A Simple Approximation to Minimum-Delay Routing *

Srinivas Vutukury
Computer Science Department

J.J. Garcia-Luna-Aceves
Computer Engineering Department

Baskin School of Engineering
University of California, Santa Cruz, CA 95064
http://www.soe.ucsc.edu/research/ccrg/
{vutukury,jj}@cse.ucsc.edu

Abstract

The conventional approach to routing in computer networks consists of using a heuristic to compute a single shortest path from a source to a destination. Single-path routing is very responsive to topological and link-cost changes; however, except under light traffic loads, the delays obtained with this type of routing are far from optimal. Furthermore, if link costs are associated with delays, single-path routing exhibits oscillatory behavior and becomes unstable as traffic loads increase. On the other hand, minimum-delay routing approaches can minimize delays only when traffic is stationary or very slowly changing.

We present a "near-optimal" routing framework that offers delays comparable to those of optimal routing and that is as flexible and responsive as single-path routing protocols proposed to date. First, an approximation to the Gallager's minimum-delay routing problem is derived, and then algorithms that implement the approximation scheme are presented and verified. We introduce the first routing algorithm based on link-state information that provides multiple paths of unequal cost to each destination that are loop-free at every instant. We show through simulations that the delays obtained in our framework are comparable to those obtained using the Gallager's minimum-delay routing. Also, we show that our framework renders far smaller delays and makes better use of resources than traditional single-path routing.

1 Introduction

The standard approach to routing in computer networks today consists of computing a single shortest path from a source to each destination using some heuristic link-cost metric, which is typically not directly associated with the transmission and queueing delays over links and paths. A less common approach to routing is that of defining the routing problem as an optimization problem (e.g., multicommodity problem [5]) with a specific objective function, such as minimizing delays or maximizing throughput, and solving the problem using any of several known optimization techniques. These two traditional approaches to routing have inherent strengths and drawbacks.

In order to provide minimum delays, all optimal routing algorithms require the input traffic and the network topology to be stationary or very slowly changing (quasi-static), and require a priori knowledge of global constants that guarantee convergence of the routing algorithm. This makes optimal routing algorithms impractical for real networks, because in real networks traffic is very bursty at any time scale and the network topology frequently experience changes. Moreover, defining global constants that work for all input traffic patterns are impossible to determine.

On the other hand, routing algorithms based on single shortest-path heuristics adapt very quickly to changing network conditions, making them far more preferable than optimal routing for implementation in real networks. The main shortcoming of single shortest-path routing is that the delays achievable with such heuristics are far longer than those achievable using optimal routing algorithms. In addition, single-shortest-path routing becomes unstable under heavy loads or very bursty traffic when the link cost metric used in the routing algorithm is related to the delays or congestion experienced over the links [3].

The fact that shortest-path routing over single paths is far less efficient than optimal dynamic routing and the oscillatory behavior of shortest-path routing when link costs are tied to link delays has been known for many years. However, implementing optimal dynamic routing in a computer network has simply been infeasible to date. The key contributions of this paper consist of: (a) introducing a new framework for near-optimum delay routing; (b) verifying, for the first time, a set of invariants that permit routing-algorithm designers to approximate Gallager's necessary and sufficient conditions for minimum-delay routing with loop-free routing conditions that can be achieved using distributed routing algorithms that do not require any global variables or global synchronization; and (c) showing an example that provides end-to-end delays that are comparable to the optimal, while being as fast as today's shortest-path routing schemes.

Section 2 presents the minimum-delay routing problem (MDRP) as described by Gallager, and Gallager's minimum-delay routing algorithm [8]. Gallager's algorithm is unsuitable for practical networks and internetworks, because its speed of convergence to the optimal routes depends on a global constant, and because it requires that the input traffic and network topology be stationary or quasi-stationary.

Several algorithms have been proposed to date that improve over Gallager's minimum-delay routing algorithm [2, 6, 23, 24]. Segall and Sidi [23, 24] extended Gallager's minimum-delay routing algorithm to handle topological changes using techniques developed by Merlin and Segall [19]. Cassandras et al. [6] present a better technique for measuring marginal delays. Bertsekas and Gallager [2] used second derivatives to speed up convergence of Gallager's algorithm. However, all these algorithms are still dependent on global constants and the requirement that network traffic be static or quasi-static.

Because of its oscillatory behavior when link costs are related to delays, attempts to improving shortest-path routing have been restricted mainly to using better link cost metrics (e.g., [18, 13]) or using multiple-paths. To avoid undetected loops, OSPF permits multiple paths to a destination only when they have the same length [20]. More recently, Zaumen and Garcia-Luna-Aceves [27] proposed an algorithm based on distance vectors that supports multiple paths of unequal costs to each destination; however, link costs are not tied to delays. Wang and Crowcroft [26] addressed the drawbacks of the shortest-path first (SPF) algorithm by using alternate paths to detour traffic around points of congestion or network failures. However, the alternate paths in SPF-EE (for emergency

*This work was supported in part by the Defense Advanced Research Projects Agency (DARPA) under grants F30602-97-1-0291 and F19628-96-C-0038.

exits) are computed on a reactive basis, i.e., once congestion occurs, which is less effective in dealing with short bursts of traffic.

Cain et al. [4] describe a routing algorithm for minimizing delays. However, this algorithm requires that the routing-table updates at all the routers be synchronized, otherwise looping occurs, which increases end-to-end delays. Because the synchronization intervals required by this algorithm must be known by all routers, this is akin to using a global constant as in Gallager's algorithm. This approach is not scalable to very large networks, because the time needed for routing-table update synchronization becomes large, and this in turn limits its responsiveness to short-term traffic fluctuations. What is seriously lacking in this algorithm is a technique for asynchronous computation of multiple paths with instantaneous loop-freedom.

Section 3 presents a new framework for approximate solutions to MDRP. The novelty of this framework stems from partitioning the computation of minimum-delay paths in two parts. First, multiple loop-free paths of unequal cost to a destination are first established using long-term link-cost information. This is followed by the allocation of flows to destinations along the multiple loop-free paths available at each router; such an allocation is based on heuristics that attempt to minimize delays using short-term link-cost information. It is this partitioning of MDRP that permits us to implement routing algorithms that provide routers with near-optimum delays while keeping the routing algorithm as responsive to traffic or topology changes as the best of today's shortest-path routing algorithms. A set of invariants is also presented that permits Gallager's necessary and sufficient conditions for minimum-delay routing to be approximated with loop-free routing conditions achievable with simple distributed routing algorithms that do not require any global variables or global synchronization.

Section 4 describes a specific routing algorithm based on our new routing framework. This algorithm consists of two key components: (a) the first link-state routing algorithm that provides multiple loop-free paths of arbitrary positive cost at every instant, and (b) flow allocation heuristics that approximate minimum delays along the predefined multiple loop-free paths available for each destination.

Section 5 presents results of simulation experiments designed to illustrate the effectiveness of our solution in static and dynamic networks. We compare our approach against the optimal routing approach and shortest-path routing based on Dijkstra's shortest-path first (SPF) algorithm, because it is used widely in the Internet today. The simulation results illustrate that the routing delays obtained with our new algorithm are comparable to the optimal delays. Furthermore, the complexity of implementing our routing framework is similar to the complexity of routing protocols that provide single-path routing in the Internet today.

2 Minimum Delay Routing

2.1 Problem formulation

The minimum-delay routing problem (*MDRP*) was first formulated by Gallager [8], and we provide the same description in this section. A computer network $G = (N, L)$ is made up of N routers and L links between them. Each link is bidirectional with possibly different costs in each direction.

Let $r_j^i \geq 0$ be the expected input traffic, measured in bits per second, entering the network at router i and destined for router j. Let t_j^i be the sum of r_j^i and the traffic arriving from the neighbors of i for destination j. And let routing parameter ϕ_{jk}^i be the fraction of traffic t_j^i that leaves router i over link (i,k). Assuming that the network does not lose any packets, from conservation of traffic we have

$$t_j^i = r_j^i + \sum_{k \in N^i} t_j^k \phi_{ji}^k \tag{1}$$

where N^i is the set of neighbors of router i.

Let f_{ik} be the expected traffic, measured in bits per second, on link (i,k). Because $t_j^i \phi_{jk}^i$ is the traffic destined for router j on link (i,k) we have the following equation to find f_{ik}.

$$f_{ik} = \sum_{j \in N} t_j^i \phi_{jk}^i \tag{2}$$

Note that $0 \leq f_{ik} \leq C_{ik}$, where C_{ik} is the capacity of link (i,k) in bits per second.

Property 1 *For each router i and destination j, the routing parameters ϕ_{jk}^i must satisfy the following conditions:*

1. *$\phi_{jk}^i = 0$ if $(i,k) \notin L$ or $i = j$. Clearly, if the link does not exist, there can be no traffic on it.*
2. *$\phi_{jk}^i \geq 0$. This is true, because there can be no negative amount of traffic allocated on a link.*
3. *$\sum_{k \in N^i} \phi_{jk}^i = 1$. This is a consequence of the fact that all incoming traffic must be allocated to outgoing links.*

Let D_{ik} be defined as the expected number of messages or packets per second transmitted on link (i,k) times the expected delay per message or packet, including the queueing delays at the link. We assume that messages are delayed only by the links of the network and D_{ik} depends only on flow f_{ik} through link (i,k) and link characteristics such as propagation delay and link capacity. $D_{ik}(f_{ik})$ is a continuous and convex function that tends to infinity as f_{ik} approaches C_{ik}. The total expected delay per message times the total expected number of message arrivals per second is given by

$$D_T = \sum_{(i,k) \in L} D_{ik}(f_{ik}) \tag{3}$$

Note that the router traffic-flow set $t = \{t_j^i\}$ and link-flow set $f = \{f_{ik}\}$ can be obtained from $r = \{r_j^i\}$ and $\phi = \{\phi_{jk}^i\}$. Therefore, D_T can be expressed as a function of r and ϕ using Eqs. (1) and (2). The minimum-delay routing problem can now be stated as follows:

MDRP: *For a given fixed topology and input traffic flow set $r = \{r_j^i\}$, and delay function $D_{ik}(f_{ik})$ for each link (i,k), the minimization problem consists of computing the routing parameter set $\phi = \{\phi_{jk}^i\}$ such that the total expected delay D_T is minimized.*

2.2 A Minimum Delay Routing Algorithm

Gallager [8] derived the necessary and sufficient conditions that must be satisfied to solve MDRP. These conditions are summarized in Gallager's Theorem stated below.

