

(19) **DANMARK**

(10) **DK/EP 3691191 T3**



(12)

Oversættelse af europæisk patentskrift

Patent- og
Varemærkestyrelsen

-
- (51) Int.Cl.: **H 04 L 12/26 (2006.01)**
- (45) Oversættelsen bekendtgjort den: **2021-11-22**
- (80) Dato for Den Europæiske Patentmyndigheds bekendtgørelse om meddelelse af patentet: **2021-08-25**
- (86) Europæisk ansøgning nr.: **20154784.1**
- (86) Europæisk indleveringsdag: **2018-09-25**
- (87) Den europæiske ansøgnings publiceringsdag: **2020-08-05**
- (30) Prioritet: **2017-11-08 US 201715807045**
- (62) Stamansøgningsnr: **18793074.8**
- (84) Designerede stater: **AL AT BE BG CH CY CZ DE DK EE ES FI FR GB GR HR HU IE IS IT LI LT LU LV MC MK MT NL NO PL PT RO RS SE SI SK SM TR**
- (73) Patenthaver: **Google LLC, 1600 Amphitheatre Parkway, Mountain View, CA 94043, USA**
- (72) Opfinder: **LEBSACK, Carl, Google LLC, 1600 Amphitheatre Parkway, Mountain View, CA California 94043, USA**
SHAKIR, Robert, Google LLC, 1600 Amphitheatre Parkway, Mountain View, CA California 94043, USA
BORMAN, Paul, Google LLC, 1600 Amphitheatre Parkway, Mountain View, CA California 94043, USA
HINES, Marcus, Google LLC, 1600 Amphitheatre Parkway, Mountain View, CA California 94043, USA
SHAIKH, Anees, Google LLC, 1600 Amphitheatre Parkway, Mountain View, CA California 94043, USA
GEORGE, Joshua, Google LLC, 1600 Amphitheatre Parkway, Mountain View, CA California 94043, USA
- (74) Fuldmægtig i Danmark: **AWA Denmark A/S, Strandgade 56, 1401 København K, Danmark**
- (54) Benævnelse: **Streamingnetværksovervågningsmellemlagringsinfrastruktur.**
- (56) Fremdragne publikationer:
US-A1- 2008 195 369
US-A1- 2013 191 436
US-A1- 2016 321 352

DESCRIPTION

RELATED APPLICATIONS

[0001] The present application claims priority to, and the benefit of, U.S. Patent Application No. 15/807,045, titled "STREAMING NETWORK MONITORING CACHING INFRASTRUCTURE" and filed on November 8, 2017.

BACKGROUND

[0002] Efficient and effective control of modern computer networks can benefit from accurate and timely information regarding network conditions. Generally speaking, information regarding network conditions, including the status of individual network devices can be referred to as "network telemetry" data. Network telemetry data can originate at the network devices (e.g., routers and switches) and be made available to network telemetry gathering tools as well as to individual clients. In some cases, frequent polling of a network device by multiple clients may contribute to overloading the device or otherwise impacting its performance for critical functions. Furthermore, the various network devices have been manufactured by different vendors, and may even represent a mix of current and legacy devices, or different device models, each using a different schema for communicating network telemetry data.

US2016321652 (A1) is directed to providing dynamic indexer discovery. An index manager, which may also be known as a cluster master, is configured to track the statuses and capabilities of indexers and provide the statuses and capabilities obtained from the indexers to data collectors, such as forwarders. The data collectors may use the statuses and capabilities associated with the indexers to load balance transmission of data to the indexers. Dynamic indexer discovery may eliminate the need to manually reconfigure data collectors when the status of an indexer changes because the information may be obtained from the index manager without the need to reinitialize the data collectors.

SUMMARY

[0003] At least one aspect is directed to a method of translating telemetry data in a network telemetry caching and distribution system, as defined according to the appended independent claim 1.

[0004] At least one other aspect is directed to a network telemetry caching and distribution system, as defined by the appended independent claim 10.

[0005] These and other aspects and implementations are discussed in detail below. The foregoing information and the following detailed description include illustrative examples of

various aspects and implementations, and provide an overview or framework for understanding the nature and character of the claimed aspects and implementations. The drawings provide illustration and a further understanding of the various aspects and implementations, and are incorporated in and constitute a part of this specification.

BRIEF DESCRIPTION OF THE DRAWINGS

[0006] The accompanying drawings are not intended to be drawn to scale. Like reference numbers and designations in the various drawings indicate like elements. For purposes of clarity, not every component may be labeled in every drawing. In the drawings:

Figure 1 illustrates an environment including a network telemetry caching and distribution system, according to an illustrative implementation;

Figure 2 shows an example data node, according to an illustrative implementation;

Figure 3 shows an example tree data structure, according to an illustrative implementation;

Figure 4 shows an example tree data structure employing read locks and write locks as a data node is updated, according to an illustrative implementation;

Figure 5 shows an example tree data structure employing read locks and write locks as a new data node is added, according to an illustrative implementation;

Figures 6A-6F show an example tree data structure during execution of an example subpath query, according to an illustrative implementation;

Figure 7 shows an example client queue including a node pointer map and a node pointer queue, according to an illustrative implementation;

Figure 8 shows a flowchart of an example network telemetry caching and distribution method, according to an illustrative implementation;

Figure 9 shows an example environment including a network telemetry collection and translation system, according to an illustrative implementation;

Figure 10 illustrates different types of matches in an example translation tree, according to an illustrative implementation;

Figure 11 illustrates an example of condensing a translation tree by regular expression match, according to an illustrative implementation;

Figure 12 illustrates using different matches based on whether a given path is correctly split, according to an illustrative implementation;

Figure 13 illustrates an example translator internal architecture, according to an illustrative implementation; and

Figure 14 is a block diagram illustrating a general architecture for a computer system that may be employed to implement elements of the systems and methods described and illustrated herein, according to an illustrative implementation.

DETAILED DESCRIPTION

[0007] This disclosure generally relates to a system and method of caching and distributing network telemetry data. The system can receive key-value-timestamp tuples of network telemetry data, store them in a tree data structure for ease of search and retrieval, and distribute them to clients via a client queue that maintains up-to-date network telemetry data even if the amount of data produced exceeds the ability of the client to process it.

[0008] The system can serve as a central hub for network telemetry updates from multiple network devices to multiple clients. In doing so, the system can reduce the query load on individual network devices and, in extreme cases, prevent denial of service attacks against the network devices.

[0009] The system can also serve as a queue for providing network telemetry updates to clients. A system receiving a query for a data set to the system has an obligation to provide the full set of data resulting from the request; however, certain network or client operating conditions may limit the client's ability to handle all of the data. Accordingly, the system can implement a client queue to ensure that the client receives up-to-date telemetry data even if it is not able to handle the full volume of the requested data, and to ensure that the data set does not disproportionately include data from frequently updated nodes.

[0010] The client queue can include a map of node pointers and counters, and a node pointer queue. The client queue ensures that data transmitted to the client is kept up to date while including an indication of any data node values dropped or skipped. An example operation of the client queue can occur as follows. At initiation, or following a complete transmission of network telemetry data to a client, the client queue can be cleared; that is, both the node pointer map and the node pointer queue will be empty. When the system receives a network telemetry data update from a network device, it can add or update a data node in the tree data structure, and add a corresponding node pointer to the data node in the client queue. If the node pointer map does not already include the pointer, the system can add the pointer to both the node pointer map, with a count equal to zero, and to the node pointer queue. If the map already includes the node pointer, the system can increment the node count (e.g., from zero to one), but not add the node pointer to the node pointer queue. In some implementations, the node pointer queue can be a first-in-first-out (FIFO) queue. For example, the node pointer queue can receive node pointers at its input, and return them from its output in the same order in which they were received. In some implementations, the network telemetry data can be transmitted to the client according to the output of the node pointer queue. Because the node

pointer queue can operate in a first-in, first-out manner, transmission of network telemetry data to the client device can begin with the first data node value updated subsequent to the previous transmission of network telemetry data to the client device. In some implementations, the node pointer queue may include only a single copy of each pointer regardless of how many times the corresponding data node has been updated since initialization or the previous transmission to the client device; thus, the next transmission to the client device will not include a disproportionate amount of data from frequently updated data nodes. Rather, the data value from each data node will be sent once, along with its corresponding count. The count can indicate to the client the number of times the value of the corresponding data node has changed since the previously transmitted network telemetry data update. In this manner, the system can enable a client to determine its inability to handle the throughput of the network telemetry data updates, while continually providing updates for every unique node and preventing the client from becoming progressively further out of sync.

[0011] The system can provide network telemetry data to clients in various manners, illustrated by the following examples. A client device can submit a query for a set of data once and receive the resulting data set in response. The client device can submit a query for a set of data once and receive resulting data set repeatedly by submitting subsequent requests. The client can submit a query for a set of data and receive a complete set of resulting data once followed by continual updates as they are received from the network devices. The system can implement other manners of providing network telemetry data without departing from the scope of this disclosure.

[0012] The system can respond to queries for subpaths or globs of the tree data structure. For example, a query may contain only a partial path with a wildcard such as a "*". The partial path may indicate an index node. Responding to the query can include returning the respective values from all data nodes directly under the indicated index node, and all data nodes indirectly under the indicated index node via any daughter index nodes of the indicated index node.

[0013] The system can provide central caching for multiple network devices. To do so, the system can maintain a device-index map. Each entry in the device-index map can include a network device name and a pointer to an indexed cache corresponding to the network device. Each indexed cache can be an instance of the tree data structure described above specific to the network device. Isolating the caches for each network device avoids lock contention among computing devices within respective caches. And, in implementations where the system is distributed among multiple computing devices, the isolated caches allow for easy migration of tree data structures between the multiple computing devices, because adding or deleting an indexed cache only introduces minimal contention at the device-index map, and remains contention free at the indexed cache level (i.e., within the tree data structure).

[0014] The system can receive network telemetry data from multiple network devices in the form of key-value-timestamp tuples, and store the data in data nodes (leaves) of a tree data structure. Data collection can be performed, for example, according to the systems and methods described in co-pending U.S. Patent Application Serial Number 15/343,876, filed

November 4, 2016, and entitled "Network Management Interface," the contents of which is incorporated herein. Each data node can include a full path (key), a data value, and a timestamp indicating when the data was produced by the network device. The tree data structure can also include index nodes forming an index for both updates and queries, including subpath queries that can return multiple values. The tree can include a top-level node and one or more tiers of index nodes. The first tier can include the top-level node, which can point to one or more index nodes in a second tier. Second-tier index nodes can each point to one or more index nodes in a third tier, and so on. An index node of the second or higher tiers can point to a data node.

[0015] The system can provide a mechanism of read and write locks to handle contention for index nodes and data nodes. Upon initiation, all nodes are inserted from scratch and the tree index must be built. From the top-level node, write locks are used when index nodes are added. For example, when a third-tier index node is added under a second-tier index node, the second-tier index node is write-locked, preventing any reading from, or writing to, or otherwise modifying of the second-tier index node. Similarly, when a data node and/or an index node is deleted, the index node above is write locked until the deletion is complete. When a data node is updated, it is write locked to prevent any reading from, or writing to, the data node. The parent index nodes above the data node being updated, however, need only be read locked, which prevents modification of those index nodes, but not their use for other read operations.

[0016] In some implementations, the system can translate network telemetry data from a vendor-specific format to a vendor-neutral format. In some cases, the network devices may not communicate network telemetry data using a consistent or standard format. For example, in some implementations, the network devices may have come from different vendors, may represent different models or model lines from a vendor, or may span both current and legacy devices. Therefore, in some situations, the vendor-specific network telemetry data produced by the various network devices may benefit from translation into a vendor-neutral format prior to being stored by the caching and distribution system. Such a system may benefit from a collection and translation system to process the various network telemetry data prior to use by the caching and distribution system.

[0017] Figure 1 illustrates an environment 100 including a network telemetry caching and distribution system 110, according to an illustrative implementation. The environment 100 also includes one or more network devices 120a, 120b, and 120c (collectively "network devices 120") and one or more client devices 130a and 130b (collectively "clients 130"). The caching and distribution system 110, the network devices 120, and the client devices 130 can communicate with each other over a computer network 105. The computer network 105 can represent a secure, local-area network (LAN) such as a data center network, or an unsecure, wide-area network (WAN) such as the Internet. The caching and distribution system 110 includes a central processing unit (CPU) 117 and a memory 118. The memory 118 includes various indices and data structures such as a device-index 111; one or more tree data structures 112a, 112b, and 112c (collectively "tree data structures 112"); a client-index 115; and one or more client queues 116a and 116b (collectively "client queues 116").

[0018] Each network device 120 can be a switch, router, modem, gateway, software-defined-network application, or other component of computer network infrastructure. In some implementations, each network device 120 can be a computing device such as the computing system 1510 described below with reference to Figure 14. Each network device 120 can receive and forward data packets and other network traffic to and from other network devices 120, hosts, servers, and/or client devices, including, for example, the client devices 130. In some implementations, the network devices 120 can operate in one or more software-defined networks (SDNs). In some implementations, the network devices can receive and forward data packets and other network traffic in a data plane, while exchanging messages in a control plane. The control plane messages can include control information and commands from SDN controllers. The control plane messages can also include status information regarding the network device 120. The status information can include network telemetry data. Generally speaking, network telemetry can refer to collecting information regarding network conditions and network device status as monitored or measured at the network device 120. The network device 120 can provide the network telemetry data to the caching and distribution system 110 and, in some implementations, directly to the client devices 130.

[0019] Each client device 130 can be a computer device involved in the control and/or monitoring of a computer network. In some implementations, each network device 120 can be a computing device such as the computing system 1510 described below with reference to Figure 14. Each client device 130 can receive network telemetry data either directly from the network devices 120, or via a cached copy from the caching and distribution system 110. When querying for network telemetry data, the client device 130 has an obligation to receive all data responsive to the query. In some situations, however, a mismatch may arise between the volume of network telemetry data provided, and the client device's 130 ability to process it. For example, network conditions may limit the amount of data the network can convey in a given time period. In some situations, the type of data storage medium used by the client device 130 may limit the speed with which the client device 130 can store the data. To address a situation in which the caching and distribution system 110 is producing network telemetry data faster than the client device 130 can receive and process it, the caching and distribution system 110 can implement a client queue for that client. The client queue can buffer, and optionally coalesce, network telemetry updates destined for the client device. The client queue is discussed in more detail below with regard to the caching and distribution system 110, and also with reference to the client queue 700 described further below with reference to Figure 7.

[0020] The caching and distribution system 110 can be a centralized or distributed system for receiving network telemetry data from one or more network devices 120, and providing the network telemetry data to one or more clients device 130. In some implementations, the caching and distribution system 110 can be a computing device such as the computing system 1510 described below with reference to Figure 14. The caching and distribution system 110 can provide several benefits in the collection and distribution of network telemetry data. For example, the caching and distribution system 110 can reduce the number of entities querying a network device 120 for network telemetry data. This can reduce the load on the network device

120 and prevent denial-of-service events caused by the network device 120 receiving more queries than it can process. Another benefit is that the caching and distribution system 110 can collect the network telemetry data and store it in a client queue until a client device 130 is able to receive and process the data.

[0021] The caching and distribution system 110 includes the CPU 117 and the memory 118. The CPU 117 can include a computer logic device or system capable of executing instructions in the form of computer code such as a kernel, an operating system, one or more computer applications, or other types of executable computer code. The CPU 117 can include one or more microprocessor, application-specific integrated circuits (ASICs), field-programmable gate arrays (FPGAs), graphics processing unit (GPU), and/or other fixed or programmable logic devices. In some implementations, the CPU 117 can include the processor 1550 or computing system 1510 described below with reference to Figure 14.

[0022] The memory 118 can include one or more memory devices including one or a combination of volatile and/or nonvolatile memory. The memory 118 can include random access memory (RAM), FLASH memory, magnetic memory such as a hard drive or tape drive, or optical memory such as optical disks. In some implementations, the memory 118 can include the cache 1575 and/or the memory 1570 described below with reference to Figure 14. In some implementations, the memory 118 can be divided up into a number of memory spaces or regions. The various memory spaces can exist within the same physical memory device, or distributed across multiple physical memory devices. Each memory space can store one or more of computer program instructions, the device-index 111, one or more tree data structures 112, the client-index 115, and/or one or more client queues 116.

