

(19) World Intellectual Property  
Organization  
International Bureau



(43) International Publication Date  
4 March 2004 (04.03.2004)

PCT

(10) International Publication Number  
**WO 2004/019181 A2**

(51) International Patent Classification<sup>7</sup>: **G06F**  
(21) International Application Number:  
PCT/US2003/026636  
(22) International Filing Date: 25 August 2003 (25.08.2003)  
(25) Filing Language: English  
(26) Publication Language: English  
(30) Priority Data:  
60/405,847 24 August 2002 (24.08.2002) US  
(71) Applicant (for all designated States except US): **INGRIAN NETWORKS, INC.** [US/US]; 475 Broadway Street, Redwood City, CA 94063 (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, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, 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, HU, IE, IT, LU, MC, NL, PT, RO, SE, SI, SK, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

(72) Inventor; and  
(75) Inventor/Applicant (for US only): **FOUNTAIN, Thomas, D.** [US/US]; 123 Positano Circle, Redwood City, CA 94065 (US).  
(74) Agent: **KUDLA, Jonathan**; Perkins Coie LLP, 101 Jefferson Drive, Menlo Park, CA 94025 (US).

**Published:**

— without international search report and to be republished upon receipt of that report

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.

(54) Title: SECURE CONTENT SWITCHING

(57) Abstract: A computer implemented method for optimizing secure content switching; the method includes a client initiating transmission of a secure content request. The secure request is transmitted through a network to a load balancer. The secure request is received at the load balancer and the secure request is forwarded to an individual server of a plurality of servers. The secure request is received and processed at the individual server. The secure request is sent to an appropriate backend web server. A requested secure content is then sent from the appropriate back-end web server to the user via the server, the load balancer, the network and the client.



WO 2004/019181 A2

## **SECURE CONTENT SWITCHING**

### **FIELD OF THE INVENTION**

5 The present invention relates to Internet traffic load balancing and more specifically to optimized load balancing of secure content services.

### **BACKGROUND OF THE INVENTION**

10 Prior art FIG. 1 is a block diagram of a typical Internet traffic communication system 10. The traffic communication system 10 includes a web browser 20, a wide area network ("WAN") such as the Internet 30, a load balancer 40 and individual web servers 50a, 50b and 50c. The web browser 20 can send requests (not shown) for content through the Internet 30. The requests are intercepted by the load balancer 40 that distributes the requests between identical web servers 50a, 50b, and 50c. In this manner, no individual web server 50a, 50b, and 50c will be  
15 overwhelmed by multiple requests for content. This method is often referred to as "horizontal scaling."

When load balancer 40 makes a switching decision based on the content of a request, it is referred to as a content switch or a "level 7" switch. Level 7 switching devices are often used during a secure connection because it is important to support application session tracking.  
20 Session tracking, also referred to as "client-server affinity," is the practice of sending all requests from a client to the same server within a time frame referred to as the "sticky period." In order to achieve this, a load balancer 40 needs to be able to distinguish the various requests it receives for the purposes of sending the requests to the proper server (50a, 50b, and 50c).

One common prior art method of achieving application session tracking is through the use of an  
25 HTTP "cookie". This approach is available to level-7 type switches only, however. Also, some users have reservations about having an HTTP cookie tracking their movements because the use of cookies is widely abused.

Another prior art method of application session tracking involves identifying a client source IP address. In this manner, a load balancer 40 can keep track of requests from the same IP  
30 address/user. However, large ISP's often use a mega-proxy to aggregate user sessions for performance reasons. As a result, a user may appear to be multiple users, even in the middle of a transaction.

Another problem with application session tracking arises when a server becomes non-functional. Typically when this occurs, a user is switched to a new server automatically. But if the translation was performed via a secure connection, the transaction usually gets lost and the user is forced to start over. In an e-commerce setting, this is often referred to as a "lost shopping  
5 cart." When this occurs, the consumer typically will get discouraged and will likely go to a competitor for the same product.

One common prior-art method of monitoring whether a server is functional is a layer 4 TCP check. A Layer 4 TCP check is commonly defined by the following procedure: An intermediate server sends a TCP SYN (transmission control protocol synchronization) packet to the  
10 destination port on a back-end web server, which indicates that the intermediate server would like to establish a connection. When the back-end web server replies with a TCP SYN-ACK (acknowledge) packet, the intermediate server assumes that the destination server is up. The intermediate server then sends a TCP RST (reset) to the destination server to abort the connection before it is fully opened. If the destination server does not send a TCP SYN-ACK  
15 packet, the server assumes that the destination server is down and the health check fails. As intermediate server exists in user space while all TCP operations are handled by the kernel, this sort of handshake is difficult to implement without either changing the kernel or filtering all incoming TCP packets in user space.

Additionally, a layer 4 TCP check will normally not cause significant processing on the  
20 destination server. Since no connection is established, generally no log entry will be produced to indicate that a connection was or was not successfully made. This is not desirable since it is preferable to have accurate log entries to research problem web servers.

Accordingly, methods and techniques are needed to maintain client-server affinity without using cookies, address the issues involved with mega-proxies and re-direct requests to functional  
25 servers when a destination server stops operating, without loss of information.

**SUMMARY OF THE INVENTION**

The present invention contemplates a variety of improved methods and systems for maintaining client-server affinity in a secure setting. Additionally, methods and systems for monitoring server functionality are also taught.

5 A computer implemented method for optimizing secure content switching, in accordance with one embodiment of the present invention, includes a client initiating transmission of a secure content request. The secure request is transmitted through a network to a load balancer. The secure request is received at the load balancer and the secure request is forwarded to an individual server of a plurality of servers. The secure request is received and processed at the  
10 individual server. The secure request is sent to an appropriate back-end web server. A requested secure content is then sent from the appropriate back-end web server to the user via the server, the load balancer, the network and the client.

A computer implemented method for optimizing secure content switching, in accordance with one embodiment of the present invention, includes a user sending a secure content request from a  
15 browser executing on a user computer. The secure request is sent through a network to a load balancer. The load balancer receives the secure request and forwards the secure request to an individual server of a plurality of servers. The individual server receives and processes the secure request and sends the secure request to an appropriate back-end web server. The appropriate back-end web server then sends a requested secure content back to the user via the server, the  
20 load balancer, the network and the browser.

A computer implemented method for optimizing secure content switching, in accordance with another embodiment of the present invention, includes a user sending a secure content request from a browser executing on a user computer. The secure request is sent through a network to a load balancer. The load balancer receives the secure request and forwards the secure request to  
25 an individual secure reverse-proxy server of a plurality of secure reverse-proxy servers. The individual secure reverse-proxy server receives the secure request and decouples a secure information component from the secure request. The individual secure reverse-proxy server forwards a decoupled request to the appropriate back-end web server. The appropriate back-end web server then sends a requested secure content back to the user via the individual secure  
30 reverse-proxy server, the load balancer, the network and the browser.

A system for optimizing secure content switching, in accordance with another embodiment of the present invention, includes a server group interface that defines a plurality of back-end servers each dedicated to hosting a specific type of secure content. Also included is a destination server group properties segment, embedded in the server group interface, wherein the destination

server group properties segment is utilized for determining an association between the specific type of secure content and a subset of the plurality of back-end servers.

5 A computer implemented method for monitoring a status of a server in a communications environment, in accordance with an embodiment of the present invention, includes opening a connection with a server once every "N" minutes. Monitoring for an acknowledgement of the connection from the server. A new request is sent to the server if the acknowledgement is received and the new request is not sent to the server if the acknowledgement is not received.

10 A computer implemented method for optimizing secure content switching, in accordance with yet another embodiment of the present invention, includes receiving a secure request and determining an appropriate back-end web server to handle the secure request. The secure request is then sent to the appropriate back-end web server wherein the appropriate back-end web server may contact a network attached encryption server for a secure component of the secure request. The secure content is then forwarded to a client.

15 A system for optimizing secure content switching, in accordance with a final embodiment of the present invention, includes a content switching means for maintaining client-server affinity. Also included is a server monitoring means for monitoring a server.

20 The present invention maintains client-server affinity in a secure setting through the content switching in the context of end-to-end secure connections (HTTPS for example), the use of cookies for content switching in a secure context as well as the use of URL-based content switching. An additional advantage of the present invention is the active monitoring methods that are employed for monitoring, logging and maintaining server states (enabled, disabled, etc) in a secure environment.

These and other advantages of the present invention will become apparent to those skilled in the art upon a reading of the following detailed descriptions and a study of the various figures

**BRIEF DESCRIPTION OF THE FIGURES**

Figure 1 is a prior art block diagram of a typical Internet traffic communication system.

Figure 2 is a block diagram of a suitable hardware architecture used for supporting secure content switching, in accordance with the present invention.

Figure 3 is a block diagram of secure content switching, communication system, in accordance with one embodiment of the present invention.

Figure 4 is a flowchart describing a method of secure content switching, in accordance with the present invention.

Figure 5 illustrates a server group definition interface, in accordance with one embodiment of the present invention.

Figure 6 illustrates a server group properties interface, in accordance with the present invention.

Figure 7A illustrates a destination server affinity properties interface, in accordance with the present invention.

Figure 7B is a flowchart that illustrates the URL rewrite method, in accordance with present invention.

FIG. 7C is a flowchart that illustrates the cookie insertion method for maintaining client-server affinity, in accordance with present invention.

Figure 8 illustrates a switching rules definition interface, in accordance with the present invention.

Figure 9A illustrates a destination server active monitoring interface, in accordance with the present invention.

Figure 9B is a flowchart describing the destination server active monitoring process, in accordance with the present invention

Figure 10 illustrates a destination server mode selection interface, in accordance with the present invention.

5 Figure 11 illustrates a forwarding rule statistics selection interface, in accordance with the present invention.

Figure 12 illustrates a destination server statistics interface, in accordance with the present invention.

10 Figure 13 illustrates a detailed view of a destination server log interface, in accordance with the present invention.

Figure 14 illustrates an exemplary implementation of the present invention.

15 Figure 15 is a flowchart describing a method of secure content switching through a secure reverse proxy server, in accordance with the present invention.

FIG. 16 illustrates an exemplary implementation of the present invention.

20 FIG. 17 is a flowchart describing a method of secure content switching utilizing a cryptographic key server, in accordance with the present invention.

**DETAILED DESCRIPTION OF THE INVENTION**

FIG. 1 was described in reference to the prior art. The present invention provides a computer implemented method for optimizing secure content switching. The method includes a client, which can take the form of a web browser executing on a user computer, initiating transmission of a secure content request. Examples of a secure content request could be financial information, health records and the like. The secure request is transmitted through a network to a load balancer. The network can possibly be a WAN, a LAN, an Internet or any equivalent network. The purpose of the load balancer is to evenly distribute incoming requests between a group of servers. The secure request is received at the load balancer and the secure request is forwarded to an individual server of a plurality of servers. The secure request is received and processed at the individual server wherein the request is decrypted and analyzed to determine what back-end web server that the request will be sent to. The secure request is sent to the appropriate back-end web server. A requested secure content is then sent from the appropriate back-end web server to the user via the server, the load balancer, the network and the client.

FIG. 2 is a block diagram of a suitable hardware architecture **70** used for supporting secure content switching, in accordance with the present invention. The hardware architecture **70** includes a central processing unit (CPU) **80**, a persistent storage device **90** such as a hard disk, a transient storage device **100** such as random access memory (RAM), a network I/O device **110**, and an encryption device **120** - all bi-directionally coupled via a databus **130**. As will be readily apparent, the hardware architecture **70** is typical of computer systems and thus the present invention is readily implementable on prior art hardware systems. Other additional components such as a graphics card, I/O devices such as a video terminal, keyboard and pointing device, may be part of the hardware architecture **70**.

Those skilled in the art will appreciate that architecture **70** is but one example of a suitable architecture to support secure content switching. For example, it is not necessary to include a persistent storage device **90** or an encryption device **120**. Other hardware architectures are well known in the art and can serve as a basis for which to implement certain teachings of the present invention.

FIG. 3 is a block diagram of a secure content switching communication system **140**, in accordance with one embodiment of the present invention. The communication system **140** includes a plurality of clients **20**, a WAN **30** such as the Internet, a load balancer **40**, a set of servers **150**, a static page server group **160**, a dynamic page server group **170** and a graphics server **180**. The browser **20** sends requests for secure content (not shown) via Internet **30**. It will

readily be recognized by one skilled in the art that any equivalent network could be used in place of Internet **30** and that the present invention is not necessarily limited in that fashion. The request for secure content is received at load balancer **40** which forwards the request to one of a plurality of servers **150**, typically based on the amount of other requests that the servers **150** are currently  
5 handling. The servers **150** perform SSL handshakes, decrypt data and determine which back end web-server group to send the request to. This decision is based on the type of the secure content request. Examples of back end web-server groups include, but not limited to, a static page server group **160**, a dynamic page server group **170** and a graphics server group **180**. Based on the amount of data for each type, there could be varying numbers of individual back end web-servers  
10 in any one particular group.

