Title

Distance-Based Loop-Free Routing with Positive and Negative Link Weights

Journal

12th International Symposium on Networks, Computers and Communications

Authors

Garcia-Luna-Aceves, J.J.

Moghaddassian, M.

Hsieh, C.

Publication Date

2025-10-01

Peer reviewed

Distance-Based Loop-Free Routing with Positive and Negative Link Weights

J.J. Garcia-Luna-Aceves, Morteza Moghaddassian, Charles Hsieh
Electrical and Computer Engineering Department, University of Toronto
jj.garcialunaaceves@utoronto.ca, m.moghaddassian@utoronto.ca, charles.hsieh@mail.utoronto.ca

***Abstract*—An efficient approach to distance-based loop-free routing is introduced called THOR (Transitive Hop-Ordered Routing) that, in contrast to all prior distributed routing algorithms and routing protocols, works correctly in the presence of negative link weights. THOR is shown to provide loop-free routing and to converge to shortest paths within a finite time. THOR is compared with OSPF using the ns-3 simulator for the case of minimum-hop routing, and the simulation results show that the approach used in THOR leads to faster convergence and less signalling overhead.**

I. Introduction

The routing protocols operating within autonomous systems in the IP Internet today are based on only a few types of shortest-path routing algorithms designed to ensure eventual convergence to correct routes, and in some cases loop-free routing at all times. As Section II further describes, all prior distributed shortest-path routing algorithms operate correctly with the assumption that link weights are positive.

Section III introduces THOR (Transitive Hop-Ordered Routing). In contrast to prior solutions to loop-free routing THOR uses two separate conditions to attain loop freedom and optimal routes. A *Condition for Loop-free Successors* (CLS) is used by each router to determine which neighbors it can use as successors (next hops) to destinations, and a *Condition for Selected Route* (CSR) is used to determine which of the loop-free routes available is the best route (the shortest distance to a destination). CLS requires that a successor of a router to a destination must report a route that has a smaller number of hops than the number of hops currently assumed by the router.

As long as CSR is satisfied, a router can change its successors to destinations independently of others in much the same was as it is done in the Distributed Bellman-Ford (DBF) algorithm. When CSR is not satisfied, a router sends a probe to all or some of its neighbors asking to find a router through which the shortest distance to the destination can be attained and such that it can show a smaller hop count to the destination than the count at the probing router. Whether or not a router is waiting for responses, it uses CLS to decide which neighbors to use as successors.

Section IV shows that THOR is loop-free and converges to shortest paths, even when link weights are negative. Section V presents results of simulations comparing THOR with OSPF [13], which is a well-known example of the state of the art in shortest-path routing in the Internet. The simulation experiments are based on our implementations of THOR in ns-3 for minimum-hop routing and an off-the-shelf implementations of OSPF in Quagga integrated with the ns-3 simulator. The results show that the approach used in THOR leads to faster converge and less signalling overhead.

II. Related Work

Many routing protocols have been developed based on the dissemination of partial or complete topology information (e.g., [1], [5], [8], [10], [12], [13]) or the use of complete path information in routing updates (e.g., BGP [17]). These approaches are intended to address the non-convergence problem of the DBF algorithm, which makes such protocols like RIPv2 [11] ineffective in large networks. Some protocols share path information incrementally (e.g., [7]) to accomplish the same result as conveying complete path information in every update. Using topology or path information in routing protocols is arguably the most common approach today; however, it does not guarantee by itself that routes are loop-free at every instant.

Several routing protocols have used destination sequence numbers to ensure loop-free routing. The many examples of this approach include DSDV [14], AODV [15], and LOADng [3]. RPL [20] is a more recent example of the use of destination sequence numbers.

There are two limitations with using destination sequence numbers to validate distances. First, flooding of new sequence numbers by destinations is necessary because only the destinations can ensure that all routers receive the most recent sequence numbers over time. Second, DSDV and AODV focus on minimum-hop routing, and additional measures would be needed to ensure optimum distances when link weights vary.

A number of distance-based approaches provide loop-free routes at every instant by requiring routers to coordinate the updating of routing tables on a multi-hop basis (e.g., [4], [9], [16], [22]). The Diffusing Update Algorithm (DUAL) [4] has become the most popular of these schemes because it is the basis of Cisco's EIGRP [18]. There are other approaches based on diffusing computations (e.g., Babel [2], DIV [16]).

There are three limitations with the diffusing computations used in DUAL and similar distributed routing algorithms. First, trying to establish local ordering at the same time that new shortest paths are established prevents routers from using loop-free paths that need not be shortest paths while shortest paths are being found. Second, a router that sends a query cannot select a new next hop until it receives replies from all its neighbors, even if some of them have replied and offer

valid loop-free routes. Third, handling multiple concurrent diffusing computations is complex, because multiple diffusing computations may be started from multiple routers for the same destination, changes in link costs, and updates received from neighbors, and all of these may occur concurrently [4].

Importantly, no prior routing algorithm or protocol can converge to valid routes or provide loop-free routing in the presence of negative link weights.

III. THOR

A. Overview

To compute routes to destinations other than itself, each router uses two local conditions and iterative ordering computations to maintain loop freedom at all times and converge to optimal loop-free routes within a finite time.

A router uses a *Condition for Loop-free Successors* (CLS) to decide which neighbors it can use as successors to destinations in a way that loop freedom is guaranteed at any given time.

CLS uses an ordering reference associated with each route, and requires that any successor selected by a router must report a shorter ordering reference than the ordering reference currently held by the router. The ordering reference of a route is the number of hops in the route to the destination.

