논문 14-39B-06-10 The Journal of Korea Information and Communications Society '14-06 Vol.39B No.06
http://dx.doi.org/10.7840/kics.2014.39B.6.414

모바일 네트워크에서 SDN/NFV 기반의 웹 성능 향상을 위한 웹 캐시 일관성 제공과 JavaScript 전송 가속화 방안

김 기 정◆, 이 성 원°

SDN/NFV Based Web Cache Consistency and JavaScript Transmission Acceleration Scheme to Enhance Web Performance in Mobile Network

Gijeong Kim◆, Sungwon Lee°

요 약

웹 페이지를 구성하는 리소소의 개수와 크기가 점점 증가하고 있으며, 이는 상대적으로 지연이 큰 모바일 네트워크에서 웹 서비스의 급격한 품질 저하로 이어지고 있다. 게다가 현재의 네트워크 구조는 폐쇄적인 구조를 가지고 있기 때문에 웹 서비스의 품질을 향상시키는 프로토콜을 개발할지라도 네트워크 기능으로 제공하기 어렵다는 문제가 있다. 본 논문에서는 모바일 네트워크에서 웹 성능을 향상시키기 위한 2가지 방안으로 Check Coded DOM 방안과 Functional JavaScript 전송 방안을 제안하고, NFV(Network Function Virtualization)를 활용하여 제안 방안이 네트워크 기능으로 제공될 수 있는 방안 모색 해본다. SMPL 라이브러리를 이용한 네트워크 시뮬레이션을 통해서 제안 방안의 성능을 평가하고 분석했으며, 제안 방안이 기존 HTTP 프로토콜보다 페이지 로딩 시간, 네트워크에 발생하는 메시지의 개수, 네트워크에 발생하는 트래픽 측면에서 향상된 성능을 제공함을 확인 할 수 있었다.

Key Words : Web Serveice, Software Defined Networking, Network Function Virtualization

ABSTRACT

The number and size of resource constituting the web page has been increasing steadily, and this circumstance leads to rapidly falling quality of web service in mobile network that offer relatively higher delay. Moreover, Improving the quality of a web services protocol is difficult to provide network function because the current network architecture has closed structure. In this paper, we suggest schemes to enhance web performance in mobile network, which are Check Coded DOM scheme and Functional JavaScript Transmission scheme, and then try to seek idea which can be provided suggested schemes as a network function using NFV(Network Function Virtualization). For the performance evaluation and analysis about the suggested schemes, we perform network simulation using SMPL library. We confirm that suggested schemes offer better performance in term of page loading time, the number of message and the amount of traffic in the network than HTTP Protocol.

※ 이 논문은 0000년도 정부(미래창조과학부)의 재원으로 한국연구재단-차세대정보 · 컴퓨팅기술개발사업의 지원을 받아 수행된 연구임 (NRF-2010-0020727)
◆ First Author : Kyunghee University Department of Computer Engineering, kimgijeong@khu.ac.kr, 학생회원
° Corresponding Author : Kyunghee University Department of Computer Engineering, drsungwon@khu.ac.kr, 정회원
논문번호 : KICS2014-05-198, Received May 27, 2014; Revised June 19, 2014; Accepted June 19, 2014

Ⅰ. 서 론

HTTP 프로토콜은 인터넷을 통해서 다양한 서비스를 제공하는데 광범위하게 사용되고 있다. 하지만 2000년대에 들어서면서 사용자의 다양한 요구사항으로 인한 웹의 규모와 복잡도가 점점 커지면서 웹 서비스의 품질이 급격하게 저하로 이어지고, HTTP 프로토콜은 1999년 이후 이러한 성능 저하를 개선하기 위한 표준화가 이루어지지 않고 있다가 최근 2012년부터 Google의 SPDY를 중심으로 표준화가 진행되고 있다. 더 나아가 사용자가 인터넷에 접속하는 환경 또한 다양해지고 있으며,. 대표적인 예로 무선 통신 환경으로 모바일 네트워크가 있다. 무선 통신 환경을 이용해 인터넷에 접속하는 사례는 점점 증가하고 있고, 모바일 네트워크의 경우, 2013년을 기준으로 가입 건수가 66억에 육박했으며, 앞으로도 계속 증가해 2019년에는 93억의 가입 건수에 도달할 것으로 예상되고 있다. 또 모바일 네트워크는 유선 네트워크보다 지연이 크다는 특징을 가지고 있으며, 3G 모바일 네트워크의 경우 실제로 평균 1.5 Mbps의 대역폭과 180.5 ms의 지연을, 4G 모바일 네트워크의 경우 평균 10.4 Mbps의 대역폭과 88 ms의 지연을 제공한다[1-5].

더 나아가 사용자의 다양한 서비스의 요구사항과 폭발적으로 증가하는 트래픽을 수용하기에 네트워크 구조 및 기술에 한계에 도달했고, 기존의 네트워크 구조를 Control Plane과 Data Plane으로 분리한 구조 및 패킷 제어 기술을 제공하는 SDN(Software Defined Networking)이 주목 받고 있다. 기존의 네트워크 장비는 Control Plane과 Data Plane이 하나의 네트워크 장비에 구현되어 완제품 형태로 제조되었고, 이는 과거의 불안정한 네트워크 환경에서 문제가 발생한 지점을 찾아서 오류를 수정하는데 효과적인 구조였다. 하지만 현재 네트워크는 과거 대비 네트워크의 안정성과 성능이 크게 향상되고, 사용자의 다양한 요구사항을 수용하고 폭발적으로 증가한 트래픽을 효과적으로 제어하는 기술이 요구되고 있다. 또 기존 네트워크 장비는 하드웨어 중심으로 구현되어 있기 때문에 다양한 네트워크 기능을 유연하게 제공하지 못하며, 각 네트워크 기능 마다 개별 장비로 구성되어 있다. 이런 문제를 해결하기 위한 기술로 NFV(Network Function Virtualization)가 있으며, 이는 표준화된 고성능 하드웨어에 다양한 네트워크 기능을 소프트웨어로 가상화하여 제공하는 기술이다. NFV는 다양한 네트워크 기능을 유연하게 제공할 수 있고, 방화벽과 라우터와 같이 서로 다른 기능의 네트워크 장비를 하나의 장비로 구축함으로써 전력 소비량을 감소시키는 효과를 얻을 수 있다[6, 7, 8, 9, 10].

