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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
20 March 2003 (20.03.2003)

PCT

(10) International Publication Number
WO 03/024007 A1

(51) International Patent Classification[7]: H04J 1/16, 3/14

(21) International Application Number: PCT/US02/28829

(22) International Filing Date:
10 September 2002 (10.09.2002)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/323,126 10 September 2001 (10.09.2001) US
60/322,899 10 September 2001 (10.09.2001) US

(71) Applicant *(for all designated States except US)*: **CENUS TECHNOLOGIES, INC.** [US/US]; 5550 Scotts Valley Drive, 2nd Floor, Scotts Valley, CA 95066 (US).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: **GARCIA-LUNA-ACEVES, J., J.** [MX/US]; 82 Lakewood Circle, San Mateo, CA 94402 (US). **PARSA, Michael** [/]; *.

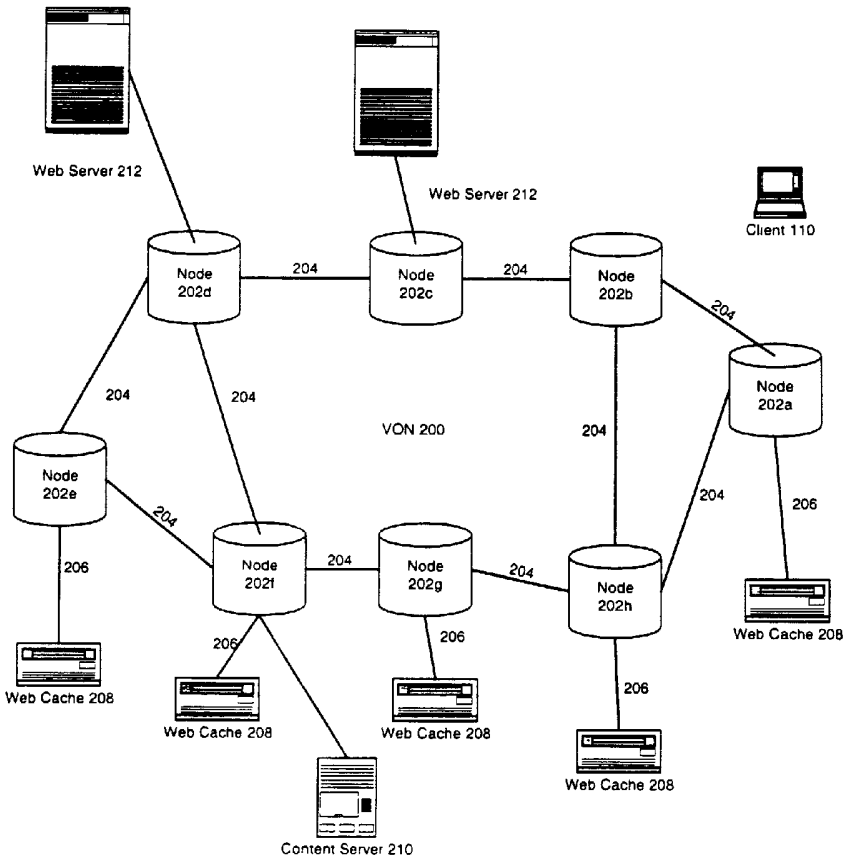
(74) Agents: **FAHMI, Tarek, N.** et al.; Blakely, Sokoloff, Taylor & Zafman LLP, 12400 Wilshire Boulevard, 7th floor, Los Angeles, CA 90025 (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, OM, PH, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZM, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, SK,

[Continued on next page]

(54) Title: SYSTEM AND METHOD FOR INFORMATION OBJECT ROUTING IN COMPUTER NETWORKS



(57) Abstract: A virtual overlay network (VON) (200) over an internetwork is dynamically maintained through inter-node communications (204) that include messages adapted to establish node neighbors (202A --- 202H), the messages being passed over tunnels within the internetwork. Node neighbors may be established according to connectivity constraints of a node, for example a limit on the maximum number of permissible neighbors and/or a requirement for a communication path to a core node. Often, node neighbors are selected according, at least in part, to link cost metrics between nodes. The selections may be periodically reevaluated and/or updated and the neighbor choosing process involves executing a chosen neighbor heuristic that connects nodes that are close to one another in the underlying internetwork and may apply where the topology is flat or hierarchical.

WO 03/024007 A1

TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:

— *with international search report*

— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

SYSTEM AND METHOD FOR INFORMATION OBJECT ROUTING IN COMPUTER NETWORKS

RELATED APPLICATIONS

[0001] The present application is related to and hereby claims the priority benefit of the following commonly-owned and co-pending U.S. Provisional Patent Applications:

(1) Application No. 60/323,126, entitled "SYSTEM AND METHOD FOR DIRECTING CLIENTS TO OPTIMAL SERVERS IN COMPUTER NETWORKS" filed September 10, 2001, by J.J. Garcia-Luna-Aceves and Srinivas Vutukury; and

(2) Application No. 60/322,899, entitled "SYSTEM AND METHOD FOR INFORMATION OBJECT ROUTING IN COMPUTER NETWORKS", filed September 10, 2001, by Jyoti Raju, J.J. Garcia-Luna-Aceves and Bradley R. Smith.

(3) Application No. 60/324,795, entitled "METHOD FOR THE DYNAMIC CONTROL OF VIRTUAL OVERLAY NETWORKS", filed September 21, 2001, by Mike Parsa and J.J. Garcia-Luna-Aceves.

[0002] The present application is also a continuation in part of commonly owned and co-pending U.S. Patent Application 09/810,148, entitled "SYSTEM AND METHOD FOR DISCOVERING INFORMATION OBJECTS AND INFORMATION OBJECT REPOSITORIES IN COMPUTER NETWORKS", filed March 15, 2001, by J.J. Garcia-Luna-Aceves.

FIELD OF THE INVENTION

[0003] The present invention relates to a system and method for establishing and maintaining a virtual overlay network within a computer network or network of networks.

BACKGROUND

[0004] An internetwork is a collection of computer networks interconnected by nodes, wherein each such node may be a general-purpose computer or a specialized device, such as a router. As such, an internetwork is often called a network of networks. The purpose of building an internetwork is to provide information services to end nodes, wherein each end node may be a general-purpose computer or a specialized device, such as a camera or a display. The Internet is an internetwork in which information is organized into packets to be distributed on a store-and-

forward manner from source to destination end nodes, and in which routers and end nodes use the Internet Protocol (IP) to communicate such packets.

[0005] An overlay network is a virtual network interconnection between routers using the IP protocol. The IP in IP encapsulation, also known as IP tunneling, has long been used to connect parts of the Internet that have disjoint capabilities (W. Simpson, "IP in IP Tunneling." Internet Engineering Task Force (IETF), Request for Comments (RFC) 1853, October 1995). The encapsulation wraps the original IP packet inside another IP header. There could be potentially other headers between the outer and inner IP headers to control security or other parameters of the tunnel. The source and destination fields in the outer IP header identify the "endpoints" of tunnel, while the fields in the inner header identify the original sender and receiver of the packet.

[0006] Overlay networks can provide more robust connectivity or additional network services, such as Virtual Private Networks (VPN) or wide-area stream broadcast/multicast without any support from the underlying network. Another benefit consists of the ability to deploy sophisticated services without the network service provider's cooperation or involvement. For example, the Multicast Backbone (MBone) or the IPv6 Backbone (6Bone) are each implemented as overlay networks that enable multicast and IPv6 functionality, respectively. The overlay networks are, however, manually configured and established.

[0007] One of the earliest examples of dynamic overlay construction was flood-d (K. Obraczka et al. "A Tool for Massively Replicating Internet Archives: Design, Implementation, and Experience," Proceedings of the 16th International Conference on Distributed Computing Systems, Hong Kong, 27-30 May 1996). Obraczka et al. investigated scalable replication mechanisms for wide-area, autonomously managed services and proposed a new architecture that extends existing replication mechanisms to explicitly address scalability and autonomy. They target replications on the order of thousands to tens of thousand of weakly consistent replicas.

[0008] In recent years, there have been several proposals to construct overlay networks to perform application-level or end-system multicasting via trees on the overlay networks. This mechanism is gaining popularity for enabling multicast in the Internet. Y. Chawathe et al ("An Architecture for Internet Content Distribution as an Infrastructure Service," unpublished report) propose a new model for Internet multicast, where the multipoint delivery is viewed as an application-level service and not a network-level one. Their architecture relies on a collection of stategically placed network agents that collaboratively provide the multicast service. Agents organize themselves into an overlay network of unicast connections and build the multicast tree on top of the overlay structure.

[0009] Chu et al (Yang-hua Chu, Sanjay G. Rao and Hui Zhang, "A Case For End System Multicast," Proc. of ACM SIGMETRICS, Santa Clara,CA, June 2000, pp 1-12) developed a protocol called Narada. In Narada, end systems self-organize into an overlay structure using a distributed protocol. Narada attempts to optimize the effciency of the overlay based on end-to-end measurements. Meanwhile, Jannotti et al (John Jannotti et al, "Overcast: Reliable Multicasting with an Overlay Network", Proceedings of OSDI, San Diego, CA, October 2000) propose Overcast, which is an application-level multicasting system for reliable single-source multicast.

[0010] The X-Bone (Joe Touch, "Dynamic Internet Overlay Deployment and Management Using the X-Bone," Proc. IEEE ICNP, Osaka, Japan, 2000) is a software system that configures overlay networks. A user, via a web-based GUI or API, sends a request to an Overlay Manager to deploy overlays. The Overlay Manager functions in phases to discover available resources and configure and monitor them. The resource discovery is a multicast-based expanding ring search. The X-Bone installs routes, configures interfaces, updates DNS entries, and installs IPSEC keys and is targeted more in establishing overlay networks for Virtual Private Networks (VPN).

