(12) STANDARD PATENT (11) Application No. **AU 2013312982 B2**
(19) AUSTRALIAN PATENT OFFICE

(54) Title
Physical security system having multiple server nodes

(51) International Patent Classification(s)
H04L 12/16 (2006.01) ***H04L 12/24*** (2006.01)
G08B 13/00 (2006.01) ***H04N 7/18*** (2006.01)

(21) Application No: **2013312982** (22) Date of Filing: **2013.09.06**

(87) WIPO No: **WO14/036656**

(30) Priority Data

(31) Number	(32) Date	(33) Country
13/607,447	**2012.09.07**	**US**

(43) Publication Date: **2014.03.13**
(44) Accepted Journal Date: **2017.11.16**

(71) Applicant(s)
Avigilon Corporation

(72) Inventor(s)
Lee, Ryan;Marlatt, Shaun;Adam, Matthew;Wightman, Ross;Magolan, Greg;Martz, Andrew

(74) Agent / Attorney
Davies Collison Cave Pty Ltd, GPO Box 3876, SYDNEY, NSW, 2001, AU

(56) Related Art
GB 2368683 A

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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
13 March 2014 (13.03.2014)

WIPO | PCT

(10) International Publication Number
WO 2014/036656 A1

(51) International Patent Classification:
H04L 12/16 (2006.01) *H04L 12/24* (2006.01)
G08B 13/00 (2006.01) *H04N 7/18* (2006.01)

(21) International Application Number:
PCT/CA2013/050690

(22) International Filing Date:
6 September 2013 (06.09.2013)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
13/607,447 7 September 2012 (07.09.2012) US

(71) Applicant: **AVIGILON CORPORATION** [CA/CA]; 4th Floor - 858 Beatty Street, Vancouver, British Columbia V6B 1C1 (CA).

(72) Inventors: **LEE, Ryan**; 4th Floor - 858 Beatty Street, Vancouver, British Columbia V6B 1C1 (CA). **MARLATT, Shaun**; 4th Floor - 858 Beatty Street, Vancouver, British Columbia V6B 1C1 (CA). **ADAM, Matthew**; 4th Floor - 858 Beatty Street, Vancouver, British Columbia V6B 1C1 (CA). **WIGHTMAN, Ross**; 4th Floor - 858 Beatty Street, Vancouver, British Columbia V6B 1C1 (CA). **MAGOLAN, Greg**; 4th Floor - 858 Beatty Street, Vancouver, British Columbia V6B 1C1 (CA). **MARTZ, Andrew**; 4th Floor - 858 Beatty Street, Vancouver, British Columbia V6B 1C1 (CA).

(74) Agents: **RIPLEY, Roch** et al.; 550 Burrard Street, Suite 2300, Vancouver, British Columbia V6C 2B5 (CA).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Published:

— *with international search report (Art. 21(3))*

(54) Title: PHYSICAL SECURITY SYSTEM HAVING MULTIPLE SERVER NODES

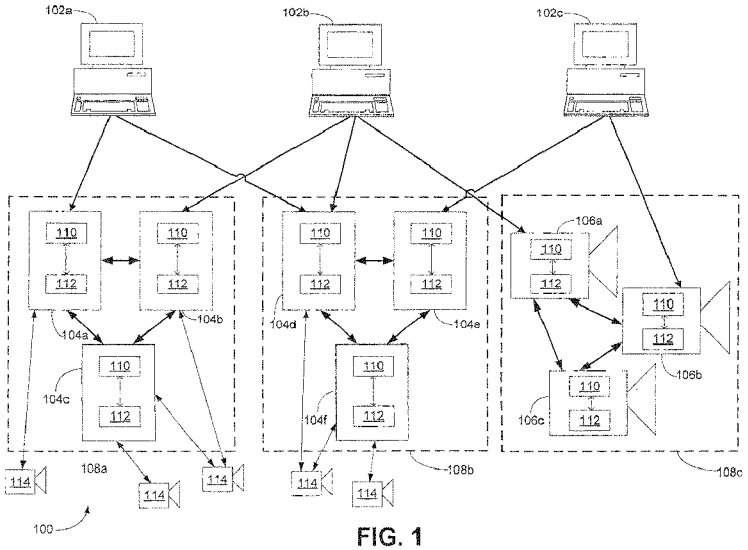

FIG. 1

(57) Abstract: A physical security system having multiple server nodes may be built as a distributed network. To send data between the nodes in the network, a first node may access a node identifier identifying a second node, with both the first and second nodes forming at least part of a server cluster, and the first node may then send the data to the second node. The node identifier forms at least part of cluster membership information identifying all and accessible by all server nodes in that server cluster. Functionality such as the ability to share views between system users and the ability for those users to control an unattended display may be implemented on a distributed network, a federated network, or another type of network.

WO 2014/036656 A1

2013312982 17 May 2017

PHYSICAL SECURITY SYSTEM HAVING MULTIPLE SERVER NODES

CROSS-REFERENCE TO RELATED APPLICATION

[0001] This application claims priority from US Patent Application 13/607,447 filed on September 7, 2012, the entirety of which is incorporated by reference herein.

TECHNICAL FIELD

[0002] The present disclosure is directed at a physical security system having multiple server nodes.

BACKGROUND

[0003] A physical security system is a system that implements measures to prevent unauthorized persons from gaining physical access to an asset, such as a building, a facility, or confidential information. Examples of physical security systems include surveillance systems, such as a system in which cameras are used to monitor the asset and those in proximity to it; access control systems, such as a system that uses RFID cards to control access to a building; intrusion detection systems, such as a home burglary alarm system; and combinations of the foregoing systems.

[0004] A physical security system often incorporates computers. As this type of physical security system grows, the computing power required to operate the system increases. For example, as the number of cameras in a surveillance system increases, the requisite amount of computing power also increases to allow additional video to be stored and to allow simultaneous use and management of a higher number of cameras. Research and development accordingly continue into overcoming problems encountered as a physical security system grows.

SUMMARY

[0005] According to a first aspect, there is provided a method for sharing data in a physical security system that comprises a plurality of server nodes. The method comprises accessing, using one of the server nodes ("first node"), a node identifier identifying another of the server nodes ("second node"), wherein the first and second nodes comprise at least part of a server cluster and wherein the node identifier comprises at least part of cluster membership information identifying all and accessible by all server nodes in the server cluster; and sending the data from the first node to the second node.

[0006] The server cluster may comprise at least three server nodes.

[0007] The server nodes may comprise cameras, network video recorders, and access control servers.

[0008] The method may further comprise accessing, using the second node, a node identifier identifying the first node; and sending additional data from the second node to the first node.

[0009] The cluster membership information may comprise a node identifier uniquely identifying each of the server nodes in the server cluster; and a cluster identifier uniquely identifying the server cluster to which the server nodes belong.

[0010] Each of the server nodes in the server cluster may persistently store its own version of the cluster membership information locally.

[0011] The method may further comprise rebooting one of the server nodes ("rebooted server node") in the server cluster; and once the rebooted server node returns online, using the rebooted server node to perform a method comprising (i) accessing the cluster identifier identifying the server cluster; and (ii) automatically rejoining the server cluster.

[0012] The method may further comprise adding a new server node to the server cluster by performing a method comprising exchanging a version of the cluster membership information stored on the new server node with the version of the cluster membership information stored on one of the server nodes that is already part of the server cluster ("membership control node"); and synchronizing the versions of the cluster membership information stored on the new server node with the versions of the cluster membership information stored on all the server nodes in the cluster prior to the new server node joining the cluster.

[0013] Sending the data may comprise looking up, using the first node, a communication endpoint for the second node from the node identifier; and sending the data from the first node to the communication endpoint.

[0014] The communication endpoint and the node identifier may comprise entries in a network map relating node identifiers for all the server nodes in the server cluster to corresponding communication endpoints, and each of the server nodes in the server cluster may persistently store its own version of the network map locally.

[0015] The network map may permit each of the server nodes in the server cluster to send the data to any other of the server nodes in the server cluster without using a centralized server.

[0016] The data may be stored locally on the first node and the method may further comprise modifying the data using the first node, wherein sending the data from the first node to the second node comprises part of synchronizing the data on the first and second nodes after the first node has modified the data.

[0017] The data may comprise version information generated using a causality versioning mechanism and different versions of the data may be stored on the first and second nodes, and synchronizing the data may comprise comparing the version information stored on the first and second nodes and adopting on both of the first and second nodes the data whose version information indicates is more recent.

[0018] The data may comprise the node identifier of the first node, heartbeat state information of the first node, application state information of the first node, and version information, and sending the data may comprise disseminating the data to all the server nodes in the server cluster using a gossip protocol that performs data exchanges between pairs of the server nodes in the cluster.

[0019] The data may be periodically disseminated to all the server nodes in the server cluster.

[0020] The data may be sent to the second node when the first node joins the cluster.

[0021] A domain populated with entries that can be modified by any of the server nodes in the server cluster may be stored locally on each of the nodes in the server cluster, and the method may further comprise generating the version information using a causality versioning mechanism such that the version information indicates which of the server nodes has most recently modified one of the entries.

[0022] The application state information may comprise a top-level hash generated by hashing all the entries in the domain.

[0023] The method may further comprise comparing, using the second node, the top-level hash with a top-level hash generated by hashing a version of a corresponding domain stored locally on the second node; and if the top-level hashes differ, synchronizing the domains on both the first and second nodes using the version information.

[0024] A status entry that can only be modified by the first node may be stored locally on the first node, and the version information may comprise a version number that the first node increments whenever it modifies the status entry.

[0025] The application state information may comprise a status entity pair comprising a status entity identifier that identifies the status entry and the version number.

[0026] The method may further comprise comparing, using the second node, the version number received from the first node with a version number of a corresponding status entry stored locally on the second node; and if the versions numbers differ, updating the status entry stored locally on the second node with the status entry stored locally on the first node.

[0027] Updating the status entry may comprise sending from the first node to the second node additional status entries stored locally on the first node that were modified simultaneously with the status entry.

[0028] The first and second nodes may comprise at least part of a group of server nodes in the cluster to which the first node can send the data in a totally ordered manner to all of the server nodes in the group, and sending the data may comprise the first node sending the data to all of the server nodes in the group.

[0029] The data may comprise non-persistent data generated during the runtime of the physical security system.

[0030] The data may also comprise streaming video streamed from another of the server nodes in the server cluster through the first node to the second node.

[0031] According to another aspect, there is provided a system for sharing data in a physical security system, the system comprising a plurality of server nodes comprising a first node and a second node, wherein the first node comprises a processor communicatively coupled to a computer readable medium that has encoded thereon statements and instructions to cause the processor to perform a method comprising accessing a node identifier identifying the second node, wherein the first and second nodes comprise at least part of a server cluster and wherein the node identifier comprises

at least part of cluster membership information identifying all and accessible by all the server nodes in the server cluster; and sending the data to the second node.

[0032] According to another aspect, there is provided a non-transitory computer readable medium having encoded thereon statements and instructions to cause a processor to perform a method for sharing data in a physical security system that comprises a plurality of server nodes, the method comprising accessing, using one of the server nodes ("first node"), a node identifier identifying another of the server nodes ("second node"), wherein the first and second nodes comprise at least part of a server cluster and wherein the node identifier comprises at least part of cluster membership information identifying all and accessible by all server nodes in the server cluster; and sending the data from the first node to the second node.

[0033] According to another aspect, there is provided a method for interacting with a unattended display in a physical security system that comprises a plurality of server nodes, the method comprising sending, from one of the server nodes ("second node") communicative with the unattended display to another of the server nodes ("first node") that is communicative with a client display, view state data indicative of a view displayed on the unattended display; and displaying, on the client display, at least a portion of the view displayed on the unattended display. In one aspect, none of the server nodes is a centralized gateway server; in an alternative aspect, at least one of the server nodes is a centralized gateway server.

[0034] The method may further comprise sending, from the first node to the second node, a message to change the view of the unattended display; and updating the unattended display according to the message sent from the first node to the second node.

[0035] The first and second nodes and at least another of the plurality of server nodes may comprise a server cluster, the first and second nodes may comprise at least part of a group of server nodes in the cluster to which the second node can send the view state data in a totally ordered manner to all other server nodes in the group, and sending

the view state data may comprise the second node sending the data to all the other server nodes in the group.

[0036] The first and second nodes and at least another of the plurality of server nodes may comprise a server cluster, the first and second nodes may comprise at least part of a group of server nodes in the cluster to which the first node can send the message to change the state of the unattended display in a totally ordered manner to all other server nodes in the group, and the first node may send the message to change the state of the unattended display to all the other server nodes in the group.

[0037] The method may further comprise sending from the second node to the first node a notification that the view of the unattended display is available to be controlled.

[0038] Sending the notification may comprise disseminating the notification to all the server nodes in the server cluster using a gossip protocol that performs data exchanges between pairs of the server nodes in the cluster.

[0039] Prior to sending the state of the unattended display to the controlling display, the method may comprise accessing, using the second node, a node identifier identifying the first node, wherein the first and second nodes comprise at least part of a server cluster and wherein the node identifier comprises at least part of cluster membership information identifying all and accessible by all server nodes in the server cluster.

[0040] The cluster membership information may comprise a node identifier uniquely identifying each of the server nodes in the server cluster; and a cluster identifier uniquely identifying the server cluster to which the server nodes belong.

[0041] Each of the server nodes in the server cluster may persistently store its own version of the cluster membership information locally.

[0042] According to another aspect, there is provided a physical security system, comprising: a client display; a unattended display; and a plurality of server nodes, wherein one of the server nodes ("first node") is communicative with the client display and another of the server nodes ("second node") is communicative with the unattended display, wherein the second node is configured to send to the first node view state data indicative of a view displayed on the second display and the first node is configured to display, on the client display, at least a portion of the view displayed on the second display. In one aspect, none of the server nodes is a centralized gateway server; in an alternative aspect, at least one of the server nodes is a centralized gateway server.

[0043] According to another aspect, there is provided a physical security system, comprising: a client having a client display; a unattended display; and a plurality of server nodes, wherein one of the server nodes ("first node") is communicative with the client and another of the server nodes ("second node") is communicative with the unattended display, wherein the second node is configured to send to the first node view state data indicative of a view displayed on the second display and the client and first node are configured to display, on the client display, at least a portion of the view displayed on the second display. In one aspect, none of the server nodes is a centralized gateway server; in an alternative aspect, at least one of the server nodes is a centralized gateway server.

[0044] The unattended display may be directly connected to the second node or indirectly connected to the second node via, for example, an unattended client or workstation.

[0045] According to another aspect, there is provided a non-transitory computer readable medium having encoded thereon statements and instructions to cause a processor to perform a method for interacting with a unattended display in a physical security system that comprises a plurality of server nodes, the method comprising sending, from one of the server nodes ("second node") communicative with the unattended display to another of the server nodes ("first node") that is communicative

with a client display, view state data indicative of a view displayed on the unattended display; and displaying, on the client display, at least a portion of the view displayed on the unattended display.

[0046] According to another aspect, there is provided a method for sharing a view ("shared view") using a physical security system that comprises a plurality of server nodes, the method comprising: sending, from a first client to one of the server nodes ("first node"), view state data representative of the shared view as displayed by the first client; sending the view state data from the first node to a second client via another of the server nodes ("second node"); updating a display of the second client using the view state data to show the shared view; in response to a change in the shared view at the second client, sending updated view state data from the second client to the second node, wherein the updated view state data is representative of the shared view as displayed by the second client; sending the updated view state data from the second node to the first client via the first node; and updating the display of the first client to show the shared view using the updated view state data. In one aspect, none of the server nodes is a centralized gateway server; in an alternative aspect, at least one of the nodes is a centralized gateway server.

[0047] The first and second nodes and at least another of the plurality of server nodes may comprise a server cluster, the first and second nodes may comprise at least part of a group of server nodes in the cluster to which the first node can send the view state data in a totally ordered manner to all other server nodes in the group, and sending the view state data may comprise the first node sending the data to all the other server nodes in the group.

[0048] The first and second nodes and at least another of the plurality of server nodes may comprise a server cluster, the first and second nodes may comprise at least part of a group of server nodes in the cluster to which the second node can send the updated view state data in a totally ordered manner to all other server nodes in the group,

and sending the updated view state data may comprise the second node sending the updated view state data to all the other server nodes in the group.

[0049] Prior to showing the shared view on the display of the second client, the method may comprise sending from the first client to the second client via the first and second nodes a notification that the shared view as displayed by the first client is available to be shared with the second client.

[0050] The first and second nodes and at least another of the plurality of server nodes may comprise a server cluster, the first and second nodes may comprise at least part of a group of server nodes in the cluster to which the first node can send the notification in a totally ordered manner to all other server nodes in the group, and sending the notification may comprise the first node sending the notification to all the other server nodes in the group.

[0051] Prior to the first node sending the state data to the second client via the second node, the method may comprise accessing, using the first node, a node identifier identifying the second node, wherein the first and second nodes comprise at least part of a server cluster and wherein the node identifier comprises at least part of cluster membership information identifying all and accessible by all server nodes in the server cluster.

[0052] The cluster membership information may comprise a node identifier uniquely identifying each of the server nodes in the server cluster; and a cluster identifier uniquely identifying the server cluster to which the server nodes belong.

[0053] Each of the server nodes in the server cluster may persistently store its own version of the cluster membership information locally.