이와 같이 사용자의 다양한 요구사항 만족시키고 향상된 서비스 품질을 제공하기 어려운 네트워크 환경에서, 웹 서비스의 품질을 향상시키는 방안을 제안하고, NFV를 활용하여 제안 방안을 네트워크 기능으로 실현하는 방안에 대해서 모색해 보려 한다. 웹 성능을 향상시킴에 있어 지연은 가장 중요한 요소이다. 사용자는 네트워크의 대역폭보다는 지연에 민감한 특성을 가지고 있다. 이런 특징 때문에 지연을 줄일 수 있는 기술이 필요하고, 이는 웹 서비스 품질을 크게 향상시키는 핵심 요소라 말할 수 있다[11, 12].

현재 웹 페이지를 수신함에 있어서 지연에 영향을 미치는 요소 가운데, 첫 번째는 웹 캐시(Web Cache)이다. 2013년 11월을 기준으로 웹 페이지의 크기는 평균 1653 KB이고, 평균 94 개의 리소스로 구성되어 있다. 특히 웹 페이지를 구성하는 리소스의 개수가 증가할수록 리소스의 요청과 응답이 많아짐으로, 페이지 로딩 시간이 증가하게 된다. 이는 상대적으로 지연이 큰 모바일 네트워크에서는 페이지 로딩 시간이 급격하게 증가는 요인이 된다. 현재는 이와 같은 성능 저하 문제를 해결하기 위해서 웹 캐시를 사용함으로써 지연 시간을 줄이고 있다. 하지만 아무리 캐시 서버(Cache Server)로부터 웹 페이지와 리소스를 제공 받는다고 해도, 웹 캐시의 존재 유무를 확인하기 위해 단말이 캐시 서버로 HTTP Request 메시지를 보내고 HTTP Response 메시지를 수신해야 하기 때문에 여전히 지연 시간이 발생하게 된다[13].

두 번째는 JavaScript이다. 웹 페이지를 구성하는 리소스들을 순차적으로 병렬로 요청하고 수신할 수 있다. 하지만 요청하는 리소스가 JavaScript인 경우, 해당 JavaScript 이후에 위치한 리소스는 병렬로 요청되지 못한다. 브라우저에서 이와 같이 구현한 이유는 JavaScript가 대부분 DOM(Document Object Model) 객체를 생성, 변경, 제거하는 스크립트로 작성되어 있기 때문이다. 만약 또 다른 JavaScript를 병렬로 요청하여 이전 JavaScript 보다 먼저 수신하여 실행하면, 웹 페이지의 일관성이 보장되지 않을 수 있는 문제가 발생할 수 있다[14].

본 논문에서는 지연이 큰 모바일 네트워크에서 페이지 로딩 시간을 줄이기 위한 방안으로, 웹 페이지와 웹 페이지를 구성하는 리소스들 간의 상관관계를 고려하여 웹 리소스의 변경 유무 확인 절차를 감소시키는 방안과 JavaScript의 재사용성을 고려하여 변경된 JavaScript 코드를 함수 단위로 전송함으로써 JavaScript 전송을 가속화하는 방안을 제안한다. 또

NFV를 활용하여 제안 방안이 네트워크 기능으로 제공될 수 있도록 하는 방안을 모색해 보고, 시뮬레이션을 통해서 제안 방안의 성능 평가 및 분석을 수행한다.

Ⅱ. 관련 연구

2.1 웹 캐시 일관성 제공 방안

네트워크에 발생하는 트래픽을 줄이고, 빠른 서비스 응답시간을 사용자에게 제공하기 위해 웹 캐시를 이용한다. 또 웹 캐시는 근원 서버(Origin Server)의 작업량과 네트워크에 발생하는 트래픽을 분산시키는 효과도 얻을 수 있다. 하지만 근원 서버에 존재하는 원본 컨텐츠가 변경되면, 웹 캐시를 제공하는 프록시 서버(Proxy Server) 또는 단말에 존재하는 복사본 컨텐츠와 근원 서버의 원본 컨텐츠 간에 일관성이 유지되지 않을 수 있는 문제가 발생한다. 웹 캐시 일관성을 제공 방안은 조금 더 구체적으로는 약한 캐시 일관성(Weak Cache Consistency)과 강한 캐시 일관성(Strong Cache Consistency)으로 분류할 수 있다. 약한 캐시 일관성은 근원 서버에 존재하는 원본 컨텐츠와, 프록시 서버 또는 단말이 가지고 있는 복사본 컨텐츠가 일관성이 유지되지 않을 수 있는 것을 허용할 수 있는 것을 말한다. 예를 들어 뉴스 기사의 경우, 근원 서버의 원본과 프록시 서버나 단말에 존재하는 복사본의 일관성이 일치하지 않아도 큰 문제를 일으키지 않으며 원본의 변경 주기 또한 상대적으로 길기 때문에 약한 캐시 일관성을 사용하는 적절한 예시라 할 수 있다. 하지만 실시간으로 변하는 주식 정보의 경우, 약한 캐시 일관성을 적용하게 되면 데이터 무결성을 보장함에 있어 심각한 문제를 발생시킬 수 있다. 그러므로 주식 정보와 같이 실시간으로 데이터가 변하고 시간에 민감한 정보들은 강한 캐시 일관성을 적용해야 한다. 강한 캐시 일관성은 근원 서버의 원본과 프록시 서버나 단말의 복사본 간의 일관성을 엄격하게 보장해 주기 때문이다[16, 17].

웹 캐시 일관성을 제공하는 방안으로 크게 약한 캐시 일관성을 제공하는 Time-to-Live와, 강한 캐시 일관성을 제공하는 Polling-every-time, Invalidation, Lease가 있다. 이 방법들은 강한 캐시 일관성을 제공하면 서버의 부하가 증가하고, 서버의 부하를 낮추면 약한 캐시 일관성을 제공하게 되는 trade-off의 문제를 가지고 있다. 현재 웹 서비스에서는 Polling-every-time과 Time-to-Live 기법이 주로 사용되고, 그림 1, 2는 Polling-every-time과 Time-to-Live의 동작과정을 나타낸 것이다. 이와 같은 기존의 웹 캐시 일관성 제공 방안은 웹 페이지와 웹 페이지를 구성하는 리소스가 상

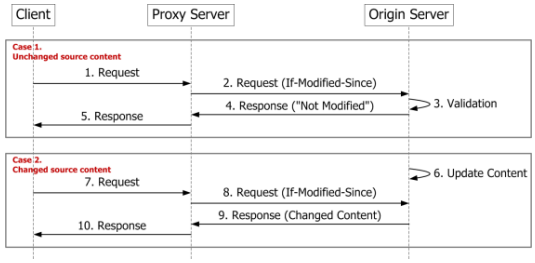

그림 1. Polling-every-time의 캐시 일관성 제공 방안
Fig 1. Cache Consistency Scheme of Polling-every-time

