웹 기반의 QoS 보장형 트래픽 제어 시스템

이 명 섭, 신 경 철, *류 명 춘, 박 창 현
영남대학교 컴퓨터공학과, *경운대학교 컴퓨터공학과
전화 : 053-810-1504 / 핸드폰 : 011-810-7101

A Web-based QoS-guranteed Traffic Control system

Myung Sub Lee, Kyung Chul Sin, Myung Choon Ryu, Chang Hyeon Park
Dept. of Computer Engineering, Yeungnam University
E-mail : skydream@cse.yu.ac.kr

Abstract

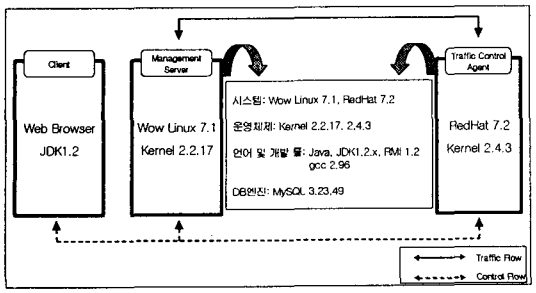
This paper presents a QoS-guaranteed traffic control system which supports QoS of realtime packet transmission for the multimedia communication. The traffic control system presented in this paper applies the integrated service model and provides QoS of packet transmission by means of determining the packet transmission rate with the policy of network manager and the optimal resource allocation according to the end-to-end traffic load. It also provides QoS for the realtime packet transmission through the AWF^2Q^+ Scheduling algorithm and per-class queuing method.

I. 서론

최근 들어 인터넷 사용자가 늘어남에 따라 인터넷에서 구할 수 있는 정보도 빠른 속도로 늘어나고 있으며, WWW (World-Wide-Web)의 대중화와 함께 다양한 인터넷 서비스가 제공되고 있다. 인터넷 서비스를 제공하는 다양한 응용 프로그램 들이 네트워크 기술과 관련되어 개발되면서 네트워크 트래픽은 계속적으로 증가하고 있다. 이러한 네트워크 트래픽의 증가는 네트워크 회선부족, 서비스 응답시간지연 등의 네트워크 전송품질과 관련된 여러 가지 문제점을 야기하고 있다. 이와 같은 네트워크 상의 여러 가지 문제점을 해결하기 위해 네트워크 상의 장비들을 모니터링하고 분석하는 시스템에 대한 연구가 활발하게 진행되고 있으며, 그 결과로 MRTG, Etherfind, TCPdump, WebTrafMon, WebTraMas[1] 등과 같은 망 관리 프로그램 들이 개발되었다. 이러한 시스템들은 관리 대상 네트워크 상에서 현재 얼마나 많은 트래픽을 유발하고 있는지, 그리고 어떤 시스템에서 병목현상을 일으키고 있으며 최대 트래픽은 얼마인지 등에 대한 정보를 네트워크 관리자에게 제공하지만 트래픽 전송에 있어 QoS를 지원하지 않는다. 본 논문에서는 멀티미디어 통신에서 실시간 패킷의 전송품질을 보장하는 트래픽 제어 시스템을 제시한다. 본 논문에서 제시하는 트래픽 제어 시스템은 기본적으로 IntServ (Integrated Service)[2] 모델을 응용하였으며, 다음과 같은 기능을 수행한다. 첫째, TCP 윈도우 크기 조절기능을 이용하여 재 전송 빈도 감소 및 종단간 데이터흐름제어와 트래픽 폭주를 방지한다. 둘째, 실시간 패킷의 경우, AWF^2Q^+ 스케줄링 알고리즘과 클래스별 큐잉 방법을 제공하여 전송품질을 보장 한다.

