



- (51) **International Patent Classification:**
G06F 9/50 (2006.01) *H04L 29/08* (2006.01)
- (21) **International Application Number:**
PCT/US2017/014056
- (22) **International Filing Date:**
19 January 2017 (19.01.2017)
- (25) **Filing Language:** English
- (26) **Publication Language:** English
- (30) **Priority Data:**
15/015,507 4 February 2016 (04.02.2016) US
- (71) **Applicant:** CITRIX SYSTEMS, INC. [US/US]; 851 West Cypress Creek Road, Fort Lauderdale, Florida 33309 (US).
- (72) **Inventor:** PARAMASIVAM, Sankar Muthu; c/o Citrix Systems, Inc., 4988 Great America Parkway, Santa Clara, California 95054 (US).
- (74) **Agents:** MCKENNA, Christopher J. et al.; FOLEY & LARDNER LLP, 3000 K Street N.W., Suite 600, Washington, District of Columbia 20007-5109 (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, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DJ, DK, DM,

DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JP, KE, KG, KH, KN, KP, KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, 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, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, 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, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Declarations under Rule 4.17:

- as to applicant's entitlement to apply for and be granted a patent (Rule 4.17(ii))
- as to the applicant's entitlement to claim the priority of the earlier application (Rule 4.17(iii))

Published:

- with international search report (Art. 21(3))

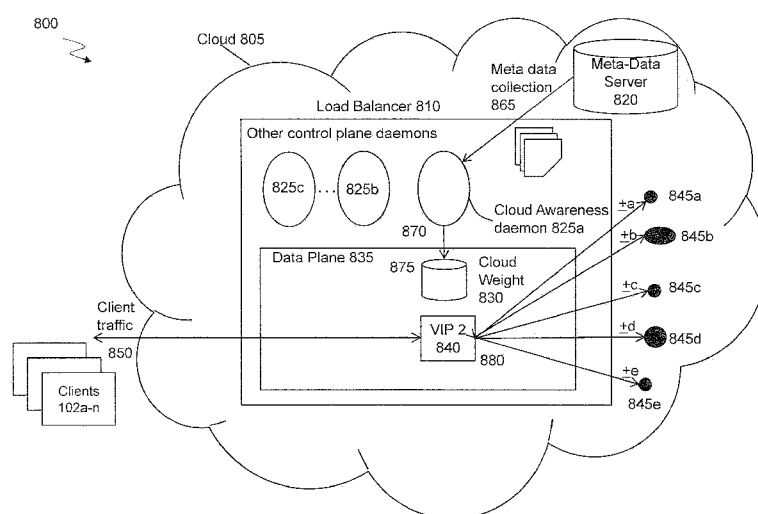
(54) Title: SYSTEM AND METHOD FOR CLOUD AWARE APPLICATION DELIVERY CONTROLLER

FIG. 8

(57) **Abstract:** The present disclosure is directed towards systems and methods of managing cloud- based services via a network. A controller intermediary to clients and servers of a cloud environment can receive account information related to each of a plurality of instances of one or more services provided by the servers of the cloud environment. The controller can determine a weight for each of the instances based on the account information. The weight can indicate a performance of the instance of the service provided via the cloud environment. The controller can select an instance of the plurality of instances to direct network traffic from a client. The controller can select the instance based on the weight for each of the instances.

SYSTEM AND METHOD FOR CLOUD AWARE APPLICATION DELIVERY CONTROLLER

RELATED APPLICATION

5 The present application claims the benefit of and priority to U.S. Non-provisional Application No. 15/015,507, titled “System and Method for Cloud Aware Application Delivery Controller” and filed on February 4, 2016, which is incorporated by reference herein in its entirety for all purposes.

FIELD OF THE DISCLOSURE

10 The present application generally relates to a cloud aware application delivery controller. In particular, the present application relates to systems and methods for managing delivery of cloud-based services using metadata related to instances of the services.

BACKGROUND OF THE DISCLOSURE

15 A client device can access or use an application, service, or infrastructure provided via a cloud environment including one or more servers. The client device can access the cloud environment or one or more servers via an intermediary device that is intermediary to the client device and the servers. However, instances of services may be distributed across
20 multiple regions or availability zones in a cloud environment. It may be challenging to efficiently provide services for instances that are distributed across the cloud environment because the distribution of instances can impact overhead, cost, latency, throughput, or load.

BRIEF SUMMARY OF THE DISCLOSURE

25 The present disclosure is directed towards systems and methods for managing cloud-based services via a network. In cloud environments, instances of services may be distributed across multiple regions, availability zones, and more. A cloud environment can refer to an infrastructure as a service (IaaS) environment, software as a service (SaaS) environment, or platform as a service (PaaS) environment. An IaaS environment can refer to cloud
30 computing where shared computing resources, data and information are provided to computing devices on-demand. A device intermediary to the servers of the cloud

environment and client devices, such as an application delivery controller (ADC), can facilitate providing or delivering the services provided by the cloud environment. A controller of the present disclosure can query a cloud application programming interface (API), such as an IaaS API, to discover or determine metadata corresponding to instances of the services provided by the IaaS cloud. The metadata can include account information associated with a provider of the cloud environment, such as a distribution, available resources, or resource consumption. The controller can use this metadata to automatically tune features or functionalities of an application delivery controller, thus bringing cloud awareness into the ADC systems. Using metadata associated with a cloud environment allows the controller to more efficiently provide services for instances that are distributed across the cloud environment by reducing overhead, cost, latency, throughput, or load.

For example, hosted systems in public or private cloud environments can include one or more APIs to identify or determine cloud partitions or boundaries. The APIs allows the hosted system or systems that interface with the hosted system to monitor internal dynamism to identify the environmental impact on overhead, cost, latency, throughput, or load. Once the system is aware of the infrastructure and its dynamism by using the available metadata, the system can automatically tune itself such that the system is more suitable to the cloud environment, and can perform more efficiently. Thus, systems and methods of the present disclosure can: 1) identify cloud partitions/boundaries; 2) monitor the internal dynamism in the cloud; 3) determine the cloud/environmental impact on the system; and 4) tune the hosted system to adapt to the environmental variables.

Providers of private or public cloud infrastructure (e.g., a cloud environment) can maintain an informational store or data repository that includes account information for customers of the cloud infrastructure. The system can use this information store as a database to track of user account information and their resource allocations for billing, maintenance, analysis, or other purposes. Cloud APIs can be configured to query the database. Using these APIs the system can fetch metadata about instances, including fetching the instances' placement in the cloud, resource allocation (CPU, memory, bandwidth), type of reservation, type of connectivity, or other attributes or characteristics associated with the instance.

An ADC can use the retrieved metadata to perform load balancing operations. For example, the ADC can use the metadata to drive or determine a cloud weight and use the cloud weight in load balancing algorithms to improve the efficiency of the cloud environment by increase throughput or reducing latency. Further, using the metadata to tune the cloud

environment can reduce load on the system and cloud infrastructure, reduce network overhead, reduce cost and provide improved load balancing service.

To do so, the system can be configured with a control plane daemon that periodically fetches some or all instances of services provided via the cloud environment, and meta-data
5 or account information related to the instances. The system can analyze the meta-data and identify, generate, or determine cloud weights for system specific configured entities. The system can associate the identified cloud weights with every instance and propagate the weights to a data plane. The weights can indicate or refer to a logical measurement of cloud environmental impact to the hosted systems. These weights can be used with various load-
10 balancing algorithms (as weighted LB algorithm) to achieve improved system performance.

For example, a high availability (HA) cloud IaaS environment can include a pair of high availability load-balancers where one of the load-balancers serves as the primary and is in an active state and the other load balancer serves as a secondary and is in a passive state. However, one of the load-balancers of the pair may have better connectivity to the load-
15 balanced services. By analyzing the retrieved metadata, the system can discover the relative locations of both servers and the load-balancers and automatically select which load-balancer would be optimal for traffic, and make that the primary load-balancer. The result is a better performing service.

For example, a load balancer can direct client traffic among eight backend servers (S1
20 to S8). These servers and load balancer instances may be hosted in different cloud partitions (such as data centers, zones, etc.) for various reasons. The active (primary) ADC may attract client traffic. The primary ADC can select any one server (out of eight servers) and forward the client request. However, approximately 75% (6 out of 8) of the traffic may be forwarded across different cloud partitions for both client requests and server responses, which would
25 negatively impact in latency, throughput, overhead, etc. By using retrieved account information or meta-data, the system can drive the cloud weight associated with the backend servers (or services) with respect to the primary and secondary instances.

The system can determine the cloud weights for the load balancers by calculating a sum of cloud weights associated with services with respect to both primary and secondary
30 load balancer instances. This total cloud weight can represent or indicate a cloud cost for load balancing traffic between load balancers and servers. For example, the cloud weight for secondary load balancer may be greater than the cloud weight for the primary load balancer

because a majority of the services (S1 to S6) are closer to the passive or secondary load balancer instance. Thus, processing (load balancing) client traffic at the secondary load balancer instance may be better for throughput, latency, overhead, or cost than processing the same client traffic at the primary load balancer instance.

5 But since the current primary is attracting client traffic, which is inefficient based on the analysis performed using the cloud information, the system can trigger HA failover of the primary load balancer instance so that the secondary load balancer instance becomes the new primary/active load balancer instance and starts processing (load balancing) client network traffic.

10 In some cases, an ADC may execute multiple virtual servers that each provide a load balancer instance bound to one or more servers. In such cases, the system can calculate could weights across all virtual servers. The system can sum the cloud weight across all virtual servers executed by an ADC to determine a total cloud weight for the ADC.

In some embodiments, the system can improve performance by caching server content
15 locally. For example, the ADC can use cloud information or meta-data to dynamically cache server content locally based on various cloud parameters, such as automatically enabling caching for contents served from specific cloud partitions, or applying apply thresholds based on cloud parameters.

In some embodiments, the system can use cloud information or meta-data to perform
20 rate limiting or throttling techniques. For example, the system can improve Access Control Lists (ACLs) and policies to operate on cloud parameters to provide control over flows and improve bandwidth utilization. The system can use include cloud parameters as part of logging and use the cloud parameters in to filter data to generated reports with helpful analytical information.

25 While the techniques disclosed herein can be used in an ADC environment, the systems and methods of the present disclosure can improve the operation of networking devices such as firewalls, routers, optimization systems, rate-limiting systems, or access control systems.

At least one aspect is directed to a method of managing cloud-based services via a
30 network. The method can be performed by a controller intermediary to a client and a cloud environment. In some embodiments, the method can include the controller receiving account information related to each of a plurality of instances of one or more services provided by one

or more servers of the cloud environment. The method can include the controller determining a weight for each of the instances based on the account information. The weight can indicate a performance of the instance of the service provided via the cloud environment. The method can include the controller selecting an instance of the plurality of instances to
5 direct network traffic from a client. The controller can select the instance based on a load balancing function and the weight for each of the instances.

The controller can use a cloud application programming interface to generate one or more queries to retrieve the account information for each of the plurality of instances. The one or more queries can be configured to retrieve information for at least one of a placement,
10 resource allocation, type of reservation, or type of connectivity.

The controller can use the cloud application programming interface to generate a query to retrieve placement information for a first instance of the plurality of instances. The controller can transmit the query to a metadata server maintained by a provider of the cloud environment. The metadata server can include a database storing placement information for
15 the first instance. The controller can receive the placement information for the first instance from the metadata server responsive to the query.

The controller can determine a first weight for the first instance provided via a first zone of the cloud environment. The controller can determine the first weight based on the account information. The controller can determine a second weight for a second instance
20 provided via a second zone of the cloud environment different from the first zone. The controller can determine the second weight based on the account information. The controller can select the first instance responsive to the first weight greater than the second weight.

The method can include the controller identifying that the first instance of a first service of the one or more services is provided via a first zone of the cloud environment. The controller can identify that the first instance is provided via the first zone based on the
25 account information. The controller can identify, based on the account information, that a second instance of a second service of the one or more services is provided via a second zone of the cloud environment. The controller can determine that a distance between the controller and the second zone can be greater than a distance between the controller and the first zone.
30 The controller can determine to cache data provided via the second instance of the second service in memory of the controller. The controller can determine to cache the data

responsive to determining the distance between the controller and the second zone is greater than the distance between the controller and the first zone.

The method can include the controller identifying a plurality of devices intermediary to the client and the one or more servers. The controller can determine a first weight for the first instance based on the account information and a first placement of a first intermediary device of the plurality of devices. The controller can determine a second weight for the first instance based on the account information and a second placement of a second intermediary device of the plurality of devices. The controller can select the first intermediary device to direct network traffic of the instance from the client based on a comparison of the first weight and the second weight. In some embodiments, the controller determines that the first weight is greater than the second weight based on a distance between the first placement and the one or more servers being less than a distance between the second placement and the one or more servers. The controller can select the first intermediary device responsive to the first weight being greater than the second weight.

The method includes the controller identifying a first virtual server executing on a device intermediary to the client and the one or more servers. The first virtual server can be associated with the instance. The controller can identify a second virtual server executing on the device intermediary to the client and the one or more servers. The second virtual server can be associated with a second instance of the one or more services. The controller can determine a first weight for the instance based on the account information and an attribute of the first virtual server. The controller can determine a second weight for the second instance based on the account information and an attribute of the second virtual server. The controller can select the first virtual server to direct network traffic of the instance from the client based on a comparison of the first weight and the second weight.

The method can include the controller identifying a first device intermediary to the client and the cloud environment. The first device can be configured to direct network traffic related to the plurality of instances. The controller can identify a second device intermediary to the client and the cloud environment. The second device can be configured to direct network traffic related to the plurality of instances. The controller can determine a first weight based on account information for each of the plurality of instances and an attribute of the first device. The controller can determine a second weight based on account information for each of the plurality of instances and an attribute of the second device. The second

weight can be less than the first weight. The controller can select the first device to direct network traffic related to the plurality of instances.

The controller can identify a first plurality of virtual servers executed by a first device intermediary to the client and the cloud environment. The first plurality of virtual servers can be configured to direct network traffic related to one or more of the plurality of instances.

The controller can identify a second plurality of virtual servers executed by a second device intermediary to the client and the cloud environment. The second plurality of virtual servers can be configured to direct network traffic related to one or more of the plurality of instances.

The controller can determine a first weight based on account information for each of the one or more plurality of instances and an attribute for each of the first plurality of virtual servers.

The controller can determine a second weight based on account information for each of the one or more plurality of instances and an attribute for each of the second plurality of virtual servers. The second weight can be less than the first weight. The controller can select the first device executing the first plurality of virtual servers to direct network traffic related to the one or more of the plurality of instances.

The method can include the controller periodically receiving updated account information. The controller can periodically select, based on the load balancing function and an updated weight for each of the instances based on the updated account information, a second instance of the plurality of instances to direct network traffic from one or more clients.

Another aspect is directed to a system to manage cloud-based services via a network. The system can include a controller intermediary to a client and a cloud environment. The controller can be configured to receive account information related to each of a plurality of instances of one or more services provided by one or more servers of the cloud environment.

The controller can be configured to determine a weight for each of the instances based on the account information, the weight indicating a performance of the instance of the service provided via the cloud environment. The controller can be configured to select, based on a load balancing function and the weight for each of the instances, an instance of the plurality of instances to direct network traffic from a client.

The controller can be configured to use a cloud application programming interface to generate one or more queries to retrieve the account information for each of the plurality of

instances. The one or more queries can be configured to retrieve information for at least one of a placement, resource allocation, type of reservation, or type of connectivity.

The controller can be further configured to use a cloud application programming interface to generate a query to retrieve placement information for a first instance of the plurality of instances. The controller can transmit the query to a metadata server maintained by a provider of the cloud environment. The metadata server can include a database storing placement information for the first instance. The controller can receive, from the metadata server responsive to the query, the placement information for the first instance.

The controller can be further configured to determine, based on the account information, a first weight for the first instance provided via a first zone of the cloud environment. The controller can determine, based on the account information, a second weight for a second instance provided via a second zone of the cloud environment different from the first zone. The controller can select the first instance responsive to the first weight greater than the second weight.

The controller can be further configured to identify, based on the account information, that the first instance of a first service of the one or more services is provided via a first zone of the cloud environment. The controller can identify, based on the account information, that a second instance of a second service of the one or more services is provided via a second zone of the cloud environment. A distance between the controller and the second zone can be greater than a distance between the controller and the first zone. The controller can determine, responsive to determining the distance between the controller and the second zone is greater than the distance between the controller and the first zone, to cache data provided via the second instance of the second service in memory of the controller.

The controller can be further configured to identify a plurality of devices intermediary to the client and the one or more servers. The controller can determine a first weight for the first instance based on the account information and a first placement of a first intermediary device of the plurality of devices. The controller can determine a second weight for the first instance based on the account information and a second placement of a second intermediary device of the plurality of devices. The controller can select the first intermediary device to direct network traffic of the instance from the client based on a comparison of the first weight and the second weight.

The controller can be further configured to determine that the first weight is greater than the second weight based on a distance between the first placement and the one or more servers being less than a distance between the second placement and the one or more servers. The controller can select the first intermediary device responsive to the first weight being greater than the second weight.

The controller can be further configured to identify a first virtual server executing on a device intermediary to the client and the one or more servers. The first virtual server can be associated with the instance. The controller can identify a second virtual server executing on the device intermediary to the client and the one or more servers. The second virtual server can be associated with a second instance of the one or more services. The controller can determine a first weight for the instance based on the account information and an attribute of the first virtual server. The controller can determine a second weight for the second instance based on the account information and an attribute of the second virtual server. The controller can select the first virtual server to direct network traffic of the instance from the client based on a comparison of the first weight and the second weight.

The controller can be further configured to identify a first device intermediary to the client and the cloud environment. The first device can be configured to direct network traffic related to the plurality of instances. The controller can identify a second device intermediary to the client and the cloud environment. The second device can be configured to direct network traffic related to the plurality of instances. The controller can determine a first weight based on account information for each of the plurality of instances and an attribute of the first device. The controller can determine a second weight based on account information for each of the plurality of instances and an attribute of the second device, the second weight less than the first weight. The controller can select the first device to direct network traffic related to the plurality of instances.

BRIEF DESCRIPTION OF THE FIGURES

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

FIG. 1A is a block diagram of an embodiment of a network environment for a client to access a server via an appliance;

FIG. 1B 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;

5 FIG. 1D 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. 1E - 1H 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;

10 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;

15 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;

20 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. 6 is a block diagram of an embodiment of a cluster system;

25 FIG. 7A is a block diagram of an embodiment of a system for managing cloud-based services via a network; and

FIG. 7B is a block diagram of an embodiment of a method of managing cloud-based services via a network.

FIG. 8 is a functional block diagram of an embodiment of a system for managing cloud-based services via a network.

FIG. 9A-9C are functional block diagrams of an embodiment of a system for managing cloud-based services via a network.

The features and advantages of the present solution will become more apparent from the detailed description set forth below when taken in conjunction with the drawings, in which like reference characters identify corresponding elements throughout. In the drawings, like reference numbers generally indicate identical, functionally similar, and/or structurally similar elements.

DETAILED DESCRIPTION

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 providing a clustered appliance architecture environment; and

Section G describes embodiments of systems and methods for managing cloud-based services via a network.

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 1A, 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. 1A 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. 1A, 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
5 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 NetScaler®
10 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
15 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.
20 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.

25 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
30 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
5 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
10 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
15 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
20 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
25 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
30 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. 1B, 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 CloudBridge® 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
5 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
10 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.

15 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
20 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
25 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
30 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. 1D, 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 102 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
5 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
10 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
15 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
20 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
25 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
30 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 Workspace Suite™ by Citrix Systems, Inc., such as XenApp® or XenDesktop® 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
5 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
10 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
15 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
20 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
25 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
30 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 embodiment 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 Workspace Suite™ by Citrix Systems, Inc., such as XenApp® or XenDesktop®, 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 Systems, Inc. of Fort Lauderdale, Florida, WebEx™ provided by Cisco Systems, Inc. of San Jose, California, or Microsoft Office Live Meeting provided by Microsoft Corporation of Redmond, Washington.

Still referring to FIG. 1D, 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
5 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
10 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,
15 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
20 198. 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
25 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
30 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
5 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
10 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
15 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
20 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
30 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
5 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,
10 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
15 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
20 capable of communicating on any type and form of network and performing the operations described herein. FIGs. 1E and 1F 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. 1E and 1F, each computing device 100 includes a central processing unit 101, and a main memory unit 122. As shown in FIG. 1E, a computing device 100 may include a visual
25 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
30 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.

5 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
10 Output DRAM (EDO DRAM), Burst Extended Data Output DRAM (BEDO DRAM), Enhanced DRAM (EDRAM), synchronous DRAM (SDRAM), JEDEC SRAM, PC100 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,
15 or any other available memory chips capable of operating as described herein. In the embodiment shown in FIG. 1E, the processor 101 communicates with main memory 122 via a system bus 150 (described in more detail below). FIG. 1F 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. 1F the main memory 122 may be DRDRAM.

20 FIG. 1F 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
25 shown in FIG. 1F, 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
30 may use an Advanced Graphics Port (AGP) to communicate with the display 124. FIG. 1F 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. 1F 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
5 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
10 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.

