



(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2015-0135845
(43) 공개일자 2015년12월04일

(51) 국제특허분류(Int. Cl.)

H04L 29/08 (2006.01)

(21) 출원번호 10-2014-0062888

(22) 출원일자 2014년05월26일

심사청구일자 없음

(71) 출원인

삼성전자주식회사

경기도 수원시 영통구 삼성로 129 (매탄동)

경희대학교 산학협력단

경기도 용인시 기흥구 덕영대로 1732 (서천동, 경희대학교 국제캠퍼스내)

(72) 발명자

임한나

서울특별시 성동구 뚝섬로 51, 103동 101호

이성원

경기도 용인시 기흥구 덕영대로 1732 경희대학교 전자정보대학관

이지철

경기도 수원시 영통구 효원로 363 신매탄위브하늘 채아파트 130동 906호

(74) 대리인

윤동열

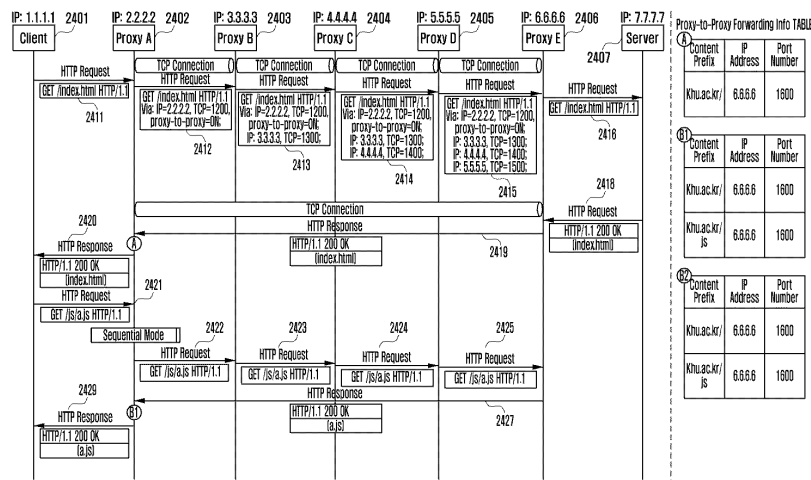
전체 청구항 수 : 총 18 항

(54) 발명의 명칭 통신 네트워크에서 성능 개선을 위한 방법 및 장치

(57) 요약

본 발명의 실시 예에 따르면, 제1 서버와 제2 서버 및 제1 서버와 제2 서버를 연결하는 적어도 하나의 연결 서버를 포함하는 통신 시스템에서 제1 서버의 통신 방법에 있어서, 적어도 하나의 클라이언트(client) 장치로부터 컨텐츠 요청을 수신하는 단계, 상기 제2 서버로부터 데이터를 직접 수신하기 위한 우회 연결을 설정하는 단계, 상기 우회 연결을 이용하여 상기 제2 서버로부터 직접 상기 컨텐츠에 대한 데이터를 수신하는 단계 및 상기 데이터를 상기 클라이언트 장치로 전송하는 단계를 포함하는 것을 특징으로 하는 방법 및 이를 수행하는 장치를 제공할 수 있다.

대표도



명세서

청구범위

청구항 1

제1 서버와 제2 서버 및 제1 서버와 제2 서버를 연결하는 적어도 하나의 연결 서버를 포함하는 통신 시스템에서 제1 서버의 통신 방법에 있어서,

적어도 하나의 클라이언트(client) 장치로부터 콘텐츠 요청을 수신하는 단계;

상기 제2 서버로부터 데이터를 직접 수신하기 위한 우회 연결을 설정하는 단계;

상기 우회 연결을 이용하여 상기 제2 서버로부터 직접 상기 콘텐츠에 대한 데이터를 수신하는 단계; 및

상기 데이터를 상기 클라이언트 장치로 전송하는 단계를 포함하는 것을 특징으로 하는 방법.

청구항 2

제1항에 있어서, 상기 우회 연결은,

상기 연결 서버를 경유하지 않고 상기 제2 서버로부터 직접 데이터를 수신할 수 있는 바이패스(by-pass) 전송 제어 프로토콜(TCP, Transmission Control Protocol)을 설정하는 것을 특징으로 하는 방법.

청구항 3

제1항에 있어서, 상기 우회 연결을 설정하는 단계는,

상기 제1 서버의 IP 주소 또는 TCP 포트 번호 중 적어도 하나의 정보에 대응하는 프록시 간 우회 연결 파라미터를 생성하는 단계,

상기 생성된 파라미터를 상기 연결 서버를 경유하여 상기 제2 서버로 전송하는 단계, 그리고

상기 제2 서버의 바이패스 TCP 연결 요청에 기반하여 우회 연결을 설정하는 단계를 포함하는 것을 특징으로 하는 방법.

청구항 4

제1항에 있어서,

상기 제1 서버, 상기 제2 서버 및 상기 연결 서버는 프록시 서버(Proxy Server)인 것을 특징으로 하는 방법.

청구항 5

제1항에 있어서, 상기 제2 서버로부터 직접 데이터를 수신하는 단계는,

상기 제1 서버와 상기 제2 서버 사이에 상기 연결 서버를 경유하지 않고, 상기 데이터를 수신하는 것을 특징으로 하는 방법.

청구항 6

제1항에 있어서,

상기 우회 연결을 통해 수신한 데이터에 대한 콘텐츠 프리픽스(content prefix), 상기 제2 서버의 IP 주소와 TCP 포트 번호를 포함하는 프록시 간 정보를 업데이트하는 단계; 및

상기 프록시 간 정보에 기반하여, 적어도 하나의 클라이언트로부터 수신하는 콘텐츠 요청에 대해 상기 설정된 우회 연결을 이용할지 여부를 결정하는 단계를 더 포함하는 것을 특징으로 하는 방법.

청구항 7

제1항에 있어서, 상기 제2 서버와 상기 데이터 수신을 위한 인증을 수행하는 단계를 더 포함하고,

상기 인증을 수행하는 단계는,

상기 제2 서버로부터 인증 요청 메시지를 수신하는 단계,

상기 상기 제2 서버로 인증 허가 메시지를 전송하는 단계, 그리고

상기 제2서버로부터 접속 허가 메시지를 수신하는 단계를 포함하는 것을 특징으로 하는 방법.

청구항 8

제1항에 있어서,

상기 우회 연결에 대한 유지시간을 설정하는 단계를 더 포함하고,

상기 제1 서버와 상기 제2 서버의 우회 연결은 상기 유지시간 동안 지속되는 것을 특징으로 하는 방법.

청구항 9

제1항에 있어서,

상기 제2 서버와 푸시(PUSH)를 설정하는 단계; 및

상기 푸시를 설정하면, 상기 제2 서버의 콘텐츠가 변경될 때 상기 우회 연결을 이용하여 변경된 데이터를 수신하는 단계를 더포함하는 것을 특징으로 하는 방법.

청구항 10

제1 서버와 제2 서버 및 제1 서버와 제2 서버를 연결하는 적어도 하나의 연결 서버를 포함하는 통신 시스템에서 통신하는 제1 서버에 있어서,

적어도 하나의 네트워크 노드와 통신을 수행하는 통신부; 및

적어도 하나의 클라이언트(client) 장치로부터 콘텐츠 요청을 수신하고, 상기 제2 서버로부터 데이터를 직접 수신하기 위한 우회 연결을 설정하며, 상기 우회 연결을 이용하여 상기 제2 서버로부터 직접 상기 콘텐츠에 대한 데이터를 수신하고, 상기 데이터를 상기 클라이언트 장치로 전송하도록 제어하는 제어부를 포함하는 것을 특징으로 하는 제1 서버.

청구항 11

제10항에 있어서, 상기 우회 연결은,

상기 연결 서버를 경유하지 않고 상기 제2 서버로부터 직접 데이터를 수신할 수 있는 바이패스(by-pass) 전송 제어 프로토콜(TCP, Transmission Control Protocol)을 설정하는 것을 특징으로 하는 제1 서버.

청구항 12

제10항에 있어서, 상기 제어부는,

상기 제1 서버의 IP 주소 또는 TCP 포트 번호 중 적어도 하나의 정보에 대응하는 프록시 간 우회 연결 파라미터를 생성하고, 상기 생성된 파라미터를 상기 연결 서버를 경유하여 상기 제2 서버로 전송하며, 상기 제2 서버의 바이패스 TCP 연결 요청에 기반하여 우회 연결을 설정하도록 제어하는 것을 특징으로 하는 제1 서버.

청구항 13

제10항에 있어서,

상기 제1 서버, 상기 제2 서버 및 상기 연결 서버는 프록시 서버(Proxy Server)인 것을 특징으로 하는 제1 서버.

청구항 14

제10항에 있어서, 상기 제어부는,

상기 제1 서버와 상기 제2 서버 사이에 상기 연결 서버를 경유하지 않고, 상기 데이터를 수신하도록 제어하는

것을 특징으로 하는 제1 서버.

청구항 15

제10항에 있어서, 상기 제어부는,

상기 우회 연결을 통해 수신한 데이터에 대한 콘텐츠 프리픽스(content prefix), 상기 제2 서버의 IP 주소와 TCP 포트 번호를 포함하는 프록시 간 정보를 업데이트하고, 상기 프록시 간 정보에 기반하여, 적어도 하나의 클라이언트로부터 수신하는 콘텐츠 요청에 대해 상기 설정된 우회연결을 이용할지 여부를 결정하도록 제어하는 것을 특징으로 하는 제1 서버.

청구항 16

제10항에 있어서, 상기 제어부는,

상기 제2 서버로부터 인증 요청 메시지를 수신하고, 상기 상기 제2 서버로 인증 허가 메시지를 전송하며, 상기 제2서버로부터 접속 허가 메시지를 수신하여 상기 제2 서버와 상기 데이터 수신을 위한 인증을 수행하도록 제어하는 것을 특징으로 하는 제1 서버.

청구항 17

제10항에 있어서, 상기 제어부는,

상기 우회 연결에 대한 유지시간을 설정하도록 제어하고,

상기 제1 서버와 상기 제2 서버의 우회 연결은 상기 유지시간 동안 지속되는 것을 특징으로 하는 제1 서버.

청구항 18

제10항에 있어서, 상기 제어부는,

상기 제2 서버와 푸시(PUSH)를 설정하고, 상기 푸시를 설정하면, 상기 제2 서버의 콘텐츠가 변경될 때 상기 우회 연결을 이용하여 변경된 데이터를 수신하도록 제어하는 것을 특징으로 하는 장치.

발명의 설명

기술 분야

[0001] 본 발명은 통신 네트워크에서 통신 성능을 개선하기 위한 방법 및 장치에 관한 것이다. 더욱 자세히, 본 발명은 네트워크에서 HTTP(Hyper Text Transfer Protocol) 프로토콜 성능 개선을 위한 방법 및 장치에 관한 것이다.

배경 기술

[0002] 최근 네트워크 기술의 발달로 전자 장치를 통하여 웹 콘텐츠(web content)를 이용하는 기술이 발전하고 있다. 웹 콘텐츠 이용 환경을 위해서 캐시 일관성 정책, 프록시 서버, 최적화 콘텐츠 제공 기술, 자바스크립트(Javascript)의 크기를 줄이는 기술 등이 적용되고 있다. 하지만 이러한 기술들은 하기와 같은 문제가 있다.

[0003] 첫 번째, 캐시 일관성 정책(Cache Consistency Scheme)에서는 강한 캐시 일관성(Strong Cache Consistency)이 제공되면 네트워크의 부하 또는 서버의 부하를 증가시키고, 네트워크의 부하 또는 서버의 부하를 낮추면 강한 캐시 일관성을 제공하지 못하게 되는 트레이드 오프(trade-off) 문제를 가지고 있다.

[0004] 두 번째, 프록시 체인(Proxy Chain)에서는, 클라이언트가 전송한 HTTP 요청 메시지는 프록시 체인을 따라 서버에게 전달된다. 또한, 서버가 전송한 HTTP 응답 메시지는, HTTP 요청 메시지가 전달 경로를 역으로 추적하여 클라이언트에게 전송된다. 다수의 프록시 서버로 프록시 체인을 구성하고 있는 경우, HTTP 메시지가 프록시 서버를 지날 때마다 전송 지연이 발생하는 문제점이 있다. 더 나아가 프록시 체인을 구성하는 프록시 서버 중에서, 특정 프록시 서버가 웹 캐시를 제공하게 되면, 웹 캐시를 제공하는 프록시 서버로 부하가 집중되는 문제점이 있다.

[0005] 세 번째, HTTP 메시지의 사용자 에이전트(User-Agent)와 같은 클라이언트 정보를 이용해, 프록시 서버 또는 웹 서버가 최적화된 콘텐츠를 제공하는 기술이 있다. 이 기술은 서로 다른 종류의 단말에게 서로 다른 웹 페이지를 전송하거나, 단말의 화면 크기에 맞게 이미지의 크기를 축소하거나 확대하여 전송할 수 있다. 하지만 이 기술은

웹 콘텐츠를 단말에서 실행되기 적합한 콘텐츠로 최적화할 뿐, 통신 및 네트워크의 상태를 고려한 자바스크립트(JavaScript) 코드 최적화를 제공하지는 못한다.

[0006] 네 번째, 자바스크립트(JavaScript)는 웹 페이지를 구성하는 리소스인 동시에, 프로그램 코드이기 때문에, 지속적으로 유지보수 과정이 수행된다. 종래의 기술은 공백 문자와 불필요한 코드를 제거하여 전송되는 자바스크립트(JavaScript) 코드를 줄일 뿐, 재사용성을 고려하여 자바스크립트(JavaScript)의 크기를 줄이는 기능을 제공하지는 못한다.

[0007] 따라서 이러한 종래 기술의 단점을 보완하여 웹 콘텐츠 이용 환경을 개선 시킬 수 있는 기술이 요구되고 있다.

발명의 내용

해결하려는 과제

[0008] 본 발명이 이루고자 하는 기술적 과제는 통신 네트워크에서 통신 성능을 개선하기 위한 방법 및 장치를 제공하는 것이다. 더욱 자세히, 본 발명은 네트워크에서 HTTP(Hyper Text Transfer Protocol) 프로토콜 성능 개선을 위한 방법 및 장치를 제공할 수 있다.

과제의 해결 수단

[0009] 본 발명의 실시 예에 따르면, 제1 서버와 제2 서버 및 제1 서버와 제2 서버를 연결하는 적어도 하나의 연결 서버를 포함하는 통신 시스템에서 제1 서버의 통신 방법에 있어서, 적어도 하나의 클라이언트(client) 장치로부터 콘텐츠 요청을 수신하는 단계, 상기 제2 서버로부터 데이터를 직접 수신하기 위한 우회 연결을 설정하는 단계, 상기 우회 연결을 이용하여 상기 제2 서버로부터 직접 상기 콘텐츠에 대한 데이터를 수신하는 단계 및 상기 데이터를 상기 클라이언트 장치로 전송하는 단계를 포함하는 것을 특징으로 하는 방법을 제공할 수 있다.

[0010] 또한, 본 발명의 실시 예에 따르면, 제1 서버와 제2 서버 및 제1 서버와 제2 서버를 연결하는 적어도 하나의 연결 서버를 포함하는 통신 시스템에서 통신하는 제1 서버에 있어서, 적어도 하나의 네트워크 노드와 통신을 수행하는 통신부 및 적어도 하나의 클라이언트(client) 장치로부터 콘텐츠 요청을 수신하고, 상기 제2 서버로부터 데이터를 직접 수신하기 위한 우회 연결을 설정하며, 상기 우회 연결을 이용하여 상기 제2 서버로부터 직접 상기 콘텐츠에 대한 데이터를 수신하고, 상기 데이터를 상기 클라이언트 장치로 전송하도록 제어하는 제어부를 포함하는 것을 특징으로 하는 제1 서버를 제공할 수 있다.

발명의 효과

[0011] 본 발명의 실시 예에 따르면, 통신 네트워크에서 HTTP(Hyper Text Transfer Protocol) 프로토콜 성능 개선을 위한 방법 및 장치를 제공할 수 있다.

[0012] 또한, 본 발명의 실시 예에 따르면, 웹 페이지를 구성하는 리소스의 변경 유무를 확인하기 위한 메시지 전송이 감소함으로써, 페이지 로딩 시간을 단축 시킬 수 있다. 본 발명의 실시 예에 따르면, 프록시 체인을 우회하여 콘텐츠를 요청하고 응답 받을 수 있기 때문에 프록시 서버를 거칠 때마다 발생하는 HTTP 메시지 처리 시간을 제거함으로써, HTTP 메시지의 전송 시간을 단축할 수 있다.

[0013] 또한, 본 발명의 실시 예에 따르면 프록시 서버가 단말의 통신 링크를 고려하여, 단말에 전송할 Javascript 코드를 바이트 코드 또는 네이티브 코드로 변환하여 전송함으로써, 단말에서 웹 페이지가 실행되는 시간을 줄일 수 있다. 또한, 단말이 웹 브라우저가 지원하지 않는 스크립트 코드를 요청했을 경우, 프록시 서버가 단말에서 실행 가능한 스크립트 언어의 코드로 변환하여 전송함으로써, 웹 서비스의 실행 가능성을 높여 준다.

[0014] 또한, 본 발명의 실시 예에 따르면, 단말이 하위 버전의 웹 브라우저에서 실행이 불가능한 스크립트 코드를 요청했을 경우, 프록시 서버가 하위 버전의 웹 브라우저에서 실행 가능하도록, 단말이 요청한 스크립트 코드에 프레임워크나 API를 추가하여 전송함으로써, 웹 서비스의 실행 가능성을 높여 준다.

[0015] 또한, 본 발명의 실시 예에 따르면, 프록시 서버가 단말의 통신 링크를 고려하여, 단말에 전송할 자바스크립트(Javascript) 코드를 바이트 코드 또는 네이티브 코드로 변환하여 전송할 경우, 변환된 파일을 분할하여 전송하는 것이 가능하다. 파일의 크기가 크거나 네트워크에 지연이 발생한 경우, 프록시 서버가 분할된 파일을 단말에 전송함으로써, 웹 서비스의 실행 시간을 단축할 수 있다.

[0016] 또한, 본 발명의 실시 예에 따르면, 자바스크립트(JavaScript) 코드는 재사용성을 고려하여 지속적인 유지보수

과정이 수행된다. 재사용성의 특성을 고려하여 변경된 함수 코드와 함수 변경 정보를 전송함으로써, 전송되는 JavaScript 코드의 크기를 줄일 수 있기 때문에, JavaScript 전송 시간과 블록킹 시간을 단축 시킬 수 있다.

또한, 본 발명의 실시 예에 따르면, 자주 쓰이거나 자주 쓰일 것이라고 생각되는 JavaScript 코드를 Core JavaScript로 선정하여 미리 전송한다. 클라이언트가 JavaScript 코드를 요청하면, 미리 전송된 Core JavaScript 코드를 제외한 코드 부분과 제외된 코드의 인덱스 정보를 전송함으로써, 전송되는 JavaScript 코드의 크기를 줄일 수 있기 때문에, JavaScript 전송 시간과 블록킹 시간을 단축 시킬 수 있다.

도면의 간단한 설명

도 1은 polling-every-time 기법에 기반하여 캐시 일관성을 제공하는 방법을 설명하는 도면이다.

도 2는 time-to-live 기법에 기반하여 캐시 일관성을 제공하는 방법을 설명하는 도면이다.

도 3은 invalidation 기법에 기반하여 캐시 일관성을 제공하는 방법을 설명하는 도면이다.

도 4는 lease 기법에 기반하여 캐시 일관성을 제공하는 방법을 설명하는 도면이다.

도 5는 복수의 프록시 체인에 대한 통신 과정을 설명하는 도면이다.

도 6은 HTTP 요청 메시지를 이용한 최적화 콘텐츠 제공 방법을 설명하는 도면이다.

도 7은 자바스크립트를 포함하는 웹 페이지와 리소스의 수신을 설명하는 도면이다.

도 8은 JavaScript를 병렬을 허용해서, 먼저 수신된 JavaScript를 실행할 경우, 문제를 발생시킬 수 있는 예제 코드이다.

도 9는 본 발명의 일 실시 예에 따른 통신 시스템을 설명하는 도면이다.

도 10은 본 발명의 일 실시 예에 따른 단말을 설명하는 도면이다.

도 11은 본 발명의 일 실시 예에 따른 웹 서버를 설명하는 도면이다.

도 12는 본 발명의 일 실시 예에 따른 프록시 서버를 설명하는 도면이다.

도 13은 본 발명의 일 실시 예에 따른 자원 아이디 테이블을 설명하는 도면이다.

도 14는 본 발명의 일 실시 예에 따른 프록시 바이패스 커넥션 매핑 테이블을 설명하는 도면이다.

도 15는 웹 브라우저가 지원하는 스크립트 언어 정보를 설명하는 도면이다.

도 16은 브라우저 버전에 따라 지원 가능한 자바스크립트 정보를 설명하는 도면이다.

도 17은 본 발명의 일 실시 예에 따른 자바 스크립트를 구성하는 함수 이름과 함수 아이디의 매핑 관계를 설명하는 도면이다.

도 18은 본 발명의 일 실시 예에 따른 자바 스크립트의 버전과 ID를 저장하는 테이블을 설명하는 도면이다.

도 19는 본 발명의 일 실시 예에 따른 자바스크립트 코드를 구성하는 함수 이름과 함수 변경 정보 매핑 관계를 설명하는 도면이다.

도 20은 본 발명의 제1 실시 예에 따른 초기 웹 페이지 생성 과정을 설명하는 도면이다.

도 21은 본 발명의 제1 실시 예에 따른 클라이언트의 웹 페이지 수신 과정을 설명하는 도면이다.

도 22는 본 발명의 제1 실시 예에 따른 리소스 변경 시 웹 페이지 수정 과정을 설명하는 도면이다.

도 23은 본 발명의 제1 실시 예에 따른 클라이언트의 웹 페이지 수신 과정을 설명하는 도면이다.

도 24는 본 발명의 제2 실시 예에 따른 바이패스 TCP 커넥션을 이용한 웹 페이지 수신 방법을 설명하는 도면이다.

도 25는 본 발명의 제2 실시 예에 따른 프록시 서버 간 인증 방법을 설명하는 도면이다.

도 26은 본 발명의 제2 실시 예에 따른 프록시 서버 간 연결 시간 설정 방법을 설명하는 도면이다.

도 27은 본 발명의 제2 실시 예에 따른 프록시 서버 간 콘텐츠 푸시 설정 방법을 설명하는 도면이다.

도 28 및 도 29는 본 발명의 제3 실시 예에서 스크립트 코드를 변환하여 전송하는 방법을 설명하는 도면이다.

도 30은 본 발명의 제3 실시 예에서 스크립트 언어를 변환하여 전송하는 방법을 설명하는 도면이다.

도 31은 본 발명의 제3 실시 예에서 브라우저 버전에 따라 스크립트 코드를 변환하여 전송하는 방법을 설명하는 도면이다.

도 32는 본 발명의 제3 실시 예에서 통신 링크에 따라 파일을 분할하여 전송하는 방법을 설명하는 도면이다.

도 33은 본 발명의 제4 실시 예에서 서버의 자바스크립트를 생성 시 처리 과정을 설명하는 도면이다.

도 34는 본 발명의 제4 실시 예에 따른 자바스크립트에 대한 변경된 정보를 전송하는 과정을 설명하는 도면이다.

도 35는 본 발명의 제4 실시 예에서 클라이언트의 자바스크립트 코드 복원 과정을 설명하는 도면이다.

도 36은 본 발명의 제4 실시 예에서 자바스크립트 코드가 변경되는 경우 동작을 설명하는 도면이다.

도 37은 본 발명의 제4 실시 예에서 자바스크립트에 대한 변경된 정보를 전송하는 과정을 설명하는 도면이다.

도 38은 본 발명의 제4 실시 예에서 클라이언트의 자바스크립트 코드 복원 과정을 설명하는 도면이다.

도 39는 본 발명의 실시 예에서 코어 자바스크립트를 미리 전송하는 과정을 설명하는 도면이다.

도 40은 본 발명의 실시 예에서 인덱스 파일과 코어 자바스크립트를 제외한 자바스크립트 코드를 전송하는 과정을 설명하는 도면이다.

발명을 실시하기 위한 구체적인 내용

- [0019] 이하, 첨부된 도면들을 참조하여 다양한 실시 예들을 상세히 설명한다. 이때, 첨부된 도면들에서 동일한 구성 요소는 가능한 동일한 부호로 나타내고 있음에 유의해야 한다. 또한 본 발명의 요지를 흐리게 할 수 있는 공지 기능 및 구성에 대한 상세한 설명은 생략할 것이다. 하기의 설명에서는 본 발명의 다양한 실시 예들에 따른 동작을 이해하는데 필요한 부분만이 설명되며, 그 이외 부분의 설명은 본 발명의 요지를 흐트리지 않도록 생략될 것이라는 것을 유의하여야 한다.
- [0020] 하기에 본 발명의 실시 예는 통신 네트워크의 예로 이동 통신 네트워크를 중심으로 설명한다. 다만, 본 발명의 권리 범위가 이동 통신 네트워크에 한정되는 것은 아니며, 유선 통신 네트워크에도 적용될 수 있다.
- [0021] 본 발명은 종래의 유무선 및 이동통신 네트워크에서 웹 서비스 성능을 향상시키는 기술로 크게 4 가지 실시 예로 분류된다.
- [0022] 제1 실시 예는 웹 페이지를 구성하는 리소스의 캐시 일관성(Cache Consistency)을 확인 하기 위해 발생하는 메시지를 감소시키고, 강한 캐시 일관성(Strong Cache Consistency)을 제공하는 기술이다.
- [0023] 제2 실시 예는 웹 서비스를 제공하기 위해 구축된 프록시 체인(Proxy Chain)에서, 인접하지 않은 프록시 서버들 간에 전송 제어 프로토콜(TCP, Transmission Control Protocol) 연결(connection)을 추가로 생성하고, 이 커넥션을 통해서 콘텐츠를 요청하고 수신함으로써, 프록시 체인을 우회하는 것을 제공하는 기술이다.
- [0024] 제3 실시 예는 프록시 서버가 단말의 무선 링크 타입, 단말의 하드웨어 및 소프트웨어 정보, 선호하는 JavaScript 코드 타입 정보를 이용해 JavaScript코드를 최적화하여 단말에게 전송하는 기술이다.
- [0025] 제4 실시 예는 웹 서버가JavaScript의 재사용성을 고려하여, 클라이언트가 요청한 JavaScript코드의 크기를 줄여 전송하는 기술이다.
- [0026] 하기에서는 각 실시 예를 구분하여 설명한다. 다만, 이는 각 실시 예에 대한 설명의 편의를 위한 것이지 실시 예의 권리 범위를 한정하는 것은 아니다. 즉, 각 실시 예는 개별적으로 실시 될 수 있을 뿐만 아니라, 서로 다른 실시 예의 조합으로 동작할 수도 있다.
- [0027] 먼저 제1 실시 예 관련 기술 및 문제점에 대해서 도 1 내지 도 4를 기반하여 설명한다. 웹 캐시를 제공하기 위해 캐시 일관성 정책(Cache Consistency Scheme)은 크게 Polling-every-time, Time-to-Live, Invalidation, Lease라는 정책이 있다.

