

**IN THE UNITED STATES DISTRICT COURT
FOR THE WESTERN DISTRICT OF TEXAS
MIDLAND/ODESSA DIVISION**

**INTELLECTUAL VENTURES I, LLC and
INTELLECTUAL VENTURES II LLC,**

Plaintiffs,

v.

SOUTHWEST AIRLINES CO.,

Defendant.

§
§
§
§
§
§
§
§
§
§

Civil Action No. 7:24-cv-00277-ADA

JURY TRIAL DEMANDED

**DEFENDANT SOUTHWEST AIRLINES CO.'S
OPENING CLAIM CONSTRUCTION BRIEF FOR U.S. PATENT NOS.
7,721,282, 8,332,844, 7,949,785, AND 8,407,722**

TABLE OF CONTENTS

I. INTRODUCTION..... 1

II. LEGAL STANDARDS 1

 A. Lexicography Standards..... 2

 B. Prosecution Disclaimer Standards..... 2

III. PERSON OF ORDINARY SKILL IN THE ART 3

IV. DISPUTED CONSTRUCTIONS 3

 A. U.S. Patent Nos. 7,721,282 and 8,332,844..... 3

 1. “root image” (’844 Patent at claim 7; ’282 Patent at claims 1 and 15)..... 5

 2. “leaf images” (’844 Patent at claim 7; ’282 Patent at claims 1 and 15)..... 7

 3. “merging the blocks of said root image with the blocks of respective leaf images to create cohesive respective application environments” (’844 Patent at claim 11; ’282 Patent at claim 15)..... 9

 4. “union block device” (’282 Patent at claims 1 and 15)..... 11

 A. U.S. Patent No. 7,949,785 12

 1. “network address” (claim 30)..... 13

 2. “distribute a virtual network address to the device” (claim 30)..... 14

 3. “network route director” (claim 30)..... 15

 B. U.S. Patent No. 8,407,722 17

 1. “respond to the live object of the data representation by determining an object identifier (ID) of the live object” (claim 14) 17

 2. “register for updates of the live object with the routing network” (claim 14) 19

 3. “input source” / “identify a category of the update message based on the input source” (claim 14)..... 20

TABLE OF AUTHORITIES

	Page(s)
Cases	
<i>AstraZeneca LP v. Apotex, Inc.</i> , 633 F.3d 1042 (Fed. Cir. 2010).....	2, 7
<i>Aylus Networks, Inc. v. Apple Inc.</i> , 856 F.3d 1353 (Fed. Cir. 2017).....	2
<i>Bell Atl. Network Servs., Inc. v. Covad Commc'ns Group, Inc.</i> , 262 F.3d 1258 (Fed. Cir. 2001).....	2
<i>Computer Docking Station Corp. v. Dell, Inc.</i> , 519 F.3d 1366 (Fed. Cir. 2008).....	3
<i>MBO Lab'ys, Inc. v. Becton, Dickinson & Co.</i> , 474 F.3d 1323 (Fed. Cir. 2007).....	2, 3
<i>O2 Micro Int'l Ltd. v. Beyond Innovation Tech. Co.</i> , 521 F.3d 1351 (Fed. Cir. 2008).....	2
<i>Omega Eng'g, Inc. v. Raytek Corp.</i> , 334 F.3d 1314 (Fed. Cir. 2003).....	2
<i>Phillips v. AWH Corp.</i> , 415 F.3d 1303 (Fed. Cir. 2005) (en banc).....	1, 2, 3
<i>Purdue Pharma L.P. v. Endo Pharms. Inc.</i> , 438 F.3d 1123 (Fed. Cir. 2006).....	3
<i>Southwall Techs., Inc. v. Cardinal IG Co.</i> , 54 F.3d 1570 (Fed. Cir. 1995).....	2

I. INTRODUCTION

Intellectual Ventures (“IV”) has built a licensing campaign around patents that span unrelated technologies—data storage, cloud computing, and wireless communications—and target industries with little in common beyond their reliance on common open-source software and networking solutions. Because many industries use the same software tools, IV has asserted untested patents against companies as varied as airlines, retailers, insurers, and banks, sometimes based on little more than a LinkedIn reference to experience with specific open-source technologies. Grouping such disparate patents in a single action magnifies the complexity of the case and makes careful claim construction difficult but essential.

IV contends that *none* of the terms in any of the Asserted Patents requires construction, advocating instead for “plain meaning” interpretations that would allow its experts to argue claim scope in reports and at trial. That approach should not be permitted where the specifications and prosecution histories include definitional language and disclaimers that limit the meanings of key terms. Because claim construction is a question of law for the Court—not a factual dispute for experts—the Court should resolve the parties’ disputes now rather than allow competing claim interpretations to reach the jury.

II. LEGAL STANDARDS

The purpose of claim construction is to give claim terms the meaning understood by a person of ordinary skill in the art at the time of invention. *Phillips v. AWH Corp.*, 415 F.3d 1303, 1312-14 (Fed. Cir. 2005) (en banc). The plain meaning of claim language ordinarily controls *unless* the patentee acts as his own lexicographer and provides a special definition for a particular claim term or the patentee disavows the ordinary scope of a claim term either in the specification or during prosecution. *Phillips*, 415 F.3d at 1313. Even if a patent uses a term’s plain meaning, the Court must resolve any genuine dispute regarding the scope of that meaning, to prevent the parties and

their experts from presenting competing claim construction arguments to the jury. *O2 Micro Int'l Ltd. v. Beyond Innovation Tech. Co.*, 521 F.3d 1351, 1360–62 (Fed. Cir. 2008).

A. Lexicography Standards

“[T]he specification may reveal a special definition given to a claim term by the patentee that differs from the meaning it would otherwise possess. In such cases, the inventor’s lexicography governs.” *AstraZeneca LP v. Apotex, Inc.*, 633 F.3d 1042, 1051–52 (Fed. Cir. 2010) (quoting *Phillips*, 415 F.3d at 1316). The specification need not reveal such a definition explicitly. *Id.* (citing *Bell Atl. Network Servs., Inc. v. Covad Commc’ns Group, Inc.*, 262 F.3d 1258, 1268 (Fed. Cir. 2001)). “[W]hen a patentee uses a claim term throughout the entire patent specification in a manner consistent with only a single meaning, he has defined that term ‘by implication.’” *Id.* (quoting *Bell Atlantic*, 262 F.3d at 1271).

B. Prosecution Disclaimer Standards

“Prosecution disclaimer ‘preclud[es] patentees from recapturing through claim interpretation specific meanings disclaimed during prosecution.’” *Aylus Networks, Inc. v. Apple Inc.*, 856 F.3d 1353, 1359 (Fed. Cir. 2017) (quoting *Omega Eng’g, Inc. v. Raytek Corp.*, 334 F.3d 1314, 1323 (Fed. Cir. 2003)). “[T]he doctrine of prosecution disclaimer ensures that claims are not ‘construed one way in order to obtain their allowance and in a different way against accused infringers.’” *Id.* at 1360 (quoting *Southwall Techs., Inc. v. Cardinal IG Co.*, 54 F.3d 1570, 1576 (Fed. Cir. 1995)). Disclaimer may occur through amendment or argument. *Id.* at 1359.

Statements disclaiming scope must be clear and unmistakable. *Omega*, 334 F.3d at 1326. Disclaimer statements include arguments made to convince the Patent Office of patentability, including when the patentee “draw[s] distinctions between the patented invention and the prior art.” *MBO Lab’ys, Inc. v. Becton, Dickinson & Co.*, 474 F.3d 1323, 1330 (Fed. Cir. 2007). Such arguments show “the patentee intended to surrender territory, since they indicate in the inventor’s

own words what the invention is not.” *Id.*; *see also Computer Docking Station Corp. v. Dell, Inc.*, 519 F.3d 1366, 1374 (Fed. Cir. 2008); *Purdue Pharma L.P. v. Endo Pharms. Inc.*, 438 F.3d 1123, 1136 (Fed. Cir. 2006).

III. PERSON OF ORDINARY SKILL IN THE ART

A court determines the proper construction of a claim term from the viewpoint of the hypothetical person of ordinary skill in the art (“POSITA”). *Phillips*, 415 F.3d at 1313. The Asserted Patents relate to computational devices and networking. A person of ordinary skill in the art as of the respective priority dates for the Asserted Patents would have had a bachelor’s degree in electrical engineering, computer science, or the like, and/or two or more years of industry experience in networking and distributed computing. Additional experience may make up for a lack of education and *vice versa*.

IV. DISPUTED CONSTRUCTIONS

A. U.S. Patent Nos. 7,721,282 and 8,332,844

The ’282 Patent (Ex. A-1) and the ’844 Patent (Ex. A-2) share a common specification lineage and focus organizing and storing information in distributed application environments.

The ’282 Patent—titled *Block-Level I/O Subsystem for Distributed Computing Environment Management*—describes a system “for distributing an application environment to a compute node (e.g., a server or a thin client workstation)” that includes (i) “a first storage unit for storing [data] blocks of a root image (e.g., an operating system) of the compute node,” (ii) “a second storage unit for storing a leaf image comprising new data blocks and changes to the blocks of the root image,” and (iii) “a union block device for interfacing between the compute node and the first and second storage units to distribute the application environment to the compute node.” ’282 Patent at 2:16–27 (*see also* Fig. 3 flow). “The union block device creates the application environment by merging the blocks of the root image stored on the first storage unit with the leaf image stored on the second

storage unit.” *Id.* at 2:27–30. The “union block devices,” or UBDs, “are effectively low-level drivers that operate as an interface between the first and second storage devices and the file system of each compute node.” *Id.* at 4:41–45. Because these UBDs “operate below the file system, they are concerned merely with the blocks of data themselves, rather than files they form,” making the system “completely file system, and thus operating system, independent.” *Id.* at 4:48–51.

The ’844 Patent—titled *Root Image Caching and Indexing for Block-Level Distributed Application Management*—is a continuation-in-part of the ’282 and recites additional elements “for ... block-level distributed application management,” including “caching blocks of the root image that have been accessed by at least one compute node” and “receiving indexing results pertaining to the root image from one compute node and providing the results for other compute nodes.” ’844 Patent at 2:35–48. Like the ’282 Patent, the ’844 Patent relies on *block*-layer flows (e.g., storing blocks of the root image and leaf images, merging blocks, and delivering the environment via a low-level driver), explaining that “[b]y storing data at the block level, embodiments are able to operate beneath the file system and thus are designed to be file system and operating system independent.” *Id.* at 7:58–62, Fig. 3A/3B. Because “the merging occurs at an operational level between the file system ... and the [root, leaf, and cache] storage,” the compute node sees “one cohesive image rather than a base image plus its additions, deletions, and modifications.” *Id.* at 8:18–26. “Thus, the file system actually seen by the compute node is a product of merging the root image portion in the leaf image portion together at the block level.” *Id.* at 9:44–47. “Thus, by operating at the block level, embodiments of the present disclosure provide file system and operating system independent systems and methods for distributing an application environment to a compute node.” *Id.* at 10:1–4.

The '282 and '844 Patents trace their lineage to U.S. Application No. 11/026,622 (the "'622 Application"), entitled *Branching Store File System*, (see Ex. A-3), which disclosed the same general concept of distributing application environments, but implemented at the *file system* level. In the '662 Application, "[a] filter operates between the compute nodes and the file system(s), which merges the changes recorded on the leaf images with the root image and delivers the result to the appropriate compute node," so that it appears to each compute node that "it is running its own unique and cohesive instance of the application environment." '282 Patent at 1:7–11, 1:58–2:12; '844 Patent at 1:7–13, 2:13–24. A compute node "mounts at least a portion of the base image ... and the change volume," and writes are handled via file-level copy-on-write ("the entire original file is copied from the root image ... into the change volume ... [and] the write operation is then performed on the copy"). Ex. A-3 at [0037]–[0038].

In such embodiments, "a separate version of the system must be created for each unique operating system because data is stored at the file system level (i.e., on a 'per file basis')." '844 Patent at 2:24–32. The '662 Application teaches a file-system-level overlay (filter-mediated, mount-based, file-level copy-on-write), whereas the asserted '282/'844 claims are directed to block-layer operation under the file system (union block device/low-level driver, block/sector merging). The patentees' and the patents' distinction between the "layers" at which data is stored and merged is the crux of the parties' claim construction disputes.

1. "root image" ('844 Patent at claim 7; '282 Patent at claims 1 and 15)

Defendant	Plaintiff
a base image of the application environment operating at the block level below the file system	plain meaning

The parties agree that a "root image" is a base image of the application environment; the parties dispute whether it operates at the block layer, below the file system level.

The claim language fixes that layer. Claim 15 of the '282 patent recites “storing blocks of a root image of a compute node on a first storage unit,” where “the blocks comprise sections of data, and ... a file of the root image comprises at least one block,” and then “merging the blocks of the root image ... with the blocks of the leaf image ... to create the application environment,” with sector-level handling by the driver. Claim 7 of the '844 Patent likewise recites “storing blocks of a root image of said compute nodes on a first storage unit,” and dependent claim 11 recites “merging the blocks of said root image with the blocks of respective leaf images to create cohesive respective application environments.” As discussed below, by drafting the claims to focus on storing and merging *blocks* of the root image (and leaf), the patentee confirmed that the root image is manipulated at the block level, *not* at the file system level.

The specifications clarify and cement the block-layer operation. Figure 2 of the '844 Patent is “a diagram of a system 200 for root image caching and indexing in a block-level distributed application environment,” '844 Patent at 5:12–14, and the '282 Patent presents the same architecture in Figure 1. Figure 3A of the '844 Patent depicts a flowchart for root image caching and indexing “in a block-level distributed application environment,” and the '282 Patent’s Figure 3 and corresponding text (5:61–63) similarly illustrate storing blocks of a root image and merging those blocks with leaf-image blocks to create the application environment. The figures and the accompanying descriptions in the specification teach block-layer operations.

The specification contrasts block-level operations on the root image with file-level operations, explaining with a concrete example that a particular file in the root image may comprise twenty blocks of data, and that modifying only a subset of those blocks results in storing only the changed blocks in the leaf image, plus minimal overhead. *See* '282 Patent at 4:33–40; '844 Patent at 5:51–58. That is block-granularity composition, not file-level copy-up.

The patents repeatedly emphasize that the claimed inventions “operate beneath the file system” and are “file system and operating system independent.” *See, e.g.*, ’282 Patent at 4:45–51, 5:67–6:2, 6:38–41; ’844 Patent at 5:63–6:3, 7:58–62, 10:1–4. The teachings are explicit: “[b]y storing data at the block level, embodiments are able to operate beneath the file system and thus are designed to be file system and operating system independent.” ’282 Patent at 5:67–6:2; ’844 Patent at 7:58–62. The patents present the block-layer operations as a deliberate departure from earlier, file-system-level approaches described in the ’662 Application, and tout that it “provid[es] file system and operating system independent systems and methods for distributing an application environment to a compute node.” ’282 Patent at 5:67–6:2, 6:38–41; ’844 Patent at 7:59–62, 10:1–4. Because the claims and the specification define the operative level as *block-level*—i.e., beneath the file system—that distinction controls. *See AstraZeneca*, 633 F.3d at 1051–52.

Because the intrinsic record consistently ties the handling of the root image to block-level storage and merging beneath the file system, the term cannot encompass file system-level implementations. Permitting IV to suggest otherwise would invite improper claim construction before the jury. The Court should therefore construe “root image” to mean *a base image of the application environment operating at the block level below the file system*.

2. “leaf images” (’844 Patent at claim 7; ’282 Patent at claims 1 and 15)

Defendant	Plaintiff
a node-specific image of the application environment operating at the block level below the file system	plain meaning

The parties’ dispute whether by claiming block-layer storage of leaf images, the patentees distinguished systems operating at the file system layer. The claim language should be dispositive on this issue. For example, claim 7 of the ’844 Patent requires “storing leaf images for respective nodes on respective second storage units,” and defines “said leaf images [as] including only

additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of the root image, wherein said leaf images of respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes.”

The claim language thus dictates block-layer operations. Dependent claim 11 confirms this by requiring “merging the blocks of said root image with the blocks of respective leaf images to create cohesive respective application environments,” underscoring that the merge operation occurs at the block layer and that each leaf image produces its own per-node application environment when merged. The ’282 Patent mirrors the block-layer architecture: system claim 1 recites “a second storage unit for storing a leaf image, the leaf image comprising new data blocks and changes to the blocks of the root image,” and method claim 15 requires “storing a leaf image comprising new data blocks and changes to the blocks of the root image” and “merging the blocks of the root image with the blocks of the leaf image” to “create the application environment” delivered “to the compute node.” Thus, the claims specify (i) the operative unit is *blocks* (not files), and (ii) each leaf image is *node-specific* (“respective ... compute nodes”; “respective leaf images”; “cohesive respective application environments”).

The specifications confirm “when a compute node ... makes changes involving the root image, modifications are made to that compute node’s leaf image,” and that “only the specific blocks that are changed are stored in the leaf image.” ’282 Patent at 4:31–40; ’844 Patent at 5:49–58. The patents illustrate this with an example: if a file in the root image comprises twenty blocks and the node modifies only blocks 4–9, then only blocks 4–9—along with minimal overhead—are stored in that node’s leaf image, not the unchanged blocks 1–3 and 10–20. That is block-level copy-on-write behavior to avoid duplication of unchanged data.

The patents further teach a union block device—a low-level driver—that “operat[es] beneath the file system,” merges blocks of the root and leaf images, and presents the compute node with “one cohesive image” that is “file system and operating system independent.” *See, e.g.*, ’282 Patent at 4:45–51, 5:67–6:2, 6:38–41; ’844 Patent at 5:63–6:3, 7:58–62, 10:1–4. By storing and merging at the block level, the inventions deliberately depart from earlier file-system-level “branching store” approaches disclosed in the ’662 Application, which used a filter and mount-based, file-level copy-up semantics and required per-OS implementations. In the asserted patents, by contrast, the merger occurs at the block layer between storage and the file system; the file system sees only the unified result. Within this design, a “leaf image” is necessarily a block-level construct.

Any broader reading would conflict with the claims’ “only” limitation and their express exclusion of unchanged root blocks, and would erase the patents’ central distinction from the prior file-system-level overlay. Nothing in the intrinsic record supports a file-level implementation or a leaf image that copies unmodified root data. The Court should therefore adopt Southwest’s construction: a “leaf image” is *a node-specific image of the application environment operating at the block level below the file system*.

3. “merging the blocks of said root image with the blocks of respective leaf images to create cohesive respective application environments” (’844 Patent at claim 11; ’282 Patent at claim 15)

Defendant	Plaintiff
combining the root image and the leaf image at the block level below the file system	plain meaning

Claim 11 of the ’844 Patent depends from claim 7 and recites “merging the blocks of said root image with the blocks of respective leaf images to create cohesive respective application environments.” Similarly, claim 15 of the ’282 Patent requires “merging the blocks of the root image stored on the first storage unit with the blocks of the leaf image stored on the second storage

unit to create the application environment.” Read in light of the specifications, these limitations require that the claimed merging occur beneath the file system—i.e., at the block level—thereby enabling file system and operating system independence.

The specifications describe the purported invention as improving cluster computing “by utilizing a root-leaf system application environment storage, [which] allow[s] creation of boot images on the fly without significantly diminishing bring-up time.” ’844 Patent at 2:52–56; ’282 Patent at 2:46–53. They explain that “creating a new boot image does not require copying the contents of the root image,” but instead “involves registering a new UBD with the system, which occurs very quickly.” ’844 Patent at 2:40–65; ’282 Patent at 2:53–55. As described, UBDs are low-level drivers operating at the block level beneath the file system. The specifications confirm: “Because UBDs ... operate below the file system, they are concerned merely with the blocks of data themselves, rather than files they form. As a result, system [] is completely file system, and thus operating system, independent.” ’844 Patent at 5:66–6:3; ’282 Patent at 4:48–51. Put differently, “the file system actually seen by the compute node is a product of merging the root image portion and the leaf image portion together at the block level.” ’844 Patent at 9:44–47.

Non-asserted claim 23 of the ’844 Patent explicitly states what the other claims implicitly require—that “merging” occurs “*at the block-level.*” *Id.* at 13:15–16. The flow diagrams confirm Defendant’s point. Step 330 of Figure 3A of the ’844 Patent shows *block-level* merging of the root and leaf images to create an application environment. ’844 Patent at 8:13–15. Likewise, the ’282 Patent’s Figure 3 and corresponding text describe “a process 300” showing “stor[ing] blocks of a root image of a compute node” and “merg[ing] the blocks of the root image with the blocks of the leaf image to create an application environment.” ’282 Patent at 5:61–63, Fig. 3. The result is that while the compute node “has access to its own unique version of an application,” “a separate and

complete boot image is not actually stored for the compute node.” ’844 Patent at 8:18–25; ’282 Patent at 6:13–16. Because both the claim language and the specifications repeatedly and unambiguously teach that the root and leaf images are combined at the block level beneath the file system, the Court should adopt Southwest’s proposed construction.

4. “union block device” (’282 Patent at claims 1 and 15)

Defendant	Plaintiff
drivers operating at the block level below the file system	plain meaning

Claim 1 of the ’282 Patent recites “a union block device for interfacing between the compute node and the first and second storage units to distribute the application environment to the compute node, wherein the union block device comprises a driver,” and further specifies that “the union block device creates the application environment by merging the blocks of the root image stored on the first storage unit with the blocks of the leaf image stored on the second storage unit.” This language confirms that the union block device is a driver operating beneath the file system, at the block level, to merge root and leaf blocks.

The patent characterizes the union block device (UBD) as a low-level, block-layer mechanism, explaining that the “union block device creates the application environment by merging the blocks of the root image stored on the first storage unit with the leaf image stored on the second storage unit.” ’282 Patent at 2:27–31. Fig. 1 depicts the UBD between the compute node and the storage units, mediating access at the block level. *Id.* at 3:64–4:8. Critically, the patent expressly distinguishes UBDs from file-based approaches: “Because UBDs ... operate below the file system, they are concerned merely with the blocks of data themselves, rather than files they form.” *Id.* at 4:48–50. The same theme is reiterated throughout: “By storing data at the block level, embodiments are able to operate beneath the file system and thus are designed to be file system and operating system independent.” ’844 Patent at 7:58–62; *see also* ’282 Patent at 6:39–42

(“Thus, by operating at the block level, embodiments ... provide a file system and operating system independent system and method for distributing an application environment to a compute node.”).

The prosecution history unequivocally confirms this construction. During prosecution, the patentees amended claim language to “further define the union block device as a ‘low level driver,’” citing paragraph [0020] of the specification as support and explaining that such low-level drivers operate on the individual blocks that make up files (data) within the root and leaf images, as recited. *See* Ex. A-4 at 8. This amendment and the accompanying remarks limit the UBD to a block-level driver, not a file-level or higher-level component.

In view of the claims, specification, and prosecution history, “union block device” should be construed as drivers operating at the block level below the file system.

A. U.S. Patent No. 7,949,785

The '785 Patent (Ex. A-5)—titled *Secure Virtual Network System*—describes a virtual networking architecture that enables devices distributed across public and private networks (including behind NATs and firewalls) to communicate as if on a common private network. At a high level, a virtual network manager registers devices (via an agent), assigns each device a virtual address that identifies it within the virtual community, and provides name-resolution responses that supply the routing information needed to reach peers. *See, e.g.*, '785 Patent at Abstract, 5:48–60, 6:27–33, 12:8–14, Fig. 4. A core aspect is the Domain Name System (DNS) server interaction: when a first device requests communication with a second device, the manager’s DNS server returns three data points—(1) a public-side IP address for a network route director that will forward traffic, (2) a private address for the destination device (when applicable), and (3) the destination’s virtual address used for identification within the virtual community. *See, e.g.*, '785 Patent at 12:26–31, 14:34–38, 14:43–46, 31:28–32, Fig. 7, Fig. 24. Devices communicate using encapsulation/security appropriate to the overlay, but the addressing distinction—

physical/public/private versus virtual—is fundamental to the claimed architecture. *See, e.g., id.* at 12:50–62, 13:25–27.

1. “network address” (claim 30)

Defendant	Plaintiff
Internet Protocol (IP) address	plain meaning

As described above, claim 30 identifies three distinct addresses returned by the DNS server: a “network address” for the network route director, a “private network address” for the destination device, and a “virtual network address” for that device. ’785 Patent at 36:50–56. The question here is what the term “network address” itself denotes.

The specification and figures make clear that the “network address” is the public-side IP address used to reach the route director. Figure 7 (step 656) depicts the DNS response returning the public IP address for the route director, while Figures 7 and 24 consistently identify the returned “IP address for RD.” *See id.* at 14:34–38, 14:43–46, 31:28–32. The patent never describes any non-IP form of network address, and the DNS context itself presupposes IP addressing.

The prosecution record reinforces this reading. In reexamination, the patentees explained that the DNS response includes a numeric IP address, describing the familiar DNS translation from a domain name to its corresponding IP address, telling the patent office that “the response from the DNS server includes a numeric IP address.” Ex. A-6 at 12–13. The patentee explained: “A DNS request is responsible for resolving a domain name, such as www.example.com, into its corresponding IP address (e.g., 93.184.216.34) to enable devices to locate and communicate with the server hosting the desired resource.” *Id.* at 12. Those statements confirm that “network address” means an IP address associated with the route director.

That is consistent with claim 30’s use of three different terms—“network address,” “private network address,” and “virtual network address”—to reflect three different concepts. The intrinsic

confirms the “network address” is a routable IP address for the route director, the “private network address” is the local address within a private domain, and the “virtual network address” is an identifier within the overlay.

2. “distribute a virtual network address to the device” (claim 30)

Defendant	Plaintiff
assigning a virtual address, different from an internet protocol or IP address, to identify the device within the virtual network	plain meaning

Claim 30 requires that the register module “distribute a virtual network address to the device when the device is registered in the virtual network.” ’785 Patent at 36:43–47. The specification makes clear that this “virtual network address” is an identifier in a separate, non-routable logical space—the virtual address realm—used by members of the virtual community to identify one another, as opposed to public or private IP addresses used for physical routing. *See id.* at 12:50–62, 13:25–27, 5:48–60, 6:27–33. The patent distinguishes between (i) physical addressing (public/private IP) used for routing on underlying networks, and (ii) virtual addressing used for member identification within the overlay. *See id.* at 12:50–62, 13:25–27.

Claim 30 recites three items returned by the DNS server: “a network address” for the route director, “a private network address” for the destination device, and “a virtual network address” for that device. *Id.* at 36:50–56. The figures depict the same three fields. *See id.* at 14:34–46, 31:28–32, Fig. 7, Fig. 24. Treating “virtual network address” as just another IP address would collapse the triad and render claim language superfluous.

The specification explains that by assigning each member a virtual address, the system “creates” a virtual address realm that members use to identify and send communications to each other, while those same members can simultaneously inhabit different physical (IP) realms. *See id.* at 12:50–62. The point of this realm is to avoid conflicts with routable IP space and to enable

overlay communication independent of physical addressing. *See id.* at 13:25–27 (non-routable virtual addresses). That design only works if the distributed “virtual network address” is different in kind from IP.

Dependent claims reinforce that the distributed virtual address is tied to a specific device’s identity within the community. Claim 35’s join/leave logic and claim 37’s maintenance of an association between a device and its virtual address confirm that the address is a persistent overlay identifier for that device, not a routable IP. *See id.* at claims 35, 37; *id.* at 36:57–37:12.

During prosecution, the patentees distinguished prior art that relied on corporate IPs or private non-routable IPs by emphasizing that the claimed approach configures a client with a virtual network address by which members identify one another in the community, in addition to any physical address. *See Ex. A-7* at 20. That distinction is incompatible with a “virtual network address” being an IP address.

A “plain meaning” construction is unhelpful for a term that the patent itself cabins within a separate addressing realm and that the claims treat as a different field from both “network address” and “private network address.” *See id.* at 36:46–56. The most natural, claim-consistent construction is the one Southwest proposes: the register module assigns a virtual address—not an IP address—to identify the device within the virtual community.

3. “network route director” (claim 30)

Defendant	Plaintiff
a publicly addressable device configured to route encapsulated packets to and from entities located in a private network portion of a virtual network	plain meaning

Claim 30 requires the DNS server to “return a network address associated with a network route director.” The term “Network Route Director” (NRD) is coined and has no ordinary technical meaning apart from the specification; thus, its scope is defined by the patent’s own usage. *See* ’785

Patent at 10:24–27, 11:9–18. The specification consistently characterizes the NRD as a device that “runs on the public side” of the Internet/NAT and enables communication with endpoints located in private address domains without firewall reconfiguration, forwarding encapsulated overlay traffic into and out of those private realms. *See id.* at 11:9–13, 13:51–59, 25:33–34, 32:53–59; *see also id.* at 15:6–7 (public IP address for the NRD included during setup).

The patent teaches that the NRD is publicly reachable—“on the public side”—and is identified to the communicating member by a public-side address returned in the DNS response. *See id.* at 14:34–46, 31:28–32, Fig. 7, Fig. 24. The NRD forwards encapsulated overlay traffic into and out of private address realms and performs re-encapsulation associated with traversal through NAT/firewalls. *See id.* at 13:51–59, 27:13–15, 28:26–27. Claim 30 and the figures depict the DNS server returning the NRD’s public address together with the destination device’s private and virtual addresses, positioning the NRD as the publicly addressed relay for reaching a device identified in the virtual address realm. *See id.* at 36:50–56, Fig. 7, Fig. 24.

The patentees confirmed this function during reexamination, stating “the problem of the current invention . . . is how to *route* through a route director to the virtual address.” Ex. A-6 at 19 (emphasis added). This characterization of “the current invention” confirms that “network route director” must route packets to a virtual address. This is not a generic router or load balancer; it is the public-side relay for encapsulated overlay traffic directed to a device identified by a virtual address within the community.

Because “Network Route Director” is a coined term, a plain-meaning construction divorced from the specification is inappropriate. The intrinsic evidence supplies boundaries that require construction of NRD as *a publicly addressable device that routes encapsulated packets to and from entities located in a private network portion of the virtual network.*

B. U.S. Patent No. 8,407,722

The '722 Patent (Ex. A-8)—titled *Asynchronous Messaging Using a Node Specialization Architecture in the Dynamic Routing Network*—concerns “transferring information for remotely updating content at client devices through the digital networks.” ’722 Patent at 1:24–27. The patent describes a publish-subscribe messaging system in which client devices receive web pages containing “live objects,” which are “real-time dynamically-updateable objects ... identified by an object identifier, or object ID.” *Id.* at 5:27–31. “When the clients receive live objects, the clients identify the object IDs associated with the objects and register the object IDs with the routing network.” *Id.* at Abstract. The routing network “maintains a registry indicating which clients have registered for which object IDs,” and selectively forwards update messages to those clients. *Id.* at 5:32–48. When a gateway receives update messages from an input source, “the gateway identifies the categories of the messages and routes the messages to the nodes of the types to which the categories are mapped.” *Id.* at 3:43–46; *see also id.* at Abstract.

1. “respond to the live object of the data representation by determining an object identifier (ID) of the live object” (claim 14)

Defendant	Plaintiff
parse the data representation to identify an object identifier of the live object	plain meaning

The dispute is whether the claimed method requires the *client device* to examine the “data representations” to identify the object ID of the live object. Rather than adopting a “plain meaning” construction and leaving it to the experts to argue claim construction to the jury, the Court should construe this phrase to require the client device to parse the provided data representation (e.g., a web page or other markup-based document) to determine the object ID of a live object.

Claim 14 recites providing “a data representation to a client device,” where the “data representation includes at least one live object recognizable by the client device,” and the client

device “*respond[s] to the live object of the data representation by determining an object identifier (ID) of the live object ... and [registers] for updates of the live object with the routing network.*”

The specification uses “web page” broadly to encompass “other representations of data served by the server ... regardless of whether these data representations include characteristics of conventional web pages,” including “application programs ... such as spreadsheet or textual (e.g., word processing) data, etc.” *Id.* at 6:39–60. Thus, the “data representation” is the client-side structure (a web page or other delivered representation) that can contain live objects.

The specification explains what “live objects” are and how their identifiers appear in the provided representation: “The web pages contain one or more ‘live objects,’ which are designated to be real-time dynamically-updateable objects. Each live object is identified by an object identifier, or object ID.” *Id.* at 5:27–31. In concrete embodiments, “the information provider 108 encodes the object ID in an object’s corresponding HTML ‘ID’ attribute,” and “the object ID identifies a point (i.e., a node in a tree) in a Document Object Model (DOM) representation of a web page or other document at the client 114.” *Id.* at 8:28–30, 9:13–15. Clients “do not need to have any a priori knowledge of object IDs—they ‘discover’ which IDs they should register when they receive the web pages 118 from the server 112.” *Id.* at 5:66–6:2.

Critically, the specification describes how the client determines the identifiers: “to register object IDs, the activation module 124 preferably parses 610 the web page 118 received from the server 112 and identifies the objectIDs of the live objects.” *Id.* at 14:28–43. The same passage makes clear this is a client-side operation performed by the activation module and depicts it at box 610 of Figure 6, and it also lists variants that still involve the client identifying which IDs to register: the module may “identif[y] only a subset of the objectIDs,” “a list of object IDs may be

pre-encoded in the web page ... thereby enabling easy identification by the activation module,” or “a user of the client 114 selects the object IDs to register.” *Id.*

The intrinsic record therefore confirms that the limitation requires the client device to parse the provided data representation to identify the embedded object ID(s). Southwest’s construction captures that operation and resolves the parties’ dispute; leaving “plain meaning” to expert testimony would improperly defer claim scope to the jury.

2. “register for updates of the live object with the routing network” (claim 14)

Defendant	Plaintiff
communicate to the routing network a vector containing the identified object identifier	plain meaning

The parties dispute whether this registration limitation requires the client to communicate the identified object ID to the routing network.

Claim 14 requires that the “client device,” after identifying an object ID of the live object, “register for updates of the live object with the routing network.” This is the same thing described in the abstract where the client receives a live object and in response the client identifies the object ID associated with the object and “register[s] the object IDs with the routing network.” ’722 Patent at Abstract. The specification explains that “[e]ach live object is identified by an object identifier, or object ID,” and the routing network “maintains a registry indicating which clients have registered for which object IDs.” *Id.* at 5:30–31, 5:34–36.

The specification teaches that “[t]he client 114 executes 220 the activation module 124, which identifies the object IDs of the live objects at the client and registers 222 the object IDs with the routing network 110.” *Id.* at 11:33–35. Figures 2 and 6 depict the same two-step client-side operation—identify the IDs, then register them with the network—confirming that “register” is an active, client-initiated transmission.

The specification teaches that the activation module “uses the connection between the client 114 and routing network 110 to register 614 the object IDs by communicating to the routing network 116 *a vector (e.g., a list or array) containing the identified object IDs.*” *Id.* at 14:55–59 (emphasis added). To traverse a firewall, “*the activation module packages the vector containing the identified object IDs* into a string, referred to as ‘object data,’ and sends it to the routing network as part of an HTTP message.” *Id.* at 14:59–63 (emphasis added). Upon receipt, “the routing network ... extracts the object data and updates the registry to indicate that the client has registered for the identified objects.” *Id.* at 15:1–4. In other words, in this claim, “register ... with the routing network” is not an abstract concept; it is the client’s communication of a vector containing the identified object IDs that the network uses to update its registry.

3. “input source” / “identify a category of the update message based on the input source” (claim 14)

Defendant	Plaintiff
identify a category of the update message based on the information provider or dynamic content provider but not on the category/topic of the message content	plain meaning

This limitation should be construed to require publisher-based categorization: the gateway identifies a category of the update message based on the input source (the information provider and/or dynamic content provider), rather than on the category/topic of message content.

The specification provides explicit lexicography for “input source.” It defines the term as “the information provider or a dynamic content provider (generically referred to as an ‘input source’).” ’722 Patent at 3:15–16. It likewise references an “information provider 108/dynamic content provider 116 (generically referred to as an ‘input source 210’),” and the “input source 210, i.e., the information provider 108 and/or dynamic content provider 116” *Id.* at 11:22–24, 13:26–28.

From those passages, the specification teaches that the “input source” is the information provider and/or dynamic content provider, not a subject-matter category.

Consistent with that definition, the specification states that “the input source 210 sends ... an update message to the routing network ... in order to change a property of a live object at the client,” and that “the information provider 108 and/or the dynamic content provider 116 provides update messages to the routing network 110.” *Id.* at 11:38–40, 9:50–54. This usage confirms “input source” denotes the publisher/producer—setting up the claim requirement that category be identified “based on the input source.”

Claim 14 then requires the gateway “to identify a category of the update message based on the input source,” and to route by mapping categories to node types. The specification summarizes this operation: “When a gateway receives messages from input sources, the gateway identifies the categories of the messages and routes the messages to the nodes of the types to which the categories are mapped.” *Id.* at 3:43–50. Thus, this particular limitation requires the gateway to assign a category for an update message based on the input source, not what the message is about.

The specification identifies this “input source” categorization as one of several categorization techniques: “There are multiple ways to assign the messages into categories. One way is to assign all messages from a given input source ... into a certain category. Another way is to explicitly specify the category in the object ID for the message. Yet another way is to utilize a hashing function or lookup table to partition messages into categories based on object IDs or other values.” *Id.* at 18:52–58. This claim focuses on categorization “based on the input source,” as opposed to other approaches.

The prosecution history confirms this distinction. In distinguishing prior art, the applicants contrasted their claims—which identify a category “based on the input source”—with systems that

identify categories “based on the category/topic of the message content.” Ex. A-9 at 18–19. They repeatedly emphasized that the claims recite “different distinguishing features, i.e., an updated message is identified based on the input source,” whereas the prior art “merely teaches using a category/topic of message content.” Ex. A-10 at 23. Those statements confirm that content/topic-based categorization is outside the claimed scope.

Based on the specification’s definition of “input source,” the claim language, the teaching in the specification, and the prosecution history, the Court should adopt a construction that requires categorization of the update message based on the information provider, not the category/topic of the message content.

Dated: October 14, 2025

Respectfully submitted,

/s/ S. Wallace Dunwoody

Michael C. Wilson

Texas Bar No. 21704590

mwilson@munckwilson.com

S. Wallace Dunwoody

Texas Bar No. 24040838

wdunwoody@munckwilson.com

MUNCK WILSON MANDALA, LLP

2000 McKinney Avenue, Suite 1900

Dallas, Texas 75201

(972) 628-3600

(972) 628-3616 fax

David G. Henry

Texas Bar. No. 09479355

dhenry@munckwilson.com

MUNCK WILSON MANDALA, LLP

510 Austin Avenue, Suite 3100

Waco, Texas 76701

(254) 362-2300

(254) 362-2304 fax

Tri T. Truong

Texas Bar No. 24102969

ttruong@munckwilson.com

MUNCK WILSON MANDALA, LLP

807 Las Cimas Parkway, Suite 300

Austin, Texas 78756

737-201-1600

737-201-1601 fax

Attorneys for Southwest Airlines Co.

CERTIFICATE OF SERVICE

The undersigned hereby certifies that on October 14, 2025, the foregoing document has been served on all counsel of record in accordance with the Federal Rules of Civil Procedure.

/s/ S. Wallace Dunwoody

S. Wallace Dunwoody

EXHIBIT A

**IN THE UNITED STATES DISTRICT COURT
FOR THE WESTERN DISTRICT OF TEXAS
MIDLAND/ODESSA DIVISION**

**INTELLECTUAL VENTURES I LLC and §
INTELLECTUAL VENTURES II LLC, §**

Plaintiffs, §

v. §

SOUTHWEST AIRLINES CO., §

Defendant. §

Civil Action No. 7:24-cv-00277-ADA

JURY TRIAL DEMANDED

DECLARATION OF S. WALLACE DUNWOODY

I, S. Wallace Dunwoody, declare as follows:

1. I am an attorney licensed to practice law in the State of Texas and am admitted before this Court. I am an attorney with the law firm Munck Wilson Mandala LLP, and counsel for Defendant Southwest Airlines Co. ("Southwest"). I submit this Declaration in support of Southwest's Opening Claim Construction Brief for U.S. Patent Nos. 7,721,282, 8,332,844, 7,949,785, and 8,407,722.

2. I have personal knowledge of the facts herein. If called as a witness, I would testify to the following facts:

3. Attached as **Exhibit A-1** is a true and correct copy of U.S. Patent No. 7,721,282.
4. Attached as **Exhibit A-2** is a true and correct copy of U.S. Patent No. 8,332,844.
5. Attached as **Exhibit A-3** is a true and correct copy of U.S. Application No. 11/026,622.

6. Attached as **Exhibit A-4** is a true and correct copy of Applicant Arguments/Remarks Made in an Amendment to the application for U.S. Patent No. 7,721,282 on October 8, 2009.

7. Attached as **Exhibit A-5** is a true and correct copy of U.S. Patent No. 7,949,785.

8. Attached as **Exhibit A-6** is a true and correct copy of patentee's Remarks made in response to the May 22, 2024 Request for *Ex Parte* Reexamination of U.S. Patent No. 7,949,785.

9. Attached as **Exhibit A-7** is a true and correct copy of Applicant Arguments/Remarks Made in an Amendment to the application for U.S. Patent No. 7,949,785 on March 22, 2007.

10. Attached as **Exhibit A-8** is a true and correct copy of U.S. Patent No. 8,407,722.

11. Attached as **Exhibit A-9** is a true and correct copy of Applicant Arguments/Remarks Made in an Amendment to the application for U.S. Patent No. 8,407,722 on February 9, 2012.

12. Attached as **Exhibit A-10** is a true and correct copy of Applicant's Brief on Appeal Under 37 C.F.R. § 41.37 for the application for U.S. Patent No. 8,407,722 filed on May 31, 2012.

I declare under penalty of perjury under the laws of the United States of America that the foregoing is true and correct. Executed this 14th day of October 2025, at Dallas, Texas.

/s/ S. Wallace Dunwoody
S. Wallace Dunwoody

EXHIBIT A-1

US007721282B1

(12) **United States Patent**
Kulkarni et al.(10) **Patent No.:** **US 7,721,282 B1**
(45) **Date of Patent:** **May 18, 2010**(54) **BLOCK-LEVEL I/O SUBSYSTEM FOR
DISTRIBUTED APPLICATION
ENVIRONMENT MANAGEMENT**(75) Inventors: **Pradip Kulkarni**, Pune (IN); **Mukul Kumar**, Pune (IN); **Adhir Potdar**, Maharashtra (IN); **Richard Au**, Woodside, CA (US); **James M. Mott**, Austin, TX (US); **Tung M. Nguyen**, Cupertino, CA (US)(73) Assignee: **Panta Systems, Inc.**, Santa Clara, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 1084 days.

(21) Appl. No.: **11/395,816**(22) Filed: **Mar. 30, 2006****Related U.S. Application Data**

(63) Continuation-in-part of application No. 11/026,622, filed on Dec. 30, 2004.

(51) **Int. Cl.****G06F 9/445** (2006.01)**G06F 9/44** (2006.01)(52) **U.S. Cl.** **717/176; 717/175; 709/203**(58) **Field of Classification Search** **717/172–177, 717/149; 709/203–204**

See application file for complete search history.

(56) **References Cited****U.S. PATENT DOCUMENTS**

5,113,523 A * 5/1992 Colley et al. 712/12
 5,127,104 A * 6/1992 Dennis 712/201
 5,764,902 A * 6/1998 Rothrock 709/205
 5,999,734 A * 12/1999 Willis et al. 717/149
 6,018,747 A * 1/2000 Burns et al. 707/203
 6,195,680 B1 * 2/2001 Goldszmidt et al. 709/203

6,292,941 B1 * 9/2001 Jollands 717/176
 6,502,238 B1 * 12/2002 Pavan et al. 717/149
 6,606,744 B1 * 8/2003 Mikurak 717/174
 6,779,177 B1 * 8/2004 Bahrs et al. 717/173
 6,990,513 B2 * 1/2006 Belfiore et al. 709/203
 7,127,712 B1 * 10/2006 Noble et al. 717/173
 7,150,015 B2 * 12/2006 Pace et al. 717/176
 7,155,714 B2 * 12/2006 Makris et al. 717/176
 7,200,715 B2 * 4/2007 Kleiman et al. 711/114
 7,246,351 B2 * 7/2007 Bloch et al. 717/175
 7,263,551 B2 * 8/2007 Belfiore et al. 709/219
 7,331,047 B2 * 2/2008 Chu et al. 717/178
 7,454,462 B2 * 11/2008 Belfiore et al. 709/203
 7,467,293 B2 * 12/2008 Zhang et al. 713/1
 7,475,274 B2 * 1/2009 Davidson 714/4
 7,496,739 B1 * 2/2009 Raghavan et al. 713/1

OTHER PUBLICATIONS

Ohta et al., "Improving parallel write by node level request scheduling", IEEE, pp. 196-203, 2009.*

Li et al., "Design issue of a novel toolkit for parallel application performance monitoring and analysis in cluster and grid environments", IEEE ISPAN, pp. 1-6, 2005.*

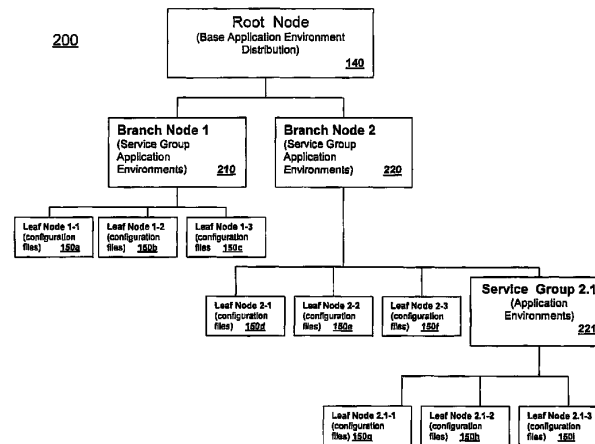
(Continued)

Primary Examiner—Anil Khatri

(57)

ABSTRACT

An embodiment of the present invention is directed to a system for distributing an application environment to a compute node. The system includes a first storage unit for storing blocks of a root image of the compute node and a second storage unit for storing a leaf image comprising new data blocks and changes to the blocks of the root image. The system further includes a union block device for interfacing between the compute node and the first and second storage units to distribute the application environment to the compute node. The union block device creates the application environment by merging the blocks of the root image stored on the first storage unit with the leaf image stored on the second storage unit.

30 Claims, 3 Drawing Sheets

US 7,721,282 B1

Page 2

OTHER PUBLICATIONS

Zhu et al, "Scheduling divisible loads in the dynamic heterogenous grid environment", ACM Infoscale, pp. 1-10, 2006.*

Chen et al, "Exploiting high level coherence information to optimize distributed shared state", ACM PPpPP, pp. 131-142, 2003.*

* cited by examiner

U.S. Patent

May 18, 2010

Sheet 1 of 3

US 7,721,282 B1

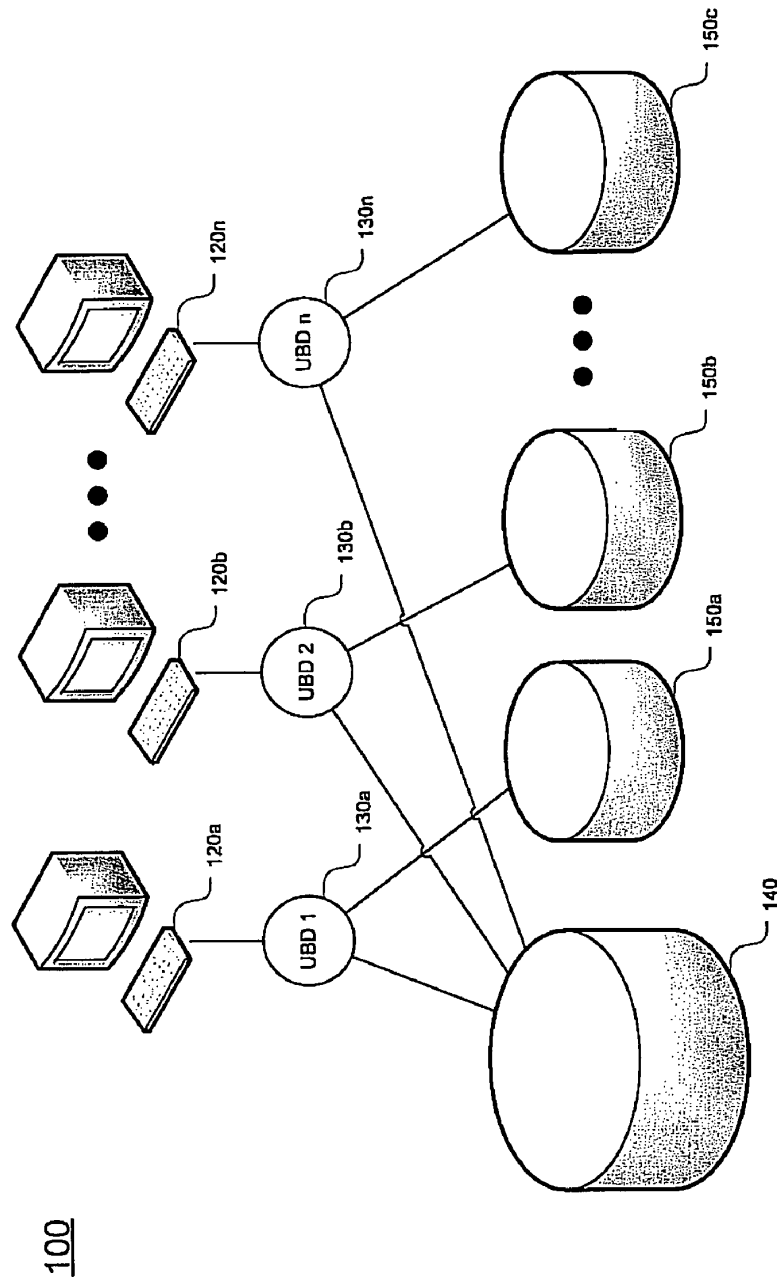


Figure 1

U.S. Patent

May 18, 2010

Sheet 2 of 3

US 7,721,282 B1

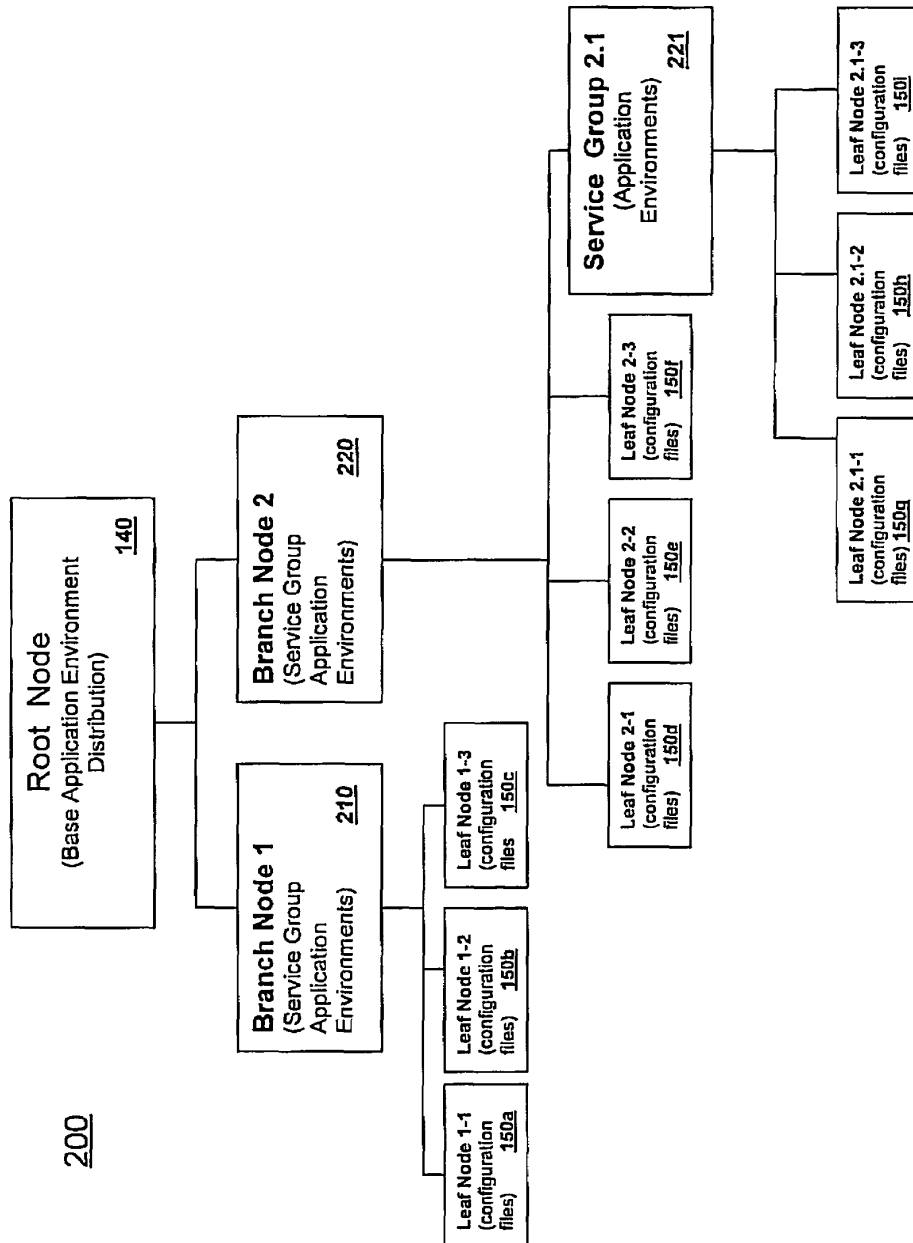


Figure 2

U.S. Patent

May 18, 2010

Sheet 3 of 3

US 7,721,282 B1

300

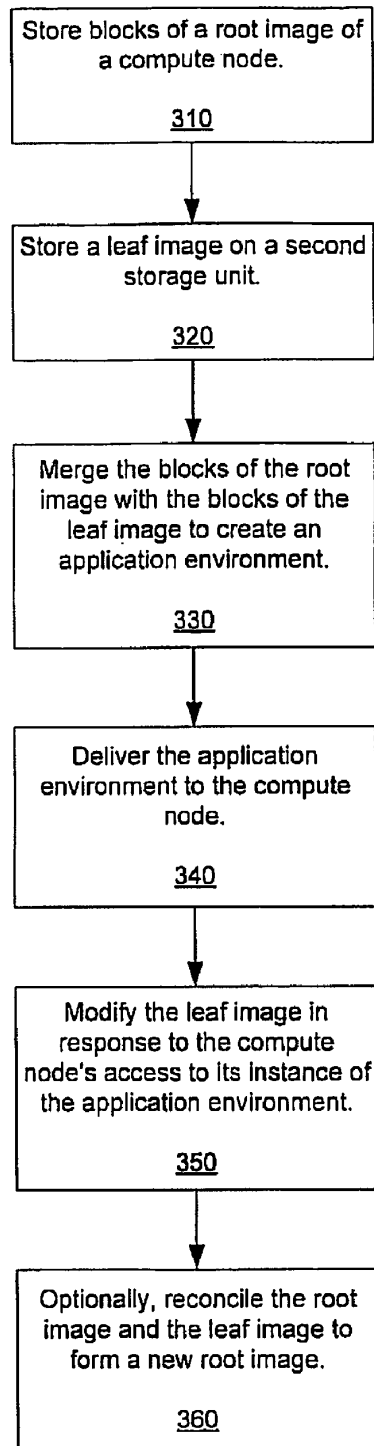


Figure 3

US 7,721,282 B1

1

BLOCK-LEVEL I/O SUBSYSTEM FOR DISTRIBUTED APPLICATION ENVIRONMENT MANAGEMENT

CLAIM OF PRIORITY UNDER 35 U.S.C. §120

The present Application for Patent is a Continuation-in-Part of patent application Ser. No. 11/026,622 entitled "BRANCING STORE FILE SYSTEM" filed Dec. 30, 2004, pending, and assigned to the assignee hereof and hereby expressly incorporated by reference herein.

BACKGROUND

1. Field of the Invention

Embodiments of the present invention relate to the field of networked computing. Specifically, embodiments of the present invention relate to distributed application environment deployment in a multi-computer system.

2. Background

Over the years, as the internet has expanded and computers have multiplied, the need for clustered computing such as High Performance Computing (HPC) has increased. Clustered computing involves multiple compute nodes, usually a server grid, that work together to achieve a common task. For example, several (typically hundreds of) compute nodes may be clustered together to share the load of serving a high-traffic website. In large-scale systems such as this, a trend in software deployment is to centralize data management on a globally accessible file system with stateless computing nodes. A common example of this is Operating System (OS) software image management, where the compute nodes are activated with the distributed application environment by either diskless booting protocols or remote software installation to local storage. Under this architecture, a boot image is required for each compute node in the cluster. The boot image necessarily contains the kernel; it may additionally contain the application software that is intended to be run on the compute node.

The primary concern in clustered computing is low cluster bring-up time. The software that provides the boot images for the cluster typically stores a master boot image. It may then either pre-create clones of this master image for each such server, or it may create them "on the fly."

Creating a boot image on the fly involves copying the entire contents of the master image, which are typically in the range of 5-15 GB. Even with a significant amount of bandwidth by today's standards, this method will result in a large bring-up time.

Pre-creating a boot image for each server is advantageous from the point of view of cluster bring-up time. However, since one often does not know in advance how many servers will ever be booted, this scheme may result in wasted disk space.

Regardless of which of the preceding methods is used, both suffer from the same major problem—updating the boot image(s) for the cluster is cumbersome, as it means updating a number of copies of the boot image.

A Branching Store File System, as described in patent application Ser. No. 11/026,622 entitled "BRANCING STORE FILE SYSTEM" filed Dec. 30, 2004, pending, and assigned to the assignee hereof, was developed as a solution to the above problem. In a branching store file system, a read-only base image (or "root" image) of the application environment is created. The root image is accessible by all compute nodes in the cluster. Changes made by a compute node to the root image are stored in a "leaf" image unique to that compute node. A filter operates between the compute

2

nodes and the file system(s), which merges the changes recorded on the leaf images with the root image and delivers the result to the appropriate compute node. From the point of view of the compute node, it is running its own unique and cohesive instance of the application environment. While this system allows for creation of boot images on the fly without severely diminishing bring-up time, a separate version of the system must be created for each unique operating system. Thus, migrating a computing cluster from one operating system to another is much more complicated than simply installing a new root image containing the new OS.

SUMMARY

An embodiment of the present invention is directed to a system for distributing an application environment to a compute node (e.g., a server or a thin-client workstation). The system includes a first storage unit for storing blocks of a root image (e.g., an operating system) of the compute node and a second storage unit for storing a leaf image comprising new data blocks and changes to the blocks of the root image. The system further includes a union block device for interfacing between the compute node and the first and second storage units to distribute the application environment to the compute node. The union block device creates the application environment by merging the blocks of the root image stored on the first storage unit with the leaf image stored on the second storage unit.

Another embodiment of the present invention is directed to a method for distributing an application environment to a compute node. The method includes storing blocks of a root image of the compute node on a first storage unit and storing a leaf image comprising new data blocks and changes to the blocks of the root image on a second storage unit. The method further includes merging the blocks of the root image stored on the first storage unit with the blocks of the leaf image stored on the second storage unit to create the application environment. The application environment is created by merging the blocks of the root image stored on the first storage unit with the blocks of the leaf image stored on the second storage unit. Lastly, the application environment is delivered to the computer node.

Thus, embodiments of the present invention provide an operating system-independent system and method for distributing an application environment to a compute node. By utilizing a root-leaf system of application environment storage, embodiments of the present invention allow creation of boot images on the fly without significantly diminishing bring-up time. This is due to the fact that creating a new boot image does not require copying the contents of the root image. Rather it involves registering a new UBD with the system, which occurs very quickly. Moreover, updating the boot image for the entire cluster simply involves updating the root image.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a diagram of a system for distributing an application environment, in accordance with an embodiment of the present invention.

FIG. 2 is a tree structure that illustrates a branching system, in accordance with an embodiment of the present invention.

US 7,721,282 B1

3

FIG. 3 is a flowchart illustrating a process 300 for distributing an application environment to a compute node, in accordance with an embodiment of the present invention.

DETAILED DESCRIPTION

Reference will now be made in detail to the preferred embodiments of the invention, examples of which are illustrated in the accompanying drawings. While the invention will be described in conjunction with the preferred embodiments, it will be understood that they are not intended to limit the invention to these embodiments. On the contrary, the invention is intended to cover alternatives, modifications and equivalents, which may be included within the spirit and scope of the invention as defined by the claims. Furthermore, in the detailed description of the present invention, numerous specific details are set forth in order to provide a thorough understanding of the present invention. However, it will be obvious to one of ordinary skill in the art that the present invention may be practiced without these specific details. In other instances, well known methods, procedures, components, and circuits have not been described in detail as not to unnecessarily obscure aspects of the present invention.

Some portions of the detailed descriptions that follow are presented in terms of procedures, logic blocks, processing, and other symbolic representations of operations on data bits within a computer or digital system memory. These descriptions and representations are the means used by those skilled in the data processing arts to most effectively convey the substance of their work to others skilled in the art. A procedure, logic block, process, etc., is herein, and generally, conceived to be a self-consistent sequence of steps or instructions leading to a desired result. The steps are those requiring physical manipulations of physical quantities. Usually, though not necessarily, these physical manipulations take the form of electrical or magnetic signals capable of being stored, transferred, combined, compared, and otherwise manipulated in a computer system or similar electronic computing device. For reasons of convenience, and with reference to common usage, these signals are referred to as bits, values, elements, symbols, characters, terms, numbers, or the like with reference to the present invention.

It should be borne in mind, however, that all of these terms are to be interpreted as referencing physical manipulations and quantities and are merely convenient labels and are to be interpreted further in view of terms commonly used in the art. Unless specifically stated otherwise as apparent from the discussion herein, it is understood that throughout discussions of the present embodiment, discussions utilizing terms such as “determining” or “outputting” or “transmitting” or “recording” or “locating” or “storing” or “displaying” or “receiving” or “recognizing” or “utilizing” or “generating” or “providing” or “accessing” or “checking” or “notifying” or “delivering” or the like, refer to the action and processes of a computer system, or similar electronic computing device, that manipulates and transforms data. The data is represented as physical (electronic) quantities within the computer system’s registers and memories and is transformed into other data similarly represented as physical quantities within the computer system memories or registers or other such information storage, transmission, or display devices.

Embodiments of the present invention provide methods and systems for distributing an application environment to one or more compute nodes. FIG. 1 is a diagram of a system 100 for distributing an application environment. In a preferred embodiment, system 100 is implemented in a multi-computer system, such as an HPC cluster. In one embodi-

4

ment, the application environment includes an operating system. In other embodiments, the application environment may contain other applications. System 100 has a number of compute nodes 120a-n coupled to first storage unit and a corresponding second storage unit 150a-n through a corresponding union block device (UBD) 130a-n. To compute nodes 120a-n, it appears that they have access to their own version of a distributed application. However, a separate and complete boot image is not created and stored for each compute node 120a-n.

System 100 has a first storage unit 140 for storing blocks of a root image of an application environment. The root image contains data initially common to the compute nodes 120a-n. The root image is not changed by compute nodes 120a-n. For example, in one embodiment, compute nodes 120a-n have read-only access to the first storage unit.

Moreover, each compute node 120a-n has a corresponding second storage unit 150a-n for storing a leaf image. The first storage unit 140 and second storage units 150a-n may each be contained on separate physical storage devices, on separate logical spaces on the same storage device, or any combination thereof. The leaf image may contain blocks of new data, blocks of changed data, or other blocks of data unique to the individual compute node. The leaf image may also contain a block modification log. In other words, a leaf image will describe the changes made by the respective compute node 120a-n to its instance of the application environment. Thus, when a compute node (e.g., node 120a) makes changes involving the root image, modifications are made to that compute node’s leaf image (e.g., leaf image stored on second storage device 150a). With respect to changes to the root image, only the specific blocks that are changed are stored in the leaf image. For example, a particular file on the root image may comprise twenty blocks of data (e.g., blocks 1-20). One compute node (e.g., compute node 120a) desires to make a change to this file which involves a modification of only a few specific blocks of the file (e.g., blocks 4-9). In this example, only the modified blocks (e.g., blocks 4-9) will be stored in the compute node’s leaf image (e.g., leaf image stored on second storage device 150a) plus some small overhead.

A compute node 120a-n mounts its instantiation of the application environment via its respective UBD 130a-n. The UBDs 130a-n are effectively low-level drivers that operate as an interface between the first and second storage devices and the file system of each compute node 120a-n. The file system may reside on the server side of the system 100. The file system may also reside on each of the compute nodes 120a-n. Because UBDs 130a-n operate below the file system, they are concerned merely with the blocks of data themselves, rather than files they form. As a result, system 100 is completely file system, and thus operating system, independent.

UBDs 130a-n determine what leaf image (from the appropriate second storage unit 150) is needed for portions of the application environment that their respective compute nodes 120a-n have changed. UBDs 130a-n also locate the portions of the application environment that are not changed by their respective their respective compute nodes 120a-n. These portions may reside in the root image. There may also be intermediate images (not depicted in FIG. 1) comprising versions of the root image from which a compute node’s instance of the application environment is derived. Further, UBDs 130a-n create a new leaf image on a respective second storage unit 150a-n when their respective compute nodes 120a-n make changes to their instantiations of the application environment. In one embodiment, the UBDs 130a-n obtain the information necessary to create the application from a block modification log located in its respective leaf image.

US 7,721,282 B1

5

UBDs **130a-n** may also modify the leaf image in response to their respective compute node's access to its instance of the application environment. For example, upon receiving a write request from their respective compute nodes for a sector **X**, the UBDs **130a-n** will create an appropriate persistent mapping for sector **X** and then write sector **X** onto their respective second storage units **150a-n**, where sector **X** can then be modified. It will be appreciated that the data block being modified may already exist in the leaf image, in which case it does not need to be mapped and copied from the root image before modification.

By providing an application environment to multiple compute nodes via a branching store system, embodiments allow for a more streamlined update/patch procedure. In one embodiment, first storage device **140** may be refreshed with a new operating system patch, thereby automatically refreshing the image for all dependent compute nodes.

Application Groups

An embodiment of the present invention provides for application groups (AG). An AG is a way to provide a group of compute nodes their own instantiation of an application environment, while allowing versions of the application environment to be frozen from time to time. Thus, an archive of the application environment is created.

At creation time, an AG comprises the original root image. A number of leaf images are created to match the desired capacity demand of the AG. The leaf images can be created very quickly and require very little storage because the leaf images reflect only the changes to the root instance.

FIG. 2 is a tree structure **200** that illustrates a branching system, in accordance with an embodiment of the present invention. Some nodes are read-only, others are read-write-modify. A node may change from read-write-modify to read-only. For example, creating a new edition of the AG may be accomplished by freezing a current leaf image structure and creating a new set of leaf images therefrom. Referring now to FIG. 2, the root node **140** (also referred to herein as the root image) is the base application environment. As previously discussed, the root node **140** is not modified by compute nodes for which the application environment is distributed.

The first level of the exemplary tree structure **200** comprises first branch node **210** and second branch node **220**. Branch node **210** has three leaf nodes **150a-c**. The leaf nodes **150** may be for blocks from configuration files, for example. The compute nodes **150** can boot from the configuration files. The leaf nodes are also referred to herein as leaf images. In another embodiment, the leaf nodes **150** comprise block modification logs.

The branch node **220** also has three leaf nodes **150d-f**, which are analogous to the leaf nodes **150a-c**. However, branch node **220** also has a service group **221**, which reflects a change to the application environment. The service group **221** has leaf nodes **150g-i**.

Embodiments provide for additional advanced features. In one embodiment for example, a particular branch, rather than the entire tree structure back to the root, is refreshed with an update to the application environment contained thereon, thereby automatically refreshing its dependent compute nodes. In another embodiment, a leaf node or a branch node may be completely detached from the original structure to make a new and completely independent branch.

FIG. 3 is a flowchart illustrating a process **300** for distributing an application environment to a compute node, in accordance with an embodiment of the present invention. Steps of process **300** may be stored as instructions on a computer readable medium and executed on a computer processor. Step **310** is storing blocks of a root image of the compute node on a first storage unit. By storing data at the block level, embodi-

6

ments are able to operate beneath the file system and thus are designed to be file system and operating system independent.

Step **320** is storing a leaf image on a second storage unit. The leaf image includes, but is not limited to, new data blocks for the compute node and blocks of the root image that the compute node has changed. The leaf image includes a block modification log in one embodiment.

Step **330** is merging the blocks of the root image with the blocks of the leaf image to create the application environment. Once the application environment has been created, it will appear to the compute node as one cohesive image rather than a base image plus its additions, deletions, and modifications. To the compute node, it appears that it has access to its own unique version of an application. However, a separate and complete boot image is not actually stored for the compute node.

Step **340** is delivering the application environment to the compute node. Step **340** may comprise a low-level driver determining which data blocks are needed for the compute node's instance of the application environment and delivering the application environment to the compute via the compute node's file system.

Step **350** is modifying the leaf image in response to the compute node's access to its instance of the application environment. The modifying of the leaf image may include copying one or more data blocks from the root image to the leaf image, and modifying the data block in the leaf image. It will be appreciated that the data block being modified may already exist in the leaf image, in which case it does not need to be copied from the root image before modification.

Step **360** is reconciling the root image and the leaf image to form a new root image. This may be desirable if, for example, the leaf image has grown to exceed a particular disk quota. Furthermore, if there are multiple compute nodes that access the root image, each having their own respective leaf image, and there is substantial commonality between the leaf images, it may also be beneficial to reconcile the leaf images with the root image.

Thus, by operating at the block level, embodiments of the present invention provide a file system and operating system independent system and method for distributing an application environment to a compute node. By utilizing a branching store system of application environment distribution, embodiments of the present invention allow creation of boot images on the fly without significantly diminishing bring-up time. This is due to the fact that creating a new boot image does not require copying the contents of the root image, but rather it involves registering a new UBD with the system, which occurs very quickly. Moreover, updating the boot image for the entire cluster simply involves updating the root image.

Embodiments of the present invention also allow for scalability and redundancy. For example, a server containing the root image may only have the resources to supply the root image to 100 compute nodes. Thus, in order to implement a 200-compute node system, two servers, each containing a copy of the root image, are used. This scalability provided by embodiments also lends itself to dynamic scaling. In other words, the number of root images required for a specific configuration can change based on the I/O workload, and new root images can be created on the fly accordingly. Moreover, additional servers containing the root image may be added into the system to provide redundancy and increase reliability. As an added security measure, some of the root nodes may be located remotely from each other (e.g., half located in a main office and half in a satellite office). In addition, it is possible to copy the contents of one or more leaf nodes into a non-UBD (regular block device) copy (e.g., for use as an isolated backup).

US 7,721,282 B1

7

The previous description of the disclosed embodiments is provided to enable any person skilled in the art to make or use the present invention. Various modifications to these embodiments will be readily apparent to those skilled in the art, and the generic principles defined herein may be applied to other embodiments without departing from the spirit or scope of the invention. Thus, the present invention is not intended to be limited to the embodiments shown herein but is to be accorded the widest scope consistent with the principles and novel features disclosed herein.

What is claimed is:

1. A system for distributing an application environment comprising:

a compute node comprising a computer system;

a first storage unit for storing blocks of a root image of the compute node, wherein the first storage unit comprises a first non-volatile memory, wherein the root image comprises a computer program, wherein the blocks comprise sections of data, and wherein a file of the root image comprises at least one block;

a second storage unit for storing a leaf image, the leaf image comprising new data blocks and changes to the blocks of the root image, wherein the second storage unit comprises a second non-volatile memory; and

a union block device for interfacing between the compute node and the first and second storage units to distribute the application environment to the compute node, wherein the union block device comprises a driver, wherein the union block device creates the application environment by merging the blocks of the root image stored on the first storage unit with the blocks of the leaf image stored on the second storage unit; the union block device comprises a low-level driver for interfacing between the first and second storage units and the file system of the compute node; and the union block device, upon receiving a write request from the compute node for a sector X, creates an appropriate persistent mapping for sector X.

2. The system as recited in claim 1 wherein the compute node comprises a server.

3. The system as recited in claim 1 wherein the compute node comprises a thin-client workstation.

4. The system as recited in claim 1 wherein the root image comprises an operating system.

5. The system as recited in claim 1 wherein the root image is concurrently accessible to a plurality of compute nodes.

6. The system as recited in claim 1 wherein the first storage unit is remotely located from the compute node.

7. The system as recited in claim 1 wherein the second storage unit is remotely located from the compute node.

8. The system as recited in claim 1 wherein the second storage unit contains a block modification log for the compute node.

9. The system as recited in claim 1 wherein the first storage unit is contained within a first partition on a hard disk and the second storage unit is contained within a second partition on the hard disk.

10. The system as recited in claim 1 wherein the union block device writes sector X on the second storage unit.

11. The system as recited in claim 1 wherein the system operates in a high performance computing cluster.

12. The system as recited in claim 1 wherein the system operates in a grid computing cluster.

13. The system as recited in claim 1 wherein the first storage unit is read only.

14. The system as recited in claim 1 further comprising an intermediate image between the root image and the leaf

8

image, wherein the intermediate image comprises an application environment service group.

15. A method for distributing an application environment comprising:

storing blocks of a root image of a compute node on a first storage unit, wherein the compute node comprises a computer system, and wherein the first storage unit comprises a first non-volatile memory, wherein the root image comprises a computer program, wherein the blocks comprise sections of data, and wherein a file of the root image comprises at least one block;

storing a leaf image comprising new data blocks and changes to the blocks of the root image on a second storage unit, wherein the second storage unit comprises a second non-volatile memory;

merging the blocks of the root image stored on the first storage unit with the blocks of the leaf image stored on the second storage unit to create the application environment; and

delivering the application environment to the compute node; wherein the modifying comprises: upon receiving a write request from the compute node for a sector X, creating an appropriate persistent mapping for sector X; and writing sector X on the second storage unit.

16. The method as recited in claim 15 further comprising: modifying the leaf image in response to the compute node's access to the application environment.

17. The method as recited in claim 15 further comprising: reconciling the root image and the leaf image to form a new root image.

18. The method as recited in claim 15 wherein the compute node comprises a server.

19. The method as recited in claim 15 wherein the compute node comprises a thin-client workstation.

20. The method as recited in claim 15 wherein the root image comprises an operating system.

21. The method as recited in claim 15 wherein the root image is concurrently accessible to a plurality of compute nodes.

22. The method as recited in claim 15 wherein the first storage unit is remotely located from the compute node.

23. The method as recited in claim 15 wherein the second storage unit is remotely located from the compute node.

24. The method as recited in claim 15 wherein the second storage unit contains a block modification log for the compute node.

25. The method as recited in claim 15 wherein the first storage unit is contained within a first partition on a hard disk and the second storage unit is contained within a second partition on the hard disk.

26. The method as recited in claim 15 wherein merging occurs at an operational level between the first and second storage units and file system of the compute node.

27. The method as recited in claim 15 wherein the method operates in a high performance computing cluster.

28. The method as recited in claim 15 wherein the system operates in a grid computing cluster.

29. The method as recited in claim 15 wherein the first storage unit is read only.

30. The method as recited in claim 15 further comprising: creating an intermediate image on a third storage unit between the root image and the leaf image, wherein the intermediate image comprises an application environment service group.

* * * * *

EXHIBIT A-2

(12) **United States Patent**
Kulkarni et al.

(10) **Patent No.:** **US 8,332,844 B1**
(45) **Date of Patent:** **Dec. 11, 2012**

(54) **ROOT IMAGE CACHING AND INDEXING
FOR BLOCK-LEVEL DISTRIBUTED
APPLICATION MANAGEMENT**

(75) Inventors: **Pradip Kulkarni**, Pune (IN); **Mukul
Kumar**, Pune (IN); **Adhir Potdar**, Pune
(IN); **Richard Au**, Woodside, CA (US);
Tung Nguyen, Cupertino, CA (US)

(73) Assignee: **Emendable Assets Limited Liability
Company**, Wilmington, DE (US)

(*) Notice: Subject to any disclaimer, the term of this
patent is extended or adjusted under 35
U.S.C. 154(b) by 1193 days.

6,101,576 A * 8/2000 Kobayashi et al. 711/122
6,195,680 B1 2/2001 Goldszmidt
6,292,941 B1 9/2001 Jollands
6,421,777 B1 * 7/2002 Pierre-Louis et al. 713/2
6,442,605 B1 8/2002 Rodriguez
6,502,238 B1 12/2002 Pavan et al.
6,597,956 B1 7/2003 Aziz
6,606,744 B1 8/2003 Mikurak
6,745,192 B1 6/2004 Libenzi
6,751,658 B1 * 6/2004 Haun et al. 709/222
6,779,177 B1 8/2004 Bahrs
6,871,219 B2 3/2005 Noordergraaf
6,938,057 B2 8/2005 Gusler
6,986,005 B2 1/2006 Vo
6,990,513 B2 1/2006 Belfiore

(Continued)

OTHER PUBLICATIONS

(21) Appl. No.: **11/709,477**

(22) Filed: **Feb. 21, 2007**

Related U.S. Application Data

(63) Continuation-in-part of application No. 11/395,816,
filed on Mar. 30, 2006, now Pat. No. 7,721,282, which
is a continuation-in-part of application No.
11/026,622, filed on Dec. 30, 2004, now abandoned.

Calzarossa, M., and M. Kubát, "Processor Allocation in Parallel
Systems," Proceedings of the IEEE International Conference on
Computer Systems and Software Engineering (COMPEURO '91),
Bologna, Italy, May 13-16, 1991, pp. 133-137.

(Continued)

Primary Examiner — Edward Martello

(74) Attorney, Agent, or Firm — Christensen O'Connor
Johnson Kindness PLLC

(51) **Int. Cl.**

G06F 9/445 (2006.01)

(52) **U.S. Cl.** **717/176**

(58) **Field of Classification Search** **345/557;**
717/176

See application file for complete search history.

(57)

ABSTRACT

Described herein is technology for, among other things root
image caching and indexing for block-level distributed appli-
cation management. The technology involves storing blocks
of a root image on a first storage unit and storing blocks of leaf
images on respective second storage units. The leaf images
include additional data blocks not previously contained in the
root image and changes made by respective compute nodes to
the blocks of the root image. The technology includes caching
blocks of the root image that have been accessed by at least
one compute node. The technology also includes receiving
indexing results pertaining to the root image from one com-
pute node and providing the results for other compute nodes.

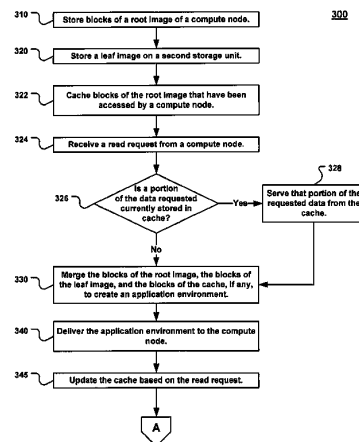
(56)

References Cited

U.S. PATENT DOCUMENTS

5,113,523 A 5/1992 Colley
5,127,104 A 6/1992 Dennis
5,175,852 A 12/1992 Johnson et al.
5,764,902 A 6/1998 Rothrock
5,974,547 A 10/1999 Klimenko
5,999,734 A 12/1999 Willis
6,018,747 A 1/2000 Burns

27 Claims, 6 Drawing Sheets



US 8,332,844 B1

Page 2

U.S. PATENT DOCUMENTS

6,990,573 B2 1/2006 Cherian
 7,127,712 B1 10/2006 Noble
 7,150,015 B2 12/2006 Pace
 7,155,714 B2 12/2006 Makris
 7,200,715 B2 4/2007 Kleiman
 7,246,221 B1 7/2007 Soltis
 7,246,351 B2 7/2007 Bloch
 7,263,551 B2 8/2007 Belfiore
 7,269,664 B2 9/2007 Hutsch et al.
 7,290,258 B2 10/2007 Steeb
 7,331,047 B2 2/2008 Chu
 7,334,157 B1 2/2008 Graf
 7,430,610 B2 9/2008 Pace
 7,454,462 B2 11/2008 Belfiore
 7,467,293 B2 12/2008 Zhang
 7,475,274 B2 1/2009 Davidson
 7,496,739 B1 2/2009 Raghavan
 7,499,988 B2 3/2009 Keohane
 7,536,686 B2 5/2009 Tan
 7,549,055 B2 6/2009 Zimmer
 7,590,653 B2 9/2009 Sparks
 7,668,938 B1 2/2010 Phillips
 7,685,148 B2 3/2010 Engquist
 7,870,106 B1 * 1/2011 Nguyen et al. 707/694
 2002/0016891 A1 2/2002 Noel
 2002/0083120 A1 6/2002 Soltis
 2002/0087813 A1 7/2002 Harris
 2002/0087846 A1 7/2002 Nickolls
 2003/0046511 A1 3/2003 Buch
 2003/0126242 A1 * 7/2003 Chang 709/222
 2003/0130832 A1 7/2003 Schalter
 2003/0233648 A1 12/2003 Earl
 2004/0015899 A1 1/2004 May
 2004/0078779 A1 4/2004 Dutt
 2004/0088473 A1 5/2004 Ogle
 2004/0199919 A1 10/2004 Tovinkere
 2005/0128319 A1 6/2005 Morino
 2005/0131962 A1 6/2005 Deshpande
 2006/0173993 A1 8/2006 Henseler
 2008/0229040 A1 9/2008 Honma

OTHER PUBLICATIONS

Chen, D., et al., "Exploiting High-Level Coherence Information to Optimize Distributed Shared State," Proceedings of the 9th ACM Symposium on Principles and Practice of Parallel Programming, San Diego, Jun. 11-13, 2003, pp. 131-142.