15 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.*, 802.11, T1, 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
20 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 130a-130n may be present in the computing device 100.
25 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. 1E. 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
30 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 130a-130n 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
5 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.

10 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-
15 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.

20 As shown in FIG. 1G, 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
25 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
30 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.

5 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.

10 In some embodiments, the computing device 100 may comprise a graphics processing unit. In one of these embodiments depicted in FIG. 1H, 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
15 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
20 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
25 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
30 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. 1E and 1F. 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. 1E and 1F. 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
5 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
10 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
15 200.

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
20 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
25 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
30 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 selective 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 10 ms. 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.

5 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,
10 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
15 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
20 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
25 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
30 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
5 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,
10 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
15 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
20 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
25 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
30 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
5 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
10 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
15 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
20 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
25 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
30 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 policies 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. 1D. In one embodiment, the appliance 200 receives the monitoring agent 197 from the monitoring service 198 or monitoring server 106 as depicted in FIG. 1D. 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
5 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
10 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
15 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 as an SSL VPN session. In still further embodiments, the monitoring agent 197 measures and monitors the memory, CPU and disk
20 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
25 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
30 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
5 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
10 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
15 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
20 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,
25 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
30 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 of 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
5 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
10 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
15 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
20 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
25 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,
30 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 and/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. 1D 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
5 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.

10 *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
15 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
20 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
25 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
30 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
5 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
10 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
15 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
20 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
25 406. 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
30 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
5 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
10 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
15 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
20 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
25 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
30 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
5 “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
10 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
15 resource 421, 428. The plurality of physical machines 100 may each be provided as computing devices 100, described above in connection with FIGs. 1E-1H 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
20 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.

25 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
30 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
5 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
10 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
15 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
20 200, 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,
25 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
30 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 290 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
5 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.
10 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)
15 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
20 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
25 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
30 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 5 200' 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 10 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.

15 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 20 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 25 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 30 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 VIP1 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 NIC1 542D and NIC2 542E. In other
5 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.

10 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
15 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
20 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
25 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
30 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
5 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.
10 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
15 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
20 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
25 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
30 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
5 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
10 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
15 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
20 (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 Scaler
25 (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
30 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 290; 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 in 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
5 505 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
10 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
15 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 embodiments, the cores 505 may comprise any portion of any
20 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
25 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.

30 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
5 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
10 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
15 embodiments, the flow distributor distribute, forwards, routes, controls and/or 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,
20 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
25 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
30 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
5 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
10 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
15 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
20 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
25 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
30 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.

5 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 be used to identify a core or entity, such as a packet engine or VIP, for distributing a network packet. In some embodiments, one or
10 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
15 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. Any portion of the hash result or the hash 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
20 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
25 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
30 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
5 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
10 505.

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
15 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
20 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
25 varying amounts or access to the global cache 580.

In some embodiments, the system 575 can comprise a control core 505A. While FIG.
30 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 Providing a Distributed Cluster Architecture

As discussed in the previous section, to overcome limitations on transistor spacing and CPU speed increases, many CPU manufacturers have incorporated multi-core CPUs to

improve performance beyond that capable of even a single, higher speed CPU. Similar or further performance gains may be made by operating a plurality of appliances, either single or multi-core, together as a distributed or clustered appliance. Individual computing devices or appliances may be referred to as nodes of the cluster. A centralized management system may perform load balancing, distribution, configuration, or other tasks to allow the nodes to operate in conjunction as a single computing system. Externally or to other devices, including servers and clients, in many embodiments, the cluster may be viewed as a single virtual appliance or computing device, albeit one with performance exceeding that of a typical individual appliance.

Referring now to FIG. 6, illustrated is an embodiment of a computing device cluster or appliance cluster 600. A plurality of appliances 200a-200n or other computing devices, sometimes referred to as nodes, such as desktop computers, servers, rack mount servers, blade servers, or any other type and form of computing device may be joined into a single appliance cluster 600. Although referred to as an appliance cluster, in many embodiments, the cluster may operate as an application server, network storage server, backup service, or any other type of computing device without limitation. In many embodiments, the appliance cluster 600 may be used to perform many of the functions of appliances 200, WAN optimization devices, network acceleration devices, or other devices discussed above.

In some embodiments, the appliance cluster 600 may comprise a homogenous set of computing devices, such as identical appliances, blade servers within one or more chassis, desktop or rack mount computing devices, or other devices. In other embodiments, the appliance cluster 600 may comprise a heterogeneous or mixed set of devices, including different models of appliances, mixed appliances and servers, or any other set of computing devices. This may allow for an appliance cluster 600 to be expanded or upgraded over time with new models or devices, for example.

In some embodiments, each computing device or appliance 200 of an appliance cluster 600 may comprise a multi-core appliance, as discussed above. In many such embodiments, the core management and flow distribution methods discussed above may be utilized by each individual appliance, in addition to the node management and distribution methods discussed herein. This may be thought of as a two-tier distributed system, with one appliance comprising and distributing data to multiple nodes, and each node comprising and distributing data for processing to multiple cores. Accordingly, in such embodiments, the node distribution system need not manage flow distribution to individual cores, as that may

be taken care of by a master or control core as discussed above.

In many embodiments, an appliance cluster 600 may be physically grouped, such as a plurality of blade servers in a chassis or plurality of rack mount devices in a single rack, but in other embodiments, the appliance cluster 600 may be distributed in a plurality of chassis, plurality of racks, plurality of rooms in a data center, plurality of data centers, or any other physical arrangement. Accordingly, the appliance cluster 600 may be considered a virtual appliance, grouped via common configuration, management, and purpose, rather than a physical group.

In some embodiments, an appliance cluster 600 may be connected to one or more networks 104, 104'. For example, referring briefly back to FIG. 1A, in some embodiments, an appliance 200 may be deployed between a network 104 joined to one or more clients 102, and a network 104' joined to one or more servers 106. An appliance cluster 600 may be similarly deployed to operate as a single appliance. In many embodiments, this may not require any network topology changes external to appliance cluster 600, allowing for ease of installation and scalability from a single appliance scenario. In other embodiments, an appliance cluster 600 may be similarly deployed as shown in FIGs. 1B-1D or discussed above. In still other embodiments, an appliance cluster may comprise a plurality of virtual machines or processes executed by one or more servers. For example, in one such embodiment, a server farm may execute a plurality of virtual machines, each virtual machine configured as an appliance 200, and a plurality of the virtual machines acting in concert as an appliance cluster 600. In yet still other embodiments, an appliance cluster 600 may comprise a mix of appliances 200 or virtual machines configured as appliances 200. In some embodiments, appliance cluster 600 may be geographically distributed, with the plurality of appliances 200 not co-located. For example, referring back to FIG. 6, in one such embodiment, a first appliance 200a may be located at a first site, such as a data center and a second appliance 200b may be located at a second site, such as a central office or corporate headquarters. In a further embodiment, such geographically remote appliances may be joined by a dedicated network, such as a T1 or T3 point-to-point connection; a VPN; or any other type and form of network. Accordingly, although there may be additional communications latency compared to co-located appliances 200a-200b, there may be advantages in reliability in case of site power failures or communications outages, scalability, or other benefits. In some embodiments, latency issues may be reduced through geographic or network-based distribution of data flows. For example, although configured as an appliance cluster 600,

communications from clients and servers at the corporate headquarters may be directed to the appliance 200b deployed at the site, load balancing may be weighted by location, or similar steps can be taken to mitigate any latency.

Still referring to FIG. 6, an appliance cluster 600 may be connected to a network via
5 a client data plane 602. In some embodiments, client data plane 602 may comprise a communication network, such as a network 104, carrying data between clients and appliance cluster 600. In some embodiments, client data plane 602 may comprise a switch, hub, router, or other network devices bridging an external network 104 and the plurality of appliances 200a-200n of the appliance cluster 600. For example, in one such embodiment, a router may
10 be connected to an external network 104, and connected to a network interface of each appliance 200a-200n. In some embodiments, this router or switch may be referred to as an interface manager, and may further be configured to distribute traffic evenly across the nodes in the application cluster 600. Thus, in many embodiments, the interface master may comprise a flow distributor external to appliance cluster 600. In other embodiments, the
15 interface master may comprise one of appliances 200a-200n. For example, a first appliance 200a may serve as the interface master, receiving incoming traffic for the appliance cluster 600 and distributing the traffic across each of appliances 200b-200n. In some embodiments, return traffic may similarly flow from each of appliances 200b-200n via the first appliance 200a serving as the interface master. In other embodiments, return traffic from each of
20 appliances 200b-200n may be transmitted directly to a network 104, 104', or via an external router, switch, or other device. In some embodiments, appliances 200 of the appliance cluster not serving as an interface master may be referred to as interface slaves 610A-610N.

The interface master may perform load balancing or traffic flow distribution in any of a variety of ways. For example, in some embodiments, the interface master may comprise a
25 router performing equal-cost multi-path (ECMP) routing with next hops configured with appliances or nodes of the cluster. The interface master may use an open-shortest path first (OSPF) In some embodiments, the interface master may use a stateless hash-based mechanism for traffic distribution, such as hashes based on IP address or other packet information tuples, as discussed above. Hash keys and/or salt may be selected for even
30 distribution across the nodes. In other embodiments, the interface master may perform flow distribution via link aggregation (LAG) protocols, or any other type and form of flow distribution, load balancing, and routing.

In some embodiments, the appliance cluster 600 may be connected to a network via a

server data plane 604. Similar to client data plane 602, server data plane 604 may comprise a communication network, such as a network 104', carrying data between servers and appliance cluster 600. In some embodiments, server data plane 604 may comprise a switch, hub, router, or other network devices bridging an external network 104' and the plurality of appliances 200a-200n of the appliance cluster 600. For example, in one such embodiment, a router may be connected to an external network 104', and connected to a network interface of each appliance 200a-200n. In many embodiments, each appliance 200a-200n may comprise multiple network interfaces, with a first network interface connected to client data plane 602 and a second network interface connected to server data plane 604. This may provide additional security and prevent direct interface of client and server networks by having appliance cluster 600 server as an intermediary device. In other embodiments, client data plane 602 and server data plane 604 may be merged or combined. For example, appliance cluster 600 may be deployed as a non-intermediary node on a network with clients 102 and servers 106. As discussed above, in many embodiments, an interface master may be deployed on the server data plane 604, for routing and distributing communications from the servers and network 104' to each appliance of the appliance cluster. In many embodiments, an interface master for client data plane 602 and an interface master for server data plane 604 may be similarly configured, performing ECMP or LAG protocols as discussed above.

In some embodiments, each appliance 200a-200n in appliance cluster 600 may be connected via an internal communication network or back plane 606. Back plane 606 may comprise a communication network for inter-node or inter-appliance control and configuration messages, and for inter-node forwarding of traffic. For example, in one embodiment in which a first appliance 200a communicates with a client via network 104, and a second appliance 200b communicates with a server via network 104', communications between the client and server may flow from client to first appliance, from first appliance to second appliance via back plane 606, and from second appliance to server, and vice versa. In other embodiments, back plane 606 may carry configuration messages, such as interface pause or reset commands; policy updates such as filtering or compression policies; status messages such as buffer status, throughput, or error messages; or any other type and form of inter-node communication. In some embodiments, RSS keys or hash keys may be shared by all nodes in the cluster, and may be communicated via back plane 606. For example, a first node or master node may select an RSS key, such as at startup or boot, and may distribute this key for use by other nodes. In some embodiments, back plane 606 may comprise a

network between network interfaces of each appliance 200, and may comprise a router, switch, or other network device (not illustrated). Thus, in some embodiments and as discussed above, a router for client data plane 602 may be deployed between appliance cluster 600 and network 104, a router for server data plane 604 may be deployed between
5 appliance cluster 600 and network 104', and a router for back plane 606 may be deployed as part of appliance cluster 600. Each router may connect to a different network interface of each appliance 200. In other embodiments, one or more planes 602-606 may be combined, or a router or switch may be split into multiple LANs or VLANs to connect to different interfaces of appliances 200a-200n and serve multiple routing functions simultaneously, to
10 reduce complexity or eliminate extra devices from the system.

In some embodiments, a control plane (not illustrated) may communicate configuration and control traffic from an administrator or user to the appliance cluster 600. In some embodiments, the control plane may be a fourth physical network, while in other embodiments, the control plane may comprise a VPN, tunnel, or communication via one of
15 planes 602-606. Thus, the control plane may, in some embodiments, be considered a virtual communication plane. In other embodiments, an administrator may provide configuration and control through a separate interface, such as a serial communication interface such as RS-232; a USB communication interface; or any other type and form of communication. In some embodiments, an appliance 200 may comprise an interface for administration, such as a front
20 panel with buttons and a display; a web server for configuration via network 104, 104' or back plane 606; or any other type and form of interface.

In some embodiments, as discussed above, appliance cluster 600 may include internal flow distribution. For example, this may be done to allow nodes to join/leave transparently to external devices. To prevent an external flow distributor from needing to be repeatedly
25 reconfigured on such changes, a node or appliance may act as an interface master or distributor for steering network packets to the correct node within the cluster 600. For example, in some embodiments, when a node leaves the cluster (such as on failure, reset, or similar cases), an external ECMP router may identify the change in nodes, and may rehash all flows to redistribute traffic. This may result in dropping and resetting all connections. The
30 same drop and reset may occur when the node rejoins. In some embodiments, for reliability, two appliances or nodes within appliance cluster 600 may receive communications from external routers via connection mirroring.

In many embodiments, flow distribution among nodes of appliance cluster 600 may

use any of the methods discussed above for flow distribution among cores of an appliance. For example, in one embodiment, a master appliance, master node, or interface master, may compute a RSS hash, such as a Toeplitz hash on incoming traffic and consult a preference list or distribution table for the hash. In many embodiments, the flow distributor may provide the hash to the recipient appliance when forwarding the traffic. This may eliminate the need for the node to recompute the hash for flow distribution to a core. In many such embodiments, the RSS key used for calculating hashes for distribution among the appliances may comprise the same key as that used for calculating hashes for distribution among the cores, which may be referred to as a global RSS key, allowing for reuse of the calculated hash. In some embodiments, the hash may be computed with input tuples of transport layer headers including port numbers, internet layer headers including IP addresses; or any other packet header information. In some embodiments, packet body information may be utilized for the hash. For example, in one embodiment in which traffic of one protocol is encapsulated within traffic of another protocol, such as lossy UDP traffic encapsulated via a lossless TCP header, the flow distributor may calculate the hash based on the headers of the encapsulated protocol (e.g. UDP headers) rather than the encapsulating protocol (e.g. TCP headers). Similarly, in some embodiments in which packets are encapsulated and encrypted or compressed, the flow distributor may calculate the hash based on the headers of the payload packet after decryption or decompression. In still other embodiments, nodes may have internal IP addresses, such as for configuration or administration purposes. Traffic to these IP addresses need not be hashed and distributed, but rather may be forwarded to the node owning the destination address. For example, an appliance may have a web server or other server running for configuration or administration purposes at an IP address of 1.2.3.4, and, in some embodiments, may register this address with the flow distributor as its internal IP address. In other embodiments, the flow distributor may assign internal IP addresses to each node within the appliance cluster 600. Traffic arriving from external clients or servers, such as a workstation used by an administrator, directed to the internal IP address of the appliance (1.2.3.4) may be forwarded directly, without requiring hashing.

G. Systems and Methods for Managing Cloud-Based Services via a Network

The present disclosure is directed towards systems and methods for managing cloud-based services via a network. In particular, the present solution relates to systems and methods for managing delivery of cloud-based services using metadata related to instances of the services. For example, systems and methods of the present solution can determine

instances of services, retrieve metadata about those instances, and use the retrieved metadata to tune features of an application delivery controller.

In cloud environments, instances of services may be distributed across multiple regions, availability zones, and more. An example of an instance of a service can include a load balancer, an application delivery controller, a virtual server configured as a load balancer, a firewall, a router, rate-limiting system, optimization system, or an access control system. The instance can be executed by a device intermediary to clients and one or more services of the cloud environment. The cloud environment can refer to an infrastructure as a service (IaaS) environment, software as a service (SaaS) environment, or platform as a service (PaaS) environment. An IaaS environment can refer to cloud computing where shared computing resources, data and information are provided to computing devices on-demand. A device intermediary to the servers of the cloud environment and client devices, such as an application delivery controller (ADC), can facilitate providing or delivering the services provided by the cloud environment. A controller of the present disclosure can query a cloud application programming interface (API), such as an IaaS API, to discover or determine metadata corresponding to instances of the services provided by the IaaS cloud. The metadata can include account information associated with a provider of the service, such as a distribution, available resources, or resource consumption. The controller can use this metadata to automatically tune features or functionalities of an application delivery controller, thus bringing cloud awareness into the ADC systems. Using metadata associated with a cloud environment allows the controller to more efficiently provide services for instances that are distributed across the cloud environment by reducing overhead, cost, latency, throughput, or load.

For example, hosted systems in public or private cloud environments can include one or more APIs to identify or determine cloud partitions or boundaries. The APIs allows the hosted system or systems that interface with the hosted system to monitor internal dynamism to identify the environmental impact on overhead, cost, latency, throughput, or load. Once the system is aware of the infrastructure and its dynamism by using the available metadata, the system can automatically tune itself such that the system is more suitable to the cloud environment, and can perform more efficiently. Thus, systems and methods of the present disclosure can: 1) identify cloud partitions/boundaries; 2) monitor the internal dynamism in the cloud; 3) determine the cloud/environmental impact on the system; and 4) tune the hosted system to adapt to the environmental variables.

Providers of private or public cloud infrastructure can maintain an informational store or data repository that includes account information for customers of the cloud infrastructure. The system can use this information store as a database to track of user account information and their resource allocations for billing, maintenance, analysis, or other purposes. Cloud
5 APIs can be configured to query the database. Using these APIs the system can fetch metadata about instances, including fetching the instances' placement in the cloud, resource allocation (CPU, memory, bandwidth), type of reservation, type of connectivity, or other attributes or characteristics associated with the instance.

An ADC can use the retrieved metadata to perform load balancing operations. For
10 example, the ADC can use the metadata to drive or determine a cloud weight and use the cloud weight in load balancing algorithms to improve the efficiency of the cloud environment by increase throughput or reducing latency. Further, using the metadata to tune the cloud environment can reduce load on the system and cloud infrastructure, reduce network overhead, reduce cost and provide improved load balancing service.

To do so, the system can be configured with a control plane daemon that periodically
15 fetches some or all instances of services provided via the cloud environment, and meta-data or account information related to the instances. The control plane can refer to a part of a router architecture that is configured to draw or determine network topology, or the information in a routing table that determines how to direct, forward, route, or relay incoming
20 packets. The system can analyze the meta-data and identify, generate, or determine cloud weights for system specific configured entities. The system can associate the identified cloud weights with every instance and propagate the weights to a data plane. For example, a data plane can refer to a user plane, forwarding plane, carrier plane or bearer plane. The data can be part of a network that carries user traffic or client traffic. The data plane can be a part of
25 telecommunications architecture. The weights can indicate or refer to a logical measurement of cloud environmental impact to the hosted systems. These weights can be used with various load-balancing algorithms (as weighted LB algorithm) to achieve improved system performance.

High availability cloud IaaS environments can include a pair of high availability load-
30 balancers where one of the load-balancers serves as the primary and is in an active state and the other load balancer serves as a secondary and is in a passive state. For example, a cloud computing environment can provide high availability of application resources using a network intermediary device (e.g., NetScaler® manufactured by Citrix Systems, Inc. of Ft.

Lauderdale, Florida) that is deployed in an active-standby pair. In an active-standby pair, a first intermediary device actively services client traffic, while a second intermediary device is maintained in a standby or passive mode. For example, and in some embodiments, a first intermediary device can be in an active mode and can parse and process an application session such as an ICA session. The first intermediary device can be paired with a second intermediary device that is maintained in a standby mode. The first intermediary device can maintain significant state information, which may be dynamic and change on a per-packet basis (e.g., as each packet is being processed). If the first intermediary device fails, then the standby device can become the new primary device. The secondary device can become the new primary device and resume the session without affecting or with minimally affecting the user experience provided by the client device.

However, one of the load-balancers of the pair may have better connectivity to the load-balanced services. By analyzing the retrieved metadata, the system can discover the relative locations of both servers and the load-balancers and automatically select which load-balancer would be optimal for traffic, and make that the primary load-balancer. The result is a better performing service.

For example, a load balancer can direct client traffic among eight backend servers (S1 to S8). These servers and load balancer instances may be hosted in different cloud partitions (such as data centers, zones, etc.) for various reasons. The active (primary) ADC may attract client traffic. The primary ADC can select any one server (out of eight servers) and forward the client request. However, approximately 75% (6 out of 8) of the traffic may be forwarded across different cloud partitions for both client requests and server responses, which would negatively impact in latency, throughput, overhead, etc. By using retrieved account information or meta-data, the system can drive the cloud weight associated with the backend servers (or services) with respect to the primary and secondary instances.