The partial derivatives of the total delay, D_T, of Eq.(3) with respect to r and ϕ play a key role in the formulation and solution of the problem; these derivatives are:

$$\frac{\partial D_T}{\partial r_j^i} = \sum_{k \in N^i} \phi_{jk}^i [D'_{ik}(f_{ik}) + \frac{\partial D_T}{\partial r_j^k}] \tag{4}$$

$$\frac{\partial D_T}{\partial \phi_{jk}^i} = t_j^i [D'_{ik}(f_{ik}) + \frac{\partial D_T}{\partial r_j^k}] \tag{5}$$

where $D'_{ik}(f_{ik}) = \partial D_{ik}(f_{ik}) / \partial f_{ik}$. and is called the *marginal delay* or *incremental delay*.

Similarly, $\partial D_T/\partial r_j^i$ is called the *marginal distance* from router i to j.

Gallager's Theorem [8]: *The necessary condition for a minimum of D_T with respect to ϕ for all $i \neq j$ and $(i,k) \in L$ is*

$$\frac{\partial D_T}{\partial \phi_{jk}^i} = \begin{cases} = & \lambda_{ij} \quad \phi_{jk}^i > 0 \\ \geq & \lambda_{ij} \quad \phi_{jk}^i = 0 \end{cases} \tag{6}$$

where λ_{ij} is some positive number, and the sufficient condition to minimize D_T with respect to ϕ is for all $i \neq j$ and $(i,k) \in L$ is

$$D'_{ik}(f_{ik}) + \frac{\partial D_T}{\partial r_j^k} \geq \frac{\partial D_T}{\partial r_j^i} \quad \square \tag{7}$$

Eq. (4) shows the relation between a router's marginal distance to a particular destination and the marginal distances of its neighbors to the same destination. Eqs. (5)-(7) indicate the conditions for *perfect load balancing*, i.e., when the routing parameter set ϕ gives the minimum delay.

The set of neighbors through which router i forwards traffic towards j is denoted by S_j^i and is called the *successor set*.[1]

Under perfect load balancing with respect to a particular destination, the marginal distances through neighbors in the successor set are equal to the marginal distance of the router, and the marginal distances through neighbors *not* in the successor set are higher than the marginal distance of the router.

Let D_j^i denote the *marginal distance* from i to j, i.e., $\partial D_T/\partial r_j^i$. Let the *marginal delay* $D'_{ik}(f_{ik})$ from i to k be denoted by l_k^i which is also called the cost of the link from i to k.

According to Gallager's Theorem, the minimum delay routing problem now becomes one of determining, at each router i for each destination j: the routing parameters $\{\phi_{jk}^i\}$, S_j^i and D_j^i, such that the following five equations are satisfied:

$$D_j^i = \sum_{k \in N^i} \phi_{jk}^i (D_j^k + l_k^i) \tag{8}$$

$$S_j^i = \{k | \phi_{jk}^i > 0 \wedge k \in N^i\} \tag{9}$$

$$D_j^i \leq D_j^k + l_k^i \qquad k \in N^i \tag{10}$$

$$(D_j^p + l_p^i) = (D_j^q + l_q^i) \qquad p, q \in S_j^i \tag{11}$$

$$(D_j^p + l_p^i) < (D_j^q + l_q^i) \qquad p \in S_j^i \quad q \notin S_j^i \tag{12}$$

This reformulation of MDRP is critical, because it is the first step in allowing us to approach the problem by looking at the next-hops and distances obtained at each router for each destination. Gallager [8] described a distributed routing algorithm for solving the above five equations. When the algorithm converges, the aggregate of the successor sets for a given destination j (S_j^i for every i) define a directed acyclic graph. In fact, in any implementation, S_j^i *must be* loop-free at every instant, because even temporary loops cause traffic to recirculate at some nodes and results in incorrect marginal delay computations, which in turn can prevent the algorithm from converging or obtaining minimum delays.

Gallager's distributed algorithm uses an interesting blocking technique to provide loop-freedom at every instant [8, 23, 24]. We refer to this algorithm as OPT in the rest of the paper. Unfortunately, OPT cannot be used in real networks for several reasons. A major drawback of OPT is that a global step size η needs to be chosen and every router must use it to ensure convergence. Because η depends on the input traffic pattern, it is impossible to determine one in practice that works for all input traffic patterns and for all possible topology modifications. The routing parameters are directly computed by OPT and the multiple loop-free paths are simply implied by the routing parameters in Eq. (9). The computation of routing parameters is, for all practical purposes, a very slow process as it is a destination-controlled process. The destination initiates every iteration that adjusts the routing parameters at every router; furthermore, each iteration takes a time proportional to the diameter of the network and number of messages proportional to number of links. This renders the algorithm slow converging and useful only when traffic and topology are stationary for times long enough for all routers to adjust their routing parameters between changes. Also, depending on the global constant η, the destination must initiate several iterations for the parameters to converge to their final values. The number of such iterations needed for convergence tends to be large for a small η, and small for a large value of η. Unfortunately, η cannot be made arbitrarily large to reduce the number of iterations and to speed up convergence, because the algorithm may not converge at all for large values of η.

Hence, Gallager's algorithm can be viewed only as a method for obtaining lower bounds under stationary traffic, rather than as an algorithm to be used in practice. The next section shows how the theory introduced in the Gallager's method can be adapted to practical networks.

3 A New Framework for Minimum-Delay Routing

We noted that in Gallager's algorithm the computation of the routing parameter set ϕ is slow converging and works only in the case of stationary or quasi-stationary traffic. In the Internet, traffic is hardly stationary and perfect load balancing is neither possible nor necessary. Intuitively, an approximate load balancing scheme based on some heuristic which can quickly adapt to dynamic traffic should be sufficient to minimize delays substantially.

The key idea in our approach is, in a sense, to reverse the way in which Gallager's algorithm solves MDRP. The intuition behind our approach is that establishing paths from sources to destinations takes a much longer time than shifting loads from one set of neighbors to another, simply because of the propagation and processing delays incurred along the paths. Accordingly, it makes sense to first establish multiple loop-free paths using long-term (end-to-end) delay information, and then adjust routing parameters along the predefined multiple paths using short-term (local) delay information.

This new approach allows us to attempt to use distributed algorithms to compute multiple loop-free paths from source to destination that, hopefully, are as fast as today's single-path routing algorithms, and local heuristics that can respond quickly to temporary traffic bursts using local short-term metrics alone. Therefore, we map Eqs. (8)-(12) derived in Gallager's method into the following three equations:

$$D_j^i = min\{D_j^k + l_k^i | k \in N^i\} \tag{13}$$

$$S_j^i = \{k | D_j^k < D_j^i \wedge k \in N^i\} \tag{14}$$

$$\phi_{jk}^i = \Psi(k, A_j^i, B_j^i) \qquad k \in N^i \tag{15}$$

where $A_j^i = \{D_j^p + l_p^i | p \in N^i\}$ and $B_j^i = \{\phi_{jp}^i | p \in N^i\}$.

These equations simply state that, for an algorithm to approximate minimum-delay routing, it must establish loop-free paths and use a function Ψ to allocate flows over those paths. We observe that Eq. (13) is the well-known Bellman-Ford (BF) equation for computing the shortest paths, and Eq. (14) is the successor set consisting of the neighbors that are closer to the destination than the router itself. Note that the paths implied by the neighbors in the successor set of a router need not be of the same length. The function Ψ in Eq. (15) is a heuristic function that determines the routing parameters. Because changing the routing parameters effects the marginal delay of the links (hence link-costs), we use regular updates of the link costs.

The main problem with attempting to solve MDRP using Eqs. (13) to (15) directly is that these equations assume that routing information is consistent throughout the network. In practice, a node (router) must choose its distance and successor set using routing information obtained through its neighbors, and this information may

[1] The term successor set was first introduced in [27].

be outdated. At any time t, for a particular destination j, the successor sets of all nodes define a *routing graph* $SG_j(t) = \{(m,n) | n \in S_j^m(t),\ m \in N\}$. In single-path routing, $S_j^i(t)$ has at most one neighbor: the neighbor that is on the shortest path to destination j. Accordingly, $SG_j(t)$ for single-path routing is a sink-tree rooted at j if loops are never created. The routing graph $SG_j(t)$ in our case should be a directed acyclic graph in order for minimum delays to be approached.

The blocking technique used in Gallager's algorithm ensures instantaneous loop-freedom. Likewise, to provide loop-free paths even when the network is in transient state within the context of our framework, additional constraints must be imposed on the choice of successors at each router, which essentially must preclude the use of neighbors that *may* lead to looping.

