

# Distributed Advance Network Reservation with Delay Guarantees

Niloofar Fazlollahi

Dept. of Electrical and Computer Engineering  
Boston University  
Boston, USA  
nfazl@bu.edu

David Starobinski

Dept. of Electrical and Computer Engineering  
Boston University  
Boston, USA  
staro@bu.edu

**Abstract**—New architectures have recently been proposed and deployed to support end-to-end advance reservation of network resources. These architectures rely on the use of a centralized scheduler, which may be unpractical in large or administratively heterogeneous networks. In this work, we explore and demonstrate the feasibility of implementing distributed solutions for advance reservation. We introduce a new distributed, distance-vector algorithm, called *Distributed Advance Reservation* (DAR), that provably returns the earliest time possible for setting up a connection between any two nodes. Our main findings in this context are the following: (i) we prove that *widest path routing* and *path switching* (i.e., allowing a connection to switch between different paths) are necessary to guarantee earliest scheduling; (ii) we propose a novel approach for loop-free distributed widest path routing, leveraging the recently proposed DIV framework. Our routing results directly extend to on-demand QoS routing problems.

**Keywords**—Cloud/grid computing; scheduling; routing; performance guarantees;

## I. INTRODUCTION

New generations of scientific collaborative applications require on-line analysis of immense volume of data at distributed sites spanning wide geographical domains. For instance, experiments run on the Large Hadron Collider project at CERN in Geneva [1] will generate up to 40 Terabytes per day that must be immediately distributed to collaborating research labs around the world for purpose of storage and analysis. Success of these and other emerging grid and cloud computing applications relies on high-speed underlying networks that supports quick transfer of bulk data to distributed sites.

Networks based on best-effort TCP/IP do not provide appropriate levels of guarantee and flexibility required by modern grid applications. Hence, new network architectures are currently being deployed to provide users with the ability to reserve in advance *dedicated* reliable circuits. For example, ESnet has recently set up the so-called Science Data Network (SDN) specifically designed to support advance reservation [2]. SDN as well as other similar advance reservation architectures are managed centrally, i.e., a central scheduler performs advance reservations based on knowledge of the entire topology of its domain. Such solutions do not scale to large network domains or administratively

heterogeneous networks, where network administrators do not wish to disclose internal topology information.

Motivated by current limitations of centralized approaches, our goal in this paper is to identify fundamental constraints and requirements for implementing distributed advance reservation with guaranteed delay performance. By distributed, we mean that the calculation of routes and scheduling of connections are performed by routing nodes rather than on a central computer. By delay guarantees, we mean that the time elapsed from the moment the request is placed until the start of the corresponding connection is minimized. We refer to such a property as achieving minimal delay or earliest scheduling. Our objective is to constructively show the feasibility of implementing distance vector routing, whereby each node only maintains a *successor* (best next hop based on some metric) and a corresponding metric value for each destination and each time slot (a time slot roughly corresponds to a period of time delineated by connection set-up or release events; a more precise definition will be given in Section III).

We divide the task of devising a distributed advance reservation algorithm into two sub-problems:

- 1) Scheduling: assuming that every node knows its successor and the metric value to all destinations at all time slots, find and reserve resources at the earliest time interval that can accommodate a connection satisfying the desired user criteria.
- 2) Routing: calculate a successor per each destination and time slot at every node. This way, every node knows its successor upon the arrival of a request.

Given the constraints imposed by the data structure available at nodes, our contributions are the following:

- 1) We show that both *widest path* routing, i.e., routing on the path with largest end-to-end bandwidth, and *path switching*, i.e., allowing connection to switch between different paths, are necessary to ensure earliest scheduling (minimal delay) of connections.
- 2) We prove that a simple implementation of *distributed asynchronous Bellman-Ford* for widest path routing [3] may suffer from *permanent* routing loops in a time-varying network supporting connection set-ups

and releases.

- 3) We propose a distributed loop-free routing module called the *Successor Selection Module* (SSM) that provably computes the widest path for each pair of nodes and each time slot, leveraging a recently proposed loop-prevention paradigm called *Distributed Path Computation with Intermediate Variables* (DIV) [4].
- 4) Based on the principles of widest path routing and path switching and using the routing information provided by SSM, we devise an algorithmic solution, called *Distributed Advance Reservation* (DAR), that provably guarantees minimal delay for each arriving request.

The rest of this paper is organized as following. We briefly review related work in Section II. In Section III, we explain our notation and assumptions and define the data structure maintained at nodes. Section IV explains the DAR algorithm and is divided into two parts: (i) scheduling; and (ii) routing. In the first part, after analyzing the requirements imposed by earliest scheduling, we present the DAR algorithm and prove its properties. In the second part, we first bring negative results showing the existence of permanent routing loops in naive implementation of distributed Bellman-Ford for widest path routing. We then review the DIV loop prevention mechanism and judiciously adapt it to our specific problem. We develop the SSM routing algorithm and prove its theoretical properties. We conclude the paper in Section V

## II. RELATED WORK

Our work relates to several areas, namely algorithms for advance reservation, distributed QoS routing and loop prevention. We review each of them next.

Most work regarding advance reservation algorithms focuses on centrally managed architectures. For example, Refs. [5] and [6] introduce centralized advance reservation algorithms that satisfy various multi-criteria optimizations. Authors in [7] analyze the effect of advance reservation on the complexity of path selection. The mentioned references all share a roughly similar time slicing (or time slots) approach, that we adopt in this paper as well. On the other hand, there appears to be little work in the literature on distributed network advance reservation, especially with guaranteed performance. Some references focus on the signaling aspects of distributed advance reservation. For example Ref. [8] discusses possible modification to RSVP protocol to support advance reservation in ATM networks.

Quality-of-Service is an important aspect for real-time and streaming work and much work studies QoS routing from various angles. Most work on QoS routing employs link state routing, especially when it comes to widest path routing [9]. Ref. [10] investigates the properties that QoS criteria must possess to allow for hop-by-hop routing and the computation of optimal paths using a generalized version of the Dijkstra algorithm.

Ref. [11] studies multi-criteria QoS routing and presents several combinations of criteria for which the problem is proved to be NP-complete. Refs. [11, 12] study hop-by-hop widest path routing based on distance vector structure. The algorithms are assumed to run synchronously (an assumption which we do not make) since all nodes must always be at the same stage of the execution. More critically, their solutions do not consider how to handle updates resulting from link bandwidth changes. We show in this work that such updates can trigger permanent routing loops, unless they are properly addressed.

Distributed distance-vector routing is notoriously known to suffer from routing loops in dynamic networks. In the case of shortest-path routing, such loops may result into the infamous count-to-infinity problem leading to slow convergence. For the case of widest-path routing, we will show that the problem is more severe, namely no convergence at all. Refs. [13, 14] introduce loop-free shortest path algorithms extended from the Bellman-Ford algorithm [3]. Specifically, Ref. [13] proposes an algorithm called DUAL which restricts selection of the successor to a set of neighbors called the *feasible successor set* and triggers a synchronous update procedure called diffusing computation to synchronize a group of nodes in case of any change. Ref. [14] defines a pair of invariant conditions called *Loop Free Invariant* (LFI) at each node based on its cost to destination and that of its neighbors. The LFI conditions prevent formation of transient loops. The update mechanism is similar to that of DUAL.

The previous references considered the specific case of shortest path routing. Ref. [4] offers a framework called DIV for loop prevention that can be used in conjunction with other metrics. DIV is roughly a hybrid of the DUAL and LFI algorithms. We explain DIV in detail in the sequel. Here, we outline some of its advantages compared to the previous references: (i) it supports multi-path routing, (ii) it has more relaxed feasibility conditions compared to the DUAL algorithm and hence triggers synchronous updates less frequently (iii) it can handle multiple overlapping updates simultaneously.

## III. MODEL

### A. Notation

We consider a weighted undirected network modeled with a graph  $G$  consisting of a set of nodes  $V$  and a set of links  $E$ . The graph is dynamic meaning that weights change over time. Nodes represent hosts and routers and links are reliable channels connecting the nodes. We denote  $e_{ij}$  the link connecting node  $i \in V$  to node  $j \in V$ . We denote  $N(i)$  the set of neighbors of node  $i$ .