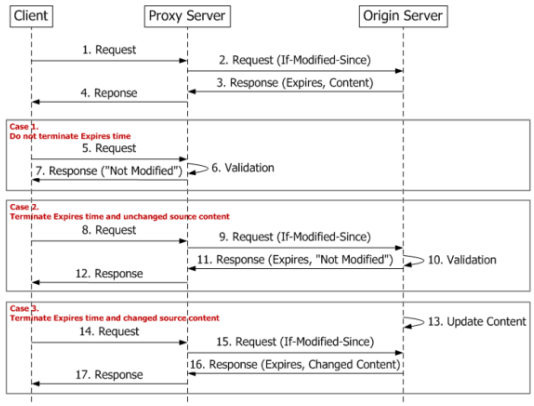

그림 2. Time-to-Live의 캐시 일관성 제공 방안
Fig 2. Cache Consistency Scheme of Time-to-Live

관관계가 있음에도 불구하고, 개별적으로 캐시 일관성을 검증하는 방안이기 때문에 웹 페이지를 구성하는 리소스가 증가하고 복잡해지고 있는 웹 서비스의 품질을 향상시키지 못하는 문제점이 있다[18].

2.2 JavaScript 성능 향상 방안

JavaScript는 네스케이프 커뮤니케이션스사가 개발한 스크립트 언어로, 비영리 표준화 기구인 ECMA International에서 1997년 6월 표준화한 ECMA-262 스크립트 프로그래밍 언어를 채택했으며, DOM(Document Object Model) API가 추가되면서 현재 웹 브라우저에서 사실상 표준 스크립트 언어로 사용되고 있다. JavaScript는 서버와 클라이언트 간에 상호교환적인(Interactive) 웹 서비스를 제공하게 해주고, 시간이나 환경에 따라서 동적인 컨텐츠를 생성하게 해준다. 2013년 11월을 기준으로 평균 웹 페이지의 크기는 1653 KB이며, 그 중에서 267 KB가 JavaScript가 차지하고 있다. JavaScript는 시간이 지남에 따라 크기가 점점 커지고 차지하는 비중이 커지고 있는 추세이다[13].

하지만 JavaScript는 페이지 로딩 시간을 증가시키

는 문제점을 가지고 있다. 이는 웹 브라우저가 다수의 리소스를 포함하는 웹 페이지를 수신할 때, JavaScript를 다운로드하고 실행하는 동안, 해당 JavaScript 이후에 위치한 리소스의 요청을 중단하기 때문이다. 이런 현상을 JavaScript의 블록킹(Blocking) 현상이라 한다[3]. 웹 브라우저가 JavaScript의 병렬 다운로드를 허용하고 수신이 완료된 JavaScript 코드부터 실행을 하게 되면 웹 페이지의 무결성이 보장되지 않을 수도 있는 문제도 있다. 이러한 이유로 웹 브라우저는 JavaScript 요청과 실행에 있어서 블록킹 현상이 발생하도록 구현되어 있다[14].

그럼에도 불구하고 병렬 다운로드를 허용하지 않는 웹 브라우저에서 JavaScript 병렬 요청 및 수신을 가능하게 하는 방법으로는 크게 XHR(XMLHttpRequest), DOM Injection, Iframe Injection 등의 방법이 있다. 해당 기법들은 JavaScript를 병렬 다운로드를 강제적으로 가능하게 하지만 JavaScript를 실행함에 있어 무결성을 보장하지 못한다[14].

웹 페이지를 구성하는 JavaScript의 개수가 증가하고, 그 크기가 점점 커지고 있다는 점을 감안했을 때, 이를 개선할 수 있는 기술은 페이지 로딩 시간을 감소시키는 핵심 요소 중 하나라고 볼 수 있다. 더 나아가 JavaScript를 실행함에 있어 무결성을 보장하기 위해 직렬 다운로드를 수행하되 JavaScript의 전송을 가속화하는 방안이 모색될 필요가 있다[13].

Ⅲ. 제안 방안

3.1 NFV 기반 HTTP 스위칭 네트워크

본 연구에서는 사전에 네트워크의 스위치를 NFV로 구현하고, 해당 NFV에서 HTTP 서비스를 인식하여, HTTP 정보 레벨에서 스위칭을 수행하는 HTTP Switching 개념을 기반으로 동작하는 새로운 개념의 네트워크를 제안한다. 그리고 해당 HTTP Switching을 지원하는 네트워크 장비에서 HTTP 서비스의 성능을 개선하는 세부적인 방안을 제안하도록 한다. HTTP Switching을 지원하는 장비들로 구성한 Switched HTTP Network (SHTTP) 구조가 그림 3에 나타나 있다[15].

그림 4에서 Proxy Access/Switch/Edge는 HTTP 프로토콜을 이해할 수 있는 기능을 구현한 네트워크 스위치 장치이다. 해당 장비는 NFV를 활용하여 본 논문에서 제안하는 Check Coded DOM과 Functional JavaScript 전송 방안을 네트워크 기능으로 유연하게 제공할 수 있다. 그림 4는 제안 방안인 네트워크 기능

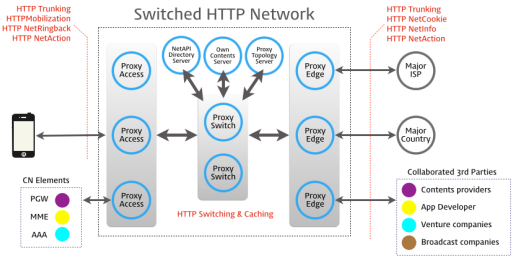

그림 3. HTTP 스위칭 네트워크 구조
Fig 3. HTTP Switching Network Architecture

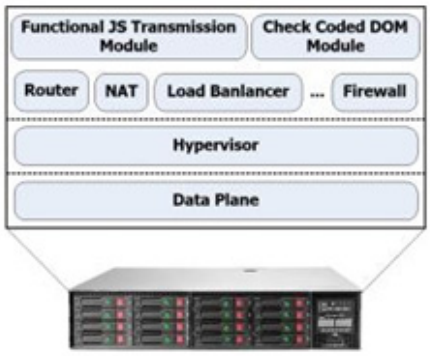

그림 4. NFV 기반의 HTTP 스위칭 장비 구조도
Fig 4. HTTP Switching Equipment Architecture based on NFV

