Theory Comput Syst (2017) 60:194–221
DOI 10.1007/s00224-015-9665-x

Distance Vector-based Advance Reservation with Delay Performance Guarantees

Niloofar Fazlollahi[1] · David Starobinski[1]

Published online: 18 November 2015

Abstract We explore and demonstrate the feasibility of implementing distributed solutions for advance reservation of network resources. 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 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 and analyze a novel approach for loop-free distributed widest path routing, leveraging the recently proposed DIV framework. Our routing results directly extend to on-demand and inter-domain QoS routing problems.

Keywords Grid and cloud computing · Scheduling · Routing · Quality of service

1 Introduction

1.1 Background

Modern grid and cloud computing applications require unprecedented network capabilities to support transfer of extremely large amounts of delay-sensitive and

✉ David Starobinski
staro@bu.edu

Niloofar Fazlollahi
niloo.fazl@gmail.com

[1] Department of Electrical and Computer Engineering, Boston University, Boston, MA 02215, USA

throughput-sensitive data among various data centers, national labs, universities, and other research centers. As a simple illustration, experiments run on the Large Hadron Collider (LHC) at CERN in Switzerland generate huge datasets, reaching the order of dozens of petabytes [25]. This information is then transferred from CERN to various sites around the world for the purpose of storage, processing, and analysis.

To address the above challenges, research and commercial providers have initiated the deployment of novel networking architectures, which principles represent a major shift from those underlying traditional TCP/IP networks. One of the most important and distinctive features of these new architectures is to support *advanced reservation*. As such, distributed hosts are provided with the ability to reserve in advance dedicated channels (circuits) to connect their resources. The goal of this design is two-fold: (i) provide deterministic quality of service *guarantees* to applications with strict bandwidth, delay, and/or jitter requirements, such as remote instrumentation and collaborative visualization applications used in grid computing [16]; (ii) provide *traffic isolation* to large bulk data transfer applications, such as GridFTP [3]. For both types of applications, current TCP/IP network architectures have been found to be inadequate, due to unacceptable throughput degradation and delay fluctuations caused by interfering traffic [12].

The advance reservation paradigm has been successfully proven and tested by a number of experimental projects, such as OSCARS [20, 28] and UltraScienceNet [30], and is now part of the operation of production networks, such as ESnet [13] and Internet2 [21]. Moreover, ESnet has recently established a second core network, called Science Data Network (SDN), that uses advance reservation services to set-up circuits and allocate dedicated bandwidth to flows. The main purpose of SDN is to provide support to the relatively small number (hundreds to thousands) of extremely large volume data flows (Gigabytes to Terabytes) that dominate ESnet traffic [22].

1.2 Problem

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 [37].

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 (based on the current network state). 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 3).

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 for 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 [6] 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`) [31].
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 follows. We first review related work in Section 2. Next, in Section 3, we explain our notation and assumptions and define the data structure maintained at nodes. Section 4 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 5

2 Related Work

Our work relates to several areas, namely joint routing and scheduling, QoS routing, and loop prevention. We briefly review each of them next.

There exists a rich literature on advance reservation services [4, 8, 9, 18, 19, 23, 29, 32, 33, 38, 41]. Most algorithmic work focuses on centralized solutions. A

notable exception is [15], but this work provides no theoretical performance guarantees. Additionally, a recent work [42] proposes a distributed advance reservation mechanism based on link-state routing. The problem with link-state routing schemes is that they require each node to have some global topology knowledge, which means that they share several of the same problems as centralized solutions. An earlier and abbreviated version of our paper appears in [14].

Some references focus on the signaling aspects of a distributed solution. For example, [34] discusses possible modifications to the RSVP protocol. More recently, several backbone networks (Internet2 and ESnet in the US, GEANT2 in Europe, and Canarie in Canada) have been working on the specifications of a new protocol, called Inter-Domain Controller Protocol (IDCP), allowing centralized schedulers in different domains to communicate [1]. Our paper complements this effort by offering a distance vector-like routing protocol that reports available bandwidth information as a function of time without revealing internal domain topologies.

Most work on QoS routing employs link-state routing, especially when it comes to widest path routing (see [7] for an excellent survey). Curado and Monterio [11] surveys various multi-criteria QoS distributed routing algorithms that try to reduce the complexity of this problem. Wang and Crowsoft [40] studies multi-criteria QoS routing and presents several combinations of criteria for which the problem is proved to be NP-complete. Sobrinho [35] investigates the properties that QoS criteria must possess to allow for computation of optimal paths using a generalized version of the Dijkstra algorithm.

We show in this paper that in order to guarantee the earliest connection starting time, selection of the widest path is required. Costa et al. [10] and Wang and Crowsoft [40] study 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 in that the problem is more severe, namely no convergence at all.

Next, we review some existing methods to prevent routing loops. Garcia-Luna-Aceves [17] and Vutukury and Garcia-Luna-Aceves [39] introduce loop-free shortest path algorithms extended from the Bellman-Ford algorithm [6]. Specifically, Ref. [17] 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. Vutukury and Garcia-Luna-Aceves [39] proposes an alternative method to prevent routing loops. Specifically, it maintains a pair of invariant conditions called *Loop Free Invariant* (LFI) at each node that depend on the node's 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 prevention of routing loops for shortest path routing. Ray et al. [31] offers a framework called DIV for loop prevention that can be used in conjunction with other metrics, which is critical for our paper. DIV is roughly a hybrid of the DUAL and LFI algorithms. We explain DIV in detail in Section 4.2. Here, we outline some of its advantages, other than its generic nature, compared to the previous references: (i) it supports multi-path routing; (ii) its feasibility conditions are more relaxed compared to the DUAL algorithm and hence triggers synchronous updates less frequently; and (iii) it can handle multiple overlapping updates simultaneously.

