US 20140304810A1

(19) United States
(12) Patent Application Publication
Khanal et al.
(10) Pub. No.: US 2014/0304810 A1
(43) Pub. Date: Oct. 9, 2014

(54) SYSTEMS AND METHODS FOR PROTECTING CLUSTER SYSTEMS FROM TCP SYN ATTACK

(71) Applicant: **Citrix Systems, Inc.**, Fort Lauderdale, FL (US)

(72) Inventors: **Krishna Khanal**, Bangalore (IN); **Saravana Annamalaisami**, Bangalore (IN); **Mahesh Mylarappa**, Bangalore (IN)

(73) Assignee: **Citrix Systems, Inc.**, Fort Lauderdale, FL (US)

(21) Appl. No.: **14/245,533**

(22) Filed: **Apr. 4, 2014**

Related U.S. Application Data

(60) Provisional application No. 61/809,319, filed on Apr. 6, 2013.

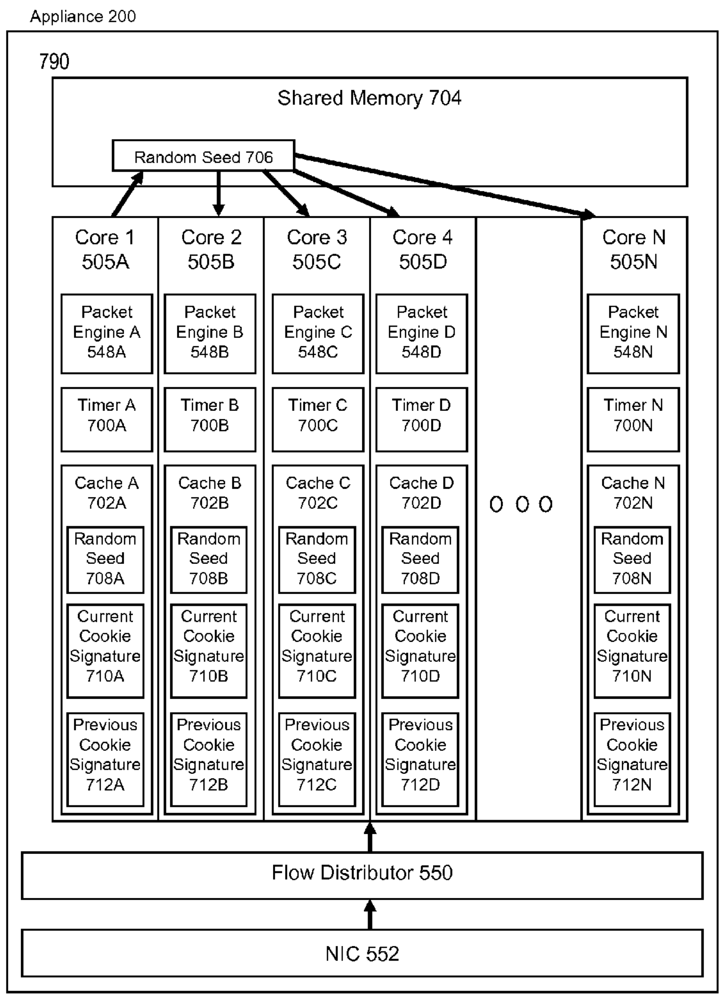
Publication Classification

(51) **Int. Cl.**
H04L 29/06 (2006.01)

(52) **U.S. Cl.**
CPC ***H04L 63/1466*** (2013.01)
USPC **726/22**

(57) ABSTRACT

The present solution is directed to systems and methods for synchronizing a random seed value among a plurality of multi-core nodes in a cluster of nodes for generating a cookie signature. The cookie signature may be used for protection from SYN flood attacks. A cluster of nodes comprises one master node and one or more other nodes. Each node comprises one master core and one or more other cores. A random number is generated at the master core of the master node. The random number is synchronized across every other core. The random number is used to generated a secret key value that is attached in the encoded initial sequence number of a SYN-ACK packet. If the responding ACK packet does not contain the secret key value, then the ACK packet is dropped.



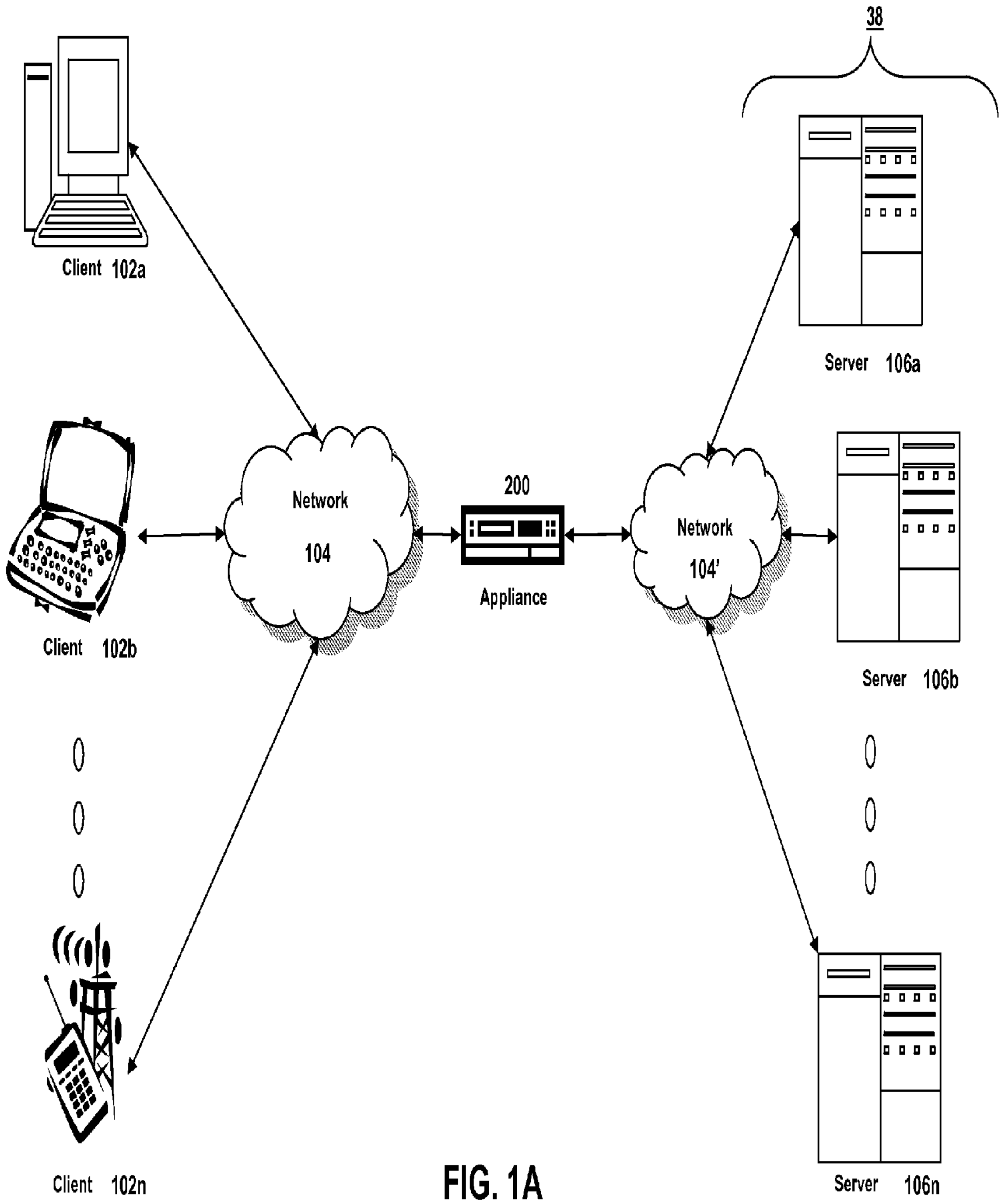

38
Client 102a
Client 102b
Client 102n
Network 104
200
Appliance
Network 104'
Server 106a
Server 106b
Server 106n

FIG. 1A

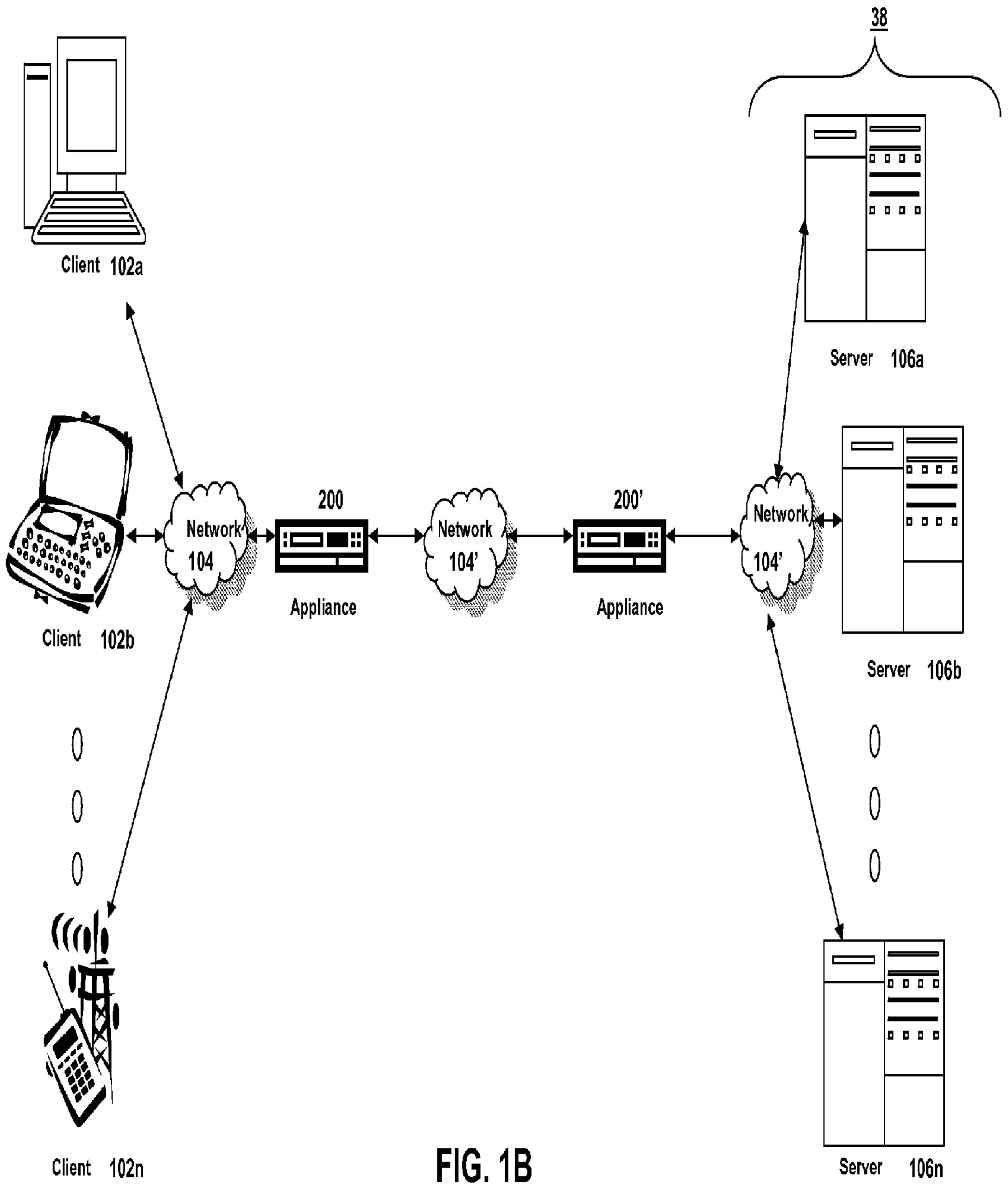

38
Client 102a
Client 102b
Client 102n
Network 104
200
Appliance
Network 104'
200'
Appliance
Network 104'
Server 106a
Server 106b
Server 106n

FIG. 1B

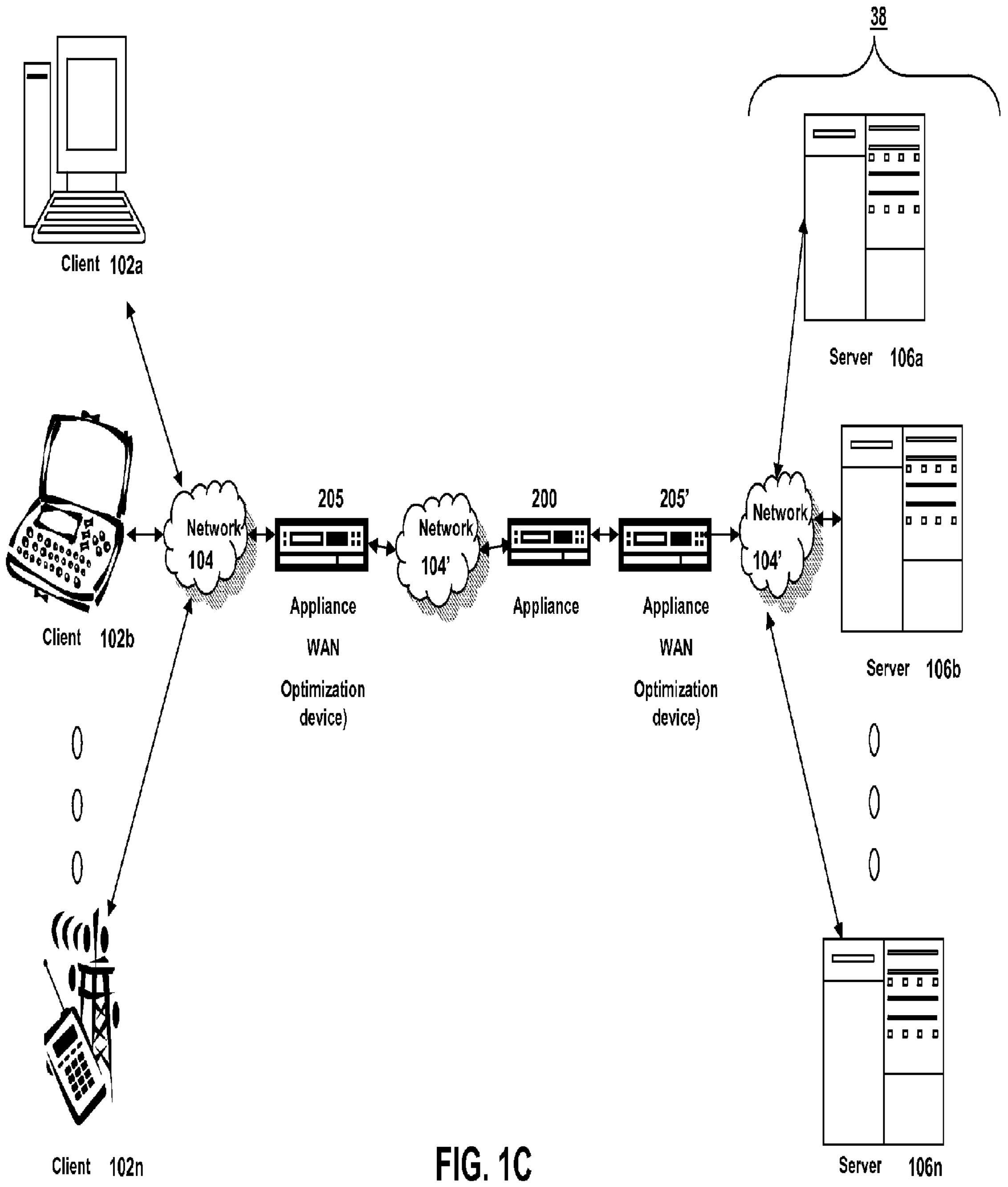

38
Client 102a
Client 102b
Client 102n
Network 104
205
Appliance
WAN
Optimization
device)
Network 104'
200
Appliance
205'
Appliance
WAN
Optimization
device)
Network 104'
Server 106a
Server 106b
Server 106n

FIG. 1C

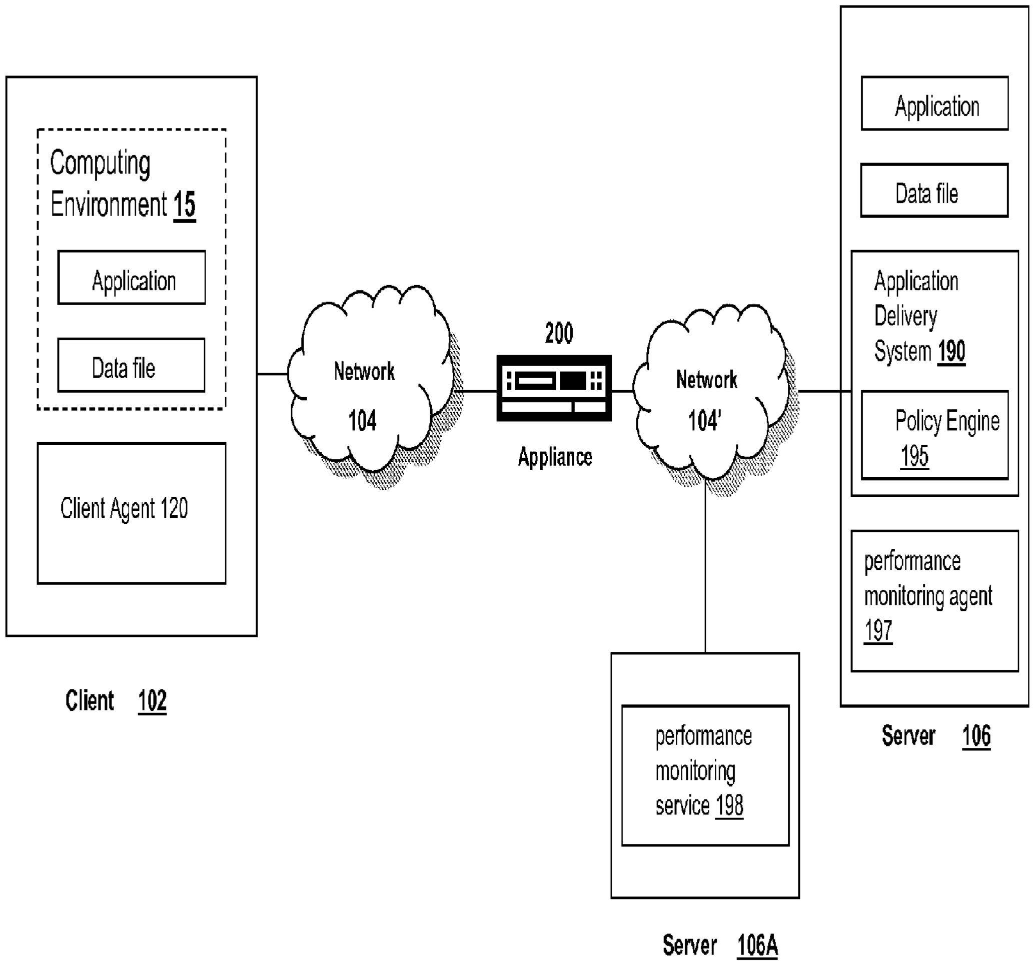

Computing Environment 15
Application
Data file
Client Agent 120
Client 102
Network 104
200
Appliance
Network 104'
Application
Data file
Application Delivery System 190
Policy Engine 195
performance monitoring agent 197
Server 106
performance monitoring service 198
Server 106A

FIG. 1D

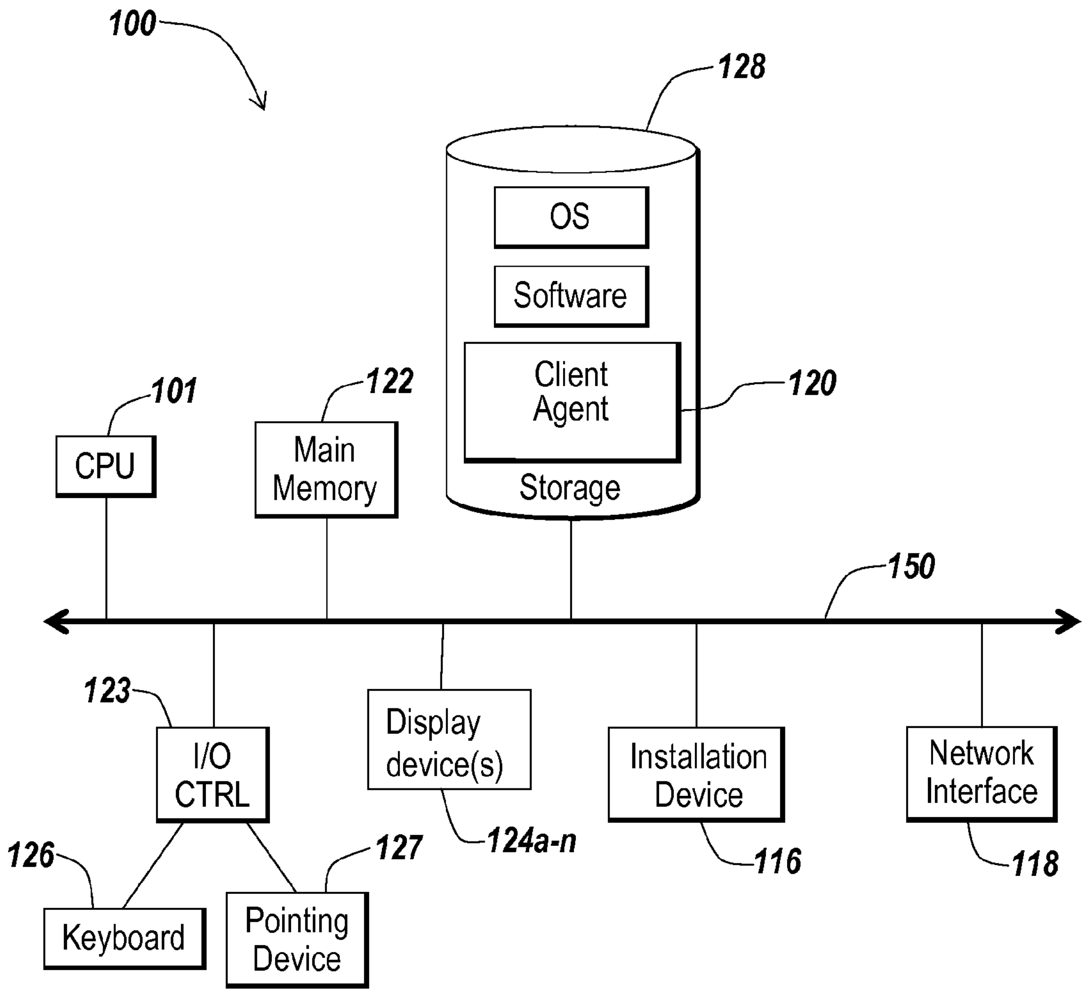

100
128
OS
Software
Client Agent
120
Storage
101
CPU
122
Main Memory
150
123
I/O CTRL
Display device(s)
124a-n
Installation Device
116
Network Interface
118
126
Keyboard
127
Pointing Device

Fig. 1E

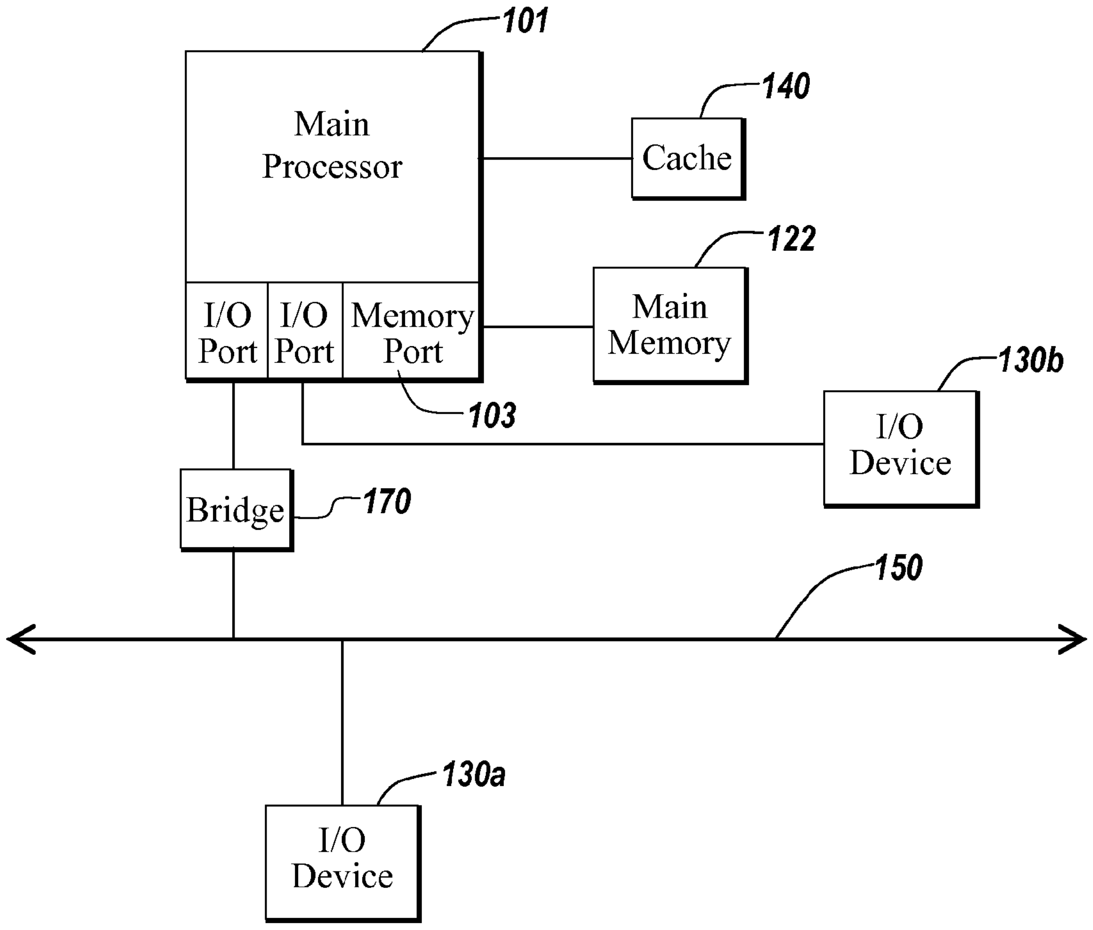

101
Main Processor
I/O Port
I/O Port
Memory Port
103
140
Cache
122
Main Memory
130b
I/O Device
Bridge
170
150
130a
I/O Device