을 제공하기 위한 NFV 네트워크 장비의 구성도이다. 기본적으로 데이터를 전송할 수 있는 Data Plane이 하드웨어로 구성되어 있고, 다양한 네트워크 기능을 소프트웨어로 가상화하는 것을 제공하는 Hypervisor가 있다. 그 상위 레이어에는 라우터나 방화벽 같은 네트워크 기능뿐만 아니라, 제안 방안인 Functional JavaScript Transmission 모듈과 Check Coded DOM 모듈이 어플리케이션 S/W 형태로 위치하게 된다. 이와 같은 구조는 하나의 네트워크 장비가 다양한 기능을 제공하는 것을 가능하게 하고, 더 나아가 하나의 네트워크 장비로 다양한 네트워크 어플라이언스를 생성할 수 있게 된다. 또 Check Coded DOM과 Functional JavaScript 전송 방안과 같이 새로 제안된 프로토콜을 개발하여 네트워크 장비에 쉽게 적용하는 것을 가능하며, 어플리케이션 또는 사용자 별로 플로우를 제어하여 Access Control, Traffic Engineering과 같은 네트워크 제어 기능을 제공할 수 있다.

3.2 Check Coded DOM 방안

본 방안은 서버가 메인 웹 페이지에 웹 페이지를 구성하는 리소스들의 체크코드를 생성하여 삽입함으로

써, 클라이언트가 메인 웹 페이지 수신만으로 웹 페이지를 구성하는 리소스의 변경 유무를 확인할 수 있는 방안이다.

그림 5는 초기 웹 페이지를 생성할 때의 본 방안의 동작 과정을 나타낸 것이다. 메인 서버는 메인 웹 페이지를 생성하고, 그 웹 페이지에 포함되는 리소스는 html 문서와 이미지 파일로, html 문서는 서버 1에 위치하게 되고 이미지 파일은 서버 2에 위치하게 된다. 서버 1과 서버 2에서는 새로운 리소스가 생성되었기 때문에 리소스 내용을 기반으로 SHA-1 알고리즘을 이용해 체크코드를 생성한다. html 문서의 경우, 문서에 작성된 텍스트 내용을 SHA-1 알고리즘의 입력 값으로 사용하고, 이미지의 경우, 이미지 데이터를 표현하는 바이너리 데이터를 SHA-1 알고리즘의 입력 값으로 사용하여 체크코드를 생성한다. 그 다음 과정으로 메인 서버는 웹 페이지에 웹 페이지를 구성하는 리소스의 체크코드가 삽입되어 있는지 확인한다. 초기 웹 페이지 생성 과정이므로, 메인 서버는 각 리소스의 src 속성에 작성되어 있는 URL을 이용해 리소스 ID를 요청한다. 서버 1과 서버 2는 해당 리소스에 대해서 20바이트의 체크코드를 응답한다. 모든 리소스의 체크코드를 응답받은 메인 서버는, 각 리소스에 해당하는 DOM 객체에 id 속성을 추가하고, id 속성 값에 체크코드를 입력한다.

그림 6은 초기 웹 페이지 생성 과정을 마치고 클라이언트가 웹 페이지 요청하여 수신하는 과정을 나타낸 것이다. 클라이언트는 HTTP 요청 메시지를 이용해 체크코드가 포함된 메인 웹 페이지를 수신한다. 클라이언트는 웹 페이지를 구성하는 리소스들의 URL과 체크코드를 Current Resource URL & ID TABLE에 저장한다. 그 다음 Previous Resource URL & ID TABLE에 엔트리가 존재하는지 확인한다. Previous Resource

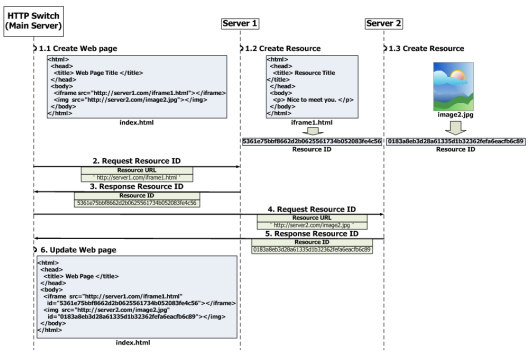

그림 5. Check Coded DOM 방안에서의 초기 웹 페이지 생성 과정
Fig 5. Initial Process of Creating Web Pages via Check Coded DOM

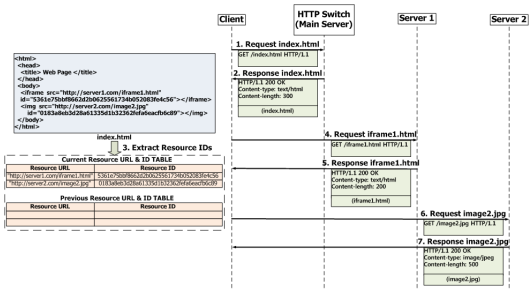

그림 6. Check Coded DOM 방안에서의 초기 웹 페이지와 리소스 요청 및 수신 과정
Fig. 6. Request and Reception Process of Initial Web Page and Resource via Check Coded DOM

URL & ID TABLE에는 엔트리가 존재하지 않게 되므로, 클라이언트가 웹 페이지를 구성하는 리소스에 대해서 요청한 적이 없음을 알 수 있다. 클라이언트는 웹 페이지를 구성하는 리소스들을 요청하여 수신한다. 수신이 완료되면 Current Resource URL & ID TABLE의 엔트리는 Previous Resource URL & ID TABLE로 이동시키고, Current Resource URL & ID TABLE의 엔트리를 모두 삭제한다.

