Distributed Scheduling for Advance Bandwidth Reservation in High-Performance Networks

Qishi Wu and Yunyue Lin

University of Memphis, Memphis, TN 38152, USA
{qishiwu,ylin1}@memphis.edu

Abstract. High-performance networks are capable of provisioning dedicated channels through circuit/lambda-switching or MPLS/GMPLS techniques to support large-scale data transfer. These dedicated links are typically shared by multiple users through advance resource reservations, resulting in varying bandwidth availability in future time periods. Most previous efforts were focused on centralized bandwidth scheduling to improve the utilization of network resources and meet the transport requirements of application users. These centralized scheduling schemes imply the use of a central control plane, posing significant reliability and scalability challenges as the network size rapidly grows. We propose distributed algorithms for path computation and bandwidth scheduling in response to four basic bandwidth reservation requests: (i) fixed bandwidth in a fixed slot, (ii) highest bandwidth in a fixed slot, (iii) first slot with fixed bandwidth and duration, and (iv) all slots with fixed bandwidth and duration. These algorithms are developed through a rigorous extension of the classical breadth first search and Bellman-Ford algorithms to a completely distributed manner, and their performances are evaluated and analyzed through extensive simulations.

Keywords: distributed scheduling, bandwidth reservation, high-performance networks.

1 Introduction

1.1 Background

A number of large-scale applications in various science, engineering and business domains are generating colossal amounts of data, on the order of terabytes currently and petabytes or even exabytes in the near future, which must be transferred over a long geographical distance for remote operations. High-performance networks that are capable of provisioning dedicated channels have proved to be a promising solution to large data transfer and several network projects are currently underway to develop such capabilities, including UltraScience Net (USN) [17], Circuit-switched High-speed End-to-End Transport ArcHitecture (CHEETAH) [7], Dynamic Resource Allocation via GMPLS Optical Networks (DRAGON) [1], Japanese Gigabit Network II [2], Bandwidth on Demand (BoD) on Geant2 network [3], On-demand Secure Circuits and Advance Reservation

System (OSCARS) [4] of DOE Energy Sciences Network (ESnet), Hybrid Optical and Packet Infrastructure (HOPI) [5], and Bandwidth Brokers [21]. Such dedicated channels are a part of the capabilities envisioned for Global Environment for Network Innovations (GENI) project [6].

The deployments of high-performance networks are expected to increase significantly and proliferate into both public and dedicated network infrastructures across the globe in the coming years. An evidence of this trend in production networks is reflected by Internet2 offering on-demand circuits and Multiple Protocol Label Switching (MPLS) tunnels. MPLS improves the forwarding speed of IP routers by adopting a key concept from the world of virtual-circuit networks: a fixed-length label. MPLS is often referred to as layer-2.5, which adds a small MPLS header between the layer-2 header and the layer-3 header in a link-layer frame. Since modern optical networks have reached a very high transfer rate (at 40 Gbit/s and beyond), the true advantages of MPLS do not lie in the potential increase in switching speeds, but rather in the new traffic management capabilities that MPLS enables. OSCARS uses MPLS and Resource Reservation Protocol (RSVP) to create virtual circuits or Label Switched Paths (LSPs), while the management and operation of end-to-end virtual circuits within the network are done at the layer-3 network level. OSCARS supports advance reservation, but its underlying path computation limits connections over links returned by traceroute; hence, it does not explore all available bandwidths inside the network.

The dedicated links in high-performance networks are typically shared by multiple users through various advance reservation techniques, resulting in varying bandwidth availability in future time periods. Most previous scheduling efforts were focused on centralized advance bandwidth reservation in high-performance networks that employ a central control plane [15,16,9]. Such centralized schemes are suited for small-scale networks, but pose significant reliability and scalability challenges as the network size increases, which calls for distributed solutions for large-scale networks. To the best of our knowledge, there are very few studies on distributed advance bandwidth reservation. In this paper, we formulate four basic advance bandwidth scheduling problems and propose distributed algorithms for path computation and bandwidth scheduling. These algorithms are developed through a rigorous extension of the classical breadth first search (BFS) and Bellman-Ford algorithms to a completely distributed manner and their performances are evaluated and analyzed through extensive simulations.

1.2 Related Work

As high-speed dedicated networks are increasingly developed and deployed, many scheduling algorithms have been designed for advance bandwidth reservation. We provide below a brief survey of such efforts.

The four basic bandwidth scheduling problems discussed in this paper were first introduced in [18] and were later investigated in [19] with a detailed description on the solution to each of these problems in the centralized scheme. Guerin *et al.* studied these basic scheduling problems with several extensions to increase the flexibility of services [13]. The scheduling algorithm proposed by

Cohen *et al.* considers the flexibility of transfer start time and the capability of path switching between different paths during the transfer to improve network utilization [8]. Grimmell *et al.* formulated a dynamic quickest path problem, which deals with the transmission of a message from a source to a destination with the minimum end-to-end delay over a network with propagation delays and dynamic bandwidth constrains on the links [12]. Veeraraghavan *et al.* transferred files with varying bandwidths in different time slots in a simple case where the path is pre-specified [20]. Ganguly *et al.* generalized the problems of finding an optimal path in a graph with varying bandwidths to minimize the total transfer time, and also proposed approaches to find the minimum number of path switchings to transfer a file in a specified number of time slots [10]. Gorinsky *et al.* proposed a Virtual Finish Time First algorithm to schedule incoming files in a preemptive manner to minimize the total transfer end time over a given dedicated channel [11]. Most recently, Lin *et al.* proposed four instant scheduling problems under different constraints (fixed or variable) on both path and bandwidth [16]. Considering the path switching delay, the scheduling problem using either fixed or variable paths with variable bandwidths are proved to be NP-complete [15].

Most of the aforementioned efforts were focused on centralized bandwidth scheduling in dedicated networks managed by a central control plane, where a global repository storing all bandwidth reservations on all network links has to be maintained and updated. Such a centralized management scheme is not adequate for large-scale networks due to its reliability and scalability issues, which motivate us to develop distributed solutions.