Connection requests arrive randomly over time across the network. Each request specifies the transmission source  $s$ , the transmission sink  $d$ , a desired bandwidth  $B$  and a connection duration  $T$ . Users can restrict the connection start

time to an interval  $[t_a, t_b]$ . Otherwise,  $t_a = t_{now}$  and  $t_b = \infty$  where  $t_{now}$  is the present time.

Because of advance reservation of connections, there should exist a common reference time frame throughout the network. Hence, we assume coarse-grained synchronization (e.g., on the order of seconds) between clocks at different nodes to agree on the set-up time and release of connections. We emphasize however that our routing algorithms, and SSM in particular, can be run in a fully asynchronous manner.

We associate a weight  $w[e_{ij}]$  with each link  $e_{ij}$  based on the desired routing optimization criterion. Examples of link weight are length (denoted  $l[e_{ij}]$ ) which in our settings is equivalent to *link hop count* (equal to 1) and bandwidth which is the bandwidth available on the link (denoted  $b[e_{ij}]$ ).

A path from node  $i$  to node  $j$  consists of an ordered list of one or more consecutive links that connect  $i$  to  $j$  and is denoted  $P_{ij}$ . The *Path weight* is a combination of weights of links forming the path. If the path weight is based on bandwidth then the path weight is given by  $\min_{e_{ij} \in P_{sd}} \{b[e_{ij}]\}$  for any given path  $P_{sd}$ . If the path weight is based on length then the path weight is  $\sum_{e_{ij} \in P_{sd}} l[e_{ij}]$  for any given path  $P_{sd}$ . A path with the optimal weight among all paths from  $s$  to  $d$  is called the *optimal path*.

We denote  $w_{ij}$  the estimated path weight from  $i$  to  $j$  by our routing algorithm. Likewise, we denote  $b_{ij}$  and  $l_{ij}$  the estimated path bandwidth and the estimated path length respectively. The optimal values of the above variables are denoted  $w_{ij}^*$ ,  $b_{ij}^*$  and  $l_{ij}^*$ .

The *successor* of node  $i$  to destination  $d$  on some path  $P_{id}$  is defined as the immediate next hop of  $i$  on the path and denoted  $\pi_{id}$ . If  $j = \pi_{id}$  then node  $i$  is the *predecessor* of  $j$ . Ancestor of a given node with respect to destination  $d$  is defined as a node that connects to  $i$  through a chain of consecutive successors. If node  $k$  is an ancestor of  $i$ , then  $i$  is called a descendant of  $k$ .

## B. Assumptions

The statements proposed in this paper are correct under the following assumptions:

- 1) Communication links are reliable.
- 2) Links never fail.
- 3) There is no Byzantine behavior at nodes.
- 4) There are no conflicts between requests over reservation.
- 5) Clocks at different nodes are coarsely synchronized.
- 6) Successor calculations for every destination have stabilized by the time a new request arrives.

This does not imply that there is no way we can resolve or alleviate these issues but rather that they are commonplace to all distributed network algorithms. Much work in literature has addressed them and the solutions are extensible to our particular case with advance reservation as well [15–17].

## C. Node data structures

In this section, we describe the data structures maintained by nodes and illustrate them with an example. Here, we detail only part of the data structure at nodes which is relevant to the performance of the DAR algorithm. This part is consistent with the usual definition of distance vector routing. In section IV-B, we add additional variables used uniquely to prevent formation of loops.

To accommodate advance reservation, every node should maintain relevant information regarding network state per all future times. Since the available link bandwidths change over time because of scheduled set-up or release of connections, the variables maintained by nodes are time dependent. To simplify the analysis, we divide the continuous time axis into discrete slots delineated by transition instances in the values of the node variables. Therefore, node variables remain fixed during each time slot. We denote  $t_1^{(id)}, t_2^{(id)}, \dots, t_n^{(id)}$  the slot transition instances for node  $i$  with respect to destination  $d$ , where  $t_1^{(id)}$  is the present time ( $t_{now}$ ) and  $t_n^{(id)} = \infty$ . Note that the time slots are not necessarily the same for different source destination pairs. They are not fixed and pre-determined but formed dynamically with scheduled set-up and release of connections.

Every node  $i$  maintains the following state variables per future time slot for each destination  $d$ : (i) a successor for destination  $d$ , denoted  $\pi_{id}(t)$  (ii) an estimate of the optimal path weight from  $i$  to  $d$  denoted  $w_{id}^*(t)$  (iii) an estimate of the optimal path weight denoted  $w_{jd}^*(t)$  from  $j$  to  $d$  for all neighbors  $j \in N(i)$  (iv) the link weight  $w[e_{ij}](t)$  from  $i$  to each neighbor  $j \in N(i)$ . The last item does not depend on the destination. This is consistent with the standard data structure used in distance vector routing with the difference that our structure must include future states to support advance reservation. Note that although all of the above variables depend on time  $t$ , they are fixed during each time slot.

We show in the next section that given the presented data structure at each node, the successors must be selected based on widest path optimization to guarantee the earliest connection start time.

### Example.

Figure 1.a shows a network consisting of four nodes and four undirected links. Link bandwidths change over time as depicted in Figure 1.b

Table I depicts the node data structures related to the network of Figure 1.a. This table shows only the data used directly by algorithm DAR. Each node maintains for each destination and time slot its successor, and the estimated path bandwidth.

We present a case study regarding node  $B$ . There are two time slots for destination  $D$ :  $\pi_{BD}(t) = C$  and  $b_{BD}^*(t) = 10$  Gbit/s for time  $t$  from  $t_1^{BD} = 12:00$  am to  $t_2^{BD} = 2:00$  am and  $\pi_{BD}(t) = D$  and  $b_{BD}^*(t) = 20$  Gbit/s for time  $t$  from

	next hop	path bandwidth	time in am (hours)
source A	destination A		
	-	-	12:00-∞
	destination B		
	B	10	12:00-∞
	destination C		
	B	10	12:00-∞
destination D			
	B	10	12:00-2:00
	B	20	2:00-∞
source B	destination A		
	A	10	12:00-∞
	destination B		
	-	-	12:00-∞
	destination C		
	C	10	12:00-∞
destination D			
	C	10	12:00-2:00
	D	20	2:00-∞
source C	destination A		
	B	10	12:00 - ∞
	destination B		
	B	10	12:00 - ∞
	destination C		
	-	-	12:00 - ∞
destination D			
	D	10	12:00 - ∞
source D	destination A		
	C	10	12:00-2:00
	B	10	2:00-∞
	destination B		
	C	10	12:00-2:00
	B	20	2:00-∞
	destination C		
	C	10	12:00-∞
destination D			
	-	-	12:00-∞

Table I

NODE DATA STRUCTURES FOR WIDEST PATH SUCCESSOR SELECTION.

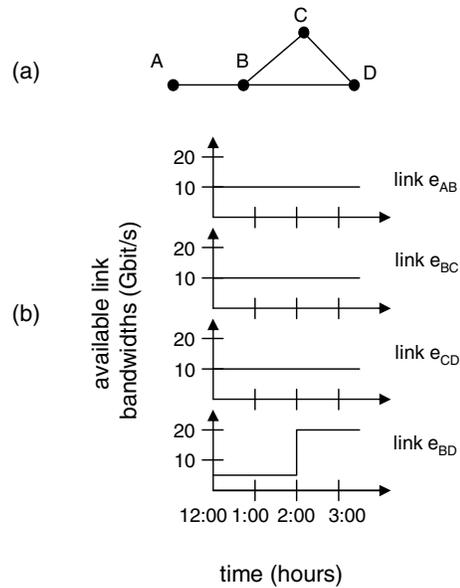


Figure 1. The figure shows a network with changing link state: (a) an undirected graph of four nodes representing a network (b) available bandwidth on links  $e_{AB}$ ,  $e_{BC}$ ,  $e_{CD}$ , and  $e_{BD}$  over time. Since the graph is undirected every link can be presented with two formats. For example,  $e_{AB}$  and  $e_{BA}$  represent the same link.

