(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
29 December 2010 (29.12.2010)

PCT

(10) International Publication Number
WO 2010/151496 A1

(51) International Patent Classification:
H04L 12/56 (2006.01)

(21) International Application Number:
PCT/US2010/039213

(22) International Filing Date:
18 June 2010 (18.06.2010)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
61/219,264 22 June 2009 (22.06.2009) US

(71) Applicant *(for all designated States except US):* **CITRIX** SYSTEMS, **INC.** [US/US]; 851 West Cypress Creek Road, Fort Lauderdale, FL 33309 (US).

(72) Inventors; and
(75) Inventors/Applicants *(for US only):* **AVDANIN, Roman** [US/US]; c/o CITRIX SYSTEMS, INC., 4988 Great America Parkway, Santa Clara, CA 95054 (US). BOTS, **Henk** [US/US]; c/o CURIX SYSTEMS, INC., 4988 Great America Parkway, Sabta Clara, CA 95054 (US). TALLA, Ramanjaneyulu, Y. [IN/IN]; c/o CITRIX SYSTEMS, INC., 4988 Great America Parkway, Santa Clara, CA 95054 (US). CHAUHAN, **Abhishek** [US/US]; c/o CITRIX SYSTEMS, INC., 4988 Great America Parkway, Santa Clara, CA 95054 (US). MIRANI, **Rajiv** [IN/US]; 27 Farrwood Dirive, Andover, MD 01810 (US).

(74) Agent: MISIC, **Mead;** Choate, Hall & Stewart LLP, Two International Place, Boston, MA 021 10 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available):* AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, **FI,** GB, GD, GE, GH, GM, GT, HN, HR, HU, **ID, IL,** IN, IS, **JP,** KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, **LT,** LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PE, PG, **PH,** PL, PT, RO, RS, RU, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, **TH, TJ,** TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available):* ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, **FI,** FR, GB, GR, HR, HU, IE, IS, IT, **LT,** LU, LV, MC, MK, MT, NL, NO, **PL,** PT, RO, SE, SI, SK, SM, TR), OAPI (BF, **BJ,** CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, **TD,** TG).

[Continued on next page]

(54) Title: SYSTEMS AND METHODS FOR PLATFORM RATE LIMITING

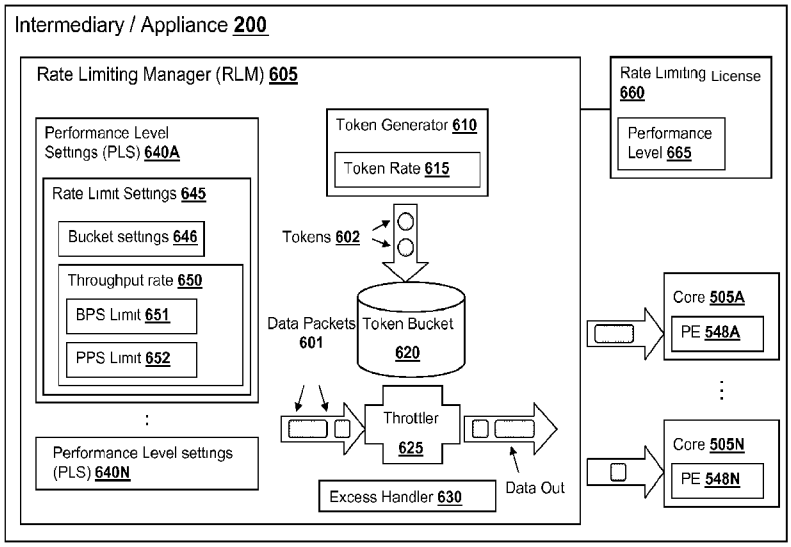

FIG. 6A

(57) Abstract: The present disclosure presents systems and methods for controlling network traffic traversing an intermediary device based on a license or a permit granted for the intermediary device. The systems and methods control a rate of a traffic of a device in accordance with a rate limit identified by a rate limiting license. A rate limiting manager of an intermediary device that processes network traffic between a plurality of clients and a plurality of servers, may identify presence of a rate limiting license that further identifies a performance level. The rate limiting manager may establish a rate limit based on the performance level of the rate limiting license. A throttler of the intermediary may control a rate of receiving network packets in accordance with the rate limit.

WO 2010/151496 A1

Published:

— *with international search report (Art. 21(3))*

SYSTEMS AND METHODS FOR PLATFORM RATE LIMITING

Related Applications

This present application claims priority to a U.S. Provisional Application No. 61/219,264, entitled "Systems and Methods for Platform Rate Limiting", filed on June 22, 2009, which is incorporated herein by reference in its entirety.

Field of the Invention

The present application generally relates to data communication networks. In particular, the present application relates to systems and methods for controlling a rate of a traffic according to a license.

Background of the Invention

An enterprise may provide a service to users accessing servers from client machines via intermediaries deployed by the enterprise between the clients and servers. The intermediaries may manage and control the network traffic to enhance the user experience. The enterprise may, for a variety of reasons, determine to control the flow of the network traffic that traverses the intermediaries. The enterprise may further determine to control the flow of the network traffic receiving the intermediaries.

Brief Summary of the Invention

The present application is directed towards systems and methods for controlling network traffic traversing an intermediary device based on a license or a permit granted for the intermediary device. The present application is also directed towards systems and methods for controlling a rate of a traffic being received by an intermediary device in accordance with a rate limit identified by a license or permit. By controlling the rate of the traffic being received by the intermediary, the rate at which the traffic is processed and the resources of the intermediary are utilized may also be controlled.

In some aspects, the present application is directed to a method for controlling a rate of a traffic of a device in accordance with a rate limit identified by a rate limiting license. A rate limiting manager of an intermediary device that processes network traffic between a plurality of clients and a plurality of servers, may identify presence of a rate limiting license

identifying a performance level. The rate limiting manager may establish a rate limit based on the performance level of the rate limiting license. A throttler of the intermediary may control a rate of receiving network packets in accordance with the rate limit.

The rate limiting manager may identify the rate limiting license is not present and establish a set of one or more rate limit parameters for the rate limit for a lower performance level. The lower performance level may include the throttler controlling the rate of receiving network packets at a slower rate than a rate for identified rate limiting licenses. In some embodiments, the rate limiting manager identifies a type of hardware platform of the intermediary device. The rate limiting manager establishes the rate limit based on the type of hardware platform and the performance level. In some embodiments, the rate limiting manager establishes a maximum size of a token bucket in milliseconds based on the rate limit for the performance level of the rate limiting license. The token bucket may determine the maximum total number of tokens used by the throttler to identify the number of data packets to propagate or throttle in a burst and not in accordance with the rate limit. In some embodiments, the throttler receives a network packet, determines that the token bucket has reached the maximum size and discards the network packet in response to the determination. In other embodiments, the throttler receives a network packet, determines that the token bucket has reached the maximum size and waits until a next available token to propagate or throttle a next data packet.

In some embodiments, the rate limiting manager establishes a throughput rate limit in bits per second based on the rate limit for the performance level of the rate limiting license. In further embodiments, a token generator generates a token for a token bucket at a rate specified by the throughput rate limit. In yet further embodiments, the rate limiting manager establishes a packet rate in packets per second based on the rate limit for the performance level of the rate limiting license. In some embodiments, the throttler receives a network packet having a number of bytes, and removes, or sends an instruction to remove, a number of tokens from a token bucket equal to the number of bytes. In some embodiments, the throttler receives a network packet having a number of bytes, determines that a number of tokens in a token bucket is less than the number of bytes and does not remove a token from the token bucket. In some embodiments, the throttler provides the network packet to an excess packet handler.

In other aspects, the present application is directed to a system for controlling a rate of a traffic of a device in accordance with a rate limit identified by a rate limiting license. The system may include a rate limiting manager of an intermediary device that processes network

traffic between a plurality of clients and a plurality of servers identifying presence of a rate limiting license identifying a performance level. The rate limiting manager may establish a rate limit based on the performance level of the rate limiting license. A throttler of the intermediary may controlling a rate of receiving network packets in accordance with the rate limit.

The details of various embodiments of the invention are set forth in the accompanying drawings and the description below.

Brief Description of the Figures

The foregoing and other objects, aspects, features, and advantages of the invention will become more apparent and better understood by referring to the following description taken in conjunction with the accompanying drawings, in which:

FIG. IA is a block diagram of an embodiment of a network environment for a client to access a server via an appliance;

FIG. IB is a block diagram of an embodiment of an environment for delivering a computing environment from a server to a client via an appliance;

FIG. 1C is a block diagram of another embodiment of an environment for delivering a computing environment from a server to a client via an appliance;

FIG. ID is a block diagram of another embodiment of an environment for delivering a computing environment from a server to a client via an appliance;

FIGs. IE - IH are block diagrams of embodiments of a computing device;

FIG. 2A is a block diagram of an embodiment of an appliance for processing communications between a client and a server;

FIG. 2B is a block diagram of another embodiment of an appliance for optimizing, accelerating, load-balancing and routing communications between a client and a server;

FIG. 3 is a block diagram of an embodiment of a client for communicating with a server via the appliance;

FIG. 4A is a block diagram of an embodiment of a virtualization environment;

FIG. 4B is a block diagram of another embodiment of a virtualization environment;

FIG. 4C is a block diagram of an embodiment of a virtualized appliance;

FIG. 5A are block diagrams of embodiments of approaches to implementing parallelism in a multi-core system;

FIG. 5B is a block diagram of an embodiment of a system utilizing a multi-core system;

FIG. 5C is a block diagram of another embodiment of an aspect of a multi-core system;

FIG. 6A are block diagrams of an embodiments of a system for controlling a rate of traffic traversing an intermediary device; and

FIG. 6B is a flow diagram of an embodiment of steps of a method for controlling a rate of traffic traversing an intermediary device.

In the drawings, like reference numbers generally indicate identical, functionally similar, and/or structurally similar elements.

Detailed Description of the Invention

For purposes of reading the description of the various embodiments below, the following descriptions of the sections of the specification and their respective contents may be helpful:

- Section A describes a network environment and computing environment which may be useful for practicing embodiments described herein;
- Section B describes embodiments of systems and methods for delivering a computing environment to a remote user;
- Section C describes embodiments of systems and methods for accelerating communications between a client and a server;
- Section D describes embodiments of systems and methods for virtualizing an application delivery controller;
- Section E describes embodiments of systems and methods for providing a multi-core architecture and environment;
- Section F describes embodiments of systems and methods for controlling a rate of traffic traversing an intermediary device

A. Network and Computing Environment

Prior to discussing the specifics of embodiments of the systems and methods of an appliance and/or client, it may be helpful to discuss the network and computing environments in which such embodiments may be deployed. Referring now to Figure IA, an embodiment of a network environment is depicted. In brief overview, the network environment comprises

one or more clients 102a-102n (also generally referred to as local machine(s) 102, or client(s) 102) in communication with one or more servers 106a-106n (also generally referred to as server(s) 106, or remote machine(s) 106) via one or more networks 104, 104' (generally referred to as network 104). In some embodiments, a client 102 communicates with a server 106 via an appliance 200.

Although FIG. IA shows a network 104 and a network 104' between the clients 102 and the servers 106, the clients 102 and the servers 106 may be on the same network 104. The networks 104 and 104' can be the same type of network or different types of networks. The network 104 and/or the network 104' can be a local-area network (LAN), such as a company Intranet, a metropolitan area network (MAN), or a wide area network (WAN), such as the Internet or the World Wide Web. In one embodiment, network 104' may be a private network and network 104 may be a public network. In some embodiments, network 104 may be a private network and network 104' a public network. In another embodiment, networks 104 and 104' may both be private networks. In some embodiments, clients 102 may be located at a branch office of a corporate enterprise communicating via a WAN connection over the network 104 to the servers 106 located at a corporate data center.

The network 104 and/or 104' be any type and/or form of network and may include any of the following: a point to point network, a broadcast network, a wide area network, a local area network, a telecommunications network, a data communication network, a computer network, an ATM (Asynchronous Transfer Mode) network, a SONET (Synchronous Optical Network) network, a SDH (Synchronous Digital Hierarchy) network, a wireless network and a wireline network. In some embodiments, the network 104 may comprise a wireless link, such as an infrared channel or satellite band. The topology of the network 104 and/or 104' may be a bus, star, or ring network topology. The network 104 and/or 104' and network topology may be of any such network or network topology as known to those ordinarily skilled in the art capable of supporting the operations described herein.

As shown in FIG. IA, the appliance 200, which also may be referred to as an interface unit 200 or gateway 200, is shown between the networks 104 and 104'. In some embodiments, the appliance 200 may be located on network 104. For example, a branch office of a corporate enterprise may deploy an appliance 200 at the branch office. In other embodiments, the appliance 200 may be located on network 104'. For example, an appliance 200 may be located at a corporate data center. In yet another embodiment, a plurality of appliances 200 may be deployed on network 104. In some embodiments, a plurality of

appliances 200 may be deployed on network 104'. In one embodiment, a first appliance 200 communicates with a second appliance 200'. In other embodiments, the appliance 200 could be a part of any client 102 or server 106 on the same or different network 104,104' as the client 102. One or more appliances 200 may be located at any point in the network or network communications path between a client 102 and a server 106.

In some embodiments, the appliance 200 comprises any of the network devices manufactured by Citrix Systems, Inc. of Ft. Lauderdale Florida, referred to as Citrix NetScaler devices. In other embodiments, the appliance 200 includes any of the product embodiments referred to as WebAccelerator and BigIP manufactured by F5 Networks, Inc. of Seattle, Washington. In another embodiment, the appliance 205 includes any of the DX acceleration device platforms and/or the SSL VPN series of devices, such as SA 700, SA 2000, SA 4000, and SA 6000 devices manufactured by Juniper Networks, Inc. of Sunnyvale, California. In yet another embodiment, the appliance 200 includes any application acceleration and/or security related appliances and/or software manufactured by Cisco Systems, Inc. of San Jose, California, such as the Cisco ACE Application Control Engine Module service software and network modules, and Cisco AVS Series Application Velocity System.

In one embodiment, the system may include multiple, logically-grouped servers 106. In these embodiments, the logical group of servers may be referred to as a server farm 38. In some of these embodiments, the serves 106 may be geographically dispersed. In some cases, a farm 38 may be administered as a single entity. In other embodiments, the server farm 38 comprises a plurality of server farms 38. In one embodiment, the server farm executes one or more applications on behalf of one or more clients 102.

The servers 106 within each farm 38 can be heterogeneous. One or more of the servers 106 can operate according to one type of operating system platform (e.g., WINDOWS NT, manufactured by Microsoft Corp. of Redmond, Washington), while one or more of the other servers 106 can operate on according to another type of operating system platform (e.g., Unix or Linux). The servers 106 of each farm 38 do not need to be physically proximate to another server 106 in the same farm 38. Thus, the group of servers 106 logically grouped as a farm 38 may be interconnected using a wide-area network (WAN) connection or medium-area network (MAN) connection. For example, a farm 38 may include servers 106 physically located in different continents or different regions of a continent, country, state, city, campus, or room. Data transmission speeds between servers 106 in the farm 38 can be increased if the

servers 106 are connected using a local-area network (LAN) connection or some form of direct connection.

Servers 106 may be referred to as a file server, application server, web server, proxy server, or gateway server. In some embodiments, a server 106 may have the capacity to function as either an application server or as a master application server. In one embodiment, a server 106 may include an Active Directory. The clients 102 may also be referred to as client nodes or endpoints. In some embodiments, a client 102 has the capacity to function as both a client node seeking access to applications on a server and as an application server providing access to hosted applications for other clients 102a-102n.

In some embodiments, a client 102 communicates with a server 106. In one embodiment, the client 102 communicates directly with one of the servers 106 in a farm 38. In another embodiment, the client 102 executes a program neighborhood application to communicate with a server 106 in a farm 38. In still another embodiment, the server 106 provides the functionality of a master node. In some embodiments, the client 102 communicates with the server 106 in the farm 38 through a network 104. Over the network 104, the client 102 can, for example, request execution of various applications hosted by the servers 106a-106n in the farm 38 and receive output of the results of the application execution for display. In some embodiments, only the master node provides the functionality required to identify and provide address information associated with a server 106' hosting a requested application.

In one embodiment, the server 106 provides functionality of a web server. In another embodiment, the server 106a receives requests from the client 102, forwards the requests to a second server 106b and responds to the request by the client 102 with a response to the request from the server 106b. In still another embodiment, the server 106 acquires an enumeration of applications available to the client 102 and address information associated with a server 106 hosting an application identified by the enumeration of applications. In yet another embodiment, the server 106 presents the response to the request to the client 102 using a web interface. In one embodiment, the client 102 communicates directly with the server 106 to access the identified application. In another embodiment, the client 102 receives application output data, such as display data, generated by an execution of the identified application on the server 106.

Referring now to FIG. IB, an embodiment of a network environment deploying multiple appliances 200 is depicted. A first appliance 200 may be deployed on a first network 104 and a second appliance 200' on a second network 104'. For example a

corporate enterprise may deploy a first appliance 200 at a branch office and a second appliance 200' at a data center. In another embodiment, the first appliance 200 and second appliance 200' are deployed on the same network 104 or network 104. For example, a first appliance 200 may be deployed for a first server farm 38, and a second appliance 200 may be deployed for a second server farm 38'. In another example, a first appliance 200 may be deployed at a first branch office while the second appliance 200' is deployed at a second branch office'. In some embodiments, the first appliance 200 and second appliance 200' work in cooperation or in conjunction with each other to accelerate network traffic or the delivery of application and data between a client and a server

Referring now to FIG. 1C, another embodiment of a network environment deploying the appliance 200 with one or more other types of appliances, such as between one or more WAN optimization appliance 205, 205' is depicted. For example a first WAN optimization appliance 205 is shown between networks 104 and 104' and a second WAN optimization appliance 205' may be deployed between the appliance 200 and one or more servers 106. By way of example, a corporate enterprise may deploy a first WAN optimization appliance 205 at a branch office and a second WAN optimization appliance 205' at a data center. In some embodiments, the appliance 205 may be located on network 104'. In other embodiments, the appliance 205' may be located on network 104. In some embodiments, the appliance 205' may be located on network 104' or network 104". In one embodiment, the appliance 205 and 205' are on the same network. In another embodiment, the appliance 205 and 205' are on different networks. In another example, a first WAN optimization appliance 205 may be deployed for a first server farm 38 and a second WAN optimization appliance 205' for a second server farm 38'

In one embodiment, the appliance 205 is a device for accelerating, optimizing or otherwise improving the performance, operation, or quality of service of any type and form of network traffic, such as traffic to and/or from a WAN connection. In some embodiments, the appliance 205 is a performance enhancing proxy. In other embodiments, the appliance 205 is any type and form of WAN optimization or acceleration device, sometimes also referred to as a WAN optimization controller. In one embodiment, the appliance 205 is any of the product embodiments referred to as WANScaler manufactured by Citrix Systems, Inc. of Ft. Lauderdale, Florida. In other embodiments, the appliance 205 includes any of the product embodiments referred to as BIG-IP link controller and WANjet manufactured by F5 Networks, Inc. of Seattle, Washington. In another embodiment, the appliance 205 includes any of the WX and WXC WAN acceleration device platforms manufactured by Juniper

Networks, Inc. of Sunnyvale, California. In some embodiments, the appliance 205 includes any of the steelhead line of WAN optimization appliances manufactured by Riverbed Technology of San Francisco, California. In other embodiments, the appliance 205 includes any of the WAN related devices manufactured by Expand Networks Inc. of Roseland, New Jersey. In one embodiment, the appliance 205 includes any of the WAN related appliances manufactured by Packeteer Inc. of Cupertino, California, such as the PacketShaper, iShared, and SkyX product embodiments provided by Packeteer. In yet another embodiment, the appliance 205 includes any WAN related appliances and/or software manufactured by Cisco Systems, Inc. of San Jose, California, such as the Cisco Wide Area Network Application Services software and network modules, and Wide Area Network engine appliances.

In one embodiment, the appliance 205 provides application and data acceleration services for branch-office or remote offices. In one embodiment, the appliance 205 includes optimization of Wide Area File Services (WAFS). In another embodiment, the appliance 205 accelerates the delivery of files, such as via the Common Internet File System (CIFS) protocol. In other embodiments, the appliance 205 provides caching in memory and/or storage to accelerate delivery of applications and data. In one embodiment, the appliance 205 provides compression of network traffic at any level of the network stack or at any protocol or network layer. In another embodiment, the appliance 205 provides transport layer protocol optimizations, flow control, performance enhancements or modifications and/or management to accelerate delivery of applications and data over a WAN connection. For example, in one embodiment, the appliance 205 provides Transport Control Protocol (TCP) optimizations. In other embodiments, the appliance 205 provides optimizations, flow control, performance enhancements or modifications and/or management for any session or application layer protocol.

In another embodiment, the appliance 205 encoded any type and form of data or information into custom or standard TCP and/or IP header fields or option fields of network packet to announce presence, functionality or capability to another appliance 205'. In another embodiment, an appliance 205' may communicate with another appliance 205' using data encoded in both TCP and/or IP header fields or options. For example, the appliance may use TCP option(s) or IP header fields or options to communicate one or more parameters to be used by the appliances 205, 205' in performing functionality, such as WAN acceleration, or for working in conjunction with each other.

In some embodiments, the appliance 200 preserves any of the information encoded in TCP and/or IP header and/or option fields communicated between appliances 205 and 205'.

For example, the appliance 200 may terminate a transport layer connection traversing the appliance 200, such as a transport layer connection from between a client and a server traversing appliances 205 and 205'. In one embodiment, the appliance 200 identifies and preserves any encoded information in a transport layer packet transmitted by a first appliance 205 via a first transport layer connection and communicates a transport layer packet with the encoded information to a second appliance 205' via a second transport layer connection.

Referring now to FIG. ID, a network environment for delivering and/or operating a computing environment on a client 102 is depicted. In some embodiments, a server 106 includes an application delivery system 190 for delivering a computing environment or an application and/or data file to one or more clients 102. In brief overview, a client 10 is in communication with a server 106 via network 104, 104' and appliance 200. For example, the client 102 may reside in a remote office of a company, e.g., a branch office, and the server 106 may reside at a corporate data center. The client 102 comprises a client agent 120, and a computing environment 15. The computing environment 15 may execute or operate an application that accesses, processes or uses a data file. The computing environment 15, application and/or data file may be delivered via the appliance 200 and/or the server 106.

In some embodiments, the appliance 200 accelerates delivery of a computing environment 15, or any portion thereof, to a client 102. In one embodiment, the appliance 200 accelerates the delivery of the computing environment 15 by the application delivery system 190. For example, the embodiments described herein may be used to accelerate delivery of a streaming application and data file processable by the application from a central corporate data center to a remote user location, such as a branch office of the company. In another embodiment, the appliance 200 accelerates transport layer traffic between a client 102 and a server 106. The appliance 200 may provide acceleration techniques for accelerating any transport layer payload from a server 106 to a client 102, such as: 1) transport layer connection pooling, 2) transport layer connection multiplexing, 3) transport control protocol buffering, 4) compression and 5) caching. In some embodiments, the appliance 200 provides load balancing of servers 106 in responding to requests from clients 102. In other embodiments, the appliance 200 acts as a proxy or access server to provide access to the one or more servers 106. In another embodiment, the appliance 200 provides a secure virtual private network connection from a first network 104 of the client 102 to the second network 104' of the server 106, such as an SSL VPN connection. It yet other embodiments, the appliance 200 provides application firewall security, control and management of the connection and communications between a client 102 and a server 106.

In some embodiments, the application delivery management system 190 provides application delivery techniques to deliver a computing environment to a desktop of a user, remote or otherwise, based on a plurality of execution methods and based on any authentication and authorization policies applied via a policy engine 195. With these techniques, a remote user may obtain a computing environment and access to server stored applications and data files from any network connected device 100. In one embodiment, the application delivery system 190 may reside or execute on a server 106. In another embodiment, the application delivery system 190 may reside or execute on a plurality of servers 106a-106n. In some embodiments, the application delivery system 190 may execute in a server farm 38. In one embodiment, the server 106 executing the application delivery system 190 may also store or provide the application and data file. In another embodiment, a first set of one or more servers 106 may execute the application delivery system 190, and a different server 106n may store or provide the application and data file. In some embodiments, each of the application delivery system 190, the application, and data file may reside or be located on different servers. In yet another embodiment, any portion of the application delivery system 190 may reside, execute or be stored on or distributed to the appliance 200, or a plurality of appliances.

The client 102 may include a computing environment 15 for executing an application that uses or processes a data file. The client 102 via networks 104, 104' and appliance 200 may request an application and data file from the server 106. In one embodiment, the appliance 200 may forward a request from the client 102 to the server 106. For example, the client 102 may not have the application and data file stored or accessible locally. In response to the request, the application delivery system 190 and/or server 106 may deliver the application and data file to the client 102. For example, in one embodiment, the server 106 may transmit the application as an application stream to operate in computing environment 15 on client 102.

In some embodiments, the application delivery system 190 comprises any portion of the Citrix Access Suite™ by Citrix Systems, Inc., such as the MetaFrame or Citrix Presentation Server™ and/or any of the Microsoft® Windows Terminal Services manufactured by the Microsoft Corporation. In one embodiment, the application delivery system 190 may deliver one or more applications to clients 102 or users via a remote-display protocol or otherwise via remote-based or server-based computing. In another embodiment, the application delivery system 190 may deliver one or more applications to clients or users via steaming of the application.

In one embodiment, the application delivery system 190 includes a policy engine 195 for controlling and managing the access to, selection of application execution methods and the delivery of applications. In some embodiments, the policy engine 195 determines the one or more applications a user or client 102 may access. In another embodiment, the policy engine 195 determines how the application should be delivered to the user or client 102, e.g., the method of execution. In some embodiments, the application delivery system 190 provides a plurality of delivery techniques from which to select a method of application execution, such as a server-based computing, streaming or delivering the application locally to the client 120 for local execution.

In one embodiment, a client 102 requests execution of an application program and the application delivery system 190 comprising a server 106 selects a method of executing the application program. In some embodiments, the server 106 receives credentials from the client 102. In another embodiment, the server 106 receives a request for an enumeration of available applications from the client 102. In one embodiment, in response to the request or receipt of credentials, the application delivery system 190 enumerates a plurality of application programs available to the client 102. The application delivery system 190 receives a request to execute an enumerated application. The application delivery system 190 selects one of a predetermined number of methods for executing the enumerated application, for example, responsive to a policy of a policy engine. The application delivery system 190 may select a method of execution of the application enabling the client 102 to receive application-output data generated by execution of the application program on a server 106. The application delivery system 190 may select a method of execution of the application enabling the local machine 10 to execute the application program locally after retrieving a plurality of application files comprising the application. In yet another embodiment, the application delivery system 190 may select a method of execution of the application to stream the application via the network 104 to the client 102.

A client 102 may execute, operate or otherwise provide an application, which can be any type and/or form of software, program, or executable instructions such as any type and/or form of web browser, web-based client, client-server application, a thin-client computing client, an ActiveX control, or a Java applet, or any other type and/or form of executable instructions capable of executing on client 102. In some embodiments, the application may be a server-based or a remote-based application executed on behalf of the client 102 on a server 106. In one embodiments the server 106 may display output to the client 102 using any thin-client or remote-display protocol, such as the Independent Computing Architecture

(ICA) protocol manufactured by Citrix Systems, Inc. of Ft. Lauderdale, Florida or the Remote Desktop Protocol (RDP) manufactured by the Microsoft Corporation of Redmond, Washington. The application can use any type of protocol and it can be, for example, an HTTP client, an FTP client, an Oscar client, or a Telnet client. In other embodiments, the application comprises any type of software related to VoIP communications, such as a soft IP telephone. In further embodiments, the application comprises any application related to real-time data communications, such as applications for streaming video and/or audio.

In some embodiments, the server 106 or a server farm 38 may be running one or more applications, such as an application providing a thin-client computing or remote display presentation application. In one embodiment, the server 106 or server farm 38 executes as an application, any portion of the Citrix Access Suite™ by Citrix Systems, Inc., such as the MetaFrame or Citrix Presentation Server™, and/or any of the Microsoft® Windows Terminal Services manufactured by the Microsoft Corporation. In one embodiment, the application is an ICA client, developed by Citrix Systems, Inc. of Fort Lauderdale, Florida. In other embodiments, the application includes a Remote Desktop (RDP) client, developed by Microsoft Corporation of Redmond, Washington. Also, the server 106 may run an application, which for example, may be an application server providing email services such as Microsoft Exchange manufactured by the Microsoft Corporation of Redmond, Washington, a web or Internet server, or a desktop sharing server, or a collaboration server. In some embodiments, any of the applications may comprise any type of hosted service or products, such as GoToMeeting™ provided by Citrix Online Division, Inc. of Santa Barbara, California, WebEx™ provided by WebEx, Inc. of Santa Clara, California, or Microsoft Office Live Meeting provided by Microsoft Corporation of Redmond, Washington.

Still referring to FIG. ID, an embodiment of the network environment may include a monitoring server 106A. The monitoring server 106A may include any type and form performance monitoring service 198. The performance monitoring service 198 may include monitoring, measurement and/or management software and/or hardware, including data collection, aggregation, analysis, management and reporting. In one embodiment, the performance monitoring service 198 includes one or more monitoring agents 197. The monitoring agent 197 includes any software, hardware or combination thereof for performing monitoring, measurement and data collection activities on a device, such as a client 102, server 106 or an appliance 200, 205. In some embodiments, the monitoring agent 197 includes any type and form of script, such as Visual Basic script, or Javascript. In one embodiment, the monitoring agent 197 executes transparently to any application and/or user

of the device. In some embodiments, the monitoring agent 197 is installed and operated unobtrusively to the application or client. In yet another embodiment, the monitoring agent 197 is installed and operated without any instrumentation for the application or device.

In some embodiments, the monitoring agent 197 monitors, measures and collects data on a predetermined frequency. In other embodiments, the monitoring agent 197 monitors, measures and collects data based upon detection of any type and form of event. For example, the monitoring agent 197 may collect data upon detection of a request for a web page or receipt of an HTTP response. In another example, the monitoring agent 197 may collect data upon detection of any user input events, such as a mouse click. The monitoring agent 197 may report or provide any monitored, measured or collected data to the monitoring service 198. In one embodiment, the monitoring agent 197 transmits information to the monitoring service 198 according to a schedule or a predetermined frequency. In another embodiment, the monitoring agent 197 transmits information to the monitoring service 198 upon detection of an event.

In some embodiments, the monitoring service 198 and/or monitoring agent 197 performs monitoring and performance measurement of any network resource or network infrastructure element, such as a client, server, server farm, appliance 200, appliance 205, or network connection. In one embodiment, the monitoring service 198 and/or monitoring agent 197 performs monitoring and performance measurement of any transport layer connection, such as a TCP or UDP connection. In another embodiment, the monitoring service 198 and/or monitoring agent 197 monitors and measures network latency. In yet one embodiment, the monitoring service 198 and/or monitoring agent 197 monitors and measures bandwidth utilization.

In other embodiments, the monitoring service 198 and/or monitoring agent 197 monitors and measures end-user response times. In some embodiments, the monitoring service 198 performs monitoring and performance measurement of an application. In another embodiment, the monitoring service 198 and/or monitoring agent 197 performs monitoring and performance measurement of any session or connection to the application. In one embodiment, the monitoring service 198 and/or monitoring agent 197 monitors and measures performance of a browser. In another embodiment, the monitoring service 198 and/or monitoring agent 197 monitors and measures performance of HTTP based transactions. In some embodiments, the monitoring service 198 and/or monitoring agent 197 monitors and measures performance of a Voice over IP (VoIP) application or session. In other embodiments, the monitoring service 198 and/or monitoring agent 197 monitors and

measures performance of a remote display protocol application, such as an ICA client or RDP client. In yet another embodiment, the monitoring service 198 and/or monitoring agent 197 monitors and measures performance of any type and form of streaming media. In still a further embodiment, the monitoring service 198 and/or monitoring agent 197 monitors and measures performance of a hosted application or a Software-As-A-Service (SaaS) delivery model.

In some embodiments, the monitoring service 198 and/or monitoring agent 197 performs monitoring and performance measurement of one or more transactions, requests or responses related to application. In other embodiments, the monitoring service 198 and/or monitoring agent 197 monitors and measures any portion of an application layer stack, such as any .NET or J2EE calls. In one embodiment, the monitoring service 198 and/or monitoring agent 197 monitors and measures database or SQL transactions. In yet another embodiment, the monitoring service 198 and/or monitoring agent 197 monitors and measures any method, function or application programming interface (API) call.

In one embodiment, the monitoring service 198 and/or monitoring agent 197 performs monitoring and performance measurement of a delivery of application and/or data from a server to a client via one or more appliances, such as appliance 200 and/or appliance 205. In some embodiments, the monitoring service 198 and/or monitoring agent 197 monitors and measures performance of delivery of a virtualized application. In other embodiments, the monitoring service 198 and/or monitoring agent 197 monitors and measures performance of delivery of a streaming application. In another embodiment, the monitoring service 198 and/or monitoring agent 197 monitors and measures performance of delivery of a desktop application to a client and/or the execution of the desktop application on the client. In another embodiment, the monitoring service 198 and/or monitoring agent 197 monitors and measures performance of a client/server application.

In one embodiment, the monitoring service 198 and/or monitoring agent 197 is designed and constructed to provide application performance management for the application delivery system 190. For example, the monitoring service 198 and/or monitoring agent 197 may monitor, measure and manage the performance of the delivery of applications via the Citrix Presentation Server. In this example, the monitoring service 198 and/or monitoring agent 197 monitors individual ICA sessions. The monitoring service 198 and/or monitoring agent 197 may measure the total and per session system resource usage, as well as application and networking performance. The monitoring service 198 and/or monitoring agent 197 may identify the active servers for a given user and/or user session. In some embodiments, the

monitoring service 198 and/or monitoring agent 197 monitors back-end connections between the application delivery system 190 and an application and/or database server. The monitoring service 198 and/or monitoring agent 197 may measure network latency, delay and volume per user-session or ICA session.

