



(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2008-0110894
(43) 공개일자 2008년12월19일

(51) Int. Cl.

H04L 12/12 (2006.01) H04L 12/56 (2006.01)
H04W 8/24 (2008.10)

(21) 출원번호 10-2008-7027550

(22) 출원일자 2008년11월11일

심사청구일자 없음

번역문제출일자 2008년11월11일

(86) 국제출원번호 PCT/US2007/066433

국제출원일자 2007년04월11일

(87) 국제공개번호 WO 2007/121241

국제공개일자 2007년10월25일

(30) 우선권주장

60/744,720 2006년04월12일 미국(US)

(71) 출원인

사이트릭스 시스템스, 인크.

미국 플로리다 33309, 포트 라우더데일, 851 더블류. 사이프레스 크릭 로드

(72) 발명자

페더슨, 브래드, 제이.

미국 플로리다 33309, 포트 로더데일, 웨스트 싸이프레스 크릭 로드 851, 사이트릭스 시스템스, 인크. 내

썬다라잔 프라바카

미국 캘리포니아 95054, 썬타 클라라, 그레이트 어메리카 파크웨이 4988, 사이트릭스 실리콘 밸리 내

(뒷면에 계속)

(74) 대리인

신영무

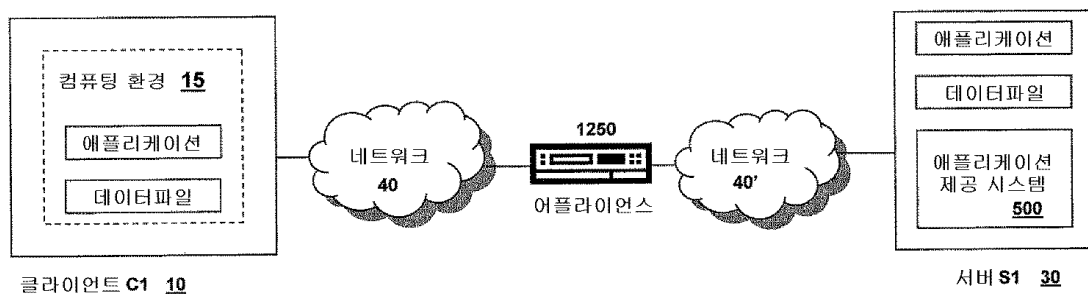
전체 청구항 수 : 총 60 항

(54) 원격 사용자로의 컴퓨팅 환경 제공을 가속하기 위한 시스템 및 방법

(57) 요약

본 발명은 원격 클라이언트의 원격 사용자에게 컴퓨팅 환경 제공을 가속화하는 것에 관한 것이다. 컴퓨팅 환경은 애플리케이션과, 이 애플리케이션에 의해서 사용 또는 처리되는 데이터 파일을 포함할 수 있다. 애플리케이션 및 데이터 파일은 클라이언트로부터 원격의 서버를 통해서 저장 또는 제공될 수 있다. 사용자는 원격의 컴퓨터를 통해서 사용자가 애플리케이션을 실행할 수 있도록 서버로부터 컴퓨팅 환경을 요청할 수 있다. 예를 들어, 서버는 원격 클라이언트로 애플리케이션을 스트리밍할 수도 있다. 클라이언트 및 서버는 클라이언트와 서버간의 통신을 가속하는 어플라이언스를 통하여 통신할 수 있다. 예를 들어, 어플라이언스는 원격의 사용자로의 애플리케이션의 스트리밍을 가속화할 수도 있다. 어떤 경우에, 애플리케이션 또는 원격의 사용자는 데이터 파일도 서버로부터 요청할 수 있는데, 어플라이언스는 원격 사용자에게로의 데이터 파일의 제공을 가속화한다. 이로써, 원격의 사용자는 원격 거리에 위치하는 애플리케이션 및 데이터 파일에 대한 액세스를 임의의 네트워크 연결된 디바이스를 통해서 가속화하여 얻는다.

대표도



(72) 발명자

신하 라지브

미국 캘리포니아 95054, 산타 클라라, 그레이트 어
메리카 파크웨이 4988, 싸이트릭스 실리콘 밸리 내

트레더 테리

미국 플로리다 33309, 포트 로더데일, 웨스트 싸이
프레스 크릭 로드 851, 사이트릭스 시스템스,
인크. 내

특허청구의 범위

청구항 1

원격 클라이언트로 컴퓨팅 환경의 제공(delivery)을 가속화하는 방법에 있어서, 상기 방법은,

(a) 상기 서버에서, 애플리케이션을 실행하라는 원격 클라이언트의 요청을 수신하는 단계 - 상기 원격 클라이언트와 상기 서버는 어플라이언스를 통해 통신함-와,

(b) 실행대상 애플리케이션을 상기 원격 클라이언트로 상기 서버에서 스트리밍하는 단계와,

(c) 상기 스트리밍된 애플리케이션이 액세스하기 위해 상기 원격 클라이언트에 의해 요청된 데이터 파일을 상기 서버에서 전송하는 단계와,

(d) 상기 데이터 파일을 상기 원격 클라이언트에 전송하는 것을 상기 어플라이언스에서 가속화하는 단계를 포함하는 방법.

청구항 2

제1항에 있어서, 상기 데이터 파일은 상기 애플리케이션에 의해 이용될 수 있는 포맷인 방법.

청구항 3

제1항에 있어서, 상기 애플리케이션은 상기 데이터 파일의 파일 유형에 연관되어 있는 방법.

청구항 4

제1항에 있어서, 상기 원격 클라이언트에 상기 애플리케이션을 스트리밍하는 것을 상기 어플라이언스에서 가속화하는 단계를 포함하는 방법.

청구항 5

제1항에 있어서, 상기 어플라이언스에서, 압축, 전송 제어 프로토콜(Transmission Control Protocol) 다중화, 전송 제어 프로토콜 버퍼링 및 캐싱중 하나의 가속화 기법을 수행함으로써 상기 데이터 파일의 전송 또는 상기 애플리케이션의 스트리밍을 가속화하는 단계를 포함하는 방법.

청구항 6

제1항에 있어서, 상기 원격 클라이언트와 상기 서버간의 통신을 상기 원격 클라이언트상의 가속화 프로그램에서 가속화하는 단계를 포함하는 방법.

청구항 7

제1항에 있어서, 상기 원격 클라이언트와의 가상 사설망 연결 또는 보안 소켓 계층(Secure Socket Layer:SSL) 연결중 하나를 상기 어플라이언스에서 개설하는 단계를 포함하는 방법.

청구항 8

제1항에 있어서, 상기 원격 클라이언트로부터 상기 서버와의 연결 및 세션중 하나를 개설하려는 요청에 따라 상기 원격 클라이언트로 가속화 프로그램을 상기 어플라이언스에서 전송하는 단계를 포함하는 방법.

청구항 9

제8항에 있어서, 상기 어플라이언스로부터 상기 가속화 프로그램이 수신되면 상기 가속화 프로그램을 상기 원격 클라이언트에서 자동 설치 및 실행하는 단계를 포함하는 방법.

청구항 10

제9항에 있어서, 상기 애플리케이션 및 상기 서버중 하나에 투명하게(transparently) 상기 가속화 프로그램을 상기 원격 클라이언트에서 실행하는 단계를 포함하는 방법.

청구항 11

제1항에 있어서, 상기 애플리케이션이 가속화될 수 있는지 상기 어플라이언스에서 판단하고, 상기 판단에 응하여 상기 어플라이언스에서 상기 원격 클라이언트에 가속화 프로그램을 전송하는 단계를 포함하는 방법.

청구항 12

제1항에 있어서, 상기 원격 클라이언트상의 가속화 프로그램에서, 압축, 전송 제어 프로토콜 다중화, 전송 제어 프로토콜 버퍼링 및 캐싱중 하나의 가속화 기법을 수행하는 단계를 포함하는 방법.

청구항 13

제1항에 있어서, 상기 어플라이언스에서 상기 데이터 파일을 캐싱하는 단계를 포함하는 방법.

청구항 14

제13항에 있어서, 상기 원격 클라이언트로부터의 상기 데이터 파일에 대한 요청을 상기 어플라이언스에서 인터셉트하고 상기 요청에 응답하여 상기 캐싱된 데이터 파일을 상기 원격 클라이언트에 전송하는 단계를 포함하는 방법.

청구항 15

제1항에 있어서, 상기 어플라이언스에서 상기 서버로부터 수신된 상기 데이터 파일을 압축하고 상기 압축된 데이터 파일을 상기 원격 클라이언트에 전송하는 단계를 포함하는 방법.

청구항 16

제1항에 있어서, 상기 어플라이언스에서 상기 복수의 클라이언트에 의해 이용된 풀링된 전송 계층 연결(pooled transport layer connection)을 통해 상기 스트리밍된 애플리케이션 및 상기 데이터 파일중 하나를 상기 서버로부터 수신하는 단계를 포함하는 방법.

청구항 17

제16항에 있어서, 상기 어플라이언스에서 상기 풀링된 전송 계층 연결을 통해 상기 스트리밍된 애플리케이션 및 상기 데이터 파일중 하나를 전송하는 네트워크 패킷을 다중화하는 단계를 포함하는 방법.

청구항 18

제1항에 있어서, 상기 원격 클라이언트는 제1 네트워크에 위치하고 상기 서버 및 상기 어플라이언스중 하나는 제2 네트워크에 위치하는 방법.

청구항 19

제1항에 있어서, 상기 원격 클라이언트는 지점에 위치하고 상기 서버 및 상기 어플라이언스중 하나는 데이터 센터에 위치하는 방법.

청구항 20

원격 클라이언트에 컴퓨팅 환경의 제공을 가속화하기 위한 시스템에 있어서, 상기 시스템은,

하나 이상의 클라이언트 및 하나 이상의 서버간의 통신을 가속화하기 위한 어플라이언스와,

애플리케이션을 실행하라는 원격 클라이언트로부터의 요청을 수신하는 서버 - 상기 서버는 상기 요청에 응하여 실행대상 애플리케이션을 상기 어플라이언스를 통해 상기 원격 클라이언트에 스트리밍함- 를 포함하고,

상기 서버는 상기 스트리밍된 애플리케이션에서 액세스하기 위해 상기 원격 클라이언트에 의해 요청된 데이터 파일을 전송하고, 상기 어플라이언스는 상기 원격 클라이언트에 상기 데이터 파일의 전송을 가속화하는 시스템.

청구항 21

제20항에 있어서, 상기 데이터 파일은 상기 애플리케이션에 의해 이용가능한 포맷으로 된 시스템.

청구항 22

제21항에 있어서, 상기 애플리케이션은 상기 데이터 파일의 파일 유형에 연관되어 있는 시스템.

청구항 23

제20항에 있어서, 상기 어플라이언스는 상기 원격 클라이언트로의 상기 애플리케이션 스트리밍을 가속화하는 시스템.

청구항 24

제20항에 있어서, 상기 원격 클라이언트와 상기 서버간의 통신을 가속화하는 상기 원격 클라이언트상의 가속화 프로그램을 포함하는 시스템.

청구항 25

제20항에 있어서, 상기 어플라이언스는 상기 원격 클라이언트와 가상사설망 연결 및 보안 소켓층(Secure Socket Layer:SSL) 연결중 하나를 개설하는 시스템.

청구항 26

제20항에 있어서, 상기 어플라이언스는 상기 서버와 연결 및 세션중 하나를 개설하려는 상기 원격 클라이언트로부터의 요청에 따라 상기 원격 클라이언트에 가속화 프로그램을 전송하는 시스템.

청구항 27

제26항에 있어서, 상기 원격 클라이언트는 상기 어플라이언스로부터 상기 가속화 프로그램이 수신되면 상기 프로그램을 자동 설치하고 실행하는 시스템.

청구항 28

제26항에 있어서, 상기 원격 클라이언트상의 상기 가속화 프로그램은 압축, 전송 제어 프로토콜 다중화, 전송 제어 프로토콜 버퍼링 및 캐싱중 하나의 가속화 기법을 수행하는 시스템.

청구항 29

제26항에 있어서, 상기 원격 클라이언트는 상기 애플리케이션 및 상기 서버중 하나에 투명하게 상기 가속화 프로그램을 실행하는 시스템.

청구항 30

제20항에 있어서, 상기 어플라이언스는 상기 애플리케이션이 가속화될 수 있는지 판단하고 상기 판단에 응하여 상기 원격 클라이언트에 가속화 프로그램을 전송하는 시스템.

청구항 31

제20항에 있어서, 상기 어플라이언스는 압축, 전송 제어 프로토콜 다중화, 전송 제어 프로토콜 버퍼링 및 캐싱중 하나의 가속화 기법을 수행함으로써 상기 데이터 파일의 전송 및 상기 애플리케이션의 스트리밍중 하나를 가속화하는 시스템.

청구항 32

제20항에 있어서, 상기 어플라이언스는 상기 데이터 파일을 캐싱하기 위한 캐시를 포함하는 시스템.

청구항 33

제32항에 있어서, 상기 어플라이언스는 상기 데이터 파일에 대한 요청을 인터셉트하고 상기 요청에 응하여 상기 캐싱된 데이터 파일을 상기 원격 클라이언트에 전송하는 시스템.

청구항 34

제23항에 있어서, 상기 어플라이언스는 상기 서버로부터 수신된 상기 데이터 파일을 압축하고 상기 원격 클라이언트에 상기 압축된 데이터 파일을 전송하는 시스템.

청구항 35

제23항에 있어서, 상기 어플라이언스는 상기 복수의 클라이언트에 의해 이용된 풀링된 전송 계층 연결을 통해 상기 스트리밍된 애플리케이션 및 상기 데이터 파일중 하나를 상기 서버로부터 수신하는 시스템.

청구항 36

제32항에 있어서, 상기 어플라이언스는 상기 풀링된 전송 계층 연결을 통해 상기 스트리밍된 애플리케이션 및 상기 데이터 파일중 하나를 전송하는 네트워크 패킷을 다중화하는 시스템.

청구항 37

제23항에 있어서, 상기 원격 클라이언트는 제1 네트워크에 위치하고 상기 서버 및 상기 어플라이언스중 하나는 제2 네트워크에 위치하는 시스템.

청구항 38

제23항에 있어서, 상기 원격 클라이언트는 지점에 위치하고 상기 서버 및 상기 어플라이언스중 하나는 데이터 센터에 위치하는 시스템.

청구항 39

클라이언트 에이전트 및 어플라이언스를 통해 원격 클라이언트에 컴퓨팅 환경의 제공을 가속화하기 위한 방법이 있어서, 상기 방법은,

- (a) 원격 클라이언트에 의해 요청된 실행대상 애플리케이션을 어플라이언스를 통해 클라이언트 에이전트에 스트리밍하는 단계와,
- (b) 상기 원격 클라이언트에 의해 요청된 데이터 파일 - 상기 데이터 파일은 상기 스트리밍된 애플리케이션에 의해 액세스됨- 을 상기 어플라이언스를 통해 전송하는 단계와,
- (c) 상기 원격 클라이언트로 상기 데이터 파일의 제공을 상기 클라이언트 에이전트 및 상기 어플라이언스중 하나에서 가속화하는 단계

를 포함하는 방법.

청구항 40

제39항에 있어서, 상기 원격 클라이언트로 상기 스트리밍된 애플리케이션의 제공을 상기 클라이언트 에이전트 및 상기 어플라이언스중 하나에서 가속화하는 단계를 포함하는 방법.

청구항 41

제39항에 있어서, 상기 클라이언트 에이전트 및 상기 어플라이언스중 하나에서, 압축, 전송 제어 프로토콜 다중화, 전송 제어 프로토콜 버퍼링 및 캐싱중 하나의 가속화 기법을 수행함으로써 상기 데이터 파일 및 상기 스트리밍된 애플리케이션중 하나의 제공을 가속화하는 단계를 포함하는 방법.

청구항 42

상기 클라이언트 에이전트 및 상기 어플라이언스를 통해, 상기 원격 클라이언트에 대한 가상사설망 연결 및 보안소켓층(SSL) 연결중 하나를 개설하는 단계를 포함하는 방법.

청구항 43

제39항에 있어서, 상기 클라이언트 에이전트 및 상기 어플라이언스중 하나에서 상기 데이터 파일을 캐싱하는 단계를 포함하는 방법.

청구항 44

제39항에 있어서, 상기 클라이언트 에이전트 및 상기 어플라이언스중 하나에서 상기 원격 클라이언트로부터의 상기 데이터 파일에 대한 요청을 인터셉트하고 상기 요청에 응하여 상기 캐싱된 데이터 파일을 제공하는 단계를 포함하는 방법.

청구항 45

제39항에 있어서, 상기 어플라이언스에서 서버로부터 수신된 상기 데이터 파일을 압축하고 상기 압축된 데이터 파일을 상기 클라이언트 에이전트에 전송하는 단계를 포함하는 방법.

청구항 46

제45항에 있어서, 상기 클라이언트 에이전트에서 상기 어플라이언스로부터 수신된 상기 압축된 데이터 파일을 압축해제하고 상기 데이터 파일을 상기 원격 클라이언트상에서 실행하는 상기 스트리밍된 애플리케이션에 제공하는 단계를 포함하는 방법.

청구항 47

제39항에 있어서, 상기 클라이언트 에이전트에서 상기 어플라이언스와 서버간의 폴링된 전송 계층 연결을 통해 상기 스트리밍된 애플리케이션 및 상기 데이터 파일중 하나를 수신하는 단계를 포함하는 방법.

청구항 48

제47항에 있어서, 상기 어플라이언스에서 상기 폴링된 전송 계층 연결을 통해 상기 스트리밍된 애플리케이션 및 상기 데이터 파일중 하나를 전송하는 네트워크 패킷을 다중화하는 단계를 포함하는 방법.

청구항 49

제39항에 있어서, 상기 클라이언트 에이전트와 상기 어플라이언스간에 개설된 제1 전송계층 연결 및 상기 어플라이언스와 서버간에 개설된 제2 전송계층 연결간에 상기 스트리밍된 애플리케이션 및 상기 데이터 파일중 하나를 전송하는 단계를 포함하는 방법.

청구항 50

클라이언트 에이전트를 통해 원격 클라이언트로 컴퓨팅 환경의 제공을 가속화하는 방법에 있어서, 상기 방법은,

(a) 클라이언트 에이전트에서, 실행요청된 스트리밍된 애플리케이션을 서버로부터 수신하는 단계와,

(b) 상기 클라이언트 에이전트에서, 상기 스트리밍된 애플리케이션에 의해 액세스되는 데이터 파일을 상기 서버로부터 요청하는 단계와,

(c) 상기 클라이언트 에이전트에서, 상기 원격 클라이언트의 상기 스트리밍된 애플리케이션으로 상기 데이터 파일의 제공을 가속화하는 단계

를 포함하는 방법.

청구항 51

제50항에 있어서, 상기 클라이언트 에이전트에서, 상기 원격 클라이언트로 상기 스트리밍된 애플리케이션의 제공을 가속화하는 단계를 포함하는 방법.

청구항 52

제50항에 있어서, 상기 클라이언트 에이전트에서, 압축, 전송 제어 프로토콜 다중화, 전송 제어 프로토콜 버퍼링 및 캐싱중 하나의 가속화 기법을 수행함으로써 상기 데이터 파일 및 상기 스트리밍된 애플리케이션중 하나의 제공을 가속화하는 단계를 포함하는 방법.

청구항 53

제50항에 있어서, 상기 서버로 상기 원격 클라이언트에 대한 가상사설망 연결 및 보안소켓층(SSL) 연결중 하나를 상기 클라이언트 에이전트를 통해 개설하는 단계를 포함하는 방법.

청구항 54

제50항에 있어서, 상기 클라이언트 에이전트에서 상기 데이터 파일 및 상기 스트리밍된 애플리케이션의 일부를 캐싱하는 단계를 포함하는 방법.

청구항 55

제54항에 있어서, 상기 클라이언트 에이전트에서 상기 데이터 파일에 대한 상기 원격 클라이언트의 요청을 인터셉트하고 상기 요청에 응하여 상기 캐싱된 데이터 파일의 일부를 제공하는 단계를 포함하는 방법.

청구항 56

제54항에 있어서, 상기 클라이언트 에이전트에서 상기 스트리밍된 애플리케이션에 대한 사용자의 요청을 인터셉트하고 상기 요청에 응하여 상기 스트리밍된 애플리케이션의 일부를 제공하는 단계를 포함하는 방법.

청구항 57

제50항에 있어서, 상기 클라이언트 에이전트에서 어플라이언스 및 상기 서버중 하나로부터 수신된 압축된 데이터 파일을 수신하고 상기 원격 클라이언트상에서 실행하는 상기 스트리밍된 애플리케이션으로 상기 데이터를 제공하는 단계를 포함하는 방법.

청구항 58

제50항에 있어서, 상기 클라이언트 에이전트에서 풀링된 전송계층 연결을 통해 상기 스트리밍된 애플리케이션 및 상기 데이터 파일중 하나를 수신하는 단계를 포함하는 방법.

청구항 59

제58항에 있어서, 상기 클라이언트 에이전트에서 상기 풀링된 전송계층 연결을 통해 상기 스트리밍된 애플리케이션 및 상기 데이터 파일중 하나에 대하여 수신된 네트워크 패킷을 역다중화(demultiplexing)하는 단계를 포함하는 방법.

청구항 60

제50항에 있어서, 상기 클라이언트 에이전트와 상기 어플라이언스간에 개설된 제1 전송계층 연결 및 상기 어플라이언스와 서버간에 개설된 제2 전송계층 연결간에 상기 스트리밍된 애플리케이션 및 상기 데이터 파일중 하나를 수신하는 단계를 포함하는 방법.

명세서

기술 분야

- <1> 본 출원은 2006년 4월 12일 "Systems and Methods for Accelerating Delivery of a Computing Environment to a Remote User" 제하로 출원된 미국 특허출원번호 60/744,720호를 우선권으로 주장하며, 본 명세서에 그 전문이 참조로서 포함된다.
- <2> 본 발명은 애플리케이션 및 데이터 파일을 포함하는 컴퓨팅 환경을 서버에서 원격에 있는 클라이언트의 원격 사용자에게 제공하는 것을 가속하기 위한 시스템 및 방법에 관한 것이다.

배경 기술

- <3> 기업 환경을 관리 및 처리하는 것은 시간, 돈 그리고 자원을 소모한다. 많은 경우에서, 이것은 애플리케이션 및 데이터 처리 프로세스가 분산되고 노동집약적이기 때문이다. 예를 들어, 관리자는 좀더 많은 저장장치를 제공하거나 회사의 데이터를 백업하거나 증가한 회사 데이터를 처리하기 위해 서버를 업데이트하는데 많은 시간을 사용한다. 또한, 증가하는 데이터를 처리하기 위한 새로운 서버들을 준비하고 생성할 필요가 있을 수 있다. 또한, 특정 사용자 애플리케이션을 제공하기 위해 서버를 업데이트하거나 준비하는데 시간을 사용할 수 있다. 또한, 회사 데이터의 대부분은 회사 데이터 센터의 외부에 존재할 수 있다. 예를 들어, 회사 문서들, 파일들 및 데이터들은 데이터 센터에서 원격의 다수의 컴퓨터들상에 존재하거나 분산될 수 있다.

<4> 회사 데이터 및 애플리케이션을 관리하는데 필요한 시간, 돈 그리고 자원들을 감소하기 위해, 많은 회사들은 통합된 그리고 집중화된 서버들, 회사 데이터 및 애플리케이션들을 갖는다. 통합화 및 집중화는 어느 정도의 경비를 감소시키고 일정한 이점을 갖지만, 집중화된 데이터 및 애플리케이션은 데이터 및 애플리케이션으로의 액세스를 제공하는데 추가적인 요청들을 초래한다. 이러한 요청중 하나는 광역네트워크(WAN) 연결을 통해 파일에 액세스하려는 원격 사용자를 포함한다. 예를 들어, 회사 데이터 센터로 LAN 보다 훨씬 느리게 동작하는 네트워크를 통해 연결하는 지사에서 원격 사용자가 회사 데이터 센터에 저장된 마이크로소프트 오피스 문서를 WAN을 통해 열려고 할 수 있다. 이러한 네트워크를 통한 원격 사용자의 액세스는 WAN의 대기정도(latency), 신뢰도 그리고 대역폭으로 인해 지연될 수 있다. 이러한 지연은 크기가 큰 파일일수록 더욱 커진다. 또한, 원격 사용자와 회사 데이터 센터간의 거리가 멀수록, 파일 액세스의 네트워크 지연의 주기와 길이 또한 증가될 것이다. 가상 사설 네트워크(virtual private network)를 추가하는 것, 보안 그리고 WAN 상의 다른 네트워크 계층들 또한 원격 사용자가 이용가능한 대역폭을 줄이고 파일로의 액세스 지연을 증가시킨다. 지사의 낮은 속도 및 대역폭은 원격 파일들로 액세스하는데 용인될 수 없는 지연들을 야기할 수 있다. 원격 파일 액세스에서의 지연들을 방지하기 위해, 원격 사용자는 집중화 작업의 목적에 반하게 파일들을 복사해서 사용할 수 있다. 또한, WAN 연결은 LAN 연결보다 신뢰도가 떨어질 수 있으며, 패킷 손실 및 네트워크 연결불능을 초래할 수 있다. WAN 차단은 문서들을 저장 또는 여는 것과 같은 파일 작업중에 지연을 야기할 수 있다.

<5> 따라서, 애플리케이션 및 데이터 파일들을 원격 사용자에게 제공하는 것을 가속하는 것을 포함해서, 집중화된 애플리케이션 및 데이터 파일들로의 원격 사용자의 액세스를 개선하기 위한 시스템 및 방법이 필요하다.

발명의 상세한 설명

<6> 본 발명은 애플리케이션 및 데이터 파일의 컴퓨팅 환경을 원격 사용자에게 제공하는 것을 가속하는 시스템 및 방법에 관한 것이다. 애플리케이션 및 데이터 파일은 클라이언트와 원격의 서버에 저장되거나 서버를 통해 제공된다. 예를 들어 원격 직원과 같은 사용자는 지사에서 애플리케이션 및 데이터 파일을 갖지 않는 컴퓨터를 사용할 수 있다. 사용자는 원격 클라이언트상에서 이용할 수 없는 워드 처리 애플리케이션으로 회사 문서를 편집하길 원할 수 있다. 사용자는 원격 클라이언트를 통해 원하는 애플리케이션 실행을 제공하는 서버로 컴퓨팅 환경을 요청할 수 있다. 예를 들어 서버는 애플리케이션을 원격 클라이언트로 스트리밍할 수 있다. 원격 클라이언트 및 서버는 원격 클라이언트 및 서버 사이의 통신을 가속시키는 어플라이언스(appliance)를 통해 통신할 수 있다. 예를 들어 어플라이언스는 원격 사용자로의 애플리케이션의 스트리밍을 가속시킬 수 있다. 일정 경우에는, 애플리케이션 또는 원격 사용자는 또한 서버로부터의 데이터 파일을 요청할 수 있고, 어플라이언스는 원격 사용자로의 데이터 파일 제공을 가속한다. 이와 같이, 본 발명은 원격 위치에 있는 사용자에게 사용자로부터 임의의 네트워크를 통해 연결된 원격의 위치에 있는 애플리케이션 및 데이터 파일에 대한 가속된 액세스를 제공한다.

<7> 일 측면에서, 본 발명은 원격 위치에 있는 클라이언트의 사용자에게 애플리케이션 및 데이터 파일의 컴퓨팅 환경을 제공하는 것을 가속시키는 방법에 관한 것이다. 상기 방법은 서버가 원격 클라이언트로부터 애플리케이션 실행을 위한 요청을 수신하는 단계를 포함한다. 원격 클라이언트 및 서버는 어플라이언스를 통해 통신한다. 상기 방법은 또한 서버가 실행을 위한 애플리케이션을 원격 클라이언트에 스트리밍하는 단계를 포함한다. 클라이언트는 상기 애플리케이션에 의해 이용될 수 있는 데이터 파일에 대한 요청을 서버에 전송하고, 어플라이언스는 원격 클라이언트로의 데이터 파일 전송을 가속시킨다.

<8> 본 발명의 일 실시예에서, 상기 방법은 어플라이언스가 원격 클라이언트로의 애플리케이션 스트리밍을 가속하는 단계를 포함한다. 다른 실시예에서, 어플라이언스는 이하의 가속 기술중 하나를 실행하여 데이터 파일의 전송 또는 애플리케이션의 스트리밍을 가속한다: 1) 압축(compression) 2) 압축풀기(decompression) 3) TCP(Transmission Control Protocol) 풀링(pooling) 4) TCP 다중화(multiplexing) 5) TCP 버퍼링(buffering) 그리고 6) 캐싱(caching). 다른 실시예에서, 상기 방법은 원격 클라이언트상의 가속 프로그램이 원격 클라이언트 및 서버 사이의 통신을 가속하는 단계를 포함한다. 일 실시예에서, 어플라이언스는 가상 사설 네트워크 연결 또는 원격 클라이언트와의 SSL(Secure Socket Layer) 연결을 구축한다. 다른 실시예에서, 상기 방법은 어플라이언스가 원격 클라이언트 및 서버 사이의 전송 계층(transport layer) 연결을 통해 통신되는 네트워크 패킷의 페이로드(payload)를 가속하는 단계를 포함한다.

<9> 본 발명의 일 실시예에서, 상기 방법은 서버와의 연결 또는 세션을 구축하기 위한 원격 클라이언트로부터의 요청에 따라 어플라이언스가 원격 클라이언트에 가속 프로그램을 전송하는 단계를 포함한다. 일 실시예에서, 원격 클라이언트는 어플라이언스로부터 가속 프로그램을 수신하면 자동적으로 설치하고 실행한다. 다른 실시예에

서, 상기 방법은 원격 클라이언트상의 가속 프로그램이 이하의 가속 기술중 하나를 수행하는 단계를 포함한다: 1) 압축 2) 압축풀기 3) TCP 풀링 4) TCP 다중화 5) TCP 버퍼링 6) 캐싱. 일 실시예에서, 원격 클라이언트는 애플리케이션 또는 서버와 무관하게(transparently) 가속 프로그램을 실행한다.

<10> 본 발명의 일 실시예에서, 상기 방법은 어플라이언스가 애플리케이션이 가속될 수 있는지를 판단하고, 상기 판단에 대응하여 원격 클라이언트에 가속 프로그램을 전송하는 단계를 포함한다. 다른 실시예에서, 어플라이언스는 데이터 파일을 캐싱한다. 일 실시예에서, 어플라이언스는 데이터 파일에 대한 요청을 인터셉트하고 상기 요청에 대응하여 캐싱된 데이터 파일을 원격 클라이언트로 전송한다.

<11> 다른 측면에서, 본 발명은 원격 위치에 있는 클라이언트의 사용자에게 애플리케이션 및 데이터 파일의 컴퓨팅 환경을 제공하는 것을 가속시키는 시스템에 관한 것이다. 상기 시스템은 하나 이상의 클라이언트와 하나 이상의 서버 사이의 통신을 가속하기 위한 어플라이언스를 포함한다. 상기 시스템은 또한 애플리케이션 실행을 위한 요청을 원격 클라이언트로부터 수신하는 서버를 포함한다. 원격 클라이언트 및 서버는 어플라이언스를 통해 통신한다. 서버는 실행을 위한 애플리케이션을 원격 클라이언트로 스트리밍한다. 클라이언트는 애플리케이션에 의해 사용될 수 있는 데이터 파일에 대한 요청을 서버에 전송하고, 어플라이언스는 원격 클라이언트의 데이터 파일 전송을 가속한다.

<12> 본 발명의 일 실시예에서, 어플라이언스는 원격 클라이언트의 애플리케이션 스트리밍을 가속시킨다. 일 실시예에서, 어플라이언스는 이하의 가속 기술중 하나를 수행하여 데이터 파일의 전송 또는 애플리케이션의 스트리밍을 가속시킨다: 1) 압축 2) 압축풀기 3) TCP 풀링 4) TCP 다중화 5) TCP 버퍼링 6) 캐싱. 다른 실시예에서, 상기 시스템은 원격 클라이언트와 서버 사이의 통신을 가속시키는, 원격 클라이언트상의 가속 프로그램을 포함한다. 일 실시예에서, 어플라이언스는 가상 사설 네트워크 연결 또는 원격 클라이언트와의 SSL(Secure Socket Layer) 연결을 구축한다.

<13> 본 발명에 따른 시스템의 일 실시예에서, 어플라이언스는 원격 클라이언트 및 서버 사이의 전송 계층 연결을 통해 통신되는 네트워크 패킷의 페이로드를 가속시킨다. 일 실시예에서, 어플라이언스는 서버와의 연결 또는 세션을 구축하기 위한 원격 클라이언트로부터의 요청에 따라 원격 클라이언트에 가속 프로그램을 전송한다. 다른 실시예에서, 원격 클라이언트는 어플라이언스로부터 가속 프로그램을 수신하면 자동적으로 설치하고 실행한다. 원격 클라이언트상의 가속 프로그램은 이하의 가속 기술중 하나를 수행한다: 1) 압축 2) 압축풀기 3) TCP 풀링 4) TCP 다중화 5) TCP 버퍼링 6) 캐싱. 일 실시예에서, 원격 클라이언트는 애플리케이션 또는 서버와 무관하게(transparently) 가속 프로그램을 실행한다.

<14> 본 발명에 따른 시스템의 다른 실시예에서, 어플라이언스는 애플리케이션이 가속될 수 있는지를 판단하고, 상기 판단에 대응하여 원격 클라이언트에 가속 프로그램을 전송한다. 일 실시예에서, 어플라이언스는 데이터 파일을 캐싱하기 위한 캐시(cache)를 포함한다. 일 실시예에서, 어플라이언스는 데이터 파일에 대한 요청을 인터셉트하고 상기 요청에 대응하여 캐싱된 데이터 파일을 원격 클라이언트로 전송한다.

실시예

<86> 이하의 다양한 실시예들의 설명을 이해하기에 다음의 각 섹션들에 대한 설명이 도움이 될 것이다.

<87> 섹션 A 는 본 명세서에 기재된 실시예들을 구현하기에 유용한 네트워크 환경 및 컴퓨팅 환경을 설명한다.

<88> 섹션 B 는 원격 사용자에게 컴퓨팅 환경을 제공하기 위한 시스템 및 방법의 실시예들을 설명한다.

<89> 섹션 C 는 클라이언트와 서버 사이의 통신을 가속하는 시스템 및 방법의 실시예들을 설명한다.

<90> 섹션 D 는 섹션 B 및 C 에서 설명된 시스템 및 방법을 이용하여 원격 사용자로의 컴퓨팅 환경 제공을 가속하는 대표적인 실시예를 설명한다.

<91> A. 네트워크 및 컴퓨팅 환경

<92> 본 발명에 따른 시스템 및 방법에 대한 실시예를 설명하기에 앞서, 실시예들이 사용되는 네트워크 및 컴퓨팅 환경에 대한 설명을 한다. 도 1a를 참조하면, 네트워크 환경(5)이 도시되어 있다. 간단히 설명하기 위해, 네트워크 환경(5)은 하나 이상의 네트워크(40, 40')를 통해 하나 이상의 서버(30-30')(서버들(30) 또는 원격 머신(30)으로도 언급됨)와 통신하는 하나 이상의 클라이언트(10-10')(클라이언트들(10) 또는 로컬 머신(10)으로도 언급됨)를 포함한다. 일 실시예에서, 클라이언트(10)는 어플라이언스(1250)를 통해 서버(30)와 통신한다.

<93> 도 1a에서는 클라이언트(10-10')와 서버(30-30') 사이에 네트워크(40)와 네트워크(40')가 있지만, 클라이언트

(10-10") 와 서버(30-30")는 동일한 네트워크(40)상에 존재할 수 있다. 네트워크(40)와 네트워크(40')는 동일한 유형의 네트워크이거나 또는 다른 유형의 네트워크일 수 있다. 네트워크(40) 및/또는 네트워크(40')는 회사 인트라넷과 같은 LAN(local area network), MAN(metropolitan area network), 또는 인터넷 또는 월드 와이드 웹과 같은 WAN(wide area network)이 될 수 있다. 일 실시예에서, 네트워크(40')는 사설 네트워크(private network)이고 네트워크(40)는 공중 네트워크(public network)일 수 있다. 일 실시예에서, 네트워크(40)는 사설 네트워크이고 네트워크(40')는 공중 네트워크일 수 있다. 다른 실시예에서, 네트워크(40 및 40')는 둘 다 사설 네트워크일 수 있다. 일 실시예에서, 클라이언트(10-10")는 회사의 데이터 센터에 위치한 서버(30-30")에 네트워크(40)의 WAN 연결을 통해 통신하는 회사의 지사에 위치될 수 있다.

<94> 네트워크(40) 및/또는 네트워크(40')는 임의의 네트워크 유형 및/또는 형식일 수 있으며 다음 중 어느 하나를 포함할 수 있다: 포인트 투 포인트 네트워크(point to point network), 브로드캐스트 네트워크(broadcast network), WAN, LAN, 전화망, 데이터 통신 네트워크, 컴퓨터 네트워크, ATM(Asynchronous Transfer Mode) 네트워크, SONET(Synchronous Optical Network) 네트워크, SDH(Synchronous Digital Hierarchy) 네트워크, 무선 및 유선 네트워크. 네트워크(40) 및/또는 네트워크(40')의 토폴로지는 버스, 스타, 또는 링 네트워크 토폴로지일 수 있다. 네트워크(40) 및/또는 네트워크(40')와 네트워크 토폴로지는 본 명세서에 설명된 작업들을 지원할 수 있는 임의의 네트워크 또는 네트워크 토폴로지일 수 있다.

<95> 도 1a에 도시된 바와 같이, 어플라이언스(1250)(인터페이스 유닛(1250)으로도 언급됨)가 네트워크(40)와 네트워크(40')사이에 있다. 일 실시예에서, 어플라이언스(1250)는 네트워크(40)상에 위치될 수 있다. 예를 들어, 회사의 지사는 지사에서 어플라이언스(1250)를 사용할 수 있다. 다른 실시예에서, 어플라이언스(1250)는 네트워크(40')상에 위치될 수 있다. 예를 들어, 어플라이언스(1250)는 회사 데이터 센터에 위치될 수 있다. 또 다른 실시예에서, 다수의 어플라이언스(1250)가 네트워크(40)상에서 사용될 수 있다. 일 실시예에서 다수의 어플라이언스(1250)는 네트워크(40')상에서 사용될 수 있다. 일 실시예에서, 제1 어플라이언스(1250)는 제2 어플라이언스(1250')와 통신한다. 다른 실시예에서, 어플라이언스(1250)는 동일 또는 다른 네트워크(40, 40')상의 클라이언트(10-10') 또는 서버(30-30")의 일부일 수 있다. 하나 이상의 어플라이언스(1250)가 네트워크내에서 또는 클라이언트(10-10")와 서버(30-30") 사의 네트워크 통신 경로내에서 임의의 지점에 위치될 수 있다.

<96> 일 실시예에서, 시스템은 로컬 머신(10)을 대신하여 애플리케이션을 실행할 수 있는 다수의, 논리적으로 그룹화된(logically-grouped) 원격 머신(30)을 포함한다. 이 실시예에서, 원격 머신의 논리 그룹은 서버 팜(server farm)(38) 또는 팜(38)이라 한다. 이 실시 예 중 일정 경우에서, 원격 머신(30)은 지리적으로 분산 배치될 수 있다. 팜(38)은 단일 엔티티로서 관리될 수 있다.

<97> 각 팜(38)내의 원격 머신(30)은 이종(heterogeneous)일 수 있다. 즉, 하나 이상의 원격 머신(30)은 일정 작업 시스템 플랫폼(예를 들어, 워싱턴, 레드몬드의 마이크로소프트사의 WINDOWS NT)에 따라 동작할 수 있으면서 또 다른 하나 이상의 원격 머신(30)은 다른 작업 시스템 플랫폼(예를 들어 Unix 또는 Linux)에 따라 동작할 수 있다. 원격 머신(30)은 팜(38)내의 다른 원격 머신(30)과 물리적으로 근접할 필요는 없다. 따라서, 팜(38)으로 논리적으로 그룹화된 원격 머신(30)의 그룹은 WAN 연결 또는 MAN 연결을 사용하여 상호연결될 수 있다. 예를 들어, 팜(38)은 다른 대륙 또는 다른 지역, 나라, 주, 도시, 캠퍼스, 또는 방에 물리적으로 위치한 원격 머신(30)을 포함할 수 있다. 팜(38)내의 원격 머신(30) 사이의 데이터 전송 속도는 원격 머신(30)이 LAN 연결 또는 다른 직접 연결 형식을 사용한다면 증가될 수 있다.

<98> 원격 머신(30)은 서버, 파일 서버, 애플리케이션 서버, 또는 원격 머신들로 언급될 수 있다. 일 실시예에서, 원격 머신(30)은 애플리케이션 서버들로서 또는 마스터 애플리케이션 서버로서 기능할 수 있다. 일 실시예에서, 원격 머신(30)은 Active Directory를 포함할 수 있다. 로컬 머신(10)은 또한 클라이언트 노드 또는 종점이라 한다. 일 실시예에서, 로컬 머신(10)은 애플리케이션으로의 액세스를 탐색하는 클라이언트 노드로서 그리고 다른 로컬 머신(10)에 호스팅된 애플리케이션에 대한 액세스를 제공하는 애플리케이션 서버로서 기능할 수 있다.

<99> 일 실시예에서, 로컬 머신(10)은 팜(38)내의 원격 머신(30)중 하나와 직접 통신할 수 있다. 다른 실시예에서, 로컬 머신(10)은 팜(38)내의 원격 머신(30)과 통신하기 위해 프로그램 근접 애플리케이션(program neighborhood application)을 실행한다. 다른 실시예에서, 원격 머신(30)은 마스터 노드의 기능성을 제공한다. 일 실시예에서, 로컬 머신(10)은 네트워크(40)를 통해 팜(38)내의 원격 머신(30)과 통신한다. 네트워크(40)를 통해, 로컬 머신(10)은 팜(38)내의 원격 머신(30, 30', 30", 30'")에 의해 호스팅된 다양한 애플리케이션의 실행을 요청할 수 있고 디스플레이를 위해 애플리케이션 실행 결과의 출력을 수신할 수 있다. 네트워크(40)는 동기식 또는 비

동기식 연결을 포함할 수 있고 LAN, MAN(Medium-Area-Network), 또는 WAN일 수 있다. 또한, 네트워크(40)는 적외선 채널 또는 위성 밴드와 같은 무선 링크를 포함할 수 있다. 일 실시예에서, 오직 마스터 노드만이 요청된 애플리케이션을 호스팅하는 원격 머신(30')과 연관된 어드레스 정보를 식별하고 제공하는데 필요한 기능성을 제공한다.

<100> 일 실시예에서, 로컬 머신(10)은 원격 머신(30'')과 통신한다. 이 실시예에서, 원격 머신(30'')은 웹 서버의 기능성을 제공한다. 다른 예에서, 원격 머신(30'')은 로컬 머신(10)으로부터 요청을 수신하고, 이 요청을 원격 머신(30)에 전달하고 원격 머신(30)으로부터의 요청에 대한 응답으로 로컬 머신(10)에 의한 요청에 응답한다. 이 실시예에서, 원격 머신(30)은 로컬 머신(10)에 이용가능한 애플리케이션 목록(enumeration) 및 애플리케이션 목록에 의해 식별된 애플리케이션을 호스팅하는 원격 머신(30')과 연관된 어드레스 정보를 획득한다. 이 실시예에서, 원격 머신(30'')은 웹 인터페이스를 사용하여 요청에 대한 응답을 로컬 머신(10)에 보낸다. 일 실시예에서, 로컬 머신(10)은 식별된 애플리케이션에 액세스하기 위해 원격 머신(30')과 직접 통신한다. 다른 실시예에서, 로컬 머신(10)은 원격 머신(30')상에서의 애플리케이션 실행으로 인해 생성된 애플리케이션 출력 데이터를 원격 머신(30'')으로부터 수신한다.

<101> 도 1b를 참조하면, 클라이언트상의 컴퓨팅 환경을 제공 및/또는 동작하기 위한 네트워크 환경이 나타나있다. 서버(30)는 하나 이상의 클라이언트에 컴퓨팅 환경 또는 애플리케이션 및 데이터 파일을 제공하기 위한 애플리케이션 제공 시스템(500)을 포함한다. 클라이언트(10)는 데이터 파일을 사용 또는 처리하는 애플리케이션을 실행하기 위한 컴퓨팅 환경(15)을 포함할 수 있다. 클라이언트(10)는 네트워크(40, 40')를 통해 서버(30)와 통신하고 어플라이언스(1250)는 서버(30)로 애플리케이션 및 데이터 파일을 요청하고, 또는 어플라이언스(1250)는 클라이언트(10)로부터 요청을 서버(30)에 전달할 수 있다. 예를 들어, 클라이언트(10)는 국부적으로(locally) 저장되거나 또는 액세스가능한 애플리케이션 및 데이터 파일을 가지고 있지 않을 수 있다. 요청에 응답하여, 서버(30)는 애플리케이션 및 데이터 파일을 클라이언트(10)에 제공한다. 예를 들어, 일 실시예에서, 서버(30)는 클라이언트(10)상의 컴퓨팅 환경(15)내에서 동작되도록 애플리케이션 스트림으로서 애플리케이션을 전송할 수 있다.

<102> 도 1c 및 1d는 클라이언트 컴퓨팅 장치(10) 및 서버 컴퓨팅 장치(30)로서 유용한 범용 컴퓨터(135)의 아키텍처의 실시예를 나타내는 블록도이다. 도 1c 및 1d에 도시된 바와 같이, 각각의 컴퓨터(135)는 주 프로세서 장치(102)와 주 메모리 장치(122)를 포함한다. 각각의 컴퓨터(135)는 또한 하나 이상의 입력/출력 장치(130a-130b)(일반적으로 도면번호 130으로 언급됨), 그리고 주 프로세서 장치(102)와 통신하는 캐시 메모리(140)와 같은 다른 선택적 요소들을 포함할 수 있다.

<103> 주 프로세서 장치(102)는 주 메모리 장치(122)로부터 나온 명령들에 대응하여 이를 처리하는 임의의 논리 회로이다. 많은 실시예들에서, 주 프로세서 장치는 캘리포니아, 마운틴뷰의 인텔사; 일리노이 스텝버그의 모토로사; 캘리포니아, 산타클라라의 트랜스메타사; 뉴욕, 화이트 플레인의 인터내셔널 비지니스 머신사; 캘리포니아, 서니벌의 어드밴스드 마이크로 디바이스사에 의해 제조된 마이크로프로세서 장치에 의해 제공된다.

<104> 주 메모리 장치(122)는 데이터를 저장하고 마이크로프로세서(102)에 의해 직접 액세스될 있는 임의의 저장 위치를 허용할 수 있는 하나 이상의 메모리 칩이 될 수 있다: Static random access memory(SRAM), Burst SRAM 또는 SynchBurst SRAM(BSRAM), Dynamic random access memory(DRAM), Fast Page Mode DRAM(FPM DRAM), Enhanced DRAM(EDRAM), Extended Data Output RAM(EDO RAM), Extended Data Output DRAM(EDO DRAM), Burst Extended Data Output DRAM(BEDO DRAM), synchronous DRAM(SDRAM), JEDEC SRAM, PC100SDRAM, Double Data Rate SDRAM(DDR SDRAM), Enhanced SDRAM(ESDRAM), SyncLink DRAM(SLDRAM), Direct Rambus DRAM(DRDRAM), Ferroelectric RAM(FRAM). 도 1c에 도시된 실시예에서, 프로세서(102)는 시스템 버스(120)를 통해 주 메모리(122)와 통신한다. 도 1b는 프로세서가 메모리포트를 통해 주 메모리(122)와 직접 통신하는 컴퓨터 시스템(135)의 실시예를 도시한다. 예를 들어, 도 1b에서, 주 메모리(122)는 DRDRAM일 수 있다.

<105> 도 1c 및 1d는 주 프로세서(102)가 "백사이드(backside)" 버스로도 언급되는 세컨더리 버스(secondary bus)를 통해 캐시 메모리(140)와 집적 통신하는 실시예를 나타낸다. 다른 실시예에서, 주 프로세서(102)는 시스템 버스(120)를 사용하여 캐시 메모리(140)와 통신한다. 캐시 메모리(140)는 일반적으로 주 메모리(122)보다 빠른 응답시간을 가지며 SRAM, BSRAM 또는 EDRAM으로 제공된다.

<106> 도 1c에 도시된 실시예에서, 프로세서(102)는 로컬 시스템 버스(120)를 통해 다양한 I/O 장치(130)와 통신한다. 주 프로세서 장치(102)를 I/O 장치(130)에 연결시키기 위해 VESA VL 버스, ISA 버스, EISA 버스, MicroChannel Architecture(MCA) 버스, PCI 버스, PCI-X 버스, PCI-Express 버스 또는 NuBus를 포함하는 다양한 버스가 이용

될 수 있다. I/O 장치가 비디오 디스플레이인 실시예에서, 프로세서(102)는 디스플레이와 통신하기 위해 Advanced Graphics Port(AGP)를 이용할 수 있다. 도 1d는 주 프로세서(102)가 HypeTransport, Rapid I/O, 또는 InfiniBand를 통해 I/O 장치(130b)와 직접 통신하는 실시예를 나타내고 있다. 도 1d 역시 로컬 버스 및 직접 통신이 혼합된 실시예를 나타내고 있다: 프로세서(102)는 I/O 장치(130b)와 직접 통신하는 반면 I/O 장치(130a)와는 로컬 상호연결 버스를 사용한다.

<107> 다양한 I/O 장치(130)들이 컴퓨터 시스템(135)내에 존재할 수 있다. 입력장치는 키보드, 마우스, 트랙패드, 트랙볼, 마이크로폰, 드로잉테이블릿을 포함한다. 출력장치는 비디오 디스플레이, 스피커, 잉크젯 프린터, 레이저 프린터 그리고 염료승화프린터(dye-sublimation printer)를 포함한다. 하드 디스크 드라이브, 3.5인치, 5.25인치 또는 쥘 디스크와 같은 플로피 디스크를 수용하는 플로피 디스크 드라이브, CD-ROM 드라이브, CD-R/RW 드라이브, DVD-ROM 드라이브, 다양한 형태의 테이프 드라이브, 그리고 캘리포니아 로스알라미토스 소재의 Twintech Industry사에 의해 제조된 USB Flash Drive 장치들과 같은 USB 저장 장치와 같은, I/O 장치 또한 컴퓨터 시스템(135)을 위해 대용량 저장소를 제공할 수 있다.

<108> 다른 실시예에서, I/O 장치(130)는 시스템 버스(120)와 외부 통신 버스(USB 버스, Apple Desktop 버스, RS-132 직렬 연결, SCSI 버스, FireWire 버스, FireWire 800 버스, Ethernet 버스, AppleTalk 버스, Gigabit Ethernet 버스, Asynchronous Transfer Mode 버스, HIPPI 버스, Super HIPPI 버스, SerialPlus 버스, SCI/LAMP 버스, FibreChannel 버스, Serial Attached small computer system interface 버스) 사이의 브리지가 될 수 있다.

<109> 도 1c 및 1d에 도시된 종류의 범용 컴퓨터는 일반적으로 작업 스케줄 및 시스템 자원으로의 액세스를 제어하는 운영시스템의 제어하에서 동작한다. 일반적인 운영시스템은 워싱턴, 레드몬드에 소재하는 Microsoft사의 MICROSOFT WINDOWS; 캘리포니아, 쿠퍼티노의 Apple Computer의 MacOS; 뉴욕, 아몬크에 소재하는 International Business Machines의 OS/2; 유타, 솔트레이크시티의 Caldera의 Linux를 포함한다.

<110> 클라이언트 머신(10) 또는 서버(30)가 모바일 장치를 포함하는 실시예에서, 이러한 모바일 장치는 일리노이, 스캬버그의 모토로라사의 i55sr, i58sr, i85s, i88s; 일본, 교토 소재의 Kyocera의 6035 또는 7135; 한국 서울 소재의 Samsung Electronics의 i300 또는 i330과 같은 JAVA 가능한 셀룰러 폰일 수 있다. 모바일 장치를 포함하는 다른 실시예에서, 모바일 장치는 캘리포니아, 밀피타스 소재 palmOne의 Tungsten W, the VII, the VIIx, the i705와 같은 PalmOS 운영체제의 제어하에서 동작하는 개인용 정보 단말기(PDA)일 수 있다. 다른 실시예에서, 클라이언트(113)는 캘리포니아, 팔로알토 소재 Hewlett-Packard Corporation의 iPAQ 4155, iPAQ 5555, iPAQ 1945, iPAQ 2215, iPAQ 4255; 캘리포니아 월넷 소재의 ViewSonic의 ViewSonic V36; 뉴욕 소재 Toshiba America의 Toshiba PocketPC e405과 같은 PocketPC 운영체제의 제어하에서 동작하는 PDA일 수 있다. 다른 실시예에서, 모바일 장치는 캘리포니아, 밀피타스 소재 palmOne의 Treo 180, Treo 270, Treo 600, Treo 650, Treo 700w과 같은 PDA와 전화의 조합일 수 있다. 다른 실시예에서, 모바일 장치는 모토로라사의 MPx200과 같은 PocketPC 운영체제의 제어하에서 동작하는 셀룰러 폰일 수 있다. 전형적인 모바일 장치는 프로세서(102) 및 주 메모리(104)를 포함하여, 도 1c 및 1d에서 상술한 많은 요소들을 포함할 수 있다.

<111> B. 컴퓨팅 환경을 제공하기 위한 시스템 및 방법

<112> 서버(30)로부터 원격 위치에 배치된 클라이언트(10)에서 원격 사용자에게 컴퓨팅 환경을 제공하기 위한 시스템 및 방법에 대해 설명한다. 이 섹션에서의 방법 및 시스템은 일반적으로 서버(30)에 대해 설명하지만, 상기 방법 및 시스템은 서버(30) 및 네트워크 어플라이언스(1250)를 사용하여 또는 이들의 임의의 조합을 사용할 수 있다.

<113> 도 1e를 참조하면, 도 1a에 도시된 원격 머신(30)이 팜(38)을 포함하는 시스템의 일 실시예를 나타내고 있다. 각각의 원격 머신(30)은 네트워크측 인터페이스(202)와 팜측 인터페이스(204)를 포함한다. 원격 머신(30)의 네트워크측 인터페이스(202)는 하나 이상의 로컬 머신(10) 또는 네트워크(210)와 통신할 수 있다. 네트워크(210)는 WAN, LAN 또는 인터넷 또는 월드와이드웹과 같은 인터넷서널 네트워크일 수 있다. 로컬 머신(10)은 네트워크(210)를 사용하여 원격 머신(30)과의 연결을 구축할 수 있다.

<114> 원격 머신(30)이 서로 통신할 수 있도록 원격 머신(30)의 팜측 네트워크(204)는 각각의 통신 링크(200)와 상호 연결된다. 각각의 원격 머신(30)상에서, 팜측 인터페이스(204)는 네트워크측 인터페이스(202)와 통신한다. 팜측 인터페이스(204) 또한 영구 저장장치(230)와 통신하고(화살표 220으로 표시됨), 일 실시예에서, 동적 저장장치(240)와도 통신한다. 원격 머신(30), 영구 저장장치(230) 그리고 동적 저장장치(240)의 조합은 선택적으로 팜(38)으로서 언급된다. 일 실시예에서, 원격 머신(30)은 영구 저장장치(230)와 통신하고 다른 원격 머신(3

0')은 영구저장장치에 저장된 정보에 액세스하기 위하여 원격 머신(30)과 통신한다.

- <115> 영구 저장장치(230)는 데이터가 판독 및 기록되는 것을 허용하며 파워가 저장장치에서 없어지더라도 기록된 데이터를 유지하는, 디스크, 디스크 팜, RAID(redundant array of independent disk), writeable compact disc, 또는 임의의 다른 장치상에 물리적으로 실현될 수 있다. 하나의 물리적 장치가 다수의 영구 저장소를 제공하기 위해 제공될 수 있다, 즉 하나 이상의 팜(38)에 대한 영구 저장장치(230)를 제공하기 위해 하나의 물리적 장치가 사용될 수 있다. 영구 저장장치(230)는 팜(38)내의 각각의 원격 머신(30)과 연관된 정적 데이터(static data)와 팜(38)내의 모든 원격 머신(30)에 의해 사용되는 전체 데이터(global data)를 유지한다. 일 실시예에서, 영구 저장장치(230)는 원격 머신 데이터를 LDAP(Lightweight Directory Access Protocol) 데이터 모델에 유지할 수 있다. 다른 실시예에서, 영구 저장장치(230)는 원격 머신 데이터를 ODBC-compliant database에 저장한다. 여기서, "정적 데이터(static data)"는 자주 변경되지 않는 데이터 즉 오직 시간마다, 날짜마다 또는 주마다 변경되는 데이터 또는 전혀 변경되지 않는 데이터를 의미한다. 각각의 원격 머신은 영구 저장장치(230)로부터 데이터를 판독하기 위해 그리고 영구 저장장치(230)로 데이터를 기록하기 위해 영구 저장장치 서브시스템을 사용한다.
- <116> 영구 저장장치(230)에 저장된 데이터는 신뢰성을 위해 물리적으로 또는 논리적으로 복제될 수 있다. 예를 들어, 데이터 복사를 제공하는, 리던던트 세트, 미러 디스크를 사용하여 물리적 리던던시가 제공될 수 있다. 다른 실시예에서, 데이터베이스의 다중 복사를 제공하기 위해 표준 데이터베이스 기술을 사용하여 데이터베이스 자체가 복제될 수 있다. 다른 실시예에서, 물리적 그리고 논리적 복제가 동시에 사용될 수 있다.
- <117> 동적 저장장치(240)(즉 모든 기록 테이블의 집합)는 다양한 방법으로 구현될 수 있다. 일 실시예에서, 동적 저장장치(240)가 집중화된다; 즉 모든 실행 데이터(runtime data)가 팜(38)내의 하나의 원격 머신(30)의 메모리에 저장된다. 이 원격 머신은 실행 데이터에 대한 액세스를 탐색할 때 팜내의 다른 모든 원격 머신(30)과 통신하는 마스터 네트워크 노드로서 동작한다. 다른 실시예에서, 팜(38)내의 각각의 원격 머신(30)은 동적 저장장치(240)의 전체 복사를 보유한다. 여기서, 각각의 원격 머신(30)은 동적 저장장치(240)의 복사를 최신으로 유지하기 위해 모든 다른 원격 머신(30)과 통신한다.
- <118> 다른 실시예에서, 각각의 원격 머신(30)은 자체 실행 데이터를 유지하고 다른 원격 머신으로부터 실행 데이터를 얻기 위해 탐색할 때 모든 다른 원격 머신(30)과 통신한다. 따라서, 예를 들어, 로컬 머신(10)에 의해 요청된 애플리케이션 프로그램을 찾기 위해 시도하는 원격 머신(30)은 요청된 애플리케이션을 호스팅하는 하나 이상의 원격 머신을 찾기 위해 팜(38)내의 모든 다른 원격 머신(30)과 직접 통신할 수 있다.
- <119> 많은 수의 원격 머신(30)을 갖는 팜(38)에서, 네트워크 트래픽이 과중될 수 있다. 일반적으로 2 이상이며 "콜렉터 포인트(collector points)"로서의, 팜(38)내의 원격 머신(30)의 서브세트(subset)를 지정함으로써(designating) 과중한 네트워크 트래픽을 완화시킨다. 일반적으로, 콜렉터 포인트는 실행 데이터를 수집하는 원격 머신이다. 각각의 콜렉터 포인트는 팜(38)내의 특정 원격 머신(30)으로부터 수집된 실행 데이터를 저장한다. 팜(38)내의 각각의 원격 머신(30)은 콜렉터 포인트로서 동작할 수 있으며, 따라서 콜렉터 포인트로서 지정될 수 있다. 일 실시예에서, 각각의 콜렉터 포인트는 전체 동적 저장장치(240)의 복사를 저장한다. 다른 실시예에서, 각각의 콜렉터 포인트는 동적 저장장치(240)의 일부를 저장한다, 즉 특정 데이터 형식의 실행 데이터를 유지한다. 원격 머신(30)에 의해 저장된 데이터 형식은 하나 이상의 기준(criteria)에 따라 미리 정해질 수 있다. 예를 들어, 원격 머신(30)은 부트 오더(boot order)에 근거하여 다른 형식의 데이터를 저장할 수 있다. 선택적으로, 원격 머신(30)에 의해 저장된 데이터 형식은 관리 툴(140)을 사용하는 관리자(administrator)에 의해 구성될 수 있다. 이러한 실시예에서, 동적 저장장치(240)는 팜(38)내의 2 이상의 원격 머신(30)에 분배된다. 다른 실시예에서, 어플라이언스(1250)는 원격 머신(30), 동적 저장장치(240), 그리고 영구 저장장치(230) 사이에서 통과되는 데이터를 가속하여 과중한 네트워크 트래픽을 완화시킬 수 있다. 이러한 가속은 이하의 섹션 C에서 설명되는 기술들에 의해 제공된다. 예를 들어, 어플라이언스(1250)는 과중한 네트워크 트래픽을 완화시키는데 사용될 수 있다.
- <120> 콜렉터 포인트로서 지정되지 않은 원격 머신(30)은 콜렉터 포인트로서 지정된 팜(38)내의 원격 머신(30)을 인식한다. 콜렉터 포인트로서 지정되지 않은 원격 머신(180)은 실행 데이터를 제공하고 요청할 때 특정 콜렉터 포인트와 통신할 수 있다. 따라서, 팜(38)내의 각각의 원격 머신(30)이 실행 데이터에 액세스하기 위해 탐색할 때 모든 다른 원격 머신(30)과 통신하지 않고 하나의 콜렉터 포인트 원격 머신(30)과 통신하기 때문에 콜렉터 포인트는 네트워크 트래픽을 완화시킨다.
- <121> 각각의 원격 머신(30)은 하나보다 많은 데이터 형식에 대해 콜렉터 포인트로서 동작할 수 있다. 예를 들어, 원

격 머신(30")은 정보를 인증하고 그리고 정보를 로딩하기 위한 콜렉터 포인트로서 동작할 수 있다. 이러한 실시예에서, 각각의 콜렉터 포인트는 다른 형식의 실행 데이터를 수집할 수 있다. 예를 들어, 원격 머신(30")은 인증 정보를 수집할 수 있는 반면 원격 머신(30")은 로딩 정보를 수집한다.

<122> 일 실시예에서, 각각의 콜렉터 포인트는 팜(38)내의 모든 원격 머신(30) 사이에 공유된 데이터를 저장한다. 이러한 실시예에서, 특정 데이터 형식의 각각의 콜렉터 포인트는 콜렉터 포인트에 의해 수집된 데이터를 팜(38)내의 모든 다른 콜렉터 포인트에 대한 데이터 형식으로 변경한다. 따라서, 이러한 데이터의 변경이 완성되면, 각각의 콜렉터 포인트(30" 및 30)는 동일한 데이터를 소유한다. 또한 이러한 실시예에서, 각각의 콜렉터 포인트(30 및 30")는 실행 데이터에 대한 임의의 업데이트를 모든 다른 콜렉터 포인트와 나란히 보유한다.

<123> 브라우징(browsing)은 로컬 머신(10)이 팜(38), 원격 머신(30) 그리고 팜(38)내의 애플리케이션을 조사하고 팜(38) 전체 세션들과 같은 가동 정보에 액세스하는 것을 가능하게 한다. 각각의 원격 머신(30)은 로컬 머신(10)에 브라우징 능력을 제공하기 위해 ICA 브라우징 서브시스템을 포함한다. 로컬 머신(10)이 임의의 원격 머신(30)의 ICA 브라우저 서브시스템(260)과의 연결을 구축한 이후에, 브라우저 서브시스템은 다양한 로컬 머신 요청을 지원한다. 이러한 로컬 머신 요청에는:(1) 팜내의 원격 머신의 명칭을 열거하는 것(enumerating), (2) 팜 내에서 공표된 애플리케이션 명칭을 열거하는 것, (3) 원격 머신 명칭 및/또는 애플리케이션 명칭을 로컬 머신(10)에서 유용한 원격 머신 어드레스로 변형시키는 것(resolving)을 포함한다. ICA 브라우저 서브시스템(260)은 또한 요청에 따라 사용자가 인증된 팜(38)내의 애플리케이션에 대한 조사를 제공하는 근접 프로그램 애플리케이션(program neighborhood application)을 실행시키는 로컬 머신(10)에 의해 만들어진 요청을 지원한다. ICA 브라우저 서브시스템(260)은 전술한 로컬 머신 요청들 모두를 원격 머신(30)내의 적절한 서브시스템으로 전달한다.

<124> 일 실시예에서, 근접 프로그램 서브시스템(270)을 갖는 팜(38)내의 각각의 원격 머신(30)은 로컬 머신(10)의 사용자에게 팜(38)내의 애플리케이션에 대한 조사를 제공할 수 있다. 근접 프로그램 서브시스템(270)은 액세스에 대한 인증을 갖는 로컬 머신(10)의 사용자에게 이러한 애플리케이션에 대한 조사를 제한할 수 있다. 일반적으로, 이러한 근접 프로그램 서비스는 사용자에게 아이콘 목록 또는 그룹으로서 애플리케이션을 보여준다.

<125> 근접 프로그램 서브시스템(270)에 의해 제공되는 기능성은 2개의 로컬 머신 유형에 이용가능하다, (1) 로컬 머신 데스크탑으로부터 직접 이러한 기능성에 액세스할 수 있는 근접 프로그램-가동 로컬 머신(program neighborhood-enabled local machine)과 (2) 원격 머신상에서 근접 프로그램-가동 데스크탑을 실행하여 이러한 기능성에 액세스할 수 있는 비-근접 프로그램-가동 로컬 머신(non program neighborhood-enabled local machine(예를 들면, 레거시 로컬 머신(legacy local machine))).

<126> 근접 프로그램-가동 로컬 머신과 근접 프로그램 서브시스템(270) 사이의 통신은 ICA 가상 채널의 상위에 구축되는 전용 가상 채널(dedicated virtual channel)을 통해 일어날 수 있다. 다른 실시예에서, 이러한 통신은 XML 서비스를 사용하여 일어날 수 있다. 이러한 실시예에 있어서, 근접 프로그램-가동 로컬 머신은 원격 머신(30)상에 근접 프로그램 기능성을 제공하는, 도 6을 참조로 이하에서 설명되는, XML 서비스(516)과 같은 XML 서브시스템으로 통신한다.

<127> 일 실시예에서, 근접 프로그램-가동 로컬 머신은 근접 프로그램 서브시스템(270)에 의한 원격 머신과의 연결을 갖지 않는다. 이 실시예에서, 로컬 머신(10)에 이용가능한 애플리케이션을 식별하기 위하여 로컬 머신(10)은 원격 머신(30)과의 연결을 구축하기 위해 ICA 브라우저 서브시스템(260)에 요청을 전송한다. 그 다음 로컬 머신(10)은 사용자의 증명서(credential)를 요하는 클라이언트측 대화(client-side dialog)를 실행한다. 증명서는 ICA 브라우저 서브시스템(260)에 의해 수신되고 근접 프로그램 서브시스템(270)에 전송된다. 일 실시예에서, 근접 프로그램 서브시스템(270)은 인증(authentication)을 위해 사용자 관리 서브시스템으로 증명서를 전송한다. 사용자 관리 서브시스템은 사용자가 속하는 계정 목록을 나타내는 구별된 명칭 세트를 반환(return)할 수 있다. 인증에 따라, 근접 프로그램 서브시스템(270)은 근접 프로그램 가상 채널을 구축한다. 이러한 채널은 애플리케이션 필터링이 완성될때까지 오픈되어 있다. 일 실시예에서, 섹션 C에서 설명되는 가속 프로그램(6120) 또한 로컬 머신(10) 요청에 대응하여 로컬 머신(10)으로 전송될 수 있다.

<128> 근접 프로그램 서브시스템(270)은 계정과 연관된 공통 애플리케이션 서브시스템(524)으로 근접 프로그램 정보를 요청한다. 공통 애플리케이션 서브시스템(524)은 영구 저장장치(230)로부터 근접 프로그램 정보를 획득한다. 근접 프로그램 정보를 수신하고, 근접 프로그램 서브시스템(270)은 포맷하고 근접 프로그램 정보를 근접 프로그램 가상 채널을 통해 로컬 머신으로 반환한다. 그 다음 부분적인 ICA 연결이 닫힌다.

- <129> 근접 프로그램-가동 로컬 머신이 원격 머신과 부분적 ICA 연결을 구축하는 다른 실시예에서는 팜(38)을 선택하는 로컬 머신(10)의 사용자를 고려한다. 팜(38)의 선택은 선택된 팜(38)내의 원격 머신(30)중 하나와의 ICA 연결을 구축하기 위해 로컬 머신(10)으로부터 ICA 브라우저 서브시스템(260)으로 요청을 전송한다. ICA 브라우저 서브시스템(260)은 팜(38)내의 원격 머신(30)을 선택하는, 근접 프로그램 서브시스템(270)으로 요청을 전송한다. 원격 머신(30)과 연관된 어드레스 정보가 식별되고 ICA 브라우저 서브시스템(260)의 방식에 의해 로컬 머신으로 반환된다. 로컬 머신(10)은 수신된 어드레스 정보에 대응하는 원격 머신(30)과 연결할 수 있다.
- <130> 다른 실시예에서, 근접 프로그램-가동 로컬 머신(10)은 ICA 연결이 지속되는 동안 오픈되는 근접 프로그램 가상 채널에 따른 ICA 연결을 갖는다. 근접 프로그램 가상 채널을 통해, 근접 프로그램 서브시스템(270)은 근접 프로그램 정보의 업데이트를 로컬 머신(10)으로 푸쉬(push)한다. 이러한 로컬 머신(10)으로의 업데이트 푸시는 이하에서 설명되는 가속 기술들에 따라 가속될 수 있다. 업데이트를 획득하기 위해, 근접 프로그램 서브시스템(270)은 공개된 애플리케이션에 대한 변경을 검출하는 것을 허용하도록 공통 애플리케이션 서브시스템(524)으로부터의 이벤트에 기명한다(subscribe).
- <131> 도 1f를 참조하면, 웹 서비스 디렉토리에서 공개된 GUI를 통해 로컬 머신에 이용가능한 다수의 애플리케이션 프로그램을 제공하기 위한 시스템 아키텍처를 나타내는 블록도이다. 이 시스템은 로컬 머신(10)과 다수의 원격 머신(30)을 포함한다. 하나의 원격 머신(30)은 콘텐츠 서버로서 기능한다. 원격 머신(30')은 웹 서버 기능을 제공한다. 원격 머신(30'')은 애플리케이션 파일에 대한 액세스를 제공하기 위한 기능을 제공하고 애플리케이션 서버 또는 파일 서버로서 동작한다. 로컬 머신(10)은 네트워크(155)를 통해 콘텐츠 서버(30), 웹 서버(30') 그리고 애플리케이션 서버(30'')로부터 콘텐츠를 다운로드할 수 있다. 일 실시예에서, 로컬 머신(10)은 클라이언트-애플리케이션 서버 통신 채널(1150)을 통해 애플리케이션 서버(30'')로부터 콘텐츠(예를 들면 애플리케이션)를 다운로드할 수 있다.
- <132> 일 실시예에서, 로컬 머신상의 웹 브라우저(11)는 콘텐츠 서버(30) 및/또는 웹 서버(30')와의 통신을 위해 SSL(Secure Socket Layer)을 사용한다. SSL은 캘리포니아 마운틴 뷰 소재의 Netscape Communication Corporation에 의해 개발된 안전 프로토콜이며, 현재는 IETF(Internet Engineering Task Force)에 의해 표준으로 공표되었다. 웹 브라우저(11)는 캘리포니아 로스앨토스 소재의 Terisa Systems에 의해 개발된 SHTTP(Secure Hypertext Transfer Protocol), 워싱턴 레드몬트 소재의 Microsoft Corporation에 의해 개발된 HTTP over SSL(HTTPS), PCT(Private Communication Technology) 그리고 IETF에 의해 표준으로 공표된 TLS(Transport Level Security)와 같이, 다른 안전 프로토콜을 사용하여 콘텐츠 서버(30) 및/또는 웹 서버(30')에 선택적으로 연결할 수 있다. 다른 실시예에서, 웹 브라우저(11)는 HTTP(Hyper Text Transfer Protocol)와 같은 암호화없는 통신 프로토콜을 사용하여 서버(30)와 통신할 수 있다.
- <133> 또한, 로컬 머신(10)은 클라이언트-애플리케이션 서버 통신 채널(1150)을 통해 애플리케이션 서버(30'')와의 통신을 구축 및 바꾸기 위한 애플리케이션 클라이언트(13)를 포함한다. 일 실시예에서, 애플리케이션 클라이언트(13)는 GUI 애플리케이션이다. 일 실시예에서, 애플리케이션 클라이언트(13)는 플로리다 포트 러더데일 소재의 Citrix Systems Inc에 의해 개발된 ICA(Independent Computing Architecture) 클라이언트일 수 있고, 이하에서 ICA 클라이언트(13)라 한다. 애플리케이션 클라이언트(13)의 다른 실시예는 워싱턴 레드몬트 소재의 Microsoft Corporation에 의해 개발된 RDP(Remote Display Protocol) 클라이언트, 멀티미디어 애플리케이션, 이메일, 자바 또는 .NET code를 실행할 수 있는 X-Windows 클라이언트(13), 클라이언트측 플레이어, 인터프리터 또는 시뮬레이터를 포함한다. 또한, 애플리케이션 서버(30'')상에서 실행되는 애플리케이션의 출력은 ICA 클라이언트(13)를 통해 로컬 머신(10)에서 디스플레이될 수 있다. 일 실시예에서, 애플리케이션 클라이언트(13)는 도 5를 참조하여 자세히 설명되는 애플리케이션 스트리밍 클라이언트(552)와 같은 애플리케이션 클라이언트이다. 일 실시예에서, 애플리케이션 클라이언트(13)는 클라이언트(10)와 서버(30) 사이의 통신을 가속하기 위한 실시예에 따른 가속 프로그램을 포함한다.
- <134> 로컬 머신(10)은 웹 서비스를 위한 웹 서비스 디렉토리(160)를 검색한다. 일 실시예에서, 이러한 검색은 매뉴얼 검색이다. 선택적으로, 검색은 자동 검색이다. 웹 서비스 디렉토리내의 웹 서비스 검색을 위해 웹 서비스 디렉토리(160) 또한 화이트 및 옐로우 페이지와 같은 서비스 기반 뷰를 제공할 수 있다. 다른 실시예에서, 웹 서비스 디렉토리(160)는 GUI 애플리케이션을 위해 구조화된 서비스 명칭 및 서비스 종류에 근거한 계층적 브라우징을 지원한다. 일 실시예에서, 웹 서비스 디렉토리(160)는 디렉토리 서버와 같이, 콘텐츠 서버(30)와 독립적인 원격 머신상에서 실행된다. 다른 실시예에서, 웹 서비스 디렉토리(160)는 다중 서버들상에서 실행된다.
- <135> 일 실시예에서, 콘텐츠 서버(30)는 웹 서비스 디렉토리(160)에 정보 또는 분석을 제공함으로써 추가적인 분석

또는 정보에 근거하여 로컬 머신(10)이 웹 서비스를 선택하도록 한다. 웹 서비스 디렉토리(160)가 포함할 수 있는 서비스 정보의 목록은 서비스를 제공하는 비즈니스의 명칭, 서비스 유형, 서비스의 텍스트 설명, 하나 이상의 서비스 액세스 포인트(service access point:SAP), 네트워크 유형, 사용자로의 경로(예를 들면, TCP 또는 HTTPS), 그리고 서비스 퀄리티(quality of service:Qos) 정보이며, 이에 국한되는 것은 아니다. 또한, 서비스 정보는 클라이언트 장치 유형 또는 사용자 특정될 수 있다. 따라서, 서비스 선택은 상기 속성들중 하나 이상에 근거하여 이뤄질 수 있다.

<136> 일 실시예에서, 서비스 유형은 로컬 머신(10)이 웹 서비스에 액세스하기 위해 반드시 사용해야 하는 프로그래밍 인터페이스를 표시한다. 예를 들어, 서비스 유형은 서비스가 WSDL(Web Service Description Language)과 같은 인터페이스 기술 언어에 의해 인코딩되었다는 것을 나타낼 수 있다.

<137> 서비스 액세스 포인트(또는 SAP)는 애플리케이션에 대한 유일한 어드레스이다. SAP는 컴퓨터 시스템이 로컬 머신(10) 및 각각의 원격 머신(30)에서 다수의 애플리케이션을 지원할 수 있도록 한다. 예를 들어, 애플리케이션 서버(30")는 전자 메일(즉, 이메일) 애플리케이션, 파일 전송 애플리케이션, 및/또는 GUI 애플리케이션을 지원할 수 있다. 일 실시예에서, 이러한 애플리케이션들은 애플리케이션 서버(30")내에서 고유 SAP 를 각각 가질 것이다. 일 실시예에서, SAP는 웹 또는 인터넷 어드레스이다(예를 들면, DNS 명칭, IP/port, 또는 URL(Uniform Resource Locator)). 따라서, 일 실시예에서 SAP는 웹 서버(30')상에 저장된 애플리케이션에 대한 어드레스의 일부로서 웹 서버(30')의 어드레스를 식별한다. 일 실시예에서, SAP는 웹 서버(30')상에 저장된 애플리케이션에 대한 어드레스의 일부로서 퍼블리싱 서버 플러그인(165)의 어드레스를 식별한다. 일 실시예에서, SAP는 UDDI 레지스트리로부터의 "accessPoint" 이다.

<138> 웹 서비스 디렉토리(160)에서의 퍼블리싱을 위한 아이템을 준비하기 위해, 콘텐츠 서버(30)는 웹 퍼블리싱 툴(170)을 포함한다. 일 실시예에서, 웹 퍼블리싱 툴(170)은 소프트웨어 모듈이다. 선택적으로, 웹 퍼블리싱 툴(170)은 콘텐츠 서버(30)내에 위치하거나 또는 외부에 위치할 수 있는 다른 서버이다.

<139> 일 실시예에서, 웹 서버(30')는 로컬 머신(10)에 웹 페이지를 전달한다. 웹 서버(30')는 로컬 머신(10)에 웹 페이지를 제공할 수 있는 임의의 원격 머신(30)이 될 수 있다. 다른 실시예에서, 웹 서버(30')는 기업정보포털(Enterprise Information Portal)(예를 들면, 회사 인트라넷 또는 보안 비즈니스-투-비즈니스 엑스트라넷)이다. 기업 포털은 체계화하기 위한 관리 툴을 제공하고 정보를 더욱 효과적으로 사용하면서, 애플리케이션, 데이터 그리고 콘텐츠를 결집하고 개인화하고 사용자들에게 제공하는 회사 웹 사이트들이다. 일부 회사에서, 가상 작업공간으로의 브라우저-기반 액세스를 갖는 종래 데스크탑 소프트웨어를 대체한다. 일 실시예에서, 임의의 가속 기술들을 사용하여 어플라이언스(1250)는 웹 페이지의 준비(provision) 제공을 가속한다. 다른 실시예에서, 가속 프로그램(6120)이 웹 페이지 제공을 가속한다.

<140> 웹 서버(30')는 또한 GUI 애플리케이션의 퍼블리싱을 가능하게 하기 위한 퍼블리싱 서버 플러그인(165)을 포함한다. 좀더 구체적으로, 퍼블리싱 서버 플러그인(165)은 새로운 웹 서비스 엔트리 URL를 GUI 애플리케이션 서비스로 번역(translate)하고 이로써 GUI는 웹 서비스 디렉토리(160)를 통해 액세스될 수 있다. 일 실시예에서, 퍼블리싱 서버 플러그인(165)은 CGI 설명서에 따르는 데이터를 받고 재전송하도록 설계된 프로그램인 CGI(Common Gateway Interface) 스크립트이다. 프로그램은 C, Perl, Java 또는 Visual Basic 과 같은 임의의 프로그래밍 언어로 작성될 수 있다. 다른 실시예에서, 퍼블리싱 서버 플러그인(165)은 Java Server Page(JSP)이다. 원격 GUI 애플리케이션의 퍼블리케이션을 쉽게 하기 위해 퍼블리싱 서버 플러그인(165)을 사용하여, 로컬 머신(10)은 프로그래밍 인터페이스 또는 웹 페이지를 통해서가 아니라 Citrix의 ICA 또는 Microsoft의 RDP와 같은 GUI 인터페이스를 통해 웹 서비스에 액세스할 수 있다. 일 실시예에서, 어플라이언스(1250) 또는 가속 프로그램(6120)은 섹션 C 에서 자세히 설명되는 임의의 가속 기술들을 사용하여 클라이언트로의 GUI 제공을 가속한다.

<141> 애플리케이션 서버(30")는 로컬 머신(10)이 사용할 수 있는 하나 이상의 애플리케이션을 호스팅한다. 이러한 애플리케이션들의 예에는 MICROSOFT WORD와 같은 워드 처리 프로그램, MICROSOFT EXCEL 과 같은 스프레드시트 프로그램, 재정정보 보고 프로그램, 고객등록프로그램, 기술 지원 정보를 제공하는 프로그램, 고객 데이터베이스 애플리케이션 또는 애플리케이션 설정 매니저를 포함한다.

<142> 일 실시예에서, 웹 퍼블리싱 툴(170)은 웹 퍼블리싱 툴(170)이 영구적 대용량 저장장치(225)의 웹 서비스 디렉토리(160)내에서 퍼블리싱하는 애플리케이션에 대한 정보를 저장한다. 일 실시예에서 이러한 정보는 퍼블리싱 서버 플러그인(165)에 대한 URL 이다. 영구적 대용량 저장장치(225)는 마그네틱 디스크 또는 마그네티-옵티컬 드라이브이다. 일 실시예에서, 영구적 대용량 저장장치(225)는 퍼블리싱된 애플리케이션과 연관된 데이터를

하나 이상의 로컬 서비스 데이터베이스에 저장하는, 데이터베이스 서버이다. 영구적 대용량 저장장치(225)는 원격 머신(30) 내부에 또는 외부에 위치되는 부품일 수 있다.

- <143> 다른 실시예에서, 콘텐츠 서버(30) 또는 웹 서버(30')는 애플리케이션의 목록을 검색하기 위해 팜(38)내의 원격 머신(30)과 통신한다. 이러한 실시예들에서, 콘텐츠 서버(30) 또는 웹 서버(30')는 영구적 대용량 저장장치(225) 대신에 팜(38)과 통신한다.
- <144> 도 2를 참조하면, 애플리케이션 프로그램 실행 방법을 선택하기 위한 단계들의 일 실시예를 나타내는 흐름도이다. 간단히 설명하면, 로컬 머신에 의해 실행 가능한 애플리케이션 목록에 대한 요청과 함께, 로컬 머신 또는 로컬 머신의 사용자와 연관된 자격(credential)이 수신된다(단계 202). 수신된 자격에 대응하여 다수의 애플리케이션 프로그램의 목록이 로컬 머신에 제공된다(단계204). 목록된 애플리케이션을 실행하기 위한 요청이 수신된다(단계206). 정책에 대응하여, 목록된 애플리케이션을 실행하기 위한 미리 지정된 수의 방법들 중 하나가 선택되고, 미리 지정된 수의 방법들은 목록된 애플리케이션의 애플리케이션 스트리밍을 위한 방법을 포함한다(단계208).
- <145> 로컬 머신에 의해 실행 가능한 애플리케이션 목록에 대한 요청과 함께, 로컬 머신 또는 로컬 머신의 사용자와 연관된 자격(credential)이 수신된다(단계 202). 일 실시예에서, 원격 머신은 자격과 함께 로컬 머신(10)으로부터 가능한 애플리케이션 목록에 대한 요청을 수신한다. 다른 실시예에서, 원격 머신(30)상의 XML 서비스는 이러한 요청 및 자격을 수신하고 요청 및 자격을 원격 머신(30)상의 관리 서비스로 전송한다.
- <146> 일 실시예에서, 웹 서버로서 기능하는 원격 머신(30)은 로컬 머신(10)으로부터 통신을 수신하고 원격 머신(30')으로 전달한다. 이러한 실시예에서, 웹 서버는 원격 머신(30')상의 XML 서비스로 상기 통신을 전송한다. 다른 예에서, 웹 서버는 로컬 머신상에 존재한다. 로컬 머신(10)으로부터의 통신이 웹 서버에 의해 원격 머신(30')으로 라우팅되는 실시예에서, 원격 머신(30)은 로컬 머신(10)의 인터넷 프로토콜(IP) 어드레스에 대응하여 선택될 수 있다.
- <147> 일 실시예에서, 로컬 머신(10)은 원격 머신(30)상에 존재하는 애플리케이션에 대한 액세스를 요청한다. 이러한 실시예에서, 로컬 머신(10)은 원격 머신(30)상에 존재하는 애플리케이션의 원격 머신(30)에 의한 실행을 요청한다. 다른 예에서, 로컬 머신(10)은 이러한 애플리케이션을 포함하는 다수의 애플리케이션 파일의 검색을 요청한다.
- <148> 일 실시예에서, 사용자는 원격 머신(30)에 의해 로컬 머신(10)에 표시되는 그래픽 사용자 인터페이스를 통해 원격 머신(30)에 자격을 제공한다. 다른 실시예에서, 웹 서버의 기능을 갖는 원격 머신(30')은 로컬 머신(10)에 그래픽 사용자 인터페이스를 제공한다. 다른 실시예에서, 원격 머신(30)에 의해 로컬 머신(10)에 전송되는 수집 에이전트는 로컬 머신(10)으로부터 자격을 모은다. 일 실시예에서, 자격은 사용자명 그리고 패스워드로 언급된다. 다른 실시예에서, 자격은 사용자명 및 패스워드에 한정되지 않고 로컬 머신(10)의 머신 ID, 운영체제 유형, 운영체제로의 패치 존재, 설치된 네트워크 카드의 MAC 어드레스, 클라이언트 장치상의 디지털 워터마크, Active Directory내의 멤버십, 바이러스 스캐너의 존재, 개인 방화벽의 존재, HTTP 헤더, 브라우저 유형, 장치 유형, 인터넷 프로토콜 어드레스 또는 어드레스 범위와 같은 네트워크 연결 정보, 원격 머신(30)의 머신 ID, 액세스 요청의 날짜 또는 시간 그리고 인증 자격을 포함한다.
- <149> 일 실시예에서, 로컬 머신과 연관된 자격은 로컬 머신의 사용자와 연관된다. 이러한 실시예에서, 자격은 사용자에 의해 소유된 정보이다. 다른 예에서, 자격은 사용자 인증 정보이다. 다른 실시예에서, 로컬 머신과 연관된 자격은 네트워크와 연관된다. 일예에서, 자격은 로컬 머신이 연결할 수 있는 네트워크와 연관된 정보이다. 다른 예에서, 자격은 로컬 머신에 대한 정보를 수집하는 네트워크와 연관된 정보이다. 다른 실시예에서, 로컬 머신과 연관된 자격은 로컬 머신의 특성이다.
- <150> 수신된 자격에 대응하여, 로컬 머신에서 사용가능하는 다수의 애플리케이션 프로그램의 목록이 제공된다(단계 204). 일 실시예에서, 로컬 머신(10)의 사용자는 원격 머신(30)에 의해 호스팅된 애플리케이션을 어디에서 찾을 것인가 알지 못하고 그리고 이러한 애플리케이션에 링크하기 위해 필요한 기술적 정보없이 원격 머신(30)에 의해 호스팅된 애플리케이션 프로그램의 가용성을 알 수 있다. 이러한 가용 애플리케이션 프로그램은 사용자의 "근접 프로그램(program neighborhood)"을 포함한다. 로컬 머신에 대한 근접 프로그램을 결정하기 위한 시스템은 애플리케이션 프로그램(이하에서는 "근접 프로그램" 애플리케이션으로 언급됨), 애플리케이션 프로그램의 컴포넌트들을 저장하기 위한 메모리 그리고 애플리케이션 프로그램을 실행하기 위한 프로세서를 포함한다. 근접 프로그램(PN) 애플리케이션은 로컬 머신(10) 및/또는 원격 머신(30)의 메모리에 설치될 수 있다.

- <151> 근접 프로그램 애플리케이션에 따라 동작하는 원격 머신(30)은 팜(38)내의 원격 머신(30) 각각으로부터 애플리케이션-연관 정보를 수집한다. 각각의 호스팅된 애플리케이션에 대한 애플리케이션-연관 정보는 다양한 정보 예를 들어 애플리케이션을 호스팅하는 원격 머신의 어드레스, 애플리케이션 명칭, 애플리케이션을 사용하도록 인증된 사용자들 또는 사용자들의 그룹, 그리고 애플리케이션을 실행하기 위해 연결을 구축하기 이전에 로컬 머신(10)에 요구되는 최소한의 성능을 포함한다. 예를 들어, 애플리케이션은 비디오 데이터를 스트리밍할 수 있고, 따라서 로컬 머신이 비디오 데이터를 지원할 수 있는 최소한의 성능이 요구된다. 다른 예로는 로컬 머신이 오디오 데이터를 지원하거나 또는 암호화된 데이터를 처리할 수 있는 성능을 갖는 지이다. 애플리케이션-연관 정보는 데이터베이스에 저장될 수 있다.
- <152> 로컬 머신(10)이 네트워크(40)에 연결될 때, 로컬 머신(10)의 사용자는 사용자 자격을 제공한다. 사용자 자격은 로컬 머신(10) 사용자의 사용자명, 사용자의 패스워드, 사용자가 인증된 도메인 명칭을 포함할 수 있다. 선택적으로, 사용자 자격은 스마트 카드, 시간-기반 토큰, 사회 보장 번호, 사용자 패스워드, 개인식별(PIN) 번호, 대칭키(symmetrical key) 또는 ECC(elliptic curve cryptography)에 근거한 디지털 증명, 사용자의 생물학적 특성, 또는 로컬 머신(10)의 사용자를 식별하는 임의의 수단으로부터 얻어질 수 있고 인증을 위해 제출된다. 로컬 머신(10)에 대응하는 원격 머신(30)은 사용자 자격에 근거하여 사용자를 인증한다. 사용자 자격은 근접 프로그램 애플리케이션이 실행되는 곳에 저장될 수 있다. 로컬 머신(10)이 근접 프로그램 애플리케이션을 실행하는 실시예에서, 사용자 자격은 로컬 머신(10)에 저장될 수 있다. 원격 머신(30)이 근접 프로그램 애플리케이션을 실행하는 실시예에서, 사용자 자격은 원격 머신(30)에 저장될 수 있다.
- <153> 사용자 자격 및 애플리케이션-연관 정보로부터, 원격 머신(30)은 원격 머신(30)에 의해 호스팅된 어떤 애플리케이션 프로그램이 로컬 머신(10)의 사용자에게 의해 사용될 수 있는지를 결정할 수 있다. 원격 머신(30)은 사용 가능한 애플리케이션 프로그램을 나타내는 정보를 로컬 머신(10)에 전송한다. 이러한 프로세스는 로컬 머신(10)의 사용자가 애플리케이션 연결을 구축할 필요가 없게 한다. 또한, 원격 머신(30)의 관리자는 로컬 머신(10)의 다수의 사용자들 중에서 애플리케이션에 대한 액세스를 제어할 수 있다.
- <154> 일 실시예에서, 애플리케이션들이 다른 원격 머신(30')에 존재할 수 있지만, 원격 머신(30)에 의해 수행되는 사용자 인증은 로컬 머신(10)에 존재하는 각각의 호스팅된 애플리케이션 프로그램 사용을 인증하는데 충분할 수 있다. 따라서, 로컬 머신(10)이 호스팅된 애플리케이션 중 하나를 시작(즉 실행을 시작)할 때, 로컬 머신(10)에 의한 추가적인 사용자 자격은 애플리케이션의 사용을 인증하는데 필요하지 않을 수 있다. 따라서, 사용자 자격의 단일 엔트리가 사용 가능한 애플리케이션을 결정하는데 그리고 사용자에게 의한 추가적인, 매뉴얼 로그인 인증 프로세스 없이 애플리케이션의 개시를 인증하는데 이용될 수 있다.
- <155> 로컬 머신(10) 또는 원격 머신(30)은 근접 프로그램 애플리케이션을 개시할 수 있다. 로컬 머신(10, 20)의 디스플레이 스크린(12, 22)상에 그 결과가 디스플레이된다. 그래픽 윈도우-기반 구현에서, 결과는 근접 프로그램 그래픽 윈도우에 디스플레이될 수 있고 각각의 인증된 애플리케이션 프로그램은 상기 윈도우에서 그래픽 아이콘으로 표현될 수 있다.
- <156> 일 실시예에서, 근접 프로그램 애플리케이션은 로컬 머신(10)이 실행을 비인증한 애플리케이션을 필터링하고 오직 인증된(즉 사용 가능한) 프로그램만을 디스플레이한다. 다른 실시예에서, 근접 프로그램 애플리케이션은 인증된 그리고 비인증된 애플리케이션들을 디스플레이할 수도 있다. 비인증된 애플리케이션이 필터링되지 않아 디스플레이될 때, 이러한 애플리케이션이 사용할 수 없다는 것을 나타내는 공지가 제공될 수 있다. 선택적으로, 어떤 애플리케이션들이 로컬 머신(10)에 의해 인증 또는 비인증된 것인지 식별하지 않고, 근접 프로그램 애플리케이션은 원격 머신(30)에 의해 호스팅된 모든 애플리케이션을 로컬 머신(10)의 사용자에게 보고할 수 있다. 로컬 머신(10)이 이러한 애플리케이션들 중 하나를 실행하려고 시도할 때 인증이 후속적으로 결정될 수 있다.
- <157> 로컬 머신(10)은 원격 머신(30)으로 애플리케이션 목록을 요청할 수 있다. 애플리케이션 목록은 로컬 머신(10)의 사용자가 모든 퍼블리쉬된 애플리케이션의 명칭을 볼 수 있도록 한다. 일 실시예에서, 로컬 머신(10)의 사용자는 사용자가 애플리케이션을 실행할 인증을 갖는지 여부에 상관없이 애플리케이션 명칭을 볼 수 있다. 다른 실시예에서, 사용자는 오직 사용자가 인증된 애플리케이션 명칭만을 볼 수 있다.
- <158> 애플리케이션 목록에 대한 요청은 로컬 머신(10)에 의해 실행되는 특정 프로세스에 따라서, ICA 브라우저 서브시스템(260)으로, 근접 프로그램 서브시스템(270)으로, 또는 공통 애플리케이션 서브시스템(524)으로 전달된다. 예를 들어, 로컬 머신(10)이 근접 프로그램 애플리케이션을 실행할 때, 애플리케이션 목록에 대한 요청은 원격 머신(30)상의 근접 프로그램 서브시스템(270)에 보내진다. 로컬 머신(10)이 목록 요청을 웹 페이지를 통해 제

출할 때, 요청은 공통 액세스 포인트 서브시스템(524)으로 전달된다. 이러한 실시예에서, 공통 애플리케이션 서브시스템(524)은 로컬 머신(10)이 애플리케이션을 열거하기를 원할 때, 근접 프로그램 서브시스템(270), ICA 브라우저 서브시스템(260), 그리고 공통 애플리케이션 서브시스템들의 초기 액세스 포인트로서 이용된다. 일 실시예에서, 로컬 머신(10)이 목록 요청을 웹 페이지를 통해 제출할 때, 웹 서버를 호스팅하는 중개(intermediate) 원격 머신(30)은 요청을 수신하고 요청을 원격 머신(30')에 전송한다.

<159> 목록 요청을 수신함에 따라, 공통 애플리케이션 서브시스템(524)은 모든 애플리케이션의 목록에 대해 영구 저장 장치(230)에 질의(query)한다. 근접 프로그램 서브시스템(270) 및 공통 액세스 포인트 서브시스템(645)으로부터 수신된 요청에 대해, 로컬 머신(10)의 사용자의 자격에 따라 애플리케이션 목록이 필터링된다(즉 사용자는 사용자가 인증된 애플리케이션에 대해서만 볼 수 있다).

<160> 로컬 머신(10)은 또한 원격 머신 목록을 요청할 수 있다. 원격 머신 목록은 로컬 머신(10)의 사용자가 팜(38) 내의 원격 머신들의 목록을 볼 수 있게 한다. 일 실시예에서, 원격 머신들의 목록은 원격 머신상의 특정 원격 머신 서브시스템에 의해 결정되는 것과 같이, 원격 머신의 유형에 따라 필터링될 수 있다.

<161> 원격 머신 목록에 대한 요청은 로컬 머신(120)에 의해 실행되는 특정 프로세스에 따라서, ICA 브라우저 서브시스템(260)으로 또는 공통 액세스 포인트 서브시스템(645)으로 전달된다. 예를 들어, 로컬 머신(120)이 웹 페이지를 통해 원격 머신 목록 요청을 제출할 때, 요청은 공통 액세스 포인트 서브시스템(645)으로 전달된다. 이러한 실시예에서, 공통 원격 머신 서브시스템(300)은 ICA 브라우저 서브시스템(260) 및 공통 액세스 포인트 서브시스템(645)에 대한 초기 액세스 포인트로서 이용된다. 원격 머신 목록 요청을 수신함에 따라, 공통 원격 머신 서브시스템은 모든 원격 머신들의 목록을 영구 저장장치(230)에 질의한다. 선택적으로, 원격 머신들의 목록은 원격 머신 유형에 따라 필터링된다.

<162> 도 3a는 월드와이드웹을 통해 프로그램 근접 애플리케이션의 실행을 개시하는 로컬 머신의 일 실시예를 도시한 블록도이다. 로컬 머신(10)은 캘리포니아, 마운틴 뷰 소재의 네스케이프 커뮤니케이션사의 네스케이프 네비게이터, 워싱턴 레드몬드 소재 마이크로소프트사의 마이크로소프트 인터넷 익스플로러, 캘리포니아, 마운틴 뷰 소재의 모질라 파운데이션의 파이어폭스, 노르웨이 오슬로의 오페라 소프트웨어 ASA 소재의 오페라, 캘리포니아 쿠파ertino 소재의 애플 컴퓨터사의 사파리와 같은 웹 브라우저 애플리케이션(80)을 실행한다.

<163> 로컬 머신(10)은 원격 머신(30)상에 존재하는 HTML 페이지에 대응하는 URL 어드레스에 액세스하기 위해 요청(82)을 웹 브라우저(80)를 통해 전송한다. 원격 머신(30)에 의해 로컬 머신(10)으로 재전송된 첫번째 HTML 페이지(84)는 로컬 머신(10)을 식별하기 위해 탐색하는 인증 페이지이다.

<164> 도 3a를 참조하여, 로컬 머신(10)이 원격 머신(30)에 의해 인증되면, 원격 머신(30)은 로컬 머신(10)이 액세스한 애플리케이션 프로그램을 나타내는 그래픽 아이콘(57, 57')이 나타난 근접 프로그램 윈도우(58)를 포함하는 HTML 페이지(88)를 로컬 머신에 전송한다. 로컬 머신(10)의 사용자는 아이콘(57)을 클릭하여 아이콘(57)에 의해 나타나는 애플리케이션의 실행을 호출한다.

<165> 일 실시예에서, 원격 머신(30)은 로컬 머신(10)의 사용자 대신에 근접 프로그램 애플리케이션을 실행한다. 이러한 실시예에서, 원격 머신(30)은 로컬 머신(10)과 원격 머신(30') 사이에 존재하는 중개 원격 머신이다.

<166> 도 3b를 참조하여, 로컬 머신이 웹 서비스 디렉토리를 이용하여 열거된 애플리케이션 프로그램을 액세스하기 위해 로컬 머신에 의해 행해지는 단계들의 일 실시예를 도시한 흐름도이다. 웹 퍼블리싱 툴(170)은 퍼블리싱을 위한 애플리케이션(예를 들면 GUI 애플리케이션)에 대한 웹 서비스 디스크립션 및 액세스 정보를 수신한다(단계 300). 일 실시예에서, 웹 서비스 디스크립션은 전술한 서비스 정보(예를 들면, 웹 서비스를 제공하는 비즈니스 명칭, 서비스 유형, 서비스의 텍스트형 디스크립션, SAP)를 포함한다. 액세스 정보는 예를 들어 퍼블리싱된 애플리케이션 명칭, TCP 브라우징 서버 팜 어드레스, 그리고 MetaFrame 서버 IP 어드레스를 포함할 수 있다. 일 실시예에서, 액세스 정보는 사용하기 위한 어드레스와 네트워크 또는 보안 게이트웨이 또는 브리지 장치를 가로지르는데 사용하기 위한 티켓을 지정한다(specify).

<167> 웹 퍼블리싱 툴(170)은 웹 서비스(즉 GUI 애플리케이션)의 퍼블리케이션을 요청하기 위해 서비스-퍼블리싱 요청을 구성한다(단계 305). 일 실시예에서, 서비스-퍼블리싱 요청은 SAP를 포함한다. 일 실시예에서, SAP 는 퍼블리싱 서버 플러그인(165)과 웹 서버(30')의 웹 어드레스를 포함하는 URL이다. 또한, 웹 어드레스는 웹상에서 오브젝트로 언급되는 명칭 및 어드레스의 유형들에 대한 일반적인 용어인, URI(Uniform Resource Identifier)가 될 수 있다. URL 은 URI의 한가지 종류이다. URI의 예는 웹 서버(30')의 명칭(예를 들어, "web-server") 과 퍼블리싱 서버 플러그인(165)에 대한 CGI 스크립트 명칭 (예를 들어, "dynamic-component")이다.

- <168> 웹 퍼블리싱 툴(170)은 영구적 대용량 저장장치(225)에 SAP와 연관된 SAP 엔트리를 저장한다(단계 310). 일 실시예에서, 웹 퍼블리싱 툴(170)은 퍼블리싱된 애플리케이션 정보(예를 들어, ICA-published-app-info)와 GUI 애플리케이션을 연관시킨다. 웹 퍼블리싱 툴(170)은 또한 콘텐츠 서버(30)가 영구적 대용량 저장장치(225)에 저장한 SAP 엔트리를 식별하기 위해 서비스-퍼블리싱 요청에 키를 포함한다. 예를 들면, 키는 "123456677" 값을 가질 수 있다. 웹 서버(30'), 퍼블리싱 서버 플러그인(165)의 CGI 스크립트 명칭, 그리고 전송한 키를 식별하는 SAP의 예로 "http://web-server/dynamic-component/?app=123456677"가 있다.
- <169> 전송한 SAP와 연관된 SAP 엔트리의 예로는 "key=123456677, value=ICA-published-app-info."가 있다. 키는 임의의 길이(예를 들어, 56 비트 키, 128 비트 키)가 될 수 있다. 일 실시예에서, 키는 암호화된 랜덤 숫자이다. 키는 또한 키 홀더에 액세스 권리를 제공할 수 있다. 키로 설명되었지만, 영구적 대용량 저장장치(225)에 저장된 SAP 엔트리에 대한 보안 형식을 제공하기 위해 임의의 수단이 사용될 수 있다.
- <170> 웹 퍼블리싱 툴(170)은 웹 서비스 디렉토리(160)에서의 퍼블리싱을 위해 콘텐츠 서버(30)에 서비스-퍼블리싱 요청을 제공한다(단계 315). 또한, 일 실시예에서, 콘텐츠 서버(30)는 특정 웹 서비스를 요청하는 로컬 머신(10)에 SAP의 키를 전송한다. 일 실시예에서, 서비스-퍼블리싱 요청의 퍼블리싱은 로컬 머신(10)의 사용자가 서비스에 액세스할 수 있도록 한다. 일 실시예에서, GUI 애플리케이션은 플로리다, 포트 러더데일 소재의 Citrix Systems의 NFUSE를 사용하여 웹 서비스 디렉토리(160)상에서 퍼블리싱된다. 일 실시예에서, GUI 애플리케이션의 퍼블리싱은 Citrix Systems의 ALE(Application Launching and Embedding)를 사용하여 웹 서비스 디렉토리(160)상에서 GUI 애플리케이션을 커스터마이징한다. ALE 는 HTML 페이지로부터 GUI 애플리케이션을 개시 또는 HTML 페이지로의 애플리케이션의 임베딩을 가능하게 한다.
- <171> 로컬 머신(10)은 웹 서비스 디렉토리(160)로 서비스 명칭을 질의한다(단계 320). 콘텐츠 서버(30)는 로컬 머신(10)으로부터 질의를 수신하고(단계 325) 웹 서비스 디렉토리(160)에서 요청된 서비스 명칭을 검색한다. 다른 실시예에서, 로컬 머신(10)의 사용자는 로컬 머신(10)의 사용자가 검색하기 위해 시도한 특정 서비스 명칭을 위치시킬때까지 웹 서비스 디렉토리(160)를 탐색한다(navigate). 로컬 머신(10)으로 설명되었지만, 임의의 웹 서비스 디렉토리 클라이언트(예를 들어, UDDI 클라이언트 또는 LDAP 브라우저)가 퍼블리싱된 웹 서비스를 발견하기 위해 웹 서비스 디렉토리(160)를 질의 또는 탐색할 수 있다.
- <172> 수신된 질의와 연관된 SAP의 위치에 따라, 콘텐츠 서버(30)는 SAP를 로컬 머신(10)에 전송한다(단계 330). 로컬 머신(10)은 SAP를 수신하고(단계 335) SAP로부터 퍼블리싱 서버 플러그인(165)의 어드레스를 결정한다. 로컬 머신(10)은 후속적으로 GUI 애플리케이션에 대한 요청을 웹 서버(30')에 전송한다(단계 340). 일 실시예에서, 로컬 머신(10)으로부터의 요청은 웹 브라우저(11)로부터 웹 서버(30')에 전송된 HTTP 요청이다. 다른 실시예에서, 로컬 머신(10)상에서 실행하는 애플리케이션(예를 들어, 일반적인 디렉토리 브라우저 또는 HTML UI)은 콘텐츠 서버(30)로부터 SAP를 수신하고 SAP를 인수(argument)로서 웹 브라우저(11)에 제공한다. 웹 브라우저(11)는 자동적으로 HTTP 요청(GUI 애플리케이션에 대한)을 웹 서버(30')에 전송할 수 있다. 웹 서버(30')에 대한 애플리케이션 요청의 특정 예로 <http://web-server/dynamic-component/?app=123456677>.
- <173> 웹 서버(30'), 그리고, 좀더 구체적으로는, 퍼블리싱 서버 플러그인(165), 는 SAP와 연관된 애플리케이션 요청을 수신하고(단계 345) 요청과 연관된 SAP 엔트리를 결정한다(단계 350). 일 실시예에서, 퍼블리싱 서버 플러그인(165)은 로컬 머신(10)으로부터 요청을 수신하고 영구적 대용량 저장장치(225)에 저장되었던 요청과 연관된 퍼블리싱된 애플리케이션 정보를 검색한다. 일 실시예에서, 퍼블리싱 서버 플러그인(165)은 콘텐츠 서버(30)로부터 수신된 SAP(또는 SAP의 일부)를 영구적 대용량 저장장치(225)에 저장된 적절한 서비스 엔트리(예를 들어, 퍼블리싱된 애플리케이션 정보)에 액세스하기 위한 키로서 사용한다.
- <174> 퍼블리싱 서버 플러그인(165)은 퍼블리싱된 애플리케이션 정보(예를 들어, 애플리케이션 서버(30"))의 HTTP 어드레스를 갖는 파일 또는 문서를 구성하고 이 문서를 로컬 머신(10)에 전송한다(단계 355). 퍼블리싱 서버 플러그인(165)은 파일이 애플리케이션 클라이언트(13)와 호환될 수 있는 형식을 갖도록 구성한다. 일 실시예에서, 문서는 MIME(Multipurpose Internet Mail Extensions) 또는 S/MIME(secure MIME) 문서이다. 다른 실시예에서, 문서는 ICA 웹 클라이언트 임베디드 오브젝트 HTML 태그를 포함하는 HTML 문서이다. 다른 실시예에서, 문서는 애플리케이션 스트리밍 클라이언트 임베디드 오브젝트 HTML 태그를 포함하는 HTML 문서이다.
- <175> 웹 브라우저(11)는 문서를 수신하고 문서를 열기위해 시도한다. 일 실시예에서, 애플리케이션 클라이언트(13)가 로컬 머신(10)상에 설치되지 않으면, 로컬 머신(10)은 애플리케이션 클라이언트(13)를 다운로드하고 설치하기 위해 애플리케이션 서버(30")와 통신한다. 애플리케이션 클라이언트(13)가 설치되면, 또는 애플리케이션 클라이언트(13)가 이미 로컬 머신(10)상에 설치되어 있으면, 로컬 머신(10)은 웹 서버(30')로부터 수신된 문서를

보기 위해 애플리케이션 클라이언트(13)를 개시한다(단계 360).

- <176> 애플리케이션 클라이언트(13)가 로컬 머신(10)상에서 설치되고 실행되면, 애플리케이션 서버(30")는 애플리케이션을 실행하고 애플리케이션 클라이언트(13)상에 애플리케이션을 디스플레이한다(단계 365). 선택적인 실시예에서, 애플리케이션 서버(30")는 애플리케이션을 포함하는 다수의 애플리케이션 파일을 로컬 머신(10)상에서의 실행을 위해 애플리케이션 클라이언트(13)에 전송한다(도 7을 참조하여 이하에서 상세히 설명됨). 다른 실시예에서, 로컬 머신(10)은 문서를 보고(애플리케이션 클라이언트(13)를 개시하기 전이라도) 애플리케이션 서버(30")로부터 GUI 애플리케이션을 얻기 위해 문서내의 정보를 사용한다. 이 실시예에서, GUI 애플리케이션의 디스플레이는 애플리케이션 클라이언트(30")의 설치 및 실행을 포함한다. 또한, 문서를 보는 것은 로컬 머신(10)의 사용자와 상관없을 수 있다. 예를 들어, 로컬 머신(10)은 웹 서버(30')로부터 문서를 수신할 수 있고 애플리케이션 서버(30")로 GUI 애플리케이션을 자동적으로 요청하기 이전에 문서를 해석(interpret)할 수 있다.
- <177> 따라서, 애플리케이션 클라이언트(13)는 퍼블리싱된 애플리케이션, 데스크탑, 데스크탑 문서, 그리고 애플리케이션 클라이언트(13)에 의해 지원되는 임의의 애플리케이션으로의 서비스-기반 액세스를 제공한다. 애플리케이션 클라이언트(13)가 액세스를 제공할 수 있는 애플리케이션의 예로는, 마이크로소프트사의 윈도우 데스크탑, 마이크로소프트 엑셀, 워드, 파워포인트와 같은 윈도우 문서, 캘리포니아 팔로 알토 소재의 썬 마이크로시스템사의 SUN SOLARIS와 같은 Unix 데스크탑, 그리고 노스 캐롤라이나 더햄에 소재하는 레드햇의 GNU/Linux 를 포함한다.
- <178> 일 실시예에서, 로컬 머신이 애플리케이션에 액세스할 수 있는지 그리고 어떻게 액세스하는지에 대한 정책 엔진에 의한 결정에 대응하여, 로컬 머신(10)에 사용 가능한 다수의 애플리케이션 프로그램의 목록이 제공된다. 정책 엔진은 이러한 결정을 하기 이전에 로컬 머신에 대한 정보를 수집할 수 있다. 도 4a를 참조하여, 로컬 머신(10), 수집 에이전트(404), 정책 엔진(406), 정책 데이터베이스(408), 팜(38), 그리고 애플리케이션 서버(30')를 포함하는 컴퓨터 네트워크의 일 실시예가 도시되어 있다. 일 실시예에서, 정책 엔진(406)은 원격 머신(30)이다. 다른 실시예에서, 애플리케이션 서버(30')는 원격 머신(30')이다. 도 4a에는 오직 하나의 로컬 머신(10), 수집 에이전트(404), 정책 엔진(406), 정책 데이터베이스(408), 팜(38), 그리고 애플리케이션 서버(30')가 도시되어 있지만, 시스템은 이러한 요소가 다수 제공할 수도 있다.
- <179> 간단히 설명하면, 로컬 머신(10)이 애플리케이션에 대한 액세스를 위한 요청(410)을 정책 엔진(406)에 전송하면, 로컬 머신(10)에 대한 정보를 검색하는, 수집 에이전트(404)는 로컬 머신(10)과 통신하고, 로컬 머신 정보(412)를 정책 엔진(406)에 전송한다. 정책 엔진(406)은 정책 데이터베이스(408)로부터 정책을 검색된 정보(412)에 적용하여 액세스 제어 결정을 만든다.
- <180> 좀더 구체적으로, 로컬 머신(10)은 리소스에 대한 요청(410)을 정책 엔진(406)에 전송한다. 일 실시예에서, 정책 엔진(406)은 애플리케이션 서버(30')상에 존재한다. 다른 실시예에서, 정책 엔진(406)은 원격 머신(30)이다. 다른 실시예에서, 애플리케이션 서버(30')는 로컬 머신(10)으로부터 요청(410)을 수신하고 이 요청(410)을 정책 엔진(406)에 전송한다. 다른 실시예에서, 로컬 머신은 리소스에 대한 요청(410)을 원격 머신(30")에 전송하고, 원격 머신(30")은 정책 엔진(406)에 요청(410)을 전송한다.
- <181> 요청을 수신하면, 정책 엔진(406)은 수집 에이전트(404)에 의한 정보 수집을 개시한다. 수집 에이전트(404)는 로컬 머신(10)에 대한 정보를 수집하고 이 정보(412)를 정책 엔진(406)에 전송한다.
- <182> 일 실시예에서, 수집 에이전트(404)는 네트워크 연결을 통해 정보를 수집하고 전송한다. 일 실시예에서, 수집 에이전트(404)는 바이트코드 프로그래밍 언어 JAVA 로 기록된 애플리케이션과 같이, 바이트코드를 포함한다. 일 실시예에서, 수집 에이전트(404)는 적어도 하나의 스크립트를 포함한다. 이 예에서, 수집 에이전트(404)는 로컬 머신(10)상의 적어도 하나의 스크립트를 실행함으로써 정보를 수집한다. 일 실시예에서, 수집 에이전트는 로컬 머신(10)상의 Active X control 을 포함한다. Active X control 은 control과 같이 보이게 하고 동작시킬 수 있는 인터페이스 집합을 실행시키는 COM(Component Object Model) 오브젝트이다.
- <183> 일 실시예에서, 정책 엔진(406)은 수집 에이전트(404)를 로컬 머신(10)에 전송한다. 다른 실시예에서, 어플라이언스(1250)는 수집 에이전트를 저장 또는 캐시할 수 있다. 어플라이언스(1250)는 수집 에이전트를 로컬 머신(10)에 전송한다. 다른 실시예에서, 어플라이언스(1250)는 수집 에이전트의 전송을 인터셉트할 수 있다. 다른 실시예에서, 어플라이언스(1250)는 수집 에이전트의 제공을 가속할 수 있다. 일 실시예에서, 정책 엔진(406)은 수집 에이전트(404)가 정보(412)를 정책 엔진(406)에 전송한 이후에 수집 에이전트(404)의 두 번째 실행을 요구할 수 있다. 이 실시예에서, 정책 엔진(406)은 로컬 머신(10)이 특정 조건을 만족시키는지 여부를 결정하기에

불충분한 정보(412)를 가질 수 있다. 다른 실시예에서, 정책 엔진(406)은 수신된 정보(412)에 따라 수집 에이전트(404)의 다수의 실행을 요구할 수 있다.

