차별화서비스를 제공하는 IP네트워크에서 대역폭관리를 위한 트래픽 제어시스템

정회원 이 명 섭*, 박 창 현*

A traffic control system to manage bandwidth usage in IP networks supporting Differentiated Service

Myung Sub Lee*, Chang Hyeon Park* *Regular Members*

요 약

최근 인터넷 기술이 급속도로 발전하고 인터넷상에서의 멀티미디어 통신이 일반화되면서 네트워크 트래픽이 급속히 증가되고 있으며, 이러한 트래픽 급증으로 인하여 네트워크 회선 부족 및 통신 서비스의 품질 저하 등의 문제점들이 발생하고 있다. 본 논문에서는 이러한 문제점을 해결하기 위해서 IP네트워크 환경에서 선별적인 QoS를 보장하는 DiffServ지원 라우터와 DiffServ 네트워크상의 트래픽 흐름을 제어함으로써 클라이언트의 대역폭 요구에 대한 동적 자원할당을 수행하는 트래픽 제어 에이전트를 제시한다. 본 논문에서는 DiffServ지원 라우터 상에서 트래픽의 선별적인 전송방식을 구현하기 위해 리눅스의 큐 관리정책을 조작하였으며, 이러한 DiffServ지원 라우터를 효율적으로 제어할 수 있는 트래픽 제어 에이전트를 제안한다. 본 논문에서 제시하는 트래픽 제어 에이전트는 클라이언트의 서비스 요청에 대한 자원할당기능을 수행하며, 네트워크 상태변화에 따라 동적으로 자원을 재 할당한다. 특히, DiffServ 지원 라우터에서는 비동기 큐잉과 클래스별 큐잉기능을 제공하는 ACWF^{2}Q$^+$(Asynchronous and Class based WF^{2}Q$^+$) 패킷 스케줄러를 제안하여 AF PHB의 처리율과 공평성(fairness)을 향상시킨다.

Key Words : DiffServ, Traffic control agent, ACWF^{2}Q$^+$

ABSTRACT

As the recent rapid development of internet technology and the wide spread of multimedia communication, massive increase of network traffic causes some problems such as the lack of network paths and the bad quality of service. To resolve these problems, this paper presents a traffic control agent that can perform the dynamic resource allocation by controlling traffic flows on a DiffServ network. In addition, this paper presents a router that can support DiffServ on Linux to support selective QoS in IP network environment. To implement a method for selective traffic transmission based on priority on a DiffServ router, this paper changes the queuing discipline in Linux, and presents the traffic control agent so that it can efficiently control routers, efficiently allocates network resources according to service requests, and relocate resources in response to state changes of the network. Particularly for the efficient processing of Assured Forwarding(AF) Per Hop Behavior(PHB), this paper proposes an ACWF^{2}Q$^+$ packet scheduler on a DiffServ router to enhance the throughput of packet transmission and the fairness of traffic services.

* 영남대학교 컴퓨터공학과 인공지능 및 지능정보시스템 연구실(skydream@cse.yu.ac.kr)
논문번호 : 030427-1006, 접수일자 : 2003년 10월 06일

Ⅰ. 서 론

실시간 응용 서비스가 요구하는 QoS를 지원하기 위해 새로운 서비스 모델에 기반을 둔 IP패킷 전달 방식에 대한 연구가 최근 IETF(Internet Engineering Task Force) WG(Working Group)에서 연구되고 있으며, 대표적인 서비스가 IntServ(Integrated Services)[1]와 DiffServ (Differentiated Services)[2]이다. IntServ모델은 단대단 패킷 전송지연시간을 극복할 수 있도록 자원 예약을 위한 RSVP (Resource reSerVation Protocol) 신호프로토콜을 사용하여 각 흐름별로 자원을 예약한 후 패킷을 전송하도록 하는 방식이다. 이 모델은 흐름별로 각기 다른 QoS가 요구되므로, 각 서비스 유형에서 요구되는 흐름명세(flow spec) 파라메타를 포함하고 있다. 따라서 실시간 응용서비스가 요구하는 전송품질을 보장하기 위해 RSVP는 흐름명세 파라메타를 이용하여 종단 호스트와 망 노드 사이에 필요한 대역폭(bandwidth)을 요청한다. 대역폭 요청을 받은 망 노드는 호스트가 요청한 파라메타와 현재 네트워크의 상태에 따라 서비스를 제공할 것인지 거부할 것인지를 결정하게 된다. 그리고 최종적으로 링크 계층에서는 패킷 스케줄링과 패킷 분류(classification) 기능을 수행한 후 요청 서비스를 처리한다.

그러나, IntServ모델을 지원하기 위해서는 각 중간라우터에서 흐름에 관련된 정보를 저장해야하기 때문에, 라우터에서 이런 정보를 저장하는 공간을 요구할 뿐만 아니라 라우터의 처리 오버헤드가 커질 수 있다. 특히, 인터넷 백본라우터의 경우 전송속도가 상당히 높고 연결된 흐름의 개수가 매우 많으므로 코어노드에서 IntServ모델을 지원하기가 매우 힘들다[3-5]. 따라서 확장성 문제를 갖고 있는 IntServ모델의 한계를 극복하고 인터넷 백본망에서 적용할 수 있는 새로운 서비스모델로 DiffServ모델의 구조 및 관련 표준안이 개발되고 있다.

DiffServ모델에서는 우선적으로 QoS를 몇 개의 클래스로 분류하고 분류된 클래스에 따라 서비스의 전송품질을 보장하도록 한다. 이에 따라 DiffServ 모델은 IP헤더의 특정 필드(IPv4 : ToS Field, IPv6 : Traffic Class Field)에 QoS정보를 마킹(marking)하여 차별화서비스를 설정하게 되며, 이렇게 설정된 QoS정보들에 의해 적절한 전송(PHB : per hop behavior)[6]을 수행하는 구조를 가진다. DiffServ모델에서는 차별화서비스 도메인의 경계라우터에서 IP헤더의 ToS필드에 DSCP를 설정하여 트래픽 흐름을 분류한다. 그리고 네트워크 내에서는 분류된 트래픽 흐름에 따라 자원할당, 패킷 폴리싱, 스케줄링 등을 수행하도록 한다. 즉, DiffServ 모델에서는 경계노드와 코어노드의 역할을 분리시켜 IntServ 구조의 확장성 문제를 해결하고자 하는 것이다.

DiffServ모델에서는 클라이언트에서 요구되는 자원과 망 노드에 설정된 SLA(Service Level Agreement)[7] 정보를 바탕으로 자원할당방법을 결정하는 장치가 필요하며, 이러한 기능을 하는 장치를 BB(Bandwidth Broker)[8]라 한다. BB는 SLA정보에 따라 할당된 대역폭을 저장하는 데이터베이스를 유지하고 있으며, 데이터베이스에 저장된 정보는 향후 대역폭 할당을 결정하기 위한 기초정보로 이용된다.

IntServ모델의 문제점을 극복하기 위해 제안된 DiffServ모델은 현재 이의 구현을 위한 연구개발이 진행 중에 있다. 그러나 제안된 모델은 현재의 네트워크에 적용(deployment)되지 않고 있다. 그 이유를 제안된 DiffServ구조 및 지금까지 진행된 연구결과의 분석을 통해 문제점을 검토한다[9-11]. 첫째, IntServ모델은 각 흐름별로 QoS를 보장하도록 하는 구조인 반면, DiffServ는 흐름별로 QoS를 지원하지 못하고, 보장하여야 할 서비스들을 클래스별로 구분하여 서비스함으로써 완벽한 QoS를 보장하지 못한다. 이러한 DiffServ모델은 코어 네트워크에서 확장성 문제를 해결할 수 있어 실제 네트워크에 적용에 있어서는 IntServ모델보다는 향상된 모델이지만, 각 흐름별 QoS보장기능을 지원하지 못하기 때문에 완벽한 QoS를 제공할 수 없다는 문제점이 있다. 둘째, 지금까지 제안된 DiffServ모델에서는 신호프로토콜의 사용을 언급하고 있지 않다. 즉, DiffServ에서는 IP헤더의 ToS필드에 QoS와 관련된 클래스정보를 추가하여 이 정보를 바탕으로 각각 PHB프로세스를 처리하기 때문에 별도의 신호프로토콜의 필요성이 제기되지 않았다. 그러나 DiffServ모델에서 동적 SLA를 지원하기 위해서는 자원예약을 위한 신호프로토콜은 필요 없다할지라도 차별화서비스 네트워크내의 BB와 경계라우터 간 또는 호스트와 차별화서비스 네트워크의 경계라우터 간 그리고 BB와 BB간의 통신을 위한 신호프로토콜의 사용은 반드시 필요하다. 셋째, DiffServ모델에서는 동적 SLA체결을 위해 BB의 역할이 매우 중요하다. 그러나 IETF DiffServ Spec.에서는 아