In some embodiments, the monitoring service 198 and/or monitoring agent 197 measures and monitors memory usage for the application delivery system 190, such as total memory usage, per user session and/or per process. In other embodiments, the monitoring service 198 and/or monitoring agent 197 measures and monitors CPU usage the application delivery system 190, such as total CPU usage, per user session and/or per process. In another embodiments, the monitoring service 198 and/or monitoring agent 197 measures and monitors the time required to log-in to an application, a server, or the application delivery system, such as Citrix Presentation Server. In one embodiment, the monitoring service 198 and/or monitoring agent 197 measures and monitors the duration a user is logged into an application, a server, or the application delivery system 190. In some embodiments, the monitoring service 198 and/or monitoring agent 197 measures and monitors active and inactive session counts for an application, server or application delivery system session. In yet another embodiment, the monitoring service 198 and/or monitoring agent 197 measures and monitors user session latency.

In yet further embodiments, the monitoring service 198 and/or monitoring agent 197 measures and monitors measures and monitors any type and form of server metrics. In one embodiment, the monitoring service 198 and/or monitoring agent 197 measures and monitors metrics related to system memory, CPU usage, and disk storage. In another embodiment, the monitoring service 198 and/or monitoring agent 197 measures and monitors metrics related to page faults, such as page faults per second. In other embodiments, the monitoring service 198 and/or monitoring agent 197 measures and monitors round-trip time metrics. In yet another embodiment, the monitoring service 198 and/or monitoring agent 197 measures and monitors metrics related to application crashes, errors and/or hangs.

In some embodiments, the monitoring service 198 and monitoring agent 198 includes any of the product embodiments referred to as EdgeSight manufactured by Citrix Systems, Inc. of Ft. Lauderdale, Florida. In another embodiment, the performance monitoring service 198 and/or monitoring agent 198 includes any portion of the product embodiments referred to as the TrueView product suite manufactured by the Symphoniq Corporation of Palo Alto, California. In one embodiment, the performance monitoring service 198 and/or monitoring agent 198 includes any portion of the product embodiments referred to as the TeaLeaf CX

product suite manufactured by the TeaLeaf Technology Inc. of San Francisco, California. In other embodiments, the performance monitoring service 198 and/or monitoring agent 198 includes any portion of the business service management products, such as the BMC Performance Manager and Patrol products, manufactured by BMC Software, Inc. of Houston, Texas.

The client 102, server 106, and appliance 200 may be deployed as and/or executed on any type and form of computing device, such as a computer, network device or appliance capable of communicating on any type and form of network and performing the operations described herein. FIGs. IE and IF depict block diagrams of a computing device 100 useful for practicing an embodiment of the client 102, server 106 or appliance 200. As shown in FIGs. IE and IF, each computing device 100 includes a central processing unit 101, and a main memory unit 122. As shown in FIG. IE, a computing device 100 may include a visual display device 124, a keyboard 126 and/or a pointing device 127, such as a mouse. Each computing device 100 may also include additional optional elements, such as one or more input/output devices 130a-130b (generally referred to using reference numeral 130), and a cache memory 140 in communication with the central processing unit 101.

The central processing unit 101 is any logic circuitry that responds to and processes instructions fetched from the main memory unit 122. In many embodiments, the central processing unit is provided by a microprocessor unit, such as: those manufactured by Intel Corporation of Mountain View, California; those manufactured by Motorola Corporation of Schaumburg, Illinois; those manufactured by Transmeta Corporation of Santa Clara, California; the RS/6000 processor, those manufactured by International Business Machines of White Plains, New York; or those manufactured by Advanced Micro Devices of Sunnyvale, California. The computing device 100 may be based on any of these processors, or any other processor capable of operating as described herein.

Main memory unit 122 may be one or more memory chips capable of storing data and allowing any storage location to be directly accessed by the microprocessor 101, such as Static random access memory (SRAM), Burst SRAM or 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), Enhanced DRAM (EDRAM), synchronous DRAM (SDRAM), JEDEC SRAM, PClOO SDRAM, Double Data Rate SDRAM (DDR SDRAM), Enhanced SDRAM (ESDRAM), SyncLink DRAM (SLDRAM), Direct Rambus DRAM (DRDRAM), or Ferroelectric RAM

(FRAM). The main memory 122 may be based on any of the above described memory chips, or any other available memory chips capable of operating as described herein. In the embodiment shown in FIG. IE, the processor 101 communicates with main memory 122 via a system bus 150 (described in more detail below). FIG. IF depicts an embodiment of a computing device 100 in which the processor communicates directly with main memory 122 via a memory port 103. For example, in FIG. IF the main memory 122 may be DRDRAM.

FIG. IF depicts an embodiment in which the main processor 101 communicates directly with cache memory 140 via a secondary bus, sometimes referred to as a backside bus. In other embodiments, the main processor 101 communicates with cache memory 140 using the system bus 150. Cache memory 140 typically has a faster response time than main memory 122 and is typically provided by SRAM, BSRAM, or EDRAM. In the embodiment shown in FIG. IF, the processor 101 communicates with various I/O devices 130 via a local system bus 150. Various busses may be used to connect the central processing unit 101 to any of the I/O devices 130, including a VESA VL bus, an ISA bus, an EISA bus, a MicroChannel Architecture (MCA) bus, a PCI bus, a PCI-X bus, a PCI-Express bus, or a NuBus. For embodiments in which the I/O device is a video display 124, the processor 101 may use an Advanced Graphics Port (AGP) to communicate with the display 124. FIG. IF depicts an embodiment of a computer 100 in which the main processor 101 communicates directly with I/O device 130b via HyperTransport, Rapid I/O, or InfiniBand. FIG. IF also depicts an embodiment in which local busses and direct communication are mixed: the processor 101 communicates with I/O device 130b using a local interconnect bus while communicating with I/O device 130a directly.

The computing device 100 may support any suitable installation device 116, such as a floppy disk drive for receiving floppy disks such as 3.5-inch, 5.25-inch disks or ZIP disks, a CD-ROM drive, a CD-R/RW drive, a DVD-ROM drive, tape drives of various formats, USB device, hard-drive or any other device suitable for installing software and programs such as any client agent 120, or portion thereof. The computing device 100 may further comprise a storage device 128, such as one or more hard disk drives or redundant arrays of independent disks, for storing an operating system and other related software, and for storing application software programs such as any program related to the client agent 120. Optionally, any of the installation devices 116 could also be used as the storage device 128. Additionally, the operating system and the software can be run from a bootable medium, for example, a bootable CD, such as KNOPPIX®, a bootable CD for GNU/Linux that is available as a GNU/Linux distribution from knoppix.net.

Furthermore, the computing device 100 may include a network interface 118 to interface to a Local Area Network (LAN), Wide Area Network (WAN) or the Internet through a variety of connections including, but not limited to, standard telephone lines, LAN or WAN links *(e.g.,* 802.1 1, Tl, T3, 56kb, X.25), broadband connections *(e.g.,* ISDN, Frame Relay, ATM), wireless connections, or some combination of any or all of the above. The network interface 118 may comprise a built-in network adapter, network interface card, PCMCIA network card, card bus network adapter, wireless network adapter, USB network adapter, modem or any other device suitable for interfacing the computing device 100 to any type of network capable of communication and performing the operations described herein. A wide variety of I/O devices 13Oa-13On may be present in the computing device 100. Input devices include keyboards, mice, trackpads, trackballs, microphones, and drawing tablets. Output devices include video displays, speakers, inkjet printers, laser printers, and dye-sublimation printers. The I/O devices 130 may be controlled by an I/O controller 123 as shown in FIG. IE. The I/O controller may control one or more I/O devices such as a keyboard 126 and a pointing device 127, e.g., a mouse or optical pen. Furthermore, an I/O device may also provide storage 128 and/or an installation medium 116 for the computing device 100. In still other embodiments, the computing device 100 may provide USB connections to receive handheld USB storage devices such as the USB Flash Drive line of devices manufactured by Twintech Industry, Inc. of Los Alamitos, California.

In some embodiments, the computing device 100 may comprise or be connected to multiple display devices 124a-124n, which each may be of the same or different type and/or form. As such, any of the I/O devices 13Oa-13On and/or the I/O controller 123 may comprise any type and/or form of suitable hardware, software, or combination of hardware and software to support, enable or provide for the connection and use of multiple display devices 124a-124n by the computing device 100. For example, the computing device 100 may include any type and/or form of video adapter, video card, driver, and/or library to interface, communicate, connect or otherwise use the display devices 124a-124n. In one embodiment, a video adapter may comprise multiple connectors to interface to multiple display devices 124a-124n. In other embodiments, the computing device 100 may include multiple video adapters, with each video adapter connected to one or more of the display devices 124a-124n. In some embodiments, any portion of the operating system of the computing device 100 may be configured for using multiple displays 124a-124n. In other embodiments, one or more of the display devices 124a-124n may be provided by one or more other computing devices, such as computing devices 100a and 100b connected to the computing device 100, for

example, via a network. These embodiments may include any type of software designed and constructed to use another computer's display device as a second display device 124a for the computing device 100. One ordinarily skilled in the art will recognize and appreciate the various ways and embodiments that a computing device 100 may be configured to have multiple display devices 124a-124n.

In further embodiments, an I/O device 130 may be a bridge 170 between the system bus 150 and an external communication bus, such as a USB bus, an Apple Desktop Bus, an RS-232 serial connection, a SCSI bus, a FireWire bus, a FireWire 800 bus, an Ethernet bus, an AppleTalk bus, a Gigabit Ethernet bus, an Asynchronous Transfer Mode bus, a HIPPI bus, a Super HIPPI bus, a SerialPlus bus, a SCI/LAMP bus, a FibreChannel bus, or a Serial Attached small computer system interface bus.

A computing device 100 of the sort depicted in FIGs. 1E and 1F typically operate under the control of operating systems, which control scheduling of tasks and access to system resources. The computing device 100 can be running any operating system such as any of the versions of the Microsoft® Windows operating systems, the different releases of the Unix and Linux operating systems, any version of the Mac OS® for Macintosh computers, any embedded operating system, any real-time operating system, any open source operating system, any proprietary operating system, any operating systems for mobile computing devices, or any other operating system capable of running on the computing device and performing the operations described herein. Typical operating systems include: WINDOWS 3.x, WINDOWS 95, WINDOWS 98, WINDOWS 2000, WINDOWS NT 3.51, WINDOWS NT 4.0, WINDOWS CE, and WINDOWS XP, all of which are manufactured by Microsoft Corporation of Redmond, Washington; MacOS, manufactured by Apple Computer of Cupertino, California; OS/2, manufactured by International Business Machines of Armonk, New York; and Linux, a freely-available operating system distributed by Caldera Corp. of Salt Lake City, Utah, or any type and/or form of a Unix operating system, among others.

In other embodiments, the computing device 100 may have different processors, operating systems, and input devices consistent with the device. For example, in one embodiment the computer 100 is a Treo 180, 270, 1060, 600 or 650 smart phone manufactured by Palm, Inc. In this embodiment, the Treo smart phone is operated under the control of the PalmOS operating system and includes a stylus input device as well as a five-way navigator device. Moreover, the computing device 100 can be any workstation, desktop computer, laptop or notebook computer, server, handheld computer, mobile telephone, any

other computer, or other form of computing or telecommunications device that is capable of communication and that has sufficient processor power and memory capacity to perform the operations described herein.

As shown in FIG. IG, the computing device 100 may comprise multiple processors and may provide functionality for simultaneous execution of instructions or for simultaneous execution of one instruction on more than one piece of data. In some embodiments, the computing device 100 may comprise a parallel processor with one or more cores. In one of these embodiments, the computing device 100 is a shared memory parallel device, with multiple processors and/or multiple processor cores, accessing all available memory as a single global address space. In another of these embodiments, the computing device 100 is a distributed memory parallel device with multiple processors each accessing local memory only. In still another of these embodiments, the computing device 100 has both some memory which is shared and some memory which can only be accessed by particular processors or subsets of processors. In still even another of these embodiments, the computing device 100, such as a multi-core microprocessor, combines two or more independent processors into a single package, often a single integrated circuit (IC). In yet another of these embodiments, the computing device 100 includes a chip having a CELL BROADBAND ENGINE architecture and including a Power processor element and a plurality of synergistic processing elements, the Power processor element and the plurality of synergistic processing elements linked together by an internal high speed bus, which may be referred to as an element interconnect bus.

In some embodiments, the processors provide functionality for execution of a single instruction simultaneously on multiple pieces of data (SIMD). In other embodiments, the processors provide functionality for execution of multiple instructions simultaneously on multiple pieces of data (MIMD). In still other embodiments, the processor may use any combination of SIMD and MIMD cores in a single device.

In some embodiments, the computing device 100 may comprise a graphics processing unit. In one of these embodiments, depicted in FIG. IH, the computing device 100 includes at least one central processing unit 101 and at least one graphics processing unit. In another of these embodiments, the computing device 100 includes at least one parallel processing unit and at least one graphics processing unit. In still another of these embodiments, the computing device 100 includes a plurality of processing units of any type, one of the plurality of processing units comprising a graphics processing unit.

In some embodiments, a first computing device 100a executes an application on behalf of a user of a client computing device 100b. In other embodiments, a computing device 100a executes a virtual machine, which provides an execution session within which applications execute on behalf of a user or a client computing devices 100b. In one of these embodiments, the execution session is a hosted desktop session. In another of these embodiments, the computing device 100 executes a terminal services session. The terminal services session may provide a hosted desktop environment. In still another of these embodiments, the execution session provides access to a computing environment, which may comprise one or more of: an application, a plurality of applications, a desktop application, and a desktop session in which one or more applications may execute.

B. Appliance Architecture

FIG. 2A illustrates an example embodiment of the appliance 200. The architecture of the appliance 200 in FIG. 2A is provided by way of illustration only and is not intended to be limiting. As shown in FIG. 2, appliance 200 comprises a hardware layer 206 and a software layer divided into a user space 202 and a kernel space 204.

Hardware layer 206 provides the hardware elements upon which programs and services within kernel space 204 and user space 202 are executed. Hardware layer 206 also provides the structures and elements which allow programs and services within kernel space 204 and user space 202 to communicate data both internally and externally with respect to appliance 200. As shown in FIG. 2, the hardware layer 206 includes a processing unit 262 for executing software programs and services, a memory 264 for storing software and data, network ports 266 for transmitting and receiving data over a network, and an encryption processor 260 for performing functions related to Secure Sockets Layer processing of data transmitted and received over the network. In some embodiments, the central processing unit 262 may perform the functions of the encryption processor 260 in a single processor. Additionally, the hardware layer 206 may comprise multiple processors for each of the processing unit 262 and the encryption processor 260. The processor 262 may include any of the processors 101 described above in connection with FIGs. IE and IF. For example, in one embodiment, the appliance 200 comprises a first processor 262 and a second processor 262'. In other embodiments, the processor 262 or 262' comprises a multi-core processor.

Although the hardware layer 206 of appliance 200 is generally illustrated with an encryption processor 260, processor 260 may be a processor for performing functions related to any encryption protocol, such as the Secure Socket Layer (SSL) or Transport Layer

Security (TLS) protocol. In some embodiments, the processor 260 may be a general purpose processor (GPP), and in further embodiments, may have executable instructions for performing processing of any security related protocol.

Although the hardware layer 206 of appliance 200 is illustrated with certain elements in FIG. 2, the hardware portions or components of appliance 200 may comprise any type and form of elements, hardware or software, of a computing device, such as the computing device 100 illustrated and discussed herein in conjunction with FIGs. IE and IF. In some embodiments, the appliance 200 may comprise a server, gateway, router, switch, bridge or other type of computing or network device, and have any hardware and/or software elements associated therewith.

The operating system of appliance 200 allocates, manages, or otherwise segregates the available system memory into kernel space 204 and user space 204. In example software architecture 200, the operating system may be any type and/or form of Unix operating system although the invention is not so limited. As such, the appliance 200 can be running any operating system such as any of the versions of the Microsoft® Windows operating systems, the different releases of the Unix and Linux operating systems, any version of the Mac OS® for Macintosh computers, any embedded operating system, any network operating system, any real-time operating system, any open source operating system, any proprietary operating system, any operating systems for mobile computing devices or network devices, or any other operating system capable of running on the appliance 200 and performing the operations described herein.

The kernel space 204 is reserved for running the kernel 230, including any device drivers, kernel extensions or other kernel related software. As known to those skilled in the art, the kernel 230 is the core of the operating system, and provides access, control, and management of resources and hardware-related elements of the application 104. In accordance with an embodiment of the appliance 200, the kernel space 204 also includes a number of network services or processes working in conjunction with a cache manager 232, sometimes also referred to as the integrated cache, the benefits of which are described in detail further herein. Additionally, the embodiment of the kernel 230 will depend on the embodiment of the operating system installed, configured, or otherwise used by the device 200.

In one embodiment, the device 200 comprises one network stack 267, such as a TCP/IP based stack, for communicating with the client 102 and/or the server 106. In one embodiment, the network stack 267 is used to communicate with a first network, such as

network 108, and a second network 110. In some embodiments, the device 200 terminates a first transport layer connection, such as a TCP connection of a client 102, and establishes a second transport layer connection to a server 106 for use by the client 102, e.g., the second transport layer connection is terminated at the appliance 200 and the server 106. The first and second transport layer connections may be established via a single network stack 267. In other embodiments, the device 200 may comprise multiple network stacks, for example 267 and 267', and the first transport layer connection may be established or terminated at one network stack 267, and the second transport layer connection on the second network stack 267'. For example, one network stack may be for receiving and transmitting network packet on a first network, and another network stack for receiving and transmitting network packets on a second network. In one embodiment, the network stack 267 comprises a buffer 243 for queuing one or more network packets for transmission by the appliance 200.

As shown in FIG. 2, the kernel space 204 includes the cache manager 232, a high-speed layer 2-7 integrated packet engine 240, an encryption engine 234, a policy engine 236 and multi-protocol compression logic 238. Running these components or processes 232, 240, 234, 236 and 238 in kernel space 204 or kernel mode instead of the user space 202 improves the performance of each of these components, alone and in combination. Kernel operation means that these components or processes 232, 240, 234, 236 and 238 run in the core address space of the operating system of the device 200. For example, running the encryption engine 234 in kernel mode improves encryption performance by moving encryption and decryption operations to the kernel, thereby reducing the number of transitions between the memory space or a kernel thread in kernel mode and the memory space or a thread in user mode. For example, data obtained in kernel mode may not need to be passed or copied to a process or thread running in user mode, such as from a kernel level data structure to a user level data structure. In another aspect, the number of context switches between kernel mode and user mode are also reduced. Additionally, synchronization of and communications between any of the components or processes 232, 240, 235, 236 and 238 can be performed more efficiently in the kernel space 204.

In some embodiments, any portion of the components 232, 240, 234, 236 and 238 may run or operate in the kernel space 204, while other portions of these components 232, 240, 234, 236 and 238 may run or operate in user space 202. In one embodiment, the appliance 200 uses a kernel-level data structure providing access to any portion of one or more network packets, for example, a network packet comprising a request from a client 102 or a response from a server 106. In some embodiments, the kernel-level data structure may

be obtained by the packet engine 240 via a transport layer driver interface or filter to the network stack 267. The kernel-level data structure may comprise any interface and/or data accessible via the kernel space 204 related to the network stack 267, network traffic or packets received or transmitted by the network stack 267. In other embodiments, the kernel-level data structure may be used by any of the components or processes 232, 240, 234, 236 and 238 to perform the desired operation of the component or process. In one embodiment, a component 232, 240, 234, 236 and 238 is running in kernel mode 204 when using the kernel-level data structure, while in another embodiment, the component 232, 240, 234, 236 and 238 is running in user mode when using the kernel-level data structure. In some embodiments, the kernel-level data structure may be copied or passed to a second kernel-level data structure, or any desired user-level data structure.

The cache manager 232 may comprise software, hardware or any combination of software and hardware to provide cache access, control and management of any type and form of content, such as objects or dynamically generated objects served by the originating servers 106. The data, objects or content processed and stored by the cache manager 232 may comprise data in any format, such as a markup language, or communicated via any protocol. In some embodiments, the cache manager 232 duplicates original data stored elsewhere or data previously computed, generated or transmitted, in which the original data may require longer access time to fetch, compute or otherwise obtain relative to reading a cache memory element. Once the data is stored in the cache memory element, future use can be made by accessing the cached copy rather than refetching or recomputing the original data, thereby reducing the access time. In some embodiments, the cache memory element may comprise a data object in memory 264 of device 200. In other embodiments, the cache memory element may comprise memory having a faster access time than memory 264. In another embodiment, the cache memory element may comprise any type and form of storage element of the device 200, such as a portion of a hard disk. In some embodiments, the processing unit 262 may provide cache memory for use by the cache manager 232. In yet further embodiments, the cache manager 232 may use any portion and combination of memory, storage, or the processing unit for caching data, objects, and other content.

Furthermore, the cache manager 232 includes any logic, functions, rules, or operations to perform any embodiments of the techniques of the appliance 200 described herein. For example, the cache manager 232 includes logic or functionality to invalidate objects based on the expiration of an invalidation time period or upon receipt of an invalidation command from a client 102 or server 106. In some embodiments, the cache

manager 232 may operate as a program, service, process or task executing in the kernel space 204, and in other embodiments, in the user space 202. In one embodiment, a first portion of the cache manager 232 executes in the user space 202 while a second portion executes in the kernel space 204. In some embodiments, the cache manager 232 can comprise any type of general purpose processor (GPP), or any other type of integrated circuit, such as a Field Programmable Gate Array (FPGA), Programmable Logic Device (PLD), or Application Specific Integrated Circuit (ASIC).

The policy engine 236 may include, for example, an intelligent statistical engine or other programmable application(s). In one embodiment, the policy engine 236 provides a configuration mechanism to allow a user to identify, specify, define or configure a caching policy. Policy engine 236, in some embodiments, also has access to memory to support data structures such as lookup tables or hash tables to enable user-selected caching policy decisions. In other embodiments, the policy engine 236 may comprise any logic, rules, functions or operations to determine and provide access, control and management of objects, data or content being cached by the appliance 200 in addition to access, control and management of security, network traffic, network access, compression or any other function or operation performed by the appliance 200. Further examples of specific caching policies are further described herein.

The encryption engine 234 comprises any logic, business rules, functions or operations for handling the processing of any security related protocol, such as SSL or TLS, or any function related thereto. For example, the encryption engine 234 encrypts and decrypts network packets, or any portion thereof, communicated via the appliance 200. The encryption engine 234 may also setup or establish SSL or TLS connections on behalf of the client 102a-102n, server 106a-106n, or appliance 200. As such, the encryption engine 234 provides offloading and acceleration of SSL processing. In one embodiment, the encryption engine 234 uses a tunneling protocol to provide a virtual private network between a client 102a-102n and a server 106a-106n. In some embodiments, the encryption engine 234 is in communication with the Encryption processor 260. In other embodiments, the encryption engine 234 comprises executable instructions running on the Encryption processor 260.

The multi-protocol compression engine 238 comprises any logic, business rules, function or operations for compressing one or more protocols of a network packet, such as any of the protocols used by the network stack 267 of the device 200. In one embodiment, multi-protocol compression engine 238 compresses bi-directionally between clients 102a-102n and servers 106a-106n any TCP/IP based protocol, including Messaging Application

Programming Interface (MAPI) (email), File Transfer Protocol (FTP), HyperText Transfer Protocol (HTTP), Common Internet File System (CIFS) protocol (file transfer), Independent Computing Architecture (ICA) protocol, Remote Desktop Protocol (RDP), Wireless Application Protocol (WAP), Mobile IP protocol, and Voice Over IP (VoIP) protocol. In other embodiments, multi-protocol compression engine 238 provides compression of Hypertext Markup Language (HTML) based protocols and in some embodiments, provides compression of any markup languages, such as the Extensible Markup Language (XML). In one embodiment, the multi-protocol compression engine 238 provides compression of any high-performance protocol, such as any protocol designed for appliance 200 to appliance 200 communications. In another embodiment, the multi-protocol compression engine 238 compresses any payload of or any communication using a modified transport control protocol, such as Transaction TCP (T/TCP), TCP with selection acknowledgements (TCP-SACK), TCP with large windows (TCP-LW), a congestion prediction protocol such as the TCP-Vegas protocol, and a TCP spoofing protocol.

As such, the multi-protocol compression engine 238 accelerates performance for users accessing applications via desktop clients, e.g., Microsoft Outlook and non-Web thin clients, such as any client launched by popular enterprise applications like Oracle, SAP and Siebel, and even mobile clients, such as the Pocket PC. In some embodiments, the multi-protocol compression engine 238 by executing in the kernel mode 204 and integrating with packet processing engine 240 accessing the network stack 267 is able to compress any of the protocols carried by the TCP/IP protocol, such as any application layer protocol.

High speed layer 2-7 integrated packet engine 240, also generally referred to as a packet processing engine or packet engine, is responsible for managing the kernel-level processing of packets received and transmitted by appliance 200 via network ports 266. The high speed layer 2-7 integrated packet engine 240 may comprise a buffer for queuing one or more network packets during processing, such as for receipt of a network packet or transmission of a network packet. Additionally, the high speed layer 2-7 integrated packet engine 240 is in communication with one or more network stacks 267 to send and receive network packets via network ports 266. The high speed layer 2-7 integrated packet engine 240 works in conjunction with encryption engine 234, cache manager 232, policy engine 236 and multi-protocol compression logic 238. In particular, encryption engine 234 is configured to perform SSL processing of packets, policy engine 236 is configured to perform functions related to traffic management such as request-level content switching and request-level cache

redirection, and multi-protocol compression logic 238 is configured to perform functions related to compression and decompression of data.

The high speed layer 2-7 integrated packet engine 240 includes a packet processing timer 242. In one embodiment, the packet processing timer 242 provides one or more time intervals to trigger the processing of incoming, i.e., received, or outgoing, i.e., transmitted, network packets. In some embodiments, the high speed layer 2-7 integrated packet engine 240 processes network packets responsive to the timer 242. The packet processing timer 242 provides any type and form of signal to the packet engine 240 to notify, trigger, or communicate a time related event, interval or occurrence. In many embodiments, the packet processing timer 242 operates in the order of milliseconds, such as for example 100ms, 50ms or 25ms. For example, in some embodiments, the packet processing timer 242 provides time intervals or otherwise causes a network packet to be processed by the high speed layer 2-7 integrated packet engine 240 at a 10 ms time interval, while in other embodiments, at a 5 ms time interval, and still yet in further embodiments, as short as a 3, 2, or 1 ms time interval. The high speed layer 2-7 integrated packet engine 240 may be interfaced, integrated or in communication with the encryption engine 234, cache manager 232, policy engine 236 and multi-protocol compression engine 238 during operation. As such, any of the logic, functions, or operations of the encryption engine 234, cache manager 232, policy engine 236 and multi-protocol compression logic 238 may be performed responsive to the packet processing timer 242 and/or the packet engine 240. Therefore, any of the logic, functions, or operations of the encryption engine 234, cache manager 232, policy engine 236 and multi-protocol compression logic 238 may be performed at the granularity of time intervals provided via the packet processing timer 242, for example, at a time interval of less than or equal to 10ms. For example, in one embodiment, the cache manager 232 may perform invalidation of any cached objects responsive to the high speed layer 2-7 integrated packet engine 240 and/or the packet processing timer 242. In another embodiment, the expiry or invalidation time of a cached object can be set to the same order of granularity as the time interval of the packet processing timer 242, such as at every 10 ms.

In contrast to kernel space 204, user space 202 is the memory area or portion of the operating system used by user mode applications or programs otherwise running in user mode. A user mode application may not access kernel space 204 directly and uses service calls in order to access kernel services. As shown in FIG. 2, user space 202 of appliance 200 includes a graphical user interface (GUI) 210, a command line interface (CLI) 212, shell services 214, health monitoring program 216, and daemon services 218. GUI 210 and CLI

212 provide a means by which a system administrator or other user can interact with and control the operation of appliance 200, such as via the operating system of the appliance 200. The GUI 210 or CLI 212 can comprise code running in user space 202 or kernel space 204. The GUI 210 may be any type and form of graphical user interface and may be presented via text, graphical or otherwise, by any type of program or application, such as a browser. The CLI 212 may be any type and form of command line or text-based interface, such as a command line provided by the operating system. For example, the CLI 212 may comprise a shell, which is a tool to enable users to interact with the operating system. In some embodiments, the CLI 212 may be provided via a bash, csh, tcsh, or ksh type shell. The shell services 214 comprises the programs, services, tasks, processes or executable instructions to support interaction with the appliance 200 or operating system by a user via the GUI 210 and/or CLI 212.

Health monitoring program 216 is used to monitor, check, report and ensure that network systems are functioning properly and that users are receiving requested content over a network. Health monitoring program 216 comprises one or more programs, services, tasks, processes or executable instructions to provide logic, rules, functions or operations for monitoring any activity of the appliance 200. In some embodiments, the health monitoring program 216 intercepts and inspects any network traffic passed via the appliance 200. In other embodiments, the health monitoring program 216 interfaces by any suitable means and/or mechanisms with one or more of the following: the encryption engine 234, cache manager 232, policy engine 236, multi-protocol compression logic 238, packet engine 240, daemon services 218, and shell services 214. As such, the health monitoring program 216 may call any application programming interface (API) to determine a state, status, or health of any portion of the appliance 200. For example, the health monitoring program 216 may ping or send a status inquiry on a periodic basis to check if a program, process, service or task is active and currently running. In another example, the health monitoring program 216 may check any status, error or history logs provided by any program, process, service or task to determine any condition, status or error with any portion of the appliance 200.

Daemon services 218 are programs that run continuously or in the background and handle periodic service requests received by appliance 200. In some embodiments, a daemon service may forward the requests to other programs or processes, such as another daemon service 218 as appropriate. As known to those skilled in the art, a daemon service 218 may run unattended to perform continuous or periodic system wide functions, such as network control, or to perform any desired task. In some embodiments, one or more daemon services

218 run in the user space 202, while in other embodiments, one or more daemon services 218 run in the kernel space.

Referring now to FIG. 2B, another embodiment of the appliance 200 is depicted. In brief overview, the appliance 200 provides one or more of the following services, functionality or operations: SSL VPN connectivity 280, switching/load balancing 284, Domain Name Service resolution 286, acceleration 288 and an application firewall 290 for communications between one or more clients 102 and one or more servers 106. Each of the servers 106 may provide one or more network related services 270a-270n (referred to as services 270). For example, a server 106 may provide an http service 270. The appliance 200 comprises one or more virtual servers or virtual internet protocol servers, referred to as a vServer, VIP server, or just VIP 275a-275n (also referred herein as vServer 275). The vServer 275 receives, intercepts or otherwise processes communications between a client 102 and a server 106 in accordance with the configuration and operations of the appliance 200.

The vServer 275 may comprise software, hardware or any combination of software and hardware. The vServer 275 may comprise any type and form of program, service, task, process or executable instructions operating in user mode 202, kernel mode 204 or any combination thereof in the appliance 200. The vServer 275 includes any logic, functions, rules, or operations to perform any embodiments of the techniques described herein, such as SSL VPN 280, switching/load balancing 284, Domain Name Service resolution 286, acceleration 288 and an application firewall 290. In some embodiments, the vServer 275 establishes a connection to a service 270 of a server 106. The service 275 may comprise any program, application, process, task or set of executable instructions capable of connecting to and communicating to the appliance 200, client 102 or vServer 275. For example, the service 275 may comprise a web server, http server, ftp, email or database server. In some embodiments, the service 270 is a daemon process or network driver for listening, receiving and/or sending communications for an application, such as email, database or an enterprise application. In some embodiments, the service 270 may communicate on a specific IP address, or IP address and port.

In some embodiments, the vServer 275 applies one or more policies of the policy engine 236 to network communications between the client 102 and server 106. In one embodiment, the policies are associated with a vServer 275. In another embodiment, the policies are based on a user, or a group of users. In yet another embodiment, a policy is global and applies to one or more vServers 275a-275n, and any user or group of users communicating via the appliance 200. In some embodiments, the policies of the policy

engine have conditions upon which the policy is applied based on any content of the communication, such as internet protocol address, port, protocol type, header or fields in a packet, or the context of the communication, such as user, group of the user, vServer 275, transport layer connection, and/or identification or attributes of the client 102 or server 106.

In other embodiments, the appliance 200 communicates or interfaces with the policy engine 236 to determine authentication and/or authorization of a remote user or a remote client 102 to access the computing environment 15, application, and/or data file from a server 106. In another embodiment, the appliance 200 communicates or interfaces with the policy engine 236 to determine authentication and/or authorization of a remote user or a remote client 102 to have the application delivery system 190 deliver one or more of the computing environment 15, application, and/or data file. In yet another embodiment, the appliance 200 establishes a VPN or SSL VPN connection based on the policy engine's 236 authentication and/or authorization of a remote user or a remote client 102 In one embodiment, the appliance 200 controls the flow of network traffic and communication sessions based on policies of the policy engine 236. For example, the appliance 200 may control the access to a computing environment 15, application or data file based on the policy engine 236.

