Distributed Path Computation without Transient Loops: An Intermediate Variables Approach

Saikat Ray[1], Roch Guérin[1], and Rute Sofia[2]

[1] Department of Electrical and Systems Engineering, University of Pennsylvania
[2] Siemens AG Corporate Technology, Information and Communications, Munich, Germany*

Abstract. Paths with loops, even transient ones, pose significant stability problems in networks. As a result, much effort has been devoted over the past thirty years to designing distributed algorithms capable of avoiding loops. We present a new algorithm, D*istributed Path Computation with* I*ntermediate* V*ariables* (DIV), that guarantees that no loops, transient or steady-state, can ever form. DIV's novelty is in that it is *not* restricted to shortest paths, can easily handle arbitrary sequences of changes and updates, and provably outperforms earlier approaches in several key metrics. In addition, when used with distance-vector style path computation algorithms, DIV also prevents *counting-to-infinity*; hence further improving convergence. The paper introduces DIV and its key properties. Simulation quantifying its performance gains are also presented.

1 Introduction

Distributed path computation is a central problem in modern communication networks, and has therefore received much attention. Its importance together with the lack of a "generic" solution is what motivated this paper. In distributed path computations, end-to-end paths are formed by concatenating individual node decisions, where for each destination a node chooses one or more successors (next-hop) based only on local information and so as to optimize some global objective function. The use of inconsistent information across nodes can then have dire consequences, including possible formation of transient routing loops[1]. Loops can severely impact performance, especially in networks with no or limited loop mitigation mechanisms (such as Time-to-Live (TTL)), where a routing loop often triggers network-wide congestion. The importance of avoiding transient routing loops remains a key requirement for path computation in both existing and emerging network technologies, e.g., see [1] for recent discussions, and is present to different extents in both link-state and distance-vector algorithms.

Link-state algorithms (e.g., OSPF [2]) decouple information dissemination and path computation, so that routing loops, if any, are short-lived, but the algorithms overhead is high in terms of communication (broadcasting updates), storage (maintaining a full network map), and computation (changes anywhere in the network trigger computations at all nodes). By combining information dissemination and path computation, distance-vector algorithms (cf. RIP [3], EIGRP [4]) avoid several of these disadvantages, which make them attractive, especially in situations of frequent local topology changes and/or when high control overhead is undesirable. However, they can suffer from frequent and

* Research supported in part by a gift to the University of Pennsylvania by the Siemens AG Corporate Technology, Information and Communications, Munich, Germany.

[1] By "routing" we mean creation of forwarding tables, irrespective of their "layer,"e.g., 2 or 3.

long lasting routing loops and slower convergence (cf. the *counting-to-infinity* problem [5]). Thus, making distance-vector based solutions attractive, calls for overcoming these problems. Since the 70's, several works [6–10] have targeted this goal in the context of shortest path computations (cf. Section 2), but the problem remains fundamental and timely (e.g., see the efforts for introducing distributed shortest path algorithms in lieu of a distributed spanning tree algorithm [11]). Furthermore, as we briefly allude to in Section 3.4, extending this capability to other types of path computation is also becoming increasingly important.

In this paper, we introduce the D*istributed Path Computation with* I*ntermediate* V*ariables* (DIV) algorithm which enables generic, distributed, light-weight, loop-free path computation. DIV is *not* by itself a routing protocol; rather it can run on top of any routing algorithm to provide loop-freedom. DIV generalizes the *Loop Free Invariant* (LFI) based algorithms [9, 10] and outperforms previous solutions including known LFI and *Diffusing Computation* based algorithms, such as the *Diffusing Update Algorithm* [8][2]:

1. *Applicability*: DIV is not tied to shortest path computations. It can be integrated with other distributed path computation algorithms, e.g., the link-reversal mechanisms of [12, 13] or algorithms targeting path redundancy or distributed control as outlined in Section 3.4.
2. *Frequency of Synchronous Updates*: When applied to shortest path computations, DIV triggers synchronous updates less frequently as well as reduces the propagation of synchronous updates (cf. Theorem 4), where synchronous updates are updates that potentially must propagate to all upstream[3] nodes before the originator is in a position to update its path. They are time and resource consuming. Thus, the less frequent the synchronous updates, the better the algorithm.
3. *Maintaining a path*: A node can potentially switch to a new successor without forming a loop provably faster with DIV than with earlier algorithms (cf. Section 3.2). This is especially useful when the original path is lost due to a link failure.
4. *Convergence Time*: When a node receives multiple overlapping updates[4] from neighbors, DIV allows the node to process them in an arbitrary manner. Thus, in DIV, the node can respond only to the latest or the best update, converging potentially faster. (cf. Theorem 2).
5. *Robustness*: DIV can tolerate arbitrary packet reordering and losses without sacrificing correctness (cf. Theorem 3).

