웹 상에서의 차별화 된 서비스 제공을 위한 Diff-HTTP

(Differentiated-HTTP for Differentiated Web Service)

현 은 실 † 이 윤 정 †† 김 태 윤 †††

(Eun Sil Hyun)(Yoon Jung Rhee)(Tai-Yun Kim)

요 약 HTTP 프로토콜은 WWW에서 HTML(HyperText Markup Language)문서를 송수신하기 위해 사용하고 있는 애플리케이션 프로토콜로서 TCP를 수송 계층 프로토콜로 이용하여 이루어지는 애플리케이션 계층 프로토콜 가운데 하나이다[12]. HTTP/1.0은 동일한 서버로부터 각각의 개체에 대하여 개별적인 TCP 연결을 생성하기 때문에 다중의 요구를 비효율적으로 처리한다. 이러한 문제를 해결하기 위한 방안으로 제안된 HTTP/1.1은 TCP 연결을 지속적인 연결(Persistent connection)이라는 개념을 도입하여 하나의 TCP 연결 상에서 다중의 요구(Request)를 처리하도록 하고 있다[9]. 네트워크가 발전됨에 따라 사용자가 늘어나고 다양해지면서 서비스의 차별화 문제가 중요한 문제로 대두되었다[3,5]. 본 논문에서 제시하는 Diff(Differentiated)-HTTP은 웹 서버에 서비스를 요청한 클라이언트들에게 차별화 된 서비스를 제공하기 위해서 사용자를 두 등급, 기본 등급과 우선 순위를 고려한 상위 등급으로 구분한다. 각 등급은 제한 시간(Holding Time)으로 차별화 되고 상위 등급에 속한 클라이언트에게 제한된 시간을 증가 시켜 지연을 최소로 함으로써 고품질의 서비스를 제공하는 방안을 제시한다.

Abstract The HyperText Transfer Protocol(HTTP), the Web's application-layer protocol, is at the heart of the Web. HTTP/1.0 establishes a new TCP connection for each HTTP request, resulting in many consecutive short-lived TCP connection[12]. HTTP/1.1 standard reduces latencies and overhead from closing and re-establishing connections by supporting persistent connections as a default, which encourage multiple transfers of objects over one connection[9]. HTTP/1.1, however, does not define explicitly connection-closing time but specifies a certain fixed holding time model. This paper proposes the mechanism of a Diff-HTTP((Differentiated-HTTP) supported by the server-side under HTTP/1.1. The current World-Wide Web services model treats all requests equivalently, while being processed by servers. Based on the policy, different levels of service are desirable[3,5]. This paper presents , service-side, application-level mechanisms to provide each different level of web service with upper class and default class. Our experiments show that upper class latencies are reduced than default one.

1. 서 론

HTTP 프로토콜은 WWW에서 HTML(HyperText Markup Language)문서를 송수신하기 위해 사용하고 있는 애플리케이션 프로토콜로서 TCP를 수송 계층 프로토콜로 이용해서 이루어지는 애플리케이션 계층 프로토콜 가운데 하나이며 클라이언트의 요청에 서버가 응답하는 요청/응답 (Request/Response) 방식으로 동작하게 된다[12].

네트워크가 발전됨에 따라 사용자가 늘어나고 다양해지면서 차별화 된 서비스의 제공이 중요한 문제로 대두되었다. 그러나 현재의 WWW 서비스는 인터넷의 단일 서비스에 의해 동일한 서비스의 품질을 제공한다. 모든 사용자들에게 동등한 서비스 제공하는 것도 중요하지만 서비스의 품질을 보장하는 방법으로 고급 사용자들에게 고품질의 서비스 제공이 이슈로 대두되고 있다. 이러한 문제점을 고려할 때 Web 서버에서의 차별화 된 서비스 제공 메커니즘이 요구된다[3,5].

웹 상에서 이미지들 각각은 독립적이며 개별적으로

† 비 회 원 : 고려대학교 컴퓨터학과
cunsil@netlab.korea.ac.kr
†† 종신회원 : 고려대학교 컴퓨터학과 교수
tykim@netlab.korea.ac.kr
논문접수 : 2000년 7월 10일
심사완료 : 2000년 11월 3일

검색되고 바뀐다. 따라서 웹 클라이언트는 보통 기본 HTML문서를 전송한 후 그 안에 삽입되어 있는 이미지들을 불러온다. 그 결과 HTTP/1.0은 동일한 서버로부터 각각의 개체에 대하여 개별적인 TCP 연결을 생성하기 때문에 다중의 요구를 비효율적으로 처리한다. HTTP/1.0에서는 각각의 문서마다 별도의 연결을 만들어야 하는 추가적인 부담이 발생한다. 따라서 많은 문서를 갖고 오려고 할 때 HTTP 프로토콜은 성능상의 저하를 발생시키게 된다.

IETF의 HTTP-NG은 이런 문제점을 포함하여 다각적인 요구를 수용할 수 있도록 HTTP/1.1(RFC 2616)을 발표하였다. HTTP/1.1은 TCP 연결을 지속적인 상태로 유지하고 연속적으로 요청과 응답을 처리하기 위하여 지속적인 연결(Persistent Connection) 개념을 도입하고 있다[12]. 그러나 연결 해제 시점을 명확하게 정의하고 있지 않고 있기 때문에 서버의 자원 낭비를 줄이는 효율적인 연결 관리 방법의 제시를 구현시의 문제로 남겨두고 있으며 최근에 구현된 HTTP 서버의 연결 해제 정책은 제한 시간(Holding Time 혹은 Time Out) 방식으로 일정한 시간을 주어 그 시간 안에 다른 요청이 없을 시 연결을 해제시키고 시간이 지나기 전에 또 다른 요청이 있을 경우 그 시점에서 새로운 시간을 할당해주는 방식이다[1].

본 논문에서는 Web 서버의 고품질의 차별화 된 서비스 제공을 위하여 효율적인 연결 관리 기법을 제시한다. 제안된 메커니즘은 제한 시간 정책을 이용하며 서버가 각기 다른 사용자에게 차별화 된 서비스를 제공하기 위해 사용자를 두 등급 즉, 우선 순위를 고려한 상위 등급과 기본 등급으로 나눈 후 제한 시간을 차별화 하여 상위 등급에 포함된 사용자에게 제한 시간을 증가 시켜 지연을 최소로 함으로써 고품질의 서비스를 제공한다.

