액티브 라우터의 피드백 메커니즘을 이용한 혼잡제어 기법

최 기 현[†] · 장 경 수[††] · 신 호 진[†] · 신 동 렬[†††]

요 약

기존의 end-to-end 방식에서는 네트워크 내부에서 혼잡(congestion)이 발생했을 경우 각 전송자가 즉시 알아 낼 수 없기 때문에 일정시간 동안 수신된 패킷(packet)의 순서에 대한 정보로 혼잡이 발생했는지에 대해 추론하는 것이다. 이와 같은 방법은 RTT(Round Trip Time)가 커지면 혼잡이 발생할 경우 전송자가 전송 양을 줄인다 해도 이미 전송된 패킷들로 인하여 혼잡이 가중되며 전체적인 TCP 동기화 (TCP Global synchronization) 현상을 피할 수 없게 된다. 반면 네트워크 내부에서 직접적으로 정보를 얻거나 처리를 해 줄 수 있다면 혼잡 발생과 동시에 처리가 가능함으로 기존 방식보다 처리율이 향상될 것이다. 본 논문에서는 액티브 라우터의(Active Network) 피드백 메커니즘을 이용하여 네트워크 내부 정보를 각 전송자가 이용할 수 있도록 하기위해 라우터와의 통신을 이용하였으며, 코어 라우터의 큐 모듈은 RED(Random Early Detection)를 응용하여 ACC의 누락 메커니즘을 개선하였다. ACC를 확장한 메커니즘인 EACC(Enhanced Active Congestion Control)를 제시하고 모의실험을 통해 기존의 혼잡제어나 ACC(Active Congestion Control)보다 성능이 향상됨을 보여준다.

Active Congestion Control Using Active Router's Feedback Mechanism

Ki-Hyun Choi[†] · Kyung-soo Jang[††] · Ho-Jin Shin[†] · Dong-Ryeol Shin[†††]

ABSTRACT

Current end-to-end congestion control depends only on the information of end points (using three duplicate ACK packets) and generally responds slowly to the network congestion. This mechanism can't avoid TCP global synchronization which TCP congestion window size is fluctuated during congestion occurred and if RTT (Round Trip Time) is increased, three duplicate ACK packets is not a correct congestion signal because congestion maybe already disappeared and the host may send more packets until receive the three duplicate ACK packets. Recently there is increasing interest in solving end-to-end congestion control using active network frameworks to improve the performance of TCP protocols. ACC (Active congestion control) is a variation of TCP-based congestion control with queue management In addition traffic modifications may begin at the congested router (active router) so that ACC will respond more quickly to congestion than TCP variants. The advantage of this method is that the host uses the information provided by the active routers as well as the end points in order to relieve congestion and improve throughput. In this paper, we model enhanced ACC, provide its algorithm which control the congestion by using information in core networks and communications between active routers, and finally demonstrate enhanced performance by simulation.

키워드 : Active Network, ACC, EACC, 혼잡 제어

1. 서 론

컴퓨터의 속도가 발전함에 따라 사용자는 멀티미디어나 실시간 데이터 통신을 보다 쉽게 접하게 되었다. 인터넷상에서 사용자가 요구하는 서비스의 종류도 다양해 졌으며 이에 따른 새로운 대역폭의 요구를 원하게 되었다. 앞으로 이러한 요구는 더욱 증가할 것으로 보이며 네트워크에서 이러한 요구를 보다 효과적으로 대처하기 위해서 현재 많은 시도가 이루어지고 있다. 혼잡제어 분야 또한 네트워크의 자원을 효율적으로 관리하기 위한 방법 중에 하나로 많은 연구가 이루어지고 있다. 본 논문에서는 한정된 네트워크의 자원을 효율적으로 관리하고 사용자의 요구를 만족시키기 위해 등장한 액티브 네트워크(Active Network) 기술을 이용하여 혼잡제어에 새로운 기법을 제공할 것이다.

기존의 end-to-end 피드백 혼잡제어는 오직 단말노드에서만 혼잡을 감지하고 제어할 수 있다. 이 방식은 혼잡이 발생하여 전송자가 혼잡을 감지할 때까지 걸리는 제어시간에 영향을 받기 때문에 혼잡 발생 시점에서 처리를 할 수 없다. 기존의 TCP 혼잡제어 방식은 수신측과 송신측과의 지연시간 즉 RTT가 클 경우 그 만큼 혼잡을 제어하는데 시간이 더 오래 걸리게 되며 네트워크의 대역폭이 클 경우 혼잡이 감지되어 어떤 처리를 시작할 때까지 이미 전송된 패킷으로 인해 혼잡이 더욱 가중되게 된다.

† 준 회 원 : 성균관대학교 대학원 정보통신공학부
†† 정 회 원 : 경인여자대학 컴퓨터정보기술학부 교수
††† 정 회 원 : 성균관대학교 전기전자 및 컴퓨터공학부 교수
논문접수 : 2001년 12월 19일, 심사완료 : 2002년 7월 11일

TCP Tahoe[7]나 TCP Reno[7]의 경우 패킷 누락(drop)에 대한 정보를 세 개의 중복된 ACK 패킷을 수신 함으로써 인지 한다. 이와 같은 메커니즘은 혼잡 발생 시점을 알 수 없기 때문에 전송자는 혼잡이 발생했음 해도 불구하고 패킷 전송을 하게 된다. 중복된 ACK 패킷을 받을 때까지 혼잡은 가중될 수도 있으며 이미 혼잡이 제거된 상태에서 TCP의 윈도우 크기를 줄이게 된다. 본 논문에서는 기존의 TCP 혼잡제어 방식에서는 제공할 수 없는 네트워크 내부 정보와 라우터간의 통신을 이용하여 혼잡을 제어하는 기법을 제시한다.

혼잡 제어에 있어 액티브 네트워크 기술의 이용은 네트워크 혼잡에 보다 신속하게 응답하게 해준다. 기존의 네트워크의 코어 라우터(core router) 들이 주로 패킷 전송과 경로할당을 처리하는 반면 액티브 네트워크에서는 패킷내에 프로그램이나 처리에 필요한 정보를 담아 전송하게 되는데 네트워크상에 있는 노드는 이 패킷(액티브 패킷(Active packet)[10] 또는 스마트 패킷(Smart packet)[1])을 읽어 들여 각 라우터의 실행환경(Execution Environment)에서 실행할 수 있고 필요한 프로그램을 다른 곳으로부터 다운로드하여 실행 가능하다. 또한 네트워크 내부 노드들은 서로에게 필요한 네트워크 상의 정보를 공유할 수 있다. 이 기술을 사용할 경우 혼잡이 발생하면 네트워크 내부에서 이를 감지하여 주변 라우터에 혼잡에 대한 정보를 알려줄 수 있으며 직접적으로 혼잡을 제어할 수 있다[10]. 이와 같은 방법은 실제 도로에서 교통방송을 들으면서 운전을 하는 운전자의 혼잡 대처 방법과 유사하다고 할 수 있다. 각 운전자는 교통방송에서 도로상태에 대한 정보를 들을 수 있으며 보다 원활한 소통 구간으로 방향을 바꿀 수 있게 된다. 액티브 라우터는 교통 방송국이라 할 수 있으며 각 패킷은 라우터를 거칠 때 마다 네트워크의 상태를 알 수 있고 패킷 내의 프로그램에 따라 더 진행할지 다른 곳으로 우회할 지를 결정하게 된다. 실제 도로상황과는 다르게 네트워크 상에서는 혼잡정보를 항시 얻을 수 없다. 패킷이 액티브 라우터에 수신 되었을 경우 라우터상에 있는 혼잡정보와 패킷상에 있는 혼잡 정보를 이용하여 패킷에 대한 처리를 수행하기 때문에 액티브 라우터를 이용한 혼잡제어는 액티브 라우터의 위치와 라우터간의 통신이 중요하다고 할 수 있다.