Finally, the rules and update mechanism of DIV and their correctness proofs are rather simple, which hopefully will facilitate correct and efficient implementations.

---

[2] The authors gratefully acknowledge J.J. Garcia-Luna-Aceves for introducing them to the LFI-based algorithms.

[3] Upstream nodes of a node x for destination z are the nodes whose path to z includes x.

[4] Two updates are overlapping if the latter appears before the algorithm has converged in response to the first.

2 Previous Works

The Common Structure The primary challenge in avoiding transient loops lies in handling inconsistencies in the information stored across nodes. Otherwise, simple approaches can guarantee loop-free operations at each step [12, 14]. Most previous distance-vector type algorithms free from transient loops and convergence problems follow a common structure: Nodes exchange update-messages to notify their neighbors of any change in their own cost-to-destination (for any destination). If the cost-to-destination decreases at a node, the algorithms allow updating its neighbors in an arbitrary manner; these updates are called *local* (asynchronous) updates. However, following an increase in the cost-to-destination of a node, these algorithms require that the node potentially update all its upstream nodes before changing its current successor; these are *synchronous* updates.

The algorithm proposed in [6] follows the above broad structure and is one of the earliest work guaranteeing loop-free operations with inconsistent information. For handling multiple overlapping updates, it relies on unbounded sequence numbers that mark update epochs. An improvement to this algorithm is presented in [7], which handles multiple overlapping updates by maintaining *bit vectors* at each node.

Diffusing Update Algorithm (DUAL) DUAL, a part of CISCO's widely used EIGRP protocol, is perhaps the best known algorithm. In DUAL, each node maintains, for each destination, a set of neighbors called the *feasible successor set*. The feasible successor set is computed using a *feasibility condition* involving *feasible distances* at a node. Several feasibility conditions are proposed in [8] that are all tightly coupled to the computation of a shortest path. For example, the *Source Node Condition* (SNC) defines the feasible successor set to be the set of all neighbors whose current cost-to-destination is *less* than the minimum cost-to-destination seen so far by the node. A node can choose any neighbor in the feasible successor set as the successor (next-hop) without having to notify any of its neighbors and without causing a routing loop regardless of how other nodes in the network choose their successors, as long as they also comply with this rule.

If the neighbor through which the cost-to-destination of the node is minimum is in the feasible successor set, then that neighbor is chosen as the successor. If the current feasible successor set is empty of does not include the best successor, the node initiates a synchronous update procedure, known as a *diffusing computation* (cf. [15]), by sending *queries* to all its neighbors and waiting for acknowledgment before changing its successor. Multiple overlapping updates—i.e., if a new link-cost change occurs when a node is waiting for replies to a previous query—are handled using a *finite state machine* to process these multiple updates sequentially.

***Loop Free Invariance* (LFI) Algorithms** A pair of invariances, based on the cost-to-destination of a node and its neighbors, called *Loop Free Invariances* (LFI) are introduced in [9] and it is shown that if nodes maintain these invariances, then no transient loops can form (cf. Section 3.2). Update mechanisms are required to maintain the LFI conditions: [9] introduces *Multiple-path Partial-topology Dissemination Algorithm* (MPDA) that uses a link-state type approach whereas [10] introduces *Multipath Distance Vector Algorithm* (MDVA) that uses a distance vector type approach. Similar to DUAL, MDVA uses a diffusing update approach to increase its cost-to-destination,

thus it also handles multiple overlapping cost-changes sequentially. The primary contribution of LFI based algorithms such as MDVA or MPDA is a unified framework applicable to both link-state and distance-vector type approaches and multipath routing.

Comparative Merits of Previous Algorithms DUAL supersedes the other algorithms in terms of performance. Specifically, the invariances of MPDA and MDVA are based directly on the cost of the shortest path. Thus, every increase in the cost of the shortest path triggers synchronous updates in MDVA or MPDA. In constrast, the feasibility conditions of DUAL are *indirectly* based on the cost of the shortest path. Consequently, an increase in the cost of the shortest path may not violate the feasibility condition of DUAL, and therefore may not trigger synchronized updates—an important advantage over MDVA or MPDA. Because of the importance of this metric, we consider DUAL the benchmark against which to compare DIV (cf. Section 4).