Several algorithms have been proposed in the past to provide loop-free paths at every instant for the case of single-path routing (e.g., the Jaffe-Moss algorithm [15], DUAL [9], LPA [11], and the Merlin-Segall algorithm [19]) and one algorithm, DASM, has been proposed for the case of multiple paths per destination [27]. All these algorithms are based on the exchange of vectors of distances, together with some form of coordination among routers spanning one or multiple hops. The coordination among routers determines when the routers can update their routing tables. This coordination is in turn guided by local conditions that depend on values of reported distances to destinations and that are sufficient to prevent loops from occurring.

We generalize the work to date on loop-free routing over single paths or multiple paths by means of the following loop-free invariant (**LFI**) conditions, which are applicable to *any* type of routing algorithm. We adopt the same terminology and nomenclature first introduced for DUAL [9] to describe the LFI conditions.

Loop-free Invariant (LFI) conditions: *Any routing algorithm designed such that the following two equations are always satisfied, automatically provides loop-free paths at every instant, regardless of the type of routing algorithm being used:*

$$FD_j^i \leq D_{ji}^k \qquad k \in N^i \tag{16}$$

$$S_j^i = \{\, k \mid D_{jk}^i < FD_j^i \wedge k \in N^i \} \tag{17}$$

where D_{jk}^i is the value of D_j^k reported to i by its neighbor k; and FD_j^i is called the feasible distance *of router i for destination j and is an estimate of D_j^i, in the sense that FD_j^i equals D_j^i in steady state but is allowed to differ from it temporarily during periods of network transitions.*

In link-state algorithms, the values of D_{jk}^i are determined locally from the link-state information supplied by the router's neighbors; in contrast, in distance-vector algorithms, the distances are directly communicated among neighbors. The following theorem verifies this key result of our framework.

Theorem 1 *If the LFI conditions are satisfied at any time t, the routing graph $SG_j(t)$ implied by the successor sets $S_j^i(t)$ is loop-free.*

Proof: Let $k \in S_j^i(t)$ then from Eq. (17) we have

$$D_{jk}^i(t) < FD_j^i(t) \tag{18}$$

At router k, because router i is a neighbor, from Eq. (16) we have $FD_j^k(t) \leq D_{jk}^i(t)$. Combining this result with Eq. (18) we obtain

$$FD_j^k(t) < FD_j^i(t) \tag{19}$$

Eq. (19) states that, if k is a successor of router i in a path to destination j, then k's feasible distance to j is strictly less than the feasible distance of router i to j. Now, if the successor sets define a loop at time t with respect to j, then for some router p on the loop, we arrive at $FD_j^p(t) < FD_j^p(t)$, an absurd relation. Therefore, the LFI conditions are sufficient for loop-freedom. □

With the result of Theorem 1, Eq. (14) can be approximated with the LFI conditions to render a routing approach that does not require routing information to be globally consistent, at the expense of rendering delays that may be longer than optimal. Accordingly, our framework for near-optimum-delay routing lies in finding the solution to the following equations using a distributed algorithm:

$$D_j^i = min\{D_j^k + l_k^i | k \in N^i\} \tag{20}$$

$$FD_j^i \leq D_{ji}^k \qquad k \in N^i \tag{21}$$

$$S_j^i = \{\, k \mid D_{jk}^i < FD_j^i \wedge k \in N^i \} \tag{22}$$

$$\phi_{jk}^i = \Psi(k, \{D_j^p + l_p^i | p \in N^i\}, \{\phi_{jp}^i | p \in N^i\})\ k \in N^i \tag{23}$$

4 Implementing Near-Optimum-Delay Routing

We present an approach based on link-state information, rather than distance information, because extending our results to minimum-delay routing with additional constraints can be done more efficiently by working with link parameters than path parameters, which are the combination of link parameters. Our approach consists of three components: computing multiple loop-free paths, distributing traffic over such paths, and computing link costs.

4.1 Computing Multiple Loop-free Paths

We describe the computation of multiple loop-free paths in two parts: computing D_j^i using a shortest-path algorithm based on link-state information, and computing S_j^i by extending that algorithm to support multiple successors along loop-free paths to each destination.

4.1.1 Computing D_j^i

There are many distributed algorithms for computing shortest paths, and any of them can be extended to provide multiple paths of equal and unequal costs as long as the extension obeys the LFI conditions introduced in the previous section.

The partial-topology dissemination algorithm (PDA) propagates enough link-state information in the network, so that each router has *sufficient* link-state information to compute shortest paths to all destinations. In this respect, it is similar to other link-state algorithms (e.g., OSPF [20], SPTA [25], LVA [10], ALP [12]). PDA combines the best features of LVA, ALP and SPTA. As in LVA and ALP, a router communicates to its neighbors information regarding only those links that are part of its minimum-cost routing tree, and like SPTA, a router validates link information based on distances to heads of links and not on sequence numbers.

PDA assumes that a router detects the failure, recovery and link-cost change of an adjacent link within a finite amount of time. An underlying protocol ensures that messages transmitted over an operational link are received correctly and in the proper sequence within a finite time and are processed by the router one at a time in the order received. These are the same assumptions made for similar routing algorithms and can be easily satisfied in practice. Each router i running PDA maintains the following information:

1. The *main topology table*, T^i, stores the characteristics of each link known to router i. Each entry in T^i is a triplet $[h, t, d]$ where h is the head, t is the tail and d is the cost of the link $h \rightarrow t$.
2. The *neighbor topology table*, T_k^i, is associated with each neighbor k. The table stores the link-state information communicated by the neighbor k. That is, T_k^i is a time-delayed version of T^k.

procedure INIT-PDA
{*Invoked when the router comes up.*}
begin
Initialize all tables;
call PDA;
end INIT-PDA

procedure PDA
{*Executed at each router i. Invoked when an event occurs*}
begin
(1) **call** NTU;
(2) **call** MTU; /* Updates T^i */
(3) **if** (there are changes to T^i) **then**
Compose an LSU message consisting of topology differences using *add*, *delete* and *change* link entries;
endif
(4) Within a finite amount time, send the LSU message to all neighbors;
end PDA

Figure 1: The Partial-topology Dissemination Algorithm

procedure NTU
begin
(1) **if** (LSU message is received from a neighbor k) **then**
(1a) Update neighbor table T^i_k. That is, add links, delete links or change links according to the specification of each entry in the LSU;
(1b) Run Dijkstra's shortest path algorithm on the resulting topology T^i_k; /*This results in finding minimum distances from k to all other nodes in T^i_k. Note T^i_k is a tree*/
(1c) Update D^i_{jk} with new distances in T^i_k;
endif
(2) **if** (adjacent link $(i,\ k)$ is up) **then**
Update l^i_k and send an LSU message to the neighbor k with link information of all links in its main topology table T^i;
endif
(3) **if** (cost of an adjacent link $(i,\ k)$ changed)**then**
Update l^i_k;
endif
(4) **if** (adjacent link $(i,\ k)$ failed)**then**
Update l^i_k and clear the table T^i_k;
endif
end NTU

Figure 2: Neighbor Topology Table Update algorithm

3. The *distance table* stores the distances from router i to each destination based on the topology in T^i and the distances from each neighbor k to each destination based on the topologies in T^i_k for each k. The distance of router i to node j in T^i is denoted by D^i_j; the distance from k to j in T^i_k is denoted by D^i_{jk}.

4. The *routing table* stores, for each destination j, the successor set S^i_j and the feasible distance FD^i_j, which is used by MPDA to enforce LFI conditions.

5. The *link table* stores, for each neighbor k, the cost l^i_k of the adjacent link to the neighbor.

The unit of information exchanged between routers is a link-state update (LSU) message. A router sends an LSU message containing one or more entries, with each entry specifying *addition*, *deletion* or *change* in cost of a link in the router's main topology table T^i. Each entry of an LSU consists of link information in the form of a triplet $[h, t, d]$ where h is the head, t is the tail, and d is the cost of the link $h \rightarrow t$. An LSU message contains an acknowledgment (ACK) flag for acknowledging the receipt of an LSU message from a neighbor (used only by MPDA).

The INIT-PDA procedure in Fig. 1 initializes the tables of a router at startup time; all variables of type distance are initialized to infinity and those of type node are initialized to null. All successor sets are initialized to the empty set. PDA is executed each time an event occurs; an event can be either a receipt of an LSU message from a neighbor or the detection of an adjacent link-cost change. Procedure NTU (Neighbor Topology Table Update) shown in Fig. 2 is used to process the received message and update the necessary tables. Procedure MTU in Fig. 3 constructs the router's own shortest path tree from the topologies reported by its neighbors. The new shortest-path tree obtained is compared with the previous version to determine the differences; only the differences are then reported to the neighbors. The router then waits for the next event and, when it occurs, the whole process is repeated.

The algorithm MTU at router i merges the topologies T^i_k and the adjacent links l^i_k to obtain T^i. The merge process is straightforward if all neighbor topologies contain disjoint sets of links, but when two or more neighbors report conflicting information regarding a particular link, the conflict has to be resolved. Sequence numbers may be used to distinguish between old and new link information as in OSPF, but PDA resolves the conflict as follows. If two or more neighbors report information of link $(m,\ n)$ then the router i should update topology table T^i with link information reported by the neighbor that offers the shortest distance from the router i to the head node m of the link. Ties are broken in favor of neighbor with the lowest address. For adjacent links, router i itself is the head of the link and thus has the shortest distance. Therefore, any information about an adjacent link supplied by neighbors will be overridden by the most current information about the link available to router i. Dijkstra's shortest path algorithm is run on T^i and only the links that constitute the shortest-path tree are retained. Note that, because there are potentially many shortest-path trees, ties should be broken consistently during the run of Dijkstra's algorithm.

In what follows, we show that PDA works correctly by showing that the topology tables at all nodes converge to the shortest paths within a finite time after the last link cost change in the network. After convergence, because there are no more changes to the topology tables, no more LSU messages are generated.