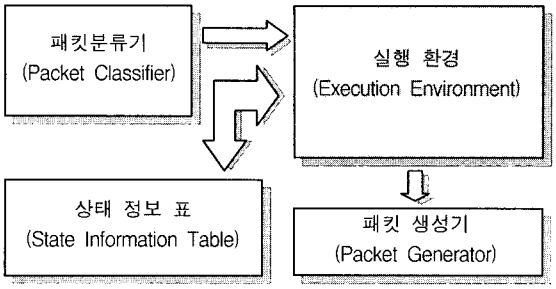
혼잡이 발생할 경우 라우터의 패킷 처리방식에 따라 네트워크의 성능에 큰 영향을 줄 수 있다. 기존의 라우터에서 큐를 관리하는 방식에는 큐(queue)가 다 찼을 경우 최후에 수신된 패킷을 누락시키는 Droptail 방식과 큐가 다 차기 전에 패킷을 누락시키는 RED(Random early Detection)방식이 있다[12]. 전자는 혼잡이 발생할 경우 네트워크상의 흐름(flow)들이 윈도우 크기를 동시에 줄이고 동시에 늘리는 TCP 동기화 현상(Global synchronization)을 유발한다. RED의 경우 윈도우 크기가 작은 흐름일지라도 다른 흐름의 영향을 받게 된다. 라우터의 큐상에 보다 많이 존재하는 다른 흐름의 패킷들과 같은 누락 확률로 처리 되기 때문에 윈도우 크기가 작은 흐름의 경우 혼잡이 발생할 경우 상대적으로 전송률이 떨어 지게 된다. 따라서 본 논문에서는 혼잡처리를 네트워크 내부로 가져와 처리를 하기위해서 액티브 라우터를 구성하여 혼잡을 제어하는 ACC(Active Congestion Control)[5]을 TCP Tahoe와 Reno에 적용하여 성능상의 이점을 보이겠다. ACC는 RTT가 작을 경우 오히려 기존의 TCP 혼잡 제어 방식보다 성능이 떨어지는 단점을 갖는다. 이를 보완하기 위해 네트워크 내부 정보를 이용하여 각 흐름들이 보낼 수 있는 윈도우 크기를 구하고 이 정보를 각 전송자에게 피드백하는 메커니즘을 사용한 EACC (Enhanced ACC)를 보이겠다. 제 2장에서는 액티브 라우터의 구조와 동작방식을 설명하고, 제 3장에서는 액티브 라우터의 통신 방식을, 제 4장에서는 모의 실험결과 분석을 보이겠다. 제 5장은 본 논문의 결론이다.

2. 액티브 라우터의 구성요소와 구조

ACC와 EACC에서 사용한 액티브 라우터의 구조를 알아보고 각 모듈의 기능에 대해서 알아 보겠다. ACC는 코어 라우터만 액티브 라우터로 동작하게 되며 EACC는 코어와 엣지 라우터(edge router) 모두 액티브 라우터로 동작 하게 된다. 하지만 코어 라우터와 엣지 라우터의 구조는 동일하며 packet내에 전송되는 프로그램에 따라서 다르게 동작하게 된다. 수신 된 패킷 내의 프로그램은 라우터의 동작을 결정 하며 일반 라우터가 액티브 라우터로 동작할 수 있는 점이 바로 액티브 네트워크 기술의 가장 큰 장점인 실행환경이라고 할 수 있다. 본 장에서는 기본적인 액티브 라우터의 구조를 알아보고 ACC와 EACC에서 사용한 라우터의 동작 원리를 알아 보겠다.

2.1 액티브 라우터의 구성요소

액티브 라우터에 패킷이 수신되면 우선 패킷 분류기가 액티브 패킷과 일반 패킷을 분류하여 액티브 패킷일 경우 패킷 내에 들어 있는 정보나 프로그램을 실행환경으로 보내어 처리 하게 한다. 실행환경에서 처리 후 얻어진 정보는 상태정보 표(State Information Table ; SIT)에 저장하며 패킷 생성기를 이용하여 주변 라우터에 혼잡에 대한 정보를 제공할 수 있다. 일반적인 액티브 라우터는 다음과 같이 4개의 모듈로 구성된다.



(그림 1) 액티브 라우터의 구성

① 실행 환경(Execution Environment)
② 패킷 분류기(Packet Classifier)
③ 상태 정보 표(State Information Table)
④ 패킷 생성기(Packet Generator)

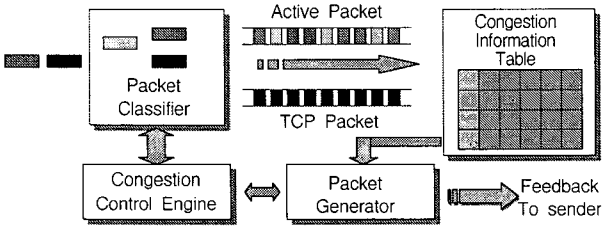
각 모듈의 기능은 다음과 같다. 패킷 생성기는 라우터간에 정보를 공유할 때 사용하게 된다. SIT는 액티브 라우터의 특성에 따라 사용 여부가 결정되며 혼잡이나 패킷 처리에 필요한 여러 정보가 저장된다. 각 액티브 라우터는 실행환경(Execution Environment)을 갖는다. 실행환경이란 packet내의 프로그램이 실행되는 곳이며 라우터의 특성을 결정 짓게 된다. 코어 라우터와 엣지 라우터가 같은 구조 하에서 다르게 동작 할 수 있는 것이 바로 실행환경 때문이다. 패킷내에 전송되는 프로그램은 전송자의 요구와 관리자의 정책에 따라서 다양할 수 있다. 이러한 요구와 정책이 일정한 규칙으로 정해질 수 있다고 가정하면 사용자는 자신의 패킷에 대한 처리를 달리 할 것을 요구 할 수 있으며 관리자는 각각의 트래픽에 대한 정책을 패킷 내에 프로그램을 전송하거나 필요한 프로그램을 전송 받아서 수행할 수 있을 것이다. 실행환경의 역할은 바로 이런 요구와 정책을 효율적으로 관리하고 통제하는 모듈이라고 볼 수 있다. 현재 인증과 보안, 액티브 패킷 자체 스케줄링 등의 연구가 활발이 진행 중에 있으며 액티브 (그림 3)은 EACC 액티브 라우터의 구조이다. 코어 라우터와 엣지 라우터는 CIT의 가동 여부와 CCE에서 처리되는 알고리즘만 차이가 있을 뿐 그 구조상 거의 같음을 볼 수 있다. 액티브 라우터의 장점 중의 하나는 특정 목적에 따라 라우터의 교체나 프로그램의 설정이 필요 없다는 것이다.

