



(51) International Patent Classification:

G06F 11/30 (2006.01) *G06F 21/00* (2013.01)
G06F 9/44 (2006.01) *G06F 15/16* (2006.01)
G06F 9/46 (2006.01)

(21) International Application Number:

PCT/US2016/029005

(22) International Filing Date:

22 April 2016 (22.04.2016)

(25) Filing Language:

English

(26) Publication Language:

English

(30) Priority Data:

62/151,188	22 April 2015 (22.04.2015)	US
62/193,930	17 July 2015 (17.07.2015)	US
62/193,927	17 July 2015 (17.07.2015)	US
62/214,124	3 September 2015 (03.09.2015)	US

(71) Applicant: **LARC NETWORKS, INC.** [US/US]; 4 Main Street, Suite 60, Los Alton, CA 94022 (US).

(72) Inventors: **KACIN, Martin**; C/o Larc Networks, Inc., 4 Main Street, Suite 60, Los Altos, CA 94022 (US). **GRAY, Michael**; C/o Larc Networks, Inc., 4 Main Street, Suite 60, Los Altos, CA 94022 (US).

(74) Agents: **HOFFMAN, Brian M.** et al.; Fenwick & West LLP, 801 California Street, Mountain View, CA 94041 (US).

(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, IR, IS, JP, KE, KG, KN, KP, KR, KZ, LA, LC, LK, LR, LS, 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, ST, 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: DEAD DROP NETWORK ARCHITECTURE

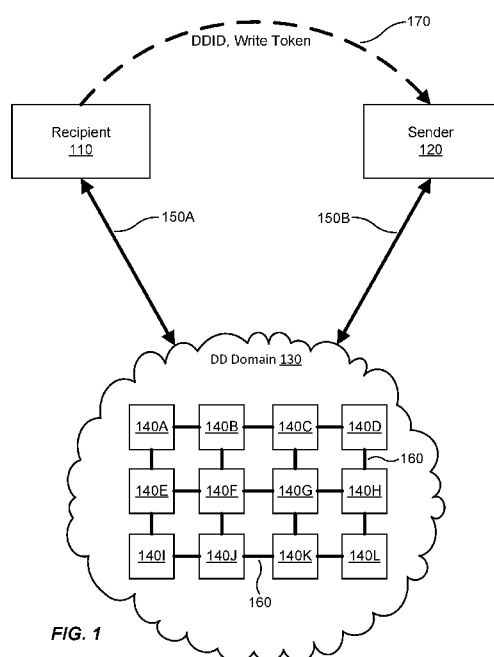


FIG. 1

(57) Abstract: A dead drop at a node in a dead drop (DD) domain performs actions responsive to detecting events in the DD domain. The node receives a notification request specifying an event to be monitored, the notification request including a dead drop identifier (DDID) referencing a storage location in the DD domain associated with the specified event and a token associated with the DDID. The node further specifies the action to perform in response to detecting the occurrence of the specified event. The node monitors for an occurrence of a specified event within the DD domain. The node detects the occurrence of the specified event within the DD domain. The node further performs a specified action in response to detecting the occurrence of the specified event within the DD domain.

DEAD DROP NETWORK ARCHITECTURE

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] This application claims the benefit of U.S. Provisional Application No. 62/151,188, filed April 22, 2015, which is incorporated by reference herein. This application claims the benefit of U.S. Provisional Application No. 62/193,927, filed July 17, 2015, which is incorporated by reference herein. This application claims the benefit of U.S. Provisional Application No. 62/193,930, filed July 17, 2015, which is incorporated by reference herein. This application claims the benefit of U.S. Provisional Application No. 62/214,124, filed September 3, 2015, which is incorporated by reference herein.

BACKGROUND

1. FIELD OF ART

[0002] The present invention generally relates to the field of computer networking and data storage and in particular to a network architecture for facilitating secure data exchange over a decentralized computer network and data storage architecture.

2. BACKGROUND OF THE INVENTION

[0003] The Internet (including the Web) enables users of computers to quickly and easily exchange data. There is a wide range of applications that leverage this ability to exchange data to achieve powerful results for individuals and enterprises alike. Examples include email, file sharing, home automation, entertainment, data management, and more.

[0004] However, the way that data is exchanged over the Internet makes the data, and those who send the data, vulnerable to malicious actors. For instance, data moving between parties or stored on a remote server typically include information associated with the sender and the recipient. Accordingly, an interceptor of the data may associate the data with the parties. If the data contain sensitive information, it may leave the parties open to identity theft or other malicious acts. As a result, many users are discouraged from sharing important information via the Internet, thereby missing out on many of the advantages that are afforded to computer users.

SUMMARY OF THE INVENTION

[0005] According to embodiments of the invention, a method for performing an action responsive to detecting an event in a dead drop (DD) domain is described. The method includes monitoring, by a node in the DD domain, for an occurrence of a specified event within the DD domain. The method further includes detecting the occurrence of the specified event within the DD domain. The method further includes performing a specified action in response to detecting the occurrence of the specified event within the DD domain.

[0006] According to embodiments of the invention, a system for exchanging data between a sender and a recipient is described. The system includes a processor for executing computer program instructions. The system also includes a non-transitory computer-readable storage medium storing computer program instructions executable by the processor to perform steps. The steps include monitoring, by a node in the DD domain, for an occurrence of a specified event within the DD domain. The steps further include detecting the occurrence of the specified event within the DD domain. The steps further include performing a specified action in response to detecting the occurrence of the specified event within the DD domain.

[0007] According to embodiments of the invention, a non-transitory computer-readable storage medium storing computer program instructions for exchanging data between a sender and a recipient. The computer program instructions are executable to perform steps. The steps include monitoring, by a node in the DD domain, for an occurrence of a specified event within the DD domain. The steps further include detecting the occurrence of the specified event within the DD domain. The steps further include performing a specified action in response to detecting the occurrence of the specified event within the DD domain.

BRIEF DESCRIPTION OF DRAWINGS

[0008] FIG. 1 is a high-level block diagram illustrating an example of passing data using a dead drop network architecture according to one embodiment.

[0009] FIG. 2 is a high-level block diagram illustrating a detailed view of the dead drop domain of FIG. 1 according to one embodiment.

[0010] FIG. 3 is a high-level block diagram illustrating an example of a dead drop storage node according to one embodiment.

[0011] FIG. 4 is a flowchart illustrating steps for using a dead drop to pass data from a sender to a recipient according to one embodiment.

[0012] FIG. 5 is a flowchart illustrating steps for using a DD to monitor events and perform actions if an event is detected according to one embodiment.

[0013] FIG. 6 is a high-level block diagram illustrating an example of two nodes establishing a direct connection according to one embodiment.

[0014] FIG. 7 is a high-level block diagram illustrating physical components of a computer used as part or all of one or more of the entities described herein in one embodiment.

DETAILED DESCRIPTION

[0015] The Figures (FIGS.) and the following description describe certain embodiments by way of illustration only. One skilled in the art will readily recognize from the following description that alternative embodiments of the structures and methods illustrated herein may be employed without departing from the principles described herein. Reference will now be made to several embodiments, examples of which are illustrated in the accompanying figures.

[0016] It is noted that wherever practicable similar or like reference numbers may be used in the figures and may indicate similar or like functionality. This description occasionally uses reference numbers in combination with letters to designate items illustrated in the figures. Herein, a reference number used without an accompanying letter (e.g., “150”) references any or all instances of the designated item, while a reference number used with an accompanying letter (e.g., “150A”) refers to the specific item designated with that label in the figure.

[0017] FIG. 1 is a high-level block diagram illustrating an example of passing data using a dead drop network architecture according to one embodiment. FIG. 1 illustrates a recipient 110 in communication with a sender 120 via a dead drop (DD) domain 130. FIG. 1 describes a unidirectional data pass between a single sender 120 and a single recipient 110. Embodiments can have multiple senders 120 and recipients 110 engaged in bidirectional one-to-one and one-to-many communications.

[0018] Briefly, the recipient 110 uses the DD domain 130 to establish a communication channel that can be used to pass data to the recipient. The recipient 110 provides the sender 120 with a dead drop identifier (DDID) referencing a storage location within the DD domain. The sender 120, in turn, uses the DDID to pass data (e.g., send a message) to the recipient 110 via the DD domain 130. The data transmission from the sender 120 to the recipient 110

is secure in the sense that it is extremely difficult for a malicious actor or other third party to locate, intercept, or decipher the data. Thus, the DD network architecture is suited to communications where security and privacy concerns are paramount. In addition, the DD network architecture may be used to provide enhanced security for general purpose communications.

[0019] In one embodiment, the recipient 110 includes software executing on a computer used by a user (e.g., a person) to perform tasks such as communicating with other users via the DD domain 130 or via other communication networks. For example, the recipient 110 may include software executing on a desktop, notebook, or tablet computer, or another electronic device with computer functionality, such as a mobile telephone, music player, television set-top-box, home automation component, industrial equipment or connected appliance. The recipient 110 may include an input device such as a keyboard or touch-sensitive display that allows for the input of data and an output device such as a display or speaker that allows for the output of data. Functionality enabling the recipient 110 to communicate via the DD domain 130 may be embedded into the hardware of the recipient 110 and/or included in software executed by the recipient 110.

[0020] Similarly, the sender 120 includes a computer used by a user to perform tasks including communicating with other users via the DD domain 130 or via other communication networks. The sender 120 may include the same components as the recipient 110. In fact, the sender 120 may act as a recipient 110 and vice versa, depending upon the direction of data flow in a given communication transaction. The users who respectively use the recipient 110 and sender 120 to communicate can be different people or the same person.

[0021] The recipient 110 and sender 120 are connected to the DD domain 130 via respective communications links 150A, 150B. The communication links 150 may include network communication links using conventional computer networking technologies. For example, the communications links 150 may use wired or wireless network communications technologies such as the wired and wireless forms of Ethernet. Likewise, the communication links 150 may use other communications technologies designed to support communication with local peripherals, such as Universal Serial Bus (USB), Thunderbolt, Bluetooth, Personal Area Network (PAN), Serial ATA, infrared, heat signatures, and/or sound. The communications links 150 may be encrypted using any encryption technologies such as secure sockets layer (SSL), transport layer security (TLS), HTTP Secure (HTTPS), virtual private networks (VPNs), Internet Protocol security (IPsec), etc. In another embodiment,

communication uses custom and/or dedicated data communications technologies instead of, or in addition to, the ones described above.