Definitions: The n-hop minimum distance of router i to node j in a network is the minimum distance possible using a path of n links or less. A path that offers the n-hop minimum distance is called n-hop minimum path. If there is no path with n hops or less from router i to j then the n-hop minimum distance from i to j is undefined. An n-hop minimum tree of a node i is a tree in which router i is the root and all paths of n hops or less from the root to any other node is an n-hop minimum path. Note that there could be more than one n-hop minimum tree.

Let $\mathbf{G}$ denote the final topology of the network after all link changes occurred as seen by an omniscient observer; we use bold font to refer to all quantities in $\mathbf{G}$. Let $\mathbf{H^i_n}$ denote an n-hop minimum tree rooted at router i in $\mathbf{G}$ and let $\mathbf{M^i_n}$ be the set of nodes that are within n hops from i in $\mathbf{H^i_n}$. Let $\mathbf{D^{i,j}_n}$ denote the distance of i to j in $\mathbf{H^i_n}$. Let $\mathbf{d_{ij}}$ be the cost of the link $i \rightarrow j$. The notation $i \rightsquigarrow j$ indicates a path from i to j of zero or more links.

Property 2 *From the principle of optimality (a sub-path of a shortest path between two nodes is also a shortest path between the end nodes of the sub-path), if H and H' are two n-hop minimum trees rooted at router i and M and M' are sets of nodes that are within n hops from i in H and H' respectively, then $M = M' = \mathbf{M^i_n}$. Also, for each $j \in \mathbf{M^i_n}$ the length of path $i \rightsquigarrow j$ in both H and H' is equal to $\mathbf{D^{i,j}_n}$. Also, $\mathbf{D^{i,j}_h} \leq \mathbf{D^{i,j}_n}$ if $h \geq n$.*

procedure MTU at router i
begin
(1) $oldT^i \leftarrow T^i$;/* Save copy */
(2) **if** (node j occurs in at least one of T_k^i) **then**
add j to the main topology table T^i;
endif
(3) **foreach** node j in T^i **do**
$MIN \leftarrow min\{D_{jk}^i + l_k^i | k \in N^i\}$;
let p be such that $MIN = (D_{jp}^i + l_p^i)$;
/* Neighbor p is the preferred neighbor for destination j. Ties are broken in favor of lower address neighbor */
done
(4) **foreach** j in T^i and its preferred neighbor p **do**
Copy all links (j, n) from T_p^i to T^i;
/* i.e., copy all links in T_p^i for which j is the head node */
done
(5) Update T^i with information of each l_k^i;
(6) Run Dijkstra's shortest path algorithm on T^i and remove those links in T^i that are *not* part of the shortest path tree;
(7) Update D_j^i with new distances in T^i;
(8) Compare $oldT^i$ with T^i and note all differences;
end MTU

Figure 3: Main Topology Table Update Algorithm

We say a router i *knows at least* the n-hop minimum tree, if the tree represented by its main topology table T^i is at least an n-hop minimum tree rooted at i in $\mathbf{G}$ and there are at least n nodes in T^i that are reachable from the root i. Note that the links in T^i that are more than n hops may have costs that do not agree with the link costs in $\mathbf{G}$.

Lemma 1 *If a router i has the final correct costs of the adjacent links and for each neighbor k the topology T_k^i is an n-hop minimum tree, then the topology T^i is $(n+1)$-hop minimum tree after the execution of MTU.*

Proof: The proof is presented in the Appendix. □

Theorem 2 *At each router i, the main topology T^i gives the correct shortest paths to all known destinations a finite time after the last change in the network.*

Proof: The proof is by induction on t_n, the global time when for each router i, T^i is at least n-hop minimum tree. Because the longest loop-free path in the network has at most $N-1$ links where N is number of nodes in the network, t_{N-1} is the time when every router has the shortest path to every other node. We need to show that t_{N-1} is finite. The base case is t_1, the time when every node has 1-hop minimum distance and because the adjacent link changes are notified within finite time, $t_1 < \infty$. Let $t_n < \infty$ for some $n < N$. Given that the propagation delays are finite each router will have each of its neighbors n-hop minimum tree in finite time after t_n. From Theorem 1 we can see that the router will have at least the $(n+1)$-hop minimum tree within a finite time after t_n. Therefore, $t_{n+1} < \infty$. From induction, we can conclude that $t_{N-1} < \infty$. □

4.1.2 Computing S_j^i

The LFI conditions introduced in Section 3 suggest a technique for computing S_j^i such that the implied routing graph SG_j is loop-free at every instant. To determine FD_j^i in Eq.(16), router i needs to know D_{ji}^k, the distance from i to node j in the topology table T_i^k. Because of propagation delays, there may be discrepancies between the main topology table T^i at router i and its copy T_i^k at the neighbor k. However, at time t, the topology table T_i^k is a copy of the main topology table T^i at some earlier time $t' < t$. Logically, if a copy of D_j^i is saved each time an LSU is sent, a feasible distance FD_j^i that satisfies the LFI conditions can be found in the history of values of D_j^i that have been saved!

procedure MPDA at router i
$\{$*invoked when an event occurs*$\}$
begin
(1) **call** NTU;
(2) **if** (node is in PASSIVE state) **then**
(2a) **call** MTU; /* update T^i and D_j^i */
(2b) $FD_j^i \leftarrow min\{FD_j^i, D_j^i\}$;
endif
(3) **if** (node is in ACTIVE state and the last ACK is received) **then**
(3a) $temp_j^i \leftarrow D_j^i$; Set node to PASSIVE state;
(3b) **call** MTU to update T^i;
(3c) $FD_j^i \leftarrow min\{temp_j^i, D_j^i\}$
endif
(4) $S_j^i \leftarrow \{k | D_{jk}^i < FD_j^i\}$;
(5) **if** (changes occur in T^i)**then**
Set node to ACTIVE state;
endif
if (no changes occur in T^i and the event is the last ACK) **then**
Set node to PASSIVE state;
endif
(6) **if** (there are changes to T^i) **then**
Compose a new LSU with the topology changes expressed as *add* link, *delete* link and *change* link;
endif
(7) **if** (input event received is an LSU message)**then**
Add the ACK entry to newly composed LSU;
endif
(8) Send the new LSU message.
end MPDA

Figure 4: Multiple-path Partial-topology Dissemination Algorithm (MPDA)

The multiple-path partial-topology dissemination algorithm, or MPDA, shown in Fig. 4 is a modification of PDA that enforces the LFI conditions by synchronizing the exchange of LSUs between neighbors. In MPDA, each LSU message sent by a router is acknowledged by all its neighbors before the router sends the next LSU. The inter-neighbor synchronization used in MPDA spans only a *single* hop, unlike the synchronization in diffusing computations [7] which potentially spans the whole network. A router is said to be in ACTIVE state when it is waiting for its neighbors to acknowledge the LSU message it sent; otherwise, it is in PASSIVE state.

Assume that, initially, all routers are in PASSIVE state with all routers having the correct distances to all destinations. Then a series of link cost changes occurs in the network resulting in some or all routers to go through a sequence of PASSIVE-to-ACTIVE and ACTIVE-to-PASSIVE state transitions, until all routers become PASSIVE with correct distances to destinations.

If a router in a PASSIVE state receives an event that does not change its topology T^i, then the router has nothing to report and remains in PASSIVE state. However, if a router in PASSIVE state receives an event that affects a change in its topology, the router sends those changes to its neighbors, goes into ACTIVE state and waits for ACKs. Events that occur during the ACTIVE period are processed to update T_k^i and l_k^i but not T^i; the updating

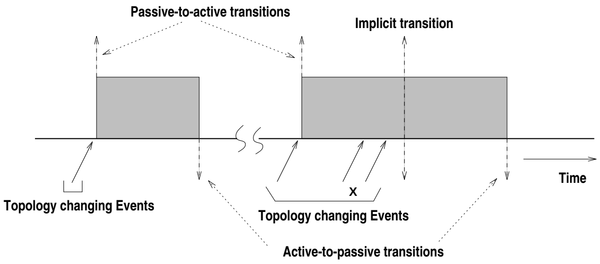

Figure 5: Active-passive phase transitions in MPDA.

of T^i by MTU is deferred until the end of the ACTIVE phase. At the end of the ACTIVE phase, when ACKs from all neighbors are received, router i updates T^i with changes that may have occurred in T^i_k due to events received during the ACTIVE phase. If no changes occurred in T^i that need reporting, then the router becomes PASSIVE; otherwise, as shown in Fig. 5, there are changes in T^i that may have resulted due to events and the neighbors need to be notified. This results in a new LSU, and the router immediately becoming ACTIVE again. In this case, there is an implicit PASSIVE period, of zero length of time, between two back-to-back ACTIVE periods, as illustrated in Fig. 5. A router i receiving an LSU message from k must send back an LSU with the ACK bit set after updating T^i_k. If the router does not have any updates to send, either because it is in ACTIVE state or because it does not have any changes to report, it sends back an empty LSU with just the ACK flag set. When a router detects that an adjacent link failed, any pending ACKs from the neighbor at the other end of the link are treated as received. Because all LSUs are acknowledged within a finite time, no deadlocks can occur.

The following theorem proves that MPDA provides loop-free multipaths at every instant.

