UC Santa Cruz

UC Santa Cruz Previously Published Works

Title

Loop-Free Routing in Computer Networks and the Internet

Permalink

https://escholarship.org/uc/item/6pm527mh

Journal

World Intellectual Property Organization, issue WO 2026/015977 A1

Authors

Garcia-Luna-Aceves, J.J.
Moghaddassian, Morteza

Publication Date

2026-01-22

Data Availability

The data associated with this publication are within the manuscript.

Peer reviewed

eScholarship.org Powered by the California Digital Library
University of California

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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
22 January 2026 (22.01.2026)

WIPO | PCT

(10) International Publication Number
WO 2026/015977 A1

(51) International Patent Classification:
H04L 45/18 (2022.01)

(21) International Application Number:
PCT/CA2025/050978

(22) International Filing Date:
15 July 2025 (15.07.2025)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
63/673,359 19 July 2024 (19.07.2024) US

(71) Applicant: **THE GOVERNING COUNCIL OF THE UNIVERSITY OF TORONTO** [CA/CA]; 100 College Street, Room 413, Banting Institute, Toronto, Ontario M5G 1L5 (CA).

(72) Inventors: **GARCIA-LUNA-ACEVES, Jose Joaquin**; 88 Cumberland St. No. 2505, Toronto, Ontario M5R 0C8 (CA). **MOGHADDASSIAN, Morteza**; 1610-20 Bloorview Pl, North York, Ontario M2J 0A6 (CA).

(74) Agent: **NORTON ROSE FULBRIGHT CANADA S.E.N.C.R.L., S.R.L. / LLP**; 2500-1 Place Ville-Marie, Montréal, Québec H3B 1R1 (CA).

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

(54) Title: LOOP-FREE ROUTING IN COMPUTER NETWORKS AND THE INTERNET

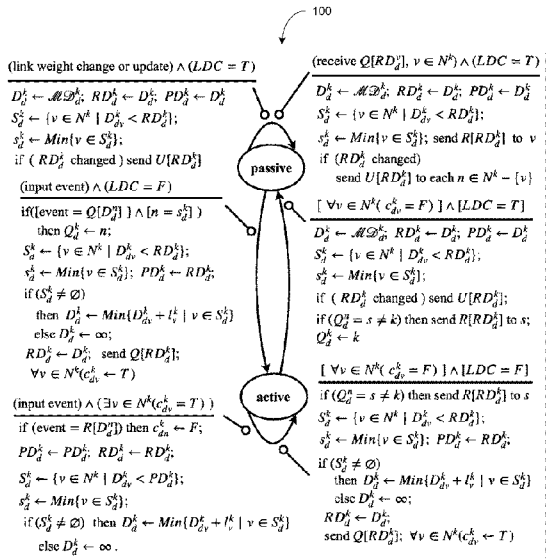

FIG. 1

(57) Abstract: There is provided a method of maintaining loop-free paths between source and destination nodes in a computer network. The method includes maintaining a list of neighbour nodes and associated link weights. The method further includes maintaining neighbor-to-destination distances associated with transmitting data from a neighbor to the destination. A set of successor nodes is selected based on having a neighbor-to- destination distance lower than the current distance from source to destination. The source node determines whether to enter an active state based on whether a local distance condition is satisfied. Upon entering the active state, a distributed computation to seek the shortest loop-free path from source to destination is initiated.

WO 2026/015977 A1

[Continued on next page]

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

Published:

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

— *in black and white; the international application as filed contained color or greyscale and is available for download from PATENTSCOPE*

LOOP-FREE ROUTING IN COMPUTER NETWORKS AND THE INTERNET

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] This claims priority to, and the benefit of, U.S. Provisional Patent Application No. 63/673,359, entitled "LOOP-FREE ROUTING IN COMPUTER NETWORKS AND THE INTERNET", filed July 19, 2024, the entire contents of which are incorporation herein by reference.

FIELD

[0002] This disclosure relates to the design of computer communication networks. More specifically, this disclosure relates to methods, systems, processes and devices for facilitating loop-free routing in a computer communication network.

BACKGROUND

[0003] In a computer network and/or the internet, data packets are routed between a source node and a destination node. The performance of a network may depend on the path selected for routing data packets from source to destination. The most common routing protocols used in today's internet are based on the exchange of vectors of distances, such as Routing Information Protocol (RIP), RIP version 2 (RIPv2), and Enhanced Interior Gateway Routing Protocol (EIGRP), topology maps, such as Open Shortest Path First (OSPF), or path information, such as the Border Gateway Protocol (BGP).

[0004] RIP, RIPv2 and a number of other similar routing protocols are based on the Distributed Bellman-Ford (DBF) algorithm for shortest-path computation, which suffers from the occurrence of undesirable temporary routing table loops, as well as the so-called "counting to infinity" problem, which limit their applicability to smaller scale computer networks that are required to use hop counts as the metric to measure distances to destinations.

[0005] Contrastingly, OSPF and other protocols based on topology broadcasting algorithms are limited by the excessive communication overhead incurred in replicating

the network topology at each routing node in the network, as well as the need to periodically refresh that information. This forces network administrators to partition the network into areas or clusters interconnected by a backbone area or cluster. As a result, OSPF and other topology-broadcasting-based approaches involve complex solutions, in terms of the required routing configurations.

[0006] BGP does not suffer from the counting-to-infinity problem because a given router can break routing loops when the router determines that a path reported by a neighbor router includes the router itself. However, BGP cannot prevent the creation of temporary routing loops and is known to take a relatively long time to converge to valid routes in some cases. This is due to the fact that the paths selected by routers to reach destinations are not ordered with respect to one another, which may cause routers to select paths that are subsequently found to have loops.

[0007] To address the above-noted limitations of DBF and topology broadcasting approaches, some routing algorithms, such as the Diffusing Update Algorithm (DUAL) may emphasize the determination of a shortest possible loop-free route. EIGRP uses DUAL, for example. DUAL may eliminate the routing-table loops that may occur using DBF, but may in some cases also incur considerable signaling overhead to accomplish this, and may prevent some routing nodes from having routes to destinations for some periods of time until such routing nodes attain consistent routing information. All routing algorithms used in computer networks and the internet at present support only single-path routing, meaning that a routing node maintains a single next hop to a destination. In some cases, a router is allowed to use more than one route to a destination as long as all routes have equal distance (or cost) values. Thus, current routing algorithms suffer from inherent limitations in various circumstances that may negatively impact efficiency and throughput.

[0008] It would be desirable to provide a routing algorithm that is scalable, efficient, quick to respond to network changes, capable of providing more than one route to each destination without requiring such routes to have equal cost, and which ensures temporary routing loops are not created.

SUMMARY

[0009] According to an aspect, there is provided a method of maintaining loop-free paths between a source node and a destination node in a computer network, the method comprising: maintaining, in a link weight table for said source node, a list of neighbor nodes directly connected to said source node, and a respective link weight associated with transmitting data from said source node to each of said respective neighbor nodes; maintaining, in a neighbor table for said source node, for one or more of said neighbor nodes, a respective neighbor-to-destination distance associated with transmitting data from said respective neighbor node to said destination node; storing, at said source node, a current source-to-destination distance associated with transmitting data from said source node to said destination node via a current loop-free path; generating, in a routing table of said source node, a set of successor nodes from among said neighbor nodes, said successor nodes being each of said neighbor nodes having a neighbor-to-destination distance that is lower than said current source-to-destination distance, wherein each of said successor nodes is a neighbor node along a loop-free path to said destination; determining, by said source node, whether to enter an active state based on whether a local distance condition has been satisfied; and upon entering said active state, initiating a distributed computation to seek a shortest loop-free path between the source node and the destination node.

[0010] According to another aspect, there is provided a non-transitory computer-readable medium having stored thereon computer-executable instructions that, when executed by one or more processors, cause the one or more processors to perform a method of maintaining loop-free paths between a source node and a destination node in a computer network, the method comprising: maintaining, in a link weight table for said source node, a list of neighbor nodes directly connected to said source node, and a respective link weight associated with transmitting data from said source node to each of said respective neighbor nodes; maintaining, in a neighbor table for said source node, for one or more of said neighbor nodes, a respective neighbor-to-destination distance associated with transmitting data from said respective neighbor node to said destination node; storing, at said source node, a current source-to-destination distance

associated with transmitting data from said source node to said destination node via a current loop-free path; generating, in a routing table of said source node, a set of successor nodes from among said neighbor nodes, said successor nodes being each of said neighbor nodes having a neighbor-to-destination distance that is lower than said current source-to-destination distance, wherein each of said successor nodes is a neighbor node along a loop-free path to said destination; determining, by said source node, whether to enter an active state based on whether a local distance condition has been satisfied; and upon entering said active state, initiating a distributed computation to seek a shortest loop-free path between the source node and the destination node.

[0011] According to still another aspect, there is provided a system for maintaining loop-free paths in a computer network, the system comprising: a source node; a destination node; said source node comprising a memory having stored thereon computer-executable instructions that, when executed by a processor of said source node, causes said source node to perform a method comprising: maintaining, in a link weight table for said source node, a list of neighbor nodes directly connected to said source node, and a respective link weight associated with transmitting data from said source node to each of said respective neighbor nodes; maintaining, in a neighbor table for said source node, for one or more of said neighbor nodes, a respective neighbor-to-destination distance associated with transmitting data from said respective neighbor node to said destination node; storing, at said source node, a current source-to-destination distance associated with transmitting data from said source node to said destination node via a current loop-free path; generating, in a routing table of said source node, a set of successor nodes from among said neighbor nodes, said successor nodes being each of said neighbor nodes having a neighbor-to-destination distance that is lower than said current source-to-destination distance, wherein each of said successor nodes is a neighbor node along a loop-free path to said destination; determining, by said source node, whether to enter an active state based on whether a local distance condition has been satisfied; and upon entering said active state, initiating a distributed computation to seek a shortest loop-free path between the source node and the destination node.

[0012] Other features will become apparent from the drawings in conjunction with the following description.

BRIEF DESCRIPTION OF DRAWINGS

[0013] In the figures which illustrate example embodiments,

[0014] FIG. 1 depicts operation of a state machine implementing a Diffusing Algorithm for Routing Efficiency (referred to herein as “DARE”);

[0015] FIG. 2 depicts operations performed on an example network in accordance with DARE in response to a link weight changing;

[0016] FIG. 3 depicts operations performed on an example network in accordance with DARE in response to a link between nodes failing;

[0017] FIG. 4 depicts an example network topology in which a link fails;

[0018] FIG. 5 is a block diagram depicting components of an example computing device, in accordance with some embodiments;

[0019] FIG. 6 is pseudocode describing example overall operation of a router or node incorporating DARE;

[0020] FIG. 7 is pseudocode describing example operations performed during initialization of a router or node;

[0021] FIG. 8 is pseudocode describing example operations relating to sending a “hello” message to neighboring nodes;

[0022] FIG. 9 is pseudocode describing example operations relating to processing a change in link weight;

[0023] FIG. 10 is pseudocode describing example operations relating to processing a neighbor node timeout event;

[0024] FIG. 11 is pseudocode describing example operations relating to processing a message received from a neighbor;

[0025] FIG. 12 is pseudocode describing example operations relating to sending an Update message;

[0026] FIG. 13 is pseudocode describing example operations relating to sending a Query message;

[0027] FIG. 14 is pseudocode describing example operations relating to sending an Update message when in the passive state;

[0028] FIG. 15 is pseudocode depicting example operations relating to sending an Update message when in the active state;

[0029] FIG. 16 is pseudocode depicting example operations relating to sending a Reply message;

[0030] FIG. 17 is pseudocode depicting example operations relating to sending a Query while in the passive state; and

[0031] FIG. 18 is pseudocode depicting example operations relating to sending a Query while in the active state.

DETAILED DESCRIPTION

[0032] Many of the routing protocols operating within and across systems in IP-based networks at present are based on only a few types of algorithms designed to ensure eventual convergence to the correct routes, and in some cases to acyclic routing at all times. Some of these routing algorithms may use destination sequence numbers to identify the most recent distance updates, inter-nodal coordination intended to avoid routing paths containing loops, or complete path and/or topology information to detect routing loops.

[0033] The two most commonly used approaches to avoiding routing loops are the use of destination sequence numbers, and inter-nodal coordination. Inter-nodal coordination tends to work more effectively in large computer networks, because inter-nodal coordination does not require periodic flooding of new sequence numbers by destinations. Among inter-nodal coordination techniques, the use of the DUAL routing algorithm has achieved widespread use, in view of it having been incorporated into the Enhanced Interior Gateway Routing Protocol (EIGRP) used by Cisco Systems, Inc.

[0034] However, DUAL has a number of inherent limitations. For example, DUAL provides only a single loop-free successor node (a so-called 'next hop') per destination at each node (e.g., a router). In some embodiments, after a network topology change, it may be possible to reduce the amount of signaling required to converge to new loop-free routes by incorporating multiple successor node options, as described herein.

