프로토콜 처리서버의 스케줄링 우선순위를 커널의 시스템 콜을 이용하여 지정한다. 스케줄링 우선순위는 기본적으로 해당 태스크의 우선순위로 지정된다.

3. 패킷 처리율 제한을 통한 자원할당 기법

본 절에서는 네트워크 응용프로그램의 QoS를 정의하기 위한 모델과 가정에 대해서 설명하며, 이를 통해서 시스템 자원할당 문제를 해결하기 위한 방안에 대해서 알아본다.

3.1 QoS모델과 가정

본 논문에서 논의하는 태스크는 다음과 같이 모델링된다.

[표 1 태스크 모델링]

$\tau_i = < T_i, D_i, E_i, P_i >$

τ_i: 태스크(task)

T_i: τ_i의 주기(Period)

D_i: τ_i의 데드라인(Deadline)

E_i: τ_i의 최대 실행시간(max. execution time)

P_i: τ_i의 우선순위(priority)

또한 네트워크 응용프로그램의 QoS를 나타내기 위해서 다음의 [표 2]와 같은 부가적인 속성을 정의한다.

[표 2 네트워크 응용프로그램의 QoS 파라미터]

R_{ik}: $(k-\Delta, k)$ 구간에서 τ_i에게 할당된 자원

$R_{ik} = < CPU_{ik}, MEM_{ik}, NET_{ik} >$

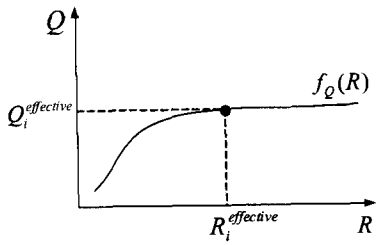
CPU_{ik}: 구간에서 τ_i에게 할당된 CPU 시간

MEM_{ik}: 구간에서 τ_i에게 할당된 메모리 량

NET_{ik}: 구간에서 τ_i에게 할당된 네트워크 처리율

Q_{ik}: 구간에서 τ_i에게 제공되는 QoS

태스크 τ_i는 주어진 시간 구간에서, 운영체제로부터 시스템 자원 R_{ik}을 할당받아야만 수행된다. 시스템 자원은 CPU시간과 메인 메모리 그리고 네트워크 프로토콜 처리율로 분류될 수 있다. 태스크에게 제공되는 QoS는 할당된 시스템 자원의 양이 증가할수록 높아진다. 그러나 본 논문에서는 아래 <그림 2>와 같이, 자원을 계속해서 추가 할당해 주어도 QoS가 증가하지 않는 지점이 존재한다고 가정한다. 이것은 응용프로그램의 고유한 특성으로 여겨질 수 있다. 예를 들면, 멀티미디어 동영상 스트리밍 재생프로그램은 동영상의 크기나 Frame Rate, 부호화 종류 등에 따라서 CPU 자원 사용량이 달라진다. 따라서 현재 재생하려는 동영상의 복호화에 필요한 시스템 자원보다 많은 양을 제공하더라도 여분의 자원은 사용하지 않는다. FTP 응용프로그램은 시스템 자원이 증가할수록 전송률이 높아지지만, 결국에는 네트워크 대역폭 등의 시스템 자원의 한계에 이르게 되어서 FTP 응용프로그램 역시 QoS가 증가하지 않는 지점이 존재한다. 본 논문에서는 이러한 지점에서의 시스템 자원 사용량을 적정자원요구량($R_i^{effective}$)이라 정의하고, 이때의 QoS를 유효 QoS($Q_i^{effective}$)라고 정의한다.



<그림 2 가정하는 시스템 자원할당량과 QoS의 관계>

또한 태스크에게 제공되는 QoS는 할당된 자원과 아래 [표 3]과 같은 함수관계 $Q_{ik} = f_Q(R_{ik})$가 성립된다고 가정한다.

[표 3 QoS 함수에 대한 가정]

$$\begin{cases} Q_{ik} = f_Q(R_{ik}) \\ \quad = f_Q(CPU_{ik}, MEM_{ik}, NET_{ik}), & if\ R_{ik} < R_i^{effective} \\ Q_{ik} = Q_i^{effective}, & if\ R_{ik} \geq R_i^{effective} \end{cases}$$

여기서, $f_Q()$는 단조 증가함수이다.

($f_Q()$ is a monotonically increasing function.)

