운영체제 수준에서 QoS를 보장하기 위한 우선순위 기반 네트워크 프로토콜 처리*

김동수, 변재희, 유민수
한양대학교 정보통신공학과
e-mail:{dskim,jhbuyn,msryu}@rtcc.hanyang.ac.kr

Priority-Based Network Protocol Processing for OS-Level QoS Provisioning

Dongsoo Kim, Jaehee Byun, Minsoo Ryu
Graduate School of Information and Communications,
Hanyang University

요약

본 논문에서는 운영체제 측면에서 QoS를 보장하기 위해 우선순위 기반의 네트워크 프로토콜 처리 기법을 제안한다. 제안된 기법에서는 우선순위에 따라 네트워크 패킷을 분류하고 프로토콜을 처리한다. 이를 위해 패킷분류기(Packet Classifier)와 프로토콜엔진(Protocol Engine)을 포함하는 QPF(QoS Provisioning Framework)를 설계하고 리눅스 커널 내부에 구현하였다. 과거의 인터럽트 기반의 방식에서는 네트워크 패킷이 선착순(first-in first-out)으로 처리되어 응용 프로그램에서 요구하는 QoS를 보장하기 어려우며, 또한 항상 네트워크 패킷의 처리가 응용 프로그램보다 우선적으로 처리되어 수신교착상태(Receive Livelock) 등의 문제가 발생한다. 본 논문에서 제안하는 QPF는 네트워크 패킷을 우선순위에 따라 처리함은 물론 네트워크 처리에 사용되는 CPU 시간을 조절할 수 있어 위와 같은 문제를 효과적으로 해결할 수 있다.

1. 서론

최근 네트워크에 기반한 응용 프로그램들이 많이 등장하면서 네트워크 데이터의 전송과 처리에 있어서 QoS(quality of service)를 보장하는 것이 중요한 연구과제가 되었다. 예를 들면, 실시간 멀티미디어 스트리밍에서 QoS를 보장하기 위해서는 스트리밍 데이터를 실시간으로 전송하고 QoS를 고려하여 네트워크 프로토콜을 처리하는 것이 요구된다. 하지만 최근까지 이 분야에 대한 연구를 살펴보면, 네트워크 측면에서 데이터 전송의 QoS 보장을 위한 연구는 활발하게 진행 되어온 반면[9, 10], 종단에 위치한 운영체제 측면에서 네트워크 프로토콜 처리의 QoS 보장에 대한 연구는 상대적으로 부족한 상황이다.

현재 사용되고 있는 대다수의 운영체제는 QoS를 보장하기에 부적합한 네트워크 처리 정책과 방식을 제공한다. 통상적으로 인터럽트에 의한 네트워크 처리방식이 사용되고 있는데, 이는 두 가지 형태의 우선순위역전(priority inversion)을 초래한다. 첫째, 인터럽트 핸들러가 네트워크 패킷을 선착순(first-in first-out)으로 처리함에 따라 우선순위가 낮은 패킷이 우선순위가 높은 패킷보다 먼저 처리될 수 있다. 여기서 패킷의 우선순위는 패킷이 전달될 응용 프로세스의 우선순위를 의미한다. 둘째, 우선순위가 낮은 패킷이 우선순위가 높은 응용 프로세스보다 먼저 처리될 수 있다. 이 경우 우선순위역전은 물론 대량의 패킷이 동시에 도착하면 우선순위가 높은 응용 프로세스들이 전혀 수행되지 못하는 수신교착상태(receive livelock)가 발생할 수도 있다.

본 논문에서는 위와 같은 문제를 해결하기 위해 운영체제 수준에서 우선순위에 따라 네트워크 패킷

* 본 논문의 연구는 한국전자통신연구원과 정보통신부의 선도기반구축사업(0401-2004-0013)의 지원을 받았다.