SUMMARY OF THE INVENTION

[0011] In one embodiment, a virtual overlay network (VON) over an internetwork is dynamically maintained through inter-node communications that include messages adapted to establish node neighbors, the messages being passed over tunnels within the internetwork. Node neighbors may be established according to connectivity constraints of a node, for example a limit on the maximum number of permissible neighbors and/or a requirement for a communication path to a core node. Further, in some cases, some of the messages are used to query neighbor nodes about other neighbors thereof.

[0012] Node neighbors may be segregated into different types, for example candidate neighbors and chosen neighbors. Often, node neighbors are selected according, at least in part, to link cost metrics between nodes. These link cost metrics may include one or more of link latency, available bandwidth, and packet loss rate. Link latency may be estimated based on packet round trip time, while available bandwidth may be estimated using a packet-pair technique and/or according to transfers of blocks of data.

[0013] One or more of the inter-node messages may include redirection messages. In addition, directory requests based on a domain name system for the internetwork may be passed. In such cases, requesting node information may be encoded in the directory requests. Responses to the directory requests generally include identifications of nodes close to a requesting node.

Candidate neighbors and/or chosen neighbors may then be selected from the responses to the directory requests.

[0014] A further embodiment provides a virtual overlay network (VON) that includes a number of nodes interconnected through virtual links so as to establish a topology, wherein the nodes are configured to execute distributed algorithms to dynamically update the topology in response to changing conditions (e.g., communication link failures and/or node failures) in an underlying internetwork by exchanging inter-node communications that include messages used to establish node neighbors. The VON may also provide for a manual topology management console.

[0015] The distributed algorithms may include a virtual link management process at one or more of the nodes. These processes are adapted to allow each of the nodes to evaluate choices for node neighbors and to allow each of the nodes to satisfy and maintain connectivity constraints for the VON. Other processes may allow nodes to update their individual sets of chosen and/or candidate neighbors or even provide for converting candidate neighbors to chosen neighbors and vice-versa.

[0016] In some cases, one or more of the nodes may be a directory node configured to receive look up requests from one or more of the nodes and to return in response thereto one or more addresses of nearby nodes in the virtual overlay network. However, the directory node(s) need not be part of the VON. Other nodes are configured to evaluate one or more link metrics when establishing node neighbors. The link metrics may include one or more of latency, available bandwidth and loss rate. Latency may be estimated using round trip time, while available bandwidth may be estimated using either or both of a packet pair technique and/or a transfer of data.

[0017] The node neighbors may be segregated into static neighbors, chosen neighbors and candidate neighbors. In some cases, each node is restricted to a limited number of node neighbors that is less than all of the nodes of the virtual overlay network. The topology of the virtual overlay network may thus reflect that of an underlying internetwork. The messages passed between nodes within the VON may be one or more of: candidate neighbors requests; candidate neighbor acknowledgements; chosen neighbor requests; chosen neighbor acknowledgements; redirection requests; redirection acknowledgements; neighbor look up requests; and neighbor look up acknowledgements.

[0018] In yet another embodiment neighbor nodes of a virtual overlay network are automatically discovered by contacting a directory node and requesting a list of one or more nearby nodes withing the virtual overlay network; and then choosing from the list of nearby nodes at least one of nearby nodes as a chosen neighbor node. Choosing the chosen neighbor

node(s) involves evaluating link cost metrics associated with communication paths to the one or more nearby nodes, wherein the link cost metrics are functions of one or more of latency, available bandwidths, and loss rates. In one partiular case, the link cost metrics are a function of a ratio of latency to available bandwidth. Part of the process of choosing neighbor nodes involves establishing communication tunnels with the chosen neighbor node.

[0019] Generally, the chosen neighbor node is selected so as to satisfy connectivity requirements of the virtual overlay network. Moreover, the selections may be periodically reevaluated and/or updated This choosing process involves executing a chosen neighbor heuristic that connects nodes that are close to one another in an underlying internetwork and may apply where the topology is flat or hierarchical.

[0020] In some cases, contacting the directory node involves transmitting a domain name system look up request. The request should include an identification of a requesting node encoded therein, for example an IP address. The list of one or more nearby nodes may then be transmitted as part of a domain name system response to the look up request.

[0021] Preferably, leaving the virtual overlay network involves affirmatively notifying neighbor nodes of the departure. This may include redirecting the neighbor nodes to other nodes.

BRIEF DESCRIPTION OF THE DRAWINGS

[0022] The present invention is illustrated by way of example, and not limitation, in the figures of the accompanying drawings in which like reference numerals refer to similar elements and in which:

[0023] **Figure 1** illustrates an example of an internetwork in which the methods and systems of the present invention may operate.

[0024] **Figure 2** illustrates a virtual overlay network of Web routers configured in accordance with an embodiment of the present invention.

[0025] **Figure 3** illustrates functional componets of a Web router in accordance with an embodiment of the present invention.

[0026] **Figure 4** illustrates an example of a node joinging a virtual overlay network in accordance with the methods of the preent invention.

DETAILED DESCRIPTION

Introduction

[0027] The present invention provides an architecture and method to dynamically build and maintain a virtual overlay network over the Internet or other internetwork. The virtual overlay network (VON) consists of portions of the Internet used to interconnect a set of servers used to deliver a family of services to clients of the VON. Accordingly, the links of the VON are tunnels in the Internet or other peering mechanisms between Web routers, and the nodes in the VON are the web-routers.

[0028] The VON is established by connecting Web routers over tunnels or other means to form a service or content distribution cloud. The connection between two Web routers constitutes a virtual link. Web routers communicate and perform their content routing computations over the virtual links of the VON.

[0029] The term Web router refers to an embodiment of a computer system configured to map or associate the identifier (e.g., a uniform resource locator or URL) of a requested information object or service with the address (e.g., a network or IP address) of the optimal information object repository that can deliver the requested information object or service to another server (i.e., a requesting server that will ultimately respond to an original client request). The performance metrics used by Web routers to choose the information object repository(ies) (e.g., servers, caches, etc.) that should provide the requested information object(s) may include any of several performance parameters, such as an average delay from one information object repository to another along the network(s) (i.e., network delays), average processing delays at an information object repository, reliability of a path from one information object repository to another, available bandwidth in said path, and loads on the various information object repositories.

[0030] To simplify its description, the Web router is described and treated herein as a separate entity. However, the functionality of a Web router can be co-located and/or implemented as part of a content server, a Web server, a Web cache, a router or as a separate entity. Web routers may be implemented in software to be executed by general-purpose (or special-purpose) computer systems, or some or all of a Web router's functionality may be implemented in hardware. Such details are not critical to the present invention.

[0031] Web routers make use of a communication protocol to pass messages between one another over a reliable transmission protocol. These messages include information that allows the Web routers to dynamically update the mappings of identifiers of information objects and services to information object repository addresses based on one or more of the specified performance metrics. As indicated, these mappings may, and preferably do, specify optimal associations of information object identifiers to information object repository addresses. Also, the messages passed using the inter-Web router communication protocol may further report an

associated address of a Web router co-located with an information object repository (optimal or otherwise) that contains the information object that is the subject of the message. Throughout this description, a Web router is referred to a node of the VON, and virtual links are called links of the VON.

[0032] In the following description, numerous specific details are set forth in order to provide a thorough understanding of the present invention. However, it will be evident to those of ordinary skill in the art that some of these specific details need not be used to practice the present invention and/or that equivalents thereof may be used. In other cases, well-known structures and components have not been shown in detail to avoid unnecessarily obscuring the present invention. Thus, although discussed with reference to certain illustrated embodiments, upon review of this specification, those of ordinary skill in the art will recognize that the present system and methods may find application in a variety of systems and the illustrated embodiments should be regarded as exemplary only and should not be deemed to be limiting in scope.

[0033] Some portions of the description that follow are presented in terms of algorithms and symbolic representations of operations on data within a computer memory (e.g., in pseudocode). These algorithmic descriptions and representations are the means used by those skilled in the computer science arts to most effectively convey the substance of their work to others skilled in the art. An algorithm is here, and generally, conceived to be a self-consistent sequence of steps 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. It 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. 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, it will be appreciated that throughout the description of the present invention, use of terms such as "processing", "computing", "calculating", "determining", "displaying" 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.

Internetwork Basics

[0034] Before describing further details of the present invention, an overview of an internetwork environment in which Web routers (i.e., nodes) may be deployed will be presented. For purposes of the present invention, a topology of nodes is defined such that a given node has only a subset of all the nodes in a given system as its neighbor nodes. Nodes that are neighbors connect to each other using tunnels across one or more communication links. The topology of nodes and tunnels between them constitutes the VON. That is, the VON interconnects the nodes and overlays an internetwork made up of conventional IP routers, information object repositories, and other network components. In the VON, each tunnel link is assigned a value that describes the performance metric across the tunnel.

[0035] **Figure 1** illustrates an internetwork 100 and the methods and systems described herein enable the direction of clients and/or servers, etc. to either information objects or the caches and servers storing information objects distributed over computer networks such as internetwork 100. One example of an internetwork 100 is the Internet. Other examples include enterprise networks, local area networks, wide area networks, metropolitan area networks and networks of such networks. In the case where internetwork 100 is the Internet, clients 110 will generally access content located at remote servers 170 through a series of networks operated by different providers. For example, clients 110 may have accounts with local ISPs 120 that enable the clients to connect to the Internet using conventional dial-up connections or one of a variety of high-speed connections (e.g., DSL connections, cable connections, hybrids involving satellite and dial-up connections, etc.). ISPs 120, in turn, may provide direct connections to the Internet or, as shown, may rely on other service providers 130, 140, 150 to provide connections through to a set of high-speed connections between computer resources known as a backbone 160. Connecting to a host (e.g., server 170) may thus involve connecting through networks operated by a variety of service providers.