In some embodiments, the vServer 275 establishes a transport layer connection, such as a TCP or UDP connection with a client 102 via the client agent 120. In one embodiment, the vServer 275 listens for and receives communications from the client 102. In other embodiments, the vServer 275 establishes a transport layer connection, such as a TCP or UDP connection with a client server 106. In one embodiment, the vServer 275 establishes the transport layer connection to an internet protocol address and port of a server 270 running on the server 106. In another embodiment, the vServer 275 associates a first transport layer connection to a client 102 with a second transport layer connection to the server 106. In some embodiments, a vServer 275 establishes a pool of transport layer connections to a server 106 and multiplexes client requests via the pooled transport layer connections.

In some embodiments, the appliance 200 provides a SSL VPN connection 280 between a client 102 and a server 106. For example, a client 102 on a first network 102 requests to establish a connection to a server 106 on a second network 104'. In some embodiments, the second network 104' is not routable from the first network 104. In other embodiments, the client 102 is on a public network 104 and the server 106 is on a private network 104', such as a corporate network. In one embodiment, the client agent 120 intercepts communications of the client 102 on the first network 104, encrypts the communications, and transmits the communications via a first transport layer connection to

the appliance 200. The appliance 200 associates the first transport layer connection on the first network 104 to a second transport layer connection to the server 106 on the second network 104. The appliance 200 receives the intercepted communication from the client agent 102, decrypts the communications, and transmits the communication to the server 106 on the second network 104 via the second transport layer connection. The second transport layer connection may be a pooled transport layer connection. As such, the appliance 200 provides an end-to-end secure transport layer connection for the client 102 between the two networks 104, 104'.

In one embodiment, the appliance 200 hosts an intranet internet protocol or IntranetIP 282 address of the client 102 on the virtual private network 104. The client 102 has a local network identifier, such as an internet protocol (IP) address and/or host name on the first network 104. When connected to the second network 104' via the appliance 200, the appliance 200 establishes, assigns or otherwise provides an IntranetIP address 282, which is a network identifier, such as IP address and/or host name, for the client 102 on the second network 104'. The appliance 200 listens for and receives on the second or private network 104' for any communications directed towards the client 102 using the client's established IntranetIP 282. In one embodiment, the appliance 200 acts as or on behalf of the client 102 on the second private network 104. For example, in another embodiment, a vServer 275 listens for and responds to communications to the IntranetIP 282 of the client 102. In some embodiments, if a computing device 100 on the second network 104' transmits a request, the appliance 200 processes the request as if it were the client 102. For example, the appliance 200 may respond to a ping to the client's IntranetIP 282. In another example, the appliance may establish a connection, such as a TCP or UDP connection, with computing device 100 on the second network 104 requesting a connection with the client's IntranetIP 282.

In some embodiments, the appliance 200 provides one or more of the following acceleration techniques 288 to communications between the client 102 and server 106: 1) compression; 2) decompression; 3) Transmission Control Protocol pooling; 4) Transmission Control Protocol multiplexing; 5) Transmission Control Protocol buffering; and 6) caching. In one embodiment, the appliance 200 relieves servers 106 of much of the processing load caused by repeatedly opening and closing transport layers connections to clients 102 by opening one or more transport layer connections with each server 106 and maintaining these connections to allow repeated data accesses by clients via the Internet. This technique is referred to herein as "connection pooling".

In some embodiments, in order to seamlessly splice communications from a client 102 to a server 106 via a pooled transport layer connection, the appliance 200 translates or multiplexes communications by modifying sequence number and acknowledgment numbers at the transport layer protocol level. This is referred to as "connection multiplexing". In some embodiments, no application layer protocol interaction is required. For example, in the case of an in-bound packet (that is, a packet received from a client 102), the source network address of the packet is changed to that of an output port of appliance 200, and the destination network address is changed to that of the intended server. In the case of an outbound packet (that is, one received from a server 106), the source network address is changed from that of the server 106 to that of an output port of appliance 200 and the destination address is changed from that of appliance 200 to that of the requesting client 102. The sequence numbers and acknowledgment numbers of the packet are also translated to sequence numbers and acknowledgement numbers expected by the client 102 on the appliance's 200 transport layer connection to the client 102. In some embodiments, the packet checksum of the transport layer protocol is recalculated to account for these translations.

In another embodiment, the appliance 200 provides switching or load-balancing functionality 284 for communications between the client 102 and server 106. In some embodiments, the appliance 200 distributes traffic and directs client requests to a server 106 based on layer 4 or application-layer request data. In one embodiment, although the network layer or layer 2 of the network packet identifies a destination server 106, the appliance 200 determines the server 106 to distribute the network packet by application information and data carried as payload of the transport layer packet. In one embodiment, the health monitoring programs 216 of the appliance 200 monitor the health of servers to determine the server 106 for which to distribute a client's request. In some embodiments, if the appliance 200 detects a server 106 is not available or has a load over a predetermined threshold, the appliance 200 can direct or distribute client requests to another server 106.

In some embodiments, the appliance 200 acts as a Domain Name Service (DNS) resolver or otherwise provides resolution of a DNS request from clients 102. In some embodiments, the appliance intercepts a DNS request transmitted by the client 102. In one embodiment, the appliance 200 responds to a client's DNS request with an IP address of or hosted by the appliance 200. In this embodiment, the client 102 transmits network communication for the domain name to the appliance 200. In another embodiment, the appliance 200 responds to a client's DNS request with an IP address of or hosted by a second

appliance 200'. In some embodiments, the appliance 200 responds to a client's DNS request with an IP address of a server 106 determined by the appliance 200.

In yet another embodiment, the appliance 200 provides application firewall functionality 290 for communications between the client 102 and server 106. In one embodiment, the policy engine 236 provides rules for detecting and blocking illegitimate requests. In some embodiments, the application firewall 290 protects against denial of service (DoS) attacks. In other embodiments, the appliance inspects the content of intercepted requests to identify and block application-based attacks. In some embodiments, the rules/policy engine 236 comprises one or more application firewall or security control policies for providing protections against various classes and types of web or Internet based vulnerabilities, such as one or more of the following: 1) buffer overflow, 2) CGI-BIN parameter manipulation, 3) form/hidden field manipulation, 4) forceful browsing, 5) cookie or session poisoning, 6) broken access control list (ACLs) or weak passwords, 7) cross-site scripting (XSS), 8) command injection, 9) SQL injection, 10) error triggering sensitive information leak, 11) insecure use of cryptography, 12) server misconfiguration, 13) back doors and debug options, 14) website defacement, 15) platform or operating systems vulnerabilities, and 16) zero-day exploits. In an embodiment, the application firewall 290 provides HTML form field protection in the form of inspecting or analyzing the network communication for one or more of the following: 1) required fields are returned, 2) no added field allowed, 3) read-only and hidden field enforcement, 4) drop-down list and radio button field conformance, and 5) form-field max-length enforcement. In some embodiments, the application firewall 290 ensures cookies are not modified. In other embodiments, the application firewall 290 protects against forceful browsing by enforcing legal URLs.

In still yet other embodiments, the application firewall 290 protects any confidential information contained in the network communication. The application firewall 290 may inspect or analyze any network communication in accordance with the rules or polices of the engine 236 to identify any confidential information in any field of the network packet. In some embodiments, the application firewall 290 identifies in the network communication one or more occurrences of a credit card number, password, social security number, name, patient code, contact information, and age. The encoded portion of the network communication may comprise these occurrences or the confidential information. Based on these occurrences, in one embodiment, the application firewall 290 may take a policy action on the network communication, such as prevent transmission of the network communication. In another

embodiment, the application firewall 290 may rewrite, remove or otherwise mask such identified occurrence or confidential information.

Still referring to FIG. 2B, the appliance 200 may include a performance monitoring agent 197 as discussed above in conjunction with FIG. ID. In one embodiment, the appliance 200 receives the monitoring agent 197 from the monitoring service 198 or monitoring server 106 as depicted in FIG. ID. In some embodiments, the appliance 200 stores the monitoring agent 197 in storage, such as disk, for delivery to any client or server in communication with the appliance 200. For example, in one embodiment, the appliance 200 transmits the monitoring agent 197 to a client upon receiving a request to establish a transport layer connection. In other embodiments, the appliance 200 transmits the monitoring agent 197 upon establishing the transport layer connection with the client 102. In another embodiment, the appliance 200 transmits the monitoring agent 197 to the client upon intercepting or detecting a request for a web page. In yet another embodiment, the appliance 200 transmits the monitoring agent 197 to a client or a server in response to a request from the monitoring server 198. In one embodiment, the appliance 200 transmits the monitoring agent 197 to a second appliance 200' or appliance 205.

In other embodiments, the appliance 200 executes the monitoring agent 197. In one embodiment, the monitoring agent 197 measures and monitors the performance of any application, program, process, service, task or thread executing on the appliance 200. For example, the monitoring agent 197 may monitor and measure performance and operation of vServers 275A-275N. In another embodiment, the monitoring agent 197 measures and monitors the performance of any transport layer connections of the appliance 200. In some embodiments, the monitoring agent 197 measures and monitors the performance of any user sessions traversing the appliance 200. In one embodiment, the monitoring agent 197 measures and monitors the performance of any virtual private network connections and/or sessions traversing the appliance 200, such an SSL VPN session. In still further embodiments, the monitoring agent 197 measures and monitors the memory, CPU and disk usage and performance of the appliance 200. In yet another embodiment, the monitoring agent 197 measures and monitors the performance of any acceleration technique 288 performed by the appliance 200, such as SSL offloading, connection pooling and multiplexing, caching, and compression. In some embodiments, the monitoring agent 197 measures and monitors the performance of any load balancing and/or content switching 284 performed by the appliance 200. In other embodiments, the monitoring agent 197 measures

and monitors the performance of application firewall 290 protection and processing performed by the appliance 200.

C. Client Agent

Referring now to FIG. 3, an embodiment of the client agent 120 is depicted. The client 102 includes a client agent 120 for establishing and exchanging communications with the appliance 200 and/or server 106 via a network 104. In brief overview, the client 102 operates on computing device 100 having an operating system with a kernel mode 302 and a user mode 303, and a network stack 310 with one or more layers 310a-310b. The client 102 may have installed and/or execute one or more applications. In some embodiments, one or more applications may communicate via the network stack 310 to a network 104. One of the applications, such as a web browser, may also include a first program 322. For example, the first program 322 may be used in some embodiments to install and/or execute the client agent 120, or any portion thereof. The client agent 120 includes an interception mechanism, or interceptor 350, for intercepting network communications from the network stack 310 from the one or more applications.

The network stack 310 of the client 102 may comprise any type and form of software, or hardware, or any combinations thereof, for providing connectivity to and communications with a network. In one embodiment, the network stack 310 comprises a software implementation for a network protocol suite. The network stack 310 may comprise one or more network layers, such as any networks layers of the Open Systems Interconnection (OSI) communications model as those skilled in the art recognize and appreciate. As such, the network stack 310 may comprise any type and form of protocols for any of the following layers of the OSI model: 1) physical link layer, 2) data link layer, 3) network layer, 4) transport layer, 5) session layer, 6) presentation layer, and 7) application layer. In one embodiment, the network stack 310 may comprise a transport control protocol (TCP) over the network layer protocol of the internet protocol (IP), generally referred to as TCP/IP. In some embodiments, the TCP/IP protocol may be carried over the Ethernet protocol, which may comprise any of the family of IEEE wide-area-network (WAN) or local-area-network (LAN) protocols, such as those protocols covered by the IEEE 802.3. In some embodiments, the network stack 310 comprises any type and form of a wireless protocol, such as IEEE 802.11 and/or mobile internet protocol.

In view of a TCP/IP based network, any TCP/IP based protocol may be used, including Messaging Application Programming Interface (MAPI) (email), File Transfer

Protocol (FTP), HyperText Transfer Protocol (HTTP), Common Internet File System (CIFS) protocol (file transfer), Independent Computing Architecture (ICA) protocol, Remote Desktop Protocol (RDP), Wireless Application Protocol (WAP), Mobile IP protocol, and Voice Over IP (VoIP) protocol. In another embodiment, the network stack 310 comprises any type and form of transport control protocol, such as a modified transport control protocol, for example a Transaction TCP (T/TCP), TCP with selection acknowledgements (TCP-SACK), TCP with large windows (TCP-LW), a congestion prediction protocol such as the TCP-Vegas protocol, and a TCP spoofing protocol. In other embodiments, any type and form of user datagram protocol (UDP), such as UDP over IP, may be used by the network stack 310, such as for voice communications or real-time data communications.

Furthermore, the network stack 310 may include one or more network drivers supporting the one or more layers, such as a TCP driver or a network layer driver. The network drivers may be included as part of the operating system of the computing device 100 or as part of any network interface cards or other network access components of the computing device 100. In some embodiments, any of the network drivers of the network stack 310 may be customized, modified or adapted to provide a custom or modified portion of the network stack 310 in support of any of the techniques described herein. In other embodiments, the acceleration program 302 is designed and constructed to operate with or work in conjunction with the network stack 310 installed or otherwise provided by the operating system of the client 102.

The network stack 310 comprises any type and form of interfaces for receiving, obtaining, providing or otherwise accessing any information and data related to network communications of the client 102. In one embodiment, an interface to the network stack 310 comprises an application programming interface (API). The interface may also comprise any function call, hooking or filtering mechanism, event or call back mechanism, or any type of interfacing technique. The network stack 310 via the interface may receive or provide any type and form of data structure, such as an object, related to functionality or operation of the network stack 310. For example, the data structure may comprise information and data related to a network packet or one or more network packets. In some embodiments, the data structure comprises a portion of the network packet processed at a protocol layer of the network stack 310, such as a network packet of the transport layer. In some embodiments, the data structure 325 comprises a kernel-level data structure, while in other embodiments, the data structure 325 comprises a user-mode data structure. A kernel-level data structure may comprise a data structure obtained or related to a portion of the network stack 310

operating in kernel-mode 302, or a network driver or other software running in kernel-mode 302, or any data structure obtained or received by a service, process, task, thread or other executable instructions running or operating in kernel-mode of the operating system.

Additionally, some portions of the network stack 310 may execute or operate in kernel-mode 302, for example, the data link or network layer, while other portions execute or operate in user-mode 303, such as an application layer of the network stack 310. For example, a first portion 310a of the network stack may provide user-mode access to the network stack 310 to an application while a second portion 310a of the network stack 310 provides access to a network. In some embodiments, a first portion 310a of the network stack may comprise one or more upper layers of the network stack 310, such as any of layers 5-7. In other embodiments, a second portion 310b of the network stack 310 comprises one or more lower layers, such as any of layers 1-4. Each of the first portion 310a and second portion 310b of the network stack 310 may comprise any portion of the network stack 310, at any one or more network layers, in user-mode 203, kernel-mode, 202, or combinations thereof, or at any portion of a network layer or interface point to a network layer or any portion of or interface point to the user-mode 203 and kernel-mode 203. .

The interceptor 350 may comprise software, hardware, or any combination of software and hardware. In one embodiment, the interceptor 350 intercept a network communication at any point in the network stack 310, and redirects or transmits the network communication to a destination desired, managed or controlled by the interceptor 350 or client agent 120. For example, the interceptor 350 may intercept a network communication of a network stack 310 of a first network and transmit the network communication to the appliance 200 for transmission on a second network 104. In some embodiments, the interceptor 350 comprises any type interceptor 350 comprises a driver, such as a network driver constructed and designed to interface and work with the network stack 310. In some embodiments, the client agent 120 and/or interceptor 350 operates at one or more layers of the network stack 310, such as at the transport layer. In one embodiment, the interceptor 350 comprises a filter driver, hooking mechanism, or any form and type of suitable network driver interface that interfaces to the transport layer of the network stack, such as via the transport driver interface (TDI). In some embodiments, the interceptor 350 interfaces to a first protocol layer, such as the transport layer and another protocol layer, such as any layer above the transport protocol layer, for example, an application protocol layer. In one embodiment, the interceptor 350 may comprise a driver complying with the Network Driver Interface Specification (NDIS), or a NDIS driver. In another embodiment, the interceptor

350 may comprise a mini-filter or a mini-port driver. In one embodiment, the interceptor 350, or portion thereof, operates in kernel-mode 202. In another embodiment, the interceptor 350, or portion thereof, operates in user-mode 203. In some embodiments, a portion of the interceptor 350 operates in kernel-mode 202 while another portion of the interceptor 350 operates in user-mode 203. In other embodiments, the client agent 120 operates in user-mode 203 but interfaces via the interceptor 350 to a kernel-mode driver, process, service, task or portion of the operating system, such as to obtain a kernel-level data structure 225. In further embodiments, the interceptor 350 is a user-mode application or program, such as application.

In one embodiment, the interceptor 350 intercepts any transport layer connection requests. In these embodiments, the interceptor 350 execute transport layer application programming interface (API) calls to set the destination information, such as destination IP address and/or port to a desired location for the location. In this manner, the interceptor 350 intercepts and redirects the transport layer connection to a IP address and port controlled or managed by the interceptor 350 or client agent 120. In one embodiment, the interceptor 350 sets the destination information for the connection to a local IP address and port of the client 102 on which the client agent 120 is listening. For example, the client agent 120 may comprise a proxy service listening on a local IP address and port for redirected transport layer communications. In some embodiments, the client agent 120 then communicates the redirected transport layer communication to the appliance 200.

In some embodiments, the interceptor 350 intercepts a Domain Name Service (DNS) request. In one embodiment, the client agent 120 and/or interceptor 350 resolves the DNS request. In another embodiment, the interceptor transmits the intercepted DNS request to the appliance 200 for DNS resolution. In one embodiment, the appliance 200 resolves the DNS request and communicates the DNS response to the client agent 120. In some embodiments, the appliance 200 resolves the DNS request via another appliance 200' or a DNS server 106.

In yet another embodiment, the client agent 120 may comprise two agents 120 and 120'. In one embodiment, a first agent 120 may comprise an interceptor 350 operating at the network layer of the network stack 310. In some embodiments, the first agent 120 intercepts network layer requests such as Internet Control Message Protocol (ICMP) requests (e.g., ping and traceroute). In other embodiments, the second agent 120' may operate at the transport layer and intercept transport layer communications. In some embodiments, the first agent 120 intercepts communications at one layer of the network stack 210 and interfaces with or communicates the intercepted communication to the second agent 120'.

The client agent 120 and/or interceptor 350 may operate at or interface with a protocol layer in a manner transparent to any other protocol layer of the network stack 310. For example, in one embodiment, the interceptor 350 operates or interfaces with the transport layer of the network stack 310 transparently to any protocol layer below the transport layer, such as the network layer, and any protocol layer above the transport layer, such as the session, presentation or application layer protocols. This allows the other protocol layers of the network stack 310 to operate as desired and without modification for using the interceptor 350. As such, the client agent 120 and/or interceptor 350 can interface with the transport layer to secure, optimize, accelerate, route or load-balance any communications provided via any protocol carried by the transport layer, such as any application layer protocol over TCP/IP.

Furthermore, the client agent 120 and/or interceptor may operate at or interface with the network stack 310 in a manner transparent to any application, a user of the client 102, and any other computing device, such as a server, in communications with the client 102. The client agent 120 and/or interceptor 350 may be installed and/or executed on the client 102 in a manner without modification of an application. In some embodiments, the user of the client 102 or a computing device in communications with the client 102 are not aware of the existence, execution or operation of the client agent 120 and/or interceptor 350. As such, in some embodiments, the client agent 120 and/or interceptor 350 is installed, executed, and/or operated transparently to an application, user of the client 102, another computing device, such as a server, or any of the protocol layers above and/or below the protocol layer interfaced to by the interceptor 350.

The client agent 120 includes an acceleration program 302, a streaming client 306, a collection agent 304, and/or monitoring agent 197. In one embodiment, the client agent 120 comprises an Independent Computing Architecture (ICA) client, or any portion thereof, developed by Citrix Systems, Inc. of Fort Lauderdale, Florida, and is also referred to as an ICA client. In some embodiments, the client 120 comprises an application streaming client 306 for streaming an application from a server 106 to a client 102. In some embodiments, the client agent 120 comprises an acceleration program 302 for accelerating communications between client 102 and server 106. In another embodiment, the client agent 120 includes a collection agent 304 for performing end-point detection/scanning and collecting end-point information for the appliance 200 and/or server 106.

In some embodiments, the acceleration program 302 comprises a client-side acceleration program for performing one or more acceleration techniques to accelerate,

enhance or otherwise improve a client's communications with and/or access to a server 106, such as accessing an application provided by a server 106. The logic, functions, and/or operations of the executable instructions of the acceleration program 302 may perform one or more of the following acceleration techniques: 1) multi-protocol compression, 2) transport control protocol pooling, 3) transport control protocol multiplexing, 4) transport control protocol buffering, and 5) caching via a cache manager. Additionally, the acceleration program 302 may perform encryption and/or decryption of any communications received and/or transmitted by the client 102. In some embodiments, the acceleration program 302 performs one or more of the acceleration techniques in an integrated manner or fashion. Additionally, the acceleration program 302 can perform compression on any of the protocols, or multiple-protocols, carried as a payload of a network packet of the transport layer protocol.

The streaming client 306 comprises an application, program, process, service, task or executable instructions for receiving and executing a streamed application from a server 106. A server 106 may stream one or more application data files to the streaming client 306 for playing, executing or otherwise causing to be executed the application on the client 102. In some embodiments, the server 106 transmits a set of compressed or packaged application data files to the streaming client 306. In some embodiments, the plurality of application files are compressed and stored on a file server within an archive file such as a CAB, ZIP, SIT, TAR, JAR or other archive. In one embodiment, the server 106 decompresses, unpackages or unarchives the application files and transmits the files to the client 102. In another embodiment, the client 102 decompresses, unpackages or unarchives the application files. The streaming client 306 dynamically installs the application, or portion thereof, and executes the application. In one embodiment, the streaming client 306 may be an executable program. In some embodiments, the streaming client 306 may be able to launch another executable program.

The collection agent 304 comprises an application, program, process, service, task or executable instructions for identifying, obtaining and/or collecting information about the client 102. In some embodiments, the appliance 200 transmits the collection agent 304 to the client 102 or client agent 120. The collection agent 304 may be configured according to one or more policies of the policy engine 236 of the appliance. In other embodiments, the collection agent 304 transmits collected information on the client 102 to the appliance 200. In one embodiment, the policy engine 236 of the appliance 200 uses the collected information to determine and provide access, authentication and authorization control of the client's connection to a network 104.

In one embodiment, the collection agent 304 comprises an end-point detection and scanning mechanism, which identifies and determines one or more attributes or characteristics of the client. For example, the collection agent 304 may identify and determine any one or more of the following client-side attributes: 1) the operating system an/or a version of an operating system, 2) a service pack of the operating system, 3) a running service, 4) a running process, and 5) a file. The collection agent 304 may also identify and determine the presence or versions of any one or more of the following on the client: 1) antivirus software, 2) personal firewall software, 3) anti-spam software, and 4) internet security software. The policy engine 236 may have one or more policies based on any one or more of the attributes or characteristics of the client or client-side attributes.

In some embodiments, the client agent 120 includes a monitoring agent 197 as discussed in conjunction with FIGs. ID and 2B. The monitoring agent 197 may be any type and form of script, such as Visual Basic or Java script. In one embodiment, the monitoring agent 197 monitors and measures performance of any portion of the client agent 120. For example, in some embodiments, the monitoring agent 197 monitors and measures performance of the acceleration program 302. In another embodiment, the monitoring agent 197 monitors and measures performance of the streaming client 306. In other embodiments, the monitoring agent 197 monitors and measures performance of the collection agent 304. In still another embodiment, the monitoring agent 197 monitors and measures performance of the interceptor 350. In some embodiments, the monitoring agent 197 monitors and measures any resource of the client 102, such as memory, CPU and disk.

The monitoring agent 197 may monitor and measure performance of any application of the client. In one embodiment, the monitoring agent 197 monitors and measures performance of a browser on the client 102. In some embodiments, the monitoring agent 197 monitors and measures performance of any application delivered via the client agent 120. In other embodiments, the monitoring agent 197 measures and monitors end user response times for an application, such as web-based or HTTP response times. The monitoring agent 197 may monitor and measure performance of an ICA or RDP client. In another embodiment, the monitoring agent 197 measures and monitors metrics for a user session or application session. In some embodiments, monitoring agent 197 measures and monitors an ICA or RDP session. In one embodiment, the monitoring agent 197 measures and monitors the performance of the appliance 200 in accelerating delivery of an application and/or data to the client 102.

In some embodiments and still referring to FIG. 3, a first program 322 may be used to install and/or execute the client agent 120, or portion thereof, such as the interceptor 350,

automatically, silently, transparently, or otherwise. In one embodiment, the first program 322 comprises a plugin component, such an ActiveX control or Java control or script that is loaded into and executed by an application. For example, the first program comprises an ActiveX control loaded and run by a web browser application, such as in the memory space or context of the application. In another embodiment, the first program 322 comprises a set of executable instructions loaded into and run by the application, such as a browser. In one embodiment, the first program 322 comprises a designed and constructed program to install the client agent 120. In some embodiments, the first program 322 obtains, downloads, or receives the client agent 120 via the network from another computing device. In another embodiment, the first program 322 is an installer program or a plug and play manager for installing programs, such as network drivers, on the operating system of the client 102.

D. Systems and Methods for Providing Virtualized Application Delivery Controller

Referring now to FIG. 4A, a block diagram depicts one embodiment of a virtualization environment 400. In brief overview, a computing device 100 includes a hypervisor layer, a virtualization layer, and a hardware layer. The hypervisor layer includes a hypervisor 401 (also referred to as a virtualization manager) that allocates and manages access to a number of physical resources in the hardware layer (e.g., the processor(s) 421, and disk(s) 428) by at least one virtual machine executing in the virtualization layer. The virtualization layer includes at least one operating system 410 and a plurality of virtual resources allocated to the at least one operating system 410. Virtual resources may include, without limitation, a plurality of virtual processors 432a, 432b, 432c (generally 432), and virtual disks 442a, 442b, 442c (generally 442), as well as virtual resources such as virtual memory and virtual network interfaces. The plurality of virtual resources and the operating system 410 may be referred to as a virtual machine 406. A virtual machine 406 may include a control operating system 405 in communication with the hypervisor 401 and used to execute applications for managing and configuring other virtual machines on the computing device 100.

In greater detail, a hypervisor 401 may provide virtual resources to an operating system in any manner which simulates the operating system having access to a physical device. A hypervisor 401 may provide virtual resources to any number of guest operating systems 410a, 410b (generally 410). In some embodiments, a computing device 100 executes one or more types of hypervisors. In these embodiments, hypervisors may be used to emulate virtual hardware, partition physical hardware, virtualize physical hardware, and execute

virtual machines that provide access to computing environments. Hypervisors may include those manufactured by VMWare, Inc., of Palo Alto, California; the XEN hypervisor, an open source product whose development is overseen by the open source Xen.org community; HyperV, VirtualServer or virtual PC hypervisors provided by Microsoft, or others. In some embodiments, a computing device 100 executing a hypervisor that creates a virtual machine platform on which guest operating systems may execute is referred to as a host server. In one of these embodiments, for example, the computing device 100 is a XEN SERVER provided by Citrix Systems, Inc., of Fort Lauderdale, FL.

In some embodiments, a hypervisor 401 executes within an operating system executing on a computing device. In one of these embodiments, a computing device executing an operating system and a hypervisor 401 may be said to have a host operating system (the operating system executing on the computing device), and a guest operating system (an operating system executing within a computing resource partition provided by the hypervisor 401). In other embodiments, a hypervisor 401 interacts directly with hardware on a computing device, instead of executing on a host operating system. In one of these embodiments, the hypervisor 401 may be said to be executing on "bare metal," referring to the hardware comprising the computing device.

In some embodiments, a hypervisor 401 may create a virtual machine 406a-c (generally 406) in which an operating system 410 executes. In one of these embodiments, for example, the hypervisor 401 loads a virtual machine image to create a virtual machine 406. In another of these embodiments, the hypervisor 401 executes an operating system 410 within the virtual machine 406. In still another of these embodiments, the virtual machine 406 executes an operating system 410.

In some embodiments, the hypervisor 401 controls processor scheduling and memory partitioning for a virtual machine 406 executing on the computing device 100. In one of these embodiments, the hypervisor 401 controls the execution of at least one virtual machine 406. In another of these embodiments, the hypervisor 401 presents at least one virtual machine 406 with an abstraction of at least one hardware resource provided by the computing device 100. In other embodiments, the hypervisor 401 controls whether and how physical processor capabilities are presented to the virtual machine 406.

A control operating system 405 may execute at least one application for managing and configuring the guest operating systems. In one embodiment, the control operating system 405 may execute an administrative application, such as an application including a user interface providing administrators with access to functionality for managing the execution of

a virtual machine, including functionality for executing a virtual machine, terminating an execution of a virtual machine, or identifying a type of physical resource for allocation to the virtual machine. In another embodiment, the hypervisor 401 executes the control operating system 405 within a virtual machine 406 created by the hypervisor 401. In still another embodiment, the control operating system 405 executes in a virtual machine 406 that is authorized to directly access physical resources on the computing device 100. In some embodiments, a control operating system 405a on a computing device 100a may exchange data with a control operating system 405b on a computing device 100b, via communications between a hypervisor 401a and a hypervisor 401b. In this way, one or more computing devices 100 may exchange data with one or more of the other computing devices 100 regarding processors and other physical resources available in a pool of resources. In one of these embodiments, this functionality allows a hypervisor to manage a pool of resources distributed across a plurality of physical computing devices. In another of these embodiments, multiple hypervisors manage one or more of the guest operating systems executed on one of the computing devices 100.

In one embodiment, the control operating system 405 executes in a virtual machine 406 that is authorized to interact with at least one guest operating system 410. In another embodiment, a guest operating system 410 communicates with the control operating system 405 via the hypervisor 401 in order to request access to a disk or a network. In still another embodiment, the guest operating system 410 and the control operating system 405 may communicate via a communication channel established by the hypervisor 401, such as, for example, via a plurality of shared memory pages made available by the hypervisor 401.

In some embodiments, the control operating system 405 includes a network back-end driver for communicating directly with networking hardware provided by the computing device 100. In one of these embodiments, the network back-end driver processes at least one virtual machine request from at least one guest operating system 110. In other embodiments, the control operating system 405 includes a block back-end driver for communicating with a storage element on the computing device 100. In one of these embodiments, the block back-end driver reads and writes data from the storage element based upon at least one request received from a guest operating system 410.

In one embodiment, the control operating system 405 includes a tools stack 404. In another embodiment, a tools stack 404 provides functionality for interacting with the hypervisor 401, communicating with other control operating systems 405 (for example, on a second computing device 100b), or managing virtual machines 406b, 406c on the computing

device 100. In another embodiment, the tools stack 404 includes customized applications for providing improved management functionality to an administrator of a virtual machine farm. In some embodiments, at least one of the tools stack 404 and the control operating system 405 include a management API that provides an interface for remotely configuring and controlling virtual machines 406 running on a computing device 100. In other embodiments, the control operating system 405 communicates with the hypervisor 401 through the tools stack 404.

In one embodiment, the hypervisor 401 executes a guest operating system 410 within a virtual machine 406 created by the hypervisor 401. In another embodiment, the guest operating system 410 provides a user of the computing device 100 with access to resources within a computing environment. In still another embodiment, a resource includes a program, an application, a document, a file, a plurality of applications, a plurality of files, an executable program file, a desktop environment, a computing environment, or other resource made available to a user of the computing device 100. In yet another embodiment, the resource may be delivered to the computing device 100 via a plurality of access methods including, but not limited to, conventional installation directly on the computing device 100, delivery to the computing device 100 via a method for application streaming, delivery to the computing device 100 of output data generated by an execution of the resource on a second computing device 100' and communicated to the computing device 100 via a presentation layer protocol, delivery to the computing device 100 of output data generated by an execution of the resource via a virtual machine executing on a second computing device 100', or execution from a removable storage device connected to the computing device 100, such as a USB device, or via a virtual machine executing on the computing device 100 and generating output data. In some embodiments, the computing device 100 transmits output data generated by the execution of the resource to another computing device 100'.

In one embodiment, the guest operating system 410, in conjunction with the virtual machine on which it executes, forms a fully-virtualized virtual machine which is not aware that it is a virtual machine; such a machine may be referred to as a "Domain U HVM (Hardware Virtual Machine) virtual machine". In another embodiment, a fully-virtualized machine includes software emulating a Basic Input/Output System (BIOS) in order to execute an operating system within the fully-virtualized machine. In still another embodiment, a fully-virtualized machine may include a driver that provides functionality by communicating with the hypervisor 401. In such an embodiment, the driver may be aware that it executes within a virtualized environment. In another embodiment, the guest operating system 410, in

conjunction with the virtual machine on which it executes, forms a paravirtualized virtual machine, which is aware that it is a virtual machine; such a machine may be referred to as a "Domain U PV virtual machine". In another embodiment, a paravirtualized machine includes additional drivers that a fully-virtualized machine does not include. In still another embodiment, the paravirtualized machine includes the network back-end driver and the block back-end driver included in a control operating system 405, as described above.

Referring now to FIG. 4B, a block diagram depicts one embodiment of a plurality of networked computing devices in a system in which at least one physical host executes a virtual machine. In brief overview, the system includes a management component 404 and a hypervisor 401. The system includes a plurality of computing devices 100, a plurality of virtual machines 406, a plurality of hypervisors 401, a plurality of management components referred to variously as tools stacks 404 or management components 404, and a physical resource 421, 428. The plurality of physical machines 100 may each be provided as computing devices 100, described above in connection with FIGs. IE- IH and 4A.

In greater detail, a physical disk 428 is provided by a computing device 100 and stores at least a portion of a virtual disk 442. In some embodiments, a virtual disk 442 is associated with a plurality of physical disks 428. In one of these embodiments, one or more computing devices 100 may exchange data with one or more of the other computing devices 100 regarding processors and other physical resources available in a pool of resources, allowing a hypervisor to manage a pool of resources distributed across a plurality of physical computing devices. In some embodiments, a computing device 100 on which a virtual machine 406 executes is referred to as a physical host 100 or as a host machine 100.