$t_2^{BD} = 2:00$  am to  $t_3^{BD} = \infty$ . However at the same node  $B$  there is only one time slot for destination  $C$ :  $\pi_{BC}(t) = C$  and  $b_{BC}^*(t) = 10$  Gbit/s for time  $t$  from  $t_1^{BC} = 12:00$  am to  $t_2^{BC} = \infty$ .

#### IV. DAR ALGORITHM

Our objective is to devise a distributed algorithm guaranteeing that each request is provided with minimal delay. We divide the problem of devising such an algorithm into two sub-problems, one for scheduling and one for routing. As shown next, these two sub-problems are not fully dissociated.

In the first part, after stating the routing requirements imposed by delay optimization, we introduce an algorithm called DAR that provably returns the earliest connection start time. In the second part, after highlighting the fundamental problems involved in distributed widest path routing, we briefly describe a recently proposed approach called DIV that provides a generic framework to solve loop issues in distributed routing. One of our main contributions is to introduce an algorithm called SSM that judiciously selects adequate optimization metrics for DIV to ensure loop-free calculation of routes. We conduct a performance analysis of SSM and prove its correctness. Note that the DAR algorithm relies on the routing tables computed by SSM.

##### A. Scheduling

We start this sub-section by mentioning the constraints imposed on routing because of the earliest scheduling op-

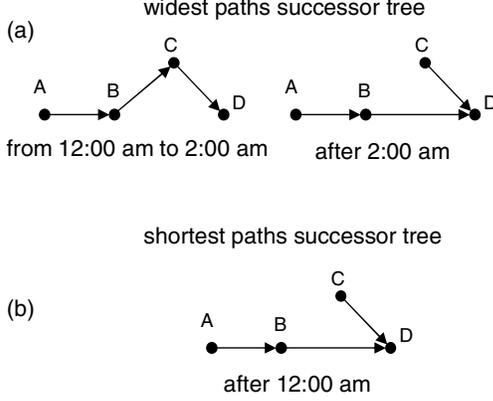


Figure 2. Illustration of various successor selection criteria regarding the graph of figure 1: (a) successor tree for destination  $D$  based on widest path optimization (b) successor tree for destination  $D$  based on shortest path optimization. Note that successor trees for destinations  $A$ ,  $B$  and  $C$  should be formed separately in a similar way.

timization. For added clarity, we occasionally refer to the example network of Figure 1 and Table I with concrete examples.

1) *Widest routing requirement:* Figures 2.a and 2.b depict the successor graphs based on widest path optimization and shortest path optimization respectively for destination  $D$  of the network illustrated in Figure 1.a.

Let us consider a particular example. Assume a request arrives at 12:00 am for a 10 Gbit/s connection lasting 3 hours from node  $A$  to  $D$ .

According to Figure 2.b, the shortest path successors of  $A$  and  $B$  toward  $D$  are  $\pi_{AD}(t) = B$  and  $\pi_{BD}(t) = D$  at all times  $t \geq 12:00$  am leading to path  $(e_{AB}, e_{BD})$ . We observe that it is not possible for a connection requesting 10 Gbit/s to start at 12:00 am because bandwidth of the mentioned path is 5 Gbit/s from 12:00 am to 2:00 am. The earliest time to start the connection is 2:00 am for that path, though we could have started the connection at 12:00 am using path  $(e_{AB}, e_{BC}, e_{CD})$ . This simple example reflects a restriction that exists with distributed hop-by-hop routing algorithms in general. With shortest path successor selection, longer paths with larger bandwidth are ignored. We prove:

*Theorem 4.1:* With the given node data structure and hop-by-hop routing paradigm, widest path routing is required to achieve earliest scheduling.

*Proof:* The proof is by contradiction. Consider a network represented with graph  $G(V, E)$ . All nodes in  $V$  store a successor and the optimal path weight based on some optimization criteria per every destination and time slot. Assume a request  $(s, d, B, T, t_a, t_b)$  arrives at a given node  $s$ .

We denote  $\Gamma$  the set of all hop-by-hop routing algorithms based on the mentioned data structure. Assume the earliest achievable connection start time among all algorithms in  $\Gamma$  is  $t' \geq t_a$ . This implies there exists at least one path from  $s$

to  $d$  with bandwidth at least  $B$  starting at  $t'$ . Assume  $\alpha$  is an algorithm in  $\Gamma$  that always returns the earliest connection start time. Now, unless the selected path between  $s$  and  $d$  by  $\alpha$  is the widest at any time  $t \in [t', t' + T]$ , one can always come up with a bandwidth request  $B$  that exceeds the estimated path bandwidth during  $[t', t' + T]$  by algorithm  $\alpha$ . Thus, the connection start time by algorithm  $\alpha$  would be later than  $t'$  which contradicts our assumption that  $\alpha$  returns the earliest connection start time. ■

2) *Path switching:* We reconsider the network of Figure 1.a and the example request from  $A$  to  $D$  described in previous sub-section. According to the table, based on the widest path optimization, we have for  $t \geq 12:00$  am,  $\pi_{AD}(t) = B$  and  $w_{AD}^*(t) = b_{AD}^*(t) = 10$  Gbit/s. Hence it seems natural to assign the requested connection to the time interval 12:00 am to 3:00 am. However, according to the same table, for  $t \in [12:00, 2:00]$  am,  $\pi_{BD}(t) = C$  and for  $t \geq 2:00$  am  $\pi_{BD}(t) = D$ . Thus the successors in table I do not provide a fixed path for connection during 12:00 am to 3:00 am. This restriction is concealed at node  $A$ . Therefore,  $A$  cannot decide to start the connection at 2:00 am to avoid the inconsistent paths throughout one connection.

We overcome the mentioned restriction with the aid of path switching. With path switching, a connection is not restricted to use the same path over all its duration, i.e., it can switch paths. Hence, we reserve in advance the paths as well as relevant switching information. The concept of path switching was first introduced in [5] in the context of centralized routing with advance reservation.

Back to our example, we see that one can reserve a connection from 12:00 am to 3:00 am from  $A$  to  $D$  with bandwidth of 10 Gbit/s provided that during interval [12:00 am, 2:00 am] the reserved path is  $(e_{AB}, e_{BC}, e_{CD})$  and during interval [2:00 am, 3:00 am] the reserved path is  $(e_{AB}, e_{BD})$ .

3) *Presentation of DAR algorithm:* Referring to the node data structure presented earlier, assume that the estimated widest path bandwidth  $b_{id}^*(t)$  from every node  $i$  to  $d$  is optimal (widest). Based on this assumption, we want to automatize the process illustrated above for finding the earliest connection start time per each arriving request.

We present next the scheduling component of DAR which provably returns the earliest connection start time and a path (or sequence of paths in case of path switching).

Upon arrival of a request  $R = (s, d, B, T, t_a, t_b)$ , DAR searches for a point in time  $t_1^R$  within the time frame  $[t_a, t_b]$  such that the bandwidth constraint is satisfied, i.e.,  $b_{sd}^*(t_i^R) \geq B$  for  $i = 1, \dots, k$ . Times  $t_1^R$  and  $t_{k+1}^R = t_1^R + T$  correspond to the scheduled start and end time of the connection respectively. Times  $t_2^R, \dots, t_k^R$  are the scheduled path switching instances.

Every node, such as  $s$ , must regularly update its time slot structure  $t_1^{(sd)}, \dots, t_n^{(sd)}$  since the first element of the list

must always correspond to the present time  $t_{now}$ . Note that  $t_i^{(sd)}$  denotes starting time of the  $i$ th slot corresponding to destination  $d$  at node  $s$  and it not necessarily the same as  $t_i^R$  which is the scheduled time of  $i$ th switching of connection  $R$ . The update process at node  $s$  consists of removal of every time slot  $k$  whose start time,  $t_k^{(sd)} < t_{now}$  and updating the indices of all remaining time slots so that the first slot is indexed 1, then setting  $t_1^{(sd)} = t_{now}$ . Node  $s$  should clear the data corresponding to every removed time slot.