[0054] According to another aspect, there is provided a physical security system, comprising a first client having a display; a second client having a display; and a plurality of server nodes, wherein one of the server nodes ("first node") is communicative with the

first display and another of the server nodes ("second node") is communicative with the second display, wherein the first and second clients and the first and second nodes are configured to send, from the first client to the first node, view state data representative of a shared view as displayed on the display of the first client; send the view state data from the first node to the second client via the second node; update the display of the second client using the view state data to show the shared view; in response to a change in the shared view at the second client, sending updated view state data from the second client to the second node, wherein the updated view state data is representative of the shared view as displayed on the display of the second client; send the updated view state data from the second node to the first client via the first node; and update the display of the first client to show the shared view using the updated view state data. In one aspect, none of the server nodes is a centralized gateway server; in an alternative aspect, at least one of the server nodes is a gateway server.

[0055] According to another aspect, there is provided a non-transitory computer readable medium having encoded thereon statements and instructions to cause a processor to perform a method for sharing a view ("shared view") using a physical security system that comprises a plurality of server nodes, the method comprising sending, from a first client to one of the server nodes ("first node"), view state data representative of the shared view as displayed by the first client; sending the view state data from the first node to a second client via another of the server nodes ("second node"); updating a display of the second client using the view state data to show the shared view; in response to a change in the shared view at the second client, sending updated view state data from the second client to the second node, wherein the updated view state data is representative of the shared view as displayed by the second client; sending the updated view state data from the second node to the first client via the first node; and updating the display of the first client to show the shared view using the updated view state data.

2013312982 16 Oct 2017

[0055A] In one aspect there is provided a method for sharing data in a physical security system that comprises a plurality of server nodes, the method comprising:

(a) adding a first server node to a server cluster comprising a second server node by performing a method comprising:

(i) exchanging a version of cluster membership information stored on the first server node with a version of the cluster membership information stored on one of the server nodes that is already part of the server cluster; and

(ii) synchronizing the version of the cluster membership information stored on the first server node with versions of the cluster membership information stored on all the server nodes that, prior to the first server node joining the cluster, comprised part of the cluster;

(b) accessing, after the first node has been added to the server cluster and using the first node, a node identifier identifying another node of the server cluster, wherein the node identifier comprises at least part of cluster membership information identifying all the server nodes in the server cluster and accessible by all the server nodes in the server cluster; and wherein each of the server nodes in the server cluster persistently stores its own version of the cluster membership information locally; and

(c) sending the data from the first node to the other node.

[0055B] In a further aspect there is provided a physical security system, comprising a plurality of server nodes, wherein the physical security system is configured to share data by performing a method comprising:

(a) adding a first server node to a server cluster comprising a second server node, the adding comprising:

(i) exchanging a version of a cluster membership information stored on the first server node with a version of the cluster membership information stored on one of the server nodes that is already part of the server cluster; and

(ii) synchronizing the version of the cluster membership information stored on the first server node with versions of the

cluster membership information stored on all the server nodes that, prior to the first server node joining the cluster, comprised part of the cluster;

(b) accessing, after the first node has been added to the server cluster and using the first node, a node identifier identifying another node of the server cluster, wherein the node identifier comprises at least part of cluster membership information identifying all the server nodes in the server cluster and accessible by all the server nodes in the server cluster, and wherein each of the server nodes in the server cluster persistently stores its own version of the cluster membership information locally; and

(c) sending the data from the first node to the other node.

[0055C] In a further aspect there is provided a non-transitory computer readable medium having encoded thereon statements and instructions to cause a processor to perform a method for sharing data in a physical security system that comprises a plurality of server nodes, the method comprising:

(a) adding a first server node to a server cluster comprising a second server node, the adding comprising:

(i) exchanging a version of cluster membership information stored on the first server node with a version of the cluster membership information stored on one of the server nodes that is already part of the server cluster; and

(ii) synchronizing the version of the cluster membership information stored on the first server node with versions of the cluster membership information stored on all the server nodes that, prior to the first server node joining the cluster, comprised part of the cluster;

(b) accessing, after the first node has been added to the server cluster and using the first node, a node identifier identifying another node of the server cluster, wherein the node identifier comprises at least part of cluster membership information identifying all the server nodes in the server cluster and accessible by all the server nodes in the server cluster, and wherein each of the server nodes in the server cluster persistently stores its own version of the cluster membership information locally.

(c) sending the data from the first node to the other node.

[0056] This summary does not necessarily describe the entire scope of all aspects. Other aspects, features and advantages will be apparent to those of ordinary skill in the art upon review of the following description of specific embodiments.

BRIEF DESCRIPTION OF THE DRAWINGS

[0057] In the accompanying drawings, which illustrate one or more exemplary embodiments:

[0058] FIG. 1 is a block diagram of a distributed physical security system, according to one embodiment.

[0059] FIG. 2 is a block diagram of a protocol suit used by the system of FIG. 1.

[0060] FIG. 3 is a UML sequence diagram showing how the system of FIG. 1 shares settings between different system users.

[0061] FIG. 4 is a UML sequence diagram showing how the system of FIG. 1 shares a state between different system users.

[0062] FIG. 5 is a UML sequence diagram showing how the system of FIG. 1 shares a view between different system users.

[0063] FIG. 6 is a UML sequence diagram showing how the system of FIG. 1 shares streams between different system users.

[0064] FIG. 7 is a view seen by a user of the system of FIG. 1.

[0065] FIG. 8 is a method for sharing data in a physical security system, according to another embodiment.

[0066] FIG. 9 is a method for automatically rejoining a cluster, according to another embodiment.

[0067] FIG. 10 is a UML sequence diagram showing how the system of FIG. 1 shares an unattended view with a system user.

[0068] FIG. 11 is a method for interacting with a unattended display in a physical security system that comprises a plurality of server nodes, according to another embodiment.

[0069] FIG. 12 is a method for sharing a view using a physical security system that comprises a plurality of server nodes, according to another embodiment.

DETAILED DESCRIPTION

[0070] Directional terms such as "top", "bottom", "upwards", "downwards", "vertically", and "laterally" are used in the following description for the purpose of providing relative reference only, and are not intended to suggest any limitations on how any article is to be positioned during use, or to be mounted in an assembly or relative to an environment. Additionally, the term "couple" and variants of it such as "coupled", "couples", and "coupling" as used in this description is intended to include indirect and direct connections unless otherwise indicated. For example, if a first device is coupled to a second device, that coupling may be through a direct connection or through an indirect connection via other devices and connections. Similarly, if the first device is communicatively coupled to the second device, communication may be through a direct connection or through an indirect connection via other devices and connections.

[0071] Once a surveillance system grows to include a certain number of cameras, it becomes impractical or impossible to operate the surveillance system using a single server because of storage capacity and processing power limitations. Accordingly, to accommodate the increased number of cameras, additional servers are added to the system. This results in a number of problems.

[0072] For example, a user of the surveillance system may want to be able to see what another user is viewing (that user's "view") and stream video that is captured using

a camera in the system or that is stored on a server in the system even if the user is not directly connected to that camera or that server, respectively. Similarly, the user may want to be able to access user states (*e.g.*: whether another user of the system is currently logged into the system) and system events (*e.g.*: whether an alarm has been triggered) that are occurring elsewhere in the system, even if they originate on a server to which the user is not directly connected. In a conventional surveillance system that has been scaled out by adding more servers, a typical way to provide this functionality is to add a centralized gateway server to the system. A centralized gateway server routes system events, user states, views, and video from one server in the system to another through itself, thereby allowing the user to access or view these events, states, views, and video regardless of the particular server to which the user is directly connected. However, using a centralized gateway server gives the surveillance system a single point of failure, since if the centralized gateway server fails then the events, states, views, and video can no longer be shared. Using a centralized gateway server also increases the surveillance system's cost, since a server is added to the system and is dedicated to providing the centralized gateway server's functionality.

[0073] The user may also want common settings (*e.g.*: user access information in the form of usernames, passwords, access rights, *etc.*) to be synchronized across multiple servers in the system. In a conventional surveillance system that has been scaled out by adding more servers, this functionality is provided either by manually exporting settings from one server to other servers, or by using a centralized management server that stores all of these settings that other servers communicate with as necessary to retrieve these settings. Manually exporting settings is problematic because of relatively large synchronization delays, difficulty of use and setup, and because large synchronization delays prejudice system redundancy. Using the centralized management server suffers from the same problems as using the centralized gateway server, as discussed above.

[0074] Some of the embodiments described herein are directed at a distributed physical security system, such as a surveillance system, that can automatically share data

such as views, video, system events, user states, and user settings between two or more server nodes in the system without relying on a centralized server such as the gateway or management servers discussed above. These embodiments are directed at a peer-to-peer surveillance system in which users connect via clients to servers nodes, such as network video recorders, cameras, and servers. Server nodes are grouped together in clusters, with each server node in the cluster being able to share data with the other server nodes in the cluster. To share this data, each of the server nodes runs services that exchange data based on a protocol suite that shares data between the server nodes in different ways depending on whether the data represents views, video, system events, user states, or user settings. FIGS. 1 to 10 depict these embodiments.

[0075] In alternative embodiments, some of the technology used to share views between different server nodes is applicable to federated networks (*i.e.*, networks that include a centralized server) and to peer-to-peer networks such as those shown in FIGS. 1 to 9. FIGS. 10 and 11 depict these embodiments.

[0076] Referring now to FIG. 1, there is shown a distributed physical security system in the form of a surveillance system 100, according to one embodiment. The system 100 includes three clients 102a-c (first client 102a to third client 102c and collectively "clients 102"), six servers 104a-f (first server 104a to sixth server 104f and collectively "servers 104"), three server node cameras 106a-c (first node camera 106a to third node camera 106c and collectively "node cameras 106"), and five non-node cameras 114.

[0077] Each of the node cameras 106 and servers 104 includes a processor 110 and a memory 112 that are communicatively coupled to each other, with the memory 112 having encoded thereon statements and instructions to cause the processor 110 to perform any embodiments of the methods described herein. The servers 104 and node cameras 106 are grouped into three clusters 108a-c (collectively "clusters 108"): the first through third servers 104a-c are communicatively coupled to each other to form a first cluster 108a; the fourth through sixth servers 104d-f are communicatively coupled to each other

to form a second cluster 108b; and the three node cameras 106 are communicatively coupled to each other to form a third cluster 108c. The first through third servers 104a-c are referred to as "members" of the first cluster 108a; the fourth through sixth servers 104d-f are referred to as "members" of the second cluster 108b; and the first through third node cameras 106a-c are referred to as "members" of the third cluster 108c.

[0078] Each of the servers 104 and node cameras 106 is a "server node" in that each is aware of the presence of the other members of its cluster 108 and can send data to the other members of its cluster 108; in contrast, the non-node cameras 114 are not server nodes in that they are aware only of the servers 104a,b,c,d,f to which they are directly connected. In the depicted embodiment, the server nodes are aware of all of the other members of the cluster 108 by virtue of having access to cluster membership information, which lists all of the server nodes in the cluster 108. The cluster membership information is stored persistently and locally on each of the server nodes, which allows each of the server nodes to automatically rejoin its cluster 108 should it reboot during the system 100's operation. A reference hereinafter to a "node" is a reference to a "server node" unless otherwise indicated.

[0079] While in the depicted embodiment none of the clusters 108 participate in intercluster communication, in alternative embodiments (not shown) the members of various clusters 108 may share data with each other. In the depicted embodiment the servers 104 are commercial off-the-shelf servers and the cameras 106,114 are manufactured by Avigilon™ Corporation of Vancouver, Canada; however, in alternative embodiments, other suitable types of servers 108 and cameras 106,114 may be used.

[0080] The first client 102a is communicatively coupled to the first and second clusters 108a,b by virtue of being communicatively coupled to the first and fourth servers 104a,d, which are members of those clusters 108a,b; the second client 102b is communicatively coupled to all three clusters 108 by virtue of being communicatively coupled to the second and fourth servers 104b,d and the first node camera 106a, which are members of those clusters 108; and the third client 102c is communicatively coupled

to the second and third clusters 108b,c by virtue of being communicatively coupled to the fifth server 104e and the second node camera 106b, which are members of those clusters 108b,c. As discussed in more detail below, in any given one of the clusters 108a-c each of the nodes runs services that allow the nodes to communicate with each other according to a protocol suite 200 (shown in FIG. 2), which allows any one node to share data, whether that data be views, video, system events, user states, user settings, or another kind of data, to any other node using distributed computing; *i.e.*, without using a centralized server. Each of the nodes has access to cluster membership information that identifies all the nodes that form part of the same cluster 108; by accessing this cluster membership information, data can be shared and synchronized between all the nodes of a cluster 108.

[0081] FIG. 2 shows a block diagram of the protocol suite 200 employed by the nodes of the system 100. The protocol suite 200 is divided into three layers and includes the following protocols, as summarized in Table 1:

Table 1: Summary of the Protocol Suite 200

Protocol Name	Protocol Layer	Receives Data from these Protocols and Applications	Sends Data to these Protocols
UDP 202	Transport	Discovery Protocol 206, Node Protocol 210, Synchrony Protocol 214	N/A

TCP/HTTP 204	Transport	Node Protocol 210, Gossip Protocol 208, Membership Protocol 212, Consistency Protocol 216, Status Protocol 218	N/A
Discovery Protocol 206	Cluster Support	Node Protocol 210	UDP 202
Gossip Protocol 208	Cluster Support	Membership Protocol 212, Consistency Protocol 216, Status Protocol 218	TCP/HTTP 204, Node Protocol 210, Membership Protocol 212
Node Protocol 210	Cluster Support	Cluster Streams Application 220, Synchrony 214, Consistency Protocol 216, Membership Protocol 212, Status Protocol 218, Gossip Protocol 208	UDP 202, TCP/HTTP 204, Discovery Protocol 206
Membership Protocol 212	Cluster Support	Synchrony Protocol 214, Gossip Protocol 208, Status Protocol 218, Consistency Protocol 216	Gossip Protocol 208, Node Protocol 210, TCP/HTTP 204

Synchrony Protocol 214	Data Sync	Shared Views and Collaboration Application 222, Shared Events and Alarms Application 224	UDP 202, Node Protocol 210, Membership Protocol 212
Consistency Protocol 216	Data Sync	Shared Settings Application 226, Shared User Objects Application 228	Node Protocol 210, Membership Protocol 212, Gossip Protocol 208, TCP/HTTP 204
Status Protocol 218	Data Sync	System Information (device, server, etc.) Application 230	Gossip Protocol 208, Membership Protocol 212, Node Protocol 210, TCP/HTTP 204

[0082] A description of the function and operation of each of the protocols in the protocol suite 200 follows.

Transport Layer

[0083] The Transport Layer corresponds to layer 4 of the Open Systems Interconnection (OSI) model, and is responsible for providing reliable data transfer services between nodes to the cluster support, data synchronization, and application layers. The Transport Layer in the system 100 includes the UDP 202 and TCP/HTTP 204 protocols.

Cluster Support Layer

[0084] The Cluster Support Layer includes the protocols used to discover nodes, verify node existence, check node liveliness, determine whether a node is a member of one of the clusters 108, and determine how to route data between nodes.

Discovery Protocol 206

[0085] The Discovery protocol 206 is based on version 1.1 of the WS-Discovery protocol published by the Organization for the Advancement of Structured Information Standards (OASIS), the entirety of which is hereby incorporated by reference herein. In the depicted embodiment, XML formatting used in the published standard is replaced with Google™ Protobuf encoding.

[0086] The Discovery protocol 206 allows any node in the system 100 to identify the other nodes in the system 100 by multicasting Probe messages to those other nodes and waiting for them to respond. A node may alternatively broadcast a Hello message when joining the system 100 to alert other nodes to its presence without requiring those other nodes to first multicast the Probe message. Both the Probe and Hello messages are modeled on the WS-Discovery protocol published by OASIS.

Gossip Protocol 208

[0087] The Gossip protocol 208 is an epidemic protocol that disseminates data from one of the nodes to all of the nodes of that cluster 108 by randomly performing data exchanges between pairs of nodes in the cluster 108. The Gossip protocol 208 communicates liveliness by exchanging "heartbeat state" data in the form of a heartbeat count for each node, which allows nodes to determine when one of the nodes in the cluster 108 has left unexpectedly (*e.g.*: due to a server crash). The Gossip protocol 208 also communicates "application state" data such as top-level hashes used by the Consistency protocol 216 and status entity identifiers and their version numbers used by the Status protocol 218 to determine when to synchronize data between the nodes, as discussed in more detail below. The data spread using the Gossip protocol 208

eventually spreads to all of the nodes in the cluster 108 via periodic node to node exchanges.

[0088] A data exchange between any two nodes of the cluster 108 using the Gossip protocol 208 involves performing two remote procedure calls (RPCs) from a first node ("Node A") to a second node ("Node B") in the same cluster 108, as follows:

1. Node A sends a GreetingReq message to Node B, which contains a list of digests for all the nodes in the cluster 108 of which Node A is aware. For each node, a digest includes a unique node identifier and version information that is incremented each time either the heartbeat state or application state for that node changes. The version information may be, for example, a one-dimensional version number or a multi-dimensional version vector. Using a version vector allows the digest to summarize the history of the state changes that the node has undergone.