[0022] The DD domain 130 is a collection of one or more DD nodes 140 (labeled as nodes 140A-L in FIG. 1). A DD node 140 includes functionality for acting in the DD domain 130 and a memory for storing data within the domain. A typical DD domain 130 includes many DD nodes 140. Each node 140 is connected to one or more other nodes via DD communication links 160. The DD communication links 160 may use the same communication technologies as the communication links 150 used by the recipient 110 and sender 120 to connect to the DD domain 130. In one embodiment, the DD nodes 140 and DD communication links 160 are arranged within the DD domain 130 such that every node is reachable by every other node. In another embodiment, the DD nodes 140 are logically or physically partitioned so that some nodes cannot reach other nodes. The path connecting two DD nodes 140 may pass through one or more intermediate nodes. In addition, the recipient 150A and sender 150B communication links respectively connect the recipient 110 and the sender 120 to at least one DD node 140 within the DD domain 130. The recipient 110 and sender 120 may also communicate with each other using other communication links that do not pass through the DD domain 130, such as a communications link 170.

[0023] To receive data using the DD domain 130, the recipient 110 sends a request to the DD domain 130 to create a DD on behalf of the recipient. A DD node 140 within the domain 130 receives the create request and either services the request or selects another node to service the request. For example, the DD node 140 that receives the request may randomly select another node within the DD domain 130 and pass the request to the selected node. The random selection may occur in a manner such that the node that receives the request does not know which node ultimately services the request.

[0024] The node 140 that services the request to create the DD establishes a DDID that uniquely identifies the created DD. In addition, the node 140 establishes a set of tokens associated with the DDID. A token describes the access rights a possessor of the token has with respect to the created DD. For example, an embodiment includes a read token giving a possessor of the token the right to read from the DD identified by the associated DDID and a write token giving the right to write to the DD identified by the associated DDID. The node 140 that services the request provides the DDID and the associated tokens to the recipient 110.

[0025] The recipient 110 typically stores the DDID and associated tokens in a secure

manner. For example, the recipient 110 may store the DDID and tokens in an encrypted data store at the recipient. The recipient 110 provides the DDID and the write token to the sender 120 via a communications link 170. This communications link 170 may be a secure or unsecure link, and may include communication over the Internet and/or dedicated communications links. For example, the recipient 110 may use encrypted or unencrypted email to send the DDID and write token to the sender 120. Alternatively, the recipient may use a different electronic communications technique, such as short-range wireless communications, or even use non-electronic techniques to exchange the information (e.g., a pen and paper). In one embodiment, the DDID and one or more tokens are combined and may be encrypted or encoded (e.g., by a hashing function) to form a single code. In this embodiment, the recipient 110 may share the code with the sender 120 instead of sharing the DDID and write token separately. The code may be decoded, for example by the sender 120 or at a DD node 140, to determine the DDID and token.

[0026] In addition, the recipient 110 and sender 120 may choose to encrypt the data sent by the sender using one or more symmetric or asymmetric encryption techniques. The recipient 110 and sender 120 may choose to exchange encryption keys, if necessary, at the same time the recipient 110 provides the DDID and write token to the sender 120. Alternatively, the recipient 110 and sender 120 may exchange encryption keys at different times, may use encryption techniques that do not require a key exchange, or may choose not to encrypt the data. In one embodiment, the DD domain 130 itself is used to perform the key exchange needed to facilitate an encrypted communications link.

[0027] The sender 120 uses the DDID and associated write token to send data to the recipient 110. In one embodiment, the sender 120 sends a write request to the DD domain 130 that includes the DDID and the write token. In another embodiment, the write request includes the DDID and data representing the write token. The data representing the write token may be encrypted (e.g., by hashing, obfuscation, etc.). This request is received by an initial DD node 140 in the DD domain 130. The receiving node 140 determines whether it has the DD identified by the DDID. If not, the receiving node 140 sends a message containing the DDID and the write token to the other nodes within DD domain 130. The node 140 storing the DD associated with the DDID receives the message and verifies the write token. If the token verifies, the node storing the DD creates a connection with the receiving node, which in turn has a connection with the sender 120. The sender 120 then writes the data to the node storing the DD associated with the DDID.

[0028] Similarly, the recipient 110 uses the DDID and associated read token to read data from the DD. In one embodiment, the recipient 110 sends a read request to the DD domain 130 that includes the DDID and the read token. In another embodiment, the read request includes the DDID and data representing the read token. The data representing the read token may be encrypted (e.g., by hashing, obfuscation, etc.). This request is received by an initial DD node 140 in the DD domain 130. The receiving node 140 determines whether it has the DD identified by the DDID. If not, the receiving node 140 broadcasts a message containing the DDID and the read token to the other nodes within the DD domain 130. The node 140 storing the DD associated with the DDID receives the message and verifies the read token. If the token verifies, the node 140 storing the DD creates a connection with the receiving node, which in turn has a connection with the recipient. The recipient 110 then reads the data from the node storing the DD associated with the DDID.

[0029] Thus, the DD network architecture described above permits secure and private communications between the recipient 110 and the sender 120. The sender 120, and/or other parties possessing the DDID and write token can send data to the recipient. However, such parties cannot read the data from the DD. Moreover, a malicious actor who obtains access to one or more nodes 140 in the DD domain 130 may be able to obtain or read data stored in individual DDs. But the malicious actor cannot determine the intended recipients of the data because there is no mapping of DDIDs to recipients. For the same reason, the malicious actor cannot determine the path between a sender and a recipient. In addition, the data stored in the DDs may be encrypted.

[0030] FIG. 2 is a high-level block diagram illustrating a detailed view of the DD domain 130 of FIG. 1 according to one embodiment. As described above, the domain 130 typically includes multiple DD nodes 140 connected by DD communication links 160. The individual nodes 140 within the DD domain 130 may be formed of physically separate computers, such as a collection of geographically disparate computers. In addition, some or all of the nodes may be formed of virtual computers. For example, the nodes 140 of a domain 130 may be instances of virtual computers hosted in a cloud environment.

[0031] Each DD node 140 has an associated set of characteristics that describe attributes of the node. The characteristics may describe the location of the node 140. The location may be specified as a geographic location. In addition, the location may be specified as a logical location (e.g., a “zone”). For example, the logical location may indicate that the node is associated with a particular enterprise (e.g., a business) or other group. The characteristics

may also describe physical properties of the node, such as a node's processing power, storage capacity, uptime, and the like.

[0032] In one embodiment, the set of DD nodes 140 in a DD domain 130 may be divided into multiple subdomains, with each subdomain including a proper subset of nodes from the set of DD nodes in the DD domain. The subdomains to which a node 140 is member may be determined based on the characteristics of the node. For example, the DD domain 130 may include nodes 140 distributed over a wide geographic area (e.g., a country), and a subdomain may include nodes physically located within a smaller area (e.g., a state within the country). Similarly, the DD domain 130 may include nodes 140 associated with multiple enterprises, and a subdomain may include only nodes associated with one of the enterprises or a part of an enterprise.

[0033] In one embodiment, the DD nodes 140 are arranged as a mesh network. Each node 140 is connected to at least one other node via a DD communication link 160. Moreover, each node 140 maintains a routing table identifying the nodes to which it is connected. A node 140 can send a message to another node by forwarding the message to the nodes to which it is connected. The nodes 140 that receive the message in turn forward the message to other nodes, until the message reaches the node to which it is directed. The path followed by the message is formed of hops from node 140 to node along the DD communication links 160.

[0034] Consider the communications between the sender 120 and the recipient 110 described in FIG. 1 in the context of FIG. 2. As shown in FIG. 2, the recipient 110 is connected to a node 140A of the domain 130 via a communication link 150A. This node 140A serves as the point of ingress to the domain 130 for the recipient. The recipient 110 sends a request to the ingress node 140A of the domain 130 to create a DD on behalf of the recipient. This request may include domain information specifying a particular subdomain in which the DD should be created. For example, the domain information may specify that the DD should be created in a node 140 located in a particular geographic area or managed by a particular enterprise.

[0035] The node 140A serving as the point of ingress for the recipient 110 receives the create request and analyzes the domain information to identify the subdomain in which the DD should be created. In one embodiment, the node 140A services the request by randomly selecting a node within the specified subdomain that will host the DD. In one embodiment, random selection is performed using a load balancing technique, which may be performed by

the node 140A or by a separate computing device. In one embodiment, the node 140 services the request by randomly selecting a number of node hops for which the request will be forwarded, and randomly selecting another node within the specified subdomain to which the node 140A is connected. The ingress node 140A then forwards the request to the randomly selected node (e.g., node 140D) and also forwards the selected value for the number of node hops. The node 140D to which the request was forwarded randomly selects another node (e.g., node 140E) in the subdomain from its routing table, decrements the value for the number of node hops, and forwards the request to the selected node. This selection, decrement, and forward process repeats until the value for the number of node hops reaches zero, at which point the final node establishes and hosts the DD associated with the request from the recipient 110. In one embodiment, each node that forwards the create request includes the path from the ingress node 150A to the forwarding node with the request. The final node that creates the DD uses the path to identify and contact the ingress node 140A for the recipient 110. For example, the node 140 may use the path to send the DDID and associated tokens to the ingress node 140A, so that the latter node can provide this information to the recipient.

[0036] For example, assume the ingress node 140A receives a create request from the recipient 110, and also assume that the request specifies a subdomain encompassing nodes 140A, 140D and 140E, as well as other nodes within the domain 130. Also assume the ingress node 140A randomly selects “2” as the number of hops. The ingress node randomly selects a node (e.g., node 140D) from the specified subdomain in its routing table, decrements the hop value, and forwards the request and decremented hop value (e.g., “1”) to the selected node. That node 140D, in turn, randomly selects another node (e.g., node 140E), decrements the hop value, and forwards the request and decremented hop value (e.g., “0”) to the selected node. The final node 140E evaluates the hop value and determines that it is “0” and, therefore, creates the DD and associated tokens. The final node 140E then returns the DDID and tokens to the ingress node 140A via the reverse of the path used to reach the final node.