직 BB의 역할을 명확하게 정의하고 있지 않다. 따라서 DiffServ를 위한 BB의 역할, 기능 등의 요구사항이 명확하게 정의될 필요가 있다. 본 논문에서는 이상에서 언급한 DiffServ의 문제점을 해결하기 위해 IP네트워크 환경에서 선별적인 QoS를 보장하는 DiffServ지원 라우터와 클라이언트의 대역폭 요구에 대한 동적 자원할당을 수행하는 트래픽 제어 에이전트를 제시한다.

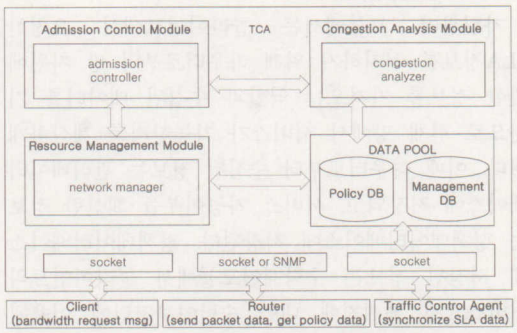
본 논문에서 제시하는 트래픽 제어 에이전트는 망 노드의 트래픽 정보를 수집하고 분석하여 동적 자원할당 기능을 수행하며, 이러한 동적 자원할당을 위해 트래픽 제어 에이전트와 클라이언트, 트래픽 제어 에이전트와 라우터 그리고 트래픽 제어 에이전트와 트래픽 제어 에이전트간의 신호체계를 정의한다. 또한, DiffServ지원 라우터에서는 비동기 큐잉과 클래스별 큐잉기능을 제공하는 ACWF^{2}Q$^+$ (Asynchronous and Class based WF^{2}Q$^+$) 패킷 스케줄러를 제안하여 AF PHB의 처리율(throughput)과 공평성(fairness)을 향상시키고, EF 마이크로흐름의 흐름별 QoS보장 기능을 제공한다.

본 논문의 구성은 다음과 같다. 2장에서는 본 논문에서 제안한 트래픽 제어 에이전트의 설계방법에 대해 설명하고 3장에서는 DiffServ지원 라우터의 설계방법에 대해 설명한다. 4장에서는 실측실험을 통하여 본 논문에서 개발한 시스템의 성능평가를 제시한다. 마지막으로 5장에서 결론을 맺고, 앞으로의 연구방향에 대해 기술한다.

Ⅱ. 트래픽 제어 에이전트

본 논문에서의 트래픽 제어 에이전트는 클라이언트에서 요구되는 자원과 설정된 SLA(Service Level Agreement)정보를 바탕으로 동적 자원관리를 수행한다. 동적 자원관리를 수행하기 위해 트래픽 제어 에이전트는 SLA별로 할당된 대역폭을 저장하는 데이터베이스를 가지고 있으며, 향후 자원할당을 결정하기 위한 기초 자료로 사용한다. 클라이언트에서 차별화서비스 네트워크의 경계라우터로 서비스를 요청하면, 경계라우터는 서비스의 유형, 목표 전송률, 최대 버스트 크기, 서비스요구시간 등의 SLA정보를 기반으로 해당 트래픽 제어 에이전트로 서비스 요청 정보를 보내게 된다.

트래픽 제어 에이전트가 서비스요청 정보를 받으면 요구대역폭을 할당할 수 있는지를 검사하여, 가능하면 경계라우터로 승낙 정보를 전송하게 된다. 즉, 트래픽 제어 에이전트는 차별화서비스 네트워크 전체의 자원을 관리하며, 또한 차별화서비스 네트워크에서의 정책에 따라 수락제어를 결정한다. 이를 바탕으로 내부 라우터들과의 통신 그리고 인접 차별화서비스 네트워크 사이의 협상기능을 수행하게 된다. 이러한 기능을 수행하기 위하여 본 논문에서의 트래픽 제어 에이전트는 2 계층(two-tier) 구조를 가지게 된다. 첫 번째 계층은 차별화서비스 도메인 내부의 자원 할당을 다루는 도메인 내부(Inter-domain) 자원관리구조로, 도메인 내부에서 트래픽 제어 에이전트와 라우터들 간의 통신을 통해 자원관리를 하게 된다. 두 번째 계층은 두 도메인 간(Inter-domain) 네트워크 경계에서 자원을 공급하고 할당하는 도메인 사이의 자원관리구조이다. 이를 위해 두 도메인간의 트래픽량과 트래픽유형 등의 정보를 트래픽 제어 에이전트 간 수락제어를 통해 교환한다.



[그림 5] 흐름 관리를 위한 트래픽 제어 에이전트 구성도

본 논문에서 제시하는 트래픽 제어 에이전트는 [그림 1]에 보이는 바와 같이 수락 제어 모듈(Admission Control Module), 자원관리 모듈(Resource Management Module), 혼잡 분석 모듈(Congestion Analysis Module)로 구성되며 각 모듈들에대한 설명은 세부 절로 구성한다.

1) 수락제어모듈

수락제어모듈은 링크 상에 흐름의 예약률 합이 링크의 용량을 초과하지 않도록 새로운 요청의 수락여부를 결정하는 모듈이다. 수락제어모듈은 정책 데이터베이스(Policy DB)를 참조하여 SLA협상을 정적으로 수행할 수도 있으며, 자원관리모듈을 연계하여 동적으로 수행할 수도 있다. 수락제어모듈의

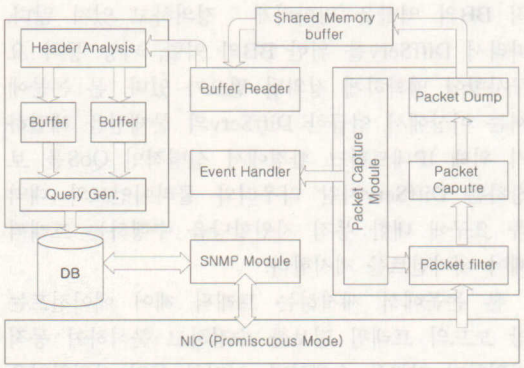
초기정책설정은 네트워크 관리자에 의해 수행되며, 설정된 정책은 명령어처리기를 통해 정책데이터베이스에 저장된다. 수락제어기는 정책데이터베이스를 참조하여 클라이언트의 요청에 대한 SLA를 수행한 후 서비스 수락여부를 결정하게 된다.

서비스 수락이 결정되면 트래픽 제어 에이전트는 DiffServ라우터내의 흐름관리기와의 통신을 통하여 트래픽 제어를 수행한다. 트래픽 제어부 내의 흐름관리기는 동적 필터생성 기능, 큐 관리정책설정, 큐 가중치할당 등의 기능을 수행한다. 동적 필터생성 기능은 경계라우터의 Ingress 인터페이스 부분에서 u32필터를 생성하여 패킷의 미터링(metering) 기능을 수행하며, Egress 인터페이스 부분에서 tc_index필터를 통해 분류된 트래픽의 DSCP마킹을 수행한다. 예약된 자원을 할당하기 위한 큐 관리정책과 가중치설정은 경계라우터와 코어라우터에 동일하게 적용된다.

