碩士學位論文

차등 서비스를 위한
네트워크 스케줄러 설계 및 구현

2007 年

漢城大學校 一般大學院

컴퓨터 工學科

컴퓨터 工學 專攻

尹 燦 熙

碩 士 學 位 論 文
指導敎授 李 珉 秔

차등 서비스를 위한 네트워크 스케줄러 설계 및 구현

Design and Implementation of Network Scheduler for Differentiated Service

2006年 12月 日

漢城大學校 一般大學院

컴퓨터 工學科

컴퓨터 工學 專攻

尹 燦 熙

碩 士 學 位 論 文
指導敎授 李 珉 秔

차등 서비스를 위한 네트워크 스케줄러 설계 및 구현

Design and Implementation of Network Scheduler for Differentiated Service

위 論文을 컴퓨터工學 碩士學位論文으로 提出함

2006年 12月 日

漢城大學校 一般大學院

컴퓨터 工學科

컴퓨터 工學 專攻

尹 燦 熙

목 차

표 목차

그림 목차

尹燦熙의 工學 碩士學位論文을 인정함

2006年 12月 日

심사위원장 정 인 환 (인)

심사위원 황 기 태 (인)

심사위원 이 민 석 (인)

요 약

많은 인터넷 웹 서비스가 제공됨에 따라 고객들은 고품질의 서비스를 받기 위해 일정 비용을 지불하고 있다. 또한 기업은 다른 경쟁 기업에 비해 더 나은 서비스를 제공하여 많은 이익을 창출하려 할 것이다. 현재 제공되고 있는 서비스들은 고객들이 지불한 비용에 따라 고객을 분류하고, 분류된 고객에게 서로 다른 서비스를 제공한다. 하지만 이러한 서비스 제공에는 기능의 차이를 보일뿐 네트워크 응답 속도에 따른 서비스 차이를 두지 않는다.

본 논문에서는 리눅스 환경에서 Percentile 스케줄 기법을 이용하여 높은 등급의 고객에게 빠른 서비스를 제공하는 네트워크 스케줄링 방법을 제안한다. Percentile 스케줄 기법은 각 등급에 따른 우선순위를 결정하는 기법으로 Percentile 값과 클래스로 분류된 패킷의 수 가운데 마감시간 전에 처리된 패킷 수의 비율과 미리 정한 percentile 비율의 차이가 클수록 높은 우선순위를 부여한다. 높은 우선순위를 가지는 패킷은 다른 패킷보다 먼저 서버 프로그램으로 전달되어 처리되므로 빠른 응답 시간을 가지게 된다.

Percentile 네트워크 스케줄러의 성능을 평가하기 위해 널리 사용되는 웹 서버인 아파치를 사용하여 제안된 시스템과 기존 리눅스의 응답속도를 비교하였다. 실험을 통해 기존 리눅스 네트워크 스케줄러는 등급에 따른 차등 서비스가 제공이 되지 않음을 확인할 수 있었으나, 제안된 Percentile 네트워크 스케줄러는 기존 리눅스 네트워크 스케줄러에 비해 등급에 따른 차등 서비스를 제공 할 수 있음을 관찰할 수 있었다.

제 1장 서론

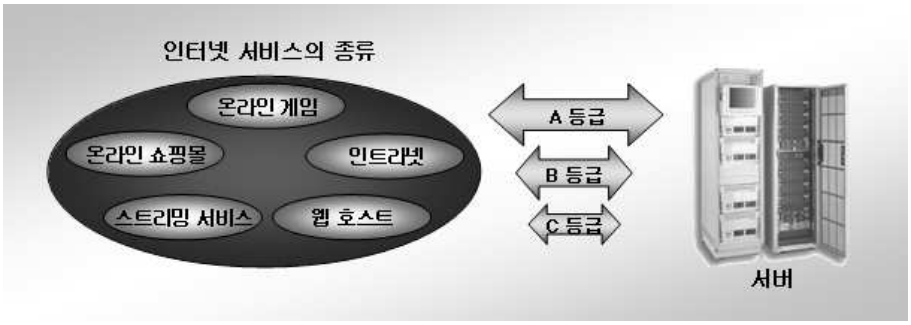
1.1 연구배경

IT와 초고속 네트워크 발전으로 웹을 이용한 여러 서비스의 수요가 증대되고 있다. 이러한 수요에 맞추어 많은 서비스들이 등장하였으며, 고객들은 양질의 서비스를 이용할 수 있게 되었다. 최근 들어 웹 호스팅이나 온라인 게임 서비스, 물품 구매를 위한 온라인 쇼핑몰, 스트리밍을 위한 VOD 서비스 등의 서비스가 많이 제공되고 있으며, 많은 고객들이 서비스 비용을 지불하고 이를 이용하고 있다.

기업은 고객을 서비스 사용 시간, 서비스 사용 비용에 따라 분류하고 분류된 고객에게 서로 다른 서비스를 제공한다. 하지만 이러한 서비스 제공에는 웹 호스팅 저장 공간, 온라인 게임 아이템, 구매 금액 할인과 같은 기능의 차이를 보일뿐 실제 네트워크 응답 속도에 따른 서비스 차이를 두지 않는다. 서버 호스팅의 경우 서로 다른 비용을 지불한 고객에 대하여 같은 수준의 서비스를 제공하는 것은 공정하지 못하다. 많은 비용을 지불한 고객의 홈페이지에 대한 참조는 적은 비용을 지불한 고객의 홈페이지보다 더 짧은 응답시간을 제공함으로써 차별화된 서비스를 할 수 있게 된다. 전자 상거래 사이트의 경우, 단순히 살펴보기 위하여 방문한 고객과 자주 물건을 구입한 고객에 대하여 다른 품질의 서비스를 제공함으로써 더 많은 구매를 유도할 수 있다.

1.2 연구 목적 및 방법

기존의 여러 인터넷 서비스는 고객을 대상으로 차등 서비스가 제공되고 있지 않다. 따라서 지불한 금액, 사용 시간, 계급 등을 기준으로 고객을 등급화 하고 각 고객의 요청에 대하여 차별 서비스를 제공한다.



[그림 1] 인터넷 서비스 종류에 따른 차등 서비스

[그림 1]은 인터넷 서비스의 종류와 각 등급에 따라 서버에서 어떻게 대응해야 할지 보여주는 그림이다. 온라인 게임, 온라인 쇼핑몰, 인트라넷 등의 서비스는 각 서비스별로 고객의 등급을 나누는 기준을 통하여 고객을 등급에 따라 분류하고 서버는 각 고객의 요청을 각 등급에 맞게 처리한다. 만일 온라인 게임에서 많은 비용을 지불한 고객에게는 적은 비용을 지불한 고객에 비하여 게임 내에서 빠른 응답 시간을 제공 받을 것이고, 인트라넷에서는 높은 직급 또는 중요 부서 사원에게는 빠른 일처리를 위한 빠른 응답 시간을 제공할 것이다. 이는 네트워크 스케줄링이라는 기법을 이용하여 각 등급에 따른 요청을 제어함으로써 차등 서비스를 제공한다.

서비스의 응답속도 향상을 위하여 스케줄링 하는 방법에 대해서는 그동안 많은 연구가 이루어져 왔다. 응답속도 향상을 위한 대표적인 연구인

SRPT-CS 스케줄링 기법[1]은 요청 파일의 용량을 기준으로 우선 처리하는 방법으로 전체 응답시간을 향상 시킬 수 있지만 차등 서비스를 하기 위한 방법으로는 적합하지 않다. 또한 대부분의 스케줄링 연구는 요청이 처리된 패킷을 클라이언트로 전송할시 우선순위를 이용한 스케줄링을 통해 높은 우선순위를 지닌 패킷을 먼저 전송하는 연구이다[2][3][4]. 하지만 각 우선순위는 서비스별로 설정되어 하나의 서비스 내에서 클라이언트에 따른 차등 서비스가 이루어지지 않기 때문에 차등 서비스에는 적당하지 않다. 따라서 본 논문에서는 차등 서비스를 위하여 패킷이 응용 프로그램까지 전송되는 단계에서 패킷의 등급을 결정하고, Percentile 기법[5]을 이용한 등급별 우선순위에 따른 스케줄링 방법을 제안하고자 한다.

1.3 논문의 구성

본 논문은 5장으로 구성되어 있으며, 1장에서는 서론, 2장에서는 관련 연구, 3장에서는 제안방법을 세부적으로 기술하였고, 4장에서는 실험 환경 및 실험 방법에 대해 기술하기로 하며, 5장에서는 결론 및 향후 연구과제에 대해서 논의한 후 참고문헌을 밝히고 논문을 마치고자 한다.

제 2장 관련연구

본 장에서는 기존 리눅스 네트워크 처리와 IP 라우터에서 사용되는 우선순위에 따른 대표적인 스케줄링 기법에 대하여 살펴보고, 응답시간 향상을 위한 스케줄링 방법에 대해서 알아본다. 또한 각 스케줄링의 장점과 단점을 살펴보고 Percentile 기법을 이용한 기존 연구를 살펴본다.

2.1 리눅스 커널 네트워크 처리

본 논문은 리눅스 커널에서 연구가 이루어졌기 때문에 기존 리눅스 커널의 네트워크 처리과정을 확인할 필요가 있다.

2.1.1 리눅스 커널 인터럽트

커널은 외부 장치로부터 신호를 받아 처리하기 위한 인터럽트 처리 루틴을 가지고 있다[6]. 일반적으로 인터럽트(interrupt)는 프로세서가 실행하는 명령어의 순서를 바꾸는 사건이라 정의한다. 이런 사건은 CPU 내부와 외부에서 하드웨어적인 회로가 발생시키는 전기적인 시그널에 해당한다.

인터럽트 처리 방법은 인터럽트의 유형에 따라 달라지며, 크게 세 가지 유형으로 구분한다. 인터럽트의 세 가지 유형은 아래와 같다.

(1) 입출력 인터럽트(I/O Interrupt)
(2) 타이머 인터럽트(Timer Interrupt)
(3) 프로세스 간 인터럽트(Interprocessor Interrupt)

일반적으로 입출력 인터럽트는 주변장치로 부터의 인터럽트를 의미한다. 예를 들면 키보드나 마우스 또는 PCI 장치들로 부터의 인터럽트 신호가 이에 해당한다. 타이머 인터럽트는 매 시간 발생하는 장치 인터럽트로서 매 시간 동작을 필요로 하는 루틴들이 실행된다. 프로세스 간 인터럽트는 멀티프로세서 시스템에서 하나의 CPU가 다른 CPU에게 보내는 인터럽트이다. 이 세 가지 유형의 인터럽트를 하드웨어 인터럽트라고 한다.

하드웨어 인터럽트가 발생할 때 해야 하는 모든 작업이 동일하게 급하지 않다. 인터럽트 핸들러가 실행 중일 때는 동일한 인터럽트의 신호를 무시하므로 시간이 오래 걸리면서도 중요하지 않은 작업은 나중으로 미루어야 한다. 리눅스는 인터럽트가 발생할 때 처리할 작업을 세 부류로 분류한다.

(1) 중요함
(2) 중요하지 않음
(3) 중요하지 않으며 미룰 수 있음

하드웨어 인터럽트 처리에 있어 커널이 수행하는 작업 중 일부는 시급하지 않다. 이 작업들은 필요하다면 한참 후로 미룰 수도 있다. 인터럽트 핸들러는 인터럽트 서비스 루틴을 직렬로 실행하고, 인터럽트가 수행중에는 해당 인터럽트가 발생하지 않는다. 반대로 미룰 수 있는 작업은 모든 인터럽트를 허용한 상태에서 실행할 수 있다. 따라서 하드웨어 인터럽트 서비스 루틴을 실행할시 오래 걸리는 작업을 수행하게 될 경우 다른 인터럽트의 발생을 막을 수 있어 오랜 시간이 필요한 작업은 미룰 수 있는 작업으로 수행되게 된다. 이것은 커널이 하드웨어 인터럽트에 반응하는 시간을 줄이는데 도움이 된다. 리눅스 2.6은 소프트웨어 인터럽트를 사용

하는 것에 의해 그러한 문제를 처리하는데, 이것은 소프트 IRQ 및 소작업과 작업큐에 의해 처리 된다.

리눅스 2.6에서는 제한된 수의 소프트 IRQ를 사용한다. [표 1]은 리눅스 2.6에서 사용하는 소프트 IRQ이다.