[0037] Variations on the techniques described above may be used in some embodiments. In one embodiment, a node forwarding a request decrements the hop value only if the node is within the specified subdomain. This variation may be used, for example, in situations in which a node receiving a create request is not within the specified subdomain and/or connected to any other nodes in the subdomain. In this situation, the nodes may randomly forward the request to other nodes until a node within the specified subdomain receives the

request, at which point the node in the subdomain decrements the hop value and forwards the request anew if the hop value is greater than zero or creates the DD and associated tokens if the hop value is zero.

[0038] The sender 120, in turn, is connected to a different node 140L that serves as the point of ingress for the sender to the domain 130 via a different communication link 150B. When the sender 120 makes a write request, the sender provides the DDID and write token to the sender's ingress node 140L. This node 140L forwards the request including the DDID and write token to the other nodes in its routing table, and the other nodes continue to forward the request until it reaches the node having the DD associated with the DDID (e.g., node 140E). This node 140E verifies the token and establishes a connection with the sender's ingress node 140L using a return path created by the forwarding nodes. Alternatively, the node 140E may establish a direct connection with the sender 120. The sender 120 provides the data to be written to the ingress node 140L, and that node forwards the data to the node 140E having the DD via the connection. A read request made by the recipient 110 is handled in a similar fashion in one embodiment, except that the recipient reads data from, rather than writes data to, the node 140E having the DD.

[0039] FIG. 3 is a high-level block diagram illustrating an example of a DD node 140 according to one embodiment. The node 140 includes a routing module 305, creation module 310, write module 315, read module 320, time-to-live (TTL) module 325, data control module 330, delete module 335, notification module 340, geo-fence module 345, and data storage 390. Other embodiments may include different or other modules in other embodiments. In addition, the behaviors of the modules may differ in other embodiments.

[0040] The data storage 390 stores data used by the node 140. The data may include data being maintained in DDs managed by the node 140, DDIDs and tokens associated with the DDs, and information used by the modules within the node 140 to perform the tasks described herein. Depending upon the embodiment, the data storage 390 may include one or more types of non-transitory computer-readable persistent storage media. For example, the data storage 390 may include a hard drive, solid-state memory device, and/or other form of persistent memory.

[0041] Turning now to the individual modules within the node 140, the routing module 305 routes messages received by node 140. As part of this task, an embodiment of the routing module 305 maintains the routing table for the node 140. The routing module 305 periodically communicates with other nodes 140 to which it is connected to ascertain

information about those nodes. This information may include, for example, network addresses of the nodes, information about the subdomains with which the nodes 140 are associated, and the like. The routing module 305 stores this information about the connected nodes in the routing table. In addition, the routing module 305 responds to routing table-related communications from routing modules 305 of other nodes 140.

[0042] The messages handled by the routing module 305 include messages related to create requests, write requests, read requests, and other types of messages and requests described herein. For a given message, the routing module 305 analyzes the message to determine whether to process the message locally on the node 140 or to route the message to another node in the routing table. For example, upon receiving a create request from another node, the routing module 305 examines the hop value to determine whether it is greater than zero. If the hop value is greater than zero, the routing module 305 decrements the hop value, randomly selects a connected node in the routing table (subject to any specified subdomain constraints, if applicable), and forwards the request and decremented hop value to the selected node. If the hop value is zero, the routing module 305 provides the request to the creation module 310.

[0043] Similarly, upon receiving a write request, the routing module 305 determines whether the DDID is associated with a DD maintained by the node 140. If the DDID is not associated with a DD maintained by the node 140, the routing module 305 forwards the request to other nodes in its routing table. If, on the other hand, the DDID is associated with a DD maintained by the node, the routing module 305 provides the request to the write module 315. The routing module 305 handles read requests, as well as other requests described herein, in the same fashion.

[0044] The creation module 310 creates DDs in response to requests received by the node 140. Upon receiving a create request from the routing module 305, the creation module 310 generates a DDID to represent the DD for the request. In addition, the creation module 310 generates a set of tokens associated with the DDID. The creation module 310 provides the DDID and tokens to the recipient 110 that requested the DD using the path to the recipient's ingress node 140A as described with respect to FIGS. 1 and 2.

[0045] In one embodiment, the creation module 310 generates the DDID as a globally unique identifier (GUID). The DDID is a value represented using a large number of bits (e.g., a 128-bit value). A small portion of the bits may be fixed to identify the value as a DDID or encode other information, while the other bits are randomly generated by the

creation module 310. The large number of bits makes it extremely unlikely that the same DDID would be generated for multiple DDs. Therefore, each DDID is unique for practical purposes. The DDID may be represented as sequence of hexadecimal digits.

[0046] In one embodiment, the tokens generated by the creation module 310 may include a write token, a read token, and an owner token. The write and read tokens respectively provide the bearer of the token the rights to write data to, and read data from, the DD associated with the DDID as described above. The owner token provides the bearer with administrative rights to the DD, such as the right to delete data within the DD or delete the entire DD.

[0047] In one embodiment, a token, like the DDID, is a value represented using a large number of bits, some of which may be fixed and others of which are randomly generated by the creation module 310. The number of bits in each token may be fewer than the number of bits in the DDID. Each token associated with a particular DDID is unique with respect to other tokens associated with that DDID.

[0048] The creation module 310 allocates space in the data storage 390 for the created DD and associates this space with the DDID. The amount of space, and the time when the space is allocated may vary in different embodiments. In one embodiment the creation module 310 allocates a fixed amount of storage space for the DD when creating the DD. In another embodiment, the creation module 310 allocates the storage space later, when receiving data to store in the DD. Likewise, the amount of space allocated for the DD can vary in different embodiments.

[0049] The write module 315 writes data to DDs in response to write requests received by the node 140. Upon receiving a write request from the routing module 305, the write module 315 initially verifies the write token included in the request. The write module 315 identifies the DDID and write token included in the request, and compares the write token to the stored write token created by the creation module 310 for the DDID. If the compared write tokens do not match, the write module 315 denies the write request. Depending upon the embodiment, the write module 315 can deny the request by acting as if the request was not received (i.e., by not sending any messages in response to the write request) or by sending a message to the sender 120 indicating that the write token is invalid.

[0050] If the compared write tokens match, the write module 315 uses the return path in the write request to open a network connection with the ingress node 140L for the sender 120. The write module 315 receives data from the sender of the write request, and writes the

data to the storage allocated for the DD identified by the DDID. In one embodiment, the write module 315 stores the data in a DD as a series of discrete messages, such that the data for each write request to a DD is saved as a logically separate message. The stored messages for the DD are organized in a queue or another data structure.

[0051] The read module 320 reads data from DDs in response to read requests received by the node 140. Upon receiving a read request from the routing module 305, the read module 320 initially verifies the read token included in the request. The read module 320 identifies the DDID and read token included in the request, and compares the read token to the stored read token created by the creation module 310 for the DDID. If the compared read tokens do not match, the read module 320 denies the read request. Like the write module 315, the read module 320 can deny the request by acting as if the request was not received (i.e., by not sending any messages in response to the read request) or by sending a message to the recipient 110 indicating that the read token is invalid.

[0052] If the compared read tokens match, the read module 320 uses the return path in the read request to open a network connection with the ingress node 140A for the recipient 110. The read module 320 reads data from the storage allocated for the DD, and sends the data to the recipient 110 via the ingress node 140A. In another embodiment, a direct connection is established between the node storing the DD and the recipient 110 and the data is sent to the recipient without passing through the ingress node 140A. If the data in the storage is organized in a queue, an embodiment of the read module 320 sends the message in the queue in response to the request (e.g., in a first-in-first-out order) and removes the message from the queue after it is sent. Other embodiments may send multiple messages per read request and/or leave message in the queue after the messages are sent to the recipient 110. In another embodiment, the contents of a DD are not read, for example, because the holder of a read token corresponding to the DD no longer wishes to communicate with the sender of the message. In this case, communication may be disconnected unilaterally by the recipient, without action, consent, or knowledge by the sender.

[0053] The TTL module 325 maintains TTL information for the node 140. The TTL information generally describes for how long an entity persists in the domain 130. For example, DDs, DDIDs, tokens, and data written to DDs may have associated TTL information that describes how long the respective entities persist within the domain 130. Once the duration described by the TTL is reached, the entity to which the TTL pertains expires, is no longer recognized as valid, and may be deleted.

[0054] In addition, the TTL module 325 enforces the TTLs by detecting when a duration described by a TTL is reached, and invalidating the entity associated with the TTL. For example, if the TTL module 325 detects that a TTL for a DD is reached, the TTL module deletes the data stored within the DD and also deletes or otherwise invalidates the DDID and tokens associated with the DD so that the DD can no longer be accessed. Similarly, if the TTL module 325 detects that a TTL for data written to a DD is reached, the TTL module 325 deletes the data from the DD so the data can no longer be read.

[0055] The TTL information may be specified as a counter (e.g., a duration of time from the present time) or as a timestamp (e.g., an explicit time after which the entity is no longer valid). Additionally, the TTL may be specified as a number of instances of particular types of access (e.g., a DD expires once it is read from 3 times, or written to once). Further, the TTL information may be specified as a category (e.g., “default,” “short,” “medium,” “long,” or “confidential”). In the latter case, the TTL module 325 converts the category description to a counter or timestamp based on a TTL policy. Different entities may have different applicable TTL policies. For example, the TTL policies may specify that the “default” TTL for a DD is 30 days and the “default” TTL for a message is 7 days. The TTL module 325 may also support an “archive” TTL that does not expire, therefore making the entity having the TTL persistent.

[0056] In one embodiment, the recipient 110 specifies the TTL for a DD when creating it. For example, the TTL information may be embedded into the create request. Likewise, the sender 120 may specify the TTL for data by embedding the TTL in the write request. For example, the recipient 110 may specify a specific amount of time or number of instances of access for which the DD is valid, or specify a category as discussed above. The TTL specified for the DD is embedded into the create request and received by the TTL module 325.