2) 자원관리모듈

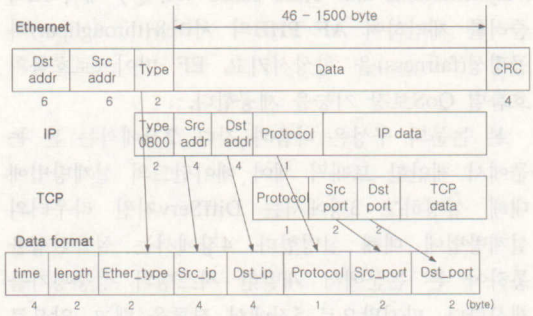
자원관리 모듈에서는 클라이언트에서 요청한 SLA정보를 처리하기 위해 라우터로부터 망 자원에 관한 정보를 가져오는 역할과 수집된 데이터를 기반으로 현재 망에서 서비스가 가능한지를 계산하게 된다. 이때 라우터로부터 수집된 정보는 관리데이터베이스에 저장되고, 서비스 가능여부를 계산한 정보는 정책데이터베이스에 저장된다. 정책데이터베이스에 저장된 정보는 수락제어모듈에서 클라이언트의 서비스요청 메시지에 대한 수락여부판단에 이용된다. 또한 차별화서비스 도메인내의 네트워크이용률을 계산하여 클라이언트와 트래픽 제어 에이전트 간의 SLA를 위한 정보로 이용된다.

본 논문의 자원관리모듈은 [그림 2]와 같이 패킷획득방법과 SNMP모듈을 사용한다. 패킷획득방법은 차별화서비스 네트워크에서 전송되고 있는 트래픽을 서비스단위로 검사하는데 사용되며, LibPcap 라이브러리를 이용하여 구현한다. 자원관리 모듈에서 패킷수집과 데이터베이스저장 부분은 별도의 프로세스로 동작을 하며 프로세스사이의 데이터 공유를 위하여 공유메모리기법을 사용한다. [그림 2]에서 패킷획득방식은 네트워크 디바이스에서 패킷필터링 과정을 거쳐 패킷을 획득하고 일련의 루틴에 따라 헤더를 풀어낸 후 공유메모리버퍼에 저장한다.



[그림 6] 자원관리모듈의 구성도

이때, 데이터베이스 변환모듈에서 이 데이터를 읽어서 데이터베이스 테이블에 맞는 형태로 일련의 변환과정을 거쳐 일정간격으로 저장한다.



[그림 7] 이더넷에서의 패킷획득 형태

[그림 3]은 이더넷 프레임에서 패킷 획득 방식을 사용하여 정보를 추출한 다음 데이터베이스에 저장되는 데이터형태를 설명한다. 데이터형태에서 time은 패킷을 수집해서 데이터베이스에 저장할 때의 시간이며, length는 패킷 전체의 크기에 CRC값(4byte)을 더한 값이다. time과 length를 제외한 나머지 정보는 모두 패킷에서 가져오는데 ether_type은 이더넷 헤더에서 정보를 가져오고, src_ip, dst_ip, protocol 등은 IP프로토콜 헤더에서, src_port, dst_port 등은 TCP/UDP헤더에서 정보를 가져온다. 데이터 포맷의 일정 필드를 사용하지 않는 프로토콜은 0으로 채운다.

SNMP모듈은 차별화서비스 네트워크에서 입출력되는 총 패킷 양을 모니터링하고 네트워크 이용률 및 에러율을 계산하는데 사용된다.

3) 혼잡분석모듈

혼잡분석모듈에서는 자원관리모듈에서 수집된 관리 정보를 기반으로 패킷의 흐름별 전송률, 흐름별 이용률, 흐름별 에러율 등을 검색하여 인증제어 모듈의 인증절차의 기초 자료로 이용된다. 혼잡분석모듈에서 사용된 네트워크의 이용률과 에러율 분석에 필요한 수식을 식 (1)에서 식 (5)까지 보인다. 본 논문에서 제시하는 이용률과 에러율을 분석하는 식에서 사용된 변수를 <표 1>에 정리한다.

<표 3> 식(1)에서 식 (5)까지 사용된 변수

변 수	설 명
x	폴링 주기에서 이전 폴링시간
t	폴링 주기
ifInOctets	인터페이스에 수신된 옥텟의 총 수
ifOutOctets	인터페이스 외부로 전송되는 옥텟의 총 수
sysUpTime	시스템이 마지막으로 재 초기화된 이후의 시간
ifSpeed	인터페이스의 현재 대역폭
ifInError	오류로 상위계층에 전달되지 못한 패킷의 수
totalPktIn	전체 도착 패킷의 수

먼저, FDDI 망에서의 네트워크의 이용률을 측정하기 위해서는 FDDI 네트워크내의 모든 라우터의 입출력 트래픽 양을 측정하여 그들의 합을 2로 나누면 된다. 모든 라우터의 입출력 트래픽 양은 식 (1)과 같이 송신측에서 전송한 패킷의 총 비트 수와 수신측에서 받은 총 비트 수를 합하여 네트워크 링크의 전체대역폭으로 나누면 된다.

FDDI 네트워크의 최대 전송률은 100Mbps이기 때문에 식 (2)와같이 이용률을 측정할 수 있다. 일반적으로 FDDI 네트워크의 이용률이 90% 이상일 때 과부하로 측정할 수 있다. 일시적으로 이용률이 90%를 초과할 때는 큰 문제가 되지 않지만, 평균 이용률이 90%이상일 때는 네트워크 전체 과부할 상태로 판명하여 네트워크를 재 구축하거나 업그레이드가 이뤄져야만 한다. 네트워크 트래픽의 이용률 계산을 위해 혼잡 제어 모듈에서는 각각의 컴포넌트 네트워크에 따라 분석 식을 다르게 사용한다. 기본적으로 이더넷은 브로드캐스팅 전송방식을 사용한다. 따라서 이더넷 네트워크의 이용률을 측정하기 위해서는 모든 입출력 트래픽을 더한 다음, 트래픽을 총 합을 최대대역폭으로 나눠주면 된다.

Input/Output traffic of router :

$$(total\_bit\_sent + total\_bit\_received)/bandwidth. \quad \text{식 (1)}$$

FDDI Utilization :

$$\frac{1}{2}\sum_{devices}\left[\frac{(ifInOctets_{(x+t)} - ifInOctets_{(x)} + ifOutOctets_{(x+t)} - ifOutOctets_{(x)}) \times 8}{(sysUpTime_{(x+t)} - sysUpTime_{x}) \times ifSpeed \times 100}\right]. \quad \text{식 (2)}$$

Ethernet Utilization :

$$\frac{(ifInOctets_{(x+t)} - ifInOctets_{(x)} + ifOutOctets_{(x+t)} - ifOutOctets_{(x)}) \times 8}{(sysUpTime_{(x+t)} - sysUpTime_{x}) \times ifSpeed \times 100}. \quad \text{식 (3)}$$

Full− duplex Serial link Utilization :

$$Max[(total\_bit\_sent + total\_bits\_received)/bandwidth],$$

$$Max\left[\frac{(ifInOctets_{(x+t)} - ifInOctets_{(x)} + ifOutOctets_{(x+t)} - ifOutOctets_{(x)}) \times 8}{(sysUpTime_{(x+t)} - sysUpTime_{x}) \times ifSpeed \times 100}\right]. \quad \text{식 (4)}$$

Traffic error rate :

$$\frac{ifInError_{(x+t)} - ifInError_{(x)}}{totalPktIn_{(x+t)} - totalPktIn_{(x)}}. \quad \text{식 (5)}$$

Total Input Packet count :

$$totalPktIn = (ifInUcastPkts + ifInBroadcastPkts + ifInMulticastPkts). \quad \text{식 (6)}$$

식 (3)에서 이더넷 트래픽의 이용률 분석 식을 보인다. 시리얼링크 이용률의 경우 반이중방식과 전이중방식의 두 가지 전송방식이 있다. 반이중 시리얼링크의 경우 입력 트래픽의 합이 링크의 이용률이 되며, 전이중 시리얼링크의 경우 링크의 연결방향에 따라 두 개의 라인으로 구성된다. 시리얼링크의 경우 이용률이 90%이상이면 네트워크 과부하상태로 판명한다. FDDI네트워크와 마찬가지로, 네트워크 이용률이 일시적으로 90%를 초과하는 경우에는 큰 문제가 없으나, 평균이용률이 90%이상 초과 시에는 네트워크에 심각한 문제가 있다고 판단하여 네트워크 재구축이나 장비 업그레이드가 반드시 필요하다. 식 (4)에서 각 링크 연결 방향에 따라 전 이중 시리얼 링크의 이용률 분석 식을 보인다.