을 처리하는 QPF(QoS provisioning framework) 방식을 설계하고 구현하였다. QPF는 패킷분류기(packet classifier)와 프로토콜엔진(protocol engine)으로 구성된다. 패킷분류기는 인터럽트 컨텍스트에서 수행되어 시스템에 도착한 패킷을 우선순위에 따라 분류한다. 분류된 패킷들은 우선순위 별로 프로토콜 처리를 전담하는 커널 쓰레드들에 의해 처리된다. 프로토콜엔진은 프로토콜 처리를 전담하는 커널 쓰레드들을 생성, 소거, 스케쥴링하는 역할을 담당한다. 본 연구에서는 QPF의 효용성을 입증하기 위해 리눅스상에 구현하여 우선순위에 따라 QoS를 보장하는 것이 가능하도록 하였다.

2장에서는 QoS를 고려한 프로토콜 처리 방식인 QPF에 대해 기술한다. 3장에서는 실제 리눅스에서 QPF를 구현하는 방법을 기술한다. 4장에서는 결론 및 향후 연구 과제를 기술한다.

2. QPF(QoS Provisioning Framework)의 설계

본 절에서는 QPF의 구조와 우선순위 기반의 패킷처리를 위한 내부 동작을 설명한다. 아울러, 네트워크 패킷의 처리에 의해 CPU가 독점되는 현상을 방지하기 위한 기법도 함께 소개한다.

2.1 QPF(QoS Provisioning Framework)의 구조와 동작

QPF는 그림 1에서 짙은 색의 사각형으로 표현된 패킷분류기(packet classifier)와 프로토콜엔진(protocol engine)으로 구성된다. 그림 1에서 붉은 색의 화살표는 패킷의 데이터흐름(data-flow)을 나타내며, 푸른 색의 화살표는 제어흐름(control-flow)을 나타낸다.

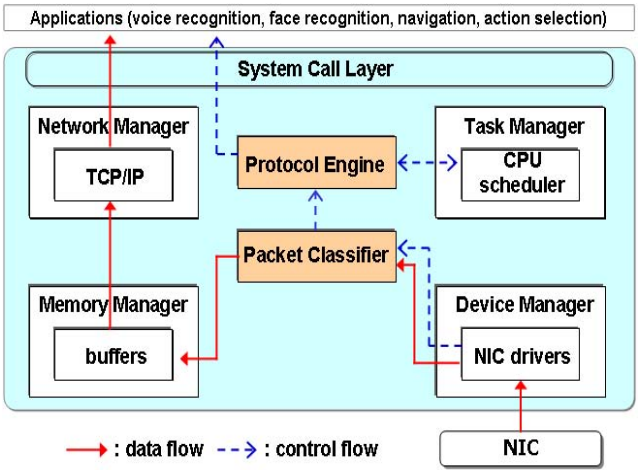

그림 1. QPF(QoS Provisioning Framework)

네트워크 패킷이 도착하면 인터럽트 핸들러가 수행되는데, 이 때 패킷분류기가 실행되어 네트워크 세션별로 패킷을 분류하고 인터럽트 핸들러로부터 복귀한다. 이 후에 프로토콜엔진과 CPU 스케쥴러에 의해 분류된 패킷의 프로토콜이 처리된다.

프로토콜엔진(protocol engine): 네트워크 세션별로 PUQ(protocol unprocessed queue) 및 PPQ(protocol processed queue)를 생성하고, 동일한 네트워크 세션의 프로토콜 처리를 담당하는 커널 쓰레드를 생성하고 할당한다. PUQ는 시스템에 도착한 패킷이 최초로 분류되어 삽입되는 큐로서 프로토콜 처리를 기다리는 패킷을 저장하기 위한 자료구조이다. 반면, PPQ는 커널 쓰레드에 의해 프로토콜 처리가 완료된 패킷을 저장하기 위한 큐이다. 프로토콜엔진에서는 이러한 PPQ와 PUQ를 생성하고, 아울러 프로토콜 처리를 담당하는 커널 쓰레드를 생성하고 CPU 스케쥴러에 의해 스케쥴링될 수 있도록 한다. 네트워크 세션이 종료되면, 프로토콜엔진은 생성하였던 PUQ, PPQ, 그리고 커널 쓰레드들을 소멸시킨다.