[0057] The data control module 330 supports management of DDs for the nodes 140 of the domain 130. The data control module 330 provides a variety of management functions that can be accessed by the recipient 110 or other entity making a request for a particular function and providing tokens granting administrative authority, reading privilege, writing privilege, etc. for a given DD.

[0058] In one embodiment, the data control module 330 provides a movement function that moves a DD from one node 140 to another node while maintaining the same DDID. The recipient 110 may issue a move request that contains the DDID, the owner token and,

optionally, a specification of a node to which the DD should be moved. In various embodiments, movement of a DD may be initiated by a user request, environmental factors (e.g., a node 140 is scheduled to be taken offline, time of day), or a policy definition (e.g., a DD may only stay on a specific node 140 for a certain time before it is required to be moved). The node 140 to which the DD should be moved may be specified, for example, by indicating a subdomain to which the DD should be moved. In response to a move request having a valid owner token, the data control module 330 of the node 140 having the DD identified by the DDID identifies a new node to which the DD is to be moved. For example, the data control module 330 may randomly select a new node within the specified subdomain using the random selection technique described above and send that node a message identifying the DDID and the data for maintaining in the DD identified by the DDID. A data control module 330 in the new node 140 establishes a new DD under the existing DDID, and stores the received data in that DD. Once the new DD is established, the data control module 330 may optionally delete the DD identified by the DDID so that the DD has effectively been moved to the new node.

[0059] The data control module 330 also provides a replicate function that replicates a DD from a node 140 to one or more other nodes. The replication is similar to the movement function, except that the original data control module 330 does not delete the DD identified by the DDID after the new DD is created. In one embodiment, replication is initiated by a recipient 110. In another embodiment, replication is initiated automatically (e.g., by executable instructions stored in the DD that specify rules for replication). When a node 140 containing a replicated DD fulfills a write request, the routing module 305 forwards the write request to other nodes in the routing table so that each instance of the replicated DD may fulfill the write request and maintain data currency.

[0060] The data control module 330 further provides an archive function that stores an archive of a DD in another node 140. To perform the archive, the data control module 330 of the node 140 storing the DD receives an archive request similar to the move request. The data control module 330 communicates with the data control module 330 of the new node 140 to create a new DD associated with the same DDID. The data control module 330 sets the TTL for the entities associated with the new DD as “persistent,” meaning that the new DD acts as an archive for the DD in the original node. The data control module 130 of the original node 130 may optionally delete the DD identified by the DDID after the archive is created.

[0061] The delete module 335 deletes DDs from a node 140. The delete module 335 receives a delete request from the recipient 110 or another entity. The delete request contains a DDID and the associated owner token. The delete module 335 verifies that the received token grants delete privileges and, if it does, deletes the DD identified by the DDID from the node 140. In another embodiment, delete module 335 may delete one or more messages stored in a DD and not the entire DD itself. In one embodiment, the delete module 335 writes data to the storage location from which the DD is deleted. This writeover data may be randomly generated or predetermined. Writing writeover data makes recovering deleted data more difficult. It also makes finding DD data more difficult by increasing the total amount of stored data in the storage location, with the multiple instances of writeover data obfuscating DD data.

[0062] A notification module 340 provides notifications to other modules of the node 140, other nodes 140, or external destinations (e.g., push notifications, mobile device notifications, emails, phone calls, text messages, etc.) when an event takes place. An event is a change in state in a node 140. Examples of events include data being written to a DD (including data being written to a DD a specified number of times), data being read from a DD (including data being read from a DD a specified number of times), data being deleted from a DD, data control functions (moving data, replicating data, archiving data, etc.), TTL functions (TTL expiration, TTL changes, TTL creation, TTL deletion, etc.), actions performed by a node, and time instances (a clock reading a particular time, a timer expiring, etc.). In one embodiment, the notification module 340 of a node 140 receives a notification request from a module of the node 140, a different node 140, a recipient 110, or another entity. The notification request specifies the event for which the notification is to be created. For events relating to DDs, the notification request includes the DDID and a token (e.g., owner token, read token) associated with the DDID. The notification request further includes a notification address to which a notification is to be made when the event occurs. In one embodiment, the notification address is specified in the form of a DDID and write token for a different DD in the domain 130. The notification address may be specified as a different form of address, such as an address on the Internet to which an email or another form of message may be sent.

[0063] The notification module 340 determines whether the token for a DDID in a notification request is correct. If the token is not correct, the notification module 340 rejects or otherwise ignores the notification request. If the token is correct, the notification module

340 examines the notification request to identify the type of event for which the notification is requested. The notification module 340 then creates a notification for the event. In one embodiment, the notification module 340 establishes a trigger that detects when the appropriate type of event occurs for the identified DD. To this end, the notification module 340 may monitor the activities of the other modules (e.g., the write module 315) to determine when such events occur. For example, if the notification request specifies a write event, the notification module 340 may monitor the write module 315 to detect writes to the indicated DD.

[0064] When the requested event is detected, the notification module 340 generates and sends a notification to the specified notification address. The notification may identify the DD to which the event occurs (e.g., by including the DDID in the notification) and the type of event that occurred (e.g., a write to the DD having the DDID). If the notification address is for a DD, the notification module 340 acts as a sender 120 and uses the write token and DDID specified in the notification request to write the notification to the DD. In this example, the recipient 110 or other entity that requested the notification can monitor a single DD to receive notifications about events occurring in multiple different DDs. If the notification address is specified as a different form of address, the notification module 340 sends the notification using the appropriate technique for the address.

[0065] A geo-fence module 345 receives and analyzes geographic-related restrictions associated with the DD or requests received by the DD. The geo-fence module 345 communicates with the other modules in the DD to enforce such restrictions. The restrictions may specify that a DD is only accessible by senders and recipients within a geographic area specified by the creator of the DD. Access may be restricted in various ways in different embodiments. For example, in one embodiment, requests received by a DD may be valid only if the originator of the request is located within a certain geographic area. In another embodiment, a DD or specific contents of the DD may be accessible only if a specified party (e.g., owner, recipient, sender, third party, etc.) is within a certain geographic area. The geo-fence module 345 may also communicate with the notification module 340 to send notifications when events (e.g., write requests, read requests, etc.) occur within specified geographic areas.

[0066] A monitoring module 350 detects events and sends action messages to an action module 355 upon detection. The monitoring module 350 can monitor events within the node 140 or outside of the node 140, such as at a different node 140. To monitor events occurring

outside of the node 140, the monitoring module 350 may send a notification request to one or more notification modules 340 of other nodes 140. The notification request may identify the event for which notification is requested and include a notification address identifying the node making the request. The notification request is received by a notification module 340 of the other node, and a notification is sent to, and received by, the monitoring module 350 in the first node when the event occurs. To monitor events within the node 140, the monitoring module 350 may send a notification request to the notification module 340 of the node 140, or the monitoring module may communicate directly with other modules of the node 140 to determine when the event occurs.

[0067] Trigger messages specify events to be monitored and actions to be performed if the events are detected. Trigger messages may be sent by users, system administrators, software developers, other modules of the node 140, other nodes 140, applications, etc. Trigger messages are received by the monitoring module 350, and the monitoring module monitors the specified events. In one embodiment, trigger message contents are stored in the data storage 390. In another embodiment, trigger message contents are stored in a DD accessible to the monitoring module 350 and the action module 355.

[0068] The action module 355 performs actions when events are detected. An action is a function performed in response to a change in state in a node 140. Examples of actions include notifications, notification requests, command sequences, writing to a DD, reading from a DD, deleting data from a DD, deleting a DD, data control functions (moving data, replicating data, archiving data, etc.), TTL functions (TTL expiration, TTL changes, TTL creation, TTL deletion, etc.), and sending messages (e.g., to address on the Internet to which an email or another form of message may be sent). Command sequences may include instructions sent to systems or devices connected to the node 140. The action module 355 is informed by the monitoring module 350 when an event is detected, and performs the actions specified in a trigger message.

[0069] In one embodiment, events and actions are connected to perform multiple actions in sequence upon detection of an event. A first action performed in response to a first event may also be specified as a second event with a corresponding second action. For example, a first trigger message specifies a first event as a read from a first DD and a first action as a write operation a second DD. Accordingly, when the first DD is read from, the second DD is written to. A second trigger message specifies a second event as a write operation to the second DD and a second action as an email to an email address.

[0070] A scheduling module 360 determines when an action is performed relative to an event. In one embodiment, a trigger message includes a specified time at which an action occurs. The specified time may be expressed as a counter (e.g., a duration of time after the event is detected) or as a timestamp (e.g., an explicit time at which the action occurs). In another embodiment, an event is a time instance, such as a timer expiring or a clock reaching a certain time. The scheduling module 360 communicates with the action module 355 to perform the action at the specified time. For example, upon detection of an event, the scheduling module 360 may instruct the action module 355 to wait to perform the action until the specified time or until further instructions are received from the scheduling module.

[0071] A handshake module 365 facilitates direct connections between two or more nodes 140 that may otherwise be unable to connect due to network security policies, such as firewall policies. In one embodiment, the handshake module 365 performs dual functions. First, a handshake module located in a handshake node 140 provides functionality allowing establishment of a direct connection between two other nodes using respective handshake modules in the other two nodes. Second, the handshake module 365 in a node 140 establishes a direct connection with the handshake module of another node by exchanging data using the handshake module of a handshake node. In other embodiments, the handshake module 365 may perform only one of these functions and/or perform additional functions.

[0072] An embodiment of the handshake module 365 performs the first function – allowing two other nodes to establish a direct connection – by exchanging handshake information between the two nodes. Handshake data may include network address information such as IP addresses and port addresses and/or information describing characteristics of the nodes. In one embodiment, the handshake module 365 within the handshake node allows for the creation of a DD for the purpose of exchanging handshake information. The node on whose behalf the DD is created receives the associated DDID and may provide the DDID to the other node with which the direct connection is to be established. The two nodes that will be party to the direct connection may then exchange handshake information using the handshake node.