The hypervisor executes on a processor on the computing device 100. The hypervisor allocates, to a virtual disk, an amount of access to the physical disk. In one embodiment, the hypervisor 401 allocates an amount of space on the physical disk. In another embodiment, the hypervisor 401 allocates a plurality of pages on the physical disk. In some embodiments, the hypervisor provisions the virtual disk 442 as part of a process of initializing and executing a virtual machine 450.

In one embodiment, the management component 404a is referred to as a pool management component 404a. In another embodiment, a management operating system 405a, which may be referred to as a control operating system 405a, includes the management component. In some embodiments, the management component is referred to as a tools stack. In one of these embodiments, the management component is the tools stack 404 described above in connection with FIG. 4A. In other embodiments, the management

component 404 provides a user interface for receiving, from a user such as an administrator, an identification of a virtual machine 406 to provision and/or execute. In still other embodiments, the management component 404 provides a user interface for receiving, from a user such as an administrator, the request for migration of a virtual machine 406b from one physical machine 100 to another. In further embodiments, the management component 404a identifies a computing device 100b on which to execute a requested virtual machine 406d and instructs the hypervisor 401b on the identified computing device 100b to execute the identified virtual machine; such a management component may be referred to as a pool management component.

Referring now to Figure 4C, embodiments of a virtual application delivery controller or virtual appliance 450 are depicted. In brief overview, any of the functionality and/or embodiments of the appliance 200 (e.g., an application delivery controller) described above in connection with FIGs. 2A and 2B may be deployed in any embodiment of the virtualized environment described above in connection with FIGs 4A and 4B. Instead of the functionality of the application delivery controller being deployed in the form of an appliance 200, such functionality may be deployed in a virtualized environment 400 on any computing device 100, such as a client 102, server 106 or appliance 200.

Referring now to FIG. 4C, a diagram of an embodiment of a virtual appliance 450 operating on a hypervisor 401 of a server 106 is depicted. As with the appliance 200 of FIGs. 2A and 2B, the virtual appliance 450 may provide functionality for availability, performance, offload and security. For availability, the virtual appliance may perform load balancing between layers 4 and 7 of the network and may also perform intelligent service health monitoring. For performance increases via network traffic acceleration, the virtual appliance may perform caching and compression. To offload processing of any servers, the virtual appliance may perform connection multiplexing and pooling and/or SSL processing. For security, the virtual appliance may perform any of the application firewall functionality and SSL VPN function of appliance 200.

Any of the modules of the appliance 200 as described in connection with FIGs. 2A may be packaged, combined, designed or constructed in a form of the virtualized appliance delivery controller 450 deployable as one or more software modules or components executable in a virtualized environment 300 or non-virtualized environment on any server, such as an off the shelf server. For example, the virtual appliance may be provided in the form of an installation package to install on a computing device. With reference to FIG. 2A, any of the cache manager 232, policy engine 236, compression 238, encryption engine 234,

packet engine 240, GUI 210, CLI 212, shell services 214 and health monitoring programs 216 may be designed and constructed as a software component or module to run on any operating system of a computing device and/or of a virtualized environment 300. Instead of using the encryption processor 260, processor 262, memory 264 and network stack 267 of the appliance 200, the virtualized appliance 400 may use any of these resources as provided by the virtualized environment 400 or as otherwise available on the server 106.

Still referring to FIG. 4C, and in brief overview, any one or more vServers 275A-275N may be in operation or executed in a virtualized environment 400 of any type of computing device 100, such as any server 106. Any of the modules or functionality of the appliance 200 described in connection with FIG. 2B may be designed and constructed to operate in either a virtualized or non-virtualized environment of a server. Any of the vServer 275, SSL VPN 280, Intranet UP 282, Switching 284, DNS 286, acceleration 288, App FW 280 and monitoring agent may be packaged, combined, designed or constructed in a form of application delivery controller 450 deployable as one or more software modules or components executable on a device and/or virtualized environment 400.

In some embodiments, a server may execute multiple virtual machines 406a-406n in the virtualization environment with each virtual machine running the same or different embodiments of the virtual application delivery controller 450. In some embodiments, the server may execute one or more virtual appliances 450 on one or more virtual machines on a core of a multi-core processing system. In some embodiments, the server may execute one or more virtual appliances 450 on one or more virtual machines on each processor of a multiple processor device.

E. Systems and Methods for Providing A Multi-Core Architecture

In accordance with Moore's Law, the number of transistors that may be placed on an integrated circuit may double approximately every two years. However, CPU speed increases may reach plateaus, for example CPU speed has been around 3.5 - 4 GHz range since 2005. In some cases, CPU manufacturers may not rely on CPU speed increases to gain additional performance. Some CPU manufacturers may add additional cores to their processors to provide additional performance. Products, such as those of software and networking vendors, that rely on CPUs for performance gains may improve their performance by leveraging these multi-core CPUs. The software designed and constructed for a single CPU may be redesigned and/or rewritten to take advantage of a multi-threaded, parallel architecture or otherwise a multi-core architecture.

A multi-core architecture of the appliance 200, referred to as nCore or multi-core technology, allows the appliance in some embodiments to break the single core performance barrier and to leverage the power of multi-core CPUs. In the previous architecture described in connection with FIG. 2A, a single network or packet engine is run. The multiple cores of the nCore technology and architecture allow multiple packet engines to run concurrently and/or in parallel. With a packet engine running on each core, the appliance architecture leverages the processing capacity of additional cores. In some embodiments, this provides up to a 7X increase in performance and scalability.

Illustrated in FIG. 5A are some embodiments of work, task, load or network traffic distribution across one or more processor cores according to a type of parallelism or parallel computing scheme, such as functional parallelism, data parallelism or flow-based data parallelism. In brief overview, FIG. 5A illustrates embodiments of a multi-core system such as an appliance 200' with n-cores, a total of cores numbers 1 through N. In one embodiment, work, load or network traffic can be distributed among a first core 505A, a second core 505B, a third core 505C, a fourth core 505D, a fifth core 505E, a sixth core 505F, a seventh core 505G, and so on such that distribution is across all or two or more of the n cores 505N (hereinafter referred to collectively as cores 505.) There may be multiple VIPs 275 each running on a respective core of the plurality of cores. There may be multiple packet engines 240 each running on a respective core of the plurality of cores. Any of the approaches used may lead to different, varying or similar work load or performance level 515 across any of the cores. For a functional parallelism approach, each core may run a different function of the functionalities provided by the packet engine, a VIP 275 or appliance 200. In a data parallelism approach, data may be paralleled or distributed across the cores based on the Network Interface Card (NIC) or VIP 275 receiving the data. In another data parallelism approach, processing may be distributed across the cores by distributing data flows to each core.

In further detail to FIG. 5A, in some embodiments, load, work or network traffic can be distributed among cores 505 according to functional parallelism 500. Functional parallelism may be based on each core performing one or more respective functions. In some embodiments, a first core may perform a first function while a second core performs a second function. In functional parallelism approach, the functions to be performed by the multi-core system are divided and distributed to each core according to functionality. In some embodiments, functional parallelism may be referred to as task parallelism and may be achieved when each processor or core executes a different process or function on the same or

different data. The core or processor may execute the same or different code. In some cases, different execution threads or code may communicate with one another as they work. Communication may take place to pass data from one thread to the next as part of a workflow.

In some embodiments, distributing work across the cores 505 according to functional parallelism 500, can comprise distributing network traffic according to a particular function such as network input/output management (NW I/O) 510A, secure sockets layer (SSL) encryption and decryption 510B and transmission control protocol (TCP) functions 510C. This may lead to a work, performance or computing load 515 based on a volume or level of functionality being used. In some embodiments, distributing work across the cores 505 according to data parallelism 540, can comprise distributing an amount of work 515 based on distributing data associated with a particular hardware or software component. In some embodiments, distributing work across the cores 505 according to flow-based data parallelism 520, can comprise distributing data based on a context or flow such that the amount of work 515A-N on each core may be similar, substantially equal or relatively evenly distributed.

In the case of the functional parallelism approach, each core may be configured to run one or more functionalities of the plurality of functionalities provided by the packet engine or VIP of the appliance. For example, core 1 may perform network I/O processing for the appliance 200' while core 2 performs TCP connection management for the appliance. Likewise, core 3 may perform SSL offloading while core 4 may perform layer 7 or application layer processing and traffic management. Each of the cores may perform the same function or different functions. Each of the cores may perform more than one function. Any of the cores may run any of the functionality or portions thereof identified and/or described in conjunction with FIGs. 2A and 2B. In this the approach, the work across the cores may be divided by function in either a coarse-grained or fine-grained manner. In some cases, as illustrated in FIG. 5A, division by function may lead to different cores running at different levels of performance or load 515.

In the case of the functional parallelism approach, each core may be configured to run one or more functionalities of the plurality of functionalities provided by the packet engine of the appliance. For example, core 1 may perform network I/O processing for the appliance 200' while core 2 performs TCP connection management for the appliance. Likewise, core 3 may perform SSL offloading while core 4 may perform layer 7 or application layer processing and traffic management. Each of the cores may perform the same function or

different functions. Each of the cores may perform more than one function. Any of the cores may run any of the functionality or portions thereof identified and/or described in conjunction with FIGs. 2A and 2B. In this the approach, the work across the cores may be divided by function in either a coarse-grained or fine-grained manner. In some cases, as illustrated in FIG. 5A division by function may lead to different cores running at different levels of load or performance.

The functionality or tasks may be distributed in any arrangement and scheme. For example, FIG. 5B illustrates a first core, Core 1 505A, processing applications and processes associated with network I/O functionality 510A. Network traffic associated with network I/O, in some embodiments, can be associated with a particular port number. Thus, outgoing and incoming packets having a port destination associated with NW I/O 510A will be directed towards Core 1 505A which is dedicated to handling all network traffic associated with the NW I/O port. Similarly, Core 2 505B is dedicated to handling functionality associated with SSL processing and Core 4 505D may be dedicated handling all TCP level processing and functionality.

While FIG. 5A illustrates functions such as network I/O, SSL and TCP, other functions can be assigned to cores. These other functions can include any one or more of the functions or operations described herein. For example, any of the functions described in conjunction with FIGs. 2A and 2B may be distributed across the cores on a functionality basis. In some cases, a first VIP 275A may run on a first core while a second VIP 275B with a different configuration may run on a second core. In some embodiments, each core 505 can handle a particular functionality such that each core 505 can handle the processing associated with that particular function. For example, Core 2 505B may handle SSL offloading while Core 4 505D may handle application layer processing and traffic management.

In other embodiments, work, load or network traffic may be distributed among cores 505 according to any type and form of data parallelism 540. In some embodiments, data parallelism may be achieved in a multi-core system by each core performing the same task or functionally on different pieces of distributed data. In some embodiments, a single execution thread or code controls operations on all pieces of data. In other embodiments, different threads or instructions control the operation, but may execute the same code. In some embodiments, data parallelism is achieved from the perspective of a packet engine, vServers (VIPs) 275A-C, network interface cards (NIC) 542D-E and/or any other networking hardware or software included on or associated with an appliance 200. For example, each core may run the same packet engine or VIP code or configuration but operate on different

sets of distributed data. Each networking hardware or software construct can receive different, varying or substantially the same amount of data, and as a result may have varying, different or relatively the same amount of load 515.

In the case of a data parallelism approach, the work may be divided up and distributed based on VIPs, NICs and/or data flows of the VIPs or NICs. In one of these approaches, the work of the multi-core system may be divided or distributed among the VIPs by having each VIP work on a distributed set of data. For example, each core may be configured to run one or more VIPs. Network traffic may be distributed to the core for each VIP handling that traffic. In another of these approaches, the work of the appliance may be divided or distributed among the cores based on which NIC receives the network traffic. For example, network traffic of a first NIC may be distributed to a first core while network traffic of a second NIC may be distributed to a second core. In some cases, a core may process data from multiple NICs.

While FIG 5A illustrates a single vServer associated with a single core 505, as is the case for VIPl 275A, VIP2 275B and VIP3 275C. In some embodiments, a single vServer can be associated with one or more cores 505. In contrast, one or more vServers can be associated with a single core 505. Associating a vServer with a core 505 may include that core 505 to process all functions associated with that particular vServer. In some embodiments, each core executes a VIP having the same code and configuration. In other embodiments, each core executes a VIP having the same code but different configuration. In some embodiments, each core executes a VIP having different code and the same or different configuration.

Like vServers, NICs can also be associated with particular cores 505. In many embodiments, NICs can be connected to one or more cores 505 such that when a NIC receives or transmits data packets, a particular core 505 handles the processing involved with receiving and transmitting the data packets. In one embodiment, a single NIC can be associated with a single core 505, as is the case with NICl 542D and NIC2 542E. In other embodiments, one or more NICs can be associated with a single core 505. In other embodiments, a single NIC can be associated with one or more cores 505. In these embodiments, load could be distributed amongst the one or more cores 505 such that each core 505 processes a substantially similar amount of load. A core 505 associated with a NIC may process all functions and/or data associated with that particular NIC.

While distributing work across cores based on data of VIPs or NICs may have a level of independency, in some embodiments, this may lead to unbalanced use of cores as illustrated by the varying loads 515 of FIG. 5A.

In some embodiments, load, work or network traffic can be distributed among cores 505 based on any type and form of data flow. In another of these approaches, the work may be divided or distributed among cores based on data flows. For example, network traffic between a client and a server traversing the appliance may be distributed to and processed by one core of the plurality of cores. In some cases, the core initially establishing the session or connection may be the core for which network traffic for that session or connection is distributed. In some embodiments, the data flow is based on any unit or portion of network traffic, such as a transaction, a request/response communication or traffic originating from an application on a client. In this manner and in some embodiments, data flows between clients and servers traversing the appliance 200' may be distributed in a more balanced manner than the other approaches.

In flow-based data parallelism 520, distribution of data is related to any type of flow of data, such as request/response pairings, transactions, sessions, connections or application communications. For example, network traffic between a client and a server traversing the appliance may be distributed to and processed by one core of the plurality of cores. In some cases, the core initially establishing the session or connection may be the core for which network traffic for that session or connection is distributed. The distribution of data flow may be such that each core 505 carries a substantially equal or relatively evenly distributed amount of load, data or network traffic.

In some embodiments, the data flow is based on any unit or portion of network traffic, such as a transaction, a request/response communication or traffic originating from an application on a client. In this manner and in some embodiments, data flows between clients and servers traversing the appliance 200' may be distributed in a more balanced manner than the other approached. In one embodiment, data flow can be distributed based on a transaction or a series of transactions. This transaction, in some embodiments, can be between a client and a server and can be characterized by an IP address or other packet identifier. For example, Core 1 505A can be dedicated to transactions between a particular client and a particular server, therefore the load 515A on Core 1 505A may be comprised of the network traffic associated with the transactions between the particular client and server. Allocating the network traffic to Core 1 505A can be accomplished by routing all data packets originating from either the particular client or server to Core 1 505A.

While work or load can be distributed to the cores based in part on transactions, in other embodiments load or work can be allocated on a per packet basis. In these embodiments, the appliance 200 can intercept data packets and allocate them to a core 505 having the least amount of load. For example, the appliance 200 could allocate a first incoming data packet to Core 1 505A because the load 515A on Core 1 is less than the load 515B-N on the rest of the cores 505B-N. Once the first data packet is allocated to Core 1 505A, the amount of load 515A on Core 1 505A is increased proportional to the amount of processing resources needed to process the first data packet. When the appliance 200 intercepts a second data packet, the appliance 200 will allocate the load to Core 4 505D because Core 4 505D has the second least amount of load. Allocating data packets to the core with the least amount of load can, in some embodiments, ensure that the load 515A-N distributed to each core 505 remains substantially equal.

In other embodiments, load can be allocated on a per unit basis where a section of network traffic is allocated to a particular core 505. The above-mentioned example illustrates load balancing on a per/packet basis. In other embodiments, load can be allocated based on a number of packets such that every 10, 100 or 1000 packets are allocated to the core 505 having the least amount of load. The number of packets allocated to a core 505 can be a number determined by an application, user or administrator and can be any number greater than zero. In still other embodiments, load can be allocated based on a time metric such that packets are distributed to a particular core 505 for a predetermined amount of time. In these embodiments, packets can be distributed to a particular core 505 for five milliseconds or for any period of time determined by a user, program, system, administrator or otherwise. After the predetermined time period elapses, data packets are transmitted to a different core 505 for the predetermined period of time.

Flow-based data parallelism methods for distributing work, load or network traffic among the one or more cores 505 can comprise any combination of the above-mentioned embodiments. These methods can be carried out by any part of the appliance 200, by an application or set of executable instructions executing on one of the cores 505, such as the packet engine, or by any application, program or agent executing on a computing device in communication with the appliance 200.

The functional and data parallelism computing schemes illustrated in FIG. 5A can be combined in any manner to generate a hybrid parallelism or distributed processing scheme that encompasses function parallelism 500, data parallelism 540, flow-based data parallelism 520 or any portions thereof. In some cases, the multi-core system may use any type and form

of load balancing schemes to distribute load among the one or more cores 505. The load balancing scheme may be used in any combination with any of the functional and data parallelism schemes or combinations thereof.

Illustrated in FIG. 5B is an embodiment of a multi-core system 545, which may be any type and form of one or more systems, appliances, devices or components. This system 545, in some embodiments, can be included within an appliance 200 having one or more processing cores 505A-N. The system 545 can further include one or more packet engines (PE) or packet processing engines (PPE) 548A-N communicating with a memory bus 556. The memory bus may be used to communicate with the one or more processing cores 505A-N. Also included within the system 545 can be one or more network interface cards (NIC) 552 and a flow distributor 550 which can further communicate with the one or more processing cores 505A-N. The flow distributor 550 can comprise a Receive Side Sealer (RSS) or Receive Side Scaling (RSS) module 560.

Further referring to FIG. 5B, and in more detail, in one embodiment the packet engine(s) 548A-N can comprise any portion of the appliance 200 described herein, such as any portion of the appliance described in FIGs. 2A and 2B. The packet engine(s) 548A-N can, in some embodiments, comprise any of the following elements: the packet engine 240, a network stack 267; a cache manager 232; a policy engine 236; a compression engine 238; an encryption engine 234; a GUI 210; a CLI 212; shell services 214; monitoring programs 216; and any other software or hardware element able to receive data packets from one of either the memory bus 556 or the one of more cores 505A-N. In some embodiments, the packet engine(s) 548A-N can comprise one or more vServers 275A-N, or any portion thereof. In other embodiments, the packet engine(s) 548A-N can provide any combination of the following functionalities: SSL VPN 280; Intranet UP 282; switching 284; DNS 286; packet acceleration 288; App FW 280; monitoring such as the monitoring provided by a monitoring agent 197; functionalities associated with functioning as a TCP stack; load balancing; SSL offloading and processing; content switching; policy evaluation; caching; compression; encoding; decompression; decoding; application firewall functionalities; XML processing and acceleration; and SSL VPN connectivity.

The packet engine(s) 548A-N can, in some embodiments, be associated with a particular server, user, client or network. When a packet engine 548 becomes associated with a particular entity, that packet engine 548 can process data packets associated with that entity. For example, should a packet engine 548 be associated with a first user, that packet engine 548 will process and operate on packets generated by the first user, or packets having a

destination address associated with the first user. Similarly, the packet engine 548 may choose not to be associated with a particular entity such that the packet engine 548 can process and otherwise operate on any data packets not generated by that entity or destined for that entity.

In some instances, the packet engine(s) 548A-N can be configured to carry out the any of the functional and/or data parallelism schemes illustrated in FIG. 5A. In these instances, the packet engine(s) 548A-N can distribute functions or data among the processing cores 505A-N so that the distribution is according to the parallelism or distribution scheme. In some embodiments, a single packet engine(s) 548A-N carries out a load balancing scheme, while in other embodiments one or more packet engine(s) 548A-N carry out a load balancing scheme. Each core 505A-N, in one embodiment, can be associated with a particular packet engine 548 such that load balancing can be carried out by the packet engine. Load balancing may in this embodiment, require that each packet engine 548A-N associated with a core 505 communicate with the other packet engines associated with cores so that the packet engines 548A-N can collectively determine where to distribute load. One embodiment of this process can include an arbiter that receives votes from each packet engine for load. The arbiter can distribute load to each packet engine 548A-N based in part on the age of the engine's vote and in some cases a priority value associated with the current amount of load on an engine's associated core 505.

Any of the packet engines running on the cores may run in user mode, kernel or any combination thereof. In some embodiments, the packet engine operates as an application or program running is user or application space. In these embodiments, the packet engine may use any type and form of interface to access any functionality provided by the kernel. In some embodiments, the packet engine operates in kernel mode or as part of the kernel. In some embodiments, a first portion of the packet engine operates in user mode while a second portion of the packet engine operates in kernel mode. In some embodiments, a first packet engine on a first core executes in kernel mode while a second packet engine on a second core executes in user mode. In some embodiments, the packet engine or any portions thereof operates on or in conjunction with the NIC or any drivers thereof.

In some embodiments the memory bus 556 can be any type and form of memory or computer bus. While a single memory bus 556 is depicted in FIG. 5B, the system 545 can comprise any number of memory buses 556. In one embodiment, each packet engine 548 can be associated with one or more individual memory buses 556.

The NIC 552 can in some embodiments be any of the network interface cards or mechanisms described herein. The NIC 552 can have any number of ports. The NIC can be designed and constructed to connect to any type and form of network 104. While a single NIC 552 is illustrated, the system 545 can comprise any number of NICs 552. In some embodiments, each core 505A-N can be associated with one or more single NICs 552. Thus, each core 505 can be associated with a single NIC 552 dedicated to a particular core 505. The cores 505A-N can comprise any of the processors described herein. Further, the cores 505A-N can be configured according to any of the core 505 configurations described herein. Still further, the cores 505A-N can have any of the core 505 functionalities described herein. While FIG. 5B illustrates seven cores 505A-G, any number of cores 505 can be included within the system 545. In particular, the system 545 can comprise "N" cores, where "N" is a whole number greater than zero.

A core may have or use memory that is allocated or assigned for use to that core. The memory may be considered private or local memory of that core and only accessible by that core. A core may have or use memory that is shared or assigned to multiple cores. The memory may be considered public or shared memory that is accessible by more than one core. A core may use any combination of private and public memory. With separate address spaces for each core, some level of coordination is eliminated from the case of using the same address space. With a separate address space, a core can perform work on information and data in the core's own address space without worrying about conflicts with other cores. Each packet engine may have a separate memory pool for TCP and/or SSL connections.

Further referring to FIG. 5B, any of the functionality and/or embodiments of the cores 505 described above in connection with FIG. 5A can be deployed in any embodiment of the virtualized environment described above in connection with FIGs. 4A and 4B. Instead of the functionality of the cores 505 being deployed in the form of a physical processor 505, such functionality may be deployed in a virtualized environment 400 on any computing device 100, such as a client 102, server 106 or appliance 200. In other embodiments, instead of the functionality of the cores 505 being deployed in the form of an appliance or a single device, the functionality may be deployed across multiple devices in any arrangement. For example, one device may comprise two or more cores and another device may comprise two or more cores. For example, a multi-core system may include a cluster of computing devices, a server farm or network of computing devices. In some embodiments, instead of the functionality of the cores 505 being deployed in the form of cores, the functionality may be deployed on a plurality of processors, such as a plurality of single core processors.

In one embodiment, the cores 505 may be any type and form of processor. In some embodiments, a core can function substantially similar to any processor or central processing unit described herein. In some embodiment, the cores 505 may comprise any portion of any processor described herein. While FIG. 5A illustrates seven cores, there can exist any "N" number of cores within an appliance 200, where "N" is any whole number greater than one. In some embodiments, the cores 505 can be installed within a common appliance 200, while in other embodiments the cores 505 can be installed within one or more appliance(s) 200 communicatively connected to one another. The cores 505 can in some embodiments comprise graphics processing software, while in other embodiments the cores 505 provide general processing capabilities. The cores 505 can be installed physically near each other and/or can be communicatively connected to each other. The cores may be connected by any type and form of bus or subsystem physically and/or communicatively coupled to the cores for transferring data between to, from and/or between the cores.

While each core 505 can comprise software for communicating with other cores, in some embodiments a core manager (not shown) can facilitate communication between each core 505. In some embodiments, the kernel may provide core management. The cores may interface or communicate with each other using a variety of interface mechanisms. In some embodiments, core to core messaging may be used to communicate between cores, such as a first core sending a message or data to a second core via a bus or subsystem connecting the cores. In some embodiments, cores may communicate via any type and form of shared memory interface. In one embodiment, there may be one or more memory locations shared among all the cores. In some embodiments, each core may have separate memory locations shared with each other core. For example, a first core may have a first shared memory with a second core and a second share memory with a third core. In some embodiments, cores may communicate via any type of programming or API, such as function calls via the kernel. In some embodiments, the operating system may recognize and support multiple core devices and provide interfaces and API for inter-core communications.

The flow distributor 550 can be any application, program, library, script, task, service, process or any type and form of executable instructions executing on any type and form of hardware. In some embodiments, the flow distributor 550 may any design and construction of circuitry to perform any of the operations and functions described herein. In some embodiments, the flow distributor distribute, forwards, routes, controls and/ors manage the distribution of data packets among the cores 505 and/or packet engine or VIPs running on the cores.. The flow distributor 550, in some embodiments, can be referred to as an interface

master. In one embodiment, the flow distributor 550 comprises a set of executable instructions executing on a core or processor of the appliance 200. In another embodiment, the flow distributor 550 comprises a set of executable instructions executing on a computing machine in communication with the appliance 200. In some embodiments, the flow distributor 550 comprises a set of executable instructions executing on a NIC, such as firmware. In still other embodiments, the flow distributor 550 comprises any combination of software and hardware to distribute data packets among cores or processors. In one embodiment, the flow distributor 550 executes on at least one of the cores 505A-N, while in other embodiments a separate flow distributor 550 assigned to each core 505A-N executes on an associated core 505A-N. The flow distributor may use any type and form of statistical or probabilistic algorithms or decision making to balance the flows across the cores. The hardware of the appliance, such as a NIC, or the kernel may be designed and constructed to support sequential operations across the NICs and/or cores.

In embodiments where the system 545 comprises one or more flow distributors 550, each flow distributor 550 can be associated with a processor 505 or a packet engine 548. The flow distributors 550 can comprise an interface mechanism that allows each flow distributor 550 to communicate with the other flow distributors 550 executing within the system 545. In one instance, the one or more flow distributors 550 can determine how to balance load by communicating with each other. This process can operate substantially similarly to the process described above for submitting votes to an arbiter which then determines which flow distributor 550 should receive the load. In other embodiments, a first flow distributor 550' can identify the load on an associated core and determine whether to forward a first data packet to the associated core based on any of the following criteria: the load on the associated core is above a predetermined threshold; the load on the associated core is below a predetermined threshold; the load on the associated core is less than the load on the other cores; or any other metric that can be used to determine where to forward data packets based in part on the amount of load on a processor.

The flow distributor 550 can distribute network traffic among the cores 505 according to a distribution, computing or load balancing scheme such as those described herein. In one embodiment, the flow distributor can distribute network traffic according to any one of a functional parallelism distribution scheme 550, a data parallelism load distribution scheme 540, a flow-based data parallelism distribution scheme 520, or any combination of these distribution scheme or any load balancing scheme for distributing load among multiple processors. The flow distributor 550 can therefore act as a load distributor by taking in data

packets and distributing them across the processors according to an operative load balancing or distribution scheme. In one embodiment, the flow distributor 550 can comprise one or more operations, functions or logic to determine how to distribute packers, work or load accordingly. In still other embodiments, the flow distributor 550 can comprise one or more sub operations, functions or logic that can identify a source address and a destination address associated with a data packet, and distribute packets accordingly.

In some embodiments, the flow distributor 550 can comprise a receive-side scaling (RSS) network driver, module 560 or any type and form of executable instructions which distribute data packets among the one or more cores 505. The RSS module 560 can comprise any combination of hardware and software, In some embodiments, the RSS module 560 works in conjunction with the flow distributor 550 to distribute data packets across the cores 505A-N or among multiple processors in a multi-processor network. The RSS module 560 can execute within the NIC 552 in some embodiments, and in other embodiments can execute on any one of the cores 505.

In some embodiments, the RSS module 560 uses the MICROSOFT receive-side-scaling (RSS) scheme. In one embodiment, RSS is a Microsoft Scalable Networking initiative technology that enables receive processing to be balanced across multiple processors in the system while maintaining in-order delivery of the data. The RSS may use any type and form of hashing scheme to determine a core or processor for processing a network packet.

The RSS module 560 can apply any type and form hash function such as the Toeplitz hash function. The hash function may be applied to the hash type or any the sequence of values. The hash function may be a secure hash of any security level or is otherwise cryptographically secure. The hash function may use a hash key. The size of the key is dependent upon the hash function. For the Toeplitz hash, the size may be 40 bytes for IPv6 and 16 bytes for IPv4.

The hash function may be designed and constructed based on any one or more criteria or design goals. In some embodiments, a hash function may be used that provides an even distribution of hash result for different hash inputs and different hash types, including TCP/IPv4, TCP/IPv6, IPv4, and IPv6 headers. In some embodiments, a hash function may be used that provides a hash result that is evenly distributed when a small number of buckets are present (for example, two or four). In some embodiments, hash function may be used that provides a hash result that is randomly distributed when a large number of buckets were present (for example, 64 buckets). In some embodiments, the hash function is determined

based on a level of computational or resource usage. In some embodiments, the hash function is determined based on ease or difficulty of implementing the hash in hardware. In some embodiments, the hash function is determined based on the ease or difficulty of a malicious remote host to send packets that would all hash to the same bucket.

The RSS may generate hashes from any type and form of input, such as a sequence of values. This sequence of values can include any portion of the network packet, such as any header, field or payload of network packet, or portions thereof. In some embodiments, the input to the hash may be referred to as a hash type and include any tuples of information associated with a network packet or data flow, such as any of the following: a four tuple comprising at least two IP addresses and two ports; a four tuple comprising any four sets of values; a six tuple; a two tuple; and/or any other sequence of numbers or values. The following are example of hash types that may be used by RSS:

- 4-tuple of source TCP Port, source IP version 4 (IPv4) address, destination TCP Port, and destination IPv4 address.
- 4-tuple of source TCP Port, source IP version 6 (IPv6) address, destination TCP Port, and destination IPv6 address.
- 2-tuple of source IPv4 address, and destination IPv4 address.
- 2-tuple of source IPv6 address, and destination IPv6 address.
- 2-tuple of source IPv6 address, and destination IPv6 address, including support for parsing IPv6 extension headers.

The hash result or any portion thereof may used to identify a core or entity, such as a packet engine or VIP, for distributing a network packet. In some embodiments, one or more hash bits or mask are applied to the hash result. The hash bit or mask may be any number of bits or bytes. A NIC may support any number of bits, such as seven bits. The network stack may set the actual number of bits to be used during initialization. The number will be between 1 and 7, inclusive.

The hash result may be used to identify the core or entity via any type and form of table, such as a bucket table or indirection table. In some embodiments, the number of hash-result bits are used to index into the table. The range of the hash mask may effectively define the size of the indirection table. ny portion of the hash result or the hast result itself may be used to index the indirection table. The values in the table may identify any of the cores or processor, such as by a core or processor identifier. In some embodiments, all of the cores of

the multi-core system are identified in the table. In other embodiments, a port of the cores of the multi-core system are identified in the table. The indirection table may comprise any number of buckets for example 2 to 128 buckets that may be indexed by a hash mask. Each bucket may comprise a range of index values that identify a core or processor. In some embodiments, the flow controller and/or RSS module may rebalance the network rebalance the network load by changing the indirection table.

In some embodiments, the multi-core system 575 does not include a RSS driver or RSS module 560. In some of these embodiments, a software steering module (not shown) or a software embodiment of the RSS module within the system can operate in conjunction with or as part of the flow distributor 550 to steer packets to cores 505 within the multi-core system 575.

The flow distributor 550, in some embodiments, executes within any module or program on the appliance 200, on any one of the cores 505 and on any one of the devices or components included within the multi-core system 575. In some embodiments, the flow distributor 550' can execute on the first core 505A, while in other embodiments the flow distributor 550" can execute on the NIC 552. In still other embodiments, an instance of the flow distributor 550' can execute on each core 505 included in the multi-core system 575. In this embodiment, each instance of the flow distributor 550' can communicate with other instances of the flow distributor 550' to forward packets back and forth across the cores 505. There exist situations where a response to a request packet may not be processed by the same core, i.e. the first core processes the request while the second core processes the response. In these situations, the instances of the flow distributor 550' can intercept the packet and forward it to the desired or correct core 505, i.e. a flow distributor instance 550' can forward the response to the first core. Multiple instances of the flow distributor 550' can execute on any number of cores 505 and any combination of cores 505.

The flow distributor may operate responsive to any one or more rules or policies. The rules may identify a core or packet processing engine to receive a network packet, data or data flow. The rules may identify any type and form of tuple information related to a network packet, such as a 4-tuple of source and destination IP address and source and destination ports. Based on a received packet matching the tuple specified by the rule, the flow distributor may forward the packet to a core or packet engine. In some embodiments, the packet is forwarded to a core via shared memory and/or core to core messaging.