DIV combines advantages of both DUAL and LFI. DIV generalizes the LFI conditions, is not restricted to shortest path computations and, as LFI-based algorithms, allows for multipath routing. In addition, DIV allows for using a feasibility condition that is strictly more relaxed than that of DUAL, hence triggering synchronous updates less frequently than DUAL (and consequently, than MPDA or MDVA) as well as limiting the propagation of any triggered synchronous updates. The update mechanism of DIV is simple and substantially different from that of previous algorithms, and allows arbitrary packet reordering/losses. Last but not least, unlike DUAL or LFI algorithms, DIV handles multiple overlapping cost-changes *simultaneously* without additional efforts resulting in simpler implementation and potentially faster convergence.

3 DIV

3.1 Overview

DIV lays down a set of rules on existing path computation algorithms to ensure their loop-free operation at each instant. This rule-set is not predicated on shortest path computation, so DIV can be used with other path computation algorithms as well.

For each destination, DIV assigns a *value* to each node in the network. To simplify our discussion and notation, we fix a particular destination and speak of *the* value of a node. The values could be arbitrary—hence the independence of DIV from any underlying path computation algorithm. However, usually the value of a node will be related to the underlying objective function that the algorithm attempts to optimize and the network topology. Some typical value assignments include: (i) in shortest path computations, the value of a node could be its cost-to-destination; (ii) as in DUAL, the value could be the minimum cost-to-destination seen by the node from time $t = 0$; (iii) as in TORA [13], the value could be the *height* of the node; etc.

As in previous algorithms, the basic idea of DIV is to allow a node to choose a neighbor as successor only if the value of that neighbor is less than its own value: this is called the *decreasing value property* of DIV, which ensures that routing loop can never form. The hard part is enforcing the decreasing value property when network topology changes. Node values must be updated in response to changes to enable efficient path selection. However, how does a node know the *current* value of its neighbors to maintain

the decreasing value property? Clearly, nodes update each other about their own current value through update messages. Since update messages are asynchronous, information at various nodes may be inconsistent, which may lead to the formation of loops. This is where the non-triviality of DIV lies: it lays down specific update rules that guarantee that loops are never formed even if the information across nodes is inconsistent.

3.2 Description of DIV

There are four aspects to DIV: (i) the variables stored at the nodes, (ii) two ordering invariances that each node maintains, (iii) the rules for updating the variables, and (iv) two semantics for handling non-ideal message deliveries (such as packet loss or re-ordering). A separate instance of DIV is run for each destination, and we focus on a particular destination.

The Intermediate Variables Suppose that a node x is a neighbor of node y. These two nodes maintain intermediate variables to track the value of each other. There are three aspects of each of these variables: whose value is this? who believes in that value? and where is it stored? Accordingly, we define $V(x; y|x)$ to be the value of node x as known (believed) by node y stored in node x; similarly $V(y; x|x)$ denotes value of node y as known by node x stored in node x.

Thus, node x with n neighbors, $\{y_1, y_2, \ldots, y_n\}$, stores, for each destination:

1. its own value, $V(x; x|x)$;
2. the values of its neighbors as known to itself, $V(y_i; x|x)$ [$y_i \in \{y_1, y_2, \ldots, y_n\}$],
3. and the value of itself as known to its neighbors $V(x; y_i|x)$ [$y_i \in \{y_1, y_2, \ldots, y_n\}$].

That is, $2n + 1$ values for each destination. The variables $V(y_i; x|x)$ and $V(x; y_i|x)$ are called intermediate variables since they endeavor to reflect the values $V(y_i; y_i|y_i)$ and $V(x; x|x)$, respectively. In steady state, DIV ensures that $V(x; x|x) = V(x; y_i|x) = V(x; y_i|y_i)$.

The Invariances DIV requires each node to maintain at all times the following two invariances based on its set of *locally stored variables*.

Invariance 1 *The value of a node is not allowed to be more than the value the node thinks is known to its neighbors. That is,*

$$V(x; x|x) \leq V(x; y_i|x) \text{ for each neighbor } y_i. \tag{1}$$

Invariance 2 *A node x can choose one of its neighbors y as a successor only if the value of y is less than the value of x* as known by node x*; i.e., if node y is the successor of node x, then*

$$V(x; x|x) > V(y; x|x). \tag{2}$$

Thus, due to Invariance 2, a node x can choose a successor only from its *feasible successor set* $\{y_i|V(x; x|x) > V(y_i; x|x)\}$. The two invariances reduces to the LFI conditions if the value of a node is chosen to be its current cost-to-destination.