The rest of the paper is organized as follows. We formulate four basic advance bandwidth scheduling problems in Section 2. The distributed algorithms for these scheduling problems are presented in Section 3. The scheduling performance is evaluated in Section 4.

2 Advance Bandwidth Scheduling Problems

We consider a generic control plane to support advance bandwidth reservation of dedicated channels in high-performance networks [18]. The distributed bandwidth scheduler runs on each switch in circuit/lambda-switching networks or on each router in MPLS-enabled networks, and computes routing paths in a distributed manner based on the available bandwidth of adjacent links. In MPLS-enabled networks, a Label Switched Path (LSP) is created by the signaling daemon at the start time of a bandwidth reservation. Each router along the computed path receives a path setup request via Resource Reservation Protocol (RSVP) and commits bandwidth to create the LSP. At the end time of a bandwidth reservation, the signaling daemon tears down the corresponding LSP.

We represent the topology of a dedicated network as a directed graph $G = (V, E)$ with n nodes and m links, where each link $l \in E$ maintains a list of residual bandwidths specified as segmented constant functions of time. An example of the available bandwidth over time for a link is shown in Fig. 1. We use a 3-tuple of time-bandwidth (TB) $(t_l[i], t_l[i+1], b_l[i])$ to represent the residual bandwidth

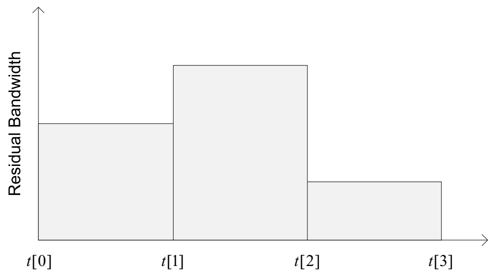

Fig. 1. A time-bandwidth example of a link

of link l at time slot $[t_l[i], t_l[i+1]]$, $i = 0, 1, 2, \dots, T_l - 1$, where T_l is the total number of time slots of link l. The current time point is denoted by $t_l[0]$, and the future time point is denoted by $t_l[i]$ $(i > 1)$. We set $t_l[T_l] = +\infty$, which indicates that there is no bandwidth reservation on link l after time point $t_l[T_l - 1]$ and therefore $b_l[T_l - 1]$ is the initial bandwidth of link l. Let T be the maximum number of time slots on a TB list of link $l \in E$. Each node $v \in V$ only knows its neighbor nodes, and maintains the TB lists for all outgoing links from itself to its neighbor nodes.

Based on different data transport constraints and application requirements, we formulate four basic advance bandwidth scheduling problems: Given a graph $G = (E, V)$ with a time-bandwidth list TB for each link $l \in E$, source v_s and destination v_d,

– Fixed-Bandwidth: compute a path from v_s to v_d with a fixed bandwidth β in a specified time slot $[t_s, t_e]$.
– Highest-Bandwidth: compute a path from v_s to v_d with the highest available bandwidth in a specified time slot $[t_s, t_e]$.
– First-Slot: compute the earliest start time of a path from v_s to v_d with a fixed bandwidth β for a specified duration t_d.
– All-Slots: compute all start time slots of all paths from v_s to v_d with a fixed bandwidth β for a specified duration t_d.

Note that the solution to the First-Slot scheduling problem is the earliest start time, while the solution to the All-Slots scheduling problem is a union of all feasible start times. If t is a feasible start time in the solution to All-Slots, the computed path has bandwidth β from time point t to $t + t_d$.

3 Distributed Scheduling Algorithms

We propose an optimal bandwidth scheduling algorithm in a distributed manner for each of these problem. The proposed algorithms are based on the BFS and Bellman-Ford algorithms and are different from the existing link-state and distance-vector routing protocols: a node makes a routing decision based on its local TB lists and connectivity information, and only broadcasts its own information to its neighbor nodes. Although link-state routing protocols are easy to implement, the periodical broadcasting of node connectivity and link TB lists incurs a significant amount of overhead. Furthermore, if the changes in node connectivity and link bandwidth are not promptly updated, the network may operate with inaccurate information.

We use two types of routing messages between nodes for distributed path exploration for advance bandwidth scheduling: (i) bandwidth reservation message, and (ii) acknowledgment (ACK) message. The bandwidth scheduler running on every node incorporates the scheduling algorithms and handles the routing messages during path exploration. Node and link states can be updated by simple periodic message exchanges between neighbor nodes. Every node broadcasts a HELLO message to its neighbor nodes. Upon the receival of a HELLO message, a node simply replies with an ACK message to let the neighbor node know that the link between them is active.

3.1 Fixed-Bandwidth

Given a fixed-bandwidth (FB) reservation request r_i^{FB}, the fixed-bandwidth scheduling problem is to compute a dedicated channel from source node v_s to destination node v_d with specified bandwidth β in time slot $[t_s, t_e]$. The source node v_s receives r_i^{FB} from an end user, and initiates path exploration by broadcasting r_i^{FB} to its neighbor nodes. When an intermediate node receives r_i^{FB} from one of its neighbor nodes, it checks the TB lists of its outgoing links and determines whether r_i^{FB} can be scheduled on these links. An example of bandwidth reservation process is shown in Fig. 2. After receiving r_i^{FB} from node v_1, node v_2 checks the TB lists of three outgoing links (v_2, v_3), (v_2, v_4), and (v_2, v_5). If r_i^{FB} is feasible only on links (v_2, v_3) and (v_2, v_5), node v_2 sends r_i^{FB} to nodes v_3 and v_5. Once r_i^{FB} reaches the destination node v_d, node v_d replies with a positive acknowledgment message, which is echoed all the way back to the source node.