3.2 문제제기 및 해결방안 제시

3.2.1 문제제기

네트워크 응용프로그램의 유효QoS를 만족시키기 위해서 운영체제가 제공해야하는 적정자원요구량을 찾는 문제는 다음과 같이 명확하게 나타낼 수 있다.

[표 4 QoS보장 방안에 대한 문제제기]

Find $Q_{ik}^{effective}$ for each task τ_i in any time interval k with any resource allocation, and

Find $R_{ik}^{effective}$ such that $f(R_{ik}^{effective}) \geq Q_{ik}^{effective}$

3.2.2 파라미터 단순화

일반적으로 컴퓨터 시스템의 운영체제는 다수의 응용프로그램을 동시에 실행시키기 위한 메커니즘을 제공한다. 스케줄러는 실행 가능한 상태의 태스크들 중에서 스케줄링 정책에 맞는 태스크를 선택하여 CPU를 사용할 수 있게 하며, 메모리 매니저는 유한한 메모리 자원을 효율적으로 관리하기 위해서 세그멘테이션이나 페이징 등의 가상 메모리 기법을 사용한다. 본 논문에서는 메모리의 양이 무한하다고 가정을 한다. 따라서 가상 메모리에 대한 고려를 하지 않으며 또한 동적으로 사용되는 스택과 힙 등의 메모리에 대한 고려도 하지 않는다. 그리하여 네트워크 응용프로그램의 QoS를 만족시키기 위해 필요한 적

정자원요구량을 고려할 때, 메모리 자원은 생략한다. 네트워크 응용프로그램은 네트워크를 통해서 데이터를 입출력하기 위해 send(), recv(), read(), write() 등의 블록킹(blocking) 시스템 호출을 사용한다. 시스템의 운영체제는 이러한 요청에 대해서 실제로 데이터를 송신하거나 수신이 이루어질 때까지 태스크를 블록한다. 응용프로그램은 네트워크 패킷 데이터가 보내어지거나 도착할 때까지 운영체제에 의해서 블록되므로 CPU 자원을 할당받지 못한다. 네트워크 패킷수신 이벤트가 발생하게 되면 운영체제는 네트워크 프로토콜 처리과정을 수행한 후, 해당 태스크를 wake up한다. 이어서 깨어난 태스크는 블록킹 시스템 호출 직후의 명령을 수행한다. 따라서 네트워크 프로토콜 처리과정을 제어하면 태스크의 스케줄링을 간접적으로 제어할 수 있게 되므로 태스크의 CPU 시간을 제어할 수 있게 된다. 따라서 네트워크 자원과 CPU 자원은 밀접한 관계가 있다. 본 논문에서는 네트워크 응용프로그램의 QoS를 만족시키기 위해 필요한 적정자원요구량을 고려할 때, CPU 자원은 생략하고 오직 네트워크 자원만으로 표현하기로 한다. 결과적으로 앞서 나타내었던 가정과 문제는 아래와 같이 표현할 수 있다.

[표 5 단순화된 가정과 문제제기]

※가정

$$\begin{cases} Q_{ik} = f_Q(R_{ik}) \\ \quad = f_Q(NET_{ik}) \quad , if\ R_{ik} < R_i^{effective} \\ Q_{ik} = Q_i^{effective} \quad , if\ R_{ik} \geq R_i^{effective} \end{cases}$$

여기서, $f_Q()$는 단조 증가함수이다.

($f_Q()$ is a monotonically increasing function.)

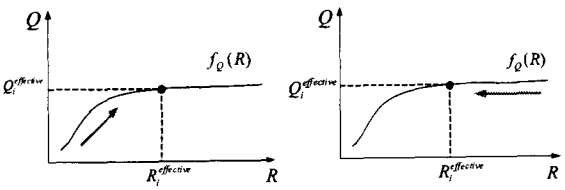
※문제제기

Find $Q_{ik}^{effective}$ for each task τ_i in any time interval k with network resource allocation, and

Find $NET_{ik}^{effective}$ such that $f(NET_{ik}^{effective}) \geq Q_{ik}^{effective}$

3.2.3 최적의 파라미터 검출과 패킷 처리율 제한 기법

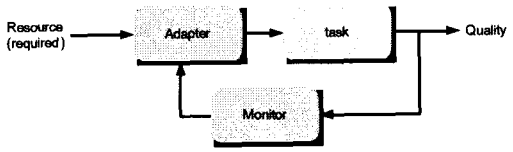
본 논문에서는 적정자원요구량을 찾기 위한 방법으로 측정에 의한 검출방법을 제안한다. 이러한 방법의 장점은 현재 시스템 자원의 범위 내에서 태스크에 대응하는 적정자원요구량을 실질적으로 검출해 내는 것이다. 따라서 운영체제는 실행되는 미지의 태스크에 대한 사전지식이 없이도 검출이 가능하다는 장점이 있다.



