동적 네트워크 상태정보 교환 오버헤드를 제거한 중앙 집중적 QoS 라우팅 구조

(A Centralized QoS Routing Architecture with No Dynamic Network State Information Exchange Overhead)

김 성 하 † 이 미 정 ††

(Sungha Kim) (Meejeong Lee)

요 약 본 논문에서는 라우팅 도메인 내의 모든 라우터들을 대신하여 라우트 서버가 QoS 경로 결정을 담당하도록 하는 중앙 집중적인 QoS 라우팅 구조를 제안한다. 라우트 서버는 QoS 경로를 할당하고 반환 받는 작업을 통해 스스로 QoS 경로 계산에 필요한 동적인 링크 QoS 상태 정보를 파악하고 유지한다. 따라서, 제안하는 QoS 라우팅 구조에서는 동적 네트워크 상태 정보 교환으로 인한 프로토콜 오버헤드를 제거하였다. 또한, 이와 같은 방식으로 네트워크 상태 정보를 유지함으로써 정확한 네트워크 상태 정보를 이용하여 경로 계산을 수행할 수 있기 때문에 라우팅 성능 또한 크게 향상시킬 수 있다. 본 논문에서는 라우트 서버의 경로 계산 오버헤드를 감소시키기 위한 경로 캐싱 스킴들을 제안하고, 시뮬레이션을 통해 그 성능을 평가하였다. 시뮬레이션 결과, 제안하는 캐싱 스킴을 통해 라우트 서버의 오버헤드가 크게 줄어드는 것을 확인할 수 있었다. 뿐만 아니라 기존에 제안된 다양한 분산 QoS 라우팅 스킴들과도 성능을 비교하였는데, 그 결과 제안하는 서버 기반 QoS 라우팅 스킴이 라우팅 성능을 크게 향상시킬 뿐 아니라, 라우팅 오버헤드 측면에서도 우수함을 볼 수 있었다.

키워드 : 서비스 품질, QoS 라우팅, 서버 기반 구조, 캐싱, 동적 네트워크 상태 정보

Abstract We propose centralized server based QoS routing schemes, where a route server is responsible for determining QoS paths on behalf of all the routers in a routing domain. In the proposed server based schemes, the dynamic link QoS state information, which is required for a QoS path computation, is implicitly maintained at route server as it assigns or gets back QoS paths. By maintaining the network state information this way, we may not only eliminate the overhead to exchange network state update message but also achieve higher routing performance by utilizing accurate network state information in path computation. We discuss path caching techniques for reducing the amount of path computation overhead at the route server, and evaluate the performance of the proposed schemes using simulation. The simulation results show that the path caching schemes may significantly reduce the route server load. The proposed schemes are also compared to the distributed QoS routing schemes proposed in the literature. It has been shown that the proposed server based schemes not only enhance the routing performance, but they are also competitive with respect to routing overheads.

Key words : QoS, QoS routing, server based architecture, caching, dynamic network state update

1. 서 론

차세대 네트워크 서비스에 있어서는 사용자 혹은 응용 프로그램 수준의 QoS를 보장하는 것이 매우 중요하다. 이에 플로우가 제시한 QoS를 지원할 수 있는 경로를 찾아 서비스를 제공하는 QoS 라우팅에 관한 연구가 활발히 진행되고 있다. QoS 라우팅은 플로우가 요구하는

· 본 연구는 한국과학재단 목적기초연구(R04-2000-000-00078) 지원으로 수행되었음.

† 비 회 원 : 이화여자대학교 컴퓨터학과
flower00@ewha.ac.kr
†† 정 회 원 : 이화여자대학교 컴퓨터학과 교수
lmj@ewha.ac.kr

논문접수 : 2002년 3월 21일
심사완료 : 2002년 5월 23일

QoS를 보장함과 동시에 네트워크 성능을 크게 향상시킬 수 있다. 그러나 QoS 라우팅은 이러한 장점들에도 불구하고 보다 복잡하고 빈번한 경로 계산으로 인한 계산 오버헤드 및 경로 계산에 필요한 네트워크 상태 정보의 교환을 위한 프로토콜 오버헤드와 같은 추가적인 라우팅 비용으로 인해 실제 IP 네트워크에서 널리 구현되지 못하고 있는 실정이다.

최근 이러한 QoS 라우팅의 비용 감소에 초점을 맞춘 연구들이 많이 발표되었는데, 이들 연구를 살펴보면 QoS 경로 계산의 프로세싱 오버헤드를 감소시키기 위한 사전 경로 계산(path pre-computation) 및 경로 캐슁(caching) 스킴들에 대한 것들과[1, 2, 3, 4, 5, 6, 7], QoS 라우팅의 프로토콜 오버헤드를 줄이기 위한 네트워크 상태 정보 갱신 정책에 관한 연구들 등이 있다[8, 9]. 뿐만 아니라 다른 라우터들로부터 QoS 라우팅으로 인한 오버헤드를 덜기 위해 서버가 QoS 라우팅 작업의 대부분을 담당하는 서버 기반 QoS 라우팅 방식도 제안된 바 있다[10].

사전 경로 계산 방안은 기존의 최선 서비스 라우팅과 유사하게 모든 목적지에 대하여 미리 경로를 계산하여 라우팅 테이블에 저장해 두되 한 개 이상의 서로 다른 QoS를 제공하는 경로들을 계산하고 이들을 일정 조건이 유지되는 동안 계속 사용하는 방식이다. 사전 경로 계산은 이와 같이 한 번 계산된 경로를 여러 번 사용함으로써 QoS 플로우가 발생할 때마다 매번 새로이 경로를 계산하는 오버헤드를 줄이고자 하는 방법이다. 캐슁 방안도 이와 유사하게 한 번 계산된 경로를 캐쉬에 저장해 두었다가 이후 동일한 목적지에 대한 라우팅 요청이 발생하면 새로 경로를 계산하지 않고 캐쉬에 저장되어 있는 경로를 다시 사용하는 방법이다. 캐슁 방안은 사전 경로 계산 방안과는 달리 미리 모든 목적지에 대하여 경로를 계산해 두지 않으므로 실제적으로 QoS 플로우의 목적지가 아닌 노드들에 대해 경로를 계산하는 오버헤드를 피할 수 있다.