[0073] An embodiment of the handshake module 365 performs the second function – establishing a direct connection with the handshake module of another node – identifying a handshake node that can be used to exchange information with the other node. In one embodiment, the handshake module 365 maintains a directory of handshake nodes. The handshake module 365 selects a particular handshake node from the directory and exchanges

the identity of the selected handshake node with the handshake module of the other node via an indirect connection. Once the handshake modules 365 of the two nodes know the identity of the handshake node, the handshake modules 365 respectively connect to the handshake node to exchange information and establish a direct connection.

[0074] FIG. 4 is a flowchart illustrating steps for using a DD to pass data from a sender 120 to a recipient 110 according to one embodiment. FIG. 4 describes the steps from the perspective of a node 140 of a domain 130. Other embodiments may include different and/or other steps than those described herein, and the steps may be performed in different orders. Likewise, some or all of the steps may be performed by entities other than the node 140 in other embodiments.

[0075] The node 140 receives 410 a create request. As described above, a recipient 110 can issue the create request and send the request to an ingress node 140A in the domain 130. The ingress node 140A randomly selects a node to service the request. Assume, then, that the node 140 receives 410 the create request after having been randomly selected to service it. In response to the create request, the node 140 generates 415 a DDID and a set of associated tokens for the new DD. In addition, the node 140 may allocate storage space for storing data written to the DD. The node 140 provides the tokens and DDID to the recipient 110.

[0076] Subsequently, the node 140 receives 420 a write request including the DDID and associated write token. The write request may have been issued by a sender 120 who received the DDID and write token from the recipient 110. The sender 120 sends the write request to an ingress node 140L in the domain 130 which, in turn, forwards the write request to other nodes in the domain until it reaches the node that created the DD associated with the DDID. The node 140 determines whether the write token is valid. If the token is valid, the node 140 responds to the write request by establishing a connection with the sender's ingress node 140L. The node 140 receives the data to be written to the DD from the sender 120 via the ingress node 140L and stores 425 the data in the DD. If the token is not valid, an embodiment of the node 140 does not respond to the write request. In another embodiment, if the token is not valid, the node 140 sends a message indicating the request is rejected.

[0077] The node 140 later receives 430 a read request including the DDID and associated read token. The read request may have been issued by the recipient 110 who established the DD identified by the DDID. Similar to a write request, the recipient 110 sends the read request to an ingress node 140A in the domain 130 which forwards the read request to other nodes in the domain until it reaches the node that created the DD associated with the DDID.

Upon receiving the read request, the node 140 determines whether the read token is valid. If the token is valid, the node 140 responds to the read request by establishing a connection with the recipient's ingress node 140A and sends it the data from the DD. For example, if the DD is maintained as a queue, the node 140 will send the data that is next in the queue. If the token is not valid, an embodiment of the node 140 does not respond to the read request. In another embodiment, if the token is not valid, the node 140 sends a message indicating the request is rejected.

[0078] FIG. 5 is a flowchart illustrating steps for using a DD to monitor events and perform actions if an event is detected according to one embodiment. A trigger message is received 505 by the monitoring module 350. The trigger message specifies an event to be monitored, and one or more actions to be performed if the event is detected. The trigger message may be stored in data storage 390 or a DD. Responsive to the trigger message being received, the monitoring module 350 begins 510 monitoring for the occurrence of the event. For example, the monitoring module 350 may send one or more notification requests to monitor the event, or a timer or clock may be set by the scheduling module 360. When the event occurs, the event is detected 515 by the monitoring module 350. The monitoring module 350 informs 520 the action module 355 that the event has occurred. The action module 355 determines the set of actions associated with the event by the trigger message and performs 525 the actions. In one embodiment, when the action module performs the set of actions, the monitoring module 350 deletes the trigger record such that the action only occurs once in response to the event detection. In another embodiment, the trigger record persists, allowing the set of actions to be performed each time the event is detected.

[0079] FIG. 6 is a high-level block diagram illustrating an example of two nodes establishing a direct connection according to one embodiment. Specifically, FIG. 6 illustrates a first node 140X establishing a direct connection with a second node 140Y. FIG. 6 also illustrates a handshake node 140Z. In FIG. 6, nodes 140X and 140Y are protected by respective network security systems 610X and 610Y. The network security system 610 may be, e.g., a firewall. The network security system 610 may filter, block, or otherwise prevent certain inbound network communications from reaching a node 140. The two nodes 140X, 140Y establishing the direct connection can overcome the respective network security systems 610X, 610Y by exchanging handshake information using the handshake node 140Z.

[0080] In one embodiment, the node 140X establishing the direct connection accesses the handshake directory which is maintained, or otherwise accessible via, the node's handshake

module 365. The node 140X selects the handshake node 140Z, and sends an identifier of the selected handshake node to the other node 140Y that will be party to the direct connection. The node 140X may send the identifier of the handshake node 140Z to the other node using an indirect connection over the DD domain 130, using standard networking communications, and/or via other techniques. In one embodiment, the node 140Y that receives a direct connection request with an identifier of a handshake node 140Z will verify that the handshake node is also included in its handshake directory. Once a handshake node 140Z is selected and known to both nodes 140X and 140Y, nodes 140X and 140Y may form a direct connection by exchanging handshake information via the handshake node 140Z.

[0081] FIG. 7 is a high-level block diagram illustrating physical components of a computer 700 used as part or all of one or more of the entities described herein in one embodiment. For example, instances of the illustrated computer 700 may be used as the recipient 110, sender 120, and/or a node 140 in the domain 130. Illustrated are at least one processor 702 coupled to a chipset 704. Also coupled to the chipset 704 are a memory 706, a storage device 708, a keyboard 710, a graphics adapter 712, a pointing device 714, and a network adapter 716. A display 718 is coupled to the graphics adapter 712. In one embodiment, the functionality of the chipset 704 is provided by a memory controller hub 720 and an I/O controller hub 722. In another embodiment, the memory 706 is coupled directly to the processor 702 instead of the chipset 704. In one embodiment, one or more sound devices (e.g., a loudspeaker, audio driver, etc.) is coupled to chipset 704.

[0082] The storage device 708 is any non-transitory computer-readable storage medium, such as a hard drive, compact disk read-only memory (CD-ROM), DVD, or a solid-state memory device. The memory 706 holds instructions and data used by the processor 702. The pointing device 714 may be a mouse, track ball, or other type of pointing device, and is used in combination with the keyboard 710 to input data into the computer 700. The graphics adapter 712 displays images and other information on the display 718. The network adapter 716 couples the computer system 700 to a local or wide area network.

[0083] As is known in the art, a computer 700 can have different and/or other components than those shown in FIG. 7. In addition, the computer 700 can lack certain illustrated components. In one embodiment, a computer 700 acting as a node 140 may lack a keyboard 710, pointing device 714, graphics adapter 712, and/or display 718. Moreover, the storage device 708 can be local and/or remote from the computer 700 (such as embodied within a storage area network (SAN)).

[0084] As is known in the art, the computer 700 is adapted to execute computer program modules for providing functionality described herein. As used herein, the term “module” refers to computer program logic utilized to provide the specified functionality. Thus, a module can be implemented in hardware, firmware, and/or software. In one embodiment, program modules are stored on the storage device 708, loaded into the memory 706, and executed by the processor 702.

[0085] The above description is included to illustrate the operation of certain embodiments and is not meant to limit the scope of the invention. From the above discussion, many variations will be apparent to one skilled in the relevant art that would yet be encompassed by the spirit and scope of the invention.

CLAIMS

What is claimed is:

1. A method for performing an action responsive to detecting an event in a dead drop (DD) domain, the method comprising:

monitoring, by a node in the DD domain, for an occurrence of a specified event within the DD domain;

detecting the occurrence of the specified event within the DD domain; and

performing a specified action in response to detecting the occurrence of the specified event within the DD domain.

2. The method of claim 1, further comprising:

receiving a notification request specifying the event to be monitored, the

notification request including a dead drop identifier (DDID) referencing a storage location in the DD domain associated with the specified event and a token associated with the DDID, and specifying the action to perform in response to detecting the occurrence of the specified event; and

determining whether the token associated with the DDID is correct;

wherein the monitoring is performed responsive to determining that the token associated with the DDID is correct.

3. The method of claim 1, further comprising:

receiving a message specifying the event to be monitored and the action to

perform in response to detecting the occurrence of the specified event, wherein the specified event is to occur within a second node of the DD domain; and

sending a notification request to the second node of the DD domain, the

notification request identifying the specified event and including a notification address to which a notification is to be made when the event occurs, the notification address referencing the node in the DD domain;

wherein detecting the occurrence of the specified event within the DD domain comprises receiving a notification from the second node indicating that the specified event occurred at the second node.

4. The method of claim 1, further comprising:

determining a notification address to which a notification is to be made when the specified event occurs, wherein the notification address includes a dead drop identifier (DDID) referencing a storage location in the DD domain and a token associated with the DDID;

wherein performing the specified action in response to detecting the occurrence of the specified event within the DD domain comprises using the token to write a notification message to the storage location referenced by the DDID.

5. The method of claim 1, wherein detecting the occurrence of the specified event within the DD domain comprises:

detecting a change in a state of a node within the DD domain.

6. The method of claim 1, wherein performing a specified action in response to detecting the occurrence of the specified event comprises:

performing a sequence of actions responsive to detecting the occurrence of the specified event.

7. The method of claim 1, further comprising:

receiving a notification request specifying the event to be monitored and a duration of time after the event is detected;

wherein performing the specified action in response to detecting the occurrence of the specified event within the DD domain comprises performing the specified action at the duration of time after the event is detected.

8. A system for exchanging data between a sender and a recipient comprising:
a processor for executing computer program instructions;

a non-transitory computer-readable storage medium storing computer program instructions executable by the processor to perform steps comprising:

- monitoring, by a node in the DD domain, for an occurrence of a specified event within the DD domain;
- detecting the occurrence of the specified event within the DD domain;
- and
- performing a specified action in response to detecting the occurrence of the specified event within the DD domain.

9. The system of claim 8, the steps further comprising:

- receiving a notification request specifying the event to be monitored, the notification request including a dead drop identifier (DDID) referencing a storage location in the DD domain associated with the specified event and a token associated with the DDID, and specifying the action to perform in response to detecting the occurrence of the specified event; and
- determining whether the token associated with the DDID is correct;

wherein the monitoring is performed responsive to determining that the token associated with the DDID is correct.