**Algorithm DAR run at node  $s$ :**

- 1) Upon arrival of a request  $R = (s, d, B, T, t_a, t_b)$ ,
  - a) Initialize connection start time  $t_1^R$  to  $t_a$
  - b) If  $b_{sd}^*(t) \geq B$  does not hold at all times  $t \in [t_1^R, t_1^R + T]$  then,
    - i) If  $t_1^R \geq t_b$ ,
      - Reject the request
    - ii) Otherwise,
      - Find a slot  $j$  with minimum value of  $j$  such that  $t_j^{(sd)} > t_1^R$  and set  $t_1^R$  to  $t_j^{(sd)}$
      - Go back to step 1b
  - c) If request is admissible, reserve connection:
    - i) Reserve the requested bandwidths from corresponding links and store successors on the scheduled path(s)
- 2) Go to step 1

After a request is found feasible, DAR runs the reservation process at step 1c: we denote  $P_{sd}(t)$  the path constructed by consecutive successors from  $s$  to  $d$ . Every node situated on path  $P_{sd}(t_i^R)$  which is scheduled for  $[t_i^R, t_{i+1}^R]$  for  $i = 1, \dots, k$  stores its successor for the given time interval and reserves the requested bandwidth  $B$  from the link to its successor during the same interval.

4) *Performance analysis:* We next prove the most important property of DAR.

**Theorem 4.2:** DAR provides the earliest connection start time per arriving request.

*Proof:* Assume the path  $P_{sd}(t)$  constructed by consecutive successors from node  $s$  to destination  $d$  is the widest path from  $s$  to  $d$  at every time  $t$  (we will prove this in theorem 4.10).

We consider two cases: (i) If we only consider  $P_{sd}(t)$ , then DAR chooses the earliest time  $t_1^R$  to set up the connection because according to step 1(b)ii, DAR always investigates the earliest slot  $j$  after  $t_a$  that is followed by a continuous duration  $T$  with sufficient resources between  $s$  and  $d$ . (ii) On the other hand, assume there exists a path  $P'_{sd}(t)$  from  $s$  to  $d$  other than  $P_{sd}(t)$ , with available bandwidth  $B$  or more during  $t \in [t_1^{R'}, t_1^{R'} + T]$  where  $t_a \leq t_1^{R'} < t_1^R$ . Since  $P_{sd}(t)$  has the largest available bandwidth at any time, bandwidth of  $P_{sd}(t)$  is at least equal to the bandwidth of  $P'_{sd}(t)$  which exceeds  $B$  for  $t \in [t_1^{R'}, t_1^{R'} + T]$ . But in this case DAR would have selected time  $t_1^{R'}$  at step 1(b)ii. ■

We may improve the performance of DAR by adding a further selection criterion: we choose the successor that acknowledges the shortest path length among all widest path successors. Although this may improve performance by encouraging shorter paths compared to random widest path selection, we prove:

**Lemma 4.3:** Given the presented node structures, shortest-widest-earliest path optimization is not feasible.

*Proof:* We prove this lemma with a negative example. Consider again the example network of Figure 1.a with the same bandwidth-time plots for links  $e_{BC}$ ,  $e_{CD}$  and  $e_{BD}$  but assume  $e_{AB}$  has constant bandwidth of 5 Gbit/s after 12:00 am. If we select the successor acknowledging the shortest among all widest paths, then  $\pi_{BD}(t) = C$  for  $t \in [12:00, 2:00]$  am. We have  $\pi_{AD}(t) = B$  at all times  $t \geq 12 : 00$  am since this is the only option. Given this,  $P_{AD} = (e_{AB}, e_{BC}, e_{CD})$  with bandwidth 5 Gbit/s from 12:00 to 2:00 am and the shorter path  $(e_{AB}, e_{BD})$  with the same bandwidth of 5 Gbit/s during the same time interval is ignored. This proves that using this data structure selection of the shortest-widest and therefore the shortest-widest-earliest path is not guaranteed. ■

### B. Pre-computation of routes

In the previous section we have assumed that nodes know the appropriate successor to every destination per all future times. We proved that given our particular node data structure only the widest path to destination guarantees earliest scheduling.

In this section we present a distributed algorithm for selection of successors which we refer to as the *Successor Selection Module* (SSM). SSM runs at every node independent of other nodes and DAR. First we explain the challenges of achieving widest paths given such a data structure. Then we prove that the paths tentatively constructed by SSM converge to the widest for every destination. Note that DAR relies on the steady state results produced by SSM.

**Notation:** to simplify the presentation, we discard the time dimension throughout this section and present all algorithms as if they were on-demand. Every algorithm presented here can be considered as an advance path calculation for a given time slot and can be directly extended to all future time slots. Therefore, we eliminate the time argument from our notation in what follows since node variables remain unchanged during every slot.

The problem of successor selection for distributed hop-by-hop routing in networks has been visited frequently in the literature. The common approach is using a distributed asynchronous version of the standard Bellman-Ford algorithm [3, 15, 18]. However, much of the focus of prior work has been on *shortest path* routing rather than any other metric for the reason explained next.

1) *Routing loops:* Assume we modify the distributed asynchronous shortest path Bellman-Ford algorithm for

widest path optimization by replacing link lengths and path lengths by link bandwidth and path bandwidths respectively and by adjusting the relaxation equation accordingly.

In our presentation below, variable  $b_{jd}^{(i)}$  for  $j \in N(i)$  is the estimate of  $b_{jd}$  stored at node  $i$  according to the last message communicated from  $j$  to  $i$ . In brief, every node  $i$  tries to maintain the largest value of  $\min\{b[e_{ij}], b_{jd}^{(i)}\}$  among all of its neighbors  $j$  and it elects as successor the neighbor  $j'$  which maximizes this term. Whenever a neighbor  $j$  changes  $b_{jd}$  it notifies all its neighbors including  $i$ . Then  $i$  modifies its own estimate of  $b_{jd}$  by setting  $b_{jd}^{(i)} = b_{jd}$ . Then  $i$  recalculates  $b_{id} = \max_{j \in N(i)} \{\min\{b[e_{ij}], b_{jd}^{(i)}\}\}$  and switches successor if necessary. If link bandwidth  $b[e_{ij}]$  changes, a similar update should take place at  $i$ . Once node  $i$  changes  $b_{id}$  (either because of a change in a neighbor's estimated bandwidth or change in an adjacent link bandwidth) it notifies all its neighbors.

We model nodes as state machines. Next we present formally the states, transitions and procedures run at any node  $i$  for calculation of the widest path to any destination  $d$ .

#### Widest path Bellman-Ford at node $i \in V$ :

*State variables:*

- $b_{id}$ ; initialized 0 if  $i \neq d$  and otherwise  $\infty$ .
- $\pi_{id} \in N(i) \cup \text{null}$ ; initialized to *null*.
- $b[e_{ij}]$  for all  $j \in N(i)$ ; initialized to full capacity of link  $e_{ij}$ .
- $b_{jd}^{(i)}$  for all  $j \in N(i)$ ; initialized 0 if  $j \neq d$  and otherwise  $\infty$ .

*Transitions:*

- if  $i$  receives a message regarding change in  $b_{jd}$  from neighbor  $j$ :
  - $i$  updates its own estimate of node  $j$  bandwidth: set  $b_{jd}^{(i)} = b_{jd}$
- if  $b_{id} \neq \max_{j \in N(i)} \{\min\{b[e_{ij}], b_{jd}^{(i)}\}\}$ ,
  - $i$  recalculates its bandwidth estimate: set  $b_{id} = \max_{j \in N(i)} \{\min\{b[e_{ij}], b_{jd}^{(i)}\}\}$
  - $i$  updates its successor: set  $\pi_{id} = \operatorname{argmax}_{j \in N(i)} \{\min\{b[e_{ij}], b_{jd}^{(i)}\}\}$
  - if  $b_{id}$  changed,  $i$  notifies all neighbors about new  $b_{id}$

In what follows we explain an important performance failure of the presented algorithm. It is well known in the context of shortest path routing that asynchronous Bellman-Ford may create transient routing loops in case of link failures which slows down its convergence [3]. Besides, if some node is completely disconnected from the destination, convergence takes for ever (this phenomenon is known as the count to infinity problem) [3].