[0035] Another inherent limitation of DUAL is that a node which has sent a query cannot use a different successor node until that node has received replies from all of its neighbour nodes. Thus, even if some of the query-sending node's neighbors have replied and offered valid loop-free routes, the query-sending node nevertheless must wait until all replies have been received, which may unnecessarily impede the throughput of data packets.

[0036] A further inherent limitation of DUAL is that DUAL uses a complex state machine to account for multiple diffusing computations, changes in link costs/weights, and updates from neighbour nodes that may occur concurrently – this complexity may limit the degree to which routing protocols incorporating DUAL may evolve.

[0037] Various routing protocols make use of the Distributed Bellman-Ford (DBF) algorithm for determining the shortest path between nodes. However, DBF has a disadvantage in that it may count to infinity (e.g., if link or node failures render a node unreachable from a set of other nodes, and those nodes may spend forever gradually increasing the estimates of the distance to the unreachable node). Moreover, routing protocols using the DBF algorithm may declare a destination to be unreasonable if a

redefined maximum distance value is reached, which may limit the types of routing metrics that can be used, and may lead to longer convergence delays.

[0038] Many routing algorithms have been developed to eliminate or reduce these weaknesses of the DBF algorithm. For example, the broadcasting of complete topology information combined with local use of the Dijkstra shortest-path first (SPF) algorithm at each router is a known alternative to DBF. Such algorithms may provide appealing convergence times on average, but do not prevent the possibility of temporary routing loops, which reduce efficiency. Other routing algorithms may incorporate partial topology information (e.g., the use of link states), distances and second-to-last hops to destinations, or complete path information in updates.

[0039] Still other shortest-path routing protocols (such as the Destination-Sequenced Distance-Vector (DSDV) routing protocol) use destination sequence numbers to attain loop freedom. Routing Protocol for Lower-Power and Lossy Networks (RPL) is an example of a recent protocol which makes use of destination sequence numbers. However, RPL is known to incur temporary routing loops under various conditions, such as router failures, volatile memory, and recycling of destination sequence numbers.

[0040] Several shortest-path routing approaches have been developed that provide loop-free routing, by requiring nodes to coordinate the updating of routing tables on a multi-hop basis, together with the use of sequence numbering.

[0041] The diffusing update algorithm (DUAL) currently used by EIGRP responds to changes in the routing topology and dynamically adjusts routing tables of a router automatically. DUAL uses a so-called ‘feasibility condition’ to ensure that only loop-free routes are ever selected for transmission. If there is no feasible route to a destination (i.e., no route which satisfies the feasibility condition), DUAL will invoke a diffusing computation to ensure that all traces of the problematic route are eliminated from the network, and the Bellman-Ford algorithm may then be used to determine a new route.

[0042] Diffusing computations use signaling messages. Messages may carry one or more updates, queries, and replies. The contents of signaling messages may include an identification of the router or node, an identification of the destination, and the distance to the destination.

[0043] According to some embodiments, there is provided herein an algorithm for maintaining loop-free paths (referred to herein as a Diffusing Algorithm for Routing Efficiency (DARE)) which may ameliorate some of the disadvantages of DUAL. In some embodiments, DARE may provide improved scalability by not requiring careful partitioning of the network topology into areas (as required with topology broadcast-based techniques), may improve efficiency by limiting the propagation of signaling message to routing nodes that need them, and may provide improved speed in responding to network changes by providing multiple choices for paths to reach destinations, as described below. Thus, in some embodiments, DARE may overcome deficiencies of previously developed routing algorithms. It is contemplated that in some embodiments, DARE may be suitable for use with several different routing protocols, including but not limited to EIGRP, BGP, and RIPv2.

[0044] In some embodiments, DARE may support distributed computations to seek a shortest path between a source node and a destination node, without ever creating a routing loop. In some embodiments, DARE may use routing messages containing updates, queries, and/or replies which include distances to destinations. In some embodiments, a router or node using DARE may re-determine or update its distance to a given destination independently of other nodes using the Bellman-Ford equation, provided a local condition is satisfied for the destination node. In some embodiments, a node using DARE may be referred to as being in a “passive” mode when the local condition is satisfied. Examples of local conditions are provided further below.

[0045] In some embodiments, when the local condition is not satisfied for a node using DARE, the node may enter an “active” state in which a distributed computation is initiated for the destination node by querying neighbor nodes and awaiting replies from

each neighbor node. In some embodiments, a distributed computation is a suitable algorithm carried out at a plurality of nodes to compute a value. For example, for routing in a computer network, a distributed computation may calculate a route from a routing node (e.g., a source node) to a destination node. Examples of distributed computations may include a diffusing computation, in which a routing node sends a query to its neighbor nodes and waits to receive replies from each of said neighbor nodes before determining whether the computation has been completed. Another example of a distributed computation may include flooding all nodes with a new sequence number to validate a locally stored value of a distance to a destination or a link weight.

[0046] DUAL defines a “feasible distance” value as the minimum distance value among all neighbour nodes for a destination. DUAL selects only a single node as a successor node (i.e. the next hop), and requires the successor node to have reported a distance which is smaller than the feasible distance, and which is the shortest distance available.

[0047] Contrastingly, DARE does not incorporate the concept of a feasible distance, and instead selects neighbor nodes for inclusion in a set of successor nodes, rather than selecting a single successor node for a given node. In some embodiments, DARE provides that in order to be added to the set of successor nodes for a given node, a neighbor node must have a reported distance to the destination node which is lower than the last distance to the destination node that the given node had while in a passive state.

[0048] In some embodiments, if there are no neighbor nodes which meet the requirements to be in the set of successor nodes, or if none of the successor nodes provide the shortest distance to the destination, then DARE causes the given node to enter an active state in which a distributed computation is initiated to seek a shortest loop-free path between the given node and the destination node. Moreover, when a neighbor node’s query causes a given node to become active in a distributed computation, DARE provides that the given node may store an identifier corresponding to the identity of the neighbor whose query forced the given node to become active.

[0049] As noted above, a significant difference between DUAL and DARE is that DARE generates multiple loop-free routes to a destination node for a given source node, and each of such loop-free routes is not required to have the same distance (or cost). Contrastingly, DUAL supports only single-path routing (i.e., only one node may be chosen as a next hop or preferred successor for a given source node).

[0050] A further key difference between DUAL and DARE is that when a given node enters an active state, DARE does not force the active node to continue using the same successor node for the duration of that the node is active. For example, using DARE, whether in an active or a passive state, a source node or router is free to use any neighbor node as a successor, provided the neighbor has reported a shorter distance to the destination node than the current source-to-destination distance of the source node the last time the source node was in a passive state. This may be particularly advantageous in that when a source node enters an active state using DARE and sends query messages to neighbor nodes, the active source node may still use a successor node in the interim, before reply messages to the query have been received from all of the active source node's neighbor nodes.

[0051] Thus, rather than the source node pausing transmission of data packets while waiting for replies from each neighbor (as would be the case using DUAL), an active source node may transmit data packets to a successor node, so long as that successor node has reported a shorter distance to the destination than the source-to-destination distance of the source node when the source node entered the active state.

[0052] In some embodiments, the successor node selected by the active source node using DARE need not have reported the minimum distance to the destination node – it may be sufficient for the successor node to have reported a distance to the destination node which is smaller than the source-to-destination distance of the source node. In this manner, the transmission of data packets may continue while waiting for reply messages from every neighbour using a loop-free route which is known to have a shorter distance, without necessarily being the shortest distance. This may allow for increased overall throughput by continuing to transmit data packets via a shorter route

sooner, rather than taking additional time to determine the shortest possible route before resuming transmission.

[0053] A further disadvantage of DUAL is that a node or router is only allowed to participate in a single diffusing computation at any given time, and the node is blocked from participating in or mixing multiple diffusing computations. DUAL uses a complex state machine which defines four different active states, which allow a node in the active state to remember whether it is a source or a relay in a diffusing computation, and whether a distance increase or a new query occurred while already in the active state. In addition, an active node using DUAL must wait until replies from all of its neighbor nodes have been received before that active node can send an update message reporting its new shortest distance, or to send a reply to its successor if it relayed a query. Thus, many operations in DUAL involve delays as an active node waits for replies to determine optimal paths, which may negatively impact the overall performance of the system.

[0054] Contrastingly, in some embodiments, DARE allows an active node to participate in multiple distributed computations which can merge with other computations at active nodes. This functionality is possible because DARE specifies that a given node stores the identity of the neighbor node whose query caused the given node to become active. Moreover, when entering the active state, a given node using DARE may store the source-to-destination distance from when it was last in a passive state, and may still answer query messages from other nodes with that distance value from when it was last passive. Once the active node has received reply messages from all of its neighbour nodes, the active node may then send an update message as needed, and may also send a reply to the neighbor node that prompted it to become active.

[0055] Various example embodiments depicting operation of DARE are described below, and make use of the following terminology:

Variable Name	Value

Router and destination of interest	k, d
Set of neighbors of k	N^k
Link from k to $n \in N^k$	(k,n)
Weight and lifetime of link (k,n)	l_n^k, LT_n^k
Link-weight table	LWT^k
Content of LWT^k for link (k,q)	l_q^k, LT_q^k
Distance reported by $v \in N^k$	D_{dv}^k
Path reported by $v \in N^k$ [if path information is used]	R_{dv}^k
Coordination flag assigned to $v \in N^k$	c_{dv}^k (value of T or F)
Content of neighbor table (NT^k) for $v \in N^k$	D_{dv}^k, c_{dv}^k
Content of neighbor table (NT^k) for $v \in N^k$ if path information is used	$D_{dv}^k, c_{dv}^k, R_{dv}^k$
Distance attained through $v \in N^k$	$D_{dv}^k + l_v^k$
Current distance for d	D_d^k
Current preferred route to d if path information is used	R_d^k
Passive and reported distances for d	PD_d^k, RD_d^k
Passive and reported routes for d if path information is used	PR_d^k, RR_d^k
Successor set for d	S_d^k
Preferred successor for d	s_d^k
Query coordination identifier d	Q_d^k
Active flag for d	A_d^k (value of T or F)
Content of routing table (RT^k) for d	$D_d^k, PD_d^k, RD_d^k, S_d^k, s_d^k, Q_d^k, A_d^k$
Content of routing table (RT^k) for d if path information is used	$D_d^k, PD_d^k, RD_d^k, S_d^k, s_d^k, Q_d^k, A_d^k, R_d^k, PR_d^k, RR_d^k$
Query (Q_k), reply (R_k) or update (U_k) for d	$[d, RD_d^k]$
Query (Q_k), reply (R_k) or update (U_k) for d if path information is used	$[d, RD_d^k, RR_d^k]$

[0056] It should be appreciated that although certain example embodiments described herein make reference to routers, principles described herein are applicable to other types of network nodes other than routers.

[0057] In some embodiments of DARE, each router *k* may maintain three tables: the Link Weight Table, the Neighbor Table, and the Routing Table.

[0058] In some embodiments, the Link Weight Table (LWT^k) lists the weight (l_q^k) and lifetime (LT_q^k) of the link (k, q) with each neighbor q. The maximum lifetime of a neighbor entry is a constant LT which may be defined for the network.

[0059] In some embodiments, the Neighbor Table (NT^k) has an entry for each $q \in N^k$. For each destination d, the entry for q lists D_{dq}^k and the value of a coordination flag c_{dq}^k. The value of c_{dq}^k is true (T) if router k requires a response from q and is false (F) otherwise. If a neighbor q has not reported any distance for d to router k, then router k sets $D_{dq}^k = \infty$ and $c_{dp}^k = F$.

[0060] In some embodiments, the Routing Table (RT^k) contains an entry for each destination d. The entry for d in RT^k states: A current-distance value (D_d^k), a passive-distance value (PD_d^k), a reported-distance value (RD_d^k), the set of neighbors (S_d^k) that can be used as successors, the preferred successor (s_d^k) in S_d^k, an *active flag* A_d^k, and a query-coordination identifier (Q_d^k).

[0061] In some embodiments, the value of A_d^k is true (T) if k is active, and is false (F) if *k* is passive. The value of Q_d^k is set to k if either k is passive, or if *k* is active and does not have to send a reply to a neighbor when it becomes passive. If router k becomes active or continues to be active with a query from its preferred successor s at the time it was passive, then Q_d^k is set to s.

[0062] In some embodiments, the distance for transmitting from k to the destination may be given by $D_d^k = D_{ds}^k + l_s^k$, if $s = s_d^k$ and $S_d^k \neq \emptyset$. If no neighbors meet the requirements to be successor nodes, then the successor set will be a null set, and the distance will be set to infinity ($D_d^k = \infty$ if $S_d^k = \emptyset$).

[0063] PD_d^k is the value of the distance from router k to destination d at the time router k was in a passive state and had to change to the active state. RD_d^k is the reported distance value for destination d that router k sends in a query message when it transitions from passive to active state, and equals $D_{ds}^k + l_s^k$, where $s \in N^k$ is the successor being used when router k transitions to the active state.