Fig. 1F

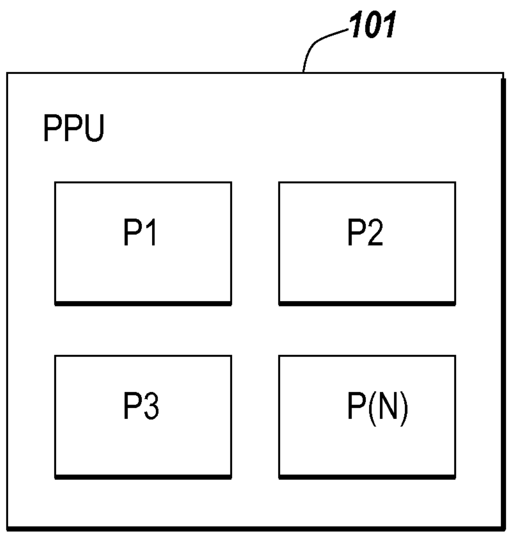


Fig. 1G

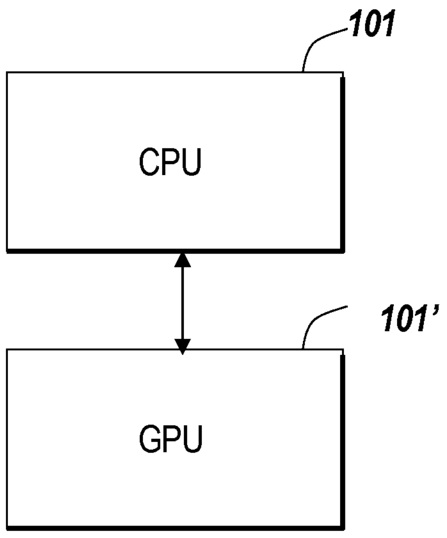


Fig. 1H

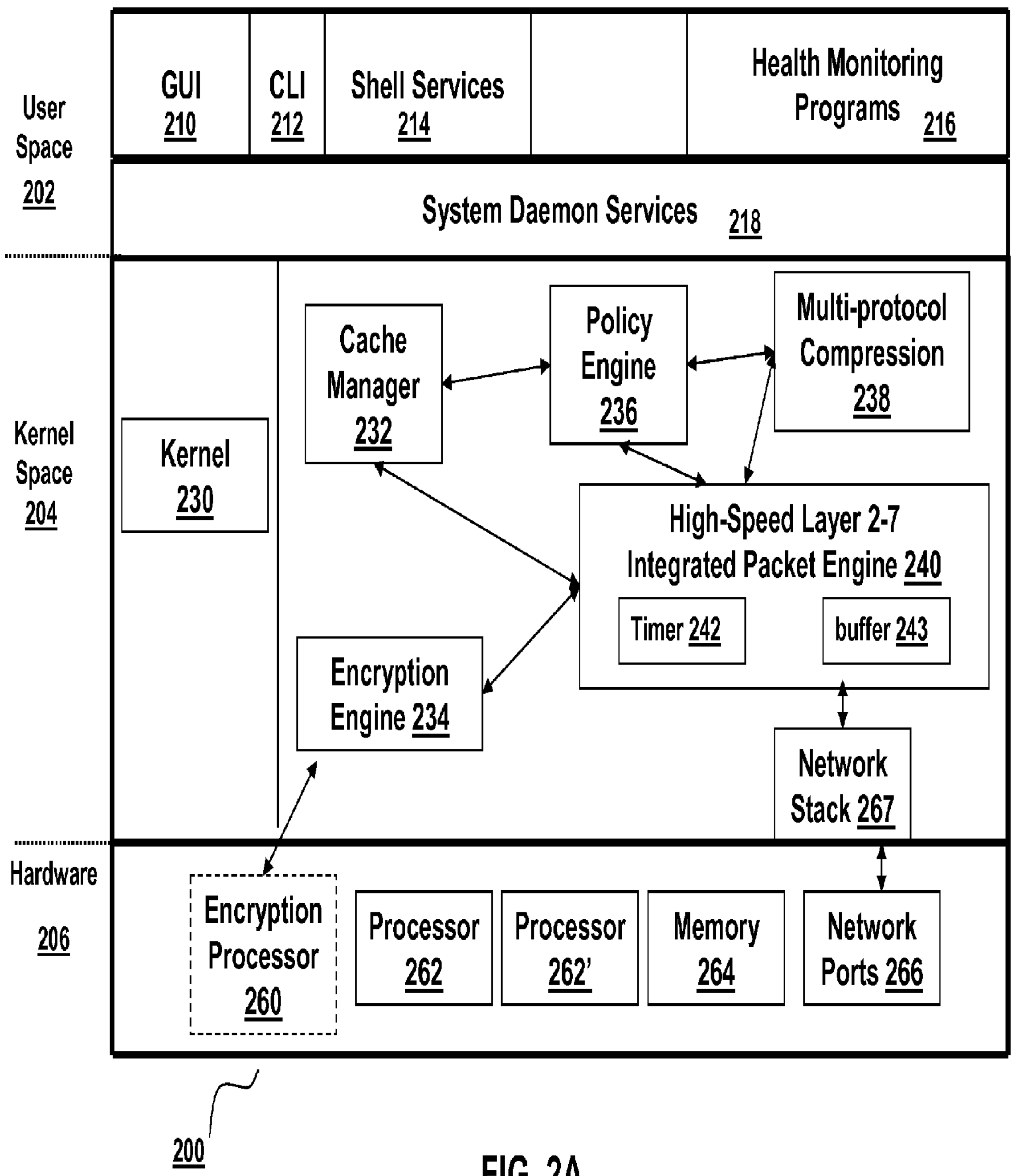

User Space 202
GUI 210
CLI 212
Shell Services 214
Health Monitoring Programs 216
System Daemon Services 218
Kernel Space 204
Kernel 230
Cache Manager 232
Policy Engine 236
Multi-protocol Compression 238
High-Speed Layer 2-7 Integrated Packet Engine 240
Timer 242
buffer 243
Encryption Engine 234
Network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

FIG. 2A

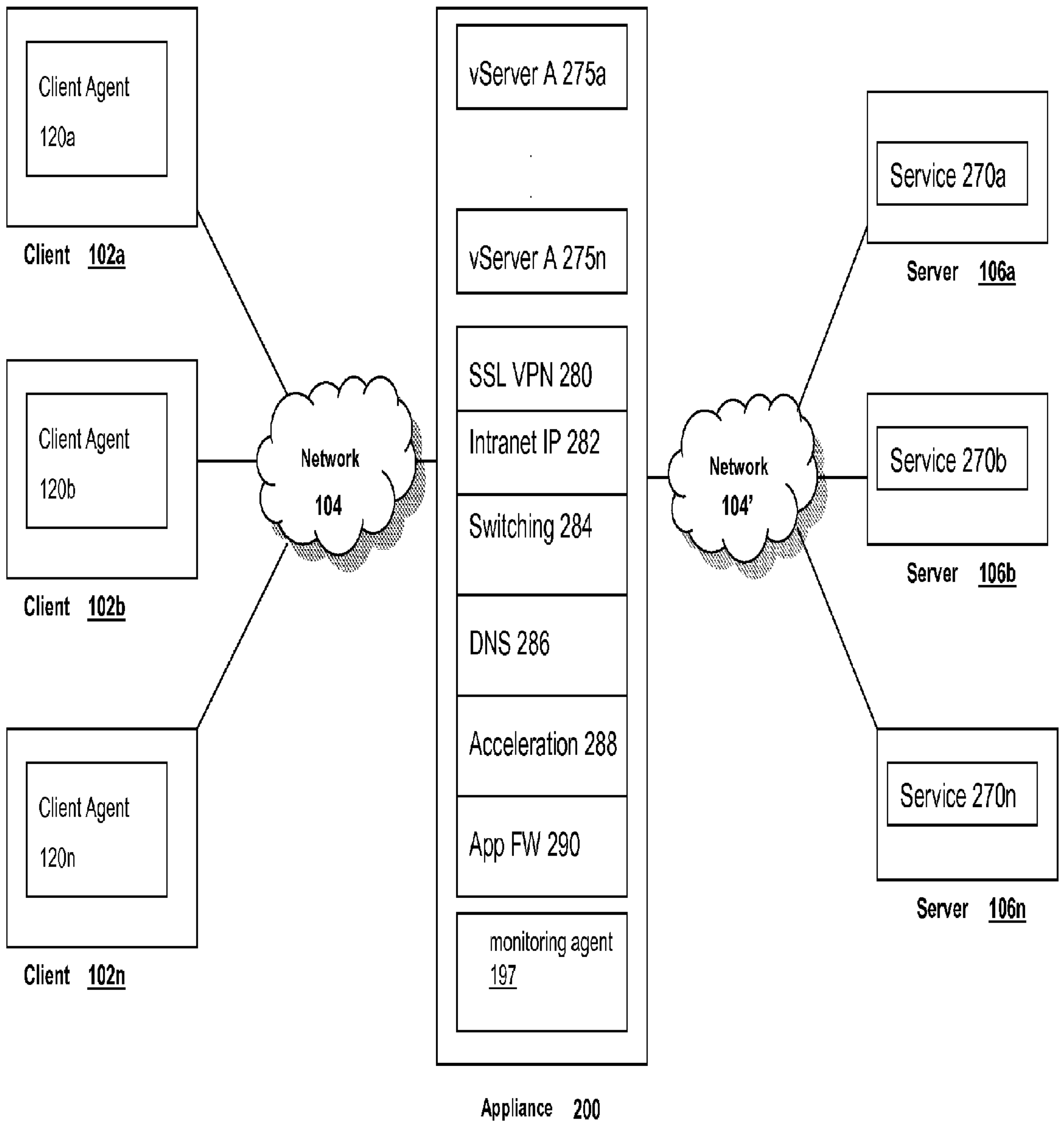

Client Agent 120a
Client 102a
Client Agent 120b
Client 102b
Client Agent 120n
Client 102n
Network 104
vServer A 275a
vServer A 275n
SSL VPN 280
Intranet IP 282
Switching 284
DNS 286
Acceleration 288
App FW 290
monitoring agent 197
Appliance 200
Network 104'
Service 270a
Server 106a
Service 270b
Server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

FIG. 2B

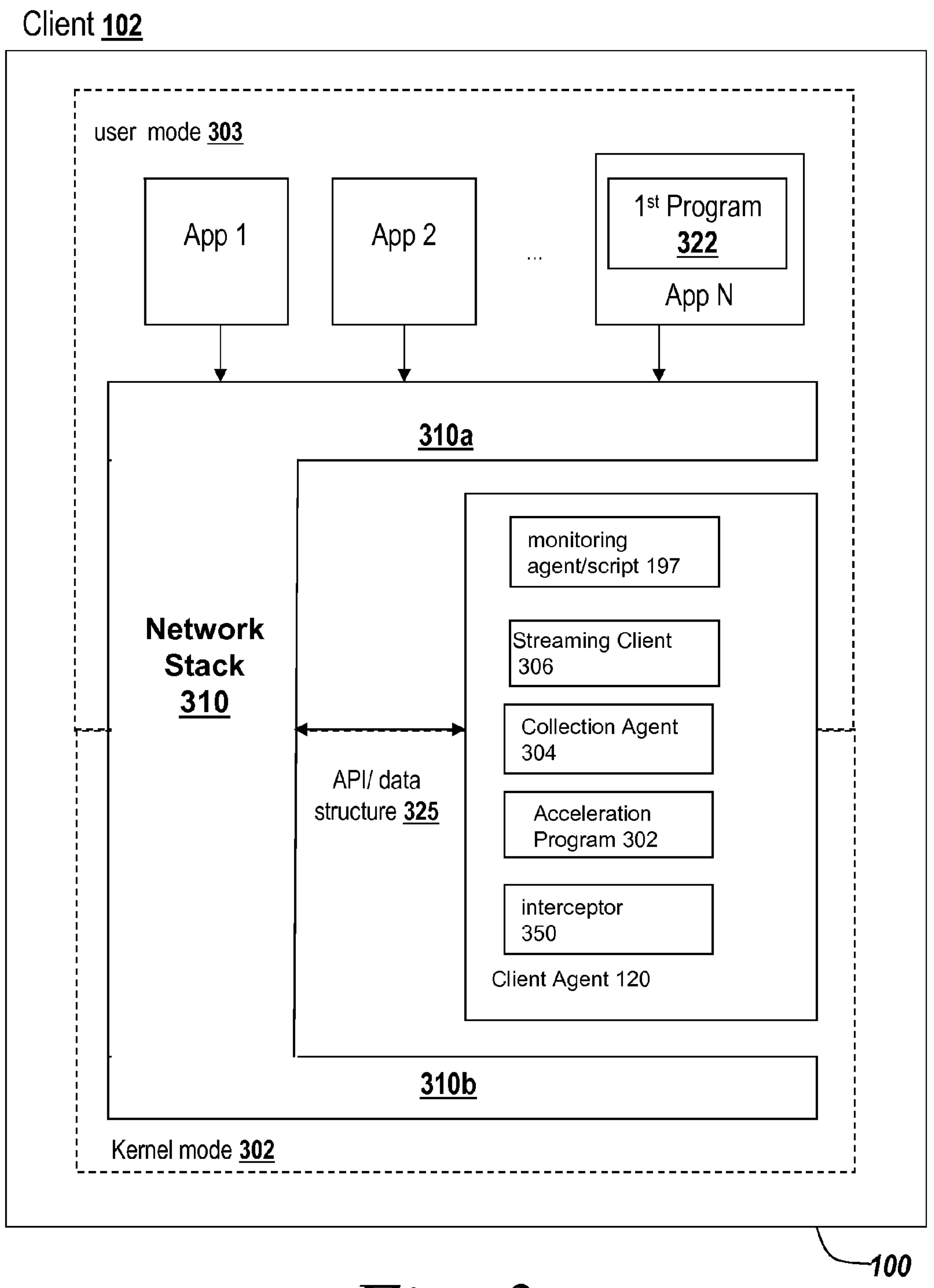

Client 102
user mode 303
App 1
App 2
...
1st Program 322
App N
310a
Network Stack 310
monitoring agent/script 197
Streaming Client 306
Collection Agent 304
Acceleration Program 302
interceptor 350
Client Agent 120
API/ data structure 325
310b
Kernel mode 302
100

Fig. 3

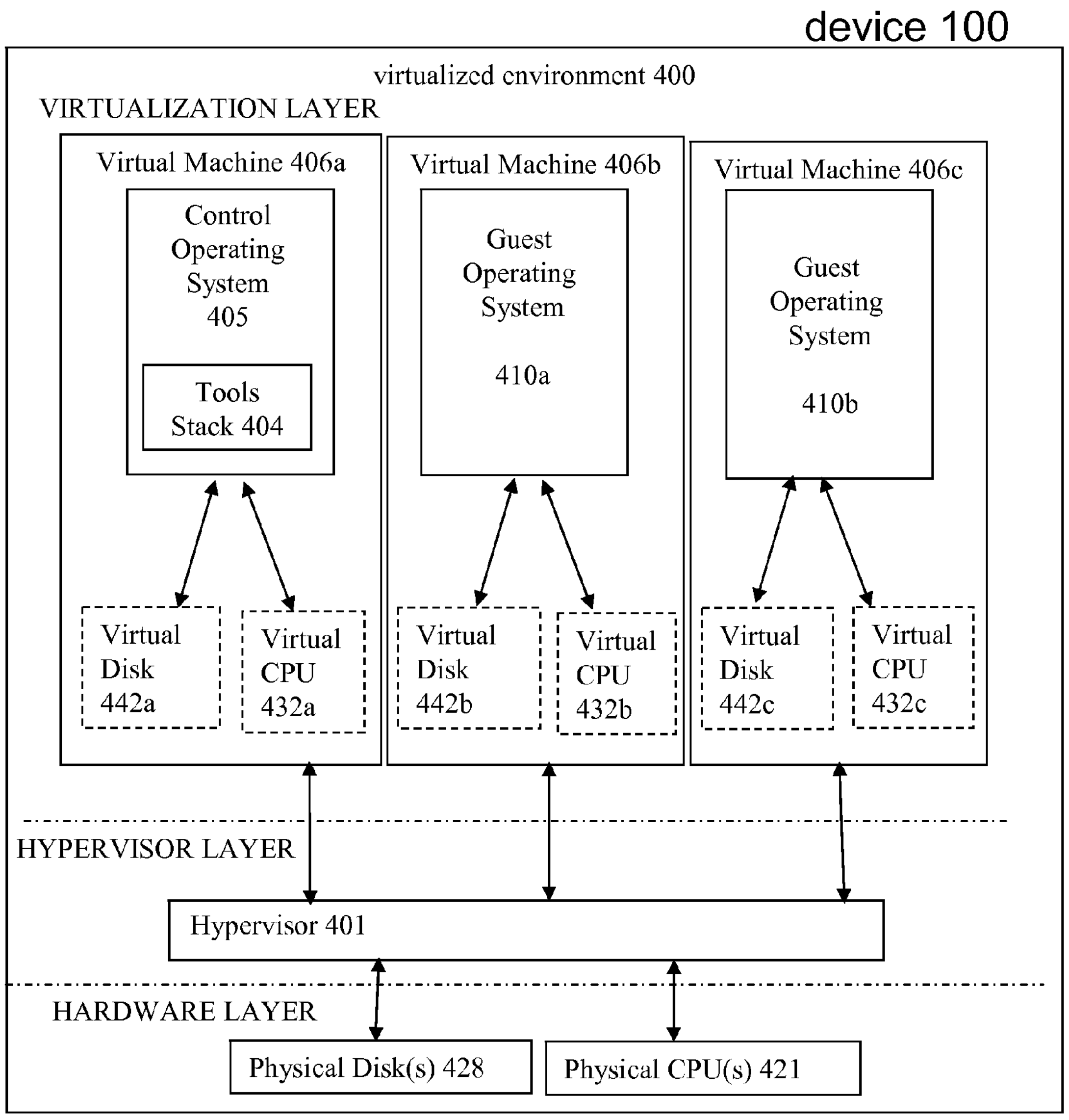

device 100
virtualized environment 400
VIRTUALIZATION LAYER
Virtual Machine 406a
Control Operating System 405
Tools Stack 404
Virtual Disk 442a
Virtual CPU 432a
Virtual Machine 406b
Guest Operating System 410a
Virtual Disk 442b
Virtual CPU 432b
Virtual Machine 406c
Guest Operating System 410b
Virtual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

Fig. 4A

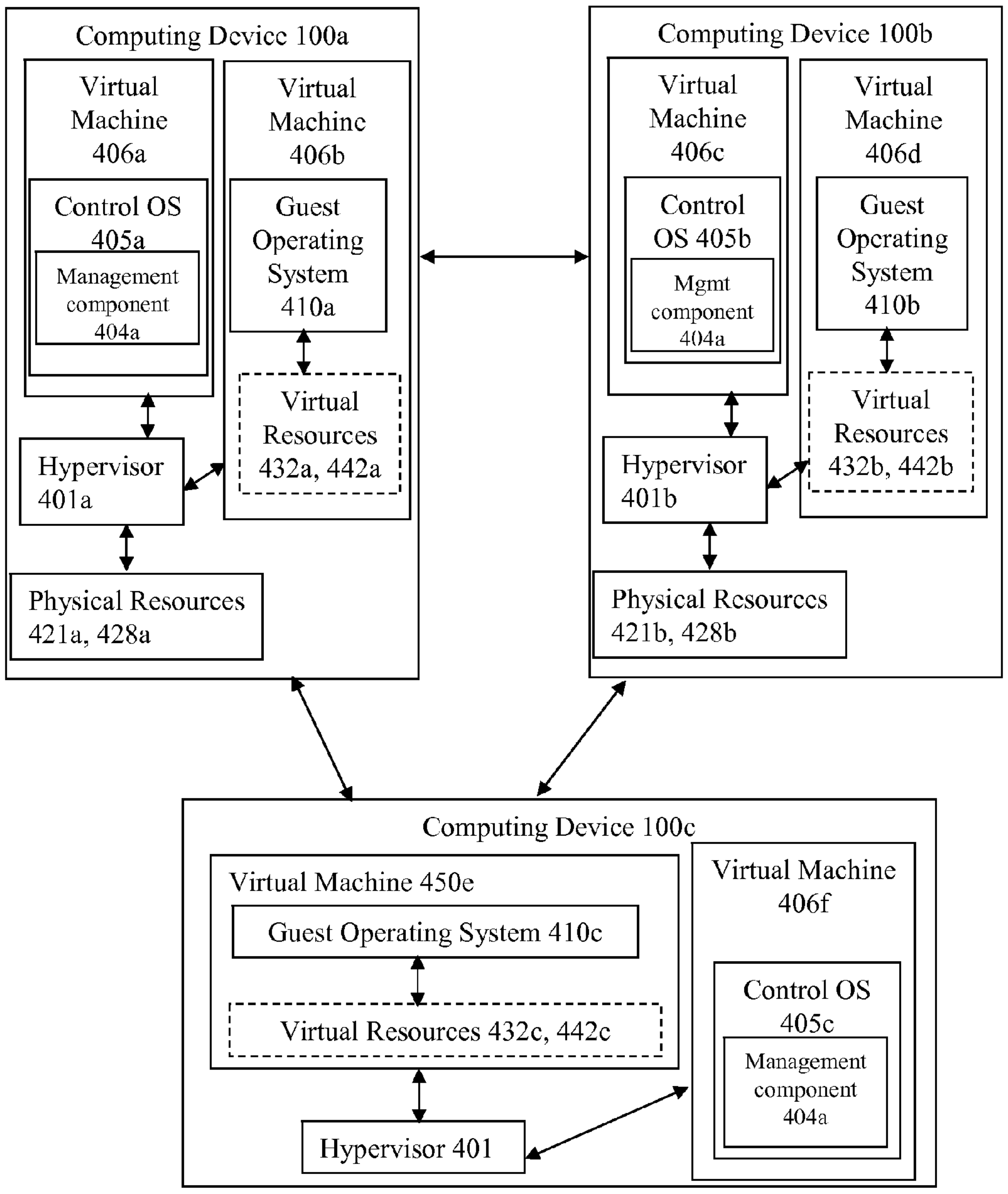

Computing Device 100a
Virtual Machine 406a
Control OS 405a
Management component 404a
Virtual Machine 406b
Guest Operating System 410a
Virtual Resources 432a, 442a
Hypervisor 401a
Physical Resources 421a, 428a
Computing Device 100b
Virtual Machine 406c
Control OS 405b
Mgmt component 404a
Virtual Machine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

Fig. 4B

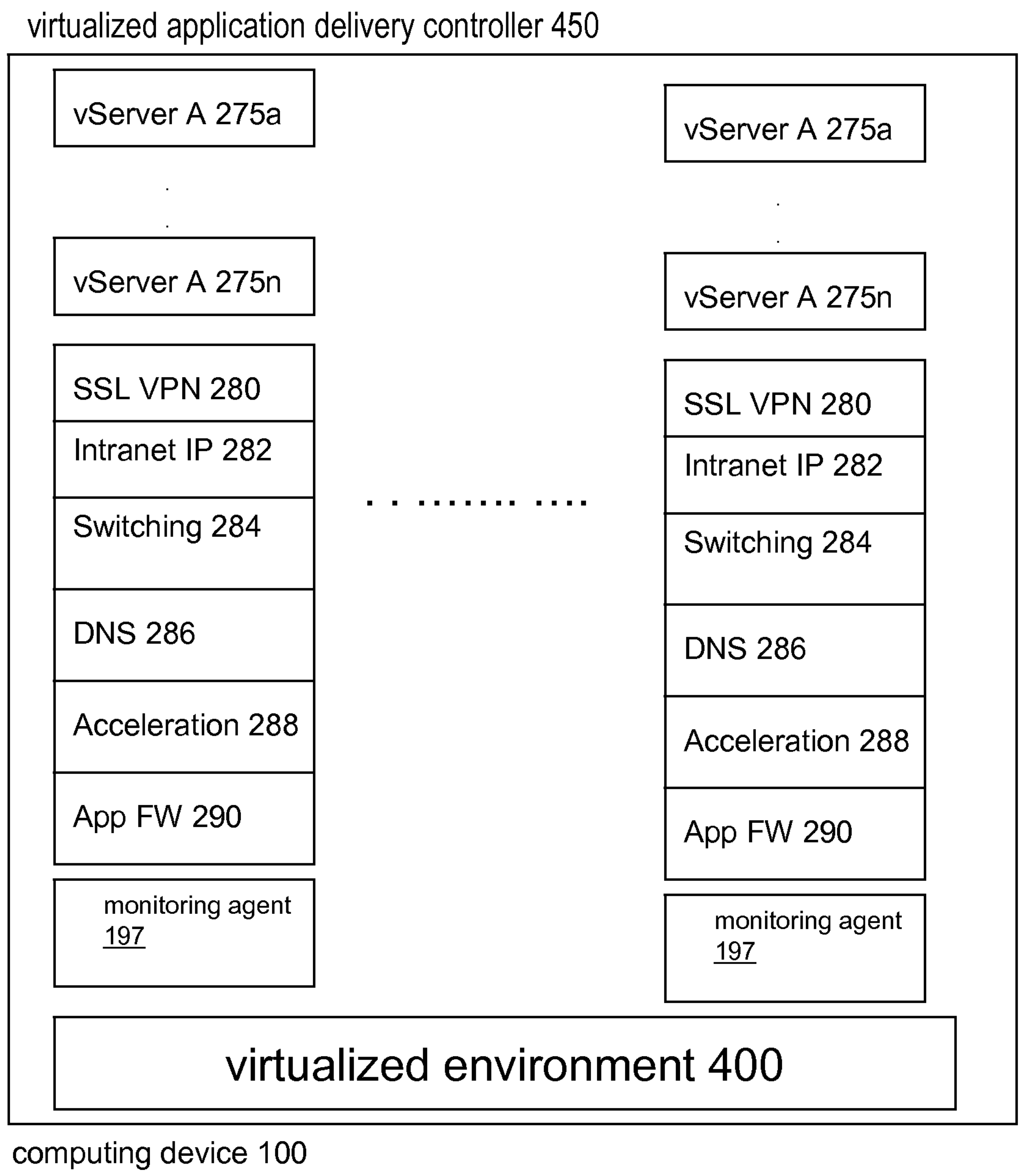

virtualized application delivery controller 450
vServer A 275a
vServer A 275n
SSL VPN 280
Intranet IP 282
Switching 284
DNS 286
Acceleration 288
App FW 290
monitoring agent 197
vServer A 275a
vServer A 275n
SSL VPN 280
Intranet IP 282
Switching 284
DNS 286
Acceleration 288
App FW 290
monitoring agent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

Fig. 4C

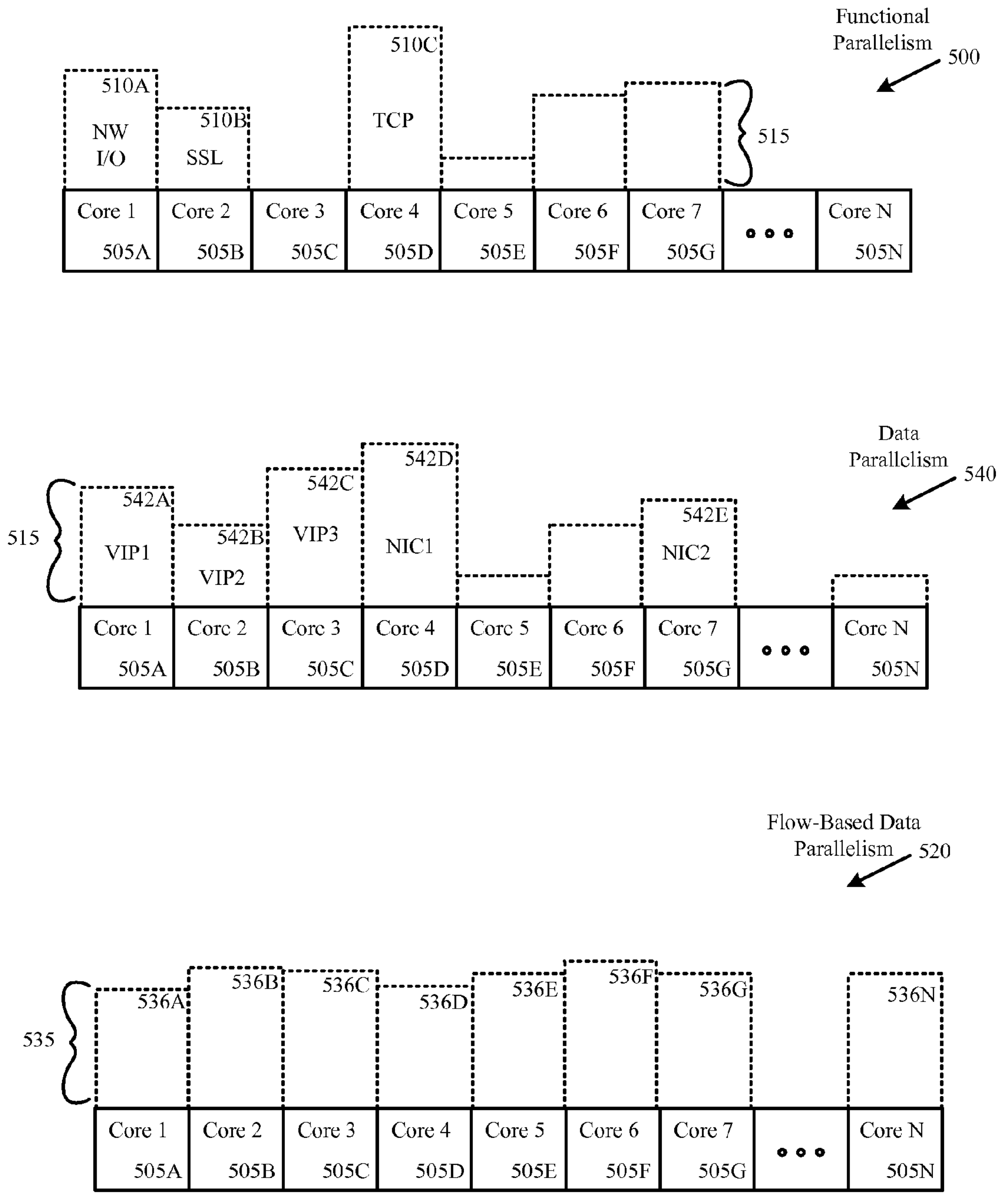

Functional Parallelism
500
510A
NW I/O
510B
SSL
510C
TCP
515
Core 1 505A
Core 2 505B
Core 3 505C
Core 4 505D
Core 5 505E
Core 6 505F
Core 7 505G
Core N 505N
Data Parallclism
540
515
542A
VIP1
542B
VIP2
542C
VIP3
542D
NIC1
542E
NIC2
Flow-Based Data Parallelism
520
535
536A
536B
536C
536D
536E
536F
536G
536N