2장에서는 기존에 연구되어 왔던 HTTP 프로토콜에 관한 관련 연구 제시하며 이들의 문제점에 대하여 살펴보고 3장에서는 연결관리를 위해 본 논문에서 제시한 알고리즘의 설계 원칙과 제안된 차별화 연결 관리에 대한 Diff-HTTP 모델 및 시나리오를 제시한다. 4장에서는 프로토타입 구현 및 실험 결과의 분석을 통해 성능을 평가한다. 마지막으로 5장에서는 결론과 향후 연구 방향을 제시한다.

2. HTTP 관련연구

본 장에서는 현재까지 진행되어 온 HTTP 프로토콜의 문제점을 분석하고 관련 연구를 살펴본다.

2.1 HTTP/1.0 연결관리

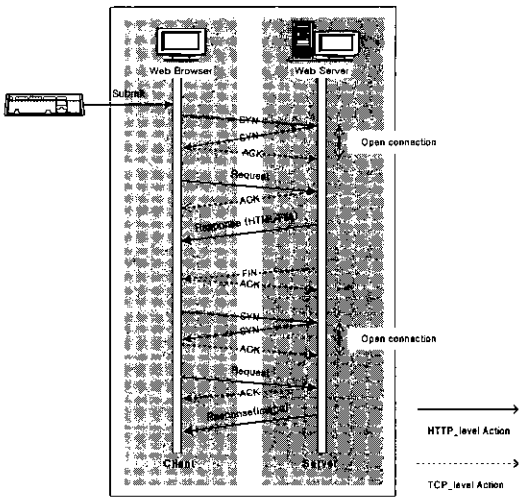


그림 1 HTTP/1.0의 비지속적인 연결

HTTP/1.0상에서 클라이언트는 같은 서버에게 연속적인 문서 요청을 함에도 불구하고 서버는 클라이언트가 요청하는 문서 파일을 전송하고 나면 자동적으로 연결을 해제하는 구조로 되어있다. 각 요청마다 새로운 TCP 연결이 설정/해제되기 때문에 다음과 같은 많은 문제점이 야기된다[9,14].

- TCP 연결 설정 시간 지연

HTTP 프로토콜은 요청 메시지를 전달하고 응답을 수신하기 위해서 클라이언트와 서버 사이에 TCP 연결을 먼저 만들어야 한다. TCP 계층의 연결은 삼단계 핸드쉐이크 방식으로 이루어진다. 따라서 같은 서버에 있는 문서를 클라이언트가 연속적으로 요청한다 하더라도 매번 요청마다 TCP 연결을 맺어야 하고 연결 설정을 위한 시간 지연이 생기게 된다.

- 슬로우 스타트(Slow Start)로 인한 지연

TCP 계층에서는 연결 네트워크에 혼잡이 탐지되면 윈도우 크기를 다시 산출하기 위해 슬로우 스타트 알고리즘이 실행된다. 이 방식의 문제점은 현재의 통신망 상황이 매우 좋음에도 불구하고 TCP 연결이 시작된 시점에서 조그만 윈도우 크기로부터 시작된다는 점이다. 만약 송수신할 데이타의 양이 많아서 오랜 시간동안 TCP 연결을 맺어 두고 있다면 이와 같은 문제는 무시할 만하다. 그러나 HTTP/1.0 응용 프로토콜은 송수신할 데이타의 양이 적기 때문에 짧은 시간 동안 존재하는 TCP 연결에서는 슬로우 스타트 알고리즘은 오히려 심각한 시간 지연을 야기 시킨다.

- TIME_WAIT 상태 문제

서버가 데이타를 전송한 후에 TCP 연결을 해제하게 되는데 예기치 않은 지연된 패킷이 수신 측에 도착한다면 이 때 발생되는 상황에 대한 추가적인 오류 제어가 필요하다. 따라서 해제된 TCP 연결에 대한 상태정보를 일정 시간동안 저장하고 있어야 하며 이 시간 동안을 TIME_WAIT 상태에 있다고 한다. 검색 엔진과 같은 활용 빈도가 높은 서버에서는 수많은 TCP 연결의 설정/해제 동작이 반복되며 이와 같은 상황은 서버에게 상당한 영향을 미치게 된다. 이로 인하여 새로운 클라이언트의 TCP 연결 요구에 대하여 적절한 응답을 해줄 수 없는 상황이 생길 수 있다.

2.2 HTTP/ 1.1 지속적인 연결

HTTP/1.1은 클라이언트의 요청에 대한 응답을 보낸 후 서버는 즉시 연결을 해제하지 않고 TCP 연결을 계속 유지한다. 동일한 클라이언트와 서버 사이에서 연속적인 요청/응답을 단일 연결을 통해서 보낼 수 있다[7,9].

그림 2에서는 HTTP/1.1의 요청/응답 과정이 기존에 설정되어 있는 TCP 연결을 계속해서 이용하여 이루어짐을 보여준다. 클라이언트의 요청이 남아 있을 경우 클라이언트가 서버에게 문서를 처음 요구를 할 때 사용한 TCP 연결을 다음 요구 때에도 계속 사용한다.

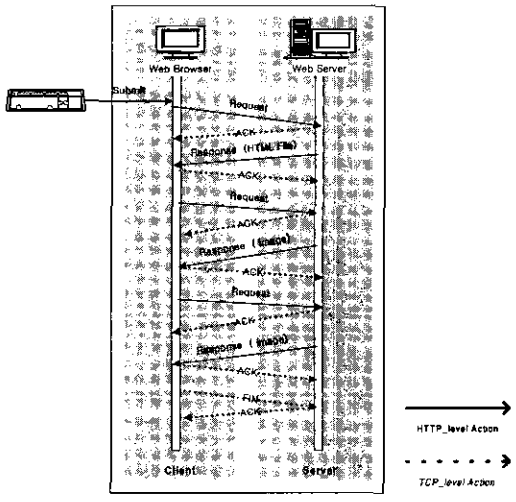


그림 2 HTTP/1.1의 지속적인 연결

2.2.1 지속적인 연결에서의 연구와 문제점

HTTP/1.1은 TCP 연결을 명시적인 'CLOSE' 연산을 실행하기 전까지 계속 연결 상태를 유지해야 하기 때문에 휴지 상태의 연결에 대한 해제 시점을 정하는 원칙이 매우 중요하다. 소켓 버퍼의 최소 크기는 가장 큰 TCP