To further illustrate, FIG. 4 is a flowchart describing a method **185** of secure content switching, in accordance with the present invention. Beginning at an operation **190**, a user sends a secure content request from a browser executing on a user computer at step **200**. The secure request is sent through a network to a load balancer, via step **210**. At step **220**, the load balancer receives  
15 the secure request and forwards the secure request to an individual server, such as based on the current amount of web traffic that the load balancer is receiving. The individual server then receives and processes the secure request and sends the secure content request to an appropriate back-end web server at step **230**. In the context of the present invention, it should be understood that the phrases "destination server" and "back-end web server" can be used interchangeably and  
20 refer to a server that hosts content. Some criteria for deciding what back-end server to use could be if it is a static web-page, a dynamic web-page or a graphic. At operation **240**, the back-end server sends a requested secure content back to the user via the server, the load balancer, the network and the browser.

The present invention is capable of maintaining client-server affinity. This is accomplished by  
25 creating and configuring a server group, creating/configuring a forwarding rule and defining a switching rule. Each of these steps will be explained in more detail.

FIG. 5 illustrates a server group definition interface **260**, in accordance with one embodiment of the present invention. The server group definition interface **260** is utilized for assigning one or more servers to a specific web-hosting task, for example: static pages, graphic pages, etc.  
30 Included is a group name description segment **270**, a number of servers segment **280** and a list of the servers **290** assigned to that particular group.

Defining the properties, via button **300**, yields a server group properties interface **310** and is shown in FIG. 6. The server affinity cookie domain segment **320** is used for maintaining server

affinity across all forwarding rules that uses the same server group. A forwarding rule defines what back-end web server to use for a particular secure-content request. By specifying a value for segment 320, it is possible for an end user to access the same back-end web server through multiple forwarding rules on different ports. When such a possibility exists, server affinity is not preserved when client browsers do not recognize that the cookies from different forwarding rules originate from the same domain. By specifying a value for this field, it is ensured that all client browsers return the same cookie regardless of which forwarding rule they accessed, thus preserving server affinity. Additionally, a user can select a load balancing algorithm 322. Algorithm 322 defines how a load balancer 40 will forward traffic to a group of servers. In this particular example, servers are selected at random. More specifically, a round robin load balancing technique is used. Other load balancing algorithms could be based on a weighted round robin technique, a least response time technique, a least connections technique, and a weighted least connections technique. These various techniques are well known to those skilled in the art and as such will not be discussed in detail.

With further reference to FIG. 6, individual servers are added via a server list segment 330. The list segment 330 is also utilized for defining the method for maintaining client-server affinity. Also included is a destination server active monitoring segment 340. Segment 340 specifies whether to monitor a server for functionality and will be described in more detail, subsequently. Additionally, HTTP GET response body rules segment 342 can be used to set rules for responses. In other words, if a response from a destination server is received that matches the rule, then that particular server is known to be up and running.

Defining the method for maintaining client-server affinity is accomplished by editing a forwarding rule 350 and is further detailed in FIG. 7A. FIG. 7A illustrates a destination server affinity properties interface 360, in accordance with the present invention. Client-server affinity can be specified via box 370. Affinity will not be maintained if "none" is selected, or will be maintained if the "client-IP" or "cookie insertion" methods are chosen. If affinity is chosen, then a time period also needs to be specified via segment 380. This time period is also sometimes referred to as the "sticky period."

In a preferred embodiment, client-server affinity is maintained via a "URL rewrite" process. Specifically, under the "URL rewrite" tracking option, the information for tracking an application session will be incorporated into the first (or last) component of all host URLs. One skilled in the art will appreciate that a URL rewrite tracking option can be employed in either an

unsecure to secure (HTTP/HTTPS) connection or a secure to secure (HTTPS/HTTPS) connection.

For example, if a particular destination server is the destination server assigned by the server 150 for a client/browser 20, under the Insert Cookie tracking option, a tracking cookie, for example "INGRIAN=\_1\_20\_", is sent back to the client's web browser 20. In subsequent visits from the client, the client's web browser sends the cookie back, which is removed by the server 150. The backend server never sees the "INGRIAN" cookie. It will be appreciated by those skilled in the art that a variety of cookie insertion techniques can be employed without departing from the true spirit and scope of the present invention and the above description is merely one example.

Under the "URL rewrite" tracking option, no tracking cookie is sent to the client's web browser 20. Instead, all the host URL's, embedded in the HTML page that refer to the back-end servers, are "rewritten" to include the session tracking information. An example of a rewritten URL is:

`http://www.xyz.com/INGRIAN=_1_20_/cgi-bin/search/public.htm`

Note that "INGRIAN=\_1\_20\_" is inserted into the URL. This rewrite is done automatically by the server 150. When a request is received from a client 20, the "session tracking" information, embedded in the URL, are automatically removed by the server 150. The back-end servers (160, 170 or 180) do not have any knowledge about the "URL rewrite" and there are no code changes necessary for the back-end server applications to take the advantage of this feature.

To further illustrate, consider this excerpt from a page sent from a back-end server 170 to a server 150:

`<BODY BGCOLOR="#FFFFFF">`

`<FORM ACTION="http://www.xyz.com/cgi-bin/search/public.pl">`

If the Secure Content Switch adds "\_1\_20\_" to associate the client with the particular server, in this case server 1 with a unique id 20, a component, "INGRIAN\_1\_20\_" is added to the host URLs embedded in the HTML page by the server 150. The following is an excerpt from the page modified by the server 150 before sending to a client/browser 20:

`<BODY BGCOLOR="#FFFFFF">`

<FORM ACTION="http://www.xyz.com/INGRIAN\_1\_20\_/cgi-bin/search/public.pl">

5 The user then clicks the "submit button" on the browser **20** and the server **150** strips off "INGRIAN\_1\_20\_", sends the request to the back end server **170**.

FIG. 7B is a flowchart that illustrates the URL rewrite method **381**, in accordance with present invention. Beginning at a start step **382**, an intermediate server assigns to a backend server at step **383**. A request is then sent from an intermediate server to a backend server at step **384**. At a step **385**, the request is received at the backend server and then a URL header for the request is modified at step **386**. The header is modified such that it will now contain information identifying the destination server. The request is then sent from the backend server to the client browser, through the intermediate server, at step **387**. At step **388**, a related request is sent from the client browser to the intermediate server and the related request is received at the intermediate server, via step **389**. The intermediate server removes the modified portion of the URL and sends the related request to the backend server where it is received, via steps **391**, **392** and **393**. The backend server then sends a requested reply to the intermediate server where it is received, at steps **394** and **395**. The related request is then sent to the client browser at step **396**. The method then ends at step **397**.

FIG. 7C is a flowchart that illustrates the cookie insertion method **398** for maintaining client-server affinity, in accordance with present invention. Beginning at a start step **399**, an intermediate server assigns to a backend server at step **401**. A request is then sent from an intermediate server to a backend server at step **402**. At a step **403**, the request is received at the backend server. The request is then sent, along with a cookie, from the backend server to the client browser, through the intermediate server, at step **404**. At step **388**, a related request is sent from the client browser to the intermediate server and the related request is received at the intermediate server, via step **389**. The intermediate server removes the modified portion of the URL and sends the related request to the backend server where it is received, via steps **391**, **392** and **393**. The backend server then sends a requested reply to the intermediate server where it is received, at steps **394** and **395**. The related request is then sent to the client browser at step **396**. The method then ends at step **397**.

As previously summarized, client-server affinity is the practice of ensuring a connection between client and a server is maintained for subsequent requests. Back-end servers may maintain persistent data for applications like shopping carts. The client-IP method of client-server affinity is generally useful but not as effective when mega-proxies are used. All client requests

originating from the same mega-proxy, such as AOL, appear to have the same IP address. Therefore, all requests originating from the same mega-proxy are routed to the same back-end web server when maintaining client-server affinity by client-IP address. The cookie-insertion method is able to handle the problems of the mega-proxy, but requires an end user to have  
5 cookie use enabled on their browser.

FIG. 8 illustrates a switching rules definition interface **390**, in accordance with the present invention. A switching rule configures what back-end server will receive a secure content request, based on the URL or cookie. In the case of a URL-based switching rule, a URL is selected at match segment **410**, a criterion **420** is selected, text **430** is input and a destination port  
10 **450** and a destination protocol **460** (HTTP or HTTPS) are selected. Specific examples include sending URL's that end in ".gif" or ".jpg" to an image servers group, ".jsp" to a dynamic page servers group and ".asp" to a dynamic page servers group.

FIG. 9A illustrates a destination server active monitoring interface **340**, in accordance with the present invention. Destination server active monitoring is a feature that periodically checks  
15 whether a destination server is successfully fulfilling client requests for the purpose of not sending client requests to an inactive web server. In a preferred embodiment of the present invention, destination servers are not checked while they are fulfilling requests, as to not interfere with regular network activity. If it is determined that a server is no longer successfully filling requests, which are sent from servers **150** of FIG. 3, new requests are no longer sent to  
20 that server. The non-functioning server will then be occasionally polled to ascertain if it has since become functional. If it is functioning once again, then new request will then be sent.

Included in segment **340** is an Enable Active Monitoring box **342** used to enable/disable monitoring. Active Monitoring Method box **344** specifies the type of health check to perform. The options for the type of health check is a layer 4 TCP connection, a layer 7 HTTP HEAD  
25 connection or a layer 7 HTTP GET connection. Layer 4 and layer 7 checks will be explained subsequently. Number of fails required box **348** is used to specify the number of successive failed health checks before a server is deemed inactive. Monitoring interval box **352** allows one to specify how often to perform a check on an active server. URL patch box **354** specifies a URL path to an object that resides on a server and is only used for HTTP requests. Logically,  
30 Expected HTTP Response Code box **356** is used to enter an expected HTTP response. An example response code might be "200" or perhaps a range of numbers such as "200-299, 300, 302-304." The aforementioned boxes can be edited via button **358**.

Also included in segment **340** is an HTTP GET Response Body Rules box **360**. The HTTP GET Response Body Rules box **340** provides flexibility in determining the success or failure of HTTP GET health checks. If the response body returned by a backend server matches an HTTP GET Response Body Rule, then the health check passes or fails based on the Active Monitoring Result specified for that rule. Multiple rules can be created for a server group. If more than one rule matches the response body, then the first rule listed takes precedence. If no rule matches, then the default Active Monitoring Result (Successful) is used. Edit **364** is used to edit a rule and Add **364** is used to add a rule.

Components of the rules are summarized in Table I:

TABLE I

| COMPONENTS                          | DESCRIPTION                                                                 |
|-------------------------------------|-----------------------------------------------------------------------------|
| Criterion                           | -“contains” or “does not contain”                                           |
| Text                                | -Used for matching a text string.                                           |
| Case Sensitive                      | -If marked, the rule is case sensitive.                                     |
| Active Monitoring Result            | -“success” or “failure”                                                     |
| Server Group Monitor                | -Configure active server monitoring for a server group.                     |
| IP Forward Server Group             | -Set the destination server affinity properties for a forwarding rule.      |
| Server Group HTTP-Monitor-Rule      | -Add an HTTP GET response body rule.                                        |
| Show Server Group HTTP-Monitor-Rule | -List all of the HTTP GET response body rules for a specified server group. |
| No Server Group HTTP-Monitor-Rule   | -Remove an HTTP GET Response Body Rule.                                     |
| Move Server Group HTTP-Monitor-Rule | -Move a rule up or down within the list of HTTP GET response body rules.    |

To further illustrate, FIG. 9B is a flowchart **500** describing the destination server active monitoring process, in accordance with the present invention. Beginning at a start operation **510**, a connection is opened with a destination server once every “N” minutes at step **520**. If a connection was opened successfully, at step **530**, the destination will continue to receive new requests at operation **540** and the destination server will continue to be tested for functionality,

via step 520. If a connection was not opened successfully, via step 530, then new requests will no longer be sent to the destination server 550. Afterward, the destination server will continue to be polled every "P" minutes at step 560. If a connection is still not successful, via step 570, then the destination server will continue to be polled at step 560 for functionality. If a connection was successful, then new requests will once again be sent to the destination server at step 580 and the destination server will again be periodically tested every "N" minutes at step 520.