Fig. 5A

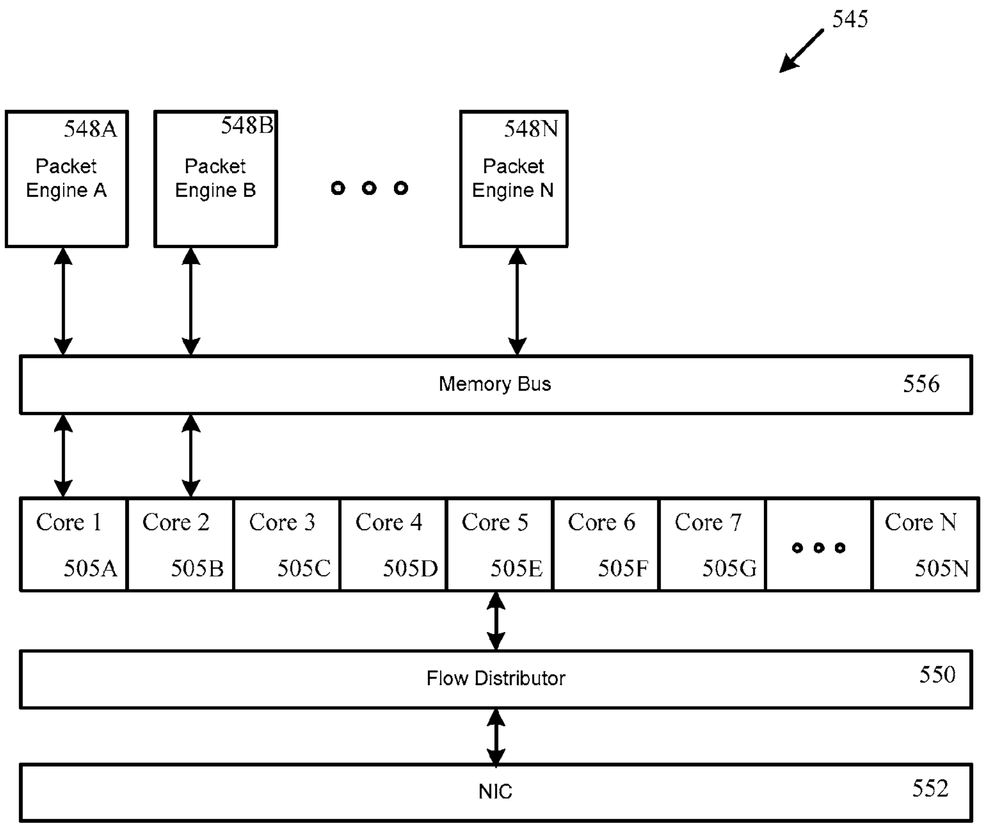

545
548A
Packet Engine A
548B
Packet Engine B
548N
Packet Engine N
Memory Bus
556
Core 1
505A
Core 2
505B
Core 3
505C
Core 4
505D
Core 5
505E
Core 6
505F
Core 7
505G
Core N
505N
Flow Distributor
550
NIC
552

Fig. 5B

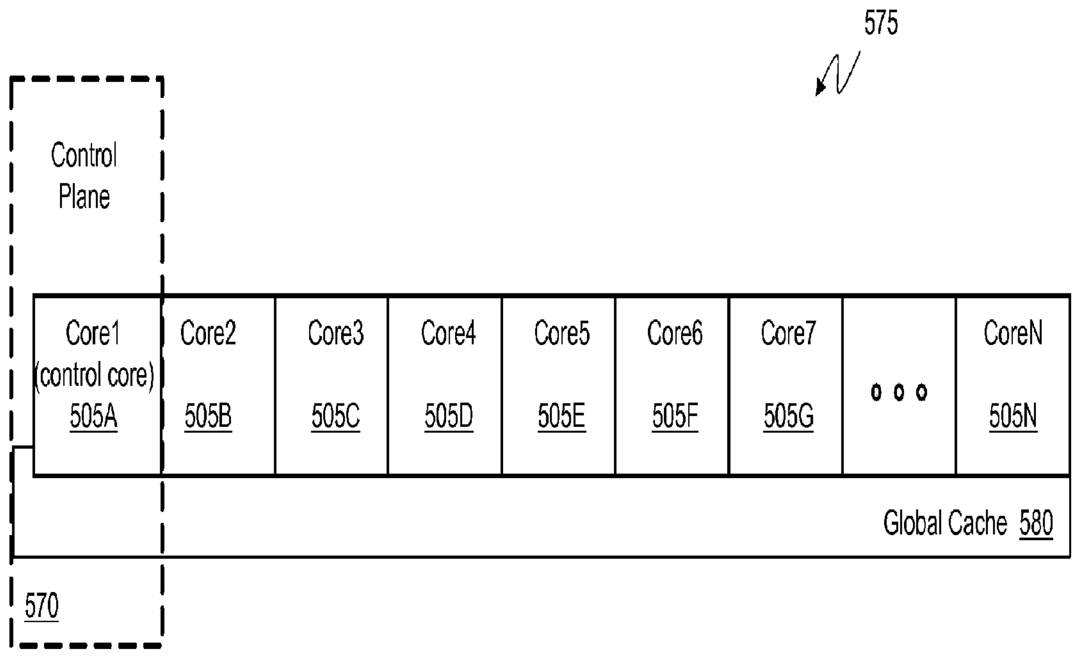

575
Control Plane
Core1 (control core) 505A
Core2 505B
Core3 505C
Core4 505D
Core5 505E
Core6 505F
Core7 505G
CoreN 505N
Global Cache 580
570

FIG. 5C

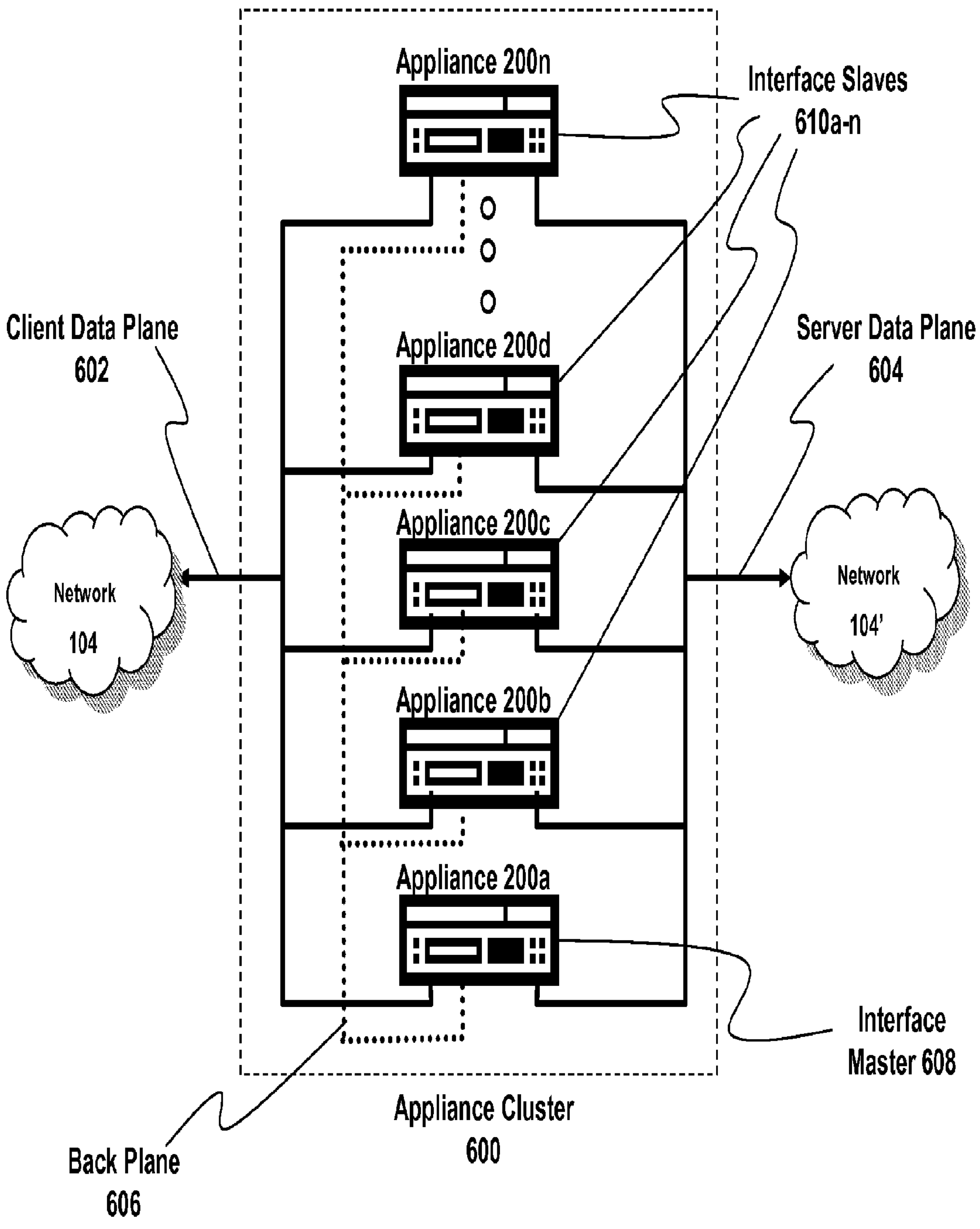

Appliance 200n
Interface Slaves 610a-n
Client Data Plane 602
Appliance 200d
Server Data Plane 604
Appliance 200c
Network 104
Network 104'
Appliance 200b
Appliance 200a
Interface Master 608
Appliance Cluster 600
Back Plane 606

FIG. 6

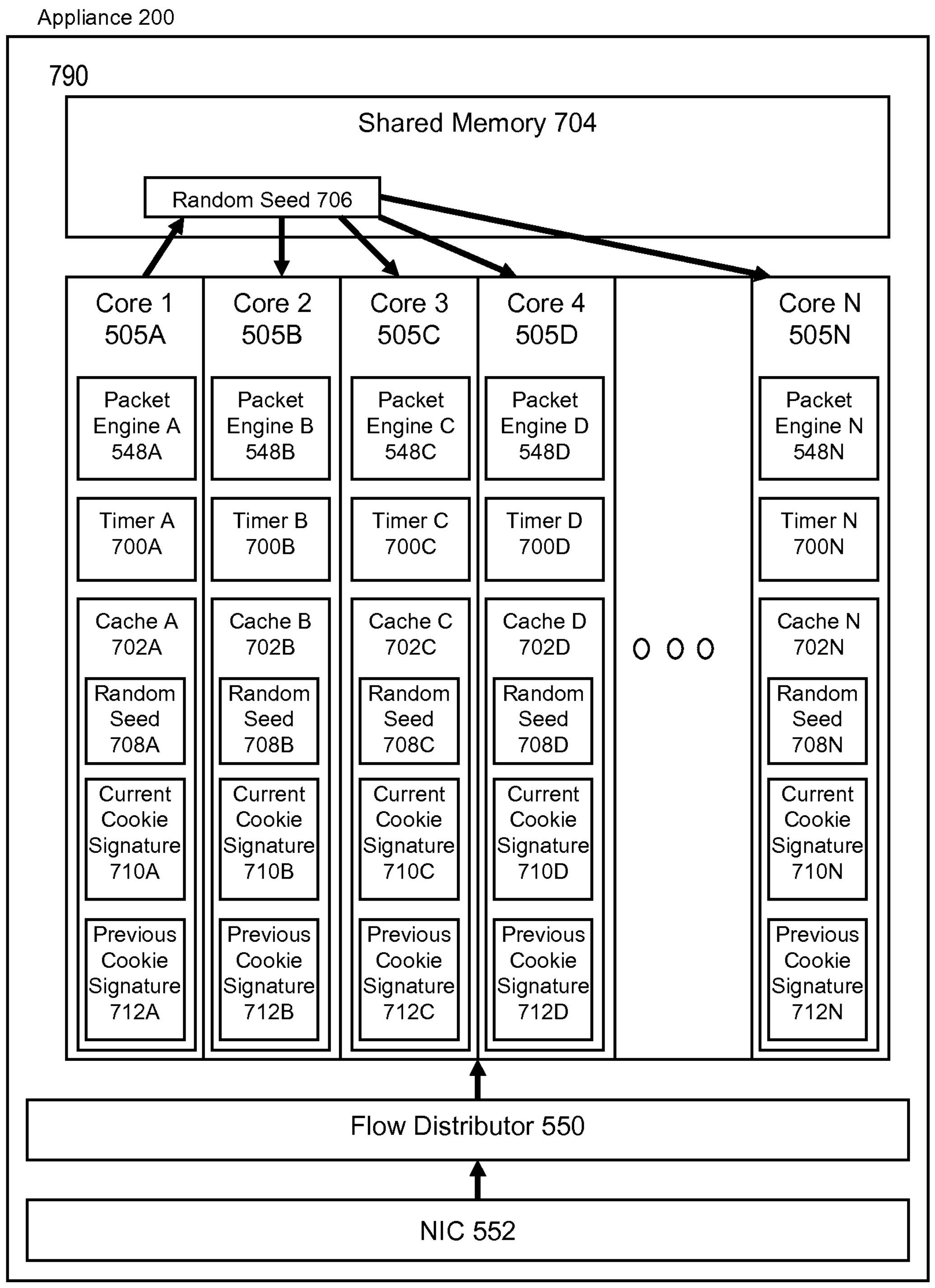

Appliance 200
790
Shared Memory 704
Random Seed 706
Core 1 505A
Core 2 505B
Core 3 505C
Core 4 505D
Core N 505N
Packet Engine A 548A
Packet Engine B 548B
Packet Engine C 548C
Packet Engine D 548D
Packet Engine N 548N
Timer A 700A
Timer B 700B
Timer C 700C
Timer D 700D
Timer N 700N
Cache A 702A
Cache B 702B
Cache C 702C
Cache D 702D
Cache N 702N
Random Seed 708A
Random Seed 708B
Random Seed 708C
Random Seed 708D
Random Seed 708N
Current Cookie Signature 710A
Current Cookie Signature 710B
Current Cookie Signature 710C
Current Cookie Signature 710D
Current Cookie Signature 710N
Previous Cookie Signature 712A
Previous Cookie Signature 712B
Previous Cookie Signature 712C
Previous Cookie Signature 712D
Previous Cookie Signature 712N
Flow Distributor 550
NIC 552

FIG. 7A

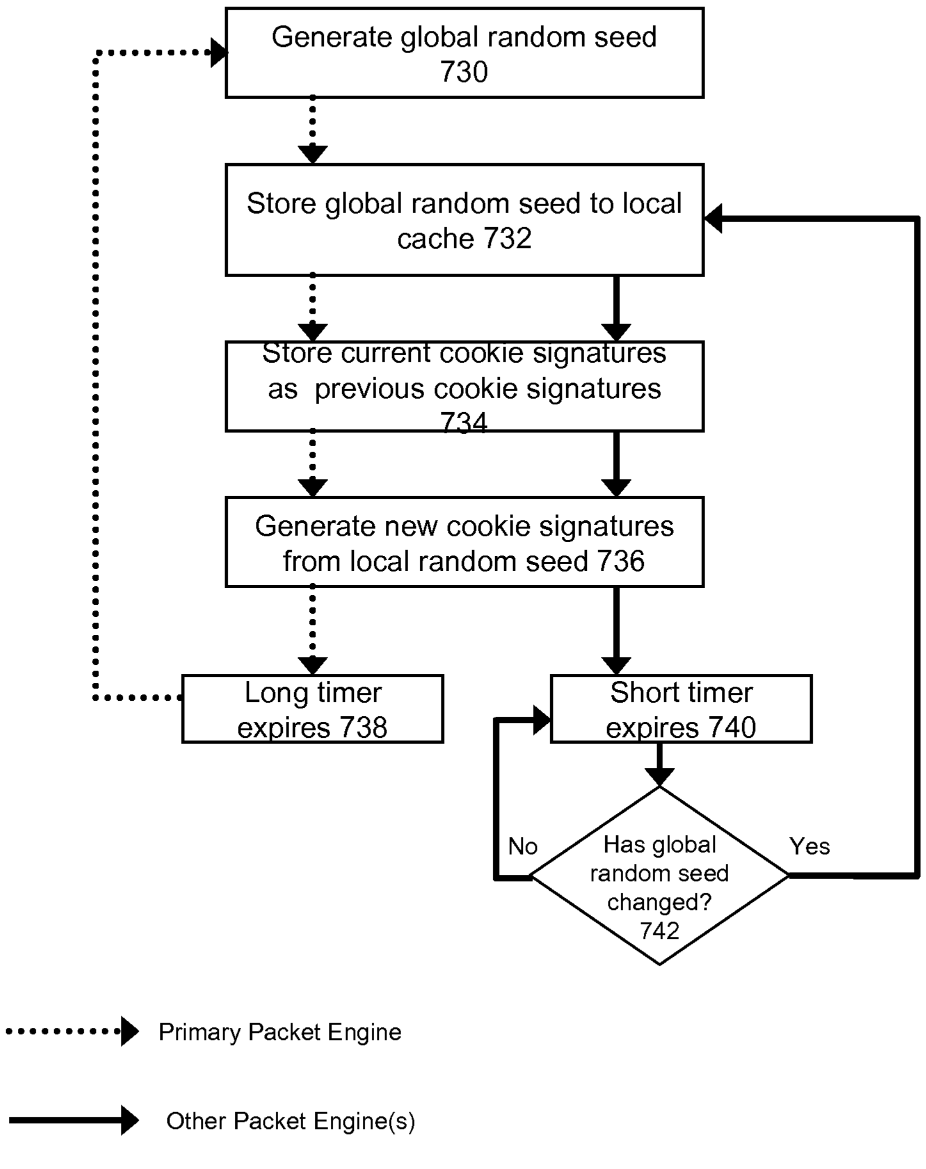

Generate global random seed 730
Store global random seed to local cache 732
Store current cookie signatures as previous cookie signatures 734
Generate new cookie signatures from local random seed 736
Long timer expires 738
Short timer expires 740
Has global random seed changed? 742
No
Yes
Primary Packet Engine
Other Packet Engine(s)

FIG. 7B

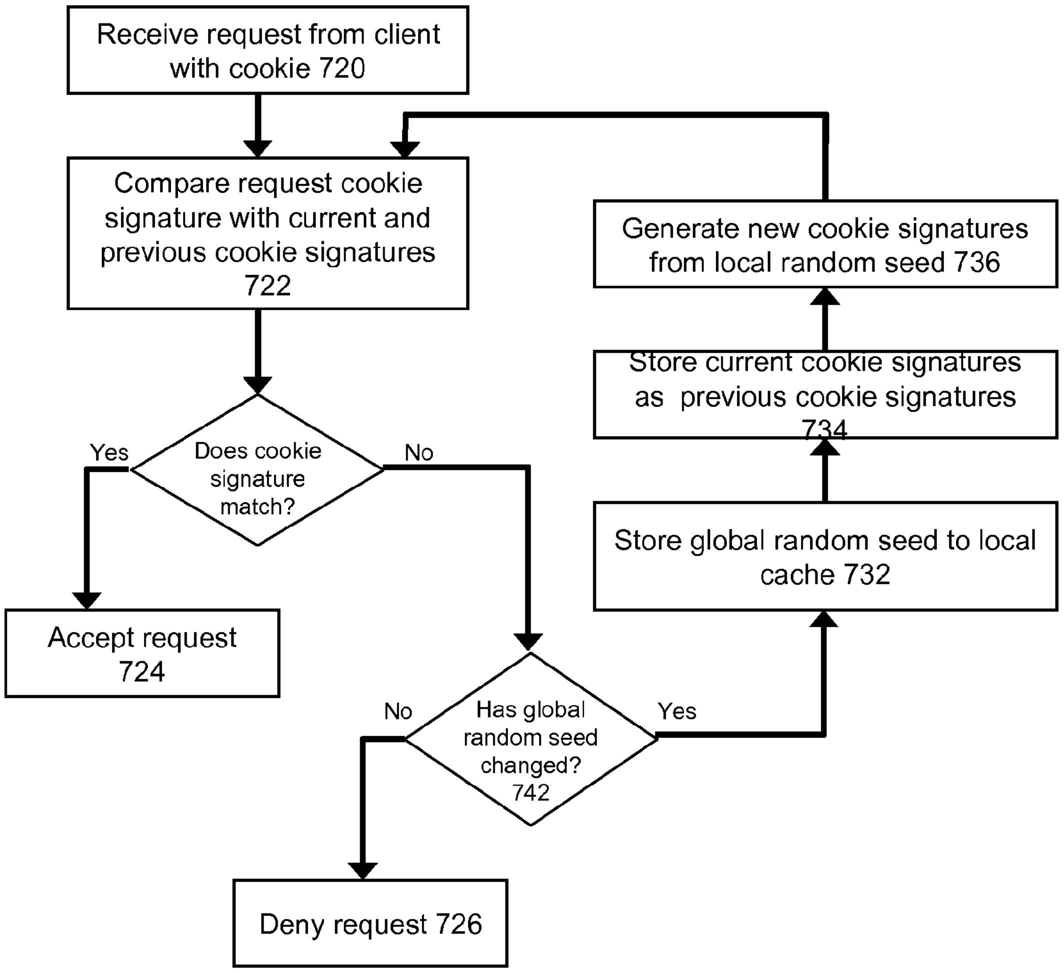

Receive request from client with cookie 720
Compare request cookie signature with current and previous cookie signatures 722
Does cookie signature match?
Yes
No
Accept request 724
Has global random seed changed? 742
No
Yes
Deny request 726
Store global random seed to local cache 732
Store current cookie signatures as previous cookie signatures 734
Generate new cookie signatures from local random seed 736

FIG. 7C

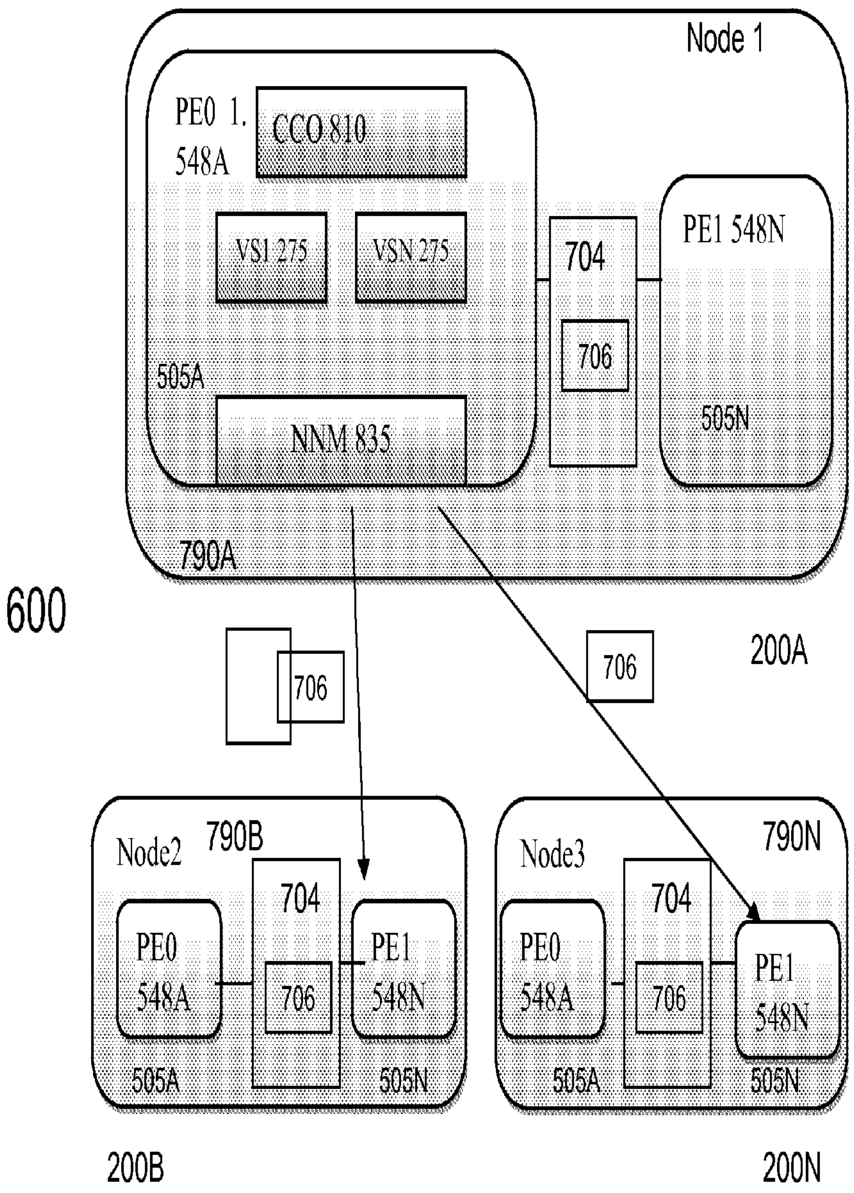

Node 1
PE0 1. 548A
CCO 810
VS1 275
VSN 275
505A
NNM 835
704
706
PE1 548N
505N
790A
600
706
706
200A
Node2
790B
704
PE0 548A
706
PE1 548N
505A
505N
Node3
790N
704
PE0 548A
706
PE1 548N
505A
505N
200B
200N

FIG. 8A

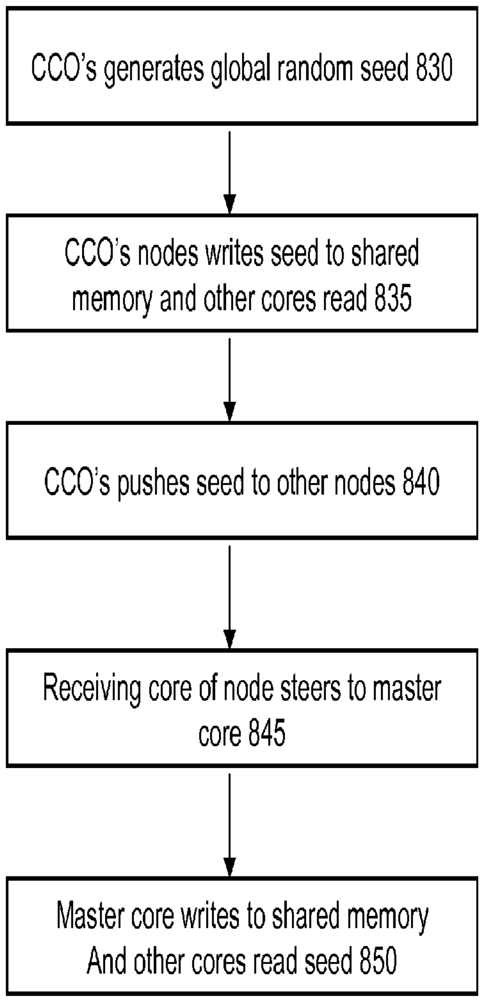

CCO's generates global random seed 830
CCO's nodes writes seed to shared memory and other cores read 835
CCO's pushes seed to other nodes 840
Receiving core of node steers to master core 845
Master core writes to shared memory And other cores read seed 850

FIG. 8B

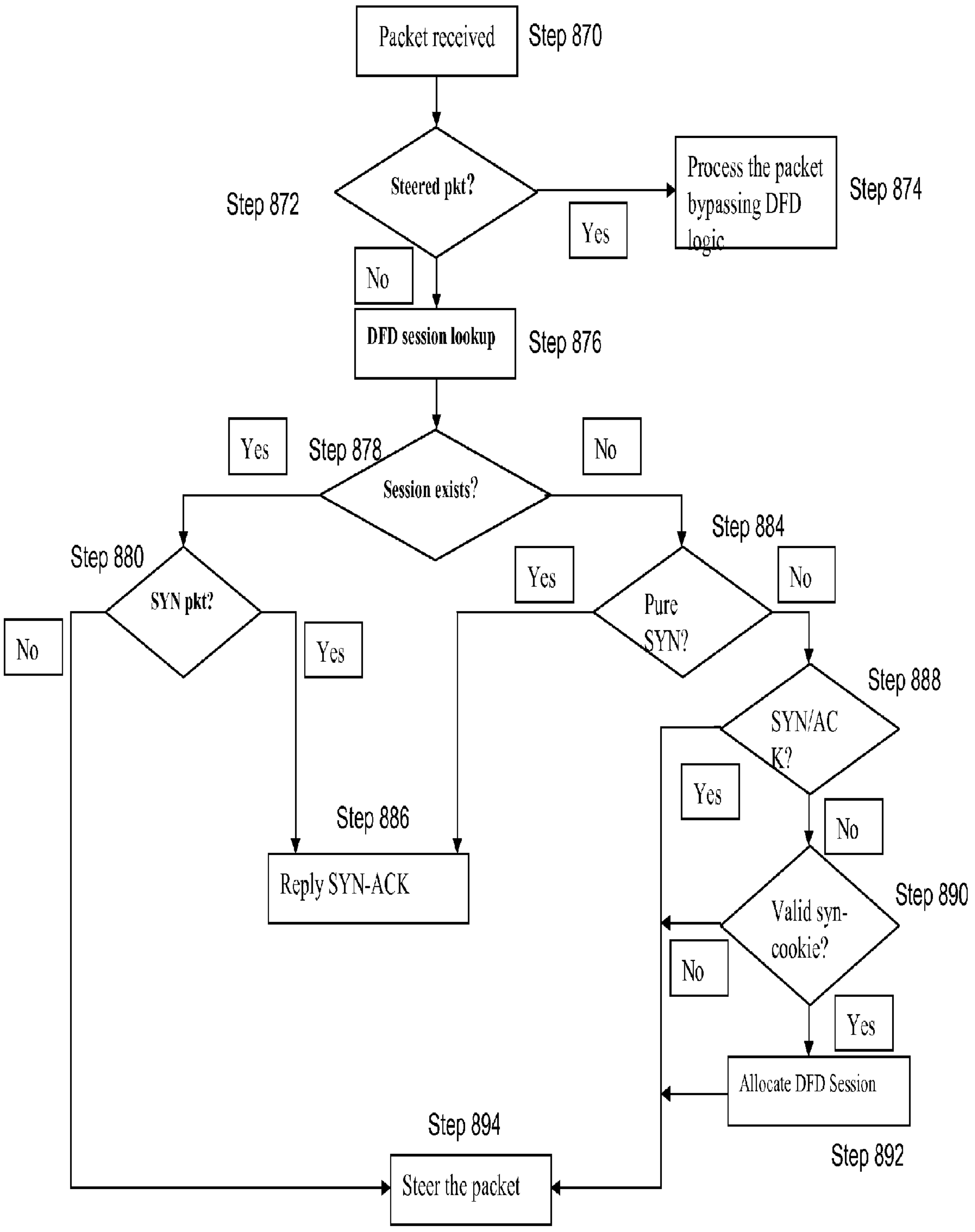

Packet received
Step 870
Steered pkt?
Step 872
Yes
Process the packet bypassing DFD logic
Step 874
No
DFD session lookup
Step 876
Yes
Step 878
Session exists?
No
Step 880
SYN pkt?
No
Yes
Step 884
Yes
Pure SYN?
No
Step 888
SYN/ACK?
Yes
No
Step 886
Reply SYN-ACK
Valid syn-cookie?
Step 890
No
Yes
Allocate DFD Session
Step 892
Step 894
Steer the packet

FIG. 8C

SYSTEMS AND METHODS FOR PROTECTING CLUSTER SYSTEMS FROM TCP SYN ATTACK

RELATED APPLICATION

[0001] The present application claims the benefit of and priority to U.S. Provisional Patent Application No. 61/809,319 entitled "Systems and Methods for Protecting Cluster Systems from TCP SYN Attack" and filed on May 9, 2013, which is hereby incorporated by reference in its entirety for all purposes.