리소스가 변경되지 않은 상태에서 클라이언트가 동일한 웹 페이지를 재요청 할 경우, 그림 7에서 나타난 과정을 수행하게 되고, 이는 그림 6과 같이 클라이언트가 웹 페이지를 요청하여 수신하는 과정과 비슷하다. 클라이언트는 동일한 웹 페이지를 요청하여 수신하고, 웹 페이지를 구성하는 리소스의 URL과 ID를 추출하여 Current Resource URL & ID TABLE의 엔트리로 추가한다. Previous Resource URL & ID TABLE과 Current Resource URL & ID TABLE의 엔트리를 비교하여, Current Resource URL & ID TABLE에는 존재하지만 Previous Resource URL & ID TABLE에는 존재하지 않거나, 두 테이블 모두에 존재하면서 Resource URL은 같지만 Resource ID가 다른 리소스

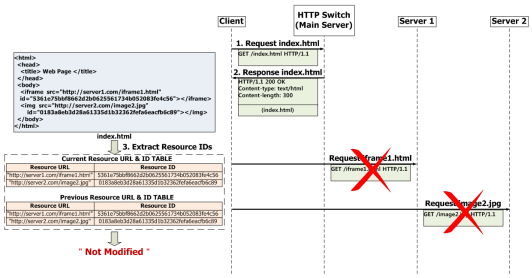

그림 7. 리소스가 변경되지 않은 상태에서 Check Coded DOM 방안의 웹 페이지 재요청 과정
Fig. 7. Recall Process of Web Page via Check Coded DOM with Unchanging Resource

를 찾아낸다. 리소스가 변경되지 않았기 때문에, 두 테이블에 존재하는 엔트리들은 Resource URL과 Resource ID가 일치하게 된다. 이를 통해서 클라이언트는 리소스가 변경되지 않았음을 알 수 있고, 리소스의 변경 유무를 확인하기 위해 리소스를 요청하지 않고 이전에 수신한 리소스를 사용하여 웹 페이지를 구성한다.

제안한 Check Coded DOM 방안은 웹 페이지 수신을 통해서 웹 페이지를 구성하는 리소스의 변경 유무를 확인할 수 있기 때문에 리소스의 변경 유무를 확인하는 절차를 감소시킴으로써 웹 성능을 향상 시킬 수 있다.

3.3 Functional JavaScript 전송 방안

본 방안은 JavaScript의 재사용성을 고려하여 변경된 함수 코드, 함수 변경 정보, 버전 정보를 보냄으로써 전송되는 JavaScript의 크기를 줄이는 방안이다.

그림 8은 서버에서 JavaScript를 생성할 때, 수행되는 처리 과정을 나타낸 그림이다. 서버에서 JavaScript가 생성되면, JavaScript 코드를 함수 단위로 분할한다. 분할된 함수 코드를 입력 값으로 사용하여 SHA-1 알고리즘을 이용해 20 byte의 Function ID를 생성한다. Function & ID TABLE에 함수 이름과 그에 해당하는 Function ID를 삽입하여 테이블을 완성한다. Script ID는 Function ID들을 입력 값으로 받아 SHA-1 알고리즘을 수행하여 Script ID를 생성하고 Script & ID TABLE에 Script Version과 Script ID 항목을 완성한다.

그림 9는 Function & ID TABLE과 Script & ID TABLE이 완성된 상태에서, 클라이언트가 해당 JavaScript를 요청하고 서버가 함수 단위 코드, 함수 변경 정보, 버전 정보를 보낸 과정을 나타낸 것이다. 클라이언트는 A.js가 아닌 A.ajs 파일을 요청한다. ajs 파일은 Archive of JavaScript의 약자로, 본 저자가 정

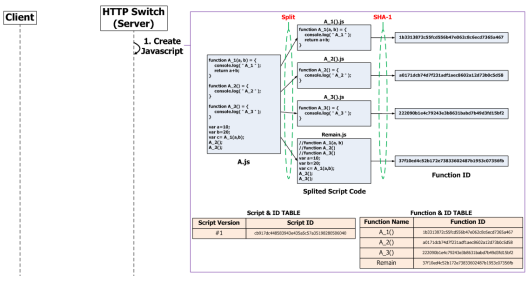

그림 8. 초기 JavaScript 생성 후 Function ID와 Script ID 생성 과정
Fig. 8. Creation Process of Function ID & Script ID After Creation of an Initial JavaScript

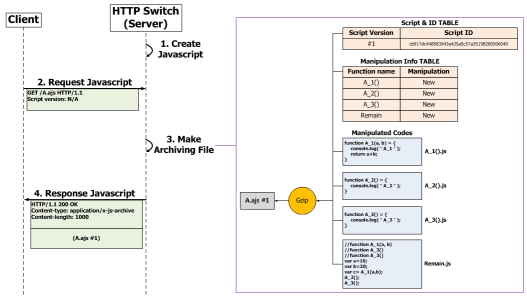

그림 9. 초기 JavaScript에 대한 변경된 함수 코드, 함수 변경 정보, 버전 정보 전송 과정
Fig. 9. Transmission Process of Modified Function Code, and Information on Function Modification and Version about an Initial JavaScript

의한 확장자이고, 함수 단위 코드, 함수 변경 정보, 버전 정보를 Gzip을 이용해 압축한 파일이다. 또 HTTP 요청 메시지에 Script version이라는 헤더를 새로 정의하고, 이는 클라이언트가 가지고 있는 해당 JavaScript의 버전을 나타낸다. 초기 요청이기 때문에 Script version의 헤더 값을 N/A로 설정한다. 이를 수신한 서버는 추가로 Manipulation Info TABLE을 생성한다. 이 테이블은 클라이언트가 가지고 있는 JavaScript와 서버가 가지고 있는 JavaScript를 비교하여 함수 변경 정보를 나타내며, Function name과 Manipulation 속성으로 구성되어 있다. Function name 속성은 JavaScript 코드에 존재하는 함수 이름을 나타내고, Manipulation 속성은 함수의 변경 정보를 나타낸다. Manipulation 속성 값으로는 New, Modified, Removed이 있다. JavaScript 코드가 변경됨에 따라 함수가 새로 생성되면 New 라는 속성 값을 사용하고, 함수의 내용이 변경된 경우에는 Modified 라는 속성 값을 사용하며, 이전 Script 대비 제거된 함수가 발생하면 Removed 라는 속성 값을 사용한다. 초기 요청이기 때문에, Manipulation Info TABLE의 모든 함수들에 대해서 Manipulation 속성 값은 New로 설정된다. Manipulation Info TABLE 생성이 완료되면 Script & ID TABLE, Manipulation Info TABLE, 그리고 Manipulation Info TABLE의 Manipulation 속성이 Modified이거나 New인 함수 코드를 Gzip으로 압축한다. 모든 함수가 New 속성 값을 가지고 있기 때문에, 모든 함수 단위 코드가 Gzip의 압축 대상이 된다. 이 과정이 끝나면, 서버는 A.ajs 파일을 HTTP 응답 메시지에 담아서 클라이언트에게 전송한다.