FIG. 10 illustrates a destination server mode selection interface 590, in accordance with the present invention. The destination server mode selection interface 590 allows for individual destination servers to be selectively enabled, disabled or disable any new connections to a server, while any current connections are allowed to complete. Included is a server identification segment and a server mode segment 610. To further elaborate, "enable" allows for client requests to be sent to the destination server. "Disable" allows for client requests to not be sent to the destination server all active connections are immediately terminated. Lastly, "disable new connections to server" provides for fulfilling client requests for active connections. However, no new active connections are allowed to be created. This setting is useful for gradually shutting down a server.

FIG. 11 illustrates a forwarding rule statistics selection interface 620, in accordance with the present invention. Forwarding rule statistics selection interface 620 can be utilized for displaying statistics relating to total connections, active connections, elapsed time up or down and general status. Included is a listing of forwarding rules 630 that includes associated information such as a local IP 640, a local port 650 and a local protocol 660. Individual statistics for a selected forwarding rule can be accessed via button 670. Alternatively, statistics for all rules can be displayed via button 680.

FIG. 12 illustrates a destination server statistics interface 690, in accordance with the present invention. Destination server statistics interface 690 displays the forwarding rules 700, 710, 720 and 740, as well as any associated switching rules 750 and 760 and associated destination servers (770 and 780).

Destination server statistics are tabulated and displayed on a per-forwarding-rule basis. Therefore, if a destination server is specified in multiple different forwarding rules or switching rules, then statistics for that destination server are displayed separately for all forwarding rules in which it is specified as a destination server. For example, destination server 770 is specified in three forwarding rules (710, 720 and 740). The destination server statistics show that forwarding

rule #2 **710** accounts for 80 connections, forwarding rule #3 **720** accounts for 65 connections and forwarding #4 accounts for 411 connections.

FIG. 13 illustrates a detailed view of a destination server log interface **790**, in accordance with the present invention. Destination server log interface **790** reflects changes in the status of destination servers and provides information as to why the status of a destination server has changed. Some aspects include the log file which is a drop-down box used for selecting the current log or older log files. The "show last number of lines" **810** is a drop-down box from which to select the number of destination server log entries to view. The show button **820** is used for displaying the last few lines of the activity log. The download button **830** is used for downloading the destination server log to a browser (not shown). The clear button **840** can be used to clear a selected activity log and the rotate now button **850** is used for closing a current log and starting a new log. Also included in interface **790** is log display segment **860** that displays the actual log.

Functional details relating to active destination server monitoring will now be presented that will allow one skilled in the art to practice the present invention. The server ought to use a simple, configurable method to determine if back-end web servers are up or down.

The health checks performed by the server should be the only method used to determine the status of a server. Because of this, health checks should be performed much more frequently. An operator of the server should be able to configure the health check interval in seconds, instead of minutes. An operator of the server should also be able to configure the number of successive health check failures that are required before a server is marked down.

For the present invention either a modified layer 4 or a layer 7 health check can be performed to determine if a destination server is functional. In the case of the layer 4 variety, the server will perform a Layer 4 health check by the following method: the server sends a TCP SYN packet to the destination port on the destination server. This indicates that the server would like to open a connection. If the destination server replies with a TCP SYN-ACK packet, health check is successful. The server then sends a TCP SYN to the destination server to fully establish the TCP connection. Immediately afterward, the server will send a TCP FIN to close its end of the connection. If the destination server does not send a TCP SYN-ACK packet, the health check will fail.

This form of health check can possibly add a very small amount of extra overhead since a full connection is established. This overhead is incurred both on the server and on the destination

server. However, in most cases this overhead is insignificant. Furthermore, this form of health check will appear as a full connection to the destination server and may be logged or cause other processing.

5 A Layer 7 HTTP check consists of an HTTP request to the destination server to make sure that the destination server is still correctly serving web pages. A check can be performed either by issuing a GET request or a HEAD request. For either request, the server will open a connection to the destination server and send the HTTP request for a user specified URL. If the server cannot connect to the destination server, or if the destination server does not reply with a valid HTTP response, the health check will be considered failed. Otherwise, the server will check the  
10 HTTP return code. If this code does not fall in the list of expected ranges, then the health check will fail. If the check method was a GET request, the server will further check the response body by matching it against a list of rules specified by the operator of the server. These rules determine if the health check is successful or not.

15 The server operator can configure what constitutes a successful health check. For a Layer 4 TCP health check, there are no parameters to configure. If a connection was established, the health check was successful. If a connection could not be established, the health check failed.

For a Layer 7 HTTP health check, the server operator can configure the following parameters: the health check URL, the URL to be sent in the HTTP request and the expected HTTP response code. The HTTP response code is the code that the server expects to receive from the destination  
20 server. The server operator can configure this to match either a specific response code or range of response codes, or to match any response code. If the destination server returns a different response code than the expected value, the server will assume the health check failed and will mark that particular destination server down.

25 For HTTP GET health checks, the server operator can configure the status of a health check based on the response body that is sent back by the destination server. In order to do so, the server operator can specify several rules for matching text in the response body. For each rule, the server operator can specify if a match indicates a successful health check or a failed health check. If multiple rules match, the first rule listed will take precedence.

30 The server can log informative messages when a health check fails so that the server operator can determine why the server thinks that the destination server is down. Each time an active monitoring check fails, a message of the following form can be logged in the destination server log:

“2002-08-19 14:56:04 adam-194 ServerMonitor: Server (192.168.1.96) of Server Group (http://192.168.1.96:80) in Forwarding Rule (http://0.0.0.0:80) failed active monitoring check 1 of 3: Connection refused by server”

5

The log message will list the destination server in question, how many successive health checks have failed (out of the number allowed before the destination server is marked down), and the reason why the health check failed. Similarly, for a TCP health check, there are a variety of error messages available.

10

When a server goes down, a message of the following form will be logged:

“2002-08-19 17:28:38 adam-194 ServerMonitor: Server (192.168.1.96) of Server Group (http://mygroup:80) in Forwarding Rule (http://0.0.0.0:80) marked inactive: Server failed 3 successive monitoring checks”

15

When a server comes up, a message of the following form will be logged:

“2002-08-19 17:39:43 adam-194 ServerMonitor: Server (192.168.1.96) of Server Group (http://mygroup:80) in Forwarding Rule (http://0.0.0.0:80) marked active: Active monitoring check succeeded”

20

FIG. 14 illustrates an exemplary implementation 870 of the present invention. In this embodiment, there is a client 20 that can send requests for secure content (not shown) via Internet 30. The request for secure content is received at load balancer 40 which forwards the request to one of a plurality of secure reverse proxy servers 910 (SRP's), typically based on the amount of other requests that the SRP's 910 are currently handling.

25

The SRP 910 is a device that intercepts requests for secure content prior to the demand being received by a back-end web server. The SRP 910 establishes an encrypted session with the web browser 880 to facilitate the SRP's ability to examine the secure content. Once the secure request is decrypted, the SRP 910 examines its cache and determines if the requested content is available and which back-end web server (160, 170 or 180) should be used, based on the type of secure content (static page, dynamic page or graphic). . If the requested content is available, the SRP 910 encrypts it using the established session keys with the web browser and transmits the

30

information. More information regarding secure reverse proxies is available in U.S. Patent Serial No.10/205,575 (Atty. Docket No. 36321-8010.US01), filed on July 24, 2002, entitled "Method and System for Caching Secure Web Content", previously incorporated by reference.

FIG. 15 is a flowchart **950** describing a method of secure content switching through a secure reverse proxy server, in accordance with the present invention. Beginning at operation **960**, a user sends a secure content request from a browser executing on a user computer at step **970**. The secure request is sent through a network to a load balancer, via step **980**. At step **990**, the load balancer receives the secure request and forwards the secure request to an individual secure reverse-proxy server of a plurality of secure reverse-proxy servers. The individual secure reverse-proxy server receives the secure request, at step **1000** and decouples a secure information component from the secure request. In the context of the present invention, the term "decouple" refers to removing a portion of an HTTP or HTTPS request wherein the part that is removed can be either a secure or insecure component. At step **1010**, the individual secure reverse-proxy server forwards a decoupled request to the appropriate back-end web server. The appropriate back-end web server then sends a requested secure content back to the user via the individual secure reverse-proxy server, the load balancer, the network and the browser, at step **1020**. The process then ends at step **1030**.

FIG. 16 illustrates an exemplary implementation **1040** of the present invention. Embodiment **1040** includes a plurality of clients **20**, a WAN **30** such as the Internet, a load balancer **40**, a set of servers **150**, a static page server group **160**, a dynamic page server group **170**, a graphics server **180** and a networked attached encryption (NAE) server **1050**. NAE server **1050**. Services requested by the clients **20** may specifically involve cryptographic services, or may precipitate the need for cryptographic services. For example, the client requested services may require the retrieval of encrypted data residing on one of back-end web-servers **160**, **170** or **180**. The NAE server **1050** is available to back-end web servers **160**, **170** or **180** to perform cryptographic services, thus offloading the computational intensities of cryptographic services from the back-end web-servers **160**, **170** or **180**. Some of these services include performing SSL handshakes, decrypting data and encrypting data. More information regarding cryptographic key servers is available in U.S. Provisional Patent Application No.60/395,685 (Atty. Docket No. 36321-8015.US00), filed on July 12, 2002, entitled "Cryptographic Key Server", previously incorporated by reference.

FIG. 17 is a flowchart **1060** describing a method of secure content switching utilizing a cryptographic key server, in accordance with the present invention. Beginning at an operation

1070, a secure request is received at a server 150. Then it is determined what back-end server that the request should be sent to, at an operation 1090. The request is sent to the correct back-end web server, at operation 1110. At 1120, the back-end server initiates the necessary processing action that may include contacting the NAE server 1050 for the secure component.  
5 The encrypted secure content is then forwarded to a client at operation 1130. The method then ends at operation 1140.

In the context of the present invention, a content switching means can include the use of cookies content switching in a secure content, use of URL-based content switching and the like. A server monitoring means includes techniques for monitoring, logging and maintaining server states  
10 (enabled, disabled, etc) in a secure context.

The present invention maintains client-server affinity in a secure setting through the use of flexible forwarding rules and associated switching rules. Additionally, robust, continual monitoring of back-end web servers is achieved that results in an improved method of health checks.

15 In addition to the above mentioned examples, various other modifications and alterations of the invention may be made without departing from the invention. Accordingly, the above disclosure is not to be considered as limiting and the appended claims are to be interpreted as encompassing the true spirit and the entire scope of the invention.

*What is claimed is:*

CLAIMS

1. A computer implemented method for optimizing secure content switching, the method comprising:

5 a client initiating transmission of a secure content request;  
transmitting the secure request through a network to a load balancer;  
receiving the secure request at the load balancer and forwarding the secure request to an individual server of a plurality of servers;  
receiving and processing the secure request at the individual server;  
10 sending the secure request to an appropriate back-end web server; and  
sending a requested secure content from the appropriate back-end web server to the user via the server, the load balancer, the network and the client.

2 The method as recited in claim 1 wherein the requested secure content is not secure.

3. The method as recited in claim 1 wherein the server decides which appropriate back-end web server to use based on a type of the secure content request.

4. The method as recited in claim 3 wherein the type of the secure content request is a static web-page.

5. The method as recited in claim 3 wherein the type of the secure content request is a dynamic web-page.

6. The method as recited in claim 3 wherein the type of the secure content request is a graphic.

7. The method as recited in claim 11 wherein a user-server affinity is maintained during a connection between the client and the server.

8. The method as recited in claim 7 wherein the user-server affinity is maintained by using a client IP address.

9. The method as recited in claim 7 wherein the user-server affinity is maintained by using a cookie insertion.

10. The method as recited in claim 7 wherein the user-server affinity is maintained by using a URL-rewrite method.

11. The method as recited in claim 7 wherein the user-server affinity is maintained for a default time period.

12. The method as recited in claim 1 wherein a plurality of back-end web servers are periodically checked for functionality.

13. The method as recited in claim 12 wherein an individual back-end web server of the plurality of back-end web servers is no longer sent the secure content request if the individual back-end web server is determined to be non-functional.

14. The method as recited in claim 1 wherein if the secure content request matches a pattern, then the secure content request is sent to a certain set of back-end servers.