Update Messages and Corresponding Rules There are two operations that a node needs to perform in response to network changes: (i) decreasing its value and (ii) increasing its value. Both operations need notifying neighboring nodes about the new value of the node. DIV uses two corresponding update messages, Update::Dec and Update::Inc, and acknowledgment (ACK) messages in response to Update::Inc (no ACKs are needed for Update::Dec). Both Update::Dec and Update::Inc contain the new value (the destination), and a sequence number[5]. The ACKs contain the sequence number and the value (and the destination) of the corresponding Update::Inc message. DIV lays down precise rules for exchanging and handling these messages which we now describe.

Decreasing Value Decreasing value is the simpler operation among the two. The following rules are used to decrease the value of a node x to a new value V_0:

- Node x first simultaneously decreases the variables $V(x; x|x)$ and the values $V(x; y_i|x)$ $\forall i = 1, 2, \ldots, n$, to V_0,
- Node x then sends an Update::Dec message to all its neighbors that contains the new value V_0.
- Each neighbor y_i of x that receives an Update::Dec message containing V_0 as the new value updates $V(x; y_i|y_i)$ to V_0.

Increasing Value In the decrease operation a node first decreases its value and then notifies its neighbors; in the increase operation, a node first notifies its neighbors (and wait for their acknowledgments) and then increases its value. In particular, a node x uses the following rules to increase its value to V_1:

- Node x first sends an Update::Inc message to all its neighbors.
- Each neighbor y_i of x that receives an Update::Inc message sends an acknowledgment (ACK) when able to do so according to the rules explained in details below (Section 3.2). When y_i is ready to send the ACK, it first modifies $V(x; y_i|y_i)$, changes successor if necessary (since the feasible successor set may change), and then sends the ACK to x; the ACK contains the sequence number of the corresponding Update::Inc message and the new value of $V(x; y_i|y_i)$. Note that it is essential that node y_i changes successor, if necessary, *before* sending the ACK.
- When node x receives an ACK from its neighbor y_i, it modifies $V(x; y_i|x)$ to V_1. At any time, node x can choose any value $V(x; x|x) \leq V(x; y_i|x) \forall i = 1, 2, \ldots, n$.

Rules for Sending Acknowledgment Suppose node y_i received an Update::Inc message from node x. Recall that node y_i must increase $V(x; y_i|y_i)$ before sending an ACK. However, increasing $V(x; y_i|y_i)$ may remove node x from the feasible successor set at node y_i. If node x is the only node in the feasible successor set of node y_i, node y_i may lose its path if $V(x; y_i|y_i)$ is increased without first increasing $V(y_i; y_i|y_i)$. Node y_i then has two options: (i) first increase $V(y_i; y_i|y_i)$, increase $V(x; y_i|y_i)$, and then send the ACK to node x; or (ii) increase $V(x; y_i|y_i)$, send ACK to node x, and then increase $V(y_i; y_i|y_i)$. We call option (i) the *normal mode*, and option (ii) the *alternate mode*.

In the normal mode, node y_i keeps the old path while it awaits ACKs from its neighbors before increasing $V(y_i; y_i|y_i)$, since it keeps x in the feasible successor set until

[5] For simplicity, sequence numbers are assumed large enough so that rollover is not an issue.

then. Thus the update request propagates to upstream nodes in the same manner as in DUAL and other previous works. However, note that DIV allows a node to respond with an ACK in response to Update::Inc messages sent its neighbors, if there is any, without the fear of any loop creation. This guarantees that DIV never enters any deadlock situations.

In the alternate mode, node y_i may have no successor for a period of time (until it is allowed to increase its value). At a first glance, it may seems unwise to use the alternate mode. However, note that if node x originated the value-increase request in the first place because the link to its successor was *down* (as opposed to only a finite cost change), then the old path does not exist and the normal mode has no advantage over the alternate mode in terms of maintaining *a* path. In fact, in the alternate mode, the downstream nodes get ACKs from their neighbors more quickly and thus can switch earlier to a new successor (which hopefully has a valid path) than in the normal mode.

Semantics for Handling Message Reordering We maintain the following two semantics that account for non-zero delays between origination of a message at the sender and its reception at the receiver and possible reordering of messages and ACKs.