Although FIG. 5B illustrates the flow distributor 550 as executing within the multi-core system 575, in some embodiments the flow distributor 550 can execute on a computing

device or appliance remotely located from the multi-core system 575. In such an embodiment, the flow distributor 550 can communicate with the multi-core system 575 to take in data packets and distribute the packets across the one or more cores 505. The flow distributor 550 can, in one embodiment, receive data packets destined for the appliance 200, apply a distribution scheme to the received data packets and distribute the data packets to the one or more cores 505 of the multi-core system 575. In one embodiment, the flow distributor 550 can be included in a router or other appliance such that the router can target particular cores 505 by altering meta data associated with each packet so that each packet is targeted towards a sub-node of the multi-core system 575. In such an embodiment, CISCO'S vn-tag mechanism can be used to alter or tag each packet with the appropriate meta data.

Illustrated in FIG. 5C is an embodiment of a multi-core system 575 comprising one or more processing cores 505A-N. In brief overview, one of the cores 505 can be designated as a control core 505A and can be used as a control plane 570 for the other cores 505. The other cores may be secondary cores which operate in a data plane while the control core provides the control plane. The cores 505A-N may share a global cache 580. While the control core provides a control plane, the other cores in the multi-core system form or provide a data plane. These cores perform data processing functionality on network traffic while the control provides initialization, configuration and control of the multi-core system.

Further referring to FIG. 5C, and in more detail, the cores 505A-N as well as the control core 505A can be any processor described herein. Furthermore, the cores 505A-N and the control core 505A can be any processor able to function within the system 575 described in FIG. 5C. Still further, the cores 505A-N and the control core 505A can be any core or group of cores described herein. The control core may be a different type of core or processor than the other cores. In some embodiments, the control may operate a different packet engine or have a packet engine configured differently than the packet engines of the other cores.

Any portion of the memory of each of the cores may be allocated to or used for a global cache that is shared by the cores. In brief overview, a predetermined percentage or predetermined amount of each of the memory of each core may be used for the global cache. For example, 50% of each memory of each code may be dedicated or allocated to the shared global cache. That is, in the illustrated embodiment, 2GB of each core excluding the control plane core or core 1 may be used to form a 28GB shared global cache. The configuration of the control plane such as via the configuration services may determine the amount of memory used for the shared global cache. In some embodiments, each core may provide a different

amount of memory for use by the global cache. In other embodiments, any one core may not provide any memory or use the global cache. In some embodiments, any of the cores may also have a local cache in memory not allocated to the global shared memory. Each of the cores may store any portion of network traffic to the global shared cache. Each of the cores may check the cache for any content to use in a request or response. Any of the cores may obtain content from the global shared cache to use in a data flow, request or response.

The global cache 580 can be any type and form of memory or storage element, such as any memory or storage element described herein. In some embodiments, the cores 505 may have access to a predetermined amount of memory (i.e. 32 GB or any other memory amount commensurate with the system 575). The global cache 580 can be allocated from that predetermined amount of memory while the rest of the available memory can be allocated among the cores 505. In other embodiments, each core 505 can have a predetermined amount of memory. The global cache 580 can comprise an amount of the memory allocated to each core 505. This memory amount can be measured in bytes, or can be measured as a percentage of the memory allocated to each core 505. Thus, the global cache 580 can comprise 1 GB of memory from the memory associated with each core 505, or can comprise 20 percent or one-half of the memory associated with each core 505. In some embodiments, only a portion of the cores 505 provide memory to the global cache 580, while in other embodiments the global cache 580 can comprise memory not allocated to the cores 505.

Each core 505 can use the global cache 580 to store network traffic or cache data. In some embodiments, the packet engines of the core use the global cache to cache and use data stored by the plurality of packet engines. For example, the cache manager of FIG. 2A and cache functionality of FIG. 2B may use the global cache to share data for acceleration. For example, each of the packet engines may store responses, such as HTML data, to the global cache. Any of the cache managers operating on a core may access the global cache to server caches responses to client requests.

In some embodiments, the cores 505 can use the global cache 580 to store a port allocation table which can be used to determine data flow based in part on ports. In other embodiments, the cores 505 can use the global cache 580 to store an address lookup table or any other table or list that can be used by the flow distributor to determine where to direct incoming and outgoing data packets. The cores 505 can, in some embodiments read from and write to cache 580, while in other embodiments the cores 505 can only read from or write to cache 580. The cores may use the global cache to perform core to core communications.

The global cache 580 may be sectioned into individual memory sections where each section can be dedicated to a particular core 505. In one embodiment, the control core 505A can receive a greater amount of available cache, while the other cores 505 can receiving varying amounts or access to the global cache 580.

In some embodiments, the system 575 can comprise a control core 505A. While FIG. 5C illustrates core 1 505A as the control core, the control core can be any core within the appliance 200 or multi-core system. Further, while only a single control core is depicted, the system 575 can comprise one or more control cores each having a level of control over the system. In some embodiments, one or more control cores can each control a particular aspect of the system 575. For example, one core can control deciding which distribution scheme to use, while another core can determine the size of the global cache 580.

The control plane of the multi-core system may be the designation and configuration of a core as the dedicated management core or as a master core. This control plane core may provide control, management and coordination of operation and functionality the plurality of cores in the multi-core system. This control plane core may provide control, management and coordination of allocation and use of memory of the system among the plurality of cores in the multi-core system, including initialization and configuration of the same. In some embodiments, the control plane includes the flow distributor for controlling the assignment of data flows to cores and the distribution of network packets to cores based on data flows. In some embodiments, the control plane core runs a packet engine and in other embodiments, the control plane core is dedicated to management and control of the other cores of the system.

The control core 505A can exercise a level of control over the other cores 505 such as determining how much memory should be allocated to each core 505 or determining which core 505 should be assigned to handle a particular function or hardware/software entity. The control core 505A, in some embodiments, can exercise control over those cores 505 within the control plan 570. Thus, there can exist processors outside of the control plane 570 which are not controlled by the control core 505A. Determining the boundaries of the control plane 570 can include maintaining, by the control core 505A or agent executing within the system 575, a list of those cores 505 controlled by the control core 505A. The control core 505A can control any of the following: initialization of a core; determining when a core is unavailable; re-distributing load to other cores 505 when one core fails; determining which distribution scheme to implement; determining which core should receive network traffic; determining how much cache should be allocated to each core; determining whether to assign a particular

function or element to a particular core; determining whether to permit cores to communicate with one another; determining the size of the global cache 580; and any other determination of a function, configuration or operation of the cores within the system 575.

F. Systems and Methods for Maintaining Operation of a Multi-Core Network Appliance Upon Failover

Referring now to FIG. 6A, an embodiment of a system for controlling a rate of a traffic traversing an intermediary 200 is illustrated. In brief overview, FIG. 6A depicts an intermediary 200 comprising a rate limiting manager (RLM) 605 in communication with a rate limiting license 660 which identifies a performance level 665. Data packets 601 are received by the RLM 605 and flow rate controlled by a throttler 625 of the RLM 605. Data packets 601 that are propagated or throttled by the throttler 625 are sent out of the RLM 605 towards the packet engines 548A-N which operate on the cores 505A-N. RLM 605 further includes a token generator 610 for generating tokens 602 at a token rate 615 into a token bucket 620 which holds or keeps a count of all the tokens 602. An excess handler 630 of the RLM 605 handles any data packets 601 that are not received or not rate controlled by the throttler 625. RLM 605 further includes a plurality of performance level settings 640A-640N. Each performance level setting (PLS) 640 may comprise a rate limit settings 645 which may have a bucket settings 646 and a throughput rate 650 comprising a bytes per second (BPS) limit 651 and a packet per second (PPL) limit 652. As illustrated by FIG. 6A, RLM 605 may set, configure and manage a rate of flow of data packets 601 traversing the throttle 625 at a rate limit that is identified by a performance level 665 of the rate limiting license 660. The rate limit for propagating data packets 601 may be controlled by a number of tokens 602 which may need to be available for each propagated data packet 601.

Referring to FIG. 6A in a greater detail, rate limiting license 660 may include any type and form of hardware, software or any combination of hardware and software for providing a license, authorization or a permit to control a rate of network traffic received and/or transmitted via an appliance 200. In some embodiments, rate limiting license 660 includes any type and form of a program, an application, an executable, a script, a function, a unit or a device for providing a license or permit. In other embodiments, rate limiting license 660 is a component of a software installed on the appliance 200. In further embodiments, rate limiting license 660 includes a file, program, script or an executable that is installed or enabled by an operator, administrator or a service provider for the appliance 200. In some embodiments, rate limiting license 660 includes a third party software. Rate limiting license

660 may include a license file validation. Intermediary 200 may use a file for validating a license and use a link, a URL or a directory path to validate the rate limiting license 600. In some embodiments, a rate limiting license 660 may be confirmed or verified via a remote database or link, such as for example MS Windows license verification model. Rate limiting license 600 may include contents which upon testing or inspection by the intermediary may be validated as the rate limiting license 600. In some embodiments, rate limiting license 660 includes a unique identifier or a serial number for the appliance 200 or for any service provided by the appliance. In some embodiments, rate limiting license 660 may include a data structure, an object or an entry in a data base. The data structure, object or the entry may identify or provide a license for an entity.

In further embodiments, rate limiting license 660 includes a component, unit, function or a program that controls a specific performance level. The specific performance level may correspond to a configuration or operation of the appliance 200 in accordance with a predetermined set of parameters or settings. In some embodiments, rate limiting license 660 enables the appliance to be operate only at a single performance level. In other embodiments, rate limiting license 660 enables a plurality of performance levels for the appliance. In further embodiments, rate limiting license 660 disables all except one performance level for the appliance 200. Rate limiting license 660 may include, provide or identify one or more performance levels, such as the performance level 665.

Performance level 665 may be any data or information for identifying or specifying a level of performance of hardware, software or hardware and software for an appliance 200, or any portion thereof. The level of performance may include a range or a limitation for a rate of receipt/transmit of network traffic or a rate of processing of data packets, data or a data stream traversing an appliance 200. Performance level 665 may be identified via a file, an executable, a program, an application, a script, function, an algorithm, a unit or a device. In some embodiments, performance level 665 includes an encryption/decryption key for decrypting and enabling a predetermined performance level to be used by the appliance 200. In other embodiments, performance level 665 includes a keyword or an instruction used by RLM 605 to identify a predetermined set of performance level settings 645. In some embodiments, performance level 665 includes an algorithm, application, executable or unit that enables access to a set of instructions and settings that enable a level of performance of the appliance 200. Performance level 665 may comprise any number of configuration settings and values, instructions, configuration files and executables, data values and any

other type and form of hardware or software to specify, identify and enable the appliance 200 to function or operate at a level specified by the performance level 665.

Performance level 665 may include or specify any type and form of information for identifying a level of performance or a level of operation of the appliance 200. In some embodiments, performance level 665 includes an information about data flow rate threshold or a limitation for the receipt and/or transmit of data in bytes of data per second or in data packets per second. The information about data flow rate may include a flow limit or an upper limit threshold for the performance, or the data rate flow, for the appliance 200. In some embodiments, the information about data flow rate includes a lower limit threshold for the performance, or the rate flow of data, of the appliance 200.

Rate limiting license 660 or the performance level 665 may identify or specify any one performance level of a plurality of performance levels supported by the appliance, such performance level 5500. Each of the performance levels may identify a model or type of the appliance 200. Each of the performance levels may be associated with a predetermined threshold of performance or rate of performance or processing of the network packets. For example, in some embodiments, performance level 5500 limits the maximum rate of flow of the data packets traversing the appliance 200 at 5500 packets per second. In other embodiments, performance level 5500 limits the maximum rate of flow of the data packets traversing the appliance 200 at 5500 bytes per second. Rate limiting license 660 or the performance level 665 may identify or specify performance level 7500. In some embodiments, performance level 7500 limits the maximum rate of flow of the data packets traversing the appliance 200 at 7500 packets per second. In other embodiments, performance level 7500 limits the maximum rate of flow of the data packets traversing the appliance 200 at 7500 bytes per second. Rate limiting license 660 or the performance level 665 may identify or specify performance level 9500. In some embodiments, performance level 9500 limits the maximum rate of flow of the data packets traversing the appliance 200 at 9500 packets per second. In other embodiments, performance level 9500 limits the maximum rate of flow of the data packets traversing the appliance 200 at 9500 bytes per second. Rate limiting license 660 or the performance level 665 may identify or specify performance level 10500. In some embodiments, performance level 10500 limits the maximum rate of flow of the data packets traversing the appliance 200 at 10500 packets per second. In other embodiments, performance level 10500 limits the maximum rate of flow of the data packets traversing the appliance 200 at 10500 bytes per second. Rate limiting license 660 or the performance level 665 may identify or specify performance level 12500. In some

embodiments, performance level limits the maximum rate of flow of the data packets traversing the appliance 200 at 12500 packets per second. In other embodiments, performance level 12500 limits the maximum rate of flow of the data packets traversing the appliance 200 at 12500 bytes per second.

Data packets 601 may include any type and form of data and any type and form of units, groups or elements of data. Data packets 601 may include any information, signal or transmission traversing an appliance 200. Data packets 601 may also include any type and form of formatted or non-formatted data. In some embodiments, data packets 601 are formatted units or chunks of data carried by a packet mode computer network. The formatted units or chunks of data may be of a same size or a varying size. In further embodiments, data packets 601 are formatted or formed network data packets for a network 104. Data packets 601 may include a header or an envelope. Data packets 601 may also include one or more data bits or bytes. In some embodiments, data packets 601 are formed or organized into groups that include 1, 2, 4, 8, 16, 24, 32, 48, 64, 96, 128, 196 or 256 bits. Data packets 601 may include 1, 2, 4, 8, 16, 24, 32, 48, 64, 96, 128, 196, 256, 512 or 1024 bytes. Data packets 601 may also include 1, 2, 4, 8, 16, 24, 32, 48, 64, 96, 128, 196, 256, 512 or 1024 Megabytes.

Data packets 601 may be formed into a stream of data packets or a stream of data bits. Data packets 601 of a stream of data may be of a same or a similar size. In some embodiments, data packets 601 of a stream of data are of varying sizes. Data packets 601 may be formatted in any number of ways. Some data packets 601 may be formatted in accordance a communication protocol, such as TCP, IP, UDP, HTTP, DHCP, POP3, SMPT, Citrix XenApp, Citrix ICA protocol or any other type and form of communication protocol for any communication layer or level. In some embodiments, data packets 601 include compressed data packets. Some data packets 601, in other embodiments, may be not compressed. In some embodiments, data packets 601 are formatted network packets, such as TCP/IP data packets. Data packets 601 may include any number of data bits or bytes. In some embodiments, data packets 601 include a data bit or a data byte. In some embodiments, data packets 601 comprise one or more data bits, or a stream of data bits. In further embodiments, data packets 601 includes a byte. In further embodiments, data packets 601 include a plurality of bytes. In some embodiments, data packets 601 include a request from a client 102. In other embodiments, data packets 601 include a response from a server 106. Data packets 601 may include any number of formatted or non-formatted data groups, chunks

or units of data. Data packets 601 may also include any number of bits, bytes, characters or any other units of information transmitted via a network 104 or via an appliance 200.

Rate limiting manager 605, also known as RLM 605, may include any type and form of algorithms or functions for managing or controlling a rate of operation, process or propagation of the network packets in accordance with the performance level identified by the license. RLM 605 may use instructions from the rate limiting license 660 to set up and configure the operation and function of the appliance 200 to process data packets of the network traffic at a predetermined rate. By way of example, RLM 605 may use a token based system to control the rate at which data packets of the network traffic are received by one or more packet engines 548. The token based system may include a token generator that generates tokens 602 at a token rate 615. The token based system may further include a token bucket 620 maintaining and keeping a track of the tokens available and a throttler 625 which receives and propagates data packets 601 conditioned by availability of tokens 602. As such the token based system controls the throughput of the data packets 601 by throttling of the data packets 601 at a rate of available tokens 602. The token based system may further include a performance level settings 640A-640N for each different performance level 665 that the license may identify. The performance level settings 640A-640N may identify various speeds or rates of processing of the data packets 601 by the RLM 605. Each performance level settings 640 may further include rate limit settings 645 that includes a bucket settings 646 and a throughput rate 650. The throughput rate 650 may also include a bytes per second limit 651 and packets per second limit 652. RLM 605 may identify a performance level settings 645 for the appliance 200 in response to the performance level 665 of the rate limiting license and operate in accordance with the identified settings. In some embodiments where the performance level 665 is not identified, RLM 605 may identify a default performance level settings 640 according to which the appliance 200 may operate. In some embodiments, the default performance level settings comprises the settings with the slowest rate of operation, propagation and throughput. RLM 605 may be operating on a single-core system or a multi-core system. In a single core system, RLM 605 may operate on the main central processing unit (CPU). In a multi-core system, RLM 605 may operate on a single or a plurality of cores 505. RLM 605 may be configured to operate on each core 505 to control the throughput rate for each packet engine 548 on each of the cores 505.

Rate limiting manager 605, also referred to as RLM 605, may include any hardware, software or any combination of hardware and software for initiating, establishing, managing, controlling and/or implementing rate limiting of data traffic. RLM 605 may include any

number of files, scripts, programs, applications, functions, algorithms, libraries, units, devices or executables for performing any function for limiting the rate of any data traffic traversing the appliance 200. RLM 605 may include any number of processors or processing units, logic circuits, analog or digital circuits for initiating, establishing, managing, controlling and implementing rate control of the data traffic. In some embodiments, rate limiting manager 605 comprises any functionality, logic, circuitry, software or applications for controlling the flow of data packets received by the appliance 200. Rate limiting manager 605 may further include any number of RLM 605 components, such as any number of the token generators 610, token rates 615, tokens 602, token buckets 620, data packets 601, throttlers 625, excess handlers 630, performance levels 640A-N, rate limit settings 645, bucket settings 646, throughput rates 650, BPS limits 651 and PPS limits 652. RLM may initiate and configure a set of RLM 605 components for each of the packet engines 548 on each of the plurality of cores 505A-N. An appliance 200 may initiate, configure, set up and implement any number of RLMs 605 for any number of PEs 548 which may run or operate on any number of cores 505A-N.

RLM 605 may include any functionality for controlling, managing, monitoring, accelerating or decelerating the flow or propagation of data packets 601. In some embodiments, RLM 605 comprises controllers, functions or units that control, organize and manage a flow of data packets 601. RLM 605 may include one or more queues for receiving or storing incoming data packets 601. RLM 605 may further use or interface with any of the existing queues of the intermediary 200. The queues accessed, monitored, managed or used by the RLM 605 may correspond to any number of network interface cards (NICs) or data ports that receive data packets 601 from one or more clients 102 or servers 106. Queues used and managed by the RLM may include queues, such as a receiving queue at the NICs or ports, SSL queues, queues storing compressed network traffic, queues storing decompressed network traffic, queues storing data specific applications, servers or clients, queues for the VIPs or virtual servers 270, queues for any component of the intermediary 200 or any network traffic for any component of the appliance 200. Similarly, RLM 605 may include one or more queues for receiving and storing data packets 601 that are being received by the throttler 625. In some embodiments, RLM 605 stores information or data packets 601 from the queues intended for one or more packet engines 548 on one or more cores 505. In some embodiments, RLM 605 includes any component, unit or function for searching for and identifying rate limiting license 660. RLM 605 may include functionality for communicating with the rate limiting license 660 and identifying performance level 665 information. RLM

605 may include functionality or means for recognizing and identifying the performance level 660. RLM 605 may further include functionality or means for implementing rate limit settings for the appliance 200 based on, or responsive to, the information identified by the performance level 660. RLM 605 may further include any functionality for generating operation and configuration settings for the appliance 200 to implement the rate limit identified by the performance level 665 of the rate limiting license 660.

Queues that are managed or accessed by the RLM 605 may be configured in a variety of ways. RLM 605 may manage, interface with or receive data packets from any number of queues, such as the NICs queues or SSL queues. The queues may configured to receive network traffic until their capacity is reached. In some embodiments, queues receiving network traffic are configured to drop, or tail drop, any additional network packets that cannot be accepted by the queue. In some embodiments, the NIC drops or tail drops packets. In further embodiments, if an amount of network packets received exceeds a predetermined threshold, the network packets may be dropped or not accepted by the queues. In further embodiments, when data packets are tail-dropped, data packets may be resent by the sender at a later time when queues are available to accept additional data packets.

A token 602 may include any value, character, number, count, object or any combination of hardware and software to be used for counting, maintaining or keeping a track of a number of data packets 601 that may be propagated. A token 602 may include any file, object, character, symbol, value or a number to be used by any component of the RLM 605, such as a throttler 625 or a token bucket 620 for maintaining a count. The count maintained using tokens 602 may be any count, sum or tally for determining an amount or a number of data packets 602 to be propagated, processed or throttled by the RLM 605. Token 602 may include an object, an executable or a file. In further embodiments, token 602 includes a cookie. In yet further embodiments, token 602 includes a set of characters, values and parameters identifying a specific data packet 601, or a specific type of data packet 601. Token 602 identifying a specific data packet 601 or a specific data packet type may be used by a throttler 625 for propagating such a data packet 601 . In some embodiments, one or more tokens 602 comprise a count or a summation value. In further embodiments, one or more tokens 602 are a value or a number inside a counter or an algorithm maintaining a count or a total for the tokens 602. Tokens 602 may be counted or maintained by an algorithm or an application that keeps the count of tokens 602 by adding new tokens 602 to the total count or subtracting existing tokens 602 from the total count. Tokens 602 may be added or counted up in a counter at a token generation rate, such as token rate 615, for each new token 602

generated. In some embodiments, tokens 602 are counted down or subtracted from a total sum of tokens 602 for each data packet 601 that is processed, propagated or throttled by the throttler 625. In further embodiments, token 602 comprises a number or a value that corresponds to a number of data packets 601 allowed for processing, propagating or throttling at present moment. A token 602 may comprise any count, count variable, value, number, object or component used for counting, keeping count of or tracking a number of data packets 601 that may be allowed to be propagated by the RLM 605.

Token generator 610 may include any hardware, software or any combination of hardware and software for generating, managing, adding, subtracting, or otherwise controlling a count of tokens 602. Token generator 610 may include any number of files, scripts, programs, applications, functions, algorithms, processing units, logic circuits, analog or digital circuits or executables for producing, managing, adding or subtracting tokens 602. Token generator 610 may comprise any functionality and means for generating, maintaining and keeping a track of a total count or a total number of tokens 602 available. Token generator 610 may subtracts a token 602 or a count for each data packet 601 that propagates, processes or throttles through the RLM 605. In such embodiments, token generator 610 may add a token 602 or a count for each period of time defined by a token rate 615 (1 / (token rate) in tokens/second) for which a data packet 601 is not propagated, processed or throttled through the RLM 605. Token generator 610 may also add a token 602 or a count for each data packet 601 that propagates, processes or throttles through the RLM 605. In such embodiments, token generator 610 subtracts a token 602 or a count for each period of time defined by a token rate 615 for which a data packet 601 is not propagated, processed or throttled through the RLM 605. Token generator 610 may add or generates any number of tokens 602 as defined by the token rate 615. In some embodiments, token generator 610 may subtract, count down or terminate any number of tokens 602 as defined by the token rate 615. Token generator 610 may include a program, an application or an algorithm counting, or maintaining a count. Token generator 610 may maintain a number of counts or tokens 602 available at each moment. In some embodiments, token generator 610 generates or adds a number of tokens to a total number of tokens 602. In some embodiments, token generator 610 subtracts or terminates a number of tokens 602 from a total number of tokens 602. Adding, subtracting, generating or terminating or any other action performed by the token generator 610 on the tokens 602 may be responsive to a timing counter defined by a token rate 615. In some embodiments, adding, subtracting, generating or terminating or any other

action performed by the token generator 610 on the tokens 602 is responsive to a data packet 602 propagated, processed or throttled by the RLM 605.

Token rate 615 may be any rate at which tokens 602 are established, counted or generated. Token rate 615 may include any hardware, software or any combination of hardware and software for establishing or generating a rate, a tempo or a pace for production, counting or generating tokens 602. Token rate 615 may include an application, an algorithm, an executable or a counter for maintaining and managing a rate at which tokens 602 are generated or terminated. In some embodiments, token rate 615 includes a rate for generating or increasing a number of tokens 602 in tokens 602 per second. In other embodiments, token rate 615 includes a rate of adding a count or counting up a counter that corresponds to a total number of tokens 602. In further embodiments, token rate 615 includes a rate for terminating or decreasing a number of tokens 602 in tokens 602 per second. In other embodiments, token rate 615 includes a rate of subtracting a count or counting down a counter that corresponds to a total number of tokens 602. In some embodiments, token rate 615 comprises any rate between 1 and 100 bytes per second, such as 1, 10, 20, 30, 40, 50, 60, 70, 80, 90 or 100 tokens 602 per second. In other embodiments, token rate 615 includes any range of rates between 100 and 1000 tokens 602 per second, such as 100, 150, 200, 250, 300, 350, 400, 450, 500, 550, 600, 650, 700, 750, 800, 850, 900, 950 or 1000 tokens 602 per second. In further embodiments, token rate 615 includes any range of rates between 1000 and 10000 tokens 602 per second, such as 1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500, 5000, 5500, 6000, 6500, 7000, 7500, 8000, 8500, 9000, 9500 or 10000 tokens 602 per second. In yet further embodiments, token rate 615 includes any range of rates between 10,000 and 100000 tokens 602 per second, such as 10000, 15000, 20000, 25000, 30000, 35000, 40000, 45000, 50000, 55000, 60000, 65000, 70000, 75000, 80000, 85000, 90000, 95000 or 100000 tokens 602 per second. In still further embodiments, token rate 615 includes any range of rates between 100,000 and 1,000,000 tokens 602 per second, such as 100000, 150000, 200000, 250000, 300000, 350000, 400000, 450000, 500000, 550000, 600000, 650000, 700000, 750000, 800000, 850000, 900000, 950000 or 1000000 tokens 602 per second. In yet further embodiments, token rate 615 includes any range of rates between 1,000,000 and 1,000,000,000 tokens 602 per second, such as 1,000,000, 5,000,000, 10,000,000, 50,000,000, 100,000,000, 500,000,000 or 1,000,000,000 tokens 602 per second. Token rate 615 may be used for managing or controlling the rate of propagation, processing or throttling of data packets 602 through the throttler 625. Token rate 615 may be created by the token generator

610 responsive to any information or settings from any of the rate limiting license 660 or any performance level settings 640.

Token bucket 620 may include any hardware, software or any combination of hardware and software for managing and maintaining a total count or a total tally of available tokens 602. Token bucket 620 may include any logic, application, function or an algorithm for maintaining a total number or tally of tokens 602. Token bucket 620 may include any logic, application, function or an algorithm for establishing a maximum size for the token bucket 620. Token bucket 620 may refuse to accept additional tokens 602 once a maximum size for the tokens has been reached. In some embodiments, token bucket 620 establishes the maximum size of the token bucket 620 responsive to information from a PLS 640 or a performance level 665. Token bucket 620 may include any functionality, logic, or means for disabling additional tokens 602 from being generated or being added to the token bucket 620 once a threshold is exceeded. In further embodiments, token bucket 620 comprises a limit or a threshold for a minimal number of tokens 602 that may be generated. In such embodiments, token bucket 620 includes any functionality, logic, or means for ensuring that no additional tokens 602 are subtracted or terminated after the threshold has been exceeded. Token bucket 620 may include, keep a track of, or keep a count of any number of tokens 602 that are available. In some embodiments, token bucket 620 subtracts a token 602 for each data packet 601 that is processed, propagated or throttled via a throttler 625. In other embodiments, token bucket 620 adds a token 602 for each data packet 601 that is processed, propagated or throttled via a throttler 625. Token bucket 620 may provide any number of tokens 602 to the throttler 625 in response to the request from the throttler 625 to send the tokens 602. Token bucket 620 may also refuse to provide tokens 602 to the throttler 625 responsive to a rule, logic or threshold limit. Token bucket 620 may use any function, device, unit or an algorithm to maintain and monitor any token 602 or the total number of available tokens 602.

Throttler 625 may include any hardware, software or any combination of hardware and software for establishing, controlling and managing the flow of any data packets 601 that are propagating, processing or throttling via RLM 605. Throttler 625 may include any number of files, programs, applications, functions, algorithms, components, processing units or logic circuits for propagating, processing, throttling or controlling the flow of any data packets 601 according to a throughput rate 650. Throttler 625 may process, propagate, throttle any number of data packets 601 responsive to availability of tokens 602 in a token bucket 620. Data packets 601 to be throttled or processed by the throttler 625 may be stored

in one or more queues. The queues may receive incoming data packets 601 from one or more network interface cards. Throttler 625 may receive incoming data packets 601 from one or more queues and throttle, propagate or process the data packets 601 at the rate limit. Throttler 625 may utilize logic, functions, algorithms or units for determining or monitoring the total number or a total count of tokens 602 available in the token bucket 620. Throttler 625 may comprise any functionality for propagating data packets 601 responsive to availability of tokens 602 in the token bucket 620. In some embodiments, throttler 625 comprises functionality for propagating data packets 601 based on BPS limit 651. In other embodiments, throttler 625 comprises functionality for propagating data packets 601 based on PPS limit 651. In further embodiments, throttler 625 comprises functionality for propagating or throttling data packets 601 based on any type and form of throughput rate 650 or token rate 615. Throttler 625 may include any means or functionality for propagating or throttling data packet 601 based on any combination of availability of tokens 602, throughput rate 650, BPS limit 651, PPS limit 652 and any PLS 640A-N.

Throttler 625 may propagate, process or throttle any number of data packets 601 responsive to availability of tokens 602. In some embodiments, throttler 625 determines to propagate a number of data packets 601 to one or more packet engines 548 on one or more cores 505 in response to a number of tokens 602 being available in the token bucket 620. In some embodiments, throttler 625 determines that a specific number of data packets 601 is waiting at a queue to be propagated to one or more packet engines 548. Throttler 625 may further determine that the total number of tokens 602 available in the token bucket 620 exceeds the number of data packets 601 awaiting the propagation. Throttler 625 may propagate the data packets 601 responsive to the tokens 602 being available for each data packet 601 propagated. In some embodiments, if the number of available tokens 602 does not exceed the number of data packets 601, throttler 625 may not propagate the data packets. In further embodiments, once the data packets 601 are propagated or throttled, the throttler 625 may send a signal to the token bucket 620 to decrease the number of available tokens 602 by the number of data packets 601 propagated. In further embodiments, throttler 625 propagates the data packets 601 one at a time, while waiting to receive a new token 602 from the token generator 610. Token rate 615 at which the token generator 610 generates tokens 602 may determine the throughput rate 650 at which the throttler 625 propagates a data packet 601. Following the propagation of each data packet 601, throttler 625 may send an instruction to the token bucket to count down, decrease or otherwise adjust the number or

tally of the available tokens 602. In other embodiments, token rate 615 at which the token generator 610 generates tokens 602 is determined by the throughput rate 650.

Throttler 625 may control the flow of the data packets 601 by using a token 602 in correspondence to each byte or bit of data packets 601 propagated by the throttler 625. In such embodiments, throttler 625 controls the flow of the data packets 601 based on the number of bytes or bits of the data packets 601 propagated. For example, throttler 625 may throttler or propagate data packets 601 towards one or more PEs 548, responsive to availability of tokens 602 for each byte or bit of data packets 601 propagated. Following the propagation of the data packets 601 based on the number of bits or bytes, the total number of tokens 602 available in the token bucket 620 may decrease or adjust accordingly to reflect the correct total sum of available tokens 602. In some embodiments, throttler 625 may determine that a token bucket 620 comprises no tokens 602. In such embodiments, throttler 625 readies a data packet 601 for propagation and awaits arrival of the next token 602. Upon arrival of the token 602 the next token 602, throttler 625 propagates the data packet 601 . The token 602 may be dropped, discounted or subtracted from the count of the total number of available tokens 602 responsive to the data packet 601 being propagated. Throttler 625 may ready another data packet 601 for transmission and await another available token 602 to implement the propagation. In some embodiments, throttler 625 decides that data packets 601 have been waiting for propagation for a period of time that exceeds a predetermined threshold. Throttler 625 may then drop, flush or erase data packets 601 stored in the queues awaiting the propagation. In some embodiments, throttler 625 sends the data packets 601 whose waiting period has exceeded the threshold to excess handler 630.

Excess handler 630 may include any hardware, software or any combination of hardware and software for controlling and managing data packets 601 sent to the excess handler from the throttler 625. Excess handler 630 may include any number of files, programs, applications, functions, algorithms, components, processing units or logic circuits for propagating, processing, terminating, refreshing or erasing any data packets 601. Excess handler 630 may send, transmit out, reject or erase any number of data packets 601 responsive to instructions from the PLS 640. Excess handler 630 may terminate or flush data packets 601. In some embodiments, excess handler 630 sends the data packets 601 back to the original sender of the data packets 601. In further embodiments, excess handler 630 sends a response to the original sender of the data packets 601 requesting from the sender to resend the data packets 601 again. In further embodiments, excess handler 630 stores the data packets 601 received into a storage or a memory. In still further embodiments, excess

handler 630 reformats or processes the data packets 601 and sends the data packets back to the queue of the throttler 625 for processing. In still further embodiments, excess handler 630 forwards the data packets 601 to an additional throttler 625 that uses an additional set of tokens 602 from another token bucket 620. Following the receipt of the data packets 601 from the excess handler 630, the additional throttler 625 propagates or throttles the data packets to one or more PEs 548 responsive to availability of the additional set of tokens 602 in the another token bucket 620.