15. The method as recited in claim 1 wherein the client is a browser executing on a user computer.

16. The method as recited in claim 1 wherein the network is the Internet.

17. The method as recited in claim 1 wherein the load balancer uses a round robin load balancing technique.

18. The method as recited in claim 1 wherein the load balancer uses a weighted round robin load balancing technique.

19. The method as recited in claim 1 wherein the load balancer uses a least connections load balancing technique.

20. The method as recited in claim 1 wherein the load balancer uses a weighted least connections load balancing technique.

21. A computer implemented method for optimizing secure content switching, the method comprising:

a client initiating transmission of a secure content request;

sending the secure request through a network to a load balancer;

5 receiving the secure request at the load balancer, and forwarding the secure request to an individual secure reverse-proxy server of a plurality of secure reverse-proxy servers;

receiving the secure request, at the individual secure reverse-proxy server and decoupling a secure information component from the secure request;

forwarding a decoupled request, from the individual secure reverse-proxy server to the

10 appropriate back-end web server; and

sending a requested secure content back from the appropriate back-end web server to the user via the individual secure reverse-proxy server, the load balancer, the network and the browser.

22. The method as recited in claim 21 wherein the individual secure reverse-proxy server  
15 decides which appropriate back-end web server to use, based on a type of the decoupled request.

23. The method as recited in claim 22 wherein the type of the decoupled request is a static web-page.

20 24. The method as recited in claim 22 wherein the type of the decoupled request is dynamic web-page.

25. The method as recited in claim 22 wherein the type of the decoupled request is a graphic.

25 26. The method as recited in claim 21 wherein a user-server affinity is maintained during a connection between the user computer and the wherein the individual secure reverse-proxy server.

27. The method as recited in claim 26 wherein the user-server affinity is maintained by using  
30 a client IP address.

28. The method as recited in claim 26 wherein the user-server affinity is maintained by using a cookie insertion.

29. The method as recited in claim 26 wherein the user-server affinity is maintained for a default time period.

30. The method as recited in claim 21 wherein a plurality of back-end web servers are periodically checked for functionality.

31. The method as recited in claim 30 wherein an individual back-end web server of the plurality of back-end web servers is no longer sent the decoupled request if the individual back-end web server is determined to be non-functional.

32. The method as recited in claim 31 wherein the individual back-end web server is periodically checked for functionality and sent a new decoupled request once it is determined that the individual back-end server is once again functional.

33. A system for optimizing secure content switching comprising:  
a server group interface that defines a plurality of back-end servers each dedicated to hosting a specific type of secure content; and  
a destination server group properties segment embedded in the server group interface wherein the destination server group properties segment is utilized for determining an association between the specific type of secure content and a subset of the plurality of back-end servers.

34. The system as recited in claim 33 wherein the destination server group properties segment further defines a server group name, a plurality of destination server monitoring properties and a plurality of server forwarding rules wherein an individual forwarding rule is active at any one time.

35. The system as recited in claim 34 further comprising a forwarding rule definition segment embedded in the destination server group properties segment wherein the forwarding rule definition segment is utilized for defining a method of client-server affinity and a plurality of switching rules for defining the association between the specific type of secure content and the subset of the plurality of back-end servers.

36. A data structure for optimizing secure content switching comprising:  
a list of switching rules; and

a plurality of servers wherein the list of switching rules defines a subset of the plurality of servers to use when a secure content request matches an individual switching rule.

37. A software package for optimizing secure content switching based upon associating a secure content request type with a dedicated group of back-end web servers.

38. The software package of claim 37 wherein the secure content is a static web-page.

39. The software package of claim 37 wherein the secure content is a dynamic web-page.

40. The software package of claim 37 wherein the secure content request type is not secure.

41. The software package of claim 37 wherein associating the secure content request type with the dedicated group of back-end web servers is accomplished by tracking a client IP address.

42. The software package of claim 37 wherein associating the secure content request type with the dedicated group of back-end web servers is accomplished by using a cookie insertion.

43. The software package of claim 37 wherein associating the secure content request type with the dedicated group of back-end web servers is accomplished by using a URL-rewrite method.

44. A computer implemented method for monitoring a status of a server in a communications environment comprising:

opening a connection with a server once every "N" minutes;  
monitoring for an acknowledgement of the connection from the server;  
sending a new request to the server if the acknowledgement is received; and  
not sending the new request to the server if the acknowledgement is not received.

45. The method as recited in claim 44 further comprising:  
opening a new connection with the server every "P" minutes wherein the acknowledgement was not received;  
monitoring for a new acknowledgement of the new connection from the server;

sending the new request to the server if the new acknowledgement is received; and  
not sending the new request to the server if the new acknowledgement is not received.

46. A computer implemented method for maintaining client-server affinity via a URL-  
rewrite, the method comprising:

sending a request from back-end web server to an intermediate server;

receiving the request at the intermediate server;

modifying a URL header for the request wherein the URL header now contains  
information identifying the back-end web server;

sending the request with the modified URL header from the intermediate server to a  
client browser;

receiving the request at the client browser;

sending a related request from the client browser to the intermediate server;

receiving the related request at the intermediate server;

identifying the appropriate back-end server based on the URL header;

removing information relating to the back-end server from the URL header; and

sending the related request to the back-end server.

47. An apparatus for secure content switching that performs SSL handshakes, decrypts data  
and determines which back-end web server should handle a request.

48. The apparatus of claim 47 wherein the request is a static web-page.

49. The apparatus of claim 47 wherein the request is a dynamic web-page.

50. The apparatus of claim 47 wherein the request is not secure.

51. The apparatus of claim 47 wherein determining which back-end web server should  
handle a request is accomplished by tracking a client IP address.

52. The apparatus of claim 47 wherein determining which back-end web server should  
handle a request is accomplished by using a cookie insertion.

53. The apparatus of claim 47 wherein determining which back-end web server should  
handle a request is accomplished by using a URL-rewrite method.

54. A computer implemented method for optimizing secure content switching, the method comprising:

receiving a secure request;

5 determining an appropriate back-end web server to handle the secure request;

sending the secure request to the appropriate back-end web server wherein the appropriate back-end web server may contact a network attached encryption server for a secure component of the secure request; and

forwarding the secure content to a client.

10

55. The method as recited in claim 54 wherein the server decides which appropriate back-end web server to use based on a type of the secure content request.

56. The method as recited in claim 55 wherein the type of the secure content request is a static web-page.

15

57. The method as recited in claim 55 wherein the type of the secure content request is dynamic web-page.

58. The method as recited in claim 55 wherein the type of the secure content request is a graphic.

20

59. A system for optimizing secure content switching, the method comprising:  
a content switching means for maintaining client-server affinity; and  
a server monitoring means for monitoring a server.

25

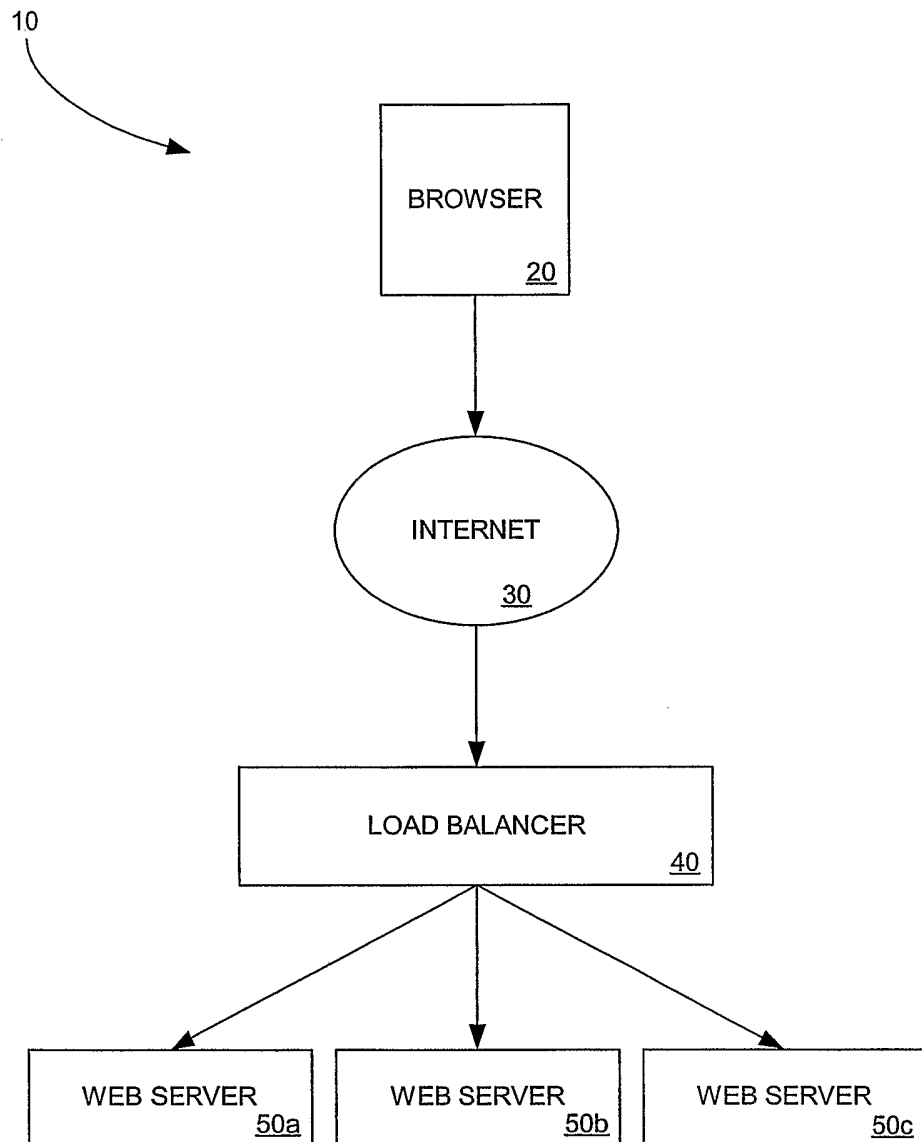


FIG. 1  
"PRIOR ART"