<그림 3 (a)추가할당 기법, (b)자원감소 기법>

이를 위해서 태스크에게 (a)할당하는 자원을 점차 늘려가면서 적정자원요구량을 검출하는 방식과 (b)최대자원을 제공 후 잉여자원을 빼앗는 방식을 생각해 볼 수 있다. (b)방식은 최대 자원할당 후, 일정시간의 검출시간이 지난 후에 사용했던 자원의 양을 검출하기 때문에 간단히 구현할 수 있는 특징을 갖는다. 본 논문에서는 <그림 3>의 (b)방식으로 리눅스 커널을 이용해서 구현하였다.

적정자원요구량 검출단계가 완료되면 QoS어댑터는 주기적인 간격으로 자원 재분배를 실시한다. <그림 4>는 모니터링과 피드백을 통한 패킷 처리율 제한 기법을 나타낸다. QoS 모니터는 앞서 등록된 유효QoS 시점의 CPU 시간과 현재 사용하고 있는 CPU 시간을 비교하여 유효QoS가 만족되지 않는다면 하위 우선순위 태스크의 패킷 처리율 제한을 실시하여 현재 우선순위의 유효QoS를 만족시킨다. QoS모니터/어댑터는 커널의 타이머 인터럽트를 이용해서 구현될 수 있다.

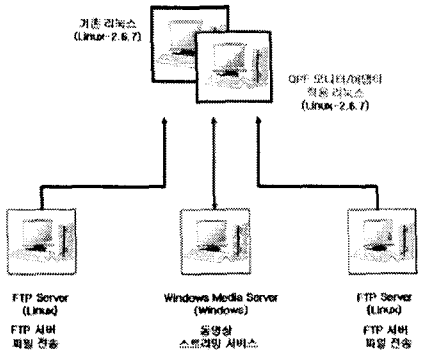


<그림 4 모니터링과 피드백을 통한 패킷제한 기법>

4. 실험 및 평가

본 장에서는 앞장에서 정의하고 가정한 현상을 확인하기 위한 실험을 실시한다. 또한 앞서 기술한 설계 및 목표에 따라 구현된 QoS지원 프레임워크가 적용된 리눅스 시스템과 일반 리눅스 시스템 상에서 다수의 네트워크 응용프로그램을 동시에 실행시켜 봄으로써, QoS보장의 관점에서 두 시스템에 대한 성능비교를 수행한다.

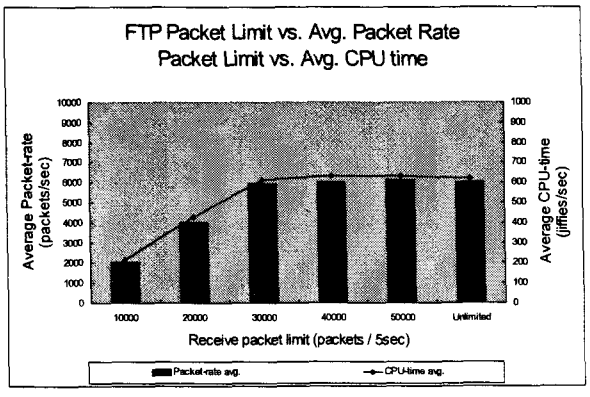
실험을 위해 구축된 시스템과 네트워크 환경은 <그림 5>와 같다.