A router uses a *Condition for Selected Route* (CSR) to determine whether it can change its successors to a destination independently of other routers and ensure that at least one of its successors offers a loop-free route with the shortest distance. CSR is satisfied when the shortest distance through any loop-free route equals the shortest distance attained through a neighbor that satisfies CLS.

A router selects its successors independently of its neighbors if CSR is satisfied. Otherwise, it engages in an iterative ordering computation by sending a probe to one or more of its neighbors asking for routers with smaller ordering references than the ordering reference at the router itself.

Routers send their responses if they either offer ordering references smaller than the ordering reference stated in a probe, or detect a potential loop. A probing router iterates on its selection of routes among those that are guaranteed to be loop free, and reorders itself (i.e., changes its ordering reference) as needed after receiving all the responses it requested.

Importantly, routers forward responses they receive while waiting for additional responses, and the signaling used in iterative ordering computations prevents a router from using as a successor a neighbor that appears to offer the shortest distance due to the presence of negative link weights but would lead to a loop if it were chosen as a successor.

B. Information Maintained and Shared by Router k

Router k maintains three tables, which are described below.

Link Weight Table (LWT^k): Lists the weight ($w(k,q)$) and lifetime ($LT(k,q)$) of link (k,q) with each neighbor q. A link weight is a real number and the maximum lifetime of a neighbor entry is a constant LT defined for the network.

Routing Table (RT^k): Contains an entry for each destination d. The entry for d in RT^k is denoted by $RT^k(d)$ and states: An ordering reference $O^k(d)$, a distance ($D^k(d)$), a probing reference ($P^k(d)$), an anchoring reference ($A^k(d)$), the set of successors ($S^k(d)$), and the preferred successor ($s^k(d)$) in $S^k(d)$.

$D^k(d)$ states the distance to destination d attained through $s^k(d)$. $O^k(d)$ equals the number of hops along the path through $s^k(d)$. $P^k(d)$ equals the smallest value of a probing reference sent in a probe by router k while it waits for responses to a probe it has sent, and it is initialized as $P^k(d) = O^k(d)$. $A^k(d)$ equals the smallest value of an anchoring reference sent in a response from router k while it is busy and waits for responses, and it is set as $A^k(d) = \infty$ in idle state. Distance values are real numbers and ordering-reference values are integer numbers.

Neighbor Table (NT^k): Has an entry for each neighbor $q \in N^k$. For each destination d, the entry for q is denoted by $NT^k(d,q)$ and contains $O^k(d,q)$, $D^k(d,q)$, a Boolean-value response flag ($R^k(d,q)$), and a Boolean-value loop flag ($L^k(d,q)$).

The values of $O^k(d,q)$ and $D^k(d,q)$ equal the the last value of $O^q(d)$ and $D^q(d)$ received from neighbor q, respectively.

The value of $R^k(d,q)$ is true (T) if router k is waiting to receive a response from q and is false (F) otherwise.

The value of $L^k(d,q)$ is true (T) if router k has determined that using q as a successor would result in a routing-table loop, and is false (F) otherwise.

If a neighbor v has not reported any distance for d to router k, then router k sets $O^k(d,v) = \infty$, $D^k(d,v) = \infty$, $R^k(d,v) = F$, and $L^k(d,v) = F$.

A router may send updates, probes or responses after updating its routing table.

Update: An update from router k for destination d is sent to all its neighbors and is a tuple $U[d, O^k(d), D^k(d)]$ stating its ordering reference and distance.

Probe: A probe from router k for destination d is the tuple $P[d, o^k(d), O^k(d), D^k(d), P^k(d)]$ and is sent to one or multiple neighbors. The variable $o^k(d)$ states the origin of the probe, and $P^k(d)$ is the probing reference stated by the router that originated the probe.

Response: A response from router k for destination d is sent to all its neighbors and is a tuple $R[d, o^k(d), O^k(d), D^k(d), A^k(d)]$, where $A^k(d)$ is an anchoring reference stated by the router that originated the response.

C. Condition for Loop-Free Successors (CLS)

Let the set of neighbor routers of router k be N^k. Router k can use v as a successor for destination d if:

$$CLS: \left(v \in N^k\right) \wedge \left(O^k(d,v) < O^k(d)\right) \tag{1}$$

Theorem 1: CLS is a valid sufficient condition for loop-free routing.

Proof: Assume that L is a routing-table loop at time t and let $L = \{v_1 \rightarrow v_2 \rightarrow ... \rightarrow v_h \rightarrow v_1\}$. For the sake of contradiction, assume that CLS is true at every router in L

when at least one router in L changes its successor when it becomes part of the loop.

Each router $v_i \in L$ informs its neighbors of its ordering reference for 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 CLS is satisfied.

The successor to destination d for router v_i at time t is denoted by $n^{v_i}(d)[t]$, 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

$$\forall v_i \in L \Big((n^{v_i}(d)[t] = n^{v_i}(d)[t_i^+]) \wedge (O^{v_i}(d)[t] = O^{v_i}(d)[t_i^+]\ ) \Big) \tag{2}$$

Let $O^{v_i}(d)[t]$ and $O^{v_i}(d, v_{i+1})[t]$ denote the value of $O^{v_i}(d) and O^{v_i}(d, v_{i+1})$ at time t, respectively. Given that CLS is satisfied at every instant at every router, Eq. (2) implies that:

$$\forall v_i \in L \Big(O^{v_i}(d)[t_i^+] = O^{v_i}(d)[t] > O^{v_i}(d, v_{i+1})[t] = O^{v_i}(d, v_{i+1})[t_i^+] = O^{v_{i+1}}(d)[t_{i+1}] = O^{v_{i+1}}(d)[t] \Big) \tag{3}$$