네트워크에 있어 실행환경은 그 연구 목적에 따라 다르기 때문에 표준적인 실행환경의 구축이 선결 해야 할 문제 중에 하나이다.

2.2 ACC와 EACC에서 라우터의 구조

2.2.1 ACC 코어 라우터의 구조

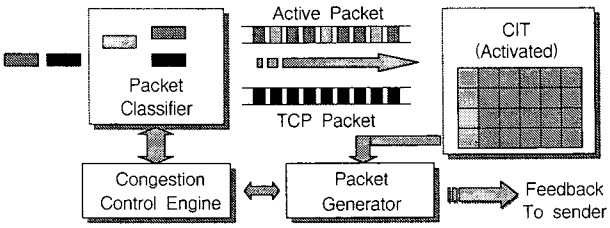
ACC는 코어에서 SIT와 유사한 CIT(Congestion Information Table)을 갖는다. 전반 적인 구조는 (그림 2)와 같다. 코어 라우터에 패킷이 도착하면 패킷 분류기에 의해서 일반 TCP 패킷과 액티브 패킷을 구별한다. 분류된 패킷은 ACC 코어 라우터에서의 실행 환경인 CCE(Congestion Control Engine)에 의해 실행되거나 정보를 얻게 된다. CCE에서 처리된 패킷은 CIT에 정보로써 저장되고 CCE는 패킷 생성기에 패킷을 피드백 할 것인지 여부를 전달하게 된다. CIT에는 흐름(flow)별로 전송자 주소, 수신자 주소, 윈도우 크기, 패킷 누락 개수 등의 정보가 저장된다. (그림 2)에서 알 수 있듯이 ACC는 코어 라우터에서 상태 정보를 관리하게 된다. 코어 라우터에 연결된 흐름의 수가 많아질 경우 상태정보의 관리가 어려워지며, 액티브 패킷과 일반 패킷의 처리에 자원을 낭비하게 된다.



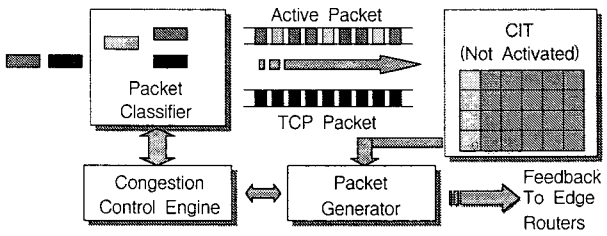
(그림 2) ACC 코어 라우터의 구조

2.2.2 EACC의 코어 라우터와 엣지 라우터

단지 패킷 속에 필요한 정보와 프로그램을 담아 전송 시키면 실행환경에서 그 목적에 맞게 동작하기 때문이다. 이러한 실행환경이 없는 일반 라우터의 경우에는 네트워크 관리자가 라우터를 직접 설정해 주어야 할 뿐만 아니라 새로운 장비의 교체까지 해야 한다. (그림 3)은 액티브 라우



EACC 엣지 라우터



EACC 코어 라우터

(그림 3) EACC 액티브 라우터의 구조

터가 그 목적에 따라 구조의 변함 없이 동작함을 보여준다.

코어 라우터는 큐 관리 방식에 있어 RED의 변형을 사용하였다. 전반적인 구조는 RED와 거의 유사하지만 문턱 값(threshold) 값에 따라 패킷을 누락시키는 RED와는 다르게 본 논문에서 제안한 RED의 변형은 피드백을 사용한다. ECN (Explicit Congestion Notification)을 사용한 RED(이하 RED ECN)와의 차이점은 혼잡지점에서 바로 전송자에게 혼잡 신호를 줄 수 있다는데 있다. RED ECN 또한 전송자가 혼잡 신호를 받게 되는데 까지 걸리는 시간이 RTT에 영향을 받으므로 EACC와 비교해서 신속한 처리가 어렵다.

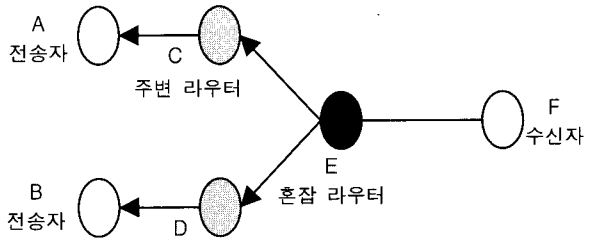
3. ACC와 EACC의 라우터 간의 통신 구조

3.1 ACC 라우터의 통신 구조

혼잡이 발생했을 경우 라우터가 혼잡 발생시 트래픽에 대한 정보를 저장하고, 주변 라우터에 혼잡발생에 대한 정보를 전송하여 혼잡에 대한 처리를 해줄 수 있다면 전송자가 즉각적인 처리를 하는 것과 같은 효과를 얻을 수 있을 것이다. 코어 라우터는 자신과 연결된 라우터들에 대한 정보를 이미 갖고 있기 때문에 혼잡에 대한 정보를 패킷에 담아서 전달하는 것은 많은 처리가 필요치 않다. 이미 사용가능한 정보와 혼잡에 대한 정보를 코어 라우터와 주변 라우터가 공유하기 때문에 이들 라우터들 사이의 통신이 일정한 규칙을 갖고 통신을 하게 된다면 혼잡제어에 있어 보다 빠르고 효율적인 처리가 가능할 것이다.

본 절에서는 ACC에서 제시한 혼잡제어를 이해하고 이보다 확장된 혼잡제어를 위한 액티브 라우터의 구성을 보이겠다. ACC는 보다 신속하게 혼잡에 반응하기 위해 혼잡이 발생한 라우터와 주변 라우터와의 통신을 이용하는 방법을 제안하였다[5]. 라우터 사이의 통신을 이용하여 혼잡을 제어할 경우 엣지 라우터에 연결된 각 전송자에 대한 독립적인 처리가 가능해 진다. 상대적으로 패킷에 대한 처리량이 작은 엣지 라우터가 코어 라우터의 정보를 이용해서 처리하게 되면 보다 향상된 결과를 얻을 것으로 기대된다. 본 논문의 3.2에서는 라우터 사이의 통신을 이용한 혼잡 제어에 대해서 살펴 볼 것이다. ACC 모의실험에서는 라우터와의 통신을 이용해서 혼잡을 제어하지 않고 단지 혼잡이 발생한 라우터에서 'Drop and Notice'방법을 이용하여 혼잡을 제어한다.