[0036] **Figure 2** illustrates a VON 200 of nodes 202a – 202h defined on top of the physical topology of an internetwork, such as the Internet, consisting of routers interconnected via point-to-point links or networks. The virtual network of nodes includes point-to-point links 204 configured between the nodes, and the links 206 configured between a node 202 and one or more Web caches 208 and content servers 210. Such links 204, 206 can be implemented using tunnels (e.g., Internet protocol (IP) tunnels) between nodes 202 and between nodes 202 and Web caches 208. As discussed above, messages can be exchanged between the nodes 202 via the tunnels. As shown in the figure, a client 110 is not necessarily part of the VON.

[0037] As indicated above, to reduce communication and processing overhead in nodes, a topology of nodes is defined such that a given node has as its neighbor nodes a subset of all the

nodes in the system (where the term system refers to all or a portion of the VON 200 discussed above). A node may thus be configured with its set of neighbor nodes. Such a configuration may be expressed as a table of neighbor nodes that is defined by a network service provider and/or is dynamically updated. In some cases, nodes may dynamically select the set of neighbor nodes with which they should communicate out of all of the nodes in the system.

[0038] A node preferably communicates with its neighbor nodes using the Web Information Locator by Distance (WILD) protocol. The WILD protocol is disclosed in commonly owned U.S. Provisional Application No. 60/200,401, entitled "System and Method for Discovering Optimum Information Object Repositories in Computer Networks (WILD Protocol), filed April 28, 2000 by J.J. Garcia-Luna-Aceves and Bradley R. Smith, now replaced by commonly owned and co-pending U.S. Patent Application 09/810,148, entitled "System and Method for Discovering Information Objects and Information Object Repositories in Computer Networks", filed March 15, 2001, by J.J. Garcia-Luna-Aceves, the complete disclosures of which are hereby incorporated by reference. The WILD communication protocol provides for the exchange of one or more inter-node messages via the tunnels. These messages carry the mappings specifying the association between optimal information object repositories and information objects according to the specified metrics. When these mappings change due to changes in the topology of the Internet, the messages carry updated distance information (e.g., as computed according to the performance metrics) in the maps. Thus, using WILD each node 202 implements a distributed algorithm and executes a communication protocol with which it determines:

(1) the address of all the other nodes participating in the same VON; and

(2) the optimum distance to each node (i.e., the associated information object repository) in the VON and the neighbor node that offers such a distance.

[0039] The nodes 202 employ special rules when updating their local maps in response to received messages. The WILD protocol (or simply WILD) running at each node then constructs tables (which are stored locally in memory or other computer-readable media) containing the optimal mapping information. Each node uses the tables computed by WILD for directing a requestor to an optimal information object repository. Further details regarding the manner in which the maps and tables are generated and shared among nodes may be found in co-pending Application No. 60/323,126, entitled "System and Method for Directing Clients to Optimal Servers in Computer Networks" filed September 10, 2001, the complete disclosure of which is hereby incorporated by reference. Briefly, however, one or more tables are constructed at each nodes and these tables contain client-to-server distance information. The tables are stored in a computer-readable medium accessible by the corresponding node and are

updated in response to revised client-to-node distance information. Such revised client-to-node distance information may be included in the inter-node communication messages and is preferably determined, at least in part, from internetwork connectivity information received through an exchange of messages according to an inter-domain routing protocol. Furthermore, the tables may be updated in response to revised server load information and the updated table information transmitted to one or more of the nodes using one or more inter-node communication messages.

Web Routers

[0040] **Figure 3** illustrates the interaction between various functional components that make up a Web router node 300. The method implemented in nodes to determine an optimum server for a given information object or service is referred to as "URL Routing" and an information object identifier or service identifier is referred to as a "URL". The node maps each URL provided by a requestor to the address of an information object repository that can optimally provide the associated information object. This mapping of URLs to addresses is accomplished by the collaboration among nodes through WILD. Accordingly, the node contacted by the requestor can return the required addresses immediately after processing the request.

[0041] In each node 300, a WILD protocol module 302 uses mappings of clients (or, more generally, requestors)-to-nodes received from other nodes along with load information regarding any local servers 306 to produce a mapping table that lists the address of the server (i.e., information object repository) that is best suited to service requests from identified clients. The WILD protocol module 302 further uses an ALP module 304 for obtaining VON connectivity and node reachability information. It also uses ALP messages to encapsulate and deliver its own WILD messages to neighbor nodes. For further information regarding the Adaptive Link-State Protocol or ALP see, J.J. Garcia-Luna-Aceves and M. Spohn, "Scalable Link-State Internet Routing," Proc. IEEE International Conference on Network Protocols (ICNP 98), Austin Texas, October 14-16, 1998, pp. 52-61, incorporated herein by reference. Other routing protocols can be used instead of ALP, provided that the protocol does not create permanent or long-term loops after link cost increases or resource failures. The ALP module 304 reports distances to the WILD module 302, so that the WILD module 302 knows about unreachable nodes while determining the validity of different maps. The inter-communication between WILD module 302 and ALP module 304 may take place through the use of shared memory 314.

[0042] ALP module 304 notifies the WILD protocol module 302 as distances to various nodes and information object repositories change. When a node 300 receives a WILD message (i.e., a

WILD message that was received encapsulated in an ALP message), it first validates the message contents, and then updates its tables based on distance. The end result is that the WILD tables at each node 300 converge to the optimum mappings.

[0043] The WILD protocol module 302 obtains load information from its local information object repository 306. This load information is subsequently used by the WILD protocol module 302 for generating the mappings discussed above. More particularly, the WILD module 302 periodically polls the local information object repository 306 to obtain the load information. If the information object repository 306 fails, a load of infinity is reported to the WILD module 302.

[0044] The WILD protocol module 304 also interacts with a Web router query (WRQ) module 308 in directing requestors to optimum information object repositories. The node 300 responds to queries from the WRQ module 308 for the nearest server, Web cache, redirector, DNS server or another node. A node may be contacted through the WRQ module 308 by a client or a server (or other information object repository) seeking to discover an optimum information object repository for one or more information objects distributed over the computer network. This scheme makes use of a neighbor protocol module 310.

VON Topology

[0045] Returning now to a discussion of the VON 200, the VON topology can be established manually via a set of command tools or Web GUIs. That is, the present invention does not preclude a manual topology set-up. However, it is more useful to allow for dynamic configuration of the VON, especially as the size of the topology grows and the dynamic nature of the distribution cloud is taken into consideration. Additionally, static (i.e., manual) and dynamic topology control can be mixed and used together for maximum flexibility and control.

[0046] As will be descibed in greater detail below, this process of dynamic VON configuration provides that as a new node is added to the network it automatically discovers its neighbors and connects to the VON. To accommodate this process, the new node is provided with a configuration file containing the minimum information necessary to become part of a VON. Using the information in its configuration file, the new node contacts a so-called "directory" server to learn about nearby nodes (called candidate neighbors below). Then, where appropriate (i.e., depending on the specified performance metrics), the new node establishes tunnels to one or more of these nearby nodes and checks to see whether, if by doing so, it has established connectivity to the VON. If so, nothing further is required (other than periodic maintenance of its links to other nodes in response to updates and other inter-node messages, etc.). If not, the new node augments is connectivity until it is connected to the VON. Thus, the

present invention provides effective distributed algorithms to automatically discover and maintain a topology of the VON that is close to optimum. The topology of the VON adapts to changing network conditions, such as node or link failures or traffic provisions, to maintain optimality. The architecture of the VON facilitates adding and removing nodes from the VON, and adapting the VON topology accordingly.

[0047] From the above, it should be apparent that the following are the main components of the architecture of the VON:

(1) A topology management console to access and configure nodes. Such a console is used for manual configuration of the VON (e.g., by a network operator).

(2) A tunnel management process (i.e., a software routine or daemon hereafter called "tund") and its configuration file on each node. The process tund is responsible for building and monitoring virtual links (i.e., tunnels) to other nodes.

(3) A WILD-enabled DNS (WDNS) and its configuration file on directory nodes. A directory node is defined as any node that runs WDNS and need not be part of the VON, depending on the VON configuration and the running of tund on the node.

[0048] Once connected to the VON, the general steps to be executed at a non-directory node to maintain VON connectivity are:

(1) Use WDNS to discover candidate neighbors. That is, make use of the directory nodes to learn about nearby nodes that may offer new paths and may be sutiable for inclusion in a set of chosen neighbors.

(2) Query neighbors about their neighbors. This way nodes learn about other nodes in a network.

(3) Evaluate choices for neighbors locally. That is, each node should use information gathered from other nodes to evaluate whether or not to choose nearby nodes as neighbors. These choices will depend on the cost of the link metrics (see below) associated with connections to the candidate nodes.

(4) Satisfy and maintain connectivity constraints. As discussed below, each node needs to maintain at least one path to a VON backbone to ensure connectivity to the VON as a whole.

(5) Update a set of chosen neighbors. Based on the evaluations and connectivity constraints, each node can establish a set of chosen neighbor

nodes with which to communicate (e.g., using WILD messages to pass maps of optimum servers for client addresses or address ranges).

The above steps achieve an overlay network that is robust and adaptive to failures or changes in internetwork (i.e., Internet) conditions. It also uses the underlying physical network in an efficient manner.

[0049] The above process further demonstrates that there are two general considerations to be taken into account in constructing the topology of a VON: The quality of the links and the connectivity characteristics of the nodes. Links in the VON can be characterized by various metrics, such as latency, available bandwidth, loss rate, and others. In one embodiment of the present invention, only latency in the form of round-trip time (RTT) is considered. In another embodiment of the present invention, forward delay latency and the available bandwidth are the metrics used to characterize links in the VON.