표 1 소프트웨어 인터럽트 우선순위

소프트 IRQ	인덱스(우선순위)	설명
HI_SOFTIRQ	0	우선순위가 높은 소작업을 처리
TIMER_SOFTIRQ	1	타이머 인터럽트와 관련된 소작업
NET_TX_SOFTIRQ	2	네트워크 카드를 통해 패킷 전송
NET_RX_SOFTIRQ	3	네트워크 카드에서 패킷을 수신
SCSI_SOFTIRQ	4	SCSI명령의 인터럽트 후 처리
TASKLET_SOFTIRQ	5	소작업 처리

소프트 IRQ의 인덱스는 우선순위를 나타내며, 낮은 인덱스 값을 가지는 소프트 IRQ는 높은 우선순위를 가짐을 의미한다.

2.1.2 네트워크 패킷 수신 처리

커널은 언제 패킷이 네트워크 카드로 도착할지 예상하지 못한다. 그러므로 패킷의 수신을 다루는 네트워킹 코드는 인터럽트 핸들러와 미룰 수 있는 함수에서 동작한다[7]. 네트워크 카드로 패킷이 수신될 때 네트워크 카드는 이를 커널에게 즉시 처리 해달라는 신호를 보낸다. 만일 커널이 빠른 시간 내에 네트워크 카드에 수신된 패킷을 처리해주지 않으면 네트워크 카드 내에 제한된 메모리에 의해 패킷의 손실이 발생할 수 있다. 따라서 네트워크 카드의 패킷 수신에 대한 처리는 인터럽트 분류 중 중요한 처리에 속하게 된다. 반면에 커널 내로 수신된 패킷 데이터가 전달된 후,

수신된 패킷이 해당 프로토콜 처리를 거쳐 응용 프로그램까지 전달되는 과정은 이미 커널 내의 메모리에 데이터가 존재하고, 오래 걸리는 작업이기 때문에 미룰 수 있는 함수에 의하여 프로토콜 처리를 거쳐 응용 프로그램까지 전달이 된다.

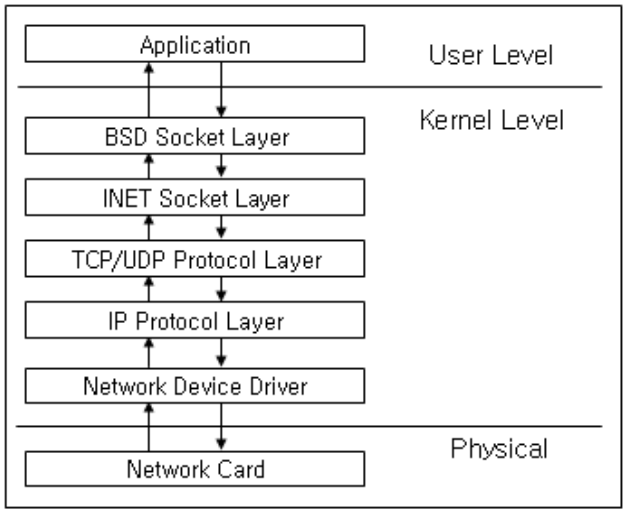
인터럽트 핸들러는 네트워크 장치에 따라 다르지만, 송수신에 대한 인터럽트 처리 루틴은 반드시 존재한다. 이 함수들은 커널 부팅 시, 각 인터럽트에 대하여 커널에 등록되며, 네트워크 카드가 송수신 신호를 발생할 경우 커널은 등록된 함수를 호출한다. 네트워크 디바이스 드라이버의 수신을 처리하는 방식은 크게 두 가지로 나뉜다. 커널 2.4 이전부터 사용된 방법과 커널 2.6에서부터 지원하기 시작한 poll 방식이다. 네트워크망을 통해 하드웨어에 전달된 데이터는 수신 인터럽트를 발생시키고, 하나의 수신 인터럽트는 하나의 패킷을 커널에 전달되었다. 이런 방식은 효율적이지는 않지만 기존의 네트워크 속도와 처리해야 되는 패킷이 많지 않을 경우 큰 문제가 되지 않았다. 하지만 기가 이더넷의 등장으로 처리해야할 패킷의 양이 늘고 많은 데이터를 신속하게 처리해야 하는 문제가 발생되었다. 커널 2.6에서 지원하는 poll 방식은 기존의 수신 인터럽트 서비스 함수에서 데이터를 처리할 때 폴링처리를 하는 것이다[7][8]. 이것은 다음 수신된 데이터에 대해서는 새로운 인터럽트 핸들러에서 처리하지 않고, 수신된 데이터를 처리 하는 동안 새로 도착한 패킷을 검사하여 한번에 처리하는 방식이다.

네트워크 장치는 장치 메모리의 버퍼에 패킷을 저장하고, 인터럽트를 발생시킨다. 커널에 등록된 인터럽트 핸들러는 패킷에 대한 새로운 소켓 버퍼를 할당하고 초기화 하며, 패킷을 장치 메모리에서 소켓 버퍼로 복사한다. 인터럽트 핸들러는 데이터 링크 프레임에 캡슐화된 패킷의 프로토

콜을 결정하고 커널 네트워킹 코드로 수신된 패킷을 전달하여 처리를 요청한다. 앞서 오래 걸리는 작업에 대하여 미룰 수 있는 함수의 수행으로 인터럽트의 반응을 빠르게 한다고 하였다. 네트워크 수신과정에서 소켓 버퍼를 할당하거나 패킷을 복사하는 과정은 오래 걸리는 작업에 속하므로 이는 미룰 수 있는 함수, 즉 소프트 IRQ에서 수행한다.

do_softirq 함수에 수행되는 소프트 IRQ는 [표 1]의 인덱스 순서로 진행된다. NET_RX_SOFTIRQ 소프트 IRQ의 처리는 *net/core/dev.c* 의 *net_rx_action* 함수로 구현이 되어있다. *net_rx_action* 함수는 디바이스 드라이버의 *poll* 함수를 호출하여 수신된 데이터를 하드웨어에서 얻어와 소켓 버퍼에 담아 커널에 전달하는 작업을 한다.

커널의 네트워크 처리는 패킷의 프로토콜 타입에 따라 전달되며, 마지막으로 응용 프로그램까지 전달된다. [그림 2]는 TCP/IP 서비스를 위한 커널의 네트워크 프로토콜 레이어 계층도이다.



[그림 2] 커널 네트워크 프로토콜 레이어 계층도

네트워드 디바이스 드라이버로부터 수신된 패킷이 최초로 전달되는 프로토콜 레이어는 IP 프로토콜 레이어이며, 이에 대응되는 함수는 *net/ipv4/ip_input.c*에 *ip_rcv* 함수이다. *ip_rcv* 함수는 패킷의 길이와 체크섬(checksum) 검사를 한다. 만일 잘못되었거나 잘렸다면 패킷을 드롭(drop) 시킨다. 또한 IP 헤더를 가져와 목적지 IP 주소를 확인하여 패킷을 다른 호스트로 전달할지, 전송 계층의 프로토콜로 전송해야 할지 결정하며 전송 계층의 프로토콜로 전송할 경우 IP 데이터그램을 재조합 한다. IP 데이터그램의 재조합이 끝난 데이터는 TCP 프로토콜 레이어로 데이터를 전송하기 위해 *tcp_v4_rcv* 함수를 호출한다.

*net/ipv4/tcp_ipv4.c*에 선언되어있는 *tcp_v4_rcv* 함수는 넘겨받은 패킷이 자신의 호스트로 보내는지 확인하고, TCP 헤더의 위치를 구한다. 또한 IP 헤더가 더 이상 필요하지 않으므로 버리는 작업을 하며, SYNC, ACK 등 여러 처리를 한다. 결국 TCP 프로토콜 레이어는 해당하는 INET sock 구조체가 연결(established) 상태에 있거나 혹은 연결대기(listen) 상태에 있는 것을 찾아 해당하는 INET 소켓의 receive_queue의 끝에다가 소켓 버퍼를 집어 넣어주게 된다.

INET 소켓 레이어 에서는 INET socket의 sleep queue에서 자고 있는 프로세스들을 전부 깨우는 역할을 한다. 만약 프로세스가 읽기를 원한다면, BSD socket에 대해서 *read* 함수를 호출 했을 것이며, 버퍼가 비어있고, 프로세스가 블로킹 모드(Blocking mode)를 사용하고 있다면, INET socket의 sleep queue에 잠들어 있게 된다. 따라서 원하는 데이터가 도착했으므로 다시 깨어나게 될 것이며, 깨어난 프로세스는 소켓 버퍼를 처리하여 응답 데이터를 만들게 된다.

커널의 네트워크 처리 과정은 일련의 FIFO (First-In First-Out) 방식으로 패킷을 하나씩 원하는 프로토콜 레이어로 전송하는 방식을 지닌다.

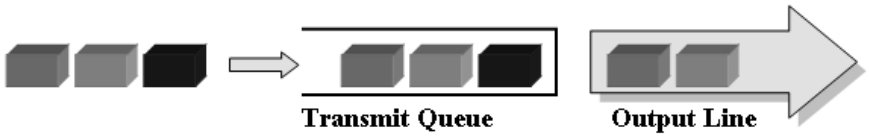
최종적으로 서버 역할을 하는 응용 프로그램으로 전달되어 응답 데이터를 만들고 패킷의 수신 경로의 반대로 패킷을 전달하여 송신한다.

2.2 네트워크 스케줄러

네트워크 스케줄러는 IP 라우터에서 패킷을 효율적으로 목적지까지 보내기 위하여 제안된 기법이다. 본 장에서는 IP 라우터에서 사용되는 대표적인 스케줄 기법에 대해서 알아본다. 또한 응답 시간 향상을 위한 기존 연구에 대해서 알아본다.

2.2.1 FIFO Queueing 스케줄러

FIFO Queueing 스케줄러는 네트워크 처리에서 가장 쉽게 사용되는 스케줄링 기법이다. FIFO (First In First Out) 방식은 먼저 들어온 것을 먼저 내보내는 방식으로 구현이 쉬운 장점이 있다.

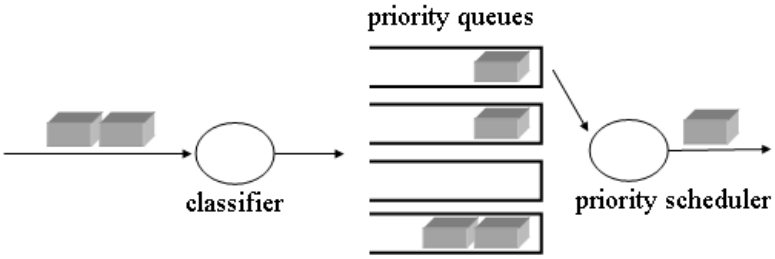


[그림 3] FIFO Queueing 구조

[그림 3]은 FIFO Queueing 기법의 구조이다. 그림에서 보다 시피 먼저 들어온 패킷이 먼저 처리됨을 확인할 수 있다. 하지만 이러한 처리는 서비스에 대한 차등 서비스를 제공할 수가 없다. 기본 커널의 패킷 수신 처리는 FIFO Queueing 스케줄의 방식으로 동작한다.

2.2.2 Priority Queueing 스케줄러

Priority Queueing 스케줄러는 우선순위 기반의 스케줄링 기법이다[2]. 서비스의 프로토콜 또는 인터페이스에 따라 큐를 정의하고 있으며, 각 큐는 서로 다른 우선순위를 지니고 있다.

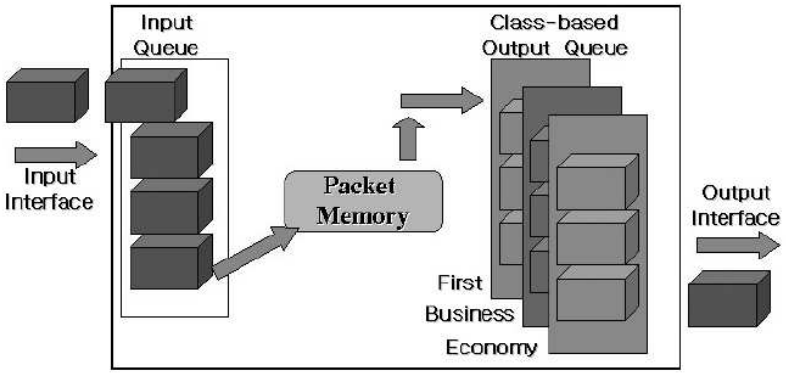


[그림 4] Priority Queueing 구조

[그림 4]는 Priority Queueing 스케줄러의 동작 구조를 나타낸다. Priority Queueing 스케줄러는 서비스별로 또는 인터페이스별로 각각의 큐를 지니고 있다. 이러한 큐들은 각각 우선순위가 있으며, 빠른 처리를 원하는 서비스 또는 인터페이스의 경우 높은 우선순위가 할당되어있다. 스케줄러는 처리해야할 패킷이 존재 하는 동안 가장 높은 우선순위부터 하나씩 패킷을 처리하여 준다. 이러한 방법은 서비스별로 차등 서비스를 제공할 수 있다. 하지만 Priority Queueing 스케줄러는 높은 우선순위의 서비스 또는 인터페이스에 많은 수의 패킷이 계속 수신될 경우 다른 우선순위를 지니는 서비스 또는 인터페이스에 패킷은 처리 되지 못하는 기아 현상(Starvation) 이 발생할 수 있다.