FIELD OF THE DISCLOSURE

[0002] The present application generally relates to data communication networks. In particular, the present application relates to systems and methods for protecting clustered networking devices from Transport Control Protocol (TCP) SYN attacks.

BACKGROUND

[0003] A cluster of network devices may have each network device configured to establish transport layer connections with clients and servers. Each network device in the cluster may be a multi-core device with each configured to establish transport layer connections. A transport control protocol (TCP) connection may be established responsive to a handshake in which SYN and SYN ACK are exchanged between the end points. A device, such as multi-core device or a node in a cluster, may be subject to a SYN flood attack. A SYN flood is a form of denial-of-service attack in which an attacker sends a succession of SYN requests to a target device to try to consume enough server resources to make the device unresponsive to legitimate traffic.

BRIEF SUMMARY

[0004] In some aspects, the present solution is directed towards protecting from SYN flood attacks in a cluster of networking devices via the generation, synchronization and use of a SYN-cookie for the cluster. For a node having multiple cores, the node follows a master-slave concept to manage the task of maintaining the SYN-cookie same across the cores. Cores use shared memory to store the cookie. A packet engine on a core, such as a first packet engine, is designated a master packet engine. The master packet engine generates the cookie seed and writes to the shared memory at a predetermined frequency, such as every 120 secs. The other packet engines on the other cores read the seed at a predetermined frequency, such as 1 sec. from shared memory. Since the same seed is used by all the packet engines, SYN-cookie generated from the same side is valid across the cores.

[0005] In further details of this process, upon bootup, the multi-core device initializes shared memory parameters. At this point, the master packet engine chooses the first seed and initializes the seed on the shared memory. All the cores initialize the seed writing time (e.g., master packet engine every 120 sec.) and reading time (e.g., other packet engines every 1 sec.) and reads the initial seed stored by the master packet engine. All cores now have the same initial seed. The master packet engine will update the seed in shared memory at the predetermined frequency and each of the other cores and their packet engines read the updated seed at a second predetermined frequency.

[0006] A core or packet engine while processing final ACK of a TCP handshake, tires to match the clients ACK with the current cookie value. If there is a match, the packet engine establishes the connection and creates resources, such as record, for the connection. If there is not a match, the packet engine tries to match the client's final ACK with the previous cookie (i.e. previous seed). If there is a match with previous cookie, the packet engine establishes the connection and creates resources for the connection. If there is still not a match, the packet engine determines if current seed is not same as seed in shared memory. If so, the packet engine reads the seed from shared memory and updates the current and previous seed. The packet engine will try to match the cookie with the updated current and previous seed values and if not a match, the packet engine drops the connection.

[0007] The present solution addresses the use of SYN-cookies for clusters by providing SYN-cookie seed generation and synchronization across the nodes in cluster and generating SYN-ACK from the flow receiver of the cluster. For SYN-cookies seed generation and synchronization, the cluster may follow a similar master-slave mechanism of a multi-core device to generate and synchronize the SYN-cookies across the nodes in the cluster. A first packet engine on a master node may have the responsibility of generating and synchronizing the cookie seed. The master node pushes seed updates by broadcasting node to node messages to all the other nodes to update the seed on all the nodes. The master node may perform a push at a predetermined frequency, such as every 120 second, when the owner or master packet engine on the master generated the new seed. For updating the cores within the master node, the packet engine may write the new seed to the shared memory at the predetermined frequency. The packet engines on the other cores within node can only read the seed from shared memory. When master node sends node to node messages to the other nodes in the cluster, the message can land onto any core or packet engine in target node. The receiving packet engine may steer the message to the master packet engine for seed updates. The master packet engine updates the seed in shared memory, such as at the predetermined frequency or next predetermined frequency and the other cores read the seed from the shared memory. So as a result of one packet engine on a master node updating the seed, the other cores within the master node as well as the other nodes in the cluster and each of their respective cores are updated with the new seed.

[0008] Before enabling the cluster instance (or before creating a one node cluster), the present solution uses master and slave mechanism between cores within that node to generate and synchronize the seed across the packet engines of that node. As soon as cluster instance is enabled on first node (e.g., on creating one node cluster), the node by default becomes the cluster configuration owner (CCO) or master node and initializes the node to node and inter-core communication channels. Once the master packet engine or core generates next seed on the master node, the master node broadcasts this new seed to the other nodes in the cluster.

[0009] When a new node joins the cluster, master node receives the node join event and broadcasts the current seed to the nodes. Until new node receives the seed from the master node, the new node remains in a SYN-cookie synchronization in progress state, in which the new node will be able to accept control connections but not the data connections. When new node receives the seed, the new node generates the current cookie and previous current cookie becomes the previous cookie. So the new node will have the same current cookie as the master node but the previous cookie will be different from the previous cookie of the master node. As a result, that new node will not be able to validate the connections initiated using previous cookie. When new node receives the seed the second time onwards, the new node's current and previous cookie will be the same as the current and previous cookie of the master node.

[0010] In some aspects, the present solution is directed to methods for synchronizing a random seed value among a plurality of multi-core nodes in a cluster of nodes for generating a cookie signature. A cookie signature can be any form of a digital signature. The methods generally include generating, by a master core on a master node of a cluster of nodes comprising a plurality of cores, a random seed to be synchronized across each core of each node in the cluster of nodes. The methods generally include storing, by the master core on the master node, the random seed to memory on the master node accessible by each core in the master node. The methods generally include receiving, by each master core on each other node in the cluster, the random seed sent by the master core of the master node. The methods generally include storing, by each master core on each other node in the cluster, the random seed to memory on each node accessible by each core in the each other node. The methods generally include generating, by each core of each node in the cluster of nodes, a cookie signature based on the random seed responsive to a predetermined timer.

[0011] In some applications of these methods, the methods may include receiving, by a receiving core on each other node in the cluster, the random seed sent by the master core of the master node, and steering, by each receiving core, the random seed to a master core in each other node in the cluster. The methods may include storing, at each core of each node in the cluster of nodes, a current cookie signature as a previous cookie signature, and the generated cookie signature as the current cookie signature. The methods may include generating, by the master core on the master node of the cluster of nodes, the random seed, responsive to a second predetermined timer set to expire longer than the predetermined timer.

[0012] In some applications of these methods, the methods may include generating, by each core of each node in the cluster of nodes, an array of cookie signatures. The methods may include generating, by each core of each node in the cluster of nodes, an array of cookie signatures, by using the random seed as an initial seed of a pseudo-random number function to generate a first cookie signature, and generating each successive cookie signature by using a preceding cookie signature as a seed for the pseudo-random number function. The methods may include generating a cookie by concatenating one or more cookie signatures in the array of cookie signatures. The methods may include using the generated cookie signature as part of a SYN cookie or a HTTP DoS cookie.

[0013] In some applications of these methods, the methods may include receiving from a client, a SYN request at a first core of a node in the cluster. The methods may further include responding to the client with a SYN-ACK message comprising a cookie with the cookie signature. The methods may further include receiving from the client, an ACK message at a second core of the node in the cluster, the ACK message comprising a client cookie signature. The methods may further include accepting the ACK message in response to matching the client cookie signature with the cookie signature.

[0014] In some applications of these methods, the methods may include storing, at each core of each node in the cluster of nodes, a current cookie signature as a previous cookie signature, and the generated cookie signature as the current cookie signature. The methods may further include comparing the client cookie signature with the current cookie signature and the previous cookie signature. The methods may further include determining whether a new random seed is stored in a memory accessible by the second core. The methods may further include storing, at the second core, the current cookie signature as the previous cookie signature. The methods may further include generating a new current cookie signature based on the new random seed in the memory accessible by the second core. The methods may further include allocating resources in response to matching the client cookie signature with the new current cookie signature.

[0015] In some aspects, the present solution is directed to systems for synchronizing a random seed value among a plurality of multi-core nodes in a cluster of nodes for generating a cookie signature. The systems may include a cluster of nodes, each node comprising a plurality of cores. The systems may include a master core on a master node of the cluster of nodes, configured to generate a random seed to be synchronized across each core of each node in the cluster of nodes, and store the random seed to memory on the master node accessible by each core in the master node. The systems may include each other node in the cluster, configured to receive, by each master core of each node, the random seed sent by the master core of the master node and store the random seed to memory on each node accessible by each core in the each other node. The systems may include a packet engine on each core of each node in the cluster of nodes, configured to generate a cookie signature based on the random seed responsive to a predetermined timer.

[0016] In some applications of these systems, each other node in the cluster further comprises a receiving core configured to receive the random seed sent by the master core of the master node and steer the random seed to each node's master core. In some applications of these systems, each node in the cluster of nodes is further configured to store, at each core of each node in the cluster of nodes, a current cookie signature as a previous cookie signature, and the generated cookie signature as the current cookie signature. In some applications of these systems the master core of the master node is further configured to generate the random seed to be synchronized across each core of each node in the cluster of nodes, responsive to a second predetermined timer set to expire longer than the predetermined timer. In some applications of these systems the packet engine is further configured to generate an array of cookie signatures.

[0017] In some applications of these systems, the packet engine is further configured to generate, an array of cookie signatures, by using the random seed as an initial seed of a pseudo-random number function to generate a first cookie signature, and generate each successive cookie signature by using a preceding cookie signature as a seed for the pseudo-random number function. In some applications of these systems the packet engine is further configured to generate a cookie by concatenating one or more cookie signatures in the array. In some applications of these systems the packet engine is further configured to use the generated cookie signature as part of a SYN cookie or a HTTP DoS cookie.

[0018] In some applications of these systems, the packet engine is further configured to receive from a client, a SYN request at a first core of a node in the cluster. The packet engine is further configured to respond to the client with a SYN-ACK message comprising a cookie with the cookie signature. The packet engine is further configured to receive from the client, an ACK message at a second core of the node in the cluster, the ACK message comprising a client cookie signature. The packet engine is further configured to accept the ACK message in response to matching the client cookie signature with the cookie signature.

[0019] In some applications of these systems, the packet engine is further configured to store a current cookie signature as a previous cookie signature, and the generated cookie signature as the current cookie signature, and compare the client cookie signature with the current cookie signature and the previous cookie signature. The packet engine is further configured to determine whether a new random seed is stored in a memory accessible by a second core, and store, at the second core, the current cookie signature as the previous cookie signature. The packet engine is further configured to generate a new current cookie signature based on the new random seed in the memory accessible by the second core, and allocate resources in response to matching the client cookie signature with the new current cookie signature.

[0020] The details of various embodiments of the invention are set forth in the accompanying drawings and the description below.

BRIEF DESCRIPTION OF THE FIGURES

[0021] The foregoing and other objects, aspects, features, and advantages of the invention will become more apparent and better understood by referring to the following description taken in conjunction with the accompanying drawings, in which:

[0022] FIG. **1**A is a block diagram of an embodiment of a network environment for a client to access a server via an appliance;

[0023] FIG. **1**B is a block diagram of an embodiment of an environment for delivering a computing environment from a server to a client via an appliance;

[0024] FIG. **1**C is a block diagram of another embodiment of an environment for delivering a computing environment from a server to a client via an appliance;

[0025] FIG. **1**D is a block diagram of another embodiment of an environment for delivering a computing environment from a server to a client via an appliance;

[0026] FIGS. **1**E-**1**H are block diagrams of embodiments of a computing device;

[0027] FIG. **2**A is a block diagram of an embodiment of an appliance for processing communications between a client and a server;

[0028] FIG. **2**B is a block diagram of another embodiment of an appliance for optimizing, accelerating, load-balancing and routing communications between a client and a server;

[0029] FIG. **3** is a block diagram of an embodiment of a client for communicating with a server via the appliance;

[0030] FIG. **4**A is a block diagram of an embodiment of a virtualization environment;

[0031] FIG. **4**B is a block diagram of another embodiment of a virtualization environment;

[0032] FIG. **4**C is a block diagram of an embodiment of a virtualized appliance;

[0033] FIG. **5**A are block diagrams of embodiments of approaches to implementing parallelism in a multi-core system;

[0034] FIG. **5**B is a block diagram of an embodiment of a system utilizing a multi-core system;

[0035] FIG. **5**C is a block diagram of another embodiment of an aspect of a multi-core system;

[0036] FIG. **6** is a block diagram of an embodiment of a cluster system;

[0037] FIG. **7**A is a block diagram of an embodiment of a multi-core system for generating cookie signatures;

[0038] FIG. **7**B is a flow chart of an embodiment of a method of generating and maintaining consistent cookie signatures in a multi-core system;

[0039] FIG. **7**C is a flow chart of an embodiment of a method of using cookie signatures for security in a multi-core system;

[0040] FIG. **8**A is a block diagram of an embodiment of a cluster system for generating cookie signatures;

[0041] FIG. **8**B is a flow chart of an embodiment of a method of generating and synchronizing seeds for cookie generation in a cluster system; and

[0042] FIG. **8**C is a flow chart of an embodiment of a method of using cookie signatures for cluster system.

[0043] The features and advantages of the present invention 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 OF THE INVENTION

[0044] 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:

- **[0045]** 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;
- **[0046]** Section C describes embodiments of systems and methods for accelerating communications between a client and a server;
- **[0047]** Section D describes embodiments of systems and methods for virtualizing an application delivery controller;
- **[0048]** Section E describes embodiments of systems and methods for providing a multi-core architecture and environment;
- **[0049]** Section F describes embodiments of systems and methods for providing a clustered appliance architecture environment;
- **[0050]** Section G describes embodiments of systems and methods for generating cookie signatures for security protection in a multi-core system; and
- **[0051]** Section H describes embodiments of systems and methods for TCP SYN attack protection in clustered systems.

[0052] A. Network and Computing Environment

[0053] 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 FIG. 1A, an embodiment of a network environment is depicted. In brief overview, the network environment comprises one or more clients **102***a*-**102***n* (also generally referred to as local machine(s) **102**, or client(s) **102**) in communication with one or more servers **106***a*-**106***n* (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**.

[0054] Although FIG. **1**A 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.

[0055] 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.

[0056] As shown in FIG. **1**A, the appliance **200**, which also may be referred to as an interface unit **200** or gateway **200**, is shown between the networks **104** and **104'**. In some embodiments, the appliance **200** may be located on network **104**. For example, a branch office of a corporate enterprise may deploy an appliance **200** at the branch office. In other embodiments, the appliance **200** may be located on network **104'**. For example, an appliance **200** may be located at a corporate data center. In yet another embodiment, a plurality of appliances **200** may be deployed on network **104**. In some embodiments, a plurality of appliances **200** may be deployed on network **104'**. In one embodiment, a first appliance **200** communicates with a second appliance **200'**. In other embodiments, the appliance **200** could be a part of any client **102** or server **106** on the same or different network **104**,**104'** as the client **102**. One or more appliances **200** may be located at any point in the network or network communications path between a client **102** and a server **106**.

[0057] In some embodiments, the appliance **200** comprises any of the network devices manufactured by Citrix Systems, Inc. of Ft. Lauderdale Fla., referred to as Citrix NetScaler devices. In other embodiments, the appliance **200** includes any of the product embodiments referred to as WebAccelerator and BigIP manufactured by F5 Networks, Inc. of Seattle, Wash. 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, Calif. In yet another embodiment, the appliance **200** includes any application acceleration and/or security related appliances and/or software manufactured by Cisco Systems, Inc. of San Jose, Calif., such as the Cisco ACE Application Control Engine Module service software and network modules, and Cisco AVS Series Application Velocity System.

[0058] In one embodiment, the system may include multiple, logically-grouped servers **106**. In these embodiments, the logical group of servers may be referred to as a server farm **38**. In some of these embodiments, the serves **106** may be geographically dispersed. In some cases, a farm **38** may be administered as a single entity. In other embodiments, the server farm **38** comprises a plurality of server farms **38**. In one embodiment, the server farm executes one or more applications on behalf of one or more clients **102**.

[0059] 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, Wash.), while one or more of the other servers **106** can operate on according to another type of operating system platform (e.g., Unix or Linux). The servers **106** of each farm **38** do not need to be physically proximate to another server **106** in the same farm **38**. Thus, the group of servers **106** logically grouped as a farm **38** may be interconnected using a wide-area network (WAN) connection or medium-area network (MAN) connection. For example, a farm **38** may include servers **106** physically located in different continents or different regions of a continent, country, state, city, campus, or room. Data transmission speeds between servers **106** in the farm **38** can be increased if the servers **106** are connected using a local-area network (LAN) connection or some form of direct connection.

[0060] Servers **106** may be referred to as a file server, application server, web server, proxy server, or gateway server. In some embodiments, a server **106** may have the capacity to function as either an application server or as a master application server. In one embodiment, a server **106** may include an Active Directory. The clients **102** may also be referred to as client nodes or endpoints. In some embodiments, a client **102** has the capacity to function as both a client node seeking access to applications on a server and as an application server providing access to hosted applications for other clients **102***a*-**102***n*.

[0061] In some embodiments, a client **102** communicates with a server **106**. In one embodiment, the client **102** communicates directly with one of the servers **106** in a farm **38**. In another embodiment, the client **102** executes a program neighborhood application to communicate with a server **106** in a farm **38**. In still another embodiment, the server **106** provides the functionality of a master node. In some embodiments, the client **102** communicates with the server **106** in the farm **38** through a network **104**. Over the network **104**, the client **102** can, for example, request execution of various applications hosted by the servers **106***a*-**106***n* in the farm **38** and receive output of the results of the application execution for display. In some embodiments, only the master node provides the functionality required to identify and provide address information associated with a server **106'** hosting a requested application.

[0062] In one embodiment, the server **106** provides functionality of a web server. In another embodiment, the server **106***a* receives requests from the client **102**, forwards the requests to a second server **106***b* and responds to the request by the client **102** with a response to the request from the server **106***b*. In still another embodiment, the server **106** acquires an enumeration of applications available to the client **102** and address information associated with a server **106** hosting an application identified by the enumeration of applications. In yet another embodiment, the server **106** presents the response to the request to the client **102** using a web interface. In one embodiment, the client **102** communicates directly with the server **106** to access the identified application. In another embodiment, the client **102** receives application output data, such as display data, generated by an execution of the identified application on the server **106**.

[0063] Referring now to FIG. **1**B, 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

[0064] Referring now to FIG. **1**C, 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'**

[0065] In one embodiment, the appliance **205** is a device for accelerating, optimizing or otherwise improving the performance, operation, or quality of service of any type and form of network traffic, such as traffic to and/or from a WAN connection. In some embodiments, the appliance **205** is a performance enhancing proxy. In other embodiments, the appliance **205** is any type and form of WAN optimization or acceleration device, sometimes also referred to as a WAN optimization controller. In one embodiment, the appliance **205** is any of the product embodiments referred to as WANScaler manufactured by Citrix Systems, Inc. of Ft. Lauderdale, Fla. 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, Wash. In another embodiment, the appliance **205** includes any of the WX and WXC WAN acceleration device platforms manufactured by Juniper Networks, Inc. of Sunnyvale, Calif. In some embodiments, the appliance **205** includes any of the steelhead line of WAN optimization appliances manufactured by Riverbed Technology of San Francisco, Calif. In other embodiments, the appliance **205** includes any of the WAN related devices manufactured by Expand Networks Inc. of Roseland, N.J. In one embodiment, the appliance **205** includes any of the WAN related appliances manufactured by Packeteer Inc. of Cupertino, Calif., 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, Calif., such as the Cisco Wide Area Network Application Services software and network modules, and Wide Area Network engine appliances.

[0066] In one embodiment, the appliance **205** provides application and data acceleration services for branch-office or remote offices. In one embodiment, the appliance **205** includes optimization of Wide Area File Services (WAFS). In another embodiment, the appliance **205** accelerates the delivery of files, such as via the Common Internet File System (CIFS) protocol. In other embodiments, the appliance **205** provides caching in memory and/or storage to accelerate delivery of applications and data. In one embodiment, the appliance **205** provides compression of network traffic at any level of the network stack or at any protocol or network layer. In another embodiment, the appliance **205** provides transport layer protocol optimizations, flow control, performance enhancements or modifications and/or management to accelerate delivery of applications and data over a WAN connection. For example, in one embodiment, the appliance **205** provides Transport Control Protocol (TCP) optimizations. In other embodiments, the appliance **205** provides optimizations, flow control, performance enhancements or modifications and/or management for any session or application layer protocol.

[0067] In another embodiment, the appliance **205** encoded any type and form of data or information into custom or standard TCP and/or IP header fields or option fields of network packet to announce presence, functionality or capability to another appliance **205'**. In another embodiment, an appliance **205'** may communicate with another appliance **205'** using data encoded in both TCP and/or IP header fields or options. For example, the appliance may use TCP option(s) or IP header fields or options to communicate one or more parameters to be used by the appliances **205**, **205'** in performing functionality, such as WAN acceleration, or for working in conjunction with each other.

[0068] 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.

[0069] 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 **10** is in communication with a server **106** via network **104**, **104'** and appliance **200**. For example, the client **102** may reside in a remote office of a company, e.g., a branch office, and the server **106** may reside at a corporate data center. The client **102** comprises a client agent **120**, and a computing environment **15**. The computing environment **15** may execute or operate an application that accesses, processes or uses a data file. The computing environment **15**, application and/or data file may be delivered via the appliance **200** and/or the server **106**.

[0070] In some embodiments, the appliance **200** accelerates delivery of a computing environment **15**, or any portion thereof, to a client **102**. In one embodiment, the appliance **200** accelerates the delivery of the computing environment **15** by the application delivery system **190**. For example, the embodiments described herein may be used to accelerate delivery of a streaming application and data file processable by the application from a central corporate data center to a remote user location, such as a branch office of the company. In another embodiment, the appliance **200** accelerates transport layer traffic between a client **102** and a server **106**. The appliance **200** may provide acceleration techniques for accelerating any transport layer payload from a server **106** to a client **102**, such as: 1) transport layer connection pooling, 2) transport layer connection multiplexing, 3) transport control protocol buffering, 4) compression and 5) caching. In some embodiments, the appliance **200** provides load balancing of servers **106** in responding to requests from clients **102**. In other embodiments, the appliance **200** acts as a proxy or access server to provide access to the one or more servers **106**. In another embodiment, the appliance **200** provides a secure virtual private network connection from a first network **104** of the client **102** to the second network **104'** of the server **106**, such as an SSL VPN connection. It yet other embodiments, the appliance **200** provides application firewall security, control and management of the connection and communications between a client **102** and a server **106**.

[0071] In some embodiments, the application delivery management system **190** provides application delivery techniques to deliver a computing environment to a desktop of a user, remote or otherwise, based on a plurality of execution methods and based on any authentication and authorization policies applied via a policy engine **195**. With these techniques, a remote user may obtain a computing environment and access to server stored applications and data files from any network connected device **100**. In one embodiment, the application delivery system **190** may reside or execute on a server **106**. In another embodiment, the application delivery system **190** may reside or execute on a plurality of servers **106**_a_-**106**_n_. 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 **106**_n_ may store or provide the application and data file. In some embodiments, each of the application delivery system **190**, the application, and data file may reside or be located on different servers. In yet another embodiment, any portion of the application delivery system **190** may reside, execute or be stored on or distributed to the appliance **200**, or a plurality of appliances.

[0072] The client **102** may include a computing environment **15** for executing an application that uses or processes a data file. The client **102** via networks **104**, **104'** and appliance **200** may request an application and data file from the server **106**. In one embodiment, the appliance **200** may forward a request from the client **102** to the server **106**. For example, the client **102** may not have the application and data file stored or accessible locally. In response to the request, the application delivery system **190** and/or server **106** may deliver the application and data file to the client **102**. For example, in one embodiment, the server **106** may transmit the application as an application stream to operate in computing environment **15** on client **102**.

[0073] In some embodiments, the application delivery system **190** comprises any portion of the Citrix Access Suite™ by Citrix Systems, Inc., such as the MetaFrame or Citrix Presentation Server™ and/or any of the Microsoft® Windows Terminal Services manufactured by the Microsoft Corporation. In one embodiment, the application delivery system **190** may deliver one or more applications to clients **102** or users via a remote-display protocol or otherwise via remote-based or server-based computing. In another embodiment, the application delivery system **190** may deliver one or more applications to clients or users via steaming of the application.

[0074] In one embodiment, the application delivery system **190** includes a policy engine **195** for controlling and managing the access to, selection of application execution methods and the delivery of applications. In some embodiments, the policy engine **195** determines the one or more applications a user or client **102** may access. In another embodiment, the policy engine **195** determines how the application should be delivered to the user or client **102**, e.g., the method of execution. In some embodiments, the application delivery system **190** provides a plurality of delivery techniques from which to select a method of application execution, such as a server-based computing, streaming or delivering the application locally to the client **120** for local execution.

[0075] In one embodiment, a client **102** requests execution of an application program and the application delivery system **190** comprising a server **106** selects a method of executing the application program. In some embodiments, the server **106** receives credentials from the client **102**. In another embodiment, the server **106** receives a request for an enumeration of available applications from the client **102**. In one embodiment, in response to the request or receipt of credentials, the application delivery system **190** enumerates a plurality of application programs available to the client **102**. The application delivery system **190** receives a request to execute an enumerated application. The application delivery system **190** selects one of a predetermined number of methods for executing the enumerated application, for example, responsive to a policy of a policy engine. The application delivery system **190** may select a method of execution of the application enabling the client **102** to receive application-output data generated by execution of the application program on a server **106**. The application delivery system **190** may select a method of execution of the application enabling the local machine **10** to execute the application program locally after retrieving a plurality of application files comprising the application. In yet another embodiment, the application delivery system **190** may select a method of execution of the application to stream the application via the network **104** to the client **102**.

[0076] A client **102** may execute, operate or otherwise provide an application, which can be any type and/or form of software, program, or executable instructions such as any type and/or form of web browser, web-based client, client-server application, a thin-client computing client, an ActiveX control, or a Java applet, or any other type and/or form of executable instructions capable of executing on client **102**. In some embodiments, the application may be a server-based or a remote-based application executed on behalf of the client **102** on a server **106**. In one embodiments the server **106** may display output to the client **102** using any thin-client or remote-display protocol, such as the Independent Computing Architecture (ICA) protocol manufactured by Citrix Systems, Inc. of Ft. Lauderdale, Fla. or the Remote Desktop Protocol (RDP) manufactured by the Microsoft Corporation of Redmond, Wash. 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.

[0077] In some embodiments, the server **106** or a server farm **38** may be running one or more applications, such as an application providing a thin-client computing or remote display presentation application. In one embodiment, the server **106** or server farm **38** executes as an application, any portion of the Citrix Access Suite™ by Citrix Systems, Inc., such as the MetaFrame or Citrix Presentation Server™, and/or any of the Microsoft® Windows Terminal Services manufactured by the Microsoft Corporation. In one embodiment, the application is an ICA client, developed by Citrix Systems, Inc. of Fort Lauderdale, Fla. In other embodiments, the application includes a Remote Desktop (RDP) client, developed by Microsoft Corporation of Redmond, Wash. 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, Wash., a web or Internet server, or a desktop sharing server, or a collaboration server. In some embodiments, any of the applications may comprise any type of hosted service or products, such as GoToMeeting™ provided by Citrix Online Division, Inc. of Santa Barbara, Calif., WebEx™ provided by WebEx, Inc. of Santa Clara, Calif., or Microsoft Office Live Meeting provided by Microsoft Corporation of Redmond, Wash.