(그림 4)에서 노드 E는 혼잡이 발생하는 라우터이며 노드 C, D는 노드 E로부터 혼잡정보를 받아 혼잡 발생시 트래픽을 필터링하게 된다. 이상적인 경우 A에서 F까지의 RTT가 100ms이고 A에서 C까지의 지연 시간이 10ms라고 하면 혼잡 발생시에 기존의 end-to-end 처리방식보다 80 ms정도 빠르게 처리가 가능하게 된다.



(그림 4) ACC 에서 사용한 라우터간의 통신

최초 E에서 C또는 D로 혼잡정보를 전송할 때를 제외한다면 노드 C와 D는 그 이후에 들어오는 트래픽을 20ms의 지연시간으로 처리가 가능하게 된다. 실제 네트워크 상에서 액티브 라우터의 설치에 있어 문제점은 어떤 라우터가 액티브 라우터로 동작하고 어떤 라우터가 일반 라우터로 동작할 것인가에 있다. 이상적인 설치에서는 혼잡을 처리 할 때 까지 걸리는 제어시간을 줄일 수 있지만 실제 네트워크 상에서 라우터 모두를 액티브 라우터로 설정할 수는 없을 것이다. 모든 액티브 라우터에서 피드백된 정보를 전송자가 모두 수신하여 분석하여 처리하는 것은 비효율적이기 때문이다. ACC는 비록 이러한 액티브 네트워크의 설치상의 문제점은 고려하지 않았지만 혼잡제어에 있어 새로운 방향을 제시 했다는데 의의가 있다.

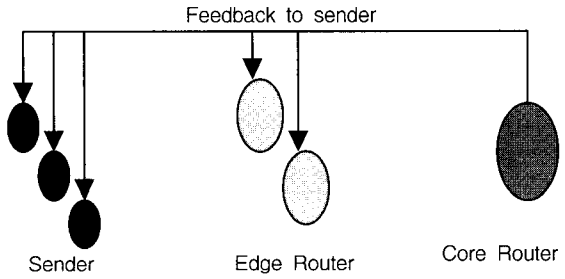
(그림 5)는 모의실험 상에서 사용한 라우터 사이의 통신을 보여준다. 모의실험에서 라우터는 액티브 패킷, 일반 TCP 패킷, ACK 패킷, UDP 패킷… 등의 여러 패킷을 처리 하게 됨으로 액티브 패킷의 누락이 발생하게 된다. 본 논문에서는 액티브 패킷은 가장 높은 우선 순위를 갖도록 설정하였다. (그림 4)의 라우터 E는 혼잡이 발생하면 E로 들어오는 패킷에서 현재의 윈도우의 크기를 액티브 패킷내에 저장하여 각 전송자에게 피드백하게 된다.

전송자는 피드백된 액티브 패킷에서 윈도우 크기를 읽어들여 새로운 윈도우 크기를 계산하게 된다[7]. 새로운 윈도우 크기를 계산하여 보내기 전까지 혼잡이 발생한 라우터에서는 패킷이 누락 될 당시의 윈도우 크기의 절반의 수까지 그 트래픽에 대해서 필터링을 하게 된다. ACC는 비록 주변 라우터와 통신을 하지 않더라도 18%정도의 성능향상을 보였다[5]. 본 논문에서는 ACC에서 모의 실험한 TCP Tahoe이외에 ACC를 TCP Reno에도 적용하여 결과를 보인다.

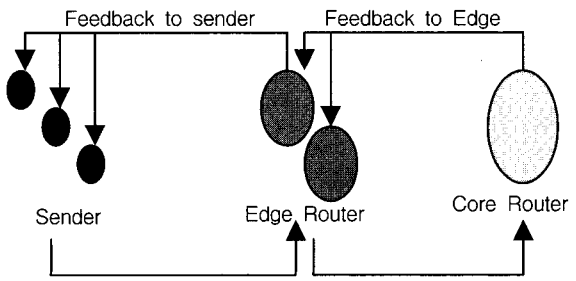
3.2 EACC 라우터의 통신 구조

ACC에서는 코어 라우터에서 상태 정보를 저장해야 하며

혼잡이 발생했을 경우 혼잡에 필요한 처리도 해야 한다. EACC에서는 코어 라우터에서 필요한 상태정보를 엣지 라우터로 옮겨 처리하고 피드백된 정보를 각 전송자로 전달하게 된다. 엣지 라우터는 코어 라우터에서 피드백된 패킷을 수신할 경우 자신에게 연결된 전송자에게 패킷을 복제하여 전송하게 된다. 전송자가 보내는 정보는 패킷 전송 시간과 코어에서 혼잡이 발생할 때 패킷 처리 프로그램이 전송된다. (그림 5)와의 차이점은 엣지 라우터에 전송된 코어 라우터 정보가 특정 전송자에만 전송되는 것이 아니고 각 전송자에게 전송되도록 한다. 또한 전송자에서 온 프로그램이나 정보를 처리하여 코어 라우터에 전송하도록 한다. 이 프로그램과 전송시간은 2장에서 설명한 CCE에서 처리된다. 엣지 라우터는 TCP 패킷이 수신되면 전송자의 개수를 저장하게 된다. 엣지 라우터에서의 상태정보는 코어에서 상태정보를 유지하는 것에 비교하면 상대적으로 부담을 덜어주게 된다. 각 엣지 라우터에서 수집된 정보는 코어 라우터에서 처리하게 되며 엣지 라우터에서 보낸 정보에 따라서 처리 방법은 달라 진다. 그 밖에 엣지 라우터는 UDP 패킷에 대해서 Zombie list[3]를 두어 수신되는 패킷을 처리한다. UDP 패킷은 지속적이며 버스티한 특성을 가지므로 혼잡이 발생할 경우 코어 라우터에 부담을 주는 요인이며 네트워크의 리소스를 낭비하는 요인이 된다. 이와 같은 Unresponsive 트래픽[2, 4]이나 Undelivered 패킷[4]을 엣지 라우터에서 처리 함으로써 네트워크 리소스를 효과적이고 빠르게 관리할 수 있다.