Cohen, A., and M. Woodring, "Win32 Multithreaded Programming," O'Reilly Online Catalog, Chapter 1, <<http://www.oreilly.com/catalog/multithread/excerpt/ch01.html>> [retrieved Nov. 21, 2006], pp. 1-8.

Couturier, R., et al., "A Compiler for Parallel Unity Programs Using OpenMp," Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, Las Vegas, Nevada, Jun. 28-Jul. 1, 1999, pp. 1992-1998.

Ismail, I.M., and J.A. Davis, "Program-Based Static Allocation Policies for Highly Parallel Computers," Proceedings of the 1995 IEEE Fourteenth Annual International Phoenix Conference on Computers and Communications, Scottsdale, Arizona, Mar. 28-31, 1995, pp. 61-68.

Kandemir, M., et al., "Minimizing Data and Synchronization Costs in One-Way Communication," IEEE Transactions on Parallel and Distributed Systems 11(12):1232-1251, Dec. 2000.

Li, K.-C., et al., "Design Issue of a Novel Toolkit for Parallel Application Performance Monitoring and Analysis in Cluster and Grid Environments," Proceedings of the 8th International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN'05), Las Vegas, Nev., Dec. 7-9, 2005, 6 pages.

Nguyen, T.M., et al., "Branching Store File System," U.S. Appl. No. 11/026,622, filed Dec. 30, 2004.

Ohta, K., et al., "Improving Parallel Write by Node Level Request Scheduling," Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, Shanghai, May 18-21, 2009, pp. 196-203.

OpenMP Fortran Application Program Interface (Version 2.0), Nov. 2000, <<http://www.openmp.org/drupal/mp-documents/fspec20.pdf>>, pp. 1-3 and 34-36.

OpenMP Fortran Application Program Interface (Version 2.0), Nov. 2000, <<http://www.openmp.org/drupal/mp-documents/fspec20.pdf>>, pp. 1-3, 18-19, 25-27, 34-37, and 67.

"Sequent's NUMA-Q SMP™ Architecture," Sequent Computer Systems, Inc., 1997, <http://parallel.ru/ftp/computers/sequent/NUMA_SMP_REV.pdf>, pp. 1-18.

Sivathanu, G., et al., "Ensuring Data Integrity in Storage: Techniques and Applications," Proceedings of the 2005 ACM Workshop on Storage Security and Survivability (StorageSS'05), Alexandria, Va., Nov. 7-10, 2005, pp. 26-36.

Zhu, T., et al., "Scheduling Divisible Loads in the Dynamic Heterogeneous Grid Environment," Proceedings of the First International Conference on Scalable Information Systems, Hong Kong, May 29-Jun. 1, 2006, 10 pages.

* cited by examiner

U.S. Patent

Dec. 11, 2012

Sheet 1 of 6

US 8,332,844 B1

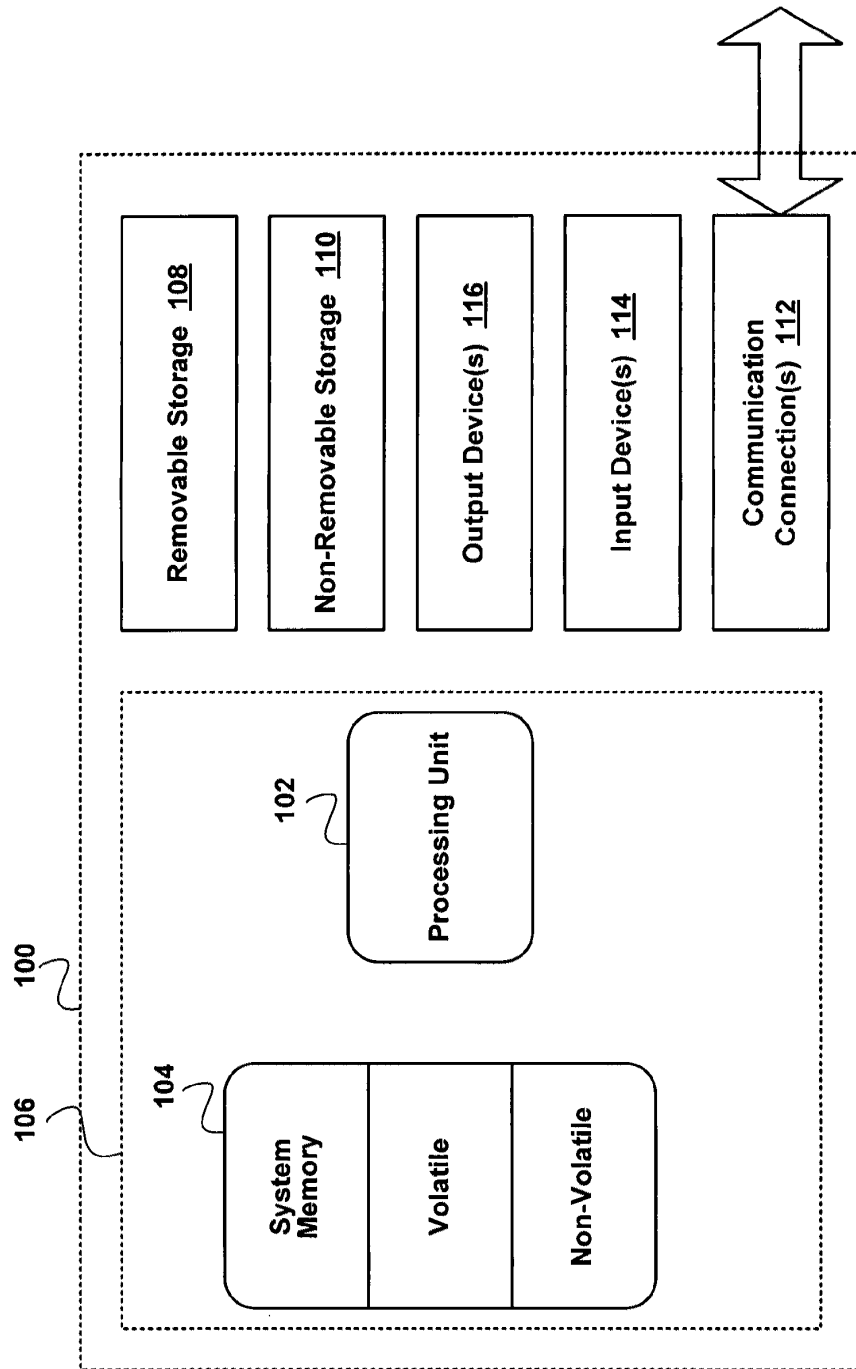


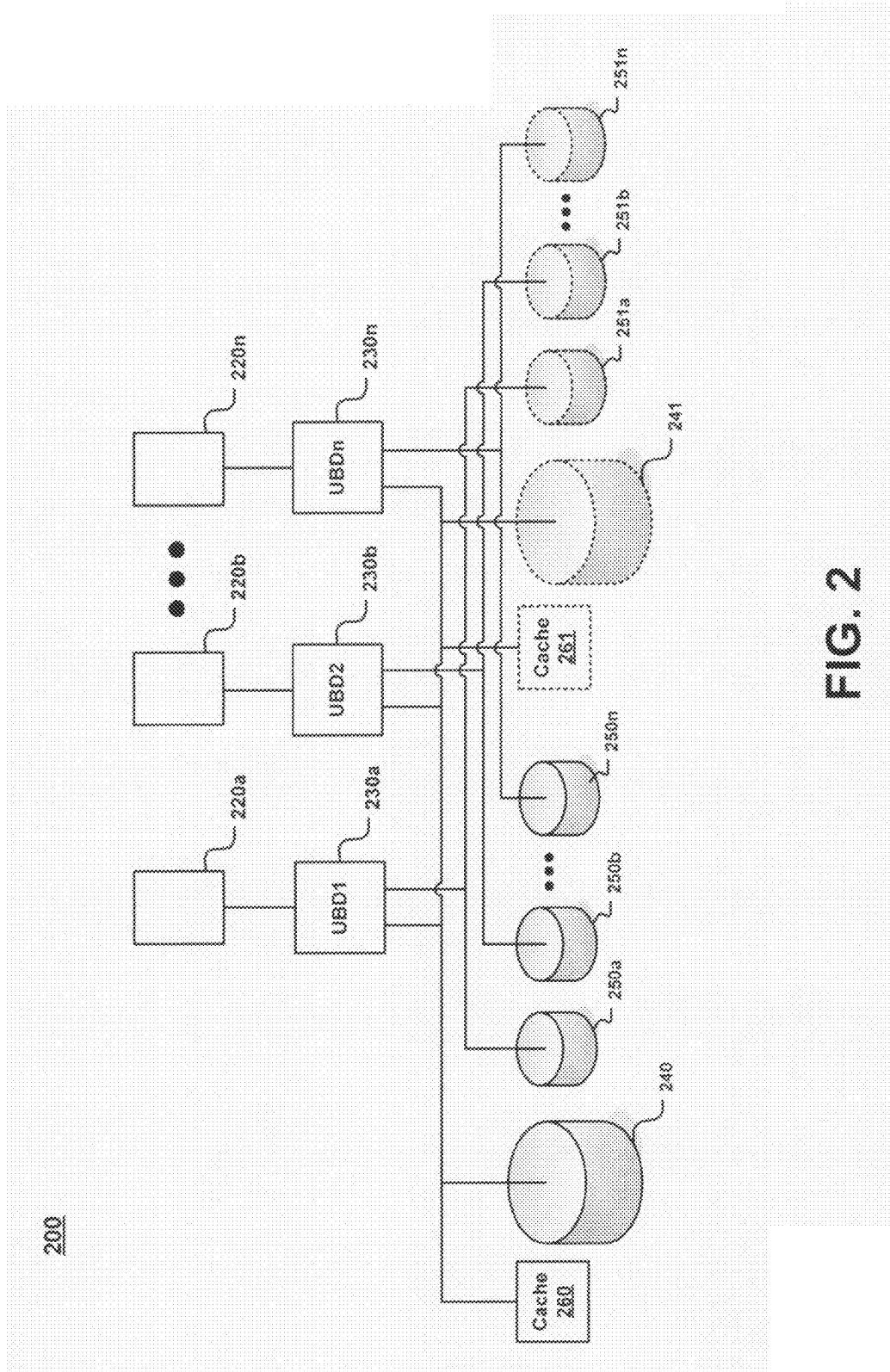
FIG. 1

U.S. Patent

Dec. 11, 2012

Sheet 2 of 6

US 8,332,844 B1



U.S. Patent

Dec. 11, 2012

Sheet 3 of 6

US 8,332,844 B1

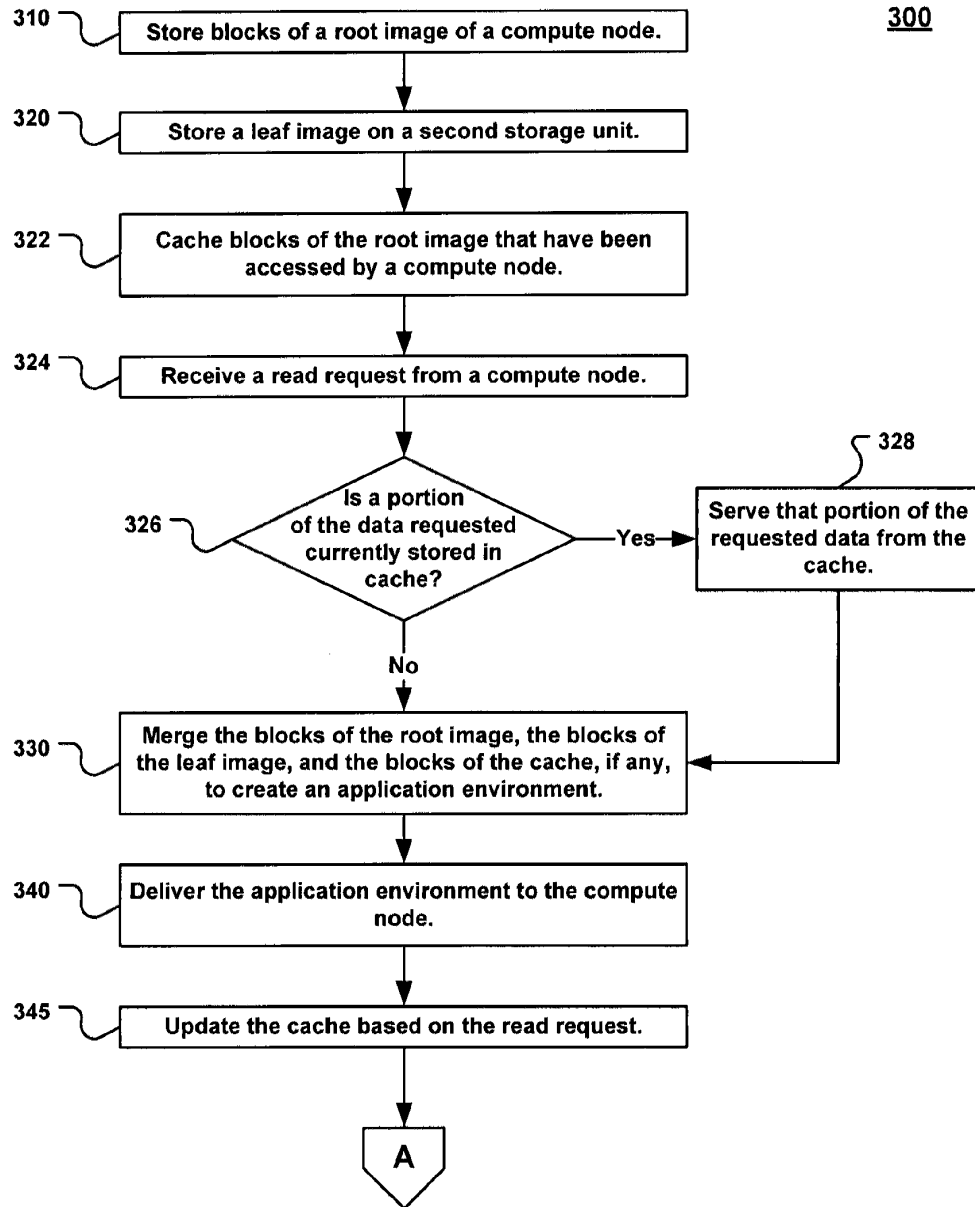


FIG. 3A

U.S. Patent

Dec. 11, 2012

Sheet 4 of 6

US 8,332,844 B1

300

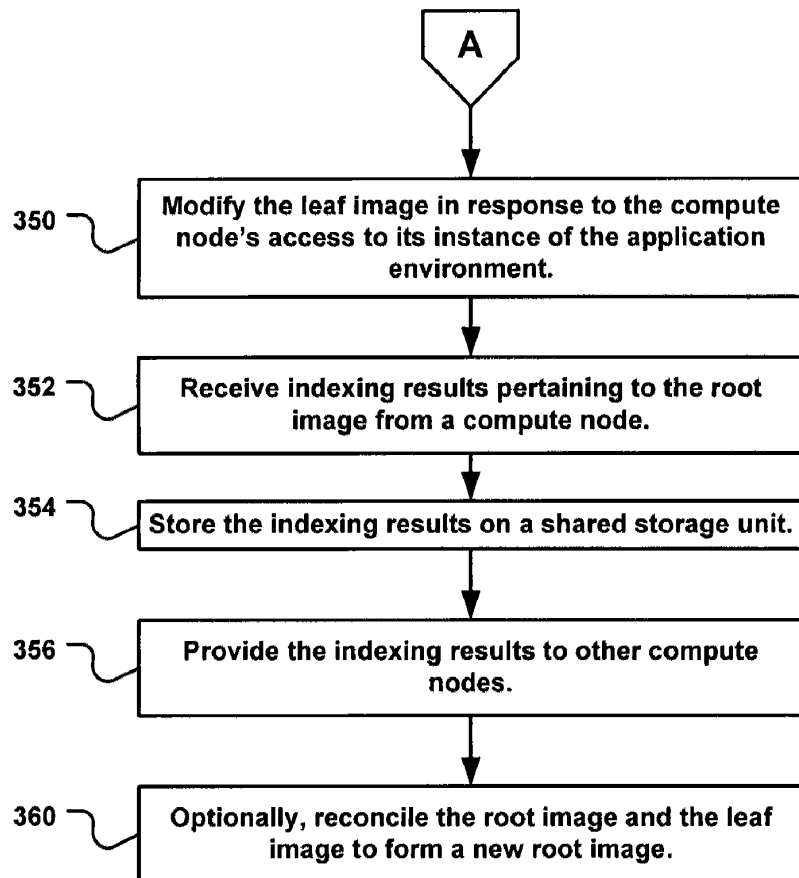


FIG. 3B

U.S. Patent

Dec. 11, 2012

Sheet 5 of 6

US 8,332,844 B1

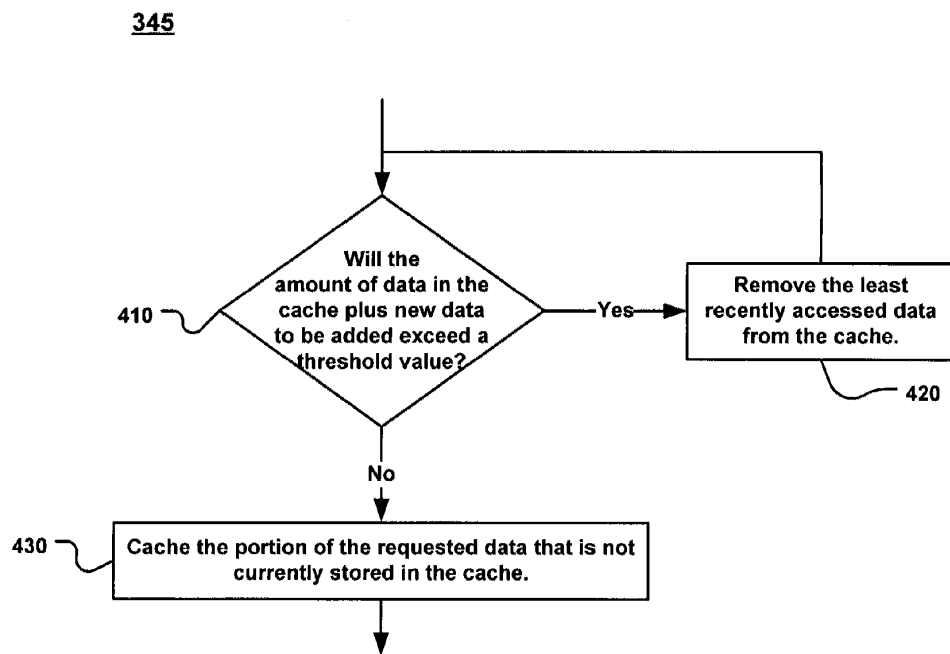


FIG. 4

U.S. Patent

Dec. 11, 2012

Sheet 6 of 6

US 8,332,844 B1

500

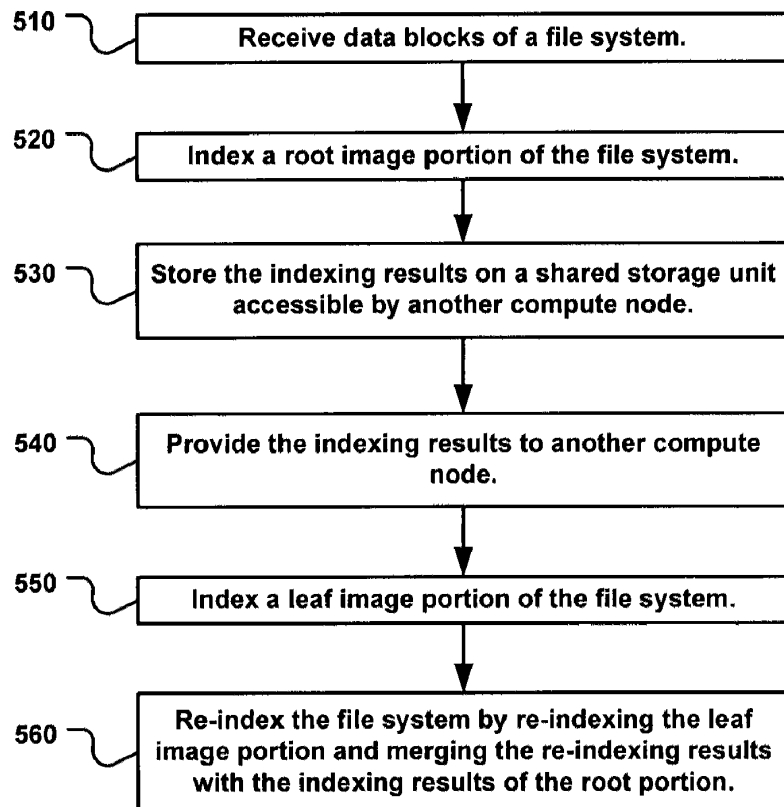


FIG. 5

US 8,332,844 B1

1

ROOT IMAGE CACHING AND INDEXING FOR BLOCK-LEVEL DISTRIBUTED APPLICATION MANAGEMENT

CLAIM OF PRIORITY UNDER 35 U.S.C. §120

The present application for patent is a continuation-in-part of application Ser. No. 11/395,816, entitled "BLOCK-LEVEL I/O SUBSYSTEM FOR DISTRIBUTED APPLICATION ENVIRONMENT MANAGEMENT," filed Mar. 30, 2006, now U.S. Pat. No. 7,721,282, assigned to the assignee hereof and hereby expressly incorporated by reference herein.

REFERENCE TO CO-PENDING APPLICATIONS FOR PATENT

The present application for patent is related to the following U.S. patent application:

"VIRUS SCANNING FOR BLOCK-LEVEL DISTRIBUTED APPLICATION MANAGEMENT," by Kulkarni et al., filed Feb. 28, 2007, now U.S. Pat. No. 8,065,737, assigned to the assignee hereof and expressly incorporated by reference herein.

BACKGROUND

Background

Over the years, as the internet has expanded and computers have multiplied, the need for clustered computing such as High Performance Computing (HPC) has increased. Clustered computing involves multiple compute nodes, usually a server grid, that work together to achieve a common task. For example, several (typically hundreds of) compute nodes may be clustered together to share the load of serving a high-traffic website. In large-scale systems such as this, a trend in software deployment is to centralize data management on a globally accessible file system with stateless computing nodes. A common example of this is Operating System (OS) software image management, where the compute nodes are activated with the distributed application environment by either diskless booting protocols or remote software installation to local storage. Under this architecture, a boot image is required for each compute node in the cluster. The boot image necessarily contains the kernel; it may additionally contain the application software that is intended to be run on the compute node.

The primary concern in clustered computing is low cluster bring-up time. The software that provides the boot images for the cluster typically stores a master boot image. It may then either pre-create clones of this master image for each such server, or it may create them "on the fly."

Creating a boot image on the fly involves copying the entire contents of the master image, which are typically in the range of 5-15 GB. Even with a significant amount of bandwidth by today's standards, this method will result in a large bring-up time.

Pre-creating a boot image for each server is advantageous from the point of view of cluster bring-up time. However, since one often does not know in advance how many servers will ever be booted, this scheme may result in wasted disk space.

Regardless of which of the preceding methods is used, both suffer from the same major problem—updating the boot image(s) for the cluster is cumbersome, as it means updating a number of copies of the boot image.

2

Additionally, once some compute nodes have booted, they will often engage in redundant activities with respect to each other. For example, assume that a cluster involves 20 compute nodes each running the same operating system and using substantially similar hardware. The 20 compute nodes will generally need to access much of the same data (e.g., drivers, library files, etc.). Moreover, when each of the 20 compute nodes index their file systems, the index results will only vary slightly to the extent that each compute node has developed its own "personality." Thus, to the extent that there is redundancy in the operations of the compute nodes, CPU resources, disk space, and data bus bandwidth are wasted.

In a branching store file system, a read-only base image (or "root" image) of the application environment is created. The root image is accessible by all compute nodes in the cluster. Changes made by a compute node to the root image are stored in a "leaf" image unique to that compute node. A filter operates between the compute nodes and the file system(s), which merges the changes recorded on the leaf images with the root image and delivers the result to the appropriate compute node. From the point of view of the compute node, it is running its own unique and cohesive instance of the application environment. While this system allows for creation of boot images on the fly without severely diminishing bring-up time, a separate version of the system must be created for each unique operating system because data is stored at the file system level (i.e., on a "per file basis"). Thus, migrating a computing cluster from one operating system to another is much more complicated than simply installing a new root image containing the new OS.

SUMMARY

Described herein is technology for, among other things root image caching and indexing for block-level distributed application management. The technology involves storing blocks of a root image on a first storage unit and storing blocks of leaf images on respective second storage units. The leaf images include additional data blocks not previously contained in the root image and changes made by respective compute nodes to the blocks of the root image. The technology includes caching blocks of the root image that have been accessed by at least one compute node. The technology also includes receiving indexing results pertaining to the root image from one compute node and providing the results for other compute nodes.

Thus, embodiments of the present disclosure provide an operating system-independent system and method for distributing an application environment to a compute node. By utilizing a root-leaf system of application environment storage, embodiments of the present disclosure allow creation of boot images on the fly without significantly diminishing bring-up time. This is due to the fact that creating a new boot image does not require copying the contents of the root image. Rather it involves registering a new UBD with the system, which occurs very quickly. Bring up time, and access time in general, can be further improved by caching commonly accessed the portions of the root image. Moreover, updating the boot image for the entire cluster simply involves updating the root image. Additionally, because of the commonality of the root image and the fact that its contents are not directly changed, certain operations performed on the root image (e.g., indexing) only need to be performed once by one compute node. Thereafter, the results of that operation can be

US 8,332,844 B1

3

shared with the other compute nodes in the cluster, thus saving the other compute nodes valuable time and resources.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a diagram of a suitable system on which embodiments may be implemented.

FIG. 2 is a diagram of a system, for root image caching and indexing in a block-level distributed application environment, in accordance with various embodiments of the present disclosure.

FIGS. 3A-3B are flowcharts illustrating a process for root image caching and indexing in a block-level distributed application environment, in accordance with various embodiments of the present disclosure.

FIG. 4 illustrates a flowchart for a process for updating the cache based on a read request, in accordance with an embodiment of the present disclosure.

FIG. 5 illustrates a flowchart for a process for indexing a root image, in accordance with various embodiments of the present disclosure.

DETAILED DESCRIPTION

Reference will now be made in detail to various embodiments of the disclosure, examples of which are illustrated in the accompanying drawings. While the disclosure will be described in conjunction with the various embodiments, it will be understood that they are not intended to limit the scope of the claims to these embodiments. On the contrary, the disclosure is intended to cover alternatives, modifications and equivalents, which may be included within the spirit and scope of the disclosure as defined by the claims. Furthermore, in the detailed description of the present disclosure, numerous specific details are set forth in order to provide a thorough understanding of the present disclosure. However, it will be obvious to one of ordinary skill in the art that the present disclosure may be practiced without these specific details. In other instances, well-known methods, procedures, components, and circuits have not been described in detail so as not to unnecessarily obscure aspects of the present disclosure.

Some portions of the detailed descriptions that follow are presented in terms of procedures, logic blocks, processing, and other symbolic representations of operations on data bits within a computer or digital system memory. These descriptions and representations are the means used by those skilled in the data processing arts to most effectively convey the substance of their work to others skilled in the art. A procedure, logic block, process, etc., is herein, and generally, conceived to be a self-consistent sequence of steps or instructions leading to a desired result. The steps are those requiring physical manipulations of physical quantities. Usually, though not necessarily, these physical manipulations take the form of electrical or Magnetic signals capable of being stored, transferred, combined, compared, and otherwise manipulated in a computer system or similar electronic computing device. For reasons of convenience, and with reference to common usage, these signals are referred to as bits, values, elements, symbols, characters, terms, numbers, or the like with reference to the present disclosure.

It should be borne in mind, however, that all of these terms are to be interpreted as referencing physical manipulations and quantities and are merely convenient labels and are to be interpreted further in view of terms commonly used in the art. Unless specifically stated otherwise as apparent from the discussion herein, it is understood that throughout discussions of the present embodiment, discussions utilizing terms

4

such as “determining” or “outputting” or “transmitting” or “recording” or “locating” or “storing” or “displaying” or “receiving” or “recognizing” or “utilizing” or “generating” or “providing” or “accessing” or “checking” or “notifying” or “delivering” or the like, refer to the action and processes of a computer system, or similar electronic computing device, that manipulates and transforms data. The data is represented as physical (electronic) quantities within the computer system’s registers and memories and is transformed into other data similarly represented as physical quantities within the computer system memories or registers or other such information storage, transmission, or display devices.

Briefly stated, described herein is technology for, among other things root image caching and indexing for block-level distributed application management. The technology involves storing blocks of a root image on a first storage unit and storing blocks of leaf images on respective second storage units. The leaf images include additional data blocks not previously contained in the root image and changes made by respective compute nodes to the blocks of the root image. The technology includes caching blocks of the root image that have been accessed by at least one compute node. The technology also includes receiving indexing results pertaining to the root image from one compute node and providing the results for other compute nodes.

Example Compute Node Operating Environment

With reference to FIG. 1, an example system for implementing embodiments includes a general purpose computing system environment, such as compute node 100. In its most basic configuration, compute node 100 typically includes at least one processing unit 102 and memory 104. Depending on the exact configuration and type of compute node, memory 104 may be volatile (such as RAM), non-volatile (such as ROM, flash memory, etc.) or some combination of the two. This most basic configuration is illustrated in FIG. 1 by dashed line 106. Additionally, compute node 100 may also have additional features/functionality. For example, compute node 100 may also include additional storage (removable and/or non-removable) including, but not limited to, magnetic or optical disks or tape. Such additional storage is illustrated in FIG. 1 by removable storage 108 and non-removable storage 110. Computer storage media includes volatile and nonvolatile, removable and non-removable media implemented in any method or technology for storage of information such as computer readable instructions, data structures, program modules or other data. Memory 104, removable storage 108 and nonremovable storage 110 are all examples of computer storage media. Computer storage media includes, but is not limited to, RAM, ROM, EEPROM, flash memory or other memory technology, CD-ROM, digital versatile disks (DVD) or other optical storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or any other medium which can be used to store the desired information and which can be accessed by compute node 100. Any such computer storage media may be part of compute node 100.

Compute node 100 may also contain communications connection(s) 112 that allow it to communicate with other devices. Communications connection(s) 112 is an example of communication media. Communication media typically embodies computer readable instructions, data structures, program modules or other data in a modulated data signal such as a carrier wave or other transport mechanism and includes any information delivery media. The term “modulated data signal” means a signal that has one or more of its characteristics set or changed in such a manner as to encode information in the signal. By way of example, and not limi-

US 8,332,844 B1

5

tation, communication media includes wired media such as a wired network or direct-wired connection, and wireless media such as acoustic, RF, infrared and other wireless media. The term computer readable media as used herein includes both storage media and communication media. Compute node 100 may also have input device(s) 114 such as a keyboard, mouse, pen, voice input device, touch input device, etc. Output device(s) 116 such as a display, speakers, printer, etc. may also be included. All these devices are well known in the art and need not be discussed at length here.

Example Systems

FIG. 2 is a diagram of a system 200 for root image caching and indexing in a block-level distributed application environment, in accordance with various embodiments of the present disclosure. In one embodiment, system 200 is implemented in a multi-computer system, such as an HPC cluster. In one embodiment, the application environment includes an operating system. In other embodiments, the application environment may contain other applications. System 200 has a number of compute nodes 220a-n coupled to first storage unit 240 and a corresponding second storage unit 250a-n through a corresponding union block device (UBD) 230a-n. To the compute nodes 220a-n, it appears that they have access to their own version of a distributed application environment. However, a separate and complete boot image is not created and stored for each compute node 220a-n.

System 200 has a first storage unit 240 for storing blocks of a root image of an application environment. The root image contains data initially common to the compute nodes 220a-n. The root image is not changed by compute nodes 220a-n. For example, in one embodiment, compute nodes 220a-n have read-only access to the first storage unit 240.

Moreover, each compute node 220a-n has a corresponding second storage unit 250a-n for storing a leaf image. The first storage unit 240 and second storage units 250a-n may each be contained on separate physical storage devices, on separate logical spaces on the same storage device, or any combination thereof. Regardless, the first storage unit 240 and the second storage units 250a-n may be contained within a single storage appliance. The leaf image may contain blocks of new data, blocks of changed data, or other blocks of data unique to the individual compute node. The leaf image may also contain a block modification log. In other words, a leaf image will describe the changes made by the respective compute node 220a-n to its instance of the application environment. Thus, when a compute node (e.g., node 220a) makes changes involving the root image, modifications are made to that compute node's leaf image (e.g., leaf image stored on second storage device 250a). With respect to changes to the root image, only the specific blocks that are changed are stored in the leaf image. For example, a particular file on the root image may comprise twenty blocks of data (e.g., blocks 1-20). One compute node (e.g., compute node 220a) desires to make a change to this file which involves a modification of only a few specific blocks of the file (e.g., blocks 4-9). In this example, only the modified blocks (e.g., blocks 4-9) will be stored in the compute node's leaf image (e.g., leaf image stored on second storage device 250a) plus some small overhead.

A compute node 220a-n mounts its instantiation of the application environment via its respective UBD 230a-n. In one embodiment, UBDs 230a-n are effectively low-level drivers that operate as an interface between the first and second storage devices and the file system of each compute node 220a-n. The file system may reside on the server side of the system 200. The file system may also reside on each of the compute nodes 220a-n. Because UBDs 230a-n operate below the file system, they are concerned merely with the blocks of

6

data themselves, rather than files they form. As a result, system 200 is completely file system, and thus operating system, independent.

UBDs 230a-n determine what leaf image (from the appropriate second storage unit 250) is needed for portions of the application environment that their respective compute nodes 220a-n have changed. UBDs 230a-n also locate the portions of the application environment that are not changed by their respective compute nodes 220a-n. These portions may reside in the root image. There may also be intermediate images (not depicted in FIG. 2) comprising versions of the root image from which a compute node's instance of the application environment is derived. Further, UBDs 230a-n create a new leaf image on a respective second storage unit 250a-n when their respective compute nodes 220a-n make changes to their instantiations of the application environment. In one embodiment, the UBDs 230a-n obtain the information necessary to create the application environment from a block modification log located in its respective leaf image.

UBDs 230a-n may also modify the leaf image in response to their respective compute node's access to its instance of the application environment. For example, upon receiving a write request from their respective compute nodes for a sector X, the UBDs 230a-n will create an appropriate persistent mapping for sector X and then write sector X onto their respective second storage units 250a-n, where sector X can then be modified. It will be appreciated that the data block being modified may already exist in the leaf image, in which case it does not need to be mapped and copied from the root image before modification.

By providing an application environment to multiple compute nodes via a branching store system, embodiments allow for a more streamlined update/patch procedure. In one embodiment, first storage device 240 may be refreshed with a new operating system patch, thereby automatically refreshing the image for all dependent compute nodes.

Caching

An embodiment of the present disclosure provides for caching of portions of the root image. Because several compute nodes in a cluster may often access the same data (e.g., same drivers, same library files, etc.) on the root image, tremendous speed improvements can be realized by caching such data in cache 260. For example, by current standards, it is common for a storage disk to achieve a data transfer rate around 76 MB/s, whereas cache memory can achieve speeds up to and greater than 800 MB/s. In one embodiment, the cache may be contained within a single storage appliance, along with the first storage unit 240 and the second storage units 250a-n.

The value of such caching becomes even more apparent in the context of initial boot sequences. For example, a cluster of 20 compute nodes may be booted for the first time. Since it is a first boot, the cache is empty. The first compute node to boot (e.g., compute node 220a) will therefore load its data directly from the pertinent disks (e.g., first storage unit 240 and second storage unit 250a). While the first compute node is loading its data, its reads are intercepted and cached on a per-block basis in cache 260. Thereafter, when a second compute node (e.g., compute node 220b) goes to boot, any data required by the second compute node that was previously accessed by the first compute node can be served out of the cache 260 rather than the first storage unit. It is appreciated that to the extent that the configurations of the first and second compute nodes are substantially the same, the relationship between the data requested by the second compute node and the data stored in the cache 260 approaches a 1:1 ratio (i.e., a greater than ten-fold improvement in root image access time is realized).

US 8,332,844 B1

7

It is further appreciated that the entire contents of the root image may not necessarily be accessed to achieve a boot (e.g., unused drives and the like). Therefore, the data capacity of cache **260** can be much less than that of the first storage unit **240**.

As new data is accessed on the root image that is not currently stored in the cache **260**, the cache **260** may thereafter be updated with the new data. Moreover, as the amount of data in the cache **260** approaches or exceeds a threshold value, certain data will need to be removed from the cache **260** to make way for the new data. The data to be removed, deleted, overwritten, etc., may be selected the basis of, for example, how recently the data was accessed, how frequently the data has been accessed, or a combination of both. It should be appreciated that other cache arbitration algorithms known in the art may similarly be used.

Indexing

An embodiment of the present disclosure provides for indexing the root image. From the standpoint of seek time, it is beneficial for each compute node to have access to an index of its file system. In a traditional clustered computing situation, each compute node would independently index its file system. To the extent that each compute node in a cluster has similar operating environments to the others, indexing performed on common data is therefore redundant.

As an added benefit of the architecture **200** depicted in FIG. **2**, because the data stored in the root image is by definition common to all the compute nodes **220a-n**, one compute node (e.g., compute node **220a**) can index the contents of the root image and then provide the indexing results to the other compute nodes (e.g., compute nodes **220b-n**). Thereafter, the compute nodes **220a-n** need only be concerned with indexing their respective leaf image, which are relatively small compared to the root image. Moreover, in embodiments where the first storage unit **240** is read-only, the root image will not have to be re-indexed because the contents of the root image do not change. It is appreciated that the indexing results can be provided to the other compute nodes in a number of ways. For example, the results may be stored on a shared storage unit (such as, but not limited to, cache **260**), the results may be provided directly to the other compute nodes, etc. Moreover, the initial indexing of the root image does not have to be performed by a compute node. For example, the indexing may be performed by a dedicated indexing host.

Example Methods

FIGS. 3A-3B are flowcharts illustrating a process **300** for root image caching and indexing in a block-level distributed application environment, in accordance with various embodiments of the present disclosure. Although specific operations are disclosed in process **300**, such operations are exemplary. Process **300** may not include all of the operations illustrated by FIGS. 3A-3B. Also, process **300** may include various other operations and/or variations of the operations shown by FIGS. 3A-3B. Likewise, the sequence of the operations of process **300** can be modified. Steps of process **306** may be stored as instructions on a computer readable medium and executed on a computer processor.

Step **310** involves storing blocks of a root image of the compute node on a first storage unit. By storing data at the block level, embodiments are able to operate beneath the file system and thus are designed to be file system and operating system independent.

Step **320** involves storing a leaf image on a second storage unit. The leaf image includes, but is not limited to, new data blocks for the compute node and blocks of the root image that the compute node has changed. The leaf image includes a block modification log in one embodiment.

8

Step **322** involves caching blocks of the root image that have been accessed by a compute node. In doing so, embodiments improve the access time for subsequent compute nodes requesting to access the same blocks of data.

Step **324** involves receiving a read request from a compute node. At step **326**, a determination is made as to whether the data requested by the compute node is currently stored in the cache. If a portion of the data is currently stored in the cache, that portion is then served from the cache (step **328**), thus improving the access time with respect to that portion of data as compared to otherwise retrieving it from a storage unit such as a hard disk.

Step **330** involves merging the blocks of the root image, the blocks of the leaf image, and the relevant blocks of the cache, if any, to create the application environment. In other words, the merging occurs at an operational level between the file system of a compute node and the first storage unit, the corresponding second storage unit, and the cache. Once the application environment has been created, it will appear to the compute node as one cohesive image rather than a base image plus its additions, deletions, and modifications. To the compute node, it appears that it has access to its own unique version of an application environment. However, a separate and complete boot image is not actually stored for the compute node.

Step **340** involves delivering the application environment to the compute node. Step **340** may comprise a low-level driver determining which data blocks are needed for the compute node's instance of the application environment and delivering the application environment to the compute via the compute node's file system.

Step **345** involves updating the cache based on the read request. For instance, to the extent that a compute node has accessed root image data not currently stored in the cache, the cache should be updated to include that cache. In some cases, the amount of data in the cache may approach a capacity limit, either due to the physical capacity of the cache itself or due to threshold limit set in software.

FIG. **4** illustrates a flowchart for a process **345** for updating the cache based on a read request, in accordance with an embodiment of the present disclosure. It should be appreciated that the steps illustrated in FIG. **4** are exemplary and the updating performed in block **345** of process **300** may be achieved in many other ways as well. Process **345** may not include all of the operations illustrated by FIG. **4**. Also, process **345** may include various other operations and/or variations of the operations shown by FIG. **4**. Likewise, the sequence of the operations of process **345** can be modified. Steps of process **345** may be stored as instructions on a computer, readable medium and executed on a computer processor.

At step **410**, a determination is made as to whether the amount of data currently in the cache plus new data to be added exceeds a threshold value (physical or software-imposed). If the threshold value will be exceeded, the least recently accessed data is removed from the cache (step **420**). This sequence repeats until there is adequate headroom in the cache to accommodate the new data. Once it is determined that the amount of data currently in the cache plus the new data to be added will not exceed the threshold value, FIG. **4** proceeds to step **430**. Step **430** involves caching the portion of the requested data that is not currently stored in the cache (i.e., the "new data").

With reference again to FIG. **3B**, step **350** involves modifying the leaf image in response to the compute node's access to its instance of the application environment. The modifying of the leaf image may include copying one or more data

US 8,332,844 B1

9

blocks from the root image to the leaf image, and modifying the data block in the leaf image. It will be appreciated that the data block being modified may already exist in the leaf image, in which case it does not need to be copied from the root image before modification.

Step 352 involves receiving indexing results pertaining to the root image from a compute node. Step 356 involves providing the indexing results to other compute nodes. It is appreciated that this may be done in a number of ways. For example, the indexing results may be provided directly to the compute nodes. Alternatively, the indexing results may be stored on a shared storage unit (step 354) and then provided to the compute nodes via the shared storage unit. By receiving the indexing result of the root image from one compute node and providing it to the rest of the compute nodes, valuable time, resources, and bandwidth are saved for the other compute nodes. Since the root image is common to all the compute nodes, any re-indexing of the root image would be redundant and a waste of resources.

Step 360 involves reconciling the root image and the leaf image to form a new root image. This may be desirable if, for example, the leaf image has grown to exceed a particular disk quota. Furthermore, if there are multiple compute nodes that access the root image, each having their own respective leaf image, and there is substantial commonality between the leaf images, it may also be beneficial to reconcile the leaf images with the root image.

FIG. 5 illustrates a flowchart for a process 500 for indexing a root image, in accordance with various embodiments of the present disclosure. Although specific operations are disclosed in process 500, such operations are exemplary. Process 500 may not include all of the operations illustrated by FIG. 5. Also, process 500 may include various other operations and/or variations of the operations shown by FIG. 5. Likewise, the sequence of the operations of process 500 can be modified. Steps of process 500 may be stored as instructions on a computer readable medium and executed on a computer processor.

Step 510 involves receiving data blocks of a file system at a compute node. The data blocks include a root image portion in the leaf image portion. As described above, the leaf image portion includes additional data blocks not previously contained in the root image portion and also changes made by the compute node to the blocks of the root image. Thus, the file system actually seen by the compute node is a product of merging the root image portion in the leaf image portion together at the block level.

Step 520 involves the compute node indexing the root image portion of its file system. In such a case, the compute node is to some extent agnostic to the fact that its file system is divided between a root image portion and a leaf image portion. At step 540, the compute node provides the indexing results of the root image portion to another compute node. It is appreciated that this may be achieved a number of ways. For example, the first compute node may simply provide the indexing results to the second compute node directly. Alternatively, the first compute node may store the indexing results on a shared storage unit that is accessible by the second compute node (step 530).

Thereafter, the compute node may complete its indexing of its file system by indexing the leaf image portion of the file system (step 550). Further down the road, the benefits of indexing the root image separately from the leaf image are realized when the compute node requires that its file system be re-indexed. At step 560, the compute node re-indexes its file system by re-indexing its corresponding leaf image portion with the previous indexing results of the root portion.

10

Thus, by operating at the block level, embodiments of the present disclosure provide file system and operating system independent systems and methods for distributing an application environment to a compute node. By utilizing a branching store system of application environment distribution, embodiments of the present disclosure allow creation of boot images on the fly without significantly diminishing bring-up time. This is due to the fact that creating a new boot image does not require copying the contents of the root image, but rather it involves registering a new UBD with the system, which occurs very quickly. Bring up time, and access time in general, can be further improved by caching commonly accessed the portions of the root image. Moreover, updating the boot image for the entire cluster simply involves updating the root image.

Embodiments of the present disclosure also allow for scalability and redundancy. For example, a server containing the root image may only have the resources to supply the root image to 100 compute nodes. Thus; in order to implement a 200-compute node system, two servers, each containing a copy of the root image, are used. This scalability provided by embodiments also lends itself to dynamic scaling. In other words, the number of root images required for a specific configuration can change based on the I/O workload, and new root images can be created on the fly accordingly. Moreover, additional servers containing the root image may be added into the system to provide redundancy and increase reliability. For example, and with reference to FIG. 2, cache 261, first storage unit 241, and second storage units 251a-n provide failover/redundancy security for system 200. As an added security measure, some of the root nodes may be located remotely from each other (e.g., half located in a main office and half in a satellite office). In addition, it is possible to copy the contents of one or more leaf nodes into a non-UBD (regular block device) copy (e.g., for use as an isolated backup).

Moreover, because of the commonality of the root image and the fact that its contents are not directly changed, certain operations performed on the root image (e.g., indexing) only need to be performed once by one compute node. Thereafter, the results of that operation can be shared with the other compute nodes in the cluster, thus saving the other compute nodes valuable time and resources.

The previous description of the disclosed embodiments is provided to enable any person skilled in the art to make or use the present disclosure. Various modifications to these embodiments will be readily apparent to those skilled in the art, and the generic principles defined herein may be applied to other embodiments without departing from the spirit or scope of the disclosure. Thus, the present disclosure is not intended to be limited to the embodiments shown herein but is to be accorded the widest scope consistent with the principles and novel features disclosed herein.

What is claimed is:

1. A system for providing data to a plurality of compute nodes, the system comprising:

- a first storage unit configured to store blocks of a root image of said compute nodes;
- a plurality of second storage units configured to store leaf images of respective compute nodes, said leaf images including only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of said root image, wherein said leaf images of respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes; and

US 8,332,844 B1

11

a cache configured to cache blocks of said root image previously accessed by at least one of said compute nodes.

2. The system as recited in claim 1 wherein said cache is configured to store X most recently accessed blocks of said root image, and wherein X represents a cache threshold value.

3. The system as recited in claim 1 wherein said first storage unit, said second storage units, and said cache are contained within a single storage appliance.

4. The system as recited in claim 1 further comprising:

a plurality of union block devices configured to interface between respective compute nodes and said first storage unit, respective second storage units, and said cache to distribute application environments to the compute nodes, wherein said union block devices are configured to create said application environments by merging the blocks of said root image with the blocks of respective leaf images.

5. The system as recited in claim 4 wherein said union block devices comprise low-level drivers for interfacing between the file systems of respective compute nodes and said first storage unit, respective second storage units, and said cache.

6. The system as recited in claim 1 wherein said first storage unit is read-only.

7. A method for providing data to a plurality of compute nodes, comprising:

storing blocks of a root image of said compute nodes on a first storage unit;

storing leaf images for respective compute nodes on respective second storage units, said leaf images including only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of the root image, wherein said leaf images of respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes; and

caching blocks of said root image that have been accessed by at least one of said compute nodes in a cache memory.

8. The method as recited in claim 7 further comprising:

receiving a read request from at least one of said compute nodes, wherein a first portion of the data requested is currently stored in said cache memory; and

providing said first portion of said data to said at least one of said compute nodes from said cache memory.

9. The method as recited in claim 8 further comprising:

updating said cache memory based on said read request.

10. The method as recited in claim 9 wherein a second portion of the data requested is not currently stored in said cache memory and said updating comprises:

caching said second portion in said cache memory; and removing the least recently accessed data from said cache memory if the amount of data in said cache memory is above a threshold value.

11. The method as recited in claim 7 further comprising:

merging the blocks of said root image with the blocks of respective leaf images to create cohesive respective application environments.

12. The method as recited in claim 11 wherein said merging occurs at an operational level between file systems of the respective compute nodes and said first storage unit, respective second storage units, and said cache memory.

13. The method as recited in claim 7 wherein said first storage unit is read-only.

14. A system for indexing file systems for a plurality of compute nodes, the system comprising:

12

a first storage unit configured to store blocks of a root image of said compute nodes;

a plurality of second storage units configured to store leaf images of respective compute nodes, said leaf images comprising only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of said root image, wherein said leaf images of respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes; and

a plurality of union block devices corresponding to said compute nodes, said union block devices configured to interface between said compute nodes and said first and second storage units to distribute said file systems to said compute nodes, wherein said union block devices are configured to create said file systems by merging the blocks of said root image stored on the first storage unit with the blocks of respective leaf images stored on respective second storage units, and wherein further at least one of said compute nodes is configured to index said root image and provide the indexing results to another of said compute nodes.

15. The system as recited in claim 14 wherein said first storage unit and said second storage units are contained within a single storage appliance.

16. The system as recited in claim 14 further comprising:

a plurality of union block devices configured to interface between respective compute nodes and said first storage unit and respective second storage units, said union block devices configured to distribute application environments to the compute nodes, wherein said union block devices are configured to create said application environments by merging the blocks of said root image with the blocks of respective leaf images.

17. The system as recited in claim 16 wherein said union block devices comprise low-level drivers for interfacing between the file systems of respective compute nodes and said first storage unit, respective second storage units, and said cache.

18. The system as recited in claim 14 wherein said first storage unit is read-only.

19. A method for indexing file systems for a plurality of compute nodes, comprising:

storing blocks of a root image of said compute nodes on a first storage unit;

storing leaf images for respective compute nodes on respective second storage units, said leaf images comprising only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of the root image, wherein said leaf images for respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes;

merging the blocks of said root image with the blocks of respective leaf images stored on respective second storage units to create respective file systems for respective compute nodes;

receiving indexing results pertaining to said root image from one of said compute nodes; and

providing said indexing results to the others of said compute nodes.

20. The method as recited in claim 19 further comprising: storing said indexing results on a shared storage unit.

21. The method as recited in claim 19 wherein said merging occurs at an operational level between respective file systems of the compute nodes and said first storage unit and respective second storage units.

US 8,332,844 B1

13

22. The method as recited in claim 19 wherein said first storage unit is read-only.

23. A computer-readable storage medium having instructions stored thereon that, in response to execution by at least one computing device, cause the at least one computing device to:

receive data blocks of a file system, said data blocks comprising a root image portion and leaf image portion, said leaf image portion comprising only additional data blocks not previously contained in said root image portion and, changes made by said first compute node to the blocks of said root image, wherein said leaf image portion does not include blocks of said root image that are unchanged by said first compute node, wherein said file system is the result of merging said root image portion and said leaf image portion together at the block-level; index said root image portion; and provide the results of said indexing to a second compute node, wherein said logic encoded in the one or more

14

tangible media comprise computer executable instructions executed by the first compute node.

24. The computer-readable storage medium of claim 23, wherein the instructions further cause the at least one computing device to store said results of said indexing on a shared storage unit accessible by said second compute node.

25. The computer-readable storage medium of claim 23, wherein the instructions further cause the at least one computing device to index said leaf image portion.

26. The computer-readable storage medium of claim 25, wherein the instructions further cause the at least one computing device to re-index said file system by re-indexing said leaf image portion and merging the results of said re-indexing of said leaf image portion with said results of said indexing of said root image portion.

27. The computer-readable storage medium of claim 23 wherein said root image portion is read-only.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

PATENT NO. : 8,332,844 B1
APPLICATION NO. : 11/709477
DATED : December 11, 2012
INVENTOR(S) : Kulkarni et al.

Page 1 of 1

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

On Title Page 2, item (56), under "OTHER PUBLICATIONS", in Column 2, Lines 28-30, delete
"OpenMP Fortran.....34-36."

In the Specifications:

In Column 3, Line 8, delete "system," and insert -- system --, therefor.

In Column 3, Line 54, delete "Magnetic" and insert -- magnetic --, therefor.

In Column 7, Line 55, delete "306" and insert -- 300 --, therefor.

In Column 8, Line 50, delete "computer, readable" and insert -- computer readable --, therefor.

In the Claims:

In Column 13, Line 11, in Claim 23, delete "and," and insert -- and --, therefor.

Signed and Sealed this
Eleventh Day of June, 2013



Teresa Stanek Rea
Acting Director of the United States Patent and Trademark Office

EXHIBIT A-3

Attorney Docket No.:PNTA-P002

Declaration and Power of Attorney for a Patent Application

Declaration

As below named inventor, I hereby declare that my residence post office address, and citizenship are as stated below my name. Further, I hereby declare that I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled:

BRANCHING STORE FILE SYSTEM

the specification of which:

 X is attached hereto, or
 was filed on as application serial no. : and
 was amended on

I hereby state that I have reviewed and understand the contents of the above identified specification, including the claims, as amended by any amendment referred to above; and

I acknowledge the duty to disclose information which is material to the examination of this application in accordance with Title 37, Code of Federal Regulations, Section 1.56.

Foreign Priority Claim

I hereby claim foreign priority benefits under Title 35, United States Code Section 119 of any foreign application(s) for patent or inventor's certificate listed below and have also identified below any foreign application for patent or inventor's certificate having a filing date before that of the application on which priority is claimed:

Number	Country	Date Filed	Priority Claimed
<u> </u>	<u> </u>	<u> </u>	<u> </u> yes <u> </u> no
<u> </u>	<u> </u>	<u> </u>	<u> </u> yes <u> </u> no

U.S. Priority Claim

I hereby claim the benefit under Title 35, United States Code, Section 120 and 119(e) of any United States application(s) listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States application in the manner provided by the first paragraph of Title 35, United States Code, Section 112, I acknowledge the duty to disclose material information as defined in Title 37, Code of Federal Regulations, Section 1.56(a) which occurred between the filing date of the prior application and the national or PCT international filing date of this application:

Serial Number	Filing Date	Status (patented/pending/abandoned)
<u> </u>	<u> </u>	<u> </u>
<u> </u>	<u> </u>	<u> </u>

BEST AVAILABLE COPY

Attorney Docket No.:PNTA-P002

Power of Attorney

As a named inventor, I hereby appoint practitioners associated with this Customer Number:

41066

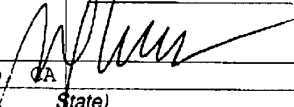
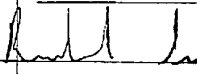
as attorney(s) or agent(s) to represent the undersigned before the United States Patent and Trademark Office (USPTO) in connection with any and all patent applications assigned only to the undersigned according to the USPTO assignment records or assignment documents attached to this form in accordance with 37 CFR 3.73(b).

Send Correspondence to:

WAGNER, MURABITO & HAO LLP
Two North Market Street
Third Floor
San Jose, California 95113
(408) 938-9060

Signatures

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

Full Name of Sole/First Inventor: Tung M. NGUYENInventor's Signature Residence Cupertino, CA
(City State)Citizenship USADate 12/27/2004P.O. Address 22346 Mc. Cellan Road, Cupertino, CA 95014Full Name of Second Inventor: Richard AUInventor's Signature Residence Woodside, CA
(City State)Citizenship USADate 12/23/2004P.O. Address 389 Moore Road, Woodside, CA 94062Full Name of Third Inventor: Eric POUYOULInventor's Signature Residence San Francisco, CA
(City State)Citizenship FranceDate 12-27-2004P.O. Address 350 Day Street, San Francisco, CA 94131**BEST AVAILABLE COPY**

BRANCHING STORE FILE SYSTEM

ABSTRACT

A branch and store file system. The file system includes a root file system having a base image of a distributed application. The base image is read-only accessible by compute nodes. The file system also has leaf images of the distributed application for respective ones of the compute nodes. The leaf images include changes made to instances of the base image by respective compute nodes. The file system also has a filter that controls accesses of the distributed application by the compute nodes. The compute nodes' respective instances of the distributed application are based on the base image and the compute node's leaf image.

UNITED STATES PATENT APPLICATION FOR
BRANCHING STORE FILE SYSTEM

Inventors:

Tung M. Nguyen

Richard Au

Eric Pouyoul

Prepared by:

WAGNER, MURABITO & HAO, LLP

TWO NORTH MARKET STREET

THIRD FLOOR

SAN JOSE, CALIFORNIA 95113

(408) 938-9060

PNTA-P002 JPH/RMP

CONFIDENTIAL

BRANCHING STORE FILE SYSTEM

FIELD OF THE INVENTION

[0001] Embodiments of the present invention relate to the field of networked computing. Specifically, embodiments of the present invention relate to distributed application deployment in a multi-computer system.

BACKGROUND ART

[0002] In large-scale systems, a trend in software deployment is to centralize Operating System (OS) software image management on a globally accessible file system with stateless computing nodes. The compute nodes are activated with the distributed application by either diskless booting protocols or remote software installation to local storage.

[0003] One approach to software image management is to use a single image with global process dispatch to lightweight OS node computing environments. A second approach is to dedicate one image per compute node.

[0004] Neither of the two general approaches provides fault isolation between different instances of the distributed application while being scalable and efficient. For example, the single OS image approach does not provide fault isolation between different instances of the distributed application. Moreover, a single OS image is a bottleneck that becomes worse as the size of the system grows.

PNTA-P002 JPH/RMP

CONFIDENTIAL

[0005] The multiple OS image approach does not use system resources efficiently and is also difficult and time-consuming to install and maintain. For example, multiple OS images result in duplicated storage; therefore, system resources are not used efficiently. Moreover, using multiple OS images adds overhead to software installation and maintenance. Therefore, while multiple OS images provide isolation between instances of the distributed application, system resources are not used efficiently and installation and maintenance is time-consuming.

SUMMARY OF THE INVENTION

[0006] A branching store file system is disclosed. In one embodiment in accordance with the present invention, the file system includes a root file system containing the operating environment or base image of a distributed application, any number of shared branch images, and at least one dedicated leaf image for each compute node. The base image and shared branch images are read-only accessible by compute nodes. Shared branch images contain changes made to the base image which are visible to all compute nodes. The dedicated leaf images include further changes made to the base or branch images, visible only by respective compute nodes. The file system also has a filter that controls accesses of the distributed application by the compute nodes. A compute node's instance of the distributed application is based on the base image, branch image(s), and the compute node's leaf image(s).

[0007] In another embodiment of the present invention, the branching store file system has branch modification logs for changes to the distributed application. The system has a root file system with a root metadata structure and a root data structure. The entries in the root metadata structure point to associated entries in the root data structure. The branching store file system also has branch modification logs for respective compute nodes. A given branch modification log has a branch data structure comprising modified data files reflecting changes made to

PNTA-P002 JPH/RMP

CONFIDENTIAL

an instance of the root data structure by a compute node. The branch modification log also has a branch metadata structure comprising unmodified and modified entries. The modified entries reflect changes made by the compute node to its instance of the root data structure. Further, the unmodified entries point to associated entries in the root data structure and the modified entries point to associated ones of the modified data files.

[0008] In another embodiment of the present invention, the branching store file system comprises a base instance of a distributed application and change volumes for changes made by compute nodes to their respective instances of the distributed application. The system also has a filter program code that directs accesses of the compute nodes' respective instances of the distributed application.

BRIEF DESCRIPTION OF THE DRAWINGS

[0009] The accompanying drawings, which are incorporated in and form a part of this specification, illustrate embodiments of the invention and, together with the description, serve to explain the principles of the invention:

[0010] Figure 1 is a diagram of a branching store file system with leaf images, in accordance with an embodiment of the present invention.

[0011] Figure 2 is a diagram of a branching store file system using branch modification logs, in accordance with an embodiment of the present invention.

[0012] Figure 3 illustrates an exemplary branching store architecture using branch modification logs, in accordance with an embodiment of the present invention.

[0013] Figure 4 is a diagram of branching store file system employing copy on write to change volumes, in accordance with an embodiment of the present invention.

[0014] Figure 5 is a tree structure that illustrates a branching file system, in accordance with an embodiment of the present invention.

PNTA-P002 JPH/RMP

CONFIDENTIAL

[0015] Figure 6 is diagram illustrating modules and structures of a file system, in accordance with an embodiment of the present invention.

[0016] Figure 7 is flowchart illustrating a process of a branching file system, in accordance with an embodiment of the present invention.

DETAILED DESCRIPTION OF THE INVENTION

[0017] In the following detailed description of embodiments of the present invention, a branching store file system, numerous specific details are set forth in order to provide a thorough understanding of the present invention. However, it will be recognized by one skilled in the art that the present invention may be practiced without these specific details or with equivalents thereof. In other instances, well-known methods, procedures, and components have not been described in detail as not to unnecessarily obscure aspects of the present invention.

NOTATION AND NOMENCLATURE

[0018] Some portions of the detailed descriptions that follow are presented in terms of procedures, steps, logic blocks, processing, and other symbolic representations of operations on data bits that can be performed on computer memory. These descriptions and representations are the means used by those skilled in the data processing arts to most effectively convey the substance of their work to others skilled in the art. A procedure, computer executed step, logic block, process, etc., is here, and generally, conceived to be a self-consistent sequence of steps or instructions leading to a desired result. The steps are those requiring physical manipulations of physical quantities. Usually, though not necessarily, these quantities take the form of electrical or magnetic signals capable of being stored, transferred, combined, compared, and otherwise manipulated in a computer system. It

PNTA-P002 JPH/RMP

CONFIDENTIAL

has proven convenient at times, principally for reasons of common usage, to refer to these signals as bits, values, elements, symbols, characters, terms, numbers, or the like.

[0019] It should be borne in mind, however, that all of these and similar terms are to be associated with the appropriate physical quantities and are merely convenient labels applied to these quantities. Unless specifically stated otherwise as apparent from the following discussions, it is appreciated that throughout the present invention, discussions utilizing terms such as “removing” or “creating” or “generating” or “freezing” or “restoring” or “computing” or “resolving” or “communicating” or “determining” or “maintaining” or “providing” or “storing” or “constructing” or “performing” or “initializing” or “selecting” or “forming” or the like, refer to the action and processes of a computer system, or similar electronic computing device, that manipulates and transforms data represented as physical (electronic) quantities within the computer system's registers and memories into other data similarly represented as physical quantities within the computer system memories or registers or other such information storage, transmission or display devices.

[0020] Embodiments of the present invention provide methods and systems for distributed application deployment in a multi-compute node environment. Embodiments of the present invention provide isolation and

PNTA-P002 JPH/RMP

CONFIDENTIAL

fault containment typical of multiple OS image approaches with operation performance typical of a single OS image approach. Furthermore, embodiments of the present invention allow the flexibility to customize the application environment that is unmatched by either approach. Embodiments of the present invention are highly scalable. Embodiments of the present invention provide these advantages and others not specifically mentioned above but described in the sections to follow.

[0021] Figure 1 is a diagram of branching store system 100 in accordance with an embodiment of the present invention for enabling distributed application deployment in a multi-computer system 116. The branching store system 100 has a number of compute nodes 114 coupled to file system 102 through a filter 110. To the compute nodes 114 it appears that they have access to their own version of a distributed application 117. However, the file system 102 does not store a separate and complete version of the distributed application 117 for each compute node 114.

[0022] The branching store system 100 has a file system 102 containing a base image or instance 104 of a distributed application. The base instance 104 is not changed by compute nodes 114. For example, the compute nodes 114 have read-only access to the base image 104. However, if changes are made to the base image 104, those changes are essentially instantaneously known to the compute nodes 114.

[0023] Moreover, the file system 102 has a number of leaf images 105. For example, each compute node 114a-n has a corresponding leaf image 105a-n. A given leaf image 105 describes changes made by the respective compute node 114 to its instance of the distributed application. Thus, when a compute node (e.g., node 114a) make changes involving the base image 104, modifications are made to that compute node's leaf image (e.g., leaf image 105a). A leaf image 105 may comprise versions of entire files of the distributed application; however, this is not required. In one embodiment, the changes in the leaf image 105 are performed on a data block-by-data block basis. In another embodiment, the changes in the leaf image 105 are performed on a file-by-file basis. There may also be intermediate images (not depicted in Fig. 1) between the base image 104 and the leaf images 105.

[0024] A compute node 114 mounts its instantiation of the distributed application via the filter 110. The filter 110 determines what leaf image 105 is needed for portions of the distributed application that the compute node 114 has changed. The filter 110 also locates the portions of the distributed application that are not changed by the compute node 114. These portions may reside in the base image 104. There may also be intermediate images (not depicted in Fig. 1) comprising versions of the base image 104 from which a compute node's instance of the distributed application is derived.

Further, the filter 110 creates new leaf images 105 when a compute node 114 makes changes to its instantiation of the distributed application.

[0025] By maintaining a separate instance of the distributed application for each compute node 114, embodiments of the present invention provide fault isolation between instances of the distributed application. Furthermore, by recording only changes in the leaf images, embodiments of the present invention conserve system resources. Furthermore, updates to the base image are provided to the compute nodes nearly instantaneously.

[0026] Figure 2 is a diagram of branching store system 200 in accordance with an embodiment of the present invention in which a branch modification log is used to record changes to the base instance of the distributed application. The system 200 has a number of compute nodes 114a-n coupled to network 112 that have access to a file system 102 through a filter 110. To the compute nodes 114a-n, it appears that they have access to their own version of a distributed application 117a-n, respectively. However, the file system 102 does not store a separate and complete version of the distributed application 117 for each compute node 114.

[0027] The branching store system 100 has a file system 102 containing a base image or instance 104 of a distributed application (e.g., a base operating system). Briefly, the base image 104 contains data and

metadata for the base image 104 of the distributed application. The base image 104 is not changed by compute nodes 114. For example, the compute nodes 114 have read-only access to the base image 104. However, if changes are made to the base image 104, those changes are essentially instantaneously known to the compute nodes 114.

[0028] Moreover, the file system 102 has a number of branch modification logs 205a-n. For example, each compute node 114a-n has a corresponding branch modification log 205a-n. The branch modification log 205 is one embodiment of a leaf image (Fig. 1, 105). A given branch modification log (e.g., 205a) describes changes made by the respective compute node (e.g., 114a) to its instance of the distributed application (e.g., 117). Thus, when a compute node 114 make changes involving the base image 104, modifications are made to that compute node's branch modification log 205. A compute node's instance is thus defined by at least a portion of the base image 104 and its branch modification log 205. This is indicated by the groupings 108a-n. It will be understood that a compute node's instance of the distributed application does not typically include all of the grouping 108. For example, some of the base image 104 may be copied into a branch modification log 205 and modified therein.

[0029] A branch modification 205 contains metadata and data for a respective compute node's instance of the distributed application. However,

the data in a given branch modification log 205 only reflects differences between the data in the base image 104 and the given compute node's instance of the distributed application, in this embodiment. A compute node 114 mounts its instance of the distributed application as pointed to by its branch modification log 205. For example, a compute node 117a accesses the base image 104 for data that is unchanged and accesses its branch modification log 205a for data that the compute node 114a has changed. Thus, this embodiment traverses inodes to determine the location of data for a compute node's instance of a distributed application.

[0030] Figure 3 illustrates an exemplary branching store architecture 150, in accordance with an embodiment of the present invention. The exemplary branching store architecture 150 may be used in the system depicted in Figure 1 or Figure 2, although systems 100 and 200 are not limited to this architecture 150. The branching store architecture 150 has a root file system 104 (also referred to herein as a base image) and a branch modification log 205. The branch modification log 205 is one embodiment of a leaf image (Fig. 1, 105). Typically, there will be many branch modification logs 205, such that one is available for each compute node in the system.

[0031] The root file system 104 has a root metadata structure 205 and a root data structure 210. Entries in the root metadata structure 205 point to associated entries in the root data structure 210. The root data structure

210 comprises the base instance of the distributed application. Neither the root data structure 210 nor the root metadata structure 205 are modified by compute nodes. However, a system management node or the like may modify the root file system 104, such that changes to the distributed application are easily and rapidly deployed to compute nodes.

[0032] The branch modification log 205 has a branch data structure 310 comprising modified data reflecting changes made to an instance of the root data structure by a compute node. For example, when a compute node makes changes to its instance of the distributed application data is added or modified in the branch data structure 310 for that compute node.

[0033] The branch modification log 205 also comprises a branch metadata structure 305 comprising unmodified entries 320 and modified entries 315. The modified entries 315 reflect changes made to the instance of the distributed application by the compute node. Thus, a modified entry 135 may be an entry that has a modification attribute attached to the entry. A modified entry 315 points to an associated modified data block 325. For example, if an entry has an attribute indicating that it is a modified entry, then inodes are followed to the correct block 325 in the branch data structure 310.

[0034] The unmodified entries 325 point to associated entries in the root data structure 210. For example, if an entry does not have an attribute indicating that it is a modified entry, then the file lookup is re-directed such that inodes are followed to the root data structure 210.

[0035] The instance of the distributed application is thus defined by the branch metadata structure 305, wherein an unmodified portion of the distributed application resides in the root data structure 210 and a modified portion of the distributed application resides in the branch data structure 310.

[0036] Figure 4 is a diagram of branching store system 400 in accordance with an embodiment of the present invention in which a copy on write to change volumes 405 are used to record changes to the base instance of the distributed application. Changes to a compute node's instance of the distributed application, may also be kept as a database referred to as a change volume 405.

[0037] The file system 102 comprises a base image 104 of a distributed application 104 (e.g., a base OS) and a number of change volumes 405a-n. When a compute node 114 performs a write operation on a file, the entire original file is copied from the root image 104 into the change volume 405, if

the file does not already exist in the change volume 405. The write operation is then performed on the copy in the change volume 405.

[0038] When a compute node 114 mounts the distributed application, the compute node 114 mounts at least a portion of the base image 104 and the change volume 405. Access to files in a compute node's instance of the distributed application may proceed by first determining if the file exists in the compute node's 114 change volume 405, then determining if the file is in the base image 104.

[0039] There is a filter 110 at each compute node 114, in this embodiment. Alternatively, there could be a single filter 110 at the file system 102. The implementation of the filter 110 in this embodiment may be different than the implementation of the filter 110 of system 200 of Figure 2.

Application Groups

[0040] An embodiment of the present invention provides for application groups (AG). An AG is a way to provide a group of compute nodes their own instantiation of an application, while allowing versions of the application to be frozen from time to time. Thus, an archive of the application is created.

[0041] At creation time, an AG comprises the original root file system. A number of leaf nodes or images are created to match the desired capacity

demand of the AG. The leaf images can be created very quickly and require very little storage because the leaf image reflect changes to the root instance. In one embodiment, the leaf images comprise BMLs. In another embodiment, the leaf images comprise change volumes.

[0042] Figure 5 is a tree structure 500 that illustrates a branching file system, in accordance with an embodiment of the present invention. Some nodes are read-only, others are read-write-modify. A node may change from read-write-modify to read-only. For example, creating a new edition of the AG may be accomplished by freezing a current leaf image structure and creating a new set of leaf images. The frozen node is then made read-only. Referring now to Figure 5, the root node 104 (also referred to herein as a base image) is the base distributed application. As previously discussed, the root node 104 is not modified by compute nodes for which the application is distributed.

[0043] The first level of the exemplary tree structure 500 comprises first branch node 604 and branch node 606, which represent two different application service groups. Compute nodes have read only access to the service group applications represented by branches 604 and 606. Branch node 604 has three leaf nodes 105a, 105b, 105c. The leaf nodes 105 may be for configuration files, for example. The compute nodes can boot off from the configuration files. The leaf nodes are also referred to herein as leaf images.

In one embodiment, the leaf nodes 105 are change volumes. In another embodiment, the leaf nodes 105 are branch modification logs.

[0044] The branch node 606 also has three leaf nodes 605d, 605e, 605f, which are analogous to the leaf nodes 605a-c associated with branch node 604. However, branch node 606 also has a service group 620, which reflects a change to the application. The service group 620 has leaf nodes 605g, 605h, and 605i.

[0045] Figure 6 is diagram illustrating components of a file system 600, in accordance with an embodiment of the present invention. The file system 600 may be used in a copy-on-write embodiment, such as the system 400 depicted in Figure 4. However, it will be understood that the system 400 of Figure 4 is not limited to file system 600. The filter 602 presents the view to the compute nodes of a regular file system. In one embodiment, the filter presents the view of a UNIX file system.

[0046] The file system 600 also has a table of locations 606, which indicates the locations of branch nodes. The file system has an XML document that indicates the locations. The XML document can be sent to compute nodes.

[0047] Also stored are base addresses 608, application addresses 610, and change addresses 612. The file system has a networking layer 614 for interfacing on the network (Fig. 4, 112).

[0048] Figure 7 is flowchart illustrating a process 700 of branching file system access, in accordance with an embodiment of the present invention. Steps of process 700 may stored as instructions on a computer readable medium and executed on a computer processor. Step 710 is creating a root file system comprising a root image of an application.

[0049] Step 720 is creating leaf images that are configured to record changes to the root image of the application. The leaf images comprise branch modification logs in one embodiment. In another embodiment, the leaf images comprise change volumes.

[0050] Step 730 is creating a directory configured to access instances of the application for respective compute nodes. A respective compute node's application instance comprises at least a portion of the root image and the respective compute node's leaf image.

[0051] Step 740 is providing the compute nodes with respective instances of the application. Step 740 may comprise a filter determining

which files or data blocks are needed for a given compute node's instance of the application.

[0052] Step 750 is modifying one of the leaf images in response to a corresponding compute node's access to its instance of the application. The modifying of the leaf image may include copying a file from the root image to the first leaf image and then modifying the file in the first leaf image. However, this step is not so limited. The modifying may also be performed by copying one or more data blocks from the root image to a leaf image, and modifying the data block in the leaf image. It will be appreciated that the file or data block being modified may already exist in the leaf image, in which case it does not need to be copied from the base image before modification.

[0053] The preferred embodiment of the present invention, branching store file system, is thus described. While the present invention has been described in particular embodiments, it should be appreciated that the present invention should not be construed as limited by such embodiments, but rather construed according to the below claims.