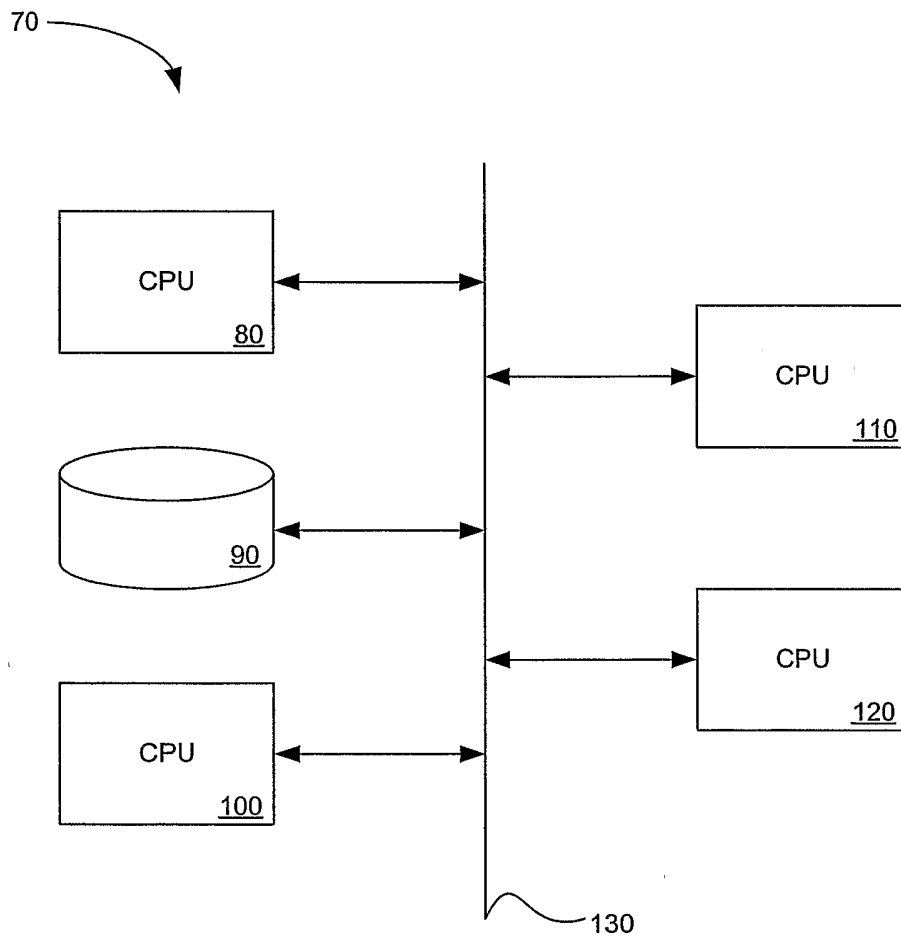


FIG. 2

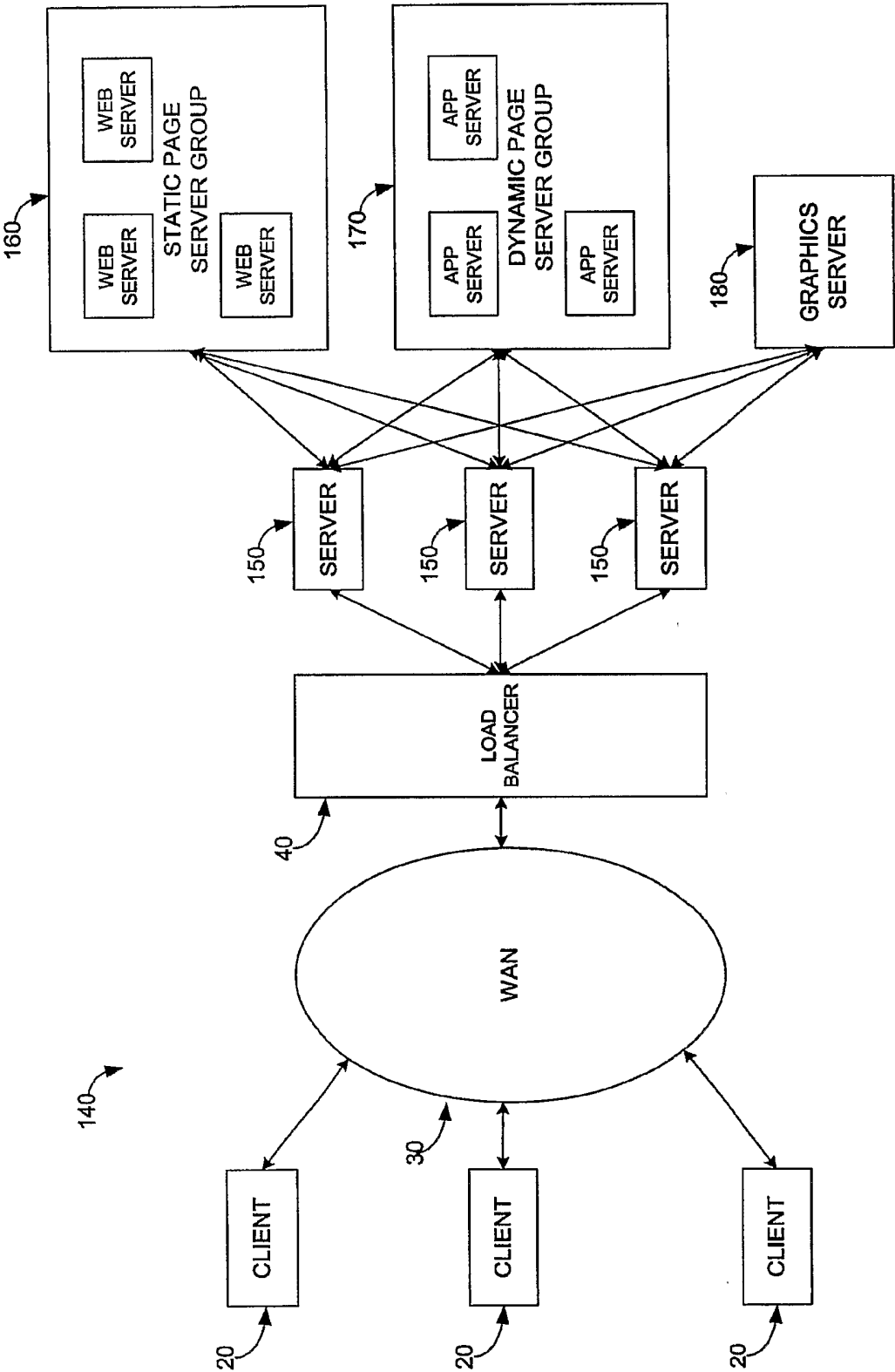


FIGURE 3

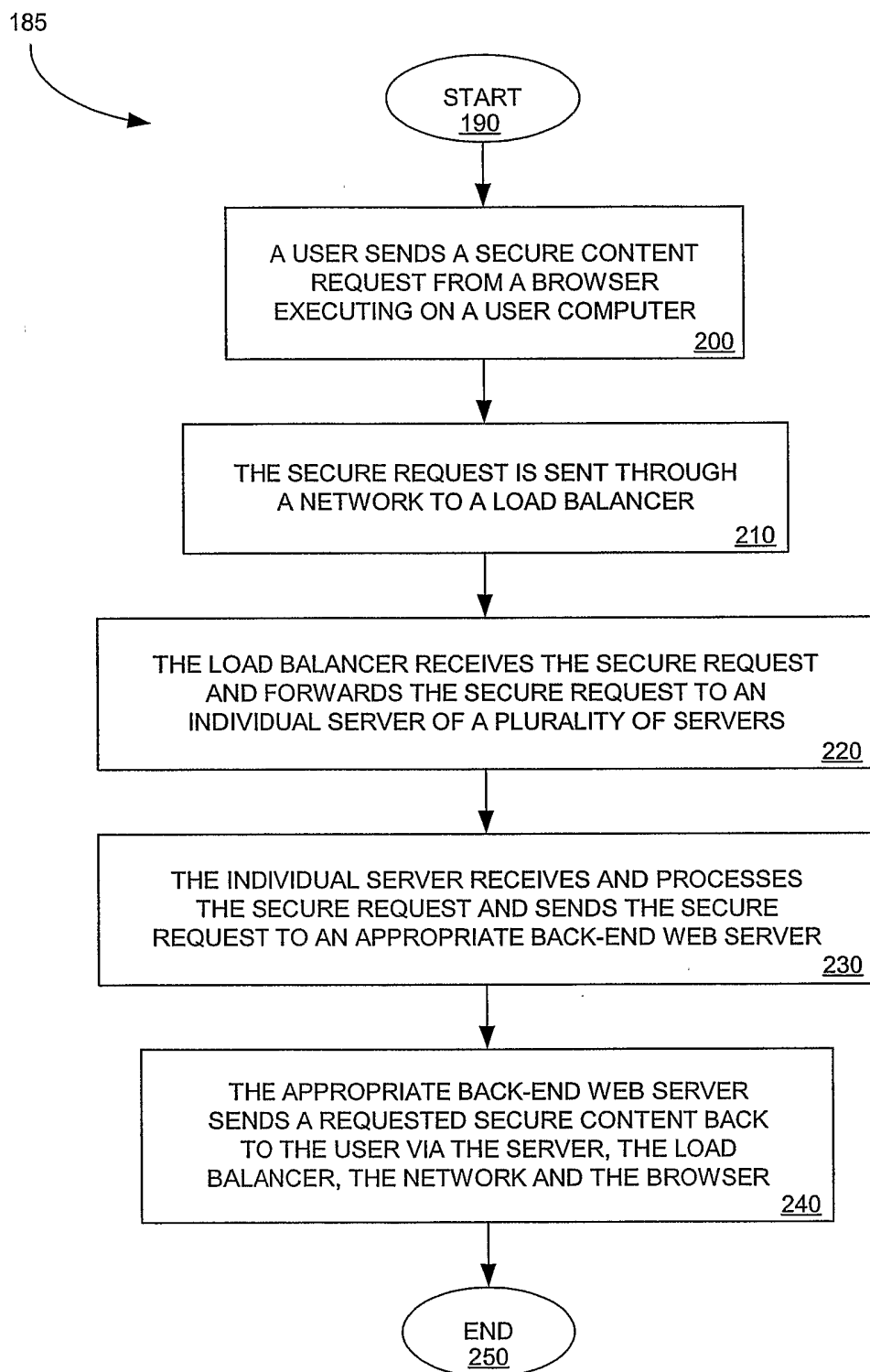


FIG. 4

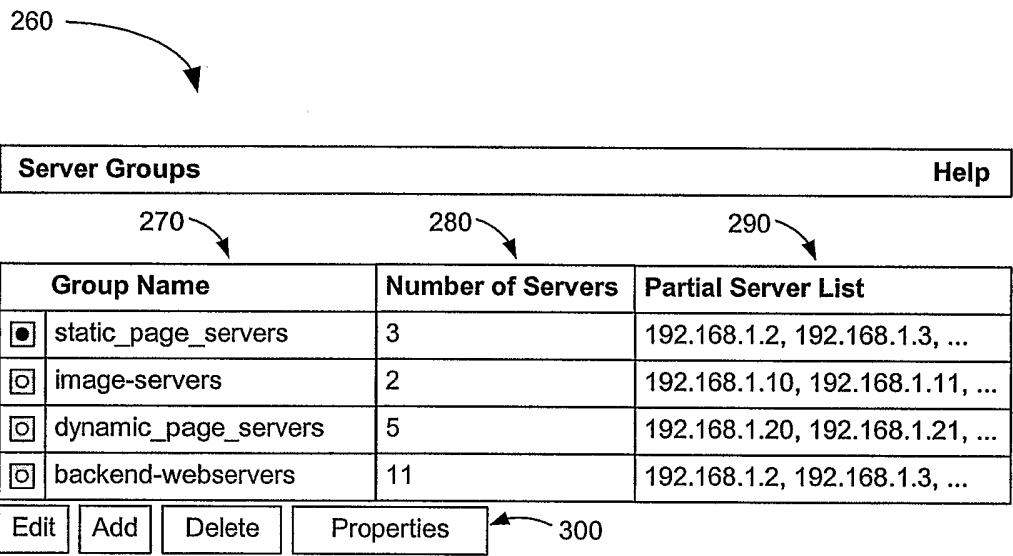


FIG. 5

310

### Destination Server Group Properties

| Server Group Properties        |                           | Help |
|--------------------------------|---------------------------|------|
| Group Name:                    | image_servers             |      |
| Server Affinity Cookie Domain: | ingrian.com               |      |
| Load Balancing Algorithm:      | Random                    |      |
| Edit                           | Load Balancing Properties | 322  |

320

330

### Server List

| Server List                                      |                                                     | Help |
|--------------------------------------------------|-----------------------------------------------------|------|
| Server IP                                        | Server Port                                         |      |
| <input type="checkbox"/> 192.168.1.10            | Port specified in forwarding rule or switching rule |      |
| <input checked="" type="checkbox"/> 192.168.1.20 | Port specified in forwarding rule or switching rule |      |

Deleting a server will restart any forwarding rules that use this server group

Edit Add Delete

350 400

340

### Destination Server Active Monitoring

| Destination Server Active Monitoring              |                                     | Help |
|---------------------------------------------------|-------------------------------------|------|
| Enable Active Monitoring:                         | <input checked="" type="checkbox"/> |      |
| Active Monitoring Method:                         | ingrian.com                         |      |
| # of Failures Required to Mark a Server Inactive: | 3                                   |      |
| Monitoring Interval for Active Servers (sec):     | 5                                   |      |
| Monitoring Interval for Inactive Servers (sec):   | 30                                  |      |
| URL Path for HTTP Requests:                       | [None]                              |      |
| Expected HTTP Response Codes:                     | [Any]                               |      |
| Edit                                              |                                     |      |

342

### HTTP GET Response Body Rules

| Criterion                           | Text                       | Case Sensitive | Active Monitoring Result |
|-------------------------------------|----------------------------|----------------|--------------------------|
| <input checked="" type="checkbox"/> | Default if no rule matches |                | Success                  |

Edit Add

FIG. 6

360

**Destination Server Affinity Properties** **Help**

370

|                                          |                                       |
|------------------------------------------|---------------------------------------|
| <b>Client-Server Affinity Algorithm:</b> | None                                  |
| <b>Affinity Period (sec):</b>            | None<br>Client IP<br>Cookie Insertion |