[0064] In some embodiments, DARE uses signaling to transmit messages between routers and nodes. In some embodiments, routers may inform their neighbors about changes made in their routing tables, or the need to coordinate, using routing messages that contain a vector of one or multiple entries. In some embodiments, each entry is an update, query, or reply for a given destination, and includes a distance to that destination. As described herein, an update, query or reply from router k for destination d is denoted by $U[d, RD_d^k]$, $Q[d, RD_d^k]$ and $R[d, RD_d^k]$, respectively.

[0065] In some embodiments, DARE may be initialized by allowing an initialization delay to elapse before router k is allowed to process messages. In some embodiments, the initialization delay is long enough to ensure that, in the event that router k is restarting after a failure, all neighbor routers will have had time to process previous routing messages from router k and also to have determined that it router *k* was not operational. A router k may be initialized in a passive state for itself and all destinations by setting $\sigma^k = T$; $D_k^k = PD_k^k = RD_k^k = 0$, $s_k^k = k$, $S_k^k = \{k\}$, and $c_k^k = F$. No entries are required for other destinations, and no entry implies infinite distances and no successors. Router k may then send a “hello" message, which is a routing message with an update for itself.

[0066] Once initialized, a router *k* may be in one of two states (active or passive) for a destination *d* based on a local distance condition (LDC). FIG. 1 is a depiction of a state machine 100 representing operation of DARE. In FIG. 1, the terminology *LDC* = *T* denotes that the local distance condition is true, and the terminology *LDC* = *F* denotes that the local distance condition is false. In some embodiments, the local distance condition is given by:

$$LDC: (\exists\, v \in N^k (D_{dv}^k < RD_d^k)) \wedge (\mathcal{MD}_d^k = \mathcal{OD}_d^k) \tag{1a}$$

$$\text{where} \quad \mathcal{MD}_d^k = Min\{D_{dv}^k + l_v^k \mid v \in N^k\};$$

$$\text{and} \quad \mathcal{OD}_d^k = Min\{D_{dv}^k + l_v^k \mid (v \in N^k) \wedge (D_{dv}^k < RD_d^k)\}$$

[0067] In some embodiments, the sharing of path information may be desirable. In some embodiments, the LDC may be defined so as to include path information:

$$LDC\colon (\exists v \in N^k(\ [\ k \notin RR_{dv}^k\ ] \wedge\ [D_{dv}^k < RD_d^k\ ]\ )) \wedge (MD_d^k = OD_d^k) \quad \text{(1b)}$$

$$\text{where } MD_d^k = Min\{D_{dv}^k + l_v^k | v \in N^k\};$$

$$\text{and } OD_d^k = Min\{D_{dv}^k + l_v^k | (v \in N^k) \wedge (D_{dv}^k < RD_d^k)\}$$

[0068] From LDC equations (1a) and (1b) above, it is clear that information is not strictly required for the correct operation of DARE systems. In alternative embodiments, when path information is shared among routers, the definition for the LDC may be modified so as to take path information into account, as in (1b). In embodiments of DARE in which path information is included, routers are able to establish an ordering between their preferred paths and the preferred paths reported by their neighbour routers. As such, although further embodiments described below might not incorporate path information, it should be appreciated that some embodiments of DARE may incorporate path information.

[0069] Router *k* will be in the passive state as long as the LDC is true. In the passive state, a router *k* using DARE may behave in accordance with the DBF algorithm as noted above.

[0070] In some embodiments, when a router *k* is in a passive state for destination *d*, router *k* updates its distances and successors as follows:

$$D_d^k \leftarrow \mathcal{MD}_d^k;\ RD_d^k \leftarrow D_d^k;\ PD_d^k \leftarrow D_d^k$$

$$S_d^k \leftarrow \{v \in N^k \mid D_{dv}^k < RD_d^k\};\ \ s_d^k \leftarrow Min\{v \in S_d^k\} \quad (2)$$

[0071] It can be seen that when router *k* is in a passive state for destination *d*, $D_d^k = RD_d^k = PD_d^k = \mathcal{MD}_d^k$. Router k may send an update $U[d, RD_d^k]$ if RD_d^k changes after an input event while LDC is satisfied. If the input event was a query from a neighbor node v, router k may send a reply $R[d, RD_d^k]$ to v.

[0072] If LDC is not satisfied when router *k* is passive, then router *k* will enter an active state for destination *d*, and will update its successors and distances in its routing table when transitioning to the active state:

$$S_d^k \leftarrow \{v \in N^k \mid D_{dv}^k < RD_d^k\}; s_d^k \leftarrow Min\{v \in S_d^k\};\ PD_d^k \leftarrow RD_d^k$$

$$\text{if}(S_d^k \neq \emptyset)\ \text{then}\ D_d^k \leftarrow Min\{D_{dv}^k + l_v^k \mid v \in S_d^k\}\ \text{else} D_d^k \leftarrow \infty$$

$$RD_d^k \leftarrow D_d^k. \quad (3)$$

[0073] As can be seen from above, router *k* stores as its passive distance for destination *d* the value of its distance before router *k* became active, and stores as its reported distance for destination *d* the values of its new distance after entering the active state. Router *k* may then select a neighbor *n* as its preferred successor provided that neighbor *n* has reported a distance to destination *d* which is smaller than RD_d^k. In some embodiments, the local distance value is not reported to neighbor nodes while router *k* is active, and is updated while router *k* is active. While active, router *k* may send a query $Q[d, RD_d^k]$ to its neighbors, and will remain active until router *k* has received replies from all of its neighbors. In some embodiments, if router k becomes active as a result of a query from its current preferred successor $s = s_d^k$ before the event, then it sets $Q_d^k = s$, otherwise it keeps $Q_d^k = k$.

[0074] It will be appreciated that the above contrasts with DUAL in that a router *k* using DUAL cannot change its successor when it becomes active, and while router *k* is active, and the active router *k* will use the minimum attained distance value of D_d^k since it was last passive to decide whether to become active.

[0075] In some embodiments, once a router *k* is active for a destination *d*, the router *k* may update its distance and successors as follows:

$$PD_d^k \leftarrow PD_d^k;\ RD_d^k \leftarrow RD_d^k$$

$$S_d^k \leftarrow \{v \in N^k \mid D_{dv}^k < PD_d^k\};\ s_d^k \leftarrow Min\{v \in S_d^k\}$$

$$\text{if}(S_d^k \neq \emptyset)\ \text{then}\ D_d^k \leftarrow Min\{D_{dv}^k + l_v^k \mid v \in S_d^k\}\ \text{else} D_d^k \leftarrow \infty. \quad (4)$$

[0076] As can be seen, when router *k* is in an active state, neither PD_d^k nor RD_d^k are modified. While active, D_d^k is updated to reflect the minimum distance through any neighbor n such that $D_{dn}^k < PD_d^k$. The preferred successor and the successor set for destination d are also updated to enforce the ordering between router k and its neighbors. If the input event is a query from neighbor *v* which is not the preferred successor ($v \neq s_d^k$) and $Q_d^k = k$, then k sends reply $R[d, RD_d^k]$ to v. If router k receives a query from preferred successor s_d^k once router *k* is active and $Q_d^k = k$, then router k sets $Q_d^k = s$ and does not send a reply to Q_d^k until router *k* has received all the replies for its own query.

[0077] Once an active router *k* has received replies from all of its neighbor nodes, router *k* determines whether the local distance condition is satisfied. If LDC is satisfied, then router *k* will return to the passive state and will update its successor set and distances in its routing table as follows:

$$S_d^k \leftarrow \{v \in N^k \mid D_{dv}^k < RD_d^k\};\ s_d^k \leftarrow Min\{v \in S_d^k\}$$

$$D_d^k \leftarrow \mathcal{MD}_d^k;\ RD_d^k \leftarrow D_d^k;\ PD_d^k \leftarrow D_d^k \quad (5)$$

[0078] If $Q_d^k = s \neq k$, then router k sends a reply $R[d, RD_d^k]$ to neighbor s, and also sends an update $U[d, RD_d^k]$ to the rest of its neighbors if RD_d^k had to be updated.

[0079] However, if LDC is not satisfied after active router k has received all the replies to its query, then router *k* will become active again. In this case, if $Q_d^k = s \neq k$, then router k will send reply $R[d, RD_d^k]$ to neighbor s, and then follows the steps in Equation 3 above (describing the routing table updates when transitioning to the active state) to re-enter the active state for destination *d*.

[0080] In some embodiments, router *k* may be configured to process a change in the properties of a link (*k, q*) between itself and a neighbour *q*. When router k receives a routing message from neighbor *q*, router *k* updates the lifetime of the entry for that neighbor (LT_q^k) and processes each update and probe included in the message.

[0081] Router k may update the link weight l_q^k after the link event. If the link has changed in weight, router k may update RT^k for destination d assuming that an update $U[d, RD_d^q = D_{dq}^k]$ is received from neighbor q with the updated values in NT^k and LT^k.

[0082] Link failures can be detected in multiple ways, including by keeping track of data packets not being acknowledged by a neighbor router. In some embodiments, DARE may handle link failures using the lifetimes of neighbor entries to determine when links to neighbor routers or the neighbor routers themselves have failed. For example, router k may assume that a neighbor router q from which no signaling has been received for an entire duration LT can be declared to have failed. If a link (k, q) has failed, then the link weight table (LWT) will be updated to reflect $l_q^k \leftarrow l_\infty$, and for each destination d, router k sets $c_{dq}^k \leftarrow F$ and $D_{dq}^k \leftarrow \infty$. If the link is established and router k is active for destination d, router k sets $c_{dq}^k \leftarrow T$ and sends query $Q[d, RD_d^k]$ to q.

[0083] In some embodiments, the local distance condition (LDC) specified by Equation (1) above being satisfied may ensure that routing paths determined in accordance with DARE are loop-free. Thus, the signaling used for DARE may ensure loop-free routing. The following proof is presented to show that using *LDC* = T when routers choose their successors is sufficient to ensure that no routing-table loops are formed, even when routers share only distances to destinations.

[0084] *Proof.* Assume that a routing-table loop L exists at time t and let $L = \{v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_h \rightarrow v_{h+1}\}$, where $v_{h+1} = v_1$. For the sake of contradiction, assume that LDC is satisfied for every router in L when at least one router in L changes its successor when it joins the loop. Each router $v_i \in L$ informs its neighbors of its distance to d at a time denoted by t_i, where $t_i < t$, and its neighbors in L use that value at a subsequent time prior to time t to determine whether LDC is satisfied. The successor for destination d at router v_i is denoted by $n_d^{v_i}$, and the time when router $v_i \in L$ makes router $v_{i+1} \in L$ a next hop to destination d is denoted by t_i^+ and $t_i^+ \leq t$. This implies that

$$\forall v_i \in L([n_d^{v_i}(t) = n_d^{v_i}(t_i^+)] \wedge [D_d^{v_i}(t) = D_d^{v_i}(t_i^+)]) \quad (6)$$

[0085] Given that LDC is satisfied at every instant at every router, Equation (6) implies that: $\forall v_i \in L(D_d^{v_i}(t_i^+) = D_d^{v_i}(t) > D_{dv_{i+1}}^{v_i}(t) = D_{dv_{i+1}}^{v_i}(t_i^+)$

$$= D_d^{v_{i+1}}(t_{i+1}) = D_d^{v_{i+1}}(t)\ ) \quad (7)$$

[0086] Equations (6) and (7) are true only if $\forall v_i \in L(D_d^{v_i}(t) > D_d^{v_{i+1}}(t))$. (8)

[0087] However, Equation (8) is a contradiction, because it implies that $\forall v_i \in L(D_d^{v_i}(t) > D_d^{v_i}(t))$. Therefore, it has been shown that using LDC = T when routers choose their successors ensures no routing-table loops are formed.

[0088] It can also be shown from the above-described proof that DARE ensures loop-free paths at every instant.

[0089] *Proof.* It follows from the above-noted proof that no path can be a loop if every router in the path uses LDC to select its successors, which according to Equation (2) is the case when all routers along a path are in the passive state. Hence, a path cannot be a loop if it involves only passive routers.

[0090] According to Equations (3) and (4) above, when a passive router k becomes active, and while that router *k* is active, router *k* can only use as successors neighbor nodes that reported distances smaller than PD_d^k, which equals the value of D_d^k just before router *k* became active. Hence, it is also true that no path can be a loop if it includes active routers. Furthermore, no loop can include a router that has no successors. Thus, the remainder of the proof is required to show that no routing-table loop can be created when an active router transitions to the passive state with a finite distance once it processes the replies from all of its neighbors.

[0091] When a router k transitions from active to passive state, according to Equation (5), router k can only use a neighbor $v \in N^k$ as a successor if $D_{dv}^k < RD_d^k$. Furthermore, a router r can send a reply to its preferred successor only if router *r* is in the passive state. Accordingly, if a neighbor router $n \in N^k$ reports a distance RD_d^n corresponding to a route to destination d that includes subpath P_{nk} to router k, it follows

that $RD_d^{v_i} > RD_{dn_i}^{v_i}$ for all $v_i \in P_{nk}$, where $n_i = s_d^{v_i}$. Given that all link weights are positive, it must then be true that $RD_d^n = D_{dn}^k > RD_d^k$, and hence router k cannot choose neighbor router p as a successor when it applies Equation (5). This implies that a router that transitions from the active state to the passive state cannot create a loop by selecting successors according to Equation (5). Therefore, it has been shown that DARE ensures loop-free paths at every instant.