TITLE: "BRANCHING STORE FILE SYSTEM"
INVENTOR(S): Tung M. Nguyen, Richard Au, and Eric Pouyoul
USSN: n/a Attorney Docket #: PNTA-P002

1 / 7

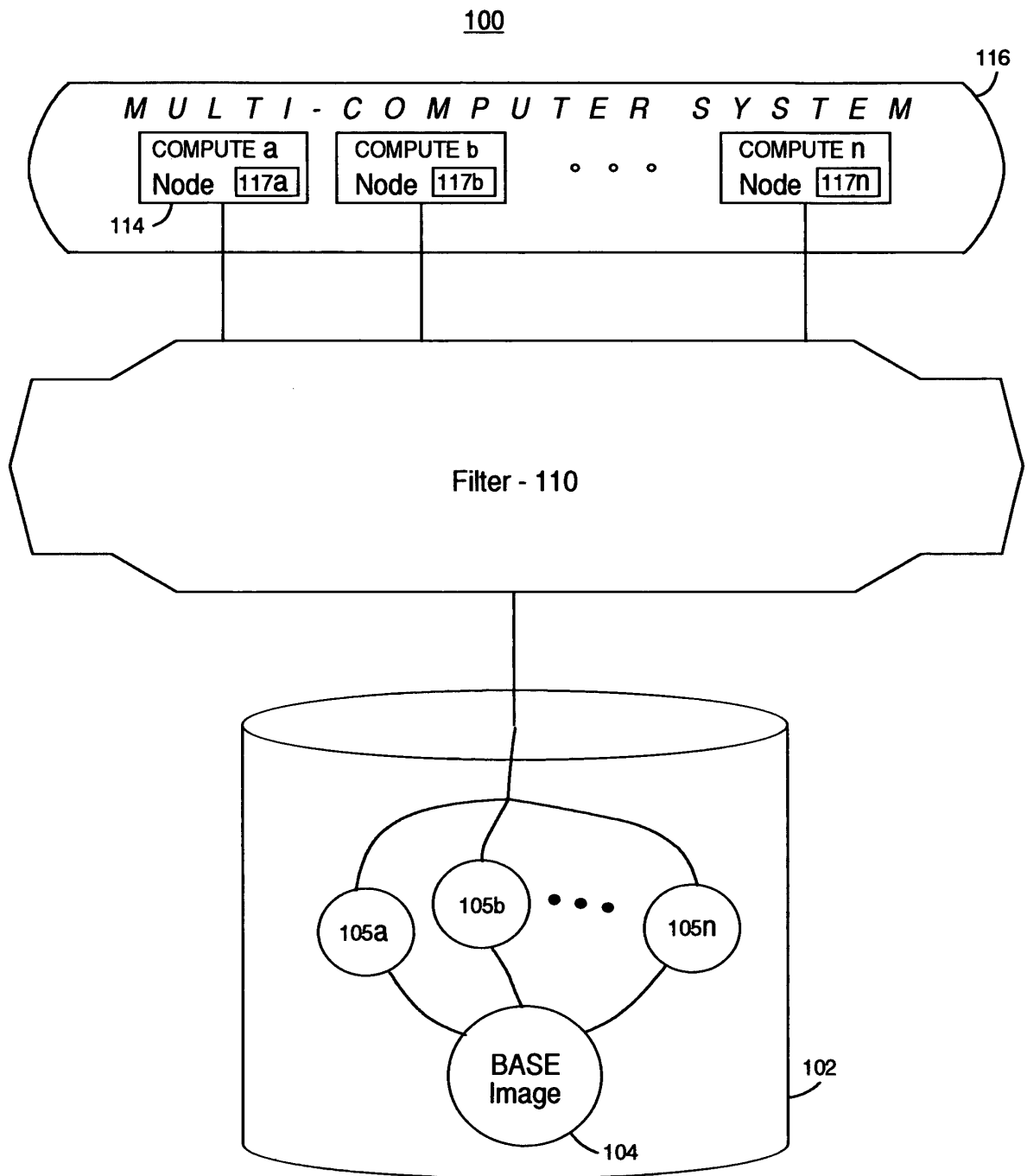


Figure 1

TITLE: "BRANCHING STORE FILE SYSTEM"
INVENTOR(S): Tung M. Nguyen, Richard Au, and Eric Pouyoul
USSN: n/a Attorney Docket #: PNTA-P002

217

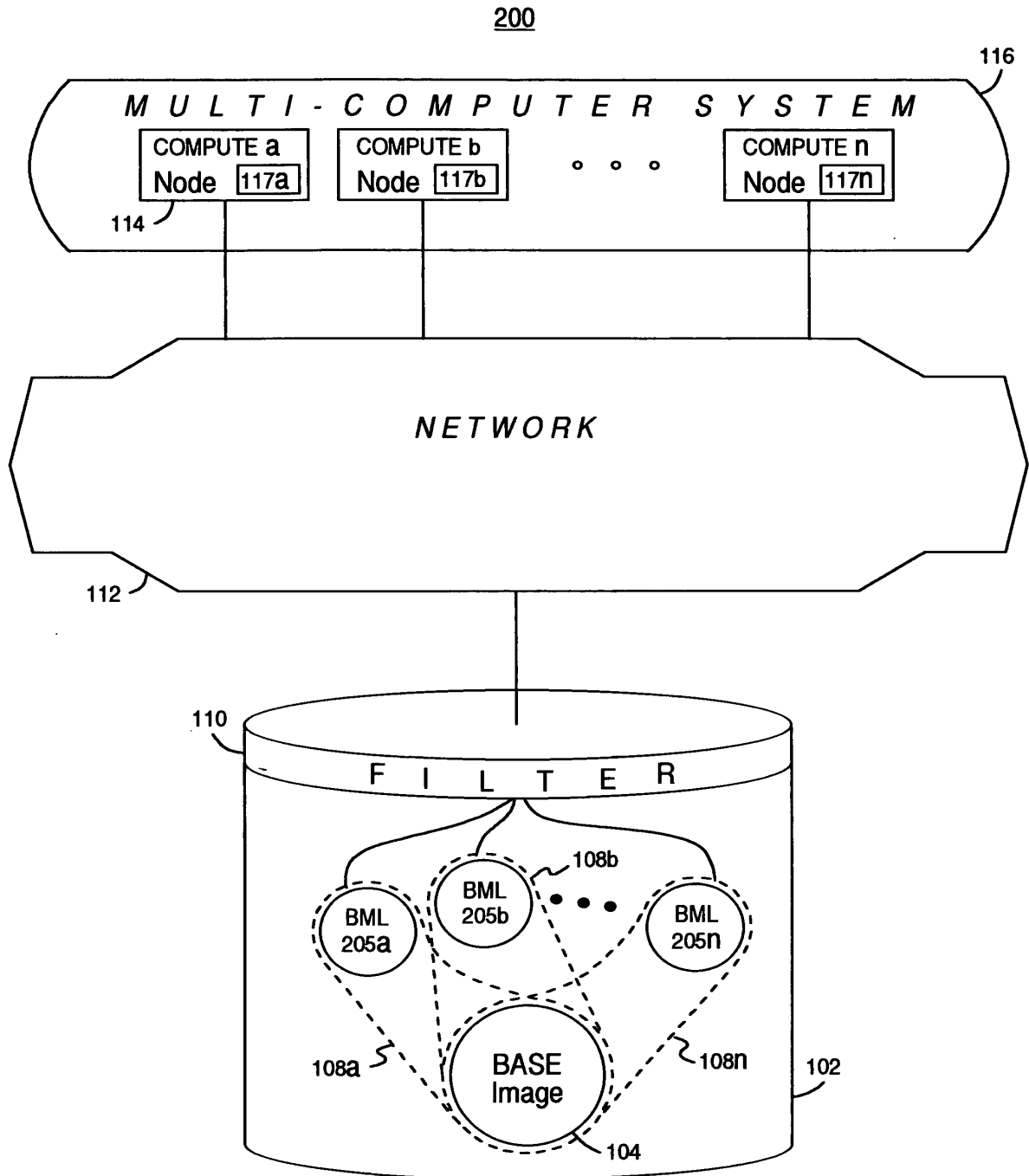


Figure 2

TITLE: "BRANCHING STORE FILE SYSTEM"
 INVENTOR(S): Tung M. Nguyen, Richard Au, and Eric Pouyoul
 USSN: n/a Attorney Docket #: PNTA-P002

3 / 7

150

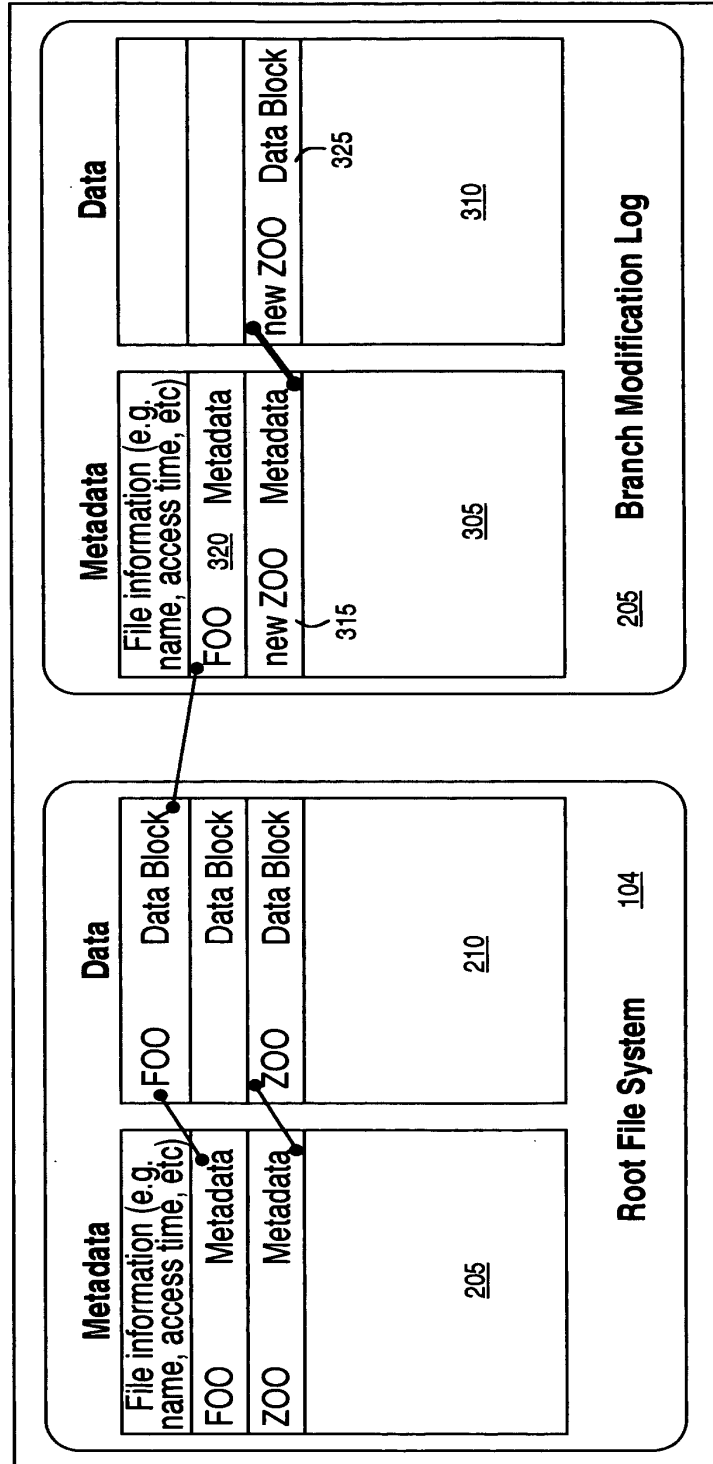


Figure 3

TITLE: "BRANCHING STORE FILE SYSTEM"
INVENTOR(S): Tung M. Nguyen, Richard Au, and Eric Pouyoul
USSN: n/a Attorney Docket #: PNTA-P002

4 / 7

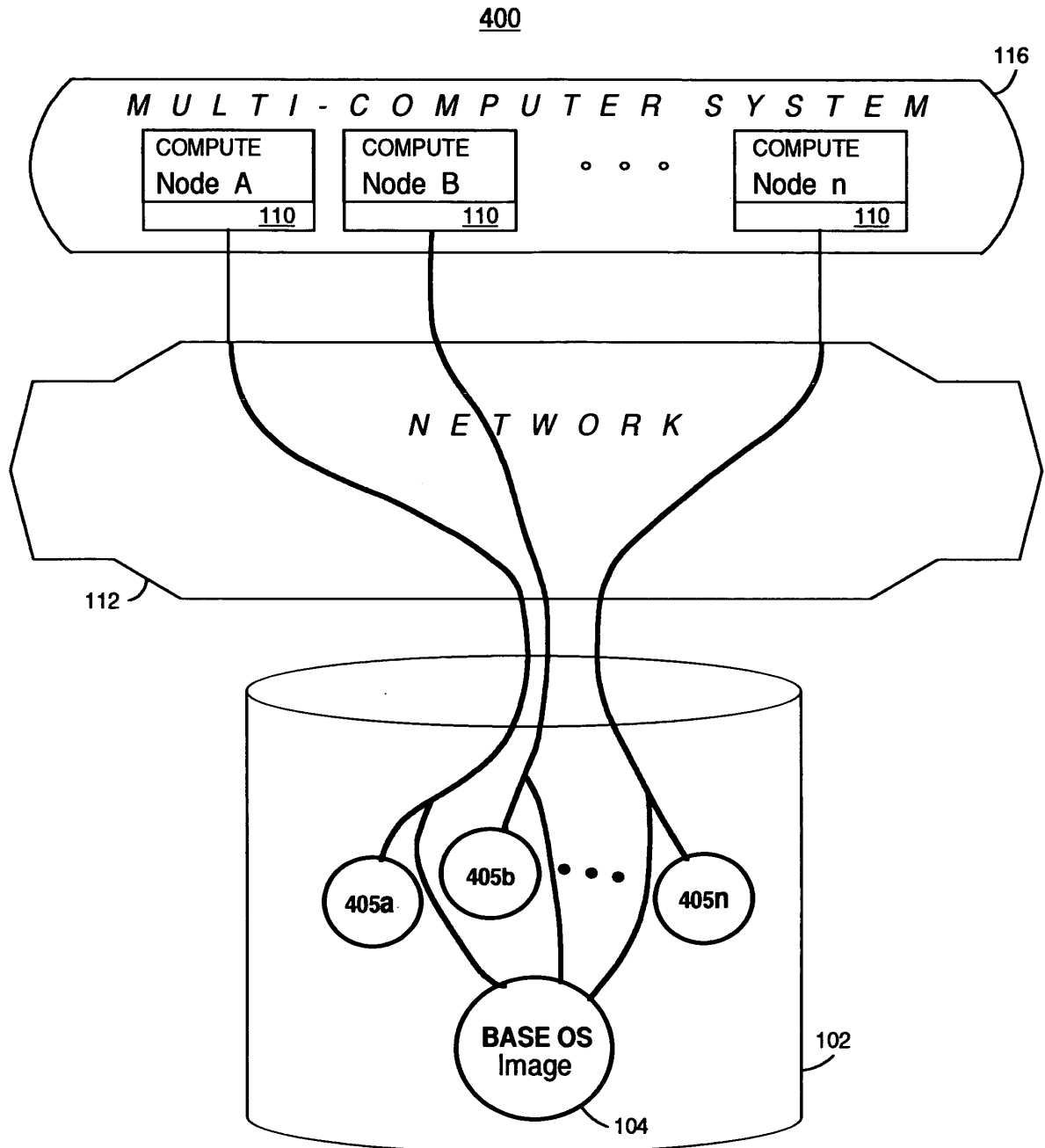


Figure 4

TITLE: "BRANCHING STORE FILE SYSTEM"
 INVENTOR(S): Tung M. Nguyen, Richard Au, and Eric Pouyoul
 USSN: n/a Attorney Docket #: PNTA-P002

5 / 7

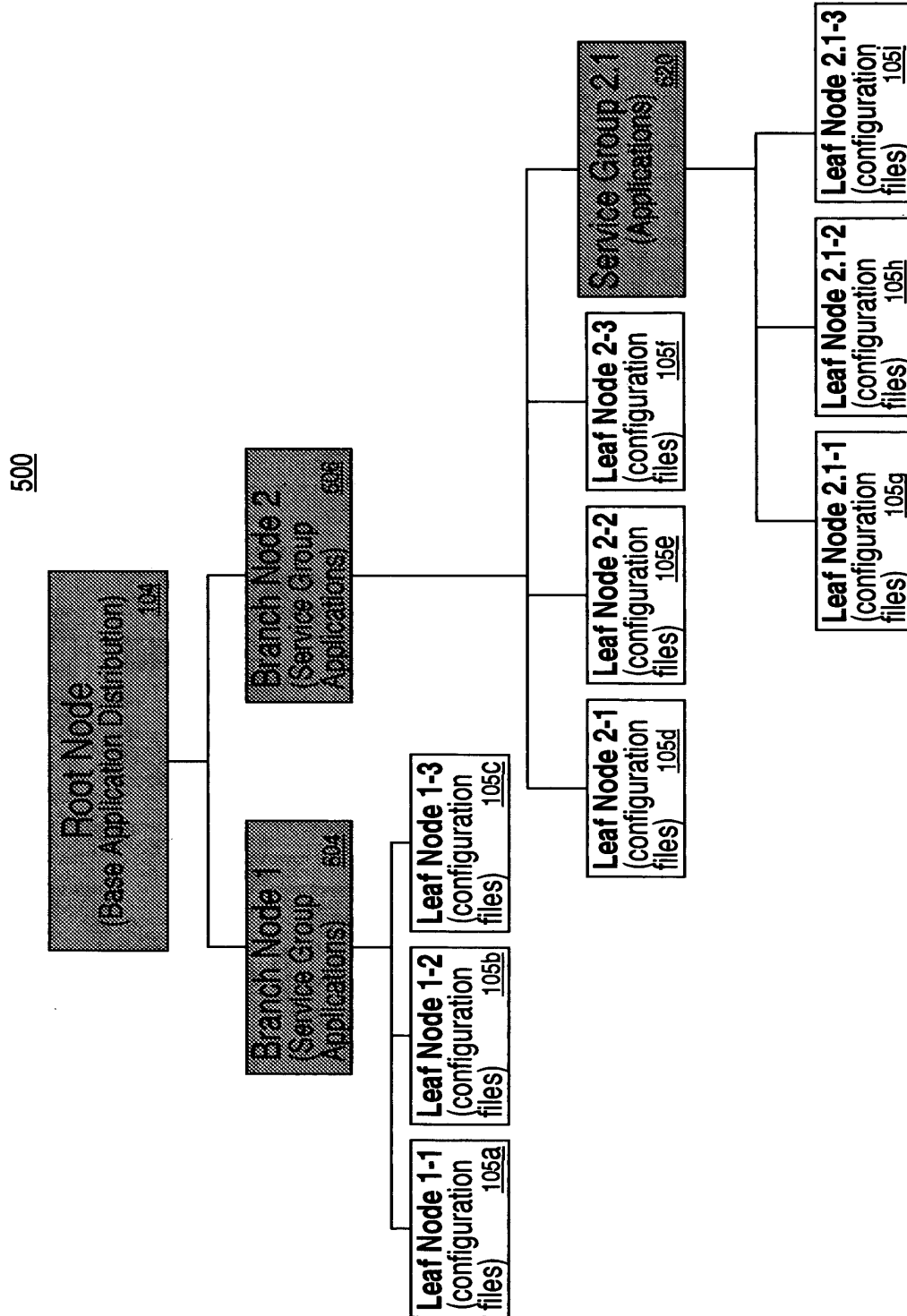


Figure 5

TITLE: "BRANCHING STORE FILE SYSTEM"
INVENTOR(S): Tung M. Nguyen, Richard Au, and Eric Pouyoul
USSN: n/a Attorney Docket #: PNTA-P002

6 / 7

600

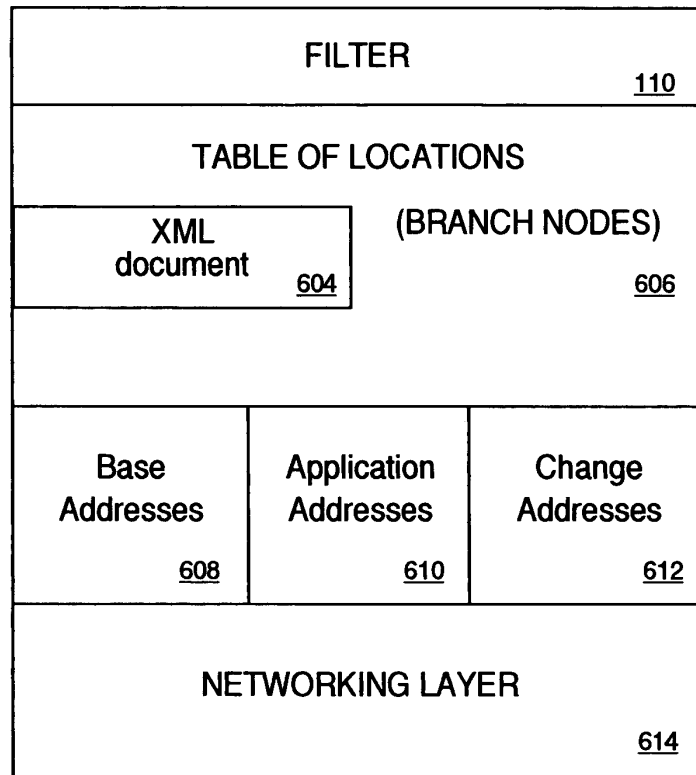


Figure 6

TITLE: "BRANCHING STORE FILE SYSTEM"
INVENTOR(S): Tung M. Nguyen, Richard Au, and Eric Pouyoul
USSN: n/a Attorney Docket #: PNTA-P002

7 / 7

700

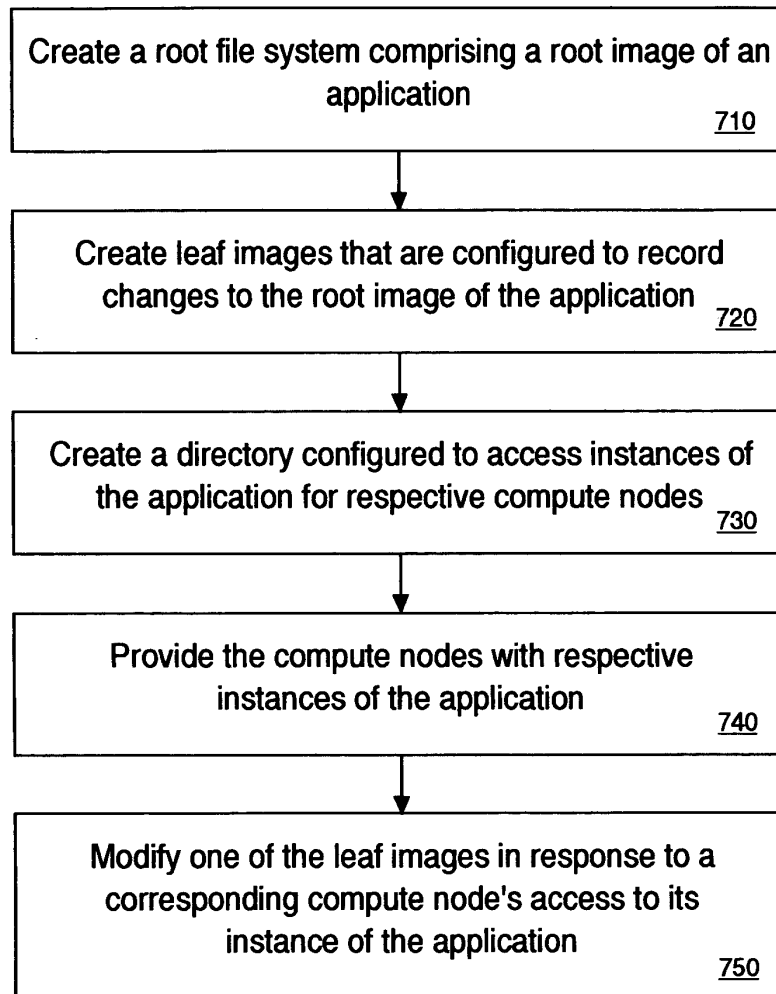


Figure 7

CLAIMS

What is Claimed, is:

1. A branching store file system, comprising:
 - 5 a root file system comprising a base image of a distributed application, wherein said base image is read-only accessible by compute nodes;
leaf images comprising modified portions of said distributed application for respective ones of said compute nodes, wherein an instance of said distributed application for a first of said compute nodes is based on said
10 base image and at least one of said leaf images; and
a filter that controls access of instances of said distributed application for said compute nodes.
2. The system of Claim 1, wherein said leaf images comprise branch
15 modification logs for said respective ones of said compute nodes.
3. The system of Claim 1, wherein said leaf images comprise change volumes for said respective ones of said compute nodes.
- 20 4. The system of Claim 1, wherein said filter is implemented at said compute nodes.

PNTA-P002 JPH/RMP

CONFIDENTIAL

5. The system of Claim 1, wherein said filter is implemented at said root file system.

6. The system of Claim 1, further comprising intermediate images
5 between said base image and said leaf images, wherein said intermediate images comprise application service groups.

7. The system of Claim 6, wherein access of said compute nodes to said intermediate images is read-only.

10

8. The system of Claim 1, wherein said distributed application comprises an operating system.

9. The system of Claim 1, wherein said base image is concurrently
15 accessible to a plurality of said compute nodes.

10. The system of Claim 1, wherein said leaf images are configured to be modified on a block-by-block basis.

20 11. The system of Claim 1, wherein said leaf images are configured to be modified on a file-by-file basis.

12. A branching store file system, comprising:

a root file system having a root metadata structure and a root data structure, said entries in said root metadata structure pointing to associated unmodified data in said root data structure; and

5 a branch modification log for respective ones of compute nodes, comprising:

a branch data structure comprising modified data reflecting changes made to an instance of said root data structure by said branch modification log's respective compute node; and

10 a branch metadata structure comprising unmodified and modified entries, said unmodified entries pointing to associated portions of said unmodified data in said root data structure, and said modified entries pointing to associated portions of said modified data in said branch data structure, wherein an instance of a distributed
15 application for a first of said compute nodes is defined by said branch metadata structure.

13. The system of Claim 12, further comprising a filter that directs accesses of said distributed application from compute nodes to a respective
20 compute node's instantiation of the distributed application.

14. The system of Claim 12, wherein said distributed application comprises an operating system.

PNTA-P002 JPH/RMP

CONFIDENTIAL

15. The system of Claim 13, wherein said filter is configured to traverse inodes to said root data structure for accesses to said un-modified data.

5 16. The system of Claim 13, wherein said filter is configured to traverse inodes to said branch data structure for accesses to said modified data.

17. The system of Claim 12, wherein access to said root data structure by said compute nodes is read-only and access to said branch data structure is
10 read-write-modify by said branch modification log's respective compute node.

18. The system of Claim 12, wherein ones of said modified entries have a modification attribute attached.

15 19. The system of Claim 12, wherein said branch modification log is configured to receive updates to said modified data on a block-by-block basis.

20. A branching store file system, comprising:
a root file system comprising a base instance of a distributed
20 application;
change volumes for respective ones of compute nodes and comprising
modified versions of files of said distributed application; and

a filter program code that directs accesses of said compute nodes' respective instances of said distributed application, wherein said compute nodes' respective instances of said distributed application are based on said base instance and respective ones of said change volumes.

5

21. The system of Claim 20, further comprising a table of locations indicating locations of said change volumes.

22. The system of Claim 20, wherein said base instance is concurrently
10 accessible to a plurality of said compute nodes.

23. The system of Claim 20, wherein said compute nodes have read-only access to files in said base instance.

15 24. The system of Claim 21, wherein said distributed application comprises an operating system.

25. The system of Claim 20, wherein said filter performs a copy-on-write to one of the change volumes when a compute node writes to said compute
20 node's instantiation of said distributed application.

26. The system of Claim 20, wherein said change volumes are modified on a file-by-file basis.

PNTA-P002 JPH/RMP

CONFIDENTIAL

27. A computer-implemented method of distributed application deployment, said method comprising:

- creating a root file system comprising a root image of an application;
- 5 creating a plurality of leaf images that are configured to record changes to said root image of the application for respective compute nodes;
- creating a directory configured to access instances of said application for said respective compute nodes, wherein a respective one of the compute node's instance of said application comprises at least a portion of said root
- 10 image and the respective compute node's leaf image; and
- providing said compute nodes with said respective instances of said application.

28. The method of Claim 27, wherein said leaf images comprise branch
15 modification logs.

29. The method of Claim 28, wherein said providing said compute nodes with respective instances of said application comprises traversing inodes as indicated by said branch modification logs.

20

30. The method of Claim 27, wherein said leaf images comprise change volumes.

PNTA-P002 JPH/RMP

CONFIDENTIAL

31. The method of Claim 30, wherein said providing said compute nodes with respective instances of said application comprises consulting a table of locations.

5 32. The method of Claim 27, wherein said application comprises an operating system.

33. The method of Claim 27, wherein access to said root image by said compute nodes is read-only.

10

34. The method of Claim 27, further comprising freezing a version of said distributed application.

35. The method of Claim 34, further comprising creating new leaf nodes
15 that are children of said frozen version of said distributed application.

36. The method of Claim 27, further comprising modifying a first of said leaf images in response to a corresponding compute node's access to its instance of the application.

20

37. The method of Claim 36, wherein said modifying comprises a copying a file from said root image to said first leaf image and then modifying said file in said first leaf image.

PNTA-P002 JPH/RMP

CONFIDENTIAL

38. The method of Claim 36, wherein said modifying comprises a copying a data block from said root image to said first leaf image and then modifying said data block data block in said first leaf image.

EXHIBIT A-4

PNTA-P006

Patent

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant : Kulkarni, et al.

Confirmation : 2625

Serial No. : 11/395,816

Examiner : Khatri, Anil

Filing Date : 03/30/2006

Art Unit : 2191

Attorney Docket No. : PNTA-P006

For: BLOCK-LEVEL I/O SUBSYSTEM FOR DISTRIBUTED APPLICATION
ENVIRONMENT MANAGEMENT

AMENDMENT AND RESPONSE TO OFFICE ACTION

Commissioner for Patents & Trademarks
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Sir:

In response to the Office Action mailed July 8, 2009, please consider the following amendments and remarks in the above captioned patent application.

Amendments to the Specification begin on Page 2 of this paper.

Amendments to the Claims are reflected in the listing of Claims which begins on page 4 of this paper, followed by Applicants' remarks.

PNTA-P006
Serial No. : 11/395,816

Page 1 of 18

Examiner: Khatri, Anil
Group Art Unit: 2191

Amendments to the Specification

Please replace Paragraph 0009 with the following:

[0009] An embodiment of the present invention is directed to a system for distributing an application environment to a compute node (e.g., a server or a thin-client workstation). The system includes a first storage unit for storing blocks of a root image (e.g., an operating system) of the compute node and a second storage unit for storing a leaf image comprising new data blocks and changes to the blocks of the root image. The system further includes a union block device for interfacing between the compute node and the first and second storage units to distribute the application environment to the compute node. The union block device creates the application environment by merging the blocks of the root image stored on the first storage unit with the leaf image stored on the second storage unit.

Please add the following paragraph between Paragraphs 0009 and 0010:

Another embodiment of the present invention is directed to a method for distributing an application environment to a compute node. The method includes storing blocks of a root image of the compute node on a first storage unit and storing a leaf image comprising new data blocks and changes to the blocks of the root image on a second storage unit. The method further includes merging the blocks of the root image stored on the first storage unit with the blocks of the leaf image stored on the second storage unit to create the application environment. The application environment is created by merging

the blocks of the root image stored on the first storage unit with the blocks fo the leaf image stored on the second storage unit. Lastly, the application environment is delivered to the computer node.

Amendments to the Claims

1. (Currently amended) A system for distributing an application environment ~~to a compute node~~ comprising:

a compute node comprising a computer system;

a first storage unit for storing blocks of a root image of the compute node, wherein the first storage unit comprises a first non-volatile memory, wherein the root image comprises a computer program, wherein the blocks comprise sections of data, and wherein a file of the root image comprises at least one block;

a second storage unit for storing a leaf image, the leaf image comprising new data blocks and changes to the blocks of the root image, wherein the second storage unit comprises a second non-volatile memory; and

a union block device for interfacing between the compute node and the first and second storage units to distribute the application environment to the compute node, wherein the union block device comprises a driver, wherein the union block device creates the application environment by merging the blocks of the root image stored on the first storage unit with the blocks of the leaf image stored on the second storage unit.

2. (Original) The system as recited in Claim 1 wherein the compute node comprises a server.

3. (Original) The system as recited in Claim 1 wherein the compute node comprises a thin-client workstation.

4. (Original) The system as recited in Claim 1 wherein the root image comprises an operating system.

5. (Original) The system as recited in Claim 1 wherein the root image is concurrently accessible to a plurality of compute nodes.

6. (Original) The system as recited in Claim 1 wherein the first storage unit is remotely located from the compute node.

7. (Original) The system as recited in Claim 1 wherein the second storage unit is remotely located from the compute node.

8. (Original) The system as recited in Claim 1 wherein the second storage unit contains a block modification log for the compute node.

9. (Original) The system as recited in Claim 1 wherein the first storage unit is contained within a first partition on a hard disk and the second storage unit is contained within a second partition on the hard disk.

10. (Original) The system as recited in Claim 1 wherein the union block device comprises a low-level driver for interfacing between the first and second storage units and the file system of the compute node.

11. (Original) The system as recited in Claim 1 wherein the union block device, upon receiving a write request from the compute node for a sector X, creates an appropriate persistent mapping for sector X.

12. (Original) The system as recited in Claim 11 wherein the union block device writes sector X on the second storage unit.

13. (Original) The system as recited in Claim 1 wherein the system operates in a high performance computing cluster.

14. (Original) The system as recited in Claim 1 wherein the system operates in a grid computing cluster.

15. (Original) The system as recited in Claim 1 wherein the first storage unit is read only.

16. (Original) The system as recited in Claim 1 further comprising an intermediate image between the root image and the leaf image, wherein the intermediate image comprises an application environment service group.

17. (Currently amended) A method for distributing an application environment ~~to a compute node~~ comprising:

storing blocks of a root image of a[[the]] compute node on a first storage unit, wherein the compute node comprises a computer system, and wherein the first storage unit comprises a first non-volatile memory, wherein the root image comprises a computer program, wherein the blocks comprise sections of data, and wherein a file of the root image comprises at least one block;

storing a leaf image comprising new data blocks and changes to the blocks of the root image on a second storage unit, wherein the second storage unit comprises a second non-volatile memory;

merging the blocks of the root image stored on the first storage unit with the blocks of the leaf image stored on the second storage unit to create the application environment; and

delivering the application environment to the compute node.

18. (Original) The method as recited in Claim 17 further comprising:
modifying the leaf image in response to the compute node's access to the application environment.

19. (Original) The method as recited in Claim 18 wherein the modifying comprises:

upon receiving a write request from the compute node for a sector X, creating an appropriate persistent mapping for sector X; and

writing sector X on the second storage unit.

20. (Original) The method as recited in Claim 17 further comprising:
reconciling the root image and the leaf image to form a new root image.

21. (Original) The method as recited in Claim 17 wherein the compute node
comprises a server.

22. (Original) The method as recited in Claim 17 wherein the compute node
comprises a thin-client workstation.

23. (Original) The method as recited in Claim 17 wherein the root image
comprises an operating system.

24. (Original) The method as recited in Claim 17 wherein the root image is
concurrently accessible to a plurality of compute nodes.

25. (Original) The method as recited in Claim 17 wherein the first storage unit
is remotely located from the compute node.

26. (Original) The method as recited in Claim 17 wherein the second storage
unit is remotely located from the compute node.

27. (Original) The method as recited in Claim 17 wherein the second storage
unit contains a block modification log for the compute node.

28. (Original) The method as recited in Claim 17 wherein the first storage unit
is contained within a first partition on a hard disk and the second storage unit is contained
within a second partition on the hard disk.

29. (Original) The method as recited in Claim 17 wherein merging occurs at an operational level between the first and second storage units and file system of the compute node.

30. (Original) The method as recited in Claim 17 wherein the method operates in a high performance computing cluster.

31. (Original) The method as recited in Claim 17 wherein the system operates in a grid computing cluster.

32. (Original) The method as recited in Claim 17 wherein the first storage unit is read only.

33. (Original) The method as recited in Claim 17 further comprising:
creating an intermediate image on a third storage unit between the root image and the leaf image, wherein the intermediate image comprises an application environment service group.

REMARKS

Claims 1-33 are pending. Claims 1 and 17 are amended. No new matter has been added as a result of these amendments.

Specification Objections

The Summary of the invention is objected to because it is allegedly identical and verbatim to the abstract. Applicants have therefore amended paragraph 0009 and added a new paragraph between paragraph 0009 and 0010 that describes another embodiment of the invention. Further, Applicants respectfully point out that paragraph 0010 provides additional information related to the advantages of the invention over the prior art by explaining the nature of the invention (e.g., rather than copying the contents of the root image to create a new boot image, a new UBD is registered, which occurs very quickly).

Claim Rejections – 35 U.S.C. §101

Claims 1-33 are rejected under 35 U.S.C. 101 for allegedly disclosing a claimed invention that is an abstract idea. The rejection contends that Claims 1-33 are non-statutory because they do not disclose how a system will be able to process and carry out its intended results incorporating a processor, memory and medium. The rejection therefore, contends that Claims 1-33 recite "storage unit," "node," and "union block device" without showing how and where they are performed. Applicants have amended Claims 1 and 17 to further define the claim terms and their functionality.

Claim Rejections – 35 U.S.C. §112

Claims 1-16 are rejected under 35 U.S.C 112, second paragraph, for allegedly providing insufficient antecedent basis for the limitation "union block device." The rejection alleges this term is unclear and vague. Applicants have amended Claim 1 to further define the union block device as a "low level driver." Support for this amendment is found in Paragraph 0020 of Applicants' Specification. Paragraph 0020 also further describes the union block device and a sample embodiment (i.e., low level driver). As described in Applicants' Specification, and as known in the art, low-level drivers may be used in data acquisition, such as with the individual blocks making up files (data) within the root images and leaf images, as claimed in Claim 1.

Claim Rejections – 35 U.S.C. §103

Claims 1-33 are rejected under 35 U.S.C. 103(a) as being allegedly unpatentable over Paven (US 6,502,238), hereinafter "Paven," in view of Davidson (US 7,475,274), hereinafter "Davidson." Applicants respectfully assert that Claims 1-33 are patentable over the combination of Paven and Davidson in view of the following.

Applicants respectfully point out that the Examiner has the burden of establishing a prima case of obviousness. To establish a prima facie case of obviousness, three basic criteria must be met. First, there must be some suggestion or motivation, either in the references themselves or in the knowledge generally available to one of ordinary skill in the art, to modify the reference or to combine reference teachings. Second, there must be a reasonable expectation of success. Finally, the prior art reference (or references when

combined) must teach or suggest all the claim features. The teaching or suggestion to make the claimed combination and the reasonable expectation of success must both be found in the prior art, and not based on Applicant's disclosure. In *re* Vaeck, 947 F.2d 488, 20 USPQ2d 1438 (Fed. Cir. 1991). See MPEP 2100-126. Specifically, "all words in a claim must be considered when judging the patentability of that claim against the prior art." *In re Wilson*, 424 F.2d., 1382 (CCPA 1970).

Moreover, in response to the recent U.S. Supreme Court decision in *KSR Int'l Co. v. Teleflex, Inc.* (U.S. 2007), new guidelines were set forth for examining obviousness under 35 U.S.C. 103. The U.S. Supreme Court reaffirmed the *Graham* factors and, while not totally rejecting the "teachings, suggestion, or motivation" test, the Court appears to now require higher scrutiny on the part of the U.S. Patent & Trademark Office. In accordance with the recently submitted guidelines, it is "now necessary to identify the reason" why a person of ordinary skill in the art would have combined the elements of cited references, or at least describe the pertinence of the elements set forth in the cited disclosure, in the manner presently claimed.

Applicants contend that the combination of Paven and Davidson fail to teach or suggest all the claimed elements of Claims 1-33 in view of the following rationale.

Claim 1:

Amended Claim 1 recites in part:

a compute node comprising a computer system;
a first storage unit for storing blocks of a root image of the compute node, wherein the first storage unit comprises a first non-volatile memory;
a second storage unit for storing a leaf image, the leaf image comprising new data blocks and changes to the blocks of the root image, wherein the second storage unit comprises a second non-volatile memory, wherein the root image comprises a computer program, and wherein the blocks comprise sections of data; and
a union block device for interfacing between the compute node and the first and second storage units to distribute the application environment to the compute node, wherein the union block device comprises a driver, wherein the union block device creates the application environment by merging the blocks of the root image stored on the first storage unit with the blocks of the leaf image stored on the second storage unit (emphasis added).

Applicants respectfully assert that Pavan does not teach or suggest "a first storage unit for storing blocks of a root image...; a second storage unit for storing a leaf image... comprising new data blocks and changes to the blocks of the root image...; and a union block device for interfacing between the compute node and the first and second storage units to distribute the application environment to the compute node, ...wherein the union block device creates the application environment by merging the blocks of the root image stored on the first storage unit with the blocks of the leaf image stored on the second storage unit," as claimed in Claim 1.

As understood by Applicants, Pavan is directed towards a distributed block-based programming model for control applications, wherein a user program specifies interconnections between a plurality of program blocks distributed across a plurality of

processing nodes (Abstract). Pavan discloses using multimedia information with Multiple-Input Multiple-Output (MIMO) control systems (Column 1, line 56 to Column 2, line 10). Pavan discloses a method wherein a user program is received. This user program specifies interconnections between a plurality of program blocks distributed across a plurality of processing nodes, wherein the user program is translated into a system-level program having a program fragment for each of the processing nodes, wherein each program fragment comprises the program blocks located on the processing node and one or more system-level blocks for establishing connections between the processing node and a different processing node across a network (Column 2, lines 19-27; and Figure 4).

Applicants respectfully assert that to the extent Pavan discloses program blocks, root nodes, and the creation of program fragments with program blocks and children, Pavan does not teach or suggest a first storage unit for storing blocks of a root image, a second storage unit for storing a leaf image, nor a union block device which creates an application environment by merging the blocks of the root image stored on the first storage unit with the blocks of the leaf image stored on the second storage unit, as claimed in Claim 1.

Applicants respectfully assert that the blocks of a root image, as claimed in amended Claim 1 are not the program blocks as taught by Pavan. Applicants have amended Claim 1 to clarify that the blocks of a root image comprise sections of data, wherein a file of the root image comprises at least one block, as claimed in Claim 1.

Support for Applicants' amendment is found in Applicants' Specification, Paragraphs 0018-0020. As claimed in Amended Claim 1, these blocks make up a root image which comprises a computer program. Support for this amendment may be found in Paragraph 0017. This is completely different from the program blocks as taught by Pavan. As understood by Applicants, Pavan teaches that a program block is part of a block-based programming model, wherein a block is a high level program abstraction (Column 3, line 64 to Column 4, line 2). Further, these program blocks may be functions, objects implemented in an object-oriented programming language, or sensors, cameras and functions performed by physical devices (Column 3, line 64 to Column 4, line 18). The functional blocks of Pavan are not blocks comprising sections of data, as claimed in Claim 1.

Applicants respectfully assert that Pavan does not teach or suggest a root image or a leaf image, as claimed in Claim 1. A root image, as claimed in Claim 1, is not a root node, as taught by Pavan. The root block, node, or fragment, as taught by Pavan, controls the system operations and executes on the control node and controls the execution of the system-level program for a MIMO control system (Column 9, lines 64-67), as compared to a root image comprising a computer program, as claimed in amended Claim 1. Support for this amendment may be found in Applicants' Specification, Paragraph 0018.

Further, Applicants respectfully assert that a second storage unit for storing a leaf image, as claimed in Claim 1, is not the same thing as the process Pavan describes for the "Grow-Fragment" block 508. Pavan teaches methods for making program fragments for

each of the processing nodes that take into account all the components or program blocks that make up each processing node as well as all interconnecting lines between the processing nodes. Pavan describes a "child" as a program block that is connected to the current block that is being processed by the "Grow-Fragment" block. Where Pavan teaches a method for implementing a programming method for interconnecting program blocks or functions that may reside on several compute nodes, Applicants claim a second storage unit for storing a leaf image, the leaf image comprising new data blocks and changes to the blocks of the root image, as claimed in Claim 1. Applicants respectfully assert that Davidson does not remedy these deficiencies.

Lastly, the rejection appears to agree that Pavan does not teach or suggest "a union block device for interfacing between the compute node and the first and second storage units to distribute the application environment to the compute node, ...wherein the union block device creates the application environment by merging the blocks of the root image stored on the first storage unit with the blocks of the leaf image stored on the second storage unit," as claimed in Claim 1. Applicants respectfully assert that Davidson does not remedy this deficiency.

The rejection contends that Davidson describes a union block device. Applicants respectfully assert that the rejection is reading the management node 105 of Davidson, which manages or assists an administrator of an HPC system, on Applicants' claimed union block device that creates an application environment by merging the blocks of the

root image stored on the first storage unit with the blocks of the leaf image stored on the second storage unit, as claimed in Claim 1. Applicants respectfully disagree.

As understood by Applicants, Davidson discloses a method and apparatus for fault tolerance and recovery of high-performance computing (HPC) systems. Davidson teaches a method for monitoring currently running nodes in the HPC system, and when a fault occurs, shutting down the defaulting node and booting the host at a free node in the HPC system. Davidson further discloses the management node 105, referred to by the rejection, comprising at least one blade substantially dedicated to managing or assisting an administrator. Davidson further discusses the methods of linking the blades comprising the management node 105 with the rest of the HPC nodes through wires, buses, LANs, MANs, WANS, an Internet connection, or a combination of two or more such links. However, Applicants do not find any teaching by Davidson wherein the management node provides an application environment to a compute node by merging the blocks of the root image stored on the first storage unit with the blocks of the leaf image stored on the second storage unit, as claimed in Claim 1

Applicants respectfully assert that the combination of Pavan and Davidson does not teach or suggest a first storage unit for storing blocks of a root image of the compute node; and a second storage unit for storing a leaf image, the leaf image comprising new data blocks and changes to the blocks of the root image, nor a union block device for interfacing between the compute node and the first and second storage units to distribute the application environment to a compute node, wherein the union block device creates

the application environment by merging the blocks of the root image stored on the first stored unit with the blocks of the leaf image stored on the second storage unit, as claimed in Claim 1.

Therefore, Applicants respectfully assert that embodiments as recited by Claim 1 are not rendered obvious by the combination of Pavan and Davidson. Accordingly, Applicants respectfully assert that dependent Claims 2-16 are patentable by virtue of their dependency on an allowable base claim, as well as for their additional recited patentable features.

Claim 17:

Independent Claim 17 recites features similar to that of independent Claim 1 and is therefore patentable for at least the same or similar reasons as recited above. Accordingly, Applicants respectfully assert that dependent Claims 18-33 are patentable by virtue of their dependency on an allowable base claim, as well as for their additional recited patentable features.

For the above reasons, Applicants request reconsideration and withdrawal of the rejections under 35 U.S.C. §103.

CONCLUSION

In light of the above listed remarks, reconsideration of rejected Claims is requested. Based on the arguments presented above, it is respectfully submitted that Claims 1-33 overcome the rejections of record and, therefore, allowance of Claims 1-33 is earnestly solicited.

Please charge any additional fees that may be required to maintain pendency of the present application, or apply any credits to our PTO deposit account number: 50-4160.

Respectfully submitted,

MURABITO, HAO & BARNES LLP

Dated: October 8, 2009

/James P. Hao/
James P. Hao
Registration No. 36,398

MURABITO, HAO & BARNES LLP
Two North Market Street
Third Floor
San Jose, California 95113

(408) 938-9060 Voice
(408) 938-9069 Facsimile

I hereby certify that this
correspondence is being electronically
transmitted to the USPTO via EFS on
the date shown below:

October 8, 2009

/Shannon Warren/
Shannon Warren

PNTA-P006
Serial No. : 11/395,816

Page 18 of 18

Examiner: Khatri, Anil
Group Art Unit: 2191

EXHIBIT A-5



US007949785B2

(12) **United States Patent**
Alkhatib et al.

(10) **Patent No.:** **US 7,949,785 B2**
(45) **Date of Patent:** **May 24, 2011**

(54) **SECURE VIRTUAL COMMUNITY NETWORK SYSTEM**

(75) Inventors: **Hasan S. Alkhatib**, Saratoga, CA (US);
Fouad A. Tobagi, Los Altos, CA (US);
Farid F. Elwailly, San Jose, CA (US)

(73) Assignee: **Inpro Network Facility, LLC**,
Wilmington, DE (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 685 days.

5,764,906	A	6/1998	Edelstein
5,777,989	A	7/1998	McGarvey
5,781,550	A	7/1998	Templin
5,790,548	A	8/1998	Sistanizadeh
5,793,763	A	8/1998	Mayes
5,805,818	A	9/1998	Perlman
5,805,820	A	9/1998	Bellovin
5,815,664	A	9/1998	Asano
5,826,014	A	10/1998	Coley
5,856,974	A	1/1999	Gervais
5,864,666	A	1/1999	Shrader
5,867,667	A	2/1999	Butman
5,884,038	A	3/1999	Kapoor

(Continued)

FOREIGN PATENT DOCUMENTS

EP 0 817 444 A2 1/1998

OTHER PUBLICATIONS

Venters, Demystifying Protocols: A comparison of Protocols Suitable for IP Telephony, Sonus Networks, pp. 1-11, 2000.

(Continued)

(21) Appl. No.: **10/403,818**

(22) Filed: **Mar. 31, 2003**

(65) **Prior Publication Data**

US 2004/0249911 A1 Dec. 9, 2004

(51) **Int. Cl.**
G06F 15/16 (2006.01)

(52) **U.S. Cl.** **709/245**; 726/15

(58) **Field of Classification Search** 709/245;
726/15

See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,159,592	A	10/1992	Perkins
5,361,256	A	11/1994	Doeringer
5,563,878	A	10/1996	Blakeley
5,623,605	A	4/1997	Keshav
5,701,427	A	12/1997	Lathrop
5,717,686	A	2/1998	Schiavoni
5,717,687	A	2/1998	Minot
5,734,651	A	3/1998	Blakeley
5,751,961	A	5/1998	Smyk
5,754,938	A *	5/1998	Herz et al. 725/116

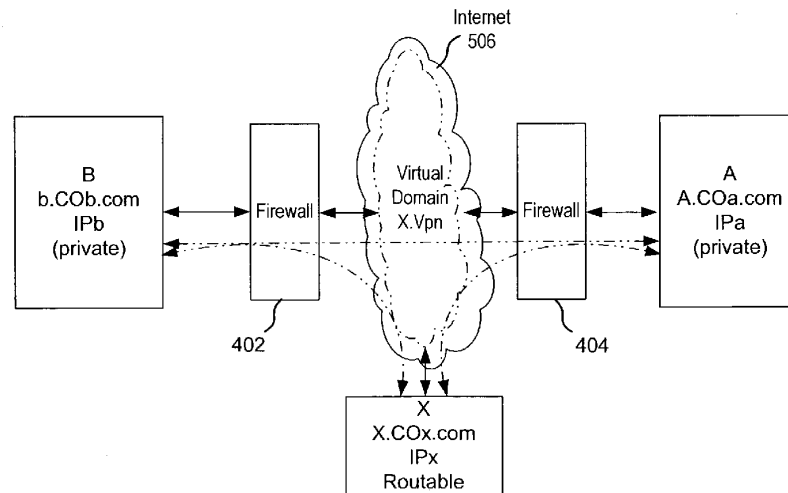
Primary Examiner — Douglas B Blair

(74) *Attorney, Agent, or Firm* — McAndrews, Held & Malloy, Ltd.

(57) **ABSTRACT**

A private virtual dynamic network is provided for computing devices coupled to public networks or private networks. This enables computing devices anywhere in the world to join into private enterprise intranets and communicate with each other. In one embodiment, the present invention provides a separate private virtual address realm, seen to each user as a private network, while seamlessly crossing public and private network boundaries. One implementation of the present invention uses an agent to enable an entity to participate in the network without requiring the member to add new hardware or software.

90 Claims, 21 Drawing Sheets



US 7,949,785 B2

Page 2

U.S. PATENT DOCUMENTS

5,884,246	A	3/1999	Boucher	6,993,012	B2	1/2006	Liu	
5,889,953	A	3/1999	Thebaut	6,993,595	B1	1/2006	Luptowski	
5,897,662	A	4/1999	Corrigan	6,996,628	B2	2/2006	Keane	
5,898,830	A	4/1999	Wesinger, Jr.	6,996,631	B1	2/2006	Aiken	
5,913,210	A	6/1999	Call	7,003,481	B2	2/2006	Banka	
5,937,162	A	8/1999	Funk	7,010,702	B1 *	3/2006	Bots et al.	726/13
5,937,163	A	8/1999	Lee	7,028,333	B2	4/2006	Tuomenoksa et al.	
6,003,084	A	12/1999	Green	7,028,334	B2	4/2006	Tuomenoksa et al.	
6,006,272	A	12/1999	Aravamudan	7,032,242	B1	4/2006	Grabelsky	
6,032,196	A	2/2000	Monier	7,054,322	B2	5/2006	D'Annunzio	
6,047,325	A	4/2000	Jain	7,058,052	B2	6/2006	Westphal	
6,055,236	A	4/2000	Nessett	7,068,655	B2	6/2006	March et al.	
6,055,575	A	4/2000	Paulsen	7,072,337	B1	7/2006	Arutyunov et al.	
6,058,431	A	5/2000	Srisuresh	7,072,935	B2	7/2006	Kehoe et al.	
6,061,349	A	5/2000	Coile	7,085,854	B2	8/2006	Keane et al.	
6,061,738	A	5/2000	Osaku	7,092,390	B2 *	8/2006	Wan	370/392
6,101,543	A	8/2000	Alden	7,107,464	B2 *	9/2006	Shapira et al.	709/200
6,119,171	A	9/2000	Alkhatib	7,107,614	B1 *	9/2006	Boden et al.	726/15
6,122,276	A	9/2000	Boe	7,110,375	B2	9/2006	Khalil et al.	
6,128,664	A	10/2000	Yanagidate	7,120,676	B2	10/2006	Nelson et al.	
6,137,791	A	10/2000	Frid	7,133,368	B2	11/2006	Zhang et al.	
6,154,777	A	11/2000	Ebrahim	7,139,828	B2	11/2006	Alkhatib	
6,154,839	A	11/2000	Arrow	7,181,542	B2 *	2/2007	Tuomenoksa et al.	709/250
6,173,399	B1	1/2001	Gilbrech	7,194,553	B2 *	3/2007	Lucco et al.	709/245
6,219,715	B1	4/2001	Ohno	7,227,864	B2	6/2007	Collins et al.	
6,226,751	B1 *	5/2001	Arrow et al.	7,263,719	B2 *	8/2007	James et al.	726/12
6,243,749	B1	6/2001	Sitaraman	7,327,721	B2 *	2/2008	Balasaygun et al.	370/352
6,249,801	B1	6/2001	Zisapel	7,424,737	B2 *	9/2008	Wesinger et al.	726/11
6,266,707	B1	7/2001	Boden et al.	7,490,151	B2 *	2/2009	Munger et al.	709/225
6,304,906	B1	10/2001	Bhatti	7,653,747	B2 *	1/2010	Lucco et al.	709/245
6,353,614	B1	3/2002	Borella	7,787,428	B2 *	8/2010	Furukawa et al.	370/338
6,353,886	B1	3/2002	Howard	7,814,228	B2 *	10/2010	Caronni et al.	709/245
6,381,638	B1	4/2002	Mahler	7,827,304	B2 *	11/2010	Park et al.	709/238
6,421,732	B1	7/2002	Alkhatib	7,844,718	B2 *	11/2010	Polcha et al.	709/229
6,430,622	B1	8/2002	Aiken, Jr.	7,853,714	B1 *	12/2010	Moberg et al.	709/238
6,430,623	B1	8/2002	Alkhatib	2001/0027474	A1	10/2001	Nachman et al.	
6,438,597	B1	8/2002	Mosberger	2001/0050914	A1 *	12/2001	Akahane et al.	370/382
6,438,612	B1	8/2002	Ylonen	2002/0013848	A1 *	1/2002	Rene Salle	709/226
6,452,925	B1	9/2002	Sistanizadeh	2002/0026525	A1	2/2002	Armitage	
6,457,061	B1	9/2002	Bal	2002/0053031	A1 *	5/2002	Bendinelli et al.	713/201
6,477,565	B1	11/2002	Daswani et al.	2002/0056008	A1 *	5/2002	Keane et al.	709/245
6,480,508	B1	11/2002	Mwikalo	2002/0078198	A1	6/2002	Buchbinder et al.	
6,490,289	B1	12/2002	Zhang et al.	2002/0091859	A1 *	7/2002	Tuomenoksa et al.	709/245
6,496,867	B1	12/2002	Beser	2002/0099937	A1 *	7/2002	Tuomenoksa	713/153
6,507,873	B1	1/2003	Suzuki	2002/0103931	A1 *	8/2002	Mott	709/245
6,510,154	B1	1/2003	Mayers	2002/0133534	A1 *	9/2002	Forslow	709/200
6,523,068	B1	2/2003	Beser	2003/0018912	A1	1/2003	Boyle et al.	
6,556,584	B1	4/2003	Horsley	2003/0041091	A1 *	2/2003	Cheline et al.	709/200
6,557,037	B1	4/2003	Provino	2003/0041136	A1 *	2/2003	Cheline et al.	709/223
6,557,306	B1	5/2003	Sekiya	2003/0055978	A1	3/2003	Collins	
6,591,306	B1	7/2003	Redlich	2003/0065785	A1	4/2003	Jain	
6,594,704	B1 *	7/2003	Birenback et al.	2003/0074472	A1 *	4/2003	Lucco et al.	709/245
6,618,757	B1	9/2003	Babbitt	2003/0084162	A1	5/2003	Johnson et al.	
6,629,137	B1	9/2003	Wynn	2003/0123421	A1	7/2003	Feige	
6,631,416	B2 *	10/2003	Bendinelli et al.	2003/0131131	A1 *	7/2003	Yamada et al.	709/238
6,651,101	B1	11/2003	Gai	2003/0152068	A1 *	8/2003	Balasaygun et al.	370/356
6,657,991	B1	12/2003	Akgun	2003/0208554	A1	11/2003	Holder	
6,662,223	B1	12/2003	Zhang	2003/0212795	A1	11/2003	Harris et al.	
6,697,377	B1	2/2004	Ju	2003/0219000	A1	11/2003	Magret	
6,701,437	B1 *	3/2004	Hoke et al.	2004/0006708	A1 *	1/2004	Mukherjee et al.	713/201
6,708,219	B1	3/2004	Borella et al.	2004/0088542	A1 *	5/2004	Daude et al.	713/156
6,722,210	B2	4/2004	Armstrong	2004/0111612	A1 *	6/2004	Choi et al.	713/163
6,731,642	B1	5/2004	Borella	2004/0148439	A1 *	7/2004	Harvey et al.	709/249
6,742,045	B1	5/2004	Jordan	2006/0190607	A1 *	8/2006	Lowery et al.	709/226
6,747,979	B1	6/2004	Banks	2006/0195524	A1 *	8/2006	Nichols et al.	709/204
6,754,706	B1	6/2004	Swidens	2006/0195539	A1 *	8/2006	Nichols et al.	709/206
6,772,210	B1	8/2004	Edholm	2006/0212545	A1 *	9/2006	Nichols et al.	709/219
6,778,528	B1	8/2004	Blair	2006/0212599	A1 *	9/2006	Lucco et al.	709/245
6,779,035	B1	8/2004	Gbadegesin	2007/0286189	A1 *	12/2007	Kreiner et al.	370/389
6,781,982	B1	8/2004	Borella	2008/0232295	A1 *	9/2008	Kreiner et al.	370/313
6,832,322	B1 *	12/2004	Boden et al.	2009/0116487	A1 *	5/2009	Read	370/392
6,948,003	B1	9/2005	Newman					
6,961,783	B1 *	11/2005	Cook et al.					
6,970,941	B1 *	11/2005	Caronni et al.					
6,973,485	B2	12/2005	Ebata					
6,981,020	B2	12/2005	Miloslavsky					
6,982,953	B1	1/2006	Swales					
6,983,319	B1	1/2006	Lu					

OTHER PUBLICATIONS

Tsuchiya, et al., Extending the IP Internet Through Address Reuse, ACM SIGCOMM Computer Communication Review, pp. 16-33, Jan. 1993.

Francis, et al., IPNL: A NAT-Extended Internet Architecture, SIGCOMM'01, Aug. 27-31, 2001, pp. 69-79.

US 7,949,785 B2

Page 3

- Yalagandula, et al., Transparent Mobility with Minimal Infrastructure, University of Texas at Austin, pp. 1-14, Jul. 2001.
- Teraoka, et al., VIP: A Protocol Providing Host Mobility, Communications of the ACM, Aug. 1994/vol. 37, No. 8, pp. 67-75, 113.
- Egevang, et al., The IP Network Address Translator (NAT), Network Working Group, RFC 1631, May 1994, pp. 1-10.
- Chatel, Classical Versus Transparent IP Proxies, Network Working Group, RFC 1919, Mar. 1996, pp. 1-35.
- Finseth, An Access Control Protocol, Sometimes Called TACACS, Network Working Group, Jul. 1993, pp. 1-18.
- Computer Networks. Third Edition, by Andrew S. Tanenbaum, 1996, pp. 643-670, 685-691.
- Perkins, Mobile IP, IEEE Communications Magazine, May 1997, pp. 84-99.
- "Non-Final Office Action", U.S. Appl. No. 10/233,288, (Dec. 21, 2009), 13 pages.
- "Final Office Action", U.S. Appl. No. 10/403,518, (Feb. 2, 2010), 26 pages.
- "Final Office Action", U.S. Appl. No. 10/403,829, (Feb. 23, 2010), 15 pages.
- "Advisory Action", U.S. Appl. No. 10/403,829, (May 14, 2010), 2 pages.
- "Non Final Office Action", U.S. Appl. No. 10/161,573, (May 26, 2009), 39 pages.
- "Non Final Office Action", U.S. Appl. No. 10/403,829, (Jun. 22, 2009), 26 pages.
- Francis, Gummadi "IPNL: A NAT-Extended Internet Architecture", SIGCOMM '01, (Aug. 2001), 69-79.
- "Final Office Action", U.S. Appl. No. 10/161,573, (Jul. 23, 2010), 21 pages.
- "Final Office Action", U.S. Appl. No. 10/233,288, (Aug. 3, 2010), 15 pages.
- "Non Final Office Action", U.S. Appl. No. 10/403,518, (Aug. 3, 2010), 30 pages.
- "Advisory Action", U.S. Appl. No. 10/233,288, (Oct. 15, 2010), 3 pages.
- Tanenbaum, Andrew S., "Computer Networks", Third Edition, (1996), 1-37.
- Chatel, M "Classical versus Transparent IP Proxies", *Network Working Group Request for Comments*, (Mar. 1996), 1-35.
- Finseth, C "An Access Control Protocol, Sometimes Called TACACS", *Network Working Group Request for Comments*, (Jul. 1993), 1-18.
- Francis, Paul et al., "IPNL: A NAT-Extended Internet Architecture", SIGCOMM '01, (Aug. 1, 2008), 69-79.
- Perkins, Charles E., "Mobile IP", *IEEE Communications Magazine*, Sun Microsystems, (May 1997), 84-99.
- Teraoka, Fumio et al., "VIP: A Protocol Providing Host Mobility", *Communications of the ACM*, vol. 37, No. 8, (Aug. 1994), 67-75, 113.
- Tsuchiya, Paul F., et al., "Extending the IP Internet Through Address Reuse", *ACM SIGCOMM Computer Communication Review*, (January 1993), 16-33.
- Venters, Tracy "Demystifying Protocols: A Comparison of Protocols Suitable for IP Telephony", *Sonus Networks*, (2000), 1-11.
- Yalagandula, Praveen et al., "Transparent Mobility with Minimal Infrastructure", *University of Texas at Austin*, (Jul. 2001), 1-14.
- Kessler, Gary C., "Mobile IP: Harbinger of Untethered Computing," <http://www.garykessler.net/library/mobileip.htm>, Jan. 20, 2004.
- Rekhter, "Cisco Systems' Tag Switching Architecture overview," Network Working Group, Feb. 1997.
- Kent, "Security Architecture for the Internet Protocol," Network Working Group, Nov. 1998.
- Computer Dictionary; Microsoft Press; 3rd Edition; 1997; p. 264.
- K. Egevang and P. Francis, RFC 1631, The IP Network Address Translator (NAT), May 1994.
- "Non Final Office Action", U.S. Appl. No. 10/403,829, (Nov. 19, 2010), 19 pages.

* cited by examiner

U.S. Patent

May 24, 2011

Sheet 1 of 21

US 7,949,785 B2

Fig. 1

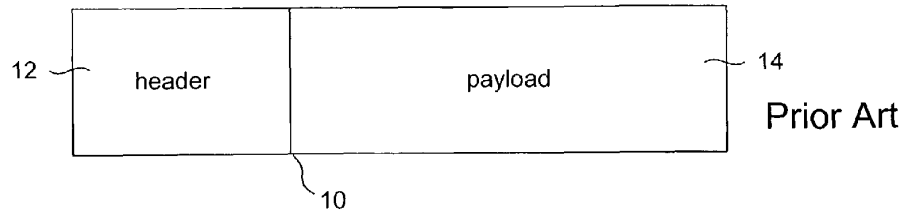


Fig. 2A

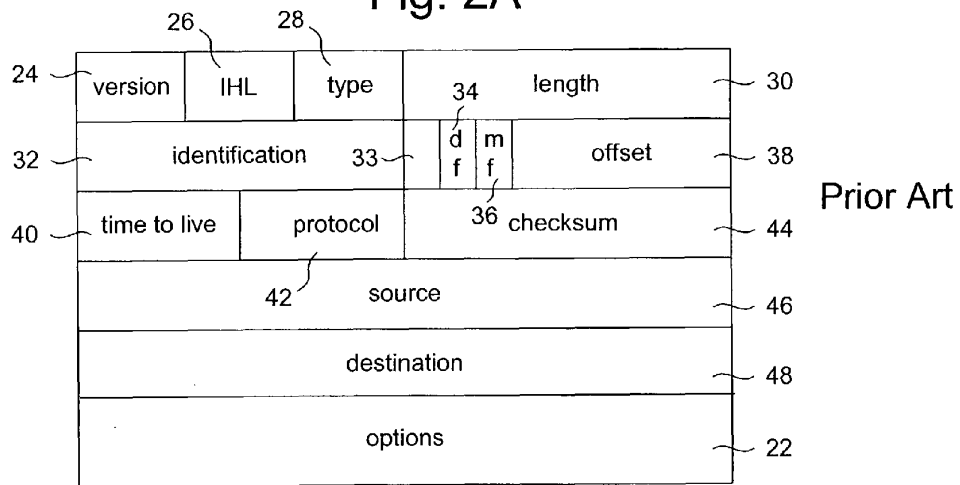
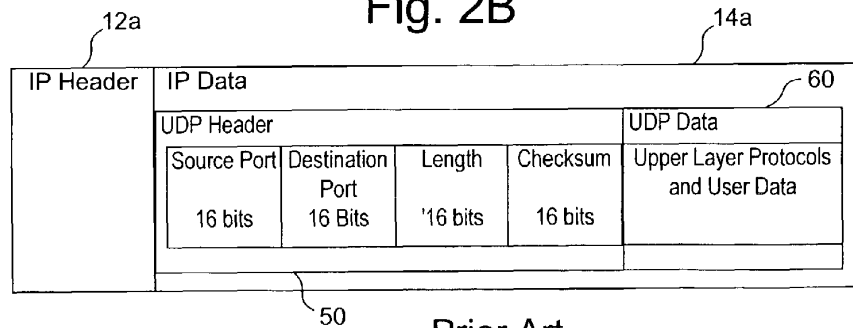


Fig. 2B



U.S. Patent

May 24, 2011

Sheet 2 of 21

US 7,949,785 B2

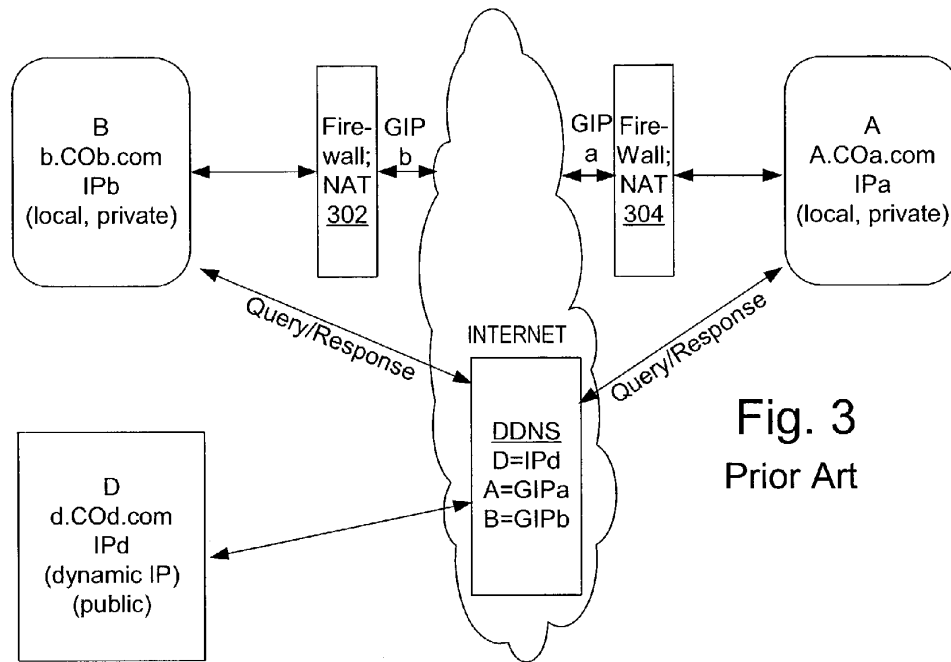
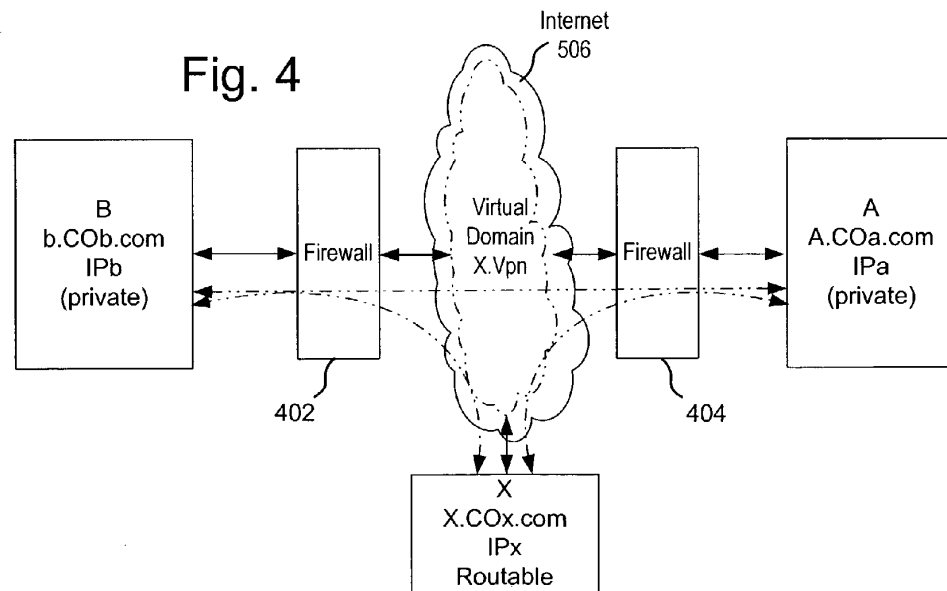


Fig. 4



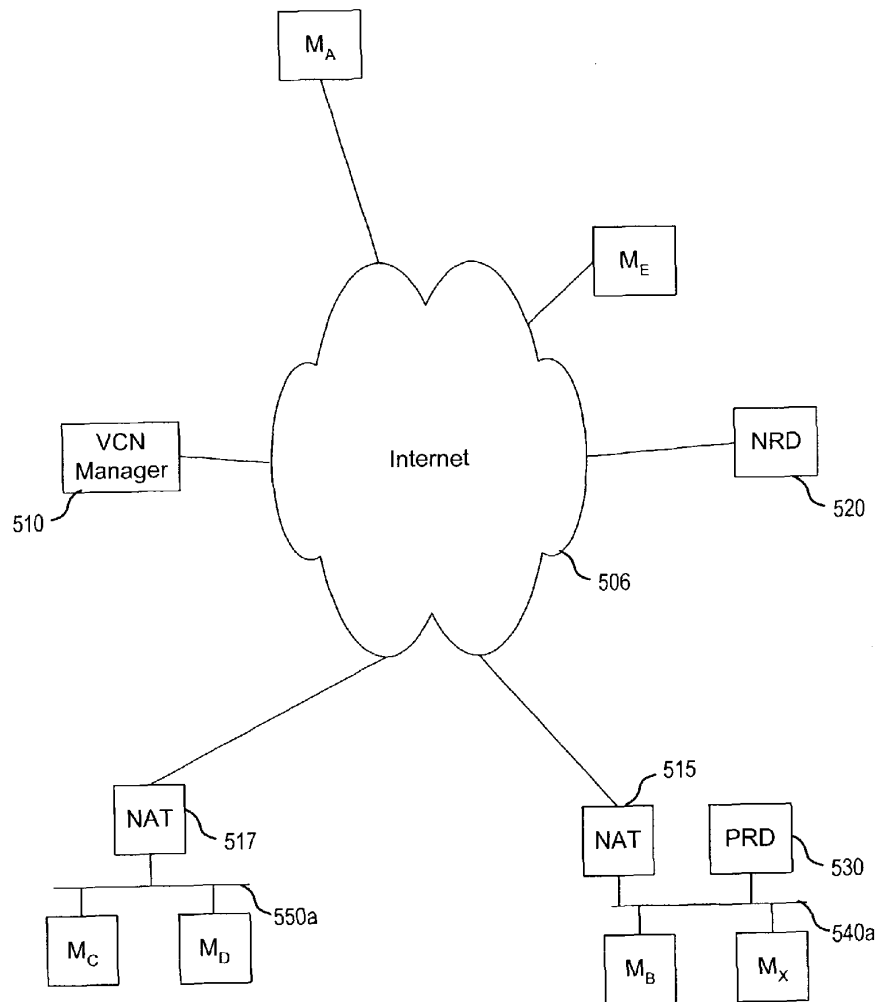
U.S. Patent

May 24, 2011

Sheet 3 of 21

US 7,949,785 B2

Fig. 5a



U.S. Patent

May 24, 2011

Sheet 4 of 21

US 7,949,785 B2

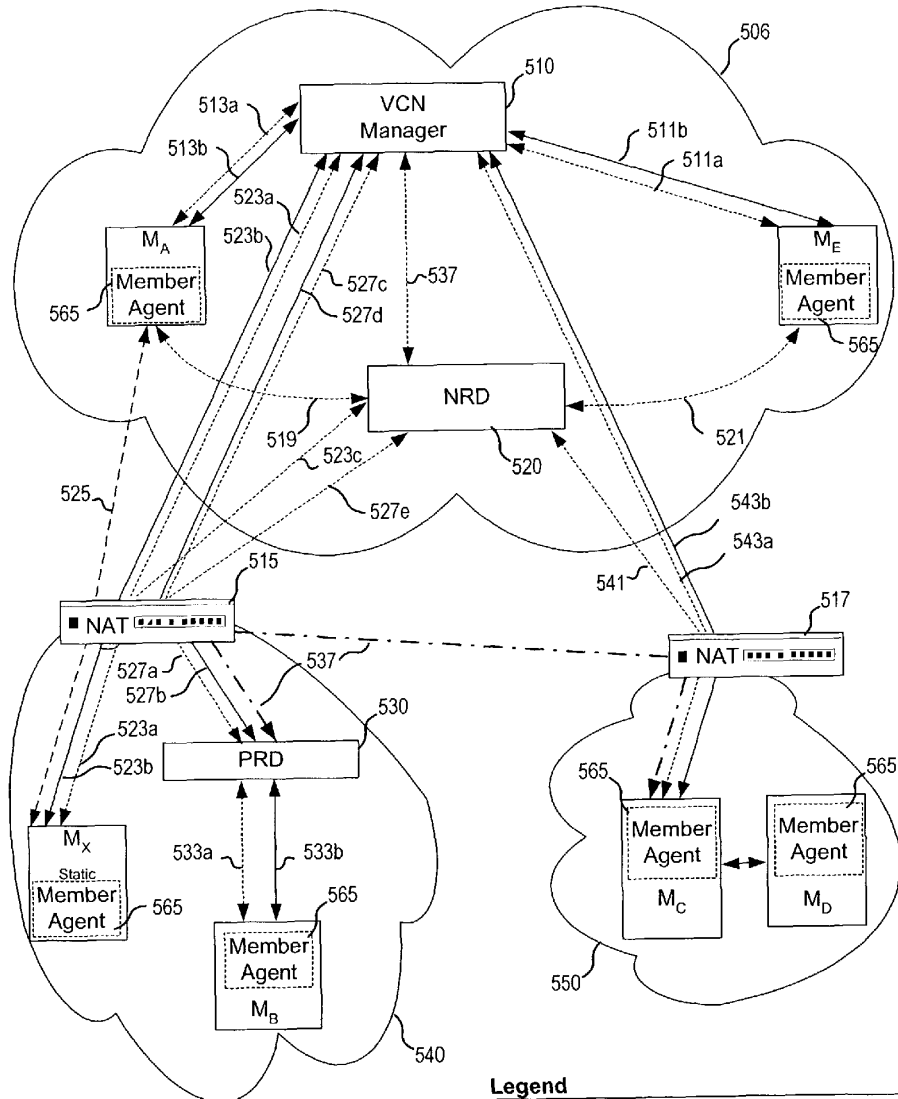
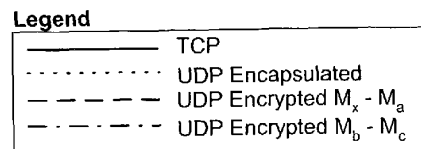


Fig. 5b



U.S. Patent

May 24, 2011

Sheet 5 of 21

US 7,949,785 B2

Fig. 6A

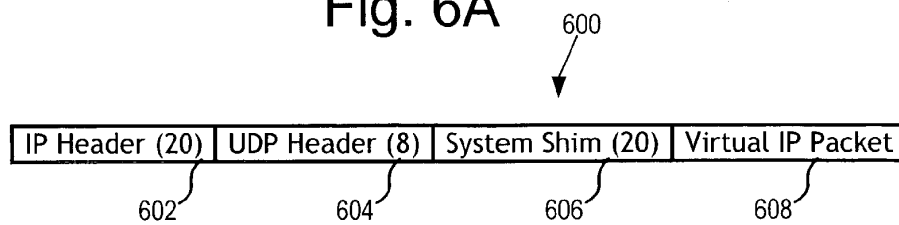


Fig. 6B

606

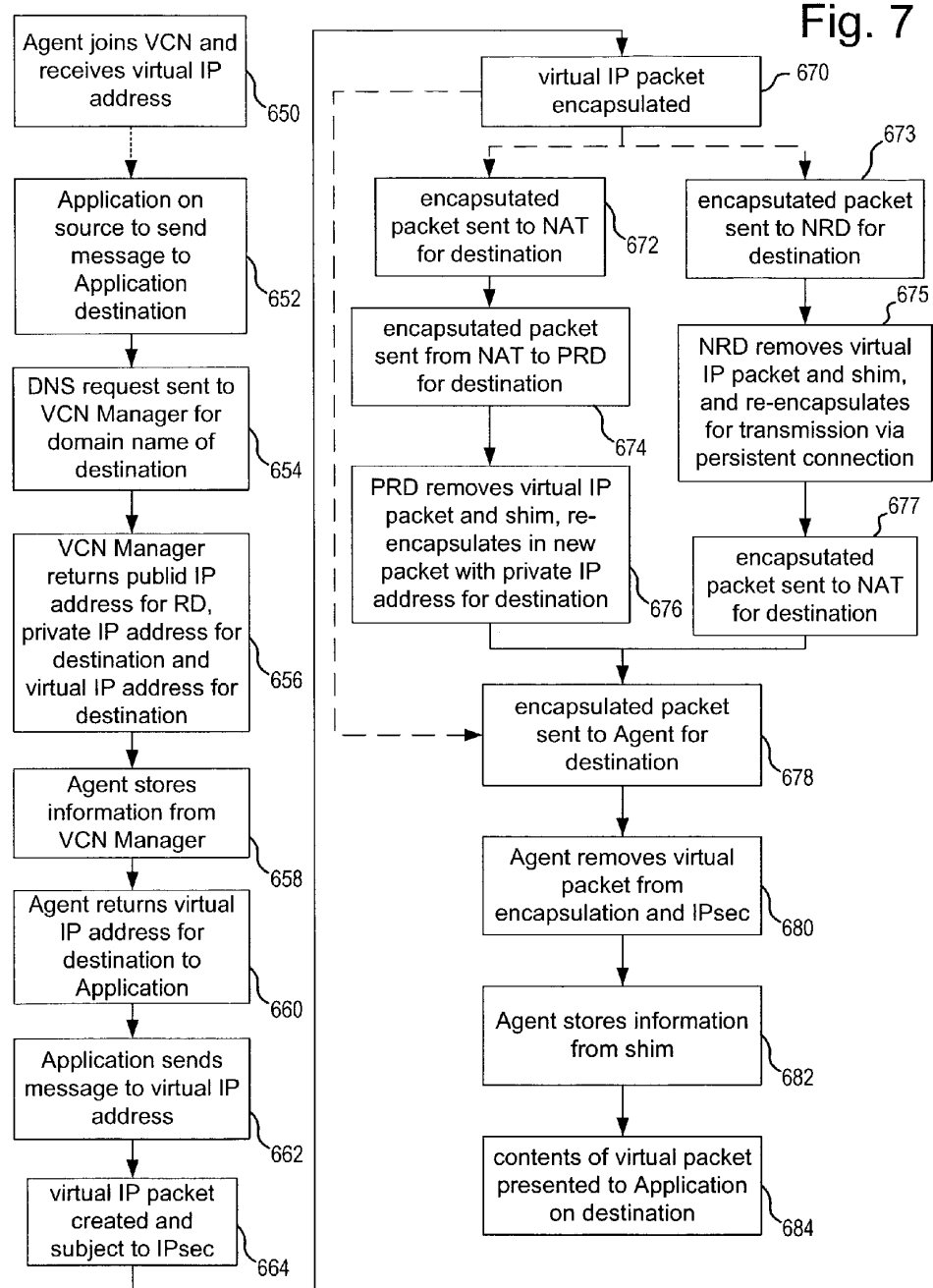
Field Name	Description
Protocol Version	Version information
Data Type	Type of data following Shim
Header Option Flags	Option Flag Information
Source RD Public IP	Source Route Director gIP
Dest RD Public IP	Destination Route Director gIP
Source Member pIP	Source Member Private IP
Dest Member pIP	Destination Member Private IP

U.S. Patent

May 24, 2011

Sheet 6 of 21

US 7,949,785 B2



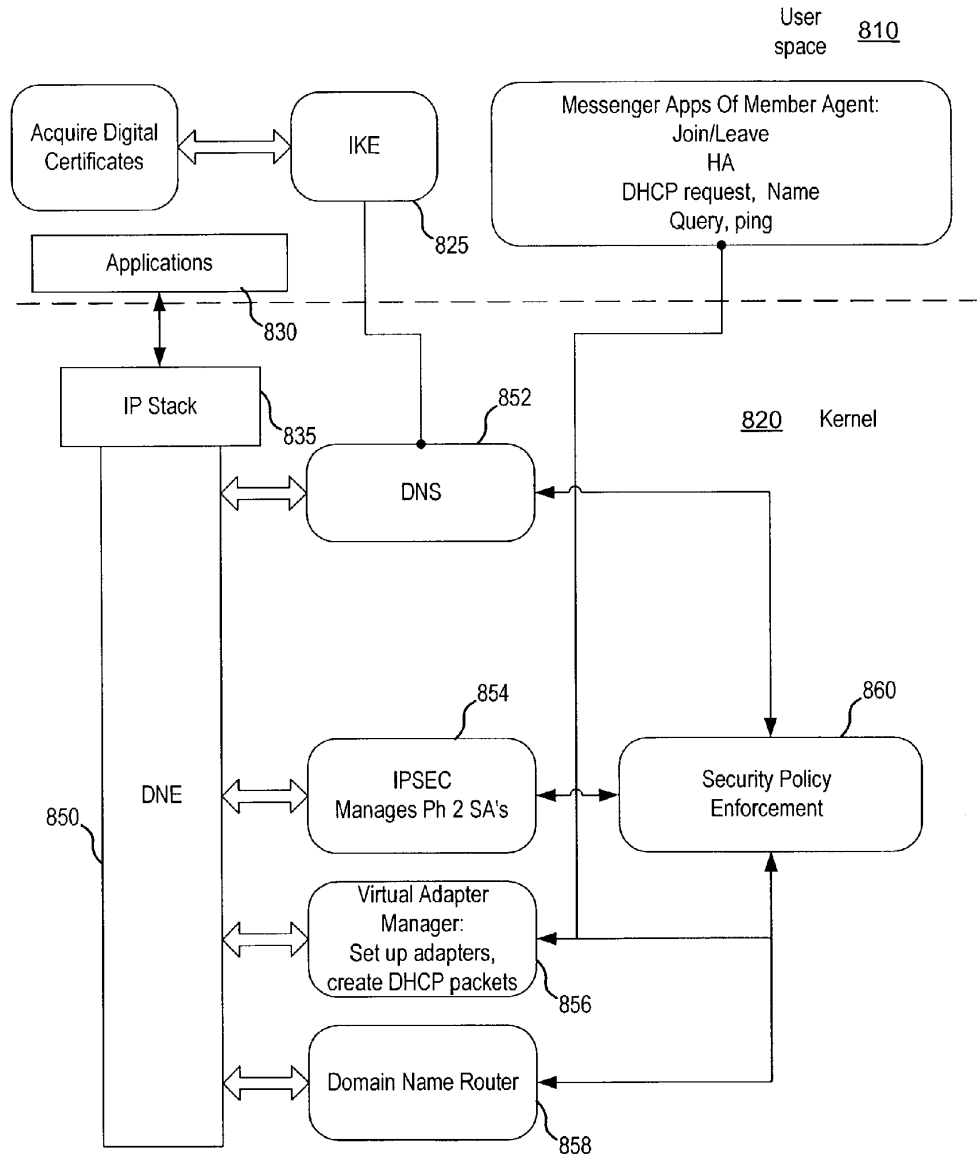
U.S. Patent

May 24, 2011

Sheet 7 of 21

US 7,949,785 B2

Fig. 8



U.S. Patent

May 24, 2011

Sheet 8 of 21

US 7,949,785 B2

Fig. 9

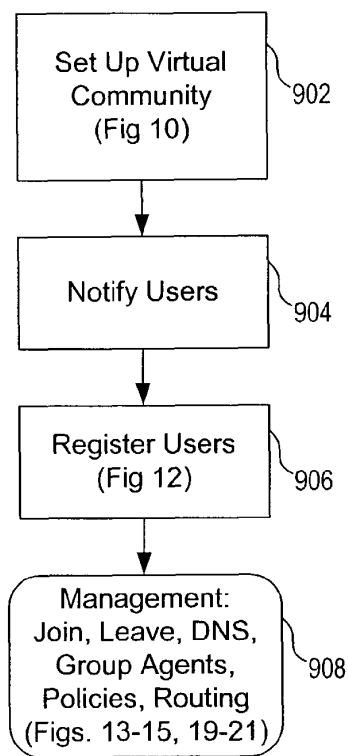
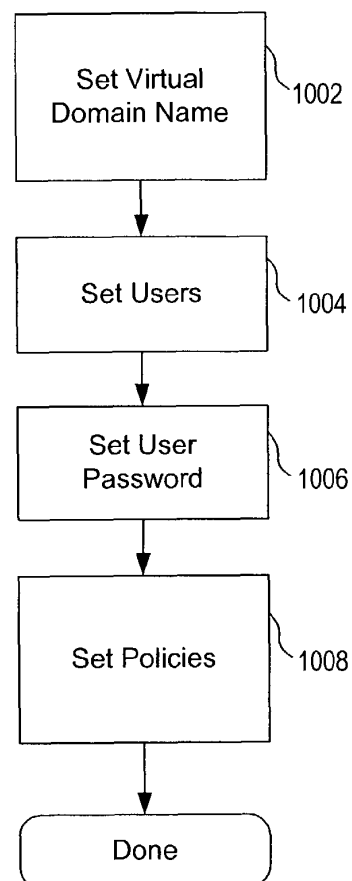