[0023] In some implementation, each tree data structure 112 can correspond to one of the network devices 120 in the environment 100. For example, the tree data structure 112a can store and index network telemetry data from the network device 120a, the tree data structure 112b can store and index network telemetry data from the network device 120b, and so on. The tree data structures 112 are described in further detail below with reference to Figures 2-6. The device-index 111 can be a map that stores, for each network device 120, a name of the network device 120 and an indexed cache pointer pointing to a tree data structure 112 corresponding to the network device 120. In some implementations, the device-index 111 allows for modularity and/or distribution of the tree data structures 112. For example, in some implementations, the caching and distribution system 110 may include multiple memories 118, with each memory storing a subset of the tree data structures 112. In some implementations, the environment 100 may include multiple caching and distribution systems 110, each responsible for receiving network telemetry data from a different subset of the network devices 120, and for storing a subset of the tree data structures 112. In systems that are distributed in this manner, the device-index 111 provides a map to the various tree data structures 112. The device-index 111 can be readily updated by the caching and distribution system 110 if a particular tree data structure 112 is moved to another memory 118 or another caching and distribution system 110.

[0024] In some implementations, each client queue 116 can correspond to a particular client device 130 in the environment 100. For example, the client queue 116a can buffer data intended for the client device 130a, the client queue 116b can buffer data intended for the client device 130b, and so on. The client-index 115 can be a map that stores, for each client device 130, a name of the client device 130 and an indexed cache pointer pointing to a client queue 116 corresponding to the client device 130. Similar to the device-index 111, the client-index 115 facilitates modularity of the system by providing a map linking individual client queues 116 to their corresponding client device 130, regardless of where the particular client queue 116 is stored.

[0025] The caching and distribution system 110 can receive network telemetry data from the network device 120 and provide caching and distribution services to the client devices 130. The following is a description of an example operation of the caching and distribution system 110. The caching and distribution system 110 can receive network telemetry data from a network device 120. The caching and distribution system 110 can store the received network telemetry data in the memory 118. The caching and distribution system 110 can store the received network telemetry data as a plurality of data nodes and/or updates to existing data nodes. In some implementations, the caching and distribution system 110 can store the data nodes in a tree data structure 112. The caching and distribution system 110 can also maintain a client queue 116 in the memory 118. The client queue 116 can include a node pointer queue and a node pointer map. The node pointer map can include a plurality of node-pointer, node-count pairs. Each node-pointer can indicate a path to a data node corresponding to the node-pointer. Each node-count indicates the number of updates to the data node elapsing since a previous transmission of the contents of the data node to the client device 130. Operation of the client queue 116 is described in further detail below with reference to Figure 7. The caching and distribution system 110 can transmit, to the client device 130, network telemetry data from the client queue 116. The data can include node-value, node-count pairs including, for each node pointer in the node pointer queue, a data value in the data node corresponding to the node-pointer, and the node-count corresponding to the node-pointer in the node pointer map.

[0026] These and other operations of the caching and distribution system 110 are described below with reference to Figures 2-13.

[0027] Figure 2 shows an example data node 210, according to an illustrative implementation. The data node includes raw data 220. The raw data 220 can be thought of as a group of key-value-timestamp tuple. Each key-value-timestamp tuple is stored in a wrapper structure that can enable synchronization for read-write access. Each key can be a unique list of strings making it suitable for insertion into a tree data structure where leaves contain the values and timestamps. An example tree data structure is described below with reference to Figure 3.

[0028] The raw data 220 can include a full path of the data node 210 in a tree data structure. The full path can be a string data type. The raw data 220 can include a data value. The data value represents a quantitative measurement of some aspect of network telemetry. The data

value can be an interface data type. In some implementations, the raw data 220 can include a timestamp. The timestamp can represent the amount of time elapsed since a previous synchronizing event. For example, the timestamp can be an 64-bit integer value data type that can record, for example, how many nanoseconds since the Unix epoch had elapsed when the data value was recorded. The data node 210 can include more or fewer pieces of raw data 220 without departing from the scope of this disclosure.

[0029] Figure 3 shows an example tree data structure 300, according to an illustrative implementation. The tree data structure 300 includes a collection of data nodes 301a-301g (collectively "data nodes 301"). The data nodes 301 can be similar to the example data node 210 previously described. The right side of the tree data structure 300 shows a tree index. The tree index can be used for both updates and for queries.

[0030] The tree index includes an arrangement of index nodes branching out from a top-level node (TL) and pointing to the collection of data nodes 301. The index nodes can make up a tiered data structure with the TL node in a first tier, index nodes a and b in a second tier, index nodes c through f in a third tier, index nodes g through m in a fourth tier, and the data nodes 301 making up the "leaves" of the data structure. In the tree data structure 300, a higher-tier node is said to be the parent of a lower-tier node to which it points, and the lower-tier node to which the parent node points is said to be the daughter. In the tree data structure 300, the TL node is the parent of each of the second-tier nodes a and b. Node a is the parent of the third-tier nodes c and d. Node b is the parent of the third-tier nodes e and f. Node c is the parent of fourth-tier nodes g and h. Node d is the parent of fourth-tier node i. Node e is the parent of fourth-tier nodes j and k. Node f is the parent of fourth-tier nodes l and m. Node g is the parent of data node 301a, node h of data node 301b, node i of data node 301c, node j of data node 301d, node k of data node 301e, node l of the data node 301f, and index node m of the data node 301g.

[0031] The network telemetry caching and distribution system 110 can execute queries by traversing the tree data structure 300 according to a search path. The network telemetry caching and distribution system 110 can use a read lock function to avoid changes in the tree data structure 300 disrupting the execution of the query. The read lock function can prevent alterations of any index nodes or data nodes currently involved in executing a query.

[0032] The network telemetry caching and distribution system 110 can also add, update, and delete data nodes and index nodes from the tree data structure 300. The network telemetry caching and distribution system 110 can use a combination of read locks and write locks to prevent the changes in the tree data structure 300 from disrupting the execution of the query. The read lock can prevent the alteration of nodes, as previously described, and the write lock can prevent a write locked node from being read as well as being altered.

[0033] Operation of read locks and write locks are described further below with reference to Figures 4 through 6.

[0034] Figure 4 shows an example tree data structure 400 employing read locks and write locks as data node 301b is updated, according to an illustrative implementation. The tree data structure 400 includes the same TL node, index nodes, and data nodes 301 as the tree data structure 300 described above with reference to Figure 3. In the tree data structure 400, the data node 301b is being updated based on a network telemetry update received by the caching and distribution system. While the data node 301b is being updated, the caching and distribution system can impose a write lock to prevent any reading from, or modifications to, the data node 301b. The write lock can prevent both disruption of the update as well as the integrity of any read operations attempted on the data node 301b.

[0035] In addition to the write lock applied to the data node 301b, the caching and distribution system can impose a read lock on any parent index nodes of the data node 301b. In particular, the caching and distribution system can impose a read lock on the TL node as well as index nodes a, c, and h. The read lock can prevent any modifications to the TL node and the index nodes a, c, and h. The read lock can prevent disruption of the update to the data node 301b, while allowing other read operations involving the TL node and the index nodes a, c, and h.

[0036] Figure 5 shows an example tree data structure 500 employing read locks and write locks as a new data node 301n is added, according to an illustrative implementation. The tree data structure 500 includes the same TL node, index nodes, and data nodes 301 as the tree data structure 300 described above with reference to Figure 3. In the tree data structure 500, the new data node 301n is being added based on network telemetry data received by the caching and distribution system. The addition also includes a new index node n that is the parent of the new data node 301n. While the new data node 301n is being added, the caching and distribution system can impose a write lock to prevent any reading from, or modifications to, the index node d that is the parent of the new index node n. The write lock can prevent both disruption of the addition as well as the integrity of any read operations that involve index node d, which may, due to the addition, undergo a change of state and thus may be unstable in its response to a read operation.

[0037] In addition to the write lock applied to index node d, the caching and distribution system can impose a read lock on any parent index nodes of index node d. In particular, the caching and distribution system can impose a read lock on the TL node as well as index node a. The read lock can prevent any modifications to the TL node and the index node a. The read lock can prevent disruption of the addition of the new data node 301n and new index node n, while allowing other read operations involving the TL node and the index node a.

[0038] The network telemetry caching and distribution system 110 can execute queries by traversing a tree data structure according to a search path provided in the query. In some implementations, the search path can be a complete path that defines a single data node. In some implementations, the search path can be a subpath that may encompass multiple data nodes. Figures 6A-6F shows an example tree data structure 600 during execution of an example subpath query, according to an illustrative implementation. The tree data structure 600 includes the same TL node, index nodes, and data nodes 301 as the tree data structure

300 described above with reference to Figure 3. Each of Figures 6A-6F illustrates a stage of a query for the subpath `"/*/c"`. This subpath, or "glob," query is intended to retrieve data from each data node 301 that has a path including `/c`. In other words, the query is intended to retrieve data from each data node 301 that is, either directly or indirectly, a daughter of index node c. The caching and distribution system can process the query and return to the querying entity the data from each data node 301 in the subpath.

[0039] In Figure 6A, execution of the query begins at the TL node. The caching and distribution system can impose a read lock on the TL node. The read lock can prevent any modification to the read-locked node while allowing other read operations involving the read-locked node. In Figure 6B, execution of the query continues to index node a. The caching and distribution system can impose a read lock on index node a. In Figure 6C, execution of the query continues to index node c. The caching and distribution system can impose a read lock on index node c. Because the query specifies the sub path `"/c"`, which includes index node c, the caching and distribution system can proceed to retrieve data from each data node 301 that is, either directly or indirectly, a daughter of index node c.

[0040] In Figure 6D, execution of the query continues to index node g and to the data node 301a. The caching and distribution system can impose a read lock on index node g and the data node 301a. The caching and distribution system can retrieve a data value from the data node 301a. In Figure 6E, execution of the query continues to index node h and to the data node 301b. The caching and distribution system can impose a read lock on index node h and the data node 301b. The caching and distribution system can retrieve a data value from the data node 301b.

[0041] Having retrieved data values from all of the data nodes 301 under index node c and index node a, the query can continue scanning the data tree 600 for other data nodes 301 satisfying the query. In Figure 6F, execution of the query continues to index node b. The caching and distribution system can impose a read lock on index node b. The caching and distribution system can determine that no other data nodes 301 under index node b satisfy the query. The caching and distribution system can thus complete execution of the query.

[0042] Figure 7 shows an example client queue 700 including a node pointer map and a node pointer queue, according to an illustrative implementation. The example client queue 700 can represent network telemetry data queued for a client device, such as the client device 130 described previously. In some implementations, the caching and distribution system may maintain a client queue for each client device serviced by the caching and distribution system. The node pointer map includes a data structure holding node-pointer, node-count pairs. For example, a first node-pointer, node-count pair includes a node pointer to a data node "C" and a node count of zero. The node count equal to zero indicates that the data in data node C has not been updated since the last transmission of node-pointer, node-count pairs to the client device. Another node-pointer, node-count pair includes a node pointer to a data node "B" and a node count of 5. The node count equal to 5 indicates the data value in data node B has been updated five times since the last transmission of node-value, node-count pairs to the client

device. This relatively high count may indicate that the data in data node B is frequently updated by receipt of network telemetry data updates from a network device, such as the network device 120 described previously.

[0043] The node pointer queue can include zero, one, or many node pointers. In the example client queue 700, the node pointer queue includes eight node pointers: I, H, G, F, E, D, C, and B. The node pointer queue can include a node pointer corresponding to each data node that has been updated since a previous transmission of node-value, node-count pairs to the client device. In some implementations, the node pointer queue only includes a single copy of any particular node pointer, regardless of how many times the corresponding data node has been updated since a previous transmission of data to client. In some implementations, the node pointer queue is a first-in-first-out (FIFO) queue. For example, in the example client queue 700, the node pointer to data node B was the first node pointer added to the node pointer queue, and will be the first node pointer transmitted to the client device during the next transmission. In this manner, the oldest data node will have its data transmitted first. Meanwhile, the node pointer to data node I was the last node pointer added to the node pointer queue, and may be the last node pointer transmitted to the client device during the next transmission, if no other node pointers are added to the node pointer queue. When the caching and distribution system is ready to transmit node-value, node-count pairs to the client device, it can retrieve data values corresponding to each node pointer in the node pointer queue, and combine them with the corresponding node count in the node pointer map.

[0044] The example client queue 700 illustrates three operations: an item of network telemetry data received in the first instance, an update to an item of network telemetry data, and a transmission of a node-value, node-count pair from the example client queue 700 to the client device.

[0045] The example client queue 700 illustrates an item of network telemetry data received in the first instance. The example client queue 700 is receiving a node pointer to a new data node N. The caching and distribution system can check the node pointer map to see if it already includes an entry for data node N. In this case, the example client queue 700 does not include an entry for data node N, thus such a new entry is added with a node count set to zero. The node pointer to the data node N is also added to the node pointer queue.

[0046] The example client queue 700 illustrates an update to an item of network telemetry data. The example client queue 700 is receiving a node pointer to a data node H that has been updated. The caching and distribution system can check the node pointer map to see if it already includes an entry for data node H. In this case, the example client queue 700 does include an entry for data node H, thus no new entry is added; rather, a node count corresponding to the node pointer to data node H is incremented to 1. The node count of 1 indicates that the data node H has been added to the example client queue 700 and updated once since the last transmission to the client. This means that the client will not receive the previous data value stored in the data node H prior to the most recent update; the client will, however, be informed that it has missed the previous data value. Because the node pointer to

the data node H already had an entry in the node pointer map, no corresponding node pointer is added to the node pointer queue. By not adding node pointers that already exist in the node pointer map to the node pointer queue, the client queue 700 avoids transmitting to the client device a disproportionate number of updates corresponding to frequently updated data nodes.

[0047] The example client queue 700 illustrates a transmission of a node-value, node-count pair from the example client queue 700 to the client device. When the caching and distribution system is ready to transmit node-value, node-count pairs to the client, it can retrieve data values corresponding to each node pointer in the node pointer queue, coupled with the corresponding node count in the node pointer map. For example, in the example client queue 700, the caching and distribution system is preparing to send node-value, node-count pairs to the client. The first node-value, node-count pair can correspond to the data node A, which has a node count of 3. The node count of 3 indicates that the data node A has been added to the example client queue 700 and updated three times since the last transmission to the client. This means that the client device will not receive the previous three data values stored in the data node A prior to the most recent update; the node count will inform the client device, however, that it has missed the previous data values. When a node-value, node-count pair is transmitted to the client, the node pointer can be removed from the node pointer queue.

[0048] An example method of operation of the a caching and distribution system employing a client queue is described below with reference to Figure 8.

[0049] Figure 8 shows a flowchart of an example network telemetry caching and distribution method 800, according to an illustrative implementation. The method 800 includes receiving a set of network telemetry data (stage 810). The method 800 can be executed by a caching and distribution system such as the caching and distribution system 110 previously described with reference to Figure 1. The method 800 includes storing the received network telemetry data as a plurality of data nodes (stage 820). The method 800 includes maintaining a client queue having a node pointer queue and a node pointer map (stage 830). The method 800 includes receiving a data node update (stage 840). The method 800 includes determining whether the pointer map already includes a node pointer corresponding to the updated data node (decision block 850). If the node pointer map does not include a node pointer corresponding to the updated data node, the method 800 includes adding a node pointer corresponding to the updated data node to the node pointer map with a count of zero, and adding the node pointer to the node pointer queue (stage 860). If the node pointer map does include a node pointer corresponding to the updated data node, the method 800 includes incrementing the node count corresponding to the updated data node, and not adding the node pointer to the node pointer queue (stage 870). The method 800 includes transmitting node-value, node-count pairs to the client device (stage 880).

[0050] The method 800 includes receiving a set of network telemetry data (stage 810). The caching and distribution system can receive the set of network telemetry data from a network device, such as the network device 120 previously described.

[0051] The method 800 includes storing the received network telemetry data as a plurality of data nodes (stage 820). The caching and distribution system can store the received network telemetry data in a physical memory, such as the memory 118. The caching and distribution system can store the received network telemetry data as a plurality of data nodes. In some implementations, the caching and distribution system can store the received network telemetry data in a tree data structure having a top-level node pointing, via a plurality of index nodes, to a plurality of data nodes. Each data node can include a path, a data value, and a timestamp. Each data node can be similar to the data node 210 or the data node 301 described previously. The tree data structure can be similar to the tree data structures 300, 400, 500, and 600 described previously.