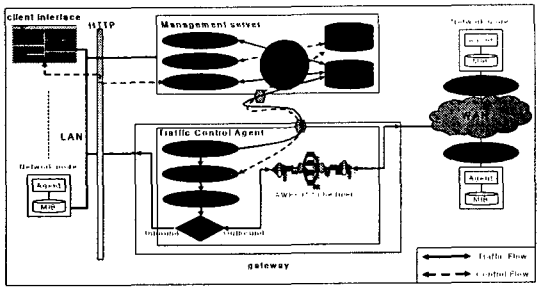
II. 트래픽 제어 시스템

본 장에서는 웹 기반 QoS 보장형 네트워크 트래픽 제어시스템의 개발 환경과 구현하는데 있어서 사용되는 각각의 요소들에 대한 사항들을 세부적으로 설명한다. 본 논문에서 제안하고 있는 트래픽 제어 시스템의 개발 환경은 [그림 1]과 같다. 본 논문에서 제시하는 웹 기반 QoS 보장형 네트워크 트래픽 제어시스템은 [그림 2]에 보이는바와 같이 트래픽 제어 에이전트(traffic control agent)와 관리 서버(management server), 클라이언트 인터페이스(client interface)로 구성되어 있다.



[그림 1] 트래픽 제어시스템 개발 환경

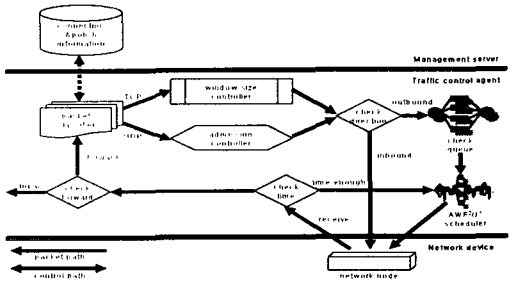
트래픽 제어 에이전트는 수집기(collector), 패킷 분류기(packet classifier), 제어기(controller), 패킷 스케줄러(packet scheduler)로 구성되어 내부 네트워크와 외부 네트워크와의 접속 노드 상에서 동작하며, 이 노드를 통과하는 패킷에 대한 트래픽 제어 정책 반영, 지속적인 패킷 정보 수집, 수집한 패킷 정보의 관리 서버로의 전송 등을 담당한다. 관리 서버는 구성기(configurator), 정책 관리기(policy manager), 트래픽 분석기(traffic analyzer)되며, 네트워크의 트래픽에 대한 정보를 트래픽 제어 에이전트로부터 전송 받아 저장한 후, 요구에 적합한 형태로 가공, 분석한 다음 클라이언트 인터페이스에게 실시간으로 정보를 전송한다. 클라이언트 인터페이스(client interface)는 네트워크 관리자가 원격으로 실시간 트래픽에 대한 정보를 보면서 트래픽 제어 정책을 반영할 수 있도록 하는 그래픽 사용자 인터페이스를 제공한다. 이러한 구성요소들에 대한 자세한 설명은 다음의 각 절에서 주어진다.



[그림 2] 트래픽 제어시스템 전체 구성도

2.1 트래픽 제어 에이전트

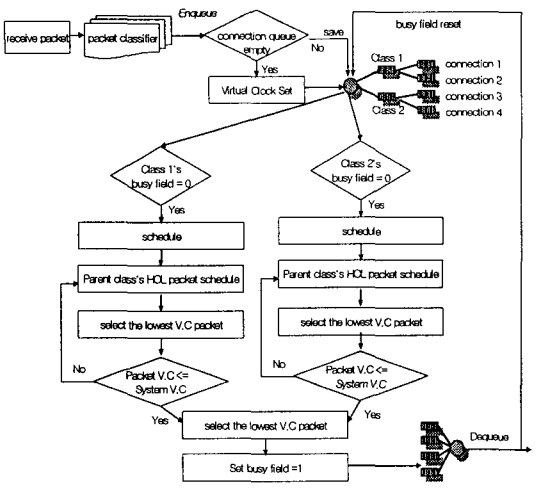
트래픽 제어 에이전트는 노드를 통과하는 패킷 정보를 수집하는 수집기(collector), 네트워크 관리자의 정책에 따라 패킷들을 해당 클래스로 분류하는 패킷 분류기(packet classifier), TCP 패킷의 윈도우 크기 제어와 UDP 패킷의 인증 제어를 담당하는 제어기(controller), AWF^2Q^+ 알고리즘에 의해 출력 패킷의 순서를 제어하는 패킷 스케줄러(packet scheduler)로 구성되며, 각 구성요소들은 커널 내부의 네트워크 스택의 IP 계층에서 동작된다. [그림 3]은 트래픽 제어 에이전트의 전반적인 동작과정을 각 구성요소의 기능별로 보여주고 있으며, 이를 간략하게 기술하면 다음과 같다. 먼저 트래픽 제어 에이전트는 관리 서버로부터 클래스 정책을 받는다. 네트워크 노드로부터 패킷이 수신되면 수신된 패킷이 포워드 패킷인지를 판단하여 포워드 패킷인 경우 클래스 정책 및 연결 구분에 따라 분류한 다음 해당 큐에 저장한다. 이 때 패킷의 종류가 실시간 패킷인 경우는 인증 제어를 거쳐 큐에 저장하고, TCP 패킷인 경우는 윈도우 크기 제어 과정을 거쳐 큐에 저장한다. 패킷을 큐에 저장하기 전에 내부(inbound) 패킷과 외부(outbound) 패킷을 구분하여 내부 패킷은 그대로 전송하고, 외부 패킷의 경우에만 큐에 저장하게 된다.