- [0028] 도 1은 polling-every-time 기법에 기반하여 캐시 일관성을 제공하는 방법을 설명하는 도면이다.
- [0029] Polling-every-time 정책은 프록시 서버 또는 단말이 콘텐츠를 요청할 때 마다, 콘텐츠의 변경 유무를 근원 서버(Origin Server)로부터 확인하는 방법이다. HTTP 프로토콜에서는 요청 메시지의 If-Modified-Since 헤더와 If-Match 헤더를 이용해 단말 또는 프록시 서버가 근원 서버에게 해당 콘텐츠의 변경 유무를 질의한다. 근원 서버의 원본 콘텐츠가 변경되지 않았으면 “Not Modified” 메시지를 응답하며, 원본 콘텐츠가 변경되었다면 변경된 콘텐츠를 응답하게 된다.
- [0030] 도 1은 HTTP 요청 메시지의 If-Modified-Since 헤더와 HTTP 응답 메시지 중 “Not Modified” 메시지를 이용해 Polling-every-time을 나타낸 시퀀스 다이어그램의 예시이다. 도 1을 참조하면, 101 단계에서 프록시 서버(130, Proxy Server)는 단말(110, client, 본 발명의 실시 예에서는 클라이언트의 일 예로 단말을 중심으로 설명한다.)로부터 HTTP 요청 메시지를 수신할 수 있다. 103 단계에서 프록시 서버(130)는 해당 메시지에 If-Modified-Since 헤더를 추가하고 현재 시간을 명시하여 근원 서버(150, Origin Server)에게 전달한다. 105 단계에서 근원 서버(150)는 원본 콘텐츠가 변경되지 않은 경우(validation), “Not Modified” 라는 HTTP 응답 메시지를 프록시 서버(130)에게 전송한다. 107 단계에서 이를 수신한 프록시 서버(130)는 근원 서버(150)로부터 이전에 수신한 복사본 콘텐츠를 단말(110)에게 전송한다. 반면 원본 콘텐츠가 변경된 경우(update content), 113 단계에서 근원서버(150)는 변경된 원본 콘텐츠를 HTTP 응답 메시지에 담아서 프록시 서버(130)에게 전송한다. 117 단계에서 이를 수신한 프록시 서버(130)는 변경된 콘텐츠를 단말(110)에게도 전송하게 된다.
- [0031] 상기 Polling-every-time 정책은 콘텐츠를 요청할 때 마다, 근원 서버에게 변경 여부를 항상 확인하기 때문에 원본 콘텐츠와 복사본 콘텐츠가 엄격하게 일관성이 유지되는 강한 캐시 일관성(Strong Cache Consistency)을 제공한다. 하지만 원본 콘텐츠가 변경되지 않았을 경우에도 항상 변경 유무를 확인하는 메시지를 발생시키기 때문에, 변경 주기가 상대적으로 긴 콘텐츠의 경우, 콘텐츠의 변경 유무를 확인하는 트래픽이 네트워크에 발생하게 된다. 또 모바일 네트워크와 같이 지연이 큰 네트워크에서는 변경 여부를 확인하는 절차가 페이지 로딩 시간을 증가시키는 요인으로 작용하며, 점점 웹 페이지를 구성하는 리소스의 개수가 증가하고 있다는 점을 고려했을 때, 웹 서비스의 사용자 만족도를 급격하게 떨어뜨릴 수 있다.
- [0032] 도 2는 time-to-live 기법에 기반하여 캐시 일관성을 제공하는 방법을 설명하는 도면이다.
- [0033] Time-to-Live 정책은 단말 또는 프록시 서버가 근원 서버에게 콘텐츠를 요청하면, 근원 서버가 일정 시간 동안 웹 캐시를 통한 콘텐츠 전송을 허용하는 방식이다. 하지만 이는 웹 캐시를 허용할 뿐 캐시의 일관성을 보장하지는 않는다. HTTP 프로토콜에서는 요청 메시지의 If-Modified-Since를 이용해 근원 서버의 콘텐츠 변경 유무를 질의하고, 응답 메시지의 Expires나 Cache-Control 헤더를 이용해 웹 캐시로 제공되도록 허용 가능한 시간을 작성하여 응답한다.
- [0034] 도 2는 HTTP 요청 메시지의 If-Modified-Since 헤더와 HTTP 응답 메시지의 Expires 헤더를 이용해 Time-to-Live를 나타낸 시퀀스 다이어그램의 예시이다. 도 2는 초기 요청을 통해서 프록시 서버가 복사본 콘텐츠를 가지고 있고 Expires 헤더를 통해 웹 캐시의 만료 시간을 근원 서버로부터 제공 받은 상태에서 3 가지 상황에 대한 Time-to-Live의 동작 과정을 나타낸 것이다. 첫 번째는 캐시 만료 시간이 만료되지 않았을 경우, 두 번째는 캐시 만료 시간이 만료되고 원본이 변경되지 않은 경우, 세 번째는 캐시 만료 시간이 만료되고 원본이 변경된 경우에 대한 Time-to-Live의 동작 과정이다.
- [0035] 201 단계에서 프록시 서버(130)은 단말(110)로부터 HTTP 요청 메시지를 수신할 수 있다. 203 단계에서 프록시 서버(130)는 해당 메시지에 If-Modified-Since 헤더를 추가하여 근원 서버(150)에 전달한다. 205 단계에서 근원 서버(150)는 Expires 헤더를 통해 웹 캐시의 만료 시간 및 콘텐츠를 프록시 서버(130)에 제공할 수 있다. 207 단계에서 프록시 서버(130)는 근원서버(150)로부터 수신한 복사본 콘텐츠를 단말(110)에 전송할 수 있다.
- [0036] 캐시 만료 시간 이전의 경우, 211 단계에서 단말(110)이 프록시 서버(130)로 콘텐츠를 요청하게 되고, 이를 수신한 프록시 서버(130)는 근원 서버에게 변경 여부를 확인하는 요청 메시지를 보내지 않고 이전에 수신한 콘텐츠를 단말에게 전송한다(213 단계).
- [0037] 캐시 만료 시간 이후 원본이 변경되지 않은 경우에는, 221 단계에서 단말(110)은 프록시 서버(130)로 HTTP 요청 메시지를 전송한다. 223 단계에서 프록시 서버(130)는 콘텐츠의 변경 유무를 확인하기 위해 If-Modified-Since 헤더를 작성해 근원 서버(150)에게 전송한다. 원본 콘텐츠가 변경되지 않았기 때문에(validation), 225 단계에서 근원 서버(150)는 “Not Modified” 메시지와 새로운 캐시 만료 시간을 Expires 헤더에 명시하여 프록시 서버(130)에게 전송한다. 이를 수신한 프록시 서버(130)는 이전에 수신한 복사본 콘텐츠를 단말(110)에게 전송하

게 된다.

- [0038] 캐시 만료 시간 이후 원본이 변경된 경우, 231 단계에서 단말(110)은 프록시 서버(130)로 HTTP 요청 메시지를 전송한다. 233 단계에서 프록시 서버(130)는 콘텐츠의 변경 유무를 확인하기 위해 If-Modified-Since 헤더를 작성해 근원 서버(150)에게 전송한다. 콘텐츠가 변경되었기 때문에(update content), 235 단계에서 근원 서버(150)는 변경된 콘텐츠와 새로운 캐시 만료 시간을 Expires 헤더에 명시하여 프록시 서버에게 전송한다. 이를 수신한 프록시 서버(130)는 변경된 콘텐츠를 단말(110)에게 전송한다(237 단계).
- [0039] Time-to-Live는 캐시 만료 시간 동안 원본 콘텐츠의 변경 유무를 확인하지 않기 때문에 네트워크에 불필요한 트래픽을 감소시키고, 지연이 큰 모바일 네트워크에서 페이지 로딩 시간을 감소시키는 장점이 있다. 하지만 캐시 만료 시간 이전에 원본 콘텐츠가 변경될 경우, 프록시 서버에 위치한 복사본 콘텐츠와 근원 서버의 원본 콘텐츠가 일관성이 유지되지 않는 약한 캐시 일관성(Weak Cache Consistency)을 제공한다는 단점이 있다.
- [0040] 이는 자주 변경되고 시간에 민감한 콘텐츠를 공급하는데 사용할 경우 문제를 발생시킬 수 있다. 그래서 콘텐츠의 변경 주기와 일관성에 대한 민감한 정도에 따라 캐시 만료 시간을 적절하게 설정해야 한다. 캐시 만료 시간을 너무 길게 설정하면 응답 시간과 트래픽을 줄일 수 있지만 강한 캐시 일관성을 제공할 수 없게 되고, 캐시 만료 시간을 너무 짧게 설정하면 강한 캐시 일관성을 제공할 수 있지만 응답 시간이 길어지고 트래픽이 증가하게 된다.
- [0041] 도 3은 invalidation 기법에 기반하여 캐시 일관성을 제공하는 방법을 설명하는 도면이다.
- [0042] Invalidation정책은 근원 서버가 배포한 콘텐츠를 웹 캐시로 제공하는 것을 무기한으로 허용하고, 콘텐츠를 변경하기 전에 프록시 서버나 단말에게 콘텐츠를 변경할 것임을 알리는 방식이다. HTTP 프로토콜에서는 기본적인 요청 메시지와 응답 메시지를 이용해 콘텐츠를 요청하고 응답하며, Invalidation 메시지는 표준 헤더로 정의되어 있지 않기 때문에 임의의 헤더를 이용해 전송하거나 별도의 메시지를 따로 정의해 단말 또는 프록시 서버에게 전송한다.
- [0043] 도 3은 Invalidation의 캐시 일관성 제공 방법의 예시를 나타낸 시퀀스 다이어그램이다. 301 단계 내지 307 단계를 통해 단말(110)에 콘텐츠가 공급되고, 프록시 서버(130)에는 콘텐츠 복사본이 존재한다. 단말의 초기 요청으로 프록시 서버에 복사본 콘텐츠가 존재하는 상태에서, 근원 서버가 콘텐츠를 변경하지 않는 경우와 변경하려고 하는 경우에 대해서 Invalidation의 동작 과정을 나타낸 것이다.
- [0044] 근원 서버가 콘텐츠를 변경하지 않는 경우(validation), 311 단계에서 단말(110)이 프록시 서버(130)에게 콘텐츠를 요청하게 된다. 요청을 받은 프록시 서버(130)는 근원 서버(150)에 콘텐츠의 변경 유무를 확인하지 않고 복사본 콘텐츠를 단말에게 전송한다(313 단계).
- [0045] 근원 서버가 콘텐츠를 변경하려고 하는 경우(update content), 근원 서버(150)는 해당 콘텐츠를 요청했던 모든 프록시 서버(130)나 단말(110)에게 콘텐츠가 변경될 것임을 알리는 Invalidation 메시지를 전송한다(321 단계). 근원 서버(150)는 모든 프록시 서버나 단말이 Invalidation 메시지를 수신했다는 것을 확인하면 콘텐츠를 변경한다.
- [0046] 단말(110)이 변경된 콘텐츠를 프록시 서버에게 요청하고(323 단계), 프록시 서버(130)는 Invalidation 메시지를 수신했기 때문에 근원 서버(150)의 원본 콘텐츠가 변경되었음을 알 수 있다. 따라서 프록시 서버(130)는 복사본을 단말(110)에게 전송하는 것이 아니라, 근원 서버(150)에게 요청 메시지를 보낸다(325 단계). 프록시 서버(130)는 상기 요청 메시지에 기반하여 상기 근원 서버(150)로부터 변경된 콘텐츠를 수신한다(327 단계). 프록시 서버(130)는 수신된 콘텐츠를 다시 단말(110)에게 전송한다(329 단계).
- [0047] 이와 같이 Invalidation 정책은 근원 서버와 프록시 서버의 콘텐츠가 일관성을 엄격하게 유지시키는 강한 캐시 일관성을 제공하고, 콘텐츠의 변경 유무를 확인하기 위해 발생하는 트래픽을 감소시킨다. 하지만 근원 서버가 콘텐츠를 배포한 단말 또는 프록시 서버 모두를 기록하고 추적 및 관리해야 하는 부하가 발생하게 된다는 단점이 있다. 또 네트워크에 문제가 생겨서 Invalidation 메시지를 수신하지 못하는 프록시 서버나 단말이 발생하게 되면, 근원 서버에서 콘텐츠 변경이 무기한 연기되는 문제점이 있다. Invalidation 방식은 강한 캐시 일관성을 제공하고 네트워크 트래픽을 감소시킬 수 있는 장점보다, 서버의 부하를 과중 시키고 콘텐츠의 변경이 무기한 연기되는 단점이 크기 때문에 실제 웹 서비스에서 잘 쓰이지 않고 있다.
- [0048] 도 4는 lease 기법에 기반하여 캐시 일관성을 제공하는 방법을 설명하는 도면이다.
- [0049] Lease정책은 Time-to-Live 정책과 Invalidation정책이 결합된 방식으로, Time-to-Live정책과 같이, 콘텐츠의

변경 유무를 확인하는 절차 없이 웹 캐시로 제공 가능한 시간인 Lease를 발행한다. 하지만 Time-to-Live와는 다르게 Lease 시간 동안 근원 서버가 콘텐츠를 변경하지 않음으로써 캐시 일관성을 보장한다. 만약 근원 서버가 콘텐츠를 변경하려고 한다면, Lease 시간이 만료된 이후 콘텐츠를 변경한다. 또 HTTP 프로토콜에서는 Lease 헤더가 정의되어 있지 않기 때문에 별도 헤더를 작성하여 제공한다.

[0050] 도 4는 Lease정책의 캐시 일관성 제공 방법의 예시를 나타낸 시퀀스 다이어그램이다. 단말의 초기 요청으로 프록시 서버에 복사본 콘텐츠가 존재하는 상태에서, Lease 시간이 만료되지 않은 경우, Lease 시간이 만료되고 근원 서버의 원본 콘텐츠가 변경되지 않은 경우, Lease 시간이 만료되고 근원 서버의 원본 콘텐츠가 변경하려고 하는 경우에 대해서 Lease 방식의 동작 과정을 나타낸 것이다.

[0051] 401 단계 내지 407 단계는 도 2의 201 단계 내지 207 단계에 대응할 수 있다. 이때, 프록시 서버(130)는 근원 서버(150)로부터 도 2의 expire 정보대신 도 4에서는 lease 관련 정보를 수신할 수 있다.

[0052] 먼저, Lease 시간이 만료되지 않은 경우, 411 단계에서 단말(110)이 프록시 서버(130)에게 콘텐츠를 요청한다. 프록시 서버(130)는 근원 서버(150)로부터 발급 받은 Lease 시간이 만료되지 않았기 때문에, 근원 서버에게 콘텐츠 변경 유무를 확인하지 않고 자신이 가지고 있는 복사본 콘텐츠를 단말(110)에게 전송한다(413 단계).

[0053] 두 번째로, Lease 시간이 만료되고 근원 서버가 콘텐츠를 변경하지 않는 경우, 421 단계에서 프록시 서버(130)는 단말(110)로부터 콘텐츠 요청을 수신한다. 프록시 서버는 Lease 시간이 만료되었기 때문에 새로운 캐시 만료 시간을 발급 받기 위해 근원 서버로 요청 메시지를 전송한다(423 단계). 근원 서버(150)는 요청 메시지를 수신한 근원 서버는 콘텐츠를 변경하지 않았으므로(validation), 해당 콘텐츠를 캐시로 제공하는 것을 허용하는 Lease만을 프록시 서버(130)로 재발급한다(425 단계). 이를 수신한 프록시 서버(130)는 자신이 가지고 있는 복사본 콘텐츠를 단말(110)에게 전송한다(427 단계).

[0054] 세 번째로, Lease 시간이 만료되고 근원 서버(150)가 콘텐츠를 변경하려고 할 경우(update content), 근원 서버(150)는 자신이 발급한 캐시 만료 시간이 지났음을 확인하고 콘텐츠를 변경한다(update content). 431 단계에서 단말(110)은 프록시 서버(130)에게 요청 메시지를 보낸다. 이를 수신한 프록시 서버(130)는 근원 서버(150)로부터 발급 받은 Lease 시간이 만료되었기 때문에 새로운 캐시 만료 시간을 발급 받기 위해 근원 서버(150)로 요청 메시지를 전송한다(433 단계). 요청 메시지를 수신한 근원 서버(150)는 콘텐츠가 변경되었기 때문에, 변경된 콘텐츠와 새로운 Lease 시간을 프록시 서버에게 전송한다(435 단계). 프록시 서버(130)는 단말(110)에게 변경된 콘텐츠를 전송하고 Lease 시간 동안 해당 콘텐츠의 변경 유무를 확인하지 않고 웹 캐시를 제공한다(437 단계).

[0055] Lease 방식은 Time-to-Live정책과 같이, 캐시 만료 시간 동안 콘텐츠의 변경 유무를 확인하지 않기 때문에 트래픽을 감소시키고 응답 시간을 줄이는 효과를 얻을 수 있다. 또한 Invalidation정책과 같이, 강한 캐시 일관성을 제공할 수 있고, 콘텐츠를 발급한 모든 단말과 프록시 서버를 추적 및 관리해야 할 필요가 없기 때문에 Invalidation 방식보다 낮은 서버 부하를 발생시킨다. 게다가 네트워크에 문제가 생겨 Invalidation 메시지를 수신하지 못하는 단말이 발생하면 근원 서버의 콘텐츠 변경이 무기한 연기되는 문제를 개선한다. 하지만 Lease 시간을 길게 하면 콘텐츠를 변경하는데 대기해야 하는 시간이 길어지는 단점이 생기고, Lease 시간을 짧게 하면 콘텐츠의 변경 유무를 확인하는 과정이 자주 발생하기 때문에 트래픽이 증가하게 된다.

[0056] 다음으로 제2 실시 예와 관련 기술 및 문제점에 대하여 도 5에 기반하여 설명한다.

[0057] 웹 콘텐츠 시스템에서는 네트워크의 부하를 줄이고 웹 서비스의 응답 시간을 단축하기 위해서 웹 캐시를 사용할 수 있다. 이를 제공하는 위치는 서버, 클라이언트, 프록시 서버 등 다양한 곳에 위치할 수 있지만, 대체로 프록시 서버를 웹 캐시를 제공하는 매개체로 이용한다. 그 이유는, ISP(Internet Service Provider), NSP(Network Service Provider) 등 지리적으로 다양한 곳에 위치하여 네트워크에 발생하는 트래픽을 줄이고 분산시킬 수 있으며, 웹 캐시에 대한 다양한 접근 경로를 제공하기 때문이다. 프록시 서버는 계층적인 구조로 연결되어 있으며, 다수의 프록시 서버가 연결 설정을 이루고 있는 프록시 체인(Proxy Chain)을 형성하게 된다. HTTP 요청 메시지는 프록시 체인을 따라 서버에 도달하고, HTTP 응답 메시지는 HTTP 요청 메시지가 전달된 경로를 역으로 따라가 클라이언트에게 도달한다.

[0058] 도 5는 복수의 프록시 체인에 대한 통신 과정을 설명하는 도면이다. 도 5를 참조하면, 통신 시스템은 클라이언트(510), 프록시 서버 A(520), 프록시 서버 B(530), 프록시 서버 C(540) 및 서버(550)를 포함할 수 있다.

[0059] 도 5는 클라이언트(510)와 서버(550) 사이에 3개의 프록시 서버(520, 530, 540)가 프록시 체인을 형성한 상태에서, 클라이언트가 서버에게 웹 페이지를 요청하고 수신 받는 과정을 나타낸 것이다. 클라이언트(510)가 서버(550)에게 HTTP 요청 메시지를 이용해 콘텐츠를 요청하면, HTTP 요청 메시지는 프록시 서버A(520), 프록시 서버

B(530), 프록시 서버C(540)를 거쳐 서버(550)에게 도달하게 된다. HTTP 요청 메시지를 수신한 서버(550)는 콘텐츠를 HTTP 응답 메시지에 담아서 클라이언트(510)에게 전송한다. 서버(550)의 HTTP 응답 메시지는 프록시 서버 C(540), 프록시 서버B(530), 프록시 서버A(520)를 거쳐 클라이언트에게 전달된다.

[0060] 다음으로 제3 실시 예 관련 기술 및 문제점에 대하여 도 6을 참조하여 설명한다. 도 6은 HTTP 요청 메시지를 이용한 최적화 콘텐츠 제공 방법을 설명하는 도면이다.

[0061] 최적화 관련 기술은 서버가 단말의 화면의 크기, 단말의 운영체제 및 브라우저 등 단말의 정보를 이용해 최적화된 콘텐츠를 단말에게 전송하는 것이다. 클라이언트가 콘텐츠 요청 시, HTTP 요청 메시지의 User-Agent 헤더에 단말의 운영체제와 브라우저 정보를 작성하고, UA-Disp 헤더에 단말의 화면 크기 정보를 작성하여 서버에게 전송한다. 이를 수신한 서버는 단말의 운영체제 및 브라우저 정보와 화면 크기 정보를 이용해 이미지를 축소해서 보내는 등 최적화된 콘텐츠를 제공한다.

[0062] 도 6은 단말(610)이 HTTP 요청 메시지의 User-Agent 헤더를 작성하여 서버에게 콘텐츠를 요청할 때, 서버(630)가 HTTP 응답 메시지를 이용해 콘텐츠를 프록시 서버(620)로 전송하면, 프록시 서버가 단말에 최적화된 콘텐츠를 제공하는 방법의 예시를 시퀀스 다이어그램으로 나타낸 것이다.

[0063] 클라이언트(610)가 모바일 단말일 경우, HTTP 요청 메시지의 User-Agent 헤더에 단말의 웹 브라우저와 운영체제 정보를 작성하여 콘텐츠를 요청한다(601 단계). User-Agent 헤더는 Mozilla/5.0 브라우저를 사용하고, Linux 운영체제를 사용하는 Android 플랫폼임을 나타낸다. 클라이언트(610)가 요청한 콘텐츠는 picture.jpg이다. HTTP 요청 메시지는 프록시 서버(620)를 거쳐 서버(630)에 도달하게 된다(603 단계). 서버(630)는 picture.jpg 파일을 담은 HTTP 응답 메시지는 프록시 서버(620)로 전송한다(605 단계). 프록시 서버(620)는 이전에 수신한 HTTP 요청 메시지의 User-Agent 헤더 정보를 통해서 클라이언트(610)가 모바일 단말임을 알 수 있다. 프록시 서버(620)는 수신된 picture.jpg 파일을 크기를 축소하고, 기존의 picture.jpg 대신 크기가 축소된 picture.jpg 파일을 HTTP 응답 메시지에 담아서 클라이언트(610)에게 전송함으로써 동작 과정을 마치게 된다(607 단계). 이런 콘텐츠 최적화 기술들은 단말의 정보만을 이용해 콘텐츠를 최적화하기 때문에, 다양한 콘텐츠 최적화를 제공하는데 한계가 있고, 통신 및 네트워크 정보를 활용하여 웹 성능을 향상시키는 효과를 제공하지는 못한다.

[0064] 다음으로 제4 실시 예 관련 기술 및 문제점에 대하여 설명한다.

[0065] 자바스크립트(JavaScript)는 클라이언트와 서버 간에 상호교환적인 통신을 가능하게 하며, 사용자에게 동적인 웹 페이지 및 GUI(Graphic User Interface)를 제공한다. JavaScript는 웹 서비스를 제공하는데 핵심적인 역할을 하는 요소 중 하나이다. 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. 하지만 JavaScript는 페이지 로딩 시간을 늘어나게 하는 주요 원인 중 하나이다. 그 이유는, 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.

[0066] 도 7은 JavaScript 2개, 이미지 1개, 스타일시트 1개, Iframe 1개로 구성되어 있는 웹 페이지를 요청할 때의 다운로드 순서를 나타낸 타임라인이다. 먼저 웹 페이지를 요청하고, 수신이 완료되면 웹 페이지를 구성하는 리소스들을 차례대로 요청을 실시한다. 리소스의 배치 순서는 JavaScript, 이미지, 스타일시트, Iframe 순이다. 첫 번째 JavaScript가 요청이 되고, 이 JavaScript가 수신 완료되고 실행이 종료될 때까지 어떤 리소스도 요청하지 않는다. JavaScript 실행이 완료되면, 다음 리소스인 두 번째 JavaScript를 요청한다. 이 리소스 또한 JavaScript이기 때문에 수신이 완료되고 실행이 완료될 때까지 다른 리소스를 요청하지 않는다. 두 번째 JavaScript 실행이 완료되면, 다음 리소스들을 요청하는데, 이후에 위치한 리소스들은 JavaScript가 아니기 때문에 병렬로 요청하고 수신하게 된다. 이와 같이 웹 브라우저들이 JavaScript에 대해서 블록킹 현상이 발생하도록 구현한 이유는 JavaScript에 독립적인 환경을 제공함으로써 안정적으로 실행되는 것을 보장하기 위해서다.

[0067] 도 8은 JavaScript를 병렬을 허용해서, 먼저 수신된 JavaScript를 실행할 경우, 문제를 발생시킬 수 있는 예제 코드이다. example1.html 파일은 a.js와 b.js로 구성되어 있다. example.html을 먼저 수신한 브라우저는 a.js를 먼저 요청하여 수신하고 실행이 완료되면 b.js를 요청하고 수신하여 실행해야 한다. 만약 a.js와 b.js를 동시에 요청하면, 네트워크 상태나 JavaScript 크기에 따라, b.js가 먼저 수신될 수 있다. b.js가 먼저 수신된 경

우, a.js보다 먼저 실행되고 “Good Bye.” 라는 메시지가 “Hello.” 라는 메시지보다 먼저 출력되기 때문에 웹 페이지의 무결성이 유지되지 않는 문제가 발생할 수 있다.

[0068] 이처럼 JavaScript는 동적인 웹 페이지와 서버 클라이언트 간 상호교환적인 서비스를 제공할 수 있다는 장점이 있지만, 블록킹 현상을 발생시키기 때문에 페이지 로딩 시간을 증가시키는 단점이 있다.