The system can determine the cloud weights for the load balancers by calculating a sum of cloud weights associated with services with respect to both primary and secondary load balancer instances. This total cloud weight can represent or indicate a cloud cost for load balancing traffic between load balancers and servers. For example, the cloud weight for secondary load balancer may be greater than the cloud weight for the primary load balancer because a majority of the services (S1 to S6) are closer to the passive or secondary load balancer instance. Thus, processing (load balancing) client traffic at the secondary load balancer instance may be better for throughput, latency, overhead, or cost than processing the

same client traffic at the primary load balancer instance.

But since the current primary is attracting client traffic, which is inefficient based on the analysis performed using the cloud information, the system can trigger HA failover of the primary load balancer instance so that the secondary load balancer instance becomes the new
5 primary/active load balancer instance and starts processing (load balancing) client network traffic.

In some cases, an ADC may execute multiple virtual servers that each provide a load balancer instance bound to one or more servers. In such cases, the system can calculate could weights across all virtual servers. The system can sum the cloud weight across all virtual
10 servers executed by an ADC to determine a total cloud weight for the ADC.

In some embodiments, the system can improve performance by caching server content locally. For example, the ADC can use cloud information or meta-data to dynamically cache server content locally based on various cloud parameters, such as automatically enabling caching for contents served from specific cloud partitions, or applying apply thresholds based
15 on cloud parameters.

In some embodiments, the system can use cloud information or meta-data to perform rate limiting or throttling techniques. For example, the system can improve Access Control Lists (ACLs) and policies to operate on cloud parameters to provide control over flows and improve bandwidth utilization. The system can use include cloud parameters as part of
20 logging and use the cloud parameters in to filter data to generated reports with helpful analytical information.

While the techniques disclosed herein can be used in an ADC environment, the systems and methods of the present disclosure can improve the operation of networking devices such as firewalls, routers, optimization systems, rate-limiting systems, or access
25 control systems.

Now referring now to FIG. 7A, a block diagram of an embodiment of a system for managing cloud-based services via a network is shown. The system 700 can include one or more appliances 200a-n. The appliances 200a-n can be configured with one or more features or functionality. The appliances 200a-n can each be configured with similar functionality, or
30 different functionality. In some embodiments, appliance 200a includes a controller 705. The controller 705 can include an interface 710 configured to interface or communicate with clients 102a-n, server 106a-n, and meta-data server 735. The controller 705 can include a

cloud daemon 715 that retrieves, from metadata server 735, metadata or account information related to instances of services in the cloud environment. The controller 705 can include a weighting engine 720 that uses the metadata to generate weights for each instance. The controller 705 can include a load balancer 725 that is configured to use the weights generated
5 by the weighting engine 720 to select an instance to direct network traffic. The controller 705 can include vserver 275. The controller 705 can execute a vserver 275. The controller 705 can execute on a vserver 275 that executes on one or more processors of appliance 200a.

The system 700 can include or access a meta-data server 735. The meta-data server can be maintained or managed by an administrator or other entity associated with the cloud
10 environment, such as a provider of an IaaS service. An IaaS service can include, for example, self-service models for accessing, monitoring, and managing remote datacenter infrastructures, such as compute (virtualized or bare metal), storage, networking, and networking services (e.g. firewalls). A user or customer of an IaaS cloud service can purchase IaaS based on consumption, similar to electricity or other utility billing, rather than purchase
15 hardware outright. In some cases, customers of the IaaS may manage applications, data, runtime, middleware, and operating systems. However, the IaaS provider may manage virtualization, servers, hard drives, storage, and networking. The IaaS provider can offer databases, messaging queues, and other services above the virtualization layer.

The meta-data server 740 can include or access a meta-data information store 740 or
20 meta-data database 740 that includes one or more meta-data data structures storing meta-data. Meta-data can include information about or associated with accounts of the cloud environment. Meta-data can include account information related to the cloud environment. For example, a provider of cloud infrastructure (or an IaaS provider) can maintain an informational store or meta-data database 740 about their customer accounts. The meta-data
25 database 740 can include customer or user account information. The account information can include the customer's resource allocations that indicate the quantity or type of resources consumed by the customer. The IaaS provider can use the user's resource allocation information stored in meta-data database 740 for billing, maintenance, analysis, etc.

In some cases, the system 700 can include an active-standby appliance pair that
30 includes a first intermediary device 200a and a second intermediary device 200b. The first device 200a and the second device 200b can have different placements. The first device 200a can be placed in a first data center, while the second device 200b can be placed in a second data center. The first and second devices can be in different zones, partitions in a cloud

environment, or geographic areas. A partition can depict a division of an IaaS and define an area of an IaaS cloud used by customer of the IaaS. A partition can be made at one or more levels, such as provider level, region level, zone level or rack level. Partition groups can be defined in the partition definition and stored in the meta-data database 740. A partition group
5 (or network partition) can include an area of an IaaS that may be bound by one network of an IaaS (e.g., region). Each region can demarcate a separate geographical area. Each region can have multiple, isolated locations that are referred to as zones. Thereby, several partitions can be included inside a network partition. In some cases, multiple partition groups can have multiple partitions for the purpose of high availability. For example, for the purpose of high
10 availability, multiple partitions can be defined to instruct an appliance 200a to spawn instances in multiple areas (e.g., region, zone or rack). Instances spawned in multiple availability zones can allow the system to remain functional even if one availability zone is down.

In some cases, the active-standby appliance pair can be referred to as an appliance
15 pair, appliance pair, intermediary appliance pair, or device. The appliance pair can include one or more functionality or component of application cluster 600 depicted in FIG. 6. The first device 200a of the active-standby pair can be referred to as a first intermediary device 200a, primary intermediary device 200a, primary device 200a, initial primary device 200a, or original primary device 200a. The second device 200b of the active-standby pair can be
20 referred to as a second intermediary device 200b, secondary intermediary device 200b, second device 200b, initial standby device 200b, and new primary device 200b. The intermediary devices 200a and 200b, or appliance pair, is intermediary to a plurality of servers 106a-n and to a plurality of clients 102a-n. The appliances 200a and 200b can manage or optimize the delivery of IaaS services, website content, services, resources or
25 applications provided via the servers 106a-n over the network 104 or network 104', by providing application-level security, optimization, or traffic management. The appliance pair may be the same or similar to the appliance cluster 600 described above with respect to FIGs. 1-6, or include one or more functionality or components thereof. In some embodiments, the clients 102a-n are the same or similar to the clients 102a-n and the servers 106a-n are the
30 same or similar to the servers 106a-n as described above with respect to FIGs. 1-6, or include one or more functionality or components thereof. In some embodiments, the clients 102 and servers 106 of system 700 are further configured to perform functions or provide features for authenticating a client. The controller 705, interface 710, cloud daemon 715, weighting

engine 720, or load balancer 725 can include digital circuitry, hardware, one or more processors, memory or software designed, constructed and configured to facilitate managing cloud-based services.

Still referring to FIG. 7A, and in further detail, appliance 200a includes an interface 710. The interface 710 can include an application programming interface, a graphical user interface, communication ports, or be configured with one or more communication or networking protocols, such as TCP/IP, etc. The interface 710 can be configured to receive or intercept data packets, receive data packets, monitor network traffic, or otherwise obtain or transmit/convey information via network 104 or network 104'.

In some embodiments, the interface 710 receives a request from a client 102a of the plurality of clients 102a-n. The interface 710 may receive the request via a networking protocol, communication protocol, application layer protocol, transport protocol, or encryption protocol. Protocols can include, e.g., HTTP, TCP, ICA, or HDX. The protocol can be a stateful protocol or a stateless protocol. The request may include information about the client or the resource to which the client 102a is requesting access. In some embodiments, the resource provided by server 106a-n may be a secure resource for which a client or user of the client device may require authentication, authorization or approval before access is granted.

In some embodiments, the interface 710 designed and constructed to receive network communications from one or more clients 102a-n and forward or direct network communications to one or more servers 106a-n. The interface 710 can be the same or similar to the interfaces 608 or 610a-n described above with respect to FIGs. 1-6, or include one or more functionality or components thereof. The interface 710 can receive data packets from a client 102a. The interface 710 can forward or relay the data packets received from client 102a to the first device 200a. For example, the interface 710 can be configured to automatically forward data packets received from a client agent 120a to the first device 200a. The interface 710 can be further configured to attract network traffic from clients 102a-n based on a state or mode of the appliance 200a. For example, appliance 200a may be the primary device 200a. The primary device 200a can refer to the intermediary device of the active-standby intermediary appliance pair that is currently in the active mode. Thus, the interface 710 of appliance 200a can receive network traffic from clients 102a-n if appliance 200a is currently the primary device configured to actively service requests from clients, and forward the requests to one or more servers 106a-n.

In some embodiments, the controller 705 includes a cloud daemon 715 designed and constructed to receive account information related to each of a plurality of instances of one or more services provided by one or more servers of a cloud environment. The cloud daemon 715 can be configured to execute on one or more processors. The cloud daemon 715 can be configured to execute or run as a background process. In some cases, the cloud daemon 715 can be configured based on a time interval, periodically, or responsive to a condition or trigger event. For example, the cloud daemon 715 can execute or run every 1 minute, 5 minutes, 10 minutes, 30 minutes, hourly, every 12 hours, daily, overnight, etc. In some cases, the cloud daemon 715 can be configured to execute or run responsive to an indication from a user or administrator of the cloud environment. In some embodiments, the cloud daemon 715 can be configured to run based on a condition, such as a determination that a new instance of a service has launched, a failover of an instance, a termination of an instance, a new request from a client 102a, or an addition, update or modification of a customer account.

The controller 705 (e.g., via interface 710 or cloud daemon 710) can determine or identify instances. An instance can refer to an instance of an appliance 200a-n, a virtual server 275, or load balancer 725 that facilitate providing the IaaS service to the one or more clients 102a-n. An instance of the appliance 200a, vserver 275, or load balancer 725 can direct packets to one or more servers 106a that provide one or more IaaS services. The instances can include one or more available instances. The instances can include an active instance, such as a primary appliance 200a, or a passive instance, such as a secondary appliance 200b. The instances can include intermediary devices 200a-n that are available to direct network packets. The instances can include intermediary devices 200a-n that are in one or more partitions of the cloud environment. The instances can include vservers 275 that are executing on the appliance 200a.

For example, an active intermediary device 200a can include one or more vservers 275 that are executing to direct network traffic from multiple clients 102a-n to multiple servers 106a-n. The cloud daemon 715 can ping, poll, request, or otherwise identify the instances. For example, the cloud daemon 715 can generate a ping or request for a response from all active instances. Responsive to the ping or request, the cloud daemon 715 can receive responses from each instance. The response can include an identifier of the instance. The response can include an account identifier of a customer associated with the instance. For example, the response can identify an IaaS service provided by a server 106a-n. The instance may be directing network traffic from a client 102a to the server 106a to access the

IaaS service.

Responsive to identifying the active instances, the controller 705 (e.g., via cloud daemon 715) can retrieve or fetch meta-data or account information associated with the instance. The meta-data can include meta-data about the service to which the instance provides access. The controller 705 can retrieve the meta-data from a meta-data server 735. The controller 705 can use a cloud application programming interface (API) to retrieve meta-data associated with the service provided via the instance. For example, the controller 705 can use the cloud API to generate one or more queries. The query may include a request for information regarding the service's resource allocation, cloud placement, resource consumption, type of reservation, or type of connectivity. Cloud placement of the service can indicate partition information, such as a zone, data center, geographic area, network partition, or drive partition on which the service executes. A type of reservation can be used to provision a machine that has sufficient capacity for a service. A type of reservation can include a policy used to provision machines with sufficient capacity for a service, such as sufficient storage, computing power, processor speed, memory, applications, graphical processing unit, bandwidth, etc. The reservation can include a virtual reservation. A virtual reservation can include a share of the memory, CPU, networking, and storage resources available to a server 106a-n to provide or execute an IaaS service. A type of connectivity can refer to a type of network 104' the server 106a uses to communicate with the appliances 200a-n or clients 102a-n. The type of connectivity can indicate an amount of bandwidth available for a service.

The controller 705 can generate one or more queries using the cloud API. In some cases, the controller 705 can be configured with one or more cloud APIs. The controller 705 can select a cloud API from a plurality of cloud APIs stored in database of the controller 705. The controller 705 can select the cloud API based on an identifier associated with the instances or services provided via the instances. For example, the controller 705 can determine, based on the identifier of the instance or the service, an account identifier. The controller 705 can use the account identifier to perform a lookup in a database stored in memory of the controller 705 to select a cloud API mapped to the account identifier. For example, a cloud monitoring API can include AMAZON CLOUDWATCH™ provided by AMAZON WEB SERVICES, Inc. of Seattle, Washington.

Using the cloud API, the controller 705 can generate one or more queries to retrieve meta-data about the service. The query can include a query for placement information for a

service. Placement information for an instance can refer to a partition in a cloud environment. Placement information for an instance can refer to a partition in a cloud environment the instance accesses to provide a service. For example, the IaaS cloud service may execute at the corresponding placement. Thus, the placement information for the
5 instance can refer to the placement of the service to which the instance provides access.

The controller 705 can transmit the generated query to the meta-data server 735 via network 104'. The controller 705 can receive, from the meta-data server 735, a response to the query that includes meta-data for the instance. Meta-data for the instance can include meta-data about the service to which the instance provides access. Meta-data for the instance
10 can include meta-data for the service associated with the instance.

In some cases, the controller 705 can generate and transmit a separate query to the meta-data server 735 for each active instance and associated service. In some cases, the controller 705 can generate and transmit a query that identifies multiple services provided by multiple instances. The controller 705 can receive a response that includes metadata for one
15 or more services. For example, the controller 705 can receive a response for each query, or a response that includes responses to multiple queries.

The controller 705 can include a weighting engine 720 designed and constructed to determine a weight for each instance using the meta-data received from the meta-data server 735. The cloud weight can indicate a level of user experience based on cloud performance
20 characteristics, such as throughput or latency. The cloud weight can indicate a logical measurement of cloud environmental impact to the hosted system, such as the IaaS services. The weight can indicate a performance of the instance of the service provided via the cloud environment.

The controller 705 can be configured with one or more techniques, functions or
25 algorithms to determine the cloud weight for an instance using the retrieved meta-data. For example, a first instance of the service can refer to an appliance 200a in a first zone directing network traffic to a service provided by a server 106a also in the first zone. A second instance of the service can refer to an appliance 200b in a second zone directing network traffic to the service provided by the server 106a in the first zone. The instances can be
30 active instances or potential, available instances. The weighting engine 720 can use the retrieved meta-data to determine that for the first instance of the service, both the service and the appliance are in the first zone, whereas for the second instance of the service, the service

and the appliance are in different zones. Thus, the weighting engine 720 can assign a higher weight to the first instance because the appliance and the service are in the same zone, which may result in reduced latency and increased throughput. The weighting engine 720 can assign a lower weight to the second instance because the appliance and the service are in different zones, which may result in increased latency and reduced throughput. For example, the weighting engine 720 can assign a weight of 1, 2, 3, 5, 10 or another number to the first instance. The weighting engine 720 can assign a weight of negative 1 (-1), 0, 1, 2, 3 or some other weight to the second instance. The weighting engine 720 can assign weights that indicate a level of performance provided by the cloud environment for the instance of the service.

The weighting engine 720 can assign weights based on a distance between the appliance and the server providing the service. For example, the weighting engine 720 can determine a geographic location for the appliances 200a-n. The weighting engine 720 can use the received meta-data to determine a geographic location of the servers 106a-n providing the services. The weighting engine 720 can determine a distance between each appliance and each service based on the geographic locations. For example, the weighting engine 720 can query a mapping engine or mapping tool that determine a geographic distance between the appliance 200 and the service. In another example, the weighting engine 720 can cause an appliance to ping a server and use the ping response time to determine a distance or response time. The weighting engine 720 can assign a weight based on the distance or the response time. For example, the weighting engine 720 can rank the instances of the appliance and service based on the response, and assign a highest weight to the instance of the service corresponding to the shortest response time, a second highest weight to the instance of the service corresponding to the second shortest response time, etc.

In some examples, the weighting engine 720 can assign the weight to the instances of the services based on one or more factors such as placement, a reservation type or capacity of the instance of the service, bandwidth, or type connectivity. For example, the weighting engine 720 can determine, based on the received meta-data, the weight for a first instance of the service as follows: $\text{instance_one_weight} = +5$ (due to same zone) $+ 2$ (due to available bandwidth of 500 Mbps) $= 7$. The weighting engine 720 can determine, based on the received meta-data, the weight for the second instance of the service as follows: $\text{instance_two_weight} = +1$ (due to different zones) $+ 4$ (due to available bandwidth of 1000 Mbps) $= 5$. The weighting engine 720 can compare the first weight with the second weight to

determine the instance corresponding to the highest weight. The controller 705 can then select the instance corresponding to the highest weight to direct network traffic.

In some embodiments, the controller 705 includes a load balancer 725 designed and constructed to use a load balancing function and the determined weights for each of the instances to select an instance to direct network traffic from the client. The load balancer 725 can distribute workloads across multiple computing resources to optimize resource use, maximize throughput, minimize response time, and avoid overload of a single resource. In some cases, the load balancing function can cause the load balancer to select the instance corresponding to the highest weight. For example, in a high availability load balancing system, the system can include a first instance of an intermediary device and a second instance of an intermediary device. The first instance can be the primary intermediary device 200a, and the second instance can be a secondary intermediary device 200b. The primary may be the active device 200a, and the secondary may be the passive device 200b. The controller 705 can determine the weights for instances of services using the active device 200a. The controller 705 can also determine the weights for potential instances of services using the passive device 200b. These may be referred to as potential instances because the active device is currently directing traffic and the passive device is passive. Thus, the weights are for potential instances because instances of services using the passive device are not yet active. The controller 705 can compare the weights and determine, based on the weights, that the cloud performance would improve if the passive device 200b became the active device. In this example, the controller 705 can cause, instruct, or trigger a failover such that the passive device 200b becomes the new active device. The new active device can attract subsequent network traffic from one or more clients and direct the traffic to one or more servers 106a-n. Thus, the controller 705 can select the second instance of the appliance 200 to direct traffic from one or more clients.

In some embodiments, the load balancer 725 can use a load balancing algorithm (or load balancing technique) that defines criteria that the appliance 200a can use to select the server 106a-n to which to send client requests. When the load balancer 725 determines that the configured criteria are met for one server, the appliance can then select a different service. In some cases, the load balancer can select instances using a least connection technique. The least connections technique can select the instance with the least number of active connections to balance the load of the active requests among instances.

The load balancer 725 can provide for further granularity of configuration by using

weights. For example, the load balancer 725 can apply the determined weights to the least connections load balancing function. The load balancer 725 can select an instance based on a weighted value $N\{w\}$ of the following expression: $N\{w\} = (\text{Number of active transactions}) * (10000 / \text{weight})$.

- 5 For example, the first instance can have 50 active transactions and be assigned a weight of 3. The second instance can have 80 active transactions and be assigned a weight of 5. Thus, the weighted value $N\{w\}$ for the first instance can be $50 * (10000/3) = 166,667$. The weighted value $N\{w\}$ for the second instance can be $80 * (10000/5) = 160,000$. The load balancer 725 can use the weighted least connection technique to select the server with the
- 10 least $N\{w\}$ value for each request. The following Table 1 illustrates how a cloud aware load balancer 725 can use meta-data to generate weights to select instances to direct network traffic for the following requests:

Request #	$N\{w\}$ of first instance	$N\{w\}$ of second instance	Selected Instance
1	166,667	160,000	Second Instance
2	166,667	162,000	Second Instance
3	166,667	164,000	Second Instance
4	166,667	166,000	Second Instance
5	166,667	168,000	First Instance
6	170,000	168,000	Second Instance
7	170,000	170,000	Round Robin – First Instance or Second Instance

15 Table 1: Illustrative Example of cloud aware load balancing

As illustrated in Table 1 above, the second instance has a the least $N\{w\}$ value for the four requests and is selected to direct network traffic from the one or more clients. As each

request is received from one or more clients, the $N\{w\}$ for the second instance increases based on the least connection function. When the fifth request comes in, the first instance has the least $N\{w\}$, so the load balancer selects the first instance. When the sixth request is received from a client, the second instance has the least $N\{w\}$, so the load balancer selects the second instance. When the seventh request is received, both the first and second instances have the same $N\{w\}$. At this point, the load balancer can be configured to select one of the instances based on a round robin approach or randomly select one or the other instance, or default to the first instance or the second instance.

In some embodiments, the controller 705 can use the meta-data to facilitate caching data in memory or storage of an appliance. The controller 705 can use the received meta-data to determine the zone information for each service provided by each instance. The controller can determine, using the received meta-data, that a first instance of a first service is provided via a first zone, and a second instance of a second service is provided via a second zone. The controller 705 can determine, using the cloud information or the weights, that the controller 705 is located in, near, or has a shorter response time with the first zone as compared to the second zone. Thus, the controller 705 can determine to store or cache data related to the second instance in memory or storage of the controller 705 or appliance 200a in order to improve cloud performance. Data may include, for example, data files or applications that are frequently requested from, or provided by the second service.