Eqs. (2) and (3) are true only if $O^{v_i}(d)[t] > O^{v_{i+1}}(d)[t]$ for each v_i in L. However, this is a contradiction, because it implies that $O^{v_i}(d)[t] > O^{v_i}(d)[t]$ for each v_i in L. Therefore, the theorem is true. ■

D. Condition for Selected Route (CSR)

Router k can select successors to any destination $d \neq k$ independently of other routers as long as the following condition is satisfied:

$$CSR: \ D^k(d) = D^k(min) \tag{4}$$

$$D^k(d) = \min_{v \in N^k} \{D^k(d, v) + w(k, v) \mid O^k(d, v) < O^k(d)\}$$

$$D^k(min) = \min_{v \in N^k} \{D^k(d, v) + w(k, v) \mid L^k(d, v) = F\}$$

Eq. (4) indicates that router k can choose its successors independently of other routers as long as the minimum distance through a neighbor that reported a shorter ordering reference than its own ordering reference equals the minimum distance through any neighbor that offers a loop-free route.

Theorem 2: If link weights equal real numbers and CSR is always satisfied when routers select their successors along loop-free routes, then the preferred routes selected by routers have minimum distances to destinations that can be reached.

Proof: By induction on ordering-reference values away from a destination d.

Without loss of generality assume that link (v_1, d) has the smallest weight among all links from d to any of its neighbors.

Consider a simple path $P = \{v_h \rightarrow v_{h-1} \rightarrow ...v_1 \rightarrow d\}$ of h hops and assume that CSR is satisfied at each router in P when the router selects its successor along path P.

Let $\mathcal{MD}(k, d)$ denote the minimum distance that can be attained from router k to destination d along loop-free paths according to the actual network topology, rather than what a routing algorithm can compute.

Because link weights are real numbers and the distance associated with a path is the sum of link weights along the path, all distances are real numbers. Hence, any router can select the minimum distance among all the distances attained through its neighbors along loop-free paths.

Because $O^{v_1}(d, d) = O^d(d)$ and $O^{v_1}(d) = 1$, and by definition, $O^d(d) = 0$, it is true that $O^{v_1}(d, d) < O^{v_1}(d)$.

By definition, $D^d(d) = D^{v_1}(d, d) = 0$. Given that CSR is always satisfied, $D^{v_1}(d) = w(v_1, d)$, and router v_1 does not have any other loop-free route to d with a distance that is shorter than $w(v_1, d)$. Hence, $D^{v_1}(min) = \mathcal{MD}(v_1, d)$ and we can conclude that

$$D^{v_1}(d) = D^{v_1}(min) = \mathcal{MD}(v_1, d) \text{ with } O^{v_1}(d) = 1. \tag{5}$$

Assume that $D^{v_j}(d) = \mathcal{MD}(v_j, d)$, with $O^{v_j}(d) = j$ and $1 \leq j < h$. Because CSR is always satisfied, router v_{j+1} must have $D^{v_{j+1}}(d) = \mathcal{MD}(v_j, d) + w(v_{j+1}, v_j)$. Furthermore, $D^{v_{j+1}}(d) = D^{v_{j+1}}(min)$, and $D^{v_{j+1}}(min)$ is the smallest distance that router v_{j+1} can attain over any loop-free path through a neighbor. Therefore, $D^{v_{j+1}}(min) = \mathcal{MD}(v_{j+1}, d)$ and we can write

$$D^{v_{j+1}}(d) = \mathcal{MD}(v_{j+1}, d) \text{ with } O^{v_{j+1}}(d) = j + 1. \tag{6}$$

It follows from Eqs. (5) and (6) that the theorem is true. ■

E. Independent Selection of Successors

Figure 1 illustrates how CLS is used in THOR. Router k can always use as a successor to destination d any neighbor v that has reported $O^v(d) = O^k(d, v) < O^k(d)$. Such neighbors constitute $S^k(d)$. However, the routes resulting from using CLS need not be the shortest routes based on all the physical paths available in the network.

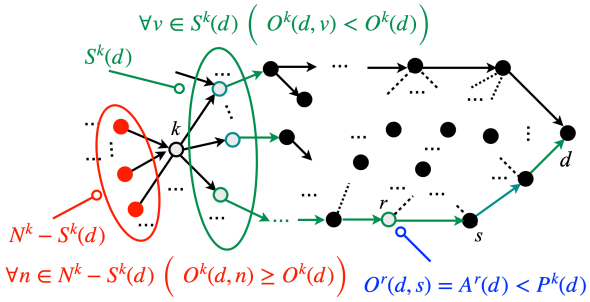

Fig. 1. Use of CLS in THOR

As long as CSR is satisfied, router k is able to update its shortest distance along loop-free paths without coordinating with its neighbors and simply sends updates as needed to report any changes to its distance or ordering reference.

Figure 2 shows an example of how a router k uses CSR to determine whether the shortest distance to destination d is attained through a neighbor in $S^k(d)$. From Eq. (4), we observe that CSR eliminates from consideration neighbors that have their loop flags set to true, which means that adopting them as successors would lead to loops.

Figure 2(a) shows an example in which all link weights are positive. Figure 2(b) shows an example with the same topology shown in Figure 2(a) but with link (a, k) having a negative weight $w(a, k) = -2$. The figures indicate the successor(s) of routers with arrowheads. The ordering reference and shortest

distance at each router v is stated by the value pairs "(ordering reference, distance)" next to each router, and link weights are indicated next to the links. The loop-flag values stored at router k are listed in each figure.