[0052] The method 800 includes maintaining a client queue having a node pointer queue and a node pointer map (stage 830). The caching and distribution system can maintain the client queue in a second memory space of the physical memory. The client queue can be, for example, the example client queue 700 described previously. The client queue can include a node pointer queue and a node pointer map. The node pointer map can include a plurality of node-pointer, node-count pairs. Each node-pointer can indicate a path to a data node corresponding to the node-pointer. The node-count can indicate the number of updates to the data node that have elapsed since a previous transmission of the contents of the data node to a client device. The client device can be a client device 130 previously described with reference to Figure 1. The client queue can correspond to a particular client device. In some implementations, the caching and distribution system can maintain multiple client queues, each client queue corresponding to a different client device.

[0053] The method 800 includes receiving a data node update (stage 840). The caching and distribution system can receive the data node update from the network device. The method 800 includes determining whether the node pointer map already includes a node pointer corresponding to the updated data node (decision block 850). If the node pointer map does not include a node pointer corresponding to the updated data node, the method includes adding a node pointer corresponding to the updated data node to the node pointer map with a count of zero, and adding the node pointer to the node pointer queue (stage 860). In some implementations, the node pointer queue can be a first-in-first-out (FIFO) queue that receives node-pointers at its input, and provides the node-pointers at its output in the order in which they were received.

[0054] If the node pointer map does include a node pointer corresponding to the updated data node, the method includes incrementing the node count corresponding to the updated data node, and not adding the node pointer to the node pointer queue (stage 870). In this manner, any particular node pointer is only added once to the node pointer queue between transmissions of network telemetry data to the client device. This can prevent frequently updated data nodes from being disproportionately represented in the set of network telemetry data transmitted to the client device and can ensure that a node that has only been updated once is transmitted to the client device. In some implementations, the client device will only receive one data value for a given data node per transmission. The node count, however, can

serve to inform the client device of how many data node updates it has not received since the previous transmission of network telemetry data. This information can be used for monitoring and diagnostic purposes, and inform an administrator of the client device that steps may need to be taken to improve the client device's ability to process all of the requested data.

[0055] The method 800 includes transmitting node-value, node-count pairs to the client device (stage 880). The caching and distribution system can transmit a set of node-value, node-count pairs to the client device. In some implementations, the caching and distribution system can begin transmitting the contents of the client queue after receiving indication from the client device that it is ready to receive data. The node-value, node-count pairs can include, for each node pointer in the node pointer queue, a data value in the data node corresponding to the node pointer, and the node count in the node pointer map corresponding to the node pointer. The caching and distribution system can use the node pointer queue to determine from which data nodes to pull data values for transmission to the client device. As stated above, the node pointer queue may only include a single copy of any particular node pointer, regardless of how many times the corresponding data node has been updated since a previous transmission of network telemetry data to the client device. In some implementations, the caching and distribution system can transmit the set of node-value, node-count pairs to the client device responsive to receiving a query from the client device, where the set of node-value, node-count pairs includes data responsive to the query. The caching and distribution system can provide network telemetry data to client devices in various manners. For example, the caching and distribution system can receive a query submitted from a client device, and provide a resulting data set in response. In another example, the caching and distribution system can receive a query submitted from a client device, provide the resulting data set in response, and provide updated resulting data sets in response to future requests from the client device, where the request need not repeat the query. In yet another example, the caching and distribution system can receive a query submitted from a client device, provide the resulting data set in response, and provide continual updates to the client device as they are received from the network devices. The caching and distribution system can implement other manners of providing network telemetry data to the client device without departing from the scope of this disclosure. In some implementations, a query can include a subpath representing an index node and all data nodes that are daughters, either directly or indirectly, of the index node specified by the subpath. In some implementations, the caching and distribution system can employ a scheme of read and write locks when adding, updating, or deleting data nodes and index nodes, as previously discussed with reference to Figures 4-6. In some implementations, the method 800 can continue cyclically by returning to stage 840, receiving additional data node updates, and repeating the intervening stages.

[0056] Figure 9 shows an example environment 900 including a network telemetry collection and translation system 910, according to an illustrative implementation. The environment 900 includes network devices 920a, 920b, and 920c (collectively "network devices 920") communicating with the collection and translation system 910 via a network 105. The network devices 920 can be similar to the network devices 120 previously described. In some cases, the network devices 920 may not communicate network telemetry data using a consistent or

standard format. For example, in some implementations, the network devices 920 may have come from different vendors, may represent different models or model lines from a vendor, or may span both current and legacy devices. Therefore, in some situations, network telemetry data produced by the various network devices 920 may benefit from translation into a vendor-neutral format prior to being stored by the caching and distribution system 110. Such a system may benefit from a collection and translation system 910 to process the various network telemetry data prior to use by the caching and distribution system 110.

[0057] The collection and translation system 910 can be a computing device such as the computing system 1510 described below with reference to Figure 14. The collection and translation system 910 includes a cache 930, a node map 935, a configuration library 940, a translator 945, a normalized data model 950, a type information database 955, and a collector 960.

[0058] The collector 960 can establish a connection with a network device, receive network telemetry data from the network device in the form of streaming update packets, unpack the packets, deserialize the relevant data, and pass the data on to the translator 945. The network telemetry data can include a data value and a data path that identifies the data. In some implementations, the data value and the data path may be in vendor-specific formats.

[0059] The normalized data model 950 can include data models based on a vendor neutral format; for example, OpenConfig. The normalized data model 950 can be expressed in a data modeling language such as YANG for all data related to the network devices 920. The normalized data model 950 can represent the vendor-neutral data model that the collection and translation system 910-and, by extension, the caching and distribution system 110-uses to present the network telemetry data to client devices. Basically, the standards will organize all data about a network device under a tree data structure. Any data about a network device can be accessed via a path defined by the normalized data model. This path can be referred to as a "standardized" or "vendor-neutral" path.

[0060] The type information database 955 can include compiled data type information defined in the normalized data model 950. The type information database 955 can be utilized by the translator 945 to perform type conversion for network telemetry data from vendor-specific formats to a vendor-neutral format.

[0061] The configuration library 940 can include configuration files with information that can indicate which vendor-specific paths should map to which vendor-neutral paths. In some implementations, the configuration files can be encapsulated into human-editable configuration files so that the code doing the actual mapping and/or translation can be more generic.

[0062] The translator 945 can receive the vendor-specific notifications from the collector 960, translate them into vendor-neutral notifications using the information from the type information database 955 and the configuration library 940, and pass the translated vendor-neutral notifications to the node map.

[0063] The node map 935 can store the vendor-neutral data previously translated by the translator 945. The node map 935 can store the vendor-neutral data in an in-memory cache, such as the cache 930, and notify a publisher for streaming the updates to subscribers such as the caching and distribution system 110 and/or the client devices 130.

[0064] The operations of the collection and translation system 910 are described in further detail below with regard to Figures 10-13.

[0065] Figure 10 illustrates different types of matches in an example translation tree 1000, according to an illustrative implementation. The translation tree 1000 includes a first-tier node A pointing to each of its second-tier daughter nodes B, C, D, and E.

[0066] The functionality of the translator 745 can be defined by its input and output as illustrated by the example translation tree 1000 and described in the following operation. The input to the translator 745 can be a vendor-specific path of a value, the value itself, and the type of the value. The output of the translator 745 can include a vendor-neutral path of the value and the translated value. The vendor-neutral path can be defined by a normalized data model as provided by the normalized data model 950, and the type information database 955. The type of the translated value can come from the same YANG models.

[0067] A path is a list of elements, either vendor-specific or vendor-neutral. Most often, a path is represented as a single string joined by elements with a pre-defined separator; for example, a slash "/". Since the separator can also appear inside an element, e.g. "/" inside an element "Ethernet3/10/1", an alternative approach that represents a path as a list/array/slice of individual elements can be more precise in delineating element boundaries. In the following examples, string representation for paths are used for simplicity, but the actual representation of a path (string representation or list representation) is not assumed. The solution presented below can address either the string or the list representation.

[0068] In a first example, the input string can be:

`"/device/interface/status/eth/phy/slice/Linecard3/intfStatus/Ethernet3/10/1/deviceName"`

[0069] And the output string can be:

`"/device/device/interfaces/interface/Ethernet3/10/1/name"`

[0070] As described above, "Ethernet3/10/1" may come as a single element if the path is given as a list representation, or it may be split into multiple elements if the path is given as a string representation. Either way, the solution presented below can correctly address it.

[0071] In the above example, the second "device" is device name and "Ethernet3/10/1" is an interface name. The two elements are dynamic parts of the paths and are common to both the vendor-specific path and the vendor-neutral path. There may be hundreds of network devices 930 and on each network device 930, there may also be hundreds or thousands of interfaces.

It may not be desirable to enumerate the network devices and interfaces in the configuration library 940, or in any other internal data structures, however. So the translator 945 may commence by parameterizing the paths. For example, after replacing the second device "device" with <device> and "Ethernet3/10/1" with <intf>, the input can become: "/<device>/interface/status/eth/phy/slice/Linecard3/intfStatus/<intf>/deviceName"

[0072] And the output can become:

"/device/<device>/interfaces/interface/<intf>/name"

[0073] The translator 945 can now can match vendor-specific path patterns as specified in the input. For a successful match, the translator 945 should be able to extract these named parameters and get a vendor-neutral path template as illustrated in the output. Then the translator 945 can expand the path template with the extracted named parameters to get the vendor-neutral path.

[0074] In some implementations, the above parameterized path pattern and templates method can employ a regular expression (regex) for each vendor-specific path pattern. The regular expression can have named groups to extract parameters. So, corresponding to the path patterns, there can exist a flat list of regular expressions. When a notification such as a network telemetry data update comes with a path, the translator 945 can match the path against the regular expressions sequentially until a match is found. The match can generate named parameters if any and also a vendor-neutral path template to expand.

[0075] This disclosure proposes a tree structure based approach to translation. For a translation tree, a path can be viewed as a list of elements. If a path is represented as a joined string, it can be converted into a list of elements by splitting at the separator. For example, interface names like "Ethernet3/10/1", can be split into multiple elements if they are not given as a single element. This approach provides ways to avoid complex and potentially fragile string-based processing.

[0076] In a translation tree, such as the translation tree 1000, each node can represent a path pattern. A leaf node represents a full path pattern for a scalar value. A link between a parent node and its child node represents a match of an element or multiple consecutive elements (both can be called as a span in a path pattern). There are several different types of matches, for example:

Literal Match: A single element matching a string literal. No parameter returned from this match.

Parameter Match: A single element of anything, and the element becomes a named parameter.

Anything Match: A single element of anything, but no parameter returned from this match.

Regex Match: A span of a single element or multiple consecutive elements matching a regular

expression, which may have named parameters.

Function Match: A span of a single element or multiple consecutive elements matching a pattern defined by a function, which may return named parameters. Function match is an enhanced version of regex match. It does not restrict the matching to be a simple regular expression match. It can also do more complex processing to get the extracted parameters.

[0077] The translation tree 1000 in Figure 10 illustrates the different types of matches. Suppose node A represents a path pattern `"<device>/interface"`. Also assume that a path matching already matched the first two elements of the path with the pattern represented by A.

[0078] The link between A and B is an example of a literal match with string literal `"status"`. Pattern B represents a path according to `"<device>/interface" + "/" + "status"`. Thus, if the first remaining element in the path is `"status"`, then the path matching can continue from A to B.

[0079] The link between A and C is an example of a parameter match with parameter of `"intf"`. Therefore, no matter what the first remaining element in the path is (as long as it is non-empty), the first remaining element becomes the value of parameter `"intf"`, and the path matching can continue from A to C.

[0080] The link between A and D is an example of an anything (`"*"`) match. No matter what the first remaining element in the path is (as long as it is non-empty), the path matching can continue from A to D with no parameters extracted.

[0081] The link between A and E is an example of a regex match. The regex can match the first remaining element or the first several consecutive elements. The regex can also have named groups which become named parameters for a match. For example, if a match is found for the regex, then any named groups becomes named parameters, and the path matching can continue from A to E.

[0082] Note that the examples given in Figure 10 are for illustration of different types of matches. The example translation tree 1000 can allow multiple matched nodes for a path pattern. For simplicity, however, a translation tree could be configured to have mutually exclusive matches pointing to its children. Having mutually exclusive nodes can make it easier to envision and maintain a large translation tree.

[0083] The translation tree 1000 is a condensed tree structure since each node represents a path pattern instead of a concrete path. The matches(links) between the nodes, except for literal matches, also describe a pattern.

[0084] Figure 11 illustrates an example of condensing a translation tree 1100 by a regular expression match, according to an illustrative implementation. There are different types of interfaces; e.g. `"phy"`, `"lag"`, `"vlan"`. They have the same set of attributes defined as interfaces.

But they have different elements in their paths. Without using regex match, the translation tree will look like the right tree in Figure 11. Nodes E3, E4, E5 will have to have to reproduce the same set of nodes as descendants, which could be hundreds of nodes depending on how many paths under the interface node. If a regex match is used, the translation tree will look like the left tree 1100 in Figure 11.

[0085] In some implementations, the translation tree can be built based on human generated configuration files. The program building the translation tree may not have the intelligence to condense the tree like the example in Figure 11 using regex matching. Therefore, the translator 945 may rely on some amount of human input to make the right choice and to use the right matches to describe different patterns. When used properly, it saves the work to enter the same set of attributes under different nodes.

[0086] Figure 12 illustrates using different matches 1200 and 1201 based on whether a given path is correctly split, according to an illustrative implementation. Some elements may contain the separators. For example, an interface name may be "Ethernet3", "Ethernet3/1", or "Ethernet3/10/1". If a path is given as a list of correctly split elements, then the interface name should be a single element even though it may have separators in it. One can use parameter match to get the single element as a parameter named as "intf" as shown in the match 1200.

[0087] If a path is given as a joined string or a list of incorrectly split elements, then the interface name could be in multiple elements ["Ethernet3", "10", "1"]. In this case, a regex match such as `(?P<intf>Ethernet\\d+(?:\\d+(?:\\d+)?)?)` can be used to extract the interface name from the path, as shown in the match 1201.

[0088] A leaf node, such as the leaf nodes E of matches 1200 and 1201 in Figure 12, represents a full-length path pattern for a scalar value. It should have a vendor-neutral path template for the scalar value. The vendor-neutral path template can be expanded with the named parameters to get the vendor-neutral path for a particular path. The mapping from a vendor-specific path pattern to the vendor-neutral path template can come from the configuration library 940.

[0089] A leaf node should also have the type information corresponding to the vendor-neutral path. When creating the leaf node, the translation tree builder looks up the type information from a "database" compiled from the normalized data models 950. The vendor-specific data have a given data type or an inferred data type. An automatic data type conversion can be performed if no complex processing is needed. For example, convert a string to an integer.

[0090] If simple type conversion does not suffice, a mapper function can be called to perform data translation. Mapper functions are discussed in further detail below with reference to Figure 13. The name of the function should be specified in configuration library 940 and cached in the leaf node when the translation tree is built.

[0091] Once a translation tree is built, the path matching process works as below for each path

in the coming notifications:

1. 1. split the path p into a list of elements if the path given is not already split.
2. 2. initialize an empty map m for the named parameters.
3. 3. node cur = root of the translation tree
4. 4. if p is an empty list, then return node cur and map m as a successful match.
5. 5. try to find a match from all matches at node cur.
6. 6. if no match is found, return nil node nil and nil map to indicate no match found for p.
7. 7. if a match is found, add named parameters to m if there are any in this match.
8. 8. remove matched elements from path p.
9. 9. cur = child node pointed by the match. goto step 4.

[0092] The translation tree approach described above has a few benefits compared with the "list of regular expressions" approach. No long and complex regular expressions matching a full path. The regular expressions are mostly used for matching a single element or a span of consecutive elements that are logically one element.

[0093] In addition, the amortized time complexity of the path matching process is roughly the depth of the translation tree because at each node the number of matches is small. And at each node, it only looks at a fraction of the path. For literal match, parameter match and anything match, the processing is very simple since only one element is involved. The depth of the translation tree is expected to be around 10. For the "list of regular expressions" approach, the time complexity of path matching process is mainly bound by the number of full path patterns, which are expected to be in the range of hundreds. And for each path pattern, it has to run a regular expression match for the full path.

[0094] Finally, it is easier to implement more powerful match like the function match using translation tree.