그림 10는 클라이언트가 서버로부터 A.ajs 파일을 수신하여 JavaScript 코드를 복원하는 과정을 나타낸 것이다. 수신된 A.ajs 파일은 Gzip을 이용해 압축을 해

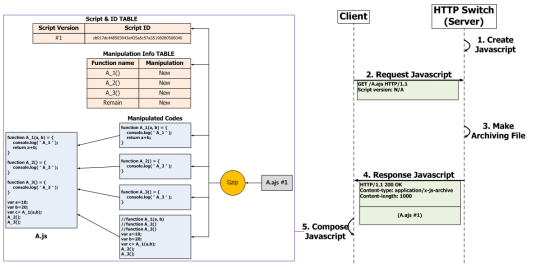

그림 10. 초기 JavaScript를 요청하여 수신된 ajs 파일을 JavaScript 코드로 복원하는 과정
Fig. 10. Restoration Process of ajs file with JavaScript Code by Requesting an Initial JavaScript

제한다. Manipulation Info TABLE을 이용해 함수 단위로 수신된 코드를 조합하여 하나의 JavaScript 파일로 만듦으로써 JavaScript 파일을 수신하게 된다. 추가적으로 분할된 함수 코드들을 SHA-1 알고리즘을 이용해 Function ID를 생성하고, 함수 별로 생성된 Function ID들을 이용해 Script ID를 생성하고 Script & ID TABLE의 Script ID와 일치하는지 확인함으로써 코드의 무결성을 확인할 수 있다.

제안한 Functional JavaScript 전송 방안은 JavaScript가 동영상이나 HTML 문서와 달리 프로그램 소스코드이기 때문에 지속적으로 유지보수가 발생한다. 유지보수로 인해 변경된 함수 부분만 전송함으로써 JavaScript 전송 효율을 높임으로써 웹 성능을 향상시킬 수 있다.

Ⅳ. 성능 평가 및 분석

4.1 시뮬레이션 환경 및 시나리오

제안 방안의 성능을 평가하기 위해서 SMPL 라이브러리를 이용한 네트워크 시뮬레이션을 수행한다[21]. 시뮬레이션 시나리오는 클라이언트가 다수의 JavaScript를 포함하는 웹 페이지를 서버에게 요청하여 응답 받는 시나리오로, 기존 HTTP 프로토콜과 제안 방안에 동일하게 적용된다. 클라이언트는 웹 페이지를 요청하고 수신이 완료되면 웹 페이지를 구성하는 리소스들을 차례대로 요청하고 수신하는 Polling-every-time 방식을 사용한다. 이와 같은 시나리오에서 Check Code DOM 방안과 HTTP 프로토콜의 성능을 비교할 경우에는 Gzip을 적용하지 않고, Functional JavaScript 전송 방안과 HTTP 프로토콜의 성능을 비교할 경우에는 Gzip을 적용하여 시뮬레이션을 실시한다. 그 이유는 Check Coded DOM 방안에는 Gzip 연산이 포함되어 있지 않고, Functional JavaScript Transmission 방안에는 포함되어 있지 않기 때문이다.

시뮬레이션 환경으로는 4G 모바일 네트워크에서 Check Coded DOM과 Functional JavaScript Transmission 기능을 네트워크 기능으로 가상화하여 제공할 수 있는 NFV 네트워크 장비를 포함하는 것으로 그림 11과 같다. 기존 HTTP 프로토콜을 시뮬레이션 할 경우에는 NFV 네트워크 장비가 라우팅 기능만 가상화하고, 제안 방안을 시뮬레이션 할 경우에는 NFV 네트워크 장비가 Check Coded DOM과 Functional JavaScript Transmission 기능을 가상화하여 제공하는 것으로 한다. 4G 모바일 네트워크는 2013년 2월을 기준으로 사용자에게 실제로 제공되고 있는 평균 대역폭과 지연시간을 고려하여, 10 Mbps의 대역폭과 90 ms의 지연시간을 시뮬레이션에 적용했다[2]. 웹 페이지의 크기 및 리소스의 개수는 Naver, Daum 등의 메인 웹 페이지를 실제로 분석하여 시뮬레이션에 적용했으며, 메인 웹 페이지의 크기는 2000 Byte, 웹 페이지를 구성하는 리소스의 개수는 30개로 설정하였다. 웹 페이지를 구성하는 JavaScript의 크기는 html로 작성되는 메인 웹 페이지보다 훨씬 크며, 시간이 지남에 따라 JavaScript 크기의 증가폭이 점점 커지고 있음을 고려하여, 평균 1MB로 지수분포를 가지도록 하였다[13,14].

JavaScript 코드는 객체지향 프로그래밍 기법으로 주로 작성되며, 객체지향 프로그래밍 기법으로 작성될 경우, 코드의 재사용률이 평균 70%라는 점을 감안하

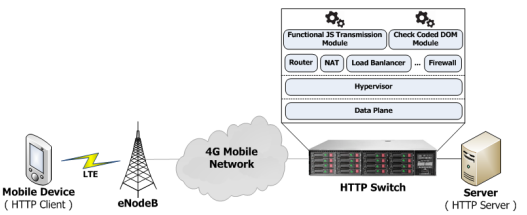

그림 11. 시뮬레이션을 위한 네트워크 아키텍처
Fig. 11. Network Architecture for Simulation

표 1. 시뮬레이션 파라미터
Table 1. Simulation Parameter

Variable	Parameters/Distribution
Network Bandwidth	10 Mbps, Constant
Network Delay	90 ms, Constant
Web Page Size	2000 Byte, Constant
Number of Resource	30
Resource Size (JavaScript)	mean=1 MB, Exponential Distribution
Cache Hit Rate of Resource	0~100%, Uniform Distribution
JavaScript Reusability	70%, Constant
Gzip Compression Ratio	30%, Constant

여 JavaScript의 재사용률을 70%로 설정했다[19]. Gzip 압축률은 파일의 내용과 크기에 무관하게 평균적으로 기대할 수 있는 Gzip의 압축 효율로 30%를 적용했다[20]. 이와 같이 시뮬레이션에 사용된 파라미터는 표 1과 같다.