Fig. 10



U.S. Patent

May 24, 2011

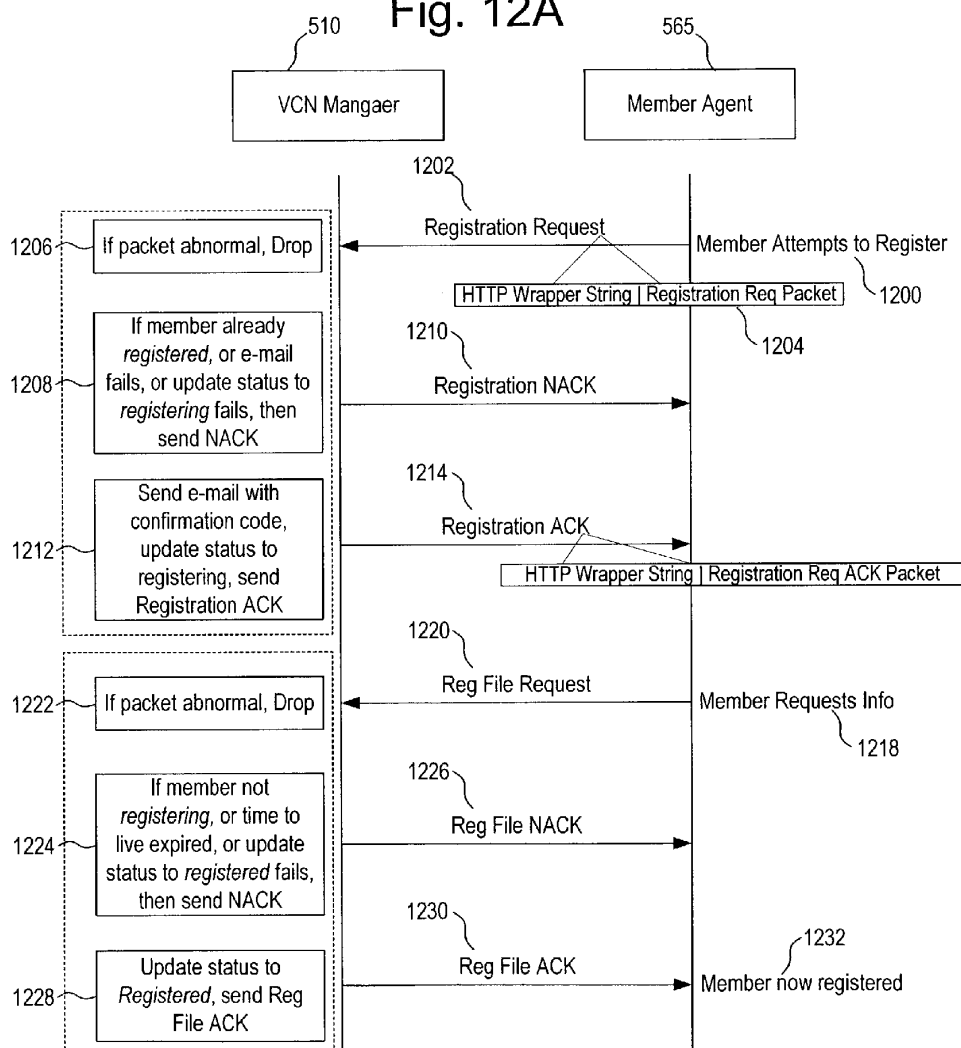
Sheet 9 of 21

US 7,949,785 B2

Fig. 11



Fig. 12A



U.S. Patent

May 24, 2011

Sheet 10 of 21

US 7,949,785 B2**Fig. 12B**

Field Name	Type	Description
Manager Seed	Array (32)	From Init Ack
Init Ack Time	Uint (8)	From Init Ack
Join Flags	Uint (2)	Indicates if rejoin and/or Group
Member Local IP	Uint (4)	The machine's local IP address
RD Local IP	Uint (4)	The Route Director local IP address
RD Global IP	Uint (4)	The Route Director Global IP
<Unused>	Uint (4)	
Member AH SPI	Uint (4)	For IPsec
Member ESP SPI	Uint (4)	For IPsec
Member Listen Port	Uint (2)	For Heartbeat Protocol
Token Timestamp	Uint (8)	The UTC time Token Key was created
Member Run ID	Array (16)	GUID Identifying current client run
Member Netbios Name	Array (16)	Microsoft NetBios support
SP Version	Uint (4)	Security Policy version
HA Version	Uint (4)	High availability version
Password Hash	Uint (20)	SHA1 of the user password string
Padding	Uint (.)	End on a multiple of 16 octets

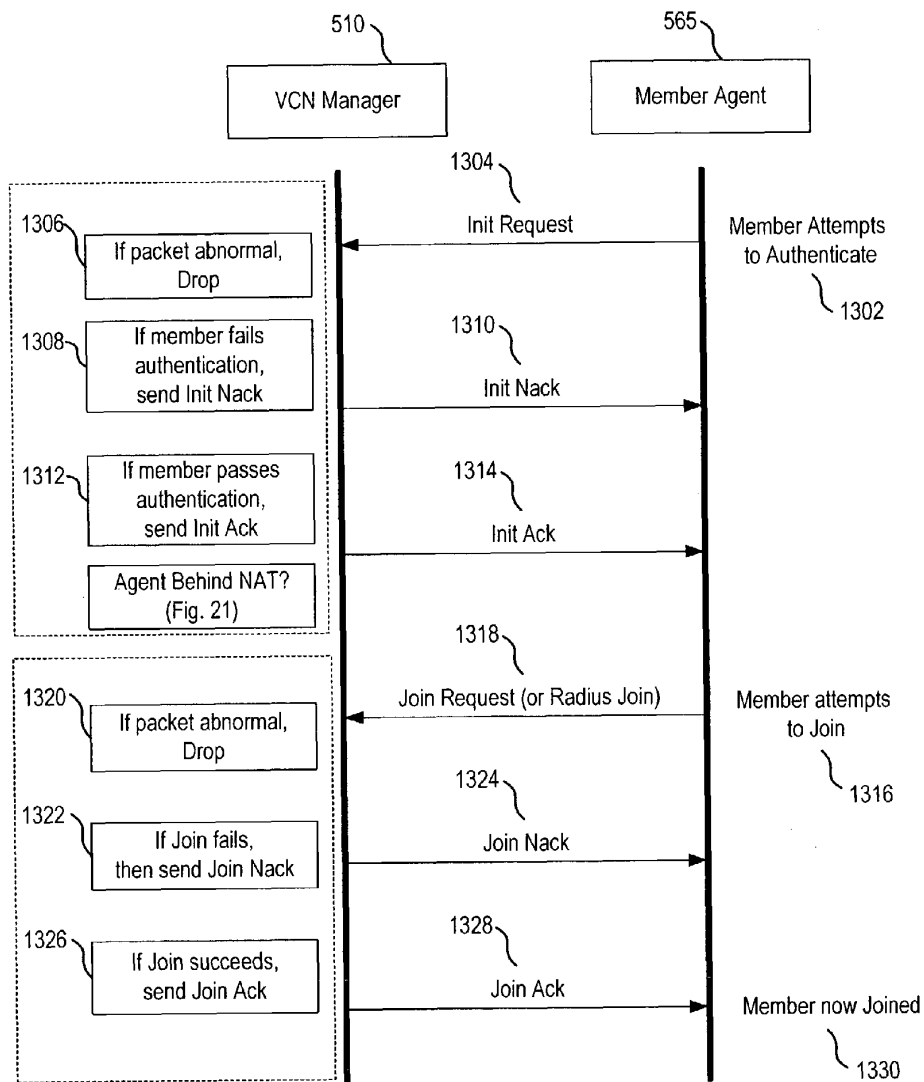
U.S. Patent

May 24, 2011

Sheet 11 of 21

US 7,949,785 B2

Fig. 13



U.S. Patent

May 24, 2011

Sheet 12 of 21

US 7,949,785 B2

Fig. 14

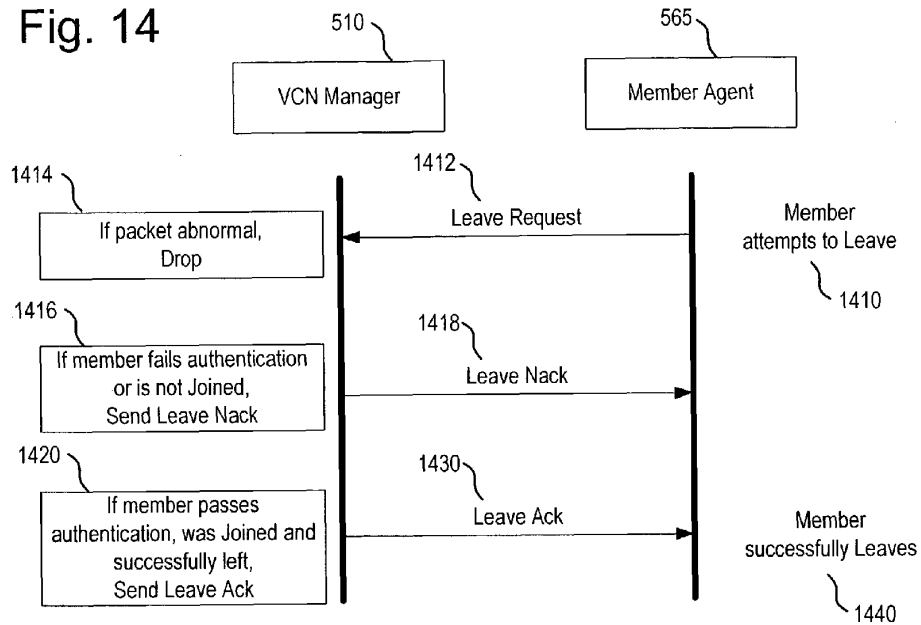
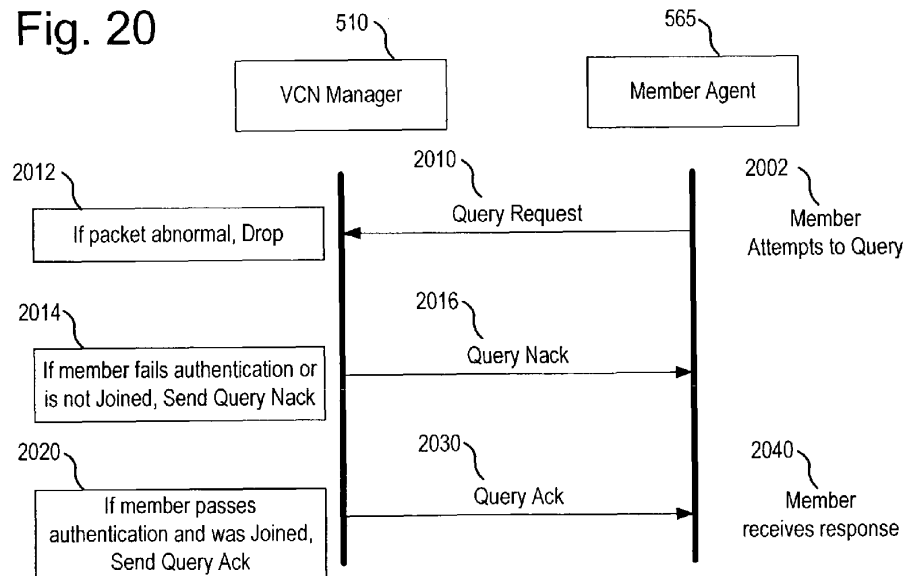


Fig. 20



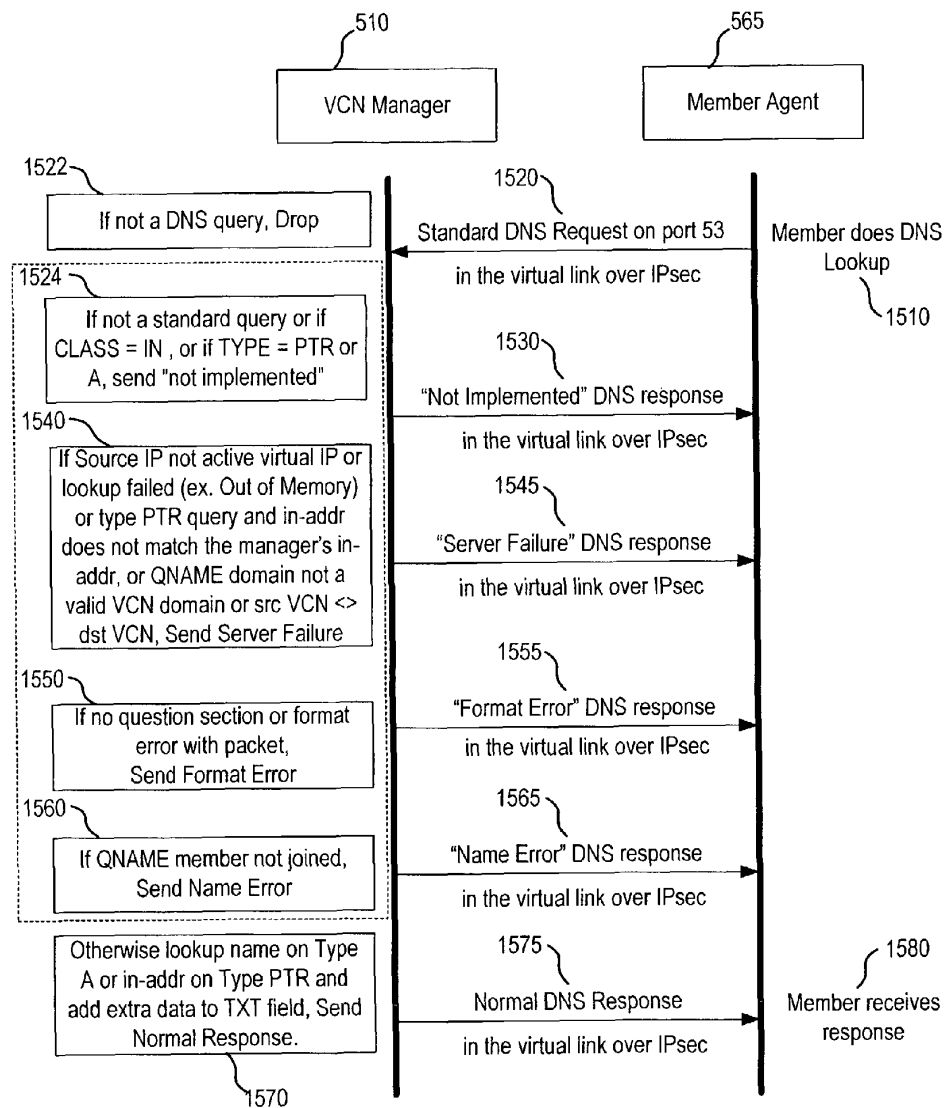
U.S. Patent

May 24, 2011

Sheet 13 of 21

US 7,949,785 B2

Fig. 15



U.S. Patent

May 24, 2011

Sheet 14 of 21

US 7,949,785 B2

Fig. 16A

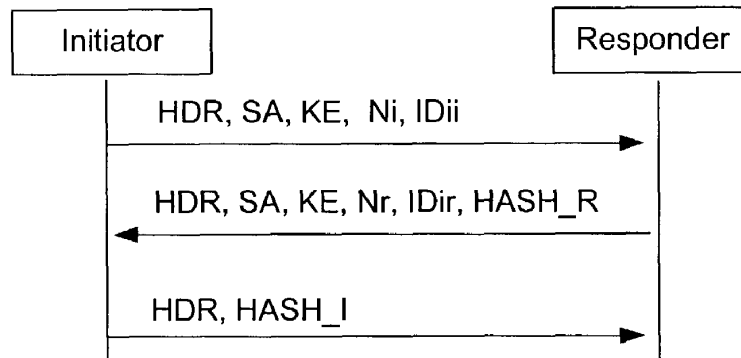
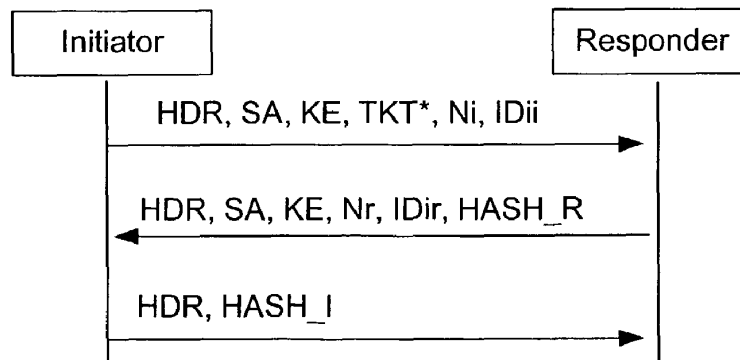


Fig. 16B



U.S. Patent

May 24, 2011

Sheet 15 of 21

US 7,949,785 B2

Fig. 17

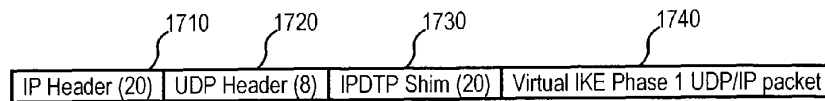
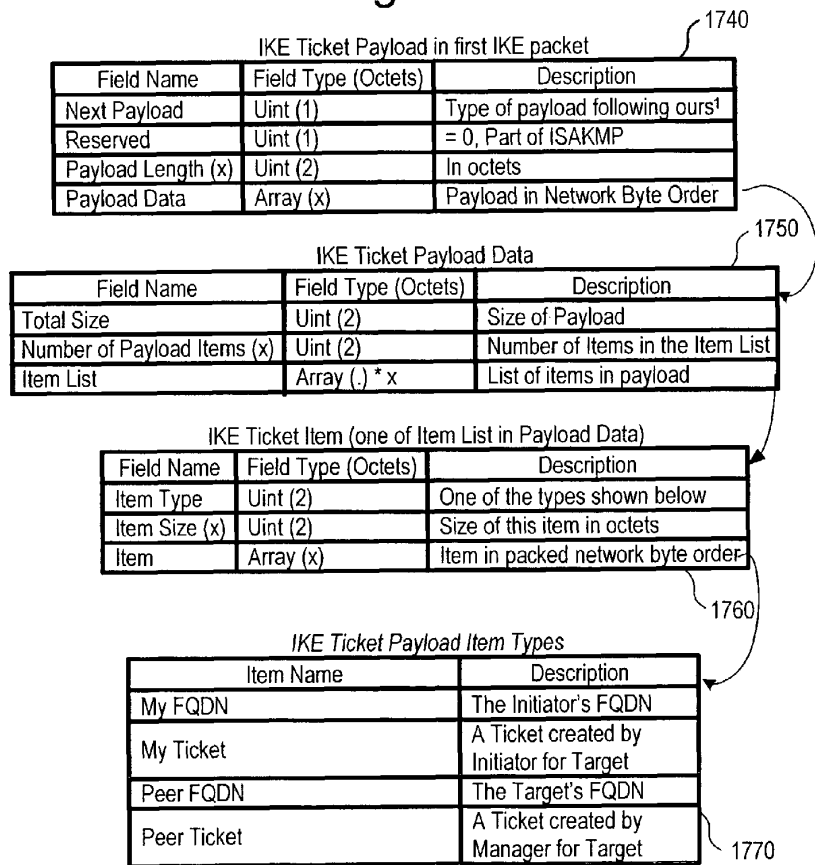


Fig. 18



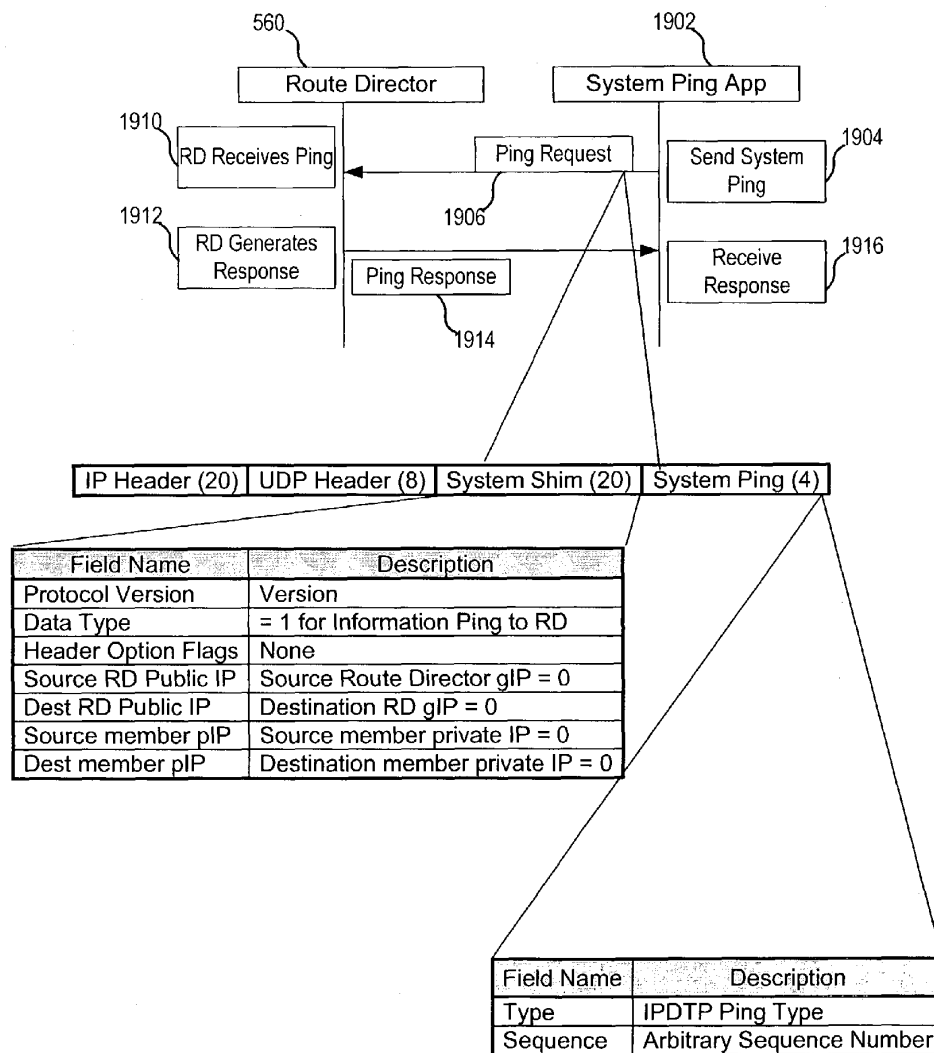
U.S. Patent

May 24, 2011

Sheet 16 of 21

US 7,949,785 B2

Fig. 19



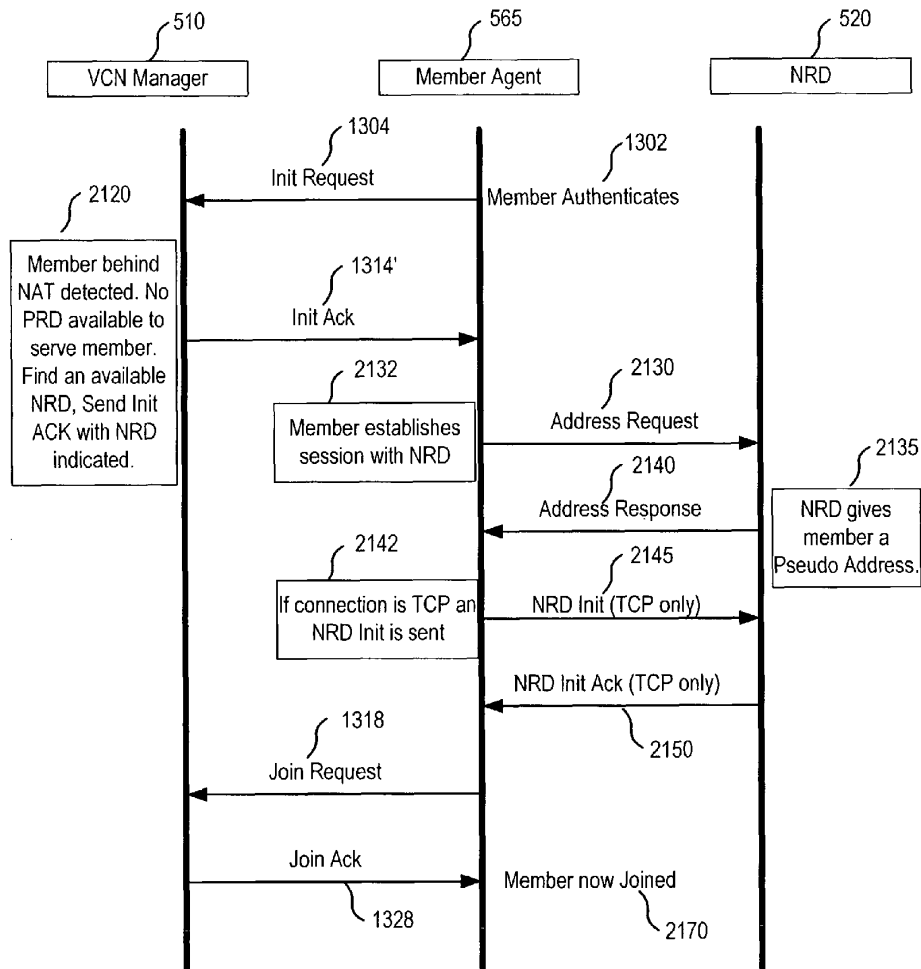
U.S. Patent

May 24, 2011

Sheet 17 of 21

US 7,949,785 B2

Fig. 21



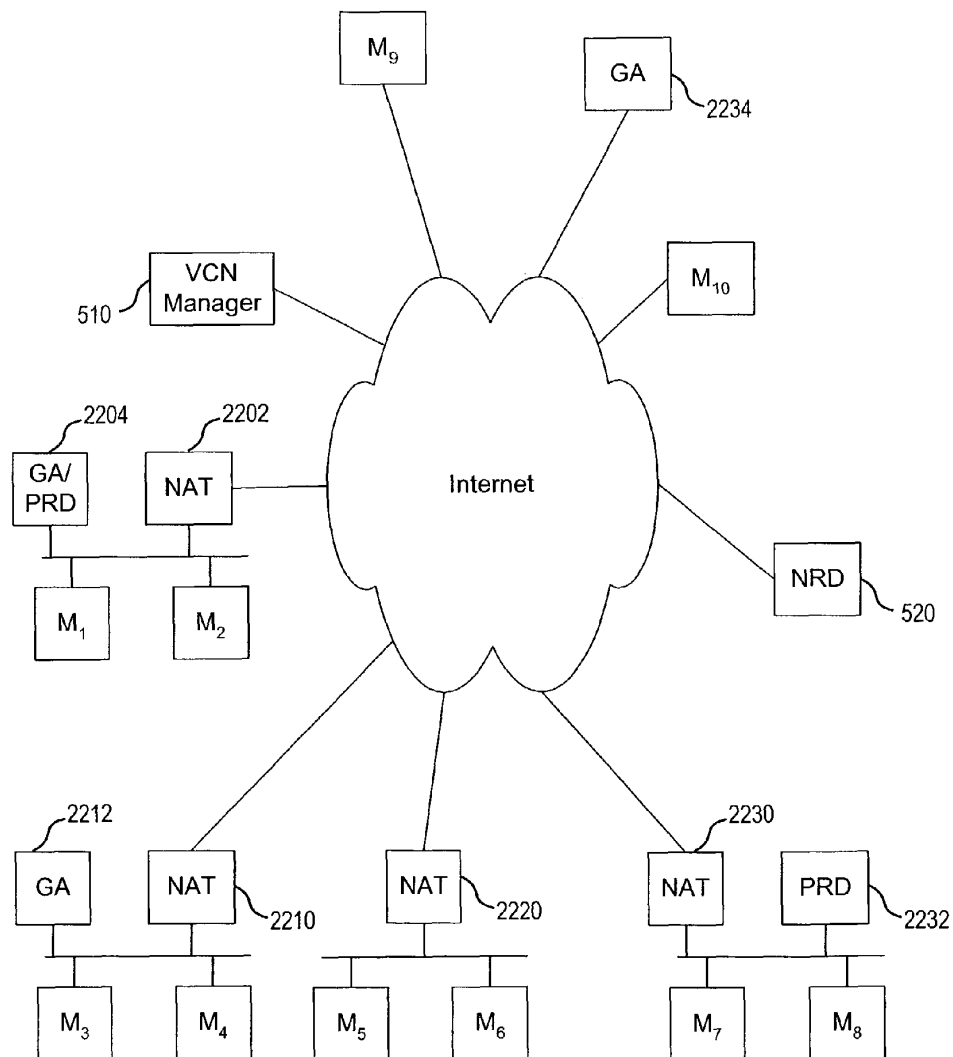
U.S. Patent

May 24, 2011

Sheet 18 of 21

US 7,949,785 B2

Fig. 22



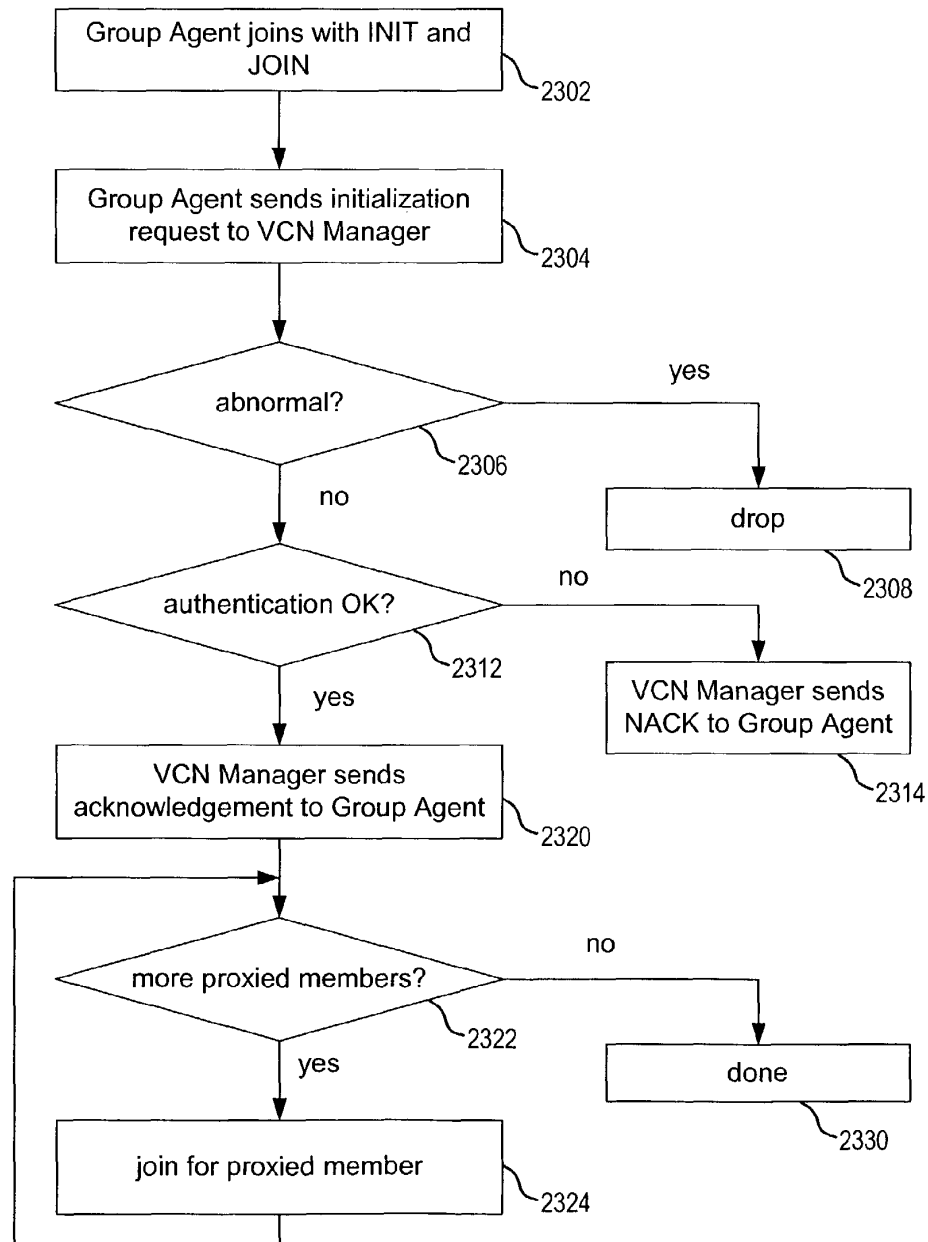
U.S. Patent

May 24, 2011

Sheet 19 of 21

US 7,949,785 B2

Fig. 23



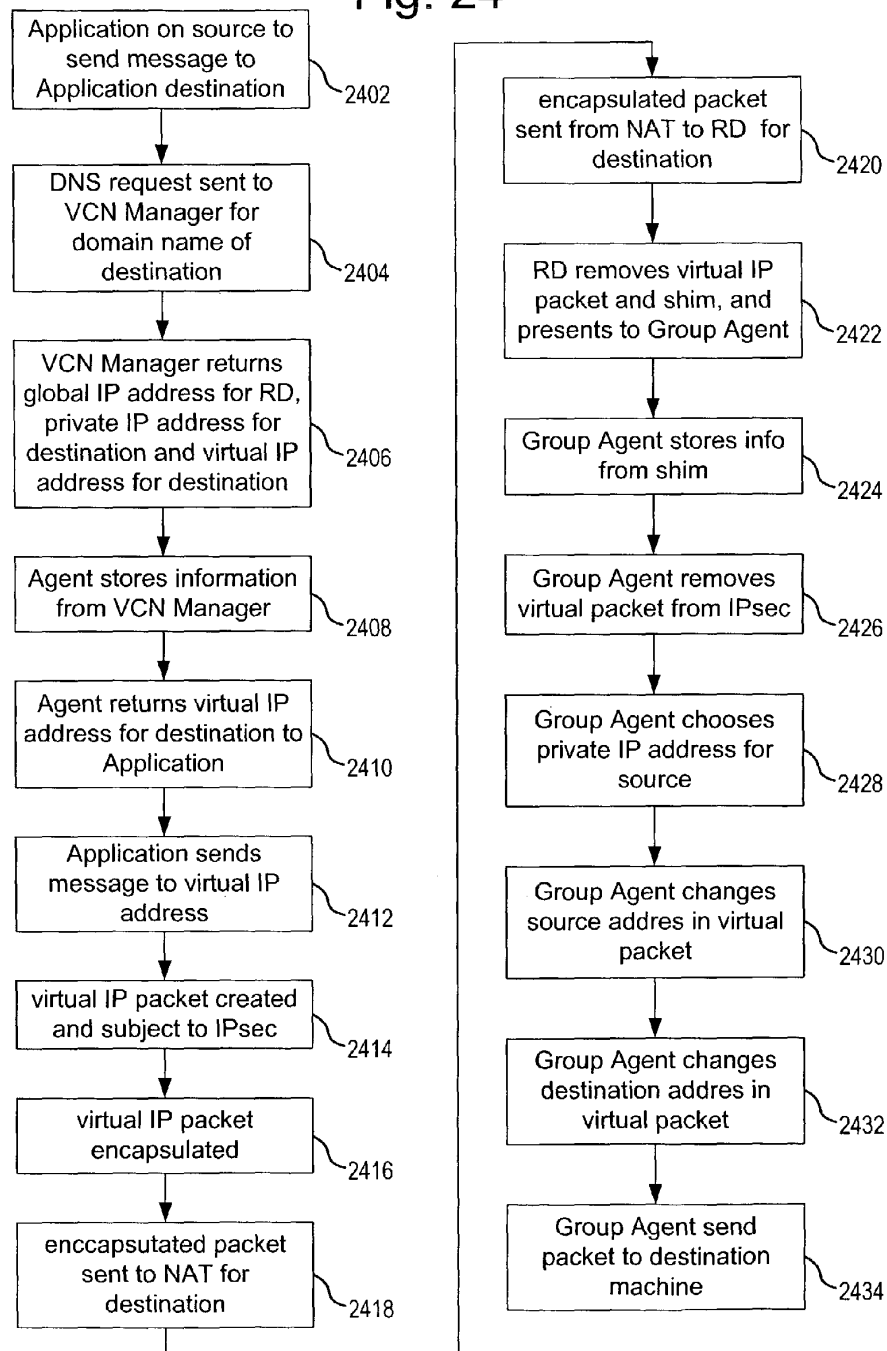
U.S. Patent

May 24, 2011

Sheet 20 of 21

US 7,949,785 B2

Fig. 24



U.S. Patent

May 24, 2011

Sheet 21 of 21

US 7,949,785 B2

Fig. 25

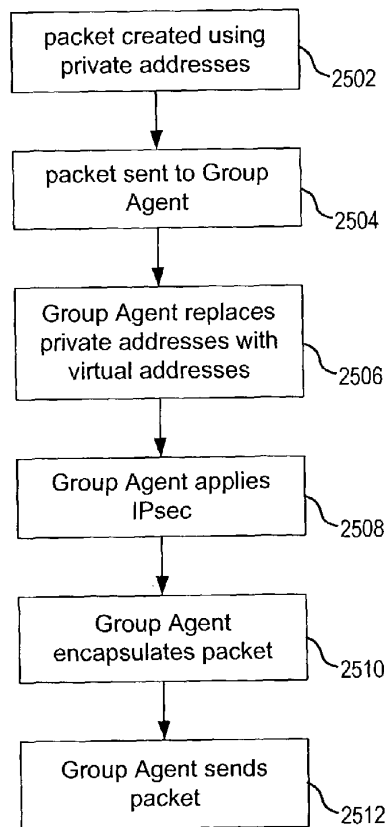
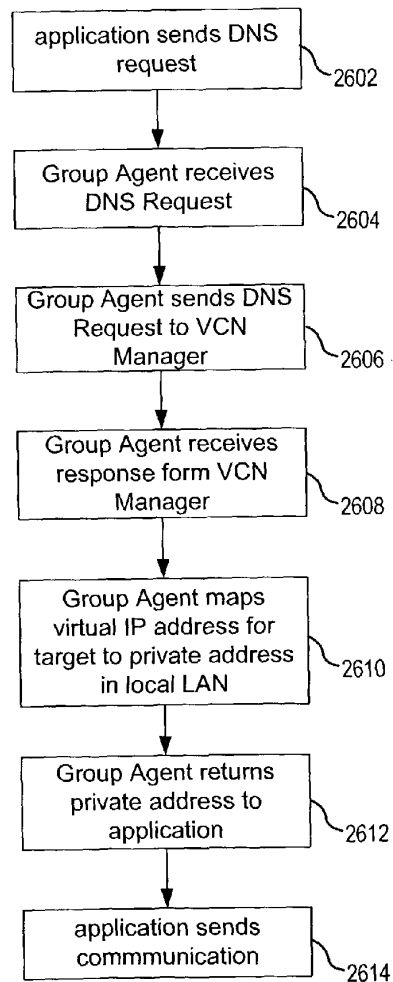


Fig. 26



US 7,949,785 B2

1

**SECURE VIRTUAL COMMUNITY NETWORK
SYSTEM****CROSS-REFERENCE TO RELATED
APPLICATIONS**

This Application is related to the following Applications: U.S. Pat. No. 7,139,828 issued Nov. 21, 2006; U.S. patent application Ser. No. 10/161,573, "Creating A Public Identity For An Entity On A Network," filed on Jun. 3, 2002; U.S. patent application Ser. No. 10/233,288, "Communicating With An Entity Inside A Private Network Using An Existing Connection To Initiate Communication," filed on Aug. 30, 2002; U.S. patent application Ser. No. 10/403,829 "Secure Virtual Address Realm," filed on Mar. 31, 2003; and U.S. patent application Ser. No. 10/403,518 "Group Agent," filed on Mar. 31, 2003. All of these related applications are incorporated herein by reference in their entirety.

BACKGROUND OF THE INVENTION**1. Field of the Invention**

The present invention is directed to a network services system.

2. Description of the Related Art

Networked data devices provide users with efficient means for communication and collaboration. In recent years, the Internet has played a central role in allowing different types of networked devices to connect and share information across a myriad of networks. As technology advances and more organizations and people rely on the Internet, new challenges are presented for enhancing the ability to communicate. One such challenge is to enable the rapid creation of a secure means that allows local and remote specified entities to communicate and collaborate from any location via a standard Internet connection. For any solution to be widely accepted, it must take into account the realities and limitations currently existing.

For communication on the Internet, the Internet Protocol (IP) has become the default protocol used by most hosts and to which communication applications are now written. To transmit data from a source to a destination, the Internet Protocol uses an IP address. An IP address is four bytes long and consists of a network number and a host number. When written out, IP version 4 addresses are specified as four numbers separated by dots (e.g. 198.68.70.1). Users and software applications do not always refer to hosts or other resources by their numerical IP address. Instead of using numbers, they use ASCII strings called domain names. The Internet uses a Domain Name System (DNS) to convert a domain name to an IP address.

The Internet Protocol has been in use for over two decades. It has worked extremely well, as demonstrated by the exponential growth of the Internet. Unfortunately, the Internet is rapidly becoming a victim of its own popularity, it is running out of addresses.

One popular solution to the depleting address problem is Network Address Translation (NAT). This concept includes predefining a number of network addresses to be private addresses and public addresses. Public addresses are unique addresses that should only be used by one entity having access to the Internet. That is, no two entities on the Internet should have the same public address. Private addresses are not unique and are typically used for entities not having direct access to the Internet. Private addresses are used in private networks. Private addresses can be used by more than one organization or network. NAT assumes that all of the machines on a network will not need to access the Internet at

2

all times. Therefore, there is no need for each machine to have a public address. A local network can function with a small number of one or more public addresses assigned to a NAT device. The remainder of the machines on the network will be assigned private addresses. Since entities on the network have private addresses, the network is considered to be a private network.

When a particular machine having a private address on the private network attempts to initiate a communication to a machine outside of the private network (e.g. via the Internet), the NAT device will intercept the communication, change the source machine's private address to a public address and set up a table for translation between public addresses and private addresses. The table can contain the destination address, port numbers, sequencing information, byte counts and internal flags for each connection associated with a host address. Inbound packets are compared against entries in the table and permitted through the NAT device only if an appropriate connection exists to validate their passage. One problem with many NAT implementations is that it only works for communication initiated by a host within the private network to a host on the Internet that has a public IP address. Many NAT implementations will not work if the communication is initiated by a host outside of the private network and is directed to a host with a private address in the private network.

For most organizations, the security of devices coupled to the Internet is a concern. As a result, not all devices are directly connected to or accessible via the Internet. Rather, many devices are placed in private networks for security concerns (in addition to the address usage issue described above). Many private networks are secured by placing a firewall device between the private network and the Internet.

Another problem with many current communication schemes is that mobile computing devices can be moved to new and different networks, including private networks. These mobile computing devices may need to be reachable so that a host outside of the private network can initiate communication with the mobile computing device. However, in this case the problem is two-fold. First, there is no means for allowing the host outside of the private network to initiate communication with the mobile computing device. Second, the host outside the private network does not know the address for the mobile computing device or the network that the mobile computing device is currently connected to.

Thus, there is a need for a system that provides for local and remote entities to communicate and collaborate using the Internet, can work with existing NAT devices and firewalls, and allows for devices to move to different physical networks. To increase the ability of such a system to be accepted by the Internet community, it is desirable for such a system to not require changes to existing applications, allow peer-to-peer applications to communicate directly across the Internet and to not require changes to existing protocols. Each of these issues will be discussed below.

Large amounts of resources have been used to purchase and deploy existing applications currently running on the millions of computing devices. Organizations and individuals are not likely to want to adopt new communications solutions that require them to absorb the additional cost of replacing all of their applications.

To provide efficient and secure communication, it is desirable for devices to have the ability to allow their IP based applications to communicate directly with each other. By allowing peer-to-peer applications to communicate directly across the Internet, security is enhanced since the recipient is specifically identified and communication is passed directly between the applications on two or more respective machines.

US 7,949,785 B2

3

Most machines on the Internet use the TCP/IP (Transmission Control Protocol/Internet Protocol) reference model to send data to other machines on the Internet. The TCP/IP reference model includes four layers: the physical and data link layer, the network layer, the transport layer, and the application layer. The physical layer portion of the physical and data link layer is concerned with transmitting raw bits over a communication channel. The data link portion of the Physical and Data Link layer takes the raw transmission facility and transforms it into a line that appears to be relatively free of transmission errors. It accomplishes this task by having the sender break the input data up into frames, transmit the frames and process the acknowledgment frames sent back by the receiver.

The network layer permits a host to inject packets into a network and have them travel independently to the destination. On the Internet, the protocol used for the network layer is the Internet Protocol (IP).

The transport layer is designed to allow peer entities on the source and destination to carry on a "conversation." On the Internet, two protocols are used. The first one, the Transmission Control Protocol (TCP), is a reliable connection-oriented protocol that allows a byte stream originating on one machine to be delivered without error to another machine on the Internet. It fragments the incoming byte stream into discrete segments and passes each one to the network layer. At the destination, the receiving TCP process reassembles the received segments into the output stream. TCP also handles flow control to make sure a fast sender cannot swamp a slow receiver with more segments than it can handle. The second protocol used in the transport layer on the Internet is the User Datagram Protocol (UDP), which does not provide the TCP sequencing or flow control. UDP is typically used for one-shot, client server type requests-reply queries for applications in which prompt delivery is more important than accurate delivery.

The transport layer is typically thought of as being above the network layer to indicate that the network layer provides a service to the transport layer. Similarly, the transport layer is typically thought of as being below the application layer to indicate that the transport layer provides a service to the application layer.

The application layer contains the high level protocols, for example, Telnet, File Transfer Protocol (FTP), Electronic Mail-Simple Mail Transfer Protocol (SMTP), and Hypertext Transfer Protocol (HTTP).

FIG. 1 depicts the basic structure of an IP version 4 packet 10 used at the Network Layer. IP packet 10 consists of header 12 and payload 14. Payload 14 stores the data received from the Transport Layer in the TCP/IP model. FIG. 2A depicts the format of a header of an IP packet. The header is depicted to include six rows. The first five rows are 32 bits wide. The first five rows of the header comprise a 20 byte fixed portion of the header. The last row of the header provides a variable sized options field 22. Version field 24 keeps track of which version of the protocol the packet belongs to. The current version used on the Internet is version 4. IHL field 26 describes the length of the header in 32 bit words. Type field 28 indicates the type of service requested. Various combinations of reliability and speed are possible. Length field 30 contains the size of the packet, including both the header and the data. Identification field 32 is needed to allow the destination host to determine which packet the received fragment belongs to. All fragments of a packet contain the same identification value. Next comes three flags, which include an unused bit 33 and then two 1 bit fields 34 and 36. DF field 34 stands for don't fragment. It is an order to the routers not to fragment the packet because the

4

destination is incapable of putting the pieces back together again. MF field 36 stands for more fragments. All fragments except for the last one have this bit set. Fragment offset field 38 indicates where in the current segment this fragment belongs. Time to Live field 40 is used to limit packet lifetime. It is supposed to count time in seconds, allowing a maximum life time of 255 seconds. In practice, it may count hops (or hops and seconds). The time is decremented on each hop by a router. When the time to live hits 0, the packet is discarded and a warning is sent back to the source using an Internet Control Messaging Protocol (ICMP) packet. This feature prevents packets from wandering around forever. Protocol Field 42 indicates which transport layer type is to receive the segment. TCP is one possibility, UDP is another. Checksum field 44 verifies the header. One method for implementing a checksum is to add up all 16 bit half words constituting the header and take the ones compliment of the result. Note that the checksum must be recomputed at each hop because the Time to Live field 40 changes or the content of the options field changes. Source field 46 indicates the IP address for the source of the packet and destination field 48 indicates the IP address for the destination of the packet. Options field 22 is a variable length field designed to hold other information. Currently, options used on the Internet indicate security, suggested routing path, previous routing path and time.

UDP is an alternative to TCP. FIG. 2B depicts the structure of a packet that uses UDP. Like the TCP, UDP uses the Internet Protocol to actually send a data unit from one computer to another. Unlike TCP, however, UDP does not provide the service of dividing a message into packets (datagrams) and reassembling it at the other end. Specifically, UDP does not provide sequencing of the packets that the data arrives in. Hence, application programs using UDP must ensure that the entire message has arrived and is in the right order. As shown in FIG. 2B, the packet includes an IP header 12a and IP payload 14a. The payload 14a comprises the UDP header 50 and UDP data 60. UDP header 50 consists of a 16 bit source port identifier, a 16 bit destination port identifier, a length field and checksum field. In FIG. 2B, the Destination Port has a meaning within the context of a particular Internet destination address, and the Length field is the length in octets of this user datagram, including its header and the data. The checksum is the 16-bit one's complement of the sum of consecutive two octaves of a pseudo header of information from the IP header, the UDP header, and the data (padded with zero octets at the end, if necessary, to make a multiple of two octets).

In addition to using the existing Internet infrastructure, another issue in allowing public-to-private, or private-to-private, communications lies in the addressing of the devices. Where a system is coupled to the public Internet with an IP address, communication packets can be routed directly to the machine. However, many devices couple to the Internet via service providers which provide them with a dynamic IP address. Thus, those wishing to communicate with this type of user must know the constantly changing address of the user in order to communicate with them. Still other hosts may be coupled to networks that use technologies other than IP.

One solution currently in use that provides for local and remote specified entities to communicate and collaborate using the Internet is the Virtual Private Network ("VPN"). The VPN uses additional network software layers to increase security between users in the public realm and those in private realms. For example, some VPNs encrypt packets using IPsec (or other protocols). The encrypted packets are then encapsulated within a standard packet and transmitted across the Internet to the destination. At the destination, the encrypted packet is decrypted. While the existing VPN provides remote

US 7,949,785 B2

5

users with secure access to a network, via the Internet, many existing VPNs have various shortcomings that prevent them from satisfying the needs of many users. For example, many VPNs do not provide for peer-to-peer communication with IPsec (or other security measures), do not work though NAT devices in all cases, are difficult to set up and maintain, do not provide for full mobility of entities communicating on the VPN, and do not always provide for communication with entities in the various private network configurations discussed herein.

One method of overcoming the mobility problem includes the use of Dynamic DNS. More information about Dynamic DNS can be found in RFC 2136 incorporated herein by reference.

Dynamic DNS is illustrated in FIG. 3. FIG. 3 shows a first computer or device B having a host name of B.CO.b.com having a dynamic or static private IP address IP_B and a second computer or device A having a host name of A.CO.a.com and having a dynamic or static private IP address IP_A. Devices B and A are coupled to the Internet 506 via firewall devices 302, 304 incorporating NAT. The addresses of B and A, as seen by devices on the Internet, are public IP addresses GIP_B and GIP_A, respectively. Also shown is device D, having a publicly address IP_D and a host name of d.CO.d.com.

Also shown is a Dynamic DNS server, DDNS, residing on the Internet. In essence, the DDNS server is a DNS server supporting the dynamic DNS protocol of RFC 2136, and in this example is an authoritative name server containing records for B, A and D. The DDNS server will update its records of B, A and D in real time, so that if or when D's address IP_D changes, any query by other devices (B, A, for example) based on D's host name, will result in the response being the correct IP-IP_D. Any DNS records for devices B and A will always reflect the public IP addresses (GIP_A and GIP_B) of the firewall/NAT devices.

Unfortunately, DDNS technology is complex and difficult to implement securely—two factors that have dramatically slowed the rate of deployment of Dynamic DNS. As a result, VPNs have not been able to adopt DDNS to solve all of the problems discussed herein.

Hence, a system is desired that allows local and remote entities to communicate and collaborate from any location via a standard Internet connection and which solves the problems discussed above.

SUMMARY OF THE INVENTION

A system is disclosed that allows local and remote entities to communicate and collaborate from various locations. In one embodiment, the system makes use of a virtual subnet with a virtual address realm. The virtual address realm allows two or more users to communicate securely via at least one public network, whether the users are both coupled directly to the public network, both coupled directly to separate private networks or whether one is in a public network and one is in a private network. The virtual address realm uses virtual addresses to identify the devices on the virtual subnet. Although the devices are in different physical subnets, from the point of view of the applications the devices of the virtual subnet appear to be in one local subnet.

In one embodiment, the virtual address realm comprises a virtual address realm definition including at least a set of users. Each user accessing the virtual address realm may have at least one virtual address in the virtual address realm and at least one physical address.

Another embodiment of the present invention comprises a virtual community network for transmitting communications

6

via at least one physical network. The virtual community network comprises a virtual address realm including a logical name assignment, a set of users capable of communicating in the virtual address realm, and a secure communication channel. In a further aspect, the virtual community network logical name is a domain name, and the domain name may be a fully qualified domain name or a virtual domain name. In another aspect, the set of users includes at least a first user coupled to a first private physical network and a second user coupled to a second private physical network, and the first private physical network and the second private physical network may have at least one common private physical subnet address. The physical addresses may be dynamic or static.

Yet another embodiment of the invention is a method comprising the steps of: defining a virtual address realm overlying a public address realm, the virtual address realm including at least a user set and a logical name for said user set; and routing communications between users in the user set by means of virtual address realm addresses. In a further aspect, this method includes the step of registering users in the user set as members of the virtual address realm. Further, the method may include the step of assigning each registered user a unique virtual address in the virtual address realm. The method may further include the steps of encrypting communications between devices in the realm, and/or applying at least one address realm group policy.

A further embodiment of the invention is a method for providing secure communications between two devices. The method comprises the steps of: providing a virtual realm identifier; defining a set of users for the virtual realm; registering users in the realm; assigning virtual addresses; and routing information between users in said virtual address realm.

A still further embodiment of the invention comprises one or more processor readable storage devices having processor readable code embodied on said processor readable storage devices. In this embodiment, said processor readable code is for programming one or more processors to perform a method comprising the steps of: providing a virtual address realm configuration including a user set and a domain name for said user set; and routing communications between users in said virtual address realm.

Another embodiment of the invention comprises a method for providing a secure virtual network. The method may include the steps of: providing a virtual network manager coupled to a public network; defining a member set of users entitled to communicate in the virtual network; registering members with the manager; assigning members a virtual address; and routing network traffic between the members in the virtual community.

An additional further embodiment of the invention may comprise one or more processor readable storage devices having processor readable code embodied on said processor readable storage devices. The processor readable code may be for programming one or more processors to perform a method comprising the steps of: managing a virtual community network realm; defining a member set of users entitled to communicate in the virtual community; registering users with the virtual community; assigning each user a virtual address; and routing network traffic between the users in the virtual community.

Another embodiment of the invention is a virtual community network system. The system includes a virtual network manager including at least one virtual community definition comprising at least a domain name and a user set; and at least one route director capable of communicating with users in the user set.

US 7,949,785 B2

7

In a further aspect, the virtual community network includes a communication agent installed on a device. The agent may comprise a virtual network adapter interfacing with the device and applications on the device to route traffic to members of the virtual community via their virtual address.

For some devices, additional software or hardware cannot be installed on a member device or it is not desirable to install such additional software or hardware on the device. For such devices, a Group Agent can be used. The Group Agent acts as a proxy for one or more members of a virtual community without requiring installation of member agent software on the client devices.

One embodiment of a system that uses a Group Agent includes communicating with a first client and acting as an intermediary between the first client and members in a first virtual address realm so that the first client can communicate in the first virtual address realm. Note that in this embodiment the first client is not configured to communicate in the first virtual address realm. For example, no software is installed on the client to add drivers, layers or interfaces specific to the first virtual address realm.

Another embodiment of a system using a Group Agent includes communicating messages with a first client and communicating the messages with members in a virtual address realm on behalf of the first client. While communicating with the first client, the messages do not use virtual addresses. During the communication process, the messages are transformed so that the messages include virtual addresses when communicating with members in the virtual address realm and the messages do not include virtual addresses when communicating with the first client.

In one implementation, the first entity resides in a first physical address realm and the second entity resides in a second physical address realm, where the first physical address realm overlaps with the second physical address realm. For example, consider the situation where the first physical address realm is a first private network using a first set of private addresses and the second physical address realm is a second private network using a second set of private addresses, where there are private addresses that exist in both the first set of private addresses and the second set of private addresses.

The present invention can be accomplished using hardware, software, or a combination of both hardware and software. The software used for the present invention is stored on one or more processor readable storage media including hard disk drives, CD-ROMs, DVDs, optical disks, floppy disks, tape drives, RAM, ROM or other suitable storage devices. In alternative embodiments, some or all of the software can be replaced by dedicated hardware including custom integrated circuits, gate arrays, FPGAs, PLDs, and special purpose computers. In one embodiment, software implementing the present invention is used to program one or more processors. The processors can be in communication with one or more storage devices, peripherals and/or one or more communication interfaces.

These and other objects and advantages of the present invention will appear more clearly from the following description in which the preferred embodiment of the invention has been set forth in conjunction with the drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

The invention will be described with respect to the particular embodiments thereof. Other objects, features, and advantages of the invention will become apparent with reference to the specification and drawings in which:

8

FIG. 1 depicts an IP packet.

FIG. 2A depicts the format of a header of an IP packet.

FIG. 2B depicts the format of a UDP packet.

FIG. 3 depicts a print-out implementation of a dynamic routing system.

FIG. 4 depicts a block diagram of an implementation of the system of the present invention.

FIG. 5A is a structural network block diagram illustrating elements used in the system of the present invention.

FIG. 5B is a logical block diagram depicting communication types in an implementation of the system of the present invention.

FIG. 6A depicts a packet structure for a communication packet in accordance with the present invention.

FIG. 6B is a table depicting the shim configuration of the packet in FIG. 6A.

FIG. 7 is a flow chart describing the overall process for communicating.

FIG. 8 is a block diagram illustrating the structure of the Member Agent application in the system of the present invention.

FIG. 9 is a flowchart illustrating the steps which occur in the creation of a virtual community.

FIG. 10 is a flowchart illustrating the process of configuring a virtual community.

FIG. 11 depicts an HTTP wrapper structure.

FIG. 12A is a flow diagram depicting a user registration process of the present invention.

FIG. 12B is a table illustrating the structure of the registration file information structure.

FIG. 13 is a flowchart depicting a member join process in accordance with the present invention.

FIG. 14 is a flowchart depicting a member leave request in accordance with the present invention.

FIG. 15 is a flow diagram depicting DNS request for a member in the virtual community in accordance with the present invention.

FIG. 16A depicts the commonly used IKE phase one aggressive mode exchange.

FIG. 16B depicts the IKE phase one aggressive mode exchange used in accordance with the present invention.

FIG. 17 depicts a packet structure of the virtual IKE phase 1 packet.

FIG. 18 depicts the IKE ticket payload data structure.

FIG. 19 is a representation of a system ping sequence in accordance with the present invention.

FIG. 20 is a flow diagram depicting a query request in the system of the present invention.

FIG. 21 depicts a flow structure for addressing and routing conducted by a Route director.

FIG. 22 is a block diagram depicting one embodiment of various components of the present invention utilizing a Group Agent.

FIG. 23 is a flow chart describing one embodiment of the process for Group Agent initialization.

FIG. 24 is a flow chart describing one embodiment of a process for initiating communication with a member that is using a Group Agent.

FIG. 25 is a flow chart describing one embodiment of a process for communicating using a Group Agent.

FIG. 26 is a flow chart describing one embodiment of a process for initiating communication by a member that is using a Group Agent.

DETAILED DESCRIPTION

One embodiment of the present invention provides for a secure Virtual Community Network or "VCN." In essence, a

US 7,949,785 B2

9

VCN is a private dynamic network which acts as a private LAN for computing devices coupled to public networks or private networks. This enables computing devices anywhere in the world to join into private enterprise intranets and communicate with each other. Thus, the invention provides a separate private virtual address realm, seen to each user as a private network while seamlessly crossing public and private network boundaries.

A basic overview of the system is shown in FIG. 4. FIG. 4 shows a computer or device B (host name—b.COb.com) in a first private domain and computer or device A (host name—a.COa.com) in a second private domain, both of which are coupled to the Internet by firewall devices 402, 404. The firewall devices are configured to implement Network Address Translation (NAT). A and B have dynamic or static private IP addresses. Computer or device X is coupled directly to the public Internet and has a public IP address.

In the present invention, a virtual community network (VCN) X.VCN is formed. Machines A, B and X can join the VCN, leave the VCN, or allow other machines in the VCN to communicate with them. More than simply a virtual private network (VPN), all members of the VCN are accessible as if they were part of a physical local network. The application running on the members of the VCN can accomplish what they would be able to accomplish if they were on the same physical routed network (e.g. all in the same private network). Membership in the VCN is controlled by an administrator via an administrative interface. In general, the administrator defines certain aspects of the VCN and then invites members into the VCN by specifying the domain name for each member. Members then register themselves with a VCN Manager, and notify the Manager via join and leave requests when they are available to participate in the community. Once registered and joined, members can communicate with other members as though connected via a local LAN.

As shown in FIG. 4, A, B and X may all make direct connections to each other through communications within the virtual domain. In FIG. 4, dashed lines represent direct communication paths seen to applications running on A, B and X, all “seeing” the virtual addresses of each other and communicating with each other using the application’s IP interface.

A hardware architecture for the machines, server or other devices used to implement the present invention should be well understood to one of average skill in the art. Suitable hardware includes one or more processors, a memory, a mass storage device, a portable storage device, a first network interface, a second network interface and I/O devices, in communication with each other. The choice of processor is not critical as long as a suitable processor with sufficient speed is chosen. The memory can be any conventional computer memory. The mass storage device can include a hard drive, CD-ROM or any other mass storage device. The portable storage can include a floppy disk drive or other portable storage device. If the computer is acting as a router, it includes two or more network interfaces. In other embodiments, the computer may include only one network interface. The network interface can include a network card for connecting to an Ethernet or other type of LAN. In addition, one or more of the network interfaces can include or be connected to a firewall. For a gateway, one of the network interfaces will typically be connected to the Internet and the other network interface will typically be connected to a LAN. However, a gateway can exist physically inside a network. I/O devices can include one or more of the following: keyboard, mouse, monitor, display, printer etc. Software used to perform the methods of the present invention are likely to be stored in mass storage (or any form of non-volatile memory), a por-

10

table storage media (e.g. floppy disk or tape) and/or, at some point, in memory. The above described hardware architecture is just one suitable example depicted in a generalized and simplified form. The present invention could include dedicated hardware, a dedicated router with software to implement the invention or other software and/or hardware architectures that are suitable.

A more detailed overview of one embodiment of the present invention is depicted in FIGS. 5A and 5B. FIG. 5A illustrates various embodiments of devices coupled to public and private networks in a manner which allows use of the present invention, and is a representation of a physical connection of such devices. FIG. 5B illustrates a logical representation of elements used to create a VCN and of the various formats of communication traffic between those elements shown in FIG. 5A.

Specifically, FIGS. 5A and 5B illustrate various user machines M_n coupled to the Internet 506, as well as other devices, described below, which implement a virtual community in accordance with the present invention. Devices M_n can be personal computers, servers or other computing devices (mobile or non-mobile). Each machine may have different communication connectivity with the Internet.

In general, an exemplary system for implementing a VCN in accordance with the invention comprises a VCN Manager 510, a Network Route Director 520 and/or Private Route Director 530, and a number of Member Agents 565. Communication between systems in various protocols is illustrated by dashed lines (UDP) and solid lines (TCP). The VCN manager and Network Route Director are shown in FIG. 5A as being coupled directly to Internet 506.

User devices M_A and M_E are connected to the Internet and have public IP addresses. These addresses may be dynamic or static. If static, M_A and M_E can easily communicate with each other in a manner well known in the art by simply addressing IP packets directly to each other. If the addresses are dynamic, then M_A and M_E require some means of knowing when the IP address of the other changes in order to communicate. As described below, the VCN system provides such a means.

Device M_B is in a private domain (e.g. private network) 540 and connected to a NAT Device 515 to access the Internet 506 by a private network path 540a. Device M_B may use have a dynamic or static IP address specific to the private domain 540. Generally, such addresses are part of the private received IP classes which are not routable via the Internet. For example, 10.0.0.0-10.255.255.255, 172.16.0.0-172.31.255.255, 192.168.0.0-192.168.255.255 are network addresses classified for private networks. Firewall/NAT device 515 translates Machine M_B ’s private IP address to a routable public IP address. NAT device 515 may include other security functions as well. M_B communicates with other non-VCN devices on the Internet via the NAT device 515, and with other members of the VCN via Private Route Director (PRD) 530, which is coupled to the private network path 540a and is a member of the private network 540. Device M_X is also shown as connected to and a member of the private domain 540 but has a static, routable public IP addresses. As such it may communicate with devices coupled to Internet 506 in a different manner, as illustrated in FIG. 5B.

Devices M_C and M_D are part of a second private domain (e.g. private network) 550 coupled to a second private network path 550a. Again, these machines have static or dynamic private IP addresses and are coupled to the Internet via a firewall/NAT device 517, which is itself coupled to network path 550a.

The VCN Manager 510 is a central server or server cluster providing management services, connection services and

US 7,949,785 B2

11

security services for the VCN. Each Member Agent **565** includes client agent software (or, in some embodiments, hardware) that runs on end machines and provides virtual network services. It enables the machine to be reached from public and private networks, and implements the security aspects of the system. In FIG. **5B**, machines M_A , M_B , M_C , M_D , M_E and M_X all include Member Agents and, therefore, can be members of a particular VCN.

The Network Route Director (NRD) **520** is a stand-alone unit that runs on the public side of the Internet enabling Member Agents and Group Agents (discussed below) to be reached inside one or more private networks from the public network. Private Route Director ("PRD") **530** is an element that runs inside a private network enabling access to machines inside the private network from machines outside the private network. It can run as a service on a platform offering other services such as a Windows Server.

Generally, VCNs are defined and managed by the VCN Manager **510**. Clients utilize Member Agents or Group Agents to access other client machines (M_n), in private or public realms. The NRD or PRD routes communications to Member Agents.

As described herein, three forms of communication are used in creating and implementing the VCN. TCP is used in the registration and joining processes; encapsulated UDP packets are used between elements in establishing communication between joined devices and for management traffic, and Encrypted UDP packets are used for agent-to-agent communications.

FIG. **5B** shows exemplary traffic between elements of the VCN. A first type of traffic is TCP traffic. TCP traffic is used between member agents to establish membership in the VCN (the registration process), and make the VCN manager aware they are available to communicate with other members of the VCN (the join process). In FIG. **5B**, this is represented by solid lines **513b**, **511b**, **523b** and **543b**. TCP is also used: by the PRD **530** to establish communications with the VCN (lines **527b**, **527d**); and by the member agent of device M_B to communicate with the PRD **530** (line **533b**). TCP may also be used by devices M_C and M_D on private network **550** to communicate with each other.

Encapsulated UDP traffic is used by elements of the VCN once membership has been established and the elements are joined in the VCN for management and administrative communications. This is represented in FIG. **5B** by lines **511a**, **513a**, **523a**, **527c**, **523c**, **527e**, **521**, **543a**, **541**, and **519**. Encrypted UDP communications are used between devices, shown in FIG. **5B** between device M_X and M_A at **525** and device M_B and M_C at **537**.

In one embodiment, secure connections are built dynamically using IPsec tunnels based on a virtual IP space that can traverse from private networks, across the Internet and into other private networks. Strong authentication and the establishment of IPsec policies at the time members join the community offers privacy and security to the members of the virtual community. This facilitates the introduction of a variety of policy-based network services with centralized management such as Virtual Private Networks for intranets and extranets, IP-telephony domains, and IP-based PDA communities.

Communication Protocol

In the present invention, an encapsulation routing protocol allows packets to traverse addressing boundaries and identifies routing endpoints by unique DNS domain names. It employs an address virtualization scheme that accomplishes backwards compatibility with IPv4 applications. With certain

12

optional exceptions, UDP encapsulation is used between elements of the Virtual Community.

One example of an encapsulated packet **600** is depicted in FIG. **6A**. It includes a 20 octet IP header **602**, an 8 octet UDP header **604**, a 20 octet system shim **606**, and a variable size virtual IP packet **608**. A virtual IP packet has the same format as a standard IP packet.

Members of a VCN communicate to other members of that VCN by means of virtual IP Packets **608**. Virtual IP addressing is used to establish unique connection identities for community connected devices. IPsec can (optionally) be applied to the Virtual IP Packet for end-to-end security. Virtual IP addresses are used to identify members in the community in the virtual packet and are not directly routable. Rather, they are unique identifiers of a given connection to the upper layer application. In one embodiment, virtual IP addresses are in IPV4 format.

In general, the UDP header is set for a default port. System shim **606** carries enough addressing information to allow the Route Directors to forward the tunneled packet through the various address realms on its way to the destination. Such a routing methodology is often referred to as source routing.

In order to route packets to a peer in a different addressing realm, the protocol stack knows the address of the appropriate Route Director that serves the peer's realm, the private address of the peer or a NAT address. The Member Agent obtains the information by using the VCN Manager as a DNS authority for DNS lookup operations as described below (FIGS. **13** and **21**). The DNS response from the VCN Manager will include the above-identified information in an optional text field of the response to the Member Agent.

TCP/IP connections require IP addresses to be used as connection identifiers at the user level. If an application wishes to make a connection to "m3.abcdef.com" and another to "m4.abcdef.com", it resolves both DNS identifiers into IPv4 addresses. The system cannot assign the private IP addresses of the members to be unique connection identifiers, because two members might have the same private address in different private address realms.

Assigning a virtual address to each peer designated by a DNS string solves the problem of ambiguous endpoints. These virtual addresses may be any legal IPv4 addresses, but conflicts will occur if the same addresses are used to directly designate a peer in an IPv4 network application. For instance, if 10.0.0.10 is used as a virtual address, and is also used on a member's network as the mail server, packets that are ordinarily destined to the mail server may be redirected to a random peer. In general, when choosing virtual addresses it is best to choose addresses that will not be routable to the member. Hence, each member is assigned a virtual IP address unique in the context of the virtual community and that will not conflict with any local addresses on its LAN or any public IP addresses on the Internet.

By having each member of a VCN use a virtual address, a virtual address realm is created. The virtual address realm is the set of addresses that can be used to identify and send communications to other members of the VCN. The virtual address realm is different from a physical address realm because the physical address realm is actually used to route on a physical network. The members of a VCN will all be in the virtual address realm of that VCN, but may be in different physical address realms. For example, looking at FIGS. **5A** and **5B**, M_B and M_C are in different physical address realms (e.g. **540** and **550**); however, they are both in the virtual address realm of the VCN. Another example can be seen with FIG. **4**, where A and B are in different physical address realms (Cob.com and Coa.com); however, they are both in the same

US 7,949,785 B2

13

virtual address realm. Applications on members of a VCN will see the VCN as the local subnet; therefore, the VCN can be thought of as a virtual subnet. Each member of the VCN will appear to the application as an entity connected to the local subnet.

Route Directors **530** and **520** facilitate routing UDP encapsulated traffic to private address domains **540**, **550**. In general, Route Directors either have public IP addresses or addresses that are statically remapped to a public IP address. The static translation of a Route Director's IP must leave the port in the UDP header unchanged to allow the UDP encapsulated destination port number to be directly used by public members, by other Route Directors, and by private members. Route Directors **520**, **530** and VCN Manager **510** may be configured to use a non-default UDP encapsulation port. In one embodiment, all outgoing messages are sent to this port on the next destination node and all incoming messages are read from this port. As discussed below, information in a "join" packet tells Member Agents the UDP encapsulation port to use in communications to Route Directors, network managers, and other members of the community.

System Shim **606** (see FIG. **6A**) has a format given by way of example in FIG. **6B**. Protocol Version information, data type, header option flags, and Router Information are potentially stored in the Shim. (The abbreviation "gIP" stands for "global" or public IP address, indicating an Internet routable IP address). The Data Type field indicates the meaning of the optional data that may follow the Shim. This may include a Virtual IP Packet **608**, a ping to a Route Director, a request for a Network Route Director virtual address, a Route Director response with a virtual address, a member leaving and releasing a virtual address, and a member-to-Route Director packet. The shim also includes the public IP address for the source machine's Route Director, the public IP address for the destination machine's Route Director, the private IP address for the source machine and the private IP address for the destination machine. In the case where the source or destination use a public IP address, then the shim will not need to include the private addresses of the source/destination or a route director.

The system can optionally use TCP encapsulation between a member and a Route Director. If the VCN Manager determines that a Member Agent (e.g., Agent software on a member machine) and a Route Director are both able to support a TCP connection, then it may instruct the member to initiate a TCP connection with the Route Director. TCP port **80** may be used as the destination port for all TCP encapsulated messages to the Route Director, and TCP port **20202** may be used as the source port for all TCP encapsulated messages from a private member to the Route Director.

As noted above, Network Route Directors (NRDs) **520** facilitate routing traffic into and out of private address domains without requiring reconfiguration of a firewall **517** protecting the domain **550**. As in UDP encapsulation, a special field in the system join packet (explained below) tells the Member Agent which TCP encapsulation port to use as the source port. The NRD will then re-encapsulate virtual-traffic in UDP-encapsulation before passing it on to other elements of the virtual network.

Note that more than one user can use the same device, with each such user having different virtual addresses in the same or different address realms. Additionally, regardless of the number of users, one device can be in multiple virtual address realms. In some embodiments, a user can have multiple identities, each in a different address realm.

FIG. **7** is a flow chart describing the overall process for communicating between members of a VCN. Any member of

14

a VCN can communicate with any other member of a VCN regardless of whether either of the members are behind a firewall, behind a NAT device, directly accessible via the public Internet, in a private network, etc. From the point of view of an application on a member device, all the other members appear to be on the same LAN. Thus all applications running on a member's device will think that all other members of the VCN are on the same LAN and in the same address realm (e.g. the virtual address realm of the VCN).

In step **650** of FIG. **7**, the Agent for the potential member joins the VCN and receives a virtual IP address. In order to establish or participate in communication within the VCN, a device must register with and join the VCN, thereby becoming a member of the VCN. The Agent will send a message to VCN Manager **510** and receive a virtual IP address back. The virtual IP address received is used to identify the member in the VCN. More details of step **650** as well as other portions of FIG. **7** will be discussed below. Note that there is a dotted line between step **650** and step **652**. This dotted line signifies that a lot of time can pass between step **650** and step **652**. Additionally, after step **650** is performed, it does not need to be repeated.

In step **652** of FIG. **7**, an application on the source machine (the member initiating communication) intends to send a message to an application on the destination machine (another member of the VCN). Examples of applications that can communicate on the VCN include instant messaging, email, FTP, browsers, data transfer programs, and other applications that can communicate. Consider an example when application on device M_A intends to initiate communication with an application on device M_B . Thus, M_A is the source device and M_B is the destination device. In step **654**, the application on M_A will initiate a DNS request using the domain name for M_B . This DNS request is sent to VCN Manager **510**. In step **656**, VCN Manager **510** returns the public address of the Route Director for the destination, the private address for the destination device and a virtual IP address for the destination device. In the example described above, step **656** includes returning the public IP address for private Route Director **530**, the private address for M_B and a virtual IP address for M_B . If the destination machine has a public IP address (and, thus, does not use a routing director) then step **656** will only include returning the public IP address for the destination. If the destination machine is in a private network that uses a Network Route Director (e.g., NRD **520**), then step **656** will return the public IP address for the Network Route Director.

In step **658**, the Agent for the source device, which received the information in step **656**, will store that received information. This information can be stored in a table, or any other data structure. In step **660**, the Agent for the source device will return the virtual IP address of the destination device to the application on the source device. In step **662**, the application on the source device will initiate the sending of a message/data using the virtual IP address for the destination device. In step **664**, a virtual IP packet will be created. The source IP address for the virtual IP packet will be the virtual IP address for the source device. The destination IP address for the virtual packet will be the virtual IP address for the destination device. The virtual IP packet will be in the same format as a standard IP packet. In step **664**, the newly created virtual IP packet will be subjected to IPsec (or another security means). In some embodiments, IPsec is not used. In step **670**, the virtual IP packet is encapsulated. For example, in one embodiment, the virtual IP packet is encapsulated as described above with respect to FIGS. **6A** and **6B**. This encapsulation will include shim **630**. As described above, shim **630** will include the public IP addresses for the Route Directors

US 7,949,785 B2

15

for the source and destination, and the private IP addresses for the source and destination devices. If either the source or destination do not have a Route Director, the shim will not need to include that information. For example, when the source machine is M_A , there is no Route Director for the source. If the source machine was M_C , then the shim, would include the public IP address for Network Route Director **520**. If the source is M_B , the shim would include the public IP address for Private Route Director **530**.

At this point, the path of the packet will depend on whether the destination is using a Private Route Director **530**, Network Route Director **520** or has a public IP address. If the destination has a public IP address, the process moves to step **678** where the encapsulated packet is forwarded to the destination agent.

If the destination is in a private network that has a PRD, then in step **672** the encapsulated packet is sent to the NAT device for the destination. For example, the encapsulated packet is sent from M_A to NAT **515**. In step **674**, the encapsulated packet is sent from the NAT device for the LAN that includes the destination to the Route Director for the destination. For example, the encapsulated packet is sent from NAT **515** to private Route Director **530**. In step **676**, the Route Director which received the encapsulated packet will remove the virtual IP packet and the shim from the encapsulation. The Route Director will read the shim to learn the local IP address to send the packet to. The shim and virtual IP address will then be re-encapsulated into a new packet with the original virtual packet. The new outer packet includes a destination IP address equal to the private IP address of the destination device. The encapsulated packet received in step **674** will include a source IP address equal to the public IP address for M_A (or the source's Route Director) and a destination IP address equal to the public IP address for the Route Director of the destination (e.g., PRD **530**). In step **678**, the new encapsulated packet will be sent from the Route Director to the Agent for the destination device.

If the destination agent is in a private network and is using a Network Route Director, then following encapsulation the encapsulated packet is sent to the Network Route Director for the destination agent in step **673**. At step **675**, the NRD will remove the virtual IP Packet and shim and re-encapsulate them in a UDP packet for transmission through a NAT device to the destination via a persistent UDP connection between the NRD and the destination. At step **677**, the encapsulated packet is forwarded to the NAT device (via the persistent UDP connection) behind which the destination Agent resides, which will perform its own routing of the encapsulated packet to the destination Agent in step **678**.

In step **680**, the Agent will remove the shim and the virtual IP packet from the encapsulation. The virtual IP packet will be removed from IPsec (e.g. decrypted). In step **682**, the information from the shim will be stored by the Agent. In step **684**, the contents of the virtual packet will be presented to the application on the destination device. That is, the application on the destination device will receive the contents of the data payload, or at least a portion of the data payload.

When the destination wishes to respond to the source or the source wishes to send additional information, the process of FIG. 7 can be repeated. However, when repeating the process of FIG. 7 between two machines where communication is already established, the steps of performing the DNS request do not need to be performed again. That is, when the destination wishes to respond, the destination will perform the steps in FIG. 7 starting at step **662**. The Agent for the destination device will already know all the information to put in the shim. Additionally, when the source device wishes to send

16

additional information to the destination device, the source device can start FIG. 7 at step **662**. In some embodiments, however, the steps of performing the DNS request will be performed again.

5 Member Agent

As noted above, devices in the VCN utilize a member agent installed on the device to communicate. One unique aspect of the system of the present invention is the integration of the agent software in a typical device.

10 FIG. 8 illustrates the functional flow diagram of an installed agent on a device such as a computer, server or other device. Region **810** illustrates the "user space," and region **820** represents the kernel area of the components interacting with the host device.