[0078] Still referring to FIG. **1**D, an embodiment of the network environment may include a monitoring server **106**A. The monitoring server **106**A may include any type and form performance monitoring service **198**. The performance monitoring service **198** may include monitoring, measurement and/or management software and/or hardware, including data collection, aggregation, analysis, management and reporting. In one embodiment, the performance monitoring service **198** includes one or more monitoring agents **197**. The monitoring agent **197** includes any software, hardware or combination thereof for performing monitoring, measurement and data collection activities on a device, such as a client **102**, server **106** or an appliance **200**, **205**. In some embodiments, the monitoring agent **197** includes any type and form of script, such as Visual Basic script, or Javascript. In one embodiment, the monitoring agent **197** executes transparently to any application and/or user of the device. In some embodiments, the monitoring agent **197** is installed and operated unobtrusively to the application or client. In yet another embodiment, the monitoring agent **197** is installed and operated without any instrumentation for the application or device.

[0079] In some embodiments, the monitoring agent **197** monitors, measures and collects data on a predetermined frequency. In other embodiments, the monitoring agent **197** monitors, measures and collects data based upon detection of any type and form of event. For example, the monitoring agent **197** may collect data upon detection of a request for a web page or receipt of an HTTP response. In another example, the monitoring agent **197** may collect data upon detection of any user input events, such as a mouse click. The monitoring agent **197** may report or provide any monitored, measured or collected data to the monitoring service **198**. In one embodiment, the monitoring agent **197** transmits information to the monitoring service **198** according to a schedule or a predetermined frequency. In another embodiment, the monitoring agent **197** transmits information to the monitoring service **198** upon detection of an event.

[0080] In some embodiments, the monitoring service **198** and/or monitoring agent **197** performs monitoring and performance measurement of any network resource or network infrastructure element, such as a client, server, server farm, appliance **200**, appliance **205**, or network connection. In one embodiment, the monitoring service **198** and/or monitoring agent **197** performs monitoring and performance measurement of any transport layer connection, such as a TCP or UDP connection. In another embodiment, the monitoring service **198** and/or monitoring agent **197** monitors and measures network latency. In yet one embodiment, the monitoring service **198** and/or monitoring agent **197** monitors and measures bandwidth utilization.

[0081] In other embodiments, the monitoring service **198** and/or monitoring agent **197** monitors and measures end-user response times. In some embodiments, the monitoring service **198** performs monitoring and performance measurement of an application. In another embodiment, the monitoring service **198** and/or monitoring agent **197** performs monitoring and performance measurement of any session or connection to the application. In one embodiment, the monitoring service **198** and/or monitoring agent **197** monitors and measures performance of a browser. In another embodiment, the monitoring service **198** and/or monitoring agent **197** monitors and measures performance of HTTP based transactions. In some embodiments, the monitoring service **198** and/or monitoring agent **197** monitors and measures performance of a Voice over IP (VoIP) application or session. In other embodiments, the monitoring service **198** and/or monitoring agent **197** monitors and measures performance of a remote display protocol application, such as an ICA client or RDP client. In yet another embodiment, the monitoring service **198** and/or monitoring agent **197** monitors and measures performance of any type and form of streaming media. In still a further embodiment, the monitoring service **198** and/or monitoring agent **197** monitors and measures performance of a hosted application or a Software-As-A-Service (SaaS) delivery model.

[0082] In some embodiments, the monitoring service **198** and/or monitoring agent **197** performs monitoring and performance measurement of one or more transactions, requests or responses related to application. In other embodiments, the monitoring service **198** and/or monitoring agent **197** monitors and measures any portion of an application layer stack, such as any .NET or J2EE calls. In one embodiment, the monitoring service **198** and/or monitoring agent **197** monitors and measures database or SQL transactions. In yet another embodiment, the monitoring service **198** and/or monitoring agent **197** monitors and measures any method, function or application programming interface (API) call.

[0083] In one embodiment, the monitoring service **198** and/or monitoring agent **197** performs monitoring and performance measurement of a delivery of application and/or data from a server to a client via one or more appliances, such as appliance **200** and/or appliance **205**. In some embodiments, the monitoring service **198** and/or monitoring agent **197** monitors and measures performance of delivery of a virtualized application. In other embodiments, the monitoring service **198** and/or monitoring agent **197** monitors and measures performance of delivery of a streaming application. In another embodiment, the monitoring service **198** and/or monitoring agent **197** monitors and measures performance of delivery of a desktop application to a client and/or the execution of the desktop application on the client. In another embodiment, the monitoring service **198** and/or monitoring agent **197** monitors and measures performance of a client/server application.

[0084] 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.

[0085] 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.

[0086] In yet further embodiments, the monitoring service **198** and/or monitoring agent **197** measures and monitors measures and monitors any type and form of server metrics. In one embodiment, the monitoring service **198** and/or monitoring agent **197** measures and monitors metrics related to system memory, CPU usage, and disk storage. In another embodiment, the monitoring service **198** and/or monitoring agent **197** measures and monitors metrics related to page faults, such as page faults per second. In other embodiments, the monitoring service **198** and/or monitoring agent **197** measures and monitors round-trip time metrics. In yet another embodiment, the monitoring service **198** and/or monitoring agent **197** measures and monitors metrics related to application crashes, errors and/or hangs.

[0087] 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, Fla. 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, Calif. 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, Calif. In other embodiments, the performance monitoring service **198** and/or monitoring agent **198** includes any portion of the business service management products, such as the BMC Performance Manager and Patrol products, manufactured by BMC Software, Inc. of Houston, Tex.

[0088] The client **102**, server **106**, and appliance **200** may be deployed as and/or executed on any type and form of computing device, such as a computer, network device or appliance capable of communicating on any type and form of network and performing the operations described herein. FIGS. **1**E and **1**F 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. **1**E and **1**F, each computing device **100** includes a central processing unit **101**, and a main memory unit **122**. As shown in FIG. **1**E, a computing device **100** may include a visual display device **124**, a keyboard **126** and/or a pointing device **127**, such as a mouse. Each computing device **100** may also include additional optional elements, such as one or more input/output devices **130***a*-**130***b* (generally referred to using reference numeral **130**), and a cache memory **140** in communication with the central processing unit **101**.

[0089] The central processing unit **101** is any logic circuitry that responds to and processes instructions fetched from the main memory unit **122**. In many embodiments, the central processing unit is provided by a microprocessor unit, such as: those manufactured by Intel Corporation of Mountain View, Calif.; those manufactured by Motorola Corporation of Schaumburg, Ill.; those manufactured by Transmeta Corporation of Santa Clara, Calif.; the RS/6000 processor, those manufactured by International Business Machines of White Plains, N.Y.; or those manufactured by Advanced Micro Devices of Sunnyvale, Calif. The computing device **100** may be based on any of these processors, or any other processor capable of operating as described herein.

[0090] Main memory unit **122** may be one or more memory chips capable of storing data and allowing any storage location to be directly accessed by the microprocessor **101**, such as Static random access memory (SRAM), Burst SRAM or SynchBurst SRAM (BSRAM), Dynamic random access memory (DRAM), Fast Page Mode DRAM (FPM DRAM), Enhanced DRAM (EDRAM), Extended Data Output RAM (EDO RAM), Extended Data Output DRAM (EDO DRAM), Burst Extended Data Output DRAM (BEDO DRAM), Enhanced DRAM (EDRAM), synchronous DRAM (SDRAM), JEDEC SRAM, 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, or any other available memory chips capable of operating as described herein. In the embodiment shown in FIG. **1**E, the processor **101** communicates with main memory **122** via a system bus **150** (described in more detail below). FIG. **1**F 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. **1**F the main memory **122** may be DRDRAM.

[0091] FIG. **1**F depicts an embodiment in which the main processor **101** communicates directly with cache memory **140** via a secondary bus, sometimes referred to as a backside bus. In other embodiments, the main processor **101** communicates with cache memory **140** using the system bus **150**. Cache memory **140** typically has a faster response time than main memory **122** and is typically provided by SRAM, BSRAM, or EDRAM. In the embodiment shown in FIG. **1**F, the processor **101** communicates with various I/O devices **130** via a local system bus **150**. Various busses may be used to connect the central processing unit **101** to any of the I/O devices **130**, including a VESA VL bus, an ISA bus, an EISA bus, a MicroChannel Architecture (MCA) bus, a PCI bus, a PCI-X bus, a PCI-Express bus, or a NuBus. For embodiments in which the I/O device is a video display **124**, the processor **101** may use an Advanced Graphics Port (AGP) to communicate with the display **124**. FIG. **1**F depicts an embodiment of a computer **100** in which the main processor **101** communicates directly with I/O device **130***b* via HyperTransport, Rapid I/O, or InfiniBand. FIG. **1**F also depicts an embodiment in which local busses and direct communication are mixed: the processor **101** communicates with I/O device **130***b* using a local interconnect bus while communicating with I/O device **130***a* directly.

[0092] The computing device **100** may support any suitable installation device **116**, such as a floppy disk drive for receiving floppy disks such as 3.5-inch, 5.25-inch disks or ZIP disks, a CD-ROM drive, a CD-R/RW drive, a DVD-ROM drive, tape drives of various formats, USB device, hard-drive or any other device suitable for installing software and programs such as any client agent **120**, or portion thereof. The computing device **100** may further comprise a storage device **128**, such as one or more hard disk drives or redundant arrays of independent disks, for storing an operating system and other related software, and for storing application software programs such as any program related to the client agent **120**. Optionally, any of the installation devices **116** could also be used as the storage device **128**. Additionally, the operating system and the software can be run from a bootable medium, for example, a bootable CD, such as KNOPPIX®, a bootable CD for GNU/Linux that is available as a GNU/Linux distribution from knoppix.net.

[0093] 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, 56 kb, X.25), broadband connections (e.g., ISDN, Frame Relay, ATM), wireless connections, or some combination of any or all of the above. The network interface **118** may comprise a built-in network adapter, network interface card, PCMCIA network card, card bus network adapter, wireless network adapter, USB network adapter, modem or any other device suitable for interfacing the computing device **100** to any type of network capable of communication and performing the operations described herein. A wide variety of I/O devices **130***a*-**130***n* may be present in the computing device **100**. Input devices include keyboards, mice, trackpads, trackballs, microphones, and drawing tablets. Output devices include video displays, speakers, inkjet printers, laser printers, and dye-sublimation printers. The I/O devices **130** may be controlled by an I/O controller **123** as shown in FIG. **1**E. The I/O controller may control one or more I/O devices such as a keyboard **126** and a pointing device **127**, e.g., a mouse or optical pen. Furthermore, an I/O device may also provide storage **128** and/or an installation medium **116** for the computing device **100**. In still other embodiments, the computing device **100** may provide USB connections to receive handheld USB storage devices such as the USB Flash Drive line of devices manufactured by Twintech Industry, Inc. of Los Alamitos, Calif.

[0094] In some embodiments, the computing device **100** may comprise or be connected to multiple display devices **124***a*-**124***n*, which each may be of the same or different type and/or form. As such, any of the I/O devices **130***a*-**130***n* 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 **124***a*-**124***n* 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 **124***a*-**124***n*. In one embodiment, a video adapter may comprise multiple connectors to interface to multiple display devices **124***a*-**124***n*. 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 **124***a*-**124***n*. In some embodiments, any portion of the operating system of the computing device **100** may be configured for using multiple displays **124***a*-**124***n*. In other embodiments, one or more of the display devices **124***a*-**124***n* may be provided by one or more other computing devices, such as computing devices **100***a* and **100***b* 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 **124***a* 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 **124***a*-**124***n*.

[0095] 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.

[0096] A computing device **100** of the sort depicted in FIGS. **1**E and **1**F typically operate under the control of operating systems, which control scheduling of tasks and access to system resources. The computing device **100** can be running any operating system such as any of the versions of the Microsoft® Windows operating systems, the different releases of the Unix and Linux operating systems, any version of the Mac OS® for Macintosh computers, any embedded operating system, any real-time operating system, any open source operating system, any proprietary operating system, any operating systems for mobile computing devices, or any other operating system capable of running on the computing device and performing the operations described herein. Typical operating systems include: WINDOWS 3.x, WINDOWS 95, WINDOWS 98, WINDOWS 2000, WINDOWS NT 3.51, WINDOWS NT 4.0, WINDOWS CE, and WINDOWS XP, all of which are manufactured by Microsoft Corporation of Redmond, Wash.; MacOS, manufactured by Apple Computer of Cupertino, Calif.; OS/2, manufactured by International Business Machines of Armonk, N.Y.; 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.

[0097] In other embodiments, the computing device **100** may have different processors, operating systems, and input devices consistent with the device. For example, in one embodiment the computer **100** is a Treo 180, 270, 1060, 600 or 650 smart phone manufactured by Palm, Inc. In this embodiment, the Treo smart phone is operated under the control of the PalmOS operating system and includes a stylus input device as well as a five-way navigator device. Moreover, the computing device **100** can be any workstation, desktop computer, laptop or notebook computer, server, handheld computer, mobile telephone, any other computer, or other form of computing or telecommunications device that is capable of communication and that has sufficient processor power and memory capacity to perform the operations described herein.

[0098] As shown in FIG. **1**G, the computing device **100** may comprise multiple processors and may provide functionality for simultaneous execution of instructions or for simultaneous execution of one instruction on more than one piece of data. In some embodiments, the computing device **100** may comprise a parallel processor with one or more cores. In one of these embodiments, the computing device **100** is a shared memory parallel device, with multiple processors and/or multiple processor cores, accessing all available memory as a single global address space. In another of these embodiments, the computing device **100** is a distributed memory parallel device with multiple processors each accessing local memory only. In still another of these embodiments, the computing device **100** has both some memory which is shared and some memory which can only be accessed by particular processors or subsets of processors. In still even another of these embodiments, the computing device **100**, such as a multi-core microprocessor, combines two or more independent processors into a single package, often a single integrated circuit (IC). In yet another of these embodiments, the computing device **100** includes a chip having a CELL BROADBAND ENGINE architecture and including a Power processor element and a plurality of synergistic processing elements, the Power processor element and the plurality of synergistic processing elements linked together by an internal high speed bus, which may be referred to as an element interconnect bus.

[0099] 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.

[0100] In some embodiments, the computing device **100** may comprise a graphics processing unit. In one of these embodiments, depicted in FIG. **1**H, the computing device **100** includes at least one central processing unit **101** and at least one graphics processing unit. In another of these embodiments, the computing device **100** includes at least one parallel processing unit and at least one graphics processing unit. In still another of these embodiments, the computing device **100** includes a plurality of processing units of any type, one of the plurality of processing units comprising a graphics processing unit.

[0101] In some embodiments, a first computing device **100***a* executes an application on behalf of a user of a client computing device **100***b*. In other embodiments, a computing device **100***a* executes a virtual machine, which provides an execution session within which applications execute on behalf of a user or a client computing devices **100***b*. In one of these embodiments, the execution session is a hosted desktop session. In another of these embodiments, the computing device **100** executes a terminal services session. The terminal services session may provide a hosted desktop environment. In still another of these embodiments, the execution session provides access to a computing environment, which may comprise one or more of: an application, a plurality of applications, a desktop application, and a desktop session in which one or more applications may execute.

[0102] B. Appliance Architecture

[0103] FIG. **2**A illustrates an example embodiment of the appliance **200**. The architecture of the appliance **200** in FIG. **2**A is provided by way of illustration only and is not intended to be limiting. As shown in FIG. **2**, appliance **200** comprises a hardware layer **206** and a software layer divided into a user space **202** and a kernel space **204**.

[0104] 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. **1**E and **1**F. 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.

[0105] 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.

[0106] 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. **1**E and **1**F. 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.

[0107] The operating system of appliance **200** allocates, manages, or otherwise segregates the available system memory into kernel space **204** and user space **204**. In example software architecture **200**, the operating system may be any type and/or form of Unix operating system although the invention is not so limited. As such, the appliance **200** can be running any operating system such as any of the versions of the Microsoft® Windows operating systems, the different releases of the Unix and Linux operating systems, any version of the Mac OS® for Macintosh computers, any embedded operating system, any network operating system, any real-time operating system, any open source operating system, any proprietary operating system, any operating systems for mobile computing devices or network devices, or any other operating system capable of running on the appliance **200** and performing the operations described herein.

[0108] The kernel space **204** is reserved for running the kernel **230**, including any device drivers, kernel extensions or other kernel related software. As known to those skilled in the art, the kernel **230** is the core of the operating system, and provides access, control, and management of resources and hardware-related elements of the application **104**. In accordance with an embodiment of the appliance **200**, the kernel space **204** also includes a number of network services or processes working in conjunction with a cache manager **232**, sometimes also referred to as the integrated cache, the benefits of which are described in detail further herein. Additionally, the embodiment of the kernel **230** will depend on the embodiment of the operating system installed, configured, or otherwise used by the device **200**.

[0109] In one embodiment, the device **200** comprises one network stack **267**, such as a TCP/IP based stack, for communicating with the client **102** and/or the server **106**. In one embodiment, the network stack **267** is used to communicate with a first network, such as network **108**, and a second network **110**. In some embodiments, the device **200** terminates a first transport layer connection, such as a TCP connection of a client **102**, and establishes a second transport layer connection to a server **106** for use by the client **102**, e.g., the second transport layer connection is terminated at the appliance **200** and the server **106**. The first and second transport layer connections may be established via a single network stack **267**. In other embodiments, the device **200** may comprise multiple network stacks, for example **267** and **267'**, and the first transport layer connection may be established or terminated at one network stack **267**, and the second transport layer connection on the second network stack **267'**. For example, one network stack may be for receiving and transmitting network packet on a first network, and another network stack for receiving and transmitting network packets on a second network. In one embodiment, the network stack **267** comprises a buffer **243** for queuing one or more network packets for transmission by the appliance **200**.

[0110] 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**.

[0111] 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.

[0112] 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.

[0113] 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).

[0114] 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.

[0115] 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 **102*a*-102*n***, server **106*a*-106*n***, 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 **102*a*-102*n*** and a server **106*a*-106*n***. 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**.

[0116] 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 **102*a*-102*n*** and servers **106*a*-106*n*** any TCP/IP based protocol, including Messaging Application Programming Interface (MAPI) (email), File Transfer Protocol (FTP), HyperText Transfer Protocol (HTTP), Common Internet File System (CIFS) protocol (file transfer), Independent Computing Architecture (ICA) protocol, Remote Desktop Protocol (RDP), Wireless Application Protocol (WAP), Mobile IP protocol, and Voice Over IP (VoIP) protocol. In other embodiments, multi-protocol compression engine **238** provides compression of Hypertext Markup Language (HTML) based protocols and in some embodiments, provides compression of any markup languages, such as the Extensible Markup Language (XML). In one embodiment, the multi-protocol compression engine **238** provides compression of any high-performance protocol, such as any protocol designed for appliance **200** to appliance **200** communications. In another embodiment, the multi-protocol compression engine **238** compresses any payload of or any communication using a modified transport control protocol, such as Transaction TCP (T/TCP), TCP with selection acknowledgements (TCP-SACK), TCP with large windows (TCP-LW), a congestion prediction protocol such as the TCP-Vegas protocol, and a TCP spoofing protocol.

[0117] 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.

[0118] 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.

[0119] 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 100 ms, 50 ms or 25 ms. 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.

[0120] 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**.

[0121] 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**.

[0122] 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.

[0123] 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 **270***a*-**270***n* (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 **275***a*-**275***n* (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**.

[0124] The vServer **275** may comprise software, hardware or any combination of software and hardware. The vServer **275** may comprise any type and form of program, service, task, process or executable instructions operating in user mode **202**, kernel mode **204** or any combination thereof in the appliance **200**. The vServer **275** includes any logic, functions, rules, or operations to perform any embodiments of the techniques described herein, such as SSL VPN **280**, switching/load balancing **284**, Domain Name Service resolution **286**, acceleration **288** and an application firewall **290**. In some embodiments, the vServer **275** establishes a connection to a service **270** of a server **106**. The service **275** may comprise any program, application, process, task or set of executable instructions capable of connecting to and communicating to the appliance **200**, client **102** or vServer **275**. For example, the service **275** may comprise a web server, http server, ftp, email or database server. In some embodiments, the service **270** is a daemon process or network driver for listening, receiving and/or sending communications for an application, such as email, database or an enterprise application. In some embodiments, the service **270** may communicate on a specific IP address, or IP address and port.

[0125] In some embodiments, the vServer **275** applies one or more policies of the policy engine **236** to network communications between the client **102** and server **106**. In one embodiment, the policies are associated with a vServer **275**. In another embodiment, the policies are based on a user, or a group of users. In yet another embodiment, a policy is global and applies to one or more vServers **275***a*-**275***n*, and any user or group of users communicating via the appliance **200**. In some embodiments, the policies of the policy engine have conditions upon which the policy is applied based on any content of the communication, such as internet protocol address, port, protocol type, header or fields in a packet, or the context of the communication, such as user, group of the user, vServer **275**, transport layer connection, and/or identification or attributes of the client **102** or server **106**.

[0126] In other embodiments, the appliance **200** communicates or interfaces with the policy engine **236** to determine authentication and/or authorization of a remote user or a remote client **102** to access the computing environment **15**, application, and/or data file from a server **106**. In another embodiment, the appliance **200** communicates or interfaces with the policy engine **236** to determine authentication and/or authorization of a remote user or a remote client **102** to have the application delivery system **190** deliver one or more of the computing environment **15**, application, and/or data file. In yet another embodiment, the appliance **200** establishes a VPN or SSL VPN connection based on the policy engine's **236** authentication and/or authorization of a remote user or a remote client **102**. In one embodiment, the appliance **200** controls the flow of network traffic and communication sessions based on policies of the policy engine **236**. For example, the appliance **200** may control the access to a computing environment **15**, application or data file based on the policy engine **236**.

[0127] In some embodiments, the vServer **275** establishes a transport layer connection, such as a TCP or UDP connection with a client **102** via the client agent **120**. In one embodiment, the vServer **275** listens for and receives communications from the client **102**. In other embodiments, the vServer **275** establishes a transport layer connection, such as a TCP or UDP connection with a client server **106**. In one embodiment, the vServer **275** establishes the transport layer connection to an internet protocol address and port of a server **270** running on the server **106**. In another embodiment, the vServer **275** associates a first transport layer connection to a client **102** with a second transport layer connection to the server **106**. In some embodiments, a vServer **275** establishes a pool of transport layer connections to a server **106** and multiplexes client requests via the pooled transport layer connections.

[0128] In some embodiments, the appliance **200** provides a SSL VPN connection **280** between a client **102** and a server **106**. For example, a client **102** on a first network **102** requests to establish a connection to a server **106** on a second network **104'**. In some embodiments, the second network **104'** is not routable from the first network **104**. In other embodiments, the client **102** is on a public network **104** and the server **106** is on a private network **104'**, such as a corporate network. In one embodiment, the client agent **120** intercepts communications of the client **102** on the first network **104**, encrypts the communications, and transmits the communications via a first transport layer connection to the appliance **200**. The appliance **200** associates the first transport layer connection on the first network **104** to a second transport layer connection to the server **106** on the second network **104**. The appliance **200** receives the intercepted communication from the client agent **102**, decrypts the communications, and transmits the communication to the server **106** on the second network **104** via the second transport layer connection. The second transport layer connection may be a pooled transport layer connection. As such, the appliance **200** provides an end-to-end secure transport layer connection for the client **102** between the two networks **104**, **104'**.

[0129] In one embodiment, the appliance **200** hosts an intranet internet protocol or IntranetIP **282** address of the client **102** on the virtual private network **104**. The client **102** has a local network identifier, such as an internet protocol (IP) address and/or host name on the first network **104**. When connected to the second network **104'** via the appliance **200**, the appliance **200** establishes, assigns or otherwise provides an IntranetIP address **282**, which is a network identifier, such as IP address and/or host name, for the client **102** on the second network **104'**. The appliance **200** listens for and receives on the second or private network **104'** for any communications directed towards the client **102** using the client's established IntranetIP **282**. In one embodiment, the appliance **200** acts as or on behalf of the client **102** on the second private network **104**. For example, in another embodiment, a vServer **275** listens for and responds to communications to the IntranetIP **282** of the client **102**. In some embodiments, if a computing device **100** on the second network **104'** transmits a request, the appliance **200** processes the request as if it were the client **102**. For example, the appliance **200** may respond to a ping to the client's IntranetIP **282**. In another example, the appliance may establish a connection, such as a TCP or UDP connection, with computing device **100** on the second network **104** requesting a connection with the client's IntranetIP **282**.

[0130] In some embodiments, the appliance **200** provides one or more of the following acceleration techniques **288** to communications between the client **102** and server **106**: 1) compression; 2) decompression; 3) Transmission Control Protocol pooling; 4) Transmission Control Protocol multiplexing; 5) Transmission Control Protocol buffering; and 6) caching. In one embodiment, the appliance **200** relieves servers **106** of much of the processing load caused by repeatedly opening and closing transport layers connections to clients **102** by opening one or more transport layer connections with each server **106** and maintaining these connections to allow repeated data accesses by clients via the Internet. This technique is referred to herein as "connection pooling".

[0131] In some embodiments, in order to seamlessly splice communications from a client **102** to a server **106** via a pooled transport layer connection, the appliance **200** translates or multiplexes communications by modifying sequence number and acknowledgment numbers at the transport layer protocol level. This is referred to as "connection multiplexing". In some embodiments, no application layer protocol interaction is required. For example, in the case of an in-bound packet (that is, a packet received from a client **102**), the source network address of the packet is changed to that of an output port of appliance **200**, and the destination network address is changed to that of the intended server. In the case of an outbound packet (that is, one received from a server **106**), the source network address is changed from that of the server **106** to that of an output port of appliance **200** and the destination address is changed from that of appliance **200** to that of the requesting client **102**. The sequence numbers and acknowledgment numbers of the packet are also translated to sequence numbers and acknowledgement numbers expected by the client **102** on the appliance's **200** transport layer connection to the client **102**. In some embodiments, the packet checksum of the transport layer protocol is recalculated to account for these translations.

[0132] 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**.

[0133] 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**.

[0134] 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.

[0135] In still yet other embodiments, the application firewall **290** protects any confidential information contained in the network communication. The application firewall **290** may inspect or analyze any network communication in accordance with the rules or polices of the engine **236** to identify any confidential information in any field of the network packet. In some embodiments, the application firewall **290** identifies in the network communication one or more occurrences of a credit card number, password, social security number, name, patient code, contact information, and age. The encoded portion of the network communication may comprise these occurrences or the confidential information. Based on these occurrences, in one embodiment, the application firewall **290** may take a policy action on the network communication, such as prevent transmission of the network communication. In another embodiment, the application firewall **290** may rewrite, remove or otherwise mask such identified occurrence or confidential information.

[0136] 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 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**.

[0137] In other embodiments, the appliance **200** executes the monitoring agent **197**. In one embodiment, the monitoring agent **197** measures and monitors the performance of any application, program, process, service, task or thread executing on the appliance **200**. For example, the monitoring agent **197** may monitor and measure performance and operation of vServers **275**A-**275**N. In another embodiment, the monitoring agent **197** measures and monitors the performance of any transport layer connections of the appliance **200**. In some embodiments, the monitoring agent **197** measures and monitors the performance of any user sessions traversing the appliance **200**. In one embodiment, the monitoring agent **197** measures and monitors the performance of any virtual private network connections and/or sessions traversing the appliance **200**, such an SSL VPN session. In still further embodiments, the monitoring agent **197** measures and monitors the memory, CPU and disk usage and performance of the appliance **200**. In yet another embodiment, the monitoring agent **197** measures and monitors the performance of any acceleration technique **288** performed by the appliance **200**, such as SSL offloading, connection pooling and multiplexing, caching, and compression. In some embodiments, the monitoring agent **197** measures and monitors the performance of any load balancing and/or content switching **284** performed by the appliance **200**. In other embodiments, the monitoring agent **197** measures and monitors the performance of application firewall **290** protection and processing performed by the appliance **200**.

[0138] C. Client Agent

[0139] Referring now to FIG. **3**, an embodiment of the client agent **120** is depicted. The client **102** includes a client agent **120** for establishing and exchanging communications with the appliance **200** and/or server **106** via a network **104**. In brief overview, the client **102** operates on computing device **100** having an operating system with a kernel mode **302** and a user mode **303**, and a network stack **310** with one or more layers **310***a*-**310***b*. The client **102** may have installed and/or execute one or more applications. In some embodiments, one or more applications may communicate via the network stack **310** to a network **104**. One of the applications, such as a web browser, may also include a first program **322**. For example, the first program **322** may be used in some embodiments to install and/or execute the client agent **120**, or any portion thereof. The client agent **120** includes an interception mechanism, or interceptor **350**, for intercepting network communications from the network stack **310** from the one or more applications.