2. Node B sends a GreetingRsp message to Node A, which contains:

   (a) a list of digests for nodes about which Node B wishes to receive more information from Node A, which Node B determines from the version information sent to it in the GreetingReq message;

   (b) a list of digests for nodes about which Node A does not know form part of the cluster 108;

   (c) a list of one or both of heartbeat and application states that will bring Node A up-to-date on nodes for which it has out-of-date information; and

   (d) a list of nodes that Node A believes form part of the cluster 108 but that Node B knows have been removed from the cluster 108.

3. Node A then sends a ClosureReq message to Node B, in which Node A sends:

(a) a list of digests for nodes about which Node A wishes to receive more information from Node B (*e.g.* Node A may request information for nodes of which Node A was unaware until Node B sent Node A the GreetingRsp message);

(b) a list of states that will bring Node B up-to-date on nodes for which it has out-of-date information; and

(c) a list of nodes that Node B believes form part of the cluster 108 but that Node A knows have been removed from the cluster 108.

4. Node B then sends a ClosureRsp message to Node A, in which Node B sends:

(a) a list of states that will bring Node A up-to-date on nodes it is out-of-date on, in response to Node A's request in ClosureReq; and

(b) a list of nodes that have been removed from the cluster 108 since GreetingRsp.

[0089] After Nodes A and B exchange RPCs, they will have identical active node lists, which include the latest versions of the heartbeat state and application state for all the nodes in the cluster 108 that both knew about before the RPCs and that have not been removed from the cluster 108.

Node Protocol 210

[0090] The Node protocol 210 is responsible for generating a view of the system 100's network topology for each node, which provides each node with a network map permitting it to communicate with any other node in the system 100. In some embodiments, the network map is a routing table. The network map references communication endpoints, which are an address (IP/FQDN), port number, and protocol by which a node can be reached over the IP network that connects the nodes.

[0091] The Node protocol 210 does this in three ways:

1. via a "Poke exchange", as described in further detail below;

2. via the Discovery protocol 206, which notifies the Node protocol 210 when a node joins or leaves the system 100. When a node joins the system 100 a "Poke exchange" is performed with that node; and

3. manually, in response to user input.

[0092] A Poke exchange involves periodically performing the following RPCs for the purpose of generating network maps for the nodes:

1. a Poke request, in which Node A sends to Node B a Node A self view and a list of other nodes known to Node A, as viewed by Node A, following which Node B updates its network map in view of this information; and

2. a Poke response, in which Node B sends to Node A a Node B self view and a list of other nodes known to Node B, as viewed by Node B, following which Node A updates its network map in view of this information.

[0093] The RPCs are performed over the TCP/HTTP protocol 204.

[0094] To reduce bandwidth usage, node information is only exchanged between Nodes A and B if the node information has changed since the last time it has been exchanged.

[0095] A Poke exchange is performed after the Discovery protocol 206 notifies the Node protocol 210 that a node has joined the system 100 because the Discovery protocol 206 advertises a node's communication endpoints, but does not guarantee that the node is reachable using those communication endpoints. For example, the endpoints may not be usable because of a firewall. Performing a Poke exchange on a node

identified using the Discovery protocol 206 confirms whether the communication endpoints are, in fact, usable.

[0096] The Node protocol 210 can also confirm whether an advertised UDP communication endpoint is reachable; however, the Node protocol 210 in the depicted embodiment does not perform a Poke exchange over the UDP protocol 202.

[0097] For any given node in a cluster 108, a network map relates node identifiers to communication endpoints for each of the nodes in the same cluster 108. Accordingly, the other protocols in the protocol stack 200 that communicate with the Node protocol 210 can deliver messages to any other node in the cluster 108 just by using that node's node identifier.

Membership Protocol 212

[0098] The Membership protocol 212 is responsible for ensuring that each node of a cluster 108 maintains cluster membership information for all the nodes of the cluster 108, and to allow nodes to join and leave the cluster 108 via RPCs. Cluster membership information is shared between nodes of the cluster 108 using the Status protocol 218. Each node in the cluster 108 maintains its own version of the cluster membership information and learns from the Status protocol 218 the cluster membership information held by the other nodes in the cluster 108. As discussed in further detail below, the versions of cluster membership information held by two different nodes may not match because the version of cluster membership information stored on one node and that has been recently updated may not yet have been synchronized with the other members of the cluster 108.

[0099] For each node, the cluster membership information includes:

1. A membership list of all the nodes of the cluster 108, in which each of the nodes is represented by:

(a) the node identifier, which is unique among all the nodes in the system 100;

(b) the node's state, which is any one of:

(i) Discover: the node is a member of the cluster 108 but has not been synchronized with the other members of the cluster 108 since having booted;

(ii) Joining: the node is in the process of joining a cluster 108;

(iii) Syncing: the node is in the process of synchronizing data using the Synchrony, Consistency, and Status protocols 214,216,218 with the cluster 108 it has just joined;

(iv) Valid: the node has completed synchronizing the cluster membership information and is a valid node of the cluster 108; and

(v) TimedOut: the node has become unresponsive and is no longer an active member of the cluster 108 (the node remains a member of the cluster 108 until removed by a user);

(c) a session token;

(d) the version number of the cluster membership information when the node joined the cluster 108; and

(e) the version number of the cluster membership information the last time it was changed.

2. A gravestone list listing all the nodes that have been removed from the cluster 108, in which each removed node is represented by:

(a) that node's node identifier; and

(b) the version of that node's cluster membership information when the node was removed.

[00100] In the depicted embodiment, a node is always a member of a cluster 108 that comprises at least itself; a cluster 108 of one node is referred to as a "singleton cluster". Furthermore, while in the depicted embodiment the membership information includes the membership list and gravestone list as described above, in alternative embodiments (not depicted) the membership information may be comprised differently; for example, in one such alternative embodiment the membership information lacks a gravestone list, while in another such embodiment the node's state may be described differently than described above.

[00101] When Node A wants to act as a new server node and wants to join a cluster 108 that includes Node B, it communicates with Node B and the following occurs:

1. Node A sends a cluster secret to Node B, which in the depicted embodiment is a key that Node B requires before letting another node join its cluster 108. One of the clients 102 provides the cluster secret to Node A. As Node B controls Node A's access to the cluster 108, Node B acts as a "membership control node".

2. Nodes A and B exchange their membership information. The versions of the membership information on Nodes A and B are updated to include the node identifiers of Node A and of all the nodes of the cluster 108 that Node A is joining.

3. Node A's state is changed to "Joining" as Node A joins the cluster.

4. Once joined, Node A's state is changed to "Syncing" as data is exchanged between Node A and the cluster 108 it has just joined. Node B also updates the version of the membership information stored on the all the other nodes of the cluster 108 using the Status protocol 218. The process of updating the versions of the membership information stored on Node A and all the members of the cluster

108 that Node A is joining is referred to as "synchronizing" the versions of the membership information stored on all of these nodes.

5. After synchronization is complete, Node A's state changes to Valid.

Data Synchronization Layer

[00102] The Data Synchronization Layer includes the protocols that enable data to be sent between the nodes in a cluster with different ordering guarantees and performance tradeoffs. The protocols in the Data Synchronization Layer directly use protocols in the Transport and Cluster Support Layers.

Synchrony Protocol 214

[00103] The Synchrony protocol 214 is used to send data in the form of messages from Node A to Node B in the system 100 such that the messages arrive at Node B in an order that Node A can control, such as the order in which Node A sends the messages. Services that transfer data using the Synchrony protocol 214 run on dedicated high priority I/O service threads.

[00104] In the depicted embodiment, the Synchrony protocol 214 is based on an implementation of virtual synchrony known as the Totem protocol, as described in Agarwal DA, Moser LE, Melliar-Smith PM, Budhia RK, "The Totem Multiple-Ring Ordering and Topology Maintenance Protocol", ACM Transactions on Computer Systems, 1998, pp. 93 – 132, the entirety of which is hereby incorporated by reference herein. In the Synchrony protocol 214, nodes are grouped together into groups referred to hereinafter in this description as "Synchrony rings", and a node on any Synchrony ring can send totally ordered messages to the other nodes on the same ring. The Synchrony protocol 214 modifies the Totem protocol as follows:

1. The Synchrony protocol 214 uses both a service identifier and a ring identifier to identify a Synchrony ring. The service identifier identifies all instances of a given

Synchrony ring, whereas the ring identifier identifies a particular instance of a given Synchrony ring. For example, each time a node joins or leaves a Synchrony ring that ring's ring identifier will change, but not its service identifier. The service identifier allows a node to multicast totally ordered messages to the group of nodes that share the same service identifier (*i.e.* the group of nodes that belong to the same Synchrony ring).

2. In the Totem protocol, in some cases when the nodes are not sending messages the Synchrony ring seen by nodes does not reflect the final ring configuration that converges when the nodes begin messaging. The Synchrony protocol 214 allows nodes to send probe messages to each other to cause Synchrony rings to converge prior to the sending of non-probe messages.

3. The Totem protocol only allows ordered messages to be sent to all nodes that form part of a Synchrony ring. In contrast, the Synchrony protocol 214 uses a Dispatch module that abstracts the network layer from the Synchrony protocol 214 by providing an interface to broadcast to all reachable nodes in the system 100; multicast to any set of nodes in the system 100 using a list of destination node identifiers; and to unicast to a single node in the system 100 using its node identifier. The Dispatch module also supports multiplexing of services on the same IP port using message filtering and routing by service identifier. Outgoing messages from a node are sent to the subset of nodes having the same service identifier unless multicast.

4. The Synchrony protocol 214 uses fragmented messages and user payload chunking and coalescing to address problems arising from the maximum transmission unit size of approximately 1,500 bytes.

5. The Synchrony protocol 214 modifies the way nodes use Join messages, which are messages nodes use in the Totem protocol to join a Synchrony ring:

(a) Join messages are sent by nodes only if they have the lowest node identifier in the current set of operational nodes in the Synchrony ring.

(b) Nodes that do not have the lowest node identifier in their operational set unicast Join messages to the nodes with the lowest node identifier in their operational set.

(c) Join messages include the service identifier, and nodes that are not part of the corresponding Synchrony ring do not respond.

Relative to the Totem protocol, these modifications help reduce aggregate bandwidth used by nodes to join Synchrony rings.

6. The Synchrony protocol 214 detects and blacklists nodes that are unable to join a Synchrony ring due to some types of network misconfigurations. For example, a node that is able to send to, but not receive messages from, the other nodes will appear to the other nodes to only ever send probe messages since all other messages in the present embodiment are solicited, and accordingly will be blacklisted.

7. The Synchrony protocol 214 performs payload encryption and authenticity verification of messages.

8. The Synchrony protocol 214 limits the time each node can hold the token used in the Totem protocol; in the depicted embodiment, each node can hold the token for 15 ms.

9. The Synchrony protocol 214 implements a TCP friendly congestion avoidance algorithm.

[00105] As discussed in more detail below, the system 100 uses the Synchrony protocol for the Shared Views and Collaboration application 222 and the Shared Events and Alarms application 224; the data shared between members of a cluster 108 in these

applications 222 is non-persistent and is beneficially shared quickly and in a known order.

Consistency Protocol 216

[00106] The Consistency protocol 216 is used to automatically and periodically share data across all the nodes of a cluster 108 so that the data that is shared using the Consistency protocol 216 is eventually synchronized on all the nodes in the cluster 108. The types of data that are shared using the Consistency protocol 216 are discussed in more detail below in the sections discussing the Shared Settings application 226 and the Shared User Objects application 228. Data shared by the Consistency protocol 216 is stored in a database on each of the nodes, and each entry in the database includes a key-value pair in which the key uniquely identifies the value and the keys are independent from each other. The Consistency protocol 216 synchronizes data across the nodes while resolving parallel modifications that different nodes may perform on different databases. As discussed in further detail below, the Consistency protocol 216 accomplishes this by first being notified that the databases are not synchronized; second, finding out which particular database entries are not synchronized; and third, finding out what version of the entry is most recent, synchronized, and kept.

[00107] In order to resolve parallel modifications that determine when changes are made to databases, each node that joins a cluster 108 is assigned a causality versioning mechanism used to record when that node makes changes to data and to determine whether changes were made before or after changes to the same data made by other nodes in the cluster 108. In the present embodiment, each of the nodes uses an interval tree clock (ITC) as a causality versioning mechanism. However, in alternative embodiments other versioning mechanisms such as vector clocks and version vectors can be used. The system 100 also implements a universal time clock (UTC), which is synchronized between different nodes using Network Time Protocol, to determine the order in which changes are made when the ITCs for two or more nodes are identical. ITCs are described in more detail in P. Almeida, C. Baquero, and V. Fonte, "Interval tree

clocks: a logical clock for dynamic systems", *Princi. Distri. Sys., Lecture Notes in Comp. Sci.*, vol. 5401, pp. 259–274, 2008, the entirety of which is hereby incorporated by reference herein.

[00108] The directory that the Consistency protocol 216 synchronizes between nodes is divided into branches, each of which is referred to as an Eventual Consistency Domain (ECD). The Consistency protocol 216 synchronizes each of the ECDs independently from the other ECDs. Each database entry within an ECD is referred to as an Eventual Consistency Entry (ECE). Each ECE includes a key; a timestamp from an ITC and from the UTC, which are both updated whenever the ECE is modified; a hash value of the ECE generating using, for example, a Murmurhash function; the data itself; and a gravestone that is added if and when the ECE is deleted.

[00109] The hash value is used to compare corresponding ECDs and ECEs on two different nodes to determine if they are identical. When two corresponding ECDs are compared, "top-level" hashes for those ECDs are compared. A top-level hash for an ECD on a given node is generated by hashing all of the ECEs within that ECD. If the top-level hashes match, then the ECDs are identical; otherwise, the Consistency protocol 216 determines that the ECDs differ. To determine which particular ECEs in the ECDs differ, hashes are taken of successively decreasing ranges of the ECEs on both of the nodes. The intervals over which the hashes are taken eventually shrinks enough that the ECEs that differ between the two nodes are isolated and identified. A bi-directional skip-list can be used, for example, to determine and compare the hash values of ECD intervals.

[00110] Two nodes that communicate using the Consistency protocol 216 may use the following RPCs:

1. SetEntries: SetEntries transmits new or updated ECEs to a node, which inserts them into the appropriate ECDs.

2. GetEntries: GetEntries transmits a key or a range of keys to a node, which returns the ECEs corresponding to those one or more keys.

3. SynEntries: SynEntries transmits a key or a range of keys to a node, and the two nodes then compare hashes of successively decreasing ranges of ECEs to determine which ECEs differ between the two nodes, as described above. If the ECEs differ, the nodes merge their ECEs so that the same ECEs are stored on the nodes by comparing the ITC timestamps; if the ITC timestamps match, the nodes compare the UTC timestamps associated with the ECEs. These timestamps act as version information that allows the two nodes to adopt the ECEs that have been most recently modified, as indicated by those ECEs' version information.

[00111] When a node changes ECEs, that node typically calls SynEntries to inform the other nodes in the cluster 108 that the ECEs have been changed. If some of the nodes in the cluster 108 are unavailable (*e.g.*: they are offline), then the Gossip protocol 208 instead of SynEntries is used to communicate top-level hashes to the unavailable nodes once they return online. As alluded to in the section discussing the Gossip protocol 208 in the cluster 108 above, each of the nodes holds its top-level hash, which is spread to the other nodes along with a node identifier, version information, and heartbeat state using the Gossip protocol 208. When another node receives this hash, it compares the received top-level hash with its own top-level hash. If the top-level hashes are identical, the ECEs on both nodes match; otherwise, the ECEs differ.

[00112] If the ECEs differ, regardless of whether this is determined using SynEntries or the Gossip protocol 208, the node that runs SynEntries or that receives the top-level hash synchronizes the ECEs.

Status Protocol 218

[00113] As discussed above, the Gossip protocol 208 shares throughout the cluster 108 status entity identifiers and their version numbers ("status entity pair") for nodes in the cluster 108. Exemplary status entity identifiers may, for example, represent different types of status data in the form of status entries such as how much storage the node has available; which devices (such as the non-node cameras 114) are connected to that node;

which clients 102 are connected to that node; and cluster membership information. When one of the nodes receives this data via the Gossip protocol 208, it compares the version number of the status entity pair to the version number of the corresponding status entry it is storing locally. If the version numbers differ, the Status protocol 218 commences an RPC ("Sync RPC") with the node from which the status entity pair originates to update the corresponding status entry.

[00114] A status entry synchronized using the Status protocol 218 is uniquely identified by both a path and a node identifier. Unlike the data synchronized using the Consistency protocol 216, the node that the status entry describes is the only node that is allowed to modify the status entry or the status entity pair. Accordingly, and unlike the ECDs and ECEs synchronized using the Consistency protocol 216, the version of the status entry for Node A stored locally on Node A is always the most recent version of that status entry.

[00115] If Node A modifies multiple status entries simultaneously, the Status protocol 218 synchronizes all of the modified status entries together to Node B when Node B calls the Sync RPC. Accordingly, the simultaneously changed entries may be dependent on each other because they will be sent together to Node B for analysis. In contrast, each of the ECEs synchronized using the Consistency protocol 216 is synchronized independently from the other ECEs, so ECEs cannot be dependent on each other as Node B cannot rely on receiving entries in any particular order.