- <184> 일 실시예에서, 정책 엔진(406)은 수집 에이전트(404)가 수집하는 정보의 유형을 결정하는 명령을 수집 에이전트(404)에 전송한다. 이 예에서, 시스템 관리자는 수집 에이전트(404)에 전송되는 명령을 정책 엔진(406)으로부터 구성할 수 있다. 이것은 수집된 정보의 유형을 넘는 보다 큰 콘텐츠를 제공한다. 수집된 정보의 유형을 넘는 보다 큰 콘텐츠 때문에, 이것은 또한 정책 엔진(406)이 할 수 있는 액세스 콘텐츠를 결정의 유형을 확장시킨다. 수집 에이전트(404)는 로컬 머신(10)의 머신 ID, 운영체제 유형, 운영체제으로의 패치 존재, 설치된 네트워크 카드의 MAC 어드레스, 클라이언트 장치상의 디지털 워터마크, 액티브 디렉토리내의 멤버쉽, 바이러스 스캐너의 존재, 개인 방화벽의 존재, HTTP 헤더, 브라우저 유형, 장치 유형, 인터넷 프로토콜 어드레스와 같은 네트워크 연결 정보, 원격 머신(30)의 머신 ID, 액세스 요청 날짜 및 시간, 인증 자격을 포함하는 정보를 수집한다. 일 실시예에서, 수집 에이전트는 애플리케이션이 가속 프로그램(6120)을 사용하여 클라이언트상에서 가속될 수 있는지를 결정하기 위해 정보를 수집한다.
- <185> 일 실시예에서, 장치 유형은 개인용 정보 단말기이다. 다른 실시예에서, 장치 유형은 셀룰러폰이다. 다른 실시예에서, 장치 유형은 랩탑 컴퓨터이다. 다른 실시예에서, 장치 유형은 데스크탑 컴퓨터이다. 다른 실시예에서, 장치 유형은 인터넷 키오스크(Internet kiosk)이다.
- <186> 일 실시예에서, 디지털 워터마크는 데이터 임베딩을 포함한다. 일 실시예에서, 워터마크는 파일에 대한 소스 정보를 제공하기 위해 파일에 삽입된 데이터 패턴을 포함한다. 다른 실시예에서, 템퍼 검출(temperature detection)을 제공하기 위해 데이터 해싱 파일을 포함한다. 다른 실시예에서, 워터마크는 파일에 대한 복사라이트 정보를 제공한다.
- <187> 일 실시예에서, 네트워크 연결 정보는 대역폭 용량(bandwidth capability)과 관련된다. 다른 실시예에서, 네트워크 연결 정보는 인터넷 프로토콜 어드레스와 관련된다. 다른 실시예에서, 네트워크 연결 정보는 인터넷 프로토콜 어드레스로 구성된다. 일 실시예에서, 네트워크 연결 정보는 인증 자격이 제공된 로컬 머신으로의 로그인 에이전트를 식별하는 네트워크 존(network zone)을 포함한다.
- <188> 일 실시예에서, 인증 자격은 사용자 명칭, 클라이언트 명칭, 클라이언트 어드레스, 패스워드, PINs, 보이스 샘플, 온-타임 패스코드, 생물학적 데이터, 디지털 증명, 티켓 등과 이들의 조합을 포함하는 많은 인증 정보 유형을 포함한다. 수집된 정보(412)를 수신한 이후, 정책 엔진(406)은 수신된 정보(412)에 근거하여 액세스 제어 결정을 만든다.
- <189> 도 4b를 참조하여, 조건 데이터베이스(422) 및 로그인 에이전트(424)를 포함하는 제1 컴포넌트(420)와 정책 데이터베이스(432)를 포함하는 제2 컴포넌트(430)를 포함하는, 정책 엔진(406)의 일 실시예가 나타나있다. 제1 컴포넌트(420)는 조건 데이터베이스(422)로부터의 조건을 로컬 머신(10)에 대해 수신된 정보에 적용하고 수신된 정보가 조건을 만족시키는지 여부를 판단한다.
- <190> 일 실시예에서, 조건은 로컬 머신(10)이 특정 운영체제를 실행하는 것을 요구할 수 있다. 일 실시예에서, 조건은 로컬 머신(10)이 특정 운영체제 패치를 실행하는 것을 요구할 수 있다. 다른 실시예에서, 조건은 로컬 머신(10)이 각각의 설치된 네트워크 카드에 대해 MAC 어드레스를 제공하는 것을 요구할 수 있다. 일 실시예에서, 조건은 로컬 머신(10)이 특정 액티브 디렉토리에 멤버쉽을 나타내는 것을 요구할 수 있다. 다른 실시예에서, 조건은 로컬 머신(10)이 바이러스 스캐너를 실행하는 것을 요구할 수 있다. 다른 실시예에서, 조건은 로컬 머신(10)이 개인 방화벽을 실행하는 것을 요구할 수 있다. 일 실시예에서, 조건은 로컬 머신(10)이 특정 장치 유형을 포함하는 것을 요구할 수 있다. 다른 실시예에서, 조건은 로컬 머신(10)이 특정 유형의 네트워크 연결을 구축하는 것을 요구할 수 있다.
- <191> 수신된 정보가 조건을 만족시키면, 제1 컴포넌트(420)는 조건에 대한 식별자(identifier)를 데이터 집합(426)에 저장한다. 일 실시예에서, 정보가 조건을 참(true)으로 만든다면, 수신된 정보는 조건을 만족시킨다. 예를 들어, 조건은 특정 운영체제가 설치될 것을 요구할 수 있다. 로컬 머신(10)이 이러한 운영체제를 구비한다면, 조건은 참이 되고 이를 만족시킨다. 다른 실시예에서, 정보가 조건을 거짓(false)으로 만든다면, 수신된 정보는 조건을 만족시킨다. 예를 들어, 조건은 로컬 머신(10)상에 스파이웨어가 존재하는지 여부를 확인할 수 있다. 로컬 머신(10)이 스파이웨어를 가지고 있지 않다면, 조건은 거짓이고 이를 만족시킨다.
- <192> 일 실시예에서, 로그인 에이전트(424)는 정책 엔진(406) 외부에 존재한다. 다른 실시예에서, 로그인 에이전트(424)는 정책 엔진(406)상에 존재한다. 일 실시예에서, 제1 컴포넌트(420)는 로컬 머신(10)에 대한 정보 수집

을 개시하는 로그온 에이전트(424)를 포함한다. 일 실시예에서, 로그온 에이전트(424)는 데이터 저장소를 더 포함한다. 이 실시예에서, 데이터 저장소는 수집 에이전트가 정보를 수집하기 위한 조건을 포함한다. 이러한 데이터 저장소는 조건 데이터베이스(422)와 다르다.

<193> 일 실시예에서, 로그온 에이전트(424)는 수집 에이전트(404)를 실행함으로써 정보 수집을 개시한다. 다른 실시예에서, 로그온 에이전트(424)는 로컬 머신(10)상에서의 실행을 위해 로컬 머신(10)에 수집 에이전트(404)를 전송함으로써 정보 수집을 개시한다. 다른 실시예에서, 로그온 에이전트(424)는 정보(412)를 수신한 이후에 추가적인 정보 수집을 개시한다. 일 실시예에서, 로그온 에이전트(424)는 또한 정보(412)를 수신한다. 이 예에서, 로그온 에이전트(424)는 수신된 정보(412)에 근거하여 데이터 집합(426)을 생성한다. 일 실시예에서, 로그온 에이전트(424)는 데이터베이스(422)로부터의 조건을 수집 에이전트(404)로부터 수신된 정보에 적용함으로써 데이터 집합을 생성한다.

<194> 다른 실시예에서, 제1 컴포넌트(420)는 다수의 로그온 에이전트(424)를 포함한다. 이 실시예에서, 다수의 로그온 에이전트(424) 중 적어도 하나는 로컬 머신(10)이 리소스 요청을 전송할 수 있는 각각의 네트워크 도메인상에 존재한다. 이 예에서, 로컬 머신(10)은 리소스 요청을 특정 로그온 에이전트(424)에 전송한다. 일 실시예에서, 로그온 에이전트(424)는 로그온 에이전트(424)에 액세스된 로컬 머신(10)으로부터 네트워크 도메인을 정책 엔진(406)에 전송한다. 일 실시예에서, 로그온 에이전트(424)에 액세스하는 로컬 머신(10)으로부터의 네트워크 도메인은 로컬 머신(10)의 네트워크 존(network zone)이라 한다.

<195> 조건 데이터베이스(422)는 제1 컴포넌트(420)가 수신된 정보에 적용하는 조건들을 저장한다. 정책 데이터베이스(432)는 제2 컴포넌트(430)가 수집된 데이터 집합(426)에 적용하는 정책들을 저장한다. 일 실시예에서, 조건 데이터베이스(422) 및 정책 데이터베이스(432)는 ODBC-호환 데이터베이스에 데이터를 저장한다. 예를 들어, 조건 데이터베이스(422) 및 정책 데이터베이스(432)는 캘리포니아, 레드우드 쇼어에 소재하는 오라클 사의 ORACLE 데이터베이스로서 제공될 수 있다. 다른 실시예에서, 조건 데이터베이스(422) 및 정책 데이터베이스(432)는 워싱턴, 레드몬드 소재하는 마이크로소프트사의 Microsoft ACCESS 데이터베이스 또는 Microsoft SQL서버 데이터베이스가 될 수 있다.

<196> 제1 컴포넌트가 조건 데이터베이스(422)의 각각 조건에 수신된 정보를 적용시킨 이후에, 제1 컴포넌트는 데이터 집합(426)을 제2 컴포넌트(430)에 전송한다. 일 실시예에서, 제1 컴포넌트(420)는 오직 데이터 집합(426)만을 제2 컴포넌트(430)에 전송한다. 따라서, 이 실시예에서, 제2 컴포넌트(430)는 정보(412)를 수신하지 않고, 오직 만족된 조건에 대한 식별자만을 수신한다. 제2 컴포넌트(430)는 데이터 집합(426)을 수신하고 데이터 집합(426)내에 식별된 조건들에 근거하여 정책 데이터베이스(432)로부터의 정책을 적용함으로써 액세스 제어 결정(access control decision)을 한다.

<197> 일 실시예에서, 정책 데이터베이스(432)는 수신된 정보(412)에 적용된 정책들을 저장한다. 일 실시예에서, 정책 데이터베이스(432)에 저장된 정책들은 시스템 관리자에 의해 적어도 일부가 지정된다(specified). 다른 실시예에서, 사용자는 정책 데이터베이스(432)에 저장된 정책들 중 적어도 일부를 지정한다. 사용자-지정된 정책 또는 정책들은 우선권(preference)으로서 저장된다. 정책 데이터베이스(432)는 휘발성 또는 비휘발성 메모리에 저장되거나 또는 다중 서버를 통해 분배될 수 있다.

<198> 일 실시예에서, 정책은 하나 이상의 조건이 만족되는 경우에만 리소스에 대한 액세스를 허용한다. 다른 실시예에서, 정책은 로컬 머신(10)으로의 리소스 전송을 금지하지만 리소스에 대한 액세스를 허용한다. 다른 정책은 보안 네트워크내에서의 액세스를 요구하는 로컬 머신(10)에 부수하는 연결을 만들 수 있다. 일 실시예에서, 리소스는 애플리케이션 프로그램이고 로컬 머신(10)은 상기 애플리케이션 프로그램의 실행을 요청했다. 이 실시예에서, 정책은 로컬 머신(10)상에서 애플리케이션 프로그램의 실행을 허용할 수 있다. 정책은 로컬 머신(10)이 애플리케이션 프로그램을 포함하는 파일들의 스트림을 수신하도록 할 수 있다. 이 예에서, 파일들의 스트림은 분리된 환경에서 저장되고 실행될 수 있다. 정책은 애플리케이션 서버와 같은 원격 기계 상에서만 애플리케이션 프로그램이 실행하도록 허용할 수 있고, 원격 머신이 애플리케이션 출력 데이터를 로컬 머신(10)에 전송하도록 요구할 수 있다.

<199> 도 4c를 참조하여, 정책 엔진이 로컬 머신(10)에 대해 수신된 정보에 근거하여 액세스 제어 결정을 만드는 단계들의 일 실시예를 나타내고 있다. 로컬 머신(10)에 대해 수집된 정보를 수신하면(단계 450), 정책 엔진(406)은 이 정보에 근거하여 데이터 집합을 생성한다(단계 452). 데이터 집합(426)은 수신된 정보에 의해 만족된 각각의 조건에 대한 식별자를 포함한다. 정책 엔진(406)은 데이터 집합(426)내의 각각의 식별된 조건에 정책을 적용한다. 애플리케이션은 로컬 머신(10)이 액세스할 수 있는 리소스들의 목록을 산출한다(단계 454). 정책 엔

진(406)은 로컬 머신(10)에 목록을 제공한다. 일 실시예에서, 정책 엔진은 로컬 머신에 목록을 제공하는데 사용되는 HTML 문서를 생성한다.

<200> 도 4d를 참조하여, 로컬 머신(10), 수집 에이전트(404), 정책 엔진(406), 정책 데이터베이스(408), 조건 데이터베이스(410), 로컬 머신(20), 세션 서버(420), 저장된 애플리케이션 데이터베이스(422), 원격 머신(30'), 제1 데이터베이스(428), 원격 머신(30"), 제2 데이터베이스(432)를 포함하는 네트워크의 일 실시예가 나타나 있다. 간단히 설명하면, 로컬 머신(10)이 애플리케이션 프로그램으로의 액세스 요청(412)을 액세스 제어 서버(406)에 전송하면, 수집 에이전트(404)는 로컬 머신(10)과 통신하여 로컬 머신(10)에 대한 정보를 검색하고, 로컬 머신 정보(414)를 정책 엔진(406)에 전송한다. 정책 엔진(406)은 도 4a 및 4b에서 설명한 바와 같이 액세스 제어 결정을 만든다. 로컬 머신(10)은 로컬 머신(10)과 연관된 사용가능한 애플리케이션의 목록을 수신한다.

<201> 일 실시예에서, 세션 서버(420)는 로컬 머신(10)과 로컬 머신(10)과 연관된 다수의 애플리케이션 세션간의 연결을 구축한다. 다른 실시예에서, 정책 엔진(406)은 로컬 머신(10)이 애플리케이션을 포함하는 다수의 애플리케이션 파일들을 검색하고 애플리케이션 프로그램을 국부적으로(locally) 실행할 수 있는 인증을 갖는지를 결정한다. 일 실시예에서, 정책 엔진(406)은 가속 프로그램(6120)을 로컬 머신(10)에 전송함으로써 애플리케이션 파일들의 제공을 가속할지를 결정한다. 이 예에서, 원격 머신(30')은 애플리케이션 세션 데이터와 애플리케이션 프로그램을 포함하는 다수의 애플리케이션 파일들을 저장한다. 이 예에서, 로컬 머신(10)은 애플리케이션 세션 데이터와 애플리케이션 프로그램을 포함하는 다수의 애플리케이션 파일들을 저장하는 원격 머신(30')과 애플리케이션 스트리밍 세션을 구축한다. 일 실시예에서, 정책 엔진(406)은 가속 프로그램(6120)을 로컬 머신(10)에 전송함으로써 스트리밍 세션의 제공을 가속할지를 결정한다. 일 실시예에서, 정책 엔진(406)은 가속 프로그램(6120)을 로컬 머신(10)에 전송함으로써 데이터의 전송을 가속할지를 결정한다.

<202> 도 4e를 참조하여, 로컬 머신(10)과 연관된 애플리케이션 세션에 로컬 머신(10)의 액세스를 제공하기 위해 세션 서버(420)에 의해 이뤄지는 단계들의 일 실시예를 나타낸다. 세션 서버(420)는 액세스 제어 결정을 갖는 정책 엔진(406)으로부터 로컬 머신(10)에 대한 정보를 수신한다(단계 480). 세션 서버(420)는 연관된 애플리케이션의 목록을 생성한다(단계 482). 세션 서버(420)는 로컬 머신(10)을 연관된 애플리케이션에 연결한다(단계 484). 일 실시예에서, 정보는 또한 로컬 머신 정보(414)를 포함한다. 다른 실시예에서, 정보는 애플리케이션 프로그램을 국부적으로 실행할 수 있는 인증을 포함한다.

<203> 세션 서버(420)는 연관된 애플리케이션의 목록을 생성한다(단계 482). 일 실시예에서, 정책 엔진(406)은 이미 로컬 머신(10)과 연관된 다수의 애플리케이션 세션을 식별한다. 다른 실시예에서, 세션 서버(420)는 로컬 머신(10)과 연관된 저장된 애플리케이션 세션을 식별한다. 이 예에서, 정책 엔진(406)으로부터 정보를 수신하는 것에 따라 세션 서버(420)는 자동적으로 저장된 애플리케이션 세션을 식별한다. 일 실시예에서, 저장된 애플리케이션 데이터베이스(422)는 세션 서버(420)상에 존재한다. 다른 실시예에서, 저장된 애플리케이션 데이터베이스(422)는 정책 엔진(406)상에 존재한다.

<204> 저장된 애플리케이션 데이터베이스(422)는 애플리케이션 세션을 실행하거나 애플리케이션 세션 데이터 및 애플리케이션 프로그램을 포함하는 애플리케이션 파일들에 대한 액세스를 제공하는 팜(38)내의 다수의 원격 머신과 연관된 데이터를 갖는다. 일 실시예에서, 로컬 머신(10)과 연관된 애플리케이션 세션을 식별하는 것은 하나 이상의 원격 머신과 연관된 저장 데이터를 참조하는 것을 요한다. 이 예에서, 세션 저장소(420)는 하나 이상의 원격 머신과 연관된 저장 데이터를 참조한다. 다른 예로, 정책 엔진(406)이 하나 이상의 원격 머신과 연관된 저장 데이터를 참조한다. 일 실시예에서, 제1 애플리케이션 세션은 원격 머신(30')상에서 실행되고 제2 애플리케이션 세션은 원격 머신(30")상에서 실행된다. 다른 실시예에서, 모든 애플리케이션 세션은 팜(38)내의 단일 원격 머신(30)상에서 실행된다.

<205> 세션 서버(420)는 사용자에게 의해 개시된 애플리케이션 세션들에 대한 정보를 포함한다. 세션 서버는 휘발성 또는 비휘발성 메모리내에 저장되거나 또는 다중 서버에 분산될 수 있다. 표 1은 세션 서버(420) 일부에 포함된 데이터를 나타낸다.

표 1

애플리케이션 세션	애플리케이션 세션1	애플리케이션 세션2	애플리케이션 세션3
사용자 ID	사용자 1	사용자 2	사용자 1
클라이언트 ID	제1 클라이언트		제1 클라이언트

클라이언트 어드레스	172.16.0.50		172.16.0.50
상태	활성	연결 해제	활성
애플리케이션	워드 프로세서	데이터베이스	스프레드시트
프로세스 넘버	1	3	2
서버	서버 A	서버 A	서버 B
어드레스	172.16.2.55	172.16.2.55	172.16.2.56

<207> 표 1에 도시된 세션 서버(420)는 각각의 애플리케이션 세션을 애플리케이션 세션을 개시한 사용자와 연관시키는 데이터, 클라이언트 컴퓨터(10 또는 20)의 ID, 그리고 클라이언트 컴퓨터(10 또는 20)의 IP 어드레스를 포함한다. 세션 서버(420)는 또한 각각의 애플리케이션 세션의 상태를 포함한다. 애플리케이션 세션 상태는 예를 들어, "활성(active)"(사용자가 애플리케이션 세션에 연결되었음을 의미함) 또는 "연결해제(disconnected)"(사용자가 애플리케이션 세션에 연결되어 있지 않음을 의미함)가 될 수 있다. 선택적인 실시예에서, 애플리케이션 세션 상태는 "실행-연결해제(executing-disconnected)"(사용자는 애플리케이션 세션으로부터 연결해제되었지만 애플리케이션 세션 내의 애플리케이션은 여전히 실행중임을 의미함) 또는 "스톨-연결해제(stalled-disconnected)"(사용자는 연결해제되었으며 애플리케이션 세션내의 애플리케이션을 실행되지 않으나, 연결해제 바로 이전의 이들의 동작 상태가 저장되었음을 의미함)가 될 수 있다. 세션 서버(420)는 각각의 애플리케이션 세션내에서 실행되는 애플리케이션(116)을 나타내는 정보와 서버상의 애플리케이션의 프로세스를 나타내는 데이터를 더 포함한다. 원격 머신(30')이 팜(38)의 일부인 실시예에서, 세션 서버(420)는 동적 저장소의 적어도 일부이며, 또한 애플리케이션이 실행되는/실행되었던 팜(38)내의 원격 머신(30)을 나타내는 표 1의 마지막 2 줄 내의 데이터와 원격 머신(30)의 IP 어드레스를 포함한다. 선택적인 실시예에서, 세션 서버(420)는 각 애플리케이션 세션내의 각각의 애플리케이션에 대한 상태 표시자를 포함한다.

<208> 예를 들어, 표 1의 예에서, 3개의 애플리케이션 세션이 존재한다. 애플리케이션 세션 1은 현재 터미널 1을 사용하는 사용자와 연관된다. 터미널 1의 IP 어드레스는 152.16.2.50 이다. 애플리케이션 세션 1의 상태는 활성이고, 애플리케이션 세션 1에서 워드 처리 프로그램이 동작되고 있다. 서버 A의 IP 어드레스는 152.16.2.55 이다. 표 1의 애플리케이션 세션 2는 연결해제된 애플리케이션 세션(118)의 예이다. 애플리케이션 세션 2는 사용자 2와 연관되나, 애플리케이션 세션 2는 로컬 머신(10 또는 20)에 연결되지 않았다. 애플리케이션 세션 2는 프로세스 넘버 3으로 IP 어드레스 152.16.2.55 서버 A상에 실행되는 데이터베이스 프로그램을 포함한다. 애플리케이션 세션 3은 사용자가 어떻게 다른 원격 머신(30)상에서 동작하는 애플리케이션 세션과 상호작용하는지에 대한 예이다. 애플리케이션 세션 3은 애플리케이션 세션 1과 같이 사용자 1과 연관된다. 애플리케이션 세션 1에 포함된 애플리케이션 세션은 서버 A상에서 실행되는 반면에, 애플리케이션 세션 3은 프로세스 넘버 2로 IP 어드레스 152.16.2.56 서버 B상에서 실행되는 스프레드시트 프로그램을 포함한다.

<209> 다른 예에서, 국부적 실행을 위해 제2 원격 머신(30")으로부터 제2 애플리케이션 프로그램을 검색하기 위해 서버 B 와 같은 제2 원격 머신(30")과 애플리케이션 스트리밍 세션을 통신하면서, 사용자는 서버 A와 같은 원격 머신(30')상에서 실행되는 애플리케이션 세션을 통해 제1 애플리케이션 프로그램에 액세스할 수 있다. 로컬 머신(10)의 사용자는 제1 애플리케이션 프로그램의 실행을 위해 필요한 요건을 만족시키지 못하더라도 제2 애플리케이션 프로그램을 국부적으로 실행시키기 위해 요구되는 인증을 가질 수 있다.

<210> 일 실시예에서, 세션 서버(420)는 로컬 머신(10)과 연관된 애플리케이션 세션을 연결해제시키기 위한 연결해제 요청을 수신하고 이 요청에 대응하여 애플리케이션 세션을 연결해제시킨다. 세션 서버(420)는 애플리케이션 세션으로부터 로컬 머신(10)을 연결해제시킨 이후에 애플리케이션 세션 실행을 계속한다. 이 실시예에서, 세션 서버(420)는 저장된 애플리케이션 데이터베이스(422)에 액세스하고 각각의 연결해제된 애플리케이션 세션과 연관된 데이터 기록을 업데이트한다. 이 기록은 로컬 머신(10)과 연관된 애플리케이션 세션이 연결해제되었다는 것을 나타낸다.

<211> 네트워크에 연결된 로컬 머신과 연관된 인증 정보를 수신한 이후, 세션 서버(420)는 로컬 머신의 사용자와 연관되나 다른 로컬 머신(예를 들어, 인증 정보가 로컬 머신(20)과 연관된다면 로컬 머신(10)과 같은)에 연결된 임의의 활성 애플리케이션 세션을 식별하기 위해 저장된 애플리케이션 데이터베이스(422)를 참조한다. 일 실시예에서, 세션 서버(420)가 임의의 활성 애플리케이션 세션을 식별하면, 세션 서버(420)는 자동적으로 로컬 머신(10)으로부터 애플리케이션 세션(들)을 연결해제시키고 애플리케이션 세션(들)을 로컬 머신(20)에 연결시킨다. 일 실시예에서, 수신된 인증 정보는 로컬 머신(10)이 재연결될 수 있는 애플리케이션 세션을 제한할 것이다. 다른 실시예에서, 수신된 인증 정보는 로컬 머신(20)상에서 애플리케이션 프로그램이 실행되는 것을 인증하고,

상기 인증은 로컬 머신(10)에는 부인되었을 것이다. 이 예에서, 세션 서버(420)는 국부적 실행을 위한 애플리케이션 프로그램을 검색하기 위해 로컬 머신 액세스 정보를 제공할 수 있다.

<212> 열거된 애플리케이션을 실행하기 위해 요청이 수신된다(단계 206). 일 실시예에서, 로컬 머신(10)의 사용자는 사용가능한 애플리케이션의 목록으로부터 실행을 위한 애플리케이션을 선택한다. 다른 실시예에서, 사용자는 수신된 목록과 독립적으로 실행을 위한 애플리케이션을 선택한다. 일 실시예에서, 사용자는 클라이언트 에이전트에 의해 로컬 머신(10)상에 표현된 애플리케이션의 그래픽 표현을 선택하여 실행을 위한 애플리케이션을 선택한다. 다른 실시예에서, 사용자는 웹 서버 또는 다른 원격 머신(30')상에서 사용자에게 제공된 애플리케이션의 그래픽 표현을 선택하여 실행을 위한 애플리케이션을 선택한다. 일 실시예에서, 어플라이언스(1250) 또는 가속 프로그램(6120)은 그래픽 표현의 제공을 가속한다. 일 실시예에서, 어플라이언스(1250)는 그래픽 표현을 캐싱 또는 저장한다. 일 실시예에서, 어플라이언스는 연관된 어플리케이션 모두 또는 연관된 어플리케이션의 일부를 캐싱 또는 저장할 수 있다.

<213> 일 실시예에서, 사용자는 파일로의 액세스를 요청한다. 이 예에서, 파일로의 액세스를 사용자에게 제공하기 위해 애플리케이션의 실행이 요구된다. 일예에서, 액세스를 위한 파일의 선택에 따라 애플리케이션이 자동적으로 선택된다. 일예에서, 파일로의 액세스에 대한 요청에 앞서, 애플리케이션은 파일의 유형과 연관되며, 요청된 파일과 연관된 파일의 유형의 식별에 따라 애플리케이션의 자동 선택이 가능하게 한다. 일 실시예에서 어플라이언스(1250) 또는 가속 프로그램(6120)은 하나 이상의 파일의 제공을 가속하는데 사용될 수 있다. 일 실시예에서 어플라이언스(1250)는 파일의 일부 또는 전부를 캐싱 또는 저장한다.

<214> 일 실시예에서, 열거된 애플리케이션은 다수의 애플리케이션 파일을 포함한다. 일 실시예에서, 다수의 애플리케이션 파일은 원격 머신(30')상에 존재한다. 일 실시예에서, 다수의 애플리케이션 파일은 격리된 파일 서버 또는 원격 머신(30'')상에 존재한다. 다른 예에서, 다수의 애플리케이션 파일은 로컬 머신(10)으로 전송될 수 있다. 다른 예에서, 다수의 애플리케이션 파일내의 파일은 다수의 애플리케이션 파일내의 두 번째 파일의 로컬 머신(10)으로의 전송 이전에 실행될 수 있다. 일 실시예에서 어플라이언스(1250) 또는 가속 프로그램(6120)은 하나 이상의 애플리케이션 파일의 제공을 가속하는데 사용될 수 있다.

<215> 일 실시예에서, 원격 머신(30)은 열거된 애플리케이션에 대한 정보를 원격 머신(30')으로부터 검색한다. 이 예에서, 원격 머신(30)은 다수의 애플리케이션 파일을 호스팅하는 원격 머신(30'')의 ID(identification)를 수신한다. 다른 예에서, 원격 머신(30)은 다수의 애플리케이션 파일의 위치의 ID를 수신하며, 상기 ID는 UNC(Universal Naming Convention)에 따른다. 다른 예에서, ID는 네트워크 위치와 애플리케이션 스트리밍 프로토콜에 대한 소켓을 포함한다.

<216> 일 실시예에서, 원격 머신(30)은 열거된 애플리케이션에 대한 정보를 갖는 파일을 검색한다. 파일은 열거된 애플리케이션을 호스팅하는 서버의 위치의 ID를 포함할 수 있다. 파일은 열거된 애플리케이션의 다양한 버전에 대한 ID를 포함할 수 있다. 파일은 열거된 애플리케이션을 포함하는 다수의 애플리케이션 파일의 목록을 포함할 수 있다. 파일은 열거된 애플리케이션을 포함하는 다수의 애플리케이션 파일을 포함하는 압축된 파일의 ID를 포함할 수 있다. 파일은 열거된 애플리케이션을 실행하는 머신에 의해 만족되어야 하는 조건(prerequisite)의 ID를 포함할 수 있다. 파일은 열거된 애플리케이션과 연관된 데이터 파일의 목록을 포함할 수 있다. 파일은 열거된 애플리케이션을 실행하는 머신상에서 실행될 스크립트의 목록을 포함할 수 있다. 파일은 열거된 애플리케이션과 연관된 레지스트리 데이터의 목록을 포함할 수 있다. 파일은 열거된 애플리케이션이 격리된 환경내에서 실행하는 실시예에서 사용하기 위한 규칙들(rules)의 목록을 포함할 수 있다. 일 실시예에서, 파일은 "일목 요연한(manifest)" 파일로서 언급될 수 있다. 파일이 가질 수 있는 정보는 도 21을 참조로 하여 이하에서 더 자세히 설명된다.

<217> 일 실시예에서, 원격 머신(30)은 로컬 머신(10)의 식별된 특성에 정책을 적용한다. 이 예에서, 원격 머신(30)은 식별된 특성에 대응하는 실행을 위해 열거된 애플리케이션의 버전을 식별한다. 다른 예로, 원격 머신(30)은 로컬 머신(10)의 특성과 호환될 수 있는 열거된 애플리케이션의 버전을 실행하기 위한 결정을 만든다. 다른 예로, 원격 머신(30)은 로컬 머신(10)상에서 실행되는 운영체제와 호환될 수 있는 열거된 애플리케이션의 버전을 실행하기 위한 결정을 만든다. 다른 예로, 원격 머신(30)은 로컬 머신(10)상의 운영체제의 수정(revision) 레벨과 호환될 수 있는 열거된 애플리케이션의 버전을 실행하기 위한 결정을 만든다. 이 예에서, 원격 머신(30)은 로컬 머신(10)상의 운영체제에 의해 특정된 언어와 호환될 수 있는 열거된 애플리케이션의 버전을 실행하기 위한 결정을 만든다.

<218> 정책에 대응하여, 열거된 애플리케이션을 실행하기 위해 미리 결정된 수의 방법들 중 하나가 선택되고, 미리 결

정된 수의 방법들은 열거된 애플리케이션의 애플리케이션 스트리밍 방법을 포함한다(단계 208). 일 실시예에서, 이러한 선택은 로컬 머신(10)과 연관되어 수신된 자격에 대한 정책에 대응하여 만들어진다. 일 실시예에서, 이러한 선택은 도 4a, 4b, 4c 를 통해 전송된 정책 엔진(406)과 같은 정책 엔진에 의해 만들어진다. 다른 예에서, 자격과 열거된 애플리케이션 실행에 대한 요청을 수신하는 원격 머신(30)은 정책 엔진(406)과 같은 것을 더 포함한다.

<219> 일 실시예에서, 미리 결정된 수의 방법들은 원격 머신(30')상에서 열거된 애플리케이션을 실행하기 위한 방법을 포함한다. 다른 예에서, 미리 결정된 수의 방법들은 로컬 머신(10)상에서 열거된 애플리케이션을 실행하기 위한 방법을 포함한다. 다른 예에서, 미리 결정된 수의 방법들은 두 번째 원격 머신(30')상에서 열거된 애플리케이션을 실행하기 위한 방법을 포함한다.

<220> 일 실시예에서, 미리 결정된 수의 방법들은 애플리케이션 스트리밍 세션을 가로질러(across) 로컬 머신(10)에 열거된 애플리케이션을 제공하기 위한 방법을 포함한다. 이 예에서, 로컬 머신(10)은 원격 머신(30')과의 연결을 개시할 수 있고 원격 머신(30')으로부터 전송된 데이터 패킷의 스트림을 수신할 수 있는 스트리밍 서비스 에이전트를 포함한다.

<221> 데이터 패킷의 스트림은 열거된 애플리케이션을 포함하는 애플리케이션 파일을 포함할 수 있다. 일 실시예에서, 애플리케이션 파일을 애플리케이션 프로그램과 연관된 데이터 파일을 포함한다. 다른 예에서, 애플리케이션 파일은 애플리케이션 프로그램의 실행을 위해 요구되는 실행가능 파일을 포함한다. 다른 예에서, 애플리케이션 파일은 위치, 호환 조건, 구성 데이터, 레지스트리 데이터, 격리된 환경에서의 사용을 위한 실행 스크립트 규칙의 ID, 또는 인증 조건과 같은 파일에 대한 정보를 포함하는 메타데이터를 포함한다. 일 실시예에서, 데이터 패킷의 스트림은 TCP/IP 패킷의 페이로드와 같은 전송 레이어 연결을 통해 전송될 수 있다.

<222> 일 실시예에서, 스트리밍된 애플리케이션을 포함하는 다수의 애플리케이션 파일내의 각각의 애플리케이션 파일의 전송이전에 스트리밍된 애플리케이션이 실행된다. 이 예에서, 스트리밍된 애플리케이션의 실행은 로컬 머신(10)이 다수의 애플리케이션내의 하나의 애플리케이션 파일을 수신하는 것에 따라 시작된다. 다른 예로, 스트리밍된 애플리케이션의 실행은 로컬 머신(10)이 다수의 애플리케이션 파일내의 실행가능한 애플리케이션 파일을 수신하는 것에 따라 시작된다. 다른 예로, 로컬 머신(10)은 다수의 애플리케이션 파일내에서 첫번째 수신된 애플리케이션 파일을 실행하고 첫번째 수신된 애플리케이션 파일은 다수의 애플리케이션 파일내의 두번째 애플리케이션 파일에 대한 액세스를 요청한다.

<223> 일 실시예에서, 스트리밍된 애플리케이션은 로컬 머신(10)상에서 영구적으로 존재할 필요없이 로컬 머신(10)상에서 실행된다. 이 예에서, 스트리밍된 애플리케이션은 로컬 머신(10)상에서 실행될 수 있고 스트리밍된 애플리케이션의 종료에 따라 로컬 머신(10)으로부터 제거될 수 있다. 다른 예에서, 각각의 애플리케이션 파일의 미리 배치된(pre-deployed) 복사본이 로컬 머신(10)상에 저장된 이후에 로컬 머신(10)상에서 스트리밍된 애플리케이션이 실행된다. 다른 예로, 각각의 애플리케이션 파일의 복사본이 로컬 머신상의 격리된 환경내에 저장된 이후에 스트리밍된 애플리케이션이 로컬 머신(10)상에서 실행된다. 다른 예로, 각각의 애플리케이션 파일의 복사본이 로컬 머신(10)상의 캐시에 저장된 이후에 스트리밍된 애플리케이션이 로컬 머신(10)상에서 실행된다.

<224> 일 실시예에서, 애플리케이션을 로컬 머신(10)에 스트리밍하기 위한 방법은 로컬 머신(10)이 스트리밍된 애플리케이션 파일을 수신할 수 있다는 결정에 대응하여 미리 결정된 수의 방법들로부터 선택된다. 다른 예에서, 애플리케이션을 로컬 머신(10)에 스트리밍하기 위한 방법은 로컬 머신(10)이 스트리밍된 애플리케이션 파일을 국부적으로 실행할 수 있는 인증을 갖는다는 결정에 대응하여 미리 결정된 수의 방법들로부터 선택된다.

<225> 다른 예에서, 미리 결정된 수의 방법들은 애플리케이션 출력 데이터를 로컬 머신(10)에 제공하기 위한 방법을 포함하고, 애플리케이션 출력 데이터는 원격 머신(30)상에서 열거된 애플리케이션의 실행으로부터 생성된다. 이 예에서, 원격 머신(30)은 열거된 애플리케이션의 실행을 위한 요청을 수신하는 원격 머신(30)이다. 다른 예로, 원격 머신(30)은 파일 서버 또는 애플리케이션 서버와 같은 제2 원격 머신(30')이다. 일 실시예에서, 열거된 애플리케이션은 열거된 애플리케이션을 실행하는 원격 머신(30')상에 존재한다. 다른 예에서, 열거된 애플리케이션을 실행하는 원격 머신(30')은 애플리케이션 스트리밍 세션을 가로질러 제2 원격 머신(30')으로부터 열거된 애플리케이션을 수신한다. 이 예에서, 원격 머신(30')은 제2 원격 머신(30')과의 연결을 개시하고 제2 원격 머신(30')으로부터 전송된 데이터의 스트림을 수신할 수 있는 스트리밍 서비스 에이전트를 포함한다. 다른 예로, 제2 원격 머신(30')은 로드 밸런싱 기술(load balancing technique)을 사용하여 식별될 수 있다. 다른 예로, 제2 원격 머신(30')은 원격 머신(30')에 대한 근접성을 근거로하여 식별될 수 있다. 이러한 예들은 도 9 를 참조하여 이하에서 더 자세히 설명된다.

- <226> 일 실시예에서, 원격 머신(30)은 열거된 애플리케이션을 실행하기 위한 미리 결정된 수의 방법들 중에서, 열거된 애플리케이션을 원격 머신(30)에 스트리밍하고, 원격 머신(30)상에서 열거된 애플리케이션을 실행하고, 열거된 애플리케이션의 실행에 의해 생성된 애플리케이션 출력 데이터를 로컬 머신(10)에 제공하는 방법을 선택한다. 이 예에서, 원격 머신(30)은 로컬 머신(10)의 평가(evaluation)에 대응하여 상기 방법을 선택한다. 다른 예로, 이러한 결정은 로컬 머신(10)의 평가에 대한 정책에 대응하여 이루어진다. 다른 예로, 이러한 결정은 수신된 자격의 평가에 대응하여 이루어진다. 일 실시예에서, 원격 머신(30)은 열거된 애플리케이션을 포함하는 다수의 애플리케이션 파일을 수신한다. 다른 예에서, 원격 머신(30)은 ICA 표현 레벨 프로토콜 또는 Remote Desktop Windows 표현 레벨 프로토콜 또는 X-Windows 표현 레벨 프로토콜과 같은 표현 레벨 프로토콜을 통해 애플리케이션 출력 데이터를 제공한다.
- <227> 일 실시예에서, 원격 머신(30)은 또한 열거된 애플리케이션과 연관된 액세스 정보를 제공하며, 액세스 정보는 선택된 방법에 대응하여 생성된다. 이 예에서, 액세스 정보는 열거된 애플리케이션 프로그램의 실행을 위해 선택된 방법의 로컬 머신(10)에 표시(indication)를 제공한다. 다른 예로, 액세스 정보는 열거된 애플리케이션의 위치의 ID를 포함하며, 이러한 ID는 UNC 에 따른다. 다른 예로, 액세스 정보는 세션 관리 서버의 ID 를 포함한다.
- <228> 일 실시예에서, 액세스 정보는 인증 정보를 포함하는 개시 티켓을 포함한다. 이 예에서, 로컬 머신(10)은 원격 머신(30)으로부터 수신된 액세스 정보를 인증하는데 개시 티켓을 사용할 수 있다. 다른 예로, 로컬 머신(10)은 열거된 애플리케이션을 호스팅하는 제2 원격 머신(30)에 대한 자신의 인증을 위해 개시 티켓을 사용할 수 있다. 다른 예로, 원격 머신(30)은 개시 티켓에 대한 로컬 머신(10)으로부터의 요청에 대응하여 액세스 정보내에 개시 티켓을 포함한다.
- <229> 도 5를 참조하여, 로컬 머신(10)이 애플리케이션 프로그램의 실행을 요청하고 원격 머신(30)을 포함하는 애플리케이션 제공 시스템(500)이 애플리케이션 프로그램을 실행하는 방법을 선택하는 실시예를 나타내고 있다. 일 실시예에서, 원격 머신(30)은 로컬 머신(10)으로부터 자격을 수신한다. 다른 예로, 원격 머신(30)은 로컬 머신(10)으로부터 가능한 애플리케이션들의 목록에 대한 요청을 수신한다.
- <230> 일 실시예에서, 다수의, 여분의 원격 머신들(30, 30', 30", 30'", 30'')이 제공된다. 이 예에서, 다수의 파일 서버, 다수의 세션 관리 서버, 다수의 스테이징 머신, 다수의 웹 인터페이스, 다수의 액세스 수트 콘솔이 있을 수 있다. 다른 예로, 어느 하나의 원격 머신이 실패하면, 여분의 원격 머신(30)이 실패된 머신의 기능을 제공하기 위해 선택될 수 있다. 다른 예에서, 원격 머신들(30, 30', 30", 30'", 30''), 웹 인터페이스(558) 그리고 액세스 수트 콘솔(520)은 관리 서버, 세션 관리 서버, 스테이징 머신, 파일 서버, 웹 서버, 액세스 수트 콘솔의 격리된 기능들을 갖는 격리된 원격 머신들(30)로서 설명되지만, 단일 원격 머신(30)이 이러한 머신들의 모든 기능을 갖는 것으로 제공될 수 있다. 다른 예에서, 원격 머신(30)은 하나 이상의 다른 원격 머신들의 기능 및 서비스를 제공할 수 있다.
- <231> 도 5를 참조하여 더 자세히 설명하면, 애플리케이션 프로그램에 대한 액세스를 제공하는 애플리케이션 제공 시스템(500)이 나타나있다. 애플리케이션 제공 시스템(500)은 하나 이상의 원격 머신(30), 어플라이언스(1250), 또는 이들의 임의의 조합을 포함할 수 있다. 도 1d와 연결하여 전술된 인터페이스 및 서브시스템에 추가적으로, 원격 머신(30)은 관리 통신 서비스(514), XML 서비스(516), 관리 서비스(504)를 더 포함할 수 있다. 관리 서비스(504)는 애플리케이션 관리 서브시스템(506), 서버 관리 서브시스템(508), 세션 관리 서브시스템(510), 라이선스 관리 서브시스템(512)을 포함할 수 있다. 원격 머신(30)은 액세스 수트 콘솔(520)과 연결될 수 있다.
- <232> 일 실시예에서, 관리 서비스(504)는 특수 원격 처리 호출 서브시스템, MetaFrame Remote Procedure Call(MRFRFC) 서브시스템(522)을 더 포함한다. 일 실시예에서, MRFRFC 서브시스템(522)은 XML 서비스(516)와 같은 원격 머신(30)상의 서브시스템과 관리 서비스(504) 사이의 통신을 라우팅한다. 다른 예에서, MRFRFC 서브시스템(522)은 관리 기능을 콜하기 위한 RPC(remote procedure call) 인터페이스를 제공하고, RPC 호출을 관리 서비스(504)에 제공하고, 그 결과를 호출한 서브시스템에 재전송한다.
- <233> 일 실시예에서, 원격 머신(30)은 도 4b에서 설명된 프로토콜 엔진(406)과 같은 프로토콜 엔진과 통신한다. 이 예에서, 원격 머신(30)은 원격 머신(30')상에 존재하는 프로토콜 엔진과 통신한다. 다른 예에서, 원격 머신(30)은 프로토콜 엔진(406)을 더 포함한다.
- <234> 원격 머신(30)은 액세스 수트 콘솔(520)과 통신할 수 있다. 액세스 수트 콘솔(520)은 관리 툴을 원격 머신(30)

또는 팜(38)의 관리자에게 호스팅할 수 있다. 일 실시예에서, 원격 머신(30)은 XML을 사용하여 액세스 수트 콘솔(520)과 통신한다. 다른 예에서, 원격 머신(30)은 SOAP(Simple Object Access Protocol)을 사용하여 액세스 수트 콘솔(520)과 통신한다.

<235> 도 1d 및 도 5를 참조하여 설명된 실시예에서, 원격 머신(30)은 서브시스템의 서브세트, 다수의 서브시스템을 포함할 수 있는 관리 서비스(504)를 포함한다. 일 실시예에서, 각각의 서브시스템은 단일-스레드 또는 다중-스레드 서브시스템이다. 스레드(thread)는 멀티-태스킹 환경에서 실행되는 독립적인 실행 스트림이다. 단일-스레드 서브시스템은 동시에 오직 하나의 스레드를 실행할 수 있다. 다중-스레드 서브시스템은 스레드들의 다수의 동시 실행을 지원한다, 즉 다중-스레드 서브시스템은 동시에 다수의 태스크를 수행할 수 있다.

<236> 애플리케이션 관리 서브시스템(506)은 스트림될 수 있는 다수의 애플리케이션과 연관된 정보를 관리한다. 일 실시예에서, 애플리케이션 관리 서브시스템(506)은 애플리케이션을 저장, 삭제, 업데이트, 목록, 결정하는 것에 대한 요청과 같은 다른 컴포넌트들로부터의 요청을 처리한다. 다른 예에서, 애플리케이션 관리 서브시스템(506)은 스트리밍될 수 있는 애플리케이션에 관한 컴포넌트에 의해 보내진 요청들을 처리한다. 이러한 이벤트들은 세가지 이벤트 유형으로 분류될 수 있다: 애플리케이션 퍼블리싱, 애플리케이션 목록 그리고 애플리케이션 개시. 다른 예에서, 애플리케이션 관리 서브시스템(506)은 애플리케이션 결정(application resolution), 애플리케이션 퍼블리케이션(application publication), 애플리케이션 퍼블리싱(application publishing)에 대한 지원을 더 포함한다. 다른 예에서, 애플리케이션 관리 서브시스템(506)은 애플리케이션 특성들과 정책들을 저장하기 위해 데이터 저장소를 사용한다.

<237> 서버 관리 서브시스템(508)은 서버 팜 구성내의 애플리케이션 스트리밍에 특정된 구성들을 처리한다. 일 실시예에서, 서버 관리 서브시스템(508)은 또한 팜(38)의 구성과 연관된 정보의 검색을 요하는 이벤트들을 처리한다. 다른 예에서, 서버 관리 서브시스템(508)은 애플리케이션 스트림을 가로질러 애플리케이션에 대한 액세스를 제공하는 원격 머신과 이들 원격 머신의 특성들과 관련된 다른 컴포넌트들에 의해 전송된 이벤트들을 처리한다. 일 실시예에서, 서버 관리 서브시스템(508)은 원격 머신 특성들 및 팜 특성들을 저장한다.

<238> 일 실시예에서, 원격 머신(30)은 하나 이상의 특정 애플리케이션 서브시스템에 대해 서비스를 제공하는 하나 이상의 공통 애플리케이션 서브시스템(524)을 더 포함한다. 이러한 원격 머신(30)은 또한 하나 이상의 특정 원격 머신 서브시스템에 서비스를 제공하는 공통 원격 머신 서브시스템을 포함할 수 있다. 다른 예에서, 공통 애플리케이션 서브시스템(524)이 제공되지 않을 수 있으며, 각각의 특정 애플리케이션 및 원격 머신 서브시스템은 모든 요구되는 기능을 수행한다.

<239> 원격 머신(30)이 공통 애플리케이션 서브시스템(524)을 포함하는 예에서, 공통 애플리케이션 서브시스템(524)은 퍼블리싱된 애플리케이션에 대한 공통 특성들을 관리한다. 일 실시예에서, 공통 애플리케이션 서브시스템(524)은 퍼블리싱된 애플리케이션과 연관된 또는 공통 특성들과 연관된 정보의 검색을 요하는 이벤트들을 처리한다. 다른 예에서, 공통 애플리케이션 서브시스템(524)은 공통 애플리케이션 및 그들의 특성들과 관련된 다른 컴포넌트들에 의해 보내진 모든 이벤트들을 처리한다.

<240> 공통 애플리케이션 서브시스템(524)은 팜(38)에 애플리케이션을 "퍼블리싱(publish)" 할 수 있으며, 이것은 각각의 애플리케이션이 목록되고 로컬 머신(10)에 의해 개시될 수 있도록 한다. 일반적으로, 애플리케이션은 애플리케이션의 가용성이 요구되는 각각의 원격 머신상에 설치된다. 일 실시예에서, 애플리케이션을 퍼블리싱하기 위해, 관리자는 애플리케이션을 호스팅하는 원격 머신(30), 각각의 원격 머신상의 실행가능 파일의 명칭, 애플리케이션을 실행하는데 필요한 로컬 머신의 성능(예를 들어, 오디오, 비디오, 암호, 등등), 그리고 애플리케이션을 사용할 수 있는 사용자의 목록과 같은 정보를 특정하는 관리 툴을 실행시킨다. 이렇게 특정된 정보는 애플리케이션-특정 정보 및 공통 정보로 카테고리가 나뉜다. 애플리케이션-특정 정보의 예로는: 애플리케이션 액세스를 위한 경로 명칭과 애플리케이션을 실행하기 위한 실행가능 파일의 명칭이다. 공통 정보(즉 공통 애플리케이션 데이터)는 애플리케이션의 사용자-위주 명칭(예를 들어, "Microsoft WORD 2000"), 애플리케이션의 유일한 ID, 그리고 애플리케이션의 사용자들을 포함한다.

<241> 애플리케이션-특정 정보 및 공통 정보는 애플리케이션을 호스팅하는 각각의 원격 머신(30)상에서 애플리케이션을 제어하는 특정 애플리케이션 서브시스템으로 보내질 수 있다. 특정 애플리케이션 서브시스템은 애플리케이션-특정 정보 및 공통 정보를 영구 저장장치(240)에 기록할 수 있다.

<242> 공통 애플리케이션 서브시스템(524) 또한 팜(38)내의 퍼블리싱된 애플리케이션을 관리하기 위한 기능을 제공한다. 공통 애플리케이션 서브시스템(524)을 통해, 관리자는 애플리케이션 그룹들을 구성하고 애플리케이션 그룹

들의 애플리케이션 트리 계층을 만들기 위한 액세스 수트 콘솔(520)과 같은 관리 툴을 사용하여 팜(38)의 애플리케이션들을 관리할 수 있다. 각각의 애플리케이션 그룹은 애플리케이션 트리 계층에서 폴더(folder)로서 표현될 수 있다. 애플리케이션 트리 계층내의 각각의 애플리케이션 폴더는 하나 이상의 다른 애플리케이션 폴더와 원격 머신들의 특정 인스턴스를 포함할 수 있다. 공통 애플리케이션 서브시스템(524)은 애플리케이션 폴더를 생성, 이동, 재명명, 삭제, 열거하는 기능들을 제공한다.

<243> 일 실시예에서, 공통 애플리케이션 서브시스템(524)은 애플리케이션 목록 및 애플리케이션 결정 요청을 처리하는 애플리케이션 관리 서브시스템(506)을 지원한다. 일 실시예에서, 공통 애플리케이션 서브시스템(524)은 데이터 파일의 유형과 데이터 파일의 유형을 처리하기 위한 애플리케이션 사이의 매핑에 대응하여 실행하기 위한 애플리케이션을 식별하는 기능을 제공한다. 다른 실시예에서, 제2 애플리케이션 서브시스템은 파일 유형 연합(association)을 위한 기능을 제공한다.

<244> 일 실시예에서, 원격 머신(30)은 정책 서브시스템을 더 포함할 수 있다. 정책 서브시스템은 애플리케이션 실행을 위한 로컬 머신(10)에 의한 요청에 따라 로컬 머신(10)에 애플리케이션이 스트리밍될 수 있는지 여부를 결정하기 위한 정책 규칙을 포함한다. 일 실시예에서, 정책 서브시스템은 액세스 수트 콘솔(520)에서 퍼블리싱된 스트리밍된 애플리케이션과 연관된 서버 액세스 옵션을 식별한다. 이 예에서, 정책 서브시스템은 정책 규칙 대신에 서버 액세스 옵션을 정책으로서 사용한다.

<245> 세션 관리 서브시스템(510)은 로컬 머신(10)과 연관된 애플리케이션 스트리밍 세션의 세션 상태를 유지하고 업데이트하며 애플리케이션 스트리밍 세션에 대한 라이선스 요구를 시행한다. 일 실시예에서 세션 관리 서브시스템(510)은 애플리케이션의 개시 또는 애플리케이션 스트리밍 세션의 종료와 같은 세션 및 로그 이벤트들을 모니터링한다. 다른 예에서, 세션 관리 서브시스템(510)은 로컬 머신(10)으로부터 원격 머신(30)에 전송되는 하트비트 메시지(heartbeat messages)와 같은 통신을 수신한다. 다른 예에서, 세션 관리 서브시스템(510)은 액세스 수트 콘솔(520)내의 툴들과 같은, 관리 툴로부터 세션에 관한 질의에 응답한다. 일 실시예에서, 관리 서비스(504)는 애플리케이션 실행을 위한 라이선스를 로컬 머신에 제공하고 유지하기 위한 세션 관리 서브시스템과 통신하는 라이선스 관리 서브시스템을 더 포함한다.

<246> 일 실시예에서, 관리 서비스(504)는 애플리케이션 목록 및 애플리케이션 결정을 위한 기능을 제공한다. 일 실시예에서, 관리 서비스(504)는 또한 애플리케이션 개시, 세션 모니터링 및 트래킹, 애플리케이션 퍼블리싱, 라이선스 시행을 위한 기능을 제공한다.

<247> 도 6을 참조하여, 애플리케이션 목록을 제공하는 관리 서비스를 포함하는 원격 머신(30)의 일 실시예를 나타내고 있다. 관리 서비스(504)는 XML 서비스(516)와 상호작용하는 웹 인터페이스를 사용하여 애플리케이션 목록을 제공할 수 있다. 일 실시예에서, XML 서비스(516)는 로컬 머신(10)의 사용자를 위해 애플리케이션을 열거한다. 다른 예에서, XML 서비스(516)는 ICA 브라우저 서브시스템과 근접 프로그램 서브시스템의 기능을 구현한다. XML 서비스(516)는 관리 통신 서비스(514)와 상호작용할 수 있다. 일 실시예에서, XML 서비스(516)는 관리 통신 서비스(514)를 사용하여 애플리케이션 목록 요청을 생성한다. 애플리케이션 목록 요청은 열거된 애플리케이션을 실행할 때 사용되는 실행 방법을 지시하는 클라이언트 유형을 포함할 수 있다. 애플리케이션 목록 요청은 공통 애플리케이션 서브시스템(524)으로 보내진다. 일 실시예에서, 공통 애플리케이션 서브시스템(524)은 애플리케이션 목록 요청의 클라이언트 유형과 연관된 애플리케이션의 목록을 재전송한다. 다른 예에서, 공통 애플리케이션 서브시스템(524)은 로컬 머신(10)의 사용자에게 사용 가능한 애플리케이션의 목록을 재전송하고, 이러한 목록은 로컬 머신(10)과 연관된 자격에 대한 애플리케이션 정책에 대응하여 선택된다. 이 예에서, 정책 엔진(406)은 도 4b에서 상술된 수집 에이전트(404)에 의해 수집된 자격에 대해 정책을 적용할 수 있다. 다른 예에서, 애플리케이션 목록은 재전송되고 로컬 머신(10)에 대한 애플리케이션 정책은 열거된 애플리케이션의 실행 요청이 있을 때까지 연기된다.

<248> 관리 서비스(504)는 애플리케이션을 호스팅하는 제2 원격 머신(30')을 식별하기 위해 애플리케이션 결정 서비스(application resolution service)를 제공할 수 있다. 일 실시예에서, 제2 원격 머신(30')은 파일 서버 또는 애플리케이션 서버이다. 일 실시예에서, 관리 서비스(504)는 애플리케이션을 호스팅하는 다수의 원격 머신(30)에 대한 식별자들을 포함하는 파일을 참조한다. 일 실시예에서, 관리 서비스(504)는 애플리케이션 실행에 대한 로컬 머신(10)의 요청에 응답하여 애플리케이션 결정 서비스를 제공한다. 다른 예에서, 관리 서비스(504)는 제1 원격 머신(30)과 다른 애플리케이션 실행 방법을 구현할 수 있는 제2 원격 머신(30')을 식별한다. 일 실시예에서, 관리 서비스(504)는 애플리케이션 프로그램을 로컬 머신(10)에 스트리밍할 수 있는 제1 원격 머신(30')과 애플리케이션 프로그램을 실행하고 애플리케이션 실행에 대응하여 생성된 애플리케이션-출력 데이터를 로

컬 머신(10)에 제공할 수 있는 제2 원격 머신(30')을 식별한다.

<249> 일 실시예에서, 웹 인터페이스는 애플리케이션 결정 요청을 XML 서비스(516)에 전송한다. 다른 예에서, XML 서비스(516)는 애플리케이션 결정 요청을 수신하고 이 요청을 MFRPC 서브시스템(522)에 전송한다.

<250> 일 실시예에서, MFRPC 서브시스템(522)은 수신된 애플리케이션 결정 요청과 포함된 클라이언트 유형을 식별한다. 다른 예에서, MFRPC 서브시스템은 클라이언트 유형에 정책을 적용하고 애플리케이션을 로컬 머신(10)에 "스트림(stream)"하도록 결정한다. 이 예에서, MFRPC 서브시스템(522)은 애플리케이션 결정 요청을 애플리케이션 관리 서브시스템(506)에 전달할 수 있다. 일 실시예에서, MFRPC 서브시스템(522)으로부터 애플리케이션 결정 요청을 수신하면, 애플리케이션 관리 서브시스템(506)은 로컬 머신(10)에 대해 세션 관리 서버(562)로서 기능하는 원격 머신(30')을 식별할 수 있다. 일 실시예에서, 로컬 머신은 세션 관리 서버(562)에 하트비트 메시지를 전송한다. 다른 예에서, 애플리케이션 관리 서브시스템(506)은 로컬 머신(10)에 스트리밍될 애플리케이션을 포함하는 다수의 애플리케이션 파일을 호스팅하는 원격 머신(30')을 식별할 수 있다.

<251> 일 실시예에서, 원격 머신(30')을 식별하기 위해 애플리케이션 관리 서브시스템(506)은 다수의 애플리케이션 파일을 호스팅하는 다수의 원격 머신을 열거하는 파일을 사용한다. 다른 예에서, 애플리케이션 관리 서브시스템(506)은 로컬 머신(10)의 IP 어드레스와 유사한 IP 어드레스를 갖는 원격 머신(30')을 식별한다. 다른 예에서, 애플리케이션 관리 서브시스템(506)은 로컬 머신(10)에 액세스가능한 IP 어드레스 범위내의 IP 어드레스를 갖는 원격 머신(30')을 식별한다.

<252> 다른 예에서, MFRPC 서브시스템(522)은 클라이언트 유형에 정책을 적용하고 원격 머신(30')상에 애플리케이션이 실행될 수 있는지 결정하고, 원격 머신(30')은 애플리케이션 실행에 의해 생성된 애플리케이션-출력 데이터를 로컬 머신(10)에 전송한다. 이 예에서, MFRPC 서브시스템(522)은 원격 머신(30')에 대한 호스트 어드레스의 식별자를 검색하기 위해 공통 애플리케이션 서브시스템(524)에 애플리케이션 결정 요청을 전달할 수 있다. 일 예에서, 식별된 원격 머신(30')은 ICA 또는 RDP 또는 X Windows와 같은 표현 레벨 프로토콜을 사용하여 로컬 머신에 애플리케이션-출력 데이터를 전송할 수 있다. 일 실시예에서, 원격 머신(30')은 애플리케이션 스트리밍 세션을 가로질러 제2 원격 머신(30')으로부터 애플리케이션을 수신한다.

<253> 일 실시예에서, 애플리케이션 목록 및 애플리케이션 결정이 완료되면, 열거된 애플리케이션에 대한 실행 방법에 대한 ID 및 열거된 애플리케이션을 호스팅하는 원격 머신(30')의 식별자를 포함하는 액세스 정보가 로컬 머신(10)에 전송된다. 관리 서비스(504)가 열거된 애플리케이션이 로컬 머신(10)상에서 실행될 것으로 결정한 예에서, 웹 인터페이스는 열거된 애플리케이션에 대한 명칭-결정 정보를 갖는 파일을 생성하고 로컬 머신(10)에 전송한다. 일 실시예에서, 파일은 ".rad" 확장자를 사용하여 식별될 수 있다. 로컬 머신(10)은 수신된 파일의 콘텐츠에 대응하여 열거된 애플리케이션을 실행할 수 있다.

<254> 표 2는 파일내에 구비된 정보의 일 실시예를 나타낸다.

표 2

필드(field)	디스크립션(description)	소스(source)
UNC path	파일서버상의 Container master manifest file에 대한 포인트	XML service
Initial program	container로부터 개시하는 프로그램	XML service
Command line	FTA 사용하여 문서를 개시하기 위한	XML service
Web server URL	RADE 클라이언트로부터 WI로의 메시지를 위한	WI config
Farm ID	하트비트 메시지를 위해 필요한 애플리케이션이 속하는 팜	WI config
Launch Ticket	애플리케이션 스트리밍 클라이언트는 프로그램 실행을 인증하는 라이선스를 얻기 위해 개시 티켓을 사용함	XML/IMA
ICA fallback launch info	폴백이 허용된다면, 폴백을 위한 임베디드 ICA 파일	XML service

- <256> 파일은 표 2에 나타난 바와 같이, 애플리케이션 실행하는데 로컬 머신에 의해 사용되는 개시 티켓(launch ticket)을 가지고 있을 수 있다. 일 실시예에서, 개시 티켓은 미리 정해진 시간 이후에 만료된다(expire). 일 실시예에서, 로컬 머신은 열거된 애플리케이션을 호스팅하는 원격 머신에 개시 티켓을 제공한다. 열거된 애플리케이션에 대한 액세스를 인증하기 위한 개시 티켓을 사용하는 것은 로컬 머신의 사용자가 애플리케이션에 대한 부적절한 액세스로 파일을 재사용하거나 파일의 비인증된 버전을 생성하는 것을 방지한다. 일 실시예에서, 개시 티켓은 큰, 랜덤방식으로 생성된 숫자를 포함한다.
- <257> 도 2를 참조하여 설명한 바와 같이, 애플리케이션 프로그램 실행 방법을 선택하기 위한 방법은 로컬 머신과 연관된 또는 로컬 머신(10)의 사용자와 연관된 자격이 수신되면 시작되고(단계 202) 수신된 자격에 대응하여 로컬 머신(10)에 대해 사용가능한 다수의 애플리케이션 프로그램의 목록이 제공된다(단계 204). 열거된 애플리케이션을 실행하기 위해 요청이 수신되고(단계 206) 정책에 대응하여 열거된 애플리케이션을 실행하기 위한 미리 결정된 수의 방법들중 하나가 선택되고, 미리 결정된 수의 방법들은 열거된 애플리케이션의 애플리케이션 스트리밍을 위한 방법을 포함한다(단계 208).
- <258> 도 7을 참조하여, 애플리케이션 프로그램을 포함하는 다수의 파일에 액세스하는 단계들을 나타내는 일 실시예가 나타나있다. 로컬 머신은 로컬 머신의 선개시 분석(pre-launch analysis)을 수행한다(단계 210). 일 실시예에서, 로컬 머신(10)은 애플리케이션 프로그램을 포함하는 다수의 애플리케이션 파일을 검색하고 실행하기 이전에 선개시 분석을 수행한다. 다른 예에서, 로컬 머신(10)은 애플리케이션 프로그램을 포함하는 다수의 애플리케이션 파일에 액세스하기 위한 인증을 위해 선개시 분석이 필요하다는 지시를 수신하는 것에 대응하여 선개시 분석을 수행한다.
- <259> 일 실시예에서, 로컬 머신(10)은 원격 머신(30)으로부터, 다수의 애플리케이션 파일과 연관된 액세스 정보를 수신한다. 이 예에서, 액세스 정보는 다수의 애플리케이션 파일을 호스팅하는 원격 머신(30)의 위치의 ID를 포함한다. 다른 예에서, 로컬 머신(10)은 애플리케이션 프로그램의 하나 이상의 버전을 포함하는 다수의 애플리케이션의 ID를 수신한다. 다른 예에서, 로컬 머신(10)은 하나 이상의 애플리케이션 프로그램을 포함하는 다수의 애플리케이션 파일의 ID를 수신한다. 다른 예에서, 로컬 머신(10)은 로컬 머신(10)에 사용가능한 애플리케이션 프로그램의 목록을 수신한다. 이 예에서, 목록은 로컬 머신(10)의 평가로부터 이루어진다. 다른 예에서, 로컬 머신(10)은 애플리케이션 프로그램을 포함하는 다수의 애플리케이션 파일의 검색된 ID에 대응하여 적어도 하나의 특성을 검색한다.
- <260> 일 실시예에서, 액세스 정보는 다수의 애플리케이션 파일로의 로컬 머신의 액세스를 인증할 수 있는 개시 티켓을 포함한다. 이 예에서, 개시 티켓은 로컬 머신(10)의 평가에 대응하여 로컬 머신(10)에 제공된다. 다른 예에서, 개시 티켓은 로컬 머신(10)에 의한 선개시 분석에 후속하여 로컬 머신(10)에 제공된다.
- <261> 다른 예에서, 로컬 머신(10)은 다수의 애플리케이션 파일의 실행을 위해 필요한 적어도 하나의 특성을 검색한다. 이 예에서, 액세스 정보는 적어도 하나의 특성을 포함한다. 다른 예에서, 액세스 정보는 로컬 머신(10)에 의한 검색을 위해 파일의 위치를 나타내며, 이 파일은 적어도 하나의 특성을 열거한다. 다른 예에서, 적어도 하나의 특성을 열거하는 파일은 다수의 애플리케이션 파일의 목록 및 다수의 애플리케이션 파일을 호스팅하는 원격 머신(30)의 ID를 더 포함한다.
- <262> 로컬 머신(10)은 로컬 머신상의 적어도 하나의 특성의 존재를 결정한다. 일 실시예에서, 로컬 머신(10)은 이러한 결정을 선개시 분석의 일부로서 정한다. 다른 예에서, 로컬 머신(10)은 로컬 머신(10)이 적어도 하나의 특성을 갖는지 여부를 결정한다.
- <263> 일 실시예에서, 로컬 머신(10)상의 적어도 하나의 특성의 존재를 결정하는 것은 장치 드라이버가 로컬 머신상에 설치되었는지 여부를 결정하는 것을 포함한다. 다른 예에서, 로컬 머신(10)상의 적어도 하나의 특성의 존재를 결정하는 것은 운영체제가 로컬 머신상에 설치되었는지 여부를 결정하는 것을 포함한다. 다른 예에서, 로컬 머신(10)상의 적어도 하나의 특성의 존재를 결정하는 것은 특정 운영체제가 로컬 머신상에 설치되었는지 여부를 결정하는 것을 포함한다. 다른 예에서, 로컬 머신(10)상의 적어도 하나의 특성의 존재를 결정하는 것은 운영체제의 특정 수정 레벨이 로컬 머신상에 설치되었는지 여부를 결정하는 것을 포함한다.
- <264> 일 실시예에서, 로컬 머신(10)상의 적어도 하나의 특성의 존재를 결정하는 것은 로컬 머신이 열거된 애플리케이션을 실행하기 위한 인증을 얻었는지 여부를 결정하는 것을 포함한다. 이 예에서, 이러한 결정은 로컬 머신(10)이 열거된 애플리케이션을 실행하기 위한 라이선스를 수신했는지 여부에 따라서 로컬 머신(10)에 의해 이루어진다. 다른 예에서, 이러한 결정은 로컬 머신(10)이 열거된 애플리케이션을 포함하는 다수의 애플리케이션

파일을 애플리케이션 스트리밍 세션을 가로질러 수신할 수 있는 라이선스를 수신했는지 여부에 따라서 로컬 머신(10)에 의해 이루어진다. 다른 예에서, 로컬 머신(10)상의 적어도 하나의 특성의 존재를 결정하는 것은 로컬 머신(10)이 열거된 애플리케이션을 검색하고 실행할 수 있는 충분한 대역폭을 갖는지 여부를 결정하는 것을 포함한다.

<265> 일 실시예에서, 로컬 머신(10)상의 적어도 하나의 특성의 존재를 결정하는 것은 로컬 머신(10)상에서 스크립트 실행을 포함한다. 다른 예에서, 로컬 머신(10)상의 적어도 하나의 특성의 존재를 결정하는 것은 로컬 머신(10)상의 소프트웨어 설치를 포함한다. 다른 예에서, 로컬 머신(10)상의 적어도 하나의 특성의 존재를 결정하는 것은 로컬 머신(10)상의 레지스트리의 수정(modification)을 포함한다. 다른 예에서, 로컬 머신(10)상의 적어도 하나의 특성의 존재를 결정하는 것은 로컬 머신(10)과 연관된 자격을 수집하도록 로컬 머신(10)상에서의 실행을 위해 수집 에이전트(404)를 로컬 머신(10)에 전송하는 것을 포함한다.

<266> 로컬 머신(10)은 원격 머신(30)으로, 다수의 애플리케이션 파일의 실행을 위한 인증을 요청하고, 이 요청은 개시 티켓을 포함한다(단계 212). 일 실시예에서, 로컬 머신(10)은 이 요청을 로컬 머신(10)상에 적어도 하나의 특성이 존재한다는 결정에 대응하여 만든다. 이 예에서, 로컬 머신(10)은 로컬 머신(10)상에 다수의 특성이 존재하는지를 결정하고, 다수의 특성은 열거된 애플리케이션과 연관되며 열거된 애플리케이션을 실행하기 위한 요청에 대응하여 수신된다. 다른 예에서, 로컬 머신(10)이 열거된 애플리케이션 파일의 실행을 위한 인증에 대한 표시를 수신하는지 여부는 로컬 머신(10)상의 적어도 하나의 특성의 존재여부에 따른다. 일예에서, 로컬 머신(10)상의 적어도 하나의 특성의 존재에 대한 결정에 따라, 로컬 머신(10)은 애플리케이션 프로그램의 목록을 수신하고, 열거된 애플리케이션의 실행을 요청하고, 적어도 하나의 특성을 포함하는 액세스 정보와 열거된 애플리케이션의 실행을 인증하는 개시 티켓을 수신한다.

<267> 일 실시예에서, 로컬 머신(10)은 다수의 애플리케이션 파일의 실행을 인증하는 라이선스를 원격 머신(30)으로부터 수신한다. 일 실시예에서, 라이선스는 특정 시간동안의 실행을 인증한다. 이 예에서, 라이선스는 다수의 애플리케이션 파일의 실행에 대한 인증을 유지하기 위해 하트비트 메시지의 전송을 요한다.

<268> 다른 실시예에서, 로컬 머신(10)은 라이선스와 다수의 애플리케이션 파일의 실행을 모니터링하는 원격 머신(30)과 연관된 식별자를 원격 머신(30)으로부터 수신한다. 일 실시예에서, 원격 머신은 도 5에 전송된 세션 관리 서버(562)이다. 이 예에서, 세션 관리 서버(562)는 로컬 머신(10)과 연관된 세션을 모니터링하는 세션 관리 서브시스템(510)을 포함한다. 다른 예에서, 격리된 원격 머신(30'')은 세션 관리 서버(562)이다.

<269> 로컬 머신(10)은 다수의 애플리케이션 파일을 수신하고 실행한다(단계 214). 일 실시예에서, 로컬 머신(10)은 애플리케이션 스트리밍 세션을 가로질러 다수의 애플리케이션 파일을 수신한다. 다른 예에서, 로컬 머신(10)은 다수의 애플리케이션 파일을 로컬 머신(10)상의 격리된 환경에 저장한다. 다른 예에서, 로컬 머신(10)은 두번째 다수의 애플리케이션 파일을 수신하기 이전에 다수의 애플리케이션 파일 중 하나를 실행한다. 일 실시예에서, 원격 머신은 다수의 애플리케이션 파일을 다수의 로컬 머신에 전송하고, 각각의 로컬 머신은 원격 머신과 격리된 애플리케이션 스트리밍 세션을 구축한다.

<270> 일 실시예에서, 로컬 머신(10)은 다수의 애플리케이션 파일을 캐시에 저장하고 애플리케이션 파일의 실행을 지연한다. 이 예에서, 로컬 머신(10)은 미리 결정된 시간동안 애플리케이션 파일을 실행하기 위한 인증을 수신한다. 다른 예에서, 로컬 머신(10)은 로컬 머신(10)이 네트워크에 대한 액세스가 부족할때(lack) 미리 결정된 시간동안 애플리케이션 파일을 실행하기 위한 인증을 수신한다. 다른 예에서, 로컬 머신은 다수의 애플리케이션 파일을 캐시에 저장한다. 이 예에서, 애플리케이션 스트리밍 클라이언트(552)는 캐시로부터 다수의 애플리케이션 파일을 검색하기 위해 내부 애플리케이션 스트리밍 세션을 구축한다. 다른 예에서, 로컬 머신(10)은 로컬 머신(10)이 네트워크에 대한 액세스가 부족할때(lack) 미리 결정된 시간동안 애플리케이션 파일을 실행하기 위한 인증을 수신한다.

<271> 로컬 머신(10)은 적어도 하나의 하트비트 메시지를 원격 머신에 전송한다(단계 216). 일 실시예에서, 로컬 머신(10)은 열거된 애플리케이션을 포함하는 다수의 애플리케이션 파일을 실행하기 위한 인증을 보유하기 위해 적어도 하나의 하트비트 메시지를 전송한다. 다른 예에서, 로컬 머신(10)은 다수의 애플리케이션 파일내의 애플리케이션 파일을 검색하기 위한 인증을 보유하기 위해 적어도 하나의 하트비트 메시지를 전송한다. 다른 예에서, 로컬 머신(10)은 미리 결정된 시간동안 다수의 애플리케이션 파일의 실행을 인증하는 라이선스를 수신한다.

<272> 일 실시예에서, 로컬 머신(10)은 제2 원격 머신(30'')에 하트비트 메시지를 전송한다. 이 예에서, 제2 원격 머신(30'')은 다수의 애플리케이션 파일의 검색 및 실행을 모니터링하는 세션 관리 서버(562)를 포함할 수 있다.

다른 예에서, 제2 원격 머신(30'')은 전송된 하트비트 메시지에 대응하여, 다수의 애플리케이션 파일의 실행을 인증하는 라이선스를 갱신할 수 있다. 다른 예에서, 제2 원격 머신(30'')은 전송된 하트비트 메시지에 대응하여, 명령(command)을 로컬 머신(10)에 전송할 수 있다.

<273> 다시 도 5를 참조하여, 로컬 머신(10)은 애플리케이션 스트리밍 클라이언트(552), 스트리밍 서비스(554) 그리고 격리된 환경(556)을 포함할 수 있다. 애플리케이션 스트리밍 클라이언트(552)는 실행가능 프로그램일 수 있다. 일 실시예에서, 애플리케이션 스트리밍 클라이언트(552)는 다른 실행가능 프로그램을 개시할 수 있다. 다른 예에서, 애플리케이션 스트리밍 클라이언트(552)는 스트리밍 서비스(554)를 개시할 수 있다. 이 예에서, 애플리케이션 스트리밍 클라이언트(552)는 애플리케이션 프로그램을 실행하는 것과 연관된 파라미터를 스트리밍 서비스(554)에 제공할 수 있다. 다른 예에서, 애플리케이션 스트리밍 클라이언트(552)는 원격 처리 호출(remote procedure call)을 사용하여 스트리밍 서비스(554)를 개시할 수 있다.

<274> 일 실시예에서, 로컬 머신(10)은 애플리케이션 프로그램의 실행을 요청하고 원격 머신(30)으로부터 실행에 대한 액세스 정보를 수신한다. 다른 예에서, 애플리케이션 스트리밍 클라이언트(552)는 액세스 정보를 수신한다. 다른 예에서, 애플리케이션 스트리밍 클라이언트(552)는 액세스 정보를 스트리밍 서비스(554)에 제공한다. 다른 예에서, 액세스 정보는 애플리케이션 프로그램을 포함하는 다수의 애플리케이션 파일의 위치의 ID를 포함한다.

<275> 일 실시예에서, 스트리밍 서비스(554)는 다수의 애플리케이션 파일과 연관된 파일을 검색한다. 일 실시예에서, 검색된 파일은 다수의 애플리케이션 파일의 위치의 ID를 포함한다. 이 예에서, 스트리밍 서비스(554)는 다수의 애플리케이션 파일을 검색한다. 다른 예에서, 스트리밍 서비스(554)는 로컬 머신(10)상의 검색된 다수의 애플리케이션 파일을 실행한다. 다른 실시예에서, 스트리밍 서비스(554)는 다수의 애플리케이션 파일을 검색 및 실행하기 위한 인증을 보유하기 위해 원격 머신에 하트비트 메시지를 전송한다.

<276> 일 실시예에서, 검색된 파일은 하나 이상의 다수의 애플리케이션 파일의 위치의 ID를 포함하며, 다수의 애플리케이션 파일 각각은 다른 애플리케이션 프로그램을 포함한다. 이 예에서, 스트리밍 서비스(554)는 로컬 머신(10)과 호환되는 애플리케이션 프로그램을 포함하는 다수의 애플리케이션 파일을 검색한다. 다른 예에서, 스트리밍 서비스(554)는 로컬 머신(10)의 평가에 대응하여, 특정한 다수의 애플리케이션 파일을 검색한다.

<277> 일 실시예에서, 다수의 애플리케이션 파일은 CAB, ZIP, SIT, TAR, JAR와 같은 보관 파일(archive file)로 파일 서버상에 압축되고 저장된다. 일 실시예에서, 다수의 애플리케이션 파일은 애플리케이션 파일을 포함하는 보관 파일내에 저장된다. 다른 예에서, 보관 파일내에 저장된 여러개의 다수의 애플리케이션 파일 각각은 다른 버전의 애플리케이션 프로그램을 포함한다. 다른 예에서, 보관 파일내에 저장된 여러개의 다수의 애플리케이션 파일 각각은 다른 애플리케이션 프로그램을 포함한다. 일 실시예에서, 보관 파일은 다수의 애플리케이션 파일내의 각각의 파일과 연관된 메타데이터를 포함한다. 이 예에서, 스트리밍 서비스(554)는 포함된 메타데이터에 대응하여 디렉토리 구조를 생성한다. 도 12와 함께 자세히 설명되었지만, 이러한 메타데이터는 디렉토리 목록에 대한 애플리케이션 프로그램의 요청을 만족시키는데 사용될 수 있다.

<278> 일 실시예에서, 스트리밍 서비스(554)는 다수의 애플리케이션 파일을 얻기위해 보관 파일을 압축해제한다. 다른 예에서, 스트리밍 서비스(554)는 다수의 애플리케이션 파일로부터 파일을 검색하기 이전에 로컬 머신(10)상의 캐시에 다수의 애플리케이션 파일내에서 파일의 로컬 복사본이 존재하는지 여부를 결정한다. 다른 예에서, 파일 시스템 필터 드라이버(564)는 로컬 복사본이 캐시내에 존재하는지 여부를 결정한다. 일 실시예에서, 스트리밍 서비스(554)는 다수의 애플리케이션 파일내에서 파일을 검색하기 이전에 레지스트리 엔트리를 수정한다.

<279> 일 실시예에서, 스트리밍 서비스(554)는 다수의 애플리케이션 파일을 로컬 머신(10)상의 캐시에 저장한다. 이 예에서, 스트리밍 서비스(554)는 다수의 애플리케이션 파일을 캐시하기 위한 요청을 수신하는 것에 따라 다수의 애플리케이션 파일을 캐시하기 위한 기능을 제공할 수 있다. 다른 예에서, 스트리밍 서비스(554)는 로컬 머신(10)상의 캐시를 보호하는 기능을 제공할 수 있다. 다른 예에서, 스트리밍 서비스(554)는 캐시의 크기 및 위치를 조정하는 알고리즘을 사용할 수 있다.

<280> 일 실시예에서, 스트리밍 서비스(554)는 로컬 머신(10)상의 격리 환경(556)을 생성한다. 이 예에서, 스트리밍 서비스(554)는 격리 환경(556)을 생성하기 위해 격리 환경 애플리케이션 프로그래밍 인터페이스를 사용한다. 다른 예에서, 스트리밍 서비스(554)는 다수의 애플리케이션 파일을 격리 환경(556)내에 저장한다. 다른 예에서, 스트리밍 서비스(554)는 격리 환경에서 다수의 애플리케이션 파일내의 파일을 실행한다. 다른 예에서, 스트리밍 서비스(554)는 격리 환경에서 애플리케이션 프로그램을 실행한다.

- <281> 로컬 머신(10)상에서 애플리케이션을 실행하기 위한 인증이 수신된 예에서, 애플리케이션의 실행은 격리 환경(556)내에서 발생할 수 있다. 일 실시예에서, 애플리케이션을 포함하는 다수의 애플리케이션 파일은 애플리케이션의 실행 이전에 로컬 머신(10)상에 저장된다. 다른 예에서, 다수의 애플리케이션 파일의 서브세트가 애플리케이션의 실행 이전에 로컬 머신(10)상에 저장된다. 다른 예에서, 다수의 애플리케이션 파일은 격리 환경(556)에 존재하지 않는다. 다른 예에서, 다수의 애플리케이션 파일의 서브세트는 로컬 머신(10)상에 존재하지 않는다. 다수의 애플리케이션 파일의 서브세트 또는 다수의 애플리케이션 파일의 각각의 애플리케이션 파일이 로컬 머신(10)상에 존재하는지 또는 격리 환경(556)내에 존재하는지 여부에 관계없이, 일 실시예에서, 다수의 애플리케이션 파일의 애플리케이션 파일은 격리 환경(556)내에서 실행될 수 있다.
- <282> 격리 환경(556)은 애플리케이션 소스 코드에 대한 임의의 변경을 요구함이 없이 애플리케이션 호환 문제를 줄이기 위해 파일 시스템 가상화(File System Virtualization), 레지스트리 시스템 가상화(Registry System Virtualization), 명명 오브젝트 가상화(Named Object Virtualization)를 제공할 수 있는 코어 시스템으로 구성될 수 있다. 격리 환경(556)은 후킹(hooking)을 사용하여 애플리케이션 리소스 요청을 레지스트리 및 명명된 오브젝트 가상화에 대한 사용자 모드내로, 그리고 파일 시스템 가상화를 위한 파일 시스템 필터 드라이버를 사용하는 커널(kernel)내로 방향을 바꿀 수 있다(redirect). 이하는 격리 환경(556)의 일 실시예에 대한 설명이다.
- <283> 도 8a를 참조하여, 애플리케이션 호환성 및 애플리케이션 친화성 문제가 감소된 운영체제(100)의 제어하에 실행되는 컴퓨터의 일 실시예를 나타낸다. 운영체제(100)은 시스템 층(108)을 통해 애플리케이션 프로그램(112, 114)에 대해 사용가능한 다양한 고유 리소스를 만든다. 시스템 층(108)에 의해 구체화된 리소스는 "시스템 범위(system scope)"로 정해될 것이다. 애플리케이션 프로그램(112, 114)에 의한 고유 리소스(102, 104, 106, 107)로의 액세스 충돌을 피하기 위하여, 격리 환경(200)이 제공된다. 도 8a에 도시된 바와 같이, 격리 환경(200)은 애플리케이션 격리층(220)과 사용자 격리층(240)을 포함한다. 개념적으로, 격리 환경(200)은 애플리케이션 격리층(240)을 통해, 애플리케이션 프로그램(112, 114)에 파일 시스템(102), 레지스트리(104), 오브젝트(106), 윈도우 명칭(107)과 같은 고유 리소스의 유일한 뷰(view)를 제공한다. 각각의 격리층은 애플리케이션에 제공된 고유 리소스의 뷰를 수정한다. 층에 의해 제공된 고유 리소스의 수정된 뷰는 층의 "격리 범위(isolation scope)"로서 언급된다. 도 8a에 도시된 바와 같이, 애플리케이션 격리층은 2개의 애플리케이션 격리 범위(222, 224)를 포함한다. 범위(222)는 애플리케이션(112)에 제공된 고유 리소스의 뷰를 나타내고 범위(224)는 애플리케이션(114)에 제공된 고유 리소스의 뷰를 나타낸다. 따라서, 도 8a에 도시된 예에서, APP1(112)이 파일 시스템(102')의 특정 뷰로 제공되는 반면, APP2(114)는 파일 시스템(102")의 다른 뷰로 제공된다. 일 실시예에서, 애플리케이션 격리층(220)은 운영체제(100)의 상위에서 실행되는 각각의 개별적인 애플리케이션 프로그램에 고유 리소스(102, 104, 106, 107)의 특정 뷰를 제공한다. 다른 예에서, 애플리케이션 프로그램(112, 114)은 집합으로 그룹화될 수 있으며, 이 예에서, 애플리케이션 격리층(220)은 애플리케이션 프로그램의 각각의 집합에 대한 고유 리소스의 특정 뷰를 제공한다. 애플리케이션 프로그램 충돌은 애플리케이션의 호환성 및 친화성을 강화하기 위해 별개 그룹으로 나뉘질 수 있다. 다른 예에서, 집합에 속하는 애플리케이션은 관리자에 의해 구성될 수 있다. 일 실시예에서, "통과(passthrough)" 격리 범위는 시스템 범위에 정확히 대응하는 것으로 규정될 수 있다. 다르게 말하면, 통과 격리 범위내에서 실행되는 애플리케이션은 시스템 범위상에서 바로 동작한다.
- <284> 일 실시예에서, 애플리케이션 격리 범위는 층형 서브-범위(layered sub-scope)로 더 나뉘질 수 있다. 메인 서브-범위는 기본 애플리케이션 격리 범위를 갖고, 추가적인 서브-범위는 애플리케이션의 다중 실행 인스턴스에 보여질 수 있는 범위에 대한 다양한 수정을 가질 수 있다. 예를 들어, 서브-범위는 애플리케이션 또는 설치의 패치 레벨에서 또는 추가적인 특징의 제거에서의 변경을 구체화하는 범위에 대한 수정을 가질 수 있다. 일 실시예에서, 애플리케이션 실행의 인스턴스에 보여질 수 있도록 만들어진 가적인 서브-범위의 집합이 구성될 수 있다. 일 실시예에서, 사용자에게 상관없이, 보여지는 서브-범위의 집합은 애플리케이션 실행의 모든 인스턴스에 대해 동일하다. 다르게는, 보여지는 서브-범위의 집합은 애플리케이션을 실행하는 다른 사용자에게 대해 변경될 수 있다. 다양한 서브-범위의 집합들이 규정될 수 있고 사용자는 어떤 집합이 사용될지에 대한 선택권을 가질 수 있다. 일 실시예에서, 서브-범위는 더 이상 필요하지 않게 되면 제거될 수 있다. 일 실시예에서, 서브-범위의 집합내에 포함된 수정은 단일 서브-범위를 형성하기 위해 서로 합쳐질 수 있다.
- <285> 도 8b를 참조하면, 애플리케이션 호환성 및 애플리케이션 친화성 문제가 줄어든 다중-사용자 컴퓨터가 도시되어 있다. 다중-사용자 컴퓨터는 전술한 격리 환경(200)뿐만 아니라, 시스템 층(108)내의 고유 리소스(102, 104, 106, 107)를 포함한다. 애플리케이션 격리층(220)은 애플리케이션 또는 애플리케이션의 그룹에 고유 리소스의

수정된 뷰를 제공한다. 개념적으로, 사용자 격리층(240)은 애플리케이션 프로그램(112, 114)에 사용자의 사용자 아이디엔티티에 근거하여 더욱 변경되는 고유 리소스의 뷰를 제공한다. 도 8b에 도시된 바와 같이, 사용자 격리층(240)은 다수의 사용자 격리 범위(242', 242'', 242''', 242'''', 242''''', 242'''''(일반적으로 242))를 포함하는 것으로 고려될 수 있다. 사용자 격리 범위(242)는 고유 리소스의 애플리케이션-특정 뷰의 사용자-특정 뷰를 제공한다. 예를 들어, 사용자 "a" 대신에 사용자 세션(110)에서 실행되는 APP1(112)이 파일 시스템 뷰(102'(a))로 제공되며, 파일 시스템 뷰는 사용자 격리 범위(242')와 애플리케이션 격리 범위(222)에 의해 변경 또는 수정된다.

<286> 다른 방식으로, 사용자 격리층(240)은 시스템 층에 의해 제공된 고유 리소스의 시스템 뷰의 "상위층"인 애플리케이션 격리 범위(222)에 의해 제공된 애플리케이션-특정 뷰 수정의 "상위"의 사용자 격리 범위(242')에 의해 제공된 사용자-특정 뷰 수정을 "계층화(layering)"함으로써 각각의 개별적인 사용자에게 대한 고유 리소스의 뷰를 변경한다. 예를 들어, APP1의 제1 인스턴스가 레지스트리 데이터베이스(104)내의 엔트리에 액세스할 때, 제1 사용자 세션 및 애플리케이션(104'(a))에 특정된 레지스트리 데이터베이스의 뷰가 참조된다. 요청된 레지스트리 키가 레지스트리(104'(a))의 사용자-특정 뷰내에서 발견되면, 레지스트리 키는 APP1(112)로 되돌려진다. 그렇지 않다면, 애플리케이션(104')에 특정된 레지스트리 데이터베이스의 뷰가 참조된다. 요청된 레지스트리 키가 레지스트리(104')의 애플리케이션-특정 뷰에서 발견되면, 레지스트리 키는 APP1(112)로 되돌려진다. 그렇지 않다면, 시스템 층(108)내의 레지스트리 데이터베이스(104)에 저장된 레지스트리 키가 APP1(112)로 되돌려진다.

<287> 일 실시예에서, 사용자 격리층(240)은 각각의 개별적 사용자에게 격리 범위를 제공한다. 다른 예에서, 사용자 격리층(240)은 조직내의 규칙에 의해 규정되거나 또는 관리자에 의해 미리 결정될 수 있는, 사용자 그룹에 격리 범위를 제공한다. 다른 예에서, 사용자 격리층(240)이 제공되지 않는다. 이 예에서, 애플리케이션 프로그램에 의해 보여지는 고유 리소스의 뷰는 애플리케이션 격리층(220)에 의해 제공된다. 다양한 사용자들에 의한 애플리케이션 프로그램의 동시 실행을 지원하는 다중-사용자 컴퓨터에 대해 설명되었지만, 격리 환경(200)은 다른 사용자들에 의한 동일 컴퓨터 시스템상의 애플리케이션 프로그램의 순차적인 실행으로부터 야기되는 애플리케이션 호환성 및 진화성 문제를 처리하기 위해 단일-사용자 컴퓨터상에서도 사용될 수 있으며, 이러한 문제는 동일한 사용자에게 의한 비호환 프로그램들의 설치 및 실행에 의해 야기된다.

<288> 일 실시예에서, 사용자 격리 범위는 서브-범위로 더 나뉜다. 사용자 격리 범위내에서 실행되는 애플리케이션에 제공되는 뷰에 대한 수정은 사용자 격리 범위내의 각각의 서브-범위내에서 갖는 수정들의 집합이다. 서브-범위들은 계층화되고, 높은 서브-범위에서 리소스에 대한 합쳐진 뷰 수정은 낮은 층에서의 동일한 리소스에 대한 수정을 오버라이드(override)한다.

<289> 일 실시예에서, 하나 이상의 서브-범위는 사용자에게 특정된 뷰에 대한 수정을 가질 수 있다. 일예에서, 하나 이상의 서브-범위는 사용자 집합(시스템 관리자에 의해 규정되거나 또는 운영체제내의 사용자 그룹에 따라 규정될 수 있음)에 특정된 뷰에 대한 수정을 가질 수 있으며, 일예에서, 이러한 서브-범위들 중 하나는 특정 로그인 세션에 특정된 뷰에 대한 수정을 가질 수 있으며, 따라서 세션이 끝나면 이것은 제거된다. 일예에서, 사용자 격리 범위와 연관된 애플리케이션 인스턴스에 의한 고유 리소스에 대한 변경은 항상 서브-범위들 중 하나에 영향을 미치고, 다른 예에서 이러한 변경은 특정 리소스가 변경됨에 따라서 다른 서브-범위에도 영향을 미칠 수 있다.

<290> 전문한 개념적 아키텍처는 사용자를 위해 실행되는 애플리케이션이, 애플리케이션과 사용자의 조합에 특정되는 고유 리소스의 합쳐진(aggregate), 또는 통일된(unified), 가상화된(virtualized) 뷰로 제공되는 것을 허용한다. 이러한 합쳐진 뷰는 "가상 범위(virtual scope)"로 언급될 수 있다. 사용자를 위해 실행되는 애플리케이션 인스턴스는 고유 리소스의 모든 가상화된 인스턴스를 반영하는 고유 리소스의 단일 뷰로 제공된다. 개념적으로 이러한 합쳐진 뷰는 먼저 시스템 범위내의 운영체제에 의해 제공된 고유 리소스들의 집합으로 구성되고, 실행되는 애플리케이션에 적용할 수 있는 애플리케이션 격리 범위내에서 구체화되는 수정에 의해 오버레이(overlay)되고, 사용자를 위해 실행되는 애플리케이션에 적용할 수 있는 사용자 격리 범위내에서 구체화되는 수정으로 다시 오버레이된다. 운영체제가 특정 사용자 또는 애플리케이션에 대해 액세스를 거부하는 경우를 제외하고는, 시스템 범위내의 고유 리소스는 시스템상의 모든 사용자 및 애플리케이션에 공통된다는 특징을 갖는다. 애플리케이션 격리 범위내에서 구체화되는 리소스 뷰에 대한 수정은 애플리케이션 격리 범위와 연관된 모든 애플리케이션의 인스턴스에 공통된다는 특징을 갖는다. 사용자 격리 범위내에서 구체화되는 리소스 뷰에 대한 수정은 사용자 격리 범위와 연관된 사용자들을 위해 실행되며 적용될 수 있는 애플리케이션 격리 범위와 연관된 모든 애플리케이션들에 공통된다는 특징을 갖는다.

- <291> 이러한 개념은 서브-범위들로 확장될 수 있다; 사용자 서브-범위내에서 구체화되는 리소스 뷰에 대한 수정은 사용자 또는 사용자 그룹을 위해 실행되는 적용할 수 있는 격리 서브-범위와 연관된 모든 애플리케이션들에 공통된다. 이 설명을 통해 일반적인 참조로 "범위"가 사용될 때마다 이것은 또한 서브-범위를 언급한다는 것을 이해해야한다.
- <292> 애플리케이션이 파일 시스템 또는 레지스트리 데이터베이스의 일부와 같은 고유 리소스의 목록을 요청할 때, 먼저 고유 리소스의 "시스템-범위(system-scoped)" 인스턴스 즉 시스템 층내에서 발견된 인스턴스를 열거함으로써 가상화된 목록이 구성된다. 다음으로, 요청된 리소스의 "애플리케이션-범위(application-scoped)" 인스턴스, 즉 적절한 애플리케이션 격리 범위내에서 발견된 인스턴스가 열거된다. 애플리케이션 격리 범위내에서 마주치는(encountered) 임의의 열거된 리소스들은 뷰로 추가된다. 열거된 리소스가 이미 뷰내에 존재하면(시스템 범위내에서 제공되었기 때문에), 이것은 애플리케이션 격리 범위내에 마주친 리소스의 인스턴스로 대체된다. 유사하게, 요청된 리소스의 "사용자-범위" 인스턴스, 즉 적절한 사용자 격리 범위내에서 발견된 인스턴스가 열거된다. 다시, 사용자 격리 범위내에서 마주친 임의의 열거된 리소스들이 뷰로 추가된다. 고유 리소스가 이미 뷰내에 존재하면(시스템 범위 또는 적절한 애플리케이션 격리 범위내에 제공되었기 때문에), 이것은 사용자 격리 범위내에서 마주친 리소스의 인스턴스로 대체된다. 이러한 방식에서, 고유 리소스의 목록은 열거된 고유 리소스의 가상화를 적절히 반영할 것이다. 개념적으로 동일한 접근법이 다수의 서브-범위들을 포함하는 격리 범위를 열거하는데 적용된다. 낮은 서브-범위로부터의 매칭 인스턴스를 대체하는 높은 서브-범위로부터 리소스로 개별적 서브-범위들이 열거된다.
- <293> 다른 예에서, 목록은 사용자 격리 범위 층으로부터 시스템 층으로 수행될 수 있다. 이 예에서, 사용자 격리 범위가 열거된다. 그 다음 애플리케이션 격리 범위가 목록되고 애플리케이션 격리 범위내에 나타나며 사용자 격리 범위내에 열거되지 않는 임의의 리소스 인스턴스들이 구성중에 있는 집합 뷰(aggregate view)에 추가된다. 유사한 프로세스가 시스템 범위내에만 나타나는 리소스들에 대해 반복될 수 있다.
- <294> 다른 예에서, 모든 격리 범위는 동시에 열거되거나 개별적인 목록들이 합쳐질 수 있다.
- <295> 애플리케이션이 리소스를 수정할 의도없이 고유 리소스의 현존 인스턴스를 오픈하려고 시도하면, 애플리케이션으로 리턴되는 특정 인스턴스가 가상 범위내에서 발견되는 것이 되거나, 또는 요청된 리소스의 부모(parent)의 가상화된 목록내에 나타날 수 있는 인스턴스가 된다. 격리 환경의 뷰의 관점으로부터, 애플리케이션이 "가상 리소스"를 오픈하기 위해 요청하는 것으로 말할 수 있으며, 이 요청을 만족시키는데 사용되는 고유 리소스의 특정 인스턴스가 요청된 리소스에 대응하는 "리터럴 리소스(literal resource)"가 된다.
- <296> 사용자를 위해 실행되는 애플리케이션이 리소스를 오픈하려고 시도하고 리소스를 수정하려는 의도임을 나타낸다면, 애플리케이션 격리 범위 및 시스템 범위내의 리소스들은 다른 사용자들을 위해 실행되는 애플리케이션들에 공통되므로 애플리케이션 인스턴스는 수정을 위해 리소스의 사적 카피(private copy)로 주어진다. 사용자-범위 인스턴스(user-scoped instance)가 이미 존재하지 않는다면, 리소스의 사용자-범위 카피(user-scoped copy)가 만들어진다. 가상 범위에 의해 제공된 집합 뷰의 정의(definition)는 애플리케이션-범위 또는 시스템-범위 리소스를 사용자 격리 범위로 카피하는 동작이 당해(in question) 사용자 및 애플리케이션, 임의의 다른 사용자, 임의의 다른 애플리케이션 인스턴스를 위한 가상 범위에 의해 제공된 집합 뷰를 변경시키지 않는다는 것을 의미한다. 사용자를 위해 실행되는 애플리케이션 인스턴스에 의해서 복사된 리소스에 대한 후속적인 수정은 동일한 사용자 격리 범위를 공유하지 않는 임의의 다른 애플리케이션 인스턴스의 집합 뷰에 영향을 미치지 않는다. 다른 말로, 이러한 수정들은 다른 사용자들, 또는 동일한 애플리케이션 격리 범위와 연관되지 않은 애플리케이션 인스턴스를 위한 고유 리소스의 집합 뷰를 변경시키지 않는다.
- <297> 애플리케이션은 특정 격리 범위로 설치될 수 있다. 격리 범위로 설치된 애플리케이션은 항상 그 범위와 연관된다. 선택적으로, 애플리케이션은 특정 격리 범위 또는 다수의 격리 범위로 개시될 수 있다. 실제로, 애플리케이션은 개시되고 하나 이상의 격리 범위와 연관된다. 연관된 격리 범위, 또는 범위들은 프로세스에 고유 리소스의 특정 뷰를 제공한다. 애플리케이션은 또한 시스템 범위로 개시될 수 있다, 즉 애플리케이션은 아무런 격리 범위와도 연관되지 않을 수 있다. 이것은 격리 환경내에서, 제3자(third party)의 애플리케이션들뿐만 아니라, Internet Explorer와 같은 운영체제 애플리케이션의 선택적 실행을 허용한다.
- <298> 애플리케이션이 어디에 설치되었는지에 상관없이 격리 범위내에서 애플리케이션을 개시할 수 있는 능력은 격리 범위내에서 애플리케이션의 분리된 설치를 요하지 않으면서 애플리케이션 호환성 및 친화성 문제를 완화시킨다. 다른 격리 범위내에서 설치된 애플리케이션을 선택적으로 개시할 수 있는 능력은 헬퍼 애플리케이션(helper

application)(Word, Notepad 등)이 필요한 애플리케이션들을 가질 수 있는 능력을 제공한다.

- <299> 또한, 다수의 격리된 환경들내에서 애플리케이션을 개시할 수 있는 능력은 격리된 애플리케이션들과 공통 애플리케이션들 사이의 더 좋은 통합(integration)을 허용한다.
- <300> 도 8c 를 참조하여, 간단히 설명하면, 프로세스에 격리 범위를 연관시키기 위한 방법은 정지된 상태(suspended state)에서 프로세스를 개시하는 단계를 포함한다(단계 882). 원하는 격리 범위와 연관된 규칙이 검색되고(단계 884) 프로세스에 대한 식별자와 검색된 규칙이 메모리 요소에 저장되고(단계 886) 정지된 프로세스는 재개된다(단계 888). 프로세스에 의해 만들어지는 고유 리소스로의 액세스를 위한 호출이 인터셉트 또는 후킹되며(단계 890) 프로세스 식별자와 연관된 규칙이 요청된 리소스로의 액세스를 가상화하는데 사용된다(단계 892).
- <301> 도 8c를 참조하여, 좀더 자세히 설명하면, 프로세스는 정지된 상태에서 개시된다(단계 882). 일 실시예에서, 이 작업을 완성하기 위해 일반적인 론처 프로그램(launcher program)이 사용된다. 이 예에서, 론처는 프로세스를 선택된 격리 범위로 개시하기 위해 명확하게 설계된다. 다른 예에서, 론처는 원하는 격리 범위의 명세(specification)를 입력으로 받아들인다(예를 들어, 커맨드 라인 옵션에 의해).
- <302> 원하는 격리 범위와 연관된 규칙들이 검색된다(단계 884). 일 실시예에서, 하드 디스크 드라이브 또는 고체 상태 메모리 요소와 같은 영구 저장장치로부터 규칙들이 검색된다. 규칙은 관계형 데이터베이스, 플랫폼 파일 데이터베이스, 트리-구조 데이터베이스, 바이너리 트리 구조, 또는 다른 영구 데이터 구조로서 저장될 수 있다. 다른 예에서, 규칙은 규칙을 저장하기 위해 특별히 구성된 데이터 구조에 저장될 수 있다.
- <303> 프로세스 id(PID)와 같은 프로세스에 대한 식별자와 검색된 규칙은 메모리 요소에 저장된다(단계 886). 일 실시예에서, 새로운 프로세스 생성에 관여하는 운영체제 메시지를 수신하는 커널 모드 드라이버가 제공된다. 이 예에서, PID와 검색된 규칙은 드라이버의 콘텍스트에 저장될 수 있다. 다른 예에서, 고유 리소스 요청을 인터셉트하는, 파일 시스템 필터 드라이버, 또는 미니-필터가 제공된다. 이 예에서, PID 와 검색된 규칙은 필터에 저장될 수 있다. 다른 예에서, 모든 인터셉션은 사용자-모드 후킹에 의해 수행되고 PID는 전혀 저장되지 않는다. 프로세스 초기화중에 규칙은 사용자-모드 후킹 장치에 의해 로딩되고, 다른 컴포넌트는 PID에 적용되는 규칙들을 알 필요가 없다.
- <304> 정지된 프로세스는 재개되고(단계 888) 프로세스에 의해 만들어진 고유 리소스로의 액세스에 대한 호출이 인터셉트 또는 후킹되고(단계 890) 프로세스 식별자와 연관된 규칙이 요청된 리소스로의 액세스를 가상화하는데 사용된다(단계 892). 일 실시예에서, 파일 시스템 필터 드라이버 또는 미니 필터 또는 파일 시스템 드라이버는 고유 리소스 액세스에 대한 요청을 인터셉트하고 인터셉트된 요청과 연관된 프로세스 식별자가 규칙 집합과 연관되었는지를 결정한다. 그렇다면, 저장된 프로세스 식별자와 연관된 규칙이 고유 리소스 액세스에 대한 요청을 가상화하는데 사용된다. 그렇지 않다면, 고유 리소스 액세스에 대한 요청은 수정되지 않고 통과된다. 다른 예에서, 동적-링크된 라이브러리(dynamically-linked library)가 새로 생성된 프로세스로 로딩되고 라이브러리는 격리 규칙을 로딩한다. 다른 예에서, 커널 모드 기술들(후킹, 필터 드라이버, 미니-필터)과 사용자-모드 기술들 모두가 고유 리소스 액세스에 대한 호출을 인터셉트하는데 이용된다. 파일 시스템 필터 드라이버가 규칙들을 저장하는 예에서, 라이브러리는 파일 시스템 필터 드라이버로부터 규칙을 로딩할 수 있다.
- <305> 격리 범위와 연관된 프로세스들의 "자식(children)" 프로세스들은 그들의 "부모(parent)" 프로세스의 격리 범위와 연관된다. 일 실시예에서, 이것은 자식 프로세스가 생성되면 파일 시스템 필터 드라이버에 통지하는(notify) 커널 모드 드라이버에 의해 이루어진다. 이 예에서, 파일 시스템 필터 드라이버는 부모 프로세스의 프로세스 식별자가 격리 범위와 연관되었는지를 결정한다. 그렇다면, 파일 시스템 필터 드라이버는 새로 생성된 자식 프로세스에 대한 프로세스 식별자와 부모 프로세스의 격리 범위 사이의 연관성(association)을 저장한다. 다른 예에서, 파일 시스템 필터 드라이버가 커널 모드 드라이버를 사용하지 않고 시스템으로부터 바로 호출될 수 있다. 다른 예에서, 격리 범위와 연관된 프로세스들에서, 새로운 프로세스를 생성하는 운영체제 기능이 후킹 또는 인터셉트된다. 새로운 프로세스를 생성하는 요청이 수신되면, 새로운 자식 프로세스와 부모의 격리 범위 사이의 연관성이 저장된다.
- <306> 일 실시예에서, 범위 또는 서브-범위는 전체 프로세스 대신에 스레드 마다(per-thread) 기본으로 수행되는 격리를 허용하는 개별적인 스레드와 연관될 수 있다. 일 실시예에서, 스레드당 격리는 서비스들과 COM+ 서버들에 대해 이용될 수 있다.
- <307> 일 실시예에서, 격리 환경들은 애플리케이션 스트리밍 클라이언트(552)에 추가적인 기능을 제공하는데 사용될 수 있다. 이 예에서, 애플리케이션 프로그램은 격리 환경내에서 실행된다. 다른 예에서, 검색된 다수의 애플

리케이션 파일은 격리 환경내에 존재한다. 다른 예에서, 로컬 머신(10)상의 레지스트리에 대한 변경이 격리 환경내에서 이루어진다.

- <308> 일 실시예에서, 애플리케이션 스트리밍 클라이언트(552)는 격리 환경(556)을 포함한다. 일 실시예에서, 애플리케이션 스트리밍 클라이언트(552)는 파일에 대한 애플리케이션 요청을 인터셉트하는 파일 시스템 필터 드라이버(564)를 포함한다. 이 예에서, 파일 시스템 필터 드라이버(564)는 현존 파일을 오픈하기 위한 애플리케이션 요청을 인터셉트하고 격리 환경(556)내에 존재하지 않는 파일을 결정한다. 다른 예에서, 파일 시스템 필터 드라이버(564)는 파일이 격리 환경(556)내에 존재하지 않는다는 결정에 대응하여 요청을 스트리밍 서비스(554)로 다시 보낸다. 스트리밍 서비스(554)는 다수의 애플리케이션 파일로부터 파일을 추출(extract)할 수 있고 파일을 격리 환경(556)내에 저장한다. 파일 시스템 필터 드라이버(564)는 그 다음은 파일의 저장된 카피로 파일에 대한 요청에 응답할 수 있다. 일 실시예에서, 파일 시스템 필터 드라이버(564)는 스트리밍 서비스(554)가 검색된 파일 또는 다수의 애플리케이션 파일을 갖지 않는다는 지시와 파일이 격리 환경(556)내에 존재하지 않는다는 결정에 대응하여, 파일에 대한 요청을 파일 서버(540)로 다시 보낸다. 일 실시예에서, 스트리밍 서비스(554)는 이하에서 설명되는 가속 기술 전부 또는 일부를 수행하기 위한 가속 프로그램을 포함할 수 있다.
- <309> 일 실시예에서, 파일 시스템 필터 드라이버(564)는 격리 환경내에서 충돌 또는 불일치 데이터가 출현하는 것을 방지하기 위해 엄격한 분리 규칙을 사용한다. 이 예에서, 사용자 격리 환경에서 리소스에 대한 요청을 인터셉트하는 파일 시스템 필터 드라이버(564)는 요청을 애플리케이션 격리 환경으로 다시 보낼 수 있다. 다른 예에서, 파일 시스템 필터 드라이버(564)는 요청을 시스템 범위로 다시 보내지 않는다.
- <310> 일 실시예에서, 스트리밍 서비스(554)는 필터 드라이버와 통신하기 위해 IOCTL 커맨드를 사용한다. 다른 예에서, 파일 서버(540)로의 통신은 Microsoft SMB 스트리밍 프로토콜로 수신된다.
- <311> 일 실시예에서, 패키징 메커니즘(530)은 가능한 애플리케이션에 따라 퍼블리싱된 파일 유형의 목록을 일목 요연한 파일내에 저장하고 이러한 정보가 애플리케이션 퍼블리싱 소프트웨어에 이용될 수 있도록 만든다. 이 예에서, 패키징 메커니즘(530)은 격리 환경으로의 애플리케이션 프로그램의 설치를 모니터링하는 것으로부터 이 정보를 수신한다. 다른 예에서, 패키징 메커니즘(530)의 사용자는 이 정보를 패키징 메커니즘(530)에 제공한다. 다른 예에서, 액세스 수트 콘솔(520)내의 애플리케이션 퍼블리싱 소프트웨어는 요청된 애플리케이션과 연관될 수 있는 가능한 파일 유형을 액세스 수트 콘솔(520)의 사용자에게 제공하기 위해 일목 요연한 파일을 참조한다. 사용자는 특정 퍼블리싱된 애플리케이션과 연관시키기 위해 파일 유형을 선택한다. 파일 유형은 애플리케이션 열거할 때 로컬 머신(10)에 제공된다.
- <312> 로컬 머신(10)은 클라이언트 에이전트(560)를 포함할 수 있다. 클라이언트 에이전트(560)는 파일 유형과 애플리케이션 프로그램을 연관시키고 이러한 연관에 대응하여 애플리케이션 프로그램의 실행 방법을 선택하는 기능을 제공한다. 일 실시예에서, 클라이언트 에이전트(560)는 근접 프로그램 애플리케이션이다.
- <313> 실행을 위해 애플리케이션 프로그램이 선택되면, 로컬 머신(10)은 애플리케이션 프로그램의 파일 유형과 연관된 실행 방법에 대한 결정을 만든다. 일 실시예에서, 로컬 머신(10)은 파일 유형이 애플리케이션 파일의 검색 및 격리 환경내에서의 실행을 위해 애플리케이션 스트리밍 세션을 요하는 실행 방법과 연관되는 것으로 결정한다. 이 예에서, 로컬 머신(10)은 애플리케이션 프로그램의 로컬 버전을 개시하는 대신에 애플리케이션 스트리밍 클라이언트(552)로 요청을 다시 보낼 수 있다. 다른 예에서, 클라이언트 에이전트(560)는 이러한 결정을 만들 수 있다. 다른 예에서, 클라이언트 에이전트(560)는 요청을 애플리케이션 스트리밍 클라이언트(552)에 다시 보낼 수 있다.
- <314> 일 실시예에서, 애플리케이션 스트리밍 클라이언트(552)는 애플리케이션 프로그램과 연관된 액세스 정보를 원격 머신(30)에 요청한다. 일 실시예에서, 애플리케이션 스트리밍 클라이언트(552)는 액세스 정보를 갖는 실행가능 프로그램을 수신한다. 이 예에서, 애플리케이션 스트리밍 클라이언트(552)는 원격 머신상의 애플리케이션 프로그램 실행으로부터 생성된 애플리케이션-출력 데이터를 로컬 머신(10)상에 디스플레이할 수 있는 실행가능 프로그램을 수신한다. 다른 예에서, 애플리케이션 스트리밍 클라이언트(552)는 애플리케이션 스트리밍 세션을 가로질러 애플리케이션 프로그램을 검색하고 로컬 머신(10)상의 격리 환경내에서 애플리케이션 프로그램을 실행할 수 있는 실행가능 프로그램을 수신한다. 이 예에서, 애플리케이션 스트리밍 클라이언트(552)는 수신된 실행가능 프로그램을 실행할 수 있다. 다른 예에서, 원격 머신(30)은 애플리케이션 결정을 수행하여 로컬 머신(10)에 제공하기 위한 실행가능 프로그램을 선택한다.
- <315> 도 9를 참조하여, 애플리케이션 프로그램을 실행시키기 위한 방법의 단계들의 일 실시예를 도시하고 있다. 단

계 214에 대한 도 7에서 설명한 바와 같이, 로컬 머신(10)은 다수의 애플리케이션 파일을 수신하고 실행한다. 간단히 설명하면, 로컬 머신(10)은 다수의 애플리케이션 파일로의 액세스를 위한 그리고 애플리케이션 스트림을 수신할 수 있는 제1 클라이언트를 실행하기 위한 액세스 정보를 포함하는 파일을 수신한다(단계 902). 로컬 머신(10)은 파일에 대응하여, 다수의 애플리케이션 파일의 ID를 검색한다(단계 904). 로컬 머신(10)은 파일에 대응하여, 다수의 애플리케이션 파일의 실행을 위해 필요한 적어도 하나의 특성을 검색한다(단계 906). 로컬 머신(10)은 로컬 머신(10)이 적어도 하나의 특성을 포함하는지 여부를 판단한다(단계 908). 로컬 머신(10)이 적어도 하나의 특성이 없다는 판단에 대응하여, 로컬 머신(10)은 제2 클라이언트를 실행하고, 제2 클라이언트는 원격 머신상에서의 다수의 애플리케이션 파일의 실행을 요청한다(단계 910).

<316> 도 9를 참조하여, 좀더 자세히 설명하면, 로컬 머신(10)은 다수의 애플리케이션 파일로의 액세스를 위한 그리고 애플리케이션 스트림을 수신할 수 있는 제1 클라이언트를 실행하기 위한 액세스 정보를 포함하는 파일을 수신한다(단계 902). 일 실시예에서, 로컬 머신(10)은 애플리케이션 프로그램을 포함하는 다수의 애플리케이션 파일의 위치의 ID를 포함하는 액세스 정보를 수신한다. 다른 예에서, 로컬 머신(10)은 애플리케이션 프로그램의 실행 요청에 대응하여 파일을 수신한다. 다른 예에서, 액세스 정보는 애플리케이션 서버 또는 파일 서버와 같은 원격 머신(30')상에 다수의 애플리케이션 파일이 존재한다는 표시(indication)를 포함한다. 다른 예에서, 액세스 정보는 로컬 머신(10)이 애플리케이션 스트리밍 세션을 통해 원격 머신(30)으로부터 다수의 애플리케이션 파일을 검색할 수 있다는 표시를 한다.

<317> 로컬 머신(10)은 파일에 대응하여, 다수의 애플리케이션 파일의 ID를 검색한다(단계 904). 일 실시예에서, 로컬 머신(10)은 액세스 정보를 포함하는 파일에 대응하여, 다수의 애플리케이션 파일이 존재하는 원격 머신을 식별한다. 다른 예에서, 로컬 머신(10)은 다수의 애플리케이션 파일을 식별하는 파일을 원격 머신(30)으로부터 검색한다. 일 실시예에서, 다수의 애플리케이션 파일은 애플리케이션 프로그램을 포함한다. 다른 예에서, 다수의 애플리케이션 파일은 다수의 애플리케이션 프로그램을 포함한다. 다른 예에서, 다수의 애플리케이션 파일은 단일 애플리케이션 프로그램의 다수의 버전을 포함한다.

<318> 도 10을 참조하여, 파일 서버(540)와 같은 원격 머신(30')상에 존재하는 다수의 애플리케이션 파일의 일 실시예를 나타내는 흐름도이다. 도 10에서, 패키지로서 언급되는, 다수의 애플리케이션 파일은 하나 이상의 애플리케이션 프로그램의 세가지 다른 버전을 포함하는 애플리케이션 파일들을 포함한다.

<319> 일 실시예에서, 하나 이상의 애플리케이션 프로그램의 버전을 포함하고 패키지내에 저장되는, 애플리케이션 파일의 각각의 서브세트는 타겟이라 한다. 예를 들어, 타겟 1은 워드 처리 애플리케이션 프로그램과 스프레드시트 프로그램의 버전을 포함하고, 이 버전은 Microsoft Windows 2000 운영체제의 영어 버전과 호환가능하다. 타겟 2는 워드 처리 애플리케이션 프로그램과 스프레드시트 프로그램의 버전을 포함하고, 이 버전은 Microsoft XP 운영체제의 영어 버전과 호환가능하다. 타겟 3은 워드 처리 애플리케이션 프로그램과 스프레드시트 프로그램의 버전을 포함하고, 이 버전은 서비스 팩 3의 Microsoft Windows 2000 운영체제의 일어 버전과 호환가능하다.

<320> 도 9를 다시 참조하여, 일 실시예에서, 다수의 애플리케이션 파일을 호스팅하는 원격 머신(30)으로부터 검색된 파일은 다수의 애플리케이션 파일에 포함된 패키지 및 타겟의 디스크립션을 포함한다. 다른 예에서, 원격 머신(30)으로부터 검색된 파일은 로컬 머신(10)에 의한 실행을 위해 요청된 애플리케이션 프로그램을 포함하는 다수의 애플리케이션 파일을 식별한다.

<321> 로컬 머신(10)은 파일에 대응하여, 다수의 애플리케이션 파일의 실행을 위해 필요한 적어도 하나의 특성을 검색한다(단계 906). 일 실시예에서, 로컬 머신(10)은 로컬 머신이 일정한 특성들을 포함하지 않는 한 애플리케이션 프로그램을 실행하지 않을 수 있다. 이 예에서, 다른 애플리케이션 프로그램들은 로컬 머신이 또 다른 애플리케이션 프로그램들에 의해 요구되는 특성들과는 다른 특성들을 포함할 것을 요구한다. 다른 예에서, 로컬 머신(10)은 로컬 머신(10)에 의해 요청된 애플리케이션 프로그램들을 포함하는 다수의 애플리케이션 파일의 실행을 위해 요구되는 적어도 하나의 특성의 ID를 수신한다.