3 Model

3.1 Notation

Consider a network modeled with a weighted graph G, either directed or undirected, 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. Any pair of nodes may request a connection at any time. 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, a common time frame must be maintained at each node of 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 π_{id}. If $j = \pi_{id}$ then node i is the

predecessor of j. An ancestor of a given node i 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.

3.2 Assumptions

The statements proposed in this paper are correct under the following assumptions which are made to simplify the analysis:

1. Communication links do not drop packets.
2. Links never fail.
3. There is no Byzantine behavior at nodes, i.e., nodes do not drop, modify, or mis-route packets in an attempt to disrupt or degrade the routing service.
4. We assume that requests do not arrive simultaneously.
5. Clocks at different nodes are coarsely synchronized (e.g., on the order of seconds). This is in order that nodes agree on the set-up time and release of connections. We do not really need this assumption for the successor calculations of the routing sub-problem.
6. Later in this paper, we show that, following any change in the network, all the variables maintained at nodes eventually converge. We assume that the convergence always happens earlier than the arrival of a new connection request to the system.

This does not imply that there is no way we can resolve or alleviate these issues but rather that they are commonplace to distributed network routing algorithms. Much work in literature has addressed them and solutions are extensible to our particular case [2, 26, 36].

3.3 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 4.2, we add additional variables used uniquely to prevent formation of loops.

To accommodate advance reservation, every node should maintain relevant information regarding network state for 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 (we introduce the node variables in the next paragraph). To simplify the analysis, we divide the continuous time axis into discrete slots delineated by changes in the values of the node variables. Therefore, node variables remain fixed during each time slot. We refer to $t_1^{(id)}, t_2^{(id)}, \ldots, t_n^{(id)}$ as 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 for each 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 1 Figure 1a shows a network consisting of four nodes and four undirected links. Link bandwidths change over time as depicted in Fig. 1b

Table 1 depicts the node data structures related to the network of Fig. 1a. 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} =12:00AM to t_2^{BD} = 2:00AM and $\pi_{BD}(t) = D$ and $b^*_{BD}(t) = 20$ Gbit/s for time t from t_2^{BD} = 2:00AM 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} = 12:00AM to $t_2^{BC} = \infty$.

4 DAR Algorithm

Our objective is to devise a distributed algorithm guaranteeing that each request is provided with *minimal* delay. By minimal delay, we mean that the time until

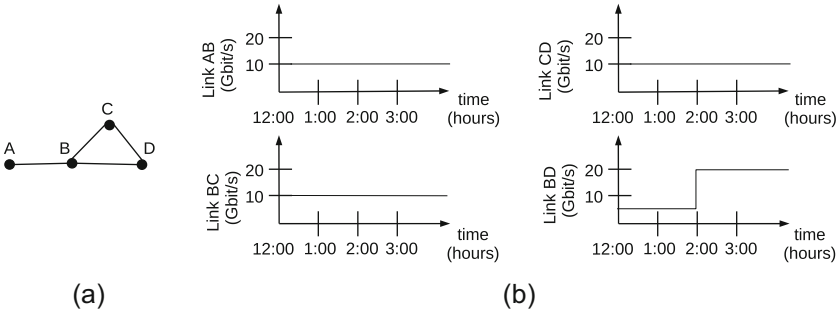

Fig. 1 The figure shows a network with changing link state: (**a**) an undirected graph of four nodes representing a network (**b**) available bandwidth on links e_{AB} , e_{BC}, e_{CD}, and e_{BD} over time. Since the graph is undirected every link can be presented with two formats. For example, e_{AB}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

Table 1 Node data structures for widest path successor selection

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

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

source	C			
destination	A	B	C	D
time (hours/AM)	12:00-∞	12:00-∞	12:00-∞	12:00-∞
next hop	B	B	-	D
path bandwidth	10	10	-	10

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

a request is scheduled is minimized, based on the network state *at the time of arrival of the request* (i.e., we consider an on-line algorithm). We also assume that previously scheduled requests cannot be reshuffled. We emphasize that our optimization metric does not guarantee the smallest possible (i.e., *minimum*) delay for each request. However, this problem is very difficult and even proven to be NP-hard for centralized, off-line variants [24]. Therefore, several advance reservation algorithms proposed in the literature for centralized settings are based on the earliest-scheduling criterion [8, 19]. We employ here the same metric for a distributed setting.

We divide the problem of devising a distributed 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.

4.1 Scheduling

We start this sub-section by mentioning the constraints imposed on routing because of the earliest scheduling optimization. For added clarity, we occasionally refer to the example network of Fig. 1 and Table 1 with concrete examples.

4.1.1 Widest Routing Requirement

Figure 2a and b depict the successor graphs based on widest path optimization and shortest path optimization respectively for destination D of the network illustrated in Fig. 1a.

Let us consider a particular example. Assume a request arrives at 12:00AM for a 10 Gbit/s connection lasting 3 hours from node A to D. According to Fig. 2b, the shortest path successors of A and B toward D are $\pi_{AD}(t) = B$ and $\pi_{BD}(t) = D$ at all times $t \geq$ 12:00AM leading to path (e_{AB}, e_{BD}). We observe that it is not possible for a connection requesting 10 Gbit/s to start at 12:00AM because bandwidth of the mentioned path is 5 Gbit/s from 12:00AM to 2:00AM. The earliest time to start the connection is 2:00AM for that path, though we could have started the connection at 12:00AM 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 1 *With the given node data structure and hop-by-hop routing paradigm, widest path routing is required to achieve earliest scheduling (i.e., one must set* $w[e_{ij}] = b[e_{ij}]$ *for each link* e_{ij}*).*

Proof The proof is by contradiction. Assume that at every node the successors and path weights for each time slot and destination are selected based on some given criteria other than widest path optimization. Also assume that we are able to achieve the earliest connection start time based on the mentioned structure for every request. Since the primary successor selection criterion is not the largest path bandwidth, there could exist some time slot $[t', t' + \Delta]$ during which the achieved path bandwidth $b_{s'd'}$