Applications

[00116] Each of the nodes in the system 100 runs services that implement the protocol suite 200 described above. While in the depicted embodiment one service is used for each of the protocols 202-218, in alternative embodiments (not depicted) greater or fewer services may be used to implement the protocol suite 200. Each of the nodes implements the protocol suite 200 itself; consequently, the system 100 is distributed and is less vulnerable to a failure of any single node, which is in contrast to conventional

physical security systems that use a centralized server. For example, if one of the nodes fails in the system 100 ("failed node"), on each of the remaining nodes the service running the Status protocol 218 ("Status service") will determine that the failed node is offline by monitoring the failed node's heartbeat state and will communicate this failure to the service running the Node and Membership protocols 210,212 on each of the other nodes ("Node service" and "Membership service", respectively). The services on each node implementing the Synchrony and Consistency protocols 214,216 ("Synchrony service" and "Consistency service", respectively) will subsequently cease sharing data with the failed node until the failed node returns online and rejoins its cluster 108.

[00117] The following describes the various applications 220-230 that the system 100 can implement. The applications 220-230 can be implemented as various embodiments of the exemplary method for sharing data 800 depicted in FIG. 8. The method 800 begins at block 802 and proceeds to block 804 where a first node in the system 100 accesses a node identifier identifying another node in the system 100. Both the first and second nodes are members of the same server cluster 108. The node identifier that the first node accesses is part of the cluster membership information that identifies all the members of the cluster 108. The cluster membership information is accessible by all the members of the cluster 108. In the depicted embodiments each of the members of the cluster 108 stores its own version of the cluster membership information persistently and locally; however, in alternative embodiments (not depicted), the cluster membership information may be stored one or both of remotely from the nodes and in a central location. After accessing the node identifier for the second node, the first node sends the data to the second node at block 806, following which the method 800 ends at block 808. For example, when using the Node service described above, the Synchrony and Consistency services running on the first node are able to send the data to the second node by using the second node's node identifier, and by delegating to the Node service responsibility for associating the second node's communication endpoint to its node identifier. Sending the data from the first node to the second node at block 806

can comprise part of a bi-directional data exchange, such as when data is exchanged in accordance with the Gossip protocol 208.

Shared Settings Application 226 and Shared User Objects Application 228

[00118] During the system 100's operation, persistently stored information is transferred between the nodes of a cluster 108. Examples of this real-time information that the shared settings and shared user objects applications 226,228 share between nodes are shared settings such as rules to implement in response to system events such as an alarm trigger and user objects such as user names, passwords, and themes. This type of data ("Consistency data") is shared between nodes using the Consistency protocol 216; generally, Consistency data is data that does not have to be shared in real-time or in total ordering, and that is persistently stored by each of the nodes. However, in alternative embodiments (not depicted), Consistency data may be non-persistently stored.

[00119] FIG. 3 shows a UML sequence diagram 300 in which Consistency data in the form of a user settings are shared between first and second users 302a,b (collectively, "users 302"). The users 302, the first and second clients 102a,b, and the first and second servers 104a,b, which are the first and second nodes in this example, are objects in the diagram 300. The servers 104a,b form part of the same cluster 108a. As the servers 104a,b with which the clients 102a,b communicate are not directly connected to each other, the Consistency protocol 216 is used to transfer data between the two servers 104a,b, and thus between the two users 302. Although the depicted embodiment describes sharing settings, in an alternative embodiment (not depicted) the users 302 may analogously share user objects.

[00120] The diagram 300 has two frames 332a,b. In the first frame 332a, the first user 302a instructs the first client 102a to open a settings panel (message 304), and the client 102a subsequently performs the SettingsOpenView() procedure (message 306), which transfers the settings to the first server 104a. Simultaneously, the second user 302b instructs the second client 102b analogously (messages 308 and 310). In the second

frame 332b, the users 302 simultaneously edit their settings. The first user 302a edits his settings by having the first client 102a run UIEditSetting() (message 312), following which the first client 102a updates the settings stored on the first server 104a by having the first server 104a run SettingsUpdateView() (message 314). The first server 104a then runs ConsistencySetEntries() (message 316), which performs the SetEntries procedure and which transfers the settings entered by the first user 302a to the second server 104b. The second server 104b then sends the transferred settings to the second client 102b by calling SettingsNotifyViewUpdate() (message 318), following which the second client 102b updates the second user 302b (message 320). Simultaneously, the second user 302b analogously modifies settings and sends those settings to the first server 104a using the Consistency protocol 216 (messages 322, 324, 326, 328, and 330). Each of the servers 104a,b persistently stores the user settings so that they do not have to be resynchronized between the servers 104a,b should either of the servers 104a,b reboot.

Shared Events and Alarms Application 224

[00121] During the system 100's operation, real-time information generated during runtime is transferred between the nodes of a cluster 108. Examples of this real-time information that the shared events and alarms application 224 shares between nodes are alarm state (*i.e.* whether an alarm has been triggered anywhere in the system 100); system events such as motion having been detected, whether a device (such as one of the node cameras 106) is sending digital data to the rest of the system 100, whether a device (such as a motion detector) is connected to the system 100, whether a device is currently recording, whether an alarm has occurred or has been acknowledged by the users 302, whether one of the users 302 is performing an audit on the system 100, whether one of the servers 104 has suffered an error, whether a device connected to the system has suffered an error, whether a point-of-sale text transaction has occurred; and server node to client notifications such as whether settings/data having changed, current recording state, whether a timeline is being updated, and database query results. In the present embodiment, the data transferred between nodes using the Synchrony protocol 214 is

referred to as "Synchrony data", is generated at run-time, and is not persistently saved by the nodes.

[00122] FIG. 4 shows a UML sequence diagram 400 in which an alarm notification is shared between the servers 104 using the Synchrony protocol 214. The objects in the diagram 400 are one of the non-node cameras 114, the three servers 104 in the first cluster 108a, and the second client 102b, which is connected to one of the servers 104c in the first cluster 108a.

[00123] At the first three frames 402 of the diagram 400, each of the servers 104 joins a Synchrony ring named "ServerState" so that the state of any one of the servers 104 can be communicated to any of the other servers 104; in the depicted embodiment, the state that will be communicated is "AlarmStateTriggered", which means that an alarm on one of the servers 108 has been triggered by virtue of an event that the non-node camera 114 has detected. At frame 404, the second server 104b is elected the "master" for the Alarms application; this means that it is the second server 104b that determines whether the input from the non-node camera 114 satisfies the criteria to transition to the AlarmStateTriggered state, and that sends to the other servers 104a,c in the Synchrony ring a message to transition them to the AlarmStateTriggered state as well.

[00124] The second user 302b logs into the third server 104c after the servers 104 join the ServerState Synchrony ring (message 406). Subsequent to the user 302b logging in, the third server 104c joins another Synchrony ring named "ClientNotification"; as discussed in further detail below, this ring is used to communicate system states to the user 302b, whereas the ServerState Synchrony ring is used to communicate only between the servers 104. The non-node camera 114 sends a digital input, such as a indication that a door or window has been opened, to the first server 104a (message 410), following which the first server 104a checks to see whether this digital input satisfies a set of rules used to determine whether to trigger an alarm in the system 100 (message 412). In the depicted embodiment, the first server 104a determines that an alarm should be triggered, and accordingly calls AlarmTrigger(), which alerts the second server 104b to change

states. The second server 104 then transitions states to AlarmStateTriggered (message 416) and sends a message to the ServerState Synchrony ring that instructs the other two servers 104a,c to also change states to AlarmStateTriggered (frame 418). After instructing the other servers 104a,c, the second server 104b runs AlarmTriggerNotification() (message 420), which causes the second server 104b to also join the ClientNotification Synchrony ring (frame 422) and pass a message to the ClientState Synchrony ring that causes the third server 104c, which is the other server on the ClientState Synchrony ring, to transition to a "NotifyAlarmTriggered" state (frame 424). Once the third server 104c changes to this state it directly informs the second client 102b that the alarm has been triggered, which relays this message to the second user 302b and waits for the user second 302b to acknowledge the alarm (messages 426). Once the second user 302b acknowledges the alarm, the second server 104b accordingly changes states to "AlarmStateAcknowledged" (message 428), and then sends a message to the ServerState Synchrony ring so that the other two servers 104a,c correspondingly change state as well (frame 430). The second server 104b subsequently changes state again to "NotifyAlarmAcknowledged" (message 432) and sends a message to the third server 104c via the ClientNotification Synchrony ring to cause it to correspondingly change state (frame 434). The third server 104c then notifies the client 102c that the system 100 has acknowledged the alarm (message 436), which relays this message to the second user 302b (message 438).

[00125] In an alternative embodiment (not depicted) in which the second server 104b fails and can no longer act as the master for the Synchrony ring, the system 100 automatically elects another of the servers 104 to act as the master for the ring. The master of the Synchrony ring is the only server 104 that is allowed to cause all of the other nodes on the ring to change state when the Synchrony ring is used to share alarm notifications among nodes.

[00126] FIG. 7 shows an exemplary view 700 presented to the users 302 when acknowledging an alarm state in accordance with the diagram 400 of FIG. 4. The view

700 includes video panels 702a-c (collectively "panels 702") showing real time streaming video from the non-node camera 114; alerts 704 indicating that an alarm has been triggered as a result of what the non-node camera 114 is recording; and an acknowledge button 706 that the second user 302b clicks in order to acknowledge the alarm having been triggered.

Shared Views and Collaboration Application 222

[00127] The users 302 of the system 100 may also want to share each others' views 700 and collaborate, such as by sending each other messages and talking to each other over the system 100, while sharing views 700. This shared views and collaboration application 222 accordingly allows the users 302 to share data such as view state and server to client notifications such as user messages and share requests. This type of data is Synchrony data that is shared in real-time.

[00128] FIG. 5 shows a UML sequence diagram 500 in which views 700 are shared between the users 302 using the Synchrony protocol 214. The diagram 500 includes six objects: the first and second users 302a,b, the first and second clients 102a,b to which the first and second users 302a,b are respectively connected, and the first and second servers 104a,b to which the first and second clients 102a,b are respectively connected.

[00129] The first user 302a logs into the first server 104a via the first client 102a (message 502), following which the first server 104a joins the ClientNotification Synchrony ring (frame 504). Similarly, the second user 302b logs into the second server 104b via the second client 102b (message 506), following which the second server 104b also joins the ClientNotification Synchrony ring (frame 508).

[00130] The first user 302a then instructs the first client 102a that he wishes to share his view 700. The first user 302a does this by clicking a share button (message 510), which causes the first client 102a to open the view 700 to be shared ("shared view 700") on the first server 104a (message 512). The first server 104a creates a shared view

session (message 514), and then sends the session identifier to the first client 102a (message 516).

[00131] At one frame 518 each of the clients 102 joins a Synchrony ring that allows them to share the shared view 700. The first server 104a joins the SharedView1 Synchrony ring at frame 520. Simultaneously, the first client 106a instructs the first server 104a to announce to the other server 104b via the Synchrony protocol 214 that the first user 302a's view 700 can be shared by passing to the first server 104a a user list and the session identifier (message 522). The first server 104a does this by sending a message to the second server 104b via the ClientNotify Synchrony ring that causes the second server 104 to change to a NotifyViewSession state (frame 524). In the NotifyViewSession state, the second server 104b causes the second client 106b to prompt the second user 302b to share the first user 302a's view 700 (messages 526 and 528), and the second user 302b's affirmative response is relayed back to the second server 104b (messages 530 and 532). The second server 104b subsequently joins the SharedView1 Synchrony ring (frame 534), which is used to share the first user 302a's view 700.

[00132] At a second frame 519 the users 106 each update the shared view 700, and the updates are shared automatically with each other. The first user 302a zooms into a first panel 702a in the shared view 700 (message 536), and the first client 102a relays to the first server 104a how the first user 302a zoomed into the first panel 702a (message 538). The first server 104a shares the zooming particulars with the second server 104b by passing them along the SharedView1 Synchrony ring (frame 540). The second server 104b accordingly updates the shared view 700 as displayed on the second client 106b (message 542), and the updated shared view 700 is then displayed to the second user 302b (message 544). Simultaneously, the second user 302b pans a second panel 702b in the shared view 700 (message 546), and the second client 102b relays to the second server 104b how the second user 302b panned this panel 702b (message 548). The second server 104b then shares the panning particulars with the first server 104a by passing them using the SharedView1 Synchrony ring (frame 550). The first server 104a

accordingly updates the shared view 700 as displayed on the first client 106b (message 552), and the updated shared view 700 is then displayed to the first user 302a (message 556).

[00133] After the second frame 519, the first user 302a closes his view 700 (message 556), which is relayed to the first server 104a (message 558). The first server 104a consequently leaves the SharedView1 Synchrony ring (message and frame 560). The second user 302b similarly closes his view 700, which causes the second server 104b to leave the SharedView1 Synchrony ring (messages 562 and 564, and message and frame 566).

[00134] In the example of FIG. 5, the users 302 pan and zoom the shared view 700. In alternative embodiments (not depicted) the users 302 may modify the shared view 700 in other ways. For example, the users 302 may each change the layout of the panels 702; choose whether video is to be displayed live or in playback mode, in which case the users 302 are also able to pause, play, or step through the video; and display user objects such as maps or web pages along with information about the user object such as revision history. In these alternative embodiments, examples of additional state information that is synchronized using a Synchrony ring include whether a video is being played, paused, or stepped through and the revision history of the user object.

[00135] While the discussion above focuses on the implementation of the shared views and collaboration application 222 in the peer-to-peer physical security system 100 of FIG. 1, more generally this application 222 may be implemented in a physical security system that has multiple servers 104, such as a federated system that includes a centralized gateway server. An example of this more general embodiment is shown in FIG. 12, which depicts an exemplary method 1200 for sharing a view using a physical security system that comprises a plurality of server nodes. The method 1200 begins at block 1202 and proceeds to block 1204, where view state data representative of the view displayed by the first client (such as the first client 102a), which is the view to be shared, is sent from the first client to a first server node (such as the first server 104a and the

view state data sent via message 538). At block 1206 the view state data is relayed from the first server node to a second client (such as the second client 102b) via a second server node (such as the second server 104b and the view state data sent via frame 540 and message 542). At block 1208 the second client then updates a display using the view state data to show the shared view (such as via message 544). In response to a change in the shared view at the second client, such as a change resulting from interaction with a user at the second client (such as via message 546), at block 1210 updated view state data is sent from the second client to the second server node (such as via message 548). The updated view state data is representative of the shared view as displayed by the second client. The updated view state data is sent from the second server node to the first client via the first server node at block 1212 (such as via frame 550 and message 552), and at block 1214 the first client's display is then updated to show the shared view as it was modified at the second client using the updated view state data (such as via message 554). The method 1200 ends at block 1216. In an alternative embodiment such as when dealing with a federated system that uses a centralized gateway server, all the view state data may be routed through that centralized server.

Unattended View Sharing Application 225

[00136] The users 302 of the system 100 may also want to be able to see and control a view on a display that is directly or indirectly connected to one of the servers 104 that the users 302 do not directly control (*i.e.*, that the users 302 control via other servers 104) (this display is an "unattended display", and the view on the unattended display is the "unattended view"). For example, the unattended display may be mounted on a wall in front of the users 302 and be connected to the server cluster 108 via one of the servers 104 in the cluster 108, while the users 302 may be connected to the server cluster 108 via other servers 104 in the cluster 108. As discussed below with respect to FIG. 10, the unattended view sharing application 225 permits the users 302 to view and control the unattended view notwithstanding that none of the users 302 is directly connected to the server 104 controlling the unattended view. The view data exchanged

between the servers 104 to enable this functionality is Synchrony data that is shared in real-time.

[00137] FIG. 10 shows a UML sequence diagram 1000 in which the unattended view is shared with the first user 302a using the Synchrony protocol 214. The diagram 1000 includes six objects: the first user 302a, the first client 102a to which the first user 302a is connected and that includes a display ("client display") with which the first user 302a interacts, the first and second servers 104a,b, a monitor instance 1004 running on hardware such as an unattended one of the clients 102 connected to both the second server 104b and the unattended display, and an administrator 1002 who sets up the monitor instance 1004. In an alternative embodiment (not depicted), the unattended display may be directly connected to the second server 104b and the monitor instance 1004 may run on the second server 104b.

[00138] In FIG. 10, the administrator 1002 creates the monitor instance 1004 (message 1006) and the monitor instance 1004 then automatically logs into the second server 104b (messages 1008 and 1010). The monitor instance 1004 makes the unattended view available to the second server 104b by calling SharedViewOpen(viewState) on the second server 104, where viewState is view state data indicative of the unattended view (message 1012). Following this the second server 104b creates a shared view session (message 1014) by running SharedViewSessionCreate() and then sends the corresponding session identifier to the monitor instance (message 1016). After receiving the session identifier the monitor instance 1004 joins the SharedView1 Synchrony ring (frame 1018), which is used to transmit view state data to and from the other servers 104 in the cluster 108 that are also members of the SharedView1 Synchrony ring.

[00139] After joining the SharedView1 Synchrony ring, the monitor instance 1020 publishes a notification to the other servers 104 in the cluster 108 that the unattended view is available to be seen and controlled. The monitor instance 1020 does this by calling RegisterMonitor(sessionId) on the second server 104b (message 1018), which causes the session identifier related to the unattended view to be registered in a view

directory (frame 1022). The view directory is shared with the other servers 104 in the cluster 108 using the Consistency protocol 216.