[0069] 전송되는 JavaScript의 크기를 줄이면, JavaScript가 전송되는 시간이 줄어들기 때문에, 블록킹 현상이 발생하는 시간이 감소하여 페이지 로딩 시간이 감소하는 효과를 얻을 수 있다. JavaScript의 크기를 줄이는 기술로, Google의 Closure Compiler가 있다.

[0070] Closure Compiler는 기존의 JavaScript 소스 코드를 크기가 작은 JavaScript 소스 코드로 바꿔준다. Closure Compiler가 JavaScript 소스 코드의 크기를 줄이는 방법은, 변수의 길이를 줄이고, 공백 문자와 불필요한 코드들을 제거함으로써 JavaScript 소스 코드의 크기를 줄인다. 바뀐 JavaScript 소스 코드는 기존의 JavaScript 소스 코드와 같은 동작을 수행한다.

[0071] 예를 들어, Closure Compiler가 수행되지 않은 자바스크립트에 Closure Compiler를 적용한 결과에 대하여 설명한다.

[0072] [closure compiler 적용 전]

표 1

[0073]

```
function hello(name) {
    // greets the user
    Alert( 'Hello,' + name);
}
Hello( 'New user' );
```

[0074] [closure compiler 적용 후]

표 2

[0075]

```
function hello(name){alert( "Hello," + name)}hello( "New user" );
```

[0076] JavaScript 코드에 Closure Compile을 수행하면, 띄어쓰기, 줄 바꿈, 주석 등을 제거하여 상기와 같이 사이즈가 줄어든 JavaScript 코드가 생성될 수 있다.

[0077] 도 9는 본 발명의 일 실시 예에 따른 적용되는 시스템을 설명하는 도면이다. 도 9에서는 이동 통신 시스템의 구성을 설명한다. 하지만 본 발명이 적용되는 통신 시스템은 이동 통신 시스템에 한정할 것이 아니라, 유선망을 이용한 통신 시스템에서도 적용될 수 있다.

[0078] 도 9를 참조하면, 클라이언트 장치(910)인 단말 및 단말과 무선 통신을 수행하는 기지국을 연결하는 RAN(920, Radio Access), 코어 네트워크(EPC, Evolved Packet Core, 930), 무선 랜(Wireless LAN, 940), 인터넷 망(950) 및 웹 서버(960)로 구성될 수 있다.

[0079] 도 10은 본 발명의 일 실시 예에 따른 단말의 구성을 설명하는 도면이다. 본 발명의 실시 예에서 단말은 무선 통신 링크만 사용하는 무선 단말일 수도 있고, 유선 통신 링크 또는 무선 통신 링크를 사용하는 단말일 수도 있다. 한편, 상기 단말의 구성을 반드시 도 10의 구성에 한정하는 것은 아니다. 예를 들어, 단말은 단말의 전반적인 동작을 제어하는 제어부, 단말의 외부 네트워크 노드와 통신을 수행하는 통신부로 구성될 수 있다.

[0080] 도 11은 본 발명의 일 실시 예에 따른 웹 서버를 설명하는 도면이다. 한편, 상기 웹 서버의 구성을 반드시 도 11의 구성에 한정하는 것은 아니다. 예를 들어, 웹 서버는 서버의 전반적인 동작을 제어하는 제어부, 서버가 외부 네트워크 노드와 통신을 수행하는 통신부로 구성될 수 있다.

[0081] 도 12는 본 발명의 일 실시 예에 따른 프록시 서버를 설명하는 도면이다. 본 발명의 실시 예에서 프록시 서버는 RAN(Radio Access Network), EPC(Evolved Packet Core), Internet 등에 위치할 수 있다. 또한, 다수의 프록시

서버를 배치하는 것이 가능하다. 프록시 서버가 RAN에 배치되는 경우 이동 통신 기지국으로 활용 가능하도록 HSDPA 모뎀을 내장하고 있을 수 있다. 나아가 프록시 서버는 LTE 모뎀 등 다양한 통신 링크를 추가하는 것이 가능하다.

[0082] 한편, 상기 프록시 서버의 구성을 반드시 도 12의 구성에 한정하는 것은 아니다. 예를 들어, 프록시 서버는 프록시 서버의 전반적인 동작을 제어하는 제어부, 프록시 서버의 외부 네트워크 노드와 통신을 수행하는 통신부로 구성될 수 있다.

[0083] 본 발명의 실시 예에 따르면 상기 프록시 서버의 제어부는 적어도 하나의 클라이언트(client) 장치로부터 콘텐츠 요청을 수신하고, 상기 제2 서버로부터 데이터를 직접 수신하기 위한 우회 연결을 설정하며, 상기 우회 연결을 이용하여 상기 제2 서버로부터 직접 상기 콘텐츠에 대한 데이터를 수신하고, 상기 데이터를 상기 클라이언트 장치로 전송하도록 제어할 수 있다. 이때, 우회 연결은 상기 연결 서버를 경유하지 않고 상기 제2 서버로부터 직접 데이터를 수신할 수 있는 바이패스(by-pass) 전송 제어 프로토콜(TCP, Transmission Control Protocol)을 설정하는 것을 지칭한다. 또한, 상기 제1 서버, 제2 서버 및 연결 서버는 프록시 서버일 수 있다.

[0084] 또한, 상기 프록시 서버의 제어부는 상기 제1 서버의 IP 주소 또는 TCP 포트 번호 중 적어도 하나의 정보에 대응하는 프록시 간 우회 연결 파라미터를 생성하고, 상기 생성된 파라미터를 상기 연결 서버를 경유하여 상기 제2 서버로 전송하며, 상기 제2 서버의 바이패스 TCP 연결 요청에 기반하여 우회 연결을 설정하도록 제어할 수 있다.

[0085] 또한, 상기 프록시 서버의 제어부는 상기 제1 서버와 상기 제2 서버 사이에 상기 연결 서버를 경유하지 않고, 상기 데이터를 수신하도록 제어할 수 있다. 또한, 상기 제어부는 상기 우회 연결을 통해 수신한 데이터에 대한 콘텐츠 프리픽스(content prefix), 상기 제2 서버의 IP 주소와 TCP 포트 번호를 포함하는 프록시 간 정보를 업데이트하고, 상기 프록시 간 정보에 기반하여, 적어도 하나의 클라이언트로부터 수신하는 콘텐츠 요청에 대해 상기 설정된 우회연결을 이용할지 여부를 결정하도록 제어할 수 있다.

[0086] 또한, 상기 제어부는 상기 제2 서버로부터 인증 요청 메시지를 수신하고, 상기 상기 제2 서버로 인증 허가 메시지를 전송하며, 상기 제2서버로부터 접속 허가 메시지를 수신하여 상기 제2 서버와 상기 데이터 수신을 위한 인증을 수행하도록 제어할 수 있다. 또한, 상기 제어부는 상기 우회 연결에 대한 유지시간을 설정하도록 제어할 수 있다. 이때, 상기 제1 서버와 상기 제2 서버의 우회 연결은 상기 유지시간 동안 지속될 수 있다.

[0087] 또한, 상기 제어부는 상기 제2 서버와 푸시(PUSH)를 설정하고, 상기 푸시를 설정하면, 상기 제2 서버의 콘텐츠가 변경될 때 상기 우회 연결을 이용하여 변경된 데이터를 수신하도록 제어할 수 있다.

[0088] 상기 도 10 내지 도 12에서 본 발명의 실시 예에 따른 엔티티(entity)에 대하여 설명하였다. 하지만 각 엔티티의 기능 및 동작은 이에 한정되지 않고, 도 13 내지 도 40의 동작을 수행하기 위한 각 엔티티의 기능을 제어할 수 있다.

[0089] 본 발명에서는 웹 성능을 향상시키기 위해서 크게 5 가지 기능을 수행하는데, 첫 번째는 웹 서버가 웹 페이지를 구성하는 리소스 마다 고유한 ID를 생성하고, 리소스 ID를 웹 페이지에 삽입하여 클라이언트에게 전송하면, 이를 수신한 클라이언트가 웹 페이지의 리소스 ID를 이용해 웹 페이지를 구성하는 리소스의 변경 유무를 확인하는 것이다.

[0090] 두 번째는, 다수의 프록시 서버로 구성된 프록시 체인에서, 인접하지 않은 프록시 서버들 간에 TCP 커넥션을 추가로 생성하고, 생성된 TCP 커넥션을 통해서 콘텐츠를 요청하고 수신하는 것을 가능하게 함으로써, 프록시 체인을 우회하는 것을 가능하게 하는 것이다.

[0091] 세 번째는, 프록시 서버가 단말의 무선 링크 타입, 단말의 하드웨어 및 소프트웨어 정보, 선호하는 JavaScript 코드 타입 정보를 이용해 JavaScript 코드를 최적화하여 단말에게 전송하는 것이다.

[0092] 네 번째는, 웹 서버에 있는 JavaScript 코드가 수정되면, 수정되기 전 JavaScript 코드를 기준으로, 수정 후 JavaScript 코드에서 추가/변경/삭제된 함수 정보와 변경되거나 추가된 JavaScript 함수 코드를 클라이언트에게 전송하는 것이다.

[0093] 다섯 번째 방안은, 자주 쓰일 것이라고 예상되거나 자주 쓰이는 JavaScript 코드를 Core JavaScript로 선정하여 단말에게 미리 전송하고, 단말이 JavaScript 코드를 요청하면, 프록시 서버가 Core JavaScript를 제외한

JavaScript 코드와 제외된 코드에 대한 인덱스 정보를 클라이언트에게 전송하는 것이다.

[0094] 첫 번째 기능은 도 10의 Check Coded DOM Client와 도 11의 Check Coded DOM Server에 의해서 수행될 수 있다. Check Coded DOM Server는 웹 페이지를 구성하는 리소스마다 고유의 리소스 ID를 생성하고, 리소스 ID를 웹 페이지에 삽입하여 무선 단말로 전송하는 기능을 수행한다. Check Coded DOM Client는 리소스 ID가 삽입된 웹 페이지를 수신하면, 리소스 ID를 추출하여 이전에 수신한 웹 페이지를 구성하는 리소스 ID와 비교하여 변경된 리소스를 찾아내고, 변경된 리소스에 대한 요청을 수행한다. 단말은 리소스 ID를 관리하기 위해 도 13과 같은 Previous Resource URL & ID TABLE과 Current Resource URL & ID TABLE을 가지고 있다. 구체적인 동작 과정은 다음 절에서 서술한다.

[0095] 두 번째 기능은 도 12의 By-pass Connection Manager에 의해 수행될 수 있다. 다수의 프록시 서버가 프록시 체인을 형성했을 때, 프록시 체인을 우회할 수 있는 By-pass TCP 커넥션을 생성하고, 단말이 요청한 콘텐츠를 우회 전송하는 것을 제어한다. 단말로부터 콘텐츠를 요청하는 HTTP 요청 메시지를 수신하면, 프록시 서버가 By-pass TCP 커넥션을 이용해 콘텐츠를 요청하기 위해서 도 14와 같은 Proxy-to-Proxy Forwarding Info TABLE을 가지고 있다. 이 테이블은 By-pass TCP 커넥션을 통해서 수신한 콘텐츠의 프리픽스와 By-pass TCP 커넥션의 IP 주소와 TCP 포트 번호를 매핑하여 저장한다. 구체적인 동작 과정은 다음절에서 서술한다.

[0096] 세 번째 기능은 도 10의 JavaScript Optimization Client와 도 12의 JavaScript Optimization Server에 의해서 수행된다. 본 기능은 조금 더 세부적으로는 4 가지 기능이 있다. 첫 번째는 단말의 통신 링크 정보와 단말이 선호하는 JavaScript 코드의 유형을 고려하여, 프록시 서버가 단말이 요청한 JavaScript 코드를 바이트 코드 또는 네이티브 코드로 변환하여 단말에게 전송하는 것이다. 두 번째는 단말의 브라우저가 지원하지 않는 스크립트 코드를 요청할 경우, 프록시 서버가 스크립트 코드를 단말의 브라우저가 지원하는 스크립트 언어의 코드로 변환하여 단말에게 전송하는 것이다. 이 때 프록시 서버가 도 15와 같은 Supported Language Info TABLE을 가지고 있다. 이 테이블은 웹 브라우저 별로 지원 가능한 스크립트 언어 정보를 담고 있다. 세 번째는 단말이 하위 버전의 브라우저를 사용하는데, 하위 버전의 브라우저에서 실행이 불가능한 JavaScript 코드를 요청할 경우, 프록시 서버가 스크립트 코드를 단말의 하위 브라우저에서 실행 가능하도록 JavaScript 라이브러리를 추가하여 단말에게 전송하는 것이다. 프록시 서버는 도 16과 같은 Supported JavaScript Library Info TABLE을 가지고 있다. 이 테이블은 브라우저 버전 마다 지원하는 JavaScript 라이브러리 정보를 담고 있어, 프록시 서버가 JavaScript 코드에 JavaScript 라이브러리를 삽입할 때 참조된다. 네 번째는 프록시 서버가 단말이 요청한 JavaScript 코드를 바이트 코드 또는 네이티브 코드로 변환되어 전송할 때, 추가적으로 수행되는 과정으로, 바이트 코드 또는 네이티브 코드를 분할하여 단말에게 전송하는 것이다.

[0097] 이와 같이 4 가지 기능 제공하기 위해서는, 도 10의 JavaScript Optimization Client가 HTTP 요청 메시지를 전송할 때, Preferred-content-processing-type, Preferred-content-processing-info, Preferred-content-processing-link 헤더를 작성하여 보내야 하고, 도 12의 JavaScript Optimization Server는 도 10의 JavaScript Optimization Client로부터 수신한 헤더 정보들과 테이블 정보를 활용하여 JavaScript 코드의 최적화를 수행한다. 구체적인 동작 과정은 다음절에서 서술한다.

[0098] 네 번째 기능은 도 10의 Functional JavaScript Client와 도 11의 Functional JavaScript Server에 의해서 수행된다. 도 11의 Functional JavaScript Server는 JavaScript 코드가 생성되거나 변경되면 도 17의 Function & ID TABLE과 도 18의 Script & ID TABLE을 생성한다. 도 10의 Functional JavaScript Client가 도 11의 Functional JavaScript Server에 JavaScript 코드를 요청하면, 도 19의 Manipulation Info TABLE을 생성하고, Manipulation Info TABLE 엔트리 중 Manipulation 속성 값이 “New” 이거나 “Modified” 인 함수 코드, Script & ID TABLE, Manipulation Info TABLE을 단말에게 전송한다. 이를 수신한 도 10의 Functional JavaScript Client는 이전에 수신한 JavaScript 코드와 현재 수신한 정보를 이용해 JavaScript 코드를 복원한다. 구체적인 동작 과정은 다음절에서 서술한다.

[0099] 다섯 번째 기능은 도 10의 Core JavaScript Client와 도 11의 Core JavaScript Server에 의해서 수행된다. 도 11의 Core JavaScript Server는 자주 쓰일 것이라고 예상되거나 자주 쓰이는 JavaScript 코드를 Core JavaScript로 추출한다. 단말이 OFF 또는 SLEEP 상태에서 ON으로 전환되면 도 10의 Core JavaScript Client가 Core JavaScript를 요청하여 수신한다. 단말이 JavaScript를 요청하면, 도 11의 Core JavaScript Server가 해당 JavaScript에서 Core JavaScript를 제거한 코드 부분과 제거된 코드에 대한 인덱스 정보를 단말에게 전송한다. 이를 수신한 도 10의 Core JavaScript Client는 Core JavaScript, Core JavaScript를 제거한 코드 부분, 제거된 코드에 대한 인덱스 정보를 이용해 JavaScript 코드를 복원한다. 구체적인 동작 과정은 다음절에서 서술

한다.

- [0100] 먼저 제1 실시 예에 대하여 설명한다. 제1 실시 예는 종래의 캐시 일관성 정책이 가지는 trade-off 문제를 해결하기 위한 것으로, 웹 페이지를 구성하는 리소스에 대해서 강한 캐시 일관성을 제공하고, 캐시 일관성을 확인하기 위해 전송되는 메시지를 줄임으로써, 네트워크의 부하를 낮추고 페이지 로딩 시간을 감소시킬 수 있는 방법 및 장치를 제공한다.
- [0101] 도 20은 본 발명의 제1 실시 예에 따른 초기 웹 페이지를 생성함으로써 발생하는 과정들을 설명하는 도면이다. 도 20을 참조하면, 메인 서버(2010)는 메인 웹 페이지를 생성한다. 본 발명의 실시 예에서는 메인 서버가 생성한 웹 페이지(2015)에 포함되는 리소스는 html 문서와 이미지 파일인 것으로 가정한다. 이때, 생성된 웹 페이지에 대한 html 문서(2035)는 서버 1(2030)에 위치하고, 이미지 파일(2055)은 서버 2(2050)에 위치한 것으로 가정한다.
- [0102] 서버 1(2030)과 서버 2(2050)에서는 새로운 리소스가 생성되었기 때문에 리소스 내용을 기반으로 체크코드를 생성한다. 이때, SHA-1 알고리즘을 이용해 체크코드를 생성할 수 있다. 예를 들어, html 문서의 경우, 문서에 작성된 텍스트 내용을 SHA-1 알고리즘의 입력 값으로 사용할 수 있다. 또한, 이미지의 경우, 이미지 데이터를 표현하는 바이너리 데이터를 SHA-1 알고리즘의 입력 값으로 사용하여 체크코드를 생성할 수 있다.
- [0103] 그 다음 과정으로 메인 서버(2010)는 웹 페이지를 구성하는 리소스들이 체크코드를 가지고 있는지 확인을 한다. 메인 서버(2010)에서 방금 생성된 웹 페이지의 리소스들은 체크코드를 가지고 있지 않다. 도 20의 index.html(2015)에는 현재 체크 코드가 포함되어 있지 않다. 따라서 메인 서버(2010)는 서버 1 및 서버 2에 리소스 ID를 요청한다(2016 단계, 2018 단계). 메인 서버는 각 리소스의 src 속성에 작성되어 있는 URL을 이용해 리소스 ID를 요청할 수 있다. 서버 1(2030)과 서버 2(2050)는 해당 리소스에 대한 체크 코드를 응답한다(2017 단계, 2019 단계). 서버 1(2030)과 서버 2(2050)는 해당 리소스에 대해서 20바이트의 체크코드를 응답할 수 있다. 모든 리소스의 체크코드를 응답 받은 메인 서버(2010)는, 각 리소스에 해당하는 DOM 객체에 id 속성을 추가하고, id 속성 값에 체크코드를 입력한다.
- [0104] 도 21은 본 발명의 제1 실시 예에 따른 클라이언트의 웹 페이지 수신 과정을 설명하는 도면이다. 도 20과 같이 메인 서버에서 초기 웹 페이지 생성 과정을 마치면, 도 21의 과정과 같이 클라이언트(2101)는 메인 웹 페이지와 웹 페이지를 구성하는 리소스들을 요청하고, 이에 대응하여 웹 페이지와 웹 페이지를 구성하는 리소스들을 수신한다.
- [0105] 클라이언트(2000)는 HTTP 요청 메시지를 메인 서버(2010)으로 전송한다(2101 단계). 클라이언트(2000)는 메인 서버(2010)로부터 상기 HTTP 요청 메시지에 대응하는 체크코드가 포함된 메인 웹 페이지를 수신한다. 클라이언트(2000)는 웹 페이지의 Resource URL과 Resource ID 관련 테이블을 관리할 수 있다. 상기 관련 테이블은 현재 URL 및 ID에 대한 테이블과, 이전 URL 및 ID에 대한 테이블 관련 정보(2060)를 관리할 수 있다.
- [0106] 클라이언트(2000) 상기 수신한 웹 페이지를 구성하는 리소스들의 URL과 ID를 Current Resource URL & ID TABLE에 저장한다. 그 다음 클라이언트(2000)는 Previous Resource URL & ID 와 Current Resource URL & ID TABLE의 엔트리(entry) 중에서 일치하지 것이 있는지 확인한다. 현재 초기 요청이기 때문에 Previous Resource URL & ID TABLE에는 엔트리가 존재하지 않게 되므로, 클라이언트(2000)는 두 테이블 간에 일치하는 엔트리가 존재하지 않는 것으로 판단할 수 있다. 이를 통해서 클라이언트(2000)는 자신이 웹 페이지를 구성하는 리소스의 복사본을 가지고 있지 않음을 알 수 있다. 클라이언트는 웹 페이지를 구성하는 각 리소스들을 요청하고 이에 대한 응답을 수신할 수 있다. 클라이언트는 상기에서 수신한 메인 웹 페이지의 정보에 기반하여, 웹 페이지를 구성하는 리소스를 저장하고 있는 서버로 HTTP 요청 메시지를 전달할 수 있다.
- [0107] 클라이언트(2000)는 웹 페이지의 html 관련 리소스를 저장하고 있는 서버 1(2030)으로 HTTP 요청 메시지를 전송하고(2105 단계), 웹 페이지의 이미지 관련 리소스를 저장하고 있는 서버 2(2050)으로 HTTP 요청 메시지를 전송할 수 있다(2107 단계). 클라이언트(2000)는 서버 1(2030)로부터 html에 대한 정보를 포함하는 응답을 수신하고(2106 단계), 서버 2(2050)로부터 이미지에 대한 정보를 포함하는 응답을 수신할 수 있다(2108 단계).
- [0108] 수신이 완료되면 Resource URL & ID Table을 업데이트 시킬 수 있다. 예를 들어, 테이블(2161)과 같이 Current Resource URL & ID TABLE의 엔트리는 Previous Resource URL & ID TABLE로 이동시키고, Current Resource URL & ID TABLE의 엔트리를 모두 삭제할 수 있다.

- [0109] S2는 리소스가 변경되지 않은 상태에서 클라이언트(2000)가 동일한 웹 페이지를 재요청하는 과정을 나타낸다. 즉, 도 21의 S1은 초기 웹 페이지 수신 과정을 설명하는 것이고, S2는 초기 웹 페이지 수신 이후 동일한 웹 페이지를 구성하는 리소스가 변경되지 않은 경우 웹 페이지 수신 과정을 설명한다.
- [0110] 클라이언트(2000)는 메인 서버(2010)로 웹 페이지 요청을 전송한다(2111 단계). 클라이언트(2000)는 메인 서버(2010)로부터 웹 페이지(2021)를 수신한다(2113 단계). 클라이언트(2000)는 웹 페이지(2021)를 구성하는 리소스의 URL과 ID를 추출하여 Current Resource URL & ID TABLE의 엔트리로 추가한다(2162).
- [0111] 클라이언트(2000)는 Previous Resource URL & ID TABLE과 Current Resource URL & ID TABLE의 엔트리를 비교한다. 클라이언트(2000)는 Current Resource URL & ID TABLE에는 존재하지만 Previous Resource URL & ID TABLE에는 존재하지 않거나, 두 테이블 모두에 존재하면서 Resource URL은 같지만 Resource ID가 다른 리소스를 찾아낸다.
- [0112] 도 21의 실시 예에서는 웹 페이지의 리소스가 변경되지 않았기 때문에, 두 테이블에 존재하는 엔트리들은 Resource URL과 Resource ID가 일치하게 된다. 이를 통해서 클라이언트(2000)는 리소스가 변경되지 않았음을 알 수 있으므로, 리소스의 변경 유무를 확인하지 않고 이전에 수신한 복사본을 재사용한다. 따라서 클라이언트(2000)는 서버 1(2030) 또는 서버 2(2050)로 HTTP 요청 메시지를 전송하지 않을 수 있다.
- [0113] 도 22는 본 발명의 제1 실시 예에 따른 리소스 변경 시 웹 페이지 수정 과정을 설명하는 도면이다. 도 22 및 도 23의 실시 예는 도 21과 동일한 웹 페이지에서 서버 1의 리소스만 변경된 경우를 가정한다. 서버1(2030)의 리소스가 변경되면(2036), 서버 1은 SHA-1 알고리즘을 이용해 새로운 체크코드를 생성한다. 체크코드가 변경되면, 서버1(2030)은 메인 서버(2010)로 해당 리소스 URL과 ID를 전송한다(2201 단계). 변경된 체크코드를 수신한 메인 서버(2010)는 웹 페이지의 체크코드를 수정한다(2022).
- [0114] 도 23은 본 발명의 제1 실시 예에 따른 클라이언트의 웹 페이지 수신 과정을 설명하는 도면이다. 도 22에서 웹 페이지의 체크코드 변경이 완료 후, 클라이언트(2000)는 메인 서버(2010)로 도 21과 동일한 웹 페이지를 요청한다(2301 단계). 웹 페이지는 도 22에서 설명한 바와 같이 업데이트 되었으므로, 클라이언트(2000)는 체크코드가 변경된 웹 페이지(2022)를 메인 서버(2010)로부터 수신한다(2303 단계).
- [0115] 클라이언트(2000)는 웹 페이지에서 리소스들의 URL과 ID를 추출하여 Current Resource URL & ID TABLE에 엔트리를 추가시킨다. 클라이언트는 이전 웹 페이지를 수신했을 때와 동일하게 Previous Resource URL & ID TABLE과 Current Resource URL & ID TABLE의 엔트리를 비교한다. 클라이언트는 Current Resource URL & ID TABLE에는 존재하지만 Previous Resource URL & ID TABLE에는 존재하지 않거나, 두 테이블 모두에 존재하지만 Resource URL은 같지만 Resource ID가 다른 리소스를 찾아낸다.
- [0116] 도 23의 실시 예에서는 서버 1(2030)의 html 관련 리소스만 변경되었으므로 두 테이블을 비교한 결과, 서버 1(2030)의 리소스 URL은 같지만 리소스 ID가 변경된 것을 알 수 있다. 이것은 서버 1의 리소스가 변경되었다는 것을 의미한다. 서버 2의 리소스는 리소스 ID가 동일하므로 변경되지 않았다. 클라이언트(2000)는 서버 1(2010)로 웹 페이지를 구성하는 리소스 중에서 서버 1의 리소스를 요청하는 http 요청 메시지를 전송한다(2305 단계). 클라이언트(2000)는 서버 1(2030)로부터 변경된 서버 1의 리소스를 수신한다(2307 단계).
- [0117] 클라이언트(2000)는 서버 1(2030)로부터 리소스 수신이 완료되면, 도면부호 2164와 같이 Current Resource URL & ID TABLE에 있는 엔트리를 Previous Resource URL & ID TABLE로 이동시키고, Current Resource URL & ID TABLE의 엔트리를 모두 삭제할 수 있다.
- [0118] 다음으로 제2 실시 예에 대하여 설명한다. 제2 실시 예는 종래의 프록시 체인에서, 다수의 프록시 서버를 거쳐 HTTP 메시지가 전송됨에 따라 전송 지연이 발생하는 문제를 해결하고, 웹 캐시를 제공하는 특정 프록시 서버의 부하가 집중되는 문제를 해결한다. 다수의 프록시 서버로 구성된 프록시 체인에서, 인접하지 않은 프록시 서버들 간에 TCP 커넥션을 추가로 생성하고, 이 TCP 커넥션을 통해서 콘텐츠를 요청하고 수신하는 것을 가능하게 함으로써, 프록시 체인을 우회하는 것을 가능하게 한다. 더 나아가 클라이언트에 인접한 프록시 서버가, 클라이언트로부터 HTTP 요청 메시지를 수신하면, 다른 프록시 서버들로 균등하게 리다이렉트(redirect)를 수행함으로써, 프록시 서버의 부하를 분산시키는 것을 제공한다. 하기에서 제2 실시 예에 대한 설명에서 프록시 A를 제1 서버, 프록시 E를 제2 서버로 명명할 수도 있다. 또한, 프록시 A와 프록시 E는 서로에 대해서 바이패스 TCP 커넥션을 위한 목적 서버로 명명될 수 있다. 프록시 간 우회 연결은 각 프록시를 운영하는 사업자가 동일한 경우, 프록시 사업자가 서로 다른 경우에도 서비스 사업자간 우회 연결을 협의한 경우 등에 적용될 수 있다.