Semantic 1 *A node ignores an update message that comes out-of-order (i.e., after a message that was sent earlier).*

Semantic 2 *A node ignores outstanding ACKs after issuing an Update::Dec message.*

These semantics are enforced using the embedded sequence numbers in update messages (an ACK includes the sequence number of the Update::Inc that triggered it).

3.3 Properties of DIV

The two main properties of DIV are: (i) it prevents loops at every instant, and (ii) it prevents counting-to-infinity in the normal mode. Due to space constraint, we only prove Property (i) (see [16] for details). Note that even in the specific case of shortest path computations where the value of a node is set to its current cost-to-destination, these properties of DIV cannot be deduced from those for the LFI conditions since DIV operates without any assumption on packet reordering, delay or losses.

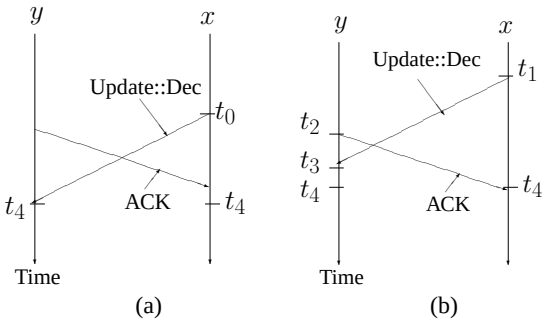

Fig. 1. Two cases of possible message exchanges between two neighboring nodes which would violate Eq. (3). Both cases are shown to be contradictory.

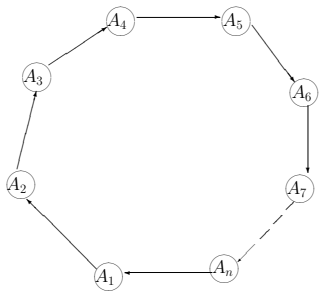

Fig. 2. A possible loop in the successor graph.

Loop-free Operation at Every Instant The following is the key proposition based on which our result follows.

Proposition 1 *For any two neighboring nodes x and y, we always have*

$$V(x; y|x) \leq V(x; y|y) \tag{3}$$

Proof. The proof is by contradiction. Suppose at time $t = 0$ condition (3) is satisfied and at time $t = t_4$ condition (3) is violated for the first time. I.e., at time $t = t_4$, we have $V(x; y|x) = V_1$ and $V(x; y|y) = V_0$ with $V_1 > V_0$. Thus, at time t_4 either $V(x; y|y)$ decreases or $V(x; y|x)$ increases. We consider these two cases separately.

Case (i): $V(x; y|y)$ decreases at time t_4 to V_0. Thus node y receives an Update::Dec message from node x at time t_4. As shown in Fig. 1(a), suppose that this message originated at node x at time t_0. Therefore, at time t_0, we have $V(x; y|x) = V_0$. But as per our assumption, $V(x; y|x) = V_1 > V_0$ at time t_4. Thus, node x must receive an ACK from node y that increases $V(x; y|x)$ during the period (t_0, t_4) (cf. Fig. 1(a)). Suppose t_2 denotes the time when node x sent the update message that triggered this ACK. We then have two cases:

- $t_2 < t_0 < t_4$: In this case, the Update::Inc message that triggered the ACK was outstanding at t_0; the time when node x sent an Update::Dec message. Thus node x would disregard this ACK due to Semantic 2, and therefore not increase $V(x; y|x)$.
- $t_0 < t_2 < t_4$: In this case, the Update::Inc message that triggered the ACK was sent by node x after the Update::Dec message, but node y received the Update::Inc message before the Update::Dec message; i.e., the Update::Dec message arrived node y out of order and thus node y would disregard the Update::Dec message due to Semantic 1, and therefore not decrease $V(x; y|y)$.

We therefore have a contradiction in both the cases.