[0092] In some embodiments, use of DARE may offer performance advantages over existing techniques, such as DUAL. The following examples provide simplified illustrations of some of the benefits associated with DARE relative to DUAL.

[0093] FIG. 2 depicts the simplified operation of DARE through an example network containing 5 nodes (depicted as nodes *a, b, c, d* and *e*). In this simplified example, the weights of each link between nodes is assumed to be equal in both directions (e.g. the weight for (*a*, *b*) is equal to the weight for (*b*, *a*)).

[0094] For each router v in the network, the distance (D_d^v), passive distance (PD_d^v), and reported distance (RD_d^v) are indicated as ordered triplets in parentheses next to the router. For example, for network 200a, router *b* has an ordered triplet of (4, 4, 4). As depicted, arrowheads on links between nodes indicate whether a given node is a successor for destination d. In contrast to DUAL, DARE can provide multiple successors for destinations. Updates, Queries and Replies from router a are stated as $U[RD_d^a]$, $Q[RD_d^a]$, and $R[RD_d^a]$, respectively, with the destination being d and the nodes that should process an update or probe indicated by dashed arrowheads.

[0095] As depicted in FIG. 2, passive nodes are indicated with a solid-color node and active nodes are indicated in by nodes without a fill color. For simplicity, it is assumed in this example that nodes operate synchronously, such that nodes process all input events received in a given step, and send the resulting messages to neighbor nodes after processing all the inputs.

[0096] As depicted in FIG. 2, block 200a shows the state of network routing tables when the weight of link (a, d) increases to 10. In response to this change, at

block 200b, router a sends query $Q[10]$ because none of router *a*'s neighbors satisfy the LDC condition. Block 200c depicts routers b, d and e remaining passive and sending responses stating the values of their current distances, and router c becomes active and forwards the query to all its neighbors. As depicted in block 200d, router b must also forward the query after receiving the query from c, which is its preferred successor, because LDC cannot be satisfied.

[0097] Blocks 200e to 200j show that all active routers (a, b, and c) receive all the needed replies from their neighbors, and all routers become passive and converge to their shortest distances. Block 200k depicts the final state of the network. In this example, it is shown that DARE provides multiple next hops at some of the routers (whereas DUAL would not provide this flexibility).

[0098] FIG. 3 depicts the simplified operation of DARE on a further example network. It will be appreciated that although the topology of the network appears similar to the network depicted in FIG. 2, various link weights in the example network of FIG. 3 are different. FIG. 3 depicts an example scenario in which link (*a*, *d*) fails and no other link changes occur.

[0099] It should be noted that in this example scenario, if DUAL were used, router a would be required to send a query reporting an infinite distance because router *a* uses router d as its successor in its route to d as a destination. Moreover, the diffusing computation initiated by router a would also block routers c and b, because routers *c* and *b* would become active and would have to report an infinite distance in their respective queries.

[00100] In contrast to the operation of DUAL, when link (*a, d*) fails, DARE allows router a to select router e as a successor for destination d when router *a* has to become active. This occurs because router e has reported a distance that is shorter than the passive distance at router a (PD_d^a, which equals D_d^a before the link event). Thus, even though routers a, c, and b become active in a distributed computation, the three routers *a*, *c*, and *b* are nevertheless able to forward data packets to destination d while they are active. In addition, in this example scenario, all routers *a*, *b*, *c*, *d*, *e* converge to the

shortest paths faster than would occur with DUAL (which would require three additional steps relative to DARE, because using DARE obviates the need for router a to send an update reporting its new finite distance).

[00101] In addition, using DARE would result in faster convergence relative to DUAL if the link (a, d) were to increase in weight (e.g., from 3 to 20) rather than fail. With DUAL, router a would have to send a query reporting a larger distance (e.g., 20), and then would have to send an update reporting its new distance of 7. Contrastingly, DARE would just send a query reporting a distance of 7, as in the previous example.

[00102] FIG. 4 depicts a network topology scenario which illustrates further performance advantages which DARE may provide relative to DUAL , particularly due to the shortest paths in DARE not needing correspond to the minimum-hop paths. As depicted, all links shown in the network of FIG. 4 with solid lines are assumed to have a weight of 1, and any link shown in dashed lines has a weight that may be much larger than 1. The weight of link (a, d) (depicted as a weight of 3) is larger than the weight of link (b, d) (depicted as a weight of 2), and the weight (w_{ab}) of link (a, b) is such that, as a minimum, $w_{ab} + D_d^b > D_d^{v_2} + 1$.

[00103] If link (a, d) fails, DUAL would require router a to set its distance to infinity and initiate a diffusing computation with a query reporting an infinite distance to destination d. Depending on the weights of the links in the network, this query may propagate all way to router b following the reverse direction of the successor entries of routing tables, and all routers propagating the query would set their distances to infinity as well under a DUAL system. Routers upstream of router a would become passive as they receive all the replies they need, and a wave of updates started by router a would occur once router a becomes passive.

[00104] In contrast to a DUAL implementation, DARE would allow router a to send a query reporting a finite distance equal to $RD_d^a = 2 + w_{ab}$, because DARE allows router *a* to become active with a new successor (in accordance with Equation (3) described above). Using DARE, the ensuing query as router *a* becomes active may propagate

from router a all the way to router b. However, each active router sends replies to its neighbors other than the first neighbor that sent the query causing the router to become active (in accordance with Equation (4)). Hence, routers are likely to attain their shortest distances while they are still active and a last round of updates is unlikely. Furthermore, while routers executing DARE converge, no router data packets would be blocked and data packets would continue to flow to destination d.

[00105] Various embodiments described herein make use of computing devices to perform various operations. FIG. 5 is a block diagram depicting components of an example computing device 502. As depicted, computing device 502 includes processor 514, memory 516, persistent storage 518, network interface 520. It will be appreciated by a person skilled in the art that a computing device 502 may include additional or fewer components than those depicted in FIG. 5, and that the example computing device 502 is merely an example.

[00106] Processor 514 may be an Intel or AMD x86 or x64, PowerPC, ARM processor, microcontroller, or the like. Processor 514 may operate under control of processor-executable software instructions stored in storage 518, and executed by processor 514 in memory 516. Memory 516 may be any suitable type of random access memory accessible by processor 514. Storage 518 may be, for example, one or more modules of memory, hard drive, or other persistent computer storage devices. Software executing on computing device 502 may include, for example, an operating system and application software configured to implement and perform various operations associated with DARE as described herein.

[00107] Network interface 520 is capable of connecting computing devices 502 to one or more communication network and/or to one or more computing devices 502. In some embodiments, network interface 520 may include one or more wired interfaces (e.g. wired ethernet) and/or wireless radios, such as WiFi, Bluetooth, or cellular networks (e.g., GPRS, GSM, EDGE, CDME, LTE, 5G, or the like). Network interface 520 may also be used to establish virtual network interfaces, such as Virtual Private Networks (VPNs).

[00108] Various embodiments described herein may be performed by routers, network nodes, and other computing devices 502 under control of computer-executable instructions. It will be appreciated that the computer-executable instructions may take many different forms and be implemented in many different possible programming languages.

[00109] For ease of reference, the operation of various example embodiments may be described through the use of pseudocode. As with any such description, many alternative formulations are contemplated which may achieve similar functionality to the pseudocode examples provided herein.

[00110] FIG. 6 is pseudocode describing example overall operation of a router or node incorporating DARE.

[00111] FIG. 7 is pseudocode describing example operations performed during initialization of a router or node.

[00112] FIG. 8 is pseudocode describing example operations relating to sending a “hello” message to neighboring nodes.

[00113] FIG. 9 is pseudocode describing example operations relating to processing a change in link weight.

[00114] FIG. 10 is pseudocode describing example operations relating to processing a neighbor node timeout event.

[00115] FIG. 11 is pseudocode describing example operations relating to processing a message received from a neighbor.

[00116] FIG. 12 is pseudocode describing example operations relating to sending an Update message.

[00117] FIG. 13 is pseudocode describing example operations relating to sending a Query message.

[00118] FIG. 14 is pseudocode describing example operations relating to sending an Update message when in the passive state.

[00119] FIG. 15 is pseudocode depicting example operations relating to sending an Update message when in the active state.

[00120] FIG. 16 is pseudocode depicting example operations relating to sending a Reply message.

[00121] FIG. 17 is pseudocode depicting example operations relating to sending a Query while in the passive state.

[00122] FIG. 18 is pseudocode depicting example operations relating to sending a Query while in the active state.

[00123] Of course, the above-described embodiments are intended to be illustrative only and in no way limiting. The described embodiments are susceptible to many modifications of form, arrangement of parts, details and order of operation. Moreover, combinations of various embodiments are contemplated and within the scope of the invention. The invention is intended to encompass all such modification within its scope, as defined by the claims.

[00124] References

[00125] [1] B. Albrightson, et al., “EIGRP – A Fast Routing Protocol based on Distance Vectors," *Proc. Interop ‘94*, 1994.

[00126] [2] A. Arora and H. Zhang, “LSRP: Local Stabilization in Shortest Path Routing," *IEEE/ACM Trans.on Networking*, June 2006.

[00127] [3] J. Behrens and J.J. Garcia-Luna-Aceves, “Hierarchical Routing Using Link Vectors,” *Proc. IEEE INFOCOM ’98*, 1998.

[00128] [4] K. Bhargavan et al., “Formal Verification of Standards for Distance Vector Routing Protocols," *Journal of the ACM*, July 2002.

[00129] [5] C. Cheng, R. Riley, P. Kumar, and J.J. Garcia-Luna-Aceves, “A Loop-Free Extended Bellman-Ford Routing Protocol without Bouncing Effect," *Proc. ACM SIGCOMM ‘89*, Aug. 1989.

[00130] [6] J. Chroboczek and D. Schinazi, “The Babel Routing Protocol," *8966*, IETF, Jan. 2021.

[00131] [7] T. Clausen, U. Herberg, and M. Philipp, “A Critical Evaluation of the IPv6 Routing Protocol for Low Power and Lossy Networks (RPL),” *Proc. IEEE WiMob ‘11*, Oct. 2011.

[00132] [8] T. Clausen, J. Yi, A.C. de Verdiere, “LOADng: Towards AODV Version 2," *Proc. IEEE Vehicular Technology Conference ‘12*, Sept. 2012.

[00133] [9] T. Clausen, J. Yi , and Ulrich Herber, “Lightweight On-demand Ad hoc Distance-vector Routing - Next Generation (LOADng): Protocol, Extension, and Applicability," *Computer Networks*, 2017.

[00134] [10] J.A. Cobb, “Convergent Multi-Path Routing," *Proc. IEEE ICNP 2000*, Nov. 2000.

[00135] [11] J.A. Cobb and M.G. Gouda, “Stabilization of General Loop-Free Routing," *Journal of Parallel and Distributed Computing*, 2002.

[00136] [12] G. D’Angelo et al., “Engineering a New Loop-Free Shortest Path Routing Algorithm," *Proc. SEA ‘12* , Sep. 2012.

[00137] [13] E. Gafni and D. Bertsekas, “Distributed Algorithms for Generating Loop-Free Routes in Networks with Frequently Changing Topologies," *IEEE Trans. Communications*, Jan. 1981.

[00138] [14] J.J. Garcia-Luna-Aceves, “A Unified Approach to Loop-Free Routing Using Distance Vectors or Link States," *Proc. ACM SIGCOMM ’89* September 1989.

[00139] [15] J.J. Garcia-Luna-Aceves, “Loop-Free Routing Using Diffusing Computations," *IEEE/ACM Trans. Networking*, 1993.

[00140] [16] J.J. Garcia-Luna-Aceves and J. Behrens, “Distributed, Scalable Routing based on Vectors of Link States," *IEEE JSAC*, Oct. 1995.

[00141] [17] J. J. Garcia-Luna-Aceves and M. Spohn, “Scalable Link-State Internet Routing," *Proc. IEEE ICNP ‘98*, Oct. 1998.

[00142] [18] C.L. Hedrick, “Routing Information Protocol" RFC 1058, 1988.

[00143] [19] P.A. Humblet, “Another Adaptive Shortest-Path Algorithm," *IEEE Trans. Communications*, June 1991.

[00144] [20] International Organization for Standardization, International Standard ISO/IEC 10589:2002(E), 2002.

[00145] [21] J.M. Jaffe and F.M. Moss, “A Responsive Routing Algorithm for Computer Networks,” *IEEE Trans. on Communications*, July 1982.

[00146] [22] D. B. Johnson and D. A. Maltz, “Dynamic Source Routing in Ad Hoc Wireless Networks," *Mobile Computing*, Chapter 5, Kluwer Academic Publishers, 1996.

[00147] [23] D. Johnson, N. Ntlatlapa, and C. Aichele, “A Simple Pragmatic Approach to Mesh Routing Using BATMAN," *Proc. IFIP Int’l Symposium on Wireless Communications and Information Technology in Developing Countries*, Oct. 2008.