Referring now to FIG. 7B, a block diagram of an embodiment of a method of maintaining a session is shown. The method 701 can be performed using one or more system or component described in FIGs. 1-7A, 8, or 9A-9C. In brief overview, and in some embodiments, a controller receives account information related to instances of services at step 780. At step 785, the controller determines a weight for each of the instances based on the account information. At step 790, the controller selects an instance to direct network traffic from a client.

Still referring to FIG. 7B, and in further detail, a controller receives account information related to instances of services at step 780. The controller can be intermediary to one or more clients and one or more servers that provide one or more services. The servers can be part of a cloud environment, such as an IaaS cloud. The cloud environment can include the controller, or the controller can provide clients access to the servers of the cloud environment. The controller can determine or identify each instance associated with the cloud environment. The controller can ping, poll, request, monitor, track or otherwise

determine the instances associated with the cloud environment. For example, the controller can determine that a high availability load balancer system includes a first instance corresponding to a primary load balancer, and a second instance corresponding to a secondary load balancer. The controller can further determine that the first instance can
5 direct traffic to one or more backend servers (e.g., S1 to S8). The controller can further determine that the second instance can also direct traffic to backend servers S1 to S8.

Upon identifying the instances configured for or available to direct client traffic to one or more services in the cloud environment, the controller can retrieve account information or metadata associated with the each of the plurality of instances. The controller
10 can use a cloud application programming interface to generate one or more queries to retrieve the account information for each of the plurality of instances. The one or more queries can be configured to retrieve information for at least one of a placement, resource allocation, type of reservation, or type of connectivity. For example, the controller can determine that each of the instances can direct traffic to one or more backend servers S1 to S8. The controller can
15 then generate queries for each instance that is configured to retrieve information about a corresponding service. For example, if the first instance can direct traffic to servers S1 to S8, the controller can generate queries for S1 to S8 for the first instance and transmit the queries to a meta-data server that stores meta-data for services or servers S1 to S8. Similarly, if the second instance can direct traffic to backend servers S1 to S8, the controller can generate
20 queries for the second instance to obtain metadata for corresponding servers S1 to S8, and transmit the queries to the corresponding meta-data server.

In some embodiments, the controller can generate a query that includes a request for information on multiple services S1 to S8. In some embodiments, the controller can generate a query for multiple instances. The controller can obtain different types of information about
25 the instance and backend service. For example, the controller can use the cloud application programming interface to generate a query to retrieve placement information for a first instance of the plurality of instances. The placement information can include, for example, a placement of a first service S1 to which the first instance directs network traffic. The placement information can include, for example, placement information corresponding to
30 each service (e.g., S1 to S8) to which the first instance directs network traffic. The controller can transmit the query to a metadata server maintained by a provider of the first instance or a provider of the services S1 to S8. The metadata server can include a database storing placement information for the first instance. The controller can receive the placement

information for the first instance from the metadata server responsive to the query.

In some embodiments, the controller can periodically receive or fetch updated account information. The controller can request account information response to receiving a request from a client, based on a time interval, or responsive to a condition or event (e.g., addition of
5 a new server, or a server going offline; a failover; new or updated load balancing algorithm or weight policies, etc.).

At step 785, the controller can determine a weight for each of the instances based on the account information. The weight can indicate a performance of the instance of the service provided via the cloud environment. The controller can determine weights using the
10 metadata or the account information. For example, the controller can determine a higher weight for an instance if the instance directs traffic to servers that are in the same zone as the instance. The controller can determine a lower weight for an instance if the instance directs traffic to servers that are in a different zone. The controller can assign positive or negative weights, or numerical weights ranging from a negative number to a positive number (e.g., -
15 100 to +100). The controller can assign weights as integers, binary values, fractions, or decimals. The controller can use an exponential weighting technique or a linear weighting technique. For example, as the distance between the instance (e.g., primary intermediary device) and the service increases, the weight assigned to the instance can increase exponentially or linearly.

For example, the controller can identify a first instance provided via a first zone of a cloud environment, and a second instance provided via a second zone of a cloud environment. The controller can determine a first weight for the first instance provided via a first zone of the cloud environment. The controller can determine the first weight based on the account information. For example, the account information can indicate a placement of
25 servers S1 to S8 to which the instance directs traffic. The controller can determine the first weight based on a distance between the first zone and servers S1 to S8. The controller can determine a second weight for a second instance provided via a second zone of the cloud environment different from the first zone. The controller can determine the second weight based on the account information and the distance between servers S1 to S8 and the second
30 instance. The controller can determine that the first weight is greater than the second weight because a majority of the services are closer to the first instance. The controller can select the first instance responsive to the first weight greater than the second weight.

In some embodiments, the controller can identify one or more virtual servers executing on a primary device instance, and one or more virtual servers executing on a secondary device instance. The virtual servers of the first and second instances can direct traffic to one or more backend servers (e.g., S1 to S8). The virtual servers may direct traffic to the same backend servers or different backend servers. To determine the cloud weight for the primary or secondary device instance, the controller can determine a weight for each virtual server based on the service or services to which the virtual server directs traffic, and then combine (e.g., sum) the weights to determine a total weight for the primary device instance or the secondary device instance.

The controller can determine the weights based on account information for the instances and an attribute of the device. Account information for an instance or a virtual server of the instance can refer to metadata of a service to which the instance directs traffic. An attribute of the device can refer to an attribute of the appliance, intermediary device, instance, or load balancer. Attributes can include, for example, location, placement, capacity, number of active connections, memory, CPU usage, bandwidth, or network connectivity. The controller can determine a weight based on the attributes of the instance itself or a virtual server of the instance, as well as using account information for the instance (e.g., account information associated with backend servers to which the instance directs traffic).

At step 790, the controller can select an instance of the plurality of instances to direct network traffic from a client. The controller can select the instance based on a load balancing function and the weight for each of the instances. For example, the controller can select the instance corresponding to highest weight if higher weights indicate improved performance. In another example, higher weights may indicate decreased performance; thus, the controller may select the instance corresponding to the lowest weight. In some embodiments, the controller can use a weighted load balancing function, such as a weighted least connections function illustrated in Table 1.

Referring now to FIG. 8, a functional block diagram of an embodiment of a system 800 for managing cloud-based services via a network is shown. The system 800 can be performed by one or more component or system depicted in FIGs. 1-7A, or 9A-9C. The system 800 can include a cloud environment 805. The cloud environment 805 can receive and transmit client traffic 850 from one or more clients 102a-n via a network (e.g., network 104). The cloud environment 805 can include, provide, or be associated with a load balancer 810, metadata server 820, and one or more services or servers 845a-e. The load balancer 810

can include one or more control plane daemons 825a-c. One or more of the control plane daemons can include a cloud awareness daemon 825a. The load balancer 810 can include a data plane 835. The data plane can include a VIP 2 840 (e.g., a vserver 275 configured to direct client traffic to one or more servers).

5 At step 865, the cloud awareness daemon 825a can fetch or retrieve meta-data from the meta-data server 820. At step 870, the cloud awareness daemon 825a determine cloud weights for each instance using the received metadata. For example, the cloud weights for the VIP 2 840 instances can be a, b, c, d, and e. At step 875, the cloud awareness daemon 825a passes the cloud weights to the data plane 835. At step 880, the load balancer 810 uses
10 the cloud weights with one or more load balancing algorithms to select the VIP 2 840 to direct traffic from the one or more clients 102a-n to the one or more services 845a-e.

Referring now to FIGs. 9A-9C, functional block diagrams of an embodiment of a cloud environment 900 having a system for managing cloud-based services via a network is shown. The cloud environment 900 provides access to services to one or more clients 102a-
15 n. The cloud environment includes one or more services or servers S1, S2, S3, S3, S4, S5, S6, S7 or S8. The cloud environment 900 can be configured with a high availability load balancer pair that includes an active appliance 200a and a passive appliance 200b. In the cloud IaaS environment 900, one of the load-balancers may have better connectivity to the load-balanced services. By analyzing the retrieved metadata, a controller of the load
20 balancers can discover the relative locations of both servers and the load-balancers and automatically select which load-balancer would be optimal for traffic from clients 102a-n to services or servers S1-S8, and make that load balancer the primary device.

As shown in FIG. 9A, the high availability load balancer includes an active 200a and passive 200b pair to direct client traffic among eight backend servers (S1 to S8). These
25 servers and load balancer instances might be hosted in different Cloud partitions (such as data centers or zones for various reasons. For example, the passive load balancer instance 200b and servers S1 to S6 can be located in zone 1. Within zone 1, the services S1 to S3 can be located in a first data center (DC 1) and the services S4 to S5 can be located in a second data center (DC 2). The active load balancer instance 200a and services S7 and S8 can be located
30 in zone 2. Within zone 2, the service S7 can be located in a third data center (DC 3) and the service S8 can be located in a fourth data center (DC 4).

Client traffic from clients 102a-n can be attracted by the active (primary) load

balancer instance 200a, which will select any one server (service) (out of eight servers (services) S1 to S8) and forward the client request. In the current situation, approximately 75% (6 out of 8) of the requests are forwarded across different cloud partitions for both client requests and server responses. The relatively high percentage (e.g., more than half) of requests forwarded across different cloud partitions can have a negatively impact on latency, throughput, or overhead. Thus, a controller can fetch metadata to determine cloud weights for the active and passive instances, and select one of the active or passive instances to direct traffic from clients.

FIG. 9B illustrates using retrieved meta-data to determine cloud weights for each instance for each service as follows: $a_1, a_2, a_3, a_4, a_5, a_6, a_7$ and a_8 are cloud weights calculated with respect to the active load balancer instance 200a; and $p_1, p_2, p_3, p_4, p_5, p_6, p_7$ and p_8 are cloud weights calculated with respect to the passive load balancer instance 200b.

Using these weights, the controller of the high availability load balancer can calculate a sum of cloud weights associated with all services with respect to both the active and passive instances 200a-b. This total cloud weight can indicate a cloud cost for load balancing traffic between the load balancers 200a-b and the servers S1 to S8.

The total Cloud weight on the primary/active instance 200a can be:

$$Ta = \sum_{k=1}^8 (a_k) = a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7 + a_8$$

The total Cloud weight on the passive/secondary instance 200b can be:

$$Tp = \sum_{k=1}^8 (p_k) = p_1 + p_2 + p_3 + p_4 + p_5 + p_6 + p_7 + p_8$$

As shown in this example, $Tp > Ta$, because a majority of the services (S1 to S6) are closer to the passive instance 200b. Therefore, processing (load balancing) client traffic at 200b may be better than processing the same at 200a, for better throughput, latency, overhead, or cost.

But since instance 200a is the current primary (attracting client traffic), which is inefficient, the controller can use the above cloud information and analysis to determine that it would be better to trigger HA Failover so that the secondary instance 200b becomes the new primary/active and starts processing (load balancing) traffic, thereby improving cloud
 5 performance.

FIG. 9C shows the optimized high availability load balancer system and traffic processing path after triggering failover based on the cloud weights. The instance 200b is the new active load balancer instance and attracts client traffic from clients 102a-n and directs the traffic to one or more servers S1 to S8. Since 75% of the servers are in the same zone as the
 10 instance 200b, the processing path is more efficient and cloud servers can be provided with reduced latency and increased throughput.

In some embodiments, multiple virtual servers on an appliance 200a or 200b might each be bound to a set of services S1 to S8. The controller can then calculate cloud weights across all virtual servers. For example, there can be 'n' virtual servers (VS) and each virtual
 15 server can be bound to 'k' services (S). Then the controller can determine the cloud weight as a summation of the weights associated with each virtual server and service as follows:

Total Cloud weight on Active/Primary instance 200a:

$$Ta = \sum_{i=1}^n (VSa_i) = \sum_{i=1}^n \left(\sum_{j=1}^k (Sa_{ij}) \right)$$

Total Cloud weight on Passive/Secondary instance 200b:

$$Tp = \sum_{i=1}^n (VSp_i) = \sum_{i=1}^n \left(\sum_{j=1}^k (Sp_{ij}) \right)$$

20 Where:

VSa_i – Cloud weight of i^{th} Virtual Server on primary instance 200a;

Sa_{ij} – Cloud weight of j^{th} Service which is bound to i^{th} Virtual Server on primary instance 200a;

VSp_i – Cloud weight of i^{th} Virtual Server on secondary instance 200b; and

Sp_{ij} – Cloud weight of j^{th} Service which is bound to i^{th} Virtual Server on secondary instance 200b.

It should be understood that the systems described above may provide multiple ones
 5 of any or each of those components and these components may be provided on either a
 standalone machine or, in some embodiments, on multiple machines in a distributed system.
 The systems and methods described above may be implemented as a method, apparatus or
 article of manufacture using programming and/or engineering techniques to produce
 software, firmware, hardware, or any combination thereof. In addition, the systems and
 10 methods described above may be provided as one or more computer-readable programs
 embodied on or in one or more articles of manufacture. The term "article of manufacture" as
 used herein is intended to encompass code or logic accessible from and embedded in one or
 more computer-readable devices, firmware, programmable logic, memory devices (e.g.,
 EEPROMs, ROMs, PROMs, RAMs, SRAMs, etc.), hardware (e.g., integrated circuit chip,
 15 Field Programmable Gate Array (FPGA), Application Specific Integrated Circuit (ASIC),
 etc.), electronic devices, a computer readable non-volatile storage unit (e.g., CD-ROM,
 floppy disk, hard disk drive, etc.). The article of manufacture may be accessible from a file
 server providing access to the computer-readable programs via a network transmission line,
 wireless transmission media, signals propagating through space, radio waves, infrared
 20 signals, etc. The article of manufacture may be a flash memory card or a magnetic tape. The
 article of manufacture includes hardware logic as well as software or programmable code
 embedded in a computer readable medium that is executed by a processor. In general, the
 computer-readable programs may be implemented in any programming language, such as
 LISP, PERL, C, C++, C#, PROLOG, or in any byte code language such as JAVA. The
 25 software programs may be stored on or in one or more articles of manufacture as object code.

References to "or" may be construed as inclusive so that any terms described using
 "or" may indicate any of a single, more than one, and all of the described terms.

While various embodiments of the methods and systems have been described, these
 embodiments are exemplary and in no way limit the scope of the described methods or
 30 systems. Those having skill in the relevant art can effect changes to form and details of the
 described methods and systems without departing from the broadest scope of the described
 methods and systems. Thus, the scope of the methods and systems described herein should

not be limited by any of the exemplary embodiments and should be defined in accordance with the accompanying claims and their equivalents.

CLAIMS

What is claimed is:

1. A method of managing cloud-based services via a network, comprising:
 - receiving, by a controller intermediary to a client and a cloud environment, account information related to each of a plurality of instances of one or more services provided by one or more servers of the cloud environment;
 - determining, by the controller, a weight for each of the instances based on the account information, the weight indicating a performance of the instance of the service provided via the cloud environment; and
 - selecting, by the controller based on a load balancing function and the weight for each of the instances, an instance of the plurality of instances to direct network traffic from a client.
2. The method of claim 1, comprising:
 - generating, by the controller using a cloud application programming interface, one or more queries to retrieve the account information for each of the plurality of instances, the one or more queries configured to retrieve information for at least one of a placement, resource allocation, type of reservation, or type of connectivity.
3. The method of claim 1, comprising:
 - generating, by the controller using a cloud application programming interface, a query to retrieve placement information for a first instance of the plurality of instances;
 - transmitting, by the controller, the query to a metadata server maintained by a provider of the cloud environment, the metadata server including a database storing placement information for the first instance; and
 - receiving, by the controller from the metadata server responsive to the query, the placement information for the first instance.
4. The method of claim 1, comprising:
 - determining, based on the account information, a first weight for the first instance provided via a first zone of the cloud environment;
 - determining, based on the account information, a second weight for a second instance provided via a second zone of the cloud environment different from the first zone; and

selecting the first instance responsive to the first weight greater than the second weight.

5. The method of claim 1, comprising:

identifying, based on the account information, that the first instance of a first service of the one or more services is provided via a first zone of the cloud environment;

identifying, based on the account information, that a second instance of a second service of the one or more services is provided via a second zone of the cloud environment, a distance between the controller and the second zone greater than a distance between the controller and the first zone; and

determining, by the controller responsive to determining the distance between the controller and the second zone is greater than the distance between the controller and the first zone, to cache data provided via the second instance of the second service in memory of the controller.

6. The method of claim 1, comprising:

identifying, by the controller, a plurality of devices intermediary to the client and the one or more servers;

determining, by the controller, a first weight for the first instance based on the account information and a first placement of a first intermediary device of the plurality of devices;

determining, by the controller, a second weight for the first instance based on the account information and a second placement of a second intermediary device of the plurality of devices; and

selecting, by the controller, the first intermediary device to direct network traffic of the instance from the client based on a comparison of the first weight and the second weight.

7. The method of claim 6, comprising:

determining the first weight is greater than the second weight based on a distance between the first placement and the one or more servers being less than a distance between the second placement and the one or more servers; and

selecting the first intermediary device responsive to the first weight being greater than the second weight.

8. The method of claim 1, comprising:

identifying, by the controller, a first virtual server executing on a device intermediary to the client and the one or more servers, the first virtual server associated with the instance;

identifying, by the controller, a second virtual server executing on the device intermediary to the client and the one or more servers, the second virtual server associated with a second instance of the one or more services;

determining, by the controller, a first weight for the instance based on the account information and an attribute of the first virtual server;

determining, by the controller, a second weight for the second instance based on the account information and an attribute of the second virtual server; and

selecting, by the controller, the first virtual server to direct network traffic of the instance from the client based on a comparison of the first weight and the second weight.

9. The method of claim 1, comprising:

identifying, by the controller, a first device intermediary to the client and the cloud environment, the first device configured to direct network traffic related to the plurality of instances;

identifying, by the controller, a second device intermediary to the client and the cloud environment, the second device configured to direct network traffic related to the plurality of instances;

determining, by the controller, a first weight based on account information for each of the plurality of instances and an attribute of the first device;

determining, by the controller, a second weight based on account information for each of the plurality of instances and an attribute of the second device, the second weight less than the first weight; and

selecting, by the controller, the first device to direct network traffic related to the plurality of instances.

10. The method of claim 1, comprising:

identifying, by the controller, a first plurality of virtual servers executed by a first device intermediary to the client and the cloud environment, the first plurality of virtual servers configured to direct network traffic related to one or more of the plurality of instances;

identifying, by the controller, a second plurality of virtual servers executed by a second device intermediary to the client and the cloud environment, the second plurality of

virtual servers configured to direct network traffic related to one or more of the plurality of instances; determining, by the controller, a first weight based on account information for each of the one or more plurality of instances and an attribute for each of the first plurality of virtual servers;

determining, by the controller, a second weight based on account information for each of the one or more plurality of instances and an attribute for each of the second plurality of virtual servers, the second weight less than the first weight; and

selecting, by the controller, the first device executing the first plurality of virtual servers to direct network traffic related to the one or more of the plurality of instances.

11. The method of claim 1, comprising:

periodically receiving updated account information; and

periodically selecting, based on the load balancing function and an updated weight for each of the instances based on the updated account information, a second instance of the plurality of instances to direct network traffic from one or more clients.

12. A system to manage cloud-based services via a network, comprising:

a controller intermediary to a client and a cloud environment configured to:

receive account information related to each of a plurality of instances of one or more services provided by one or more servers of the cloud environment;

determine a weight for each of the instances based on the account information, the weight indicating a performance of the instance of the service provided via the cloud environment; and

select, based on a load balancing function and the weight for each of the instances, an instance of the plurality of instances to direct network traffic from a client.

13. The system of claim 1, wherein the controller is further configured to:

generate, using a cloud application programming interface, one or more queries to retrieve the account information for each of the plurality of instances, the one or more queries configured to retrieve information for at least one of a placement, resource allocation, type of reservation, or type of connectivity.

14. The system of claim 1, wherein the controller is further configured to:

generate, using a cloud application programming interface, a query to retrieve placement information for a first instance of the plurality of instances;

transmit the query to a metadata server maintained by a provider of the cloud environment, the metadata server including a database storing placement information for the first instance; and

receive, from the metadata server responsive to the query, the placement information for the first instance.

15. The system of claim 1, wherein the controller is further configured to:

determine, based on the account information, a first weight for the first instance provided via a first zone of the cloud environment;

determine, based on the account information, a second weight for a second instance provided via a second zone of the cloud environment different from the first zone; and

select the first instance responsive to the first weight greater than the second weight.

16. The system of claim 1, wherein the controller is further configured to:

identify, based on the account information, that the first instance of a first service of the one or more services is provided via a first zone of the cloud environment;

identify, based on the account information, that a second instance of a second service of the one or more services is provided via a second zone of the cloud environment, a distance between the controller and the second zone greater than a distance between the controller and the first zone; and

determine, responsive to determining the distance between the controller and the second zone is greater than the distance between the controller and the first zone, to cache data provided via the second instance of the second service in memory of the controller.