In our case, link states change dynamically because of scheduled set-up and release of connections according to step 1c of DAR algorithm. Along with the changes in future

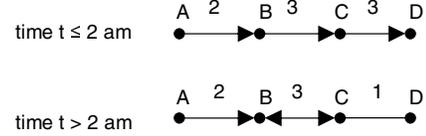


Figure 3. Illustration of permanent loops with widest path routing in a linear 4-node network: the widest path successors toward destination  $D$  are demonstrated with arrows and the numbers above links show available link bandwidth in Gbit/s at the given time.

available link bandwidths, the estimated successors and path bandwidths for future time slots must be updated to remain consistent.

*Lemma 4.4:* Distance vector routing based on the distributed asynchronous widest path Bellman-Ford presented above suffers from permanent routing loops in dynamic networks.

*Proof:* We prove this via an example showing the formation of a permanent routing loop following a change in the network state. In Figure 3 we show a linear network consisting of 4 nodes and 3 links. Assume a 2 Gbit/s connection from  $C$  to  $D$  is scheduled in advance starting from 2:00 am. The figure reflects this event with a change in link bandwidth  $b[e_{CD}]$  at 2:00 am. Since node  $C$  knows about this event in advance, it performs a successor transition from  $\pi_{CD} = D$  to  $\pi_{CD} = B$ . Then the estimated bandwidth at  $C$  remains  $b_{CD} = 3$  Gbit/s.  $B$  keeps  $C$  as its successor  $\pi_{BD} = C$  with  $b_{BD} = 3$  Gbit/s instead of 1 Gbit/s. Assuming no further change in link states, the loop  $\pi_{BD} = C$  and  $\pi_{CD} = B$  runs for ever. ■

We proved in sub-section IV-A that given our node data structure it is impossible to guarantee shortest-widest-earliest path optimization because construction of shortest-widest path by successors is not guaranteed. Here, we show that selecting at each node the shortest length among all widest path successors does not help to prevent formation of loops either.

We show this by an example based on the same figure 3. We assume every node selects the successor with smaller estimate of path length in case of a tie regarding path bandwidth. Then, after 2:00 am we have at  $C$ ,  $\pi_{CD} = B$  and again at  $B$ ,  $\pi_{BD} = C$  since  $C$  falsely offers  $B$  a wider path than  $A$  does. The estimated path lengths at  $B$  and  $C$  keep increasing in a loop without a bound because  $C$  sets  $l_{CD} = l_{BD} + 1$  ( $l_{CD}$  denotes estimated length of a path from  $C$  to  $D$ ) and vice versa for  $B$ . Soon we will have  $l_{BD} > l_{AD}$ . However this loop never breaks because invariably  $C$  offers a wider path than  $A$ , i.e.  $b_{BD} > b_{AD}$ .

The two previous examples show whenever routing optimization criterion is path width, formation of permanent loops is inevitable using the straight-forward extension of the shortest path Bellman-Ford. This explains why distance vector routing with widest path QoS is not explored in the literature, while shortest path QoS or link-state strategies

are very well studied. Loops are less likely with link state strategies since every node maintains a copy of the network topology.

On the other hand, re-initializing estimated state variables at all nodes after every change in current or future state of a single link is not a scalable solution because of excessive messaging overhead.

The literature offers practical methods for preventing formation of loops in distributed algorithms without having to re-initialize the whole network [13, 14]. However, most of the offered solutions are particularly based on shortest path (or minimum delay) routing optimization and either do not apply to or need a lot of modification to fit our scenario.

2) *Loop prevention*: We exploit a recently proposed algorithm called Distributed Path Computation with Intermediate Variable (DIV) to prevent formation of loops [4]. The DIV has the advantage that it decouples routing optimization from loop prevention process and this makes DIV applicable to various routing algorithms or successor selection criteria. The authors in [4] present it as a generic framework that can be adjusted to any distributed distance vector routing algorithm not limited to shortest path routing.

The DIV prevents loop formation using the concept *feasible successor set* defined per every destination at all nodes. The feasible successor set of  $i$  per every destination is a subset of  $N(i)$ . Successor to each destination is selected from the feasible successor set based on the routing optimization criteria.

In order to use DIV in our routing computations we must modify the data structure at nodes presented in sub-section III-C. Other than the path bandwidth and successor which are essential information for route calculation, every node must store intermediate variables called *values* which are solely added to determine the feasible successor set at every node for loop prevention purpose. Using the intermediate variables every node can track its own *value* and that of its neighbors.

Each value has the format  $val(i; j|k)$  which represents the value of node  $i$  known by node  $j$  based on the last update received from  $i$  and stored at node  $k$  (authors in [4] use the notation  $V(i; j|k)$ ). Hence, in addition to the data structure described in sub-section III-C, every node  $i$  stores for each destination:

- 1) The *value* of  $i$  as known to itself and stored at  $i$ ,  $val(i; i|i)$
- 2) The *value* of neighbor  $j$  as estimated by node  $i$  based on the last update from  $j$  and stored at  $i$ ,  $val(j; i|i)$  for  $j \in N(i)$
- 3) The *value* of  $i$  as estimated by neighbor  $j$  and then transferred to and stored at  $i$ ,  $val(i; j|i)$  for  $j \in N(i)$

The first and third variables are not equal in general for a given neighbor  $j$  but in steady state, DIV ensures that  $val(i; i|i) = val(i; j|i) = val(i; j|j)$  for every  $j \in N(i)$ .

Throughout the paper, if we mention *value* of node  $i$  without specifying stored or known by whom, we refer to  $val(i; i|i)$ . **Adapting DIV to our case.**  $val(i; j|k)$  is a generic variable which the DIV framework does not define it specifically. For our particular purpose, we define it as a two dimensional vector  $val(i; j|k) = \langle val_1(i; j|k), val_2(i; j|k) \rangle$ . For any given node  $i$ , the first component  $val_1(i; j|k)$  inversely relates to the estimated path bandwidth from  $i$  to  $d$ ,  $b_{id}$  and the second component  $val_2(i; j|k)$  relates to the estimated path length from  $i$  to  $d$ ,  $l_{id}$ . We will prove that  $val_1(i; i|i)$  converges to  $-b_{id}^*$  and  $val_2(i; i|i)$  converges to  $l_{id}^*$  in steady state. The intuition behind this choice of *values* is that the first component accounts for widest routing optimization. Thus, we give it the higher priority. The second component is required to satisfy the DIV constraints. Its role is to break the uniformity between neighboring node *values* with the same path bandwidth estimate; according to an invariance that we present later, every node must have a strictly larger *value* than its successor. With path bandwidth alone, it is not always possible to satisfy this invariance. In that case, some nodes could have no successor.

We set the following relation between the path bandwidth estimate  $b_{id}$  at any given node  $i$  and the *value* of its successor as known by  $i$ :  $b_{id} = \min\{b[e_{ij}], -val_1(j; i|i)\}$  where  $j = \pi_{id}$  and the following relation between the estimated path length  $l_{id}$  and *value* of  $i$ :  $l_{id} = 1 + val_2(j; i|i)$  where  $j = \pi_{id}$ .

Although the *value* of every node  $i$ , has to eventually be consistent with  $b_{id}$  and  $l_{id}$ , the *values* are restricted to satisfy certain invariant conditions. The invariances are responsible for preventing formation of loops. Our invariant conditions are very similar to those presented in [4] with the difference that we replace the standard comparators with the lexicographic comparators  $\prec_L$  and  $\succ_L$  defined below. Thus,  $val(i_1; j_1|k_1) \preceq_L val(i_2; j_2|k_2)$  implies:

$$\left\{ \begin{array}{l} 1. \ val_1(i_1; j_1|k_1) < val_1(i_2; j_2|k_2) \\ \text{or} \\ 2. \ val_1(i_1; j_1|k_1) = val_1(i_2; j_2|k_2), \\ \text{and } val_2(i_1; j_1|k_1) \leq val_2(i_2; j_2|k_2) \end{array} \right.$$

This does not change the results presented in [4].