네트워크 상태 정보 갱신 정책에 관한 연구에서는 QoS 라우팅을 위한 동적인 네트워크의 상태 정보를 주고 받는 프로토콜 오버헤드를 줄이기 위해 링크 혹은 라우터의 상태에 변화가 있을 때마다 이를 알리는 것을 피하고, 일정 기간마다 혹은 네트워크 상태 변화의 정도가 특정 조건에 이르렀을 때에만 이를 알리기 위한 제어 메시지를 주고받는 방안들이 제안되었다. 또한, 이들 방안이 QoS 라우팅의 성능 및 오버헤드 감소에 어떤 영향을 미치는지에 관하여 연구가 이루어졌다.

그런데 이러한 기존의 QoS 라우팅 스킴에서는 공통적으로 QoS 경로 계산에 사용되는 네트워크 상태 정보를 얻기 위한 라우터들간 네트워크 상태 정보 교환이 요구된다. 그런데 이러한 패러다임의 라우팅 스킴에서는 근본적으로 라우팅 성능과 네트워크 상태 정보 갱신의 빈도 사이에 서로 상반되는 관계가 발생한다. 또한, 변화가 생길 때마다 링크 QoS 상태 정보를 갱신하는 극단적인 경우를 제외하고는 근본적으로 부정확한 정보를 기반으로 QoS 라우팅 결정을 내리게 된다.

이에 본 논문에서는 이와 같은 문제들을 피할 수 있는 서버 기반 QoS 라우팅 메카니즘을 제안한다. 본 논문에서 제안하는 서버 기반 QoS 라우팅 스킴은 [10]에서 제안된 서버 기반 QoS 라우팅 스킴과 마찬가지로 라우트 서버를 두어 라우팅 도메인 내의 모든 QoS 경로 계산을 담당하도록 한다. 즉, 도메인 내의 다른 라우터들은 모두 클라이언트가 되어 QoS 플로우에 대해 라우트 서버에게 경로를 요청한다. 그러나 제안하는 스킴에서는 [10]의 스킴과는 달리 라우트 서버가 원격 라우터들로부터 QoS 라우팅을 위한 동적인 링크 상태 정보를 수집하지 않는다. 대신 라우트 서버는 클라이언트 라우터들에게 QoS 경로를 할당하고 클라이언트 라우터로부터 다시 경로를 반환 받으면서 자신이 QoS 경로 계산을 위해 유지하고 있는 동적 링크 상태 정보 데이터베이스를 갱신한다. 제안하는 스킴에서는 라우트 서버의 독자적 네트워크 상태 정보 데이터베이스 유지가 가능하도록 하기 위해 일정량의 네트워크 자원이 QoS 플로우에게 예약되었거나 QoS 플로우가 최선(best effort) 트래픽에 비해 높은 우선 순위로 서비스된다고 가정한다. 이러한 가정 하에서는 라우트 서버가 QoS 플로우에게 가용한 네트워크 자원의 양을 알 수 있기 때문이다. 즉, 라우트 서버는 네트워크 용량 및 현재 사용량을 모두 알기 때문에 다른 라우터들로부터 링크 QoS 상태 정보를 수집하지 않고도 동적인 네트워크 상태 정보를 정확히 유지할 수 있다.

따라서 제안하는 서버 기반 QoS 라우팅 스킴에서는 원격 라우터로부터 링크 QoS 상태 정보를 수집하기 위한 프로토콜 오버헤드가 없다. 뿐만 아니라 항상 정확한 링크 QoS 상태 정보를 기반으로 라우팅 결정을 내리므로 라우팅 성능 및 네트워크 자원 활용률을 향상시킨다. 또한, [10]에서의 서버 기반 방식과 마찬가지로 라우트 서버에서 QoS 라우팅의 대부분의 작업을 수행하기 때문에 다른 라우터들에 요구되는 변화 및 오버헤드가 최소이다.

서버 기반 QoS 라우팅 구조는 IETF(Internet Engineering Task Force)가 제안한 정책 기반(policy based) 네트워크 구조에 매우 적합하다[11, 12]. 정책

기반 네트워크에서는 네트워크로 들어오는 전송 요구를 처리하기 위해 항상 원격의 정책 서버와 접속해야 하는데, QoS 라우트 서버를 정책 서버와 함께 둔다면 새 플로우에 대한 경로 질의 및 응답이 정책 서버에 대한 질의 응답과 함께 이루어질 수 있다.

제안하는 스킴은 플로우 기반 소스 라우팅을 사용하는 네트워크 환경에 가장 적합하다. 제안하는 스킴에서 프로토콜 오버헤드의 가장 큰 원인은 라우트 서버와 클라이언트 라우터 사이에 교환하는 경로 요청 및 응답 메시지라 할 수 있는데, 이런 종류의 오버헤드는 라우팅 결정이 각 홉마다 패킷별로 이루어지는 경우보다 각 송신지 라우터에서 플로우별로 이루어질 때 최소화되기 때문이다. 역시 이와 동일한 종류의 오버헤드를 줄이기 위해 제안하는 스킴에서는 라우터 서버는 QoS 플로우에 대한 라우팅 결정만을 담당하고, 최선 서비스에 해당하는 패킷들은 기존의 라우팅 모듈(예: OSPF)에 의해 라우팅된다고 가정한다.

라우트 서버와 클라이언트 라우터간의 경로 요청 및 응답 메시지 교환으로 인한 프로토콜 오버헤드와 함께 제안하는 스킴에서 가장 문제가 되는 오버헤드는 라우트 서버에서의 프로세싱 및 스토리지 오버헤드이다. 본 논문에서는 라우트 서버에서의 경로 계산 오버헤드를 감소시키기 위한 방안으로서 라우트 서버에서의 경로 캐슁 방법을 제안하고 그 성능을 평가하였다. 제안하는 스킴은 일반적으로 서버 기반 접근 방식이 갖는 서버에서의 병목 혹은 단일 지점에 의한 실패와 같은 문제점들이 있으나, 본 논문에서는 이러한 문제점들을 다루지 않고 향후 연구 과제로 두도록 한다. 또한, 제안하는 QoS 라우팅 스킴에서는 링크 실패와 같은 토폴로지 변화에 대한 정보를 얻기 위한 메카니즘은 따로 제공하지 않고, 라우트 서버에서의 QoS 라우팅 모듈이 최선 트래픽 라우팅 모듈로부터 이와 같은 정보를 얻는다고 가정한다.

본 논문의 구성은 다음과 같다. 1장의 서론에 이어 2장에서는 제안하는 서버 기반 QoS 라우팅 구조에 대해 자세히 논의한다. 3장에서는 시뮬레이션 환경을 설명하고, 시뮬레이션 결과를 분석한다. 마지막으로 4장에서는 결론을 제시한다.