[0095] Figure 13 illustrates an example translator 945 internal architecture, according to an illustrative implementation. The translator 945 is part of a system 1300 that includes inputs from the configuration library 940 and the type information database 955. The translator 945 can output vendor-neutral notifications 920a and 920b (collectively "vendor-neutral notifications 920"). The translator 945 includes a configuration manager 1305, a translation tree builder 1310, a mapper manager 1315, a translation engine 1345, a configuration protocol buffer 1320, and a translation tree 1325.

[0096] The configuration manager 1305 monitors the changes in configuration files in the configuration library 940. The configuration manager 1305 reloads a configuration file when it detects a change in that configuration file. In some implementations, the configuration manager 1305 can employ a lock manager to control access to the configuration files in the configuration library 940. In some implementations, the configuration manager 1305 can use a distributed lock service. The configuration manager's 1305 use of files stored in the distributed

lock service can facilitate easy reloading of configuration changes when the configuration manager 1305 detects a change.

[0097] The translation tree builder 1310 can build translation trees. The translation tree structure is designed to facilitate translation of a vendor-specific path into a vendor-neutral path. On the leaf node, it also has the type information of the vendor-neutral scalar data. The translation tree builder 1310 can build a translation tree from the configuration protocol buffer 1320. The translation tree builder 1310 can also cache the type information into the leaf node by looking up the type information when building the tree data structure. Then, following the initial tree building, the translator 945 should not need to lookup type information in the database again.

[0098] The mapper manager 1315 can implement additional translation when necessary. When translating vendor-specific data to vendor-neutral data, there are cases where more complex processing than a simple type conversion may be needed. In these situations, the mapper 1315 can perform the translation. Mapper functions, such as the mapper functions 1-n, can be registered to Mapper Manager using unique names. Then in the configuration files, mapper function names can be specified to indicate that the translator 945 can call the corresponding mapper functions to perform the translation. The mapper functions can be applied to the data value as well as the named parameter extracted from the path by the translation tree 1325.

[0099] The translation engine 1345 can translate the vendor-specific notifications into vendor-neutral notifications using the translation tree 1325 and, in some cases, the mapper functions. A notification may include both deletes and updates. For deletes, the translation engine 1345 need only to translate the vendor-specific paths into vendor-neutral paths. For updates, the translation engine 1345 may need to perform type conversion of the values, if the vendor-specific format does not match the vendor-neutral format. If a mapper function name is specified in the configuration library 940, the corresponding leaf node in the translation tree 1325 will also have the mapper function name. The translation engine 1345 will find the function address using the function name through mapper manager 1315 and call the function to translate data.

[0100] In an example operation, the translator 945 can retrieve from the configuration library 940 information regarding which vendor-specific path translates into which vendor-neutral path, for all interested values. Because many values share the same common path prefix, the configuration file can be organized in two levels: PathMapEntry and ValueMapEntry. PathMapEntry is for all values under a common path prefix. ValueMapEntry is for an individual value. Because a ValueMapEntry may be inside a PathMapEntry, the key and std_name in ValueMapEntry can be relative to the vendor_path and std_path in the PathMapEntry. For example, each ValueMapEntry can tell the translator 945 that <vendor_path>/<key> maps to <std_path>/<std_name>. For example: // ValueMapEntry contains translation information from a vendor-specific // value to a vendor-neutral value message ValueMapEntry { // Required. Standardized vendor-neutral name for the value string std_name = 1; // Optional. The name of

a function that transforms the value before type // conversion and writing to the vendor-neutral tree. string mapper = 2; } // PathMapEntry contains translation information from a vendor-specific // path (and values under this path) to a vendor-neutral path message PathMapEntry { // Required. A string that specifies the pattern of the vendor-specific paths // matching this entry. string vendor_path = 1; // Required. Standardized vendor-neutral path template. string std_path = 2; // Required. Mapping for values (fields) under this path. map<string, ValueMapEntry> val_map = 3; } // PathMap contains translation information for all paths (and values // under the paths) message PathMap { repeated PathMapEntry entries = 1; }

[0101] In PathMapEntry, both vendor_path and key to val_map are path patterns. The concatenated path pattern <vendor_path>/<key> corresponds to scalar values. Because the path pattern specified in PathMap is used to build the translation tree 1325, there should be enough information for the builder to know which span in a path pattern should be converted to what match in a translation tree 1325. Instead of specifying the path pattern in a list of spans, we choose to specify it in a string, which should be more friendly for a human to read and edit the configuration file.

[0102] For example, here is how the string is parsed into spans which in turn are converted to matches in a translation tree:

"/" is still used to separate the spans. For spans corresponding to literal match, parameter match and anything match, the rule is simple.

"<device>" is a parameter match because it has "<" and ">" as the first and last characters.

"*" is an anything match,

[0103] Any span that is not a span for a parameter match, an anything match, a regex match, or a function match will be a span for a literal match.

[0104] In some implementations, the collection and translation system 110 may need to determine whether a span is a literal match or a regular expression match. The translation system 110 can use a heuristic to determine, with a certain probability, whether a string is a regular expression. It would be beneficial, however, if the translation system 110 could use a configuration file to determine precisely whether a given span is a regular expression. Similarly, it would be beneficial for the translation system 110 to know if the regular expression can match multiple elements in a path or not. (That is, if the translation system 110 can only match one element of the path, there may be no need to join the path for regular expression matching.) So a configuration file could use "{ {regex} }" to enclose a regular expression span. If the regex raw string has "/" in it, then the translation system 110 can consider it a potential-multiple-element regular expression span. Otherwise, the translation system 110 can consider it a single-element regular expression span. For example:

"/{Linecard(?P<lnum>\\d+)}" is a single-element regex span.

"/{({P<intf>Ethernet\\d+(?:\\d+(?:\\d+)?)})}" is a potential-multiple-element regex span.

[0105] If we ever need a function match, the span can be represented as:

"<para1, para2, ...>the-name-of-the-function()"

[0106] When parsing a span, if the first character is "<" and the last two characters are "()", then it is a function span.

[0107] Both std_path in PathMapEntry and std_name in ValueMapEntry can be path templates with parameters to expand. A path template is represented as a single string, with elements separated by "/". The elements can either be a literal or a parameter. A parameter is surrounded by "<" and ">". For example:

"/device/<device>/interfaces/interface/<intf>"

[0108] Below is an example configuration from a configuration file in the configuration library 940:

```
entries: {

    vendor_path:
    "/<device>/interface/counter/eth///(slice/phy/[^.]+|lag|vlan)///intfCounterDir///(?P<intf>Ethernet\\d+(?:\\d+(?:\\d+)?)?|Port-Channel\\d+)///intfCounter/current"
    std_path: "/device/<device>/interfaces/interface/<intf>/counters"
    val_map: {
        key: "statistics/inBroadcastPkts"
        value: { std_name: "in_broadcast_pkts" }
    }
    val_map: {
        key: "statistics/inDiscards"
        value: { std_name: "in_discards" }
    }
    val_map: {
        key: "statistics/inErrors"
        value: { std_name: "in_errors" }
    }
}
```

Meaning:

"/<device>/interface/counter/eth///(slice/phy/[^.]+|lag|vlan)///intfCounterDir///(?P<intf>Ethernet\\d+(?:\\d+(?:\\d+)?)?|Port-Channel\\d+)///intfCounter/current/statistics/inBroadcastPkts" maps to "/device/<device>/interfaces/interface/<intf>/counters/in_broadcast_pkts"

"/<device>/interface/counter/eth///(slice/phy/[^.]+|lag|vlan)///intfCounterDir///(?P<intf>Ethernet\\d+(?:\\d+(?:\\d+)?)?|Port-Channel\\d+)///intfCounter/current/statistics/inDiscards" maps to

```
"device/<device>/interfaces/interface/<intf>/counters/in_discards"
```

```
"<device>/interface/counter/eth///(slice/phy/[^/]+|lag|vlan)///intfCounterDir///(?P<intf>Ethernet\\d+(?:\\d+(?:\\d+)?|Port-Channel\\d+))///intfCounter/current/statistics/inErrors" maps to  
"device/<device>/interfaces/interface/<intf>/counters/in_errors"
```

[0109] Figure 14 is a block diagram illustrating a general architecture for a computer system 1500 that may be employed to implement elements of the systems and methods described and illustrated herein, according to an illustrative implementation. The computing system 1500 can be used to implement one or more of the devices described above, including the caching and distribution system 110, the network devices 120, the client devices 130, the collection and translation system 910, or the network devices 920. The computing system 1500 may be utilized in implementing the caching and distribution method 800 shown in Figure 8.

[0110] In broad overview, the computing system 1510 includes at least one processor 1550 for performing actions in accordance with instructions, and one or more memory devices 1570 or 1575 for storing instructions and data. The illustrated example computing system 1510 includes one or more processors 1550 in communication, via a bus 1515, with at least one network interface controller 1520 with one or more network interface ports 1522 connecting to one or more network devices 1524, memory 1570, and any other devices 1580; e.g., an I/O interface. Generally, a processor 1550 will execute instructions received from memory. The processor 1550 illustrated incorporates, or is directly connected to, cache memory 1575.

[0111] In more detail, the processor 1550 may be any logic circuitry that processes instructions, e.g., instructions fetched from the memory 1570 or cache 1575. In many embodiments, the processor 1550 is a microprocessor unit or special purpose processor. The computing device 1500 may be based on any processor, or set of processors, capable of operating as described herein. In some implementations, the processor 1550 can be capable of executing the caching and distribution method 800 shown in Figure 8. The processor 1550 may be a single core or multi-core processor. The processor 1550 may be multiple processors. In some implementations, the processor 1550 can be configured to run multi-threaded operations. In some implementations, the processor 1550 may host one or more virtual machines or containers, along with a hypervisor or container manager for managing the operation of the virtual machines or containers. In such implementations, the caching and distribution method 800 shown in Figure 8 can be implemented within the virtualized or containerized environments provided on the processor 1550.

[0112] The memory 1570 may be any device suitable for storing computer readable data. The memory 1570 may be a device with fixed storage or a device for reading removable storage media. Examples include all forms of non-volatile memory, media and memory devices, semiconductor memory devices (e.g., EPROM, EEPROM, SDRAM, and flash memory devices), magnetic disks, magneto optical disks, and optical discs (e.g., CD ROM, DVD-ROM, and Blu-ray® discs). A computing system 1500 may have any number of memory devices 1570. In some implementations, the memory 1570 can include instructions corresponding to the caching and distribution method 800 shown in Figure 8. In some implementations, the

memory 1570 supports virtualized or containerized memory accessible by virtual machine or container execution environments provided by the computing system 1510.

[0113] The cache memory 1575 is generally a form of computer memory placed in close proximity to the processor 1550 for fast read times. In some implementations, the cache memory 1575 is part of, or on the same chip as, the processor 1550. In some implementations, there are multiple levels of cache 1575, e.g., L2 and L3 cache layers.

[0114] The network interface controller 1520 manages data exchanges via the network interfaces 1522 (also referred to as network interface ports). The network interface controller 1520 handles the physical and data link layers of the OSI model for network communication. In some implementations, some of the network interface controller's tasks are handled by the processor 1550. In some implementations, the network interface controller 1520 is part of the processor 1550. In some implementations, a computing system 1510 has multiple network interface controllers 1520. The network interfaces 1522 are connection points for physical network links. In some implementations, the network interface controller 1520 supports wireless network connections and an interface port 1522 is a wireless receiver/transmitter. Generally, a computing device 1510 exchanges data with other network devices 1524 via physical or wireless links to a network interfaces 1522. In some implementations, the network interface controller 1520 implements a network protocol such as Ethernet.

[0115] The other network devices 1524 are connected to the computing device 1510 via a network interface port 1522. The other network devices 1524 may be peer computing devices, network devices, or any other computing device with network functionality. For example, a first network device 1524 may be a network device such as a hub, a bridge, a switch, or a router, connecting the computing device 1510 to a data network such as the Internet.

[0116] The other devices 1580 may include an I/O interface, external serial device ports, and any additional co-processors. For example, a computing system 1510 may include an interface (e.g., a universal serial bus (USB) interface) for connecting input devices (e.g., a keyboard, microphone, mouse, or other pointing device), output devices (e.g., video display, speaker, or printer), or additional memory devices (e.g., portable flash drive or external media drive). In some implementations, a computing device 1500 includes an additional device 1580 such as a coprocessor, e.g., a math co-processor can assist the processor 1550 with high precision or complex calculations.

[0117] Implementations of the subject matter and the operations described in this specification can be implemented in digital electronic circuitry, or in computer software embodied on a tangible medium, firmware, or hardware, including the structures disclosed in this specification and their structural equivalents, or in combinations of one or more of them. Implementations of the subject matter described in this specification can be implemented as one or more computer programs embodied on a tangible medium, i.e., one or more modules of computer program instructions, encoded on one or more computer storage media for execution by, or to control the operation of, a data processing apparatus. A computer storage medium can be, or be

included in, a computer-readable storage device, a computer-readable storage substrate, a random or serial access memory array or device, or a combination of one or more of them. The computer storage medium can also be, or be included in, one or more separate components or media (e.g., multiple CDs, disks, or other storage devices). The computer storage medium may be tangible and non-transitory.

[0118] The operations described in this specification can be implemented as operations performed by a data processing apparatus on data stored on one or more computer-readable storage devices or received from other sources. The operations may be executed within the native environment of the data processing apparatus or within one or more virtual machines or containers hosted by the data processing apparatus.

[0119] A computer program (also known as a program, software, software application, script, or code) can be written in any form of programming language, including compiled or interpreted languages, declarative or procedural languages, and it can be deployed in any form, including as a stand-alone program or as a module, component, subroutine, object, or other unit suitable for use in a computing environment. A computer program may, but need not, correspond to a file in a file system. A program can be stored in a portion of a file that holds other programs or data (e.g., one or more scripts stored in a markup language document), in a single file dedicated to the program in question, or in multiple coordinated files (e.g., files that store one or more modules, sub programs, or portions of code). A computer program can be deployed to be executed on one computer or on multiple computers or one or more virtual machines or containers that are located at one site or distributed across multiple sites and interconnected by a communication network. Examples of communication networks include a local area network ("LAN") and a wide area network ("WAN"), an inter-network (e.g., the Internet), and peer-to-peer networks (e.g., ad hoc peer-to-peer networks).

[0120] The processes and logic flows described in this specification can be performed by one or more programmable processors executing one or more computer programs to perform actions by operating on input data and generating output. The processes and logic flows can also be performed by, and apparatus can also be implemented as, special purpose logic circuitry, e.g., an FPGA (field programmable gate array) or an ASIC (application specific integrated circuit).

[0121] While this specification contains many specific implementation details, these should not be construed as limitations on the scope of any inventions or of what may be claimed, but rather as descriptions of features specific to particular implementations of particular inventions. Certain features that are described in this specification in the context of separate implementations can also be implemented in combination in a single implementation. Conversely, various features that are described in the context of a single implementation can also be implemented in multiple implementations separately or in any suitable sub-combination. Moreover, although features may be described above as acting in certain combinations and even initially claimed as such, one or more features from a claimed combination can in some cases be excised from the combination, and the claimed combination

may be directed to a sub-combination or variation of a sub-combination.

[0122] Similarly, while operations are depicted in the drawings in a particular order, this should not be understood as requiring that such operations be performed in the particular order shown or in sequential order, or that all illustrated operations be performed, to achieve desirable results. In certain circumstances, multitasking and parallel processing may be advantageous. Moreover, the separation of various system components in the implementations described above should not be understood as requiring such separation in all implementations, and it should be understood that the described program components and systems can generally be integrated together in a single software product or packaged into multiple software products.

[0123] References to "or" may be construed as inclusive so that any terms described using "or" may indicate any of a single, more than one, and all of the described terms. The labels "first," "second," "third," and so forth are not necessarily meant to indicate an ordering and are generally used merely to distinguish between like or similar items or elements.

REFERENCES CITED IN THE DESCRIPTION

Cited references

This list of references cited by the applicant is for the reader's convenience only. It does not form part of the European patent document. Even though great care has been taken in compiling the references, errors or omissions cannot be excluded and the EPO disclaims all liability in this regard.