- 1)  $val(i; i|i) \preceq_L val(i; j|i)$  where  $j \in N(i)$ .
- 2)  $j$  is in the feasible successor set of  $i$  if and only if  $val(i; i|i) \succ_L val(j; i|i)$ .

The first condition sets a bound on the choice of *value*. Every node has to keep its *value* below or equal to the estimate of its *value* communicated by its neighbors. This implies that if a node wants to increase its *value*, it should first notify its neighbors. The second condition defines the feasible successor set which restricts selection of successors only to neighbors that offer a better (lexicographically lower) value. This condition is set to prevent creation of routing loops.

The first invariance requires use of a special technique to update *values*. Communication between nodes is through three types of DIV messages: Update::Inc, Update::Dec and ACK. Update::Inc is a message that a node sends to its neighbors *before* it increases its *value*. Update::Dec is a message that a node sends to its neighbors *after* it decreases its *value*. ACK is sent in response to Update::Inc (only to the sender) after the appropriate actions are performed at the receiver of Update::Inc. For more details on the structure of these messages we refer the reader to [4].

When a given node  $i$  wants to increase its *value* it will first notify its neighbors before the actual increase. In turn, the neighbors that precede  $i$  will notify their own neighbors, etc. The recursive updates will finally extend to all ancestors of  $i$ . Every node that receives an Update::Inc and does not have to change its own *value* responds with an ACK immediately. Node  $i$  will eventually increase its *value* once it receives ACK from its neighbors. When a node needs to decrease its *value* it performs the decrease and then issues an Update::Dec to its neighbors (pretty much like the standard Bellman-Ford).

The DIV uses the following semantics for handling out of order messages:

- 1) A node ignores an update message that comes out of order.
- 2) A node ignores ACK messages after issuing an Update::Dec message.

Since Update and ACK messages have sequence numbers nodes can know the order. The two mentioned semantics rule out old messages in favor of the more recent ones leading to less messaging overhead and faster convergence. Regarding the second semantic, the receiver of the ACK ignores it which means it does not increase its *value* as specified in the **receipt of an ACK** procedure explained below, but if the node has received an Update::Inc from some neighbor earlier, it should still send an ACK to the neighbor which issued the Update::Inc.

3) *Presentation of SSM*: In the following, we describe our algorithm SSM for selection of successors. Next, we prove that the tentative paths constructed by SSM (by concatenation of successors) converge to the optimal (widest) paths.

As mentioned earlier, we present only the subroutines and states at node  $i$  per one destination  $d$  and for one particular time slot. The SSM must be repeated independently per every destination and for all future time slots at every node  $i$ . In our presentation  $\infty$  denotes a sufficiently large number.

On the high level, SSM is a combination of the asynchronous widest path Bellman-Ford and the DIV. Again, nodes are modeled as state machines. After listing the state variables and their initial settings at any given node  $i$ , we detail four events and the state transitions and actions they trigger.

To simplify the presentation, we assume no message re-ordering has happened but in that situation the two semantics

of DIV must be considered.

**state variables:**

- $b_{id}$ ; initialized 0 if  $i \neq d$  and otherwise  $\infty$ .
- $\pi_{id} \in N(i) \cup \text{null}$ ; initialized to *null*.
- $b[e_{ij}]$  for all  $j \in N(i)$ ; initialized to full capacity of link  $e_{ij}$ .
- $\langle val_1(i; i|i), val_2(i; i|i) \rangle$ ; initially set to  $\langle 0, \infty \rangle$  if  $i \neq d$ . Otherwise if  $i = d$  we set  $\langle -\infty, 0 \rangle$ .
- $\langle val_1(j; i|i), val_2(j; i|i) \rangle$  where  $j \in N(i)$ ; initially set to  $\langle 0, \infty \rangle$  if  $j \neq d$ . Otherwise if  $j = d$  we set  $\langle -\infty, 0 \rangle$ .
- $\langle val_1(i; j|i), val_2(i; j|i) \rangle$  where  $j \in N(i)$ ; initially set to  $\langle 0, \infty \rangle$  if  $i \neq d$ . Otherwise if  $i = d$  we set  $\langle -\infty, 0 \rangle$ .

First, we introduce the DecreaseV module. Whenever a node  $x$  wants to decrease its *value* it performs a certain set of tasks explained below. Assume  $y$  is the chosen successor of  $x$  and  $d$  the destination. Then,  $x$  decreases its *value*, the estimated *value* of  $x$  as known by any neighbor  $z$  and  $x$ 's estimated path bandwidth  $b_{xd}$  based on the parameters of successor  $y$ . Then  $x$  will send Update::Dec message to notify all its neighbors.

**Module DecreaseV**( $x, y, d$ ):

- 1) set  $-val_1(x; x|x)$  and  $-val_1(x; z|x)$  and  $b_{xd}$  equal to  $\{\min\{b[e_{xy}], -val_1(y; x|x)\}\}$  and set  $val_2(x; x|x)$  and  $val_2(x; z|x)$  equal to  $val_2(y; x|x) + 1$  for all  $z \in N(x)$
- 2) send Update::Dec to all neighbors  $z$  of  $x$  with the content  $val(x; x|x)$

When node  $i$  receives Update::Inc message with content  $\langle V_1, V_2 \rangle$  from a neighbor  $j''$ , this is a notification that  $j''$  wants to increase  $val(j''; j''|j'')$  according to  $\langle V_1, V_2 \rangle$ . If  $j''$  is the successor of  $i$ , this triggers an increase in *value* of  $i$ . To increase its *value*,  $i$  will send an Update::Inc message containing the *value* that  $i$  wants to have ( $(-\min\{b[e_{ij''}], -V_1\}, V_2 + 1)$ ) to all of its neighbors including  $j''$  and then waits for an ACK response from neighbors (node transition after reception of ACK will be explained separately). If  $j''$  is not the successor of  $i$ , then  $i$  will just respond with an ACK since it does not need to increase its *value*.

**receipt of an Update::Inc** with the desired *value*,  $\langle V_1, V_2 \rangle$  from neighbor  $j''$ :

- 1) if  $j''$  is successor of  $i$  then,
  - a) send an Update::Inc with  $\langle -\min\{b[e_{ij''}], -V_1\}, V_2 + 1 \rangle$  to all neighbors  $j \in N(i)$
- 2) else if  $j''$  is not successor of  $i$ ,
  - a) set  $val(j''; i|i)$  equal to  $\langle V_1, V_2 \rangle$
  - b) send to  $j''$  an ACK holding  $val(i; i|i)$  which is unchanged and  $val(j''; i|i)$  which equals  $\langle V_1, V_2 \rangle$

If  $i$  receives an Update::Dec message from neighbor  $j''$  with content  $\langle V_1, V_2 \rangle$  this indicates  $j''$  wants to decrease  $val(j''; j''|j'')$  according to  $\langle V_1, V_2 \rangle$ . If  $j''$  is  $i$ 's successor,  $i$  decreases its *value* by performing DecreaseV. If  $j''$  is not the successor of  $i$ , then  $i$  decreases its *value* only if  $j''$  becomes the new successor again by performing DecreaseV.

**receipt of an Update::Dec** with the desired *value*,

$\langle V_1, V_2 \rangle$  from neighbor  $j''$ :

- 1) set  $val(j''; i|i) = \langle V_1, V_2 \rangle$
- 2) if  $j''$  is successor of  $i$  then,
  - a) decrease *value* of  $i$  by calling DecreaseV( $i, j'', d$ )
- 3) else if  $j''$  is not successor of  $i$  then,
  - a) set  $J = \operatorname{argmax}_{j \in N(i)} \{\min\{b[e_{ij}], -val_1(j; i|i)\}\}$  where  $j$  is in the feasible successor set of  $i$ . If  $\pi_{id} \notin J$  then  $i$  switches successor:
    - i) set  $\pi_{id} = j'$  for any  $j' \in J$
    - ii) decrease *value* of  $i$  by calling DecreaseV( $i, j', d$ )

If  $i$  receives an ACK message from  $j''$ , it will first update its estimate of the *value* of  $j''$  and then its own *value* can increase according to the invariances 2. Note that ACK message must contain the *value* of its generator  $j''$  and because it is triggered in response to an Update::Inc issued earlier by  $i$ , it must contain the *value* that  $i$  has requested to increase to. If the increase in *value* of  $i$  is because of an Update::Inc message  $i$  has received earlier from a neighbor  $j^*$ ,  $i$  will modify  $val(j^*; i|i)$  as well. After  $i$  increases its *value* and  $b_{id}$ , it can search for a better successor and in case of a successor switch,  $i$  will decrease its *value* by performing DecreaseV. Finally,  $i$  must send an ACK if it has received an Update::Inc ( $i$  must have stored the content  $\langle V_1, V_2 \rangle$  of Update::Inc in its memory).