<322> 로컬 머신은 로컬 머신(10)이 적어도 하나의 특성을 포함하는지 여부를 판단한다(단계 908). 일 실시예에서, 로컬 머신(10)은 로컬 머신(10)이 적어도 하나의 특성을 포함하는지를 결정하기 위해 로컬 머신(10)상의 운영체제를 평가한다. 다른 예에서, 로컬 머신(10)은 로컬 머신(10)이 적어도 하나의 특성을 포함하는지를 판단하기 위해 로컬 머신(10)상의 운영체제에 의해 사용되는 언어를 식별한다. 다른 예에서, 로컬 머신(10)은 로컬 머신(10)이 적어도 하나의 특성을 포함하는지를 판단하기 위해 로컬 머신(10)상의 운영체제의 수정 레벨을 식별한다. 다른 예에서, 로컬 머신(10)은 로컬 머신(10)이 적어도 하나의 특성을 포함하는지를 결정하기 위해 로컬 머신(10)상에 존재하는 애플리케이션 프로그램의 애플리케이션 버전을 식별한다. 일 실시예에서, 로컬 머

신(10)은 로컬 머신(10)이 적어도 하나의 특성을 포함하는지를 판단하기 위해서 로컬 머신(10)이 장치 드라이버를 포함하는지 여부를 판단한다. 다른 예에서, 로컬 머신(10)은 로컬 머신(10)이 적어도 하나의 특성을 포함하는지를 판단하기 위해서 로컬 머신(10)이 운영체제를 포함하는지 여부를 판단한다. 다른 예에서, 로컬 머신(10)은 로컬 머신(10)이 적어도 하나의 특성을 포함하는지를 결정하기 위해서 로컬 머신(10)이 다수의 애플리케이션 파일을 실행하기 위한 라이선스를 포함하는지 여부를 판단한다.

<323> 로컬 머신(10)에 적어도 하나의 특성이 없다는 판단에 대응하여, 로컬 머신(10)은 제2 클라이언트를 실행하고, 제2 클라이언트는 원격 머신(30)상의 다수의 애플리케이션 파일의 실행을 요청한다(단계 910). 일 실시예에서, 로컬 머신(10)이 로컬 머신(10)이 적어도 하나의 특성이 없다고 판단하면, 로컬 머신(10)은 애플리케이션 스트림을 수신할 수 있는 제1 클라이언트를 실행하지 않는다. 다른 예에서, 로컬 머신(10)에 적어도 하나의 특성이 없다면, 정책은 로컬 머신(10)은 애플리케이션 스트림을 통해 다수의 애플리케이션 파일을 수신하는 것을 금지한다. 일 실시예에서, 로컬 머신은 로컬 머신(10)이 적어도 하나의 특성을 포함하는 것으로 판단한다. 이 예에서, 로컬 머신(10)은 제1 클라이언트를 실행하고, 제1 클라이언트는 로컬 머신상의 실행을 위해 다수의 애플리케이션 파일을 포함하는 애플리케이션 스트림을 원격 머신(30)으로부터 수신한다.

<324> 일 실시예에서, 로컬 머신(10)에 적어도 하나의 특성이 없다는 판단에 따라, 로컬 머신(10)은 원격 머신상에서의 다수의 애플리케이션 파일의 실행을 요청하는 제2 클라이언트를 실행한다. 이 예에서, 제2 클라이언트는 다수의 애플리케이션 파일을 호스팅하는 원격 머신(30)에 이 요청을 전송한다. 다른 예에서, 원격 머신(30)은 애플리케이션 프로그램을 포함하는 다수의 애플리케이션 파일을 실행하고 애플리케이션 출력 데이터를 생성한다. 다른 예에서, 제2 클라이언트는 원격 머신상에서 다수의 애플리케이션 파일의 실행에 의해 생성된 애플리케이션 출력 데이터를 수신한다. 일 실시예에서, 제2 클라이언트는 ICA(Independent Computing Architecture) 표현 레벨 프로토콜, Remote Desktop Window 표현 레벨 프로토콜 또는 X-Windows 표현 레벨 프로토콜을 통해 애플리케이션 출력 데이터를 수신한다. 다른 예에서, 제2 클라이언트는 로컬 머신(10)상에 애플리케이션 출력 데이터를 디스플레이한다.

<325> 일 실시예에서, 제2 클라이언트는 이 요청을 다수의 애플리케이션 파일을 호스팅하지 않는 원격 머신(30)에 전송한다. 이 예에서, 원격 머신(30)은 다수의 애플리케이션 파일을 호스팅하는 제2 원격 머신(30)에 다수의 애플리케이션 파일을 요청할 수 있다. 다른 예에서, 원격 머신(30)은 애플리케이션 스트리밍 세션을 통해서 제2 원격 머신(30)으로부터 다수의 애플리케이션 파일을 수신할 수 있다. 다른 예에서, 원격 머신(30)은 수신된 다수의 애플리케이션 파일을 격리 환경내에 저장하고 격리 환경내에서 애플리케이션 프로그램을 실행한다. 다른 예에서, 원격 머신은 생성된 애플리케이션 출력 데이터를 로컬 머신상의 제2 클라이언트로 전송한다.

<326> 도 5를 다시 참조하여, 일 실시예에서, 애플리케이션 스트림을 수신할 수 있는 제1 클라이언트는 애플리케이션 스트리밍 클라이언트(552)이다. 애플리케이션 스트리밍 클라이언트(552)는 파일을 수신하고, 다수의 애플리케이션 파일의 ID와 다수의 애플리케이션 파일의 실행을 위해 필요한 적어도 하나의 특성을 검색하고, 로컬 머신(10)이 적어도 하나의 특성을 포함하는지 여부를 판단한다. 다른 예에서, 제2 클라이언트는 클라이언트 에이전트(560)이다. 일 실시예에서, 클라이언트 에이전트(560)는 로컬 머신(10)이 적어도 하나의 특성이 없다는 애플리케이션 스트리밍 클라이언트(552)에 의한 판단에 대응하여 애플리케이션 스트리밍 클라이언트(552)로부터 파일을 수신한다.

<327> 일 실시예에서, 로컬 머신(10)상에서 실행되는 애플리케이션(556)은 Win32FindFirstFile() 및 FindNextFile() API 호출을 사용하여 애플리케이션(556)과 연관된 파일들을 열거한다. 이 예에서, 다수의 애플리케이션 파일은 애플리케이션(556)을 포함한다. 다른 예에서, 다수의 애플리케이션 파일들의 모든 파일들이 로컬 머신(10)상에 존재하는 것은 아니다. 다른 예에서, 스트리밍 서비스(554) 보관 파일내에서 다수의 애플리케이션 파일을 검색하나 다수의 애플리케이션 파일 중 서브세트만을 추출한다. 다른 예에서, 요청된 파일이 로컬 머신(10)상에 존재하는 않더라도, 스트리밍 서비스(554)와 파일 시스템 필터 드라이버(564)는 목록 요청을 만족시키는 기능을 제공한다.

<328> 일 실시예에서, 이러한 기능은 목록 요청을 인터셉트하고 마치 다수의 애플리케이션 파일내의 모든 파일이 로컬 머신(10)상에 존재하는 것과 같이 데이터를 제공함으로써 제공된다. 다른 예에서, 이러한 기능은 IRP_MJ_DIRECTORY_CONTROL IOCTL 과 같은 IOCTL 커맨드로서 전송되는 목록 요청을 파일 시스템 필터 드라이버(564)가 인터셉트하는 것에 의해 제공된다. 파일 시스템 필터 드라이버(564)가 호출을 인터셉트하면, 파일 시스템 필터 드라이버(564)는 이 요청을 스트리밍 서비스(554)로 다시 보낸다. 일 실시예에서, 파일 시스템 필터 드라이버(564)는 스트리밍 서비스(554)에 요청을 다시 보내기 이전에 로컬 머신(10)상의 격리 환경에 요청된 목

록이 존재하는지를 결정한다. 다른 예에서, 스트리밍 서비스(554)는 다수의 애플리케이션 파일내의 파일을 사용하여 요청을 수행하고, 상기 파일은 다수의 애플리케이션 파일과 연관된 디렉토리 구조의 목록을 포함한다. 다른 예에서, 스트리밍 서비스(554)는 목록 요청에 대해 충족시키도록 요청에 대한 응답을 파일 시스템 필터 드라이버(564)에 제공한다.

<329> 도 11을 참조하여, 원격 저장된 파일들에 연관된 파일 메타데이터에 대한 요청에 국부적으로 응답하는(locally responding) 방법에서 행해지는 단계들의 일 실시예를 도시한 흐름도이다. 간단히 설명하면, (i) 원격 머신에 의해 저장된 애플리케이션 프로그램을 나타내는 디렉토리 구조, (ii) 저장된 애플리케이션 프로그램을 포함하는 각각의 파일과 연관된 메타데이터는 원격 머신으로부터 수신된다(단계 1102). 디렉토리 구조 및 메타데이터는 저장된다(단계 1104). 디렉토리 구조내의 특정 파일과 연관된 메타데이터에 대한 액세스를 위한 적어도 하나의 요청이 수신된다(단계 1106). 적어도 하나의 요청은 저장된 메타데이터를 사용하여 응답된다(단계 1108).

<330> 도 11을 참조하여, 좀더 자세히 설명하면, 원격 머신에 의해 저장된 애플리케이션 프로그램을 나타내는 디렉토리 구조, 그리고 저장된 애플리케이션 프로그램을 포함하는 각각의 파일과 연관된 메타데이터가 원격 머신으로부터 수신된다(단계 1102). 일 실시예에서, 스트리밍 서비스(554)가 디렉토리 구조 및 메타데이터를 수신한다. 다른 예에서, 스트리밍 서비스(554)가 저장된 애플리케이션 프로그램을 포함하는 다수의 애플리케이션 파일을 검색할 때 스트리밍 서비스(554)는 디렉토리 구조 및 메타데이터를 수신한다. 다른 예에서, 디렉토리 구조 및 메타데이터는 다수의 애플리케이션 파일내의 파일에 저장된다.

<331> 일 실시예에서, 각각의 파일과 연관된 메타데이터는 적어도 하나의 파일에 대한 선택적 명칭(alternate name)을 포함한다. 다른 예에서, 각각의 파일과 연관된 메타데이터는 적어도 하나의 파일에 대한 짧은 명칭(short name)을 포함하고, 상기 명칭은 8 문자, 점(dot), 그리고 3-문자의 확장명을 갖는다. 다른 예에서, 각각의 파일과 연관된 메타데이터는 적어도 하나의 파일에 대한 선택적 명칭과 적어도 하나의 파일에 대한 짧은 명칭 사이의 매핑을 포함한다. 일 실시예에서, 다수의 애플리케이션 파일내의 파일은 선택적 파일명을 갖는다. 이 예에서, 스트리밍 서비스(554)에 의해 파일이 검색되면, 상기 파일에 대한 선택적 명칭과 적어도 하나의 파일에 대한 짧은 명칭 사이의 매핑에 대응하여 상기 파일은 짧은 명칭과 연관된다.

<332> 디렉토리 구조 및 메타데이터가 저장된다(단계 1104). 일 실시예에서, 디렉토리 구조 및 메타데이터는 격리 환경(556)에 저장된다. 다른 예에서, 디렉토리 구조 및 메타데이터는 캐시 메모리 요소에 저장된다. 다른 예에서, 원격 머신에 의해 저장된 애플리케이션 프로그램을 나타내는 디렉토리 구조는 로컬 머신상에서 실행되는 애플리케이션 프로그램을 나타내는 디렉토리 구조의 목록을 생성하는데 이용된다.

<333> 디렉토리 구조내의 특정 파일과 연관된 메타데이터에 대한 액세스를 위한 적어도 하나의 요청이 수신된다(단계 1106). 일 실시예에서, 이 요청은 파일의 목록에 대한 요청이다. 다른 예에서, 이 요청은 저장된 애플리케이션 프로그램을 포함하는 파일의 복사본이 국부적으로 존재하는지 여부를 결정하기 위한 요청이다.

<334> 일 실시예에서, 요청은 로컬 머신상의 격리 환경내에서 실행되는 애플리케이션(566)에 의해 만들어진다. 다른 예에서, 요청은 애플리케이션 스트리밍 클라이언트(552)에 의해 만들어진다. 다른 예에서, 요청은 애플리케이션(566)을 위해 만들어진다.

<335> 일 실시예에서, 요청이 파일 시스템 필터 드라이버(564)에 의해 인터셉트된다. 다른 예에서, 요청이 파일 시스템 필터 드라이버(564)에 의해 애플리케이션 스트리밍 클라이언트(552)로 전달된다. 다른 예에서, 요청이 파일 시스템 필터 드라이버(564)에 의해 스트리밍 서비스(554)로 전달된다.

<336> 일 실시예에서, 디렉토리를 열거하기 위한 운영체제 기능 또는 기능들을 대체하는 기능에 의해 후킹된다. 다른 예에서, 요청을 인터셉트하기 위해 후킹 동적-링크된 라이브러리가 이용된다. 후킹 기능은 사용자 모드 또는 커널 모드에서 실행될 수 있다. 후킹 기능이 사용자 모드에서 실행되는 예에서, 후킹 기능은 프로세스가 생성될 때 프로세스의 어드레스 공간으로 로딩될 수 있다. 후킹 기능이 커널 모드에서 실행되는 예에서, 후킹 기능은 파일 동작에 대한 요청을 디스패치하는데 이용되는 운영체제 리소스와 연관될 수 있다. 파일 동작의 각각의 유형에 대해 분리된 운영체제 기능이 제공되는 예에서, 각각의 기능이 개별적으로 후킹될 수 있다. 선택적으로, 파일 동작의 여러 개의 유형에 대한 생성 또는 오픈 호출을 인터셉트하는 단일 후킹 기능이 제공될 수 있다.

<337> 저장된 메타데이터를 이용하여 적어도 하나의 요청에 응답한다(단계 1108). 일 실시예에서, 파일 시스템 필터 드라이버(564)가 요청에 응답한다. 다른 예에서, 애플리케이션 스트리밍 클라이언트(552)가 요청에 응답한다. 다른 예에서, 스트리밍 서비스(554)가 요청에 응답한다. 일 실시예에서, 요청에 응답하기 위해 저장된 메타데

이터가 액세스된다. 다른 예에서, 파일의 원격 복사본이 국부적으로 존재한다는 실패 표시로 요청에 응답한다.

<338> 일 실시예에서, 수신된 메타데이터에 대응하여 Windows Operating System FindFirst 동작이 이루어진다. 다른 예에서, 수신된 메타데이터에 대응하여 Windows Operating System FindNext 동작이 이루어진다. 다른 예에서, 수신된 메타데이터에 대응하여 디렉토리 구조내의 루트 노드(root node)를 식별하기 위한 동작이 이루어진다. 일 실시예에서, WIN32_FIND_DATA API 와 같은 애플리케이션 층 API 가 상기 동작에 응답하는데 이용된다. 다른 예에서, 상기 동작에 응답하는데 FILE_BOTH_DIR_INFORMATION 과 같은 커널 층 API 가 이용된다.

<339> 일 실시예에서, 메타데이터가 디렉토리 구조내의 노드와 연관된 액세스 시간을 식별하기 위한 동작을 만족시킨다. 다른 예에서 메타데이터가 디렉토리 구조내의 노드와 연관된 수정 시간을 식별하기 위한 동작을 만족시킨다. 다른 예에서, 메타데이터가 디렉토리 구조내의 수정된 노드를 식별하기 위한 동작을 만족시킨다.

<340> 도 12를 참조하여, 원격으로 저장된 파일과 연관된 파일 메타데이터에 대한 요청을 국부적으로 응답하기 위한 시스템의 일 실시예를 나타내고 있으며, 상기 시스템은 스트리밍 서비스(554), 파일 시스템 필터 드라이버(564), 디렉토리 구조(570), 다수의 애플리케이션 파일(572), 메타데이터(574), 캐시 메모리 요소(576)를 포함한다. 간단히 설명하면, 디렉토리 구조(570)는 적어도 하나의 애플리케이션 프로그램과 연관된 다수의 파일을 식별한다. 메타데이터(574)는 사익 다수의 파일 중 적어도 하나와 연관되며, 다수의 파일 중 적어도 하나는 원격 머신상에 존재한다. 일 실시예에서, 디렉토리 구조(570)는 메타데이터(574)를 포함한다. 캐시 메모리 요소(576)는 디렉토리 구조(570)를 저장한다. 파일 시스템 필터 드라이버(564)는 적어도 하나의 원격으로 저장된 파일과 연관된 메타데이터에 액세스하기 위한 요청을 인터셉트하고, 캐시 메모리 요소에 액세스하고, 저장된 디렉토리 구조를 이용하여 적어도 하나의 요청에 응답한다.

<341> 일 실시예에서, 스트리밍 서버(554)는 디렉토리 구조(570)와 메타데이터(574)를 수신한다. 이 예에서, 디렉토리 구조(570)는 애플리케이션 프로그램과 연관된 다수의 애플리케이션 파일(572)을 나타내고, 다수의 애플리케이션 파일(572)은 원격 머신(30)과 같은 원격 머신상에 존재한다. 다른 예에서, 메타데이터(574)는 Windows Operating System FindFirst 요청에 응답하기 위한 정보를 포함한다. 다른 예에서, 메타데이터(574)는 Windows Operating System FindNext 요청에 응답하기 위한 정보를 포함한다. 다른 예에서, 메타데이터(574)는 디렉토리 구조내의 루트 노드의 ID에 대한 요청에 응답하기 위한 정보를 포함한다. 다른 예에서, 메타데이터(574)는 디렉토리 구조내의 노드의 ID에 대한 요청에 응답하기 위한 정보를 포함한다. 일 실시예에서, WIN32_FIND_DATA API 와 같은 애플리케이션 층 API 가 상기 동작에 응답하는데 이용된다. 다른 예에서, 상기 동작에 응답하는데 FILE_BOTH_DIR_INFORMATION 과 같은 커널 층 API 가 이용된다.

<342> 일 실시예에서, 파일에 대한 소량의 메타데이터(574)는 메타데이터 표시자(metadata indicator)를 가상 명칭에 접미사로 붙이는 것(suffixing)과 같이, 리터럴 파일명에 바로 저장될 수 있으며, 메타데이터 표시자는 특정 메타데이터 상태와 유일하게 연관된 스트링(string)이다. 메타데이터 표시자는 메타데이터의 하나 또는 여러 비트를 표시하거나 또는 인코딩할 수 있다. 가상 파일명에 의한 파일 액세스에 대한 요청은 메타데이터 표시자의 존재로 인한 리터럴 파일명의 가능한 변경을 체크하고, 파일 자체의 명칭을 검색하기 위한 요청은 리터럴 명칭으로 응답하기 위해 후킹되거나 또는 인터셉트된다. 다른 예에서, 파일에 대한 하나 이상의 선택적 명칭(alternate name)이 가상 파일 명칭과 메타데이터 표시자로부터 형성될 수 있으며, 파일 시스템에 의해 제공되는 하드 링크 또는 소프트 링크 기능을 사용하여 생성될 수 있다. 링크의 명칭을 사용하여 파일에 액세스하는 요청이 주어지면 파일이 발견되지 않는다고 표시하는 것에 의해, 이러한 링크들의 존재는 격리 환경에 의해 애플리케이션으로부터 숨겨질 수 있다. 특정 링크의 존재 또는 부재는 각각의 메타데이터 표시자에 대한 메타데이터의 하나의 비트를 지시할 수 있고, 또는 메타데이터의 여러 비트를 지시하기 위한 다수의 상태를 취할 수 있는 메타데이터 표시자를 갖는 링크가 존재할 수 있다. 파일 시스템이 선택적 파일 스트림(alternate file streams)을 지원하는 다른 예에서, 메타데이터의 여러 비트를 표시하는 스트림 크기를 갖는, 선택적 파일 스트림이 메타데이터를 포함하기 위해 생성될 수 있다. 다른 예에서, 파일 시스템은 파일 시스템내의 각각의 파일에 대한 제3자(3rd party) 메타데이터를 저장하는 능력을 바로 제공할 수 있다. 다른 예에서, 분리된 서브-범위가 삭제된 파일을 기록하는데 이용될 수 있고, 그 서브-범위내의 파일의 존재(푸를 레이스홀더로서 마크되지 않은)는 파일이 삭제되었다는 것을 의미하기 위한 것이다.

<343> 일 실시예에서, 사용자 격리 환경, 애플리케이션 격리 환경 그리고 시스템 범위내의 데이터는 애플리케이션을 나타내는 디렉토리 구조의 로컬 목록을 형성하기 위해 조합된다. 다른 예에서, 스트리밍 서비스(554)는 애플리케이션 격리 환경을 파플레이트(populate)하기 위해 메타데이터(574)와 디렉토리 구조(570)에 액세스한다. 다른 예에서, 파일 시스템 필터 드라이버(564)는 디렉토리 구조의 로컬 목록을 생성한다. 다른 예에서, 디렉토리

구조의 로컬 목록은 다수의 애플리케이션 파일(572)내의 적어도 하나의 파일을 식별하고, 적어도 하나의 파일은 로컬 머신이 아니라 원격 머신상에 존재한다. 일 실시예에서, 디렉토리 구조의 로컬 목록은 캐시 메모리 요소(576)상에 저장된다. 다른 예에서, 스트리밍 서비스(554)는 애플리케이션 격리 환경과 디렉토리 구조의 로컬 목록을 생성한다.

<344> 일 실시예에서, 파일 시스템 필터 드라이버(564)는 디렉토리 구조의 로컬 목록에 대한 액세스를 위해 시스템 범위에 전송된 요청을 인터셉트한다. 다른 예에서, 파일 시스템 필터 드라이버(564)는 상기 요청을 인터셉트한 이후에 로컬 목록을 생성한다. 다른 예에서, 파일 시스템 필터 드라이버(564)는 로컬 목록에 대한 요청을 사용자 격리 환경으로 다시 보낸다. 다른 예에서, 파일 시스템 필터 드라이버(564)는 로컬 목록에 대한 요청을 애플리케이션 격리 환경에 다시 보낸다.

<345> 일 실시예에서, 파일 시스템 필터 드라이버(564)는 디렉토리의 로컬 목록내의 파일에 액세스하기 위한 요청을 인터셉트하고, 상기 파일은 원격 머신상에 존재한다. 일예에서, 파일 시스템 필터 드라이버(564)는 스트리밍 서비스(554)에 의한 파일의 검색을 요청하며, 이에 대해서는 도 13을 참조하여 이하에서 자세히 설명한다.

<346> 격리 환경내에서 실행되는 애플리케이션이 파일에 대한 요청을 하는 것에 따라, 필터 드라이버는 이러한 요청들을 인터셉트한다. 요청이 파일을 오픈하는 것이면, 격리 환경에 의해 이 요청이 충족될 수 있는 것인지 여부를 결정하기 위해, 필터 드라이버는 먼저 요청을 격리 환경에 다시 보낼 것이다. 호출이 성공적이면, 필터 드라이버는 격리 환경내에 배치된 파일의 인스턴스로 요청에 응답할 것이다.

<347> 그러나 요청된 파일이 격리 환경내에 존재하지 않는다면, 필터 드라이버는 다수의 애플리케이션 파일로부터 파일을 검색하기 위해 요청을 스트리밍 서비스(554)에 보내고, 요청이 완성될 때까지 차단하고, 그 다음 원래 오픈을 다시 시도한다. 일 실시예에서, 필터 드라이버로부터의 요청을 수신함에 따라 다수의 애플리케이션 파일로부터 파일을 검색하기 위한 스트리밍 서비스(554)의 기능은 "온-디맨드 캐싱(on-demand caching)" 이라 한다.

<348> 도 13을 참조하여, 국부적으로 실행되는 애플리케이션 프로그램과 연관된 디렉토리 구조내의 파일에 액세스하기 위한 방법의 단계들을 나타내는 흐름도이다. 간단히 설명하면, 파일에 대한 애플리케이션의 요청이 인터셉트된다(단계 1302). 요청은 제1 격리 환경으로 다시 보내진다(단계 1304). 요청된 파일이 제1 격리 환경내에 존재하지 않는다는 결정이 이루어진다(단계 1306). 원격 머신상에 존재하는 다수의 애플리케이션 파일과 연관된 디렉토리 구조의 목록내에 파일이 식별된다는 결정에 대응하여 제2 격리 환경으로 요청이 다시 보내진다(단계 1308). 제2 격리 환경은 파일을 갖지 않고 파일이 목록내에서 식별된다는 결정에 대응하여, 요청된 파일이 원격 머신으로부터 검색된다(단계 1310).

<349> 도 13을 참조하여, 좀더 자세히 설명하면, 파일로 액세스하기 위한 애플리케이션의 요청이 인터셉트된다(단계 1302). 일 실시예에서, 파일 시스템 필터 드라이버에 의해 요청이 인터셉트된다. 다른 예에서, 파일 시스템 필터 드라이버는 파일로의 액세스에 대한 모든 요청을 인터셉트한다. 다른 예에서, 애플리케이션 스트리밍 클라이언트(552)가 요청을 인터셉트한다. 일 실시예에서, 실행가능 파일에 대한 액세스를 위한 애플리케이션의 요청, 로컬 머신상에서 실행되는 애플리케이션의 일부가 인터셉트된다.

<350> 요청은 제1 격리 환경으로 다시 보내진다(단계 1304). 일 실시예에서, 애플리케이션은 제1 격리 환경내에서 실행된다. 일 실시예에서, 애플리케이션은 워드 프로세싱 프로그램 또는 스프레드시트 프로그램과 같은 애플리케이션 프로그램이다. 다른 예에서, 애플리케이션은 애플리케이션 스트리밍 클라이언트(552)이다. 다른 예에서, 애플리케이션은 로컬 머신(10)의 사용자를 위해 애플리케이션 프로그램을 개시하려고 시도하는 애플리케이션 스트리밍 클라이언트(552)내의 컴포넌트이다. 다른 예에서, 파일 시스템 필터 드라이버는 요청을 제1 격리 환경으로 다시 보낸다.

<351> 요청된 파일이 제1 격리 환경내에 존재하지 않는다는 결정이 만들어진다(단계 1306). 일 실시예에서, 파일 시스템 필터 드라이버는 요청된 파일이 제1 격리 환경내에 존재하지 않는다는 지시를 수신한다.

<352> 원격 머신상에 존재하는 다수의 애플리케이션 파일과 연관된 디렉토리 구조의 목록내에서 파일이 식별되었다는 결정에 대응하여 제2 격리 환경으로 요청이 다시 보내진다(단계 1308). 일 실시예에서, 제1 애플리케이션의 실행에 대한 액세스 정보와 함께 디렉토리 구조의 목록이 수신된다. 다른 예에서, 목록은 제2 애플리케이션을 포함하는 다수의 애플리케이션 파일을 식별한다. 이 예에서, 제1 애플리케이션은 제2 애플리케이션의 로컬 복사본이다.

<353> 제2 격리 환경이 파일을 갖지 않고 파일이 목록내에서 식별된다는 결정에 대응하여, 요청된 파일이 원격 머신으로부터 검색된다(단계 1310). 일 실시예에서, 요청된 파일은 제2 원격 머신으로부터 검색된다. 다른 예에서,

요청된 파일은 파일 서버로부터 검색된다. 일 실시예에서, 디렉토리 구조의 목록은 로컬 머신상에 존재하는 다수의 애플리케이션 파일을 식별한다. 다른 예에서, 디렉토리 구조의 목록은 다수의 애플리케이션 파일이 로컬 머신상에 존재하는 것을 지시한다. 이 예에서, 디렉토리 구조의 목록이 로컬 머신상에 존재하는 것으로 지시한, 애플리케이션이 다수의 애플리케이션 파일내의 파일에 대한 액세스를 요청하면, 액세스 요청의 인터셉션에 따라 파일 서버로부터 파일이 얻어진다. 다른 예에서, 파일 서버는 요청된 파일을 로컬 머신에 스트리밍한다. 다른 예에서, 요청된 파일을 수신함에 따라, 요청된 파일이 제2 격리 환경내에 저장된다. 다른 예에서, 디렉토리 구조의 목록이 로컬 머신상에 존재하는 것으로 지시한, 애플리케이션이 다수의 애플리케이션 파일내의 파일에 대한 액세스를 요청하면, 로컬 캐시로부터 파일의 복사본이 애플리케이션에 제공된다.

<354> 일 실시예에서, 요청된 파일은 암호화된다(encrypted). 다른 예에서, 요청된 파일은 부호화된 형식으로 저장된다. 다른 예에서, 애플리케이션이 요청된 파일에 액세스하기 위한 인증이 없다면, 파일을 요청하는 애플리케이션은 요청된 파일을 암호해제할 수 없게 금지될 수 있다.

<355> 일 실시예에서, 디렉토리 구조의 목록이 파일을 식별하지 못한다는 결정이 만들어진다. 이 예에서, 파일 액세스에 대한 요청은 제1 격리 환경 및 제2 격리 환경의 외부 환경으로 다시 보내질 수 있다.

<356> 일 실시예에서, 파일 액세스에 대한 제2 요청이 인터셉트된다. 이 예에서, 파일 액세스에 대한 요청은 제2 애플리케이션에 의해 만들어진다. 다른 예에서, 제2 애플리케이션은 제3 격리 환경내에서 실행될 수 있다. 다른 예에서, 파일이 목록내에서 열거되었고 제2 격리 환경이 파일을 갖는다는 결정에 대응하여, 요청이 제2 격리 환경으로 다시 보내진다. 파일 서버로부터의 파일 수신에 따라 로컬 머신이 파일을 제2 격리 환경내에 저장한 것으로 결정이 만들어질 수 있다. 다른 예에서, 파일은 제3 격리 환경내에 저장된다.

<357> 도 14를 참조하여, 애플리케이션과 연관된 디렉토리 구조내의 파일에 액세스하기 위한 시스템의 일 실시예가 나타나 있다. 간단히 설명하면, 로컬 머신(10)은 애플리케이션 스트리밍 클라이언트(552), 스트리밍 서비스(554), 격리 환경(556), 파일 시스템 필터 드라이버(564), 그리고 제1 애플리케이션(566)을 포함한다. 로컬 머신(10)은 파일 서버(540), 원격 머신(30), 웹 인터페이스(558), 그리고 제2 애플리케이션(566')과 상호작용할 수 있다.

<358> 로컬 머신(10)은 제1 애플리케이션(566)을 실행하기 위해 애플리케이션 스트리밍 클라이언트(552)를 초기화한다. 일 실시예에서, 애플리케이션 스트리밍 클라이언트(552)는 제1 애플리케이션(566)을 검색하고 실행하기 위해 스트리밍 서비스(554)를 초기화한다. 일 실시예에서, 다수의 애플리케이션 파일은 제1 애플리케이션(566)을 포함한다. 이 예에서, 스트리밍 서비스(554)는 다수의 애플리케이션 파일을 검색하고 격리 환경(566)내에 저장한다. 이 예에서, 스트리밍 서비스(554)는 다수의 애플리케이션 파일이 존재하는 원격 머신의 위치를 식별하나 다수의 애플리케이션 파일을 검색하지는 않는다. 다른 예에서, 스트리밍 서비스(554)는 다수의 애플리케이션 파일내의 파일들의 서브셋을 검색한다. 다른 예에서, 스트리밍 서비스(554)는 다수의 애플리케이션 파일을 갖는 보관 파일을 검색한다.

<359> 일 실시예에서, 제1 애플리케이션(566)은 원격 머신(30)상에 존재하는 제2 애플리케이션(566')의 로컬 복사본을 포함한다. 다른 예에서, 다수의 애플리케이션 파일은 원격 머신(30)상에 존재하고 원격 머신(30)상에 존재하는 제2 애플리케이션(566')을 포함한다. 다른 예에서, 제2 애플리케이션(566')을 실행하기 위해, 로컬 머신상의 제1 애플리케이션(566)을 생성하는, 로컬 머신(10)은 다수의 애플리케이션 파일을 검색하고, 제1 애플리케이션(566)을 실행한다. 일 실시예에서, 애플리케이션들(566, 566')은 워드 프로세싱 애플리케이션 또는 스프레드시트 애플리케이션 또는 프레젠테이션 애플리케이션과 같은 사용자 애플리케이션들이다.

<360> 일 실시예에서, 다수의 애플리케이션 파일은 원격 머신(30)상의 다수의 애플리케이션 파일과 연관된 디렉토리 구조를 식별하는 파일을 포함한다. 이 예에서, 파일은 다수의 애플리케이션 파일내의 각각의 애플리케이션 파일에 대한 메타데이터를 포함한다. 이 예에서, 스트리밍 서비스(554)는 도 12에서 설명한 바와 같이, 다수의 애플리케이션 파일과 연관된 디렉토리 구조의 목록을 생성하기 위해 파일로부터 메타데이터를 검색한다. 이 예에서, 스트리밍 서비스(554)는 제2 애플리케이션(566')을 포함하는 다수의 애플리케이션 파일과 연관된 디렉토리 구조의 목록을 저장한다. 일 실시예에서, 스트리밍 서비스(554)는 이러한 목록을 제2 격리 환경내에 저장한다.

<361> 일 실시예에서, 스트리밍 서비스(554)는 제1 애플리케이션(566)과 연관된 초기 실행가능 파일을 검색한다. 다른 예에서, 스트리밍 서비스(554)는 초기 실행가능 파일의 검색에 따라 로컬 머신(10)상에서 제1 애플리케이션(566)을 실행한다. 다른 예에서, 제1 애플리케이션(566)은 제1 애플리케이션(566)의 연속된 실행을 위해 파일

들이 필요함에 따라 다수의 애플리케이션 파일내의 다른 파일들에 대한 액세스를 요청한다. 일 실시예에서, 제 1 애플리케이션(566)은 격리 환경(556)내에서 실행된다.

<362> 파일 시스템 필터 드라이버(564)는 다수의 애플리케이션 파일내의 파일에 대한 액세스를 위해 격리 환경(556)내에서 실행되는 제1 애플리케이션(566)에 의한 요청을 인터셉트한다. 파일 시스템 필터 드라이버(564)는 요청을 격리 환경(556)에 다시 보낸다. 요청된 파일이 격리 환경(556)내에 존재하면, 요청된 파일에 대한 액세스가 제 1 애플리케이션(566)에 제공된다.

<363> 요청된 파일이 격리 환경(556)내에 존재하지 않으면, 파일 시스템 필터 드라이버(564)는 요청을 제2 격리 환경으로 다시 보낸다. 일 실시예에서, 제2 격리 환경은 스트리밍 서비스(554)에 의해 생성되고 제2 애플리케이션(566')을 포함하는 다수의 애플리케이션 파일과 연관된 디렉토리 구조의 목록을 포함한다. 다른 예에서, 요청된 파일이 디렉토리 구조의 목록내에서 식별된다는 결정이 이루어진다.

<364> 일 실시예에서, 스트리밍 서비스(554)는 격리 환경(556)에 신호(semaphore)를 제공한다. 이 예에서, 신호를 사용하여, 파일 시스템 필터 드라이버(564)는 다수의 애플리케이션 파일내의 파일에 대한 액세스가 요구된다는 것을 스트리밍 서비스(554)에 지시한다. 다른 예에서, 파일 시스템 필터 드라이버(564)는 파일에 대한 액세스가 요구된다는 것을 스트리밍 서비스(554)에 지시하기 위해 스레드(thread)를 사용한다.

<365> 파일 시스템 필터 드라이버(564)로부터의 통지를 수신함에 따라, 스트리밍 서비스(554)는 다수의 애플리케이션 파일로부터 요청된 파일을 검색한다. 이 예에서, 스트리밍 서비스(554)는 요청된 파일을 제2 애플리케이션 격리 환경내에 저장한다. 일 실시예에서, 파일 액세스에 대한 요청은 다수의 애플리케이션 파일로부터 검색되고 제2 격리 환경내에 저장된 파일의 인스턴스로 이루어진다. 다른 예에서, 요청된 파일은 또한 제1 격리 환경내에 저장된다.

<366> 일 실시예에서, 제2 격리 환경(the second isolation environment)이 그 파일을 포함하지 않으며 그 파일은 목록(enumeration)내에서 확인된다고 판단한다. 이러한 경우 일 실시예에서, 그 파일은 제2 애플리케이션(566')을 포함하는 복수의 애플리케이션 파일과 연관된 디렉토리 구조 목록에서 식별되고, 그 파일은 복수의 애플리케이션 파일 중의 파일이다. 이러한 경우 다른 실시예에서, 스트리밍 서비스(streaming service; 554)는 원격 머신에서 그 파일을 검색(retrieve)하지 않았다. 이러한 경우 다른 실시예에서, 스트리밍 서비스(554)는 요청된 파일을 포함하는 복수의 애플리케이션 파일을 검색하지 않았다. 이러한 경우 다른 실시예에서, 스트리밍 서비스(554)는 보관 파일(archieved file)에서 복수의 애플리케이션 파일은 검색했으나, 보관 파일에서 요청된 파일을 검색하지는 않았다.

<367> 일 실시예에서, 스트리밍 서비스(554)는 파일 시스템 필터 드라이버와 통신하는 트랜시버를 포함한다. 또 다른 실시예에서, 트랜시버는 파일 시스템 필터 드라이버로부터 리디렉팅되는 요청(redirected request)을 수신한다. 또 다른 실시예에서, 트랜시버는 파일에 대한 요청을, 요청된 파일을 호스팅하는 원격 머신으로 전달한다. 일 실시예에서, 원격 머신은 파일 서버(540)이다. 또 다른 실시예에서, 요청은 원격 머신(30)으로 전달되고, 원격 머신(30)이 그 요청을 파일 서버(540)로 라우팅(route)한다. 일 실시예에서, 파일 서버(540)는 요청된 파일을 로컬 머신(10)상의 트랜시버로 스트리밍한다. 다른 실시예에서, 원격 머신(30)은 요청된 파일을 로컬 머신(10)상의 트랜시버로 스트리밍한다. 또 다른 실시예에서, 파일 서버(540)로부터 요청된 파일을 수신하면, 트랜시버는 수신된 파일을 제2 격리 환경에 저장한다.

<368> 일 실시예에서, 파일 시스템 필터 드라이버(564)는 제3 격리 환경에서 로컬 머신(10)상에서 실행 중인 제3 애플리케이션(566'')에 의해 이루어지는 파일에 대한 제2 액세스 요청을 인터셉트(intercept)한다. 또 다른 실시예에서 파일 시스템 필터 드라이버(564)는 파일에 대한 액세스 요청을 제2 격리 환경으로 리디렉팅한다. 또 다른 실시예에서, 파일 시스템 필터 드라이버(564)는, 제3 애플리케이션(566'')에 의한 액세스 요청을 인터셉트하기 이전에 스트리밍 서비스(554)가 수신 파일을 제2 격리 환경에 저장했다고 판단한다.

<369> 일 실시예에서, 초기화시, 스트리밍 서비스(554)는 애플리케이션 프로그램의 실행 이전에 격리 환경에 캐시들(populate) 수도 있다. 일 실시예에서, 스트리밍 서비스(554)는 격리 환경내로 레지스트리 파일을 설치한다. 이러한 경우 다른 실시예에서, 스트리밍 서비스(554)는 파일의 긴 명칭과 짧은 명칭간의 매핑을 저장한다.

<370> 일 실시예에서, 로컬 머신 상에서의 공간을 절약하기 위해, 캐시의 크기가 제한될 수도 있다. 일 실시예에서, 캐시가 그 크기 제한에 가까워지면, 신규 파일들을 위한 공간을 확보하기 위하여 캐시내 가장 오래된 파일들이 자동으로 제거될 것이다. 이러한 경우 일 실시예에서, 파일의 에이지(age)는 운영 체제가 유지하며 '마지막 액세스' 타임스탬프의 시각을 나타내는 타임스탬프에 의해서 결정된다. 파일의 에이지에 더하여, 파일 유형을 고

려할 수도 있다 - 바이너리 실행 파일(.EXE, .DLL 등)이 비슷하게 오래된 다른 유형의 파일들에 비해 더 오래 유지될 수 있다.

<371> 초기화시, 스트리밍 서비스(554)는 캐시내에 파일들을 목하(currently) 열거하고 캐시의 전체 크기를 결정할 수도 있다. 캐시에 파일이 추가된 후, 스트리밍 서비스(554)는 신규 파일, 그 위치 및 그 크기의 캐시 시스템을 통보하기 위하여 격리 환경(556) 및 스트리밍 서비스(554) 중 하나에 의해 함수를 호출한다. 실행 중인 현재의 캐시의 전체 크기에 각각의 새로이 캐싱된 파일들의 크기가 추가된다. 이러한 새로운 총계는 그 후 캐시 크기 제한에 대해서 비교되고, 제한이 초과되었으면, 코드는 캐시를 에이징(age)하기 위한 스레드(thread)를 시작한다(fires off). 이러한 스레드의 인스턴스는 임의의 주어진 시각에 단 하나만 실행 중일 수 있다.

<372> 스레드는 캐시내에서 목하 전체 파일의 리스트를 생성하고, 마지막 액세스 타임스탬프에 의해서 이 리스트를 정렬하고, 그 다음에 스레드 빠져나가기 기준(exit criteria)을 충족시킬만큼 충분한 디스크 공간을 확보할 때까지 파일들을 삭제하여 리스트를 줄여나가기 시작한다. 빠져나가기 기준은 백분율 제한(디폴트 값은 10%임)으로서 결정되는 제한 미만의 레벨까지 캐시의 크기를 감소시키는 것을 기본으로 한다. 제한을 초과하는 것을 방지하는 데 필요한 것 이상으로 더 많이 삭제하면, 신규 파일이 추가될 때마다 캐시를 조작하게 되는 번거로움을 없앨 수 있다.

<373> 일 실시예에서, 스트리밍 서비스(554)는 애플리케이션 프로그램을 포함하는 복수의 애플리케이션 파일내의 모든 파일을 압축 파일의 형태로 로컬 머신(10)에 복사하는 기능을 제공한다. 이러한 기능을 "프리-캐싱(pre-caching)"이라 한다. 이러한 경우 일 실시예에서, 그 후에 애플리케이션 프로그램이 실행되면, 모든 패키지 요청은 네트워크를 통해 이리저리 왔다 갔다하는 것이 아니라 그 로컬 복사로 간다. 이러한 실시예에 의해서 로컬 머신(10)의 사용자는 그 사용자가 네트워크를 액세스하지 않는 시각에도 그 애플리케이션 프로그램을 실행시킬 수 있게 한다.

<374> 원격 머신(30)은 로컬 머신(10)에 의하여 애플리케이션 사용을 모니터링하는 기능(functionality)을 포함한다. 원격 머신(30)은 예를 들어, 애플리케이션의 실행 또는 종료시에 로컬 머신(10)에 의해 사용되는 각 애플리케이션의 상태를 모니터링할 수도 있다. 일 실시예에서, 원격 머신(30)은 로컬 머신(10)이 실행하는 애플리케이션의 상태에 관한 메시지를 로컬 머신(10)이 전달할 것을 요구한다. 또 다른 실시예에서, 로컬 머신(10)은 원격 머신(30)이 상주하는 네트워크에 연결할 때, 로컬 머신(10)은 그 로컬 머신(10)이 네트워크에 연결했음을 나타내는 메시지를 전달한다.

<375> 일 실시예에서, 로컬 머신(10)은 로컬 머신(10)이 원격 머신(30)과 상호작용하고 일 이상의 애플리케이션을 실행시키는 경우 세션을 갖는 것이라 한다. 또 다른 실시예에서, 원격 머신(30)은 세션 중에는 로컬 머신이 원격 머신으로부터 수신한 애플리케이션의 실행 권한을 유지할 것을 요구한다. 또 다른 실시예에서, 세션은 원격 머신이 할당하는 고유 세션 식별자를 가진다.

<376> 일 실시예에서, 로컬 머신(10)은 애플리케이션 프로그램을 수신하고 실행하도록 상호작용하는 원격 머신(30)으로 메시지를 전달한다. 또 다른 실시예에서, 로컬 머신(10)은 원격 머신(30)으로부터 세션 관리 서버(562)와 같은 제2 원격 머신의 식별자를 수신하며, 제2 원격 머신은 로컬 머신(10)상의 세션과 관련하여 전달된 모든 메시지를 수신하고 저장한다.

<377> 일 실시예에서, 세션 관리 서버(562)는 라이선스 관리(license management) 및 세션 모니터링 서비스를 제공하는 원격 머신(30)이다. 이러한 경우 일 실시예에서, 세션 관리 서버(562)는 이러한 서비스들을 제공하는 서버 관리 서브시스템(508)을 포함한다.

<378> 일 실시예에서, 로컬 머신(10)은 메시지를 세션 관리 서버(562)로 직접 전달한다. 또 다른 실시예에서, 로컬 머신(10)은 메시지를 원격 머신(30)으로 전달하고, 원격 머신(30)이 그 메시지를 로컬 머신(10)의 식별자와 함께 세션 관리 서버(562)로 전달한다.

<379> 로컬 머신(10)은 하트비트 메시지(heartbeat message)를 원격 머신(30)으로 전달할 수도 있다. 일 실시예에서, 하트비트 메시지는 라이선스 요청을 포함한다. 이러한 실시예에서, 로컬 머신(10)은 그 로컬 머신(10)이 그 실행 허가(authorization)를 요청한 애플리케이션 프로그램과 관련된 액세스 정보를 수신한 이후에 그 하트비트 메시지를 전달할 수도 있다. 로컬 머신(10)은 애플리케이션을 실행하기 이전에 하트비트 메시지를 전달할 수도 있다. 일 실시예에서, 로컬 머신(10)은 액세스 정보와 함께 수신한 개시 티켓(launch ticket)을 하트비트 메시지와 함께 포함한다. 이러한 실시예에서, 원격 머신(30)은 개시 티켓의 검증이 성공하면, 로컬 머신(552)에 라이선스를 부여할 수도 있다.

- <380> 또 다른 실시예에서, 하트비트 메시지는 애플리케이션의 실행을 개시했다는 표시를 포함한다. 또 다른 실시예에서, 하트비트 메시지는 로컬 머신이 애플리케이션의 실행을 종료했다는 표시를 포함한다. 또 다른 실시예에서, 하트비트 메시지는 애플리케이션을 실행하는 데 오류가 있다는 표시를 포함한다.
- <381> 일 실시예에서, 하트비트 메시지는 제2 세션 관리 서버, 예를 들어 세션 관리 서버(562)의 식별자에 대한 요청을 포함한다. 또 다른 실시예에서, 하트비트 메시지는 원격 머신(30)이 상주하는 네트워크에 로컬 머신(10)이 연결했다는 표시를 포함한다.
- <382> 일 실시예에서, 하트비트 메시지는 애플리케이션 스트리밍 세션을 리셋하는 요청을 포함한다. 이러한 경우 일 실시예에서, 로컬 머신(10)은 에러가 발생했고, 원격 머신(30)이 상주하는 네트워크와 로컬 머신(10)간의 연결이 끊긴 경우 이러한 하트비트 메시지를 전달한다. 이러한 경우 다른 실시예에서, 로컬 머신(10)은 세션과 연관된 하트비트 메시지 정보를 함께 전달한다. 이러한 경우 다른 실시예에서, 원격 머신(30)은 세션의 기간이 만료되지 않았으면 세션과 관련된 데이터를 로컬 머신(10)으로 전달할 수 있다.
- <383> 이러한 경우 다른 실시예에서, 원격 머신(30)이 네트워크에서 연결 해제되면, 로컬 머신(10)은 원격 머신(30)으로 전달되는 하트비트 메시지에 대한 응답을 수신할 수 없다. 일 실시예에서, 로컬 머신(10)은 원격 머신(30)으로 세션 리셋을 요청하는 메시지를 전달하여 세션을 재구축할 수 있다. 또 다른 실시예에서, 로컬 머신(10)은 제2 원격 머신(30)으로 세션 리셋을 요청하는 메시지를 전달하여 세션을 재설정할 수도 있다. 일 실시예에서, 원격 머신(30)이 네트워크에 재연결하면, 원격 머신(30)이 연결해제되었던 동안 수신한 세션 리셋 요청 각각에 대하여 새로운 세션을 생성할 것이다. 이러한 경우 일 실시예에서, 신규 세션은 재연결 및 재인가된 상태와 연관되어질 것이다. 이러한 경우 다른 실시예에서, 신규 세션에 대해서 어떠한 새로운 인가 사항은 없을 것이다. 이러한 경우 다른 실시예에서, 로컬 머신(10)이 애플리케이션을 실행하면, 새로운 라이선스를 획득할 것이며, 로컬 머신(10)과 연관된 모든 세션은 바로 그(active) 인가된 상태와 연관될 것이다.
- <384> 일 실시예에서, 로컬 머신(10) 상의 애플리케이션 스트리밍 클라이언트(552)는 하트비트 메시지를 생성한다. 이러한 경우 일 실시예에서, 애플리케이션 스트리밍 클라이언트(552)는 웹 인터페이스(558)로 하트비트 메시지를 전달하여 로컬 머신(10)으로 전달되고, 이어서 원격 머신(30)으로 전달된다. 다른 실시예에서, 원격 머신(30)상의 관리 서비스(504)는 로컬 머신(10)으로부터 웹 인터페이스(558)를 통하여 하트비트 메시지를 수신한다. 또 다른 실시예에서, 콜렉터 포인트(collector point; 240)(도 1d와 관련하여 상술함)를 포함하는 원격 머신(30)이 하트비트 메시지를 수신하여 저장한다.
- <385> 일 실시예에서, 애플리케이션 스트리밍 클라이언트(552)는 원격 머신(30)으로부터 라이선스를 요청한다. 이러한 경우 일 실시예에서, 라이선스는 로컬 머신(552) 상에서의 애플리케이션 프로그램의 실행을 허가한다. 이러한 경우 다른 실시예에서, 원격 머신(30)은 라이선스를 제공하기 위하여 제2 원격 머신을 액세스할 수도 있다. 이러한 경우 다른 실시예에서, 원격 머신(30)은 로컬 머신으로 라이선스를 제공할 수도 있다. 이러한 경우 다른 실시예에서, 원격 머신(30)은 허가에 수락될만한 라이선스를 제2 원격 머신으로 제공할 수도 있다. 일 실시예에서, 라이선스는 애플리케이션의 프로그램의 실행 종료시에 취소된다(revoked).
- <386> 일 실시예에서, 팜(38)의 원격 머신(30)은 동작에 라이선스를 요구하는 서브시스템들에 대한 라이선스를 구성 및 유지하고 그러한 서브시스템으로의 다수의 연결을 제어하기 위한 라이선스 관리 서브시스템을 포함한다. 또 다른 실시예에서, 원격 머신(30)은 애플리케이션 관리 서브시스템 및 세션 관리 서브시스템과 같은 다른 서브시스템들내에 라이선스 관리 서브시스템의 기능을 포함시킨다. 일 실시예에서, 각 원격 머신(30)은 라이선스 관리 서브시스템 또는 라이선스 관리 서브시스템과 관련된 기능을 포함한다. 라이선스 관리 서브시스템은 2개 유형의 라이선스: (1) 피처 라이선스(feature license) 및 (2) 연결 라이선스를 관리한다. 간단히 살펴보면, 라이선스 관리 서브시스템은 부하 관리와 같은 인가된 소프트웨어 제품의 "피처들"에 대한 액세스를 제공하기 위한 피처 라이선스와, 그 인가된 소프트웨어 제품에 의해 허용되는 다수의 사용자 연결을 제어하기 위한 연결 라이선스를 이용한다. 피처는 소프트웨어 제품의 어떤 측면 또는 특정 기능일 수도 있고, 또는 피처 라이선스 없이는 동작하지 않을 전체 제품일 수도 있다.
- <387> 도 15는 원격 머신(30)이 이벤트 버스(1570)와 통신하는 라이선스 관리 서브시스템(1510), 그룹 서브시스템(1520), 영구 저장 시스템 서비스 모듈(1570), 동적 저장 시스템 서비스 모듈(1580), 관계 서브시스템(1530), 전문화된 원격 머신 서브시스템(1540) 및 공통의 액세스 포인트 서브시스템(524)을 포함한다. 도 15에 도시된 이러한 서브시스템들은 라이선스 관리 서브시스템(1510)은 라이선스 관리 서브시스템(1510)의 동작을 설명하기 위한 것이다. 원격 머신(30)은 다른 유형의 서브시스템들을 포함할 수 있다.

- <388> 라이선스 관리 서브시스템(1510)은 라이선스 풀(license pools), 할당 및 그룹을 용이하게 하기 위하여 라이선스의 논리적 그룹(이하, "라이선스 그룹")을 형성하고 유지하기 위하여 이벤트 버스를 통해 그룹 서브시스템(1520)과 통신한다. 라이선스 그룹은 라이선스 스트림의 집합(아래에 기술) 및/또는 다른 라이선스 그룹들을 포함한다. 라이선스 그룹은 유사한 피처들이 수집하고 결과적으로 라이선스의 풀화(pooling)를 가능하게 한다. 풀화된 라이선스는 팜(38)의 임의의 원격 머신(30)에 의해서 이용될 수 있는 라이선스이다. 각 라이선스 그룹은 라이선스 그룹 및 다른 라이선스 서브그룹(즉, 라이선스 그룹내의 다른 라이선스 그룹들)의 라이선스들의 총체적인 기능들을 유지한다. 라이선스 풀에 관련된 정보는 일 실시예에서 동적 저장 장치(240)에 유지된다. 이러한 실시예에서, 각 라이선스 관리 서브시스템(1610)은 전체 개수의 라이선스들과, 팜(38)의 원격 머신(30)에 할당되는 개수의 라이선스들을 논리적으로 저장한다. 풀화된 라이선스의 부여시, 라이선스를 제공하는 관리 서브시스템(1510)은 풀화된 라이선스가 "사용중"임을 나타내는 엔트리를 동적 저장 장치(240)에 형성한다. 모든 다른 라이선스 관리 서브시스템(1510)은 이러한 풀화된 라이선스가 부여에 이용될 수 없다고 인식한다. 일 특정 실시예에서, 동적 저장 장치(240)는 사용 중인 풀화된 라이선스를 식별하기 위하여 각 라이선스 그룹과 연관된 머신 ID/클라이언트 ID 쌍을 저장한다.
- <389> 관계 서브시스템(1530)은 라이선스들과 원격 머신들(30)간의 그리고 라이선스 그룹들과 원격 머신들(30)간의 연관관계(association)를 유지한다. 이 연관관계들은 연관된 원격 머신(30)만이 얻을 수 있는 각각의 라이선스 및 라이선스 그룹(즉, "로컬 라이선스")에 대한 라이선스 개수를 정한다. 로컬 라이선스는 팜(38)의 하나의 원격 머신에 할당되는 라이선스이며 다른 원격 머신(38)에 의해서는 공유되지 않는다. 라이선스 관리 서브시스템(1510)은 관리 서브시스템(1530)과 통신하여 이러한 연관관계들을 생성하고, 질의(query)하며 업데이트한다. 통신 액세스 포인트 서브시스템(524)은 원격 머신(30)상에 상주하는 소프트웨어 제품에 의해 이용될 원격 프로시저 호출(RPC)을 제공한다. 이러한 RPC 인터페이스는 이러한 소프트웨어 제품이 공통의 액세스 서브시스템(524)을 통해 통신하여 라이선스 정보(licensing information)를 액세스할 수 있게 한다.
- <390> 도 15를 계속 참조하면, 전문화된 원격 머신 서브시스템(1540)은 라이선스 관리 서브시스템(1510)과 통신하여 라이선스를 필요로하는 전문화된 원격 머신 서브시스템(1540)의 각 기능에 한 피처 라이선스를 얻는다. 이는 전문화된 원격 머신 서브시스템(1540)의 초기화 및 임의의 라이선스 이벤트 이후에 발생한다. 피처 라이선스를 얻는 것이 불가능한 경우, 전문화된 원격 머신 서브시스템(1540)은 그 서브시스템이 라이선스를 제공하는 기능을 제한한다. 또한, 전문화된 원격 머신 서브시스템(1540)은 클라이언트 세션이 원격 머신(30)으로 개시될 때마다 라이선스 관리 서브시스템(1510)을 이용하여 클라이언트 연결 라이선스를 얻는다.
- <391> 라이선스 관리 서브시스템(1510)은 영구적 저장 시스템 서비스 모듈(352)과 통신하여 피처 및 연결 라이선스들을 명명 규칙(naming convention)에 따라 형성된 라이선스 스트링으로서 라이선스 저장소(license repository; 1550)에 저장한다. 라이선스 저장소(1550)는 영구적인 저장 장치(230)에 있다. CRC(cyclical redundancy checks)는 이러한 라이선스들이 라이선스 저장소(1550)에 저장되는 동안 라이선스의 탬퍼링(tampering)을 방지한다. 라이선스 관리 서브시스템(1510)은 라이선스 스트링들에 관련된 정보 또한 라이선스 저장소(1550)에 저장한다. 예를 들어, 정보는 어느 라이선스가 팜(38)이 어느 머신(30)에 할당되는지, 그리고 일 실시예에서, 각 라이선스의 활성화 상태를 나타낼 수도 있다. 일 실시예에서, 연결 라이선스 테이블(1560)은 연결 라이선스를 얻은 그 로컬 머신들의 식별자들을 저장한다.
- <392> 일 실시예에서, 라이선스 관리 서브시스템(1510)은 서브시스템들로부터의 인가된 기능의 이용 요청 이벤트, 예를 들어, 이용 가능한 풀화된 라이선스에 대한 요청을 지원한다. 이벤트는 라이선스를 요청하는 서브시스템의 UID 및 그 서브시스템상에 상주하는 원격 머신(30)의 UID를 포함한다. 이벤트는 또한 라이선스 그룹 ID의 형태로 요청되는 라이선스 유형(즉, 피처 또는 연결 라이선스)을 포함한다. 영구적 저장 장치(230)에 저장되는 실제의 라이선스 그룹 ID는 임의적(arbitrary)이지만, 명명 규칙에 따르면 원격 머신(30)에 신규 소프트웨어 제품(즉, 서브시스템)이 추후 추가되는 경우 융통성이 있게 된다.
- <393> 라이선스를 구하는 요청 서브시스템에 의해 전송되는 이벤트는 라이선스 그룹 유형의 표시와, 라이선스를 요청하는 로컬 머신 및 원격 머신의 ID와, "포스 획득(force acquire)" 플래그를 포함한다. 라이선스 그룹 유형의 표시는 부하 관리와 같은 피처 라이선스 또는 소프트웨어 애플리케이션 제품과 같은 연결 유형 라이선스의 표시를 포함한다. 라이선스를 구하는 로컬 머신 및 원격 머신을 식별하는 필드는 원격 머신 및 로컬 머신과 연관된 고유의 식별자를 포함할 수도 있다. 포스 획득 플래그는 예를 들어, 라이선스 변화 이벤트 이후 연결 라이선스를 재획득하는 데에 이용될 수 있다. 라이선스 변경 이벤트는 영구적 저장 장치(230)내 라이선스 정보가 변경, 예를 들어, 라이선스가 삭제, 추가 또는 할당되었음을 나타낸다. 라이선스 변경 이벤트시, 각 원격 머신(30)은 라이선스 변경 이벤트의 특정한 원인이 그 원격 머신에 공지되지 않은 것이므로 라이선스 변경 이벤트 이전에

소유한 모든 연결 라이선스들을 재획득하려 한다. 이러한 플래그가 설정되면, 그렇게 하여 원격 머신(30)으로의 연결 수를 미리 결정된 허용 가능한 연결의 최대 수의 초과하여 증가시키더라도 연결 라이선스가 획득되어야 함을 나타낸다. 그 후에는, 사용중인 연결 라이선스의 개수가 이러한 미리 정해진 최대 수 미만으로 떨어질 때까지 어떠한 신규의 연결 라이선스도 부여되지 않는다. 이러한 방식으로 로컬 머신 연결은 라이선스 변경 이벤트로 인해 세션 중(mid-session)에는 종료되지 않을 것이다.

<394> 이제 도 16을 참조하면, 블록도는 라이선스 집행(licensing enforcement)에 포함되는 컴포넌트들의 일 실시예를 도시한다. 원격 머신(30)은 서버 관리 서브시스템(508) 및 라이선스 관리 서브시스템(512)을 포함한다. 일 실시예에서, 서버 관리 서브시스템(508) 및 라이선스 관리 서브시스템(512)은 상술한 라이선스 관리 서브시스템(1510)의 기능을 제공한다. 다른 실시예에서, 애플리케이션 관리 서브시스템(506) 및 세션 관리 서브시스템(510)은 상술한 라이선스 관리 서브시스템(1510)의 기능을 제공한다. 또 다른 실시예에서, 다른 서브시스템은 상술한 라이선스 관리 서브시스템(1510)의 기능을 제공한다.

<395> 일 실시예에서, 서버 관리 서브시스템(508)은 라이선스의 발행 및 취소를 요청하는 데에 이용되는 라이선싱 컴포넌트를 포함할 수도 있다. 또 다른 실시예에서, 라이선스 관리 서브시스템(512)은 서버 관리 서브시스템(508)으로부터 수신한 라이선스의 발행 또는 취소 요청에 대하여 정책을 적용할 수도 있다. 또 다른 실시예에서, 라이선스 관리 서브시스템(512)은 요청을 라이선스 집행 기능을 제공하는 원격 머신(30)으로 전달할 수도 있다. 일 실시예에서, 관리 서비스(504)는 라이선스 집행 기능을 제공하는 제2 원격 머신(30)과 연결을 유지할 수도 있다. 다른 실시예에서, 원격 머신(30)은 라이선스 집행 기능을 제공한다.

<396> 일 실시예에서, 원격 머신으로 소정 개수의 하트비트 메시지를 전달하기 위하여 로컬 머신(10)의 고장시에는 라이선스가 만료되고 유효화가 중단된다. 이러한 경우 일 실시예에서, 라이선스의 만료에 의하여, 로컬 머신(10)에 의한 애플리케이션 프로그램의 실행의 허가가 취소된다.

<397> 다른 실시예에서, 소정의 시간의 경과시 세션이 종료된다. 일 실시예에서, 관리 서비스(504)는 라이선스의 만료 후 세션의 종료시까지 세션과 관련된 데이터를 유지한다. 일 실시예에서, 세션과 관련된 데이터는 세션명, 세션 ID, 클라이언트 ID, 클라이언트명, 세션 시작 시각, 서버명(파일 서버의 UNC 경로), 애플리케이션명(브라우저명을 기초로 하여 로컬 머신에 의하여 생성된 고유명), 앨리어스 명(alias name), 세션 상태(활성/인가, 활성/비인가, 재연결됨/비인가)와 같은 정보를 포함할 수도 있다. 또 다른 실시예에서, 로컬 머신(10)은 하트비트 메시지의 전송을 중단하고 추후에 하트비트 메시지의 전송을 재시작한다. 또 다른 실시예에서, 로컬 머신(10)이 세션의 만료 이전에 하트비트 메시지의 전송을 재시작하면, 관리 서비스(504)는 라이선스를 재발행하고 유지한 세션 관련된 데이터를 로컬 머신(10)에 이용 가능하게 만들 수도 있다.

<398> 이제 도 17을 참조하면, 흐름선도는 로컬 머신(10)상에서 세션의 존속(duration) 동안 원격 머신(30)으로부터 라이선스를 요청하고 유지하기 위하여 취하는 단계들의 일 실시예를 도시한다. 간단히 살펴보면, 애플리케이션 스트리밍 클라이언트는 라이선스를 요청한다(단계 1702). 원격 머신(30)은 라이선스에 대한 요청을 수신하고 그 요청과 관련된 티켓을 검증하여, 라이선스를 생성한다(단계 1704). 원격 머신(30)은 라이선스와, 라이선스와 관련된 정보를 로컬 머신(10)으로 제공한다(단계 1706). 로컬 머신(10)은 도 7의 단계 214와 관련하여 위에서 설명한 바와 같이 애플리케이션을 실행시킨다. 로컬 머신은 그 로컬 머신이 애플리케이션을 실행했음을 나타내는 하트비트 메시지를 전달한다(단계 1708). 원격 머신(30)은 하트비트 메시지를 수신하고 하트비트 메시지와 함께 전달되는 식별 정보를 검증한다(단계 1708). 원격 머신(30)은 실행된 애플리케이션 및 로컬 머신(10)과 연관된 세션을 생성한다(단계 1710). 세션을 생성한 결과는 로컬 머신(10)으로 전달된다(단계 1712). 로컬 머신은 도 7의 단계 216과 관련하여 위에서 설명한 바와 같이 애플리케이션의 실행 전반에 걸쳐 하트비트 메시지를 전달한다. 로컬 머신은 전달된 하트비트 메시지에 대한 응답(response)을 수신한다(단계 1714). 로컬 머신은 애플리케이션이 실행 종료를 나타내는 하트비트 메시지를 전달한다(단계 1716). 원격 머신(30)은 하트비트 메시지를 수신하고 세션 관련 데이터를 제거할 지 그리고 로컬 머신(10) 및 종료된 애플리케이션과 연관된 라이선스를 해제(release)할 지를 결정한다. 원격 머신(30)에 의해서 이루어진 결정의 결과는 로컬 머신(10)으로 전달된다(단계 1720).

<399> 도 17을 참조하면, 보다 상세히, 로컬 머신(10) 상에서 애플리케이션 스트리밍 클라이언트가 라이선스를 요청한다(단계 1702). 일 실시예에서, 로컬 머신(10)은 애플리케이션 프로그램과 연관된 액세스 정보의 수신시 라이선스를 요청한다. 이러한 경우 일 실시예에서, 로컬 머신은 로컬 머신(10)에 의해서 애플리케이션 프로그램을 실행시키는 것에 대하여 허가를 하는 라이선스를 원격 머신(30)으로부터 요청한다. 일 실시예에서, 라이선스에 대한 요청은 원격 머신(30)으로부터 수신한 액세스 정보 및 개시 티켓을 포함한다. 다른 실시예에서, 로컬 머

신(10)상의 애플리케이션 스트리밍 클라이언트(552)는 이 요청을 웹 인터페이스(558)로 전달하고 웹 인터페이스(558)는 원격 머신(30)으로 이 요청을 전달한다. 또 다른 실시예에서, 원격 머신 상의 세션 관리 서브시스템(510)이 이 라이선스 요청을 수신하여 소유한다.

<400> 원격 머신(30)은 라이선스 요청을 수신하고, 검증하고, 이 요청과 관련된 티켓을 검증하고, 라이선스를 생성한다(단계 1704). 일 실시예에서, 원격 머신(30)은 로컬 머신(10)이 애플리케이션을 실행하도록 허가되는 지를 검증한다. 또 다른 실시예에서, 원격 머신(30)은 로컬 머신(10)이 기존 라이선스와 이미 연관되어 있는 지를 판단한다. 또 다른 실시예에서, 원격 머신(30)은 로컬 머신(10)이 기존 라이선스와 연관되어 있다고 판단하고 기존 라이선스를 관리하는 세션 관리 서버(562)의 식별자를 로컬 머신(10)에게 제공한다. 또 다른 실시예에서, 원격 머신(30)은 신규 라이선스, 세션 식별자 및 그 신규 라이선스를 관리하는 세션 관리 서버(562)의 ID를 생성하여 로컬 머신(10)에게 제공한다.

<401> 일 실시예에서, 원격 머신(30)은 라이선스 요청에 대하여 응답하기 위하여 라이선스 관리 서브시스템(1510)을 이용한다. 라이선스 관리 서브시스템(1510)이 라이선스 요청을 수신한다. 이 요청은 피처 라이선스 또는 연결 라이선스 요청일 수 있다. 라이선스 관리 서브시스템(1510)은 라이선스가 이미 부여되었는 지, 즉, 피처가 이미 시작되었거나 또는 로컬 머신에 대한 연결이 이미 존재하는 지를 판단한다. 라이선스가 이미 부여되어 있으면, 라이선스 관리 서브시스템(1510)은 라이선스 요청자(license requestor)에게 "부여" 이벤트를 전송한다. 그 라이선스가 이전에 부여되지 않았으면, 라이선스 관리 서브시스템(1510)은 로컬 라이선스, 즉, 원격 머신(30)에 영구적으로 할당된 라이선스가 이용 가능한 지를 판단한다. 일 실시예에서, 라이선스 관리 서브시스템(1510)은 로컬 메모리를 검사함으로써 이러한 판단을 한다. 로컬 라이선스가 이용 가능, 즉, 원격 머신(30)이 현재 부여된 것보다 영구적으로 할당된 라이선스를 더 많이 가지면, 라이선스 관리 서브시스템(1510)은 라이선스 요청자에게 "부여" 이벤트를 전송한다.

<402> 원격 머신(30)은 라이선스와, 그 라이선스와 관련된 정보를 로컬 머신(10)으로 제공한다(단계 1706). 일 실시예에서, 라이선스, 세션 식별자 및 세션 관리 서버(562)의 ID를 원격 머신(30)으로부터 수신하면, 로컬 머신(10)이 애플리케이션을 실행시킨다. 로컬 머신(10)은 도 7의 단계 214와 관련하여 위에서 설명한 바와 같이 애플리케이션을 실행시킬 수도 있다. 로컬 머신은 그 로컬 머신이 애플리케이션을 실행했다는 것을 나타내는 하트비트 메시지를 전달한다(단계 1708). 일 실시예에서, 로컬 머신은 원격 머신(30)으로 하트비트 메시지를 전달하고, 원격 머신(30)이 그 하트비트 메시지를 세션 관리 서버(562)로 전달한다. 또 다른 실시예에서, 로컬 머신(10)은 원격 머신(30)으로부터 수신한 세션 관리 서버(562)의 식별자에 따라서 하트비트 메시지를 직접 세션 관리 서버(562)로 전달한다.

<403> 원격 머신(30)은 하트비트 메시지를 수신하고 이 하트비트 메시지와 함께 전달되는 식별 정보를 검증한다(단계 1708). 일 실시예에서, 원격 머신(30)은 세션 관리 서버(562)이다. 또 다른 실시예에서, 세션 관리 서버(562)는 로컬 머신(10)에 의해서 하트비트 메시지와 함께 제공되는 서버 식별자를 검증한다. 또 다른 실시예에서, 서버 식별자는 원격 머신(30)에 의해서 로컬 머신(10)에 제공되는 식별자이다.

<404> 원격 머신(30)은 실행되는 애플리케이션 및 로컬 머신(10)과 연관되는 세션을 생성한다(단계 1710). 일 실시예에서, 세션 관리 서버(562)는 하트비트 메시지 수신시 실행 중인 애플리케이션과 연관된 신규의 세션을 생성한다. 또 다른 실시예에서, 제3 원격 머신(30)이 신규의 세션을 생성한다. 일 실시예에서, 세션 관리 서버(562)는 신규의 세션 생성시 세션 관련 정보를 저장한다.

<405> 세션 생성 결과가 로컬 머신(10)으로 전송된다(단계 1712). 일 실시예에서, 이 결과는 세션의 생성을 확인한다. 다른 실시예에서, 이 결과는 세션과 연관된 일 또는 복수의 애플리케이션을 식별한다. 로컬 머신은 도 7의 단계 216과 관련하여 상술한 바와 같이 애플리케이션 실행 전반에 걸쳐 하트비트 메시지를 전달한다. 일 실시예에서, 로컬 머신(10)은 애플리케이션 프로그램의 실행 전반에 걸쳐 주기적으로 규칙적인 간격으로 하트비트 메시지를 세션 관리 서버(562)로 계속 전달한다. 로컬 머신은 전달된 하트비트 메시지에 대한 응답을 수신한다(단계 1714). 일 실시예에서, 로컬 머신(10)은 세션 관리 서버(562)로부터 하트비트 메시지를 수신했다는 확인을 수신한다. 또 다른 실시예에서, 로컬 머신(10)은 세션 관리 서버(562)의 하트비트 메시지 수신에 응답하여 세션 관리 서버(562)로부터 실행 명령을 수신한다.

<406> 로컬 머신은 애플리케이션의 실행의 종료를 나타내는 하트비트 메시지를 전달한다(단계 1715). 원격 머신(30)은 하트비트 메시지를 수신하고 세션 관련 데이터를 제거할 지 그리고 로컬 머신(10) 및 종료된 애플리케이션과 연관된 라이선스를 해제할 지를 판단한다(단계 1718). 원격 머신(30)에 의해 이루어지는 판단의 결과는 로컬 머신(10)으로 전달된다(단계 1720).

- <407> 이제 도 18을 참조하면, 블록도는 관리 서비스(504)에 의해 모니터링되는 세션에 연관될 수 있는 상태들의 일 실시예를 도시한다. 일 실시예에서, 관리 서비스(504) 상의 세션 관리 서브시스템(510)은 로컬 머신(10)의 세션을 모니터링하고 그 세션에 상태를 지정(assign)한다. 또 다른 실시예에서, 세션 관리 서브시스템(510)은 라이선스 관련 데이터의 리스트 — 로컬 머신과 연관된 식별자, 세션과 연관된 식별자, 세션 상태 및 원격 머신(30)이 로컬 머신(10)으로부터 메시지를 수신한 최종 시각을 나타내는 타임스탬프를 포함할 수 있음 — 를 유지한다. 일 실시예에서, 세션 관리 서브시스템(510)은 세션 모니터링 스레드를 포함한다. 이러한 경우 일 실시예에서, 세션 모니터링 스레드는 라이선스 관련 데이터의 리스트를 스캔하고 세션의 세션 상태를 업데이트하기 위하여 주기적인 라이선스 타임아웃 간격으로 깨어난다(awaken).
- <408> 세션의 상태일 수 있는 제1 상태는 활성 그리고 인가 상태이다. 일 실시예에서, 이러한 상태인 경우, 로컬 머신(10)은 애플리케이션의 실행을 허가하는 유효 라이선스를 유지했다. 또 다른 실시예에서, 세션 관리 서버(562)는 세션 관련 데이터를 유지한다. 일 실시예에서, 세션 관리 서버(562)는 제2 원격 머신상에 세션 관련 데이터를 저장한다. 일 실시예에서, 로컬 머신(10)이 애플리케이션을 최초로 실행시키는 경우, 로컬 머신의 세션은 활성 그리고 인가 상태이다.
- <409> 세션의 상태일 수 있는 제2 상태는 활성 그리고 비인가 상태이다. 일 실시예에서, 로컬 머신(10)이 하트비트 메시지를 전달하는 데에 실패하고 로컬 머신(10)으로의 라이선스가 만료된 경우에 세션은 이 상태이다. 또 다른 실시예에서, 라이선스는 만료되었으나 세션이 만료되기까지는 시간이 남아 세션이 활성 상태로 간주되는 경우에 세션은 활성 그리고 비인가 상태이다. 일 실시예에서, 세션이 활성 그리고 비인가 상태인 동안, 원격 머신(30) 또는 세션 관리 서버(562)가 로컬 머신(10)을 대신하여 세션 관련 데이터를 저장한다. 일 실시예에서, 로컬 머신(10)이 세션의 만료 이전에 하트비트 메시지를 전달하면, 세션 관련 데이터는 신규의 라이선스와 함께 로컬 머신(10)으로 전달되고, 그 세션은 활성 및 인가 상태로 되돌아간다. 일 실시예에서, 원격 머신(30)은 그 세션이 만료되지 않았음을 검증하고 로컬 머신에 적합한 세션 관련 데이터를 제공하기 위해 로컬 머신과 연관된 식별자와 세션 식별자를 이용한다.
- <410> 세션의 상태일 수 있는 제3 상태는 연결 해제 및 부재 상태일 수 있다. 세션이 만료한 경우, 세션 관련 데이터는 삭제된다.
- <411> 세션의 상태일 수 있는 제4 상태는 재연결 및 비인가 상태일 수 있다. 일 실시예에서, 로컬 머신(10)상의 세션이 만료한 경우, 세션 관련 데이터는 삭제된다. 또 다른 실시예에서, 로컬 머신(10)이 새로운 하트비트 메시지를 전달하면, 로컬 머신(10)을 위하여 신규의 로컬 머신 식별자 및 세션 식별자가 생성된다. 일 실시예에서, 로컬 머신(10)이 원격 머신에 대해 재허가되고, 신규 라이선스를 수신하고 활성 및 인가 상태로 들어간다.
- <412> 표 3은 세션과 연관될 수 있는 상태들을 요약하였다.

표 3

세션 상태	설명
활성 \ 인가	정상 동작 모드
활성 \ 비인가	하트비트 메시지 손실 기간>라이선스 타임아웃이며 하트비트 메시지 손실 기간<세션 타임아웃
재연결 \ 비인가	하트비트 메시지 손실 기간>세션 타임아웃 또는 그 세션의 CPS/RADE 호스팅이 다운되고 다시 온라인으로 복귀

- <414> 일 실시예에서, 패키징 메커니즘은 애플리케이션 프로그램과 연관된 복수의 애플리케이션 파일의 생성을 가능하게 한다. 이러한 경우 일 실시예에서, 패키징 메커니즘은 복수의 애플리케이션 파일들의 식별을 가능하게 한다. 이러한 경우 다른 실시예에서, 패키징 메커니즘은 개별 애플리케이션 파일들을 복수의 애플리케이션 파일로 그룹화하는 것을 가능하게 한다. 이러한 경우 다른 실시예에서, 패키징 메커니즘은 파일 서버 또는 애플리케이션 서버와 같은 원격 머신 상에서 복수의 애플리케이션 파일의 호스팅을 가능하게 한다.
- <415> 일 실시예에서, 패키징 메커니즘은 "스테이징 머신(staging machine)"으로 설명되는 원격 머신 상에서 실행한다. 또 다른 실시예에서, 패키징 메커니즘은 "클린 머신(clean machine)" 상에서 실행한다. 클린 머신

은 추가되는 소프트웨어, 드라이버, 레지스트리 개체 등의 파일들 없이 자체 상에 운영체제만을 갖는 원격 머신 일 수 있다. 또 다른 실시예에서, 패키징 머신은 원격 머신 상에서 실행하는데, 이 원격 머신은 자체 상에서 애플리케이션 프로그램을 실행시킬 수 있는 로컬 머신과 유사하다. 일 실시예에서, 패키징 메커니즘을 실행시키는 원격 머신은 애플리케이션이 설치될 수도 있는 클린 머신 환경을 제공하는 격리 환경을 포함하며, 여기서, 원격 머신은 클린 머신 자체는 아니다.

<416> 일 실시예에서, 복수의 애플리케이션 파일들은 "패키지"라 한다. 또 다른 실시예에서, 패키지는 복수의 애플리케이션 파일을 저장하는 보관 파일일 수도 있다. 또 다른 실시예에서, 패키지는 복수의 애플리케이션 파일들을 저장하는 보관 파일일 수도 있고, 복수의 애플리케이션 파일에 적어도 하나의 파일과 연관된 메타데이터를 포함하는 하나의 파일일 수도 있다. 일 실시예에서, 패키지는 애플리케이션 프로그램을 포함하는 복수의 애플리케이션 파일을 포함한다. 다른 실시예에서, 패키지는 한 조의 애플리케이션 프로그램을 포함하는 복수의 애플리케이션 파일을 포함한다. 또 다른 실시예에서, 패키지는 애플리케이션 프로그램 및 그 애플리케이션 프로그램의 실행에 필요한 필요조건을 포함하는 복수의 애플리케이션 파일들을 포함한다.

<417> 일 실시예에서, 패키징 메커니즘은 설치 프로그램의 실행을 격리 환경에서 시작한다. 또 다른 실시예에서, 패키징 메커니즘은 설치 프로그램에 의해 생성되는 격리 환경에의 변화를 모니터링한다. 또 다른 실시예에서, 패키징 메커니즘은 격리 환경에 설치 프로그램에 의한 파일의 생성을 모니터링한다. 또 다른 실시예에서, 패키징 메커니즘은 격리 환경에서 파일의 설치 프로그램에 의한 변경을 모니터링한다. 일 실시예에서, 복수의 애플리케이션 파일들은 설치 프로그램에 의해 생성되거나 변경되는 파일을 포함한다. 또 다른 실시예에서, 패키징 메커니즘은 격리 환경을 모니터링 하기 위하여 파일 시스템 필터 드라이버(564)를 구현한다.

<418> 일 실시예에서, 패키징 메커니즘은 복수의 애플리케이션 파일들을 생성하는데, 각각의 애플리케이션 파일은 서로 다른 타겟 환경에서 실행되도록 구성된 서로 다른 버전의 애플리케이션 프로그램을 포함한다. 이러한 경우 일 실시예에서, 복수의 애플리케이션 파일은 특정 운영 체제, 개정 레벨(revision level), 언어 구성 및 마스터 드라이브를 갖는 로컬 머신 상에서 실행하도록 구성된다(예를 들어, 일 복수의 애플리케이션 파일은 개정 레벨 SP2 이상을 갖는 Windows XP 프로페셔널 운영 체제를 가지고 마스터 드라이브 C:\를 갖는 로컬 머신 상에서 실행하도록 구성될 수 있다). 이러한 경우 다른 실시예에서, 일 이상의 복수의 애플리케이션 파일은 단일의 보관 파일에서 결합될 수도 있다. 이러한 경우 다른 실시예에서, 각각의 복수의 애플리케이션 파일을 "타겟"이라 할 수도 있다. 이러한 경우 다른 실시예에서, 하나 이상의 복수의 애플리케이션 파일을 포함하는 보관 파일을 "패키지"라 할 수도 있다.

<419> 이제 도 19를 참조하면, 블록도는 두 개의 타겟을 포함하는 패키지를 도시하는데, 각각의 타겟은 애플리케이션을 포함하는 복수의 애플리케이션 파일들을 포함한다. 도 19에서, 애플리케이션 프로그램 'Foo'는 두 개의 타겟으로 패키징된다. 두 개의 타겟들간의 차이는 '타겟 언어'이다. 특히, 타겟1은 '영어'를 지원하고 타겟2는 '독어'를 지원한다. 일 실시예에서, 이용 가능한 애플리케이션 프로그램의 목록은 애플리케이션 프로그램 'Foo'를 리스트할 수도 있다. 또 다른 실시예에서, 복수의 애플리케이션 파일은 애플리케이션 프로그램에 대해 액세스를 요청하는 로컬 머신으로 전달된다. 또 다른 실시예에서, 로컬 머신의 평가에 응답하여, 로컬 머신으로 특정 타겟을 전달하는 판단이 이루어진다. 또 다른 실시예에서, 패키지와 연관된 파일은 그 패키지내의 타겟과 연관되고 로컬 머신상에서의 실행에 필요한 적어도 하나의 특징을 식별한다.

<420> 일 실시예에서, 패키징 메커니즘(530)은 애플리케이션 프로그램과 연관된 설치 프로그램을 실행시켜 애플리케이션 프로그램을 스트리밍할 준비를 한다. 이러한 경우 일 실시예에서, 패키징 메커니즘은 패키징 메커니즘이 실행되는 원격 머신(30)상의 격리 환경을 생성한다. 이러한 경우 다른 실시예에서, 패키징 메커니즘은 격리 환경에서 애플리케이션 프로그램을 실행시킨다. 이러한 경우 다른 실시예에서, 패키징 메커니즘은 설치 프로그램에 의해서 생성되거나 수정되는 복수의 애플리케이션 파일을 식별한다. 이러한 경우 다른 실시예에서, 패키징 메커니즘은 복수의 애플리케이션 파일을 포함하는 보관 파일을 생성한다. 이러한 경우 일 실시예에서, 패키징 메커니즘은 복수의 애플리케이션 파일을 포함하는 CAB 파일을 생성한다. 이러한 경우 일 실시예에서, 패키징 메커니즘은 디렉토리를 생성하고, 그 디렉토리에 복수의 애플리케이션 파일을 저장한다. 일 실시예에서, 패키징 메커니즘은 복수의 애플리케이션 파일을 일 파일 서버 또는 다른 원격 머신(30)상에 저장한다. 다른 실시예에서, 패키징 메커니즘은 복수의 애플리케이션 파일을 복수의 원격 머신들에 저장한다.

<421> 이제 도 20을 참조하면, 흐름선도는 운영체제를 재부팅하지 않고도 애플리케이션 프로그램을 효과적 설치하는 정책 기반 방법으로 행하는 단계들의 일 실시예를 도시한다. 간단히 살펴보면, 패키징 메커니즘은 격리 환경에서 설치자 프로그램을 실행하는데, 이 설치자 프로그램이 제2 애플리케이션과 연관된 적어도 하나의 애플리케이션

선 파일을 격리 환경내로 설치한다(단계 2002). 적어도 하나의 API(애플리케이션 프로그래밍 인터페이스)로의 호출 — 이 호출은 운영 체제의 재부팅 후에 동작 수행을 필요로 함 — 을 설치자 프로그램이 인터셉트한다(intercept). 운영체제의 재부팅없이 적어도 하나의 인터셉트된 호출의 동작을 실행한다(단계 2006). 적어도 하나의 애플리케이션 파일의 파일 유형 식별자를 수신한다(단계 2008). 식별된 파일 유형에 응하여, 설치된 적어도 하나의 애플리케이션 파일에 적어도 하나의 실행 방법을 연관시킨다(2010). 적어도 하나의 서버에 상기 설치된 적어도 하나의 애플리케이션 파일을 저장한다(단계 2012). 제2 애플리케이션, 상기 설치된 적어도 하나의 애플리케이션 파일, 적어도 하나의 서버 위치 및 적어도 하나의 실행 방법에 대한 목록을 생성한다(단계 2014).

<422> 이제 도 20을 참조하여 보다 상세히 설명하면, 패키징 메커니즘은 격리 환경내에서 설치자 프로그램을 실행시키는데, 이 설치자 프로그램은 제2 애플리케이션과 연관된 적어도 하나의 파일을 격리 환경 내로 설치한다(단계 2002). 일 실시예에서, 격리 환경 내에서 설치자 프로그램을 실행함으로써 패키징 메커니즘이 설치자 프로그램에 의한 로컬 머신상의 파일 또는 레지스트리에 대한 변화를 격리시킬 수 있게 한다. 또 다른 실시예에서, 패키징 메커니즘은 설치자 프로그램에 의해 요청되는 변화를 인터셉트하고 격리 환경에 대한 변화를 리디렉팅(redirect)하여 그 변화가 로컬 머신 상에서 발생하는 것을 방지한다. 또 다른 실시예에서, 패키징 메커니즘은 격리 환경에서 제2 설치자 프로그램을 실행시키고, 제2 애플리케이션은 제3 애플리케이션과 연관된 적어도 하나의 애플리케이션 파일을 격리 환경 내로 설치한다.

<423> 일 실시예에서, 패키징 메커니즘은 격리 환경내에서 설치자 프로그램을 실행시키고, 설치자 프로그램은 격리 환경 내부의 애플리케이션과 연관되어 있는 적어도 하나의 실행 가능한 애플리케이션을 실행한다. 설치자가 애플리케이션을 실행하는 일 실시예에서, 애플리케이션이 실행되면 제2 애플리케이션의 설치를 가능하게 한다.

<424> 이러한 경우 일 실시예에서, 애플리케이션의 설치는 설치자 프로그램을 실행시키는 것에 더하여 적어도 하나의 실행 가능한 애플리케이션의 실행을 필요로 한다. 이러한 경우 다른 실시예에서, 애플리케이션의 설치는 설치자 프로그램을 실행시키는 것에 더하여, 인터넷 브라우저 애플리케이션의 실행을 필요로 한다. 일 실시예에서, 프로그램을 실행하기 위해 설치자 프로그램이 실행되는데, 설치자 프로그램의 실행은 그 프로그램을 설치하는데 필요한 제2 프로그램의 실행을 포함한다. 이러한 경우 일 실시예에서, 프로그램은 플러그-인이다. 이러한 경우 일 또 다른 실시예에서, 프로그램은 액티브 X 컴포넌트이다. 이러한 경우 다른 실시예에서, 프로그램은 플래시 컴포넌트이다. 이러한 경우 다른 실시예에서, 프로그램은 Yahoo! 또는 Google 툴바와 같은 특수화된 툴바이다. 다른 실시예에서, 프로그램은 제2 프로그램내로 설치되지만 제2 프로그램과 독립적으로는 실행할 수 없는 컴포넌트이다.

<425> 설치자 프로그램에 의한 적어도 하나의 API(application programming interface)로의 호출 — 이 호출은 운영 체제의 재부팅 후에 동작을 실행시키는 것을 필요로 함 — 을 인터셉트한다(단계 2004). 인터셉트한 적어도 하나의 호출 동작은 운영체제의 재부팅없이 실행된다(단계 2006). 일 실시예에서, 동작의 실행은 설치 도중에 변경되는 레지스트리 엔트리의 동작을 실행시키는 것을 포함한다. 운영 체제의 재부팅없이 인터셉트한 적어도 하나의 호출을 실행하는 것에 대해서는 아래에서 도 25와 관련하여 더 설명한다.

<426> 적어도 하나의 애플리케이션 파일의 파일 유형의 식별자를 수신한다(단계 2008). 식별된 파일 유형에 따라 설치된 적어도 하나의 애플리케이션 파일을 적어도 하나의 실행 방법과 연관시킨다(단계 2010). 일 실시예에서, 적어도 하나의 실행 방법은 적어도 하나의 애플리케이션 파일을 클라이언트로 스트리밍시키는 것을 가능하게 한다. 또 다른 실시예에서, 적어도 하나의 실행 방법은 설치된 적어도 하나의 애플리케이션 파일을 클라이언트 상에서 실행시키는 것을 가능하게 한다. 또 다른 실시예에서, 적어도 하나의 실행 방법은 설치된 적어도 하나의 애플리케이션 파일을 서버 상에서 실행시키는 것을 가능하게 한다. 또 다른 실시예에서, 적어도 하나의 실행 방법은 적어도 하나의 애플리케이션 파일을 서버로 스트리밍하는 것을 가능하게 한다.

<427> 설치된 적어도 하나의 애플리케이션 파일을 적어도 하나의 서버 상에 저장한다(단계 2012). 일 실시예에서, 설치된 적어도 하나의 애플리케이션 파일을 적어도 하나의 서버 상에 저장하기 이전에 설치된 애플리케이션 프로그램을 격리 환경내에서 실행한다. 이러한 경우 일 실시예에서, 설치된 애플리케이션 프로그램의 실행에 따라 추가의 애플리케이션 파일이 생성된다. 이러한 경우 일 실시예에서, 데이터 파일이 생성된다. 이러한 경우 다른 실시예에서, 설치된 애플리케이션 프로그램은 설치를 완료하기 위한 정보를 필요로 하는데, 초기 설치 프로세스 이후에는 이 정보가 필요하다. 이러한 경우 다른 실시예에서, 소프트웨어 제품 식별자, 라이선스 식별자 또는 다른 증명서(credential)와 같은 정보를 필요로 한다.

<428> 일 실시예에서, 적어도 하나의 서버 상에 설치된 적어도 하나의 애플리케이션 파일의 위치를 식별하는 식별자가

제공된다. 이러한 경우 일 실시예에서, 식별자는 UNC(Universal Naming Convention)를 따른다(conform). 다른 실시예에서, 설치된 적어도 하나의 애플리케이션 파일은 CAB 파일과 같은 보관 파일내에 위치한다. 이러한 경우 일 실시예에서, 복수의 애플리케이션 파일은 보관 파일 내에 저장되고, 보관 파일은 적어도 하나의 서버 상에 저장된다. 이러한 경우 다른 실시예에서, 설치된 적어도 하나의 애플리케이션 파일은 복수의 서버 상에 저장된다. 또 다른 실시예에서, 적어도 하나의 애플리케이션 파일은 애플리케이션 파일들을 저장하는 디렉토리 내에 위치한다.

<429> 제2 애플리케이션, 상기 설치된 적어도 하나의 애플리케이션 파일, 적어도 하나의 서버의 위치 및 적어도 하나의 실행 방법의 목록을 생성한다(단계 2014). 일 실시예에서, 목록은 파일에 저장된다. 다른 실시예에서, 목록은 일목 요연한(manifest) 파일로 저장된다. 또 다른 실시예에서, 목록은 XML 파일로 저장된다.

<430> 일 실시예에서, 복수의 애플리케이션, 복수의 애플리케이션 각각과 연관되어 있는 복수의 설치된 애플리케이션 파일 및 상기 복수의 설치된 애플리케이션 파일을 저장하는 적어도 하나의 서버의 목록이 생성된다. 또 다른 실시예에서, 제2 애플리케이션과 복수의 설치된 애플리케이션 파일들간의 연관관계를 포함하는 목록이 생성된다. 또 다른 실시예에서, 제2 애플리케이션과, 적어도 하나의 설치된 애플리케이션 파일을 포함하는 압축 파일과의 연관관계를 포함하는 목록이 생성된다.

<431> 이제 도 21을 참조하면, 운영체제를 재부팅하지 않고도 애플리케이션 프로그램을 실행시키는 정책 기반 방법으로 행하는 단계들의 일 실시예를 도시한다. 간단히 살펴보면, 패키징 메커니즘은 격리 환경 내에서 설치자 프로그램을 실행시키는데, 이 설치자 프로그램은 제2 애플리케이션과 연관된 적어도 하나의 애플리케이션 파일을 격리 환경 내로 설치한다(단계 2102). 설치자 프로그램에 의한 적어도 하나의 API(application programming interface)로의 호출 - 이 호출은 운영체제의 재부팅 이후에 동작의 실행을 필요로 함 - 을 인터셉트한다. 인터셉트한 적어도 하나의 호출의 동작을 운영체제의 재부팅 없이 실행한다(단계 2106). 적어도 하나의 애플리케이션 파일의 특성의 식별자를 수신한다(단계 2108). 식별된 특성에 따라서 적어도 하나의 실행 필요요건을 적어도 하나의 설치된 애플리케이션 파일과 연관시킨다(단계 2110). 적어도 하나의 설치된 애플리케이션 파일은 적어도 하나의 서버 상에 저장된다(단계 2112). 제2 애플리케이션, 적어도 하나의 설치된 애플리케이션 파일, 적어도 하나의 서버의 위치 및 적어도 하나의 실행 필요요건의 목록을 생성한다(단계 2114).

<432> 이제 도 21을 참조하면 보다 상세히, 패키징 메커니즘이 설치자 프로그램을 격리 환경내에서 실행시키는데, 이 설치자 프로그램은 제2 애플리케이션과 연관된 적어도 하나의 애플리케이션 파일을 격리 환경내로 설치한다(단계 2102). 일 실시예에서, 격리 환경내에 설치자 프로그램을 실행시킴으로써, 패키징 메커니즘이, 설치자 프로그램에 의한 로컬 머신상의 파일 또는 레지스트리에 대한 변화를 격리시킬 수 있게 한다. 또 다른 실시예에서, 패키징 메커니즘은 설치자 프로그램이 요청하는 변화를 인터셉트하고 격리 환경으로 리디렉팅하여 그 변화가 로컬 머신 상에서 발생하는 것을 방지한다. 또 다른 실시예에서, 패키징 메커니즘은 격리 환경내에서 제2 설치자 프로그램을 실행하는데, 제2 애플리케이션은 제3 애플리케이션과 연관된 적어도 하나의 애플리케이션 파일을 격리 환경내로 설치한다.

<433> 일 실시예에서, 패키징 메커니즘은 격리 환경내에서 설치자 프로그램을 실행시키는데, 이 설치자 프로그램은 격리 환경내부의 애플리케이션과 연관되어 있는 적어도 하나의 실행 가능한 애플리케이션을 실행시킨다. 설치자가 애플리케이션을 실행하는 일 실시예에서, 애플리케이션의 실행은 제2 애플리케이션의 설치를 가능하게 한다. 이러한 경우 다른 실시예에서, 애플리케이션의 설치의 설치자 프로그램을 실행에 더하여, 적어도 하나의 실행 가능한 애플리케이션의 실행을 필요로 한다. 이러한 경우 다른 실시예에서, 애플리케이션의 설치의 설치자 프로그램의 실행에 더하여, 인터넷 브라우저 애플리케이션의 실행을 필요로 한다.

<434> 도 23을 참조하면, 블록도는 설치자 프로그램(2350)을 격리 환경(532)내에서 실행시키는 패키징 메커니즘(530)과, 이 패키징 메커니즘(530) 및 격리 환경(532)과 통신하는 파일 시스템 필터 드라이버(534)를 포함하는 시스템의 일 실시예를 도시한다.

<435> 일 실시예에서, 패키징 메커니즘(530)은 애플리케이션 프로그램을 격리 환경(532)내로 설치함으로써 패키지를 생성한다(도 21과 관련하여 위에서 설명). 또 다른 실시예에서, 패키징 메커니즘(530)은, 설치자 프로그램(2350)을 실행하여 격리 환경(532)내로 애플리케이션 프로그램을 설치한다. 일 실시예에서, 패키징 메커니즘(530)은 그래픽 사용자 인터페이스(GUI)를 포함한다. 이러한 경우 일 실시예에서, 그래픽 사용자 인터페이스는 패키징 메커니즘(530)의 사용자가 패키징 메커니즘(530)에 의한 패키지의 생성을 특수화할 수 있게 한다. 이러한 경우 다른 실시예에서, 패키징 메커니즘(530)은 액세스 제어 스위트(access control suite; 520)상에서 그래픽 사용자 인터페이스와 통신하여 상기 액세스 제어 스위트(520)의 사용자가 상기 패키징 메커니즘(530)에 의한 패키

지의 생성을 특수화할 수 있게 한다.

- <436> 일 실시예에서, 파일 시스템 필터 드라이버(532)는 격리 환경(532)에서 애플리케이션 프로그램을 설치할 수 있게 한다. 이러한 경우 일 실시예에서, 파일 시스템 필터 드라이버(532)는 설치자 프로그램(2350)에 의한 요청을 인터셉트한다. 이러한 경우 다른 실시예에서, 파일 시스템 필터 드라이버(532)는 설치자 프로그램(2350)에 의한 요청을 격리 환경(532)으로 리디렉팅한다. 이러한 경우 다른 실시예에서, 파일 시스템 필터 드라이버(532)는 설치자 프로그램(2350)에 의해 이루어지는 요청의 레코드를 저장한다. 이러한 경우 다른 실시예에서, 파일 시스템 필터 드라이버(532)는 설치자 프로그램(2350)에 의해 생성되거나 변경된 파일의 복사를 저장한다. 일 실시예에서, 파일 시스템 필터 드라이버(532)에 의해 생성되어 저장된 레코드는 애플리케이션 프로그램을 포함하는 복수의 애플리케이션 파일로서 함께 저장된다. 다른 실시예에서, 복수의 애플리케이션 파일은 파일 서버(540)에 저장된다.
- <437> 도 21을 다시 참조하면, 설치자 프로그램에 의한 적어도 하나의 API(application programming interface)로의 호출 - 이 호출은 운영체제의 재부팅 이후 동작의 실행을 필요로 함 - 을 인터셉트한다(단계 2104). 상기 인터셉트한 적어도 하나의 호출의 동작은 운영체제의 재부팅 없이 실행된다(단계 2106). 일 실시예에서, 동작의 실행은 운영체제의 부팅시 시작되도록 구성된 드라이버의 실행을 포함한다. 다른 실시예에서, 동작의 실행은 설치 도중에 변경된 레지스트리 엔트리의 동작을 실행시키는 것을 포함한다.
- <438> 적어도 하나의 애플리케이션 파일의 식별자를 수신한다(단계 2108). 일 실시예에서, 운영체제 유형의 식별자를 수신한다. 다른 실시예에서, 운영체제가 사용하는 언어의 식별자를 수신한다. 또 다른 실시예에서, 제2 애플리케이션의 서버의 식별자를 수신한다.
- <439> 식별된 특성에 따라서, 적어도 하나의 설치된 애플리케이션 파일과 적어도 하나의 실행 필요요건을 연관시킨다(단계 2110). 일 실시예에서, 상기 특성에 정책을 적용하는 것에 응하여, 적어도 하나의 설치된 애플리케이션 파일과 적어도 하나의 실행 필요요건을 연관시킨다. 또 다른 실시예에서, 적어도 하나의 설치된 애플리케이션 파일과 스크립트를 연관시키는데, 이 스크립트는 클라이언트 상에서 적어도 하나의 실행 필요요건의 존재를 판단하는 실행 가능 프로그램을 포함한다. 이제 도 22를 참조하면, 스크린 샷은 로컬 머신 상에서 실행되는 스크립트의 목록의 일 실시예를 도시한다. 스크립트의 유형(2202)은 그 스크립트가 실행되어야 하는 시기, 예를 들어, 애플리케이션의 실행 이전 및 애플리케이션 실행 종료 중 하나로 그 시기를 나타낸다. 격리 표시자(isolation indicator; 24)는 스크립트가 로컬 머신(10) 상의 격리 환경에서 실행되어야 하는 지를 나타낸다. 도 22에 도시하는 바와 같이, 일 실시예에서, 스크립트는 복수의 애플리케이션 파일들이 함께 패키징되는 때에 그 애플리케이션 프로그램과 연관되고, 복수의 애플리케이션 파일을 호스팅하는 원격 머신(30')상에 저장되었다.
- <440> 일 실시예에서, 적어도 하나의 실행 필요요건은, 설치된 적어도 하나의 애플리케이션 파일을 실행시키는 버전의 운영체제를 시스템 상에 설치하는 것을 필요로 한다. 다른 실시예에서, 적어도 하나의 실행 필요요건은, 설치된 적어도 하나의 애플리케이션 파일을 실행시키는 제2 애플리케이션 버전을 시스템 상에 설치하는 것을 필요로 한다. 또 다른 실시예에서, 인스트럭션(instruction)은 설치된 적어도 하나의 애플리케이션 파일과 연관되는데, 이 인스트럭션은 적어도 하나의 실행 필요요건을 충족시키는 데에 실패한 클라이언트가 이용하도록 설치된 제2 애플리케이션 파일을 나타낸다. 또 다른 실시예에서, 인스트럭션은 적어도 하나의 설치된 애플리케이션 파일과 연관되는데, 이 인스트럭션은 적어도 하나의 실행 필요요건을 충족시키는 데에 실패한 클라이언트가 이용하도록 설치된 애플리케이션 파일의 제2 실행 방법을 나타낸다. 이러한 경우 일 실시예에서, 실행 방법은 적어도 하나의 설치된 애플리케이션 파일과 연관되는데, 이 실행 방법은 제2 애플리케이션을 포함하는 복수의 애플리케이션 파일을 로컬 머신 상에서 실행되도록 로컬 머신으로 스트리밍하는 것을 허가한다. 이러한 경우 다른 실시예에서, 로컬 머신의 평가에 의해, 로컬 머신 상에 포함되지 않은 상기 적어도 하나의 설치된 애플리케이션 파일과 연관된 적어도 하나의 특징이 식별된다. 이러한 경우 다른 실시예에서, 복수의 애플리케이션 파일의 실행에 대한 허가가 취소된다. 이러한 경우 다른 실시예에서, 이 복수의 애플리케이션 파일을 실행시키기 위하여 제2 실행 방법이 제공되는데, 이 제2 실행 방법은 복수의 애플리케이션 파일을 원격 머신 상에서 실행하고, 애플리케이션 출력 데이터를 원격 머신으로부터 로컬 머신으로 전달할 수 있게 한다.
- <441> 설치된 적어도 하나의 애플리케이션 파일을 적어도 하나의 서버 상에 저장한다(단계 2112). 일 실시예에서, 상기 설치된 적어도 하나의 애플리케이션 파일을 적어도 하나의 서버 상에 저장하기 이전에 설치된 애플리케이션 프로그램이 격리 환경내에서 실행된다. 이러한 경우 일 실시예에서, 설치된 애플리케이션 프로그램의 실행에 따라 추가의 애플리케이션 파일이 생성된다. 일 예로, 데이터 파일이 생성된다. 또 다른 실시예에서, 설치된

애플리케이션 프로그램은 설치를 완료하기 위한 정보를 필요로 하는데, 초기 설치 프로세스 이후에는 이 정보가 필요하다. 또 다른 실시예에서, 소프트웨어 제품 식별자, 라이선스 식별자 또는 다른 인증서와 같은 정보를 필요로 한다.

- <442> 일 실시예에서, 적어도 하나의 서버 상에 상기 설치된 적어도 하나의 애플리케이션 파일의 위치를 식별하는 식별자가 제공된다. 이러한 경우 일 실시예에서, 식별자는 UNC(Universal Naming Convention)를 따른다. 다른 실시예에서, 설치된 적어도 하나의 애플리케이션 파일은 CAB 파일과 같은 보관 파일내에 위치한다. 이러한 경우 일 실시예에서, 복수의 애플리케이션 파일을 보관 파일 내에 저장하고, 보관 파일은 적어도 하나의 서버 상에 저장된다. 이러한 경우 다른 실시예에서, 상기 설치된 적어도 하나의 애플리케이션 파일은 복수의 서버 상에 저장된다. 또 다른 실시예에서, 적어도 하나의 애플리케이션 파일은 애플리케이션 파일들을 저장하는 디렉토리 내에 위치한다.
- <443> 제2 애플리케이션, 상기 설치된 적어도 하나의 애플리케이션 파일, 적어도 하나의 서버의 위치 및 적어도 하나의 실행 방법의 목록이 생성된다(단계 2014). 일 실시예에서, 목록은 파일에 저장된다. 다른 실시예에서, 목록은 일목 요연한(manifest) 파일로 저장된다. 또 다른 실시예에서, 목록은 XML 파일로 저장된다.
- <444> 일 실시예에서, 복수의 애플리케이션, 복수의 애플리케이션 각각과 연관되어 있는 복수의 설치된 애플리케이션 파일 및 상기 복수의 설치된 애플리케이션 파일을 저장하는 적어도 하나의 서버의 목록이 생성된다. 또 다른 실시예에서, 제2 애플리케이션과 복수의 설치된 애플리케이션 파일들간의 연관관계를 포함하는 목록이 생성된다. 또 다른 실시예에서, 제2 애플리케이션과, 적어도 하나의 설치된 애플리케이션 파일을 포함하는 압축 파일과의 연관관계를 포함하는 목록이 생성된다.
- <445> 단계 2106을 다시 참조하면, 운영체제를 재부팅하지 않고도 애플리케이션 프로그램을 실행시키기 위한 정책 기반 방법으로 행하는 단계들의 일 실시예를 도시한다. 일 실시예에서, 설치된 애플리케이션을 실행하기 이전에 시스템을 재부팅할 필요가 없게 하는 설치 및 실행의 가시화 환경을 제공한다.
- <446> 이제 도 24를 참조하면, 흐름선도는 설치자 프로그램의 실행이, 그 설치자 프로그램이 실행되는 로컬 머신 상에서 운영체제를 재부팅하는 것을 필요로 하는 일 실시예를 도시한다. 종래의 애플리케이션 설치자는 애플리케이션이 설치되는 원격 머신으로 파일을 복사한다(단계 2402). 일 실시예에서, 파일의 복사는 원격 머신이 재부팅되게 할 수 있다. 애플리케이션 설치자는 잠긴 파일들에 쓰기를 시도한다(단계 2404). 일 실시예에서, 잠긴 파일은 운영체제가 실행되는(또는 "재부팅되는") 경우에만 쓰기가 이루어질 수 있다. MoveFileEx()Win32API에 MOVE_FILE_DELAY_UNTIL_REBOOT 옵션을 설정하고(단계 2406), 애플리케이션 설치자는 셋다운/재부팅 함수를 호출한다(단계 2408). 재부팅에 이어, 재부팅시 본래의 잠긴 파일들이 설치된다(단계 2410).
- <447> 이제 도 25를 참조하면, 블록도는 패키징 메커니즘이 애플리케이션 프로그램을 설치하는 원격 머신(30)의 일 실시예를 도시한다. 원격 머신(30)은 시스템 리소스(2502), 시스템 API(2504), 및 애플리케이션을 설치하는 데 이용되는 애플리케이션 설치자(2506)를 포함한다. 원격 머신(30)은 또한 함수-후킹 메커니즘(function-hooking mechanism; 2508), 설치후처리 프로세서 모듈(post-install processor module; 2510) 및 애플리케이션 격리 환경(2512)을 포함한다. 일 실시예에서, 애플리케이션을 격리 환경(2512)내로 설치함으로써, 원격 머신(30)을 재부팅하지 않고도 설치가 가능하게 된다. 이러한 경우 일 실시예에서, 격리 환경(2512)에서 보이는 시스템 리소스(2502)에 대해 이루어지는 변경은 원격 머신(30) 상에서의 대응 시스템 리소스(2502)를 변경시키지 않는다. 원격 머신(30) 상에서 시스템 리소스가 변경되지 않으므로, 시스템 리소스가 올바르게 변경되는 것을 방지하기 위한 재부팅은 필요 없게 된다.
- <448> 이제 도 25를 참조하면 보다 상세히, 시스템 리소스(2502)는 레지스트리 엔트리, 시스템 DLL, 및 원격 머신(30)이 실행중인 경우에는 운영 체제가 여기에 쓰기를 방지하는 다른 잠금 파일들을 포함할 수 있다. 시스템 API(2504)는 원격 머신(30)을 재부팅하는 것을 방지하기 위해, 애플리케이션 설치자(2506)에 의해 호출되어 함수-후킹 메커니즘(2508)에 의해 후킹되는, 시스템 재부팅에 이용되는 API를 포함한다.
- <449> 애플리케이션 격리 환경(2512)은 운영체제 리소스의 뷰를 갖는 환경을 애플리케이션 설치자(2506)에게 제공한다. 일 실시예에서, 애플리케이션 격리 환경(2512)은 격리 환경(556)이다. 일 실시예에서, 애플리케이션 격리 환경(2512)은 파일 시스템, 레지스트리 및 명명된 객체들과 같은 가상의 운영체제(virtualizaiton of operating system)를 제공한다. 일 실시예에서, 애플리케이션 설치자(2506)는 애플리케이션 격리 환경(2512)내에서 실행된다. 또 다른 실시예에서, 애플리케이션 설치자(2506)는 애플리케이션 프로그램을 애플리케이션 격리 환경(2512)에 설치한다. 또 다른 실시예에서, 애플리케이션 설치자(2506)는 애플리케이션 격리 환경(2512)

외부에서 실행되고 그 애플리케이션 프로그램을 애플리케이션 격리 환경(2512)내부에 설치한다.

- <450> 일 실시예에서, 애플리케이션 설치자(2506)가 애플리케이션 격리 환경(2512)내로 애플리케이션을 설치하면 애플리케이션 격리 환경(2512)은 원격 머신(30)이 재부팅될 필요성을 피하게 할 수 있다. 일 실시예에서, 애플리케이션 격리 환경(2512)은 잠긴 파일로 애플리케이션 파일을 복사하는 요청을 인터셉트한다. 또 다른 실시예에서, 애플리케이션 격리 환경(2512)은 잠긴 파일로 애플리케이션 파일을 복사하는 요청을 리디렉션한다. 또 다른 실시예에서, 애플리케이션 격리 환경(2512)은 가상 파일로 애플리케이션 파일을 복사하는 요청을 리디렉션한다. 또 다른 실시예에서, 애플리케이션 파일을 복사하는 요청을 리디렉션함으로써 원격 머신(30)을 재부팅할 필요 없이 애플리케이션 파일의 설치가 가능하게 된다. 예로서, 애플리케이션 설치자(2506)는 잠긴 파일, 예를 들어, C:\windows\system32\mfc40.dll로 쓰기를 시도하고, 애플리케이션 격리 환경(2512)은 그 요청을 인터셉트하여 파일을 또 다른 잠기지 않은 위치로 리디렉션한다. 잠긴 파일을 피하는 이러한 기능은 그 파일이 MoveFileEx() API와 MOVE_FILE_DELAY_UNTIL_REBOOT 플래그를 이용할 필요 없이 설치될 수 있음을 의미한다. 이러한 기능에 의해 원격 머신(30)을 재부팅할 필요가 없게 된다.
- <451> 일 실시예에서, 함수-후킹 메커니즘(2508)은 파일 시스템 필터 드라이버(564)이다. 또 다른 실시예에서, 파일 시스템 필터 드라이버(564)는 함수-후킹 메커니즘(2508)을 포함한다. 또 다른 실시예에서, 함수-후킹 메커니즘(2508)은 애플리케이션 설치자(2506)로부터의 요청을 인터셉트하여 원격 머신(30)을 재시동한다. 일 실시예에서, 애플리케이션 격리 환경(2512)은 잠기지 않은 파일로 애플리케이션 파일을 복사한다. 그러나, 애플리케이션 격리 환경(2512)은 애플리케이션 설치자(2506)가 요청하는 원격 머신(30) 재부팅을 행하지 않는다. 함수-후킹 메커니즘(2508)은 재부팅 요청을 인터셉트하고 애플리케이션 설치자(2506)에게 응답한다.
- <452> 애플리케이션 격리 환경(2512)은 잠기지 않은 파일로 애플리케이션 파일을 복사하는 것을 가능하게 한다. 그러나, 일 실시예에서, 애플리케이션을 설치하는 데에는 다른 동작들이 필요하고, 이들 동작들은 재부팅시에 발생할 수 있다. 재부팅을 방지하는 것이, 설치 프로세스에서 이들 동작들을 완료할 필요성을 없애는 것은 아니다. 함수-후킹 메커니즘(2508)은 애플리케이션의 설치와 연관된 동작을 수행하기 위한 기능을 제공할 수 있다.
- <453> 예를 들어, 애플리케이션의 설치 도중, HKLM\SYSTEM\CurrntControlSet\Control\Session_Manager\Pending-FileRenameOperations may be written과 같은 레지스트리 엔트리가 씌어질 수도 있다. 다른 애플리케이션들은 머신의 부팅시에 시동될 필요가 있는 드라이버 또는 서비스를 설치할 수도 있다. Post Install Processor Module(2510)은 설치 도중에 변경된 애플리케이션 파일들을 식별하고 그 애플리케이션 파일과 연관된 동작들을 수행한다.
- <454> 이제 도 26을 참조하면, 흐름선도는 애플리케이션 격리 환경(2512)에 애플리케이션을 설치하기 위하여 따르는 단계들의 일 실시예를 나타낸다. 애플리케이션 격리 환경(2512)은 서버 운영 체제의 가상화 뷰를 애플리케이션 설치자에게 제공한다(단계 2602). 애플리케이션 설치자(2506)가 재부팅을 하게 하는 것을 방지하기 위해 시스템 재부팅 및 셧다운과 관련된 서버상의 API가 후킹된다(단계 2604). 애플리케이션 설치자(2506)가 잠긴 파일로 파일-복사 동작을 요청하며, 이 요청은 인터셉트되어 비충돌 위치로 리디렉션된다(단계 2606). 애플리케이션 설치자(2506)가 시스템 API를 호출하여 재부팅을 시도하면, 그 요청을 인터셉트하여 재부팅이 방지된다(단계 2608). 설치-후속 프로세서 모듈(2510)은 재부팅 이후에 통상적으로 발생하는 동작들을 수행하고(단계 2610), 이어서 애플리케이션은 원격 머신(30)을 재부팅하지 않고 애플리케이션 격리 환경내에서 실행될 수 있다(단계 2612).
- <455> 일 실시예에서, 애플리케이션을 애플리케이션 격리 환경(2512)내로 설치한 후에, 패키징 메커니즘은 애플리케이션 프로그램의 설치 중에 생성되거나 변경된 복수의 애플리케이션 파일들을 식별한다. 이러한 경우 일 실시예에서, 복수의 애플리케이션 파일들은 원격 머신에 저장된다. 이러한 경우 다른 실시예에서, 복수의 애플리케이션 파일들을 검색하는 로컬 머신이 애플리케이션 프로그램을 실행시킬 수도 있다.
- <456> 일 실시예에서, 패키징 메커니즘(530)은 격리 환경(532) 및 파일 시스템 필터 드라이버(534)를 포함하는 원격 머신 상에서 실행되고 애플리케이션 파일을 격리 환경(532)에 설치한다. 이러한 경우 일 실시예에서, 원격 머신은 "클린 머신" 또는 "스테이징 머신"이라 한다. 이러한 경우 다른 실시예에서, 격리 환경(532)은 클린 머신 상에 운영체제가 제공하는 본래의 리소스의 수정 가능한 가상화된 인스턴스를 제공하는 애플리케이션 격리 범위(application isolation scope)를 포함한다. 이러한 경우 다른 실시예에서, 격리 환경(532)은 본래 리소스의 읽기 전용 뷰(read-only view)를 제공하는 시스템 격리 범위를 포함한다. 이러한 경우 다른 실시예에서, 본래 리소스의 읽기 전용 뷰는 클린 머신에 있는 파일 시스템 및 레지스트리의 스냅샷을 포함한다.