[00148] [24] G. Malkin, “RIP Version 2" RFC 2453, 1998.

[00149] [25] J. M. McQuillan, I. Richer, and E. C. Rosen, “The New Routing Algorithm for the ARPANET," *IEEE Trans. Communications*, 1980.

[00150] [26] P.M. Merlin and A.Segall, “A Failsafe Distributed Routing Protocol,” *IEEE Trans. on Communications*, Sept. 1979.

[00151] [27] J. Moy, “OSPF Version 2" RFC 2328, 1998.

[00152] [28] V. Park and M.S. Corson, “A Highly Adaptive Distributed Routing Algorithm, *Proc. IEEE INFOCOM ‘97*, April 1997.

[00153] [29] C. E. Perkins and P. Bhagwat, “Routing over Multihop Wireless Network of Mobile Computers," *Proc. ACM SIGCOMM ‘94*, 1994.

[00154] [30] J. Raju and J.J. Garcia-Luna-Aceves, “A New Approach to On-Demand Loop-Free Multipath Routing," *Proc. IEEE ICCCN ‘99*, Oct. 1999.

[00155] [31] S. Ray, R. Guerin, K.-W. Kwong, and R. Sofia, “Always Acyclic Distributed Path Computation," *IEEE/ACM Trans. on Networking*, 2010.

[00156] [32] Y. Rekhter and T. Li, “A Border Gateway Protocol 4 (BGP-4),” RFC 1771, March 1995.

[00157] [33] D. Savage et al., “Cisco’s Enhanced Interior Gateway Routing Protocol (EIGRP)" RFC 7868, 2016.

[00158] [34] M. Schwartz and T. Stern, “Routing Techniques Used in Computer Communication Networks," *IEEE Trans. Communications*, Apr. 1980.

[00159] [35] A. Segall, “Distributed Network Protocols,” *IEEE Trans. on Information Theory*, Jan. 1983.

[00160] [36] R. Van Glabbeek et al., “Sequence Numbers Do Not Guarantee Loop Freedom–AODV Can Yield Routing Loops," *Proc. ACM MSWiM ‘13*, Nov. 2013.

[00161] [37] S. Vutukury and J.J. Garcia-Luna-Aceves, “A Simple Approximation to Minimum-Delay Routing,” *Proc. ACM SIGCOMM ‘99*, Aug. 1999.

[00162] [38] S. Vutukury and J.J. Garcia-Luna-Aceves, “MDVA: A Distance-Vector Multipath Routing Protocol,” *Proc. IEEE INFOCOM ‘01*, April 2001.

[00163] [39] T. Winter, et al., “RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks," RFC 6550, IETF, March 2012.

[00164] [40] W.T. Zaumen and J.J. Garcia-Luna-Aceves, “Dynamics of Distributed Shortest-Path Routing Algorithms," *Proc. ACM SIGCOMM ‘91*, Sept. 1991.

[00165] [41] W.T. Zaumen and J.J. Garcia-Luna-Aceves, “Loop-Free Multipath Routing Using Generalized Diffusing Computations," *Proc. IEEE INFOCOM ‘98*, March 29–April 2, 1998.

[00166] [42] W.T. Zaumen and J.J. Garcia-Luna-Aceves, “System for Maintaining Multiple Loop-Free Paths between Source Node and Destination Node in Computer Network," U.S. Patent 5,881,243, March 9, 1999.

WHAT IS CLAIMED IS:

1. A method of maintaining loop-free paths between a source node and a destination node in a computer network, the method comprising:

maintaining, in a link weight table for said source node, a list of neighbor nodes directly connected to said source node, and a respective link weight associated with transmitting data from said source node to each of said respective neighbor nodes;

maintaining, in a neighbor table for said source node, for one or more of said neighbor nodes, a respective neighbor-to-destination distance associated with transmitting data from said respective neighbor node to said destination node;

storing, at said source node, a current source-to-destination distance associated with transmitting data from said source node to said destination node via a current loop-free path;

generating, in a routing table of said source node, a set of successor nodes from among said neighbor nodes, said successor nodes being each of said neighbor nodes having a neighbor-to-destination distance that is lower than said current source-to-destination distance, wherein each of said successor nodes is a neighbor node along a loop-free path to said destination;

determining, by said source node, whether to enter an active state based on whether a local distance condition has been satisfied; and

upon entering said active state, initiating a distributed computation to seek a shortest loop-free path between the source node and the destination node.

2. The method of claim 1, wherein said local distance condition is based on a comparison of said current source-to-destination distance with said neighbor-to-destination distances for each neighbor node in said neighbor table.

3. The method of claim 1, further comprising selecting a loop-free path to said destination node comprising one of said successor nodes.

4. The method of claim 3, wherein said one of said successor nodes in said selected loop-free path to said destination node has a higher neighbor-to-destination distance than at least one other of said successor nodes.

5. The method of claim 3, wherein said one of said successor nodes in said selected loop-free path to said destination node has a lowest neighbor-to-destination distance among said successor nodes.

6. The method of claim 1, wherein said successor set is a null set when said reported distance is lower than or equal to said neighbor-to-destination distance for each of said neighbor nodes.

7. The method of claim 1, further comprising receiving an input event at said source node.

8. The method of claim 1, wherein said distributed computation comprises at least one diffusing computation, said at least one diffusing computation comprising sending a query to said neighbor nodes and determining, based on replies received from said neighbor nodes, whether said source node can complete said diffusing computation.

9. The method of claim 1, wherein said source node is configured to participate in multiple distributed computations simultaneously.

10. The method of claim 9, wherein participating in said multiple distributed computations comprises:

sending a reply to a query received from a neighbor node in said neighbor set other than a current preferred successor node having a lowest neighbor-to-destination distance among said successor nodes; and

storing an identifier of said current preferred successor node in response to said source node receiving a query from said successor node.

11. The method of claim 1, further comprising setting an activation flag at said source node when said source node initiates said distributed computation.

12. The method of claim 1, further comprising, after entering said active state:

storing, as a passive distance for said source node, said current source-to-destination distance from prior to entering said active state;

selecting a set of successor nodes from among said neighbor nodes, said successor nodes comprising each of said neighbor nodes having a neighbor-to-destination distance that is lower than said passive distance, wherein each of said successor nodes is a neighbor node along a loop-free path to said destination node;

selecting a successor node from said successor set as a loop-free path to said destination, said successor node having a distance less than said passive distance, and

transmitting, by said source node, a query to said neighbor nodes.

13. The method of claim 12, wherein said query includes a reported distance for said source node, said reported distance comprising said current source-to-destination distance after entering said active state.

14. The method of claim 12, further comprising said source node remaining in said active state until replies to said query have been received from all of said neighbor nodes.

15. The method of claim 14, further comprising said source node entering a passive state upon determining that said local distance condition has been satisfied.

16. The method of claim 7, wherein said input event is one of an update, a reply, and a query from one of said neighbor nodes.

17. The method of claim 7, wherein said input event represents a failed link of a neighbor node, and said method further comprises said source node responding to said input event by:

deleting a state of said neighbor associated with said failed link; and

sending one of an update and a query based on whether said local distance condition is satisfied after said deleting said state of said neighbor associated with said failed link.

18. The method of claim 7, wherein said input event represents an establishment of a new link to a neighbor node, and said method further comprises said source node responding to said input event by:

adding a state for said neighbor node associated with said new link; and

sending an update to said neighbour node for each destination for which said source node has stored a finite current source-to-destination distance.

19. A non-transitory computer-readable medium having stored thereon computer-executable instructions that, when executed by one or more processors, cause the one or more processors to perform a method of maintaining loop-free paths between a source node and a destination node in a computer network, the method comprising:

maintaining, in a link weight table for said source node, a list of neighbor nodes directly connected to said source node, and a respective link weight associated with transmitting data from said source node to each of said respective neighbor nodes;

maintaining, in a neighbor table for said source node, for one or more of said neighbor nodes, a respective neighbor-to-destination distance associated with transmitting data from said respective neighbor node to said destination node;

storing, at said source node, a current source-to-destination distance associated with transmitting data from said source node to said destination node via a current loop-free path;

generating, in a routing table of said source node, a set of successor nodes from among said neighbor nodes, said successor nodes being each of said neighbor nodes having a neighbor-to-destination distance that is lower than said current source-to-destination distance, wherein each of said successor nodes is a neighbor node along a loop-free path to said destination;

determining, by said source node, whether to enter an active state based on whether a local distance condition has been satisfied; and

upon entering said active state, initiating a distributed computation to seek a shortest loop-free path between the source node and the destination node.

20. A system for maintaining loop-free paths in a computer network, the system comprising:

a source node;

a destination node;

said source node comprising a memory having stored thereon computer-executable instructions that, when executed by a processor of said source node, causes said source node to perform a method comprising:

maintaining, in a link weight table for said source node, a list of neighbor nodes directly connected to said source node, and a respective link weight associated with transmitting data from said source node to each of said respective neighbor nodes;

maintaining, in a neighbor table for said source node, for one or more of said neighbor nodes, a respective neighbor-to-destination distance associated with transmitting data from said respective neighbor node to said destination node;

storing, at said source node, a current source-to-destination distance associated with transmitting data from said source node to said destination node via a current loop-free path;

generating, in a routing table of said source node, a set of successor nodes from among said neighbor nodes, said successor nodes being each of said neighbor nodes having a neighbor-to-destination distance that is lower than said current source-to-destination distance, wherein each of said successor nodes is a neighbor node along a loop-free path to said destination;

determining, by said source node, whether to enter an active state based on whether a local distance condition has been satisfied; and

upon entering said active state, initiating a distributed computation to seek a shortest loop-free path between the source node and the destination node.

21. The method of claim 1, wherein said neighbor table for said source node further comprises a respective neighbor-to-destination path associated with transmitting data from said respective neighbor node to said destination node, and wherein said local distance condition is based on a comparison of said current source-to-destination distance with said neighbor-to-destination distances for each neighbor node having a respective neighbor-to-destination path which does not include said source node.