Patent documents cited in the description

- [US80704517 \[0001\]](#)
- [US2016321652A1 \[0002\]](#)
- [US34387616 \[0014\]](#)

P A T E N T K R A V

1. Fremgangsmåde til at oversætte netværkstelemetridata til mellemlagring og fordeling, hvilken fremgangsmåde omfatter:

5 fra en netværksindretning (120a-120b) at modtage (810) et sæt af netværkstelemetridata i ét eller flere leverandørspecifikke formater, hvor sættet af netværkstelemetridata omfatter dataværdier, der har én eller flere leverandørspecifikke datatyper;

at oversætte netværkstelemetridataet til et leverandørneutralt format, hvor dataværdierne konverteres til en leverandørneutral datatype;

10 at lagre (820) det oversatte netværkstelemetridata som en flerhed af dataknudepunkter;

at opretholde (830) en klientkø, som har en knudepunktsmarkørkø og en knudepunktsmarkørafbildning, hvor knudepunktsmarkørafbildningen omfatter en flerhed af knudepunktsmarkør-, knudepunktstællerpar, hvor hver knude-

15 punktsmarkør indikerer en sti til et dataknudepunkt svarende til knudepunktsmarkøren, og knudepunktstælleren indikerer et antal af opdateringer af dataknudepunktet siden en forudgående transmission af indholdet af dataknudepunktet til en klientindretning (130a-130b);

fra netværksindretningen at modtage (840) leverandørspecifikke netværkstelemetriopdateringsdata i ét af de leverandørspecifikke formater;

20 at oversætte det leverandørspecifikke netværkstelemetriopdateringsdata til det leverandørneutrale format, og at opdatere det tilsvarende dataknudepunkt af flerheden af dataknudepunkter; og

til klientindretningen at transmittere (880) et sæt af knudepunktsværdi-, knudepunktstællerpar, som for hver knudepunktsmarkør i knudepunktsmarkørkøen omfatter en konverteret neutral dataværdi i dataknudepunktet svarende til knudepunktsmarkøren, og hvor knudepunktstælleren i knude-

25 punktsmarkørafbildningen svarer til knudepunktsmarkøren.

30 2. Fremgangsmåde ifølge krav 1, hvor oversættelsen af sættet af netværkstelemetri-data desuden omfatter at konvertere datastier til stier i overensstemmelse med en forud bestemt neutral datamodel (950).

3. Fremgangsmåde ifølge krav 2, hvor oversættelsen udføres i overensstemmelse med et oversættelsestræ, som har mindst to lag af knudepunkter, hvor hvert af knude-

35 punkterne repræsenterer et stimønster i overensstemmelse med den forud bestemte neutrale datamodel.

4. Fremgangsmåde ifølge krav 3, hvor oversættelsen udføres ved at matche elemen-

ter i de forud bestemte stier med knudepunkterne i oversættelsestræet.

5. Fremgangsmåde ifølge krav 3, hvor oversættelsen udføres ved anvendelse af én eller flere afbildningsfunktioner.

5

6. Fremgangsmåde ifølge ét af kravene 3 til 5, som desuden omfatter:

- at indlæse én eller flere konfigurationsfiler fra et konfigurationsbibliotek;
- at opbygge oversættelsestræet baseret på de indlæste konfigurationsfiler.

10

7. Fremgangsmåde ifølge krav 6, som desuden omfatter:

- at detektere en ændring i én af konfigurationsfilerne;
- at låse adgang til konfigurationsbiblioteket;
- at genindlæse konfigurationsfilerne for at genopbygge oversættelsestræet.

15

8. Fremgangsmåde ifølge ét af kravene 1 til 7, som desuden omfatter:

- hvis dataknudepunktsopdateringen svarer til et dataknudepunkt, som har en tilsvarende knudepunktsmarkør, der ikke allerede forefindes i datamarkørafbildningen: at tilføje (860) den tilsvarende knudepunktsmarkør til knudepunktsmarkørafbildningen med en tæller på nul, og at tilføje den tilsvarende knudepunktsmarkør til knudepunktsmarkørkøen.

20

9. Fremgangsmåde ifølge ét af kravene 1 til 8, som desuden omfatter:

- hvis dataknudepunktsopdateringen svarer til et dataknudepunkt, som har en tilsvarende knudepunktsmarkør, der allerede forefindes i datamarkørafbildningen: at øge (870) knudepunktstælleren for det tilsvarende knudepunkt i knudepunktsmarkørafbildningen, og ikke at tilføje den tilsvarende knudepunktsmarkør til knudepunktsmarkørkøen.

25

10. Netværkstelemetrioversættelsessystem, som omfatter én eller flere processorer konfigureret til:

30

- fra en netværksindretning (120a-120b) at modtage et sæt af netværkstelemetridata i ét eller flere leverandørspecifikke formater, hvor netværkstelemetridataet omfatter dataværdier, der har én eller flere leverandørspecifikke datatyper;

35

- at oversætte sættet af netværkstelemetridata til et leverandørneutralt format, hvor dataværdierne konverteres til en leverandørneutral datatype;
- at lagre det oversatte netværkstelemetridata som en flerhed af dataknudepunkter (210);
- at opretholde en klientkø, som har en knudepunktsmarkørkø og en knude-

punktsmarkørafbildning, hvor knudepunktsmarkørafbildningen omfatter en
 flerhed af knudepunktsmarkør-, knudepunktstællerpar, hvor hver knude-
 punktsmarkør indikerer en sti til et dataknudepunkt svarende til en knude-
 punktsmarkør, og knudepunktstælleren indikerer et antal af opdateringer af
 5 dataknudepunktet siden en forudgående transmission af indholdet af data-
 knudepunktet til en klientindretning (130a-130b);
 fra netværksindretningen at modtage et leverandørspecifikt netværksteleme-
 triopdateringsdata i ét af de leverandørspecifikke formater;
 at oversætte det leverandørspecifikke netværkstelemetriopdateringsdata til
 10 det leverandørneutrale format, og at opdatere det tilsvarende dataknude-
 punkt af flerheden af dataknudepunkter; og
 til klientindretningen at transmittere et sæt af knudepunktsværdi-, knude-
 punktstællerpar, som for hver knudepunktsmarkør i knudepunktsmarkørkøen
 omfatter en konverteret neutral dataværdi i dataknudepunktet svarende til
 15 knudepunktsmarkøren, og hvor knudepunktstælleren i knudepunktsmarkøraf-
 bildningen svarer til knudepunktsmarkøren.

11. Netværkstelemetrioversættelsessystem ifølge krav 10, hvor oversættelsen af
 sættet af netværkstelemetridata desuden omfatter at konvertere datastier til stier i over-
 20 ensstemmelse med en forud bestemt neutral datamodel.

12. Netværkstelemetrioversættelsessystem ifølge krav 11, hvor oversættelsen udfø-
 res i overensstemmelse med et oversættelsestræ, som har mindst to lag af knudepunkter,
 hvor hver af knudepunkterne repræsenterer et stimønster i overensstemmelse med den
 25 forud bestemte neutrale datamodel.

13. Netværkstelemetrioversættelsessystem ifølge krav 12, hvor oversættelsen udfø-
 res ved at matche elementer i de forud bestemte stier med knudepunkter i oversættelses-
 træet.