15 Applications **830** interact directly with an IP stack **835** which is built on a deterministic network extender or DNE **850**. The deterministic network extender is provided by Deterministic Networks, Inc. The network extender allows one to implement any driver level functions such as IPsec, browsers, traffic redirectors, and the like, through a series of plug-ins. As shown in FIG. 8, a number of plug-ins, including a DNS plug-in **852**, an IPsec plug-in **854**, a virtual adapter manager **856** and a domain name router **858** (see U.S. Pat. No. 6,119,171, incorporated herein by reference) interact with the deterministic network extender **850**. The DNS plug-in processes DNS queries by applications via the stack. The virtual adapter manager allows for additional adapters to co-exist in the same device.

20 The DNE is an NDIS compliant module (in Windows environments) which appears as a network device spire to the protocol stack **835**. The DNE supports the TCP/IP and UDP protocols and various adaptor types. In essence, this forms a virtual network adaptor in device installations. This means that various configurations for each virtual community can be provided for each DNE. This provides the advantage that a number of members or users can utilize the same device, with the device installing a different virtual adaptor using plug-ins for the deterministic network enhancer and each user does not need to reconfigure the network settings of the machine to join each community. Each plug-in is constructed in accordance with standards promulgated by the DNE provider.

25 As shown in FIG. 8, user applications interact directly with the IP stack of the virtual adapter. Each adapter provides its own stack to the device's applications. To the applications, each stack appears separate, and no separate routing of the application is required. For example, the IKE user application **825** can interact with the domain name plug-in, the IPsec plug-in, the virtual area management communicator **856**, and the security policy manager **860**. The Adapter Manager **856** allows parallel stacks for each VCN to be implemented and used on each machine. This means a user and device can belong to several VCNs without needing to re-configure IP settings to access each VCN.

VCN Definition

30 FIG. 9 shows the steps which occur at the VCN Manager to define and establish a virtual community. At step **902**, an administrator sets up a virtual community by providing information to the VCN Manager via a user interface (not shown). It will be understood that the user interface and the method of providing information to the VCN Manager is not critical to the system of the present invention. One particularly advantageous interface is implemented in a World Wide Web browser. The interface may be implemented using HTML, XML, Java Applets, Java Server, Active Server Pager or any of a number of well-known Web technologies. The particular information required to set up the community network is illustrated in FIG. 10. Next, at step **904**, users who are desig-

US 7,949,785 B2

17

nated by the administrator as being allowed access to the virtual community defined in step 902 are notified by any number of appropriate means. In one implementation, the users are notified by e-mail, although other forms of notification, such as verbal or personal notification, may be used. The e-mail may contain an invitation to join the community, community authentication information, a link to a location to download the Member Agent, and/or a link to the Group Agent. Next, at step 906, using information contained in the e-mail, users register with the VCN Manager. Registration in this context means registering a machine using a Member Agent or a Group Agent with the VCN Manager. Users may be required to install the Member Agent software. The link in the notification e-mail may provide the user with a location for downloading the Member Agent software as well as instructions for installing the software. Other information, such as system password information, is provided in the e-mail. Finally, at step 908, management of the community is handled by the VCN Manager. This includes monitoring which registered users have joined and left the system, answering DNS requests, and applying security policies, as described herein. This may include notifying new users and registering those users as the definition of the VCN changes.

FIG. 10 is a flowchart illustrating the process of setting up a virtual community which is performed by an Administrator. Initially, the Administrator sets up a virtual domain name and address domain at step 1002. This information is used by the VCN Manager and Route Directors to allow members to communicate. Next, users are defined at step 1004. In one embodiment, the users are defined by providing the domain names of each of the users who will be invited to participate in the VCN. At step 1006, default passwords are set. This is the password a user first uses to access the system. The user may be forced to change the password for security reasons at a later date. Finally, at step 1008, policies may be defined at the domain level, the user level, or for individual machines in the domain. Policies are used to allow or deny access to individual machines, services, or other users.

Member Registration

The registration process is the first step that a user and machine go through with the VCN Manager before becoming a member of a virtual community. There are two phases to member registration: a registration request and a registration file request. Registration happens outside of the IPsec tunnels because at registration time no IPsec tunnel exists for the user registering.

In one implementation, all traffic between Member Agents and the VCN Manager (with the exception of Registration and Join function traffic is encapsulated using UDP encapsulation, represented by lines 511a, 513a, 523a, 519, 521, 527a, 527c, 537, 523c, 527e, 541, 543a, 533a of FIG. 5B). This encapsulated traffic will carry management traffic between the members and the VCN Manager. A service port is reserved at the VCN Manager in the virtual network for the establishment of a TCP connection between a member and the VCN Manager. A member that joins a VCN is informed of the service port as part of the join protocol. A DNS port is also open on the virtual side of the VCN Manager. In one embodiment, the VCN Manager virtual ports are TCP Port 9001 and DNS UDP Port 53. The packets used for the initial Registration and the subsequent Join operations use TCP port 80 outside the virtual network. This is represented in FIG. 5 by the solid lines 513b, 511b, 523b and 543b.

VCN system protocol packets that are sent outside the system tunneling are HTTP wrapped and sent to the VCN Manager on TCP port 80 to facilitate NAT traversal. More information about NAT traversal will be provided below.

18

Each packet includes a header and payload. For compatibility with firewalls, the HTTP wrapper makes the packet a SOAP-compatible XML document. Including a binary data portion after XML is consistent with Direct Internet Message Encapsulation (DIME), a proposed lightweight, binary message format that can be used to encapsulate one or more application-defined payloads of arbitrary type and size into a single message construct.

The format of the HTTP wrapper from the Member Agent to the VCN Manager is shown in FIG. 11. The wrapper includes an HTTP HEADER, XML HEADER and P-LENGTH fields. The HTTP header includes a POST HTTP field, an Identity field, a Content Type field, and a Content Length field. The XML header includes version information and packet type information. The Header includes error codes and explanations, as well as the content type and length.

The Registration exchange between the member agent and the VCN Manager is shown in FIG. 12A. When a Member Agent 565 attempts to register at step 1200, Member Agent 565 will send a registration request packet 1202 to the VCN Manager 510 to start registration. The packet is included in an HTTP wrapper as illustrated at 1204. The registration packet will carry the client Fully Qualified Domain Name (FQDN) and a Diffie-Hellman Key Exchange request to the VCN Manager. The packet further includes packet version information, type and length information, a user authenticator, the length of FQDN in octets, a VCN name offset, the member's FQDN in DNS format, the length of Diffie-Hellman value in octets, and the member's initial Diffie-Hellman value.

An Initial Password is passed to the user through an outside channel, such as e-mail (or other channel). In a setting where security is a concern, an alternative method such as personal communication or encrypted e-mail may be used.

The registration authenticator is computed using the first sixteen bytes of the HMAC-SHA-1 applied to the packet data with the password hash as the key. (Normally, the HMAC-SHA-1 algorithm generates a 20-byte result; in one embodiment of the system only the first (leftmost) 16 octets are used in the MAC.) The formulas are as follows:

$$\text{HMAC-Key-Field} = \text{SHA-1}(\text{Initial Password})$$

$$\text{Authenticator} = \text{HMAC-SHA-1}_{16}(\text{HMAC-Key-Field}, \text{Data})$$

The VCN Manager will respond in one of three ways 1206, 1208, 1212, to the registration packet 1202. VCN Manager 510 will acknowledge (ACK) the packet at step 1212 with the Registration Acknowledge (ACK) packet 1214 if the member belongs to a community but its status is "not registered." The VCN Manager also sends a phase-2 confirmation code to the member's e-mail address and changes the member status to "registering." The packet will transfer a "Time To Live" value to the member which is a timeout period in seconds for hearing back from the member with the confirmation code. Phase two of registration must be completed in this time period.

If the packet is abnormal (in size, FQDN length, bad type, etc.), the authenticator check fails, the member does not exist in the database, or the VCN server is not running, the VCN Manager drops the packet in step 1206. The packet will be not-acknowledged (NACK) 1210 if the member is already in the "registered" state, the send e-mail operation fails, the process of updating the member to the "registering" state fails, or the random generator fails in attempting to generate a confirmation code in step 208.

Upon receiving the Registration ACK 1214, the user will send Registration File Request Packet 1220 to the VCN Man-

US 7,949,785 B2

19

ager **510** to start registration. This packet will transfer the following information to the VCN Manager: a registration file User Authenticator, the length of FQDN in octets, a VCN name Offset, and the member's FQDN in DNS format.

The registration file user Authenticator contained in the File Request packet **1220** is slightly different from the Registration Request Authenticator in packet **1202**, in that the Initial Password, passed to the user through an outside channel (e-mail for example), and the Confirmation Code, a temporary code sent in the second e-mail with a timeout to the user, are used:

HMAC-Key-Field=SHA-1(Initial Password)

User Authenticator=HMAC-SHA-116(HMAC-Key-Field,SHA-1(Confirmation Code)+Data)

The VCN Manager will send a Registration File acknowledgement Packet **1230** to the member **565** in response to the Registration File Request packet **1220** in step **1228** if it is determined that the member belongs to a community. The Acknowledge Packet **1230** will transfer the following information to the member:

Crypto Type—a code indicating how the rest of the packet is encrypted. This is provided in a crypto type field preceding the registration file.

Registration File—a structure with a number of fields containing the information required by the member to successfully perform a join. It is sent encrypted using the key generated via the Diffie-Hellman key exchange.

A table representing the Registration File structure is shown in FIG. **12B**. As shown therein, the Registration File contains an Authentication Type, a File Type, a File Length definition, a Token Key, a Key Timestamp, a Key Type, a Key Length, a High Availability (HA) Version, a HA Number, Server Information and Padding. The Token key is the shared secret with the server. The Token Key is identified by the Key Type, which is normally a Diffie-Hellman type in one embodiment. The VCN Manager also changes the member status to “registered.”

If at step **1224**, the member agent status is not set at “registering,” or the timeout for phase 2 has expired, a NACK packet **1226** is sent. If the packet is abnormal (in size, FQDN length, bad type, etc.), the authenticator check fails, or the user does not exist in the database, or the VCN server is not running, the packet is dropped in step **1222**. While current encrypting method for the file is to generate a key/IV pair from the combined Diffie-Hellman value using a Key Derivation Function, any number of suitable substitutes may be used.

In addition to the registration authentication described above, the system uses a packet authenticator to verify communications between the VCN Manager and Agents. The Authenticator is a cryptographically strong hash of the packet keyed by a shared secret between the source and destination. The authenticator is calculated over the entire packet using the HMAC-SHA-1 algorithm. The inputs to this Authenticator are not available until after Registration. As illustrated above, during registration an Authenticator based on the user password is used. The fields used in the calculation are a Diffie-Hellman Key comprising a multi-octet random string that is the result of the Diffie-Hellman key exchange protocol, and the entire packet data, with the Authentication field zeroed out.

The Authorization Key, AuthKey, described above, is derived from Diffie-Hellman using the KDF2 function, and used for general packet Authentication. AuthKey=KDF2 (Diffie-Hellman Key)MAC=HMAC-SHA-116(AuthKey,

20

Data). The AuthKey need be calculated only once per registration from the KDF2 of the DH key and then stored for later use. It is known as the Authentication key and is 20 octets long.

Member Join/Leave

Once a user is registered, the user can join the VCN. Once joined, the user is a member of the VCN. Joining allows the user to let the VCN Manager and other members know that the user/member is on-line and available to communicate. The Join protocol runs between a Member Agent and the VCN Manager to establish a session key with the Manager, authenticate to the Manager, and make use of connections via the Route Directors in the community.

The Member Join process is illustrated in FIG. **13**. There are two phases to the Join operation: phase one is an Initialize Request and phase two is the actual Join. At join time, the IPSec tunnel for secure communication is not established, so the first few join packets are sent outside the IPSec tunnel.

Initially, at **1302**, the Member Agent **565** will send an Init Request Packet **1304** to the VCN Manager **510** to start authentication. The packet will carry the following information to the VCN Manager:

Version Variant Mask—used to indicate optional variations of the protocol from the member to the VCN;

Token Timestamp—identifying when a Token Key was created (in seconds);

Member seed—a random number to thwart so called “man-in-the-middle” attacks;

Member Local IP—the member's Local LAN IP address; and

Client FQDN—the client's fully qualified domain name.

Again, three responses **1306**, **1308**, **1312** are possible. At step **1306**, if the packet is abnormal, it will be dropped. Similarly, if the member fails authentication, at step **1308** a NACK packet **1310** will be sent.

If the member passes authentication at **1312**, the Manager will send an acknowledgement packet **1314** to Member **565**. An AuthKey is also transmitted as part of the INIT Request Packet, and the Manager authenticates the packet by checking this against a secure key digest. The acknowledgement packet **1314** packet will transfer the following information to the member:

Version Variant Mask—version information;

Member Seed—from the Init Request to protect against playback attacks;

Manager Seed—a unique seed used to thwart man in the middle attacks;

Session AH Key—used in IPsec;

Session ESP Key—used in IPsec;

Ink ACK Time—a time stamp used to tie this packet to the following Join Request. There is a timeout if the Join is delayed too long;

Authentication Method—the method used for authentication;

Route Director Private IP—if member is in a private network, this is the private IP address of the Private Route Director. The value is zero if no Route Director is used;

Route Public Public IP—if member is in a private network, this is the public IP address of the Private Route Director. The value is zero if no Route Director is used;

Tunneling Port—used for UDP encapsulation Route Director Flags. Only Bits **0** and **1** currently used to direct the agent to a Private Route Director, Network Route Director with UDP, or a Network Route Director with TCP.

At step **1312** of the join process, the VCN Manager will also determine whether the Member Agent attempting to join

US 7,949,785 B2

21

is behind a NAT device. The method by which this occurs is discussed below with respect to FIG. 21 and the results of this determination may force the Agent to negotiate a session with a Network Route Director which will provide a pseudo address value used as the Agent's local IP value in the Join request, discussed below. This process is discussed in depth with respect to FIG. 21.

Following receipt of the acknowledgement packet, at step 1316, the member will send Join Request Packet 1318. The packet's payload includes a manager seed, the initial ACK time, join flags, the member machine's local IP address (or pseudo address), the Route Director's private IP address, the Route Director's public IP, the member AH SPI and ESP SPI, the member listen port, a token timestamp, a GUID identifying the current client run, NetBIOS name support, a password hash, version information for high availability and security policy information.

In step 1320, the VCN Manager will drop this packet if it is abnormal. In step 1322, the VCN Manager will send a NACK packet 1324 if the member is not in the registered state, the timeout for Join from INIT ACK has expired, or updating the member to the Joined state fails.

If the join action succeeds, then in step 1326 the VCN Manager will send an acknowledge packet 1328. The acknowledge packet payload includes the RMI Port at the VCN Manager, the Tunneling Port, VCN Manager Authentication Header (AH), Security Parameters Index (SPI) for IPsec, VCN Manager Encapsulating Security Payload (ESP) SPI for IPsec, the UTC Time (in seconds) when the member joined, VCN Manager Heartbeat interval (in seconds), the Member Heartbeat interval (in seconds), the VCN Manager's Virtual IP address, the Manager Subnet Prefix Length, the VCN Member Subnet Prefix Length, the Route Director's private IP address, the Route Director's public IP address, an indication of the updates that follow, an update of the Token Key, an update of the Security Policies, an update of High Availability indication, and padding. The member has now joined the community at 1330. Once the member has joined, all communications can be encrypted using IPsec.

The system of the present invention also includes RADIUS authentication support. Generally, if a member attempts to authenticate using a RADIUS server, the VCN Manager will perform the same steps as when it receives the Join Request packet except for having a RADIUS server authenticate the member. The payload for this packet is similar to the Join Request packet except that it has an additional field for RADIUS password that is used by the RADIUS server to authenticate the member.

FIG. 14 shows the sequence for leaving a VCN. Initially, at step 1410 when a Member Agent is prepared to leave the community, a Leave Request Packet 1412 will be sent to the VCN Manager 510 to start the leave process. The Leave Request Packet will carry a Leave Flag, the FQDN Length, FQDN VCN Offset and Member FQDN. Unless dropped at step 1414 or a problem exists at 1416 which would generate a NACK packet 1418, the VCN Manager will send the Leave ACK Packet 1430 to the member in step 1420. This packet has no payload. At step 1440, the member has successfully left the community.

DNS/Lookups

For one member to communicate with another member, the initiating member's agent will perform a query or DNS lookup (step 654 in FIG. 7) to determine the address of the other member, and then establish a secure communication channel between them.

The Member Agent utilizes the VCN Manager to perform lookups of other virtual members in the virtual community

22

using DNS. The DNS operation is conducted using the DNS protocol with the VCN Manager. The member software sends a DNS query to the VCN Manager, and the manager returns information about the target member. Besides the target member's virtual IP address, the DNS response also contains additional information about the target member. While the VCN Manager accepts all DNS requests, it will only acknowledge the DNS requests of type A, which are targeted to virtual domain names in the secure virtual communities of the VCN Manager. The VCN Manager is the authoritative name server for those virtual domain names; other requests will simply be responded to with a "Server Failure."

The DNS query and response sequence is illustrated in FIG. 15. At step 1510, a member will request a DNS lookup by sending a DNS request 1520 to port 53 of the VCN Manager. The DNS request 1520 will result in a normal DNS response 1575 unless any of a number of failure conditions are met, shown at steps 1522, 1540, 1550, and 1560.

If the Manager receives a query on port 53 that is not a DNS query, then at step 1522, it is dropped. If, at step 1524, the query is not a standard query, such as a DNS STATUS (a status lookup) or IQUERY (inverse DNS), or the class of the DNS query indicates an Internet (IN) query, or if the TYPE value of the request is a pointer (PTR) or host address (A), then, a "Not implemented" DNS response 1530 is sent over the virtual link to the Member Agent 565.

At step 1540, the VCN checks to see if the source IP address is for a member of the VCN. If the source IP address is not for a member of the VCN, the virtual IP look up fails or the type PTR query does not match, a "server failed" response 1545 is returned. At step 1550, if there is no DNS Question section or there is another type of format error with the packet, a "format error" response 1555 is sent. The VCN determines whether the member has joined the VCN, at step 1560. If not, a name error response is sent at 1565.

If the name is a successful lookup (step 1570), then the VCN Manager will send a DNS Response Packet 1575 to the member 565. The response includes the following information: the target member FQDN; the target member virtual IP address; a source Route Director flag, the source member virtual IP address; the target member join time; the target member agent version; the target member private IP address; the target Route Director public IP address; the target member NetBIOS name; the shared key, shared by the two members; a target Ticket length; a target member Ticket encrypted for target member only and containing a Ticket issue time, Ticket expiration time, and shared key; the target member virtual IP address, the source member virtual IP address, the source member private IP address; the source Route Director public IP address, the source member FQDN; and padding (to obtain a multiple of 16 octets). The Ticket provided by the DNS request is detailed further in the description of FIG. 16B.

The target member's virtual IP address will be presented using an A-type resource record in the answer section of the normal DNS response packet. All the remaining information will be transferred using a TXT-type resource record in an additional section of the DNS Response.

Encapsulation

Traffic between Member Agents in the VCN is encapsulated using UDP encapsulation. Encapsulated traffic between members can carry any IP traffic from any standard IPv4 compliant application. However, one port is set aside in the virtual network at the application layer to establish a connection between two members in the community. In one embodiment, this is UDP port 500, which is set aside for Internet Key Exchange (IKE), allowing negotiation of an IPsec tunnel between the two members.

US 7,949,785 B2

23

When an application on a member device wishes to open a connection to a peer on a target member device, a protocol is followed for the establishment of the IPsec session between the two. Member Agents are configured to listen on the UDP tunneling port (which defaults to port **20202**) for virtual traffic. As an accepted procedure non-IPsec traffic on the virtual IKE port **500** is accepted and passed to an IKE and Security Policy Module to allow for the possibility of initiating, an IKE exchange and the establishment of an IPsec session between the two members.

IKE is fully described in RFC-2409. IKE provides a key exchange with a peer and creates a security association (SA) for the IPsec module. IKE access between peers occurs in two phases: phase 1 establishes a secure, authenticated, channel with which to communicate, and phase 2, which negotiates a Security Association on behalf of IPsec. Phase 1 is generally referred to as the ISAKMP Security Association (SA) and includes a "Main Mode" and "Aggressive Mode." In one implementation, Aggressive Mode is used for phase 1. One of the allowed phase 2 exchanges is "Quick Mode" and in one implementation, Quick Mode is used in the system for phase 2.

FIG. **16B** shows the aggressive mode IKE phase one exchange utilized in the present invention. FIG. **16A** shows the standard aggressive mode IKE phase one exchange for comparison. In these figures, in accordance with standard IKE parlance, one device is considered an exchange "initiator" and the other a "responder".

In general, in phase one of the IKE exchange, peers are authenticated, an IKE Security Association (SA) policy between peers is established, a Diffie-Hellman exchange is performed to produce matching shared secret keys and a secure tunnel to negotiate IKE phase two parameters is established.

In FIG. **16A**, an initiator exchanges a header (HDR), SA negotiation payload, Key Exchange (KE), nonce payload (Ni, where "i" indicates an "initiator" payload) and identity of the party (ID_i). The key exchange payload contains the public information exchanged in a Diffie-Hellman exchange. The responder returns the HDR, SA, KE, a responder nonce payload (Nr), its identity (ID_r) and the HASH responder (HASH_R) payload, which is generally a hash of the ID payload and is used to authenticate the exchange. The initiator returns the HDR and HASH initiator (HASH_I) payload, and the initial secure tunnel is established.

In accordance with the present invention, FIG. **16B** illustrates the use of an extension to aggressive mode exchange. This is performed by adding a ticket payload (TKT*) to distribute a shared-key to remote peers.

The Ticket is the means by which one Agent can identify whether another Agent is a trusted member of the virtual community. Essentially, the Ticket includes information identifying the initiating member to the receiving member.

The ticket is issued by the VCN Manager and encrypted for the receiving member with a member-server shared secret; only the receiving member can open it. The ticket contains a time-stamp, a sequence number and a shared-secret used for IPSEC between the two members. To communicate with the receiving member, the initiating member obtains the ticket from the VCN Manager (through, for example, the DNS sequence, described above) and passes it to the receiving member in a payload that can ride on an IKE packet. The receiving member receives the Ticket, decrypts the Ticket using the member-server shared secret, and uses the information in the Ticket to establish secure communication with the initiating member.

24

This payload, TKT, is formatted as shown in FIG. **17**. The packet **1700** contains an IP header **1710**, UDP header **1720**, System shim **1730**, and Virtual IKE Phase 1 UDP/IP packet **1740**. FIG. **18** illustrates the structure and data of an exemplary first packet **1740**. The IKE Ticket Payload in the first IKE packet contains information on the payload and the payload data, including the type of payload following next and the ticket payload data **1756**. The payload data **1750** includes the size, number and item list **1766**. The item list **1760** includes type, size and item information. The IKE ticket payload data **1740** contains the item types

The item types **1770** are shown in the last table of FIG. **18**: "My FQDN"—a null terminated FQDN in label dot format. (For example, a human readable FQDN "bob.system.com" is encoded as "bob.system.com\0x00.") My Ticket data which includes a Manager Current Time (UTC of Manager as known by initiator); and an Initiator Virtual IP (Obtained at Join time) Target Virtual IP (Obtained from Manager) and Padding. The Peer FQDN Format is a null terminated FQDN in label dot format. For example, a human readable FQDN "bob.system.com" is encoded as "bob.system.com\0x00". Finally, the Peer Ticket data includes: a Ticket Issue Time (UTC (seconds) on VCN Manager); a Ticket Expiration (UTC (seconds) for ticket expiration); a Shared Key used by the two Members for IKE; a Target Virtual IP Address; the Initiator NAT Port; the Initiator Subnet; the Initiator Virtual IP Address; Initiator Private IP Address, the Initiator Public IP Address, the Initiator FQDN and padding.

The Diffie-Hellman key exchange protocol is utilized in the Internet Key Exchange (IKE) protocol that precedes IPsec. The prime and generator value are taken from a well-known Internet Draft entitled, "More Diffie-Hellman groups for IKE," Kivinen et al., IP Security Protocol Working Group (IPSE) Internal Draft, Dec. 13, 2001, for proposed use within the Internet Key Exchange (IKE) protocol.

Administrative Functions

The system can simulate pings to any of the Route Directors (NRD or PRD). This ping is sent inside a packet generated from an application used to trouble-shoot the virtual network by touching the Route Directors and insuring that they are alive.

An exemplary ping process is illustrated in FIG. **19**. As shown therein a pinging application **1902** at step **1904** sends a ping request packet **1906** to the Route Director **560**. The Route Director receives the ping at **1910** and generates a response at step **1912**. The ping response packet **1914** is returned to the application **1902** at step **1916**.

The ping request packet is sent and returned on the default tunneling UDP port (which in FIG. **5B** is UDP Port **20202**). The Ping Request **1906** packet is formatted as shown in FIG. **19**, with the content of the shim and the ping packets exploded in further detail. In the shim **1930**, the source and destination Route Director public IP addresses (gIP) and private IP addresses (pIP) are all set to 0. In the system ping field **1960**, the ping type field may be a ping request or a ping response. The Sequence field in the response is set to the same value as the Sequence field in the request packet.

Another function which may be performed is an Agent Query. FIG. **20** illustrates a query request from a Member Agent requiring system/status information of another member. The first Member Agent **565** at step **1802** will send a Query Request Packet **1810** to the VCN Manager **510**. The Packet contains definitions for the Maximum Number of Records to return, a status Mask indicating whether the members are registered and/or joined, a platform Mask, indicating the platform (desktop, server, laptop, mail server, etc) of the member, the FQDN Length, FQDN VCN Offset, Search

US 7,949,785 B2

25

String Length, Member FQDN and a Search String for pattern matching. This packet is sent over the IPsec tunnel between the VCN Manager and the Member Agent.

If the packet is abnormal it is dropped at step **2012** if there is an error in the query authentication file or the member is not joined, NACK **2016** is sent in step **2014**. If the packet is not abnormal, no error is present, the member passes authentication and was joined, then the VCN Manager sends a query acknowledgement Packet **2030** to the member. The packet payload includes a number of records with the following fields: the status of the member, the platform type of the member, and the member's FQDN String. Note that the FQDN is null-terminated and consistent with the FQDN in DNS format definitions. Here, a human readable FQDN "www.abcdefg.com" is encoded as "www.abcdefg.com\0x00".

Each joined member will periodically send a heartbeat packet to the VCN Manager so that the VCN Manager can determine if the member is alive. In addition, the member's heartbeat packet is used to determine how long the member is actually joined. This information can be used to allow a provider of the infrastructure for the system to generate revenue from users on a per-use time basis. Conversely, the VCN Manager periodically sends its heartbeat packet to joined members so that the joined members can know if the Manager is alive. This can be used by the member to determine if it should switch to a backup VCN Manager. Heartbeat packets are sent inside the system tunnel on the default tunneling port (UDP port **20202**) and are IPsec protected.

Route Directors

Route directors allow communication to a member that is in a private network. The PRD resides within a private network, behind the NAT device. The NRD resides on the public side of the NAT device. The NAT device can be a Network Address Port Translation (NAPT) NAT device or a basic NAT where only addresses are translated.

Note that devices with public addresses do not require Route Directors (RDs). In the context of FIG. 5B, devices M_A and M_E do not require either a PRD or NRD, since they reside in the public address space. Likewise, M_X does not require the PRD for its domain **540**, since it has a public address (or is statically mapped to a public address at the NAT device **515**).

The NRD implements a NAT traversal architecture which allows System packets to enter and exit a private network. The NRD maintains persistent connections/sessions with members that are behind NAT devices. Member Agents initiate a TCP or UDP persistent connection/session with the NRD and use keep-alive messages to ensure that the NAT device does not drop the member's connection/session. The use of TCP or UDP is based on a bit set by the VCN Manager in the INIT ACK packet of the Join Protocol. Communication to a member uses NRD tunnels between the NRD and the Member Agent through the persistent connection/session. More information about a suitable NAT traversal architecture is found in U.S. Pat. No. 7,139,828 and U.S. patent application Ser. No. 10/233,288, "Communicating With An Entity Inside a Private Network Using an Existing Connection to Initiate Communication," both of which are incorporated herein by reference.

When a member behind NAT attempts to join a virtual network by contacting the VCN Manager as discussed above, the VCN Manager will detect the presence of the NAT device, direct the member to a specific NRD and indicate the use of TCP or UDP. The presence of the NAT device must be detected during the initialize Init message portion of the join sequence so the VCN Manager can inform the member that it needs to setup a session with the assigned NRD prior to the Join message sequence.

26

In one embodiment, the NRD uses a pseudo address. The pseudo address is different than the virtual address and is used solely for NRD routing. The pseudo address assigned by the NRD will allow the NRD to map a packet to the member's post-NAT IP address and UDP port.

The process for establishing a session with a NRD and obtaining an address therefrom is shown in FIG. 21. Initially (as described in FIG. 13), a member behind a firewall will send an init request **1304** to VCN Manager **510**. In addition to the functions performed with respect to step **1306**, **1308**, and **1311** (of FIG. 13), the VCN Manager will detect that the member is behind a NAT device and whether it is served by a PRD in step **2120**.

The detection of a NAT device is accomplished using a field with the member's local IP address in the Init packet sent to the VCN Manager. The VCN Manager detects the fact that Agent **565** is behind a NAT device by comparing the member's private IP address in the INIT message payload with the source address in the TCP/IP headers. If the addresses differ, then a NAT device is present. At step **2120**, the VCN Manager must next check if the member's NAT device is associated with a known PRD (Premises Route Director). If no PRD is found, then the VCN Manager will check if the member can use a known NRD. If an NRD is found then the INIT ACK **1314'** will indicate that the member must establish a session with the NRD prior to joining the VCN. If the NRD can support TCP tunneling, the INIT ACK will also indicate to the member to start a TCP connection rather than a UDP connection. If neither a PRD nor an NRD is found for the private member, an Init NACK will be returned. The VCN Manager will find an available NRD and send an INIT ACK **1314'** with the NRD info data contained therein. The contents of an INIT ACK packet **1314** are discussed above with respect to FIG. 13.

At step **2132**, the member will seek to establish a session with the NRD and will send address request **2130** to the NRD. The member sends the Address Request Packet **2130** to the NRD. The packet includes an IP Header, UDP Header or TCP header, and Shim. In this case, the shim includes the NRD's public address. At step **2135**, the NRD assigns the member a pseudo address. This is used as the member's private address in the Shim when external members send packets to the NRD for forwarding to the private member. The NRD will store the association of the pseudo address with the Member's NAT device public address and port number. When other members of the VCN perform a DNS request on the member behind the NRD, as part of the DNS response, they will receive the public address of the NRD and they will receive the pseudo address as the private address for the member behind the NRD. When sending a communication to the member behind the NRD, the outer packet will be addressed to the public address of the NRD. The NRD will access the shim to identify the pseudo address. The NRD uses the pseudo address as a handle to the persistent connection/session when sending packets through a NAT to the member. Once identifying the appropriate persistent connection/session, the shim and virtual packet are encapsulated and sent to the destination, via the NAT device, using the identified persistent connection/session. When the NAT device receives the message, it will do the address and port translation, and forward the message to the member agent's real local address and port on the private network.

In one aspect, the pseudo address comprises portions of the public IP address space in IP version 4 that had been reserved for special purposes. This ensures that no machine within a private realm will have the same pseudo IP address as the private address of a machine in any separate and distinct

US 7,949,785 B2

27

private realm. In order to prevent this from happening, the system of the present invention uses addresses that are otherwise not used or reserved. Examples include the 0.0.0.0, the 10.0.0.0, 14.0.0.0, 127.0.0.0, 169.2454.0.0, 172.16.0.0 and the like.

The NRD provides the pseudo address to the Agent **565** in a NAT Traversal Address Response Packet **2140**. The packet includes a shim having an indicator that the response is a pseudo address response, as well as the source Route Director public IP address and Destination Route Director public IP address, (which is set equal to the NRD address), the Source member private IP address (0) and the Destination member private IP address, which is the Virtual IP. Hence all communications for the Member Agent will be routed to the NRD **520**. After acquiring the pseudo address from the NRD, the member designates that pseudo address to the VCN Manager as its local address in Join packet **1318**.

Note that there is an extra packet exchange **2145**, **2150** when establishing a TCP connection that is not required when establishing a UDP connection with the NRD. This packet is the NRD Init Request Packet **2145**. The previous TCP session for the Address Request and Address Response must be closed and a new TCP session opened which will remain open for the duration of the tunnel session. The member sends the Init Request message to the NRD as the first message in this TCP session. This message only has a TCP header and System Shim preamble, and has no data following the Shim.

After receiving the pseudo address, the join process completes with the join request packet **1318**, and join ack packets **1328**. After the join processing is complete, TCP/UDP encapsulated traffic flows between the member and the NRD through the NAT. The member is now ready to establish connections with other members.

When outbound System traffic from a member behind a NAT device to the NRD is not frequent enough to keep the NAT device's mappings active, the member may to send keep alive messages to the NRD's TCP port **80** or UDP encapsulation port to ensure that the NAT device does not drop the TCP/UDP encapsulation session. The member sends this message to the NRD to ensure that the NAT device does not drop the TCP/UDP port mapping. The keep-alive messages do not need to be sent when other messages were sent out during the keep-alive interval.

The member sends this NAT Traversal Leave Packet message to the NRD to release the session and pseudo address when the member is leaving the VCN. The NAT traversal switch will verify that the receive-from socket address this message came from matches the lookup table entry before deleting the member's address.

Group Agent

For some devices, member client software cannot be installed on the device or it is not desirable to install member client software on the device. For example, a printer or other networked devices may not be able to load software. For various reasons, some entities may not desire to add network software onto their machines. Additionally, some devices may use operating systems that do not support running the member agent software. For those devices that cannot or choose not to use member agent software, a Group Agent can be used. The Group Agent acts as a proxy for one or more members of a VCN without requiring installation of member agent software on the client devices. Thus, a device can become a member of a VCN without changing any of the software on the device by using the Group Agent.

FIG. **22** is a block diagram depicting various components of the present invention utilizing a Group Agent. Note that some of the members of a VCN can use a Group Agent while

28

other members do not use a Group Agent. FIG. **22** shows ten devices M_1 - M_{10} . Devices M_1 , M_2 , M_3 , M_4 and M_{10} are devices that are participating in the VCN without using member agent software. Rather, they will use a Group Agent.

Devices M_5 , M_6 , M_7 , M_8 and M_9 are devices which include member agent software and, therefore, do not need to use a Group Agent. FIG. **22** shows some of the many varieties of permutations of using Group Agents, Network Route Directors and Private Route Directors. Other permutations can also be used with the present invention. Also note that FIG. **22** is more of a structural view of additional components of the present invention, as compared to FIG. **5B** which is more of a symbolic/logical view.

FIG. **22** includes a first private network which connects to the Internet (or other network) via NAT device **2202**. In addition to NAT **2202**, the private network includes device M_1 , device M_2 and Group Agent **2204**. In one embodiment, the Group Agent runs on the same machine as a Private Route Director. Devices M_1 and M_2 do not include agent software. Rather, they participate in the VCN using Group Agent **2204**.

A second private network allows devices to communicate to the Internet (or other network) via NAT device **2210**. That private network also includes Group Agent **2212**, device M_3 and device M_4 . Note that Group Agent **2212** does not include a Private Route Director. Thus, for devices M_3 and M_4 to be reached by other members of the VCN, communication is performed via Network Route Director **520**. In one embodiment, each of the private networks are different physical address realms. That is each uses a different set of private IP addresses. The present invention will also work if the private networks have overlapping addresses. That is, one or more private IP addresses are used by more than one private network involved in a VCN.

A third private network, which includes devices M_5 and M_6 , provides for communication on the Internet via NAT device **2220**. A fourth private network, which includes devices M_7 and M_8 , allows its components to communicate on the Internet (or other network) via NAT device **2230**. This fourth private network includes Private Route Director **2232**. Because devices M_7 and M_8 include member agent software, there is no need for a Group Agent on this network. Device M_9 has a public IP address and includes member agent software; therefore, it communicates and participates in the VCN as described above. Device M_{10} has a public IP address; however, device M_{10} does not include member agent software. For device M_{10} to participate in the VCN, it must make use of Group Agent **2234**, which has a public IP address.

In one embodiment, a Group Agent includes software running on a computing device, including (but not limited to) a server, router, personal computer, minicomputer, mainframe, mobile computing device, or other suitable computing device. In other embodiments, the Group Agent, or any other of the software components described above, can be performed by its special purpose computing device.

A Group Agent acts as a proxy for one or more members which do not have member agent software. For example, Group Agent **2204** is a proxy for devices M_1 and M_2 ; and Group Agent **2212** is a proxy for devices M_3 and M_4 . Group Agent **2234** is a proxy for device M_{10} . Although FIG. **22** shows the Group Agents being proxies for one or two members, a Group Agent can be a proxy for more than two members.

Communication between two members of a VCN can pass through two NAT devices, one NAT device for each member. For example, communication between device M_1 and device M_4 will pass through NAT device **2202** and NAT device **2210**. Additionally, communication between device M_1 and another

US 7,949,785 B2

29

entity in a private network configured like the private network of device M_1 will require communication to pass through two NAT devices.

Note that a single Group Agent can participate in more than one VCN. For example, Group Agent 2204 can be used to allow device M_1 participate in a first VCN and concurrently (for all or part of the time) allow device M_2 to participate in a second VCN. Because there can be multiple virtual community networks, the VCN Manager is able to create and maintain, the multiple virtual community networks. As described herein, the VCN Manager identifies which entities can communicate on each VCN by maintaining a list of domain names for each VCN. For example, one list may include the domain names for devices M_1 - M_{10} .

The Group Agent requires initialization so it may know which members it is a proxy for. Group Agent initialization cannot occur until after the Group Agent has joined the VCN Manager as a member. The join prior to Group Agent initialization is for a pseudo-member so the proxy initialization messages can use virtual IP addresses and the tunneling protocols described above to travel on an IPsec tunnel between VCN Manager 510 and the particular Group Agent. Separate member specific tunnels will be created when the Group Agent later joins for the proxied members. The Group Agent join will require the Group Agent to be registered with the VCN Manager. In one embodiment, this registration will be done during the installation of the Group Agent.

FIG. 23 provides a flowchart describing one embodiment of the process for Group Agent initialization. In step 2302, the Group Agent joins as a pseudo-member. This process of step 2302 is performed as described above. In step 2304, the Group Agent will send an initialization request to the VCN Manager. The initialization request is a packet that includes the fully qualified domain name for the Group Agent, a packet version indication, a packet type indicating that it is an initialization request, a packet length, and an authentication code. If the received initialization request is abnormal (step 2306), then the initialization request is dropped in step 2308. If the packet was normal but authentication failed (step 2312), then the VCN Manager sends a NACK to the Group Agent in step 2314. If authentication was successful and the initialization request was normal, then the VCN Manager sends an acknowledgement to the Group Agent in step 2320.

After completion of step 2320, the Group Agent is initialized and now must join each of the members it is proxying. To do so, the Group Agent will perform a loop where each iteration will include joining for one of the members that is being proxied. In step 2322, it is determined whether there are any more members being proxied that have to be joined. If there are more members to join, then the Group Agent will join for one member being proxied in step 2324 and the method will loop back to step 2322. When there are no more proxied members to be joined, the process of FIG. 23 is complete (step 2330). The process of joining for a proxy member is the same as the process of joining described above; however, the process is performed for the member by the Group Agent rather than by the member agent. The virtual IP address received by the Group Agent is for the member being proxied. The Group Agent stores that virtual IP address in a data structure for the member. In one embodiment, there is a separate data structure for each member being proxied. In another embodiment, there is one data structure which has separate records for each member being proxied. This data stored will include the member's virtual IP address and private address in the local LAN, as well as other information that will be described below.

30

The acknowledgement sent from the VCN Manager to the Group Agent in step 2320 contains all of the member information required so the Group Agent can send the appropriate join requests for each member being proxied. The following information is in the acknowledgement sent from the VCN Manager to the Group Agent: packet version indicating the version of the header, packet type indicating that it is a acknowledgement, packet length, authentication code, subnet IP address, subnet prefix, public IP address of the Private Route Director (when the Group Agent also includes a Private Route Director), Private Route Director ping flag (1 if PRD responds to pings), the number of members being proxied, the number of permanent connections from proxied members to other members that the proxy must make, the member record list and the connect record list. The member record list is a set of records for each member being proxied. The following information is stored in each record: member password (password SHA1 digest), member's local private IP address, the length of the member's fully qualified domain name, the member's fully qualified domain name, offset to the VCN name, authentication type (0=preshared 1=certificate), time stamp indicating when the token key was created, token length in bytes, token key (member shared secret with VCN Manager), HA version number, number of HA records, HA record list and authentication key (generated by server for the proxy). Each HA record in the HA record list includes a VCN Manager IP address for the next VCN Manager, a join port for the next VCN Manager and a service port for the next VCN Manager in case the current VCN Manager goes down. The connect record list includes the following three fields: proxy member index (zero based index into member list), DNS request length and a DNS request.

Once the Group Agent joins for all of its members, then any of its members can receive communication from other members of the VCN or can initiate communication to other members in the VCN. Members that are being proxied by the Group Agent can communicate with other members being proxied by the Group Agent, with other members proxied by a different Group Agent (regardless of whether they are in a private network or available on the public Internet), and members that do not use a Group Agent (regardless of whether those other members are in a private network or available on the Internet with a public IP address), as long as they are all members of the same VCN. From the point of view of an application on a member being proxied by the Group Agent, the VCN appears as a local subnet (or LAN) and all of the members of the VCN appear to be on the local subnet (or LAN). However, the VCN is not a local subnet (or LAN); rather it is a virtual subnet (or LAN). Each of the members of the VCN will have a Virtual IP Address on the virtual subnet.

Because members of a VCN register with the VCN Manager, the VCN Manager will know how to find the members (e.g. it will know its public and private addresses). Thus, a Group Agent acting as a proxy for a first member will be able to act as an intermediary for that first member when talking to a second member of a VCN even if the second member moves to a different or new private network and/or changes its IP address. That is, when the second member makes the move or change, the second member will so inform the VCN Manager. The VCN Manager will inform the Group Agent of the change so that the Group Agent can continue to act as an intermediary between the first member and the group member. More information about communicating with mobile entities can be found in U.S. Pat. No. 7,139,828; U.S. patent application Ser. No. 10/161,573, "Creating A Public Identity For An Entity On A Network," filed on Jun. 3, 2002; and U.S. patent application Ser. No. 10/233,288, "Communicating With An Entity

US 7,949,785 B2

31

Inside A Private Network Using An Existing Connection To Initiate Communication,” filed on Aug. 30, 2002, all three applications are incorporated herein by reference in their entirety.

FIG. 24 is a flowchart describing one embodiment of a process for initiating communication with a member that is being proxied by a Group Agent. In FIG. 24, the communication is being initiated by a member that is not being proxied by a Group Agent. That is, the member initiating communication is a member that has agent software loaded on it. That member is initiating communication with a member that does not have agent software. Prior to the process of FIG. 24 being performed, the Group Agent has joined the member into the VCN and the member initiating communication has already joined the VCN. For example purposes, looking at FIG. 22, the process of FIG. 24 can be performed by M₅, M₆, M₇, M₈ or M₉ initiating communication with any of devices M₁, M₂, M₃, M₄ or M₁₀. Assume, for example purposes, that device M₉ is initiating communication with device M₁. The destination device is also assumed to be served by a Private Route Director.

In step 2402, an application on the source device (e.g., device M₉) is intending to send a message to an application on the destination device (e.g., device M₁). In step 2404, the application on the source device sends a DNS request to the VCN Manager using the domain name of the destination. For example, the application on M₉ will send a DNS request using the domain name for M₁. In step 2406, the VCN Manager returns a response to the DNS request. This response includes the public IP address for private Route Director 2204, the private IP address for member M₁ and the virtual IP address for member M₁. In some embodiments, the DNS response also includes a token key (as described above). The Agent on the source device (e.g., M₉) stores the information it received from the VCN Manager. In step 2410, the Agent on the source device returns the virtual IP address for the destination to the application which initiated the DNS request. In step 2412, the application on the source device initiates the sending of a message to M₁ using the virtual IP address. That is, a virtual IP packet is created where the source address is the virtual IP address for the source machine (M₉) and the destination address in the virtual IP packet is the virtual IP address for the destination device (device M₁). The virtual IP packet is created in step 2414 and subject to IPsec (or other security means). In step 2416, the virtual IP packet is encapsulated. Encapsulation includes adding a shim, a UDP header and an IP header, all as described above. The shim includes the public IP address for the Private Route Director associated with the destination member (e.g., PRD 2204) and the private IP address for destination (e.g. member M₁). If the source, the member initiating communication was in a private network, then the shim would also include the public IP address for the Route Director associated with the source and the private IP address for the source. In step 2418, the encapsulated packet is sent to the NAT for the destination’s private network. That is, in the example above, the newly created packet is sent to NAT 2202 via the Internet (or other network). In step 2420, the encapsulated packet is sent from the NAT (e.g., NAT 2202) to the Route Director for the destination (e.g., PRD 2204).

In step 2422 of FIG. 24, the Route Director removes the virtual IP packet and the shim from the encapsulation and presents both the virtual IP address and the shim to the Group Agent (e.g., Group Agent 2204). The Group Agent stores the information from the shim, as described above with respect to the member agent software. In one embodiment, the Group Agent includes the necessary logic to received the communi-

32

cation so that no route director is necessary. In step 2426, the Group Agent extracts the virtual IP packet from IPsec. In step 2428, the Group Agent chooses a private IP address for the source entity. In the above example, the source entity is device M₉. Group Agent 2204 will pick a private IP address that is routable in the private network associated with device M₁ and NAT 2202. This private address will be used by device M₁ to identify M₉. In step 2430, the Group Agent will change the source address in the virtual packet. That is, the virtual packet originally had a source address equal to the virtual IP address for the source device (e.g., M₉) and a destination address equal to the virtual IP address for the destination device (e.g., M₁). In step 2430, the source address in the virtual IP packet is changed to be equal to the private IP address chosen by the Group Agent for M₉ in step 2428. In step 2432, the Group Agent changes the destination address in the virtual IP packet to be equal to the private IP address in the local LAN for M₁. In step 2434, the packet is sent by the Group Agent to the destination machine (e.g., sent to M₁).

The above discussion of FIG. 24 was made with respect to the example that the communication has been initiated by device M₉ which is a member having a public IP address and member agent software. The process of FIG. 24 can also be performed by a member which does not have a public IP address, including members using a Private Route Director and/or a Network Route Director. When the process of FIG. 24 is being performed by members using Route Directors, the shim that is added to the encapsulated packet in step 2416 includes the private IP address for the member and the public IP address for the Route Director. The example used in discussing FIG. 24 also assumes that the destination was a device in a private network where the private network included a Group Agent and a Private Route Director. The process of FIG. 24 can also be adapted to work with a member using a Group Agent where the member has a public IP address and/or a member in a private network that uses a Network Route Director.

When sending communication to an entity with a public IP address that uses a Group Agent, the shim will include the public IP address for the Group Agent and the public IP address for the member in step 2416. The packet will not be sent to a NAT or to a Route Director. Rather, the packet will be sent directly to the Group Agent. The Group Agent will access the virtual IP packet in step 2426. Rather than choosing a private IP address to identify the source, the Group Agent will choose a public IP address for the destination to use to refer to the source. The virtual IP packet will be changed so that the source address is equal to the public IP address chosen by the Group Agent to represent the source and the destination address will become the public IP address for the destination. The virtual IP packet (which is no longer virtual) will be sent to the destination.

If the destination member participates in a VCN using a Network Route Director (e.g., device M₃ using NRD 520), then the encapsulated packet is sent to the Network Route Director in step 2418. The Network Route Director will communicate the shim and virtual IP packet to the Group Agent (e.g., Group Agent 2212) via a persistent connection as described above in FIG. 7, steps 673-675, 677, except that the packet is forwarded to the Group Agent rather than the NAT device. The Group Agent will then perform steps 2424-2434, as described above.

FIG. 25 is a flowchart describing one embodiment of a process for the destination device responding to the initiation of communication from the source device. In the example discussed above, this process would include device M₁ responding to device M₉. In step 2502, the application on

US 7,949,785 B2

33

device M_1 attempts to send a communication. A packet is then created using the private addresses. The packet would include a source address equal to the private IP address in the LAN for device M_1 . The destination address would be equal to the private IP address in the LAN that is used to identify device M_5 . That packet is sent to the Group Agent in step 2504. In step 2506, the Group Agent changes the packet by replacing the private addresses with virtual addresses. The source address is replaced with the virtual IP address in the VCN used to identify device M_1 . The destination address in the packet is replaced with the virtual IP address in the VCN used to identify device M_5 . In step 2508, this packet is subject to IPsec, and encapsulated as described above.

In step 2512 of FIG. 25, the Group Agent sends the packet. In the example above, the Group Agent would send the encapsulated packet to device M_5 using the public IP address for device M_5 . If the Group Agent is sending the packet to a member that is in a private network, then the encapsulated packet would be sent to the Group Agent/Private Route Director for the member or to the Network Route Director for that member. If the responding device (e.g. M_1) is using a Network Route Director, then the Group Agent can still send the reply packet directly to the original source without going through the Network Route Director. If the replying device (e.g. M_1) has a public IP address, then the packet sent to the Group Agent would include the public IP addresses for the responding device and the public IP address that the Group Agent told the replying device to use to identify the original source device. In one embodiment, when the member using the Group Agent has a public IP address, an IPsec tunnel can be created between the Group Agent and the member device.

FIG. 24 contemplates communication being initiated by a device that has member software loaded on it. FIG. 26, on the other hand, describes a process that is performed when communication is initiated by a device that does not have member software and, therefore, uses Group Agent software. In step 2602, the application on the initiating member will send a DNS request. This DNS request will be sent to the Group Agent. The Group Agent receives the DNS request in step 2604. In step 2606, the Group Agent sends its own DNS request for the same domain name to the VCN Manager. In step 2608, the Group Agent receives a response from the VCN Manager. This response will include the virtual IP address for the target, the public IP address of the Route Director for the target (if any) and the private IP address for the target (if any). If the target has a public IP address, then the public IP address will be returned. In step 2610, the Group Agent will choose a private IP address in the local LAN to be used to identify the target. The virtual IP address returned from the DNS request will be mapped to this private IP address in the local LAN. In step 2612, the Group Agent will return that chosen private IP address to the application as a response to the application's DNS response. In step 2614, the application will initiate the communication. In one embodiment, the process of 2614 includes performing the steps of FIG. 25. Note that if the member who is initiating the communication in the process of FIG. 26 has a public IP address, rather than being in a private network, then in step 2610 the Group Agent will map the virtual IP address to a public IP address that the Group Agent and member will use to identify the target. It is this mapped public IP address that is returned to the application in response to the DNS request. The mapped public IP address is routable to the Group Agent.

Although the Group Agent is described above with respect to use with a VCN, the technology of the Group Agent is broader than a VCN. Thus, the Group Agent can be used to

34

allow entities to participate in other types of networks, communities, groups, organizations, communications, etc.

Note that in one embodiment, the member being serviced by the Group Agent must be in the same physical address realm as the Group Agent. In this manner, the Group Agent is used to bridge that physical address realm with the virtual space. The Group Agent contains the functionality to allow the member to participate in the virtual community; however, all the protocols of the virtual community terminate at the Group Agent. This way, the member receives a normal IP packet. From the point of view of the member, the Group Agent works like a router which connects the member to the subnet of virtual nodes.

The foregoing detailed description of the invention has been presented for purposes of illustration and description. It is not intended to be exhaustive or to limit the invention to the precise form disclosed. Many modifications and variations are possible in light of the above teaching. For example, any number of VCN Managers may be used. Any combination of NRDs and PRDs may be used to improve network efficiency. Any combination of Member Agents and Group Agents may be used, and the Virtual Community may be of any size. Additionally, while the above description provided an example using the protocols and addressing currently used on the Internet, the present invention can be used with other protocols and addressing schemes. The described embodiments were chosen in order to best explain the principles of the invention and its practical application to thereby enable others skilled in the art to best utilize the invention in various embodiments and with various modifications as are suited to the particular use contemplated. It is intended that the scope of the invention be defined by the claims appended hereto.

We claim:

1. A virtual network system, comprising:

a virtual network manager implemented with a first device memory and a first device processor of a first computing device, the virtual network manager configured to register devices in a virtual network that is defined by a domain name, each device in the virtual network being identified to the other devices by a virtual network address that is unique for each device and not directly routable via a public network, the virtual network manager further configured to distribute a virtual network address to a device when the device is registered in the virtual network;

a route director implemented with a second device memory and a second device processor of a second computing device, the route director configured to communicate data between the devices that are registered in the virtual network, the data being communicated as encapsulated packets from a source device to a destination device, an encapsulated packet including a first virtual network address that corresponds to the source device and a second virtual network address that corresponds to the destination device; and

the virtual network manager further configured to receive a DNS request from the source device, and return a public network address of the route director, a private network address for the destination device, and the second virtual network address that corresponds to the destination device.

2. The system of claim 1 wherein each of the devices communicate with the virtual network manager and the other devices in the virtual network via an associated agent.

3. The system of claim 2 wherein the associated agent is installed on a proxy device which is a proxy agent for one or

US 7,949,785 B2

35

more of the devices in the virtual network with respect to the virtual network manager and the route director.

4. The system of claim 2 wherein the virtual network manager is further configured to:

receive a request from the agent associated with a device in a private network to register the device in the virtual network; and
detect a presence of a NAT device via which the request is routed from the agent to the virtual network manager.

5. The system of claim 4 wherein the virtual network manager is further configured to receive the request from the agent as a message, and detect the presence of the NAT device by comparing a field in a payload of the message with a source address in a header of the message.

6. The system of claim 1 wherein each of the devices include an agent configured to communicate with the virtual network manager and agents of the other devices in the virtual network.

7. The system of claim 1 wherein the virtual network manager is coupled to the public network and has an associated public network address.

8. The system of claim 1 wherein the route director is further configured to receive the encapsulated packets when routed to a public network address corresponding to the route director.

9. The system of claim 1 wherein each of the devices in the virtual network are further identified by at least one physical network address.

10. The system of claim 9 wherein the physical network addresses that are associated with the devices in the virtual network are dynamic.

11. The system of claim 9 wherein the physical network addresses that are associated with the devices in the virtual network are static.

12. The system of claim 9 wherein the physical network addresses that are associated with the devices in the virtual network are private network addresses, and wherein the virtual network addresses are unique in the virtual network so as not to conflict with the private network addresses.

13. The system of claim 9 wherein the source device is coupled to a first private network and accesses a public network via a first NAT device configured to transmit the encapsulated packets, an encapsulated packet further including the physical network address of the source device as a private network address in the first private network.

14. The system of claim 13 wherein the physical network address of the source device is dynamic.

15. The system of claim 13 wherein the physical network address of the source device is static.

16. The system of claim 13 wherein the destination device is coupled to a second private network and accesses the public network via a second NAT device, the encapsulated packet further including the physical network address of the destination device as a private network address in the second private network.

17. The system of claim 16 wherein the physical network address of the destination device is dynamic.

18. The system of claim 16 wherein the physical network address of the destination device is static.

19. The system of claim 16 wherein said first private network and said second private network share at least one network address.

20. The system of claim 1 wherein the route director is further configured to:

provide a device-specific pseudo address assignment for the device in the virtual network when the device communicates with the route director via a NAT device; and

36

store an association of the device-specific pseudo address with a public network address of the NAT device and a port number of the NAT device.

21. The system of claim 1 wherein the route director includes a translator for virtual network address information for a device in the virtual network that is implemented in a private network.

22. The system of claim 1 wherein the virtual network manager is coupled to the public network and includes a public network address.

23. The system of claim 22 wherein the route director is coupled to the public network and includes a different public network address than the public network address of the virtual network manager.

24. The system of claim 22 wherein the route director is coupled to a private network and includes a private network address.

25. The system of claim 1 wherein the encapsulated packets further include virtual IP packets configured for communication of the data, the virtual IP packets being encrypted prior to being encapsulated.

26. The system of claim 25 wherein the virtual IP packets are encrypted using IPSEC.

27. The system of claim 25 wherein the virtual IP packets are encrypted using DES or triple DES.

28. The system of claim 1 wherein the virtual network manager includes a virtual community definition that is defined by the domain name, and includes one or more of the devices that are registered in the virtual network.

29. The system of claim 1 wherein the route director is a public route director in the public network, the system further comprising a private route director in a private network configured to enable access to devices of the private network that are also in the virtual network from devices of the public network that are also in the virtual network.

30. A virtual network manager, comprising:

a network interface configured for data communication via a virtual network that is defined by a domain name having an associated public network address;

a memory and a processor to implement a register module configured to register devices in a virtual network, the register module further configured to:

receive a registration request from an agent associated with a device;

distribute a virtual network address to the device when the device is registered in the virtual network, the device being identified to other devices in the virtual network by the virtual network address; and

a DNS server for the virtual network, the DNS server configured to receive a DNS request from a first device in the virtual network, and return a network address associated with a network route director, a private network address associated with a second device in the virtual network, and a virtual network address associated with the second device.

31. The virtual network manager of claim 30 further comprising an additional network interface configured for data communication via a public network.

32. The virtual network manager of claim 31 wherein the network interface is a UDP port configured for data communication via the virtual network, and wherein the additional network interface is a TCP port configured for registration communications via the public network.

33. The virtual network manager of claim 30 wherein the network interface is a UDP port configured for data communication via the virtual network.

US 7,949,785 B2

37

34. The virtual network manager of claim 30 wherein the register module is further configured to receive the registration request from the agent that is installed on the device for data communication via the virtual network.

35. The virtual network manager of claim 30 further comprising a join module configured to receive a join request from the agent associated with the device to indicate that the device is connected for data communication within the virtual network, the join module further configured to receive a leave request from the agent associated with the device to indicate that the device will be disconnected from data communication within the virtual network.

36. The virtual network manager of claim 35 wherein the join module is further configured to provide virtual network addresses to the devices that are registered in the virtual network.

37. The virtual network manager of claim 35 wherein the join module is further configured to maintain data to associate a virtual network address with a device in the virtual network.

38. A virtual network system, comprising:

a computing device that includes at least a memory and a processor configured to implement a network manager of a virtual network that is defined by a public domain name, the network manager configured to distribute virtual network addresses to devices that register as members in the virtual network, each device in the virtual network being identified to the other devices by a virtual network address associated with the device;

a first virtual network agent associated with a first device that is registered as a member in the virtual network;

at least a second virtual network agent associated with at least a second device that is registered as a member in the virtual network;

a route director configured to route communications between the first device and the at least second device in the virtual network via the respective first and second virtual network agents, the communications configured for routing as encapsulated packets that include a first virtual network address that is not directly routable corresponding to the first device and a second virtual network address that is not directly routable corresponding to the at least second device; and

the network manager includes a DNS server configured to provide authoritative responses for DNS queries in the virtual network, the DNS server further configured to receive a DNS query from the first device and return a network address of the route director, a network address of the second device, and the virtual network address of the second device.

39. The system of claim 38 wherein the route director is a public network route director that includes a public network interface and the network address is a public network address by which the first and second virtual network agents communicate with the public network route director.

40. The system of claim 38 wherein the route director is a private route director that includes a private physical network interface and the network address is a private physical network address.

41. The system of claim 38 wherein the first virtual network agent is installed on the first device for data communication via the virtual network.

42. The system of claim 41 wherein the first device is in a private physical network, has an associated private network address, and has the associated virtual network address that is not directly routable by which the first device can be identified by the at least second virtual network agent from outside of the private physical network.

38

43. The system of claim 41 wherein the first device is coupled to a public physical network, has an associated public network address, and has the associated virtual network address by which the first device can be identified by the at least second virtual network agent.

44. The system of claim 38 wherein the first virtual network agent is a proxy agent for the first device with respect to the network manager and the route director.

45. The system of claim 38 wherein the first and second devices are configured to access the virtual network from separate private address physical realms.

46. The system of claim 38 wherein the network manager includes at least a first community definition and a second community definition, the first community definition being defined by the public domain name and includes one or more of the devices that are registered in the virtual network, and the second community definition being defined by a different public domain name and includes one or more different devices that are registered in the virtual network.

47. The system of claim 38 wherein the network manager includes a member authenticator configured to authenticate the devices that request to register with the network manager.

48. A computer-implemented method, comprising:

receiving registration requests from devices that request to be registered as members of a virtual network that is defined by a domain name having an associated public network address in a public network, each of the devices having an associated private network address;

distributing a virtual network address to a device to register the device as a member in the virtual network, each device in the virtual network being identified to the other devices by the virtual network address that is associated with the device;

routing communications between the devices that are registered in the virtual network, the communications being routed as encapsulated packets from a source device to a destination device, an encapsulated packet including a first virtual network address that corresponds to the source device and a second virtual network address that corresponds to the destination device; and

transmitting a response to a DNS request received from one of the devices that are the members in the virtual network, the response to the DNS request including a public network address of a route director that registers the devices, a public network address of the destination device, and the second virtual network address that corresponds to the destination device.

49. The method of claim 48 further comprising providing the devices in the virtual network with a communication agent.

50. The method of claim 49 wherein said providing the devices in the virtual network with a communication agent includes providing a proxy agent.

51. The method of claim 48 further comprising authenticating the devices as the members in the virtual network.

52. The method of claim 48 further comprising defining a member set of the virtual network that includes one or more of the devices, and assigning the domain name that defines the virtual network.

53. The method of claim 48 further comprising defining at least two member sets of the virtual network that each include one or more of the devices, the at least two member sets having at least one different device.

54. The method of claim 48 further comprising assigning the virtual network address to the device as an IPV4 compliant address.

US 7,949,785 B2

39

55. The method of claim 54 wherein the IPV4 compliant address is non-routable.

56. The method of claim 48 further comprising routing network traffic via the public network from a first device having a public network address to a second device having a different public network address, the network traffic being routed as data packets having a source address which is the virtual network address of the first device and a destination address which is the virtual network address of the second device, the data packets being encapsulated to include a second source address which is a public network address of the first device and a second destination address which is the different public network address of the second device.

57. The method of claim 48 further comprising routing network traffic from a first device in a private physical network having a private network address to a second device having a public network address, the network traffic being routed as data packets having a source address which is the virtual network address of the first device, a destination address which is the virtual network address of the second device, and a shim which includes the private network address, the data packets being encapsulated to include a second destination address which is the public network address of the second device.

58. The method of claim 48 further comprising routing network traffic from a first device in a private physical network having a first private network address to a second device in a different private physical network having a second private network address, the network traffic being routed as encapsulated packets having a source address which is the virtual network address of the first device, a destination address which is the virtual network address of the second device, and a shim which includes the first and second private network addresses.

59. The method of claim 58 wherein the first private network address and the second private network address are identical.

60. The method of claim 48 further comprising:
receiving a join status request from a first device in the virtual network as a query to determine the status of a second device in the virtual network; and
responding to the join status request to indicate whether or not the second device is joined to the virtual network for data communication.

61. The method of claim 48 further comprising applying a group policy to the devices that are registered as the members of the virtual network.

62. One or more processor readable storage media devices comprising processor readable code that, if executed by a computer device, implements a virtual network manager to:
receive registration requests from devices that request to be registered as members of a virtual network that is defined by a domain name having an associated public network address in a public network, each of the devices having an associated private network address;
distribute a virtual network address to a device to register the device as a member in the virtual network, each device in the virtual network being identified to the other devices by the virtual network address that is associated with the device;

manage communications routed between the devices that are registered in the virtual network, the communications routed as encapsulated packets from a source device to a destination device, an encapsulated packet including a first virtual network address that corresponds to the source device and a second virtual network address that corresponds to the destination device; and

40

transmit a response to a DNS request received from one of the devices that are the members in the virtual network, the response to the DNS request including a public network address of the virtual network manager, a public network address of the destination device, and the second virtual network address that corresponds to the destination device.

63. One or more processor readable storage media devices as recited in claim 62 further comprising processor readable code that, if executed, implements the virtual network manager to provide the devices in the virtual network with a communication agent.

64. One or more processor readable storage media devices as recited in claim 63 wherein the virtual network manager provides the as a proxy agent.

65. One or more processor readable storage media devices as recited in claim 62 further comprising processor readable code that, if executed, implements the virtual network manager to define a member set of the virtual network that includes one or more of the devices, and the domain name that defines the virtual network.

66. One or more processor readable storage media devices as recited in claim 62 wherein the virtual network address includes a non-routable IPV4 compliant address.

67. One or more processor readable storage media devices as recited in claim 62 further comprising processor readable code that, if executed, implements the virtual network manager to route network traffic via the public network from a first device having a public network address to a second device having a different public network address.

68. One or more processor readable storage media devices as recited in claim 62 further comprising processor readable code that, if executed, implements the virtual network manager to route network traffic from a first device in a private physical network having a private network address to a second device having a public network address.

69. One or more processor readable storage media devices as recited in claim 62 further comprising processor readable code that, if executed, implements the virtual network manager to route network traffic from a first device in a private physical network having a first private network address to a second device in a different private physical network having a second private network address.

70. One or more processor readable storage media devices as recited in claim 69 wherein the network traffic is routed to the first or second device via a NAT device.

71. One or more processor readable storage media devices as recited in claim 69 wherein the first private network address and the second private network address are identical.

72. One or more processor readable storage media devices as recited in claim 67 wherein the network traffic is routed as encapsulated data packets.

73. One or more processor readable storage media devices as recited in claim 67 wherein the network traffic is routed as encrypted data traffic.

74. One or more processor readable storage media devices as recited in claim 62 further comprising processor readable code that, if executed, implements the virtual network manager to apply a group policy to the devices that are registered as the members of the virtual network.

75. A virtual network system, comprising:
a computing device that includes at least a memory and a processor configured to implement a virtual network manager having a network interface coupled to a virtual network, the virtual network manager including at least one virtual community definition that is defined by a domain name having an associated public network

US 7,949,785 B2

41

address and a user set of one or more devices that are registered in the virtual network, each device in the virtual network being identified to the other devices by a virtual network address that is associated with the device, the virtual network manager configured to exchange virtual network information with the one or more devices of the user set, the virtual network being accessible by devices in the user set and devices outside of the user set, and the virtual network manager further configured to receive a DNS request from a source device, and return a public network address of a route director, a private network address for a destination device, and a virtual network address that corresponds to the destination device.

76. The system of claim 75 wherein the virtual network manager includes a member register module.

77. The system of claim 75 wherein the virtual network manager includes a member join module.

78. The system of claim 77 wherein the member join module provides a virtual network address to a device that is registered as a member of the network.

79. The system of claim 75 wherein the virtual network manager is further configured to maintain data on an association between at least one virtual network address with at least one device that is registered as a member of the network.

80. The system of claim 75 wherein the virtual network manager includes a DNS server for the virtual community network.

81. The system of claim 75 wherein the virtual network manager includes a NAT device detector for devices connecting with the virtual network manager behind a NAT device.

42

82. The system of claim 75 wherein the virtual network manager includes at least a second virtual community definition.

83. The system of claim 75 wherein the virtual network manager includes a member authenticator.

84. The system of claim 75 wherein the virtual network manager includes a DNS server configured to provide authoritative responses for DNS queries from devices in the virtual community.

85. The system of claim 75 further comprising at least one route director configured to communicate with the one or more devices in the user set.

86. The system of claim 75 wherein each device registered in the network is configured to communicate with the virtual network manager and other devices in the user set via at least one agent.

87. The system of claim 75 wherein the user set includes at least a first device and a second device, and wherein at least one of said first device and said second device is coupled to a first private network and accesses a public network via a NAT device.

88. The system of claim 87 wherein said first device is coupled to said first private network, and said second device is coupled to a second private network and accesses the public network via a second NAT device.

89. The system of claim 75 wherein communications between the one or more devices in the user set are encrypted.

90. The system of claim 89 wherein the virtual network manager is configured to provide a shared message to the one or more devices in the user set to establish encrypted communications.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

PATENT NO. : 7,949,785 B2
APPLICATION NO. : 10/403818
DATED : May 24, 2011
INVENTOR(S) : Alkhatib et al.

Page 1 of 1

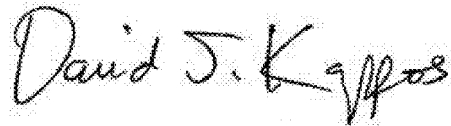
It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Title Page 3, item (56), under "Other Publications", in Column 1, Line 11, delete "Nelowrks." and insert -- Networks. --.

Title Page 3, item (56), under "Other Publications", in Column 2, Line 22, delete "Infrastucture" and insert -- Infrastructure --.

Column 40, line 15, in Claim 64, delete "the" and insert -- the communication agent --.

Signed and Sealed this
First Day of November, 2011

A handwritten signature in black ink, reading "David J. Kappos". The signature is stylized with a large, looped "D" and a cursive "K".

David J. Kappos
Director of the United States Patent and Trademark Office

(12) **EX PARTE REEXAMINATION CERTIFICATE** (12910th)
United States Patent
Alkhatib et al. (10) **Number:** **US 7,949,785 C1**
(45) **Certificate Issued:** **May 7, 2025**

(54) **SECURE VIRTUAL COMMUNITY NETWORK SYSTEM**

(75) Inventors: **Hasan S. Alkhatib**, Saratoga, CA (US);
Fouad A. Tobagi, Los Altos, CA (US);
Farid F. Elwailly, San Jose, CA (US)

(73) Assignee: **INTELLECTUAL VENTURES I LLC**

Reexamination Request:
No. 90/019,519, May 22, 2024

Reexamination Certificate for:
Patent No.: **7,949,785**
Issued: **May 24, 2011**
Appl. No.: **10/403,818**
Filed: **Mar. 31, 2003**

Certificate of Correction issued Jan. 11, 2011

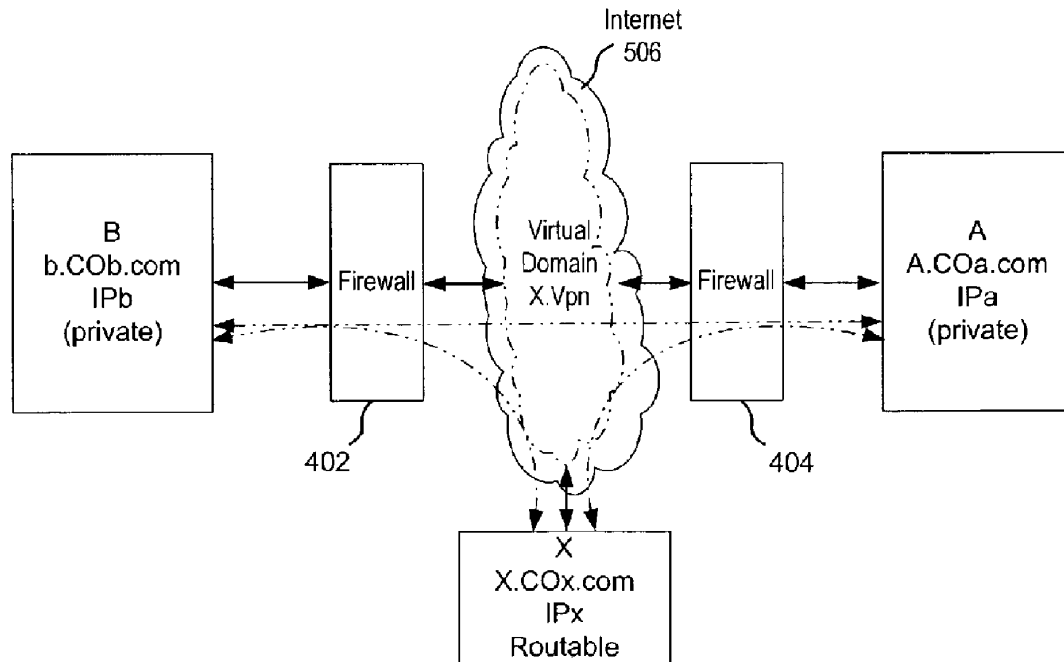
(51) **Int. Cl.**
G06F 15/16 (2006.01)
G06Q 10/109 (2023.01)
G06Q 30/02 (2023.01)
(52) **U.S. Cl.**
CPC **G06Q 10/109** (2013.01); **G06Q 30/02** (2013.01)

(58) **Field of Classification Search**
None
See application file for complete search history.

(56) **References Cited**
To view the complete listing of prior art documents cited during the proceeding for Reexamination Control Number 90/019,519, please refer to the USPTO's Patent Electronic System.

Primary Examiner — Dennis G Bonshock

(57) **ABSTRACT**
A private virtual dynamic network is provided for computing devices coupled to public networks or private networks. This enables computing devices anywhere in the world to join into private enterprise intranets and communicate with each other. In one embodiment, the present invention provides a separate private virtual address realm, seen to each user as a private network, while seamlessly crossing public and private network boundaries. One implementation of the present invention uses an agent to enable an entity to participate in the network without requiring the member to add new hardware or software.



US 7,949,785 C1

1

2

**EX PARTE
REEXAMINATION CERTIFICATE**

NO AMENDMENTS HAVE BEEN MADE TO 5
THE PATENT

AS A RESULT OF REEXAMINATION, IT HAS BEEN
DETERMINED THAT:

The patentability of claims **30** and **34** is confirmed. 10
Claims **1-29**, **31-33** and **35-90** were not reexamined.

* * * * *

EXHIBIT A-6

REMARKS

I. Status of Claims

Claims 30 and 34 of U.S. Patent No. 7,949,785 ('785 Patent) are under reexamination. Claims 1-29, 31-33, and 35-90 are not subject to this reexamination proceeding.

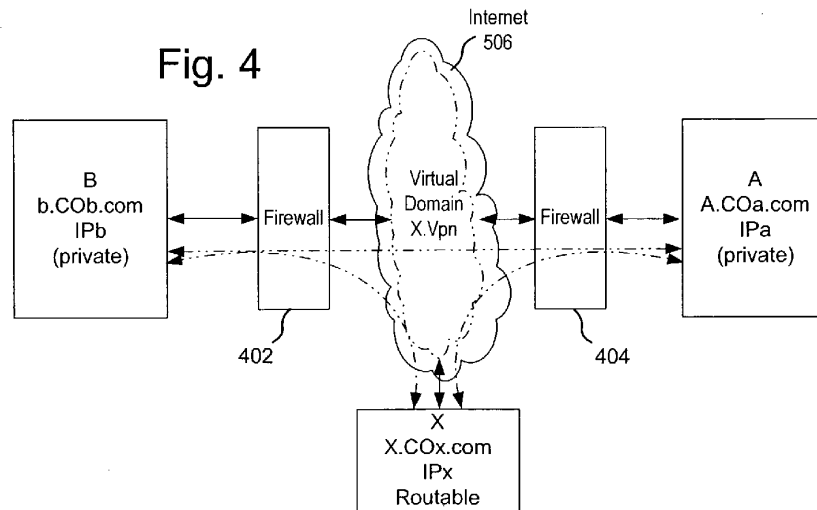
Pursuant to 37 C.F.R. 1.530(e), Patent Owner identifies the status of all patent claims as follows:

Claims 30 and 34: original and under reexamination;

Claims 1-29, 31-33 and 35-90: original and not under reexamination.

II. Example of the Invention

The '785 Patent states that "[a] basic overview of the system is shown in FIG. 4." ('785 Patent, 9:9). Specifically, Figure 4 (reproduced below) shows a Virtual Community Network (VCN) that allows devices, such as A and B, to communicate as if they were on the same physical local network.

Patent Owner: Intellectual Ventures I LLC**Application No.:** 90/019,519

In an example, the ‘785 Patent explains that, “[a]s shown in FIG. 4, A, B and X may all make direct connections to each other through communications within the virtual domain. In FIG. 4, dashed lines represent direct communication paths seen to applications running on A, B and X, all “seeing” the virtual addresses of each other and communicating with each other using the application’s IP interface.” (‘785 Patent, 9:36-41). In addition, the ‘785 Patent explains that “A and B have dynamic or static private IP addresses. Computer or device X is coupled directly to the public Internet and has a public IP address.” (‘785, 9:15-17). The interface between “X.Cox.com IPx Routable” and “Virtual Domain X.Vpn” is an example of a network interface configured for data communication via a virtual network that is defined by a domain name having an associated public network address. In the example of Figure 4, “X.Vpn” is the domain name defining the virtual network. In this example, the interface for accessing the virtual network from outside the virtual network uses

Patent Owner: Intellectual Ventures I LLC

Application No.: 90/019,519

“X.Cox.com IPx Routable” which has a public network address as identified by the top level domain suffix “.com”.

III. Response to Claim Rejections

Issue 1:

- Claims 30 and 34 were rejected under 35 U.S.C. §103 over U.S. Patent No. 7,036,143 to Leung et al. (hereinafter, “Leung”) in view of Patent No. 7,155,518 by Forslow (hereinafter, “Forslow”) and U.S. Publication No.: 2003/0028671 by Mehta (hereinafter, “Mehta”).

Patent Owner respectfully traverses the foregoing obviousness rejections and submits that Claims 30 and 34 recite features not taught, suggested, or otherwise yielded by the primary reference, Leung, or any proper combination of the cited references. Accordingly, the Patent Owner submits that the claims are patentable for at least the following reasons.

A. Mehta, Forslow, And Leung, Alone Or In Combination, Fail To Teach Or Suggest A “Domain Name” As Recited In The Challenged Claims.

Claim 30, recites, in part:

a network interface configured for data communication via
a virtual network that is defined by a domain name having
an associated public network address;

None of the cited references teach or suggest “a virtual network that is defined by a domain name having an associated public network address,” as recited in Claim 30.

Patent Owner: Intellectual Ventures I LLC
Application No.: 90/019,519

The Office Action relies on Leung for this limitation. However, Leung makes no mention of a “domain name” in any respect, let alone “a domain name having an associated public network address.” In order to fill this gap in Leung, the Office Action argues that, “A POSITA would have understood that the subnetwork address obtained from the Internet service provider disclosed in Leung is based on the domain name of the public network” (Non-Final Office Action, Pg. 12). The Office Action cites no evidence in support of this assertion.

It is the Examiner's burden to demonstrate that each and every element of Claim 30 is met by the relied on references, either expressly or inherently. The Office Action does contend that Leung expressly discloses that its subnetwork address obtained from the Internet service provider is “based on the domain name of the public network.” That is because Leung makes no mention of a “domain name” -- the term appears nowhere in the reference. Nor does the Office Action allege (let alone demonstrate) that this feature is inherent in Leung. Inherency requires that the limitation must be present in the reference, which is not the case here. The fact that the feature is possibly present in the reference is insufficient to demonstrate inherency.

The Patent Owner respectfully submits that obtaining a network address from an Internet service provider (as disclosed in Leung) does not inherently require that such network address is associated with, or based on, a domain name. More specifically, the assignment of an IP address by an ISP does not inherently require a “domain name.” For example, in a case where an ISP assigns the network/

Patent Owner: Intellectual Ventures I LLC
Application No.: 90/019,519

subnetwork address of “192.168.1.10” to a customer, there is no requirement the address be associated with, or based on, a domain name -- and ISP's routinely assign network/subnetwork addresses that are not based on, or associated with, a domain name (See Ex. 2001, ¶35).

The Microsoft Computer Dictionary 5th Ed (Ex. 2002) defines a “domain name” as:

An address of a network connection that identifies the owner of that address in a hierarchical format: server.organization.type. For example, www.whitehouse.gov identifies the Web server at the White House, which is part of the U.S. government (Ex. 2002, Pg. 3).

Similarly, as explained by ““What’s In A Domain Name?” A Critical Analysis Of The National And International Impact On Domain Name Cybersquatting” (Ex. 2003):

For people to find a business web page on the Internet, individuals need to know the business’ Internet address. These addresses are extremely important tools for communication. By typing in a certain series of letters, numbers, and symbols, referred to as an Internet domain name, an Internet user gains access to a desired website (Ex. 2003) (internal citations omitted).

Accordingly, a domain name is a human-readable address that identifies a specific location on the Internet, such as a website or an online service. It serves as an *alias* for an IP address, which is the numerical address (e.g., “192.168.1.10”) computers use to locate resources on a network. Domain names simplify access to online resources by allowing users to type a recognizable name, like example.com,

Patent Owner: Intellectual Ventures I LLC

Application No.: 90/019,519

instead of a complex numerical IP address. They are structured hierarchically and consist of multiple parts separated by dots, such as www.example.com, where www is the subdomain (optional), example is the second-level domain, and com is the top-level domain (TLD). The Domain Name System (DNS) translates these domain names into their corresponding IP addresses, enabling computers to locate and communicate with the correct servers or devices (see Ex. 2001, ¶32-34). However, there is no requirement that a numerical IP address such as that assigned by the ISP in Leung (e.g., “192.168.1.10”) be associated with a domain name maintained by a DNS (Ex. 2001, ¶35).