In Figure 2(a), neighbors a and c are not in S_d^k because they reported $O^a(d) \geq O^k(d)$ and $O^c(d) \geq O^k(d)$. We observe that $L^k(d,a) = T$, which in this case results from router a using k as its preferred successor. We also observe that $D^a(d) > D^k(d)$ and $D^c(d) > D^k(d)$, which is a consequence of all link weights being positive, because adding more hops to a route can only increase its distance.

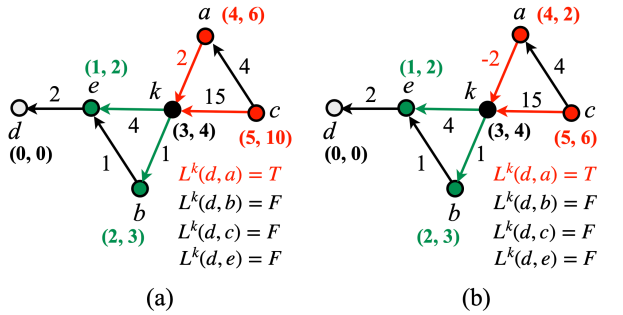

Fig. 2. Use of CSR in THOR: The pair (x, y) next to a node indicates the ordering reference (x) and shortest distance (y) reported by a router

Because $w(a,k) = -2$ in the example of Figure 2(b), router a reports $D^a(d) < D^k(d)$ even though it has a longer path ($O^a(d) > O^k(d)$) going through k. However, router k cannot consider neighbor a to obtain a loop-free route with minimum distance, because $L^k(d,a) = T$ and router k knows that adopting a route through a would create a loop.

How routers set loop flags that correctly prevent routing-table loops even when link weights are negative is part of the signaling used in iterative ordering computations, which we describe next.

F. Iterative Ordering Computations

Router k starts an *Iterative Ordering Computation* if CSR is not satisfied after an input event. An iterative ordering computation started by router k is aimed at: (a) adding neighbors that offer valid shortest paths to $S^k(d)$; (b) changing the value of $O^k(d)$ if needed without creating a loop; and (c) detecting neighbors whose distances correspond to paths that, if taken, would create loops.

We say that router k is *idle* or in idle state for destination d if CSR is satisfied. Otherwise, we say that the router is *busy* or in the busy state, i.e., involved in an iterative ordering computation. A router engages in an iterative ordering computation by sending probes to one, some, or all its neighbors.

The *probing reference* ($P^k(d)$) stated in a probe originated by router k is equal to its ordering reference ($O^k(d)$). The probe asks for a response to be sent back if either a router r is found that reports an *anchoring reference* ($A^r(d)$) such that $A^r(d) < P^k(d)$, or a possible loop is detected by a router v and the anchoring reference it reports is $A^v(d) \geq P^k(d)$.

Router k sends a probe to all its neighbors other than its preferred successor if CSR is not satisfied and none of its neighbors satisfies CLS. If CSR is not satisfied and some neighbors satisfy CLS, router k sends a probe to any neighbor that does not satisfy CLS but offers a route with a distance shorter than $D^k(d)$ to either detect loops or obtain routes with shorter distances that do not involve a loop.

Once router k is busy in an iterative ordering computation for d, it changes the value of $O^k(d)$ once it receives the responses it requested from its neighbors. A router keeps track of those neighbors from which responses are needed using the response flag. If neighbor n owes a response to router k, then $R^k(d,n) = T$.

If a router remains idle when it is asked to process a probe, it forwards the probe to its successor, and forwards the probe to all neighbors other than its successor if the router becomes busy. If a router is already busy in an iterative ordering computation and is asked to process another probe, it forwards the probe if the probe states a smaller probing reference than the one it currently stores, and remembers the new probing-reference value.

A router creates a response to a probe received from a neighbor v and originated by router k in the following cases:

Case 1: Router r is idle when it receives a probe from $v \neq s^r(d)$ and $O^r(d) = P^k(d)$. In this case, router r remains idle after processing the probe and sends a response that states $A^r(d) = O^r(d) - 1 < P^k(d)$.

Case 2: Router n is idle when it receives a probe from $v = s^r(d)$ and remains idle after processing the probe. The response from n states $A^n(d) = O^n(d) > P^k(d)$ in this case.

Case 3: Router k is busy and receives its own probe from v after its probe traverses a loop. The response from router k to its own probe states $A^k(d) = P^k(d)$.

If router k is busy and receives a response from neighbor n stating $A^n(d) < P^k(d)$, it associates the response with a loop-free path and sets $L^k(d,n) = F$. Importantly, a busy router k associates a response from neighbor n that states $o^n(d) = k$ and $A^n(d) \geq P^k(d)$ with a loop and sets $L^k(d,n) = T$. This results in router k blocking those neighbors that use the router in their own routes to d.

With reliable transmissions, a response originated by any router necessarily flows back to the origin of a probe because, independently of whether a relaying router is idle, busy, or transitions to the idle state, it forwards a response if the anchoring it carries is smaller than its own ordering reference and, if the router is busy, is also smaller than the smallest anchoring in any response it has forwarded to all neighbors.

Figure 3 shows two examples of how loop flags are set. In the example shown in Figure 3(a), router k sets $L^k(d,a) = T$ and $L^k(d,b) = T$ because a and b send responses with invalid anchoring references. The response from router b states $A^b(d) = 2 > P^k(d) = 1$ because it is a leaf node, which always remain idle and $k = s^b(d)$. The response from a carries $A^a(d) = P^k(d)$. In the example of Figure 3(b), router k sets $L^k(d,c) = F$ and $L^k(d,b) = F$ because both neighbors send valid responses.

