ISSN:0975-9646

Continuously Distributed Path Estimation by Using DIV

K. Anusha[#] R. Lakshmi Tulasi[*]
[#]*Department of Computer Science & engineering*
QISCET, India
[*]*Professor& HOD, Dept of CSE*
QISCET, India

***Abstract*— Paths with loops, even transient ones ,can pose significant stability problems in networks. Some earlier approaches like Shortest path routing (Dijkstra),Flooding, Flow-based routing, Distance vector routing(OSPF),Link state routing(Bellmen-Ford),Hierarchical routing, Broadcast routing, Multicast routing have problems maintaining the balance between node delays and link delays. We present a new algorithm, Distributed Path Computation with Intermediate Variables (DIV) that guarantees that no loops, transient or steady-state, can never downgrade network dynamics. Besides its ability to operate with existing distributed routing algorithms to guarantee that the directed graph induced by the routing decisions stays acyclic by handling multiple overlapping updates and packet losses and frequency of synchronous updates, and provably outperforms earlier approaches in several key metrics. In addition, when used with distance-vector style path computation algorithms, the main drawbacks of Distance Vector are limited scalability due to slow convergence time, bandwidth consumption and routing loops. DIV also prevents counting-to-infinity; hence further improving convergence. Simulation quantifying its performance gains is also presented.**

***Keywords*— DIV, Distance-vector routing, loop-free routing, flow-based routing**

I. INTRODUCTION

The multiple autonomous computers that communicate through computers is called Distributed computing.In this, the systems interact with each other to reach the destination. The computer program that runs in the distributed program and distributed programming. This will solve the computational problems which refer to Distributed computing. Many routing protocols have been proposed for MANETs, *e.g.*, DSDV and OLSR. In both approaches, nodes choose successor (next-hop) nodes for each destination based only on local information, with the objective that the chosen paths to the destination be *efficient* in an appropriate sense—*e.g.*, having the minimum cost.

Inconsistent information at different nodes can have dire consequences that extend beyond not achieving the desired efficiency. Of particular significance is the possible formation of transient routing loops,1 which can severely impact network performance, especially in networks with no or limited loop mitigation mechanisms, e.g., no Time-to-Live (TTL) field in packet headers or a TTL set to a large value. 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

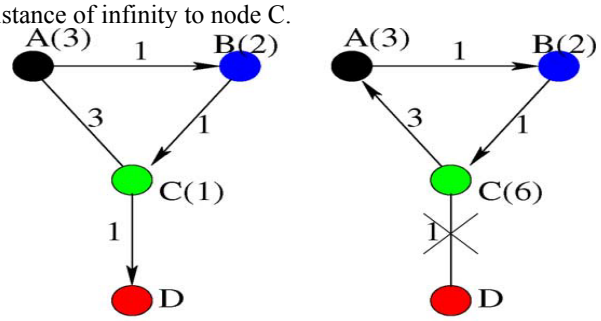
(Changes anywhere in the network trigger computations at all nodes). By combining information dissemination and path computation, distance vector algorithms (cf. RIP [3], EIGRP [4]) avoid several of these disadvantages, which make them attractive, especially in situations of frequent local topology changes and/or when high control overhead is undesirable. However, they can suffer from frequent and 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.

II. BACKGROUND WORK

ROUTING LOOPS AND COUNTING-TO-INFINITY

We begin our discussion with a simple classical example of a routing loop and counting-to-infinity

which illustrates that these problems can occur quite frequently as they neither require complex topologies nor an unlikely sequence of events. Consider the network shown in Fig. 1(a). In this figure, the nodes compute a shortest path to the destination D. The cost of each link is shown next to the link and the cost-to-destination of the nodes are shown in parenthesis next to the node. We assume that nodes use *poison reverse*; i.e., each node reports an infinite cost-to-destination to its successor node. Thus, node C believes that node A can reach the destination at a cost of 3 whereas node B cannot reach the destination since node B reported a distance of infinity to node C.



(a) Original topology. (b) Node *D* is down.

Fig. 1. A simple example of counting-to-infinity problem

Now suppose that the link between nodes C and D goes down, as shown in Fig. 1(b). Node C detects this change and attempts to find a new successor. According to the information node C has at that moment, node A is its best successor. So node C chooses node A as its successor, reports a distance of infinity to node A and distance of 6 to node B. As Fig. 1(b) shows, a routing loop has been created due to node C's choice of successor. To see how counting-to-infinity takes place in this example, note that due to poison reverse, node B believes that the destination is unreachable through node A. Thus, when it receives the update from C containing C's new cost-to-destination as 6, node B simply changes its own cost-to-destination to 7 keeping node C as its successor, reports unreachability to node C and its new cost, 7, to node A . This way, each node increases its cost to D by a finite amount each time. So, unless a maximum diameter of the graph is assumed (e.g., it is 6 in RIP) and the destination declared unreachable once the cost reaches that value, the computation never ends.

A. 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 costto-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 main drawbacks of Distance Vector are limited scalability due to slow convergence time, bandwidth consumption and routing loops.

B. 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.

C. Loop Free Invariance (LFI) Algorithms

A pair of invariances, based on the costto- 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 maintainthe 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.

D. 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.

III. DIV

A. 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.

B. 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 reordering). 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, $\{y_1,y_2,\ldots, y_n\}$, it 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∈{y_1; y_2; : : : ; y_n}],
3. and the value of itself as known to its neighbors
V (x;y_i|x) [y_i ∈ {y_1, y_2, . . . , 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 (x;x|y)=V(x;y_i)=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) ≤ V (x; y_i|x) for each neighbor y_i. (1)

Invariance2: 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). (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 number5. 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,…,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 yi 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; $x|x) \leq V (x; y_i||x), \forall_l = 1,2,\ldots,n$.

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 its own value is adjusted appropriately.

IV. 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 numbers 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.

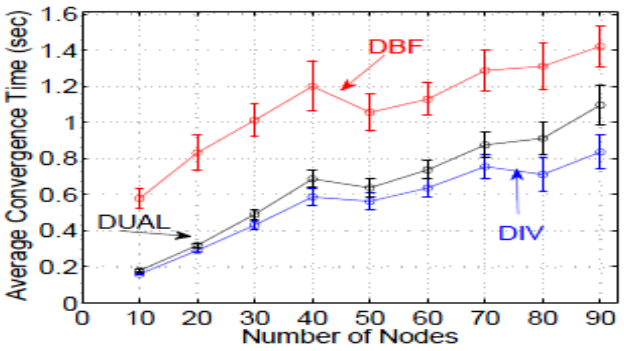


Fig. 2. Mean convergence time

Fig. 2. 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). the fraction of times the condition of DIV is satisfied given that SNC is not satisfied; this fraction exceeds 80% for larger graphs.

V. 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. onaventure, "*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].
[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. Gu´erin, 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