The Microsoft Computer Dictionary 5th Ed (Ex. 2002) defines an “IP Address” as:

Short for Internet Protocol address. A 32- bit (4-byte) binary number that uniquely identifies a host (computer) connected to the Internet to other Internet hosts, for the purposes of communication through the transfer of packets. An IP address is expressed in “dotted quad” format, consisting of the decimal values of its 4 bytes, separated with periods; for example, 127.0.0.1. The first 1, 2, or 3 bytes of the IP address identify the network the host is connected to; the remaining bits identify the host itself. The 32 bits of all 4 bytes together can signify almost 2³², or roughly 4 billion, hosts (A few small ranges within that set of numbers are not used) (Ex. 2002, Pg. 4).

Accordingly, a POSITA would understand an IP address to be a unique numerical identifier assigned to each device connected to a network that uses the Internet Protocol for communication. It serves two primary purposes: identifying the host or network interface and providing the location of the device within the network, enabling data to be sent and received (see Ex. 2001, ¶33).

Patent Owner: Intellectual Ventures I LLC

Application No.: 90/019,519

In computer networking, IP addresses are obtained through either dynamic allocation or static assignment, depending on the network configuration. Dynamic allocation, the most common method, is managed by a Dynamic Host Configuration Protocol (DHCP) server. When a device connects to a network, it sends a request to the DHCP server, which assigns an available IP address from a predefined pool. This process is automated and ensures that devices on the network are assigned unique IP addresses without manual intervention. The assigned address is typically leased for a specific duration, after which the lease may be renewed, possibly resulting in a new IP address. This method is widely used in home and enterprise networks due to its simplicity and efficiency (see Ex. 2004).

Alternatively, an IP address can be obtained through static assignment, where a specific address is manually configured on the device. This ensures that the IP address remains consistent and is commonly used for devices that require reliable identification, such as servers, printers, or security cameras. For external-facing devices on the Internet, Internet Service Providers (ISPs) assign public IP addresses, which can be either static or dynamic. Additionally, modern networks may use either IPv4 or IPv6 protocols, with IPv6 designed to address the limitations of IPv4's finite address space (see Ex. 2004).

In neither the static nor the dynamic cases is a domain name required. Instead, when a device connects to a network, the DHCP server assigns it a dynamic IP address according to IETF RFC 2131 (see Ex. 2005). Nowhere does IETF RFC 2131 require that obtaining an IP address require registration with or the existence of a

Patent Owner: Intellectual Ventures I LLC
Application No.: 90/019,519

domain name. In fact, the vast majority of IP addresses (including the laptop utilized to write this document) have been assigned IP addresses and are not associated with a domain name (see Ex. 2001, ¶34).

The same hold true for IP address assigned by ISPs. ISPs assign IP addresses to devices connecting to their networks using methods that depend on the type of connection the user's needs. Most ISPs rely on a DHCP server to assign numerical IP addresses dynamically. When a device connects to the network, it sends a request to the DHCP server, which assigns an available numerical IP address from the ISP's pool along with configuration details like the subnet mask, gateway, and DNS servers. These addresses are leased for a set period, after which the device may either renew the lease or receive a new IP address. In some cases, such as businesses hosting servers or running services requiring remote access, ISPs assign static IP addresses that remain constant. In this case, the ISP manually reserves a numeric IP address for the customer's account and configures it in their network infrastructure (see Ex. 2006). However, in neither case, does the assignment of the numeric IP address by the ISP require an association of the numeric address with a domain name (see Ex. 2001, ¶35).

Therefore, Leung's disclosure that an ISP assigns a public IP address does not inherently teach "a virtual network that **is defined by a domain name** having an associated public network address," as recited in claim 30. As explained by the Federal Circuit, the use of inherency, "a doctrine originally rooted in anticipation, must be carefully circumscribed in the context of obviousness" and "[t]he mere fact

Patent Owner: Intellectual Ventures I LLC
Application No.: 90/019,519

that a certain thing may result from a given set of circumstances is not sufficient to establish inherency” (*Par Pharm., Inc. v. TWi Pharm., Inc.*, 773 F.3d 1186, 1194-95 (Fed. Cir. 2014)) (internal markings and citations omitted). The Court has “emphasized that the limitation at issue necessarily must be present in order to be inherently disclosed by the reference” (*Southwire Co. v. Cerro Wire LLC*, 870 F.3d 1306, 1310-11 (Fed. Cir. 2017)) (internal citation omitted). Therefore, obviousness cannot be established by the mere fact that something may happen inherently. Instead, in the cited reference, the alleged feature must necessarily be present.

Here, as Dr. Noubir explains “that the assignment of IP addresses is completely unrelated to the existence of a domain name. Therefore, there is nothing required in the assignment of an IP address by an ISP to a device that requires the addressee to have a domain name or even be a member of a domain” (Ex. 2001, ¶35).

Further, the ‘785 Patent makes clear that the assignment of an IP addresses (IPa, IPb and IPx) is completely independent of domain names (X.vpn) (see Fig 4 above). Therefore, Leung does not inherently teach “a virtual network that **is defined by a domain name** having an associated public network address,” as recited in the claim 30.

Further, Leung’s teaching of a VPN does not inherently require a “domain name.” The Microsoft Computer Dictionary 5th Ed (Ex. 2002) defines a “Virtual Private Network” as:

Nodes on a public network such as the Internet that communicate among themselves using encryption technology so that their messages are as safe from being intercepted and

Patent Owner: Intellectual Ventures I LLC
Application No.: 90/019,519

understood by unauthorized users as if the nodes were connected by private lines (Ex. 2002, Pg. 6).

Accordingly, a POSITA would understand that a Virtual Private Network (VPN) is a secure communication method that creates a connection between a user's device and a remote server over the Internet to replicate the functionality of being connected by a physical wire. The Office Action has proffered no explanation for why computers connected via VPN inherently require a domain name. In fact, the majority of computer networks operate without a domain name. As Dr. Noubir explains, "[t]here is absolutely nothing about a VPN that requires the existence of a domain or the connecting device to have a domain name" (Ex. 2001, ¶36).

Therefore, a POSITA would appreciate that the VPN taught in Leung does not inherently require "a network interface configured for data communication via a virtual network that is defined by a domain name having an associated public network address."

The suggested additions of Mehta and Forslow do not cure the shortcomings of Leung. The Office Action does not allege, nor do Mehta and Forslow teach a "virtual network that is defined by a domain name having an associated public network address" as recited in claim 30. As a result, Mehta and Forslow, alone or in combination with Leung, fail to teach or suggest every feature recited in claim 30.

Therefore, a *prima facie* case of obviousness has not been established because the combination of the cited references fails to yield the features recited in the claims.

For at least the foregoing reasons, it is respectfully submitted that claim 30 is allowable over Mehta, Forslow, and Leung.

Patent Owner: Intellectual Ventures I LLC
Application No.: 90/019,519

Claim 34 is directly dependent upon claim 30, and the Patent Owner submits the claim is allowable over the cited references of record based on at least its dependency.

Based on at least the arguments presented above, the withdrawal of the 35 U.S.C. §103 rejections of the claims are respectfully requested.

B. Mehta, Forslow And Leung, Alone Or In Combination, Fail To Suggest A DNS Server As Recited In The Challenged Claims.

Claim 30, recites, in part:

a DNS server for the virtual network, the DNS server configured to receive a DNS request from a first device in the virtual network, and return a network address associated with a network route director, a private network address associated with a second device in the virtual network, and a virtual network address associated with the second device.

None of the cited references teach or suggest “a DNS server for the virtual network that is configured to receive a DNS request” and “return a network address associated with a network route director, a private network address associated with a second device in the virtual network, and a virtual network address associated with the second device,” as recited in claim 30.

Instead, as the Office Action observes, “Leung does not expressly disclose such a DNS server” (Non-Final Office Action, Pg. 26). Accordingly, Leung does not teach a DNS server for the virtual network that is configured to receive a DNS request and “return a network address associated with a network route director, a private network address associated with a second device in the virtual network, and a virtual

Patent Owner: Intellectual Ventures I LLC
Application No.: 90/019,519

network address associated with the second device,” as recited in claim 30. Therefore, Leung fails to teach or suggest every feature recited in claim 30.

The suggested addition of Forslow does not cure the shortcomings of Leung. The Office Action alleges that “A POSITA would have understood that Forslow’s mobile service manager 7 functions as a DNS server by receiving HTTPS requests from mobile nodes 3a and/or 3b and returning “user fully qualified domain name 155b/155a, the mobile node 3 private IP address (10.0.0.1) and the virtual home agent’s 1a/1b public IP address (204.8.9.1 and 204.8.8.1 respectively)” (Non-Final Office Action, Pg. 26). The Patent Owner submits that Forslow’s mobile service manager 7 does not function as a DNS server that “is configured to receive a **DNS request**”.

Forslow states that “the mobile nodes 3 a and 3 b ... retrieve a list of workgroup members 26 by doing **HTTPS requests** 22 a and 22 b to the mobile service manager 7 and its portal” (Forslow, 10:7-10). However, HTTPS requests are fundamentally different from DNS requests.

The fact that a server receives an HTTPS request and resolves an IP address does make the sever a “DNS server” or the request a “DNS request,” as claimed. . A DNS request is responsible for **resolving a domain name**, such as www.example.com, into its corresponding IP address (e.g., 93.184.216.34) to enable devices to locate and communicate with the server hosting the desired resource. Significantly, **a DNS request includes the domain name to be resolved**, and the

Patent Owner: Intellectual Ventures I LLC
Application No.: 90/019,519

response from the DNS server includes a numeric IP address. By contrast, the HTTPS request relied on from Forslow **lacks any domain name**.

The primary protocol used in the Domain Name System (DNS) is the User Datagram Protocol (UDP), which operates on port 53. UDP is a lightweight and efficient protocol that allows DNS queries and responses to be transmitted quickly without establishing a persistent connection. DNS requests are small and efficient, typically consisting of a query for a specific record type (e.g., A, AAAA, CNAME, MX) and a server response containing the requested information. Query latency is further reduced through caching at multiple levels, including local resolvers, ISP-level servers, and content delivery networks (CDNs) (see Ex. 2007)(see Ex. 2001 ¶40).

In contrast, HTTPS requests are designed for secure communication between a client, such as a web browser, and a server, using the HyperText Transfer Protocol over an encrypted SSL/TLS connection. HTTPS operates at the application layer and relies on TCP at the transport layer to ensure reliable data delivery, typically over port 443. The SSL/TLS handshake, which occurs at the start of an HTTPS session, establishes a secure channel by negotiating encryption algorithms, exchanging certificates, and deriving session keys. This process ensures confidentiality by encrypting data, integrity by detecting tampering, and authentication by verifying the server's identity. HTTPS requests involve more complex and voluminous data exchanges compared to DNS, encompassing HTTP headers, cookies, query parameters, and the encrypted payload, which can include sensitive user data or website content. This added complexity increases latency, as the TCP three-way

Patent Owner: Intellectual Ventures I LLC
Application No.: 90/019,519

handshake and the SSL/TLS setup introduce additional overhead (see Ex. 2008))(see Ex. 2001 ¶41).

Accordingly, as explained by Dr. Noubir “a POSITA would appreciate that HTTPS requests and DNS requests are completely different things. For example, the different types of requests operate on different ports, require/ don’t require a persistent connection, and perform completely different functions” (Ex. 2001, ¶42). Therefore, Forslow’s disclosure of a server that handles HTTPS requests is irrelevant to “a DNS server for the virtual network that is configured to receive a DNS request” and “return a network address associated with a network route director, a private network address associated with a second device in the virtual network, and a virtual network address associated with the second device,” as recited in the claim 30.

In fact, Forslow is wholly silent with regard to any mention of DNS. Instead, Forslow explains that “The virtual home agents 1 a and 1 b are using the lightweight directory access protocol (LDAP) 25 to retrieve the extranet workgroup configurations” (Forslow, Col. 10:57-60).

The Microsoft Computer Dictionary 5th Ed (Ex. 2002) defines a “Lightweight Directory Access Protocol” as:

A network protocol designed to work on TCP/IP stacks to extract information from a hierarchical directory such as X.500. This gives users a single tool to comb through data to find a particular piece of information, such as a user name, an e-mail address, a security certificate, or other contact information (Ex. 2002, Pg. 5).

Accordingly, LDAP is a protocol for accessing and managing directory information services over a network. It enables applications to query, search, and

Patent Owner: Intellectual Ventures I LLC
Application No.: 90/019,519

modify hierarchical directories that store data about users, groups, devices, or other organizational resources. Operating over TCP/IP, typically on port 389 for standard communication or port 636 for secure (SSL/TLS) communication, LDAP supports operations such as authentication, searching, and modifying entries. It is commonly used for user authentication, centralized access control, and resource management, integrating with systems like Microsoft Active Directory and OpenLDAP.

Therefore, as explained by Dr. Dr. Noubir “A POSITA would recognize that LDAP and DNS serve fundamentally different purposes and operate with distinct functionalities” (Ex. 2001, ¶44). Specifically, LDAP differs from DNS (in its purpose and functionality, focusing on managing and accessing directory-based information rather than resolving names to network addresses. LDAP is designed for centralized management of user accounts, groups, and permissions, enabling authentication and access control within enterprise environments. Unlike DNS, which primarily maps human-readable domain names to IP addresses for efficient routing, LDAP operates as a protocol for querying and modifying hierarchical directory structures, such as those found in Microsoft Active Directory or OpenLDAP. LDAP relies on TCP for reliable communication, typically over port 389 or port 636 for secure connections, and supports advanced features like secure authentication, role-based access, and integration with Single Sign-On (SSO) systems.

As a result, Forslow’s mobile service manager 7 is not “a DNS server for the virtual network that is configured to receive a DNS request” and “return a network address associated with a network route director, a private network address

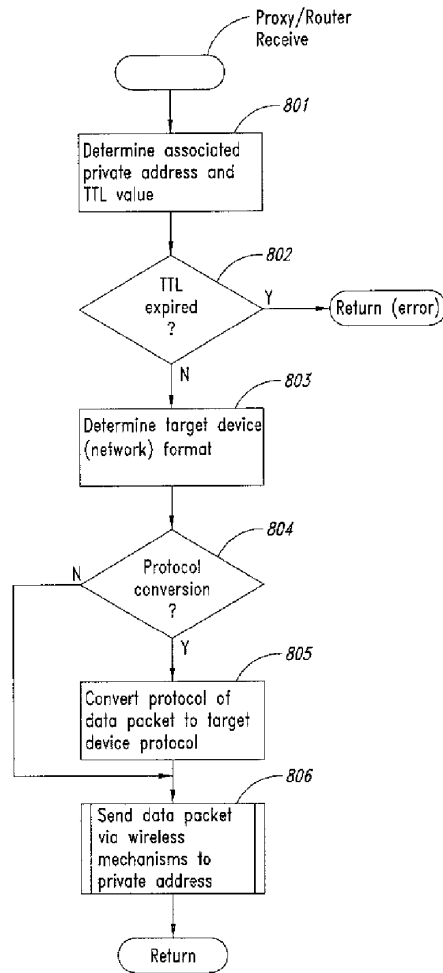
Patent Owner: Intellectual Ventures I LLC

Application No.: 90/019,519

associated with a second device in the virtual network, and a virtual network address associated with the second device,” as recited in claim 30 (Ex. 2001, ¶39). As a result, Forslow, alone or in combination with Leung, fails to teach or suggest every feature recited in claim 30.

Mehta does not cure the shortcomings of Forslow and Leung. The Office Action states, “to the extent it is determined that Forslow’s mobile service manager is not a DNS server, it would have been obvious to a POSITA to implement it as a DNS server, as disclosed in Mehta, with the functionality to return the same addresses in the triplet disclosed in Forslow, as well as the public IP address for the route director, as disclosed in Mehta” (Non-Final Office Action, Pg. 31).

The Office Action identifies that Mehta’s “FIG. 8 is a process flow diagram showing modifications to the server that can be used to provide a VPN client with the same enterprise address” (Non-Final Office Action, Pg. 23). However, Fig. 8 (reproduced below) shows a routine, implemented by an Address Proxy/Router within an Address Management Proxy System (AMPS), that handles data received from wired devices and facilitates communication with wireless devices by determining the private wireless address corresponding to the invoked public address. It verifies the validity of the address mapping using a TTL parameter, performs any necessary protocol or format conversions, and forwards the data to the wireless device’s private address, ensuring compatibility with the wireless network’s requirements (See Ex. 2001, ¶46).

Patent Owner: Intellectual Ventures I LLC**Application No.:** 90/019,519*Fig. 8*

However, in Mehta the sender only learned the address of the Proxy/Router, and it is the Proxy/Router that does the conversion. The sender is not aware of both the route director and the virtual/private address of the destination. Therefore, Mehta cannot “return a network address associated with a network route director, a private network address associated with a second device in the virtual network, and a virtual network address associated with the second device,” as recited in the Challenged Claims. Instead, Mehta teaches an architecture that is remarkably

Patent Owner: Intellectual Ventures I LLC
Application No.: 90/019,519

similar to the Dynamic DNS system illustrated in FIG. 3 (reproduced below) of the '785 Patent which is labeled as prior art (See Ex. 2001, ¶47).

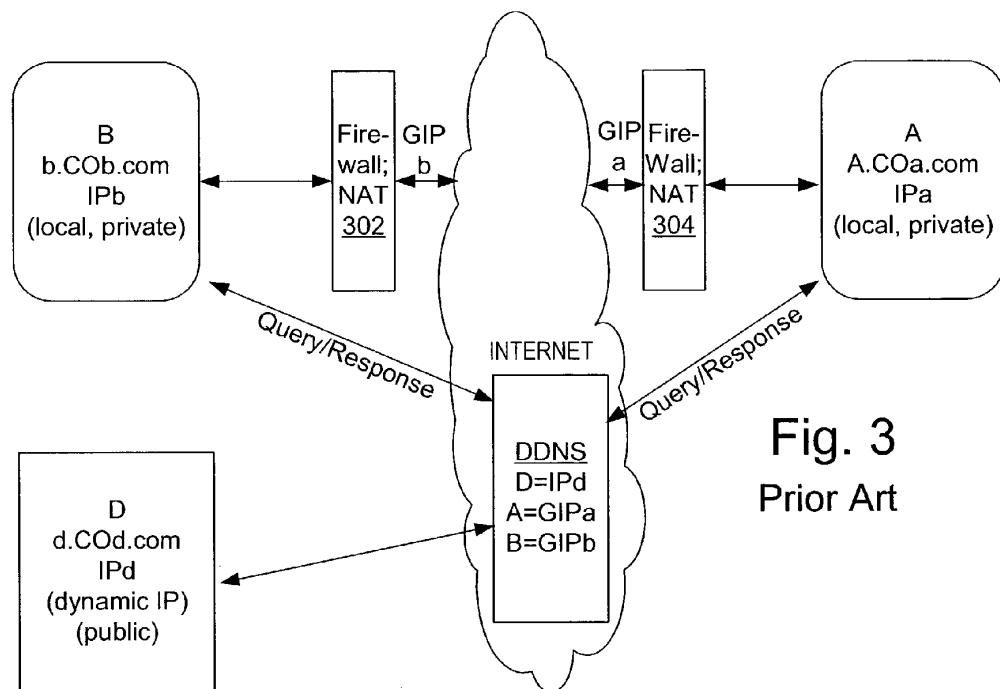


Fig. 3
Prior Art

In fact, providing all three addresses would completely defeat Mehta's purpose of reducing security risks. As explained by the Federal Circuit, "no suggestion to modify a prior art device where the modification would render the device inoperable for its intended purpose" (*Tec Air, Inc. v. Denso Mfg. ch., Inc.*, 192 F.3d 1353, 1360 (Fed. Cir. 1999)). Here, Mehta expressly states, "such addressing capability would expose each device to further security risks, because each such device is part of a publicly accessible network and it becomes more difficult to implement and enforce security measures" (Mehta, ¶025). Therefore, the Office Action's proposed

Patent Owner: Intellectual Ventures I LLC
Application No.: 90/019,519

modification would render the proposed combination inoperable for its intended purpose.

Instead, at most, Mehta allows a client to learn a temporary public IP address from a DNS server. The reference is wholly silent with regard to how to solve the problem of the current invention, which is how to route through a route director to the virtual address. Therefore, Mehta does not teach “a DNS server for the virtual network” that is configured to “return a network address associated with a network route director, a private network address associated with a second device in the virtual network, and a virtual network address associated with the second device,” as recited in the Challenged Claims.

Instead, the Office Action is impermissibly stitching together an obviousness rejection by not considering Mehta as a whole. Specifically, the Federal Circuit “explains that § 103 does not permit [an Examiner] to stitch together an obviousness finding from discrete portions of prior art references without considering the references as a whole” (*In re Enhanced Sec. Research, LLC*, 739 F.3d 1347, 1355 (Fed. Cir. 2014)). Accordingly, “[i]t is impermissible within the framework of section 103 to pick and choose from any one reference only so much of it as will support a given position, to the exclusion of other parts necessary to the full appreciation of what such reference fairly suggests to one of ordinary skill in the art” (*In re Wesslau*, 353 F.2d 238 (CCPA 1965)).

Here, Mehta makes it clear that a POSITA would not want to “return a network address associated with a network route director, a private network address

Patent Owner: Intellectual Ventures I LLC
Application No.: 90/019,519

associated with a second device in the virtual network, and a virtual network address associated with the second device,” as recited in claim 30 because sharing such information would compromise the security of the internal resources that Mehta is attempting to protect.

Therefore, Mehta, alone or in combination with Forslow and Leung, fails to teach or suggest every feature recited in claim 30. As a result, the Office Action’s reliance on Leung, Forslow, and Mehta to reject claim 30 fails to demonstrate that the cited references, individually or in combination, teach or suggest “a DNS server for the virtual network configured to receive a DNS request” and “return a network address associated with a network route director, a private network address associated with a second device in the virtual network, and a virtual network address associated with the second device,” as explicitly recited claim 30. Leung lacks disclosure of the DNS server functionality required by the claims, Forslow’s HTTPS-based communication does not equate to DNS operations, and Mehta’s proposed modifications contradict its stated purpose of maintaining security, rendering the combination inoperable for its intended purpose. Furthermore, the Office Action improperly isolates discrete portions of the references without considering them as a whole, as required under § 103.

Therefore, a *prima facie* case of obviousness has not been established because the combination of the cited references fails to yield the features recited in claim 30.

For at least the foregoing reasons, it is respectfully submitted that claim 30 is allowable over Mehta, Forslow, and Leung.

Patent Owner: Intellectual Ventures I LLC
Application No.: 90/019,519

Claim 34 is directly dependent upon claim 30, and the Patent Owner submits the claim is allowable over the cited references of record based on at least its dependency.

Based on at least the arguments presented above, the withdrawal of the 35 U.S.C. §103 rejections of the claims are respectfully requested.

C. The Office Action Improperly Conflates “Registration” and “Join” Processes, Failing to Establish That the Cited References Teach the Claimed Registration.

Claim 34, recites, in part:

wherein the register module is further configured to receive the registration request from the agent that is installed on the device for data communication via the virtual network.

None of the cited references teach or suggest a registration model that is “configured to receive the registration request from the agent that is installed on the device for data communication via the virtual network,” as recited in claim 34.

The Office Action states that “A POSITA would have understood that Leung’s VPN clients would send registration requests via an associated agent” (Non-Final Office Action Pg. 41). However, the Office Action appears to be conflating a “registration request¹” with a “join request.”

¹ The '785 Patent will expire is several weeks, on or about February 13, 2025. In a reexamination proceeding involving claims of an expired patent, claim construction is controlled by the principles set forth by the court in *Phillips v. AWH Corp.*, 415 F.3d 1303 (Fed. Cir. 2005). Under *Phillips*, where the specification reveals a special definition given to a claim term by the patentee that differs from the meaning it

Patent Owner: Intellectual Ventures I LLC
Application No.: 90/019,519

The '785 Patent expressly states:

Registration in this context means registering a machine using a Member Agent or a Group Agent with the VCN Manager. Users may be required to install the Member Agent software. ('785 Patent, 17:10-12)

The registration process is the first step that a user and machine go through with the VCN Manager before becoming a member of a virtual community. There are two phases to member registration: a registration request and a registration file request. Registration happens outside of the IPsec tunnels because at registration time no IPsec tunnel exists for the user registering ('785 Patent, 17:40-42).

Accordingly, the '785 Patent defines registration as the process by which the machine and the user are registered with the virtual community. Specifically, the Applicant has acted as their own lexicographer and provided an express definition of registration by stating, "Registration in this context means registering a machine using a Member Agent or a Group Agent with the VCN Manager."

In contrast, the '785 Patent explains the Join/ Leave Process as the following:

Once a user is registered, the user can join the VCN. Once joined, the user is a member of the VCN. Joining allows the user to let the VCN Manager and other members know that the user/member is on-line and available to communicate. The Join protocol runs between a Member Agent and the VCN Manager to establish a session key with the Manager, authenticate to the Manager, and make use of connections via the Route Directors in the community.

would otherwise possess, "the inventor's lexicography governs." Id., at 1316. Thus, to the extent the reexamination proceedings continue beyond February 13, 2025, claim terms such as "registration" must be construed as defined in the specification.

Patent Owner: Intellectual Ventures I LLC
Application No.: 90/019,519

The Member Join process is illustrated in FIG. 13. There are two phases to the Join operation: phase one is an Initialize Request and phase two is the actual Join. At join time, the IPsec tunnel for secure communication is not established, so the first few join packets are sent outside the IPsec tunnel ('785 Patent, 20:6-18).

Accordingly, the '785 Patent teaches that join/leave process is separate process from the registration process. Specifically, the join/leave process is utilized to establish the connection subsequent to the user and the machine being registered by the registration process.

The Office Action has proffered no evidence that Leung, Forslow, or Mehta, alone or in combination, teach "registering a machine using a Member Agent or a Group Agent with the VCN Manager." Instead, the Office Action cites to Leung's "access request" (Non-Final Office Action, Pg. 39), which is at most analogous to the "join" aspect of the join/leave process in the '785 Patent, as opposed to the registration process which is the subject of claim 34. Accordingly, the Office Action has not established a *prima facie* case of obviousness.

To establish a *prima facie* case of obviousness, "there must be some articulated reasoning with some rational underpinning to support the legal conclusion of obviousness" (*KSR Int'l v. Teleflex Inc.*, 127 S. Ct. 1727, 1741 (2007)). Therefore, the Federal Circuit Court of Appeals requires "record evidence to support an assertion that the structural features of claims ... were known prior art elements" (*K/S HIMPP v. Hear-Wear Techs., LLC*, 751 F.3d 1362, 1365 (Fed. Cir. 2014)). And although "'common sense' can be invoked, even potentially to supply a limitation missing from the prior art, it must still be supported by evidence and a reasoned explanation"

Patent Owner: Intellectual Ventures I LLC

Application No.: 90/019,519

(*Arendi S.A.R.L. v. Apple Inc.*, 832 F.3d 1355, 1363 (Fed. Cir. 2016)). As a result, a *prima facie* case of “[o]bviousness requires a suggestion of all limitations in a claim” (*CFMT, Inc. v. YieldUp Int’l Corp.*, 349 F.3d 1333, 1342 (Fed. Cir. 2003)).

Here, the Office Action has offered no evidence that the cited references teach or suggest a registration model that is “configured to receive the registration request from the agent that is installed on the device for data communication via the virtual network,” as recited in the Challenged Claim. As most, the Office Action has suggested that the cited references perform a join function.

Therefore, a *prima facie* case of obviousness has not been established because the combination of the cited references fails to yield the features recited in the claims.

For at least the foregoing reasons, it is respectfully submitted that claim 34 is allowable over Mehta, Forslow, and Leung.

Based on at least the arguments presented above, the withdrawal of the 35 U.S.C. §103 rejections of the claims are respectfully requested

Patent Owner: Intellectual Ventures I LLC
Application No.: 90/019,519

IV. Conclusion

In view of the foregoing, the Patent Owner respectfully request reconsideration. The Patent Owner submits that the U.S. Patent No. 7,949,785, including claims 1-90, is in condition for confirmation of patentability and a solicits a notice to that effect.

Respectfully submitted,

Intellectual Ventures I LLC

By /Brandon R. Theiss/
Brandon R. Theiss
Registration No. 70,507

EXHIBIT A-7

REMARKS

The above Amendments and these Remarks are in reply to the Office Action mailed September 22, 2006

Currently, claims 1-109 are pending. Claims 1-6, 8-16, 19, 20, 25, 26, 28, 29, 30-36, 38-54, 56-59, 62-64, 69, 73-75, 78-80, 84-87, 93, 100, 103, 104 and 106. No new matter is entered. Claim 101 is cancelled.

Claims 1-109 have been rejected under 35 U.S.C. 102 as being anticipated by Cheline et al. (US2003/0041136) (Cheline). Applicants respectfully traverse the rejection.

Cheline is concerned with automatically configuring a virtual private network (VPN). A VPN provides a secure tunnel between a server side system 130 and a client side system 108 (Fig. 1). In this approach, a service provider system 146 provides instructions to configure and control a modem 106 at the client side system 108 and a VPN concentrator 136 at the server side system 130 to provide a VPN tunnel between the modem and the VPN concentrator (par. 31, 33 and 42). The modem can also provide network address translation (NAT) (par. 47). This approach allows a number of client computers in a LAN 104 on the client side system 108 to communicate with the server side system 130 via the VPN tunnel.

However, Applicants' invention differs from Cheline's system in many important aspects. For example, claim 1 sets forth a virtual community network system which includes a virtual network manager including at least one virtual community definition comprising at least a domain name and a user set. Cheline resolves host names in a VPN using multiple DNS servers so that a client knows whether to communicate with an internal DNS Server, which contains private IP addresses, or an external DNS Server, which contains public IP addresses (par. 87 and 88). However, this is not the same as providing a virtual community definition, that is, a definition of a virtual community, which includes a domain name. The fact that Cheline resolves a host name is different than using a domain name as part of a definition of a virtual community.

Further, claim 1 sets forth that a virtual network manager provides virtual network addresses to users in the user set when the users join the virtual community, the users are identified by one another in the virtual community by their virtual network addresses and the virtual network addresses are not directly routable. In Cheline's system, a user's client computer uses either a corporate assigned IP address that can be routed within the server side system, or a private non-routable IP address which is translated using NAT to assume a corporate assigned IP

address (par. 51). The corporate assigned IP address that can be routed is not a virtual network address that is not directly routable. Further, the private non-routable IP address is not a virtual address by which users are identified by one another in a virtual community. Moreover, there is no indication that such an IP address is provided by a virtual network manager when the user joins a virtual community.

In view of the above, claim 1 is clearly patentable over the cited reference.

Further, the dependent claims of claim 1 provide further patentable features. For example, claim 5 sets forth that a virtual network manager detects a presence of a NAT device when an agent installed in a private network on a processing device behind the NAT device contacts the virtual network manager in an attempt to join the virtual community. Even if, arguendo, the modem 106 of Cheline is a NAT device and the VPN provider can receive communications from the NAT device, the VPN provider does not know, and does not detect, if the communication is from behind the NAT device.

The specific technique used to detect the presence of a NAT device as set forth in claim 29 similarly is neither disclosed nor suggested by Cheline.

Further, claim 8 sets forth that processing devices of the users in the virtual community are configured with both a virtual network address and a physical address. In contrast, Cheline indicates that a user's client computer uses either a corporate assigned IP address or a private non-routable IP address (par. 51). There is no indication that the user's client computer is configured with a virtual network address and a physical address, e.g., where the users are identified by one another in the virtual community by their virtual network addresses. Cheline does not provide both types of addresses to the clients 102.

Claim 12 sets forth details of how a first NAT device associated with a first processing device on a first private network transmits packets to a second processing device. The first NAT device transmits packets to a route director, and the packets include encapsulated packets provided by the first processing device in which a source address is the virtual network address of the first processing device, a destination address is the virtual network address of the second processing device and a shim includes the physical address of the first processing devices as a private network address in the first private network. Cheline fails to disclose or suggest this feature.

Claim 15, which depends on claim 12, sets forth that the second processing device is coupled to a second private network and accesses a public network via a second NAT device. In this case, the shim also includes the physical address of the second processing devices as a private network address in the second private network. Cheline is not concerned with processing devices in different private networks communicating in this manner. Instead, Cheline is concerned with clients 102 on a common LAN at client side system 108 communicating with a server side system 130 via a VPN. The clients 102 on the LAN do not communicate with clients on another private network

The remaining dependent claims of claim 1 are similarly patentable over Cheline.

Regarding independent claim 32, the client 102 of Cheline can hardly be considered to be a management device for a virtual community since the client does not perform any managing as claimed. Further, while each client can have an IP address which can be routed through the server side system (par. 51), the clients do not include a network interface coupled to a public address realm. Moreover, the Examiner asserts that the service provider system 146 is a network traffic router as claimed. However, the service provider system 146 is used for providing instructions to configure and control a modem 106 at the client side system 108 and a VPN concentrator 136 at the server side system 130 to provide a VPN tunnel (par. 31, 33 and 42). There is no disclosure or suggestion that the service provider system 146 routes traffic between members of a virtual community as claimed. Moreover, there is no need for the service provider system to route traffic as claimed since the client side system and the server side system communicate directly via the VPN tunnel.

Claim 32 is therefore clearly patentable over Cheline.

Further, the dependent claims of claim 32 provide additional patentable features. For example, regarding claim 35, the client of Cheline is not a management device with a TCP port capable of communicating network traffic in a virtual community to allow members of the virtual community to register and join in the virtual community. The client of Cheline is also not involved with allowing other members of the virtual community to register and join in the virtual community.

Regarding claim 41, Cheline at par. 67 and 68 refers to a client requesting the initiation of a VPN session. However, Cheline does not disclose or suggest that a management device includes a member join module which allows a particular member to notify the management

device via a join request that the particular member is available to participate in the virtual community, and to notify the management device via a leave request that the particular member is no longer available to participate in the virtual community.

Regarding claim 44, Cheline resolves host names in a VPN using multiple DNS servers so that a client knows whether to communicate with an internal DNS Server or an external DNS Server (par. 87 and 88). However, Cheline does not disclose or suggest the features of claim 44 as amended. For example, the client 102(1) of Cheline does not communicate with the DNS server 120 to obtain a virtual network address and private network address of another client such as client 102(2) and to obtain a public network address of the service provider system 146, which is asserted by the Examiner to be a network traffic router as claimed.

The remaining dependent claims of claim 32 are similarly patentable over Cheline.

Regarding independent claim 53, Cheline at par. 86 states that multiple VPNs can be established over a single modem, thus avoiding the need for multiple modems. Cheline indicates that previously, separate modems and telephone lines were need for two clients to communicate via separate VPN tunnels. For example, a husband would communicate with his office via one modem and VPN tunnel, and a wife would communicate with her office via another modem and VPN tunnel (par. 7). Thus, the husband and wife as clients would not be part of the same community. In contrast, claim 53 sets forth routing communications between first and second community members in a community.

Claim 53 and its dependent claims are therefore clearly patentable over Cheline.

Independent claims 64 sets forth, in part, assigning members a respective virtual address, where the virtual addresses are not directly routable, and routing network traffic between the members in a virtual community. Cheline at par. 55 refers to using a DHCP server to allow network administrators to assign IP addresses to client computers. However, these addresses are not virtual addresses which are not directly routable.

Claim 64 and its dependent claims are therefore clearly patentable over Cheline.

Independent claim 80 sets forth, in part, managing a virtual community network realm, defining a member set of users entitled to communicate in the virtual community, and registering users with the virtual community to allow the users to communicate with one another. In contrast, as discussed, Cheline is only concerned with enabling clients to communicate via one or more VPN tunnels with a server, but not with other clients (par. 67).

Claim 80 and its dependent claims are therefore clearly patentable over Cheline.

Claim 93 sets forth, in part, a virtual network manager having a network interface coupled to a network, the manager including at least one virtual community definition comprising at least a domain name and a user set. As discussed above in connection with claim 1, Cheline resolves host names in a VPN using multiple DNS servers so that a client knows whether to communicate with an internal DNS Server or an external DNS Server (par. 87 and 88). However, this is not the same as providing a virtual community definition, that is, a definition of a virtual community, which includes a domain name. The fact that Cheline resolves a host name is different than using a domain name as part of a definition of a virtual community.

Claim 93 and its dependent claims are therefore clearly patentable over Cheline.

Based on the above amendments and these remarks, reconsideration of claims is respectfully requested.

The Examiner's prompt attention to this matter is greatly appreciated. Should further questions remain, the Examiner is invited to contact the undersigned attorney by telephone.

The Commissioner is authorized to charge any underpayment or credit any overpayment to Deposit Account No. 501826 for any matter in connection with this response, including any fee for extension of time, which may be required.

Respectfully submitted,

Date: March 22, 2007

By: /Ralph F. Hoppin/
Ralph F. Hoppin
Reg. No. 38,494

VIERRA MAGEN MARCUS & DENIRO LLP
575 Market Street, Suite 2500
San Francisco, California 94105-4206
Telephone: (415) 369-9660
Facsimile: (415) 369-9665

EXHIBIT A-8

US008407722B2

(12) **United States Patent**
Tuttle et al.(10) **Patent No.:** **US 8,407,722 B2**
(45) **Date of Patent:** **Mar. 26, 2013**(54) **ASYNCHRONOUS MESSAGING USING A
NODE SPECIALIZATION ARCHITECTURE
IN THE DYNAMIC ROUTING NETWORK**5,699,523 A 12/1997 Li et al.
5,706,516 A 1/1998 Chang et al.
5,732,219 A 3/1998 Blumer et al.
5,754,939 A 5/1998 Herz et al.

(Continued)

(75) Inventors: **Timothy Tuttle**, San Francisco, CA
(US); **Karl E. Rumelhart**, Palo Alto,
CA (US)

FOREIGN PATENT DOCUMENTS

(73) Assignee: **Shaw Parsing L.L.C.**, Las Vegas, NV
(US)EP 0 733 983 A2 9/1996
EP 0 749 081 A1 12/1996

(Continued)

(*) Notice: Subject to any disclaimer, the term of this
patent is extended or adjusted under 35
U.S.C. 154(b) by 1127 days.

OTHER PUBLICATIONS

Franklin et al., "Dissemination-Based Information Systems," IEEE
Data Engineering Bulletin, vol. 19, No. 3, Sep. 1996, 9 pages.(21) Appl. No.: **11/396,251**

(Continued)

(22) Filed: **Mar. 30, 2006**(65) **Prior Publication Data**

US 2007/0239822 A1 Oct. 11, 2007

Primary Examiner — Andy Ho*Assistant Examiner* — Tuan Dao(74) *Attorney, Agent, or Firm* — Sterne, Kessler, Goldstein
& Fox P.L.L.C.**Related U.S. Application Data**(63) Continuation of application No. 10/105,018, filed on
Mar. 21, 2002, now Pat. No. 7,051,070, which is a
continuation-in-part of application No. 10/017,182,
filed on Dec. 14, 2001, now Pat. No. 7,043,525.

(Continued)

(51) **Int. Cl.****G06F 3/00** (2006.01)**G06F 15/16** (2006.01)(52) **U.S. Cl.** **719/316; 709/206**(58) **Field of Classification Search** **709/203,**
709/219; 719/309

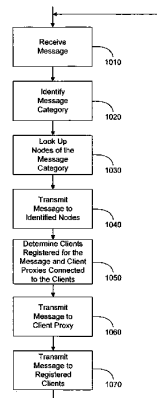
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,230,048 A * 7/1993 Moy 707/1
5,535,335 A 7/1996 Cox et al.
5,692,193 A 11/1997 Jagannathan et al.(57) **ABSTRACT**

A network routes update messages containing updates to properties of live objects from input sources to clients having the objects. When the clients receive live objects, the clients identify the object IDs associated with the objects and register the object IDs with the routing network. The routing network is adapted to selectively send update messages to nodes in the network and the nodes forward the messages to the clients. One implementation uses a hierarchy of registries to indicate which nodes and clients receive which update messages. Another implementation assigns update messages to one or more of N categories and nodes to one or more of M types, and the gateways maintain mapping between categories and types. To ensure that clients receive all of the update messages for which they register, the clients connect to client proxies that in turn connect to at least one node of each type.

37 Claims, 12 Drawing Sheets

US 8,407,722 B2

Page 2

Related U.S. Application Data

- (60) Provisional application No. 60/256,613, filed on Dec. 18, 2000, provisional application No. 60/276,847, filed on Mar. 16, 2001, provisional application No. 60/278,303, filed on Mar. 21, 2001, provisional application No. 60/279,608, filed on Mar. 28, 2001, provisional application No. 60/280,627, filed on Mar. 29, 2001.

(56) References Cited

U.S. PATENT DOCUMENTS

5,822,543 A 10/1998 Dunn et al.
 5,845,324 A 12/1998 White et al.
 5,878,420 A 3/1999 de la Salle
 5,886,643 A 3/1999 Diebboll et al.
 5,919,247 A 7/1999 Van Hoff et al.
 5,933,429 A 8/1999 Bubenik et al.
 5,938,733 A 8/1999 Heimsoth et al.
 5,964,839 A 10/1999 Johnson et al.
 5,974,457 A 10/1999 Wacławsky et al.
 6,018,619 A 1/2000 Allard et al.
 6,029,175 A 2/2000 Chow et al.
 6,052,447 A 4/2000 Golden et al.
 6,055,493 A 4/2000 Ries et al.
 6,091,724 A * 7/2000 Chandra et al. 370/390
 6,094,681 A 7/2000 Shaffer et al.
 6,112,240 A 8/2000 Pogue et al.
 6,138,158 A 10/2000 Boyle et al.
 6,173,406 B1 1/2001 Wang et al.
 6,233,600 B1 5/2001 Salas et al.
 6,240,451 B1 5/2001 Campbell et al.
 6,253,167 B1 6/2001 Matsuda et al.
 6,256,747 B1 7/2001 Inohara et al.
 6,292,835 B1 9/2001 Huang et al.
 6,308,209 B1 10/2001 Lecheler
 6,314,459 B1 * 11/2001 Freeman 709/220
 6,324,587 B1 * 11/2001 Trenbeath et al. 719/310
 6,363,421 B2 3/2002 Barker et al.
 6,366,926 B1 * 4/2002 Pohlmann et al. 707/104.1
 6,405,245 B1 6/2002 Burson et al.
 6,408,282 B1 6/2002 Buist
 6,418,448 B1 7/2002 Sarkar
 6,418,467 B1 7/2002 Schweitzer et al.
 6,446,257 B1 9/2002 Pradhan et al.
 6,449,638 B1 9/2002 Wecker et al.
 6,460,036 B1 10/2002 Herz
 6,484,143 B1 11/2002 Swildens et al.
 6,502,131 B1 12/2002 Vaid et al.
 6,510,323 B1 * 1/2003 Stocker et al. 455/466
 6,539,427 B1 3/2003 Natarajan et al.
 6,553,413 B1 4/2003 Leighton et al.
 6,560,611 B1 5/2003 Nine et al.
 6,567,411 B2 5/2003 Dahlen
 6,577,328 B2 6/2003 Matsuda et al.
 6,606,596 B1 8/2003 Zirngibl et al.
 6,606,643 B1 8/2003 Emens et al.
 6,609,138 B1 8/2003 Merriam
 6,654,804 B1 11/2003 Fleming
 6,658,652 B1 12/2003 Alexander et al.
 6,687,729 B1 2/2004 Sievert et al.
 6,691,165 B1 2/2004 Bruck et al.
 6,725,446 B1 4/2004 Hahn et al.
 6,728,747 B1 4/2004 Jenkins et al.
 6,751,663 B1 6/2004 Farrell et al.
 6,760,324 B1 7/2004 Scott et al.
 6,769,009 B1 7/2004 Reisman
 6,789,115 B1 9/2004 Singer et al.
 6,792,458 B1 9/2004 Muret et al.
 6,829,642 B1 12/2004 Giroir et al.
 6,832,222 B1 12/2004 Zimowski
 6,836,886 B2 12/2004 Tuerke et al.
 6,871,346 B1 3/2005 Kumbalimutt et al.
 6,918,084 B1 7/2005 Slaughter et al.
 6,970,924 B1 11/2005 Chu et al.
 7,020,082 B2 3/2006 Bhagavath et al.

7,043,525 B2 5/2006 Tuttle et al.
 7,051,070 B2 5/2006 Tuttle et al.
 7,107,326 B1 9/2006 Fijolek et al.
 7,127,720 B2 10/2006 Cano et al.
 7,139,844 B2 11/2006 Smith et al.
 7,159,034 B1 1/2007 Rai
 7,207,043 B2 4/2007 Blythe et al.
 7,209,959 B1 * 4/2007 Campbell et al. 709/219
 7,249,197 B1 7/2007 Roostenburg et al.
 7,263,547 B2 8/2007 Kloba et al.
 7,277,917 B2 10/2007 Tuttle et al.
 7,293,074 B1 * 11/2007 Jellinek et al. 709/218
 7,350,213 B2 3/2008 Deutesfeld et al.
 7,412,518 B1 8/2008 Duigou et al.
 7,426,721 B1 9/2008 Saulpaugh et al.
 7,430,610 B2 9/2008 Pace et al.
 7,516,177 B2 4/2009 Knapp et al.
 7,565,359 B2 7/2009 Nazem et al.
 2001/0012299 A1 8/2001 Dahlen
 2001/0047426 A1 11/2001 Hunter
 2002/0010757 A1 1/2002 Granik et al.
 2002/0013852 A1 1/2002 Janik
 2002/0024536 A1 2/2002 Kahan et al.
 2002/0056004 A1 5/2002 Smith et al.
 2002/0073165 A1 6/2002 McNulty et al.
 2002/0078251 A1 6/2002 Lewis
 2002/0087630 A1 * 7/2002 Wu 709/203
 2002/0095399 A1 7/2002 Devine et al.
 2002/0120717 A1 * 8/2002 Giotta 709/219
 2002/0165977 A1 * 11/2002 Novaes 709/238
 2003/0026254 A1 2/2003 Sim
 2003/0041110 A1 2/2003 Wenocur et al.
 2003/0120817 A1 6/2003 Ott et al.
 2003/0140111 A1 7/2003 Pace et al.
 2004/0139433 A1 7/2004 Blythe et al.
 2004/0148606 A1 7/2004 Hosoe
 2004/0199926 A1 10/2004 Gilgen et al.
 2004/0215493 A1 10/2004 Koppes et al.
 2005/0027815 A1 2/2005 Christodoulou et al.
 2005/0033841 A1 2/2005 McCarthy et al.
 2005/0125557 A1 6/2005 Vasudevan et al.
 2005/0278726 A1 12/2005 Cano et al.
 2006/0031282 A1 2/2006 Tuttle et al.
 2006/0031283 A1 2/2006 Tuttle et al.
 2006/0041681 A1 2/2006 Rumelhart
 2006/0075279 A1 4/2006 Cameros et al.
 2006/0117318 A1 6/2006 Rumelhart et al.
 2006/0265488 A1 11/2006 Tuttle et al.
 2007/0033293 A1 2/2007 Rumelhart
 2007/0050519 A1 3/2007 Cano et al.
 2007/0061811 A1 3/2007 Rumelhart et al.
 2009/0077173 A1 3/2009 Lowery et al.

FOREIGN PATENT DOCUMENTS

EP 0 889 421 A1 1/1999
 WO WO 97/16796 A1 5/1997
 WO WO 01/63837 A2 8/2001
 WO WO 2005/046184 A1 5/2005

OTHER PUBLICATIONS

Nagami et al., "Toshiba's Flow Attribute Notification Protocol (FANP) Specification," Apr. 1997, RFC 2129, Internet RFC/STD/FYI/BCP Archives [online], [retrieved on May 16, 2002]. Retrieved from the Internet: <landfield.com/rfcs/rfc2129.html>, 16 pages.
 Strom et al., "Gryphon: An Information Flow Based Approach to Message Brokering," International Symposium on Software Reliability Engineering '98, 1998, 2 pages.
 Sturman et al., "Reflection in the Gryphon Message Brokering System," Reflection Workshop of the 13.sup.th ACM Conference on Object Oriented Program Systems, Languages and Applications (OOPSLA '98), 1998, 5 pages.
 International DOI Foundation, "Introduction to the Digital Object Identifier," [online]. Apr. 1998 [retrieved on May 16, 2002]. Retrieved from the Internet: <doi.org/introduction.html>, 4 pages.
 Aksoy et al., "Research in Data Broadcast and Dissemination", Proc. 1st Int'l Conf. on Advanced Multimedia Content Processing, Osaka University, Osaka, Japan, Nov. 1998.

US 8,407,722 B2

Page 3

Banavar et al., "An Efficient Multicast Protocol for Content-Based Publish-Subscribe Systems," Proc. of the 19th International Conference on Distributed Computing Systems, 1999, 9 pages.

Banavar et al., "Information Flow Based Event Distribution Middleware," Proceedings of the 1999 ICDCS Workshop on Electronic Commerce and Web-Based Applications, 1999, 8 pages.

Aguilera et al., "Matching Events in a Content-based Subscription System," Eighteenth ACM Symposium on Principles of Distributed Computing (PODC '99), Atlanta, GA, May 4-6, 1999, 9 pages.

Banavar et al., "A Case for Message Oriented Middleware," Distributed Computing, 13.sup.th International Symposium, Bratislava, Slovak Republic, Sep. 27-29, 1999, 18 pages.

Aguilera et al., "Efficient Atomic Broadcast Using Deterministic Merge," Symposium on Principles of Distributed Computing, 2000, 10 pages.

Opyrchal et al., "Exploiting IP Multicast in Content-Based Publish-Subscribe Systems," Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2000), Apr. 2000, 23 pages.

Caplin Systems Ltd., White Paper "Real Time Text Protocol (RTTP)," Version 1.1, Sep. 2000, 11 pages.

Reuters, "Reuters Market Data Systems and the Trading Solutions Architecture," Version 1.0, Jan. 12, 2001, 51 pages.

Ramamrithan et al., "Dissemination of Dynamic Data on the Internet," [online]. Powerpoint Presentation, Spring 2001, [retrieved on Feb. 6, 2002], 5 pages. Retrieved from the Internet <cs.umbc.edu/courses/graduate/CMSC691T/spring.sub.—2001/rlist/amit.ppt>.

ComputerLetter, vol. 17, No. 23, Jul. 16, 2001, pp. 1-8.

ComputerLetter, vol. 17, No. 31, Sep. 24, 2001, pp. 1-6.

ComputerLetter, vol. 17, No. 35, Nov. 5, 2001, pp. 1-6.

Tuttle et al., "Upstream Delivery of Information in a Digital Network", U.S. Appl. No. 09/901,582, filed Jul. 9, 2001.

"Repackaging the Net", ComputerLetter, vol. 17, No. 35, Nov. 5, 2001, pp. 1-5.

"Reckoning with IP", ComputerLetter, vol. 17, No. 37, Nov. 19, 2001, pp. 1-6.

"Persistence Counts", ComputerLetter, vol. 17, No. 23, Jul. 16, 2001, pp. 1, 5-7.

Zhao et al., "A Workflow-centric Study of Organizational Knowledge Distribution," Proceedings of the 33rd Hawaii International Conference on System Sciences; 2000; pp. 1-10; IEEE.

Gribble, et al., "The Ninja Architecture for Robust Internet-scale Systems and Services," Computer Networks; 2001; pp. 473-497; vol. 35; Elsevier Science B.V.

Carmona, David; "Programming the Thread Pool in the .NET Framework"; 'Online' Jun. 2002, pp. 1-17, XP002357234; retrieved on Dec. 1, 2005 from the Internet: URL: <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndotnet/html/progthreepool.asp>, pp. 1-17.

Welsh, Matthew D.; "An Architecture for Highly Concurrent, Well-Conditioned Internet Services"; URL: <http://www.eecs.harvard.edu/{mdw/papers/mdw-phdthesis/pdf>, 2005, pp. 48-54, 101, and 113-114.

Written Opinion of the International Searching Authority for International Application No. PCT/US2005/029162, recorded Jan. 17, 2006, 7 pages.

Written Opinion of the International Searching Authority for International Application No. PCT/US2005/029021, recorded Dec. 14, 2005, 10 pages.

Written Opinion of the International Searching Authority for International Application No. PCT/US2005/029158, recorded Jan. 25, 2006, 7 pages.

Non-Final Office Action dated Oct. 26, 2009, U.S. Appl. No. 11/205,263, Rumelhart et al., filed Aug. 15, 2005.

Final Office Action dated Nov. 24, 2009, U.S. Appl. No. 11/205,237, Rumelhart et al., filed Aug. 15, 2005.

Non-Final Office Action dated May 3, 2005, 8 pages in U.S. Appl. No. 10/017,182, Tuttle et al., filed Dec. 14, 2001.

Notice of Allowance dated Jan. 3, 2006, U.S. Appl. No. 10/105,018, Tuttle et al., filed Mar. 21, 2002.

Notice of Allowance dated Jul. 5, 2006, U.S. Appl. No. 10/105,018, Tuttle et al., filed Mar. 21, 2002.

Non-Final Office Action dated Aug. 9, 2005, U.S. Appl. No. 10/213,269, Cano et al., filed Aug. 5, 2002.

Final Office Action dated Jan. 26, 2006, U.S. Appl. No. 10/213,269, Cano et al., filed Aug. 5, 2002.

Notice of Allowance dated Jun. 6, 2006, U.S. Appl. No. 10/213,269, Cano et al., filed Aug. 5, 2002.

Non-Final Office Action dated Feb. 3, 2009, U.S. Appl. No. 11/205,233, Rumelhart et al., filed Aug. 15, 2005.

Non-Final Office Action dated Aug. 6, 2009, U.S. Appl. No. 11/205,233, Rumelhart et al., filed Aug. 15, 2005.

Non-Final Office Action dated Apr. 22, 2008, U.S. Appl. No. 11/205,237, Cameros et al., filed Aug. 15, 2005.

Non-Final Office Action dated Apr. 29, 2009, U.S. Appl. No. 11/205,237, Cameros et al., filed Aug. 15, 2005.

Final Office Action dated Nov. 25, 2008, U.S. Appl. No. 11/205,237, Cameros et al., filed Aug. 15, 2005.

Notice of Allowance dated May 22, 2007, U.S. Appl. No. 11/347,802, Tuttle et al., filed Feb. 3, 2006.

Non-Final Office Action dated Jan. 26, 2007, U.S. Appl. No. 11/347,802, Tuttle et al., filed Feb. 3, 2006.

Non-Final Office Action dated Mar. 6, 2009, U.S. Appl. No. 11/515,233, Rumelhart et al., filed Aug. 31, 2006.

Final Office Action dated Oct. 7, 2009, U.S. Appl. No. 11/515,233, Rumelhart et al., filed Aug. 31, 2006.

Non-Final Office Action dated Jan. 7, 2010, U.S. Appl. No. 11/205,233, Rumelhart et al., filed Aug. 15, 2005.

Non-Final Office Action dated Feb. 22, 2010, U.S. Appl. No. 11/515,233, Rumelhart et al., filed Aug. 31, 2006.

Final Office Action dated Mar. 23, 2010, U.S. Appl. No. 11/205,263, Rumelhart et al., filed Aug. 15, 2005.

U.S. Appl. No. 13/617,168, Tuttle et al., "Asynchronous Messaging Using A Node Specialization Architecture in the Dynamic Routing Network," filed Sep. 14, 2012.

Non-Final Rejection mailed Aug. 9, 2005 for U.S. Appl. No. 10/213,269, filed Aug. 5, 2002; 11 pages.

Final Rejection mailed Jan. 26, 2006 for U.S. Appl. No. 10/213,269, filed Aug. 5, 2002; 8 pages.

Notice of Allowance mailed Jun. 6, 2006 for U.S. Appl. No. 10/213,269, filed Aug. 5, 2002; 4 pages.

Non-Final Rejection mailed Aug. 26, 2010 for U.S. Appl. No. 11/515,366, filed Aug. 31, 2006; 16 pages.

Final Rejection mailed Feb. 17, 2011 for U.S. Appl. No. 11/515,366, filed Aug. 31, 2006; 30 pages.

Notice of Allowance mailed Nov. 10, 2005 for U.S. Appl. No. 10/017,182, filed Dec. 14, 2001; 6 pages.