[0050] Where used, metrics such as link latency and bandwidth need to be estimated or measured periodically. For example, a sender can measure round-trip latency via a probe packet that is reflected back at the destination. Ideally, the evaluations of such metrics take into account the asymmetry of these qualities on the virtual links and provide the values bth for the forward direction and the reverse direction. Moreover, the frequency of any probe package transmissions has to be such that the traffic overhead is minimal yet it must also provide for (i.e., allow for the capture information regarding) adaptability in the VON topology. For example, probes may be sent on the order of minutes (here, it is expected that the topology changes in the VON are rather infrequent, on the order of hours, because the underlying physical links and network routing provide adaptivity to near-term changes in network conditions).

[0051] The characteristics of interest in link quality and their estimators are thus:

(1) latency (e.g., as measured by probe RTTs); and

(2) available bandwidth (e.g., as measured using a packet-pair technique or a transfer of a block of (dummy) data).

The shortcoming of both techniques, however, is that they cannot isolate the latency and available bandwidth in the forward direction from the reverse one. The latency and bandwidth values for each link l can, nevertheless, be combined into a single cost value for the link, e.g., $cost_l = latency_l / bandwidth_l$. Then, this cost can be used in determining the links to be used in the topology of the VON.

[0052] In terms of connectivity, the VON topology should use the underlying network effciently. For practical and performance considerations, a node can have up to a maximum of k_{max} neighbors. Because each node communicates with all its neighbors to maintain several

types of routing information, allowing a node to have more than k_{max} neighbors may adversely affect and degrade the performance of the routing protocols running at the node.

[0053] For the sake of fault-tolerance, it is desirable for the topology of the VON to be k-connected, where $k \geq 2$. In a k-connected graph, the removal of k - 1 nodes and their incident links does not disconnect the graph. Note that the minimum degree of a graph needs to be "k" as a necessary condition for graph to be k-connected graph. As the value of k increases, the connectivity of the graph increases. This tends to reduce the diameter of the graph, which is also good. That is, the maximum number of (virtual) hops between nodes is decreased, thereby reducing the latencies between any two nodes of the VON. The increased connectivity of the VON topology also provides more choices for information dissemination over the VON.

[0028] To simplify the discussion of the present invention, the construction of a 2-connected topology (i.e., $k = 2$) is described. The method can be generalized to construct a k-connected graph in a distributed manner. Further, for the VON to scale with a large number of nodes its topology could include a hierarchical structure. For example, the present invention could be extended to be a two-level hierarchy, where the levels reflect inter-AS and intra-AS node communication. In the terminology of the Internet, a routing domain is called an autonomous system (AS). An example of an inter-domain routing protocol (suitable for inter-AS node communication) is the Border Gateway Protocol (BGP). An example of an intra-domain routing protocol (sutiable for intra-AS node communication) is the Open Shortest Path First (OSPF) protocol. Extending the approach described herein to a multi-level hierarchy is also possible. However, for simplicity the focus of the remaining description is on a flat topology.

[0054] The closest neighbor heuristic used in establishing the VON topology connects nodes that are close in the underlying cloud. That is, the topological structure of the VON reflects that of the underlying network. Therefore, nodes that are far according to the underlying cloud will not be neighbors. Also, in the discussion below, selecting the least-cost edges locally is a heuristic motivated by existing distributed minimum-spanning tree (MST) algorithms. Minimizing the cost of the VON corresponds to minimizing the utilization fn the network resources.

Tund Neighbor and Query Messages

[0055] As alluded to above, a node can have three types of neighbors: A static neighbor is one that is established via a management console and is mandatory. A candidate neighbor is one that a node knows about but is not established as being adjacent in the VON topology. A chosen neighbor is one that is considered an actual neighbor in the VON topology. The same terms can be applied to the (tunnel) links connecting a node to its neighbors according to the neighbor type.

[0056] The type of neighbor is determined and updated in response to the sending and receiving of the following messages, whether between nodes or between management consoles and nodes (i.e., these messges are used in the process of setting up and maintaining the VON):

(1) Static-neighbor: request for a static neighbor.

(2) Static-ack: acknowledgment for static-neighbor request.

(3) Candidate-neighbor: request for a candidate neighbor.

(4) Candidate-ack: acknowledgment for candidate-neighbor request.

(5) Redirect-candidate: redirect a request for candidate.

(6) Chosen-neighbor: request to change a candidate to a chosen neighbor.

(7) Chosen-ack: acknowledgment for chosen-neighbor request.

(8) Redirect-chosen: redirect a request for chosen neighbor.

(9) Redirect-neighbor: request to redirect a neighbor (or remove neighbor).

(10) Redirect-ack: acknowledgment to redirect-neighbor.

A static-neighbor message is compulsory in that a node receiving such a message must become a static neighbor of an identified node. However, candidate-neighbor and chosen-neighbor messages can be redirected by a node receiving such a message. In this way, a node can limit its connectivity within the VON. The redirect responses usually carry alternatives addresses.

[0057] Nodes may also send queries to other nodes to learn about their connectivity. The following is a query message and response between neighbor nodes to communicate such information between one another.

(1) Lookup-neighbor: query for neighbors of a neighbor

(2) Lookup-ack: response to lookup-neighbor request

[0058] It is the establishment of neighbors that provides the connectivity between nodes and maintians the architecture of the VON. As network conditions change, a given node may need to revise its connectivity to other nodes in order to adapt to these changes. The following pseudo-code specifies the basic operations (SetCandidate, SetChosen, UnsetChosen, and RedirectNeighbor) performed at each node to initiate a change in the status of its neighbors. Each node maintains three sets: staticset, candidateset, and chosenset, corresponding to type of neighbors it has. The procedures SetCandidate and SetChosen are executed to make a neighbor candidate and chosen, respectively. The procedure UnsetChosen changes a chosen neighbor to a candidate neighbor. The procedure RedirectNeighbor redirects existing neighbors to other nodes.

SetCandidate (*x; candidateset; chosenset*)

1 for 1 to *max_retry*

2 do send candidate-neighbor request to *x*

```
3                   wait for response rsp from x or time-out
4                   if time-out
5                           then continue
6 if rsp = candidate-ack
7           then initialize tunnel link to x for metric evaluation
8                   candidateset ← candidateset + x
9           else if rsp = redirect-candidate
10            then remove x as a potential candidate for rednegcache_time
11                  rset ← {addresses in response rsp}
12                  for each y ∈ rset          ▶ done sequentially or
concurrently
13                          do SetCandidate (y; candidateset; chosenset)
14                                  evaluate link metric to y
```

```
SetChosen (x; candidateset; chosenset)
1 for 1 to max_retry
2           do send chosen-neighbor request to x
3                   wait for response rsp from x or time-out
4                   if time-out
5                           then continue
6 candidateset ← candidateset - x
7 if rsp = redirect-chosen
8           then remove x as a potential candidate for rednegcache_time
9                   rset ← {addresses in response}
10                  for each x ∈ rset     ▶ done sequentially or
concurrently
11                          do SetCandidate (x; candidateset; chosenset)
12                                  evaluate link metric to x
13                                  rset ← rset - x
14          else if rsp = chosen-ack
15                  then                ▶ set candidate link to chosen link
16                          chosenset ← chosenset + x
17                          candidateset ← candidateset - x
18          else                        ▶ failed to receive a response
19                  remove and cleanup the tunnel link to x
```

```
UnsetChosen (x; candidateset; chosenset)
    1 for 1 to max_retry
    2       do send candidate-neighbor request to x
    3           wait for response rsp from x or time-out
    4           if time-out
    5               then continue
    6 chosenset ← chosenset - x
    7 if rsp = candidate-ack
    8       then candidateset ← candidateset + x
    9       else                    ▶ failed to receive a response
    10      remove and cleanup the tunnel link to x
```

```
RedirectNeighbor (x; candidateset; chosenset)
    1 ▶ x can be candidate or chosen
    2 for 1 to max_retry
    3       do send redirect-neighbor request to x
    4           wait for response rsp from x or time-out
    5           if time-out
    6               then continue
    7 if rsp = redirect-ack
    8       then if x ∈ candidateset
    9           then candidateset ← candidateset - x
    10          else if x ∈ chosenset
    11              then chosenset ← chosenset - x
    12                  remove and cleanup the tunnel link to
    x
```

[0059] The following pseudo-code describes the steps taken by a node in response to receiving one of the above requests. The requests initiate a change to a neighbor's type. The process first determines whether the node has already reached its maximum permitted number of connections. If so, the request is redirected. If not, the requesting node is added as the appropriate type (in the case of a chosen or candidate request) or removed from consideration (in the case of a redirect request).

```
ReceiveRequest (req)
    1 x ← req.sender
    2 if k_max is reached
    3       then                    ▶ pick alternates from any of
neighbor sets
    4           altset = {alternate addresses}
    5           switch
    6               case req = candidate-neighbor :
    7                   send redirect-candidate with altset to x
    8               case req = chosen-neighbor :
    9                   send redirect-chosen with altset to x
    10      else switch
    11          case req = candidate-neighbor :
    12              if x ∉ candidateset
    13                  then send candidate-ack to x
    14                  candidateset ← candidateset + x
    15          case req = chosen-neighbor :
    16              if x ∉ chosenset
    17                  then send chosen-ack to x
    18                  chosenset ← chosenset + x
    19          case req = redirect-neighbor :
    20           remove x as a potential candidate for
rednegcache_time
    21              if x ∈ chosenset
    22                  then send redirect-ack to x
    23                  chosenset ← chosenset - x
    24              else if x ∈ candidateset
    25                  then send redirect-ack to x
    26                  candidateset ← candidateset - x
```

Joining and Leaving a Topology

[0060] To allow for the dynamic creation and maintenance of the VON, a mechanism is needed to add new nodes into a topology and to remove them as needed. When a new node is installed in the field, it is installed with a configuration file containing the minimum information needed to become part of a VON. However, secure remote access (i.e., ssh) via a

management console can be used to update the configuration file if needed. Removing a node (for maintenance, etc.) from a VON ends its participation in the topology control process. The automatic bootstrapping of a new node builds on the DNS (the Internet domain name system, see P. Mockapetris, "Domain Names – Concepts and Facilities", IETF RFC 1034, November 1987, and P. Mockapetris, "Domain Names – Implementation and Specification", IETF RFC 1035, November 1987, each of which are incorporated herein by reference) mechanism and as such benefits from the fault-tolerance and robustness of the DNS.