- <457> 일 실시예에서, 리디렉터(redirector)는 본래 리소스에 대한 변경 요청을 인터셉트한다. 일 실시예에서, 리디렉터는 파일 시스템 필터 드라이버(534)이다. 또 다른 실시예에서, 패키징 메커니즘(530)이 실행시키는 설치자 프로그램이 변경 요청을 한다. 또 다른 실시예에서, 본래 리소스에 대한 변경은 클린 머신상으로 애플리케이션 프로그램을 설치할 것을 요구한다. 또 다른 실시예에서, 리디렉터는 그 요청을 격리 환경(532)으로 리디렉팅한다.
- <458> 일 실시예에서, 본래 리소스에 대한 변경 요청을 격리 환경(532)으로 리디렉팅함으로써 애플리케이션 프로그램의 설치와 관련된 변경이 격리된다. 다른 실시예에서, 본래 리소스에 대한 변경 요청은 저장장치부에 기록되어 저장된다. 이러한 경우 일 실시예에서, 애플리케이션 프로그램의 설치와 연관된 모든 변경들은 저장소 컴포넌트에 존재한다. 이러한 경우 다른 실시예에서, 로컬 머신(552)이 저장장치부의 내용을 검색하고 로컬 머신(552)상의 격리 환경(556)에 있는 본래 리소스에 대한 변경을 구현하는 것에 의해 로컬 머신(552) 상에 애플리케이션 프로그램이 설치된다.
- <459> 일 실시예에서, 로컬 머신(10)의 개시 전 분석(pre-launch analysis)이 필요할 수도 있다. 이러한 경우 일 실시예에서, 로컬 머신(10)은 로컬 머신(10)에 적어도 하나의 특성이 포함되어 있음을 검증한다. 이러한 경우 다른 실시예에서, 로컬 머신(10)에 적어도 하나의 특성이 없다고 판단한 후에 로컬 머신(10)에 적어도 하나의 특성이 추가된다. 이러한 경우 다른 실시예에서, 적어도 하나의 특성은 애플리케이션 프로그램을 호스팅하는 원격 머신에 포함되고, 원격 머신이 이 적어도 하나의 특성을 포함하지 않으면 그 애플리케이션 프로그램의 실행을 저지할 것이다. 또 다른 실시예에서, 애플리케이션 프로그램은 실행되는 데에, 로컬 머신상에 적어도 하나의 특성이 존재할 것을 요구한다.
- <460> 일 실시예에서, 패키징 메커니즘은 로컬 머신 상에서의 개시 전 분석에 이용되는 적어도 하나의 특성이 식별될 수 있게 한다. 다른 실시예에서, 패키징 메커니즘은 로컬 머신 상에서의 실행에 이용할 수 있는 적어도 하나의 특성과 애플리케이션 프로그램이 연관될 수 있게 한다. 또 다른 실시예에서, 패키징 메커니즘은 실행 가능한 스크립트와 애플리케이션 프로그램이 연관될 수 있게 하며, 로컬 머신은 개시 전 분석을 완료하기 위한 실행 가능한 스크립트를 실행시킨다. 또 다른 실시예에서, 애플리케이션 프로그램이 실행된 이후에 적어도 하나의 특성이 로컬 머신 상에 존재할 것을 요구한다.
- <461> 패키징 메커니즘은 복수의 애플리케이션 파일을 사인(sign)하는 기능을 제공할 수도 있다. 일 실시예에서, 복수의 애플리케이션 파일을 사인하면, 로컬 머신이 그 복수의 애플리케이션 파일의 완전성(integrity)을 검증할 수 있게 된다. 또 다른 실시예에서, 복수의 애플리케이션 파일들을 사인함으로써 오류가 있는(corrupted) 애플리케이션 프로그램이 실행되는 것이 방지된다. 일 실시예에서, 복수의 애플리케이션 파일내 파일의 암호 체크섬(cryptographic checksum), 예를 들어, MD4 해시, MD5 해시 또는 SHA-1 해시가 컴퓨팅된다.
- <462> 다른 실시예에서, 복수의 애플리케이션 파일들의 모든 파일의 암호 체크섬이 컴퓨팅된다. 이러한 경우 일 실시예에서, 암호 체크섬은 제2 파일에 저장된다. 이러한 경우 다른 실시예에서, 제2 파일은 복수의 애플리케이션 파일과 연관된다. 일 실시예에서 제2 파일은 복수의 애플리케이션 파일에 추가된다. 다른 실시예에서, 제2 파일은 X.509 인증서와 같은 인증서를 이용하여 사인된다. 또 다른 실시예에서, 복수의 애플리케이션 파일들을 검색하는 로컬 머신은 인증서의 공용부를 이용하여 시그니처(signature)를 검증한다. 또 다른 실시예에서, 로컬 머신은 인증서의 공용부와, 인증서 신뢰 리스트의 식별자를 수신하여 시그니처를 검증한다. 이러한 경우 일 실시예에서, 로컬 머신은 인증서 신뢰 리스트의 식별자를 포함하는 레지스트 키를 수신한다.
- <463> 일 실시예에서, 패키징 메커니즘은 격리 환경을 전문화(customizing)하는 기능을 제공한다. 또 다른 실시예에서, 패키징 메커니즘은 격리 환경의 데피니션(definition)을 저장하는 파일을 생성하는 기능을 제공한다. 또 다른 실시예에서, 패키징 메커니즘은 애플리케이션 프로그램을 포함하는 복수의 애플리케이션 파일을 갖는 파일을 포함한다. 또 다른 실시예에서, 로컬 머신은 원격 머신으로부터 액세스 정보를 갖는 파일을 수신한다.
- <464> 일 실시예에서, 복수의 애플리케이션 파일들은 보관 파일에 저장된다. 이러한 경우 일 실시예에서, 보관 파일은 CAB 파일 형식이다. 이러한 경우 다른 실시예에서, 보관 파일 형식은 애플리케이션 프로그램에 의해 파일에 짧은 파일명이 지정되는 것을 지원하지 않는다. 이러한 경우 다른 실시예에서, WINDOWS 2000과 같은 운영체제는 애플리케이션 프로그램에 의해 파일에 짧은 파일명이 지정되는 것을 지원하지 않을 수도 있다. 다른 실시예에서, WINDOWS XP와 같은 운영체제는 애플리케이션 프로그램에 의해 파일에 짧은 파일명이 지정되는 것을 지원한다. 이러한 경우 일 실시예에서, 그 파일의 실행 요청은 합당한 그 파일의 짧은 파일명을 포함해야 한다.

- <465> 일 실시예에서, 복수의 애플리케이션 파일들내 파일의 긴 파일명을 그 파일의 짧은 명칭과 연관시키도록 매핑이 이루어질 수도 있다. 또 다른 실시예에서, 매핑은 복수의 애플리케이션 파일내 파일에 저장된다. 또 다른 실시예에서, 파일은 그 파일의 긴 파일명이 12개의 문자보다 긴 경우에만 짧은 파일명을 갖는다. 일 실시예에서, 짧은 파일명은 그 파일과 연관된 가상의 파일명이다. 이러한 경우 일 실시예에서, 파일은 실행될 로컬 머신(10)으로 전달되는데, 이 로컬 머신에서는 긴 파일명으로 파일이 저장된다. 이러한 경우 다른 실시예에서, 로컬 머신(10)상의 애플리케이션 파일은 짧은 파일명을 이용하여 파일의 실행을 요청한다. 이러한 경우 다른 실시예에서, 매핑에 의해, 그 파일의 실행 요청이 로컬 머신상의 파일명(긴 파일명)을 이용하지 않더라도 파일의 실행이 가능하게 된다.
- <466> 일 실시예에서, 패키징 메커니즘(530)은 매핑을 생성한다. 이러한 경우 일 실시예에서, 패키징 메커니즘(530)은 긴 파일명을 갖는 파일을 위해 짧은 파일명을 선택한다. 이러한 경우 다른 실시예에서, 패키징 메커니즘(530)이 실행 중인 원격 머신(30)의 운영체제는 긴 파일명을 갖는 파일을 위하여 짧은 파일명을 선택한다. 이러한 경우 다른 실시예에서, 원격 머신(30') 상에서 제2 짧은 파일명과 충돌하지 않는 고유한 짧은 파일명을 선택한다. 이러한 경우 다른 실시예에서, 패키징 메커니즘(530)이 실행하는 설치자 프로그램은 긴 파일명과 짧은 파일명간의 매핑을 포함하는 파일을 생성한다. 다른 실시예에서, 매핑은 그 파일을 검색하는 로컬 머신(10)으로 전달된다. 이러한 경우 일 실시예에서, 로컬 머신(10)은 그 파일을 실행 중이면 그 파일을 가리킨다.
- <467> 다음의 예는 복수의 파일을 선택하고, 로컬 머신으로 스트리밍하고, 로컬 머신 상에서 실행시키기 위해 상술한 방법 및 시스템이 어떻게 사용되는 지를 설명한다. 이들 예는 예시일 뿐 한정하고자 하는 것은 아니다.
- <468> 예1
- <469> 일 실시예에서, 로컬 머신(10)의 사용자는 애플리케이션 프로그램의 목록에서 확인되는 워드 처리 프로그램, 웹 브라우저 애플리케이션 또는 스프레드시트 프로그램과 같은 애플리케이션 프로그램에 대한 액세스를 요청한다. 이러한 경우 일 실시예에서, 로컬 머신(10)은 로컬 머신(10)에 이용 가능한 애플리케이션들의 목록을 원격 머신(30)으로부터 수신하는 근접 프로그램 애플리케이션(program neighborhood program)을 실행시킨다. 이러한 경우 또 다른 예로, 로컬 머신(10)은 원격 머신(30'')과 같은 웹 서버와 통신하여 애플리케이션의 목록을 수신한다. 로컬 머신(10)의 사용자는 목록의 애플리케이션 프로그램을 나타내는 그래프 도시를 선택함으로써 목록의 애플리케이션 프로그램에 대한 액세스를 요청할 수도 있다. 로컬 머신(10)의 사용자는 로컬 머신(10)에 미리 설치되지 않은 애플리케이션 프로그램에 대한 액세스를 요청할 수도 있다.
- <470> 로컬 머신(10)은 애플리케이션 프로그램에 대한 액세스 요청을 원격 머신(30)으로 전달한다. 로컬 머신(10)은 그 애플리케이션 프로그램을 포함하는 복수의 애플리케이션 파일들을 액세스를 할 수 있게 하는 원격 머신(30'')의 식별자를 수신한다. 로컬 머신(10)은 그 애플리케이션 프로그램의 실행에 필요한 적어도 하나의 특성을 확인한다. 이러한 실시예 중 일 예로, 로컬 머신(10)은 적어도 하나의 특성과 함께, 원격 머신(30)이 로컬 머신(10)으로 전달한 원격 머신(30'')의 식별자를 수신한다. 이러한 실시예 중 또 다른 예로, 로컬 머신(10)은 원격 머신(30'')의 식별자를 수신한 후에 원격 머신(30'')으로부터 적어도 하나의 특성을 검색한다. 로컬 머신(10)은 복수의 애플리케이션 파일들을 검색하기 위한 허가를 수신하기 이전에 적어도 하나의 특성을 포함할 것이 요구될 수도 있다. 이와 다르게, 로컬 머신(10)은 복수의 애플리케이션 파일을 실행시키기 이전에 적어도 하나의 특성을 포함하는 것이 요구될 수도 있다. 이러한 실시예 중 일 예로, 로컬 머신(10)은 복수의 애플리케이션 파일의 실행 전반에 걸쳐 적어도 하나의 특성을 포함하는 것이 요구될 수도 있다.
- <471> 로컬 머신(10)이 적어도 하나의 특성을 포함한다는 것을 로컬 머신(10)이 검증하면, 로컬 머신(10)은 복수의 애플리케이션 파일 내에서 적어도 하나의 애플리케이션 파일을 검색하고, 검색된 애플리케이션 파일을 실행시켜 그 애플리케이션 프로그램을 실행시킨다.
- <472> 예2
- <473> 원격 머신(30)은 애플리케이션 프로그램에 대한 액세스 요청을 로컬 머신(10)으로부터 수신한다. 원격 머신(30)은 로컬 머신(10)을 허가한다. 이러한 실시예 중 일 예로, 원격 머신(30)은 사용자명 및 패스워드와 같은 인증서를 로컬 머신(10)으로부터 요청한다. 이러한 실시예 중 또 다른 예로, 원격 머신(30)은 수집 에이전트(collection agent; 404)를 로컬 머신(10)으로 전달한다. 수집 에이전트(404)는 로컬 머신(10)에 관한 정보를 수집하고 로컬 머신(10) 허가에 이용되도록 그 정보를 원격 머신(30)으로 전달한다. 이러한 실시예 중 또 다른 예로, 원격 머신(30)은 로컬 머신(10)의 허가를 위해 로컬 머신(10)에 관한 정보를 정책 엔진(policy engine; 406)으로 제공한다. 원격 머신(30)은 정책 엔진(406)을 포함할 수도 있다. 이와 다르게, 원격 머신(30)은 정

책 엔진(406)을 포함하는 원격 머신(30')과 통신할 수도 있다.

- <474> 원격 머신(30)은 애플리케이션 프로그램의 실행 방법을 선택한다. 원격 머신(30)은 로컬 머신(10)의 허가에 응하여 선택을 할 수도 있다. 이러한 실시예 중 일 예로, 원격 머신(30)은 로컬 머신(10)에 관해 수집된 정보에 정책을 적용한다. 이러한 실시예 중 또 다른 예로, 원격 머신(30)은 애플리케이션 프로그램에 적용된 정책에 응하여 선택을 한다. 이러한 실시예 중 또 다른 예로, 원격 머신(30)은 애플리케이션 프로그램과 연관된 파일 유형에 적용된 정책에 응하여 선택을 한다. 원격 머신(30)은 애플리케이션 프로그램의 실행 방법을 선택하기 위하여 파일을 참고(consult)할 수도 있다.
- <475> 원격 머신(30)은 원격 머신(30') 상에 애플리케이션 프로그램을 실행하여 생성되는 애플리케이션-출력 데이터를 로컬 머신(10)이 수신할 수 있게 하는 애플리케이션 프로그램 실행 방법을 선택할 수도 있다. 원격 머신(30)은 그 애플리케이션 프로그램을 포함하는 복수의 애플리케이션 파일을 검색한 후 로컬 머신이 애플리케이션 프로그램을 로컬로(locally) 실행할 수 있게 하는 애플리케이션 프로그램 실행 방법을 선택할 수도 있다.
- <476> 일 실시예에서, 원격 머신(30)은 애플리케이션 스트리밍 세션에 걸쳐서 애플리케이션 프로그램을 포함하는 복수의 애플리케이션 파일을 검색하는 동안 로컬 머신(10)이 그 애플리케이션 프로그램을 로컬로 실행할 수 있게 하는 애플리케이션 프로그램 실행 방법을 선택한다. 이러한 실시예 중 일 예로, 로컬 머신(10)은 복수의 애플리케이션 파일을 호스팅하는 원격 머신과 애플리케이션 스트리밍 세션을 구축하는데, 로컬 머신(10)은 애플리케이션 스트리밍 세션에 걸친 복수의 애플리케이션 파일의 검색을 개시하고, 로컬 머신(10)은 복수의 애플리케이션 파일들 중 검색된 제1 애플리케이션 파일을 실행시키며, 이와 동시에 복수의 애플리케이션 파일들 중 제2 애플리케이션 파일을 검색한다. 이러한 실시예 중 또 다른 예로, 로컬 머신(10)은 복수의 애플리케이션 파일들 중 제1 애플리케이션 파일을 실행하고, 제1 애플리케이션 파일로부터 제2 애플리케이션 파일에 대한 액세스 요청을 수신하면 복수의 애플리케이션 파일들 중 제2 애플리케이션 파일을 검색한다.
- <477> 선택된 실행 방법에 의해서, 로컬 머신(10)이, 애플리케이션 프로그램을 포함하는 복수의 애플리케이션 파일 중 적어도 하나의 애플리케이션 파일을 검색할 수 있게 하는 실시예에서, 원격 머신(30)은 로컬 머신(10)에 의해 액세스하는 데에 이용될 수 있는, 애플리케이션 프로그램을 호스팅하는 원격 머신(30")을 식별한다. 원격 머신(30")은 애플리케이션 프로그램을 포함하는 복수의 애플리케이션 파일을 호스팅한다. 원격 머신(30")은 각종 애플리케이션 프로그램들을 포함하는 복수의 애플리케이션 파일들을 여러 개 호스팅할 수도 있다. 이러한 실시예들 중 일 예로, 원격 머신(30")은 애플리케이션 프로그램의 서로 다른 여러 개의 버전들 각각에 대한 복수의 애플리케이션 파일들을 호스팅한다.
- <478> 원격 머신(30")은 특정 애플리케이션 프로그램을 포함하는 복수의 애플리케이션 파일을 그 애플리케이션에 대한 설명과 연관시키는 파일을 호스팅한다. 이 파일은 또한 복수의 애플리케이션 파일을 머신으로 전달하기 이전에 머신 상에서 확인될 하나 이상의 실행 필요요건을 식별할 수도 있다. 이 파일은 원격 머신(30")의 네트워크 상에서의 위치 식별자를 더 포함할 수도 있다. 이러한 실시예 중 일 예로, 원격 머신(30)은 원격 머신(30")의 네트워크 상에서의 위치를 식별하기 위해 그 파일을 참조한다.
- <479> 원격 머신(30)은 원격 머신(30")을 선택한다. 원격 머신(30)은 로컬 머신(10)에 액세스할 수 있는 네트워크상의 위치를 갖는 원격 머신(30")을 선택할 수도 있다. 원격 머신(30)은 로컬 머신(10)과 호환되는 버전의 애플리케이션 프로그램을 호스팅하는 원격 머신(30")을 선택할 수도 있다. 원격 머신(30)은 애플리케이션 프로그램에 대한 액세스 요청을 수신하는 것에 응하여, 선택된 애플리케이션 프로그램 실행 방법의 식별자와 원격 머신(30")의 식별자를 로컬 머신(10)으로 전달한다. 원격 머신(30)은 또한 로컬 머신(10)으로 그 파일을 전달할 수도 있다.
- <480> 예3
- <481> 일 실시예에서, 로컬 머신(10)은 애플리케이션 프로그램을 포함하는 복수의 애플리케이션 파일을 액세스하는 원격 머신(30")의 식별자와 선택된 애플리케이션 실행 방법의 식별자를 수신한다. 로컬 머신(10)은 그 애플리케이션 프로그램에 대한 액세스 허가를 검증한다. 이러한 실시예 중 일 예로, 로컬 머신(10)은 자체의 개시 전 분석을 수행한다. 로컬 머신(10)은 적어도 하나의 특성을 식별하고, 로컬 머신(10)상에 그 적어도 하나의 특성이 존재하는 지를 검증한다. 적어도 하나의 특성은 애플리케이션 프로그램을 액세스하여 실행하는 것에 대한 허가를 유지하기 위한 필요요건일 수도 있다. 로컬 머신(10) 상에 적어도 하나의 특성이 존재한다는 것을 검증하면, 로컬 머신(10)의 특성들과 애플리케이션 프로그램의 시스템 요건들간의 적합성이 보장될 수 있고, 또한 보안 정책 또는 인가된 협정 준수가 보장될 수 있다.

- <482> 개시 전 분석을 성공적으로 완료하면, 로컬 머신(10)은 복수의 애플리케이션 파일들에 대한 액세스를 제공하는 원격 머신(30)과 애플리케이션 스트리밍 세션을 설정한다. 애플리케이션 스트리밍 세션은, 로컬 머신(10)이 복수의 애플리케이션 파일내의 파일을 요청하여 수신할 수도 있는 임의의 연결일 수 있다. 애플리케이션 스트리밍 세션을 설정함으로써, 복수의 애플리케이션 파일의 모든 파일들이 검색되기 이전에 복수의 애플리케이션 파일 중 제1 애플리케이션 파일이 로컬 머신(10)에 의해서 실행되게 할 수 있다. 로컬 머신(10)은 복수의 애플리케이션 파일 중 다른 애플리케이션 파일을 계속 검색하며 그 애플리케이션 프로그램의 실행을 시작할 수 있다. 이와 다르게, 로컬 머신(10)은 보관 파일에서 복수의 애플리케이션 파일을 검색하고, 보관 파일로부터 제2 애플리케이션 파일을 추출하는 동안 추출된 제1 애플리케이션 파일을 실행시킬 수도 있다.
- <483> 예4
- <484> 일 실시예에서, 로컬 머신(10)상의 애플리케이션 스트리밍 클라이언트(552)는 복수의 애플리케이션 파일을 원격 머신(30)에서 검색한다. 애플리케이션 스트리밍 클라이언트는 스트리밍 서비스(554), 격리 환경(556) 및 파일 시스템 필터 드라이버(564)를 포함한다. 스트리밍 서비스(554)는 복수의 애플리케이션 파일들을 요청하고 검색하기 위해 원격 머신(30)과 애플리케이션 스트리밍 세션을 설정한다. 스트리밍 서비스(554)는 그 애플리케이션 파일을 격리 환경(556)내에서 실행시킨다. 파일 시스템 필터 드라이버(564)는 실행 애플리케이션 파일로부터의 요청을 인터셉트하여 격리 환경(556)으로 리디렉션함으로써 애플리케이션 파일이 격리 환경(556)내에서 실행될 수 있게 한다.
- <485> 이러한 실시예 중 일 예로, 스트리밍 서비스(554)는 애플리케이션 프로그램을 포함하는 복수의 애플리케이션 파일을 포함하는 보관 파일을 검색한다. 스트리밍 서비스(554)는 보관 파일에서 복수의 애플리케이션 파일로부터 제1 애플리케이션 파일을 추출한다. 제1 애플리케이션 파일은 실행 가능한 파일일 수도 있다. 스트리밍 서비스(554)는 격리 환경(556)내에서 제1 애플리케이션 파일을 실행할 수도 있다. 제1 애플리케이션 파일의 실행이 애플리케이션 프로그램의 실행을 개시할 수도 있다.
- <486> 또 다른 실시예에서, 격리 환경(556)내에서 실행 중인 제1 애플리케이션 파일은 로컬 머신(10)으로부터 복수의 애플리케이션 파일 목록을 요청한다. 파일 시스템 필터 드라이버(564)는 이 목록에 대한 요청을 인터셉트하여 스트리밍 서비스(554)로 리디렉션한다. 스트리밍 서비스(554)가 복수의 애플리케이션 파일을 검색한 실시예에서, 스트리밍 서비스(554)는 복수의 애플리케이션 파일의 목록을 생성할 수도 있다. 스트리밍 서비스(554)가, 복수의 애플리케이션 파일을 포함하는 보관 파일을 검색한 실시예에서, 스트리밍 서비스(554)는 검색된 보관 파일에 포함된 목록에 응하여 복수의 애플리케이션 파일의 목록을 생성할 수도 있다. 다른 실시예에서, 스트리밍 서비스(554)는 복수의 애플리케이션 파일 중 적어도 하나의 애플리케이션이 원격 머신(30)에는 존재하지만 스트리밍 서비스(554)에 의해서 로컬 머신(10)에게는 아직 검색되지 않은 동안에만 복수의 애플리케이션 파일 목록을 검색한다. 이들 실시예에서, 스트리밍 서비스(554)는 검색된 목록에 응하여, 복수의 애플리케이션 파일 목록을 생성할 수도 있다. 이러한 실시예 중 일 예로, 스트리밍 서비스(554)는, 복수의 애플리케이션 파일이 로컬 머신(10)에 있지만 그 목록만 로컬 머신(10)에 있다는 것을 제1 애플리케이션 파일에게 나타낸다.
- <487> 예5
- <488> 일 실시예에서, 격리 환경(556)내에서 실행 중인 제1 애플리케이션 파일은 복수의 애플리케이션 파일의 목록에 의하여 확인된 파일에 대한 액세스를 로컬 머신(10)으로부터 요청한다. 요청 파일이 제1 애플리케이션 파일에게 액세스 가능한 격리 환경(556)의 사용자 범위에 있으면, 제1 애플리케이션 파일은 그 요청 파일을 액세스한다.
- <489> 요청 파일이 격리 환경(556) 또는 사용자 범위에 존재하지 않으면, 파일 시스템 필터 드라이버(564)는 그 요청을 인터셉트하여 스트리밍 서비스(554)로 리디렉션한다. 요청 파일이 복수의 애플리케이션 파일을 포함하는 보관 파일 내의 파일이면, 스트리밍 서비스(554)는 요청 파일을 추출하여 로컬 머신(10)에 저장한다. 스트리밍 서비스(554)는 그 파일을 격리 환경(556)내에 저장할 수도 있다. 파일에 대한 요청은, 그 파일이 격리 환경(556)에 저장되면 실행 완료된다(satisfied).
- <490> 요청 파일이 복수의 애플리케이션 파일을 포함하는 보관 파일내 또는 격리 환경(556)내에 존재하지 않으면, 스트리밍 서비스(554)는 그 파일을 원격 머신(30)으로부터 요청한다. 스트리밍 서비스(554)는 그 파일을 원격 머신(30)으로부터 애플리케이션 스트리밍 세션을 통하여 수신할 수도 있다. 스트리밍 서비스(554)는 수신한 파일을 격리 환경(556)에 저장한다. 파일에 대한 요청은, 그 파일이 격리 환경(556)에 저장되면 실행 완료된다.
- <491> 이러한 실시예 중 일 예로, 제2 애플리케이션 파일이 격리 환경(556)내 제2 사용자 범위에서 실행된다. 제2 애플

플리케이션 파일은 제1 애플리케이션 파일이 최초로 요청한 파일에 대하여 액세스를 요청한다. 요청 파일의 복사본이 제2 사용자 범위에 존재하지 않으면, 그 애플리케이션 파일에 대한 요청을 이행 완료하는 데에, 격리 환경(556)에 저장된 요청 파일의 복사본을 이용한다.

<492> 예6

<493> 일 실시예에서, 로컬 머신(10)은 원격 머신(30)으로부터 애플리케이션 프로그램의 선택된 실행 방법의 식별자와, 그 애플리케이션 프로그램을 포함하는 복수의 애플리케이션 파일에 대한 액세스를 제공하는 원격 머신(30')의 식별자를 수신한다. 로컬 머신(10)은 로컬 머신(10)의 개시 전 분석을 성공적으로 마친다. 로컬 머신(10)은 원격 머신(30)으로부터 애플리케이션 프로그램의 실행을 허가하는 라이선스를 수신한다. 이러한 경우 일 예로, 라이선스는 로컬 머신이, 하트비트 메시지를 세션 관리 서버(562)로 전달하여, 애플리케이션 실행에 대한 라이선스를 유지할 것을 요구한다. 하트비트 메시지는 애플리케이션 프로그램의 실행 개시 애플리케이션 프로그램의 실행 종료를 나타내는 메시지들 및 애플리케이션 프로그램의 실행 전반에 걸쳐 주기적으로 전송되는 메시지들을 포함할 수도 있다. 하트비트 메시지는 로컬 머신(10)이 네트워크에 연결되는 때, 또는 로컬 머신(10)이 네트워크에 연결 종료되는 때와 같이 로컬 머신(10)의 상태에 관한 메시지도 포함할 수 있다. 이러한 실시예 중 또 다른 예로, 라이선스는, 로컬 머신(10)이 그 애플리케이션 프로그램의 실행 라이선스를 갖는 소정 기간을 지정한다.

<494> 로컬 머신(10)은 원격 머신(30')과 애플리케이션 스트리밍 세션을 설정하고, 복수의 애플리케이션 파일내에서 적어도 하나의 애플리케이션 파일을 검색한다. 적어도 하나의 애플리케이션 파일의 실행 도중, 수신한 라이선스에 의해서 하트비트 메시지 전달이 요구되는 실시예에서, 적어도 하나의 애플리케이션 파일을 실행시키는 권한을 유지하기 위해 로컬 머신(10)은 하트비트 메시지를 세션 관리 서버(562)로 전송한다.

<495> 예7

<496> 일 실시예에서, 로컬 머신(10)은 애플리케이션의 프로그램의 선택된 실행 방법의 식별자와, 그 애플리케이션 프로그램을 포함하는 복수의 애플리케이션 파일에 대한 액세스를 제공하는 원격 머신(30')의 식별자를 수신한다. 로컬 머신(10)은 로컬 머신(10)의 개시 전 분석을 성공적으로 마친다. 로컬 머신(10)은 로컬 머신(10)이 그 애플리케이션 프로그램의 실행 권한을 갖는 소정의 기간을 지정하는 라이선스를 수신한다.

<497> 로컬 머신(10)은 원격 머신(30')과 애플리케이션 스트리밍 세션을 설정하고 적어도 하나의 애플리케이션 파일을 복수의 애플리케이션 파일내에서 검색한다. 이러한 실시예 중 일 예로, 로컬 머신(10)은 복수의 애플리케이션 파일의 서브세트를 검색하는데, 이 서브세트는 로컬 머신(10)이 네트워크에 연결되지 않으면 애플리케이션 프로그램을 실행하는 데에 필요한 각각의 파일을 포함한다. 로컬 머신(10)은 로컬 머신(10)의 개시에 그 서브세트를 저장한다.

<498> 소정의 기간 중 일 시점에서, 로컬 머신(10)은 네트워크로부터 연결 해제되고, 로컬 머신(10)의 사용자로부터 애플리케이션 프로그램에 대한 액세스 요청을 수신한다. 이러한 경우 일 예로, 로컬 머신(10)은 랩탑과 같은 디바이스이고, 로컬 머신(10)은 네트워크로의 연결을 금지하는, 기내와 같은 환경이다. 사용자로부터 요청을 수신하면, 로컬 머신(10)은 복수의 애플리케이션 파일 중의 애플리케이션 파일을 캐시로부터 검색하여 그 애플리케이션 프로그램을 실행시킬 수도 있다.

<499> 예8

<500> 또 다른 실시예에서, 로컬 머신(10)은 애플리케이션 파일의 선택된 실행 방법의 식별자와, 애플리케이션 프로그램을 포함하는 복수의 애플리케이션 파일에 대한 액세스를 제공하는 원격 머신(30')의 식별자를 수신한다. 로컬 머신(10)은 로컬 머신(10)상에 있는 제1 클라이언트 에이전트의 식별자를 검색하여, 복수의 애플리케이션 파일, 예를 들어, 애플리케이션 스트리밍 클라이언트의 검색을 실행한다.

<501> 이러한 실시예 중 일 예로, 로컬 머신(10)은 로컬 머신의 개시 전 분석을 성공적으로 마치지 못한다. 로컬 머신(10)에는 특정 디바이스 드라이버 또는 운영체제와 같이 그 애플리케이션 프로그램의 요건에 맞는 특성이 없을 수도 있다. 로컬 머신(10)에는 보안 정책에 따라 필요가 있는 특성, 예를 들어, 특정한 액티브 디렉토리(Active Directory)의 멤버십 또는 사설 네트워크에 대한 액세스 권한이 없을 수도 있다. 로컬 머신(10)은 애플리케이션 프로그램의 요건에는 적합하지 않은 머신 유형일 수 있으며, 예를 들어, 계산량이 상당한 애플리케이션 프로그램을 PDA가 액세스하려 하거나, 또는 사설 네트워크에서 원격 머신에 의해 호스팅되는 보안 애플리케이션을 키오스크(kiosk)의 공용 머신이 실행하려는 경우일 수 있다.

- <502> 로컬 머신(10)은 그 로컬 머신(10)에 애플리케이션 프로그램을 액세스하는 데에 필요한 적어도 하나의 특성이 없다고 판단하는 것에 따라, 애플리케이션 스트리밍 세션을 통해서 복수의 애플리케이션 파일을 검색하지 않기로 결정한다. 로컬 머신(10)은 식별된 제1 클라이언트 에이전트를 실행하는 대신에, 로컬 머신(10)에 있는 제2 클라이언트 에이전트를 실행한다. 이러한 실시예 중 일 예로, 로컬 머신(10)은 개시 전 분석을 성공적으로 마치지 못하면 실행시킬 제2 클라이언트 에이전트의 식별자를 수신한다. 로컬 머신(10)은 원격 머신(30') 상에서 애플리케이션 프로그램의 실행을 요청한다. 제2 클라이언트 에이전트는 원격 머신(30') 상에서 애플리케이션 프로그램을 실행함으로써 생성되는 애플리케이션 출력 데이터를 수신한다. 제2 클라이언트 에이전트는 로컬 머신(10)에 그 애플리케이션 출력 데이터를 디스플레이한다.
- <503> 예9
- <504> 일 실시예에서, 네트워크의 관리자는 로컬 머신(10)의 사용자가 애플리케이션 프로그램을 액세스할 수 있게 한다. 관리자는 애플리케이션 프로그램을 포함하는 복수의 애플리케이션 파일을 생성하기 위하여 원격 머신(30') 상에서 애플리케이션을 실행한다. 애플리케이션은 그래픽 사용자 인터페이스를 포함할 수도 있다. 관리자는 애플리케이션 프로그램과 그 애플리케이션 프로그램과 연관된 설치자 프로그램을 식별하기 위해, 애플리케이션 프로그램에 대한 액세스 권한에 적용할 정책을 정의하기 위해, 그리고 애플리케이션 프로그램을 액세스하거나 실행하려 하는 로컬 머신(10)이 충족시켜야 하는 요건 등의 제공 액세스 유형에 관한 특성들을 규정하기 위해 그래픽 사용자 인터페이스를 사용할 수 있다. 관리자는 애플리케이션 프로그램 전체, 또는 업그레이드 또는 패치와 같은 애플리케이션 프로그램의 일부를 설치하는 설치자 프로그램을 식별할 수도 있다.
- <505> 이러한 실시예 중 일 예로, 원격 머신(30)은 패키징 메커니즘(530)을 포함한다. 패키징 메커니즘(530)은 원격 머신(30)의 격리 환경(532)내에서 설치자 프로그램을 실행시킨다. 설치자 프로그램을 실행함으로써, 애플리케이션 프로그램과 연관된 적어도 하나의 애플리케이션 파일의 격리 환경(532)이 설치된다. 원격 머신(30)은 파일 시스템 필터 드라이버(534)를 포함하고, 로컬 머신(10) 상에 애플리케이션 파일을 설치하려는 설치자 프로그램의 요청을 인터셉트하고 격리 환경(532)으로 리디렉팅하여 애플리케이션 파일의 설치가 격리 환경(532)에서 이루어지게 한다. 패키징 메커니즘(530)은 각 애플리케이션 파일의 레코드를 격리 환경(532)내에 유지하는 데에 파일 시스템 필터 드라이버(534)를 이용할 수도 있다.
- <506> 설치자 프로그램은 복수의 애플리케이션 파일을 격리 환경(532)내로 설치할 수도 있다. 패키징 메커니즘(530)은 복수의 애플리케이션 파일의 애플리케이션 파일들의 목록을 포함하는 파일을 생성한다. 파일은 애플리케이션 프로그램의 유형과 같은, 복수의 애플리케이션 파일과 연관된 정보를 포함할 수도 있는데, 이 복수의 애플리케이션 파일들은 애플리케이션 프로그램의 버전, 그 애플리케이션 프로그램과 연관된 실행 필요 요건 및 정책 요건, 예를 들어 특정 애플리케이션 프로그램에 필요한 실행 방법을 포함한다. 패키징 메커니즘(530)은 복수의 애플리케이션 파일 및 파일 목록을 포함하는 파일을 원격 머신(30')에 저장한다.
- <507> 일 실시예에서, 네트워크의 관리자는 기존 애플리케이션 프로그램 또는 애플리케이션 프로그램을 포함하는 복수의 애플리케이션 파일 중의 애플리케이션 파일의 업그레이드 버전을 포함하는 애플리케이션 프로그램을 식별한다.
- <508> C. 클라이언트-서버 통신을 가속화하는 시스템 및 방법.
- <509> 본 발명의 일 실시예는 클라이언트-서버 통신을 가속화하는 시스템 및 방법에 관한 것이다. 이들 시스템 및 방법은 단독으로 또는 협조하여 이용될 수도 있고, 상술한 컴퓨팅 환경을 제공하는 임의의 시스템 및 방법과 결합되어서도 이용이 가능하다. 특히, 가속 기술의 네 가지 일반적인 범주는 아래와 같다.
- <510> 1. 동적 생성 객체의 캐싱(caching) : 일 실시예에서, 클라이언트-서버 통신은 데이터 통신 네트워크에서 동적으로 생성되는 객체를 캐싱하는 어플라이언스(1250)에 의해서 가속화된다.
- <511> 2. 연결 풀링(connection pooling) : 일 실시예에서, 클라이언트-서버 통신은 연결 풀링 기술을 행하는 어플라이언스(1250)에 의해서 가속화된다.
- <512> 3. 통합 캐싱(integrated caching) : 또 다른 예로, 클라이언트-서버 통신은 복수의 가속 기술과 통합된 캐싱을 행하는 어플라이언스(1250)에 의해서 가속화된다.
- <513> 4. 클라이언트측 가속화 : 또 다른 실시예에서, 클라이언트-서버 통신은 하나 이상의 가속 기술을 행하는, 클라이언트(10) 상의 실행 프로그램에 의해서 가속화된다.

- <514> 1. 동적으로 생성되는 객체의 캐싱
- <515> 여기서 보다 상세히 설명하는 일 실시예에서, 어플라이언스(1250)는 운영 체제의 커널 레벨에서, 캐싱 기능을 하나 이상의 다른 처리 작업, 예를 들어 암호해독, 압축 해제 또는 인증 및/또는 허가 등과 통합할 수도 있다. 이러한 예 구조를 본 명세서에서는 도 27에 따라 설명하였으나, 여기서 설명하는 동작들을 실행하는 데에 있어 다른 구조들을 이용할 수도 있다.
- <516> 도 27은 어플라이언스(1250)의 예 구조(3200)를 도시한다. 위에서 명시한 바와 같이, 구조(3200)는 예로서 제공되는 것일 뿐 제한하려는 것이 아니다. 도 2에 도시하는 바와 같이, 예 구조(3200)는 사용자 공간(3202)과 커널 공간(3204)으로 분리되는 소프트웨어 계층 및 하드웨어 계층(3206)으로 구성된다.
- <517> 하드웨어 계층(3206)은, 프로그램 및 서비스가 커널 공간(3204) 및 사용자 공간(3202)내에서 실행되는 하드웨어 요소를 제공한다. 하드웨어 계층(3206)은 또한 커널 공간(3204) 및 사용자 공간(3202)내의 프로그램 및 서비스가 어플라이언스(1250)에 대하여 내적으로 그리고 외적으로 데이터를 통신할 수 있게 하는 구조 및 요소들을 제공한다. 도 27에 도시하는 바와 같이 하드웨어 계층(3206)은 소프트웨어 프로그램 및 서비스를 실행하기 위한 처리 유닛(3262)과, 소프트웨어 및 데이터를 저장하기 위한 메모리(3264), 네트워크를 통하여 데이터를 송수신하기 위한 네트워크 포트(3266), 및 보안 소켓 레이어(Secure Sockets Layer) 연관 함수를 수행하여 네트워크를 통해 송수신되는 데이터를 처리하는 암호화 프로세서(3260)를 포함한다. 일 실시예에서, 중앙 처리 유닛(3262)은 단일 프로세서에서 암호화 프로세서(3260)의 함수를 수행할 수도 있다. 또한, 하드웨어 계층(3206)은 처리 유닛(3262)과 암호화 프로세서(3260) 각각을 위하여 다수의 프로세서를 포함할 수도 있다. 어플라이언스(1250)의 하드웨어 계층(3206)이 암호화 프로세서(3260)와 함께 도시되어 있으나, 프로세서(3260)는 SSL(Secure Socket Layer) 또는 TLS(Transport Layer Security) 프로토콜과 같은 임의의 암호화 프로토콜과 관련된 함수를 수행하는 프로세서일 수도 있다. 일 실시예에서, 프로세서(3260)는 GPP(general purpose processor)일 수도 있고, 다른 실시예에서는 임의의 보안 관련 프로토콜의 처리를 수행하는 실행 가능 명령어를 가질 수도 있다.
- <518> 어플라이언스(1250)의 하드웨어 계층(3206)은 도 27에서 특정 요소들과 함께 도시되어 있으나, 어플라이언스(1250)의 하드웨어부 또는 컴포넌트는 본 명세서에서 도 1c 및 1d에 도시하고 이와 관련하여 설명한 컴퓨팅 디바이스(예를 들어, 컴퓨팅 디바이스(135))의 임의의 유형 및 형태의 컴포넌트, 하드웨어 또는 소프트웨어를 포함할 수도 있다. 일 실시예에서, 어플라이언스(1250)는 서버, 게이트웨이, 라우터, 스위치, 브리지 또는 다른 유형의 컴퓨팅 또는 네트워크 디바이스를 포함할 수도 있고, 이들과 연관된 임의의 하드웨어 및/또는 소프트웨어 컴포넌트를 포함할 수도 있다.
- <519> 어플라이언스(1250)의 운영체제는 커널 공간(3204) 및 사용자 공간(3202)으로 가용 시스템 메모리를 할당, 관리하고, 또는 분리한다. 일 예의 소프트웨어 구조(3200)에서, 운영 체제는 임의의 유형 및/또는 형태의 Unix 운영체제일 수도 있다. 그러한 경우, 어플라이언스(1250)는 임의의 운영체제, 예를 들어, 임의 버전의 마이크로소프트® 윈도우즈 운영체제, 그와 다른 Unix 및 Linux 운영체제 릴리스, 매킨토시 컴퓨터용의 임의 버전의 맥 OS®, 임의의 임베디드 운영체제, 임의의 네트워크 운영체제, 임의의 실시간 운영체제, 임의의 오픈 소스 운영체제, 임의의 독점적 운영체제, 모바일 컴퓨팅 디바이스 또는 네트워크 디바이스용의 임의의 운영체제, 또는 어플라이언스(1250)상에서 실행하여 본 명세서에서 기술한 동작들을 수행할 수 있는 임의의 다른 운영 체제일 수 있다.
- <520> 커널 공간(3204)은 임의의 디바이스 드라이버, 커널 확장부(kernel extentions) 또는 다른 커널 관련 소프트웨어를 포함하는 커널(3230)을 실행하도록 예약된다. 당업자에게 공지된 바와 같이, 커널(3230)은 운영체제의 핵심이며 애플리케이션(1250)의 하드웨어 관련 컴포넌트들과 자원들에 대하여 액세스, 제어 및 관리를 한다. 일 실시예에 따르면, 커널 공간(3204)은 캐시 관리자(323)(통합 캐시라고도 하며 그 이점에 대해서는 아래에서 보다 상세히 설명)와 조업하는 네트워크 서비스 또는 프로세스를 다수 포함할 수도 있다. 또한, 커널(3230)의 실시예는 설치되거나 구성된, 또는 디바이스(1250)에 의해 사용되는 운영체제 실시예에 따라서 달라질 것이다.
- <521> 일 실시예에서, 디바이스(1250)는 클라이언트(10) 및/또는 서버(30)와 통신하기 위한 일 네트워크 스택(3267), 예를 들어, TCP/IP 기반 스택을 포함한다. 일 실시예에서, 네트워크 스택(3267)은 네트워크(30)와 같은 제1 네트워크 및 제2 네트워크(40)와 통신하는 데에 이용된다. 일 실시예에서, 디바이스(1250)는 클라이언트(10)의 제1 전송 계층 연결, 예를 들어 TCP 연결을 종료하고, 클라이언트(10)에 의해 사용될 서버(30)로의 제2 전송 계층 연결을 설정하는데, 예를 들어, 제2 전송 계층 연결은 어플라이언스(1250) 및 서버(30)에서 종료된다. 제1 및 제2 전송 계층 연결은 단일의 네트워크 스택(3267)을 통해서 설정될 수도 있다. 다른 실시예에서, 디바이

스(1250)는 다수의 네트워크 스택, 예를 들어, (3267), (3267')을 포함할 수도 있고, 제1 전송 계층 연결은 일 네트워크 스택(3267)에서, 그리고 제2 전송 계층 연결은 제2 네트워크 스택(3267')에서 설정되거나 종료될 수도 있다. 예를 들어, 일 네트워크 스택은 제1 네트워크 상에서 네트워크 패킷을 송수신할 수도 있고, 다른 네트워크 스택은 제2 네트워크 상에서 네트워크 패킷을 송수신할 수도 있다. 일 실시예에서, 네트워크 스택(3267)은 어플라이언스(1250)에 의해 전송되는 하나 이상의 네트워크 패킷을 큐잉(queueing)하기 위한 버퍼(3243)를 포함한다.

<522> 도 27에 도시하는 바와 같이, 커널 공간(3204)은 캐시 관리자(3232), 고속 계층 2-7 통합 패킷 엔진(3240), 암호화 엔진(3234), 정책 엔진(3236) 및 다중-프로토콜 압축 로직(3238)을 포함한다. 사용자 공간(3202) 대신 커널 공간(3204) 또는 커널 모드로 이들 컴포넌트 또는 프로세스들(3232, 3240, 3234, 3236, 3238)을 실행함으로써, 이들 컴포넌트들 각각의 단독 또는 조합으로의 수행을 향상시킨다. 커널 동작은, 이들 컴포넌트 또는 프로세스들(3232, 3240, 3234, 3236, 3238)이 디바이스(1250)의 운영체제의 코어 어드레스 공간에서 실행한다는 것을 의미한다. 예를 들어, 커널 모드에서 암호화 엔진(3234);을 실행함으로써, 커널로 암호화 및 암호 해독 연산을 이동시키고, 이로써 커널 모드의 커널 스레드 또는 메모리 공간과 사용자 모드의 스레드 또는 메모리 공간간의 이동 횟수를 감소시켜 암호화 성능을 향상시킨다. 예를 들어, 커널 모드에서 얻은 데이터는 사용자 모드에서 실행 중인 프로세스 또는 스레드로 전달되거나 복사, 예를 들어, 커널 레벨 데이터 구조로부터 사용자 레벨 데이터 구조로 될 필요가 없을 수 있다. 또 다른 측면에서 커널 모드와 사용자 모드간의 전환 횟수 또한 감소된다. 또한, 임의의 컴포넌트 또는 프로세스들(3232, 3234, 3236, 3238)간의 통신의 동기화는 커널 공간(3204)에서 보다 효율적으로 수행될 수 있다.

<523> 일 실시예에서, 컴포넌트(3232, 3234, 3236, 3238) 중 일부는 커널 공간(3204)에서 실행 또는 동작될 수 있고, 이들 컴포넌트(3232, 3234, 3236, 3238) 중 다른 일부는 사용자 공간(3202)에서 실행 또는 동작될 수도 있다. 일 실시예에서, 커널-레벨 데이터 구조는 하나 이상의 네트워크 패킷, 예를 들어, 클라이언트(10)로부터의 요청 또는 서버(30)로부터의 응답을 포함하는 네트워크 패킷의 일부를 액세스하는 데에 이용된다. 일 실시예에서, 커널-레벨 데이터 구조는 패킷 엔진(3240)에 의해 전송 계층 드라이버 인터페이스 또는 필터를 통해서 네트워크 스택(3267)으로 연결될 수 있다. 커널-레벨 데이터 구조는 네트워크 스택(3267)과 관련하여 커널 공간(3204)을 통해 액세스할 수 있는 임의의 인터페이스 및/또는 데이터를 포함할 수도 있다. 다른 실시예에서, 커널-레벨 데이터 구조는 임의의 컴포넌트 또는 프로세스(3232, 3234, 3236, 3238)에 의해서 이용되어 원하는 컴포넌트 또는 프로세스 동작을 수행할 수도 있다. 일 실시예에서, 컴포넌트(3232, 3234, 3236, 3238, 3240)는 커널-레벨 데이터 구조를 이용할 때에 커널 모드(3204)에서 실행중이고, 또 다른 실시예에서, 컴포넌트(3232, 3234, 3236, 3238, 3240)는 커널-레벨 데이터 구조를 이용할 때에 사용자 모드에서 실행 중이다. 일 실시예에서, 커널-레벨 데이터 구조는 제2 커널-레벨 데이터 구조로, 또는 원하는 임의의 사용자-레벨 데이터 구조로 전달되거나 복사될 수도 있다.

<524> 캐시 관리자(3232)는 임의의 유형 및 형태의 콘텐츠, 예를 들어, 원래 서버(originating servers; 30)에 의해서 서비스되는 동적으로 생성된 일 또는 다수의 객체를 캐시 액세스, 제어 및 관리하기 위한 소프트웨어, 하드웨어, 또는 소프트웨어 및 하드웨어의 임의의 조합을 포함할 수도 있다. 캐시 관리자(3232)가 처리 및 저장하는 데이터, 객체 또는 콘텐츠는 마크업 언어(markup language)와 같은 임의의 포맷의 데이터를 포함할 수 있으며, 또는 임의의 프로토콜을 통해 통신될 수 있다. 일 실시예에서, 캐시 관리자(3232)는 그 밖의 장소에 저장된 원시 데이터, 또는 이전에 계산된, 생성된 또는 전송된 데이터를 복제하는데, 원시 데이터는 패치(fetch), 계산하는 데에 액세스 시간이 더 걸리고, 또는 캐시 메모리 요소를 관독하는 데에 있어서 획득하는 액세스 시간이 더 걸릴 수도 있다. 일단 데이터가 캐시 메모리 소자에 저장되면, 그 후의 이용은 원시 데이터를 다시 패치하거나 다시 계산하는 것이 아니라 캐시된 복사본을 액세스함으로써 이루어지고, 이로써 액세스 시간을 감소시킨다. 일 실시예에서, 캐시 메모리 소자는 디바이스(1250)의 메모리(3264)에 데이터 객체를 포함한다. 다른 실시예에서, 캐시 메모리 소자는 메모리(3264)보다 고속 액세스 시간을 갖는 메모리를 포함할 수 있다. 또 다른 실시예에서, 캐시 메모리 소자는 임의의 유형 및 형태의 디바이스(1250) 저장소 컴포넌트, 예를 들어 하드 디스크의 일부를 포함할 수도 있다. 일 실시예에서, 처리 유닛(3262)은 캐시 관리자(3232)에 의해서 사용될 캐시 메모리를 제공할 수도 있다. 또 다른 실시예에서, 캐시 관리자(3232)는 데이터, 객체 및 다른 콘텐츠를 캐싱하기 위한 메모리, 저장장치 또는 처리 유닛 중 하나를 또는 이들의 조합을 이용할 수도 있다.

<525> 더욱이, 캐시 관리자(3232)는 본 명세서에서 설명하는 기술의 실시예를 수행하기 위한 임의의 로직, 함수, 규칙 또는 연산을 포함한다. 예를 들어, 캐시 관리자(3232)는 클라이언트(10) 또는 서버(30)로부터의 무효 명령 수

신시 또는 무효 기간의 만료를 기초로 하여 객체를 무효화하는 로직 또는 기능을 포함한다. 일 실시예에서, 캐시 관리자(3232)는 커널 공간(3204)에서 실행하는 프로그램, 서비스 프로세스 또는 작업으로서 동작할 수 있고, 다른 실시예에서는, 사용자 공간(3202)에서 실행하는 프로그램, 서비스 프로세스 또는 작업으로서 동작할 수도 있다. 일 실시예에서, 캐시 관리자(3232)의 제1 부분은 사용자 공간(3202)에서 실행하고, 제2 부분은 커널 공간(3204)에서 실행한다. 일 실시예에서, 캐시 관리자(3232)는 임의의 유형의 GPP(general purpose processor), 또는 임의의 다른 유형의 집적 회로, 예를 들어, FPGA(Field Programmable Gate Array), PLD(Programmable Logic Device), 또는 ASIC(Application Specific Integrated Circuit)를 포함할 수 있다.

<526>

정책 엔진(3236)은 예를 들어 인텔리전트 통계 엔진 등 프로그램 가능한 애플리케이션(들)을 포함할 수도 있다. 일 실시예에서, 정책 엔진(3236)은 사용자가 캐싱 정책을 확인, 특정, 정의 또는 구성할 수 있게 하는 구성 메커니즘을 제공한다. 정책 엔진(3236)은 일 실시예에서, 사용자가 선택한 캐싱 정책 결정을 가능하게 하는 데이터 구조, 예를 들어, 룩업 테이블 또는 해시 테이블을 지원하는 메모리를 액세스한다. 다른 실시예에서, 정책 엔진(3236)은 어플라이언스(1250)가 수행하는 보안, 네트워크 트래픽, 네트워크 액세스, 압축 또는 임의의 다른 함수 또는 연산을 액세스 제어 및 관리하는 데 더하여, 어플라이언스(1250)가 캐싱한 객체, 데이터 또는 콘텐츠를 결정하여 액세스, 제어 및 관리하기 위한 임의의 로직, 규칙, 함수 또는 연산을 포함할 수도 있다. 일 실시예에서, 정책 엔진(3236)에는 정책 엔진(406)의 기능이 통합될 수도 있다. 일 실시예에서, 정책 엔진(3236)은 수집 에이전트(404)가 제공한 정보를 토대로 하여 캐싱 정책을 행할 수도 있다. 일 실시예에서, 정책 엔진(3236)은 애플리케이션 실행 유형을 토대로 하여 캐싱 정책 결정을 행할 수도 있다. 일 실시예에서, 정책 엔진은 애플리케이션이 클라이언트(10)에 스트리밍되는 지를 토대로 하여 캐싱 정책 결정을 행할 수도 있다. 특수한 캐싱 정책의 다른 예에 대해서는 아래에서 더 설명한다.

<527>

암호화 엔진(3234)은 SSL 또는 TLS와 같은 임의의 보안 관련 프로토콜, 또는 이와 관련된 임의의 함수의 처리를 처리하기 위한 임의의 로직, 비즈니스 규칙, 함수 또는 연산을 포함한다. 예를 들어, 암호화 엔진(3234)은 어플라이언스(1250)를 통해서 통신되는 네트워크 패킷, 또는 이들의 일부를 암호화하고 암호화 해제한다. 암호화 엔진(3234)은 또한 클라이언트(10), 서버(30) 또는 어플라이언스(1250)를 대신하여 SSL 또는 TLS 연결을 설정할 수도 있다. 그러한 경우, 암호화 엔진(3234)은 SSL, 처리의 오프로드(offloading) 및 가속화(acceleration)를 제공한다. 일 실시예에서, 암호화 엔진(3234)은 터널링 프로토콜(tunneling protocol)을 이용하여 클라이언트(10)와 서버(30)간의 가상의 사설 네트워크를 제공한다. 일 실시예에서, 암호화 엔진(3234)은 암호화 프로세서(3260)와 통신한다. 다른 실시예에서, 암호화 엔진(3234)은 암호화 프로세서(3260)상에서 실행 중인 실행 가능 명령어들을 포함한다.

<528>

다중-프로토콜 압축 엔진(3238)은 디바이스(1250)의 네트워크 스택(3267)이 사용하는 임의의 프로토콜과 같은 하나 이상의 네트워크 패킷 프로토콜을 압축하기 위한 임의의 로직, 비즈니스 규칙, 함수 또는 연산을 포함한다. 일 실시예에서, 다중-프로토콜 압축 엔진(3238)은 MAPI(Messaging Application Programming Interface)(이메일), FTP(File Transfer Protocol), HTTP(HyperText Transfer Protocol), CIFS(Common Internet File System) 프로토콜(파일 전송), ICA(Independent Computing Architecture) 프로토콜, RDP(Remote Desktop Protocol), WAP(Wireless Application Protocol), 모바일 IP 및 프로토콜 VoIP(Voice Over IP) 프로토콜을 포함하는 임의의 TCP/IP 기반 프로토콜을 클라이언트(10)와 서버(30)간에 양방향으로 압축한다. 다른 실시예에서, 다중-프로토콜 압축 엔진(3238)은 HTML(HyperText Transfer Protocol) 기반 프로토콜의 압축을 제공하고, 다른 실시예에서는 XML(Extensible Markup Language)와 같은 임의의 마크업 언어의 압축을 제공한다. 일 실시예에서, 다중-프로토콜 압축 엔진(3238)은 어플라이언스(1250)용으로 설계된 임의의 프로토콜과 같은 임의의 고성능 프로토콜의 압축을 어플라이언스(1250) 통신에 제공한다. 또 다른 실시예에서 다중-프로토콜 압축 엔진(3238)은 임의의 페이로드(payload) 또는 임의의 통신을 T/TCP(Transaction TCP), TCP-SACK(TCP with selection acknowledgements), TCP-LW(TCP with large windows), TCP-베가스 프로토콜과 같은 혼잡 예측 프로토콜 및 TCP 스푸핑 프로토콜(TCP spoofing protocol)과 같은 수정된 전송 제어 프로토콜을 이용하여 압축시킨다.

<529>

이로써, 다중 프로토콜 압축 엔진(3238)은 데스크탑 클라이언트, 예를 들어, 마이크로소프트 아웃룩 및 비웹형 클라이언트(non-Web thin client), 인기있는 기업형 애플리케이션 예를 들어, 오라클, SAP 및 Siebel, 그리고 모바일 클라이언트, 예를 들어 Pocket PC를 통해서 사용자가 애플리케이션을 액세스하는 성능을 가속화한다. 일 실시예에서, 다중 프로토콜 압축 엔진(3238)은 커널 모드(3204)에서 실행시키고, 네트워크 스택(3267)을 액세스하는 패킷 처리 엔진(3240)과 통합함으로써 TCP/IP 프로토콜이 수행하는 프로토콜, 예를 들어 임의의 애플리케이션 계층 프로토콜을 압축할 수 있다.

<530> 고속 계층 2-7 통합 패킷 엔진(3240)은 또한 패킷 처리 엔진 또는 패킷 엔진이라고 하며 네트워크 포트(3266)를 통해서 어플라이언스(1250)가 수신 및 전송하는 패킷의 커널 레벨 처리를 관리한다. 고속 계층 2-7 통합 패킷 엔진(3240)은 네트워크 패킷의 수신 또는 네트워크 패킷의 전송과 같은 처리 도중에 하나 이상의 네트워크 패킷을 큐잉하는 버퍼를 포함할 수도 있다. 또한, 고속 계층 2-7 통합 패킷 엔진(3240)은 네트워크 포트(3266)를 통해서 네트워크 패킷을 전송 및 수신하기 위하여 하나 이상의 네트워크 스택(3267)과 통신한다. 고속 계층 2-7 통합 패킷 엔진(3240)은 암호화 엔진(3234), 캐시 관리자(3232), 정책 엔진(3236) 및 다중 프로토콜 압축 로직(3238)과 함께 작업한다. 특히, 암호화 엔진(3234)은 패킷의 SSL 처리를 수행하도록 구성되고, 정책 엔진(3236)은 요청-레벨 콘텐츠 스위칭 및 요청-레벨 캐시 리디렉션과 같은 트래픽 관리와 관련된 기능들을 수행하도록 구성되며, 다중 프로토콜 압축 로직(3238)은 데이터의 압축 및 압축 해제와 관련된 기능들을 수행하도록 구성된다.

<531> 고속 계층 2-7 통합 패킷 엔진(3240)은 패킷 처리 타이머(3242)를 포함한다. 일 실시예에서, 패킷 처리 타이머(3242)는 들어오는, 즉 수신하는, 또는 나가는, 즉 전송되는 네트워크 패킷의 처리를 트리거하기 위한 하나 이상의 시간 간격을 제공한다. 일 실시예에서, 고속 계층 2-7 통합 패킷 엔진(3240)은 타이머(3242)에 의하여 네트워크 패킷을 처리한다. 패킷 처리 타이머(3242)는 임의의 유형 및 형태의 신호를 패킷 엔진(3240)으로 제공하여 시간 관련 이벤트, 간격 또는 발생을 통지, 트리거 또는 통신한다. 다수의 실시예에서, 패킷 처리 타이머(3242)는 예를 들어, 100ms, 50ms 또는 25ms와 같은 밀리초 단위로 동작한다. 예를 들어, 일 실시예에서, 패킷 처리 타이머(3242)는 시간 간격을 제공하거나, 또는 네트워크 패킷이 고속 계층 2-7 통합 패킷 엔진(3240)에 의해서 10ms 시간 간격으로 처리되게 하지만, 다른 실시예에서, 5ms 시간 간격으로 그리고 또 다른 실시예에서는, 3, 2 또는 1ms 시간 간격으로도 처리되게 한다. 고속 계층 2-7 통합 패킷 엔진(3240)은 동작 중에 암호화 엔진(3234), 캐시 관리자(3232), 정책 엔진(3236) 및 다중-프로토콜 압축 엔진(3238)과 상호작용, 통합, 또는 통신한다. 그러한 경우, 암호화 엔진(3234), 캐시 관리자(3232), 정책 엔진(3236) 및 다중-프로토콜 로직(3238)의 임의의 로직, 함수, 또는 연산은 패킷 처리 타이머(3242) 및/또는 패킷 엔진(3240)에 의하여 수행될 수도 있다. 그러므로, 암호화 엔진(3234), 캐시 관리자(3232), 정책 엔진(3236) 및 다중-프로토콜 압축 로직(3238)의 임의의 로직, 함수 또는 연산은 패킷 처리 타이머(3242)를 통해서 제공되는 시간 간격으로, 예를 들어, 10ms 이하의 시간 간격으로 수행될 수도 있다. 예를 들어, 일 실시예에서, 캐시 관리자(3232)는 고속 계층 2-7 통합 패킷 엔진(3240) 및/또는 패킷 처리 타이머(3242)에 의하여 캐싱된 임의의 객체의 무효화를 수행할 수도 있다. 또 다른 실시예에서, 캐싱된 객체의 만료 또는 무효화는 패킷 처리 타이머(3242)의 시간 간격으로, 예를 들어, 10ms 마다로 설정될 수도 있다.

<532> 커널 공간(3204)과는 대조적으로, 사용자 공간(3202)은 사용자 모드 애플리케이션 또는 프로그램에 의해서 사용 되는, 그렇지 않으면 사용자 모드에서 실행 중인 운영체제의 메모리 영역 또는 부분이다. 사용자 모드 애플리케이션은 커널 서비스를 액세스하기 위하여 커널 공간(3204)을 직접 액세스하지 않을 수 있고, 서비스 호출을 이용한다. 도 27에 도시하는 바와 같이, 어플라이언스(1250)의 사용자 공간(3202)은 그래픽 사용자 인터페이스(GUI; 3210), 명령 라인 인터페이스(CLI; 3212), 셸 서비스(3214), 안정 모니터링 프로그램(health monitoring program; 3216) 및 데몬 서비스(3218)를 포함한다. GUI(210) 및 CLI(3212)는 시스템 관리자 또는 다른 사용자가 상호작용하여 어플라이언스(1250)의 동작을 제어하기 위한 수단을 예를 들어, 어플라이언스(1250)의 운영체제를 통하여 사용자 공간(3202)이나 커널 공간(3204) 중의 하나로 제공한다. GUI(3210)는 임의의 유형 및 형태의 그래픽 사용자 인터페이스일 수도 있고, 텍스트, 그래픽을 통하여, 또는 이와 다르게 브라우저와 같이 임의의 유형의 프로그램이나 애플리케이션으로 표현될 수도 있다. CLI(3212)는 임의의 유형 및 형태의 명령 라인 또는 텍스트 기반 인터페이스, 예를 들어, 운영 체제가 제공하는 명령 라인일 수도 있다. 예를 들어, CLI(3212)는 사용자가 운영 체제와 상호작용할 수 있게 하는 톨인 셸을 포함할 수도 있다. 일 실시예에서, CLI(3212)는 bash, csh, tcsh 또는 ksh형 셸을 통하여 제공될 수도 있다. 셸 서비스(3214)는 사용자가 GUI(3210) 및/또는 CLI(3212)를 통하여 어플라이언스(1250) 또는 운영체제와 상호작용하는 것을 지원하기 위한 프로그램, 서비스, 작업, 프로세스 또는 실행 가능 명령어들을 포함한다.

<533> 안정 모니터링 프로그램(3216)은, 네트워크 시스템이 올바르게 기능을 하고 있고, 네트워크를 통하여 사용자가 요청한 콘텐츠를 수신하고 있음을 모니터, 검사, 보고 및 보장하는 데에 이용된다. 안정 모니터링 프로그램(3216)은 하나 이상의 프로그램, 서비스, 작업, 프로세스 또는 실행 가능 명령어를 포함하여 어플라이언스(1250)의 임의의 동작을 모니터링하기 위한 로직, 규칙, 함수 또는 연산을 제공한다. 일 실시예에서, 안정 모니터링 프로그램(3216)은 어플라이언스(1250)를 통해 지나는 임의의 네트워크 트래픽을 인터셉트하여 점검한다. 다른 실시예에서, 안정 모니터링 프로그램(3216)은 임의의 적합한 수단 및/또는 메커니즘에 의해 다음 : 암호화

엔진(3234), 캐시 관리자(3232), 정책 엔진(3236), 다중-프로토콜 압축 로직(3238), 패킷 엔진(3240), 데몬 서비스(3218) 및 셸 서비스(3214) 중 하나 이상과 상호작용한다. 그러한 경우, 안정 모니터링 프로그램(3216)은 임의의 API(application programming interface)를 호출하여 어플라이언스(1250)의 어떤 부분의 상태, 스테이터스(status) 또는 건강을 판단할 수도 있다. 예를 들어, 안정 모니터링 프로그램(3216)은 프로그램, 프로세스, 서비스 또는 작업이 활성이며 현재 실행 중인 지를 검사하기 위하여 주기적으로 상태 질의를 핑(ping)하거나 전송할 수도 있다. 또 다른 실시예에서, 안정 모니터링 프로그램(3216)은 임의의 프로그램, 프로세스, 서비스 또는 작업이 제공하는 임의의 스테이터스, 에러 또는 히스토리 기록을 검사하여 어플라이언스(1250)의 어떤 부분의 상태, 스테이터스 또는 에러를 판단할 수도 있다.

<534> 데몬 서비스(3218)는 백그라운드에서 또는 지속적으로 실행되어 어플라이언스(1250)가 수신한 주기적인 서비스 요청을 처리(handle)하는 프로그램이다. 일 실시예에서, 데몬 서비스는 이 요청을 다른 프로그램 또는 프로세스, 예를 들어 적절한 또 다른 데몬 서비스(3218)로 전달할 수도 있다. 당업자에게 공지된 바와 같이, 데몬 서비스(3218)는 네트워크 제어와 같은 지속적 또는 주기적 시스템 범위 함수(system wide functions)를 수행하도록 또는 임의의 원하는 작업을 수행하도록 실행될 수 있다. 일 실시예에서, 하나 이상의 데몬 서비스(3218)가 사용자 공간(3202)에서 실행되고, 다른 실시예에서 하나 이상의 데몬 서비스(3218)는 커널 공간에서 실행된다.

<535> 하나 이상의 동적으로 생성되는 객체와 같은 동적 콘텐츠는 도 1a에 도시한 로컬 또는 원격인 하나 이상의 클라이언트(10)로부터의 객체 요청을 처리하는 백-엔드 데이터베이스 및/또는 애플리케이션 또는 원래 서버(30)라 하는 서버에 의하여 생성될 수도 있다. 이들 애플리케이션 또는 데이터베이스가 클라이언트로부터 수신한 입력 관련 데이터 등의 데이터를 처리하기 때문에, 이들 데이터베이스 및 애플리케이션이 서브하는 응답 객체들은 변경될 수도 있다. 객체가 원래 서버의 애플리케이션 또는 데이터베이스에 의해 생성되고 나서야 비로소, 새로 발생한(fresh)한 것이 되므로 캐시에 저장될 것이다. 예를 들어, 동일한 입력 세트가 주어진 경우, 제1 인스턴스의 동적으로 생성된 객체는 제2 인스턴스의 동적으로 생성된 객체와 상이할 것이다. 또 다른 예로, 동일한 객체가 상이한 입력 세트를 이용하여 동적으로 생성될 수 있어, 제1 인스턴스의 객체가 제2 인스턴스의 객체와 상이하게 생성된다.

<536> 네트워크 성능을 향상시키기 위해, 어플라이언스(1250)는 다양한 방법(아래에서 상세히 설명함)를 통하여 동적으로 생성된 콘텐츠를 캐싱하는 데 있어 발생할 수 있는 문제를 해결하도록 설계 및 구성되었다. 본 명세서에 기술한 실시예에서는, 캐시에 저장된 동적 생성 콘텐츠를 어플라이언스(1250)가 보다 효율적이고 효과적으로 무효화시키는 방법을 한 가지 이상 포함한다. 또한, 어플라이언스는 새로 발생한 객체를 제어하여 캐싱하는 방법을 포함할 수 있다. 캐시 메모리는 통상적으로 객체 요청에 대한 모든 응답을 저장하는데, 그러한 응답이 캐싱 불가능한 것으로 표시되지 않을 만큼만 길게 저장한다. 본 명세서에서 설명하는 바와 같이, 동적 생성 콘텐츠를 효율적으로 캐싱하는 데에는, 원래 서버에서 변경된 캐시 메모리의 객체를 시기 적절하게(timely) 무효화시킬 수 있게 하는 방법이 필요하다. 시기 적절한 무효화에 의해서, 캐시가 갱신되지 않은 콘텐츠(stale content) — 특히 동적 생성 콘텐츠와 관련된 작업을 (특히, 콘텐츠에 대한 변경이 불규칙하게 발생하는 경우) 제공하는 것을 방지할 수 있다. 동적으로 생성된 콘텐츠의 시기 적절한 무효화를 보장하는 여러가지 방법들을 아래에 제시한다.

<537> a. 통합 기능(Integrated Functionality)

<538> 일 측면에서, 동적으로 생성된 객체의 캐싱은 패킷 처리 타이머(3242)에 의하여 캐시 관리자(3232), 정책 엔진(3236), 암호화 엔진(3234) 및/또는 다중-프로토콜 압축 엔진(3238)의 함수, 로직, 또는 연산을 고속 계층 2-7 통합 패킷 엔진(3240)의 패킷 처리 연산과 통합하는 방법에 관한 것이다. 예를 들어, 캐시 관리자(3232)의 연산은 네트워크 패킷의 송수신과 같은 패킷 처리 연산에 이용되는 패킷 처리 타이머(3242)의 시간 간격내에서 수행될 수 있다. 일 실시예에서, 패킷 처리 연산과의 통합 및/또는 패킷 처리 타이머를 이용하면, 캐시 관리자(3232)는 만료 시간을 매우 짧게 줄이며 객체를 캐싱할 수 있는데, 이에 대해서는 아래에서 더 설명하기로 한다. 일 실시예에서, 패킷 처리 타이머(3242)에 따르는 캐시 관리자(3232)는 객체를 캐싱하는 매우 짧은 기간 내에 객체를 무효화시키는 무효화 명령을 수신할 수도 있다.

<539> 도 28a에 도시하는 방법(3300)은 고속 계층 2-7 통합 패킷 엔진 또는 패킷 처리 엔진(3240)에 의해 네트워크 패킷을 처리하는 시간 간격과 관련하여 또는 처리 도중에 동작을 수행하기 위한 캐시 관리자(3232), 정책 엔진(3236), 암호화 엔진(3234) 및/또는 다중-프로토콜 압축 엔진(3238)의 일 실시예를 도시한다. 간단히 살펴보면, 방법(330)의 단계 3310에서, 디바이스(1250)는 네트워크 패킷을 수신하거나 또는 네트워크 패킷을 전

송하라는 요청을 받는다. 단계 3315에서, 디바이스(1250)는 패킷 처리 엔진(3240)에게 패킷 처리 타이머(3242)에 따라 네트워크 패킷을 처리하라는 요청을 한다. 패킷 처리 동작의 일부로서 또는 그와 연관하여, 단계 3320에서는 패킷 처리 엔진(240)이 캐시 관리자(3232), 정책 엔진(3236), 암호화 엔진(234) 및/또는 다중-프로토콜 압축 엔진(3238)에게 캐싱된 객체에 대하여 동작을 수행하라는 요청을 한다. 단계 3325에서, 캐시 관리자(3232), 정책 엔진(3236), 암호화 엔진(234) 및/또는 다중-프로토콜 압축 엔진(3238)은 요청 동작을 수행하는데, 요청 동작은 본 명세서에서 설명한 방법 중 하나 또는 방법들의 조합을 포함할 수도 있다. 일 실시예에서, 캐시 관리자(3232)는 캐싱된 객체의 무효화를 판단하고 캐싱된 객체를 무효로 표시한다. 일 실시예에서, 캐시 관리자(3232)는 패킷 처리 엔진(3240)의 요청에 응하여 무효 객체를 삭제(flush)한다. 캐시 관리자(3232)는 패킷 처리 타이머(3242)에 따라서 이들 동작을 수행 중이므로, 객체의 무효화는 밀리초 크기의 시간내에 발생할 수 있으며, 객체는 패킷 처리 타이머(3242)가 제공하는 시간 간격, 예컨대 10ms으로 만료될 수 있다.

<540> 방법(3300) 중 보다 상세히 단계 3310에서는, 어플라이언스(1250)가 하나 이상의 네트워크 패킷을 수신 및/또는 전송한다. 일 실시예에서, 어플라이언스(1250)는 네트워크(40) 또는 네트워크(40') 상에서 하나 이상의 네트워크 패킷을 전송하라고 요청을 한다. 또 다른 실시예에서, 어플라이언스(1250)는 하나의 포트(3266) 상에서 네트워크 패킷을 수신하고 그 포트(3266) 또는 그와 다른 포트(3266') 상에서 네트워크 패킷을 전송한다. 일 실시예에서, 어플라이언스(1250)의 패킷 엔진(3240)은 하나 이상의 네트워크 패킷을 전송하거나 전송하라는 요청을 한다. 일 실시예에서, 어플라이언스(1250)는 제1 네트워크(40) 상에서 패킷을 수신 또는 전송하고, 또 다른 실시예에서 어플라이언스(1250)는 제2 네트워크(40') 상에서 패킷을 수신 또는 전송한다. 일 실시예에서, 어플라이언스(1250)는 동일한 네트워크(40) 상에서 패킷을 수신 또는 전송한다. 다른 실시예에서, 어플라이언스(1250)는 네트워크 패킷을 수신 및/또는 하나 이상의 클라이언트(10)로 전송한다. 다른 실시예에서, 어플라이언스(1250)는 네트워크 패킷을 수신 및/또는 하나 이상의 서버(30)로 전송한다.

<541> 단계 3315에서, 디바이스(1250)는 디바이스(1250)의 네트워크 포트(3266)에서 네트워크 패킷을 수신할 때, 디바이스(1250)로부터 네트워크 패킷을 전송하라는 요청을 할 때, 또는 하나 이상의 네트워크의 패킷 수신 및/또는 전송의 임의의 조합에서 패킷 처리 엔진(3240)의 패킷 처리 동작을 트리거하거나 요청할 수 있다. 일 실시예에서, 패킷 처리 엔진(3240)의 패킷 처리 동작은 패킷 처리 타이머(3242)가 제공하는 신호를 통해서 트리거된다. 일 실시예에서, 패킷 처리 타이머(3242)는 하나 이상의 네트워크 패킷의 수신 및/또는 전송에 관련된 인터럽트-구동 또는 이벤트 구동형 타이머를 제공할 수도 있다. 일 실시예에서, 패킷 처리 타이머(3242)는 디바이스(1250)를 통한 네트워크 패킷의 수신 및/또는 전송 레이트에 의해서, 또는 패킷의 각 패킷 또는 배치 처리되는 레이트에 의해서 구동된다. 그러한 경우, 패킷 처리 타이머(3242)는 하나 이상의 패킷 처리 동작의 각 세트 이후에 트리거되어 리셋될 수도 있다. 또 다른 실시예에서, 패킷 처리 타이머(3242)는 수신 패킷 처리 또는 수신 패킷 전송과 같은 기능 또는 동작을 수행하도록 패킷 처리 엔진(3240)을 트리거, 웨이크업 또는 시그널링하기 위하여 동일한 또는 일정하지 않은 시간 간격을 제공한다. 도 27이 디바이스(1250)와 관련하여 위에서 설명한 바와 같이, 패킷 처리 타이머(3242)는 밀리초의 크기로 동작, 예를 들어, 시간 간격의 형성 또는 패킷 처리 동작의 트리거를 10ms 이하의 간격으로 할 수도 있다. 패킷 처리 타이머의 유지 단위(granular) 타이머 기능은 다양한 방식으로 패킷 처리 엔진(3240)의 패킷 처리 동작에 제공되어 동작에 이용될 수 있다.

<542> 방법(330)의 단계 3320에서, 패킷 처리 엔진(3240)은 캐시 관리자(3232), 정책 엔진(3236), 암호화 엔진(3236) 및/또는 다중-프로토콜 압축 엔진(3238) 중 하나 이상의 동작을 수행하라는 요청을 한다. 일 실시예에서, 패킷 처리 엔진(3240) 또는 패킷 처리 타이머(3242)는 캐시 관리자(3232), 정책 엔진(3236), 암호화 엔진(3234) 및/또는 다중-프로토콜 압축 엔진(3238) 중 하나 이상에게 시그널링을 한다. 패킷 처리 엔진(3240)은 네트워크 패킷, 하나 이상의 패킷들의 패킷 처리 동작 도중, 그 이전 또는 그 이후의 임의의 시점에서 동작을 시그널링하거나 요청할 수도 있다. 일 실시예에서, 패킷 처리 엔진(240)은 그 요청을 패킷 처리 타이머(3242)가 제공하는 시간 간격의 만료 또는 패킷 처리 타이머(3242)의 트리거시에 네트워크 패킷에 관한 패킷 처리 동작을 수행하기 이전에 한다. 또 다른 실시예에서 패킷 처리 엔진(3240)은 하나 이상의 패킷 처리 동작을 수행하는 과정 중에 그 요청을 한다. 예를 들어, 함수 호출내에서와 같이 동작의 실행 도중, 패킷 처리 엔진(240)은 캐시 관리자(323), 정책 엔진(3236), 암호화 엔진(3234) 및/또는 다중-프로토콜 압축 엔진(238) 중 하나 이상에게 API(application programming interface) 호출을 할 수도 있다. 다른 실시예에서, 패킷 처리 엔진(3240)은 네트워크 패킷 처리 동작의 완료시에 그 요청을 한다.

<543> 단계 3325에서, 캐시 관리자(323), 정책 엔진(3236), 암호화 엔진(3234) 및/또는 다중-프로토콜 압축 엔진(3238) 중 하나 이상이 요청된 동작을 수행한다. 일 실시예에서, 커널(3204), 예를 들어 커널 API(application programming interface)를 통해 제공되는 임의의 기능 또는 동작의 실행이 요청될 수도 있다. 그러한 경우, 디

바이스(1250)의 기능들 중 임의의 기능은 패킷 처리 타이머(3232)를 통한 패킷 처리의 타이밍 또는 타이밍 간격과 연관되어 수행될 수도 있다. 일 실시예에서, 요청된 동작은 패킷 처리 엔진(3240)의 패킷 처리 동작과 연관되어 동기적으로 수행된다. 예를 들어, 패킷 처리 동작은 요청된 동작의 완료 또는 그로부터의 응답을 대기하며 동작의 완료 또는 그로부터의 응답시에 계속된다. 다른 실시예에서, 요청된 동작은 패킷 처리 동작과 비동기적으로 수행된다. 예를 들어, 패킷 처리 엔진(3240)은 동작을 수행하라는 요청은 전송하지만 동작으로부터 응답을 수신하는 것을 차단하거나 또는 대기하지 않는다. 도 28b에 도시한 방법(3350)과 관련하여 보다 상세히 설명하는 바와 같이, 패킷 처리 엔진(3240)은 캐시 관리자(3232)로 하여금 임의의 캐시 관리 기능, 예를 들어 객체의 만료 또는 무효화에 대한 검사, 객체를 무효로 표시, 또는 무효 또는 만료된 객체를 플러싱을 수행하라는 요청을 할 수도 있다.

<544> 일 실시예에서, 단계 3320에서 패킷 처리 엔진(3240)은 다중 요청을, 예를 들어, 캐시 관리자(232)에게 제1의 요청을 그리고 암호화 엔진(3234)에게 제2의 요청을 전송한다. 다른 실시예에서, 단계 3320에서 패킷 처리 엔진(3240)은 커널(3230)을 통해서와 같이 디바이스(1250)가 분배하는 다중 요청을 포함하는 단일의 요청을 디바이스(1250)가 의도한 컴포넌트로 전송한다. 일 실시예에서, 요청은 후속적으로 서로 통신된다. 또 다른 실시예에서, 요청은 이전의 요청의 스테이터스, 결과, 성공 또는 완료 여부에 따라 달라질 수도 있다. 예를 들어, 정책 엔진(3236)으로의 제1 요청은 네트워크 패킷과 연관된 또 다른 디바이스 또는 사용자로부터의 네트워크 패킷을 처리하기 위한 정책을 결정하는 데에 이용될 수도 있다. 정책 엔진(3236)의 정책을 기초로 하여, 제1 요청의 결과에 따라서 캐시로의 제2 요청이 이루어지거나 이루어지지 않을 수도 있다. 커널 공간(204)에서 캐시 관리자(3232), 정책 엔진(3236), 암호화 엔진(3234) 및/또는 다중-프로토콜 압축 엔진(3238)이 디바이스(1250)의 패킷 처리 엔진(3240)과 통합되면, 본 명세서에서 설명한 바와 같이 패킷 처리 동작과 통합되고 그에 의해서 트리거될 수 있는 디바이스(1250) 동작이 다양하다.

<545> b. 무효화 유지 단위(Invalidation Granularity)

<546> 또 다른 측면에서, 동적으로 생성된 객체의 캐싱은 패킷 처리 타이머가 제공하는 시간 단위와 같은 짧은 단위 시간 간격으로 캐시가 저장하는 객체의 만료 시간을 구성하는 기능을 포함한다. 이러한 특성을 "무효화 유지 단위"라 한다. 그러한 경우 일 실시예에서, 매우 짧은 시간 간격으로 단축된 만료 시간으로 객체가 캐싱될 수 있다. 다른 실시예에서, 캐시 관리자는 또한 패킷 처리 타이머에 따라서 매우 짧은 객체 캐싱 시간내에 객체를 무효화하기 위한 무효화 명령을 수신할 수 있다. 캐시는 일초에 여러번 자주 변경되는 객체를 캐싱하여 제공할 수 있다. 한 가지 방법은 시간 증분이 밀리초의 크기로 동작할 수 있는 일 실시예의 디바이스가 이용하는 패킷 처리 타이머에 영향을 주어 무효화 또는 만료 유지 단위를 10ms 미만으로 낮추게 하는 것이다. 이와 반대로, 전통적인 캐시는 일반적으로 1초 미만의 만료 또는 무효화 유지 단위를 설정하지 않는다.

<547> 이제 도 28b를 참조하면, 패킷 처리 타이머(3242) 및/또는 패킷 처리 엔진(3240)에 따라서 캐싱된 객체를 무효화시키거나 만료시키는 방법(3350)의 일 실시예가 도시되어 있다. 그러한 경우, 일 실시예에서, 캐싱된 객체는 예를 들어, 10ms 미만의 밀리초의 단위로 무효화되거나 만료될 수 있다. 간략히 살펴보면, 방법(3350)의 단계 3355에서, 캐시 관리자(3232)는 패킷 처리 타이머(3242)에 따라서 패킷 처리 엔진(3240)을 통해 동작을 수행하라는 시그널 또는 요청을 수신한다. 단계 3360에서, 캐시 관리자(3232)는 캐싱된 객체, 예를 들어, 동적으로 생성된 객체가 무효화 또는 만료된 상태인 지를 판단한다. 단계 3365에서, 객체가 무효화되었으면, 캐시 관리자(3232)는 그 객체를 무효로 표시하고, 단계 3370에서 캐시 관리자(3232)로부터 무효화 객체를 삭제한다(flush).

<548> 단계 3355를 보다 세부적으로, 일 실시예에서, 캐시 관리자(3232)는 캐시 관련 동작을 수행하라는 요청 또는 시그널을 네트워크 패킷 처리 중의 임의의 시점에 수신할 수도 있다. 일 실시예에서, 단계 3355에서 캐시 관리자(3232)는 디바이스(1250)가 수신한 또는 전송할 네트워크 패킷을 처리하기 이전에 동작 요청을 수신한다. 또 다른 실시예에서, 캐시 관리자(3232)는 네트워크 패킷의 처리를 완료하면 동작 요청을 수신한다. 예를 들어, 패킷 처리 엔진(3240)은 네트워크 패킷의 처리를 완료하고, 타이머(3242)의 다음 시간 간격을 대기하기 전이나 또는 네트워크 패킷을 처리하기 이전에 캐시에게 동작 수행을 요청한다. 다른 실시예에서, 패킷 처리 동작 중에 패킷 처리 엔진(3240)은 캐시 관리자(3232)에게 동작 요청을 전달한다. 또 다른 실시예에서, 캐시 관리자(3232)는 캐시 관리자(3232)가 동작을 수행하도록 트리거하는 신호를 예를 들어 패킷 처리 엔진(3240) 또는 패킷 처리 타이머(3242)로부터 수신한다. 일 실시예에서, 신호는 캐싱된 객체를 무효화시키거나 또는 캐싱된 객체를 만료시키라는 것을 나타낸다.

<549> 일 실시예에서, 캐시 관리자(3232)는 캐시 관리자(3232) 외부의 개체로부터 캐시 동작을 수행하라는 요청, 예를

들어, 서버(30)가 전달하여 패킷 처리 엔진(3240)이 처리하는 객체를 무효화시키라는 요청을 수신할 수도 있다. 일 실시예에서, 캐시 관리자(3232)는 객체 캐싱의 10ms 미만으로 무효화 요청을 수신하지만, 또 다른 실시예에서는 5ms, 2ms 또는 1ms 정도로 요청을 수신한다. 다른 실시예에서, 캐시 관리자(3232)는 캐시 관리자(3232)의 동작 또는 기능, 예를 들어, 객체가 무효화되게 하는 타이머의 만료에 따라, 또는 임의의 캐시 명령어의 처리 중에 캐시 동작을 수행할 수도 있다. 다른 실시예에서, 캐시 관리자(3232)는 디바이스(1250)의 패킷 처리 타이머(3242)를 이용하여 캐시 동작을 트리거한다. 예를 들어, 타이머(2342)는 캐시를 트리거 또는 시그널링하여 타이머(3242)가 설정할 수 있는 임의의 시간 간격에 캐싱된 객체의 무효화 또는 만료에 대해 검사할 수도 있다. 일 실시예에서, 타이머(3242)는 10ms 이하로 설정되게 또는 또 다른 실시예에서, 5ms, 2ms 또는 1ms로 설정되게 캐시를 트리거하거나 시그널링하도록 설정될 수도 있다. 일 실시예에서, 원래 서버(30)는 객체의 만료 시간을 설정할 수도 있다. 다른 실시예에서, 어플라이언스(1250) 또는 클라이언트(10)는 객체의 만료 시간을 설정할 수도 있다.

<550> 단계 3360에서, 캐시 관리자(3232)는 캐시에 저장된 객체의 무효화 또는 만료를 판단한다. 일 실시예에서, 캐시의 객체는 타이머의 만료에 따라서 무효화된다. 일 실시예에서, 캐시 관리자(3232)는 타이머의 만료를 기초로 하여 객체에 대한 무효화 명령을 내릴 수도 있다. 또 다른 실시예에서, 캐시에 저장된 객체는 타이머, 예를 들어 패킷 처리 타이머(3242)와 함께 설정된 타이머의 만료에 따라서 캐시 관리자(3232)가 자동으로 무효화시킨다. 일 실시예에서, 패킷 처리 타이머(3242)에 따라서, 캐시 관리자(3232)는 캐싱된 객체에 대해서 임의의 타이머의 만료를 검사한다. 일 실시예에서, 캐시 관리자(3232)는 객체 타이머가 만료했다고 판단하지만, 다른 실시예에서 캐시 관리자(3232)는 객체 타이머가 만료되지 않았다고 판단한다. 또 다른 실시예에서, 캐시 관리자(3232)는 패킷 처리 타이머(3242)의 제2 타이머 간격 또는 제2 트리거에 따라서, 이전에 검사한 객체 타이머가 만료했는지 두 번째 검사를 할 것이다.

<551> 일 실시예에서, 캐시 관리자(3232)는 무효화 명령 또는 요청을 파싱, 인터셉트, 액세스, 판독하고, 그렇지 않으면 처리하여 객체를 식별하여 캐시내에서 무효화시킨다. 일 실시예에서, 캐시 관리자(3232) 외부의 개체는 캐시 관리자(3232)에게 객체를 무효화시키기 위한 무효화 명령을 내린다. 또 다른 실시예에서, 외부 개체는 패킷 처리 타이머(3242)에 따라서 무효화 명령을 발행할 수도 있다. 객체가 유효하고/유효하거나 무효화되지 않았으면, 캐시 관리자(3232)는 요청에 따라서 객체를 무효화한다. 일 실시예에서, 캐시 관리자(3232)가 처리하는 무효화 요청은 패킷 처리 엔진(3240)의 패킷 처리 동작에 따르고 이어서 패킷 처리 타이머(3242)에 응할 수도 있다.

<552> 단계 3365에서, 캐시 관리자(3232)는 객체를 무효화로 표시한다. 캐시 관리자(3232)는 각 객체를 적합한 또는 원하는 방식으로 무효 표시할 수도 있다. 일 실시예에서, 객체는 저장된 객체의 특성, 속성 또는 플래그를 설정함으로써 무효로 표시된다. 예를 들어, 플래그는 객체가 무효임을 캐시 관리자(3232)에게 나타내는 값으로 설정될 수도 있다. 또 다른 실시예에서, 객체는 무효화 객체를 저장하기 위한 캐시의 영역이나 부분으로 객체를 이동시킴으로써 무효로 표시될 수도 있다. 다른 실시예에서, 캐시 관리자(3232)는 데이터베이스 또는 연결 리스트 또는 임의의 유형 및 형태의 데이터 구조로 저장된 객체의 무효 및/또는 유효 상태를 식별 또는 추적할 수 있다. 일 실시예에서, 캐시 관리자(3232)는 하나 이상의 객체들을 이용하여 캐시에 저장된 하나 이상의 객체들의 유효 또는 무효성을 식별 또는 추적할 수 있다. 또 다른 실시예에서, 객체는 저장된 객체를 변경, 수정 또는 개조, 예를 들어, 사용될 수 없는 객체의 일부를 삭제 또는 제거함으로써, 또는 객체의 명칭을 변경하거나 조립함으로써 무효로 표시된다.

<553> 단계 3370에서, 캐시 관리자(3232)는 일 실시예에서 캐시로부터 무효로 표시된 이들 객체를 삭제한다. 또 다른 실시예에서, 캐시 관리자(3232)는 예를 들어 클라이언트(10)에 의한 객체의 요청시 무효 객체를 캐시로부터 삭제한다. 일 실시예에서, 캐시 관리자(3232)는 객체의 무효화 또는 만료 이후에 수신한 갱신된 객체의 복사본 또는 버전으로 무효 객체를 겹쳐쓴다. 또 다른 실시예에서, 캐시 관리자(3232)는 캐시 메모리의 동일한 부분에 또 다른 객체를 저장함으로써 무효 객체가 차지하는 캐시 메모리를 재사용한다. 또 다른 실시예에서, 캐시 관리자(3232)는 무효 표시된 객체를 삭제하지 않고, 캐시의 메모리 또는 저장 장치에 저장된 객체를 유지한다.

<554> 방법(3350)은 무효화 유지 단위를 제공하기 위하여 패킷 처리 동작과 관련하여 및/또는 패킷 처리 타이머에 따라 캐싱된 객체를 무효화 및 삭제하는 것을 도시하지만, 무효화 유지 단위, 임의의 캐시 동작 및 임의의 캐시 관리 방법 및 본 명세서에서 기술한 디바이스(1250)의 임의의 다른 동작은 패킷 처리 타이머에 의해서 제공되는 보다 짧은 시간 간격으로 실행될 수도 있다. 일 실시예에서, 캐싱된 객체의 무효화 또는 만료는 100ms 시간 간격으로 발생할 수 있고, 또 다른 실시예에서는 50ms 시간 간격으로 발생할 수도 있다. 일 실시예에서, 캐싱된 객체의 무효화 또는 만료는 25ms 시간 간격으로 발생할 수 있고, 다른 실시예에서는 10ms 시간 간격으로 발생할

수도 있다. 다른 실시예에서, 캐싱된 객체의 무효화 또는 만료는 5ms 시간 간격으로 발생할 수 있고, 또 다른 실시예에서는 3, 2 또는 1ms 시간 간격으로도 발생할 수 있다.

<555> 위 도 28a 및 28b와 관련하여 방법(3300, 3350)에서 설명한 매우 작은 증분치의 시간 경과 이후, 객체를 무효화하는 기능을 포함함으로써, 동적 생성된 콘텐츠의 캐싱이 향상될 수 있다. 일부 동적 콘텐츠는 실제 매우 짧은 시간 동안 캐시에 저장되어 그로부터 제공되도록 개정될 수 있다. 그러나, 이러한 콘텐츠를 성공적으로 캐싱하기 위하여, 일 실시예에 따른 방법은 그 객체가 무효화되어 캐시 메모리로부터 삭제되기 이전에 매우 짧은 시간 동안 객체를 캐싱한다. 예를 들어, 어떤 동적으로 생성된 객체는 1초 동안 캐싱될 수 있지만, 더 길어지는 것은 지속적으로 변하는 콘텐츠에 대해서는 수락 가능하지 않을 수 있다. 일 실시예에서, 접근 방법은 제2의 짧은 시간 경과 이후 캐싱된 콘텐츠를 무효화 또는 만료시키는 것을 포함한다. 예로서, 애플리케이션이 동적 응답을 생성하는 데에 100 밀리초가 소요되는 경우, 캐시는 데이터의 최신성을 손상하지 않으며 100 밀리초 이하의 기간 동안 그 응답을 저장 및 제공할 수 있다. 신규의 객체를 생성하는 데에 소요되는 시간보다 짧으므로 100 밀리초 기간 동안 신규 생성되는 객체는 없을 것이다. 따라서, 어플라이언스(1250)는 그 주기 동안에 이전의 객체를 제공하도록 설치될 수 있다. 어플라이언스(1250)가 매우 짧은 시간 간격으로 무효화시킬 수 있는 능력은 데이터베이스 처리 격리 레벨이 반복 읽기 (Repeatable Reads) 또는 일렬 읽기(Serialized Reads)를 허용하게 설정된 어플라이언스 환경에서 매우 유용하다.

<556> c. 무효화 명령

<557> 종래의 캐싱 방법은 통상적으로 관리자에 의해서 구성되거나, 또는 그 객체를 제공한 서버로부터 수신되어 저장된 콘텐츠를 콘텐츠에 대해서 사전 정의된 만료 시간을 기초로 하여 무효화한다. 동적으로 생성된 콘텐츠를 보다 효율적으로 캐싱하기 위하여 콘텐츠를 무효화시키는 또 다른 방법을 아래에 설명한다. 이 방법은 캐시내에서 이전에 저장된 하나 이상의 객체를 실시간으로 무효로 식별하는 무효화 명령을 어플라이언스(1250)에서 수신하는 기능을 포함한다. 예를 들어, 무효화 명령은 클라이언트 또는 API(application programming interface)로 전송되는 네트워크 패킷을 통하여 전달될 수 있다. 이는 객체가 제공되는 시각에 서버가 객체 헤더에 포함하는 캐시 만료 시간을 간단히 설정하는 점이 전형적인 방법과 다르다.

<558> 이 방법은 도 29a 및 29b에 보다 상세히 도시되어 있다. 도 29a는 컴퓨터 메모리 캐시와 같은 캐시를 유지하는 방법을 설명하는 흐름선도이다. 간단히 살펴보면 단계 3410에서는, 이전에 원래 서버(30)로부터 제공된 동적 생성 객체는 캐시에 저장되어 있다. 예를 들어, 동적으로 생성된 객체는 캐싱될 수 있는 것으로 식별되지 않고, 또는 이와 다르게 임의의 캐시 또는 캐시 제어 정보를 포함한다. 단계 3420에서, 무효화 명령은 캐시 또는 캐시 관리자(3232)에서 수신된다. 무효화 명령은 이전에 제공된 하나 이상의 객체를 무효로 식별한다. 단계 3430에서 무효화 명령에 따라, 캐시 또는 캐시 관리자(3232)는 식별된 객체를 무효로 표시한다.

<559> 더욱 상세히 단계 3410에서, 캐시 관리자(3232)는 임의의 소오스로부터 전달된, 획득한 또는 수신한 동적 생성 객체를 캐시 메모리 소자에 저장한다. 일 실시예에서, 동적 생성 객체는 생성되어 서버(30)로부터 제공될 수도 있다. 다른 실시예에서, 동적 생성 객체는 클라이언트(10)에 의해서 생성 및 전달될 수도 있다. 일 실시예에서, 어플라이언스(1250)의 또 다른 부분, 컴포넌트 또는 프로세스는 객체를 생성하여 캐시에 그 객체를 저장한다. 다른 실시예에서, 동적으로 생성된 객체는 네트워크상의 또 다른 어플라이언스(1250) 또는 또 다른 컴퓨팅 디바이스에 의해 생성되어 어플라이언스(1250)에 전송 또는 전달될 수도 있다. 일 실시예에서, 동적으로 생성된 객체는 캐싱 가능한 것으로 식별되지 않거나 또는 캐싱 불가능한 것으로 식별된다. 다른 실시예에서, 동적으로 생성된 객체는 캐싱 가능한 것으로 식별되거나, 캐시 제어 하이다.

<560> 단계 3420에서, 캐시 관리자(3232)는 무효화시킬 객체, 예를 들어, 동적으로 생성되어 캐시에 저장된 객체를 식별하기 위한 무효화 명령을 수신한다. 일 실시예에서, 무효화 명령은 객체가 무효화 상태이거나, 또는 이와 다르게 쓸모없을 수 있음을 캐시에게 나타내는 임의의 유형의 명령 또는 명령어를 포함할 수도 있다. 일 실시예에서, 무효화 명령은 객체를 식별하고, 또한, 그 객체가 무효화 상태인 시점과 함께 객체의 어느 부분이 무효화될 수 있는지를 식별할 수 있다. 일 실시예에서, 캐시 관리자(3232)는 원래 서버(30)에 의해 원격으로 호출될 수도 있는 API를 제공한다. 일 실시예에서, 캐시 관리자(3232)는 명령어를 수신하고 하나 이상의 네트워크 패킷을 통해 명령어에 대한 응답을 하기 위한 임의의 유형 및 형태의 프로토콜을 제공한다. 일 실시예에서, 캐시 관리자(3232) 또는 디바이스(1250)는 무효화 명령을 수신 및 처리하기 위한 XML(Extensible Markup Language) API 인터페이스를 제공한다. 예를 들어, 캐시 관리자(3232)는 웹 서비스 인터페이스를 제공할 수도 있다. 일 실시예에서 캐시 관리자(3232)는 확인 응답, 상태 등의 응답을 원래 서버(30)로 전송함으로써 무효화 명령에 응답할 수도 있다. 다른 실시예에서, 캐시 관리자(3232)는 무효화 명령에 응답하지 않는다. 일 실시예에서, 원

래 서버(30)에서 실행 중인 애플리케이션이 그 저장된 객체를 쓸모없게 만든 동작을 수행했으면, 객체는 예를 들어 객체의 신규 또는 갱신 버전을 생성함으로써 무효로 표시된다. 이는 예를 들어 새로운 에디터가 신속하게 전개되는 신규 스토리로 변경을 하고, 따라서 그 스토리의 최신 버전이 클라이언트에게 제공되게 보장되기를 원하는 경우에 발생할 수 있다.

<561> 무효화 명령은 객체를 생성한 애플리케이션에 의해, 또 다른 서버(30) 또는 또 다른 어플라이언스(1250)에 의해 원래 서버로부터 발행될 수 있다. 일 실시예에서, 원래 서버(30)는 원래 서버(30) 상에서 동적으로 생성된 객체에 대한 변경에 대응하여 자동으로 캐시(3232)에 무효화 명령을 발행하거나 전달한다. 무효화 명령은 또한 서버(30) 및 어플라이언스(1250) 외부의 관리자 제어에 의해서 생성될 수 있다. 예를 들어, 관리자 제어는 네트워크 상에서 실행 중이며 어플라이언스(1250)와 통신하는 임의의 유형 및 형태의 프로그램 또는 애플리케이션, 예를 들어 관리자 콘솔일 수도 있다. 또한, 클라이언트(10)는 어플라이언스(1250) 또는 캐시 관리자(3232)에게 무효화 명령을 발행하거나 전달할 수 있다. 예를 들어, 원래 서버의 요청된 객체를 변경시키게 될 것이라는 것을 클라이언트(10)가 인식하는 동작을 클라이언트가 취하는 경우, 클라이언트는 무효화 명령을 전달할 수 있다. 캐시에 저장된 임의의 객체는 캐시에서 로컬로 수행되거나 또는 XML API 인프라스트럭처를 이용하여 원격으로 호출되는 사용자 명령에 따라 캐시로 전달함으로써 무효화될 수 있다.

<562> 단계 3430에 따라, 무효화 명령에 따라서 무효로 식별되었던 캐시에 저장되어 있는 객체, 예를 들어, 이전에 제공되었던 동적으로 생성된 객체는 무효로 표시된다. 무효화 객체는 캐시로부터 요청 클라이언트에게로 제공되지 않지만, 원래 서버로부터는 직접 제공될 것이다. 캐시 관리자(3232)는 임의의 적합한 또는 원하는 방식으로 각 객체를 무효로 표시할 수 있다. 일 실시예에서, 저장된 객체의 특성, 속성 또는 플래그를 설정하는 것에 의해 객체를 무효로 표시한다. 예를 들어, 플래그는 객체가 무효임을 캐시 관리자(3232)에게 나타내는 값으로 설정될 수도 있다. 또 다른 실시예에서, 객체는 무효화 객체를 저장하기 위한 캐시의 영역이나 부분으로 객체를 이동시킴으로써 무효로 표시될 수도 있다. 다른 실시예에서, 캐시 관리자(3232)는 데이터베이스 또는 연결 리스트 또는 임의의 유형 및 형태의 데이터 구조로 저장된 객체의 무효 및/또는 유효 상태를 식별 또는 추적할 수 있다. 일 실시예에서, 캐시 관리자(3232)는 하나 이상의 객체를 이용하여 캐시에 저장된 하나 이상의 객체의 유효성 또는 무효성을 식별 또는 추적한다. 또 다른 실시예에서, 객체는 저장된 객체를 변경, 수정 또는 개조, 예를 들어, 사용될 수 없는 객체의 일부를 삭제 또는 제거함으로써, 또는 객체의 명칭을 변경하거나 조립함으로써 무효로 표시된다.

<563> 일 실시예에서 어플라이언스(1250)는 캐시로부터 무효로 표시된 이들 객체를 삭제한다. 또 다른 실시예에서, 어플라이언스(1250)는 예를 들어 클라이언트(10)에 의한 객체의 요청시 무효 객체를 캐시로부터 삭제한다. 일 실시예에서, 어플라이언스(1250)는 객체의 무효화 또는 만료 이후에 수신한 갱신된 객체의 복사본 또는 버전으로 무효 객체를 겹쳐쓴다. 또 다른 실시예에서, 어플라이언스(1250)는 캐시 메모리의 동일한 부분에 또 다른 객체를 저장함으로써 무효 객체가 차지하는 캐시 메모리를 재사용한다.

<564> 캐시 관리자(3232)의 명령 무효화 API를 이용하여, 어플라이언스(1250)와 통신하는 임의의 컴퓨팅 디바이스 또는 사용자는 캐시에 저장되어 있는 객체, 예를 들어 동적으로 생성된 객체를 무효화시킬 것을 요청할 수 있다. 그러한 경우, 캐시에 저장된 객체의 무효화는 사전에 결정된 구성 만료 또는 무효화 기간을 이용하는 것 대신 실시간으로 제어될 수 있다. 따라서, 이러한 기술을 이용하면, 캐싱된 객체의 수명은 외부 애플리케이션 처리 노드, 예를 들어, 데이터베이스 또는 원시 애플리케이션 서버로부터 제어될 수 있다. 예를 들어, 어플라이언스(1250)는 데이터베이스와 작업하도록 구성되어, 데이터베이스가 변경되면 무효화 명령이 데이터베이스(또는 애플리케이션)로부터 어플라이언스(1250)로 자동 트리거되어 특정한 하나 이상의 객체를 삭제하게 된다.

<565> d. 무효화 명령을 이용한 그룹의 무효화

<566> 다른 실시예에서, 어플라이언스(1250)는 캐시가 저장하는 객체의 그룹을 동시에 식별하여 무효화한다. 전통적인 캐시 메모리에 저장된 객체들은 그 객체가 쓸모없는 것인식을 판단하는 데 있어서 캐시에 의해 개별적으로 그리고 별도로 각각 처리된다. 각 객체가 그 지정된 만료 시간(일반적으로 서버에 의해서 설정되어 캐시에 의해 테이블로 저장됨)에 도달하면, 그 아이템은 캐시 메모리로부터 삭제된다. 이러한 전형적인 접근 방법은 비효율적이며 결과적으로 불충분하지만 동적으로 생성된 콘텐츠를 캐싱하는 데에 시도하여 발생하는 문제를 성공적으로 처리하기 위한 것이다.

<567> 도 29b는 컴퓨터 메모리 캐시와 같은 캐시를 유지하기 위한 방법의 또 다른 실시예 - 여기서, 어플라이언스(1250)는 원래 서버(30)로부터 이전에 제공되었던 관련 객체들의 그룹을 생성, 저장 및 무효화하는 기능을 가짐 - 를 도시한다. 간단히 살펴보면, 단계 3410에서, 원래 서버(30)로부터 제공된 객체, 예를 들어, 동적 생성된

객체는 캐시에 저장된다. 단계 3412에서, 캐시 관리자(3232)는 캐시에 저장되어 있는 이전에 제공된 객체들의 그룹을 형성한다. 일 실시예에서, 그룹은 후술하는 바와 같이 하나 이상의 객체 결정자들과 연관 또는 이들에 의해서 식별될 수도 있다. 단계 3414에서, 캐시 관리자(3232)는 객체들의 그룹의 레코드를 유지한다. 단계 3422에서, 캐시 관리자는 객체들의 그룹을 무효화시키기 위한 무효화 명령을 수신한다. 단계 3432에서, 캐시 관리자(3232)는 무효화 명령에 따라서 객체들의 그룹을 무효로 표시한다.

<568> 단계 3410는 도 29a — 원래 서버(30)로부터 이전에 제공된 객체, 예를 들어 동적 생성 객체가 어플라이언스(1250)의 캐시에 저장됨 — 와 동일하다. 일 실시예에서, 객체들 중 하나 이상이 캐싱 가능한 것으로 식별되지 않을 수도 있고, 또는 임의의 캐시 또는 캐시 제어 정보를 가지지 않을 수도 있다. 예를 들어, 서버(30)는 동적 생성 객체가 캐싱되지 않는다고 가정할 수 있다.

<569> 단계 3412에 따라, 어플라이언스(1250)는 원래 서버(30)로부터 이전에 제공되어 캐시에 저장된 객체들의 세트 중에서 일 그룹을 형성한다. 임의의 적합한 또는 원하는 객체들의 세트는 서로 연관되어 그룹을 형성할 수 있다. 예를 들어, 웹 페이지 제공을 위하여 생성되거나 이와 연관된 임의의 동적 생성 객체가 일 그룹을 형성할 수도 있다. 일 실시예에서, 객체는 다수의 그룹과 연관될 수도 있다. 다른 실시예에서, 객체들의 일 그룹은 또 다른 객체 그룹의 부분집합을 이룰 수도 있다. 일 실시예에서, 형성된 객체 그룹은 동일 서버(30)로부터 제공된 객체를 갖지만, 다른 실시예에서는 형성된 객체 그룹이 그와 다른 서버(30)로부터 제공된 객체들을 가질 수도 있다. 다른 실시예에서, 형성된 객체 그룹은 클라이언트(10)로부터의 객체, 서버(30)로부터의 객체, 또는 클라이언트(10)와 서버(30) 둘 다에 의해서 생성되거나 이들로부터 제공된 객체들을 포함할 수도 있다. 일 실시예에서, 그룹내의 일 객체는 정적이지만, 그룹내 다른 객체는 동적으로 생성된다. 어떤 경우에, 그룹내 일 객체는 캐싱 불가능한 것으로 식별되지만, 그룹내 다른 객체는 캐싱 가능한 것으로 식별된다. 다른 경우에, 그룹내 객체에는 서버(30)가 제공하는 기능 또는 애플리케이션에 따라 논리적으로 관련될 수도 있다. 또 다른 경우에, 그룹내 객체는 동일한 클라이언트(10) 또는 동일한 사용자와 연관된 것으로 관련될 수 있다.

<570> 단계 3414에서, 객체 그룹의 레코드가 유지된다. 본 명세서에서 설명하는 동작의 일부 실시예들을 실시하는 데에 객체 그룹의 레코드를 기록 및 유지, 또는 객체들을 연관시키는 다양한 방법이 이용될 수 있다. 일 실시예에서, 레코드는 예를 들어 특정 테이블에 직접적으로 유지될 수도 있다. 또 다른 실시예에서, 레코드는 해시 테이블 형태로 표시될 수 있다. 일 실시예에서, 캐시 관리자(3232)는 객체의 연관 관계 데이터베이스, 또는 데이터 구조 또는 메모리내 객체에 유지한다. 또 다른 실시예에서, 그룹내 각 객체의 플래그, 특성 또는 속성은 그 그룹을 식별하는 값, 예를 들어, 아래에 보다 상세히 설명하는 그룹의 객체 결정자와 같이 그룹의 명칭 또는 식별자를 식별하거나 가리키는 값으로 지정 또는 설정된다. 일 실시예에서, 객체의 그룹은 그룹을 유지하는 것으로 식별된 캐시 메모리부에 배치, 위치 설정 또는 위치한다.

<571> 단계 3422에서, 어플라이언스(1250) 또는 캐시 관리자(3232)에서 무효화 명령어를 수신한다. 도 29b에 도시하는 실시예에 따라, 무효화 명령어는 하나 이상의 객체가 무효이거나 쓸모없음을 확인한다. 일 실시예에서, 무효화 명령어는 객체 그룹의 명칭 또는 식별자를 식별 또는 지정한다. 일 실시예에서, 무효화 명령어는 그룹내 모든 객체들을 무효화하기 위한 단일의 무효화 요청을 포함한다. 또 다른 실시예에서, 무효화 명령어는 무효화할 일 객체를 그룹내에서 식별한다. 다른 실시예에서, 무효화 명령어는 그룹내의 복수의 객체들을 무효화하기 위한 복수의 무효화 요청을 포함한다.

<572> 단계 3432에 따라, 이전에 제공된 객체의 그룹은, 무효화 명령이 그룹 중의 일 객체를 무효로, 그룹내 각 객체가 무효로 또는 그 그룹이 무효로 식별하면 무효로 표시된다. 일 실시예에서, 무효화 명령어가 그룹내 객체를 무효로 식별하면, 캐시 관리자(3232)는 객체를 무효로 표시한다. 다른 실시예에서, 무효화 명령어가 그룹내 객체를 무효로 식별하면, 캐시 관리자(3232)는 그 객체 그룹을 무효로 표시하거나, 그룹내 각 객체를 무효로 표시한다. 또 다른 실시예에서, 캐시 관리자(3232)는 하나 이상의 무효화 명령을 통해 복수의 객체가 무효로 식별되는 경우에만 그 객체들 그룹을 무효화할 수도 있다. 또 다른 실시예에서, 무효화 명령어는 그룹의 명칭 또는 식별자를 지정할 수도 있고, 캐시 관리자(3232)는 그 그룹을 무효로 표시하거나 그룹내 각 객체를 무효로 표시한다.

<573> 일 실시예에서, 어플라이언스(1250) 또는 캐시 관리자(3232)는 무효로 표시된, 객체의 그룹을 캐시 메모리로부터 삭제한다. 일 실시예에서, 그룹내에서 각 객체가 무효로 표시되는 경우에만 그룹내 객체가 캐시 메모리로부터 삭제될 수도 있다. 다른 실시예에서, 그룹의 일 객체가 무효로 표시되면, 전체 그룹이 삭제된다. 또 다른 실시예에서, 클라이언트(10)가 객체들의 그룹 또는 그룹내 임의의 객체에 대한 요청을 수신하면 무효로 표시된 객체들의 그룹 또는 그룹내 임의의 객체가 삭제될 수도 있다. 다른 실시예에서, 서버(30)로부터의 응답 수신에

의해 그룹내에 하나 이상의 신규 객체가 제공되면, 무효로 표시된 객체들의 그룹 또는 그 그룹내 임의의 객체가 삭제될 수도 있다.

<574> 위에서 설명한 실시예는 다음과 같다. CRM(Customer resource management) 애플리케이션은 자원 관리의 모든 측면들을 추적 및 평가하기 위하여 다수의 비즈니스에 의해 이용된다. 종종, CRM 애플리케이션은 인터넷을 포함하는 공용 및 사설 네트워크 상에서 구현되어 액세스된다. 자주 액세스되는 대량의 데이터에 대해서 액세스를 제공하는 이들 애플리케이션은 따라서 이러한 애플리케이션이 생성한 데이터를 캐싱하는 것에서 이익을 얻는다. 예를 들어, 판매 보고서는 생성되어 원격 연결 사용자에게 제공되기도 한다. 이들 판매 보고서는 적절한 애플리케이션이 그 애플리케이션 서버 및/또는 내장 데이터베이스에 포스팅되는 판매 정보로부터 데이터를 수집함으로써 형성된다. 다수의 사용자가 캐싱을 하지 않고 동일한 문서(즉, 특정한 판매 보고서)를 요청하므로, 애플리케이션 서버는 각 요청마다 객체를 재생성해야 한다. 그러나, 이러한 객체가 캐시에 저장될 수 있으면, 값비싼 전송 용량(bandwidth) 등의 애플리케이션 및 데이터베이스 처리가 절약될 수 있는데, 그 이유는 캐시가 요청 클라이언트에게 보다 가까이 위치하기 때문이다.

<575> 이러한 객체들을 캐싱하는 데에는, 신규의 판매 정보가 원래 서버에서 실행 중인 애플리케이션(또는 그 데이터베이스)에 포스팅되어, 판매 보고서내의 정보가 갱신될 필요가 있으므로 문제가 발생한다. 그 결과, 이들 애플리케이션 서버를 지원하는 임의의 캐시에 저장되었던 모든 판매 보고서는 무효화되어야 하고 그 콘텐츠는 캐시 메모리에서 삭제되어야 한다. 그러나, 기존의 캐싱 방법은 데이터베이스 또는 애플리케이션에 대한 변경이 언제 발생할 것인지를 정확히 판단할 방법이 없어서 동적 콘텐츠의 갱신 여부를 올바르게 평가할 수 없다. 데이터베이스나 애플리케이션 또는 원래 서버에서 변경이 발생할 때마다 캐시는 변경이 이루어졌음을 그리고, 이러한 변경의 결과로서 어떤 그룹의 객체들이 무효화되어야 하는지를 식별할 수 있어야 한다. 위에서 설명한 바와 같이, 이전에 제공된 객체의 그룹으로 연결되는 객체 결정자 포함 무효화 명령어를 생성함으로써 이러한 필요성을 충족시킬 수 있다.

<576> 관련된 객체들의 다수의 그룹이 단일의 계층 레벨에서 형성될 수도 있다. 이와 다르게, 객체들의 부 그룹들이 형성되어 다중 계층 레벨을 생성할 수도 있다. 일 실시예에서, 객체들의 그룹이나 부 그룹들이 사용자에게 의해서 미리 설계될 수도 있다. 또 다른 실시예에서, 사용자는, 어플라이언스(1250)가 관련된 객체들의 그룹들을 자동으로 형성하고, 그 그룹들과 객체 결정자들을 연관시키는 규칙을 설정할 수도 있다.

<577> e. 클라이언트 요청 또는 응답시 객체 결정자의 식별

<578> 일 실시예는, 원시 애플리케이션 서버(30)(및/또는 그 데이터베이스)에서의 상태 변경에 의해 영향을 받는 모든 객체들을 식별할 수 있어야 한다는 문제를 객체들 그룹을 생성하고 무효화를 파라미터로 나타냄으로써 해결한다. 이러한 실시예에서, 임의의 객체 또는 객체들의 사전 정의된 그룹은 객체 결정자를 식별하기 위하여 캐시가 파싱하는 예를 들어 클라이언트로부터 인터셉트하여진 HTTP 요청에 의하여 무효화될 수 있다. 용어 "객체 결정자"란 일 객체 또는 객체들의 세트를 고유하게 가리키거나, 식별하거나 지정하는 임의의 정보, 데이터, 데이터 구조, 파라미터, 값, 데이터 패턴, 요청, 응답 또는 명령어를 지칭한다. 일 실시예에서, 객체 결정자는, 객체와 연관될 수 있는 통신의 또는 그 통신이 그 오브젝트와 연관되어 있거나 참조하고 있음을 고유하게 식별하는 데에 사용되는 바이트 또는 캐릭터의 패턴이다. 일 실시예에서, 객체 결정자는 객체 결정자가 연관되어 있는 캐시 관리자(3232)에 저장되어 있는 이전에 제공된 객체들의 그룹에게로 원래 서버에서 변경이 발생했는지 또는 발생할 것인지를 나타낸다. 일 실시예에서, 객체들 그룹내의 객체는 그들이 적어도 하나의 개체 결정자와 연관되어 있다는 점에서 관련되어 있다. 특히, 객체 결정자의 예들과 이들 사용에 대한 추가의 설명은 아래에서 보다 상세히 설명한다.

<579> 본 실시예의 일부 실시예에서, 객체 결정자는 클라이언트 요청 또는 응답에 포함 또는 내장되는 특정한 사전 정의된 파라미터 또는 데이터 구조이다. 다른 실시예에서, 클라이언트(10), 서버(30) 또는 어플라이언스(1250)는 객체 결정자를 나타내는 사전 정의된 캐릭터 세트 또는 스트링과 같은 객체 결정자를 통신에 하나 이상 포함시킨다. 객체 결정자는 이러한 요청이 원래 서버(30) 또는 그에 연결된 데이터베이스에 저장되어 있는 객체의 상태를 변경시키는 결과를 가질지를 나타낸다. 일 실시예에서, 요청시 객체 결정자가 존재한다는 것은, 객체에 대한 변경을 갖거나 변경이 발생할 것이라는 것을 나타낸다. 또 다른 실시예에서, 객체 결정자의 신택스, 구조, 파라미터 또는 값이 객체에 대한 변경을 갖거나 변경이 발생할 것이라는 것을 나타낸다. 일 실시예에서, 캐시는 클라이언트(10)로부터 객체 요청을 수신한다. 요청은 캐시가 인식하는 특정 파라미터 또는 값(객체 결정자)이 원래 서버 또는 애플리케이션 서버의 상태를 변경하고, 그 결과 이러한 원래 서버 또는 애플리케이션 서버(30)에 의해서 이전에 생성되었고 캐시 관리자(3232)에 의해 저장된 관련 특정 객체들을 쓸모없게 만들 것

인지를 포함할 수도 있다. 사용자에게 의해서 설정되는 무효화 정책에 따라, 파라미터(객체 결정자)는 캐시에 저장된, 원래 서버에 의해서 이전에 제공된 객체들 또는 객체들의 그룹의 하나 이상을 무효화할 것을 요구할 수도 있다. 캐시는 이러한 상태 변경에 의해서 영향을 받을 적절한 객체들(즉, 객체 결정자로 연결되는 객체들 또는 객체들 그룹)을 식별하고, 이 객체들 각각을 무효로 표시 및/또는 이러한 객체들을 캐시 메모리로부터 삭제하는 방법을 통해 이들 객체를 무효화하도록 구성된다.