에러율의 경우 전체대역폭에 1%이상이 에러로 판명되면 네트워크 관리자는 네트워크 디바이스를 점검해봐야 한다. 식 (5)에서 에러율 분석 식을 보인다. 식 (5)에서 $totalPktIn$ 은 전체입력패킷의 수를 의미한다. 식 (6)과 같이 전체입력패킷의 수는 입력 유니캐스트 패킷의 수, 입력 브로드캐스트 패킷의 수 그리고 입력 멀티캐스트 패킷의 수를 합하여 측정할 수 있다.

4) 트래픽 제어 에이전트를 이용한 동적 SLA

[그림 1]에서와 같이 클라이언트가 트래픽 제어 에이전트로 대역폭을 요구하면, 트래픽 제어 에이전트는 이 정보를 데이터베이스에 저장하고 데이터베이스내의 SLA테이블의 정책과 일치하는지 확인한다. 이때, 동일한 서비스 정책을 이전에 맺었던 적이 있다면 트래픽 제어 에이전트는 서비스 종료 일자를 확인하고 그 서비스가 유효한 기간이면, 서비스가능메시지를 클라이언트에게 전송한다. 새로 갱신된 정책인 경우에는 트래픽 제어 에이전트내의 수락제어모듈에서 데이터베이스에 있는 정보를 바탕으로 망 상태를 감시하는 단계를 거쳐 서비스가능상태를 조사한다. 서비스가능상태이면 트래픽 제어 에이전트 간의 통신을 통하여 서비스 수락여부를 판단하게 된다.

다음은 클라이언트와 트래픽 제어 에이전트간의 SLA 절차이다.

① 클라이언트는 트래픽 제어 에이전트로 서비스 가능여부를 문의한다.

② 현재 차별화서비스 도메인에서 가능한 서비스이면, 다시 패킷송출 대상 인접 도메인으로 서비스 가능여부 메시지를 전송한다.

③ 마지막 도메인의 트래픽 제어 에이전트까지 메시지 전송이 되고 최종적으로 서비스할 수 있다고 판단되면 서비스 수락 메시지를 리턴 한다.

클라이언트와의 제어정보 교환이 끝나면 트래픽 제어 에이전트는 통신 모듈을 통하여 라우터로부터 망 자원에 관한 정보를 가져오는 역할과 라우터에 QoS 정보를 설정하는 작업을 수행한다. 이때, 라우터로부터 수집된 정보는 데이터베이스에 저장되고, 수집된 데이터를 기반으로 자원관리모듈에서는 현재 망에서 서비스할 수 있는지를 계산하게 된다. 계산된 정보는 인증제어 모듈에서 클라이언트로부터의 요청에 대한 수락여부를 판단하는 기준이 되며 트래픽 제어 에이전트가 효율적으로 망 자원을 모니터링 할 수 있도록 도와주는 정보로도 사용되게 된다.

수락제어모듈에서 서비스가 가능하다는 판단이 내려지면 라우터로 QoS 설정을 위한 SLA구성 메시지를 보낸다. 다음은 트래픽 제어 에이전트와 라우터간의 SLA 동작절차이다.

① 트래픽 제어 에이전트는 데이터베이스에서 SLA 정책을 유지하고 있다가 정책기간이 완료되면 정책을 삭제한다.

② 새로운 SLA요청이 들어오면 트래픽 제어 에이전트는 요청타입, 전송률을 비교한다.

ⓐ 만약 요청에 맞는 정책이 없으면, 트래픽 제어 에이전트는 경계라우터로 새로운 클래스의 라우터 구성메시지를 보내고 DSCP값을 할당한다.

ⓑ 요청에 맞는 정책이 있으면, 트래픽 제어 에이전트는 라우터의 대역폭할당요청에 사용된 SLA ID값을 사용자에게 반환한다.

SLA는 특정사용자에게 특정서비스를 할당하는 메커니즘으로 Customer ID, 서비스타입, 서비스 타입인자, 서비스 제약조건 등의 정보를 포함하고 있다. 본 논문의 트래픽 제어 에이전트는 책임영역에서 할당된 링크용량을 초과하지 않도록 SLA를 보장하며, [그림 4]와 같은 명령으로 데이터베이스에 SLA정보를 추가, 갱신, 삭제한다.

```
○ tcasla [-p port] -a customer_id service_type
  rate rate_units{g|m|k} drop_pr start_date
  end_date (add)
○ tcasla [-p port] -u sla_id customer_id
  service_type rate rate_units{g|m|k} drop_pr
  start_date end_date (update)
○ tcasla [-p port] -d sla_id (delete)
```

[그림 4] 트래픽 제어 에이전트와 DB간 SLA명령 인자

[그림 4]에서 'customer_id'는 트래픽 제어 에이전트에 의해 유지되는 사용자 정보를 의미하며, 'service_type'은 공급되는 서비스의 형태를 의미한다. 'rate'는 클라이언트와의 서비스수준에서 최대 협약가능 대역폭으로 G(gigabits per second), M(megabits per second), K(kilobits per second)의 인자를 가진다. 'drop_pr'은 서비스 수준에서 제외될 가능성을 의미한다. 'start_date'는 SLA가 발효되는 날짜와 시간을, 'end_date'는 SLA가 무효화되는 날짜와 시간을 의미한다. 'sla_id'는 SLA가 성공적으로 데이터베이스에 추가되었을 때 반환되는 값을 의미하며 데이터베이스갱신 및 삭제 시에 SLA요청 파라메타로 전달된다.

클라이언트는 SLA에 의해 할당된 서비스를 사용하기 위해 트래픽 제어 에이전트에게 BAR을 요청하며 구성요소는 [그림 5]와 같다.

```
○ Client ID, SLA ID
○ Service Level arguments
  (rate, maximum burst, etc)
○ Source identifier
  (port number, IP address, protocol)
○ Destination identifier
  (port number, IP address, protocol)
○ Request duration(start date, end date)
```

[그림 5] BAR 구성요소

트래픽 제어 에이전트는 클라이언트의 요청을 수락하기 전에 책임영역 내에서 SLA의 한계를 초과하지 않도록 BAR을 보장한다. 클라이언트와 트래픽 제어 에이전트, 트래픽 제어 에이전트와 트래픽 제어 에이전트간의 BAR 동작절차는 다음과 같다.

① 트래픽 제어 에이전트가 대역폭할당요청을 받으면 SLA ID를 사용하여 대역폭할당요청이 서비스단계 협약에 일치하는지를 식별하고 요청을 허용할지 결정한다.

② 요청이 수락되면 트래픽 제어 에이전트는 대역폭할당요구 시에 분류된 라우터 ID에 맞는 경계라우터 정책수행과 마킹을 위해 라우터 구성메시지와 흐름에 할당된 DSCP값을 전송한다.

③ 트래픽 제어 에이전트는 데이터베이스 내에 SLA정책을 유지하고 있다가 정책기간이 완료되면 정책을 삭제한다.

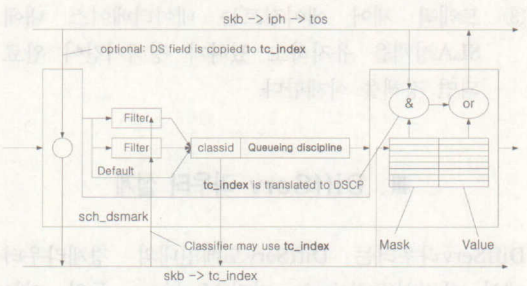
Ⅲ. DiffServ 라우터 설계

DiffServ라우터는 DiffServ도메인내의 경계라우터에서 폴리싱(Policing), 마킹(Marking) 등의 기능을 수행하며, 코어라우터에서는 IP헤더의 ToS필드 중 첫 번째 6비트를 이용하여 PHB를 수행한다. DiffServ는 그룹을 기반으로 동작하기 때문에 각각의 흐름들이 어디로 흘러가는지 보다는 각각의 흐름들을 하나의 그룹으로 취급하고, 이러한 그룹들을 차별화서비스 네트워크 내에 하나의 흐름으로 취급한다. 따라서 이러한 흐름을 처리하기 위한 행동양식(PHB)을 라우터에 설정해야만 한다. 그리고 다른 차별화서비스 도메인으로 패킷이 송수신될 때, 모든 경계라우터들은 다른 도메인에서 입력된 패킷들을 자신의 도메인에 적합하도록 큐 관리정책을 설정해 주어야 한다. <표 2>에서 경계라우터의 dsmark 큐 관리정책을 구성하기 위한 구성 파라메타를 보인다.