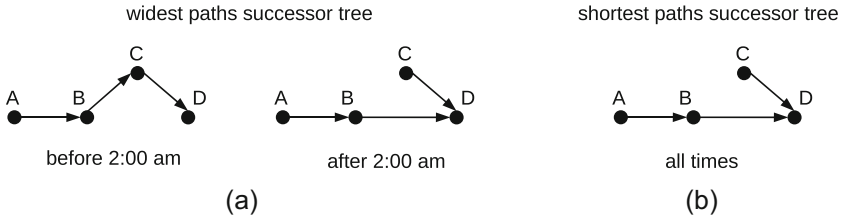

Fig. 2 Illustration of various successor selection criteria regarding the graph of Fig. 1: (**a**) successor tree for destination D based on widest path optimization (**b**) successor tree for destination D based on shortest path optimization. Note that successor trees for destinations A, B and C should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

from a given source s' to a given destination d' is less than the widest path bandwidth between the same pair during this interval, i.e., $b_{s'd'} < b^*_{s'd'}$. In that case, we can generate a request $(s', d', B', \Delta, t', \infty)$ where B' is larger than $b_{s'd'}$ and smaller than $b^*_{s'd'}$. Given such a request, the earliest connection starting time possible based on the mentioned structure is $t' + \Delta$ or later. However, the network resources are sufficient to start this connection at t'. This contradicts the assumption that we are able to achieve the earliest connection start time for every request using the mentioned structure and the theorem follows. □

4.1.2 Path Switching

We reconsider the network of Fig. 1a and the example request from A to D described in previous sub-section. According to the table of node A, based on the widest path optimization, we have for $t \geq$ 12:00AM, $\pi_{AD}(t) = B$ and $w^*_{AD}(t) = b^*_{AD}(t) =$ 10 Gbit/s. Hence it seems natural to assign the requested connection to the time interval 12:00AM to 3:00AM. However, according to the same table, for $t \in$ [12:00AM, 2:00AM] we have $\pi_{BD}(t) = C$, and for $t \geq$ 2:00AM we have $\pi_{BD}(t) = D$. Thus the successors in Table 1 do not provide a fixed path for connection during 12:00AM to 3:00AM. This restriction is concealed at node A. Therefore, A cannot decide to start the connection at 2:00AM 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 [8] in the context of centralized routing with advance reservation.

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

4.1.3 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 automate the process illustrated above for finding the earliest connection start time for 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^R within the time frame $[t_a, t_b]$ such that the bandwidth constraint is satisfied, i.e., $b^*_{sd}(t^R) \geq B$ for all $t \in [t^R, t^R + T]$.

Every node, such as s, must regularly update its time slot structure $t_1^{(sd)}, \ldots, t_n^{(sd)}$ since the first element of the list must always correspond to the present time t_{now}. The update process at node s consists of removing every time slot k whose start time $t_k^{(sd)} < t_{now}$, updating the indices of all remaining time slots so that the first slot is indexed 1 and $t_1^{(sd)} = t_{now}$.

Algorithm 1 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^R to t_a
   (b) If $b^*_{sd}(t) \geq B$ does not hold at all times $t \in [t^R, t^R + T]$ then,
      i. If $t^R \geq t_b$,
         – Reject the request
      ii. Otherwise,
         – Find a slot j with minimum value of j such that $t_j^{(sd)} > t^R$ and set t^R to $t_j^{(sd)}$
         – Go back to step 1b
   (c) If request is admissible, reserve connection
2. Go to step 1

After a request is found feasible, a distributed signalling protocol runs the reservation process. The source s sends a reservation request message to its successor(s). The message indicates the destination, the amount of reserved bandwidth and the reservation time interval. Note that, due to path switching, s may have different successors at different times during $[t^R, t^R + T]$, and thus may need several messages. Every node that receives a reservation request message, updates its routing table and sends reservation messages to its own successor(s) and so forth till the destination node d is reached.

Note that due to the propagation and processing delay of reservation messages flowing from sources to destinations, resource reservation conflicts between various incoming requests may arise. We do not get into the details of this problem here, as there already exist standard solutions in the literature for resolving reservation conflicts and re-routing of connections [36].

4.1.4 Performance Analysis

We next prove the most important property of DAR:

Theorem 2 *DAR provides the earliest connection start time for each 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).

We consider two cases: (i) If we only consider $P_{sd}(t)$, then DAR chooses the earliest time t^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 $t \in [t'^R, t'^R + T]$ where $t_a \leq t'^R < t^R$. 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 $t \in [t'^R, t'^R + T]$. But in this case DAR would have selected time t'^R_1 at step 1(b)ii. □

We may improve the performance of DAR by choosing the successor that returns 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 1 *Given the presented node structures, shortest-earliest path optimization is not feasible.*

Proof We prove this lemma with a negative example. Consider again the example network of Fig. 1a 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 12:00AM. If we select the successor returning the shortest among all widest paths, then $\pi_{BD}(t) = C$ for $t \in$ [12:00AM, 2:00AM]. We also have $\pi_{AD}(t) = B$ at all times $t \geq$ 12:00AM since this is the only option. Given this, we get $P^*_{AD} = (e_{AB}, e_{BC}, e_{CD})$ with bandwidth 5 Gbit/s from 12:00 to 2:00AM, while 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-earliest path is not guaranteed. □

Comparison with Performance of a Centralized Algorithm The above results show that DAR must employ widest-earliest scheduling. A centralized approach, on the other hand, could implement shortest-earliest scheduling (i.e., returns the shortest paths among all the earliest available) [8]. Simulation results, reported in [8, Section V.C.2], compare the performance of two similar schemes (called widest-shortest and shortest-widest). It is assumed that requests arrive according to a Poisson process with the same rate at each node, the destination for each request is selected uniformly at random among all nodes (excluding the source), the connection duration is exponentially distributed, and the bandwidth requested is distributed uniformly on a given range. Simulations run for different topologies suggest that the *maximum sustainable load* of widest-earliest scheduling is about 10-15% lower than that of shortest-earliest scheduling, where the maximum sustainable load is defined as the maximum rate of request arrivals before the average delay of requests becomes unbounded. This result illustrates the trade-off between deploying a fully distributed solution, such as DAR, and achieving higher performance with a centralized solution. In this case, the performance gap between the two solutions seems reasonable.