DRAWINGS

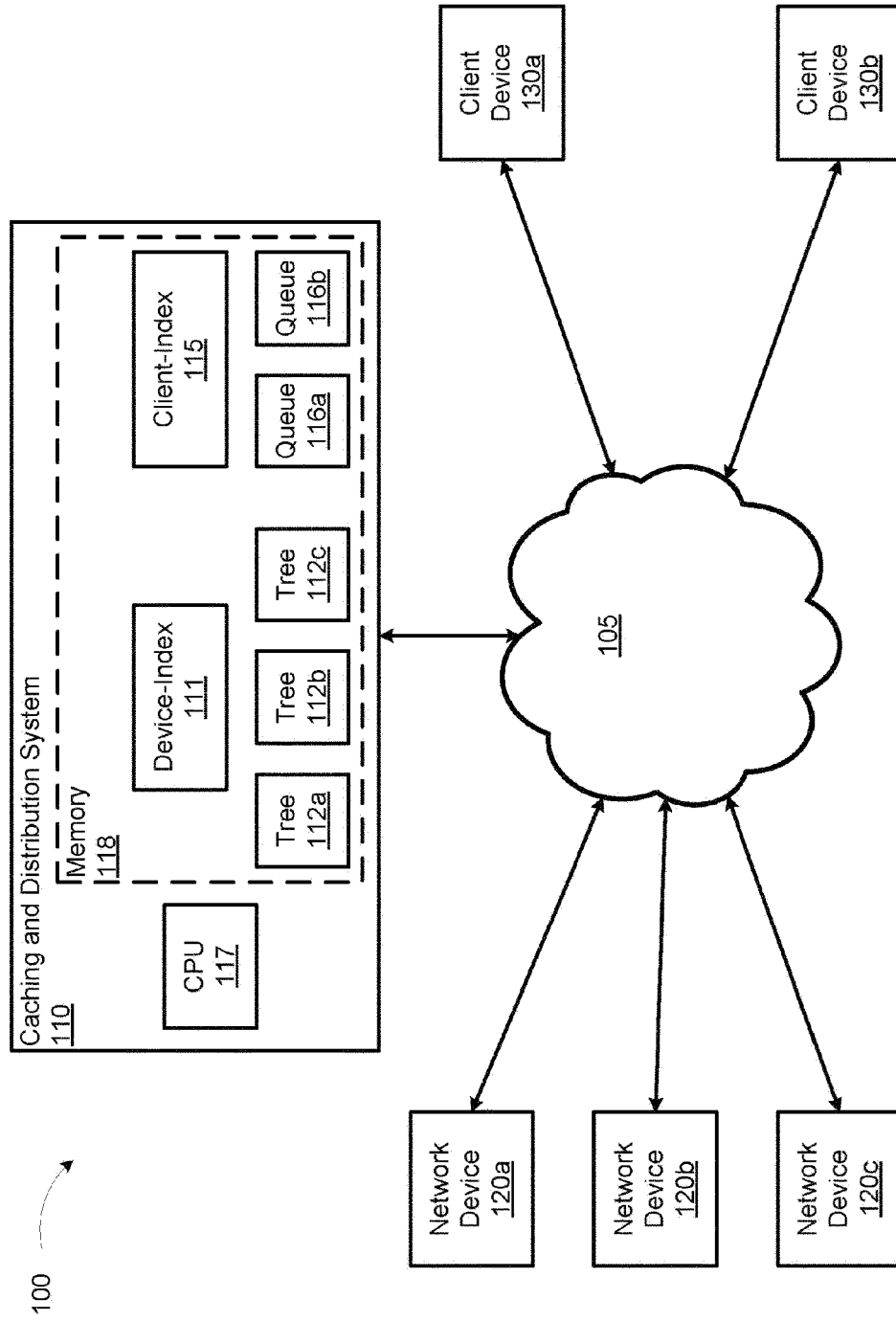


FIG. 1

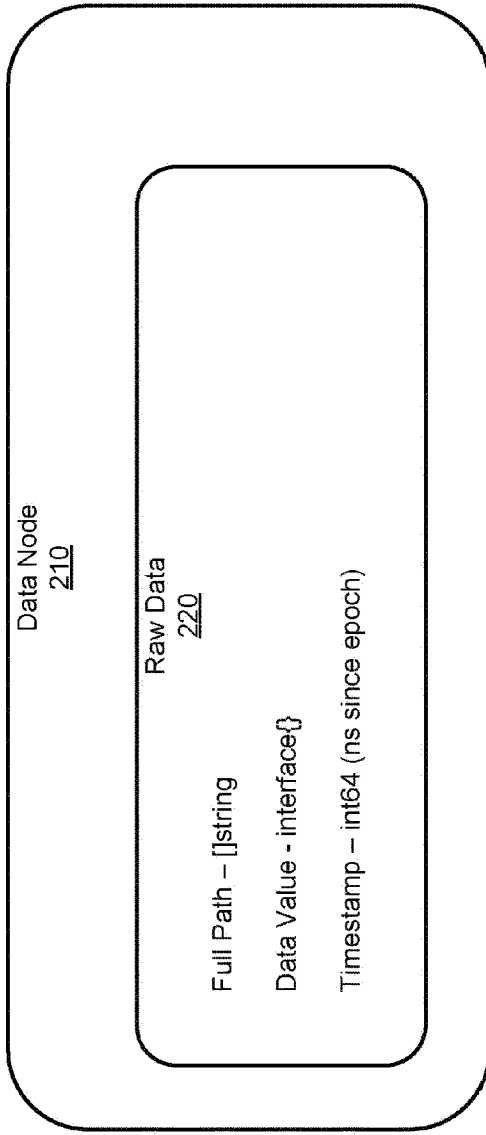
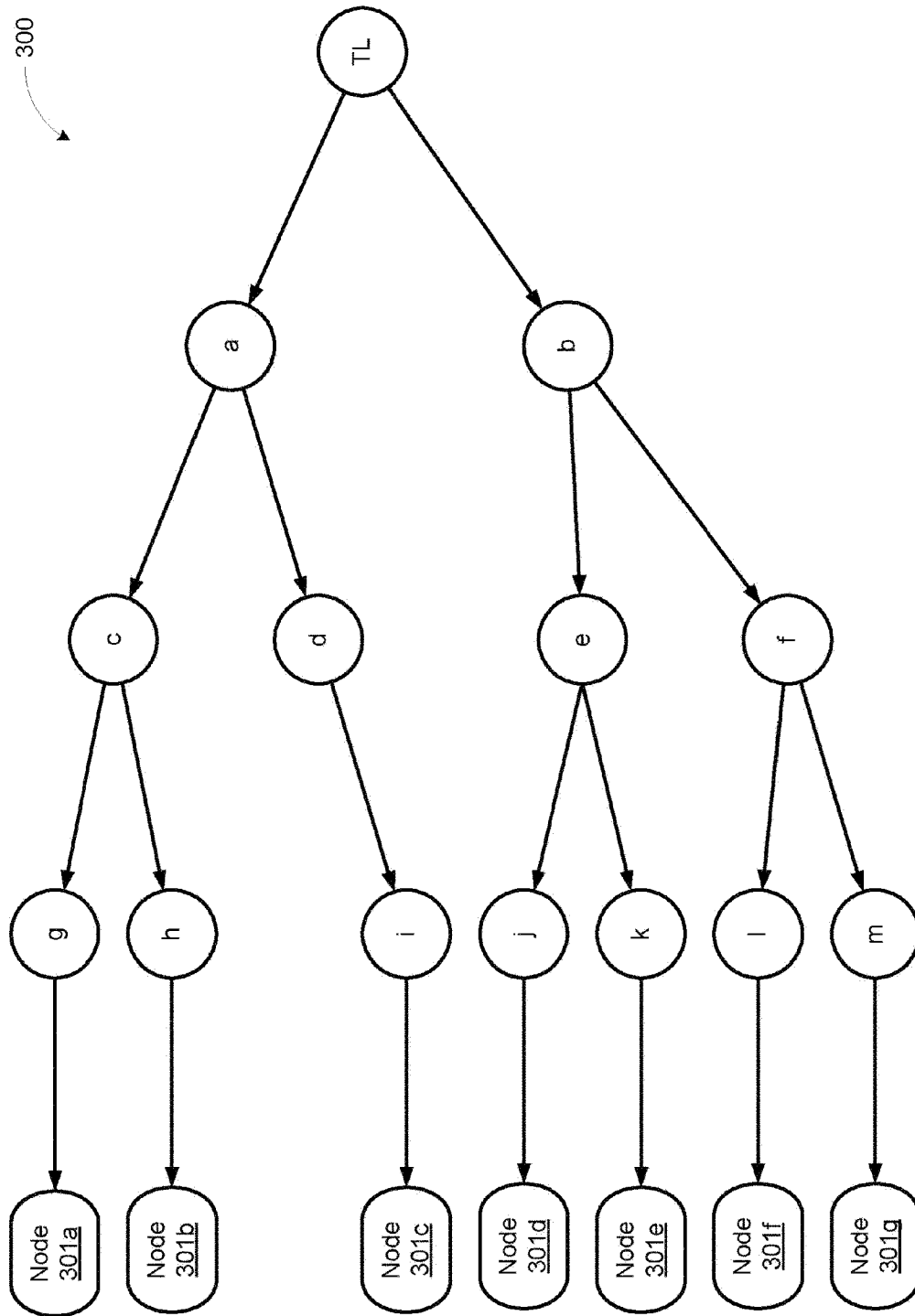
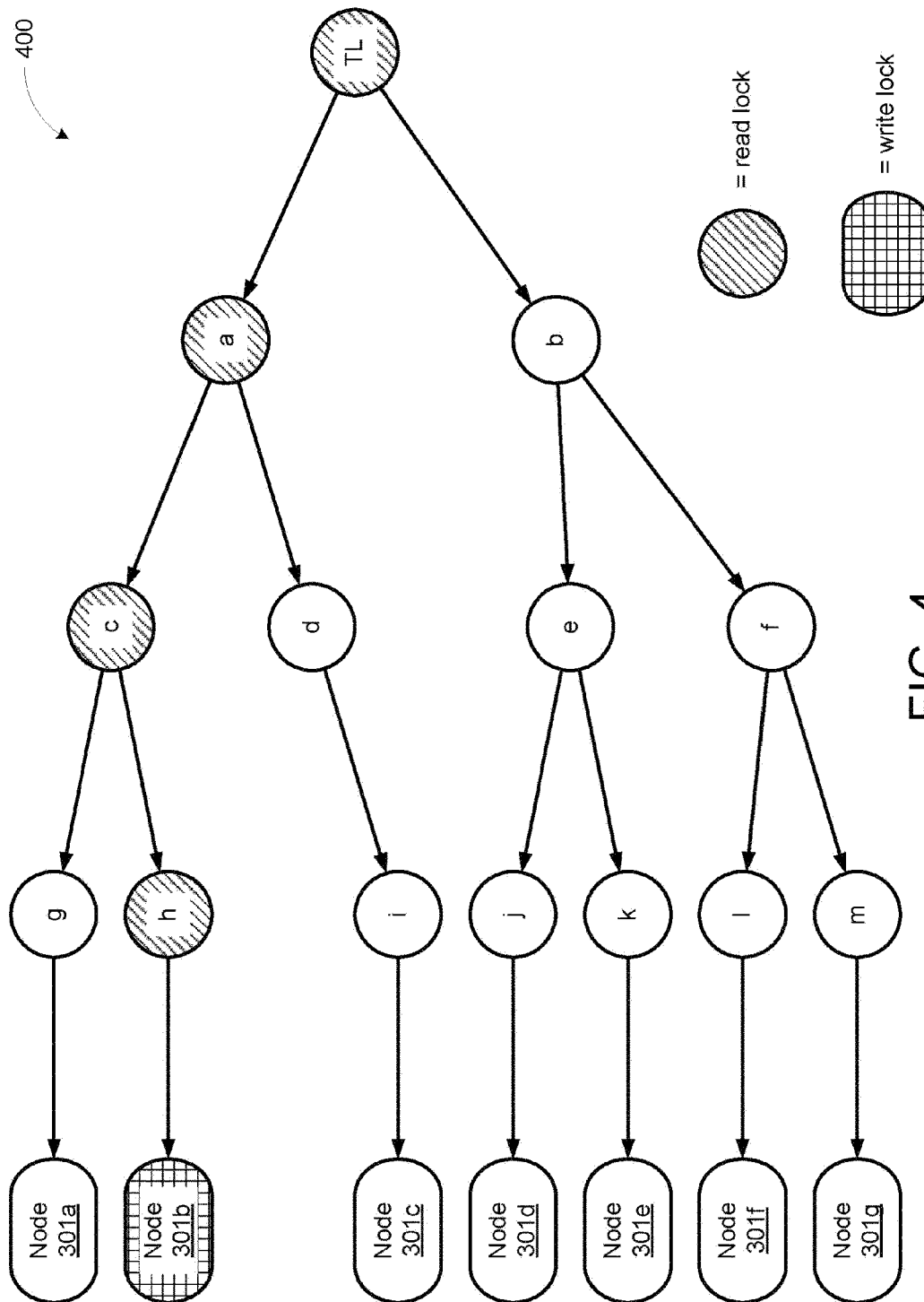
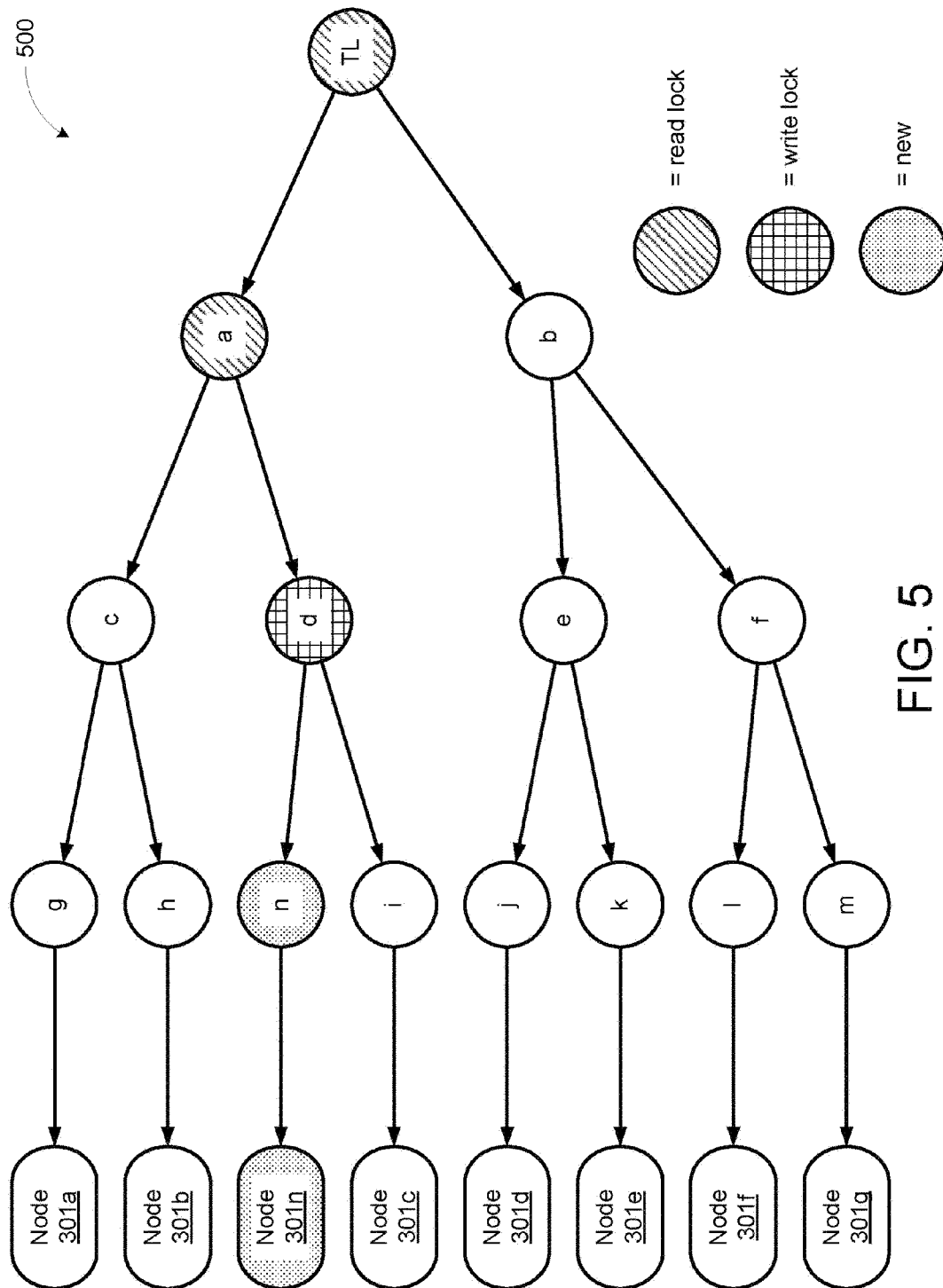