[그림 3] 트래픽 제어 에이전트 내부 구성도

패킷 스케줄러는 패킷 큐잉 알고리즘으로 잘 알려진 WFQ(Weighted Fair Queueing)의 제약점을 보완한 WF^2Q(Worst-case Fair Weighted Fair Queueing)[3]와 이의 구현 복잡도를 낮춘 WF^2Q^+ 알고리즘[4]을 기반으로 비동기 큐잉과 클래스 기반 큐잉을 지원하도록 변형한 AWF^2Q^+ (Asynchronous WF^2Q^+) 알고리즘으로 동작한다. 본 논문의 AWF^2Q^+ 스케줄링 알고리즘은 WF^2Q^+의 가상 서비스 시간 함수를 그대로 이용하고 있으며, 클래스마다 busy field를 두어 비동기 큐잉을 지원하며, WF^2Q^+ 알고리즘에서 흐름의 수가 많아 질 경우 정렬 작업의 복잡도가 증가하는 문제를 해결하기 위해 클래스별로 정렬 작업을 수행한다. [그림 4]에서 본 논문의 AWF^2Q^+ 스케줄러에서의 큐 입력과 큐 출력의 동작 과정을 보이고 있으며, 이를 각 단계별로 설명하는 다음과 같다.

1) 패킷을 수신하면 패킷 분류기를 거쳐 큐 입력 단계로 간다.
2) 이 때, 해당 연결의 큐가 비어 있으면 연결의 가상 시간을 설정한 후 입력하고, 그렇지 않으면 바로 큐 입력한다.
3) 패킷의 부모 클래스의 busy field가 0이면 스케줄링을 수행한다.



[그림 4] AWF^{2}Q$^+$ 스케줄러에서의 큐 입력과 큐 출력

3-1) 스케줄링이 끝난 후, 패킷의 부모 클래스의 연결 중 HOL(head of line)에 있는 패킷들에 대해서 스케줄링을 수행한다.

3-2) 가상 시간이 가장 작은 연결이 선택되고, 이 연결의 패킷이 다시 부모 클래스의 최종 패킷으로 선택한다.

3-3) 루트 노드 아래의 모든 클래스 중 하나가 같은 방식으로 다시 선출한다.

3-4) 선출된 클래스의 해당 패킷은 다른 클래스와의 경쟁을 통하여 가장 작은 가상시간을 갖는 패킷이 서비스할 패킷으로 선택한다.

4) 서비스할 패킷이 결정되면 busy field를 1로 설정하고 큐 출력을 실행한다.

5) 큐 출력 후 다음 서비스를 위해 최근 전송된 클래스와 연결을 찾아 busy field를 리셋한다. 이러한 과정을 반복한다.