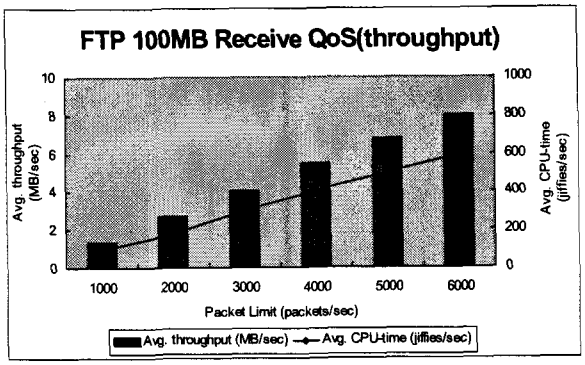
<그림 5 실험환경 설정>

실험은 FTP 파일전송서버 #1, FTP 파일전송서버 #2, 동영상 스트리밍 서버, 그리고 QPF(QoS Provisioning Framework)가 탑재된 클라이언트로 구성된다. 특히 실험에 사용되는 클라이언트에는 네트워크 어댑터 3개가 장착되어 있으며 각각의 서버들과 100Mbps로 1:1 연결되어 있다. 이러한 네트워크 연결방법은 네트워크 패킷들이 중간에 허브나 라우터를 거치지 않게 함으로써 최대의 전송속도로 클라이언트와 직접 그리고 동시에 접속되게 하기 위해서이다. 만약 각 서버들이 허브나 라우터를 통해 클라이언트와 접속된다면, 패킷 충돌에 의한 손실이나 네트워크 장비의 스케줄링에 의해서 각 서버에서 전송되는 패킷들이 클라이언트에 공평하게 전달되지 않는 문제가 발생하며, 각 연결에 대한 전송속도 역시 최대 전송속도의 1/3 수준으로 저하된다.

<그림 6 ~ 그림 9>은 앞서 가정한 유효QoS와 적정자원요구량이 존재함을 나타낸다. <그림 6>은 패킷 처리율 제한을 점차 완화시켜 가면서 FTP 응용프로그램의 성능을 측정한 것이다. FTP 응용프로그램의 패킷전송률과 CPU시간은 점차 증가하다가 NIC 전송속도의 한계에 도달하여 수렴하는 현상이 나타난다. <그림 7>은 FTP의 실제 데이터 전송률 변화를 나타내며, 적정자원 요구량 이하의 구간에서 $f_Q()$가 단조 증가하는 것을 알 수 있다.

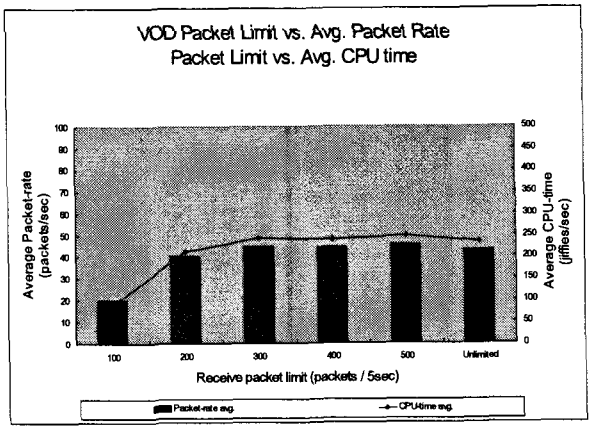


<그림 6 패킷 처리율 제한 완화에 따른 FTP 성능변화>

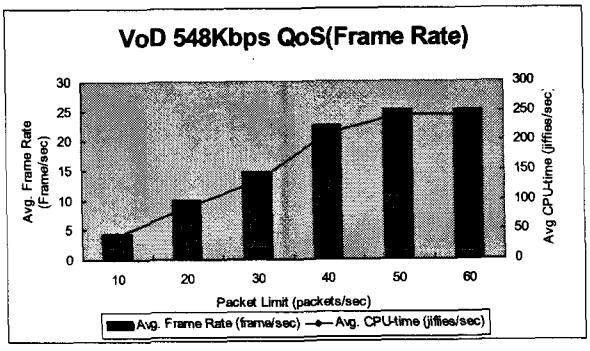


<그림 7 적정자원요구량 이하에서 $f_Q()$>

<그림 8>은 패킷 처리율 제한을 점차 완화시켜 가면서 VoD 응용프로그램의 성능을 측정한 것이다. VoD 응용프로그램의 패킷 전송률과 CPU시간은 역시 점차 증가하다가 재생하는 동영상의 규격에 도달하여 수렴하는 현상이 나타난다. <그림 9>는 VoD의 실제 프레임레이트 변화를 나타내며, 역시 적정자원요구량 이하의 구간에서 $f_Q()$가 단조 증가하는 것을 알 수 있다.