[0061] In a VON with only a few nodes and no distance measures between nodes, the topology of the VON can be a full mesh. A full mesh though is not scalable to a large number of nodes and should be applied only in a small distribution cloud. In a full mesh mode, a node starts by setting up a link with one existing node already in the VON. This can be managed, for instance, via the management console manually. After the link is established, the new node can discover all other nodes by obtaining the topology information from the existing node that is already a member of the VON. The new node can then set up tunnels to all other nodes in the VON.

Joining a VON

[0062] For implementations other than the full mesh approach, the following procedure may be used. A new node with an IP address 0xaabbccdd (in hex) that needs to join the VON (with domain name d.isp.net), makes a DNS look up for aabbccdd.d.isp.net using the standard library routines for DNS access (i.e., gethostbyname). When the look up request is received by a WILD-enabled DNS (WDNS) server d.isp.net (an example of a directory server), it returns (again using the standard DNS protocols and packets) a set of nodes that are "close" to the new node. In doing so, the WDNS server uses the IP address of the new node encoded in the look up string to select the set of candidate neighbors for the new node. The IP address of the new node is encoded in the look up request because the standard DNS query does not carry the IP address of the originating client for the lookup request, but rather carries only the address of the local DNS server.

[0063] At a minimum, the size of the set of nodes returned by the WDNS server could be one; the size of the set does not imply how many neighbors the new node will have in the VON. However, returning a larger set allows for fault-tolerance in that another candidate can be contacted if one of the nodes in the set is down or unreachable due to transient failures or delays. Returning only one value is just the simplest approach.

[0064] Through the DNS mechanism, a response arrives at the new node with a set of the IP addresses of candidate neighbors. The new node evaluates the link metric for each of the candidates. Then it establishes at least k_{min} links by adding the "cheapest" links according to the

link metric(s). If the size of the candidate set from the WDNS server is less than k_{min}, the node builds up its candidate set from its existing neighbors.

[0065] For example, consider the network shown in **Figure 4**, where node A is the new node to join the VON consisting of nodes B, C, D and E. For the sake of the example, let node B be the nearest node, and the only one returned by the WDNS server in response to a DNS look up request by node A. Then, node A will establish a (tunnel) link with node B.

[0066] Once a new node establishes its links, it performs a connectivity test. The connectivity test determines whether or not the new node is connected to a "core" node (i.e., has a route, through one or more neighbors, to a core node). A core node is any node that is on the virtual backbone of the topology. The virtual backbone is a set of nodes that are monitored and maintained to be up and available 24 hours a day/7 days a week via a Network Operation Center (NOC). It could consist of only one node. If there is a set of nodes on the backbone, then it is assumed that all of them are guaranteed, via network monitoring tools (e.g., the Simple Network Management Protocol (SNMP)), to be connected to each other. Otherwise, if the cores are not connected, isolated islands of nodes can be formed around each core and the overall VON topology would be disconnected.

[0067] The core nodes to use are also determined by a DNS look up, i.e., by resolving core.d.isp.net. As a convention, the core nodes could be the WDNS nodes. At a minimum, there is one entry in the core set. If there are multiple core nodes in the response, the new node will check to see if it has a route to any of them. As long as there is a route to one core node then the new node is connected to the VON topology.

[0068] The following pseudo-code describe the actions taken by a WDNS node in the process of a DNS look up (the term WR referes to Web router or simply a node). An anchor is a node that has an EGP and IGP feeds (e.g., BGP and OSPF, respectively), and is responsible for computing the mapping of client addresses or address ranges to a best match server. A WDNS node should therefore be connected to an anchor to allow for its computation of matching address prefixes to the best nodes. A topology graph represents the local knowledge of the network nodes and their interconnections, i.e., a "map."

```
WDNS_GetClosestWR ()
    1 extract IP address r from look up string in request
    2 if WDNS anchor exists in AS of r
    3         then refer DNS request to that WDNS
    4 find closest WR x to r a la WILD (using EGP + IGP)
    5 ▶ when r is already in the topology
    6 if x = r
```

```
7           then add closest neighbor y of x to the response set
8                   x ← y
9                   add x to the response set
10 if k_min is encoded in lookup string
11          then extract k_min from lookup string
12          else set k_min ← 2
13 for 1 to k_min
14          do add closest neighbor y of x to the response set
15                  set x ← y
```

[0069] The procedure TUND_Join below is a pseudo-code representation of the process discssed above and describes the steps taken by a node to join the VON. The calls to ConnectivityTest, MinConnect , MaxConnect , and AugConnect (each presented in detail below) relate to the connectivity requirements and are described in the following paragraphs.

```
TUND_Join ()
    1 encode IP address and domain into a look up string
    2 perform DNS lookup request
    3 rset ← {addresses in DNS response}
    4 for each x ∈ rset                    ▶ done sequentially or concurrently
    5       do SetCandidate (x; candidateset; chosenset)
    6             rset ← rset - x
    7 for each x ∈ candidateset
    8       do evaluate link metric to x
    9 for 1 to min (k_min, |candidateset|)
    10      do select the best x ∈ candidateset according to link metric
    11            SetChosen (x; candidateset; chosenset)
    12 test ← ConnectivityTest ()
    13 if test = false
    14      then                           ▶ not connected to core
    15            retry TUND_Join after join_core_timeout
    16      else MinConnect ()
    17      MaxConnect ()
    18      AugConnect ()
```

Leaving a VON

[0070] For fast propagation of topology changes, it is desirable to make the departure of a node from a VON explicit. That is, the node being removed from a VON should send explicit notifications to this effect, rather than using an implicit "removal by silence" timeout. VON links are removed reliably to distinguish them from link failures, to minimize any possible disruption to the topology. A node can leave only when its departure would not partition the VON topology. The process TUND_Leave in the following pseudo-code describes the steps taken by a node leaving a VON. Esentially, all of the nodes existing chosen and candidate neighbors are redirected to other nodes.

```
TUND_Leave ()
    1 nset ← candidateset ⋃ chosenset
    2 for each x ∈ nset                    ► done sequentially or concurrently
    3   do RedirectNeighbor (x; candidateset; chosenset)
```

Failures in VON Topology

[0071] Occasinally, nodes or links in the VON will fail and the remaining nodes may need to take action to restore lost connectivity to the VON. Thus, two general tasks for a node adjacent to a link or node failure to perform after the failure are:

(1) perform a connectivity test to determine if the node is still connected to the core; and

(2) a node may acquire new neighbors to meet the minimum connectivity of a VON either to satisfy k_{min} or to augment the connectivity.

[0072] The links of a VON, or tunnels, adapt to changes in the underlying network by virtue of adaptive routing in the underlying network. In other words, the VON links may follow different paths in the underlying network over time. A VON link functions as long as the underlying network is connected between the two ends of the link. There could be transient VON link failures during the period when the underlying routing changes between the tunnel endpoints. Note that there could be race conditions in which the VON topology may seem connected but, in fact, it is not because the underlying routing has not converged.

[0073] However, in the event of network partitions in the underlying network, VON links will fail within a finite time. Any permanent or long-lived partitions in the underlying network will result in partitioning of the VON topology. If the implementation considerations allow it, a link failure indication will be sent from the routing process to the "tund" process, since the routing

process has a link failure detection mechanism. Otherwise, a "hello" protocol may be used to detect VON link failures.

[0074] When a node detects a VON link failure, it will attempt to replace the failed endpoint with a subset of the neighbors of the failed endpoint. This information is gathered from its local topology graph. This is the preferred method; nevertheless, the node could, alternatively, perform a DNS look up for aabbccee.d.isp.net, where aabbccee is the IP address of the failed endpoint. In response, the requesting node would receive a set of nearby nodes to use to establish a new link. Considering the example in **Figure 4**, if node B failed, then node A would try to establish links with nodes C and/or E, depending on the link metric(s) and its connectivity requirements. The neighbors of B are known from its routing advertisements to A, which A maintains in its topology graph.

[0075] A node may fail due to any number of reasons, such as power outages, hardware or software failures, etc. A node failure can result in a partitioning of the VON topology, even though the underlying network is connected. To avoid this partitioning, the connectivity of a VON needs to be "rich", i.e., at least k-connected. When a node detects that its neighbor has failed, it executes a ConnectivityTest. In the process, a node knows whether or not it is still connected to the core of a VON. Thereafter, the node may augment its local connectivity, so that it has no "cut" neighbors by performing the AugConnect process (see below). This procedure results in a 2-connected topology in this example.

```
ConnectivityTest ()
    1 for 1 to max_core_lookup
    2   do perform DNS lookup request for core.d.isp.net
    3         if DNS response rsp received
    4               then                          ▶ update core set
    5                     core_set ← rsp.addresses
    6         for 1 to |core_set|
    7               do select a node x ∈ core_set
    8                     for max_routeconverge_time
    9                           do test for a route to x
    10                                if no route to x
    11                                then break
    12                                else return true
    13 return false
```

Neighbor Changes and Maintenance

[0076] The minimum and maximum number of neighbors to maintain, k_{min} and k_{max} are configurable variables. In the pseudo-code examples set forth in this description, there are also "low and high water marks", k_{low} and k_{high} respectively, between these minimum and maximum values. These low and high water marks are a function of k_{min} and k_{max}, and may be used as warning markers that the minimumor maximum values are being reached.