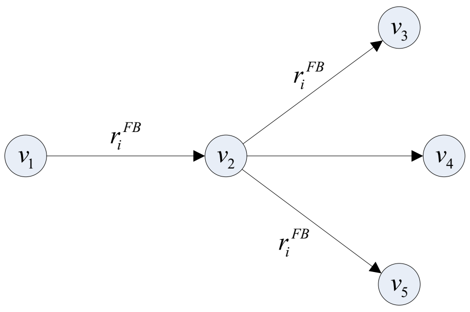

Fig. 2. An example of the bandwidth reservation process for the fixed-bandwidth problem

Algorithm 1. Scheduling algorithm for the fixed-bandwidth problem

```
1: Create a job queue $Q$ to store all reservation requests.
2: Wait for routing messages.
3: if a fixed-bandwidth reservation request $r_i^{FB}$ is received from its neighbor $u$ then
4:    Add $u$ to $V_i^{pre}$.
5:    if $r_i^{FB}$ is not in $Q$ then
6:       Add $r_i^{FB}$ to $Q$ and mark $r_i^{FB}$ as "pending".
7:       if the current node is the destination of $r_i^{FB}$ then
8:          Send a positive acknowledgment of $r_i^{FB}$ to $u$.
9:       else
10:         Compute the neighbor node set $S_i$ (excluding $u$) such that $r_i^{FB}$ can be
            scheduled on each link between the current node and any neighbor node in
            $S_i$. If $S_i \neq \emptyset$, broadcast $r_i^{FB}$ to all nodes in $S_i$; otherwise, send a negative
            acknowledgment of $r_i^{FB}$ to $u$ and mark $r_i^{FB}$ as "failed". Initialize $n_i = |S_i|$.
11:   else if $r_i^{FB}$ is marked as "failed" in $Q$ then
12:      Send a negative acknowledgment of $r_i^{FB}$ to $u$.
13:   Return to line 2.
14: if an acknowledgment of request $r_i^{FB}$ is received from its neighbor node $u$ and $u$
    is in $S_i$ then
15:   Remove $u$ from $S_i$.
16:   if the acknowledgment is positive then
17:      Allocate the bandwidth on the link between the current node and node $u$ for
         $r_i^{FB}$. Mark $r_i^{FB}$ as "successful". Send a positive acknowledgment of $r_i^{FB}$ to the
         first node that is added to $V_i^{pre}$.
18:   else
19:      $n_i = n_i - 1$.
20:      if $n_i \leq 0$ then
21:         Mark $r_i^{FB}$ as "failed", and send a negative acknowledgment of $r_i^{FB}$ to all
            nodes in $V_i^{pre}$.
22:   Return to line 2.
```

The algorithm details for the fixed-bandwidth scheduling problem are provided in Algorithm 1. Each node maintains a job queue Q that stores bandwidth reservation requests. When a bandwidth reservation request arrives, Q is dynamically updated and the request state is changed. A bandwidth reservation request in Q is in one of three states: "pending", "successful", and "failed". The scheduling daemon waits for control messages and processes bandwidth reservation messages in lines 3-13 and acknowledgment messages in lines 14-22.

– When the current node receives a fixed-bandwidth reservation request r_i^{FB} from its neighbor node u, the algorithm first adds u to the node set V_i^{pre} that stores all the previous nodes from which r_i^{FB} is received and is used for sending back the acknowledgment. The algorithm then checks whether r_i^{FB} is in Q. If r_i^{FB} is not in Q, the algorithm adds r_i^{FB} to Q, marks r_i^{FB} as "pending", and replies with a positive acknowledgment if r_i^{FB} is destined to itself; otherwise, the algorithm computes the potential qualified neighbor node set S_i. A neighbor node is qualified if r_i^{FB} can be successfully scheduled on the link between the current node and that neighbor node based on its time-bandwidth list. If S_i is not empty, the current node broadcasts r_i^{FB} to all the nodes in S_i, which is an expansion performed in BFS. If S_i is an empty set, which means that there does not exist any qualified neighbor node, the current node replies with a negative acknowledgment to u where r_i^{FB} is received from and marks r_i^{FB} in Q as "failed". In the case that r_i^{FB} is in Q but r_i^{FB} is already marked as "failed", the current node sends a negative acknowledgment of r_i^{FB} to u since there does not exist any feasible path that passes the current node to satisfy r_i^{FB}.

– When the current node receives an acknowledgment of r_i^{FB} from its neighbor u and u is in S_i, the algorithm removes u from S_i to avoid receiving duplicate acknowledgments from the same neighbor node and checks whether the acknowledgment is positive or negative. If the acknowledgment is positive, the algorithm allocates the bandwidth on the link between the current node and u for r_i^{FB} and sends a positive acknowledgment of r_i^{FB} to the first node that is added to V_i^{pre}. Otherwise, the algorithm decreases n_i by 1. Note that n_i is initialized to be $|S_i|$ in line 10 and is used to count the number of negative acknowledgments received from neighbor nodes in S_i. If n_i reaches 0, which indicates that the current node receives negative acknowledgments from all the nodes in S_i and there does not exist a qualified link of the current node, the algorithm marks r_i^{FB} as "failed" and sends a negative acknowledgment of r_i^{FB} to all the nodes in V_i^{pre}. An example of the algorithm processing acknowledgment messages is shown in Fig. 3, where the solid line represents the fixed-bandwidth reservation request and the dashed line represents the acknowledgment. The current node is v_3 that receives r_i^{FB} from both v_1 and v_2, and broadcasts it to v_4 and v_5. In this example, $V_i^{pre} = \{v_1, v_2\}$, $S_i = \{v_4, v_5\}$. The positive acknowledgment process is shown in Fig. 3 (a): once v_3 receives a positive acknowledgment of r_i^{FB} from one node in S_i (v_5), it sends the acknowledgment to the node that is firstly added to V_i^{pre} (v_1). The negative acknowledgment process is shown in Fig. 3 (b): only when v_3 receives negative acknowledgments of r_i^{FB} from all the nodes in S_i, v_3 broadcasts the negative acknowledgment to all the nodes in V_i^{pre}. The path exploration process for r_i^{FB} is terminated when the source node of r_i^{FB} receives an acknowledgment.