Once a router k receives responses from all the neighbors that were asked for a response, the router computes a new

minimum distance from the distances reported by neighbors that have not been blocked with the loop flag. A preferred successor is selected among those neighbors that offer the shortest distance along a loop-free path, and the ordering reference is set to one hop more than the ordering reference reported by its new preferred successor.

In the absence of local input events, router k can send a probe periodically to a neighbor n for which $L^k(d,n) = T$ to take into account the possibility that the path offered by neighbor n router stops involving a loop, even though $O^k(d,n)$ and $D^k(d,n)$ do not change.

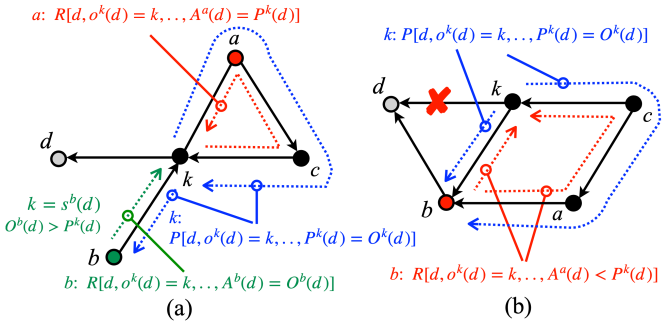

Fig. 3. Detection of valid and invalid routes in THOR

G. Updating Routing Tables

There are three important features in THOR that result in much higher efficiency than the use of destination sequence numbers or diffusing computations. First, a busy router continues to update its preferred successor and distance to a destination, rather than having to wait until it becomes idle again to make those updates. This reduces the amount of time a busy router may have to be blocked from having valid routes that may also be the shortest routes. Second, an idle router forwards a response as long as it states an anchoring reference shorter than its own ordering reference, and a busy router forwards a response as long as it carries an anchoring reference shorter than the smallest anchoring reference in a prior response it has forwarded. This prevents busy routers that experience long delays in receiving all their responses from slowing down other routers waiting for responses. Third, a busy router must become idle shortly after receiving the first valid response from a neighbor. This is because a busy router forwards a valid response to all its neighbors, and they in turn send the remaining responses that allow the router to become idle.

H. Initialization

Router k can process input events once an initialization time elapses that is long enough to ensure that, in the event that router k is restarting after a failure, all neighbor routers have processed previous routing messages from router k and also determined that they cannot communicate with the router directly.

Router k is initialized in idle state for itself and all destinations by setting $D^k(k) = 0$, $O^k(k) = 0$, $P^k(k) = 0$, $s^k(k) = k$, and $S^k(k) = \{k\}$. No entries need to exist for any other destination, and a no entry is interpreted as infinite distances and no successors. Router k then sends a "hello" message, a routing message with no updates or probes, to announce its presence to its neighbors.

I. Processing a Link Change

Router k updates the link weight $w(k,q)$ after the link event. If the link just changed weight, router updates RT^k for destination d assuming that an update is received from neighbor q with the updated values in NT^k and LWT^k.

THOR handles link failures using the lifetimes of neighbor entries to determine when links to neighbor routers or the neighbor routers themselves have failed. Router k assumes that a neighbor router q from which no signaling has been received for LT seconds can be declared to have failed. If link (k,q) fails, then $w(k,q) \leftarrow \infty$, and for each destination d it sets $R^k(d,q) \leftarrow F$, $O^k(d,q) \leftarrow \infty$, $D^k(d,q) \leftarrow \infty$, and $L^k(d,q) \leftarrow F$. If a new link is established and router k is waiting for responses from neighbors for destination d, router k sets $R^k(d,q) \leftarrow T$ and sends a probe to q.

J. Example of THOR Operation with Negative Link Weights

We present an example that focuses on the operation of THOR for a given destination d in the presence of negative link weights. The weights of the links are assumed to be the same in both directions. The tuple $(O^v(d), D^v(d))$ is shown next to each router v in the example of Fig. 4. Preferred successors are indicated with solid arrowheads and other successors are shown with dashed arrowheads.

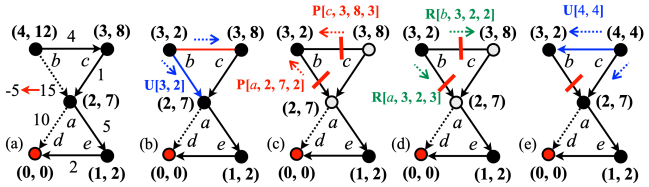

Fig. 4. Loop-free convergence in THOR with negative weights

An update from router v is indicated by $U[O^v(d), D^v(d)]$. A probe originated by o and send by v is indicated by $P[o, O^v(d), D^v(d), P^v(d)]$, and a response originated by o and send by v is indicated by $R[o, O^v(d), D^v(d), A^v(d)]$. Dashed arrowheads indicate the neighbors that receive updates, probes or responses from a router. For simplicity, it is assumed that routers operate synchronously, such that they process all input events received in a given step and send the resulting messages to neighbors after processing all the inputs.

Fig. 4(a) shows that link $w(a,b)$ changes from 15 to -5. As Fig. 4(b) indicates, this prompts router b to send an update reporting smaller values of $O^b(d)$ and $D^b(d)$. Importantly, we observe that b deletes c from $S^b(d)$ because $O^b(d) = O^b(d,c) = 3$. In turn, the update from b causes CSR not to be satisfied at routers a and c. As a result, as shown in Fig. 4(c), routers a and c become busy and send their probes to router b. We observe that both routers block the use of router b as a successor (indicated with a red bar).