380

Save Cancel

FIG. 7A

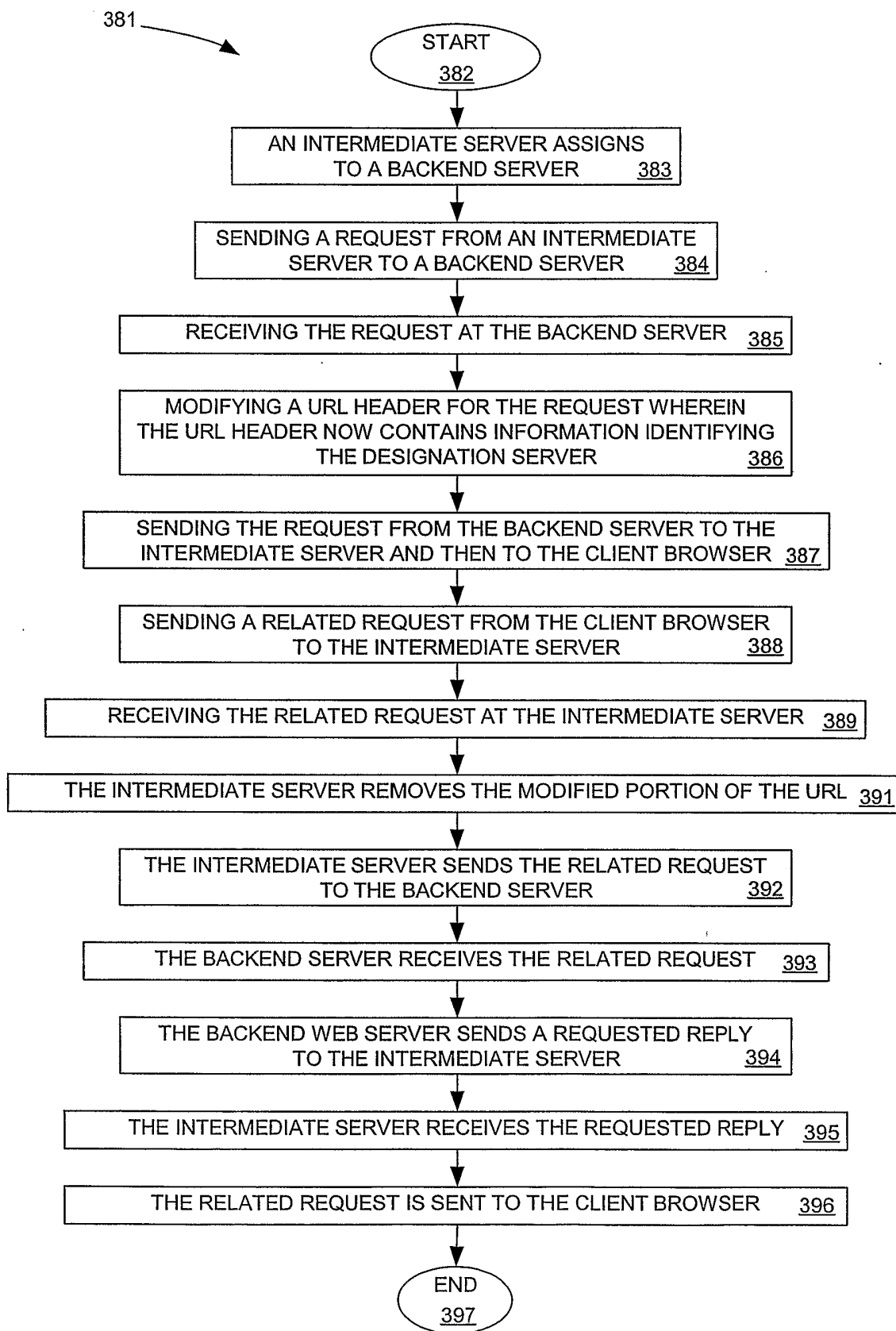


FIG. 7B

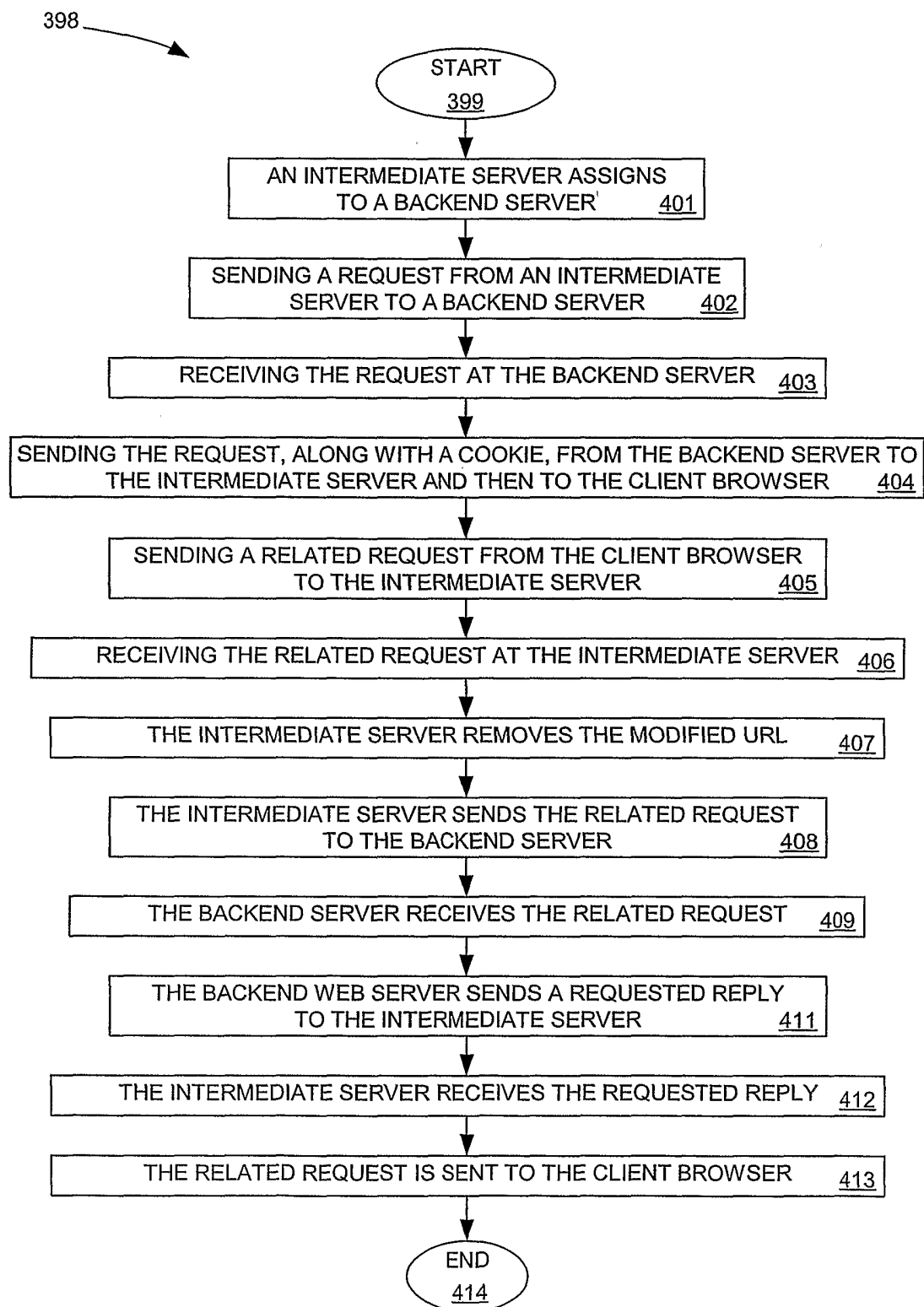


FIG. 7C

390

| Switching Rules                                                                                                           |                          |                                                                           |                                   |                                                                                                                                                              |                                 |                                     | Help |
|---------------------------------------------------------------------------------------------------------------------------|--------------------------|---------------------------------------------------------------------------|-----------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------|-------------------------------------|------|
| 410                                                                                                                       |                          | 420                                                                       | 430                               | 440                                                                                                                                                          | 450                             | 460                                 |      |
| Matches                                                                                                                   | Case Sensitive           | Criterion                                                                 | Text                              | Dest IP or Server Group                                                                                                                                      | Dest Port                       | Dest Protocol                       |      |
| <input checked="" type="checkbox"/> <input type="checkbox"/> URL<br><input type="checkbox"/> Cookie: <input type="text"/> | <input type="checkbox"/> | <input type="checkbox"/> Does not<br><input type="checkbox"/> Ends with ▼ | <input type="text" value=".gif"/> | <input type="checkbox"/> Dest IP <input type="text"/><br><input checked="" type="checkbox"/> Dest Server Group: <input type="text" value="image-servers"/> ▼ | <input type="text" value="80"/> | <input type="text" value="HTTP"/> ▼ |      |
| <input type="checkbox"/> URL                                                                                              | <input type="checkbox"/> | Ends with                                                                 | .jpg                              | Group: image-servers                                                                                                                                         | 80                              | HTTP                                |      |
| <input type="checkbox"/> URL                                                                                              | <input type="checkbox"/> | Ends with                                                                 | .jsp                              | Group: dynamic_page_servers                                                                                                                                  | 80                              | HTTP                                |      |
| <input type="checkbox"/> URL                                                                                              | <input type="checkbox"/> | Ends with                                                                 | .asp                              | Group: dynamic_page_servers                                                                                                                                  | 80                              | HTTP                                |      |
| Default specified in Forwarding Rule Properties above                                                                     |                          |                                                                           |                                   | Group: backend-webservers                                                                                                                                    | 80                              | HTTP                                |      |

FIG. 8

340 →

|                                      |                                                   |                                     |
|--------------------------------------|---------------------------------------------------|-------------------------------------|
| Destination Server Active Monitoring |                                                   | Help                                |
| 342 →                                | Enable Active Monitoring:                         | <input checked="" type="checkbox"/> |
| 344 →                                | Active Monitoring Method:                         | Establish TCP Connection            |
| 346 →                                | # of Failures Required to Mark a Server Inactive: | 3                                   |
| 348 →                                | Monitoring Interval for Active Servers (sec):     | 5                                   |
| 352 →                                | Monitoring Interval for Inactive Servers (sec):   | 30                                  |
| 354 →                                | URL Path for HTTP Requests:                       | [None]                              |
| 356 →                                | Expected HTTP Response Codes:                     | [Any]                               |
| 358 →                                | <div>Edit</div>                                   |                                     |

HTTP GET Response Body Rules

362 →

| Criterion                           | Text                       | Case Sensitive | Active Monitoring Result |
|-------------------------------------|----------------------------|----------------|--------------------------|
| <input checked="" type="checkbox"/> | Default if no rule matches |                | Success                  |