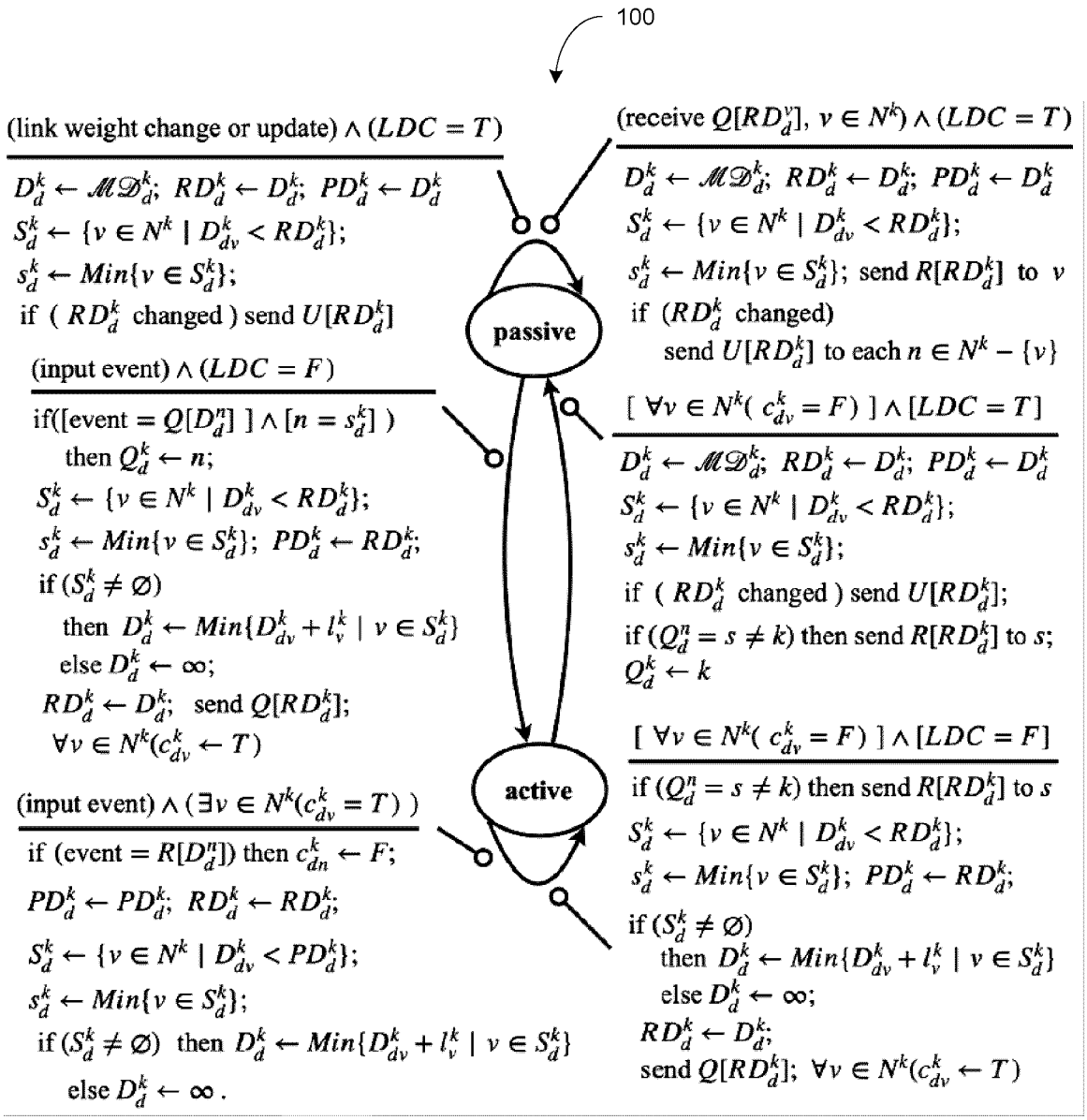
100
(link weight change or update) ∧ (LDC = T)
$D_d^k \leftarrow \mathcal{MD}_d^k$; $RD_d^k \leftarrow D_d^k$; $PD_d^k \leftarrow D_d^k$
$S_d^k \leftarrow \{v \in N^k \mid D_{dv}^k < RD_d^k\}$;
$s_d^k \leftarrow Min\{v \in S_d^k\}$;
if (RD_d^k changed) send $U[RD_d^k]$
(input event) ∧ (LDC = F)
if([event = $Q[D_d^n]$] ∧ [$n = s_d^k$])
then $Q_d^k \leftarrow n$;
$S_d^k \leftarrow \{v \in N^k \mid D_{dv}^k < RD_d^k\}$;
$s_d^k \leftarrow Min\{v \in S_d^k\}$; $PD_d^k \leftarrow RD_d^k$;
if ($S_d^k \neq \emptyset$)
then $D_d^k \leftarrow Min\{D_{dv}^k + l_v^k \mid v \in S_d^k\}$
else $D_d^k \leftarrow \infty$;
$RD_d^k \leftarrow D_d^k$; send $Q[RD_d^k]$;
$\forall v \in N^k(c_{dv}^k \leftarrow T)$
(input event) ∧ ($\exists v \in N^k(c_{dv}^k = T)$)
if (event = $R[D_d^n]$) then $c_{dn}^k \leftarrow F$;
$PD_d^k \leftarrow PD_d^k$; $RD_d^k \leftarrow RD_d^k$;
$S_d^k \leftarrow \{v \in N^k \mid D_{dv}^k < PD_d^k\}$;
$s_d^k \leftarrow Min\{v \in S_d^k\}$;
if ($S_d^k \neq \emptyset$) then $D_d^k \leftarrow Min\{D_{dv}^k + l_v^k \mid v \in S_d^k\}$
else $D_d^k \leftarrow \infty$.
passive
active
(receive $Q[RD_d^v]$, $v \in N^k$) ∧ (LDC = T)
$D_d^k \leftarrow \mathcal{MD}_d^k$; $RD_d^k \leftarrow D_d^k$; $PD_d^k \leftarrow D_d^k$
$S_d^k \leftarrow \{v \in N^k \mid D_{dv}^k < RD_d^k\}$;
$s_d^k \leftarrow Min\{v \in S_d^k\}$; send $R[RD_d^k]$ to v
if (RD_d^k changed)
send $U[RD_d^k]$ to each $n \in N^k - \{v\}$
[$\forall v \in N^k(c_{dv}^k = F)$] ∧ [LDC = T]
$D_d^k \leftarrow \mathcal{MD}_d^k$; $RD_d^k \leftarrow D_d^k$; $PD_d^k \leftarrow D_d^k$
$S_d^k \leftarrow \{v \in N^k \mid D_{dv}^k < RD_d^k\}$;
$s_d^k \leftarrow Min\{v \in S_d^k\}$;
if (RD_d^k changed) send $U[RD_d^k]$;
if ($Q_d^n = s \neq k$) then send $R[RD_d^k]$ to s;
$Q_d^k \leftarrow k$
[$\forall v \in N^k(c_{dv}^k = F)$] ∧ [LDC = F]
if ($Q_d^n = s \neq k$) then send $R[RD_d^k]$ to s
$S_d^k \leftarrow \{v \in N^k \mid D_{dv}^k < RD_d^k\}$;
$s_d^k \leftarrow Min\{v \in S_d^k\}$; $PD_d^k \leftarrow RD_d^k$;
if ($S_d^k \neq \emptyset$)
then $D_d^k \leftarrow Min\{D_{dv}^k + l_v^k \mid v \in S_d^k\}$
else $D_d^k \leftarrow \infty$;
$RD_d^k \leftarrow D_d^k$;
send $Q[RD_d^k]$; $\forall v \in N^k(c_{dv}^k \leftarrow T)$

FIG. 1

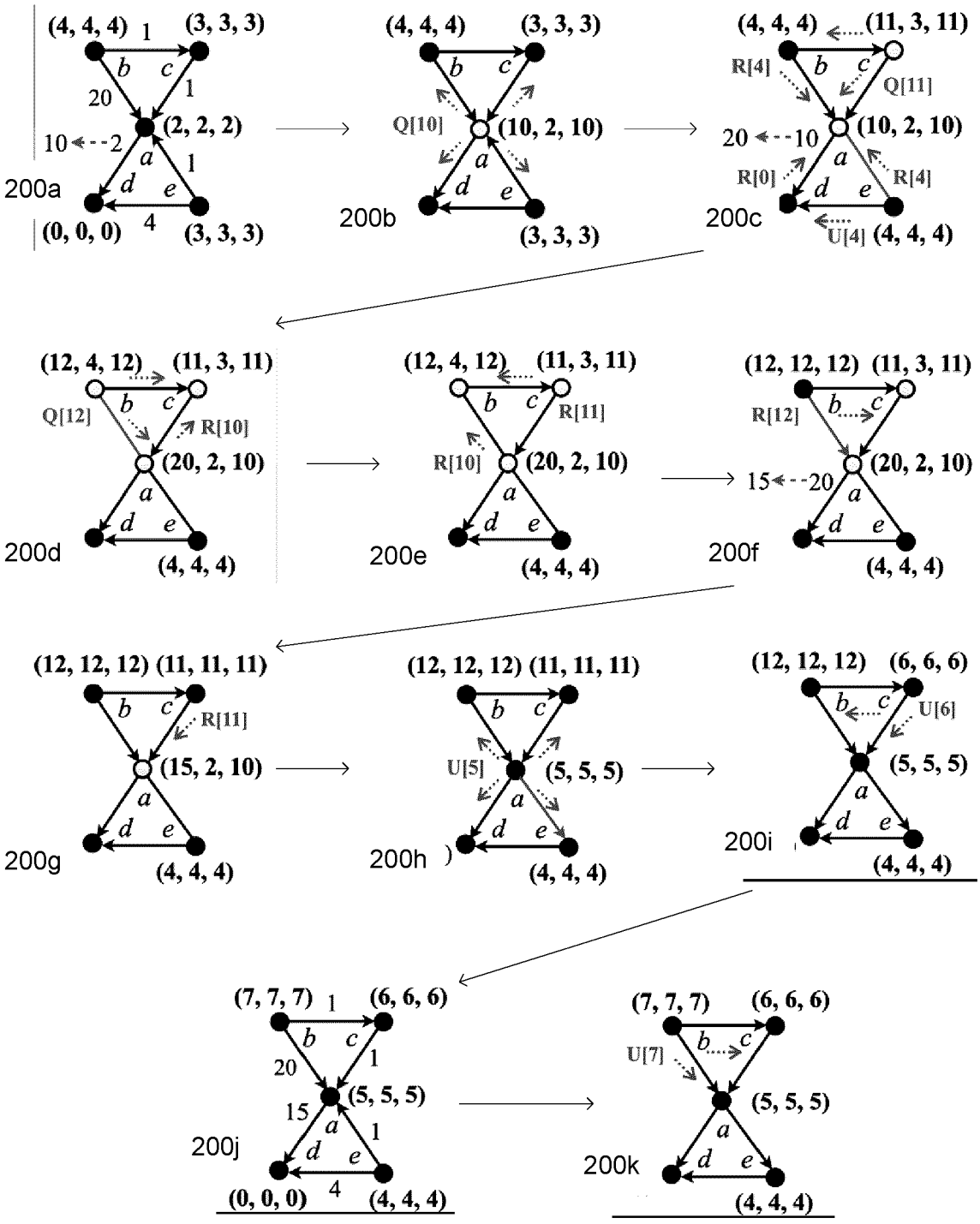
(4, 4, 4)
(3, 3, 3)
(2, 2, 2)
(0, 0, 0)
200a
200b
200c
(11, 3, 11)
(10, 2, 10)
Q[10]
R[4]
Q[11]
R[0]
U[4]
(12, 4, 12)
(11, 3, 11)
(20, 2, 10)
Q[12]
R[10]
R[11]
R[12]
200d
200e
200f
(12, 12, 12)
(11, 11, 11)
(15, 2, 10)
(5, 5, 5)
U[5]
U[6]
200g
200h
200i
(7, 7, 7)
(6, 6, 6)
U[7]
200j
200k

FIG. 2

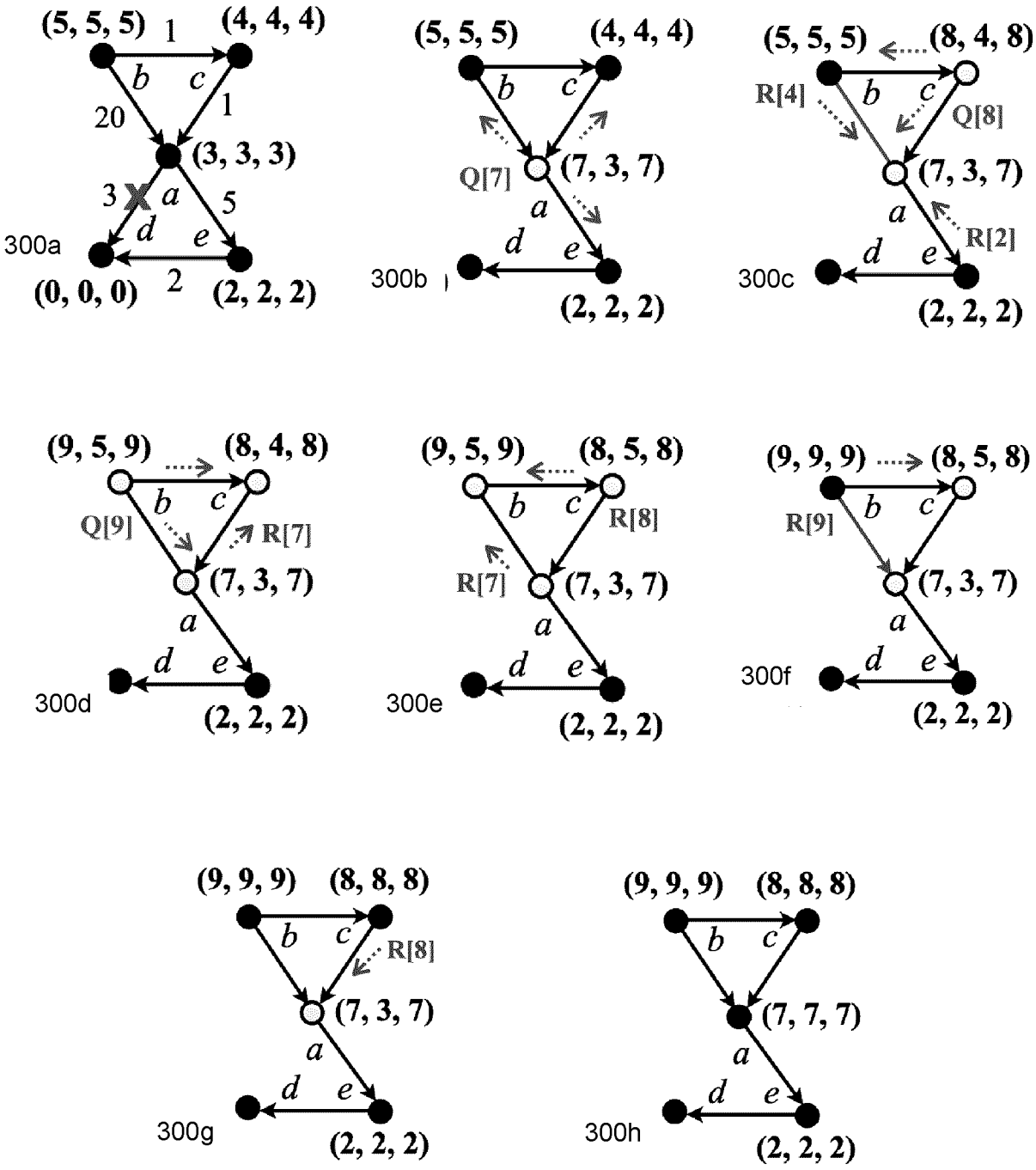
(5, 5, 5)
1
(4, 4, 4)
b
c
20
1
(3, 3, 3)
3
a
5
d
e
300a
2
(0, 0, 0)
(2, 2, 2)
(5, 5, 5)
(4, 4, 4)
b
c
Q[7]
(7, 3, 7)
a
d
e
300b
(2, 2, 2)
(5, 5, 5)
(8, 4, 8)
R[4]
b
c
Q[8]
(7, 3, 7)
a
R[2]
d
e
300c
(2, 2, 2)
(9, 5, 9)
(8, 4, 8)
b
c
Q[9]
R[7]
(7, 3, 7)
a
d
e
300d
(2, 2, 2)
(9, 5, 9)
(8, 5, 8)
b
c
R[8]
R[7]
(7, 3, 7)
a
d
e
300e
(2, 2, 2)
(9, 9, 9)
(8, 5, 8)
b
c
R[9]
(7, 3, 7)
a
d
e
300f
(2, 2, 2)
(9, 9, 9)
(8, 8, 8)
b
c
R[8]
(7, 3, 7)
a
d
e
300g
(2, 2, 2)
(9, 9, 9)
(8, 8, 8)
b
c
(7, 7, 7)
a
d
e
300h
(2, 2, 2)