(그림 5) ACC의 모의 실험상에서의 라우터 통신



(그림 6) EACC에서 사용한 라우터 간의 통신

코어 라우터는 RED와 유사한 문턱 값을 관리하며 현재 큐의 길이에 따라서 피드백을 결정하게 된다. 엣지 라우터로 전송되는 윈도우 값의 계산은 다음과 같다.

$$\text{Estimate window} = \frac{\left(\text{qlim}_{+}\dfrac{\text{DelayTime}}{\dfrac{\text{packetlength}^{*}8}{\text{TargetRate}}}\right)}{\text{Sourcenum}} \approx \frac{\left(qlim_{+}\dfrac{BandwithDelayProduct}{Packetsize}\right)}{Sourcenum}$$

위 공식은 각 전송자가 보낼 수 있는 윈도우 크기가 된다. 'Delay Time'은 전송자에서 코어 라우터까지의 시간이며 'Target rate'은 코어 라우터의 처리 속도가 된다. 실제로 패킷의 도착 속도는 'Target rate'에 맞추어 전송되는 것이 아니고 TCP의 전송 알고리즘에 따라서 증가 되었다가 감소되는 특성을 갖는다. 공식에서는 코어 라우터가 하나의 패킷을 처리하는데 걸리는 시간마다 하나의 패킷이 도착하는 것으로 계산했다. 'Bandwidth delay product'는 네트워크로 보낼 수 있는 패킷의 양을 의미한다. 패킷 크기로 이 양을 나누면 각 전송자가 보낼 수 있는 윈도우 크기라고 할 수 있다[13]. 라우터의 버퍼 크기인 'qlim_'과 더해지면 네트워크에 전송할 수 있는 전체 양이 된다. EACC는 코어 네트워크에서 얻을 수 있는 정보인 지연시간, 전송자의 수 등을 이용하여 윈도우 크기를 전송하게 된다. 각 엣지 라우터에서 전송 받은 전송자의 수는 코어 라우터에서 자원 관리 정책에 따라서 처리 되게 된다. 본 논문에서는 각 엣지 라우터에 같은 양의 자원을 할당하였다. 엣지 라우터는 피드백된 패킷에서 윈도우 값을 읽어 들여 패킷을 복제하여 각 전송자에게 전송한다. ACC는 'Drop and Notice'방식으로 혼잡을 제어하는 반면 EACC는 패킷 누락이 발생하기 전에 미리 처리 한다는 점에서 다르다. 이는 ACC의 피드백 메커니즘과 RED의 혼잡제어 방식을 조합한 것으로 볼 수 있다. (그림 7)은 EACC의 코어 라우터에서 패킷에 대한 처리는 나타낸다. RED와 거의 유사한 구조를 갖고 있다. RED와 다른 점은 현재의 큐 크기를 바로 사용하는 점이다. 이는 현재의 네트워크 상태를 신속하게 처리해 줄 수 있으며 RED의 경우 문턱 값을 넘을 경우 패킷을 누락 시키거나 패킷내에 혼잡이 발생했음을 나타내는 정보 비트를 사용하지만 EACC 코어 라우터의 큐에서는 패킷 누락은 발생하지 않고, 피드백을 하게 된다. 라우터의 큐가 다 찼을 경우에는 Droptail FIFO큐와 같은 방식으로 패킷을 누락시킨다.

(그림 8)은 코어 라우터에서의 피드백 메커니즘을 나타낸다. 'estimate_wnd'는 3.2절의 공식에서 얻은 값이다. RED와 유사한 패킷 처리 방식을 거친 후 각 패킷은 문턱 값에

따른 피드백을 하게 된다. 현재 패킷의 윈도우 크기와 피드백 확률 값으로 최종적으로 피드백 함을 알 수 있다. 각 엣지 라우터는 코어 라우터에서 전송된 액티브 패킷을 수신하면 패킷 내의 윈도우 크기, 즉 'estimate_wnd'를 모든 전송자에게 다시 피드백 한다.

```
void Chameleon::enque(Packet * p ) {
        중략 ...
        if (q_-> length() > qlim_ * 0.8) active_queue
          (1,wnd,avgwnd,avgrtt,srcnum,p );
        else if ( q_-> length() > qlim_ * 0.5) active_queue
              (2,wnd,avgwnd,avgrtt,srcnum,p) ;
        중략 ...
}
```

(그림 7) EACC 코어 라우터에서 패킷 처리

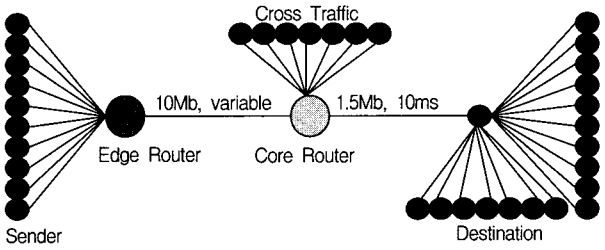
```
void Chameleon :: active_queue(int fb, int wnd, float avgwnd,
     float avgrtt, int srcnum, Packet * p ) {
   중략 ...
    double fb_prob = 0.15;
    if (fb ==1) {
            if ((wnd > estimate_wnd) &&
              (Random::uniform() <= fb_prob))
               feedback(p,(int)estimate_wnd,ss,0) ;
    } else if (fb == 2 ) {
            if ((wnd > estimate_wnd) &&
              (Random :: uniform() <= fb_prob-0.05))
               feedback(p,(int)estimate_wnd,ss,0 ) ;
    }
    중략 ...
```

(그림 8) EACC 코어 라우터의 피드백 메커니즘

4. 모의 실험 결과

4.1 모의 실험 환경

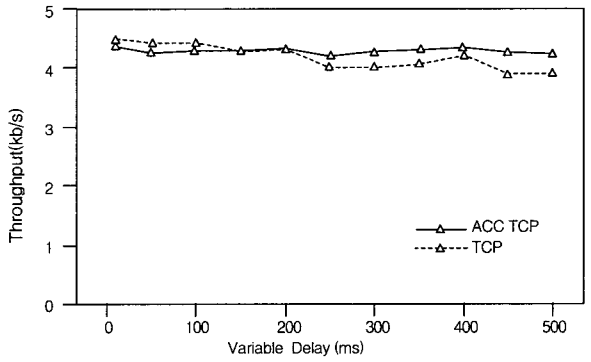
본 논문에서는 NS[11]를 이용하여 모의실험을 진행하였다. 액티브 라우터를 구성하기 위해서 NS에서 제공하는 큐 클래스 관련 모듈을 사용하였으며 액티브 패킷에 대한 처리 모듈을 추가하였다. (그림 9)는 모의실험에서 사용한 네트워크 구성이다. 이 그림에서 ACC의 경우 코어 라우터만 액티브 라우터로 동작하며 EACC는 엣지 라우터도 액티브 라우터로 동작한다. ACC와 동일한 결과를 얻기 위해서 ACC의 모의실험에서는 엣지 라우터를 일반 라우터로 동작시키고 엣지 라우터와 코어 라우터 사이의 지연시간을 변화시켜 가면서 성능을 측정하였다. 여기서 'sender'에서 엣지 라우터까지의 지연 시간은 10ms이고 대역폭은 각각 1Mb로 고정 시키고, 각 'sender'는 실험 시작부터 버스티하게 패킷을 전송한다. 코어 라우터의 큐 관리 방법에는 Droptail과 RED방법을 사용한다. 실험에서 각 라우터의 큐 길이는 50이고, 모의실험 시간은 200초로 설정하였다. Cross 트래픽은 패킷 누락에 대한 처리를 하지 않는 실시간 트래픽으로 본 논문의 혼잡제어 방식이 혼잡제어를 하지 않는 트래픽과 특정 노드(node)에서 만나서 혼잡을 발생 시킬 경우에 성능에서의 차이점을 관찰하기 위해서 구성하였다. EACC의 경우 엣지 라우터가 액티브 라우터로 동작하며 코어 라우터에서 피드백된 패킷을 처리하게 된다. ACC와 동일한 조건에서 모의실험하기 위해 엣지 라우터와는 TCP만 연결되며 UDP 패킷에 대한 엣지 라우터의 필터링(zombie list) 기능은 사용하지 않았다.