[00140] Once the view directory is disseminated to the other servers 104 in the cluster 108, those other servers 104 can access the view directory to determine which unattended views are available to view and control. After the first server 104a receives the view directory, the first user 302a via the first client 102a logs into the first server 104a, thereby gaining access to the cluster 108 (messages 1024) and the view directory. The first user 102a instructs the first client 102a to display the unattended view by calling UIDisplayMonitor(sessionId) (message 1026), which causes the first client 102a to send the unattended view's session identifier to the first server 104a with instructions to open the unattended view (message 1028). The first server 104a acknowledges the instructions of the first client 102a (message 1030) and then joins the SharedView1 Synchrony ring (frame 1032) in order to automatically receive view state data describing the current view of the unattended display (message 1034) and to automatically stay apprised of any subsequent changes to the unattended view.

[00141] The first user 302a subsequently pans one of the panels of the unattended view as it is displayed on the client display (message 1036), and the first client 102a relays the panning action and the identity of the particular panel that is panned to the first server 104a by calling SharedViewUpdate(action=pan, panelId=2) (message 1038). The first server 104a sends updated view state data to all the servers 104 that are members of the SharedView1 Synchrony ring (frame 1040), which allows all of those servers 104 to reproduce the updated version of the unattended view. The second server 104b receives this updated view state data and relays it to the monitor instance 1004 by calling NotifySharedViewUpdate(action=pan, params, panelId=2) (message 1042). The monitor instance 1004 then updates the unattended display to show the unattended view as modified by the first user 302a (message 1044).

[00142] In the example of FIG. 10, the first user 302a pans one of the panels of the unattended view. In alternative embodiments (not depicted) the first user 302a may

modify the unattended view in other ways. For example, the first user 302a may change the layout of any one or more of the unattended view's panels; choose whether video is to be displayed live or in playback mode, in which case the first user 302a is also able to pause, play, or step through the video; and display user objects such as maps or web pages along with information about the user object such as revision history. In these alternative embodiments, examples of additional state information that is synchronized using a Synchrony ring include whether a video is being played, paused, or stepped through and the revision history of the user object.

[00143] In another alternative embodiment (not depicted), the unattended view sharing application 225 may be used to create an aggregate display comprising a matrix of n x m unattended displays. For example, where n = m = 2 and there are consequently four unattended displays, the first user 302a may control all four of the unattended displays simultaneously to create one, large virtual display. A single video can then be enlarged such that each of the unattended views is of one quadrant of the video, thereby allowing the video to be enlarged and shown over the four unattended displays. In this embodiment, the monitor instances 1004 for the unattended displays may be communicative with the server cluster 108 via any of one to four of the servers 104.

[00144] While FIG. 10 shows only the first user 302a, in alternative embodiments (not depicted) more than one of the users 302 can see and control the unattended view by also joining the SharedView1 Synchrony ring. In the above example of the aggregated display comprising the n x m matrix of unattended displays, the aggregated display can be mounted in the room for simultaneous viewing several of the users 302 with each of the users 302 having the ability to control each of the unattended views.

[00145] While the discussion above focuses on the implementation of the unattended view sharing application 225 in the peer-to-peer physical security system 100 of FIG. 1, more generally this application 225 may be implemented in a physical security system that has multiple servers 104, such as a federated system that includes a centralized gateway server. An example of this more general embodiment is shown in

FIG. 11, which depicts an exemplary method 1100 for interacting with the unattended display in a physical security system comprising multiple server nodes. The method begins at block 1102 and proceeds to block 1104 where a second server node (such as the second server 104b) that is communicative with the unattended display sends to a first server node (such as the first server 104a) view state data indicative of the unattended view (such as via the Synchrony ring at frames 1020 and 1032 of FIG. 10). The method 1100 then proceeds to block 1106 where at least a portion of the unattended view is displayed on the client display (such as the update of the client display that results from message 1034 of FIG. 10). In an alternative embodiment such as when dealing with a federated system that uses a centralized gateway server, all the view state data may be routed through that centralized server.

Cluster Streams Application 220

[00146] One of the users 302 may also want to stream video from one of the cameras 106,114 if a point-to-point connection between that user 302 and that camera 106,114 is unavailable; the cluster streams application 220 enables this functionality. FIG. 6 shows a UML sequence diagram 500 in which video is streamed from the non-node camera 114 to the first user 302a through the first and second servers 104a,b and the first client 102a. The UML diagram has five objects: the first user 302a, the first client 102a, the first and second servers 104a,b, and the non-node camera 114. The first client 102a can directly communicate with the first server 104a, but cannot directly communicate with the second server 104b. However, the first and second servers 104a,b can communicate directly with each other. Additionally, while the second server 104b and the non-node camera 114 can communicate directly with each other, the first server 104a and the non-node camera 114 cannot directly communicate.

[00147] The second server 104b first establishes a session with the non-node camera 114 so that video is streamed from the non-node camera 114 to the second server 104b. The second server 104b first sets up a Real Time Streaming Protocol (RTSP) session with the non-node camera 114 (messages 602 and 604), and instructs the non-

node camera 114 to send it video (messages 606 and 608). The non-node camera 114 subsequently commences streaming (message 610).

[00148] The first user 302a establishes a connection with the first client 102a (message 612) and then instructs the first client 102a to open a window showing the streaming video (message 614). The first client 102a then calls LookupRoute() to determine to which server 104 to connect; because the first client 102a cannot connect directly to the second server 104b, it sets up an RTSP connection with the first server 104a (message 618). The first server 104b then calls LookupRoute() to determine to which node to connect to access the real-time video, and determines that it should connect with the second server 104b (message 620). The first server 104a subsequently sets up an RTSP connection with the second server 104b (message 622), and the second server 104b returns a session identifier to the first server 104a (message 624). The first server 104a relays the session identifier to the first client 102a (message 626). Using this session identifier, the first client 102a instructs the second server 104b to begin playing RTSP video (messages 628 to 634), and the second server 104b subsequently streams video to the first user 302a via the second server 104b, then the first server 104a, and then the first client 102a (messages 636 to 640).

[00149] While FIG. 6 routes video from one of the non-node cameras 114 connected to one of the servers 104 in a cluster 108 to other servers 104 in the same cluster 108, in alternative embodiments (not depicted) video may also be routed from one of the node cameras 106 in a cluster 108 through the other node cameras 106 in the same cluster 108.

Rebooting

[00150] In the present embodiment, the cluster membership information is persistently stored locally on each of the nodes. When one of the nodes reboots, it automatically rejoins the cluster 108 of which it was a member prior to rebooting. This is depicted in the exemplary method 900 shown in FIG. 9. After performing block 806, one

of the nodes in the cluster 108 reboots (block 902). Upon rebooting, this node accesses the persistently stored cluster membership information that identifies the cluster 108 of which it was a member prior to rebooting (block 904), and subsequently rejoins this cluster 108 (block 906) before returning to block 808. Having the nodes automatically rejoin a cluster 108 following rebooting is beneficial in that it helps the system 100 recover following restarting of any one or more of its servers. As each of the nodes persistently stores the Consistency information, upon rejoining the cluster 108 only that Consistency information that has changed since the node last left the cluster 108 is synchronized again, thereby saving bandwidth.

[00151] While certain exemplary embodiments are depicted, alternative embodiments, which are not depicted, are possible. For example, while in the depicted embodiment the node cameras 106 and non-node cameras 114 are distinct from each other, in alternative embodiments (not depicted) a single camera may be simultaneously a node camera and a non-node camera. For example, in FIG. 1 the first camera 106a is a node that is a member of the third cluster 108c; however, if the first camera 106a were also directly coupled to the fifth server 104e but retained only its cluster membership information for the third cluster 108c, the first camera 106a would remain a member of the third cluster 108c while simultaneously acting as a non-node camera 114 from the perspective of the fifth server 104e.

[00152] The processor used in the foregoing embodiments may be, for example, a microprocessor, microcontroller, programmable logic controller, field programmable gate array, or an application-specific integrated circuit. Examples of computer readable media are non-transitory and include disc-based media such as CD-ROMs and DVDs, magnetic media such as hard drives and other forms of magnetic disk storage, semiconductor based media such as flash media, random access memory, and read only memory.

[00153] It is contemplated that any part of any aspect or embodiment discussed in this specification can be implemented or combined with any part of any other aspect or embodiment discussed in this specification.

[00154] For the sake of convenience, the exemplary embodiments above are described as various interconnected functional blocks. This is not necessary, however, and there may be cases where these functional blocks are equivalently aggregated into a single logic device, program or operation with unclear boundaries. In any event, the functional blocks can be implemented by themselves, or in combination with other pieces of hardware or software.

[00155] While particular embodiments have been described in the foregoing, it is to be understood that other embodiments are possible and are intended to be included herein. It will be clear to any person skilled in the art that modifications of and adjustments to the foregoing embodiments, not shown, are possible.

[00156] Throughout this specification and the claims which follow, unless the context requires otherwise, the word "comprise", and variations such as "comprises" or "comprising", will be understood to imply the inclusion of a stated integer or step or group of integers or steps but not the exclusion of any other integer or step or group of integers or steps.

[00157] The reference in this specification to any prior publication (or information derived from it), or to any matter which is known, is not, and should not be taken as, an acknowledgement or admission or any form of suggestion that that prior publication (or information derived from it) or known matter forms part of the common general knowledge in the field of endeavour to which this specification relates.

2013312982 16 Oct 2017

The claims defining the invention are as follows:

1. A method for sharing data in a physical security system that comprises a plurality of server nodes, the method comprising:

(a) adding a first server node to a server cluster comprising a second server node by performing a method comprising:

(i) exchanging a version of cluster membership information stored on the first server node with a version of the cluster membership information stored on one of the server nodes that is already part of the server cluster; and

(ii) synchronizing the version of the cluster membership information stored on the first server node with versions of the cluster membership information stored on all the server nodes that, prior to the first server node joining the cluster, comprised part of the cluster;

(b) accessing, after the first node has been added to the server cluster and using the first node, a node identifier identifying another node of the server cluster, wherein the node identifier comprises at least part of cluster membership information identifying all the server nodes in the server cluster and accessible by all the server nodes in the server cluster; and wherein each of the server nodes in the server cluster persistently stores its own version of the cluster membership information locally; and

(c) sending the data from the first node to the other node.

2. A method as claimed in claim 1 wherein the server cluster comprises at least three server nodes.

3. A method as claimed in claim 1 or 2 wherein the server nodes comprise cameras, network video recorders, and access control servers.

4. A method as claimed in any one of claims 1 to 3 further comprising:

(a) accessing, using the second node, a node identifier identifying the first node; and

(b) sending additional data from the second node to the first node.

5. A method as claimed in any one of claims 1 to 4 wherein the cluster membership information comprises:

(a) a node identifier uniquely identifying each of the server nodes in the server cluster; and

(b) a cluster identifier uniquely identifying the server cluster to which the server nodes belong.

6. A method as claimed in any one of claims 1 to 5 further comprising:

(a) rebooting one of the server nodes ("rebooted server node") in the server cluster; and

(b) once the rebooted server node returns online, using the rebooted server node to perform a method comprising:

(i) accessing the cluster identifier identifying the server cluster; and

(ii) automatically rejoining the server cluster.

7. A method as claimed in any one of claims 1 to 6 wherein sending the data comprises:

(a) looking up, using the first node, a communication endpoint for the second node from the node identifier; and

(b) sending the data from the first node to the communication endpoint.

8. A method as claimed in claim 7 wherein the communication endpoint and the node identifier comprise entries in a network map relating node identifiers for all the server nodes in the server cluster to corresponding communication endpoints, and wherein each of the server nodes in the server cluster persistently stores its own version of the network map locally.

9. A method as claimed in claim 8 wherein the network map permits each of the server nodes in the server cluster to send the data to any other of the server nodes in the server cluster without using a centralized server.

10. A method as claimed in any one of claims 1 to 9 wherein the data is stored locally on the first node and further comprising modifying the data using the first node, wherein sending the data from the first node to the second node comprises part of

synchronizing the data on the first and second nodes after the first node has modified the data.

11. A method as claimed in claim 10 wherein the data comprises version information generated using a causality versioning mechanism and different versions of the data are stored on the first and second nodes, and wherein synchronizing the data comprises comparing the version information stored on the first and second nodes and adopting on both of the first and second nodes the data whose version information indicates is more recent.

12. A method as claimed in any one of claims 1 to 11 wherein the data comprises the node identifier of the first node, heartbeat state information of the first node, application state information of the first node, and version information, and wherein sending the data comprises disseminating the data to all the server nodes in the server cluster using a gossip protocol that performs data exchanges between pairs of the server nodes in the cluster.

13. A method as claimed in claim 12 wherein the data is periodically disseminated to all the server nodes in the server cluster.

14. A method as claimed in claim 12 or 13 wherein the data is sent to the second node when the first node joins the cluster.

15. A method as claimed in any one of claims 12 to 14 wherein a domain populated with entries that can be modified by any of the server nodes in the server cluster is stored locally on each of the nodes in the server cluster, and further comprising generating the version information using a causality versioning mechanism such that the version information indicates which of the server nodes has most recently modified one of the entries.

16. A method as claimed in claim 15 wherein the application state information comprises a top-level hash generated by hashing all the entries in the domain.

17. A method as claimed in claim 16 further comprising:

(a) comparing, using the second node, the top-level hash with a top-level hash generated by hashing a version of a corresponding domain stored locally on the second node; and

(b) if the top-level hashes differ, synchronizing the domains on both the first and second nodes using the version information.

18. A method as claimed in claim 12 to 17 wherein a status entry that can only be modified by the first node is stored locally on the first node, and wherein the version information comprises a version number that the first node increments whenever it modifies the status entry.

19. A method as claimed in claim 18 wherein the application state information comprises a status entity pair comprising a status entity identifier that identifies the status entry and the version number.

20. A method as claimed in claim 19 further comprising:

(a) comparing, using the second node, the version number received from the first node with a version number of a corresponding status entry stored locally on the second node; and

(b) if the versions numbers differ, updating the status entry stored locally on the second node with the status entry stored locally on the first node.

21. A method as claimed in claim 20 wherein updating the status entry comprises sending from the first node to the second node additional status entries stored locally on the first node that were modified simultaneously with the status entry.

22. A method as claimed in any one of claims 1 to 21 wherein the first and second nodes comprise at least part of a group of server nodes in the cluster to which the first node can send the data in a totally ordered manner to all of the server nodes in the group, and wherein sending the data comprises the first node sending the data to all of the server nodes in the group.

23. A method as claimed in claim 22 wherein the data comprises non-persistent data generated during the runtime of the physical security system.

24. A method as claimed in any one of claims 1 to 23 wherein the data comprises streaming video streamed from another of the server nodes in the server cluster through the first node to the second node.

25. A method as claimed in any one of claims 1 to 24, wherein the other node is the second node.

26. A physical security system, comprising a plurality of server nodes, wherein the physical security system is configured to share data by performing a method comprising:

(a) adding a first server node to a server cluster comprising a second server node, the adding comprising:

(i) exchanging a version of a cluster membership information stored on the first server node with a version of the cluster membership information stored on one of the server nodes that is already part of the server cluster; and

(ii) synchronizing the version of the cluster membership information stored on the first server node with versions of the cluster membership information stored on all the server nodes that, prior to the first server node joining the cluster, comprised part of the cluster;

(b) accessing, after the first node has been added to the server cluster and using the first node, a node identifier identifying another node of the server cluster, wherein the node identifier comprises at least part of cluster membership information identifying all the server nodes in the server cluster and accessible by all the server nodes in the server cluster, and wherein each of the server nodes in the server cluster persistently stores its own version of the cluster membership information locally;

(c) sending the data from the first node to the other node.

27. One or more non-transitory computer readable mediums having encoded thereon statements and instructions to cause one or more processors to perform a method for sharing data in a physical security system that comprises a plurality of server nodes, the method comprising:

(a) adding a first server node to a server cluster comprising a second server node, the adding comprising:

(i) exchanging a version of cluster membership information stored on the first server node with a version of the cluster membership information stored on one of the server nodes that is already part of the server cluster; and

(ii) synchronizing the version of the cluster membership information stored on the first server node with versions of the cluster membership information stored on all the server nodes that, prior to the first server node joining the cluster, comprised part of the cluster;

(b) accessing, after the first node has been added to the server cluster and using the first node, a node identifier identifying another node of the server cluster, wherein the node identifier comprises at least part of cluster membership information identifying all the server nodes in the server cluster and accessible by all the server nodes in the server cluster, and wherein each of the server nodes in the server cluster persistently stores its own version of the cluster membership information locally;

(c) sending the data from the first node to the other node.