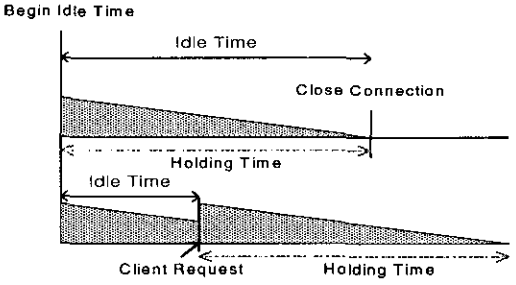


그림 3 제한 시간 모델 - Time out

패킷보다 커야 하며 또한 연결이 새로이 설정될 것을 대비하여 버퍼를 미리 할당해 두어야 한다. 그리고 가용한 소켓의 수는 한정되어 있기 때문에 TCP 연결 해제 시점이 적절하지 않을 경우 휴지 상태 연결이 서버의 자원인 CPU와 소켓과 버퍼의 메모리 공간을 낭비하게 된다. 따라서 무조건 연결을 계속 유지시키는 것은 서버 자원의 낭비뿐만 아니라 서버의 오버헤드와 자원 부족으로 인하여 새로운 클라이언트와의 연결을 지연시키는 결과를 야기한다. 결국 휴지상태의 연결이 많을 시에는 서버의 처리율을 저하시키는 요인으로 작용한다[2,15].

연결 해제 시점에 관한 연구 방향은 크게 제한 시간 방식과 캐시 모델 방식으로 나뉜다. 최근의 아파치 HTTP 서버 1.3은 그림 3과 같은 제한 시간 방식을 사용하여 구현하고 있다. 제한 시간 방식은 일정한 제한 시간을 설정하여 연결을 그 시간동안만 활성화시키는 방법이다[1]. 서버는 새로운 클라이언트와 TCP 연결을 처음 설정하거나 동일한 클라이언트로부터 새로운 요청이 도착했을 때 제한 시간을 시작한다. 제한 시간이 지속되고 있는 동안에는 서비스를 위한 연결이 활성화되고 제한 시간 안에 요청이 다시 들어오면 새로운 제한 시간이 시작되고 그렇지 않으면 연결이 해제된다.

적절한 제한 시간을 선택하는 문제는 고정 제한 시간 모델과 동적 제한 시간 모델이 있다. 고정된 제한 시간 모델은 모든 클라이언트의 요청에 대하여 같은 제한 시간을 할당하는 방식이며 동적 제한 시간 모델은 제한 시간의 값을 고정된 값으로 사용하지 않고 테이블 기반의 다수의 시간 값을 가지고 있어서 현 서버 자원의 상태를 측정하여 적절한 제한 시간 값을 선택하는 방식이다. 따라서 서버 자원의 상태가 여유로운 경우 제한 시간 테이블 중 큰 값을 선택하여 제한 시간을 길게 설정하여 연결을 오랫동안 유지하고 만약 서버 자원의 로드가 심한 경우에는 제한 시간 값 중 작은 값을 선택한다[2].

캐시 모델은 연결이 활성화된 상태를 캐시 상태라고

하고 최대 허용할 수 있는 연결 최대값을 정하여 캐시된 연결의 수를 제한하는 알고리즘이다. 연결은 클라이언트에 의해 'CLOSE' 되기 전까지 혹은 새로운 클라이언트와의 연결로 대체될 때까지 연결 상태로 남아 있다. 다른 연결을 설정하는 과정에서 캐시가 포화상태가 되어 더 이상의 연결을 설정할 수 없을 경우 대체 알고리즘에 따라 데이타의 이동이 없는 휴지 상태의 클라이언트들 사이에서 해제할 연결을 선택한 후 연결 해제 작업이 진행된다. 이 때 적용되는 대체 알고리즘은 여러 가지가 있지만 그 중에 가장 많이 사용하는 것은 LRU(Least Recently Used) 방식이다. LRU는 가장 오랜 시간 동안 휴지 상태로 존재하는 연결을 선택하여 휴지상태의 연결 중에서 해제가 필요한 시점이 왔을 때 그 연결을 해제하는 것이다. 이것의 기본 원칙은 지역성에 따라 해제될 연결을 선택하는 데 있다[1,10].

2.2.2 차별적인 서비스 제공을 위한 연구

최근의 연구 동향에서 보여지는 것은 클라이언트가 요청한 서비스에 대하여 서버가 클라이언트를 차별화하여 서비스를 제공하기 위한 여러 방안이다.

동시에 많은 클라이언트가 서버에 작업 요청을 했을 경우 우선 순위 기반의 스케줄링 기법을 제공한다. 서버의 운영체제에서 FCFS 알고리즘이 아닌 우선 순위가 높은 요청들을 낮은 요청들에 비해 먼저 처리해 주는 알고리즘이다. 현재 웹 서버는 요청들 사이에서 우선 순위를 고려 해주는 메커니즘이 제공되고 있지 않다. 따라서 이것을 구현하기 위하여 커널 상에서 우선 순위를 고려 해 주어야 한다[3].

또는 각 서비스를 상위 그룹(foreground)와 하위 그룹(background)으로 나누어 두 등급의 서비스를 실시한다. 상위 그룹 요청에 대한 응답을 위해서는 하위 그룹 요청에 비해 서버의 자원을 더 할당한다. 우선 서버는 상위 그룹과 하위 그룹의 두 개의 큐를 가지고 하위 그룹의 큐가 사용하는 서버의 자원을 제한한다. 따라서 이것은 상위 그룹의 요청은 하위 그룹의 요청들의 로드가 증가해도 크게 영향을 받지 않는다[5].

3. 차별화된 서비스 제공을 위한 Diff-HTTP

본 장에서는 본 논문에서 제공하고 있는 Diff-HTTP에 대한 설계 원칙을 제시하고 이를 기반으로 한 Diff-HTTP 모델과 시나리오를 제안하고자 한다.

3.1 설계 원칙

(1) 차별화 된 서비스 제공

네트워크가 발전됨에 따라 사용자가 증가하고 다양해지면서 서비스의 차별화 문제가 중요한 문제로 대두되었다. 그러나 현재의 웹서비스는 인터넷의 단일 서비스에 의해 동일한 서비스를 제공한다. 모든 사용자들에게 동일한 서비스를 제공하는 것도 중요하지만 서비스의 품질을 보장하기 위하여 서버 측에서의 차별화 된 정책을 통해 서버에게 서비스를 요청한 클라이언트 중 우선 순위를 고려하여 고품질의 서비스를 제공하는 것도 또한 중요한 문제이다[3,5].