4.2 Pre-Computation of Routes

In the previous section we have assumed that nodes know the appropriate successor to every destination for 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 [2, 6, 27]. However, much of the focus of prior work has been on *shortest path* routing rather than any other metric for the reason explained next.

4.2.1 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, the 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 $\min\left\{b[e_{ij}], b_{jd}^{(i)}\right\}$ 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 $b_{id} = \max_{j\in N(i)}\left\{\min\left\{b[e_{ij}], b_{jd}^{(i)}\right\}\right\}$ 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 ∞.
- $\pi_{id} \in N(i) \cup null$; 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 ∞.

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 $\pi_{id} = \texttt{argmax}_{j \in N(i)}\{\min\{b[e_{ij}], b_{jd}^{(i)}\}\}$
  - 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 [6]. Besides, if some node is completely disconnected from the destination, convergence may take forever (this phenomenon is known as the count to infinity problem) [6].

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

Lemma 2 *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 Fig. 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 2:00AM. The figure reflects this event with a change in link bandwidth $b[e_{CD}]$ at 2:00AM. Since node C knows about this event in advance, it performs a successor transition from $\pi_{CD} = D$ to $\pi_{CD} = B$. Then

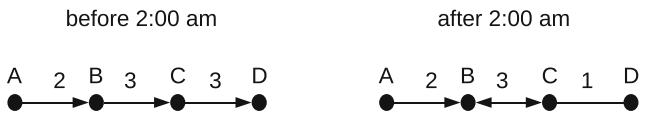

Fig. 3 Illustration of permanent loops with widest path routing in a linear 4-node network: the widest path successors toward destination D are demonstrated with arrows and the numbers above links show available link bandwidth in Gbit/s at the given time

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 forever. □

We proved in Section 4.1 that given our node data structure it is not possible to guarantee shortest-earliest path optimization. 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 Fig. 3. We assume every node selects the successor with smaller estimate of path length in case of a tie regarding path bandwidth. Then, after 2:00AM 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 verse 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.

The looping problem can, in principle, be solved by re-initializing the state variables at all nodes in the network after every change. However, this is not scalable because of excessive message overhead. Hence, the literature offers practical methods for preventing formation of loops in distributed algorithms without having to re-initialize the whole network [17, 39]. However, most of the offered solutions are based on shortest path (or minimum delay) routing optimization and either do not apply to or need a lot of modifications to fit our scenario.

4.2.2 Loop Prevention

We exploit a recently proposed algorithm called Distributed Path Computation with Intermediate Variable (`DIV`) to prevent formation of loops [31]. `DIV` has the advantage of decoupling routing optimization from loop prevention process. This makes `DIV` applicable to various routing algorithms or successor selection criteria. The authors in [31] 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 for each destination at all nodes. The feasible successor set of i for each 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 Section 3.3. 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 (believed) by node j and stored at node k (authors in [31] use the notation $V(i; j|k)$). Hence, in addition to the data structure described in Section 3.3, every node i stores for each destination d:

1. The *value* of i as known to itself, $val(i; i|i)$;
2. The *value* of neighbor j as known to itself, $val(j; i|i)$, for each $j \in N(i)$;
3. The *value* of itself as known to neighbor j, $val(i; j|i)$, for each $j \in N(i)$.

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 $val(i; i|i)$. Note that in order to simplify notation, and stay consistent with the notation used in [31], we do not explicitly include the dependency of the above variables in the destination d.

4.2.3 Adapting DIV to our Problem

Defining Values The quantity $val(i; j|k)$ is a generic variable that the DIV framework does not define 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. We will prove that $val_1(i; i|i)$ converges to $-b^*_{id}$, i.e., the optimal (widest) bandwidth between i and d, and $val_2(i; i|i)$ converges to the length of the optimal path 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 $j = \pi_{id}$.

Invariances 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 [31] with the difference that we replace the standard comparators with the lexicographic comparators defined next. Thus, $val(i_1; j_1|k_1) \succ_L val(i_2; j_2|k_2)$ implies:

$$\begin{cases} 1.\ val_1(i_1; j_1|k_1) > val_1(i_2; j_2|k_2) \\ \text{or} \\ 2.\ val_1(i_1; j_1|k_1) = val_1(i_2; j_2|k_2), \\ \ \text{and } val_2(i_1; j_1|k_1) > val_2(i_2; j_2|k_2) \end{cases}$$

Similarly, $val(i_1; j_1|k_1) \succeq_L val(i_2; j_2|k_2)$ implies:

$$\begin{cases} 1.\ val_1(i_1; j_1|k_1) > val_1(i_2; j_2|k_2) \\ \text{or} \\ 2.\ val_1(i_1; j_1|k_1) = val_1(i_2; j_2|k_2), \\ \ \text{and } val_2(i_1; j_1|k_1) \geq val_2(i_2; j_2|k_2) \end{cases}$$

The invariances are:

1. $val(i; j|i) \succeq_L val(i; i|i)$, where $j \in N(i)$.
2. Node j is in the feasible successor set of node 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.

Updating Values 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 [31].

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 per-

forms the decrease and then issues an Update::Dec to its neighbors (pretty much like the standard Bellman-Ford).

4.2.4 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 for one destination d and for one particular time slot. SSM must be repeated independently for each destination and for all future time slots at every node i. In our presentation ∞ 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. We illustrate each of these events using the example of Fig. 3.

State variables:

- b_{id}; initialized 0 if $i \neq d$ and otherwise ∞.
- $\pi_{id} \in N(i) \cup null$; 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 $\langle -\infty, 0\rangle$.