FIG. 3

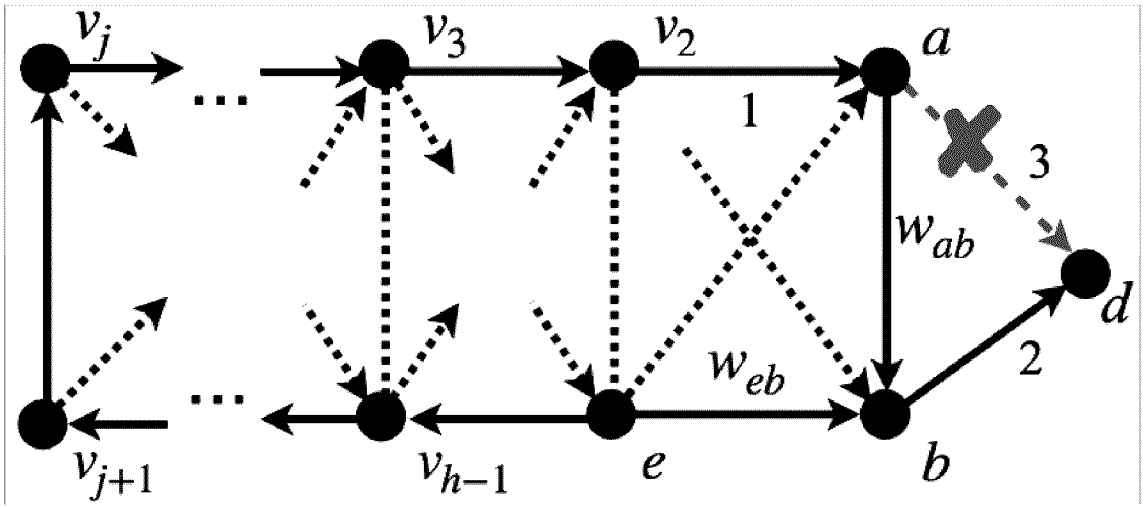

FIG. 4

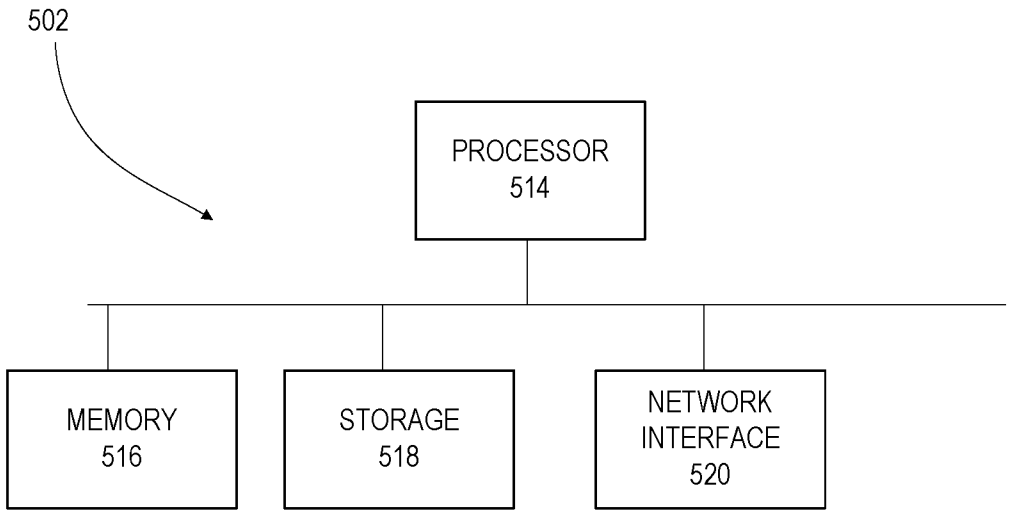
502
PROCESSOR
514
MEMORY
516
STORAGE
518
NETWORK
INTERFACE
520

FIG. 5

Algorithm 1 Router

INPUT: t_{min}, LT, RT^k, NT^k, LWT^k N^k
if (initialization signal) **then call Algorithm** 2 **end if**
if (UT^k expires) **then call Algorithm** 3 **end if**
if (weight of link (k, q) changes) **then call Algorithm** 4 **end if**
if (($LT_q^k = 0$) **then call Algorithm** 5 **end if**
if (M^q is received) **then call Algorithm** 6 **end if**

FIG. 6

Algorithm 2 Initialization

INPUT: t_{min}, LT, RT^k, NT^k, LWT^k
$N^k \leftarrow \{n \mid \exists(k, n)\}$; $N \leftarrow \{k\}$;
$D_k^k \leftarrow 0$ $RD_k^k \leftarrow 0$ $PD_k^k \leftarrow 0$; $Q_k^k \leftarrow k$; $A_k^k \leftarrow F$; $s_k^k \leftarrow k$; $S_k^k \leftarrow \{k\}$;
for each $n \in N^k$ **do**
  $l_n^k \leftarrow$ weight of link (k, n); $LT_n^k \leftarrow LT$;
  for each $v \in N$ **do** $D_{nv}^k \leftarrow \infty$; $c_{nv}^k \leftarrow F$ **end for**
  add $U[k, RD_k^k = 0]$ to $M^k(n)$; $UT^k \leftarrow 0$ % to send a "hello" message %
end for

FIG. 7

Algorithm 3 Send Message (UT^k expires)

INPUT: N^k, $\forall v \in N^k(M^k(v))$
for each $v \in N^k$ **do**
  if ($M^k(v) = \phi$) **then** add $U[k, 0]$ to $M^k(v)$;
  send $M^k(v)$ to v; $M^k(v) \leftarrow \phi$
end for
$UT^k \leftarrow t_{max}$

FIG. 8

Algorithm 4 Link-Weight Change (weight of link (k, q))

INPUT: q, N, LWT^k, weight of (k, q)
$l^* \leftarrow l_q^k$ *% remember prior link weight %*; $l_q^k \leftarrow$ weight of (k, q)
if ($l_q^k = \infty$) **then**
  if ($Q_d^k = q$) **then** $Q_d^k = k$ **end if**;
  $M^k(q) \leftarrow \emptyset$ *% delete message meant for q %*;
  for each $d \in N$ **do**
    $D_{dq}^k \leftarrow \infty$; $c_{dq}^k \leftarrow F$; **call Algorithm 7** $(U[d, D_d^q = D_{dq}^k],\ q)$
  end for
end if
if ($[l_q^k \neq \infty] \wedge [l^* \neq \infty]$) **then**
  for each $d \in N$ **do call Algorithm 7** $(U[d, D_d^q = D_{dq}^k],\ q)$ **end for**
end if
if ($[l_q^k \neq \infty] \wedge [l^* = \infty]$) **then**
  for each $d \in N$ **do**
    if $(D_d^k < \infty)$ **then** $M^k(q) \leftarrow U[d, D_d^k]$ *% update new neighbor q %*
  end for
end if
if ($\exists\ q \in N^k(M^k(q) \neq \emptyset)$) **then**
  if ($l_q^k = \infty$) **then** $UT^k \leftarrow 0$
  else if $(UT^k > t_{min}$) **then** $UT^k \leftarrow t_{min}$ **end if**
end if

FIG. 9

Algorithm 5 Neighbor Timeout

INPUT: N, N^k, NT^k, LWT^k, M^k
for each $q \in N^k$ **do**
  if $LT_q^k = 0$ **then**
    $M^k(q) \leftarrow \emptyset$ *% delete message meant for q %*;
    if ($Q_d^k = q$) **then** $Q_d^k = k$ **end if**
    for each $d \in N$ **do**
      $D_{dq}^k \leftarrow \infty$; $c_{dq}^k \leftarrow F$; **call Algorithm 7** $(U[d, \infty],\ q)$
    end for
  end if
end for
if ($[\exists\ q \in N^k(M^k(q) \neq \emptyset)] \wedge [UT^k > t_{min}]$) **then** $UT^k \leftarrow 0$ **end if**

FIG. 10

Algorithm 6 Message Received

INPUT: q, NT^k, RT^k, M^q, LT
Update lifetime for neighbor q: $LT_q^k \leftarrow LT$
if $q \notin N^k$ **then**
  Initialize entry for new neighbor q:
  Update NT^k **for** q: $c_{qq}^k \leftarrow F$; $D_{qq}^k \leftarrow 0$;
  Update RT^k **for** q:
  $S_q^k \leftarrow S_q^k \cup \{q\}$; $D_q^k \leftarrow Min\{D_{qn}^k + l_n^k \mid n \in S_q^k\}$;
  $s_q^k \leftarrow Min\{n \in S_q^k \mid D_{qn}^k + l_n^k = D_q^k\}$;
  for each $d \in N \setminus k$ **do**
    if $(D_d^k < \infty)$ **then** $U_d^k \leftarrow U[d, D_d^k]$
  end for
end if
for each update $U[d, D_d^q] \in M^q$ **do**
  call Algorithm 7 ($U[d, D_d^q]$, q)
end for
for each query $Q[d, D_d^q] \in M^q$ **do**
  call Algorithm 8 ($Q[d, D_d^q]$, q)
end for
for each reply $R[d, D_d^q] \in M^q$ **do**
  call Algorithm 11 ($R[d, D_d^q]$, q)
end for
if ($[\exists\, q \in N^k (M^k(q) \neq \emptyset)] \wedge [UT^k > t_{min}]$) **then** $UT^k \leftarrow t_{min}$ **end if**

FIG. 11

Algorithm 7 Update

INPUT: LT, LWT^k, NT^k, RT^k, $U[d, D_d^q]$, q
Remember current distance value: $D^* \leftarrow D_d^k$;
Update $NT^k(d)$: $D_{dq}^k \leftarrow D_d^q$;
if ($A_d^k = T$ % *Router k is currently active* %) **then**
  call Algorithm 10 ($U[d, D_d^q]$, D^*); **return**
end if
if ($A_d^k = F$ % *Router k is currently passive* %) **then**
  call Algorithm 9 ($U[d, D_d^q]$, D^*); **return**
end if

FIG. 12

Algorithm 8 Query

INPUT: LT, LWT^k, NT^k, RT^k, $Q[d, D_d^q]$, q
Remember current distance value: $D^* \leftarrow D_d^k$;
Update $NT^k(d)$: $D_{dq}^k \leftarrow D_d^q$;
if ($A_d^k = T$ % *Router k is currently active* %) **then**
  call Algorithm 13 ($Q[d, D_d^q]$, D^*); **return**
end if
if ($A_d^k = F$) % *Router k is currently passive* % **then**
  call Algorithm 12 ($Q[d, D_d^q]$, D^*); **return**
end if

FIG. 13

Algorithm 9 Passive Update

INPUT: LT, LWT^k, NT^k, RT^k, $U[d, RD_d^q]$, q, D^*
if ($\forall n \in N^k$ $(D_{dn}^k = \infty)$) **then**
  $\mathcal{MD} \leftarrow \infty$; $\mathcal{OD} \leftarrow \infty$; $S_d^k \leftarrow \emptyset$; $s_d^k \leftarrow 0$
else
  $\mathcal{MD} \leftarrow \min_{v \in N^k} \{D_{dv}^k + l_v^k\}$
  if ($D_{dq}^k < D_d^k$) **then** $S_d^k \leftarrow S_d^k \cup \{q\}$ **else** $S_d^k \leftarrow S_d^k \setminus \{q\}$;
  if ($S_d^k = \emptyset$) **then** $\mathcal{OD} \leftarrow \infty$ **else** $\mathcal{OD} \leftarrow \min_{v \in S_d^k} \{D_{dv}^k + l_v^k\}$
end if
if ($\mathcal{MD} = \mathcal{OD}$ % *Router remains passive:* %) **then**
  $D_d^k \leftarrow \mathcal{MD}$; $R_d^k \leftarrow D_d^k$; $P_d^k \leftarrow D_d^k$;
  if ($\mathcal{MD} < \infty$) **then**
    for all $v \in N^k$ **do**
      if ($D_{dv}^k < D_d^k$) **then** $S_d^k \leftarrow S_d^k \cup \{v\}$ **else** $S_d^k \leftarrow S_d^k \setminus \{v\}$
    end for
    if ($D_d^k = \infty$) **then** $s_d^k \leftarrow 0$ **else** $s_d^k \leftarrow Min\{v \in S_d^k \mid D_{dv}^k + l_v^k = D_d^k\}$;
  end if
  if ($D_d^k \neq D^*$) **then** % *send update:* % $U_d^k \leftarrow U[d, D_d^k]$
else
  % *Router becomes active:* %
  $PD_d^k \leftarrow RD_d^k$; **for each** $v \in N^k$ **do** $c_{dv}^k \leftarrow T$ **end for**;
  if ($S_d^k = \emptyset$) **then**
    if ($s_d^k = 0$) **then** $D_d^k \leftarrow \infty$ **else** $D_d^k \leftarrow D_{ds_d^k}^k + l_{s_d^k}^k$ **end if**;
  else
    $D_d^k \leftarrow SD$; $s_d^k \leftarrow Min\{v \in S_d^k \mid D_{dv}^k + l_v^k = D_d^k\}$
  end if
  for each $v \in N^k$ **do** $M^k(v) \leftarrow Q[d, RD_d^k]$ % *send query* % **end for**
end if
return