차별화 된 서비스를 제공하기 위한 차별화 정책은 다음과 같이 정의 할 수 있다.

- History 기반의 정책

서버의 히스토리 정보를 기반으로 하여 그 동안 서버에 많이 접속한 사용자에게 높은 우선 순위를 부여한다.

- Membership 기반의 정책

사용자를 회원과 비회원으로 분류한 후 회원에게 좀 더 고품질의 서비스를 제공하기 위해 높은 우선 순위의 서비스를 부여한다.

- Paying 기반의 정책

서비스에 대한 요금이 부여 될 경우 요금을 지불한 사용자와 그렇지 않는 사용자에 서비스를 차별화하고 요금을 부여한 사용자에게 높은 우선 순위를 부여한다.

(2) 지연의 최소화

네트워크 지연을 줄이기 위한 수단은 네트워크로 연결하는데 있어서 RTT를 줄이는 데 있다. 그러나 웹에서 사용하는 HTTP 프로토콜은 TCP를 기반으로 하기 때문에 많은 수의 RTT를 발생시킨다. 따라서 최소한의 RTT는 지연을 최소로 하고 그로 인해 사용자가 느끼는 지연이 최소화된다[6,8].

(3) 신뢰성 보장

서버는 TCP 연결을 임의대로 해제 할 수 없다. 연결은 서버가 하나의 요청을 다 처리한 후 그것이 계속된 요청이 없을 거라는 것이 확인된 후에 해제가 가능하다. 클라이언트는 서버가 연결을 해제하기 전에 요청에 대한 응답을 받아야 하는 것을 보장받아야 한다. 어떤 상황이 새로운 요청 클라이언트의 전달과 서버의 TCP 연결에 대한 해제 사이에서 일어난다면 이 경우 클라이언트는 응답을 받기 전에 연결이 해제될 것이다. 그러므로 클라이언트는 응답을 받기 위해 다시 연결 성립을 하고 그 응답에 대한 요청을 다시 보낼 것이다. 아파치 서버에서 제공하는 제한 시간방식은 클라이언트가 더 이상 요청을 하지 않고 휴지 상태에 있다면 제한 시간의 카운트가 시작되고 제안 시간이 감소하여 0으로 만기되면 연결은 자동적으로 해제되는 것이다. 제한 시간이 끝나기 전에 클라이언트 요청이 오면 새로운 제한 시간이 시작된다. 그러나 이 방식은 네트워크의 지연으로 클라

이언트의 요청이 늦게 서버에 도착되었거나 요청 자체가 소실되었을 경우 서버의 입장에서는 더 이상 클라이언트의 요청이 없는 것으로 간주하여 연결을 해제한다. 따라서 나중에 요청이 도착했을 경우 이미 연결이 존재하지 않는 상태이기 때문에 다시 연결을 위한 TCP 연결 설정과정이 필요하게 된다[11].

(4) 효율성

클라이언트 입장에서는 연결이 요청/응답 후 해제되는 것보다는 연속적인 연결을 가지고 서비스를 받는 것이 유리하다. 클라이언트의 요청 메시지가 지속적인 연결을 통해 서비스가 되어지기 때문에 지연을 감소시킬 수 있기 때문이다. 그러나 서버의 입장에서는 다수의 클라이언트가 오랜 휴지 시간 동안 다음 사용을 목적으로 연결을 유지한다면 서버의 자원이 낭비된다. 서버는 일정한 휴지 시간 후에는 클라이언트의 연결을 해제해야 한다. 그래서 다른 필요한 연결이 요구되었을 때 새로운 연결을 다시 설정할 수 있어야 한다. 따라서 전체적인 서버의 자원을 효율적으로 사용하기 위한 적절한 제한 시간의 선택이 중요하다[4].

(5) 서버자원의 처리율

서버는 몇 가지의 자원을 사용한다. 예를 들어 CPU 시간이나 버퍼와 소켓의 메모리 그리고 PCB 테이블 공간 등이다. 만약 평균 TCP 연결 보다 다수가 요청되어 한계 치 이상이 된 다면 서버는 모든 클라이언트의 연결에 대한 서비스를 제공하기 위해 각 클라이언트에게 제공되는 서버의 CPU 시간을 줄이려고 시도할 것이다. 이 결과 각 클라이언트가 받아야 할 서비스는 지연이 발생된다[2].

3.2 Diff-HTTP 모델

네트워크가 발전하면 발전할수록 서버의 부하가 증가되기 마련이다. 서버는 모든 클라이언트가 요청하는 서비스를 모두 처리할 수 있다면 별 문제가 되지 않지만 클라이언트의 서비스에 대한 요청이 많을 경우 그 요청을 모두 고품질의 환경으로 처리해 주는 데는 한계가 있다. 따라서 본 논문에서 제안한 방법은 이전의 HTTP/1.1의 제한 시간 방식을 기본으로 하되, 우선 순위를 부여한 상위 등급을 만든 후 제한 시간의 조절을 통하여 상위 등급 사용자가 서비스를 받을 경우 연결의 설정/해제 과정의 비율을 상대적으로 낮추어 상위 등급 사용자로 하여금 지연을 덜 느끼도록 하는 것이다.

작동원리는 다음과 같다.

(1) 클라이언트는 TCP 연결을 설정한다.

(2) 클라이언트는 HTTP 요청을 서버에게 보낸다. 서버는 먼저 이 사용자가 등급 중 어느 등급에 속하는 지

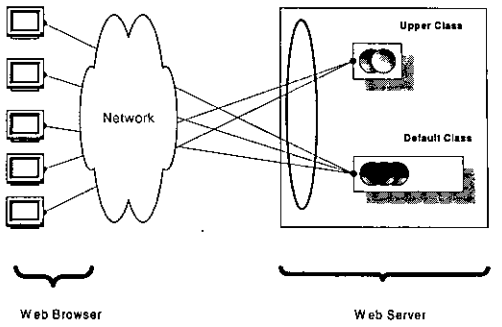


그림 4 서버의 차별화된 서비스

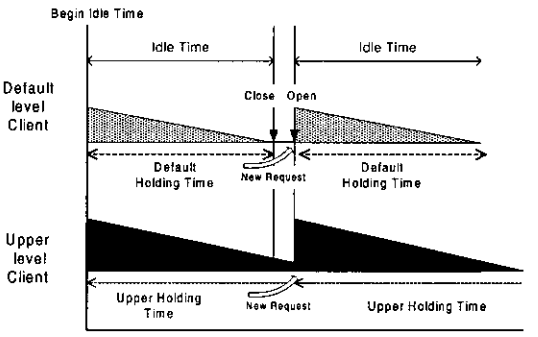


그림 5 차별화된 제한 시간

분석 한 후 그 등급에 부합된 제한 시간을 부여한다. 디스크로부터 클라이언트가 요구한 것을 조회해서 서비스를 완수한다. 그리고 클라이언트에게 응답을 보낸다.