* cited by examiner

U.S. Patent

Mar. 26, 2013

Sheet 1 of 12

US 8,407,722 B2

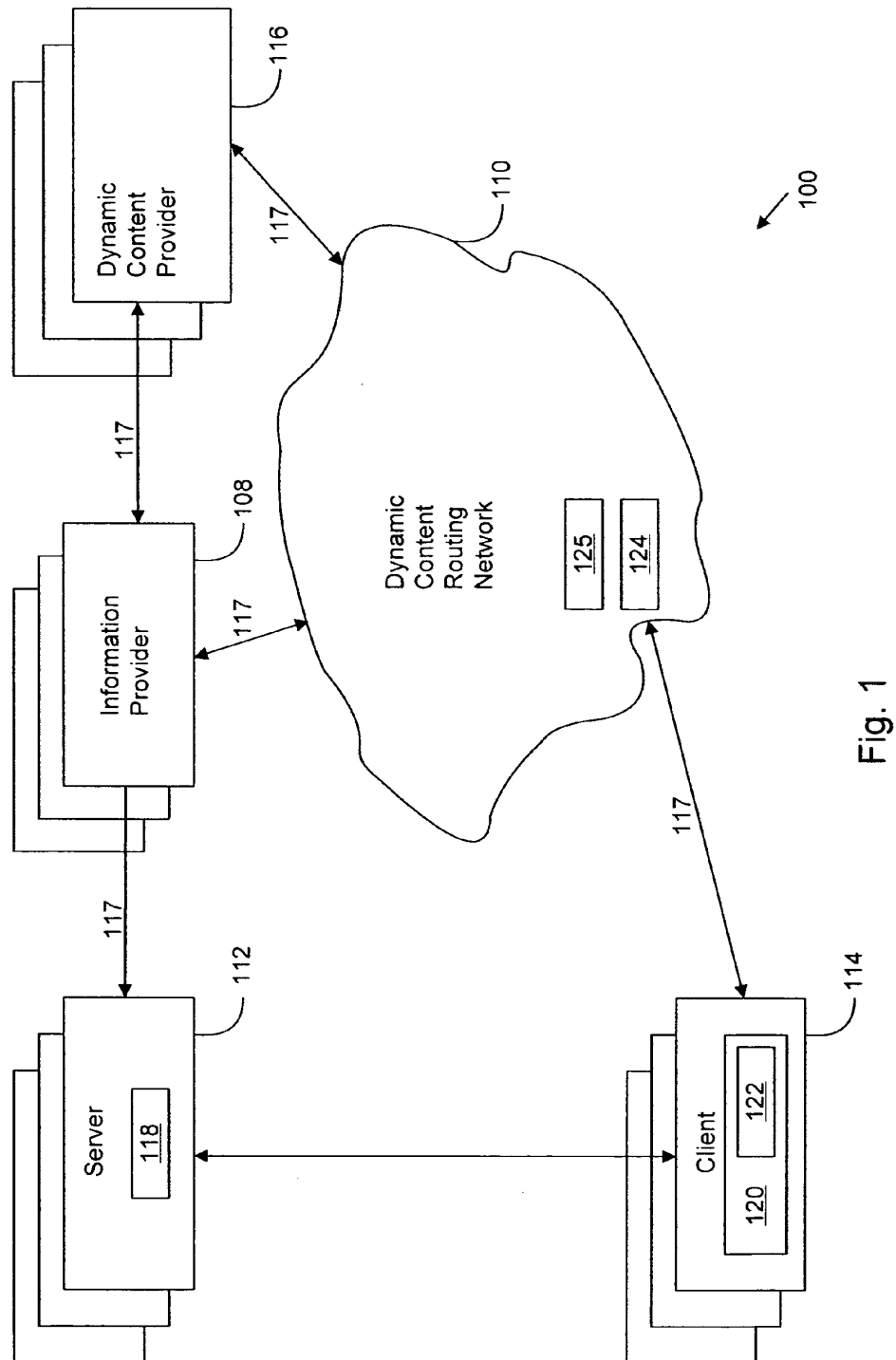


Fig. 1

U.S. Patent

Mar. 26, 2013

Sheet 2 of 12

US 8,407,722 B2

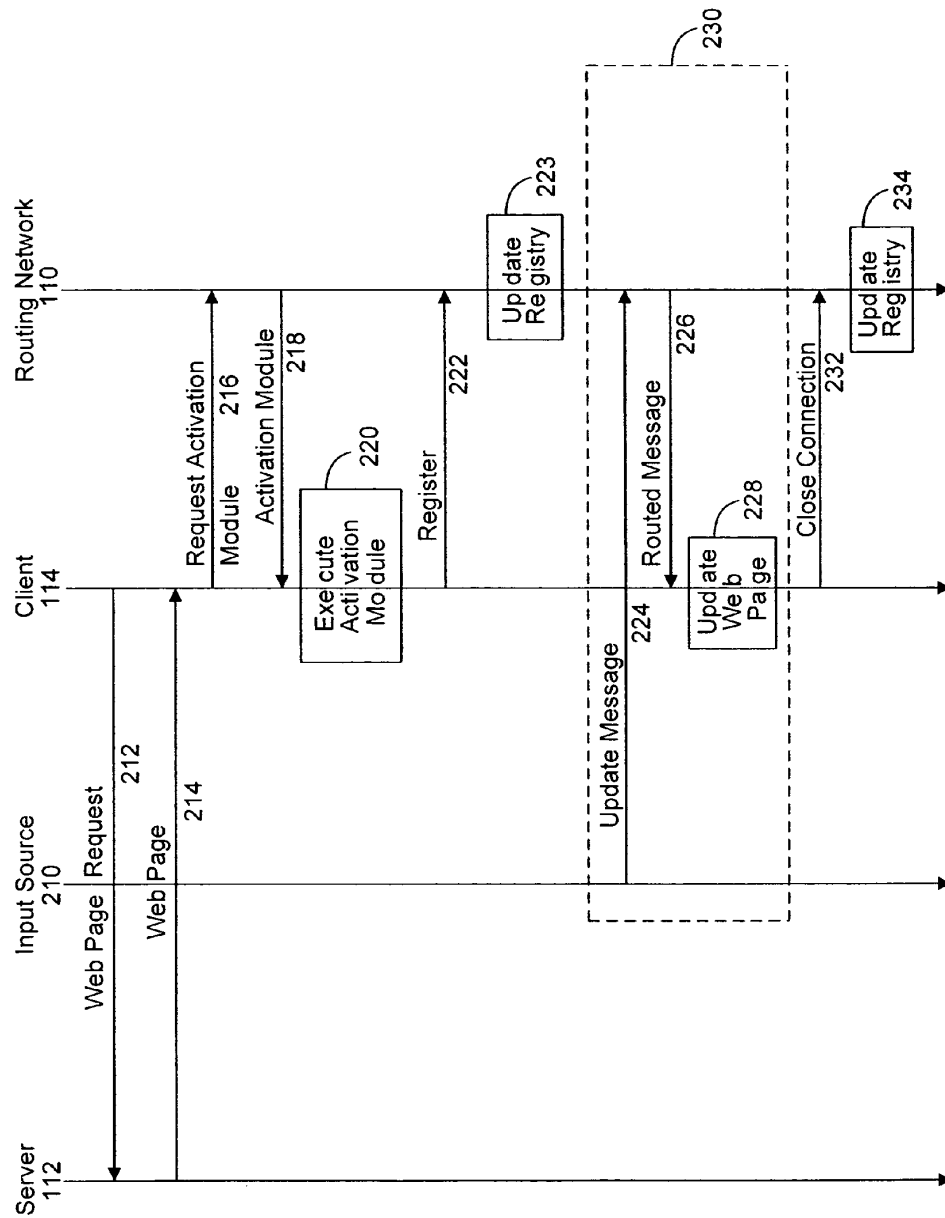


Fig. 2

U.S. Patent

Mar. 26, 2013

Sheet 3 of 12

US 8,407,722 B2

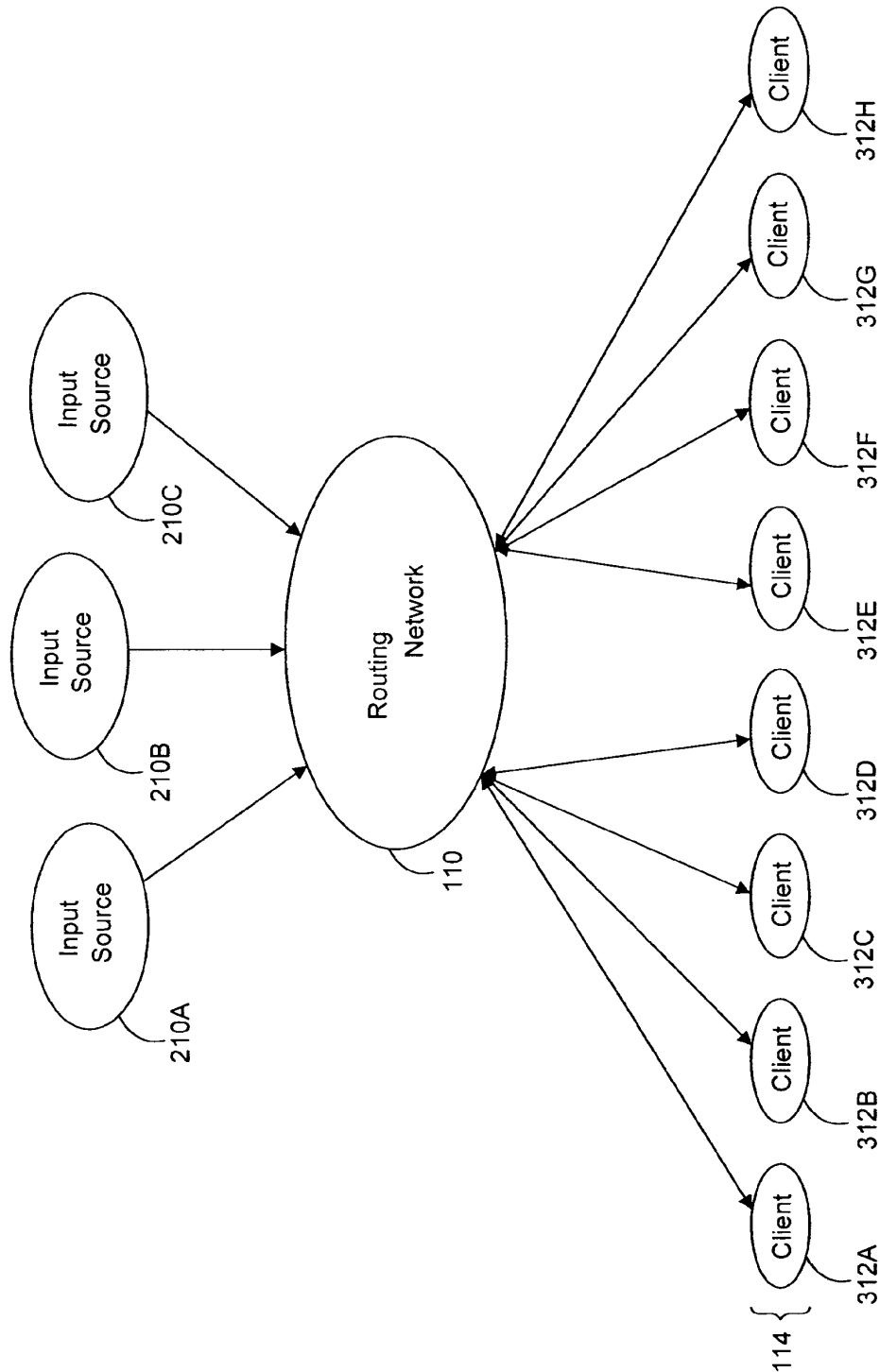


Fig. 3

U.S. Patent

Mar. 26, 2013

Sheet 4 of 12

US 8,407,722 B2

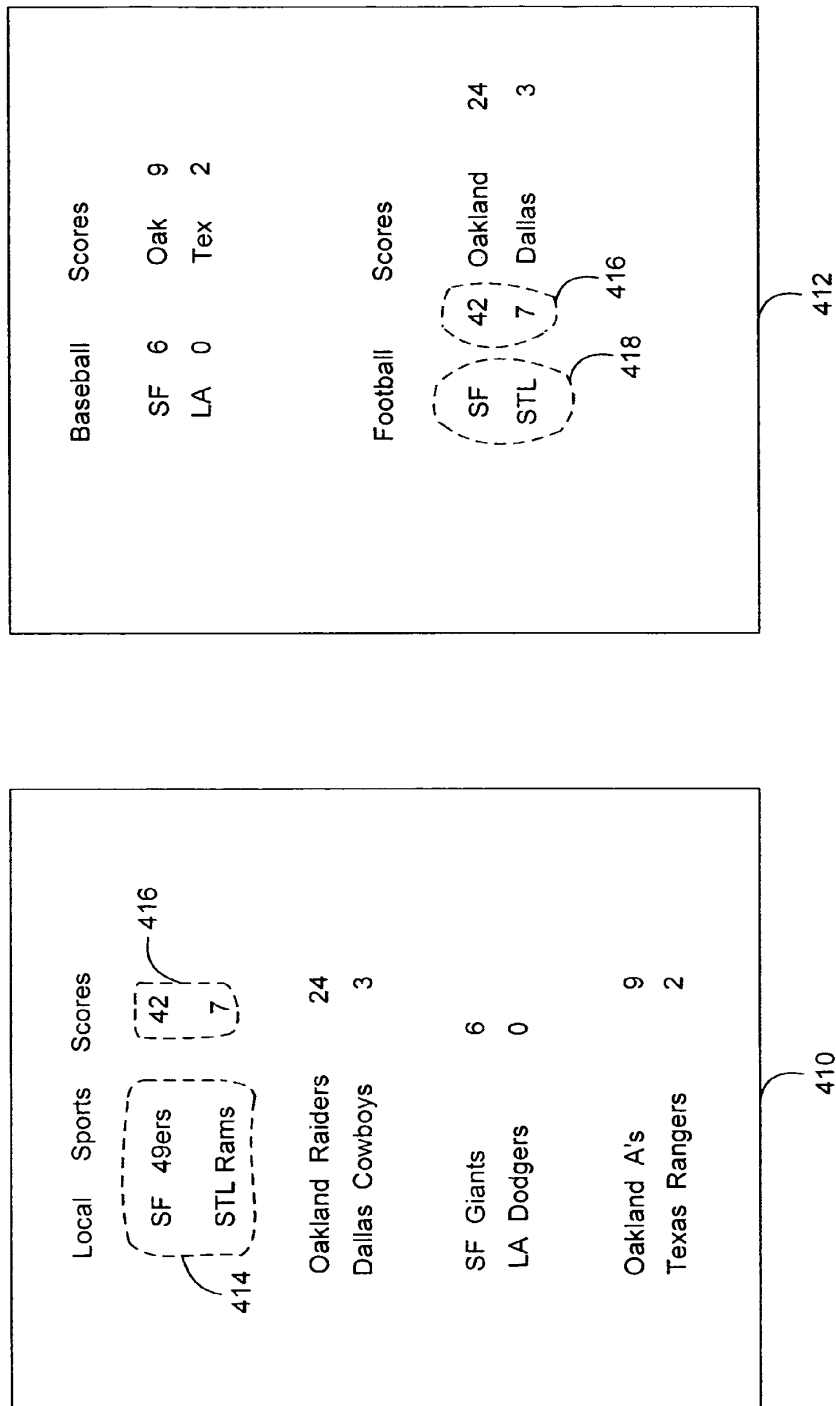


Fig. 4

U.S. Patent

Mar. 26, 2013

Sheet 5 of 12

US 8,407,722 B2

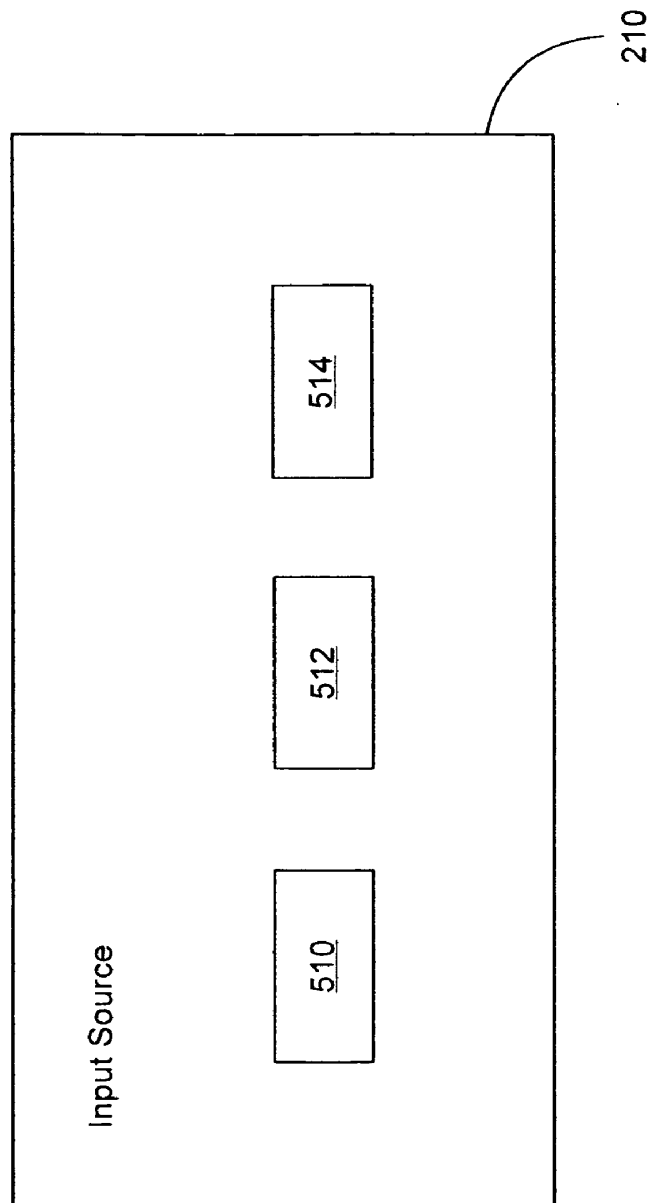


Fig. 5

U.S. Patent

Mar. 26, 2013

Sheet 6 of 12

US 8,407,722 B2

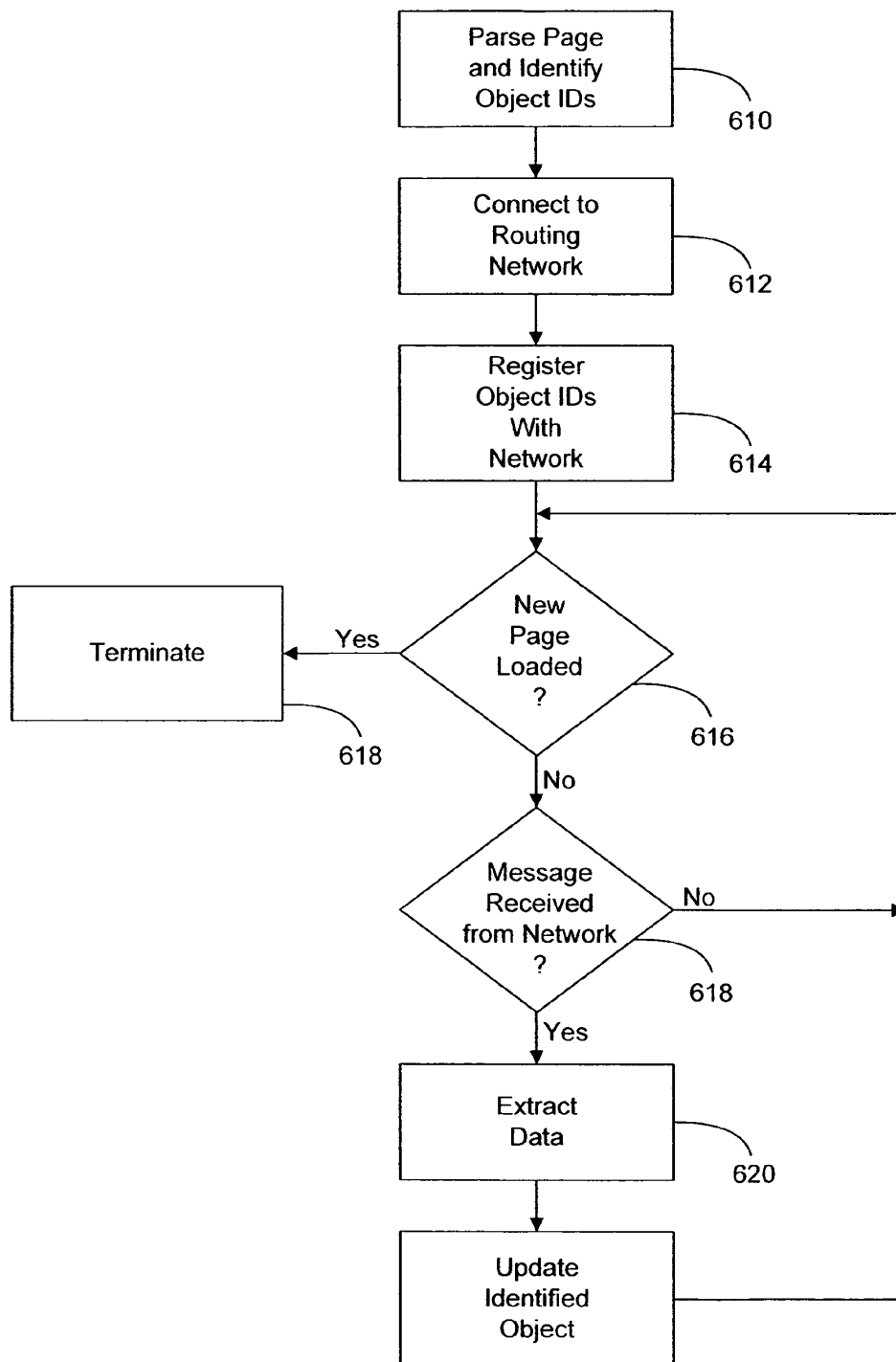


Fig. 6

U.S. Patent

Mar. 26, 2013

Sheet 7 of 12

US 8,407,722 B2

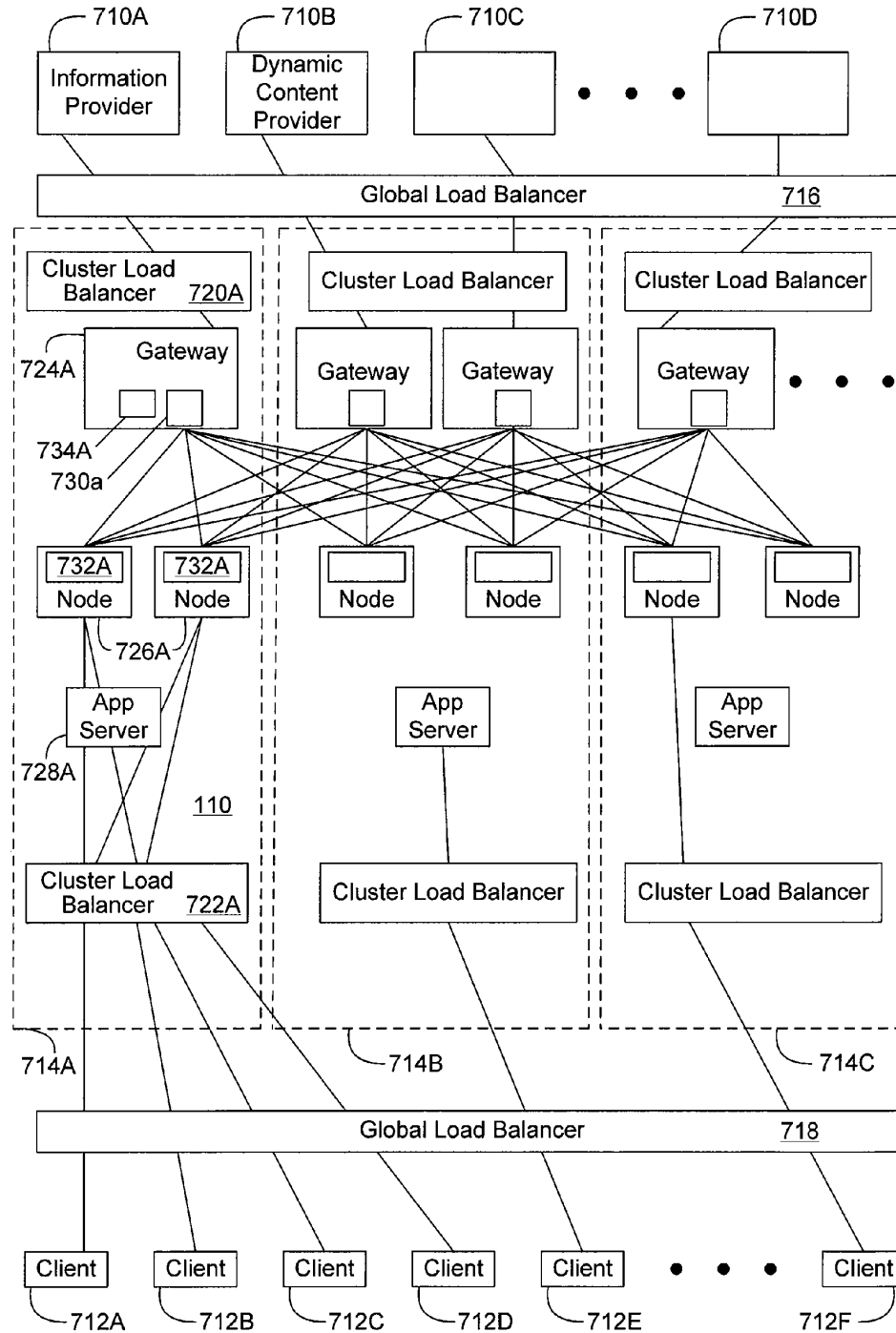


Fig. 7

U.S. Patent

Mar. 26, 2013

Sheet 8 of 12

US 8,407,722 B2

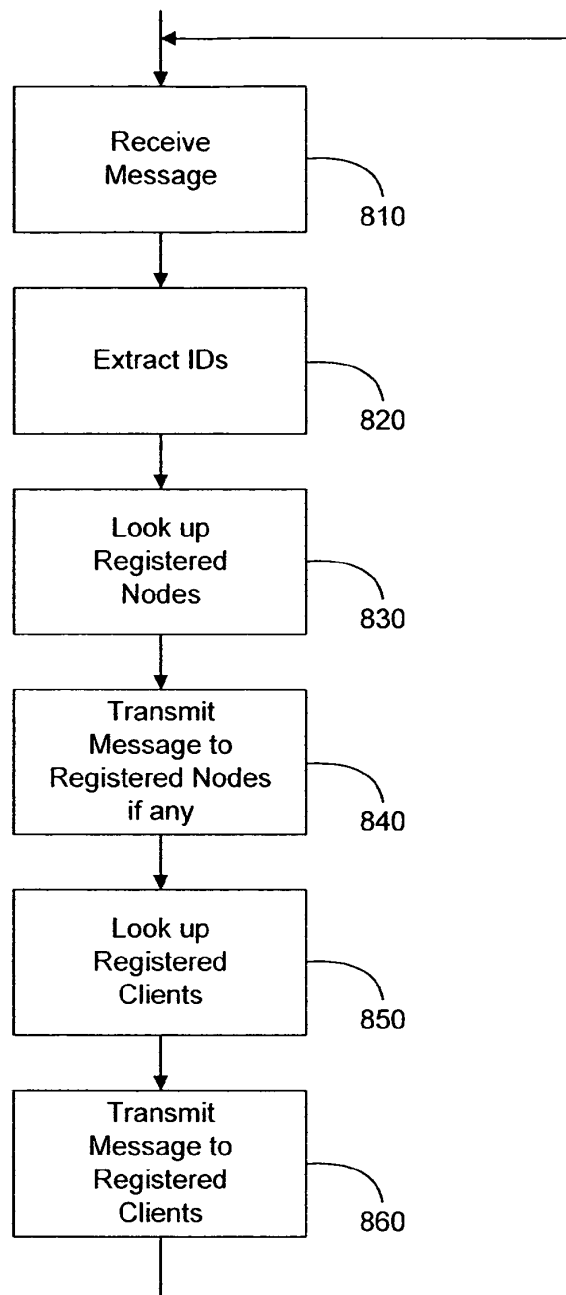


Fig. 8

U.S. Patent

Mar. 26, 2013

Sheet 9 of 12

US 8,407,722 B2

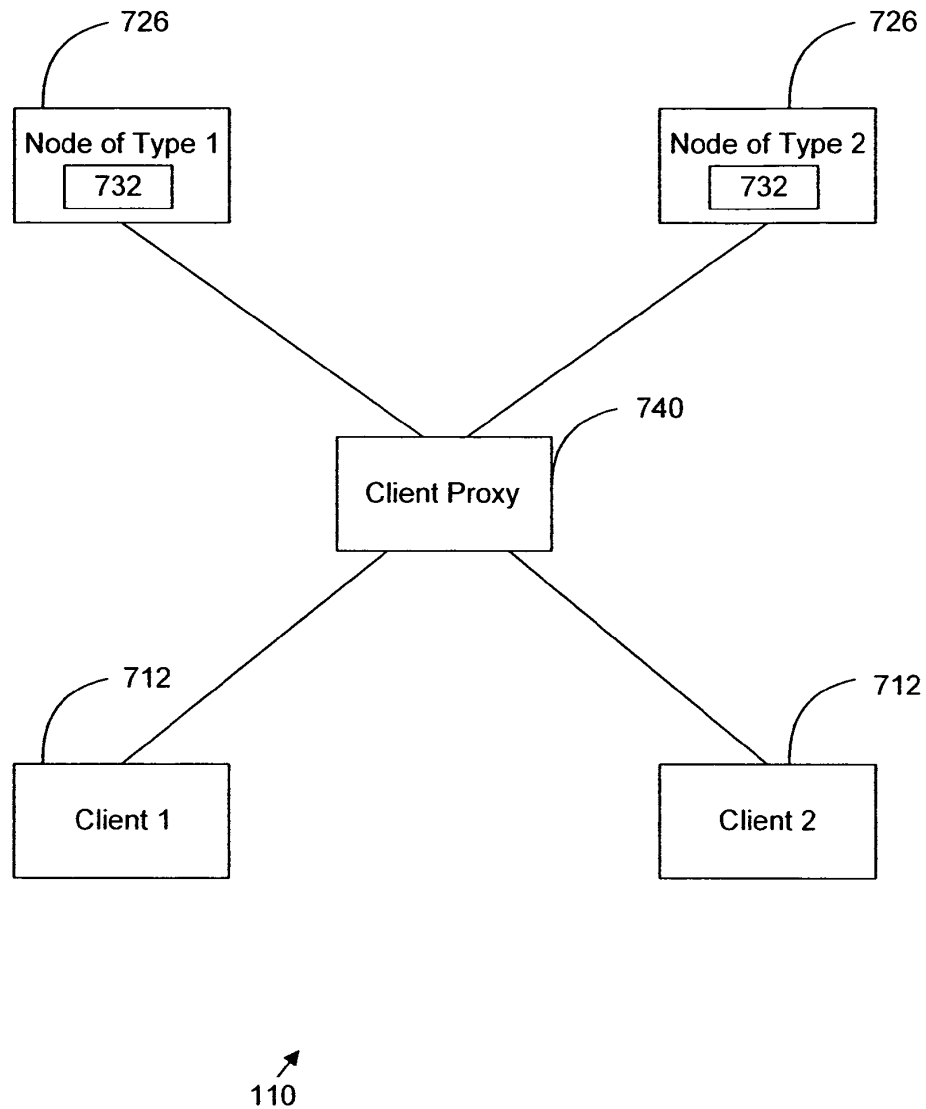


Fig. 9

U.S. Patent

Mar. 26, 2013

Sheet 10 of 12

US 8,407,722 B2

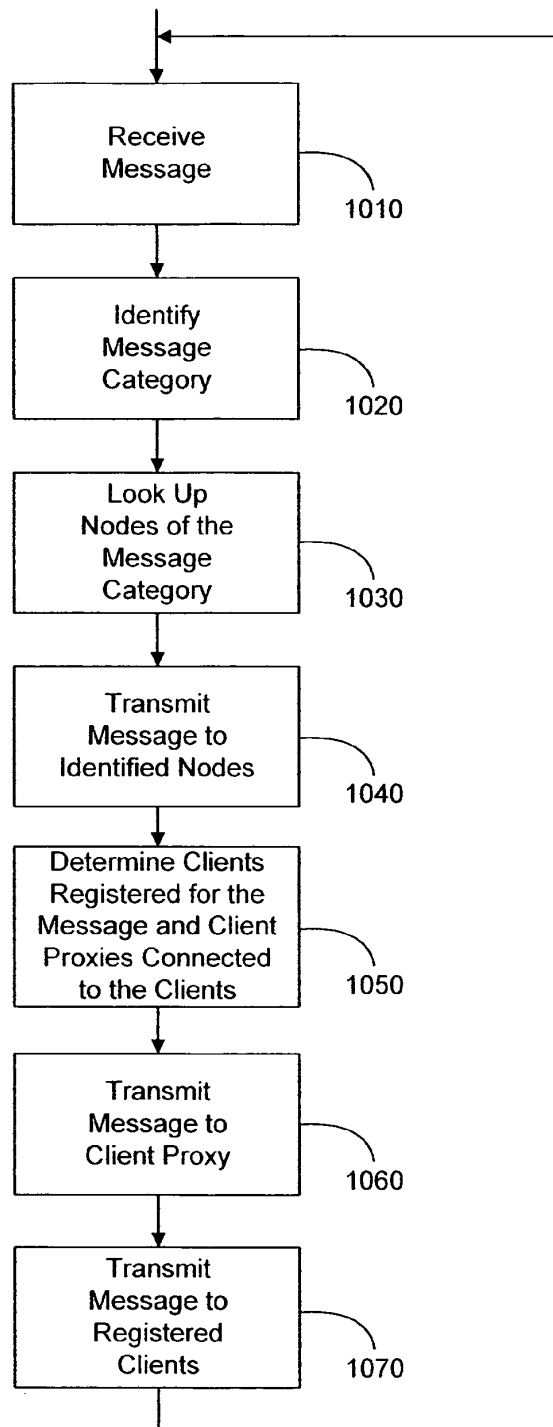


Fig. 10

U.S. Patent

Mar. 26, 2013

Sheet 11 of 12

US 8,407,722 B2

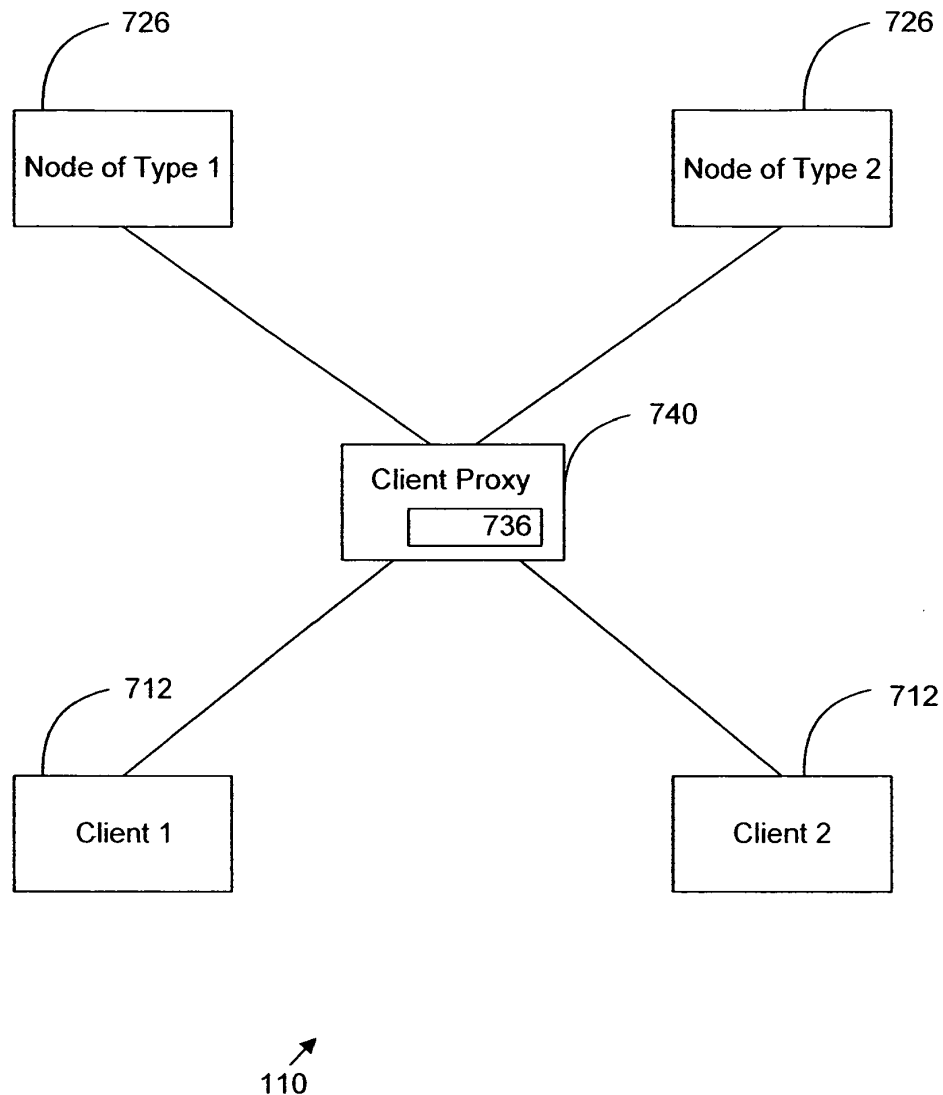


Fig. 11

U.S. Patent

Mar. 26, 2013

Sheet 12 of 12

US 8,407,722 B2

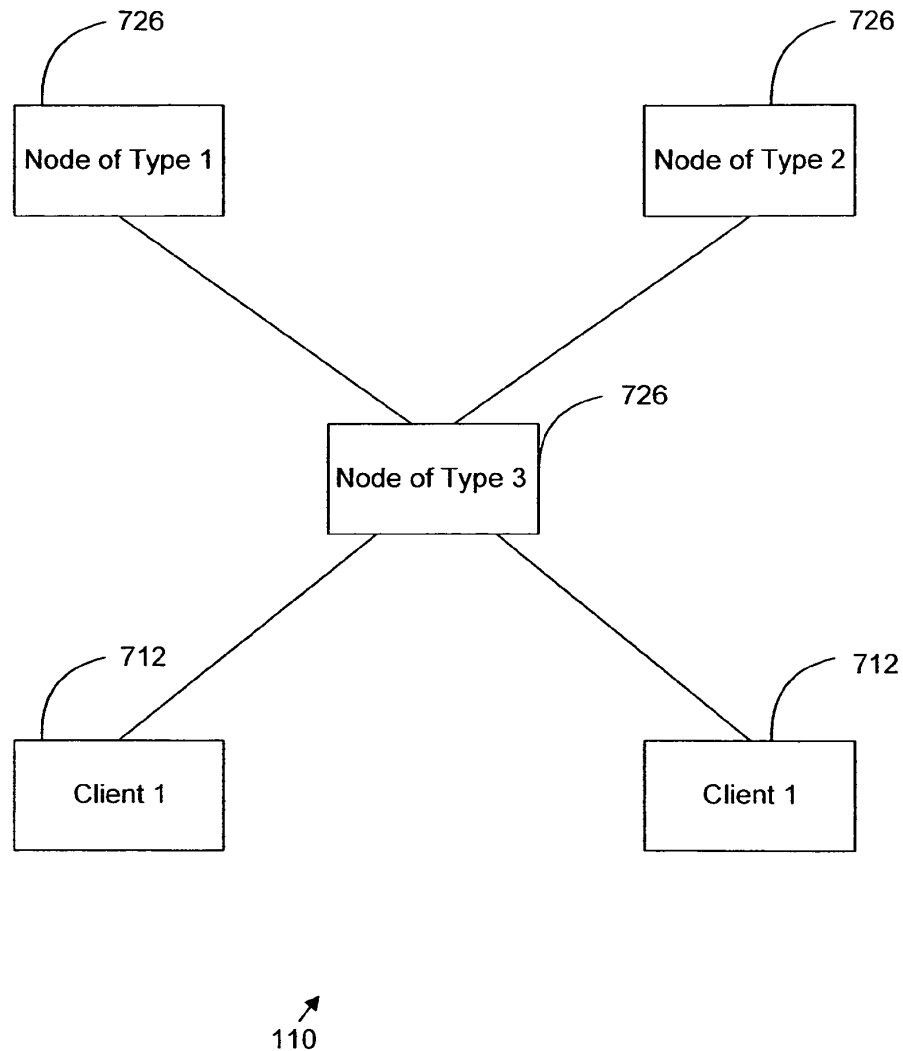


Fig. 12

US 8,407,722 B2

1

ASYNCHRONOUS MESSAGING USING A NODE SPECIALIZATION ARCHITECTURE IN THE DYNAMIC ROUTING NETWORK

CROSS-REFERENCE TO RELATED APPLICATIONS

This application is a continuation of U.S. application Ser. No. 10/105,018 filed Mar. 21, 2002 (now U.S. Pat. No. 7,051,070), which is a continuation-in-part of U.S. application Ser. No. 10/017,182 filed Dec. 14, 2001 now U.S. Pat. No. 7,043,525, which claims the benefit of U.S. Provisional Application No. 60/256,613, filed Dec. 18, 2000, U.S. Provisional Application No. 60/276,847, filed Mar. 16, 2001, U.S. Provisional Application No. 60/278,303, filed Mar. 21, 2001, U.S. Provisional Application No. 60/279,608, filed Mar. 28, 2001, and U.S. Provisional Application No. 60/280,627, filed Mar. 29, 2001, all of which are hereby incorporated by reference herein.

BACKGROUND

1. Field of the Invention

This invention pertains in general to transferring information through digital networks and in particular to transferring information for remotely updating content at client devices through the digital networks.

2. Background Art

The Internet is a digital network of computers. An individual computer on the Internet is typically identified by an internet protocol (IP) address. A computer on the Internet sends a packet of information to another computer by routing the packet to a logical port at the destination computer's IP address. The destination computer interprets the packet according to one of several possible protocols determined by the port to which the packet was sent.

The World Wide Web (the "Web") is a collection of technology and content available on the Internet that allows the content to be routed from server computers to particular destination computers. The Web includes a large number of web pages residing on many different servers. Web pages contain one or more files, or references to one or more files, specifying instructions for presenting the web page and content, such as text, images, applets, video, and/or audio.

Web pages use a variety of definitional and programming languages to control how information is presented. The most fundamental of these is the Hypertext Markup Language (HTML). HTML uses a system of "tags" to specify how content should be displayed. Recent advances in HTML introduce "style sheets" which help separate content information from display information. HTML has also been modified and extended to provide new capabilities. For example, Extensible Markup Language (XML) adds semantic content to web pages. In addition, Dynamic HTML (DHTML) adds some dynamic content to web pages.

A web page may also include one or more programs for controlling how the web page is displayed. For example, JAVA® applets and JAVASCRIPT® scripts may be used to control the display of a web page. In addition, DHTML uses scripts to control the dynamic content. Thus, a web page designer can use applets and scripts to produce animation effects or modify the display based on user interaction. For example, the designer can write a script that changes the color of a piece of text when a user clicks on a button.

Devices that display/execute web pages are often called "client devices" or simply "clients." Client devices include personal computers, web-enabled set-top boxes and televi-

2

sions, cellular telephones, personal digital assistants and other handheld devices, and special-purpose web-browsing appliances. Client devices typically employ a program called a "web browser" for interpreting the HTML or other display instructions in the web page and displaying the content accordingly. Most web browsers include special functionality, such as a Java Virtual Machine, for executing JAVA® applets and/or other applets or scripts embedded in the web pages.

A client device specifies a web page or other document on the web using a Uniform Resource Locator (URL). A URL has the form "service://server/path/file." Here "service" refers to the protocol to be used, such as the file transfer protocol (FTP) or the hypertext transport protocol (HTTP). "Server" is the IP address of the server containing the page, and "path/file" specifies the particular web page on the server.

The Web suffers from a substantial limitation with respect to dynamically updating content in a web page at a client device. The Web's only mode of operation is for a client device to first request a page from a server and then for the server to send the requested page to the client device. Once the server delivers the page to the client, it typically terminates its connection to the client, and does not retain any information about the client or the page that was sent. For this reason, servers are typically "stateless." As a result, client devices drive and control the flow of information around the Web. While client-side control is appropriate in some situations, it does not permit efficient updating of data at the client devices. For example, if a web page contains information that may change, such as the score of a baseball game or a stock quote, the server has no way to inform the client devices that are viewing the page of the change. Instead, the client devices must ask the server for the updated information. However, the client devices do not know when the information on the web page has changed, and thus do not know to ask for the update.

There are some simple web programming techniques that attempt to update content on client device-side web pages. One approach that web designers use is to rely on the client devices to periodically re-request web pages. This updating can be performed as the result of user action (such as pressing the "refresh" button) or can be automated to occur on a particular schedule (such as by using the HTML Meta Refresh tag to cause the client device to request the page every "X" seconds). Although this technique provides client devices with more up-to-date information, it is very wasteful of resources. In particular, the web server must resend the page even if nothing has changed, and, even when something has changed, it must resend the entire web page rather than just the updated information, which may be only a very small part of the page. Further, attempting to reduce unnecessary requests by decreasing the request rate results in decreasing the currency of the data. This is an unalterable trade off in a client-driven approach.

The performance of automatic refreshing can be improved somewhat by putting information that may change in a separate frame from information that is less likely to change, and only refreshing the separate frame. A few web designers even write custom JAVA applets to limit refreshing to individual components on a page, such as the score of a soccer game. A willingness to go to such effort illustrates the serious drain of resources caused by frequent refreshing. Nevertheless, even custom JAVA applets are not a meaningful attack on this problem. Custom applets require a large separate development effort for each item on each page that might need to be updated. More importantly, most custom applets still update content based upon client-driven requests, although it is possible to design an applet that accepts "pushed" messages. This

US 8,407,722 B2

3

solution is not scalable to provide updated information for large numbers of client devices and for large numbers of web pages.

Therefore, there is a need in the art for an efficient way to provide dynamic content to a web page at a client device.

DISCLOSURE OF THE INVENTION

The above need is met by a dynamic content routing network that routes messages containing data for updating properties of live objects to clients displaying web pages or other representations of data containing the live objects. The web server that initially provides the web pages to the clients does not need to track which clients are currently displaying the live objects. Instead, the information provider or a dynamic content provider (generically referred to as an "input source") that provided the live object simply sends an update message to the routing network. This routing utilizes bandwidth efficiently because the update messages are provided to the clients only when the live objects change.

The routing network is adapted to selectively send messages to the nodes in the network. In one embodiment, a hierarchy of registrations is used. Each gateway in the routing network maintains the mappings between the live objects and the nodes that have registered for the live objects. Each node in the routing network, in turn, maintains the mappings between the live objects and the clients that display them. An input source provides a message to a gateway in each cluster in the routing network. Each gateway forwards to each node only messages that reference the objects for which it has registered. Each node forwards to each client only messages that reference the objects for which it has registered. Adding node functionality to the gateway and client functionality to the node advantageously allows the routing network to decide which nodes should receive an update message. As a result, messages are sent to only nodes that have registered for the messages. Furthermore, each node receives all the messages that the clients connected to that node are interested in.

In another embodiment, all messages from an input source are assigned to one or more of N categories. Also, the nodes are assigned to one or more of M types, and mappings are created between message categories and node types. Each gateway keeps track of these mappings. When a gateway receives messages from input sources, the gateway identifies the categories of the messages and routes the messages to the nodes of the types to which the categories are mapped. To ensure that clients have access to the messages they need, clients are allowed to communicate with nodes of several types using client proxies connected between the clients and the nodes. There are at least two ways to implement the client proxy embodiment. The implementations differ primarily in where the client registration information is stored. In the first implementation, client registration information is stored at the nodes, and the client proxy merely passes messages among the clients and nodes. When a node directs a message to the client, the node passes the message to the client proxy along with a pointer to the client socket. When the client proxy receives the message, it simply pushes it to the client socket.

In the second implementation of the client proxy embodiment, the client proxy stores client registration information. In this embodiment, each node stores the object IDs registered by client proxies connected to the node. The client proxy registers with the nodes for all objects for which it needs to receive updates. The nodes receive the messages from the input sources, determine which client proxies have registered for the messages, and send the messages to the appropriate

4

client proxies. The client proxies, in turn, transmit the messages to the clients that have registered for the messages.

The features and advantages described in this summary and the following detailed description are not all-inclusive, and particularly, many additional features and advantages will be apparent to one of ordinary skill in the art in view of the drawings, specification, and claims hereof.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a high-level block diagram illustrating an environment containing a dynamic content routing network;

FIG. 2 is an interaction diagram illustrating interactions among a server, information provider, dynamic content provider, client, and routing network to update a property of a live object on a web page;

FIG. 3 is a high-level diagram graphically indicating the many-to-many mapping performed by the routing network;

FIG. 4 illustrates two different web pages containing sports scores;

FIG. 5 is a block diagram illustrating an input source and the tools available to it for generating the update messages;

FIG. 6 is a flow chart illustrating the steps performed by an embodiment of an activation module;

FIG. 7 is a block diagram illustrating a lower-level view of the routing network according to an embodiment of the present invention;

FIG. 8 is a flow chart illustrating steps performed by a gateway and a node in a cluster to perform object-based routing of a message received from an input source in an embodiment using a hierarchy of registries;

FIG. 9 is a block diagram illustrating a high-level view of the routing network in an embodiment adapted to use message categories, node types, and client proxies;

FIG. 10 is a flow chart illustrating steps performed by a gateway, a node that stores client registration information, and a pass-through client proxy to perform object-based routing of a message received from an input source;

FIG. 11 is a block diagram illustrating a high-level view of the routing network for an embodiment in which the client proxy stores the client registration information; and

FIG. 12 is a block diagram illustrating a high-level view of the routing network for an embodiment in which the nodes adopt client proxy functionality.

The figures depict an embodiment of the present invention for purposes of illustration only. One skilled in the art will readily recognize from the following description that alternative embodiments of the structures and methods illustrated herein may be employed without departing from the principles of the invention described herein.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

FIG. 1 is a high-level block diagram illustrating an environment 100 containing a dynamic content routing network 110 (hereafter referred to as the "routing network"). The environment 100 also contains a server 112 in communication with a client 114, an information provider 108, and a dynamic content provider 116. Although a typical environment 100 will have hundreds of servers 112 and information providers 108, thousands (or even millions) of clients 114, and multiple dynamic content providers 116, FIG. 1 illustrates only one of each of these entities in order to enhance the clarity of this description.

The server 112, client 114, information provider 108, dynamic content provider 116, and routing network 110 are

US 8,407,722 B2

5

preferably in communication via conventional communications links 117 such as those comprising the Internet. The communications links 117 include known wired communications media, such as dedicated or shared data, cable television or telephone lines, and/or known wireless communications media, such as communications over the cellular telephone network using protocols such as the global system for mobile communications (GSM), code division multiple access (CDMA), time division multiple access (TDMA), etc.

In one embodiment, the entities may each be in communication with one or more Internet Service Providers (ISPs) (not shown) that provide each entity with access to other computers on the Internet. In addition, the server 112, client 114, information provider 108, dynamic content provider 116, and routing network 110 are preferably each identified by at least one Internet Protocol (IP) address such as "66.35.209.224." The IP address may also have one or more domain names associated with it, such as "bangnetworks.com." Alternative embodiments of the present invention may use alternative addressing schemes and/or naming conventions instead of, or in addition to, those described herein. For example, embodiments wherein one or more of the clients are cellular telephones or other portable devices may rely on different addressing schemes.

Preferably, the information provider 108 provides web pages or other representations of data to the server 112. The web pages contain one or more "live objects," which are designated to be real-time dynamically-updateable objects. Each live object is identified by an object identifier, or object ID. Preferably, the server 112 provides the web pages 118 to multiple clients 114. The clients 114 contact the routing network 110 and register for update messages for the object IDs on the web page. The routing network 110, in turn, preferably maintains a registry indicating which clients have registered for which object IDs.

The information provider 108 and/or dynamic content provider 116 send update messages to the routing network 110. These messages can be sent any time the information provider 108 or dynamic content provider 116 wants to update a property of a live object. Each update message preferably identifies a live object and contains data for updating a property of the identified live object. The routing network 110 accesses the registry and determines which clients have registered for the identified object. Then, the routing network 110 routes the update message to the appropriate clients. Upon receipt of an update message, the clients 114 update the specified property of the live object.

The routing network 110 provides an efficient one-to-many mapping of objects to clients (and by inference of information, a many-to-many mapping of information providers 108/dynamic content providers 116 to clients) through object-based routing. Messages provided by the information provider 108 and/or dynamic content provider 116 to the routing network 110 are not routed to the clients 114 based entirely on a specified destination; more specifically, they are not routed based on the IP address of the client, as in conventional IP routing schemes. Instead, the messages are routed based on the live objects referenced by the message.

The mapping and object-based routing provided by the routing network 110 allow the information provider 108 and dynamic content provider 116 to update properties of live objects at a dynamically changing cross-section of clients in real-time, without requiring the information provider or dynamic content provider to track the clients or web pages being viewed by the clients. The clients 114, in turn, do not need to have any a priori knowledge of object IDs—they

6

"discover" which IDs they should register when they receives the web pages 118 from the server 112.

Object-based routing also allows information providers 108 to dynamically update content on web pages without requiring the clients 114 to re-request the content, and without requiring the information providers 108 or servers 112 to maintain connections with the clients. In this manner, significantly more clients can receive updated content from a given information provider 108 than would be possible utilizing conventional client-side request-driven transmission control protocol/Internet Protocol (TCP/IP) connections between the clients and the server 112.

Turning now to the individual entities illustrated in FIG. 1, the server 112 is preferably a conventional computer system configured to act as a web server and serves web pages 118 and other data representations to clients 114. The web pages 118 provided by the server 112 are associated with one or more information providers 108.

An information provider 108 is an entity providing one or more web pages 118, information contained in web pages, and/or other representations of data served by the server 112. The information provider 108 preferably has a conventional computer system coupled to the Internet. In one embodiment, the server 112 is directly controlled by the information provider 108 (e.g., the server is physically located at the information provider and/or is dedicated to serving only the information provider's web pages). In this embodiment, the server 112 and information provider 108 can be treated as the same entity. In an alternative embodiment, the server 112 serves web pages from multiple information providers.

As is known in the art, the web pages 118 and other content on the server 112 are specified by uniform resource locators (URLs) having the form "service://server/path/web page." Typically, web pages 118 are obtained via the hypertext transport protocol (HTTP) and thus an exemplary URL for retrieving the web page "b1.html" from the web server having the domain name "www.bangnetworks.com" is "http://www.bangnetworks.com/news/b1.html."

As used herein, a "web page" is a block of data available from the server 112. In the simplest case, a web page is a file written in the hypertext markup language (HTML). The web page may also contain or refer to one or more other blocks of data, such as other files, text, images, applets, video, and/or audio. In addition, the web page may contain instructions for presenting the web page and its content, such as HTML tags and style sheets. The instructions may also be in the Extensible Markup Language (XML), which is related to HTML and adds semantic content to web pages or the Dynamic HTML (DHTML), which adds some dynamic content to web pages. Additionally, the instructions may take the form of one or more programs such as JAVA® applets and JAVASCRIPT® and/or DHTML scripts.

As used herein, the phrase "web page" also refers to other representations of data served by the server 112 regardless of whether these data representations include characteristics of conventional web pages. These data representations include, for example, application programs and data intended for the web browser 120 or other application programs residing at the clients 114 or elsewhere, such as spreadsheet or textual (e.g., word processing) data, etc.

In a preferred embodiment, objects at the client, such as web pages and elements of web pages, can be designated as "live" by the information provider 108. Properties of a live object can be dynamically updated in real-time at the client 114 by the information provider 108 or another entity acting on behalf of the information provider. As used herein, an "object" is any datum or data at the client 114 that can be

US 8,407,722 B2

7

individually identified or accessed. Examples of objects include elements of web pages such as text characters and strings, images, frames, tables, audio, video, applets, scripts, HTML, XML, and other code forming the web page, variables and other information used by applets, scripts and/or code, URLs embedded in the web page, etc. Application and operating system constructs are also objects. For example, cells of spreadsheets, text in word processor documents, and title bars and messages displayed by the operating system or applications are objects. Preferably, multiple objects can be grouped together into a single, logical object. Thus, an object can be defined at any desired or useful level of granularity.

Since content on a web page is conceptualized and organized by "object," the present invention essentially abstracts web pages and web page content, and other modules and/or functionality at the client 114, away from the HTML code or other conventional representation. This abstraction allows the information provider 108 to update a property of an object without concern for the location, display format, or other specifics of how the data is being represented at the client 114.

Live objects have associated "properties" which include any modifiable data related to the object or referenced with respect to the object. The information provider 108 typically, but not necessarily, provides initial settings for the properties of live objects provided to the client 114. The properties may or may not affect the visual representation of the object in the web page or other data representation. A property may affect an internal aspect of the object and, thus, a change to the property may not have any direct effect on a web page containing the object. For example, the property may affect whether particular aspects of the object are modifiable, how the object responds to user input or other stimuli, etc. Additionally, a property may also have a direct effect on how the object is displayed at the client 114. For example, the property may affect the content, color, typeface, size, formatting, or other attribute of text, images, or other data displayed by the object. Other properties may occupy parts of the spectrum between having no effect on the visible representation of the object and having a direct effect on the visible representation of the object. For example, a web page showing scores of football games may include a list of games and the current scores of the games as of the time the server 112 serves the web page. The list of games, subset of games to be displayed, and the scores of the games can be designated as live objects (or properties of a single live object) and updated as necessary or desired.

A property can also preferably include instantiating an instance of the object or invoking functionality of the object. For example, a property of a browser window object may include functionality for instantiating another browser window. This function can be invoked as a logical change to a property of the object. The second browser window can be referenced through the original browser window (i.e., object) or designated as a new live object.

An information provider 108 or other entity preferably updates a live object at a client 114 via an update message. In general, an update message identifies the live object and, if necessary, the property of the live object, and contains data for updating the property. In one embodiment, the data may be the actual value for the property or executable code for causing the object's property to be updated. For example, the data may be a simple numerical or textual value, e.g., "4," to which the property should be set, and/or the data may be JAVASCRIPT® code or a call to a JAVASCRIPT® function at the client that effects the desired change to the property of the object.

8

The update message preferably implicitly or explicitly identifies a handler at the client 114 for use in updating the live object's property. In one embodiment, the client 114 utilizes a default handler when the message implicitly specifies the handler (e.g. when the message does not identify a specific handler). In one embodiment, if the update message specifies the actual value for the property, a default handler generates JAVASCRIPT® code for changing the property to the specified value. If the data in the update message are JAVASCRIPT® code, the default handler does not perform any processing of the code. In either case, the default handlers preferably use LiveConnect to execute the JAVASCRIPT® code in a Java Virtual Machine (JVM) 122 at the client 114 and thereby update the property of the live object.

For certain objects and/or data types, the default handlers are not appropriate. In these cases, the message preferably explicitly identifies a handler for performing the update. For example, the message may explicitly specify a function to call on the data or the message may explicitly identify the environment in which the data should be executed. For example, the data in the update message may include code for execution by a software "plug-in" such as MACROMEDIA FLASH® and the message may explicitly identify FLASH as the handler.

The information provider 108 preferably designates an object as "live" by including a unique identifier for the object, the object ID, in the web page or other data representation provided to the client 114. In one embodiment, the information provider 108 encodes the object ID in an object's corresponding HTML "ID" attribute using the following HTML expression:

ID="elementIdentifier;"

where "elementIdentifier" is the object ID and is preferably a string. The string can encode any information desired by the information provider 108 or other entity establishing the object ID and in one embodiment is a simple textual and/or numeric identifier. In one embodiment, the information provider 108 begins the object ID with a predefined token, such as "Bang\$," in order to distinguish live objects from other objects that happen to have defined ID attributes. For example, an object can have the object ID "Bang\$elementIdentifier."

In the preferred embodiment, each information provider 108 optionally encodes a unique information provider ID in its object IDs in order to prevent naming collisions between the object IDs of different information providers. In one embodiment, the information provider ID is a textual and/or numeric identifier. The information provider 108 may specify the information provider ID and the object ID as part of a hierarchical namespace. For example, in one embodiment objects are named as follows: "\$namespace1\$[namespace2\$. . . \$namespaceN\$]objectId," where "\$namespace1" is the information provider ID and the "\$" operates as the name separator and defines additional optional levels of a namespace hierarchy. One embodiment of the system 100 supports typical directory services functionality. For example, two dollar sign characters appearing together, "\$\$", refers to the top level of the namespace hierarchy.

Thus, the object ID for a live object is preferably formed from a combination of the predefined token, the information provider ID namespace, and a value assigned by the information provider 108. For example, the object ID for a live object representing the real time price of a stock having the symbol "BANG" might be: "Bang\$\$informationProviderID-Sequrities\$realtime\$Bang." In this example, "Bang\$" is the predefined token that signifies a live object, "\$information-ProviderID" is the ID identifying the information provider,

US 8,407,722 B2

9

“Sequitur\$realtime\$” defines levels of a namespace hierarchy, and “bang” identifies the specific object.

In some embodiments and situations, the object ID utilizes relative names. For example, an information provider **108** referring to its own object IDs is implicitly in its own namespace. Accordingly, the information provider **108** does not need to include the information Provider ID in the object IDs it utilizes internally. In one embodiment, the information provider ID is not explicitly encoded into the object ID. Instead, the information provider ID is encoded elsewhere in the web page in order to provide scope to the page’s object IDs.

In one embodiment, the object ID identifies a point (i.e., a node in a tree) in a Document Object Model (DOM) representation of a web page or other document at the client **114**. The DOM is a platform- and language-neutral interface that represents a document as a hierarchy of objects. The DOM also provides an interface that allows programs and scripts to dynamically access and update properties of the objects. Object properties can be inherited by descendent objects.

In this embodiment, the client **114** preferably executes an update message in the context of the specified point in the DOM representation. The update may specify a change to a property of the object at the identified point. The update also may specify a change to a parent or descendent of the object at the identified point. In each case, the update is executed relative to the specified point in the DOM representation. In one embodiment, points in the DOM representation specify how to update properties of live objects located at those points. Thus, the same update may be interpreted differently depending upon the identified live object’s location in the DOM representation.

For example, assume there is an object in the DOM representation identified as “window.document.frame[3].ObjectID.” Also assume that the object has an “innerText” property located at “window.document.frame[3].ObjectID.innerText” that specifies the text displayed by the object. An update message can change the text displayed by the object by specifying “ObjectID” and the new value for the innerText property.

An advantage of utilizing object IDs to specify objects is that the information provider **108** or other entity providing the update message can access and change properties of objects without knowing the object’s actual location in the DOM representation. Indeed, the object may be in different locations in different DOM representations and/or in multiple locations in the same DOM representation. In any of these cases, the update message will change the specified properties of all of the objects having the given object ID.

Depending upon the particular embodiment of the environment **100**, the information provider **108** and/or the dynamic content provider **116** provides update messages to the routing network **110**. The dynamic content provider **116** is preferably a conventional computer system operated by an entity that provides real-time information, such as stock prices and/or sports scores. In one embodiment, the information provider **108** receives updated properties for the live objects from the dynamic content provider **116** or another source (or generates the updated properties internally). Then, the information provider **108** sends an update message specifying the object ID and the change to the object property to the routing network **110**. In this embodiment, the dynamic content provider **116** may be absent from the environment **100**.

In another embodiment, the dynamic content provider **116** provides the object IDs for live objects to one or more information providers **108** and the information providers **108** distribute the live objects to the clients **114**. Then, the dynamic

10

content provider **116** sends messages specifying the changes to the properties of the live objects to the routing network **110**. For example, the dynamic content provider **116** distributes an object ID associated with the score of a particular baseball game to the information providers **108**. Then, the dynamic content provider **116** sends a message specifying the object ID and an update to a property of the object that controls the displayed score of the particular baseball game to the routing network **110**. These two embodiments are not mutually exclusive and, therefore, some updates may be provided to the routing network **110** by the information provider **108** while others are provided by the dynamic content provider **116**.

The client **114** is a device that retrieves web pages **118** and/or other information from the server **112**. In one embodiment, the client **114** is a conventional personal computer used by a person to access information on the Internet. In alternative embodiments, the client **114** is a different consumer electronic device having Internet connectivity, such as an Internet-enabled television, a cellular telephone, a personal digital assistant (PDA), a web browsing appliance, etc. The client **114** preferably, but not necessarily, has an associated display device.

The client **114** preferably executes a web browser **120**, such as MICROSOFT INTERNET EXPLORER®, for retrieving web pages and displaying them on the display device. In embodiments where the client receives data representations from the server **112** other than conventional web pages, the web browser **120** does not necessarily share similarities with conventional web browsers. Preferably, the web browser **120** contains a JVM **122** for executing JAVA® applets and/or scripts. The web browser **120** also preferably contains Dynamic HTML capabilities, such as support for JAVASCRIPT® (or another scripting language, such as VBScript) and the Document Object Model (DOM), and enables communications between JAVA® and the scripting languages. In one embodiment, the web browser **120** supports the LiveConnect standard for enabling communication between JAVA® applets and scripts written in the supported scripting languages. The web browser **120** can also be extended through software plug-ins such as MACROMEDIA FLASH®, REAL NETWORKS REALPLAYER®, and/or APPLE QUICKTIME®. In alternative embodiments, the functionality of the JVM **122** and/or other aspects of the web browser **120** are provided by one or more other functional units within the client **114**. The term “module” is used herein to refer to software computer program code and/or any hardware or circuitry utilized to provide the functionality attributed to the module. The web browser **120** and JVM **122** are examples of modules in the client **114**.

In some embodiments, the client **114** does not necessarily have a display device, web browser **120** and/or other components associated with a typical consumer device. The client **114**, for example, may be a dedicated purpose device having certain aspects of web connectivity such as an embedded HTTP client in a web-enabled appliance or in a controller for an automobile, audio-visual equipment, or some other device.

A web page **118** provided from the server **112** to the client **114** preferably includes instructions for enabling the live objects on the web page. The instructions cause the client **114** to automatically and transparently (i.e., without user interaction) contact the routing network **110** and download an activation module **124** for activating the live objects. In one embodiment, the instructions comprise a URL specifying the location of the activation module **124** at the routing network **110**. In an alternative embodiment, the client **114** obtains the activation module **124** from the server **112** or another source.

US 8,407,722 B2

11

The activation module 124 preferably contains JAVA® instructions for execution by the JVM 122. However, alternative embodiments of the module 124 may encode the instructions in the web page 118 and/or the activation module 124 using different languages and/or techniques. For example, the instructions and/or activation module 124 can be embedded in the web browser 120 or operating system, either as native code or as plug-ins. In these alternative embodiments, the web browser 120 does not have to download the activation module 124 from an external source.

The activation module 124 preferably registers object IDs from the web page 118 downloaded by the client 114 with the routing network 110 and updates the live objects in response to update messages received from the network. The routing network 110 records the registrations in the registry 125. The client's registrations preferably remain in effect as long as the client is displaying the associated web page 118, although other embodiments of the system 100 may use different criteria for determining when to terminate the client's registrations.

FIG. 2 is an interaction diagram illustrating interactions among the server 112, information provider 108/dynamic content provider 116 (generically referred to as an "input source 210"), client 114, and the routing network 110 to update a property of a live object. Initially, the client 114 sends 212 a web page request to the server 112. In response, the server 112 provides 214 to the client 114 the web page containing or otherwise identifying the one or more live objects. Instructions encoded in the web page preferably cause the client 114 to transparently request 216 the activation module 124 from the routing network 110. In response, the routing network 110 sends 218 the activation module 124. The client 114 executes 220 the activation module 124, which identifies the object IDs of the live objects at the client and registers 222 the object IDs with the routing network 110. The routing network 110 updates 223 its registry to identify the object IDs for which the client 114 has registered.

At some point, the input source 210 sends 224 an update message to the routing network 110 in order to change a property of a live object at the client 114. In one embodiment, the message from the input source 210 to the routing network 110 contains only a single object ID and an update to a property of the identified object. In another embodiment, the message contains multiple object IDs and the corresponding property updates. In this latter embodiment, the message may have an associated "Batch ID" that identifies the message as having multiple object IDs and updates. Preferably, the information provider 108 can include a batch ID in a web page 118 in the same manner as including an object ID. Likewise, the client 114 can preferably register for a batch ID with the routing network 110 in the same manner as an object ID. In fact, the batch ID can be the same as the object ID so that the client 114 registers for both batch and non-batch messages by registering one ID. Alternatively, separate procedures can be established for registering batch messages. The client 114 preferably processes the component messages of a batch as if each message were delivered separately.

The routing network 110, in turn, routes 226 the message to each client 114 that has registered for the specified object ID, preferably by utilizing standard Internet communications protocols, such as IP addresses, etc. The activation module 124 at the client 114 processes the message and updates 228 the property of the identified live object. If live objects having the same object ID appear in multiple locations at the client 114 (e.g., at multiple locations on a web page being displayed at the client), the activation module 124 preferably updates each of the live objects having the specified ID. As a result, the

12

routing network 110 allows live objects at the client 114 to be dynamically updated. Preferably, this routing and updating happens quickly enough to be considered "real-time" for the purposes of the input source 210.

This update process, indicated within the dashed box 230 in FIG. 2, can repeat an indefinite number of times and is fully asynchronous as to the information provider 210 and client 114. For example, the input source 210 may send regular update messages to the routing network 110 as the score of a sporting event changes or a stock price fluctuates, but may stop sending update messages once the sporting event ends or stock market closes. When the client 114 ends the display of a web page containing the live object, or otherwise no longer desires to receive update messages, the client preferably closes 232 the connection with the routing network 110. The routing network 110, in turn, updates 234 the registry 125 to remove the client's object registrations. In another embodiment, the client 114 sends messages to the routing network 110 that selectively register and/or de-register the client from one or more objects yet leaves the connection open in order to receive update messages pertaining to other objects.

FIG. 3 is a high-level diagram graphically indicating the many-to-many mapping performed by the routing network 110. Multiple input sources (labeled 210A-C) send update messages to the routing network 110. Each update message preferably specifies at least one object ID and an update to a property of the identified object. The routing network 110, in turn, selectively routes the update messages to the clients 114 that have registered for the given object ID from the given input source 210. In FIG. 3, assume for example that clients 312A and 312B have registered for a given object ID while the other clients have not registered for the object ID. Accordingly, the routing network 110 routes the update message to clients 312A and 312B, but does not route the message to clients 312C-312H.

FIG. 4 illustrates an example of the capabilities of the dynamic content routing network 110. FIG. 4 illustrates two different web pages 410, 412 containing sports scores. Although the web pages are formatted differently, each page contains the same scores for two professional football games and two professional baseball games. Web page 410 contains all four games under the heading "Local Sports Scores" while web page 412 contains the baseball games under the heading "Baseball Scores" and the football games under the heading "Football Scores."

There are various ways to internally represent the games and scores in the web pages using live objects. In one embodiment, a "game" object is defined having properties for the two teams involved in the game and the score associated with each team. The game object is placed at a selected position in the web page and the properties of the object cause the information about the game to be displayed on the page. In another embodiment, "team" and "score" objects are defined, with the team object having a property defining the name of a team and the score object having a property defining a score. In this second embodiment, the team and score objects are placed at selected locations on the page so that the proper teams and scores are aligned when the page is rendered. In yet another embodiment, an object is defined having properties for the name of one team and a score associated with that team. Then, pairs of the objects are placed in the page in the proper alignment to indicate the games and scores. In another embodiment, an object is defined having properties specifying names of two teams and a separate object is defined having properties specifying two scores. In this last embodiment, the two objects are placed in the page so that the names

US 8,407,722 B2

13

of the teams align with the associated scores. Obviously, additional variations of these representations are possible.

Assume for the example of FIG. 4 that the names of teams in a game are specified by a “names” object having properties for the two team names and the scores in the game are specified by a “scores” object having properties for two scores. In web page 410, a names object 414 having properties set to identify the “SF 49ers” and the “STL Rams” is located directly under the “Local Sports Scores” heading. A scores object 416 having a property set to identify the score of the game as “42” to “7” is directly to the right of the names object 414. In web page 412, the properties of the second names object 418 identify the same game using slightly different terminology: “SF” and “STL.” However, this names object 418 is aligned with the same scores object 416 as is utilized in web page 410.

Thus, the same scores object 416 is utilized in different positions in each web page 410, 412. In order to update the score of the San Francisco 49ers vs. St. Louis Rams football game on both web pages, the input source 210 simply sends an update message to the routing network 110 specifying the object ID for the scores object 416 and the update to the score property. The routing network 110 routes the update message to the appropriate clients 114, and the clients update the appropriate score regardless of the particular page layout.

The input source 210, i.e., the information provider 108 and/or dynamic content provider 116 can use a variety of tools to generate the update messages. FIG. 5 is a block diagram illustrating an input source 210 and the tools available to it for generating the update messages. Other tools can be utilized in addition to or instead of the ones described herein.

Preferably, the tools allow the input source 210 to access an application programming interface (API) provided by the routing network 110 for accepting messages. In one embodiment, the messages sent by the input source 210 are in the same format as utilized by the activation module 124 at the client 114. In an alternative embodiment, the messages provided to the routing network 110 are in a different format and the routing network translates the messages into the format utilized by the activation module 124.

In one embodiment, the input source 210 utilizes a data pump module 510 to access the API. The data pump module 510 reads an extensible markup language (XML) file containing one or more object IDs and the new values for the identified objects at regular intervals and automatically generates API calls that send messages representing changes to object properties to the routing network 110. In another embodiment, the data pump module 510 is event-driven and reads the XML file in response to a change in the file or some other occurrence.

In another embodiment, the input source 210 utilizes a director console module 512 to access the API. Preferably, the director console module 512 presents an administrator with a graphical interface displaying the contents of the web page 118. For example, the administrator may use the director console 512 to edit textual data, images, and/or any objects or properties of objects on the web page. After editing, the administrator uses a “send update” button or similar technique to cause the director console module 512 to send messages for the changed objects and properties to the routing network 110 via the API.

In another embodiment, the information provider 108 and dynamic content provider 116 work together as the input source 210 by using a content management system module 514 to access the API. Preferably, the content management system module 514 resides at the information provider 108

14

and receives object property updates from the dynamic content provider 116. The content management system module 514 preferably updates the properties of the live objects in the web page 118 stored at the server 112 and also sends messages for the changed properties to the routing network 110. In this manner, the web page 118 at the server 112 and the web page displayed at the client 114 are updated almost simultaneously. In one embodiment, the dynamic content provider 116 sends the update messages to the routing network 110 instead of to the information provider 108. Embodiments of the system 100 can also utilize any combination of the content management techniques described herein.

For example, the tools described above can generate a message having the following code for updating the text displayed by a score object to “2”:

```
LiveObject score=new LiveObject(“Bang$HomeScoreID”);
score.setProperty(“innerText”, “2”).
```

This code sets the innerText property of the object having object ID “Bang\$HomeScoreID” to “2.” The tools use the API to pass this message to the routing network 110.

Turning now to the actions performed at the client 114, FIG. 6 is a flow chart illustrating the steps performed by an embodiment of the activation module 124. Those of skill in the art will recognize that different embodiments may perform the steps of FIG. 6 in different orders. The activation module 124 generally performs three functions: register object IDs with the routing network 110, handle messages received by the client 114 from the network in order to update the properties of live objects, and control communications between the client and the network.

In order to register object IDs, the activation module 124 preferably parses 610 the web page 118 received from the server 112 and identifies the object IDs of the live objects. In an alternative embodiment, the activation module 124 identifies only a subset of the object IDs, such as the IDs of only live objects that are currently being displayed by the web browser 120. Alternatively, a list of object IDs may be pre-encoded in the web page in addition to the objects themselves, thereby enabling easy identification by the activation module 124. In yet another embodiment, a user of the client 114 selects the object IDs to register.

The activation module 124 preferably opens 612 a connection between the client 114 and the routing network 110. The activation module 124 can open 612 this connection before or after the activation module receives and/or parses the web page 118. In some cases, the client 114 is located behind a firewall that puts a restriction on the types of connection requests the client can make. A firewall might, for example, block all non-HTTP traffic. For this reason, the activation module 124 preferably wraps the connection request in an HTTP header in order to get the request to the routing network 110 through the firewall.

The activation module 124 uses the connection between the client 114 and routing network 110 to register 614 the object IDs by communicating to the routing network 116 a vector (e.g., a list or array) containing the identified object IDs. In order to accomplish this task through the firewall, the activation module 124 preferably puts the vector into a string, referred to as “object data,” and then preferably creates an HTTP message to communicate the object data to the routing network 110. A schematic example is as follows:

```
POST/HTTP/1.\r\n
Content-Length: <length of object data>\r\n
\r\n
<object data>
```

US 8,407,722 B2

15

where <object data> is the object ID list. When the routing network 110 receives such an HTTP request, it extracts the object data and updates the registry 125 to indicate that the client 114 has registered for the identified objects.

If the web browser 120 loads 616 a new page, or otherwise terminates display of the objects on the initial page, the activation module 124 associated with the initial web page preferably terminates 618 the client's connection with the routing network 110. Those of skill in the art will recognize that this termination 618 can occur asynchronously with the other steps illustrated in FIG. 6. Thus, the location of steps 616 and 618 represents only one possible place in the sequence of steps where the termination may occur.

If the connection is not terminated, the activation module 124 preferably waits until it receives 618 a message from the routing network 110 specifying an object ID and an update to a property of the identified object. In one embodiment, this message is received as HTTP data. Upon receipt of the message, the activation module 124 preferably extracts 620 the object ID and update from the HTTP data. Then, the activation module 124 updates 622 the property of the identified object, or causes the object to be updated, as specified by the message.

The sequence of receiving messages 618, extracting data 620, and updating objects 622 is preferably repeated until a new page is loaded 616 or the connection with the routing network 110 is otherwise terminated. Although not shown in FIG. 6, in certain circumstances, such as when a user action with respect to the web page 118 activates a new live object, the activation module 124 may register new object IDs with the routing network 110 without first downloading and parsing a new page. In one embodiment, if the newly-loaded page contains live objects, then the process of downloading the activation module 124 and updating the objects as described by FIG. 6 is repeated. In an alternative embodiment, the activation module 124 remains active at the client 114 and, therefore, the client does not re-download the activation module from the routing network 110. Instead, the already-present activation module 124 performs the live-enabling process on the new page.

FIG. 7 is a block diagram illustrating a lower-level view of the routing network 110 according to one embodiment of the present invention. FIG. 7 illustrates multiple input sources (labeled 710A-D) representative of sources providing messages to the routing network 110, such as an information provider 710A and a dynamic content provider 710B. FIG. 7 also illustrates multiple clients (labeled 712A-F) representative of the many clients in communication with the routing network 110 at any given instant.

Internally, the routing network 110 is preferably divided into one or more clusters 714. In FIG. 7, the routing network 110 has three clusters 714A, 714B, 714C, although the number of clusters can vary depending upon the processing needs of the network. An input-side global load balancer 716 preferably routes messages from the input sources 710 to the clusters 714. Similarly, a client-side global load balancer 718 preferably routes connection requests from the clients 712 to the clusters 714. The load balancers 716, 718 are designed to ensure that load is distributed among the clusters 714 according to a predetermined heuristic. For example, the load may be distributed evenly among the clusters 714 or a more powerful cluster may be distributed a majority of the load. In one embodiment, one load balancer performs the functions of the input-side 716 and client-side 718 load balancers and utilizes conventional Domain Name System-(DNS-) based load balancing.

16

Each cluster 714, of which cluster 714A is representative, preferably contains an input-side cluster load balancer 720A and a client-side cluster load balancer 722A. The cluster load balancers 720A, 722A function similarly to the corresponding global load balancers 716, 718 in that the input-side cluster load balancer 720A balances and routes incoming messages among one or more gateways 724A and the client-side cluster load balancer 722A balances and routes incoming connection requests among one or more nodes 726A and application servers 728A. The gateways 724A are connected to the nodes 726A. In one embodiment every gateway 724A is connected to every node 726A and in another embodiment certain gateways are connected to only certain nodes.

Preferably, the routing network 110 utilizes conventional single-processor computer systems executing the Linux operating system (OS). Preferably, each component of the routing network 110 is implemented by a separate, dedicated computer system in order to enable the separate optimization of the components. The input/output (I/O) functionality of the OS is preferably enhanced through the use of a non-blocking OS package such as NBIO available from the University of California, Berkeley, Calif. Based on the assumption that connections with the nodes 728 are long-lived, the OS is preferably configured to not allocate resources toward monitoring idle connections. Instead, the well-known/dev/poll patch is preferably applied to the OS in order to provide advanced socket polling capabilities.

Those skilled in the art will recognize that there are many ways to use the functionality of the routing network 110 to route update messages to clients 710. For example, in one embodiment, every message is distributed to every node 726. In another embodiment, the routing network 110 selectively sends messages to the nodes 726 in the routing network 110. Selectively sending messages to the nodes in the routing network presents at least the two difficulties. First, for a given message, a decision needs to be made as to which nodes should receive it. Second, a client must receive all messages in which it is interested. There are at least two approaches meeting these difficulties. The first approach uses a hierarchy of registries at the gateways and nodes to respectively keep track which messages to send to the nodes and clients. The second approach assigns messages to one or more categories, assigns nodes to one or more types, and maintains mappings between categories and types. This latter approach also uses client proxies to allow clients 712 to communicate with multiple nodes of different types.

FIG. 7 illustrates the embodiment using the hierarchy of registries. A node 726 registers with each gateway 724 in all of the clusters and indicates which messages it needs and the clients do the same with the nodes. Each gateway 724 preferably maintains a registry 734 containing the object IDs registered by the nodes 726 connected to the gateway 724. In one embodiment, the gateway registry 734 associates each object ID with a linked list containing one entry for each node 726 that has registered for that object ID. In another embodiment, the gateway registry 734 is a hash table containing the object ID registered by the nodes 726 connected to the gateway 724. A node 726 preferably maintains a node registry 732 containing the object IDs registered by clients 712 connected to the node. The gateways 724 in each cluster 714 receive the messages from the input sources 710 and direct the messages to the appropriate node or nodes 726. The nodes 726 preferably transmit messages received from the gateways 724 to the clients 712 that have registered in the object IDs identified by the messages.

In one embodiment, the node registry 732 associates each object ID with a linked list containing one entry for each

US 8,407,722 B2

17

client 712 that has registered for that object ID. In another embodiment, the node registry 732 is a hash table containing the object ID registered by the clients 712 connected to the nodes 726. Each entry in the linked list or hash table preferably contains a pointer to a socket representing the connection to the corresponding client 712. As is known in the art, the pointer to the socket, typically called a "file descriptor," represents an address to which the node can write in order to send the message to the corresponding client. Gateways 724 can also use file descriptors in this manner to store node addresses. Alternative embodiments of the present invention utilize other data structures in addition to, or instead of, the hash table and linked list, and/or may utilize different data within the data structures.

Preferably, the node 726 adds an entry to its registry 732 every time a client 712 registers an interest in an object and deletes the corresponding entry from the registry when the client 712 disconnects from the node or otherwise indicates that it is no longer interested in a particular object. If the node 726 determines that the client 712 registered for an object ID that was not previously registered on that node, the node preferably registers that object ID with the gateways 724 to which it is connected. Similarly, if the node 726 determines that the client 712 deregistered an object ID for which it was the last interested client, the node 726 deregisters that object ID with the gateways 724 to which it is connected. The gateways 724 update their registries 734 in response to the communication from the node 726.

In alternative embodiments of the present invention, when the client 712 disconnects from the node 726 or otherwise indicates that it is no longer interested in a particular object, the node 726 waits for a period of time or until some event occurs before deregistering the object ID with the gateway 724. For example, the node 726 could wait until it receives a message associated with that object ID to do the deregistration. Alternatively, the wait time can be fixed, random, or based on the frequency of registrations for that object ID, or registrations for the same input source 710 as the object being deregistered. This latter approach keeps the object IDs from more frequently used input sources 710 registered for a longer period of time. This waiting advantageously reduces the number of registration changes required between nodes and gateways in cases where it is likely that another client will soon register for the same object ID.

Since a gateway 724 does not control the rate at which it receives messages from input sources 710, it is possible for the gateway to receive messages faster than it can process them (i.e., send the messages to the nodes). Therefore, each gateway 724 preferably maintains a queue 730 of messages that have been received but not yet processed in order to avoid losing messages. In one embodiment, the gateway 724 drops messages if the queue 730 becomes too long. In another embodiment, the gateway 724 utilizes priorities assigned to certain messages or input sources to determine which messages to drop.

The application server 728 within each node 714 preferably serves the activation module 124 to the clients 712 in response to client requests. In addition, the application server 728 serves any other modules that may be required or desired to support the environment 100. In an alternative embodiment of the routing network, a single application server 728 fulfills all of the client requests. This application server 728 may be within a certain cluster 714 or independent of the clusters. However, this single-application-server embodiment is less desirable because it lacks redundancy.

FIG. 8 is a flow chart illustrating steps performed by a gateway 724 and a node 726 in a cluster 714 to perform

18

object-based routing of a message received from an input source 710 in the embodiment using a hierarchy of registries. Initially, the gateway 724 receives 810 the message from the input source 710. The gateway 724 extracts 820 the object ID from the message. The gateway 724 examines its registry 734 to determine the nodes 726 that have registered in the object ID. The gateway 724 transmits 840 the message to each of the registered nodes 726. Each node 726 that receives the message uses its registry 732 to determine 850 which clients 712 have registered for the message. Each node 726 then forwards 860 the message to the registered clients 712.

Adding node functionality to the gateway and client functionality to the node advantageously allows the routing network 110 to solve the difficulties identified above. For example, it allows the routing network 110 to decide which nodes should receive an updated message. As a result, messages are sent to only nodes that have registered for the message. Further, no matter which node a client connects to, that node will receive all messages that the client wants. One skilled in the art would understand that while the present invention allows registration at both a gateway level and node level, registration could be extended to any number of levels.

In the second approach for meeting the difficulties described above, all messages in the routing network 110 are assigned to one or more of N categories, and all of the nodes are assigned to one or more of M types. Mappings are created that specify which categories of messages are forwarded to which types of nodes. The mappings allow control over the amount of traffic processed by the nodes.

FIG. 9 is a block diagram illustrating a high-level view of the routing network 110 of FIG. 7 in an embodiment adapted to use the approach having message categories and node types. Although FIG. 9 illustrates only two node types—nodes of type 1 726 and nodes of type 2 726—for purposes of simplicity, this embodiment of the network 110 can have any number of node types. FIG. 9 also illustrates a client proxy 740 for reasons described below. However, an embodiment of the present invention using message categories and node types does not necessarily utilize a client proxy 740.

There are many different possible mappings between message categories and node types. In the simple case, there is one-to-one mapping between message categories and node types. For example, if the message is of category 1, it is forwarded to the nodes of type 1. In more complicated mappings, messages of one category are mapped to nodes of multiple types. For example, messages of category 1 are mapped to nodes of types 1, 2, and 3, whereas messages of category 2 are mapped to nodes of types 2, 3 and 4. In short, any possible mapping of message categories to node types is possible and the number of message categories, N, does not have to be the same as the number of node types, M.

There are multiple ways to assign the messages into categories. One way is to assign all messages from a given input source 710 into a certain category. Another way is to explicitly specify the category in the object ID for the message. Yet another way is to utilize a hashing function or lookup table to partition messages into categories based on object IDs or other values. For example, in one embodiment a hash function is applied to the object ID to generate an integer between 1 and N, and this integer is the message category.

Nodes are preferably assigned to types based on information stored in the gateways 724. In one embodiment, each gateway 724 holds a lookup table or other data structure that specifies the types to which each node is assigned. The lookup table also preferably stores the mappings between message categories and node types. When a gateway 724 receives a message from an input source, the gateway preferably deter-

US 8,407,722 B2

19

mines the category of the message using one of the techniques described above. Then, the gateway **724** determines the node type (or types) to which the message category maps, and determines which nodes are of the given type. The gateway **724** routes the message to the appropriate nodes.

In one embodiment, the gateway **724** uses a combination of multiple techniques to determine the message categories, node types, and/or mappings. For example, a lookup table can be used to encode a priori knowledge about categories, types, and/or mappings and a hash table can be used to route messages for which there is no a priori knowledge. Continuing this example, assume that certain messages are assigned to a given category based on a table lookup, while other messages are assigned to categories based on a hashing function. In this example, the gateway **724** looks up the object ID (or other information, such as an input source ID) of an arriving message in a lookup table to determine if it has a specified category. If the object ID is stored in the lookup table, the gateway **724** determines the mappings for the category and routes the messages to the nodes of the appropriate types. If the object ID is not stored in the lookup table, the gateway **724** utilizes a hash function on the object ID (or other information) to determine the message category.

Message categorization advantageously allows the routing network **110** to decide which nodes get which messages. However, message categorization does not ensure that clients **712** have access to the messages they need. Assume a client **712** connects to a node of type **1** and the client **712** wants to receive messages of category **3**. If there is a simple one-to-one mapping of message categories to node types, nodes of type **1** will never receive messages of category **3**, and neither will the client **712**. To ensure that clients have access to the messages they need, the embodiment shown in FIG. **9** uses client proxies to allow clients **712** to communicate with multiple nodes of different types. As shown in FIG. **9**, the clients **712** connect to a client proxy **740** instead of the nodes **726**. Each client proxy **740** is connected to at least one node of each type. For example, as shown in FIG. **9**, the client proxy **740** connects to a node of type **1** **726** and a node of type **2** **726**.

There are at least two ways to implement the embodiment that uses client proxies to ensure that clients connect to multiple nodes of different types. The implementations vary primarily in where the client registration information is stored. In the first variation, client registration information is stored at the nodes, and the client proxy is adapted to simply pass update messages and registration information among the clients and nodes. In the second variation, the client proxy **740** is responsible for keeping track of client registrations.

FIG. **9** illustrates the variation where the nodes store the client registration information, and the client proxy **740** is adapted to pass messages and registration information among the clients and nodes. When a client sends registration information to the client proxy **740**, the client proxy passes it to the appropriate node or nodes. In one embodiment, the client proxy **740** sends client registration information to at least one node of every type, and the nodes ignore irrelevant registration information (e.g., registrations for messages of categories not handled by the node). In another embodiment, the client proxy **740** itself analyzes the registrations, and passes only registrations relevant for a given node type to a node of that type. In a third embodiment, the client contains functionality for determining which node types handle which registration requests, and the client tells the proxy **740** to which node types to pass the registration information.

The client proxy **740** preferably stores an identifier of its connection to each client **712**, such as a pointer to the socket for the client connection, and sends this identifier to the nodes

20

along with the registration information. Each node **726**, in turn, maintains a registry **732** storing the relevant registration information it receives from the client proxy **740**. Preferably, at most one node of each type will contain registration information for a given client for messages of a category mapped to the node type. In addition, a node preferably does not store registration information for messages of categories not mapped to its node type.

The registry **732** for each node **726** preferably indicates the object IDs registered by the clients, the client proxy **740** to which the client is connected (e.g., a pointer to the socket at the node to which the client proxy is connected), and the identifier indicating the client's connection to the client proxy (e.g., the pointer to the socket at the client proxy to which the client is connected). When the node **726** receives an update message, it uses the registry to identify the clients to which it should forward the message, the client proxies to which the clients are connected, and the specific connections between the client proxies and the clients. The node routes the update message to the identified client proxies and includes the identifier (e.g., the pointer to the client socket) telling the client proxy where to send the update message. When the client proxy **740** receives the update message, it uses the identifier to send the message to the client or clients.

In one embodiment, the node registry **732** contains, for each object ID, a list with one entry for each client registered for that ID. Each entry contains a (name of client proxy, name of client) pair where each "name" is a socket identifier or some other information for identifying how to route the message to the named entity. In another embodiment, the registry **732** is adapted to more efficiently handle the case where multiple clients at a given client proxy are registered for the same object ID by placing a list of clients registered for the object ID in each entry. Thus, each entry in the list for a given object ID contains a (name of client proxy, list of names of clients on that proxy) pair. When a message having a particular object ID arrives at the node, the node walks down the corresponding entries in the registry for that ID, pushing one copy of the message to each listed client proxy. Along with the message, the node includes the list of clients on that proxy (e.g., a list of pointers to sockets for the clients) who have registered for the object ID. When the client proxy receives the message and the list, it simply pushes a copy of the message to each client.

The approach using the pass-through client proxy **740** described above is advantageous because it reduces memory loads because the client proxy stores only minimal state, and each node has a restricted list of object IDs for which it is responsible. It also expedites message routing because the client proxy does very little processing to forward a message. Plus, this approach separates different potential stress points in the network into different components, allowing the network to be tuned to provide good performance. In particular, the nodes handle a large amount of data and store a large amount of state, but do not have to hold open a large number of connections. The client proxies do not store a large amount of data or state, but hold open a large number of connections to the clients. In different embodiments, the ratios of nodes to client proxies can be varied to match the network to its requirements.

In the second variation of client proxies, the client proxy **740** stores client registration information. FIG. **11** illustrates a high-level diagram of the routing network **110** in which the client proxy **740** stores the client registration information. In this embodiment, the node registry **732** stores the object IDs registered by client proxies **740** connected to the node. Each client proxy **740** preferably maintains a client proxy registry

US 8,407,722 B2

21

736 containing the object IDs registered by clients 712 connected to the client proxy 740. The client proxy 740 registers with the nodes for all objects for which it needs to receive updates. Thus, this variation is similar to the hierarchical registry embodiment described above.

An alternative to maintaining a separate client proxy as described above is to have the nodes themselves provide the client proxy functionality. In this alternative, clients connect to a node of a first type. When a client registers for messages of a category not handled by the node, the node passes the registration to a node of the appropriate type along with an identifier of the client connection. In this case, the node connected to the client adopts the functionality of the pass-through client proxy. The node can also adopt the functionality of the client proxy that stores client registration information, thereby creating a hierarchy of registrations among the nodes themselves.

FIG. 12 is a high-level block diagram of the routing network 110 of the embodiment in which the nodes themselves adopt client proxy functionality. There are at least two ways to implement this embodiment. In one implementation there is a preferred node type, such as node of type 3, that always serves as a client proxy. This node type may be designated to handle the most common categories of messages. As a result, the client proxy functionality would be utilized very rarely. As shown in FIG. 12, client 1 712 and client 2 712 are connected to the node of type 3 726, which in turn is connected to a node of type 1 726 and a node of type 2 726.

In another implementation, instead of having one preferred node type that always serves as a client proxy, each node type can serve as a node and as a client proxy. For example, when a client 712 connects to a node 726, it passes the node all of the object IDs the client wishes to register. For object IDs of message categories handled by that node, the node stores the registrations in its registry. For object IDs of messages in other categories, the node acts as a client proxy and passes the registration requests to nodes of the appropriate types. This implementation can be made more efficient by providing functionality in the activation module for identifying the most common categories of messages sought by the client, and causing the client to connect to a node of a type that receives those categories of messages.

An alternative to connecting clients 712 to nodes 726 using a client proxy 740 is having a client 712 maintain multiple connections to the routing network 110. This can be accomplished, for example, by attaching message-categorization functionality to the activation module 124. The activation module 124 preferably determines which categories of messages it needs and to which nodes types it should connect. The activation module 124 then makes a connection to one node of each relevant type and registers the appropriate object IDs with the appropriate nodes.

An alternative to using client proxies is to allow clients to register for only categories of messages handled by one type of node. For example, if each node type handles messages from only one input source, each client 712 can be required to register for messages from only one input source. This can be done, for example, by configuring the load balancer 718 to ensure that clients registering for messages from one input source connect to a node of the type that receives messages from that input source.

FIG. 10 is a flow chart illustrating steps performed by a gateway 724, a node 726 that stores client registration information, and a pass-through client proxy 740 to perform object-based routing of a message received from an input source 710. Initially, the gateway 724 receives 1010 the message from an input source 710. The gateway 724 determines

22

1020 to which category the message belongs using one or more of the techniques described above. After the gateway 724 has determined to which category the message belongs, the gateway 724 identifies 1030 node types to which the message category is mapped and routes 1040 the message to the nodes of those types. Each node 726 that receives the message extracts 1050 the object ID from the message. Each node 726 examines its registry 732 to determine which clients have registered for the message, the client proxies to which the clients are connected, and the connections between the client proxies and the clients. Each node 726 then forwards 1060 the message to the identified client proxies 740 along with the identifiers for the client connections (e.g., the client socket or a list of client sockets). When a client proxy 740 receives the message from a node 726, it pushes 1070 the message to the identified client socket or sockets.

The above description is included to illustrate the operation of the preferred embodiments and is not meant to limit the scope of the invention. The scope of the invention is to be limited only by the following claims. From the above discussion, many variations will be apparent to one skilled in the relevant art that would yet be encompassed by the spirit and scope of the invention. For example, the invention may be implemented using a tangible computer-readable medium having stored thereon computer-executable instructions.

The invention claimed is:

1. A method comprising:

receiving, using a processing device, an update message from an input source, the update message identifying a live object and containing data for updating a property of the live object;
identifying a category of the update message based on the input source;
determining a node having a node type to which the update message is to be routed based on a mapping of categories of update messages to node types, the mapping controlling an amount of update message traffic through nodes of a routing network;
routing, using the processing device, the update message to the node having the determined node type;
causing the node, through the update message, to determine a client, different from the input source, that has registered for updates of the live object;
causing the node to route the update message from the node to the client; and
causing the client to process the update message and to update the property of the live object.

2. The method of claim 1, further comprising:

causing the node to extract an object identifier (ID) from the update message, to establish a connection with the client, and to determine if the client has registered for updates of the live object based on the object ID.

3. The method of claim 2, further comprising:

causing the node to determine at least one client proxy with which the client communicates;
causing the node to route the update message to the client proxy; and
causing the client proxy to route the update message to the client to determine the connection.

4. The method of claim 3, further comprising causing the node to maintain client registration information concerning the client connection.

5. The method of claim 3, further comprising causing the node to maintain client registration information concerning the client connection at the client proxy.

6. The method of claim 1, wherein the property of the live object has a direct effect on a visual representation of the live

US 8,407,722 B2

23

object in a data representation, has an effect on an internal aspect of the live object and has no effect on the visual representation of the live object in the data representation, or has a direct effect on one aspect of the visual representation of the live object in the data representation and has no effect on other aspect of the visual representation of the live object in the data representation.

7. A routing network comprising:

a gateway device configured to:

receive an update message from an input source, the update message identifying a live object and containing data for updating a property of the live object, identify a category of the update message based on the input source;

determine a node having a node type to which the update message is to be routed based on a mapping of a categories of update messages to node types, the mapping controlling an amount of update message traffic through nodes of a routing network; and route the update message; and

the node configured to:

receive the update message from the gateway device, wherein the node is configured to be mapped to the node type,

determine a client, different from the input source, that has registered for updates of the live object, and route the update message from the node to the client, wherein the client is adapted to process the update message and to update the property of the live object.

8. The routing network of claim 7, wherein the node is configured to extract an object identifier (ID) from the update message and to determine a connection to the client to route the data to the client.

9. The routing network of claim 8, wherein the node is configured to determine the connection by determining at least one client proxy with which the client communicates and to route the update message to the client proxy.

10. The routing network of claim 9, wherein client registration information concerning the client connection is configured to be maintained at the node.

11. The routing network of claim 9, wherein client registration information concerning the client connection is configured to be maintained at the client proxy.

12. The routing network of claim 7, wherein:

the gateway device is configured to route the update message to the node adapted to receive messages of more than one message category; and

the node is configured to determine a connection with at least one client proxy with which the client communicates and to route the data to the client proxy.

13. The routing network of claim 7, wherein the node type is configured to identify the node that receives the update message from the gateway device.

14. A method comprising:

providing, using a processing device of an input source, a data representation to a client device, different from the input source, coupled to a routing network, wherein the data representation includes at least one live object recognizable by the client device, and causing the client device to respond to the live object of the data representation by determining an object identifier (ID) of the live object and to register for updates of the live object with the routing network, such that registering the client device with the routing network provides client connection information to a node in the routing network; and sending, using the processing device of the input source, an update message to the routing network, wherein the

24

update message identifies the live object and contains update data that updates a property of the live object, wherein a gateway device at the routing network is configured to identify a category of the update message based on the input source, to determine a node type to which the identified category maps, and to route the update message to the node, having the node type, at the routing network,

wherein the node is configured to identify the client device as a registered device and to route the update message to the client device, and

wherein the client device processes the update message upon receipt to update the property of the live object at the client device.

15. The method of claim 14, wherein providing the data representation to the client device includes providing the live object that causes the client device to register with a client proxy of the routing network.

16. The method of claim 14, wherein providing the data representation to the client device includes providing the live object that causes the client device to register with the node of the routing network.

17. The method of claim 14, wherein providing the data representation to the client device includes providing an activation module that is executed by the client device and that registers the live object with the routing network.

18. The method of claim 17, wherein providing the activation module includes providing the activation module that is configured to determine a node type that handles registration and that causes the client device to register with the node corresponding to the node type.

19. The method of claim 17, wherein providing the activation module includes providing the activation module that is configured to determine a message category of the data representation and that causes the client device to register with a node having a node type corresponding to the message category.

20. An apparatus comprising:

an input source device configured to provide a data representation to a client device, different from the input source, coupled to a routing network, wherein the data representation includes at least one live object that is recognizable by the client device, and that causes the client device to determine an object identifier (ID) of the live object to register for updates of the live object with the routing network, such that registering the client device with the routing network provides client connection information to the routing network,

wherein the input source device is configured to route an update message to the routing network, wherein the update message identifies the live object and contains update data for updating a property of the live object, wherein a gateway device at the network is configured to identify a category of the update message based on the input source, to determine a node type to which the identified category maps, and to route the update message to a node of the node type at the routing network, wherein the node is configured to identify the client device as a registered device and route the update message containing the update data to the client device, and

wherein the client device is configured to process the update message upon receipt to update the property of the live object at the client device.

21. The apparatus of claim 20, wherein the live object of the data representation is configured to cause the client device to register with a client proxy of the routing network.

US 8,407,722 B2

25

22. The apparatus of claim 20, wherein the live object of the data representation is configured to cause the client device to register with the node of the routing network.

23. The apparatus of claim 20, wherein the received data representation includes an activation module that is configured to be executed by the client device and adapted to register the live object with the routing network.

24. The apparatus of claim 23, wherein the activation module is configured to determine a node type for handling registration and to cause the client device to register with the node of the node type.

25. The apparatus of claim 23, wherein the activation module determines a message category of the data representation and causes the client device to register with the node having a node type corresponding to the message category.

26. An article of manufacture including a computer-readable storage medium having instructions stored thereon, execution of which by a computing device causes the computing device to perform operations comprising:

providing, using a processing device of an input source, a data representation to a client device, different from the input source, coupled to a routing network, wherein the data representation includes at least one live object that is recognizable by the client device, and that causes the client device to respond to the live object by determining an object identifier (ID) of the live object to register for updates of the live object with the routing network, such that registering the client device with the routing network provides client connection information to the routing network; and

sending, using the processing device of the input source, an update message to the routing network, wherein the update message identifies the live object and contains update data for updating a property of the live object, wherein a gateway device at the routing network is configured to identify a category of the update message based on the input source, to determine a node type to which the identified category maps, and to route the update message to a node of the node type at the routing network,

wherein the node is configured to identify the client device as a registered device and to route the update message to the client device, and

wherein the client device is configured to process the update message upon receipt to update the property of the live object at the client device.

27. The article of manufacture of claim 26, wherein the live object of the data representation causes the client device to register with a client proxy of the routing network.

28. The article of manufacture of claim 26, wherein the live object of the data representation causes the client device to register with the node of the routing network.

29. The article of manufacture of claim 26, wherein the received data representation includes an activation module executed by the client device and adapted to register the live object with the routing network.

30. The article of manufacture of claim 29, wherein the activation module determines a node type that handles registration and causes the client device to register with the node of the node type.

26

31. The article of manufacture of claim 29, wherein the activation module determines a message category of the data representation and causes the client device to register with the node having a node type corresponding to the message category.

32. A non-transitory computer readable storage medium having instructions stored thereon, the instructions comprising:

instructions for providing a data representation to a client device coupled to a routing network, wherein the data representation includes at least one live object recognizable by the client device, and wherein the client device is configured to respond to the live object of the data representation by determining an object identifier (ID) of the live object to register for updates of the live object with the routing network, such that registering the client device with the routing network provides client connection information to the routing network; and

instructions for providing, using a processing device of an input source, different from the client device, an update message to the routing network, wherein the update message identifies the live object and contains update data for updating a property of the live object,

wherein a gateway device at the routing network is configured to identify a category of the update message based on the input source, to determine a node type to which the identified category maps, and to route the update message to a node of the node type at the routing network,

wherein the node is configured to identify the client device as a registered device and to route the update message containing the update data to the client device, and

wherein the client device is configured to process the update message upon receipt to update the property of the live object at the client device.

33. The non-transitory computer readable storage medium of claim 32, wherein the live object of the data representation causes the client device to register with a client proxy of the routing network.

34. The non-transitory computer readable storage medium of claim 32, wherein the live object of the data representation causes the client device to register with the node of the routing network.

35. The non-transitory computer readable storage medium of claim 32, wherein the data representation includes an activation module that is executed by the client device and that is adapted to register the live object with the routing network.

36. The non-transitory computer readable storage medium of claim 35, wherein the activation module determines a node type for handling registration and causes the client device to register with the node of the node type.

37. The non-transitory computer readable storage medium of claim 35, wherein the activation module determines a message category of the data representation and causes the client device to register with the node having the node type corresponding to the message category.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

PATENT NO. : 8,407,722 B2
APPLICATION NO. : 11/396251
DATED : March 26, 2013
INVENTOR(S) : Tuttle et al.

Page 1 of 1

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

On Title Page 2, Item (56), under "OTHER PUBLICATIONS", in Column 2, Line 3,
delete "Retrived" and insert -- Retrieved --, therefor.