프로토콜 처리 쓰레드(protocol processing thread): 프로토콜엔진이 생성하는 프로토콜 처리 쓰레드는 실행가능(ready)과 실행대기(waiting)의 두 상태를 가진다. 기본적으로 자신이 담당하는 PUQ에 패킷이 존재하면 실행가능 상태가 되며, PUQ가 비워지면 실행대기 상태가 된다. 또한 프로토콜 처리 쓰레드는 우선순위를 가지고 CPU 스케쥴러에 의해 스케쥴링 되는데, 기본적으로 네트워크 세션을 생성한 응용 프로세스의 우선순위를 상속받는다. 필요에 따라 응용 프로세스가 프로토콜 처리 쓰레드의 우선순위를 조절하는 것도 가능하다. 따라서 모든 네트워크 패킷은 응용 프로세스에 의해 결정되는 우선순위에 따라 프로토콜이 처리된다.

패킷분류기(packet classifier): 도착한 각 패킷의 헤더로부터 포트(port)번호를 추출하여 해당 네트워크 세션의 PUQ에 삽입하고, 해당 커널 쓰레드를 실행 가능한 상태가 되도록 깨운다(wake up).

2.2 CPU 자원의 효율적 분배

위에서 설명한 QPF는 우선순위를 고려하여 QoS를 보장할 수 있는 장점을 제공하는 반면, 일반적인

우선순위 기반의 스케쥴링 방법에서는 우선순위가 높은 쓰레드에 의해 CPU가 독점되는 단점을 가진다. 예를 들면, 특정 프로토콜 처리 쓰레드의 우선순위가 제일 높고 해당 네트워크 패킷이 대량으로 연속적으로 도착하면 우선순위가 낮은 다른 프로토콜 처리 쓰레드 및 응용 프로세스들은 CPU를 할당받지 못하여 전혀 수행되지 않는 현상이 발생할 수 있다. 일반적으로 네트워크의 트래픽은 대량으로(bursty) 발생하는 특징을 가지기 때문에 위와 같은 CPU 독점 현상을 방지하는 것이 중요하다.

QPF는 특정 프로토콜 처리 쓰레드에 의한 CPU 독점을 방지하기 위해 DS(Deferrable Server) [6] 방식의 스케쥴링 기법을 사용한다. DS 방식을 사용하면 임의의 태스크는 일정한 주기(period) 내에서 고정된 크기의 CPU 시간만을 사용할 수 있다. 만약 주어진 CPU 시간 안에 처리해야 될 일을 완료하지 못하면, 그 다음 주기(period)에서 주어지는 CPU 시간을 사용하여 처리하여야 한다.

QPF에서는 모든 프로토콜 처리 쓰레드 P_i에 대해 주기(period) T_i와 수행허용시간(execution time budget) E_i를 정의하고, DS 방식에 따라 프로토콜 처리 쓰레드의 실행상태를 조절한다. 앞서 2.1에서는 프로토콜 처리 쓰레드가 자신이 담당하는 PUQ의 상태에 따라 실행가능(ready)과 실행대기(waiting)의 상태를 가질 수 있음을 이미 기술한 바 있다. 이 때 DS 방식을 함께 고려하면, PUQ에 패킷이 존재하고 수행허용시간(execution time budget)이 남아있는 경우에만 프로토콜 처리 쓰레드가 실행가능 상태가 되고, 다른 모든 경우에는 실행대기 상태로 남아 있는다.

3. 리눅스 내부에서의 QPF 구현

앞서 제안된 QPF의 효용성을 입증하기 위해 QPF를 리눅스 커널 내부에 구현하였다. QPF는 프로토콜엔진과 패킷분류기의 두 개 컴포넌트로 설계되었지만, 리눅스에서는 두 컴포넌트를 하나의 네트워크 디바이스 드라이버 모듈로서 구현하였다.