2.2.3 Class-Based Queueing 스케줄러

Class-Based Queueing (이하 CBQ) 스케줄러는 Priority Queueing의 변형으로 서비스에 따라 큐를 정의하고 있으며, 특정 높은 우선순위의 서비스 클래스의 트래픽이 시스템 자원과 대역폭을 독점하는 것을 막음으로써 공평성을 제공하고, 상대적으로 낮은 우선순위를 지니는 클래스의 기아(Starvation)를 막기 위하여 고안되었다[3][9][10]. 즉, 서로 다른 등급에 따라서 우선순위에 기초하여 보다 대역폭을 할당하는 것이다.



[그림 5] Class-Based Queueing 구조

[그림 5]는 High, Medium, Low 3개의 우선순위를 갖는 CBQ를 보여주고 있다[10]. 각 우선 순위 큐에 한계값을 할당하며, 각 큐의 트래픽들은 자신의 스케줄링 라운드에 설정된 서비스 한계값에 도달할 때까지 서비스 되거나 큐가 비워질 때까지 서비스 받는다. CBQ는 각 큐의 우선순위에 따라 각기 다른 양의 서비스 한계 값을 두고 이 값에 따라 스케줄링 라운드 마다 각 큐를 서비스함으로써 특정 서비스 클래스가 서비스 받지 못하는 기아(Starvation)문제를 막을 수 있다.

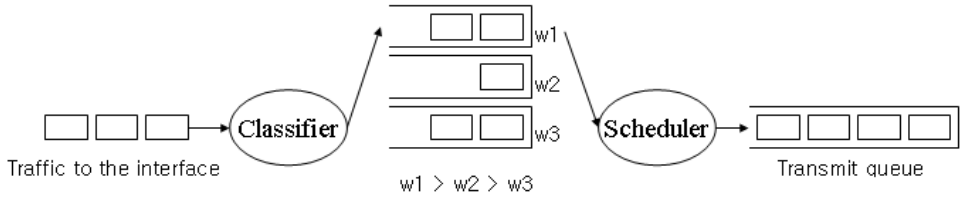
CBQ는 높은 우선순위의 서비스에 더 많은 양의 자원을 할당하고

낮은 우선순위의 서비스에는 조금 적은 양의 자원을 할당함으로써, Priority Queueing에서 높은 우선순위를 갖는 서비스를 무조건 먼저 처리함에 따라 나타나는 낮은 우선순위의 서비스가 자원을 할당받지 못하여 발생하는 굶주림 문제를 해결할 수 있다. 즉 Priority Queueing과 비교해 낮은 우선순위를 갖는 트래픽에 대해 자원을 무조건 빼앗는 것이 아니라 약간의 일정한 자원을 할당하여 서비스를 수행하게 된다.

CBQ 스케줄링 기법은 기존의 우선순위 기반에서 기아(Starvation) 현상은 방지 해주지만 근본적으로 서비스별 클래스로 우선순위를 할당하기 때문에 하나의 서비스 내에서 각 고객을 등급화 하는 기능을 제공하지 않는다. 따라서 본 논문에서 제안하고자 하는 서비스 내에서의 차등 서비스 제공에 적합하지 않은 기법이다.

2.2.4 Weighted Fair Queueing 스케줄러

Weighted Fair Queueing (이하 WFQ) 스케줄 기법은 어플리케이션별 트래픽 흐름을 선별하여 각각의 큐에 분류하고 정해진 우선순위에 따라 트래픽을 전송 순서를 스케줄하여 실시간 어플리케이션의 QoS(Quality of Service) 보장을 가능하게 하는 기법이다[4][11]. [그림 6]은 WFQ 스케줄러의 동작 구조를 나타낸다.

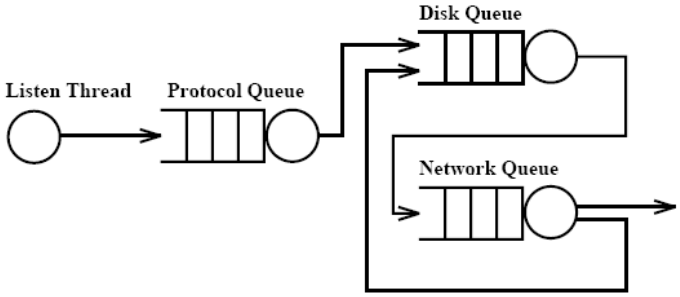


[그림 6] Weighted Fair Queueing 구조

주어진 가중치에 따라 분류된 저장된 각 큐의 패킷들은 검증된 알고리즘에 의해 비순차적인 처리를 수행하도록 스케줄링 된다. FIFO Queueing이 우선순위가 부여 되지 않은 순차적인 방법이라면 WFQ는 패킷 단위의 우선순위 기반의 처리를 가능하게 한다.

2.2.5 SRPT 커넥션 스케줄링 기법

M.E.Crovella et al.는 SRPT[12]를 이용한 커넥션 스케줄링(이하 SRPT-CS)을 제안하였다[1][13]. SRPT-CS는 클라이언트와 커넥션이 형성된 후에 커넥션이 서비스 받는 과정을 프로토콜 큐, 디스크 큐, 그리고 네트워크 큐를 통해서 구성하였다. [그림 7]은 커넥션이 서비스 받는 과정을 보여준다[1].



[그림 7] SRPT-CS 서버 구조

클라이언트들의 요청은 Listen Thread 에 의하여 수신되며, 프로토콜 큐에 할당된다. 프로토콜 큐에서는 요청들이 어떤 문서에 대한 요청인지 모르기 때문에 들어온 순서로 큐에 있는 요청들이 요청한 문서의 크기 정보를 얻는다. 이후, 요청한 문서의 크기에 따라 디스크 큐에 저장되며, 요청된 문서의 크기와 현재 처리가 남은 데이터양에 따라 네트워크 큐에 저장되어 스케줄링 된다. 따라서 문서의 크기가 작거나 처리가 남은 데이터의 양이 적을수록 빨리 스케줄링 되기 때문에 문서에 대한 평균 응답시간이 향상되었으며, 문서의 크기를 고려하지 않는 스케줄링 기법에 비해 평균 응답시간이 향상되었다.

SRPT-CS 스케줄링 기법은 패킷 처리의 순서를 바꾸어 전체 평균

응답 시간을 향상시켰다. 하지만 이것은 패킷들의 차등 서비스를 하기 위함이 아닌 응답 시간 향상을 위한 기법으로 차등 서비스에 적합하지 않은 기법이라 할 수 있다.

2.3 Percentile 리눅스 프로세스 스케줄러 기법 연구

기존 연구 중에서 차등 서비스를 위해 패킷 스케줄에 사용된 Percentile 기법을 웹 서버에 적용하여 서버에 들어오는 다양한 서비스 요구를 서로 다른 클래스로 분류하여 차등적인 서비스를 제공하는 방법이 제안되었다[14].

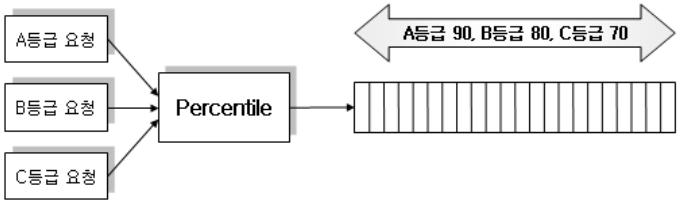
2.3.1 Percentile 기법

Percentile 스케줄링 기법은 패킷 별 우선순위에 기반을 둔 네트워크 라우터에 사용된 스케줄링 기법이다[5][14]. Percentile 스케줄에서 클래스 *i*에 속하는 패킷의 스케줄 우선순위 계산식은 다음과 같다.

$$Pr(i) = \frac{S_i}{N_i * P} \qquad \text{수식 1}$$

식에서 S*i*는 주어진 마감 시간 안에 처리된 패킷의 개수이며, N*i*는 현재 클래스로 분류된 패킷의 수이다. P는 현재 클래스에 주어진 Percentile 값이며 계산된 식에 따라 해당 패킷의 Pr 값이 결정된다. 계산된 Pr 값이 낮을수록 해당 *i* 클래스의 우선순위가 높아지며, 해당 클래스의 패킷들은 높은 Pr 값, 즉 낮은 우선순위를 지니는 패킷들 보다 먼저 처

리된다. 우선순위 값은 계산식에 의하여 현재 클래스로 분류된 패킷의 수가 많고, 마감시간 안에 처리된 패킷의 수가 적으며, 클래스에 할당된 Percentile 값이 클수록 우선순위가 높아짐을 알 수 있다.

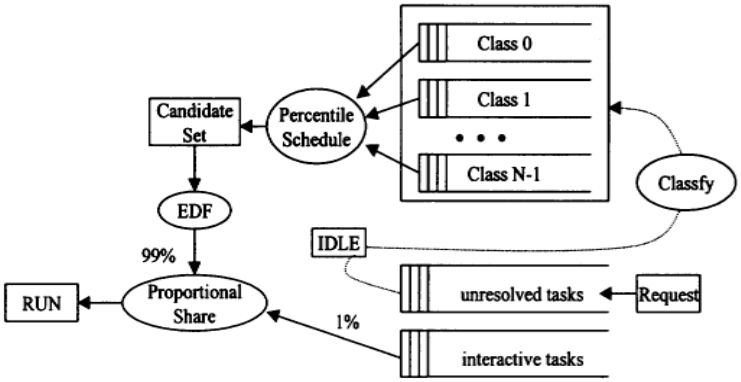


[그림 8] 등급에 따른 Percentile 스케줄러의 처리

[그림 8]은 Percentile 에 따른 패킷 처리 비율을 나타낸다. A등급의 Percentile 값을 90, B등급의 Percentile 값을 80, C등급의 Percentile 값을 70으로 하였을 때, 클라이언트로부터 A, B, C 등급의 요청이 들어왔을 경우 Percentile 식에 의하여 결정된 우선순위에 따라 요청의 처리 순서는 A 등급이 우선처리 되며, B등급, C등급이 처리됨을 그림에서 알 수 있다.

2.3.2 Percentile에 기반을 둔 프로세스 스케줄러

리눅스에서 Percentile 프로세스 스케줄러의 구성은 [그림 9]와 같다[14]. 크게 등급 부여과 Percentile 프로세스 스케줄러로 분류 된다. 등급 부여는 요청의 등급을 결정하고 각 등급은 Percentile 프로세스 스케줄러에 의해 우선순위를 부여 받으며 먼저 도착한 요청 순으로 실행을 하게 된다.



[그림 9] Percentile 프로세스 스케줄러 동작 구조

수신된 요구는 클래스를 부여 받기 위해 대기 큐에 들어가며 CPU가 더 이상 실행할 태스크가 없을 경우 이 요구들을 등급에 맞게 분류한다. 각 클래스에 맞게 분류된 요청들은 Percentile 계산식에 의하여 우선순위를 부여 받고, 클래스들의 우선순위를 비교하여 가장 큰 우선순위를 가지는 클래스 큐에서 EDF(Earliest Deadline First) 기준에 따라 가장 먼저 deadline에 근접한 프로세스를 실행 시켜준다.

실제 프로토콜 처리는 네트워크 하반부에서 이루어지며 FIFO방식으로 수행된다. 프로토콜 처리 과정을 거친 요청은 스케줄에 의하여 서버 프로그램이 수신한 이후에 Percentile 계산식에 의해 우선순위가 결정된다. 따라서 리눅스 스케줄에 의하여 서버 프로그램이 요청을 수신하기 이전에는 기존 네트워크 처리에 따라 FIFO로 동작하기 때문에 Percentile 기법에 의한 제어가 되지 않음을 의미 한다. 또한 등급화와 우선순위를 결정하는 방법은 사용자 레벨에서 동작하는 방식으로 리눅스에서 사용되는 프로세스 스케줄러 방식에 의하여 실행된다. 프로세스 전환(context switch)이 수행되는데 오버헤드가 큰 작업에 속한다. 따라서 요청을 처리하는데 있어 많은 부하가 발생할 수 있다.