Performance Tuning. Algorithm 1 is simple and scalable, but some extra work is needed to improve its performance. A deadlock may occur during the acknowledgment message processing, as shown in Fig. 4, where there is a cycle of r_i^{FB} among v_2, v_3 and v_4, but v_2 only sends r_i^{FB} to v_3 once. With the qualified neighbor node set $S_i = \{v_2, v_5\}$ for r_i^{FB}, v_4 receives a negative acknowledgment from v_5 and is waiting for the acknowledgment from v_2 before sending any acknowledgments to v_3. However, v_2 is waiting for the acknowledgement from v_3 and v_3 is waiting for the acknowledgment from v_4. Therefore, there is a deadlock among v_2, v_3 and v_4. To address this issue, we can encode a set of nodes that a bandwidth reservation request have traversed. When the algorithm computes S_i for r_i^{FB}, S_i only includes the qualified neighbor nodes that are not in the set

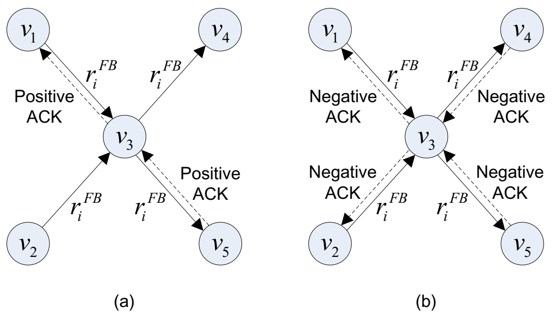

Fig. 3. An example of the acknowledging process for the fixed-bandwidth problem

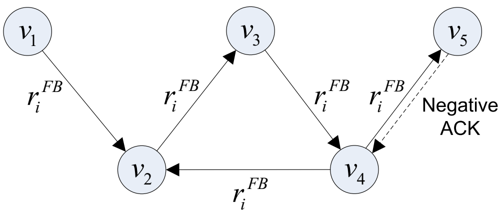

Fig. 4. An example of deadlock in Algorithm 1

of nodes that r_i^{FB} have traversed. Therefore, there is no bandwidth reservation request from v_4 to v_2 in the above example and the deadlock is avoided.

In a special case where a neighbor node u in S_i breaks down right after the current node broadcasted r_i^{FB}, and the current node receives negative acknowledgments from all the nodes in S_i except u, n_i never reaches 0 in line 23 of Algorithm 1. If there does not exist a feasible path to satisfy r_i^{FB}, the source node of r_i^{FB} may never receive a negative acknowledgment. The solution is that detecting the breakdown of a neighbor node u in S_i should be equivalent to receiving a negative acknowledgment from u for the pending bandwidth reservation request r_i^{FB}. Also, all the scheduled bandwidth reservations using u must be canceled. In this case, the current node sends a CANCEL message along the path for each scheduled bandwidth reservation request using the current node and u, and the reserved bandwidths on the corresponding links will be released. Once the source node of a bandwidth reservation request receives a CANCEL message, the source node initiates another path exploration process to find a new path. The handling of node failures can also be applied to the rest algorithms.

Algorithm Analysis. Algorithm 1 exhibits several salient features.

(i) Loop free: The job queue that maintains all incoming bandwidth reservation requests and the verification condition in line 5 ensure that each node broadcasts a bandwidth reservation request at most once. Hence, there is no loop for a bandwidth reservation request. Furthermore, the condition of whether u is in S_i in line 17 and the update of S_i in line 18 ensure that each node receives at most one acknowledgment from a neighbor node in S_i. Hence, there is no loop for an acknowledgment.

(ii) Fault tolerant: Any node and link failures can be detected by periodical HELLO messages exchanged between nodes. Hence, any node failure does not affect the path exploration process if there still exists a feasible path.

(iii) Time efficient: The runtime complexity of this algorithm is $O(T \cdot m)$ in the worst case. Unlike most distributed routing algorithms where each node must wait for a constant time period to collect all messages from its neighbor nodes, this algorithm processes each incoming routing message immediately to speed up path exploration. In the worst case, the algorithm involves $O(m)$ message communications in the entire network.

3.2 Highest-Bandwidth

Given a highest-bandwidth reservation request r_i^{HB}, the highest-bandwidth scheduling problem is to compute a dedicated channel from source node v_s to destination node v_d with the highest available bandwidth during time slot $[t_s, t_e]$. This problem can be solved by extending Dijkstra's shortest path algorithm in the centralized scheme. We propose a distributed solution based on Bellman-Ford algorithm to this problem. The source node v_s receives r_i^{HB} from an end user, initializes the highest bandwidth of r_i^{HB} to be infinity, and initiates the path exploration process by broadcasting r_i^{HB} to its neighbor nodes. The neighbor nodes compute their highest bandwidth according to the incoming r_i^{HB}, and broadcast their results only if the bandwidth value is increased. Note that the highest available bandwidth of the entire path is determined by the bottleneck bandwidth of all component links in the specified time slot. Hence, the highest bandwidth of r_i^{HB} on each node is dynamically updated during the path exploration process.