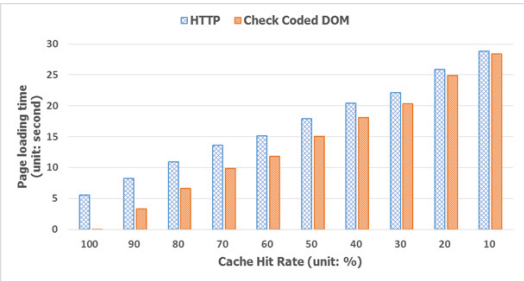

그림 12. HTTP와 Check Code DOM 방안의 페이지 로딩 시간 비교 그래프
Fig. 12. A Comparison Graph of the Page Loading Time in HTTP without Gzip and Check Code DOM

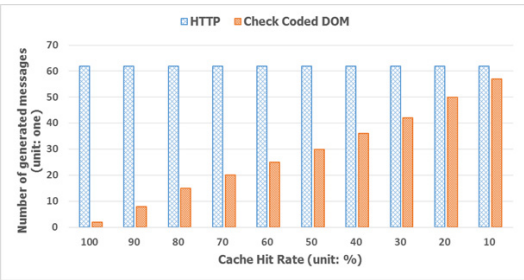

그림 13. HTTP와 Check Code DOM 방안에서 발생한 메시지의 개수 비교 그래프
Fig. 13. A Comparison Graph of the Number of Messages Generated in HTTP without and Check Code DOM

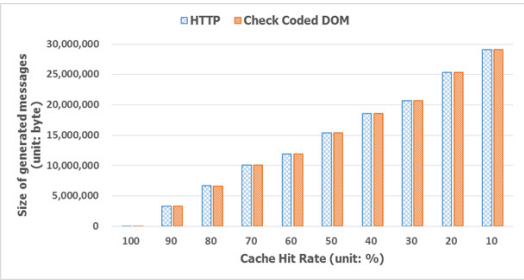

그림 14. HTTP와 Check Code DOM 방안에서 발생한 메시지의 총량 비교 그래프
Fig. 14. A Comparison Graph of the Amount of Messages Generated in HTTP and Check Code DOM

4.2 시뮬레이션 결과 및 분석

캐시 적중률을 100%에서 10%까지 감소시키면서 기존의 HTTP 프로토콜과 제안 방안에 대해서 성능 평가를 수행했다. 난수를 발생시켜 각 케이스마다 16 번씩 시뮬레이션을 수행하고, 측정된 값의 평균을 결과값으로 사용했다.

그림 12, 13, 14는 기존 HTTP 프로토콜과 Check Coded DOM 방안의 페이지 로딩 시간, 네트워크에 발생하는 메시지의 개수, 네트워크에 발생하는 트래픽을 비교한 그래프로, Check Coded DOM 방안이 3 가지 측정 항목에서 더 우수한 성능을 보인다. 특히 캐시 적중률이 높을수록 높은 비율로 성능이 향상되는데, 그 이유는 메인 웹 페이지 수신을 통해서 웹 페이지를 구성하는 리소스의 변경 유무를 확인할 수 있기 때문이다. 그러므로 변경되지 않은 리소스에 대해서 HTTP 요청 메시지를 보내고 HTTP 응답 메시지를 수신하면서 발생하는 불필요한 네트워크 지연을 줄이고, 네트워크에 발생하는 메시지의 개수를 감소시킨다. 또 Check Coded DOM 방안은 Resource ID만큼 트래픽이 증가하지만 변경되지 않은 리소스를 요청하는 HTTP 요청 메시지와 이에 대한 응답 메시지인 Not Modified 메시지를 발생시키지 않기 때문에, 트래픽을 감소시키는 효과를 얻을 수 있다. Check Coded DOM 방안은 기존 HTTP 프로토콜 대비 페이지 로딩 시간을 평균 28.49%, 네트워크에 발생하는 메시지의 개수를 평균 54.03%, 네트워크에 발생하는 트래픽을 평균 8.93% 감소시킨다.

그림 15, 16, 17은 기존 HTTP 프로토콜과 Functional JavaScript 전송 방안의 페이지 로딩 시간, 네트워크에 발생하는 메시지의 개수, 네트워크에 발생하는 트래픽을 비교한 그래프로, 페이지 로딩 시간과 네트워크에 발생하는 트래픽 측면에서 더 우수한 성능을 보인다. Functional JavaScript 정송 방안은 JavaScript의 변경 유무를 확인하기 위한 메시지의 발생을 감소시키는 것이 아닌, 전송되는 JavaScript의 크기를 감소시키기 때문에 네트워크에 발생하는 메시지의 개수 측면에서는 성능을 향상 시키지 못하지만 페이지로딩 시간과 네트워크에 발생하는 트래픽 측면에서 성능을 향상시킬 수 있다. 특히 캐시 적중률이 감소할수록 높은 비율로 성능이 향상되며, 그 이유는 JavaScript 코드의 변경된 부분만 전송하는 기회가 증가하기 때문이다. Functional JavaScript 전송 방안은 기존 HTTP 프로토콜 대비 페이지 로딩 시간을 평균 35.55%, 네트워크에 발생하는 트래픽을 평균 62.73% 감소시킴을 확인 할 수 있다.

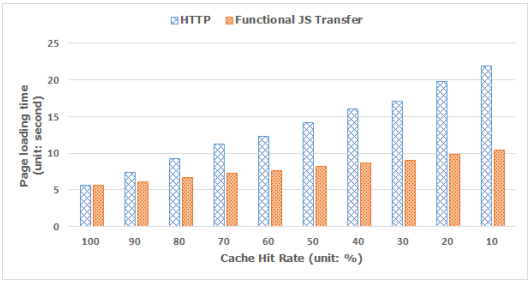

그림 15. HTTP와 Functional JavaScript 전송 방안의 페이지 로딩 시간 비교 그래프
Fig. 15. A Comparison Graph of the Page Loading Time in HTTP with Gzip and Functional JavaScript Transmission

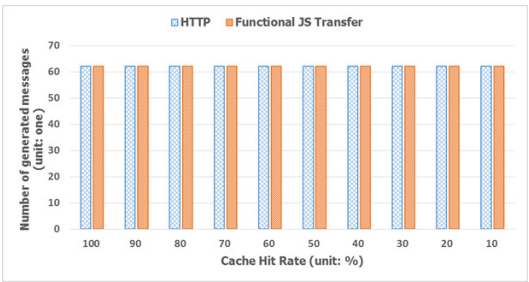

그림 16. HTTP와 Functional JavaScript 전송 방안에서 발생한 메시지의 개수 비교 그래프
Fig. 16. A Comparison Graph of the Number of Messages Generated in HTTP with Gzip and Functional JavaScript Transmission