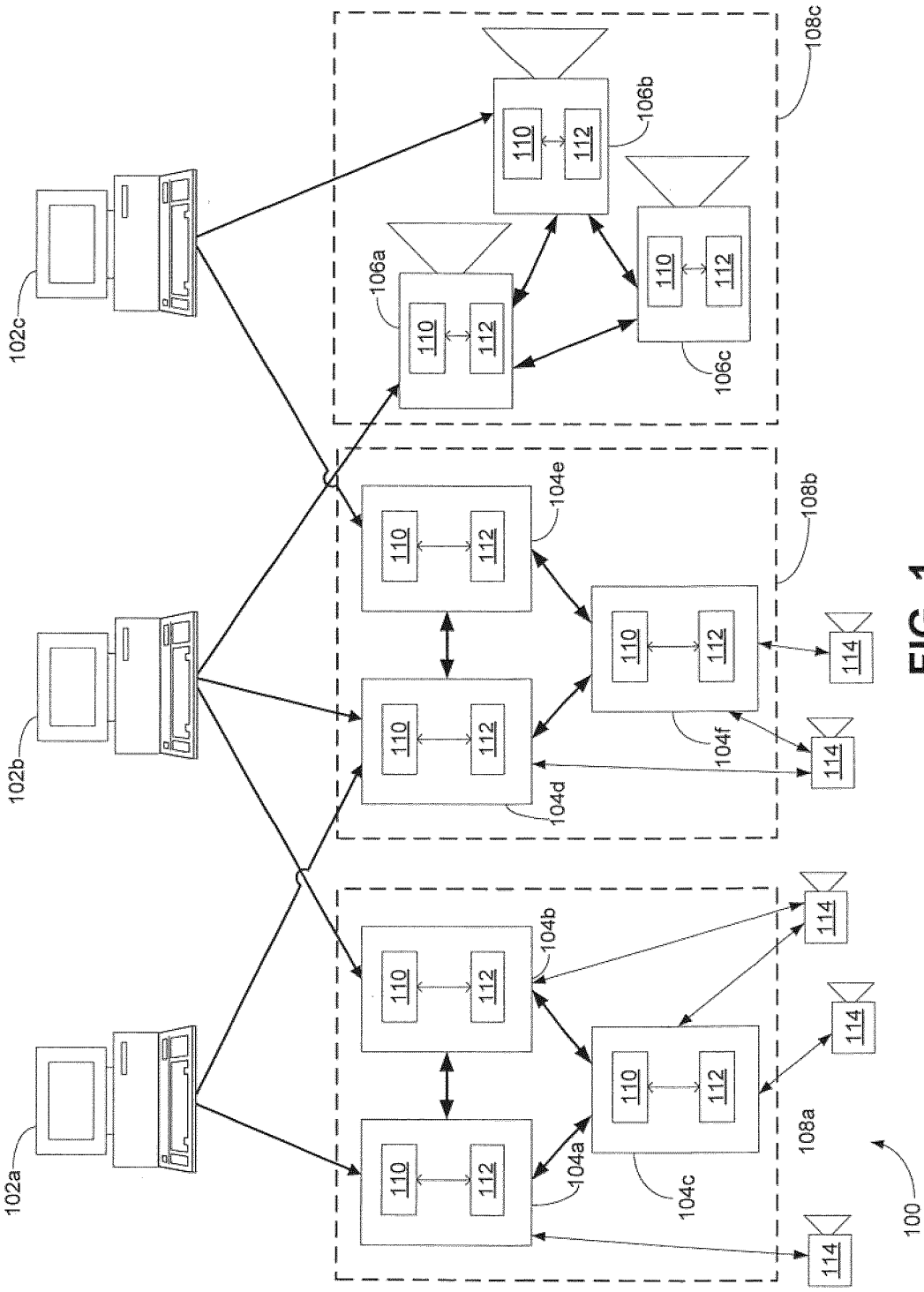
102a
102b
102c
100
108a
108b
108c
104a
104b
104c
104d
104e
104f
106a
106b
106c
110
112
114

FIG. 1

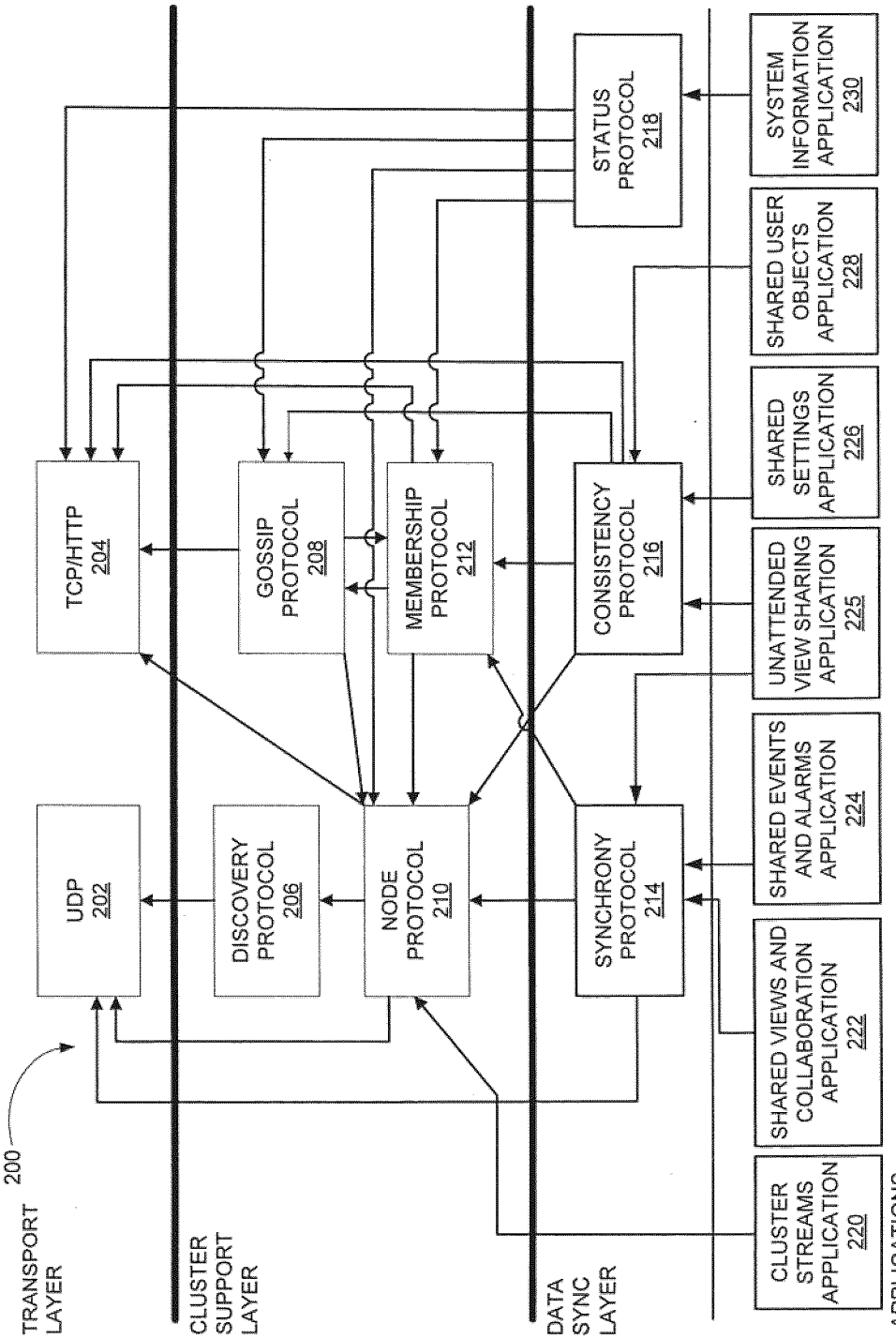
200
TRANSPORT LAYER
CLUSTER SUPPORT LAYER
DATA SYNC LAYER
APPLICATIONS
UDP 202
TCP/HTTP 204
DISCOVERY PROTOCOL 206
GOSSIP PROTOCOL 208
NODE PROTOCOL 210
MEMBERSHIP PROTOCOL 212
SYNCHRONY PROTOCOL 214
CONSISTENCY PROTOCOL 216
STATUS PROTOCOL 218
CLUSTER STREAMS APPLICATION 220
SHARED VIEWS AND COLLABORATION APPLICATION 222
SHARED EVENTS AND ALARMS APPLICATION 224
UNATTENDED VIEW SHARING APPLICATION 225
SHARED SETTINGS APPLICATION 226
SHARED USER OBJECTS APPLICATION 228
SYSTEM INFORMATION APPLICATION 230

FIG. 2

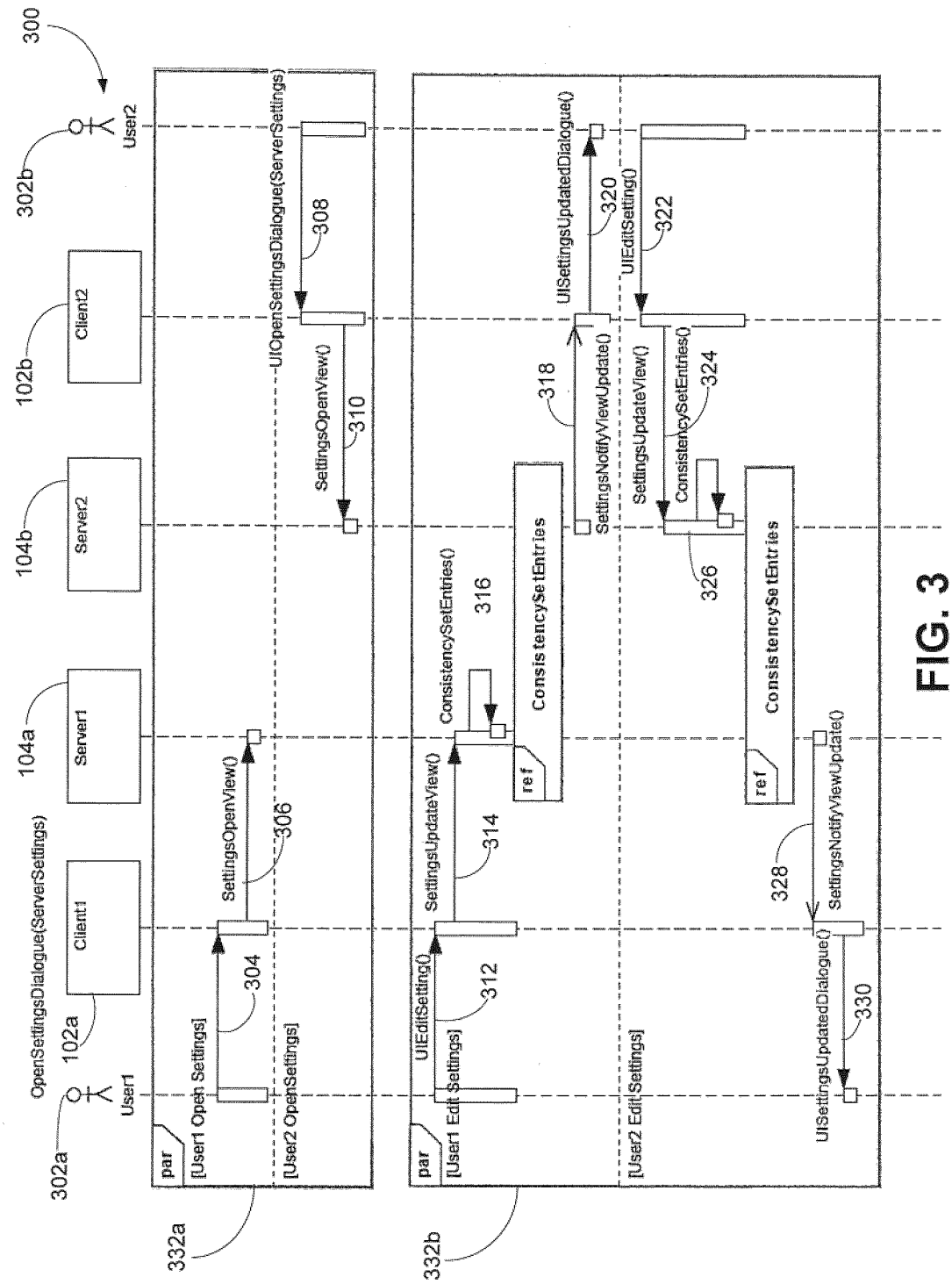
300
User2
302b
Client2
102b
Server2
104b
Server1
104a
Client1
OpenSettingsDialogue(ServerSettings)
102a
User1
302a
par
[User1 Open Settings]
SettingsOpenView()
304
306
[User2 OpenSettings]
UIOpenSettingsDialogue(ServerSettings)
308
SettingsOpenView()
310
332a
par
[User1 Edit Settings]
UIEditSetting()
312
SettingsUpdateView()
314
ConsistencySetEntries()
316
ref
ConsistencySetEntries
SettingsNotifyViewUpdate()
318
UISettingsUpdatedDialogue()
320
[User2 Edit Settings]
UIEditSetting()
322
SettingsUpdateView()
324
ConsistencySetEntries()
326
ref
ConsistencySetEntries
328
SettingsNotifyViewUpdate()
UISettingsUpdatedDialogue()
330
332b

FIG. 3

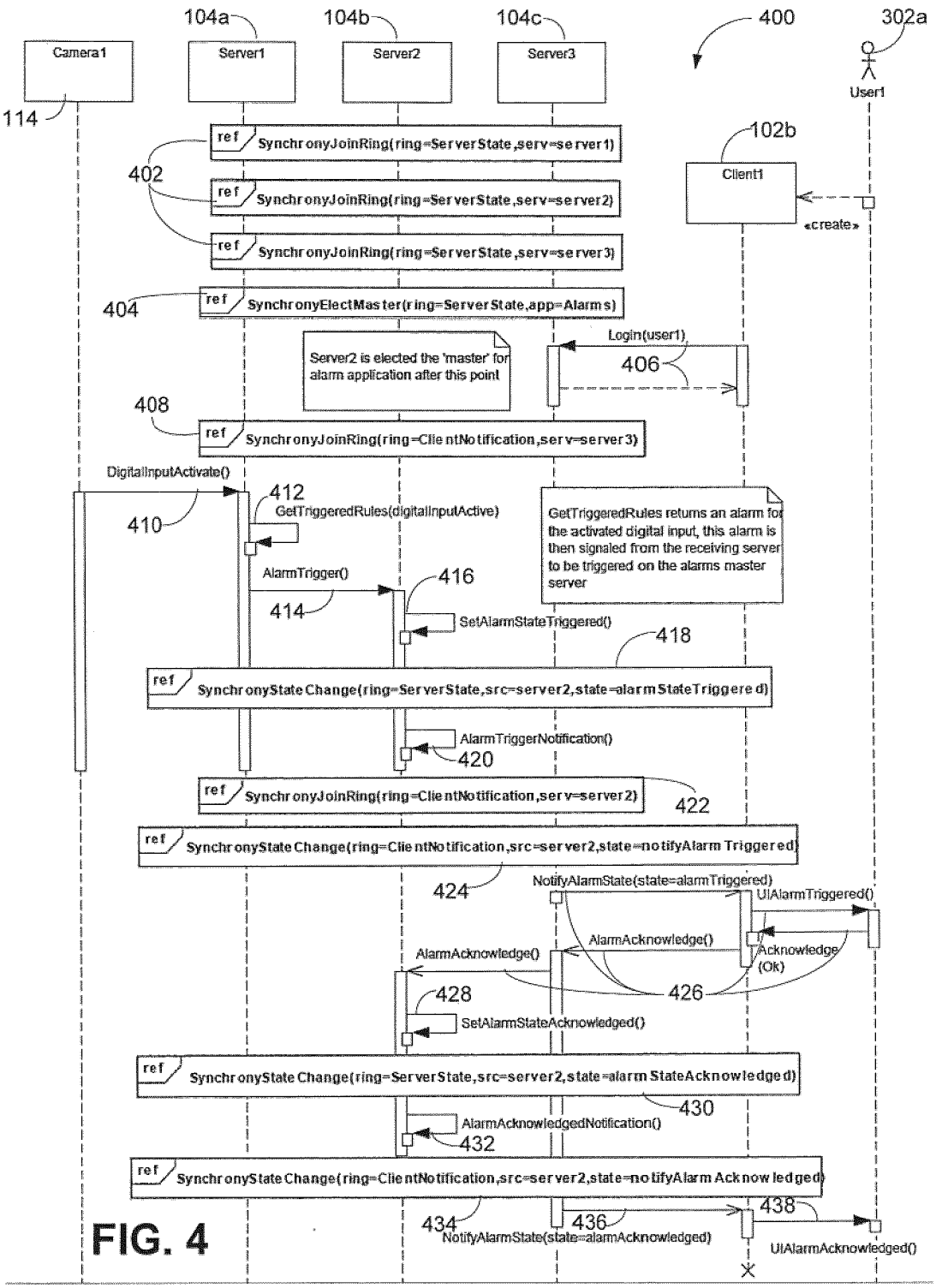
104a
104b
104c
400
302a
Camera1
Server1
Server2
Server3
User1
114
ref SynchronyJoinRing(ring=ServerState,serv=server1)
102b
Client1
402
ref SynchronyJoinRing(ring=ServerState,serv=server2)
«create»
ref SynchronyJoinRing(ring=ServerState,serv=server3)
404
ref SynchronyElectMaster(ring=ServerState,app=Alarms)
Login(user1)
Server2 is elected the 'master' for alarm application after this point
406
408
ref SynchronyJoinRing(ring=ClientNotification,serv=server3)
DigitalInputActivate()
412
GetTriggeredRules(digitalInputActive)
410
GetTriggeredRules returns an alarm for the activated digital input, this alarm is then signaled from the receiving server to be triggered on the alarms master server
AlarmTrigger()
416
414
SetAlarmStateTriggered()
418
ref SynchronyStateChange(ring=ServerState,src=server2,state=alarmStateTriggered)
AlarmTriggerNotification()
420
ref SynchronyJoinRing(ring=ClientNotification,serv=server2)
422
ref SynchronyStateChange(ring=ClientNotification,src=server2,state=notifyAlarmTriggered)
424
NotifyAlarmState(state=alarmTriggered)
UIAlarmTriggered()
AlarmAcknowledge()
AlarmAcknowledge()
Acknowledge (Ok)
428
426
SetAlarmStateAcknowledged()
ref SynchronyStateChange(ring=ServerState,src=server2,state=alarmStateAcknowledged)
430
AlarmAcknowledgedNotification()
432
ref SynchronyStateChange(ring=ClientNotification,src=server2,state=notifyAlarmAcknowledged)
434
436
438
NotifyAlarmState(state=alarmAcknowledged)
UIAlarmAcknowledged()