(3) 서버는 클라이언트가 연결을 해제하지 않고 다른 요청이 들어올 때까지 TCP 연결을 유지한다. 각 등급에 따라 클라이언트에 대한 제한 시간이 지나기 전에 클라이언트의 요청이 있다면 다시 해당 클라이언트에 부합된 제한 시간이 설정되고 해당 클라이언트의 요청이 더 이상 없을 경우 연결을 해제한다.

상위 등급과 기본 등급에 따른 차별화 된 서비스는 제한 시간을 등급에 따라 부여함으로써 정의 할 수 있다. 그림 5에서 볼 수 있듯이 우선 순위를 부여받은 상위 등급의 클라이언트는 기본 등급의 클라이언트에 비해 제한 시간에 추가 시간 'a' 가 부여된다.

만약 네트워크의 혼잡으로 인해 패킷이 지연되거나 또는 폐기되는 경우 클라이언트의 요청이 있었음에도 불구하고 서버에게 전달이 되지 않을 상황이 발생하게 된다. 또한 HTTP 프로토콜은 이전에는 텍스트 문서의 전송을 위해 쓰였지만 사용자의 서비스의 폭이 증가함

으로써 음성, 화상 등 여러 종류의 데이타 형식을 MIME(Multipurpose Internet Mail Extensions)로 정의하여 전송한다. 따라서 HTTP가 전송하는 파일의 용량이 커질수록 기존의 제한 시간 안에 모든 서비스를 다 처리하는 것이 어려워졌다. 만약 기본 등급에서 많은 용량의 파일을 전달하는 데 많은 시간을 소요하여 제한 시간이 만기되기 이전에 해당 클라이언트의 요청이 도착하지 못했다면 다시 연결의 해제/설정을 시작해야 할 것이다. 또한 기본제한시간 < 휴지시간 < 상위등급제한시간인 경우를 가정한다면, 상위 등급 클라이언트는 기본 등급의 제한 시간 보다 더 긴 제한 시간을 부여받음으로써 기본 등급의 경우에 이미 제한 시간이 만기되어 연결이 해제되어 다시 연결을 재 설정해야 하지만 상위 등급의 경우는 연결이 해제가 되지 않았기 때문에 연결을 재 설정할 필요가 없이 기존의 연결을 통해 서비스를 받을 수 있다. 따라서 상위 등급은 연결에 따르는 처리 오버헤드(Processing Overhead)와 연결을 새로이 설정함으로 야기되는 시간의 지연을 막을 수 있어 더 나은 고품질의 차별화 된 서비스를 받을 수 있다.

3.3 Diff-HTTP 시나리오

3.3.1 Diff-HTTP 환경 설정

서버와 클라이언트의 연결이 설정되는 과정에서 클라이언트들의 등급 결정 과정을 담당하는 것을 Broker라 정의한다. Broker를 통해 서비스를 요청한 클라이언트가 어느 등급에 포함되는 지의 여부를 결정한 후 서버는 해당 클라이언트의 등급에 합당한 제한 시간을 부여하여 클라이언트의 요청을 처리한다. 이 과정을 그림으로 나타내면 다음과 같다.

본 논문에서는 앞서 3.1에서 제시한 차별화 정책 중 히스토리 기반의 정책을 이용하여 구현하기로 한다.

(1) Broker

Broker의 역할은 크게 두 가지로 구분된다.

1) 등급테이블의 관리

• 서버에 접근하는 클라이언트의 접속 정보 히스토리를 기반으로 접속 횟수와 시간 정보에 따라 클라이언트들을 상위 등급과 기본 등급으로 구분한다.

• 전체의 클라이언트 중에 일정한 비율만큼을 상위 등급으로 규정한다.

• 최신의 접속 정보의 히스토리를 바탕으로 상위 등급을 관리한다.

차별화 된 서비스를 제공하기 위해서 가장 중요한 작업은 상위 등급의 사용자를 선별하는 것이다. 회원 기반 내지 요금 기반의 정책은 회원 가입 여부와 요금 지불 여부에 따라 선별이 가능하다. 그러나 히스토리 기반 정책의 사용자는 다음과 같은 알고리즘을 기반으로 상위 등급을 선별할 수 있다.

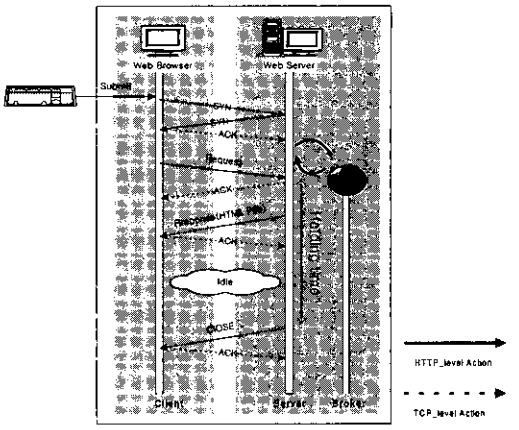


그림 6 서버의 작업 처리 과정

히스토리 기반의 최적의 알고리즘은 현 시점을 기준으로 앞으로도 많은 서비스를 서버에게 요청하는 사용자를 상위 등급 안에 포함시키는 것이다. 그러나 서버는 어떤 사용자가 서버의 서비스를 얼마나 많이 요청할 지를 예상하기 어렵기 때문에 최적 알고리즘의 근사치를 찾아야 할 것이다. 따라서 세 가지 알고리즘을 제한하고 그 성능을 비교하여 가장 효율적인 알고리즘인 LRFR 알고리즘을 본 논문에서는 사용하여 시뮬레이션을 하였다.

① LRR(Latest Recently Request) 알고리즘

가장 최근에 서비스를 요청한 사용자를 상위 등급 안에 포함시키는 방안으로 최적 알고리즘의 근사치이다. 즉 오랜 기간 동안 해당 서버에게 서비스를 요청하지 않은 사용자를 상위 등급에서 제외시키기 위한 것으로 상위 등급 안에 포함된 사용자는 가장 최근에 서버에게 서비스를 받은 사용자들이 될 것이다. 그러나 최근에 접속한 사용자들이 곧 앞으로도 많은 서비스를 해당 서버에게 요청한다는 보장이 없기 때문에 이 알고리즘에 의해 만들어진 상위 등급은 서버에 간헐적으로 접근하는 사용자를 포함한다.