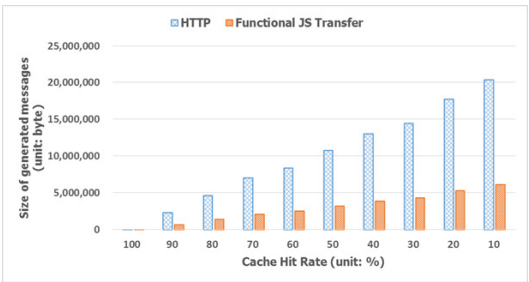

그림 17. HTTP와 Functional JavaScript 전송 방안에서 발생한 메시지의 총량 비교 그래프
Fig. 17. A Comparison Graph of the Amount of Messages Generated in HTTP with Gzip and Functional JavaScript Transmission

V. 결 론

본 논문에서는 상대적으로 지연이 큰 모바일 네트워크에서 웹 성능을 향상시키기 위한 Check Coded DOM과 Functional JavaScript 전송 방안을 제안했다. 또 제안 방안을 네트워크 기능으로 제공할 수 있는 HTTP 스위칭 네트워크 구조를 제안하고, 이를 실현하기 위한 NFV 장비 구조를 모색해 보았다. Check Coded DOM 방안은 변경되지 않은 리소스에 대해서 변경 유무를 확인하는 절차를 감소시키고, Functional JavaScript 전송 방안은 전송되는 JavaScript의 재사용성을 고려하여 전송되는 JavaScript의 크기를 줄임으로써 웹 성능을 향상시킴을 시뮬레이션을 통해서 확인할 수 있었다. 더 나아가 NFV를 통해서 새로운 요구사항을 반영한 프로토콜을 네트워크 기능으로써 유연하고 제공할 수 있는 방안을 모색해 봄으로써, 새로운 네트워크 장비를 추가하지 않고 소프트웨어 개발만으로 네트워크 기능을 확장할 수 있음을 확인해 볼 수 있었다.

웹 서비스뿐만 아니라, 다양한 사용자의 요구사항을 충족시키기 위해서는 폐쇄적인 현재의 네트워크 구조에 큰 변화가 필요하다. 그 변화에 핵심 주축으로 SDN과 NFV가 자리 잡고 있고, 다양한 요구사항을 반영한 기능을 네트워크에 배치할 수 있을 것이라 기대되고 있다. 앞으로 SDN과 NFV의 실현 가능성을 확장하고, 다양한 서비스 요구사항을 충족시키는 프로토콜에 대한 연구개발이 활발히 이루어진다면 더욱 향상된 서비스 품질을 제공할 수 있는 네트워크를 실현할 수 있을 것이라 전망된다.

References

[1] Ericsson, Ericsson Mobility Report(2013), Retrieved May, 18, 2014, from http://www.ericsson.com/res/docs/2013/ericsson-mobility-report-november-2013.pdf.

[2] OpenSignal, The State of LTE(2013), Retrieved May 18, 2014, from http://opensignal.com/reports/state-of-lte/.

[3] I. Grigorik, *High Performance Browser Networking*, O'Reilly, 2013.

[4] A. Technology, The State of the Internet(2013), Retrieved May, 18, 2014, from http://www.akamai.com/dl/documents/akamai_soti_q213.pdf.

[5] M. Belshe, et al., Hypertext Transfer Protocol version 2(2014), Retrieved Jun. 18, 2014, from http://tools.ietf.org/html/draft-ietf-httpbis-http2-13

[6] Open Networking Foundation, Software-Defined Networking: The New Norm for Networks (2012), Retrieved May, 18, 2014, from https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newn

orm.pdf.

[7] European Telecommunications Standards Institute (ETSI), Network Function Virtualization (2013), Retrieved May, 18 2014, from http://portal.etsi.org/nfv/nfv_white_paper2.pdf

[8] J. Kim, "Building OpenFlow based SDN Testbed for future service demonstration," *J. KICS*, vol. 30, no. 3, pp. 43-50, Mar. 2013.

[9] S. Lee and H. Lee, "SDN technology in wireless mobile network," *J. KICS*, vol. 30, no. 3, pp. 16-21, Mar. 2013.

[10] J. Lee, S. Lee, G. Choi, and B. Lee, "NFV(Network Functions Virtualisation)," *J. KICS*, vol. 30, no. 3, pp. 51-57, Mar. 2013.

[11] Jakob Nielsen, Response Times: The 3 Important Limits(1993), Retrieved May, 18, 2014, from http://www.nngroup.com/articles/response-times-3-important-limits/.

[12] J. Arvind and G. Jason, Use Compression to make the web faster(2012), Retrieved May, 18, 2014, from https://developers.google.com/speed/articles/use-compression.

[13] HTTP Archive, Trend in web technology (2014), Retrieved May, 18, 2014, form http://httparchive.org/.

[14] Steve Souders, *Even Faster Web Sites*, O'Reilly, 2009.

[15] H. Kim and S. Lee, "HTTP switching network for mobile broadband networks," in *Proc. FTRA AIM-14*, pp. 1-2, Jeju, Korea, Feb. 2014.

[16] H. Liu and M. Chen, "Evaluation of web caching consistency," *Int. Conf. Advanced Comput. Theory and Eng.*, vol. 5, pp. 130-132, Chengdu, Aug. 2010.

[17] J. Liu, Y. Wang, and H. Du, "Strong cache consistency on world wide web," *Int. Conf. Advanced Comput. Theory and Eng.*, vol. 5, pp. 62-65, Aug. 2010.

[18] P. Cao and C. Liu, "Maintaining strong cache consistency in the world," *IEEE Trans. Computers*, vol. 47, no. 4, pp. 445-457, Apr. 1998.

[19] P. Gandhi and P. K. Bhatia, "Estimation of generic reusability for object-oriented software an empirical approach," *ACM SIGSOFT Softw. Eng. Notes*, vol. 36, no. 3, pp. 1-4, May 2011.

[20] J.-L. Gailly, GNU Gzip(2013), Retrieved Apr. 8, 2014, from http://www.gnu.org/software/gzip/manual/gzip.html.

[21] H. M. MacDougall, *Simulating Computer System: Techniaues and Tools*, MIT Press Series in Computer System, 1987.

김 기 정 (Gijeong Kim)

2012년 : 경희대학교 컴퓨터공학과 학사

2014년 : 경희대학교 컴퓨터공학과 석사

2014년 : 9월~현재 : 경희대학교 컴퓨터공학과 박사과정

<관심분야> 이동통신, 미래인터넷, 클라우드 컴퓨팅

이 성 원 (Sungwon Lee)

1998년 경희대학교 박사

1999년~2008년 삼성전자 정보통신 총괄

2008년~현재 : 경희대학교 컴퓨터공학과 부교수

<관심분야> 이동통신, 이동통신 서비스, SDN,미래인터넷