[0140] The network stack **310** of the client **102** may comprise any type and form of software, or hardware, or any combinations thereof, for providing connectivity to and communications with a network. In one embodiment, the network stack **310** comprises a software implementation for a network protocol suite. The network stack **310** may comprise one or more network layers, such as any networks layers of the Open Systems Interconnection (OSI) communications model as those skilled in the art recognize and appreciate. As such, the network stack **310** may comprise any type and form of protocols for any of the following layers of the OSI model: 1) physical link layer, 2) data link layer, 3) network layer, 4) transport layer, 5) session layer, 6) presentation layer, and 7) application layer. In one embodiment, the network stack **310** may comprise a transport control protocol (TCP) over the network layer protocol of the internet protocol (IP), generally referred to as TCP/IP. In some embodiments, the TCP/IP protocol may be carried over the Ethernet protocol, which may comprise any of the family of IEEE wide-area-network (WAN) or local-area-network (LAN) protocols, such as those protocols covered by the IEEE 802.3. In some embodiments, the network stack **310** comprises any type and form of a wireless protocol, such as IEEE 802.11 and/or mobile internet protocol.

[0141] In view of a TCP/IP based network, any TCP/IP based protocol may be used, including Messaging Application Programming Interface (MAPI) (email), File Transfer Protocol (FTP), HyperText Transfer Protocol (HTTP), Common Internet File System (CIFS) protocol (file transfer), Independent Computing Architecture (ICA) protocol, Remote Desktop Protocol (RDP), Wireless Application Protocol (WAP), Mobile IP protocol, and Voice Over IP (VoIP) protocol. In another embodiment, the network stack **310** comprises any type and form of transport control protocol, such as a modified transport control protocol, for example a Transaction TCP (T/TCP), TCP with selection acknowledgements (TCP-SACK), TCP with large windows (TCP-LW), a congestion prediction protocol such as the TCP-Vegas protocol, and a TCP spoofing protocol. In other embodiments, any type and form of user datagram protocol (UDP), such as UDP over IP, may be used by the network stack **310**, such as for voice communications or real-time data communications.

[0142] 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**.

[0143] 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.

[0144] 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 **310*a*** of the network stack may provide user-mode access to the network stack **310** to an application while a second portion **310*a*** of the network stack **310** provides access to a network. In some embodiments, a first portion **310*a*** 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 **310*b*** of the network stack **310** comprises one or more lower layers, such as any of layers 1-4. Each of the first portion **310*a*** and second portion **310*b*** 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**.

[0145] The interceptor **350** may comprise software, hardware, or any combination of software and hardware. In one embodiment, the interceptor **350** intercept a network communication at any point in the network stack **310**, and redirects or transmits the network communication to a destination desired, managed or controlled by the interceptor **350** or client agent **120**. For example, the interceptor **350** may intercept a network communication of a network stack **310** of a first network and transmit the network communication to the appliance **200** for transmission on a second network **104**. In some embodiments, the interceptor **350** comprises any type interceptor **350** comprises a driver, such as a network driver constructed and designed to interface and work with the network stack **310**. In some embodiments, the client agent **120** and/or interceptor **350** operates at one or more layers of the network stack **310**, such as at the transport layer. In one embodiment, the interceptor **350** comprises a filter driver, hooking mechanism, or any form and type of suitable network driver interface that interfaces to the transport layer of the network stack, such as via the transport driver interface (TDI). In some embodiments, the interceptor **350** interfaces to a first protocol layer, such as the transport layer and another protocol layer, such as any layer above the transport protocol layer, for example, an application protocol layer. In one embodiment, the interceptor **350** may comprise a driver complying with the Network Driver Interface Specification (NDIS), or a NDIS driver. In another embodiment, the interceptor **350** may comprise a mini-filter or a mini-port driver. In one embodiment, the interceptor **350**, or portion thereof, operates in kernel-mode **202**. In another embodiment, the interceptor **350**, or portion thereof, operates in user-mode **203**. In some embodiments, a portion of the interceptor **350** operates in kernel-mode **202** while another portion of the interceptor **350** operates in user-mode **203**. In other embodiments, the client agent **120** operates in user-mode **203** but interfaces via the interceptor **350** to a kernel-mode driver, process, service, task or portion of the operating system, such as to obtain a kernel-level data structure **225**. In further embodiments, the interceptor **350** is a user-mode application or program, such as application.

[0146] 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**.

[0147] 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**.

[0148] 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'**.

[0149] The client agent **120** and/or interceptor **350** may operate at or interface with a protocol layer in a manner transparent to any other protocol layer of the network stack **310**. For example, in one embodiment, the interceptor **350** operates or interfaces with the transport layer of the network stack **310** transparently to any protocol layer below the transport layer, such as the network layer, and any protocol layer above the transport layer, such as the session, presentation or application layer protocols. This allows the other protocol layers of the network stack **310** to operate as desired and without modification for using the interceptor **350**. As such, the client agent **120** and/or interceptor **350** can interface with the transport layer to secure, optimize, accelerate, route or load-balance any communications provided via any protocol carried by the transport layer, such as any application layer protocol over TCP/IP.

[0150] Furthermore, the client agent **120** and/or interceptor may operate at or interface with the network stack **310** in a manner transparent to any application, a user of the client **102**, and any other computing device, such as a server, in communications with the client **102**. The client agent **120** and/or interceptor **350** may be installed and/or executed on the client **102** in a manner without modification of an application. In some embodiments, the user of the client **102** or a computing device in communications with the client **102** are not aware of the existence, execution or operation of the client agent **120** and/or interceptor **350**. As such, in some embodiments, the client agent **120** and/or interceptor **350** is installed, executed, and/or operated transparently to an application, user of the client **102**, another computing device, such as a server, or any of the protocol layers above and/or below the protocol layer interfaced to by the interceptor **350**.

[0151] The client agent **120** includes an acceleration program **302**, a streaming client **306**, a collection agent **304**, and/or monitoring agent **197**. In one embodiment, the client agent **120** comprises an Independent Computing Architecture (ICA) client, or any portion thereof, developed by Citrix Systems, Inc. of Fort Lauderdale, Fla., and is also referred to as an ICA client. In some embodiments, the client **120** comprises an application streaming client **306** for streaming an application from a server **106** to a client **102**. In some embodiments, the client agent **120** comprises an acceleration program **302** for accelerating communications between client **102** and server **106**. In another embodiment, the client agent **120** includes a collection agent **304** for performing end-point detection/scanning and collecting end-point information for the appliance **200** and/or server **106**.

[0152] In some embodiments, the acceleration program **302** comprises a client-side acceleration program for performing one or more acceleration techniques to accelerate, enhance or otherwise improve a client's communications with and/or access to a server **106**, such as accessing an application provided by a server **106**. The logic, functions, and/or operations of the executable instructions of the acceleration program **302** may perform one or more of the following acceleration techniques: 1) multi-protocol compression, 2) transport control protocol pooling, 3) transport control protocol multiplexing, 4) transport control protocol buffering, and 5) caching via a cache manager. Additionally, the acceleration program **302** may perform encryption and/or decryption of any communications received and/or transmitted by the client **102**. In some embodiments, the acceleration program **302** performs one or more of the acceleration techniques in an integrated manner or fashion. Additionally, the acceleration program **302** can perform compression on any of the protocols, or multiple-protocols, carried as a payload of a network packet of the transport layer protocol.

[0153] 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.

[0154] 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**.

[0155] In one embodiment, the collection agent **304** comprises an end-point detection and scanning mechanism, which identifies and determines one or more attributes or characteristics of the client. For example, the collection agent **304** may identify and determine any one or more of the following client-side attributes: 1) the operating system an/or a version of an operating system, 2) a service pack of the operating system, 3) a running service, 4) a running process, and 5) a file. The collection agent **304** may also identify and determine the presence or versions of any one or more of the following on the client: 1) antivirus software, 2) personal firewall software, 3) anti-spam software, and 4) internet security software. The policy engine **236** may have one or more policies based on any one or more of the attributes or characteristics of the client or client-side attributes.

[0156] 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.

[0157] 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**.

[0158] In some embodiments and still referring to FIG. **3**, a first program **322** may be used to install and/or execute the client agent **120**, or portion thereof, such as the interceptor **350**, automatically, silently, transparently, or otherwise. In one embodiment, the first program **322** comprises a plugin component, such an ActiveX control or Java control or script that is loaded into and executed by an application. For example, the first program comprises an ActiveX control loaded and run by a web browser application, such as in the memory space or context of the application. In another embodiment, the first program **322** comprises a set of executable instructions loaded into and run by the application, such as a browser. In one embodiment, the first program **322** comprises a designed and constructed program to install the client agent **120**. In some embodiments, the first program **322** obtains, downloads, or receives the client agent **120** via the network from another computing device. In another embodiment, the first program **322** is an installer program or a plug and play manager for installing programs, such as network drivers, on the operating system of the client **102**.

[0159] D. Systems and Methods for Providing Virtualized Application Delivery Controller

[0160] Referring now to FIG. **4**A, a block diagram depicts one embodiment of a virtualization environment **400**. In brief overview, a computing device **100** includes a hypervisor layer, a virtualization layer, and a hardware layer. The hypervisor layer includes a hypervisor **401** (also referred to as a virtualization manager) that allocates and manages access to a number of physical resources in the hardware layer (e.g., the processor(s) **421**, and disk(s) **428**) by at least one virtual machine executing in the virtualization layer. The virtualization layer includes at least one operating system **410** and a plurality of virtual resources allocated to the at least one operating system **410**. Virtual resources may include, without limitation, a plurality of virtual processors **432***a*, **432***b*, **432***c* (generally **432**), and virtual disks **442***a*, **442***b*, **442***c* (generally **442**), as well as virtual resources such as virtual memory and virtual network interfaces. The plurality of virtual resources and the operating system **410** may be referred to as a virtual machine **406**. A virtual machine **406** may include a control operating system **405** in communication with the hypervisor **401** and used to execute applications for managing and configuring other virtual machines on the computing device **100**.

[0161] 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 **410***a*, **410***b* (generally **410**). In some embodiments, a computing device **100** executes one or more types of hypervisors. In these embodiments, hypervisors may be used to emulate virtual hardware, partition physical hardware, virtualize physical hardware, and execute virtual machines that provide access to computing environments. Hypervisors may include those manufactured by VMWare, Inc., of Palo Alto, Calif.; the XEN hypervisor, an open source product whose development is overseen by the open source Xen.org community; HyperV, VirtualServer or virtual PC hypervisors provided by Microsoft, or others. In some embodiments, a computing device **100** executing a hypervisor that creates a virtual machine platform on which guest operating systems may execute is referred to as a host server. In one of these embodiments, for example, the computing device **100** is a XEN SERVER provided by Citrix Systems, Inc., of Fort Lauderdale, Fla.

[0162] In some embodiments, a hypervisor **401** executes within an operating system executing on a computing device. In one of these embodiments, a computing device executing an operating system and a hypervisor **401** may be said to have a host operating system (the operating system executing on the computing device), and a guest operating system (an operating system executing within a computing resource partition provided by the hypervisor **401**). In other embodiments, a hypervisor **401** interacts directly with hardware on a computing device, instead of executing on a host operating system. In one of these embodiments, the hypervisor **401** may be said to be executing on "bare metal," referring to the hardware comprising the computing device.

[0163] In some embodiments, a hypervisor **401** may create a virtual machine **406***a-c* (generally **406**) in which an operating system **410** executes. In one of these embodiments, for example, the hypervisor **401** loads a virtual machine image to create a virtual machine **406**. In another of these embodiments, the hypervisor **401** executes an operating system **410** within the virtual machine **406**. In still another of these embodiments, the virtual machine **406** executes an operating system **410**.

[0164] In some embodiments, the hypervisor **401** controls processor scheduling and memory partitioning for a virtual machine **406** executing on the computing device **100**. In one of these embodiments, the hypervisor **401** controls the execution of at least one virtual machine **406**. In another of these embodiments, the hypervisor **401** presents at least one virtual machine **406** with an abstraction of at least one hardware resource provided by the computing device **100**. In other embodiments, the hypervisor **401** controls whether and how physical processor capabilities are presented to the virtual machine **406**.

[0165] A control operating system **405** may execute at least one application for managing and configuring the guest operating systems. In one embodiment, the control operating system **405** may execute an administrative application, such as an application including a user interface providing administrators with access to functionality for managing the execution of a virtual machine, including functionality for executing a virtual machine, terminating an execution of a virtual machine, or identifying a type of physical resource for allocation to the virtual machine. In another embodiment, the hypervisor **401** executes the control operating system **405** within a virtual machine **406** created by the hypervisor **401**. In still another embodiment, the control operating system **405** executes in a virtual machine **406** that is authorized to directly access physical resources on the computing device **100**. In some embodiments, a control operating system **405***a* on a computing device **100***a* may exchange data with a control operating system **405***b* on a computing device **100***b*, via communications between a hypervisor **401***a* and a hypervisor **401***b*. In this way, one or more computing devices **100** may exchange data with one or more of the other computing devices **100** regarding processors and other physical resources available in a pool of resources. In one of these embodiments, this functionality allows a hypervisor to manage a pool of resources distributed across a plurality of physical computing devices. In another of these embodiments, multiple hypervisors manage one or more of the guest operating systems executed on one of the computing devices **100**.

[0166] In one embodiment, the control operating system **405** executes in a virtual machine **406** that is authorized to interact with at least one guest operating system **410**. In another embodiment, a guest operating system **410** communicates with the control operating system **405** via the hypervisor **401** in order to request access to a disk or a network. In still another embodiment, the guest operating system **410** and the control operating system **405** may communicate via a communication channel established by the hypervisor **401**, such as, for example, via a plurality of shared memory pages made available by the hypervisor **401**.

[0167] In some embodiments, the control operating system **405** includes a network back-end driver for communicating directly with networking hardware provided by the computing device **100**. In one of these embodiments, the network back-end driver processes at least one virtual machine request from at least one guest operating system **110**. In other embodiments, the control operating system **405** includes a block back-end driver for communicating with a storage element on the computing device **100**. In one of these embodiments, the block back-end driver reads and writes data from the storage element based upon at least one request received from a guest operating system **410**.

[0168] 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 **100***b*), or managing virtual machines **406***b*, **406***c* 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**.

[0169] 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'**.

[0170] In one embodiment, the guest operating system **410**, in conjunction with the virtual machine on which it executes, forms a fully-virtualized virtual machine which is not aware that it is a virtual machine; such a machine may be referred to as a "Domain U HVM (Hardware Virtual Machine) virtual machine". In another embodiment, a fully-virtualized machine includes software emulating a Basic Input/Output System (BIOS) in order to execute an operating system within the fully-virtualized machine. In still another embodiment, a fully-virtualized machine may include a driver that provides functionality by communicating with the hypervisor **401**. In such an embodiment, the driver may be aware that it executes within a virtualized environment. In another embodiment, the guest operating system **410**, in conjunction with the virtual machine on which it executes, forms a paravirtualized virtual machine, which is aware that it is a virtual machine; such a machine may be referred to as a "Domain U PV virtual machine". In another embodiment, a paravirtualized machine includes additional drivers that a fully-virtualized machine does not include. In still another embodiment, the paravirtualized machine includes the network back-end driver and the block back-end driver included in a control operating system **405**, as described above.

[0171] Referring now to FIG. 4B, a block diagram depicts one embodiment of a plurality of networked computing devices in a system in which at least one physical host executes a virtual machine. In brief overview, the system includes a management component **404** and a hypervisor **401**. The system includes a plurality of computing devices **100**, a plurality of virtual machines **406**, a plurality of hypervisors **401**, a plurality of management components referred to variously as tools stacks **404** or management components **404**, and a physical resource **421**, **428**. The plurality of physical machines **100** may each be provided as computing devices **100**, described above in connection with FIGS. **1**E-**1**H and **4**A.

[0172] In greater detail, a physical disk **428** is provided by a computing device **100** and stores at least a portion of a virtual disk **442**. In some embodiments, a virtual disk **442** is associated with a plurality of physical disks **428**. In one of these embodiments, one or more computing devices **100** may exchange data with one or more of the other computing devices **100** regarding processors and other physical resources available in a pool of resources, allowing a hypervisor to manage a pool of resources distributed across a plurality of physical computing devices. In some embodiments, a computing device **100** on which a virtual machine **406** executes is referred to as a physical host **100** or as a host machine **100**.

[0173] The hypervisor executes on a processor on the computing device **100**. The hypervisor allocates, to a virtual disk, an amount of access to the physical disk. In one embodiment, the hypervisor **401** allocates an amount of space on the physical disk. In another embodiment, the hypervisor **401** allocates a plurality of pages on the physical disk. In some embodiments, the hypervisor provisions the virtual disk **442** as part of a process of initializing and executing a virtual machine **450**.

[0174] In one embodiment, the management component **404***a* is referred to as a pool management component **404***a*. In another embodiment, a management operating system **405***a*, which may be referred to as a control operating system **405***a*, 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. **4**A. In other embodiments, the management component **404** provides a user interface for receiving, from a user such as an administrator, an identification of a virtual machine **406** to provision and/or execute. In still other embodiments, the management component **404** provides a user interface for receiving, from a user such as an administrator, the request for migration of a virtual machine **406***b* from one physical machine **100** to another. In further embodiments, the management component **404***a* identifies a computing device **100***b* on which to execute a requested virtual machine **406***d* and instructs the hypervisor **401***b* on the identified computing device **100***b* to execute the identified virtual machine; such a management component may be referred to as a pool management component.

[0175] Referring now to FIG. **4**C, embodiments of a virtual application delivery controller or virtual appliance **450** are depicted. In brief overview, any of the functionality and/or embodiments of the appliance **200** (e.g., an application delivery controller) described above in connection with FIGS. **2**A and **2**B may be deployed in any embodiment of the virtualized environment described above in connection with FIGS. **4**A and **4**B. Instead of the functionality of the application delivery controller being deployed in the form of an appliance **200**, such functionality may be deployed in a virtualized environment **400** on any computing device **100**, such as a client **102**, server **106** or appliance **200**.

[0176] Referring now to FIG. **4**C, 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. **2**A and **2**B, the virtual appliance **450** may provide functionality for availability, performance, offload and security. For availability, the virtual appliance may perform load balancing between layers 4 and 7 of the network and may also perform intelligent service health monitoring. For performance increases via network traffic acceleration, the virtual appliance may perform caching and compression. To offload processing of any servers, the virtual appliance may perform connection multiplexing and pooling and/or SSL processing. For security, the virtual appliance may perform any of the application firewall functionality and SSL VPN function of appliance **200**.

[0177] Any of the modules of the appliance **200** as described in connection with FIG. **2**A 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. **2**A, 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**.

[0178] Still referring to FIG. **4**C, and in brief overview, any one or more vServers **275**A-**275**N 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. **2**B may be designed and constructed to operate in either a virtualized or non-virtualized environment of a server. Any of the vServer **275**, SSL VPN **280**, Intranet UP **282**, Switching **284**, DNS **286**, acceleration **288**, App FW **280** and monitoring agent may be packaged, combined, designed or constructed in a form of application delivery controller **450** deployable as one or more software modules or components executable on a device and/or virtualized environment **400**.

[0179] In some embodiments, a server may execute multiple virtual machines **406***a*-**406***n* 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.

[0180] E. Systems and Methods for Providing a Multi-Core Architecture

[0181] 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.

[0182] 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. **2**A, 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 7× increase in performance and scalability.

[0183] Illustrated in FIG. **5**A 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. **5**A 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 **505**A, a second core **505**B, a third core **505**C, a fourth core **505**D, a fifth core **505**E, a sixth core **505**F, a seventh core **505**G, and so on such that distribution is across all or two or more of the n cores **505**N (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.

[0184] In further detail to FIG. **5**A, in some embodiments, load, work or network traffic can be distributed among cores **505** according to functional parallelism **500**. Functional parallelism may be based on each core performing one or more respective functions. In some embodiments, a first core may perform a first function while a second core performs a second function. In functional parallelism approach, the functions to be performed by the multi-core system are divided and distributed to each core according to functionality. In some embodiments, functional parallelism may be referred to as task parallelism and may be achieved when each processor or core executes a different process or function on the same or different data. The core or processor may execute the same or different code. In some cases, different execution threads or code may communicate with one another as they work. Communication may take place to pass data from one thread to the next as part of a workflow.

[0185] 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) **510**A, secure sockets layer (SSL) encryption and decryption **510**B and transmission control protocol (TCP) functions **510**C. This may lead to a work, performance or computing load **515** based on a volume or level of functionality being used. In some embodiments, distributing work across the cores **505** according to data parallelism **540**, can comprise distributing an amount of work **515** based on distributing data associated with a particular hardware or software component. In some embodiments, distributing work across the cores **505** according to flow-based data parallelism **520**, can comprise distributing data based on a context or flow such that the amount of work **515**A-N on each core may be similar, substantially equal or relatively evenly distributed.

[0186] In the case of the functional parallelism approach, each core may be configured to run one or more functionalities of the plurality of functionalities provided by the packet engine or VIP of the appliance. For example, core 1 may perform network I/O processing for the appliance **200'** while core 2 performs TCP connection management for the appliance. Likewise, core 3 may perform SSL offloading while core 4 may perform layer 7 or application layer processing and traffic management. Each of the cores may perform the same function or different functions. Each of the cores may perform more than one function. Any of the cores may run any of the functionality or portions thereof identified and/or described in conjunction with FIGS. **2**A and **2**B. 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. **5**A, division by function may lead to different cores running at different levels of performance or load **515**.

[0187] In the case of the functional parallelism approach, each core may be configured to run one or more functionalities of the plurality of functionalities provided by the packet engine of the appliance. For example, core 1 may perform network I/O processing for the appliance **200'** while core 2 performs TCP connection management for the appliance. Likewise, core 3 may perform SSL offloading while core 4 may perform layer 7 or application layer processing and traffic management. Each of the cores may perform the same function or different functions. Each of the cores may perform more than one function. Any of the cores may run any of the functionality or portions thereof identified and/or described in conjunction with FIGS. **2**A and **2**B. 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. **5**A division by function may lead to different cores running at different levels of load or performance.

[0188] The functionality or tasks may be distributed in any arrangement and scheme. For example, FIG. **5**B illustrates a first core, Core 1 **505**A, processing applications and processes associated with network I/O functionality **510**A. 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 **510**A will be directed towards Core 1 **505**A which is dedicated to handling all network traffic associated with the NW I/O port. Similarly, Core 2 **505**B is dedicated to handling functionality associated with SSL processing and Core 4 **505**D may be dedicated handling all TCP level processing and functionality.

[0189] While FIG. **5**A illustrates functions such as network I/O, SSL and TCP, other functions can be assigned to cores. These other functions can include any one or more of the functions or operations described herein. For example, any of the functions described in conjunction with FIGS. **2**A and **2**B may be distributed across the cores on a functionality basis. In some cases, a first VIP **275**A may run on a first core while a second VIP **275**B with a different configuration may run on a second core. In some embodiments, each core **505** can handle a particular functionality such that each core **505** can handle the processing associated with that particular function. For example, Core 2 **505**B may handle SSL offloading while Core 4 **505**D may handle application layer processing and traffic management.

[0190] 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) **275**A-C, network interface cards (NIC) **542**D-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**.

[0191] 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.

[0192] While FIG. **5**A illustrates a single vServer associated with a single core **505**, as is the case for VIP1 **275**A, VIP2 **275**B and VIP3 **275**C. 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.

[0193] 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 **542**D and NIC2 **542**E. In other embodiments, one or more NICs can be associated with a single core **505**. In other embodiments, a single NIC can be associated with one or more cores **505**. In these embodiments, load could be distributed amongst the one or more cores **505** such that each core **505** processes a substantially similar amount of load. A core **505** associated with a NIC may process all functions and/or data associated with that particular NIC.

[0194] 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. **5**A.

[0195] In some embodiments, load, work or network traffic can be distributed among cores **505** based on any type and form of data flow. In another of these approaches, the work may be divided or distributed among cores based on data flows. For example, network traffic between a client and a server traversing the appliance may be distributed to and processed by one core of the plurality of cores. In some cases, the core initially establishing the session or connection may be the core for which network traffic for that session or connection is distributed. In some embodiments, the data flow is based on any unit or portion of network traffic, such as a transaction, a request/response communication or traffic originating from an application on a client. In this manner and in some embodiments, data flows between clients and servers traversing the appliance **200'** may be distributed in a more balanced manner than the other approaches.

[0196] In flow-based data parallelism **520**, distribution of data is related to any type of flow of data, such as request/response pairings, transactions, sessions, connections or application communications. For example, network traffic between a client and a server traversing the appliance may be distributed to and processed by one core of the plurality of cores. In some cases, the core initially establishing the session or connection may be the core for which network traffic for that session or connection is distributed. The distribution of data flow may be such that each core **505** carries a substantially equal or relatively evenly distributed amount of load, data or network traffic.

[0197] In some embodiments, the data flow is based on any unit or portion of network traffic, such as a transaction, a request/response communication or traffic originating from an application on a client. In this manner and in some embodiments, data flows between clients and servers traversing the appliance **200'** may be distributed in a more balanced manner than the other approached. In one embodiment, data flow can be distributed based on a transaction or a series of transactions. This transaction, in some embodiments, can be between a client and a server and can be characterized by an IP address or other packet identifier. For example, Core 1 **505**A can be dedicated to transactions between a particular client and a particular server, therefore the load **515**A on Core 1 **505**A may be comprised of the network traffic associated with the transactions between the particular client and server. Allocating the network traffic to Core 1 **505**A can be accomplished by routing all data packets originating from either the particular client or server to Core 1 **505**A.

[0198] While work or load can be distributed to the cores based in part on transactions, in other embodiments load or work can be allocated on a per packet basis. In these embodiments, the appliance **200** can intercept data packets and allocate them to a core **505** having the least amount of load. For example, the appliance **200** could allocate a first incoming data packet to Core 1 **505**A because the load **515**A on Core 1 is less than the load **515**B-N on the rest of the cores **505**B-N. Once the first data packet is allocated to Core 1 **505**A, the amount of load **515**A on Core 1 **505**A is increased proportional to the amount of processing resources needed to process the first data packet. When the appliance **200** intercepts a second data packet, the appliance **200** will allocate the load to Core 4 **505**D because Core 4 **505**D 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 **515**A-N distributed to each core **505** remains substantially equal.

[0199] In other embodiments, load can be allocated on a per unit basis where a section of network traffic is allocated to a particular core **505**. The above-mentioned example illustrates load balancing on a per/packet basis. In other embodiments, load can be allocated based on a number of packets such that every 10, 100 or 1000 packets are allocated to the core **505** having the least amount of load. The number of packets allocated to a core **505** can be a number determined by an application, user or administrator and can be any number greater than zero. In still other embodiments, load can be allocated based on a time metric such that packets are distributed to a particular core **505** for a predetermined amount of time. In these embodiments, packets can be distributed to a particular core **505** for five milliseconds or for any period of time determined by a user, program, system, administrator or otherwise. After the predetermined time period elapses, data packets are transmitted to a different core **505** for the predetermined period of time.

[0200] Flow-based data parallelism methods for distributing work, load or network traffic among the one or more cores **505** can comprise any combination of the above-mentioned embodiments. These methods can be carried out by any part of the appliance **200**, by an application or set of executable instructions executing on one of the cores **505**, such as the packet engine, or by any application, program or agent executing on a computing device in communication with the appliance **200**.

[0201] The functional and data parallelism computing schemes illustrated in FIG. **5**A can be combined in any manner to generate a hybrid parallelism or distributed processing scheme that encompasses function parallelism **500**, data parallelism **540**, flow-based data parallelism **520** or any portions thereof. In some cases, the multi-core system may use any type and form of load balancing schemes to distribute load among the one or more cores **505**. The load balancing scheme may be used in any combination with any of the functional and data parallelism schemes or combinations thereof.

[0202] Illustrated in FIG. **5**B 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 **505**A-N. The system **545** can further include one or more packet engines (PE) or packet processing engines (PPE) **548**A-N communicating with a memory bus **556**. The memory bus may be used to communicate with the one or more processing cores **505**A-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 **505**A-N. The flow distributor **550** can comprise a Receive Side Scaler (RSS) or Receive Side Scaling (RSS) module **560**.

[0203] Further referring to FIG. **5**B, and in more detail, in one embodiment the packet engine(s) **548**A-N can comprise any portion of the appliance **200** described herein, such as any portion of the appliance described in FIGS. **2**A and **2**B. The packet engine(s) **548**A-N can, in some embodiments, comprise any of the following elements: the packet engine **240**, a network stack **267**; a cache manager **232**; a policy engine **236**; a compression engine **238**; an encryption engine **234**; a GUI **210**; a CLI **212**; shell services **214**; monitoring programs **216**; and any other software or hardware element able to receive data packets from one of either the memory bus **556** or the one of more cores **505**A-N. In some embodiments, the packet engine(s) **548**A-N can comprise one or more vServers **275**A-N, or any portion thereof. In other embodiments, the packet engine(s) **548**A-N can provide any combination of the following functionalities: SSL VPN **280**; Intranet UP **282**; switching **284**; DNS **286**; packet acceleration **288**; App FW **280**; monitoring such as the monitoring provided by a monitoring agent **197**; functionalities associated with functioning as a TCP stack; load balancing; SSL offloading and processing; content switching; policy evaluation; caching; compression; encoding; decompression; decoding; application firewall functionalities; XML processing and acceleration; and SSL VPN connectivity.

[0204] The packet engine(s) **548**A-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.