Case (ii): $V(x; y|x)$ increases at time t_4 to V_1. Thus node x receives an ACK from node y at time t_4. As shown in Fig. 1(b), suppose that this ACK originated at node y at time t_2. Thus, we have $V(x; y|y) = V_1$ at time t_2. But by assumption, $V(x; y|y) = V_0$ at time t_4. Thus, node y must receive an Update::Dec message during the period (t_2, t_4), say at time t_3. Suppose that node x originated this Update::Dec message at time t_1 (cf. Fig. 1(b)). Moreover, suppose node x originated at time t_0 the Update::Inc message that triggered the ACK it receives from node y at time t_4. Then, there are two possibilities:

- $t_0 < t_1 < t_4$: In this case, the Update::Inc message that triggered the ACK was outstanding at t_1; the time when node x sent an Update::Dec message. Thus node x would disregard this ACK due to Semantic 2, and not increase $V(x; y|x)$ to V_1 at time t_4.
- $t_1 < t_0 < t_4$: In this case, the Update::Inc message that triggered the ACK was sent by node x after the Update::Dec message, but node y received the Update::Inc message before the Update::Dec message; i.e., the Update::Dec message arrived node y out of order and thus node y would disregard the Update::Dec message due to Semantic 1, and not decrease $V(x; y|y)$ to V_0 at time t_3.

We therefore again have a contradiction in both the cases.

Thus we have shown that both case (i) and case (ii) lead to contradictions. Hence, we conclude that it is not possible to violate Eq. (3). □

Theorem 1 *The successor graph created following DIV's update algorithm is an acyclic graph at each instant.*

Proof. The proof is again by contradiction. Suppose at some instant of time there is a loop in the successor graph, as shown in Fig. 2. Since the number of nodes in this loop is finite, there is a node in this loop whose value is smaller than or equal to the value of its successor. Without any loss of generality, let A_n be this node and let A_1 be its successor. Thus,

$$V(A_1; A_1|A_1) \geq V(A_n; A_n|A_n). \tag{4}$$

But since node A_1 maintains the first invariance, we have

$$V(A_1; A_1|A_1) \leq V(A_1; A_n|A_1). \tag{5}$$

Also since node A_n maintains the second invariance, we have

$$V(A_n; A_n|A_n) > V(A_1; A_n|A_n). \tag{6}$$

But equations (4), (5) and (6) together imply that $V(A_1; A_n|A_1) > V(A_1; A_n|A_n)$, which contradicts Proposition 1. □

Multiple Overlapping Updates and Packet Losses Unlike earlier algorithms [6–10], DIV can handle multiple updates without additional efforts. A node can send multiple Update::Inc or Update::Dec messages in any order; a neighbor can hold on to sending an ACK for an arbitrary time—e.g., use a hold-down timer—and when replying with an ACK, it can choose to respond to only a subset of pending updates—even just one; none of these actions results in routing loops. This is because these handling f multiple overlapping updates still preserve the Semantics and the Invariances. Semantics are satisfied at each node using the sequence numbers of updates, and invariances depend only on the locally stored variables. Thus they are never violated. We summarize this important property in the following theorem, which establishes the tremendous flexibility DIV gives in choosing policies for replying with ACKs to optimize different criteria.

Theorem 2 *The correctness of DIV remains valid under arbitrary policies for handling multiple overlapping updates.*

DIV can also handle an arbitrary sequence of lost packets without jeopardizing its correctness. If an Update::Dec message sent by node x to neighbor y is lost, then $V(x; y|x)$ is lowered (by x), but not $V(x; y|y)$; i.e., we have $V(x; y|x) < V(x; y|y)$. But this still satisfies Proposition 1, hence does not affect DIV's correctness. If an Update::Inc message sent by node x to neighbor y is lost, then node x cannot increase its value, but the invariances remains valid. Finally, if an ACK is lost, then $V(x; y|y)$ is increased (by y), but not $V(x; y|x)$; i.e., we have $V(x; y|x) < V(x; y|y)$. Again, this satisfies Proposition 1 and DIV remains correct. When combined with the fact that Semantics 1 and 2 handle arbitrary reordering and delay of messages, this leads to the following important property of DIV:

Theorem 3 *The correctness of DIV remains valid under arbitrary sequence of loss, reordering or delay of messages.*

Frequency of Synchronous Updates: A Comparison with DUAL

Claim. Suppose x and y are neighbors. If SNC is true at x through y, then with DIV x can choose y as a successor.