Edit

Add

364      366

FIG. 9A

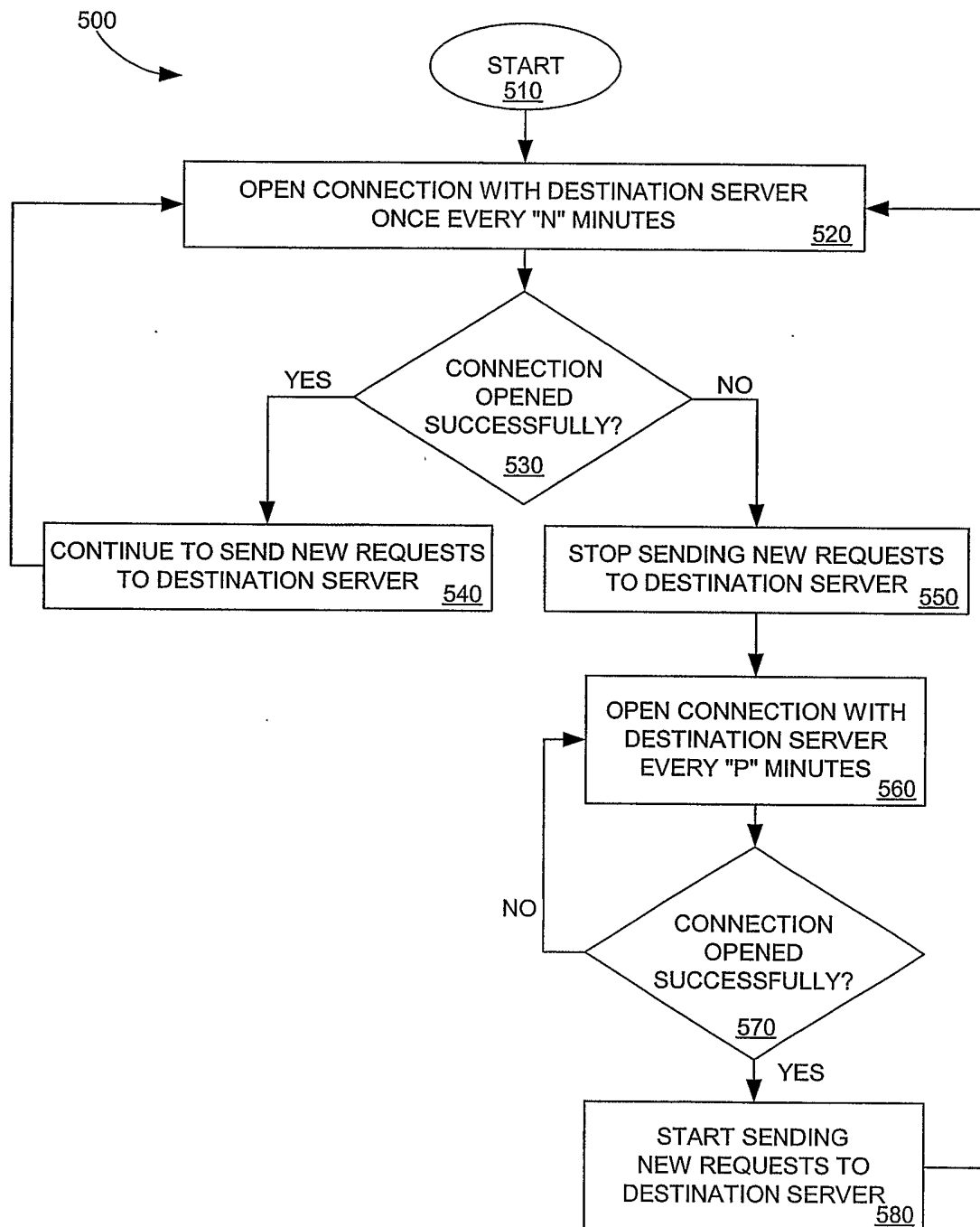


FIG. 9B

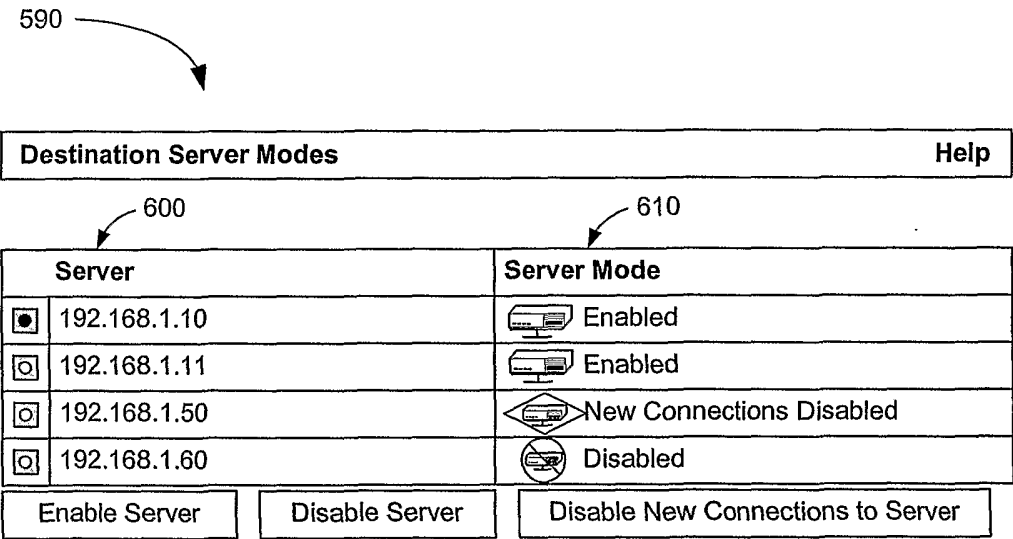


FIG. 10

620

Choose a Forwarding Rule

Help

|     | 640<br>Forwarding Rule                                 | 650<br>Local IP | 660<br>Local Port | Local Protocol |
|-----|--------------------------------------------------------|-----------------|-------------------|----------------|
| 630 | <input checked="" type="checkbox"/> Forwarding Rule #1 | [All]           | 80                | HTTP           |
|     | <input type="checkbox"/> Forwarding Rule #2            | [All]           | 90                | HTTP           |
|     | <input type="checkbox"/> Forwarding Rule #3            | [All]           | 100               | HTTP           |

Show Server Statistics

Show Server Statistics for All Forwarding Rules

670 680

FIG. 11

690

**Destination Server Statistics: All Forwarding Rules**

|                                                                                                   |  |                                                                                              |  |             |  |
|---------------------------------------------------------------------------------------------------|--|----------------------------------------------------------------------------------------------|--|-------------|--|
| <b>Refresh Statistics</b>                                                                         |  |                                                                                              |  | <b>Help</b> |  |
| Refresh Every:                                                                                    |  | <div style="border: 1px solid black; padding: 2px; display: inline-block;"> Never ▼ </div>   |  |             |  |
| <div style="border: 1px solid black; padding: 2px; display: inline-block;">Set Refresh Time</div> |  | <div style="border: 1px solid black; padding: 2px; display: inline-block;">Refresh Now</div> |  |             |  |

| <b>Destination Server Statistics</b>                                                       |                               |                      |                    |                   | <b>Help</b> |  |
|--------------------------------------------------------------------------------------------|-------------------------------|----------------------|--------------------|-------------------|-------------|--|
| Server                                                                                     | Status                        | Elapsed Time Up/Down | Active Connections | Total Connections |             |  |
| 700 <b>Forwarding Rule #1: Local IP: [All] Port 443 HTTPS -- Server Group: backend_app</b> |                               |                      |                    |                   |             |  |
| 192.168.1.30.443 (HTTPS)                                                                   | ↑ Active                      | 20:51:20             | 4                  | 11                |             |  |
| 192.168.1.31.443 (HTTPS)                                                                   | ○ Inactive (Not Enabled)      | 00:00:45             | 0                  | 0                 |             |  |
| 710 <b>Forwarding Rule #2: Local IP: [All] Port 80 HTTP -- Server Group: backend_web</b>   |                               |                      |                    |                   |             |  |
| 750 <b>Switching Rule #1: URL Ends-with ".gif" -- Server Group: image_servers</b>          |                               |                      |                    |                   |             |  |
| 770 192.168.1.60.80 (HTTP)                                                                 | ↑ Active                      | 00:45:04             | 11                 | 68                |             |  |
| 780 192.168.1.61.80 (HTTP)                                                                 | ○ Inactive (Unreachable)      | 00:01:25             | 0                  | 11                |             |  |
| 760 <b>Switching Rule #2: URL Ends-with ".jpg" -- Server Group: image_servers</b>          |                               |                      |                    |                   |             |  |
| 770 192.168.1.60.80 (HTTP)                                                                 | ↑ Active                      | 00:45:04             | 2                  | 19                |             |  |
| 780 192.168.1.61.80 (HTTP)                                                                 | ○ Inactive (Unreachable)      | 00:01:25             | 0                  | 17                |             |  |
| <b>Default Destination: Server Group: backend_web</b>                                      |                               |                      |                    |                   |             |  |
| 770 192.168.1.10.80 (HTTP)                                                                 | ↑ Active                      | 1 day, 20:06:33      | 10                 | 80                |             |  |
| 192.168.1.11.80 (HTTP)                                                                     | ↑ Active (No New Connections) | 00:05:36             | 2                  | 19                |             |  |
| 192.168.1.12.80 (HTTP)                                                                     | ↑ Active                      | 00:12:27             | 0                  | 0                 |             |  |
| 720 <b>Forwarding Rule #3: Local IP: [All] Port 8080 HTTP</b>                              |                               |                      |                    |                   |             |  |
| 770 192.168.1.10.80 (HTTP)                                                                 | ↑ Active                      | 1 day, 20:06:33      | 12                 | 65                |             |  |
| 740 <b>Forwarding Rule #4: Local IP: [All] Port 81 HTTP</b>                                |                               |                      |                    |                   |             |  |
| 192.168.1.10.81 (HTTP)                                                                     | ↑ Active                      | 1 day, 20:06:33      | 195                | 411               |             |  |