2.2 관리 서버

관리 서버의 기본 기능은 트래픽 제어 에이전트와 클라이언트 인터페이스가 원활하게 동작될 수 있도록 이들에서 발생하는 정보들을 저장, 처리, 분석, 전송 등을 수행하는 것이다. 특히, 관리 서버는 트래픽 제어 에이전트와 클라이언트 인터페이스 사이의 정보 전송을 실시간으로 처리해야 하며, 이들과의 정보 전송이 서로 독립적으로 이루어져야 하기 때문에 서로간의 작업이 비동기적으로 동작되어야 한다. 이러한 동작은 다음과 같은 Java 메소드 코드로 설명될 수 있다.

```
private Thread thread;
 private void start() throws Exception()
    thread = new Thread(this);
    thread.setPriority(Thread.MIN_PRIORITY);
    thread.setName("ManagerServer");
    thread.start(); // 클라이언트 인터페이스로 정보를
                    // 전달하기 위한 쓰레드 시작
    receiveDate();  // 에이전트로부터의 실시간 정보를
                    // 받는 데이터그램 서버 소켓
```

관리 서버가 클라이언트 인터페이스에게 실시간 정보를 보내는데 필요한 통신은 데이터그램 소켓을 통해 전송하지만, 정책에 대해서는 패킷의 손실이 있을 수 있기 때문에 정보 유실방지와 신뢰도를 유지하기 위해 Java RMI(remote method invocation)를 이용하여 관리한다. 다음은 원격 메소드(remote method)를 이용한 정책의 송수신작업을 수행하는 Java 코드를 보여주고 있다.

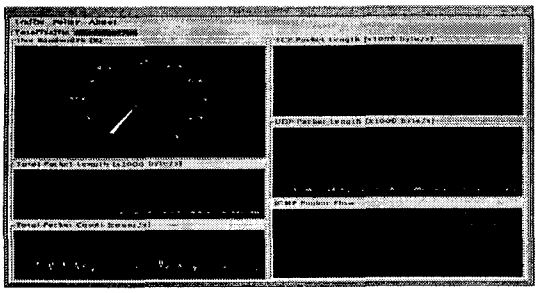
```
public synchronized Void setRealtimePKinfo(boolean flag)

    pkdata[pkcnt].source = ntohs(tcph->source);
          :
    //1초 간격으로 수집된 패킷을 매니저 서버로 전달한다.
    if((current_time=time(NULL) >
        last_time + TIME_INTERVAL)
          :
    sendto(sockfd, buf, n, 0,
    (struct sockaddr *) & ser_addr, sizeof(ser_addr));
          :
```

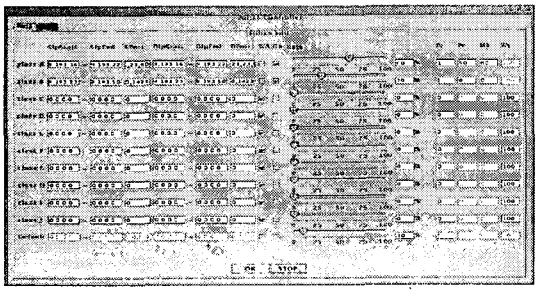
2.3 클라이언트 인터페이스

본 논문의 클라이언트 인터페이스는 트래픽 모니터링 인터페이스와 정책 작성 인터페이스로 구성 되어 있다. 트래픽 모니터링 인터페이스는 Java 2D로 구현되어 대상 네트워크 상의 전체 트래픽 및 호스트별, 클래스별 트래픽을 그래픽 화면으로 제공한다. 트래픽 모니터링 인터페이스는 데이터그램 소켓을 열어 실시간으로 트래픽 정보를 받음과 동시에 비동기적 메소드를 이용하여 1초 간격으로 패킷 정보를 화면에 나타내어 주고, 관리 서버의 원격 메소드를 호출하여 정보를 주고받는다. [그림 5]에서 네트워크의 트래픽을 모니터링하고 있는 화면을 보이고 있다.

정책 작성 인터페이스는 관리자가 다양하게 정책을 적용할 수 있도록 10개의 클래스에 대한 항목 값을 입력할 수 있도록 작성되었으며, 각 클래스는 IP 어드레스, 포트번호 등을 입력할 수 있고, 나머지는 기본 클래스로 적용되게 된다. [그림 6]에서 정책 작성 인터페이스의 실행화면을 보인다.