Excess handler 630 may process any data packets 601 not throttled by the throttler 625 in accordance with any number of processes and procedures. In some embodiments, excess handler 630 discards the data packets 601 that are not received or processed by the throttler 625. For example, the queues storing data packets 601 may be flushed out if there are no tokens 602 for processing the data packets 601. In further embodiments, excess handler 630 stores or maintains excess data packets 601 until the tokens 602 become available. In still further embodiments, excess handler 630 maintains another token bucket for handling excess data packets 601. Excess handler 630 may use active queue management to handle any data packets 601 that are not processed, throttled or propagated by the throttler 625. Excess handler 630 may include an algorithm or a function to use one or more proportional integrals to calculate the number of data packets 601 to be flushed or handled in an alternative matter, such as the additional token bucket. Excess handler 630 may use probability functions or algorithms to calculate the probability of the data packets 601 being dropped or flushed from the queues. Active queue management may also employ current queue length, size of data packets, the number of data packets, token and throughput rates and BPS and PPS limits to compute the probability of data packets 601 being dropped or flushed. In some embodiments, active queue management may use current queue length, size of data packets, the number of data packets, token and throughput rates and BPS and PPS limits to compute the probability to perform additional processes, such as additional token bucket for processing the non-throttled data packets 601. Active queue management may determine to proceed with processing of the data packets 601 for which the probability of being dropped exceeds a predetermined threshold.

Performance level settings 640, also referred to as PLS 640, may include any hardware, software or any combination of hardware and software for setting or configuring operation of the appliance 200. PLS 640 may include any number of files, scripts, programs, applications, functions, algorithms, processing units, logic circuits, analog or digital circuits or executables for configuring or setting operation or functionality of any number of

components of the appliance 200. PLS 640 may include any functionality for configuring or setting the performance level or operation of the appliance 200 in accordance with information identified by the performance level 665. In some embodiments, PLS 640 includes a compilation of configuration and operation settings for configuring or maintaining the rate of flow of the data packets 601 traversing the appliance at a predetermined level. PLS 640 may comprise one or more settings, parameters input values, instructions and commands for one or more components of the intermediary 200 or the RLM 605. In some embodiments, PLS 640 includes parameters, inputs, instructions and settings for any one of, or any combination of, the token generator 610, token rate 615, throttler 625, token bucket 620 and excess handler 630. The parameters, inputs, instructions and settings may include any combination of values, configuration points and commands for any number of components of the RLM 605 to maintain a rate of flow of data packets 601 within a predetermined level or threshold.

PLS 640 may include any type and form of functionality for storing, identifying, setting and configuring any parameters, settings and instructions for any part or component of the RLM 605. In some embodiments, PLS 640 includes settings, parameters and instructions for a token generator 610 to generate tokens 602 at a predetermined rate. In further embodiments, PLS 640 includes settings, parameters and instructions identifying or specifying a token rate 615. PLS 640 may include settings, parameters and instructions for specifying or identifying a type of tokens 602 to be generated. In some embodiments, PLS 640 includes settings, parameters and instructions for generating tokens 602 for data packets 601. In some embodiments, PLS 640 includes settings, parameters and instructions for generating tokens 602 for data bytes or data bits of the data packets 601 . PLS 640 may include settings, parameters and instructions for initiating, generating and maintaining a token bucket 620 which may include a maximum token size of any number of tokens 602. In some embodiments, PLS 640 includes settings, parameters and instructions to establish and maintain a token bucket 620 that comprises any number of tokens 602, such as anywhere between 100 and 1000, 1000 and 100000 or 100000 and 10,000,000 tokens. In some embodiments, PLS 640 includes settings, parameters and instructions that generate a throttle 625. In further embodiments, PLS 640 includes settings, parameters and instructions that set up, initiate and maintain the operation of throttle 625 to throttle or control rate or flow of data packets 601 at any rate or speed. In some embodiments, PLS 640 includes settings, parameters and instructions that initiate, establish, control and maintain an excess handler 630. In further embodiments, PLS 640 includes settings, parameters and instructions that

control and maintain operation of excess handler 630 to handle, operate on or process any data packets 601 that are not processed or throttled by throttler 625.

PLS 640 may further include any additional settings, instructions or parameters for using additional methods for controlling of rate of propagation or processing. In some embodiments, PLS 640 includes settings and instructions for limiting amount of memory visible to the system, or the RLM 605. In further embodiments, PLS 640 includes settings and instructions for limiting a number of cores 505 available to the system or the RLM 605. In still further embodiments, PLS 640 includes settings and instructions for limiting the number of SSL chips visible to the system. In yet further embodiments, PLS 640 includes settings and instructions for adjusting a clock for running of the processors, such as CPUs or the processors used by the RLM 605. In still further embodiments, PLS 640 includes settings and instructions for managing processor cache-miss rate. In still further embodiments, PLS 640 includes settings and instructions for tweaking or fine-tuning of the netio pipeline parameters. In yet further embodiments, PLS 640 includes settings and instructions for running or operating RLM 605 on a plurality of cores 505. In still further embodiments, PLS 640 includes settings and instructions for running or operating RLM 605 on a single-processor (single-core) system.

Rate limit settings 645 may include any hardware, software or any combination of hardware and software for setting or configuring rate of flow of data packets 601. Rate limit settings 645 may include any number of files, scripts, programs, applications, functions, algorithms, processing units, logic circuits, analog or digital circuits or executables for configuring or setting rate of flow or rate of processing of data packets 601. Rate limit settings 645 may include any type and form of settings, configuration points or setting points for any number of components of the RLM 605, such as the throttler 625, for controlling or limiting rate of flow or rate of propagation of data packets 601. In some embodiments, rate limit settings 645 include any number of configuration and operation settings for establishing and operating a token generator 610. In further embodiments, rate limit settings 645 include any number of configuration and operation settings for establishing a token rate 615. In yet further embodiments, rate limit settings 645 include any number of configuration and operation settings for establishing and operating a throttler 625. In still further embodiments, rate limit settings 645 include any number of configuration and operation settings for establishing and operating an excess handler 630. Rate limit settings 645 may include any number files, instructions, data, applications, processing units, hardware or software for configuring, establishing and operating any of the RLM 605 components to maintain a rate of

flow or propagation of the data 601 through the throttler 625 within performance level settings identified by the performance level 665 of the rate limiting license 660.

RLS 465 may include any type and form of configuration or operation instructions, parameters or settings. In some embodiments, RLS 465 sets the limits or thresholds for any of the throughput rate 650 or token rate 615 based on the hardware platform of the model of the appliance 200. In further embodiments, RLS 465 sets the limits or thresholds of the throughput and token rates at a minimum or the slowest rate settings if a performance level 665 is not identified.

RLS 465 may configure a rate limit using a setting, such as: **netscaler.do_rate_limit=l.** In such embodiments, setting the netscaler.do_rate_limit variable to non-zero activates or enables the rate limiting settings. In other embodiments, if the setting is at a zero, the rate limiting setting or code is not active. default value is zero, and means, that rate limiting code is not active.

RLS 465 may configure a size of a token bucket 620 using another setting, such as: **netscaler.rate_limit_bucket_size=1000.** In such embodiments, the size of the token bucket 620 is set in milliseconds. This value may determine size of the maximum burst in traffic which will be able to pass through throttler 625 without restrictions. This value may identify a maximum burst that is allowed to propagate or be received by the throttler 625.

RLS 465 may configure a limit for a throughput rate 650 using a setting such as: **netscaler.rate_limit_mbits=3072.** In such embodiments, the limit or the threshold for throughput rate is defined in Megabits per second, such as 3072 Mb/s, or 3Gb/s.

RLS 465 may configure a packet rate limit in packets per second using a setting, such as: **netscaler.rate_limit_packets=1000000.** In such embodiments, packet rate limit is defined as 1000000 packets per second.

RLS 465 may also allow confirmation of the values set. Such confirmations may be initiated using an instruction, such as: ***dmesg \grep*** *platform*

RLS 465 may further configure or set rate limiting parameters, using instructions, such as: **nsapimgr -B "w ns_rl_bucket_size 0x400"** - for setting a token bucket 620 size in milliseconds using hexadecimal values, **nsapimgr -B "w ns_rl_mbits OxCOO"** - for setting throughput rate in megabits per second, using hexadecimal values, and **nsapimgr** -B "w **ns_rl_packets** 0xF4240" - for setting a packet rate in packets per second, also using hexadecimal values.

Bucket settings 646 may include any hardware, software or any combination of hardware and software for setting or configuring of token bucket 620. Bucket settings 646

may include any number of files, scripts, programs, applications, functions, algorithms, processing units, logic circuits, analog or digital circuits or executables for configuring or setting any components features or functions of the token bucket 620. Bucket settings 646 may configure or set up a type or operation of the token bucket 620. In some embodiments, bucket settings 646 configure or set up the token bucket 620 as a bucket that stores a predetermined amount of tokens 602. Bucket settings 646 may configure or set up the token bucket 620 to enable a burst of data having a number of data packets 601 which does not exceed the number of tokens 602 stored in the token bucket 620 to be throttled or processed by the throttler 625 without slowing the data 601 down. In further embodiments, bucket settings 646 may maintain the rate of generating tokens 602 by the token generator 610 at a predetermined token rate 615. In some embodiments, token rate 615 may be any rate of generating tokens 602, such as 10, 50, 100, 500, 1000, 2000, 5000, 7000, 10000, 15000, 20000, 30000, 50000, 100000 or 1000000 tokens/second. In some embodiments, bucket settings 646 configure or set up the token bucket 620 as a bucket that does not store a predetermined amount of tokens 602. Instead, token bucket 620 may be set by the bucket settings 646 to simply hold a token 602 for a predetermined amount of time. The token bucket 620 may be configured to drop the token 602 after the predetermined amount of time expires and wait for the next token 602. Bucket settings 646 may configure or set up the token bucket 620 not to enable a burst of data greater than a predetermined rate limit of data to be throttled. Instead, bucket settings 646 may set up the token bucket to generate tokens 602 at a predetermined rate to ensure that data packets 601 are throttled or processed by the throttler 625 at the predetermined rate limit, such as the throughput rate.

Throughput rate 650 may comprise any limit, threshold or a configuration setting for a rate of processing or throttling of data packets 601 traversing the appliance 200. Throughput rate 650 may include a rate or propagation in packets per second or bytes per second of data packets 601. Throughput rate 650 may include any hardware, software or any combination of hardware and software for setting or configuring of token bucket 620. Throughput rate 650 may include any number of files, scripts, programs, applications, functions, algorithms, processing units, logic circuits, analog or digital circuits or executables for configuring or setting rate of processing or throttling of data packets 601. In some embodiments, throughput rate 650 includes a threshold for a rate of propagation of data packets 601. In some embodiments, throughput rate 650 is identified in terms of data packets 601 to be processed, propagated or throttled per second. In other embodiments, throughput rate 650 is identified in terms of a number of packets or chunks of data packets 601 to be processed, propagated or

throttled per second. In further embodiments, throughput rate 650 is identified in terms of a number of bits of data packets 601 to be propagated, processed or throttled per second. In yet further embodiments, throughput rate 650 is identified in terms of a number of requests of a client 102 to be propagated, processed or throttled per second. In still further embodiments, throughput rate 650 is identified in terms of a number of responses of a server 106 to be propagated, processed or throttled per second. In yet further embodiments, throughput rate 650 is identified in terms of a number of transmissions for a specific destination to be processed, propagated or throttled per second. In still further embodiments, throughput rate 650 is identified in terms of a number of transmission from a specific source to be processed, propagated or throttled per second. In yet further embodiments, throughput rate 650 is identified in terms of a number of data packets 601, data bits, data bytes or transmissions to be throttled, processed or propagated and forwarded to a specific PE 548 or a specific core 505 of the appliance 200. Throughput rate 650 may include any type and form of propagation rate for data packets 601 traversing the appliance 200.

Bytes per second limit 651, also referred to as BPS limit 651, may comprise any limit, threshold or a configuration setting in bytes per second for a rate of processing or throttling of data packets 601. BPS limit 651 may include any rate of propagation in bytes per second. BPS limit 651 may include any limit or threshold for a maximum rate of propagation in bytes per second. In some embodiments, BPS limit 651 includes or identifies any rate between 1 byte per second and 1 terabyte per second. In some embodiments, BPS limit 651 includes any range of rates between 1 and 100 bytes per second, such as 1, 10, 20, 30, 40, 50, 60, 70, 80, 90 or 100 bytes per second. In other embodiments, BPS limit 651 includes any range of rates between 100 and 1000 bytes per second, such as 100, 150, 200, 250, 300, 350, 400, 450, 500, 550, 600, 650, 700, 750, 800, 850, 900, 950 or 1000 bytes per second. In further embodiments, BPS limit 651 includes any range of rates between 1000 and 10000 bytes per second, such as 1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500, 5000, 5500, 6000, 6500, 7000, 7500, 8000, 8500, 9000, 9500 or 10000 bytes per second. In yet further embodiments, BPS limit 651 includes any range of rates between 10,000 and 100000 bytes per second, such as 10000, 15000, 20000, 25000, 30000, 35000, 40000, 45000, 50000, 55000, 60000, 65000, 70000, 75000, 80000, 85000, 90000, 95000 or 100000 bytes per second. In still further embodiments, BPS limit 651 includes any range of rates between 100,000 and 1,000,000 bytes per second, such as 100000, 150000, 200000, 250000, 300000, 350000, 400000, 450000, 500000, 550000, 600000, 650000, 700000, 750000, 800000, 850000, 900000, 950000 or 1000000 bytes per second. In yet further embodiments, BPS limit 651 includes

any range of rates between 1,000,000 and 1,000,000,000 bytes per second, such as 1,000,000, 5,000,000, 10,000,000, 50,000,000, 100,000,000, 500,000,000 or 1,000,000,000 bytes per second. Throughput rate 650 may include any hardware, software or any combination of hardware and software for setting or configuring of token bucket 620.

Packets per second limit 652, also referred to as PPS limit 652, may comprise any limit, threshold or a configuration setting in packets per second for a rate of processing or throttling of data packets 601. PPS limit 652 may include any rate of propagation in packets per second. PPS limit 652 may include any limit or threshold for a maximum rate of propagation in packets per second. In some embodiments, PPS limit 652 includes or identifies any rate between 1 packet per second and 1,000,000,000 packets per second. In some embodiments, PPS limit 652 includes any range of rates between 1 and 100 packets per second, such as 1, 10, 20, 30, 40, 50, 60, 70, 80, 90 or 100 packets per second. In other embodiments, PPS limit 652 includes any range of rates between 100 and 1000 packets per second, such as 100, 150, 200, 250, 300, 350, 400, 450, 500, 550, 600, 650, 700, 750, 800, 850, 900, 950 or 1000 packets per second. In further embodiments, PPS limit 652 includes any range of rates between 1000 and 10000 packets per second, such as 1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500, 5000, 5500, 6000, 6500, 7000, 7500, 8000, 8500, 9000, 9500 or 10000 packets per second. In yet further embodiments, PPS limit 652 includes any range of rates between 10,000 and 100000 packets per second, such as 10000, 15000, 20000, 25000, 30000, 35000, 40000, 45000, 50000, 55000, 60000, 65000, 70000, 75000, 80000, 85000, 90000, 95000 or 100000 packets per second. In still further embodiments, PPS limit 652 includes any range of rates between 100,000 and 1,000,000 packets per second, such as 100000, 150000, 200000, 250000, 300000, 350000, 400000, 450000, 500000, 550000, 600000, 650000, 700000, 750000, 800000, 850000, 900000, 950000 or 1000000 packets per second. In yet further embodiments, PPS limit 652 includes any range of rates between 1,000,000 and 1,000,000,000 packets per second, such as 1,000,000, 5,000,000, 10,000,000, 50,000,000, 100,000,000, 500,000,000 or 1,000,000,000 packets per second.

Referring now to FIG. 6B, embodiments of steps of a method for controlling a rate of traffic of a device in accordance with a rate limit identified by a rate limiting license is illustrated. In brief overview, at step 605 a rate limiting manager 605 of an intermediary 200 identifies presence of a rate limiting license 660 that identifies a performance level 665. At step 610, the rate limiting manager 605 establishes a rate limit, such as a token rate 615, based on the performance level 665. At step 615, a token generator 610 generates tokens 602 for a token bucket 620 in accordance with the rate limit. At step 620, the intermediary 200

receives a plurality of network packets, such as data packets 601. At step 625, a throttler 625 identifies, from the token bucket 620, tokens 602 for the plurality of network packets. At step 630, the throttler 630 controls a rate of receiving of the network packets, such as the throughput rate 650, based on the rate limit. At step 635, the rate limiting manager 605 transmits the throttled network packets to one or more packet engines 548 and transmits the network packets that were not throttled to an excess handler 630. At step 640, the rate limiting manager 605 transmits the network packets that were not throttled to an excess handler 630.

In further overview of FIG. 6B, at step 605 a rate limiting manager 605 of an intermediary 200 identifies presence of a rate limiting license 660 which includes an information about a performance of the intermediary 200. In some embodiments, a rate limiting manager (RLM) 605 of the intermediary 200 identifies a presence of a rate limiting license 660. In other embodiments, RLM 605 identifies a component of a rate limiting license 660. In yet further embodiments, RLM 605 receives a file or a message from the rate limiting license 660 comprising information about a performance of the intermediary 220. Rate limiting license 660 may sand to the RLM 605 any information about a performance level 665 for the appliance 200. The information about the performance level 665 may include any information about throughput rate 650 or a propagation or throttling rate of data packets 601 . The information identifying the performance level 665 may include any number, value or a parameter uniquely identifying the performance level 665 from any other performance level 665. The performance level 665 may be matched by the RLM 605 with a corresponding performance level settings 640. The performance level 665 may include any information regarding the rate of throughput, propagation, throttling or processing of the data packets 601 by the intermediary 200. In some embodiments, performance level 665 includes a maximum threshold rate of throughput or propagation for processing or throttling data packets 601 via the throttler 625. In further embodiments, performance level 665 includes information identifying the rate of generating tokens 602. In still further embodiments, performance level 665 includes information identifying the maximum number of tokens 602 to be stored in a token bucket 620.

At step 610, the rate limiting manager 605 establishes a rate limit based on the performance level 665. RLM 605 may establish a rate limit based on the PLS 640 that is identified by the information from the performance level 665. In some embodiments, PLS 640 generates configuration and operation settings for the RLM based on the information from the performance level 665. RLM 605 may establish or determine a rate limit responsive

to configuration and operation settings from the PLS 640. RLM 605 may establishes any rate limiting or placing a threshold for controlling throughput, propagation or throttling of data packets 601. In some embodiments, RLM 605 establishes a throughput rate 650. In other embodiments, RLM 605 establishes a token rate 615. In further embodiments, RLM 605 establishes a BPS limit 651. In yet further embodiments, RLM 605 establishes a PPS limit 652. In further embodiments, RLM 605 establishes one or more bucket settings 646 for a token bucket 620. In some embodiments, RLM 605 establishes a maximum token 602 number to be allowed by the token bucket 620. RLM 605 may utilize PLS 640 to identify or establish any rates or rate limits for the RLM 605. In some embodiments, RLM 605 identifies or establishes a maximum or a minimum threshold or limit for a token rate 615. In other embodiments, RLM 605 identifies or establishes a maximum or a minimum threshold or limit for a throughput rate 615.

At step 615, a token generator 610 generates tokens 602 for a token bucket 620 in accordance with the rate limit. In some embodiments, token generator 610 generates tokens 602 in accordance with, or based on, the throughput rate 650. In other embodiments, token generator 610 generates tokens 602 in accordance with, or based on, the token rate 615. In further embodiments, token generator 610 generates tokens 602 in accordance with, or based on, the BPS limit 651. In yet further embodiments, token generator 610 generates tokens 602 in accordance with, or based on, PPS limit 652. In still further embodiments, token generator 610 generates tokens 602 in accordance with, or based on, a PLS 640. In some embodiments, token generator 610 generates tokens 602 in accordance with, or based on, information from the performance level 665. In yet further embodiments, token generator 610 generates tokens 602 in accordance with, or based on, the information about hardware platform for the appliance 200. In still further embodiments, token generator 610 generates tokens 602 in accordance with, or based on, bucket settings 645, such as a token bucket 620 size limit. Token generator 610 may generate tokens 602 responsive to a type of data packets 601 traversing the appliance 200. Token generator 610 may generate tokens 602 responsive to any information from any of the RLM 605 components, such as the PLS 640, token bucket 620, throttler 625 or an excess handler 630.

At step 620, the intermediary 200 receives a plurality of network packets, such as data packets 601. In some embodiments, the intermediary 200 receives one or more requests from a client 102. In other embodiments, the intermediary 200 receives one or more responses to client 102 requests from a server 106. In further embodiments, the intermediary 200 receives one or more data bits or data bytes. In still further embodiments, the intermediary 200

receives one or more streams of data, such as stream data of audio or video streams. In some embodiments, the intermediary 200 receives one or more data packets 601. In further embodiments, the intermediary 200 receives a network data packet, such as a data packet traversing the network 104. The received plurality of network data packets may be received by the intermediary 200 and stored in one or more queues, registers or storages. The received plurality of network data packets may be forwarded to the throttler 625 for further processing, propagating or forwarding.

At step 625, a throttler 625 identifies, from a token bucket 620, tokens 602 for the plurality of network packets. In some embodiments, throttler 625 identifies a number of tokens 602 available in the token bucket 620. In other embodiments, throttler 625 identifies specific tokens 602 to be used for processing or propagating specific data packets 601. In further embodiments, throttler 625 identifies a current count or sum of the tokens 602 available. In further embodiments, throttler 625 requests from the token bucket 620 a total sum of tokens 602 currently available. Token bucket 620 may respond to the throttler 625 with a response identifying the total sum or a total number of currently available tokens 602. In some embodiments, throttler 625 identifies if there is at least one token 602 available in the token bucket 620. In further embodiments, throttler 625 identifies if there is at least one token 602 above a minimum threshold for the number of tokens available in the token bucket 620. Throttler 625 may identify each token 602 for each data packet 601 awaiting the propagation or processing. In some embodiments, throttler 625 identifies specific tokens 602 for specific data packets 601 based on the type of tokens 602 and types of data packets 601. Throttler 625 may assign one or more tokens 602 for one or more network packets, such as data packets 601. In some embodiments, throttler 625 assigns one or more tokens 602 from the token bucket 620 to a data packet 601. In other embodiments, throttler 625 assigns a token 602 for one or more data packets 601. In further embodiments, throttler 625 assigns a token for each predetermined amount of bits, bytes or megabytes of the network packets or data packets 601. In still further embodiments, throttler 625 assigns one or more tokens for each bit, byte or megabyte of the network traffic or data packets 601.

At step 630, throttler 625 controls a rate of receiving, propagating or throttling of network packets based on any rate limit. In some embodiments, throttler 625 controls a rate of receiving or propagating of network packets, such as the data packets 601, based on the established rate limit. In some embodiments, throttler 625 controls a rate of receiving, propagating or throttling of the data packets 601 based on the throughput rate 650. In other embodiments, throttler 625 controls a rate of receiving, propagating or throttling of the data

packets 601 based on the bytes per second (BPS) limit 651. In other embodiments, throttler 625 controls a rate of receiving, propagating or throttling of the data packets 601 based on the packet per second (PPS) limit 652. In further embodiments, throttler 625 controls a rate of receiving, propagating or throttling of the data packets 601 based on the combination of BPS and PPS limits. In some embodiments, a throttler 625 receives, throttles or propagates data packets 601 based on a rate that does not exceed a predetermined packets per second limit 652 in addition to not exceeding another predetermined bytes per second limit 651. A throttle 625 may propagate, process or receive data packets 601 based on any rate of bytes per second or packets per second provided that the rate does not exceed a BPS or PPS limit. In some embodiments, throttle 625 propagates, processes or receives data packets 601 based on any rate of bytes per second that does not exceed a bytes per second limit 652. In still further embodiments, throttle 625 propagates, processes or receives data packets 601 based on a token rate 615. The token rate 615 may further be based on any one of the BPS limit 651 or PPS limit 652. The token rate 615 may also be based on the combination of BPS limit 651 and PPS limit 652. Throttler 625 may throttle, propagate or receive the network packets at any rate based on any combination of any of the token bucket 620 maximum size, token rate 615, throughput rate 650, BPS limit 651 and PPS limit 652.

At step 635, the rate limiting manager 605 transmits data packets 601 that were received, propagated or throttled by the throttler 625 to one or more packet engines 548. In some embodiments, RLM 605 transmits the data packets 601 from the throttler 625 to a PE 548. In other embodiments, RLM 605 transmits some data packets 601 to a PE 548 identified by the data packets 601. In further embodiments, RLM 605 transmits subsets of data packets 601 to some specific or predetermined PEs 548. The subsets of data packets 601 may include any number of data packets. Such data packets may be distributed across any number of PEs 548 operating on any number of cores 505. In further embodiments, RLM 605 distributes the data packets 601 to the intended or packet engines 548 based on the information from the data packets 601 or from the appliance 200.

At step 640, the rate limiting manager 605 transmits data packets 601 that were not received, propagated or throttled by the throttler 625 to an excess handler 630. In some embodiments, RLM 605 determines that one or more data packets 601 are pending at the throttler 625 for a period of time that exceeds a predetermined threshold. RLM 605 may transmit or forward the one or more data packets to the excess handler 630 based on the determination. In some embodiments throttler 623 determines that some data packets 601 need to be forwarded to the excess handler 630. In further embodiments, excess handler 630

monitors performance of the throttler 625 and determines which data packets 601 need to be processed by the excess handler 630. In some embodiments, data packets 601 that were not received or propagated by the throttler 625 are sent to the excess handler 630 for discarding or erasing. In further embodiments, data packets 601 that were not received or propagated by the throttler 625 are sent to the excess handler 630 for further processing or analyzing. In still further embodiments, data packets 601 that were not received or propagated by the throttler 625 are sent to the excess handler 630 which notifies the sender of the data packets 601 that data packets 601 are not received. Excess handler 630 may request the sender of the data packets 601 to resend the data packets that were received by the handler 625.

We Claim:

1. A method for controlling a rate of a traffic of a device in accordance with a rate limit identified by a rate limiting license, the method comprising:

a) identifying, by a rate limiting manager of an intermediary device, presence of a rate limiting license, the intermediary device processing network traffic between a plurality of clients and a plurality of servers, the rate limiting license identifying a performance level;

b) establishing, by the rate limiting manager, a rate limit based on the performance level of the rate limiting license; and

c) controlling, by a throttler of the intermediary, a rate of receiving network packets in accordance with the rate limit.

2. The method of claim 1, wherein step (a) further comprises identifying, by the rate limiting manager, the rate limiting license is not present, and wherein step (b) comprises establishing a set of one or more rate limit parameters for the rate limit for a lower performance level.

3. The method of claim 1, wherein step (a) further comprises identifying, by the rate limiting manager, a type of hardware platform of the intermediary device, and wherein step (b) further comprises establishing the rate limit based on the type of hardware platform and the performance level.

4. The method of claim 1, wherein step (b) further comprises establishing, by the rate limiting manager, a maximum size of a token bucket in milliseconds based on the rate limit for the performance level of the rate limiting license.

5. The method of claim 4, wherein step (c) further comprises receiving, by the throttler, a network packet, determining that the token bucket has reached the maximum size and discarding the network packet in response to the determination.

6. The method of claim 1, wherein step (b) further comprises establishing, by the rate limiting manager, a throughput rate limit in bits per second based on the rate limit for the performance level of the rate limiting license.

7. The method of claim 6, wherein step (c) further comprises generating, by a token generator, a token for a token bucket at a rate specified by the throughput rate limit.

8. The method of claim 1, wherein step (b) further comprises establishing, by the rate limiting manager, a packet rate in packets per second based on the rate limit for the performance level of the rate limiting license.

9. The method of claim 8, wherein step (c) further comprises receiving, by the throttler, a network packet having a number of bytes, and removing, by the throttler, a number of tokens from a token bucket equal to the number of bytes.

10. The method of claim 8, wherein step (c) further comprises receiving, by the throttler, a network packet having a number of bytes, determining, by the throttler, that a number of tokens in a token bucket is less than the number of bytes and not removing a token from the token bucket.

11. The method of claim 10, further comprises providing, by the throttler, the network packet to an excess packet handler.

12. A system for controlling a rate of a traffic of a device in accordance with a rate limit identified by a rate limiting license, the system comprising:

a rate limiting manager of an intermediary device identifying presence of a rate limiting license, the intermediary device processing network traffic between a plurality of clients and a plurality of servers, the rate limiting license identifying a performance level;

the rate limiting manager establishing a rate limit based on the performance level of the rate limiting license; and

a throttler of the intermediary controlling a rate of receiving network packets in accordance with the rate limit.

13. The system of claim 12, further comprising the rate limiting manager identifying the rate limiting license is not present and establishing a set of one or more rate limit parameters for the rate limit for a lower performance level.

14. The system of claim 12, further comprising the rate limiting manager identifying a type of hardware platform of the intermediary device and establishing the rate limit based on the type of hardware platform and the performance level.

15. The system of claim 12, further comprising the rate limiting manager establishing a maximum size of a token bucket in milliseconds based on the rate limit for the performance level of the rate limiting license.

16. The system of claim 15, further comprising the throttler receiving a network packet, determining that the token bucket has reached the maximum size and discarding the network packet in response to the determination.

17. The system of claim 12, further comprising the rate limiting manager establishing a throughput rate limit in bits per second based on the rate limit for the performance level of the rate limiting license.

18. The system of claim 17, further comprising a token generator generating a token for a token bucket at a rate specified by the throughput rate limit.

19. The system of claim 12, further comprising the rate limiting manager establishing a packet rate in packets per second based on the rate limit for the performance level of the rate limiting license.

20. The system of claim 19, further comprising the throttler receiving a network packet having a number of bytes and removing a number of tokens from a token bucket equal to the number of bytes.

21. The system of claim 19, further comprising the throttler receiving a network packet having a number of bytes, determining that a number of tokens in a token bucket is less than the number of bytes and not removing a token from the token bucket.