[0077] As mentioned above, a node learns about other candidates via its existing neighbors. For example, every node can query any of its neighbors (of any type) for the neighbors of the neighbor. A node can also learn about other candidates via it chosen set of neighbors. This technique of querying neighbors can be used to implement an expanding ring search to find nodes at various hops in the topology. Note, however, that a node may not know all the chosen neighbors of its chosen neighbors if only partial topology information is used in the routing protocol (e.g., ALP).

[0078] In order to satisfy its minimum connectivity requirements, a node performs the MinConnect process (e.g., any time k_{min} criteria is not satisfied at a node). At the same time, a node should actively stay below its maximum connectivity k_{max}. As new neighbors acquire a node, therefore, the node should unload some of its existing neighbors. This procedure is described in MaxConnect pseudo-code below. Finally, when a node reaches its k_{high}, it starts sending redirections to a subset of its other neighbors.

```
MinConnect ()
    1 if k_low is reached
    2   then if candidateset = 0
    3           then                    ▶ using lookup-neighbor request
    4             repeat query a neighbor for its candidate and chosen
    neighbors
    5             until |candidateset| > k_min - |chosenset|
    6   for each x ∈ candidateset
    7           do evaluate link metric
    8   repeat select the best x ∈ candidateset according to link metric
    9           SetChosen (x; candidateset; chosenset)
    10  until k_low is not reached or candidateset = 0

MaxConnect ()
    1 ▶ use candidate and chosen neighbors
    2 while k_high is reached
```

```
3        do select the worst x ∈ chosenset according to link metric
4        UnsetChosen (x; candidateset; chosenset)
```

[0079] Each node performs the process AugConnect to establish a topology robust to failures. The procedure consists of a k_{max} application of Dijkstra's algorithm. Therefore, it would be straightforward to achieve an $O(k_{max} \_ n \log(n)) = O(n \log(n))$ running time for this process through proper design of the various data structures. This efficiency is desirable to scale the process to large networks.For the purpose of augmenting the topology, a chosen neighbor is classified as either primary or secondary. The AugConnect procedure augments the topology by adding a secondary chosen neighbor in case a primary chosen neighbor fails.

```
AugConnect ()
  1 for each primary n ∈ chosenset
  2   do get copy TG of topology graph
  3      remove n from TG
  4      run Dijkstra on TG
  5      if destination d is unreachable
  6         then let m be the closest neighbor m of n that makes d
                 reachable
  7              SetChosen (m; candidateset; chosenset)
  8              flag m as secondary
```

In the example of **Figure 4**, node A would augment its connectivity by establishing a link with either node C or E, because if node B, which a primary chosen neighbor, fails, then the rest of the nodes become unreachable.

[0080] When the underlying topology information is available, a node on an on-going basis performs to a DNS look up to discover new nodes that are close to it. In the example, if a new node is added between A and B or if E become closer to A than B according to the metric, then A can update it link accordingly. Procedure ClosestConnect provides for this process.

```
ClosestConnect ()
  1 encode IP address and domain into a look up string
  2 perform DNS look up request
  3 rset ← {addresses in DNS response}
  4 if rset ≠ 0
  5   then for each x ∈ rset          ► done sequentially or concurrently
  6        do SetCandidate (x; candidateset; chosenset)
```

```
7               evaluate link metric to x
8               rset ← rset - x
9 for 1 to |candidateset|
10  do select the best x ∈ candidateset according to link metric
11          SetChosen (x; candidateset; chosenset)
12          if k_high is reached
13            then select the worst x ∈ chosenset according to link metric
14                  UnsetChosen (x; candidateset; chosenset)
15 AugConnect ()
```

Main Event-Handling Loop

[0081] The following pseudo-code representation describes the main procedure executed at a node. When the node first boots up, it consults its configuration file as discussed above. Next, any static tunnels that have been manually configured are established. Thereafter, the node executes TUND_Join to join the VON by dynamically establishing its candidate and chosen neighbor sets. After joining the VON, the node waits for messages from other nodes or other indications of network condition changes and acts accordingly to reestablish its connectivity.

```
MainLoop ()
    1 read configuration file
    2 read static tunnel information
    3 set-up static tunnels
    4 TUND_Join ()
    5 while 1
    6   do wait for message msg or timeout
    7          if timeout
    8                  then service time-out handlers,
    9                    i.e., receiving a response, retries, or periodic events
    10                   if closestconnect_time expired
    11                         then ClosestConnect ()
    12                   else if msg is a shutdown request
    13                         then TUND_Leave ()
    14                               break
    15                   if msg is a request
    16                         then ReceiveRequest (msg)
    17                   else if msg is a response
```

27

```
18                    then pass to response handler
19              if n ∈ chosenset failed      ▶ failure indication
20                    then chosenset ← chosenset - n
21                    remove and cleanup the tunnel link to n
22                    test ← ConnectivityTest ()
23                    if test = false
24                          then TUND_Join ()
25                          else MinConnect ()
26                          MaxConnect ()
27                          AugConnect ()
28 tear down static tunnels
29 exit
```

Summary of Configurable Variables

[0082] The following is a list of variables used in the above pseudo-code descriptions. Note that the specified default values can always be overridden in a node configuration file as needed.

k_{min}: the minimum number of non-static neighbors to have (default: 1):

k_{max}: the maximum number of non-static neighbors to have (default: 10).

k_{low}: the low water number of non-static neighbors to have (should be a function of k_{min}).

k_{high}: the high water number of non-static neighbors to have (should be a function of k_{max}).

closestconnect_time: time to check if there is a new closest node.

max_retry: maximum number o time to retry sending a request.

rednegcache_time: time to keep a negative cache of a redirecting neighbor.

max_core_lookup: number of core lookup trials.

join_core_timeout: interval to wait before retrying failed connectivity test.

max_routeconverge_time: maximum time for the underlying routing protocol to converge.

[0083] Thus, an architecture and method to dynamically build and maintain a virtual overlay network over the Internet or other internetwork has been described. Although the foregoing description and accompanying figures discuss and illustrate specific embodiments, the present invention is to be measured only in terms of the claims that follow, and their equivalents.

CLAIMS

What is claimed is:

1. A method, comprising dynamically maintaining a virtual overlay network (VON) over an internetwork through inter-node communications that include messages adapted to establish node neighbors, the messages being passed over tunnels within the internetwork.

2. The method of claim 1 wherein some of the messages are used to query neighbor nodes about other neighbors thereof.

3. The method of claim 1 wherein node neighbors are established according to connectivity constraints of a node.

4. The method of claim 3 wherein one of the connectivity constraints comprises a maximum number of neighbors.

5. The method of claim 3 wherein one of the connectivity constraints comprises a communication path to a core node.

6. The method of claim 1 wherein node neighbors comprise candidate neighbors and chosen neighbors.

7. The method of claim 6 wherein node neighbors are selected according, at least in part, to link cost metrics between nodes.

8. The method of claim 7 wherein the link cost metrics comprise one or more of link latency, available bandwidth, and packet loss rate.

9. The method of claim 8 wherein link latency is estimated based on packet round trip time.

10. The method of claim 8 wherein available bandwidth is estimated using a packet-pair technique.

11. The method of claim 8 wherein available bandwidth is estimated according to transfers of blocks of data.

12. The method of claim 1 wherein one or more of the messages comprise redirection messages.

13. The method of claim 1 wherein the inter-node communication include directory requests based on a domain name system for the internetwork.

14. The method of claim 13 wherein requesting node information is encoded in the directory requests.

15. The method of claim 13 wherein responses to the directory requests include identifications of nodes close to a requesting node.

16. The method of claim 15 wherein candidate neighbors are selected from the responses to the directory requests.

17. The method of claim 15 wherein chosen neighbors are selected from the responses to the directory requests.

18.A virtual overlay network comprising:

a number of nodes interconnected through virtual links so as to establish a topology,

wherein the nodes are configured to execute distributed algorithms to dynamically update the topology in response to changing conditions in an underlying internetwork by exchanging inter-node communications that include messages used to establish node neighbors.

19. The virtual overlay network of claim 18 wherein the changing conditions comprise one or more of: communication link failures and node failures.

20. The virtual overlay network of claim 18 further comprising a manual topology management console.

21. The virtual overlay network of claim 18 wherein the distributed algorithms include a virtual link management process at one or more of the nodes.

22. The virtual overlay network of claim 21 wherein at least one of the nodes, comprises a directory node configured to receive look up requests from one or more of the nodes and to return in response thereto one or more addresses of nearby nodes in the virtual overlay network.

23. The virtual overlay network of claim 18 wherein the distributed algorithms include processes adapted to allow each of the nodes to evaluate choices for node neighbors.

24. The virtual overlay network of claim 18 wherein the distributed algorithms include processes adopted to allow each of the nodes to satisfy and maintain connectivity constraints for the virtual overlay network.

25. The virtual overlay network of claim 18 wherein the distributed algorithms include processes adopted to allow nodes to update their individual sets of chosen neighbors.

26. The virtual overlay network of claim 18 wherein the nodes are further configured to evaluate one or more link metrics when establishing node neighbors.

27. The virtual overlay network of claim 26 wherein the link metrics comprise one or more of latency, available bandwidth and loss rate.

28. The virtual overlay network of claim 27 wherein latency is estimated using round trip time.

29. The virtual overlay network of claim 27 wherein available bandwidth is estimated using either or both of a packet pair technique and/or a transfer of data.

30. The virtual overlay network of claim 18 wherein each node is restricted to a limited number of node neighbors that is less than all of the nodes of the virtual overlay network.

31. The virtual overlay of claim 30 wherein the topology of the virtual overlay network reflects that of an underlying internetwork.

32. The virtual overlay network of claim 18 wherein the messages comprise one or more of:

candidate neighbors requests;
candidate neighbor acknowledgements;
chosen neighbor requests;
chosen neighbor acknowledgements;
redirection requests;
redirection acknowledgements;
neighbor look up requests; and
neighbor look up acknowledgements.