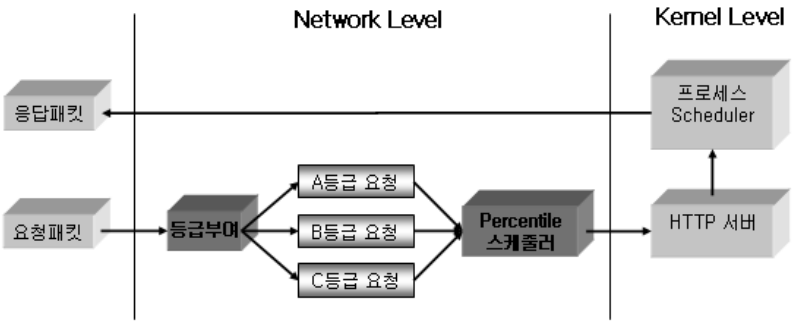
제 3장 네트워크 스케줄러 설계 및 구현

3.1 Percentile 네트워크 스케줄러 설계

본 논문에서 구현하고자 하는 네트워크 스케줄러가 가져야 할 구조와 기능을 언급한다. Percentile 기법[5]을 이용한 네트워크 스케줄러는 널리 쓰이는 운영체제 중 하나인 리눅스에서 구현되었으며, 버전은 2.6.16 에서 구현하였다.

3.1.1 Percentile 네트워크 스케줄러 설계

본 논문에서는 등급에 따른 차등 서비스를 위해 우선순위에 기반을 둔 스케줄러를 이용하여 수신된 요청 패킷의 처리 순서를 제어하고자 한다. 차등화 서비스를 위한 네트워크 스케줄러는 크게 두 부분으로 나눌 수 있다. 첫째, 등급 결정은 패킷의 등급을 결정한다. 기존의 서비스별로 등급을 결정 하는 방법과는 달리 하나의 고객, 즉 하나의 클라이언트가 등급을 지니는 형태가 된다. 둘째는 패킷의 처리 순서를 결정하는 스케줄러이다. 스케줄러는 2장에서 관련연구로서 언급하였던 Percentile 기법을 이용한다. Percentile 기법은 차등 서비스를 위한 스케줄 기법이며, 각 등급을 일정 마감 시간 안에 처리 되는 것을 만족시키기 위한 기법이다. 또한 하나의 등급으로 많은 요청의 수신에 의한 타 등급의 기아(Starvation) 현상을 막을 수 있는 효율적인 기법으로 차등 서비스 제공에 있어 적합한 방법이다.

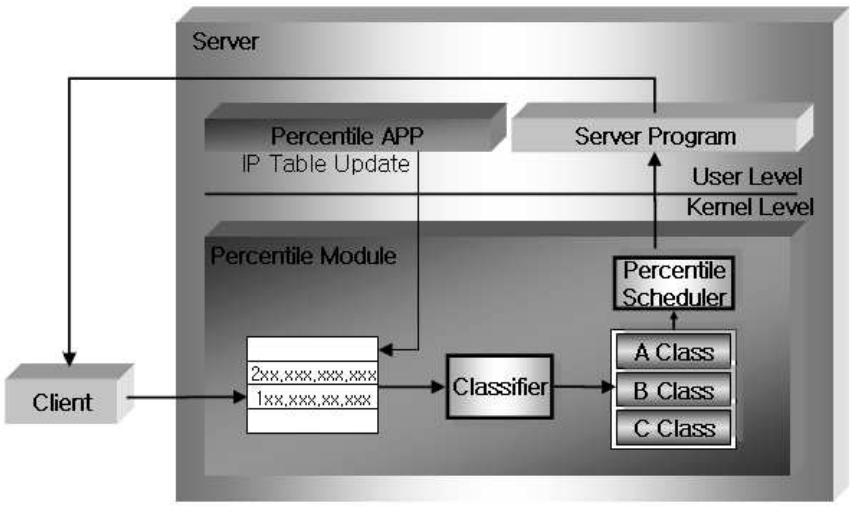


[그림 10] 차등 서비스를 위한 네트워크 스케줄러 구조

[그림 10]은 네트워크 스케줄러의 구조를 나타낸 것이다. 클라이언트로부터 수신된 요청 패킷은 서버에 수신된 직후 등급을 부여 받는다. 패킷의 등급은 클라이언트 주소로 결정된다. 즉 수신된 패킷의 IP 주소인 소스 어드레스(Source Address)를 확인 하여 해당 패킷의 등급을 결정한다. IP를 이용하여 등급을 결정하는 방식에는 하나의 호스트만 등급을 받을 수 있기 때문에 사용자가 다른 호스트로 이동할 경우 등급에 따른 서비스를 받지 못하는 문제가 따른다. 실제 웹 서비스에서는 사용자의 ID 인증을 거쳐 접속을 하며, IP와 더불어 이용하여 등급을 결정할 것이다. 하지만 본 논문에서는 네트워크 프로세싱 단계의 수정을 통해 스케줄러의 동작을 확인함으로 IP를 통한 등급 결정 방법을 사용한다. 각 등급에 따른 IP 주소는 테이블 형태로 유지되며 수신된 패킷들은 해당 IP 주소와 비교를 통해 등급을 결정한다.

등급이 결정된 패킷은 해당 등급에 해당하는 등급 큐(Class Queue)에 쌓이게 되며 2장에서 언급한 Percentile 기법을 이용한 스케줄러에 의하여 선택된 요청 패킷이 HTTP 서버에 전송이 된다. 요청 패킷이 HTTP 서버로 전송된 후, 응답 패킷의 생성은 커널 프로세스 스케줄러에 의하여 수행이 될 때 생성이 되며, 생성된 응답 패킷은 해당 요청을 보낸 클라이언트로 전송이 된다.

3.1.2 Percentile 네트워크 스케줄러 세부구조



[그림 11] Percentile 네트워크 스케줄러 세부구조

[그림 11]은 Percentile 네트워크 스케줄러의 세부 구조를 보여준다. 클라이언트로부터 수신된 요청 패킷은 IP 테이블에 등록되어있는 IP 주소와 요청 패킷의 송신지 주소를 비교하여 해당 등급을 부여 받는다. 등급에 따라 분류된 패킷들은 각 클래스 큐에 저장이 된다. 다음으로 Percentile 스케줄러에 의하여 가장 높은 우선순위의 클래스의 큐에서 패킷을 꺼내어 해당 요청의 처리를 하게 된다. 각 클래스 큐는 FIFO (First In First Out) 구조를 가짐으로 먼저 도착한 패킷이 우선 처리 된다.

Percentile 스케줄러에 의하여 선택된 패킷은 서버 프로그램으로 전달되어 해당 요청의 응답을 생성하고 네트워크를 거쳐 클라이언트로 응답 패킷을 전송하게 된다.

Percentile APP는 IP 테이블에 IP 주소와 해당 주소의 등급을 입력하고 삭제 하는 역할을 하게 된다. 또한 Percentile 계산식에 사용되는 여러 입력 값들을 설정하는 역할을 포함한 여러 기능을 한다.

3.1.3 인증 및 등급 부여

Percentile 기법을 이용하기 위하여 등급이 필요하며, 등급에 따라 우선순위가 결정되어진다.

표 2 계약 기간에 따른 등급 부여

A 등급	B 등급	C 등급	D 등급	E 등급
3년 약정	1년 약정	6개월 약정	3개월 약정	그 외

[표 2]는 계약 기간에 따른 등급 부여의 예를 보여준다. 가장 높은 등급인 A 등급은 3년 약정으로 계약한 사용자이며 점차 등급이 낮아질수록 계약 기간이 짧아진다. 이러한 계약 기간에 따른 등급의 분류는 웹 호스팅 업체, 온라인 게임 서비스 같은 종류의 서비스에서 사용되어질 수 있다. 고객이 서버에 서비스를 요청을 할 경우 고객 데이터에서 서비스 계약 기간의 정보를 가져와 등급을 분류하고 등급에 맞는 차등 서비스를 제공한다.

[표 3]은 거래 액수에 따른 등급 부여의 예이다. 가장 높은 등급인 A 등급은 거래 액수가 100만원 이상인 고객이 되며, 점차 등급이 낮아질수록 거래 액수가 줄어든다.

표 3 거래 액수에 따른 등급 부여

A 등급	B 등급	C 등급	D 등급	E 등급
100만원 이상	50만원 이상	10만원 이상	5만원 이상	5만원 미만

이러한 등급 부여는 쇼핑몰과 같은 서비스에서 이루어 질수 있는 등급 분류를 나타낸다. 거래량이 많은 고객이 서버로 서비스 요청을 할 경우 고객의 데이터에서 과거 거래량 정보를 가져와 등급을 부여하고 등급에 맞는 차등 서비스를 제공한다.

표 4 IP에 따른 등급 부여

분류	등급
220.x.x.x	A
192.x.x.x	B
210.x.x.x	C
그 외	D

[표 4]는 IP를 통한 등급을 부여의 예를 보여준다. 각 클라이언트가 [표 4]와 같은 IP 주소를 가진다고 할 때, 각 IP 주소는 클래스 A, 클래스 B, 클래스 C에 대응된다. 이러한 등급 분류는 인트라넷과 같은 서비스에서 이용 가능하다.

본 논문의 경우 실험을 위하여 IP 주소에 따른 등급 분류를 사용한다. 패킷이 서버에 수신되면 우선 해당 패킷의 IP가 IP 테이블에 존재하는지 확인한다. 만일 해당 패킷의 IP 주소가 IP 테이블에 이미 존재 한다면 해당 IP 주소의 등급을 수신된 패킷에 부여하게 된다. 수신된 패킷의 IP 주소가 IP 테이블에 존재하지 않는다면, A, B, C 등급을 제외한 기본

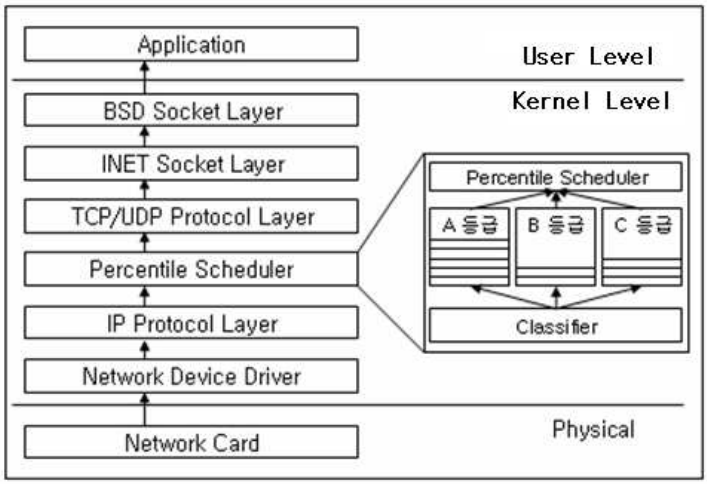
최하위 등급을 부여한다. 즉 우선 처리되어야할 A, B, C 등급이 부여된 패킷을 처리하기 위해 기본 최하위 등급인 D 등급을 부여하는 것이다. 최하위 등급인 D 등급을 부여 받은 패킷은 Percentile 계산식에 따라 다른 등급에 비하여 선택이 늦게 되며, 다른 등급의 처리 비율이 높아져 최하위 등급인 D 등급의 우선순위가 높아지는 역전 현상이 일어날 때 D 등급의 요청 패킷이 처리 된다.

IP 테이블을 업데이트하기 위하여 IP 정보를 입력하게 될 때 해당 IP 와 함께 등급 정보를 같이 저장하는 방법을 사용한다. IP 테이블의 업데이트, 즉 입력 및 삭제 등의 제어를 하기 위하여 Percentile APP 제작하여 사용한다.

3.1.4 네트워크 스케줄러

앞장에서 IP를 통하여 등급을 결정하고 등급을 기준으로 Percentile 스케줄을 적용한다고 하였다. 기존의 커널의 네트워크 처리 방식은 먼저 수신된 패킷이 먼저 처리 되는 FIFO방식으로 동작한다. FIFO 방식은 가장 구현이 쉽고 기본적으로 쓰이는 구조로 차등 서비스를 제공하기 위하여 이를 수정하여 스케줄러의 기능을 부여해야 한다.