② MFR(Most Frequently Request)알고리즘

해당 서버에게 서비스를 많이 요청한 사용자들을 요청 횟수 순으로 상위 등급 안에 포함시키는 방안이다. 이것은 활발하게 서버에게 서비스를 요청하는 사용자는 앞으로도 많은 서비스를 요청한다는 가정 하에서 만들어진 것이다. 그러나 이 알고리즘은 어떤 사용자가 전 단계에서 해당 서버에게 많은 서비스를 요청했지만 그 후로 다시 서비스를 요청하지 않을 경우에 문제점을 야

기한다. 접속 횟수로는 많은 횟수가 되어 상위 등급 안에 포함되지만 더 이상의 서비스를 요청하지 않는다면 상위 등급의 한정 수를 하나 차지하기 때문에 다른 사용자 즉 최근에 많은 서비스를 요청하는 사용자가 상위 등급 안에 포함되지 않을 수 있기 때문이다. 따라서 이 알고리즘에서 가장 중요한 것은 어떤 일정한 시간 간격으로 정보를 수정해야 한다는 것이다.

③ LRFR(Latest Recently/Frequently Request)알고리즘

앞에서 정의한 두 알고리즘의 문제점을 해결하기 위한 방안으로 만들어진 것이다. 이 알고리즘은 일정한 시간 간격 내에 가장 많은 서비스를 요청했던 사용자들을 상위 등급 안에 포함시키는 방안이다. 즉 사용자 접속 횟수와 최후 접속 시간 정보를 포함하고 있어 오랜 시간 동안 서버에게 요청을 하지 않는 사용자를 상위 등급 안에서 제외시키는 것이다. 따라서 시간 정보를 저장하고 있기 때문에 오랜 시간 동안 서버에게 요청을 하지 않는 간헐적인 사용자를 상위 등급에서 제외시킬 수 있다.

2) 클라이언트의 제한 시간 결정

클라이언트가 서버에게 HTML 문서를 받기 위한 연결을 요청하는 경우 그 요청을 처리 하기 앞서 해당 클라이언트가 어느 등급에 해당하는 지를 검사하여 서버로 하여금 그 클라이언트에게 해당하는 제한 시간을 부여하도록 한다.

상위 등급 안에 포함될 사용자의 정보를 저장하고 만약 서비스의 요청이 들어온다면 서비스를 요청하는 사용자들의 히스토리 정보를 비교하여 사용자가 어느 등급에 해당하는 지를 분석한다. 이 분석을 바탕으로 접근한 사용자가 일정한 상위 그룹 내에 포함되어 있다면 기본 제한 시간 + 'α' 를 부여하고, 그렇지 않은 경우는 일반 제한 시간을 부여하여 제한 시간을 할당한다.

LRFR을 기반으로 한 Diff-HTTP Broker의 알고리즘은 다음과 같다.

```
begin
  do
  if client is in Upperclass
    then assign DefaultTime + 'α'  to client
              accesscount is incremented by 1
  else
   assign DefaultTime to client
   accesscount is incremented by 1
  while (true)

  if (accesscount * time) is larger than averagevalue
   Upperclass include client
end
```

Diff-HTTP Broker 알고리즘

(2) 요청/응답 프로세싱

서버는 Broker에 의해 주어진 제한 시간을 기반으로 하여 해당 클라이언트와의 연결을 유지한다. 클라이언트가 요청한 서비스를 제공해 주고 만약 휴지 상태일 경우 해당 클라이언트의 제한 시간이 0으로 만기 될 때까지 더 이상의 요청이 없을 경우 연결을 해제한다. 제한 시간이 만기되기 전에 해당 클라이언트가 다른 서비스를 다시 요청한다면 그 요청을 처리해 주고 새로이 클라이언트에 해당하는 제한 시간을 적용한다.

3.3.2 서버와 클라이언트 알고리즘

차별화 된 서비스를 제공하기 위해서 서버 관점에서 알고리즘을 서술하면 다음과 같다. 구현을 위한 프로토타입에서는 요청은 'GET'과 'CLOSE'로 제한하였다.

(1) 서버

서버는 새로운 클라이언트의 요청이 들어오면 broker()함수를 호출한다. broker()를 사용하여 서버는 해당 클라이언트의 히스토리를 분석한 후 그 히스토리에 따라 등급을 결정한다. broker에 의해 결정되어진 등급에 따라 해당 클라이언트에게 제한 시간을 부여한다. 만약 클라이언트가 상위 등급에 속한 경우 그 클라이언트에게 기본적인 제한 시간보다 더 많은 제한 시간을 부여한다. 그렇지 않은 경우는 기본적인 제한 시간을 제공한다. 클라이언트가 휴지 상태에 있을 때 제한 시간이 감소하기 시작하여 시간이 만기되면 자동적으로 해제된다. 또는 클라이언트로부터 'CLOSE' 메시지를 받을 경우에도 연결을 해제할 수 있다. 다른 클라이언트로부터의 연결 요청이 일어날 경우 대비하여 비 활성상태에 있는 클라이언트와의 연결을 해제시킨 후 자원을 다른 사용자들의 사용을 위해 가용한 상태로 만든다.

(2) 클라이언트

클라이언트가 더 이상 서비스 요청이 없을 경우 서버에게 'CLOSE' 메시지를 보낼 수 있다. 또는 휴지 상태로 있다면 서버의 broker에 의해 설정된 제한 시간이 만기되어 서버에 의해 연결이 자동적으로 해제된다.

Diff-HTTP 서버의 알고리즘은 다음과 같다.

```
begin
  establish server.socket
  do
  if accept "SYN" from client
    then open client.socket
              time = call Broker()
  do
    read stream
    if method in stream is "GET"
     get filename from stream
```

```
    if file exist then response requested file
    else response "file not found"
  else if method in stream is "CLOSE"
    then close client socket and break
 while (time)
 while (true)
end
```

Diff-HTTP 서버 알고리즘

4. 실험결과 및 분석

본 장에서는 본 논문에서 제안한 Diff-HTTP와 HTTP/1.1의 Time out 방식의 연결관리 부분을 각각 프로토타입으로 구현하여 시뮬레이션한 결과를 비교·분석한다.