22. The system of claim 21, further comprising the throttler providing the network packet to an excess packet handler.

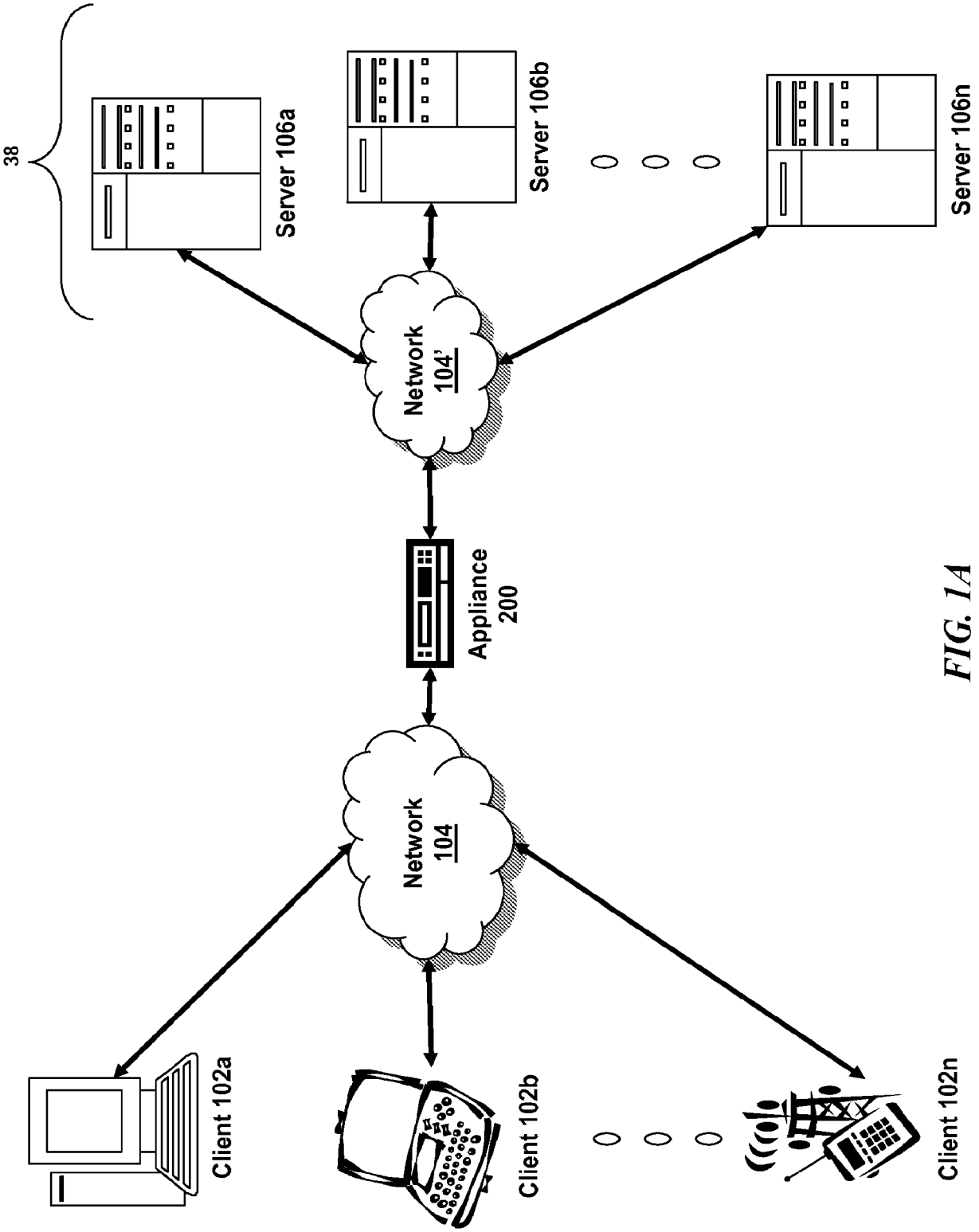
38
Server 106a
Server 106b
Server 106n
Network 104'
Appliance 200
Network 104
Client 102a
Client 102b
Client 102n

FIG. 1A

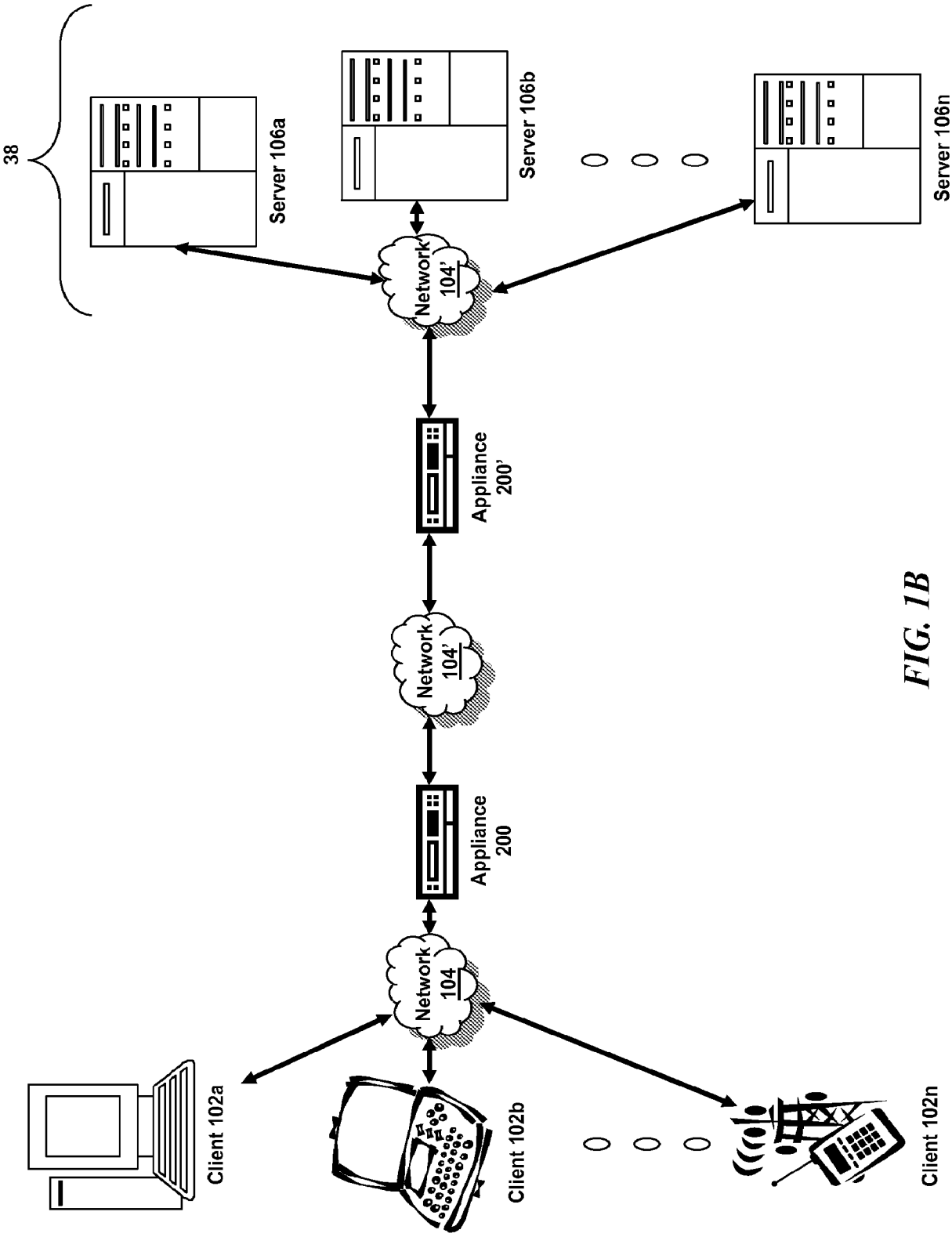
38
Server 106a
Server 106b
Server 106n
Network 104'
Appliance 200'
Network 104'
Appliance 200
Network 104
Client 102a
Client 102b
Client 102n

FIG. 1B

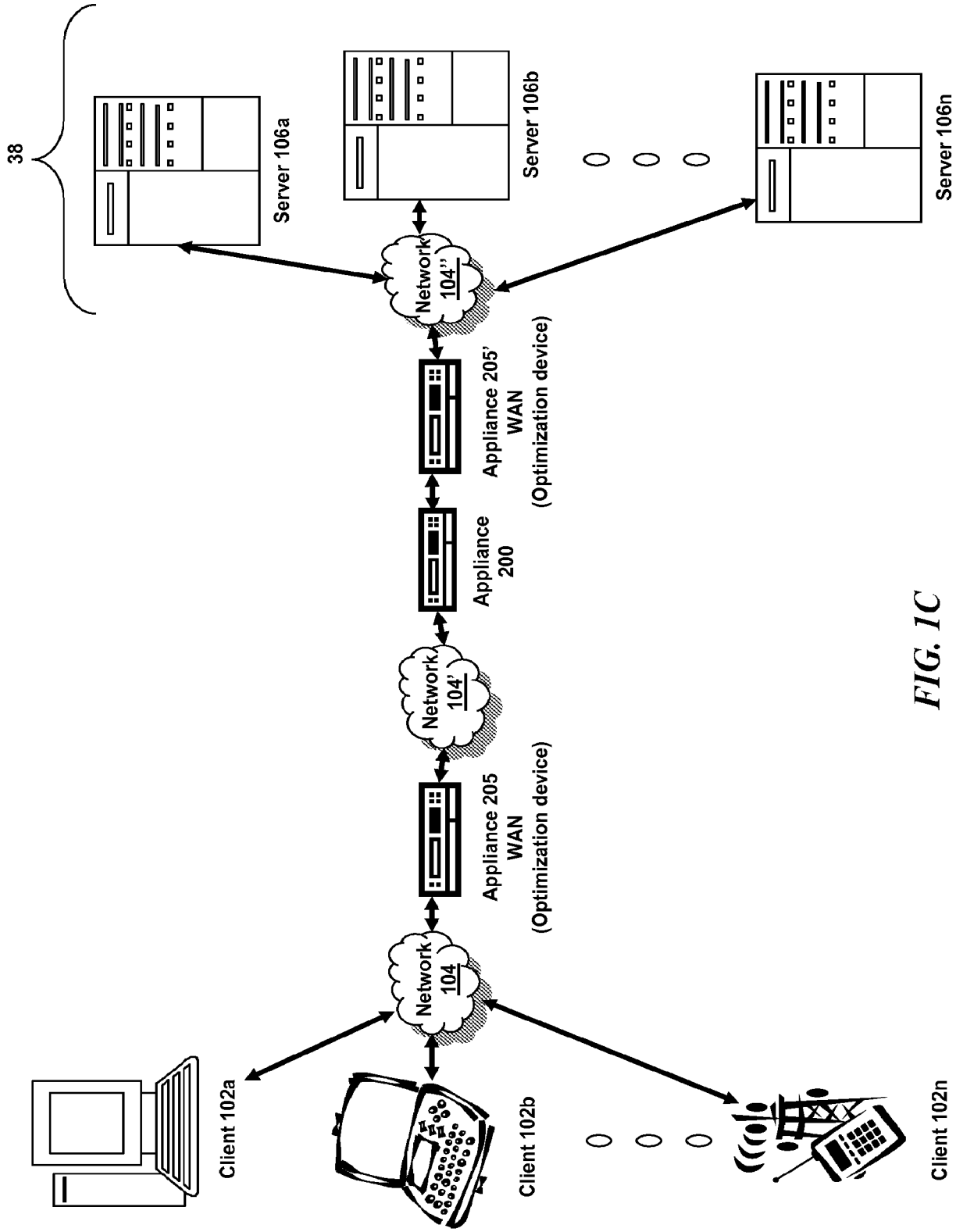
38
Server 106a
Server 106b
Server 106n
Network 104''
Appliance 205' WAN (Optimization device)
Appliance 200
Network 104'
Appliance 205 WAN (Optimization device)
Network 104
Client 102a
Client 102b
Client 102n

FIG. 1C

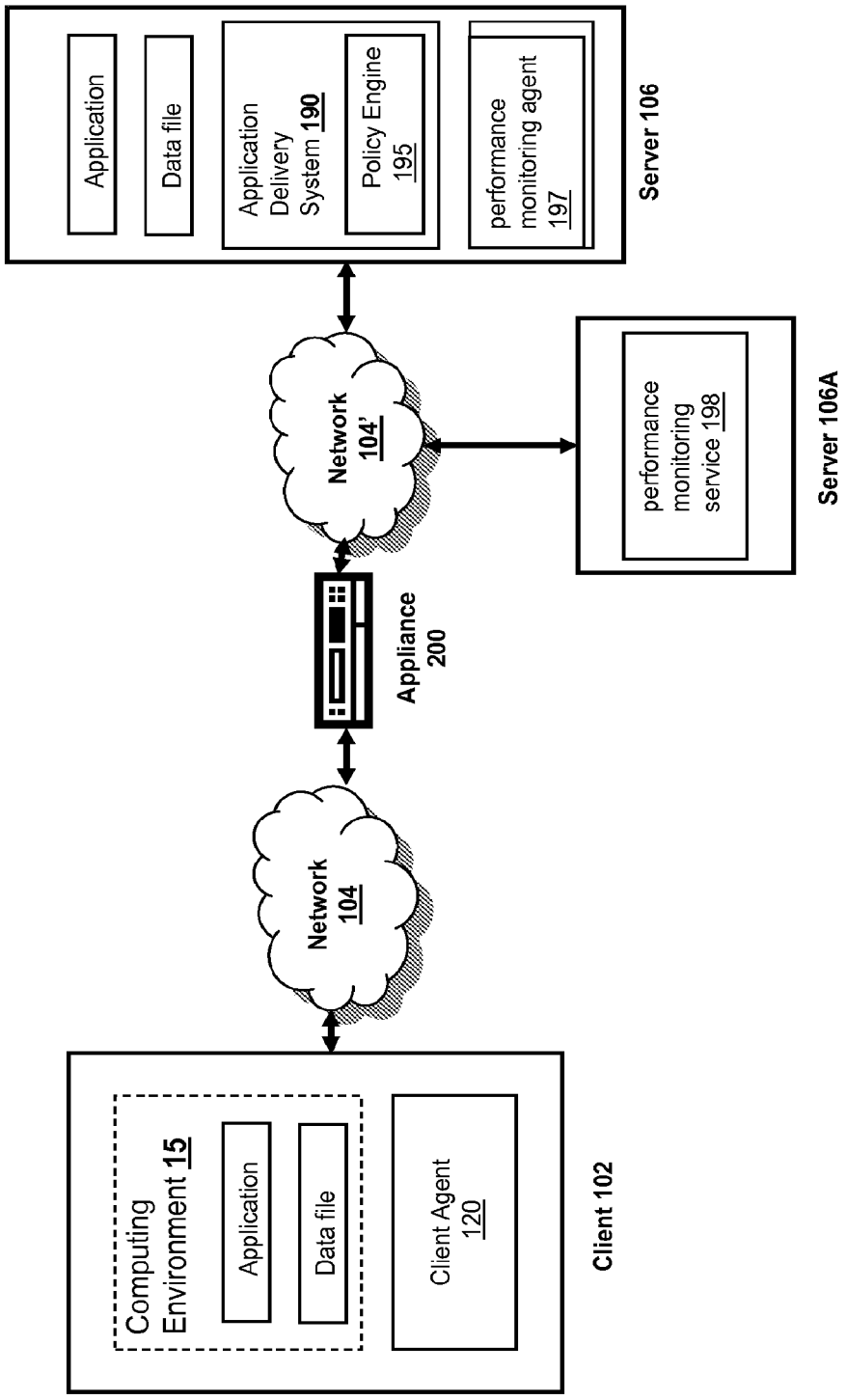
Application
Data file
Application Delivery System 190
Policy Engine 195
performance monitoring agent 197
Server 106
Network 104'
performance monitoring service 198
Server 106A
Appliance 200
Network 104
Computing Environment 15
Application
Data file
Client Agent 120
Client 102

FIG. 1D

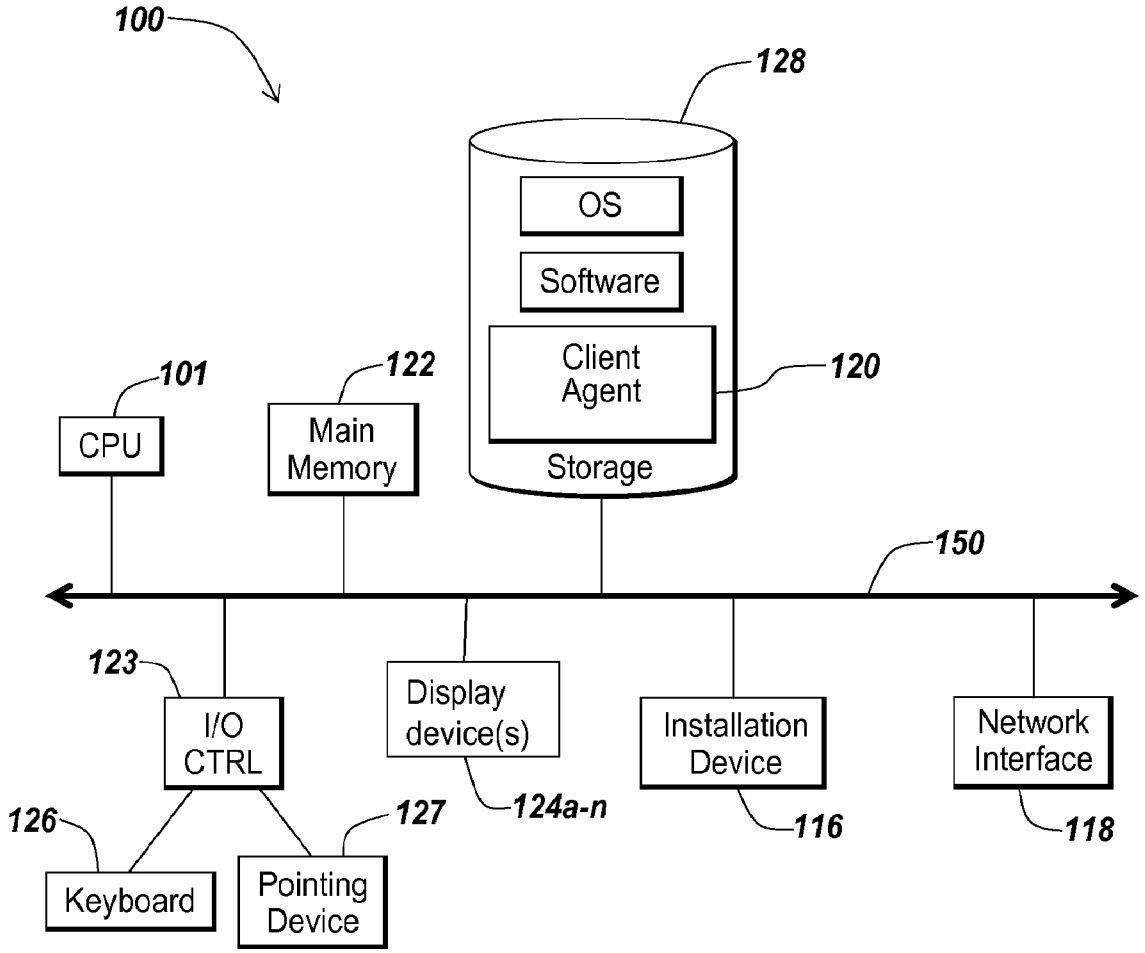
100
128
OS
Software
Client Agent
120
Storage
101
CPU
122
Main Memory
150
123
I/O CTRL
Display device(s)
124a-n
Installation Device
116
Network Interface
118
126
Keyboard
127
Pointing Device

FIG. 1E

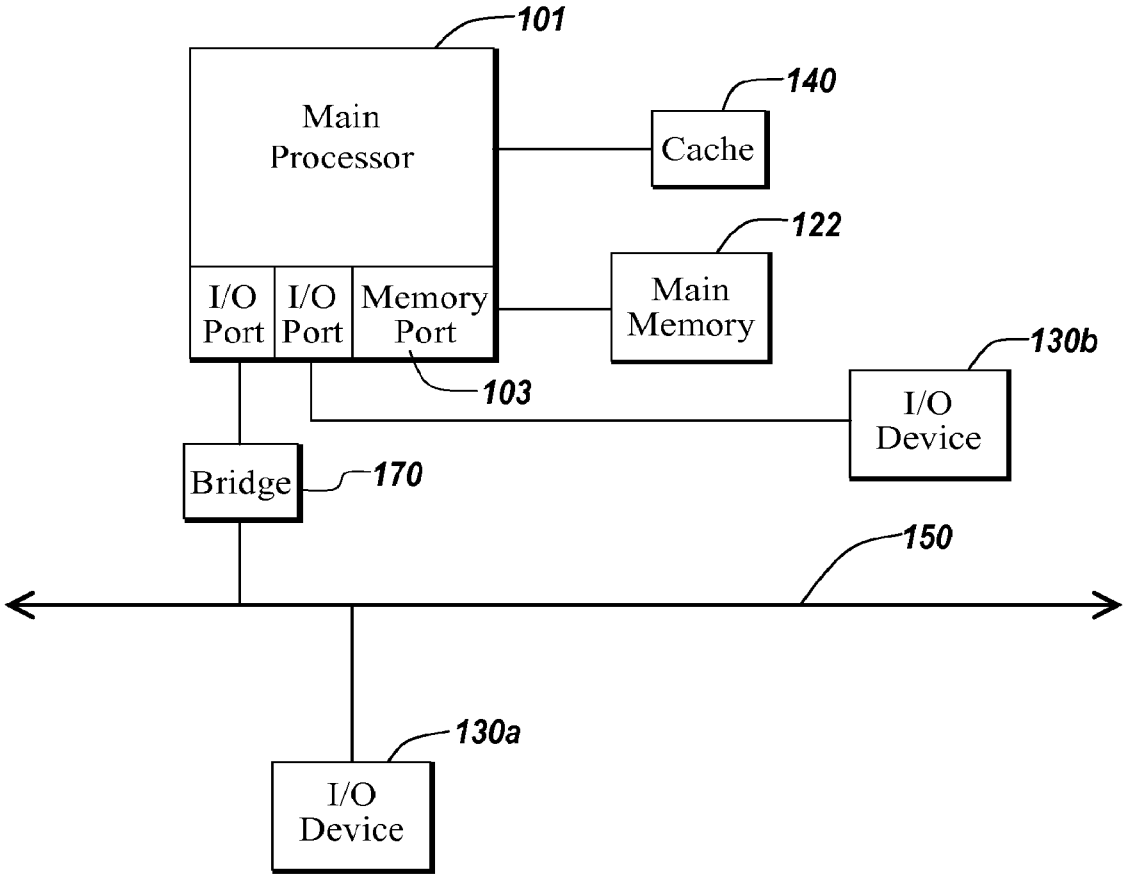
101
Main Processor
140
Cache
122
Main Memory
I/O Port
I/O Port
Memory Port
103
130b
I/O Device
Bridge
170
150
130a
I/O Device

FIG. 1F

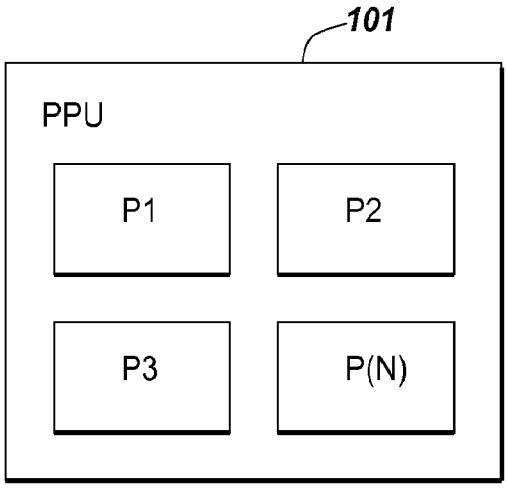

FIG. 1G

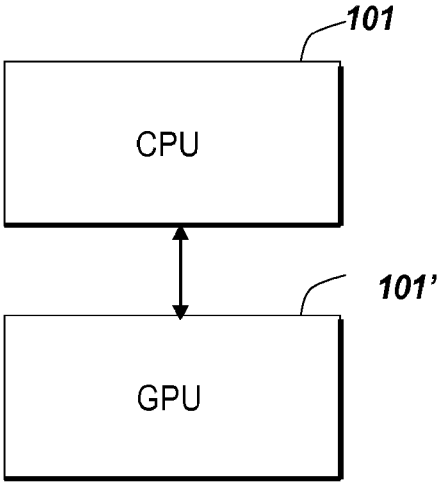

FIG. 1H

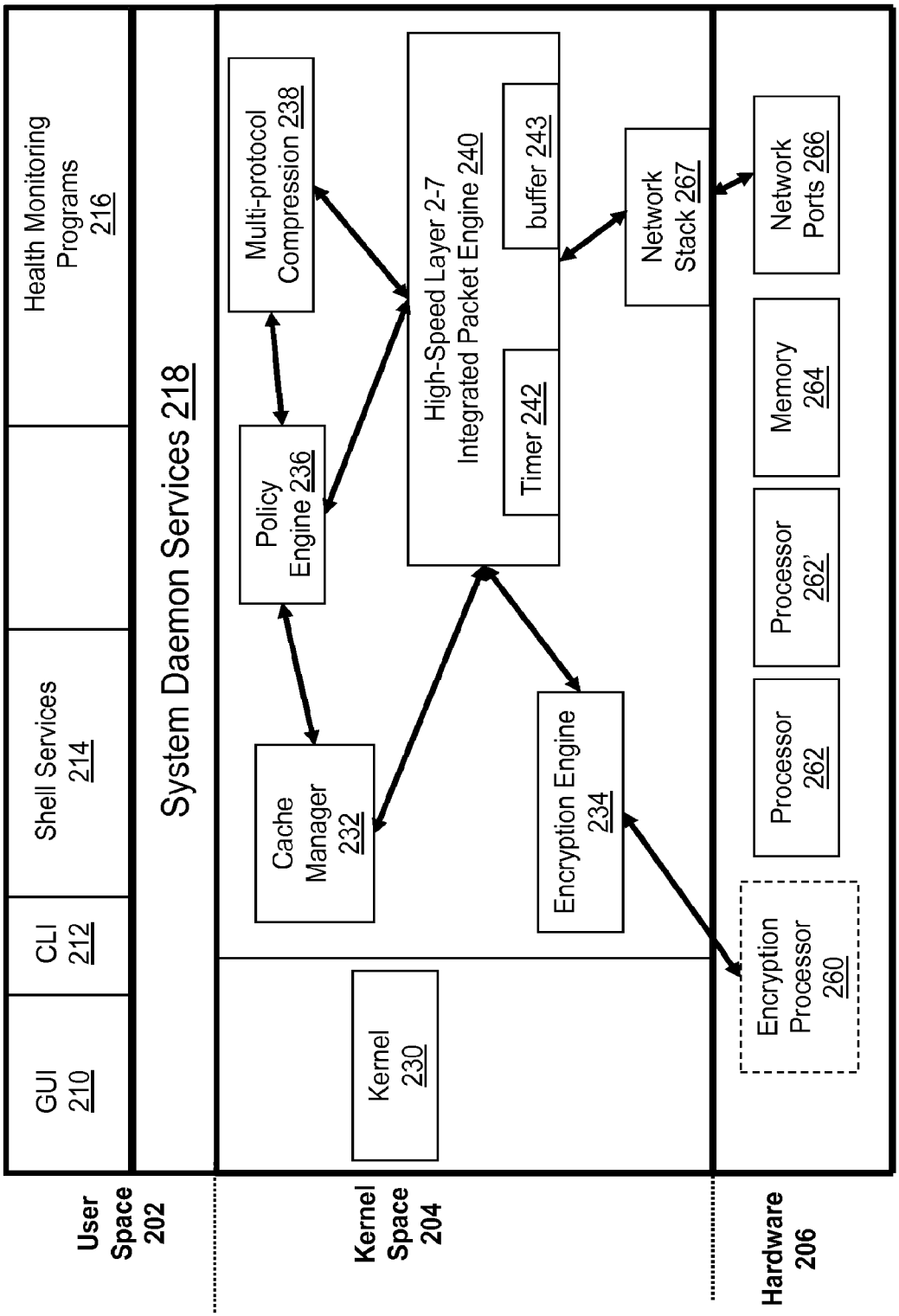
User Space 202
Kernel Space 204
Hardware 206
GUI 210
CLI 212
Shell Services 214
Health Monitoring Programs 216
System Daemon Services 218
Kernel 230
Cache Manager 232
Encryption Engine 234
Policy Engine 236
Multi-protocol Compression 238
High-Speed Layer 2-7 Integrated Packet Engine 240
Timer 242
buffer 243
Network Stack 267
Encryption Processor 260
Processor 262
Processor 262'
Memory 264
Network Ports 266
200

FIG. 2A

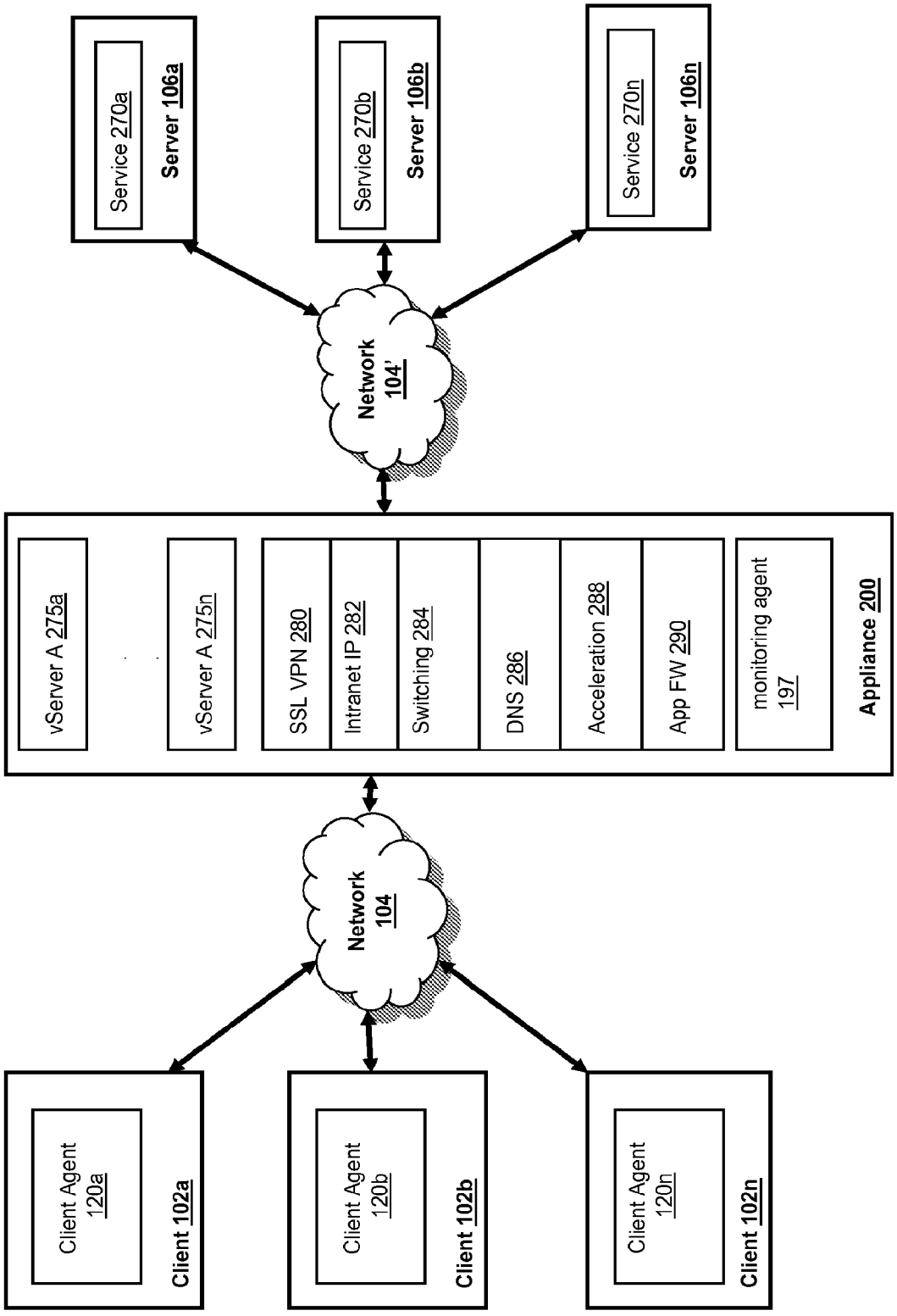
Client Agent 120a
Client 102a
Client Agent 120b
Client 102b
Client Agent 120n
Client 102n
Network 104
vServer A 275a
vServer A 275n
SSL VPN 280
Intranet IP 282
Switching 284
DNS 286
Acceleration 288
App FW 290
monitoring agent 197
Appliance 200
Network 104'
Service 270a
Server 106a
Service 270b
Server 106b
Service 270n
Server 106n

FIG. 2B

10/18

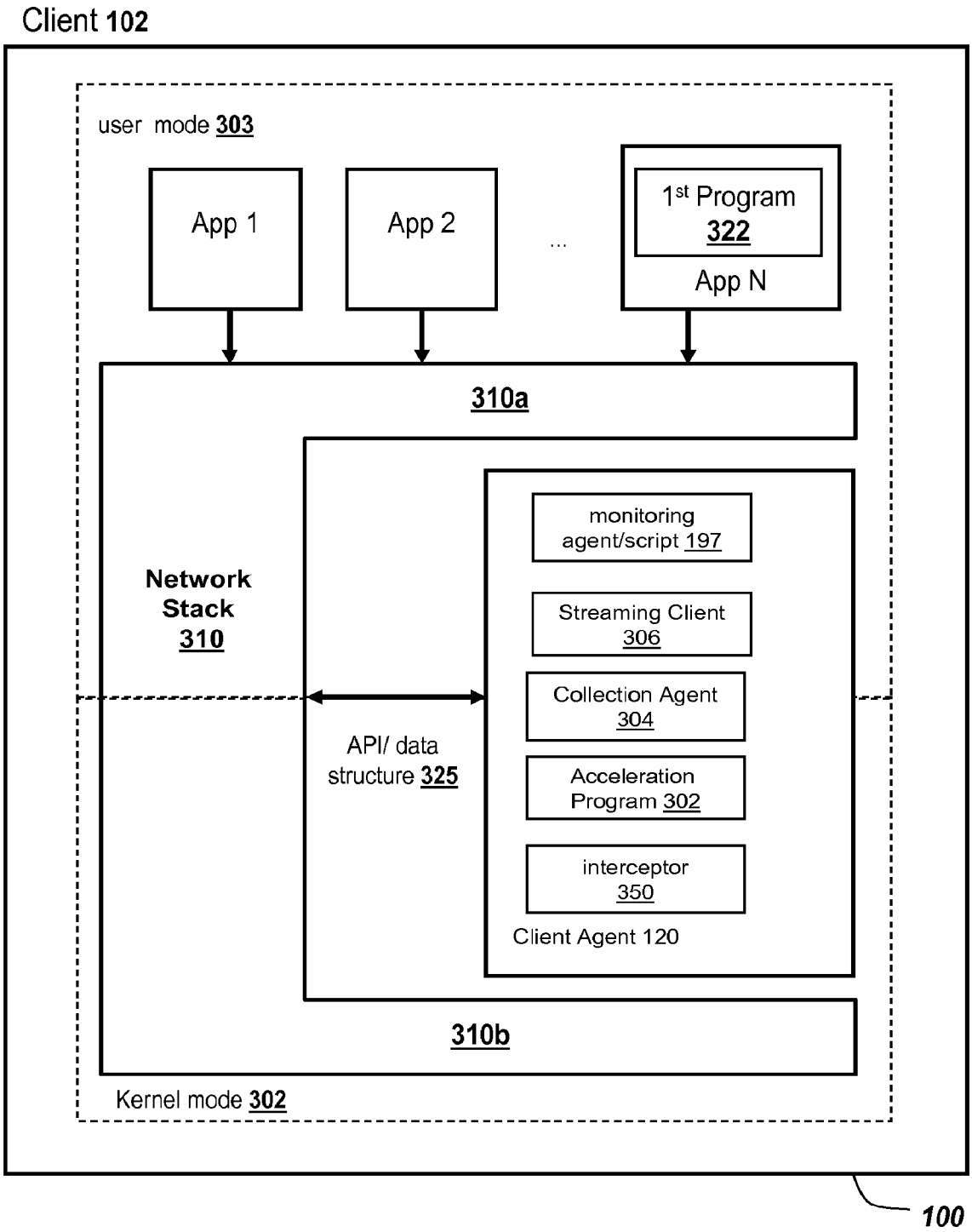
Client 102
user mode 303
App 1
App 2
...
1st Program 322
App N
310a
Network Stack 310
monitoring agent/script 197
Streaming Client 306
Collection Agent 304
API/ data structure 325
Acceleration Program 302
interceptor 350
Client Agent 120
310b
Kernel mode 302
100