As Fig. 4(d) shows, router b sends an invalid reply to a with $A^b(d) = O^b(d) = 3$ because $O^b(d) = 3 > 2 = O^b(d, a)$ and $D^b(d) = 2 < 7 = D^b(d, a)$, which causes a to continue blocking b as a successor. Router b sends a valid reply to c with $A^b(d) = 2 < 3 = P^b(d)$ because $P^b(d) > O^b(d) - 1$, which allows c to use b as a successor. As Fig. 4(e) shows, the replies from b allow routers a and c to become idle again and all routers converge to shortest distances without loops.

IV. Correctness of THOR

The proofs of the following theorems show that THOR ensures loop-free routing at every instant and converges to shortest paths within a finite time, without requiring positive link weights.

Theorem 3: THOR is loop-free at every instant.

Proof: It follows from Theorem 1 that router k cannot create a loop for destination d if it selects a neighbor v as a new successor according to CLS, i.e., if $O^k(d) > O^k(d, v)$. This is the case in THOR when a router is idle or busy. Furthermore, a router that transitions from idle to busy does not change $O^k(d)$ and does not change $O^k(d)$ while it remains busy. Therefore, the proof needs to show that a router k that transitions from busy to idle cannot create a loop when it selects a neighbor as its new preferred successor if it offers a loop-free path to destination d that has the shortest distance.

A router k transitions from busy to idle state only after it receives all the responses it requested from some of its neighbors. A response received by router k from neighbor n may be valid ($A^n(d) < P^k(d)$) or invalid ($A^n(d) \geq P^k(d)$). However, neighbor $i \in N^k$ is blocked from consideration as a successor by router k if it sends an invalid response to k, because then $L^k(d, i) = T$. Therefore, the rest of the proof needs to show that, when router k transitions from busy to idle, it cannot create a loop when it selects a new successor from the subset of neighbors that either sent valid responses or were not asked to send a response.

Case 1: v sent a valid response: A router r that originates a valid response to the probe from router k must be idle and must have a successor s such that $O^r(d, s) < P_{min}$, where $P_{min} \leq \mathcal{P}^k(d)$ because the probe from any router v forwarding the probe from router k must state $P^v(d) \leq P^k(d)$. Therefore, a valid response originated by a router r must state

$$A^r(d) = O^r(d, s) < P_{min} \leq P^k(d). \tag{7}$$

If router q is idle and receives a response from its successor s stating $A^s(d) < O^q(d)$, then it must forward the response stating $A^q(d) = A^s(d)$. If router q is busy and receives a response from neighbor n stating $A^n(d) < P^q(d)$, then it must forward the response stating $A^q(d) = A^n(d)$, unless it has already forwarded a prior copy of the response. Hence, it follows from Eq. (7) that, when router k receives a response from $v \in N^k$, the response results from the forwarding of the response originated by router r, and it must be true that $A^v(d) = A^r(d) < P_{min} \leq P^k(d)$.

The path traversed by the response received from neighbor v and originated by a router r is loop-free, and the anchoring reference $A^r(d)$ equals the number of hops along a loop-free path that exists or existed from router r to destination d. Accordingly, the ordering reference $O^k(d, v)$ reported in the response from router v to router k corresponds to the number of hops of a loop-free path $\mathcal{P}_{vd}$ that exists or existed from router v to destination d. Therefore, a router cannot create a loop if it chooses as a successor a neighbor that has sent a valid response.

Case 2: v was not asked to send a response: If router k enters the iterative state by sending a probe to only a subset of its neighbors, it must have a non-empty successor set $S^k(d)$ and any neighbor n that is asked to send a response is such that $[O^k(d, v) \geq O^k(d)] \wedge [D^k(d, v) + w(k, v) \leq D^k(d)]$.

Assume that router k sets $v = s^k(d)$ when it transitions to idle state. If $v \in S^k(d)$ when router k was busy, then no loop can be created because CLS was satisfied when v was chosen as a successor; therefore, v must be part of a loop-free path to destination d.

On the other hand, if $v \notin S^k(d)$ when router k was busy, then $D^k(d, v) + w(k, v) > D^k(d)$, for otherwise v would have been asked to send a response. Accordingly, router k cannot choose v as its preferred successor when it becomes idle because $S^k(d) \neq \emptyset$ and there is at least one neighbor in $S^k(d)$ that offers a loop-free route with a shorter distance and an ordering reference smaller than $O^k(d)$. Furthermore, router v cannot be in $S^k(d)$ when k becomes idle because $O^k(d, v) \geq O^k(d)$. Therefore, a router a cannot create a loop if it selects as its preferred successor a neighbor that was not asked to send a response. Given that no loops can occur in the above two cases, it follows that the theorem is true. ■

Lemma 1: In a network with a static topology, a busy router using THOR must become idle within a finite time after it receives one valid response to its probe.

Proof: Because the network has a static topology, N^k does not change for any given router k. A valid response to a probe sent by router k received from a neighbor q states $A^q(d) < P^k(d)$. According to THOR's operation, if router k receives the first valid response to its probe, it must set $R^k(d, q) = F$ and must send the response to all its neighbors. In turn, each neighbor $n \in N^k \setminus \{q\}$ must send a response stating $A^n(d) \leq A^q(d)$ after processing the response from k or processing a prior response with a smaller anchoring reference than $A^q(d)$. Router k must set $R^k(d, n) = F$ when it receives a valid response from $n \in N^k$. Therefore, router k must become idle within a finite time and the lemma is true. ■

Theorem 4: Routers using THOR become idle for all reachable destinations within a finite time after network changes stop occurring in a finite network.