Theorem 3 *(Safety property) At any time t, the directed graph $SG_j(t)$ implied by the successor sets $S^i_j(t)$ computed by MPDA at each router is loop-free.*

Proof: The proof is presented in the Appendix, and is based on showing that FD^i_j and S^i_j, as computed by MPDA, satisfy the LFI conditions. □

Theorem 4 *(Liveness property) A finite time after the last change in the network, D^i_j gives the correct shortest distance and*

$$S^i_j = \{k | D^k_j < D^i_j, k \in N^i\} \quad \textit{at each router } i$$

Proof: The convergence of MPDA follows directly from the convergence of PDA, because the update messages in MPDA are only delayed a finite time as allowed in line 4 in algorithm PDA. Therefore, the distances D^i_j in MPDA also converge to shortest distances. Because changes to T^i are always reported to the neighbors and are incorporated by the neighbors in their tables in finite time, $D^i_{jk} = D^k_j$ for $k \in N^i$ after convergence. From line 3c in MPDA, we observe that when router i becomes PASSIVE, and $FD^i_j = D^i_j$ holds true. Because all routers are PASSIVE at convergence time it follows that the set $\{k | D^i_{jk} < FD^i_j, k \in N^i\}$ is the same as the set $\{k | D^k_j < D^i_j, k \in N^i\}$. □

4.2 Distributing Traffic over Multiple Paths

In general, the function Ψ can be any function that satisfies Property 1, but our objective is to obtain a function Ψ that performs load balancing that is as close as possible to perfect load balancing (Eqs.(10)-(12)).

procedure IH
begin
(1) $\forall k \notin S^i_j, \phi^i_{jk} \leftarrow 0$;
(2) **if** ($|S^i_j| = 1$) **then**
$\forall k \in S^i_j, \phi^i_{jk} \leftarrow 1$;
endif
(3) **if** ($|S^i_j| > 1$) **then**

$$\phi^i_{jk} \leftarrow \frac{1 - \frac{D^i_{jk} + l^i_k}{\sum_{m \in S^i_j} (D^i_{jm} + l^i_m)}}{(|S^i_j| - 1)}, \quad \forall k \in S^i_j;$$

endif
end IH

Figure 6: Heuristic for initial load assignment.

procedure AH
begin
(1) $D^{ij}_{min} \leftarrow min\{D^i_{jk} + l^i_k \ | k \in S^i_j\}$;
(2) **let** $D^{ij}_{min} = (D^i_{jk_0} + l^i_{k_0})$;
// That is, k_0 be the neighbor
that offers this minimum)
(3) **foreach** $k \in S^i_j$ **do**
$a^i_{jk} \leftarrow D^i_{jk} + l^i_k - D^{ij}_{min}$;
done
(4) $\Delta \leftarrow \frac{1}{2} min\{\frac{\phi^i_{jk}}{a^i_{jk}} | k \in S^i_j \wedge a^i_{jk} \neq 0\}$;
(4) **foreach** $k \neq k_0 \wedge k \in S^i_j$ **do**
$\phi^i_{jk} \leftarrow \phi^i_{jk} - \Delta \times a^i_{jk}$;
done
(5) **for** $k = k_0$ **do**
$\phi^i_{jk} \leftarrow \phi^i_{jk} + \sum_{q \in S^i_j} \Delta \times a^i_{jq}$;
done
end AH

Figure 7: Heuristic for incremental load adjustment.

The function Ψ should also be suitable for use in dynamic networks, where the flows over links are continuously changing, causing continuous link-cost changes. To respond to these changes, queueing delays at the links must be measured periodically and routing paths must be recomputed. However, re-computing paths frequently consumes excessive bandwidth and may also cause oscillations. Therefore, routing-path changes should only be done at sufficiently long intervals. Unfortunately, a network cannot be responsive to short-term traffic bursts if only long-term updates are performed. For this reason, we use link costs measured over two different intervals; link costs measured over short intervals of length T_s are used for routing-parameter computation and link costs measured over longer intervals of length T_l are used for routing-path computation [17]. In general, T_l must be several times longer than T_s. Long-term updates are designed to handle long-term traffic changes and are used by the routing protocol to update the successor sets at each router, so that the new routing paths are the shortest paths under the new traffic conditions. The short-term updates made every T_s seconds are designed to handle short-term traffic fluctuations that occur between long-term routing path updates and are used to compute the routing parameters ϕ^i_{jk} in Eq. (15) locally at each router. Accordingly, our traffic distribution heuristics assume a constant successor set and successor graph.

When S^i_j is computed for the first time or recomputed again due to long-term route changes, traffic should be freshly distributed. In this case, the allocation heuristic function Ψ is a function of only the marginal distances through the successor set. That is, Eq. (15)

reduces to the form $\{\phi_{jk}^i\} = \Psi(k, \{D_j^p + l_p^i | p \in N^i\})$. When a new successor set S_j^i is computed, algorithm IH in Fig. 6 is first used to distribute traffic over the successor set [17]. Note that $\{\phi_{jk}^i\}$, computed in IH, satisfy Property 1. Furthermore, when more than one successor is present, if $D_{jp}^i + l_p^i > D_{jq}^i + l_q^i$ for successors p and q, then $\phi_{jp}^i < \phi_{jq}^i$. The heuristic makes sense because the greater the marginal delay through a particular neighbor becomes, the smaller the fraction of traffic that is forwarded to that neighbor.

After the first flow assignment is made over a newly computed successor set using algorithm IH, a different flow allocation heuristic algorithm AH shown in Fig. 7 is used to adjust the routing parameters every T_s seconds until the successor set changes again. The heuristic function Ψ computed in AH is incremental and, unlike IH, is a function of current flow allocation on the successor sets and the marginal distances through the successors. AH also preserves Property 1 at every instant. In AH traffic is incrementally moved from the links with large marginal delays to links with the least marginal delay. The amount of traffic moved away from a link is proportional to how large the marginal delay of the link is compared to the best successor link. The heuristic tends to distribute traffic in such a way that Eqs. (10)-(12) hold true. This is important, because the initial distribution obtained by IH is far from being balanced. The computation complexity of the heuristic allocation algorithms is $O(N^i)$. Because the heuristics are run for each active destination, the whole load-balancing activity is $O(N)$.

Unlike η in Gallager's algorithm, T_l and T_s are local constants that are set independently at each router. Convergence of our algorithm does not critically depend on these constants like optimal routing does on η. Also, T_l and T_s need not be static constants and can be made to vary according to congestion at the router. The value of T_l, however, should be such that it is sufficiently longer than the time it takes for computing the shortest paths. The long-term update periods should be phased randomly at each router, because of the problems that would result due to synchronization of updates [3].

4.3 Computing Link Costs

As mentioned earlier, the cost of a link is the marginal delay over the link $D'(f_{ik})$.

If the links are assumed to behave like M/M/1 queues, then the marginal delay $D'(f_{ik})$ can be obtained in a closed form expression by differentiating the following equation [16].

$$D_{ik}(f_{ik}) = \frac{f_{ik}}{(C_{ik} - f_{ik})} + \tau_{ik} f_{ik} \tag{24}$$

where f_{ik} is the flow through the link (i, k), and C_{ik} and τ_{ik} are the capacity and propagation delay of the link. Because the M/M/1 assumption does not hold in practice in the presence of very bursty traffic, and because Eq. (24) becomes unstable when f_{ik} approaches C_{ik}, an on-line estimation of the marginal delays is desirable.

There are several techniques for computing marginal delays that are currently available (e.g., [23, 22, 6]). For the purposes of simulations, we borrow a technique introduced by Cassandras, Abidi and Towsley [6] for on-line estimation of the marginal delay $D'(f_{ik})$. The technique uses perturbation analysis (PA) for the on-line estimation and is shown to perform better than the M/M/1 estimation. In addition, the PA estimation does not require a priori knowledge of the link capacities. This is very significant, because the capacity available to best-effort traffic in real networks varies according to the capacity allocated to other types of traffic, such as real-time traffic. We must emphasize that our approach does not depend on which specific technique is used for marginal-delay estimation, although some methods may be better than others. The convergence or stability of our routing algorithm does not depend on the specific technique used for marginal-delay estimation.

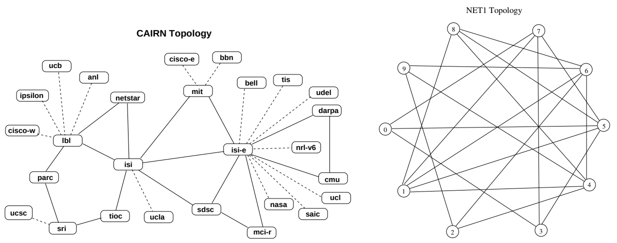

Figure 8: Topologies used in simulations

5 Simulations

The simulations discussed in this section illustrate the effectiveness of our near-optimal framework, and demonstrate the significant improvements achieved by our approach over single-path routing in static and dynamic environments. The delays obtained by optimal routing, single-path routing and our approximation scheme are compared under identical topological and traffic environments. The results show that the average delays achieved via our approximation scheme are comparable (within a small percentage difference rather than several times difference) to the optimal routing under quasi-static environment and the same are significantly better than single-path routing in a dynamic environment.

For optimal routing, we implemented the algorithm described by Gallager [8], and label it with 'OPT'. The plots of our approximation scheme are labeled with 'MP'. To obtain representative delays for single-path routing algorithms, we opted to restrict our multipath routing algorithm to use only the best successor for packet forwarding, instead of simulating any specific shortest-path algorithm. Because of the instantaneous loop-freedom property that MPDA exhibits, the shortest-path delays obtained this way are better than or similar to the delays obtained with either EIGRP [1], which is based on DUAL and requires much more internodal synchronization than our scheme, rendering longer delays, and RIP [14] or OSPF [20], which do not prevent temporary loops. We use the label 'SP' for single-path routing in the graphs.