33. The virtual overlay network of claim 18 wherein the node neighbors are segregated into static neighbors, chosen neighbors and candidate neighbors.

34. The virtual overlay network of claim 33 wherein the distributed algorithms provide for converting candidate neighbors to chosen neighbors and vice-versa.

35. A method, comprising:

automatically discovering neighbor nodes of a virtual overlay network by contacting a directory node and requesting a list of one or more nearby nodes withing the virtual overlay network; and

choosing from the list of nearby nodes at least one of the nearby nodes as a chosen neighbor node.

36. The method of claim 35 wherein choosing the chosen neighbor node(s) involves evaluating link cost metrics associated with communication paths to the one or more nearby nodes.

37. The method of claim 36 wherein the link cost metrics are functions of one or more of latency, available bandwidths, and loss rates.

38. The method of claim 36 wherein the link cost metrics are a function of a ratio of latency to available bandwidth.

39. The method of claim 35 further comprising periodically reevaluating choices for the chosen neighbor node.

40. The method of claim 35 wherein choosing the chosen neighbor node is done so as to satisfy connectivity requirements of the virtual overlay network.

41. The method of claim 35 further comprising updating choices for one or more chosen neighbor nodes.

42. The method of claim 35 wherein choosing comprises executing a chosen neighbor heuristic that connects nodes that are close to one another in an underlying internetwork.

43. The method of claim 35 wherein the topology is hierarchical.

44. The method of claim 35 wherein the topology is flat.

45. The method of claim 35 wherein contacting the directory node comprises transmitting a domain name system look up request.

46. The method of claim 45 wherein the domain name system look up request includes an identification of a requesting node encoding therein.

47. The method of claim 46 wherein the identification of the requesting node comprises an IP address.

48. The method of claim 45 wherein the list of one or more nearby nodes is transmitted as part of a domain name system response to the look up request.

49. The method of claim 35 wherein choosing comprises establishing communication tunnels with the chosen neighbor node.

50. The method of claim 35 further comprising leaving the virtual overlay network by affirmatively notifying neighbor nodes of a departure.

51. The method of claim 50 wherein affirmatively notifying neighbor nodes includes redirecting the neighbor nodes to other nodes.

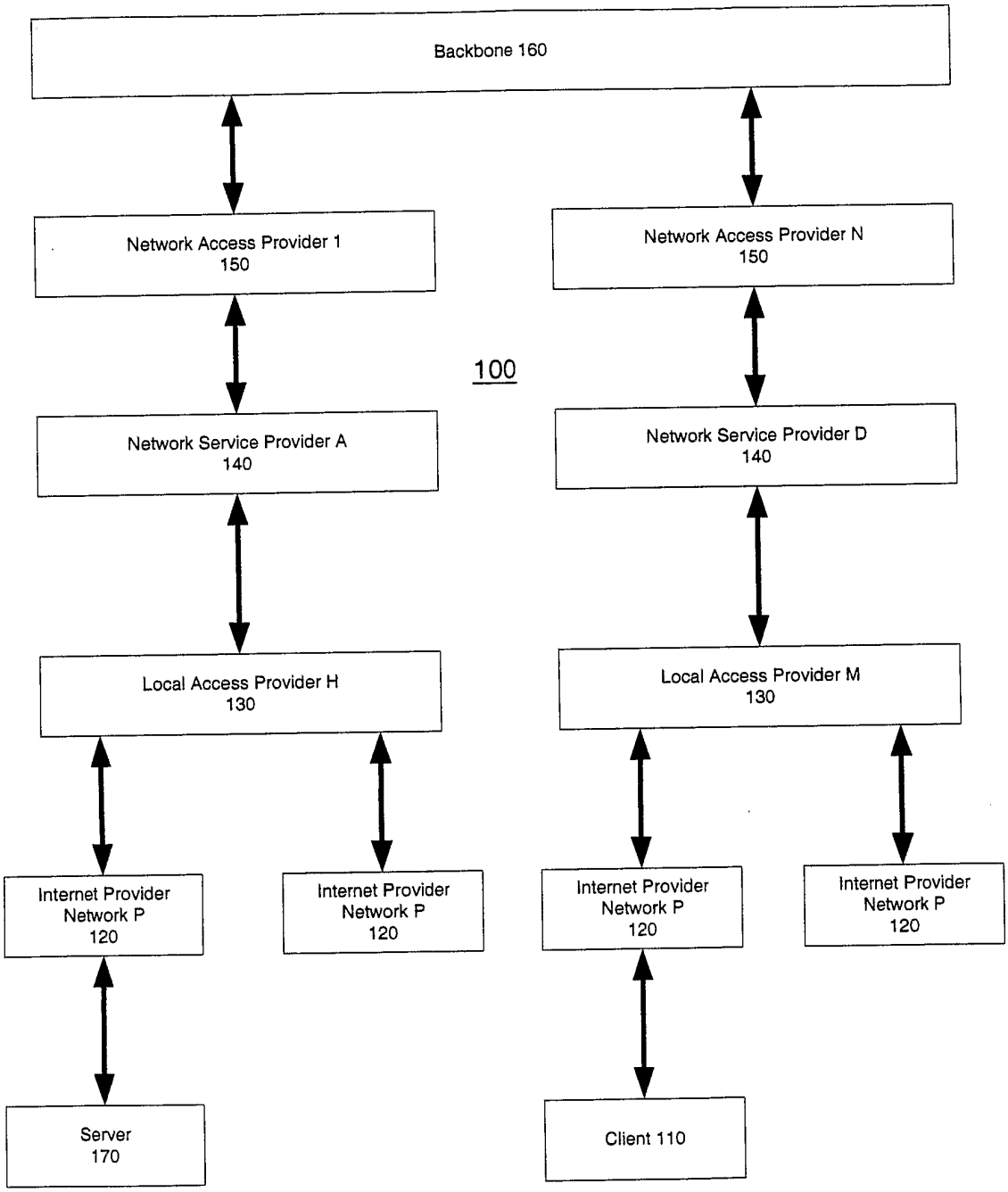

Backbone 160
Network Access Provider 1
150
Network Access Provider N
150
100
Network Service Provider A
140
Network Service Provider D
140
Local Access Provider H
130
Local Access Provider M
130
Internet Provider
Network P
120
Internet Provider
Network P
120
Internet Provider
Network P
120
Internet Provider
Network P
120
Server
170
Client 110

Fig. 1

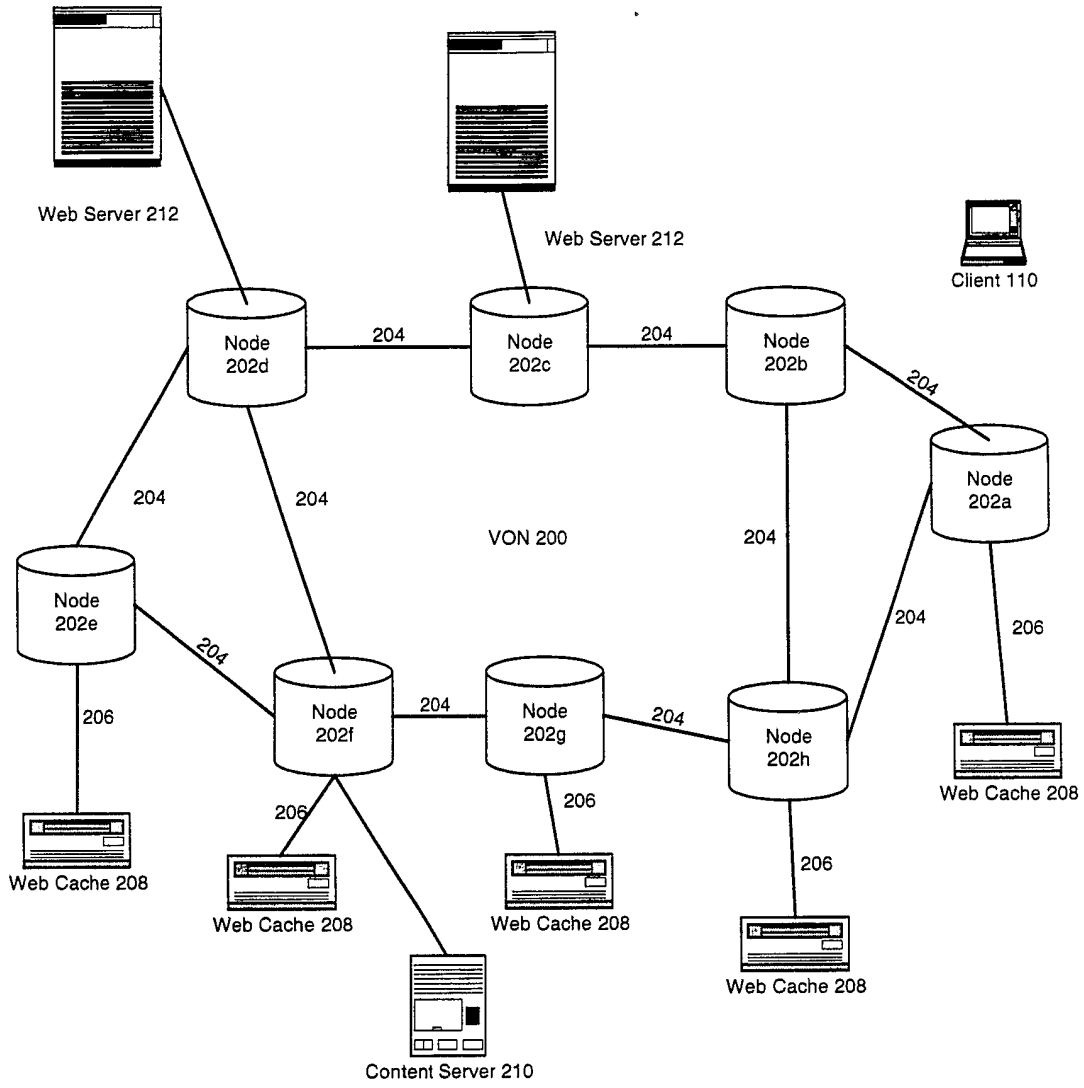

Web Server 212
Web Server 212
Client 110
Node 202d
Node 202c
Node 202b
Node 202a
Node 202e
Node 202f
Node 202g
Node 202h
204
206
VON 200
Web Cache 208
Content Server 210