FIG. 3

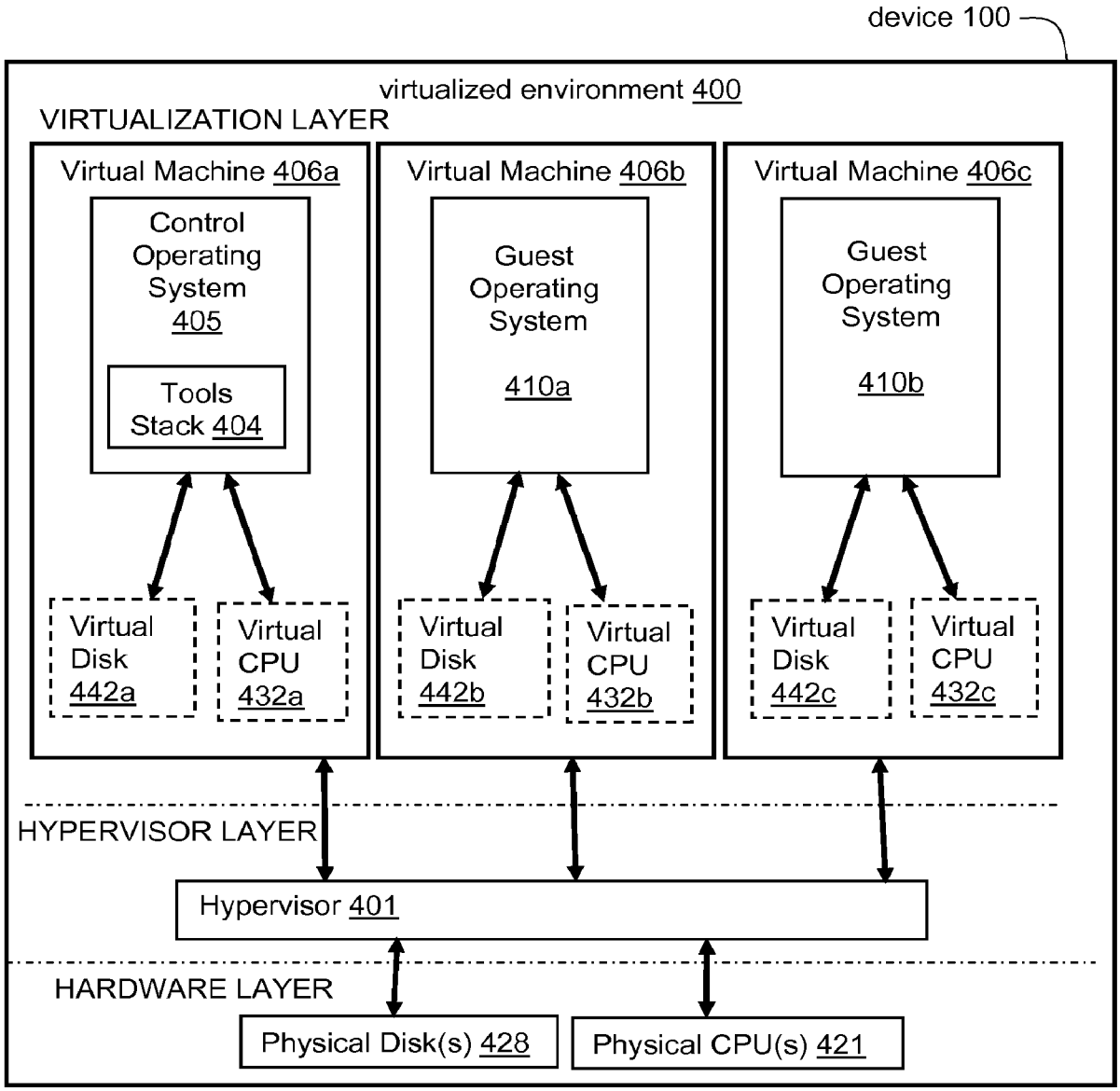
device 100
virtualized environment 400
VIRTUALIZATION LAYER
Virtual Machine 406a
Control Operating System 405
Tools Stack 404
Virtual Disk 442a
Virtual CPU 432a
Virtual Machine 406b
Guest Operating System 410a
Virtual Disk 442b
Virtual CPU 432b
Virtual Machine 406c
Guest Operating System 410b
Virtual Disk 442c
Virtual CPU 432c
HYPERVISOR LAYER
Hypervisor 401
HARDWARE LAYER
Physical Disk(s) 428
Physical CPU(s) 421

FIG. 4A

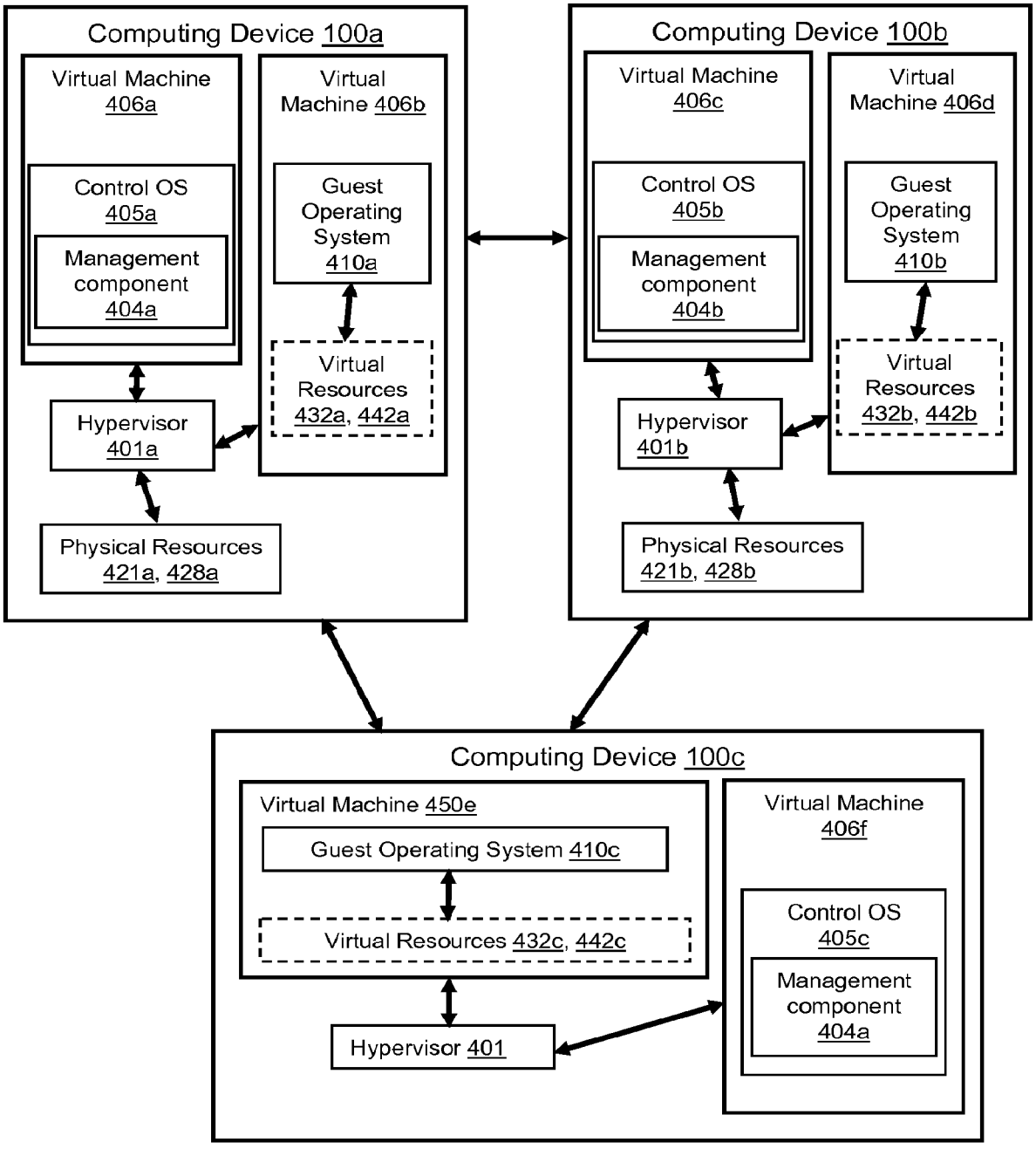
Computing Device 100a
Virtual Machine 406a
Control OS 405a
Management component 404a
Virtual Machine 406b
Guest Operating System 410a
Virtual Resources 432a, 442a
Hypervisor 401a
Physical Resources 421a, 428a
Computing Device 100b
Virtual Machine 406c
Control OS 405b
Management component 404b
Virtual Machine 406d
Guest Operating System 410b
Virtual Resources 432b, 442b
Hypervisor 401b
Physical Resources 421b, 428b
Computing Device 100c
Virtual Machine 450e
Guest Operating System 410c
Virtual Resources 432c, 442c
Hypervisor 401
Virtual Machine 406f
Control OS 405c
Management component 404a

FIG. 4B

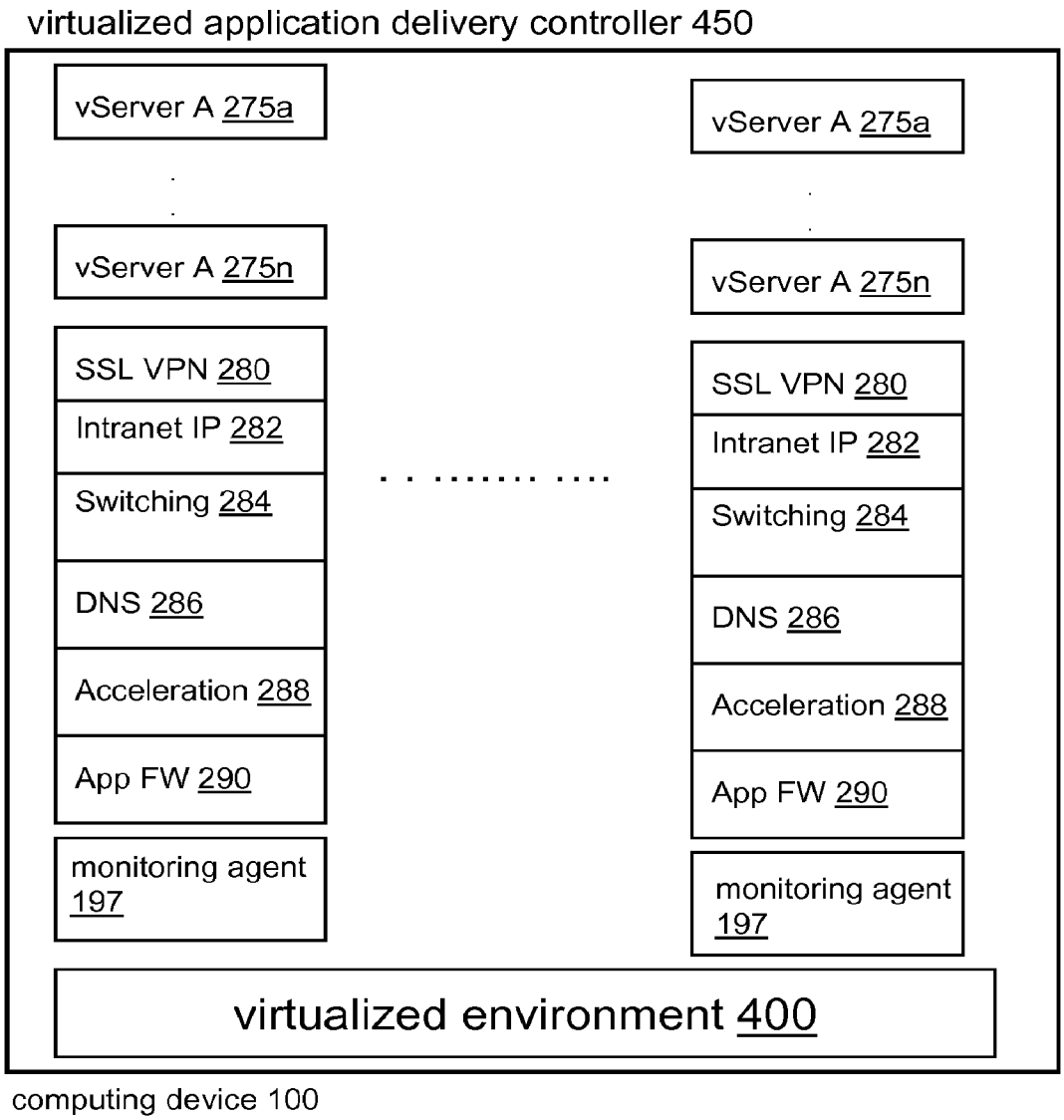
virtualized application delivery controller 450
vServer A 275a
vServer A 275n
SSL VPN 280
Intranet IP 282
Switching 284
DNS 286
Acceleration 288
App FW 290
monitoring agent 197
vServer A 275a
vServer A 275n
SSL VPN 280
Intranet IP 282
Switching 284
DNS 286
Acceleration 288
App FW 290
monitoring agent 197
virtualized environment 400
computing device 100

FIG. 4C

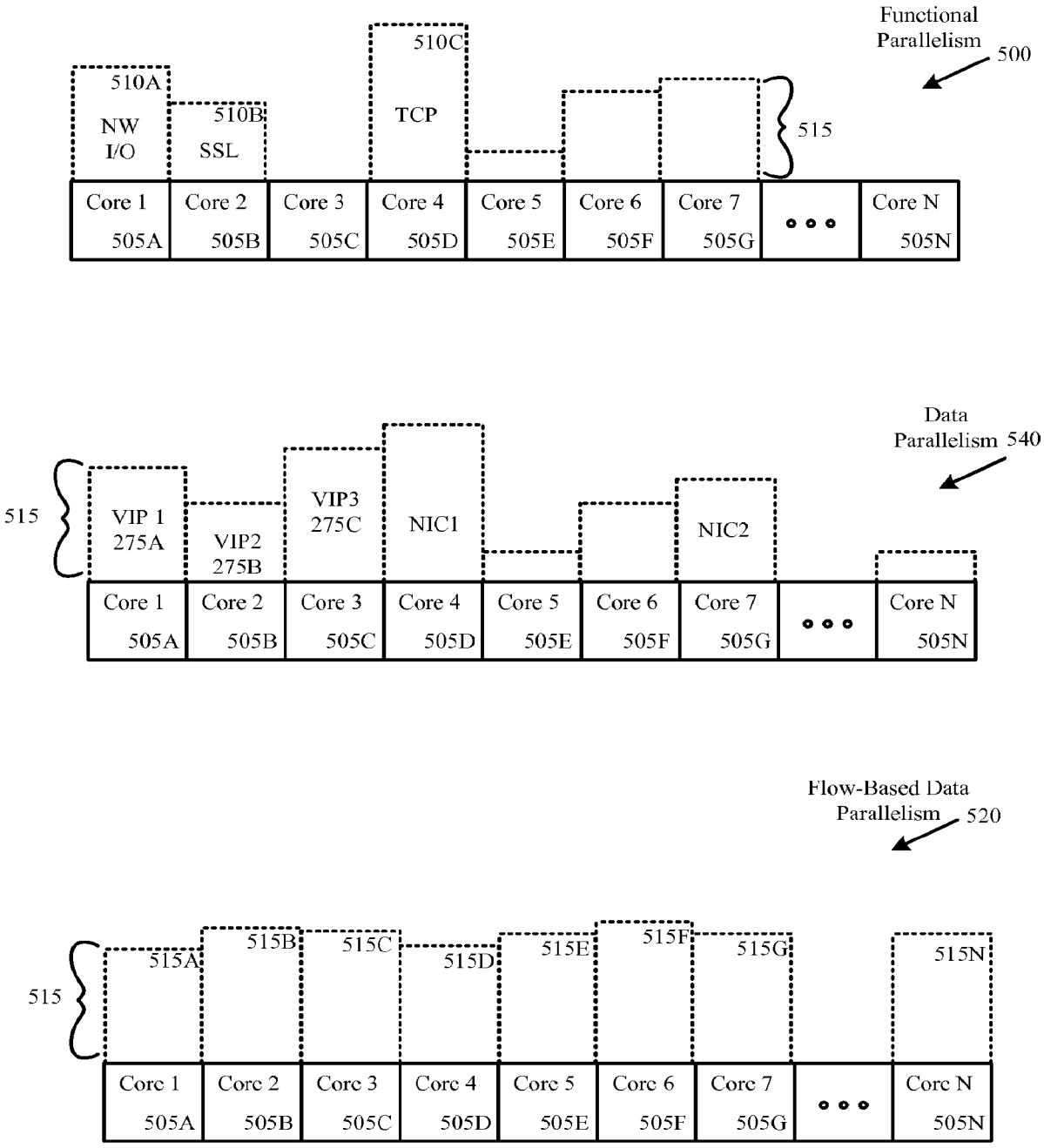
Functional Parallelism
500
510A
NW I/O
510B
SSL
510C
TCP
515
Core 1 505A
Core 2 505B
Core 3 505C
Core 4 505D
Core 5 505E
Core 6 505F
Core 7 505G
Core N 505N
Data Parallelism 540
515
VIP 1 275A
VIP2 275B
VIP3 275C
NIC1
NIC2
Flow-Based Data Parallelism 520
515
515A
515B
515C
515D
515E
515F
515G
515N

FIG. 5A

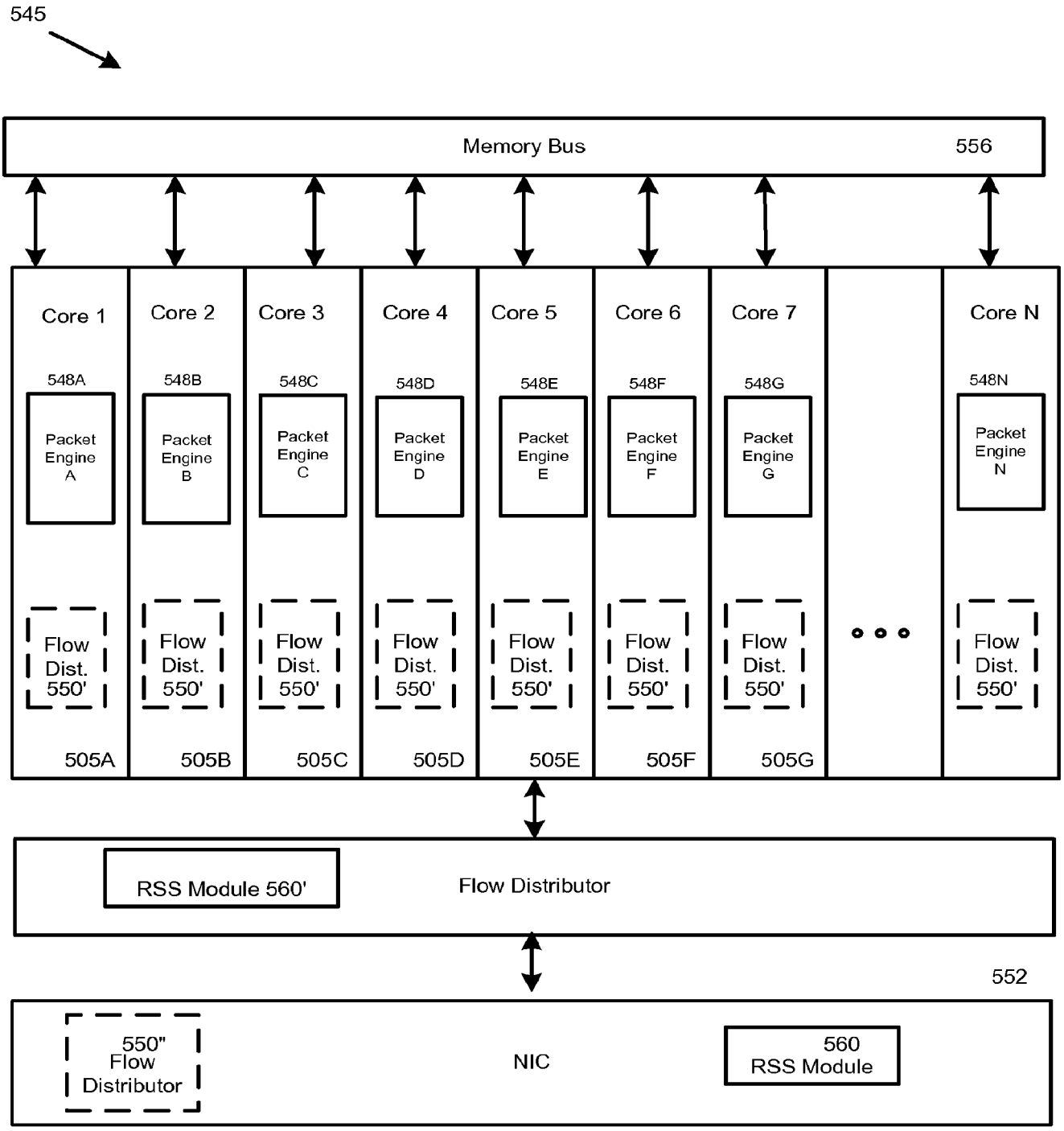
545
Memory Bus
556
Core 1
Core 2
Core 3
Core 4
Core 5
Core 6
Core 7
Core N
548A
548B
548C
548D
548E
548F
548G
548N
Packet Engine A
Packet Engine B
Packet Engine C
Packet Engine D
Packet Engine E
Packet Engine F
Packet Engine G
Packet Engine N
Flow Dist. 550'
505A
505B
505C
505D
505E
505F
505G
RSS Module 560'
Flow Distributor
552
550" Flow Distributor
NIC
560 RSS Module

FIG. 5B

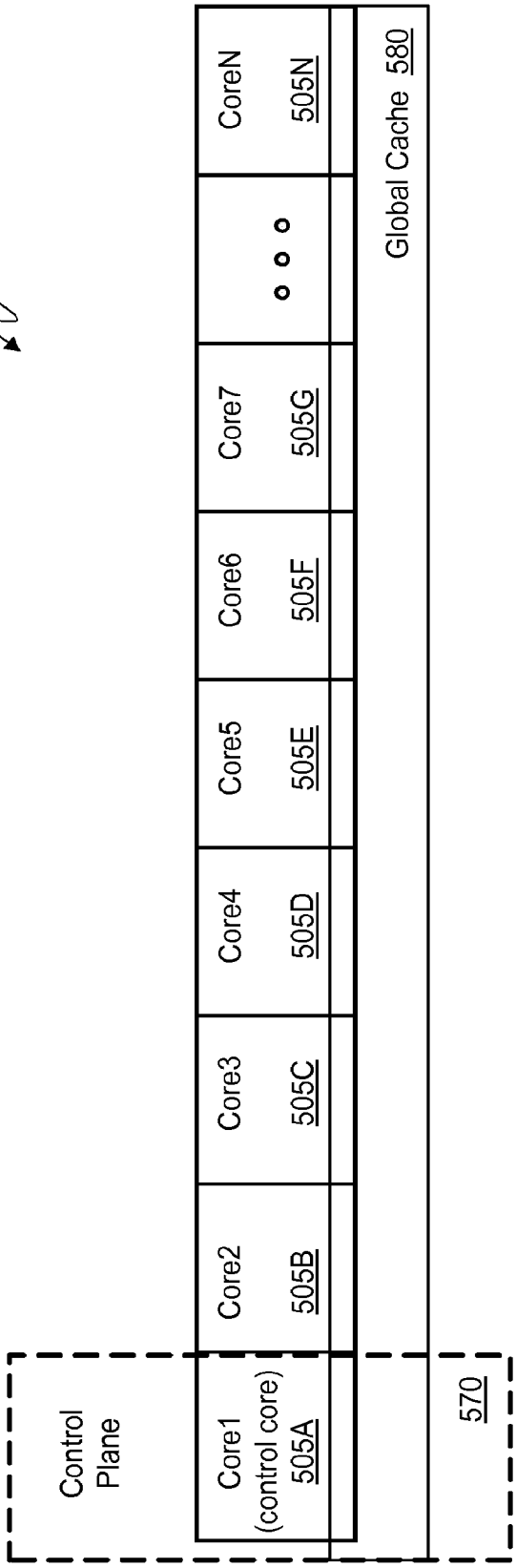
575
Control Plane
Core1 (control core) 505A
Core2 505B
Core3 505C
Core4 505D
Core5 505E
Core6 505F
Core7 505G
CoreN 505N
Global Cache 580
570

FIG. 5C

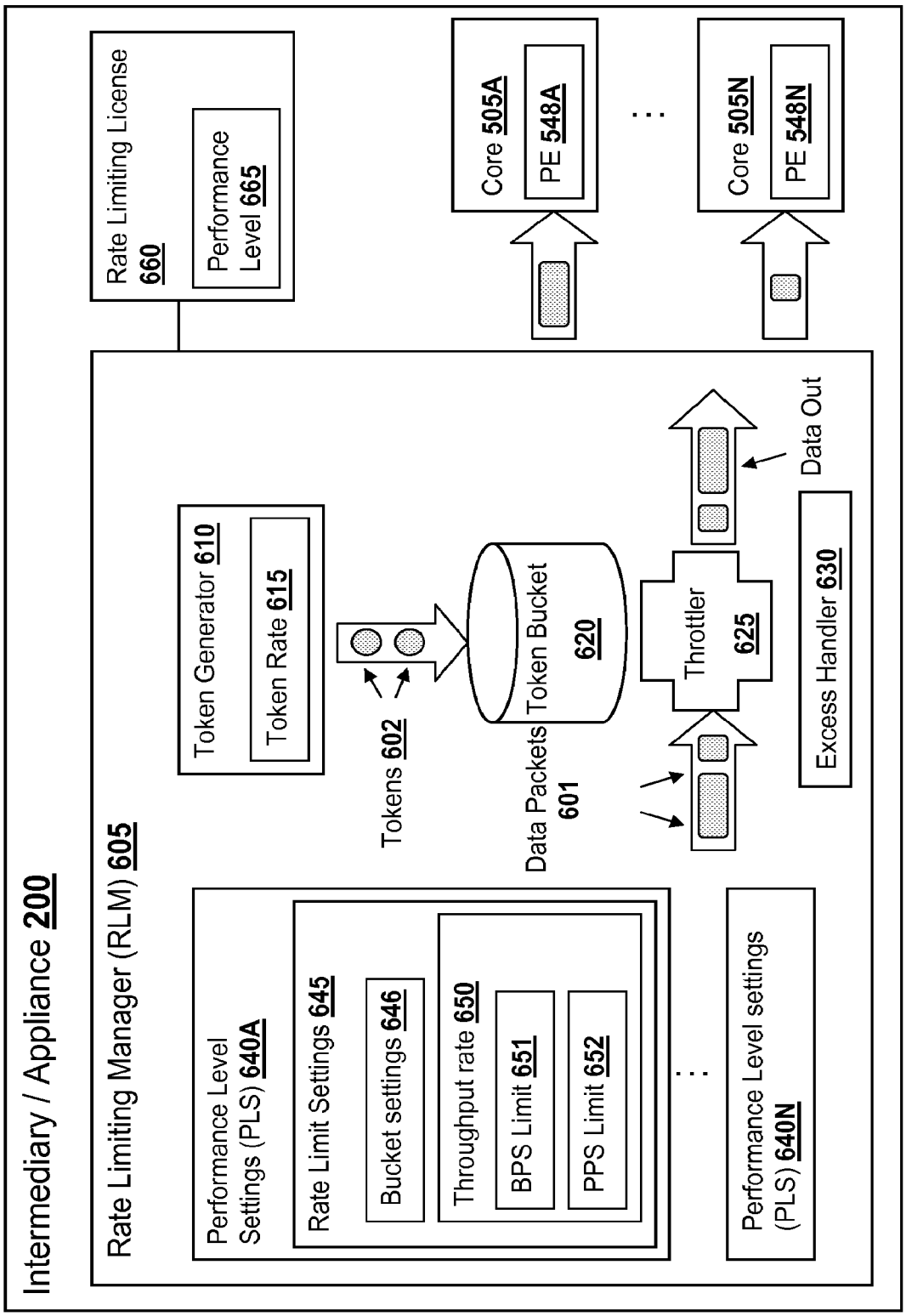
Intermediary / Appliance 200
Rate Limiting Manager (RLM) 605
Rate Limiting License 660
Performance Level 665
Core 505A
PE 548A
Core 505N
PE 548N
Token Generator 610
Token Rate 615
Tokens 602
Token Bucket 620
Data Packets 601
Throttler 625
Data Out
Excess Handler 630
Performance Level Settings (PLS) 640A
Rate Limit Settings 645
Bucket settings 646
Throughput rate 650
BPS Limit 651
PPS Limit 652
Performance Level settings (PLS) 640N

FIG. 6A

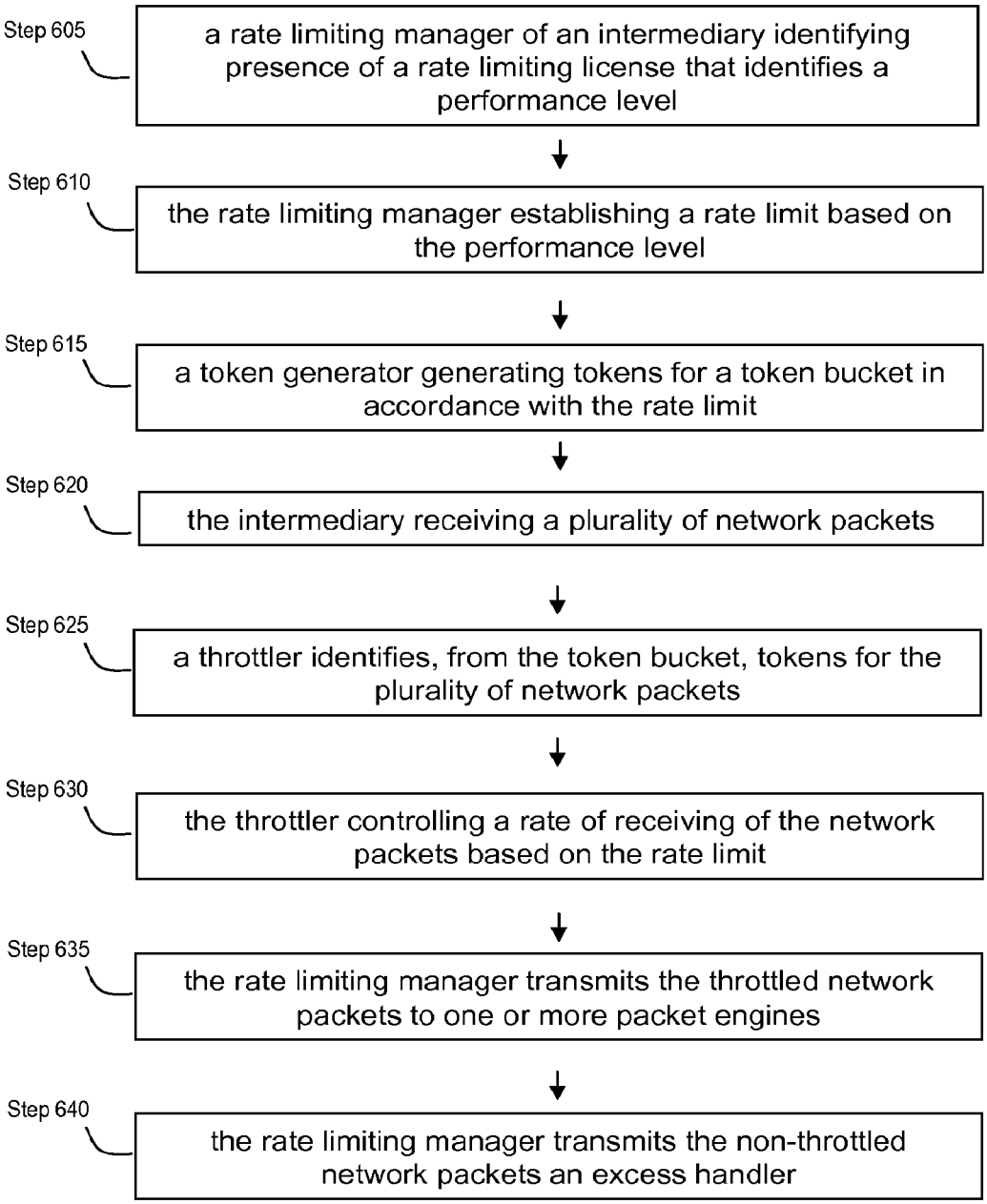
Step 605
a rate limiting manager of an intermediary identifying presence of a rate limiting license that identifies a performance level
Step 610
the rate limiting manager establishing a rate limit based on the performance level
Step 615
a token generator generating tokens for a token bucket in accordance with the rate limit
Step 620
the intermediary receiving a plurality of network packets
Step 625
a throttler identifies, from the token bucket, tokens for the plurality of network packets
Step 630
the throttler controlling a rate of receiving of the network packets based on the rate limit
Step 635
the rate limiting manager transmits the throttled network packets to one or more packet engines
Step 640
the rate limiting manager transmits the non-throttled network packets an excess handler

FIG. 6B

INTERNATIONAL SEARCH REPORT

International application No
PCT/US2010/039213

A. CLASSIFICATION OF SUBJECT MATTER
INV . H04L12/56
ADD .

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
H04L

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal , COMPENDEX, INSPEC, WPI Data

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No
X	US 2002/138643 Al (SHIN KANG G [US] ET AL) 26 September 2002 (2002-09-26) paragraph [0001] - paragraph [0094]; figures 1-6	1-22
X,P	WO 2010/068436 Al (CITRIX SYSTEMS INC [US]; KAMATH SANDEEP [US]; KHEMANI PRAKASH [US]) 17 June 2010 (2010-06-17) page 1 - page 3, line 16 page 10, line 24 - page 13, line 9 page 31, line 32 - page 33, line 27 page 41, line 1 - page 56, line 6 ; figures 4A,4B	1-22

☐ Further documents are listed in the continuation of Box C

☒ See patent family annex

* Special categories of cited documents

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance, the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance, the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Dale of the actual completion of the international search	Date of mailing of the international search report
15 September 2010	22/09/2010

Name and mailing address of the ISA/	Authorized officer
European Patent Office, P B 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel (+31-70) 340-2040, Fax (+31-70) 340-3016	**Garcia** Bolós, **Ruth**

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No
PCT/US2010/039213

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2002138643 A1	26-09-2002	NONE	
WO 2010068436 A1	17-06-2010	US 2010131668 A1	27-05-2010