<표 2> dsmark 구성 파라메타

Variable name/tc keyword	Value	Default
indicies	2^n	none
default_index	0 ... indicies-1	absent
set_tc_index	none(flag)	absent
mask	0 ... 0xff	0xff
value	0 ... 0xff	0

<표 2>에서 dsmark를 구성하는 인자들로 indicies, default_index, set_tc_index가 있다. indicies 인자는 (mark, value)쌍으로 표현되며 테이블의 최대 크기를 나타낸다. indicies를 설정할 때에는 테이블의 최대 크기를 계산하여 그 값을 설정하며 최대 값은 2^n이다. set_tc_index는 dsmark에게 DS 필드의 값을 알리고, 이 값을 필터에 저장시킨다. [그림 6]에서 dsmark 큐 관리정책의 동작절차를 도식화하며, 동작 절차를 정리하면 다음과 같다.



[그림 6] dsmark 큐 관리정책

① 만약, 입력패킷에 set_tc_index가 세팅되었다면, DS필드는 이 값을 확인하고 필터링 된 값을 skb->tc_index에 저장시킨다.
② 분류기가 실행되어 skb->tc_index에 저장된 클래스ID를 반환한다. 이때, 필터를 발견하지 못하면 default_index옵션을 적용시킨다. 모두 만족시키지 못하면 tc_index필터 값을 재설정한다.
③ ①번과 ②번 과정을 마치면 패킷은 inner qdisc로 이동한다. 여기에서 패킷은 필터에서의 반환값(skb->tc_index에 저장되고 반환되는 클래스ID)을 다시 사용하게 되며, 이 값은 (mask, value) 테이블에 인덱스로 사용된다.
④ 마지막으로, 패킷에 할당되는 DS필드의 값은 식 (7)에 의해 변환된다.

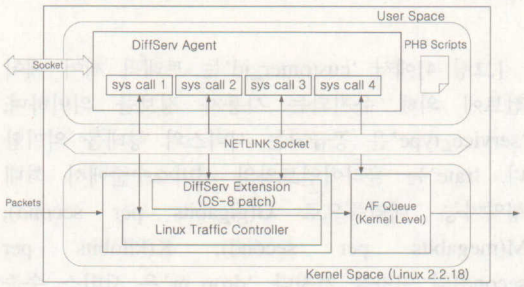
$$new\_DS\_field = (old\_DS\_field \ \& \ mask) \mid value \quad (7)$$

식 (7)은 경계라우터의 tc_index 필터에서 코어라우터의 PHB 수행을 위한 새로운 DSCP값을 추출해내는 식이다. tc_index 필터는 입력 트래픽에서 추출된 DS필드에 0xfc를 사용하여 마스킹을 수행하고, 마스킹 된 비트를 오른쪽으로 2비트 쉬프트하여 새로운 DSCP값을 변환한다.

1) 라우터설정을 위한 시스템호출

본 논문에서 이용된 모든 라우터는 리눅스를 운영체제로 하는 PC를 이용하며, 리눅스박스를 라우터로 이용하기 위해 시스템이 부팅될 때 routed 데몬을 로딩한다. 이러한 리눅스라우터가 DiffServ의 경계라우터와 코어라우터의 기능을 수행할 수 있도록 하기 위해 시스템호출과 넷링크 소켓(netlink socket)을 라우터설정 인터페이스로 이용한다.

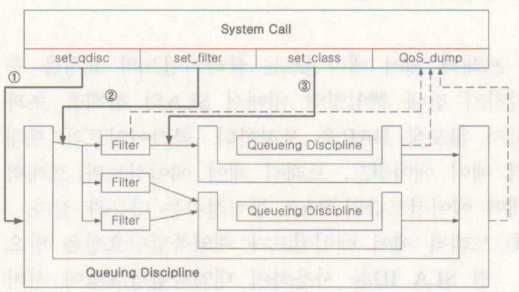
커널 내에 새로운 스케줄러를 생성하거나 생성된 패킷스케줄러를 설정하기 위한 방법은 크게 두 가지로 분류할 수 있다. 첫 번째는 넷링크 소켓을 이용하는 방법이고, 두 번째는 사용자 애플리케이션이 커널과 통신을 위해 이용되는 시스템호출을 이용하는 것이다. [그림 7]에서 라우터설정을 위한 시스템호출 구성도를 보인다.



[그림 7] 라우터설정을 위한 시스템호출 구성도

본 논문에서는 커널 내의 스케줄러를 조절하기 위해 넷링크 소켓을 이용하며, 코어라우터의 큐 설정과 커널로 전달된 데이터를 처리하는 부분은 시스템호출을 이용한다. 본 논문의 시스템호출은 넷링크 소켓과 동일한 구조로 tc_modify_qdisc(), tc_ctl_tclass(), tc_ctl_filter()를 이용하며, 처리데이터가 커널로 전달되는 방법만 넷링크 소켓과 다르게 설정한다.

본 논문에서는 라우터의 큐 구조를 동적으로 제어하기 위해 세 개의 시스템호출을 정의하고, 시스템호출에 대한 정보를 커널에 전달하기 위해 세 개의 구조체를 정의한다. [그림 8]에서 시스템호출이 커널 내부의 트래픽 제어 블록에 대응되는 구조를 보인다.



[그림 8] 시스템호출이 트래픽제어블럭에 대응되는 구조

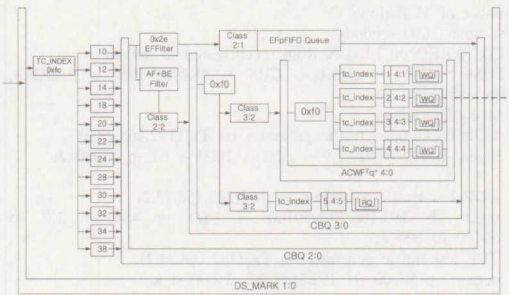
[그림 8]에 보이는 세 개의 구조체는 set_qdisc, set_filter, set_class이고, 대응되는 구조체는 qdisc_msg, filter_msg, class_msg이다. 이 구조

체는 리눅스 커널 내부의 tc_modify_qdisc(), tc_ctl_tfilter(), tc_ctl_ tclass() 함수와 대응된다.

2) 코어라우터 설계

차별화서비스 네트워크내의 코어라우터에서는 경계라우터에서 분류되어진 패킷의 DSCP값만으로 각 PHB를 분류(BA classification)해서 각각의 대기큐로 입력된다. EF, AF, default PHB로 설정된 패킷들에게 최상의 서비스를 제공하기 위해 적절한 버퍼관리정책과 스케줄링을 수행하게 된다. 경계라우터의 설정이 고 비용이고 동적인데 반해 코어라우터에서의 설정은 저 비용에 정적인 구성을 가지게 된다.

코어라우터에서 패킷들은 필터에 의해 AF, EF, BE 서비스로 분류되고, 분류된 패킷들은 해당클래스에 연결된 패킷스케줄러의 동작에 따라 스케줄링 된다. [그림 9]에서 코어라우터의 상세도를 보인다.



[그림 10] 코어라우터의 상세도

[그림 9]에서 ds_mark 큐 관리정책은 출력장치와 직접적으로 연관되어있으며, 여기에서는 입력되는 패킷으로부터 ToS바이트를 추출하여 tc_index 필터로 전송한다. tc_index필터는 추출된 ToS바이트에 0xfc를 사용하여 마스킹을 수행하고, 마스킹된 비트를 오른쪽으로 2비트 쉬프트 하여 DSCP값을 추출한다. 추출된 DSCP값은 skb->tc_index에 저장되고, 패킷은 CBQ(Class Based Queue)로 넘어 간다.