10. The system of claim 8, the steps further comprising:

- receiving a message specifying the event to be monitored and the action to perform in response to detecting the occurrence of the specified event, wherein the specified event is to occur within a second node of the DD domain; and
- sending a notification request to the second node of the DD domain, the notification request identifying the specified event and including a notification address to which a notification is to be made when the event occurs, the notification address referencing the node in the DD domain;

wherein detecting the occurrence of the specified event within the DD domain comprises receiving a notification from the second node indicating that the specified event occurred at the second node.

11. The system of claim 8, the steps further comprising:

determining a notification address to which a notification is to be made when the specified event occurs, wherein the notification address includes a dead drop identifier (DDID) referencing a storage location in the DD domain and a token associated with the DDID;

wherein performing the specified action in response to detecting the occurrence of the specified event within the DD domain comprises using the token to write a notification message to the storage location referenced by the DDID.

12. The system of claim 8, wherein detecting the occurrence of the specified event within the DD domain comprises:

detecting a change in a state of a node within the DD domain.

13. The system of claim 8, the steps further comprising:

receiving a notification request specifying the event to be monitored and a duration of time after the event is detected;

wherein performing the specified action in response to detecting the occurrence of the specified event within the DD domain comprises performing the specified action at the duration of time after the event is detected.

14. The system of claim 8, the steps further comprising:

receiving a notification request specifying the event to be monitored and a duration of time after the event is detected;

wherein performing the specified action in response to detecting the occurrence of the specified event within the DD domain comprises performing the specified action at the duration of time after the event is detected.

15 . A non-transitory computer-readable storage medium storing computer program instructions executable by a processor to perform steps comprising:

monitoring, by a node in the DD domain, for an occurrence of a specified event within the DD domain;

detecting the occurrence of the specified event within the DD domain; and

performing a specified action in response to detecting the occurrence of the specified event within the DD domain.

16. The non-transitory computer-readable storage medium of claim 15, the steps further comprising:

receiving a notification request specifying the event to be monitored, the notification request including a dead drop identifier (DDID) referencing a storage location in the DD domain associated with the specified event and a token associated with the DDID, and specifying the action to perform in response to detecting the occurrence of the specified event; and
determining whether the token associated with the DDID is correct;
wherein the monitoring is performed responsive to determining that the token associated with the DDID is correct.

17. The non-transitory computer-readable storage medium of claim 15, the steps further comprising:

receiving a message specifying the event to be monitored and the action to perform in response to detecting the occurrence of the specified event, wherein the specified event is to occur within a second node of the DD domain; and
sending a notification request to the second node of the DD domain, the notification request identifying the specified event and including a notification address to which a notification is to be made when the event occurs, the notification address referencing the node in the DD domain;
wherein detecting the occurrence of the specified event within the DD domain comprises receiving a notification from the second node indicating that the specified event occurred at the second node.

18. The non-transitory computer-readable storage medium of claim 15, the steps further comprising:

determining a notification address to which a notification is to be made when the specified event occurs, wherein the notification address includes a dead

drop identifier (DDID) referencing a storage location in the DD domain and a token associated with the DDID;

wherein performing the specified action in response to detecting the occurrence of the specified event within the DD domain comprises using the token to write a notification message to the storage location referenced by the DDID.

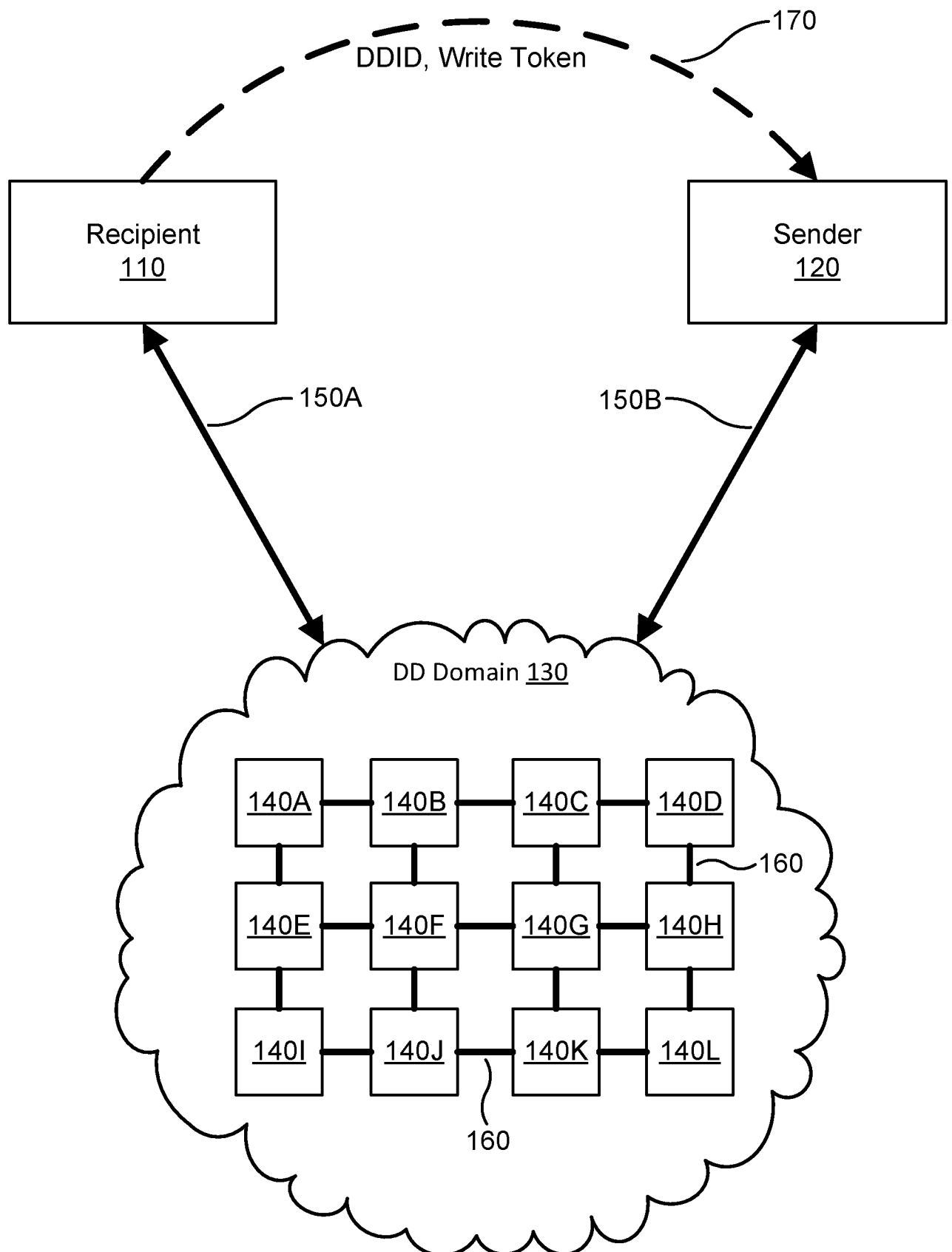
19. The non-transitory computer-readable storage medium of claim 15, wherein detecting the occurrence of the specified event within the DD domain comprises:

detecting a change in a state of a node within the DD domain.

20. The non-transitory computer-readable storage medium of claim 15, the steps further comprising:

receiving a notification request specifying the event to be monitored and a duration of time after the event is detected;

wherein performing the specified action in response to detecting the occurrence of the specified event within the DD domain comprises performing the specified action at the duration of time after the event is detected.