- [0119] 도 24는 본 발명의 제2 실시 예에 따른 바이패스 TCP 커넥션을 이용한 웹 페이지 수신 방법을 설명하는 도면이다. 도 24를 참조하면, 먼저 프록시 체인이 구축된 네트워크에서, 클라이언트(2401)가 웹 페이지를 요청하면서 바이 패스(By-pass) TCP 커넥션을 설정하는 과정을 설명한다. 제2 실시 예에서는 프록시 A(2402)와 프록시 E(2406)가 바이패스 커넥션을 설정을 하고자 하는 경우를 가정하고 설명한다.
- [0120] 클라이언트(2401)는 HTTP 요청 메시지를 프록시 A(2402)에게 전송한다(2401 단계). 프록시 A(2402)는 클라이언트(2401)가 보낸 HTTP 응답 메시지에 Via 헤더를 생성하고, By-pass TCP 커넥션 설정에 사용될 자신의 IP 주소와 TCP 포트 번호를 작성한다. 또한, 프록시 A(2402)가 By-pass TCP 커넥션을 생성할 것임을 알리기 위해, “proxy-to-proxy=ON”이라는 메시지를 추가한다. 만약 By-pass TCP 커넥션을 생성할 의사가 없으면 “proxy-to-proxy=ON” 메시지를 추가 하지 않는다. 프록시 A(2402)에 의해서 수정된 HTTP 요청 메시지는 프록시 B(2403)에게 전송한다(2412 단계).
- [0121] 이를 수신한 프록시 B(2403)는, Via 헤더에 자신의 IP 주소와 TCP 포트 번호를 추가한다. 프록시 B(2403)는 By-pass TCP 커넥션을 생성할 의사가 없기 때문에(제2 실시 예에서의 가정) 자신의 IP 주소와 TCP 포트 번호에 대해서 “proxy-to-proxy=ON” 메시지를 추가하지 않는다. 프록시 B(2403)에 의해서 수정된 HTTP 요청 메시지는 프록시 C(2404)에게 전송된다(2413 단계). 즉, 프록시 A에 대해서만 “proxy-to-proty=ON”이 생성되어 있는 상태이다.
- [0122] 이와 같은 과정은 프록시 C(2404) 및 프록시 D(2405)에서 계속해서 반복된다(2413, 2414 단계). HTTP 요청 메시지는 마지막 프록시 서버인, 프록시 E(2406)까지 도달하게 된다. 프록시 E(2406)는 HTTP 요청 메시지의 Via 헤더 값 중에서 “proxy-to-proxy=ON”이라는 메시지를 포함하고 있는 IP 주소와 TCP 포트 번호로 By-pass TCP 커넥션 설정을 수행한다(2417 단계). 즉, 프록시 A(2402)의 IP 주소와 TCP 포트 번호에 대해서 “proxy-to-proxy=ON” 메시지를 포함하고 있기 때문에, 프록시 E(2406)는 프록시 A(2402)와 By-pass TCP 커넥션을 생성한다. 이때, 프록시 E(2406)와 프록시 A(2402)의 바이패스 TCP 커넥션은 프록시 E(2406)의 직접 연결 요청 및 프록시 A의 응답으로 생성될 수 있다. 프록시 E(2406)는 By-pass TCP 커넥션을 생성하고,, HTTP 요청 메시지를 웹 서버(2407 단계)로 전송한다(2416 단계).
- [0123] 2418 단계에서 프록시 E(2406)는 웹 서버(2407)로부터 HTTP 응답 메시지를 수신한다. 프록시 E(2406)는 인접한 프록시 서버인 프록시 D(2405)로 HTTP 응답 메시지를 전달하는 것이 아니라, 프록시 A(2402)와 연결된 By-pass TCP 커넥션을 통해서 HTTP 응답 메시지를 전달한다(2419 단계). 프록시 A(2402)는 프록시 E(2406)로부터 수신한 HTTP 응답 메시지를 클라이언트(2401)로 전달할 수 있다(2420 단계).
- [0124] 프록시 A(2402)는 Proxy-to-Proxy Forwarding Info TABLE을 가지고 있다. Proxy-to-Proxy Forwarding Info TABLE은 By-pass TCP 커넥션을 통해서 수신한 콘텐츠에 대한 콘텐츠 프리픽스(Content Prefix), 콘텐츠를 전달 받은 프록시 서버의 IP 주소와 TCP 포트 번호를 저장한다. 이 테이블은 클라이언트(2401)로부터 콘텐츠를 요청하는 HTTP 요청 메시지를 수신할 경우, 프록시 체인을 통해서 HTTP 요청 메시지를 전달할 것인지, By-pass TCP 커넥션 설정을 통해서 HTTP 요청 메시지를 보낼지 결정하는데 사용된다. 프록시 A(2402)는 클라이언트(2401)로부터 수신한 HTTP 요청 메시지에 대한 웹 콘텐츠를 바이 패스 TCP 커넥션이 설정된 프록시를 경유해 수신해야 하는 경우, 바이패스 TCP 커넥션이 설정된 프록시로 HTTP 요청 메시지를 전송할 수 있다.
- [0125] 하기 단계에서는 바이패스 TCP 커넥션을 이용하는 경우의 HTTP 요청 및 응답 과정을 설명한다. 하기에서는 시퀀스 모드(sequence mode)와 병렬 모드(parallel mode)에 대해 설명한다.
- [0126] 먼저 By-pass TCP 커넥션을 이용한 콘텐츠 요청 방법으로, 시퀀스 모드에 대한 구체적인 동작 과정이다. 클라이언트(2401)는 콘텐츠를 요청하기 위해 HTTP 요청 메시지를 프록시 A(2402)로 전송한다(2421 단계).
- [0127] 프록시 A는 시퀀스 모드로 동작하고 있으므로, Proxy-to-Proxy Forwarding Info TABLE을 탐색하여, 클라이언트가 요청한 콘텐츠 프리픽스에 대한 엔트리가 존재하는지 확인한다. Proxy-to-Proxy Forwarding Info TABLE에 존재한다면, 콘텐츠 프리픽스에 해당하는 By-pass TCP 커넥션으로 HTTP 요청 메시지를 전달하여 프록시 체인을 우회한다. Proxy-to-Proxy Forwarding Info TABLE에 존재하지 않으면, 이웃한 프록시 서버인 프록시 B로 전달한다.
- [0128] 도 24의 실시 예는 현재(2421 단계에서 클라이언트로부터 HTTP 요청 메시지를 수신하는 순간) Proxy-to-Proxy Forwarding Info TABLE에 클라이언트가 요청한 콘텐츠에 대한 프리픽스가 존재하지 않은 상황에 대한 도면이다. 따라서 프록시 A(2402)는 HTTP 요청 메시지를 프록시 B(2403)로 전달한다(2422 단계). 또한 클라이언트가 요청한 콘텐츠가 프록시 E가 가지고 있다고 가정한 것이기에, 프록시 체인을 거쳐 HTTP 요청 메시지가 프록시 E에

도달한다(2423, 2424, 2425 단계). 현재 프록시 A(2402)와 프록시 E(2406)은 바이 패스 TCP 커넥션 상태이다(2417 과정을 통하여). 따라서 프록시 E(2406)는 콘텐츠를 담은 HTTP 응답 메시지를 설정된 By-pass TCP 커넥션을 통해서 프록시 A(2402)에게 전송할 수 있다.

[0129] 이를 수신한 프록시 A는 수신 받은 콘텐츠에 대한 프리픽스와 콘텐츠를 전달 받은 프록시 서버의 IP 주소와 TCP 포트 번호를 Proxy-to-Proxy Forwarding Info TABLE에 추가한다(B1). Proxy-to-Proxy Forwarding Info TABLE에 추가하는 작업을 수행하는 동시에, 프록시 A(2402)는 프록시 E(2406)로부터 수신한 콘텐츠를 담은 HTTP 응답 메시지를 클라이언트(2401)에게 전송한다(2429 단계).

[0130] Proxy-to-Proxy Forwarding Info TABLE에 클라이언트(2401)가 2411 단계 및 2421 단계에서 요청한 콘텐츠에 대한 정보가 저장되어 있다(B1). 따라서 이후 클라이언트(2401)가 2411 단계 및 2421 단계에서 요청한 콘텐츠와 동일한 콘텐츠를 요청하면, 프록시 A(2402)는 프록시 E와의 바이 패스 TCP 커넥션을 통해 HTTP 요청 메시지를 전송하고, 대응하는 응답 메시지를 수신할 수 있다.

[0131] By-pass TCP 커넥션을 이용한 콘텐츠 요청 방법으로, 병렬 모드에 대한 구체적인 동작은 다음과 같다. 클라이언트(2401)는 콘텐츠를 요청하기 위해 HTTP 요청 메시지를 프록시 A(2402)로 전송한다(2431 단계).

[0132] 프록시 A(2402)는 병렬 모드로 동작하고 있음으로, Proxy-to-Proxy Forwarding Info TABLE를 탐색하지 않고, 이웃한 프록시 서버인 프록시 B와 By-pass TCP 커넥션이 설정되어 있는 프록시 E에 HTTP 요청 메시지를 동시에 전송한다(2433, 2434 단계). 도 24는 프록시 B(2403)와 프록시 E(2406)에 클라이언트(2401)가 요청한 콘텐츠가 있다고 가정한 상황에서의 동작 과정을 나타낸 것이다. 프록시 A(2402)로부터 HTTP 요청 메시지를 받은 프록시 B(2403)와 프록시 E(2406)는, 콘텐츠를 담은 HTTP 응답 메시지를 프록시 A로 전송한다(2435, 2436 단계). 이를 수신한 프록시 A(2402)는 Proxy-to-Proxy Forwarding Info TABLE을 업데이트하는 동시에, 콘텐츠를 담은 HTTP 응답 메시지를 클라이언트(2401)에게 전송한다(2437 단계). 프록시 A(2402)는 프록시 B(2403) 또는 프록시 E(2406) 중 먼저 수신한 메시지를, 클라이언트(2401)로 전송할 수 있다.

[0133] 도 25는 본 발명의 제 2 실시 예에 따른 프록시 서버 간 인증 방법을 설명하는 도면이다. 도 25를 참조하여, By-pass TCP 커넥션을 설정한 프록시 서버 간에 콘텐츠 접근을 위한 인증 방법의 구체적인 동작 과정을 설명한다. 인증 과정은 By-pass TCP 커넥션을 생성(2517 단계)한 이후에 선택적으로 수행될 수 있다. 2511 단계 ~ 2517 단계는 도 24의 2411 단계 ~ 2417 단계에 대응한다.

[0134] 프록시 E(2506)는 콘텐츠 접근에 대한 인증을 요구하는 인증(Authentication) 메시지를 프록시 A(2502)에게 전송한다(2521 단계). 인증(Authentication) 메시지는 Proxt-to-Proxy Authentication 헤더를 가지고 있고, 인증을 요구하는 프록시 서버의 이름을 작성하여 전송된다. 인증(Authentication) 메시지를 수신한 프록시 A(2502)는, 콘텐츠 접근에 허가 받기 위해서, 허가(Authorization) 메시지의 Proxy-to-Proxy Authorization 헤더에 패스워드를 작성하여 프록시 E(2506)에게 전송한다(2523 단계).

[0135] 허가(Authorization) 메시지를 수신한 프록시 E(2506)는, 프록시 A(2502)가 전송한 패스워드가 올바른 패스워드인지 확인한다. 패스워드가 일치하면, "Permit accessing contents" 메시지를 프록시 A(2502)에게 전송함으로써, By-pass TCP 커넥션을 통해서 프록시 E(2506)가 가지고 있는 콘텐츠에 접근 가능하다는 것을 알린다(2531 단계). 패스워드가 일치하지 않으면, "Refuse accessing contents" 메시지를 프록시 A(2502)에게 전송함으로써, By-pass TCP 커넥션을 통해서 프록시 E(2506)가 가지고 있는 콘텐츠에 접근하는 것이 불가능하다는 것을 알린다(2533 단계).

[0136] 도 26은 본 발명의 제2 실시 예에 따른 프록시 서버 간 연결 시간 설정 방법을 설명하는 도면이다. 도 26을 참조하여, 프록시 체인을 우회하기 위해 설정한 By-pass TCP 커넥션의 유지시간을 협상하는 방법의 구체적인 동작 과정을 설명한다. 이 과정은 By-pass TCP 커넥션을 생성(2617 단계)한 이후에 선택적으로 수행된다. 2611 단계 ~ 2617 단계는 도 24의 2411 단계 ~ 2417 단계에 대응한다.

[0137] 프록시 E(2606)는 By-pass TCP 커넥션의 유지시간을 협상을 요청하기 위해 세션 협상 요청(Session Negotiation Request) 메시지를 프록시 A(2602)로 전송한다(2621 단계). 이를 수신한 프록시 A(2602)는, 지속적으로 By-pass TCP 커넥션을 유지하려면 세션 협상 응답(Session Negotiation Response) 메시지의 Session-Timeout 헤더에 "permanent"로 작성하여 프록시 E로 전송한다(2623 단계). 프록시 E(2606)는 By-pass TCP 커넥션을 종료하지 않고 지속적으로 유지한다(2625 단계).

[0138] 프록시 A가 특정 시간 동안만 바이 패스 TCP 연결을 유지하는 경우는 다음과 같다. 프록시 E는 2621 단계에서 설명한 바와 같이, 세션 협상 요청 메시지를 프록시 A로 전송할 수 있다(2631 단계). 프록시 A(2602)는 특정 시

간 동안만 By-pass TCP 커넥션을 유지 하려면, 세션 협상 응답(Session Negotiation Response) 메시지의 Session-Timeout 헤더에 By-pass TCP 커넥션 유지시간을 작성하여 프록시 E(2606)로 전송한다(2633 단계). 도 26의 실시 예에서는 100 초 동안 By-pass TCP 커넥션을 유지하기를 요구하는 것으로 설정되었다. 프록시 A가 Session-Timeout 헤더를 “100 Second” 로 작성하여 Session Negotiation Response 메시지를 프록시 E로 전송한다(2633 단계). 프록시 E(2606)는 세션 협상 응답(Session Negotiation Response) 메시지를 수신한 시간으로부터 100 초 뒤 프록시 A(2602)와 By-pass TCP 커넥션을 해제한다. 한편, 상기 유지시간은 프록시 E가 세션 협상 요청 메시지를 설정할 때, 요구할 수도 있을 것이다.

[0139] 도 27은 본 발명의 제2 실시 예에 따른 프록시 서버 간 콘텐츠 푸시 설정 방법을 설명하는 도면이다. 도 27을 참조하여, By-pass TCP 커넥션을 설정한 프록시 서버 간에 콘텐츠 푸시(PUSH)를 협상하는 방법의 구체적인 동작 과정을 설명한다. 이 과정은 By-pass TCP 커넥션을 생성(2717 단계)한 이후에 선택적으로 수행된다. 2711 단계 ~ 2717 단계는 도 24의 2411 단계 ~ 2417 단계에 대응한다.

[0140] 프록시 E(2706)는, 프록시 A(2702)가 콘텐츠 푸시를 받을 지에 대한 여부를 질의하기 위해, Proactive Content Push Negotiation Request 메시지를 프록시 A(2702)에게 전송한다(2721 단계). 이를 수신한 프록시 A(2702)는 Proactive Content Push Negotiation Response 메시지의 Content-Push 헤더에 “Proactive” 라고 작성해서 프록시 E(2706)에게 전송한다(2723 단계).

[0141] 프록시 E(2706)는 Content-Push 헤더 값이 “Proactive” 인 Proactive Content Push Negotiation Response 메시지를 수신했기 때문에, 자신의 콘텐츠가 변경될 때마다 By-pass TCP 커넥션을 통해서 프록시 A에게 콘텐츠 푸시를 한다. 제 3의 단말 또는 프록시 서버가 프록시 E에게 콘텐츠를 요청하고 수신하는 과정을 통해서, 프록시 E의 콘텐츠가 변경되었다면, 프록시 A에게 변경된 콘텐츠를 푸시한다(2729 단계).

[0142] 만약 클라이언트가 Content-Push 헤더를 “Reactive” 로 작성하여 Proactive Content Push Negotiation Response 메시지를 프록시 E에게 전송하면(2733 단계), 제 3의 단말 또는 프록시 서버가 프록시 E(2706)에게 콘텐츠를 요청하고 수신하는 과정을 통해서, 프록시 E의 콘텐츠가 변경된다고 할지라도, 프록시 E(2706)는 프록시 A(2702)에게 콘텐츠 푸시를 하지 않는다.

[0143] 한편, 상기 도 25, 도 26 및 도 27에서 인증, 연결 유지 시간 설정, 푸시 설정은 프록시 E에서 트리거(trigger)되는 것으로 설명하였으나, 반드시 이에 한정하는 것은 아니며, 프록시 A로부터 각 설정 요청이 트리거 될 수도 있다. 프록시 E가 수행할 수 있는 기능을 프록시 A가 수행할 수 있음은 자명하다.

[0144] 다음으로 제3 실시 예에 대하여 설명한다. 제3 실시 예는 통신 및 네트워크 상태를 고려한 자바스크립트(JavaScript) 코드 최적화를 제공하지 못한다는 문제를 해결하기 위한 것으로, 단말의 통신 링크 타입에 따라, 프록시 서버 서버가 자바스크립트(JavaScript) 코드를, 바이트 코드(Byte code) 또는 네이티브 코드(Native code)로 변환하여 클라이언트에게 전송하는 것을 제공한다.

[0145] 또 서로 다른 웹 브라우저가 지원하는 스크립트 언어가 다를 경우, 프록시 서버가 클라이언트의 웹 브라우저에서 실행 가능한 스크립트 언어로 변환하여 클라이언트에게 전송하는 것을 제공하며, 웹 브라우저 버전에 따라 지원하는 스크립트 라이브러리가 다를 경우, 프록시 서버가 하위 버전의 웹 브라우저에서 실행 가능한 스크립트 코드로 변환하여 클라이언트에게 전송하는 것을 제공한다.

[0146] 도 28 및 도 29는 클라이언트의 통신 링크에 따라, 프록시 서버가 자바스크립트(JavaScript) 코드를 바이트 코드 또는 네이티브 코드로 변환하여 전송하는 방법의 구체적인 동작 과정을 설명하는 도면이다. 도 28을 참조하면, 클라이언트(2801)가 3G 모바일 네트워크를 이용해 자바스크립트(JavaScript) 코드를 요청하고, 프록시 A(2602)로부터 바이트 코드나 네이티브 코드가 아닌 자바스크립트(JavaScript) 소스 코드 그대로 받아오게 되는 과정을 설명한다.

[0147] 클라이언트는 HTTP 요청 메시지를 이용해 자바스크립트(JavaScript) 코드를 요청한다(2811 단계). 클라이언트(2801)는 그 HTTP 요청 메시지에 Preferred-content-processing-type, Preferred-content-processing-info, Preferred-content-processing-link 헤더를 추가로 작성하여, 프록시 A(2802)에게 전송한다.

[0148] 프록시 A(2802)는 시퀀스 모드로 자바스크립트(JavaScript)를 요청하고 수신한다(2813 단계 및 2815 단계). 프록시 A(2802)는 수신된 자바스크립트(JavaScript) 코드는 Preferred-content-processing-type, Preferred-content-processing-info, Preferred-content-processing-link 헤더 정보를 이용해, 어떤 코드의 형태로 변환

하여 클라이언트에게 전송할지 결정한다(2817 단계).

- [0149] Preferred-content-processing-link 헤더 값이 WLAN-only일 경우, 클라이언트가 사용하는 통신 링크가 WLAN일 때만 JavaScript 코드를 변환한다는 것을 나타낸다. 도 28에서 클라이언트는 3G 모바일 네트워크를 이용하기 때문에 자바스크립트(Javascript) 코드를 바이트 코드나 네이티브 코드로 변환하지 않고 자바스크립트(Javascript) 소스 코드 그대로 클라이언트에게 전송한다(2819 단계).
- [0150] 만약 도 28에서 클라이언트(2801)의 통신 링크가 WLAN을 사용할 경우 동작은 다음과 같다. 클라이언트(2802)가 자바스크립트(Javascript) 코드를 요청하는 HTTP 요청 메시지를 프록시 A(2802)에게 전송한다. 프록시 A(2802)는 시퀀스 모드를 이용해 자바스크립트(Javascript) 코드를 요청하고 수신한다.
- [0151] 자바스크립트(Javascript) 수신을 완료한 프록시 A는 클라이언트로부터 수신한 HTTP 요청 메시지의 Preferred-content-processing-link 헤더 값이 WLAN-only이기 때문에 Preferred-content-processing-type, Preferred-content-processing-info 헤더의 정보를 이용해 JavaScript 코드를 바이트 코드 또는 네이티브 코드로 변환하는 과정을 수행한다. Preferred-content-processing-type 헤더 값이 바이트 코드(Byte-code)이기 때문에 자바스크립트(Javascript) 코드를 바이트 코드로 변환한다. 또한, 클라이언트에서 실행 가능하게 하기 위해서 Preferred-content-processing-info 헤더 정보를 이용해 클라이언트에서 실행 가능한 바이트 코드로 변환하여 클라이언트에게 전송한다.
- [0152] 도 29는 클라이언트가 WLAN을 통해서, 자바스크립트(Javascript) 코드를 네이티브 코드로 변환하여 수신하기를 원하는 경우의 동작 과정을 설명하는 도면이다. 클라이언트(2901)는 3G 모바일 네트워크를 통해서 프록시 A(2902)에게 HTTP 요청 메시지를 보낸다(2911 단계). 이를 수신한 프록시 A(2902)는 병렬 모드로 JavaScript 코드를 요청하고(2913, 2917 단계) 수신한다(2915, 2919 단계). 병렬 모드는 일 실시 예일 뿐 HTTP 요청은 다양한 방법으로 이루어 질 수 있다. 클라이언트(2901)가 보낸 HTTP 요청 메시지의 Preferred-content-processing-link 헤더 값이 WLAN-only이고, 현재 클라이언트(2901)는 3G를 통해 프록시 A와 접속하고 있다. 따라서 프록시 A(2902)는 자바스크립트(Javascript) 코드를 바이트 코드 또는 네이티브 코드로 변환하지 않는다(2921 단계). 프록시 A(2902)는 자바스크립트(Javascript) 소스 코드를 클라이언트(2901)에게 전송한다(2923 단계).
- [0153] 만약 클라이언트(2901)가 사용하는 네트워크가 3G 모바일 네트워크가 아닌, WLAN을 사용하는 경우의 동작은 다음과 같다. 동일하게, 클라이언트(2901)는 HTTP 요청 메시지를 이용해 자바스크립트(Javascript)를 프록시 A(2902)에게 요청한다. 프록시 A(2902)는 자바스크립트(Javascript)를 요청하고 수신한다. 클라이언트(2901)로부터 수신한 HTTP 요청 메시지의 Preferred-content-processing-link 헤더 값이 WLAN-only이고, 단말 또한 WLAN을 사용하고 있다. 따라서 프록시 A(2902)는 JavaScript 코드를 Preferred-content-processing-type 헤더에 명시되어 있는 네이티브 코드로 변환하여 클라이언트에게 전송한다. JavaScript 코드를 네이티브 코드로 변환 시, Preferred-content-processing-info 헤더 정보를 이용해 클라이언트에서 실행 가능한 네이티브 코드로 변환한다.
- [0154] 도 30은 본 발명의 제3 실시 예에 따른 스크립트 언어를 변환하여 전송하는 방법을 설명하는 도면이다. 도 30을 참조하여, 클라이언트가 웹 브라우저가 지원하지 않는 스크립트 코드를 요청하면, 프록시 서버가 단말이 요청한 스크립트 코드를 요청하여 수신한 후, 단말이 지원하는 스크립트 코드로 변환하여 클라이언트에게 전송하는 과정을 설명한다.
- [0155] 클라이언트(3001)가 Preferred-content-processing-type, Preferred-content-processing-info, Preferred-content-processing-link 헤더를 작성하고, 브라우저가 지원하지 않는 스크립트 코드를 요청한다(3011 단계). 프록시 A(3002)는 클라이언트가 요청한 스크립트 코드를 요청한다(3013 단계). 프록시 A(3002)는 상기 스크립트 코드 요청에 대응하여 스크립트 코드를 수신한다(3015 단계).
- [0156] 프록시 A(3002)는 Support Language Info TABLE(도면 15)을 참조하여 클라이언트가 지원하는 스크립트 코드를 확인한다. Support Language Info TABLE은 브라우저 마다 지원하는 스크립트 언어 정보를 저장해 놓은 테이블이다. 프록시 A(3002)는 클라이언트가 지원하는 스크립트 코드로 변환한다(3017 단계). 프록시 A는 클라이언트가 사용하는 통신 링크와 Preferred-content-processing-link 헤더에 작성된 통신 링크를 비교한다. 클라이언트의 통신 링크가 WLAN이고, Preferred-content-processing-link 헤더에 작성된 통신 링크가 WLAN-only임으로, 프록시 A(3002)는 변환된 스크립트 코드를 Preferred-content-processing-type에 작성된 코드 타입인 바이트 코드로 변환하여 클라이언트(3001)에게 전송한다(3019 단계).
- [0157] 도 30의 실시 예에서 클라이언트가 선호하는 코드의 타입이 네이티브 코드인 경우, 클라이언트의 웹 브라우저가

지원하지 않는 스크립트 코드를, 웹 브라우저가 지원하는 스크립트 코드로 변환하고, 변환된 스크립트 코드를 네이티브 코드로 변환하여 클라이언트에게 전송할 수 있다.