<580> 상술한 방법은 도 29c에 도시되어 있다. 본 명세서에서 설명하는 다른 실시예들과 마찬가지로 단계 3410은 원래 서버로부터 이전에 제공된 동적 생성 객체들과 같은 객체를 캐시에 저장하는 것을 포함한다. 객체들은 원래 서버(30)상에서 실행 중인 애플리케이션에 의해서 생성될 수도 있고, 또는 원래 서버(30)에 의해서 액세스되는 데이터베이스로부터 도출될 수 있다. 일 실시예에서, 동적 생성 객체는 캐싱이 가능하지 않은 것으로 식별되거나, 캐싱 가능한 것으로 식별되지 않는다.

<581> 단계 3421에서, 캐시는 클라이언트로부터의 요청 또는 서버로부터의 응답과 같은 클라이언트와 서버간의 통신을 수신한다. 일 실시예에서, 요청은 지정된 객체에 대한 것이며, 이 객체는 이전에 제공된 것으로 캐시에 저장되어 있는 것이다. 또 다른 실시예에서, 통신은 요청 객체를 가지고 있다는 서버로부터의 응답을 포함한다. 일 실시예에서, 이러한 수신 또는 인터셉트는 설정된 캐싱 프로토콜 및 통신 표준에 따라서 발생한다. 일반적으로 캐시 관리자(3232) 또는 어플라이언스(1250)가 요청, 응답 또는 통신을 수신한다고 설명될 수 있으나, 이러한 요청, 응답 또는 통신을 수신하는데 있어서 캐시(3232) 또는 어플라이언스(1250)는 임의의 적합한 수단 및/또는 메커니즘에 의해서 캐시에 직접적으로 또는 명시적으로는 전달되지 않는 요청, 응답 또는 통신을 인터셉트 또는 획득할 수도 있다.

<582> 단계 3423에서, 객체 결정자가 인터셉트한 통신에서 식별된다. 캐시 관리자(3232)는 그 통신에서 하나 이상의 객체 결정자를 판단 또는 식별하기 위하여 인터셉트한 통신을 추출, 번역, 파싱, 액세스, 관독 또는 처리할 수 있다. 통신의 하나 이상의 특성의 임의의 세트, 파라미터, 값, 선택스, 데이터 또는 구조가 객체 결정자를 식별하는 데에 이용될 수도 있다. 일 실시예에서, 캐시 관리자(3232)는 클라이언트(10)로부터 서버(30)로의 요청 — 클라이트가 객체를 요청 — 에서 객체의 명칭 또는 식별자를 식별할 수도 있다. 또 다른 실시예에서, 캐시 관리자(3232)는 캐시에 저장되어 있는 제2 객체에 변경이 발생했거나 발생할 것이라는 것을 나타내는 클라이언트(10)의 요청 또는 서버(30)로부터의 응답에서 제1 객체의 명칭 또는 식별자를 식별할 수도 있다. 다른 실시예에서, 캐시 관리자(3232)는 요청에서 임의의 캐리더 패턴이 캐시내의 일 객체 또는 객체들의 그룹과 연관된 임의의 객체 결정자와 일치하는 지를 판단한다. 일 실시예에서, 객체 결정자는 캐시에 현재 저장되어 있지 않는 객체에 대한 것일 수도 있다. 다른 실시예에서, 객체 결정자는 현재 무효로 표시된 객체에 대한 것일 수도 있다. 다른 실시예에서, 요청된 객체에 대한 객체 결정자는 캐싱된 객체의 객체 결정자와 연관된 것으로 판단된다. 또 다른 실시예에서, 통신에서 객체에 대한 제1 참조, 요청 또는 응답시에, 캐시 관리자(3232)는 그 식별된 객체 결정자를 그 객체에 대한 객체 결정자로서 설정한다.

<583> 객체 결정자를 식별하기 위해 클라이언트 요청 또는 서버 응답과 같은 통신을 수신 및 파싱함으로써, 캐시 관리자(3232) 또는 어플라이언스(1250)는 식별된 객체 결정자와 연관되어 있는 캐싱 객체를 무효로 표시하는 지를 효과적으로 판단할 수 있다. 따라서, 단계 3425에 따라, 객체 결정자가 캐싱된 오브젝트에 대한 변경을 나타내는 지에 대해 판단을 한다. 일 실시예에서, 식별된 객체 결정자는 객체를 개조, 수정 또는 생성하지 않는 통신의 요소일 수도 있다. 다른 실시예에서, 식별된 객체 결정자는 그 객체 결정자와 연관된 객체에 변경이 발생했거나, 발생할 것이라는 것을 나타내는 통신의 요소이다. 예를 들어, 통신은 동적 생성 객체에 대한 획득 요청이거나, 또는 하나 이상의 동적 생성 객체에 이용되는 데이터를 변경할 제안 요청일 수도 있다. 일 실시예에서, 통신에서 객체 결정자의 존재는 하나 이상의 객체에 대해 변경이 발생했거나 발생할 것이라는 것을 나타낸다. 또 다른 실시예에서, 객체 결정자와 함께 통신에서의 명령어, 지령 또는 인스트럭션의 유형 또는 명령이 하나 이상의 객체에 변경이 발생했거나 발생할 것이라 것을 나타낸다. 또 다른 실시예에서, 명령어, 지령 또는 인스트럭션의 파라미터나 변수의 존재, 값 또는 설정치가 객체 결정자와 연관된 하나 이상의 객체에 변경이 발생했거나 발생할 것이라는 것을 나타낸다.

<584> 다른 실시예에서, 캐시 관리자(3232)는 객체에서 변경이 발생했는 지를 판단하기 위하여 객체 결정자 또는 인터셉트한 통신에 대한 해시 함수, 알고리즘 또는 연산을 수행한다. 일 실시예에서, 그 객체에 대한 해시 값을 이전에 저장된 해시값과 비교하여 상이하면, 캐시 관리자(3232)는 그 객체가 변경되었다고 인식한다. 또 다른 실시예에서, 객체에 대한 해시값은 통신 또는 객체 결정자에 포함될 수도 있다. 일 실시예에서, 통신은, 불 플래그(Boolean flag)와 같은 파라미터 또는 설정치에 의하여 그 객체가 변경되었음을 나타낸다. 다른 실시예에서, 객체를 식별하여 그 객체가 변경되었는 지를 판단하기 위하여 전체 태그 제어 및 유효화 메커니즘(아래에서

보다 상세히 설명함)이 사용된다.

- <585> 변경이 표시되면, 이어서 단계 3431에서 객체 결정자에 의해 식별된 또는 그와 연관된 객체를 무효로 표시한다. 일 실시예에서, 인터셉트된 통신이 요청하는 객체를 단계 3431에 따라 무효로 표시하고, 단계 3440에 따라서 원래 서버(30)로부터 검색한다. 변경이 표시되지 않으면, 다른 실시예에서 요청된 객체를 단계 3450에 따라 캐시로부터 검색한다. 일 실시예에서, 무효로 표시된 모든 객체가 캐시로부터 삭제될 것이다.
- <586> f. 객체 결정자를 토대로 한 객체들의 그룹의 무효화
- <587> 위의 실시예가 클라이언트 요청시 객체 결정자의 식별을 토대로 하여 캐시 관리자(3232)에서 이전에 제공된 객체를 무효화하는 경우를 설명한다. 또 다른 실시예에서는 하나 이상의 객체 결정자가 연관되어 있는 객체들 그룹을 식별하여 무효화시키기 위해 이러한 일반적인 개념을 이용할 수도 있다. 이 실시예를 도 29d에 도시한다.
- <588> 도 29d에 도시하는 방법은 도 29c의 방법과 동일한 방식으로 시작된다. 단계 3410는 원래 서버로부터 이전에 제공된 동적 생성 객체와 같은 객체들을 캐시에 저장하는 것을 포함한다. 일 실시예에서, 하나 이상의 객체들이 캐싱 불가능한 것으로 식별된다. 도 29b와 유사하게 단계 3412에 따라, 이전에 제공된 객체들이 그룹으로 형성된다. 일 실시예에서, 객체 결정자 기술에 따라, 일 객체 그룹이 적어도 하나의 객체 결정자와 연관되거나 또는 그에 의해서 식별된다. 아래에 보다 상세히 설명하는 바와 같이, 일 실시예에서 객체 결정자와 그룹과의 연관관계는 사용자 캐싱 정책, 예를 들어, 정책 엔진(3236)이 정의하거나, 제어하거나 또는 이용하는 정책의 특성 및 세부사항에 따라서 달라진다. 다른 실시예에서, 그룹의 하나 이상의 객체 결정자는 그룹내에 객체들의 하나 이상의 객체 결정자들을 포함한다. 또 다른 실시예에서, 그룹의 객체 결정자는 그 그룹내 객체의 객체 결정자들의 조합을 포함한다.
- <589> 단계 3414에 따라, 그룹의 레코드는 적절한 경우 그와 연관된 객체 결정자들과 함께 유지된다. 이 단계는 도 29b에서 도시하는 단계 3414와 유사하다. 일 실시예에서, 그룹의 레코드 및/또는 모든 객체 결정자들은 록업 테이블에 유지된다. 다른 실시예에서, 그룹의 레코드 및/또는 모든 객체 결정자는 해시 테이블 형태로 유지될 수도 있다. 해시 테이블은 그 알파벳과 숫자 시퀀스 사이에 넓은 공백을 가질 수도 있는 비연속적인 키(non-contiguous keys)를 효율적으로 저장할 수 있도록 설계된다. 또 다른 실시예에서, 인덱싱 시스템은 해시 테이블의 상부에 형성될 수 있다. 일 실시예에서, 캐시 관리자(3232)는 일 그룹의 객체와 하나 이상의 객체 결정자들의 연관관계를 메모리내의 객체나 데이터 구조, 또는 데이터베이스에 유지한다. 또 다른 실시예에서, 그룹내 각 객체의 플래그, 특성 또는 속성은 그룹의 명칭 또는 식별자, 또는 그룹의 객체 식별자를 식별하거나 참조하는 값으로 배정 또는 설정된다. 일 실시예에서, 객체들의 그룹은 그 그룹을 유지하는 것으로 확인되는 캐시 메모리부에 배치, 위치 설정 또는 위치된다. 또 다른 실시예에서, 하나 이상의 객체 결정자는 객체들의 그룹과 연관되어 저장된다.
- <590> 단계 3421 및 3423은 도 29c에 도시한 단계 3421 및 3423과 유사하다. 단계 3421에 따라, 캐시 관리자(3232) 또는 어플라이언스(1250)는 클라이언트(10)와 서버(30)간의 통신, 예를 들어, 이전에 제공되었고 캐시에 저장되어 있는 객체에 대한 클라이언트로부터의 요청을 가로채거나 수신한다. 일 실시예에서, 캐시 관리자(3232)는 클라이언트(10)로부터 서버(30)로의 요청을 인터셉트한다. 일 실시예에서, 그 요청은 캐시에 저장되어 있는 객체에 대한 것이다. 다른 실시예에서 그 요청은 캐시에 저장되어 있는 객체에 대한 변경을 유발할, 예를 들면, 객체가 동적으로 생성되게 하는 서버(30)로의 인스트럭션, 명령어 또는 지령이다. 또 다른 실시예에서, 캐시 관리자(3232)는 캐시에 저장되어 있는 객체를 식별하거나 포함하는, 서버(30)로부터 클라이언트(10)로의 응답을 인터셉트한다.
- <591> 단계 3423에서, 인터셉트된 통신에서 객체 결정자를 식별한다. 위에 명시한 바와 같이, 객체 결정자는 원래 서버(30)에서 요청된 객체에 변경이 발생했거나 발생할 것인지를 나타낸다. 그러나, 도 29d의 실시예에서, 객체 결정자는 객체 그룹과 연관되어 있을 수도 있다. 이는 일 특정 객체 결정자에 의해서 영향을 받을 수 있는 캐시에 저장된 모든 개체들을 효율적으로 무효화시킬 수 있게 한다. 일 실시예에서, 일 객체의 객체 결정자가 식별된다. 다른 실시예에서는, 객체 그룹에 대한 객체 결정자, 예를 들어, 일 그룹 객체 결정자가 식별된다. 또 다른 실시예에서는, 그룹내에 있는 하나 이상의 객체들의 객체 결정자들의 조합이 식별된다.
- <592> 따라서, 단계 3427에 따라, 객체 결정자가 이전에 제공된 객체들의 그룹이 변경되었음을 나타내는 지를 판단한다. 일 실시예에서는 인터셉트된 통신에 그 그룹의 객체 결정자가 존재한다는 것이 그룹내 객체들 중 하나 이상 또는 전체 객체에 변경이 발생했거나 발생할 것이라는 것을 나타낸다. 다른 실시예에서는, 인터셉트된 통신

에서 명령어, 지령 또는 인스트럭션의 명칭 또는 유형에 의해서 이러한 변경을 나타낸다. 또 다른 실시예에서는, 통신에서 임의의 파라미터나 변수의 존재, 값 또는 설정치가 이러한 변경을 나타낼 수도 있다.

<593> 단계 3427에서, 객체 결정자가 그룹내에서의 변경을 나타내면, 단계 3435에 따라 이전에 제공된 객체들의 그룹이 캐시내에서 무효로 표시된다. 일 실시예에서, 그룹내 객체들 중 하나 이상 또는 전체가 요청되고 단계 3440에 따라 원래 서버(30)로부터 검색된다. 단계 3427에서, 객체 결정자가 그룹내의 변경을 나타내지 않으면, 일 실시예에서, 단계 3450에 따라 인터셉트된 통신의 요소로서 이전에 제공되었고 캐시에 저장되어 있는 임의의 객체가 캐시 관리자(3232)로부터 검색된다. 일 실시예에서, 무효로 표시된 임의의 객체 또는 객체들의 그룹은 캐시 관리자(3232)에 의해서 캐시로부터 삭제될 수도 있다.

<594> g. 그룹의 지정

<595> 캐시 관리자는 특정 그룹으로 포함될 객체를 특정적으로 지정할 수도 있다. 객체가 캐시에 저장되어 있을 때마다, 관리자는 그 객체를 그 구성에 따라 구성된 또는 내재 그룹들 중 하나의 컴포넌트로 만들 수도 있다. 구성된 그룹은 관리자가 이전에 설정한 구성 또는 이와 다르게는 애플리케이션 동작 및 객체 무효화와 관련된 다른 데이터를 토대로 할 수 있다. 객체는 그 구성된 그룹이 동적인 경우 내재 그룹의 일부일 수도 있다. 내재 그룹내 객체는 중요한 무효화 파라미터의 값에 의해서 그룹화된다.

<596> 객체를 매우 융통성있게 그룹화할 수 있게 함으로써, 동적으로 생성된 콘텐츠를 효과적으로 캐싱하는 데에 필요한 무효화의 융통성 및 공동 작용 레벨을 달성할 수 있다. 캐시는 특정 객체 그룹을 동시에 무효화하고, 그에 따라 동적으로 생성된 콘텐츠의 무효화를 빈번하게 필요로 하는 것에 보다 잘 응할 수 있게 만든다. 캐시가 일 객체를 일 그룹으로 할당할 때마다, 그 그룹은 하나 이상의 객체 결정자를 그와 연관시키기 위하여 그 객체에 대해서 다수의 것들, 무효화 파라미터 및 히트 결정자 등을 결정한다.

<597> CRM(customer resource management) 예에서, 캐시 관리자가 각 그룹화를 미리 지정할 수도 있다. 예를 들어, 관리자는 각 판매 부서를 명칭에 의해서 그룹화하도록 캐시를 구성한다. 따라서, 관리자는 자동차 부서, 오토바이 부서 등을 지정할 수 있고, 캐시에 들어오는 요청에서 객체 결정자가 인식될 때마다, 캐시는 객체 결정자를 통해 해당 부서로 연결되는 지정 그룹에 저장된 객체들 모두를 무효화시킬 수 있다.

<598> h. 규칙 기반 그룹화

<599> 이와 다르게, 캐시 관리자는 특정 그룹 또는 그룹들에 포함할 객체를 캐시 어플라이언스가 실행시에 결정할 수 있게 하는 규칙을 설정할 수도 있다. 이러한 규칙 기반 그룹화는 캐시가 적절한 그룹을 생성하기 위해 이용하는 중요한 객체 결정자로 객체를 연결하는 설정 규칙에 의한 그룹 지정을 따를 수도 있다. 이러한 접근 방법의 예는 각 그룹에 넣을 객체를 인식하기 위해 캐시가 이용하는 규칙으로 캐시를 구성하는 것을 포함할 수 있다.

<600> 다시 CRM 예를 참조하면, 애플리케이션에 설치되는 판매 부서의 각 세분은 판매 부서 소유의 그룹으로 캐시가 인식해야 함을 규칙이 명시할 수 있다. 이러한 방식으로, 캐시 관리자가 각 그룹화를 세부적으로 식별해야 할 필요 없이 캐시가 그 적절한 규칙을 기반으로 하여 판단할 수 있게 하는 그룹화가 이루어질 수 있다. 이 기술은 그룹화를 지정하기 위한 보다 융통성 있고, 작업 집중적인 게 덜한 방식을 만든다. 캐시 관리자는 판매의 모든 세분 부서(즉, 판매\자동차, 판매\오토바이 등)가 캐시에 의해서 새로운 그룹을 생성해야 한다는 것을 명시한다. 자동차 판매 부서로부터의 요청이 처리되어 캐시를 통해서 애플리케이션에 의해 리턴되면, 캐시는 판매의 각 세부 그룹을 인식하고 사전 구성된 규칙을 기반으로 하여 각 세부 그룹에 대한 그룹화를 자동으로 생성할 수 있다.

<601> 규칙은 보고서/매출/자동차 또는 보고서/매출/오토바이 등 객체에 대한 신규 요청을 확인할 때마다 캐시에 의해 구현될 수도 있다. 그러면 이러한 처리는, 오토바이 판매 부서, 그 후 자전거 판매 부서 등이 판매 부서의 세부 그룹인 것을 표시할 것을 요청할 때 반복될 수 있는데, 그 이유는 캐시가 이들 세부 그룹을 인식하고 그들 각각에 대한 객체 그룹화를 설정하기 때문이다. 공지된 무효화 요청이 캐시로 들어와 이들 그룹 중 하나로 연결되면, 또는 클라이언트 요청에서 적절한 객체 결정자가 식별되면(예를 들어, 그 요청 파싱시에 오토바이 판매 부서 판매/오토바이로의 판매 보고서의 포스트에서 발견되면), 캐시는 오토바이 판매 부서 그룹화에서 캐싱된 모든 개체들을 무효화시키는 것을 인식한다.

<602> 이러한 방식으로, 애플리케이션이 제공하는 데이터에 변경이 발생했거나 발생할 것이라는 것을 캐시가 인식하면, 위의 기술에 의하여 그 캐시는 (캐시가 수신하는 요청된 콘텐츠가, 애플리케이션에서의 변경을 트리거할 것이라는 것을 캐시가 인식하기 때문에 또는 일부 외부 변경 발생으로 인하여) 그룹화 처리를 통해 어느 객체가 무효화될 필요가 있는 지를 신속하고 간단히 식별할 수 있게 된다. 이러한 방식으로, 애플리케이션 또

는 데이터베이스 상태의 변경으로 인해 더 이상 신규가 아닌 동적 생성 객체들을 대량으로 무효화시킬 수 있다.

- <603> 그 캐시 메모리에 동적 생성 콘텐츠를 저장하여 제공하는 캐시 기능은 시간에 걸쳐서 캐싱의 혜택을 가장 잘 누릴 객체 세트를 결정하기 위하여 요청 및 응답 패턴 트래픽을 검사하는 인텔리전트 통계 엔진을 이용하여 개선될 수도 있다. 이 엔진은 캐시 어플라이언스 자체내로 통합되거나 또는 별도의 컴퓨터에서 동적 캐싱 적합성을 판단하기 위해 더 조사할 객체들의 서브세트를 선택하기 위한 발견적 방법(heuristic)으로서 실행될 수 있다.
- <604> i. 객체 결정자의 다른 이용
- <605> 위에서 기술하는 바와 같이, 객체 결정자는 객체 결정자가 연관되어 있으며, 원래 서버에서 캐시에 저장되어 있으며 이전에 제공된 객체들의 그룹에 변경이 발생했는 지 또는 발생할 것인지를 나타내는 임의의 데이터 구조일 수도 있다. 객체 결정자는 요청에 포함된 사전 정의된 스트링 값을 기초로 설정될 수 있다. 예를 들어, 어떤 USERID를 이용하여 요청이 들어오면, USERID는 그 USERID로부터의 포스트 등의 요청이 들어올 때마다 무효화되어야 하는 캐시 메모리내의 객체 그룹으로 연결될 수 있다. 객체 결정자의 잠재 후보로서는 그 객체를 원래 제공했던 서버의 서비스 식별자 이용을 포함할 수도 있다. 서비스 식별자는 HTTP 요청에 표시된 서비스 IP 어드레스, TCP 포트 및 서비스 식별자를 포함한다.
- <606> 요청에 표시된 또 다른 잠재적 객체 결정자는 URL(uniform resource locator)이다. 정적 객체를 캐싱하는 경우, 요청 URL은 일반적으로 그 객체를 고유하게 식별하기에 충분하다. 그러나, 동적 생성 콘텐츠에 대한 요청에서, URL에 표시된 정보는 캐싱된 객체를 식별하기에 충분하지 않을 수도 있다. 따라서, 캐시는 HTTP 헤더, 쿠키 헤더 또는 다른 관계적 헤더에 포함된 객체 결정자를 찾기 위한 요청내의 다른 정보를 검사해야 한다. 캐시는 적절한 파라미터 정보의 서브세트를 URL 질의 스트링에서, POST 바디에서, 쿠키 헤더에서 또는 임의의 다른 요청 또는 응답 헤더에서 등 클라이언트 요청내의 다른 다양한 위치에서 추가로 찾을 수 있다.
- <607> 객체 결정자를 위하여 URL을 파싱하는 데 있어서 문제는 URL 및 다른 헤더들이 캐시의 결정에 무엇이 적절한 지에 대해 추가로 많은 정보를 포함할 수 있다는 것이다. 따라서, 캐시는 적합한 객체 결정자를 식별할 수 있는 상당히 많은 정보를 파싱할 수 있어야 한다. 게다가, 헤더내의 데이터는 임의 흐름선일 수도 있고, 따라서 그러한 데이터가 HTTP 헤더내로 배치되는 데에 기준이 되는 방식이 없으므로 이 스트링내에서 적절한 객체 결정자를 찾는 데에는 간단한 비교 방식으로 불충분할 수 있다.
- <608> 또 다른 실시예에서, 특정 객체 결정자를 캐시 메모리내에 저장되어 있는 적절한 객체 또는 객체들 그룹과 매핑시키기 위한 사전 구성된 정책이 없는 경우에도, 캐시가 그와 같이 판단할 것이다. 예를 들어, 캐시는 요청의 다양한 측면을 검사하고 파싱하여서 이러한 요청에서 임의의 다른 객체 결정자가 발견될 수 있는 지 그리고 이러한 요청을 캐시 메모리 내에 저장되어 있는 무효화되어야 하는 특정 객체로 연결시키는 데에 이용될 수 있는지를 확인할 수도 있다. 이와 다르게, 어떤 사전 정의된 발견적 방법을 기초로 하여 캐시가 결정하는 어떤 객체 결정자에 대한 요청이 특정한 객체 또는 객체들의 그룹으로 의미 있게 연결될 수 있다는 것을 캐시가 검사할 수 있게 한다. 예를 들어, 특정한 USERID와 연관되어 있는 캘린더의 업데이트에 대한 요청이 캐시로 들어오면, 일 실시예는 모든 캐싱된 객체들을 그 캘린더를 업데이트하는 요청의 USERID와 동일한 USERID를 이용하여 인식하고, 임의의 일 특정일에 대한 사용자의 캘린더(무효화될 필요가 있음)를 포함한다는 것을 인식하도록 설정될 수 있다.
- <609> 캐시는 또한 URL Stem에서, URL에 표시된 질의에서, POST 바디에서, 또는 쿠키 헤더에서 비지정된 순서로 name=value 또는 이와 유사한 쌍들의 그룹으로서 표시된다고 가정할 수도 있다. 일 실시예에서, 그 질의는 name=value 쌍의 리스트의 형태라 가정된다. 따라서 사용자는 어느 파라미터 명칭이 중요한지를 설정할 수 있다. 모든 캐싱된 객체는 먼저 액세스 URL을 이용하여 맞춰진다(keyed). URL은 /site/application/special/file.ext?p1=v1&p2=v2&p3=v3와 같을 수도 있다. /site/application/special/file.ext part는 URL stem이다. p1=v1&p2=v2&p3=v3는 URL 질의이며 파라미터-값 쌍들을 포함한다. 파라미터-값 쌍은 또한 POST 바디에서 또는 쿠키 헤더에서 표시될 수도 있다.
- <610> 일 실시예에서, 사용자 또는 관리자는 p1 및 p2가 무효화 파라미터 또는 객체 결정자가 될 것이라고 설정한다. 캐시는 그 후 짝을 이루는 p1 및 p2 값을 갖는 객체들을 자동으로 그룹화할 것이다. 이러한 그룹화를 구현하는 한 가지 방법은 p1 및 p2를 데이터베이스 테이블내 1차 키들로, 즉, 유효 상태를 판단하기 위해 캐시가 그 참조 방법을 알고 있는 테이블내에서 고유하게 식별될 수 있는 객체들로 매핑하는 것이다. 그 데이터베이스 테이블

에서 어떤 것을 업데이트하기 위하여, 캐시에 저장되어 있는 데이터는 더 이상 유효하지 않음을 반영하도록 캐시는 p1 및 p2에 대한 새로운 값을 지정할 것이고, 그 다음 번엔 이러한 콘텐츠를 제공하게 될이 새로운 값을 캐시가 인식하면, 그 메모리에 저장되어 있는 연결된 객체들을 무효화하는 것을 캐시가 알 것이다. 캐시가 이러한 요청을 받으면, 업데이트 요청을 보면, 캐시가 짝을 이루는 p1 및 p2값으로 그 그룹을 무효화시켜야 함을 인식하는데, 그 이유는 캐시가 본래의 데이터가 변경되고, 그에 따라 그 p1 및 p2 객체 결정자와 관련된 모든 객체들에 영향을 미칠 것이라는 것을 알기 때문이다.

<611> 관리자가 요청에 포함되는 객체 결정자로서 특정 파라미터를 사전에 구성하지 않은 보다 복잡한 경우, 캐시는 객체의 그룹을 무효화할 경우 식별을 돕기 위해 사용자가 구성한 정책을 전개하여 요청으로부터 적절한 객체 결정자를 추출할 수 있다. 객체 결정자 스트링은 그 후 캐시내에 저장된 객체들의 그룹의 위치를 찾고 이러한 객체들을 무효화하는 데에 이용된다. 이러한 객체 결정자는 중요 파라미터 값의 리스트를 캐시가 생성하도록 구성하는 데에 이용될 수 있다. 들어오는 기록-요청이 중요 파라미터값에 대해 짝을 이루는 값을 가지면, 그 파라미터 명칭에 연결된 객체들이 무효화되어야 한다. 이와 다르게, 사용자는 그 요청으로부터 객체 결정자 스트링을 추출할 수 있는 정책 프레임워크 동작을 지정할 수 있다. 객체 결정자 스트링은 기록-요청으로부터 추출과 짝을 이루는 결정자 스트링을 이용하여 모든 객체들이 무효화된다. 이와 다른 방식으로는, 요청이 캐시에 도착하고, 캐시가 그 요청 스트링이 무효화 정책과 일치하는 지를 판단한다. 무효화 정책은 콘텐츠 그룹이 무효화되어야 하는 객체들을 지정한다.

<612> 이와 다르게, 캐시는 클라이언트 요청에 나타날 수도 있는 임의의 다른 사용자 정보를 이용할 수 있다. 위에서 명시한 바와 같이, 인증 및 허가를 통합함으로써 캐시는 사용자 정보를 액세스할 수 있다. USERID 또는 GROUPID는 캐싱된 객체들의 그룹이 사용자 또는 사용자의 그룹에 연결되는 이벤트에서 결정자들 중 하나가 될 수 있다. 사용자 정보가 중요한 객체 결정자이기도 하지만, 사용자 정보가 HTTP 요청에서 나타나지 않는 경우도 있다. 다른 실시예에서, 동적 캐싱 측면이, 어떤 종류의 허가, 액세스 제어 및 심사(AAA) 인프라스트럭처를 수행하는 기능을 포함하는 다양한 다른 네트워킹 요소를 캐시에 통합하는 시스템 및 방법과 결합될 수 있다. 따라서, 애플리케이션에 의해서 생성된 데이터에 따르는 보안 레벨이 캐시로부터 제공되는 대신에 데이터에 적용된다. 이러한 기술 방법에 의해서 애플리케이션은 캐싱되지 않을 수도 있는 제어된 정보를 캐시가 액세스할 수 있게 한다.

<613> 이러한 방법은 HTTP 요청에서는 식별 가능한 사용자 정보를 포함하지 않지만, 통합형 캐싱 특권(Integrated Caching patent)에 기술된 AAA 접근 방법을 통하여 식별될 수 있는 사용자를 캐시가 식별할 수 있게 한다. 이러한 접근 방법은 AAA 처리에 의해 알 수 있는 허가 상태 정보를 검사함으로써 특정 요청에 대한 적절한 사용자를 캐시가 식별할 수 있게 한다. 다른 실시예에서, 이러한 통합은 캐시에 저장된 정보에 대해 보안 정책을 라이선스하여 캐시에 저장된 정보를 비허가된 사용자가 액세스하는 것을 방지하게 한다.

<614> 이러한 접근 방법에서 동적으로 생성된 데이터의 중요부는, 캐시가 클라이언트로부터의 적절한 요청에 응답할 수 있기 이전에 클라이언트가 이러한 데이터가 라이선스 및 허가되기를 요청할 것을 필요로 한다. 캐시는 액세스-제어되는 객체들을 애플리케이션이 캐싱할 수 있도록 인증된 사용자에게 의해서 그리고 이러한 보안이 달성될 수 있는 인증 및 허가 정보를 동적 캐싱 기술에 통합함으로써 요청이 인증되도록 하는 기능을 가져야만 한다. USERID 또는 GROUPID는 그 객체가 사용자 또는 사용자들의 그룹에 전유되는 경우 객체 결정자들 중 하나일 것이다. 따라서, 애플리케이션에 의해 생성된 데이터에 따르는 보안 레벨은 캐싱된 정보에도 적용된다. 이러한 기술 방법은 캐싱되지 않을 수도 있는 액세스가 제어된 정보를 애플리케이션이 캐싱할 수 있게 한다.

<615> 마지막으로, 캐시에 저장되어 있는 객체가 여전히 유효한 지를 판단하기 위하여 본래의 데이터베이스의 상태 등의 다른 시간 변화형 정보가 요청으로부터 파생되어 객체 결정자로서 이용될 수도 있다. 캐시는 이러한 외부 변수들에 민감하게 구성되는 객체들의 그룹에서 적절한 만료 동작을 구성하는 것에 의해 이러한 상태를 관리할 수 있다.

<616> 동적 콘텐츠가 캐시에 의해서 파생되어 해석되어야 한다는 사실로써 제기되는 문제를 해결하기 위하여, 일 실시예에 따른 캐시는 어느 파라미터가 캐시에 대한 적절한 객체 결정자인 지를 제한할 수 있다. 이러한 방식에서는, 적용 가능한 애플리케이션 서버로 이러한 요청을 전달하는 것이 아닌 캐시로부터 객체를 제공하는 성공 레이트가 감소될 수 있다. 예로서, 클라이언트로부터의 요청 질의는 도시와 주 파라미터를 둘 다 포함할 수 있다. 그러나, 캐시는 도시 값과 관련 없이 주어진 주에서 모든 클라이언트로부터 들어오는 질의를 나타내는 클라이언트로부터 들어오는 요청에 대한 응답을 제공할 수 있는 지를 인식하기 위하여, 콘텐츠를 저장하기 위한 애플리케이션의 요건에 따르도록 캐시가 구성될 수도 있다. 이를 위해서는, 도시 파라미터가 적절하지 않으며

이 사실을 캐시는 인식할 수 있다. 또 다른 실시예는 도시 파라미터가 그 주 파라미터에 대해서 특정되는 것과 무관하게 짝을 이루는 경우에 응답이 캐시로부터 제공될 수 있도록 캐시를 구성하는 것을 포함한다.

<617> 요약하면, 캐시는 일반화된 파라미터화 객체 매칭을 구현한다. 이러한 접근 방법에서, 캐시는 객체 결정자로서 유용할 것이며, 이러한 객체 결정자들이 인식되면 캐시가 그 객체 또는 객체들의 그룹이 신규로 남고 캐시로부터 제공될 수 있는 지를 평가하는 데에 그 존재를 (또는 반대로 이러한 결정자의 부재를) 이용할 수 있도록 특정한 객체로 연결되는 요청내의 정보 서브세트를 인식하도록 구성된다. 캐시는 요청된 데이터가 신규로 남는 지를 판단하기 위하여 구성된 파라미터들에 대해서 요청이 들어올 때마다 조사하며, 또한, 캐시가 캐시 메모리에 저장되어 있는 적당한 객체로 적절한 데이터를 매칭할 수 있게 하는 테이블을 유지한다.

<618> j. 구현 횃수(Incarnation Numbers)

<619> 또 다른 실시예에서, 캐시는 객체들의 그룹을 무효화시키기 위하여 구현 횃수를 이용할 수 있다. 캐시가 원래의 상태에서의 변경으로 인하여 객체들의 그룹의 각 상태를 한번에 변경할 필요가 있는 경우에, 이러한 무효화를 행하는 데에 있어서 구현 횃수가 단단한 방법을 제공한다. 각 객체를 식별하고 그 상태를 개별적으로 변경하는 것은 캐시에 저장되어 있는 데이터의 신규성을 보장하기 위한 접근 방법으로서 비효율적이고, 구현 횃수를 이용하는 것이 객체들의 그룹을 무효화하는 데에 훨씬 더 간단하고 효과적인 방법이다. 본 실시예는 그룹을 나타내는 데이터 구조를 각 객체가 가리키므로, 서버는 그 그룹에 대한 데이터 구조에서의 상태를 변경하는 명령어를 전송하기만 하는 방법을 설명한다. 캐싱된 객체에 대한 후속 요청이 클라이언트로부터 도착하면, 캐시는 그 상태가 변했는 지를 먼저 확인해야 한다. 그렇게 하기 위해서, 캐시는 그 그룹에 대해서 상태가 변경되었는 지를 참조하기 위해 데이터 구조를 찾아본다.

<620> 데이터 구조를 효과적으로 구현하기 위해서, 캐시는 상태 변경에 대해 찾아 볼 지를 판단할 수 있어야 한다. 그러므로, 캐시는 그 그룹에서의 상태 변경을 이미 확인했는 지 여부를 판단할 수 있어야 한다. 이것이 구현 횃수가 유용한 경우이다. 캐시는 동적 생성 객체들을 콘텐츠 그룹과 연관시킨다. 이들 콘텐츠 그룹 각각은 데이터 구조에 포함되는 "구현 횃수" 또는 특정한 인덱스 값을 이용한 해시 테이블 록업 처리에 의해 표시될 수 있다. 그 후, 캐시의 상태 변경을 야기하는 것으로 인식되는 클라이언트 요청을 수신할 때마다, 클라이언트는 적절한 파라미터 대하여 그 클라이언트 요청을 파싱하고, 인식된 객체 결정자들을 토대로 해시 록업을 행하고, 그 데이터 구조에서 인덱스 또는 구현 횃수를 1 증가시킨다. 지정된 그룹내에 저장된 객체가 클라이언트에 의해 요청될 때마다, 캐시는 객체에 관한 해시 알고리즘을 수행하고 이 결과 값과 이러한 콘텐츠 그룹에 대하여 데이터 구조에 저장되어 있는 원래 값을 비교한다. 저장된 값이 이러한 객체에 대해서 캐시가 계산한 값과 동일한 경우, 캐시는 그 콘텐츠가 신규한 것으로 남아 있으며 요청자에게 제공될 수 있을 것으로 인식한다. 캐시가 이러한 객체에 대해서 계산된 현재의 구현 횃수와, 이 콘텐츠 그룹에 대해 그 데이터 구조에 저장되어 있는 값이 다르다는 것을 검출하는 경우에, 캐시는 그 저장된 객체가 더 이상 신규하지 않다는 것을 인식한다. 그러면 캐시는 그 저장된 객체를 무효화시키고 애플리케이션 서버에게 요청을 전송한다. 응답이 되돌아오면, 캐시 어플라이언스는 캐시 메모리에 그 신규 응답치로 저장을 하고 이러한 응답을 다시 새로운 데이터 구조로 연결시킬 것이다. 그 후, 캐시가 그 그룹내의 객체에 대한 요청을 수신할 때마다, 캐시는 비교를 행하고 그 데이터 구조에 다른 변경이 이루어지지 않았다 판단하면, 캐시는 그 새로 저장된 객체를 제공할 수 있다.

<621> 이러한 방식으로 객체들의 그룹의 무효화를 이용하면, 캐시는 무효화를 보다 신속히 행할 수 있으며, 이 때 무효화되는 객체의 수와 상관없이 일정하게 무효화가 이루어질 수 있다. 이러한 무효화의 보다 신속하고도 보다 효율적인 처리를 통해, 이 기술 방법은 동적 생성 객체들을 캐시가 보다 효과적으로 처리할 수 있게 한다. 이 접근 방법은 이러한 데이터의 신속한 변경으로 인하여 무효 또는 쓸모없는 콘텐츠를 제공하지 않으며, 이와 동시에 애플리케이션의 앞에 배치되는 캐시 어플라이언스가 동적 생성 객체들을 보다 적극적으로 저장 및 제공할 수 있게 한다. 일 실시예는 자주 또는 예측 불가능하게 변경되는 데이터를 제공할 수 있고, 이에 따라 캐시의 성능을 향상시킬 수 있다. 캐시는 사용자 명령어를 이용하여, 또한 각종 웹 트래픽을 검사 및 그룹화함으로써, 캐시 메모리에 저장되어 있는 객체 및 객체들의 그룹을 무효화시킬 수도 있다.

<622> 2. 연결 풀링(Connection Pooling)

<623> 일 실시예에서, 네트워크 어플라이언스(1250)(본 명세서에서는 인터페이스 유닛(1250)이라고도 함)는 클라이언트로의 연결을 반복적으로 개설 및 폐쇄함으로써, 각 서버와의 하나 이상의 연결을 개설하고 인터넷을 통해서 클라이언트에 의해 반복되는 데이터 액세스를 가능하게 이들 연결을 유지함으로써 야기되는 서버(30)의 많은 처리 부하를 덜어준다. 이러한 기술 방법을 본 명세서에서는 "연결 풀링"이라 한다.

- <624> 이행을 위한 연결 폴링의 동작은 그 다음 도 30에 간단히 도시되어 있다. 도 30에서 클라이언트(10)가 인터페이스 유닛(1250)에 의해 도달된 서버 팜의 서버들 중 하나에 대한 액세스를 요청하면 이 처리가 시작된다. 단계 4302에 도시하는 바와 같이 인터페이스 유닛(1250)과 요청 클라이언트간에 연결이 개설되고, 인터페이스 유닛(1250)은 그 서버를 액세스하라는 클라이언트 요청을 수신한다. 단계 4304에 도시하는 바와 같이 인터페이스 유닛(1250)은 요청된 서버의 정체를 판단한다. 일 실시예에서, 이는 클라이언트 요청에 의해 지정된 목적지 네트워크 어드레스를 검사함으로써 이루어진다. 또 다른 실시예에서, 이는 클라이언트 요청에 의해 지정된 네트워크 어드레스 및 경로명을 검사함으로써 이루어진다.
- <625> 클라이언트 요청이 지정되어야 하는 서버(30)의 정체를 결정한 후에, 인터페이스 유닛(1250)은 단계 4306에 도시하는 바와 같이 서버로 비어있는 연결(즉, 이용중이 아닌 일 연결)이 이미 개설되어 있는 지를 판단한다. 이용 중이 아닌 연결이 개설되어 있으면, 처리는 단계 4310으로 다시 시작한다. 그러한 연결이 개설되어 있지 않으면, 인터페이스 유닛(1250)은 단계 4308에 도시하는 바와 같이 서버로 연결을 개설한다. 그 후, 인터페이스 유닛(1250)은 단계 4310에 도시하는 바와 같이, 그리고 아래에서 도 31과 관련하여 더 설명하는 바와 같이 그 클라이언트 요청을 해석하여 서버로 전달한다. 서버 처리 이후에, 인터페이스 유닛은 단계 4312에 도시하는 바와 같이 서버로부터 응답을 수신한다. 단계 4313에서 도시하는 바와 같이 그리고 아래에서 더 설명하는 바와 같이 서버 응답은 해석되어 그 요청 클라이언트로 전달된다. 마지막으로 인터페이스 유닛(1250)은 단계 4316에 도시하는 바와 같이 클라이언트와의 연결을 폐쇄한다. 그러나, 인터페이스 유닛(1250)과 서버간의 연결은 연결 해제되지 않는다. 서버와 개설된 연결을 유지하고 클라이언트와의 연결을 필요에 따라 개설 및 폐쇄함으로써 인터페이스 유닛(1250)은 인터넷 상에서 제공 클라이언트들과 관련된 연결 부하 문제의 거의 전부로부터 서버(30)가 자유롭게 한다.
- <626> 아래에 보다 상세히 설명하는 바와 같이, 일 실시예는 단계 4316에 관한 것이며, 여기서 인터페이스 유닛(1250)은 클라이언트(10)와의 연결을 폐쇄한다. 인터페이스 유닛(1250)이 클라이언트와의 연결을 폐쇄하게 하는 시나리오에는 여러가지가 있다. 예를 들어, 클라이언트는 FIN(finish) 명령어 또는 RST(reset) 명령어를 개시할 수도 있다. 이들 시나리오 둘 다에서, 인터페이스 유닛(1250)은 그것과 클라이언트간의 연결이 폐쇄되기 이전에 이들 명령어들 중 하나를 수신할 때까지 대기한다. 연결 폴링의 비효율성은 그 클라이언트가 연결을 이용하지 않거나 종료 상태이고 그 기간 동안 인터페이스 유닛(1250)으로의 이러한 정보에 의존하지 않는 경우에 발생한다. 인터페이스 유닛(1250)은 또 다른 클라이언트를 위해 이 연결을 재사용하기 위하여 클라이언트로부터의 명령어를 대기 중이므로, 그 연결은 불필요하게 연결되어 있다.
- <627> 아래에 보다 상세히 설명하는 바와 같이, HTTP(Hyper-Text Transfer Protocol) 1.1(디폴트) 그리고 HTTP 1.0(연결 : 킵-어라이브 기법을 이용하는 경우)는 클라이언트 및/또는 인터페이스 유닛(1250)이 요청에 대한 서버 응답을 수신한 이후에도 서버와 그 연결을 개설 상태로 유지할 수 있게 한다. 그러면, 클라이언트 및/또는 인터페이스 유닛(1250)은 그와 동시에 또는 상당한 시간(또는 "선택 시간(think time)") 이후 동일한 연결을 통해 다른 요청을 할 수도 있다. 클라이언트는, 클라이언트의 조작자가 브라우저 상에서 다른 링크를 클릭 등 하기로 결정하는 "선택 시간"에 있다. 이는, 서버가 그 연결을 통하여 임의의 요청을 처리하지 않더라도 연결이 서버에 의해 유지될 것을 초래한다. 여기서, 서버 관리자는 유휴 상태(idle) 또는 "선택 시간" 상태였던 연결이 연결 해제되는 킵-어라이브 타임아웃을 설정하여 서버 상에 너무 많은 동시 연결이 생기는 것을 방지하도록 강제될 수도 있다. 일 실시예는 그 클라이언트(10)가 "선택 중"이지만 클라이언트(10')에 의해서는 사용되도록 서버로의 연결을 허용한다. 물론, 클라이언트(10)가 서버 연결을 이용 중일 때에 클라이언트(10')가 요청을 하면, 클라이언트(10')는 다른 서버 연결을 이용해야만 한다. 그러나, 일 실시예에서 매우 적은 수의 연결이 초과되어 일반적인 경우로 옮겨가면 연결 폴링의 효율성이 달성된다. 일반적인 경우란, 'n'개의 클라이언트 연결이 'm' - 여기서 'n'은 'm'보다 큼 - 개의 서버 연결으로 정적 다중화될 수 있는 경우이다.
- <628> 도 31은 단계 4310 및 4313(도 30)에 도시하는 바와 같이 클라이언트 및 서버 요청을 해석하는 일 실시예의 동작을 설명하는 흐름도이다. 일 실시예에서, 메시지 트래픽은 TCP/IP 패킷, 해당 기술 분야에 공지되어 있는 프로토콜(protocol suite)의 형태이다. TCP/IP 프로토콜은 Telnet, FTP(File Transfer Protocol, 전자 메일 및 HTTP와 같은 다수의 애플리케이션을 지원한다. 이 실시예를 HTTP 프로토콜의 경우로 설명한다. 그러나, 본 명세서로부터 당업자라면 이러한 본 발명의 개념을 다른 TCP/IP 애플리케이션에도 동일하게 적용할 수 있다는 것이 자명할 것이다.
- <629> 각 TCP 패킷은 TCP 헤더 및 IP 헤더를 포함한다. IP 헤더는 32 비트 소스 IP 어드레스 및 32 비트 목적지 IP 어드레스를 포함한다. TCP 헤더는 16 비트 소스 포트 번호 및 16 비트 목적지 포트 번호를 포함한다. 소스 IP 어드레스 및 포트 번호는 소스 어드레스라 통칭하며, 패킷의 소스 인터페이스를 고유하게 식별한다. 이와 마찬가지로

가지로, 목적지 IP 어드레스 및 포트 번호는 목적지 네트워크 번호라 통칭하며, 그 패킷에 대한 목적지 인터페이스를 고유하게 식별한다. 패킷의 소스 및 목적지 어드레스는 연결을 고유하게 식별한다. TCP 헤더는 또한 32 비트 시퀀스 번호 및 32 비트 확인 응답 번호를 포함한다.

<630> 패킷의 TCP부는 TCP 세그먼트라고 한다. TCP 세그먼트는 TCP 헤더 및 바디를 포함한다. TCP 세그먼트의 바디부는 HTTP 헤더 및 메시지를 포함한다. 메시지의 길이를 결정하기 위한 메커니즘이 두 가지 있는데, 하나는 청크 전송 인코딩(chunked transfer encoding)을 기초로 하는 메커니즘이고, 다른 하나는 콘텐츠-길이를 기초로 하는 메커니즘이다. 콘텐츠-길이 헤더 파일은 HTTP 헤더에서 발견된다. 콘텐츠-길이 헤더 필드가 존재하면, 바이트 단위의 그 값은 메시지-바디의 길이를 나타낸다. 이와 다르게, 청크 전송 인코딩 헤더가 HTTP 헤더에 존재하고 "청크" 전송 코딩이 적용되었음을 표시하는 경우에 메시지의 길이는 청크 인코딩에 의해서 한정된다. 청크 인코딩은 메시지를 일련의 청크 - 각 청크는 그 청크-사이즈 필드에 포함된 그 자신의 표시자를 가짐 - 로서 전송하기 위하여 메시지의 바디를 수정한다.

<631> 아래에서 상세히 설명하는 바와 같이, 일 실시예는 클라이언트가 "선택 시간" 상태인 경우를 회피하여 서버와 클라이언트간의 연결 풀링의 효율성을 증대기 위해, 콘텐츠-길이 파라미터 및/또는 청크 전송 인코딩 헤더를 이용한다. 이러한 실시예가 아닌 경우에, 인터페이스 유닛(1250)은 또 다른 클라이언트를 위해 그 연결을 재사용하기 이전에는 클라이언트로부터의 명령을, 또는 그 연결이 너무 오랫동안 유향 상태였다면 연결 타임 아웃을 대기한다.

<632> 위에서 언급한 32 비트 시퀀스 번호는 TCP 세그먼트내의 제1 바이트 데이터가 나타내는 전송 TCP로부터 수신 TCP로의 데이터 스트림 중의 바이트를 식별한다. 교환되는 모든 바이트에 번호가 매겨지므로, 확인 응답의 전송자가 수신할 것이라 예상하는 다음 시퀀스 번호를 포함한다. 그러므로, 이는 그 마지막 연속적으로 수신된 데이터 바이트에 1을 더한 시퀀스 번호이다. 체크섬은 TCP 세그먼트, 즉, TCP, 헤더와 그 응답 데이터(또는 바디)를 포괄한다. 이는 전송자가 계산하여 저장해야 하고 그 후 수신자가 검증해야 하는 필수 필드이다.

<633> 클라이언트로부터 들어온 패킷을 의도된 서버로 성공적으로 라우팅하거나 서버로부터 나간 패킷을 클라이언트로 성공적으로 라우팅하기 위해, 인터페이스 유닛(1250)은 "네트워크 어드레스 해석"으로 알려진 처리를 이용한다. 네트워크 어드레스 해석은 종래 기술에 공지되어 있는 것이며, URL <http://www.safetynet.net/RFC1631.txt>에서 볼 수 있는 RFC(request for comments; 1631)에 의해서 지정된다.

<634> 그러나, 클라이언트 및 서버 연결을 끊임없이 슬라이스하기 위한 신규의 해석 기법은 공동 소유의 1998년 11월 10일에 출원되었고 그 명칭이 "Internet Client-Server Multiplexer"(본 명세서에서는 "연결 다중화(connection multiplexing)"라 함)인 U.S 특허 출원 제 09/188,709호에 상세히 기술되었다. 이러한 기법에 따르면, 패킷은 TCP 프로토콜 레벨에서의 그 시퀀스 번호 및 긍정 응답 번호를 수정함으로써 해석된다. 이러한 기법의 중요한 점은 애플리케이션 계층 해석이 필요하지 않다는 것이다.

<635> 도 31을 참조하면, 단계 4402에 도시하는 바와 같이, 패킷의 네트워크 어드레스가 해석된다. 들어오는 패킷(즉, 클라이언트로부터 수신하는 패킷)의 경우에, 패킷의 소스 네트워크 어드레스는 인터페이스 유닛(1250)의 출력 포트의 어드레스로 변경되고, 목적지 네트워크 어드레스는 그 의도된 서버의 어드레스로 변경된다. 나가는 패킷(즉, 서버로부터 수신되는 패킷)의 경우에는, 소스 네트워크 어드레스는 서버의 어드레스로부터 인터페이스 유닛(1250)의 출력 포트의 어드레스로 변경되고, 목적지 어드레스는 인터페이스 유닛(1250)의 어드레스로부터 요청 클라이언트의 어드레스로 변경된다. 단계 4404에 도시하고 아래에서 상세히 설명하는 바와 같이, 패킷의 시퀀스 번호 및 긍정 응답 번호도 해석된다. 마지막으로 단계 4408에 도시하는 바와 같이, 이들 해석을 설명하도록 패킷 체크섬이 재계산된다.

<636> 위에서 언급한 바와 같이, 일 실시예는 특히, 서버와 클라이언트간의 연결 풀링의 효율성을 증대하기 위하여 콘텐츠-길이 파라미터 및/또는 청크 전송 인코딩 헤더를 통해 네트워크 클라이언트-서버 연결을 효율적으로 풀링하는 장치, 방법 및 컴퓨터 프로그램에 관한 것이다. 효율의 증대는 클라이언트가 "선택 시간" 상태인 동안에 그 연결을 점유하는 것을 방지하는 결과이다. 일 실시예에서, 콘텐츠 길이 파라미터는 메시지의 길이를 결정하는 데에 이용된다. 또 다른 실시예에서, 청크 전송 인코딩은 메시지의 길이를 결정하는 데에 이용된다. 다음 두 가지 실시예를 각각 도 32 및 33을 참조하여 설명한다.

<637> 도 32는 TCP 패킷의 TCP부가 TCP 세그먼트(4500)를 호출한 것을 도시한다. TCP 세그먼트(4500)는 TCP 헤더(4502) 및 바디(4504)를 포함한다. 바디(4504)는 다른 정보와 함께 HTTP 헤더 및 그 메시지를 포함한다. 콘텐츠 길이 파라미터(4506)는 HTTP 헤더에서 발견된다. 보다 효율적인 콘텐츠 풀링을 제공하기 위하여 콘텐츠 길

이 파라미터(4506)를 이용하는 방식은 도 35 및 36을 참조하여 아래에서 설명한다.

- <638> 도 33은 TCP 패킷의 TCP부가 TCP 세그먼트(4600)를 호출한 것을 도시한다. 위에서 언급한 바와 같이, 체크 전송 인코딩 헤더가 HTTP 헤더에 존재하고 그 "체크" 전송 인코딩이 적용되었음을 표시하면, 그 메시지의 길이는 체크 인코딩에 의해 한정된다. 체크 인코딩은, 일련의 체크 - 각각의 체크는 그 체크-사이즈 필드에 포함되는 자신 소유의 표시자를 가짐 - 로서 메시지를 전송하기 위하여 메시지의 바디를 수정한다. TCP 세그먼트(4600)는 TCP 헤더(미도시) 및 바디를 포함한다. 바디는 HTTP 헤더(4602A-4602C) 및 그 메시지 등을 포함한다. HTTP 헤더(4602A-4602C)는 7개의 체크-사이즈 필드(4606A-4606G)와, 6개의 체크 메시지 데이터(4604A-4604F)로 이루어진다.
- <639> 체크-사이즈 필드(4606A-4606G)는 도 33에 도시하는 바와 같이 서로 연결된다. 체크-사이즈 필드(4606A)는 체크 메시지 데이터(4604A)에 그 메시지의 길이를 나타내고, 체크-사이즈 필드(4606C)는 체크 메시지 데이터(4604C)에 그 메시지의 길이를 나타내는 등으로 한다. 마지막 체크-사이즈 필드(4606G)는 뒤따르는 메시지 데이터가 더 이상 없다는 것을 표시하는 길이 값 0을 항상 포함한다. 이것이 클라이언트로 전송된 메시지 전체의 표시이다. 보다 효율적인 콘텐츠 풀링을 제공하기 위하여 체크-길이 필드(4606A-4606G)를 이용하는 방식은 도 37 및 38을 참조하여 아래에서 설명한다. 도 33에서 TCP 세그먼트(4600)는 단지 예시를 위한 것이다.
- <640> 연결 풀링의 효율성을 높이기 위하여 실시예가 콘텐츠 길이 파라미터를 이용하는 방법을 상세히 설명하기 이전에, "인터넷 클라이언트-서버 다중화기"라는 제하에 1998년 11월 10일에 출원된 미국 특허 출원 제09/188,709호에 개시된 연결 풀링을 먼저 설명하겠다. 도 34는 연결 풀링을 설명하는 메시지 흐름도이다. 도 34는 2개의 클라이언트 C1 및 C2를 서버(S)로 연결시키는 인터페이스 유닛(1250)을 도시한다. 2개의 클라이언트 C1 및 C2는 본 명세서에서 기술한 클라이언트(10) 중 임의의 것을 포함하며, 서버(S)는 본 명세서에서 기술한 서버(30) 중 임의의 것을 포함한다. 먼저, 인터페이스 유닛(1250)은 플로우(4702)로 도시하는 바와 같이 클라이언트(C1)가 제공하는 네트워크 어드레스 1을 이용하여 클라이언트 C1와의 연결을 개설한다. 흐름선(4702)은 TCP/IP 프로토콜이 연결을 개설하기 위하여 다단 핸드셰이크(multi-stage handshake)를 이용하므로 양방 흐름으로서 도시되어 있다.
- <641> 연결이 개설되자마자, 인터페이스 유닛(1250)은 흐름선(704)에 의해 도시하는 바와 같이, /sales/forecast.html의 경로명을 지정하는 클라이언트 C1로부터의 GET 요청을 수신한다. 인터페이스 유닛(1250)과 서버(S)간에 자유 연결이 개설되어 있지 않으므로, 인터페이스 유닛(1250)은 서버(S)와의 연결을 개설한다. 인터페이스 유닛(1250)은 흐름선(4706)으로 나타내는 바와 같이 서버(S)를 지정하는 네트워크 어드레스 2로 이러한 요청을 매핑한다. 인터페이스 유닛(1250)은 흐름선(4708)에 의해 도시하는 바와 같이 그 서버로의 GET 요청도 전달한다. 서버(S)는 흐름선(4710)에 의해 도시하는 바와 같이 요청된 웹 페이지를 이용하여 응답한다. 인터페이스 유닛(1250)은 흐름선(4712)에 의해 도시하는 바와 같이 클라이언트(C1)으로 웹 페이지를 전달한다. 마지막으로, 흐름선(4714)에 의해 도시하는 바와 같이 클라이언트(C1)와 인터페이스 유닛(1250)간의 연결이 폐쇄된다. TCP/IP 프로토콜에 따라, 네트워크 연결을 폐쇄하는 것은 다단 처리를 포함할 수 있다. 그러므로, 흐름선(4714)은 양방향으로 도시되어 있다. 이는 인터페이스 유닛(1250)이 서버(S)와의 연결을 폐쇄하지 않고 다른 데이터 흐름을 수용하기 위해 개설 상태를 유지한다는 것을 명시한다.
- <642> 다음으로, 흐름선(4716)에 의해 도시하는 바와 같이, 클라이언트(C2)가 제공하는 네트워크 어드레스 1을 이용하여 인터페이스 유닛(1250)과 클라이언트(C2)간에 연결이 개설된다. 다음으로, 흐름선(4718)에 의해 도시하는 바와 같이 웹 페이지 /sales/forecast.html를 지정하는 클라이언트(C2)로부터의 GET 요청을 수신한다. 인터페이스 유닛(1250)과 서버(S)간에 이미 자유 연결이 개설되어 있으므로, 인터페이스 유닛(1250)이 추가 연결을 개설하는 처리 부하를 서버(S)가 갖게 할 필요가 없다. 인터페이스 유닛(1250)은 단지 자유 개설 연결을 이용한다. 인터페이스 유닛(1250)은 흐름선(4720)에 의해 도시하는 바와 같이 그 GET 요청을 서버(S)로 매핑하고 그 것을 전달하고 서버(S)로 전달한다. 흐름선(4722)에 도시하는 바와 같이 인터페이스 유닛(1250)은 서버(S)로부터의 응답을 수신하고 흐름선(4724)에 도시하는 바와 같이 그 응답을 클라이언트(C2)로 전달한다. 마지막으로, 흐름선(4726)에 도시하는 바와 같이 인터페이스 유닛(1250)은 클라이언트(C2)와의 연결을 폐쇄한다. 한번 더, 인터페이스 유닛(1250)은 서버(S)와의 연결을 폐쇄하지 않는다. 그 대신에, 인터페이스 유닛(1250)은 추가의 데이터 흐름을 수용하기 위하여 연결 개설 상태를 유지한다.
- <643> 위에서 설명한 바와 같이, 순서 흐름(4724)에 도시하는 바와 같이, 인터페이스 유닛(1250)이 클라이언트(C2)와의 연결을 폐쇄하게 하는 시나리오에는 여러가지가 있다. 예를 들어, 클라이언트는 모든 요청한 데이터(또는 메시지)를 클라이언트가 수신하자마자 발생하는 FIN(finish) 명령어를 시작할 수 있다. 클라이언트는 또한

RST(reset) 명령어를 시작할 수도 있다. 인터페이스 유닛(1250)과 클라이언트간의 연결을 폐쇄하는 데 더하여, RST 명령어는 다수의 하우스키퍼 동작이 수행되어 서버측 연결을 양호하게 유지하게 한다. 특히, TCP 프로토콜은 RST 명령어가, TCP 세그먼트를 서버가 수락하도록 올바른 SEQ(sequence) 수를 가질 것이지만, RST 명령어는 올바른 ACK(acknowledge) 수를 갖도록 보장되지는 않는다. 이러한 시나리오를 처리하기 위하여, 인터페이스 유닛(1250)은 서버가 전송한 데이터 바이트와 클라이언트에 의한 그 바이트 확인 응답을 추적한다. 클라이언트가 서버에 의해서 모든 데이터를 확인 응답하지 않았으면, 인터페이스 유닛(1250)은 확인 응답되지 않은 바이트를 계산하고 서버로 ACK를 전송한다. 또한, 서버측 PCB는 임의의 미결정 서버 데이터 전송이 차츰 소모될 수 있도록 타임 아웃 큐에 배치될 수도 있다.

<644> 또한, 도 34에 도시하지 않으나, 서버는 서버와 인터페이스 유닛(1250)간의 연결을 폐쇄할 수 있다. 서버는 인터페이스 유닛(1250)으로 FIN 명령어를 전송할 것이다. 이러한 경우에, 서버와 인터페이스 유닛(1250)간의 연결과, 인터페이스 유닛(1250)과 클라이언트간의 연결이 폐쇄될 것이다.

<645> 또 다른 측면은 서버가 그 연결을 폐쇄하는 경우를 최소화함으로써 서버로부터의 연결 처리의 부담을 최대로 더는 것이다. 여기에는 세가지 경우가 있다.

<646> (1) 프로토콜 버전 HTTP/1.1이 사용된다. 이러한 경우에, 명시적인 킵 어라이브 헤더가 필요하지 않다. 디폴트로써, 서버는 그 연결을 개설 상태로 유지하는 것인데, 이는 클라이언트가 그 연결을 폐쇄하는 것에 좌우된다. 일 실시예는 서버측 연결을 재사용함으로써 서버의 부담을 덜어준다. 이는 클라이언트가 그 연결을 폐쇄하는 것에 좌우되므로, 연결 폴링의 비효율성은, 클라이언트가 그 연결과 종료되지만 이러한 정보를 일정 기간 동안 인터페이스 유닛(1250)으로 전달하지 않는 경우에 발생한다. 인터페이스 유닛(1250)이 클라이언트 유닛으로부터의 명령어를 대기 중이므로, 또 다른 클라이언트를 위한 연결을 재사용하기 위해, 그 연결은 불필요하게 연결되어 있다.

<647> (2) 프로토콜 버전 HTTP/1.0가 사용되고 "Connection : Keep-Alive" 헤더가 클라이언트에 의해서 제공된다. 이러한 경우에, 서버는 그 연결을 개설된 상태로 유지하는데, 이는 클라이언트가 그 연결을 폐쇄하는 데에 좌우된다. 일 실시예는 그 서버측 연결을 재사용함으로써 서버의 부담을 덜어준다. 프로토콜 버전 HTTP/1.1을 이용하는 경우에, 비효율성은 클라이언트가 그 연결로 종료되지만, 이러한 정보를 소정 기간 동안 인터페이스 유닛(1250)으로 전달하지 않은 경우에 발생한다.

<648> (3) 프로토콜 버전 HTTP/1.0이 사용되고 "Connection : Keep-Alive" 헤더가 클라이언트에 의해서 제공되지 않는다. 이러한 경우에, 서버는 하나의 GET 요청을 완전히 충족시킨 이후에 연결을 정상적으로 폐쇄할 것이다. 이는 인터넷의 대부분이 "Connection : Keep Alive" 없이 HTTP/1.0을 사용한다는 것으로 판명된다. 이러한 특정한 중요한 경우에 서버측 연결을 재사용하는 것을 가능하게 하는 신규의 방법은 공동 소유의 "Internet Client-Server Multiplexer"라는 명칭의 1998년 11월 10일에 출원된 미국 특허 출원 제 09/188,709호에 상세히 기술되어 있다. 인터페이스 유닛(1250)은 이러한 상황을 검출하기 위하여 GET 패킷을 검사한다. 이러한 경우가 검출되면, 인터페이스 유닛(1250)이 "Connection : Keep-Alive"를 GET 패킷내로 삽입한다. 이는 클라이언트로 비가시적으로 수행되므로, 인터페이스 유닛(1250)이 서버측 연결 상에서 "추가되는 바이트" 수를 추적해야 한다. "추가되는 바이트"는 GET 패킷의 시퀀스 번호에 영향을 주지는 않는데, 그 이유는 그 시퀀스 번호가 제1 바이트의 시퀀스 번호이기 때문이다. 그러나, 인터페이스 유닛(1250)은 클라이언트로부터 서버로 후속 패킷들의 시퀀스 번호에 "추가되는 바이트"를 추가해야 한다. 이와 반대로, 서버는 그 추가 바이트를 확인 응답할 것이지만, 인터페이스 유닛(1250)은 클라이언트 — 이들 바이트가 추가되었다는 것을 모름 — 로 확인 응답을 전송하기 이전에 이들을 빼야만 한다.

<649> 위에서 언급한 바와 같이, 연결 다중화는 시퀀스 번호 및 확인 응답 번호를 곱셈함으로써 달성된다. 인터페이스 유닛(1250)에 의해 수신되는 세그먼트의 시퀀스 번호 및 확인 응답 번호는 수신자가 예상하는 값으로 수정 및 매핑된다. 예를 들어, "인플로"가 인터페이스 유닛(1250)에 의해 수신되는 세그먼트를 나타내고, "아웃플로"가 이에 대응해서 나가는 세그먼트를 나타내는 경우에, 시퀀스 및 확인 응답 번호는 다음이 방식으로 변경된다:

<650> 아웃플로 시퀀스 번호 = 인플로 시퀀스 번호 - 인플로 시작 시퀀스 번호 + 아웃플로 시작 시퀀스 번호

<651> 아웃플로 확인 응답 번호 = 인플로 확인 응답 번호 - 인플로 시작 확인 응답 번호 + 아웃플로 시작 확인 응답 번호

<652> "Connection : Keep-Alive" 헤더를 HTTP/1.0 패킷에 추가하는 문제를 해결하기 위하여, 인터페이스 유닛(1250)

0)은 연결을 적절한 절반에 — 이러한 경우에 서버측 — "추가 바이트"를 추적해야 한다. 시퀀스 번호 및 확인 응답 번호 식은 다음과 같이 변경된다.

- <653> 아웃플로 시퀀스 번호 = 인플로 시퀀스 번호 - 인플로 시작 시퀀스 번호 + 아웃플로 시작 시퀀스 번호 + 아웃플로 추가 바이트
- <654> 아웃플로 확인 응답 번호 = 인플로 확인 응답 번호 - 인플로 시작 확인 응답 번호 + 아웃플로 시작 확인 응답 번호 - 인플로 추가 바이트
- <655> 보다 효율적인 연결 풀링을 제공하기 위하여 일 실시예의 콘텐츠 길이 파라미터 기법을 통합하며 이들 수식을 이용하여 수행되는 해석의 특정 예를 도 35 및 36(콘텐츠 길이 파라미터 관련)과, 도 37 및 38(체크-사이즈 필드 관련)을 참조하여 아래에서 더 설명한다.
- <656> 도 35는 콘텐츠 길이 파라미터 기법을 통합하며 일 실시예에 의해 수행되는 확인 응답 번호 및 시퀀스 번호의 해석을 나타내는 상세한 흐름도이다. 도 35의 각 흐름에 대한 라벨은 T:S,A(L)의 형태인데, 여기서 T는 TCP 세그먼트 유형을 나타내고, S는 시퀀스 번호이고, A는 확인 응답 번호이고, L은 콘텐츠 길이 파라미터이다. 콘텐츠 길이 파라미터는 메시지에서 데이터의 바이트 개수를 나타낸다.
- <657> 플로우(4802A-4802C)가 클라이언트(C1)와 인터페이스 유닛(1250)간의 연결을 개설하는 한 가지 방법을 나타낸다. 각 플로우는 TCP 세그먼트를 나타낸다. TCP 세그먼트(4802A)에서 TCP 헤더내의 SYN 플래그가 설정되어 있는데, 이는 클라이언트(C1)로부터의 신규 연결 요청을 나타낸다. 클라이언트(C1)는 시작 시퀀스 번호 2000 및 확인 응답 번호 2000을 설정했다. 인터페이스 유닛(1250)은 플로우 (4802B)에 의해 나타내는 바와 같이 시작 시퀀스 번호 4000을 지정하고 확인 응답 번호를 2001로 1 증가시키는 SYN ACK 세그먼트로 응답을 한다. 각각의 엔티티(예를 들어, 클라이언트, 서버, 인터페이스 유닛)는 종래 기술 분야에 공지되어 있는 바와 같이 그 네트워크내에서 그 소유의 고유의 시퀀스 번호 및/또는 확인 응답 번호를 설정한다. 클라이언트(C1)는 플로우(4802C)에 나타내는 바와 같이 시퀀스 번호 2001로 지정을 하고, 확인 응답 번호를 4001로 1 증가시키는 ACK 세그먼트로 응답을 한다. 그 다음, 클라이언트(C1)는 플로우(4804)에 나타내는 바와 같이 49 바이트의 길이를 지정하는 GET 세그먼트를 전송한다.
- <658> 인터페이스 유닛(1250)이 서버(S)와의 자유 개설 연결이 존재하지 않는다고 판단하면, 플로우(4806A)에 나타내는 바와 같이, 시작 시퀀스 번호를 1950으로 지정하는 SYN 세그먼트를 서버(S)로 전송한다. 서버(S)는 플로우 (4906B)에 의해 나타내는 바와 같이 시작 시퀀스 번호를 6000으로 지정하고 확인 응답 번호를 1951로 1 증가시키는 SYN ACK 세그먼트로 응답을 한다. 인터페이스 유닛(1250)은 플로우(8060)로 나타내는 바와 같이 ACK 세그먼트로 응답을 한다. 그러면 위에서 설명한 해석 수식에 따라서 시퀀스 및 확인 응답 번호를 수정한 이후에 인터페이스 유닛(1250)은 흐름선(4808)에 의해 나타내는 바와 같이 클라이언트(C1)으로부터 서버(S)로 GET 세그먼트를 전달한다.
- <659> 서버(S)는 플로우(4810)에 의해 나타내는 바와 같이 시퀀스 번호 6001, 확인 응답 번호 2000 및 콘텐츠 길이 파라미터 999를 지정하는 요청 데이터로 응답을 한다. 인터페이스 유닛(1250)은 RESP 세그먼트를 수신하고, 이 시퀀스 및 확인 응답 번호를 해석하고 RESP 세그먼트를 흐름선(4812A)에 의해 나타내는 바와 같이 클라이언트(C1)으로 전달한다.
- <660> 여기서, 인터페이스 유닛(1250)은 클라이언트(C2)에 의해서 연결을 개설하라는 요청을 수신한다. 위와 같이, 플로우(4816A-4816C)가 클라이언트(C2)와 인터페이스 유닛(1250)간의 연결을 개설하는 한 가지 방법을 나타낸다. 또, 각 플로우는 TCP 세그먼트를 나타낸다. TCP 세그먼트(4816A)에서, TCP 헤더내의 SYN 플래그가 설정되어 있는데, 이는 클라이언트(C2)로부터의 새로운 연결 요청을 나타낸다. 클라이언트(C2)가 시작 시퀀스 번호 999와 확인 응답 번호 999를 설정했다. 인터페이스 유닛(1250)은 플로우(4816B)에 의해서 나타내는 바와 같이 시작 시퀀스 번호를 4999로 지정하고 확인 응답 번호를 1000으로 1 증가시키는 SYN ACK 세그먼트로 응답을 한다. 클라이언트(C2)는 시퀀스 번호 1000으로 지정하고 확인 응답 번호를 5000으로 1 증가시키는 ACK 세그먼트로 응답을 한다. 클라이언트(C2)는 그 후 플로우(4818)로 나타내는 바와 같이 50 바이트 길이를 지정하는 GET 세그먼트를 전송한다.
- <661> 이 때 인터페이스 유닛(1250)은 서버(S)로의 가용 연결이 없다고 가정한다. 이 목적은 클라이언트(C1)가 연결과 종료되었거나 또는 "선택 시간"인 경우 클라이언트(C1)를 위해 이전에 사용되었던 서버(S)로의 동일한 연결을 재사용하는 것이다. 클라이언트(C1)가 연결을 자유롭게 하도록 FIN(종료) 명령어 또는 RST(재설정) 명령어를 개시하는 것을 대기하는 것 대신에, 인터페이스 유닛(1250)은 콘텐츠 길이 파라미터를 이용하여 요청한 데이

터 모두가 클라이언트(C1)에 의해서 수신되었음을 확인한다. 이 때 플로우(4812B)에서는, 인터페이스 유닛(1250)이 클라이언트(C1)로부터, 클라이언트(C1)가 요청 데이터 모두를 수신했다는 확인을 받는다. 이는 클라이언트(C1)가 FIN 또는 RST 명령어를 전송하기 이전에 어떤 이유로 잠시 멈추어질 수 있더라도, 클라이언트(C1)가 그 연결과 종료된 상태임을 인터페이스 유닛(1250)에게 표시한다. 인터페이스 유닛(1250)은 플로우(4812C)로 나타내는 바와 같이 확인 응답 및 시퀀스 번호를 수정하고 RESP ACK 세그먼트를 서버(S)로 전달한다.

<662> 클라이언트(C1)가 사용된 동일한 연결을 이용하면, 인터페이스 유닛(1250)은 위에서 설명한 해석 수식에 따라 시퀀스 및 확인 응답 번호를 수정한 이후에 흐름선(4820)으로 나타내는 바와 같이 그 후 클라이언트(C2)로부터 서버(S)로 GET 세그먼트를 전달한다. 서버(S)는 플로우(4822)로 나타내는 바와 같이 시퀀스 번호 7000과, 확인 응답 번호 2050과, 콘텐츠 길이 파라미터 500을 지정하는 요청 데이터로 응답을 한다.

<663> 인터페이스 유닛(1250)은 흐름선(4824A)으로 나타내는 바와 같이 RESP 세그먼트를 수신하고, 이 시퀀스 및 확인 응답 번호를 해석하고, RESP 세그먼트를 클라이언트(C2)로 전달한다. 여기서, 플로우(4824B)에서, 인터페이스 유닛(1250)은 클라이언트(C2)로부터, 클라이언트(C2)가 요청 데이터 모두를 수신했다는 사실을 확인 받는다. 인터페이스 유닛(1250)은 플로우(4824C)로 나타내는 바와 같이 확인 응답 및 시퀀스 번호를 수정하고 RESP ACK 세그먼트를 서버(S)로 전달한다.

<664> 그 후 클라이언트(C2)와 인터페이스 유닛(1250)간의 연결은, 플로우(4826A 내지 4826D)로 나타내는 바와 같이 인터페이스 유닛(1250) 클라이언트(C2)로부터 FIN 또는 RST 명령어를 수신하자마자 폐쇄되거나 연결 해제된다. 이와 마찬가지로, 클라이언트(C1)와 인터페이스 유닛(1250)간의 연결은 플로우(4814A 내지 4814D)로 표시되는 바와 같이 인터페이스 유닛(1250)이 클라이언트(C1)로부터 FIN 또는 RST 명령어를 수신하자마자 폐쇄되거나 연결 해제된다. 그러나, 인터페이스 유닛(1250)은 서버(S)와의 연결을 유지하는 점이 중요하다. 또한, 도 36을 참조하여 설명한 바와 같이 이벤트의 시퀀스들은 예시를 목적일 뿐이다.

<665> 도 36은 일 실시예에 따라 클라이언트와 서버간의 연결의 풀링의 효율성을 증가시키기 위하여 콘텐츠 길이 파라미터를 이용하는 동작을 나타내는 흐름도이다. 인터페이스 유닛(1250)은 복수의 서버와의 연결을 유지하고, 클라이언트 요청에서 지정된 경로명을 기초로 하여 클라이언트 요청들을 이들 서버로 라우팅한다. 먼저, 단계 4902에 도시하는 바와 같이 인터페이스 유닛(1250)은 서버와의 연결을 개설한다. 다음으로, 단계 4904에 도시하는 바와 같이, 클라이언트(C1) 요청에 응답하여, 인터페이스 유닛(1250)이 클라이언트(C1)로의 연결을 개설하고 경로명을 이용하여 데이터를 검색하라는 클라이언트(C1)로부터의 요청을 수신한다.

<666> 인터페이스 유닛(1250)은 그 후 단계 4906에 도시하는 바와 같이 경로명에 의해서 지정되는 콘텐츠를 호스팅하는 서버를 선택한다. 다른 실시예에서, 인터페이스 유닛(1250)은 적절한 서버를 선택하기 위하여 다른 사전 정의된 정책들, 예를 들어 서버의 부하 및 서버의 상태를 조사한다. 인터페이스 유닛(1250)은 그것이 의도한 서버 팜 및 서버의 데이터베이스를 관리 및 유지한다. 다른 것들 중에서, 이러한 데이터베이스의 정보는 현재 사용중인, 즉, 인터페이스 유닛(1250)이 입력 패킷들을 올바른 서버로 지향시킬 수 있게 하는 정책 및 규칙들을 포함한다. 원하는 네트워크 상태 및 서비스들에 따라, 이들 정책 및 규칙들은 매우 빠르게 변할 수 있다.

<667> 그 후 인터페이스 유닛(1250)은 단계 4908에 도시하는 바와 같이 요청을 해석하고 해석된 요청을 선택된 서버로 전달한다. 인터페이스 유닛(1250)은 단계 4910에 도시하는 바와 같이 서버(S)로부터 그 응답을 수신한다. 그 다음 인터페이스 유닛(1250)은 단계 4912에 도시하는 바와 같이 그 응답을 해석하고, 해석된 응답을 클라이언트(C1)로 전달한다.

<668> 이 때, 인터페이스 유닛(1250)이 클라이언트(C2)로부터 데이터를 검색하라는 요청을 수신한다 가정하자. 인터페이스 유닛(1250)은 단계 4914에 도시하는 바와 같이, 클라이언트(C2) 요청에 응답하여, 클라이언트(C2)로의 연결을 개설하고, 경로명을 이용하여 데이터를 검색하라는 클라이언트(C2)로부터의 요청을 수신한다. 그 다음 단계 4916에 도시하는 바와 같이, 인터페이스 유닛(1250)은 경로명으로 지정되는 콘텐츠를 호스팅하는 서버를 선택한다.

<669> 단계 4918)에서, 인터페이스 유닛(1250)은 클라이언트(C2)가 클라이언트(C1)와 동일한 서버를 선택했는 지를 판단한다. 단계 4918)에 대한 결과가 부정이면, 인터페이스 유닛(1250)은 클라이언트(C2)의 요청을 충족시키기에 데 필요한 방식으로 진행된다(이 실시예에서는 중요하지 않음). 이 시점에서, 도 36의 순서는 종료된다. 이와 다르게, 단계 4918)에 대한 결과가 긍정이면, 인터페이스 유닛(1250)은 단계 4920에 도시하는 바와 같이, 선택된 서버로의 임의의 개설된 연결이 있는 지를 판단한다.

- <670> 단계 4920에 대한 결과가 긍정이면, 인터페이스 유닛(1250)은 클라이언트(C2)의 요청을 충족시키기는 데 필요한 방식으로 진행된다(이 실시예에서는 중요하지 않음). 이 시점에서, 도 36의 순서는 종료된다. 이와 다르게, 단계 4920에 대한 결과가 부정이면, 인터페이스 유닛(1250)은 단계 4922에 도시하는 바와 같이, 콘텐츠 길이 파라미터를 이용하여 클라이언트(C1)가 요청한 모든 데이터를 클라이언트(C1)가 수신했음을 확인한다. 인터페이스 유닛(1250)은 클라이언트(C1)가 연결 종료되었거나 "선택 시간"임을 판단하기 위하여 클라이언트(C1)가 FIN 또는 RST 명령어를 전송하는 것을 대기하지 않는다는 것이 중요하다. 이로써, 또 다른 클라이언트를 위하여 그 연결을 재사용하기 이전에 인터페이스 유닛(1250)이 클라이언트가 연결을 폐쇄하는 것을 대기하는 것보다 인터페이스 유닛(1250)이 각 연결을 보다 신속하게 이용할 수 있다는 사실로 인해 연결 풀링이 보다 효율적으로 된다.
- <671> 단계 4924에서, 인터페이스 유닛(1250)은 그 다음 요청을 해석하고, 그 해석된 요청을 단계 4924에 도시하는 바와 같이 클라이언트(C1)가 사용한 것과 동일한 연결을 이용하여 선택된 서버로 전달한다. 인터페이스 유닛(1250)은 단계 4926에 도시하는 바와 같이 서버(S)로부터 응답을 수신한다. 인터페이스 유닛(1250)은 그 후 응답을 해석하고, 해석된 응답을 클라이언트(C2)로 전달한다. 인터페이스 유닛(1250)은 단계 4930에 도시하는 바와 같이 콘텐츠 길이 파라미터를 이용하여 클라이언트(C2)가 요청한 데이터 전체를 클라이언트(C2)가 수신했음을 확인한다.
- <672> 다음으로, 인터페이스 유닛(1250)은 단계 4932에서, 클라이언트(C2)와의 연결을 폐쇄 또는 연결 해제한다. 마지막으로 인터페이스 유닛(1250)은 단계 4934에서 클라이언트(C1)와의 연결을 폐쇄 또는 연결 해제하고, 도 36에서의 순서는 종료된다. 도 35를 참조하여 위에서 언급한 바와 같이, 도 36을 참조하여 설명한 이벤트의 시퀀스는 예시일 뿐이다.
- <673> 도 37은 체크-사이즈 필드 기술을 통합하며 일 실시예가 수행하는 확인 응답 및 시퀀스 번호 해석을 설명하는 상세한 흐름도이다. 도 37에서의 각 순서에 대한 라벨은 T:S,A(L)의 형태인데, 여기서 T는 TCP 세그먼트 유형을 나타내고, S는 시퀀스 번호이고, A는 확인 응답 번호이고, L은 체크-사이즈 필드이다. 체크-사이즈 필드의 전체값은 TCP 세그먼트에서 데이터 바이트의 개수를 나타낸다.
- <674> 간단히 하기 위하여, 클라이언트(C1)와 클라이언트(C2) 둘 다에 대한 연결이 이미 설정되었다고 가정하자. 그 다음 클라이언트(C1)는 플로우(4002)에 의해 나타내는 바와 같이 49 바이트의 길이를 지정하는 GET 세그먼트를 전송한다. 인터페이스 유닛(1250)은 서버(S)와의 자유 개설 연결이 없다고 판단하므로, 서버(S)와의 연결을 개설한다(도 37에는 미도시). 그 다음 인터페이스 유닛(1250)은 위에서 설명한 해석 수식에 따라 시퀀스 및 확인 응답 번호들을 수정한 이후에 흐름선(4004)에 의해 나타내는 바와 같이, 클라이언트(C1)로부터 서버(S)로 GET 세그먼트를 전달한다.
- <675> 예로, 응답 세그먼트에서 그 데이터는 999의 전체 콘텐츠 데이터 길이를 갖는다고 가정하자. 또, 그 데이터는 2개의 300 데이터 체크 및 하나의 399 데이터 체크로 해석될 것이라 가정하자. 이는 예시일 뿐이며 이에 제한하려는 것이 아니다. 그러므로, 서버(S)는 먼저 플로우(4008A)에 의해 나타내는 바와 같이 시퀀스 번호 6001, 확인 응답 번호 2000, 그리고 체크-사이즈 필드를 300을 지정하는 요청 데이터(또는 메시지)의 체크로 응답을 한다. 인터페이스 유닛(1250)은 RESP 세그먼트를 수신하고 그 시퀀스 및 확인 응답 번호를 해석하고 흐름선(4006A)로 나타내는 바와 같이 RESP 세그먼트를 클라이언트(C1)으로 전달한다. 클라이언트(C1)는 흐름선(4006B)으로 나타내는 바와 같이 인터페이스 유닛(1250)이 데이터를 수신했음을 확인 응답한다. 인터페이스 유닛(1250)은 흐름선(4008B)으로 나타내는 바와 같이 서버(S)상으로 이러한 확인 응답을 되돌려 전달한다.
- <676> 서버(S)는 다음으로 플로우(4012A)로 나타내는 바와 같이, 시퀀스 번호 6301, 확인 응답 번호 2001, 그리고 체크-사이즈 필드 300을 지정하는, 요청된 데이터의 제2 체크로 응답을 한다. 인터페이스 유닛(1250)은 흐름선(4010A)로 나타내는 바와 같이, RESP 세그먼트를 수신하고, 그 시퀀스 및 확인 응답 번호를 해석하고, RESP 세그먼트를 클라이언트(C1)으로 전달한다. 클라이언트(C1)는 흐름선(4010B)으로 나타내는 바와 같이, 인터페이스 유닛(1250)으로 그 데이터가 수신되었음을 확인 응답한다. 인터페이스 유닛(1250)은 흐름선(4012B)으로 나타내는 바와 같이 서버(S) 상으로 이러한 확인 응답을 되돌려 전달한다.
- <677> 서버(S)는 플로우(4016A)로 나타내는 바와 같이 시퀀스 번호 6001, 확인 응답 번호 2002, 그리고 체크-사이즈 필드 399를 지정하는, 요청된 데이터의 제3 체크로 응답을 한다. 인터페이스 유닛(1250)은 흐름선(4014A)로 나타내는 바와 같이 RESP 세그먼트를 수신하고, 이 시퀀스 및 확인 응답 번호를 해석하고, RESP 세그먼트를 클라이언트(C1)으로 전달한다. 클라이언트(C1)는 흐름선(4014B)으로 나타내는 바와 같이, 인터페이스 유닛(1250)으로 그 데이터가 수신되었음을 확인 응답한다. 인터페이스 유닛(1250)은 흐름선(4016B)으로 나타내는 바와 같이

서버(S) 상으로 이러한 확인 응답을 되돌려 전달한다.

- <678> 마지막으로, 서버(S)는 플로우(4020)로 나타내는 바와 같이 시퀀스 번호 7000, 확인 응답 번호 2003, 그리고 체크-사이즈 필드 0을 지정하는, 0 데이터의 마지막 체크(체크-사이즈 필드가 0과 동일하게 표시됨)로 응답을 한다. 인터페이스 유닛(1250)은 흐름선(4018)으로 나타내는 바와 같이 RESP 세그먼트를 수신하고, 이 시퀀스 및 확인 응답 번호를 해석하고, RESP 세그먼트를 클라이언트(C1)으로 전달한다. 이로써 요청된 데이터의 모두가 전달되었음을 인터페이스(1250) 및 클라이언트(C1)에게 표시된다.
- <679> 이 시점에서, 클라이언트(C2)는 흐름 4022로 도시된 바와 같이 50 바이트의 길이를 지정하는 GET 세그먼트를 전송한다. 이 시점에서 인터페이스 유닛(1250)은 서버(S)에의 이용 가능한 연결이 없는 것으로 가정한다. 목표는 클라이언트(C1)가 연결 종료되거나 "선택 시간(think time)" 상태인 경우 클라이언트(C1)를 위해 이전에 사용되었던 서버(S)에 대해 동일 연결을 다시 이용하는 것이다. 클라이언트(C1)가 연결을 비우기 위한 FIN(finish) 명령 또는 RST(reset) 명령을 시작하기를 기다리지 않고, 인터페이스 유닛은 요청된 데이터 모두가 클라이언트(C1)에 의해 수신된 것으로 확인하기 위해 0인 체크-사이즈 필드를 이용한다. 이는 FIN 또는 RST 명령을 전송하기 전에 클라이언트(C1)가 몇 가지 이유로 중단되어 있더라도, 클라이언트(C1)의 연결 종료를 인터페이스 유닛(1250)에게 알린다. 인터페이스 유닛(1250)은 흐름 4024로 도시된 바와 같이 확인 응답 및 시퀀스 번호를 수정하고, GET 세그먼트를 서버(S)로 전송한다.
- <680> 예시를 위해, 응답 세그먼트 내의 데이터는 500의 전체 콘텐츠 데이터 길이를 갖고 있다고 가정한다. 또한, 데이터는 하나의 300 데이터 체크 및 하나의 200 데이터 체크 내에서 전송되는 것으로 가정한다. 이는 단지 예시를 위한 것으로 한정을 의도한 것이 아님에 주의한다. 그러므로, 서버(S)는 먼저 흐름 1028A로 도시된 바와 같이 7000의 시퀀스 번호, 2050의 확인 응답 횟수, 및 300의 체크-사이즈 필드를 명시한 요청된 데이터의 체크로 응답한다. 인터페이스 유닛(1250)은 흐름선 1026A로 도시된 바와 같이 RESP 세그먼트를 수신하고, 시퀀스 및 확인 응답 횟수를 변환하고, RESP 세그먼트를 클라이언트(C2)에 전송한다. 클라이언트(C2)는 흐름선 4026B로 도시된 바와 같이 데이터의 수신에 대해 인터페이스 유닛(1250)에 확인 응답한다. 또한, 인터페이스 유닛(1250)은 흐름선 4028B로 도시된 바와 같이 이 확인 응답을 서버(S)에 전달한다.
- <681> 서버(S)는 다음에 흐름 4032A로 도시된 바와 같이 7300의 시퀀스 번호, 2051의 확인 응답 횟수, 및 200의 체크-사이즈 필드를 명시한 요청된 데이터의 제2 체크로 응답한다. 인터페이스 유닛(1250)은 흐름선 4030A로 도시된 바와 같이 RESP 세그먼트를 수신하고, 시퀀스 및 확인 응답수를 변환하고, RESP 세그먼트를 클라이언트(C2)에 전송한다. 클라이언트(C2)는 흐름선 4030B로 도시된 바와 같이 데이터의 수신에 대해 인터페이스 유닛(1250)에 확인 응답한다. 인터페이스 유닛(1250)은 또한 흐름선 4032B로 도시된 바와 같이 이 정보를 서버(S)에 전달한다.
- <682> 마지막으로, 서버(S)는 흐름 4036으로 도시된 바와 같이 7500의 시퀀스 번호, 2052의 확인 응답 횟수, 및 0의 체크-사이즈 필드를 명시한 제로 데이터의 최종 체크(0인 체크-사이즈 필드로 나타냄)로 응답한다. 인터페이스 유닛(1250)은 흐름선 4034로 도시된 바와 같이 RESP 세그먼트를 수신하고, 시퀀스 및 확인 응답 횟수를 변환하고, RESP 세그먼트를 클라이언트(C2)에 전송한다. 이는 인터페이스 유닛(1250) 및 클라이언트(C2)에게 상기 요청된 데이터가 모두 전송되었음을 알린다.
- <683> 다음에, 일단 인터페이스 유닛(1250)이 클라이언트(C2)로부터 FIN 또는 RST 명령을 수신하면 흐름 4038로 도시된 바와 같이 클라이언트(C2)와 인터페이스 유닛(1250)간의 연결이 폐쇄되거나 링크 해제된다. 마찬가지로, 클라이언트(C1)로부터 FIN 또는 RST 명령을 일단 수신하면 흐름 4040으로 도시된 바와 같이 클라이언트(C1)와 인터페이스 유닛(1250)간의 연결이 폐쇄되거나 링크 해제된다. 그러나, 인터페이스 유닛(1250)이 서버(S)와의 연결을 유지하는 것에 주의하는 것이 중요하다. 이벤트 시퀀스는 도 37을 참조하여 설명한 바와 같이 단지 예시를 위한 것으로 한정하는 것이 아님에 주의하는 것이 중요하다.
- <684> 도 38은 일 실시예에 따라 클라이언트와 서버간의 연결의 폴링 효율을 증가시키는 체크-사이즈 필드를 이용하는 동작을 나타낸 흐름도이다. 인터페이스 유닛(1250)은 복수의 서버와 연결을 유지하고, 클라이언트 요청에 지정된 경로명에 기초하여 클라이언트 요청을 이들 서버에 라우팅한다. 먼저, 인터페이스 유닛(1250)은 단계 4102에 도시된 바와 같이 서버와의 연결을 개방한다. 다음에, 클라이언트(C1) 요청에 응답하여, 인터페이스 유닛(1250)은 단계 4104에 도시된 바와 같이 클라이언트(C1)에의 연결을 개방하고 경로명을 이용하여 데이터를 검색하라는 요청을 클라이언트(C1)로부터 수신한다.
- <685> 다음에, 인터페이스 유닛(1250)은 단계 4106에 도시된 바와 같이 경로명에 의해 지정된 서버 호스팅 콘텐츠를

선택한다. 다음에, 인터페이스 유닛(1250)은 단계 4108에 도시된 바와 같이 요청을 변환하고 변환된 요청을 선택된 서버에 전달한다. 인터페이스 유닛(1250)은 단계 4110에 도시된 바와 같이 서버(S)로부터 응답을 수신한다. 다음에, 인터페이스 유닛(1250)은 단계 4112에 도시된 바와 같이 체크-사이즈 필드가 0일 때까지 응답을 변환하고 변환된 응답을 클라이언트(C1)에 전달한다.

<686> 예시를 위해, 이 시점에서 인터페이스 유닛(1250)은 클라이언트(C2)로부터 연결을 개방하라는 요청을 수신한다고 가정한다. 인터페이스 유닛(1250)은 단계 4114에 도시된 바와 같이 클라이언트(C2) 요청에 응답하여, 클라이언트(C2)에 대한 연결을 개방하고 경로명을 이용하여 데이터를 검색하라는 클라이언트(C2)로부터의 요청을 수신한다. 인터페이스 유닛(1250)은 단계 4116에 도시된 바와 같이 경로명에 의해 지정된 서버 호스팅 콘텐츠를 선택한다.

<687> 단계 4118에서, 인터페이스 유닛(1250)은 클라이언트(C2)가 클라이언트(C1)와 동일한 서버를 선택하였는지를 판단한다. 단계 4118에 대한 결과가 부정이면, 인터페이스 유닛(1250)은 클라이언트(C2)의 요청을 충족시키는데 필요한 형식으로 진행한다. 이 시점에서 도 38의 흐름도는 종료된다. 대안으로, 단계 4118에 대한 결과가 긍정이면, 인터페이스 유닛(1250)은 단계 4120에 도시된 바와 같이 선택된 서버에의 개방된 연결이 있는지를 판단한다.

<688> 단계 4120에 대한 결과가 긍정이면, 인터페이스 유닛(1250)은 클라이언트(C2)의 요청을 충족시키는데 필요한 형식으로 진행한다. 이 시점에서 도 38의 흐름도는 종료된다. 대안으로, 단계 4120에 대한 결과가 부정이면, 인터페이스 유닛(1250)은 단계 4112에서 클라이언트(C1)가 요청한 모든 메시지 데이터를 수신하였음을 확인하기 위해 체크-사이즈 필드가 0이라는 사실을 이용한다. 인터페이스 유닛(1250)은 클라이언트(C1)가 연결 종료되거나 "선택 시간" 상태인 것으로 판단하기 위해서 클라이언트(C1)가 FIN 또는 RST 명령을 전송하는 것을 기다리지 않음에 주의하는 것이 중요하다.

<689> 단계 4122에서, 인터페이스 유닛(1250)은 요청을 변환하고 사용된 클라이언트(C1)와 동일한 연결을 이용하여 변환된 요청을 선택된 서버에 전달한다. 인터페이스 유닛(1250)은 단계 4124에 도시된 바와 같이 서버(S)로부터 응답을 수신한다. 다음에, 인터페이스 유닛(1250)은 단계 4126에 도시된 바와 같이 체크-사이즈 필드가 0이 될 때까지 응답을 변환하고 변환된 응답을 클라이언트(C2)에 전달한다. 인터페이스 유닛(1250)은 클라이언트(C2)가 요청한 모든 메시지 데이터가 수신되었음을 확인하기 위해 체크-사이즈 필드를 이용한다.

<690> 다음에, 인터페이스 유닛(1250)은 단계 4128에서 클라이언트(C2)와의 연결을 폐쇄하거나 링크 해제한다. 마지막으로, 인터페이스 유닛(1250)은 단계 4130에서 클라이언트(C1)와의 연결을 폐쇄하거나 링크 해제하고, 도 38의 흐름도는 종료된다. 도 37을 참조하여 위에서 설명한 바와 같이, 이벤트 시퀀스는 도 38을 참조하여 설명되는 바와 같이 단지 예시를 위한 것으로 한정하는 것이 아니다.

<691> 이전 실시예들이 서버로부터의 연결 처리 오버헤드를 오프로딩하기 위해 팜 내의 서버에 연결된 인터페이스 유닛(1250)과 같은 인터페이스 유닛 내에 구현되는 경우를 구체적으로 설명한다. 그러나, 실시예들은 또한 클라이언트와 서버간의 네트워크 연결 경로에 존재하는 다른 종류의 디바이스 내에 적용될 수 있다. 네트워크 트래픽이 이러한 디바이스를 통해 흐름에 따라, 트래픽 모두는 연결 처리를 오프로딩할 기회를 갖는다. 이러한 디바이스들의 일부 예는 다음과 같다:

<692> - 서버 팜(로컬 또는 지리적으로 분산됨) 내의 한 세트의 서버들 간의 클라이언트 네트워크 연결을 분산시키는 부하 밸런서.

<693> - 네트워크 트래픽을 감시하고 패킷 흐름을 측정하는 대역폭 관리자.

<694> - 방화벽은 패킷을 감시하고 권한 검증된 패킷만이 흐르도록 한다.

<695> - 라우터 및 교환기도 네트워크 트래픽의 경로 내에 있다. 업계 추세는 부가적인 기능(예컨대, 부하 밸런싱, 대역폭 관리 및 방화벽 기능)을 이들 디바이스 내에 통합하는 것일 수 있다.

<696> 실시예들은 또한 네트워크 연결의 엔드포인트인 컴퓨터 시스템 내에 적용될 수 있다. 이 경우에, 애드온 카드(add-on cards)가 컴퓨터 시스템 내의 메인 처리 소자들을 오프로딩하는데 사용될 수 있다.

<697> 3. 통합된 캐싱

<698> 도 39는 일 실시예에 따라 통합된 캐싱 기능을 제공하는 어플라이언스에서 일어날 수 있는 이벤트 시퀀스의 흐름도(5300)를 예시한다. 그러나, 일 실시예는 흐름도(5300)에 의해 제공되는 설명에 한정되지 않는다.

오히려, 당업자(들)에게는 다른 기능 흐름들이 실시예의 범위 및 취지 내에 속함을 여기서 제공된 콘텐츠로부터 명백할 것이다. 이들 다른 기능 흐름은 캐싱의 통합 상에 다른 처리, 다른 시퀀싱, 및 다른 변동을 포함할 수 있다.

- <699> 흐름도(5300)의 방법은 데이터 통신망에 통신 가능하게 연결된 하나 이상의 디바이스(들)로 구현될 수 있다. 예컨대, 흐름도(5300)의 방법은, 도 27을 참조하여 설명된 바와 같은 소프트웨어 아키텍처(3200)를 가진 어플라이언스, 예컨대 도 1a를 참조하여 설명된 어플라이언스(1250)에 구현될 수 있다. 흐름도(5300)의 방법은 이 실시예를 계속 참조하여 설명한다.
- <700> 도 39에 도시된 바와 같이, 흐름도(5300)의 방법은 단계 5302에서 시작되며, 어플라이언스(1250)는 클라이언트(10) 중 하나로부터 암호화된 패킷을 수신한다. 일 실시예에서, 어플라이언스(1250)는 서버(30)에 대해 프록시 SSL 엔드포인트의 역할을 하도록 구성되어, 클라이언트(10)로부터 수신된 암호화된 패킷을 암호 해독한 다음에, 필요한 다른 처리를 위해 결국 암호화된 패킷 내의 주소 정보에 기초하여 적절한 자원에 전송한다. 적절한 자원은 예컨대 서버(30) 또는 어플라이언스(1250)에 의해 관리되는 캐시일 수 있다. 단계 5304에서, 어플라이언스(1250)는 암호 해독 처리를 패킷에 대해 수행한다.
- <701> 단계 5306에서, 액세스 제어를 위한 AAA 정책을 실행하도록 일 실시예에 따라 구성된 어플라이언스(1250)는 암호화된 패킷이 수신된 클라이언트를 인증 및/또는 권한 검증한다.
- <702> 단계 5308에서, 일정한 유형의 패킷 처리를 수행하도록 일 실시예에 따라 구성된 어플라이언스(1250)는 응용 가능 네트워크 프로토콜에 의해 발생된 연결 오버헤드 처리 요건을 감소시키기 위해 암호 해독된 패킷에 대해 패킷 처리를 실행한다.
- <703> 단계 5310에서, 어플라이언스(1250)는, 일 실시예에 따라 콘텐츠를 압축 및 압축 해제하도록 구성되고, 패킷과 연관된 요청을 압축 해제한다. 일 실시예에서, 요청은 웹 객체 요청을 포함한다.
- <704> 단계 5312에서, 어플라이언스(1250)는 캐시 기능을 활성화시킬 수 있고, 객체에 대한 클리어 및/또는 권한 검증 및/또는 압축 해제 및/또는 패킷-처리된 요청을 수신한다. 단계 5302, 5304, 5306, 5308, 5310을 참조하여 설명한 종전 처리 때문에, 캐시 관리 로직은 클리어/권한 검증/압축해제/패킷처리된 요청에 기초하여 객체가 캐싱되었는지 아니면 캐싱가능한 지에 대해 판단을 할 수 있으며, 그러므로 전통적인 캐시보다 훨씬 넓은 어레이의 요청을 처리할 수 있고, 전통적인 접근법들보다 효율적으로 캐싱을 실행할 수 있다. 또한, 캐시 관리 로직이 다른 프로세스와 함께 커널 공간에서 일을 하기 때문에, 캐시 관리 로직은 다른 애플리케이션 각각과 이러한 데이터 구조에 관련하여 동일한 상태를 가진 데이터 구조로서의 해당 객체에 관련되며, 그러므로 통합은 매우 효율적인 방식으로 실행된다.
- <705> 단계 5314에 도시된 바와 같이, 객체가 캐시 메모리에 아직 존재하지 않으면, 어플라이언스(1250)는 요청을 하나 이상의 서버(30)에 전송한다. 그러나, 요청이 전송되기 전에 여러 부가적인 처리 단계들이 일어날 수 있다.
- <706> 예컨대, 단계 5316에서, 어플라이언스(1250)는 서버(S)로의 요청의 효율적인 전달을 보장하기 위해 선택적으로 연결 처리를 수행하고, 단계 5318에서, 어플라이언스(1250)는 요청이 가장 적절한 서버(S)에게 전송되도록 보장하기 위해 선택적으로 부하 밸런싱 판단을 한다. 또한, 일 실시예에서, 요청은 백-엔드 암호화 프로세스(back-end encryption process)를 통해 서버(S)에 전송되기 전에 암호화되며, 이에 따라 단대단 네트워크 보안이 제공된다. 단계 5320에서, 요청이 서버(S)로 전송된다.
- <707> 단계 5322에서, 어플라이언스(1250)는 서버(30)들 중 하나로부터 응답을 수신한다. 백-엔드 암호화가 위에서 설명한 바와 같이 지원되면, 어플라이언스(1250)는 서버로부터의 응답을 암호 해독한다.
- <708> 단계 5324에서, 어플라이언스(1250)는 서버로부터의 응답과 연관된 객체를 압축한다. 일 실시예에서, 객체는 웹 객체를 포함한다.
- <709> 단계 5326에서, 어플라이언스(1250) 내의 캐시 관리 로직은 캐시 압축된 형태로 객체를 저장한다. 캐시 관리 로직은 저장 압축된 객체를 처리 능력으로 인해 이 형식으로 저장할 수 있다. 일단 객체가 캐시에 저장되면, 객체에 대한 장래 클라이언트 요청은 위에서 설명된 바와 같은 단계 5316, 5318, 5320, 5322, 5324, 5326의 수행 없이 캐시로부터 서빙될 수 있다. 이는 흐름도(5300)에서 판단 단계 5314 내지 단계 5328을 직접 연결하는 선으로 나타내어져 있다.
- <710> 종료 5328에서, 객체가 서버로부터 수신되거나 캐시로부터 검색된 후, 어플라이언스(1250)는 원래 클라이언트 요청을 보다 효율적으로 서비스하기 위해 연결에 대해 패킷 처리를 수행한다. 단계 5330에서, 응답 객체는 제

암호화되어 클라이언트에게 다시 전달된다.

- <711> 위에서 설명된 처리 단계들의 각각은 어플라이언스(1250)의 커널/OS 레벨에서 일어난다. 커널/OS 공간에서 다른 처리 단계의 중간에 그리고 그 처리 단계와 통합된 캐시를 구현함으로써, 일 실시예는 부가적인 기능을 제공하고 캐시의 성능을 향상시킬 수 있다.
- <712> 이러한 통합으로, 일 실시예에 따른 캐시 구현은 전통적으로 캐시의 기능적 능력을 벗어난 부가적인 기능을 수행할 수 있다. 예컨대, 일 실시예는 캐시가 암호화 및/또는 압축된 객체와 일을 할 수 있도록 한다.
- <713> 일 실시예에 의해 달성될 수 있는 부가적인 기능의 다른 예로는 단대단 암호화 HTTPS 트래픽의 캐싱을 들 수 있다. 전형적으로, 캐시들은 서버로부터의 암호화되지 않은 HTTP 응답만을 저장한다. 어떤 캐시들은 일부 경우에 캐시로부터 클라이언트로의 SSL 암호화 HTTPS 제공을 지원할 수 있으나, 어떤 경우라도, 전통적인 캐시들은 서버에 의해 암호화된 응답을 캐싱할 수 없으며, 따라서 단대단(즉, 서버 대 클라이언트) 암호화를 지원할 수 없다. 전형적으로, 응답이 HTTPS의 형태로 서버에 의해 암호화될 때, 캐시는 이러한 응답을 암호 해독할 수 없으며, 그러므로 그 캐시 메모리에 응답을 저장할 수 없다. 이 때문에, 전통적인 캐시들은 일 실시예에서 단대단 암호화된 트래픽에도 불구하고 이익을 제공하지 못하며, 통합된 캐싱 어플라이언스는 SSL 암호화 HTTPS 트래픽에 대해 2-방식 중단 포인트의 역할을 한다.
- <714> 예컨대, 일 실시예에서, 통합된 캐싱 어플라이언스는 암호화된 트래픽 서버와 어플라이언스 사이에서 그리고 어플라이언스와 클라이언트 사이에서 중단 포인트의 역할을 한다. 이 방식으로, 어플라이언스는 서버로부터 수신된 SSL-암호화된 응답을 암호 해독 및 캐싱하고, 이러한 응답을 클라이언트에 서빙할 때, 이러한 응답을 재암호화하여 안전하게 요청 클라이언트에 전달할 수 있으며, 이에 따라 단대단 암호화가 가능하고, 따라서 매우 다양한 웹 트래픽에의 캐싱의 응용성이 증가된다.
- <715> 일 실시예에서, 어플라이언스는 또한 SSL 가상 사설망(SSL VPN)에서 엔드포인트의 역할을 할 수 있다. 특히, 어플라이언스는 사설 데이터 통신망 내의 자원에 대한 프록시 SSL 엔드포인트의 역할을 하여, 클라이언트로부터 수신된 암호화된 패킷을 암호 해독한 다음에 암호화된 패킷 내의 주소 정보에 기초하여 적절한 목적지 서버 자원으로 전송할 수 있다. 클라이언트와 게이트웨이 사이에 개설된 데이터 통신 세션은 본 출원의 이전의 단락에서 설명된 바와 같은 암호화 엔드포인트의 역할을 하는 게이트웨이로 암호화될 수 있다. 설명한 바와 같이, 클라이언트는 암호화된 데이터 통신 세션을 개설하기 위해 SSL(Secure Sockets Layer), IPSec, 또는 일부 다른 암호화 방법을 이용할 수 있으며, 이 방법으로 클라이언트 상의 인터셉트 메커니즘은 클라이언트 브라우저가 목적지 서버 또는 목적지 네트워크와 직접 통신하고 있다고 생각하게 만들면서 트래픽을 게이트웨이에 보낸다. 이러한 실시예에서, 암호화된 데이터 통신 세션은 게이트웨이에서 종료될 수 있으며, 게이트웨이는 또한 여기서 설명한 바와 같이 통합된 캐시를 포함한다. 이 방식으로 캐싱 기능은 SSL VPN 기능에 통합될 수 있다.
- <716> 게이트웨이는 요청에 대해 응용 가능 AAA 정책을 또한 수행할 수 있고, 따라서, 게이트웨이는 적절히 인증된 클라이언트에게만 캐싱된 객체를 서빙할 뿐만 아니라, 특별한 캐싱된 객체를 액세스하도록 권한 검증된 사용자에게 대한 요청만을 허용한다. 이는, 게이트웨이의 액세스 제어 정책이 캐시가 어떤 특별한 요청을 접하기 전에 시행되는 방식으로 캐시가 통합되기 때문에 가능하다. 따라서, 캐싱된 객체는 캐시 자체가 인증 및 권한 검증 수행할 필요 없이 액세스 제어의 이익을 얻는다. 이러한 다른 기능과의 캐시의 통합을 통해, 캐시 자체는 오늘날의 네트워크를 통과하는 다양한 데이터의 처리에 보다 효율적이고 보다 효과적이다. 일 실시예는 또한 캐시 기능의 이익을 보다 넓은 어레이의 웹 트래픽에 도입함으로써 전체 네트워크 성능의 효율을 향상시킬 수 있다.
- <717> 일 실시예에 따라 위에서 설명된 통합 모드의 일부 다른 특유한 결과는 다음과 같다. 한 결과는 사전 압축된 데이터를 캐싱하여 압축-인식 클라이언트에 서빙하는 능력이다. 다른 결과는 액세스 제어형 데이터를 캐싱하는 능력이다. 또 다른 결과는 캐시의 스칼라빌리티를 제공하기 위해 외부 캐시와 함께 일하는 능력이다. 캐시가 게이트웨이에서 리디렉션 및 트래픽 관리 능력과 통합되기 때문에, 외부 캐시는 캐싱의 제2-타이어(tier)를 제공하도록 배치될 수 있으며, 이에 따라 캐싱의 용량(및 이익)이 크게 늘어난다. 일 실시예를 통해, 이 용량은 캐시 리디렉션 정책을 명백하게 수행해야 하는 캐시 모듈 자체 없이도 생성된다.
- <718> 성능 면에서, 위에서 설명된 바와 같은 캐시를 통합함으로써, 캐시의 프로세서는 네트워크 상의 노드로서 작용하는 캐시가 전통적으로 수행하도록 요구되는 다양한 연결 처리 태스크를 수행하지 않아도 되며, 따라서 캐싱 기능을 최고 성능 수준으로 수행할 수 있다. 실제로, 압축된 데이터의 캐싱을 가능하게 함으로써, 캐시는 훨씬 더 효율적으로 기능할 수 있고, 사용자가 훨씬 더 높은 성능을 실현할 수 있도록 한다.
- <719> 이 애플리케이션에서 이전에 언급한 바와 같이, 효율은 캐시가 다른 네트워크 서비스, 및 부하 밸런싱 기술, 암

호화, AAA, 압축 및 다른 유형의 가속화 및 패킷 처리를 포함한 기술과 통합되는 방식의 결과로서 얻어진다. 결과적으로, 전통적인 통합 모드에 의해 도입되는 처리 복제 및 다른 비효율성이 회피된다. 불필요한 복사 및 콘텍스트 교환에 의해 초래되는 이들 비효율성은 디바이스에 의해 수신되는 각각의 객체가 메시지에 복사된 다음에 해당 애플리케이션에 의한 처리 전에 프로세서 메모리에 입력되어야 하기 때문에 생긴다. 다음에, 요청은 캐시에 의한 처리를 위해 객체 또는 패킷 수준으로 다시 복사되어야 하며, 이에 부가적인 메모리 복사가 도입된다. 대조적으로, 일 실시예는 OS 또는 커널 수준에서 통합을 실행하며, 이에 따라 캐시는, 캐시가 이러한 데이터 구조에 관련시키고 그 데이터 구조를 처리함에 있어 다른 애플리케이션 및/또는 프로세스와 동일한 상태를 갖고 모든 프로세스들이 동일한 데이터 구조로 일을 함에 따라 이러한 부가적인 메모리 복사에 대한 필요성이 회피되는 데이터 구조를 가짐에 따라, 객체 상에서 동작할 수 있다. 결과적으로 통합은 보다 효율적이다.

<720> a. 데이터 통신망에서의 순리적 유효화(Proactive Validation)를 이용한 캐싱

<721> 웹 객체들은 시간에 따라 변할 수 있기 때문에, 각각의 잠재적으로 캐싱가능 객체는 유용한 수명 또는 "신규성(freshness)"을 갖는다고 한다. 신규성의 개념은 원래 콘텐츠를 발생한 애플리케이션 서버가 또한 이러한 객체가 이 객체를 저장할 수 있는 캐시에 의해 서빙될 수 있는 기간을 결정한다는 사실을 말한다. 캐시는 메모리에 저장된 객체의 복사가 여전히 "신규(fresh)"한지의 여부 또는 캐시가 원래 서버로부터 객체의 새로운 복사를 검색해야 하는지를 판단할 수 있어야 한다. 일 실시예는 객체 신규성을 보장하는 새로운 접근법을 구현한다. 많은 종래 캐시 구현예들은 선정된 스케줄로 원래 서버부터 콘텐츠를 폐칭함으로써 캐싱된 콘텐츠를 신규하게 유지하려고 한다. 원래 서버로부터의 콘텐츠 폐칭은, 전형적으로 다음의 접근법들 중 하나 또는 둘에 기초하여, 즉 (i) 규칙적인 지정 간격으로 또는 (ii) 콘텐츠가 만료되려는 때, 캐시 관리자가 설정한 시간에 일어난다.

<722> 전형적으로 상기 흔히 이용되는 접근법들과 연관된 2가지 문제점이 있다. 먼저, 불필요한 처리 부하가 원래 서버에 가해지는데, 이는, 콘텐츠가 결국 클라이언트에 서빙되어야 하는지에 상관없이 리프레쉬(이러한 리프레쉬가 지정된 간격으로 일어나든 아니면 콘텐츠가 만료되는 순간에 일어나든)를 요청하는 캐시에 콘텐츠를 서버가 제공해야 하기 때문이다. 둘째, 캐시는, 캐시가 리프레쉬되어야 하는 요소들의 트랙 및 리프레쉬되어야 하는 시간을 유지해야 하기 때문에 발생하는 과잉 처리 오버헤드에 기초하여 부가적인 프로세서 부하를 초래한다.

<723> 일 실시예에 따른 캐시는 새로운 사전 폐칭 접근법을 이용하여 상기한 문제점을 해결한다. 콘텐츠 폐칭은 미리 정의된 스케줄에 따라 또는 콘텐츠 만료 직전에는 수행되지 않는다. 대신에, 일 실시예는 두 조건, 즉 (1) 클라이언트가 지정된 콘텐츠를 요청하고, (2) 그 콘텐츠가 '만료 직전'이다라는 두 조건이 충족될 때에만 사전 폐칭을 수행한다.

<724> 이 접근법은 위에서 설명된 두 문제점을 처리한다. 순리적인 재유효화(Pro-active revalidation)는 이러한 콘텐츠가 활발히 액세스되는 경우에만 원래 서버로부터의 콘텐츠 리프레싱 요청을 발생할 가능성이 매우 높다. 이는 원래 서버 상에서 '불필요한' 부하의 양을 최소화시킨다. 위에서 설명한 바와 같이, 캐시가 클라이언트에 결국 서빙되는(또는 캐시의 민감성에 따라 단지 드물게 서빙되는) 객체의 리프레쉬를 요청하는 경우, 캐시는 자기 자신의 자원뿐만 아니라 원래 서버의 자원을 비효율적으로 이용하게 된다. 일 실시예는 활발히 액세스되는 그 콘텐츠만을 요청함으로써 캐시 자원 및 서버 자원의 비효율적인 이용을 회피한다. 접근법은 또한, 동일한 이유로, 사전 폐칭에 사용되는 대역폭을 감소시키며, 그러므로 전통적인 접근법들보다 네트워크 자원을 효율적으로 이용한다.

<725> 또한, 일 실시예는 원래 서버로부터 객체의 리프레쉬를 요청할지를 판단하기 위해 캐싱된 객체 자체에 포함된 만료 정보를 이용한다. 이러한 만료 정보는 전형적으로 해당 객체의 헤더에 포함된다. 따라서 이 실시예는 리프레쉬 스케줄을 추적하는 테이블을 유지할 것을 캐시에게 요구하는 많은 전통적인 접근법들과는 달리 폐칭에 대한 부가적인 정보를 노리는 비효율성을 회피한다. 또한, '요구-기반' 사전 폐칭 기술을 이용하면 사전 폐칭에 고유한 이익을 개진시킨다. 이 기술은 빈번하게 액세스되는 객체의 캐시 부적중의 횟수를 감소시키는데, 이는 이러한 객체가 만료 직전에 순리적인 재유효화를 겪을 가능성이 높기 때문이다. 이 기술은 또한 수요가 많은 큰 응답이 만료될 때 일어날 수 있는 원래 서버에 대한 트래픽의 서지를 방지할 수 있다. 전통적인 접근법에서, 이러한 콘텐츠에 대한 모든 요청은 캐시 콘텐츠가 만료되었기 때문에 캐시를 부적중시키고 원래 서버로 전송된다. 대조적으로, 일 실시예에서, 캐시 메모리의 콘텐츠는 일반적으로 만료 직전에 리프레쉬될 것이며, 그러므로 캐시가 리프레싱되고 있는 동안에 캐시 부적중이 일어나던 상황이 훨씬 덜 생길 수 있다.