2. 서버 기반 QoS 라우팅

본 장에서는 먼저 제안하는 스킴의 구성 요소와 그들 간의 통신 형태 및 기본적인 동작원리와 같은 전체적인 구조를 살펴본 후, 경로 계산 알고리즘 및 이로 인한 오버헤드를 감소시키기 위한 캐슁 방법들에 대해 자세히 설명하겠다.

2.1 서버 기반 QoS 라우팅의 구조

그림 1은 제안하는 서버 기반 QoS 라우팅 구조의 기본적인 구성 요소 및 라우트 서버와 클라이언트 라우터들 간의 통신 형태를 도식화한 것이다. 제안하는 서버 기반 QoS 라우팅 구조는 그림 1에서와 같이 라우트 서버와 클라이언트 두 가지 요소로 구성된다. 라우트 서버는 네트워크 토폴로지 정보 및 모든 링크들의 링크 QoS 상태 정보를 유지하는 NTDB(Network Topology Data Base)와 QoS 경로 정보를 일시적으로 저장하여 재 사용할 수 있도록 하는 RTC(Routing Table Cache)를 이용하여 클라이언트의 QoS 경로 질의에 대해 응답한다.

클라이언트 라우터에 새로운 QoS 플로우의 전송 요구가 발생하면 클라이언트는 라우트 서버에게 해당 QoS 요구사항을 실은 Path Query(PQ) 메시지를 보낸다(그림 1의 (a)). 이 PQ 메시지를 받은 라우트 서버는 NTDB의 현재 네트워크 토폴로지 상태 정보를 기반으로 QoS 요구사항을 만족하는 경로를 새로 계산하거나, RTC를 검색하여 적정 경로 정보를 선택하여 클라이언트에게 Path Reply(PR) 메시지로 응답한다(그림 1의 (b)). 이 PR 메시지는 클라이언트가 질의한 목적지에 대하여 명시적인 소스 라우팅 정보를 알려준다. 또한, 라우트 서버는 해당 경로에 대한 자원 할당 사실을 NTDB에 반영하기 위해 NTDB를 갱신한다. 만약 적정 경로가 존재하지 않는다면 Path Block(PB) 메시지를 클라이언트에게 보낸다(그림 1의 (c)). QoS 플로우의 전송이 끝나면 클라이언트는 이를 라우트 서버에게 보고하기 위해 Path Return(PT) 메시지를 보낸다(그림 1의 (d)). PT는 라우트 서버가 할당했던 경로의 자원이 다시 가용해졌음을 의미하므로 라우트 서버는 PR 메시

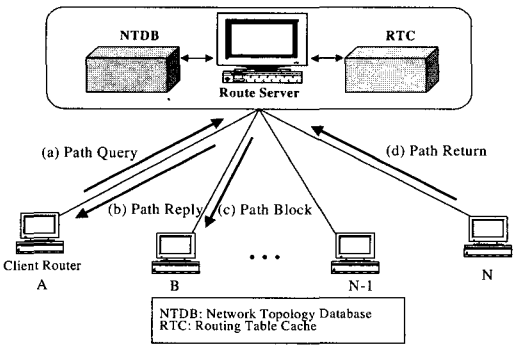


그림 1 제안하는 서버 기반 QoS 라우팅 구조의 구성 요소 및 그들 간의 통신 형태

지를 보낼 때와 마찬가지로 반환된 경로 상의 가용 자원 정보 변화를 반영하기 위해 NTDB를 갱신한다. 이와 같은 클라이언트/서버 교류는 신뢰성 있게 이루어져야 하는데, 이를 위해 COPS(Common Open Policy Service) 프로토콜을 이용해 이들 메시지를 교환할 수 있다. COPS 프로토콜은 정책 기반 네트워크 환경에서 클라이언트와 서버의 신뢰성 있는 메시지 교환을 위해 정의되었다[13].

2.2 경로 계산 및 캐싱

제안하는 스킴에서는 [2]의 QoS를 위한 OSPF 확장 방안에서와 마찬가지로 QoS에 대한 요구가 대역폭으로 표시된다고 가정한다. 따라서 라우트 서버가 파악해야 하는 QoS 메트릭은 각 링크의 가용 대역폭이다. 경로 계산 알고리즘으로는 변형된 Dijkstra의 최단 경로 알고리즘[1, 5, 6]과 QoS를 위해 확장된 Bellman Ford 알고리즘[2]을 사용한다. 변형된 Dijkstra 알고리즘이란 QoS 경로를 계산하는 시점에서 링크의 가용 대역폭이 클라이언트가 요구한 대역폭보다 작게 남아 있는 링크들을 모두 제거한 후, 남은 네트워크 토폴로지 상에 Dijkstra 알고리즘을 적용하여 widest-shortest 경로를 계산하는 것을 말한다. Bellman Ford 알고리즘은 모든 가능한 홉 수에 대하여 홉 수 별로 적정 경로를 계산한다.

일반적으로 네트워크 상에서 일어나는 통신의 송·수신지 쌍 분포에는 상당한 지역성이 있다. 만약 본 논문의 서버 기반 QoS 라우팅에서 라우트 서버가 사전 경로 계산 정책에 기반하여 라우팅 테이블을 미리 계산하여 유지한다면, m을 라우팅 도메인내의 모든 노드의 수라고 가정했을 때 m^2만큼의 송·수신지 쌍에 대한 QoS 경로를 모두 라우팅 테이블에 유지해야 한다. 그러나 통신 형태의 지역성으로 인해 이중 상당수의 라우팅 테이블 엔트리들은 거의 혹은 전혀 사용되지 않게 될 것이다. 이에 반해서 캐싱 스킴을 사용하는 경우에는 이러한 통신 형태의 지역성으로 인해 적은 수의 캐쉬 엔트리만으로도 라우트 서버의 경로 계산 오버헤드를 상당히 줄일 수 있다. 따라서 본 논문에서는 라우트 서버의 오버헤드를 줄이기 위한 방안으로 경로 캐싱을 채택하였다.