**FIG. 1**

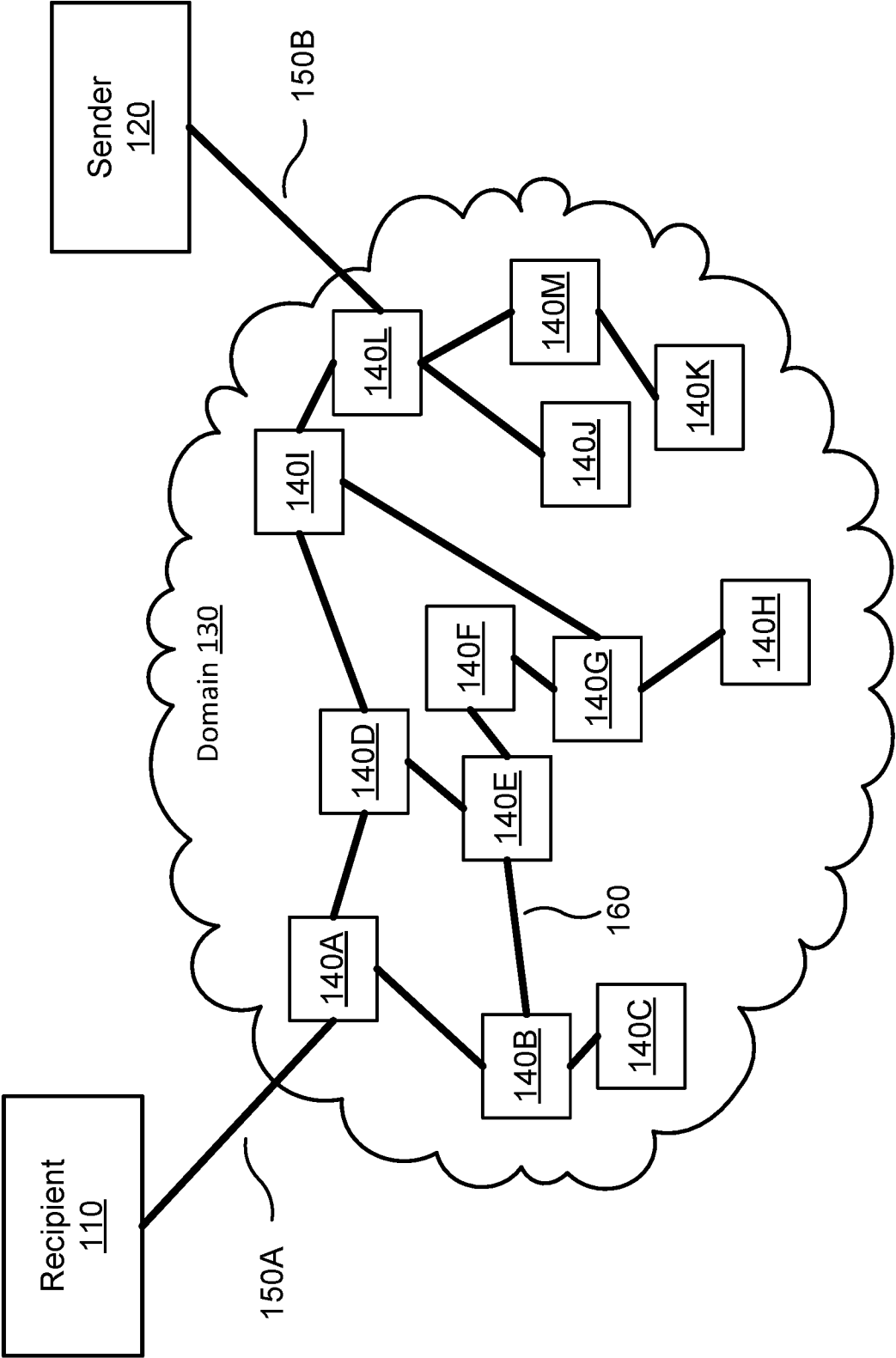


FIG. 2

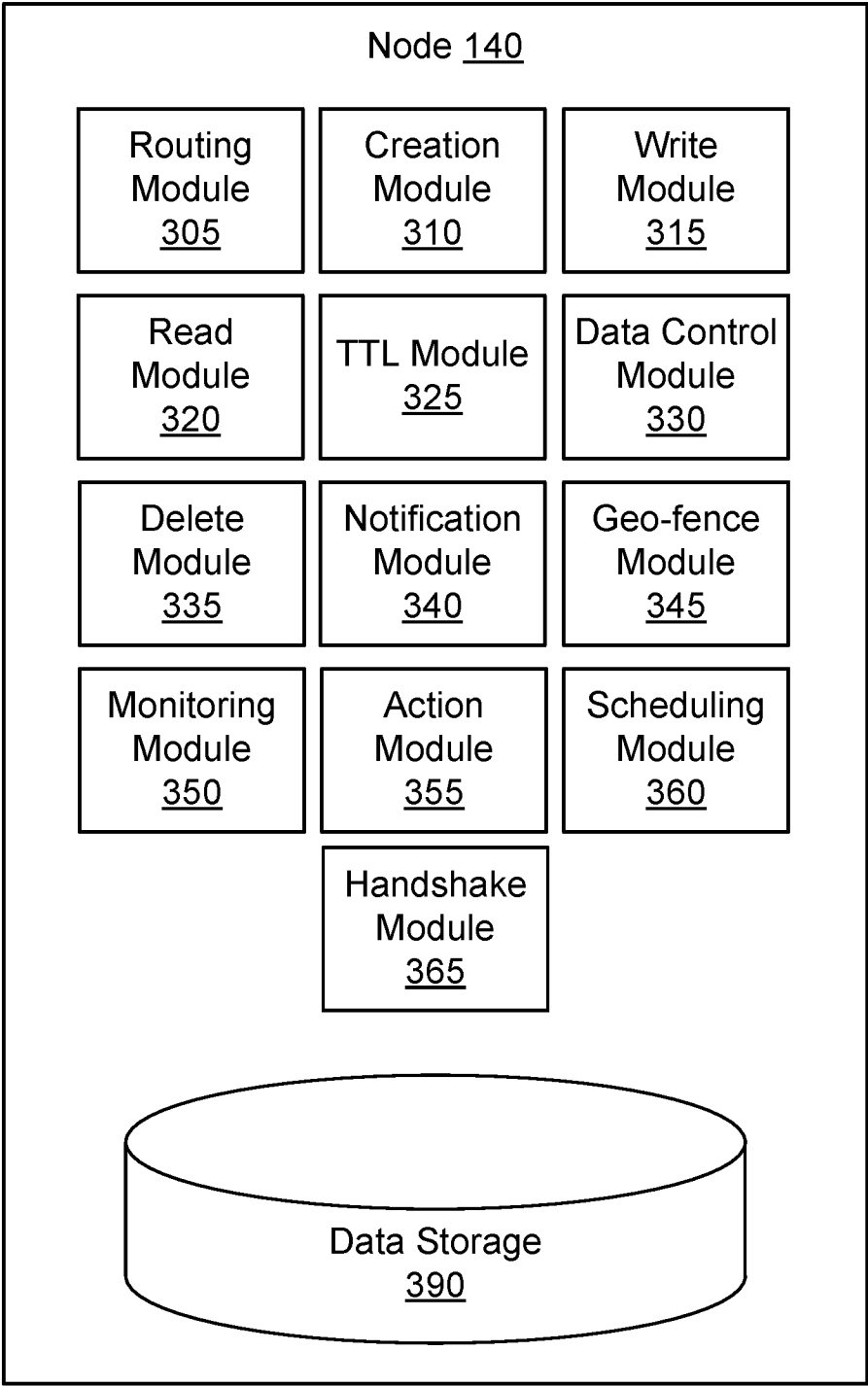
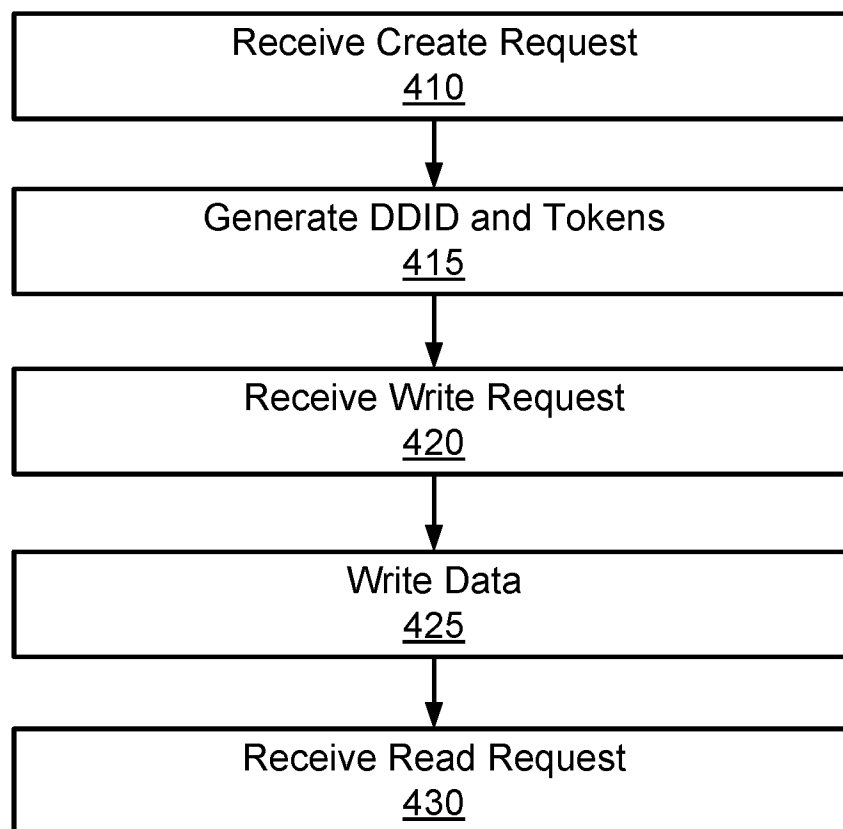
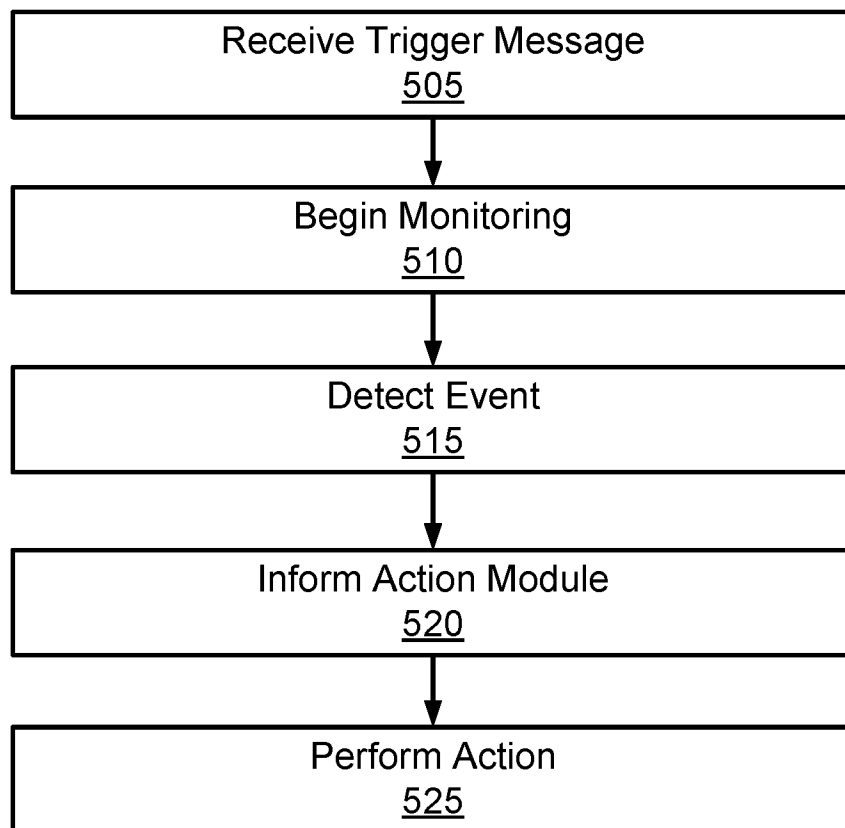