본 논문에서의 CBQ는 high, medium, low의 우선순위를 가지며 우선순위 큐에는 EF, AF BE트래픽 순서로 분리되어 처리된다. CBQ에서 사용되는 스케줄링 알고리즘은 링크-공유 스케줄러(link-sharing scheduler)와 GPS(generalized packet scheduler)이다. 링크-공유 스케줄러는 탑-레벨 링크-공유 스케줄러로 망이 과부하 상태가 되었을 때 동작하여 우선순위가 낮은 클래스의 링크 사용량을 한정된 값으로 제한하고, 우선순위가 높은 클래스가 할당된 대역폭보다 더 많이 사용할 수 있도록 대역폭사용량을 조절하는 기능을 수행한다.

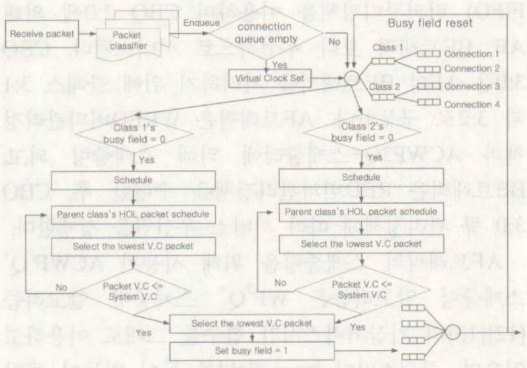
CBQ 2:0은 패킷이 입력되었을 때 AF, BE, EF 트래픽으로 분류를 수행한다. EF 패킷은 0x2e 필터에 의해 클래스 2:1로 이동하고 그 외의 다른 패킷들은 0필터를 통해 클래스 2:2로 이동한다. CBQ 2:0에서의 EF PHB는 같이 수행되는 다른 종류의 PHB보다 높은 우선순위를 가지고 서비스 우선권을 가로채기 할 수 있어야 하므로 pFIFO(priority FIFO) 버퍼관리정책을 이용하며 CBQ 2:0에 의해 AF, BE트래픽 보다 우선적으로 서비스된다. CBQ 3:0은 AF와 BE트래픽을 처리하기 위해 클래스 3:1과 3:2로 구분한다. AF트래픽은 WRED버퍼관리정책과 ACWF2Q+스케줄러에 의해 스케줄링 되고, BE트래픽은 RED버퍼관리정책을 수행한 후, CBQ 3:0 큐 관리정책에 따라 서비스 우선권을 경쟁한다.

AF트래픽의 스케줄링을 위해 사용된 ACWF^{2}Q$^+$ 스케줄링 알고리즘은 WF^{2}Q$^+$ 스케줄링 알고리즘[12][13]의 가상서비스시간 함수를 그대로 이용하고 있으며, 클래스마다 busy field를 두어 비동기 큐잉을 지원한다. 또한, WF^{2}Q$^+$ 알고리즘에서 흐름의 수가 많아질 경우 정렬 작업의 복잡도가 증가하는 문제를 해결하기 위해 클래스별로 정렬작업을 수행한다.

비동기 큐잉은 기존 방식에서 송수신시 하나의 흐름에 대해 하나의 큐잉 작업이 존재하는 것과는 달리, 큐 입력(enqueue)할 때 수신 인터럽트에 의한 하나의 작업과 큐 출력(dequeue)할 때 송신 인터럽트에 의한 작업이 독립적으로 존재한다. 즉, 비동기 큐잉에서는 큐 입력이 수행된 후 연결의 부모 클래스와 루트 노드의 상태에 따라 바로 큐 출력이 실행될 수도 있고, 수신 여부와 상관없이 큐 출력만 독립적으로 발생할 수도 있다. 이 경우 전송 중임을 나타내는 busy field를 두어 큐 입력 작업에서 스케줄러를 호출하지 못하도록 하여, 특정 클래스에만 빈번한 스케줄링이 일어나는 것을 방지함으로써 공평한 전송을 제공한다.

클래스기반 큐잉은 일반적인 스케줄링 알고리즘의 구현에 있어서 정렬의 대상 수가 많아질 때 구현 복잡도 증가 문제를 보완하기 위해서, 흐름의 서비스율이 비슷한 종류의 흐름들을 하나의 클래스로 구성하여 클래스 단위로 정렬하고 전송할 패킷을 결정한다. 이는 패킷의 크기가 고정된 ATM 네트

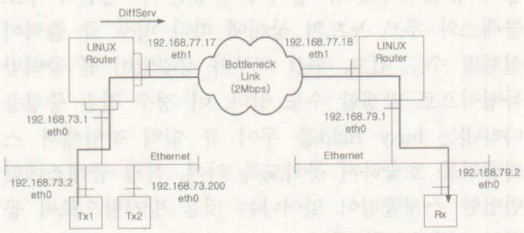
워크에서 서비스 요구율에 따라 클래스를 구성하는 것과는 달리, 패킷크기가 가변인 패킷 네트워크의 경우, 흐름을 서비스 요구율에 따라 클래스를 구성하는 것은 의미가 없기 때문이다. 따라서 본 논문의 클래스 기반 큐잉은 패킷의 서비스 간격에 따라서 흐름을 특정 클래스로 분류하고, 이를 기반으로 정렬하여 전송 패킷을 결정한다. [그림 10]에서 본 논문의 ACWF^{2}Q$^+$ 스케줄러에서의 큐 입력과 큐 출력의 동작 과정을 보이고 있으며, 이를 각 단계별로 설명하면 다음과 같다.



[그림 10] ACWF^{2}Q$^+$ 스케줄러의 큐 입력, 출력 동작 과정

Ⅳ. 실측실험을 통한 성능분석

이 절에서는 실측실험을 통하여 본 논문에서 제안한 DiffServ 환경에서의 트래픽 제어 시스템의 성능을 분석한다. 제안 시스템의 실측실험 모형은 [그림 11]과 같으며 지연, 지터, 패킷 폐기율을 이용하여 시스템의 성능을 측정한다.



[그림 11] 실측실험 평가모형

지연 값 측정방법은 RFC 2679[14]에 정의되어 있으며, 이 값은 수신측에서 측정된 패킷의 마지막 비트의 통신시간(wire time)에서 송신측에서 측정된 링크 상에 도착한 패킷의 통신시간 차이를 이용하여 측정된다. 지터 값은 IETF의 IPPM WG[15][16]에서 정의한 측정방법을 이용하며 지연 값을 이용하며 두 개의 패킷의 연속성을 이용하여 측정된다. 따라서 지터 값을 측정하기 위해서는 두 개의 패킷이 필요하게 된다. i번째 패킷의 지연 값을 D_i라 한다면, 지터 값은 $|D_i - D_{i-1}|$ 이 된다.

1) AF PHB의 성능분석

실험 1에서는 세 개의 TCP흐름(flow)과 한 개의 UDP흐름에 srTCM, RED 버퍼관리정책 그리고 PQ(Priority Queue)알고리즘을 적용했을 경우와 본 논문에서 제안한 trTCM, WRED, ACWF^{2}Q$^+$ 스케줄링 알고리즘을 적용했을 경우에서 트래픽의 처리율과 서비스의 공평성을 측정한다.

```
UDP background traffic : 500Kbps
packet size(header included) : Flow A = 512byte,
Flow B, Flow C, Flow D = 512byte
number of TCP flows A = number of TCP flows B =
number of TCP flows C
test time : 600 seconds
RED, WRED on LINUX Router configured to default
trTCM : CIR = 256Kbps, CBS = 5Kbps, PIR = 512Kbps,
PBS = 5Kbps
host configuration :
  UDP background traffic generator on Tx1(192.168.73.2)
  mgen -b 192.168.79.3:9 -i eth0 -s 1000 -r 61 -p 50900 &

  UDP test traffic generator on Tx1(192.168.73.2)
  mgen -b 192.168.79.2:50100 -i eth0 -s 484 -r 1.0 -p
  50100 -d 600 &
  TCP test traffic generator on Tx2(192.168.73.2)
  netperf -p 50200 -H 192.168.79.2 -l 600 -f k
  .....
  netperf -p 50400 -H 192.168.79.2 -l 600 -f k
```