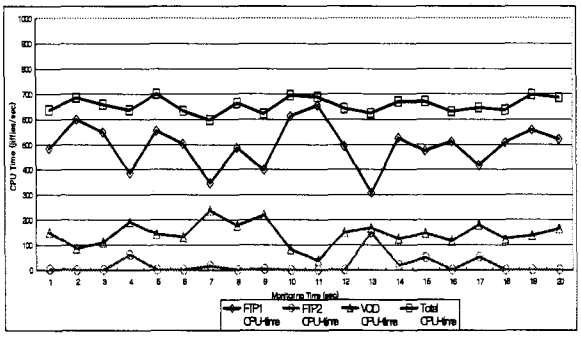


<그림 8 패킷 처리율 제한 완화에 따른 VoD 성능변화>



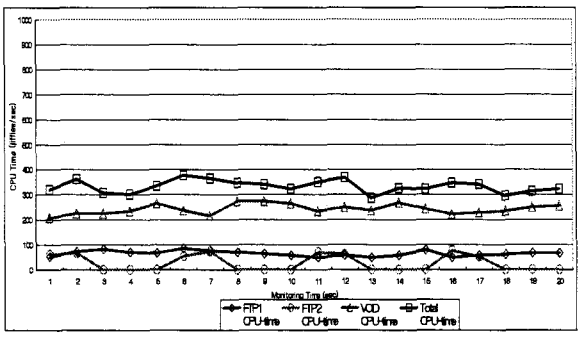
<그림 9 적정자원요구량 이하에서 $f_Q()$>

<그림 10>은 일반 리눅스 시스템과 제안하는 프로토콜 처리를 수행하는 리눅스 시스템의 성능차이를 알아보았다. FTP 응용프로그램 2개와 VoD응용 프로그램을 동시에 실행시키고 VoD의 우선순위를 가장 높게 설정하였다.



<그림 10 일반 리눅스 시스템에서 수행>

VoD의 우선순위가 가장 높지만 앞서 측정한 적정자원요구량인 250 (jiffies/sec)를 확보하지 못하므로 유효QoS를 만족시키지 못한다.



<그림 11 제안하는 프로토콜 처리 시스템에서 수행>

<그림 11>에서는 VoD의 우선순위가 가장 높으므로 나머지 응용프로그램의 프로토콜 처리에 패킷 처리율 제한을 수행하여, 적정자원요구량과 유효QoS를 만족시키는 것을 보여준다. 그러나 시스템의 전체적인 CPU utilization이 감소하였다. 이는 네트워크 응용프로그램의 QoS가 다중 요소들으로 구성되고, 상호간에 연관이 있기 때문이다.

5. 결론 및 향후과제

본 논문에서는 네트워크 응용프로그램의 최적의 성능(QoS)을 보장하기 위해서, 패킷 처리율 제한을 통한 우선순위 기반의 프로토콜 처리기법을 제안하였다. 본문에서는 시스템 자원할당을 통한 QoS 보장기법을 제안하기 위해서 QoS 모델을 정의하고 가정을 기술하였다. 이를 통해서 문제를 명확하게 제시하였으며, 해결방안으로 최적의 파라미터 검출과 패킷 처리율 제한기법을 제안하였다. 또한 실험을 통하여, 가용한 자원의 양이 제한된 시스템에서 동작하는 다수의 네트워크 응용프로그램들 중에서 높은 우선순위 응용프로그램의 QoS가 보장됨을 보였다.

그러나 제안하는 시스템에서는 전체적인 CPU utilization이 감소하는 문제가 발견되었다. 따라서 향후에는 개별 응용프로그램의 QoS를 만족시키면서, 전체적인 시스템 자원의 이용률을 증가시키는 방안에 대해서 연구를 진행하고자 한다.

참고 문헌

[1] R. Rajkumar and C. Lee and J. Lehoczky and D. Siewiorek, "A Resource Allocation Model for QoS Management," *IEEE Real-Time Systems Symposium p.298-p.307 Dec, 1997.*

[2] Christian Poellabauer, Hasan Abbasi, Karsten Schwan, "Q-Fabric: System Support for Continuous Online Quality Management", *Proceedings of the tenth ACM international conference on Multimedia, 2002.*

[3] Y. Zhang and R. West. "Process-Aware Interrupt Scheduling and Accounting," *In Proceedings of the 27th IEEE International Real-Time Systems Symposium (RTSS'06), December 2006*

[4] Jaehee Byun, Dongsoo Kim, Minsoo Ryu, Choul-Soo Jang, and Joong-Bae Kim, "Priority-Driven Network Protocol Processing for OS-Level QoS Support," *The Second International Conference on Ubiquitous Robots and Ambient Intelligence, Daejon, Korea, November, 2005*

[5] 변재희, "Bottom-half 기반 실시간 네트워크 I/O 기법," 한양대학교 석사학위 논문, 2006.

[6] G. Banga and P. Druschel, "Lazy Receiver Processing (LRP): A Network Subsystem Architecture for Server Systems," *Proceedings of USENIX Symposium on Operating System Design and Implementation, 1996.*

[7] Linux, available at http://www.kernel.org/