4.1 프로토타입 구현 환경 및 실험 환경

본 논문에서 제안하는 Diff-HTTP와 HTTP/1.1의 Time out방식의 서버와 클라이언트는 Window 2000 server Pentium II 400MHZ 환경에서 자바 JDK1.2버전을 사용하여 구현하였다.

통신 링크는 10 Mbps LAN을 이용하며 서버와 클라이언트의 실험 환경은 표 1과 같다.

표 1 Diff-HTTP 서버와 클라이언트의 실험 환경

시스템사양 / 통신주체	기종	운영체제	메모리
Server	Pentium III 450	NT Server	256M
Client	Pentium II 450	Win 98	128M

실험 및 성능 평가 방법은 다음과 같다.

- 30개의 클라이언트는 각각의 환경에서 작동한다.
- 서버가 동시에 클라이언트와 맺을 수 있는 지속적인 연결은 10개로 한정한다.
- 각각의 클라이언트들은 각각 상위 등급과 기본 등급으로 구분한다.
- 클라이언트 상에서 Diff-HTTP와 HTTP/1.1(제한시간 15sec)의 휴지시간을 증가시키면서 HTTP/1.1(15s)과 Diff-HTTP는 클라이언트에게 HTML 문서와 그 안에 포함되어 있는 이미지가 디스플레이 되는 시간을 제외한 네트워크 조회 시간을 측정함으로써 지연 시간을 측정하였다.
- Diff-HTTP는 기본 등급과 상위 등급을 고려하여 비교하였다.

4.2 성능평가

그림 7에서의 실험은 상위등급의 제한 시간을 30초로 제한하고 휴지 시간을 0초에서 5초씩 증가했을 때 HTTP/1.1(default = 15sec), Diff-HTTP의 평균치를 그래프로 나타낸 것이다. Diff-HTTP는 상위등급과 기본등급으로 나뉘기 때문에 이중화하여 표시하였다. HTTP/1.1(15sec), Diff-HTTP(기본등급), Diff-HTTP(상위등급)을 비교해 보면, 세 가지 경우 모두 처음에는 별다른 지연의 차이가 없지만 휴지 시간이 증가할수록 지연의 차이가 현저함을 알 수 있다. 휴지시간이 20초 전후에서 다시 HTTP/1.1과 Diff-HTTP 기본 등급은 이전의 연결을 해제하고 다시 재 설정하기 때문에 그만큼의 연결 설정 시간만큼의 시간이 지연됨을 보여준다.

이것은 Diff-HTTP(상위등급)이 기존의 HTTP/1.1과 Diff-HTTP(기본등급)에 비해 사용자가 느끼는 지연을 최소화시킴으로 차별화 된 서비스를 제공함을 보여주는 예이다. 따라서 Diff-HTTP(상위등급)은 네트워크의 혼잡도가 증가하는 경우나 HTTP를 통해 전달하고자 하는 용량이 클수록 신뢰성과 고품질의 서비스를 제공한다.

그림 8에서는 상위 등급의 제한 시간을 0, 20, 30, 40, 50 초로 설정하고 휴지 시간을 증가시키면서 지연을 측정한 것이다. 상위 등급의 제한 시간에 따라 지연의 차이를 알 수 있다. 제한 시간이 0에 해당하는 HTTP/1.0

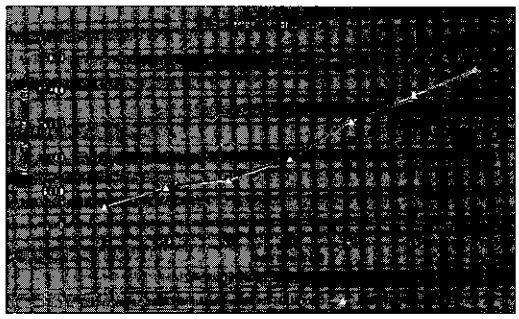


그림 7 휴지시간에 따른 네트워크 지연

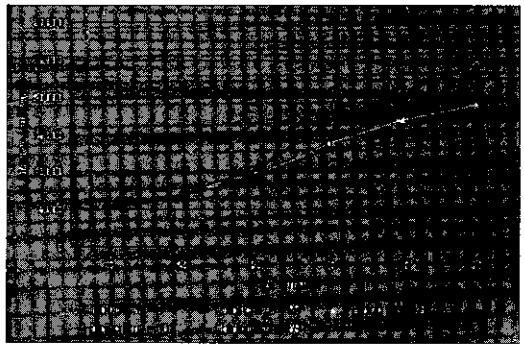


그림 8 제한 시간에 따른 지연

방식은 한번의 요청을 처리한 후 연결이 바로 끊어져서 재 연결을 요구하기 때문에 TCP 연결에 해당하는 시간적 지연이 발생한다. 그러나 제한 시간이 길어짐에 따라서 지속적인 연결을 유지하기 때문에 제한시간의 범위 내에서 신뢰성 있는 요청에 대한 응답을 받을 수 있다. 또한 네트워크의 혼잡으로 인해 패킷이 지연되거나 또는 폐기되는 경우 복잡한 네트워크 상에서 패킷이 지연이 있어도 동적으로 대처 할 수 있다. HTTP/1.1의 지속적인 연결을 계속 유지하는 경우 항상 서버의 연결을 유지해야 하기 때문에 서버의 자원에 대한 낭비가 이루어진다. 따라서 HTTP/1.1(time out)방식이나 Diff-HTTP 방식은 일정한 시간동안 휴지상태인 클라이언트의 연결을 해제함으로써 서버 자원의 낭비를 막을 수 있다.

표 2는 Diff-HTTP, HTTP/1.0 및 HTTP/1.1방식을 차별화 서비스, 지연, 신뢰성 등 여러 가지 비교항목을 통하여 비교한다.