Fig. 2

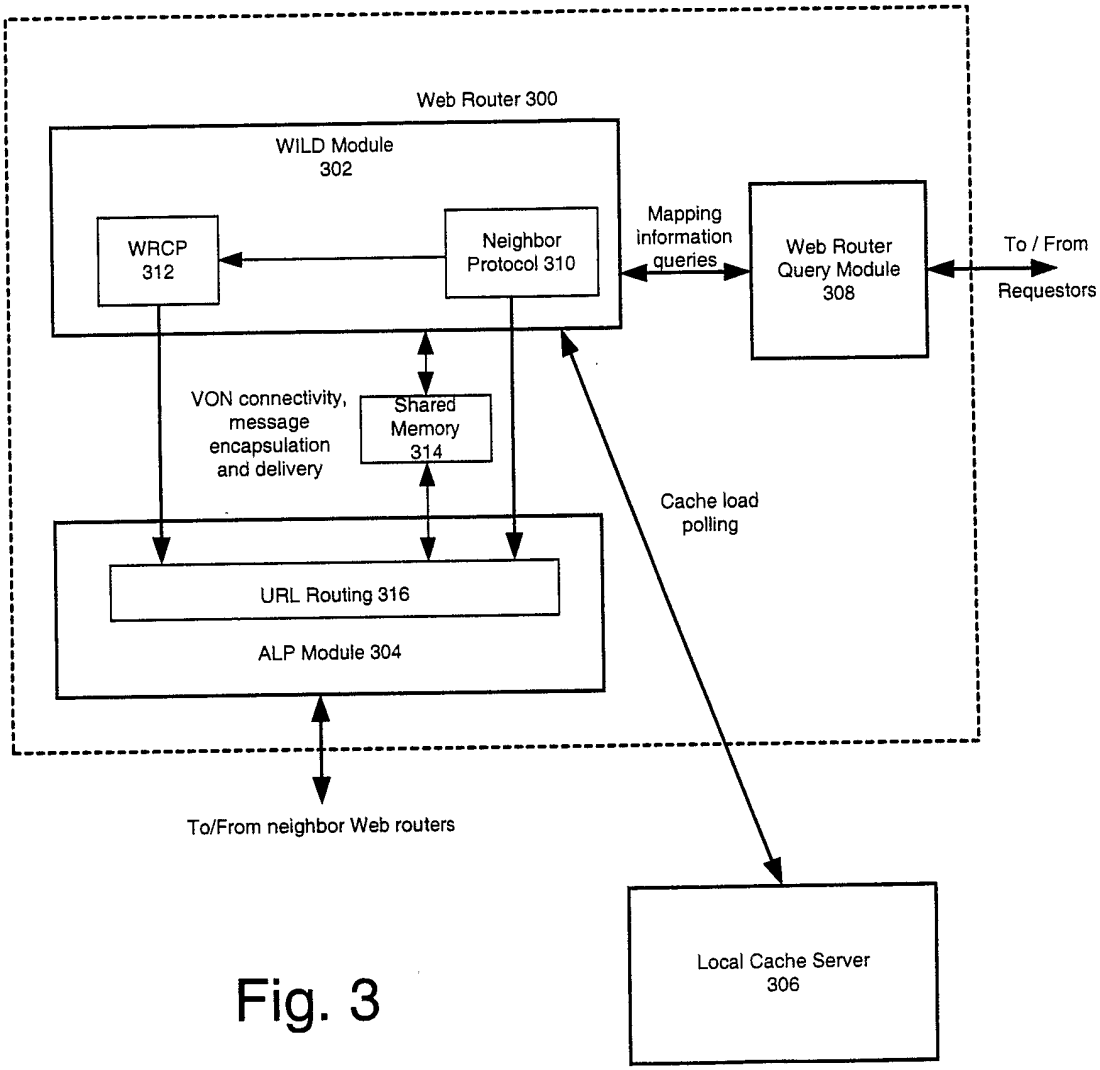

Web Router 300
WILD Module
302
WRCP
312
Neighbor
Protocol 310
Mapping
information
queries
Web Router
Query Module
308
To / From
Requestors
VON connectivity,
message
encapsulation
and delivery
Shared
Memory
314
Cache load
polling
URL Routing 316
ALP Module 304
To/From neighbor Web routers
Local Cache Server
306

Fig. 3

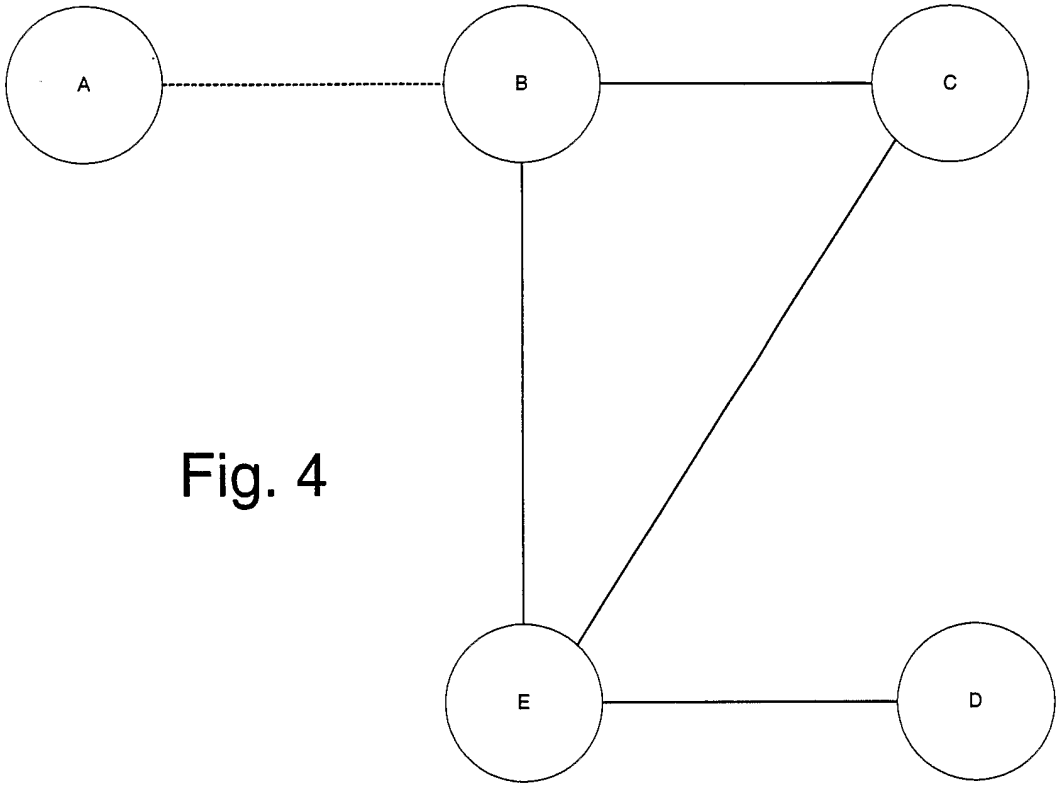


Fig. 4

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US02/28829

A. CLASSIFICATION OF SUBJECT MATTER

IPC(7) :H04J 1/16,3/14
US CL :370/238

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

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 370/229,238,238.1,351,389,392,400,401

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

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X,P	US 6,400,681 B1 (BERTIN ET AL) 04 June 2002, see entire reference.	1-51
A,E	US 2002/0163889 A1 (YEMINI ET AL) 07 November 2002, see figs. 3,4.	1-51
A,P	US 6,377,551 B1 (LUO ET AL) 23 April 2002, see entire reference.	1-51
A	US 6,130,881 A (STILLER ET AL) 10 October 2000, see fig. 1.	1-51

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

*	Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be of particular relevance	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"E"	earlier document published on or after the international filing date	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"&"	document member of the same patent family
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed		

Date of the actual completion of the international search 15 DECEMBER 2002	Date of mailing of the international search report 28 JAN 2003
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 305-3230	Authorized officer AJIT PATEL Telephone No. (703) 305-3900

Form PCT/ISA/210 (second sheet) (July 1998)★

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US02/28829

Box I Observations where certain claims were found unsearchable (Continuation of item 1 of first sheet)

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

1. ☐ Claims Nos.:
because they relate to subject matter not required to be searched by this Authority, namely:

2. ☐ Claims Nos.:
because they relate to parts of the international application that do not comply with the prescribed requirements to such an extent that no meaningful international search can be carried out, specifically:

3. ☐ Claims Nos.:
because they are dependent claims and are not drafted in accordance with the second and third sentences of Rule 6.4(a).

Box II Observations where unity of invention is lacking (Continuation of item 2 of first sheet)

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

1. ☐ As all required additional search fees were timely paid by the applicant, this international search report covers all searchable claims.

2. ☐ As all searchable claims could be searched without effort justifying an additional fee, this Authority did not invite payment of any additional fee.

3. ☐ As only some of the required additional search fees were timely paid by the applicant, this international search report covers only those claims for which fees were paid, specifically claims Nos.:

4. ☐ No required additional search fees were timely paid by the applicant. Consequently, this international search report is restricted to the invention first mentioned in the claims; it is covered by claims Nos.:

Remark on Protest

☐ The additional search fees were accompanied by the applicant's protest.

☐ No protest accompanied the payment of additional search fees.