- [0158] 도 31은 본 발명의 제3 실시 예에서 브라우저 버전에 따라 스크립트 코드를 변환하여 전송하는 방법을 설명하는 도면이다. 도31을 참조하여, 클라이언트가 하위 버전의 웹 브라우저를 사용하고, 상위 버전의 웹 브라우저에서만 실행 가능한 스크립트 코드를 요청할 경우, 프록시 서버가 상위 버전의 웹 브라우저에서만 실행 가능한 스크립트 코드를, 하위 버전의 웹 브라우저에서도 실행 가능하도록 변환하여 전송하는 과정을 설명한다.
- [0159] 클라이언트(3101)가 Preferred-content-processing-type, Preferred-content-processing-info, Preferred-content-processing-link 헤더를 작성하고, 브라우저가 지원하는 언어로 작성되었지만 하위 브라우저에서 실행할 수 없는 스크립트 코드를 요청한다(3111 단계). 프록시 A(3102)는 스크립트 코드를 프록시 B(3103)에 요청한다(3113 단계). 프록시 A(3102)는 상기 요청에 대응하여 프록시 B(3103)로부터 스크립트 코드를 수신한다(3115 단계). 프록시 A(3102)는 Supported JavaScript Library Info TABLE(도 16)의 정보와 Preferred-content-processing-info 정보를 확인하여 클라이언트에서 실행 가능한지 판단한다.
- [0160] 클라이언트의 웹 브라우저에서 실행 불가능한 스크립트 코드인 경우, 프록시 A는 하위 브라우저에서 실행 가능하도록 하위 브라우저가 지원하지 않는 프레임워크나 API를 추가한다(3117 단계). 프록시 A(3102)는 추가적으로 클라이언트의 통신 링크와 Preferred-content-processing-link 헤더 정보를 비교하여 추가 변환 여부를 결정한다. 클라이언트의 통신 링크가 WLAN이고, Preferred-content-processing-link 헤더에 작성된 통신 링크가 WLAN-only임으로, 프록시 A(3102)는 변환된 스크립트 코드를 Preferred-content-processing-type에 작성된 코드 타입인 바이트 코드로 변환하여 클라이언트에게 전송한다(3119 단계).
- [0161] 만약 도 31의 실시 예에서 클라이언트가 선호하는 코드의 타입이 네이티브 코드이면, 프록시 A(3102)는 클라이언트의 웹 브라우저에서 실행 불가능한 스크립트 코드에, 하위 브라우저에서 실행 가능하도록 하위 브라우저가 지원하지 않는 프레임워크나 API를 추가하고, 네이티브 코드로 변환하여 클라이언트에게 전송할 수 있다.
- [0162] 도 32는 본 발명의 제3 실시 예에서 통신 링크에 따라 파일을 분할하여 전송하는 방법을 설명하는 도면이다. 이는 클라이언트의 통신 링크에 따라, 프록시 서버가 자바스크립트(JavaScript) 코드를 바이트 코드 또는 네이티브 코드로 변환하여 전송하는 방법에, 변환된 코드를 분할하여 전송하는 과정이 추가된 것이다. 도 32와 도 28의 차이점은 Preferred-content-processing-type 헤더 값이 Byte-code streaming으로 작성되어 있다는 것이다.
- [0163] Preferred-content-processing-type 헤더 값이 Byte-code streaming으로 작성된 경우, 프록시 A(3202)는 자바스크립트(JavaScript) 코드를 바이트 코드로 변환한다(3217 단계). 프록시 A(3202)는 변환된 바이트 코드를 분할하여 클라이언트에게 전송한다(3219, 3221 단계).
- [0164] 만약 도 32의 실시 예에서 클라이언트가 선호하는 코드의 타입이 네이티브 코드이면, Preferred-content-processing-type 헤더 값이 Native-code streaming으로 작성되었기 때문에, 프록시 A(3202)는 자바스크립트(JavaScript) 소스 코드를 네이티브 코드로 변환하고, 변환된 네이티브 코드를 분할하여 클라이언트에게 전송할 수 있다.
- [0165] 다음으로 제4 실시 예에 대하여 설명한다. 제4 실시 예는 공백 문자와 불필요한 코드를 제거하여 전송되는 JavaScript 코드의 크기를 줄이는데 한계가 있다는 문제를 해결하기 위한 것으로, JavaScript 코드의 재사용성을 고려함으로써 전송되는 JavaScript 코드의 크기를 줄이는 것이다.
- [0166] 자주 쓰이는 JavaScript 코드를 클라이언트에게 미리 전송하고, 클라이언트가 JavaScript 코드를 요청하면, 미리 전송된 JavaScript 코드를 제외하고 남은 코드 부분과 제외된 코드 부분에 대한 인덱스 정보를 전송함으로써, 전송되는 JavaScript의 크기를 줄인다.
- [0167] 또 JavaScript 코드가 유지보수 과정을 거치면서 수정된 경우, 이전 JavaScript 코드를 기준으로 현재 JavaScript 코드에서 변경된 부분을 함수 단위로 클라이언트에게 전송함으로써, 전송되는 JavaScript의 크기를 줄이는 것을 제공한다.
- [0168] 도 33은 본 발명의 제4 실시 예에서 서버가 자바스크립트(JavaScript)를 생성할 때 처리 과정을 설명하는 도면이다. 서버에서 자바스크립트(JavaScript)가 생성되면, 자바스크립트(JavaScript) 코드를 함수 단위로 분할한다. 서버(3303)는 분할된 함수 코드를 입력 값으로 사용하여 SHA-1 알고리즘을 이용해 20 byte의 Function ID를 생성한다(3311 단계). Function & ID TABLE에 함수 이름과 그에 해당하는 Function ID를 삽입하

여 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.

[0169] 도 34는 Function & ID TABLE과 Script & ID TABLE이 완성된 상태에서, 클라이언트가 해당 JavaScript를 요청하고 서버가 함수 단위 코드, 함수 변경 정보, 버전 정보를 보낸 과정을 설명하는 도면이다.

[0170] 도 34를 참조하면, 클라이언트(3401)는 A.js가 아닌 A.ajs 파일을 요청한다(3411 단계). ajs 파일은 archive of JavaScript의 약자로, 본 발명의 실시 예에서 새롭게 정의한 확장자이다. 확장자 'ajs'는 함수 단위 코드, 함수 변경 정보, 버전 정보를 Gzip을 이용해 압축한 파일이다. 또 HTTP 요청 메시지에 Script version이라는 헤더를 새로 정의한다. 이는 클라이언트가 가지고 있는 해당 JavaScript의 버전을 나타낸다.

[0171] 초기 요청이기 때문에 Script version의 헤더 값을 N/A로 설정한다. 이를 수신한 서버(3403)는 추가로 Manipulation Info TABLE을 생성한다. 이 테이블은 클라이언트가 가지고 있는 자바스크립트(Javascript)와 서버가 가지고 있는 자바스크립트(Javascript)를 비교하여 함수 변경 정보를 나타내며, Function name과 Manipulation 속성으로 구성되어 있다. 함수 이름(Function name) 속성은 자바스크립트(Javascript) 코드에 존재하는 함수 이름을 나타낸다.

[0172] 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, 서버(3403)는 A.ajs 파일을 HTTP 응답 메시지에 담아서 클라이언트(3401)에게 전송한다(3415 단계).

[0173] 도 35는 클라이언트가 서버로부터 A.ajs 파일을 수신하여 자바스크립트(Javascript) 코드를 복원하는 과정을 설명하는 도면이다. 3511 ~ 3517 단계는 도 33 및 도 34에서 설명한 동작에 대응한다. 클라이언트(3501)은 수신된 A.ajs 파일을 Gzip를 이용해 압축해제 한다(3519 단계).

[0174] 클라이언트(3501)는 Manipulation Info TABLE을 이용해 함수 단위로 수신된 코드를 조합하여 하나의 자바스크립트(Javascript) 파일로 만듦으로써 자바스크립트(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.

[0175] 도 36은 본 발명의 제4 실시 예에서 자바스크립트 코드가 변경되는 경우 동작을 설명하는 도면이다. 서버(3603)는 자바스크립트(Javascript) 코드가 변경되면 도 33과 마찬가지로 함수 단위로 분할한다. 분할된 함수는 코드를 기반으로 SHA-1 알고리즘을 수행해 Function ID를 생성한다. 함수 이름과 Function ID를 Function & ID TABLE에 삽입하여 테이블을 완성한다. 그 다음 Function ID들을 SHA-1 알고리즘의 입력 값으로 사용하여 Script ID를 생성하고, 이를 Script & ID TABLE에 삽입하여 테이블을 완성한다. 도 33에서는 Script Version이 #1 이라면, 도 36에서는 #2가 된다(3611 단계).

[0176] 도 37은 본 발명의 제4 실시 예에서 자바스크립트에 대한 변경된 정보를 전송하는 과정을 설명하는 도면이다. 자바스크립트(Javascript) 코드가 분할되고, Function & ID TABLE과 Script & ID TABLE 생성이 완료된 상태에서(3711 단계), 클라이언트가 HTTP 요청 메시지를 이용해 동일한 JavaScript 코드를 요청한다(3713 단계). 3715 단계에서 서버(3703)는 이전 자바스크립트(Javascript) 코드와 현재의 자바스크립트(Javascript) 코드를 함수 단위로 비교하여 새로 생성된 함수, 변경된 함수, 제거된 함수를 찾아낸다. 이를 바탕으로 Manipulation Info TABLE을 생성한다. A_2() 함수와 함수를 제외하고 남은 코드 부분을 의미하는 Remain은 코드가 변경되었기 때문에 Manipulation 속성을 Modified로 설정한다. A_3() 함수는 JavaScript 코드가 변경되면서 제거되었기 때문에 Manipulation 속성을 Removed로 설정한다. A_4() 함수는 이전 JavaScript에 없었던 함수기 때문에 Manipulation 속성을 New로 설정한다. Manipulation TABLE이 완성되면, Script & ID TABLE, Manipulation Info TABLE, 그리고 Manipulation Info TABLE의 Manipulation 속성이 Modified 또는 New로 설정된 함수 코드를 Gzip을 이용해 A.ajs 파일을 생성한다. A_2(), A_4()함수와 함수를 제외한 코드 부분인 Remain의 속성 값이 Modified이거나 New이기 때문에, 이 3 가지 코드가 Gzip의 압축 대상이 된다.

- [0177] 서버(3703)는 HTTP 응답 메시지를 이용해 A.ajs 파일을 클라이언트(3701)에게 전송한다(3717 단계).
- [0178] 도 38은 본 발명의 제4 실시 예에서 클라이언트의 자바스크립트 코드 복원 과정을 설명하는 도면이다. 클라이언트가 도 37의 과정과 같이 A.ajs 파일을 수신하면, 도 38과 같이 A.ajs 파일 JavaScript 코드로 변환한다.
- [0179] 클라이언트(3801)는 A.ajs 파일을 Gzip을 이용해 압축을 해제한다(3819 단계). 클라이언트는 이전에 수신한 JavaScript #1, Manipulation Info TABLE, 변경된 함수 코드를 이용해 JavaScript #2를 생성한다. Manipulation Info TABLE의 Manipulation 속성 값이 New인 함수는 JavaScript #1 코드에 추가 삽입하고, Manipulation 속성 값이 Removed인 함수는 JavaScript #1 코드에서 삭제하고, Manipulation 속성 값이 Modified인 함수는 JavaScript #1 코드에서 삭제되고 새로 수신된 함수 코드가 삽입된다. 그러므로 JavaScript #1 코드에서 있던 A_2() 함수와 함수를 제외한 나머지 Remain 코드를 삭제되고, 새로 수신된 A_2() 함수와 함수를 제외한 Remain 코드가 삽입된다.
- [0180] A_3() 함수는 JavaScript #1 코드에서 삭제되고, A_4() 함수는 새롭게 추가된다. 이 과정을 통해서 JavaScript #2를 생성한다. 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.
- [0181] 제4 실시 예에서 자바스크립트(Javascript) 코드가 변경되지 않은 경우의 동작은 웹 캐시가 발생하는 경우와 비슷하다. 도 38을 기준으로 할 때, 클라이언트는 JavaScript #2를 가지고 있으므로 HTTP 프로토콜 요청 메시지의 스크립트 버전(Script version) 헤더 값을 #2로 설정하여 서버로 자바스크립트(Javascript)를 요청한다.
- [0182] 이를 수신한 서버는 요청 메시지의 스크립트 버전(Script version) 헤더 값 통해서 해당 자바스크립트(Javascript) 코드가 변경되지 않아, 클라이언트가 가지고 있는 자바스크립트(Javascript)의 버전과 서버가 가지고 있는 자바스크립트(Javascript)의 버전이 같음을 알 수 있다. 그러므로 서버는 HTTP 프로토콜의 Not Modified 응답 메시지를 클라이언트에게 전송한다. 클라이언트는 클라이언트에 저장되어 있는 자바스크립트를 이용할 수 있다.
- [0183] 추가 실시 예로, 자주 쓰일 것이라고 예상되거나 자주 쓰이는 자바스크립트(Javascript) 코드를 코어 자바스크립트(Core JavaScript)로 선정하여 단말에게 미리 전송하고, 단말이 자바스크립트(Javascript) 코드를 요청하면, 프록시 서버 또는 웹 서버에서, 코어 자바스크립트(Core JavaScript)를 제외한 자바스크립트(Javascript) 코드와 제외된 코드에 대한 인덱스 정보를 클라이언트에게 전송하는 방안에 대하여 설명한다.
- [0184] 코어 자바스크립트(Core JavaScript)를 미리 전송하는 과정은, 자주 사용될 것이라고 예상되는 자바스크립트(Javascript) 코드를 코어 자바스크립트(Core JavaScript)로 미리 선정하고, 클라이언트의 자바스크립트(Javascript)를 요청하기 전에 서버가 코어 자바스크립트(Core JavaScript)를 클라이언트에게 미리 전송하는 것이다.
- [0185] 도 39는 본 발명의 실시 예에서 코어 자바스크립트를 미리 전송하는 과정을 설명하는 도면이다. 도 39를 참조하면, 서버(3903)는 자주 사용될 것이라고 예상되는 자바스크립트(Javascript) 함수를 미리 선정한다. 예를 들어, 서버(3903)는 100 개의 함수가 코어 자바스크립트(Core JavaScript)로 선정하고, 이를 Gzip으로 압축하여 Core.gz 파일을 생성할 수 있다.
- [0186] 클라이언트(3901)가 모바일 단말이라고 가정할 경우, 전원이 OFF 또는 SLEEP인 상태에서 ON으로 전환되면, 클라이언트(3901)는 코어 자바스크립트(Core JavaScript)를 압축한 파일인 Core.gz 파일을 서버(3903)에 요청한다(3911 단계). 이를 수신한 서버(3903)는 Core.gz 파일을 클라이언트(3901)에게 전송하고(3913 단계), 전송이 완료되면 클라이언트(3903)는 Gzip으로 압축을 해제하여 코어 자바스크립트(Core JavaScript)를 생성한다.
- [0187] 도 40은 본 발명의 실시 예에서 인덱스 파일과 코어 자바스크립트를 제외한 자바스크립트 코드를 전송하는 과정을 설명하는 도면이다. 도 39와 같이, 코어 자바스크립트(Core JavaScript)를 미리 전송하는 과정이 완료되면 클라이언트(4001)는 서버(4003)에게 코어 자바스크립트(Core JavaScript) 전송 방안을 이용해서 자바스크립트(Javascript) 코드를 효율적으로 전송 받을 수 있다.
- [0188] 도 40을 참조하면, 2 개의 JavaScript 리소스를 포함하는 웹 페이지를 요청하는 과정을 설명한다. 웹 페이지는

index.html 파일이고, 웹 페이지를 구성하는 리소스는 자바스크립트(JavaScript)로 Original_A.js와 Original_B.js이다.

[0189] 이와 같이 웹 페이지와 이를 구성하는 리소스가 생성되면, 서버(4003)는 웹 페이지를 구성하는 자바스크립트(JavaScript) 코드에서 코어 자바스크립트(Core JavaScript)를 제외한 코드를 생성하고, 어떤 코어 자바스크립트(Core JavaScript)가 사용되었는지를 나타내는 인덱스 파일을 생성한다. 이 과정을 통해서 Core JavaScript가 제거된 코드인 A.js와 B.js가 생성되고, 인덱스 파일인 Core.index 파일이 생성된다.

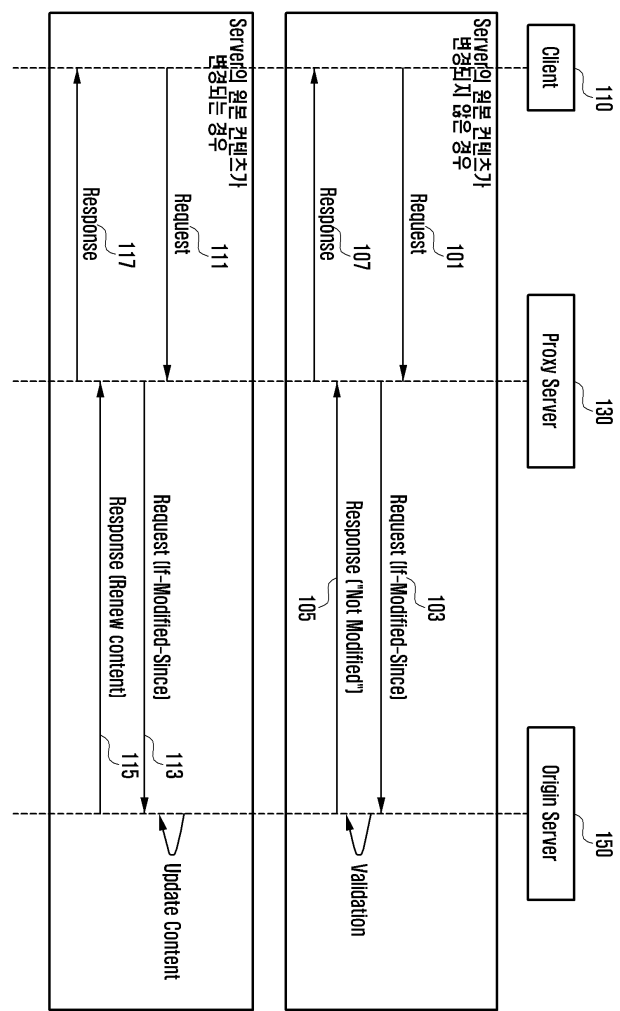
[0190] 클라이언트(4001)는 웹 페이지를 서버(4003)에게 요청한다(4011 단계). 클라이언트(4001)는 서버(4003)로부터 웹 페이지를 수신한다(4013 단계). 웹 페이지를 수신한 클라이언트(4001)는 웹 페이지를 구성하는 리소스인 자바스크립트(JavaScript)를 요청하기에 앞서 서버(4003)에 인덱스 파일인 Core.index 파일을 요청한다(4021 단계). 클라이언트(4001)는 서버(4003)로부터 Core.index를 수신한다(4023 단계). Core.index 요청 이후, 클라이언트(4001)는 서버(4003)로 다음 A.js와 B.js를 요청한다(4031, 4041 단계). Core.index, A.js, B.js 파일 수신이 완료되면, 클라이언트(4001)는 인덱스 파일의 정보를 이용해 JavaScript 코드를 복원한다.

[0191] Core.index 파일에 A.js 파일이 코어 자바스크립트(Core JavaScript) 중에서 core_1()부터 core_70()까지의 함수가 사용되었다는 인덱스 정보가 작성되어 있기 때문에, A.js 파일에 core_1()부터 core_70()까지의 함수를 삽입하여 Original_A.js를 복원한다. B.js 파일도 마찬가지로, Core.index 파일의 인덱스 정보를 확인하고 core_1()부터 core_80()까지의 함수를 삽입하여 Original_B.js를 복원한다.

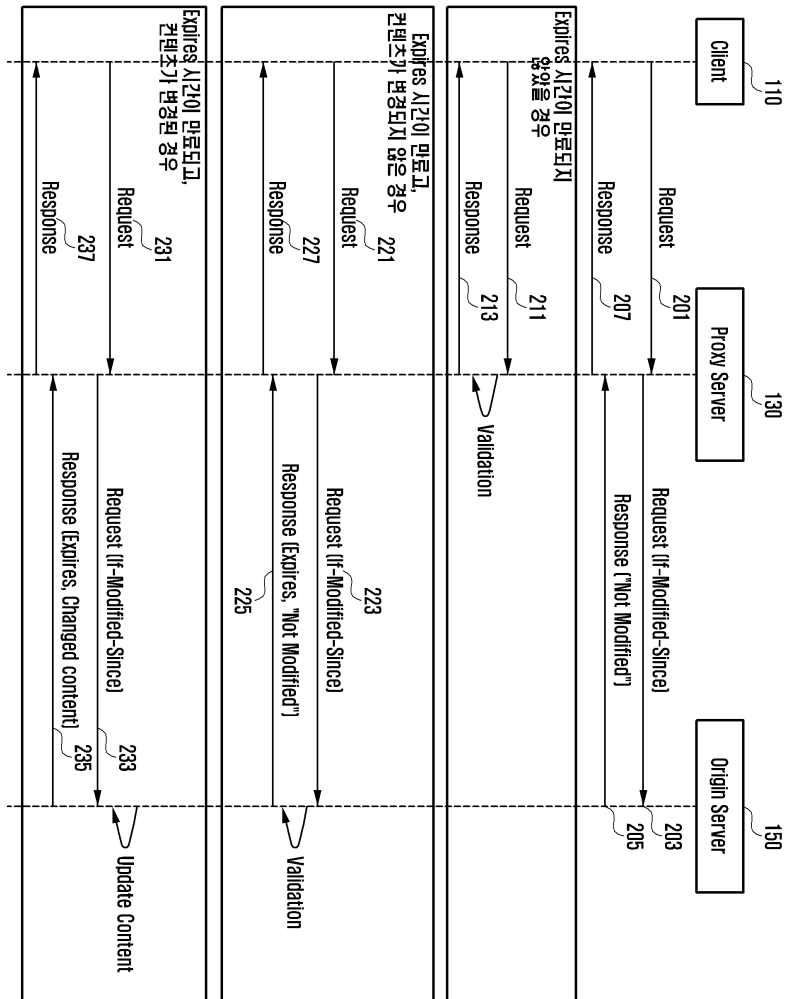
[0192] 본 명세서와 도면에 개시된 실시 예들은 본 발명의 내용을 쉽게 설명하고, 이해를 돕기 위해 특정 예를 제시한 것일 뿐이며, 본 발명의 범위를 한정하고자 하는 것은 아니다. 따라서 본 발명의 범위는 여기에 개시된 실시 예들 이외에도 본 발명의 기술적 사상을 바탕으로 도출되는 모든 변경 또는 변형된 형태가 본 발명의 범위에 포함되는 것으로 해석되어야 한다.