<726> 일 실시예에서, 사전 폐칭의 공세(aggressiveness)는, 콘텐츠가 만료 직전이라고 판단되는 경우 만료 전의 기간의 길이 및 해당 객체의 캐시에 의해 리프레쉬를 트리거하는데 필요한 클라이언트 요청들의 수를 조절함으로써, 제어될 수 있다.

- <727> b. "네거티브 셀"을 이용한 큰 캐싱 불가능 응답의 처리의 최적화
- <728> 일 실시예에 따라, 캐시는 객체 적중률을 향상시키기 위해 지정된 사이즈 이상인 객체를 인식하여 저장하지 않는다. 캐시는 전형적으로 캐싱된 객체 저장을 위한 제한된 메모리 공간을 갖고 있으며, 그러므로 할당된 메모리 공간을 초과하는 일정한 응답은 결국 캐싱 불가능한 것으로 거절되어 캐시에 의해 저장되지 않는다. 전통적인 캐시의 경우, 캐시는 캐시 메모리에 큰 응답을 저장하는 시도를 하며, 일단 캐시가 응답 사이즈가 미리 정의된 최대 사이즈를 초과한다는 사실을 인식하면 응답 저장을 중단할 뿐이다. 전통적인 캐시는 이러한 응답에 대한 요청이 서버로부터 캐시에 의해 수신될 때마다 큰 응답을 캐시하는 시도를 반복적으로 행한다. 각각의 경우에, 캐시는 객체가 메모리 공간을 초과하여 캐싱 불가능한 것으로 판단해야 할 것이며, 따라서, 이는 분명히 비효율적인 접근법이다.
- <729> 일 실시예에 따라, 캐시는 이러한 응답을 저장하는 노력을 피하기 위해 최적화를 이용한다. 캐시가 응답 사이즈로 인해 캐싱 불가능으로 된 응답을 검출할 때마다, "네거티브 셀(negative cell)"로 칭하는 데이터 구조에 대응 요청에 관한 표기를 저장한다. 표기는 요청이 캐싱 불가능함을 나타낸다. 클라이언트가 동일 객체를 요청하면, 요청은 데이터 구조에 저장된 제1 요청에 관한 표기와 매칭된다. 매칭으로, 캐시는 응답을 캐싱하려 하지 않을 것이고, 대신에 요청이 캐시를 완전히 바이패스시킬 것이다.
- <730> 네거티브 셀이 캐시 내에 남아 있어야 하는 기간을 지정하는데 필요한 사용자 구성은 없다. 사실상, 사용자들은 이 특별한 메커니즘이 이용되고 있음을 알지도 못한다. 일 실시예에서, 캐시는, 큰 응답에 관한 "네거티브 정보"를 캐싱하기 위해서 큰 응답을 캐싱하는데 이용한다는 규칙적인 만료 정보를 이용한다.
- <731> 4. 클라이언트측 가속화
- <732> 일 실시예에서, 클라이언트측 가속화 프로그램은 클라이언트의 서버와의 통신 및/또는 서버에 의해 제공되는 애플리케이션의 액세스와 같은 서버에의 액세스를 가속화, 개선 또는 향상시키기 위해 하나 이상의 가속화 기술을 수행할 수 있다. 이제 도 40a를 참조하면, 가속화 프로그램(6120)을 가진 클라이언트(6205)가 도시되어 있다. 간단히 살펴보면, 클라이언트(6205)는 커널 모드(6202) 및 사용자 모드(6202)를 가진 운영체제, 및 하나 이상의 계층(6210a-6210b)을 가진 네트워크 스택(6210)을 가진 컴퓨팅 디바이스(6100) 상에서 동작한다. 클라이언트(6205)는 이전에 설명된 모든 클라이언트(10)를 포함할 수 있다. 하나의 클라이언트(6205)만이 도시되어 있지만, 임의의 개수의 클라이언트(10)가 클라이언트(6205)를 포함할 수 있다. 클라이언트(6205)는 하나 이상의 애플리케이션(6220a-6220n)을 설치 및/또는 실행할 수 있다. 일부 실시예들에서, 하나 이상의 애플리케이션(6220a-6220n)은 네트워크 스택(6210)을 통해 네트워크에 전달될 수 있다. 애플리케이션들(6220n) 중 하나는 또한 제1 프로그램(6222), 예컨대, 일부 실시예들에서 가속화 프로그램(6120)을 설치 및/또는 실행하는데 사용될 수 있는 프로그램을 포함할 수 있다.
- <733> 클라이언트(6205)의 네트워크 스택(6210)은 네트워크에의 연결 및 네트워크와의 통신을 제공하기 위해 임의 유형 및 형태의 소프트웨어, 또는 하드웨어, 또는 그 조합을 포함할 수 있다. 일 실시예에서, 네트워크 스택(6210)은 네트워크 프로토콜 스위트에 대한 소프트웨어 구현을 포함한다. 네트워크 스택(6210)은 하나 이상의 네트워크 계층, 예컨대, 당업자가 인식하고 이해하는 OSI(Open Systems Interconnection) 통신 모델의 네트워크 계층을 포함할 수 있다. 이로써, 네트워크 스택(6210)은 OSI 모델의 계층, 즉 1)물리 링크 계층, 2)데이터 링크 계층, 3)네트워크 계층, 4)전송 계층, 5)세션 계층, 6)프리젠테이션 계층, 및 7)애플리케이션 계층을 위한 임의의 유형 및 형태의 프로토콜을 포함할 수 있다. 일 실시예에서, 네트워크 스택(310)은 일반적으로 TCP/IP로 칭하는 인터넷 프로토콜(IP)의 네트워크 계층 프로토콜 상의 전송 제어 프로토콜(TCP)을 포함할 수 있다. 일부 실시예들에서, TCP/IP 프로토콜은 이더넷 프로토콜 상에서 운반될 수 있고, 이더넷 프로토콜은 IEEE 802.3에 의해 지원되는 프로토콜과 같은 IEEE WAN 또는 LAN 프로토콜의 패밀리를 포함할 수 있다. 일부 실시예들에서, 네트워크 스택(6210)은 IEEE 802.11 및/또는 모바일 인터넷 프로토콜과 같은 임의의 유형 및 형태의 무선 프로토콜을 포함한다.
- <734> TCP/IP 기반 네트워크를 고려하여, 메시징 애플리케이션 프로그래밍 인터페이스(Messaging Application Programming Interface; MAPI)(이메일), 파일 전송 프로토콜(FTP), 하이퍼텍스트 전송 프로토콜(HTTP), 공통 인터넷 파일 시스템(Common Internet File System; CIFS) 프로토콜(파일 전송), 독립 컴퓨팅 아키텍처(Independent Computing Architecture; ICA) 프로토콜, 원격 데스크톱 프로토콜(Remote Desktop Protocol; RDP), 무선 애플리케이션 프로토콜(WAP), 모바일 IP 프로토콜, 및 VoIP(Voice Over IP) 프로토콜을 포함한 TCP/IP 기반 프로토콜이 사용될 수도 있다. 다른 실시예에서, 네트워크 스택(210)은 수정된 전송 제어 프로토콜과 같은 임의의 유형 및 형태의 전송 제어 프로토콜, 예컨대 트랜잭션(Transaction) TCP(T/TCP), 선택 확인응답

(selection acknowledgements)을 이용한 TCP(TCP-SACK), 큰 윈도우를 이용한 TCP(TCP-LW), TCP-베가스(Vegas) 프로토콜과 같은 폭주 예측 프로토콜, 및 TCP 스푸핑(spoofing) 프로토콜을 포함한다. 다른 실시예들에서, UDP(user datagram protocol) 오버 IP(UDP over IP)와 같은 임의 유형 및 형태의 UDP가, 예컨대 음성 통신 또는 실시간 데이터 통신을 위해 네트워크 스택(6210)에 의해 사용될 수 있다.

<735> 또한, 네트워크 스택(6210)은 TCP 구동기 또는 네트워크 계층 구동기와 같은 하나 이상의 계층을 지원하는 하나 이상의 네트워크 구동기를 포함할 수 있다. 네트워크 구동기는 컴퓨팅 디바이스(100)의 운영체제의 일부로서 또는 컴퓨팅 디바이스(6100)의 네트워크 인터페이스 카드 또는 다른 네트워크 액세스 구성 요소의 일부로서 포함될 수 있다. 일부 실시예들에서, 네트워크 스택(6210)의 네트워크 구동기는 여기서 설명되는 기술을 지지하여 네트워크 스택(6210)의 맞춤 또는 수정된 부분을 제공하도록 맞춤 제작, 수정 또는 적응될 수 있다. 다른 실시예들에서, 가속화 프로그램(6120)은 클라이언트(205)의 운영체제에 의해 설치 또는 제공되는 네트워크 스택(6210)과 함께 동작하거나 일을 하도록 설계 및 구성된다.

<736> 네트워크 스택(6210)은 클라이언트(6205)의 네트워크 통신에 관련된 정보 및 데이터를 수신, 획득, 제공 또는 액세스하기 위한 임의 유형 및 형태의 인터페이스를 포함한다. 일 실시예에서, 네트워크 스택(6210)에 대한 인터페이스는 애플리케이션 프로그래밍 인터페이스(API)를 포함한다. 인터페이스는 또한 기능 호출, 후킹 또는 필터링 메커니즘, 이벤트 또는 호출 백 메커니즘, 또는 임의 유형의 인터페이싱 기술을 포함할 수 있다. 인터페이스를 통한 네트워크 스택(6210)은 네트워크 스택(6210)의 기능 또는 동작에 관련된 객체와 같은 임의 유형 및 형태의 데이터 구조를 수신 및 제공할 수 있다. 예컨대, 데이터 구조는 네트워크 패킷 또는 하나 이상의 네트워크 패킷에 관련된 정보 및 데이터를 포함할 수 있다. 일부 실시예들에서, 데이터 구조는 전송 계층의 네트워크 패킷과 같은 네트워크 스택(6210)의 프로토콜 계층에서 처리된 네트워크 패킷의 일부를 포함한다. 일부 실시예들에서, 데이터 구조(6225)는 커널-레벨 데이터 구조를 포함하나, 다른 실시예들에서, 데이터 구조(6225)는 사용자-모드 데이터 구조를 포함한다. 커널-레벨 데이터 구조는 커널-모드(6202)로 동작하는 네트워크 스택(6210)의 일부에 관련된 또는 획득된 데이터 구조, 또는 커널-모드(6202)로 구동되는 네트워크 구동기 또는 다른 소프트웨어, 또는 운영체제의 커널-모드로 구동 또는 동작하는 서비스, 프로세스, 태스크, 스레드(thread) 또는 다른 실행 가능 명령에 의해 획득 또는 수신되는 데이터 구조를 포함할 수 있다.

<737> 부가적으로, 네트워크 스택(6210)의 일부는 커널-모드(6202), 예컨대, 데이터 링크 또는 네트워크 계층에서 실행 또는 동작할 수 있고, 다른 부분은 네트워크 스택(6210)의 애플리케이션 계층과 같은 사용자-모드(6203)로 실행 또는 동작할 수 있다. 예컨대, 네트워크 스택의 제1 부분(6210a)은 네트워크 스택(6210)에 대한 사용자-모드 액세스를 애플리케이션(6220a-6220n)에 제공할 수 있는 반면에, 네트워크 스택(6210)의 제2 부분(6210b)은 네트워크에 대한 액세스를 제공한다. 일부 실시예들에서, 네트워크 스택의 제1 부분(6210a)은 계층 5-7과 같은 네트워크 스택(6210)의 하나 이상의 상부 계층을 포함할 수 있다. 다른 실시예들에서, 네트워크 스택(6210)의 제2 부분(6210b)은 계층 1-4와 같은 하나 이상의 하부 계층을 포함한다. 네트워크 스택(6210)의 제1 부분(6210a) 및 제2 부분(6210b)의 각각은 네트워크 스택(6210)의 일부를, 하나 이상의 네트워크 계층, 사용자-모드(6203), 커널-모드(6202), 또는 그 조합에, 또는 네트워크 계층의 일부 또는 네트워크 계층에 대한 인터페이스 지점, 또는 사용자-모드(6203) 및 커널-모드(6203)의 일부 또는 그 인터페이스 지점에 포함할 수 있다.

<738> 본 발명의 가속화 프로그램(6120)은 소프트웨어, 하드웨어, 또는 소프트웨어와 하드웨어의 조합을 포함할 수 있다. 일부 실시예들에서, 가속화 프로그램(6120)은 여기서 설명한 바와 같이 기능 및 동작을 실행 또는 제공하도록 구성 및 설계된 임의 유형 및 형태의 실행 가능 명령을 포함한다. 일부 실시예들에서, 가속화 프로그램(6120)은 임의 유형 및 형태의 애플리케이션, 프로그램, 서비스, 프로세스, 태스크 또는 스레드를 포함한다. 일 실시예에서, 가속화 프로그램(6120)은 네트워크 스택(6210)과 인터페이싱하여 일을 하도록 구성 및 설계된 네트워크 구동기와 같은 구동기를 포함한다. 가속화 프로그램(6120)의 실행 가능 명령의 로직, 기능, 및/또는 동작은 가속화 기술, 즉 1)멀티-프로토콜 압축(6238), 2)전송 제어 프로토콜 폴링(6224), 3)전송 제어 프로토콜 다중화(6226), 4)전송 제어 프로토콜 버퍼링(6228), 및 5)캐시 관리자(6232)를 통한 캐싱중 하나 이상을 수행할 수 있으며, 이에 대해서는 보다 상세히 후술된다. 부가적으로, 가속화 프로그램(6120)은 클라이언트(6205)에 의해 수신 및/또는 전송된 통신의 암호화(6234) 및/또는 암호 해독을 수행할 수 있다. 일부 실시예들에서, 가속화 프로그램(6120)은 또한 클라이언트(6205)와 서버(30)와 같은 다른 컴퓨팅 디바이스(6100)간의 터널링을 수행한다. 다른 실시예들에서, 가속화 프로그램(6120)은 서버(30)에의 가상 사설망 연결을 제공한다.

<739> 일부 실시예들에서, 가속화 프로그램(6120)은 네트워크 스택(6210)의 하나 이상의 계층에서, 예컨대 전송 계층에서, 동작한다. 일 실시예에서, 가속화 프로그램(6120)은, 필터 구동기, 후킹 메커니즘, 또는 전송 구동기 인터페이스(transport driver interface; TDI)와 같은 네트워크 스택의 전송 계층에 인터페이싱되는 임의의 형태 및

유형의 적합한 네트워크 구동기 인터페이스를 포함한다. 일부 실시예들에서, 가속화 프로그램(6120)은 상기 전송 계층과 같은 제1 프로토콜 계층, 및 전송 프로토콜 계층 상의 어떤 계층, 예컨대, 애플리케이션 프로토콜 계층과 같은 다른 프로토콜 계층에 인터페이스된다. 일 실시예에서, 가속화 프로그램(6120)은 NDI(Network Driver Interface Specification)를 따르는 구동기, 즉 NDIS 구동기를 포함할 수 있다. 다른 실시예에서, 가속화 프로그램(6120)은 민(min)-필터 또는 미니-포트 구동기를 포함할 수 있다. 일 실시예에서, 가속화 프로그램(6120), 또는 그 일부는 커널-모드(6202)로 동작한다. 다른 실시예에서, 가속화 프로그램(6120) 또는 그 일부는 사용자-모드(6203)로 동작한다. 일부 실시예들에서, 가속화 프로그램(6120)의 일부는 커널-모드(6202)로 동작하나, 가속화 프로그램(6120)의 다른 부분은 사용자-모드(6203)로 동작한다. 다른 실시예들에서, 가속화 프로그램(6120)은 사용자-모드(6203)로 동작하나, 예컨대 커널-레벨 데이터 구조(6225)를 얻기 위해 커널-모드 구동기, 프로세스, 서비스, 태스크 또는 운영체제의 일부와 인터페이스된다. 다른 실시예들에서, 가속화 프로그램(6120)은 애플리케이션(6220a-6220n)과 같은 사용자-모드 애플리케이션 또는 프로그램이다.

<740> 가속화 프로그램(6120)은 네트워크 스택(6210)의 다른 프로토콜 계층에 투명한 방식으로 프로토콜 계층에서 동작하거나 프로토콜 계층과 인터페이스할 수 있다. 예컨대, 일 실시예에서, 가속화 프로그램(6120)은 전송 계층 아래의 프로토콜 계층, 예컨대 네트워크 계층, 및 전송 계층 위의 프로토콜 계층, 예컨대 세션, 프리젠테이션 또는 애플리케이션 계층 프로토콜에 투명하게 네트워크 스택(6210)의 전송 계층과 함께 동작하거나 그 네트워크 스택(6210)의 전송 계층에 인터페이스된다. 이에 따라, 네트워크 스택(6210)의 다른 프로토콜 계층은 원하는 대로 그리고 가속화 프로그램(6120)을 이용하기 위한 수정 없이 동작할 수 있다. 이로써, 가속화 프로그램(6120)은 TCP/IP 상의 애플리케이션 계층 프로토콜과 같은 전송 계층에 의해 운반행되는 프로토콜을 통해 제공되는 통신을 가속화하기 위해 전송 계층과 인터페이스할 수 있다.

<741> 또한, 가속화 프로그램(6120)은 애플리케이션(6220a-6220n), 클라이언트(6205)의 사용자, 및 클라이언트(6205)와 통신하는 서버와 같은 다른 컴퓨팅 디바이스에 투명한 방식으로 네트워크 스택(6210)에서 동작하거나 그 네트워크 스택(6210)과 인터페이스될 수 있다. 가속화 프로그램(6120)은 가속화 프로그램(6120)이 애플리케이션(6220a-6220n)의 수정 없이 애플리케이션(6220a-6220n)의 통신을 가속시킬 수 있는 방식으로 클라이언트(6205) 상에 설치 및/또는 실행될 수 있다. 일부 실시예들에서, 클라이언트(6205)와 통신하는 클라이언트(6205)의 사용자 또는 컴퓨팅 디바이스는 가속화 프로그램(6120)의 존재, 실행 또는 동작을 알지 못한다. 이로써, 일부 실시예들에서, 가속화 프로그램(6120)은 애플리케이션(6220a-6220n), 클라이언트(6205)의 사용자, 서버와 같은 다른 컴퓨팅 디바이스, 또는 가속화 프로그램(6120)에 의해 인터페이스되는 프로토콜 계층 위 및/또는 아래의 프로토콜 계층에 투명하게 설치, 실행, 및/또는 동작된다.

<742> 일부 실시예들에서, 가속화 프로그램(6120)은 가속화 기술(6224, 6226, 6228, 6232) 중 하나 이상을 통합된 방식 또는 형식으로 수행한다. 일 실시예에서, 가속화 프로그램(6120)은 네트워크 스택(6210)의 전송 프로토콜 계층에서 통신을 인터셉트하고, 후킹하고, 필터링하고, 또는 수신하기 위해 임의의 유형 및 형태의 메커니즘을 포함한다. 전송 계층에서 클라이언트(6205)의 네트워크 패킷을 인터셉트하고 커널-레벨 데이터 구조(6225)와 같은 데이터 구조를 통해 전송 계층에서 네트워크 스택(6210)에 인터페이스함으로써, 가속화 프로그램(120)은 전송 제어 프로토콜(TCP) 버퍼링, TCP 폴링 및 TCP 다중화와 같은 전송 계층 관련 가속화 기술을 네트워크 패킷에 대해 수행할 수 있다. 부가적으로, 가속화 프로그램(6120)은 전송 계층 프로토콜의 네트워크 패킷의 페이로드로서 운반되는 프로토콜, 또는 다수의-프로토콜 상에서 압축(6225)을 수행할 수 있다.

<743> 일 실시예에서, 가속화 프로그램(6120)은 커널-레벨 데이터 구조(6225)를 이용하며, 이 구조는 하나 이상의 네트워크 패킷, 예컨대, 클라이언트(6205)로부터의 요청 또는 서버로부터의 응답을 포함하는 네트워크 패킷의 어떤 부분에 대한 액세스를 제공한다. 일 실시예에서, 커널-레벨 데이터 구조는 원하는 가속화 기술을 수행하기 위해 가속화 프로그램(6120)에 의해 사용될 수 있다. 일 실시예에서, 가속화 프로그램(6120)은 커널-레벨 데이터 구조(6225) 사용시 커널 모드(6202)로 구동되나, 다른 실시예에서는, 가속화 프로그램(6120)은 커널-레벨 데이터 구조(6225)를 사용할 때 사용자-모드(6203)로 구동된다. 일부 실시예들에서, 커널-레벨 데이터 구조는 제2 커널-레벨 데이터 구조 또는 어떤 원하는 사용자-레벨 데이터 구조로 복사되거나 전달될 수 있다. 가속화 프로그램(6120)은 일반적으로 사용자-모드(6203)로 동작하는 제1 부분 및 커널-모드(6202)로 동작하는 제2 부분을 가진 것으로 도 40a에 도시되어 있지만, 일부 실시예들에서, 가속화 프로그램(6120)의 어떤 부분은 사용자-모드(6203) 또는 커널-모드(6202)로 구동될 수 있다. 일부 실시예들에서, 가속화 프로그램(6120)은 사용자-모드(6203)로만 동작할 수 있으나, 다른 실시예들에서, 가속화 프로그램(6120)은 커널-모드(6202)로만 동작할 수 있다.

<744> 또한, 네트워크 스택(6210)의 전송 계층에서 인터셉트하거나 커널-레벨 데이터 구조(6225)를 통해 네트워크 패

킷에 대한 액세스를 획득함으로써, 가속화 프로그램(6120)은 단일 인터페이스 지점에서 또는 단일 실행 지점에서 또는 가속화 프로그램(6120)의 실행 가능 명령을 실행하는 시간에 복수의 가속화 기술을 수행 또는 적용할 수 있다. 예컨대, 일 실시예에서, 가속화 프로그램(6120)의 명령들의 기능 또는 세트에서, 복수의 가속화 기술은 예컨대 가속화 기술을 수행하도록 구성 및 설계된 한 세트의 실행 가능 명령을 호출함으로써 실행될 수 있다. 일부 실시예들에서, 가속화 프로그램(6120)은 하나의 인터페이스 지점, 즉 실행 장소에서, 또는 한 세트의 명령에서, 1) 멀티-프로토콜 압축(6238), 2) 전송 제어 프로토콜 폴링(6224), 3) 전송 제어 프로토콜 다중화(6226), 4) 전송 제어 프로토콜 버퍼링(6228), 및 5) 캐시 관리자(6232)를 통한 캐싱, 일부 실시예들에서는 암호화(6234)를 제공하도록 설계 및 구성된 프로그램, 서비스, 프로세스, 태스크, 쓰레드, 또는 실행 가능 명령에 대한 하나 이상의 애플리케이션 프로그래밍 인터페이스(API)를 호출한다.

<745> 가속화 프로그램(6120)의 실행 가능 명령 내의 한 장소 또는 위치 또는 전송 계층과 같은 네트워크 스택(6210)의 하나의 프로토콜 계층에서 복수의 가속화 기술을 실행함으로써, 이들 가속화 기술의 통합은 보다 효율적으로 그리고 효과적으로 수행된다. 일 양태에서, 프로세스들 간의 컨텍스트 교환의 횟수가 감소될 수 있을 뿐만 아니라, 사용되는 데이터 구조 또는 필요한 또는 달리 사용되는 메모리 내의 데이터 구조의 복사의 개수가 감소된다. 부가적으로, 가속화 기술들의 동기화 및 가속화 기술들 간의 통신은 가속화 프로그램(6120)의 한 세트의 실행 가능 명령에서 보다 효율적으로, 예컨대 단단한 결합 방식으로 수행될 수 있다. 이로써, 가속화 기술의 순서, 수행할 기술, 및 기술들 간에 공유 또는 전달될 데이터 및 정보에 관한 로직, 규칙, 기능 또는 동작은 보다 효율적으로 수행될 수 있다. 가속화 프로그램(6120)은 전송 계층에서 TCP 패킷을 인터셉트하고, 커널-레벨 데이터 구조(6225)를 통해 TCP 패킷의 페이로드를 획득한 다음에, 원하는 가속화 기술들을 원하는 순서로 수행할 수 있다. 예컨대, 네트워크 패킷은 먼저 압축된 다음에 캐싱될 수 있다. 다른 예에서, 압축된 캐싱된 데이터는 버퍼링, 폴링, 및/또는 다중화된 TCP 연결을 통해 서버에 전달될 수 있다.

<746> 일부 실시예들에서 도 40a를 계속 참조하면, 제1 프로그램(6222)은 가속화 프로그램(6120)을 자동으로, 사일런트(silent)하게, 투명하게 또는 달리 설치 및/또는 실행하는데 사용될 수 있다. 일 실시예에서, 제1 프로그램(6222)은 플러그인 구성 요소, 예컨대 애플리케이션(6220a-6220n)에 로딩되어 실행되는 ActiveX 제어 또는 자바 제어 또는 스크립트를 포함한다. 예컨대, 제1 프로그램은, 애플리케이션(6220)의 메모리 공간 또는 컨텍스트에서와 같이, 웹 브라우저 애플리케이션(6220)에 의해 로딩되어 구동되는 ActiveX 제어를 포함한다. 다른 실시예에서, 제1 프로그램(6222)은 브라우저와 같은 애플리케이션(6220a-6220n)에 로딩되거나 애플리케이션(6220a-6220n)에 의해 구동되는 한 세트의 실행 가능 명령을 포함한다. 일 실시예에서, 제1 프로그램(6222)은 가속화 프로그램(6120)을 설치하기 위해 설계 및 구성된 프로그램을 포함한다. 일부 실시예들에서, 제1 프로그램(6222)은 다른 컴퓨팅 디바이스로부터 네트워크를 통해 가속화 프로그램(6120)을 획득, 다운로드, 또는 수신한다. 다른 실시예에서, 제1 프로그램(6222)은 클라이언트(6205)의 운영체제 상에 네트워크 구동기와 같은 프로그램을 설치하기 위한 설치러 프로그램 또는 플러그 및 플레이 관리자이다.

<747> 다른 실시예들에서, 제1 프로그램(6222)은 여기서 B절에 설명된 기능들을 모두 포함할 수 있다. 일 실시예에서, 제1 프로그램(6222)은 수집 에이전트(404)를 포함할 수 있다. 다른 실시예에서, 제1 프로그램은 수집 에이전트를 설치하기 위한 프로그램을 포함할 수 있다. 다른 실시예에서 제1 프로그램(6222)은 또한 컴퓨팅 환경(15)을 포함할 수 있다. 일 실시예에서 제1 프로그램(6222)은 컴퓨터 환경과 같은 실행 환경 또는 가상 실행 환경을 설치하기 위한 수단을 포함할 수 있다. 일 실시예에서 제1 프로그램(6222)은 이전에 설명한 바와 같이 애플리케이션 스트리밍 클라이언트(442)를 포함할 수 있다. 다른 실시예에서 제1 프로그램(6222)은 클라이언트(10) 상에서 실행될 애플리케이션을 포함할 수 있다.

<748> 다른 실시예들에서, 제1 프로그램(6222)은 가속화 기술과 같은 여기서 설명된 가속화 프로그램(6120)의 기능, 동작 및 로직을 용이하게 하거나 수행하기 위해 가속화 프로그램(6120)의 기능, 동작 및 로직의 일부를 포함할 수 있다. 일부 실시예들에서, 제1 프로그램(6222)은 어플라이언스 또는 서버와의 전송 계층 연결과 같은 연결 또는 SSL(Secure Socket Layer) 통신 세션과 같은 통신 세션을 개설하는데 사용된다. 일 실시예에서, 제1 프로그램(6222)은 가상 사설망 연결 및 통신 세션을 개설하거나 개설을 용이하게 하는데 사용된다.

<749> 도 40a에 도시된 바와 같은 가속화 프로그램(6120) 또는 클라이언트(6205)의 캐시 관리자(6232)는 서버(30)에 의해 서빙되는 객체 또는 동적으로 발생된 객체와 같은 임의 유형 및 형태의 콘텐츠의 캐시 액세스, 제어 및 관리를 제공하기 위해 소프트웨어, 하드웨어 또는 소프트웨어와 하드웨어의 조합을 포함할 수 있다. 캐시 관리자(6232)에 의해 처리 및 저장된 데이터, 객체 또는 콘텐츠는 생성 언어와 같은, 또는 프로토콜을 통해 전달되는 임의 포맷의 데이터를 포함할 수 있다. 일부 실시예들에서, 캐시 관리자(6232)는 다른 장소에 저장된 원래 데이터 또는 이전에 계산, 발생 또는 전송된 데이터를 복제하며, 원래 데이터는 캐시 메모리 소자의 관독에 비해

폐칭, 계산 또는 획득하는데 보다 긴 액세스 시간을 필요로 할 수 있다. 일단 데이터가 캐시 메모리 소자에 저장되면, 원래 데이터를 다시 폐칭하거나 다시 계산하기보다는 캐싱된 복사를 액세스함으로써 장래 이용이 행해질 수 있으며, 이에 따라 액세스 시간이 감소된다. 일부 실시예들에서, 캐시 메모리 소자는 클라이언트(6205)의 메모리에 데이터 객체를 포함할 수 있다. 다른 실시예들에서, 캐시 메모리 소자는 클라이언트(6205)에 의해 달리 사용되는 메모리보다 빠른 액세스 시간을 가진 메모리를 포함할 수 있다. 다른 실시예에서, 캐시 메모리 소자는 하드 디스크의 일부와 같은 클라이언트(6205)의 임의 유형 및 형태의 기억 장치 소자를 포함할 수 있다. 또 다른 실시예에서, 캐시 관리자(6232)는 데이터, 객체, 및 다른 콘텐츠를 캐싱하기 위해 메모리, 기억 장치, 또는 처리 유닛의 일부 또는 조합을 사용할 수 있다.

<750> 또한, 캐시 관리자(6232)는 여기서 설명된 기술의 실시예들을 수행하기 위해 로직, 기능, 규칙, 또는 동작을 포함할 수 있다. 예컨대, 캐시 관리자(6232)는 무효화 기간의 만료에 기초하여 또는 클라이언트(6205a-6205n) 또는 서버(30)로부터의 무효화 명령의 수신시 객체를 무효화하는 로직 또는 기능을 포함한다. 일부 실시예들에서, 캐시 관리자(6232)는 커널 공간(6202)에서, 다른 실시예들에서는 사용자 공간(6203)에서 실행되는 프로그램, 서비스, 프로세스 또는 태스크로서 동작할 수 있다. 일 실시예에서, 캐시 관리자(6232)의 제1 부분은 사용자 공간(6203)에서 실행되고 제2 부분은 커널 공간(6202)에서 실행된다. 일부 실시예들에서, 캐시 관리자(6232)는 임의 유형의 범용 프로세서(GPP), 또는 어떤 다른 유형의 통합된 회로, 예컨대 필드 프로그램머블 게이트 어레이(Field Programmable Gate Array; FPGA), 프로그램머블 로직 디바이스(Programmable Logic Device; PLD), 또는 주문형 반도체(Application Specific Integrated Circuit; ASIC)를 포함할 수 있다.

<751> 가속화 프로그램(6120) 또는 클라이언트(6205)의 암호화 엔진(6234)은 보안 관련 프로토콜, 예컨대 SSL 또는 TLS, 또는 관련 기능의 처리를 다루기 위한 로직, 비즈니스 규칙, 기능 또는 동작을 포함한다. 예컨대, 암호화 엔진(6234)은 클라이언트(6205)에 의해 전달된 네트워크 패킷 또는 그 일부를 암호화 및 암호 해독한다. 암호화 엔진(6234)은 또한 클라이언트(6205)를 위해 SSL 또는 TLS 연결을 셋업 또는 개설한다. 이로써, 암호화 엔진(6234)은 SSL 처리의 오프로딩 및 가속화를 제공한다. 일 실시예에서, 암호화 엔진(6234)은 클라이언트(6205) 및 서버와 같은 다른 컴퓨팅 디바이스 사이에 가상 사설망을 제공하기 위해 터널링 프로토콜을 사용한다.

<752> 도 40a를 계속 참조하면, 가속화 프로그램(6120) 또는 클라이언트(6205)의 멀티-프로토콜 압축 엔진(6238)은 클라이언트(6205)의 네트워크 스택(6210)에 의해 사용되는 프로토콜과 같은 네트워크 패킷의 하나 이상의 프로토콜을 압축하기 위한 로직, 비즈니스 규칙, 기능 또는 동작을 포함한다. 예컨대, 멀티-프로토콜 압축(6238)은 압축 및 압축 해제 유틸리티를 포함할 수 있고, 압축 및 압축 해제 유틸리티는 네트워크를 통해 전송될 데이터를 압축 및 압축 해제하기 위한 GZip 압축 및 압축 해제, 차동 압축 및 압축 해제, 또는 어떤 다른 독점적으로 또는 공개적으로 이용 가능한 유틸리티를 포함한다. 일 실시예에서, 멀티프로토콜 압축 엔진(6238)은 클라이언트(6205) 및 서버와 같은 다른 컴퓨팅 디바이스 사이에서 양방향으로, 메시징 애플리케이션 프로그래밍 인터페이스(MAPI)(이메일), 파일 전송 프로토콜(FTP), 하이퍼텍스트 전송 프로토콜(HTTP), 공통 인터넷 파일 시스템(CIFS) 프로토콜(파일 전송), 독립 컴퓨팅 아키텍처(ICA) 프로토콜, 원격 데스크톱 프로토콜(RDP), 무선 애플리케이션 프로토콜(WAP), 모바일 IP 프로토콜, 및 VoIP 프로토콜을 포함한 TCP/IP 기반 프로토콜을 압축한다. 다른 실시예들에서, 멀티-프로토콜 압축 엔진(238)은 하이퍼텍스트 생성 언어(HTML) 기반 프로토콜의 압축을 제공하고, 일부 실시예들에서, 생성 언어, 예컨대 확장성 생성 언어(XML)의 압축을 제공한다. 이로써, 멀티-프로토콜 압축 엔진(6238)은, 데스크톱 클라이언트를 통해, 예컨대, 오라클, SAP 및 시벨(Siebel), 및 심지어 포켓 PC와 같은 모바일 클라이언트 같은 기업 애플리케이션에 의해 공급되는 클라이언트와 같은 마이크로소프트 아웃룩 및 비-웹 씬 클라이언트를 통해, 사용자가 애플리케이션을 액세스하는 성능을 가속시킨다.

<753> 가속화 프로그램(6120)은 또한, 보다 상세히 후술되는 바와 같이, 버퍼링, 폴링 및 다중화의 전송 프로토콜 계층 가속화 기술을 포함한다. 이로써, 가속화 프로그램(6120)은 여기서 설명된 바와 같은 기술을 수행하기 위해 로직, 규칙, 기능 및 동작을 가진 임의 유형 및 형태의 실행 가능 명령을 포함한다. 가속화 프로그램(120)은 네트워크 스택(210)의 전송 계층에서 네트워크 스택(6210)을 통해 애플리케이션(6220a-6220n)에 의해 행해지는 전송 계층 애플리케이션 프로그래밍 인터페이스(API) 호를 인터셉트하고, 제어하고, 관리한다. 가속화 프로그램(6120)은 투명 방식으로 클라이언트(6205)의 요청에 응답함으로써, 클라이언트(6205)는 네트워크 스택(6210)의 전송 프로토콜 계층으로부터 기대되는 응답을 수신할 수 있다. 예컨대, 일 실시예에서, 가속화 프로그램(6120)은 클라이언트(6205)의 네트워크 스택(6210)에서 서버와 같은 다른 컴퓨팅 디바이스와의 전송 계층 연결을 개설하라는 요청을 인터셉트하고, 요청에 응답하기 위해 가속화 프로그램(6120)에 의해 개설된 하나 이상의 전송 계층 연결들의 풀을 이용할 수 있다. 다른 실시예에서, 가속화 프로그램(6120)은 제2 애플리케이션

(6220b)에 의해 사용되는 개설된 전송 계층 연결을 통해 제1 애플리케이션(6220a)으로부터의 요청을 다중화한다.

<754> 일부 실시예들에서, 가속화 프로그램(6120)은 네트워크 상에서 전송하기 전에 클라이언트(6205)에서 클라이언트(6205)의 통신을 버퍼링 및 홀딩하기 위한 메커니즘을 포함한다. 예컨대, 네트워크로부터의, 예컨대 서버로부터의 수신된 통신의 클라이언트(6205)에 의한 소비율은, 네트워크 상의 클라이언트(6205)에 의해 전송되는 통신의 생산율보다 낮을 수 있다. 이로써, 클라이언트(6205)는 클라이언트(6205)가 이러한 요청으로부터의 응답을 소비 및 처리할 수 있는 비율보다 높은 비율로 보다 많은 요청들을 서버(30)에 전송할 수 있다. 가속화 프로그램(6120)은 통신을 인터셉트하고 클라이언트(6205)의 소비율 및/또는 생산율이 사용자, 클라이언트(6205) 또는 다른 컴퓨팅 디바이스에 의해 구성되는 임계치와 같은 선정된 임계치 미만인지를 판단한다. 판단된 비율이 원하는 임계치 미만이면, 가속화 프로그램(6120)은 클라이언트(6205)의 성능이 소비율 및/또는 생산율을 선정된 또는 원하는 임계치와 동일하거나 큰 비율까지 증가시킬 때까지 인터셉트한 통신을 클라이언트의 메모리 소자에 저장한다. 이 시점에서, 가속화 프로그램(6120)은 클라이언트의 통신을 네트워크 상에서 전달한다. 이로써, 클라이언트측 메커니즘은 클라이언트(6205)에 의한 통신의 소비 및/또는 생산 성능에 기초하여 클라이언트(6205)의 통신을 조절하기 위해 제공된다.

<755> 도 40a에 도시된 애플리케이션(6220a-6220n)은 클라이언트(6205) 상에서 실행되거나 네트워크(6204)를 통해 전달될 수 있는 임의 유형 및/또는 형태의 애플리케이션, 예컨대 임의 유형 및/또는 형태의 웹 브라우저, 웹-기반 클라이언트, 클라이언트-서버 애플리케이션, 쉘-클라이언트 컴퓨팅 클라이언트, ActiveX 제어, 또는 자바 애플릿, 또는 기타 다른 유형 및/또는 형태의 실행 가능 명령일 수 있다. 애플리케이션(6220a-6220n)은 임의 유형의 프로토콜을 이용할 수 있고, 예컨대, HTTP 클라이언트, FTP 클라이언트, 오스카(Oscar) 클라이언트, 또는 텔넷 클라이언트일 수 있다. 일부 실시예들에서, 애플리케이션(6220a-6220n)은 원격 디스플레이 또는 프리젠테이션 레벨 프로토콜을 이용한다. 일 실시예에서, 애플리케이션(6220a-6220n)은 미국 플로리다 포트 로더데일 소재의 사이트릭스 시스템즈, 인코포레이티드(Citrix Systems, Inc.)가 개발한 ICA 클라이언트이다. 다른 실시예들에서, 애플리케이션(6220a-6220n)은 미국 워싱턴 레이몬드 소재의 마이크로소프트 코퍼레이션이 개발한 원격 데스크톱(Remote Desktop; RDP) 클라이언트를 포함한다. 다른 실시예들에서, 애플리케이션(6220a-6220n)은 소프트웨어 IP 전화기와 같은 VoIP 통신에 관련된 임의 유형의 소프트웨어를 포함한다. 다른 실시예들에서, 애플리케이션(6220a-6220n)은 비디오 및/또는 오디오를 스트리밍하기 위한 애플리케이션과 같은 실시간 데이터 통신에 관련된 애플리케이션을 포함한다.

<756> 도 40b는 도 27에 도시된 어플라이언스 아키텍처와 유사한 어플라이언스(1250)의 아키텍처를 예시한다. 간단히 살펴보면, 어플라이언스(1250)는 하드웨어 계층(6206), 사용자 공간(6203)과 커널 공간(6202)으로 분리된 소프트웨어 계층을 포함한다. 하드웨어 계층(6206)은 커널 공간(6202) 및 사용자 공간(6203) 내의 프로그램 및 서비스가 실행되는 하드웨어 요소를 제공한다. 하드웨어 계층(6206)은 또한 커널 공간(6202) 및 사용자 공간(6203) 내의 프로그램 및 서비스가 어플라이언스(1250)에 내부적으로 그리고 외부적으로 데이터를 전달할 수 있도록 하는 구조 및 소자를 제공한다. 소프트웨어 계층은 어플라이언스(1250)의 로직, 기능, 및 동작을 제공하기 위해 프로그램, 서비스, 프로세스, 태스크, 쓰레드 및 다른 실행 가능 명령을 포함한다.

<757> 어플라이언스(1250)는 애플리케이션 가속화 판단 메커니즘(6275) 및 클라이언트측 가속화 프로그램(6120)을 포함한다. 애플리케이션 가속화 판단 메커니즘(6275)은 소프트웨어, 하드웨어, 또는 하드웨어와 소프트웨어의 조합을 포함한다. 일부 실시예들에서, 애플리케이션 가속화 판단 메커니즘(6275)은 클라이언트(6205) 상 및/또는 서버(30) 상에서 실행되는 애플리케이션(6220a-6220n)이 가속될 수 있는지 아니면 클라이언트(6205)와 서버(30)간의 액세스 또는 통신이 가속될 수 있는지를 판단하기 위한 로직, 기능, 규칙, 또는 동작을 가진 임의 유형 및 형태의 실행 가능 명령, 예컨대 프로그램, 서비스, 프로세스, 태스크 또는 쓰레드를 포함한다. 일 실시예에서, 데이터베이스는 애플리케이션(6220a-6220n)이 가속될 수 있는지를 판단하기 위해 애플리케이션 가속화 판단 메커니즘(6275)에 의해 이용된다. 예컨대, 데이터베이스는 애플리케이션(6220a-6220n)을, 애플리케이션(6220a-6220n)을 가속시킬 수 있는 하나 이상의 가속화 기술과 연관시킬 수 있고, 또한 클라이언트(6205) 및/또는 서버(30)의 사용자, 유형, 형태, 위치, 처리 능력 및 다른 특성에 기초할 수 있다. 일부 실시예들에서, 애플리케이션 가속화 판단 메커니즘(6275)은 이름, 유형 또는 카테고리에 의한 애플리케이션(6220a-6220n)이 가속화 기술에 의해 가속될 수 있는지를 식별하는 정보를 포함하는 메모리 내의 룩업 테이블, 파일, 데이터 구조 또는 객체를 이용한다. 다른 실시예들에서, 어플라이언스(1250) 및/또는 애플리케이션 가속화 판단 메커니즘(6275)은 애플리케이션(6220a-6220n) 또는 서버(30)에의 액세스가 가속될 수 있는지를 식별, 지정 또는 구성하기 위해 사용자 입력을 수신하기 위하여, 구성 메커니즘, 예컨대 사용자 인터페이스, 그래픽, 명령 라인 등을 포함한다.

- <758> 일부 실시예들에서, 애플리케이션 가속화 판단 메커니즘(6275)은 애플리케이션(6220a-6220n)이 가속될 수 있는지, 다른 실시예들에서는, 어떤 가속화 기술(들) 및 어떤 유형 및 형태의 클라이언트(6205)인식을 식별하는 서버(30)로부터의 정보를 요청한다. 또 다른 실시예에서, 애플리케이션 가속화 판단 메커니즘(6275)은, 클라이언트측 가속화 기술을 이용하여 애플리케이션(6220a-6220n)이 가속되거나 가속될 수 있는 경우에 관한 상대적이고 발견적인 정보의 데이터베이스를 제공하기 위해, 하나 이상의 클라이언트측 가속화 기술이 있는 없는 클라이언트(6205)와 서버(30)간의 애플리케이션(6220a-6220n)의 성능에 관한 히스토리 정보의 데이터베이스를 포함한다. 예컨대, 어플라이언스(1250)는 애플리케이션(6220a-6220n)의 성능에 관련된 네트워크 관련 성능 정보를 클라이언트(6205)로부터 캡처할 수 있다. 이로써, 애플리케이션(6220a-6220n)이 가속될 수 있는지에 대한 판단은, 네트워크(6204)의 동작 및 성능 특성을 변화시키는 것에 적용될 수 있고, 또는 동작 및 성능 특성을 변화시키는 것에 기초할 수 있고, 또는 동작 및 성능 특성을 변화시킴으로써 영향을 받을 수 있다.
- <759> 일 양태에서, 애플리케이션(6220a-6220n)은 가속될 수 없을 수도 있고, 또는 가속될 수는 있으나 가속은 효과적이지 않거나 최소한이다. 일 실시예에서, 애플리케이션(6220a-6220n)의 유형 및 형태는 프로토콜을 이용하지 않을 수도 있으나 방식 가속화 기술과 함께 사용하기에 적합한 방식으로 전달할 수 있다. 다른 실시예에서, 애플리케이션(6220a-6220n)이 통신하는 프로토콜 또는 방식은 가속화 기술의 수행을 가능하게 하나, 클라이언트(6205), 어플라이언스(1250) 또는 서버(30)의 동작 또는 성능 특성에 기초할 수 있으며, 가속화 기술은 효과적이지 않거나 최소의 가속화를 제공한다. 이로써, 애플리케이션 가속화 판단 메커니즘(6275)은, 애플리케이션(6220a-6220n)이 가속될 수 있는지 또는 가속화가 원하는 선정된 성능 향상 임계치를 충족하는지에 기초하여 애플리케이션(6220a-6220n)이 가속되기를 원하지 않는 것으로 판단할 수 있다.
- <760> 다른 양태에서, 어플라이언스(6250)는 클라이언트측 가속화 프로그램(6120)을 어플라이언스(1250)의 기억 장치 또는 메모리 소자, 예컨대 어플라이언스의 하드웨어 계층(6206)에 의해 제공되는 기억 장치 또는 메모리에 저장한다. 일 실시예에서, 어플라이언스(1250)는 클라이언트(6205)에 의해 사용될 또는 사용되고 있는 애플리케이션(6220a-6220n)이 클라이언트(6205) 상에서 실행되는 가속화 프로그램(6120)에 의해 가속될 수 있는 것으로 애플리케이션 가속화 판단 메커니즘(6275)을 통해 동적으로 판단하고, 어플라이언스(1250)의 기억 장치 또는 메모리로부터의 가속화 프로그램(6120)을 클라이언트(6205)에 전송 또는 전달한다. 다른 실시예에서, 어플라이언스(1250)는 클라이언트(6205)와 서버(30)간의 통신이 클라이언트(6205) 상에서 실행되는 가속화 프로그램(6120)에 의해 가속될 수 있는 것으로 판단하고 가속화 프로그램(6120)을 클라이언트(6205)에 전달한다. 일부 실시예들에서, 어플라이언스(1250)는 서버(30)와 같은 다른 컴퓨팅 디바이스(6100)로부터 가속화 프로그램(6120)을 수신, 다운로드 또는 획득한다.
- <761> 일부 실시예들에서, 가속화 프로그램(6120)은 어플라이언스(1250)의 정책 엔진(3236)으로부터 정책 정보를 수신, 다운로드 또는 획득한다. 다른 실시예들에서, 가속화 프로그램(6120)은 정책 엔진을, 어플라이언스(1250)의 정책 엔진(3236)과 독립적으로 또는 그 정책 엔진(3236)과 함께, 실행 및 동작시킨다. 다른 실시예들에서, 패킷 엔진(3240, 또는 그 일부는 클라이언트(6205) 상에서, 예컨대 가속화 프로그램(6120)의 일부로서 동작될 수 있다. 이로써, 가속화 프로그램(6120)은 위에서 설명된 바와 같은 패킷 처리 타이머(3242)에 따라 클라이언트(6205) 상에서 동작될 수 있다. 일 실시예에서, 가속화 프로그램(6120)은 하나의 실행 지점에서 그리고 팩 처리 타이머(3242)에 의해 제공되는 정밀한 시간 간격에 응답하여 통합된 가속화 기술을 수행할 수 있다.
- <762> 일부 실시예들에서, 안정 모니터링 프로그램(3216)은 어플라이언스(1250) 또는 어플라이언스(1250)이 전송된 가속화 프로그램(6120)과 통신하여 클라이언트(6205) 상에서 클라이언트측 가속화 프로그램(6120)의 상태, 에러 또는 히스토리를 체크 및 결정할 수 있다. 일부 실시예들에서, 안정 모니터링 프로그램(3216) 또는 그 일부는 클라이언트(6205) 상에서 실행된다.
- <763> 이제 도 41a를 참조하면, 어플라이언스(1250)에 의해 가속화 프로그램(6120)을 동적으로 제공하고 클라이언트(6205)에 의해 가속화 프로그램(6120)을 동적으로 설치 및 실행하는 방법(6300)의 일 실시예가 도시되어 있다. 간단히 살펴보면, 단계 6310에서, 어플라이언스(1250)는 클라이언트(6205)로부터 요청을 인터셉트하여 서버와 통신 세션을 개설한다. 단계 6315에서, 어플라이언스(1250)는 클라이언트(6205)가 자동으로 설치 및 실행하도록 가속화 프로그램(6120)을 클라이언트(6205)에 전송한다. 단계 6320에서, 가속화 프로그램(6120)의 수신시, 클라이언트(6205)는 가속화 프로그램(6120)의 사일런트 설치를 자동으로 실행 또는 수행한다. 단계 6325에서, 가속화 프로그램(6120)의 설치의 완료시, 클라이언트(6205)는 클라이언트(6205)와 서버(30)간의 통신을 인터셉트하기 위해 네트워크 스택(6210)에서 가속화 프로그램(6120)을 자동으로 실행한다. 단계 6330에서, 가속화 프로그램(6120)은 복수의 가속화 기술을 수행하고 통신을 암호화 및/또는 암호 해독할 수 있다.

- <764> 보다 상세하게는, 단계 6310에서, 어플라이언스(1250)는 서버(30)와의 통신 세션을 개설하기 위해 클라이언트(6205)로부터의 요청을 어떤 적합한 수단 및 메커니즘에 의해 인터셉트하거나 수신할 수 있다. 일 실시예에서, 어플라이언스(1250)의 패킷 엔진(6240)은 클라이언트(6205)로부터 통신을 인터셉트한다. 다른 실시예들에서, 어플라이언스(1250)는 클라이언트(6205), 예컨대, 가속화 프로그램(6120)과 제1 전송 계층 연결을 개설하고, 클라이언트(6205)를 위해 서버(6205)와 제2 전송 계층 연결을 개설한다. 이로써, 어플라이언스(1250)는 서버(30)로 전송되는 클라이언트의 통신을 수신하거나 인터셉트하거나 획득할 수 있다. 일부 실시예들에서, 어플라이언스(1250)는 서버(30)와의 전송 계층 연결을 개설하기 위해 클라이언트(6205)에 대한 요청을 인터셉트한다. 다른 실시예들에서, 어플라이언스(1250)는 HTTP의 애플리케이션 계층 프로토콜과 같은 전송 계층 연결 상의 프로토콜 계층을 통해 통신 세션을 개설하라는 요청을 인터셉트한다. 상기 방법의 이 실시예는 클라이언트(6205)의 네트워크 스택(6210)의 프로토콜 계층에 통신 세션을 개설하라는 요청으로 실시될 수 있다.
- <765> 단계 6315에서, 어플라이언스(1250)는 가속화 프로그램(6120)을 클라이언트(6205)에 전송한다. 어플라이언스(1250)는 클라이언트(6205)에 의해 요청된 통신 세션의 개설 전에, 동안에 또는 후에 어떤 시점에서 가속화 프로그램(6120)을 전송할 수 있다. 일 실시예에서, 어플라이언스(1250)는 클라이언트 요청을 인터셉트한 것에 응답하여 가속화 프로그램(6120)을 클라이언트(6205)에 전송한다. 다른 실시예에서, 어플라이언스(1250)는 요청을 서버(30)에 전송하고 가속화 프로그램(6120)을 클라이언트(6205)에 전송한다. 일부 실시예들에서, 어플라이언스(1250)는 서버(30)와의 통신 세션을 개설하고, 통신 세션의 개설시, 어플라이언스(1250)는 가속화 프로그램(6120)을 전송한다. 또 다른 실시예에서, 어플라이언스(1250)는 클라이언트(6205)의 인증 및/또는 권한 검증을 수행하고, 또는 클라이언트(6205)의 사용자, 인증된 사용자 또는 클라이언트(6205)가 이렇게 권한 검증되면, 어플라이언스(1250)는 가속화 프로그램(6120)을 클라이언트(6205)에 전송한다. 일 실시예에서, 어플라이언스(1250)는 인증 및/또는 권한 검증을 위해 클라이언트의 요청을 서버(30)에 전송하고, 서버(30)가 클라이언트의 요청을 인증 및/또는 권한 검증하면, 어플라이언스(1250)는 가속화 프로그램(6120)을 클라이언트(6205)에 전송한다.
- <766> 일부 실시예들에서, 어플라이언스(1250)는 어플라이언스(1250)의 기억 장치 또는 메모리로부터의 가속화 프로그램(6120)을 전송한다. 다른 실시예들에서, 어플라이언스(1250)는 서버(30)로부터의 가속화 프로그램(6120)을 요청하고 수신된 가속화 프로그램(6120)을 클라이언트(6205)에 전송한다. 다른 실시예에서, 서버(30)는 가속화 프로그램(6120)을 클라이언트(6205)에 전송한다. 일 실시예에서, 어플라이언스(1250)는 클라이언트(6205)가 가속화 프로그램을 획득, 다운로드 또는 수신하도록 URL(Uniform Resource Locator)를 클라이언트(6205)에 전송한다. 일부 실시예들에서, URL은 어플라이언스(1250)의 기억 장치 또는 메모리 내의 가속화 프로그램(6120)의 위치를 식별하는 반면에, 다른 실시예들에서, URL은 다운로드를 위한 가속화 프로그램(6120)을 제공하는 웹 서버와 같은 서버(30) 상의 가속화 프로그램(6120)을 식별한다. 일 실시예에서, 가속화 프로그램(6120)은 클라이언트(6205) 상에 저장되고, 어플라이언스(1250)는 클라이언트(6205)가 클라이언트(6205) 상에 저장된 가속화 프로그램(6120)을 설치 및 이용할 수 있도록 암호화 또는 라이선스 키와 같은 키를 클라이언트(6205)에 전송한다. 일부 실시예들에서, 어플라이언스(1250)는 클라이언트(6205) 상의 가속화 프로그램(6120)을 설치 및 실행하는데 사용될 파일, 구성, 데이터 또는 다른 정보를 클라이언트(6205)에 전송한다.
- <767> 일 실시예에서, 가속화 프로그램(6120)은 클라이언트(6205)에 의해 자동으로 설치 및 실행되도록 설계 및 구성된다. 가속화 프로그램(6120)은 운영체제의 유형 및 형태에 따라 가속화 프로그램(6120)이 클라이언트(6205)의 운영체제에 의해 등록 또는 인식되도록 하기 위해 파일, 엔트리, 구성, 데이터, 또는 명령을 포함할 수 있다. 일 실시예에서, 서버와 같은 다른 컴퓨팅 디바이스 또는 어플라이언스는 가속화 프로그램을 클라이언트(6205)에 전송하고 클라이언트(6205)는 가속화 프로그램(6120)을 자동으로 설치 및 실행한다. 일 실시예에서, 가속화 프로그램(6120)은 구동되는 컴퓨팅 디바이스(6100)에 추가될 플러그-및-플레이(PnP) 디바이스가 되도록 설계 및 구성된다. 일부 실시예들에서, 가속화 프로그램(6120)은 자체-설치 실행 가능하며, 예컨대 설치러 프로그램 및 가속화 프로그램(6120)을 포함하는 실행 가능 프로그램이다. 다른 실시예들에서, 가속화 프로그램(6120)은 복수의 파일, 예컨대 클라이언트(6205)의 운영체제에 가속화 프로그램(6120)을 등록 및 설치하는데 필요한 파일과 같은 설치 패키지 또는 설치 다운로드를 포함할 수 있다. 예컨대, 가속화 프로그램(6120)은 inf 파일 및 .sys 파일을 포함할 수 있다. .inf 파일은, 디바이스에 대한 유효 논리 구성 및 디바이스와 연관된 드라이버 파일의 이름의 리스트와 같이 디바이스를 설치하는데 필요한 정보를 가진 마이크로소프트 윈도우즈 패밀리 내의 윈도우즈 셋업을 제공한다. 일부 실시예들에서, .inf 파일은 autorun .inf 파일을 포함할 수 있고, 이 파일은 운영체제에게 어느 실행가능 파일이 개시될 지, 및 실행 가능 파일의 개시에 관련된 구성 정보를 알리는 구성 파일이다. 일 실시예에서, .sys 파일은 가속화 프로그램(6120) 또는 그 일부를 포함하는 드라이버 파일이다.

- <768> 단계 6320에서, 클라이언트(6205)는 가속화 프로그램(6120)을 자동으로 설치한다. 가속화 프로그램(6120)은 클라이언트(6205)의 운영체제에 따라 어떤 적합한 방식으로 설치될 수 있다. 일 실시예에서, 클라이언트(6205)는 가속화 프로그램(6120)의 수신시 가속화 프로그램(6120)을 설치한다. 일부 실시예들에서, 클라이언트(6205)는 가속화 프로그램(6120)의 사일런트 설치를 자동으로 수행 또는 실행한다. 일 실시예에서, 사일런트 설치의 클라이언트(6205)의 사용자 또는 애플리케이션에 투명하게 수행된다. 다른 실시예들에서, 가속화 프로그램(6120)의 사일런트 설치의 클라이언트(6205)의 재부팅 또는 재시작을 요하지 않는다. 다른 실시예에서, 사일런트 설치의 설치를 개시 및/또는 완료하기 위해 사용자의 상호 작용을 요하지 않는다. 다른 실시예들에서, 가속화 프로그램(6120)의 사일런트 설치의 클라이언트(6205)가 네트워크 계층, 세션 계층, 및/또는 네트워크 스택(6210)의 애플리케이션 계층에 투명하게 구동되는 동안에 일어난다. 일부 실시예들에서, 가속화 프로그램(6120)은 클라이언트(6205)에 의해 실행되는 자체-설치 실행 가능하다. 다른 실시예들에서, 클라이언트(6205)는 가속화 프로그램(6120)을 설치하기 위해 플러그 및 플레이 관리자를 이용한다. 일 실시예에서, 클라이언트(6205)는 가속화 프로그램(6120)을 수신 및 설치하는 설치 관리자를 포함한다. 다른 실시예에서, 어플라이언스(1250)에 의해 전송되는 가속화 프로그램(6120)은 또한 가속화 프로그램(6120)을 설치하는 설치 프로그램을 포함한다.
- <769> 다른 실시예에서, 가속화 프로그램(6120)은 사일런트 설치를 통해 자동으로 설치된다. 일 실시예에서, 사일런트 설치의 사용자가 참가하지 않는 설치를 포함한다. 다른 실시예에서, 사일런트 설치의 사용자 설치를 개시 및/또는 완료하기 위해 사용자의 상호 작용을 필요로 하지 않거나 상호 작용이 없는 설치를 포함한다. 일부 실시예들에서, 설치의 설치 프로세스는 설치의 상태 또는 진행에 관한 정보를 디스플레이하지 않는다는 점에서 사일런트하다. 일 실시예에서, 설치의 사용자에게 투명하다는 점에서 사일런트하다. 다른 실시예들에서, 설치의 가속화 프로그램(6120)의 설치의 클라이언트(6205)의 재부팅 또는 재시작을 요하지 않기 때문에 사일런트하다. 다른 실시예에서, 설치의 클라이언트의 동작의 중단 또는 불통없이 클라이언트(6205)의 동작 중에 끊임없이 일어난다는 점에서 사일런트하다. 이로써, 가속화 프로그램(6120)은 재부팅을 요구하지 않고 어떤 정보를 설치 관련 사용자에게 디스플레이하지 않음으로써 클라이언트(6205)의 사용자 또는 애플리케이션에 투명한 방식으로 설치될 수 있다.
- <770> 클라이언트(6205)의 재부팅 또는 재시작을 방지 또는 회피하기 위해, 일부 실시예들에서, 클라이언트(6205), 예컨대 클라이언트(6205)의 운영체제는 운영체제가 구동되는 동안에 플러그 및 플레이 디바이스를 위해 가속화 프로그램(6120)의 일 실시예에서 네트워크 구동기와 같은 구동기를 설치 및 구성하기 위해 플러그 및 플레이 관리자를 갖고 있다. 일 실시예에서, 플러그 및 플레이 관리자는 가속화 프로그램(6120)의 설치 패키지의 구성에 기초하여 클라이언트(6205)를 재부팅 또는 재시작하라는 지시를 받지 않는다. 다른 실시예에서, .inf 파일은 컴퓨터를 재부팅 또는 재시작하라는 명령을 포함하고 있지 않다. 일 실시예에서, 가속화 프로그램(6120)은 일반적으로 이용되는 공유 동적-링크 라이브러리(DLL)를 대체하지 않고 나란한 구성 요소로서 구현될 수 있다. 다른 특정 실시예들에서, 가속화 프로그램(6120)의 네트워크 구동기의 경우, 가속화 프로그램(6120)은 INetCfgPnpReconfigCallback 네트워크 구동기 API를 이용하여, 사용자가 구성 변화가 구동기에서 일어나도록 운영체제를 재부팅하지 않아도 되도록 한다. 부가적으로, 가속화 프로그램(6120)은 구성 정보를 객체를 소유한 네트워크 구성 요소의 구동기에 전송하기 위해 INetCfgComponentControl의 ApplyPnpChanges 방법의 구현 내에서 SendPnpReconfig API를 호출하는 통보 객체(notify object)를 가질 수 있다. SendPnpReconfig API는 데이터를 구동기에 전송하기 위해 통보 객체에게 메커니즘을 제공하고, 일부 실시예들에서, 사용자가 구성 변화가 일어나기 전에 운영체제를 재부팅할 필요가 없도록 하는데 사용된다.
- <771> 단계 6325에서, 가속화 프로그램(6120)이 자동으로, 사일런트하게, 투명하게, 또는 달리 설치 완료될 때, 가속화 프로그램(6120)은 클라이언트(6205) 상에서 자동으로 실행된다. 일부 실시예들에서, 가속화 프로그램(6120)을 설치하는 설치 프로그램은 가속화 프로그램(6120)을 개시 또는 실행시킨다. 일부 실시예들에서, 가속화 프로그램(6120)을 위한 설치러 프로그램은 가속화 프로그램(6120)을 클라이언트(6205)의 메모리에 로딩 또는 실행하기 위해 시스템 호출을 행한다. 일 실시예에서, 가속화 프로그램(6120)의 설치의 가속화 프로그램(6120)을 개시시키라는 명령, 명령 또는 지령을 포함한다. 일 실시예에서, 가속화 프로그램(6120)은 클라이언트(6205)에게 가속화 프로그램(6120)을 자동으로 실행하도록 통보하는 자동 실행 구성, 예컨대 autorun.inf 파일을 포함한다. 다른 실시예들에서, 플러그 및 플레이 관리자 또는 클라이언트(6205)의 운영체제는 설치시 자동으로 가속화 프로그램(6120)을 실행한다. 일 실시예에서, 가속화 프로그램(6120)은 클라이언트(6205)에 의해 개시되는 서비스, 프로세스, 스레드 또는 태스크를 포함한다. 일부 실시예들에서, 가속화 프로그램(6120)은 자동으로 개시하도록 구성되는 운영체제의 서비스이다. 일 실시예에서, 가속화 프로그램(6120)은 클라이언트의 운영체제의 네트워크 스택의 메모리에 로딩되는 네트워크 구동기를 포함한다.

- <772> 다른 실시예에서, 가속화 프로그램(6120)은 클라이언트(6205)의 메모리에 로딩되는 네트워크 구동기를 포함한다. 일부 실시예들에서, 가속화 프로그램(6120)은 네트워크 스택(6210)에 할당된 메모리에 로딩된다. 일부 경우에, 가속화 프로그램(6120)은 가속화 프로그램(6120)이 전송 계층과 같은 네트워크 스택의 프로토콜 계층을 액세스하는 것을 가능하게 하는 메모리 영역 또는 공간에 로딩 및 실행된다. 다른 경우에, 가속화 프로그램은 가속화 프로그램(6120)이 커널-레벨 데이터 구조(6225)를 액세스하는 것을 가능하게 하는 메모리에 로딩 및 실행된다. 다른 실시예들에서, 가속화 프로그램(6120)은 애플리케이션(6220a-6220n)의 메모리에 로딩된다. 다른 실시예에서, 가속화 프로그램(6120)은 자기 자신의 메모리 공간 또는 컨텍스트에서 독립적으로 실행된다. 일 실시예에서, 가속화 프로그램(6120)은 애플리케이션(6220a-6220n)의 메모리 공간 또는 컨텍스트에서 구동된다. 일부 실시예들에서, 가속화 프로그램(6120)은 사용자-모드 메모리 또는 사용자-모드(6203)에 할당된 메모리에 로딩되나, 다른 실시예들에서, 가속화 프로그램(6120)은 커널-모드 메모리 또는 커널-모드(6202)에 할당된 메모리에 로딩된다.
- <773> 일부 실시예들에서, 가속화 프로그램(6120)은 메모리에 로딩되고 및/또는 클라이언트의 사용자, 클라이언트(6205)의 애플리케이션, 어플라이언스(1250) 또는 서버(30)에 투명하게 클라이언트(6205) 상에서 실행된다. 다른 실시예들에서, 가속화 프로그램(6120)은 네트워크 스택(6210)의 전송 계층과 인터페이스하도록 실행되고, 세션 또는 애플리케이션 계층과 같은 전송 계층 위의 프로토콜 계층, 및 네트워크 계층과 같은 전송 계층 아래의 프로토콜 계층에 투명하게 실행된다. 일 실시예에서, 가속화 프로그램(6120)은 클라이언트(6205)의 전송 계층 연결, 또는 전송 계층 자체에 투명하게 실행된다.
- <774> 단계 6330에서, 로딩, 개시 또는 실행되는 가속화 프로그램(6120)은 가속화 프로그램(6120)의 복수의 가속화 기술, 예컨대 1)멀티-프로토콜 압축(6238), 2)전송 제어 프로토콜 폴링(6224), 3)전송 제어 프로토콜 다중화(6226), 4)전송 제어 프로토콜 버퍼링(6228), 및 5)캐시 관리자(6232)를 통한 캐싱에 의해 제공되는 기술을 수행한다. 가속화 프로그램(6120)은 또한 클라이언트(6205)와 서버(30) 사이에서 통신의 암호화 및/또는 암호 해독을 수행할 수 있다. 일 실시예에서, 가속화 프로그램(6120)은 멀티-프로토콜 압축을 수행한다. 다른 실시예에서, 가속화 프로그램(6120)은 전송 제어 프로토콜 폴링을 수행하고, 다른 실시예에서, 가속화 프로그램(6120)은 폴링된 전송 계층 연결을 통한 다중화를 수행한다. 일 실시예에서, 가속화 프로그램(6120)은 전송 제어 프로토콜 버퍼링을 수행한다. 일부 실시예들에서, 가속화 프로그램(6120)은 캐싱을 수행한다. 다른 실시예들에서, 가속화 프로그램(6120)은 캐싱 및 압축을 수행한다. 일 실시예에서, 가속화 프로그램(6120)은 캐싱을 전송 계층 폴링 및 다중화와 함께 수행한다. 다른 실시예에서, 가속화 프로그램(6120)은 멀티-프로토콜 압축을 전송 계층 폴링 및 다중화와 함께 수행한다. 다른 실시예에서, 가속화 프로그램(6120)은 캐싱 및/또는 압축을 TCP 버퍼링과 함께, 다른 실시예에서는, TCP 폴링 및 다중화와 함께 수행한다.
- <775> 이로써, 클라이언트측 가속화 프로그램(6120)은 어플라이언스(1250)에 의해 동적으로 제공되고, 클라이언트(6205)와 서버(30)간의 통신을 위해 하나 이상의 클라이언트측 가속화 기술을 수행하기 위해 사일런트 방식 또는 클라이언트(6205)의 사용자 또는 애플리케이션에 투명하게 클라이언트(6205) 상에서 자동으로 설치 및 실행된다. 가속화 프로그램(6120)은 네트워크 스택의 프로토콜 계층에 투명하게 그리고 클라이언트의 사용자, 클라이언트의 애플리케이션, 어플라이언스, 또는 서버에 투명하게 이들 가속화 기술을 수행할 수 있다.
- <776> 다른 양태에서, 어플라이언스(1250)는 클라이언트(6205)에 의해 액세스 요청된 애플리케이션이 가속될 수 있는지를 판단할 수 있고, 애플리케이션이 가속될 수 있으면 가속화 프로그램(6120)을 클라이언트(6205)에 제공할 수 있다. 이제 도 41b를 참조하면, 방법의 다른 실시예가 도시된다. 방법은 서버 상의 애플리케이션을 액세스하라는 요청 뿐만 아니라 연결 또는 통신 세션을 개설하라는 요청이 있을 때 실시될 수 있다. 방법 6350을 간단히 살펴보면, 단계 6355에서, 어플라이언스(1250)는 서버(30)상의 애플리케이션(6220a-6220n)에 대한 액세스를 요청하는 요청을 클라이언트(6205)로부터 인터셉트한다. 단계 6260에서, 어플라이언스(1250)는 애플리케이션(6220)이 가속될 수 있는지를 판단한다. 단계 6365에서, 애플리케이션(6220)이 가속될 수 있으면, 애플리케이션은 단계 6267에서 요청을 서버에 전송한다. 단계 6365에서, 애플리케이션(6220)이 가속될 수 있으면, 어플라이언스(1250)는 가속화 프로그램(6120)이 클라이언트(6205) 상에 설치되었는지 아니면 클라이언트(6205)에 이전에 전송되었는지를 판단한다. 가속화 프로그램(6120)이 클라이언트(6205)에 아직 제공되지 않은 경우에는, 방법(6350)은 위에서 설명된 방법(6300)의 단계 6315에서 계속하여 가속화 프로그램을 전송, 설치 및 실행한다. 가속화 프로그램(6120)이 설치되어 클라이언트(6205) 상에서 실행되고 있으면, 어플라이언스(1250)는, 단계 6375에서, 애플리케이션(6220)을 가속화하라는 메시지를 클라이언트(6205) 상의 가속화 프로그램(6120)에 전송한다. 방법(6350)의 단계 6330에서, 가속화 프로그램(6120)은 애플리케이션(6220)에 대한 통신을 위해 복수의 가속화 기술을 수행하고, 이러한 통신을 암호화 및/또는 암호 해독할 수 있다.

- <777> 보다 상세하게는, 단계 6355에서, 어플라이언스(1250)는 어떤 적합한 수단 및 메커니즘에 의해 서버(30)에 의해 제공되는 애플리케이션을 액세스하라는 클라이언트(6205)로부터의 요청을 가로챌 수 있다. 일 실시예에서, 어플라이언스(1250)의 패킷 엔진(6240)은 클라이언트(6205)로부터 통신을 인터셉트한다. 다른 실시예들에서, 어플라이언스(1250)는 클라이언트(6205), 예컨대, 가속화 프로그램(6120)과 제1 전송 계층 연결을 개설하고, 클라이언트(6205)를 위해 서버(6205)와 제 2 전송 계층 연결을 개설한다. 이로써, 어플라이언스(1250)는 서버(30)에 전송되는 클라이언트의 통신을 수신하거나, 인터셉트하거나, 획득할 수 있다. 일부 실시예들에서, 어플라이언스(1250)는 서버(30)와 개설된 전송 계층 연결을 통해 애플리케이션(6220)을 액세스하라는 클라이언트(6205)에 대한 요청을 인터셉트한다. 다른 실시예들에서, 어플라이언스(6205)은 HTTP의 애플리케이션 계층 프로토콜과 같은 전송 계층 연결 상의 프로토콜 계층을 통해 통신 세션을 개설하라는 요청을 인터셉트한다. 일 실시예에서, 어플라이언스(6205)은 ICA 또는 RDP와 같은 원격 디스플레이 프로토콜을 통해 서버(30)로부터의 애플리케이션(6220)을 디스플레이 및 제공하라는 클라이언트(205)로부터의 요청을 인터셉트한다.
- <778> 단계 6360에서, 어플라이언스(1250)는 클라이언트(6205)에 의해 요청된 애플리케이션(6220)이 가속될 수 있는지를 판단한다. 일부 실시예들에서, 어플라이언스(1250)는 이름, 유형 또는 카테고리에 의해 애플리케이션을 식별하는, 인터셉트한 클라이언트 요청으로부터 애플리케이션 식별자를 식별, 추출 또는 처리한다. 일 실시예에서, 애플리케이션 가속화 판단 메커니즘(6275)은 애플리케이션(6220)이 가속될 수 있는지를 판단하기 위해 어플라이언스(1250)에 의해 사용된다. 일부 실시예들에서, 애플리케이션 가속화 판단 메커니즘(6275)은 애플리케이션(6220)이 가속될 수 있는지를 판단하기 위해 데이터베이스, 룩업 테이블, 또는 데이터 구조 또는 객체와 같은 메모리 또는 기억 장치 내의 데이터의 다른 구성된 소스에서 질의 또는 룩업을 수행한다. 다른 실시예에서, 어플라이언스(1250)는 애플리케이션(6220)이 가속될 수 있는지를 판단하기 위해 요청과 같은 통신을 서버(30)에 전송한다.
- <779> 다른 실시예들에서, 어플라이언스(1250)는 애플리케이션(6220)이 전에 가속되었는지 그리고 가속화가 애플리케이션(6220)의 성능 및 동작을 향상시키는지를 판단하기 위해 성능 로그 또는 히스토리를 갖고 있다. 이로써, 어플라이언스(1250)는 이러한 가속이 애플리케이션(6220)의 성능 또는 동작에 대한 향상에 대한 선정한 임계치를 충족하면 애플리케이션(6220)이 가속될 수 있다고 판단할 수 있다. 또 다른 실시예에서, 어플라이언스(1250)는 네트워크(6204), 클라이언트(6205) 또는 서버(30)의 현재 동작 및 성능에 기초하여 발견적인(heuristic) 규칙을 제공한다. 일 실시예에서, 애플리케이션(6220)은, 클라이언트(6205)가 일정한 성능 및 동작 특성 또는 능력, 예컨대, 일정한 속도 프로세서 또는 최소량의 메모리를 갖고 있으면 가속될 수 있는 것으로 판단될 수 있다. 일부 실시예들에서, 애플리케이션(6220)은 예컨대 어플라이언스(1250)의 정책 관리자에서 구성된 정책 또는 규칙에 기초하여 가속될 수 있는 것으로 판단될 수 있다. 예컨대, 일정한 유형의 애플리케이션(220)을 액세스하는 일정한 유형의 클라이언트(6205)를 가진 원격 사용자 및/또는 서버(30) 사이에서 전달될 애플리케이션(6220)이 가속될 수 있다. 다른 실시예들에서, 애플리케이션(6220)은 사용자 또는 클라이언트(6205)의 인증 및 권한 검증에 기초하여 가속될 수 있는 것으로 판단될 수 있다. 또 다른 실시예에서, 애플리케이션(6220)은 가속화를 원하지 않는 것으로 판단할 수 있다. 예컨대, 애플리케이션(6220)은 빈번하게 사용되지 않는 유형이다.
- <780> 단계 6365에서, 애플리케이션(6220)이 가속될 수 없는 것으로 판단되면, 또는 가속화 기술을 클라이언트(6205) 상의 애플리케이션(6220)에 적용하지 않기를 원하는 것으로 판단되면, 어플라이언스(1250)는 단계 6368에서 인터셉트한 클라이언트 요청을 서버(30)에 전송하고 가속화 프로그램(6120)을 클라이언트(6205)에 전송 또는 제공하지 않는다. 일 실시예에서, 어플라이언스(1250)는 어플라이언스(6220)의 어플라이언스-기반 가속화를 수행 또는 제공할 수 있다. 다른 실시예들에서, 어플라이언스(1250)는 어플라이언스(1250) 상에서 애플리케이션(6220)의 가속화를 수행하지 않는다. 또 다른 실시예에서, 어플라이언스(1250)는 어플라이언스(1250)이 애플리케이션(6220)이 가속될 수 없거나 가속되기를 원하지 않는다고 판단하면 애플리케이션(6220)을 위해 일부 가속화 기술을 수행하고 다른 가속화 기술은 수행하지 않을 수 있다.
- <781> 단계 6365에서, 애플리케이션(6220)이 가속될 수 없거나 클라이언트(6205) 상의 애플리케이션에 가속화 기술을 적용하기를 원하는 것으로 판단되면, 어플라이언스(1250)는 가속화 프로그램(6120)이 클라이언트(6205)에 제공되었는지를 판단한다. 일 실시예에서, 어플라이언스(1250)는 가속화 프로그램(6120)이 클라이언트(6205) 상에 설치되어 있는지 아니면 클라이언트(6205) 상에서 실행되고 있는지를 판단한다. 일부 실시예들에서, 어플라이언스(1250)는 가속화 프로그램(6120)이 클라이언트(6205) 상에서 구동되고 있는지를 판단하기 위해 통신을 클라이언트(6205) 상의 가속화 프로그램(6120)에 전송한다. 다른 실시예들에서, 어플라이언스(1250)는 가속화 프로그램(6120)이 클라이언트(6205)에 전송되었는지를 판단하기 위해 로그 파일 또는 히스토리 파일을 체크한다.

다른 실시예에서, 어플라이언스(1250)는 가속화 프로그램(6120)이 클라이언트(6205) 상에서 실행되고 있는지를 판단하기 위해 어플라이언스(1250) 또는 클라이언트(6205)의 안정 모니터링 프로그램(6216)으로 체크한다.

<782> 어플라이언스(1250)이 가속화 프로그램(6120)이 클라이언트(6205) 상에 전송, 설치 및/또는 실행되지 않은 것으로 판단하면, 어플라이언스(1250)는 도 41a를 참조하여 설명되는 방법(6300)의 단계에 따라 가속화 프로그램(6120)을 제공할 것이다. 예컨대, 어플라이언스(1250)는 가속화 프로그램(6120)을 클라이언트(6205)에 전송하며, 클라이언트(205)는 수신시 자동으로 설치 및 실행한다. 일 실시예에서, 방법(6300)의 일 실시예의 적합한 단계의 수행시, 어플라이언스(1250)는 단계 6275에서 가속화 기술들 중 하나 이상을 애플리케이션(6220)에 적용하라는 메시지를 가속화 프로그램에 전달할 수 있다. 다른 실시예들에서, 가속화 프로그램(6120)이 이미 설치 및 실행되고 있다면, 단계 6375에서, 어플라이언스(1250)는 가속화 기술 중 하나 이상을 애플리케이션(6220)에 적용하라는 메시지를 가속화 프로그램(6120)에 전달한다.

<783> 일부 실시예들에서, 가속화 프로그램(6120)은 식별된 애플리케이션(6120)에 대한 가속화 프로그램(6120)에 의해 이용 가능한 가속화 기술을 수행한다. 다른 실시예들에서, 어플라이언스(1250)는 애플리케이션(6220)에 대해 어떤 가속화 기술을 수행할지를 가속화 프로그램(6120)에게 알려준다. 일 실시예에서, 가속화 프로그램(6120)은 세션마다 애플리케이션(6120)에 대해 원하는 가속화 기술을 적용할 수 있다. 즉, 어플라이언스(1250)로부터 가속화 프로그램(6120)으로의 메시지는 가속화 프로그램(6120)에게 애플리케이션(6220)의 이 인스턴스 또는 세션에 대해 가속화 기술을 수행하라고 알릴 뿐이다. 다른 실시예들에서, 일단 가속화 프로그램(6120)이 어플라이언스(1250)로부터 식별된 애플리케이션(6220)의 가속화 기술을 적용하라는 메시지를 수신하면, 가속화 프로그램(6120)은 애플리케이션(6220)의 인스턴스 또는 세션 동안 또는 클라이언트(6205)가 재부팅 또는 재시작되거나 어플라이언스(6205)가 재부팅 또는 재시작될 때까지 가속화 기술을 적용한다.

<784> 일 실시예에서, 단계 6375에서 어플라이언스(1250)로부터의 메시지는 애플리케이션에 특정되어 있지 않다. 예컨대, 메시지는 어떤 클라이언트(6205)의 애플리케이션의 가속화 기술 중 하나 이상을 실행하라고 가속화 프로그램(6120)에게 알린다. 일부 실시예들에서, 클라이언트(6205)에 전송된 메시지는 가속화 프로그램(6120)을 애플리케이션(6220) 또는 모든 애플리케이션(6220a-6220n)의 가속화 기술 중 하나 이상을 이용하여 중지를 알린다. 다른 실시예에서, 어플라이언스(1250)는 애플리케이션(6220)을 무시하라는 메시지를 가속화 프로그램(6120)에 전달한다. 또 다른 실시예에서, 어플라이언스(1250)는 가속화 기술에 대한 업데이트 또는 새로운 가속화 기술의 적용과 같은 구성 데이터 또는 정보를 가속화 프로그램(6120)에 제공하라는 메시지를 가속화 프로그램(6120)에 전달한다.

<785> 단계 6330에서, 가속화 프로그램(6120)은 애플리케이션(6220)에 대해 가속화 프로그램(6120)의 복수의 가속화 기술, 예컨대, 1) 멀티-프로토콜 압축(6238), 2)전송 제어 프로토콜 폴링(6224), 3)전송 제어 프로토콜 다중화(6226), 4) 전송 제어 프로토콜 버퍼링(6228), 및 5)캐시 관리자(6232)를 통한 캐싱에 의해 제공되는 기술을 수행한다. 가속화 프로그램(6120)은 또한 클라이언트(6205)와 서버(30)간의 애플리케이션(6220)의 통신의 암호화 및/또는 암호 해독을 수행할 수 있다. 일 실시예에서, 가속화 프로그램(6120)은 애플리케이션 관련 데이터의 멀티-프로토콜 압축을 수행한다. 다른 실시예에서, 가속화 프로그램(6120)은 전송 제어 프로토콜 폴링을 수행하고, 다른 실시예에서, 가속화 프로그램(6120)은 폴링된 전송 계층 연결을 통해 다중화를 수행한다. 일 실시예에서, 가속화 프로그램(6120)은 전송 제어 프로토콜 버퍼링을 수행한다. 일부 실시예들에서, 가속화 프로그램(6120)은 캐싱을 수행한다. 다른 실시예들에서, 가속화 프로그램(6120)은 캐싱 및 압축을 수행한다. 일 실시예에서, 가속화 프로그램(6120)은 전송 계층 폴링과 함께, 다른 실시예에서는 다중화와 함께 캐싱을 수행한다. 다른 실시예에서, 가속화 프로그램(6120)은 TCP 버퍼링과 함께, 다른 실시예에서, 전송 계층 폴링과 함께, 또 다른 실시예에서는, 다중화와 함께 멀티프로토콜 압축을 수행한다. 다른 실시예에서, 가속화 프로그램(6120)은 압축과 함께, 또 다른 실시예에서는, TCP 폴링과 함께, 또 다른 실시예에서는 다중화와 함께 캐싱을 수행한다.

<786> 이로써, 어플라이언스(1250)는 애플리케이션을 가속시킬지 아니면 애플리케이션이 가속될 수 있는지를 동적으로 판단하여, 애플리케이션(6220)에 대한 가속화 기술들 중 하나 이상을 클라이언트(6205)에 대해 수행하도록 클라이언트측 가속화 프로그램(6120)에 전달한다. 또한, 일부 실시예들에서, 복수의 가속화 프로그램(6120)은 어플라이언스에 의해 클라이언트(6205)에 동적으로 전달되고 클라이언트(6205)에 의해 자동으로 설치 및 실행될 수 있다. 예컨대, 가속화 프로그램은 서버(6205)에의 각각의 연결, 또는 애플리케이션(6220)과의 각각의 통신 세션을 위한 기술 및 방법에 따라 제공될 수 있다. 이로써, 클라이언트(6205)은 각각의 서버(630) 또는 각각의 애플리케이션(6220a-6220n)에 대해 가속화를 처리 및 수행하기 위해 복수의 가속화 프로그램(6120)을 자동으로

설치 및 실행할 수 있다.

<787> 이제 도 41c를 참조하면, 복수의 가속화 기술을 통합된 방식으로 수행하는 방법(6380)의 일 실시예가 도시되어 있다. 간단히 살펴보면, 단계 6280에서, 가속화 프로그램(6120)은, 전송 계층에서, 전송 계층 연결을 통한 클라이언트(6205)와 서버(30)간의 통신의 네트워크 패킷을 인터셉트한다. 단계 6390에서, 가속화 프로그램(6120)은 커널-레벨 데이터 구조, 예컨대, API를 통해 클라이언트(6205)의 네트워크 스택(6210)에 제공되는 데이터 구조를 통해 전송 계층에서 네트워크 패킷을 액세스한다. 단계 6395에서, 가속화 프로그램(6120)은 가속화 프로그램(6120)에서 인터페이스 지점 또는 실행 시점에서 커널-레벨 데이터 구조를 이용하여 통합된 방식으로 복수의 가속화 기술을 수행한다.

<788> 보다 상세하게는, 단계 6385에서, 가속화 프로그램(6120)은 전송 계층 연결을 통한 클라이언트(6205)와 서버(30)간의 통신의 네트워크 패킷을 적합한 수단 및 메커니즘에 의해 인터셉트한다. 일 실시예에서, 가속화 프로그램(6120)은 클라이언트(6205)와 서버(30) 사이에 전송 계층 연결을 개설하기 위해 클라이언트의 요청, 또는 이 요청에 대한 응답의 네트워크 패킷, 또는 관련 네트워크 패킷을 인터셉트한다. 다른 실시예에서, 가속화 프로그램(6120)은 클라이언트(6205)와 서버(30)간의 전송 계층 연결을 통해 애플리케이션(6220)을 액세스 또는 이용하기 위해 요청, 또는 이 요청에 대한 응답의 네트워크 패킷 또는 관련 네트워크 패킷을 인터셉트한다. 일 실시예에서, 가속화 프로그램(6120)은 전송 구동기 인터페이스 또는 네트워크 스택(6210)의 전송 프로토콜 계층에서 인터페이스된 네트워크 구동기를 통해 전송 프로토콜 계층에서 네트워크 패킷을 인터셉트한다. 다른 실시예에서, 가속화 프로그램(6120)은 NDI(Network Driver Interface Specification) 구동기, 또는 미니-포트 구동기, 또는 미니-필터구동 기를 통해 전송 프로토콜 계층, 또는 네트워크 스택(6210)의 다른 프로토콜 계층에서 네트워크 패킷을 인터셉트한다. 일부 실시예들에서, 가속화 프로그램(120)은 후킹 또는 필터링 메커니즘을 통해 전송 계층에서 네트워크 패킷을 인터셉트한다.

<789> 단계 6390에서, 가속화 프로그램(6120)은 커널-레벨 데이터 구조(6225)를 통해 전송 계층에서 인터셉트한 네트워크 패킷의 정보 및 데이터를 액세스 또는 획득한다. 커널-레벨 데이터 구조(6225)를 이용함으로써, 가속화 프로그램(6120)은 전송 계층에서 네트워크 패킷에 의해 운반 또는 전송되는 페이로드(들) 또는 하나 이상의 프로토콜상의 정보 및 데이터를 획득할 수 있다. 일부 실시예들에서, 전송 계층에서 및/또는 전송 계층 상에서 네트워크 스택의 계층에서 네트워크 패킷을 나타내기 위해 커널-레벨 데이터 구조를 이용하면, 가속화 프로그램(6120)은 전송 계층에서 그리고 전송 계층 네트워크 패킷에 의해 운반되는 프로토콜 계층에 대해 복수의 가속화 기술을 수행 또는 동작시킬 수 있다. 일 실시예에서, 단일 커널-레벨 데이터 구조(6225)를 이용하면, 컨텍스트 교환과 함께 복사 및 메모리 할당이 네트워크 스택(6210)의 각종 프로토콜 계층에서 다수의 데이터 구조를 이용하는 것이 방지된다. 일 실시예에서, 가속화 프로그램(6120)은 커널-레벨 데이터 구조(6225)를, 다른 커널-레벨 데이터 구조 또는 사용자-레벨 데이터 구조를 포함할 수 있는 제2 데이터 구조로 복사한다.

<790> 단계 6395에서, 가속화 프로그램(6120)은 프로그램(6210) 내 또는 한 세트의 실행 가능 명령 내의 단일 인터페이스 지점 또는 위치에서 또는 프로그램(6210)의 한 실행 지점에서 복수의 가속화 기술을 수행, 실행 또는 동작시킬 수 있다. 가속화 프로그램(6120)은 복수의 가속화 프로그램(6120)의 가속화 기술, 예컨대, 1) 멀티-프로토콜 압축(6238), 2)전송 제어 프로토콜 폴링(6224), 3)전송 제어 프로토콜 다중화(6226), 4)전송 제어 프로토콜 버퍼링(6228), 5)캐시 관리자(6232)를 통한 캐싱에 의해 제공되는 기술을 수행한다. 가속화 프로그램(6120)은 또한 가속화 프로그램(6120)의 가속화 기술의 실행시 동일 시점에서 클라이언트(6205)와 서버(30)간의 애플리케이션(6220)의 통신의 암호화 및/또는 암호 해독을 수행할 수 있다.

<791> 일 실시예에서, 가속화 프로그램(6120)은 한 세트의 실행 가능 명령에서, 예컨대 기능 호출 또는 한 장소 또는 위치에서, 서로 후속하는 원하는 복수의 가속화 기술을 수행한다. 예컨대, 가속화 프로그램(6120)은 커널-레벨 데이터 구조를 통해 인터셉트한 네트워크 패킷을 획득한 다음에, 서로 후속하는 가속화 기술의 로직, 기능, 규칙 또는 동작을 나타내는 명령을 실행한다. 이로써, 네트워크 패킷의 정보 및 데이터는 커널-레벨 데이터 구조(6225)를 통해 한번 추출 또는 획득되고 가속화 기술을 나타내는 가속화 프로그램(6120)의 명령에 대한 입력, 파라미터, 아규먼트(arguments) 및 조건으로 사용된다. 네트워크 패킷이 보다 높은 레벨의 프로토콜 데이터 및 정보를 운반하더라도, 가속화 프로그램(6120)은 일부 실시예들에서 네트워크 패킷 및 보다 높은 레벨 프로토콜 데이터 및 정보를 하나의 지점에서 그리고 한번의 실행 동안에 처리한다. 부가적으로, 가속화 프로그램(6120)은 캐시 관리자(6232)에 저장된 압축 데이터, 또는 캐시로부터 검색된 압축/압축 해제 데이터에 대해 통합된 방식으로 원하는 순서로 복수의 가속화 기술의 각각을 수행할 수 있다.

<792> 일 실시예에서, 가속화 프로그램(6120)은 서로 후속하여 멀티-프로토콜 압축 및 캐싱을 수행한다. 다른 실시예

에서, 가속화 프로그램(6120)은 서로 후속하여 폴링된 전송 계층 연결을 통해 동작 관련 전송 제어 프로토콜 폴링 및 다중화를 수행한다. 일 실시예에서, 가속화 프로그램(6120)은 압축 및 캐싱에 후속하여, 또는 TCP 폴링 및/또는 다중화에 후속하여 전송 제어 프로토콜 버퍼링을 수행한다. 일부 실시예들에서, 가속화 프로그램(6120)은 캐싱을 수행한다. 일 실시예에서, 가속화 프로그램(6120)은 전송 계층 폴링 및 다중화에 후속하여 캐싱을 수행한다. 다른 실시예에서, 가속화 프로그램(6120)은 전송 계층 폴링 및 다중화에 후속하여 멀티-프로토콜 압축을 수행한다. 다른 실시예에서, 가속화 프로그램(6120)은 TCP 버퍼링에 후속하여, 다른 실시예에서는 TCP 폴링 및 다중화에 후속하여 캐싱 및/또는 압축을 수행한다.

<793> 가속화 프로그램은 일반적으로 가속화 기술을 후속 수행하는 것으로 설명되지만, 후속 실행은 또한 가속화와 관련이 없으나 각각의 가속화 기술 사이에 통합 및 실행되는 다른 로직, 기능, 및 동작을 포함할 수도 있다. 가속화 프로그램은 가속화 기술을 위한 실행 가능 명령으로서의 이러한 통합으로 여전히 동작 및 성능 효율성을 획득하고, 기타 다른 동작 또는 기능은 가속화 프로그램 내의 단일 인터페이스 지점에서 또는 실행 지점에서 실행된다. 또한, 전송 프로토콜 계층에 의해 운반되거나 전송 프로토콜 계층 위에 있는 프로토콜 계층의 가속화 기술이 한번에 및/또는 전송 계층의 한 위치에서 처리된다. 이로써, 보다 높은 레벨의 이들 프로토콜에 대한 가속화 기술은, 네트워크 패킷이 전송되어 네트워크 스택(6210)의 보다 높은 이들 레벨에서 또는 네트워크 스택(6210)의 나중 지점에서 처리됨에 따라, 다시 적용될 필요가 없다.

<794> 다른 양태에서는, 제1 프로그램(6222) 및 가속화 프로그램(6120)(또는 이 실시예에서 제2 프로그램이라도 함)을 이용할 수 있다. 일 실시예에서, 제1 프로그램(6222)은 제2 프로그램(6120)과 함께, 예컨대 어플라이언스(1250)를 통해, 서버(30)와의 가상 사설망 연결을 용이하게 하고 개설하는데 사용될 수 있으며, 어플라이언스(1250)를 통해 클라이언트측 가속화 기술이 적용될 수 있다. 다른 실시예에서, 제1 프로그램(6222)은 제2 프로그램, 즉 가속화 프로그램(6120)을 설치 및 실행하는데 사용된다.

<795> 이제 도 42a를 참조하면, 이 양태를 실시하기 위한 방법(6400)의 일 실시예가 도시된다. 간단히 살펴보면, 단계 6402에서, 클라이언트(6205)가 로그인하여 어플라이언스(6205)와 통신 세션을 개설하고, 단계 6404에서, 어플라이언스(1250)는 제1 프로그램(6222)을 클라이언트(6205)에 전송한다. 단계 6406에서, 클라이언트(6205)는 제1 프로그램(6222)을 설치 및 실행하고, 제1 프로그램(6222)은 가속화 프로그램(6120), 즉, 제2 프로그램을 설치 및 실행한다. 단계 6407에서, 클라이언트(6205)는 개설된 암호화된 데이터 통신 세션을 통해 사설망 상의 자원과 통신하고 자원을 액세스한다. 단계 6410에서, 클라이언트(6205)는 어플라이언스(1250)로부터 로그아웃하고 제품(1250)과의 통신 세션을 종료한다.

<796> 방법(6400)의 단계 6402에서, 클라이언트(6205)는 절차에서 로그를 수행하고, 네트워크(6204)를 통해 어플라이언스(1250)와 암호화된 데이터 통신 세션을 개설한다. 일 실시예에서, 암호화된 데이터 통신 세션은 클라이언트(6205)로부터의 트래픽을 사설 데이터 통신망에서 어플라이언스(1250) 뒤에 존재하는 서버(30)에 연결하기 위해, 터널로서 사용된다. 일 실시예에서, 클라이언트(6205)는 SSL(Secure Sockets Layer) 또는 다른 암호화 방법, 예컨대 IPSec, 및 전송 계층 보안(TLS)을 이용하여 로그인하고 어플라이언스(1250)와 데이터 통신 세션을 개설하기 위해 마이크로소프트 인터넷 익스플로러[®] 또는 넷스케이프 내비게이터[®]와 같은 웹 브라우저를 이용한다. 다른 실시예에서, 암호화된 데이터 통신 세션을 개시하기 위해 HTTP(Hypertext Transfer Protocol over Secure Sockets Layer)와 같은 프로토콜이 사용될 수 있다.

<797> 단계 6404에서, 로그인 및 암호화된 데이터 통신 세션의 개설에 응답하여, 어플라이언스(1250)는 네트워크(6204)를 통해 제1 프로그램을 클라이언트(6205)에 전송한다. 제1 프로그램은 암호화된 데이터 통신 세션을 통한 통신을 위해 터널 엔드포인트의 역할을 하기 위해 설계 및 구성되거나, 또는 형성된다. 일 실시예에서, 제1 프로그램은 클라이언트(6204)의 브라우저에 의해 자동으로 설치 및 실행되는 플러그-인 애플리케이션을 포함한다. 예컨대, 제1 프로그램은 마이크로소프트 인터넷 익스플로러[®] 웹 브라우저에 의해 실행될 플러그-인으로서 제공되는 ActiveX 제어를 포함할 수 있다. 다른 실시예에서, 제1 프로그램은 네트워크 환경을 통해 일을 하는 넷스케이프 내비게이터[®] 웹 브라우저 또는 다른 제어 또는 프로그래밍 구성 요소에 의해 실행될 플러그-인으로서 제공되는 자바 애플릿을 포함할 수 있다.

<798> 단계 406에서, 클라이언트(6205)는 제1 프로그램(6222)을 설치 및 실행하고, 제1 프로그램의 실행은 클라이언트(6205) 상에 제2 프로그램의 설치를 포함한다. 일 실시예에서, 제1 프로그램(6222)은 예컨대 방법(6300) 및 도 41a를 참조하여 설명된 기술을 이용하여 자동으로 설치 및 실행될 수 있다. 일부 실시예들에서, 제1 프로그램(6222)은 제2 프로그램, 즉 가속화 프로그램(6120)을 어플라이언스(1250)으로부터 획득, 다운로드 또는 수

신한다. 다른 실시예에서, 제1 프로그램(6222)은, 예컨대 사일런트 설치 또는 클라이언트(6205)의 사용자, 클라이언트(6205)의 애플리케이션(6220), 어플라이언스(1250) 또는 서버(30)에 투명한 설치에 의해 제2 프로그램을 자동으로 설치 및 실행하기 위해 가속화 프로그램(6120)과 같은 제2 프로그램을 위한 설치러 또는 설치 관리자를 포함한다.

<799> 일 실시예에서, 제2 프로그램은, 부분적으로 네트워크(6204) 상의 자원을 위한 클라이언트(6205) 상에서 구동되는 애플리케이션(6220)으로부터의 통신을 인터셉트하고, 암호화된 데이터 통신 세션을 통해 어플라이언스(1250)에 제공하기 위해 인터셉트한 통신을 제1 프로그램(6222)에 제공하도록 구성된다. 제2 프로그램은 또한 인터넷 네트워크명 해결 서비스 및 선택적으로 분리된 네트워크 트래픽을 제공하도록 구성된다. 트래픽을 분리함으로써, 일 실시예는 어느 트래픽이 제1 프로그램(6222)의 SSL 터널 또는 암호화 터널로 전송되는지 그리고 어느 트래픽이 클라이언트(6205)의 통상적인, 또는 틀에 박힌, 또는 전형적인 동작 하에서 네트워크 스택(6210)의 전송 계층에 의한 처리를 위해 지속될 수 있는지를 판단할 수 있다. 일 실시예에서, 제2 프로그램은 클라이언트(6205)의 운영체제에 "후크(hook)"로서 삽입되는 동적 인터셉터(interceptor)(예컨대, 필터 디바이스 구동기)를 포함한다. 예컨대, 제2 프로그램은 마이크로소프트 윈도우즈 운영체제의 전송 계층 스택과 같은 클라이언트 운영체제의 전송 계층 스택에 부착되는 필터 디바이스 구동기를 포함할 수 있다.

<800> 단계 6408에서, 일단 제1 및 제2 프로그램이 설치되면, 클라이언트(6205) 상에서 구동되는 애플리케이션은, 개설되고 암호화된 데이터 통신 세션을 통해 사실 데이터 통신망(6204)상의 애플리케이션 및 데이터와 같은 자원과 통신하고 액세스할 수 있다. 이 통신이 일어나는 방식은 도 42b에 대해 보다 상세히 후술된다. 하나의 실시예에서, 위에서 설명된 바와 같은 제1 프로그램 및 제2 프로그램의 기능은 클라이언트(6205)에 의해 자동으로 설치 및 실행되는 가속화 프로그램(6120)과 같은 단일 제어 또는 프로그래밍 구성 요소에 의해 수행됨에 주의한다. 가상 사설망 연결 및 통신을 제공하는 외에, 제1 프로그램(6222) 및/또는 가속화 프로그램(6120)과 같은 제2 프로그램은 가상 사설망 연결, 예컨대, 어플라이언스(1250)에의 암호화된 터널 또는 브리지를 통해 클라이언트의 통신에 대해 여기서 설명된 가속화 기술을 수행할 수 있다.

<801> 단계 6410에서, 클라이언트(6205)는 네트워크(6204)로부터 분리되기 위해 로그 아웃 절차를 수행하며, 이는 어플라이언스(1250)와의 암호화된 데이터 통신 세션을 종료시킨다. 일 실시예에서, 로그 아웃시, 제1 프로그램(6222)은 운영체제를 제1 프로그램(6222) 및/또는 제2 프로그램의 설치 전의 상태로 복귀시키기 위해 클라이언트(6205)의 운영체제에 대해 행해진 수정을 자동으로 제거한다. 일 실시예에서, 제1 프로그램(6222) 및/또는 제2 프로그램은 또한 클라이언트(6205)의 계속된 동작을 방해하지 않고 제1 및 제2 프로그램을 클라이언트(6205)의 운영체제로부터 또는 클라이언트(6205) 상에서의 추가 동작으로부터 제거하기 위해 언설치러(uninstaller) 또는 설치취소 명령을 포함한다. 또 다른 실시예에서, 제1 프로그램(6222) 및/또는 가속화 프로그램(6120)은 제공되는 통신 연결 또는 세션 동안에 클라이언트(6205)의 애플리케이션에 의해 사용되는 임시 파일 또는 쿠키와 같은 파일을 제거한다.

<802> 도 42b는 클라이언트(6205)가 사실 데이터 통신망(6204) 상의 자원과 통신하고 액세스하는 다른 방법(6450)의 일 실시예를 도시한다. 예컨대, 방법(6450)은 방법(6400)의 단계 6408이 실행될 수 있는 방법을 나타낸다. 간단히 살펴보면, 단계 6452에서, 클라이언트(6205)는 새로운 연결을 개설하거나, TCP/IP 도메인명 해결과 같은 제1 프로그램 및/또는 제2 프로그램을 통해 도메인명을 해결한다. 단계 6454에서, 제2 프로그램이 실행된다. 단계 6456에서, 제2 프로그램은 사설망으로 향하는 클라이언트(6205)로부터의 통신을 인터셉트하고 통신을 제1 프로그램(6222)에 재라우팅하거나 전송한다. 단계 6458에서, 제1 프로그램(6222)은 연결을 종료 또는 프록싱하고, 페이로드를 분리하고, 개설된 암호화된 통신 세션을 통한 제공을 위해 페이로드를 캡슐화한다. 단계 6460에서, 제1 프로그램(6222)은 인터셉트한 통신을 공중망을 통해 사전 개설된 암호화된 통신 세션을 통해 사설망 내의 어플라이언스(1250)에 전송한다. 단계 6462에서, 어플라이언스(1250)는 제1 프로그램으로부터 수신된 통신을 암호 해독하고, 암호 해독된 통신을 적절한 목적지 자원, 예컨대 서버(30)에 전송한다. 단계 6464에서, 목적지 자원은 암호 해독된 통신을 처리하고, 단계 6464에서 목적지 자원은 응답 통신을, 만약 있다면, 어플라이언스(1250)에 전송한다. 단계 6468에서, 어플라이언스(1250)는 응답 통신을 암호화하고 암호화된 통신을 공중망 상에서 사전 개설된 암호화된 통신 세션을 통해 클라이언트(6205)의 제1 프로그램(6222)에 전송한다. 단계 6470에서, 제1 프로그램(6222)은 응답 통신을 암호 해독하고 암호 해독된 통신을 제2 프로그램을 통해 적절한 클라이언트 애플리케이션에 전송한다.

<803> 단계 6452에서, 클라이언트(6205)의 애플리케이션(6220)은 새로운 연결을 행하거나 클라이언트(6205)의 네트워크 스택(6210)의 전송 프로토콜 계층을 통해 도메인명을 해결한다. 일 실시예에서, 애플리케이션(6220)은 개설 클라이언트(6205)와 서버(30) 사이에 또는 클라이언트(6205)와 어플라이언스(1250) 사이에 전송 계층 연결을 개

설할 것을 요청할 수 있다. 다른 실시예에서, 애플리케이션(220) 또는 클라이언트(6205)는 서버(30)에 의해 제공되는 애플리케이션(6220)을 액세스할 것을 요청할 수 있다. 또 다른 실시예에서, 애플리케이션(220) 또는 클라이언트(6205)는 서버(30)에 의해서 제공되는 애플리케이션(6220)에 대한 액세스를 요청할 수도 있다. 예컨대, 서버(30)는 서버(30) 상에서 실행되는 애플리케이션(6220)의 출력을 나타내는 ICA 또는 RDP의 원격 디스플레이 프로토콜을 전송함으로써 서버-기반 컴퓨팅 또는 쉘-클라이언트 컴퓨팅(thin-client computing)을 제공할 수 있다. 다른 실시예에서, 클라이언트(6205)는 서버(30)의 자원, 예컨대 파일 또는 디렉토리, 또는 이메일 서비스에 대한 액세스를 요청할 수 있다. 일부 실시예들에서, 클라이언트(6205)는 공중망(40) 상에 있을 수 있고 서버(30)는 사설망(40') 상에 있을 수 있다. 다른 실시예들에서, 클라이언트(6205) 및 서버(30)는 서로 다른 사설망들 상에 있을 수 있다.

<804> 단계 6454에서, 제2 프로그램은 자동으로 또는 어떤 전송 계층 기능이 개시되기 전에 하나 이상의 기능을 실행한다. 일부 실시예들에서, 제2 프로그램은 가속화 프로그램(6120)이고 또는 가속화 프로그램(6120)을 포함한다. 일 실시예에서, 제2 프로그램은 단계 6452의 클라이언트 요청을 인터셉트하거나 수신한다. 일부 실시예들에서, 클라이언트(6205)의 애플리케이션(6220)은 제2 프로그램이 인터셉트한 네트워크 스택(6210)에 대한 API 호를 발생시킨다. API 호가 네트워크 스택(6210)의 전송 계층에 의해 처리되기 전에, 제2 프로그램은 통신이 전송 계층 연결을 통한 전송을 위해 전송 또는 처리되기 전에 로직, 규칙, 기능 또는 동작을 수행토록 네트워크 스택(6210)에 후킹되거나 인터페이스된다.

<805> 단계 6456에서, 제2 프로그램은, 예컨대 네트워크(40') 상의 자원에 대한 클라이언트(6205) 상의 애플리케이션(6220a-6220n)에 의해, 클라이언트(205)로부터의 통신을 인터셉트하여, 일 실시예에서 네트워크 환경을 통해 일을 하는 ActiveX 제어 플러그-인, 자바 애플릿 또는 다른 제어 또는 프로그래밍 구성 요소를 포함하는 제1 프로그램(6222)에 재라우팅한다. 제2 프로그램은 통신이 네트워크, 예컨대 어플라이언스(1250) 뒤의 사설망(40')을 위한 것인지를 판단하기 위해 인터셉트한 통신을 제공하는 네트워크 패킷 또는 패킷들로부터 목적지 정보를 액세스, 판독 또는 획득한다. 예컨대, 제2 프로그램은 네트워크 패킷으로부터 목적지 IP 주소 및/또는 포트를 추출 또는 해석할 수 있다. 판단시, 인터셉트한 통신은 네트워크(40')에 전용되며, 제2 프로그램은 어떤 적합한 인터페이스 수단 및 메커니즘을 통해, 예컨대 인터-프로세스 통신 인터페이스 또는 API 호를 통해 인터셉트한 통신을 제1 프로그램(6222)에 전달한다. 일 실시예에서, 인터셉트한 통신은 제1 프로그램(6222)에 그대로 전송되고, 또는 다른 실시예들에서, 인터셉트한 통신은 제1 프로그램(6222)에 전송되기 전에 제2 프로그램에 의해 전처리된다. 예컨대, 제2 프로그램은 인터셉트한 통신으로부터 페이로드를 제거하고 페이로드를 제1 프로그램(6222)에 전송할 수 있다.

<806> 단계 6458에서, 각각의 인터셉트한 통신은 제1 프로그램(6222)에 의해 종료 또는 프록싱되고, 제1 프로그램(6222)은, 개설되고 암호화된 데이터 통신 세션을 통한 전송을 위해 인터셉트한 통신을 준비한다. 일 실시예에서, 제1 프로그램(6222)은 페이로드를 분리하고, 개설된 암호화된 데이터 통신 세션을 통한 전송을 위해 페이로드를 캡슐화한다. 다른 실시예에서, 제1 프로그램(6222)은 제2 프로그램으로부터 수신되는 대로 인터셉트한 통신을 캡슐화한다. 일부 실시예들에서, 페이로드는 TCP 페이로드이고, 예컨대 어플라이언스(1250)를 통해 클라이언트(6205)와 서버(30)간의 새로운 TCP 연결에 캡슐화된다.

<807> 단계 6460에서, 제1 프로그램(6222)은 인터셉트한 통신을 네트워크(6204) 상에서 사전 개설된 암호화된 데이터 통신 세션을 통해 어플라이언스(1250)에 전송한다. 일부 실시예들에서, 제1 프로그램(6222)은 인터셉트한 통신을 암호화하고 암호화된 인터셉트한 통신을 어플라이언스(1250)에 전송한다. 일 실시예에서, 암호화는 SSL 프로토콜에 따라 실행된다. 다른 실시예에서, 암호화는 TLS 기초한다. 임의 유형 및 형태의 암호화 및/또는 암호 해독은 제1 프로그램(6222) 또는 가속화 프로그램(6120)에 의해 이용될 수 있다.

<808> 단계 6462에서, 어플라이언스(1250)는 제1 프로그램(6222)에 의해 전송되는 연결을 종료하는 프록시의 역할을 한다. 어플라이언스(1250)는 제1 프로그램(6222)으로부터 수신되는 통신을 암호 해독하고 암호 해독된 통신을, 어플라이언스(1250)이 네트워크(40') 상의 목적지 자원과 개설한 제2 연결을 통해 네트워크(40') 상의 적절한 목적지 자원에 전송한다. 일 실시예에서, 암호 해독은 SSL 프로토콜 또는 다른 응용 가능 암호화 및 암호 해독 프로토콜에 따라 실행된다. 일부 실시예들에서, 어플라이언스(1250)는 목적지 자원에 전송된 통신에 대해 하나 이상의 가속화 기술, 1)멀티-프로토콜 압축(6238'), 2)전송 제어 프로토콜 폴링(6224'), 3)전송 제어 프로토콜 다중화(6226'), 4)전송 제어 프로토콜 버퍼링(6228'), 및 5)캐시 관리자(6232')을 통한 캐싱에 의해 제공되는 기술들 중 하나 이상을 수행한다.

<809> 단계 6464에서, 목적지 자원은 암호 해독된 통신을 처리한다. 일 실시예에서, 암호 해독된 통신은 연결 또는

통신 세션을 개설하라는 요청이다. 다른 실시예에서, 암호 해독된 통신은 클라이언트(6205)를 위해 애플리케이션(6220)을 개시 또는 액세스하라는 요청이다. 다른 실시예들에서, 암호 해독된 통신은 웹 서버(30)로부터 웹 페이지를 수신하라는 HTTP 요청과 같은 웹 페이지 요청이다.

<810> 단계 6466에서, 암호 해독된 통신이 응답이 존재하는 요청을 포함하면, 목적지 자원은 응답 통신을 어플라이언스(1250)에 전송한다. 일부 실시예들에서, 응답은 클라이언트(6205)에 의해 요청되는 대로 연결 또는 통신 세션을 개설하는 확인 응답을 포함한다. 다른 실시예들에서, 응답은 에러 메시지를 포함한다. 일 실시예에서, 응답은 인증 요청 또는 도전-응답 메커니즘을 포함한다. 일부 실시예들에서, 응답은 클라이언트(6205)에 의해 사용될 가속화 프로그램(6120)을 포함한다. 다른 실시예에서, 응답은 클라이언트(6205)에 의해 디스플레이될 웹 페이지와 같은 HTML를 포함한다. 다른 실시예들에서, 응답은 객체, 예컨대 동적으로 발생된 객체를 포함한다.

<811> 단계 6468에서, 어플라이언스(1250)는 응답 통신을 네트워크(40) 상에서 사전 개설된 암호화된 데이터 통신 세션을 통해 클라이언트(6205) 상의 제1 프로그램(6220)에 전송한다. 일 실시예에서, 어플라이언스(1250)는 응답 통신을 암호화하고 암호화된 응답 통신을 제1 프로그램(6222)에 전송한다. 일부 실시예들에서, 암호화는 SSL 프로토콜 또는 다른 적용 가능 암호화 및 암호 해독 프로토콜에 따라 실행된다. 또한, 어플라이언스(1250)는 클라이언트(6205)로의 통신에 대해 가속화 기술, 예컨대 멀티-프로토콜 압축(6238'), 캐싱(6232') 또는 TCP 버퍼링(6228')을 수행할 수 있다.

<812> 단계 6470에서, 제1 프로그램(6222)은 응답 통신을 암호 해독하고 제2 프로그램을 통해 통신을 적절한 애플리케이션(6222)에 전송한다. 제1 프로그램(6222)은, 예컨대 임의 유형 및 형태의 인터-프로세스 통신 메커니즘 또는 API 호를 통해 제2 프로그램에 전달하기 위해 적합한 인터페이스 수단 및 메커니즘을 이용할 수 있다. 제2 프로그램은 응답 통신을 클라이언트(6205)의 네트워크 스택(6210)을 통해 애플리케이션(6220)에 제공한다. 이로써, 애플리케이션(6220)은 애플리케이션(6220)에 대한 변화 또는 수정 없이 응답 통신을 투명하게 수신한다.

<813> 다른 실시예에 따라, 클라이언트(6205)는, 단계 6458에서 통신을 네트워크(40)를 통해 전송하기 전에, 인터셉트한 통신의 부가적인 처리를 수행한다. 일 실시예는 이러한 데이터를 암호화하기 전에 클라이언트에서 연결을 종료하는 프록시로서 작용하는 VPN 솔루션을 제공하기 때문에, 부가적인 처리는 보다 효과적으로 수행될 수 있다. 이러한 처리는 클라이언트 애플리케이션이 자신이 선택한 IP 주소를 이용할 수 있도록 할 뿐만 아니라, 이들 주소를 런 타임으로 동적으로 변화시키기 위해 인터셉트한 통신의 도메인명 서비스(DNS)명 해결을 포함한다. 이러한 부가적인 처리로, 실시예들은, 분산된 게이트웨이들 또는 서버들 사이에서 보다 높은 이용 가능성 및 보다 높은 효율을 달성하기 위해 전역 서비스 부하 밸런싱과 같은 다른 기술과 효과적으로 통합될 수 있도록 한다. 부가적인 연결 처리는 또한 인터셉트한 통신에 관한 상세한 로그 및 통계의 유지를 가능하게 할 수 있다.

<814> 다른 실시예에서, 어플라이언스(1250)는 클라이언트(6205) 상의 제1 프로그램으로부터 수신된 통신을 종료하고, 단계 6462에 도시된 바와 같이 통신을 네트워크(40') 상에서 목적지에 전송하기보다는 내부에 포함된 하나 이상의 요청을 추가 처리한다. 이 추가 처리는 네트워크(40') 상의 적절한 목적지로의 제공 전에 통신이 어플라이언스(1250)에 의해 재암호화되는 백-엔드 암호화를 포함할 수 있으며, 이에 따라 단대단 네트워크 보안이 제공된다. 그 후, 목적지는 트래픽을 암호 해독하고 적절히 응답할 것이다. 또한, 이러한 처리는 네트워크(40)의 효율성 및 응답성을 향상시키기 위해 목적지 서버에 의한 부가적인 일을 요구하기 보다는 어플라이언스(1250)이 응답을 캐시로부터 서빙할 수 있도록 하고, 로컬 네트워크 부하 밸런싱, 전역 서비스 부하 밸런싱 및/또는 압축을 통신에 대해 수행한다.

<815> 위에서 설명된 방법들에 따라, 암호화된 데이터 통신 세션에 기반을 둔 VPN이 클라이언트(205)와 네트워크(40) 사이에 개설된다. 예컨대, 일 실시예에서, 보안 VPN이 HTTPS를 통해 개설된다. 그 후, 클라이언트(6205)로부터 네트워크(40)로의 모든 통신이 이 암호화된 데이터 통신 세션을 통해 제1 프로그램을 통해 어플라이언스(1250)로 그리고 그 반대로 라우팅된다. 암호화된 데이터 통신 세션이 HTTPS를 이용하여 개설될 수 있지만, 암호화된 데이터 통신 세션을 통해 전달되는 통신은 HTTPS 패킷 데이터 또는 심지어 HTTP 패킷 데이터일 필요 없음에 주의해야 한다. 예컨대, 통신은 또한 TCP/UDP(Transmission Control Protocol/User Datagram Protocol) 또는 ICMP(Internet Control Message Protocol) 패킷 데이터를 포함할 수 있으나, 이들 예는 한정하도록 의도한 것이 아니다. 또한, 도 42b를 참조하여 설명한 방법은 클라이언트(6205) 상의 애플리케이션과 네트워크(40) 상의 자원 간의 요청-응답 유형 통신을 설명하지만, 암호화된 통신은 요청-응답에 기반을 둘 필요는 없다. 오히려, 통신은 임의 유형일 수 있다. 따라서, UDP 세션과 같은 연결 또는 통신 세션을 개설할 수 있는

클라이언트 애플리케이션은 암호화된 통신을 전송 및 수신할 수 있다.