RTC는 경로 캐싱을 위해 라우트 서버에서 사용하는 캐쉬이다. RTC에는 계산된 QoS 경로 정보를 임시로 저장하며 그 경로를 구성하고 있는 링크들의 리스트만을 유지한다. 즉, 가용 대역폭과 같은 해당 경로의 QoS 상태 정보는 따로 저장하지 않고, 대신 클라이언트로부터 경로 요청이 들어와 RTC에서 적정 경로를 선택할 때마다 NTDB를 검색함으로써 파악한다. 임의의 클라이언트로부터 특정 수신지로의 QoS 경로를 요청 받은 라우트 서버는 우선 현재 RTC에 해당 요구에 대한 QoS 경로 정보가 이미 존재하는지 판단하기 위해 RTC를 검색한다. RTC에 해당 경로 정보가 존재한다면 해당 경로를 구성하고 있는 모든 링크들에 대해 NTDB에서의 현재 가용 대역폭을 검색하여 그 경로의 현재 병목 대역폭을 확인한다. 라우트 서버의 RTC 및 NTDB 검색 작업을 효율적으로 하기 위해 각 자료 구조는 그림 2와 같은 형태를 갖는다.

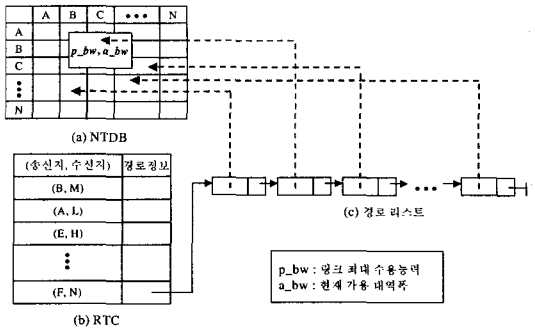


그림 2 라우트 서버가 유지하는 자료 구조 및 그 연결 관계

RTC의 크기는 K와 n의 두 가지 파라미터로 제한된다. K는 RTC에서 유지할 수 있는 송신지, 수신지 쌍의 수이고, n은 하나의 송·수신지 쌍에 대해 유지할 수 있는 경로 개수를 말한다. 클라이언트로부터 경로 요청이 들어오면 RTC에서 해당 송·수신지 쌍에 대한 엔트리를 찾아 거기에 저장되어 있는 n개의 경로들 중 최단 경로부터 우선적으로 선택하여 적정성을 판단한다. 만약 해당 엔트리의 n개의 경로들 중 적정한 경로가 하나도 존재하지 않는다면 라우트 서버는 현재의 네트워크 토폴로지 정보를 기반으로 새로운 QoS 경로를 계산해야 한다. 변형된 Dijkstra 알고리즘을 사용하여 경로 계산을 수행한다면 현재 요구에 만족하는 단 하나의 경로만이 계산될 것이다. 이 때 RTC의 해당 송·수신지 관련 엔트리에 저장되어 있는 경로 정보의 개수가 n보다 작다면 새로 계산된 경로 정보는 단순히 그 RTC 엔트리에 추가될 것이다. 그러나 해당 송·수신지 관련 RTC 엔트리에 n개의 경로가 이미 저장되어 있다면 그 가운데 가장 가용 대역폭 크기가 가장 적은 경로를 선택하여 교체한다. 이것은 남아 있는 가용 대역폭의 크기가 적을수록 앞으로 추가적으로 플로우를 수용할 수 있는 가능성이 적은 경로이기 때문이다. Bellman Ford 알고리즘을 사용하는 경우에는 해당 RTC 엔트리의 n개의 경로가 모두 한꺼번에 교체된다.

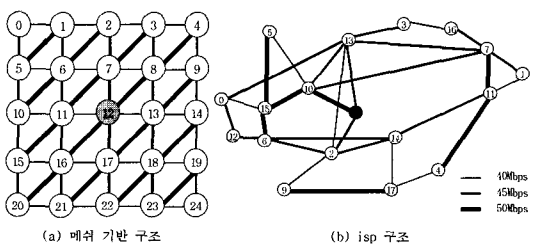
RTC에 새로운 송신지, 수신지 쌍에 대한 QoS 경로 정보를 추가해야 할 때 RTC에 이미 K개의 송·수신지 쌍에 대한 엔트리가 존재한다면 송·수신지 쌍 수준의 RTC 엔트리 교체가 일어나야 한다. 만약 통신 형태에 있어서 통계적인 지역성이 존재한다면 캐쉥 스킴은 앞으로 빈번하게 참조될 엔트리가 희생자로 뽑혀 교체되지 않도록 이런 지역성을 반영해야 한다. 임의의 몇몇 송·수신지 쌍에 대하여 QoS 플로우가 보다 빈번하게 발생된다는 가정 하에 본 논문에서는 그런 송·수신지에 해당하는 엔트리가 캐쉬에 남아있도록 교체 정책을 정하였다. 이를 위해 RTC의 각 송·수신지 엔트리에는 참조 회수를 나타내는 카운터를 유지하고 카운터의 참조 횟수가 가장 적은 송신지·수신지 쌍을 교체하였다. 또한 정기적으로 이 카운터 값을 오른쪽으로 쉬프팅함으로써 한동안 빈번하게 참조되었으나 더 이상 QoS 플로우 발생이 없는 송·수신지 쌍이 계속해서 RTC 엔트리를 차지하는 것을 방지하였다.

3. 서버 기반 QoS 라우팅 성능 평가

본 논문에서는 사건 주도형 (event driven) 시뮬레이션을 통해 제안하는 서버 기반 QoS 라우팅의 성능을 평가하였다. 시뮬레이션은 C 언어로 구현되었으며 LINUX 기반의 PC에서 수행되었다. 시뮬레이션은 크게 제안하는 스킴들간의 비교와 제안하는 스킴과 기존에 제안된 QoS 라우팅 스킴들과의 비교로 나뉜다. 우선 시뮬레이션 환경에 대해 설명하고, 시뮬레이션 결과를 분석하도록 한다.

3.1 시뮬레이션 환경

시뮬레이션에서 사용된 네트워크 토폴로지는 그림 3과 같은 메쉬 기반 구조와 일반적인 Internet Service Provider(ISP) 구조이다. 라우트 서버는 클라이언트와의 통신비용을 감소시키기 위해 토폴로지 중앙에 위치하도록 하였다. 네트워크 크기 변화에 따른 성능 변화를 보기 위한 실험을 제외한 모든 시뮬레이션은 그림 3과 같은 5×5 메쉬와 ISP에서 수행하였다. 링크의 대역폭은



(a) 메쉬 기반 구조 (b) isp 구조

그림 3 네트워크 토폴로지

메쉬에서는 수평, 수직, 대각선 방향으로 각각 40, 45, 50Mbps를 분포시켰고 ISP에서는 이를 그림 3(b)에서와 같이 임의로 분포시켰다. 이들 링크 및 대역폭은 QoS 트래픽만을 위해서 할당된 것만을 표시한 것이다.