Proof: Assume that all routers execute THOR correctly and that network changes stop occurring at time t, and let C denote the connected component of the network in which destination d is located at time t. A router knows which other routers are its neighbors within a finite time. Therefore, a finite time after time t each router in C is either idle or busy and

waiting for some of its neighbors to send responses, and knows that it is not a neighbor of destination d. The proof needs to show that any busy router must receive all the responses it requested within a finite time after time t.

Consider a probe originated by router $k \in C$ for destination d at time $t_k > t$. According to the operation of THOR, a probe can traverse either paths defined by the successor entries of idle routers that forward the probe, or paths defined by the incremental dissemination of the probe by busy routers that forward the probe to some of their neighbors.

Because $O^d(d) = 0$, destination d is always idle for itself, and it can send a valid response to any probe about itself. Furthermore, any neighbor router $n \in N^d$ must learn that d is one of its neighbors and receive an update from d stating $O^d(d) = 0$ within a finite time after time t. Given that any router $n \in N^d$ is at least one hop away from n, $O^n(d) \geq 1$, and hence one or more neighbors of d must have a one-hop path to d and become idle within a finite time after t, because the weights of their links to d constitutes their shortest distances to d. Hence, the probe from router k must state $P^k(d) \geq 1$ because $O^k(d) \geq 1$ after time t.

Assume that the probe from router k never reaches an idle router r with $O^r(d) < O^k(d)$, then there are no simple paths that can be traversed from k to r. This is a contradiction to three facts, namely: C is finite, simple paths in C defined by either successor entries or the incremental forwarding of the probe to neighbors at a relay router are finite, and $d \in C$ is always idle. Therefore, an idle router $r \in C$ must originate a response to the probe from router k within a finite time $t_r > t_k$. Because $O^r(d) < O^k(d)$ the response states $A^r(d) = O^r(d) - 1 < O^k(d)$.

The response from router r must propagate back to router k within a finite time $t_r^+ > t_r$ because all simple paths in C defined by the reverse paths traversed by the probe from k are finite, THOR is loop free and hence $A^r(d) < O^n(d)$ with n being a relay router between r and k, and the following is true according to the operation of THOR: (a) an idle router i that receives a response with $A^r(d)$ sends a response to *all* its neighbors if $A^r(d) < O^i(d)$, and (b) a busy router b that receives a response with $A^r(d)$ sends a a response to *all* its neighbors if $A^r(d)$ is smaller than the anchoring reference of the last response it sent while in the busy state.

Given that router k must receive the first valid response to its probe within a finite time t_r^+ and no topology changes occur in C after time $t < t_r^+$, it follows from Lemma 1 that router k must become idle within a finite time $t_i > t_r^+$ and the theorem is true. ■

Theorem 5: Each router using THOR converges to the shortest distances for all reachable destinations within a finite time after network changes stop occurring in a finite network.

Proof: The proof follows from Theorems 2 and 4. ■

Theorem 6: THOR converges to ∞ for all unreachable destinations within a finite time after network changes stop occurring in a finite network.

Proof: Let U be a connected component that does not include destination d starting at time t_0. Assume that THOR is executed correctly and that no network changes occur after time $t \geq t_0$. For the sake of contradiction, assume that all routers are idle, router $n_k \in U$ becomes idle and converges to $D^{n_k}(d) < \infty$ at time $t_k \geq t$, and no routers make any routing-table changes after that time. Because $D^{n_k}(d)[t_k] < \infty$, and because n_k is idle and no routers make any routing-table changes after time t_k, it must be the case that

$$\exists n_{k-1} \in N^{n_k} \Big(D^{n_k}(d, n_{k-1}) = D^{n_{k-1}}(d) < D^{n_k}(d) < \infty \Big)$$

Given that THOR is loop-free and routers are idle and stop making routing-table changes after time t_k, all routers in the loop-free path $P_{n_k n_o}$ have finite distances to d and the distance $D^{n_o}(d) < \infty$ is the smallest finite distance to d reported along the path. This is a contradiction to the assumption that THOR is executed correctly, because n_o must know its neighbors within a finite time and no router in U can perceive d as a neighbor after time t. ■

V. Performance Comparison

We compare THOR and OSPF using the ns-3 network simulator. We use our ns-3 implementation of THOR for the case of minimum-hop routing, which is RIPPLE [6] for networks with point-to-point links, and an off-the-shelf Quagga implementation of OSPF [13] integrated into ns-3 using the ns-3 direct code execution framework [19]. OSPF is run using its default RFC-compliant configuration. Transmissions are reliable across links. We focus on minimum-hop routing to show that THOR is far more efficient than OSPF even assuming simple link weights. Hence, each link has a cost of one unit. We only discuss the results for the failure of a single link or a single node.

Our comparison focuses on the time and overhead required by routing protocols to converge to minimum-hop paths after a link failure or a node failure. The time needed for convergence is measured from the time the input event is injected to the first time when the routing table at each router is stable and reflects its true shortest paths. Overhead is defined as the sum of the bytes of all routing messages sent between the time when network event occurs and the instant of convergence.