[그림 12]는 네트워크 스케줄러가 기존 커널의 네트워크 처리 과정에서 처리 되는 위치를 보여준다.



[그림 12] Percentile 네트워크 계층도

Percentile 네트워크 스케줄러는 IP 프로토콜 레이어 와 TCP/UDP 프로토콜 레이어 사이에서 동작한다. 본 논문에서는 서비스별 차등 서비스가 아닌 하나의 서비스에서의 차등 연결을 위하여 IP를 통한 등급 분류를 한다고 하였다. IP 프로토콜 레이어는 수신된 패킷의 IP 헤더 위치를 구하고 체크섬(checksum) 검사와 재조합이 이루어진다. 따라서 IP 헤더의 위치를 구함으로서 송신지의 IP 주소를 얻을 수가 있으며, 이를 이용하여 등급 분류가 가능하게 된다. 등급이 분류된 패킷은 각 등급 큐에 쌓이게 되며 Percentile 스케줄러에 의하여 가장 높은 우선순위의 큐에서 패킷을 가져와 상위 TCP/UDP 프로토콜 레이어로 전송한다.

수신된 패킷은 등급을 분류하여 각 등급 큐에 쌓인다. 이러한 동작은 소프트 인터럽트 루틴에 의하여 동작하게 되며, 스케줄러에 의한 상위 TCP/UDP 프로토콜 레이어로의 패킷 전송이 이루어진다. 소프트 인터럽트의 동작 함수인 *do_softirq* 함수는 HI_SOFTIRQ, TIMER_SOFTIRQ, NET_TX_SOFTIRQ, NET_RX_SORFIRQ, SCSI_SOFTIRQ,

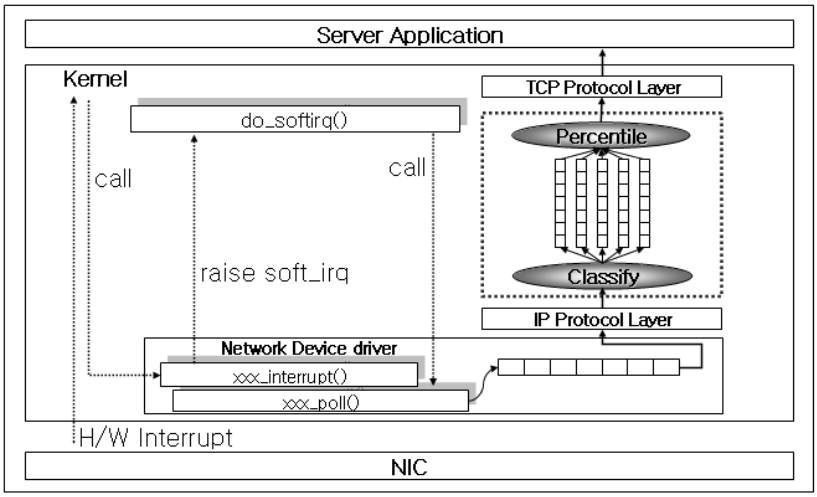
TASKLET_SOFTIRQ 순으로 해당 함수를 동작한다. Percentile 계산식에 의한 스케줄러 동작은 TASKLET_SOFTIRQ에 의하여 동작하도록 한다. 이것은 NET_RX_SOFTIRQ 루틴에 의하여 패킷들이 각 큐에 쌓이고 난 뒤 스케줄을 하여야 하는데, NET_RX_SOFTIRQ 이후에 동작이 가능한 소프트 인터럽트의 종류는 TASKLET_SOFTIRQ 이다. 따라서 Percentile 에 의한 스케줄러는 TASKLET_SOFTIRQ 에 의하여 동작하도록 한다.

3.2 Percentile 네트워크 스케줄러 구현

Percentile 네트워크 스케줄러는 리눅스 커널 2.6.16 기반에서 구현되었으며, 모듈 형식으로 제작되었다. 모듈은 커널이 부팅될 시 동작 할 수 있도록 빌트인(built-in)방식으로 구현되었다.

3.2.1 Percentile 네트워크 스케줄러의 동작

[그림 13]은 네트워크 디바이스 드라이버와 소프트 인터럽트와의 동작 관계를 보여준다.



[그림 13] 인터럽트와의 동작 구조

네트워크 인터페이스 카드는 패킷을 수신하면 커널에게 수신된 패킷이 있다는 것을 알리고 커널은 네트워크 디바이스 드라이버의 인터럽트 루틴을 실행 시킨다. 네트워크 디바이스 드라이버 데이터가 커널 내에 등록이 되며 디바이스 폴 리스트가 추가 되고 소프트 인터럽트인 NET_RX_SOFTIRQ 인터럽트를 활성화 시킨다. 커널은 소프트 인터럽트인 *do_softirq* 함수를 호출하며, 패킷 수신 인터럽트 루틴인 *net_rx_action* 함수를 호출하고 이 함수는 폴 리스트에서 디바이스 정보를 가져와 디바이스 드라이버의 *poll* 함수를 동작시킨다. *poll* 함수는 소켓버퍼를 할당하고 수신데이터와 연결하여 IP 프로토콜 레이어로 전송한다. IP 프로토콜 레이어에서 기존의 역할 즉 체크섬(checksum) 검사와 재조합(fragment)을 한 뒤, TCP/UDP 프로토콜 레이어로 보내기 전에 수신된 패킷의 IP 헤더에서 IP 주소를 가져와 Percentile 모듈의 IP 테이블과 비교하여 해당 패킷의 등급을 구한다. 등급이 결정된 패킷은 해당 등급의 큐에 삽입된다.

Percentile 스케줄러는 Percentile 계산식을 이용하여 각 등급의 우

선순위를 계산하고, 가장 높은 우선순위를 지니는 등급의 큐에서 패킷을 꺼내 상위 TCP 프로토콜 레이어로 전송한다.

3.2.2 패킷의 성공적인 처리

Percentile 스케줄러에 사용되는 Percentile 계산식은 매우 중요한 역할을 한다. 각 클래스 별로 차등 서비스를 제공할 뿐만 아니라 기아(Starvation) 현상도 방지해준다. Percentile 계산식은 현재 클래스로 분류된 패킷의 수와 마감 시간 내에 성공적으로 처리된 패킷의 수에 따라 클래스의 우선순위 값의 변화가 일어나게 된다.

서버내에서의 패킷 처리의 성공 실패는 제안된 스케줄러가 동작하는 시점을 시작으로 프로세스에 의하여 요청이 처리되고 난 뒤 응답 패킷을 만들어 네트워크로 내려 보내는 시점의 시간차를 이용하여 성공 실패를 결정한다. 성공과 실패가 검사되는 시점은 IP 헤더가 생성이 되는 IP 프로토콜 레이어이며, 송신할 패킷의 IP 주소를 가져와 해당 등급의 패킷 처리가 마감시간 내에 처리 되었는지 여부를 검사한다. 성공과 실패를 결정하는 기준이 되는 값은 패킷들의 평균 처리 시간 값을 이용하여 평균 처리 시간 보다 작을 경우 성공, 클 경우 실패로 판단한다.

3.2.3 Percentile 네트워크 스케줄러 데이터 구조 및 중요 함수

Percentile 네트워크 스케줄러에서 사용되는 데이터 구조는 [표 5]과 같다. *PercentileClass* 구조체는 모듈 초기화시 클래스의 수만큼 메모리를 할당하며, 구조체 배열 형태로 유지 된다.

표 5 PercentileClass 데이터 구조

```
struct PercentileClass {
        int pk_count;
        int pk_success;
        int pk_fail;
        int percentile_value;
        int priority;
        struct sk_buff_head list;
};
```

PercentileClass 구조체는 각 클래스 마다 하나씩 지니며, Percentile 계산식에 사용되는 분류된 패킷의 수, 마감시간 내에 처리된 패킷의 수, 클래스의 Percentile 값을 포함한다. 또한 우선순위 값을 확인 할 수 있는 *priority* 변수와 해당 클래스의 큐를 나타내는 소켓 버퍼 리스트 구조체를 지닌다. 소켓 버퍼 리스트 구조체는 *linux/skbuff.h* 에 선언되어 있으며 이전 소켓 버퍼와 이후 소켓 버퍼를 가리키는 포인터, 큐 리스트 길이를 저장하는 변수와 락을 제어하기 위한 변수를 지닌다.

[표 6]는 IP 테이블을 관리하기 위한 데이터 구조이다. *ClassifyEntry* 구조체는 IP 테이블에 하나의 엔트리를 보여준다.

표 6 ClassifyEntry 데이터 구조

```
struct ClassifyEntry {
        __be32 ip_addr;
        unsigned int IP_Class;
        struct list_head list;
};
```

IP를 저장할 수 있는 *__be32* 타입의 *ip_addr* 변수가 존재하며 일반적인 인터넷 호스트 주소가 아닌 네트워크 바이트 오더 형태의 IP를 저장한다. 또한 해당 IP 주소의 등급이 되는 *IP_Class* 변수를 지니며 다음 IP 엔트리를 가리킬 수 있는 *list_head* 구조체를 지닌다.

표 7 Percentile 스케줄러 중요 함수

함수	기능
per_input_skb	수신된 소켓 버퍼를 해당 클래스 큐에 input
per_fail_check	마감 시간 내에 소켓 버퍼가 처리 되었는지 검사
percentile_sched	Percentile 기법을 이용한 네트워크 스케줄러 함수

[표 7]는 Percentile 네트워크 스케줄러 모듈의 중요 함수를 나타낸다. *per_input_skb* 는 수신된 패킷의 클래스를 구하고 해당 클래스의 소켓 버퍼 리스트의 맨 마지막에 연결시킨다. 리스트에 소켓 버퍼를 연결시키는 함수는 *skb_queue_tail* 함수이며, *include/linux/skbuff.h* 에 선언되어있다. 마지막으로 *per_input_skb* 함수는 현재 클래스의 Percentile 계산식 값인 분류된 패킷수를 증가 시켜준다. *per_input_skb* 함수는 IP 프로토콜 레이

어의 마지막에서 호출된다.

per_fail_check 함수는 마감 시간 내에 패킷이 처리 되었는지 검사를 하는 함수이다. 패킷이 마감 시간 내에 성공적으로 처리가 되었는지를 판단하여 Percentile 계산식의 성공적으로 처리된 패킷의 수를 갱신해 준다. *per_fail_check* 함수는 응답 패킷이 생성이 된 후 IP 프로토콜 레이어를 거쳐 클라이언트로 송신하기 직후에 호출이 된다. IP 프로토콜 레이어에서 IP 헤더를 생성함으로 IP 헤더에서 IP 주소를 가져와 해당 클래스의 패킷이 마감 시간 내에 처리 되었는지 검사가 가능하다.

per_input_skb 와 *per_fail_check* 함수는 EXPORT_SYMBOL 타입으로 선언한다. EXPORT_SYMBOL은 외부 모듈에서 현재 모듈의 함수의 사용이 가능하게 만들어주는 매크로 함수이다.

percentile_sched 함수는 Percentile 계산식을 통해 가장 높은 우선순위의 등급을 구하여 해당 큐의 패킷을 상위 프로토콜 레이어로 전송하는 역할을 한다. 우선순위 계산의 결과 값은 정수 이하의 소수점 값이므로 우선순위 비교의 오버헤드를 최소화하기 위하여 높은 계산 값이 높은 우선순위로 결정되게 하였다. 상위 프로토콜 레이어로 전송할 패킷이 결정되면 *net/ipv4/tcp_input.c*에 선언되어있는 TCP 프로토콜 레이어의 시작 함수인 *tcp_v4_rcv* 함수를 호출한다. *percentile_sched* 함수는 TASKLET 소프트 인터럽트로 실행되며 *per_input_skb* 함수 마지막에서 TASKLET 소프트 인터럽트로 등록이 된다.

3.2.4 리눅스 커널 네트워크 수정

기존의 리눅스 네트워크 처리를 수정하여 본 논문에서 제안하는 네트워크 스케줄 기능을 추가하였다. [표 8]은 기존 네트워크 처리에서 수정된 함수를 나타낸다.