**receipt of an ACK** with content  $val(j''; j''|j'')$  and  $val(i; j''|j'')$  from neighbor  $j''$ :

- 1) set  $val(j''; i|i) = val(j''; j''|j'')$
- 2) set  $val(i; j''|i) = val(i; j''|j'')$
- 3) increase  $val(i; i|i)$  as much as possible as long as  $val(i; i|i) \leq_L val(i; j|i)$  holds for all  $j \in N(i)$
- 4) if  $i$  has received an Update::Inc with  $\langle V_1, V_2 \rangle$  from a neighbor  $j^*$  which is not acknowledged yet,
  - a) set  $val(j^*; i|i) = \langle V_1, V_2 \rangle$
  - b) set  $b_{id} = \min\{b[e_{ij^*}], -val_1(j^*; i|i)\}$
- 5)  $i$  can now search for a better successor: set  $J = \operatorname{argmax}_{j \in N(i)} \{\min\{b[e_{ij}], -val_1(j; i|i)\}\}$  where  $j$  is in the feasible successor set of  $i$
- 6) if  $\pi_{id} \notin J$ ,  $i$  switches successor:
  - a) set  $\pi_{id} = j'$  for any  $j' \in J$
  - b) decrease *value* of  $i$  by calling DecreaseV( $i, j', d$ )

- 7) if  $i$  has received an Update::Inc with  $\langle V_1, V_2 \rangle$  from a neighbor  $j^*$  which is not acknowledged yet, (i) set  $val(j^*; i|i) = \langle V_1, V_2 \rangle$  (ii) send an ACK to  $j^*$  holding  $val(i; i|i)$  and  $val(j^*; i|i)$

Inconsistency between  $b_{id}$  and  $\max_{j \in N(i)} \{\min\{b[e_{ij}], -val_1(j; i|i)\}\}$  may happen if bandwidth of a link adjacent to  $i$  changes or right after initialization. In either case,  $i$  can immediately update its successor if needed. Whether or not the successor changes  $b_{id}$  must be re-calculated. If  $b_{id}$  changes,  $i$  needs to update its *value* according to the DIV update rules mentioned earlier.

**inconsistency between  $b_{id}$  and**

$\max_{j \in N(i)} \{\min\{b[e_{ij}], -val_1(j; i|i)\}\}$ :

- 1) set  $\pi_{id} = j'$  for any  $j' \in J$  and  $J = \operatorname{argmax}_{j \in N(i)} \{\min\{b[e_{ij}], -val_1(j; i|i)\}\}$  where  $j$  is in the feasible successor set of  $i$
- 2) set  $b_{id} = \{\min\{b[e_{ij'}], -val_1(j'; i|i)\}\}$
- 3) if  $val(i; i|i) \prec_L \langle -b_{id}, val_2(j'; i|i) + 1 \rangle$ ,
  - a) send an Update::Inc with the desired *value* for  $i$ ,  $\langle -b_{id}, val_2(j'; i|i) + 1 \rangle$ , to all neighbors  $j \in N(i)$
- 4) else if  $val(i; i|i) \succ_L \langle -b_{id}, val_2(j'; i|i) + 1 \rangle$ ,
  - a) decrease *value* of  $i$  by calling DecreaseV( $i, j', d$ )

4) *Performance analysis*: In this section, first we analyze the worst case memory complexity at nodes. Then, we prove the time elapsed from issuing an Update::Inc message until receipt of corresponding ACK is finite. Based on this we prove that  $b_{id}$  and  $-val_1(i; d|d)$  converge to the bandwidth of the optimal path for every  $i$  and  $d$ . Using this and the loop-freedom property from [4], we prove that the paths constructed by SSM between every pair of nodes converge to the widest. Our analysis is based on the assumptions from section III-B. Hence, request inter-arrival time is long enough to allow for convergence of SSM path computations, there is no Byzantine behavior at nodes and links are reliable.

*Theorem 4.5*: The memory complexity at nodes is  $O(D_{max} \cdot |V| \cdot R)$  where  $D_{max}$  is the maximum node degree and  $R$  is the number of pending requests in the system.

*Proof*: Every node stores:

- 1) a path bandwidth estimate, successor and *value* of itself per destination and per future time slot (memory complexity: number of destinations multiplied by number of time slots)
- 2) bandwidth of all its adjacent links per future time slot (memory complexity: node degree multiplied by number of time slots)
- 3) estimated *value* of all of its neighbors and its neighbor's estimate of its own *value* per destination and per future time slot (memory complexity: node degree

multiplied by number of destinations multiplied by number of time slots)

The third item has the dominant memory complexity. Hence, we only consider it. Total number of slots at any node is in the worst case equal to  $2R + 1$  (if user specifies an arbitrary connection window  $[t_a, t_b]$ ). This happens if the node senses a successor or path bandwidth change per every set up or tear down of a connection throughout the network. For example this can happen in the line network of figure 3 at node  $A$  regarding destination  $D$ . Thus the worst case memory complexity is  $O(D_{max} \cdot |V| \cdot R)$  since the maximum number of destinations is  $|V|$ . ■

We borrow the following lemma from [4]. Its proof can be found therein.

*Lemma 4.6:* The successor graph is a *directed acyclic graph* (DAG) or a collection of DAGs at all times [4].

The proof is similar to the one in [4]. Because our initialization respects the two invariances of DIV they will always remain valid. The only difference is the replacement of regular inequalities with lexicographic ones.

We present the following two lemmas without proof due to space constraint.

*Lemma 4.7:* The worst case time from the moment a node issues an Update::Inc until it receives the corresponding ACK response is finite.

In the case of Update::Dec there are no ACK messages and the decrease happens immediately at the initiating node.

Next, we prove using the following lemma and corollary that a network whose nodes are initialized according to SSM, will eventually reach a steady state even if a finite number of links change bandwidth. By steady state, we mean all node variables remain fixed.

*Lemma 4.8:* Assuming network is in steady state, the total number of update messages after a bandwidth change on any link is finite.

*Corollary 4.9:* Assuming network is in steady state, the total number of messages triggered by any finite number of link changes is finite.

We infer from corollary 4.9 that assuming the network state is initialized according to SSM and bandwidth on a finite number of links changes afterwards, the network will eventually stabilize. To understand this, first assume that there will be no link bandwidth change in the network after initialization. In this case, all nodes will keep decreasing (improving) their *value* because except for  $d$ , all nodes are initialized with the largest (worst) possible *value* and there is no link bandwidth decrease to trigger an increase in node *values*. The process of decreasing *value* is no different than the standard Bellman-Ford update procedure and its convergence in an unchanging network is provable according to [3].

Now, assume some link bandwidths change after initialization. In this case, we have a superimposition of update traffic due to initial conditions and update traffic due to link

changes. Again, using the same reasoning used for corollary 4.9, the total number of messages will be finite.

Next, we prove the paths resulting from SSM are optimal:

*Theorem 4.10:* The path constructed by consecutive successors from every node  $i$  to any given destination  $d$  converges to the widest among all paths connecting  $i$  to  $d$ .

*Proof:* We prove by contradiction.

According to corollary 4.9 the network will eventually reach steady state. We assume the network has reached steady state. According to lemma 4.6, the path constructed from every node  $i$  by consecutive successors is loop-free: so either it is a simple path connecting  $i$  to  $d$  or it is a simple path that does not connect  $i$  to  $d$  and terminates at some node  $j \neq d$ . We denote such path  $P_{ij}$  in either case where in the first case  $j = d$ .