We performed simulations on the topologies shown in Fig. 8. CAIRN (www.cairn.net) is a real network and NET1 is a contrived network. We are only interested in the connectivity of CAIRN, and its topology as used differs from the real network in the capacities and propagation delays assumed in the simulation experiments. We restricted the link capacities to a maximum of 10Mbs, so that it becomes easy to sufficiently load the networks. NET1 has a connectivity that is high enough to ensure the existence of multiple paths, and small enough to prevent a large number of one-hop paths. The diameter of NET1 is four and the nodes have degrees between 3 and 5. In each network we setup flows between several source-destination pairs and measure the average delays of each flow. The flows in CAIRN are setup between these source-destination pairs: (lbl, mci-r),(netstar, isie), (isi, darpa), (parc, sdsc), (sri, mit) ,(tioc, sdsc),(mit, sri),(isie, netstar), (sdsc, parc),(mci-r, tioc),(darpa, isi). For NET1, the source-destination pairs are: (9,2), (8,3), (7,0), (6,1), (5,8), (4,1), (3,8), (2,9), (1,6), (0,7).

The flows have bandwidths in the range 0.2-1.0 Mbs. For simplicity, we used a stable topology (links or nodes do not fail) in all the simulations. In the presence of link failures, MP can only perform better than SP, because of availability of alternate paths. Furthermore, OPT is not fast enough to respond to drastic topology changes. Because MP is parameterized by the T_l and T_s update intervals, its delay plots are represented by MP-TL-*xx*-TS-*yy*, where *xx* is the T_l update interval and *yy* is the T_s update interval measured in seconds. Similarly, the delays of shortest-path routing are represented by SP-TL-*xx*, where *xx* is the T_l update period.

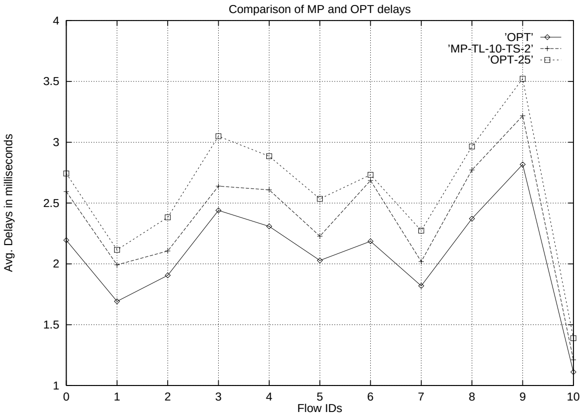

Figure 9: Delays of OPT and MP in CAIRN.

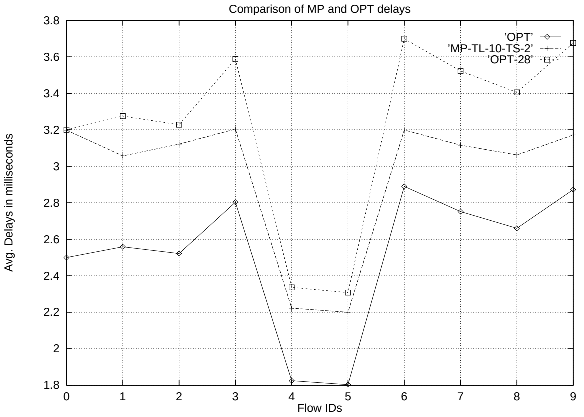

Figure 10: Delays of OPT and MP in NET1.

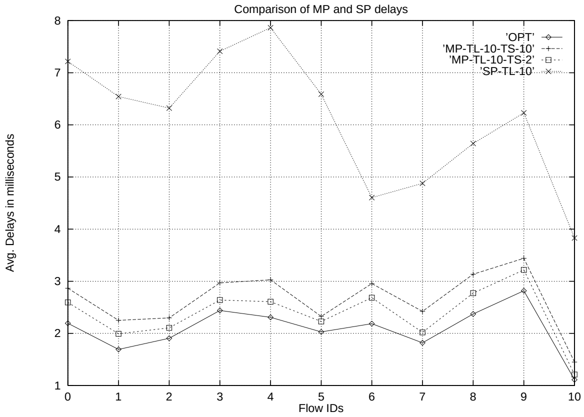

Figure 11: Delays of MP and SP in CAIRN.

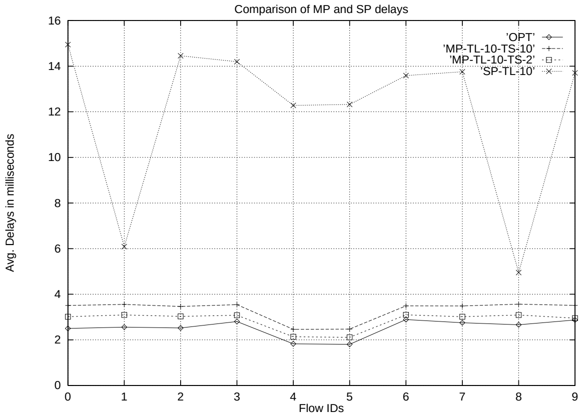

Figure 12: Delays of MP and SP in NET1.

5.1 Performance under Stationary Traffic

Fig. 9 shows the average delays of flows in CAIRN for OPT and MP routing. The flow IDs are plotted on the x-axis and average delays of the flows are plotted on the y-axis. Plot OPT-25 represents the 25% 'envelope', that is, the delays of OPT are increased by 25% to obtain the OPT-25 plot. As can be seen, the average delays of flows under MP routing are within the OPT-25 envelope. Similarly, in Fig. 10, the delays obtained using MP routing for NET1 are within 28% envelopes of delays obtained using OPT routing. We say delays of MP are 'comparable' to OPT if the delays of MP are within a small percent of those of OPT.

Fig. 11 compares the average delays of MP and SP for CAIRN. We observe that the delays of SP for some flows are two to four times those of MP. In Fig. 12, for NET1, MP routing performs even better; average delays of SP are as much as five to six times those of MP routing which is due to higher connectivity available in NET1. Also observe that, because of load-balancing used in MP, the plots of MP are less jagged than those of SP. MP routing performs much better than SP under high-connectivity and high-load environments. When connectivity is low or network load is light, MP routing cannot offer any advantage over SP.

5.2 Effect of Tuning Parameters T_l and T_s

The performance of MP depends on the update intervals T_l and T_s. The setting of T_l and T_s, however, is simple. They are local and can be set independently at each node without affecting convergence, unlike the global constant η which is critical for convergence of OPT. For CAIRN, Fig. 13 show the effect of increasing T_l when T_s and the input traffic is fixed. Observe that when T_l is increased from 10 to 20 seconds, the delays in SP have more than doubled, while the delays of MP remain relatively unchanged. This effect indicates that T_l can be made longer in MP without significantly effecting performance. This is significant, because sending frequent update messages consume bandwidth and can also cause oscillations under high loads. Similarly, for NET1, delays for SP increased significantly while there is negligible change in delays of MP as can be observed in Fig. 14, respectively. Our new routing framework provides the means for a trade-off between update messages and local load-balancing.

At T_s intervals, the load-balancing heuristics are executed, which are strictly local computations and require no communication. Therefore, T_s can be set according to the processing power available at the router. T_l can be made from a few times to orders of magnitude greater than T_s. In the simplest case, T_s can be set to the same value of T_l and *still* gain significant performance as shown in Figs. 11 and 12. In the figures, we observe that MP-TL-10-TS-10 is much closer to OPT than SP-TL-10. Just the long-term routes with load-balancing, without short-term routing parameter updates,

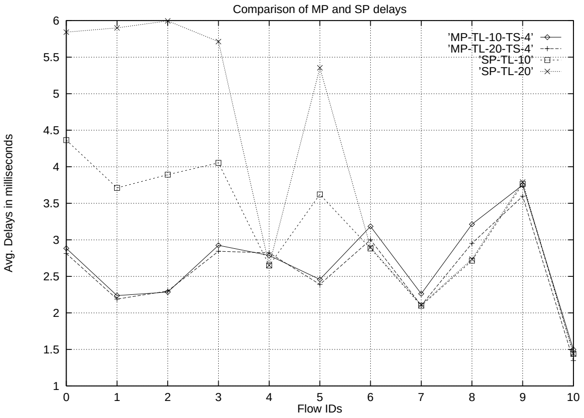

Figure 13: Delays when T_s is kept constant and T_l is increased in CAIRN.

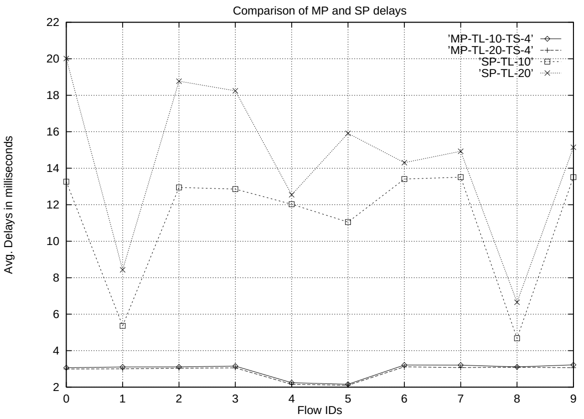

Figure 14: Delays when T_s is kept constant and T_l is increased in NET1.

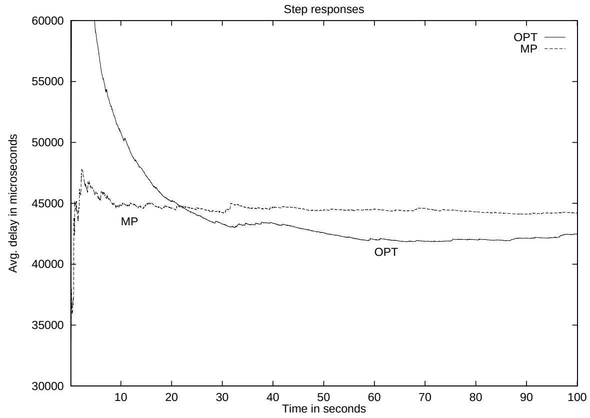

Figure 15: Step response in NET1 using OPT and MP routing.