표 2 Diff-HTTP, HTTP/1.0, HTTP/1.1의 비교
(○: High △: Low ×: None)

프로토콜 / 비교항목	HTTP/1.0	HTTP/1.1 (동등한 제한시간)	Diff-HTTP (상위 등급)
차별화 된 서비스 제공	×	×	○
지연의 최소화	×	△	○
신뢰성 보장	×	△	○
서버 자원의 효율적 사용	○	△	△
고품질의 서비스 제공	×	△	○

5. 결론 및 향후 연구 방향

본 논문에서는 HTTP/1.1에서 하나의 TCP 연결 위에 다수의 요청들을 실행하기 위한 지속적인 연결에서의 적절한 TCP 연결 해제 시점을 서버 측에서 차별화된 정책을 사용하여 지원할 수 있는 알고리즘을 제안하였다. 네트워크가 발전하면 발전할수록 서버의 부하가 증가되기 마련이다. 서버는 모든 클라이언트가 요청하는 서비스를 모두 처리할 수 있다면 별 문제가 되지 않지만 클라이언트의 서비스에 대한 요청이 많을 경우, 그 요청을 모두 고품질의 환경으로 처리해 주는 데는 한계가 있다. 따라서 모든 클라이언트들에게 공평하게 서버의 자원을 제공하는 것도 중요하지만 사용자가 다양해지고 서비스를 요구하는 클라이언트가 증가함에 따라 서비스를 차별화 하여 제공하는 것 또한 중요한 문제이다. 본 논문에서는 서버에게 서비스를 요청하는 클라이언트를 상위 등급과 기본 등급의 두 개의 등급으로 나누고 우선 순위를 부여하여 상위 등급의 클라이언트에게 서비스 제한 시간의 가중치를 주어 서비스를 제공하는 Diff-HTTP를 제안하였다. 제안된 Diff-HTTP는 다음과 같은 몇가지 성과를 얻을 수 있었다. 먼저 클라이언트를 차별화 하여 다양한 서비스를 제공하였다. 차별적인 서비스를 제공하는 데 있어 지연을 최대한 줄여 높은 서비스의 품질을 제공하였다. 이는 사용자에게 가중치의 제한 시간을 부여하여 상위 등급의 사용자는 기본 등급의 사용자에 비해 차별성과 신뢰성을 제공하기 위해서이다. 모든 서비스에게 지연을 최소화하기 위해서는 연속적인 제한 시간을 부여하는 방법도 있지만 이것은 서버 자원의 효율적인 측면에 있어서 좋은 방법이 아니다. 따라서 서버 자원의 효율적 사용 측면에 있어서도 좋은 성과를 가져온 것임을 알 수 있다.

본 논문에서 제안한 방안은 서비스의 상용화나 회원 위주의 서비스를 제공하는 경우 꼭 필요한 연구 중에 하나가 될 것이다. 향후 연구 과제는 효율적인 차별화 서비스를 위한 Broker의 효율적인 알고리즘의 연구가 더욱더 필요하다. 또한 프로토콜 차원에서의 서비스의 차별화로부터 더 나아가 OS 차원에서 사용자의 요청에 따른 서비스를 등급 별로 나누어 서버의 리소스를 차별화 하여 제공하는 방법을 연구하고자 한다.

참 고 문 헌

[1] Edith Cohen, Haim Kaplan and Jeffrey Oldham, "Managing TCP connections under persistent HTTP," Elsevier Science B.V, 1999.

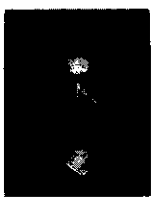
[2] M. Elaoud, C. J Screenan, P. Ramanathan and P. Agrawal, "Use of server load to dynamically select connection closing time for HTTP/1.1 servers," http://www.cae.wisc.edu/~elaoud/research.html

[3] Jussara Almeida, Mihaela Dabu, Anand Manikutty and Pei Cao, "Providing Differentiated Levels of Service in Web Content Hosting, In Proceedings of the 1998 SIGMETRICS Worshop on Internet Server Performance, Madison, WI, USA, PP. 91-102, June 1998.

[4] H.Saran and S.Keshav, "An Empirical Evaluation of Virtual Circuit Holding Times in IP-over-ATM Networks," Journal on Selected Areas Communication 13(1995).

[5] Lars Eggert and John Heidemann, "Application-Level Differentiated Services for Web Servers,"

In World Wide Web Journal, Volume 3, Issue 2, PP.133-142, 1999.
[6] A. Fedlman, R. Caceres, F. Douglis, G.Glass and M.Rabinovich, "Performance of Web proxy caching in heterogeneous bandwidth environments," In Proceedings of the IEEE INFOCOM' 99 Conference, 1999.
[7] H. Frystrk Njelsen, J. Gettys, A. Baird-Smith, E.Prud' hommeaux, H.W. Lie and C.Lilley, "Network Performance effects of HTTP/1.1, CSS1 and PNG," In Proceeding of the ACM SIGCOM M' 97 Conference, Cannes, France, August 1997.
[8] H.Kaplan and E.Cohen, "Reducing user-perceived latency by perfeching connection and per-warming servers," http://www.research.att.com/~edith/publications.html, 1999.
[9] Z.Wang and P.Cao, "Persistent connection behavior of popular browsers," http://www.cs.wisc.edu/cao/paper/persistent-connection.html.
[10] Edith Cohen, Balachander Krishnamurthy, and Jennifer Rexford, "Evaluating Server-Assisted Cache Replacement in the Web," Revision of an ESA' 98.
[11] Venkata N.Padmanabhan and Jeffrey C.Mogul, "Improving HTTP Latency," http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings/DDay/mogul/HTTPLatency.html.
[12] Roy Fielding, "Hypertext Transfer Protocol - HTTP/1.1," Internet Draft (Text / postScript), IETF HTTP WG, August 1996.
[13] John Franks and others, "An Extension to HTTP: Digest Access Authentication," Internet Draft, IETF HTTP WG, September 1996.
[14] Koen Holtman and Andrew Mutz, "Transparent Content Negotiation in HTTP," Internet Draft, IETF HTTP WG, September 1996.
[15] 이윤정, 김태윤, "HTTP에서 서버 자원의 효율적인 이용을 위한 연결관리 연구", 정보처리학회 춘계 학술대회, 2000



현 은 실
1998년 충남대학교 국어국문학과 학사. 1999년 ~ 현재 고려대학교 컴퓨터학과 석사과정 재학. 관심분야는 네트워크 QoS, Wireless network, HTTP

이 윤 정
1993년 숙명여자대학교 전산학과 학사. 1998년 숙명여자대학교 전산학과 석사. 2000년 ~ 현재 고려대학교 컴퓨터학과 박사과정 재학. 관심분야는 컴퓨터 네트워크, 네트워크 보안, 이동통신

김 태 윤
1981년 고려대학교 산업공학과 학사. 1983년 Wayne state University 전산과학과 석사. 1987년 Auburn University 전산과학과 박사. 1998년 ~ 현재 고려대학교 컴퓨터학과 교수. 관심분야는 전자상거래, 컴퓨터 네트워크, EDI, 이동통신 등.