On Title Page 3, Item (56), under "OTHER PUBLICATIONS", in Column 1, Line 12,
delete "Slavak Republic," and insert -- Slovak Republic, --, therefor.

On Title Page 3, Item (56), under "OTHER PUBLICATIONS", in Column 1, Line 48,
delete "library/default.asp?url=/library/en-us/dndotnet/html/progthrepool." and
insert -- library/default.asp?url=/library/en-us/dndotnet/html/progthrepool. --, therefor.

In the Specifications

In Column 3, Line 24, delete "live: objects" and insert -- live objects --, therefor.

In Column 12, Line 7, delete "provider 210" and insert -- provider 108 --, therefor.

In Column 14, Line 57, delete "network 116" and insert -- network 110--, therefor.

In the Claims

In Column 22, Line 47, in Claim 1, delete "of the lire" and insert -- of the live --, therefor.

In Column 24, Line 45, in Claim 20, delete "of the bye" and insert -- of the live --, therefor.

In Column 24, Line 64, in Claim 20, delete "bye object" and insert -- live object --, therefor.

In Column 26, Line 17, in Claim 32, delete "routing, network" and insert -- routing network --,
therefor.

Signed and Sealed this
Thirteenth Day of May, 2014



Michelle K. Lee
Deputy Director of the United States Patent and Trademark Office

(12) **EX PARTE REEXAMINATION CERTIFICATE** (12886th)
United States Patent
Tuttle et al. (10) **Number:** **US 8,407,722 C1**
(45) **Certificate Issued:** **Apr. 4, 2025**

(54) **ASYNCHRONOUS MESSAGING USING A
NODE SPECIALIZATION ARCHITECTURE
IN THE DYNAMIC ROUTING NETWORK**

(75) Inventors: **Timothy Tuttle**, San Francisco, CA
(US); **Karl E. Rumelhart**, Palo Alto,
CA (US)

(73) Assignee: **INTELLECTUAL VENTURES I
LLC**

H04L 67/63 (2022.01)
H04L 67/1001 (2022.01)
H04L 67/1008 (2022.01)
H04L 67/1014 (2022.01)
H04L 67/1029 (2022.01)

(52) **U.S. Cl.**
CPC **H04L 67/55** (2022.05); **H04L 45/306**
(2013.01); **H04L 67/63** (2022.05); **H04L**
67/1001 (2022.05); **H04L 67/1008** (2013.01);
H04L 67/1014 (2013.01); **H04L 67/1029**
(2013.01)

Reexamination Request:

No. 90/019,662, Sep. 16, 2024

Reexamination Certificate for:

Patent No.: **8,407,722**
Issued: **Mar. 26, 2013**
Appl. No.: **11/396,251**
Filed: **Mar. 30, 2006**

Certificate of Correction issued May 13, 2014

(58) **Field of Classification Search**

None
See application file for complete search history.

(56) **References Cited**

To view the complete listing of prior art documents cited during the proceeding for Reexamination Control Number 90/019,662, please refer to the USPTO's Patent Electronic System.

Primary Examiner — Woo H Choi

Related U.S. Application Data

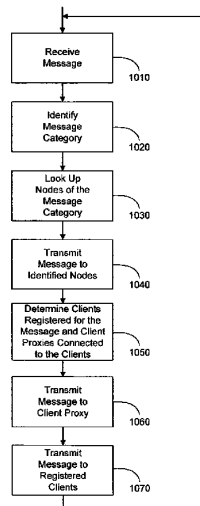
(63) Continuation of application No. 10/105,018, filed on Mar. 21, 2002, now Pat. No. 7,051,070, which is a continuation-in-part of application No. 10/017,182, filed on Dec. 14, 2001, now Pat. No. 7,043,525.

(60) Provisional application No. 60/278,303, filed on Mar. 21, 2001, provisional application No. 60/280,627, filed on Mar. 29, 2001, provisional application No. 60/279,608, filed on Mar. 28, 2001, provisional application No. 60/276,847, filed on Mar. 16, 2001, provisional application No. 60/256,613, filed on Dec. 18, 2000.

(51) **Int. Cl.**
G06F 3/00 (2006.01)
G06F 15/16 (2006.01)
H04L 45/302 (2022.01)
H04L 67/55 (2022.01)

(57) **ABSTRACT**

A network routes update messages containing updates to properties of live objects from input sources to clients having the objects. When the clients receive live objects, the clients identify the object IDs associated with the objects and register the object IDs with the routing network. The routing network is adapted to selectively send update messages to nodes in the network and the nodes forward the messages to the clients. One implementation uses a hierarchy of registries to indicate which nodes and clients receive which update messages. Another implementation assigns update messages to one or more of N categories and nodes to one or more of M types, and the gateways maintain mapping between categories and types. To ensure that clients receive all of the update messages for which they register, the clients connect to client proxies that in turn connect to at least one node of each type.



US 8,407,722 C1

1

2

**EX PARTE
REEXAMINATION CERTIFICATE**

NO AMENDMENTS HAVE BEEN MADE TO
THE PATENT 5

AS A RESULT OF REEXAMINATION, IT HAS BEEN
DETERMINED THAT:

The patentability of claims 1, 6-7, 14-17, 20-23, 26-29¹⁰
and 32-35 is confirmed.

Claims 2-5, 8-13, 18-19, 24-25, 30-31 and 36-37 were not
reexamined.

* * * * *

EXHIBIT A-9

**Amendment Under 37 C.F.R. § 1.116
Expedited Procedure – Art Unit 2194**

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of:

TUTTLE *et al.*

Appl. No.: 11/396,251

Filed: March 30, 2006

For: **Asynchronous Messaging Using a
Node Specialization Architecture in
the Dynamic Routing Network**

Confirmation No.: 1129

Art Unit: 2194

Examiner: DAO, TUAN C.

Atty. Docket: 2222.775000E

Reply Under 37 C.F.R. § 1.116

Mail Stop AF

Commissioner for Patents
PO Box 1450
Alexandria, VA 22313-1450

Sir:

In reply to the Final Office Action dated **December 30, 2011**, (“Office Action”),
Applicants submit the following Amendment and Remarks.

Status of the Claims is reflected in the listing of claims which begins on page **2**
of this paper.

Remarks and Arguments begin on page **16** of this paper.

It is not believed that extensions of time or fees for net addition of claims are
required beyond those that may otherwise be provided for in documents accompanying
this paper. However, if additional extensions of time are necessary to prevent
abandonment of this application, then such extensions of time are hereby petitioned
under 37 C.F.R. § 1.136(a), and any fees required therefor (including fees for net
addition of claims) are hereby authorized to be charged to our Deposit Account No.
19-0036.

PETITIONERS - AMERICAN/SOUTHWEST, Exhibit 1002
Page 950 of 1173

AA/SWA Ex. 1014, p.230 of 288
American Airlines, et. al. v. Intellectual Ventures, et. al.
IPR2025-01511

Reply to Office Action of December 30, 2011

TUTTLE *et al.*
Appl. No. 11/396,251

Status of the Claims

This listing of claims will replace all prior versions, and listings, of claims in the application.

1-25. (Cancelled)

26. (Previously Presented) A method comprising:

receiving, using a processing device, an update message from an input source, the update message identifying a live object and containing data for updating a property of the live object;

identifying a category of the update message based on the input source;

determining a node having a node type to which the update message is to be routed based on a mapping of categories of update messages to node types, the mapping controlling an amount of update message traffic through nodes of a routing network;

routing, using the processing device, the update message to the node having the determined node type;

causing the node, through the update message, to determine a client, different from the input source, that has registered for updates of the live object;

causing the node to route the update message from the node to the client; and

causing the client to process the update message and to update the property of the live object.

Atty. Dkt. No. 2222.775000E

- 3-

Reply to Office Action of December 30, 2011

TUTTLE *et al.*
Appl. No. 11/396,251

27. (Cancelled)

28. (Previously Presented) The method of claim 26, further comprising:

causing the node to extract an object ID from the update message, to establish a connection with the client, and to determine if the client has registered for updates of the live object based on the object ID.

29. (Previously Presented) The method of claim 28, further comprising:

causing the node to determine at least one client proxy with which the client communicates;

causing the node to route the update message to the client proxy; and

causing the client proxy to route the update message to the client to determine the connection.

30. (Previously Presented) The method of claim 29, further comprising causing the node to maintain client registration information concerning the client connection.

31. (Previously Presented) The method of claim 29, further comprising causing the node to maintain client registration information concerning the client connection at the client proxy.

32. (Cancelled)

Atty. Dkt. No. 2222.775000E

PETITIONERS - AMERICAN/SOUTHWEST, Exhibit 1002
Page 952 of 1173

AA/SWA Ex. 1014, p.232 of 288
American Airlines, et. al. v. Intellectual Ventures, et. al.
IPR2025-01511

- 4-

Reply to Office Action of December 30, 2011

TUTTLE *et al.*
Appl. No. 11/396,251

33. (Previously Presented) A routing network comprising:

a gateway device configured to:

receive an update message from an input source, the update message identifying a live object and containing data for updating a property of the live object,

identify a category of the update message based on the input source;

determine a node having a node type to which the update message is to be routed based on a mapping of a categories of update messages to node types, the mapping controlling an amount of update message traffic through nodes of a routing network; and

route the update message; and

the node configured to:

receive the update message from the gateway device, wherein the node is configured to be mapped to the node type,

determine a client, different from the input source, that has registered for updates of the live object, and

route the update message from the node to the client, wherein the client is adapted to process the update message and to update the property of the live object.

34. (Previously Presented) The routing network of claim 33, wherein the node is configured to extract an object ID from the update message and to determine a connection to the client to route the data to the client.

Atty. Dkt. No. 2222.775000E

PETITIONERS - AMERICAN/SOUTHWEST, Exhibit 1002
Page 953 of 1173

AA/SWA Ex. 1014, p.233 of 288
American Airlines, et. al. v. Intellectual Ventures, et. al.
IPR2025-01511

- 5-

Reply to Office Action of December 30, 2011

TUTTLE *et al.*
Appl. No. 11/396,251

35. (Previously Presented) The routing network of claim 34, wherein the node is configured to determine the connection by determining at least one client proxy with which the client communicates and to route the update message to the client proxy.

36. (Previously Presented) The routing network of claim 35, wherein client registration information concerning the client connection is configured to be maintained at the node.

37. (Previously Presented) The routing network of claim 35, wherein client registration information concerning the client connection is configured to be maintained at the client proxy.

38. (Previously Presented) The routing network of claim 33, wherein:
the gateway device is configured to route the update message to the node adapted to receive messages of more than one message category, and
the node is configured to determine a connection with at least one client proxy with which the client communicates and to route the data to the client proxy.

39. (Previously Presented) A method comprising:
providing, using a processing device of an input source, a data representation to a client device, different from the input source, coupled to a routing network, wherein the data representation includes at least one live object recognizable by the client device, and causing the client device to respond to the live object of the data representation by

Atty. Dkt. No. 2222.775000E

- 6-

Reply to Office Action of December 30, 2011

TUTTLE *et al.*
Appl. No. 11/396,251

determining an object ID of the live object and to register for updates of the live object with the routing network, such that registering the client device with the routing network provides client connection information to a node in the routing network; and

sending, using the processing device of the input source, an update message to the routing network, wherein the update message identifies the live object and contains update data that updates a property of the live object,

wherein a gateway device at the routing network is configured to identify a category of the update message based on the input source, to determine a node type to which the identified category maps, and to route the update message to the node, having the node type, at the routing network,

wherein the node is configured to identify the client device as a registered device and to route the update message to the client device, and

wherein the client device processes the update message upon receipt to update the property of the live object at the client device.

40. (Previously Presented) The method of claim 39, wherein providing the data representation to the client device includes providing the live object that causes the client device to register with a client proxy of the routing network.

41. (Previously Presented) The method of claim 39, wherein providing the data representation to the client device includes providing the live object that causes the client device to register with the node of the routing network.

Atty. Dkt. No. 2222,775000E

PETITIONERS - AMERICAN/SOUTHWEST, Exhibit 1002
Page 955 of 1173

AA/SWA Ex. 1014, p.235 of 288
American Airlines, et. al. v. Intellectual Ventures, et. al.
IPR2025-01511

- 7 -

Reply to Office Action of December 30, 2011

TUTTLE *et al.*
Appl. No. 11/396,251

42. (Previously Presented) The method of claim 39, wherein providing the data representation to the client device includes providing an activation module that is executed by the client device and that registers the live object with the routing network.

43. (Previously Presented) The method of claim 42, wherein providing the activation module includes providing the activation module that is configured to determine a node type that handles registration and that causes the client device to register with the node corresponding to the node type.

44. (Previously Presented) The method of claim 42, wherein providing the activation module includes providing the activation module that is configured to determine a message category of the data representation and that causes the client device to register with a node having a node type corresponding to the message category.

45. (Previously Presented) An apparatus comprising:

an input source device configured to provide a data representation to a client device, different from the input source, coupled to a routing network, wherein the data representation includes at least one live object that is recognizable by the client device, and that causes the client device to determine an object ID of the live object to register for updates of the live object with the routing network, such that registering the client device with the routing network provides client connection information to the routing network,

Atty. Dkt. No. 2222.775000E

PETITIONERS - AMERICAN/SOUTHWEST, Exhibit 1002
Page 956 of 1173

AA/SWA Ex. 1014, p.236 of 288
American Airlines, et. al. v. Intellectual Ventures, et. al.
IPR2025-01511

wherein the input source device is configured to route an update message to the routing network, wherein the update message identifies the live object and contains update data for updating a property of the live object,

wherein a gateway device at the network is configured to identify a category of the update message based on the input source, to determine a node type to which the identified category maps, and to route the update message to a node of the node type at the routing network,

wherein the node is configured to identify the client device as a registered device and route the update message containing the update data to the client device, and

wherein the client device is configured to process the update message upon receipt to update the property of the live object at the client device.

46. (Previously Presented) The apparatus of claim 45, wherein the live object of the data representation is configured to cause the client device to register with a client proxy of the routing network.

47. (Previously Presented) The apparatus of claim 45, wherein the live object of the data representation is configured to cause the client device to register with the node of the routing network.

48. (Previously Presented) The apparatus of claim 45, wherein the received data representation includes an activation module that is configured to be executed by the client device and adapted to register the live object with the routing network.

Atty. Dkt. No. 2222.775000E

- 9-

Reply to Office Action of December 30, 2011

TUTTLE *et al.*
Appl. No. 11/396,251

49. (Previously Presented) The apparatus of claim 48, wherein the activation module is configured to determine a node type for handling registration and to cause the client device to register with the node of the node type.

50. (Previously Presented) The apparatus of claim 48, wherein the activation module determines a message category of the data representation and causes the client device to register with the node having a node type corresponding to the message category.

51. (Previously Presented) An article of manufacture including a computer-readable storage medium having instructions stored thereon, execution of which by a computing device causes the computing device to perform operations comprising:

providing, using a processing device of an input source, a data representation to a client device, different from the input source, coupled to a routing network, wherein the data representation includes at least one live object that is recognizable by the client device, and that causes the client device to respond to the live object by determining an object ID of the live object to register for updates of the live object with the routing network, such that registering the client device with the routing network provides client connection information to the routing network; and

sending, using the processing device of the input source, an update message to the routing network, wherein the update message identifies the live object and contains update data for updating a property of the live object,

Atty. Dkt. No. 2222.775000E

- 10-

Reply to Office Action of December 30, 2011

TUTTLE *et al.*
Appl. No. 11/396,251

wherein a gateway device at the routing network is configured to identify a category of the update message based on the input source, to determine a node type to which the identified category maps, and to route the update message to a node of the node type at the routing network,

wherein the node is configured to identify the client device as a registered device and to route the update message to the client device, and

wherein the client device is configured to process the update message upon receipt to update the property of the live object at the client device.

52. (Previously Presented) The article of manufacture of claim 51, wherein the live object of the data representation causes the client device to register with a client proxy of the routing network.

53. (Previously Presented) The article of manufacture of claim 51, wherein the live object of the data representation causes the client device to register with the node of the routing network.

54. (Previously Presented) The article of manufacture of claim 51, wherein the received data representation includes an activation module executed by the client device and adapted to register the live object with the routing network.

Atty. Dkt. No. 2222.775000E

PETITIONERS - AMERICAN/SOUTHWEST, Exhibit 1002
Page 959 of 1173

AA/SWA Ex. 1014, p.239 of 288
American Airlines, et. al. v. Intellectual Ventures, et. al.
IPR2025-01511

- 11 -

Reply to Office Action of December 30, 2011

TUTTLE *et al.*
Appl. No. 11/396,251

55. (Previously Presented) The article of manufacture of claim 54, wherein the activation module determines a node type that handles registration and causes the client device to register with the node of the node type.

56. (Previously Presented) The article of manufacture of claim 54, wherein the activation module determines a message category of the data representation and causes the client device to register with the node having a node type corresponding to the message category.

57. (Previously Presented) A non-transitory computer readable storage medium having instructions stored thereon, the instructions comprising:

instructions for providing a data representation to a client device coupled to a routing network, wherein the data representation includes at least one live object recognizable by the client device, and wherein the client device is configured to respond to the live object of the data representation by determining an object ID of the live object to register for updates of the live object with the routing network, such that registering the client device with the routing network provides client connection information to the routing network; and

instructions for providing, using a processing device of an input source, different from the client device, an update message to the routing network, wherein the update message identifies the live object and contains update data for updating a property of the live object,

Atty. Dkt. No. 2222.775000E

PETITIONERS - AMERICAN/SOUTHWEST, Exhibit 1002
Page 960 of 1173

AA/SWA Ex. 1014, p.240 of 288
American Airlines, et. al. v. Intellectual Ventures, et. al.
IPR2025-01511

- 12-

Reply to Office Action of December 30, 2011

TUTTLE *et al.*
Appl. No. 11/396,251

wherein a gateway device at the routing network is configured to identify a category of the update message based on the input source, to determine a node type to which the identified category maps, and to route the update message to a node of the node type at the routing network,

wherein the node is configured to identify the client device as a registered device and to route the update message containing the update data to the client device, and

wherein the client device is configured to process the update message upon receipt to update the property of the live object at the client device.

58. (Previously Presented) The non-transitory computer readable storage medium of claim 57, wherein the live object of the data representation causes the client device to register with a client proxy of the routing network.

59. (Previously Presented) The non-transitory computer readable storage medium of claim 57, wherein the live object of the data representation causes the client device to register with the node of the routing network.

60. (Previously Presented) The non-transitory computer readable storage medium of claim 57, wherein the data representation includes an activation module that is executed by the client device and that is adapted to register the live object with the routing network.

Atty. Dkt. No. 2222.775000E

PETITIONERS - AMERICAN/SOUTHWEST, Exhibit 1002
Page 961 of 1173

AA/SWA Ex. 1014, p.241 of 288
American Airlines, et. al. v. Intellectual Ventures, et. al.
IPR2025-01511

- 13-

Reply to Office Action of December 30, 2011

TUTTLE *et al.*
Appl. No. 11/396,251

61. (Previously Presented) The non-transitory computer readable storage medium of claim 60, wherein the activation module determines a node type for handling registration and causes the client device to register with the node of the node type.

62. (Previously Presented) The non-transitory computer readable storage medium of claim 60, wherein the activation module determines a message category of the data representation and causes the client device to register with the node having the node type corresponding to the message category.

63. (Previously Presented) A method comprising:

providing a live object to a client device;

sending, using a processing device of an input source, different from the client device, an update message to a routing network identifying the live object and containing update data to update the live object at the client device,

wherein in response to determining that the client device is registered to receive update data for the live object, causing a category of the update message to be identified based on the input source, a node type associated with the category be determined, and the update message be routed from the gateway device to a node having the node type at the routing network, and causing the node to route the update message to the client device.

Atty. Dkt. No. 2222.775000E

PETITIONERS - AMERICAN/SOUTHWEST, Exhibit 1002
Page 962 of 1173

AA/SWA Ex. 1014, p.242 of 288
American Airlines, et. al. v. Intellectual Ventures, et. al.
IPR2025-01511

- 14-

Reply to Office Action of December 30, 2011

TUTTLE *et al.*
Appl. No. 11/396,251

64. (Previously Presented) An apparatus comprising:

an input source arranged to provide a live object to a client device, different from the input source; and

the input source arranged to provide an update message to a routing network identifying the live object and containing update data for updating the live object,

wherein the routing network is configured to identify a category of the update message based on the input source, determine a node type associated with the category, and to route the update message from a gateway device at the routing network to a node having the node type at the routing network, and wherein the node is configured to route the update message to the client device if the client device is registered to receive the update data for the live object.

65. (Cancelled)

66. (Previously Presented) A non-transitory computer readable storage medium having instructions stored thereon, the instructions comprising:

instructions for providing a live object to a client device; and

instructions for providing an update message from an input source, different from the client device, to a routing network identifying the live object and containing update data for updating the live object at the client device,

wherein a category of the update message is identified based on the input source, a node type associated with the category is determined, and the update message is routed from a gateway device at the routing network to a node having the node type at the

Atty. Dkt. No. 2222,775000E

- 15-

Reply to Office Action of December 30, 2011

TUTTLE *et al.*
Appl. No. 11/396,251

routing network, wherein the update message is routed from the node to the client device in response to a determination that the client device is registered to receive the update data for the live object.

67. (Previously Presented) The routing network of claim 33, wherein the node type is configured to identify the node that receives the update message from the gateway device.

68. (Previously Presented) The method of claim 26, wherein the property of the live object has a direct effect on a visual representation of the live object in a data representation, has an effect on an internal aspect of the live object and has no effect on the visual representation of the live object in the data representation, or has a direct effect on one aspect of the visual representation of the live object in the data representation and has no effect on other aspect of the visual representation of the live object in the data representation.

Atty. Dkt. No. 2222.775000E

PETITIONERS - AMERICAN/SOUTHWEST, Exhibit 1002
Page 964 of 1173

AA/SWA Ex. 1014, p.244 of 288
American Airlines, et. al. v. Intellectual Ventures, et. al.
IPR2025-01511

- 16-

Reply to Office Action of December 30, 2011

TUTTLE *et al.*
Appl. No. 11/396,251

Remarks

Reconsideration of this Application is respectfully requested.

Claims 26, 28-31, 33-64, and 66-68 are pending in the application, with claims 26, 33, 39, 45, 51, 57, 63, 64, and 66 being the independent claims. No claim amendments are sought.

Based on the following remarks, Applicants respectfully request that the Examiner reconsider all outstanding rejections and that they be withdrawn.

Rejections under 35 U.S.C. § 103

Claims 26, 28, 33, 34, 39, 42-45, 48-51, 54-57, 60-62, 67, and 68

Claims 26, 28, 33, 34, 39, 42-45, 48-51, 54-57, 60-62, 67, and 68 were rejected under 35 U.S.C. § 103(a) as allegedly being unpatentable over U.S. Patent No 6,091,724 to Chandra et al. ("Chandra") in view of U.S. Patent No. 6,510,323 to Stocker et al. ("Stocker") and further in view of U.S. Patent No. 6,324,578 to Trenbeath et al. ("Trenbeath"). Applicants respectfully traverse this rejection.

Independent claims 26, 33, 39, 45, 51, and 57 recite features that distinguish over the applied references. For example, claim 26 recites, *inter alia*, (emphasis added):

identifying a category of the update message based on the input source.

Independent claim 33 recites similar distinguishing features, using its respective language. Further, independent claim 39 recites, *inter alia*, (emphasis added):

wherein a gateway device at the routing network is configured to *identify a category of the update message based on the input source,* to determine a node type to which the identified category maps, and to route the update

Atty. Dkt. No. 2222.775000E

PETITIONERS - AMERICAN/SOUTHWEST, Exhibit 1002
Page 965 of 1173

AA/SWA Ex. 1014, p.245 of 288
American Airlines, et. al. v. Intellectual Ventures, et. al.
IPR2025-01511

- 17 -

Reply to Office Action of December 30, 2011

TUTTLE *et al.*
Appl. No. 11/396,251

message to the node, having the node type, at the routing network.

Independent claims 45, 51, and 57 recite similar distinguishing features, using their respective language.

These features are neither disclosed nor suggested by the applied references.

The Examiner, for example, on page 3 of the Office Action relies on columns 1:25-35 and 5:45-65 of Chandra (in view of paragraphs 103 and 105 of the specification of instant application) to allegedly show the above-noted distinguishing features of independent claims 26, 33, 39, 45, 51, and 57. Applicants respectfully disagree.

Chandra discloses a method for “routing messages within a network using the data content of the message.” (Chandra, Title.) When discussing the background art, Chandra discloses that (emphasis added):

One example of a system that includes such a network environment is a publish/subscribe system. In publish/subscribe systems, publishers post messages and ***subscribers independently specify categories of events in which they are interested.*** The system takes the posted messages and includes in each message header the destination information of those subscribers indicating interest in the particular message. The system then uses the destination information in the message to forward the message through the network to the appropriate subscribers.

(Chandra, 1:26-35.)

This section of Chandra, relied on by the Examiner, merely discloses that subscribers can specify categories of event in which they are interested. This is not the same as “identify a category of the update message based on the input source.” as recited in claims 26, 33, 39, 45, 51, and 57, using their respective language.

Atty. Dkt. No. 2222.775000E

PETITIONERS - AMERICAN/SOUTHWEST, Exhibit 1002
Page 966 of 1173

AA/SWA Ex. 1014, p.246 of 288
American Airlines, et. al. v. Intellectual Ventures, et. al.
IPR2025-01511

- 18-

Reply to Office Action of December 30, 2011

TUTTLE *et al.*
Appl. No. 11/396,251

Chandra further discusses the content based subscription, where “one or more of the clients can specify a subscription. Each subscription provides a set of conditions indicating the types of messages those clients are interested in receiving.” (Chandra, 5:45-48.) Chandra discloses that (emphasis added):

Consider an example in which messages are related to the stock market. In such an example, a subscription may have the following three fields (or attributes): stock issue, stock price and stock value. The client specifies values for these fields, e.g., IBM, 100, 200, and then, these fields are used to determine if the client is interested in a particular message. *If the data content of the message matches the specified values of the subscription, then the message is delivered to the one or more subscribers (or clients) of that subscription.* It will be apparent to those skilled in the art that the use of three fields in the subscription is only one example. Subscriptions may include any number of desired fields and tests may include relationships other than equality. Further, the stock market scenario is only one example. Messages and subscriptions can be used for any type of data or events.

(Chandra, 5:53-67.)

This section of Chandra, relied on by the Examiner, discloses that if the *data content of the message* matches the specified values of the subscription, then the message is delivered to the one or more subscribers (or clients) of that subscription. This is not the same as “identify a category of the update message based on the input source,” as recited in claims 26, 33, 39, 45, 51, and 57, using their respective language.

The Examiner, for example, on page 3 of the Office Action, relies on the above sections of Chandra to allegedly show “identifying each category based on the category/topic of the message content.” By this statement, the Examiner appears to agree that Chandra teaches identifying categories based on category/topic of the message content. In contrast, claims 26, 33, 39, 45, 51, and 57 recite, *inter alia*,

Atty. Dkt. No. 2222.775000E

- 19 -

Reply to Office Action of December 30, 2011

TUTTLE *et al.*
Appl. No. 11/396,251

“identify a category of the update message based on the input source,” using their respective language.

The Office Action, for example, on pages 4 and 5, states, which Applicants do not acquiesce to, that Stocker teaches “GIMR 300 determining a message control node 210 to send a general information message 220 to MSS 110,” “if the cell serving mobile station in part of the served area 195, the step 430 is entered,” “determining if the served area ... associated with the MS 100,” “routing the update message to the node having determined node type,” and “sending general message invocation ... to determined message control nod.” However, the Office Action does not use Stocker to disclose, nor does it teach or suggest, at least the above-noted distinguishing features of claims 26, 33, 39, 45, 51, and 57. Therefore, Stocker does not cure the deficiencies of Chandra.

Further, the Office Action, for example, on page 5, states, which Applicants do not acquiesce to, that Trenbeath teaches “subscribed clients C, D, E receives [sic] update data, attached data object in update messages from a publishing client A.” However, the Office Action does not use Trenbeath to disclose, nor does it teach or suggest, at least the above-noted distinguishing features of claims 26, 33, 39, 45, 51, and 57. Therefore, Trenbeath does not cure the deficiencies of Chandra and Stocker. Thus, the combination of Chandra, Stocker, and Trenbeath does not render the claims obvious.

Applicants therefore respectfully request the reconsideration and withdrawal of the rejection of claims 26, 33, 39, 45, 51, and 57 under 35 U.S.C. § 103(a). Accordingly, at least based on their respective dependencies to claims 26, 33, 39, 45, 51, and 57,

Atty. Dkt. No. 2222.775000E

PETITIONERS - AMERICAN/SOUTHWEST, Exhibit 1002
Page 968 of 1173

AA/SWA Ex. 1014, p.248 of 288
American Airlines, et. al. v. Intellectual Ventures, et. al.
IPR2025-01511

- 20-

Reply to Office Action of December 30, 2011

TUTTLE *et al.*
Appl. No. 11/396,251

claims 28, 34, 42-44, 48-50, 54-56, 60-62, 67, and 68 should be found allowable over the applied references, as well as for their additional distinguishing features.

Claims 29-31 and 35-38

Claims 29-31 and 35-38 were rejected under 35 U.S.C. § 103(a) as allegedly being unpatentable over Chandra in view of Stocker, further in view of Trenbeath, and in further view of U.S. Patent No. 7,209,959 to Campbell et al. (“Campbell”). Applicants respectfully traverse this rejection.

Dependent claims 29-31 and 35-38 are allowable at least because they recite all of the features of independent claims 26 and 33, from which they depend. As discussed above, Chandra, Stocker, and Trenbeath, taken alone or in combination, fail to disclose all of the features of claims 26 and 33.

Further, the Office Action, on pages 30-34, states, which Applicants do not acquiesce to, that Campbell teaches “causing the node to determine at least one client proxy with which the client communicates; causing the node to route to the client proxy; and causing the client proxy to rout the data to the client to determine the connection,” “client information is extracted and replaced with predetermined substitute information at server proxy which is considered as a client proxy,” and “wherein the node is configured to determine a connection by determining at least one client proxy with which the registered client communicates and to route the data to the client proxy.” However, the Office Action does not use Campbell to disclose, nor does it teach or suggest, at least the above-noted distinguishing features of claims 26 and 33. Therefore, Campbell does not cure the deficiencies of Chandra, Stocker, and Trenbeath.

Atty. Dkt. No. 2222.775000E

PETITIONERS - AMERICAN/SOUTHWEST, Exhibit 1002
Page 969 of 1173

AA/SWA Ex. 1014, p.249 of 288
American Airlines, et. al. v. Intellectual Ventures, et. al.
IPR2025-01511

- 21 -

Reply to Office Action of December 30, 2011

TUTTLE *et al.*
Appl. No. 11/396,251

Accordingly, at least based on their respective dependencies to claims 26 and 33, claims 29-31 and 35-38 should be found allowable over the applied references, as well as for their additional distinguishing features.

Claims 40, 41, 46, 47, 52, 53, 58, and 59

Claims 40, 41, 46, 47, 52, 53, 58, and 59 were rejected under 35 U.S.C. § 103(a) as allegedly being unpatentable over Chandra in view of Stocker, further in view of Trenbeath, and in further view of U.S. Patent No. 6,314,459 to Freeman ("Freeman"). Applicants respectfully traverse this rejection.

Dependent claims 40, 41, 46, 47, 52, 53, 58, and 59 are allowable at least because they recite all of the features of independent claims 39, 45, 51, and 57, from which they depend. As discussed above, Chandra, Stocker, and Trenbeath, taken alone or in combination, fail to disclose all of the features of claims 39, 45, 51, and 57.

Further, the Office Action, on pages 35-38, states, which Applicants do not acquiesce to, that Freeman teaches "wherein providing the data representation to the client device includes providing the live object that causes the client device to register with a client proxy of the routing network," "wherein providing the data representation to the client device includes providing the live object that causes the client device to register with the node of the routing network," "wherein the live object of the data representation is configured to cause the client device to register with a client proxy of the routing network," and "wherein the live object of the data representation is configured to cause the client device to register with the node of the routing network," However, the Office Action does not use Freeman to disclose, nor does it teach or

Atty. Dkt. No. 2222.775000E

- 22 -

Reply to Office Action of December 30, 2011

TUTTLE *et al.*
Appl. No. 11/396,251

suggest, at least the above-noted distinguishing features of claims 39, 45, 51, and 57.

Therefore, Freeman does not cure the deficiencies of Chandra, Stocker, and Trenbeath.

Accordingly, at least based on their respective dependencies to claims 39, 45, 51, and 57, claims 40, 41, 46, 47, 52, 53, 58, and 59 should be found allowable over the applied references, as well as for their additional distinguishing features.

Claims 63, 64, and 66

Claims 63, 64, and 66 were rejected under 35 U.S.C. § 103(a) as allegedly being unpatentable over Chandra in view of Stocker. Applicants respectfully traverse this rejection.

Independent claims 63, 64, and 66 recite features that distinguish over the applied references. For example, independent claim 63 recites, *inter alia*, (emphasis added):

wherein in response to determining that the client device is registered to receive update data for the live object, ***causing a category of the update message to be identified based on the input source***, a node type associated with the category be determined, and the update message be routed from the gateway device to a node having the node type at the routing network, and causing the node to route the update message to the client device.

Further, independent claim 64 recites, *inter alia*, (emphasis added):

wherein the routing network is configured to ***identify a category of the update message based on the input source***, determine a node type associated with the category, and to route the update message from a gateway device at the routing network to a node having the node type at the routing network, and wherein the node is configured to route the update message to the client device if the client device is registered to receive the update data for the live object.

Also, independent claim 66 recites, *inter alia*, (emphasis added):

Atty. Dkt. No. 2222.775000E

- 23 -

Reply to Office Action of December 30, 2011

TUTTLE *et al.*
Appl. No. 11/396,251

wherein *a category of the update message is identified based on the input source*, a node type associated with the category is determined, and the update message is routed from a gateway device at the routing network to a node having the node type at the routing network, wherein the update message is routed from the node to the client device in response to a determination that the client device is registered to receive the update data for the live object.

As discussed above with respect to independent claims 26, 33, 39, 45, 51, and 57, Chandra and Stocker, taken alone or in combination, fail to disclose the above-noted distinguishing features of claims 63, 64, and 66. Applicants therefore respectfully request the reconsideration and withdrawal of the rejection of claims 63, 64, and 66 under 35 U.S.C. § 103(a).

Atty. Dkt. No. 2222.775000E

PETITIONERS - AMERICAN/SOUTHWEST, Exhibit 1002
Page 972 of 1173

AA/SWA Ex. 1014, p.252 of 288
American Airlines, et. al. v. Intellectual Ventures, et. al.
IPR2025-01511

- 24-

Reply to Office Action of December 30, 2011

TUTTLE *et al.*
Appl. No. 11/396,251

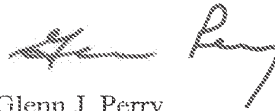
Conclusion

All of the stated grounds of rejection have been properly traversed, accommodated, or rendered moot. Applicants therefore respectfully request that the Examiner reconsider all presently outstanding rejections and that they be withdrawn. Applicants believe that a full and complete reply has been made to the outstanding Office Action and, as such, the present application is in condition for allowance. If the Examiner believes, for any reason, that personal communication will expedite prosecution of this application, the Examiner is invited to telephone the undersigned at the number provided.

Prompt and favorable consideration of this Amendment and Reply is respectfully requested.

Respectfully submitted,

STERNE, KESSLER, GOLDSTEIN & FOX P.L.L.C.



Glenn J. Perry
Attorney for Applicants
Registration No. 28,458

Date: 9 Feb. 2012

1100 New York Avenue, N.W.
Washington, D.C. 20005-3934
(202) 371-2600

1465715...1.DOCX

Atty. Dkt. No. 2222.775000E

PETITIONERS - AMERICAN/SOUTHWEST, Exhibit 1002
Page 973 of 1173

AA/SWA Ex. 1014, p.253 of 288
American Airlines, et. al. v. Intellectual Ventures, et. al.
IPR2025-01511

EXHIBIT A-10

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of:

TUTTLE *et al.*

Appl. No.: 11/396,251

Filed: March 30, 2006

For: **Asynchronous Messaging Using a
Node Specialization Architecture in
the Dynamic Routing Network**

Confirmation No.: 1129

Art Unit: 2194

Examiner: DAO, TUAN C.

Atty. Docket: 2222.775000E

Brief on Appeal Under 37 C.F.R. § 41.37

Mail Stop Appeal Brief - Patents

Commissioner for Patents
PO Box 1450
Alexandria, VA 22313-1450

Sir:

A Notice of Appeal from the Final Office Action, dated December 30, 2011, was filed March 28, 2012, along with a Pre Appeal Conference Request and Arguments. The Pre Appeal Brief Conference Decision was mailed May 1, 2012. Appellants hereby file one copy of this Appeal Brief, together with the required fee set forth in 37 C.F.R. § 41.20(b)(2).

It is not believed that extensions of time are required beyond those that may otherwise be provided for in documents accompanying this paper. However, if additional extensions of time are necessary to prevent abandonment of this application, then such extensions of time are hereby petitioned under 37 C.F.R. § 1.136(a), and any fees required therefor (including fees for net addition of claims) are hereby authorized to be charged to our Deposit Account No. 19-0036.

Table of Contents

I.	Real Party In Interest (37 C.F.R. § 41.37(c)(1)(i))	3
II.	Related Appeals and Interferences (37 C.F.R. § 41.37(c)(1)(ii))	4
III.	Summary of Claimed Subject Matter (37 C.F.R. § 41.37(c)(1)(iii))	5
	A. Independent Claim 26	5
	B. Independent Claim 33	6
	C. Independent Claim 39	7
	D. Independent Claim 45	8
	E. Independent Claim 51	9
	F. Independent Claim 57	10
	G. Independent Claim 63	11
	H. Independent Claim 64	11
	I. Independent Claim 66	12
IV.	Argument (37 C.F.R. § 41.37(c)(1)(iv))	13
	A. The Office committed reversible errors as the references lack at least the claimed identifying based on input source and rather Chandra teaches using a category/topic of message content	13
	B. Chandra, Stocker, and Trenbeath do not teach or suggest at least identifying a category of the update message based on the input source as respectively recited in claims 26, 33, 39, 45, 51, and 57	13
	C. Campbell does not cure the deficiencies of Chandra, Stocker, and Trenbeath	19
	D. Freeman does not cure the deficiencies of Chandra, Stocker, and Trenbeath	21
	E. Chandra and Stocker do not teach or suggest at least the claimed category of an updated message is identified based on the input source as respectively recited in claims 63, 64, and 66	22
	F. Conclusion	24
V.	Claims Appendix (37 C.F.R. § 41.37(c)(1)(v))	25

I. Real Party In Interest (37 C.F.R. § 41.37(c)(1)(i))

The real party in interest in this appeal is the Shaw Parsing LLC, having its principal place of business at 2215-B Renaissance Drive, Suite 5, Las Vegas, Nevada, 89119. An Assignment assigning all right, title, and interest in and to the patent application from the inventors to Bang Networks, Inc. was recorded in the Assignment Branch of the United States Patent and Trademark Office on June 21, 2007, at reel 019462, frame 0914. An Assignment assigning all right, title, and interest in and to the patent application from the Bang Networks, Inc. to Shaw Parsing LLC was recorded in the Assignment Branch of the United States Patent and Trademark Office on June 21, 2007, at reel 019462, frame 0936.

Atty. Dkt. No. 2222.775000E

II. Related Appeals and Interferences (37 C.F.R. § 41.37(c)(1)(ii))

To the best of the knowledge of Appellants, Appellants' legal representative, and Appellants' assignee, there are no other appeals, interferences, or judicial proceedings which are related to, directly affect, or are directly affected by or have a bearing on a decision by the Board of Patent Appeals and Interferences ("the Board") in the pending appeal.

Appellants will comply with 37 C.F.R. § 41.8(a) and will notify the Board of any change during the proceeding, within 20 days of the change, in (1) the real party-in-interest, and (2) each judicial or administrative proceeding that could affect, or be affected by, the Board proceeding.

III. Summary of Claimed Subject Matter (37 C.F.R. § 41.37(c)(1)(iii))

A concise explanation of the subject matter recited in the independent claims on appeal (i.e., claims 26, 33, 39, 45, 51, 57, 63, 64, and 66) is provided below. The explanation refers to the specification by page and line number and to the drawings by reference characters. Reference is made to example supporting embodiments disclosed in the specification, although it is understood that the claims should not be limited to the specific embodiments to which reference is made.

A. Independent Claim 26

<i>Claim Element</i>	<i>Example Support</i>
A method comprising:	FIGs. 7, 8, and 10
receiving, using a processing device, an update message from an input source, the update message identifying a live object and containing data for updating a property of the live object;	FIG. 7, element 724A; FIG. 8, element 810; FIG. 10, element 1010; page 6, lines 2-10; page 38, lines 10-11; page 46, line 20 – page 47, line 1
<i>identifying a category of the update message based on the input source;</i>	FIG. 7, element 724A; FIG. 10, element 1020; page 39, lines 3-7; page 40, lines 3-18; page 47, lines 2-3 [emphasis added for at least one distinguishing feature of the claim]
determining a node having a node type to which the update message is to be routed based on a mapping of categories of update messages to node types, the mapping controlling an amount of update message traffic through nodes of a routing network;	FIG. 7, elements 724A and 726A; FIG. 10, element 1030; page 39, lines 3-7; page 47, lines 3-5
routing, using the processing device, the update message to the node having the determined node type;	FIG. 7, elements 724A and 726A; FIG. 10, element 1040; page 47, lines 3-5
causing the node, through the update message, to determine a client, different from the input source, that has registered for updates of the live object;	FIG. 7, elements 724A, 726A, 712; FIG. 8, element 850; FIG. 10, element 1050; page 38, lines 14-16; page 47, lines 5-9
causing the node to route the update	FIG. 7, elements 724A, 726A, 712; FIG. 8,

Atty. Dkt. No. 2222,775000E

message from the node to the client; and	element 860; FIG. 10, elements 1060 and 1070; page 38, lines 14-16; page 47, lines 9-12
causing the client to process the update message and to update the property of the live object.	page 16, line 18 – page 17, line 5

B. Independent Claim 33

<i>Claim Element</i>	<i>Example Support</i>
A routing network comprising:	FIG. 3, element 110
a gateway device configured to:	FIG. 7, element 724A
receive an update message from an input source, the update message identifying a live object and containing data for updating a property of the live object,	FIG. 7, element 724A; FIG. 8, element 810; FIG. 10, element 1010; page 6, lines 2-10; page 38, lines 10-11; page 46, line 20 – page 47, line 1
<i>identify a category of the update message based on the input source;</i>	FIG. 7, element 724A; FIG. 10, element 1020; page 39, lines 3-7; page 40, lines 3-18; page 47, lines 2-3 [emphasis added for at least one distinguishing feature of the claim]
determine a node having a node type to which the update message is to be routed based on a mapping of a categories of update messages to node types, the mapping controlling an amount of update message traffic through nodes of a routing network; and	FIG. 7, element 724A; FIG. 10, element 1030; page 39, lines 3-7; page 47, lines 3-5
route the update message; and	FIG. 7, element 724A; FIG. 10, element 1040; page 47, lines 3-5
the node configured to:	FIG. 7, element 726A
receive the update message from the gateway device, wherein the node is configured to be mapped to the node type,	FIG. 7, elements 724A and 726A; FIG. 10, element 1040; page 39, lines 3-7; page 47, lines 3-5
determine a client, different from the input source, that has registered for updates of the live object, and	FIG. 7, elements 724A, 726A, 712; FIG. 8, element 850; FIG. 10, element 1050; page 38, lines 14-16; page 47, lines 5-9
route the update message from the node to the client, wherein the client is adapted to process the update message and to update the property of the live object.	FIG. 7, elements 724A, 726A, 712; FIG. 8, element 860; FIG. 10, elements 1060 and 1070; page 16, line 18 – page 17, line 5; page 38, lines 14-16; page 47, lines 9-12

C. Independent Claim 39

<i>Claim Element</i>	<i>Example Support</i>
A method comprising:	FIGs. 7, 8, and 10
providing, using a processing device of an input source, a data representation to a client device, different from the input source, coupled to a routing network, wherein the data representation includes at least one live object recognizable by the client device, and causing the client device to respond to the live object of the data representation by determining an object ID of the live object and to register for updates of the live object with the routing network, such that registering the client device with the routing network provides client connection information to a node in the routing network; and	FIG. 3, element 210; FIG. 7, element 710; page 11, lines 3-19
sending, using the processing device of the input source, an update message to the routing network, wherein the update message identifies the live object and contains update data that updates a property of the live object,	FIG. 3, element 210; FIG. 7, element 710; FIG. 8, element 810; FIG. 10, element 1010; page 6, lines 2-10; page 38, lines 10-11; page 46, line 20 – page 47, line 1
wherein a gateway device at the routing network is configured to <i>identify a category of the update message based on the input source</i> , to determine a node type to which the identified category maps, and to route the update message to the node, having the node type, at the routing network,	FIG. 7, element 724A; FIG. 8, element 810; FIG. 10, elements 1010, 1020, and 1030; page 6, lines 2-10; page 38, lines 10-11; page 39, lines 3-7; page 40, lines 3-18; page 46, line 20 – page 47, lines 1-5 [emphasis added for at least one distinguishing feature of the claim]
wherein the node is configured to identify the client device as a registered device and to route the update message to the client device, and	FIG. 7, elements 724A, 726A, 712; FIG. 8, elements 850 and 860; FIG. 10, elements 1050, 1060, and 1070; page 38, lines 14-16; page 47, lines 5-12
wherein the client device processes the update message upon receipt to update the property of the live object at the client device.	page 16, line 18 – page 17, line 5

D. Independent Claim 45

<i>Claim Element</i>	<i>Example Support</i>
An apparatus comprising:	FIGs. 3 and 7
an input source device configured to provide a data representation to a client device, different from the input source, coupled to a routing network, wherein the data representation includes at least one live object that is recognizable by the client device, and that causes the client device to determine an object ID of the live object to register for updates of the live object with the routing network, such that registering the client device with the routing network provides client connection information to the routing network,	FIG. 3, element 210; FIG. 7, element 710; page 11, lines 3-19
wherein the input source device is configured to route an update message to the routing network, wherein the update message identifies the live object and contains update data for updating a property of the live object,	FIG. 3, element 210; FIG. 7, element 710; FIG. 8, element 810; FIG. 10, element 1010; page 6, lines 2-10; page 38, lines 10-11; page 46, line 20 – page 47, line 1
wherein a gateway device at the network is configured to <i>identify a category of the update message based on the input source</i> , to determine a node type to which the identified category maps, and to route the update message to a node of the node type at the routing network,	FIG. 7, element 724A; FIG. 8, element 810; FIG. 10, elements 1010, 1020, and 1030; page 6, lines 2-10; page 38, lines 10-11; page 39, lines 3-7; page 40, lines 3-18; page 46, line 20 – page 47, lines 1-5 [emphasis added for at least one distinguishing feature of the claim]
wherein the node is configured to identify the client device as a registered device and route the update message containing the update data to the client device, and	FIG. 7, elements 724A, 726A, 712; FIG. 8, elements 850 and 860; FIG. 10, elements 1050, 1060, and 1070; page 38, lines 14-16; page 47, lines 5-12
wherein the client device is configured to process the update message upon receipt to update the property of the live object at the client device.	page 16, line 18 – page 17, line 5

E. Independent Claim 51

<i>Claim Element</i>	<i>Example Support</i>
An article of manufacture including a computer-readable storage medium having instructions stored thereon, execution of which by a computing device causes the computing device to perform operations comprising:	FIGs. 7, 8, and 10; page 12, line 20 – page 13, line 11; page 20, line 14-18; page 21, line 13 – page 22, line 15
providing, using a processing device of an input source, a data representation to a client device, different from the input source, coupled to a routing network, wherein the data representation includes at least one live object that is recognizable by the client device, and that causes the client device to respond to the live object by determining an object ID of the live object to register for updates of the live object with the routing network, such that registering the client device with the routing network provides client connection information to the routing network; and	FIG. 3, element 210; FIG. 7, element 710; page 11, lines 3-19
sending, using the processing device of the input source, an update message to the routing network, wherein the update message identifies the live object and contains update data for updating a property of the live object,	FIG. 3, element 210; FIG. 7, element 710; FIG. 8, element 810; FIG. 10, element 1010; page 6, lines 2-10; page 38, lines 10-11; page 46, line 20 – page 47, line 1
wherein a gateway device at the routing network is configured to identify a category of the update message based on the input source , to determine a node type to which the identified category maps, and to route the update message to a node of the node type at the routing network,	FIG. 7, element 724A; FIG. 8, element 810; FIG. 10, elements 1010, 1020, and 1030; page 6, lines 2-10; page 38, lines 10-11; page 39, lines 3-7; page 40, lines 3-18; page 46, line 20 – page 47, lines 1-5 [emphasis added for at least one distinguishing feature of the claim]
wherein the node is configured to identify the client device as a registered device and to route the update message to the client device, and	FIG. 7, elements 724A, 726A, 712; FIG. 8, elements 850 and 860; FIG. 10, elements 1050, 1060, and 1070; page 38, lines 14-16; page 47, lines 5-12
wherein the client device is configured to process the update message upon receipt to update the property of the live object at the client device.	page 16, line 18 – page 17, line 5

F. Independent Claim 57

<i>Claim Element</i>	<i>Example Support</i>
A non-transitory computer readable storage medium having instructions stored thereon, the instructions comprising:	FIGs. 7, 8, and 10; page 12, line 20 – page 13, line 11; page 20, line 14-18; page 21, line 13 – page 22, line 15
instructions for providing a data representation to a client device coupled to a routing network, wherein the data representation includes at least one live object recognizable by the client device, and wherein the client device is configured to respond to the live object of the data representation by determining an object ID of the live object to register for updates of the live object with the routing network, such that registering the client device with the routing network provides client connection information to the routing network; and	FIG. 3, element 210; FIG. 7, element 710; page 11, lines 3-19
instructions for providing, using a processing device of an input source, different from the client device, an update message to the routing network, wherein the update message identifies the live object and contains update data for updating a property of the live object,	FIG. 3, element 210; FIG. 7, element 710; FIG. 8, element 810; FIG. 10, element 1010; page 6, lines 2-10; page 38, lines 10-11; page 46, line 20 – page 47, line 1
wherein a gateway device at the routing network is configured to <i>identify a category of the update message based on the input source</i> , to determine a node type to which the identified category maps, and to route the update message to a node of the node type at the routing network,	FIG. 7, element 724A; FIG. 8, element 810; FIG. 10, elements 1010, 1020, and 1030; page 6, lines 2-10; page 38, lines 10-11; page 39, lines 3-7; page 40, lines 3-18; page 46, line 20 – page 47, lines 1-5 [emphasis added for at least one distinguishing feature of the claim]
wherein the node is configured to identify the client device as a registered device and to route the update message containing the update data to the client device, and	FIG. 7, elements 724A, 726A, 712; FIG. 8, elements 850 and 860; FIG. 10, elements 1050, 1060, and 1070; page 38, lines 14-16; page 47, lines 5-12
wherein the client device is configured to process the update message upon receipt to update the property of the live object at the client device.	page 16, line 18 – page 17, line 5

G. Independent Claim 63

<i>Claim Element</i>	<i>Example Support</i>
A method comprising:	FIGs. 7, 8, and 10
providing a live object to a client device;	page 11, lines 3-19
sending, using a processing device of an input source, different from the client device, an update message to a routing network identifying the live object and containing update data to update the live object at the client device,	FIG. 3, element 210; FIG. 7, element 710; FIG. 8, element 810; FIG. 10, element 1010; page 6, lines 2-10; page 38, lines 10-11; page 46, line 20 – page 47, line 1
wherein in response to determining that the client device is registered to receive update data for the live object, causing <i>a category of the update message to be identified based on the input source</i> , a node type associated with the category be determined, and the update message be routed from the gateway device to a node having the node type at the routing network, and causing the node to route the update message to the client device.	FIG. 7, elements 724A, 726A, 712; FIG. 8, element 810, 850, and 860; FIG. 10, elements 1010, 1020, 1030, 1050, 1060, and 1070; page 6, lines 2-10; page 38, lines 10-16; page 39, lines 3-7; page 40, lines 3-18; page 46, line 20 – page 47, lines 1-12 [emphasis added for at least one distinguishing feature of the claim]

H. Independent Claim 64

<i>Claim Element</i>	<i>Example Support</i>
An apparatus comprising:	FIGs. 3 and 7
an input source arranged to provide a live object to a client device, different from the input source; and	FIG. 3, element 210; FIG. 7, element 710; page 11, lines 3-19
the input source arranged to provide an update message to a routing network identifying the live object and containing update data for updating the live object,	FIG. 3, element 210; FIG. 7, element 710; FIG. 8, element 810; FIG. 10, element 1010; page 6, lines 2-10; page 38, lines 10-11; page 46, line 20 – page 47, line 1
wherein the routing network is configured to <i>identify a category of the update message based on the input source</i> , determine a node type associated with the category, and to route the update message from a gateway device at the routing network to a node having the node type at the routing network, and wherein the node is configured to route the update message to the client device if the client device is registered to receive	FIG. 7, elements 724A, 726A, 712; FIG. 8, element 810, 850, and 860; FIG. 10, elements 1010, 1020, 1030, 1050, 1060, and 1070; page 6, lines 2-10; page 38, lines 10-16; page 39, lines 3-7; page 40, lines 3-18; page 46, line 20 – page 47, lines 1-12 [emphasis added for at least one distinguishing feature of the claim]

the update data for the live object.	
--------------------------------------	--

I. Independent Claim 66

<i>Claim Element</i>	<i>Example Support</i>
A non-transitory computer readable storage medium having instructions stored thereon, the instructions comprising:	FIGs. 7, 8, and 10; page 12, line 20 – page 13, line 11; page 20, line 14-18; page 21, line 13 – page 22, line 15
instructions for providing a live object to a client device; and	page 11, lines 3-19
instructions for providing an update message from an input source, different from the client device, to a routing network identifying the live object and containing update data for updating the live object at the client device,	FIG. 3, element 210; FIG. 7, element 710; FIG. 8, element 810; FIG. 10, element 1010; page 6, lines 2-10; page 38, lines 10-11; page 46, line 20 – page 47, line 1
wherein <i>a category of the update message is identified based on the input source</i> , a node type associated with the category is determined, and the update message is routed from a gateway device at the routing network to a node having the node type at the routing network, wherein the update message is routed from the node to the client device in response to a determination that the client device is registered to receive the update data for the live object.	FIG. 7, elements 724A, 726A, 712; FIG. 8, element 810, 850, and 860; FIG. 10, elements 1010, 1020, 1030, 1050, 1060, and 1070; page 6, lines 2-10; page 38, lines 10-16; page 39, lines 3-7; page 40, lines 3-18; page 46, line 20 – page 47, lines 1-12 [emphasis added for at least one distinguishing feature of the claim]

IV. Argument (37 C.F.R. § 41.37(c)(1)(iv))

A. *The Office committed reversible errors as the references lack at least the claimed identifying based on input source and rather Chandra teaches using a category/topic of message content*

Appellants demonstrate Chandra, Stocker, Trenbeath, Campbell and Freeman do not teach or suggest at least the claimed identifying a category of the update message based on the input source *or* an updated message is identified based on the input source. Rather, the reference relied upon by the Office, Chandra, merely teaches using a category/topic of message content. Thus, as the claims are allowable over the references, the Office should be reversed and the claims passed to issue.

B. *Chandra, Stocker, and Trenbeath do not teach or suggest at least identifying a category of the update message based on the input source as respectively recited in claims 26, 33, 39, 45, 51, and 57*

The Examiner maintained the rejection of claims 26, 28, 33, 34, 39, 42-45, 48-51, 54-57, 60-62, 67, and 68 under 35 U.S.C. § 103(a) as allegedly being unpatentable over Chandra in view of Stocker, and further in view of Trenbeath. (Final Office Action mailed December 30, 2011 (“Final Office Action”), pp. 2-30; Advisory Action mailed February 28, 2012 (“Advisory Action”), pp. 2.)

Independent claims 26, 33, 39, 45, 51, and 57 recite a combination of features not taught or suggested by the applied references.

For example, independent claim 26 recites, *inter alia*, (emphasis added):

identifying a category of the update message based on the input source.

Independent claim 33 recites similar distinguishing features, using its respective language. Further, independent claim 39 recites, *inter alia*, (emphasis added):

wherein a gateway device at the routing network is configured to ***identify a category of the update message based on the input source***, to determine a node type to which the identified category maps, and to route the update message to the node, having the node type, at the routing network.

Independent claims 45, 51, and 57 recite similar distinguishing features, using their respective language.

The combination of Chandra, Stocker, and Trenbeath does not teach or suggest at least the above-noted distinguishing features, and the Examiner's rejection based on 35 U.S.C. § 103(a) is therefore improper as being legally and factually deficient.

The Examiner relies on Chandra for this claim feature. (Final Office Action, p. 3 citing Chandra 1:25-35 and 5:45-65 (in view of paragraphs 103 and 105 of the specification of instant application).) Appellants disagree. Rather than teaching or suggesting at least “identify a category of the update message ***based on the input source***,” as respectfully recited in claims, Chandra teaches identifying categories ***based on category/topic of the message content***.

Chandra discloses a method for “routing messages within a network using the data content of the message.” (Chandra, Title.) When discussing the background art, Chandra discloses that (emphasis added):

One example of a system that includes such a network environment is a publish/subscribe system. In publish/subscribe systems, publishers post messages and ***subscribers independently specify categories of events in which they are interested***. The

system takes the posted messages and includes in each message header the destination information of those subscribers indicating interest in the particular message. The system then uses the destination information in the message to forward the message through the network to the appropriate subscribers.

(Chandra, 1:26-35.)

Chandra also discusses the content based subscription, where “one or more of the clients can specify a subscription. Each subscription provides a set of conditions indicating the types of messages those clients are interested in receiving.” (Chandra, 5:45-48.) Chandra further discloses that:

Consider an example in which messages are related to the stock market. In such an example, a subscription may have the following three fields (or attributes): stock issue, stock price and stock value. The client specifies values for these fields, e.g., IBM, 100, 200, and then, these fields are used to determine if the client is interested in a particular message. *If the data content of the message matches the specified values of the subscription, then the message is delivered to the one or more subscribers (or clients) of that subscription.* It will be apparent to those skilled in the art that the use of three fields in the subscription is only one example. Subscriptions may include any number of desired fields and tests may include relationships other than equality. Further, the stock market scenario is only one example. Messages and subscriptions can be used for any type of data or events.

(Chandra, 5:53-67; emphasis added.)

These sections of Chandra, relied on by the Examiner, merely disclose that subscribers can specify categories of events in which they are interested and if the *data content of the message* matches the specified values of the subscription, then the message is delivered to the one or more subscribers (or clients) of that subscription. This is not the same as “identif[ying] a category of the update message *based on the input source*,” as recited in claims 26, 33, 39, 45, 51, and 57, using their respective language.

The Examiner also states:

Chandra column 1:25-35; column 5:45-65 discloses “publisher post messages and subscribers independently specify categories or [sic] event [sic] in which they are interested” and “if the data content of the message matches the specified value of the subscription, then the message is delivered [to] the one or more subscribers (or clients)” → ***identifying each category based on the category/topic of the message content***);

(Final Office Action, p. 3; emphasis added.) This proves the Examiner agrees that Chandra teaches identifying categories ***based on category/topic of the message content***. In contrast, claims 26, 33, 39, 45, 51, and 57 recite, *inter alia*, “identify a category of the update message ***based on the input source***,” using their respective language.

Appellant rebutted the Examiner stating:

The Examiner, for example, on page 3 of the Office Action, relies on the above sections of Chandra [Chandra, 1:25-35 and 5:45-65] to allegedly show “identifying each category based on the category/topic of the message content.” By this statement, the Examiner appears to agree that Chandra teaches identifying categories **based on category/topic of the message content**. In contrast, claims 26, 33, 39, 45, 51, and 57 recite, *inter alia*, “identify a category of the update message **based on the input source**,” using their respective language.

(Reply under 37 C.F.R. 1.116 filed February 9, 2012 (“After Final Response”), pp. 18-19)

The Examiner then responded:

More specifically, in Chandra, a publish/subscriber system and publisher is equivalent as input source and a specific publisher publishes/posts messages related to “sport”, “weather”, and “stocks”. A particular subscriber selects category “stocks” (which he/she is interested” [sic] is equivalent identifying a category “stocks” of update messages based on available categories (“sport”, “weather”, and “stock”) of the publisher/input source. Put another way, if said specific publisher only

Atty. Dkt. No. 2222.775000E

publishes/posts messages related to “sports”, “weather”, “celebrities”, then the particular subscriber cannot identify the category “stocks” (which he/she is interested) and should subscribes [sic] for another publisher based on available categories of the another publisher. Accordingly, Chandra indeed fully discloses identifying a category of the update message based [sic] a plurality of available categories of the input source.

(Advisory Action, p. 2.) Appellants respectfully disagree.

First, the Examiner states that “if said specific publisher only publishes/posts messages related to ‘sports’, ‘weather’, ‘celebrities’, then the *particular subscriber cannot identify the category ‘stocks’ (which he/she is interested).*” (Advisory Action, p. 2, emphasis added.) Appellants respectfully disagree. Chandra specifically discloses that subscriptions are based on the *type of messages those clients are interested* in receiving (Chandra, 5:46-48.) Therefore, following the Examiner’s example, if the particular subscriber is interested in “stocks” type of messages, Chandra discloses that the particular subscriber will subscribe to “stocks” type of messages no matter what a specific publisher publishes. Therefore, the Examiner’s statement that “particular subscriber *cannot identify* the category ‘stocks’ (which he/she is interested)” because the specific publisher only publishes/posts messages related to “sports,” “weather,” and “celebrities” (Advisory Action, p. 2, emphasis added) is not correct.

Second, the Examiner states that “if said specific publisher only publishes/posts messages related to ‘sports’, ‘weather’, ‘celebrities’, then the particular subscriber cannot identify the category ‘stocks’ (which he/she is interested) and *should subscribes [sic] for another publisher based on available categories of the another publisher.*” (Advisory Action, p. 2, emphasis added.) Appellants respectfully disagree. Chandra specifically states

that “[e]ach subscription provides a set of conditions indicating the *type of messages those clients are interested* in receiving.” (Chandra, 5:46-48, emphasis added.) Therefore, Chandra discloses that subscriptions are based on *type of messages* that clients are interested in, and not based on available categories of *another publisher*, as the Examiner suggests.

Moreover, following the Examiner’s example, if a particular subscriber selects a category of “stocks”, the particular subscriber is subscribed to “stocks” type of messages. Chandra determines whether a data content of a message matches the specified values of the subscription (such as “stock”), then the message is delivered to the particular subscriber. Chandra does not identify messages based on the publisher, which the Examiner analogizes as input source of independent claims, to which Applicants do not concede.

Therefore, Chandra, the only reference relied upon by the Examiner, does not teach or suggest “identifying a category of the update message based on the input source” *or* “an updated message is identified based on the input source,” as respectively claimed in claims 26, 33, 39, 45, 51, and 57, but rather Chandra teaches identifying categories based on category/topic of the message content. The Examiner does not rely on Stocker, Trenbeath, Campbell, and Freeman to disclose at least this distinguishing feature, nor do these references teach or suggest at least the above noted distinguishing feature.

With regards to Stocker, the Examiner alleges Stocker teaches “GIMR 300 determining a message control node 210 to send a general information message 220 to MSS 110,” “if the cell serving mobile station in part of the served area 195, the step 430 is entered,” “determining if the served area ... associated with the MS 100,” “routing the update message to the node having determined node type,” and “sending general message

Atty. Dkt. No. 2222.775000E

invocation ... to determined message control nod.” (Final Office Action, pp. 4-5.) However, the Examiner does not use Stocker to disclose, nor does it teach or suggest, at least the above-noted distinguishing features of claims 26, 33, 39, 45, 51, and 57. Therefore, Stocker does not cure the deficiencies of Chandra.

With regards to Trenbeath, the Examiner alleges Trenbeath teaches “subscribed clients C, D, E receives [sic] update data, attached data object in update messages from a publishing client A.” (Final Office Action, p. 5.) However, the Examiner does not use Trenbeath to disclose, nor does it teach or suggest, at least the above-noted distinguishing features of claims 26, 33, 39, 45, 51, and 57. Therefore, Trenbeath does not cure the deficiencies of Chandra and Stocker.

Thus, as neither Stocker nor Trenbeath cures the deficiencies of Chandra, the applied references cannot be used to establish a prima facie case of obviousness.

Accordingly, Appellants respectfully request reconsideration and withdrawal of the rejection of independent claims 26, 33, 39, 45, 51, and 57, and they be found allowable over the applied references. Also, dependent claims 28, 34, 42-44, 48-50, 54-56, 60-62, 67, and 68 are allowable for at least the same reasons as independent claims 26, 33, 39, 45, 51, and 57 are allowable, and further for their own respective features.

C. Campbell does not cure the deficiencies of Chandra, Stocker, and Trenbeath

The Examiner rejected claims 29-31 and 35-38 under 35 U.S.C. § 103(a) as allegedly being unpatentable over Chandra in view of Stocker, further in view of Trenbeath, and in further view of Campbell.

Atty. Dkt. No. 2222.775000E

Dependent claims 29-31 and 35-38 are allowable at least because they recite all of the features of independent claims 26 and 33, from which they depend. As discussed above, the combination of Chandra, Stocker, and Trenbeath fails to disclose all of the features of claims 26 and 33.

The Examiner alleges Campbell teaches “causing the node to determine at least one client proxy with which the client communicates; causing the node to route to the client proxy; and causing the client proxy to rout the data to the client to determine the connection,” “client information is extracted and replaced with predetermined substitute information at server proxy which is considered as a client proxy,” and “wherein the node is configured to determine a connection by determining at least one client proxy with which the registered client communicates and to route the data to the client proxy.” (Final Office Action, pp. 30-34). Appellants disagree. However, the Examiner does not use Campbell to disclose, nor does it teach or suggest, at least the above-noted distinguishing features of claims 26 and 33. Therefore, as Campbell does not cure the deficiencies of Chandra, Stocker, and Trenbeath, the applied references cannot be used to establish a *prima facie* case of obviousness for claims 26 and 33.

Accordingly, at least based on their respective dependency to claim 26 and 33, claims 29-31 and 35-38 should be found allowable over the applied references, as well as for their additional distinguishing features.

Atty. Dkt. No. 2222.775000E

D. Freeman does not cure the deficiencies of Chandra, Stocker, and Trenbeath

The Examiner rejected claims 40, 41, 46, 47, 52, 53, 58, and 59 under 35 U.S.C. § 103(a) as allegedly being unpatentable over Chandra in view of Stocker, further in view of Trenbeath, and in further view of Freeman.

Dependent claims 40, 41, 46, 47, 52, 53, 58, and 59 are allowable at least because they recite all of the features of independent claims 39, 45, 51, and 57, from which they depend. As discussed above, the combination of Chandra, Stocker, and Trenbeath fails to disclose all of the features of claims 39, 45, 51, and 57.

The Examiner alleges Freeman teaches “wherein providing the data representation to the client device includes providing the live object that causes the client device to register with a client proxy of the routing network,” “wherein providing the data representation to the client device includes providing the live object that causes the client device to register with the node of the routing network,” “wherein the live object of the data representation is configured to cause the client device to register with a client proxy of the routing network,” and “wherein the live object of the data representation is configured to cause the client device to register with the node of the routing network.” (Final Office Action, on pages 35-38.) Appellants disagree. However, the Examiner does not use Freeman to disclose, nor does it teach or suggest, at least the above-noted distinguishing features of claims 39, 45, 51, and 57. Therefore, as Freeman does not cure the deficiencies of Chandra, Stocker, and Trenbeath, the applied references cannot be used to establish a *prima facie* case of obviousness for claims 39, 45, 51, and 57.

Atty. Dkt. No. 2222.775000E

Accordingly, at least based on their respective dependency to claim 39, 45, 51, and 57, claims 40, 41, 46, 47, 52, 53, 58, and 59 should be found allowable over the applied references, as well as for their additional distinguishing features.

E. Chandra and Stocker do not teach or suggest at least the claimed category of an updated message is identified based on the input source as respectively recited in claims 63, 64, and 66

The Examiner maintained the rejections of claims 63, 64, and 66 under 35 U.S.C. § 103(a) as allegedly being unpatentable over Chandra in view of Stocker. (Final Office Action, pp. 38-44; Advisory p. 2.)

Independent claims 63, 64, and 66 recite features that distinguish over the applied references. For example, independent claim 63 recites, *inter alia*, (emphasis added):

wherein in response to determining that the client device is registered to receive update data for the live object, ***causing a category of the update message to be identified based on the input source***, a node type associated with the category be determined, and the update message be routed from the gateway device to a node having the node type at the routing network, and causing the node to route the update message to the client device.

Further, independent claim 64 recites, *inter alia*, (emphasis added):

wherein the routing network is configured to ***identify a category of the update message based on the input source***, determine a node type associated with the category, and to route the update message from a gateway device at the routing network to a node having the node type at the routing network, and wherein the node is configured to route the update message to the client device if the client device is registered to receive the update data for the live object.

Also, independent claim 66 recites, *inter alia*, (emphasis added):

wherein *a category of the update message is identified based on the input source*, a node type associated with the category is determined, and the update message is routed from a gateway device at the routing network to a node having the node type at the routing network, wherein the update message is routed from the node to the client device in response to a determination that the client device is registered to receive the update data for the live object.

The Examiner again relies on Chandra to allegedly teach the above-noted distinguishing features. (Final Office Action, pp. 38-44.) Appellant disagrees for reasons similar to those discussed above. While these claims recite different distinguishing features, i.e., an updated message is identified based on the input source, the reference relied upon by the Office, Chandra, merely teaches using a category/topic of message content. Thus, for similar reasons as discussed above with respect to independent claims 26, 33, 39, 45, 51, and 57, Chandra and Stocker, taken alone or in combination, fail to disclose at least the above-noted distinguishing features of claims 63, 64, and 66. Applicants therefore respectfully request the reconsideration and withdrawal of the rejection of claims 63, 64, and 66 under 35 U.S.C. § 103(a).

F. Conclusion

Therefore, as the Office has failed to show the applied references teach at least the claimed identifying a category of the update message based on the input source *or* an updated message is identified based on the input source, Appellants respectfully request that the Board reverse the Office's final rejection of claims 26, 28-31, 33-64, and 66-68, and remand this application for allowance of claims 26, 28-31, 33-64, and 66-68.

Respectfully submitted,

STERNE, KESSLER, GOLDSTEIN & FOX P.L.L.C.

Jason D. Eisenberg
Attorney for Appellants
Registration No. 43,447

Date: 5/31/12

1100 New York Avenue, N.W.
Washington, D.C. 20005-3934
(202) 371-2600
1532103_2.docx

V. *Claims Appendix (37 C.F.R. § 41.37(c)(i)(v))*

1-25. (Cancelled)

26. A method comprising:

receiving, using a processing device, an update message from an input source, the update message identifying a live object and containing data for updating a property of the live object;

identifying a category of the update message based on the input source;

determining a node having a node type to which the update message is to be routed based on a mapping of categories of update messages to node types, the mapping controlling an amount of update message traffic through nodes of a routing network;

routing, using the processing device, the update message to the node having the determined node type;

causing the node, through the update message, to determine a client, different from the input source, that has registered for updates of the live object;

causing the node to route the update message from the node to the client; and

causing the client to process the update message and to update the property of the live object.

27. (Cancelled)

28. The method of claim 26, further comprising:

causing the node to extract an object ID from the update message, to establish a connection with the client, and to determine if the client has registered for updates of the live object based on the object ID.

29. The method of claim 28, further comprising:

causing the node to determine at least one client proxy with which the client communicates;

causing the node to route the update message to the client proxy; and

causing the client proxy to route the update message to the client to determine the connection.

30. The method of claim 29, further comprising causing the node to maintain client registration information concerning the client connection.

31. The method of claim 29, further comprising causing the node to maintain client registration information concerning the client connection at the client proxy.

32. (Cancelled)

33. A routing network comprising:
a gateway device configured to:

receive an update message from an input source, the update message identifying a live object and containing data for updating a property of the live object,

identify a category of the update message based on the input source;

determine a node having a node type to which the update message is to be routed based on a mapping of a categories of update messages to node types, the mapping controlling an amount of update message traffic through nodes of a routing network; and

route the update message; and

the node configured to:

receive the update message from the gateway device, wherein the node is configured to be mapped to the node type,

determine a client, different from the input source, that has registered for updates of the live object, and

route the update message from the node to the client, wherein the client is adapted to process the update message and to update the property of the live object.

34. The routing network of claim 33, wherein the node is configured to extract an object ID from the update message and to determine a connection to the client to route the data to the client.

Atty. Dkt. No. 2222.775000E

35. The routing network of claim 34, wherein the node is configured to determine the connection by determining at least one client proxy with which the client communicates and to route the update message to the client proxy.
36. The routing network of claim 35, wherein client registration information concerning the client connection is configured to be maintained at the node.
37. The routing network of claim 35, wherein client registration information concerning the client connection is configured to be maintained at the client proxy.
38. The routing network of claim 33, wherein:
the gateway device is configured to route the update message to the node adapted to receive messages of more than one message category, and
the node is configured to determine a connection with at least one client proxy with which the client communicates and to route the data to the client proxy.
39. A method comprising:
providing, using a processing device of an input source, a data representation to a client device, different from the input source, coupled to a routing network, wherein the data representation includes at least one live object recognizable by the client device, and causing the client device to respond to the live object of the data representation by determining an object ID of the live object and to register for updates of the live object with the routing network, such that registering the client device with the routing network provides client connection information to a node in the routing network; and
sending, using the processing device of the input source, an update message to the routing network, wherein the update message identifies the live object and contains update data that updates a property of the live object,
wherein a gateway device at the routing network is configured to identify a category of the update message based on the input source, to determine a node type to which the

identified category maps, and to route the update message to the node, having the node type, at the routing network,

wherein the node is configured to identify the client device as a registered device and to route the update message to the client device, and

wherein the client device processes the update message upon receipt to update the property of the live object at the client device.

40. The method of claim 39, wherein providing the data representation to the client device includes providing the live object that causes the client device to register with a client proxy of the routing network.

41. The method of claim 39, wherein providing the data representation to the client device includes providing the live object that causes the client device to register with the node of the routing network.

42. The method of claim 39, wherein providing the data representation to the client device includes providing an activation module that is executed by the client device and that registers the live object with the routing network.

43. The method of claim 42, wherein providing the activation module includes providing the activation module that is configured to determine a node type that handles registration and that causes the client device to register with the node corresponding to the node type.

44. The method of claim 42, wherein providing the activation module includes providing the activation module that is configured to determine a message category of the data representation and that causes the client device to register with a node having a node type corresponding to the message category.

45. An apparatus comprising:
an input source device configured to provide a data representation to a client device, different from the input source, coupled to a routing network, wherein the data

representation includes at least one live object that is recognizable by the client device, and that causes the client device to determine an object ID of the live object to register for updates of the live object with the routing network, such that registering the client device with the routing network provides client connection information to the routing network,

wherein the input source device is configured to route an update message to the routing network, wherein the update message identifies the live object and contains update data for updating a property of the live object,

wherein a gateway device at the network is configured to identify a category of the update message based on the input source, to determine a node type to which the identified category maps, and to route the update message to a node of the node type at the routing network,

wherein the node is configured to identify the client device as a registered device and route the update message containing the update data to the client device, and

wherein the client device is configured to process the update message upon receipt to update the property of the live object at the client device.

46. The apparatus of claim 45, wherein the live object of the data representation is configured to cause the client device to register with a client proxy of the routing network.

47. The apparatus of claim 45, wherein the live object of the data representation is configured to cause the client device to register with the node of the routing network.

48. The apparatus of claim 45, wherein the received data representation includes an activation module that is configured to be executed by the client device and adapted to register the live object with the routing network.

49. The apparatus of claim 48, wherein the activation module is configured to determine a node type for handling registration and to cause the client device to register with the node of the node type.

50. The apparatus of claim 48, wherein the activation module determines a message category of the data representation and causes the client device to register with the node having a node type corresponding to the message category.

51. An article of manufacture including a computer-readable storage medium having instructions stored thereon, execution of which by a computing device causes the computing device to perform operations comprising:

providing, using a processing device of an input source, a data representation to a client device, different from the input source, coupled to a routing network, wherein the data representation includes at least one live object that is recognizable by the client device, and that causes the client device to respond to the live object by determining an object ID of the live object to register for updates of the live object with the routing network, such that registering the client device with the routing network provides client connection information to the routing network; and

sending, using the processing device of the input source, an update message to the routing network, wherein the update message identifies the live object and contains update data for updating a property of the live object,

wherein a gateway device at the routing network is configured to identify a category of the update message based on the input source, to determine a node type to which the identified category maps, and to route the update message to a node of the node type at the routing network,

wherein the node is configured to identify the client device as a registered device and to route the update message to the client device, and

wherein the client device is configured to process the update message upon receipt to update the property of the live object at the client device.

52. The article of manufacture of claim 51, wherein the live object of the data representation causes the client device to register with a client proxy of the routing network.

53. The article of manufacture of claim 51, wherein the live object of the data representation causes the client device to register with the node of the routing network.

54. The article of manufacture of claim 51, wherein the received data representation includes an activation module executed by the client device and adapted to register the live object with the routing network.

55. The article of manufacture of claim 54, wherein the activation module determines a node type that handles registration and causes the client device to register with the node of the node type.

56. The article of manufacture of claim 54, wherein the activation module determines a message category of the data representation and causes the client device to register with the node having a node type corresponding to the message category.

57. A non-transitory computer readable storage medium having instructions stored thereon, the instructions comprising:

instructions for providing a data representation to a client device coupled to a routing network, wherein the data representation includes at least one live object recognizable by the client device, and wherein the client device is configured to respond to the live object of the data representation by determining an object ID of the live object to register for updates of the live object with the routing network, such that registering the client device with the routing network provides client connection information to the routing network; and

instructions for providing, using a processing device of an input source, different from the client device, an update message to the routing network, wherein the update message identifies the live object and contains update data for updating a property of the live object,

wherein a gateway device at the routing network is configured to identify a category of the update message based on the input source, to determine a node type to which the identified category maps, and to route the update message to a node of the node type at the routing network,

wherein the node is configured to identify the client device as a registered device and to route the update message containing the update data to the client device, and

Atty. Dkt. No. 2222.775000E

wherein the client device is configured to process the update message upon receipt to update the property of the live object at the client device.

58. The non-transitory computer readable storage medium of claim 57, wherein the live object of the data representation causes the client device to register with a client proxy of the routing network.

59. The non-transitory computer readable storage medium of claim 57, wherein the live object of the data representation causes the client device to register with the node of the routing network.

60. The non-transitory computer readable storage medium of claim 57, wherein the data representation includes an activation module that is executed by the client device and that is adapted to register the live object with the routing network.

61. The non-transitory computer readable storage medium of claim 60, wherein the activation module determines a node type for handling registration and causes the client device to register with the node of the node type.

62. The non-transitory computer readable storage medium of claim 60, wherein the activation module determines a message category of the data representation and causes the client device to register with the node having the node type corresponding to the message category.

63. A method comprising:

providing a live object to a client device;

sending, using a processing device of an input source, different from the client device, an update message to a routing network identifying the live object and containing update data to update the live object at the client device,

wherein in response to determining that the client device is registered to receive update data for the live object, causing a category of the update message to be

Atty. Dkt. No. 2222.775000E

identified based on the input source, a node type associated with the category be determined, and the update message be routed from the gateway device to a node having the node type at the routing network, and causing the node to route the update message to the client device.

64. An apparatus comprising:

an input source arranged to provide a live object to a client device, different from the input source; and

the input source arranged to provide an update message to a routing network identifying the live object and containing update data for updating the live object,

wherein the routing network is configured to identify a category of the update message based on the input source, determine a node type associated with the category, and to route the update message from a gateway device at the routing network to a node having the node type at the routing network, and wherein the node is configured to route the update message to the client device if the client device is registered to receive the update data for the live object.

65. (Cancelled)

66. A non-transitory computer readable storage medium having instructions stored thereon, the instructions comprising:

instructions for providing a live object to a client device; and

instructions for providing an update message from an input source, different from the client device, to a routing network identifying the live object and containing update data for updating the live object at the client device,

wherein a category of the update message is identified based on the input source, a node type associated with the category is determined, and the update message is routed from a gateway device at the routing network to a node having the node type at the routing network, wherein the update message is routed from the node to the client device in response to a determination that the client device is registered to receive the update data for the live object.

67. The routing network of claim 33, wherein the node type is configured to identify the node that receives the update message from the gateway device.

68. The method of claim 26, wherein the property of the live object has a direct effect on a visual representation of the live object in a data representation, has an effect on an internal aspect of the live object and has no effect on the visual representation of the live object in the data representation, or has a direct effect on one aspect of the visual representation of the live object in the data representation and has no effect on other aspect of the visual representation of the live object in the data representation.