17. The system of claim 1, wherein the controller is further configured to:

identify a plurality of devices intermediary to the client and the one or more servers;

determine a first weight for the first instance based on the account information and a first placement of a first intermediary device of the plurality of devices;

determine a second weight for the first instance based on the account information and a second placement of a second intermediary device of the plurality of devices; and

select the first intermediary device to direct network traffic of the instance from the client based on a comparison of the first weight and the second weight.

18. The system of claim 17, wherein the controller is further configured to:

determine the first weight is greater than the second weight based on a distance between the first placement and the one or more servers being less than a distance between the second placement and the one or more servers; and

select the first intermediary device responsive to the first weight being greater than the second weight.

19. The system of claim 1, wherein the controller is further configured to:

identify a first virtual server executing on a device intermediary to the client and the one or more servers, the first virtual server associated with the instance;

identify a second virtual server executing on the device intermediary to the client and the one or more servers, the second virtual server associated with a second instance of the one or more services;

determine a first weight for the instance based on the account information and an attribute of the first virtual server;

determine a second weight for the second instance based on the account information and an attribute of the second virtual server; and

select the first virtual server to direct network traffic of the instance from the client based on a comparison of the first weight and the second weight.

20. The system of claim 1, wherein the controller is further configured to:

identify a first device intermediary to the client and the cloud environment, the first device configured to direct network traffic related to the plurality of instances;

identify a second device intermediary to the client and the cloud environment, the second device configured to direct network traffic related to the plurality of instances;

determine a first weight based on account information for each of the plurality of instances and an attribute of the first device;

determine a second weight based on account information for each of the plurality of instances and an attribute of the second device, the second weight less than the first weight; and

select the first device to direct network traffic related to the plurality of instances.

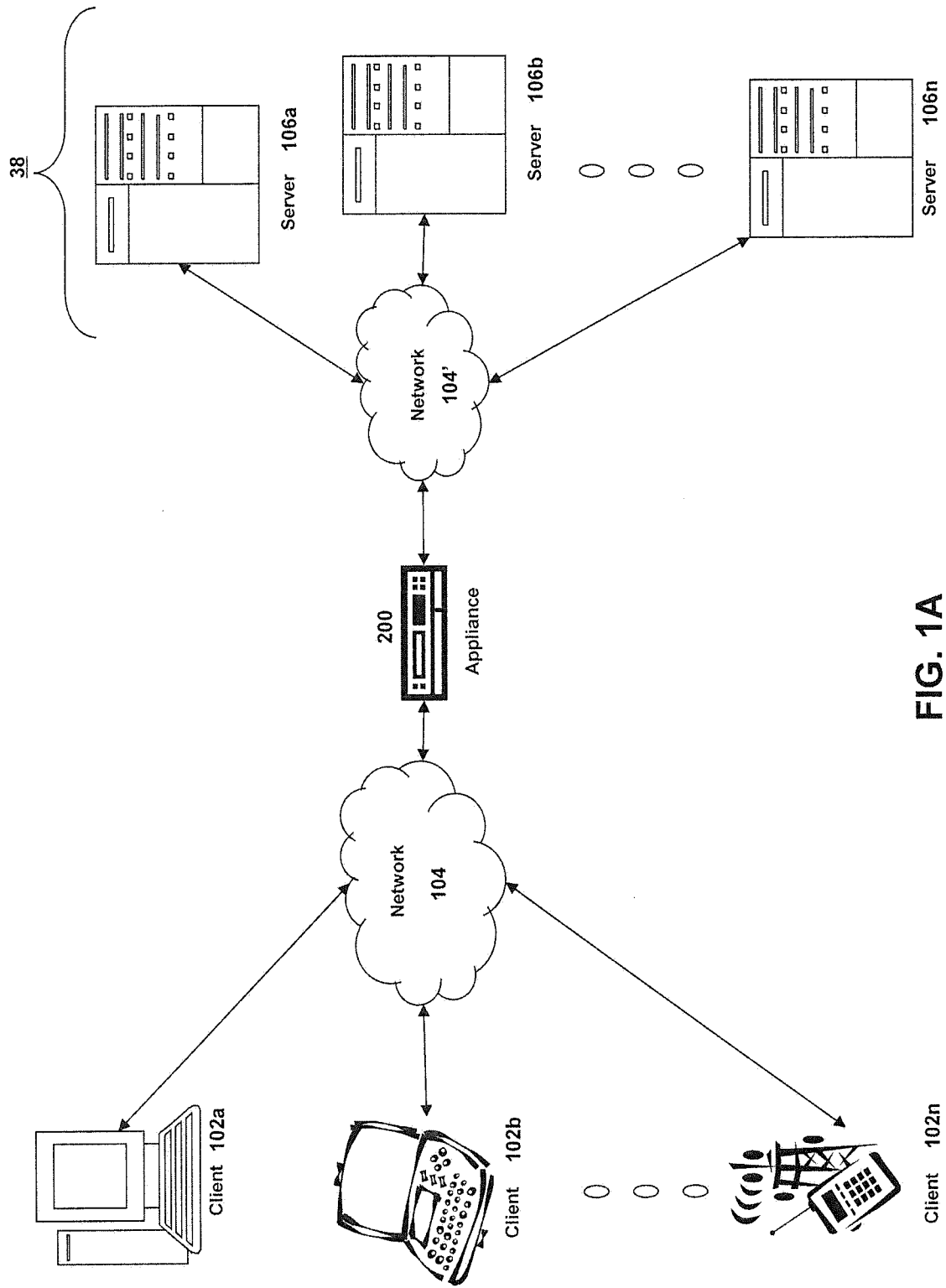


FIG. 1A

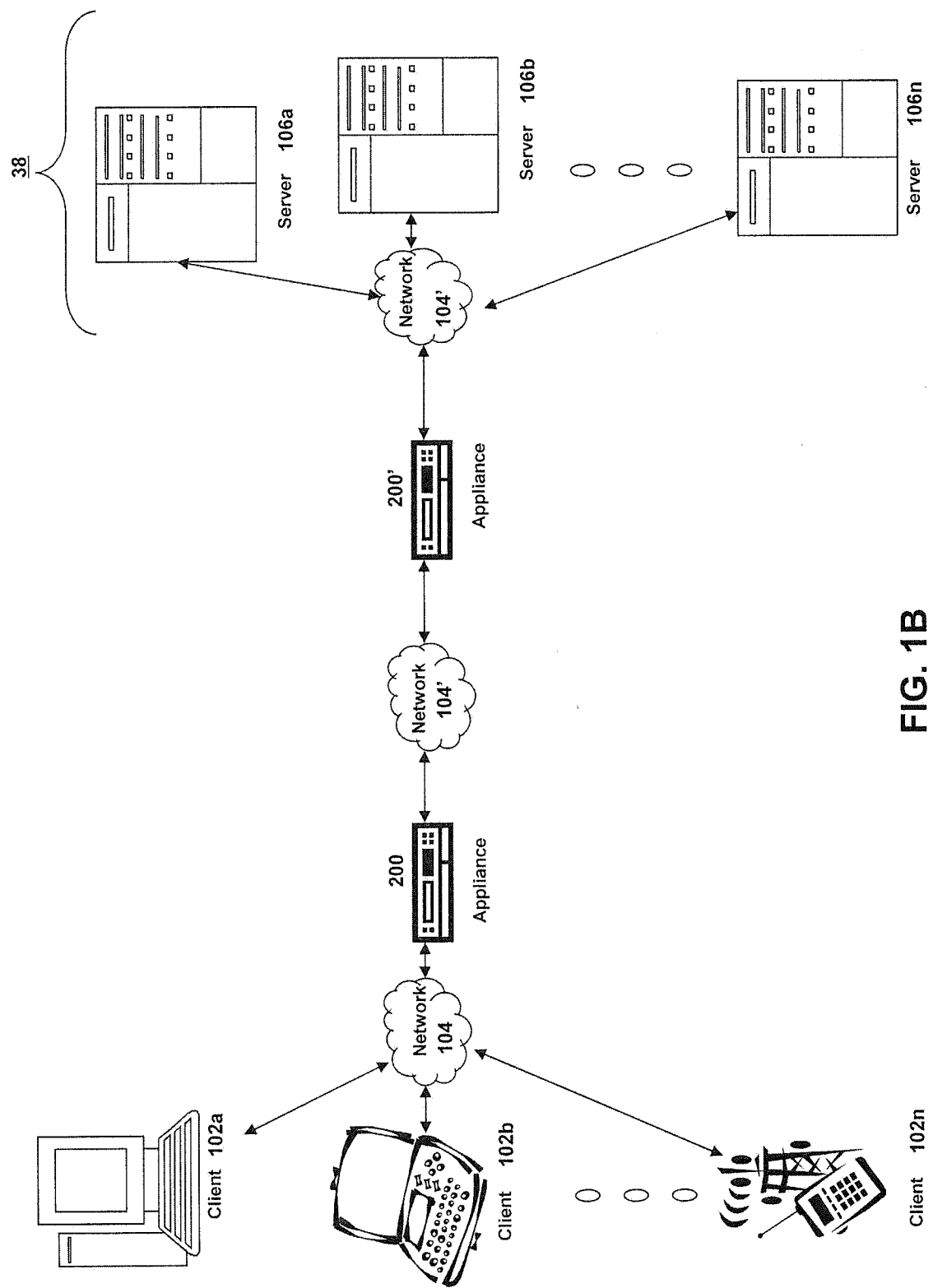


FIG. 1B

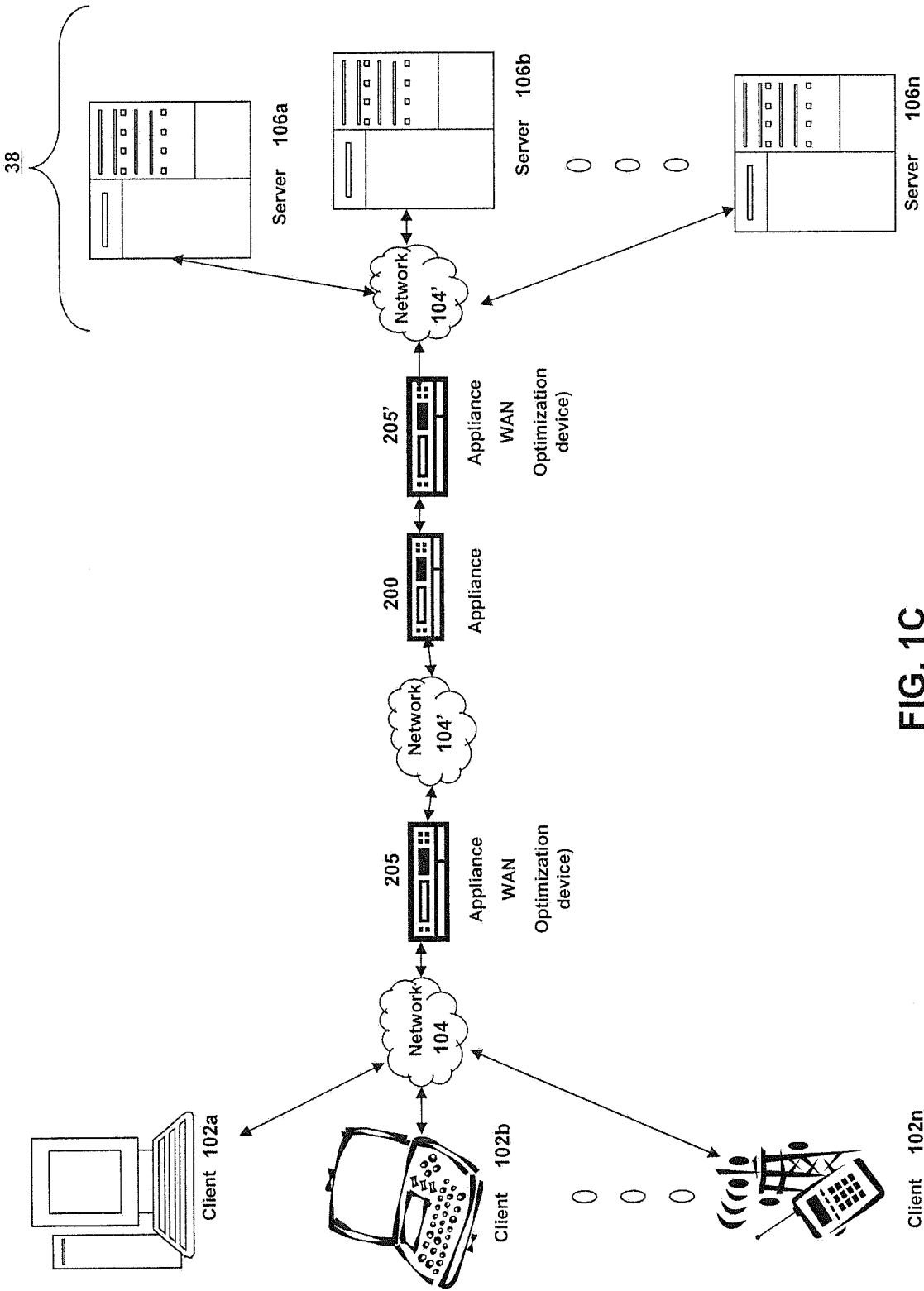


FIG. 1C

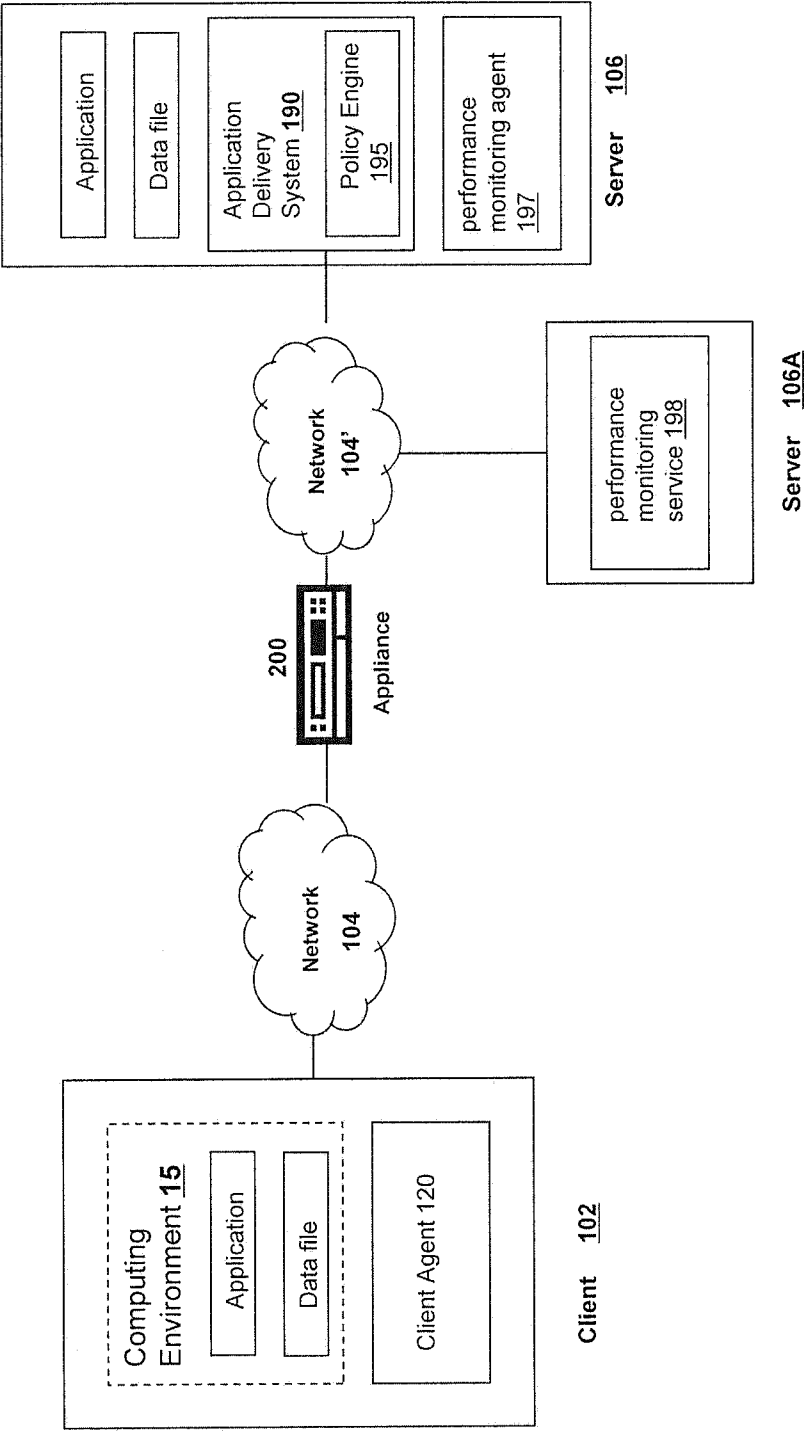
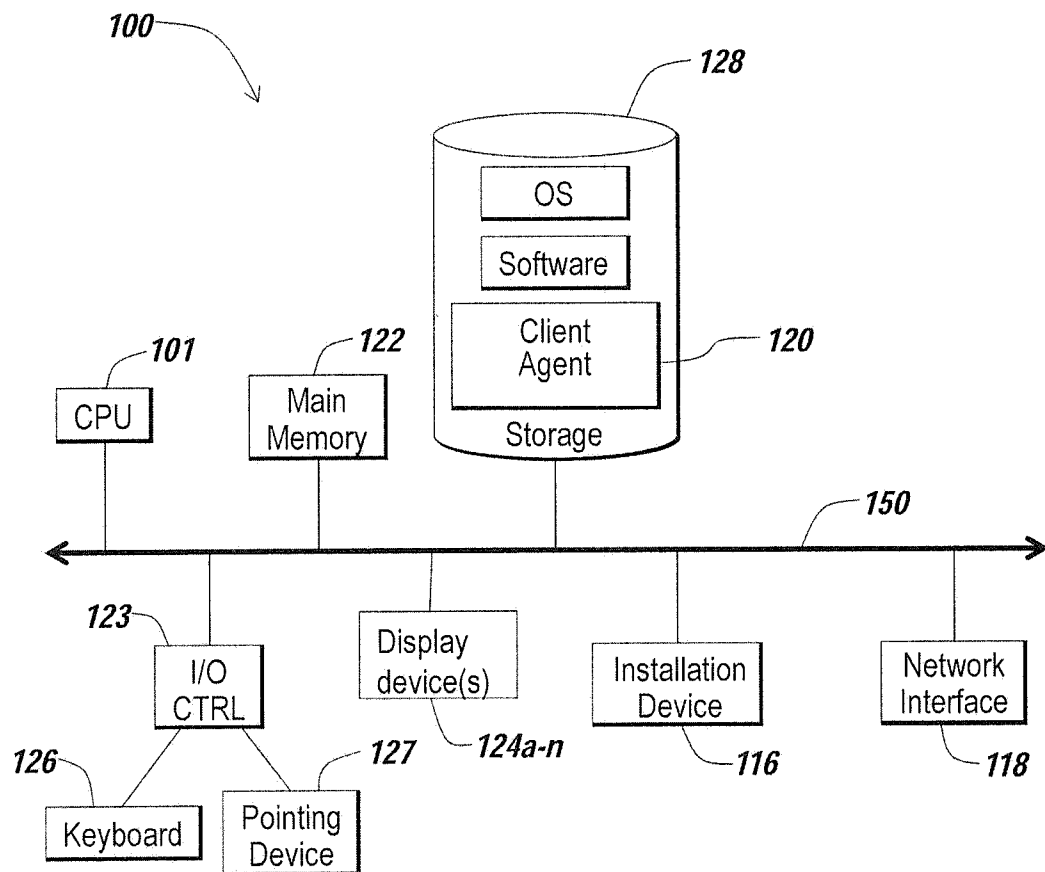
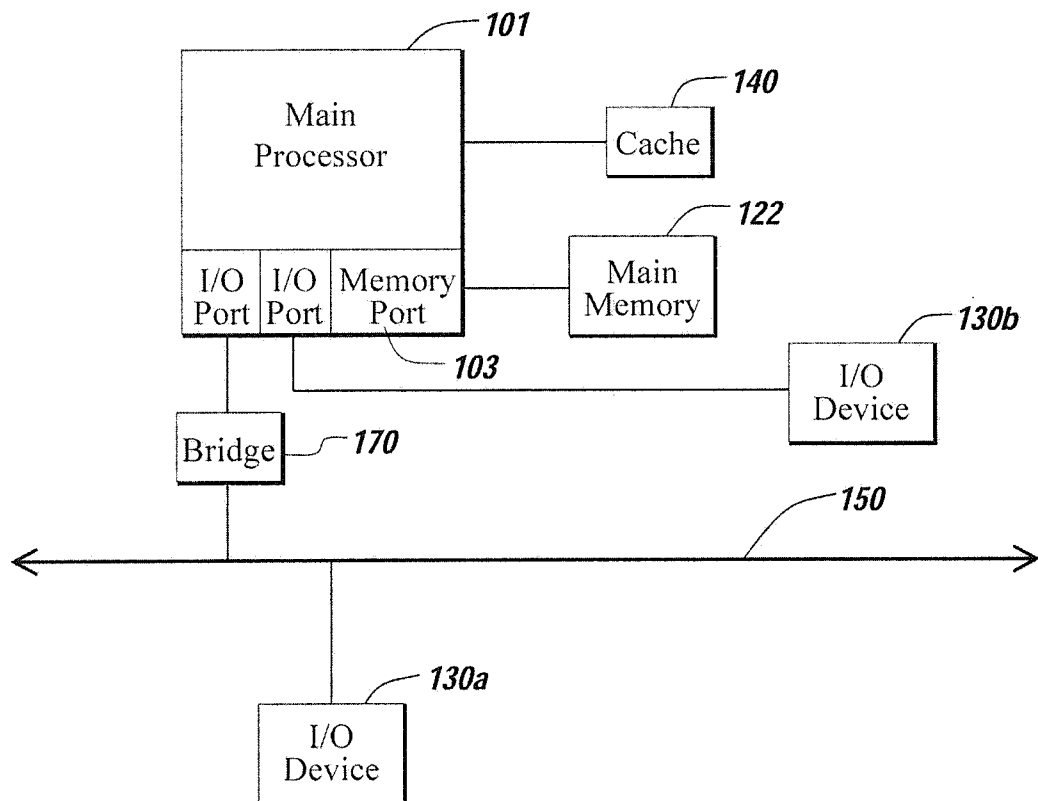
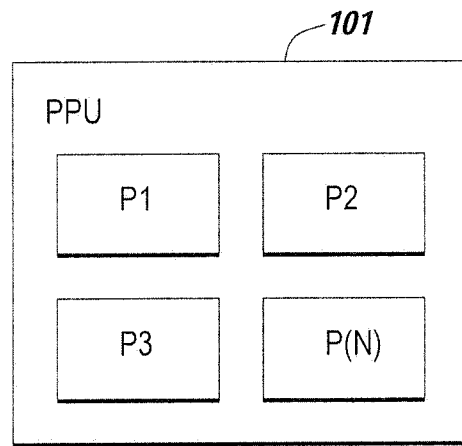
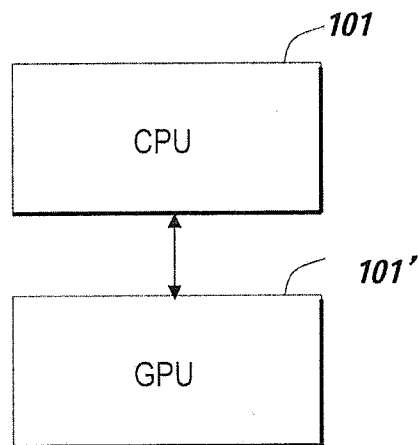