FIG. 4

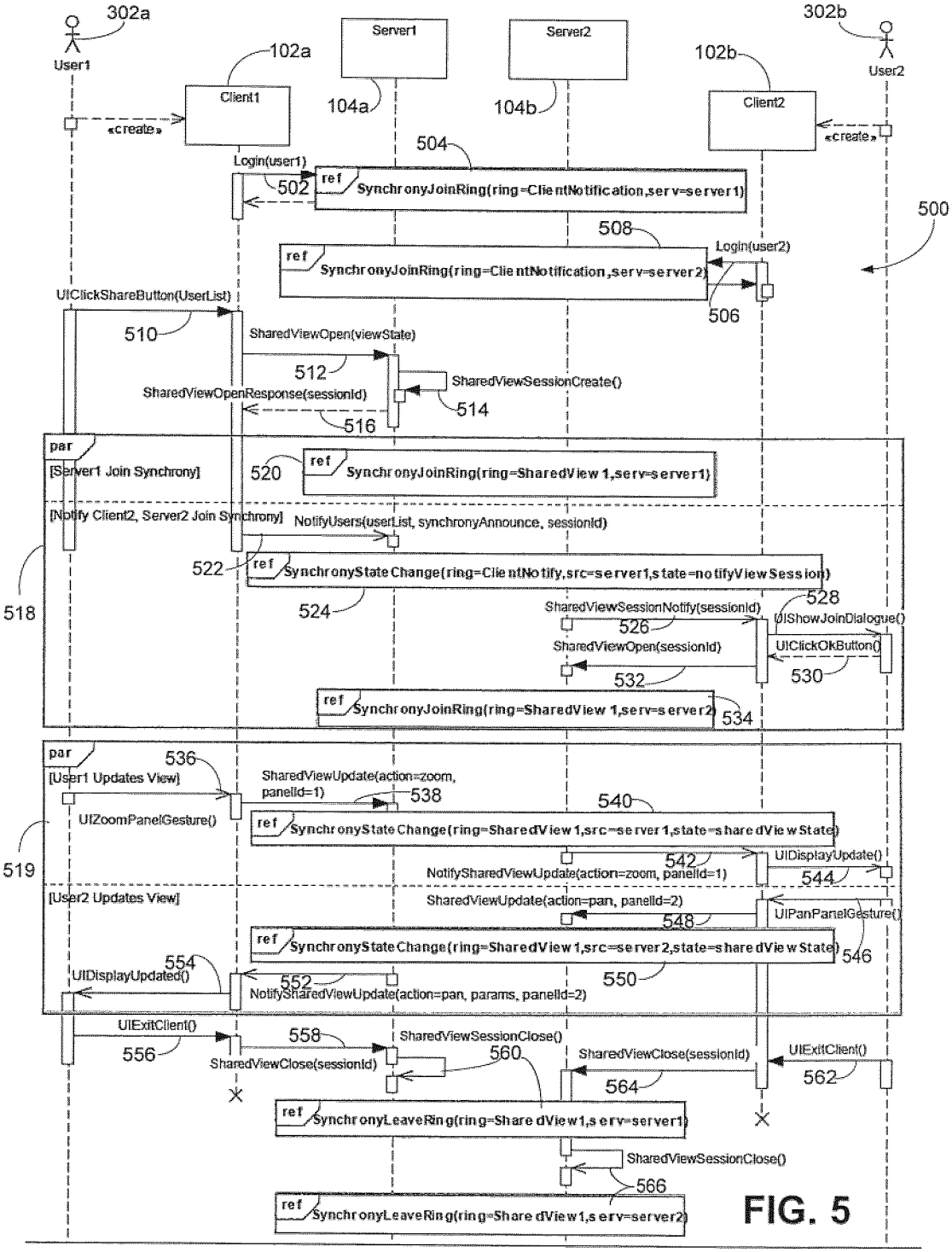
302a
User1
102a
Client1
Server1
104a
Server2
104b
102b
Client2
302b
User2
«create»
«create»
504
Login(user1)
502
ref
SynchronyJoinRing(ring=ClientNotification,serv=server1)
500
508
ref
SynchronyJoinRing(ring=ClientNotification,serv=server2)
Login(user2)
506
UIClickShareButton(UserList)
510
SharedViewOpen(viewState)
512
SharedViewSessionCreate()
514
SharedViewOpenResponse(sessionId)
516
par
[Server1 Join Synchrony]
520
ref
SynchronyJoinRing(ring=SharedView1,serv=server1)
[Notify Client2, Server2 Join Synchrony]
NotifyUsers(userList, synchronyAnnounce, sessionId)
518
522
ref
SynchronyStateChange(ring=ClientNotify,src=server1,state=notifyViewSession)
524
SharedViewSessionNotify(sessionId)
526
528
UIShowJoinDialogue()
SharedViewOpen(sessionId)
532
UIClickOkButton()
530
ref
SynchronyJoinRing(ring=SharedView1,serv=server2)
534
par
[User1 Updates View]
536
SharedViewUpdate(action=zoom, panelId=1)
538
UIZoomPanelGesture()
519
540
ref
SynchronyStateChange(ring=SharedView1,src=server1,state=sharedViewState)
542
UIDisplayUpdate()
NotifySharedViewUpdate(action=zoom, panelId=1)
544
[User2 Updates View]
SharedViewUpdate(action=pan, panelId=2)
548
UIPanPanelGesture()
ref
SynchronyStateChange(ring=SharedView1,src=server2,state=sharedViewState)
546
554
UIDisplayUpdated()
552
550
NotifySharedViewUpdate(action=pan, params, panelId=2)
UIExitClient()
556
558
SharedViewSessionClose()
SharedViewClose(sessionId)
560
SharedViewClose(sessionId)
564
UIExitClient()
562
ref
SynchronyLeaveRing(ring=SharedView1,serv=server1)
SharedViewSessionClose()
566
ref
SynchronyLeaveRing(ring=SharedView1,serv=server2)

FIG. 5

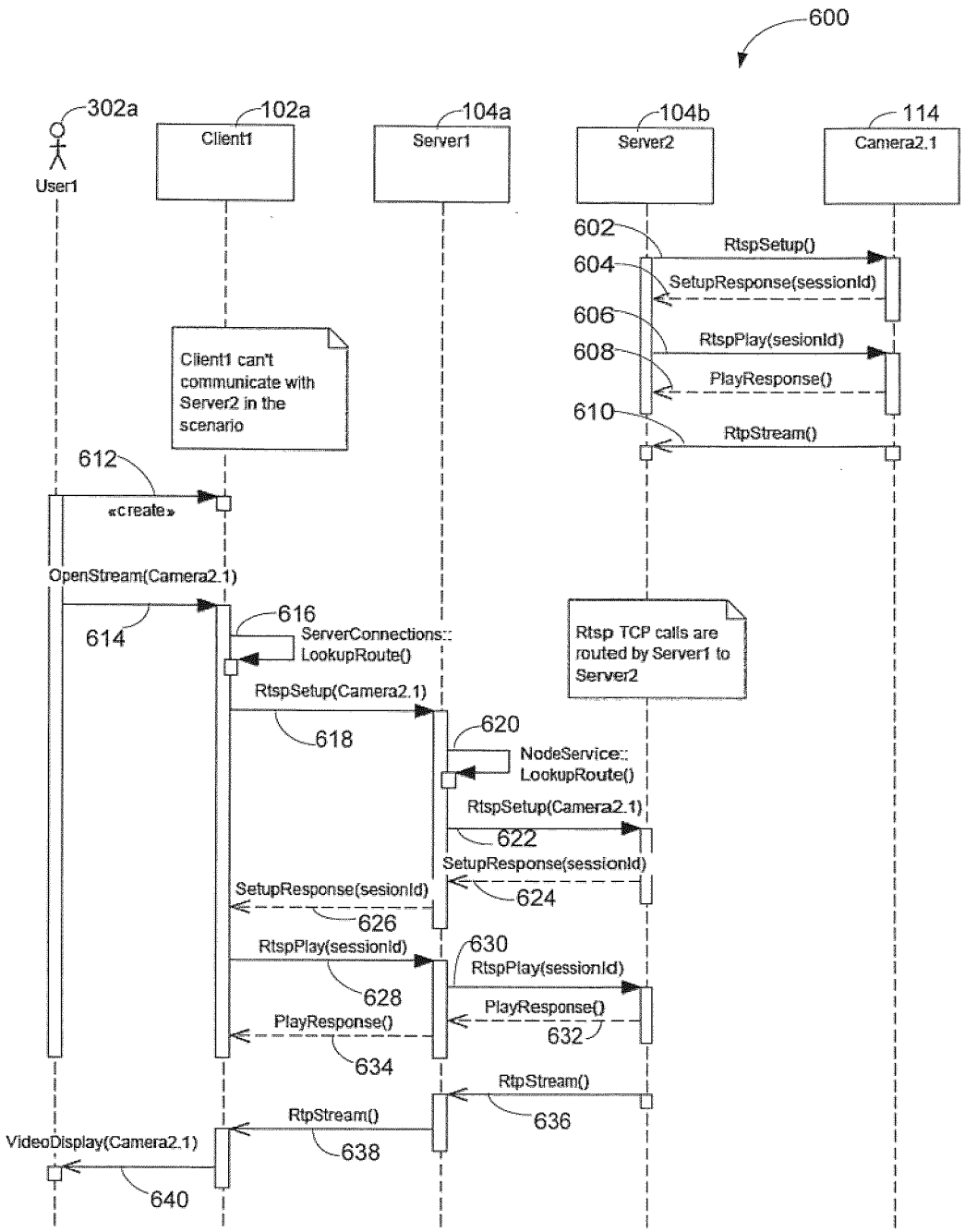
600
302a
User1
102a
Client1
104a
Server1
104b
Server2
114
Camera2.1
602
RtspSetup()
604
SetupResponse(sessionId)
606
RtspPlay(sesionId)
608
PlayResponse()
610
RtpStream()
Client1 can't communicate with Server2 in the scenario
612
«create»
OpenStream(Camera2.1)
614
616
ServerConnections:: LookupRoute()
Rtsp TCP calls are routed by Server1 to Server2
RtspSetup(Camera2.1)
618
620
NodeService:: LookupRoute()
RtspSetup(Camera2.1)
622
SetupResponse(sessionId)
624
SetupResponse(sesionId)
626
RtspPlay(sessionId)
628
630
RtspPlay(sessionId)
PlayResponse()
632
PlayResponse()
634
RtpStream()
636
RtpStream()
638
VideoDisplay(Camera2.1)
640

FIG. 6

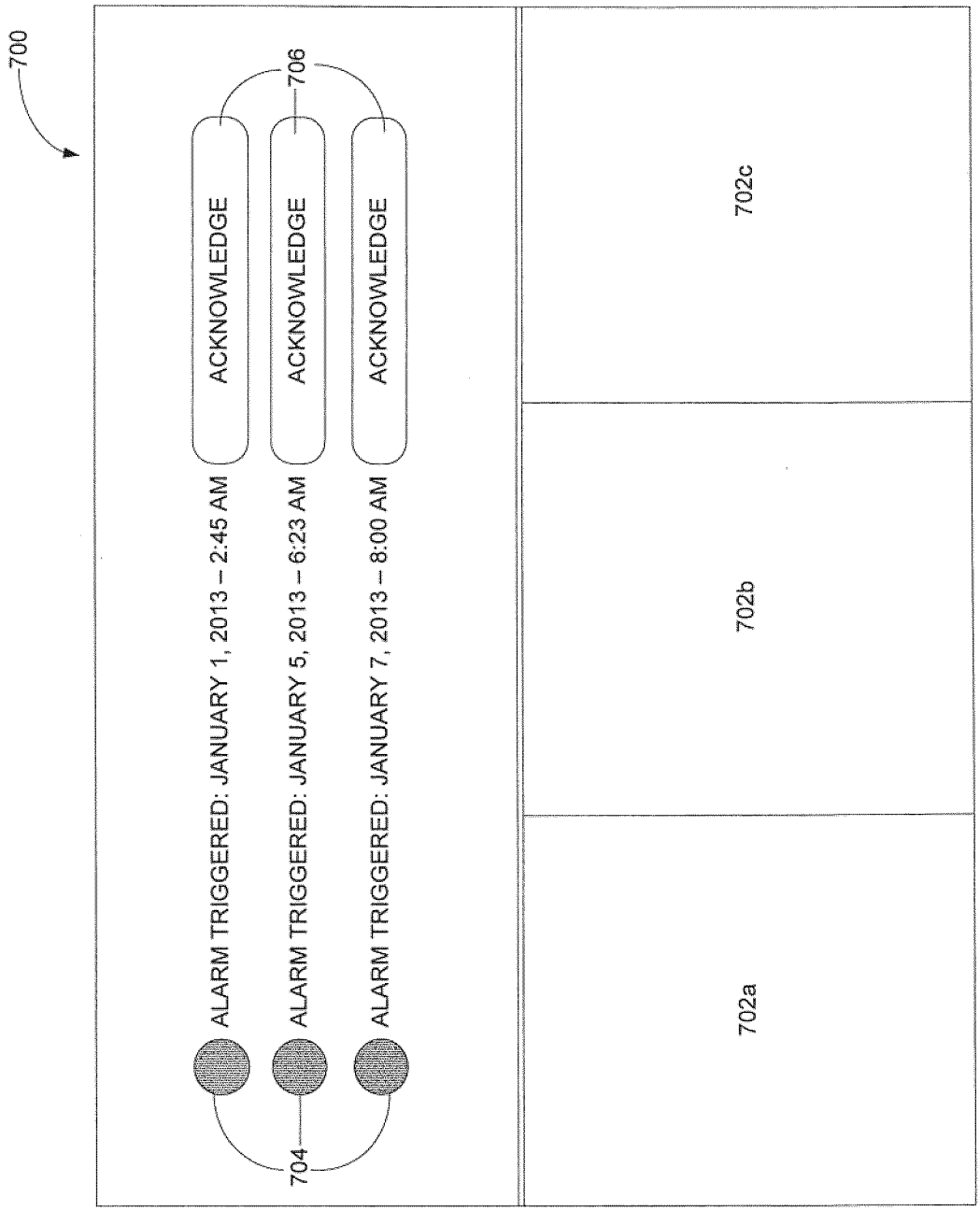
700
ALARM TRIGGERED: JANUARY 1, 2013 – 2:45 AM
ALARM TRIGGERED: JANUARY 5, 2013 – 6:23 AM
ALARM TRIGGERED: JANUARY 7, 2013 – 8:00 AM
ACKNOWLEDGE
ACKNOWLEDGE
ACKNOWLEDGE
704
706
702a
702b
702c

FIG. 7

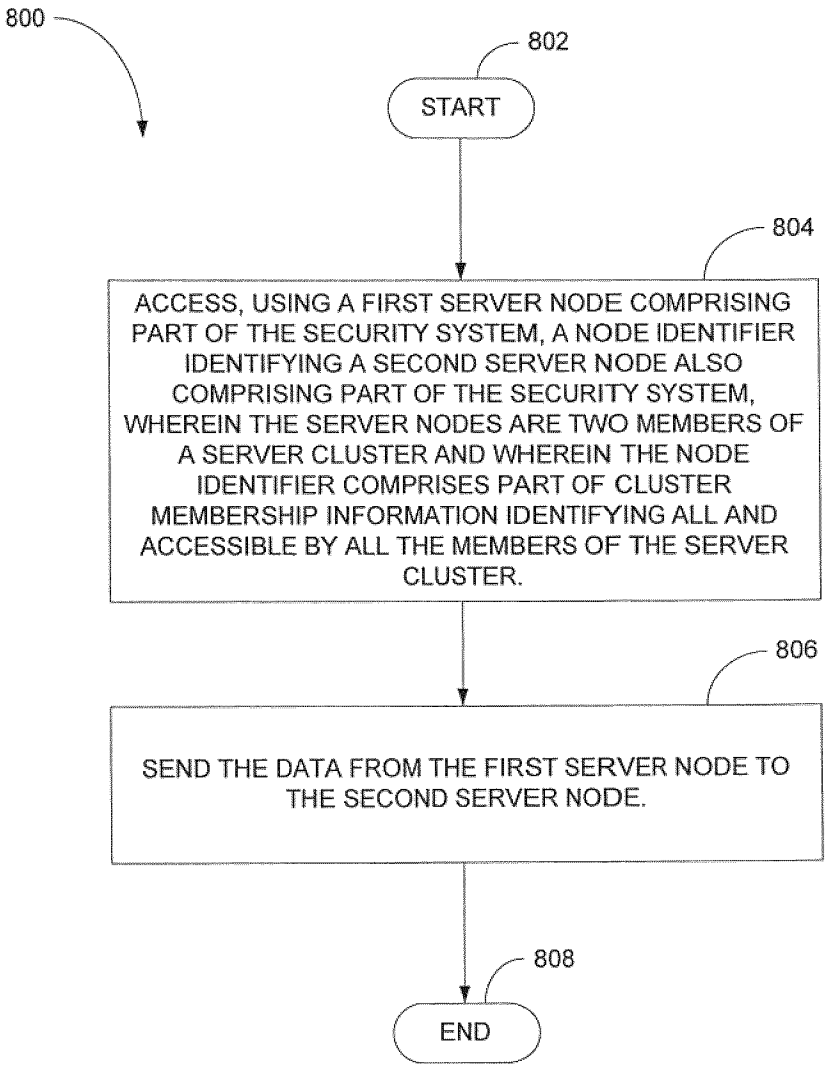
800
802
START
804
ACCESS, USING A FIRST SERVER NODE COMPRISING PART OF THE SECURITY SYSTEM, A NODE IDENTIFIER IDENTIFYING A SECOND SERVER NODE ALSO COMPRISING PART OF THE SECURITY SYSTEM, WHEREIN THE SERVER NODES ARE TWO MEMBERS OF A SERVER CLUSTER AND WHEREIN THE NODE IDENTIFIER COMPRISES PART OF CLUSTER MEMBERSHIP INFORMATION IDENTIFYING ALL AND ACCESSIBLE BY ALL THE MEMBERS OF THE SERVER CLUSTER.
806
SEND THE DATA FROM THE FIRST SERVER NODE TO THE SECOND SERVER NODE.
808
END