The algorithm details for the highest-bandwidth scheduling problem are provided in Algorithm 2. Let $BW_i(v_s, v_{cur})$ denote the highest bandwidth of the path found so far from source node v_s to the current node v_{cur} for r_i^{HB} in Q, and $BW_i'(v_s, v_{cur})$ denote that for the incoming r_i^{HB}. The algorithm waits for control messages, and processes bandwidth reservation messages in lines 3-17 and acknowledgment messages in lines 18-20. When the current node receives a highest-bandwidth reservation request r_i^{HB} from its neighbor node u, the algorithm checks whether r_i^{HB} is in Q; if not, the algorithm adds r_i^{HB} to Q. If the highest bandwidth of the incoming r_i^{HB} is larger than that of r_i^{HB} in Q (i.e. $BW_i'(v_s, v_{cur}) > BW_i(v_s, v_{cur})$), the algorithm updates the highest bandwidth of r_i^{HB} in Q. The algorithm computes the highest bandwidth of the path found so far from source node v_s to every neighbor node v by calculating $BW_i(v_s, v) = min\{BW_i(v_s, v_{cur}), BW_i(v_{cur}, v)\}$, where $BW_i(v_{cur}, v)$ is the highest bandwidth of the link (v_{cur}, v) during the time slot specified in r_i^{HB}. The algorithm encodes $BW_i(v_s, v)$ in r_i^{HB} and sends r_i^{HB} to v. If the highest bandwidth of r_i^{HB} in Q does not increase (i.e. $BW_i'(v_s, v_{cur}) \leq BW_i(v_s, v_{cur})$), the algorithm returns to line 2 directly to avoid message broadcasting. If the

Algorithm 2. Scheduling algorithm for the highest-bandwidth problem

```
1: Create a job queue $Q$ to store all reservation requests.
2: Wait for routing messages.
3: if a highest-bandwidth reservation request $r_i^{HB}$ is received from its neighbor node $u$ then
4:     if $r_i^{HB}$ is destined to the current node then
5:         Restart a timer for $r_i^{HB}$.
6:     if $r_i^{HB}$ is not in $Q$ then
7:         Add $r_i^{HB}$ to $Q$.
8:     else if $BW_i'(v_s, v_{cur}) > BW_i(v_s, v_{cur})$ then
9:         $BW_i(v_s, v_{cur}) = BW_i'(v_s, v_{cur})$
10:    else
11:        Return to line 2.
12:    Set $v_i^{pre} = u$.
13:    if $r_i^{HB}$ is not destined to the current node then
14:        Compute the neighbor node set $S_i$ (excluding $u$).
15:        for all $v \in S_i$ do
16:            $BW_i(v_s, v) = min\{BW_i(v_s, v_{cur}), BW_i(v_{cur}, v)\}$. Encode $BW_i(v_s, v)$ in $r_i^{HB}$ and send $r_i^{HB}$ to $v$.
17:    Return to line 2.
18: if an acknowledgment of request $r_i^{HB}$ is received from its neighbor node $u$ then
19:    Allocate the bandwidth on the link between the current node and node $u$ for $r_i^{HB}$. Forward acknowledgment of $r_i^{HB}$ to node $v_i^{pre}$.
20:    Return to line 2.
```

current node is the destination node of r_i^{HB}, the algorithm restarts a timer for r_i^{HB}. This timer is used by the destination node to acknowledge the granting of the request since the destination node does not know when the path exploration process reaches an equilibrium. If the destination node does not receive any updated r_i^{HB} from its neighbor nodes for a period of time, it is very likely that the path exploration process for r_i^{HB} has reached an equilibrium. Once the timer for r_i^{HB} expires, the destination node determines the highest bandwidth of the entire path and sends an acknowledgment of r_i^{HB} that carries the highest bandwidth to v_i^{pre}, which is the best neighbor node on the widest path from the source node to the current node. When the current node receives an acknowledgment of r_i^{HB} from its neighbor node u, it allocates the bandwidth on the link between the current code and node u, and forwards the acknowledgment of r_i^{HB} to node v_i^{pre}. This backtracking process continues until the source node of r_i^{HB} receives the acknowledgment.

Performance Tuning. The time cost of the path exploration process is affected by the timer on the destination node, which needs to be carefully decided according to the network size and link delay. Let $DELAY$ denote the average delay of a message communication between two adjacent nodes, which includes the processing delay on two end nodes and the link delay between them. A node can estimate $DELAY$ by measuring the round trip time of a message between itself and its neighbor nodes. Since a bandwidth reservation message traverses at least 1 hop and at most $n-1$ hops from v_s to v_d, the difference between the arrival times of any two request messages is at most $(n-2) \cdot DELAY$, which could be used to set the timer on v_d.

Algorithm Analysis. Algorithm 2 also exhibits similar features as Algorithm 1.

(i) Loop free: A node broadcasts r_i^{HB} to its neighbor nodes only when the highest bandwidth of r_i^{HB} increases. When a node broadcasts r_i^{HB} to its neighbors, the highest bandwidth of r_i^{HB} does not increase during the path exploration process. Hence, there is no loop for a bandwidth reservation request. Since node v_i^{pre} is set only when the highest bandwidth of r_i^{HB} increases and the acknowledgment is sent to v_i^{pre}, there is no loop for an acknowledgment, either.

(ii) Fault tolerant: Since each node makes a local decision and acts as an autonomous system, a node or link failure would not affect the path exploration process. If one node on the computed path for r_i^{HB} breaks down after an equilibrium is achieved but before the acknowledgment of r_i^{HB} is forwarded, the source node of r_i^{HB} will never receive the acknowledgment. This problem can be solved as follows. The failure of a node can be detected by its neighbor nodes by periodical HELLO message exchanges between them. Each neighbor node then sends a negative acknowledgment of r_i^{HB} to its v_i^{pre}. The source node of r_i^{HB} eventually receives the negative acknowledgment of r_i^{HB} and may initiate another path exploration process for r_i^{HB}.

(iii) Time efficient: The runtime complexity of this algorithm is $O(m \cdot T)$ in the worst case. Except for the destination node, all other nodes process each incoming routing message immediately. In the worst case, the algorithm requires $O(n^3)$ message broadcasting as the distributed Bellman-Ford algorithm.

3.3 First-Slot and All-Slots

Given a first-slot or all-slots bandwidth reservation request, r_i^{FS} or r_i^{AS}, the first-slot or all-slots bandwidth scheduling problem is to compute the time slot with the earliest start time or all possible time slots of a dedicated channel from v_s to v_d with a fixed bandwidth β for duration t_d. Obviously, first-slot is a special case of all-slots, and the solution to all-slots can be applied to first-slot. We propose a distributed algorithm based on Bellman-Ford algorithm for these two problems.