각 노드에서의 세션 발생은 독립적으로 이루어지며 포아송(Poisson) 분포를 따른다. 각 세션의 요구 대역폭은 최소 64Kbps, 최대 6Mbps의 범위에서 균일하게 결정된다. 세션 길이는 지수 분포 랜덤 변수 (exponentially distributed random variable)로 평균 세션 길이는 3분이다. 네트워크에 부가되는 트래픽 로드를 변화시키기 위해 평균 세션 발생 간격을 0.2에서 0.6의 범위에서 변화시켰다. 본 논문에서는 가능한 송·수신지 쌍들 중 10%를 랜덤하게 추출하여 이 쌍들간에 전체 세션의 90%가 발생하도록 하였다.

3.2 캐쉥 스킴 평가

라우팅 성능을 평가하기 위해서는 BAR(Bandwidth Acceptance Ratio)을 측정하였고, 경로 계산 오버헤드를 분석하기 위해서는 PCR(Path Computation Ratio)을 측정하였다. BAR과 PCR은 각각은 다음과 같이 계산된다:

$$BAR = \frac{\sum_{i \in Accepted\ Query} bandwidth\ (i)}{\sum_{i \in Rrequested\ Query} bandwidth\ (i)},$$

$$PCR = \frac{the\ number\ of\ Path\ Computation}{the\ number\ of\ Requested\ Query}$$

표 1, 2는 각각 메쉬 및 ISP 구조에서 n값이 변화함에 따른 제안하는 서버 기반 QoS 라우팅 스킴들의 성능 변화를 보여 준 것이다. 이 때 K값은 메쉬와 ISP 구조에서 각각 250과 40으로 고정하였다. 표 1, 2를 보면 n값이 증가하여도 PCR은 크게 줄어들지 않는데, 이는 n값이 커질수록 경로 선택의 효율성이 줄어들기 때문이다. 한 송·수신지 쌍에서 저장하는 경로의 수가 늘어나면 임의의 한 경로 요청을 처리하는 데 있어 계산 오버헤드를 줄일 수 있지만, 비효율적인 경로 선택으로 인해 이후의 다른 경로 요청들에 대해서는 오히려 새로운 경로 계산 수행의 가능성을 높이는 결과를 초래하게 된다. 경로 계산 알고리즘으로 Bellman Ford 알고리즘을 사용하는 경우에는 Bellman Ford 알고리즘의 특성상 경로를 계산할 때마다 경로 길이가 한 홉인 경우로부터 시작해 점진적으로 최대 경로 길이까지 각 홉수 별로 가능한 경로를 계산해 나기가 때문에 어차피 계산된 이들 경로를 모두 저장해 둘 수 있도록 n을 설정하였다. Bellman Ford 경로 계산 알고리즘을 사용하는 캐쉥 스킴의 경우 비효율적인 경로 계산 및 선택으로 인해 계산 오버헤드가 큰 동시에 라우팅 성능도 가장 낮다.

표 1 n 변화에 따른 제안하는 서버 기반 QoS 라우팅 성능 비교 (메쉬, 90% 핫 트래픽)

(a) 대역폭 수용률(BAR)

스킴 / 평균 세션간격(초)	Dijkstra (n = 1)	Dijkstra (n = 3)	Dijkstra (n = 6)	Bellman Ford (n = 9)
0.2	0.646613	0.645050	0.642022	0.590061
0.3	0.876526	0.880073	0.876067	0.720929
0.4	0.994584	0.995269	0.994740	0.815564
0.5	0.999908	0.999908	0.999908	0.872478
0.6	0.999965	0.999965	0.999965	0.913831

(b)경로 계산 오버헤드(PCR)

스킴 / 평균 세션간격(초)	Dijkstra (n = 1)	Dijkstra (n = 3)	Dijkstra (n = 6)	Bellman Ford (n = 9)
0.2	0.663810	0.657341	0.662509	0.715191
0.3	0.665837	0.668466	0.671683	0.599267
0.4	0.405337	0.403132	0.403393	0.509679
0.5	0.263818	0.261161	0.260416	0.434663
0.6	0.207252	0.209494	0.217788	0.361654

표 2 n 변화에 따른 제안하는 서버 기반 QoS 라우팅 성능 비교 (ISP, 80% 핫 트래픽)

(a)대역폭 수용률(BAR)

스킴 / 평균 세션간격(초)	Dijkstra (n = 1)	Dijkstra (n = 3)	Dijkstra (n = 6)	Bellman Ford (n = 9)
0.2	0.794980	0.800788	0.798988	0.731165
0.3	0.872222	0.877407	0.875953	0.794203
0.4	0.931448	0.929243	0.926939	0.843271
0.5	0.957371	0.957089	0.959494	0.870634
0.6	0.970142	0.970366	0.970517	0.903589

(b)경로 계산 오버헤드(PCR)

스킴 / 평균 세션간격(초)	Dijkstra (n = 1)	Dijkstra (n = 3)	Dijkstra (n = 6)	Bellman Ford (n = 9)
0.2	0.511965	0.509107	0.512246	0.544751
0.3	0.464510	0.453203	0.473987	0.470980
0.4	0.399328	0.395446	0.394401	0.417799
0.5	0.352283	0.329829	0.330334	0.378774
0.6	0.313955	0.286569	0.300495	0.344144

표 3 K 변화에 따른 제안하는 서버 기반 QoS 라우팅의 성능 평가(메쉬, 90% 핫 트래픽)

(a)대역폭 수용률(BAR)

K / 평균 세션간격(초)	K = 0 (on-demand)	K = 100	K = 250	K = 400	K = all
0.2	0.644213	0.643369	0.646613	0.644558	0.640610
0.3	0.881772	0.875310	0.876526	0.875381	0.873176
0.4	0.994555	0.994820	0.994584	0.995330	0.995223
0.5	0.999908	0.999908	0.999908	0.999908	0.999908
0.6	0.999965	0.999965	0.999965	0.999965	0.999965

(b)경로 계산 오버헤드(PCR)

K / 평균 세션간격(초)	K = 0 (on-demand)	K = 100	K = 250	K = 400	K = all
0.2	1.00	0.687065	0.663810	0.639086	0.545470
0.3	1.00	0.735483	0.665837	0.653138	0.549172
0.4	1.00	0.676396	0.405337	0.383810	0.345355
0.5	1.00	0.628623	0.263818	0.249464	0.221549
0.6	1.00	0.597545	0.207252	0.189262	0.165947

표 4 K 변화에 따른 제안하는 서버 기반 QoS 라우팅의 성능 비교(ISP, 80% 핫 트래픽)