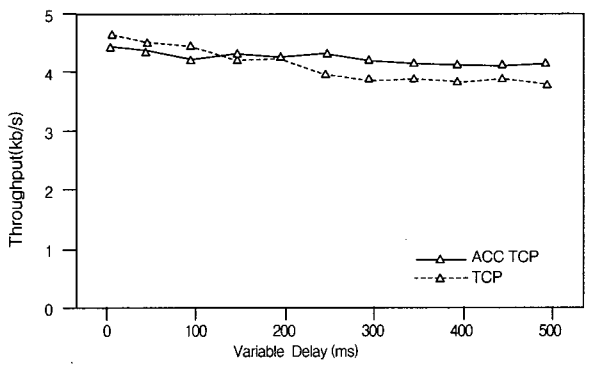
(그림 9) 실험에 사용한 topology

4.2 ACC 모의실험결과

액티브 라우터를 이용해서 혼잡을 제어할 경우 Tahoe[6]나 Reno[6] 모두 혼잡이 발생한 라우터에서 소스까지 지연시간이 짧을 경우 네트워크상에 전송되어있는 패킷의 수가 상대적으로 적다. 따라서 누락되는 패킷이 차지하는 비율이

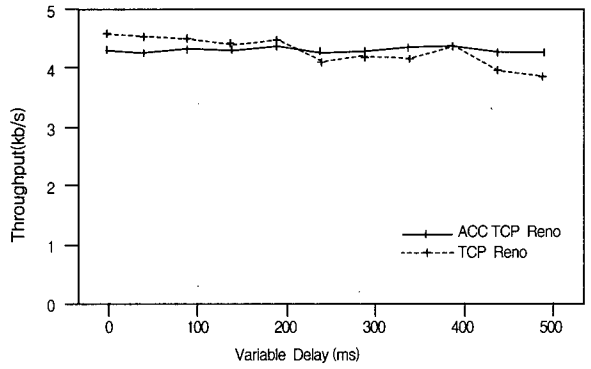


(그림 10) TCP Tahoe와 ACC TCP Tahoe(Droptail)

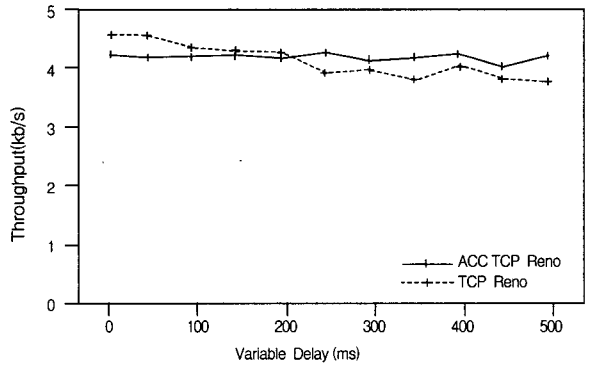


(그림 11) TCP Tahoe와 ACC TCP Tahoe(RED)

상대적으로 커지게 되므로 (그림10)~(그림 13)처럼 성능이 떨어지게 된다. 반면 지연시간이 클 경우에는 혼잡을 발생시키는 트래픽의 양이 상대적으로 많아 지게 되므로 성능이 향상되었다. (그림 10)~(그림 11)은 TCP Tahoe에 ACC를 적용한 결과이다. bandwidth * delay가 작을 경우에는 기존의 제어방식이 오히려 더 높은 처리량(throughput)을 보였다. 이 경우 ACC의 액티브 라우터가 전송자를 대신해서 트래픽의 양을 조절하는 역할에 있어 첫 번째 문제점이다. 두 번째 문제점은 ACC의 액티브 라우터는 혼잡이 발생할 경우 무차별적인 패킷 누락이 발생한다는 것이다. 상대적으로 bandwidth * delay가 클 경우에는 네트워크상에 있는 패킷의 양이 상대적으로 많은 반면 bandwidth * delay가 작을 경우 액티브 라우터에서의 패킷 필터링이 상대적으로 너무 많게 되며, RED와는 다르게 Droptail의 경우 혼잡이 발생할 경우 특정 전송자가 계속해서 필터링 될 확률이 높게 된다. 이는 ACC가 RTT의 변화에 따른 처리와 필터링 메커니즘에 전반적인 문제가 있음을 보여준다. (그림 12), (그림 13)은 TCP Reno에 ACC를 적용한 결과이다. 그림에서 볼 수 있듯이 TCP Reno에도 ACC가 성능 향상에 효과가 있음을 알 수 있다.



(그림 12) TCP Reno와 ACC TCP Reno(Droptail)

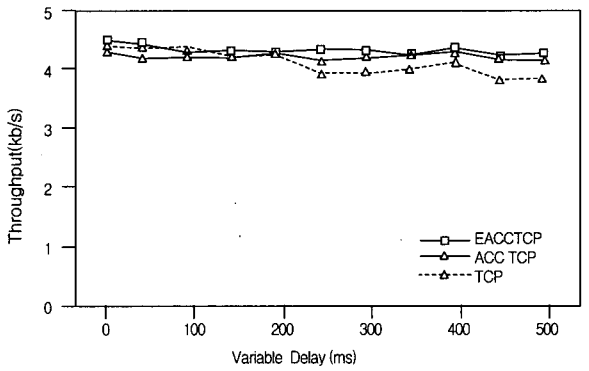


(그림 13) TCP Reno와 ACC TCP Reno(RED)

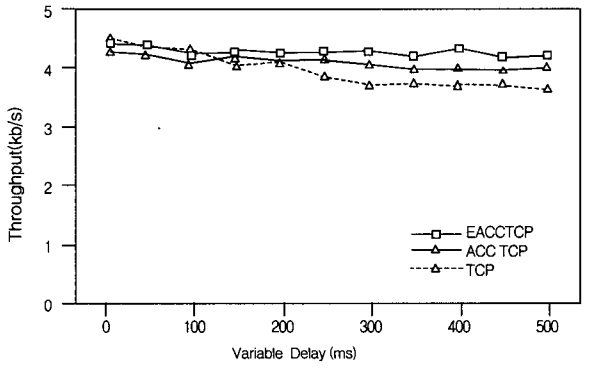
비록 TCP Tahoe에서의 문제점은 있으나 bandwidth * delay가 큰 네트워크 환경에서 ACC가 혼잡제어에 성능상의 이점을 제공함을 확인 할 수 있다.