표 8 리눅스 커널 네트워크의 수정된 함수

분류	함수
패킷 수신	ip_local_deliver_finish
패킷 전송	ip_finish_output

ip_local_deliver_finish 함수는 *net/ipv4/ip_input.c* 에 선언되어있으며, IP 프로토콜 레이어에 속하는 함수이다. *ip_local_deliver_finish* 함수는 TCP 프로토콜 레이어에 속하는 함수를 호출하기 직전의 함수로서 패킷의 재조합이 종료된 시점이 된다. 또한 클라이언트로부터 수신된 패킷은 IP 헤더 주소를 설정한 상태이므로 IP를 통한 등급 분류가 가능하다. 따라서 *ip_local_deliver_finish* 함수의 종료 시점에서 등급을 분류하고 해당 등급의 큐에 패킷을 삽입하는 *per_input_skb* 함수를 호출한다.

ip_finish_output 함수는 *net/ipv4/ip_output.c* 에 선언되어있는 함수이며, 이 함수 역시 IP 프로토콜 레이어에 속하는 함수이다. 응답 패킷이 네트워크를 통하여 클라이언트로 보내어지는 직전을 요청 패킷의 처리 종료로 판단하였을 경우 디바이스 드라이버의 송신 인터럽트를 발생하는 직전도 가능하다. 하지만 *ip_finish_output* 함수 내부에서는 현재 응답 패킷의 사이즈가 한번에 보낼 수 있는 크기인지 조사하여 크기가 클 경우 패킷을 분할하게 된다. 따라서 하나의 요청에 따른 응답이 여러 개의 패킷으

로 분할될 경우 성공 실패의 계산에 어려움이 따름으로 패킷을 분할하기 진적에 응답 패킷의 성공 실패를 판단하였다. 또한 클라이언트로 보내어질 패킷은 이 단계에서 IP 헤더까지 포함한 응답 패킷이 완성된 상태이며 IP 를 통하여 해당 등급 요청에 따른 성공 실패를 판단 할 수 있으므로 Percentile 모듈의 *per_fail_check* 함수를 호출한다.

3.2.5 Percentile API 기능

Percentile 네트워크 스케줄러는 모듈로 제작되었으며 세부적인 제어를 위한 *ioctl* 명령을 지니고 있다. 이를 이용하여 Percentile 네트워크 스케줄러를 제어하기 위한 상위 응용 프로그램을 제작하였다.

표 9 Percentile API

API	기능
perOpen	디바이스 파일 open
perClose	디바이스 파일 Close
perStart	Percentile 네트워크 스케줄러 시작
perStop	Percentile 네트워크 스케줄러 정지
perReset	Percentile 데이터 초기화
perGetStatus	Percentile 네트워크 스케줄러 정보를 디스플레이
perSetPvalue	등급의 Percentile 값 변경
perSetNvalue	Percentile 스케줄러의 성공 실패 기준값 변경
perSetQlen	Percentile 스케줄러의 동작 Tick 설정
perAddIP	IP 테이블에 IP와 해당 등급을 등록
perDeleteIP	IP 테이블에 IP 삭제

[표 9]는 응용 프로그램에서 호출이 가능한 API들이다. *perOpen* 과 *perClose* 함수는 캐릭터 디바이스 파일인 Percentile 디바이스 파일을 열고 닫을 때 사용하는 함수이다. *perOpen*과 *perClose* 함수 호출 성공 시 파일 디스크립터 넘버를 반환하지만 실패 시 -1을 반환하게 된다.

perStart 와 *perStop* 함수는 Percentile 네트워크 스케줄러 동작의 시작과 정지를 수행하며, *perReset* 함수는 Percentile 네트워크 스케줄러에 사용되는 데이터 구조 정보를 초기화 하는 역할을 한다. 초기화 되는 정보는 등급 부여에 필요한 IP 테이블과 Percentile 계산식에 사용되는 분류된 패킷의 수, 성공적으로 마감시간 내에 처리된 패킷의 수가 된다.

perGetStatus 함수는 현재 Percentile 스케줄러에 사용되는 데이터 구조의 정보를 보여준다. Percentile 네트워크 스케줄러 모듈은 *proc* 파일 시스템을 이용하여 정보를 출력하며, *perGetStatus* 함수는 이와 동일한 정보를 보여준다. *perGetStatus* 함수에 의하여 보여주는 정보는 각 등급의 분류된 패킷의 수와 성공 실패의 수와 설정된 Percentile 값과 우선순위 값이 되며 그 밖에 성공 실패의 기준이 되는 평균 응답시간의 수와 IP 테이블 정보를 출력하게 된다.

perSetPvalue 와 *perSetNvalue* 함수는 Percentile 스케줄러에 사용되는 Percentile 계산식의 변수 값을 변경하는데 사용된다. *perSetPvalue* 함수는 등급에 부여된 Percentile 값을 변경하는데 사용되며, 입력 값으로 등급과 Percentile 값을 받는다. *perSetNvalue* 함수는 마감시간 안에 패킷이 성공적으로 처리 되었는지 검사하는 기준이 되는 평균 응답시간을 설정하는데 사용된다.

perSetQlen 함수는 Percentile 스케줄러를 동작하는 시점을 설정하기 위하여 사용된다. *default* 값으로 수신된 패킷이 있을 경우 스케줄을 시작하도록 설정되어있으며, 사용자의 값 입력 시 *do_softirq* 함수의 호출 횟수가 입력된 값보다 클 경우 스케줄러가 동작하게 된다. Percentile 스케줄러가 성공적으로 동작할 수 있는 조건으로 많은 패킷이 쌓여있어야 한다는 전제가 필요하다. 따라서 기존의 네트워크 방식에서는 FIFO로 패킷을 처리하기 때문에 패킷이 쌓이지 않으며, 제안된 Percentile 기법을 사용 시 수신된 패킷들을 쌓이게 하는데 사용되는 함수이다.

perAddIP 와 *perDeleteIP* 함수는 클라이언트의 IP 주소와 등급을 받아 IP 테이블을 갱신하는데 사용된다. *perAddIP* 함수는 IP 주소를 받

아 일반적인 인터넷 호스트 주소에서 네트워크 바이트 오더 형태의 IP 주소로 변환하며, 변환된 IP 주소와 해당 IP 주소가 분류될 등급을 IP 테이블에 엔트리로 입력한다. *perDeleteIP* 함수는 IP 주소만을 받으며 바이트 오더 형태의 주소로 변환시켜 IP 테이블에서 해당 엔트리를 검색하여 삭제 한다.

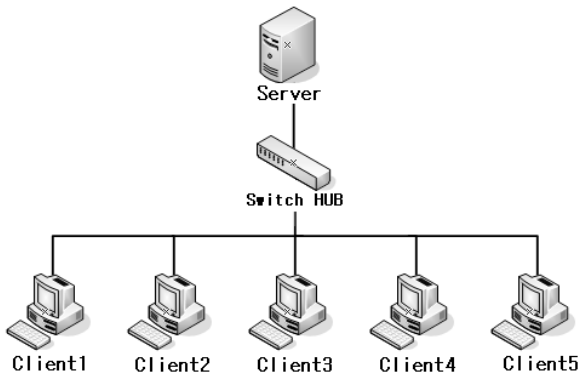
제 4장 실험 및 성능 평가

4.1 실험 방법

본 논문에서 구현된 Percentile 네트워크 스케줄러의 실험 환경과 실험 방법은 다음과 같으며, 이를 통하여 성능을 평가한다. 제안된 Percentile 네트워크 스케줄러의 실험을 위하여 등급을 나누는 기준으로 IP 주소가 사용되었다.

4.1.1 실험 환경

실험 환경은 [그림 14]와 같은 구조의 네트워크 환경으로 이루어졌다.



[그림 14] 실험 네트워크 환경

[그림 14]는 Percentile 네트워크 스케줄러 실험 및 성능평가를 위해 사용된 네트워크 환경이다. Percentile 네트워크 스케줄러를 적용 시킨 서버와, 클라이언트 PC 5대가 사용되었으며, 스위치 허브를 이용한

100Mbps의 네트워크 환경을 지닌 한성대학교 임베디드 시스템 연구실에서 이루어졌다.

서버로 사용된 시스템은 [표 10]와 같다. Percentile 기법을 이용한 스케줄러는 많은 패킷의 양이 수신될 때 성능이 나타나므로 시스템에 충분한 부하를 주기 위하여 낮은 성능의 서버 시스템을 사용하였다. 사용된 서버 시스템은 700Mhz의 CPU와 256Mbyte의 메모리로 충분한 과부하가 발생할 수 있도록 하였다.

표 10 서버 시스템 성능

분 류	Server
CPU	700Mhz
Memory	256Mbyte
Network Card	10/100Mbps NIC
O S	Kernel 2.6.16

클라이언트로 사용된 시스템은 [표 11]과 같다. 클라이언트 시스템은 모두 서버 시스템에서 사용되는 운영체제와 같은 운영체제를 사용하였다. 또한 버전 역시 같은 버전으로 사용하였다.

Percentile 네트워크 스케줄러 성능 평가에 있어 Percentile 등급은 총 다섯 개로 나누었으며 각 등급 별로 하나씩의 클라이언트가 할당 된다. 등급은 숫자가 낮을수록 높은 등급이 된다. 즉 0번 등급인 클라이언트가 가장 높은 등급의 클라이언트가 된다.

표 11 클라이언트 시스템 성능

분 류	Client1	Client2	Clinet3	Client4	Client5
Class	Class 0	Class 1	Class 2	Class 3	Class 4
CPU	2.8Ghz	1.7Ghz	1.5Ghz	2.4Ghz	2.8Ghz
Memory	1Gbyte	512Mbyte	512Mbyte	256Mbyte	512Mbyte
Network Card	10/100Mbps NIC	10/100Mbps NIC	10/100Mbps NIC	10/100Mbps NIC	10/100Mbps NIC
O S	Kernel 2.6.16	Kernel 2.6.16	Kernel 2.6.16	Kernel 2.6.16	Kernel 2.6.16

클라이언트 PC들은 서버 시스템과 같은 운영체제인 리눅스 커널 2.6.16 이며 서버 시스템으로 부하를 발생시키게 된다.

4.1.2 관찰 대상

실험에 대한 관찰 대상으로 각 클라이언트의 응답 시간을 측정하였다. 이것은 서버로 요청을 보낸 시간과 응답을 받은 시간의 차이로 계산한다. 대상 서버 프로그램으로는 널리 사용되는 웹 서버 프로그램인 아파치 [15]가 사용되었으며, 클라이언트는 *index.html* 페이지를 요청하는 내용을 담은 패킷을 전송하게 된다.

클라이언트에서는 요청 패킷이 클라이언트에서 송신된 때부터 서버로부터 다시 수신될 때까지의 경과 시간을 측정한다. 이러한 동작을 하기 위하여 패킷의 생성과 경과 시간의 측정은 별도의 프로그램을 필요로 한다. Percentile 네트워크 스케줄러를 실험하기 위하여 패킷 생성기와 시간

측정 프로그램을 구현하였다.

패킷 생성 프로그램은 소켓을 이용한 네트워크 프로그램이다. 소켓을 이용하여 세션을 연결하고 작은 사이즈의 *index.html* 파일을 요청하는 내용을 담은 버퍼를 *send* 함수를 통하여 송신하게 되며, *recv* 함수를 통하여 수신하게 된다.

패킷 생성 프로그램은 각 클라이언트별로 동작하고 클라이언트의 IP 주소를 이용하여 패킷을 생성하며, 서버 시스템으로 패킷을 전송하게 된다. 패킷 생성 프로그램은 하나의 프로세스로 동작할 경우 다수의 패킷을 동시에 전송하지 못한다. 따라서 패킷 생성 프로그램은 사용자에게 입력받은 수의 쓰레드를 생성하여 지정한 요청의 수만큼 패킷을 생성하여 서버 측으로 전송하게 된다.