We first define a list of start time slots $[t_i, t_{i+1}]$ for each link $l \in E$, denoted as $ST(l)$. For any time point t during a start time slot $[t_i, t_{i+1}]$, i.e. $t \in [t_i, t_{i+1}]$, link l has available bandwidth of β from time point t to time point $t + t_d$. The time slots on ST are disjoint and arranged in an ascending order. The ST list of a link can be constructed from its TB list in $O(T)$ time, and the ST list of a path can be constructed by combining the ST lists of all component links. Let $ST(v_s, v)$ denote the union of the ST lists of all paths from source node v_s to node v. Hence, $ST(v_s, v_d)$ contains all start time slots of all paths from v_s to v_d with bandwidth β for duration t_d. Let $\bigoplus$ and $\bigotimes$ denote the point-wise merging and intersection operations of the time slots in two ST lists, respectively. We have $ST(l) \bigoplus \emptyset = ST(l)$, $ST(l) \bigoplus \Re^+ = \Re^+$, $ST(l) \bigotimes \emptyset = \emptyset$,

Algorithm 3. Scheduling algorithm for the all-slots problem

```
 1: Create a job queue $Q$ to store all reservation requests.
 2: Wait for routing messages.
 3: if an all-slots bandwidth reservation request $r_i^{AS}$ is received from its neighbor node
    $u$ then
 4:   if $r_i^{AS}$ is destined to the current node then
 5:     Restart a timer for $r_i^{AS}$.
 6:   if $r_i^{AS}$ is not in $Q$ then
 7:     Add $r_i^{AS}$ to $Q$. Set $v_i^{pre} = u$.
 8:   else if $ST_i'(v_s, v_{cur}) \subsetneq ST_i(v_s, v_{cur})$ then
 9:     $ST_i(v_s, v_{cur}) = ST_i(v_s, v_{cur}) \bigoplus ST_i'(v_s, v_{cur})$.
10:   else
11:     Return to line 2.
12:   if $r_i^{AS}$ is not destined to the current node then
13:     Compute the neighbor node set $S_i$ (excluding $u$).
14:     for all $v \in S_i$ do
15:       $ST_i(v_s, v) = ST_i(v_s, v_{cur}) \bigotimes ST_i(v_{cur}, v)$. Encode $ST_i(v_s, v)$ in $r_i^{AS}$ and
          send $r_i^{AS}$ to $v$.
16:   Return to line 2.
17: if an acknowledgment of request $r_i^{AS}$ is received from its neighbor node $u$ then
18:   Forward acknowledgment of $r_i^{AS}$ to node $v_i^{pre}$.
19:   Return to line 2.
```

and $ST(l) \bigotimes \Re^+ = ST(l)$, where $\emptyset$ is the empty time slot and $\Re^+$ is the infinite time slot of non-negative real values.

The algorithm details for the all-slots bandwidth scheduling problem are provided in Algorithm 3. The source node v_s receives r_i^{AS} from an end user, initializes the ST list of r_i^{AS} to be $ST(v_s, v_s) = \Re^+$, and initiates the path exploration process by broadcasting r_i^{AS} to its neighbor nodes. Let $ST_i(v_s, v_{cur})$ denote the list of start time slots of the paths found so far from source node v_s to the current node v_{cur} for r_i^{AS} in Q, and $ST_i'(v_s, v_{cur})$ denote that for the incoming r_i^{AS}. The algorithm is modified from Algorithm 2 by replacing the bandwidth operation with the ST list operation. If the ST list of the incoming r_i^{AS} is not a subset of the ST list of r_i^{AS} in Q (i.e. $ST_i'(v_s, v_{cur}) \subsetneq ST_i(v_s, v_{cur})$), the algorithm updates the ST list of r_i^{AS} in Q (i.e. $ST_i(v_s, v_{cur}) = ST_i(v_s, v_{cur}) \bigoplus ST_i'(v_s, v_{cur})$). Here, the relationship $\subsetneq$ of two ST lists holds if at least one time slot in $ST_i'(v_s, v_{cur})$ does not belong to any time slots on $ST_i(v_s, v_{cur})$. Due to the monotonicity property of $\bigoplus$ operation, once start time slots are placed on $ST_i(v_s, v_{cur})$, they will not be removed. The algorithm then computes the start time slots of all paths found so far from source node v_s to every neighbor node v by calculating $ST_i(v_s, v) = ST_i(v_s, v_{cur}) \bigotimes ST_i(v_{cur}, v)$, where $ST_i(v_{cur}, v)$ is the ST list of link (v_{cur}, v) for r_i^{AS}. The algorithm encodes $ST_i(v_s, v)$ in r_i^{AS} and sends r_i^{AS} to v. If the current node is the destination node of r_i^{AS}, the algorithm restarts a timer for r_i^{AS}. Once the timer for r_i^{AS} expires, the destination node sends an acknowledgment of r_i^{AS} that carries all start time slots on $ST_i(v_s, v_d)$ to v_i^{pre}.

For the first-slot problem, the earliest start time is the lower boundary of the first time slot on the returned ST list. For the all-slots problem, the end user at the source node may choose one or multiple start times from the returned ST list. Once the start time t for a feasible path is decided, we can apply Algorithm 1 for the fixed-bandwidth problem to perform the actual path computation and bandwidth scheduling with $t_s = t$ and $t_e = t + t_d$.

The runtime complexity of Algorithm 3 is $O(m)$ in terms of $\bigoplus$ and $\bigotimes$ operations. Since the complexities of $\bigoplus$ and $\bigotimes$ operations are determined by the length of the ST list, which is at most $m \cdot T$ in the algorithm, the complexities of $\bigoplus$ and $\bigotimes$ operations are of $O(m \cdot T)$. Therefore, the algorithm complexity is of $O(m^2 \cdot T)$ in the worst case. Due to the similarity in the algorithm structure, the performance tuning and algorithm analysis for Algorithm 2 are applicable to Algorithm 3.