도면

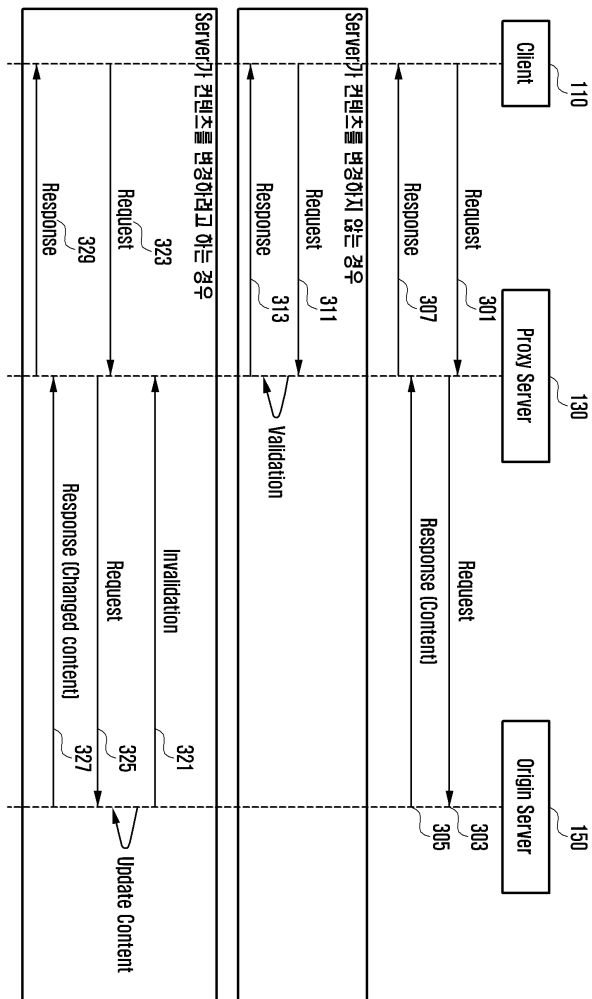
도면1



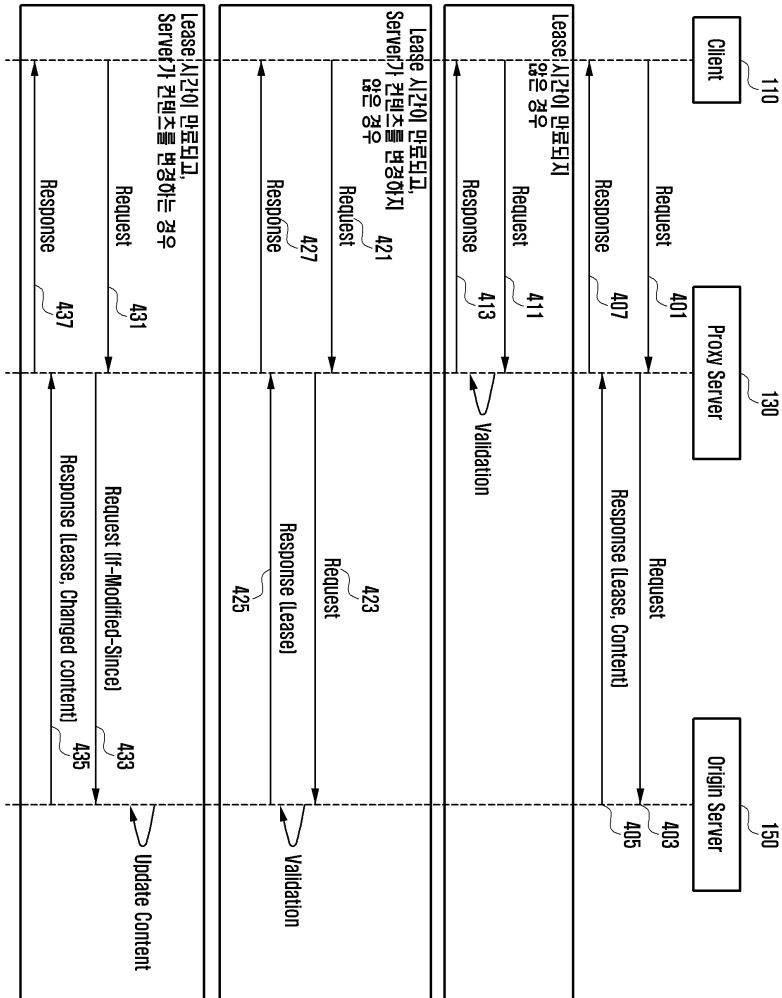
도면2



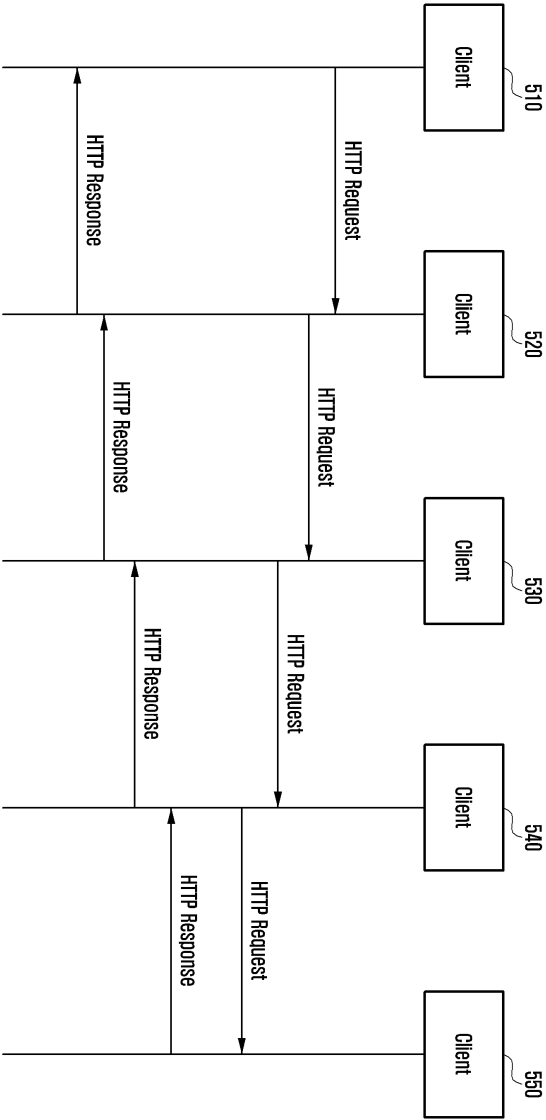
도면3



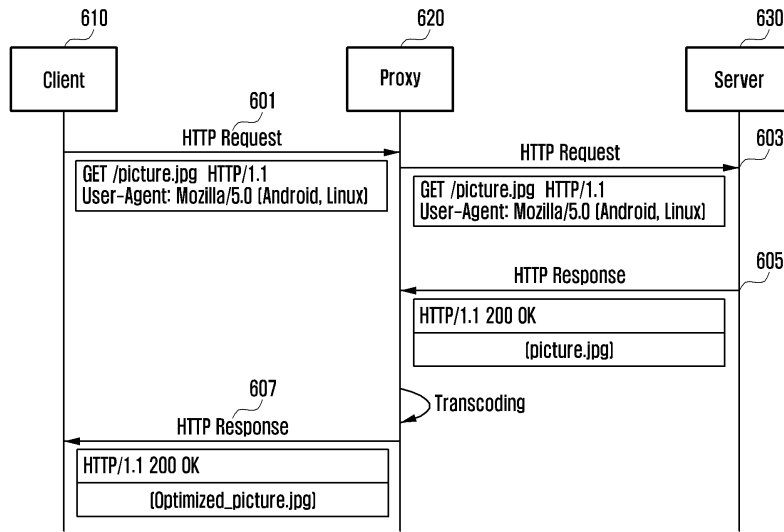
도면4



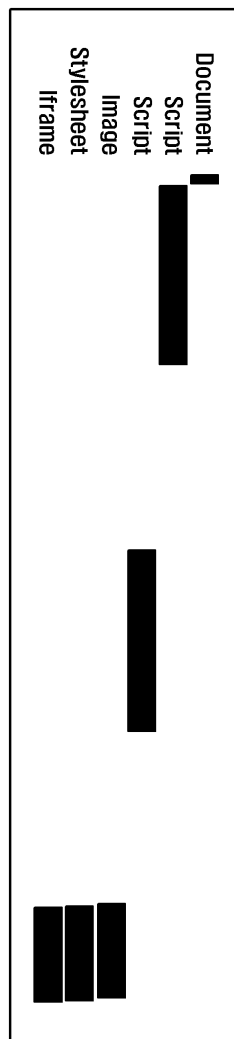
도면5



도면6



도면7



도면8

```
<html>
  <head>
    <title> example 1 </title>
  </head>

  <body>
    <script type= " text/javascript " src= " ./a.js " > </script>
    <script type= " text/javascript " src= " ./b.js " > </script>
  </body>
</html>
```

example1. html

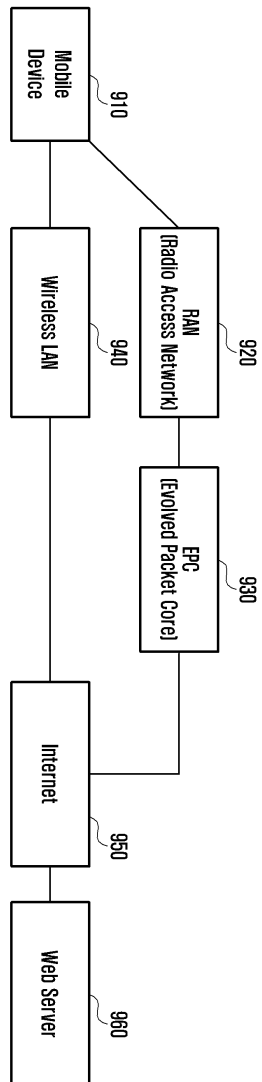
```
alert( " Hello. " );
```

a.js

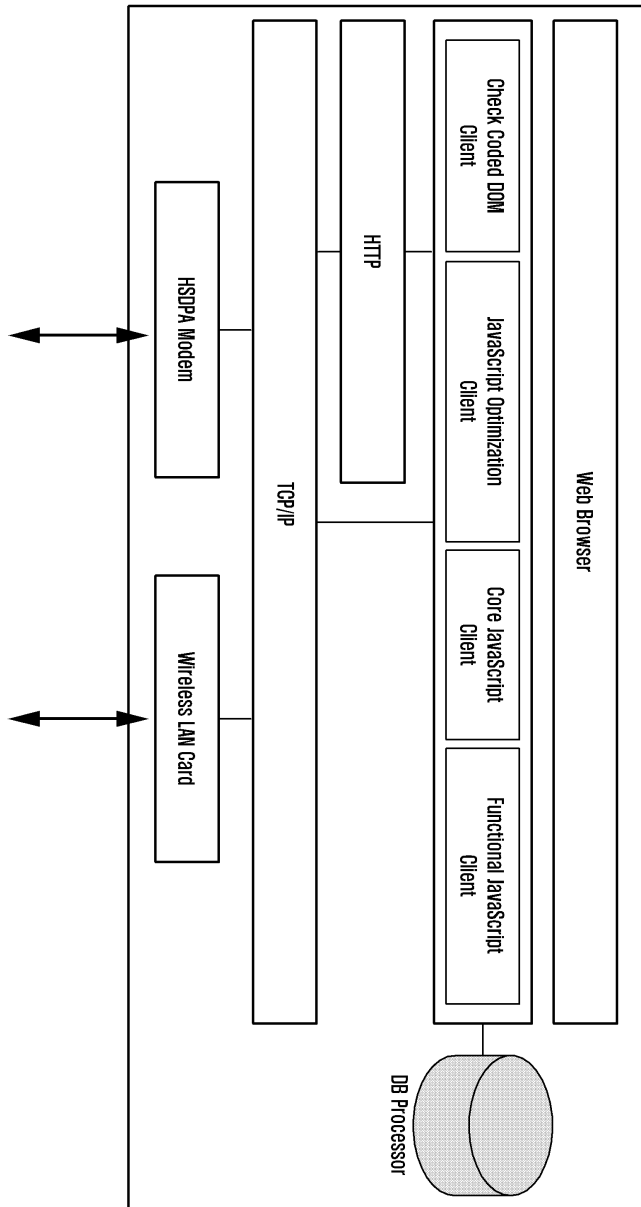
```
alert( " Good Bye. " );
```

b.js

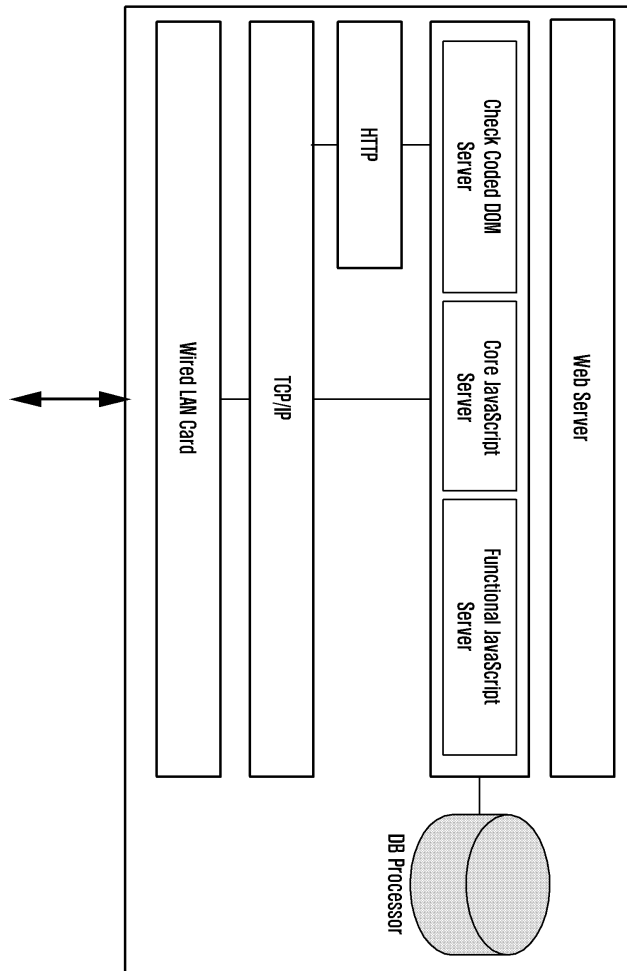
도면9



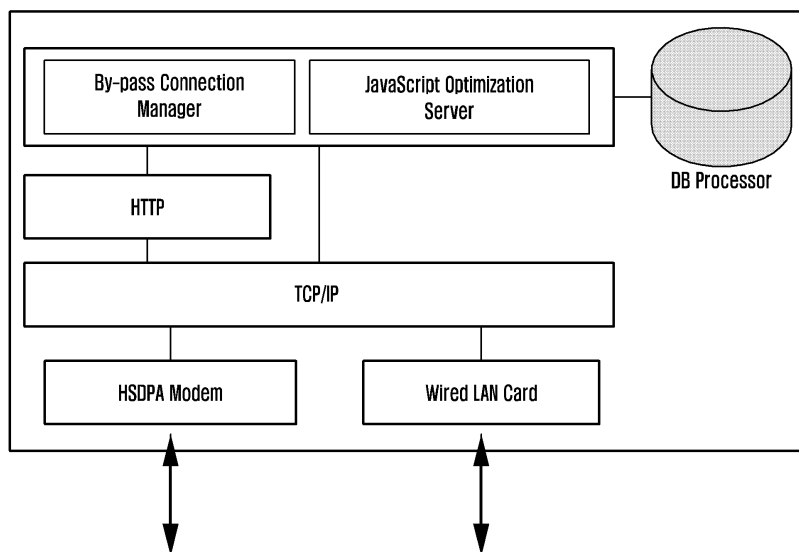
도면10



도면11



도면12



도면13

Current Resource URL & ID TABLE	
Resource URL	Resource ID
"http://server1.com/frame1.html"	5361e75bdf8662d2b0625561734b052083fe4c56
"http://server2.com/image2.jpg"	0183a8eb3d28a61335d1b32362efaf6eacfb6c89

Previous Resource URL & ID TABLE	
Resource URL	Resource ID
"http://server1.com/frame1.html"	5361e75bdf8662d2b0625561734b052083fe4c56
"http://server2.com/image2.jpg"	0183a8eb3d28a61335d1b32362efaf6eacfb6c89

도면14

Proxy-to-Proxy Forwarding Info TABLE

Content Prefix	IP Address	Port Number
Khu.ac.kr/	6.6.6.6	1600
Khu.ac.kr/js	6.6.6.6	1600
Khu.ac.kr/dart	6.6.6.6	1600

도면15

Supported Language Info TABLE

Browser	Supported language		
	JavaScript	Python	Dart
Firefox	O	O	X
Explorer	O	X	X
Chrome	O	O	O
Safari	O	X	O

도면16

Supported JavaScript Library Info TABLE

Version	Supported Frameworks & Extensions			
	jQuery	YUI	SVG	Dojo
Chrome 28	O	X	X	X
Chrome 29	O	X	O	O
Chrome 30	O	O	O	O

Version	Supported Frameworks & Extensions			
	jQuery	YUI	SVG	Dojo
Safari 4	O	X	X	X
Safari 5	O	X	O	X
Safari 6	O	O	O	O

Version	Supported Frameworks & Extensions			
	jQuery	YUI	SVG	Dojo
Firefox 23	O	X	X	X
Firefox 24	O	O	X	O
Firefox 25	O	O	O	O

Version	Supported Frameworks & Extensions			
	jQuery	YUI	SVG	Dojo
Explorer 8	O	X	X	X
Explorer 9	O	O	X	X
Explorer 10	O	O	O	O

도면17

Function & ID TABLE

Function Name	Function ID
A_1()	1b3313872c55fcd556b47e062c8c6ecd7365a467
A_2()	a0171dcb74d7f231adf1aec8602a12d73b0c5d58
A_3()	222090b1e4c79243e3b8631babb7b49d3fd15bf2
Remain	37f10ed4c52b172e73833602487b1953c07356fb

도면18

Script & ID TABLE

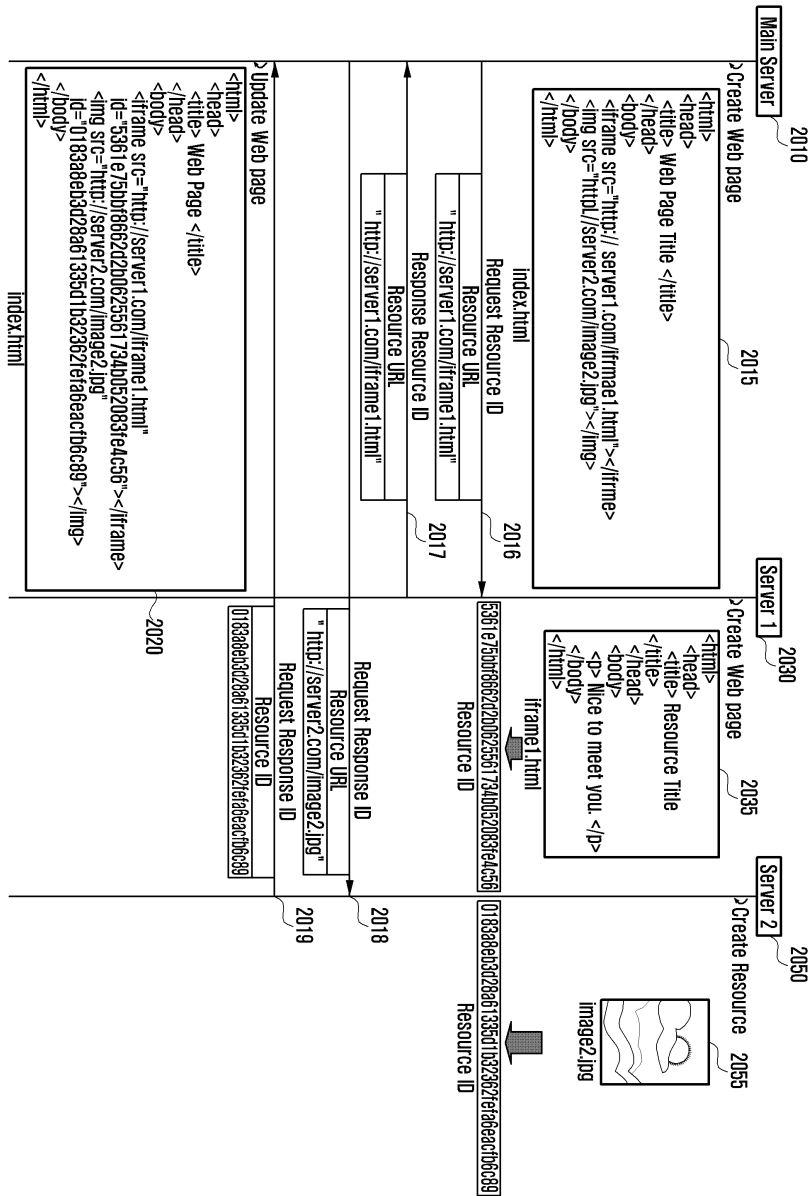
Script Version	Script ID
#1	cb917dc448563943e435a5c57a35198280506040