시간 측정을 위한 프로그램으로 TsMon 드라이버가 사용되었다[16]. 펜티엄 프로세서의 개발 이후 인텔 프로세서들은 좀 더 짧은 간격의 정밀한 시간 측정을 하기 위해 Time Stamp Counter 레지스터를 제공한다. Time Stamp Counter 레지스터는 64-bit의 MSR (model specific register)이며, 프로세서에서 일어나는 매 사이클의 정밀한 수를 유지하고 있다. 이 레지스터에 대한 프로그래머의 접근을 허용하기 위해 인텔은 RDTSC (Read Time-Stamp Counter) 명령을 제공하고 TsMon 드라이버는 이 명령을 이용하여 측정된 시간을 커널 내부 메모리에 기록한다. TsMon 드라이버가 메모리에 Time Stamp Counter 값을 기록하는데 걸리는 시간은 작은 범위에서 일정하며, 작은 오버헤드를 가지므로 패킷 생성 프로그램에서 사용하기에 충분하다.

```
nr_thread = 4;
total_packet_num = 15000;

generated_packet() {
        packet_num = 0;
        sock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
        connect(sock, server_address);

        while (1) {
                request_tsc = tsGetTSC();              /* Get TimeStampCounter */

                send(sock, request_msg);               /* Send request */
                recv(sock, response_msg);              /* Receive request */

                response_tsc = tsGetTSC();             /* Get TimeStampCounter */

                /* Write response TimeStampCounter in memory */
                tsUsrStamp (response_tsc - request_tsc);

                if (packet_num == total_packet_num) {
                        break;
                }
                packet_num ++;
        }
}
```

[그림 15] 패킷 생성기 pseudo 코드

[그림 15]는 패킷 생성기의 pseudo 코드를 나타낸다. *main* 함수는 사용자에게서 생성할 쓰레드의 수와 전송할 패킷의 수를 받아 쓰레드를 생성한다. *pthread_create* 함수를 이용하여, *generated_packed* 함수를 생성할 쓰레드 수만큼 생성하며, 사용자의 입력이 들어오는 순간 모든 쓰레드는 전송할 패킷의 수만큼 패킷을 전송하게 된다.

패킷 생성기는 TsMon 드라이버 API인 *tsGetTSC* 함수와 *tsUsrStamp* 함수를 이용하여 Time Stamp Counter 값을 기록한다. 요청 메시지를 보내는 함수인 *send* 직전에 *tsGetTSC* 함수를 호출하여 요청 메시지를 전송한 시간을 가져오고 응답 메시지가 수신되면 다시 *tsGetTSC* 함수를 호출하여 응답 메시지를 받은 시간을 가져와 차이 값을 이용하여 응답시간을 구한 후 *tsUsrStamp* 함수를 이용하여 메모리에 기록한다.

4.1.3 실험 방법

실험 방법으로 5대의 클라이언트는 *n*개의 쓰레드를 생성하여 *m*개의 패킷을 생성하여 서버 시스템으로 송신한다. 실험에서 서버는 들어오는 페이지 요구를 다섯 개의 클라이언트로 분류하고 각 클래스에 대하여 50%에서 90%까지 10% 단위로 Percentile 값을 할당한다.

표 12 등급별 Percentile 값

Class	Percentile value
Class0	90
Class1	80
Class2	70
Class3	60
Class4	50

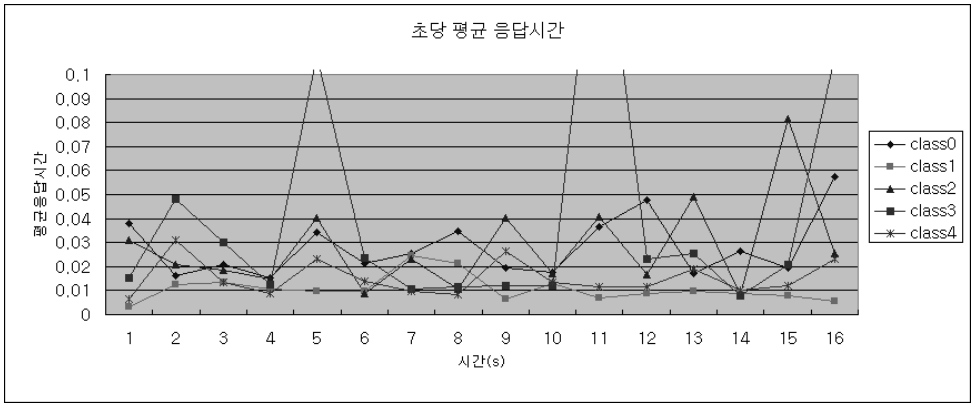
[표 12]은 각 클래스별 Percentile 값을 나타낸다. Percentile 값은 처리 비율을 결정하며 높은 값을 가지는 클래스는 전체적으로 먼저 처리가 된다. 0번 클래스는 90의 Percentile 값을 지니며 등급이 낮아질수록 낮은 Percentile 값을 지닌다.

3장에서 패킷의 성공 실패 여부 판단으로 사용된 기준 값은 패킷 Percentile 스케줄이 시작된 시간과 처리 시간의 차, 즉 응답 패킷 생성에 걸린 시간의 평균값을 사용한다고 하였다. 평균값을 구하기 위해 패킷 생성기를 이용하여 패킷을 발생 시키고 TsMon 을 이용하여 평균 응답 시간을 측정하였다. 기준으로 사용된 평균값으로 약 0.004초 의 값을 사용하였다.

4.2 실험 결과 및 분석

4.2.1 기존 커널에서의 응답시간

4.1 장에서 언급한 실험 환경에서 기존의 커널의 네트워크 처리를 실험하였다. 각 클라이언트는 해당 IP를 이용하여 Percentile 스케줄러를 적용하지 않은 원래의 커널 시스템으로 패킷을 전송한다. [그림 16]은 기존 시스템에서 아파치 서버의 수정 없이 각 클라이언트별 응답 시간을 그래프로 나타낸 것이다.

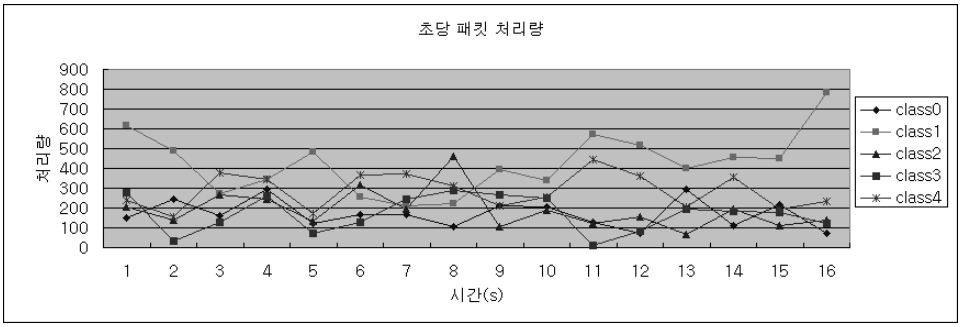


[그림 16] 기존 시스템의 클래스별 초당 평균 응답시간

[그림 16]은 다섯 대의 클라이언트를 이용하여 Percentile 스케줄러가 적용되지 않은 시스템으로 패킷을 전송하였을 경우 응답시간을 나타낸다. [그림 16]의 그래프는 앞뒤 시간을 제외한 중간 결과 그래프이다. 클라이언트에서 측정되는 시간은 TCP 세션 연결 작업인 3-handshake 와 SYNC, ACK 패킷을 제외한 실제 데이터를 포함한 패킷의 응답 시간이 된다. 즉 *send* 함수 직전의 시간과 *recv* 함수 직후의 시간의 차를 계산하여 기록된 응답시간을 그래프로 표현한 것이다. 실험을 위하여 각 클라이

언트는 4개의 쓰레드를 생성하고 쓰레드 별로 15000개의 패킷을 전송하였으며, 이에 따른 초당 응답시간을 표현하였다.

그래프에서 나타나듯이 각 클라이언트의 평균 응답 시간은 불규칙적인 모습을 보인다. 3번 등급의 클라이언트 4번의 응답 시간이 다른 클라이언트에 비하여 높은 것을 발견할 수 있으며, 다른 등급의 클라이언트들은 비슷한 응답 시간을 보임을 알 수 있다. [그림 17]은 서버 측에서 측정된 각 클래스별 패킷 처리량의 그래프이다.



[그림 17] 기존 시스템의 클래스별 초당 패킷 처리량

[그림 17]은 [그림 16]과 동일한 시간대에서의 서버 측에서 클라이언트별 패킷 처리량을 나타낸다. [그림 16]에서 높은 응답 시간을 보인 클래스 3번의 패킷 처리량은 다른 클래스에 비하여 100 이하의 낮은 패킷 처리량을 보인다. 반대로 [그림 16]에서 빠른 응답 시간을 보이는 1번 클래스의 경우, [그림 17]의 같은 시간대에서 패킷 처리량을 볼 때 높은 처리량을 나타냄을 알 수 있다. 결과 그래프를 통하여 원래의 시스템에서의 동작 형태를 파악할 수 있으며, 이것은 서버에 수신된 패킷의 양과 연관지을 수 있다. 기존의 서버 시스템이 한쪽 클라이언트에서 많은 패킷을 수신할 경우 해당 등급의 패킷 처리에 많은 시간이 소요됨을 알 수 있다. 즉 네트워크에서 발생하는 버스트 현상에 의하여 일정 클라이언트에서 많은

부하가 발생할 경우, 다른 클라이언트에서 대역폭을 차지하지 못할 때 이러한 현상이 발생하게 된다. 이것은 기존 네트워크의 동작이 어느 한쪽의 부하 발생에 따라 조절해주는 역할을 하지 못함을 보여준다.

표 13 기존 시스템의 클래스별 평균 응답시간

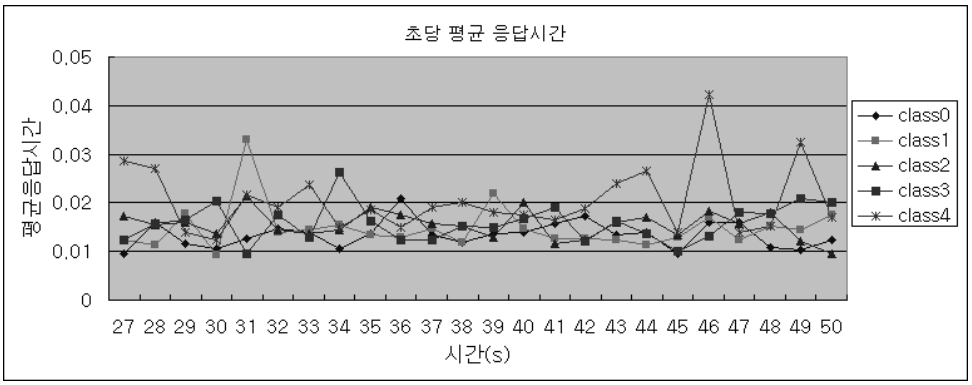
	Class0	Class1	Class2	Class3	Class4
평균 응답 시간	0.02395	0.01004	0.01819	0.02596	0.01270

[표 13]은 기존 시스템에서의 등급별 평균 응답 시간을 보여준다. 기존 시스템에서의 등급별 응답시간을 측정한 결과 우선순위에 따른 차등 서비스가 제공이 되지 않음을 확인할 수 있다.

기존 시스템의 응답 시간 측정 실험을 통하여 각 클라이언트의 응답 시간은 시스템 성능에 따른 변화는 보이지 않았으며, 서버 측의 패킷 수신량에 따라 응답시간이 달라짐을 확인 할 수 있었다.

4.2.2 제안된 시스템에서의 응답 시간

제안된 시스템에서의 응답 시간 측정은 기존 시스템에서의 실험과 같은 환경에서 동일한 방법으로 진행되었다. [그림 18]은 제안된 시스템에서 클라이언트별 초당 평균 응답 시간을 나타낸 그래프이다.



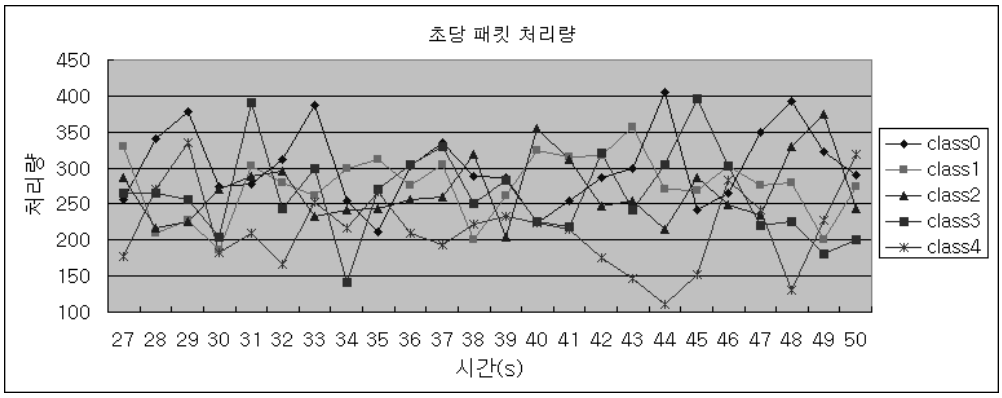
[그림 18] Percentile 스케줄러에 의한 클래스별 초당 평균 응답시간

전체적으로 기존 시스템에 비하여 큰 응답 시간을 보이는 현상은 일어나지 않았음을 확인 할 수 있다. 이것은 Percentile 기법의 특성상 하나의 패킷이라고 있을 경우 우선순위에 기반하여 스케줄링 되기 때문에 많은 양의 패킷이 수신된 클래스가 존재하더라도 스케줄러에 의하여 선택될 수 있음을 의미한다.