4 Performance Evaluation

We perform simulation-based evaluations for the proposed distributed scheduling algorithms. For performance comparison, we also design and implement a simple greedy algorithm. In the simulations, each simulated network is randomly generated with an arbitrary network topology with 50 nodes and 200 links, and the TB list of each link is also randomly generated with residual bandwidths ranging from 0.2 Gbps to 10 Gbps in each time slot with an identical length of 1 second. The residual bandwidths follow a normal distribution:

$$b_l[i] = 0.2 + 10 \cdot (1 - e^{-\frac{1}{2}(3x)^2}), \tag{1}$$

where x is a random variable within the range of [0,1]. There are 600 time slots in the TB list of each link.

4.1 Experimental Results for Algorithm 1

We conduct performance comparison between Algorithm 1 and the traceroute-based method for the fixed-bandwidth problem using various simulated networks. Note that traceroute is implemented in OSCARS to find the shortest path within ESnet that MPLS LSP traverses [14]. Once the entire path controlled by OSCARS is obtained, each link on the path is then checked for available bandwidth.

Fixed-bandwidth is a decision problem and the satisfiability of a fixed-bandwidth request is determined by the availability of the network resources. Algorithm 1 is an optimal algorithm that is able to find a feasible solution when there exists one. We randomly generate 200 network instances of different topologies, in each of which, we randomly generate a series of fixed-bandwidth requests with requested bandwidth β ranging from 0.24 Gbps to 2.4 Gbps at an interval of 0.24 Gbps. The duration of a request $t_e - t_s$ is constrained within the range of [1, 10]. We run Algorithm 1 and traceroute on these fixed-bandwidth requests and plot a series of acceptance rates in response to different β values in Fig. 5. The acceptance rate is defined as the ratio of successfully scheduled requests

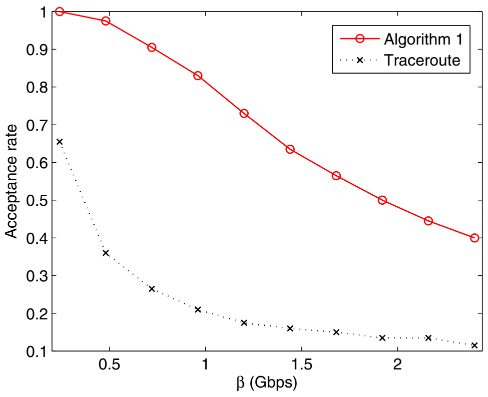

Fig. 5. Acceptance rates of Algorithm 1 and traceroute for the fixed-bandwidth problem

and the total 200 submitted requests. We observe that Algorithm 1 exhibits superior performance over the traceroute-based method. Since the requests with larger β values require more network resources, the acceptance rate decreases as β increases.

4.2 Experimental Results for Algorithm 2

We compare the performance of Algorithm 2 with that of a greedy algorithm for the highest-bandwidth problem. In the greedy algorithm, a node always chooses one neighbor node whose link has the highest available bandwidth in the specified time slot. In each of 200 randomly generated network instances, we generate a series of highest-bandwidth requests with duration $t_d = t_e - t_s$ ranging from 1 to 10 seconds at an interval of 1 second. We run Algorithm 2 and the greedy algorithm on these highest-bandwidth requests and plot the average and standard deviation of the highest available bandwidth in response to different t_d values in Fig. 6. We observe that Algorithm 2 outperforms the greedy approach in all the cases we studied. We also observe that the average highest available bandwidth decreases as t_d increases.

4.3 Experimental Results for Algorithm 3

Algorithm 3 is designed for both the first-slot and all-slots problems. We first compare the performance of Algorithm 3 with that of a greedy method for the first-slot problem. In the greedy algorithm, a node always chooses one neighbor node such that the earliest start time of the path from the source node to the neighbor node for that request is minimized. We randomly generate a series of first-slot requests with $t_d = 5$ seconds and requested bandwidth β ranging from 0.24 Gbps to 2.4 Gbps at an interval of 0.24 Gbps. We plot the average and

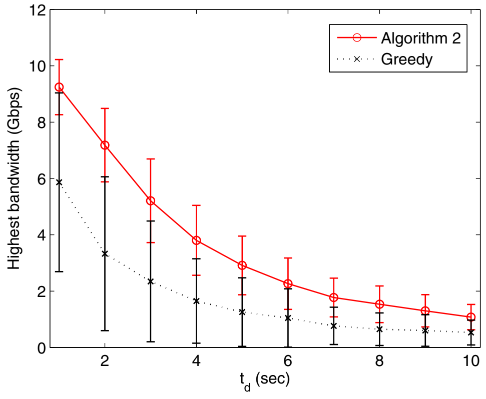

Fig. 6. The highest bandwidths (mean and standard deviation) of Algorithm 2 and the greedy method for the highest-bandwidth problem

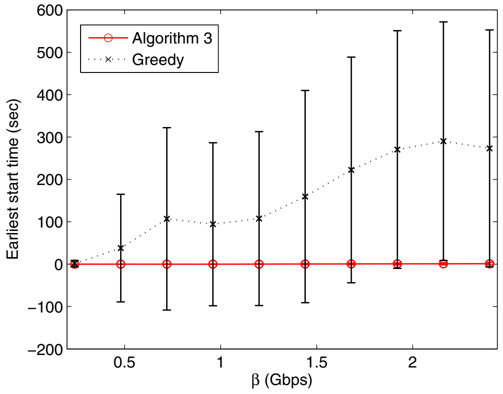

Fig. 7. The earliest start time (mean and standard deviation) of Algorithm 3 and the greedy method for the first-slot problem

standard deviation of the earliest start time in response to different β values in Fig. 7. In most of the cases, the earliest start time computed by Algorithm 3 is 0 second. The largest earliest start time is 600 seconds since we assume that there is no bandwidth reservation on each link after 600 seconds. We observe that the average of 200 earliest start times computed by Algorithm 3 is much less than that computed by the greedy method.