FIG. 1D

*Fig. 1E*

*Fig. 1F*

*Fig. 1G**Fig. 1H*

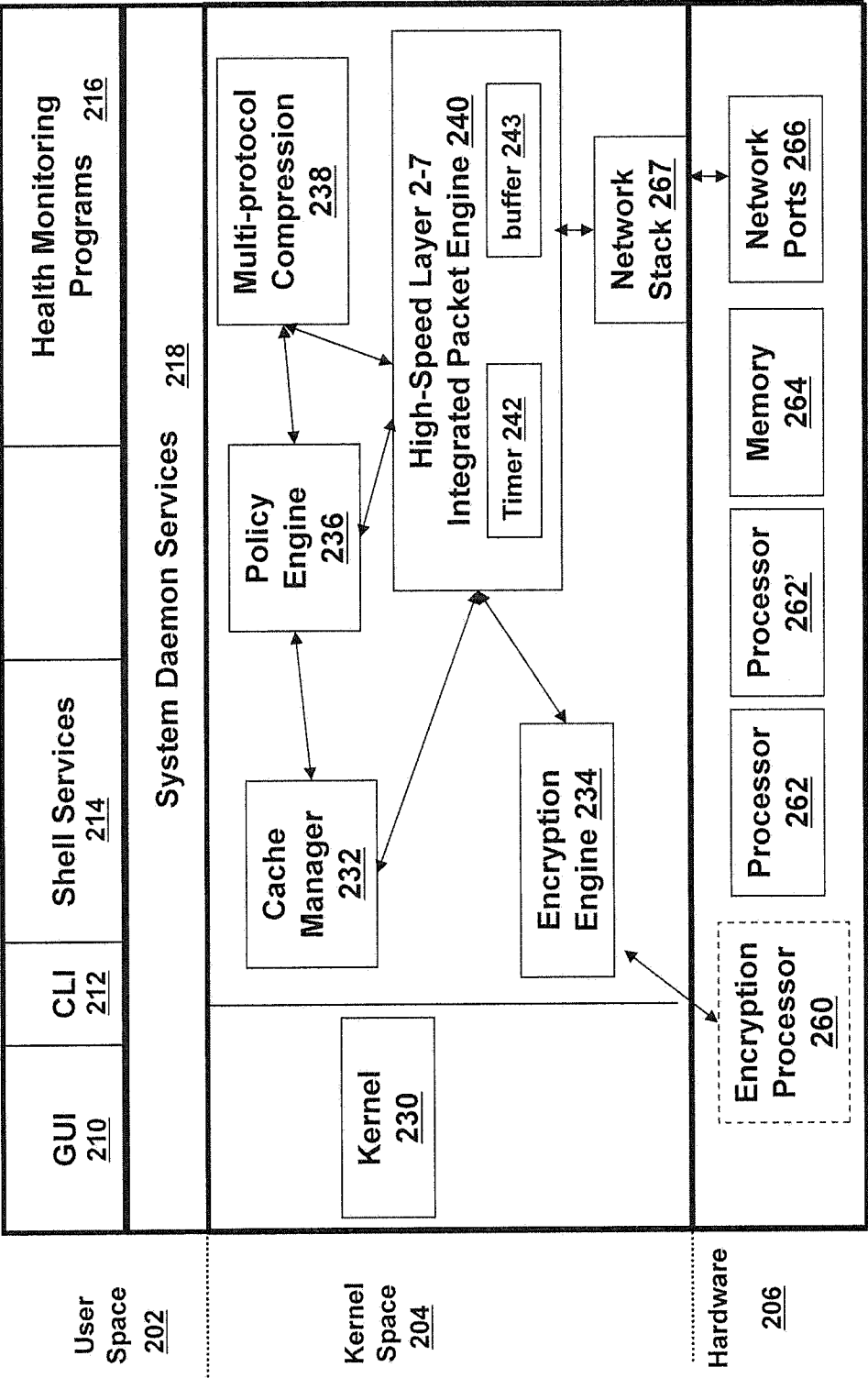


FIG. 2A

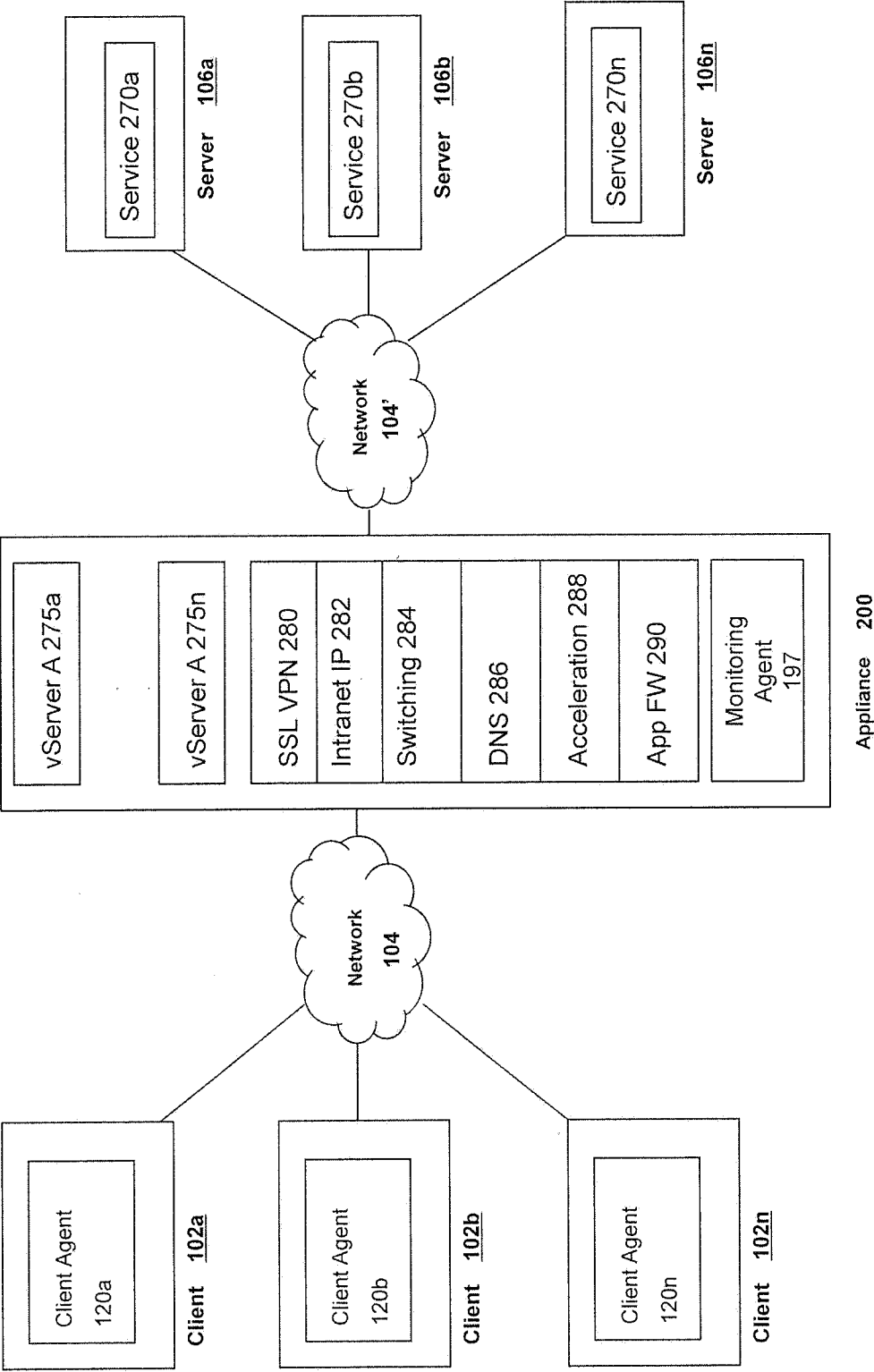
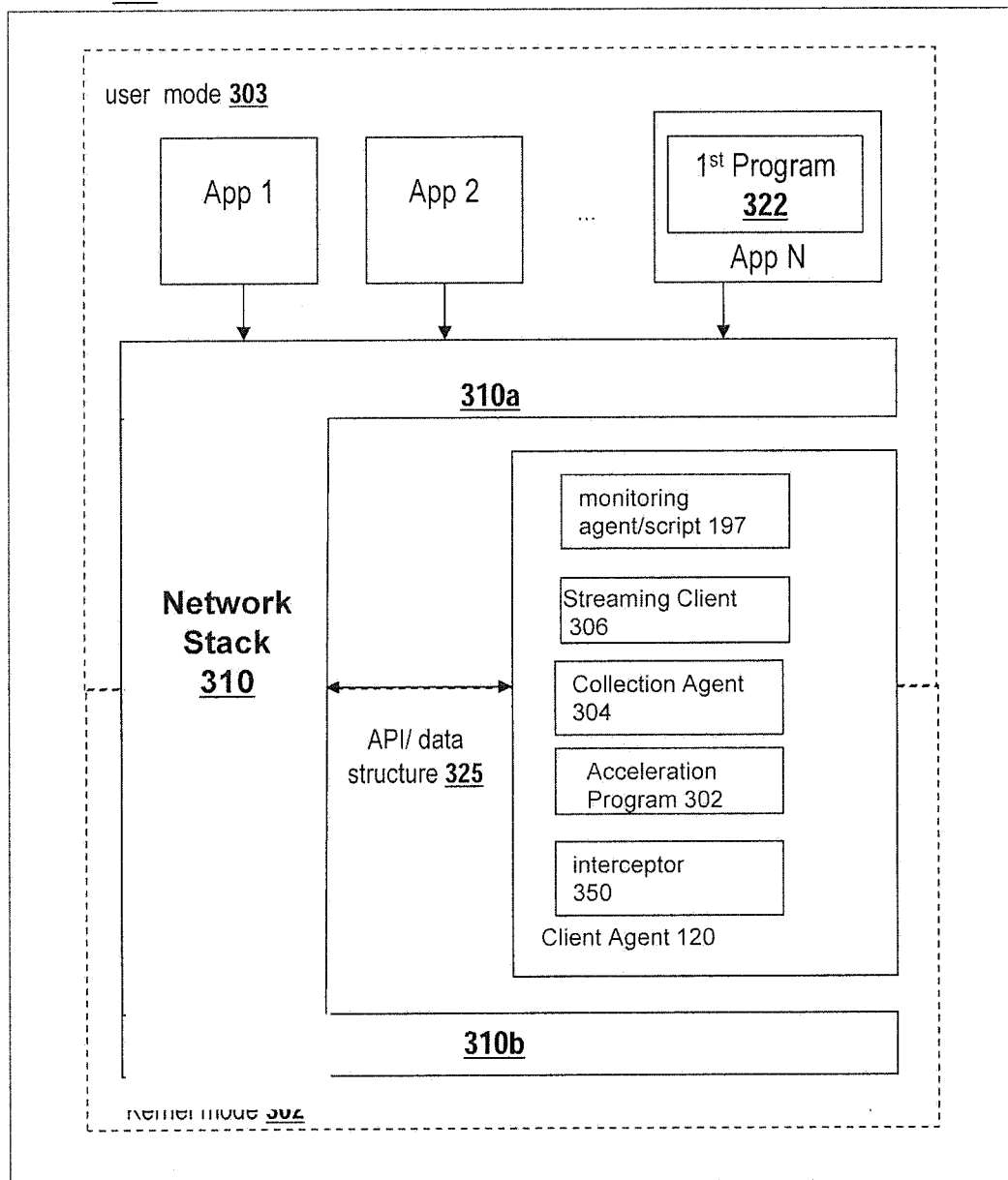
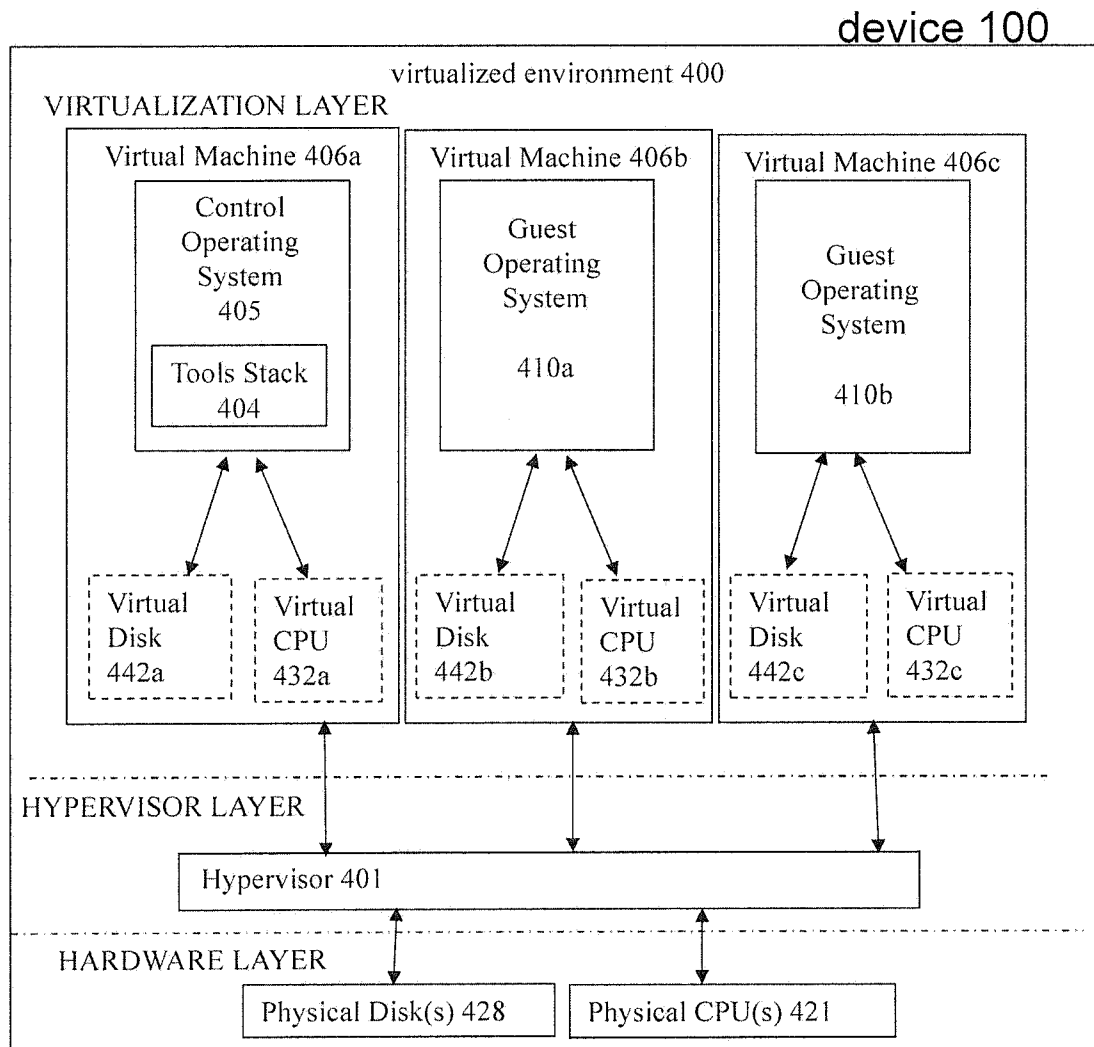
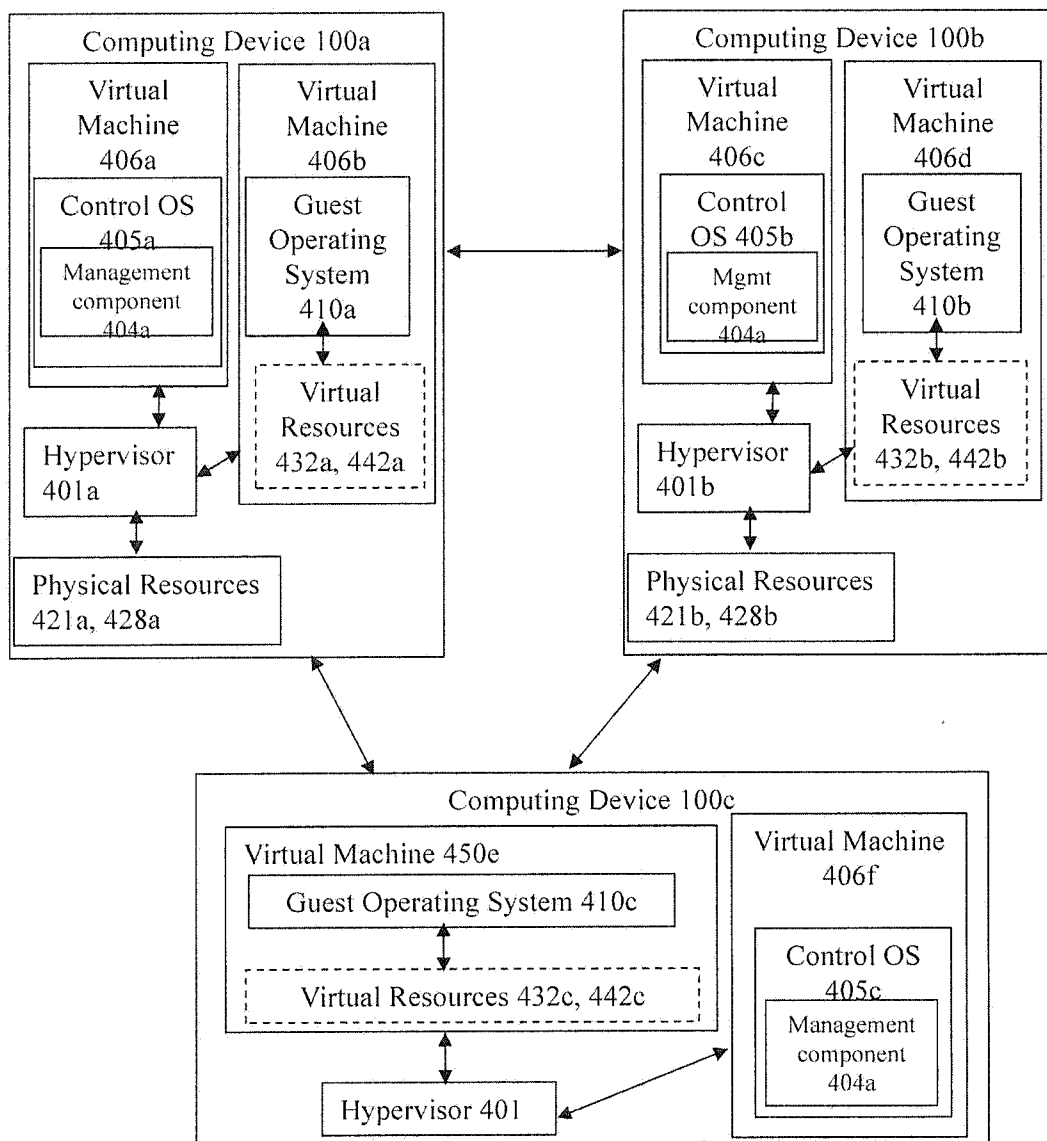
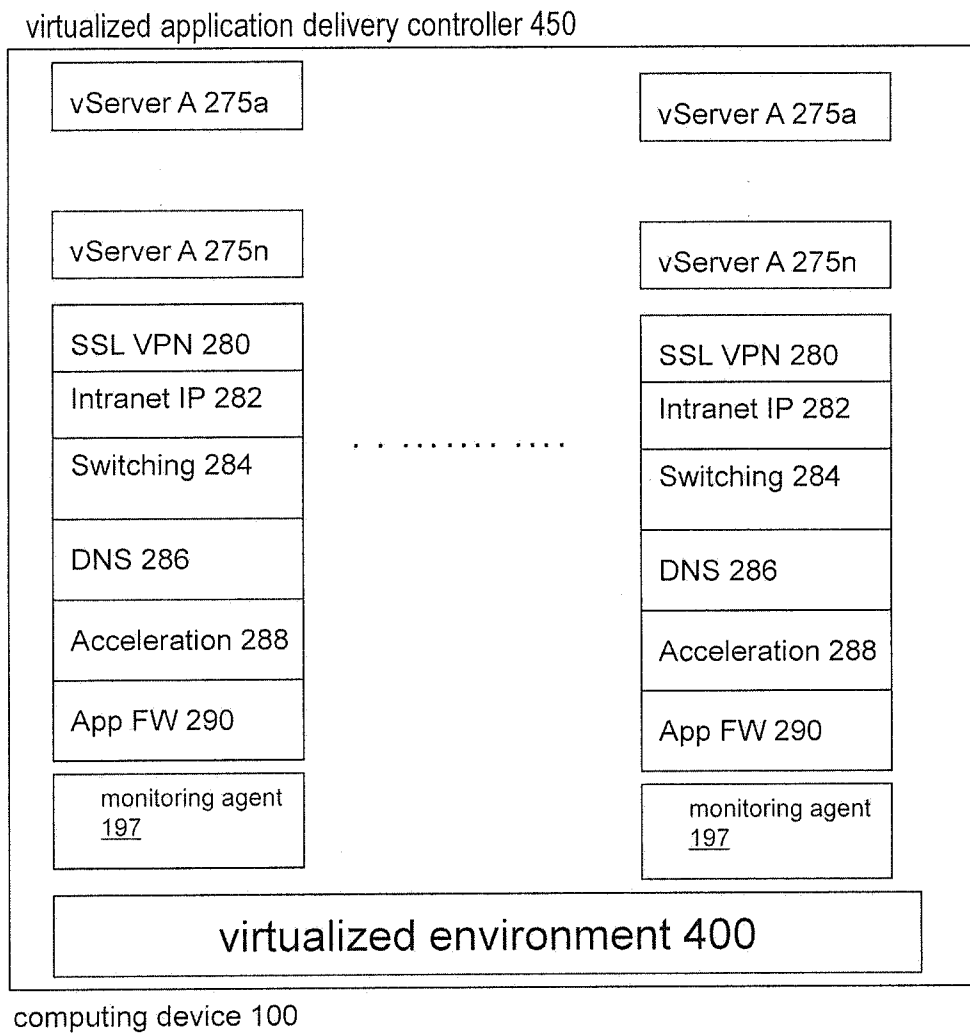