Our ns-3 implementation of THOR uses periodic messaging. A router maintains an update timer to ensure that it sends a routing message soon after it updates its routing table and sends routing messages often enough to inform its neighbors of its presence. If a router needs to send a routing message with updates, it does so after a minimum amount of time t_{min} has elapsed from the time it sent its prior routing message. In the absence of any changes in its routing table, a router sends a "hello" routing message, to update the lifetime entries maintained for itself by its neighbors no later than t_{max} seconds from the time it sent its last message. The timer t_{max} is shorter than a maximum lifetime LT. A router sets its update timer equal to t_{max} after sending a routing message, and sets it equal to t_{min} after preparing updates or queries to be sent in response to an input event. A router accumulates all the updates, probes and responses it needs to send in the next routing message and sends the updates when its update timer

TABLE I
CONVERGENCE TIME AND OVERHEAD AFTER A SINGLE FAILURE

			Link Failure		Node Failure	
nodes	average degree (diameter)	protocol	time	overhead	time	overhead
50	2 ($d = 3.7$)	THOR	7.16 ± 5.62 s	42171.40 ± 46487.25 B	5.65 ± 0.49 s	46872.20 ± 29019.98 B
50	2 ($d = 3.7$)	OSPF	38.20 ± 1.70 s	92782.00 ± 10178.89 B	37.45 ± 1.65 s	133585.60 ± 51342.24 B
50	3 ($d = 5.4$)	THOR	5.40 ± 0.60 s	32634.80 ± 8967.19 B	5.69 ± 0.52 s	58441.20 ± 29428.05 B
50	3 ($d = 5.4$)	OSPF	38.71 ± 1.76 s	161699.20 ± 23094.42 B	37.45 ± 1.65 s	339681.60 ± 204379.57 B
100	2 ($d = 3.8$)	THOR	6.22 ± 1.77 s	79463.60 ± 55608.84 B	5.80 ± 0.33 s	101875.00 ± 94634.04 B
100	2 ($d = 3.8$)	OSPF	37.87 ± 2.60 s	239337.20 ± 40972.66 B	36.15 ± 3.39 s	410580.80 ± 206001.02 B
100	3 ($d = 5.6$)	THOR	5.63 ± 0.91 s	67305.20 ± 29454.67 B	5.83 ± 0.46 s	104103.60 ± 24637.25 B
100	3 ($d = 5.6$)	OSPF	37.64 ± 2.56 s	412585.33 ± 55674.48 B	36.15 ± 3.39 s	725300.40 ± 380454.11 B

expires. THOR is configured to use $LT = 6$s, $t_{max} = 2$s and $t_{min} = 250$ms.

Table I presents the results of the simulation experiments. Each table entry represents the mean and standard deviations of 20 trials. Simulations are first run for a sufficiently long time such that all routing tables reflect steady state, and then the input event occurs. A single random link or node is removed once steady state is reached. We show results for random topologies with 50 and 100 routers with an average degree of two or three neighbors per router, which results in a different network diameter (d). All network links have a data rate of 1 Gbps and propagation delay of 1ms.

The results for THOR and OSPF after a node-failure should be expected when compared with the results shown for a link failure, given that a node failure can be considered as multiple link events related to a given node. THOR is more efficient in terms of convergence time and overhead, and provides multiple loop-free routes that need not have the same distance values. THOR converges more than six times faster than OSPF while incurring less than 25% signalling overhead on average over all networks.

The high efficiency of THOR for the case of minimum-hop routing results from the fact that probes, responses, and updates propagate only as needed, and routers use minimum-hop paths as soon as they are verified by the responses sent to answer probes.

VI. CONCLUSIONS

We presented THOR, a loop-free multipath routing algorithm that, to the best of our knowledge, is the first distributed routing algorithm that works correctly in the presence of negative link weights.

We proved that THOR is loop-free at every instant and converges to shortest paths within a finite time. The results of simulation experiments for the case of minimum-hop routing provide evidence that the approach used in THOR is more efficient than routing protocols used today.

There are several promising areas of future work related to THOR. For example, routing in very large networks with thousands of routers is unlikely to be based on "flat routing" and a hierarchical routing approach based on THOR would be needed. The performance of THOR when negative link weights exist was not addressed in this paper and should be analyzed. Lastly, its dynamic response [21] to one or multiple topology changes should be studied.

REFERENCES

[1] J. Behrens and J.J. Garcia-Luna-Aceves, "Hierarchical Routing Using Link Vectors," *Proc. IEEE INFOCOM '98*, 1998.

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

[3] 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.

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

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

[6] J.J. Garcia-Luna-Aceves and D. Cirimelli-Low, "Simple and Efficient Loop-Free Multipath Routing in Wireless Networks," *Proc. ACM MSWIM 2023*, Oct. 2023.

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

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

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

[10] 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.

[11] G. Malkin, "RIP Version 2" RFC 2453, 1998.

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

[13] J. Moy, "OSPF Version 2" RFC 2328, 1998.

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

[15] C. E. Perkins and E. M. Royer, "Ad-Hoc On-Demand Distance Vector Routing," *Proc. IEEE WMCSA '99*, 1999.

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

[17] Y. Rekhter and T. Li, "A Border Gateway Protocol 4 (BGP-4)," RFC 1771, March 1995.

[18] D. Savage et al., "Cisco's Enhanced Interior Gateway Routing Protocol (EIGRP)" RFC 7868, 2016.

[19] H. Tazaki et al., "Direct code execution: revisiting library OS architecture for reproducible network experiments," *Proc. CoNEXT '13*, Dec. 2013.

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

[21] W.T. Zaumen and J.J. Garcia-Luna-Aceves, "Dynamics of Distributed Shortest-Path Routing Algorithms," *Proc. ACM SIGCOMM '91*, Sept. 1991.

[22] 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.