(a)대역폭 수용률(BAR)

K / 평균 세션간격(초)	K = 0 (on-demand)	K = 15	K = 40	K = 60	K = all
0.2	0.804747	0.798035	0.794980	0.796358	0.800421
0.3	0.874503	0.873298	0.872222	0.875190	0.870383
0.4	0.930651	0.932643	0.931448	0.929884	0.928386
0.5	0.961173	0.960390	0.957371	0.957431	0.958701
0.6	0.971409	0.972011	0.970142	0.972702	0.970629

(b)경로 계산 오버헤드(PCR)

K / 평균 세션간격(초)	K = 0 (on-demand)	K = 15	K = 40	K = 60	K = all
0.2	1.00	0.573614	0.511965	0.482486	0.348372
0.3	1.00	0.515490	0.464510	0.435163	0.299346
0.4	1.00	0.457742	0.399328	0.374767	0.239044
0.5	1.00	0.446809	0.352283	0.319906	0.175847
0.6	1.00	0.397514	0.313955	0.283017	0.145341

표 2의 ISP에서의 결과는 트래픽 집중률의 차이 등으로 인해 절대적인 결과 값은 표 1의 메쉬 구조 경우와 차이가 나지만 전체적인 경향은 표 1의 메쉬 구조에서와의 결과와 동일하다.

표 3, 4는 K값 변화에 따른 캐싱 스킴들의 성능 변화를 비교한 것이다. K=0의 경우는 요구기반(on-demand) 경로 계산을 의미한다. 표 3, 4의 (a)를 보면 제안하는 캐싱 스킴으로 인해 거의 라우팅 성능(BAR)이 저하되지 않을 뿐 아니라 캐쉬의 크기가 라우팅 성능에 크게 영향을 미치지 않음을 알 수 있다. 표 3과 4의 (b)는 K 값 변화에 따른 경로 계산 오버헤드(PCR)를 보인 것인데 K값이 QoS 플로우가 상대적으로 빈번하게 발생하는 송·수신지 쌍의 수(메쉬와 ISP 구조에서 각각 250개와 40개) 이상으로 커지면 K 값 증가에 따른 PCR 감소 정도 크지 않음을 볼 수 있다. 즉, K 값이 최소한 QoS 플로우가 상대적으로 빈번하게

발생하는 송·수신지 쌍의 개수 정도만 된다면 효과적으로 *PCR*을 감소시킬 수 있음을 볼 수 있다. 표 3의 메쉬에서의 결과와 표 4의 ISP에서의 결과를 비교해보면, 앞의 표 1, 2에서와 마찬가지로 수치적인 차이는 있으나 그 경향은 일치함을 알 수 있다.

3.3 기존의 분산 QoS 라우팅 스킴들과의 비교

그림 4, 5와 그림 6은 제안하는 경로 캐싱 스킴 가운데 n=1인 경우와 기존에 제안된 분산 QoS 라우팅 스킴들과의 성능을 비교한 것이다. 비교 대상이 된 기존의 분산 QoS 라우팅 스킴들은 모두 OSPF에 기반한 QoS 라우팅 스킴들로서 임계치 기반 및 지수 클래스 기반 네트워크 상태 정보 갱신 정책을 사용하는 스킴들이다[9]. 임계치 기반 방식은 가용 대역폭 값이 마지막으로 선전한 가용 대역폭 값의 일정 비율(임계치) 이상으로 변하면 다시 해당 링크에 대한 가용 대역폭 정보를 선전하는 방식이다. 한편, 지수 클래스 기반 방식은 가용 대역폭 값을 몇 개의 클래스로 분류하고 가용 대역폭 값이 클래스 경계를 넘어 변화하게 되면 다시 해당 링크의 가용 대역폭 값을 선전하는 방식인데, 각 클래스의 범위는 '기본 클래스 크기' 및 '클래스 성장 파라미터'라는 두 파라미터에 의하여 결정된다. 큰 가용 대역폭 값일수록 하나의 클래스에 해당하는 가용 대역폭 값의 범위(클래스의 최대 가용 대역폭 값 - 클래스의 최소 가용 대역폭 값)가 더 커지게 되는데, '기본 클래스 크기'는 가장 작은 크기의 클래스의 크기로서 하나의 QoS 플로우가 요구할 수 있는 최대 대역폭 요구량의 몇 %인가로 표시되고, '클래스 성장 파라미터'는 임의의 클래스의 범위가 그 전 단계 클래스 범위의 몇 배인지를 의미한다. 본 논문의 시뮬레이션에서는 기존의 분산 QoS 라우팅 스킴들의 파라미터 값에 대하여 이들 정책에 관한 연구에서 가장 일반적으로 권고하는 값들을 사용하였다. 임계치 기반 갱신 정책의 경우 임계치 값으로 60%를, 지수 클래스 기반 갱신 정책의 경우 기본 클래스 크기 및 클래스 성장 파라미터로 각각 200%와 2를 사용하였다. 뿐만 아니라 임계치 및 지수 클래스 기반 갱신 정책과 클램프 다운(clamp down) 타이머 정책을 조합한 경우도 비교하여 보았다. 클램프 다운 타이머는 가용 대역폭 선전 오버헤드를 제어하기 위하여 선전 후 일정 시간(클램프 다운 타임)이 경과한 경우에만 새로운 선전을 할 수 있도록 하는 방법이다.

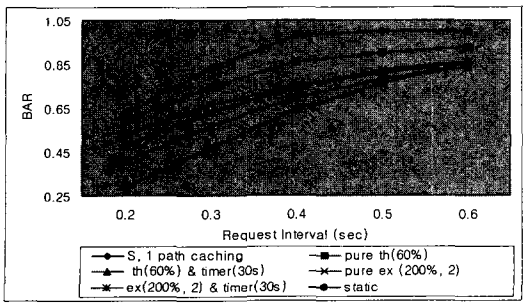
그림 4, 5, 6에서의 라인들은 다음과 같이 각각의 스킴에 해당하는 결과를 보여주고 있다.