FIG. 8

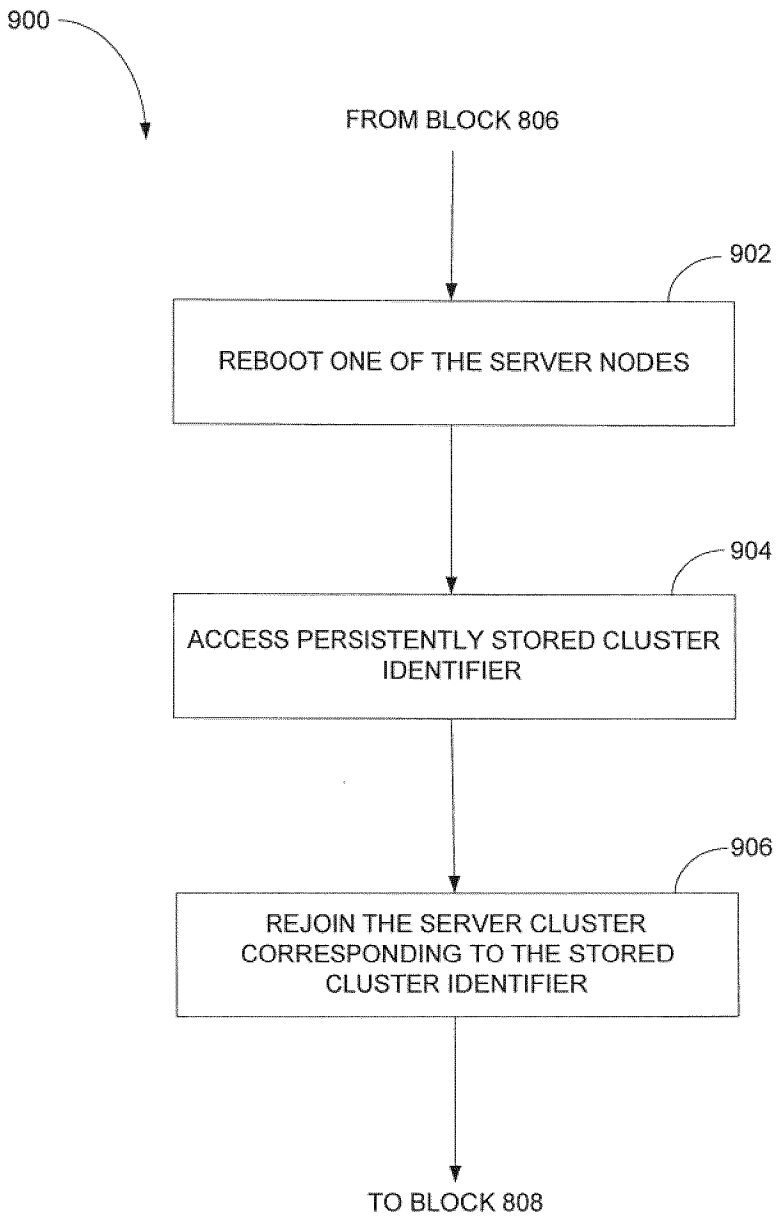
900
FROM BLOCK 806
902
REBOOT ONE OF THE SERVER NODES
904
ACCESS PERSISTENTLY STORED CLUSTER IDENTIFIER
906
REJOIN THE SERVER CLUSTER CORRESPONDING TO THE STORED CLUSTER IDENTIFIER
TO BLOCK 808

FIG. 9

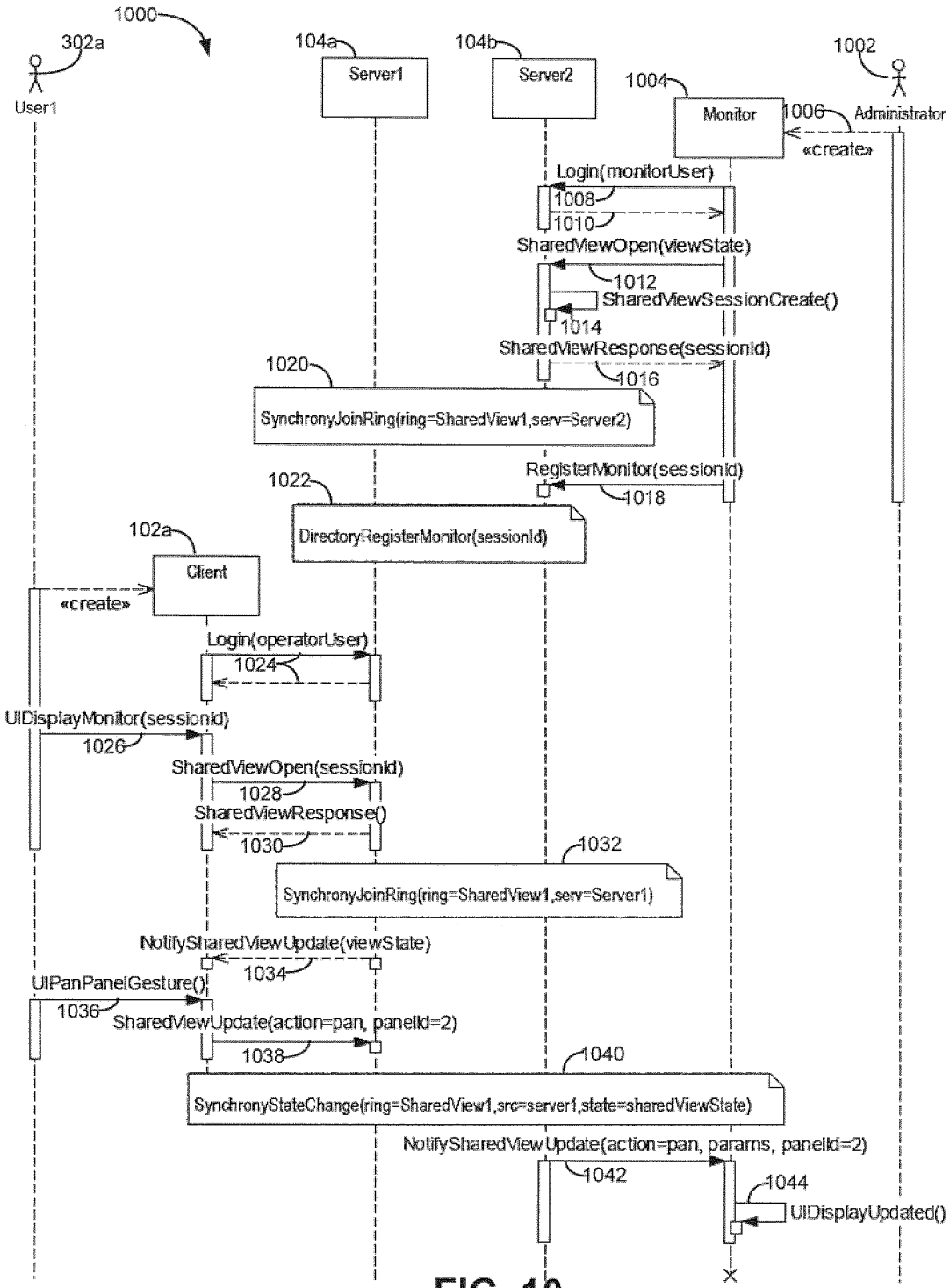
1000
302a
User1
104a
Server1
104b
Server2
1004
Monitor
1002
Administrator
1006
«create»
Login(monitorUser)
1008
1010
SharedViewOpen(viewState)
1012
SharedViewSessionCreate()
1014
SharedViewResponse(sessionId)
1016
1020
SynchronyJoinRing(ring=SharedView1,serv=Server2)
RegisterMonitor(sessionId)
1018
1022
DirectoryRegisterMonitor(sessionId)
102a
Client
«create»
Login(operatorUser)
1024
UIDisplayMonitor(sessionId)
1026
SharedViewOpen(sessionId)
1028
SharedViewResponse()
1030
1032
SynchronyJoinRing(ring=SharedView1,serv=Server1)
NotifySharedViewUpdate(viewState)
1034
UIPanPanelGesture()
1036
SharedViewUpdate(action=pan, panelId=2)
1038
1040
SynchronyStateChange(ring=SharedView1,src=server1,state=sharedViewState)
NotifySharedViewUpdate(action=pan, params, panelId=2)
1042
1044
UIDisplayUpdated()

FIG. 10

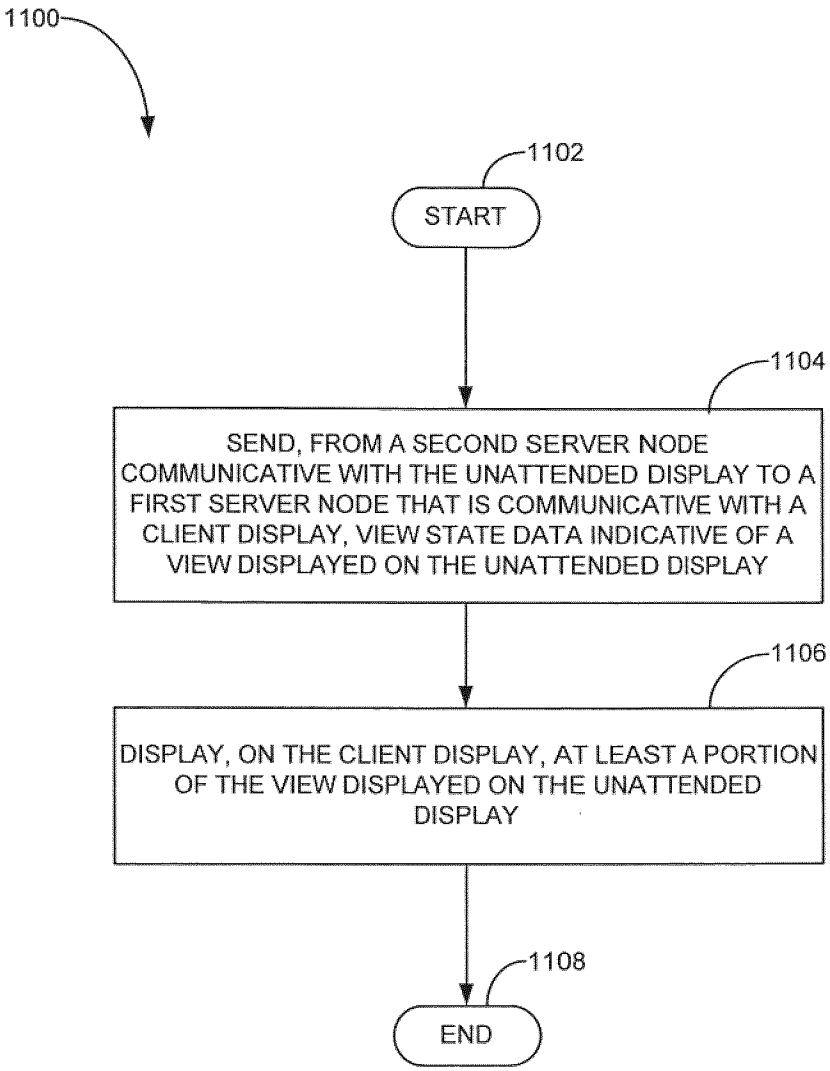
1100
1102
START
1104
SEND, FROM A SECOND SERVER NODE COMMUNICATIVE WITH THE UNATTENDED DISPLAY TO A FIRST SERVER NODE THAT IS COMMUNICATIVE WITH A CLIENT DISPLAY, VIEW STATE DATA INDICATIVE OF A VIEW DISPLAYED ON THE UNATTENDED DISPLAY
1106
DISPLAY, ON THE CLIENT DISPLAY, AT LEAST A PORTION OF THE VIEW DISPLAYED ON THE UNATTENDED DISPLAY
1108
END

FIG. 11

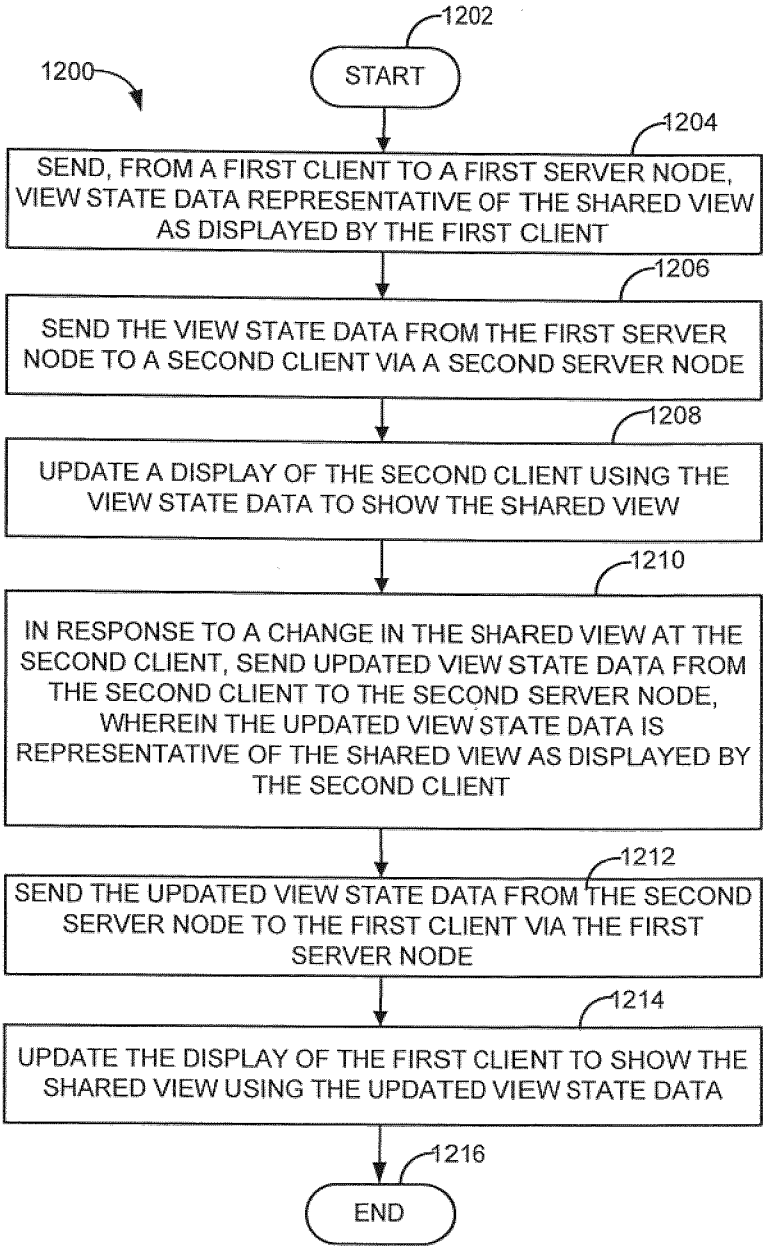
1200
1202
START
1204
SEND, FROM A FIRST CLIENT TO A FIRST SERVER NODE, VIEW STATE DATA REPRESENTATIVE OF THE SHARED VIEW AS DISPLAYED BY THE FIRST CLIENT
1206
SEND THE VIEW STATE DATA FROM THE FIRST SERVER NODE TO A SECOND CLIENT VIA A SECOND SERVER NODE
1208
UPDATE A DISPLAY OF THE SECOND CLIENT USING THE VIEW STATE DATA TO SHOW THE SHARED VIEW
1210
IN RESPONSE TO A CHANGE IN THE SHARED VIEW AT THE SECOND CLIENT, SEND UPDATED VIEW STATE DATA FROM THE SECOND CLIENT TO THE SECOND SERVER NODE, WHEREIN THE UPDATED VIEW STATE DATA IS REPRESENTATIVE OF THE SHARED VIEW AS DISPLAYED BY THE SECOND CLIENT
1212
SEND THE UPDATED VIEW STATE DATA FROM THE SECOND SERVER NODE TO THE FIRST CLIENT VIA THE FIRST SERVER NODE
1214
UPDATE THE DISPLAY OF THE FIRST CLIENT TO SHOW THE SHARED VIEW USING THE UPDATED VIEW STATE DATA
1216
END

FIG. 12