We also compare the performance of Algorithm 3 with that of a greedy method for the all-slots problem. The objective function in the all-slots problem is the total length of start times. In the greedy method, a node always chooses one neighbor node such that the total length of start times of the path from the source node to the neighbor node for that request is maximized. The simulation

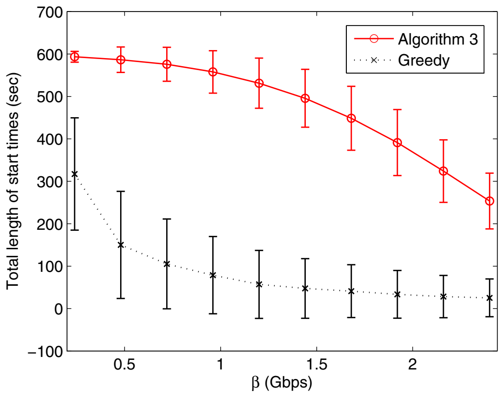

Fig. 8. The total length of start times (mean and standard deviation) of Algorithm 3 and the greedy method for the all-slots problem

settings for the all-slots problem are the same as those for the first-slot problem. We plot the average and standard deviation of the total length of start times in response to different β values in Fig. 8, and observe that the performance superiority of Algorithm 3 is dominant in all the cases.

5 Conclusion

We formulated four basic bandwidth scheduling problems in high-performance networks that support advance bandwidth reservations. We proposed a distributed algorithm for each of these problems. These algorithms are based on a rigorous extension of the classical breadth first search and Bellman-Ford algorithms. The extensive experimental results in a large set of simulated networks demonstrate the performance superiority of these algorithms in comparison with greedy approaches.

References

1. DRAGON: Dynamic Resource Allocation via GMPLS Optical Networks, `http://dragon.maxgigapop.net`
2. JGN II: Advanced Network Testbed for Research and Development, `http://www.jgn.nict.go.jp`
3. Geant2, `http://www.geant2.net`
4. OSCARS: On-demand Secure Circuits and Advance Reservation System, `http://www.es.net/oscars`
5. HOPI: Hybrid Optical and Packet Infrastructure, `http://networks.internet2.edu/hopi`
6. GENI: Global Environment for Network Innivations, `http://www.geni.net`

7. CHEETAH: Circuit-switched High-speed End-to-End Transport ArcHitecture, http://www.ece.virginia.edu/cheetah
8. Cohen, R., Fazlollahi, N., Starobinski, D.: Graded channel reservation with path switching in ultra high capacity networks. In: Proc. of Broadnets, San Jose, CA (2006)
9. Cohen, R., Fazlollahi, N., Starobinski, D.: Path switching and grading algorithms for advance channel reservation architectures. IEEE/ACM Transactions on Networking 17(5), 1684–1695 (2009)
10. Ganguly, S., Sen, A., Xue, G., Hao, B., Shen, B.H.: Optimal routing for fast transfer of bulk data files in time-varying networks. In: Proc. of IEEE Int. Conf. on Communications (2004)
11. Gorinsky, S., Rao, N.S.V.: Dedicated channels as an optimal network support for effective transfer of massive data. In: INFOCOM 2006 Workshop on High-Speed Networks (2006)
12. Grimmell, W.C., Rao, N.S.V.: On source-based route computation for quickest paths under dynamic bandwidth constraints. Int. J. on Foundations of Computer Science 14(3), 503–523 (2003)
13. Guerin, R.A., Orda, A.: Networks with advance reservations: the routing perspective. In: Proc. of the 19th IEEE INFOCOM (2000)
14. Guok, C., Robertson, D., Thompson, M., Lee, J., Tierney, B., Johnston, W.: Intra and interdomain circuit provisioning using the oscars reservation system. In: Proc. of the BROADNETS, San Jose, CA, October 1-5, pp. 1–8 (2006)
15. Lin, Y., Wu, Q.: Path computation with variable bandwidth for bulk data transfer in high-performance networks. In: Proceedings of INFOCOM HSN Workshop, Rio de Janeiro, Brazil (April 24, 2009)
16. Lin, Y., Wu, Q., Rao, N.S.V., Zhu, M.: On design of scheduling algorithms for advance bandwidth reservation in dedicated networks. In: The 2008 INFOCOM High-Speed Networks Workshop, Phoenix, Arizona (April 13, 2008)
17. Rao, N.S.V., Wing, W.R., Carter, S.M., Wu, Q.: Ultrascience net: Network testbed for large-scale science applications. IEEE Communications Magazine 43(11), s12–s17 (2005), An expanded version www.csm.ornl.gov/ultranet
18. Rao, N.S.V., Wu, Q., Carter, S.M., Wing, W.R., Ghosal, D., Banerjee, A., Mukherjee, B.: Control plane for advance bandwidth scheduling in ultra high-speed networks. In: INFOCOM 2006 Workshop on Terabits Networks (2006)
19. Sahni, S., Rao, N.S.V., Ranka, S., Li, Y., Jung, E., Kamath, N.: Bandwidth scheduling and path computation algorithms for connection-oriented networks. In: Proc. of Int. Conf. on Networking (2007)
20. Veeraraghavan, M., Lee, H., Chong, E.K.P., Li, H.: A varying-bandwidth list scheduling heuristic for file transfers. In: Proc. of IEEE Int. Conf. on Communications (2004)
21. Zhang, Z.L., Duan, Z., Hou, Y.T.: Decoupling QoS control from core routers: A novel bandwidth broker architecture for scalable support of guaranteed services. In: Proc. of ACM SIGCOMM (2000)