Events:

1) **Inconsistency between b_{id} and $\max_{j\in N(i)}\{\min\{b[e_{ij}], -val_1(j; i|i)\}\}$.** Such an inconsistency may happen if the bandwidth of a link adjacent to i changes, or right after initialization. In either case, node i immediately updates its successor, if needed. Whether or not the successor changes, b_{id} must be re-calculated. If b_{id} changes, node i needs to update its $value$ according to the DIV update rules mentioned earlier. Therefore, the steps to be taken are the following (the $DecreaseV$ module is described in the sequel):
   1. set $\pi_{id} = j'$ for any $j' \in J$ and $J = \texttt{argmax}_{j\in N(i)}\{\min\{b[e_{ij}], -val_1(j; i|i)\}\}$ 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 $val(i; i|i) \prec_L \langle -b_{id}, val_2(j'; i|i) + 1\rangle$,
      (i) 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 $val(i; i|i) \succ_L \langle -b_{id}, val_2(j'; i|i) + 1\rangle$,
      (i) decrease $value$ of i by calling DecreaseV(i, j', d)

Example 2 Consider Fig. 3. Before the change at 2:00AM, node C holds the following values for itself and its neighbors B and D: $val(C; C|C) = val(C; B|C) = val(C; D|C) = \langle -3 \text{ Gbit/s}, 1\rangle$; $val(B; C|C) = \langle -3 \text{ Gbit/s}, 2\rangle$; and $val(D; C|C) = \langle -\infty, 0\rangle$. Based on the second invariance, only node D belongs to the feasible successor set of node C. After 2:00AM, the available bandwidth of link e_{CD} goes down to 1 Gbits/s. In that case, D remains the successor of C (step 1) and the new estimate for the bandwidth from C to D is $b_{CD} = 1$ Gbit/s (step 2). Next, node C sends an Update::Inc message with its new desired value $\langle -1 \text{ Gbit/s}, 1\rangle$ to both nodes B and D (step 3).

2) **Receipt of an Update::Inc message.** When node i receives an 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 $(\langle -\min\{b[e_{ij''}], -V_1\}, V_2 + 1\rangle)$ 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 $value$. In summary, the steps to be taken are the following:

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

Example 3 We continue with the previous example. First, consider the action of node B upon the receipt of of an Update::Inc message from neighbor C with desired value $\langle -1 \text{ Gbit/s}, 1\rangle$. Node B first sets $val(C; B|B) = \langle -1 \text{ Gbit/s}, 1\rangle$ (step 1). Since node C is the successor of node B, node B sends an Update::Inc message with value $\langle -1 \text{ Gbit/s}, 2\rangle$ to both nodes A and C (step 2). Next, consider the action of node D upon the receipt of the Update::Inc message from node C. Node D first sets $val(C; D|D) = \langle -1 \text{ Gbit/s}, 1\rangle$ (step 1). Since C is not the successor of D, D sends an ACK to C with $val(C; D|D) = \langle -1 \text{ Gbit/s}, 1\rangle$ (step 3).

3) **Receipt of an ACK** that contains $val(i; j''|j'')$ from neighbor j''. When i receives an ACK message from j'', it first updates its estimate of the $value$ of j'' and then its own $value$ can increase according to invariance 1. 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. After i receives an ACK from each of its neighbors, it can search for a better successor. In the case of a successor switch, i will

decrease its $value$ by calling the function `DecreaseV`, defined below. 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). The steps in that case are the following:

1. set $val(i; j''|i) = val(i; j''|j'')$
2. increase $val(i; i|i)$ as much as possible such that $val(i; i|i) \preceq_L val(i; j|i)$ holds for all $j \in N(i)$
3. if i has not received an ACK message from each of its neighbors, exit.
4. if i has received an ACK message from each of its neighbors it can now search for a better successor:
   (a) set $J = \texttt{argmax}_{j\in N(i)}\{\min\{b[e_{ij}], -val_1(j; i|i)\}\}$ where j is in the feasible successor set of i
   (b) set $b_{id} = \min\{b[e_{ij'}], -val_1(j'; i|i)\}$, for any $j' \in J$
5. 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)
6. if i has received an Update::Inc with $\langle V_1, V_2\rangle$ from a neighbor j^* which has not been acknowledged yet, send an ACK to j^* holding $val(j^*; i|i)$

Example 4 We pursue the previous example. Consider node A. Upon receiving the Update::Inc message with value $\langle -1$ Gbit/s, $2\rangle$ from neighbor B, it sends an Update::Inc message with value $\langle -1$ Gbit/s, $3\rangle$ to neighbor B. Neighbor B then sends to A an ACK with value $val(A; B|B) = \langle -1$ Gbit/s, $3\rangle$. As a result of the receipt of this ACK, node A performs the following actions: set $val(A; B|A) = \langle -1$ Gbit/s, $3\rangle$ (step 1) and set $val(A; A|A) = \langle -1$ Gbit/s, $3\rangle$ (step 2). After receiving the ACK of node B, node A looks for a better successor. In this example, the successor remains node B (steps 4 and 5). Finally, node A sends an ACK to node B containing $val(B; A|A) = \langle -1$ Gbit/s, $2\rangle$ (step 6). Upon receiving this ACK message, node B performs similar steps as node A and sends an ACK message to node C, at which point the protocol converges.

4) **Receipt of an Update::Dec** message with the desired $value$, $\langle V_1, V_2\rangle$ from neighbor j''. The decrease of values is conceptually much simpler, as this action is similar to the standard Bellman-Ford algorithm. 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`. The following pseudo-code summarizes these steps:

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 $J = \texttt{argmax}_{j \in N(i)}\{\min\{b[e_{ij}], -val_1(j; i|i)\}\}$ 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 $\texttt{DecreaseV}(i, j', d)$

Next, we introduce the DecreaseV module. Assume y is the chosen successor of x and d the destination. Whenever a node x wants to decrease its value it performs the following task: 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 sends an Update::Dec message to notify all its neighbors.