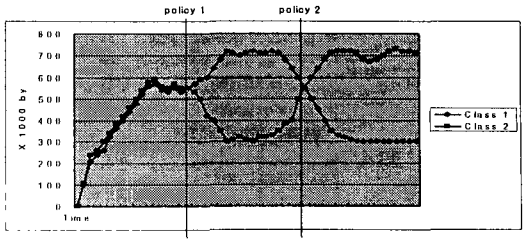
[그림 5] 트래픽 모니터링 실행화면



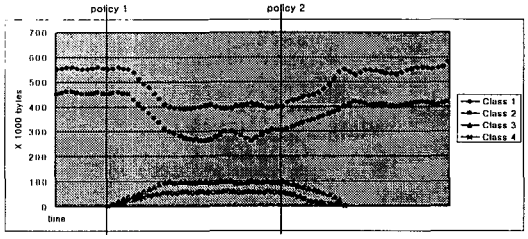
[그림 6] 정책 작성 사용자 인터페이스

III. 실험 및 고찰

본 논문의 QoS 보장형 트래픽 제어 에이전트를 통한 네트워크 트래픽 관리를 실험하기 위해서, Windows Media Encoder 7.0을 사용하여 높은 대역폭을 가지는 동영상 스트림 데이터 타입의 advanced stream format(asf) 파일을 생성하여 전송 데이터로 이용하였으며, 각 종단 시스템에서는 패킷 생성기를 사용하여 임의로 트래픽을 발생시켰다. [그림 7], [그림 8]은 이러한 실험 데이터를 이용하여 실제 트래픽 제어를 적용한 결과를 그래프로 나타낸 것이다. 발생된 트래픽은 모두 외부로 흘러나가는(outbound) 패킷이며, 그 대역폭은 1.2Mbyte/sec 보다 크도록 하였다. [그림 7]에서 각 클래스에 정책을 적용하기 전에는 동일한 전송 량으로 전송이 이루어지다가 정책 1에서 클래스 1과 클래스 2에 각각 70%, 30%의 비율로 전송 량을 조절한 결과 7:3의 비율로 전송이 이루어지는 것을 보이고 있다. 정책 2에서 전송 량을 각각 30%, 70% 비율로 재조정한 결과 전송 량이 3:7의 비율로 전송이 이루어짐을 볼 수 있다. [그림 8]에서는 클래스 1, 클래스 2, 클래스 3, 클래스 4에 각각 40%, 30%, 15% 10%의 비율로 전송 량을 주었으며, 기본 클래스에는 5%의 전송 량을 배정한 경우의 그래프 화면이다. [그림 8]에서와 같이 클래스 3, 클래스 4가 유휴상태에 있을 때 클래스 1과 클래스 2가 유휴 대역폭을 할당받아 사용하다가 클래스 3, 클래스 4가 유휴 상태에서 깨어나면 클래스 1과 클래스 2는 자신에게 주어진 대역폭을 사용함을 보여준다.



[그림 7] 정책 적용 전·후의 트래픽 그래프



[그림 8] 타 클래스 휴지·활동 시 트래픽 그래프

IV. 결론

본 논문에서는 IntServ(Integrated Service) 모델의 구조를 응용한 웹 기반 QoS 보장형 트래픽 제어 시스템을 제시하였다. 본 논문에서 제시한 트래픽 제어 시스템은 내, 외부 네트워크의 접속 노드에 트래픽 제어 에이전트를 동작시켜 패킷의 재 전송 빈도 감소 및 종단간 데이터흐름제어와 트래픽 폭주를 방지하고, 연결 회선별로 대역폭을 관리하여 같은 클래스 내에서 모든 접속 세션이 균등한 대역폭을 가질 수 있도록 구현하였다. 또한 실시간 패킷의 경우, 인증제어기와 ACWF^{2}Q$^+$ 패킷 스케줄러를 두어 전송 품질 보장을 지원하고 있다. 이 연구의 후속 연구는 본 논문에서 제시한 트래픽 제어 시스템에 지능형 에이전트가 개입되어 동작하는 지능형 네트워크 관리 시스템 구조를 설계, 구현하는 것이다.

참고문헌

[1] Myung-Sub Lee and Chang-Hyeon Park, "Design and Implementation of Web-based Traffic Monitoring and Analysis System", Proceedings of the 2001 International Conference on Internet Computing, pp. 479-485, Las Vegas, June 2001.

[2] IETF, Integrated Services(IntServ) Working Group. http://www.ietf.org/html.charters/intserv-charter.html

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

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