- S, 1 path caching: 제안하는 서버기반 1-path caching 스킴
- pure th(60%): 60%를 임계치로 하는 순수 임계치 기반 갱신 정책
- th(60%)&timer(30s): 30초 크기의 클램프 다운 타이머를 사용하고 60%를 임계치로 하는 임계치 기반 갱신 정책
- pure ex(200%, 2): '기본 클래스 크기'가 200%이고 '클래스 성장 파라미터'가 2인 순수 지수 클래스 기반 갱신 정책
- ex(200%, 2)&timer(30s): 30초 크기의 클램프 다운 타이머를 사용하고 '기본 클래스 크기'가 200%이고 '클래스 성장 파라미터'가 2인 지수 클래스 기반 갱신 정책
- static: 정적 라우팅

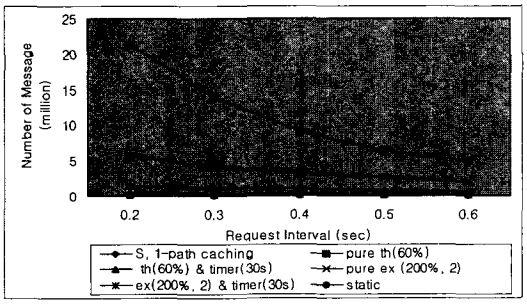
본 논문에서는 QoS 라우팅의 오버헤드로 크게 다음과 같이 두 가지를 분석하였다: (a) 네트워크 대역폭 소모, (b) 라우트 서버 혹은 하나의 라우터에서의 메시지 프로세싱 오버헤드. (a)의 경우, 총 시뮬레이션 시간 동안 각 링크에서 전송된 모든 프로토콜 제어 메시지, 즉 네트워크 상태 정보 갱신 메시지 혹은 경로 질의 및 응답 메시지의 양을 측정하여, 전체 네트워크 상의 모든 링크들에서의 이 값을 모두 합산함으로써 구하였다. (b)의 경우, 제안하는 스킴에서의 라우트 서버 혹은 분산 QoS 라우팅에서의 임의의 라우터 하나에서 전송하거나 전송 받은 평균 메시지 수로서 측정하였다.

그림 4, 5는 각각 메쉬와 ISP 구조에서 네트워크에 부가되는 트래픽 로드 변화에 따른 성능 변화를 살펴본 것이다. 이 실험에서는 평균 세션 요청 발생 간격을 0.3에서 1.4의 범위에서 변화시켰는데, 세션 요청 발생 간격이 짧아짐에 따라 트래픽 로드가 커지게 된다. 0.2~0.6의 평균 세션 발생 간격 하에 정적 라우팅 스킴은 28%~85%의 라우팅 성능을 보인다. 그림 6에서는 네트워크 크기 변화에 따른 각 스킴들의 라우팅 성능 변화를 비교하였다. 이 실험에서는 각 노드에서의 세션 발생 간격을 동일하게 유지하면서 전체 클라이언트 노드 수를 9개에서 121개까지 변화시켰다.

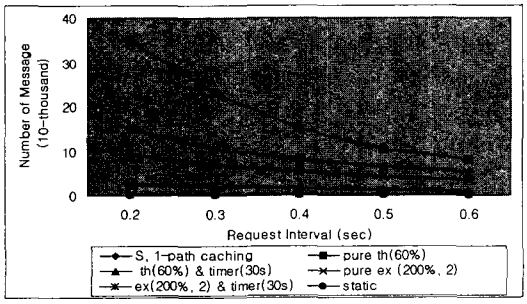
라우팅 성능을 표시하는 *BAR*을 비교해 보면 모든 경우에 대하여 제안하는 스킴이 기존의 분산 QoS 라우팅 스킴들에 비해 우수하다(그림 4(a), 5(a), 6(a) 참조). 네트워크의 대역폭 소모량 면에서 보면 제안하는 스킴은 기존 스킴들 중 클램프 다운 타이머 정책과 조합한 경우의 네트워크 대역폭 소모량과 유사한 정도의 오버헤드를 발생시킨다(그림 4(b), 5(b), 6(b) 참조). 기존 분산 QoS 라우팅 스킴들 중 가장 라우팅 성능이 높은 순수 임계치 기반 갱신 정책과 비교해보면 제안하는 스킴



(a) 대역폭 수용률

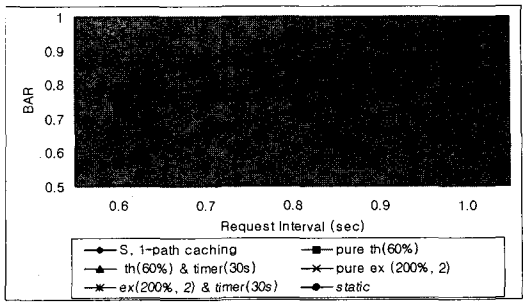


(b) 네트워크 대역폭 소모율

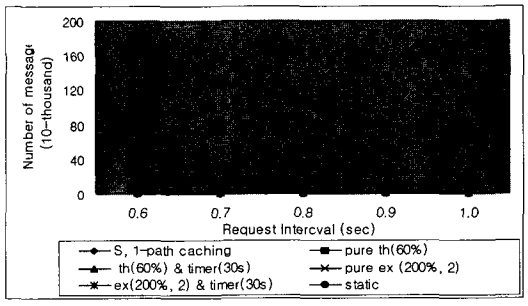


(c) 라우트 서버/라우터에서의 메시지 프로세싱 오버헤드

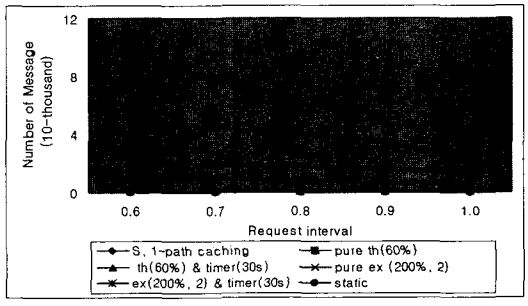
그림 4 트래픽 로드 변화에 따른 제안하는 서버 기반 QoS 라우팅과 기존의 분산 QoS 라우팅과의 성능 비교(메쉬)



(a) 대역폭 수용률



(b) 네트워크 대역폭 소모율



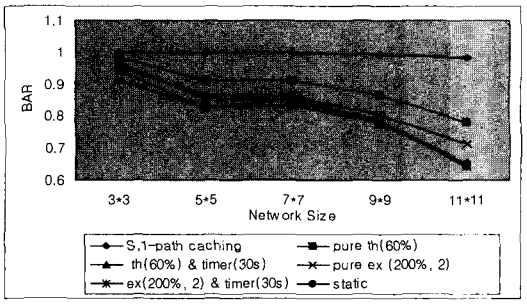
(c) 라우트 서버/라우터에서의 메시지 프로세싱 오버헤드

그림 5 트래픽 로드 변화에 따른 제안하는 서버 기반 QoS 라우팅과 기존의 분산 QoS 라우팅과의 성능 비교(ISP)