Module $\texttt{DecreaseV}(x, y, d)$:

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 $val(x; x|x)$

Example 5 Figure 4 demonstrates the state transitions of SSM at each node for the network of Fig. 3. After the bandwidth change on link e_{CD}, SSM eventually converges to correct estimates of the path bandwidth at every node.

4.2.5 Performance Analysis

In this section, we first analyze the worst-case memory complexity of SSM. Then, we prove the time elapsed from issuing an Update::Inc message until receipt of the corresponding ACK is finite. Based on this, we prove that b_{id} and $-val_1(i; i|i)$ converge to the bandwidth of the optimal path for every i and d. Using this and the loop-freedom property from [31], 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 3.2. 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 3 *The memory complexity of SSM at each node is $O(D_{max}.|V|.R)$ 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 for each destination and future time slot (memory complexity: number of destinations multiplied by number of time slots)
2. Bandwidth of all its adjacent links for each future time slot (memory complexity: node degree multiplied by number of time slots)

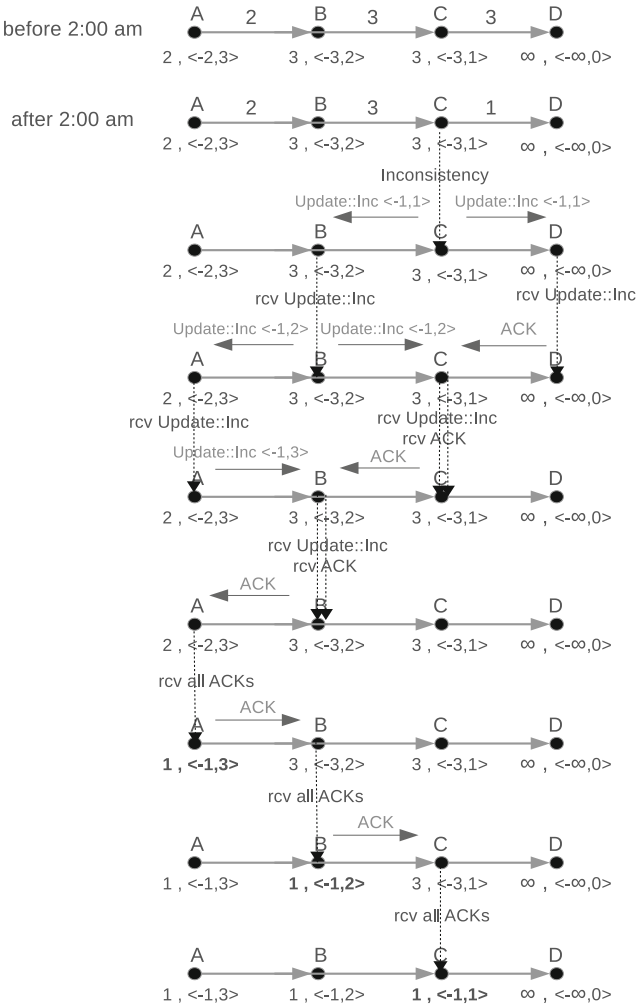

Fig. 4 State transitions and events for the network depicted in Fig. 3, after a bandwidth change on link e_{CD} at 2:00AM from 3 Gbits/s to 1 Gbits/s. Among all the state variables of SSM, only b_{ij} and $val(i;i|i)$ are shown, where i can represent any node A, B, C or D and j represents node D. The format for the state variables is as follows: b_{ij}, $\langle val_1(i;i|i), val_2(i;i|i)\rangle$. A dashed arrow depicts one of the four events of SSM

3. Estimated *value* of all of its neighbors and its neighbor's estimate of its own *value* for each destination and 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. The total number of slots at any node is in the worst case equal to $2R + 1$. This happens if the node changes its successor or the path bandwidth changes for each set up or tear down of a connection throughout the network (note that path switching can

only happen when another connection is set up or torn down). Thus, the worst case memory complexity is $O(D_{max}.|V|.R)$ since the maximum number of destinations is $|V|$. □

Note that the above memory complexity analysis only considers the variables required by SSM for distributed path computation. For resource reservation, each node on each connection path must also store the successor for each time slot throughout the connection duration and the corresponding connection bandwidth. Hence, the memory required by DAR for resource reservation at every node is at most the number of reserved (pending) requests in the system multiplied by the total number of time slots, i.e., $O(R^2)$.

Next we show that the first invariance always holds.

Lemma 3 *For all nodes i, $j \in V$, $val(i; i|i) \preceq_L val(i; j|i)$, where $j \in N(i)$.*

Proof The only situation in which node i is allowed to increase its $value$ is Step 2 of the procedure **Receipt of an ACK**. However, this step enforces $val(i; i|i) \preceq_L val(i; j|i)$ for all $j \in N(i)$. Similarly, the only situation in which node i decreases its value is Step 1 of the DecreaseV module. This step sets $val(i; j|i) = val(i; i|i)$ for all $j \in N(i)$, and therefore the lemma holds. □

We borrow the following lemmas from [31]. Its proof can be found therein.

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

The proof is similar to the one in [31]. Because our initialization respects the invariances of DIV, they will always remain valid. The only difference is the replacement of regular inequalities with lexicographic ones.

Lemma 5 *The worst case time from the moment a node issues an Update::Inc until it receives the corresponding ACK response is finite.*

Proof We focus on a given destination d. Assume $val(i; i|i)$ for some node i has to increase. Thus, i sends Update::Inc message to all of its neighbors and waits for ACKs from all of them. If a neighbor is not the predecessor of i, then according to step 3a of the procedure **receipt of Update::Inc**, it responds i with an ACK immediately. The lemma obviously holds in this case. If a neighbor is predecessor of i with respect to d it will recursively send Update::Inc messages to all of its neighbors before responding to i with an ACK. The recursion continues up to the leaf nodes of the successor graph, for which all neighbors respond with ACK messages immediately (the successor graph must have leaf nodes at every branch according to Lemma 4). Every node which receives ACK from all of its neighbors, sends an ACK back to the node from which it received an Update::Inc.