본 연구에서 QPF를 구현한 HW/SW 환경은 표 1과 같다.

3.1 프로토콜엔진(Protocol Engine)의 구현

PUQ 생성: 모든 네트워크 세션마다 qpf_puq_create( )를 호출하여 그림 2와 같이 정의되는 PUQ 를 생성한다. PUQ는 해당 네트워크 세션의 포트번호(qpf_port)를 포함하며, 최대 1024개의 패킷을 동시에 저장할 수 있는 순환버퍼(cyclic buffer)로서 리눅스 sk_buff의 배열(qpf_skb[1024])로 정의되어 있다. 또한 패킷분류기가 패킷을 PUQ에 삽입하기 위한 위치를 지시하기 위하여 in_ptr를 사용하고, 프로토콜 처리 쓰레드가 패킷을 PUQ로부터 꺼내기 위하여 out_ptr를 사용한다.

구 분	사 양	제 조 사
프로세서	Pentium4 2.4(A) GHz	인텔
메인 메모리	DDR SDRAM 1GZ	삼성전자
캐쉬 메모리	512 KB (On Die)	인텔
HDD	60 GB, 7200 rpm	삼성전자
Ethernet	10/100 RTL-8139 PCI	RealTek
O S	Linux (kernel-2.6.7)	
S/W	Linux 기본 프로그램	

표 1. QPF 구현 환경

```
typedef struct qpf_puq {
        int qpf_port;                       // line 1
        int in_ptr;                         // line 2
        int out_ptr;                        // line 3
        struct sk_buff *qpf_skb[1024];      // line 4
} qpf_t;
```

그림 2. PUQ의 자료구조

프로토콜 처리 쓰레드의 생성 및 처리: QPF는 qpf_thread_create( )를 제공하여 프로토콜 처리를 담당하는 리눅스 커널 쓰레드를 생성하도록 한다. qpf_thread_create( )에서는 리눅스의 kernel_thread( )를 호출하여 커널 쓰레드를 생성하고, 별도의 지정이 없으면 응용 프로세스의 우선순위(static_prio)를 상속받는다. 이 후에 응용 프로세스가 종료되면 qpf_thread_destroy( ) 함수를 통해 해당 프로토콜 처리 쓰레드가 삭제된다.

프로토콜 처리 쓰레드의 동작: 프로토콜 처리 쓰레드는 qpf_puq_process( ) 함수를 수행한다. qpf_puq_process( ) 함수에서는 그림 2에서 정의된 PUQ의 sk_buff의 배열(qpf_skb[1024])을 순환하면서 저장되어 있는 skb를 추출하고, 프로토콜 스택 처리를 위해 netif_receive_skb( ) 함수를 호출한다. 만약 PUQ에 처리할 패킷이 없으면 프로토콜 처리 쓰레드는

실행대기 상태로 바뀐다.

3.2 패킷분류기(Packet Classifier)의 구현

패킷분류기는 qpf_classifier(struct sk_buff *skb) 함수로 구현되었으며, 이 함수는 인터럽트 핸들러 내부에서 호출되어 도착한 패킷을 해당 PUQ에 삽입하는 역할을 담당한다. 패킷분류기 함수 qpf_classifier( )가 호출되는 과정은 다음과 같다. 네트워크 디바이스 드라이버에 패킷이 수신되면 디바이스 의존적인 인터럽트 핸들러 xxx_interrupt( )* 가 호출되고, 패킷을 소켓버퍼(sk_buff)로 복사하기 위해 xxx_rx( )† 가 호출된다. 이 때 패킷분류기 qpf_classifier( )가 호출되어 파라미터로 넘겨받은 소켓버퍼로부터 해당 네트워크 세션의 포트정보(skb->h.tcphdr->dport)를 추출한 후, 동일한 포트번호를 갖는 PUQ를 찾아 그림 2에서 선언된 in_ptr를 이용하여 패킷을 삽입한다.