[0205] In some instances, the packet engine(s) **548**A-N can be configured to carry out the any of the functional and/or data parallelism schemes illustrated in FIG. **5**A. In these instances, the packet engine(s) **548**A-N can distribute functions or data among the processing cores **505**A-N so that the distribution is according to the parallelism or distribution scheme. In some embodiments, a single packet engine(s) **548**A-N carries out a load balancing scheme, while in other embodiments one or more packet engine(s) **548**A-N carry out a load balancing scheme. Each core **505**A-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 **548**A-N associated with a core **505** communicate with the other packet engines associated with cores so that the packet engines **548**A-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 **548**A-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**.

[0206] Any of the packet engines running on the cores may run in user mode, kernel or any combination thereof. In some embodiments, the packet engine operates as an application or program running is user or application space. In these embodiments, the packet engine may use any type and form of interface to access any functionality provided by the kernel. In some embodiments, the packet engine operates in kernel mode or as part of the kernel. In some embodiments, a first portion of the packet engine operates in user mode while a second portion of the packet engine operates in kernel mode. In some embodiments, a first packet engine on a first core executes in kernel mode while a second packet engine on a second core executes in user mode. In some embodiments, the packet engine or any portions thereof operates on or in conjunction with the NIC or any drivers thereof.

[0207] 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. **5**B, 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**.

[0208] 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 **505**A-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 **505**A-N can comprise any of the processors described herein. Further, the cores **505**A-N can be configured according to any of the core **505** configurations described herein. Still further, the cores **505**A-N can have any of the core **505** functionalities described herein. While FIG. **5**B illustrates seven cores **505**A-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.

[0209] 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.

[0210] Further referring to FIG. **5**B, any of the functionality and/or embodiments of the cores **505** described above in connection with FIG. **5**A can be deployed in any embodiment of the virtualized environment described above in connection with FIGS. **4**A and **4**B. Instead of the functionality of the cores **505** being deployed in the form of a physical processor **505**, such functionality may be deployed in a virtualized environment **400** on any computing device **100**, such as a client **102**, server **106** or appliance **200**. In other embodiments, instead of the functionality of the cores **505** being deployed in the form of an appliance or a single device, the functionality may be deployed across multiple devices in any arrangement. For example, one device may comprise two or more cores and another device may comprise two or more cores. For example, a multi-core system may include a cluster of computing devices, a server farm or network of computing devices. In some embodiments, instead of the functionality of the cores **505** being deployed in the form of cores, the functionality may be deployed on a plurality of processors, such as a plurality of single core processors.

[0211] In one embodiment, the cores **505** may be any type and form of processor. In some embodiments, a core can function substantially similar to any processor or central processing unit described herein. In some embodiment, the cores **505** may comprise any portion of any processor described herein. While FIG. **5**A illustrates seven cores, there can exist any "N" number of cores within an appliance **200**, where "N" is any whole number greater than one. In some embodiments, the cores **505** can be installed within a common appliance **200**, while in other embodiments the cores **505** can be installed within one or more appliance(s) **200** communicatively connected to one another. The cores **505** can in some embodiments comprise graphics processing software, while in other embodiments the cores **505** provide general processing capabilities. The cores **505** can be installed physically near each other and/or can be communicatively connected to each other. The cores may be connected by any type and form of bus or subsystem physically and/or communicatively coupled to the cores for transferring data between to, from and/or between the cores.

[0212] While each core **505** can comprise software for communicating with other cores, in some embodiments a core manager (not shown) can facilitate communication between each core **505**. In some embodiments, the kernel may provide core management. The cores may interface or communicate with each other using a variety of interface mechanisms. In some embodiments, core to core messaging may be used to communicate between cores, such as a first core sending a message or data to a second core via a bus or subsystem connecting the cores. In some embodiments, cores may communicate via any type and form of shared memory interface. In one embodiment, there may be one or more memory locations shared among all the cores. In some embodiments, each core may have separate memory locations shared with each other core. For example, a first core may have a first shared memory with a second core and a second share memory with a third core. In some embodiments, cores may communicate via any type of programming or API, such as function calls via the kernel. In some embodiments, the operating system may recognize and support multiple core devices and provide interfaces and API for inter-core communications.

[0213] The flow distributor **550** can be any application, program, library, script, task, service, process or any type and form of executable instructions executing on any type and form of hardware. In some embodiments, the flow distributor **550** may any design and construction of circuitry to perform any of the operations and functions described herein. In some embodiments, the flow distributor distribute, forwards, routes, controls and/ors manage the distribution of data packets among the cores **505** and/or packet engine or VIPs running on the cores. The flow distributor **550**, in some embodiments, can be referred to as an interface master. In one embodiment, the flow distributor **550** comprises a set of executable instructions executing on a core or processor of the appliance **200**. In another embodiment, the flow distributor **550** comprises a set of executable instructions executing on a computing machine in communication with the appliance **200**. In some embodiments, the flow distributor **550** comprises a set of executable instructions executing on a NIC, such as firmware. In still other embodiments, the flow distributor **550** comprises any combination of software and hardware to distribute data packets among cores or processors. In one embodiment, the flow distributor **550** executes on at least one of the cores **505**A-N, while in other embodiments a separate flow distributor **550** assigned to each core **505**A-N executes on an associated core **505**A-N. The flow distributor may use any type and form of statistical or probabilistic algorithms or decision making to balance the flows across the cores. The hardware of the appliance, such as a NIC, or the kernel may be designed and constructed to support sequential operations across the NICs and/or cores.

[0214] 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.

[0215] 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.

[0216] 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 **505**A-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**.

[0217] In some embodiments, the RSS module **560** uses the MICROSOFT receive-side-scaling (RSS) scheme. In one embodiment, RSS is a Microsoft Scalable Networking initiative technology that enables receive processing to be balanced across multiple processors in the system while maintaining in-order delivery of the data. The RSS may use any type and form of hashing scheme to determine a core or processor for processing a network packet.

[0218] The RSS module **560** can apply any type and form hash function such as the Toeplitz hash function. The hash function may be applied to the hash type or any the sequence of values. The hash function may be a secure hash of any security level or is otherwise cryptographically secure. The hash function may use a hash key. The size of the key is dependent upon the hash function. For the Toeplitz hash, the size may be 40 bytes for IPv6 and 16 bytes for IPv4.

[0219] The hash function may be designed and constructed based on any one or more criteria or design goals. In some embodiments, a hash function may be used that provides an even distribution of hash result for different hash inputs and different hash types, including TCP/IPv4, TCP/IPv6, IPv4, and IPv6 headers. In some embodiments, a hash function may be used that provides a hash result that is evenly distributed when a small number of buckets are present (for example, two or four). In some embodiments, hash function may be used that provides a hash result that is randomly distributed when a large number of buckets were present (for example, 64 buckets). In some embodiments, the hash function is determined based on a level of computational or resource usage. In some embodiments, the hash function is determined based on ease or difficulty of implementing the hash in hardware. In some embodiments, the hash function is determined based on the ease or difficulty of a malicious remote host to send packets that would all hash to the same bucket.

[0220] The RSS may generate hashes from any type and form of input, such as a sequence of values. This sequence of values can include any portion of the network packet, such as any header, field or payload of network packet, or portions thereof. In some embodiments, the input to the hash may be referred to as a hash type and include any tuples of information associated with a network packet or data flow, such as any of the following: a four tuple comprising at least two IP addresses and two ports; a four tuple comprising any four sets of values; a six tuple; a two tuple; and/or any other sequence of numbers or values. The following are example of hash types that may be used by RSS:

[0221] 4-tuple of source TCP Port, source IP version 4 (IPv4) address, destination TCP Port, and destination IPv4 address.

[0222] 4-tuple of source TCP Port, source IP version 6 (IPv6) address, destination TCP Port, and destination IPv6 address.

[0223] 2-tuple of source IPv4 address, and destination IPv4 address.

[0224] 2-tuple of source IPv6 address, and destination IPv6 address.

[0225] 2-tuple of source IPv6 address, and destination IPv6 address, including support for parsing IPv6 extension headers.

[0226] The hash result or any portion thereof may used to identify a core or entity, such as a packet engine or VIP, for distributing a network packet. In some embodiments, one or more hash bits or mask are applied to the hash result. The hash bit or mask may be any number of bits or bytes. A NIC may support any number of bits, such as seven bits. The network stack may set the actual number of bits to be used during initialization. The number will be between 1 and 7, inclusive.

[0227] The hash result may be used to identify the core or entity via any type and form of table, such as a bucket table or indirection table. In some embodiments, the number of hash-result bits are used to index into the table. The range of the hash mask may effectively define the size of the indirection table. Any portion of the hash result or the hast result itself may be used to index the indirection table. The values in the table may identify any of the cores or processor, such as by a core or processor identifier. In some embodiments, all of the cores of the multi-core system are identified in the table. In other embodiments, a port of the cores of the multi-core system are identified in the table. The indirection table may comprise any number of buckets for example 2 to 128 buckets that may be indexed by a hash mask. Each bucket may comprise a range of index values that identify a core or processor. In some embodiments, the flow controller and/or RSS module may rebalance the network rebalance the network load by changing the indirection table.

[0228] In some embodiments, the multi-core system **575** does not include a RSS driver or RSS module **560**. In some of these embodiments, a software steering module (not shown) or a software embodiment of the RSS module within the system can operate in conjunction with or as part of the flow distributor **550** to steer packets to cores **505** within the multi-core system **575**.

[0229] 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 **505**A, 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**.

[0230] 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.

[0231] Although FIG. **5**B 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.

[0232] Illustrated in FIG. **5**C is an embodiment of a multi-core system **575** comprising one or more processing cores **505**A-N. In brief overview, one of the cores **505** can be designated as a control core **505**A 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 **505**A-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.

[0233] Further referring to FIG. **5**C, and in more detail, the cores **505**A-N as well as the control core **505**A can be any processor described herein. Furthermore, the cores **505**A-N and the control core **505**A can be any processor able to function within the system **575** described in FIG. **5**C. Still further, the cores **505**A-N and the control core **505**A 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.

[0234] 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, 2 GB of each core excluding the control plane core or core 1 may be used to form a 28 GB 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.

[0235] The global cache **580** can be any type and form of memory or storage element, such as any memory or storage element described herein. In some embodiments, the cores **505** may have access to a predetermined amount of memory (i.e. 32 GB or any other memory amount commensurate with the system **575**). The global cache **580** can be allocated from that predetermined amount of memory while the rest of the available memory can be allocated among the cores **505**. In other embodiments, each core **505** can have a predetermined amount of memory. The global cache **580** can comprise an amount of the memory allocated to each core **505**. This memory amount can be measured in bytes, or can be measured as a percentage of the memory allocated to each core **505**. Thus, the global cache **580** can comprise 1 GB of memory from the memory associated with each core **505**, or can comprise 20 percent or one-half of the memory associated with each core **505**. In some embodiments, only a portion of the cores **505** provide memory to the global cache **580**, while in other embodiments the global cache **580** can comprise memory not allocated to the cores **505**.

[0236] 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. **2**A and cache functionality of FIG. **2**B may use the global cache to share data for acceleration. For example, each of the packet engines may store responses, such as HTML data, to the global cache. Any of the cache managers operating on a core may access the global cache to server caches responses to client requests.

[0237] In some embodiments, the cores **505** can use the global cache **580** to store a port allocation table which can be used to determine data flow based in part on ports. In other embodiments, the cores **505** can use the global cache **580** to store an address lookup table or any other table or list that can be used by the flow distributor to determine where to direct incoming and outgoing data packets. The cores **505** can, in some embodiments read from and write to cache **580**, while in other embodiments the cores **505** can only read from or write to cache **580**. The cores may use the global cache to perform core to core communications.

[0238] 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 **505**A can receive a greater amount of available cache, while the other cores **505** can receiving varying amounts or access to the global cache **580**.

[0239] In some embodiments, the system **575** can comprise a control core **505**A. While FIG. **5**C illustrates core 1 **505**A 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**.

[0240] 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.

[0241] The control core **505**A 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 **505**A, 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 **505**A. Determining the boundaries of the control plane **570** can include maintaining, by the control core **505**A or agent executing within the system **575**, a list of those cores **505** controlled by the control core **505**A. The control core **505**A 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**.

[0242] F. Systems and Methods for Providing a Distributed Cluster Architecture

[0243] 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.

[0244] Referring now to FIG. **6**, illustrated is an embodiment of a computing device cluster or appliance cluster **600**. A plurality of appliances **200***a*-**200***n* or other computing devices, sometimes referred to as nodes, such as desktop computers, servers, rackmount 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.

[0245] 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 rackmount 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.

[0246] 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.

[0247] In many embodiments, an appliance cluster **600** may be physically grouped, such as a plurality of blade servers in a chassis or plurality of rackmount 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.

[0248] In some embodiments, an appliance cluster **600** may be connected to one or more networks **104**, **104'**. For example, referring briefly back to FIG. **1**A, 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. **1**B-**1**D 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 **200***a* may be located at a first site, such as a data center and a second appliance **200***b* 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 **200***a*-**200***b*, 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 **200***b* deployed at the site, load balancing may be weighted by location, or similar steps can be taken to mitigate any latency.

[0249] Still referring to FIG. **6**, an appliance cluster **600** may be connected to a network via 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 **200***a*-**200***n* 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 **200***a*-**200***n*. 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 interface master may comprise one of appliances **200***a*-**200***n*. For example, a first appliance **200***a* may serve as the interface master, receiving incoming traffic for the appliance cluster **600** and distributing the traffic across each of appliances **200***b*-**200***n*. In some embodiments, return traffic may similarly flow from each of appliances **200***b*-**200***n* via the first appliance **200***a* serving as the interface master. In other embodiments, return traffic from each of appliances **200***b*-**200***n* 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.

[0250] 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 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 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.

[0251] 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 **200***a*-**200***n* 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 **200***a*-**200***n*. In many embodiments, each appliance **200***a*-**200***n* 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.

[0252] In some embodiments, each appliance **200***a*-**200***n* 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 **200***a* communicates with a client via network **104**, and a second appliance **200***b* 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 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 **200***a*-**200***n* and serve multiple routing functions simultaneously, to reduce complexity or eliminate extra devices from the system.

[0253] 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 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 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.

[0254] 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 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 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.

[0255] 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 it's 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.

[0256] G. Systems and Methods for Generating Cookie Signatures for Security Protection in a Multi-Core System

[0257] Before discussing the specifics of cookie signatures, it may be helpful to first describe a few types of malicious attacks for which cookies are used for protection. Synchronization (SYN) attacks, sometimes called SYN floods, and HTTP Denial of Service (HTTP DoS) attacks are two similar methods that malicious attackers can use to slow down or disable a remote server by tying up memory and resources to prevent innocent users from accessing said resources.

[0258] In the standard 3-way handshaking protocol of TCP and similar transport layer protocols, a client requests a connection by sending a SYN message to a server or appliance. Classically, the server or appliance allocates memory and/or resources to a client-side socket, and responds with an acknowledgement message (SYN-ACK). The client then responds with an acknowledgement (ACK) and the connection is established.

[0259] In the SYN flood or SYN attack, a malicious client or clients send a plurality of SYN requests. As is usual, the appliance or server allocates memory and resources for each request and responds with SYN-ACK messages. The malicious client never responds to these SYN-ACK messages with acknowledgement messages, and the connections are not established. Rather, the server or appliance remains in a listening state waiting for the acknowledgement messages from the client or clients, and the memory and resources stay allocated to these connections, until the server or appliance times out, which may be several minutes.

[0260] One solution proposed for SYN floods was to simply not allocate the resources on the server or appliance until the client responds to the SYN-ACK message. However, the server or appliance must still remember which clients have received which SYN-ACK messages so that when a corresponding ACK arrives, the server or appliance may respond properly. Accordingly, a small amount of data regarding the connection is stored in a SYN queue. As more SYN requests come in, the SYN queue can become overloaded. Some systems will ignore further SYN requests, including those from legitimate users; other systems remove the oldest SYN requests in the SYN queue, which again, may be from legitimate users. A further solution is to increase the size of the SYN queue. However, this may be undesirable.

[0261] The use of SYN cookies is a solution that avoids the use of the SYN queue in cases of overflow. Information that would be placed in the SYN queue as an entry for any particular SYN request is instead encoded into the initial sequence number transmitted in the SYN-ACK packet to the client. If an ACK packet is later received from the client acknowledging the initial sequence number, the server or appliance may decode from the sequence number the original information. This also prevents an extension of the TCP flood attack, in which a malicious attacker sends acknowledgement packets with spoofed IP addresses to forge a connection to another host. Because these seem to be legitimate connections to the server, memory and resources are allocated to a connection that doesn't actually exist. By including a secret key value in the encoded initial sequence number, the server can ignore acknowledgement packets that don't include the encoded secret key, making it prohibitively difficult for a malicious attacker to guess sequence numbers sent to other IP addresses.

[0262] In many implementations of SYN cookies, the initial sequence number (ISN) for a SYN-ACK packet is determined by using a hash function, such as MD5, on an input of the source IP and port and destination IP and port indicated by the original SYN request, along with one or more random numbers provided by a random or pseudo-random number generator. This value may be concatenated with a number representing the maximum segment size (MSS) value of the connection, and a timer value that is slowly increased. The timer value is used to ensure that ISNs increase over time, as is required by the TCP protocol. In one implementation of SYN cookies described in IETF RFC 4987, the result may be further concatenated with a second hash of the source IP and port, destination IP and port, timer value, and a second random number. When an ACK packet arrives, the server or appliance may create one or more ISNs for the source IP and port and destination IP and port indicated in the ACK using the current or one or more previous timer values and random number values to determine if the acknowledgement number included in the ACK packet corresponds to an ISN that could have been created in the last few minutes. If so, then the server may create a transmission control block (TCB) for the connection, using the MSS value indicated in the ISN. If the acknowledgement number of the ACK packet does not correspond to any ISN, then the ACK packet may be dropped. In many implementations, there aren't enough bits in the encoded ISN to fully indicate the maximum segment size for the connection. In these implementations, an index of 8 common MSS values is created, and the closest index value to the MSS indicated in the SYN request is chosen and encoded in the ISN as a 3-bit value. In a further implementation of SYN cookies, additional TCP options, such as window scaling or others may be included in the timestamp option field. Because a timestamp is echoed by a receiver, if timestamp options are enabled on a client, it will respond to the SYN-ACK packet with an ACK including the encoded additional TCP options in the timestamp echo field, and the server may create a TCB with corresponding TCP options.

[0263] Thus, in responding to SYN flood attacks, the server or appliance uses random numbers for entropy in calculating the hash function for the initial sequence number of SYN-ACK packets. In a multi-core system, this may present complications. In one implementation in which each core or packet engine of a multi-core system maintains its own random number generator or seed from which random numbers are generated, one packet engine may create a SYN-ACK packet with an ISN from a first random number, but another packet engine may receive the corresponding ACK packet. If the second packet engine does not have the same random number in a cache, the ISNs created by the second packet engine won't correspond to the ISN in the ACK packet, and the second packet engine will drop the packet, even though it's from a legitimate client.

[0264] Although discussed above in reference to TCP SYN flood attacks, similar attacks and corresponding solutions and implementation difficulties exist with other transport layer protocols.

[0265] A similar attack to the SYN flood is the HTTP Denial of Service (DoS) attack. In this attack, a malicious attacker or attackers establish legitimate connections with the appliance or server and send HTTP GET requests for files. In some implementations, the HTTP GET requests are incomplete requests, which tie up the server or appliance connection waiting for the remainder of the request until a timeout value expires. In other implementations, the GET requests are complete requests for very large files, which are immediately discarded on receipt by the attacker, who then issues another GET request. In these implementations, the attacker will frequently spoof or change his IP address, preventing successful packet filtering solutions. The same behavior can occur non-maliciously when a breaking news event leads a large number of users to request the same data simultaneously, overloading the capabilities of the server. Worse, a mix of malicious and non-malicious requests exacerbates the problem.

[0266] Some proposed solutions for the HTTP DoS attack include distributed caching and precaching and other methods of increasing the ability of the system to serve content to clients. However, a malicious attacker, particularly one that is discarding responses, may be able to overload a system faster than resources may be added to it.

[0267] Another solution takes advantage of the fact that malicious attackers will drop responses. In implementations of this solution, when the server or appliance receives an HTTP GET request, it may respond with an HTTP reply that includes a cookie. In one embodiment, the reply may be the response requested by the GET request, while in other embodiment, the reply may be a short response comprising the cookie and a refresh command, such as a javascript refresh or an http metatag with a refresh command. This later embodiment may be used, for example, when a large number of requests arrive simultaneously, so that the server or appliance may determine if the are legitimate requests or malicious attacks. A legitimate requestor will reply to the refresh command with a second GET request, this time including the cookie, while a malicious attacker will drop the reply without processing it, and generate future GET requests without the cookie. Thus, the presence of the cookie may be used to identify a legitimate client.

[0268] In cases where a server or appliance is not currently overloaded with requests, the server or appliance may, in some embodiments, still include the cookie in the response. Future requests from the same client will include the cookie, up until the time the cookie expires, the client cache is cleared, or a new cookie is set. During a later HTTP DoS attack, the server or appliance may service requests from this client with a higher priority, knowing that the client was, at least at one prior time, a legitimate client rather than a malicious attacker.

[0269] In many embodiments, the cookie used in these implementations is generated from a hash function with a random seed that is changed frequently. This decreases the likelihood of a malicious attacker being able to receive a first cookie in response to a request, and then generate a plurality of requests by reusing the cookie or guessing future cookies. In one such embodiment, the server or appliance maintains an array of cookie signatures. This array may be any size, depending on the amount of entropy and frequency of cookie variance desired. In one implementation, the cookie array may include 64, 128, 192, 256, or more cookie signatures, each of 8, 16, 24, 32, 40, or more bits in length. In some implementations, the cookie may be generated by concatenating one or more cookie signatures from the array. In another implementation, the cookie may be generated by applying a hash function to one or more cookie signatures from the array. In many implementations, a pointer within the array is advanced, or the values of the array are rotated, such that each successive cookie uses one or more new values from the array. Additionally, in some implementations, the cookie signature array may be replaced periodically with a new array. To prevent accidentally treating legitimate requests with cookies from just before the array was replaced as malicious, the previous cookie signature array may be temporarily stored for a period of time. Accordingly, requests containing cookies may be compared to both the current cookie signature array and the previous cookie signature array to determine if the request is legitimate.

[0270] Thus, in responding to HTTP DoS attacks, similar to SYN flood attacks, the server or appliance uses random numbers for entropy in creating the cookie signature array. As discussed above, in a multi-core system, this may present complications. In many embodiments, the cookie signatures of the array are generated by a random number generator or pseudo-random number generator using a seed. In one implementation in which each core or packet engine of a multi-core system maintains its own random number generator or seed from which random numbers are generated, one packet engine may create a cookie signature array from a first seed and send a response to a client containing a first cookie, but another packet engine may receive the next request from the client. If the second packet engine does not have the same random number in a cache, the cookie signature array created by the second packet engine won't correspond to the cookie in the request, and the second packet engine will drop the packet, even though it's from a legitimate client. As discussed above, the cookies may be used in transport layer headers, such as in SYN cookies; in application layer headers, such as in HTTP cookies; or in one or more headers of one or more layers of the OSI model.

[0271] Shown in FIG. 7A is a system **790** for generating cookie signatures in a multi-core system. Briefly, appliance **200** may comprise one or more cores **505**A-**505**N, flow distributors **550**, and NICs **552**, discussed above. Each core **505**A-**505**N may comprise a packet engine **548**A-**548**N, discussed above, a timer **700**A-**700**N (referred to generally as timer(s) **700**), and a cache **702**A-**702**N (referred to generally as cache(s) **702** or local cache(s) **702**). The appliance may also comprise a shared memory **704**, which may be part of a main memory **122**, cache **140**, storage **128**, or any other memory element similar to those discussed herein. In some embodiments, the shared memory **704** may comprise a random seed **706**, which may also be referred to as a global random seed **706**. Each cache of the one or more caches **702** may comprise a random seed **708**, which may also be referred to as a local random seed **708**. Each cache may further comprise a current cookie signature **710**A-**710**N (referred to generally as current cookie signature(s) **710**) and a previous cookie signature **712**A-**712**N (referred to generally as previous cookie signature(s) **712**). Appliance **200** may also comprise one or more random number generators or pseudo-random number generators (not shown), which may use an internal or external source of entropy. In some embodiments, a random number generator may comprise a function, subroutine, or service executed by a packet engine **548**.

[0272] Still referring to FIG. 7A and in more detail, in some embodiments, each core **505** may be configured with a timer **700**. Timer **700** may comprise a service, daemon, process, function, subroutine, application, or any type and form of executable instructions for setting and operating a timer, and sending a notification on expiration of the timer. Timer **700** may comprise hardware, software, or any combination of hardware and software. In some embodiments, timer **700** may count upwards or downwards from a first predetermined value to a second predetermined value, and may count seconds, milliseconds, microseconds, or any interval selected by an administrator. In some embodiments, timer **700** may be used to initiate a reset of a global or local random seed, as described below in connection with FIG. **7**B. In many embodiments, timer **700** may comprise a plurality of timers with different durations, frequencies, or periods, such that timing of different events is possible.

[0273] In some embodiments, appliance **200** may comprise a shared memory **704**. A shared memory **704** may comprise a storage device or element, such as main memory **122** or cache **140**, a global cache **580** discussed above, or any other type and form of memory element capable of storing a random seed **706** and accessible by a plurality of cores **505**. In some embodiments, shared memory **704** may comprise a mutex or semaphor. In some embodiments, shared memory **704** may comprise functionality for locking a shared memory location, such that a first packet engine or core may write to the location while other packet engines or cores are prevented from reading from the location. In other embodiments, the shared memory **704** may comprise a native integer such that the first packet engine or core may update the integer with a lockless read-modify-write transaction.

[0274] In some embodiments, a random seed **706** or global random seed **706** may comprise a random or pseudo random number generated by a random or pseudo-random number generator. In one embodiment, the global random seed may be an integer of 8, 16, 24, 32 bits or more. In many embodiments, the global random seed **706** may be generated by a primary packet engine and placed in the shared memory **704**. At intervals dictated by the expiration of a timer executing on the primary packet engine, the primary packet engine may generate a new global random seed **706** and replace the global random seed stored in the shared memory **704**.

[0275] In many embodiments, each core **505** may comprise a cache **702**. A cache **702** may comprise a buffer, cache, or memory element, or any other type and form of memory structure or portion thereof, accessible by a packet engine executing on a core **505**. As shown, a cache **702** may comprise a local random seed **708**, a current cookie signature **710**, and a previous cookie signature **712**.

[0276] In some embodiments, a local random seed **708** may comprise a copy of a global random seed **706**, copied to a local cache of each core. In one such embodiment, when a global random seed **706** has been changed, such as in response to the expiration of a timer on a primary packet engine, each packet engine may copy the global random seed **706** into a cache **702** as a local random seed **708**.

[0277] In many embodiments, random seed **708** may be used to construct a current cookie signature **710**. A cookie signature can be any form of a digital signature. In one embodiment, a current cookie signature **710** may comprise one or more hash seeds, such as for a SYN cookie hash as discussed above. In another embodiment, a current cookie signature **710** may comprise one or more cookie signatures for creation of an HTTP DoS cookie, as discussed above. In many embodiments, the current cookie signature **710** may comprise an array of cookie signatures, used for both SYN and HTTP DoS cookies. The cookie signatures may be created by using a pseudo-random function, such as the BSD or Linux random( ) or rand( ) functions, the rand( ) function of Microsoft Windows, or any other function that creates one or more random numbers. In many embodiments, the pseudo-random function may be initialized with the global random seed **706** or local random seed **708** for generating the first random number, and each successive random number of the array may be generated using the previous random number as a seed. Because pseudo-random number generators are deterministic, each packet engine will generate the same array of cookie signatures provided each starts from the same random seed. Thus, using a global random seed removes the need to communicate a large amount of lengthy cookie signatures generated by a primary packet engine to one or more other packet engines.

[0278] As discussed above, to allow for legitimate requests that arrive after creation of a new set of cookie signatures to not be discarded, each cache may comprise a previous cookie signature **712**. Before generating current cookie signatures **710**, each packet engine may copy the array to the previous cookie signature **712**, which may then be compared against incoming requests. Thus, if the global random seed **706** is updated every minute and a new set of cookie signatures are created, a request including a cookie signature will have a potential lifetime of two minutes before becoming invalid. Similarly, if the global random seed **706** is updated every two minutes, a request including a cookie signature will have a potential lifetime of four minutes before expiring.