The above statements hold even if nodes receiving an Update::Inc message switch successor. If a node that has received an Update::Inc from its successor needs to

switch successor according to step 5 of the procedure **receipt of an ACK**, then it makes sure to issue an ACK toward its original successor afterwards according to step 6. Hence, the original successor will not wait forever for an ACK response. □

Next, we prove 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 6 *Assuming that the network is in steady state, the time of update messages after a bandwidth change on any link is finite.*

Proof First consider the situation for one particular destination d. If bandwidth of a link e_{ij} increases, this may only trigger Update::Dec messages. The update procedure in this case is similar to standard widest path or shortest path Bellman-Ford applied to an unchanging network. In this case node variables are proved to converge after a finite number of message emissions [6].

Assume bandwidth on a link e_{ij} decreases. If j is the successor of i with respect to d, i.e. $\pi_{id} = j$, this change causes a decrease in the estimated bandwidth at i, i.e. b_{id}, according to the SSM procedures. This triggers i to issue Update::Inc messages to all of its neighbors in order to increase $val_1(i; i|i)$ accordingly. According to Lemma 5 the time elapsed until every ACK response arrives at i is finite. Afterwards, if i needs to change successor to achieve a lower $value$, then the total number of update messages in the network is finite using a similar reasoning as the one regarding standard shortest path Bellman-Ford in [6]. Note that, in the worst-case, the number of messages generated may be exponential in the number of nodes in the network $|V|$ [5, p. 450].

Now consider, node k that is initially an ancestor of i. The state at such a node stabilizes, as long as the total number of update messages it receives is finite. Since the network size is finite, the number of update messages reaching every such node k through various paths will indeed be finite. The superposition of all messages for different destinations will lead to a similar result since variables corresponding to different destinations are updated independently. □

Corollary 1 *Assuming that the network is in steady state, the total number of messages triggered by any finite number of link changes is finite.*

We infer from Corollary 1 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 in a way similar to [6, p. 404–410].

Now, assume some link bandwidths change after initialization. In this case, we have a superposition of update traffic due to initial conditions and update traffic due to link changes. Again, using the same reasoning used for Corollary 1, the total number of messages will be finite and the network will reach steady-state in finite time. Since SSM follows the same steps as the standard distributed Bellman-Ford algorithm, except for preventing loops between a node and one or more of its ancestors due to inconsistent information about the node's *value*, we obtain the following:

Theorem 4 *The path constructed by consecutive successors from any node i to any given destination d converges in finite time to the widest among all paths connecting i to d.*

Proof We prove by contradiction. According to Corollary 1 the network will eventually reach steady state. We assume the network has reached steady state. According to Lemma 4, 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. $\min_{e_{xy} \in P_{ij}}\{b[e_{xy}]\}$. 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. Finally, 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 $j = d$. □

5 Summary

In this paper, we investigated the feasibility and requirements to implement end-to-end advance reservation with delay guarantees based on a distance-vector approach. This problem is practically relevant to the design of distributed network architectures supporting grid computing applications, and possibly also cloud computing applications in the future. Our analysis revealed the importance of proper choice of the path optimization criterion. We first proved that earliest scheduling requires widest path routing and that shortest-earliest routing is infeasible given our node data structure.

Next, we highlighted the possible emergence of routing loops with widest path distance-vector routing. We addressed this problem using the 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 proofs exploit the property of loop-freedom resulting from DIV. The DAR algorithm uses the route tables computed by SSM to find the earliest schedule for connections.

While the focus of our paper is on distributed advance reservation, our results have broader scope. Thus, SSM can be used for the design of on-demand distance-vector QoS algorithms. Such algorithms can serve as the basis for inter-domain routing protocols, since they avoid the need of sharing global topology information. Another broader contribution is in the formal description of SSM using states, transitions (events) and procedures, since [31] did not provide such.

The paper opens interesting avenues for future work, such as the problem of handling link failures. While SSM addresses the impact of link failures on routing tables, such failures may also force rescheduling of on-going and future connections. Thus, how to implement such rescheduling in a localized and efficient manner is an important topic left for future work. Besides, earlier work, for centralized advance network reservation, shows that multi-path routing (i.e., the ability of setting-up a connection across multiple paths) can lead to significant performance gains [9]. It would, therefore, be of interest to investigate ways of extending DAR to support

multi-path routing. Finally, DAR could be extended to handle requests with different priorities [19].

Acknowledgments This material is based upon work supported by the U.S. National Science Foundation under grant CNS-1117160.