- <816> 다른 양태에서, 가속화 프로그램(6120)은 서버(30)와 연결 또는 통신하기 위해 클라이언트로부터 어떤 매개자 디바이스를 동적으로 바이패스시킬 수 있다. 예컨대, 클라이언트(6205)는 어플라이언스(1250)와 같은 하나 이상의 매개자를 통해 서버와 연결할 수 있다. 한가지 이유 또는 다른 이유로, 매개자는 서버(30)와 통신하기 위한 클라이언트(6205)가 이용할 수 있도록 더 이상 이용 가능하지 않을 수 있으며, 예컨대, 어플라이언스(1250)는 유지를 위해 다운될 수 있고 또는 재부팅 또는 재시작 프로세스 중 일 수 있다. 가속화 프로그램(6120)은 매개자가 이용 가능하지 않고 서버(30)와 다른 연결 또는 통신 세션 경로를 자동으로 개설한다. 이는 연결 및/또는 통신 세션이 변경되거나 또는 불통된 것으로 나타나지 않도록 클라이언트(6205)의 사용자 또는 애플리케이션에게 투명하게 일어날 수 있다.
- <817> 이제 도 43을 참조하면, 매개자를 자동으로 바이패싱하는 방법(6500)의 일 실시예가 도시된다. 간단히 살펴보면, 단계 6505에서, 가속화 프로그램(6120)은 어플라이언스(1250)와 같은 매개자를 통해 클라이언트(6205)와 서버(30) 사이에 전송 계층 연결을 개설한다. 단계 6510에서, 가속화 프로그램(6120)은 개설된 전송 계층 연결을 통해 클라이언트(6205)에 의한 서버(30)로의 전달을 위해 이용할 수 없다고 판단한다. 단계 6515에서, 가속화 프로그램(6120)은 클라이언트(6205) 상에서 클라이언트(6205)에서 서버(30)로의 통신을 인터셉트한다. 단계 6520에서, 가속화 프로그램(6120)은 클라이언트(6205)와 서버(30) 사이에 제2 전송 계층 연결을 개설하고, 결과적으로, 서버(30)로의 클라이언트의 전달에 이용할 수 없는 것으로 판단된 매개자를 바이패스시킨다. 단계 6525에서, 가속화 프로그램(6120)은 제2 전송 계층 연결을 통해 클라이언트(6205)의 인터셉트한 통신을 서버(30)로 전송한다.
- <818> 보다 상세하게는, 단계 6505에서, 가속화 프로그램(120)은 매개자를 통해 클라이언트(6205)와 서버(30) 사이에 전송 계층 연결을 개설한다. 일 실시예에서, 매개자는 어플라이언스(6205)를 포함한다. 다른 실시예들에서, 매개자는, 이들 유형 및 형태의 매개자의 역할을 하거나 이 매개자의 기능 및 동작을 제공하는 캐시, 서버, 게이트웨이, 방화벽, 브리지, 라우터, 교환기, 허브, 프록시, 또는 어떤 소프트웨어 애플리케이션 또는 프로그램 중 하나를 포함한다. 일 실시예에서, 매개자는 서버(30) 상에서 동작할 수 있다. 일부 실시예들에서, 전송 계층 연결은 동일한 유형 및 형태 또는 다른 유형 및 형태의 복수의 매개자를 통해 개설된다. 다른 실시예에서, 전송 계층 연결은 클라이언트(6205)로서 또는 어플라이언스(1250)에 개설된 전송 계층 연결의 폴의 연결을 포함한다.
- <819> 단계 6510에서, 가속화 프로그램(120)은 매개자가 이용 가능하지 않거나 개설된 전송 계층 연결을 통해 클라이언트(6205)가 서버(30)로 전달하는데 이용될 수 없는 것으로 판단한다. 가속화 프로그램(6120)은 적합한 수단 및/또는 메커니즘에 의해 매개자의 상태 또는 이용 가능성을 판단할 수 있다. 일 실시예에서, 가속화 프로그램(6120)은 매개자로의 전송과 관련된 에러 메시지 또는 장애 응답을 수신함으로써 이용 가능하지 않은 것으로 판단한다. 예컨대, 가속화 프로그램(6120)은, 개설된 전송 계층 연결을 통해 클라이언트(6205)로부터의 통신을 전송할 때, 실패한 전송 계층 통신 응답을 수신할 수 있다. 다른 실시예에서, 가속화 프로그램(6120)은 매개자의 상태 및 이용 가능성을 감시하기 위해 핑 명령(ping command)을 선정된 빈도로 매개자에게 전송할 수 있다. 가속화 프로그램(6120)이 매개자로부터 응답을 수신하지 못하거나, 일부 실시예들에서, 지연된 응답 또는 원하는 대기 시간보다 긴 응답을 수신하면, 가속화 프로그램(6120)은 매개자가 이용 가능하지 않거나 클라이언트(6205)에 의해 이용될 수 없는 것으로 판단할 수 있다. 다른 실시예들에서, 서버(30), 어플라이언스(1250) 또는 매개자는, 매개자가 이용 가능하지 않거나 클라이언트(6205)에 의해 이용될 수 없음을 나타내는 정보를 제공하는 메시지를 클라이언트(6205) 또는 가속화 프로그램(6120)에 전송할 수 있다. 일부 실시예들에서, 개설된 전송 계층 연결은 불통 또는 중단되고, 또는 다른 실시예들에서는 폐쇄된다.
- <820> 단계 6515에서, 가속화 프로그램(6120)은 개설된 전송 계층 연결을 통해 매개자를 경유하여 이동하도록 된, 클라이언트(6205)에서 서버(30)로의 통신을 인터셉트한다. 가속화 프로그램(6120)은 임의 지점에서 그리고 네트워크 스택(6210) 내의 어떤 프로토콜 계층에서 통신을 가로챌 수 있다. 일 실시예에서, 가속화 프로그램(6120)은 개설된 전송 계층 연결 상에서의 전송 전에 전송 프로토콜 계층에서 통신을 인터셉트한다. 예컨대, 일부 실시예들에서, 가속화 프로그램(6120)은 전송 구동기 인터페이스를 갖고 있거나 전송 프로토콜 계층에 인터페이스된 네트워크 구동기를 포함한다. 다른 실시예들은 도 42a 및 도 42b를 참조하여 설명한 바와 같이 제1 프로그램(6222) 및 제2 프로그램인 가속화 프로그램(6120)을 포함할 수 있으며, 제1 프로그램(6222) 또는 가속화 프로그램(6120)은 통신을 인터셉트한다.
- <821> 단계 6520에서, 가속화 프로그램(6120)은 단계 6510에서 이용가능하지 않거나 클라이언트가 이용할 수 없는 것

으로 판단되는 매개자를 바이패스하기 위해 클라이언트(6205)를 위해 서버(6205)로의 제2 전송 계층 연결을 개설한다. 일 실시예에서, 가속화 프로그램(6120)은, 예컨대, 클라이언트(6205) 및 서버가 동일한 네트워크(6205) 상에 또는 클라이언트(6205)와 서버(30) 사이에서 라우팅 가능한 다른 네트워크 상에 있을 때, 서버(30)에 직접 제2 전송 계층 연결을 개설한다. 다른 실시예에서, 가속화 프로그램(6120)은 제2 어플라이언스(1250')와 같은 제2 매개자와 제2 전송 계층 연결을 개설한다. 일부 실시예들에서, 가속화 프로그램(6120)은 서버(1250)와 다른 전송 계층 연결을 개설할 것을 어플라이언스(1250)에 요청한다. 일 실시예에서, 어플라이언스(1250)는 서버(30)에의 전송 계층 연결들의 풀 중의 제2 전송 계층 연결을 이용한다. 다른 실시예에서, 가속화 프로그램(6120)은 서버(30)에게 제2 전송 계층 연결을 개설할 것을 요청한다. 일부 실시예들에서, 가속화 프로그램(6120)은 가속화 프로그램(6120)에 의해 서버(30)와 개설된 전송 계층 연결들의 풀 중의 제2 전송 계층 연결을 이용한다.

<822> 일 실시예에서, 가속화 프로그램(120)은 단계 6520에서 사용자 또는 클라이언트(6205)의 애플리케이션(6220)에 투명하게, 또는 일부 실시예들에서는 전송 계층의 위 또는 아래의 프로토콜 계층에 투명하게 제2 전송 계층 연결을 개설한다. 일부 양태에서, 제2 전송 계층 연결은, 단계 6510에서 매개자가 클라이언트(6205)에 의해 이용 가능하지 않거나 사용되어서는 안 된다고 판단될 때, 클라이언트(6205)를 위해 자동으로 개설된다. 다른 실시예들에서, 제2 전송 계층 연결은 인터셉트한 통신의 서버(30)로의 전송, 예컨대, 통신을 전송하는 제1 시도의 실패시 자동으로 개설된다. 일부 실시예들에서, 제2 전송 계층 연결은 하나 이상의 재시도된 통신 전송들의 장애시, 또는 선정된 재시도 횟수의 소진시 자동으로 개설된다. 다른 실시예에서, 제2 전송 계층 연결은 매개자가 네트워크 패킷의 전송을 또는 수신율을 지연시켜 대기를 초래하거나 전송 계층의 이용에 연결 원하지 않는 방식으로 영향을 준다고 있다고 판단될 때 개설된다. 일 실시예에서, 가속화 프로그램(6120)은 부하-밸런싱을 수행하고, 클라이언트(6205) 및/또는 제2 매개자에 대한 매개자의 처리또는 동작을 오프로딩하기 위해 매개자를 바이패싱하는 제2 전송 계층 연결을 개설한다.

<823> 단계 6525에서, 가속화 프로그램(6120)은 클라이언트(6205)의 인터셉트한 통신을 제2 전송 계층 연결을 통해 서버(30)에 전송한다. 일 실시예에서, 가속화 프로그램(6120)은 인터셉트한 통신을 직접 서버(30)에 전송한다. 다른 실시예들에서, 가속화 프로그램(6120)은 인터셉트한 통신을 제2 어플라이언스(1250)와 같은 제2 매개자에 전송한다. 제2 전송 계층 연결을 사용함으로써, 가속화 프로그램(6120)은 매개자를 바이패스시키고 서버(30)와 함께 클라이언트(6205)의 애플리케이션(6220)의 동작을 계속한다. 일 실시예에서, 클라이언트(6205)의 애플리케이션(6220)은 애플리케이션(6220)이 이전의 또는 제1 개설된 전송 계층 연결을 계속 사용하고 있는 것처럼 서버(6220)와 함께 동작 및 통신을 계속한다. 이로써, 가속화 프로그램(6120)은 가속화 프로그램(6120)에 의해 바이패스되지 않는 경우 일어날 수 있는 통신 중단, 불통, 대기, 지연 또는 다른 동작 또는 성능 문제를 방지, 회피 또는 피한다. 다른 양태에서, 이 기술은 중간 디바이스로부터의 액세스에 문제가 있거나 액세스가 불통이 되더라도 클라이언트(6205)에게 서버(30) 또는 원격 액세스되는 애플리케이션에 대한 연속 액세스를 자동으로 제공한다.

<824> 또한, 위에서 설명된 리디렉션 및 바이패싱 기술은 애플리케이션(6220a-6220n) 또는 다른 콘텐츠 및 기능을 클라이언트(6205)에 제공하는 하나 이상의 서버(30)를 액세스하기 위해 클라이언트(6205) 상에서 부하-밸런싱 및 트래픽 관리를 수행하는데 사용될 수 있다. 예컨대, 일 실시예에서, 서버를 액세스하기 위해 클라이언트에 의해 사용되는 매개자 또는 어플라이언스는 전송 계층 연결의 증가 및 응답 속도, 성능 또는 다른 동작의 감소로 과부하가 걸릴 수 있다. 매개자 또는 어플라이언스의 성능이 감소한다고 판단될 때, 가속화 프로그램(6120)은 서버에의 클라이언트의 단대단 연결에서의 성능 병목현상을 바이패스시키기 위해 클라이언트를 다른 매개자 또는 어플라이언스, 또는 서버로 리디렉션할 수 있다.

<825> 다른 양태에서, 클라이언트측 가속화 기술은 클라이언트의 네트워크 스택의 전송 프로토콜 계층에 관련될 수 있거나 그 전송 프로토콜 계층에서 수행될 수 있다. 가속화 프로그램(6120)은 1)전송 제어 프로토콜(TCP) 버퍼링(6228), 2)TCP 연결 풀링(6224), 3)TCP 다중화(6226) 중 하나 이상을 수행하기 위해 실행 가능 명령을 포함할 수 있다. 일부 실시예들에서, 가속화 프로그램(6120)이 클라이언트의 네트워크 스택의 전송 프로토콜 계층에서 인터셉트한 통신을 투명하게 처리함에 따라, 가속화 프로그램(6120)은 클라이언트의 TCP 연결, 및 클라이언트(6205)의 애플리케이션(6220a-6220n)에 의한 연결 상에서의 이용 및 전송을 제어 및 관리할 수 있다. 도 44는 TCP 버퍼링 기술을 실시하는 방법(6600)의 일 실시예를 도시하나, 도 45a 및 도 48b는 TCP 연결 풀링 기술의 일 실시예를 도시하며, 도 46, 도 47, 도 48은 TCP 다중화 기술을 도시한다.

<826> 도 44에 도시된 방법(6600)의 일 실시예의 간단히 살펴보면, 단계 6605에서, 가속화 프로그램(6120)은 클라이언트(205)에 의해 서버(30)를 액세스하라는 요청과 같은 클라이언트(6205)로부터 서버로의 통신을 인터셉트한다.

단계 610에서, 가속화 프로그램(6120)은 수신된 서버 응답의 소비율과 클라이언트에 의해 전송되는 요청의 생산율 간의 차이가 선정된 임계치 미만인지의 여부를 판단한다. 단계 6615에서, 생산율과 소비율 간의 차이가 선정된 임계치 아래로 떨어지지 않으면, 가속화 프로그램(6120)은 단계 6617에서 통신을 서버(260)에게 전송한다. 단계 6615에서, 생산율과 소비율 간의 차이가 선정된 임계치 미만이면, 단계 6620에서, 가속화 프로그램(6120)은 클라이언트(6205)의 메모리에 통신을 저장한다. 단계 6625에서, 가속화 프로그램(6120)은 생산율과 소비율 간의 차이가 상기한 선정된 임계치로 변화하였는지를 판단하며, 변한 경우, 저장된 통신을 서버(30)에 전송한다. 그렇지 않으면, 가속화 프로그램(6120)은 생산율과 소비율의 시점 차가 단계 6625에서 선정된 임계치 이상으로 변화할 때까지 통신을 클라이언트(6205)의 메모리 내에 유지한다. 예컨대, 클라이언트(6205)가 발생된 응답을 소비할 수 있는 비율보다 높은 비율로 요청들을 서버(30)에 전송하고 있으면, 가속화 프로그램(6120)은 또한 생산율과 소비율 간의 차이가 변한 장래 시점까지 전송을 유지한다.

<827> 보다 상세히 설명하면, 단계 6605에서, 가속화 프로그램은 클라이언트(6205)에서 서버(30)로의 통신을 인터셉트한다. 가속화 프로그램(6120)은 네트워크 스택(6210) 내의 어떤 지점 및 어떤 프로토콜 계층에서 통신을 가로챌 수 있다. 일 실시예에서, 가속화 프로그램(6120)은 개설된 전송 계층 연결 상에서의 전송 전에 전송 프로토콜 계층에서 통신을 인터셉트한다. 예컨대, 일부 실시예들에서, 가속화 프로그램(6120)은 전송 구동기 인터페이스를 갖고 있는 또는 전송 프로토콜 계층에 인터페이스된 네트워크 구동기를 포함한다. 다른 실시예들은 도 42a 및 도 42b를 참조하여 설명되는 바와 같이 제1 프로그램(6222), 및 제2 프로그램인 가속화 프로그램(6120)을 포함할 수 있으며, 제1 프로그램(6222) 또는 가속화 프로그램(6120)은 통신을 인터셉트한다. 일 실시예에서, 통신은 애플리케이션(6220)과 같은 서버(30)의 자원을 이용 또는 액세스하라는 클라이언트(6205)의 요청을 수신한다.

<828> 단계 6610에서, 가속화 프로그램(6120)은 클라이언트(6205)의 소비율과 생산율 간의 차이가 선정된 임계치 미만인지를 판단한다. 일 실시예에서, 가속화 프로그램(6120)은 클라이언트(6205)에 의해 서버(30)로 전송된 요청들의 수를 카운트 및 추적하고, 다른 실시예들에서, 가속화 프로그램(6120)은 클라이언트(6205)에 의해 서버(30)로부터 수신된 응답들의 수를 카운트 및 추적한다. 일부 실시예들에서, 클라이언트(6205)는 애플리케이션(6220)마다 전송된 응답들 및 수신되는 요청들을 추적한다. 응답 및 요청은 네트워크 스택(6210)의 프로토콜 계층에서 추적될 수 있다. 일 실시예에서, 클라이언트(6205) 또는 애플리케이션(6220)에 의해 전송되는 요청들의 수는 제출 시점으로부터 전송 계층까지 또는 클라이언트(6205)와 서버(30) 사이의 전송 계층 연결까지 카운트 및 추적된다. 마찬가지로, 다른 실시예에서, 서버(30)로부터 클라이언트(6205) 또는 애플리케이션(6220)에 의해 수신되는 응답들의 수는 전송 계층에서의 수신 시점으로부터 또는 클라이언트(6205)와 서버(30)간의 전송 계층 연결로부터, 및/또는 응답이 네트워크 스택(6210)의 전송 계층 위의 애플리케이션 계층과 같은 프로토콜 계층에 제공되는 시점에 카운트 및 추적된다.

<829> 일부 실시예들에서, 가속화 프로그램(6120)은 클라이언트(6205)와 서버(30) 사이에서 가속화 프로그램(6120)에 의해 개설된 전송 계층 연결의 전송 및 수신 TCP 버퍼에 관한 정보 및 데이터를 액세스, 검사 또는 획득한다. 예컨대, 가속화 프로그램(6120)은 클라이언트(6205)에서 서버(30)로의 네트워크 패킷의 전송율과 수신율 간의 차이를 결정하기 위해 TCP/IP 버퍼의 디폴트 및 최대 사이즈 및 버퍼의 현재 사용된 부분을 결정할 수 있다. 다른 실시예들에서, 가속화 프로그램(6120)은 클라이언트(6205)에서 서버(30)로의 네트워크 패킷의 소비와 생산 간의 차이에 의해 초래되는 폭주가 있는지를 판단하기 위해 임의 유형 및 형태의 폭주 알고리즘을 이용한다. 다른 실시예에서, 가속화 프로그램(6120)은 전송 계층 연결에 의해, 예컨대 네트워크 구동기 또는 TCP 서비스 제공자에 의해 사용되는 폭주 알고리즘과 인터페이스하거나 그 폭주 알고리즘으로부터 정보 또는 데이터를 획득한다. 예컨대, 일 실시예에서, 가속화 프로그램(6120)은 연결에 의해 사용되는 폭주 윈도우에 관한 정보 및 데이터를 결정한다.

<830> 선정된 임계치는 가속화 프로그램(6120)의 적합한 수단 및 메커니즘에 의해 구성, 지정, 정의 또는 식별될 수 있다. 일 실시예에서, 임계치는 클라이언트(6205) 및/또는 애플리케이션(6220)의 생산율과 소비율 사이에 퍼센트, 상대값, 절대값 등으로 지정될 수 있다. 소비율 및/또는 생산율은 일정한 기간에 소비된 수신들 및 생성된 전송들의 수에 의해 각각 어떤 정밀도로 식별될 수 있다. 일부 실시예들에서, 임계치는 클라이언트(6205) 및/또는 애플리케이션(6220)의 생산율과 소비율 간의 양적인 차이, 일부 실시예들에서는 일정한 기간에서의 양적인 차이로서 지정될 수 있다. 예컨대, 임계치는 클라이언트(6205)가 소비한 것보다 많은 요청을 생성(6100)한 시점으로서 지정될 수 있다. 다른 예에서, 임계치는 클라이언트(6205)가 동일한 기간 동안 소비한 요청들보다 많이 기간마다 서버(30)에 대한 요청을 생성(610)하는 시점으로서 지정될 수 있다.

<831> 단계 6615에서, 클라이언트(6205) 및/또는 애플리케이션(6220)의 생산율과 소비율의 차이가 선정된 임계치 미만

이 아니면, 가속화 프로그램(6120)은 단계 6617에서 통신을 서버(6260)에 전달한다. 일부 실시예들에서, 가속화 프로그램은 통신을 위해 가속화 기술을 수행한다. 예컨대, 통신은 풀링되고 다중화된 전송 계층 연결을 통해 서버에 전송될 수 있고, 부가적으로, 압축될 수 있다. 다른 실시예들에서, 클라이언트(6205)는 통신을, 서버(30)에의 클라이언트(6205)의 연결을 제공하는 어플라이언스(1250)에 전달할 수 있다.

<832> 단계 6615에서, 클라이언트(6205) 및/또는 애플리케이션(6220)의 생산율과 소비율의 차이가 선정된 임계치 미만이면, 가속화 프로그램(6120)은 단계 6620에서 통신을 클라이언트(6205)의 메모리에 저장한다. 일부 실시예들에서, 메모리는 클라이언트(6205)의 커널-모드(6202)의 메모리일 수 있고, 다른 실시예들에서, 메모리는 클라이언트(6205)의 사용자-모드(6203)일 수 있다. 일 실시예에서, 가속화 프로그램(6120)은 통신을 캐시 관리자(6232)를 통해 캐시에 저장할 수 있다. 다른 실시예들에서, 가속화 프로그램(6120)은 인터셉트한 통신을 버퍼링, 유지 또는 저장하기 위해 가속화 프로그램(6120)에 의해 액세스 가능한 객체, 데이터 구조 또는 다른 데이터 요소를 이용할 수 있다. 일 실시예에서, 인터셉트한 통신은 메모리에 압축 방식으로 저장될 수 있다. 다른 실시예에서, 가속화 프로그램(6120)은 나중의 시점에서의 전송을 위해 메모리에 저장 또는 유지하기 위하여 인터셉트한 통신을 제1 프로그램(6222)에 전송한다.

<833> 단계 6625에서, 가속화 프로그램(6120)은 저장된 통신을 서버(30)에 전송할 시기를 판단한다. 일 실시예에서, 가속화 프로그램(6120)은 클라이언트(6205)의 생산율과 소비율의 차이가 임계치 이상인지를 판단하기 위해 단계 6610 및 6615를 수행하고 단계 6617에서 저장된 통신을 서버(30)에 전송한다. 일부 실시예들에서, 가속화 프로그램(6120)은 규칙적인 빈도 또는 선정된 빈도로 또는 풀링 또는 이벤트 별로 생산율과 소비율의 차이를 비교하고, 차이가 선정된 임계치 이상으로 증가할 때, 가속화 프로그램(6120)은 통신을 서버(30)에 전송한다. 다른 실시예들에서, 가속화 프로그램(6120)은 인터셉트한 통신을 얼마나 오래 동안 저장해야 하는지를 판단하기 위해 타이머를 설정 또는 구성한다. 타이머 만료시, 가속화 프로그램(6120)은 저장된 통신을 서버(30)에 전송한다. 다른 실시예에서, 가속화 프로그램(6120)은 인터셉트한 통신을 저장한 이래 클라이언트(6205)에 의해 소비된 서버 응답들의 수를 체크한다. 소비된 응답들의 수가 선정된 수보다 크면, 가속화 프로그램(6120)은 메모리 버퍼 또는 기억 장치로부터 인터셉트한 통신을 해제하여 서버(30)로의 전송을 위해 통신을 제공한다.

<834> 단계 6625에서, 가속화 프로그램(6120)은 생산율 또는 소비율이 적합한 방식으로 변화되지 않은 것으로 판단되면, 가속화 프로그램(6120)은 적합한 시점에 도달할 때까지 인터셉트한 통신을 메모리에 유지한다. 일 실시예에서, 가속화 프로그램(6120)은 생산율 및/또는 소비율이 변하지 않더라도 단계 6617에서 통신을 서버에 전달한다. 예컨대, 생산율 및/또는 소비율이 변화하기를 일정 시간 기다린 후 변화하지 않으면, 가속화 프로그램(6120)은 통신을 서버(30)에 전달한다.

<835> TCP 버퍼링 기술이 일반적으로 인터셉트한 통신 또는 요청과 관련하여 설명되지만, 방법(6600)의 실시예들은 서버(30)로의 클라이언트(6205)의 다수의 인터셉트한 통신들에 대해 후속하여, 거의 동시에, 또는 동시에 실시될 수 있다. 부가적으로, 다른 실시예에서, 방법(6600)은 클라이언트로부터 다수의 서버(30)로의 통신에 관하여 클라이언트 상에서 실시될 수도 있다. 예컨대, 방법(6600)의 제1 예는 클라이언트(6205)와 제1 서버(30') 사이에서 실시될 수 있고, 방법(6600)의 제2 예는 클라이언트(6205)와 제2 서버(30'') 사이에서 실시될 수 있다. 또한, 일부 실시예들에서, 방법(6600)은 각각의 애플리케이션의 각각의 생산율 및 소비율을 이용하여 제1 애플리케이션(6200a) 및 제2 애플리케이션(6200b)에 대해 실시될 수 있다. 다른 실시예들에서, 방법(6600)은 제1 애플리케이션(6200a)에 대해서 실시될 수 있으나 제2 애플리케이션(6200n)에 대해서는 실시될 수 없다.

<836> 다른 양태에 따라, 클라이언트측 가속화 프로그램(6120)은, 각각의 서버와의 하나 이상의 연결을 개방하고 클라이언트(6205)의 애플리케이션이 서버(30)에 대해 반복되는 데이터 액세스가 가능하도록 이들 연결을 유지함으로써, 클라이언트의 연결을 반복적으로 개폐함으로써 초래되는 서버(30) 및/또는 어플라이언스(1250)의 처리 부하를 감소시킨다. 이 기술은 여기서는 일반적으로 "연결 풀링(connecting pooling)"이라고 한다. 이제 도 45a를 참조하여 방법(6700)을 간단히 살펴보면, 단계 6702에서, 가속화 프로그램(6120)은 서버를 액세스하라는 애플리케이션의 요청을 인터셉트하고, 단계 6704에서, 요청과 연관된 서버의 신원을 판단한다. 단계 6706에서, 가속화 프로그램(6120)은 가속화 프로그램(6120)이 애플리케이션(6220)이 이용할 수 있도록 비어있는, 서버(30)에 대해 개설된 전송 계층 연결을 갖고 있는지를 판단한다. 애플리케이션(6220)이 이용할 수 있도록 비어있는 서버(30)에 대한 전송 계층 연결이 없으면, 가속화 프로그램(6220)은 단계 6708에서, 클라이언트(6205)가 이용할 수 있도록 서버(30)에 대한 전송 계층 연결을 개설한다. 단계 6706에서, 애플리케이션(6220)이 이용할 수 있도록 이용 가능한 전송 계층 연결이 있으면, 단계 6710에서, 가속화 프로그램(6120)은 이용 가능한 전송 계층 연결을 통한 전송 또는 통신에 대한 애플리케이션의 요청을 변환한다.

- <837> 또한, 단계 6712에서, 가속화 프로그램(6120)은 서버(30)로부터 요청에 대한 응답을 수신하고, 단계 6714에서 응답을 애플리케이션(6220)에 대한 응답으로 변환한다. 단계 6716에서, 가속화 프로그램(6120)은 클라이언트(6205)의 애플리케이션(6220a-6220n)이 이용할 수 있도록 전송 계층 연결을 개방 상태로 유지할 수 있다. 클라이언트(6205) 상에서 전송 계층 서버와의 연결(30)을 개방 상태로 유지함으로써 그리고 애플리케이션과의 연결을 필요한 대로 개폐함으로써, 가속화 프로그램(6120)은 인터넷과 같은 네트워크(40)를 통해 클라이언트(6205)를 서빙하는 것과 관련된 TCP 연결 로딩 문제로부터 서버를 해방한다. 단계 6718에서, 일부 지점에 있는 가속화 프로그램(6120)은 서버(30)를 액세스하기 위해 하나 이상의 클라이언트(6205)의 애플리케이션(6220)이 더 이상 연결을 이용하지 않는 것으로 판단되면 전송 계층 연결을 폐쇄한다.
- <838> 보다 상세하게, 단계 6702에서, 가속화 프로그램(6120)은 서버(30)를 액세스하라는 클라이언트(6205)의 애플리케이션(6220a-6220n)에 의한 요청을 인터셉트한다. 일부 실시예들에서, 전송 계층 연결을 통해 요청을 개설 또는 전송하기 전에 전송 프로토콜 계층에서 요청이 인터셉트하여진다. 다른 실시예들에서, 요청은 전송 계층 또는 전송 계층 연결 위의 프로토콜 계층에서 인터셉트하여진다. 일 실시예에서, 애플리케이션(6220)의 요청은 서버(30)와의 전송 계층 연결을 개방 또는 개설하라는 요청이다. 일부 실시예들에서, 요청에 응답하여, 가속화 프로그램(6120)은 클라이언트(6205)의 애플리케이션(6220a-6220n)이 이용할 수 있도록 전송 계층 연결들의 풀 중의 제1 전송 계층 연결을 개설한다. 다른 실시예에서, 애플리케이션 요청은 클라이언트(6205)의 개설된 전송 계층 연결을 통해 서버를 액세스하라는 요청이다.
- <839> 단계 6704에서, 가속화 프로그램(6120)은 적합한 수단 및 메커니즘에 의한 요청으로부터 서버(30)의 신원을 판단한다. 일부 실시예들에서, 서버(30)의 도메인명 또는 인터넷 프로토콜 주소는 요청의 콘텐츠에 의해 식별 또는 참조되는데, 예컨대 요청의 텍스트 스트림이 서버(30)의 도메인명을 식별할 수 있다. 일 실시예에서, 서버(30)의 신원은 목적지 인터넷 프로토콜 주소 및 포트 번호와 같은 TCP 패킷의 헤더 정보에 의해 판단된다. 다른 실시예에서, 서버(30)는 애플리케이션(6220)과 연관되고, 가속화 프로그램(6120)은 데이터베이스 또는 다른 구성된 정보 기억 장치에서 연관성을 찾거나 질의한다.
- <840> 단계 6706에서, 가속화 프로그램(6120)은 애플리케이션(6220)이 이용할 수 있도록 이용 가능하거나 빈 전송 계층 연결이 존재하는지를 판단한다. 일 실시예에서, 가속화 프로그램(6120)은 아직 서버(30)와의 전송 계층 연결을 개설하지 않은 수도 있으며, 이로써, 애플리케이션(6220)이 이용할 수 있도록 이용 가능한 전송 계층 연결이 존재하지 않는다. 다른 실시예에서, 가속화 프로그램(6120)은 이전에 개설된 서버(30)와의 전송 계층 연결을 가질 수 있으나, 다른 애플리케이션(6220)은 현재 활발히 연결을 이용하는 것으로 판단한다. 보다 상세히 후술되는 바와 같이, 가속화 프로그램(6120)은 개설된 전송 계층 연결이 다른 애플리케이션이 이용할 수 있도록 이용 가능한지 아니면 요청에 대한 응답과 같은 애플리케이션(6220)을 위해 서버(30)로부터 수신되는 메시지의 길이에 기초하여 애플리케이션(6220s-6220n)에 의해 공유될 수 있는지, 및/또는 서버(30)와 애플리케이션(6220) 간의 통신이 현재 유희 상태인지를 판단한다.
- <841> 단계 6708에서, 가속화 프로그램(6120)이 전송 계층 연결이 애플리케이션(6220)이 이용할 수 있도록 이용 가능하지 않은 것으로 판단하면, 가속화 프로그램(6120)은 서버(30)와 전송 계층 연결을 개설한다. 일부 실시예들에서, 단계 6708에서 개설된 전송 계층 연결은 제1 서버(30)와의 전송 계층 연결이고, 다른 실시예들에서, 전송 계층 연결은 서버(30)에의 복수의 전송 계층 연결 중 제2 전송 계층 연결이다. 또 다른 실시예에서, 가속화 프로그램(6120)은 이미 개설된 전송 계층 연결이 이용 가능해지기를 또는 애플리케이션의 요청을 서버(30)에 전달하기 위해 비워지기를 기다린다. 예컨대, 가속화 프로그램(6120)은 제1 애플리케이션(6220a)이 개설된 연결을 통해 서버(30)와의 트랜잭션을 바로 완료할 수 있는 것으로 판단할 수 있다.
- <842> 단계 6710에서, 가속화 프로그램(6120)은 전송 계층 연결을 통해 서버(6106)에 전송될 애플리케이션의 요청을 변환한다. 일부 실시예들에서, 가속화 프로그램(6120)은 연결을 공유하는 모든 클라이언트(6205)의 애플리케이션(6220a-6220n)에 대한 전송 계층 연결 통신을 위해 하나의 포트 번호를 이용한다. 일부 경우에, 가속화 프로그램(6120)은 애플리케이션 별로 애플리케이션 상에서 요청 및 요청에 대한 ACK되지 않은 응답을 추적한다. 이로써, 가속화 프로그램(6120)은 어느 애플리케이션(6220)이 주어진 시점에서 서버(30)에 대한 전송 계층 연결을 통해 네트워크 패킷을 전송 및 수신하고 있는지를 인식한다. 일 실시예에서, 한번에 하나의 애플리케이션(6220)만이 전송 계층 연결 상에서 전송 및 수신하며, 따라서 가속화 프로그램(6120)은 어느 애플리케이션(6220)이 연결을 이용하는지를 이해한다. 일부 실시예들에서, 가속화 프로그램(6120)은 애플리케이션(6220)의 프로세스 id를 요청과 연관시킨다. 다른 실시예들에서, 가속화 프로그램(6120)은 포트 번호를 제공하고 포트 번호를 애플리케이션(6220)과 연관시키고, 애플리케이션의 할당된 포트 번호로 전송될 TCP 네트워크 패킷 내의 포트 번호를 수정한다. 다른 실시예에서, 포트 번호는 애플리케이션(6220)에 의해 제공되고, 가속화 프로그램

(6120)은 따라서 TCP 네트워크 패킷 내의 포트 번호를 변화시키거나 제공한다.

- <843> 단계 6712에서, 가속화 프로그램(6120)은 애플리케이션의 요청에 대한 응답을 서버(30)로부터 수신한다. 일 실시예에서, 서버(30)는 요청에 응답하지 않는다. 다른 실시예에서, 서버(30)는 에러 또는 장애 메시지로 응답한다. 일부 실시예들에서, 서버(30)는 다수의 응답으로 응답한다. 다른 실시예들에서, 서버(30)는 다수의 네트워크 패킷 또는 다수의 TCP 세그먼트를 포함하는 응답으로 응답한다. 다른 실시예에서, 서버(30)는 애플리케이션(6220)과 연관되거나 애플리케이션(6220)에 할당된 소스 포트 번호를 식별하는 하나 이상의 네트워크 패킷으로 응답한다. 일 실시예에서, 서버(30)는 클라이언트(6205)의 다수의 애플리케이션에 사용되는, 전송 계층 연결의 소스 포트 번호를 식별하는 하나 이상의 네트워크 패킷으로 응답한다.
- <844> 단계 6714에서, 가속화 프로그램(6120)은 서버로부터의 응답(30)을 애플리케이션(6220)에 응답하는 방식으로 변환 또는 처리한다. 일 실시예에서, 가속화 프로그램(6120)은 수신된 네트워크 패킷 또는 패킷들의 소스 포트 번호를 애플리케이션(6220)의 포트 번호로 대체한다. 다른 실시예에서, 가속화 프로그램(6120)은 추적 메커니즘을 통해 애플리케이션(6220)이 현재 전송 계층 연결을 이용하고 있는지를 판단하고 응답을 네트워크 스택(6210)을 통해 애플리케이션(6220)에 전송한다. 일 실시예에서는, 응답은 연결의 전송 계층 위의 네트워크 스택(6210)의 프로토콜 계층을 통한 처리를 위해 변경 및 전달되지 않는다. 일부 실시예들에서, 가속화 프로그램(6120)은 응답을 처리하여 애플리케이션(6220)에 전송하기 전에 응답의 TCP 세그먼트와 같은 다수의 부분이 수신되기를 기다린다. 일 실시예에서, 가속화 프로그램(6120)은 응답을 제1 프로그램(6222)에 전달하고 제1 프로그램(6222)은 애플리케이션(6220)과 인터페이스하여 응답을 애플리케이션(6220)에 제공한다.
- <845> 단계 6716에서, 가속화 프로그램(6120)은 클라이언트(6205)에서 서버(30)로의 하나 이상의 전송 계층 연결들의 풀에서 전송 계층 연결을 개방 상태로 유지한다. 일 실시예에서, 가속화 프로그램(6120) 또는 네트워크 스택(6210)의 전송 계층 구동기는 연결이 유향 상태일 때, 예컨대 전송할 데이터가 없는 경우 연결의 타단을 주기적으로 검사하는 킵-어라이브 메커니즘을 포함한다. 킵-어라이브 메커니즘은 연결이 유향 상태일 수 있더라도 연결이 여전히 액티브 상태임을 확인하기 위해 응답을 수신하기 위하여 이 메시지를 전송할 수 있다. 킵-어라이브 메시지 및 대응 응답은 임의 유형 및 형태의 포맷, 명령, 지령 또는 통신을 포함할 수 있다. 이로써, 일부 실시예들에서, 가속화 프로그램(6120)은 전송 계층 구동기를 통해 킵-어라이브 메시지를 전송 계층 연결에 전송하거나 전송되게 한다. 일부 실시예들에서, 가속화 프로그램(6120)은 킵-어라이브 메시지에 대한 빈도를 설정하고, 다른 실시예들에서, 연결을 이용하는 애플리케이션(6220a-6220n)의 거동 또는 활동에 기초하여 킵-어라이브 메시지의 빈도를 변화시킨다.
- <846> 일부 실시예들에서, 가속화 프로그램(6120)은 전송 계층 연결을 통해 수신되는, TCP 연결을 리셋 및/또는 종료시키라는 RST 및/또는 FIN 명령, 즉, TCP/IP 명령을 인터셉트한다. 일 실시예에서, 가속화 프로그램(6120)은 인터셉트한 RST 및/또는 FIN 명령을 무시하거나 아무 동작을 행하지 않고 또는 없애거나 삭제하거나 보내버린다. 다른 실시예에서, 가속화 프로그램(6120)은 연결을 개방 상태로 유지하기 위해 RST 및/또는 FIN 명령을 인터셉트하여 수신하나, 메시지를 연결의 타단에 전송한다. 다른 실시예들에서, 가속화 프로그램(6120)은 RST 및/또는 FIN 명령의 처리로 인한 개설된 전송 계층 연결의 폐쇄에 응답하여 새로운 전송 계층 연결을 개설한다.
- <847> 다른 실시예들에서, 가속화 프로그램(6120)은 클라이언트(6205)가 명령을 전송하지 않는 한 연결을 개방 상태로 유지하도록 또는 연결을 폐쇄하지 않도록 서버(30)에게 지시하기 위해 지시, 명령 또는 지령을 클라이언트(6205)의 인터셉트한 통신에 삽입한다. 예컨대, 일 실시예에서, 가속화 프로그램(6120)은 프로토콜 버전 1.0와 같은 HTTP 프로토콜의 GET 요청의 통신을 인터셉트하고 킵-어라이브 헤더, 예컨대, "연결: 킵-어라이브"를 서버(30)로의 통신에 삽입한다. 다른 실시예들에서, GET 요청 또는 다른 HTTP 명령은 킵-어라이브 헤더를 포함할 수 있다. 이들 실시예에서, 가속화 프로그램(6120)은 통신을 가로챌 수 있고, 킵-어라이브 헤더를 체크한 다음에, 통신을 서버(30)에 전송한다. 일부 실시예들에서, 버전 1.1 이상의 HTTP를 이용하며, 버전 1.1 이상의 HTTP으로, 킵-어라이브 메커니즘은 클라이언트(6205)가 연결을 폐쇄하라고 요청할 때까지 서버(30)가 연결을 개방 상태로 유지할 수 있도록 조용히 있다. 다른 실시예들에서, 가속화 프로그램(6120)은 클라이언트(6205)가 재부팅 또는 재시작될 때까지 전송 계층 연결을 서버(30)에 대해 개방 상태로 유지하고, 네트워크(40)는 이용 가능하지 않게 되거나, 클라이언트(6205)는 네트워크(40)로부터 분리되거나, 서버(30)는 재부팅 또는 재시작된다.
- <848> 단계 6718에서, 가속화 프로그램(6120)은 원하는 시점에 클라이언트(6205)와 서버(30)간의 전송 계층 연결들 중 하나 이상을 폐쇄할 수 있다. 일부 실시예들에서, 가속화 프로그램(6120)은 연결을 이용하여 클라이언트(6205)

상에서 하나 이상의 애플리케이션(6220a-6220n)의 종료시 전송 계층 연결을 폐쇄한다. 다른 실시예들에서, 가속화 프로그램(6120)은 애플리케이션(6220a-6220n)이 연결을 이용하는 타임아웃 기간의 만료시 전송 계층 연결을 폐쇄한다. 예컨대, 가속화 프로그램(6120)은 선정된 기간에 만료하도록 타이머를 구성, 설정 또는 제공할 수 있고, 상기 기간 동안에 연결이 유향 상태이거나 유향 상태로 유지되면, 가속화 프로그램(6120)은 연결을 폐쇄한다. 일부 실시예들에서, 서버(30)는 재부팅, 재시작, 또는 연결불통 또는 중단될 수 있고, 가속화 프로그램(6120)은 연결을 폐쇄한다. 일부 실시예들에서, 가속화 프로그램(6120)은 서버(30)로의 요청들의 전송 완료시 그리고 서버(30)로부터의 모든 응답 데이터 수신시 연결을 폐쇄하라는 RST 및/또는 FIN 명령을 전송하거나 전송되게 한다. 다른 실시예들에서, 전송 계층 연결 또는 전송 계층 연결들의 풀은 클라이언트(6205)의 재시작 또는 재부팅, 네트워크(40) 분리 또는 네트워크(40)의 이용 불가능, 또는 서버(30)의 재시작 또는 재부팅시 폐쇄된다.

<849> 일부 실시예들에서, 가속화 프로그램(6120)이 제1 전송 계층 연결만이 클라이언트(6205)의 하나 이상의 애플리케이션(6220a-6220n)에 의해 서버(30)에 대한 연결을 공유하는데 필요하다고 판단함에 따라 서버(30)에의 제1 전송 계층 연결은 개방 상태로 유지되나, 서버에의 제2 전송 계층 연결은 폐쇄된다. 다른 실시예들에서, 가속화 프로그램(6120)은 서버(30)에 대한 하나의 전송 계층 연결들의 풀을 유지하고 클라이언트(6205) 상의 애플리케이션(6220a-6220n)의 증가된 요청, 통신 또는 전송 계층 연결 이용에 기초하여 주어진 서버(30)에 대한 제2 또는 복수의 연결을 개설한다.

<850> 방법(6700)의 일 실시예가 일반적으로 클라이언트(6205)에서 서버(30)로의 하나 이상의 전송 계층 연결들의 풀에 관하여 설명하나, 가속화 프로그램(6120)은 클라이언트와 복수의 서버(30)의 각각 간의 전송 계층 연결들의 풀을 후속하여, 거의 동시에, 또는 동시에 개설할 수 있다. 이로써, 제1 애플리케이션(6220a) 및 제2 애플리케이션(6220b)은 서버(30a)에의 하나 이상의 전송 계층 연결의 제1 풀을 이용할 수 있고, 제3 애플리케이션(6220c) 및 제4 애플리케이션(6220d)은 서버(30b)에의 하나 이상의 전송 계층 연결들의 제2 풀을 이용할 수 있다. 또한, 방법(6700)의 실시예의 단계들 각각은 다른 예로 그리고 다른 빈도로 수행될 수 있다. 일부 실시예들에서, 가속화 프로그램(6120)의 다수의 예는 각각의 서버(30)에 대한 하나 이상의 전송 계층 연결들의 각각의 풀을 처리하는데 사용될 수 있다.

<851> 이제 도 48b를 참조하면, 클라이언트(6205)의 두 애플리케이션(6220a, 6220b)이 이용할 수 있도록 일 실시예에서는 서버(30) 또는 다른 실시예에서는 어플라이언스(1250)에 대한 전송 계층 연결을 제공하는 가속화 프로그램(6120)의 흐름도가 도시되어 있다. 클라이언트(6205) 상의 가속화 프로그램(6120)은 단계 6752에 의해 지정된 바와 같이 애플리케이션(6220)에 의해 제공되는 네트워크 주소 1을 이용하여 제1 클라이언트(6205)와 서버(30) 또는 어플라이언스(1250) 사이의 전송 계층 연결을 개방한다. 단계 6752는 TCP/IP 프로토콜이 연결을 개방하기 위해 다단 핸드셰이크를 이용하기 때문에 2-방식 단계로서 도시되어 있다.

<852> 일단 전송 계층 연결이 개설되면, 가속화 프로그램(6120)은 단계 6754로 도시된 바와 같이 /sales/forecast.html의 경로명을 지정하는 애플리케이션(6220a)으로부터 GET 요청을 인터셉트한다. 빈 전송 계층 연결이 가속화 프로그램(6120)과 서버(30) 또는 어플라이언스(6205) 사이에서 개방되지 않았기 때문에, 가속화 프로그램(6120)이 전송 계층 연결을 개방한다. 일 실시예에서, 단계 6756로 도시된 바와 같이 가속화 프로그램(6120)은 애플리케이션(6220a)의 요청을 서버(30)를 지정하는 네트워크 주소 2의 제2 네트워크 주소에 맵핑시킨다. 예컨대, 가속화 프로그램(120)은 목적지 IP 주소 및/또는 목적지 포트를 애플리케이션(6220a)이 요청한 서버(30') 또는 요청을 처리 또는 응답할 수 있는 다른 서버(30'')로 수정하기 위해 네트워크 주소 변환을 수행한다. 다른 실시예에서, 가속화 프로그램(6120)은 요청을 애플리케이션(6220s)에 의해 수신 또는 발생되는 대로 서버(30)로 또는 어플라이언스(1250)로 전송한다.

<853> 가속화 프로그램(6120)은 또한 단계 6758로 도시된 바와 같이 GET 요청을 그 서버(30), 또는 어플라이언스(1250)에 전달한다. 일 실시예에서, 어플라이언스(1250)는 요청을 서버(30)에 전송하고, 다른 실시예에서, 어플라이언스(1250)는 요청을 어플라이언스(1250)와 서버(30) 사이의 풀링된 또는 풀링 및 다중화된 전송 계층 연결을 통해 전송한다. 일부 실시예들에서, 서버(30)는 단계 6760으로 도시된 바와 같이 요청된 웹 페이지로 응답한다. 가속화 프로그램(6120)은 단계 6762로 도시된 바와 같이 웹 페이지를 애플리케이션(6220a)으로 전송한다. 일 실시예에서, 전송 계층 가속화 프로그램(6120)과 서버(30) 또는 어플라이언스(1250)간의 연결은 단계 6764로 도시된 바와 같이 폐쇄된다. 다른 실시예들에서, 가속화 프로그램(6120)은 폐쇄 요청을 인터셉트하고 전송 계층 연결을 개방 상태로 두라는 요청을 무시한다. TCP/IP 프로토콜에 따라, 네트워크 연결 폐쇄는 다단 프로세스를 포함할 수 있다. 그러므로, 단계 6764의 흐름선은 양방향으로 도시된다. 다른 실시예들에서 풀링 양태의 기술에 따라, 제1 애플리케이션(6220)을 위해 그리고 제1 애플리케이션(6220)에 의해 개설된 전송 계층

연결은 개방 상태로 유지되거나 또는 동일한 애플리케이션(6220a) 또는 다른 애플리케이션, 예컨대 제2 애플리케이션(6220b)으로부터 다른 데이터 스탬프들을 수용하기 위해 유지된다.

<854> 단계 6766에서, 가속화 프로그램(6120)은 제2 애플리케이션(6220a)으로부터 서버(30) 또는 어플라이언스(1250)로의 요청을 인터셉트한다. 제1 애플리케이션(6220a)에 대한 단계 6756에서 개설된 전송 계층 연결과 같은, 개방 상태인 및/또는 제2 애플리케이션(6220b)이 이용할 수 있도록 빈 전송 계층 연결이 없으면, 가속화 프로그램(6120)은 이 이전에 개설된 전송 계층 연결을 이용한다. 이로써, 제2 전송 계층 연결은 단계 6766에서 개방될 필요가 없다. 그렇지 않으면, 가속화 프로그램(6120)은 서버(30) 또는 어플라이언스(1250)를 위해 제2 전송 계층 연결을 개설한다. 단계 6768에서, 가속화 프로그램은, 예컨대 요청 웹 페이지 /sales/forecast.html를 요청하는 제2 애플리케이션(6220b)으로부터의 요청을 인터셉트하여, 단계 6770에서 요청을 서버(30) 또는 어플라이언스(1250)에 전송한다. 빈 연결이 이미 가속화 프로그램(6120)과 서버(6120) 사이에서 개방되어 있으면, 가속화 프로그램(6120)이 서버(6120)에게 다른 연결을 개방하라는 처리 부하로 부담을 주는 것이 불필요하다. 단계 6772에서, 가속화 프로그램(6120)은, 예컨대 전송 계층 연결로부터 어플라이언스(1250)를 통해 서버(30)로부터의 응답을 인터셉트하거나 수신하고, 응답을 제2 애플리케이션(6220b)에 전송한다. 단계 6776에서, 가속화 프로그램(120)은 제2 애플리케이션(6220b)으로부터의 폐쇄 요청을 인터셉트하고, 일부 실시예들에서, 연결을 폐쇄하는 반면에, 다른 실시예들에서는 요청을 무시하고 제1 애플리케이션(6220a), 제2 애플리케이션(6220b), 또는 클라이언트(6205)의 또 다른 애플리케이션(6220c-6220n)으로부터의 다른 데이터 요청들을 수용하기 위해 연결을 유지한다.

<855> 가속화 프로그램(6120)이 단계 6776에서 서버(30) 또는 애플리케이션(1250)과의 연결을 폐쇄하는 다수의 시나리오가 있다. 예컨대, 클라이언트(6205) 또는 가속화 프로그램(6120)은 클라이언트(6205)가 애플리케이션(6220a, 6220b)에 대해 모든 요청된 데이터를 수신한 것으로 판단될 때, 또는 종료시 FIN(finish) 명령을 개시할 수 있으며, 애플리케이션(6220a, 6220b)은 중단되거나 종료된다. 일부 실시예들에서, 클라이언트(6205) 또는 가속화 프로그램(6120)은 또한 유사한 조건 하에서 RST(reset) 명령을 개시할 수도 있다. 가속화 프로그램(6120)과 서버(30) 또는 어플라이언스(1250) 간의 연결을 폐쇄하는 것 외에, RST 명령은 하우스키핑 동작들이 양호한 순서로 서버측 연결을 유지하기 위해 수행되게 한다. 특히, TCP 프로토콜은 RST 명령이 올바른 SEQ(시퀀스) 번호를 가져 서버가 세그먼트를 받아들이도록 보장한다. 그러나, RST 명령은 올바른 ACK(acknowledge) 번호를 갖도록 보장되지 않는다. 이 시나리오에 대비하여, 가속화 프로그램(6120)은 서버(30) 또는 어플라이언스(1250)에 의해 전송되는 데이터의 바이트의 트랙 및 클라이언트(6205)에 의해 확인 응답된 바이트를 유지한다. 클라이언트(6205)가 서버(30)에 의한 모든 데이터에 대해 아직 확인 응답하지 않았으면, 가속화 프로그램(6120)은 확인 응답되지 않은 바이트를 계산하고 ACK를 서버(6205)에 전송한다.

<856> 또한, 도 48b에 도시되어 있지 않지만, 서버(30) 또는 어플라이언스(1250)는 자신과 클라이언트(6205)간의 연결을 폐쇄할 수 있다. 서버(30) 또는 어플라이언스(1250)는 FIN 명령을 클라이언트(6205)에 전송한다. 이에 응답하여, 일부 실시예들에서, 가속화 프로그램(6120)은 연결을 폐쇄하고 다른 실시예에서는 서버(30) 또는 어플라이언스(1250)와의 다른 연결을 다시 개설한다.

<857> 또한, 도 45a의 방법(6700)의 실시예 및 도 48b의 예시적인 흐름도는 복수의 애플리케이션이 이용할 수 있도록 하나 이상의 전송 계층 연결들을 풀링하는 것으로 설명되지만, 풀링 기술은 이들 연결을 통해 복수의 전송 계층 연결들 및 요청들을 요청 또는 개시시키는 단일 애플리케이션(6220)에 적용될 수 있다. 예컨대, HTTP 프로토콜의 일 실시예에서, 전송 계층 연결은 애플리케이션으로부터의 각각의 HTTP 요청에 대해 개설될 수 있다. 이 기술들을 이용하여, 각각의 요청에 대한 전송 계층 연결들을 개방하고 폐쇄 않고 하나 이상의 전송 계층 연결들의 풀이 애플리케이션(220)에 의해 사용될 수 있다.

<858> 다른 양태에서, 도 45a 및 도 45b를 참조하여 설명한 풀링 기술을 통해 개설된 전송 계층 연결과 같은 동일 또는 공유 전송 계층 연결을 통해 애플리케이션 요청들을 다중화하는 기술들을 이용할 수도 있다. 일부 실시예들에서, 개설된 전송 계층 연결의 이용 가능성이 판단되며, 애플리케이션의 요청에 대한 서버로부터의 응답(30)의 콘텐츠가 완전히 수신되었는지를 체크함으로써 연결을 통한 복수의 애플리케이션으로부터의 요청들이 다중화될 수 있다. 보다 상세하게 후술되는 바와 같이, 모든 응답 데이터들이 수신되었는지를 체크하기 위해 일 실시예에서는 응답의 콘텐츠-길이 파라미터가 사용되고 다른 실시예에서는 응답의 청크된 전송 인코딩 헤더가 사용된다. 일 양태에서, 응답으로부터의 모든 데이터가 수신되었는지의 여부를 체크하여, 도 45에 도시된 방법(6700)의 단계 6706, 6708)와 같이 풀링된 연결이 현재 애플리케이션이 이용할 수 있도록 빈 상태인지, 및/또는 서버에의 연결들의 풀에 대한 다른 전송 계층 연결을 개설할지를 판단한다. 다른 실시예에서, 응답에 대한 콘텐츠 길이를 체크하는 기술이 동일한 전송 계층 연결을 통해 복수의 애플리케이션으로부터의 요청들을 다중화하는

기술로서 사용된다.

- <859> 이제 도 46을 참조하면, 클라이언트(6205)에서 서버(30)로의 단일 전송 계층 연결을 통한 요청을 다중화하기 위한 방법(6800)의 실시예가 도시되어 있다. 간단히 살펴보면, 단계 6805에서, 가속화 프로그램(6120)은 클라이언트(6205)와 서버(30) 사이에 전송 계층 연결을 개설한다. 단계 6810에서, 가속화 프로그램(6120)은 서버(30)로의 제1 애플리케이션(6220a)의 요청을 인터셉트한다. 단계 6815에서, 가속화 프로그램(6120)은 전송 계층 연결이 현재 다른 애플리케이션에 의해 사용되고 있는지 아니면 유향 상태인지를 판단한다. 단계 6817에서, 전송 계층 연결이 애플리케이션(6220a)이 이용할 수 있도록 이용 가능하면, 단계 6820에서, 가속화 프로그램(6120)은 요청을 서버에 전송한다. 그렇지 않으면, 단계 6817에서, 전송 계층 연결이 애플리케이션(6220a)이 이용할 수 있도록 이용 가능하지 않으면, 가속화 프로그램(6120)은 단계 6819에서 일정한 기간 동안 대기하고, 단계 6815로 복귀하거나, 애플리케이션(6220)이 이용할 수 있도록 제2 전송 계층 연결을 개설한다. 단계 6825에서, 가속화 프로그램(6120)은 애플리케이션의 요청에 대한 응답을 서버로부터 수신한다. 단계 6830에서, 가속화 프로그램(6120)은 제2 애플리케이션(6220b)에 의해 제2 요청을 인터셉트해서 단계 6815로 진행하여 전송 계층 연결이 제2 애플리케이션(6220b)이 이용할 수 있도록 이용 가능한지를 판단한다. 일부 실시예들에서, 가속화 프로그램(6120)은 단계 6830에서, 수신 단계 6825에서의 제1 요청의 응답의 수신 전에 또는 모든 응답 데이터를 수신하기 전에 제2 애플리케이션(6220b)의 요청을 인터셉트한다. 여기서 추가로 설명되는 바와 같이, 일부 실시예들에서, 가속화 프로그램(6120)은 전송 계층 연결이 유향 상태이거나 애플리케이션이 요청에 대한 모든 응답 데이터를 수신할 때를 결정하기 위해 콘텐츠 길이 체크 기술을 이용한다.
- <860> 보다 상세하게는, 단계 6805에서, 가속화 프로그램(6120)은 클라이언트(6205)와 서버(30) 사이에 전송 계층 연결을 개설한다. 일부 실시예들에서, 가속화 프로그램(6120)은 어플라이언스(1250), 즉 매개자와 또는 매개자를 통해 전송 계층 연결을 개설한다. 일 실시예에서, 가속화 프로그램(6120)은 전송 계층 연결을 서버(30)에의 전송 계층 연결들의 풀로서 개설한다. 이로써, 일부 실시예들에서, 전송 계층 연결은 서버(30)에의 제2 또는 제3 전송 계층 연결을 포함할 수 있다. 다른 실시예들에서, 가속화 프로그램(6120)은 여기서 이전에 설명한 바와 같이 제1 프로그램(6222)을 통해 전송 계층 연결을 개설할 수 있다. 일부 실시예들에서, 가속화 프로그램(6120)은 클라이언트(6205)의 제1 애플리케이션(6220a)에 의한 요청에 응답하여 전송 계층 연결을 개설한다.
- <861> 단계 6810에서, 가속화 프로그램(6120)은 서버(30)를 액세스하라는 제1 애플리케이션(6220a)의 제1 요청을 인터셉트한다. 일부 실시예들에서, 전송 계층 연결을 통해 요청을 개설 또는 전송하기 전에 요청은 전송 프로토콜 계층에서 인터셉트하여진다. 다른 실시예들에서, 요청은 전송 계층 또는 전송 계층 연결 위의 프로토콜 계층에서 인터셉트하여진다. 일부 실시예들에서, 요청은 제1 프로그램(6222)에 의해 가로채어진다. 일 실시예에서, 애플리케이션(6220a)의 요청은 서버(30)와의 전송 계층 연결을 개방 또는 개설하라는 요청이다. 다른 실시예에서, 애플리케이션 요청은 개설된 전송 계층 연결을 통해 또는 어플라이언스(1250)를 통해 서버를 액세스하라는 요청이다.
- <862> 단계 6815에서, 가속화 프로그램(120)은 전송 계층 연결이 유향 상태인지 아니면 제1 애플리케이션(6220a)이 이용할 수 있도록 이용 가능한지를, 또는 제1 애플리케이션(6220a)의 제1 요청을 전달할 것인지를 판단한다. 일부 실시예들에서, 가속화 프로그램(6120)은 하나 이상의 전송 계층 연결들의 풀로부터, 풀 내의 어느 전송 계층 연결이 제1 애플리케이션(6220a)이 이용할 수 있도록 유향 상태 또는 빈 상태인지를 판단한다. 일 실시예에서, 가속화 프로그램(6120)은 전송 계층 연결이 유향 상태인 것으로 판단하며, 이는 가속화 프로그램(6120)이 요청에 응답하여 또는 요청 직전에 전송 계층 연결을 개설하였기 때문이다. 일부 실시예들에서, 가속화 프로그램(6120)은 애플리케이션(6220)으로부터 요청을 수신하지 않았을 수도 있으며, 이 요청을 가속화 프로그램(6120)이 인터셉트하여 처리될 제1 요청으로 인식한다. 다른 실시예에서, 가속화 프로그램(6120)은 전송 계층 연결 상에서 전송되는 어떤 요청에 대한 ACK되지 않은 응답의 수를 추적하여, ACK되지 않은 응답이 없으면, 가속화 프로그램(6120)은 전송 계층 연결이 제1 애플리케이션(6220a)이 이용할 수 있도록 이용 가능한 것으로 인식한다. 또 다른 실시예에서, 가속화 프로그램(6120)은 전송 계층 연결이 현재 유향 상태임을 인식한다. 예컨대, 가속화 프로그램(6120)은 연결을 개방 상태로 유지하기 위해 서버에 대해 킵 얼라이브 요청을 개시하고 있을 수 있다. 일부 실시예들에서, 전송 계층 연결은, 마지막 트랜잭션이 완료되었으나 서버(30) 및/또는 클라이언트(6205)는 RST 및/또는 FIN 명령을 아직 전송하지 않았기 때문에, 유향 상태이다.
- <863> 일부 실시예들에서, 가속화 프로그램(6120)은 제1 애플리케이션(6202a)의 제1 요청에 대한 서버로부터의 응답(30)이 완료되었는지 아니면 가속화 프로그램(6120)이 모든 응답 데이터를 수신하였는지를 판단하기 위해 응답의 콘텐츠 길이를 체크할 수 있다. 위에서 언급된 바와 같이, 이들 기술은 일부 실시예들에서 또한 폴링 기술을 위한 다른 연결을 개설하도록 판단하는데 이용된다. 이 기술에 관하여, 도 47 및 도 48은, 일 실시예에서

응답의 콘텐츠-길이 파라미터를, 또는 다른 실시예에서는 모든 응답 데이터가 수신되었는지를 판단하기 위해 응답의 체크된 전송 엔코딩 헤더를 체크하는 것을 설명하는데, 사용될 것이다. 도 47은 TCP 세그먼트(6900)라고도 하는 TCP 패킷의 TCP 부부분을 도시한다. TCP 세그먼트(6900)는 TCP 헤더(6902) 및 바디(6904)를 포함한다. 바디(6904)는 일 실시예에서, 다른 데이터 및 정보 중에서도, HTTP 헤더 및 메시지를 포함하고, TCP 패킷은 HTTP의 애플리케이션 계층 프로토콜을 운반한다. 일부 실시예들에서, 콘텐츠 길이 파라미터(6906)는 HTTP 헤더에 의해 또는 HTTP 헤더에 위치되거나 발견되거나 참조된다. 일 실시예에서, 가속화 프로그램(120)은 모든 응답 데이터가 수신되는지를 판단하는데 콘텐츠 길이 파라미터(690)를 이용한다.

<864> 도 48은 TCP 패킷의 TCP 세그먼트의 다른 실시예를 도시한다. 전송 계층 연결 상에서 HTTP 프로토콜을 이용하는 일부 실시예들에서는, 체크된 전송 엔코딩 헤더가 존재할 수 있고, 이는 체크된 전송 엔코딩이 TCP 세그먼트 또는 패킷에 적용되었음을 나타낸다. 이로써, 이 실시예에서, 메시지의 길이는 체크된 엔코딩에 의해 정의된다. 체크된 엔코딩은 전송 메시지를 일련의 체크로서 전송하기 위해 메시지의 바디를 수정하며, 각각의 체크는 체크-사이즈 필드 내에 자기 자신의 길이 지시자를 갖고 있다. TCP 세그먼트(7600)는 TCP 헤더(도시되지 않음) 및 바디를 포함한다. 바디는, 다른 정보 중에서도, HTTP 헤더(7602A-7602C) 및 메시지를 포함한다. HTTP 헤더(7602A-7602C)는 7개의 체크-사이즈 필드(7606A-7606C), 및 6개의 체크 메시지 데이터(7604A-7604F)를 포함한다.

<865> 체크-사이즈 필드(7606A-7606G)는 도 48에 예시된 바와 같이 함께 링크되거나 참조되거나 연관된다. 체크-사이즈 필드(7606A)는 체크 메시지 데이터(7604A) 내의 메시지의 길이를 나타내고, 체크-사이즈 필드(7606C)는 체크 메시지 데이터(7604C) 내의 메시지의 길이를 나타낸다. 마지막 체크-사이즈 필드(7606G)는 체크 또는 이어지는 메시지가 더 이상 없음을 나타내는 길이값 0을 포함한다. 다른 실시예에서, 가속화 프로그램(6120)은 체크-사이즈 필드를 통해 클라이언트(6205)가 모든 응답 데이터를 수신하였는지를 결정한다.

<866> 도 47 및 도 48은 일반적으로 요청에 대한 모든 응답 데이터가 수신되었는지를 체크하는 기술을 설명하지만, 이들 기술은 비동기 메시지 또는 통신을 클라이언트(6205)에 전송하는 서버(30) 또는 어플라이언스(1250)에 적용 가능하다. 또한, 이들 기술은 일반적으로 HTTP 프로토콜에 대해 도 47 및 도 48을 참조하여 설명되지만, 이들 기술은 클라이언트(6205)에 의해 전송된 또는 수신될 데이터의 길이의 지시를 제공하는 어떤 프로토콜 계층에서 어떤 프로토콜을 위해 이용될 수 있다. 이로써, 일부 실시예들에서, 모든 데이터가 아직 클라이언트와 서버 또는 어플라이언스 사이에서 요청, 응답 또는 통신과 연관되어 수신되는지를 판단하기 위해 가속화 프로그램(6120)은 어떤 프로토콜 계층에서 네트워크 패킷의 어느 부분을 액세스, 추출, 검사, 분석 또는 처리한다. 또 다른 실시예에서, 가속화 프로그램(6120)은 바이트가 애플리케이션(6220)에 대해 클라이언트(6205)와 서버(30) 사이에서 확인 응답되지 않았는지를 결정하기 위해 클라이언트(6205)와 서버(30) 사이에서 전송, 수신 및 확인 응답된 바이트의 수를 추적한다.

<867> 위에서 설명된 콘텐츠 길이 기술을 이용하여, 가속화 프로그램(6120)은 이전에 사용된 또는 클라이언트(6205)의 다른 애플리케이션(6220a-6220n)이 사용하는 서버(30)로의 동일 전송 계층 연결을 재이용할 수 있다. 단계 6817에서, 가속화 프로그램(6120)은 전송 계층 연결이 제1 요청을 전송하기 위해 이용가능한지를 판단하고, 이용 가능하면 단계 6820에서 요청을 서버(30)에 전송한다. 이용 가능하지 않으면, 단계 6819에서, 가속화 프로그램(6120)은 모든 데이터가 애플리케이션의 ACK되지 않은 요청을 위해 수신될 때까지 대기할 수 있다. 예컨대, 가속화 프로그램(6120)은 타이머를, 예컨대, 짧은 기간으로 설정하고, 단계 6815로 진행할 수 있다. 일부 실시예들에서, 가속화 프로그램(6120)은 모든 데이터가 클라이언트(6205)의 네트워크 스택(6210)의 패킷 처리 타이머에 응답하여 수신되었는지를 체크한다. 다른 실시예에서, 단계 6819에서, 가속화 프로그램(6120)은 제1 애플리케이션(6220a)의 제1 요청을 전송하기 위해 다른 전송 계층 연결을 개설한다.

<868> 단계 6820에서, 가속화 프로그램(6120)은 어느 애플리케이션(6220)이 현재 연결 상에서 ACK되지 않은 요청 또는 응답을 갖고 있는지, 아니면 현재 연결을 이용하고 있는지를 추적할 수 있다. 예컨대, 한 번에는 하나의 애플리케이션(6220)만이 요청을 전송하고 연결 상에서 응답을 수신할 수 있다. 이로써, 가속화 프로그램(6120)은 어느 애플리케이션(6220)이 연결을 이용하고 있는지를 이해하게 된다. 일부 실시예들에서, 가속화 프로그램(6120)은 연결을 공유하는 클라이언트(6205)의 모든 애플리케이션(6220a-6220n)에 대한 전송 계층 연결 통신을 위해 하나의 포트 번호를 이용한다. 일부 경우에, 가속화 프로그램(6120)은 애플리케이션 별로 애플리케이션 상에서 요청 및 요청에 대한 ACK되지 않은 응답을 추적한다. 일부 실시예들에서, 가속화 프로그램(6120)은 애플리케이션(6220)의 프로세스 id를 요청과 연관시킨다. 또 다른 실시예에서, 가속화 프로그램(6120)은 제1 애플리케이션(6220a)의 요청을 동일한 네트워크 패킷 또는 패킷들, TCP 세그먼트 또는 세그먼트들에서 제2 애플리케이션(6220b)의 요청과 함께 전송한다. 다른 실시예들에서, 가속화 프로그램(6120)은 하나 이상의 TCP 세그먼트

트 윈도우의 일련의 TCP 세그먼트의 일부로서 동일한 전송 계층 연결을 통해 애플리케이션(6220a-6220n)의 복수의 요청을 전송한다.

<869> 다른 실시예들에서, 가속화 프로그램(6120)은 수신된 어느 응답 또는 메시지가 어느 애플리케이션(6220a-6220n)을 위한 것인지를 추적 및 인식하기 위해 포트 넘버링 메커니즘 및/또는 기법을 이용한다. 다른 실시예들에서, 가속화 프로그램(6120)은 포트 번호를 제공하고 애플리케이션(6220)과 연관시키며, 전송될 TCP 네트워크 패킷 내의 포트 번호를 애플리케이션의 할당된 포트 번호로 수정한다. 다른 실시예에서, 포트 번호는 애플리케이션(6220)에 의해 제공되고, 가속화 프로그램(6120)은 TCP 네트워크 패킷 내에서 포트 번호를 변화시키거나 제공한다. 이로써, 일부 실시예들에서, 가속화 프로그램(6120)은 클라이언트(6205)의 복수의 애플리케이션(6220a-6220n)으로부터의 요청들을 조화시켜 애플리케이션(6220a-6220n)이 전송 계층 연결을 동시에 이용할 수 있도록 한다.

<870> 단계 6825에서, 가속화 프로그램(6120)은 예컨대 어플라이언스(6205)를 통해 제1 애플리케이션(6220a)의 요청에 대한 응답을 서버(30)로부터 수신하고, 제1 애플리케이션(6220a)에 응답을 제공한다. 일부 실시예들에서, 가속화 프로그램(6120)은 네트워크 스택(6210)을 통해 제1 애플리케이션(6220a)에 응답을 제공하며, 예컨대 연결의 전송 계층 위의 프로토콜 계층에 의한 응답의 처리가 가능해지거나 개시된다. 다른 실시예에서, 제1 프로그램(6222)은 제1 애플리케이션(6220a)에 응답을 제공한다. 다른 실시예들에서, 가속화 프로그램(6120)은 인터-프로세스(inter-process) 통신 메커니즘 또는 인터페이스, 예컨대 API를 통해 제1 애플리케이션(6220a)에 응답을 제공할 수 있다. 일부 실시예들에서, 가속화 프로그램(6120)은 도 48에 설명한 바와 같이 멀티-체크 메시지 내의 제1 체크와 같은 응답의 일부를 수신할 뿐이다.

<871> 단계 6830에서, 가속화 프로그램(6120)은 서버(30)를 액세스하라는 제2 애플리케이션(6220b)의 요청을 인터셉트한다. 일부 실시예들에서, 가속화 프로그램(6120)은 단계 6825 전에 제2 애플리케이션(6220b)의 요청을 인터셉트한다. 다른 실시예들에서, 가속화 프로그램(6120)은 단계 6825에서 응답의 수신 동안에 제2 애플리케이션(6220b)의 요청을 인터셉트한다. 다른 실시예에서, 가속화 프로그램(6120)은, 클라이언트(6205) 또는 가속화 프로그램(6120)이 제1 애플리케이션(6220a)의 제1 요청에 대한 모든 응답 데이터를 수신하기 전에, 제2 애플리케이션(6220b)의 요청을 인터셉트한다. 제2 애플리케이션(6220b)의 요청을 가로챌 때, 가속화 프로그램(6120)은 일 실시예에서 단계 6815로 진행하여, 전송 계층 연결을 통해 제2 요청을 다중화할 지 또는 다른 전송 계층 연결, 예컨대 연결들의 풀 내의 다른 연결을 개설할 지를 판단한다. 다른 실시예들에서, 가속화 프로그램(6120)은 제1 애플리케이션(6220a)과 동일한 연결을 통해 제2 애플리케이션(6220b)의 요청을 전송하는 반면에, 제1 애플리케이션(6220a)은 ACK되지 않은 응답을 갖고 있거나 제1 요청의 응답으로부터 모든 데이터를 수신하지 않는다. 다른 실시예에서, 가속화 프로그램(6120)은 제1 애플리케이션(6220a)이 응답을 수신한 후 그리고 발생된 RST 전에 및/또는 FIN 명령이 제1 애플리케이션(6220a)과 관련하여 발생되기 전에 제2 애플리케이션(6220b)의 요청을 전송한다.

<872> 가속화 프로그램(6120)은 일반적으로 가속화 기술의 클라이언트측 구현 및 실행과 관련하여 설명되었지만, 가속화 프로그램(6120)은 어플라이언스측 가속화 기술을 구현 및 실행하는 어플라이언스(1250)와 인터페이스하고 함께 일을 한다. 일 실시예에서, 클라이언트측 가속화 프로그램(6120) 및 어플라이언스(1250)는 클라이언트(6205)와 서버(30)간의 통신을 위해 복수의 가속화 기술을 수행하기 위해 서로 함께 일을 한다. 일부 실시예들에서, 클라이언트측 가속화 프로그램(120) 및 어플라이언스(1250)는, 예컨대 클라이언트(6205)와 서버(30)간의 종속 연결 또는 단대단 폴링 및 다중화 메커니즘을 제공하기 위해, TCP 폴링 및 다중화를 제공한다. 예컨대, 가속화 프로그램(6120)은 제1 폴링된 전송 계층 연결을, 서버(30)에 대한 제2 폴링된 전송 계층 연결을 제공하는 어플라이언스(1250)에 제공할 수 있다. 다른 예에서, 가속화 프로그램(6120)은 클라이언트(6205) 상에서 제1 폴링된 전송 계층 연결을 통해 애플리케이션 요청을 다중화할 수 있고, 요청은 서버(30)로의 제2 폴링된 전송 계층 연결을 통해 어플라이언스(1250)에 의해 다중화된다. 일부 실시예들에서, 가속화 프로그램(120)은 클라이언트(6205)로부터 요청을 전송하기 위한 조절 메커니즘(throttling mechanism)을 제공하는 반면에, 어플라이언스(1250)는 서버(30)로부터의 응답을 클라이언트(6205)로 전송하기 위한 조절 메커니즘을 제공한다. 다른 실시예에서, 가속화 프로그램(6120)은 클라이언트(6205)에 대해 클라이언트측 캐싱을 수행하는 반면에, 어플라이언스(1250)는 다른 클라이언트(6205)와 함께 클라이언트(6205)를 위해 동적으로 발생된 객체와 같은 객체의 캐싱을 제공한다.

<873> 일부 실시예들에서, 클라이언트(6205) 및/또는 어플라이언스 상에서 가속화 기술을 수행하는 외에 또는 그 가속화 기술을 수행함과 함께, 가속화 프로그램(6120) 및 어플라이언스는 어플라이언스(1250)를 통해 클라이언트(6205)와 네트워크(40) 사이에서 가상 사설망 연결 및 통신을 제공할 수 있다. 다른 실시예에서, 가속화 프로

그램(6120)은 애플리케이션(6220)으로부터 전달되는 데이터를 압축할 수 있고, 어플라이언스(1250)는 압축된 데이터를, 수신시, 압축 해제할 수 있다. 반대로, 어플라이언스(1250)는 사실 데이터 통신망(40') 상의 서버(30) 상의 애플리케이션(6220)으로부터 전달되는 데이터를 압축할 수 있고, 가속화 프로그램(6120)은 압축 데이터를, 수신시, 압축 해제할 수 있다. 또한, 가속화 프로그램(6120) 및 어플라이언스(1250)는 암호화된 데이터 통신 또는 터널링 세션에서 엔드포인트의 역할을 하여, 가속화 프로그램(6120)은 애플리케이션(6220)으로부터 전달되는 데이터를 암호화하고, 어플라이언스(1250)는 암호화된 데이터를, 수신시, 암호 해독한다. 유사한 방식으로, 어플라이언스(1250)는 사실 데이터 통신망 상의 애플리케이션(6220)으로부터 전달되는 데이터를 암호화하고 가속화 프로그램(6120)은 데이터를, 수신시, 암호 해독한다.

<874> D. 컴퓨팅 환경의 제공을 가속시키는 예

B 절 및 C 절에서 위에서 설명된 구조, 기능, 및 동작을 고려하여, 일부 실시예들에서, 클라이언트로의 컴퓨팅 환경의 제공이 가속될 수 있다. 예컨대, 여기서 설명된 실시예들은 중앙 회사 데이터 센터로부터 원격 사용자 위치, 예컨대 회사의 지점 사무실로 애플리케이션에 의해 액세스 가능한 스트리밍 애플리케이션 및 데이터 파일을 전달하는데 사용될 수 있다. 어플라이언스 및 가속화 프로그램은 서버로부터 원격 클라이언트로의 스트리밍된 애플리케이션 및 데이터 파일과 같은 전송 계층 페이로드를 가속시키기 위한 단대단 가속화 기술을 제공한다. 애플리케이션 제공 관리 시스템은 복수의 실행 방법에 기초하여 그리고 정책 엔진을 통해 적용되는 인증 및 권한 검증 정책에 기초하여 컴퓨팅 환경을 원격 사용자의 데스크톱에 전달하기 위해 애플리케이션 제공 기술을 제공한다. 이들 기술로, 원격 사용자는 컴퓨팅 환경을 얻을 수 있고 네트워크 연결 디바이스로부터 서버 저장된 애플리케이션 및 데이터 파일을 액세스할 수 있다.

이제 도 49a를 참조하면, 위에서 설명된 가속화 및 애플리케이션 제공 시스템 및 방법을 실시하기 위한 일 실시예가 도시되어 있다. 간단히 살펴보면, 클라이언트(10)는 네트워크(40, 40') 및 어플라이언스(1250)를 통해 서버(30)와 통신한다. 예컨대, 클라이언트(10)는 회사의 원격 사무실, 예컨대, 지점 사무실에 존재할 수 있고, 서버(30)는 회사 데이터 센터에 존재할 수 있다. 클라이언트(10)는 클라이언트 에이전트(560) 및 컴퓨팅 환경(15)을 포함한다. 컴퓨팅 환경(15)은 데이터 파일 액세스, 처리 또는 사용하는 애플리케이션을 실행 또는 동작시킬 수 있다. 컴퓨팅 환경(15), 애플리케이션 및/또는 데이터 파일은 어플라이언스(1250) 및/또는 서버(30)를 통해 전달될 수 있다. 일부 실시예들에서, 클라이언트(10)는 또한 가속화 프로그램(4120), 수집 에이전트(404), 및 스트리밍 클라이언트(562)를 포함한다. 서버(30)는 애플리케이션 제공 시스템(500), 그리고 일부 실시예들에서는 정책 엔진(406)을 포함한다.

일 실시예에서, 애플리케이션 제공 시스템(500)은 서버(30) 상에서 존재 또는 실행될 수 있다. 다른 실시예에서, 애플리케이션 제공 시스템(500)은 복수의 서버(30-30") 상에서 존재 또는 실행될 수 있다. 일부 실시예들에서, 애플리케이션 제공 시스템(500)은 서버 팜(server farm)에서 실행될 수 있다. 일 실시예에서, 애플리케이션 제공 시스템(500)을 실행하는 서버(30)는 또한 애플리케이션 및 데이터 파일을 저장 또는 제공한다. 다른 실시예에서, 제1 세트의 하나 이상의 서버(30)는 애플리케이션 제공 시스템(500)을 실행할 수 있고, 다른 서버(30')는 애플리케이션 및 데이터 파일 저장 또는 제공할 수 있다. 일부 실시예들에서, 애플리케이션 제공 시스템(500), 애플리케이션, 및 데이터 파일의 각각은 다른 서버 상에 존재 또는 위치될 수 있다. 일 실시예에서, 애플리케이션 제공 시스템(500)은 또한 정책 엔진(406)을 포함한다. 다른 실시예에서, 정책 엔진(406)은 애플리케이션 제공 시스템(500)과는 별개로 실행된다. 일부 실시예들에서, 정책 엔진(406)은 애플리케이션 제공 시스템(500)과 동일한 서버(30) 상에 있다. 다른 실시예들에서, 정책 엔진(406)은 어플라이언스(1250) 상에서 실행된다. 또 다른 실시예에서, 애플리케이션 제공 시스템(500) 및/또는 정책 엔진(406)의 어떤 부분은 어플라이언스(1250) 또는 복수의 어플라이언스상에 존재, 실행 또는 저장될 수 있다.

일부 실시예들에서, 클라이언트 에이전트(560)는 위에서 이전에 설명한 바와 같이 스트리밍 클라이언트(562), 수집 에이전트(404), 및/또는 가속화 프로그램(6120)을 포함한다. 일 실시예에서, 클라이언트 에이전트(560), 스트리밍 클라이언트(562), 수집 에이전트(404), 및/또는 가속화 프로그램(6120)은 각각의 기능, 로직 및 동작을 제공하는 단일 프로그램 또는 한 세트의 실행 가능 명령을 개설하거나 그 단일 프로그램 또는 한 세트의 실행 가능 명령에 포함된다. 다른 실시예들에서, 스트리밍 클라이언트(562), 수집 에이전트(404), 및 가속화 프로그램(6120)의 각각은 클라이언트 에이전트(560)와 별도로 실행된다. 일 실시예에서, 클라이언트(10)는 클라이언트 에이전트(560)를 실행한다. 다른 실시예에서, 클라이언트(10)는 스트리밍 클라이언트(562)를 실행한다. 일부 실시예들에서, 클라이언트(10)는 수집 에이전트(404)를 실행한다. 일 실시예에서, 클라이언트(10)는 가속화 프로그램(6120)을 실행한다. 일부 실시예들에서, 클라이언트(10)는 스트리밍 클라이언트(562), 수집 에이전트(404), 또는 가속화 프로그램(6120) 중 하나 이상과 함께 클라이언트 에이전트(560)를 실행한다. 다른 실시

예들에서, 클라이언트(10)는 스트리밍 클라이언트(562) 및 가속화 프로그램(6120)을 실행한다. 일 실시예에서, 클라이언트(10)는 가속화 프로그램(6120) 및 수집 에이전트(404)를 실행한다.

<879>

일부 실시예들에서, 클라이언트(10)는 서버(30)로부터 클라이언트 에이전트(560), 스트리밍 클라이언트(562), 및/또는 수집 에이전트(404)를 획득한다. 다른 실시예들에서, 클라이언트(10)는 어플라이언스(1250)으로부터 클라이언트 에이전트(560), 스트리밍 클라이언트(562), 및/또는 수집 에이전트(404)를 획득한다. 일 실시예에서, 클라이언트 에이전트(560), 스트리밍 클라이언트(562), 및/또는 수집 에이전트(404)는 어플라이언스(1250) 상에 저장될 수 있다. 예컨대, 일부 실시예들에서, 클라이언트 에이전트(560), 스트리밍 클라이언트(562), 및/또는 수집 에이전트(404)는 어플라이언스(1250) 내에 캐싱될 수 있다. 다른 실시예들에서, 어플라이언스(1250)이 애플리케이션이 가속될 수 있다고 판단할 때, 어플라이언스(1250)는 클라이언트 에이전트(560), 스트리밍 클라이언트(562), 가속화 프로그램(6120) 및/또는 수집 에이전트(404)를 클라이언트(10)에 전송할 수 있다. 일부 실시예들에서, 클라이언트(10)는 클라이언트 에이전트(560), 스트리밍 클라이언트(562), 가속화 프로그램(6120) 및/또는 수집 에이전트(404)를 자동으로 설치 및 실행할 수 있다. 또 다른 실시예에서, 클라이언트 에이전트(560), 스트리밍 클라이언트(562), 가속화 프로그램(6120) 및/또는 수집 에이전트(404)는 클라이언트의 사용자 또는 애플리케이션에, 또는 클라이언트의 네트워크 스택의 어떤 부분에 투명하게 실행될 수 있다.

<880>

일부 실시예들에서, 어플라이언스(1250)는 클라이언트(10)를 위해 서버(30) 또는 네트워크(40')에 VPN 또는 SSL VPN 연결을 개설한다. 다른 실시예들에서, 어플라이언스(1250)는 하나 이상의 서버(30)에 대한 액세스를 제공하기 위해 프록시, 액세스 서버 또는 부하-밸런서의 역할을 한다. 일 실시예에서, 어플라이언스(1250) 및/또는 가속화 프로그램(6120)은 클라이언트(10)에의 스트리밍 클라이언트(562), 수집 에이전트(404), 및/또는 클라이언트 에이전트(560)의 제공을 가속시킨다. 일 실시예에서, 어플라이언스(1250)는 클라이언트(10)에의 가속화 프로그램(6120)의 제공을 가속시킨다. 다른 실시예들에서, 어플라이언스(1250) 및/또는 가속화 프로그램(6120)은 클라이언트(10)에의 컴퓨팅 환경(15), 애플리케이션, 및/또는 데이터 파일의 제공을 가속시킨다. 일 실시예에서, 클라이언트(10)는 컴퓨팅 환경(15) 및 어플라이언스(1250)를 갖고 있고 및/또는 가속화 프로그램(6120)은 애플리케이션 및/또는 데이터 파일의 제공을 가속시킨다. 일 실시예에서, 어플라이언스(1250) 및/또는 가속화 프로그램(6120)은 애플리케이션의 제공을 가속시킨다. 다른 실시예에서, 어플라이언스(1250) 및/또는 가속화 프로그램(6120)은 데이터 파일의 제공을 가속시킨다. 또 다른 실시예에서, 어플라이언스(1250) 및/또는 가속화 프로그램(6120)은 여기서 이전에 설명된 실행 환경 또는 가상화된 실행 환경과 같은 컴퓨팅 환경(15)의 제공을 가속시킨다.

<881>

일 실시예에서, 어플라이언스(1250)는 컴퓨팅 환경(15), 애플리케이션, 및/또는 데이터 파일이 가속될 수 있는지를 판단하기 위해 수집 에이전트(404)로부터 수집되는 정보를 이용한다. 일부 실시예들에서, 애플리케이션(1250)의 정책 엔진은 정책 엔진(406)을 포함한다. 다른 실시예들에서, 어플라이언스(1250)는 서버(30)로부터의 컴퓨팅 환경(15), 애플리케이션, 및/또는 데이터 파일을 액세스하기 위해 원격 사용자 또는 원격 클라이언트(10)의 인증 및/또는 권한 검증을 판단하기 위하여, 정책 엔진(406)과 통신 또는 인터페이스한다. 다른 실시예에서, 어플라이언스(1250)는 애플리케이션 제공 시스템(500)이 컴퓨팅 환경(15)), 애플리케이션, 및/또는 데이터 파일 중 하나 이상을 전달하게 하기 위해 원격 사용자 또는 원격 클라이언트(10)의 인증 및/또는 권한 검증을 판단하기 위하여, 정책 엔진(406)과 통신 또는 인터페이스한다. 또 다른 실시예에서, 어플라이언스(1250)는 정책 엔진(404)의 원격 사용자 또는 원격 클라이언트(10) 인증 및/또는 권한 검증에 기초하여 VPN 또는 SSL VPN 연결을 개설한다. 일 실시예에서, 어플라이언스(1250)는 정책 엔진(406)의 정책에 기초하여 네트워크 트래픽의 흐름 및 통신 세션을 제어한다. 예컨대, 어플라이언스(1250)는 정책 엔진(406)에 기초하여 컴퓨팅 환경(15), 애플리케이션 또는 데이터 파일에 대한 액세스를 제어할 수 있다.

<882>

이제 도 49b를 참조하면, 원격 위치에 있는 원격 클라이언트의 사용자로의 컴퓨팅 환경의 제공을 가속시키는 방법의 일 실시예가 도시되어 있다. 방법 8000을 간단히 살펴보면, 단계 8005에서, 서버(30)는 클라이언트(10) 상에서 애플리케이션을 실행하라는 요청을 수신한다. 단계 8010에서, 서버(30)는 실행하기 위한 애플리케이션을 클라이언트(10)에 스트리밍한다. 단계 8015에서, 어플라이언스(1250) 및/또는 클라이언트-측 가속화 프로그램(6120)은 애플리케이션의 클라이언트(10)로의 전송 또는 제공을 가속시킨다. 단계 8020에서, 클라이언트(10) 또는 애플리케이션은 애플리케이션이 이용할 수 있도록 서버(30)로부터의 데이터 파일을 요청한다. 단계 8025에서, 서버(30) 및/또는 어플라이언스(1250)는 데이터 파일을 클라이언트(10)에 전송한다. 단계 8030에서, 어플라이언스(1250) 및/또는 클라이언트-측 가속화 프로그램(6120)은 클라이언트(10)로의 데이터 파일의 전송 또는 제공을 가속시킨다.

<883>

보다 상세하게는, 단계 8005에서, 서버(30)는 클라이언트(10) 상에서 애플리케이션을 실행하라는 요청을 수신한

다. 일부 실시예들에서, 클라이언트(10)의 사용자가 요청을 한다. 다른 실시예들에서, 애플리케이션, 운영체제 또는 컴퓨팅 환경(15)이 요청을 전송한다. 다른 실시예에서, 어플라이언스(1250)는 클라이언트(10)로부터의 요청을 인터셉트하고 요청을 서버(30)에 전송한다. 일 실시예에서, 어플라이언스(1250)는 사용자 또는 클라이언트(10)의 인증 및/또는 권한 검증에 기초하여 요청을 서버(30)에 전송한다. 다른 실시예에서, 어플라이언스(1250)는 수집 에이전트(404)에 의해 제공된 정보에 기초하여 요청을 서버(30)에 전송한다. 일 실시예에서, 요청은 복수의 실행 방법 중 한 방법에 의해 애플리케이션을 실행하라는 요청을 포함한다. 예컨대, 클라이언트(10)의 사용자는 애플리케이션을, 서버로부터 스트리밍되는 애플리케이션으로서, 국부적으로 설치 및 실행되는 애플리케이션으로서, 또는 서버(30) 상에서 실행되고 클라이언트(10)에 원격 디스플레이되는 서버-기반 애플리케이션으로서 실행하라고 요청한다. 일부 실시예들에서, 요청은 파일-유형 연관성에 기초한다. 예컨대, 사용자는 파일을 판독 또는 액세스하는데 이용되는 애플리케이션과 연관된 파일을 선택할 수 있다.

<884> 단계 8010에서, 단계 8005의 요청에 응답하여, 서버(30)는 실행하기 위한 애플리케이션을 클라이언트(10)에 전송한다. 일부 실시예들에서, 서버(30)는 애플리케이션을 클라이언트(10)에 스트리밍한다. 예컨대, 일부 실시예들에서 애플리케이션을 스트리밍함으로써, 애플리케이션은 설치 없이 클라이언트(10) 상에서 동작한다. 다른 실시예들에서, 서버(30)는 로컬 설치 및 실행을 위한 애플리케이션을 클라이언트(10)에 전송한다. 예컨대, C절에서 가속화 프로그램(6120)과 함께 설명된 자동 설치 및 실행 기술을 이용하여, 클라이언트(10)는 애플리케이션을, 수신시, 자동으로 설치 및 실행할 수 있다. 다른 실시예에서, 서버(30)는 클라이언트를 위해 서버 상의 애플리케이션을 실행하고, 원격 디스플레이 또는 프리젠테이션 계층 프로토콜을 통해 디스플레이를 클라이언트(10)에 전송한다. 또 다른 실시예에서, 어플라이언스(1250)는 설치 및/또는 실행을 위해 애플리케이션을 클라이언트(10)에 스트리밍하거나 애플리케이션을 클라이언트(10)에 전송한다. 일부 실시예들에서, 어플라이언스(1250) 및/또는 서버(30)는 애플리케이션을 포함한 컴퓨팅 환경(15)을 전송한다. 다른 실시예들에서, 어플라이언스(1250) 및/또는 서버(30)는 요청에 응답하여 컴퓨팅 환경(15)을 전송한다.

<885> 단계 8015에서, 어플라이언스(1250) 및/또는 가속화 프로그램(6120)은 실행하기 위한 애플리케이션의 클라이언트(10)로의 제공을 가속시킨다. 일 실시예에서, 어플라이언스(1250)는 C절에서 위에서 설명된 복수의 가속화 기술 중 하나 이상을 수행 또는 적용한다. 다른 실시예에서, 가속화 프로그램(6120)은 또한 위에서 C 절에서 설명된 복수의 클라이언트측 가속화 기술의 하나 이상을 수행 또는 적용한다. 일부 실시예들에서, 가속화 프로그램(1250) 및 어플라이언스(6120)은 클라이언트(10) 및 어플라이언스(1250) 상에서 복수의 가속화 기술을 수행하기 위해 서로 함께 일을 한다. 예컨대, 가속화 프로그램(6120)은 제1 세트의 하나 이상의 가속화 기술을 수행할 수 있으나, 어플라이언스(1250)는 제2 세트의 하나 이상의 가속화 기술을 수행할 수 있다. 일 실시예에서, 가속화 프로그램(1250) 및 어플라이언스(6120)은 동일한 가속화 기술을 수행한다. 다른 실시예에서, 가속화 프로그램(1250) 및 어플라이언스(6120)은 다른 가속화 기술을 수행한다.

<886> 일 실시예에서, 어플라이언스(1250) 및/또는 가속화 프로그램(6120)은 클라이언트(10)와 서버(30)간의 전송 계층 연결을 통해 페이로드를 가속시킨다. 일부 실시예들에서, 서버(30)는 전송 계층 연결을 통해 애플리케이션을 TCP/IP 패킷의 페이로드와 같은 하나 이상의 데이터 파일로서 스트리밍한다. 다른 실시예들에서, 서버(30)는 애플리케이션 계층 프로토콜 또는 전송 계층 연결 상의 스트리밍 프로토콜을 통해 애플리케이션을 스트리밍한다. 다른 실시예에서, 서버(30)는 전송 계층 연결을 통해 ICA 또는 RDP 프로토콜을 통해서 디스플레이 출력을 전송한다. 이들 실시예들에서, 어플라이언스(1250) 및/또는 가속화 프로그램(6120)은 전송 계층 패킷의 페이로드를 통해 애플리케이션의 제공을 가속시킨다.

<887> 단계 8020에서, 클라이언트(10)는 애플리케이션 또는 컴퓨팅 환경(15)이 이용할 데이터 파일에 대한 요청을 전송한다. 일부 실시예들에서, 데이터 파일에 대한 요청은 단계 8005에서 애플리케이션을 실행하라는 요청과 함께 전송된다. 일 실시예에서, 애플리케이션을 실행하라는 요청은 데이터 파일에 대한 요청을 포함한다. 다른 실시예들에서, 애플리케이션 또는 컴퓨팅 환경은 애플리케이션 또는 컴퓨팅 환경의 어떤 기능, 동작, 또는 로직을 수행하는 과정에서 데이터 파일을 요청한다. 예컨대, 애플리케이션 또는 컴퓨팅 환경(15)은 서버(30)로부터 어떤 매크로, 스크립트, 구성 데이터, 프로파일, 템플릿 또는 규칙을 요청할 수 있다. 일부 실시예들에서, 애플리케이션 또는 컴퓨팅 환경(15)의 사용자는 애플리케이션 또는 컴퓨팅 환경으로 데이터 파일을 판독, 액세스 또는 처리하기 위해 데이터 파일을 요청한다. 예컨대, 사용자는 애플리케이션을 통한 편집을 위해 파일을 열 수 있는데, 예컨대 워드 처리 애플리케이션을 통한 편집을 위해 문서를 열 수 있다. 일부 실시예들에서, 사용자는 데이터 파일을 요청하기 위해 파일을 컴퓨팅 환경의 애플리케이션에 드래그 및 드롭시킨다. 다른 실시예들에서, 사용자는 중앙 서버의 네트워크 구동기와 같은 네트워크 또는 원격 기억 장치 시스템의 기억 장

치에 파일 및 디렉토리 인터페이스, 예컨대, 윈도우즈 운영체제 내의 파일 익스플로러를 통해 데이터 파일을 요청할 수 있다.

<888> 단계 8025에서, 서버(30) 또는 어플라이언스(1250)는 요청된 데이터 파일을 클라이언트(10)에 전송한다. 일부 실시예들에서, 서버(30) 또는 어플라이언스(1250)는 단계 8020의 요청에 응답하여 데이터 파일을 클라이언트(10)에 전송한다. 다른 실시예들에서, 서버(30) 또는 어플라이언스(1250)는 클라이언트(10)로부터의 요청 없이도 데이터 파일을 클라이언트(10)에 전송한다. 예컨대, 서버(30)는 데이터 파일 업데이트를 클라이언트(10)에 "푸시(push)"할 수 있다. 일 실시예에서, 서버(30)는 요청된 데이터 파일을 클라이언트(10)에 전송한다. 다른 실시예에서, 어플라이언스(1250)는 요청된 데이터 파일을 클라이언트(10)에 전송한다. 예컨대, 일 실시예에서, 어플라이언스(1250)는 데이터 파일에 대한 요청을 인터셉트해서, 데이터 파일을 위해 어플라이언스(1250)의 캐시를 체크하고, 캐싱된 데이터 파일을 클라이언트(10)에 전송한다. 또 다른 실시예에서, 가속화 프로그램(6120)은 클라이언트(10)에서 데이터 파일 요청을 인터셉트하고 가속화 프로그램(6120)의 캐시를 통해 데이터 파일을 클라이언트(10)에 제공한다. 일부 실시예들에서, 어플라이언스(1250) 또는 서버(30)는 스트리밍 프로토콜, 또는 스트림을 통해 데이터 파일을 전송한다. 다른 실시예들에서, 어플라이언스(1250) 또는 서버(30)는 임의의 유형 및 형태의 캐싱 프로토콜을 통해 데이터 파일을 전송한다.

<889> 단계 8030에서, 어플라이언스(1250) 및/또는 가속화 프로그램(6120)은 클라이언트(10)에의 데이터 파일의 제공 또는 전송을 가속시킨다. 일부 실시예들에서, 데이터 파일은 전송 계층 프로토콜 상의 애플리케이션 계층 프로토콜과 같은 임의의 유형 및 형태의 프로토콜을 통해 전송될 수 있다. 일 실시예에서, 어플라이언스(1250)는 데이터 파일의 전송을 가속시킨다. 다른 실시예에서, 가속화 프로그램(6120)은 데이터 파일의 전송을 가속시킨다. 일부 실시예들에서, 어플라이언스(1250)는 가속화 프로그램(1250)과 함께 데이터 파일의 전송을 가속시킨다. 여기서 설명되는 바와 같이, 어플라이언스(1250) 및/또는 가속화 프로그램(6120)은 데이터 파일의 전송을 가속시키기 위해 클라이언트(10) 및 어플라이언스(30)에 대해 복수의 가속화 기술 중 하나 이상을 수행할 수 있다. 일부 실시예들에서, 어플라이언스(1250) 및/또는 가속화 프로그램(6120)은 애플리케이션 또는 컴퓨팅 환경(15)이 이용할 클라이언트(10) 또는 어플라이언스(1250) 상에 하나 이상의 데이터 파일을 캐싱할 수 있다.

<890> 대표 예

<891> 일 실시예로서, 사용자는 로컬 머신(10) 상에 일하는 지점 사무실에 위치될 수 있다. 사용자는 중앙 사무실에 위치한 원격 머신(30)에 존재하는, 회사 문서를 편집하기 위해 마이크로소프트 워드와 같은 워드 처리 애플리케이션을 이용하기를 원할 수 있다. 사용자는 웹 브라우저로 원격 머신(30)에 의해 호스팅되는 회사 웹 사이트를 네비게이팅할 수 있다. 일단 사용자가 원격 머신(30)에 의해 인증되면, 원격 머신(30)은 로컬 머신(10)이 액세스하는 애플리케이션 프로그램을 나타내는 그래픽 아이콘이 나타난 도 3a 및 도 3b에서 여기서 설명한 바와 같이 프로그램 네이버후드(Program Neighborhood) 윈도우를 포함하는 HTML 페이지를 준비하여 로컬 머신(10)에 전송할 수 있다. 로컬 머신(10)의 사용자는 아이콘을 클릭함으로써 애플리케이션의 실행을 호출할 수 있다. 정책 엔진은 도 4a 내지 도 4d에 설명한 바와 같이 로컬 머신(10)이 워드 처리 애플리케이션을 액세스할 수 있는지의 여부 및 액세스 하는 방법을 결정할 수 있다. 애플리케이션은 도 20 및 도 21에서 설명된 기술을 이용하여 국부적으로 설치 및 실행될 수 있다. 다음에, 사용자는 편집을 위해 원격 머신(30) 상의 문서를 선택하기 위해 애플리케이션을 이용할 수 있다. 어플라이언스(1250)는 TCP 다중화와 같은 어떤 여기서 설명된 기술을 이용하여 로컬 머신(10)에의 파일의 제공을 가속시킬 수 있다.

<892> 다른 예로서, 제2 사용자는 로컬 머신(10) 상에서 일하는 지점 사무실에 위치될 수 있다. 사용자는 사용자의 회사 계정을 통해 첨부된 파일을 포함하는 이메일을 액세스하기를 원할 수 있다. 이메일 애플리케이션 및 이메일 데이터 파일은 중앙 사무실에 존재할 수 있다. 이메일 애플리케이션을 액세스하라는 사용자 요청시, 정책 엔진은 도 4a 내지 도 4d에서 설명한 바와 같이 여기서 설명된 스트리밍 기술을 이용하여 이메일 애플리케이션을 사용자에게 스트리밍하도록 판단할 수 있다. 정책 엔진은 또한 여기서 설명한 바와 같이 로컬 머신(10) 상에 가속화 프로그램을 설치하는 것으로 판단할 수 있다. 애플리케이션 스트리밍은 동적 캐싱과 같은 여기서 설명된 기술을 이용하여 가속화될 수 있다. 로컬 설치시, 사용자는 이메일 및 첨부 파일을 볼 수 있다. 어플라이언스(1250)는 여기서 설명한 바와 같이 TCP 폴링과 같은 가속화 기술을 이용하여 파일의 제공을 가속시킬 수 있다. 어플라이언스는 또한 나중의 요청을 가속시키기 위해 원격 머신에 전달된 데이터 파일의 일부 또는 모두를 캐싱할 수 있다. 캐싱은 가속화 프로그램과 함께 어플라이언스(1250) 또는 로컬 머신(10) 상에서 행해질 수 있다.

- <893> 제3 예로서, 지점 사무실에 위치한 사용자는 스프레드시트를 업데이트하기 위해 마이크로소프트 엑셀과 같은 스프레드시트 프로그램을 액세스하기를 원할 수 있다. 사용자는 중앙 사무실에서 원격 머신(30)에의 SSL 연결을 개설하기 위해 로컬 머신(10)을 이용할 수 있고, 도 3a 및 도 3b에 설명한 바와 같이 프로그램 네이버후드로부터 스프레드시트 애플리케이션을 선택할 수 있다. 수집 에이전트는 도 4d에 설명한 바와 같이 스프레드시트 애플리케이션이 로컬 머신(10)에 스트리밍될 수 있는지를 판단하기 위해 로컬 머신에 관한 정보를 수집할 수 있다. 스프레드시트 애플리케이션은 SSL 연결을 통해 로컬 머신(10)에 스트리밍될 수 있다. SSL 연결은 여기서 설명된 바와 같은 SSL 또는 TCP 연결 풀링 및 다중화를 제공하는 어플라이언스(1250)에 의해 가속될 수 있다. 애플리케이션 스트리밍은 또한 여기서 설명된 동적 캐싱 기술을 제공하는 어플라이언스(1250)에 의해 가속될 수 있다. 사용자는 편집을 위해 스프레드시트 애플리케이션 내로부터 파일을 선택할 수 있다. 로컬 머신(10)은 파일 요청을 원격 머신에 전송할 수 있다. 어플라이언스(1250)는 사용자로의 파일의 제공을 가속시키기 위해 여기서 설명된 압축 기술을 이용할 수 있다.
- <894> 일반적으로 위에서 애플리케이션 제공 시스템 및 어플라이언스가 클라이언트로의 컴퓨팅 환경의 제공을 가속시키는 것으로 설명되었지만, 애플리케이션 제공 시스템 및 어플라이언스는 복수의 컴퓨팅 환경, 애플리케이션 및/또는 데이터 파일을 클라이언트에 제공하는 것을 가속시킬 수 있다. 예컨대, 애플리케이션 제공 시스템 및 어플라이언스는 하나의 유형의 운영체제와 연관된 제1 컴퓨팅 환경 및 제2 유형의 운영체제와 연관된 제2 컴퓨팅 환경의 클라이언트로의 제공을 가속시킬 수 있다. 부가적으로, 애플리케이션 제공 시스템 및 어플라이언스는 복수의 클라이언트로의 컴퓨팅 환경, 애플리케이션, 및/또는 데이터 파일의 제공을 가속시킬 수 있다. 또한, 일반적으로 위에서 애플리케이션 제공 시스템 및 어플라이언스가 원격 사용자 또는 원격 클라이언트로의 컴퓨팅 환경의 제공을 가속시키는 것으로 설명되었지만, 애플리케이션 제공 시스템 및 어플라이언스는 서버의 LAN 상의 클라이언트와 같은 로컬, 원격 등의 클라이언트로의 컴퓨팅 환경, 애플리케이션, 및/또는 데이터 파일의 제공을 가속시킬 수 있다.
- <895> 또한, 클라이언트와 애플리케이션 제공 시스템 사이의 어플라이언스로서 일반적으로 위에서 설명되었지만, 복수의 어플라이언스는 하나 이상의 클라이언트와 하나 이상의 서버 사이에서 사용될 수 있다. 일부 실시예들에서, 제1 어플라이언스는 클라이언트의 네트워크상에 있고, 제2 어플라이언스는 서버의 네트워크상에 있다. 일 실시예에서, 제1 어플라이언스 및 제2 어플라이언스는 여기서 설명된 동작을 수행할 때 서로 통신한다. 예컨대, 제1 어플라이언스 및 제2 어플라이언스는 어플라이언스 통신 프로토콜에 대한 인터노드(internode), 고성능, 또는 어플라이언스를 통해 통신할 수 있다. 부가적으로, 복수의 애플리케이션 제공 시스템은 하나의 어플라이언스 또는 복수의 어플라이언스와 함께 사용될 수 있다. 애플리케이션 제공 시스템 및 어플라이언스는 다양한 네트워크 환경 및 인프라구조 아키텍처에 배치될 수 있다.
- <896> 실시예들은 하나 이상의 컴퓨터-판독 가능 프로그램으로서 제공될 수 있다. 제조 물품은 플로피 디스크, 하드 디스크, 콤팩트 디스크, DVD(digital versatile disk), 플래시 메모리 카드, PROM, RAM, ROM, 또는 자기 테이프일 수 있다. 일반적으로, 컴퓨터-판독 가능 프로그램은 프로그래밍 언어로 구현될 수 있다. 사용될 수 있는 언어의 일부 예로는 C, C++, C#, 또는 자바를 포함한다. 소프트웨어 프로그램은 하나 이상의 제조 물품 상에 또는 그 제조 물품 내에 객체 코드로서 저장될 수 있다.

도면의 간단한 설명

- <15> 본 발명의 전술한 특징 및 다른 특징들은 하기의 상세한 설명과 첨부된 도면을 통해 쉽게 이해될 수 있다. 도면은 본 발명을 한정하기 위한 것은 아니며, 예시적인 것에 불과하다.
- <16> 도 1a는 네트워크 환경을 도시한 블록도이다.
- <17> 도 1b는 네트워크 환경에서 원격의 컴퓨팅 환경 실시예를 도시한 블록도이다.
- <18> 도 1c 및 1d는 실시예들에 관련하여 유용한 컴퓨터 실시예를 도시한 블록도이다.
- <19> 도 1e는 컴퓨팅 환경을 클라이언트에 제공(delivery)하기에 적합한 환경을 도시한 블록도이다.
- <20> 도 1f는 웹 서비스 디렉토리의 GUI 퍼블리싱을 통해 로컬 머신에 가용한 복수의 애플리케이션 프로그램들을 제공하기 위한 시스템의 일 실시예를 도시한 블록도이다.
- <21> 도 2는 애플리케이션 프로그램의 실행 방법을 선택하기 위해 행해지는 단계들의 일 실시예를 도시한 흐름도이다.

- <22> 도 3a는 월드와이드웹을 통해 프로그램 근접 애플리케이션의 실행을 개시하는 로컬 머신의 일 실시예를 도시한 블록도이다.
- <23> 도 3b는 로컬 머신이 웹 서비스 디렉토리를 이용하여 목록된 애플리케이션 프로그램을 액세스하기 위해 로컬 머신에 의해 행해지는 단계들의 일 실시예를 도시한 흐름도이다.
- <24> 도 4a는 로컬 머신을 위해 애플리케이션 프로그램으로의 정책-기반 액세스를 제공하는 네트워크 환경의 실시예를 도시한 블록도이다.
- <25> 도 4b는 정책 엔진의 좀더 구체적인 실시예를 도시한 블록도이다.
- <26> 도 4c는 로컬 머신에 관하여 수신된 정보에 기반하여 액세스 제어를 판단하기 위해 정책 엔진에 의해 행해지는 단계들의 일 실시예를 도시한 흐름도이다.
- <27> 도 4d는 복수의 애플리케이션 세션으로 허가된 원격 액세스가 제공되는 컴퓨터 환경의 실시예를 도시한 블록도이다.
- <28> 도 4e는 로컬 머신과 이에 연관된 애플리케이션 세션을 연결하기 위해 세션 서버에 의해 행해지는 단계들의 일 실시예를 도시한 흐름도이다.
- <29> 도 5는 클라이언트 노드와 이에 연관된 애플리케이션 세션을 연결하기 위해 세션 서버에 의해 행해지는 단계들의 일 실시예를 도시한 흐름도이다.
- <30> 도 6은 애플리케이션 목록을 제공하는 관리 서비스를 포함하는 원격 머신의 일 실시예를 도시한 블록도이다.
- <31> 도 7은 애플리케이션 프로그램을 포함하는 복수의 파일들을 액세스하기 위해 행해지는 단계들의 일 실시예를 도시한 블록도이다.
- <32> 도 8a는 애플리케이션 호환성 및 애플리케이션 사회성(socialbility) 문제가 줄어든 운영 체제의 제어하에서 작동하는 컴퓨터의 일 실시예를 도시한 블록도이다.
- <33> 도 8b는 애플리케이션 호환성 및 애플리케이션 사회성 문제가 줄어든 다중-사용자 컴퓨터의 일 실시예를 도시한 블록도이다.
- <34> 도 8c는 프로세스와 격리 범위를 연관시키기 위한 방법의 단계들의 일 실시예를 도시한 흐름도이다.
- <35> 도 9는 애플리케이션 프로그램을 실행시키기 위한 방법의 단계들의 일 실시예를 도시한 흐름도이다.
- <36> 도 10은 원격 머신상에서 존재하는 복수의 애플리케이션 파일들의 일 실시예를 도시한 흐름도이다.
- <37> 도 11은 원격 저장된 파일들에 연관된 파일 메타데이터에 대한 요청에 로컬 저장된 정보를 이용하여 응답하는 (locally responding) 방법에서 행해지는 단계들의 일 실시예를 도시한 흐름도이다.
- <38> 도 12는 원격 저장된 파일들에 연관된 파일 메타데이터에 대한 요청에 로컬 저장된 정보를 이용하여 응답하는 (locally responding) 시스템의 일 실시예를 도시하는 블록도이다.
- <39> 도 13은 로컬 실행되는 애플리케이션 프로그램에 연관된 디렉토리 구조의 원격 파일을 액세스하기 위한 방법에서 행해지는 단계들의 일 실시예를 도시한 흐름도이다.
- <40> 도 14는 애플리케이션에 연관된 디렉토리 구조의 파일을 액세스하기 위한 시스템의 일 실시예를 도시한 블록도이다.
- <41> 도 15는 라이선스 관리 서브시스템을 포함하는 원격 머신의 일 실시예를 도시한 블록도이다.
- <42> 도 16은 원격 머신상의 관리 서비스의 컴포넌트들의 일 실시예들을 도시한 블록도이다.
- <43> 도 17은 원격 머신으로부터 라이선스를 요청하여 유지하기 위해 행해지는 단계들의 일 실시예를 도시한 흐름도이다.
- <44> 도 18은 관리 서비스에 의해 모니터링되는 세션에 연관된 상태들의 일 실시예를 도시한 블록도이다.
- <45> 도 19는 각각이 애플리케이션을 포함하는 복수의 애플리케이션 파일들을 포함하는 2개의 타겟을 포함하는 패키지의 일 실시예를 도시한 블록도이다.
- <46> 도 20은 운영체제를 재부팅하지 않고 애플리케이션 프로그램을 설치하기 위한 정책-기반 방법에서 행해지는 단

계들의 일 실시예를 도시한 흐름도이다.

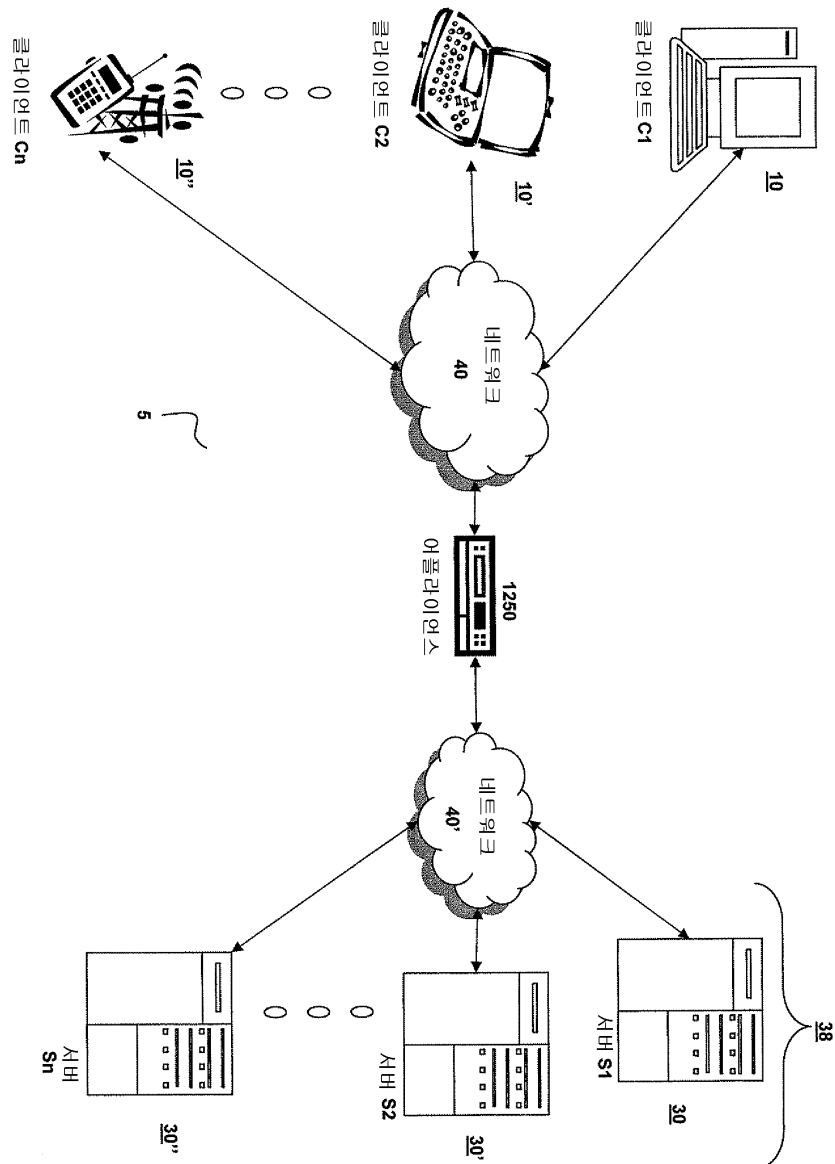
- <47> 도 21은 운영체제를 재부팅하지 않고 애플리케이션 프로그램을 설치하기 위한 정책-기반 방법에서 행해지는 단계들의 일 실시예를 도시한 흐름도이다.
- <48> 도 22는 로컬 머신상에서 실행될 스크립트 목록의 일 실시예를 도시한 스크린샷이다.
- <49> 도 23은 설치자 프로그램을 격리 환경에서 실행시키는 패키징 메커니즘을 포함하는 시스템의 일 실시예를 도시한 블록도이다.
- <50> 도 24는 설치자 프로그램의 실행이 운영체제의 재부팅을 필요로 하는 환경에서 행해지는 단계들의 일 실시예를 도시한 흐름도이다.
- <51> 도 25는 패키징 메커니즘이 애플리케이션 프로그램을 설치하는 원격 머신의 일 실시예를 도시한 블록도이다.
- <52> 도 26은 애플리케이션 격리 환경에 애플리케이션을 설치하기 위해 행해지는 단계들의 일 실시예를 도시한 흐름도이다.
- <53> 도 27은 통합 캐싱을 수행하는 어플라이언스의 구조의 일 실시예를 예시한 블록도이다.
- <54> 도 28a는 디바이스 동작들과 패킷 처리 및 패킷 처리 타이머를 통합하기 위한 방법의 일 실시예에서 행해지는 단계들의 흐름도이다.
- <55> 도 28b는 도 3a의 관점에서 무효화 정밀(invalidation granularity) 기법들을 실행하기 위한 방법의 일 실시예에서 행해지는 단계들의 흐름도이다.
- <56> 도 29a는 쓸모없는(stale) 객체들을 무효화하기 위해 무효화 명령을 이용하는 방법의 일 실시예에서 행해지는 단계들의 흐름도이다.
- <57> 도 29b는 객체 그룹의 무효화를 통합한 방법의 일 실시예에서 행해지는 단계들의 흐름도이다.
- <58> 도 29c는 클라이언트 요청이 객체 결정자를 위해 파싱되는 방법의 일 실시예에서 행해지는 단계들의 흐름도이다.
- <59> 도 29d는 객체 결정자를 이용하여 객체 그룹들을 무효화하는 것을 통합한 방법의 일 실시예에서 행해지는 단계들의 흐름도이다.
- <60> 도 30은 연결 풀링(connection pooling) 방법의 일 실시예에서 행해지는 단계들의 흐름도이다.
- <61> 도 31은 클라이언트 및 서버 요청을 해석하는 방법의 일 실시예에서 행해지는 단계들의 흐름도이다.
- <62> 도 32는 콘텐츠 길이 파라미터의 일 실시예를 도시한다.
- <63> 도 33은 청크 사이즈(chunk-size) 필드의 일 실시예를 도시한다.
- <64> 도 34는 연결 풀링의 일 실시예를 도시한 메시지 흐름도를 도시한다.
- <65> 도 35는 클라이언트와 서버간의 연결 풀링의 효율성을 높이도록 콘텐츠 길이 파라미터를 이용하기 위해 행해지는 단계들의 일 실시예를 예시한 상세 흐름도이다.
- <66> 도 36은 클라이언트와 서버간의 연결 풀링의 효율성을 높이도록 콘텐츠 길이 파라미터를 이용하기 위해 행해지는 단계들의 일 실시예를 도시한 흐름도이다.
- <67> 도 37은 클라이언트와 서버간의 연결 풀링의 효율성을 높이도록 청크 사이즈 필드를 이용하기 위해 행해지는 단계들의 일 실시예를 예시한 상세 흐름도이다.
- <68> 도 38은 클라이언트와 서버간의 연결 풀링의 효율성을 높이도록 청크 사이즈 필드를 이용하기 위해 행해지는 단계들의 일 실시예를 도시한 흐름도이다.
- <69> 도 39는 통합 캐싱 기능을 제공하기 위해 행해지는 단계들의 일 실시예를 도시한 흐름도이다.
- <70> 도 40a는 클라이언트측 가속화 프로그램의 일 실시예를 도시한 블록도이다.
- <71> 도 40b는 클라이언트측 가속화 프로그램을 제공하기 위한 시스템의 일 예를 도시한 블록도이다.
- <72> 도 41a는 클라이언트측 가속화 프로그램을 동적 제공하고 자동으로 설치 및 실행하는 방법 일 실시예의 흐름도

이다.