4.3 EACC 모의 실험 결과

EACC는 엣지 라우터에서 각 전송자로 피드백된 정보로 전송자는 윈도우 크기를 조절하게 된다. ACC와 비교할 때 엣지 라우터에서 전송자까지의 지연시간이 작더라도 기존의 TCP와 같은 처리량을 나타냈다. ACC는 4.2절에 언급한 문제점 이외에 다른 문제점을 갖는다. ACC는 각 전송자가 네트워크에 보낼 수 있는 패킷의 크기를 알 수 없고, 패킷 누락이 발생한 전송자에 대해서만 코어 라우터에서 피드백이 이루어지게 된다. 또한 혼잡이 발생할 경우 특정 전송자의 트래픽만을 필터링하게 된다. 이는 어떤 전송자는 지속적으로 처리량에 영향을 받는 반면, 어떤 전송자는 지속적으로 높은 윈도우 크기로 전송 가능하기 때문에 공평성(fairness)에 문제점을 갖는다. EACC는 이를 개선하기 위해서 엣지 라우터와 코어 라우터 간의 통신(communication)을 사용하였다. (그림 14)~(그림 15)는 TCP Tahoe에 EACC를 적용했을 경우와 ACC를 적용했을 경우, 그리고 기존의 혼잡제어 방식을 적용했을 경우의 결과이다. 작은 bandwidth * delay의 경우에도 EACC는 기존의 방식과 거의 비슷한 결과를 얻었음을 알 수 있다. Bandwidth * delay가 클 경우 EACC는 기존의 방식뿐만 아니라 ACC보다 성능이 개선되었다.



(그림 14) TCP Tahoe, ACC, EACC (Droptail)

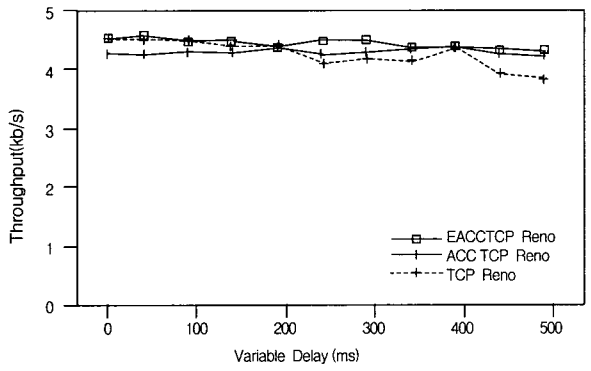


(그림 15) TCP Tahoe, ACC, EACC (RED)

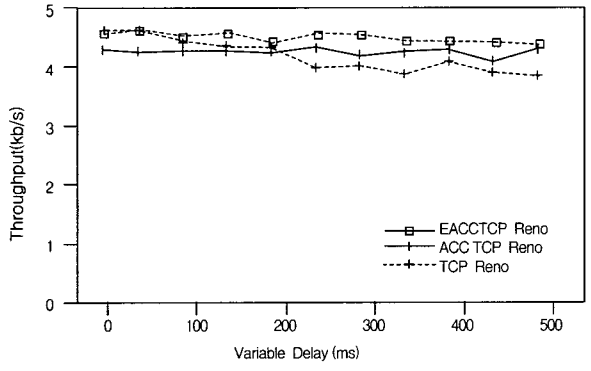
〈 표 1 〉 평균 Throughput과 표준편차 및 Throughput 증가(RED Queuing, TCP Tahoe)

Delay	TCP		ACC TCP		Chameleon		Throughput 증가	
	평 균	표준편차	평 균	표준편차	평 균	표준 편차	$\frac{ACC\ TCP}{TCP}-1$	$\frac{Chameleon}{TCP}-1$
10	4.63	0.350	4.40	0.196	4.56	0.051	-4.9%	-1.5%
50	4.49	0.286	4.35	0.228	4.52	0.041	-3.1%	0.6%
100	4.41	0.242	4.19	0.258	4.37	0.042	-4.9%	-0.9%
150	4.18	0.399	4.30	0.427	4.38	0.058	2.8%	4.7%
200	4.19	0.423	4.23	0.597	4.37	0.091	0.9%	4.2%
250	3.95	0.503	4.29	0.514	4.40	0.096	8.6%	11%
300	3.83	0.300	4.17	0.574	4.40	0.052	8.8%	14%
350	3.85	0.354	4.11	0.521	4.30	0.080	6.7%	11%
400	3.82	0.344	4.10	0.799	4.44	0.124	7.3%	16%
450	3.83	0.488	4.08	0.634	4.29	0.238	6.5%	12%
500	3.75	0.204	4.13	0.429	4.34	0.150	10%	15%

(그림 16)~(그림 17)은 TCP Reno의 경우이다. TCP Tahoe에서 얻은 성능상의 이점은 Reno의 경우에도 나타남을 확인 할 수 있다. EACC는 혼잡이 발생하여 코어 라우터에서 피드백 될 경우 엣지 라우터에서 각 전송자에게 네트워크에 전송 가능한 윈도우 크기를 전송해 주기 때문에 특정 전송자가 불이익을 받지 않게 된다. 또한 RTT에 따른 처리가 가능하기 때문에 bandwidth * delay의 크기에 무관하게 처리량이 높게 나타났다. <표 1> 은 기존 TCP와 ACC TCP 및 Chameleon의 평균과 표준편차, Throughput 증가량을 나타낸다. ACC TCP는 표에서 보는 바와 같이 지연시간이 길어짐에 따라 Throughput이 향상됨을 보여준다. Chameleon의 경우는 Delay시간이 작은 경우에도 ACC TCP에서와 같은 Throughput의 감소가 작을 뿐만 아니라 Delay 시간이 클 경우에도 Throughput의 증가율이 ACC TCP 보다 큼을 알 수 있다. Chameleon 큐의 표준편차 값이 ACC TCP보다 작기 때문에 보다 안정적임을 알 수 있다.



(그림 16) TCP Reno, ACC, EACC (Droptail)



(그림 17) TCP Reno, ACC, EACC (RED)

5. 결 론

기존의 혼잡제어 방식은 혼잡 발생 시점에서 혼잡에 대한 처리를 할 수 없다. 즉 혼잡이 발생한 후 패킷 누락에 대한 신호가 전송자에게 도착 했을 경우에 혼잡을 감지 하게 된다. 혼잡을 감지 하는데 걸리는 이런 제어시간이 길어지면 길어 질수록 혼잡은 더욱 악화 된다. 혼잡 발생과 동시에 혼잡에 대한 처리를 할 수 있다면 전송자는 보다 빠르게 혼잡에 대처 할 수 있을 것이다. 본 논문에서는 네트워크의 내부 정보를 이용하여 혼잡 발생 시점에서 혼잡을 제어하는 ACC와 EACC를 이용하여 혼잡을 제어 하였다. ACC와 EACC 모두 bandwidth * delay가 클 경우 기존의 혼잡 제어 방식보다 모의실험에서와 같이 액티브 라우터를 이용해서 혼잡을 제어할 경우 기존의 혼잡제어의 비효율성을 제거하여 성능이 향상됨을 알 수 있다. ACC는 TCP 혼잡제어 방식마다 단순히 누락시키고 이 정보를 각 전송자에게 알려주는 'Drop and Notice'을 이용하여 혼잡을 제어하였다[5]. 하지만 이와 같은 방법은 RTT에 따른 처리가 불가능하고 특정 전송자에게 불리한 처리를 할 가능성이 있으며 패킷 누락 방법에 있어서 문제점을 갖는다. EACC는 ACC의 단점을 해결하기위해 혼잡 처리방식에 라우터와의 통신을 추가하여 네트워크 내부에서의 정보를 이용하고 이를 각 라우터가 이용하여 ACC의 문제점을 해결했으며 엣지 라우터의 패킷 복사 기능으로 모든 전송자는 혼잡이 발생할 경우 네트워크 상태를 알 수 있다. EACC는 라우터