References

1. DICE InterDomain Controller Protocol (IDCP). http://www.controlplane.net/
2. Albrightson, R., Garcia-Luna-Aceves, J.J., Boyle, J.: EIGRP - a fast routing protocol based on distance vectors. In: Proceedings of the Network/Interop (1994)
3. Allcock, W., Bresnahan, J., Kettimuthu, R., Link, M., Dumitrescu, C., Raicu, I., Foster, I.: The globus striped gridftp framework and server. In: Proceedings of the 2005 ACM/IEEE conference on Supercomputing, p. 54. IEEE Computer Society (2005)
4. Andrews, M., Fernandez, A., Goel, A., Zhang, L.: Source routing and scheduling in packet networks. J. ACM (JACM) **52**(4), 582–601 (2005)
5. Bertsekas, D.P., Tsitsiklis, J.N.: Parallel and distributed computation. Prentice Hall Inc., Old Tappan (1989)
6. Bertsekas, D., Gallager, R.: Data Networks. Prentice-Hall, Inc. (1992)
7. Chen, S., Nahrstedt, K.: An overview of quality-of-service routing for the next generation high-speed networks: Problems and solutions. IEEE Netw. **12**(6), 64–79 (1998)
8. Cohen, R., Fazlollahi, N., Starobinski, D.: Path switching and grading algorithms for advance channel reservation architectures. IEEE/ACM Trans. Netw. (TON) **17**(5), 1684–1695 (2009)
9. Cohen, R., Fazlollahi, N., Starobinski, D.: Throughput-competitive advance reservation with bounded path dispersion. IEEE/ACM Trans. Netw. (ToN) **19**(5), 1265–1275 (2011)
10. Costa, L.H., Fdida, S., Duarte, O.C.: Distance-vector QoS-based routing with three metrics. In: Proceedings of Broadband Communications, High Performance Networking, and Performance of Communication Networks - Networking, pp. 847–858, Paris (2000)
11. Curado, M., Monterio, E.: A survey of QoS routing algorithms. In: Proceedings of the International Conference on Information Technology (ICIT'04), Istanbul (2004)
12. Petravick, D., et al.: DOE Workshop Report: Advanced Networking for Distributed Petascale Science, Gaithersburg (2008)
13. ESnet. http://www.es.net/
14. Fazlollahi, N., Starobinski, D.: Distributed advance network reservation with delay guarantees. In: Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS'10), Atlanta (2010)
15. Ferrari, D., Gupta, A., Ventre, G.: Distributed advance reservation of real-time connections. IMultimedia Systems'97 **5**(3), 187–198 (1997)
16. Foster, I., Kesselman, C.: The Grid 2: Blueprint for a New Computing Infrastructure. Morgan Kaufmann (2003)
17. Garcia-Luna-Aceves, J.J.: Loop-free routing using diffusing computations. IEEE/ACM Trans. Netw. **1**(1), 130–141 (1993)
18. Greenberg, A.G., Srikant, R., Whitt, W.: Resource sharing for book-ahead and instantaneous-request calls. IEEE/ACM Trans. Netw. **7**(1), 10–22 (1999)
19. Guerin, R., Orda, A.: Networks with advance reservations: The routing perspective. In: Proceedings of IEEE INFOCOM'00 (2000). Tel-Aviv, Israel
20. Guok, C., Robertson, D., Thompson, M., Lee, J., Tierney, B., Johnston, W.: Intra and interdomain cicruit provisioning using the OSCARS resevation system. In: Proceedings IEEE Gridnets '06, San-Jose (2006)
21. internet2. http://networks.internet2.edu/
22. Johnston, W.: Esnet: Advanced networking for science. SciDAC Rev. **4**, 48 (2007)
23. Lee, H., Veeraraghavan, M., Li, H., Chong, E.K.P.: Lambda scheduling algorithm for file transfers on high-speed optical circuit. In: IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2004), Chicago (2004)
24. Lewin-Eytan, L., Naor, J., Orda, A.: Routing and admission control in networks with advance reservations. Approximation Algorithms for Combinatorial Optimization, pp. 215–228 (2002)

25. Large Hadron Collider (LHC) project. http://lhc.web.cern.ch/lhc/
26. Lin, S., Costello, D.J.: Error Control Coding: Fundamentals and Applications. Prentice-Hall, Englewood Cliffs (1983)
27. McQuillan, J.M., Walden, D.C.: The ARPANET design decisions. Comput. Netw. **1**(5) (1977)
28. On-demand secure circuits and advanced reservation systems. http://www.es.net/oscars/index.html
29. Rajah, K., Ranka, S., Xia, Y.: Advance reservation and scheduling for bulk transfers in research networks. IEEE Trans. Parallel and Distrib. Syst. **20**(11), 1682–1697 (2009)
30. Rao, N.S.V., Wing, W., Carter, S., Wu, Q.: UltraScience net: Network testbed for large-scale science applications. IEEE Commun. Mag. **43**(11), S12–S17 (2005)
31. Ray, S., Guerin, R., Kwong, K., Sofia, R.: Always acyclic distributed path computation. IEEE/ACM Trans. Netw. **18**(1), 307–319 (2010)
32. Sahni, S., Rao, N., Ranka, S., Yan, L., Eun-Sung, J., Kamath, N.: Bandwidth scheduling and path computation algorithms for connection-oriented networks. In: Proceedings of the International Conference on Networking ICN. Sainte-Luce, Martinique (2007)
33. Schelén, O., Pink, S.: Resource sharing in advance reservation agents. J. High Speed Netw. **7**(3–4), 213–228 (1998)
34. Schill, A., Kuhn, S., Breiter, F.: Resource reservation in advance in heterogeneous networks with partial ATM infrastructures. In: Proceedings of IEEE INFOCOM'97. Kobe, Japan (1997)
35. Sobrinho, J.L.: Algebra and algorithms for QoS path computation and hop-by-hop routing in the internet. IEEE/ACM Trans. Netw. **10**, 541–550 (2001)
36. The ATM Forum, Ahmed, M., Rus, J.H.: Private network-network interface specification version 1.0 (pnni 1.0) (1996)
37. Veeraraghavan, M., Starobinski, D.: A routing architecture for scheduled dynamic circuit services. In: Proceedings of the Re-Architecting the Internet Workshop, p. 6. ACM (2010)
38. Virtamo, J.T.: A model of reservation systems. IEEE Trans. Commun. **40**, 109–118 (1992)
39. Vutukury, S., Garcia-Luna-Aceves, J.J.: A simple approximation to minimum-delay routing. ACM SIGCOMM Comput. Commun. Rev. **29**(4), 227–238 (1999)
40. Wang, Z., Crowcroft, J.: Quality-of-service routing for supporting multimedia applications. IEEE J. Sel. Areas Commun. **14**(7), 1228–1234 (1996)
41. Wolf, L.C., Steinmetz, R.: Concepts for resource reservation in advance. Multimedia Tools Appl. **4**(3), 255–278 (1997)
42. Xie, C., Alazemi, H., Ghani, N.: Routing and scheduling in distributed advance reservation networks. In: GLOBECOM 2010, 2010 IEEE Global Telecommunications Conference, IEEE (2010)