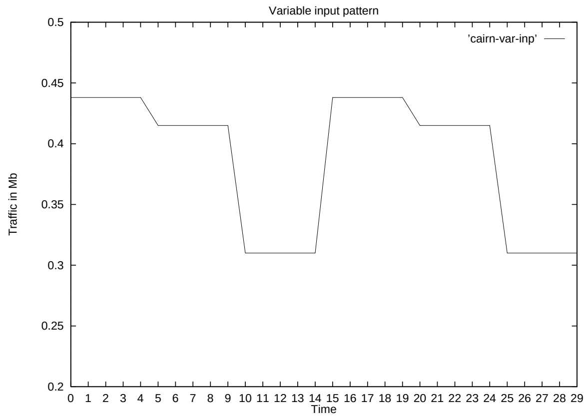

Figure 16: Variable input traffic pattern

seem to give significant gains; the major gains here are due to the mere presence of multiple successors and load-balancing. Our experience from simulations indicates that a T_l that is only a few times of longer than T_s suffices to gain significant benefits. This is great news, because it means that fine tuning of T_l and T_s is not important for our approach to be efficient.

5.3 Performance under Dynamic Traffic

It was stated earlier that OPT has very poor response to traffic fluctuations. This becomes evident in Fig. 15, which shows a typical response in NET1 when the flow rate is a step function (i.e.., the flow rate is increased from 0 to a finite amount at time 0). The dampened response of the network using MP indicates the fast responsiveness of MP, making it suitable for dynamic environments. Because OPT cannot respond fast enough to traffic fluctuations, it is impossible to find the optimal delays for dynamic traffic. However, we can find a reasonable lower bound if the input traffic pattern is predictable like the pattern shown in Fig 16, which shows only one cycle of the input pattern. To obtain a lower bound for this traffic pattern that represents 'ideal' OPT (the one that has instantaneous response) we first obtain the lower bound for each interval during which traffic is steady by running a separate off-line simulation with traffic rate that corresponds to that interval, and combine the results to obtain the *lower bound*. It is with this lower bound that we compare delays of MP. Fig. 17 shows the average delays of the flows for OPT, MP and SP routing. The results indicate that delays of MP routing are again in the comparable range of delays of an 'ideal' optimal-routing algorithm.

Ultimately, MP will be used in real networks where traffic is bursty at any time-scale; therefore, it is important to see how MP performs in that environment. We extracted 10 flows from the Internet traffic traces obtained from LBL [21] and used them as input for the 10 flows in the CAIRN. Fig. 18 shows the delays for SP and MP. We do not perform this simulation with OPT because Internet traffic is too bursty for OPT to converge. Observe that, except for flows 4, 6 and 8, delays of MP are much better than those of SP. The reason SP delays of these flows are better than those of MP is because of uneven distribution of load in the network and low loads in some sections of the network — in low-load environments SP can perform slightly better than MP. This can be easily rectified by modifying IH to use a small threshold cost for the best link, the crossing of which actually triggers the load-balancing scheme.

6 Conclusions

We have presented a practical approach to near-optimal delay routing in computer networks. To overcome the limitations of optimal routing algorithms, we proposed an approximation scheme and suggested algorithms that implement various components of the approximation. The resulting framework is both implementable in real networks and also provides delays that are close to those ob-

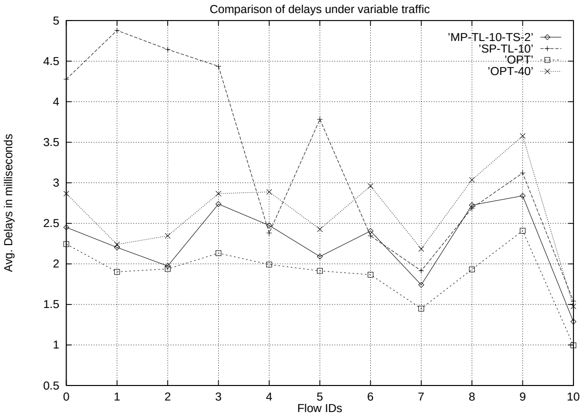

Figure 17: Delays under variable traffic in CAIRN.

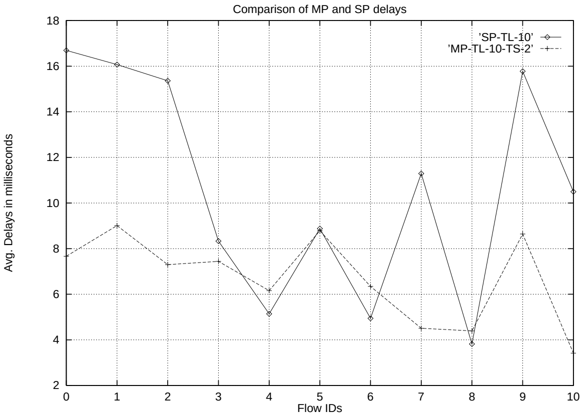

Figure 18: Delays under Internet traffic in CAIRN.

tainable using the Gallager's method. An important element of our framework is our generalization of sufficient conditions for loop-free routing, which are applicable to any type of routing algorithm.

We presented one of many possible implementations of the new routing framework. In doing so, we introduced the first link-state routing algorithm that provides multiple paths that are loop-free at every instant and that need not be of equal cost. We have shown through simulations that our implementation of the proposed framework performs significantly better than single-path routing, and that it offers delays that are within a small percentage of the lower bound delays under stationary traffic. The simulations are by no means exhaustive, but the results clearly indicate that the framework does offer potential for obtaining delays that compare with the optimal routing.

Additional work is needed to study flow allocation heuristics that are better suited for specific end-to-end services, e.g., trying to avoid out-of order packets for certain flows. Furthermore, our new routing framework opens up many interested research opportunities for quality-of-service (QoS) routing, because the loop-free invariant conditions on which it is based can be further constrained to satisfy different types of service. Similarly, because the traffic allocation heuristics depend on local rather than global parameters and, new heuristics can be defined to account for QoS constraints.

7 References

[1] R. Albrightson, J.J. Garcia-Luna-Aceves, and J. Boyle. EIGRP-A Fast Routing Protocol Based on Distance Vectors. *Proc. Networld/Interop 94*, May 1994.

[2] D. Bersekas and R. Gallager. Second Derivative Algorithm for Minimum Delay Distributed Routing in Networks. *IEEE Trans. Commun.*, 32:911–919, 1984.

[3] D. Bertsekas. Dynamic Behavior of Shortest-Path Algorithms for Communication Networks. *IEEE Trans. Automatic Control*, 27:60–74, 1982.

[4] J.B. Cain, S.L. Adams, M.D. Noakes, Tom Kryst, and E.L. Althouse. A Near-Optimum Multiple Path Routing Algorithm for Space-Based SDI Networks. *MILCOMM*, pages 29.3.1–29.3.7, 1987.

[5] D.G. Cantor and M. Gerla. Optimal Routing in a Packet-Switched Computer Network. *IEEE Trans. Computers*, 23:1062–1069, October 1974.

[6] C.G. Cassandras, M.V. Abidi, and D. Towsley. Distributed Routing with Onn-Line Marginal Delay Estimation. *IEEE Trans. Commun.*, 18:348–359, March 1990.

[7] E.W.Dijkstra and C.S.Scholten. Termination Detection for Diffusing Computations. *Information Processing Letters*, 11:1–4, August 1980.

[8] R. G. Gallager. A Minimum Delay Routing Algorithm Using Distributed Computation. *IEEE Trans. Commun.*, 25:73–84, January 1977.

[9] J.J. Garcia-Luna-Aceves. Loop-Free Routing Using Diffusing Computations. *IEEE/ACM Trans. Networking*, 1:130–141, February 1993.

[10] J.J. Garcia-Luna-Aceves and J. Behrens. Distributed, scalable routing based on vectors of link states. *IEEE Journal on Selected Areas in Communications*, October 1995.

[11] J.J. Garcia-Luna-Aceves and S. Murthy. A path-finding algorithm for loop-free routing. *IEEE/ACM Trans. Networking*, February 1997.

[12] J.J. Garica-Luna-Aceves and M. Spohn. Scalable link-state internet routing. *Proc. International Conference on Network Protocols*, October 1998.

[13] D.W. Glazer and C. Tropper. A new metric for dynamic routing algorithms. *IEEE Trans. Commun.*, 38:360–367, March 1990.

[14] C. Hendrick. Routing Information Protocol. *RFC*, 1058, june 1988.

[15] J. M. Jaffe and F. H. Moss. A Responsive Distributed Routing Algorithm for Computer Networks. *IEEE Trans. Commun.*, 30:1758–1762, July 1982.

[16] L. Klienrock. *Communication Nets: Stochastic Message Flow and Delay*. McGraw-Hill, New York, 1964.

[17] D. Kourkouzelis. *Multipath Routing Using Diffusing Computations, M.S. Thesis*. University of California, Santa Cruz, March 1997.

[18] J. M. McQuillan, I. Richer, and E. Rosen. The new routing algorithm for the arpanet. *IEEE Trans. Commun.*, 28:711–719, May 1980.

[19] P. M. Merlin and A. Segall. A Failsafe Distributed Routing Protocol. *IEEE Trans. Commun.*, 27:1280–1287, September 1979.

[20] J. Moy. OSPF Version 2. *RFC*, 1247, August 1991.

[21] V. Paxson, P. Danzig, J. Mogul, and M. Schwartz. Web page: ita.ee.lbl.gov/html/traces.html. *Lawrence Berkeley National Laboratory*, July 1997.