The proof consists of two parts:

**Part 1.** First we prove  $b_{id}$  and  $-val_1(i; i|i)$  for every node  $i$  equal the bandwidth of the path  $P_{ij}$ , i.e.  $\min_{e_{xy} \in P_{ij}} \{b[e_{xy}]\}$ .

Again the proof is by contradiction. Assume  $-val_1(i; i|i) \neq \min_{e_{xy} \in P_{ij}} \{b[e_{xy}]\}$  at steady state. Starting at node  $j$ , moving on predecessors one by one on  $P_{ij}$ , we call  $k$  the first node on the path with inconsistent  $-val_1(k; k|k)$  and path bandwidth. Assume  $\pi_{kd} = h$  and according to our assumption  $-val_1(h; h|h) = \min_{e_{xy} \in P_{hj}} \{b[e_{xy}]\}$ . At steady state, we have  $val(h; k|k) = val(h; h|h)$  because after every decrease in *value* of  $h$ ,  $h$  should have updated  $k$  and before every increase  $val(h; k|k)$  is set to the new *value* even before  $val(h; h|h)$  was updated.

Therefore, we have  $\min\{b[e_{kh}], -val_1(h; k|k)\} = \min_{e_{xy} \in P_{kj}} \{b[e_{xy}]\}$ . If we assume  $b_{kd}$  is not equal to  $\min\{b[e_{kh}], -val_1(h; k|k)\}$ , according to the **inconsistency** procedure,  $k$  has to update  $b_{kd}$  and this contradicts the node steady state assumption. So, we conclude that  $b_{kd}$  equals bandwidth of path  $P_{kj}$ .

But at steady state we also know that  $-val_1(k; k|k) = b_{kd}$  because otherwise  $k$  has to update its *value* by issuing update messages. So, we conclude that both  $-val_1(k; k|k)$  and  $b_{kd}$  equal bandwidth of path  $P_{kj}$ . Therefore, by recursive reasoning we conclude the same is true for  $i$ .

**Part 2.** Next, we prove by contradiction that if all nodes are at steady state, path  $P_{ij}$  must be an optimal path connecting  $i$  to  $d$ . At all times, we have for  $j' = \pi_{id}$ ,  $b_{id} = \min\{b[e_{ij'}], -val_1(j'; i|i)\}$  which equals the bandwidth of path  $P_{ij}$  formed by consecutive successors at steady state. If  $P_{ij}$  is not the widest possible path from  $i$  to  $d$ , because of the inconsistency between  $\max_{j \in N(i)} \min\{b[e_{ij}], -val_1(j; i|i)\}$  and  $b_{id}$ ,  $i$  has to update its successor according to the **inconsistency** procedure. This contradicts the steady state assumption.

Also we note that according to the first part of the proof, if  $P_{ij}$  does not connect  $i$  to  $d$ , then  $b_{id} = 0$ . Therefore, as long as there exists some path with positive bandwidth from  $i$  to  $d$ , we must have  $j = d$ .

## V. CONCLUSION AND FUTURE DIRECTIONS

In this paper, we investigated feasibility and requirements to implement end-to-end advance reservation with delay guarantees based on a distance-vector approach. Our analysis revealed the importance of proper choice of the path optimization criterion. We proved that earliest scheduling requires widest path routing and showed that both shortest-earliest and shortest-widest-earliest routing are infeasible given our node data structure. We highlighted the possible emergence of routing loops with widest path distance-vector routing (which may explain the absence of distance-vector QoS algorithms in the literature). We addressed this problem using the very recent DIV loop-prevention algorithm that lends itself to various routing optimization metric. Specifically, we defined the intermediate variables of DIV structure (called *values*) to be two-element tuples. The first element reflects path bandwidth and the second element, which has a lower priority than the first, reflects path length. The rationale behind our choice is that we first consider path bandwidth because of widest path routing and then path length to break uniformity of *values* (loop-prevention of DIV requires that the *value* of every node is larger than that of its successor).

We proved that our loop-free routing module SSM, based on DIV, converges to widest routing within finite time. Our proof is based on induction and uses the property of loop-freeness resulting from DIV. The DAR algorithm uses the route tables computed by SSM to find the earliest schedule for connections.

This work opens several new interesting directions for research. For instance, one problem is how to deal with situations where routing results of SSM are not stabilized by the time a new request arrives. Another problem is to deal with the fact that path switching does not occur instantaneously. Hence, in future work, we should consider the possibility of disconnection periods as part of our scheduling algorithm.

## VI. ACKNOWLEDGEMENTS

This work was supported in part by NSF under grant CCF-0729158.

## REFERENCES

- [1] *Large Hadron Collider (LHC)*, <http://lhc.web.cern.ch/lhc/>.
- [2] *Energy Sciences Network (ESnet)*, <http://www.es.net>.
- [3] D. Bertsekas and R. Gallager, *Data Networks*, Prentice-Hall, Inc., 1992.
- [4] S. Ray, R. Guerin, K. Kwong and R. Sofia, *Always Acyclic Distributed Path Computation*, IEEE/ACM Transactions on Networking (ToN), vol. 17, no. 6, Dec. 2009.
- [5] R. Cohen, N. Fazlollahi and D. Starobinski, *Path Switching and Grading Algorithms for Advance Channel Reservation*, IEEE/ACM Transactions on Networking (ToN), vol. 17, no. 5, pp. 1684-1695, Oct. 2009.
- [6] S. Sahni, N. Rao, S. Ranka, L. Yan, J. Eun-Sung and N. Kamath, *Bandwidth Scheduling and Path Computation Algorithms for Connection-Oriented Networks*, in proc. of International Conference on Networking ICN, April 2007, Sainte-Luce, Martinique, France.
- [7] R. A. Guerin and A. Orda, *Networks with Advance Reservation: The Routing Perspective*, in proc. of IEEE INFOCOM, March 2000, Tel-Aviv, Israel.
- [8] A. Schill, S. Kuhn, and F. Breiter, *Resource Reservation in Advance in Heterogeneous Networks with Partial ATM Infrastructures*, in proc. of INFOCOM, April 1997, Kobe, Japan.
- [9] M. Curado and E. Monterio, *A survey of QoS Routing Algorithms*, in proc. of the International Conference on Information Technology (ICIT'04), Dec. 2004, Istanbul, Turkey.
- [10] J. L. Sobrinho, *Algebra and Algorithms for QoS Path Computation and Hop-by-Hop Routing in the Internet*, IEEE/ACM Transactions on Networking, vol. 10, no. 4, pp. 541-550, Aug. 2002.
- [11] Z. Wang and J. Crowcroft, *Quality-of-Service Routing for Supporting Multimedia Applications*, IEEE Journal on Selected Areas in Communications, vol. 14, no. 7, pp. 1228-1234, Sep. 1996.
- [12] Z. Wang and J. Crowcroft, *Distance-vector QoS-based Routing with Three Metrics*, in proc. of Broadband Communications, High Performance Networking, and Performance of Communication Networks, May 2000, pp. 847-858, Paris, France.
- [13] J. J. Garcia-Luna-Aceves, *Loop-free routing using diffusing computations*, IEEE/ACM Transactions on Networking, vol. 1, no. 1, pp. 130-141, Feb. 1993.
- [14] S. Vutukury and J. J. Garcia-Luna-Aceves, *A simple approximation to minimum-delay routing*, ACM SIGCOMM Computer Communication Review, vol. 29, no. 4, pp. 227-238, Oct. 1999.
- [15] R. Albrightson, J. J. Garcia-Luna-Aceves and J. Boyle, *EIGRP - a fast routing protocol based on distance vectors*, ACM SIGCOMM Computer Communication Review, in proc. of Network/Interop, 1994.
- [16] J. M. Scott and I. G. Jones, *The ATM Forums private network/network interface*, BT Technology Journal, vol. 16, no. 2, pp. 37-46, April 1998.
- [17] S. Lin and D. J. Costello, *Error Control Coding: Fundamentals and Applications*, Englewood Cliffs, NJ: Prentice-Hall, 1983.
- [18] J. M. McQuillan and D. C. Walden, *The ARPANET Design Decisions*, Computer Networks, vol. 1, no. 5, Aug. 1977.