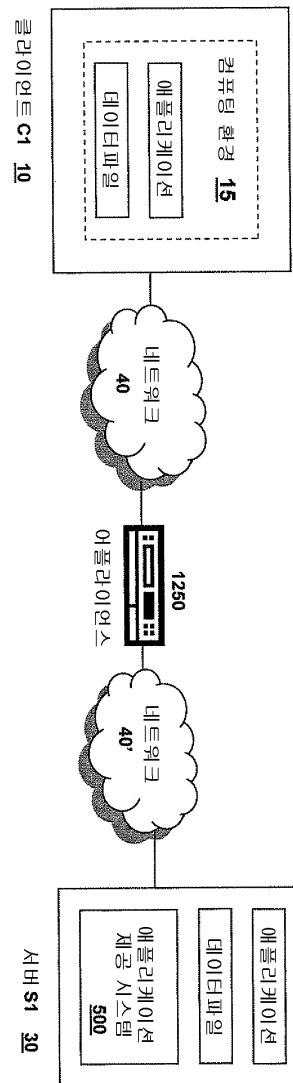
- <73> 도 41b는 애플리케이션을 결정하는 방법이 가속화될 수 있는 일 실시예의 흐름도이다.
- <74> 도 41c는 가속화 프로그램이, 전송 계층에서 인터셉트하고 커널-레벨 데이터 구조를 이용하는 복수의 가속화 기법을 행하는 방법의 또 다른 실시예의 흐름도이다.
- <75> 도 42a는 제1 프로그램을 통해 클라이언트상에서 가속화 프로그램을 자동으로 설치 및 실행하는 방법의 또 다른 실시예의 흐름도이다.
- <76> 도 42b는 제1 프로그램 및 가속화 프로그램이 가상의 사실 네트워크 연결을 제공하고 하나 이상의 가속화 기법을 수행하는 방법 일 실시예의 흐름도이다.
- <77> 도 43은 클라이언트의 통신 내용을 서버로 리디렉팅하여 중개자가 그 통신 내용을 서버로 전달하는 것이 유용하지 않다고 판단하는 것을 피하는 방법의 일 실시예의 흐름도이다.
- <78> 도 44는 전송 제어 프로토콜 버퍼링의 클라이언트측 가속화 기법을 수행하는 방법의 일 실시예의 흐름도이다.
- <79> 도 45a는 전송 제어 프로토콜 접속 폴링의 클라이언트측 가속화 기법을 수행하는 방법의 일 실시예의 흐름도이다.
- <80> 도 45b는 일 실시예에서 하나 이상의 전송 계층 연결의 풀을 통해서 복수의 애플리케이션이 수행하는 HTTP 트랜잭션 세트의 도면이다.
- <81> 도 46은 클라이언트측에서 전송 제어 프로토콜 다중화 기법을 수행하는 방법의 일 실시예의 흐름도이다.
- <82> 도 47은 전송 계층 패킷의 콘텐츠 길이 식별자의 일 실시예의 도면이다.
- <83> 도 48은 다중 청크를 통해 전송되는 메시지의 콘텐츠 길이 식별자의 또 다른 실시예의 도면이다.
- <84> 도 49a는 네트워크형 컴퓨터 시스템이 원격 클라이언트로 컴퓨팅 환경 제공을 가속화하는 일 실시예를 설명하는 블록도이다.
- <85> 도 49b는 원격 클라이언트로 컴퓨팅 환경의 제공을 가속화하기 위한 방법의 단계들의 일 실시예를 도시한 흐름도이다.

도면

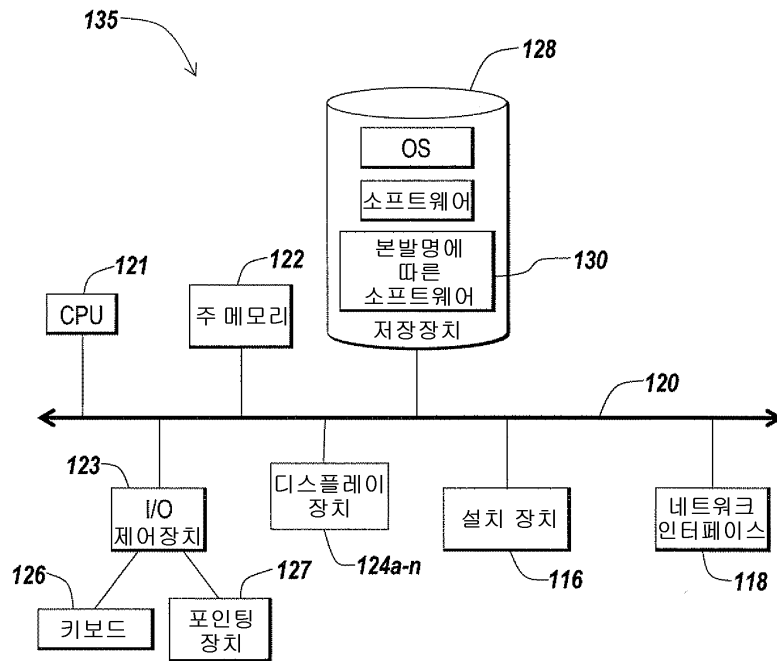
도면1a



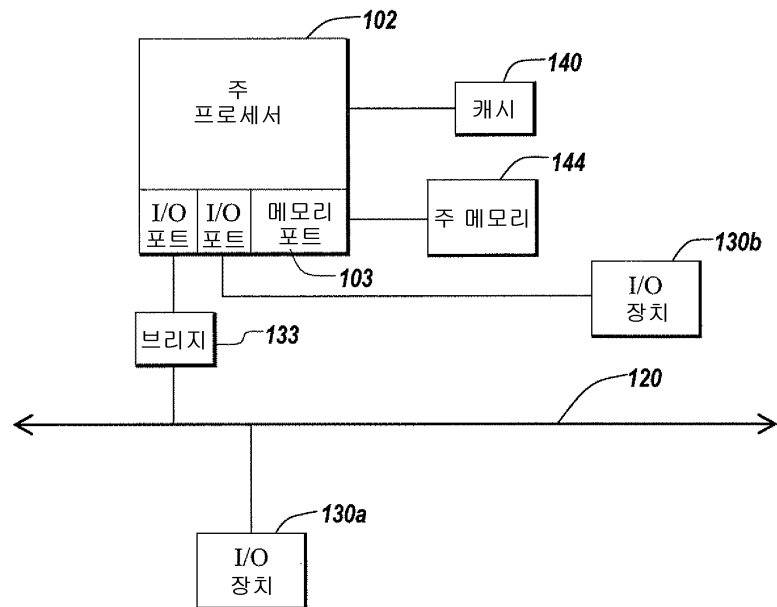
도면1b



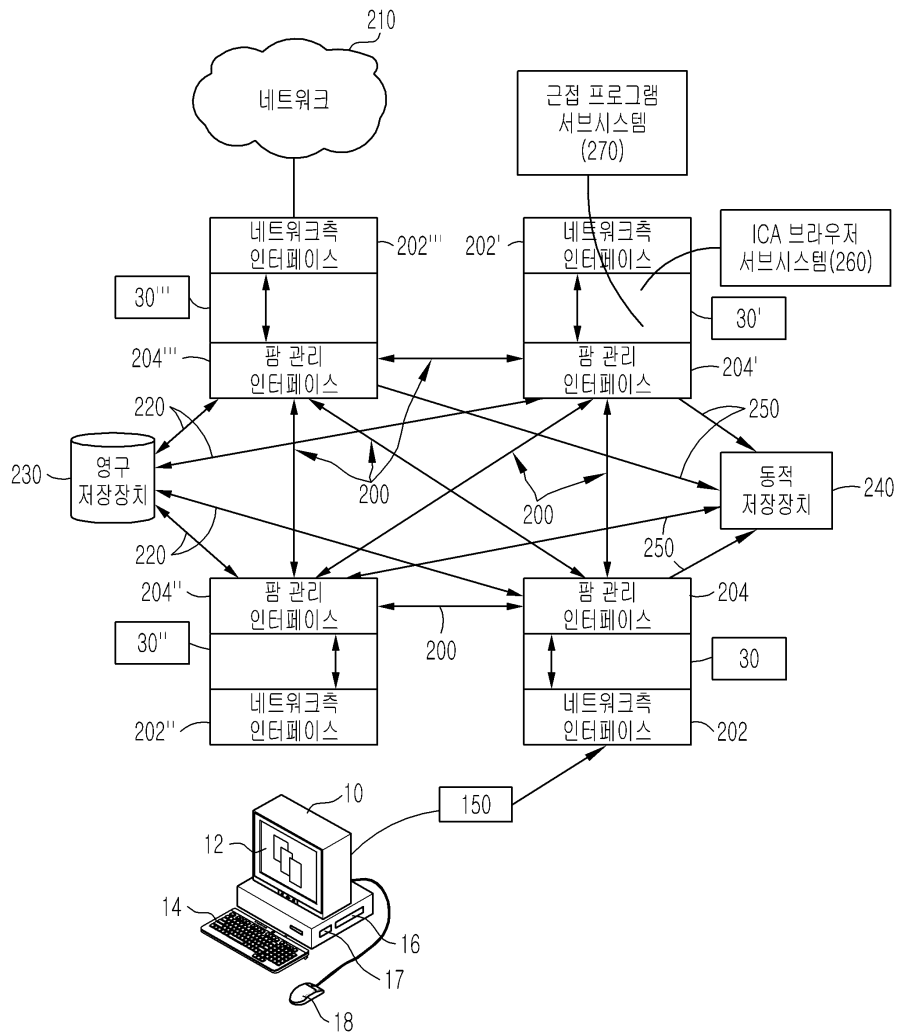
도면1c



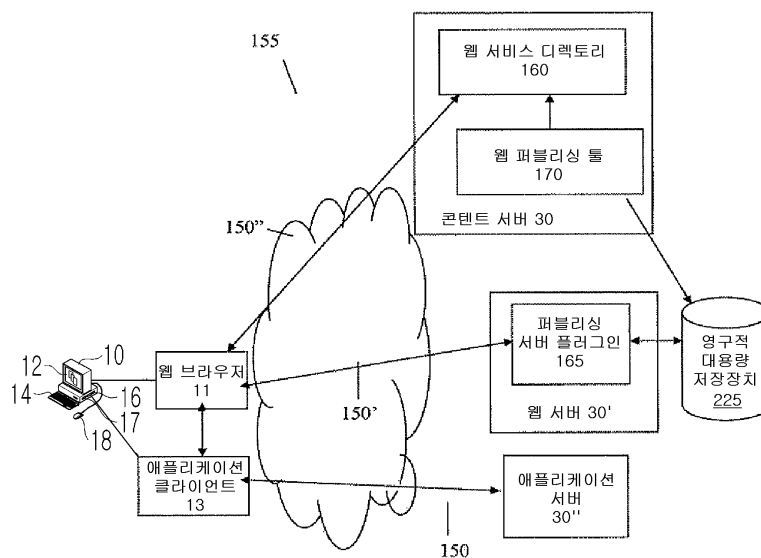
도면1d



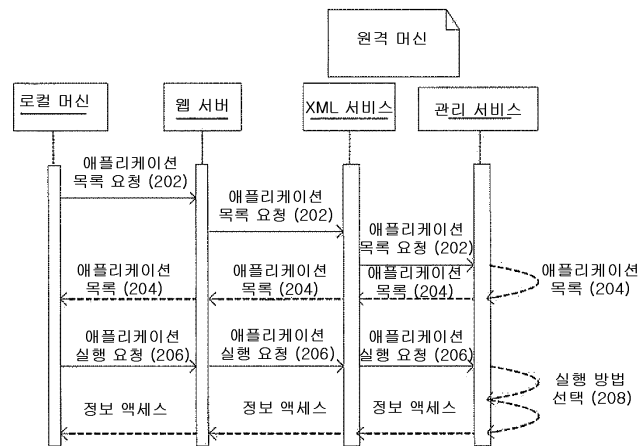
도면1e



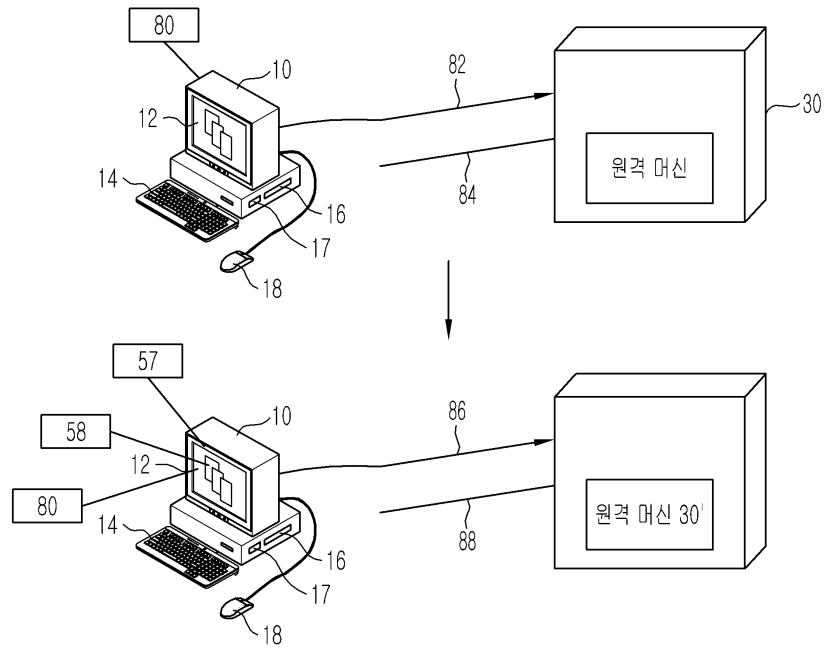
도면1f



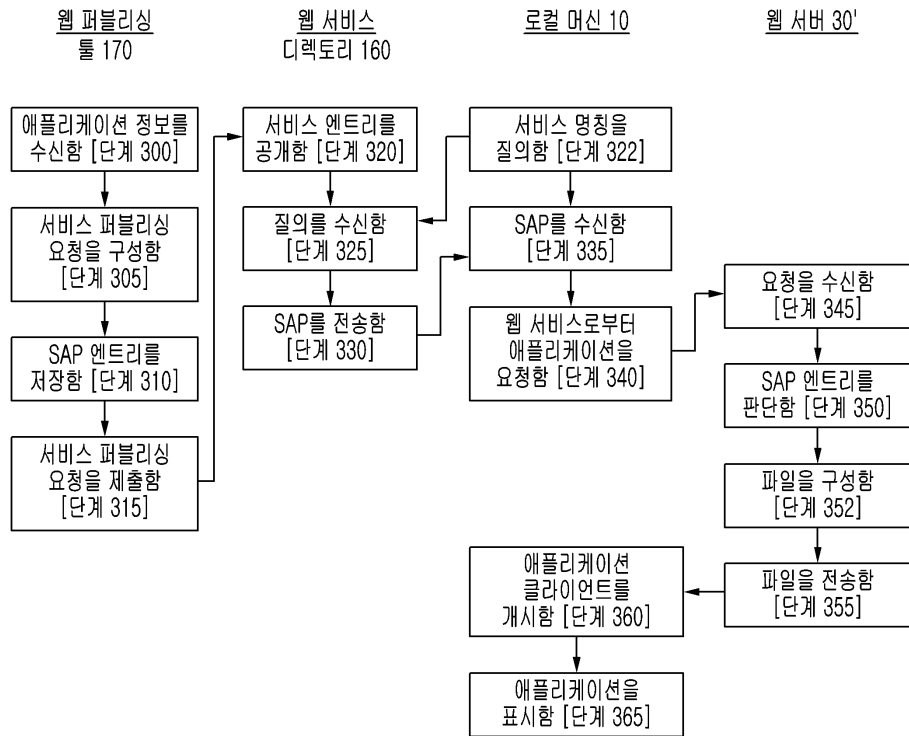
도면2



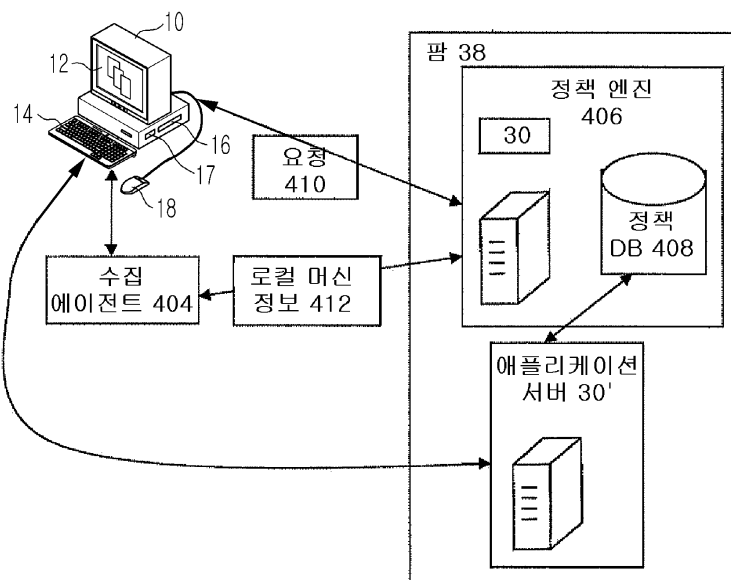
도면3a



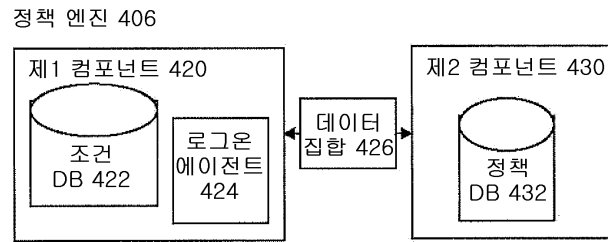
도면3b



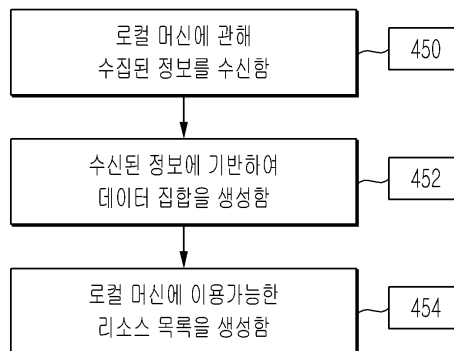
도면4a



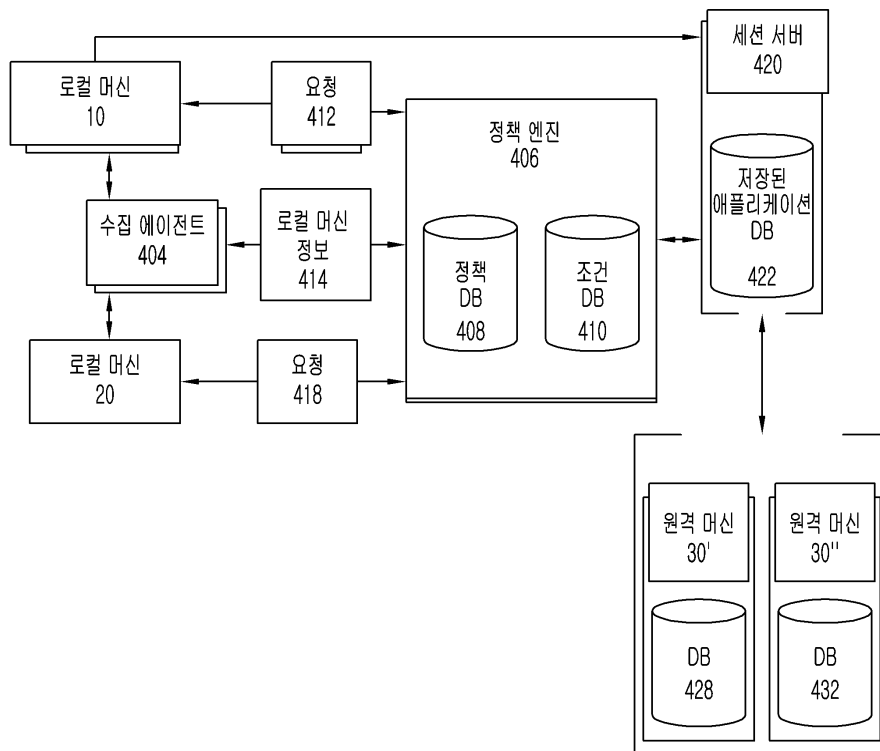
도면4b



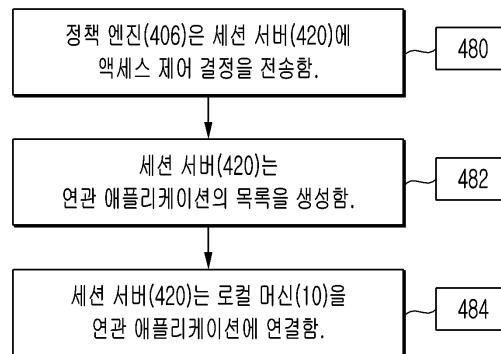
도면4c



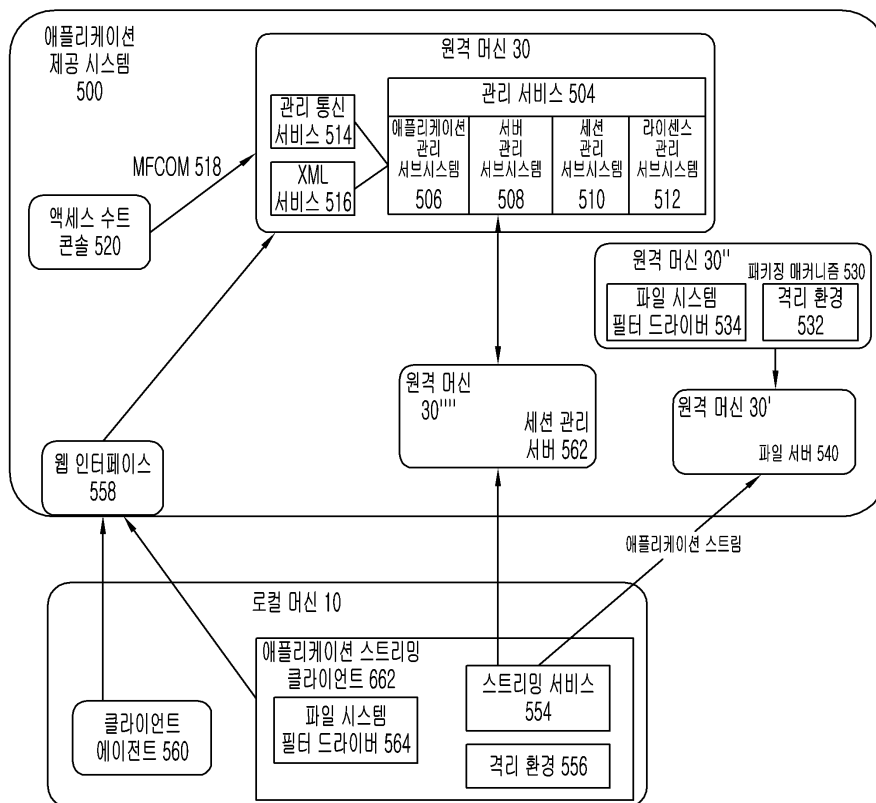
도면4d



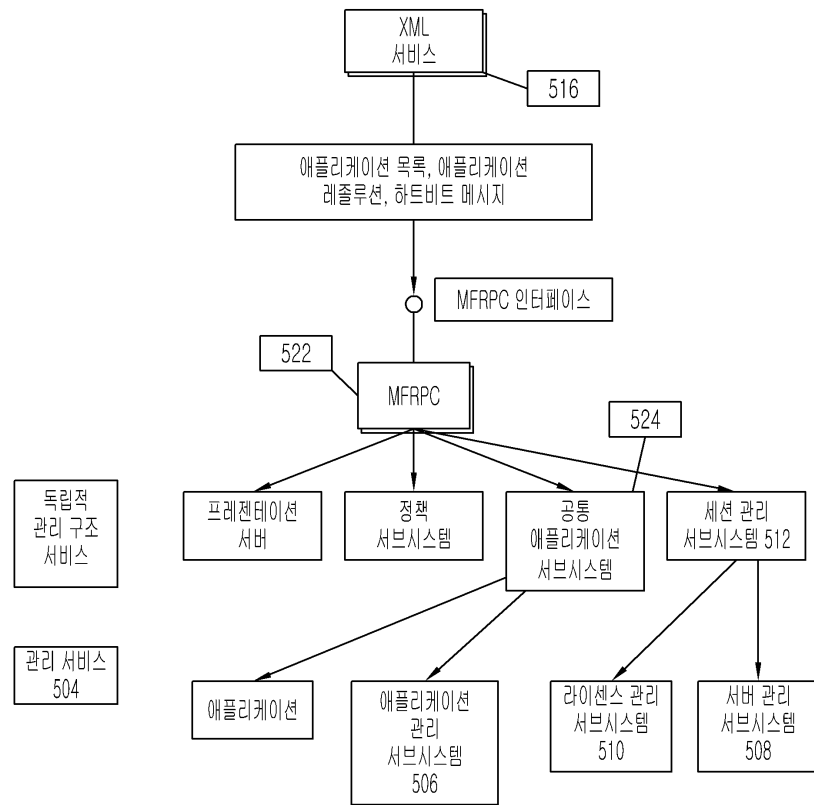
도면4e



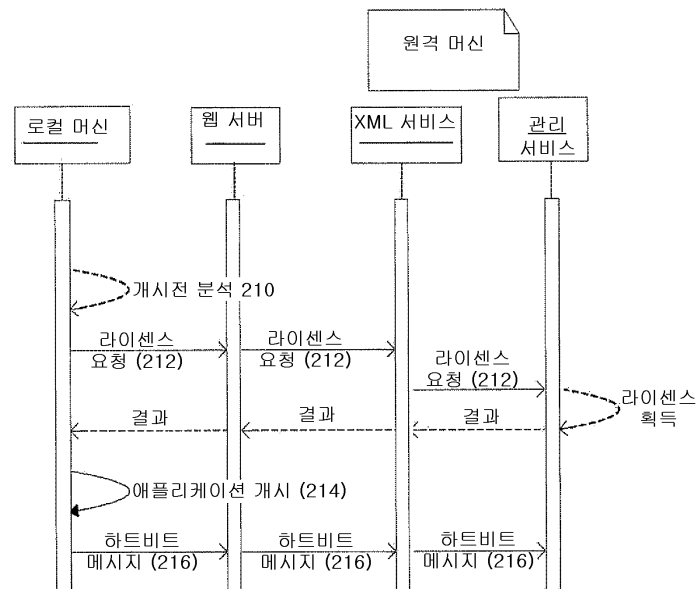
도면5



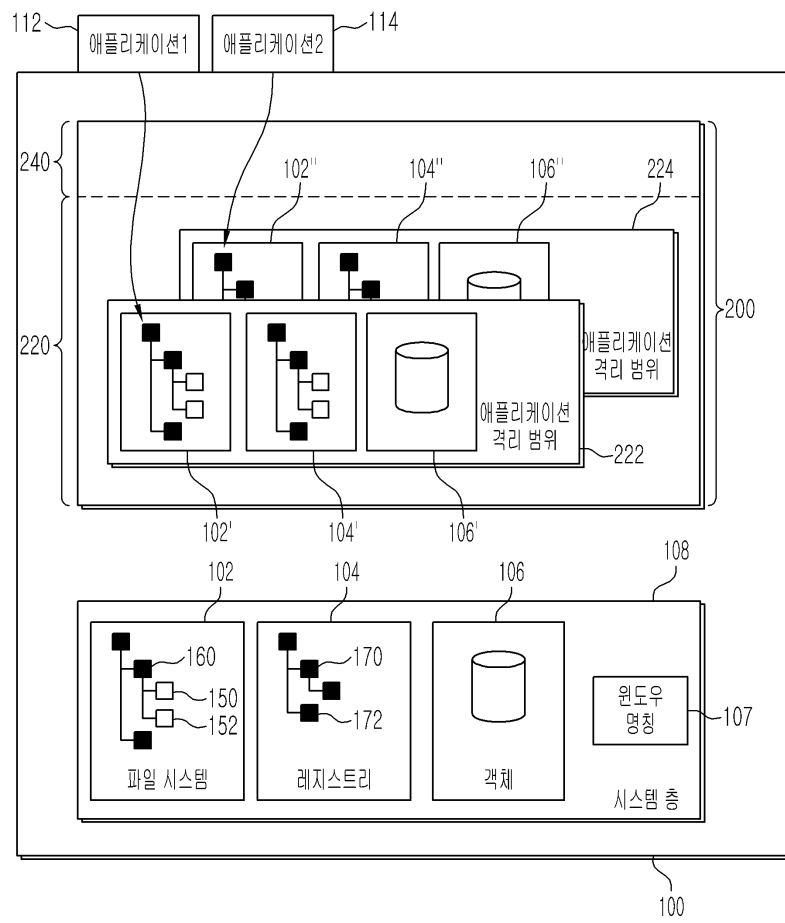
도면6



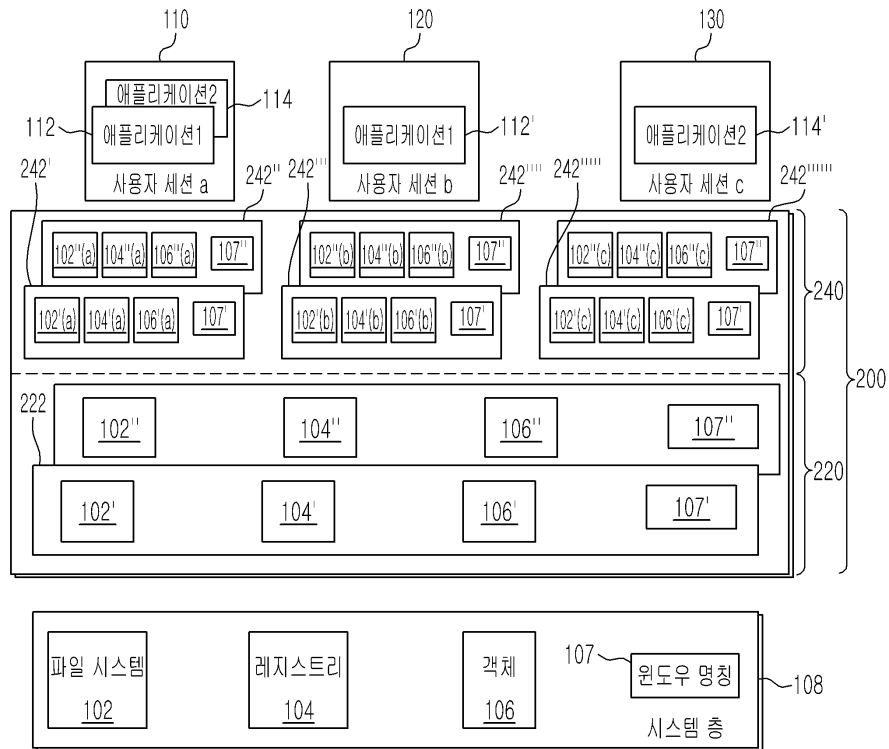
도면7



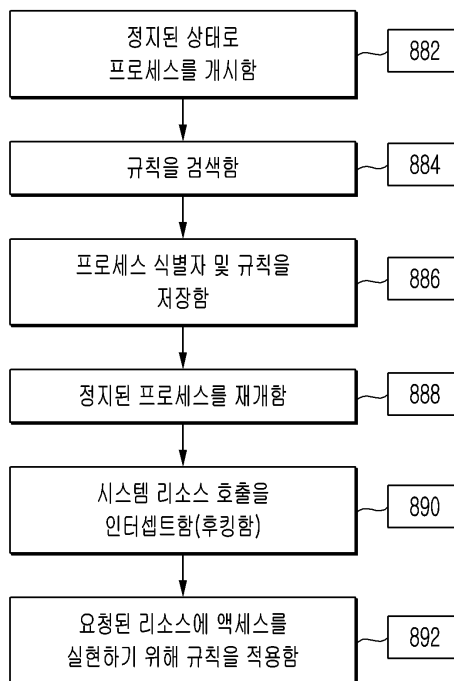
도면8a



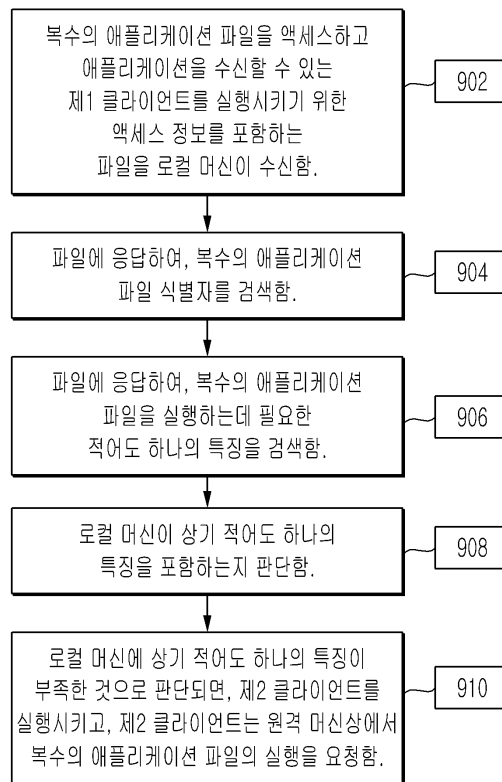
도면8b



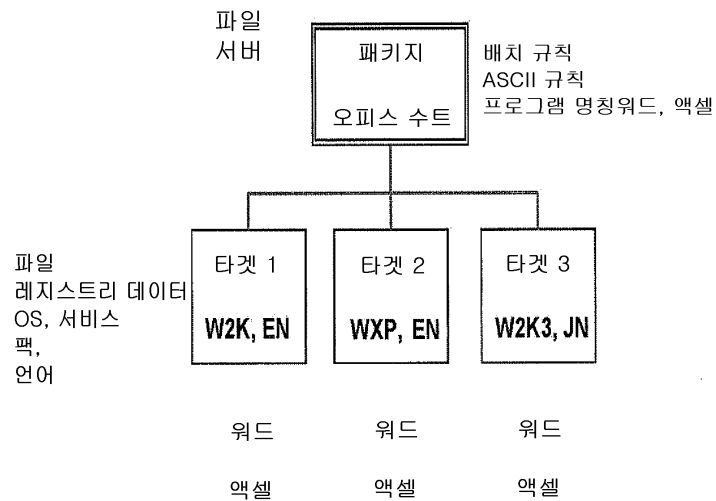
도면8c



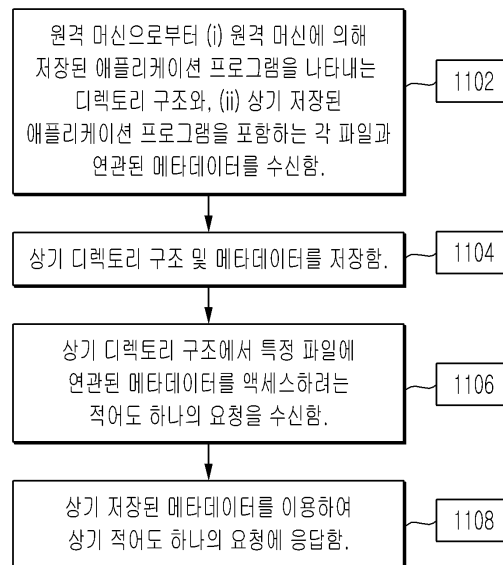
도면9



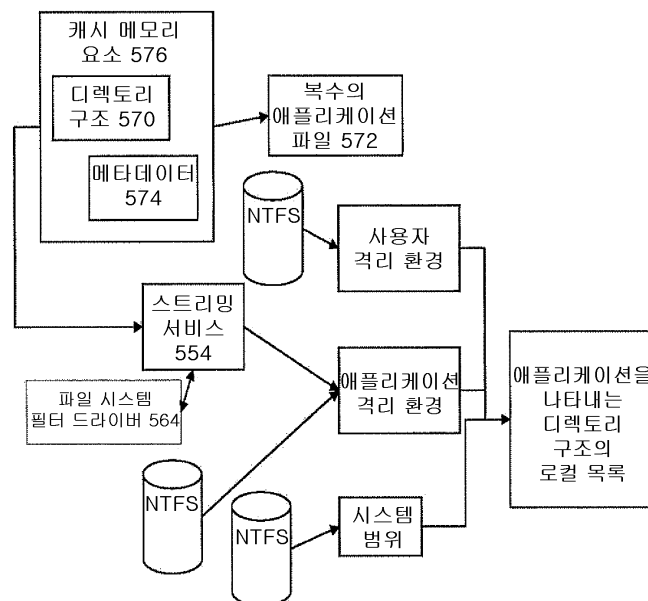
도면10



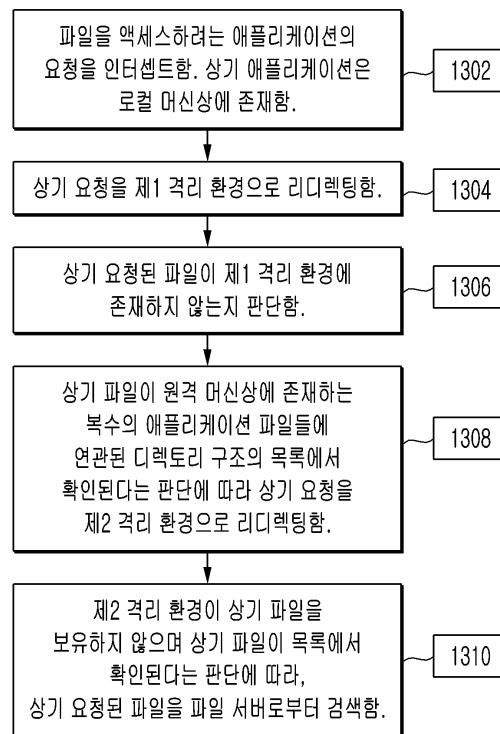
도면11



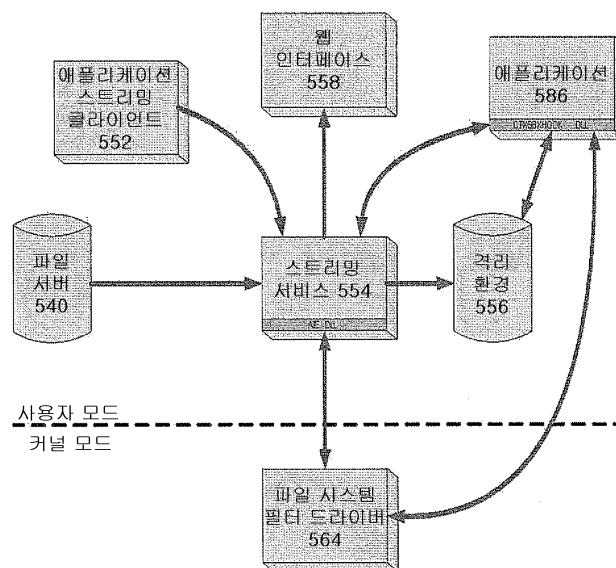
도면12



도면13

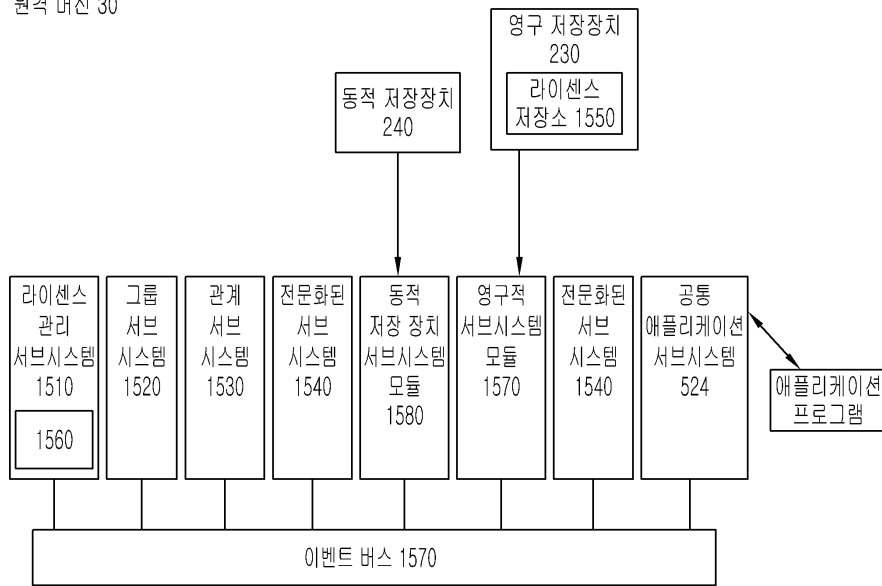


도면14

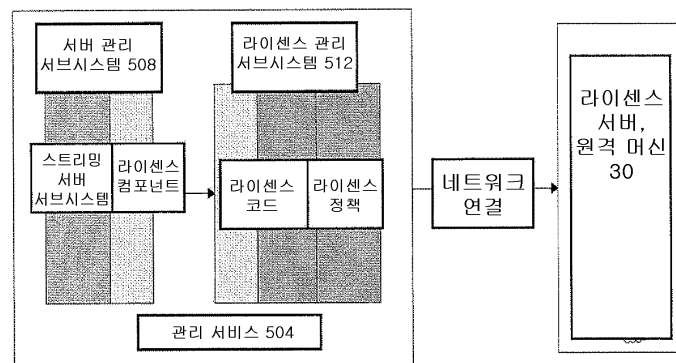


도면15

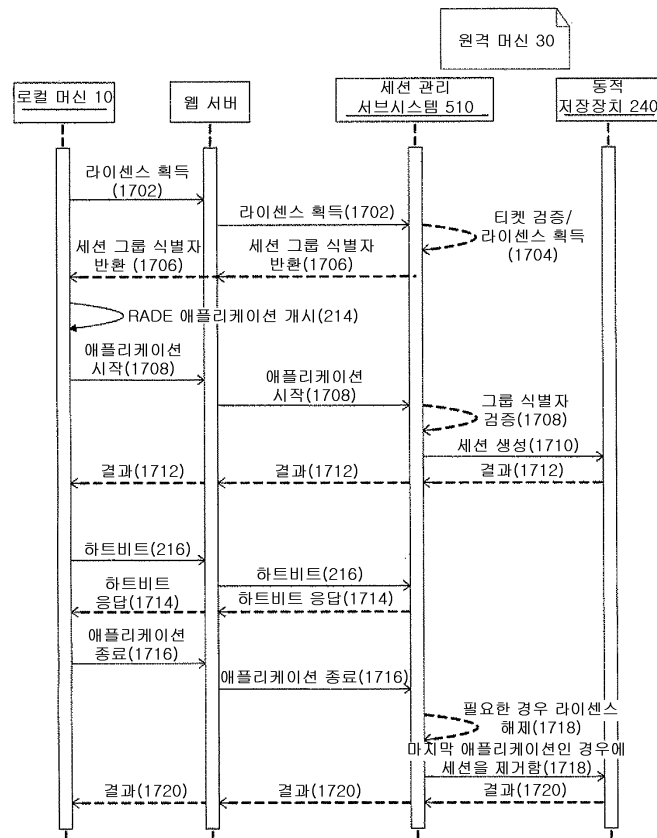
원격 머신 30



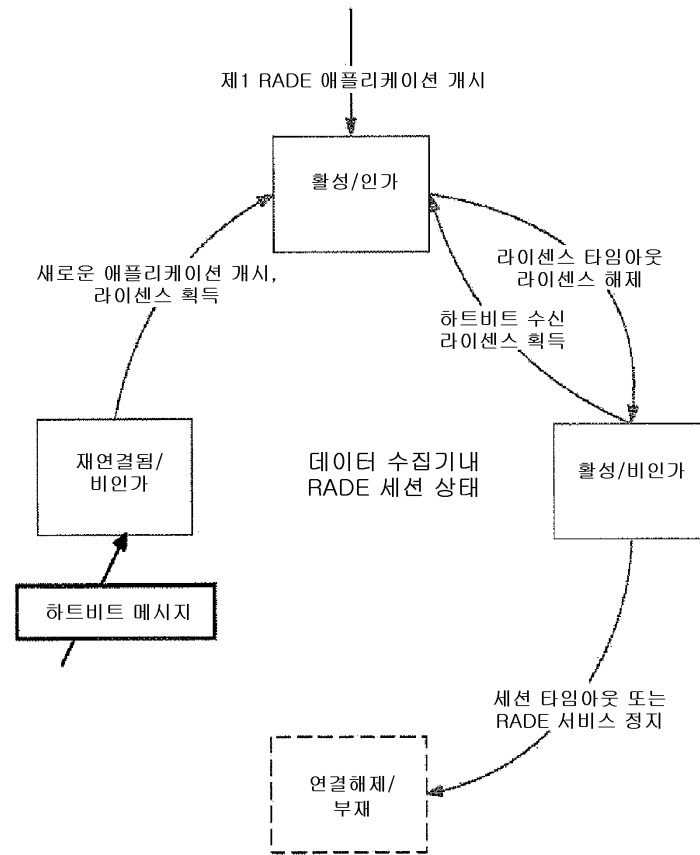
도면16



도면17



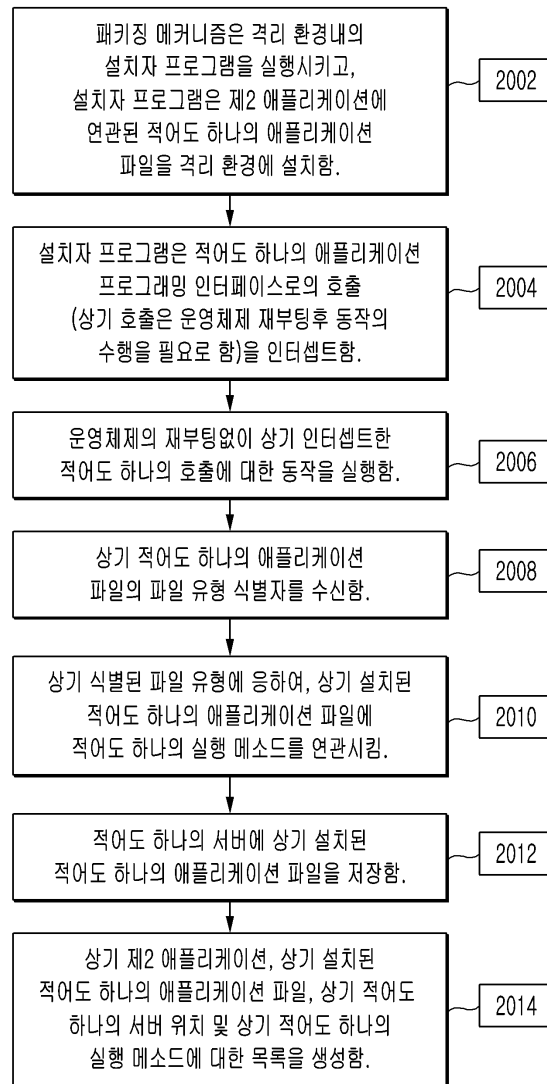
도면18



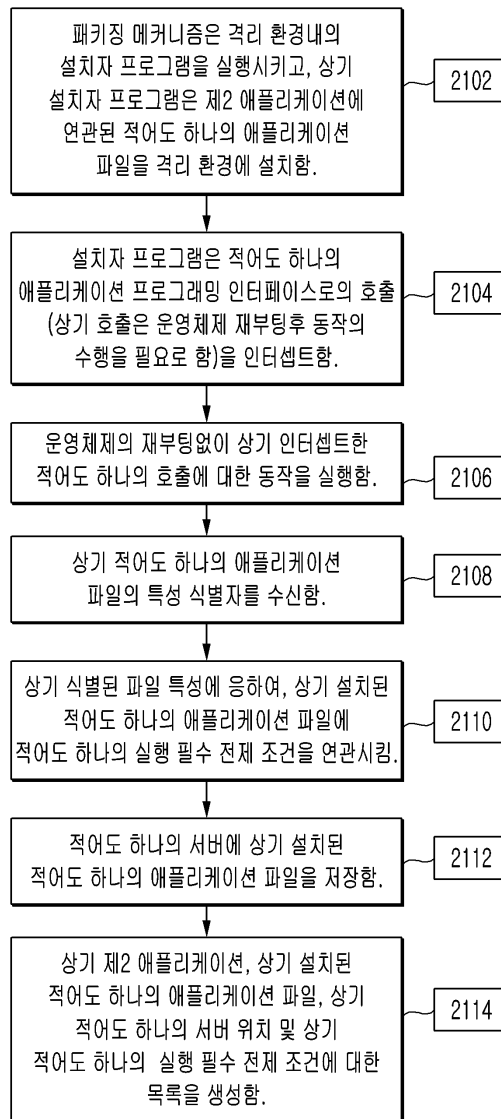
도면19

패키지	
타겟 1	
애플리케이션	Foo
타겟 OS:	W2K PRO (All Service Packs) WXP PRO (All Service Packs) W2K3 (All Service Packs)
타겟 언어:	English
마스터 드라이브:	C:\
타겟 2	
애플리케이션	Foo
타겟 OS:	W2K PRO (All Service Packs) WXP PRO (All Service Packs) W2K3 (All Service Packs)
타겟 언어:	German
마스터 드라이브:	C:\

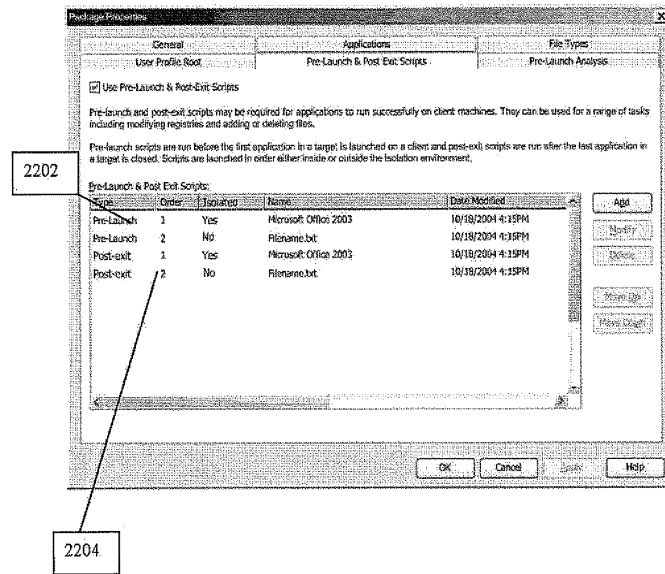
도면20



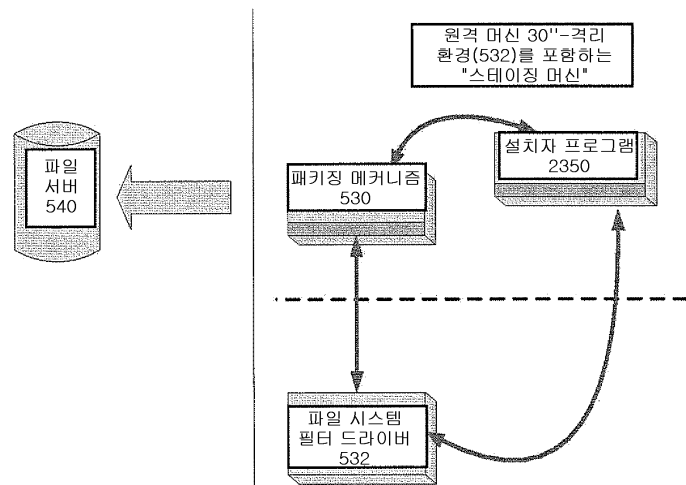
도면21



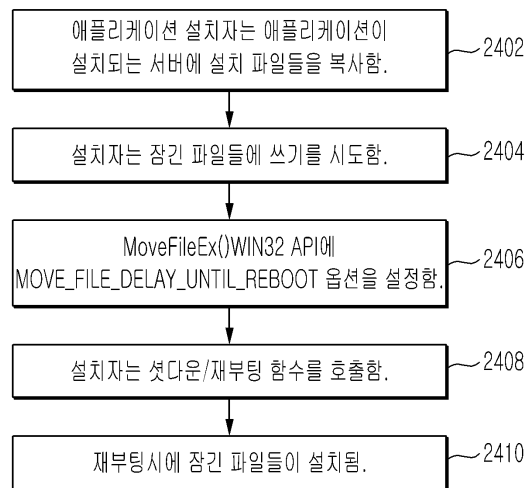
도면22



도면23

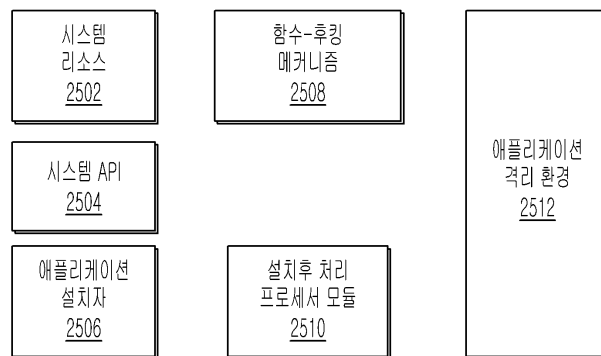


도면24

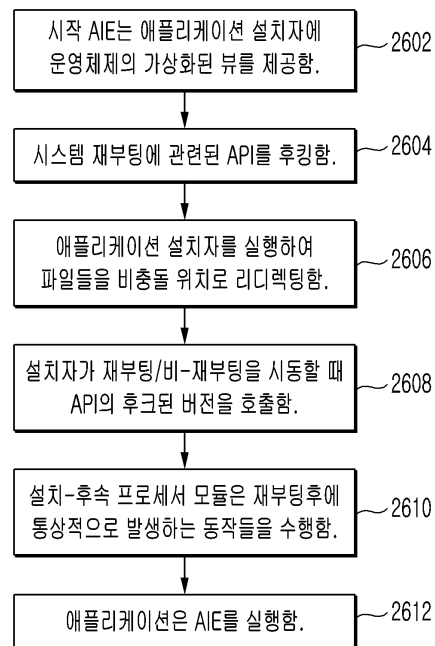


도면25

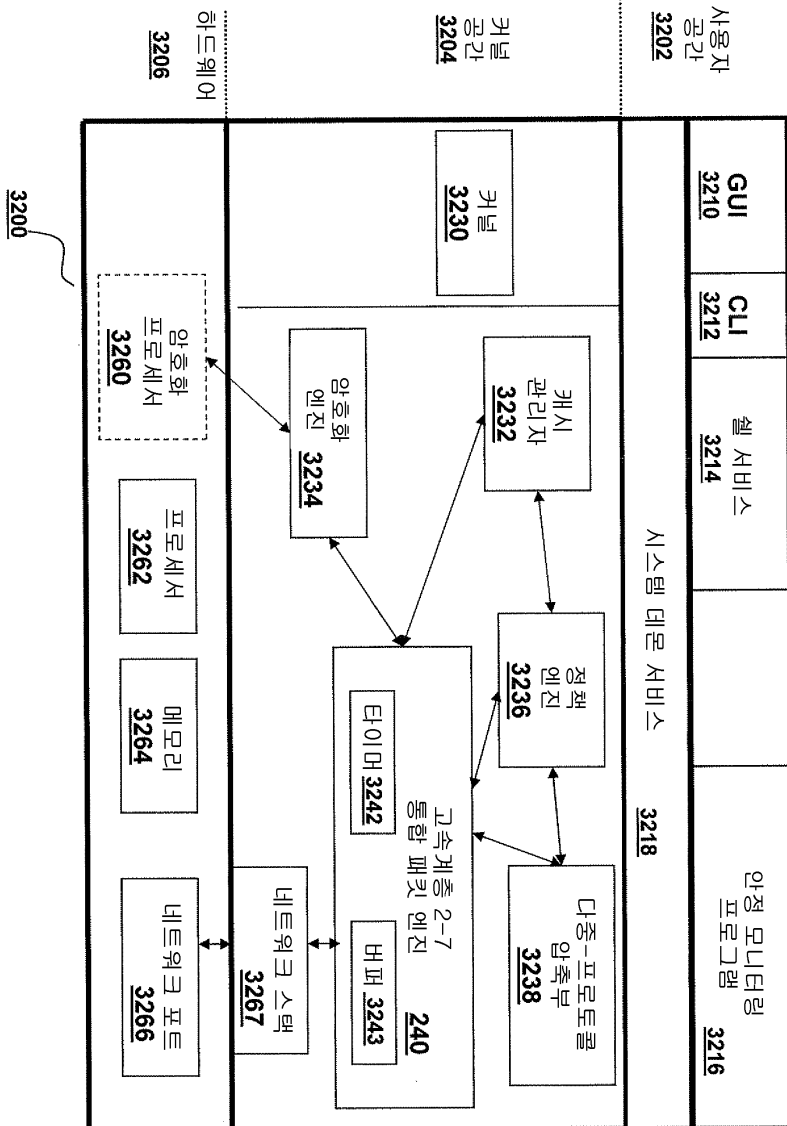
원격 머신 30



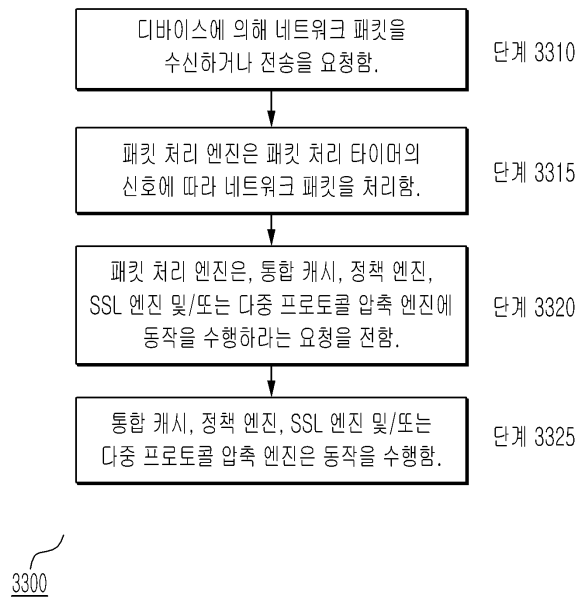
도면26



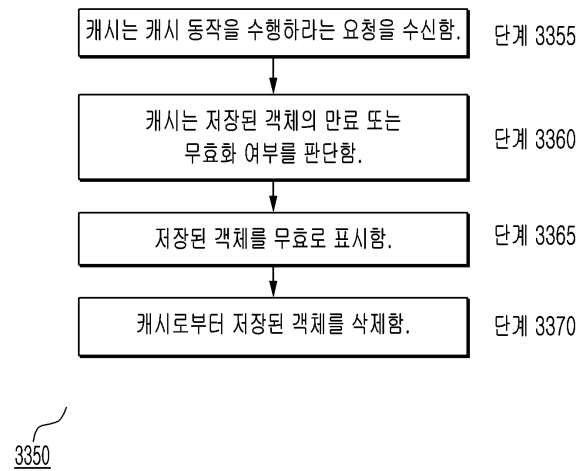
도면27



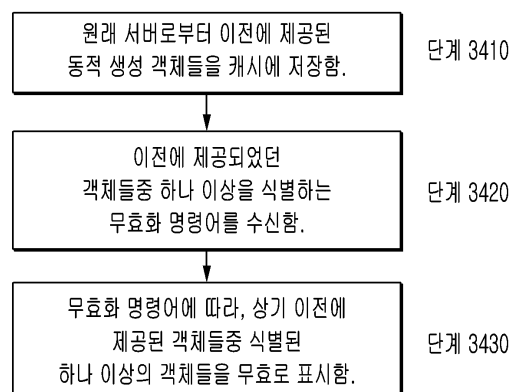
도면28a



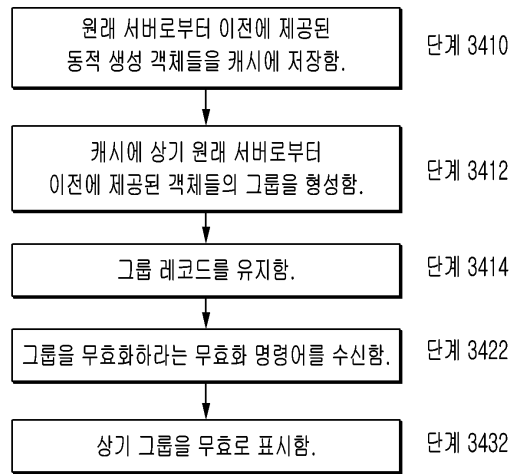
도면28b



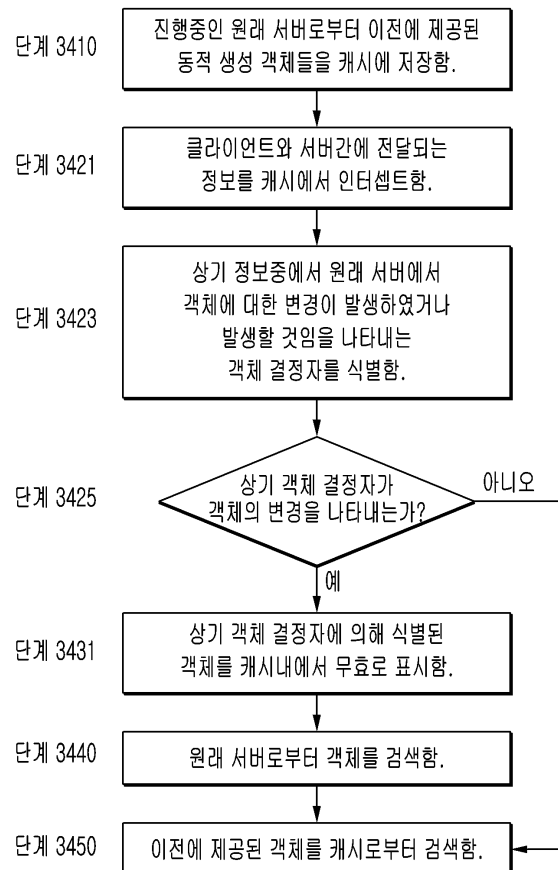
도면29a



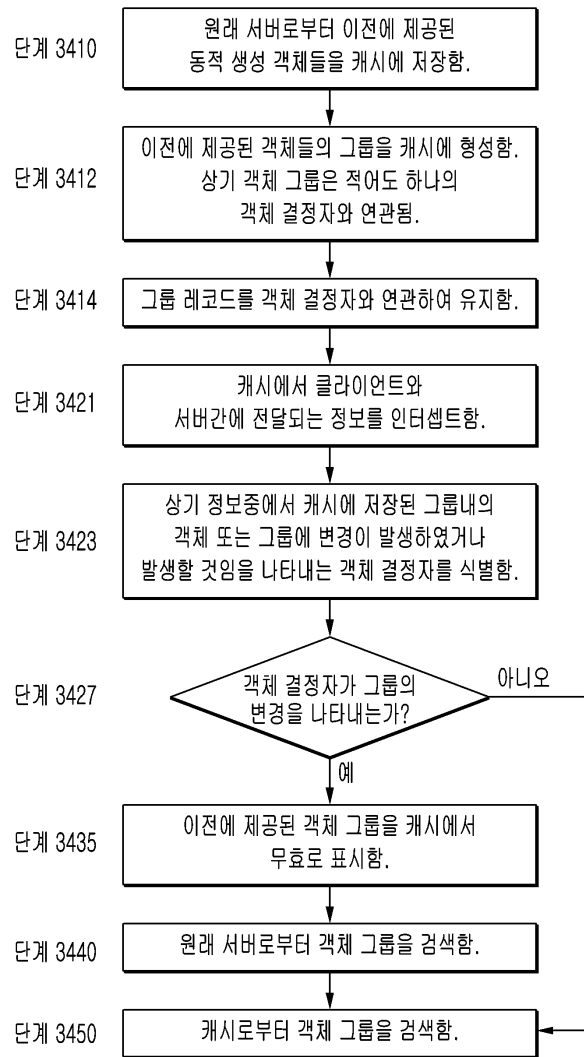
도면29b



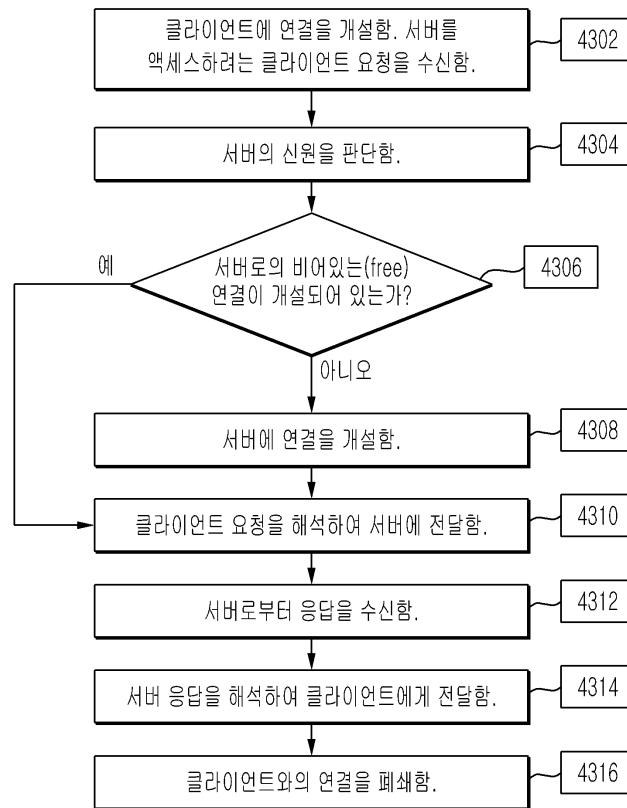
도면29c



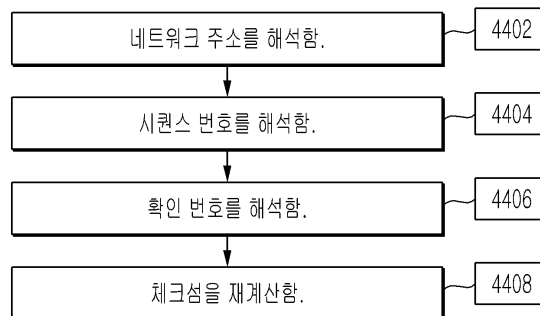
도면29d



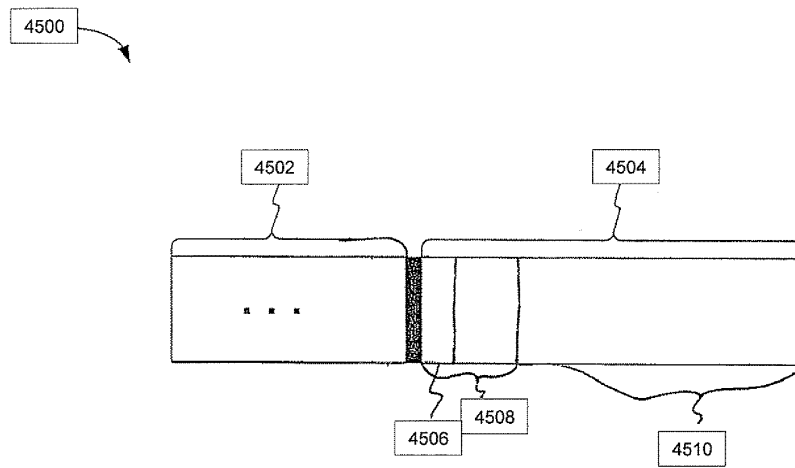
도면30



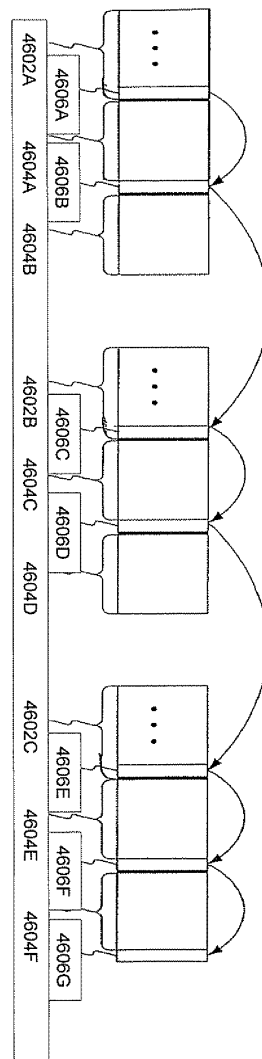
도면31



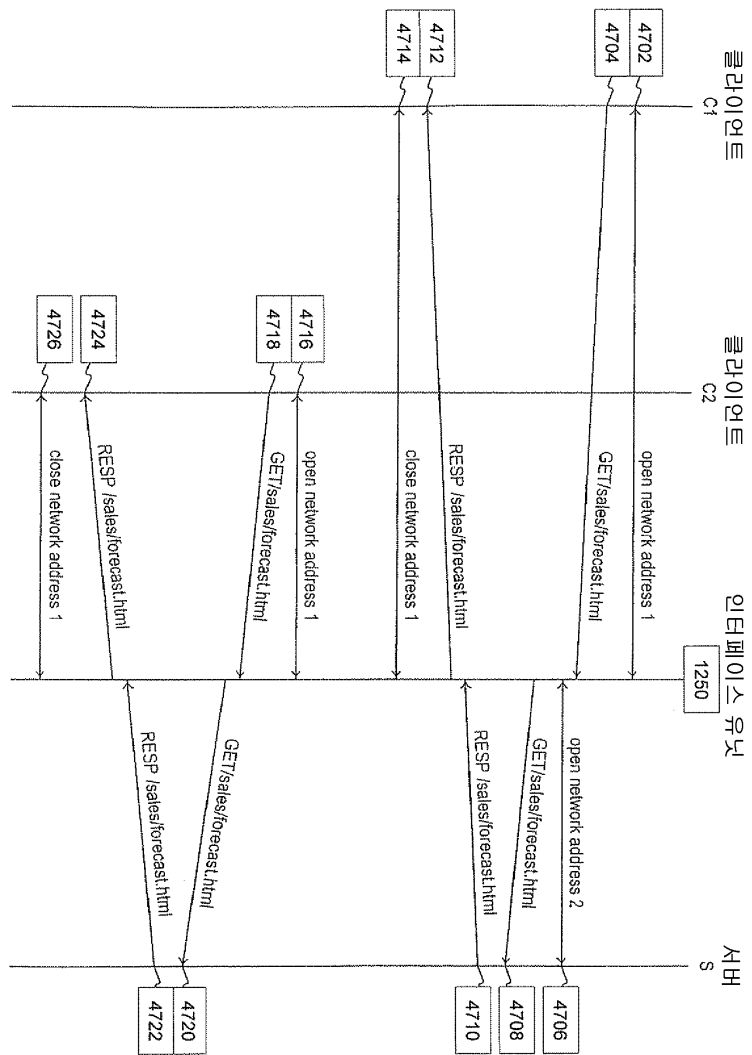
도면32



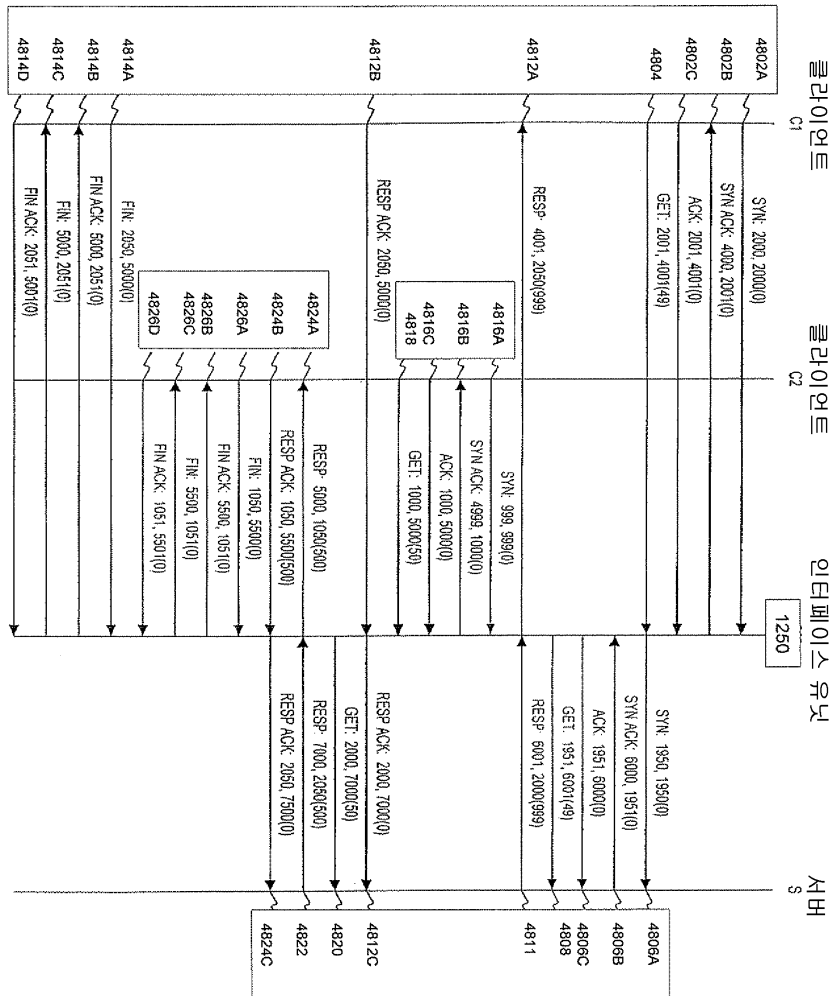
도면33



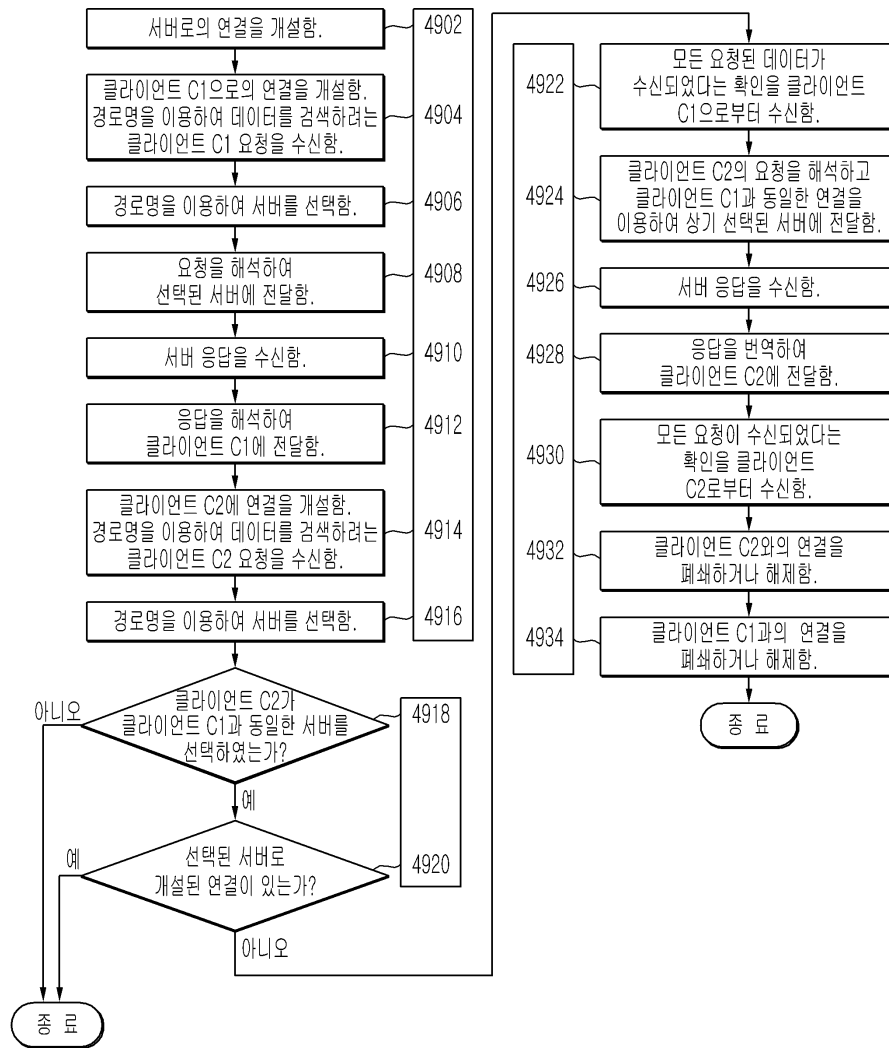
도면34



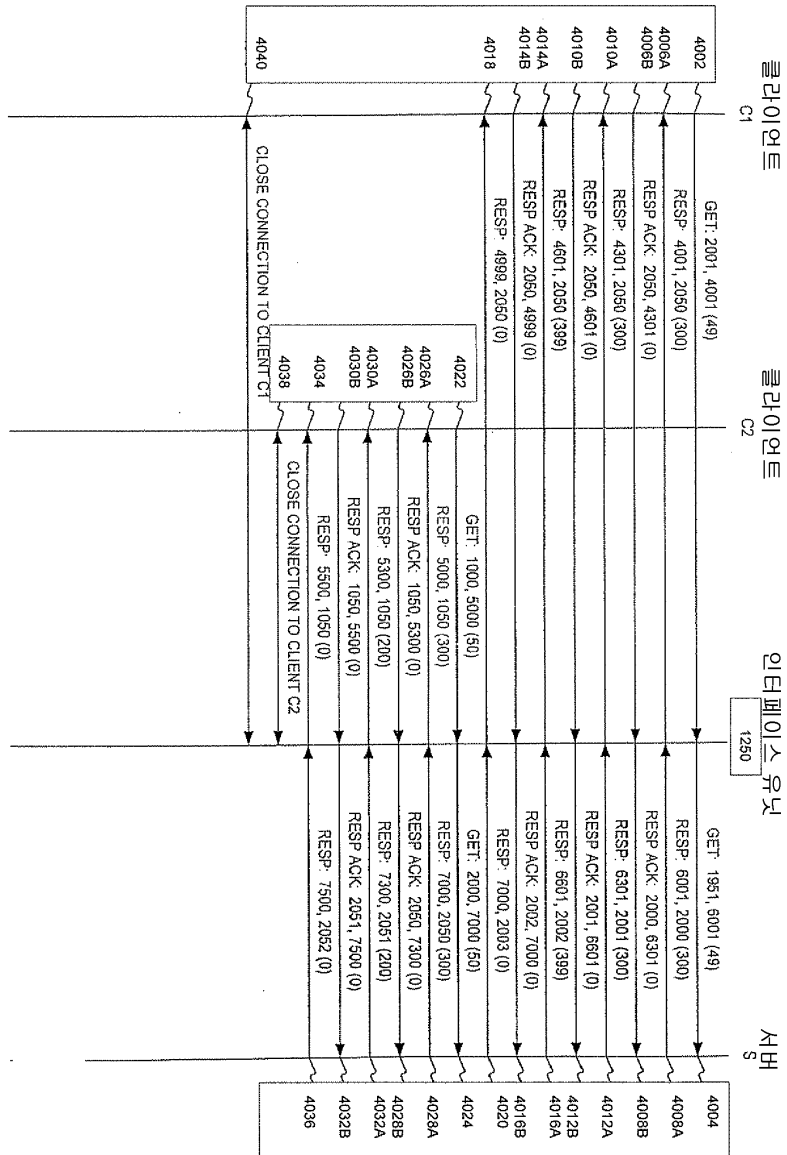
도면35



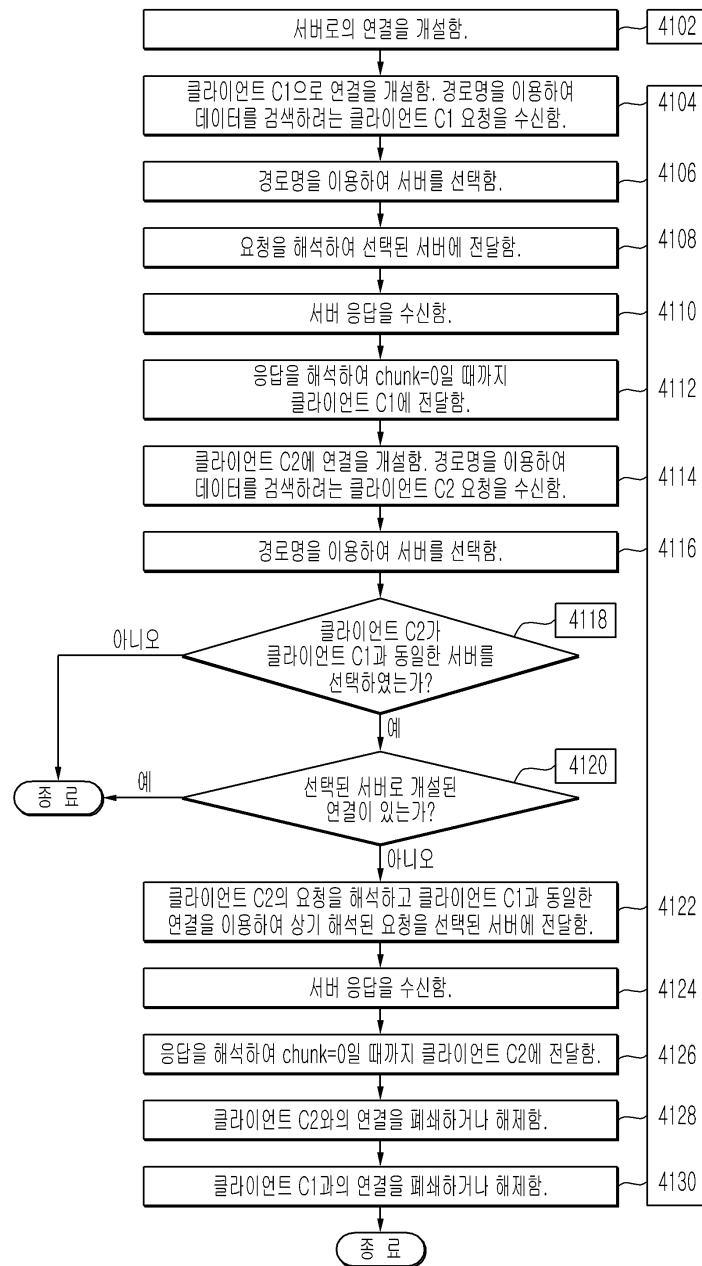
도면36



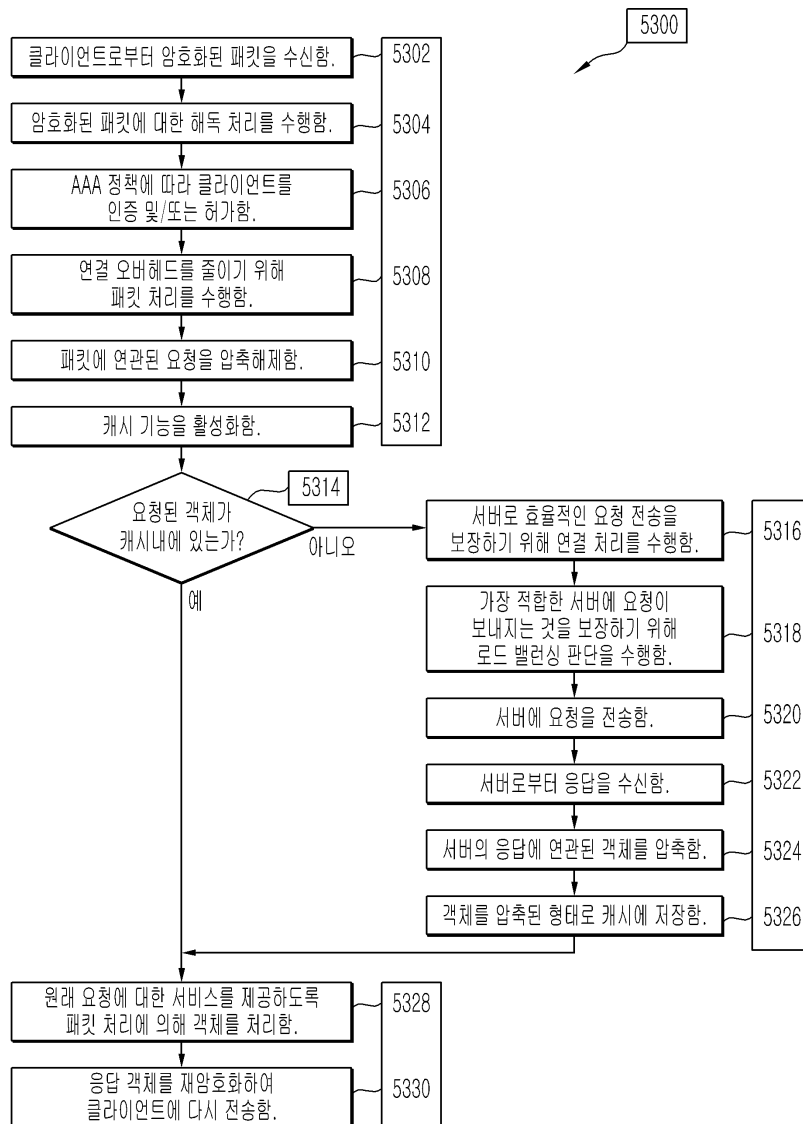
도면37



도면38

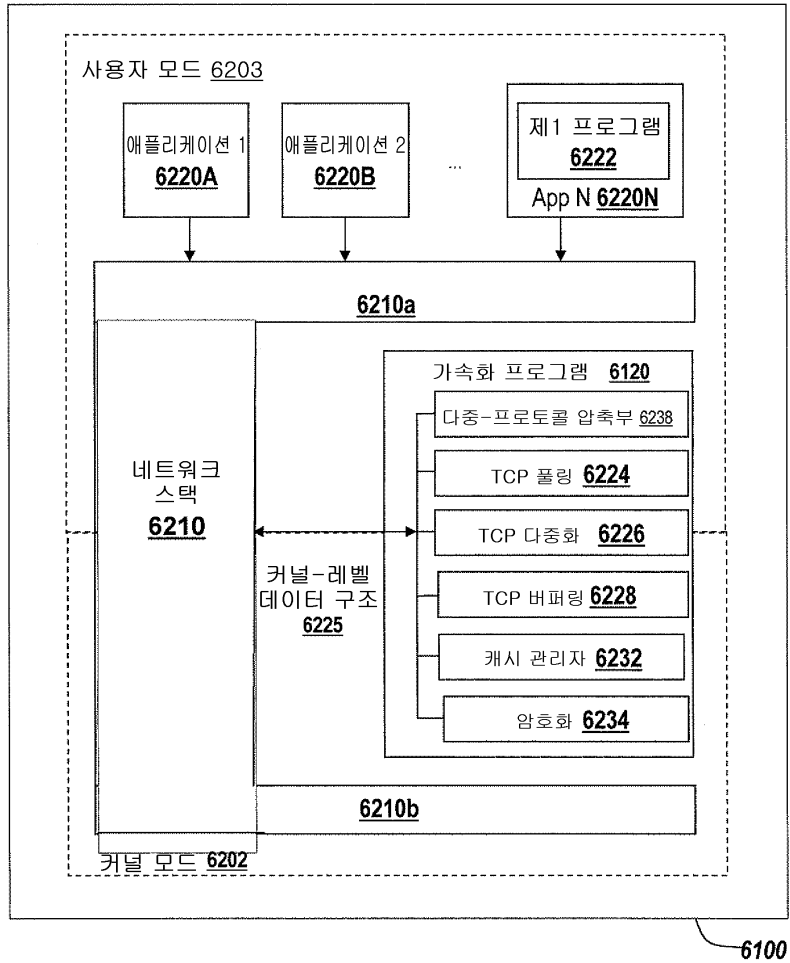


도면39

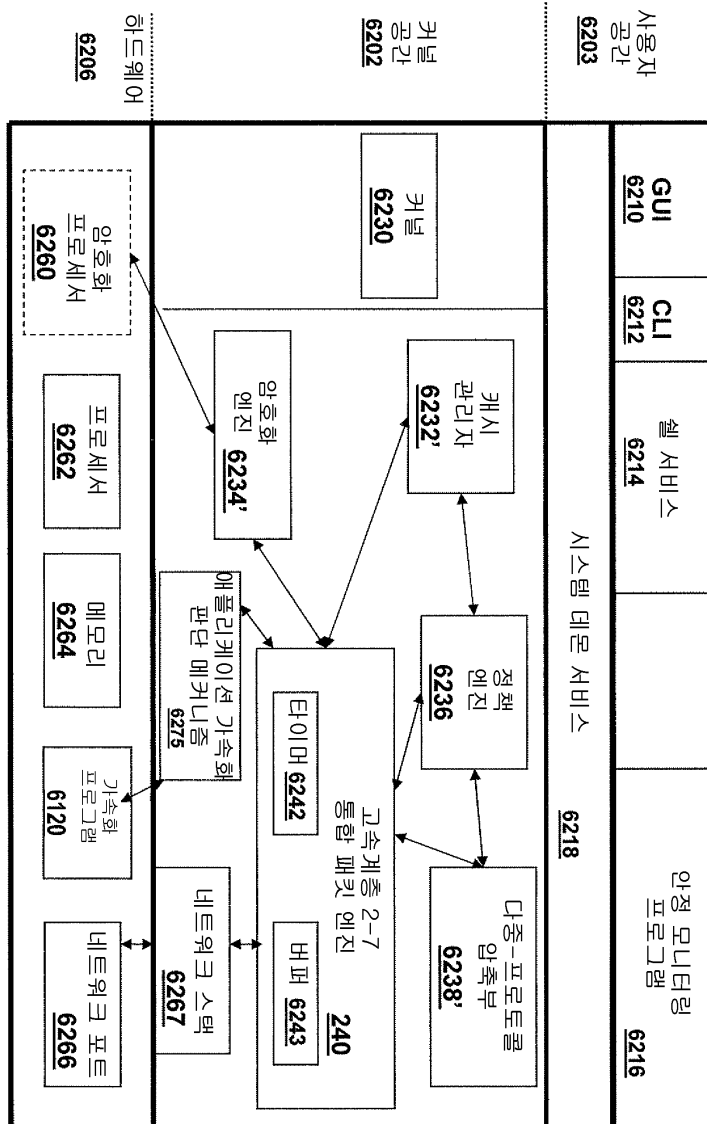


도면40a

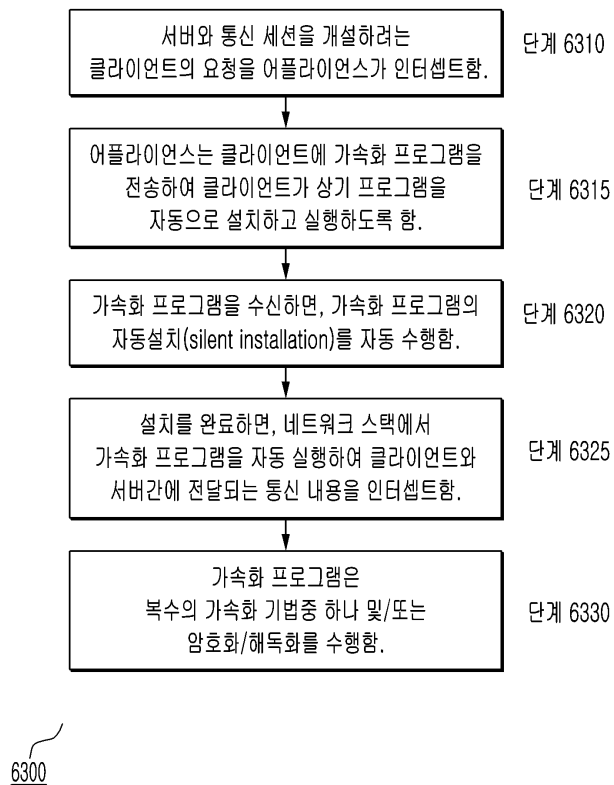
클라이언트 6205



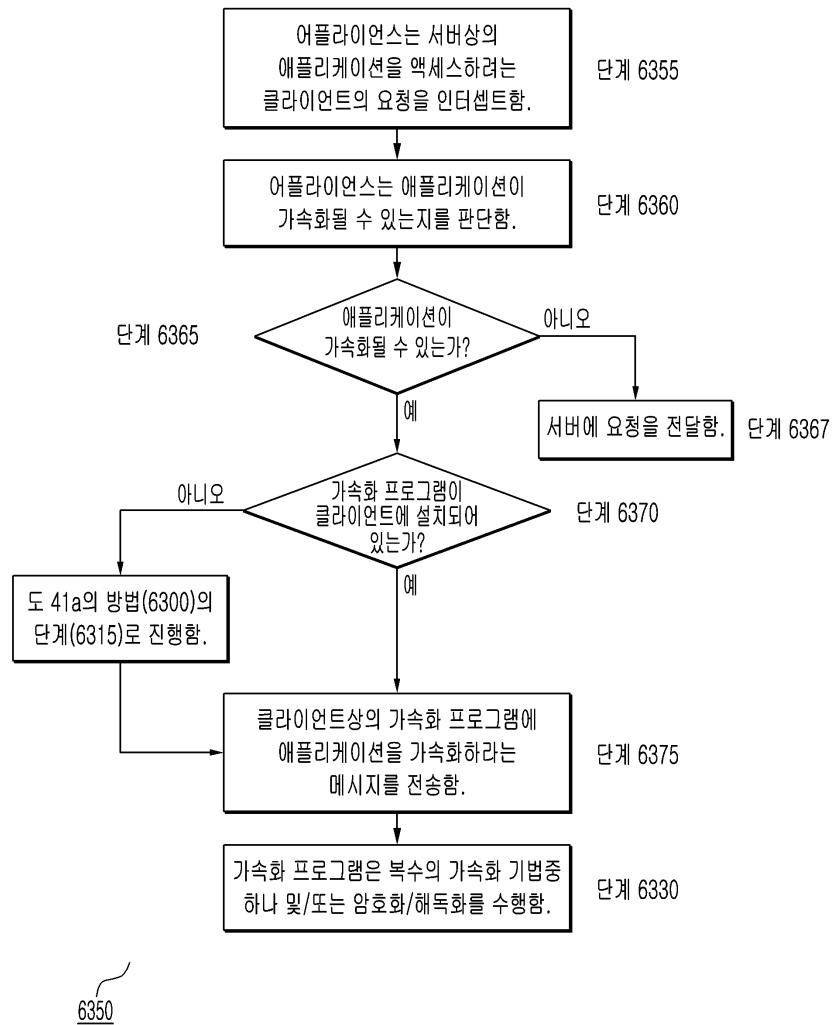
도면40b



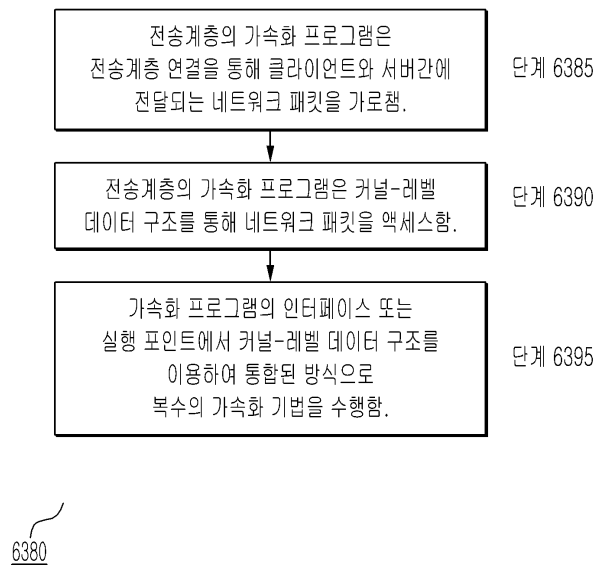
도면41a



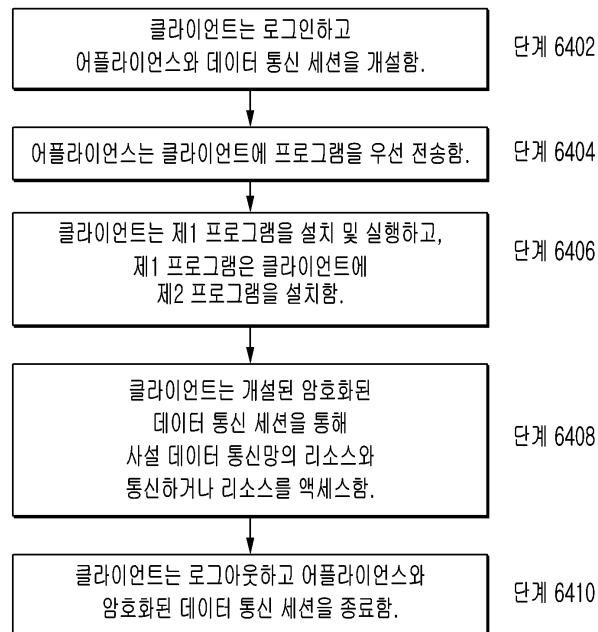
도면41b



도면41c

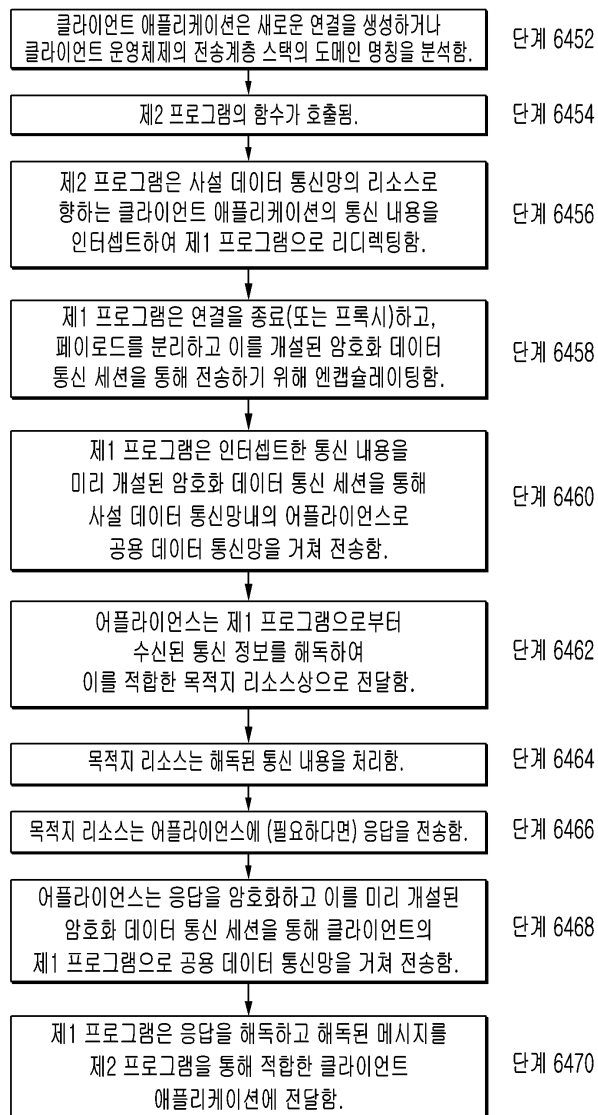


도면42a



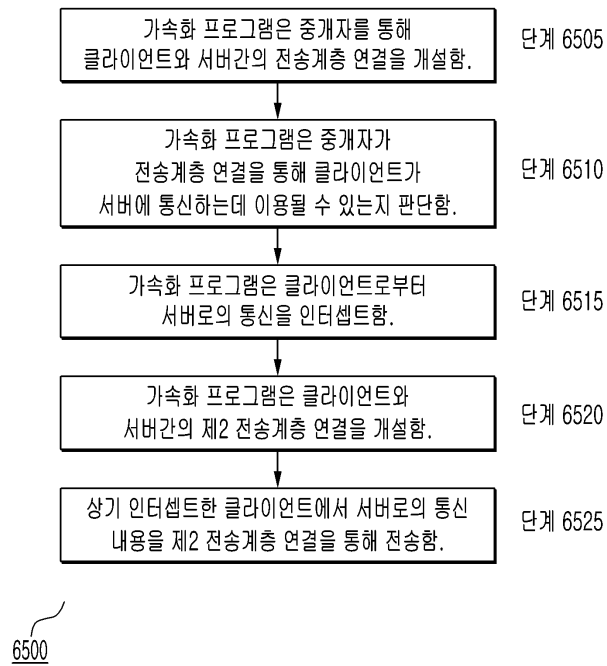
6400

도면42b

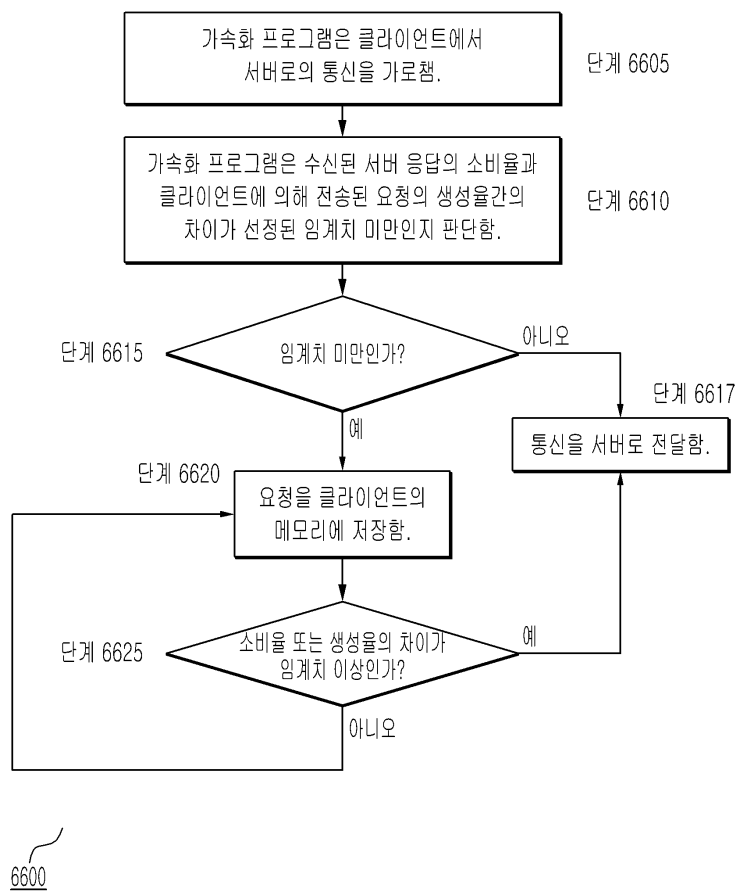


6450

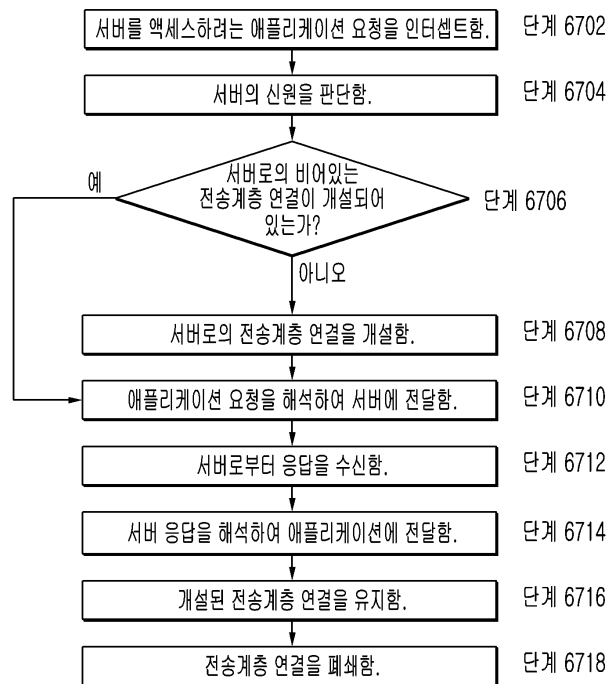
도면43



도면44

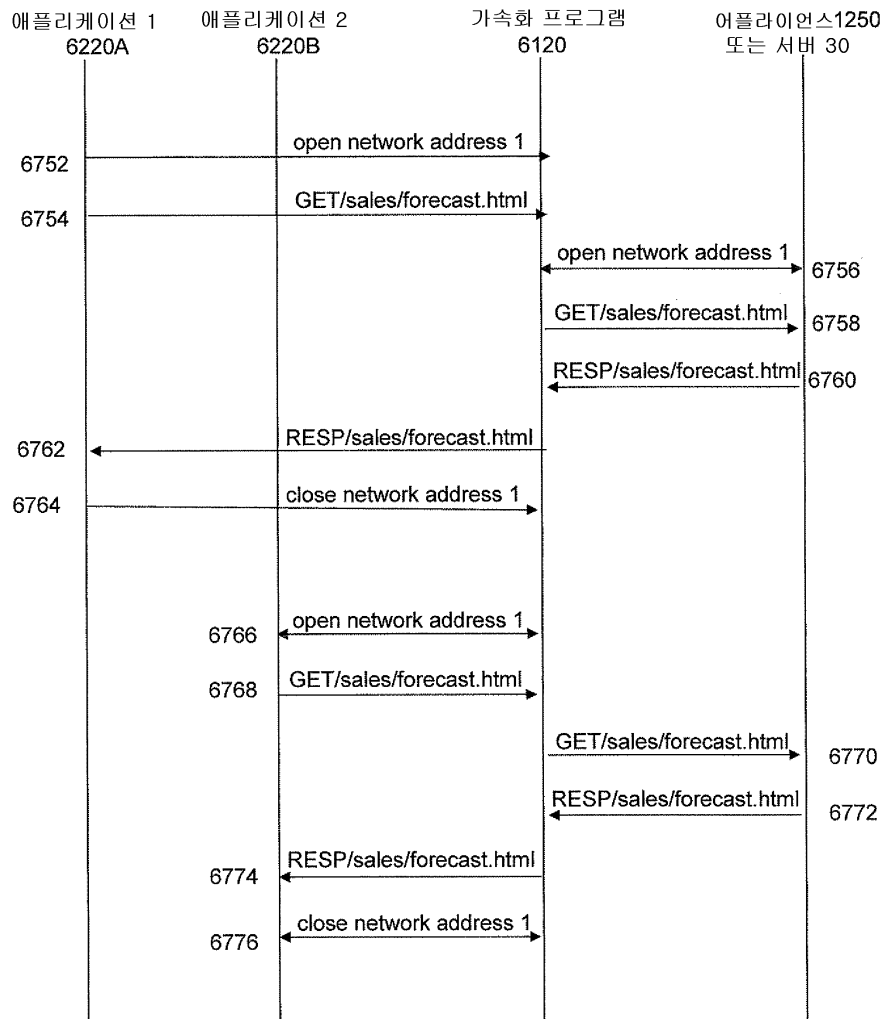


도면45a

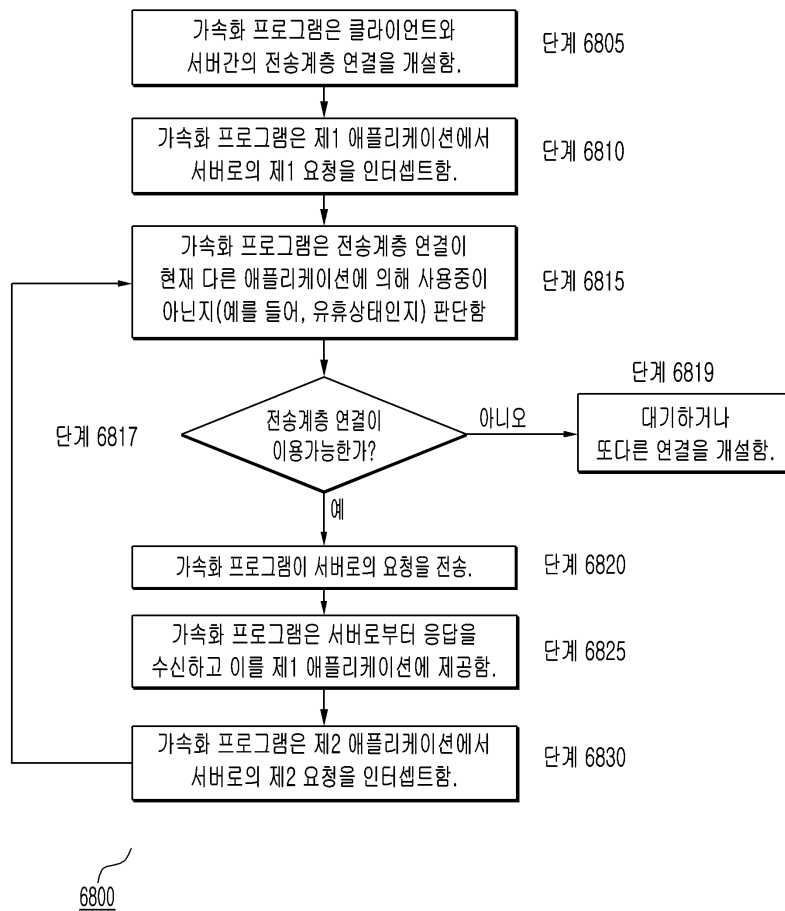


6700

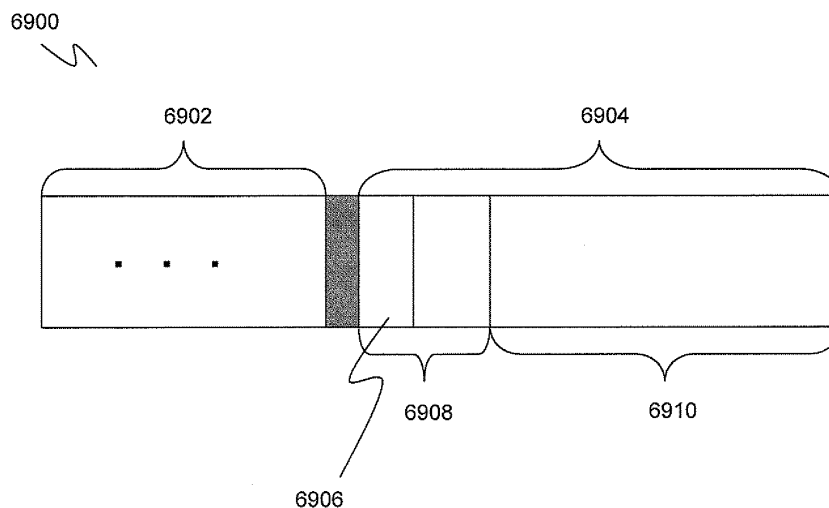
도면45b



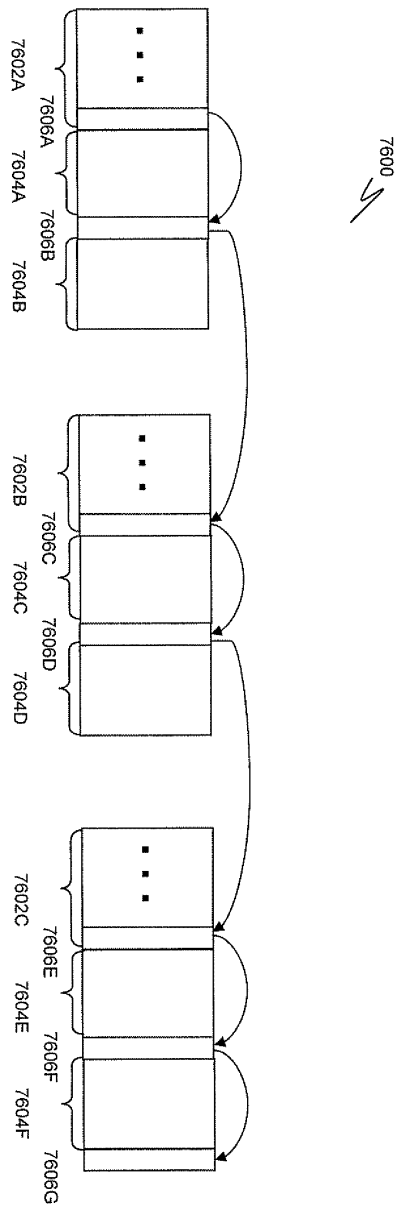
도면46



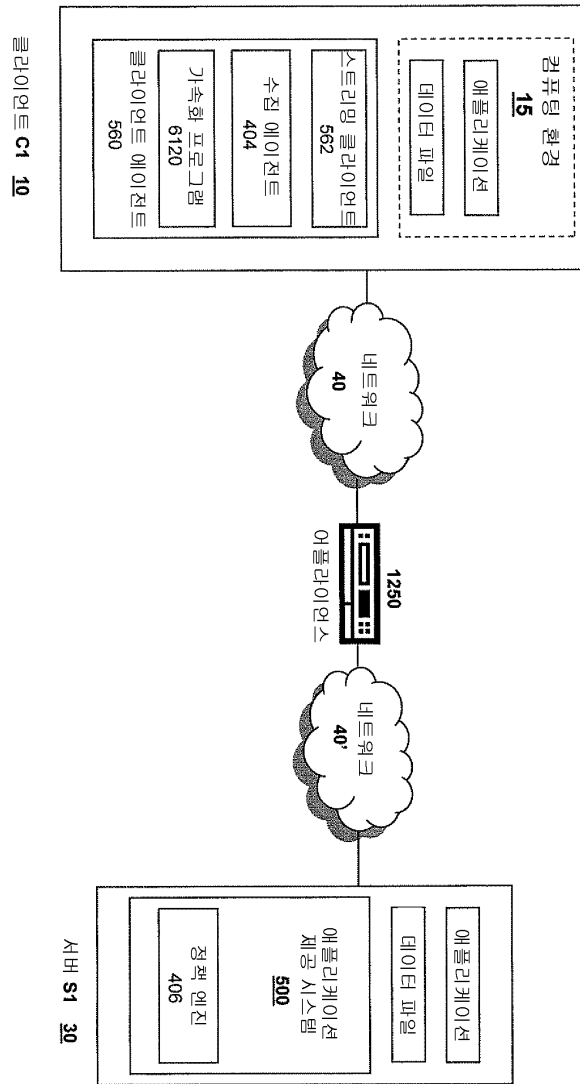
도면47



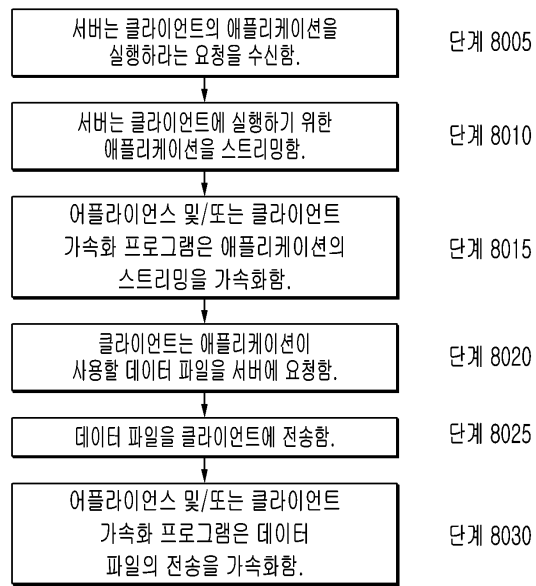
도면48



도면49a



도면49b



8000