[22] M.I. Reiman and A. Weiss. Sensitivity analysis for simulations via likelihood rations. *Proc. 1986 Winter Simulation Conf.*, pages 285–289, 1986.

[23] A. Segall. The Modeling of Adaptive Routing in Data Communication Networks. *IEEE Trans. Commun.*, 25:85–95, January 1977.

[24] A. Segall and M. Sidi. A Failsafe Distributed Protocol for Minimum Delay Routing. *IEEE Trans. Commun.*, 29:689–695, May 1981.

[25] J. Spinelli and R. Gallager. Event Driven Topology Broadcast without Sequence Numbers. *IEEE Trans. Commun.*, 37:468–474, 1989.

[26] Z. Wang and J. Crowcroft. Shortest Path First with Emergency Exits. *Proc. of ACM SIGCOMM*, pages 166–176, 1990.

[27] W. T. Zaumen and J.J. Garcia-Luna-Aceves. Loop-Free Multipath Routing Using Generalized Diffusing Computations. *Proc. IEEE INFOCOM*, March 1998.

Appendix

Proof of Lemma 1: Let $A^i = \bigcup_{k \in N^i} A^i_k$ where A^i_k is the set of nodes in T^i_k. Since T^i_k is at least a $(n-1)$-hop minimum tree and node i can appear at most once in each of A^i_k, each A^i_k has at least $n-1$ unique elements. Therefore A^i has at least $n-1$ elements.

Let M^i_n be the set of $n-1$ nearest elements to node i in A^i. That is $M^i_n \subseteq A^i$ and $|M^i_n| = n-1$ and for each $j \in M^i_n$ and $v \in A^i - M^i_n$, $min\{D^i_{jk} + l^i_k | k \in N^i\} \leq min\{D^i_{vk} + l^i_k | k \in N^i\}$.

The theorem is proved in the following two parts:

1. Let G^i_n represent the graph constructed by MTU on line 4 and 5. (i.e., before applying Dijkstra on line 6). For each $j \in M^i_n$ there is a path $i \rightsquigarrow j$ in G^i_n such that its length is at most $\mathbf{D^{i,j}_n}$.
2. After running Dijkstra on G^i_n on line 6 in MTU, the resulting tree is at least an n-hop minimum tree.

Let us first assume Part 1 is true and prove Part 2, and then proceed to prove Part 1. From the statement in Part 1, for each node $j \in M^i_n$ there is a path $i \rightsquigarrow j$ in G^i_n with length at most $\mathbf{D^{i,j}_n}$. After running Dijkstra's algorithm, in the resulting graph, we can infer that there is a path $i \rightsquigarrow j$ with length at most $\mathbf{D^{i,j}_n}$. Because there are $n-1$ nodes in M^i_n, the tree constructed has at least n nodes with node i included. Accordingly, it follows from Property 1 that the tree constructed is at least an n-hop minimum tree.

Now we prove Part 1. Order the nodes in M^i_n in non-decreasing order. The proof is by induction on the sequence of elements in M^i_n as they are added to G^i_n. The base case is when G^i_n contains just one link $l^i_{m_1} = min\{l^i_k | k \in N^i\}$ and m_1 is the first element of M^i and $l^i_{m_1} = \mathbf{D^{i,m_1}_1}$. Let the statement hold for the first $m-1$ elements of M^i_n and consider the m-th element $j \in M^i_n$. Let K be the highest priority neighbor for which $D^i_{jK} + l^i_K = min\{D^i_{jk} + l^i_k | k \in N^i\}$. At Most $m-2$ nodes in T^i_K can have a smaller or equal distance than j, which implies path $K \rightsquigarrow j$ exists with at most $m-1$ hops. Let v be the neighbor of j in T^i_K. Then the path $K \rightsquigarrow v \rightarrow j$ has at most $m-1$ hops. Because T^i_K is at least a $(n-1)$-hop minimum tree, the cost of link $v \rightarrow j$ must agree with $\mathbf{G}$. Since $D^i_{vK} + l^i_K < D^i_{jK} + l^i_K$, from our inductive hypothesis , there is a path $i \rightsquigarrow v$ in G^i_n such that the length is at most $\mathbf{D^{i,v}_n}$.

Now we need to show that the preferred neighbor for v is also K, so that the link $v \rightarrow j$ will be included in the construction of G^i_n, thus ensuring the existence of the path $i \rightsquigarrow j$ in G^i_n. If some other neighbor K' instead of K is the preferred neighbor for v, then one of the following two cases should have occurred: (a) $D^i_{vK'} + l^i_{K'} < D^i_{vK} + l^i_K$ or, (b) $D^i_{vK'} + l^i_{K'} = D^i_{vK} + l^i_K$ and priority of K' is greater than priority of K.

Case (a): If $D^i_{vK'} + l^i_{K'} < D^i_{vK} + l^i_K$, then given that $D^i_{jK} + l^i_K \leq D^i_{jK'} + l^i_{K'}$ it follows that the path $v \rightsquigarrow j$ in $T^i_{K'}$ is greater than cost $v \rightarrow j$ in $\mathbf{G}$ which implies that $T^i_{K'}$ is not a $(n-1)$ hop minimum tree – a contradiction to our assumption! Therefore, $D^i_{vK} + l^i_K = min\{D^i_{vk} + l^i_k | k \in N^i\}$.

Case (b): Let Q_j be the set of neighbors that give the minimum distance to j, i.e., for each $k \in Q_j$, $D^i_{jk} + l^i_k = min\{D^i_{jk} + l^i_k | k \in N^i\}$. Similarly, let Q_v be such that for each $k \in Q_v$, $D^i_{vk} + l^i_k = min\{D^i_{vk} + l^i_k | k \in N^i\}$. If $k \in Q_v$ and $k \notin Q_j$, then it follows from the same argument used in case (a) that $v \rightsquigarrow j$ in T^i_k is greater than $v \rightarrow j$ in $\mathbf{G}$, which implies that T^i_k is not a $(n-1)$-hop minimum tree – a contradiction to our assumption again. Therefore, $Q_v \subseteq Q_j$. Also, from the same argument used in case (a) above it can be inferred that $K \in Q_v$. Because K has the highest priority among all members of Q_j and $Q_v \subseteq Q_j$, and because $k \in Q_v$, K must also have the highest priority among all members of Q_v. This proves that $v \rightarrow j$ will be included in the construction of G^i_n. Because $\mathbf{D^{i,v}_n} + \mathbf{d_{vj}} = \mathbf{D^{i,j}_n}$ in $\mathbf{G}$, where $\mathbf{d_{vj}}$ is the final cost of link $v \rightarrow j$, and the length of $i \rightsquigarrow v$ in G^i_n is less than $\mathbf{D^{i,v}_n}$ from our inductive hypothesis, we obtained that the length of $i \rightsquigarrow j$ in G^i_n less than $\mathbf{D^{i,j}_n}$. This proves Part 1 of the theorem. □

Proof of Theorem 3: Let t_n be the time when FD^i_j is updated for the n-th time. The proof is by induction on the time intervals $[t_n, t_{n+1}]$. As inductive hypothesis assume that

$$FD^i_j(t) \leq D^k_{ji}(t) \qquad k \in N^i, t \leq t_n \tag{25}$$

We show that

$$FD^i_j(t) \leq D^k_{ji}(t) \qquad t \in [t_n, t_{n+1}] \tag{26}$$

We observe from the description of MPDA in Fig. 4 that, when FD^i_j is updated at lines 2b and 3c, D^i_j is also updated at lines 2a and 3b respectively. We also observed that FD^i_j is updated only during state transitions, and regardless of whether the transition is from PASSIVE-to-ACTIVE or from ACTIVE-to-PASSIVE, the Eq. (27) below is true. Note that there is an implicit PASSIVE state between two back-to-back ACTIVE states.

$$FD^i_j(t_n) \leq min\{D^i_j(t_{n-1}), D^i_j(t_n)\} \tag{27}$$

Let t' be the time when LSU sent by i at t_n is received and processed by neighbor k. Because of the non-zero propagation delay across any link, t' is such that $t_n < t' < t_{n+1}$. We then have

$$D^k_{ji}(t') = D^i_j(t_n) \tag{28}$$

Because FD^i_j is modified at t_n and then remains unchanged within (t_n, t_{n+1}), we obtain from Eq. (25) that

$$FD^i_j(t) \leq D^k_{ji}(t) \qquad t \in [t_n, t') \tag{29}$$

From Eqs. (27) and (28) we obtain the following.

$$FD^i_j(t) \leq D^k_{ji}(t) \qquad t \in [t', t_{n+1}) \tag{30}$$

From Eq. (29) and (30) we have

$$FD^i_j(t) \leq D^k_{ji}(t) \qquad t \in [t_n, t_{n+1}) \tag{31}$$

At t_{n+1}, again from the design of MPDA we have,

$$FD^i_j(t_{n+1}) \leq min\{D^i_j(t_n), D^i_j(t_{n+1})\} \tag{32}$$

Also, because propagation delays are positive, node k at t_{n+1} cannot yet have the value $D^i_j(t_{n+1})$. So, we have

$$D^k_{ji}(t_{n+1}) = D^i_j(t_n) \tag{33}$$

Combining Eq. (33) and (32) for time t_{n+1}, we get

$$FD^i_j(t_{n+1}) \leq D^k_{ji}(t_{n+1}) \tag{34}$$

and Eq. (26) follows from combining Eqs. (31) and (34).

Because $FD^i_j(t_0) \leq D^k_{ji}(t_0)$ at initialization, from induction we have that $FD^i_j(t) \leq D^k_{ji}(t)$ for all t. Given that the successor sets are computed based on FD^i_j, it follows that the LFI conditions are always satisfied. According to Theorem 1, this implies that the successor graph SG_j is always loop-free. □