[그림 12] 공평성(fairness) 실험조건

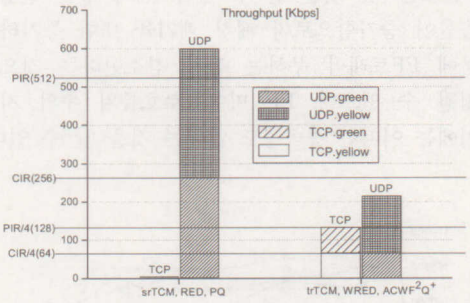
실험 1를 위한 실험조건은 [그림 12]와 같으며, AF PHB의 공평성(fairness)을 계산하기 위해 본 논문에서는 식 (8)을 사용한다[17].

$$FI = \frac{(\sum_i x_i)^2}{N \times \sum_i x_i^2}. \quad (8)$$

식 (8)에서 FI(Fairness Index)는 공평성 측정을 위한 값으로써 0과 1 사이의 값을 가진다. x_i값은 i번째 트래픽 소스의 평균 처리율을 의미하며, N값은 현재 실험에 참여하고 있는 송신측 트래픽의 총 수를 의미한다. 공평성 측정을 위한 값 FI가 1에 가까워지면 질수록 수신측으로 전달되는 트래픽의 대역폭 분포는 좀더 공평해짐을 의미한다.

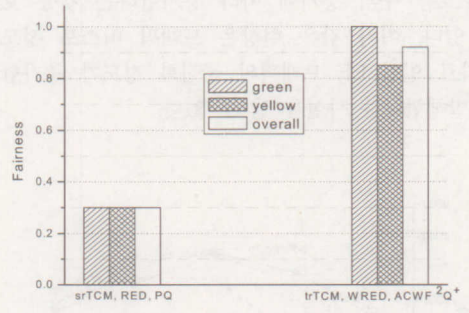
[그림 13]은 srTCM, RED 버퍼관리정책, PQ

알고리즘을 적용하였을 경우와 본 논문에서 제안한 트래픽 제어 환경인 trTCM, WRED, ACWF^{2}Q$^+$를 적용하였을 경우의 평균 처리율을 실험한 결과이다.



[그림 13] TCP, UDP 마이크로흐름의 평균 처리율

먼저, IETF DiffServ Spec.에서 제안한 방법으로 실험한 결과 병목현상으로 인하여 트래픽의 포화상태가 되기 전까지 각 흐름은 인증대역폭(CIR : Committed Information Rate)을 만족하며, 잉여 대역폭을 공평하게 할당받는다. 포화상태 이후, [그림 13]에 보이는 것처럼 UDP 마이크로흐름이 자신의 대역폭뿐만 아니라 잉여 대역폭까지 할당받음으로써, TCP흐름은 대부분 폐기되어 대역폭을 전혀 할당받지 못한다.



[그림 14] TCP, UDP 마이크로흐름의 공평성 측정

두 번째로 본 논문에서 제안한 트래픽 제어 환경을 적용하였을 경우의 실험결과는 [그림 14]에 보이는 것처럼 TCP와 UDP흐름 모두에서 AF트래픽에 대한 공정한 트래픽 분배를 제공하며, 잉여대역폭에 대해서도 공정한 트래픽 분배를 제공한다. 또한, trTCM 설정에서 절정대역폭(PIR : Peak Information Rate)값을 인증대역폭에 가깝게 설정하면 할수록 공정성은 더 높게 나타난다. 그리고 절정대역폭과 인증대역폭의 차가 크면 클수록 UDP 마이크로흐름이 가용대역폭을 더 많이 차지한다.

2) EF PHB의 성능분석

이 절에서는 EF트래픽 부하와 EF 마이크로흐름들의 군집화(aggregation)가 전체 EF트래픽의 전송률에 어떠한 영향을 미치는가에 대한 실험을 수행한다. 전송률측정을 위해 사용된 성능 인자는 패킷 폐기율, 지연, 지터 등의 값을 이용한다. EF PHB 실험을 위한 실험조건은 [그림 15]와 같다.

Variables:
traffic aggregation degree (microflow count) : [1, 20]
EF traffic load: [400, 1200] Kbps
EF packet size (different between microflows)
frame size of EF reference stream used for measurement: [256, 1024] bytes

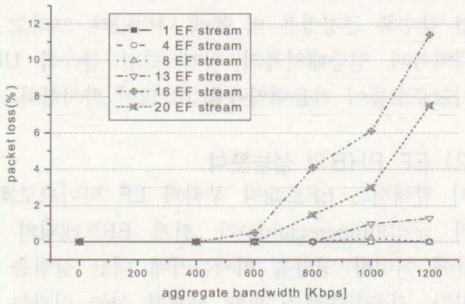
Stream profiles:
EF stream : constant bit rate 300 Kbps, packet size variable BE stream : constant bit rate 2.0 Mbps

Test conditions:
EF queue-limit : 10 packets (constant)
bottleneck link bandwidth : 2 Mbps
queueing algorithm for EF traffic: PQ
policing rate: 1.2 M
policing rate at ingress interfaces : 1200 Kbps
exceed action: drop
burst tolerance: 100,000 bytes

[그림 15] EF PHB 실험조건

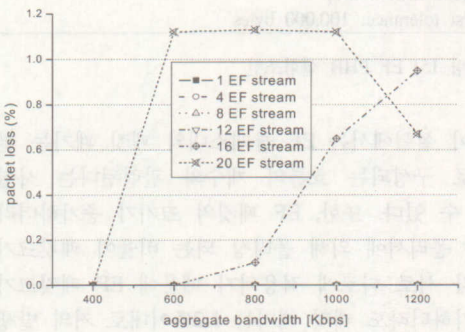
이 실험에서는 EF 클래스내의 패킷 폐기는 클래스로 구성되는 흐름의 개수와 관련된다는 사실을 알 수 있다. 또한, EF 패킷의 크기가 증가하더라도 PQ 폴리서에 의해 폴리싱 되는 비율이 패킷크기에 따라 서로 다르게 적용되기 때문에 EF 패킷크기가 증가하더라도 패킷 폐기는 1.2%이내로 거의 발생하지 않음을 알 수 있다.

EF 트래픽 부하를 증기 시킴으로써 패킷 폐기율은 증가하는 반면 지연은 감소한다는 것을 알 수 있다. 지터는 EF 트래픽 부하 때문에 흐름의 수와 같이 증가하다가 EF 마이크로흐름의 개수가 12이상일 때부터 지터 값은 거의 일정하게 유지됨을 알 수 있다. [그림 16]은 여러 개의 EF 마이크로흐름이 군집화(aggregation)될 때 패킷 폐기율을 보이고 있다.



[그림 16] EF 마이크로흐름 수에 따른 패킷 폐기율 (pkt size: 256byte)

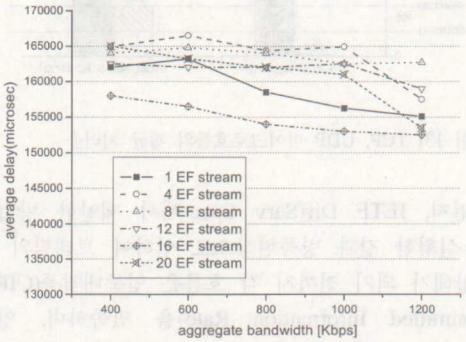
EF 패킷의 폐기율은 클래스내의 EF 마이크로흐름들이 증가함에 따라 같이 증가함을 알 수 있다. [그림 16]에서와 같이 EF 트래픽 부하는 전체 링크 용량의 10분에 3에 해당하는 600Kbps부터 발생하기 시작하며 패킷 폐기는 EF 마이크로흐름의 수가 16 이상일 때부터 발생한다. 또한, EF 마이크로흐름의 수가 증가하면 할수록 패킷 폐기로 인한 EF 부하는 감소한다는 사실을 알 수 있다. [그림 17]에서는 EF 프레임 크기를 256byte에서 1024byte로 증가 시켰을 때 패킷의 폐기율을 보이고 있으며, 최대 패킷 폐기율은 1.2%이하이다.