FIG. 3

**FIG. 4**

**FIG. 5**

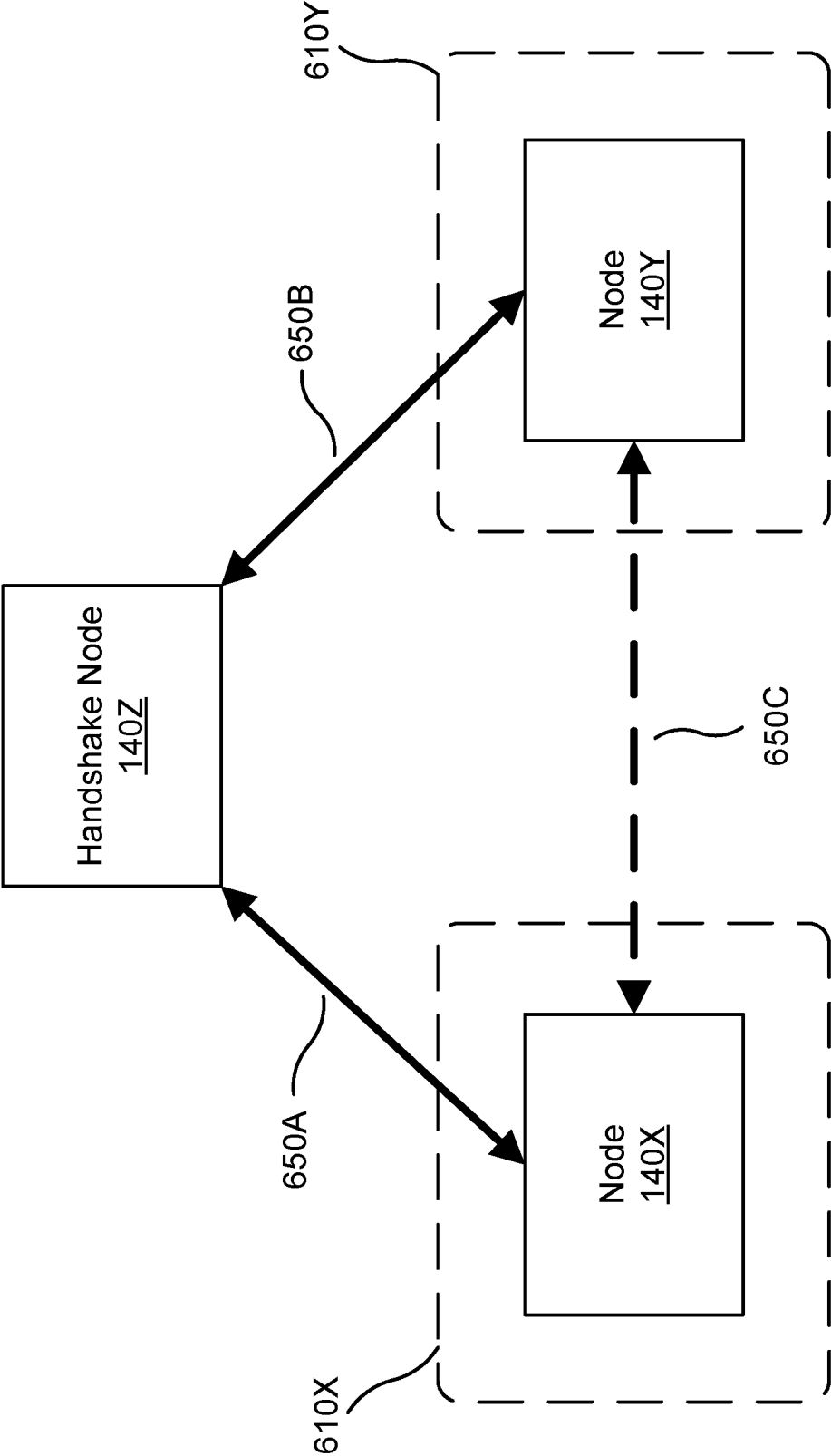
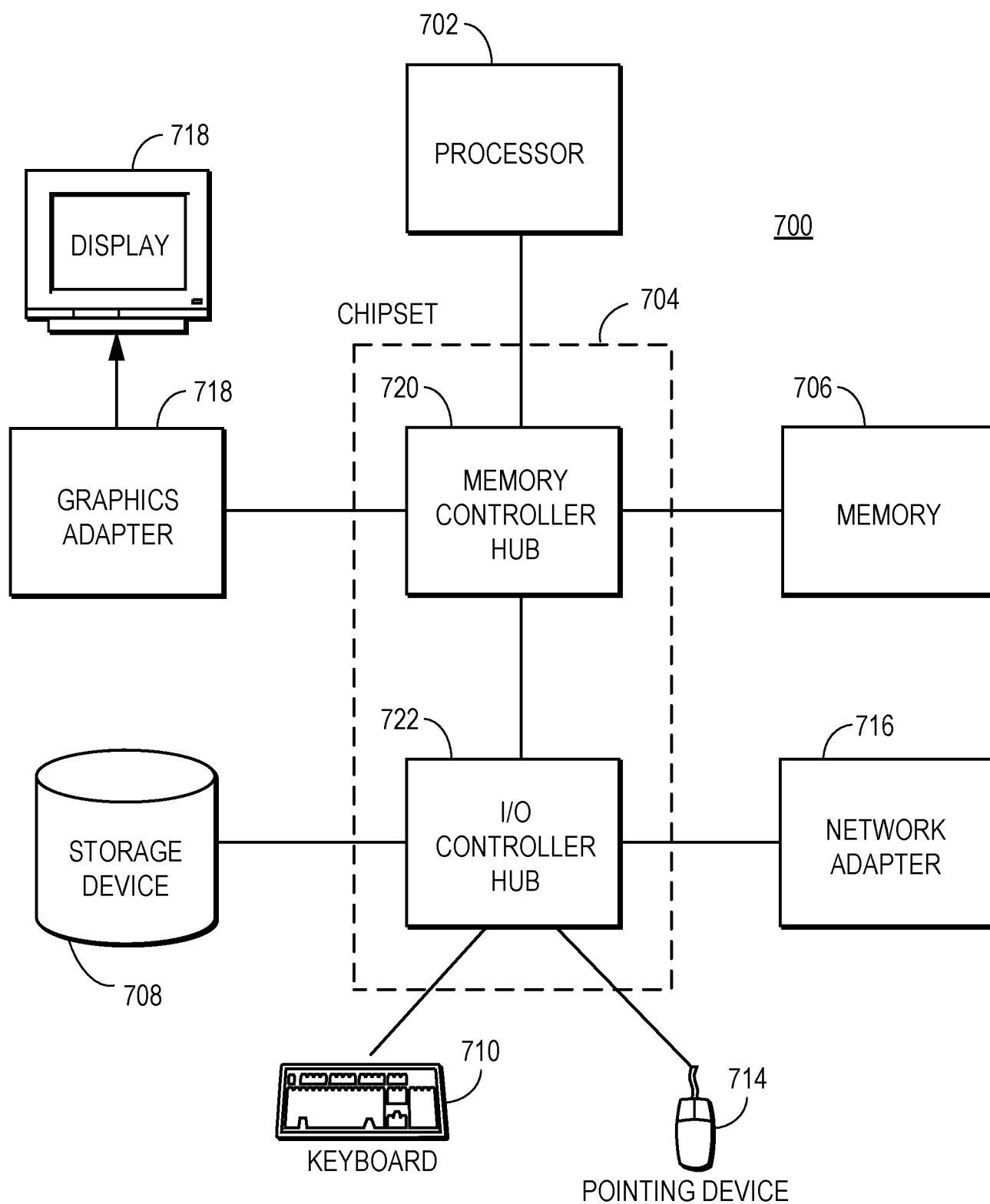


FIG. 6

7/7

**FIG. 7**

INTERNATIONAL SEARCH REPORT

International application No.

PCT/US2016/029005

A. CLASSIFICATION OF SUBJECT MATTER

IPC(8) - G06F 11/30; G06F 9/44; G06F 9/46; G06F 21/00; G06F 15/16 (2016.01)

CPC - G06F 9/542; G06F 11/3006; G06F 11/3476; G06F 21/552 (2016.05)

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

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

IPC(8) - G06F 11/30; G06F 9/44; G06F 9/46; G06F 21/00; G06F 15/16 (2016.01)

CPC - G06F 9/542; G06F 11/3006; G06F 11/3476; G06F 21/552 (2016.05)

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

US: 709/203; 709/224; 719/318 (keyword delimited)

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

PatBase, Google Patents, ProQuest

Search terms used: dead drop, events, detect, monitor, action, identifier, token, notification

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US 2008/0307525 A1 (NICKLE) 11 December 2008 (11.12.2008) entire document	1-20
Y	US 2009/0254392 A1 (ZANDER) 08 October 2009 (08.10.2009) entire document	1-20
Y	US 6,654,782 B1 (O'BRIEN et al) 25 November 2003 (25.11.2003) entire document	2-4, 9-11, 16-18
Y	US 2011/0321148 A1 (GLUCK) 29 December 2011 (29.12.2011) entire document	2, 4, 9, 11, 16, 18
A	US 2003/0133417 A1 (BADT) 17 July 2003 (17.07.2003) entire document	1-20
A	US 2007/0256081 A1 (COMER) 01 November 2007 (01.11.2007) entire document	1-20
A	US 6,088,804 A (HILL et al) 11 July 2000 (11.07.2000) entire document	1-20
A	US 8,082,224 B2 (MARINESCU) 20 December 2011 (20.12.2011) entire document	1-20
A	US 6,366,926 B1 (POHLMANN et al) 02 April 2002 (02.04.2002) entire document	1-20
A	US 5,237,684 A (RECORD et al) 17 August 1993 (17.08.1993) entire document	1-20
A	US 8,631,283 B1 (SIVASUBRAMANIAN et al) 14 January 2014 (14.01.2014) entire document	1-20

☐ Further documents are listed in the continuation of Box C.

☐ See patent family annex.

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search

21 June 2016

Date of mailing of the international search report

26 JUL 2016

Name and mailing address of the ISA/

Mail Stop PCT, Attn: ISA/US, Commissioner for Patents

P.O. Box 1450, Alexandria, VA 22313-1450

Facsimile No. 571-273-8300

Authorized officer

Blaine R. Copenheaver

PCT Helpdesk: 571-272-4300

PCT OSP: 571-272-7774