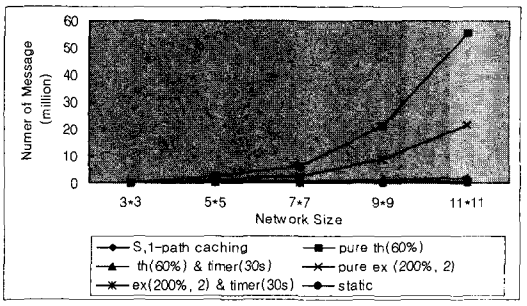
은 상당히 낮은 수준의 대역폭 소모 오버헤드를 가짐을 알 수 있다. 메시지 프로세싱 오버헤드에 있어서는(그림 4(c), 5(c), 6(c) 참조), 제안하는 스킴의 라우트 서버가 분산 QoS 라우팅 스킴에서 지수 클래스 기반 갱신 정책을 사용하는 경우의 임의의 한 라우터보다 약간 높은 정도의 오버헤드를 부담한다. 그러나 임계치 기반 갱신 정책을 사용하는 경우의 임의의 한 라우터보다는 훨씬 낮은 메시지 프로세싱 오버헤드를 부담함을 볼 수 있다. 그림 4의 메쉬에서의 결과와 그림 5의 ISP에서의 결과를 살펴보면, ISP의 경우 네트워크 크기가 줄어듦에 따라 수치적으로 결과 값이 낮아졌으나 전체적인 경향은 유사함을 볼 수 있다.

4. 결 론

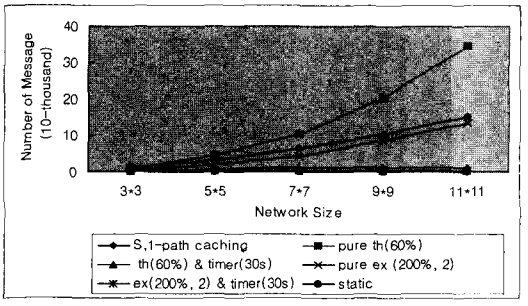
제안하는 서버 기반 QoS 라우팅 스킴에서는 라우트 서버가 QoS 경로를 할당하고 반환 받는 과정을 통해 네트워크 상태 정보 데이타베이스를 독자적으로 유지하도록 하기 때문에 네트워크 상태 정보 교환으로 인한



(a) 대역폭 수용률



(b) 네트워크 대역폭 소모율



(c) 라우트 서버/라우터에서의 메시지 프로세싱 오버헤드

그림 6 네트워크 크기 변화에 따른 제안하는 서버 기반 QoS 라우팅과 기존의 분산 QoS 라우팅과의 성능 비교

프로토콜 오버헤드가 제거되었다. 뿐만 아니라 정확한 링크 QoS 상태 정보를 이용하여 경로 계산을 수행할 수 있기 때문에 라우팅 성능 또한 향상되었다. 또한, 본 논문에서는 라우트 서버에서의 경로 계산 오버헤드를 감소시키기 위한 경로 캐싱 스킴들을 제안하고 그 성능을 평가하였는데, 통신 패턴에 지역성이 존재할 때 단순한 캐싱 스킴을 적용하는 것만으로도 계산 오버헤드가 크게 줄어듦을 알 수 있었다. 또 트래픽 로드 및 네트워크 크기를 변화시키면서 제안하는 스킴과 기존의 분산 QoS 라우팅 스킴들과의 성능을 비교하였는데, 제안하는 서버 기반 QoS 라우팅 스킴은 기존의 분산 QoS 라우팅 방식에 비해 라우팅 성능을 향상시킬 뿐만 아니라 프로토콜 오버헤드 측면에서도 우수함을 볼 수 있었다.

참 고 문 헌

[1] Q. Ma, P. Steenkiste, "On Path Selection for Traffic with Bandwidth Guarantees," IEEE ICNP 1997.

[2] R. Guerin, A. Orda, D. Williams, "QoS Routing Mechanisms and OSPF Extensions," IEEE GLOBECOM 1997.

[3] J.-Y. Le Boudec, T. Przygienda, "A Route Precomputation Algorithm for Integrated Services Networks," Journal of Networks and Systems Management, vol. 3, no. 4, pages 427-449, 1995.

[4] A. Shaikh, J. Rexford, K. Shin, "Efficient Precomputation of Quality-of-Service Routes," Proceedings Workshop on Network and Operating Systems Support for Digital Audio and Video 1998.

[5] G. Apostolopoulos, S. K. Tripathi, "On the Effectiveness of Path Pre-Computation in Reducing the Processing Cost of On-Demand QoS Path Computation," IEEE INFOCOM 1998.

[6] G. Apostolopoulos, R. Guerin, S. Kamat, S. K. Tripathi, "On Reducing the Processing Cost of On-Demand QoS Path Computation," Journal of High Speed Networking.

[7] M. Peyravian, A. D. Kshemkalyani, "Network Path Caching: Issues, Algorithms, and a Simulation Study," Computer Communications, vol. 20, pages 605-614, 1997

[8] A. Shaikh, J. Rexford, K. Shin, "Dynamics of qulity-of-service routing with inaccurate link-state information," University of Michigan Technical Report CSE-TR-350-97, 1998.

[9] G. Apostolopoulos, R. Guerin, S. Kamat, S. K. Tripathi, "QoS Routing : A Performance Perspective," ACM SIGCOMM 1998.

[10] G. Apostolopoulos, R. Guerin, S. Kamat, S. K. Tripathi, "Server based QoS Routing," IEEE GLOBECOM 1999.

[11] S. Herzog, "RSVP Extensions for Policy Control," IEEE RFC 2750, 2000

[12] R. Yavatkar, D. Pendarakis, R.Guerin, "A Framework for Policy-based Admission Control," IEEE RFC 2753, 2000.

[13] J. Boyle, R. Cohen, D. Durham, S. Herzog, R. Rajan, A. Sastry, "The COPS(Common Open Policy Service) Protocol," IEEE RFC 2748, 2000.

김 성 하
2000년 이화여자대학교 컴퓨터학과 졸업(학사). 2002년 이화여자대학교 과학기술대학원 컴퓨터학과 졸업(공학석사). 2002년 ~ 현재 삼성전자 정보통신총괄 연구원. 관심분야는 IP QoS, 라우팅 프로토콜 설계 및 성능 분석

이 미 정
정보과학회논문지 : 정보통신
제 29 권 제 1 호 참조