도면19

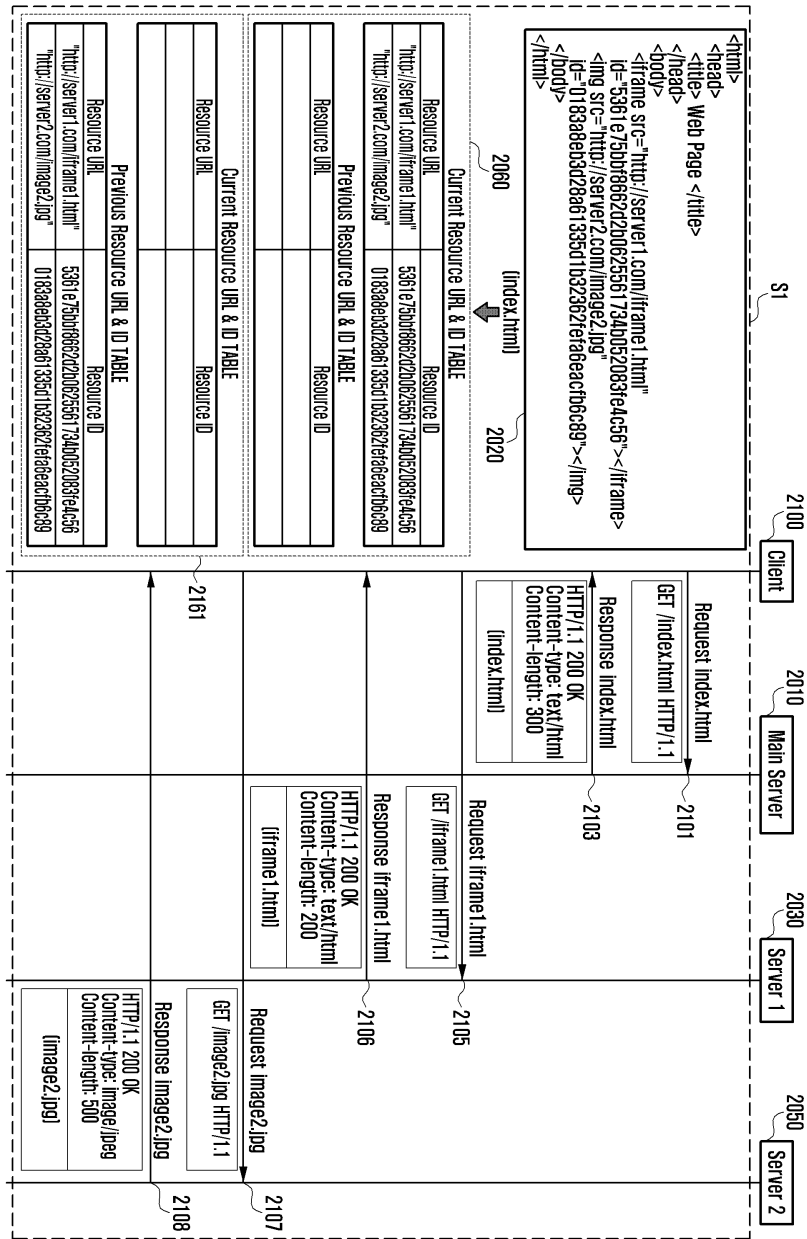
Manipulation Info TABLE

Function name	Manipulation
A_1()	New
A_2()	New
A_3()	New
Remain	New

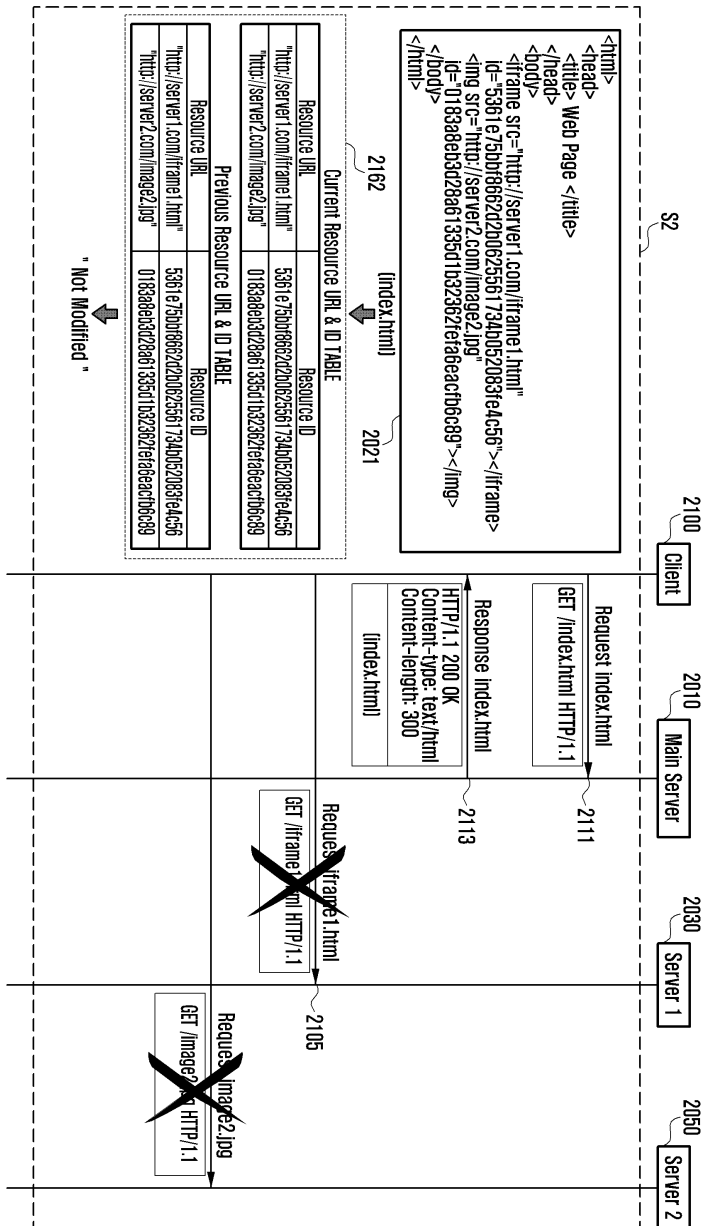
도면20



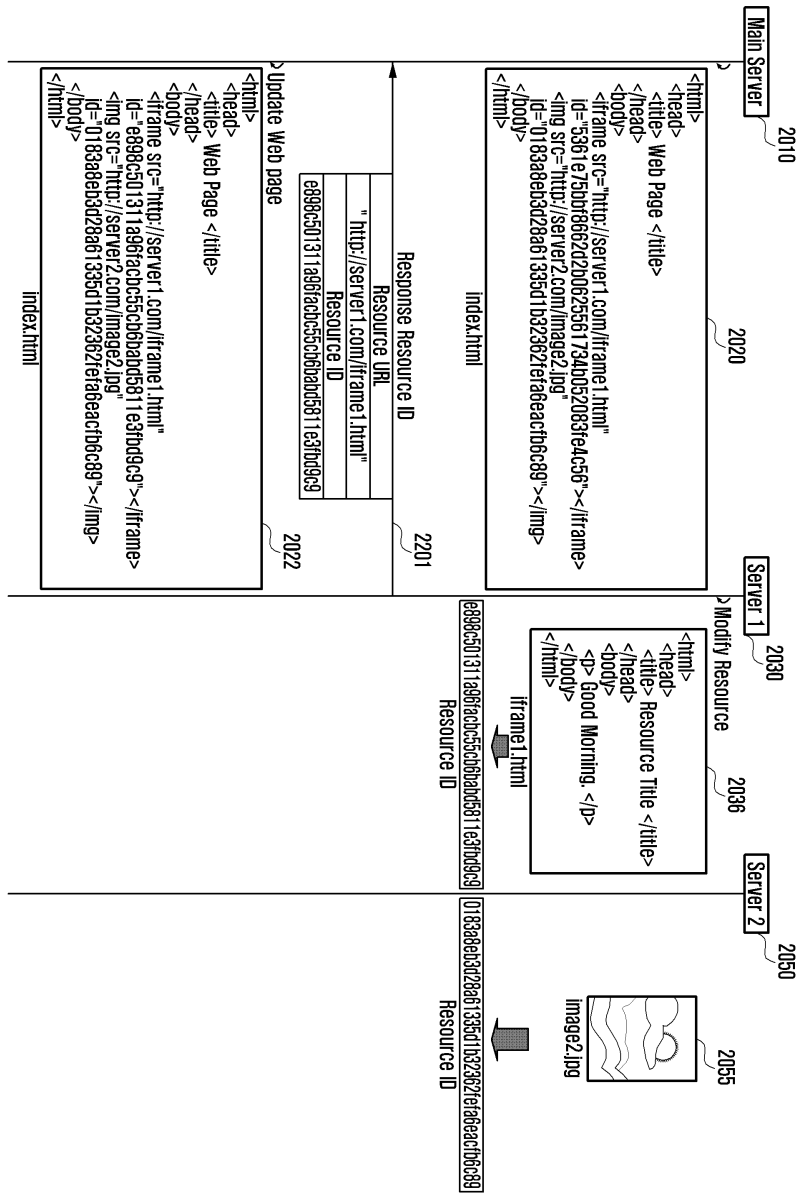
도면21a



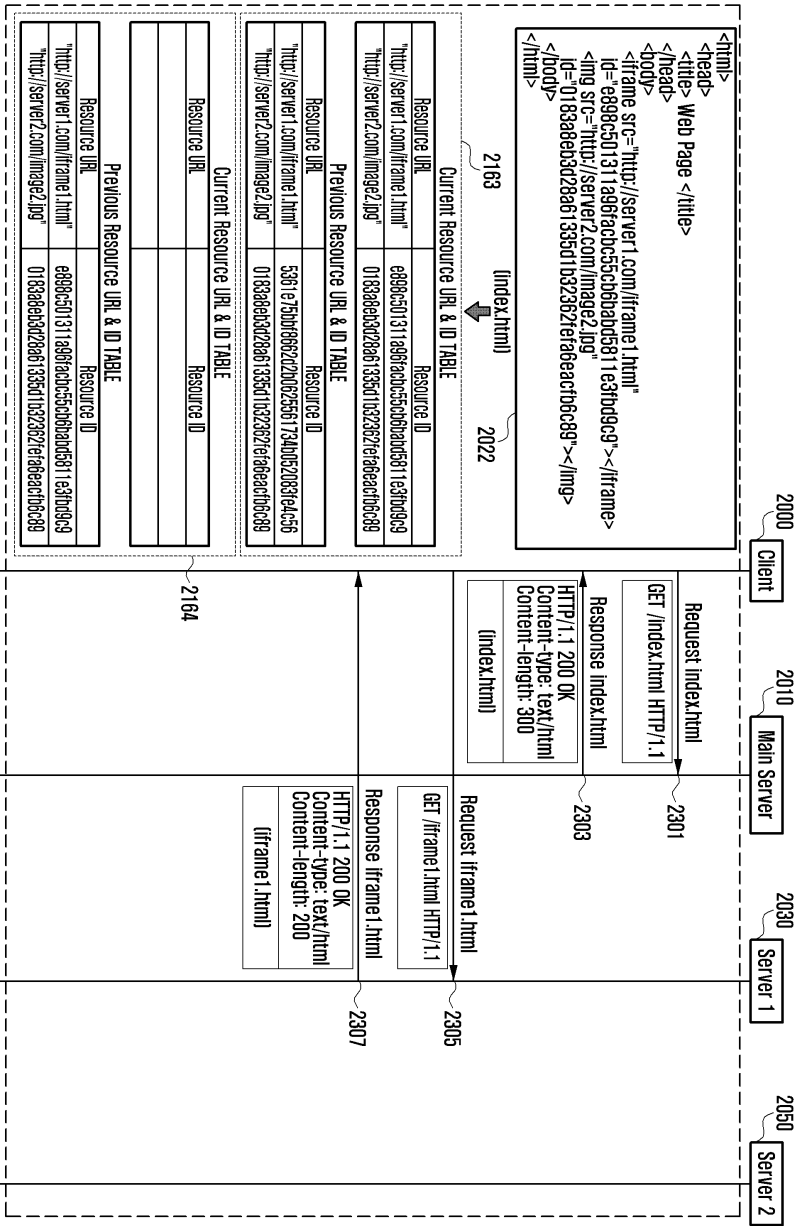
도면21b



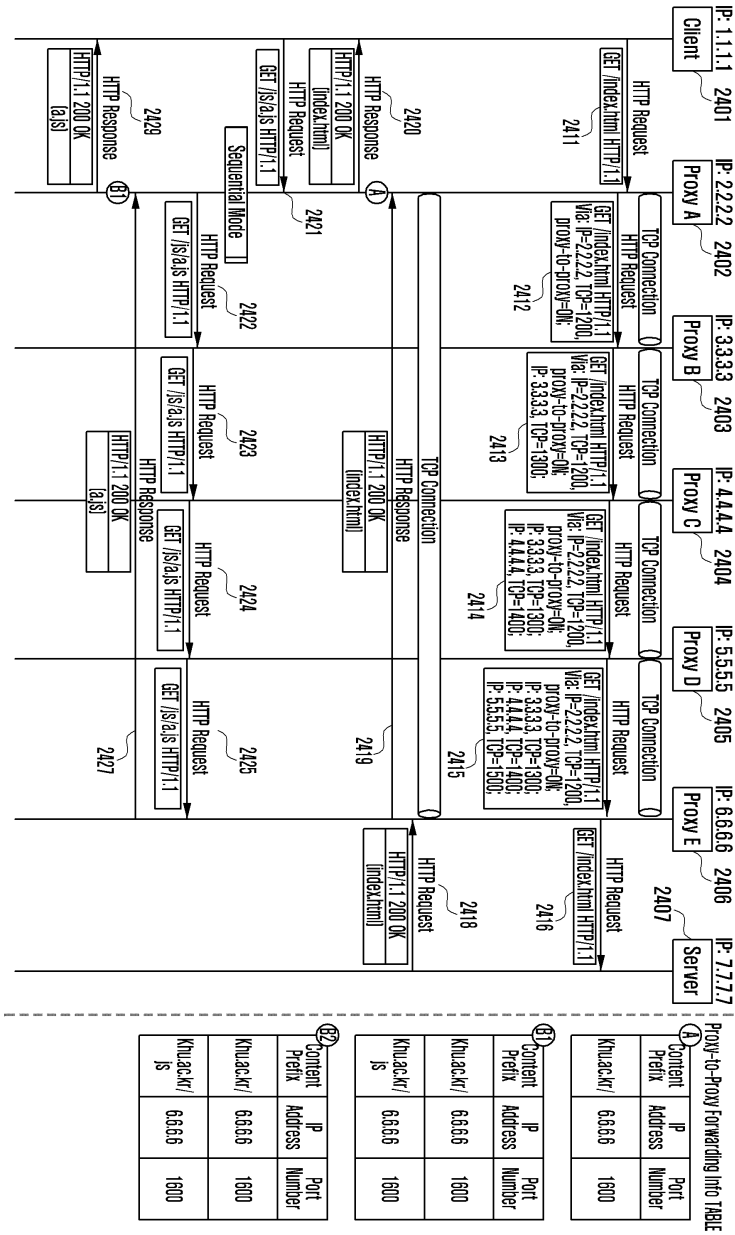
도면22



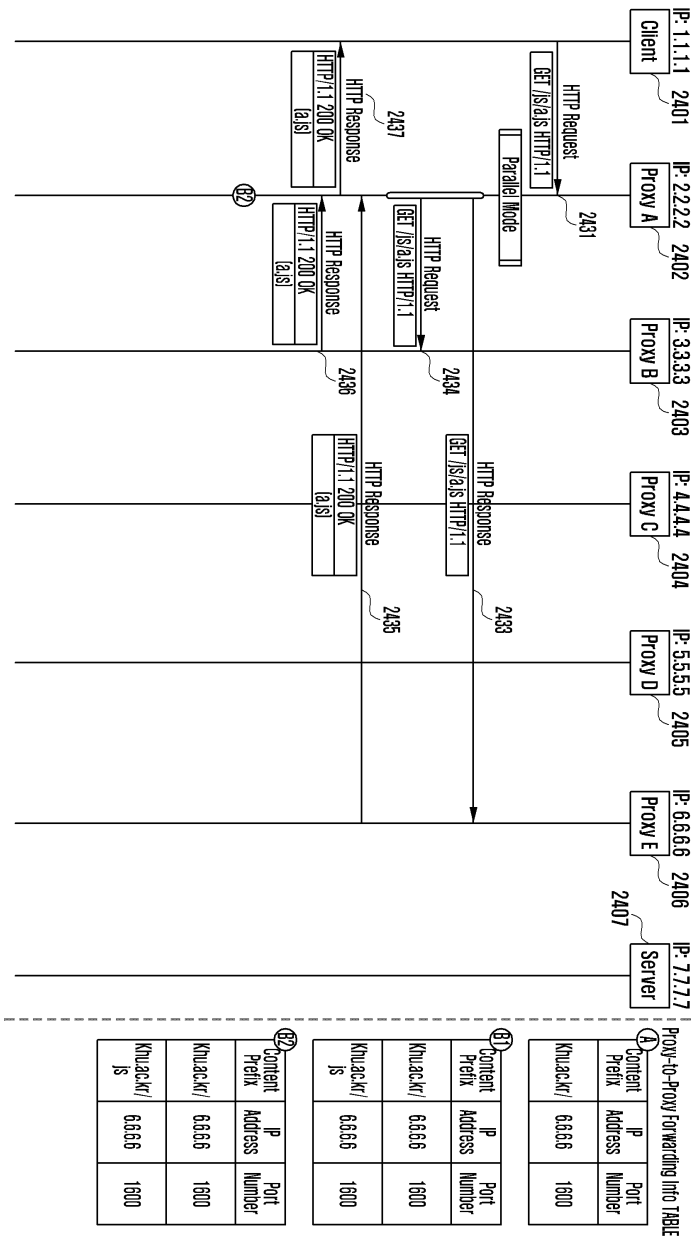
도면23



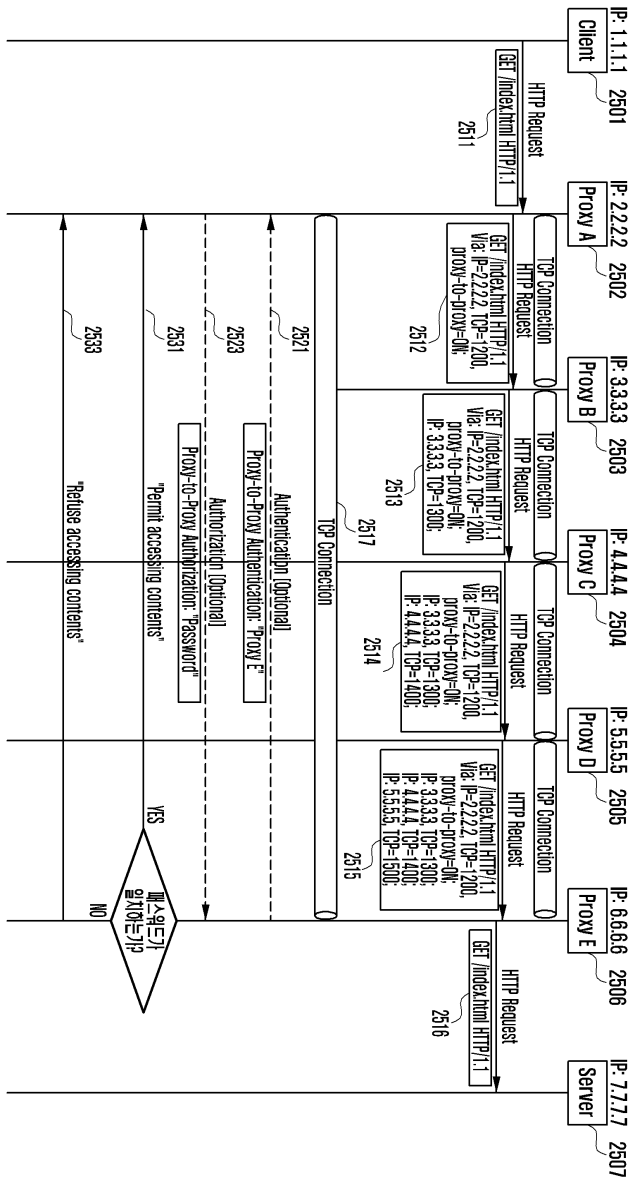
도면24a



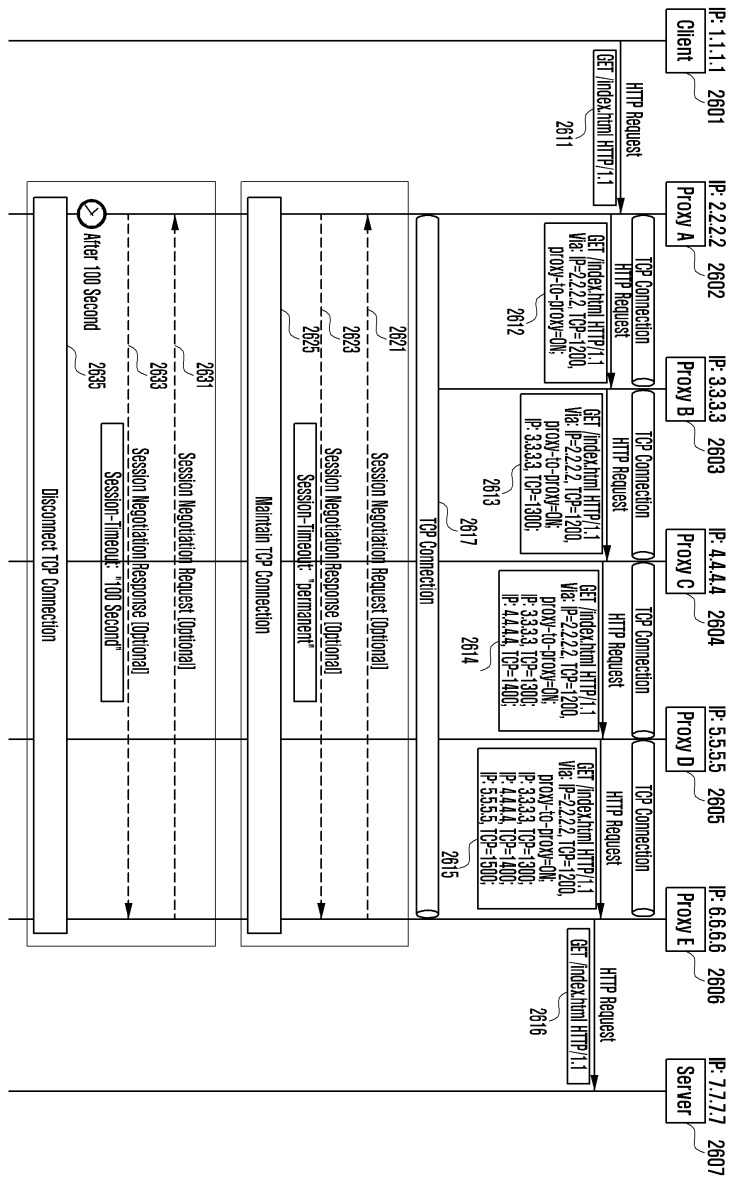
도면24b



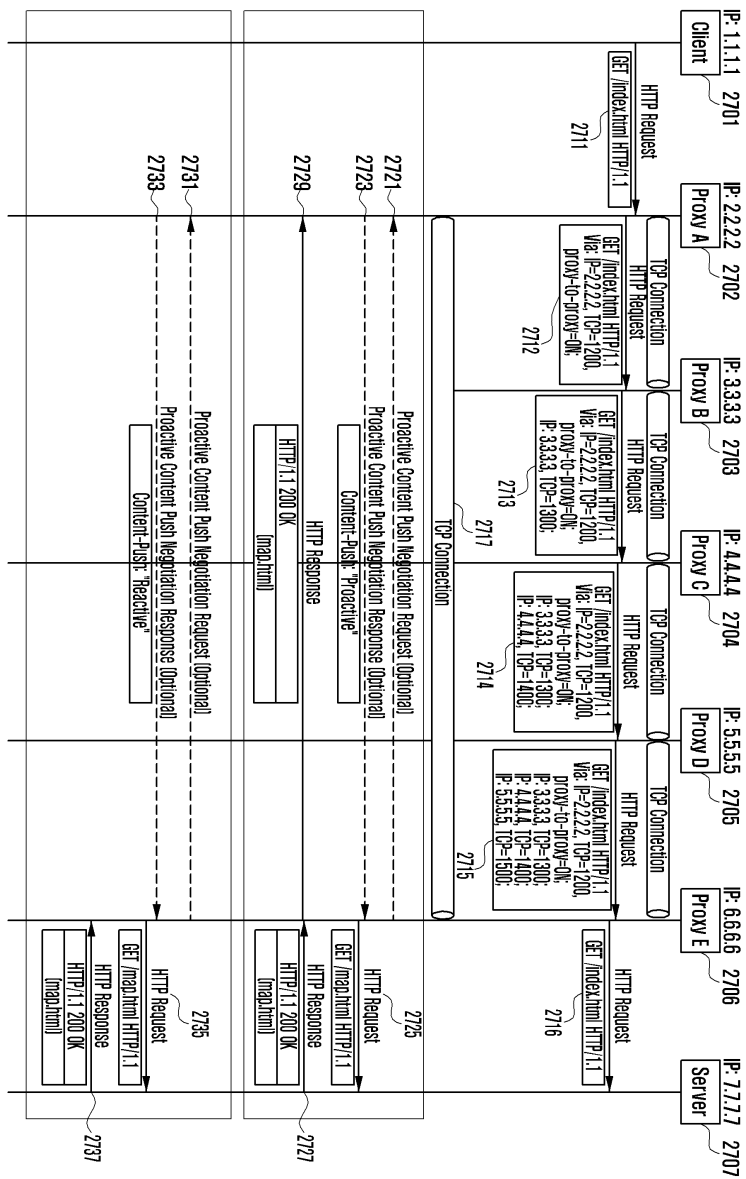
도면25



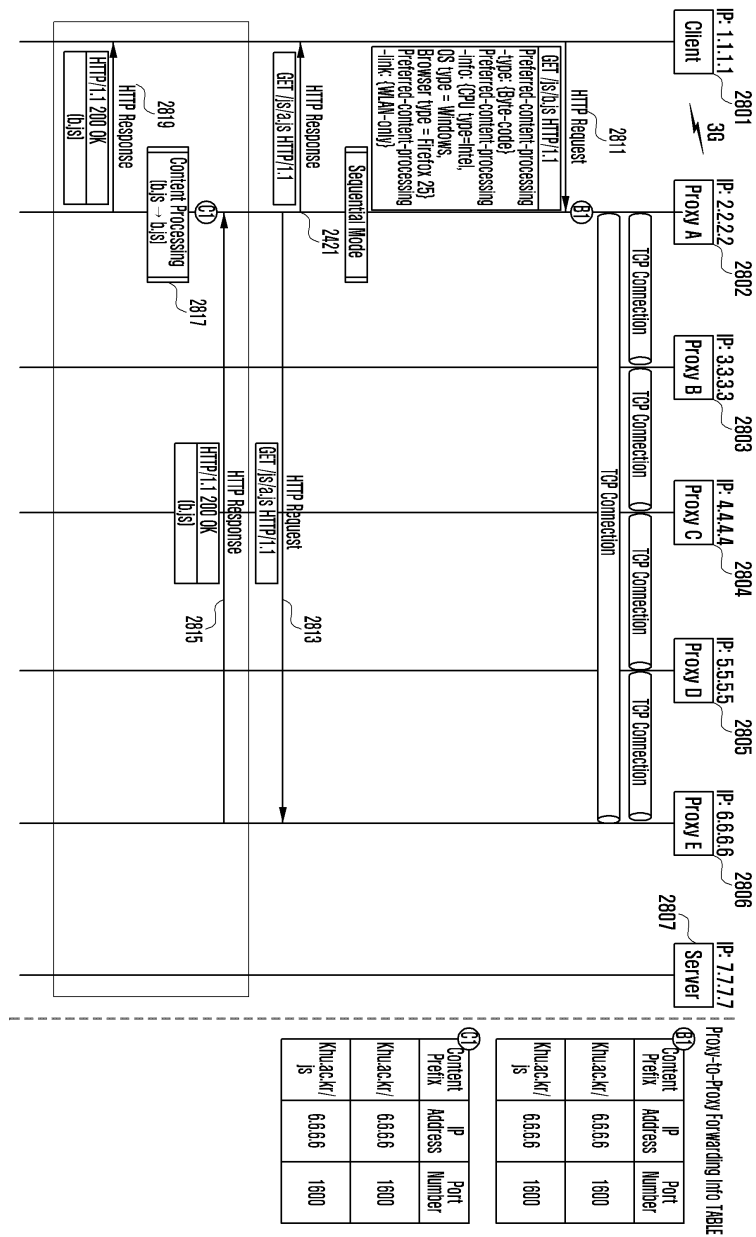