FIG. 14

Algorithm 10 Active Update

INPUT: LWT^k, NT^k, RT^k, $U[d, RD_d^q]$, q, D^*
if ($\forall n \in N^k$ $(D_{dn}^k + l_n^k = \infty)$) **then**
  $D_d^k \leftarrow \infty$; $S_d^k \leftarrow \emptyset$; $s_d^k \leftarrow 0$; $U_d^k \leftarrow U[d, D_d^k]$ % *send update* %;
end if
if ($D_{dq}^k < PD_d^k$) **then** $S_d^k \leftarrow S_d^k \cup \{q\}$ **else** $S_d^k \leftarrow S_d^k - \{q\}$;
if ($S_d^k = \emptyset$) **then**
  if ($s_d^k \neq 0$) **then** $D_d^k \leftarrow D_{ds_d^k}^k + l_{s_d^k}^k$ **else** $D_d^k \leftarrow \infty$ **end if**
else
  $D_d^k \leftarrow \min_{v \in S_d^k} \{D_{dv}^k + l_v^k\}$; $s_d^k \leftarrow \min\{v \in S_d^k \mid D_{dv}^k + l_v^k = D_d^k\}$
end if
return

FIG. 15

Algorithm 11 Reply

INPUT: LWT^k, NT^k, RT^k, $R[d, RD_d^q]$, q, D^*
$c_{dq}^k \leftarrow F$;
if ($\forall n \in N^k$ $(D_{dn}^k + l_n^k = \infty)$) **then**
  $D_d^k \leftarrow \infty$; $S_d^k \leftarrow \emptyset$; $s_d^k \leftarrow 0$
else
  if ($D_{dq}^k < PD_d^k$) **then** $S_d^k \leftarrow S_d^k \cup \{q\}$ **else** $S_d^k \leftarrow S_d^k \setminus \{q\}$;
end if
if ($S_d^k = \emptyset$) **then**
  if ($s_d^k \neq 0$) **then** $D_d^k \leftarrow D_{ds_d^k}^k + l_{s_d^k}^k$ **else** $D_d^k \leftarrow \infty$ **end if**
else
  $D_d^k \leftarrow \min_{v \in S_d^k} \{D_{dv}^k + l_v^k\}$; $s_d^k \leftarrow \min\{v \in S_d^k \mid D_{dv}^k + l_v^k = D_d^k\}$
end if
if ($\forall v \in N^k (c_{dv}^k = F)$) **then**
  $D_d^k \leftarrow \min_{v \in N^k} \{D_{dv}^k + l_v^k\}$; $RD_d^k \leftarrow D_d^k$; $PD_d^k \leftarrow D_d^k$;
  if ($RD_d^k = \infty$) **then** $s_d^k \leftarrow 0$ **else** $s_d^k \leftarrow \min\{v \in S_d^k \mid D_{dv}^k + l_v^k = D_d^k\}$;
  if ($Q_d^k = v \neq k$) **then** $M^k(v) \leftarrow R[d, RD_d^k]$ **end if**
  for each $n \in N^k$ **do** $M^k(n) \leftarrow U[d, RD_d^k]$ **end for**
end if
$Q_d^k \leftarrow k$
return

FIG. 16

Algorithm 12 Passive Query

INPUT: LT, LWT^k, NT^k, RT^k, $Q[d, RD_d^q]$, q, D^*
if ($\forall n \in N^k$ $(D_{dn}^k + l_{n_k}^k = \infty)$) **then**
  $D_d^k \leftarrow \infty$; $S_d^k \leftarrow \emptyset$; $s_d^k \leftarrow 0$
  $M^k(q) \leftarrow R[d, \infty]$ % *send response to q* %;
  if $(D^* \neq \infty)$ **then for each** $v \in N^k$ **do** $M^k(v) \leftarrow U[d, \infty]$ % *send update*%
  return
end if
$\mathcal{MD} \leftarrow \min_{v \in N^k} \{D_{dv}^k + l_v^k\}$
if ($D_{dq}^k < RD_d^k$) **then** $S_d^k \leftarrow S_d^k \cup \{q\}$ **else** $S_d^k \leftarrow S_d^k \setminus \{q\}$ **end if**;
if ($S_d^k = \emptyset$) **then** $\mathcal{OD} \leftarrow \infty$ **else** $\mathcal{OD} \leftarrow \min_{v \in S_d^k} Min\{D_{dv}^k + l_v^k\}$ **end if**;
if ($[S_d^k \neq \emptyset] \wedge [\mathcal{MD} = \mathcal{OD}]$) **then**
  % *Router remains passive:* %
  $D_d^k \leftarrow \mathcal{MD}$; $RD_d^k \leftarrow D_d^k$; $PD_d^k \leftarrow D_d^k$;
  for all $v \in N^k$ **do**
    if ($D_{dv}^k < RD_d^k$) **then** $S_d^k \leftarrow S_d^k \cup \{v\}$ **else** $S_d^k \leftarrow S_d^k \setminus \{v\}$ **end if**
  end for
  $s_d^k \leftarrow \min\{v \in S_d^k \mid D_{dv}^k + l_v^k = D_d^k\}$;
  $M^k(q) \leftarrow U[d, RD_d^k]$ % *send response to q* %
  if $(RD_d^k \neq D^*)$ **then**
    for each $v \in N^k$ **do** $M^k(v) \leftarrow U[d, RD_d^k]$ **end for**
  end if
else
  % *Router becomes active:* %
  $PD_d^k \leftarrow RD_d^k$;
  if $(q = s_d^k)$ **then** $Q_d^k \leftarrow q$ **else** $Q_d^k \leftarrow k$ **end if**;
  if $(S_d^k = \emptyset)$ **then**
    if ($s_d^k = 0$) **then** $D_d^k \leftarrow \infty$ **else** $D_d^k \leftarrow D_{d s_d^k}^k + l_{s_d^k}^k$ **end if**;
  else
    $D_d^k \leftarrow Min\{D_{dv}^k + l_v^k \mid v \in S_d^k\}$;
    $s_d^k \leftarrow Min\{v \in S_d^k \mid D_{dv}^k + l_v^k = D_d^k\}$
  end if
  for each $v \in N^k$ **do**
    $c_{dv}^k \leftarrow T$; $M^k(v) \leftarrow P[d, RD_d^k]$ % *send query* %
  end for
end if
return

FIG. 17

Algorithm 13 Active Query

INPUT: LT, LWT^k, NT^k, RT^k, $P[d, RD_d^q]$, q, D^*
if ($\forall n \in N^k$ $(D_{dn}^k + l_n^k = \infty)$) **then**
  $D_d^k \leftarrow \infty$; $S_d^k \leftarrow \emptyset$; $s_d^k \leftarrow 0$
  $M^k(q) \leftarrow R[d, \infty]$ *% send response to q %*;
  return
end if
if ($D_{dq}^k < PD_d^k$) **then** $S_d^k \leftarrow S_d^k \cup \{q\}$ **else** $S_d^k \leftarrow S_d^k \setminus \{q\}$ **end if**;
if ($S_d^k = \emptyset$) **then**
  if ($s_d^k \neq 0$) **then** $D_d^k \leftarrow D_{ds_d^k}^k + l_{s_d^k}^k$ **else** $D_d^k \leftarrow \infty$ **end if**
else
  $D_d^k \leftarrow \min_{v \in S_d^k} \{D_{dv}^k + l_v^k\}$; $s_d^k \leftarrow \min\{v \in S_d^k \mid D_{dv}^k + l_v^k = D_d^k\}$
end if
if ($[q = s_d^k] \wedge [Q_d^k = k]$) **then** $Q_d^k \leftarrow q$ **end if**
if ($[q \neq s_d^k] \wedge [q \neq Q_d^k]$) **then** $M^k(q) \leftarrow R[d, RD_d^k]$ **end if**
return

FIG. 18

INTERNATIONAL SEARCH REPORT

International application No.
PCT/CA2025/050978

A. CLASSIFICATION OF SUBJECT MATTER

IPC: ***H04L 45/18*** (2022.01)

CPC: H04L 45/18 (2022.05)

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

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC (2022.01): H04L

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

Electronic database(s) consulted during the international search (name of database(s) and, where practicable, search terms used)
Questel Orbit, Canadian Patent Office Database (Intellect), IEEE Xplore.
Keywords: maintain, loop, free, path, source, destination, node, link weight, rout*, table, distance, condition, activ*, state, short.

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 2014/0219275 A1 (Allan et al.), 07 August 2014 (07-08-2014) *See abstract, paragraphs [0006] – [0009], [0054], [0071], figures 1-10	1-21
A	US 2016/0173386 A1 (Garcia-Luna-Aceves), 16 June 2016 (16-06-2016) *See whole document	1-21
A	US 2018/0302807 A1 (Chen et al.), 18 October 2018 (18-10-2018) *See whole document	1-21

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

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

Date of the actual completion of the international search 08 August 2025 (08-08-2025)	Date of mailing of the international search report 16 October 2025 (16-10-2025)
Name and mailing address of the ISA/CA Canadian Intellectual Property Office Place du Portage I, C114 - 1st Floor, Box PCT 50 Victoria Street Gatineau, Quebec K1A 0C9 Facsimile No.: 819-953-2476	Authorized officer Sajith Bandaranayake (819) 639-5760

INTERNATIONAL SEARCH REPORT
Information on patent family members

International application No.
PCT/CA2025/050978

Patent Document Cited in Search Report	Publication Date	Patent Family Member(s)	Publication Date
US2014219275A1	07 August 2014 (07-08-2014)	US9178799B2	03 November 2015 (03-11-2015)
		AU2014210787A1	03 September 2015 (03-09-2015)
		AU2014210787B2	01 December 2016 (01-12-2016)
		CN105052093A	11 November 2015 (11-11-2015)
		CN105052093B	23 July 2019 (23-07-2019)
		CN107257320A	17 October 2017 (17-10-2017)
		CN107257320B	30 July 2021 (30-07-2021)
		EP2951965A2	09 December 2015 (09-12-2015)
		EP2951965B1	25 September 2019 (25-09-2019)
		EP3253011A2	06 December 2017 (06-12-2017)
		EP3253011A3	07 March 2018 (07-03-2018)
		EP3253011B1	10 March 2021 (10-03-2021)
		IL240121B	31 July 2018 (31-07-2018)
		US2016028612A1	28 January 2016 (28-01-2016)
		US9461909B2	04 October 2016 (04-10-2016)
		WO2014118718A2	07 August 2014 (07-08-2014)
		WO2014118718A3	06 November 2014 (06-11-2014)
US2016173386A1	16 June 2016 (16-06-2016)	US10237189B2	19 March 2019 (19-03-2019)
		AU2015264822A1	30 June 2016 (30-06-2016)
		CN105704030A	22 June 2016 (22-06-2016)
		CN105704030B	04 December 2020 (04-12-2020)
		EP3035611A1	22 June 2016 (22-06-2016)
		EP3035611B1	11 July 2018 (11-07-2018)
		JP2016116218A	23 June 2016 (23-06-2016)
		KR20160073304A	24 June 2016 (24-06-2016)
US2018302807A1	18 October 2018 (18-10-2018)	US11758421B2	12 September 2023 (12-09-2023)
		CN107873126A	03 April 2018 (03-04-2018)
		CN107873126B	05 February 2021 (05-02-2021)
		EP3284287A1	21 February 2018 (21-02-2018)
		EP3284287B1	05 June 2019 (05-06-2019)
		JP2018516499A	21 June 2018 (21-06-2018)
		JP6535387B2	26 June 2019 (26-06-2019)
		KR20180008489A	24 January 2018 (24-01-2018)
		WO2016168603A1	20 October 2016 (20-10-2016)
		WO2016168603A9	08 December 2016 (08-12-2016)