[그림 17] EF 프레임 크기 증가에 따른 패킷 폐기율 (pkt size: 1024byte)

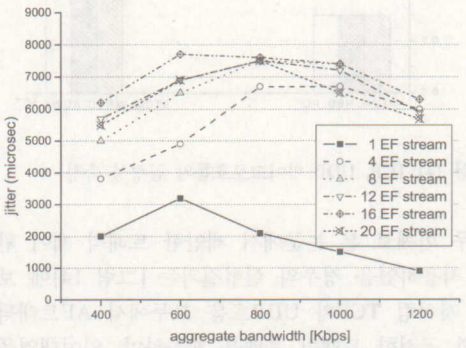
우선순위 큐에 입력된 패킷들은 패킷 폴리싱 정책에 의해 패킷을 전송할 것인가 아니면 폐기할 것인가를 결정한다. 전자의 경우, 용량이 큰 패킷은 더 많은 토큰을 소비하며 이로 인하여 다른 마이크로흐름에는 작은 토큰이 할당된다. 따라서 작은 수의 마이크로흐름을 가진 EF 흐름들은 더 큰 폐기확률을 가지게 되어 더 많은 패킷을 폐기하게 된다.

[그림 18]에서는 EF 마이크로흐름수의 증가에 따른 평균 지연 값을 보여고 있으며, 입력 마이크로흐름이 주어질 때 전송률이 증가하면 할수록 지연은 감소한다는 것을 알 수 있다. 이와 같은 현상은 전송률이 증가함으로써 패킷 폐기율 또한 증가하기 때문에 EF트래픽 부하는 점차 감소한다는 것으로 설명될 수 있으며, EF 마이크로흐름의 수와 지연 사이에는 어떠한 종속성도 없다는 것을 알 수 있다.



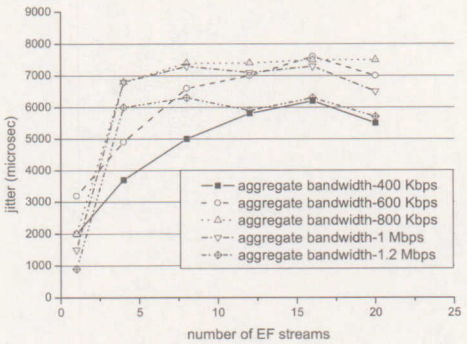
[그림 18] EF 마이크로흐름수 증가에 따른 평균 지연

[그림 19]에서는 EF 마이크로흐름수의 증가에 따른 지터 값을 보이고 있으며, 지터 값은 EF 마이크로흐름 수의 증가에 따라 증가한다는 것을 보이고 있다. 이와 같은 현상은 트래픽 버스트 정도의 증가로 인한 EF 트래픽의 군집화 정도가 증가함으로 발생한다는 사실을 알 수 있다.



[그림 19] EF 마이크로흐름 수 증가에 따른 지터 값

[그림 20]에서는 EF 마이크로흐름수를 증가시키면서 군집화 대역폭을 증가시킴으로써 얻어지는 지터 값을 측정하였으며, 지터 값은 EF 마이크로흐름수 증가에 따라 점차 증가하다가 EF 마이크로흐름의 수가 12개 이상일 경우 곡선의 정도는 다소 감소되고 지터 값은 EF 패킷크기와 관계없이 일정하게 유지된다는 사실을 알 수 있다.



[그림 20] 군집화 대역폭 증가에 따른 지터 값

V. 결 론

본 논문에서는 DiffServ에서 다양한 서비스를 제공하기 위한 PHB의 서비스 차별화 및 각각의 서비스 형태 분석을 목적으로 DiffServ라우터와 트래픽 제어 에이전트의 기능 요소를 분석하고 설계하였다. 성능 평가에서는 트래픽 패턴을 변경시켜가면서 DiffServ QoS모델의 QoS보장 여부를 실험하였고 이 실험에서 각기 다른 PHB의 성능을 관찰하였다. 실험에서 주의할 성능 척도는 지연, 지터, 패킷 폐기율과 패킷 전송률이었으며 여러 가지 매개 변수 변경에 의해 다양한 결과가 관측하였다. 실험 결과, EF트래픽의 경우는 트래픽의 증가와 관계없이 할당된 대역폭만큼 전송됨을 볼 수 있었고 AF트래픽의 경우 네트워크 부하에 따라 전송률이 변화하며 잉여 대역폭이 존재할 경우 AF트래픽에서 잉여 대역폭을 할당받음을 확인할 수 있었으며, 네트워크 과부하 상태에서도 최소 전송률은 보장됨을 확인할 수 있었다.

참 고 문 헌

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

[2] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An Architecture for Differentiated Services," IETF RFC 2475, December 1998.

[3] W. Almesberger, "Linux Traffic Control Implementation Overview," Technical Report, EPFL, November 1998.

[4] P. Salembier, S. Shenker, C. Partridge, R. Guerin, "Specification of Guaranteed Quality of Service," IETF RFC 2212, September 1997.

[5] A. Parekh, "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks," Ph.D. dissertation, MIT, February 1992.

[6] K. Nicholas, S. Blake, F. Barker and D. Blake, "Definition of the Differentiated Services Field(DS Field) in the IPv4 and IPv6 Herders," IETF RFC 2474, December 1998.

[7] D. Verma, "Supporting Service Level Agreement in IP Networks," Macmillan Technical Publishing, 1999.

[8] B. Evans, "Differentiated Service Implementation," http://www.ittc.ku.edu/~kdrao /BB/.

[9] Andreas Terzis et al., "A Prototype Implementation of the Two-Tier Architecture for Differentiated Services," RTA 99.

[10] Benjamin Teitelbaum and et al., "Internet 2 QBone : A test Bed for Differentiated Services", INET 99.

[11] 김효곤, "Architectural and Practical Problems in DiffServ Deployment," 제 3회 차세대 인터넷 워크샵 : Session III-QoS, 2000년 11월.

[12] J. C. R. Bennett, H. Zhang, "Why WFQ Is Not Good Enough for Integrated Services Networks," Proceedings of NOSSDAV'96, Zushi, Japan, April 1996.

[13] J. C. R. Bennett, H. Zhang, "WF2Q+:

Worst-case fair weighted fair queueing," Proceedings of IEEE INFOCOM'96, San Francisco, CA, pp. 120-128, March 1996.

[14] G. Almes, S. Kalidindi, M. Zekauskas, "One-way Delay Metric for IPPM," IETF RFC 2679, September 1999.

[15] IP Performance Metrics; http://www.ietf.org/html.chaters/ippm-charter.html

[16] C. Demichelis, P. Chimento, "Instantaneous Packet Delay Variation Metric for IPPM," IETF IPPM draft, work in progress.

[17] R. Jain, "The Art of Computer Systems Performance Analysis," John Wiley and Sons Inc., 1991.

이 명 섭(Myung Sub Lee) 정회원

e-mail:skydream@yu.ac.kr

1998년 2월:경일대학교 컴퓨터공학 졸업 (공학사)

2000년 2월:영남대학교 대학원 컴퓨터공학과 졸업(공학석사)

2003년 8월:영남대학교 대학원 컴퓨터공학과 수료(공학박사)

<주관심 분야> 망관리 시스템, 데이타 마인닝, 에이전트, QoS(Quality of Service)

박 창 현(Chang Hyeon Park) 정회원

e-mil: park@yu.ac.kr

1986년:경북대학교 전자공학과 졸업 (공학사)

1988년:서울대학교 계산통계학과 전산학전공 (이학석사)

1992년: 서울대학교 계산통계학과 전산학전공(이학박사)

1992년 - 1993년: 서울대학교 컴퓨터신기술공동연구소 특별연구원

1998년 - 1999년: University of Maryland, Institute of Advanced Computer Systems, Visiting Researcher

1993-현재: 영남대학교 컴퓨터공학과 부교수

<주관심 분야> 인공지능, 데이타 마인닝, 에이전트, 지능형 망관리 시스템