Proof. We need to show that SNC is true at x through y implies $V(x;x|x) > V(y;y|y)$. From the definition of SNC (cf. Section 2), since SNC is satisfied, we have the minimum cost-to-destination of x, $V(x;x|x)$, is more than the *current* cost-to-destination of y. However, the current cost-to-destination of y is clearly as large as the minimum cost-to-destination of y, $V(y;y|y)$; i.e., $V(x;x|x) > V(y;y|y)$. □

However, the other direction is clearly not true. Suppose $V(x;x|x) = 2$, $V(y;y|y) = 1$ and the current cost-to-destination of y is 3. Then SNC is not satisfied, but with DIV, x can still choose y as its successor. Since the condition of DIV is strictly more relaxed than SNC, and a synchronous update is issued only when the condition of DIV (or SNC for DUAL) is not satisfied, we have

Theorem 4 *DIV issues synchronous updates less frequently than DUAL under SNC.*

Note that this cannot be remedied simply by replacing SNC in DUAL with the conditions of DIV since without DIV's update mechanisms, these are not sufficient to guarantee loop-free operation.

Theorem 4 is stated in terms of SNC as it is the most common condition used in practice (e.g., it is used in EIGRP). However, the theorem remains true if SNC is replaced by other conditions of DUAL, such as CSC or DIC [8]; the proof is similar.

3.4 Other Applications of DIV

By decoupling loop-freedom from the path computation metric (e.g., shortest-path), DIV opens up new possibilities. Due to space limitations, we only mention two such applications; these are discussed further in [16].

Redundant-Path Routing An assignment of values (along with the Invariances) induces an acyclic successor graph (i.e., a routing), and if each node other than the destination has at least one outgoing link, the destination is reachable from every node. By an appropriate choice of values, DIV can be used to maximize a measure of the multitude of paths to the destination. This can be appealing when bandwidth is cheap and reliability takes a higher priority.

Distributed Vehicular Formation Forming rigid patterns, e.g., of unmanned aerial vehicles, using localized sensing is an important problem. It is known that stabilizing the pattern is easy if the underlying *formation graph* is acyclic [17]. DIV, especially its alternate mode, can be used to ensure acyclicity of the formation graph in a distributed fashion under dynamic environments.

Nodes	10	20	30	40	50
$T_{\text{loop}}(s)$	2.282	2.456	2.344	2.702	2.108
Conf. (s)	0.259	0.365	0.259	0.391	0.276
Nodes	60	70	80	90	—
$T_{\text{loop}}(s)$	2.126	2.339	2.290	2.354	—
Conf. (s)	0.237	0.273	0.311	0.250	—

Fig. 3. Average loop-retention time, T_{loop}, in seconds.

Nodes	10	20	30	40	50
Fraction	0.717	0.784	0.823	0.843	0.846
Nodes	60	70	80	90	—
Fraction	0.832	0.843	0.846	0.840	—

Fig. 4. Fraction of times DIV is satisfied given that SNC is not.

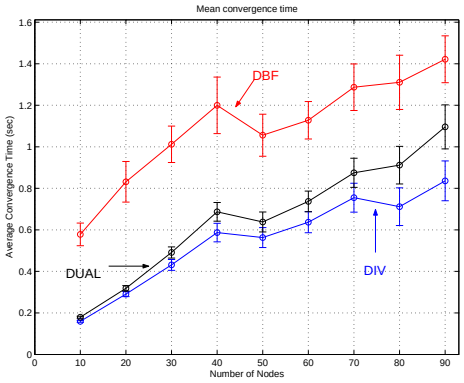

Fig. 5. Mean convergence time.

4 Performance Evaluation

This section presents simulation results comparing the performances of DIV (with normal mode used with DBF to compute shortest paths) in terms of routing loops, convergence times and frequency of synchronous updates against DUAL (cf. Section 2). The performance of DBF without DIV is also presented as a reference. The simulations are performed on random graphs with fixed average degree of 5. The number of nodes are varied from 10 to 90 in increments of 10. For each graph-size, 100 random graphs are generated. Link costs are drawn from a bi-modal distribution: with probability 0.5 a link cost is uniformly distributed in [0,1]; and with probability 0.5 it is uniformly distributed in [0,100]. For each graph, 100 random link-cost changes are introduced, again drawn from the same bi-modal distribution. All three algorithms are run on the same graphs and sequence of changes. Processing time of each message is random: it is 2 s with probability 0.0001, 200 ms with probability 0.05, and 10 ms otherwise.

The table in Fig. 3 shows the average loop-retention time in seconds, T_{loop}—the time from when a routing loop is detected to when it eventually subsides—given that a loop is formed, as the size of the graphs varies. As expected, no loops were found with DUAL or DIV, so the table only shows results for DBF, which illustrate that without loop-prevention mechanisms, loops can be retained for a significant time.