에서 누락된 정보이외에 라우터 내부의 통계적 정보나 연결된 트래픽에 대한 정보, 큐에 저장된 패킷의 개수 등을 이용해서 혼잡을 제어 했기 때문에 보다 안정적이고 효율적인 제어가 가능하다. 또한, EACC는 엣지 라우터에서 실시간 트래픽과 비 실시간 트래픽을 구별하여 처리 할 수 있는 구조를 제공함으로 네트워크 관리자로 하여금 보다 원활한 네트워크 관리를 제공 해 줄 수 있다. 이 구조는 엣지 라우터에서 설정한 정책에 따라서 코어 라우터가 그 정책에 따른 처리를 할 수 있는 DiffServ[6]와 유사한 구조를 제공 한다. DiffServ는 기존의 패킷 처리 방식과 같은 정적인 구조를 갖기 때문에 네트워크의 상태에 따른 자원의 신속한 관리가 불가능 하다. EACC는 네트워크의 상태에 따라서 관리자가 현재의 정책을 바꿀 수 있는 구조를 제공할 뿐만 아니라 액티브 라우터의 실행환경이 구축되어 있기 때문에 이종의 네트워크 도메인(domain)간의 처리의 일관성을 제공할 수 있다. 앞으로 본 논문의 모의실험 결과를 토대로 전체적인 라우터의 구성을 확장하여 모든 트래픽에 대해서 범용적으로 혼잡제어가 가능한 라우터 구성을 목표로 연구가 이루어져야 할 것이다.

참 고 문 헌

[1] Schwartz, B., A. W. Jackson, W. T. Strayer, W. Zhou, R. D. Rockwell, and C. Partridge, "Smart Packets : Applying Active Networks to Network Management," ACM Transactions on Computer Systems, Vol.18, No.1, pp.67-88, 2000.

[2] M. Parris, K. Jeffay, F.D. Smith, "Responsive vs. Unresponsive Traffic : Active Queue Management for a Better-Than-Best-Effort Service," Technical Report, September 2000. http://www.cs.unc.edu/~jeffay/papers/IEEE-Networks-01.pdf

[3] TJ Ott, TV Lakshman, and LH Wong, "SRED : stabilized RED," Proc. IEEE INFOCOM Conf., pp.1346-1355, Mar., 1999.

[4] Floyd, S. ; Fall, K., "Promoting the use of end-to-end congestion control in the Internet," Networking, IEEE/ACM Transactions on, Vol.7, Issue : 4 , pp.458-472, Aug., 1999.

[5] Theodore Faber, "ACC : Using Active Networking to Enhance Feedback Congestion Control Mechanisms," IEEE Network, IEEE, pp.61-65, May/June, 1998.

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

[7] Stevens, W., TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms, RFC 2001, January, 1997.

[8] S. Alexander, C. Gunter, A. Keromytix, G. Minden, D. Wetherall, B. Braden, A. Jackson, "The Active Network Encapsulation Protocol (ANEP)," 1997. http://www.cis.upenn.edu/~switchware/ANEP/

[9] Y. Yemini and S. de Silva, "Towards Programming Networks," IFIP/IEEE International Workshop on Distributed Systems : Operations and Management, L'Aquila, Italy, October, 1996.

[10] S. Bhattachariee, K. Calvert, and E. Zegura, "On Active Networking and Congestion," Technical Report GIT-CC-96-02, College of Computing, Georgia Tech., Atlanta, GA, 1996.

[11] S. McCanne, S. Floyd, K. Fall, UCB/LBL Network Simulator NS, 1996.

[12] S. Floyd, V. Jacobson, "Random Early Detection gateways for Congestion Avoidance," IEEE/ACMTrans. on Networking, Vol.1, No.4, pp.397-413, Aug., 1993.

[13] V. Jacobson, R. Braden & D. Borman, "TCP Extensions for High Performance," Network Working Group, IETF. RFC 1323. May, 1992.

[14] V. Jacobson, "Congestion Avoidance and Control," Computer Communication Review, Vol.18, No.4, pp.314-329, Aug., 1988.

최 기 현

e-mail : gyunee@ece.skku.ac.kr

2000년 성균관대학교 전기전자 및 컴퓨터 공학부 졸업(학사)

2002년 성균관대학교 전기전자 및 컴퓨터 공학과 졸업(석사)

2002년~현재 성균관대학교 정보통신공학부 박사과정 재학

관심분야 : Active Network, AQM, 혼잡제어 등

장 경 수

e-mail : ksjang@kic.ac.kr

1994년 성균관대학교 전기공학과 졸업 (학사)

1998년 성균관대학교 대학원 졸업(석사)

2000년 성균관대학교 대학원 전기전자 및 컴퓨터 공학과 박사과정 수료

2001년~현재 경인여자대학 컴퓨터정보기술학부 전임강사

1994년~1995년 LG산전㈜

관심분야 : ATM, Discrete Event Systems, Industrial Networks 등

신 호 진

e-mail : hjshin@ece.skku.ac.kr

1994년 성균관대학교 전기공학과 졸업 (학사)

1999년 성균관대학교 대학원 졸업(석사)

2001년 성균관대학교 대학원 전기전자 및 컴퓨터 공학과 박사과정 수료

2001년~현재 성균관대학교 대학원 정보통신 공학과 박사 학위 과정

1994년~1995년 삼성중공업㈜

관심분야 : ATM, Industrial Networks, Discrete Event Systems 등

신 동 렬

e-mail : drshin@ece.skku.ac.kr

1980년 성균관대학교 졸업(학사)

1982년 한국과학기술원(석사)

1992년 Georgia Institute of Tech. (Ph.D)

1994년~현재 성균관대학교 전기전자 및 컴퓨터공학부 부교수

1982년~1986년 Assistant researcher, Daewoo Heavy Ind., Limited

1992년~1994년 Senior researcher, Samsung Data Systems

관심분야 : Industrial Networks including MMS,ICCP, and Fields, High-Speed telecommun-ication networks with emphasis on ATM modeling and switching. Performance evaluation with markovian and queuing networks 등