도면26



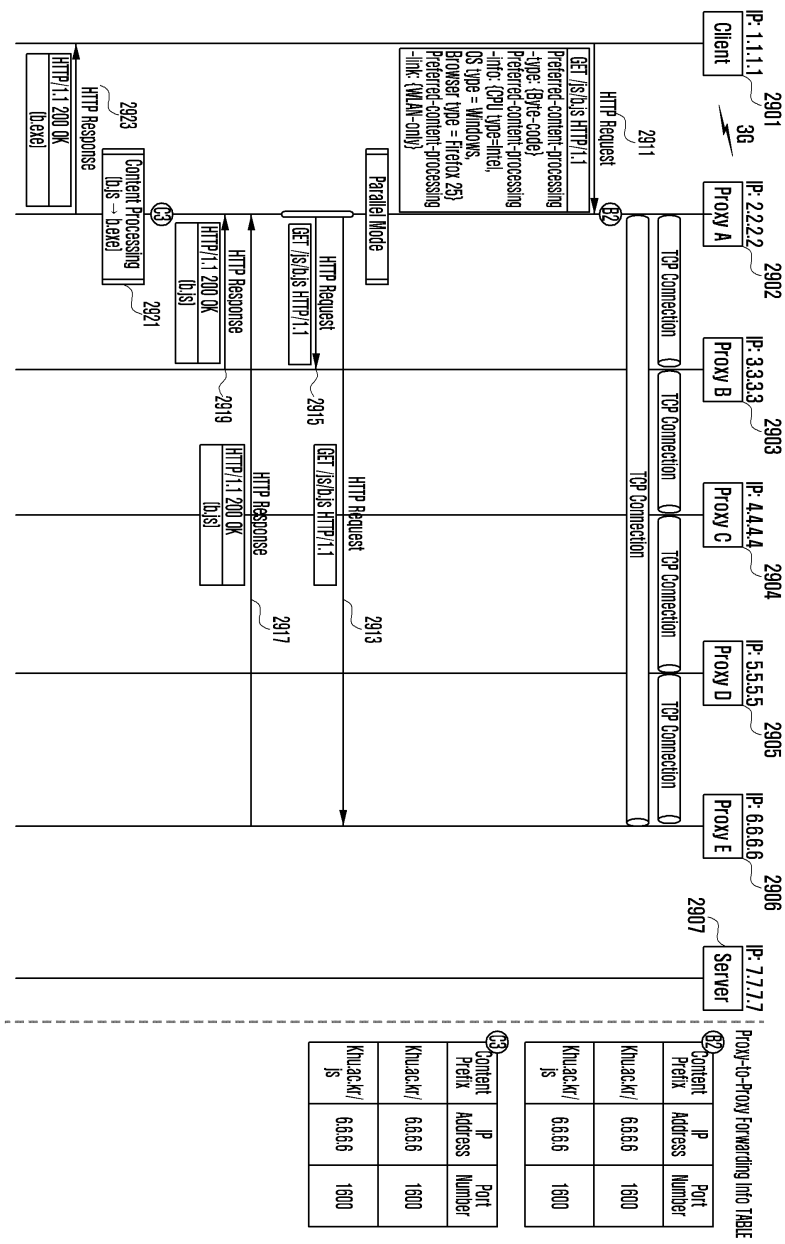
도면27

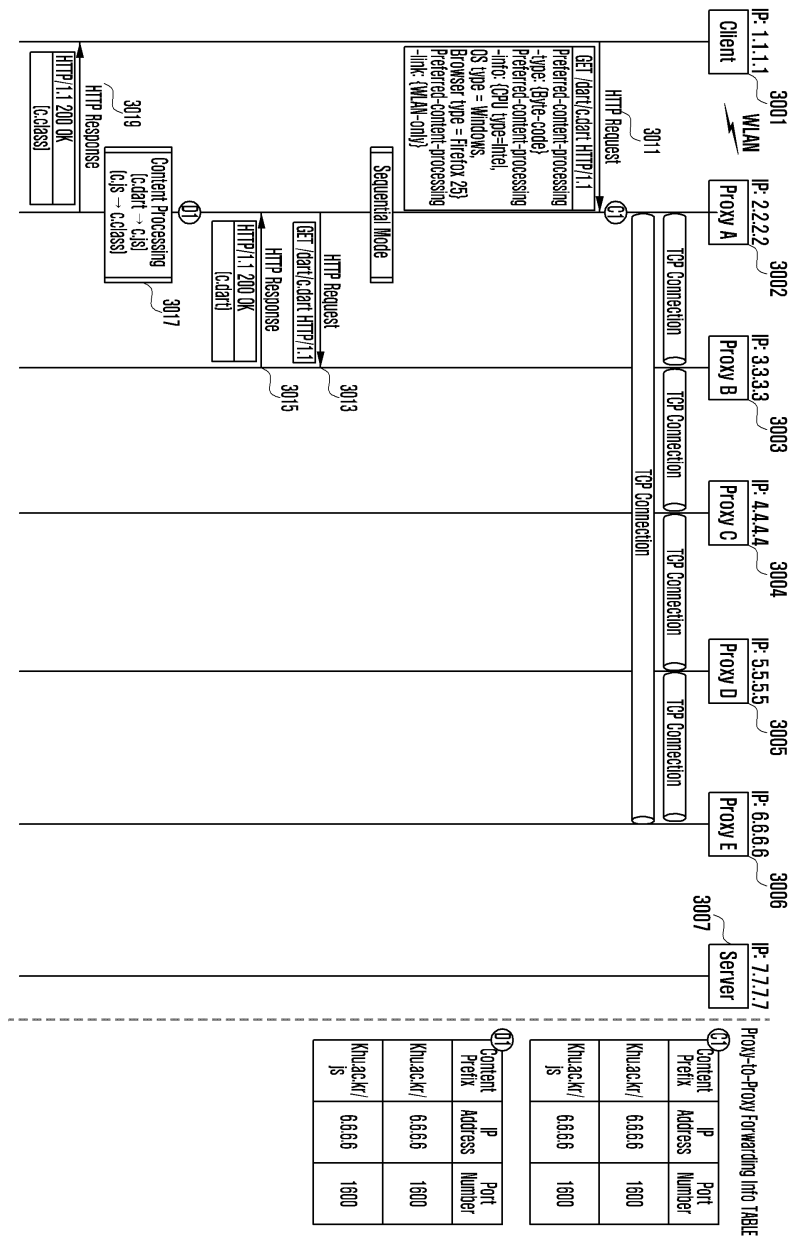


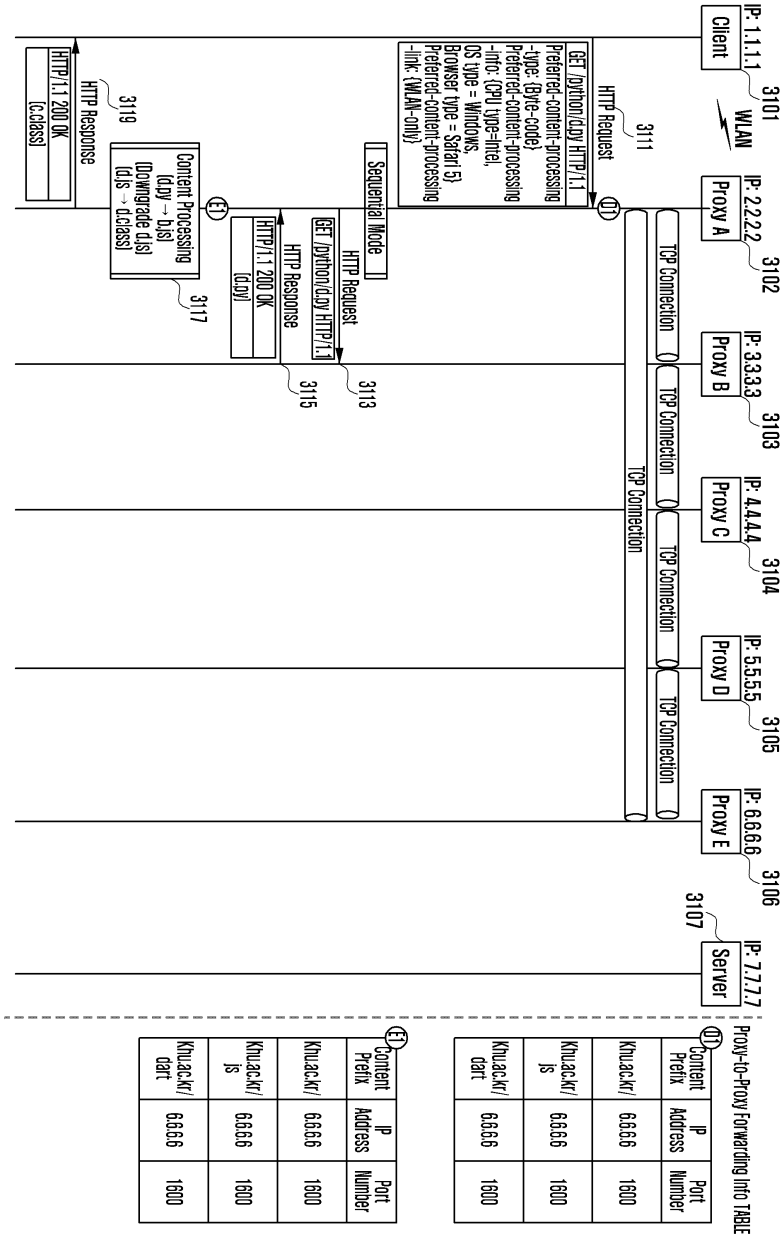
도면28



도면29

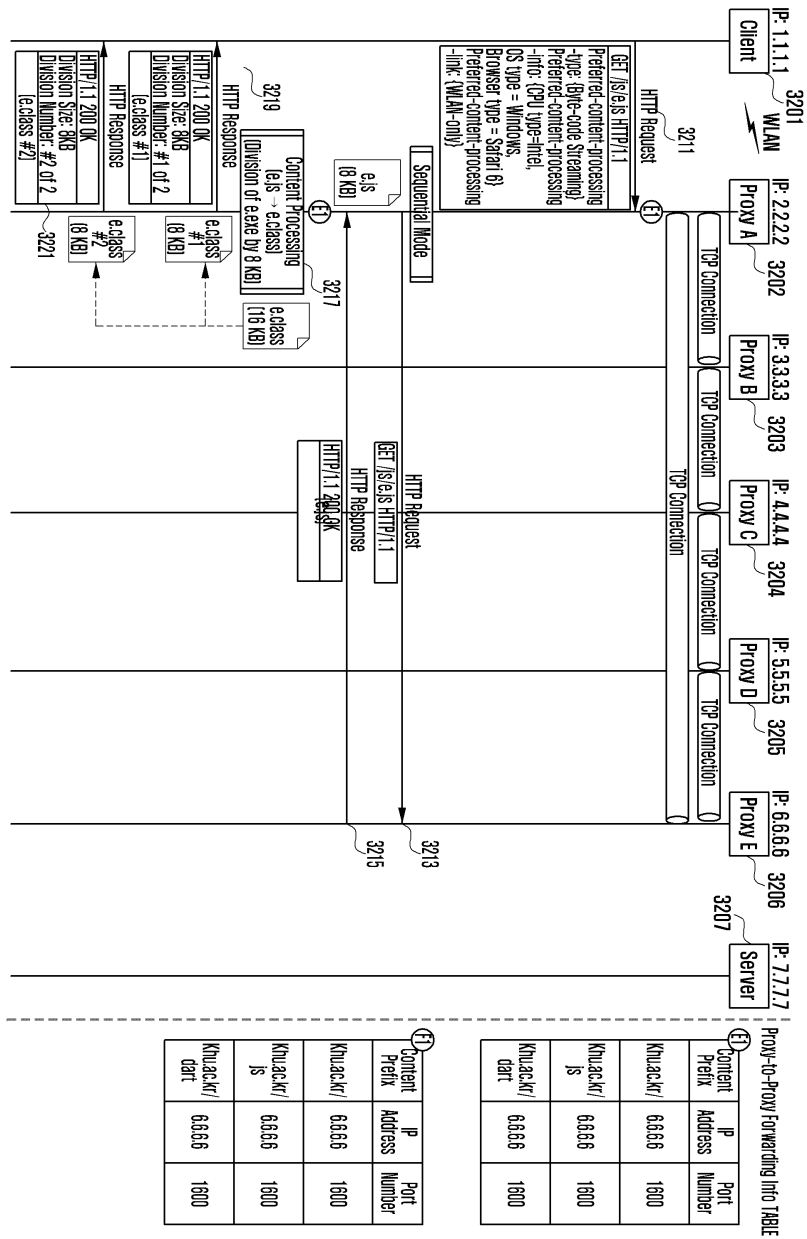


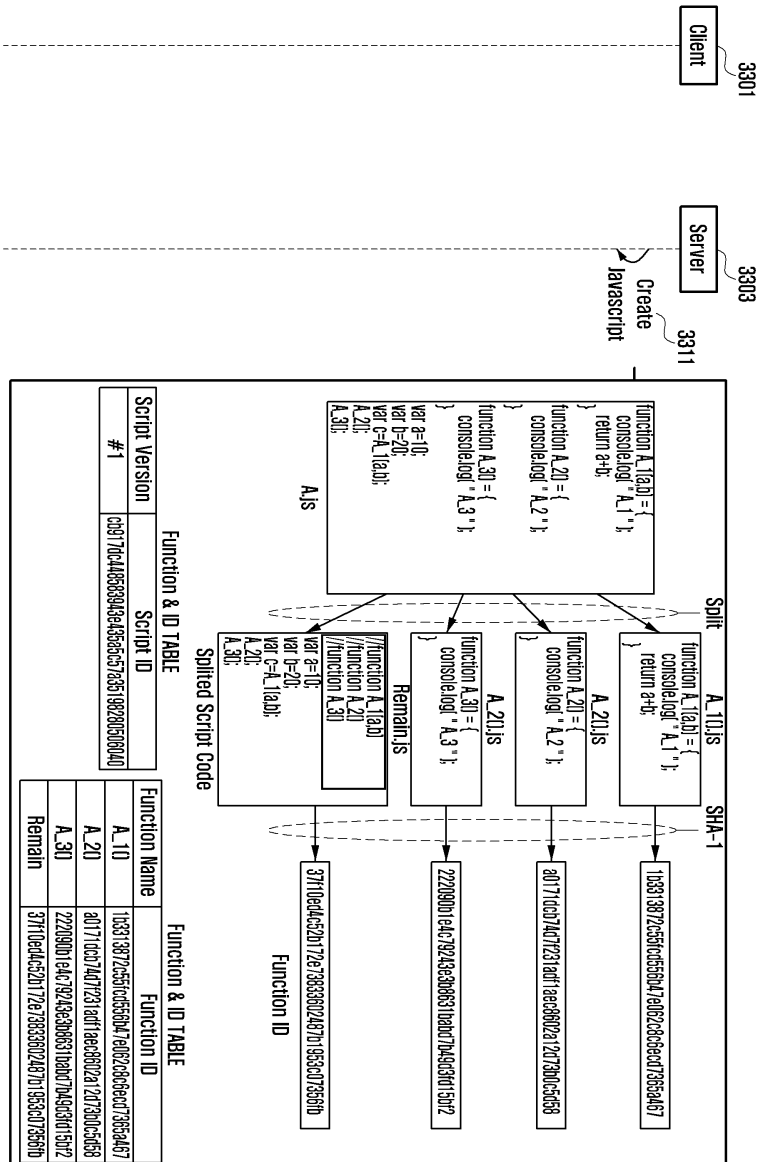




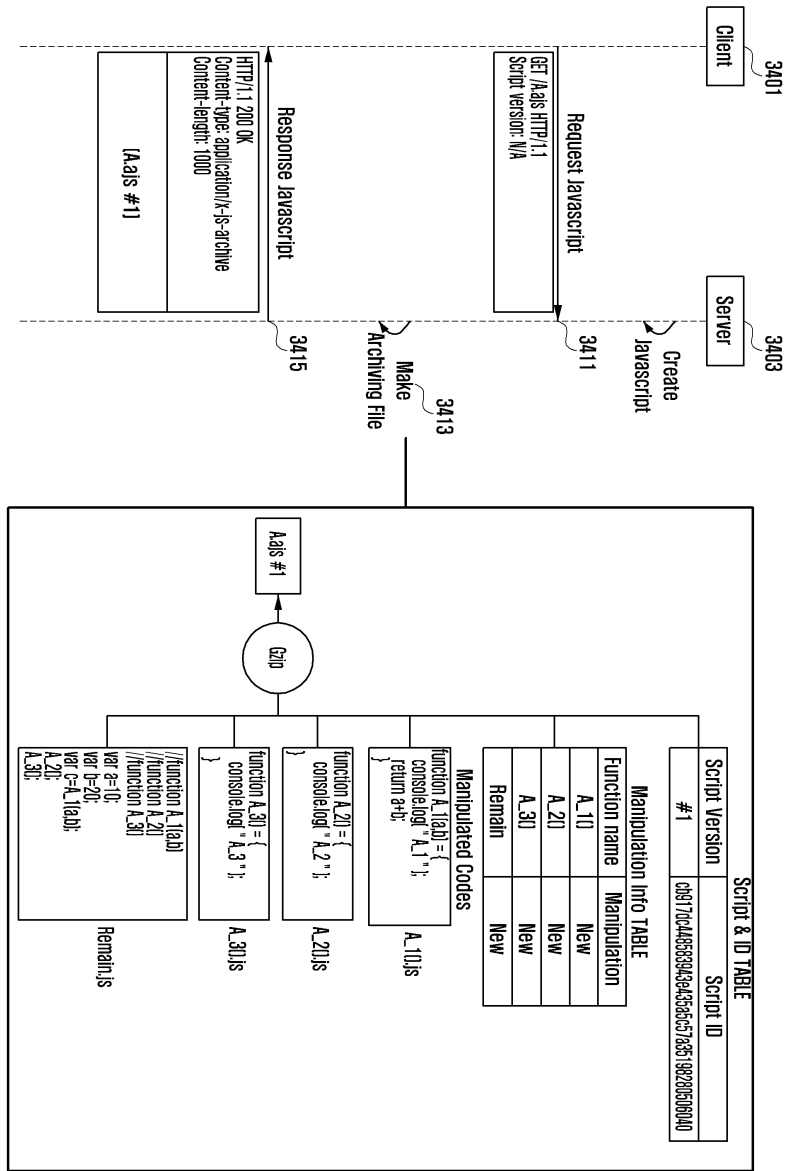
도면31

도면32

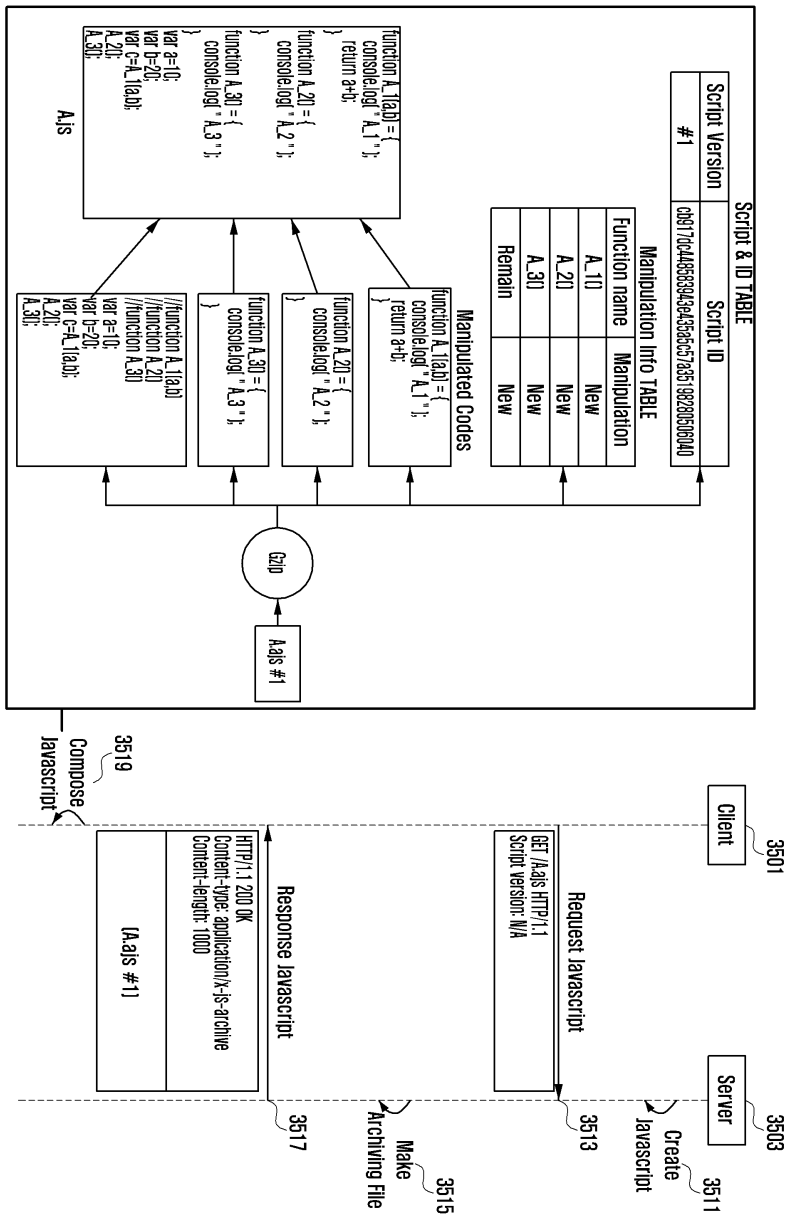


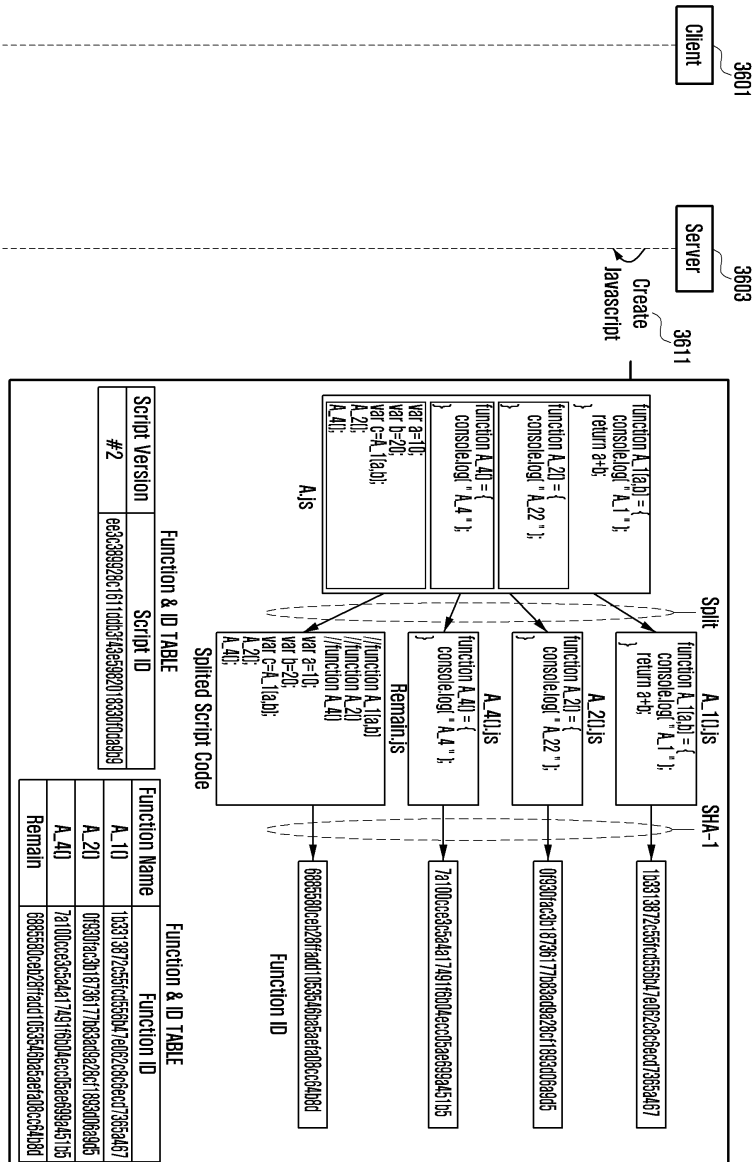


도면34

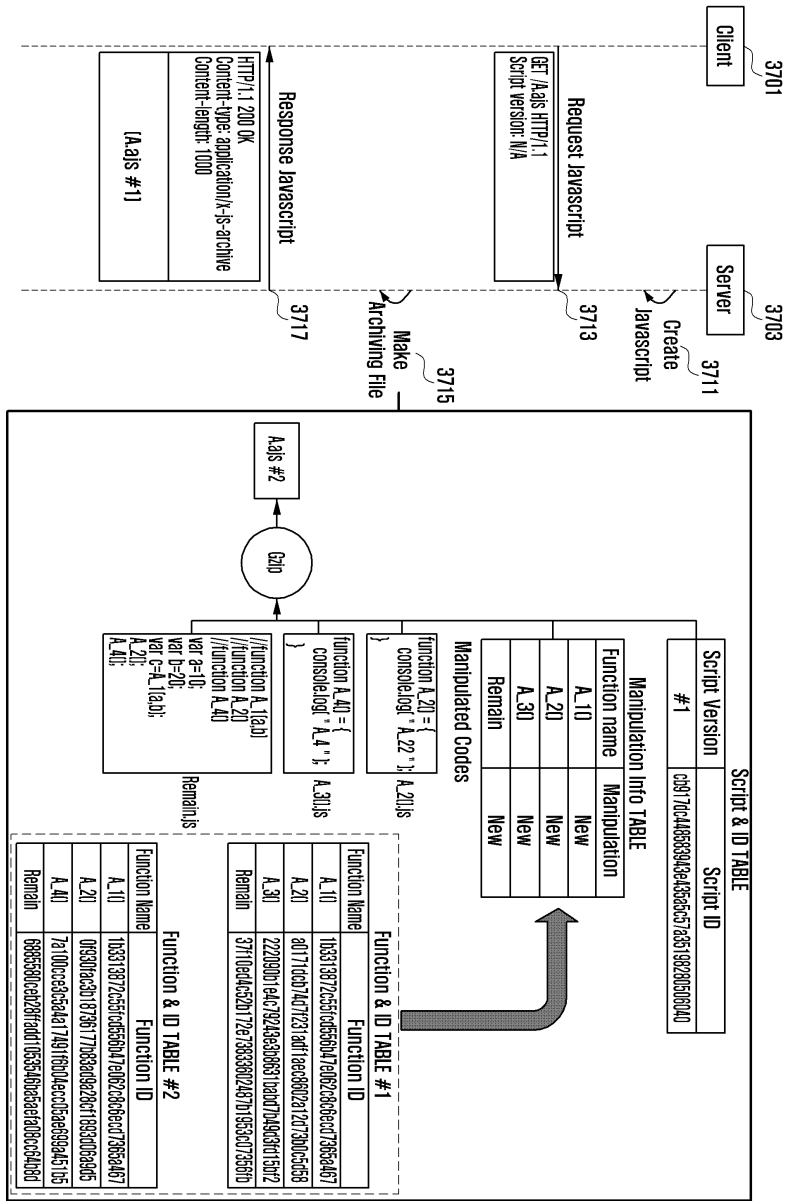


도면35

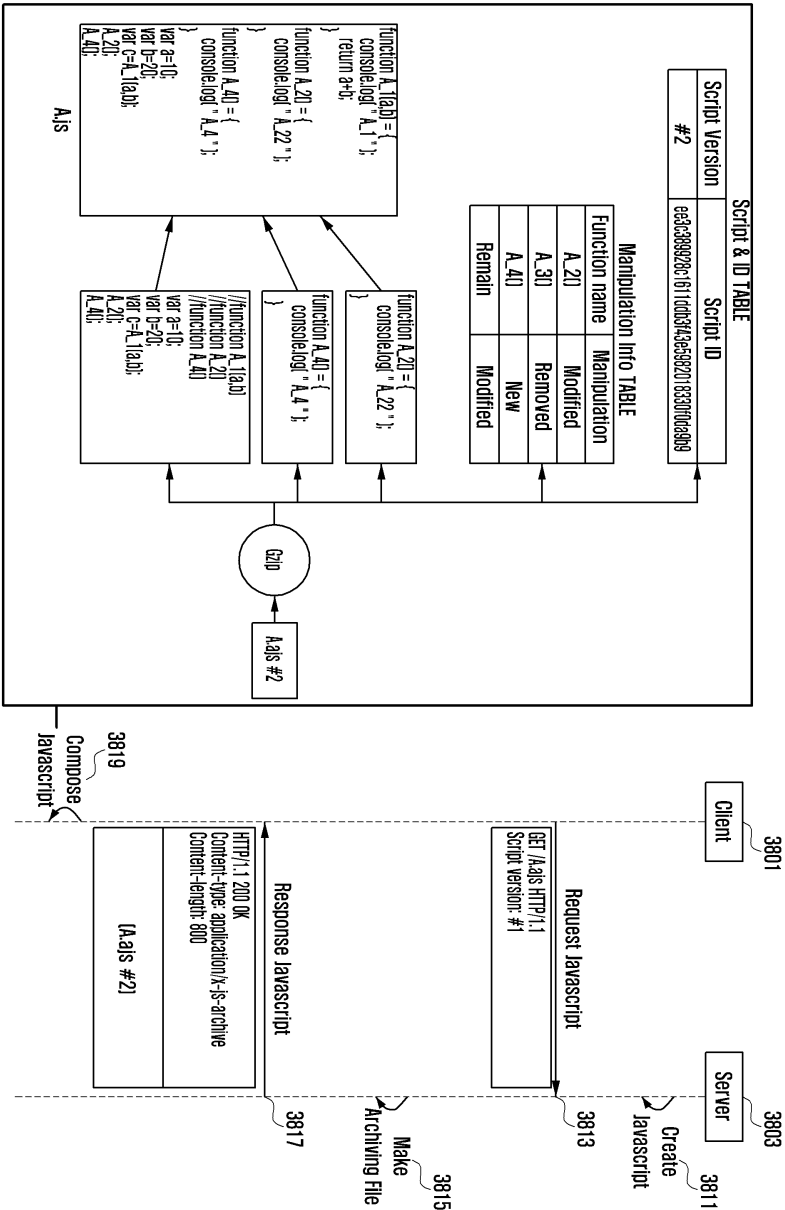




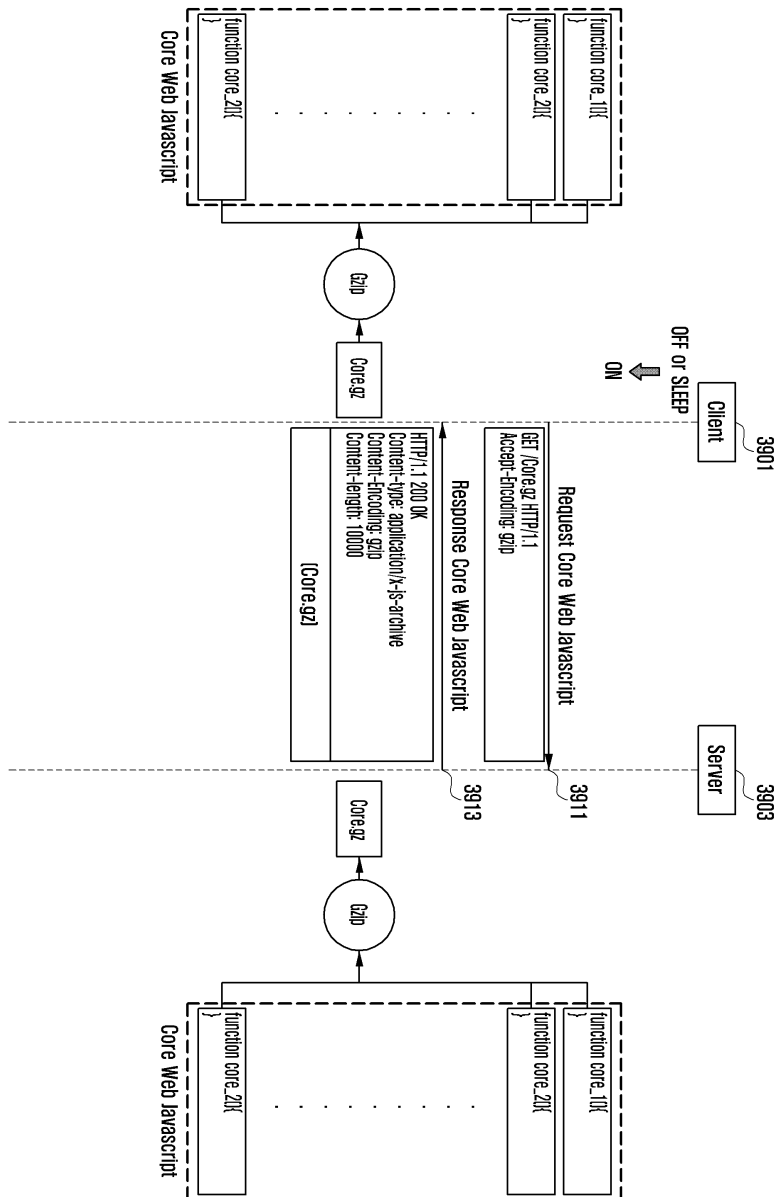
도면37



도면38



도면39



도면40