FIG. 2







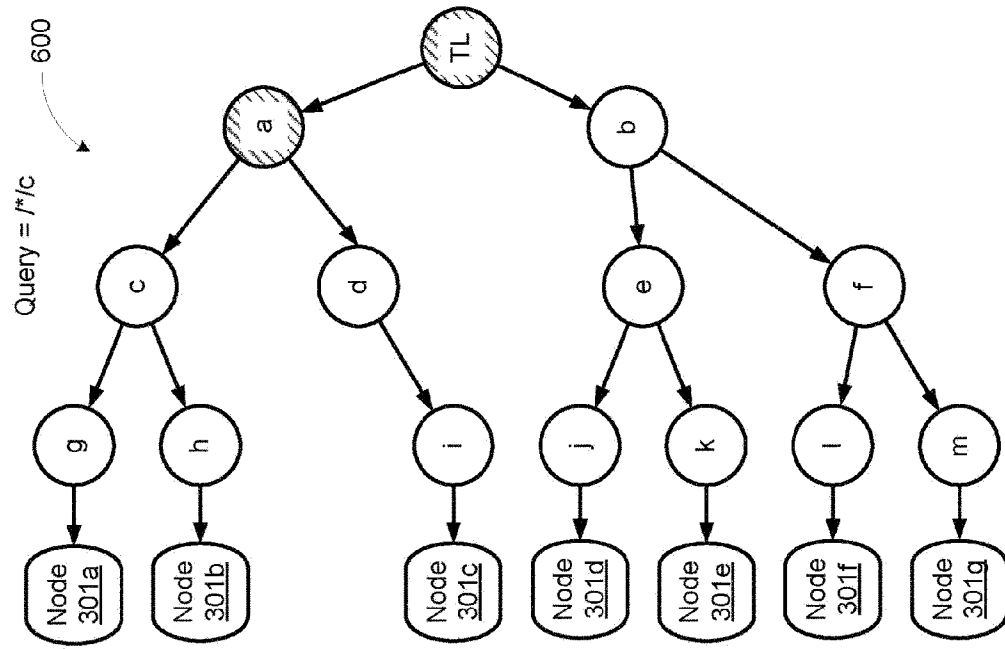


FIG. 6B

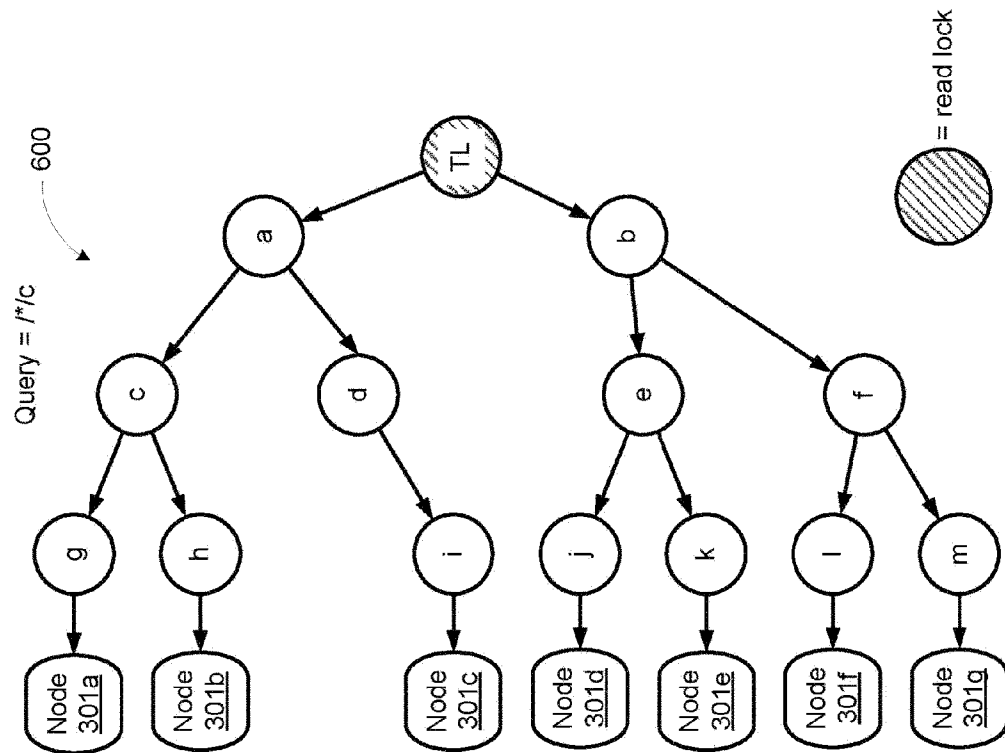


FIG. 6A

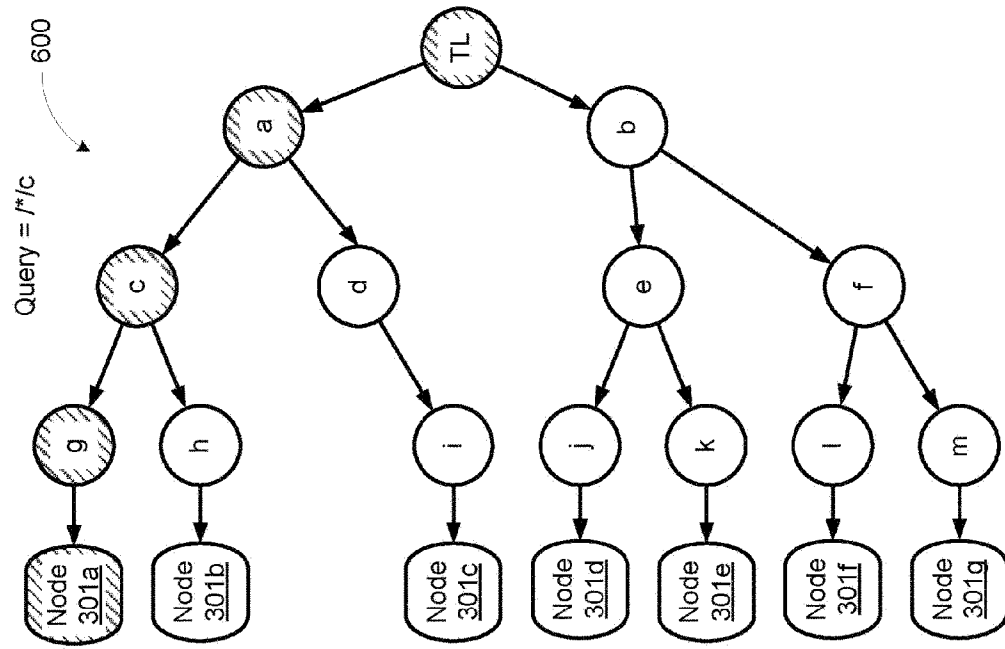


FIG. 6D

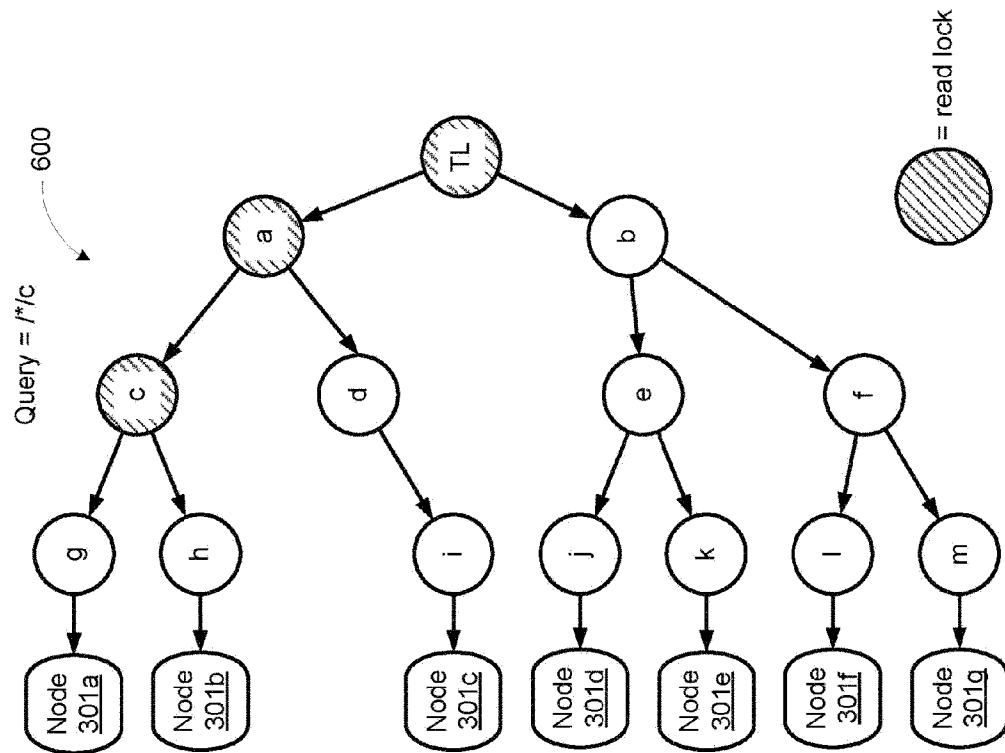


FIG. 6C

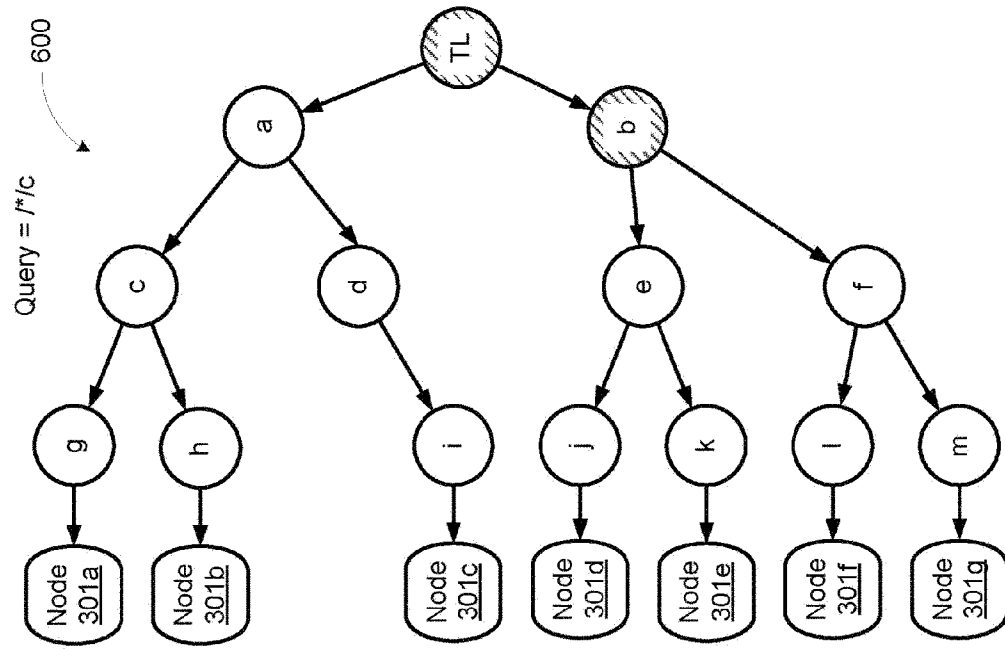


FIG. 6F

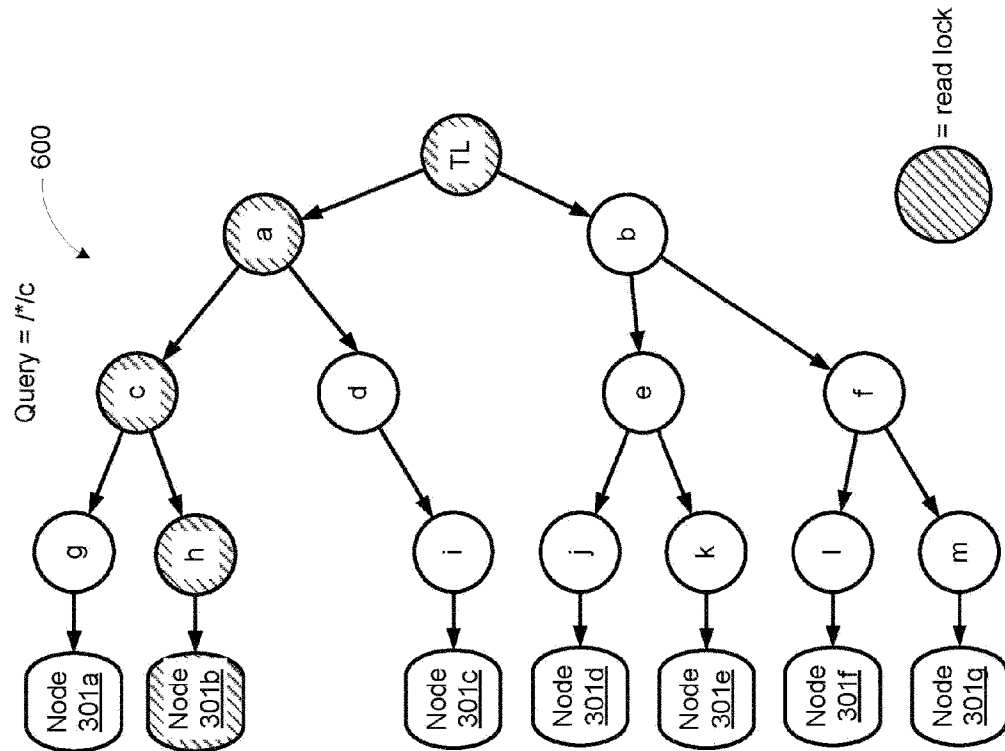


FIG. 6E

700

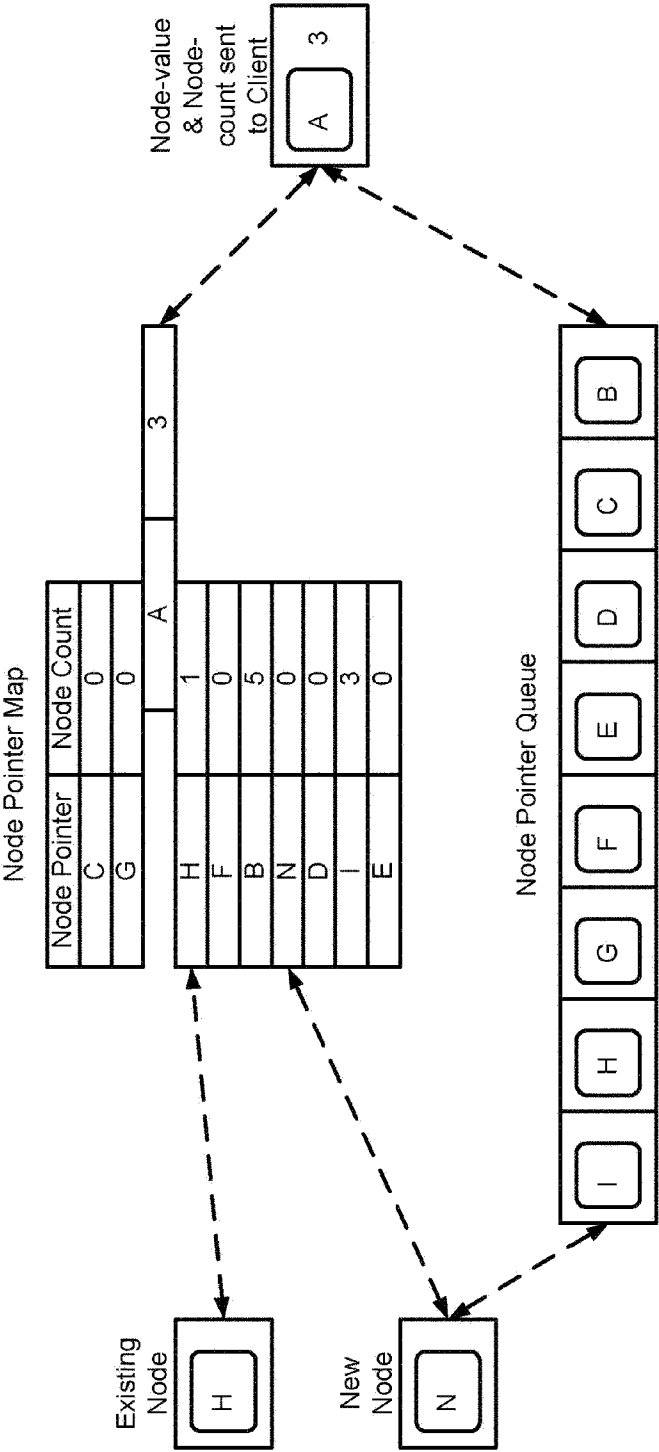