FIG. 2B

Client **102***Fig. 3*

*Fig. 4A*

*Fig. 4B*

*Fig. 4C*

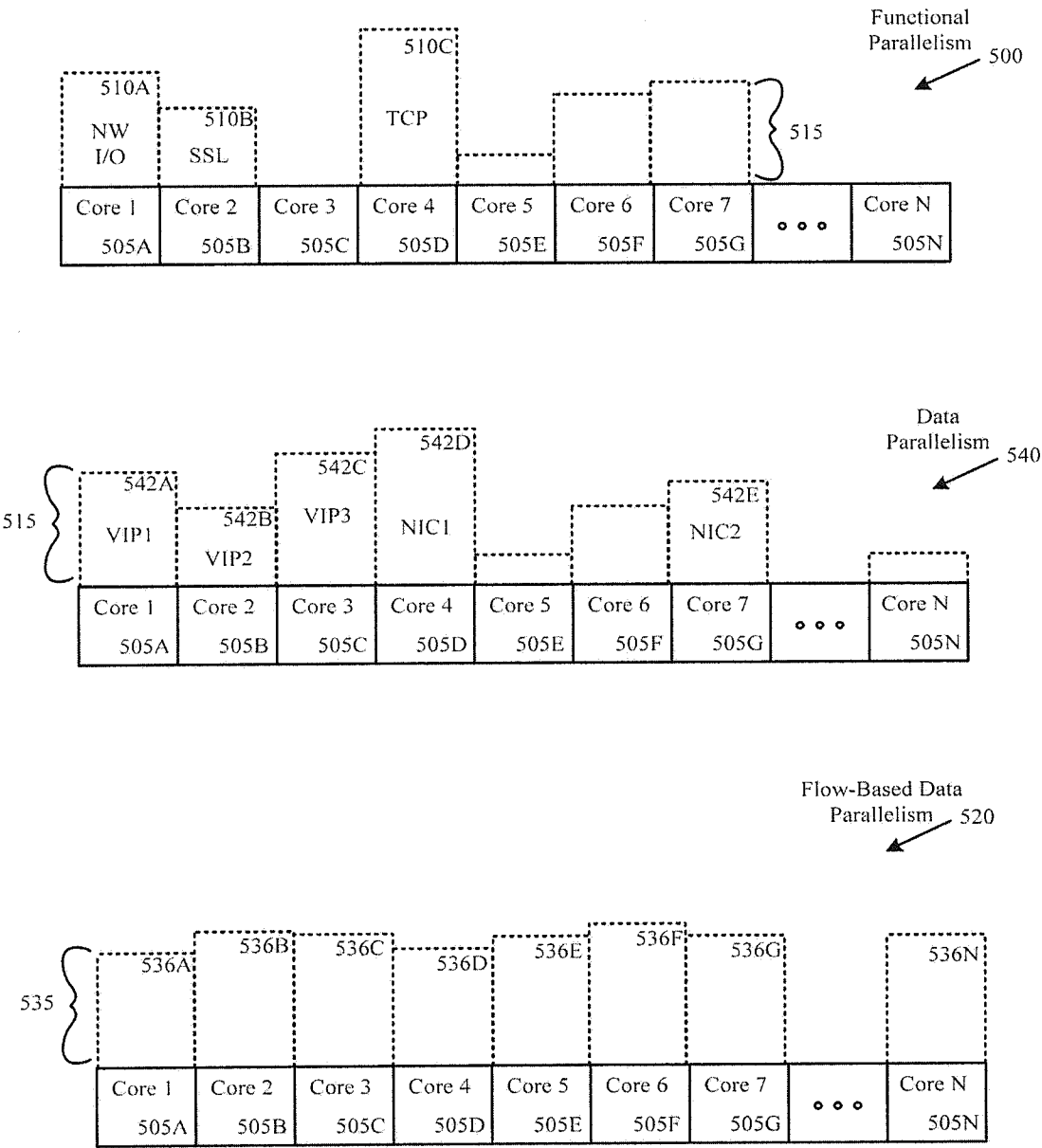


Fig. 5A

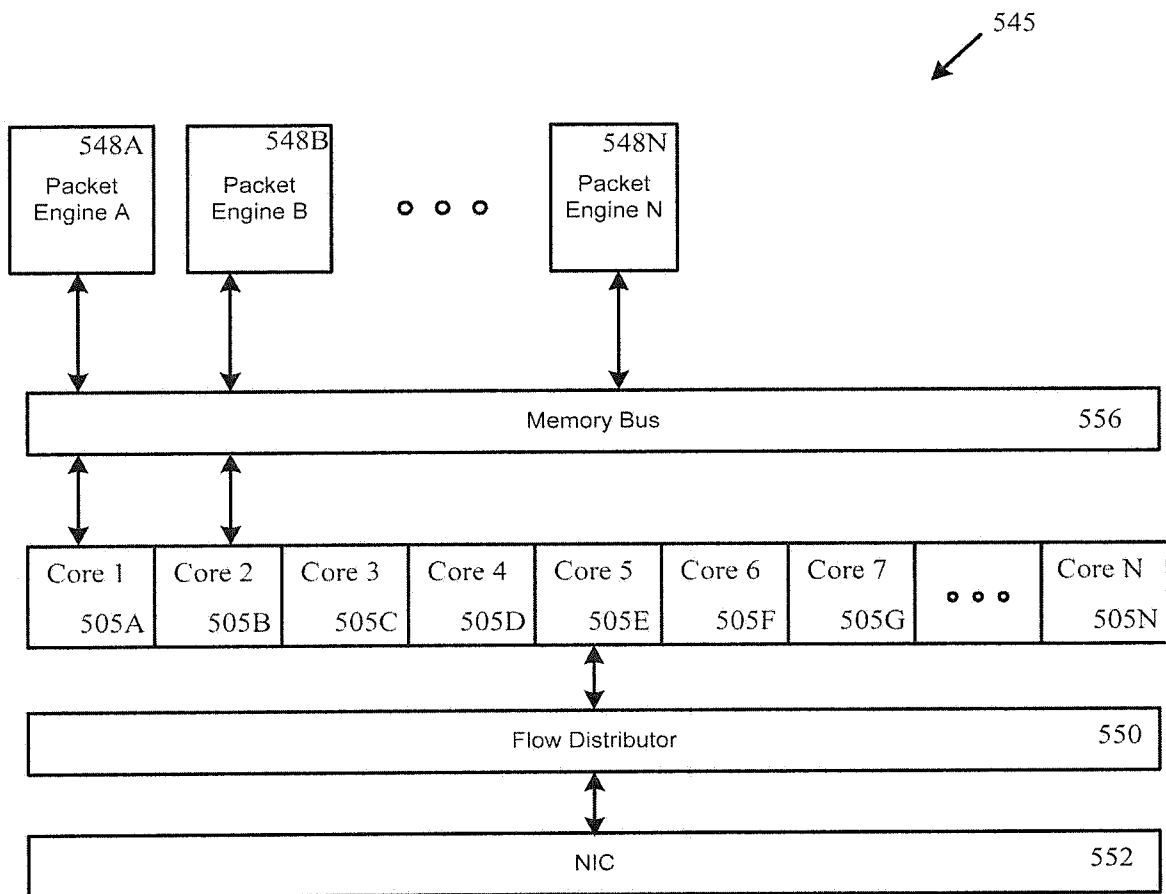


Fig. 5B

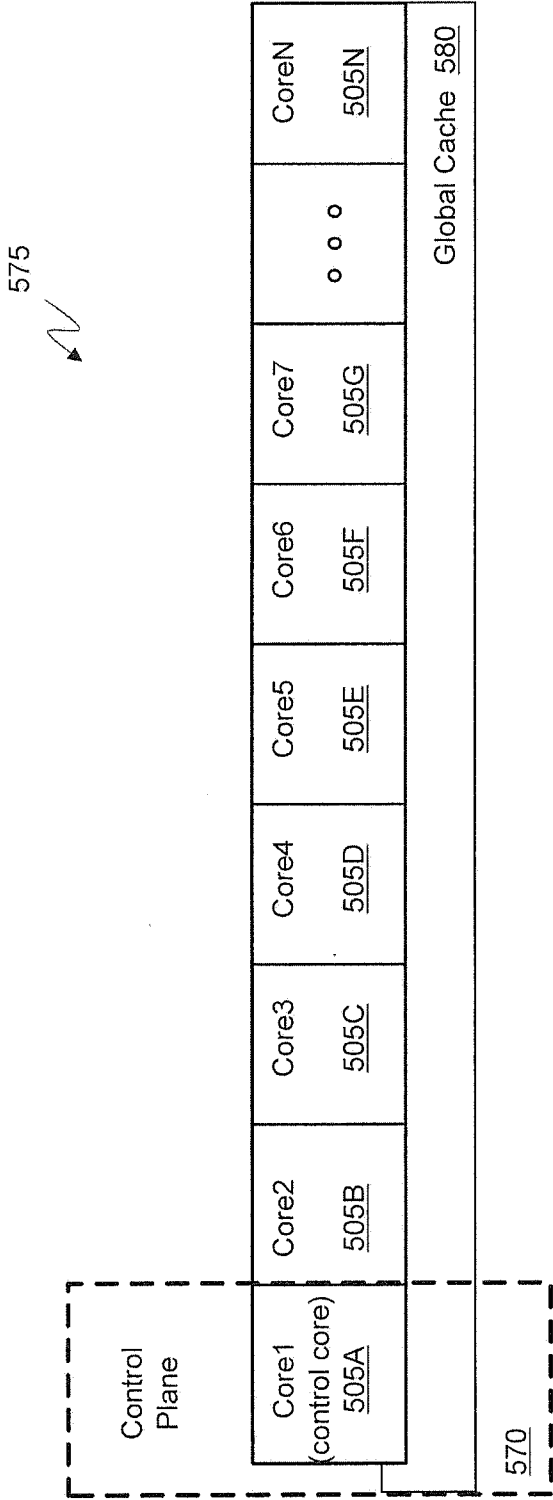


FIG. 5C

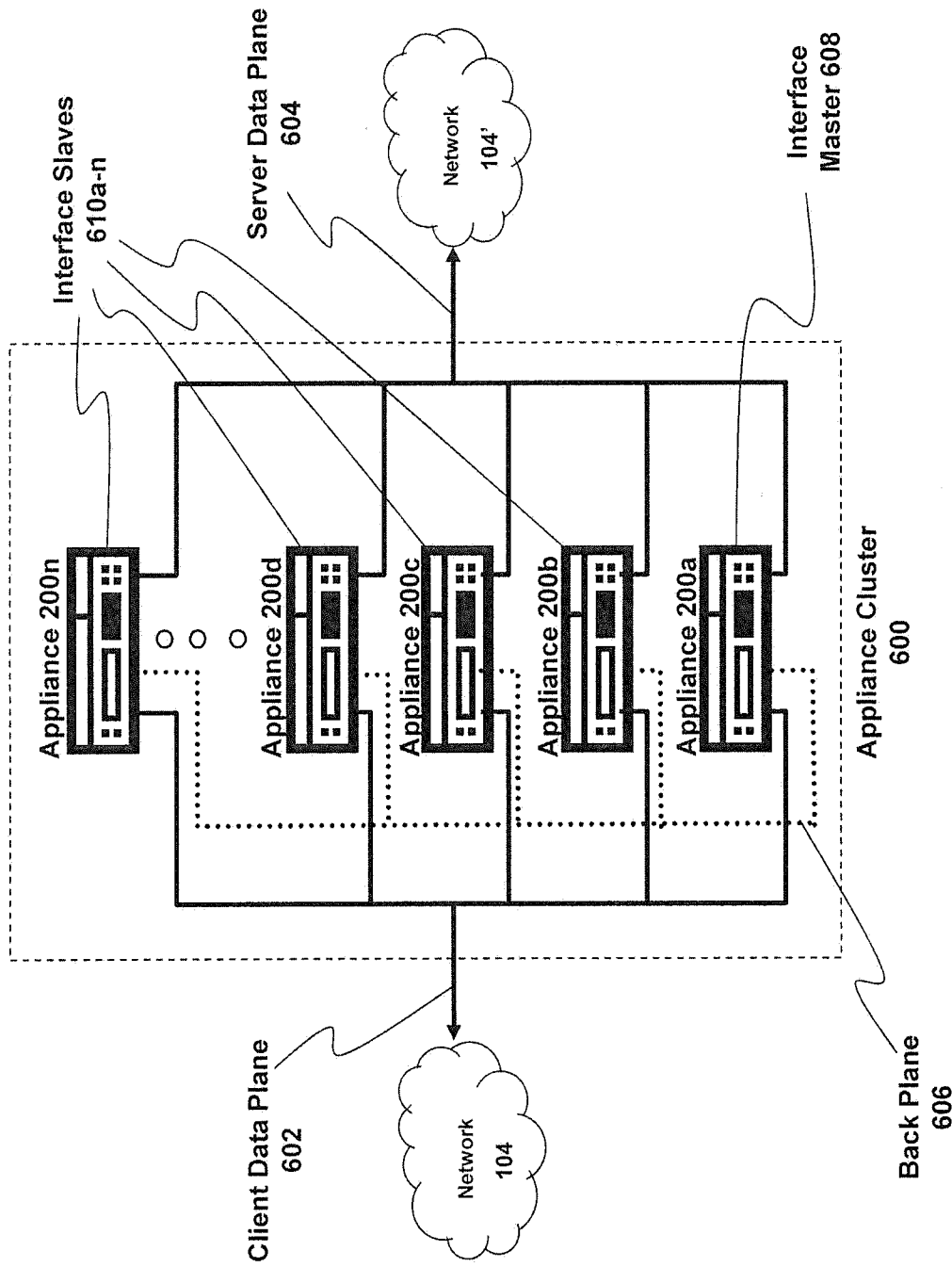


FIG. 6

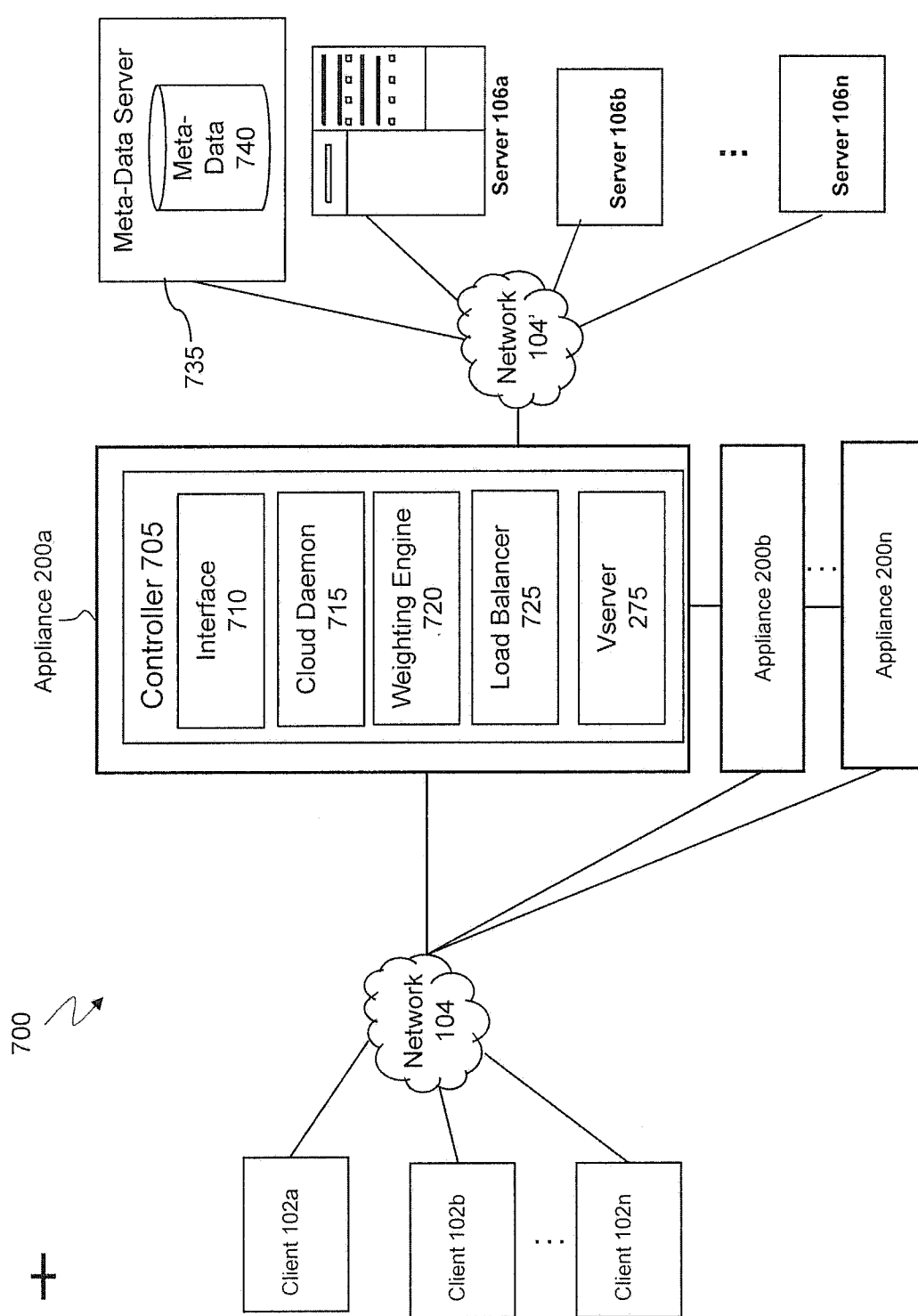


FIG. 7A

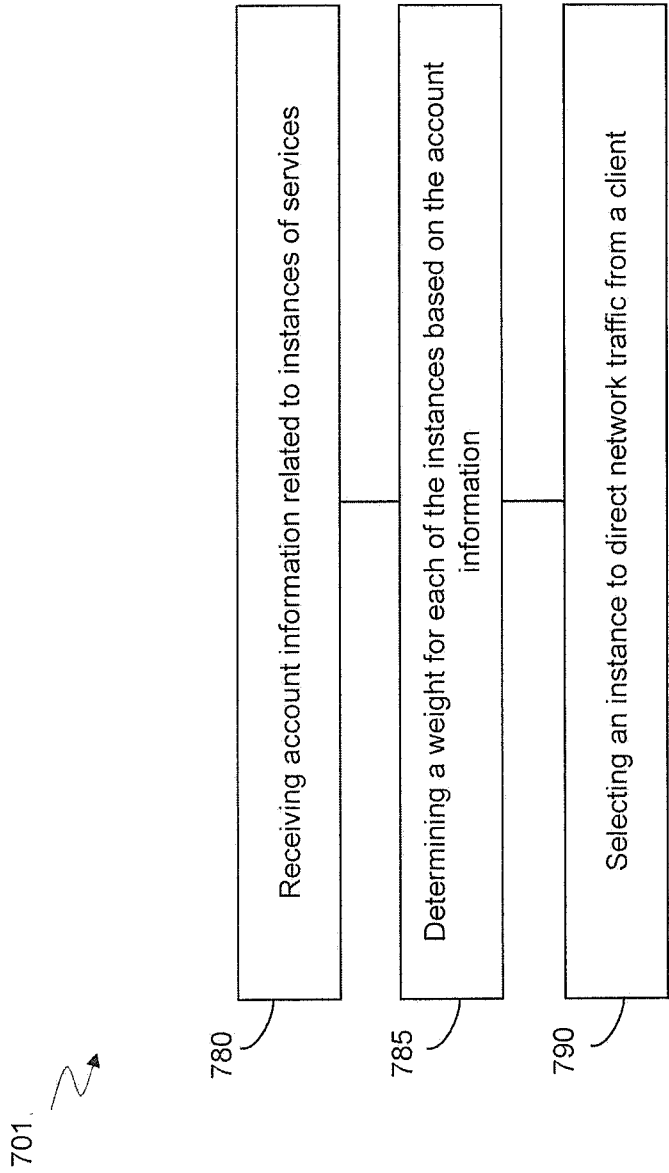
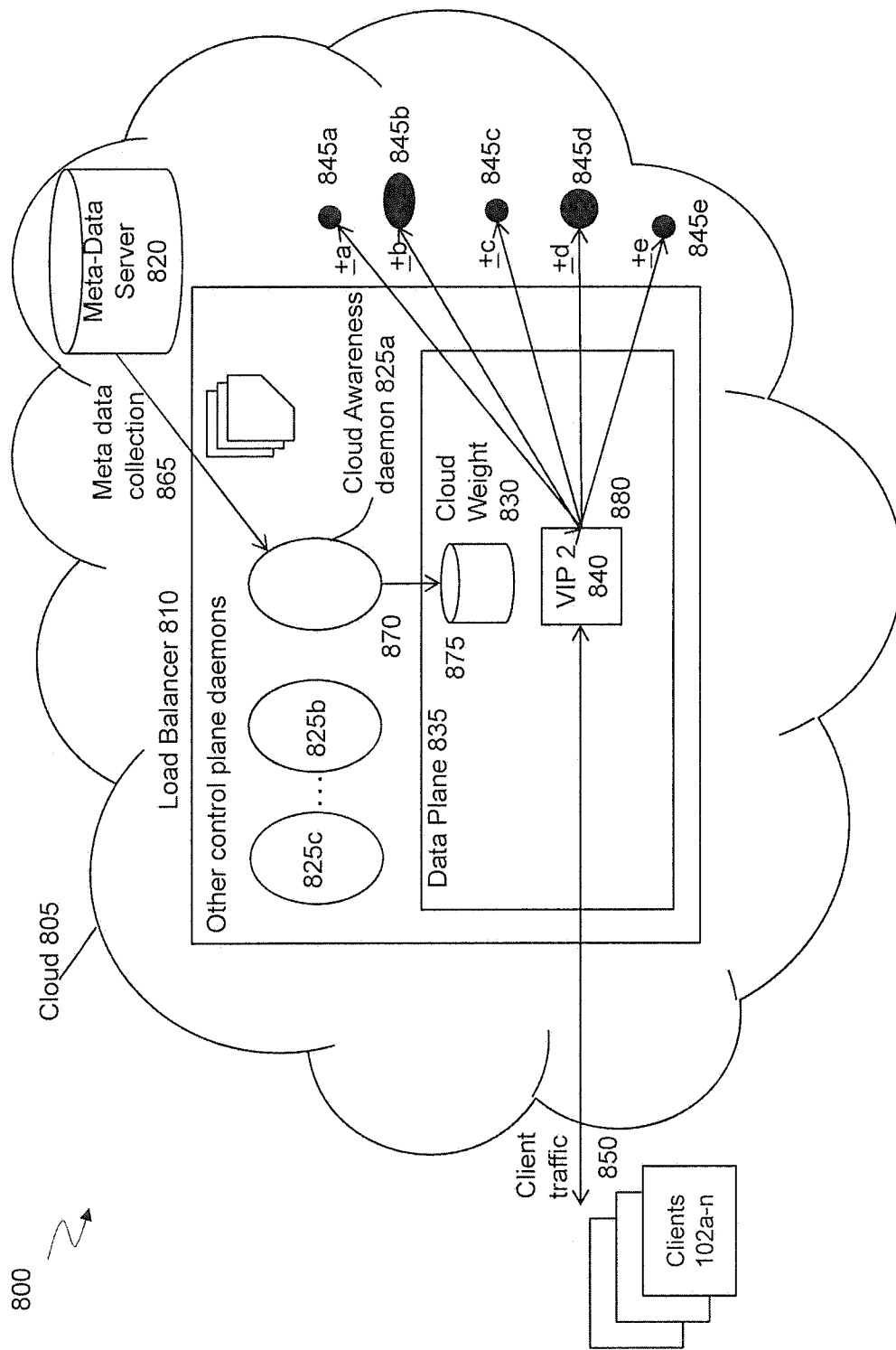


FIG. 7B



8
G.
L

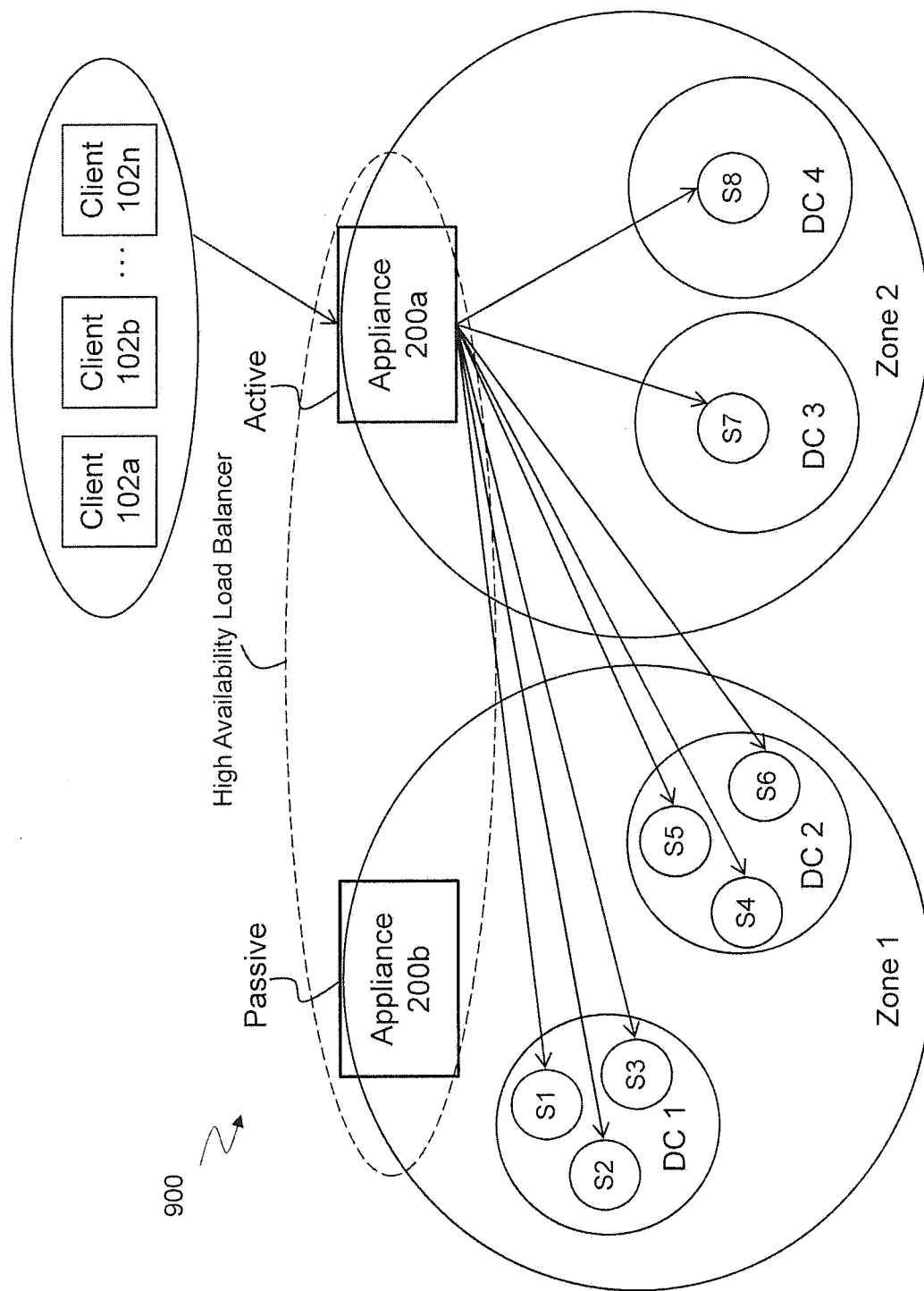


FIG. 9A

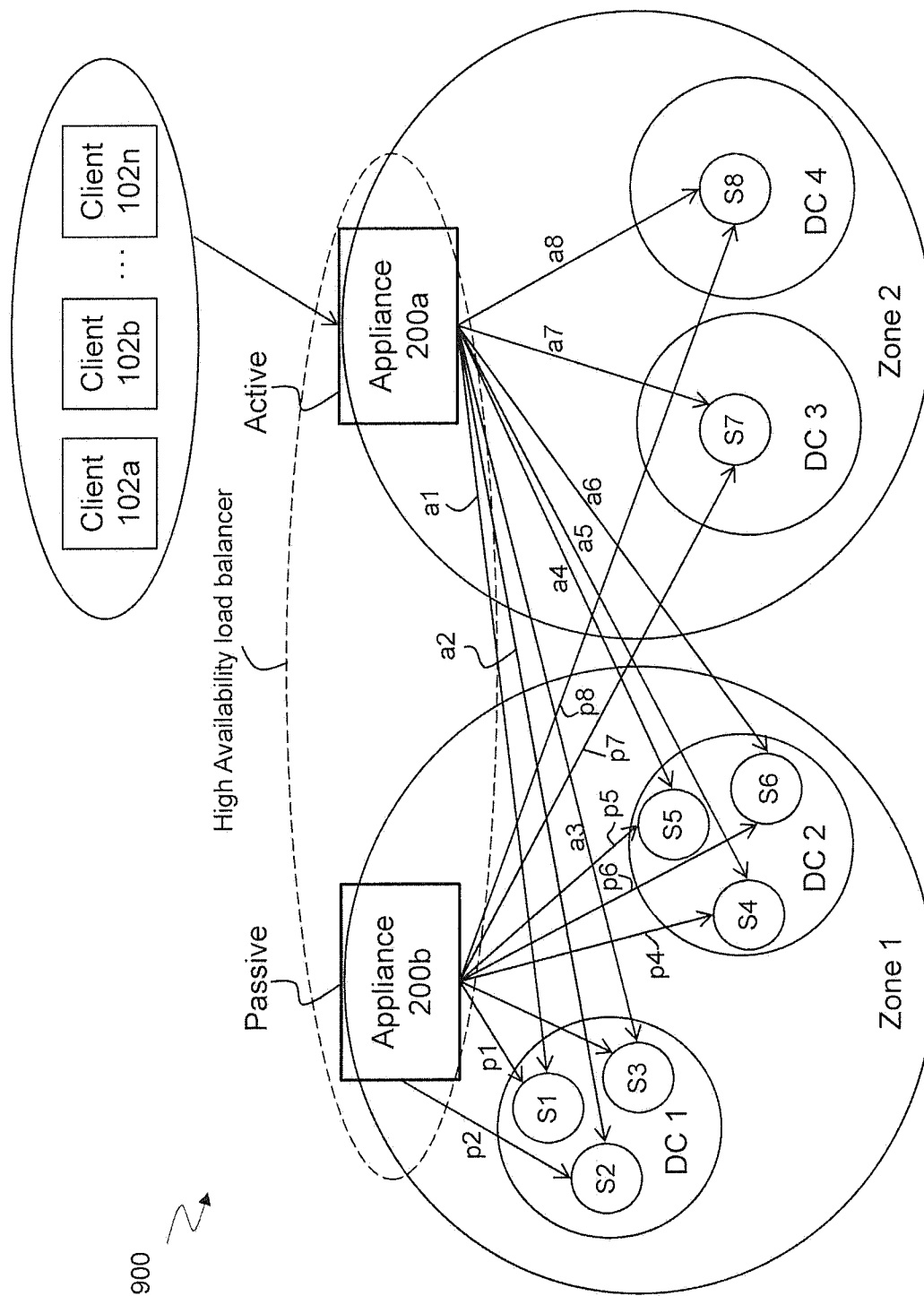


FIG. 9B

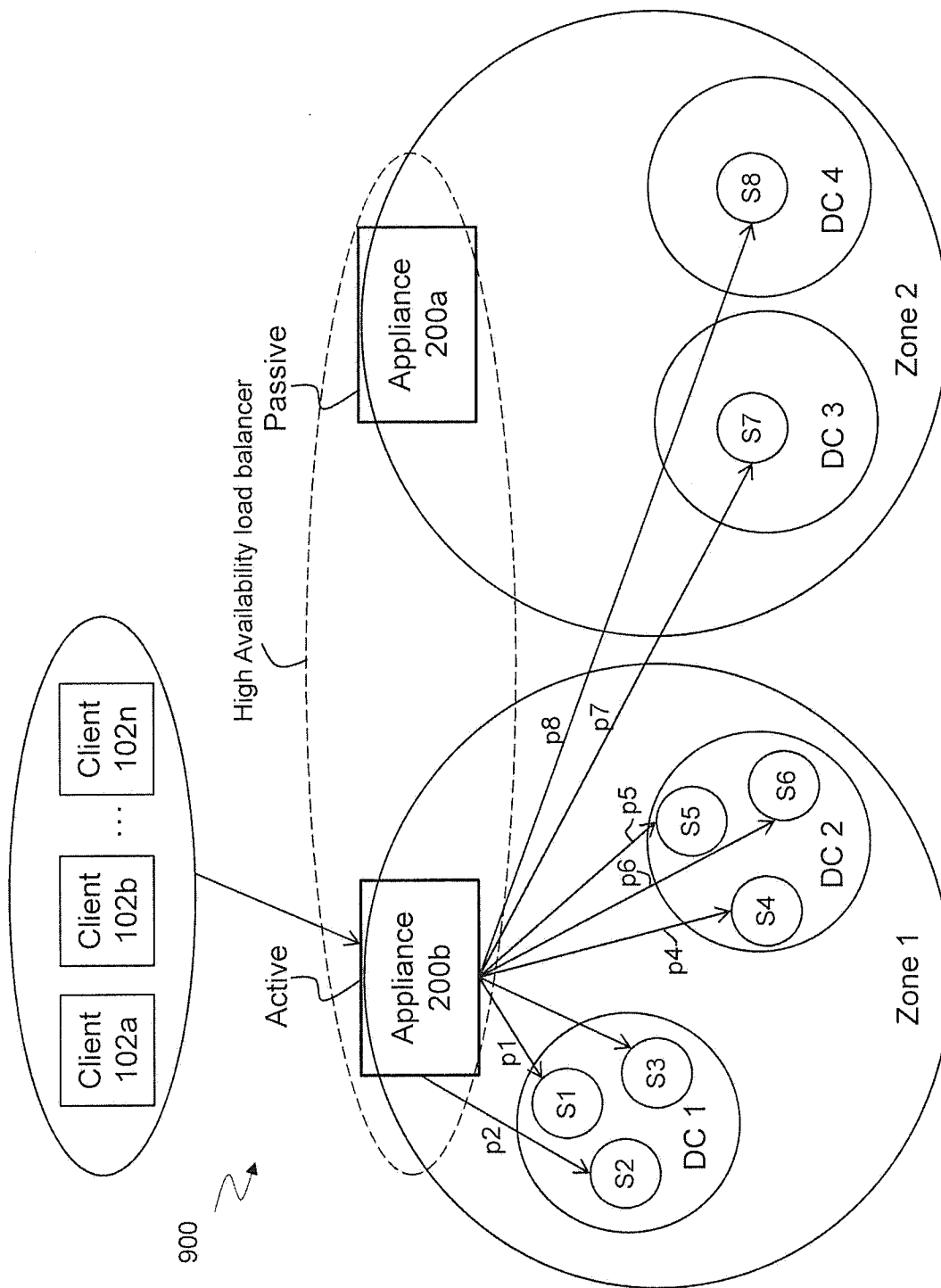


FIG. 9C

INTERNATIONAL SEARCH REPORT

International application No
PCT/US2017/014056

A. CLASSIFICATION OF SUBJECT MATTER
INV. G06F9/50 H04L29/08
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)
G06F 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 practicable, search terms used)

EPO-Internal, 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 2014/101226 A1 (KHANDEKAR MILIND M [US] ET AL) 10 April 2014 (2014-04-10) paragraphs [0014], [0015], [0035], [0041]; figure 2 -----	1,2,12, 13



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 application or patent 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

Date of the actual completion of the international search

22 March 2017

Date of mailing of the international search report

24/05/2017

Name and mailing address of the ISA/

European Patent Office, P.B. 5818 Patentlaan 2
NL - 2280 HV Rijswijk
Tel. (+31-70) 340-2040,
Fax: (+31-70) 340-3016

Authorized officer

Mühlenbrock, Martin

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2017/014056

Box No. II Observations where certain claims were found unsearchable (Continuation of item 2 of first sheet)

This international search report has not been established in respect of certain claims under Article 17(2)(a) for the following reasons:

1. ☐ Claims Nos.:
because they relate to subject matter not required to be searched by this Authority, namely:
2. ☐ Claims Nos.:
because they relate to parts of the international application that do not comply with the prescribed requirements to such an extent that no meaningful international search can be carried out, specifically:
3. ☐ Claims Nos.:
because they are dependent claims and are not drafted in accordance with the second and third sentences of Rule 6.4(a).

Box No. III Observations where unity of invention is lacking (Continuation of item 3 of first sheet)