[0279] Shown in FIG. 7B is a flow chart of an embodiment of a method of generating and maintaining consistent cookie signatures in a multi-core system. Briefly, a primary packet engine may generate a global random seed at step **730**. At step **732**, the primary packet engine may store the global random seed to a local cache. At step **734**, the primary packet engine may store current cookie signatures as previous cookie signatures. At step **736**, the primary packet engine may generate new cookie signatures from the locally cached random seed. Responsive to the expiration of a timer at step **738**, the primary packet engine may repeat steps **730**-**738**. Simultaneously, when the global random seed has changed, one or more other packet engines may store the global random seed to a local cache at step **732**. At step **734**, the one or more other packet engines may store current cookie signatures as previous cookie signatures. At step **736**, the one or more other packet engines may generate new cookie signatures from the locally cached random seed. Responsive to the expiration of a timer at step **740**, the one or more other packet engines may determine if the global random seed has changed at step **742**. If not, the one or more other packet engines may repeat steps **740**-**742**. If so, the one or more other packet engines may repeat steps **732**-**742**. Although one of the packet engines may be referred to as a primary packet engine, any of the packet engines may be designated as a primary packet engine in using any of the techniques and methods described herein.

[0280] Still referring to FIG. 7B and in more detail, at step **730**, a primary packet engine may generate a global random seed. In some embodiments, generating a global random seed may comprise executing a function call to generate a random seed, or may comprise requesting a random number from a random or pseudo-random number generator. In a further embodiment, generating a global random seed may comprise accessing a source of entropy, such as a clock timer, a network packet string, a temperature sensor, a voltage sensor, or any other type and form of random value that may be used as a seed for a random number generator. In some embodiments, generating the global random seed may comprise replacing an existing global random seed stored in a shared memory.

[0281] At step **732**, in some embodiments, the primary packet engine may store the global random seed to a local cache. In many embodiments, the primary packet engine may have already stored the global random seed to a local cache as part of generating the global random seed. Accordingly, in these embodiments, the primary packet engine may skip this step.

[0282] At step **734**, in some embodiments, the primary packet engine may store the current cookie signatures as a set of previous cookie signatures. In one embodiment, storing the current cookie signatures as a set of previous cookie signatures may comprise copying or moving the current cookie signature array to a different position in memory. In another embodiment, storing the current cookie signatures as a set of previous cookie signatures may comprise replacing a previous cookie signature array.

[0283] At step **736**, the primary packet engine may generate a set of new cookie signatures from the locally-cached random seed. In one embodiment, generating a set of new cookie signatures may comprise executing a pseudo-random number function using the locally-cached random seed as an initial seed to generate the first cookie signature, and generating each successive cookie signature using the previous cookie signature as a seed for the pseudo-random number function. In many embodiments, generating a set of new cookie signatures may comprise generating an array of 64, 128, 192, 256, or more cookie signatures, each of 8, 16, 24, 32, 40, or more bits in length. In one embodiment, the number of cookie signatures in the array and length of each cookie signature may be determined by a policy set by a user or administrator.

[0284] At step **738**, responsive to the expiration of a timer, the primary packet engine may repeat steps **730**-**738**. The long timer period expiring at step **738** may comprise any value longer than the value of the short timer period expiring at step **740**, discussed below. In one embodiment, the long timer period may be 10 seconds, 30 seconds, 1 minute, 2 minutes, 5 minutes, any value between these times, or any value longer or shorter, provided that the period is longer than the value of the short timer discussed below.

[0285] Still referring to FIG. **7**B, one or more other packet engines may, responsive to the global random seed having been changed by the primary packet engine, store the global random seed to respective local caches at step **732**. In one embodiment, storing the global random seed to a local cache may comprise copying the global random seed to a local cache. In an embodiment in which the primary packet engine locks the shared memory for reading as part of generating a global random seed at step **730**, step **732** may comprise the one or more other packet engines waiting in a spinlock condition for the global random seed to become unlocked such that they may copy the global random seed to local caches.

[0286] The one or more other packet engines may, in some embodiments, store current cookie signatures as previous cookie signatures at step **734** and generate new cookie signatures from a locally-cached random seed at step **736** as discussed above in connection with the primary packet engine. As discussed above, because each packet engine generates cookie signatures starting with an initial value of the global random seed, the current cookie signature array on each packet engine will be identical after completion of step **736**.

[0287] In response to expiration of a timer at step **740**, each packet engine of the one or more packet engines may determine if the global random seed has changed at step **742**. Due to natural skew of local timers across a plurality of packet engines as well as processing delays due to other tasks, in many embodiments where there are multiple packet engines other than the primary packet engine, each of the one or more other packet engines may not necessarily execute steps **740** and **742** simultaneously. By selecting a small value for the short timer, each packet engine will detect a change in the global random seed within a short time of each other. For example, if the short timer is set to a period of one second, then at worst one packet engine may detect the change less than a second after another packet engine. By retaining the set of previous cookie signatures for the duration of the long timer, substantially longer than the short timer, then any lag between the timers of different packet engines becomes inconsequential. As discussed above, the short timer may be set to any period less than the value of the longer timer, such as one second, two seconds, or any other value. The minimum difference between the period of the short timer and the long timer may be determined based on the time to execute steps **732**-**736**. If the difference is less than this time, the global random seed may change faster than the cookie signatures may be updated. However, this is unlikely to occur in modern, high speed systems unless extreme values are selected, on the order of nanoseconds.

[0288] At step **742**, each of the one or more packet engines may determine, independently, if the global random seed has changed. In one embodiment, determining if the global random seed has changed may comprise a packet engine comparing the global random seed to a locally cached random seed. If there is no difference and the global random seed has not changed, then the packet engine may repeat steps **740**-**742**. If the global random seed has changed, the packet engine may repeat steps **732**-**742**.

[0289] Shown in FIG. **7**C is a flow chart of an embodiment of a method of using cookie signatures for security in a multi-core system. Briefly, at step **720**, a packet engine may receive a request from a client with a cookie. At step **722**, the packet engine may compare cookie signatures of the request with current and previous cookie signatures. If the request cookie signatures match the current or previous cookie signatures, then at step **724** the packet engine may accept and process the request. If the cookie signature does not match, then the packet engine may determine if the global random seed has changed, as described above in step **742** of FIG. **7**B. If not, then the packet engine may deny the request at step **726**. However, if the global random seed has changed, then the packet engine may execute steps **732**-**736** as described above in FIG. **7**B and repeat steps **722**-**726**.

[0290] Still referring to FIG. **7**C and in more detail, in some embodiments, a packet engine may receive a request from a client at step **720**. In some embodiments, the request may not contain a cookie, and the packet engine may process the request according to other policies. For example, if the request is a SYN request and does not include a cookie, then the packet engine may respond with a SYN-ACK including a cookie, as described above. Similarly, if the request is an HTTP GET request and does not include a cookie, then the packet engine may respond with the requested file and include a cookie, or a substitute file with a cookie such as the refresh command described above, responsive to a policy detecting an attempted HTTP DoS attack, as described above. In other embodiments, the request may include a cookie. In some embodiments, cookies are explicit, such as a cookie value in a header of an HTTP GET request. In other embodiments, such as in implementations using SYN cookies, cookies are not explicit but are encoded into an acknowledgement field of an ACK packet. In these embodiments, the request may be treated as if it has a cookie for the purpose of the method shown in FIG. **7**C.

[0291] At step **722**, in some embodiments, the packet engine may compare the cookie in the request with one or more current and/or previous cookie signatures. For example, in an embodiment in which the cookie comprises one or more concatenated cookie signatures of predetermined lengths, the packet engine may parse the cookie into the one or more cookie signatures, and then attempt to locate the one or more cookie signatures in the current cookie signature array. If the packet engine cannot locate the cookie signatures, then in some embodiments, the packet engine may attempt to locate the one or more cookie signatures in the previous cookie signature array.

[0292] In embodiments in which the cookie comprises one or more hash results, the packet engine may execute a reverse hash function to determine the cookie signatures. In another embodiment where a reverse hash function is not available, the packet engine may create one or more cookies using a hash function and information available in the packet request. For example, in one implementation of SYN cookies discussed above, the initial sequence number is selected based on a hash of one or more of the source IP and port and destination IP and port of the SYN request, a slowly increasing counter value, and one or more cookie signatures, and concatenated with a 3-bit encoded MSS value. Accordingly, in one embodiment, the packet engine may create one or more cookies using the 3-bit MSS value and a hash of the source IP and port and destination IP and port of the ACK, the current and previous few counter values based on the time to live of the SYN-ACK, and one or more cookie signatures from the current cookie signatures array or previous cookie signatures array. For example, if the counter value increases once per minute and the SYN-ACK has a five minute time-to-live, the packet engine may use the current and previous four counter values in the hash. Each test cookie may then be compared to the sequence number in the received request to determine if the sequence number represents a legitimate cookie. In one embodiment, the sequence number is tested after each test cookie is created, while in another embodiment, the sequence number is tested after a set of test cookies are created.

[0293] If the received request contains a legitimate cookie, then at step **724**, the request is accepted. Accepting the request may comprise the packet engine further processing the request; establishing a connection; allocating memory and/or resources; initializing a transmission control block or packet control buffer; forwarding the request to a server, service, or virtual server; or otherwise handling the request. In one embodiment, in which a server is currently experiencing a flood of HTTP GET requests due to a DoS attack or a surge due to a breaking news event, the packet engine may buffer the received request and assign it a higher priority than other requests because of the legitimate cookie in the received request.

[0294] If the received request does not contain a cookie signature that corresponds to a cookie signature in the current cookie signature array or previous cookie signature array, then at step **742**, the packet engine may determine if the global random seed has changed, using any of the methods described above in connection with FIG. **7**B. If the global random seed has changed, then in some embodiments, the packet engine may execute steps **732**-**736**, also described above in connection with FIG. **7**B and repeat steps **722** for the newly generated current cookie signatures.

[0295] If the global random seed has not changed, then at step **726**, in some embodiments, the packet engine may deny the request. In other embodiments, the packet engine may take other actions, depending on a policy in place. For example, in an embodiment in which the packet engine is not currently experiencing a flood of HTTP GET requests, the packet engine may forward the request or process the request normally. In another embodiment, in which the packet engine is experiencing a flood of requests, the packet engine may buffer the received request and assign it a lower priority than other requests with legitimate cookies. In yet another embodiment, the packet engine may reply to the received request with a small response comprising a refresh command and a legitimate cookie, as described above. Thus, the cookies may be used to ensure priority processing for legitimate clients while not treating new legitimate clients who have not yet received cookies as malicious attackers.

[0296] H. Systems and Methods for Cookie Signatures and SYN Attack Prevention in Clustered Systems.

[0297] The systems and methods of the present solution illustrated in FIGS. **8**A-**8**B are directed towards protecting from SYN flood attacks in a cluster of networking devices via the generation, synchronization and use of a SYN-cookie for the cluster. As described above for a node having multiple cores, the node follows a master-slave concept to manage the task of maintaining the SYN-cookie same across the cores. Cores use shared memory to store the cookie. A packet engine on a core, such as a first packet engine, is designated a master packet engine. The master packet engine generates the cookie seed and writes to the shared memory at a predetermined frequency, such as every 120 secs. The other packet engines on the other cores read the seed at a predetermined frequency, such as 1 sec. from shared memory. Since the same seed is used by all the packet engines, SYN-cookie generated from the same side is valid across the cores.

[0298] The present solution addresses the use of SYN-cookies for clusters by providing SYN-cookie seed generation and synchronization across the nodes in cluster and generating SYN-ACK from the flow receiver of the cluster. For SYN-cookies seed generation and synchronization, the cluster may follow a similar master-slave mechanism of a multi-core device to generate and synchronize the SYN-cookies across the nodes in the cluster. A first packet engine on a master node may have the responsibility of generating and synchronizing the cookie seed. The master node pushes seed updates by broadcasting node to node messages to all the other nodes to update the seed on all the nodes. The master node may perform a push at a predetermined frequency, such as every 120 second, when the owner or master packet engine on the master generated the new seed. For updating the cores within the master node, the packet engine may write the new seed to the shared memory at the predetermined frequency. The packet engines on the other cores within node can only read the seed from shared memory. When master node sends node to node messages to the other nodes in the cluster, the message can land onto any core or packet engine in target node. The receiving packet engine may steer the message to the master packet engine for seed updates. The master packet engine updates the seed in shared memory, such as at the predetermined frequency or next predetermined frequency and the other cores read the seed from the shared memory. So as a result of one packet engine on a master node updating the seed, the other cores within the master node as well as the other nodes in the cluster and each of their respective cores are updated with the new seed.

[0299] Referring now to FIG. **8**A, an embodiment of a system for providing a cluster based cookie generation and synchronization is depicted. A cluster **600** may have a plurality of nodes (e.g., nodes 1-3) of appliances **200**A-**200**N (generally referred to as **200**). Each appliance **200** may comprise multiple cores **505**A-**505**N (generally referred to as **505**). Each core may operate or execute a packet processing engine **548**A-**548**N (generally referred to as PE **548**). Any core of packet processing engine may operate or execute one or more virtual servers **275**. One core or packet processing engine on a node may communicate with another core or packet processing engine via intercore communication **704**, such as shared memory or core to core messaging. Any of the nodes may implement any embodiments of the cookie generation and synchronization of cookies described in FIGS. **7**A-**7**C. For example, each node may implement the system **790**A-**790**N (generally referred to as **790**) described in FIG. **7**B for cookie generation and synchronization between cores. The system **790** may use shared memory **740** for reading and writing a random seed **706** for cookie generation and synchronization.

[0300] One node of the cluster, such as node **1** may be the master node or otherwise be the cluster configuration owner **810**. Each of the nodes may use node to node messaging (NNM) **835** to communication with each of the other nodes, such as to share a random seed **706**. One core in the node may receive the random seed from another node and use the system **790** to propagate or write the random seed **706** to shared memory **704** for other cores to obtain and use in accordance with the systems and methods of FIGS. **7**A-**7**C.

[0301] Each of the nodes in the cluster may be designed, constructed and configured to use the master and slave mechanism between cores within that node to generate and synchronize the seed across the packet engines of that node, such as via any embodiments of the systems and methods described in connection with FIGS. **7**A-**7**C. In some embodiments, each node may practice these embodiments before enabling the cluster instance or before creating a one node cluster.

[0302] Any node may be any type and form of multi-core device **100**. The device may be any type and form of multi-core device deployed as an intermediary device or appliance **200**. The device may include any embodiments of the multi-core appliance depicted and/or described in connection with FIGS. **5**A-**5**C. Each core or packet engine may establish transport layer connections over a network to one or more destinations, such as clients, and communicate over such connections with clients.

[0303] Each core may communicate with another core via inter-core communications **720**. Inter-core communications may include core to core messaging. Inter-core communications may include reading and/or writing to a shared memory. Any component on one core, such as a packet engine, may communicate with another component on another core via an inter-core communication. A packet engine on one core may communicate updates or changes to a random seed **706** to another packet engine on another code via inter-core communications.

[0304] In some embodiments, each appliance or node in the cluster may be a single processor appliance. In some embodiments, each appliance or node in the cluster may be a multi-core device. In the clustered system, each node may communicate with another node via a data plane, such as back plane **606** described in connection with FIG. **6**. Each node may communicate with other nodes via a data plane or black plane using an interface slave **610**. Each node may send node to node messages (NNM **735**) via the data plane or back plane. One of the appliances in the cluster **600** may be designated or identified as a master node or appliance. The master node or appliance may execute an interface master **608** for coordinating and managing the cluster.

[0305] The cluster and/or nodes in the cluster may be designed, constructed and/or configured to follow a master-slave mechanism to generate and synchronize the SYN-cookies across the nodes in the cluster. A packet engine on a core may be designated as the CCO or otherwise own the cluster configuration and may be responsible for generating and synchronizing the cookie seed. The master node may follow a push model or push message and broadcast NNM message all other nodes to update the seed on all the nodes. The master node may perform this push every predetermined frequency, such as every 120 sec., for example when a first packet engine or core on the master node or CCO node generates the new seed. On each node, one packet engine or core can only write to shared memory, such as the seed **706** to the shared memory and rest of the packet engines or core can only read, such as the seed **706** from the share message. When CCO node sends NNM message to other nodes, message can land onto any packet engine in target node. The receiving packet engine steers the message to the master packet engine or core, such as PEO.

[0306] In some embodiments, the cluster or a node in the cluster is designed, constructed and/or configured to handle different cluster views from the establishment of the cluster to new nodes joining or existing nodes leaving. In some embodiments, as soon as a cluster instance is enabled on one node (e.g., on creating one node cluster), the node by default becomes the cluster configuration owner (CCO) or master node **815**. The master node may be responsible for establishing, maintaining and propagating any of the configuration and changes thereto of the cluster. In some embodiments, the master node initializes the node to node NNM and inter-core communication channels. Once the master packet engine or core generates next seed **706** on the master node, the master node broadcasts this new seed to the other nodes in the cluster. When a new node joins the cluster, the master node may be designed, constructed and/or configured to receive the node join event and broadcast the current seed to the nodes. Until a new node receives the seed from the master node, the new node may be designed, constructed and/or configured remains in a SYN-cookie synchronization in progress state, in which the new node will be able to accept control connections but not the data connections. When the new node receives the seed, the new node generates the current cookie and previous current cookie becomes the previous cookie. In some embodiments, the new node will have the same current cookie as the master node but the previous cookie will be different from the previous cookie of the master node. As a result in some embodiments, that new node will not be able to validate the connections initiated using previous cookie. When the new node receives the seed the second time onwards, the new node's current and previous cookie will be the same as the current and previous cookie of the master node.

[0307] In some embodiments, when a node in a cluster boots up, the node generates the first seed and initializes that seed in all the cores. Similarly, in some embodiments, when each nodes comes up after joining the cluster, the nodes generates the first seed and initializes that seed in all the cores. In some embodiments, when a node leaves the cluster, the node falls back to the multi-core logic and embodiments of FIGS. **7**A-**7**C. In some embodiments, the node still continue to use the current and previous cookies received from the CCO node, until next trigger, such as the 2 min. trigger on the master core that generates new cookie.

[0308] In some embodiments, if the master or CCO node changes, one of the other nodes in the cluster declares itself CCO and will have the responsibility of generating and synchronizing the seed. The next node to take over as CCO may be established by configuration. The next node to take over as CCO may be established based on an order of joining the cluster. A master core on the new CCO node owns the responsibility of generating seed. All the nodes note the last seed receipt time from the previous CCO. When the new node becomes CCO, the master core of new CCO node sets new seed generation time as (last seed receipt time plus predetermined time period, such as 2 min.)

[0309] The nodes in the cluster may be designed, constructed and/or configured to use any type and form of NNM **835**. In some embodiments, the NNM may comprise a message containing the current seed generated by the master core or packet engine on the master or CCO node. In some embodiments, the NNM may comprise a message containing the current seed stored in shared memory by the master core or packet engine on the master or CCO node. The random seed may comprise a predetermined number of bits, such as 32. The NNM message may comprise the random seem and/or previous cookie and/or current cookie. The NNM may comprise any timing information regarding synchronization or generation of the seed and/or cookies from the seed, such as a change in the frequency of NNM pushes of a new side.

[0310] In some embodiment, any node in the cluster may be designed, constructed and/or make the decision to reply SYN-ACK or steer packet to flow receiver. Since the SYN-cookie is synchronized across the nodes in cluster, any node can reply the SYN-ACK back to SYN for a TCP handshake. The decision is made based on DFD establishment. A DFD session will be created only on the node which receives the final ACK packet. This DFD session will be present in only one node. A DFD session is used to determine the Flow Processor for the flow to which a packet belongs. The DFD session can be a 64-byte object (that contains the following pieces of information:

[0311] Source IPv4 address and destination IPv4 address of the flow

[0312] Source port and destination port of the flow

[0313] Node ID of the target node (Flow Processor)

[0314] Current (OVS) View ID of the cluster

[0315] Referring now to FIG. **8**B, an embodiment of a method of cookie generation and synchronization for a cluster is depicted. In brief overview, at step **830**, the CCO node of the cluster generates the random seed. At step **835**, the core of the CCO writes the seed to shared memory and other cores of the CCO node read the seed from shared memory. At step **840**, the CCO sends NNM messages comprising the seed to the other nodes in the core, At step **845**, a core of a node receiving the NNM determines the master core of that node and steers the NNM message or seed to that master core. At step **850**, the master core on each node writes the seed received from the CCO to shared memory and each of the other cores on that node read the seed from shared memory.

[0316] At step **830**, a master node or CCO node generates the global random seed for the cluster. The CCO may establish or generate the random seed in accordance with a predetermined frequency. The CCO may establish or generate the random seed responsive to a timer. The CCO may generate the random seed in accordance with any of the embodiments of step **730** described in FIG. **7**B.

[0317] At step **835**, the master core of the master node may write the generated random seed to the shared memory. The master core of the master node may write the random seed to shared memory upon generation of the random seed. The master core of the master node may write the random seed to shared memory responsive to a predetermined frequency or a long timer. The other cores of the master node may read the random seed from shared memory, such as responsive to a short timer or on a predetermined frequency. The other cores read the random seed from shared memory and use the random seed for generation of cookies as described in connection with FIGS. **7**A-**7**C. For example, each core may generate an array of cookie signatures by using the random seed as an initial seed of a pseudo-random number function to generate a first cookie signature, and generating each successive cookie signature by using a preceding cookie signature as a seed for the pseudo-random number function.

[0318] At step **840**, the CCO node or master node communicates the random seed to the other nodes of the cluster. The CCO node may transmit a node to node message to each of the other nodes in the cluster. The CCO node may transmit a NNM comprising the random seed to each of the other nodes. The CCO node may transmit the NNM to any core of the other node. The CCO node may transmit the NNM via the backplane of the node.

[0319] At step **845**, a core of the node receiving the NNM from the CCO node may be different from or not the master core on that node with write privileges to the shared memory for sharing the random seed with other cores. If the core is not the master core, the core forwards, steers or otherwise provides the NNM or the random seed therefrom to the master core. The receiving core may steer the packet via intercore communications or messages to the master core. If the core is the master core, the core identifies the random seed from the NNM message and uses this random seed for synchronization with the other cores on the same node.

[0320] At step **850**, the master cores writes the random seed, received via the CCO node, to the shared memory on that device. The master core may write the random seed to shared memory upon receipt. The master core may write the random seed to shared memory responsive to a long timer or in accordance with a predetermined frequency. The other cores read the random seed from shared memory responsive to a short timer. The master cores and the other cores may use the random seed for generation of cookies as described in connection with FIGS. **7**A-**7**C. For example, each core may generate an array of cookie signatures by using the random seed as an initial seed of a pseudo-random number function to generate a first cookie signature, and generating each successive cookie signature by using a preceding cookie signature as a seed for the pseudo-random number function.

[0321] Referring now to FIG. **8**C, another embodiment of a method of providing attack protection using the cookie generation and synchronization for a cluster is depicted. At step **870**, a packet received by a core of a node in a cluster of nodes. At step **872**, the receiving core determined if the packet is a steered packet. If not, perform a DFD session backup at step **876** and if it is, process the packet at step **874** to bypass the DFD lookup logic. At step **876**, if the session exists then at step **880** the core determines if the packet is a SYN packet and if so then replies with SYN-ACK packet at step **886**. If the session does not exist, the core determines at step **888** whether the packet is a PURE SYN packet, such as a valid and/or initial SYN packet and if yes then at step **886**, replies with a SYN-ACK packet. If it is not a pure SYN packet then at step **888**, the core determined with there packet is a SYN/ACK packet and if not whether or not at step **890**, the packet has a valid SYN-cookie. If there is a valid SYN cookie then at step **892**, the core establishes and allocates the DFD session. If it is a SYN-ACK packet at Step **888** or the SYN-cookie as step **890** is not valid or the packet is not a SYN packet at step **880** then the packet is steered at step **894**, such as to the node and/or core (e.g., flow processor) for the DFD session.

[0322] It should be understood that the systems described above may provide multiple ones 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 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, 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 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 software programs may be stored on or in one or more articles of manufacture as object code.

[0323] 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 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.

What is claimed is:

1. A method for synchronizing a random seed value among a plurality of multi-core nodes in a cluster of nodes for generating a cookie signature, the method comprising:

(a) generating, by a master core on a master node of a cluster of nodes comprising a plurality of cores, a random seed to be synchronized across each core of each node in the cluster of nodes;

(b) storing, by the master core on the master node, the random seed to memory on the master node accessible by each core in the master node;

(c) receiving, by each master core on each other node in the cluster, the random seed sent by the master core of the master node;

(d) storing, by each master core on each other node in the cluster, the random seed to memory on each node accessible by each core in each node; and

(e) generating, by each core of each node in the cluster of nodes, a cookie signature based on the random seed responsive to a predetermined timer.

2. The method of claim **1**, wherein (c) further comprises:

receiving, by a receiving core on each other node in the cluster, the random seed sent by the master core of the master node; and

steering, by each receiving core, the random seed to a master core in each other node in the cluster.

3. The method of claim **1**, further comprising

storing, at each core of each node in the cluster of nodes, a current cookie signature as a previous cookie signature, and the generated cookie signature as the current cookie signature.

4. The method of claim **1**, wherein (a) further comprises

generating, by the master core on the master node of the cluster of nodes, the random seed, responsive to a second predetermined timer set to expire longer than the predetermined timer.

5. The method of claim **1**, wherein (e) further comprises

generating, by each core of each node in the cluster of nodes, an array of cookie signatures.

6. The method of claim **1**, wherein (e) further comprises

generating, by each core of each node in the cluster of nodes, an array of cookie signatures, by using the random seed as an initial seed of a pseudo-random number function to generate a first cookie signature, and generating each successive cookie signature by using a preceding cookie signature as a seed for the pseudo-random number function.

7. The method of claim **6**, further comprising

generating a cookie by concatenating one or more cookie signatures in the array of cookie signatures.

8. The method of claim **1**, further comprising

using the generated cookie signature as part of a SYN cookie or a HTTP DoS cookie.

9. The method of claim **1**, further comprising:

receiving from a client, a SYN request at a first core of a node in the cluster;

responding to the client with a SYN-ACK message comprising a cookie with the cookie signature;

receiving from the client, an ACK message at a second core of the node in the cluster, the ACK message comprising a client cookie signature; and

accepting the ACK message in response to matching the client cookie signature with the cookie signature.

10. The method of claim **9**, further comprising

storing, at each core of each node in the cluster of nodes, a current cookie signature as a previous cookie signature, and the generated cookie signature as the current cookie signature;

comparing the client cookie signature with the current cookie signature and the previous cookie signature;

determining whether a new random seed is stored in a memory accessible by the second core;

storing, at the second core, the current cookie signature as the previous cookie signature;

generating a new current cookie signature based on the new random seed in the memory accessible by the second core; and allocating resources in response to matching the client cookie signature with the new current cookie signature.

11. A system for synchronizing a random seed value among a plurality of multi-core nodes in a cluster of nodes for generating a cookie signature, the system comprising:

a cluster of nodes, each node comprising a plurality of cores;

a master core on a master node of the cluster of nodes, configured to:

generate a random seed to be synchronized across each core of each node in the cluster of nodes; and

store the random seed to memory on the master node accessible by each core in the master node;

each other node in the cluster, configured to:

receive, by each master core of each node, the random seed sent by the master core of the master node; and

store the random seed to memory on each node accessible by each core in the each node; and

a packet engine on each core of each node in the cluster of nodes, configured to

generate a cookie signature based on the random seed responsive to a predetermined timer.

12. The system of claim **11**, wherein each other node in the cluster further comprises a receiving core configured to:

receive the random seed sent by the master core of the master node; and

steer the random seed to each node's master core.

13. The system of claim **11**, wherein each node in the cluster of nodes is further configured to

store, at each core of each node in the cluster of nodes, a current cookie signature as a previous cookie signature, and the generated cookie signature as the current cookie signature.

14. The system of claim **11**, wherein the master core of the master node is further configured to

generate the random seed to be synchronized across each core of each node in the cluster of nodes, responsive to a second predetermined timer set to expire longer than the predetermined timer.

15. The system of claim **11**, wherein the packet engine is further configured to generate an array of cookie signatures.

16. The system of claim **15**, wherein the packet engine is further configured to

generate, an array of cookie signatures, by using the random seed as an initial seed of a pseudo-random number function to generate a first cookie signature, and generate each successive cookie signature by using a preceding cookie signature as a seed for the pseudo-random number function.

17. The system of claim **16**, wherein the packet engine is further configured to

generate a cookie by concatenating one or more cookie signatures in the array.

18. The system of claim **11**, wherein the packet engine is further configured to

use the generated cookie signature as part of a SYN cookie or a HTTP DoS cookie.

19. The system of claim **11**, wherein the packet engine is further configured to:

receive from a client, a SYN request at a first core of a node in the cluster;

respond to the client with a SYN-ACK message comprising a cookie with the cookie signature;

receive from the client, an ACK message at a second core of the node in the cluster, the ACK message comprising a client cookie signature; and

accept the ACK message in response to matching the client cookie signature with the cookie signature.

20. The system of claim **19**, wherein the packet engine is further configured to

store a current cookie signature as a previous cookie signature, and the generated cookie signature as the current cookie signature;

compare the client cookie signature with the current cookie signature and the previous cookie signature;

determine whether a new random seed is stored in a memory accessible by a second core;

store, at the second core, the current cookie signature as the previous cookie signature;

generate a new current cookie signature based on the new random seed in the memory accessible by the second core; and

allocate resources in response to matching the client cookie signature with the new current cookie signature.

* * * * *