FIG. 7

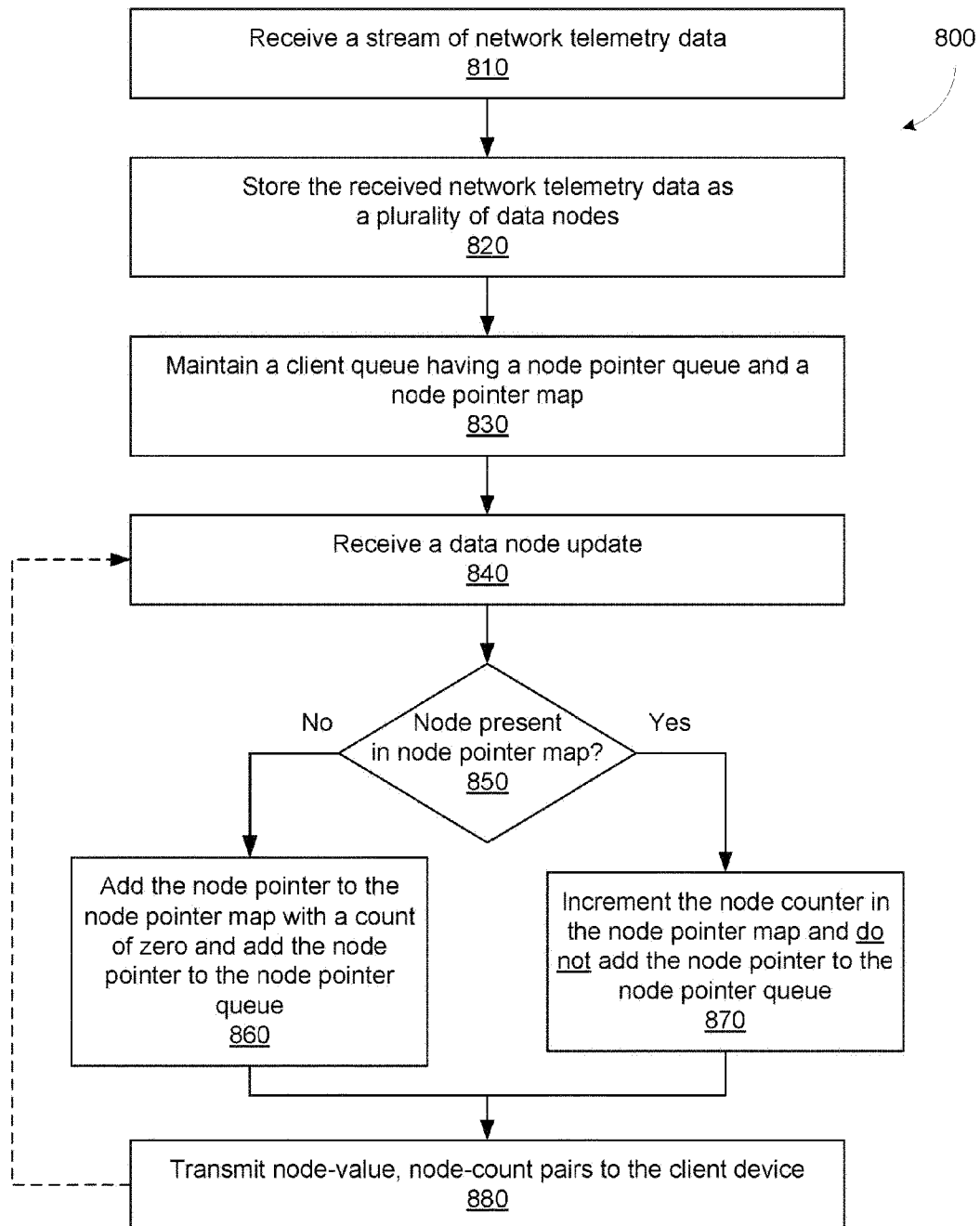


FIG. 8

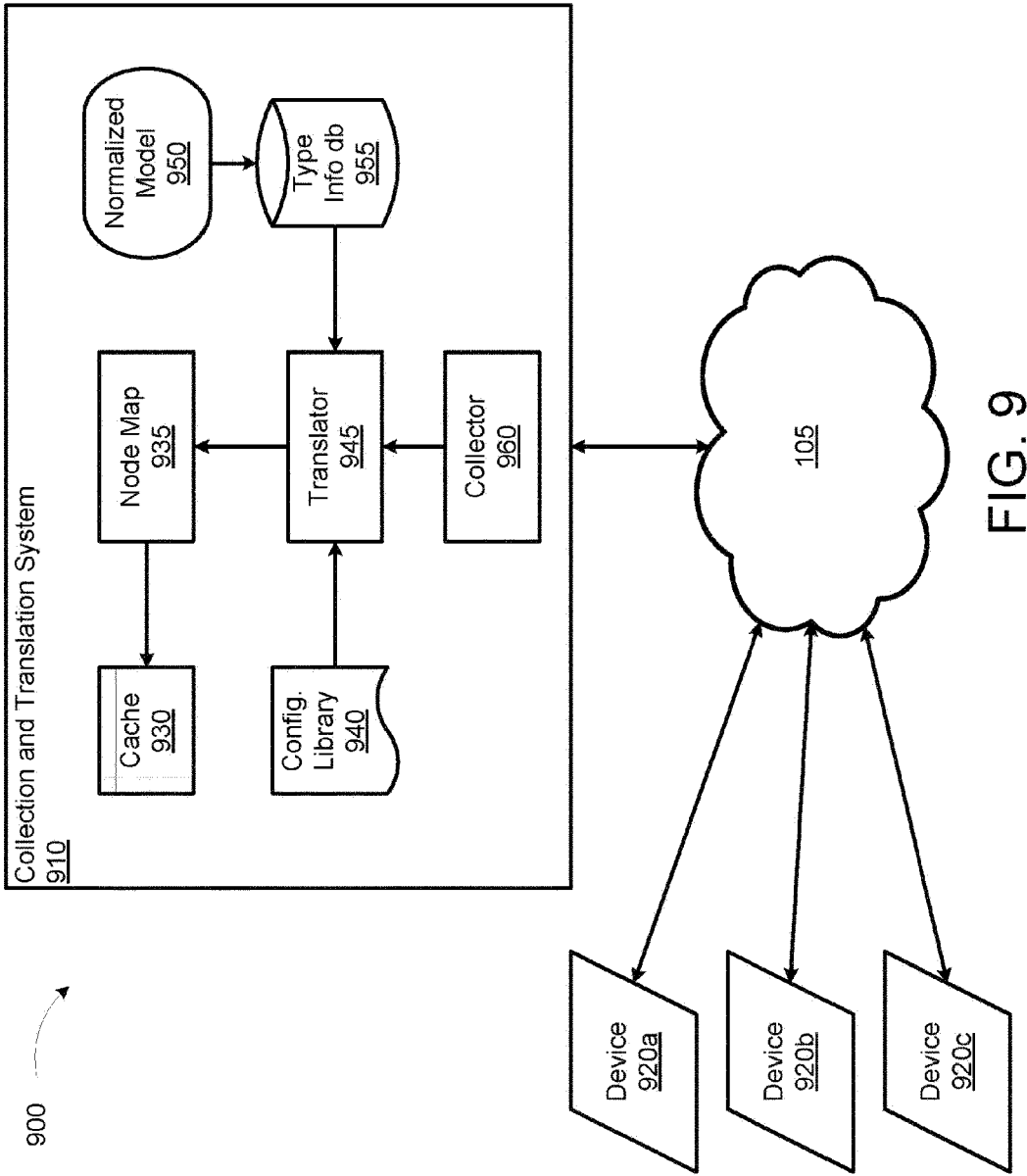


FIG. 9

1000

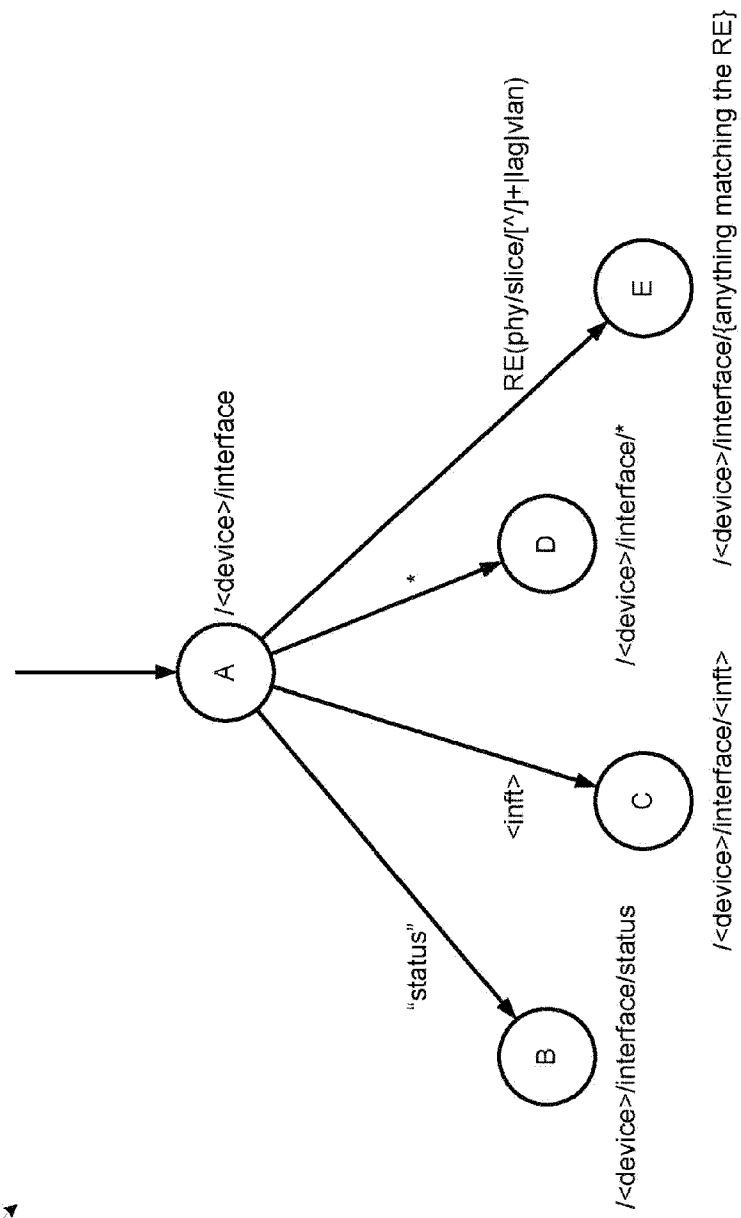


FIG. 10

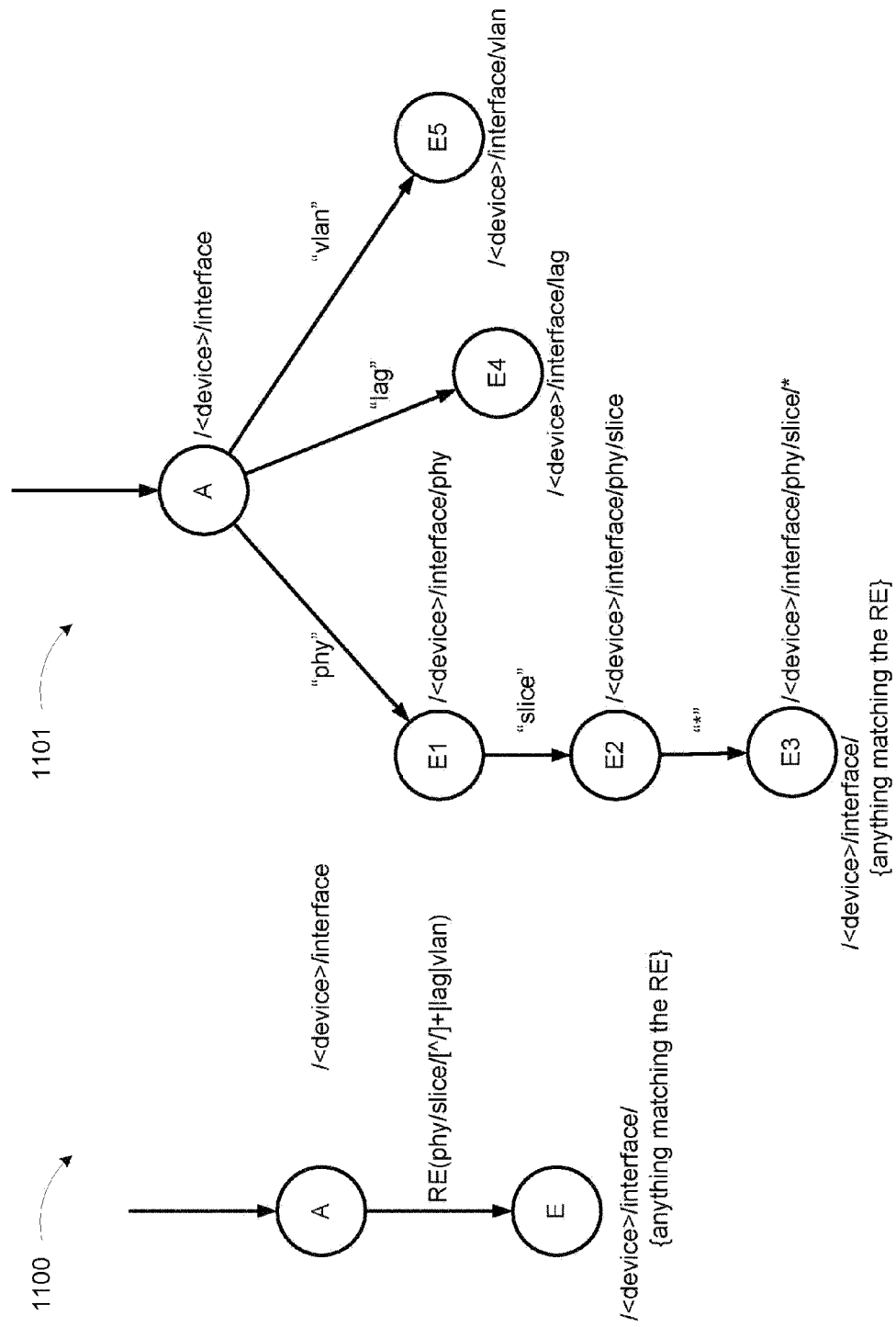


FIG. 11

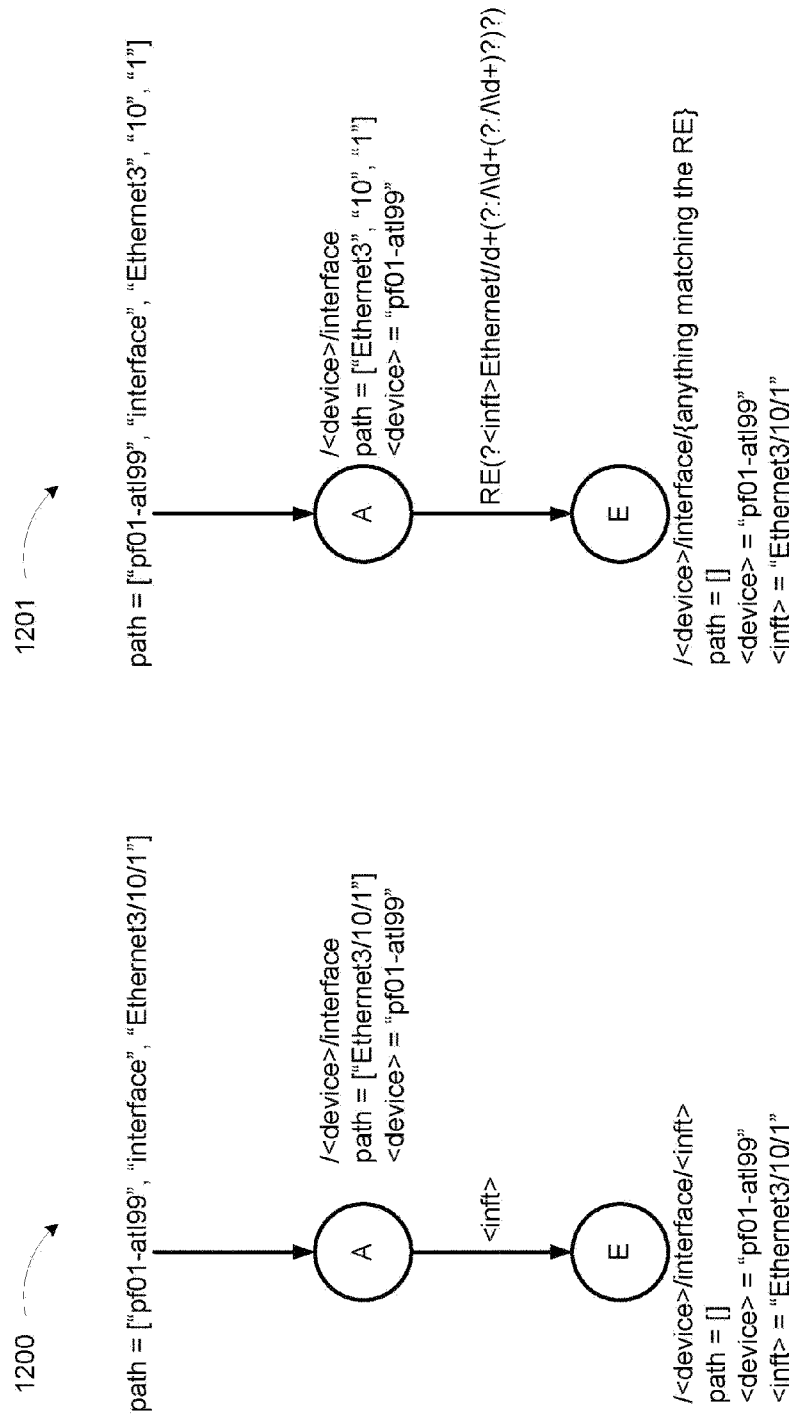
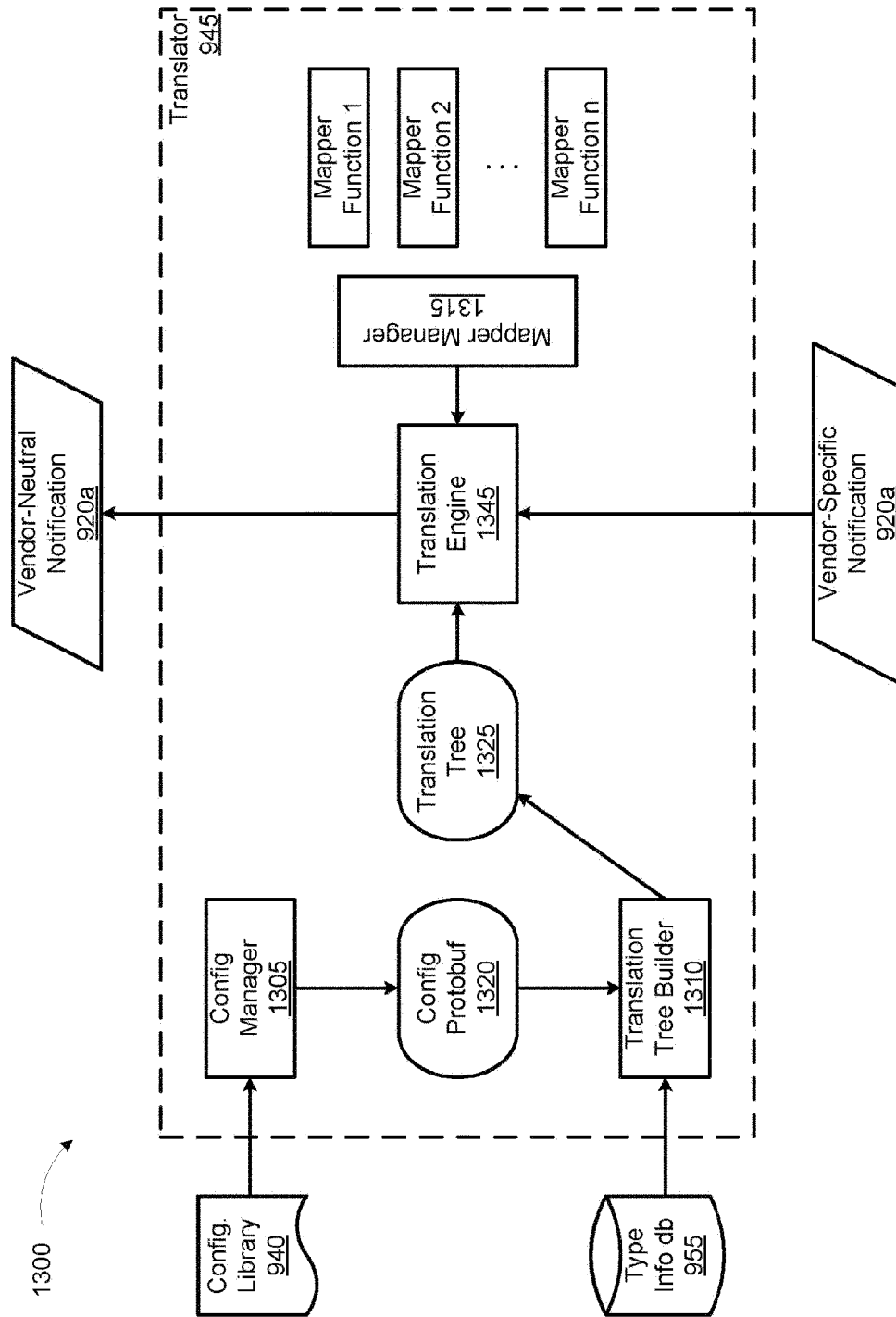


FIG. 12



1500

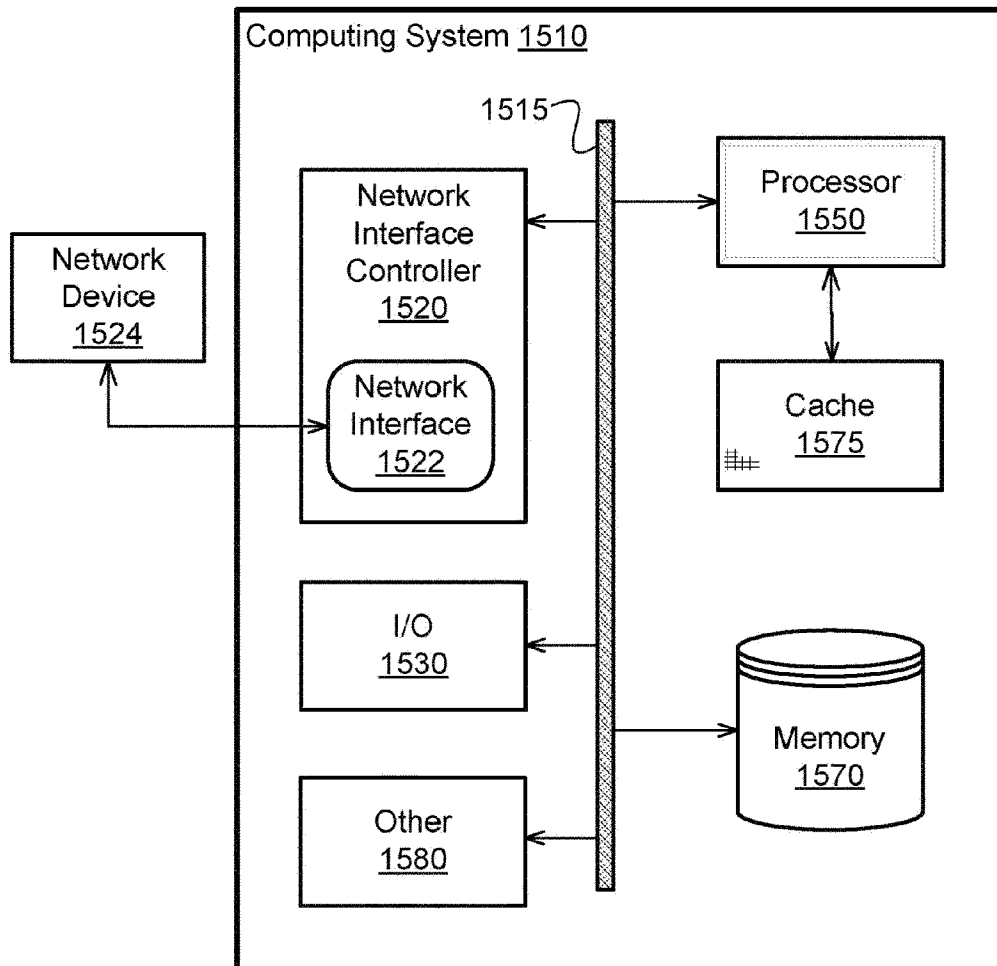


FIG. 14