This International Searching Authority found multiple inventions in this international application, as follows:

see additional sheet

1. ☐ As all required additional search fees were timely paid by the applicant, this international search report covers all searchable claims.
2. ☐ As all searchable claims could be searched without effort justifying an additional fees, this Authority did not invite payment of additional fees.
3. ☐ As only some of the required additional search fees were timely paid by the applicant, this international search report covers only those claims for which fees were paid, specifically claims Nos.:
4. ☒ No required additional search fees were timely paid by the applicant. Consequently, this international search report is restricted to the invention first mentioned in the claims; it is covered by claims Nos.:

1, 2, 12, 13

Remark on Protest

- ☐ The additional search fees were accompanied by the applicant's protest and, where applicable, the payment of a protest fee.
- ☐ The additional search fees were accompanied by the applicant's protest but the applicable protest fee was not paid within the time limit specified in the invitation.
- ☐ No protest accompanied the payment of additional search fees.

FURTHER INFORMATION CONTINUED FROM PCT/ISA/ 210

This International Searching Authority found multiple (groups of) inventions in this international application, as follows:

1. claims: 1, 2, 12, 13

Generating queries to retrieve account information for each of a plurality of instances.

2. claims: 3, 14

Querying a metadata server including a database to retrieve placement information.

3. claims: 4, 15

Determining and comparing weights for instances provided via zones of the cloud environment.

4. claims: 5, 16

Determining to cache data in memory of a controller based on distances to the controller.

5. claims: 6, 7, 9, 17, 18, 20

Identifying and selecting devices intermediary to a client and servers to direct network traffic.

6. claims: 8, 10, 19

Identifying and selecting virtual servers executing on intermediary devices associated with instances.

7. claim: 11

Periodically updating account information and selecting a second instance based on an updated weight for each of the instances.

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No

PCT/US2017/014056

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2014101226 A1	10-04-2014	US 2014101226 A1	10-04-2014
		WO 2014058566 A2	17-04-2014