그림 4는 제안하는 QPF를 실제 리눅스상에 구현하고, 패킷이 수신될 때의 데이터 플로우(flow)를 도식화한 것이다.

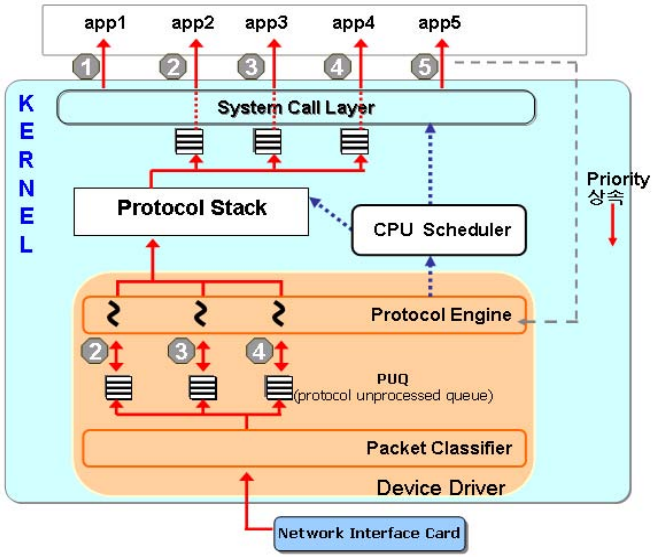

그림 4. QPF에서의 수신 Flow

4. 결론 및 향후 연구 과제

본 논문에서는 운영체제 측면에서 QoS를 보장하기 위해 우선순위 기반의 네트워크 처리 방식인 QPF를 설계하였고, 그 효용성을 입증하기 위해 리눅스상에 구현하여 우선순위에 따라 QoS를 보장하는 것이 가능하도록 하였다. 향후에는 QPF가 동적으로 QoS를 모니터링하고 적응적으로 QoS를 조절할 수 있는 기능을 추가하고자 한다.

* RealTek NIC의 rtl8139_interrupt( ) 함수
† RealTek NIC의 rtl8139_rx( ) 함수

참고문헌

[1] Chris Maeda and Brian N Bershad, "Protocol service decomposition for hign-performance networking," *ACM Symposium on Operating Systems Principles,* 1994

[2] Chandramohan A. Thekkath, Thu D. Nguyen, Evelyn Moy, and Edward D. Lazowska, "Implementing network protocols at user level," *Applications, Technologies, Architectures,and Protocols for Computer Communication*, 1993.

[3] 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.

[4] José Brustoloni, Eran Gabber, Abraham Silberschatz, and Amit Singh, "Signaled Receiver Processing," *USENIX* nd Performance of a Real-time I/O Subsystem," *Proceedings of the Fifth IEEE Real-Time Technology and Applications Symposium table of contents*, 1999.

[5] Fred Kuhns, Douglas C. Schmidt, and David L. Levine, "The Design and Performance of a Real-time I/O Subsystem," *Proceedings of the Fifth IEEE Real-Time Technology and Applications Symposium table of contents*, 1999.

[6] J. K. Strosnider, J. P. Lehoczky and L. Sha, "The Deferrable Server Algorithm for Enhanced Aperiodic Responsiveness in Hard Real-Time Environments", *IEEE Transactions on Computers*, January 1995.

[7] B. Sprunt, L. Sha, and J. P. Lehoczky, "Aperiodic Task Scheduling for Hard-Real-Time Systems," *The Journal of Real-Time Systems*, 1989.

[8] Brinkley Sprunt, John Lehoczky, and Lui Sha, "Exploiting Unused Periodic Time
for Aperiodic Service Using The Extended Priority Exchange Algorithm," *IEEE Real-Time Systems Symposium*, December 1988.

[9] R. Braden, D. Clark, and S. Shenker, "Integrated Services in the Internet Architecture: an Overview," *Internet RFC 1633*, June 1994.

[10] R. Braden et al., "Resource ReSerVation Protocol (RSVP) Version 1 Functional Specification," *Internet RFC 2205*, Sept. 1997.