전반적으로 그래프에서는 높은 등급을 지니는 클라이언트에서 수신된 패킷이 우선 처리되기 때문에 타 클래스에 비하여 빠른 응답 시간을 보인다. 즉 높은 Percentile 값을 지니는 Class0 번의 클라이언트의 응답 시간은 다른 타 클래스의 클라이언트들의 응답 시간에 비하여 빠른 응답 시간을 지니게 되며 그래프는 전반적으로 아래에 분포함을 보인다.

또한 그래프 중간에 클래스들의 역전 현상을 발견할 수 있다.

Class0 의 클라이언트의 응답 시간이 Class1 의 클라이언트의 응답 시간보다 높게 측정되는 구간은 Percentile 기법에 의하여 발생될 수 있는 역전 현상이 일어남을 의미한다. Percentile 기법은 마감시간 내에 처리된 패킷의 수가 높을수록 우선순위가 낮아짐으로 다른 등급의 클래스가 선택될 수 있음을 의미한다.



[그림 19] Percentile 스케줄러에 의한 클래스별 초당 패킷 처리량

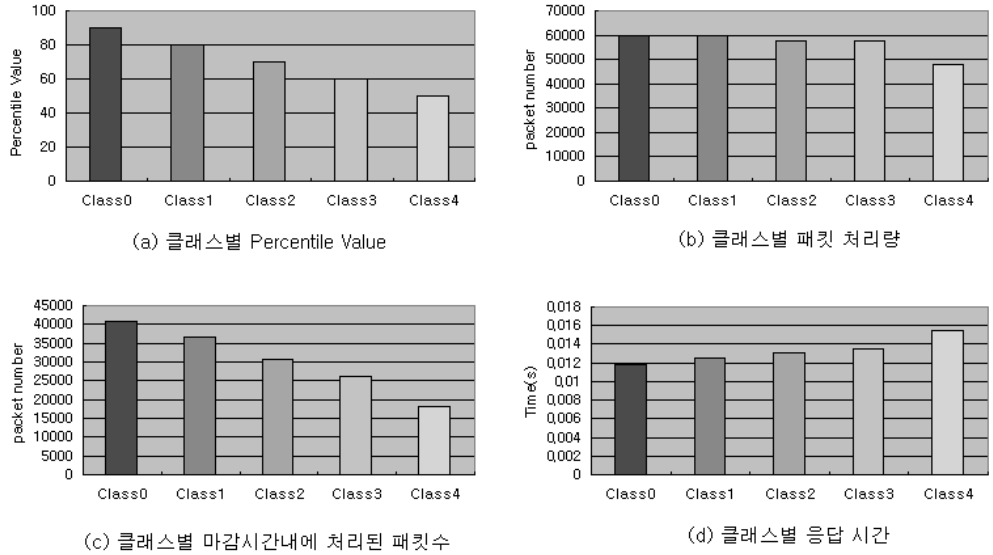
[그림 19]는 [그림 18]과 동일한 시간대의 클래스별 초당 패킷 처리량을 보여준다. 서버에 특정 클래스의 수신된 패킷이 많을 경우 Percentile 기법에 의하여 우선순위가 높아져 빠른 응답 시간을 지니므로 결국 많은 양의 패킷이 처리됨을 확인 할 수 있다.

[표 14]는 서버가 하나의 클래스에서 6만개 패킷을 처리하였을 때, 실험 종료 후 정보를 보여준다.

표 14 Percentile 네트워크 스케줄러 최종 정보

	Class0	Class1	Class2	Class3	Class4
Total Input Packet	60000	60000	57551	57350	48139
Packet number within deadline	40756	36512	30658	26172	18304
Percentile Value	90	80	70	60	50
Priority	1324	1314	1314	1314	1315
Response Time	0.011851	0.012481	0.013106	0.013465	0.015447

[그림 20]은 [표 14]의 정보를 그래프로 표현한 것이다.



[그림 20] Percentile 네트워크 스케줄러 최종 정보 그래프

서버에서 처리된 패킷의 양은 Class0이 가장 많으며 Class4의 패킷이 가장 적게 처리 되었다. 이것은 Percentile 스케줄러에 의하여 높은 등

급을 지닌 클라이언트의 패킷을 우선 처리됨을 나타내며, 마감 시간 내에 처리된 패킷의 경우 등급에 따라 높은 등급의 패킷이 우선 처리됨으로 성공적으로 처리한 패킷양이 많은 것을 확인할 수 있다. 각 클라이언트의 평균 응답 시간은 Class0이 가장 빨랐으며, 등급 순으로 점차 높아짐을 확인하였다.

제 5장 결론 및 향후 연구

본 논문에서는 Percentile 기법을 이용한 네트워크 스케줄러를 설계하고 구현하였다. 기존의 차등 서비스를 위한 연구는 네트워크 처리에 대한 고려가 이루어 지지 않았고 응용 프로그램 레벨에서 구현이 되어 많은 오버헤드가 발생되었다. 또한 IP 라우터에서 사용된 스케줄러는 각 서비스 단위로 클래스를 두어 차등 서비스를 구현하였기 때문에 본 논문에서 제안하는 고객별로 차등 서비스를 하기 위한 스케줄러로는 적합하지 않다.

구현된 Percentile 네트워크 스케줄러는 하나의 서비스에서 각 클라이언트의 차등 서비스를 제공하였다. Percentile 네트워크 스케줄러는 패킷을 단위로 스케줄링 하여 차등 서비스를 구현한 것으로, 스케줄에 사용된 우선순위 결정 방법으로 Percentile 기법이 사용되었다. Percentile 기법은 서로 다른 클래스의 우선순위에 따른 계산법으로 클래스들은 각각 Percentile 값에 따라 처리 비율이 결정이 되며, 성공 실패의 비율에 따라 우선순위가 역전되어 낮은 등급의 클래스의 기아 현상(Starvation)을 막을 수 있는 기법이다.

서버로 수신된 패킷은 IP 프로토콜 레이어에서 IP 주소를 기준으로 등급이 결정되며 각 등급 큐에 삽입된다. TASKLET 으로 동작하는 Percentile 스케줄러는 각 등급의 우선순위를 구하여 가장 높은 우선순위의 등급 큐에서 먼저 수신된 패킷을 상위 프로토콜 레이어인 TCP 프로토콜 레이어로 보내고 최종적으로 응용프로그램에 의하여 응답 패킷을 생성하게 된다.

4장의 실험에서 Percentile 네트워크 스케줄러와 기존의 커널 시스템을

이용하여 실험을 하였다. 기존의 커널 시스템은 FIFO 방식으로 동작하기 때문에 먼저 수신된 패킷이 먼저 처리 되는 동작을 하게 된다. 두 시스템을 비교하기 위하여 사용된 서버 프로그램으로는 아파치를 사용하였으며 실험을 위한 패킷 생성기와 시간 측정 프로그램인 TsMon을 이용하였다. 기존 시스템에서 각 클라이언트의 응답시간의 측정 결과 패킷의 과부하를 발생시킨 클라이언트에 대하여 우선 처리가 보였으며, 비슷한 패킷의 수신량을 보인 클라이언트들에 대한 처리는 수신된 순서로 이루어짐을 확인할 수 있었다. 제안된 시스템에서의 각 클라이언트의 응답시간은 스케줄 루틴에 의하여 약간의 처리 시간이 걸림을 확인 할 수 있었지만 적은 시간이 소비 되었으며, 각 등급에 따른 서비스를 제공하려고 하는 모습을 확인할 수 있었다.

실험 및 성능 평가를 통하여 Percentile 기법을 이용한 네트워크 스케줄러가 정상적으로 작동되는 것을 확인하였고 기존의 FIFO 방식으로 동작하는 리눅스 네트워크 처리보다 클래스 별로 차등 서비스가 이루어지는 것을 확인할 수 있었다.

본 논문에 이은 향후 연구로 커널의 프로세스 스케줄러에 Percentile 기법을 적용하여 전체 처리 과정에 대해 차등 서비스를 이루는 연구가 필요하다. 시스템 전체적으로 요청에 대한 응답의 순서를 제어하는 부분은 네트워크 프로세싱과 프로세스 프로세싱으로 분류 할 수 있다. 본 논문은 기존의 네트워크 처리에 Percentile 기법을 적용한 스케줄러를 제안하였다. 따라서 제안된 기법을 이용한 시스템에서는 커널 프로세스 스케줄러에 의하여 우선순위 변화에 따라 좀 더 무딘 결과가 나타날 수 있다. 시스템 전반에 걸쳐 Percentile 기법을 적용을 하게 된다면 정확한 등급에 따른 차등 서비스가 가능할 것이라고 생각된다.

참고문헌

[1] Crovella M. E, Frangioso R, Harchol-Balter M, "Connection Scheduling in Web Servers", Proceedings of the 1999 USENIX Symposium on Internet Technologies and System, pp.243-254, October 1999

[2] Linux Kernel Man Page 8 Priority Queueing , /usr/share/man/man8/tc-prio.8.gz

[3] Floyd Sally, Jacobson Van, "Link-sharing and Resource Management Models for Packet Networks", IEEE/ACM Transactions on Networking Vol3 No4, 1995

[4] Demers A, Keshav S, Shenker S, "Analysis and simulation fair queueing algorithm", Internet Res. and Exper., vol.1, 1990

[5] Agarwal N, "Percentile Goal As A Perfoemance Criterion In A Multiclass GI/G/G1 Queueing System", Ph.D. thesis, North Carolina State University, Raleigh, NC, USA, 1995

[6] Bovet Daniel Pierre, "Understanding the Linux Kernel, 2nd Edition", OReilly, 2002

[7] 유영창, "리눅스 디바이스 드라이버", 한빛미디어, 2004

[8] http://www.cyberus.ca/~hadi/usenix-paper.tgz

[9] Floyd Sally, "Notes On CBQ and Guaranteed Service", 1995

[10] 문준현, "인터넷 QoS 제공을 위한 CBQ 기반의 스케쥴링 기법에 관한 연구", 광주대 경상대학원 석사 학위 논문, 2001

[11] Bennett J.C.R, Zhang H., "WF2Q: Worst-case fair weighted fair queueing", In Proceeding of IEEE INFOCOM'96. pp 120-128, SanFrancisco, CA, March 1996

[12] Harchol-Balter Mor, Bansal Nikhil, Schroeder Bianca, Agrawal

Mukesh, "SRPT Scheduling for Web Servers", Lecture Notes in Computer Science, Vol. 2221, pp. 11-20, 2001

[13] 방지호, 하란, "응답시간 향상을 위한 커넥션 스케줄링 기법", 한국정보과학회, 정보과학회 논문지 제33권, 제1,2호, pp. 69-78. 2006.2

[14] 최동준, "리눅스에서의 Percentile 스케쥴 기법 구현", 한성대학교 대학원 석사 학위 논문, 2002년

[15] The Apache Group, Apache web server. http://www.apache.org

[16] 윤찬희, 송진석, 김다현, 이민석 "시스템 시간 측정을 위한 TsMon 드라이버", 한성대학교 공학연구 논문집 Vol.4 No.2 pp.59-66, August 2006

ABSTRACT

Design and Implementation of Network Scheduler for Differentiated Service

Yun, Chanhee
Major in Computer Engineering
Dept. of Computer Engineering
Graduate School
Hansung University

These day's internet services classify customers into several class based on the expenses customers paid, and offer different services to the classified customers. However, the differences are only in the features provided, not in the network response time.

In this paper, we propose a percentile based network packet scheduling mechanism, which provides the better response time to the highly classified customers. And we describe the implementation of proposed mechanism on Linux kernel.

Percentile scheduler determines the priority based on percentile value of the given classes of requests. The high priority packets are delivered to the server program on ahead, so they show the faster response times than low priority packets.

In order to assess the efficiency of the Percentile network scheduler, We compared the response time of the proposed packet scheduling system with original Linux network system, using the Apache web server which is widely used. With this experiment, we found that the proposed percentile network scheduler clearly provides the differential service based on the given class.