Back

FIG. 12

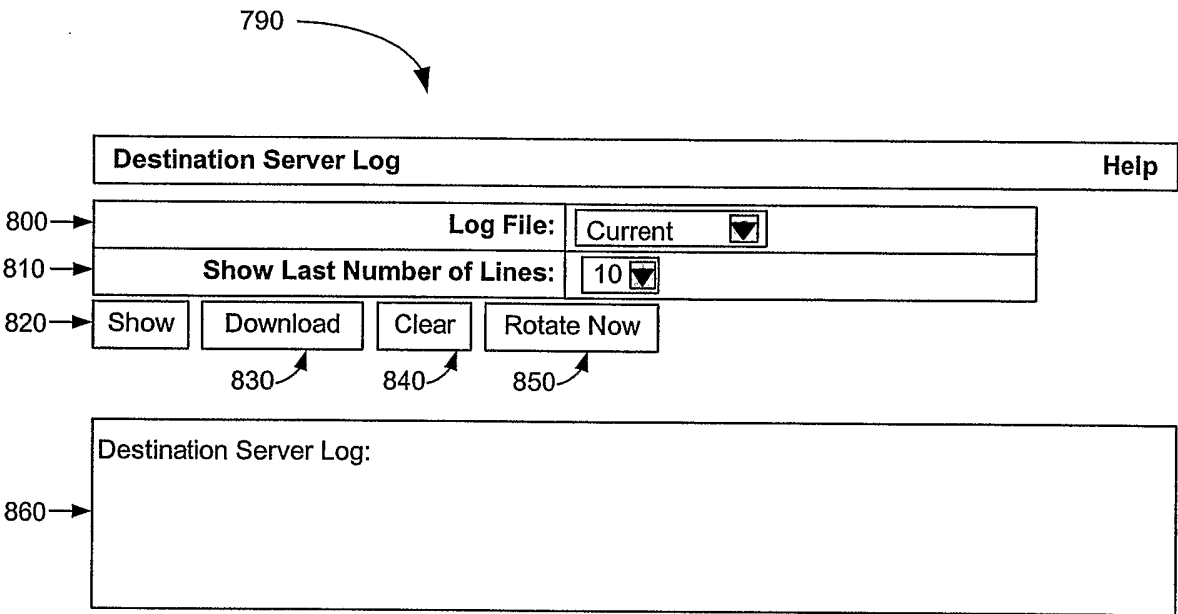


FIG. 13

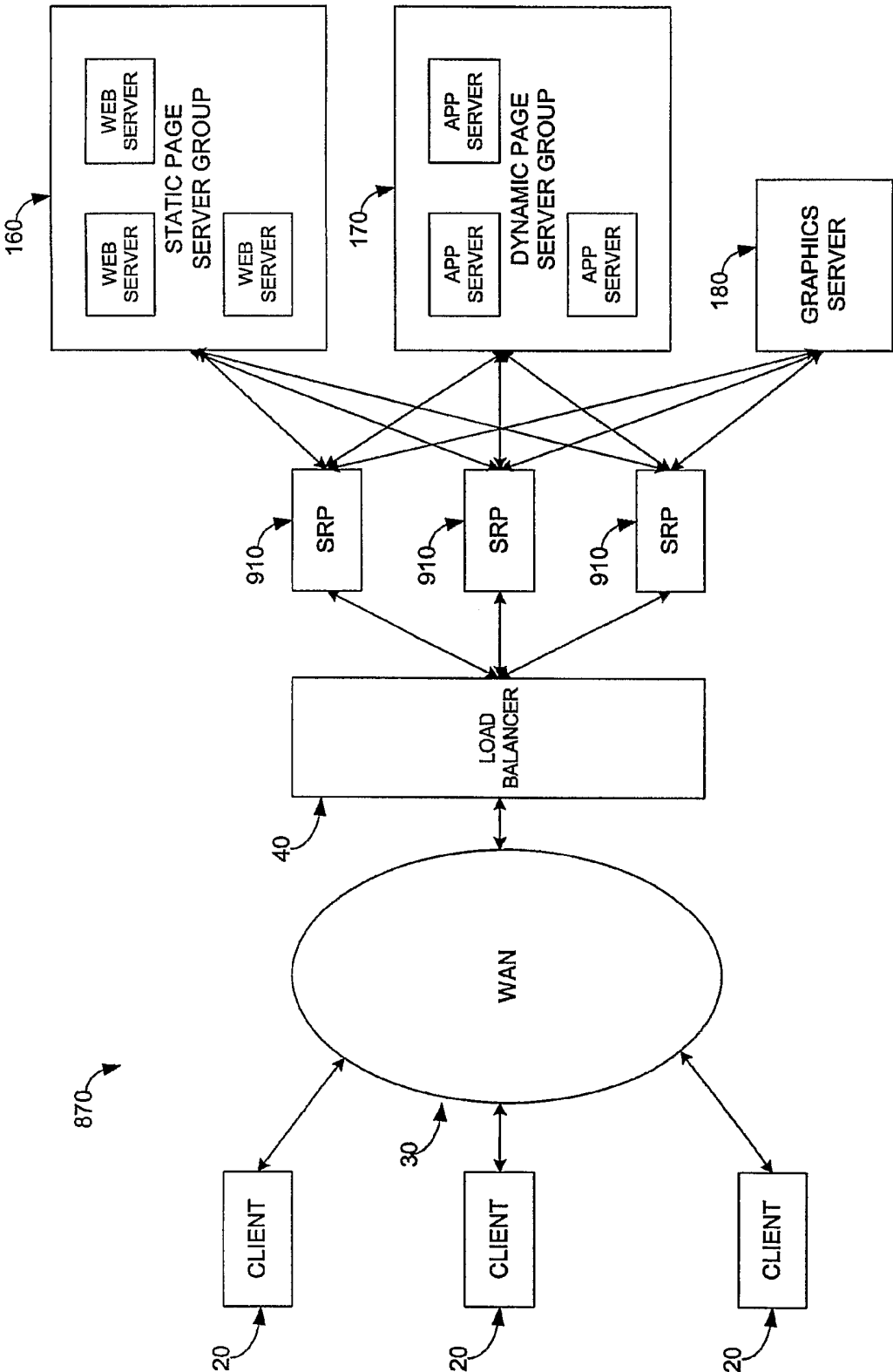


FIGURE 14

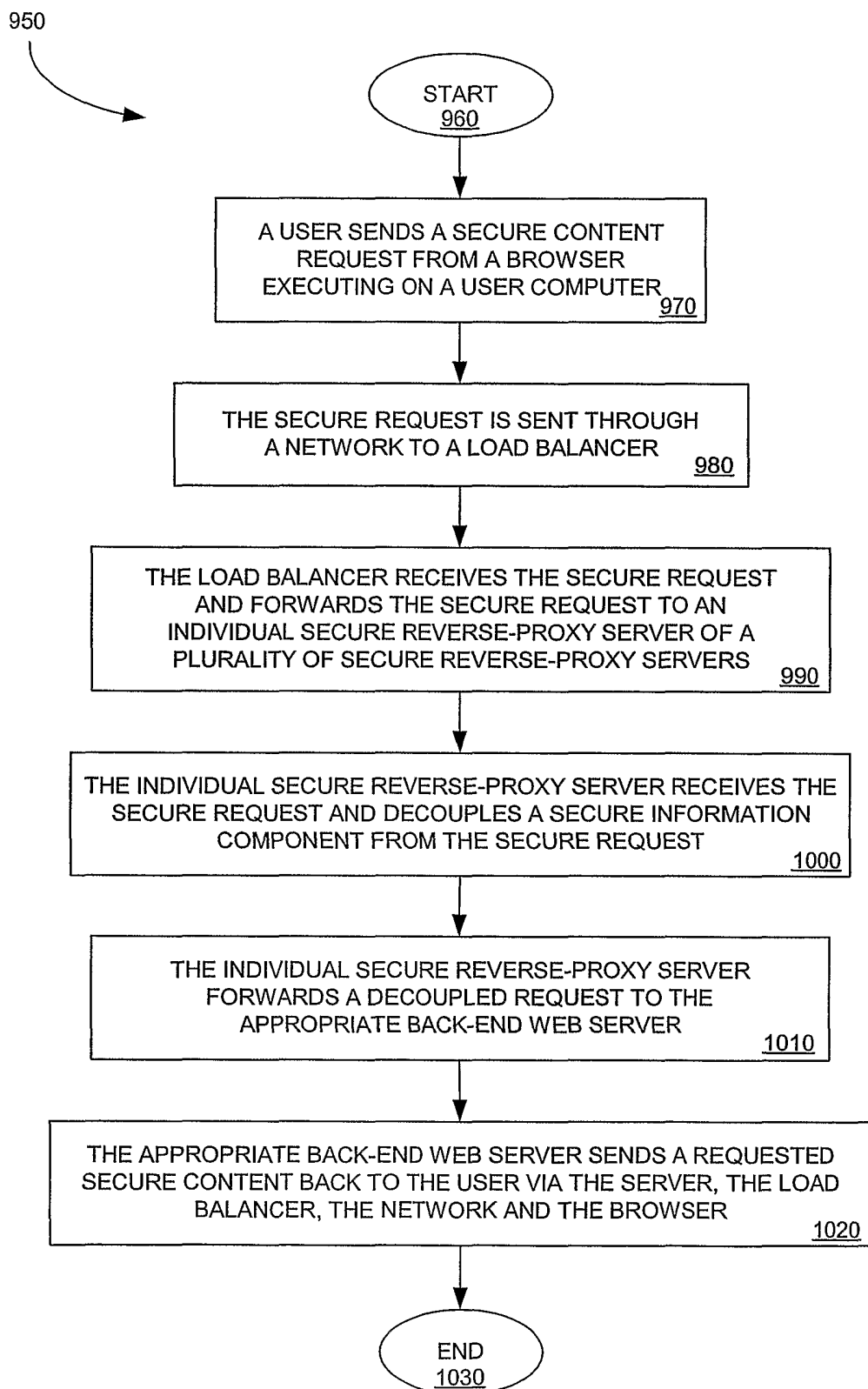


FIG. 15

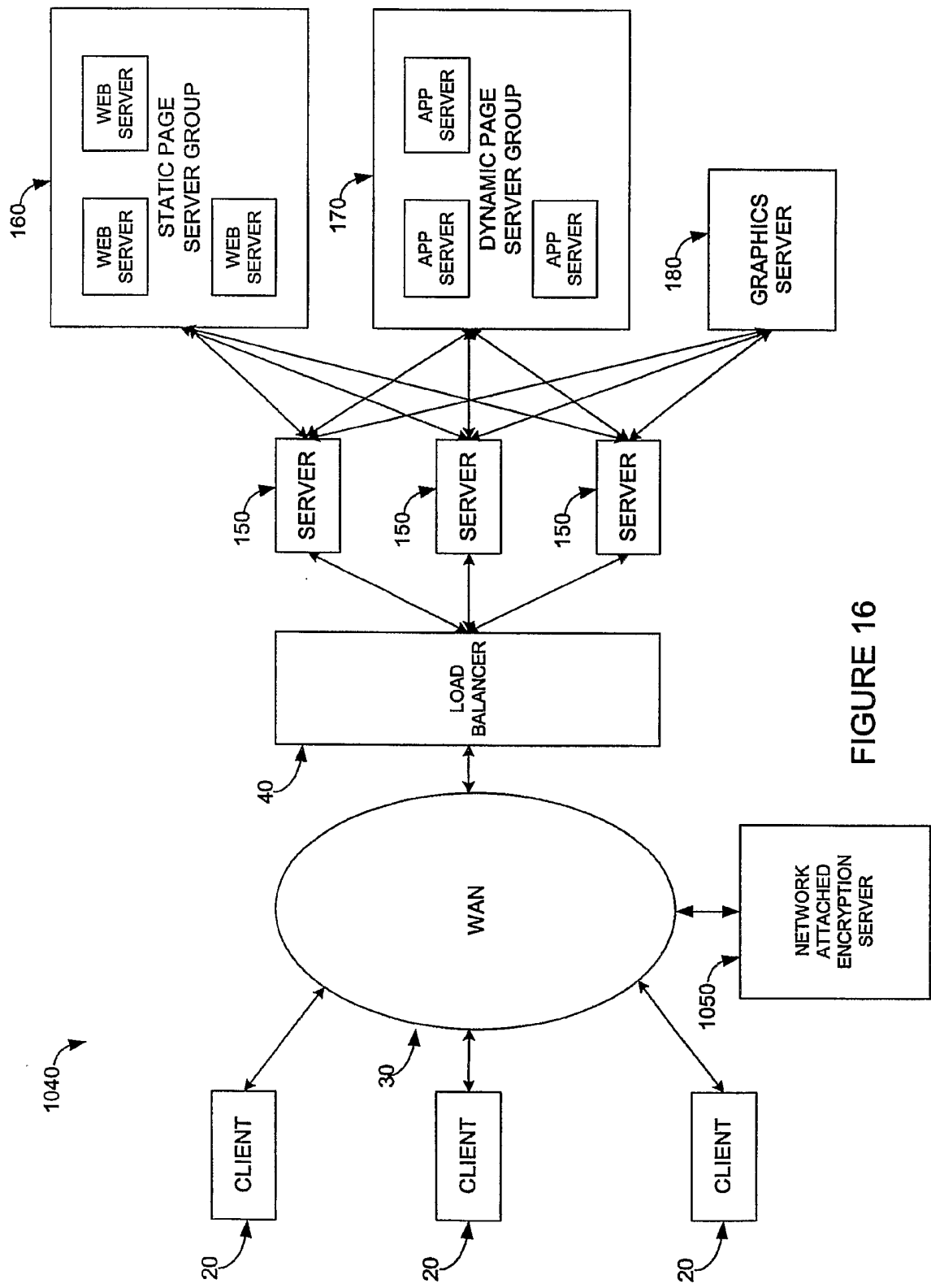


FIGURE 16

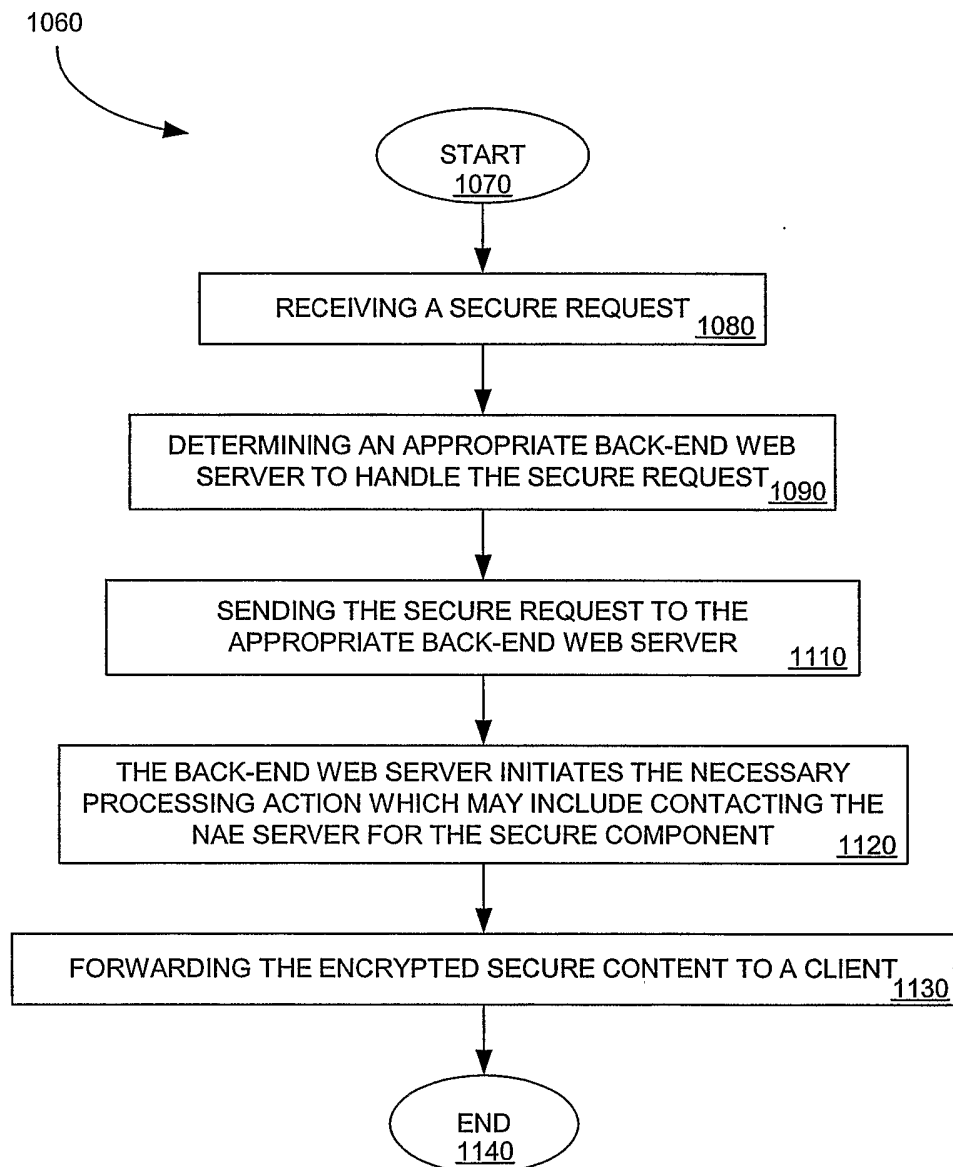


FIG. 17