Fig. 5 shows average convergence times—the time from a cost change to when no more updates are exchanged—for all three algorithms as the size of the graphs varies. The vertical bars show standard deviations. Both DIV and DUAL converge faster than DBF; however, DIV performs better, especially for larger graphs. This is because DIV's conditions are satisfied more easily, so that synchronous updates often complete earlier (recall that a node with a feasible neighbor replies immediately). This is supported by the table in Fig. 4, which shows the fraction of times the condition of DIV is satisfied given that SNC is not satisfied; this fraction exceeds 80% for larger graphs.

5 Conclusion

Distance-vector path computation algorithms are attractive candidates not only for shortest path computations, but also in several important areas involving distributed path

computations due to their simplicity and scalability. Leveraging those benefits, however, calls for eliminating several classical drawbacks such as transient loops and slow convergence. The algorithm proposed in this paper, DIV, meets these goals, and which unlike earlier solutions is not limited to shortest path computations. In addition, even in the context of shortest path computations, DIV outperforms earlier approaches in several key performance metrics, while also providing greater operational flexibility, e.g., in handling lost or out-of-order messages. Given these many benefits and the continued and growing importance of distributed path computations, we believe that DIV can play an important role in improving and enabling efficient distributed path computations.

References

1. P. Francois, C. Filsfils, J. Evans, and O. Bonaventure, "Achieving sub-second IGP convergence in large IP networks," *ACM SIGCOMM Computer Communication Review*, July 2005.
2. J. Moy, "OSPF version 2," Internet Engineering Task Force, RFC 2328, Apr. 1998. [Online]. Available: http://www.rfc-editor.org/rfc/rfc2328.txt
3. G. Malkin, "RIP version 2," Internet Engineering Task Force, RFC 2453, Nov. 1998. [Online]. Available: http://www.rfc-editor.org/rfc/rfc2453.txt
4. R. Albrightson, J. J. Garcia-Luna-Aceves, and J. Boyle, "EIGRP–A fast routing protocol based on distance vectors," in *Proceedings of Network/Interop*, Las Vegas, NV, May 1994.
5. D. Bertsekas and R. Gallager, *Data Networks*, 2nd ed. Prentice Hall, 1991.
6. P. M. Merlin and A. Segall, "A failsafe distributed routing protocol," *IEEE Transactions on Communications*, vol. COM-27, no. 9, pp. 1280–1288, September 1979.
7. J. M. Jaffe and F. M. Moss, "A responsive routing algorithm for computer networks," *IEEE Transactions on Communications*, vol. COM-30, no. 7, pp. 1768–1762, July 1982.
8. J. J. Garcia-Lunes-Aceves, "Loop-free routing using diffusing computations," *IEEE/ACM Transactions on Networking*, vol. 1, no. 1, pp. 130–141, February 1993.
9. S. Vutukury and J. J. Garcia-Luna-Aceves, "A simple approximation to minimum-delay routing," in *Proceedings of ACM SIGCOMM*, Cambridge, MA, September 1999.
10. ——, "MDVA: A distance-vector multipath routing protocol," in *Proceedings of IEEE INFOCOM*, Anchorage, AK, April 2001.
11. K. Elmeleegy, A. L. Cox, and T. S. E. Ng, "On count-to-infinity induced forwarding loops in Ethernet networks," in *Proceedings of IEEE INFOCOM*, Barcelona, Spain, April 2006.
12. E. Gafni and D. Bertsekas, "Distributed algorithms for generating loop-free routes in networks with frequently changing topology," *IEEE/ACM Transactions on Communications*, January 1981.
13. V. D. Park and M. S. Corson, "A highly adaptive distributed routing algorithm for mobile wireless networks," in *Proceedings of IEEE INFOCOM*, 1997. [Online]. Available: citeseer.ifi.unizh.ch/park97highly.html
14. R. G. Gallager, "A minimum delay routing algorithm using distributed computation," *IEEE Transactions on Communications*, January 1977.
15. E. W. Dijkstra and C. S. Scholten, "Termination detection for diffusing computations," *Information Processing Letters*, vol. 11, no. 1, pp. 1–4, August 1980.
16. S. Ray, R. Guérin, and S. Rute, "Distributed path computation without transient loops: An intermediate variables approach," University of Pennsylvania, Tech. Rep., 2006. [Online]. Available: http://www.seas.upenn.edu/~saikat/loopfree.pdf
17. J. Baillieul and A. Suri, "Information patterns and hedging Brockett's theorem in controlling vehicle formations," in *Conference on Decision and Control*, Maui, Hawaii, December 2003.