Path Switching and Grading Algorithms for Advance Channel Reservation Architectures

Reuven Cohen, Niloofar Fazlollahi, and David Starobinski

***Abstract*—As a result of perceived limitations of TCP/IP in supporting high throughput applications, significant efforts have recently been devoted to develop alternative architectures based on the concept of advance channel reservation. In this paper, we develop a polynomial-time algorithmic framework, called Graded Channel Reservation (`GCR`), to support the implementation of such architectures. This framework enables users to specify minimum bandwidth and duration requirements for their connections. Upon receiving a request, `GCR` returns the highest graded path, selected according to a general, multi-criteria optimization objective. In particular, if the optimization criterion is delay, we prove that `GCR` returns the earliest time available to establish the connection. Thereafter, we present a generalization of `GCR`, called $\texttt{GCR}_{\texttt{switch}}$, that is capable of supporting path switching throughout a connection. We present practical methods for minimizing or limiting the number of path switches. Through extensive simulations, we evaluate the performance of `GCR` and its variants under various topological settings and applications workload. Our results show that, for certain traffic parameters, optimized path selection combined with path switching can reduce the average delay of requests by an order of magnitude and increase the maximum sustainable load by as much as 50%.**

***Index Terms*—Algorithms, reservation, scheduling, routing, switching, performance evaluation.**

I. Introduction

Network backbones are often presumed to be over-provisioned. Yet, the emergence of new applications with unprecedented bandwidth requirements is likely to quickly change the current state of affairs.

For instance, future Grid applications will require transfer of extremely large files between different national labs and research centers [1]. As a simple illustration, experiments run on the new Large Hadron Collider (LHC) accelerator at CERN are expected to generate prodigious volume of data, reaching the order of ExaBytes (1 ExaByte = 10^{18} bytes). This data will have to be transferred from CERN to various sites around the world, for the purpose of storage, processing, and analysis [2].

Needs for large file transfers are not confined to Grid applications. For instance, many corporations rely on distributed storage area networks (SANs) to seamlessly perform various information management functions such as data backup, mirroring, and recovery [3]. To implement the above functions, distributed SANs must support quick and reliable transfer of very large files between remote sites.

In order to meet the throughput and delay requirement of the above applications, one must be able to make full use of the network backbone resources. Yet, recent experiments have shown that the standard TCP/IP protocol stack may be inadequate for this purpose. Indeed, it has been observed that, in ultra high speed networks, there is a large gap between the capacity of network links and the maximum end-to-end throughput achieved by TCP [4]. The major cause of this discrepancy is the shared nature of TCP/IP where e-mails and WWW traffic interfere with large file transfers [5].

As a result of the existing limitations of TCP/IP, significant efforts have recently been devoted to develop an alternative protocol stack based on the concept of *advance channel reservation* [5–11], that is specifically tailored for large file transfers and other high throughput applications. This protocol stack is not intended to replace TCP/IP but rather complement it. The most important property of advance channel reservation is to offer hosts and users the ability to reserve in advance *dedicated* channels (paths) to connect their resources.

Advance channel reservation protocols are run directly on top of Layer 2 (SONET or Gigabit Ethernet) and thus bypass IP. They make it possible for users to send requests and specify minimum bandwidth and duration requirements for their connection. Several testbeds, such as UltraScience Net [5] and OSCARS [11], are testing possible implementations of such protocols.

At a first glance, advance channel reservation architectures may appear similar to standard circuit switching architectures. One of the major contributions in this paper is to show that these two types of architecture do actually differ in some fundamental ways and that advance channel reservation architectures offer a great deal of flexibility that can be exploited to significantly improve performance.

First, advance reservation architectures allow to schedule the starting time of a connection so that general, multi-criteria optimization objectives can be satisfied. We can thus perform *path grading*, that is, assign grades to paths according to a certain optimization objective and then select the highest graded path. Examples of possible grading, as explained in the sequel, include selecting the earliest path, shortest path, widest path, or a combination of those, satisfying certain bandwidth and duration requirements.

Second, advance reservation architectures do not restrict a connection to use the same path over its entire duration. Thus, they make it possible to implement *path switching*, where a connection can switch between different physical paths throughout its life-time. As shown in the sequel, path switching can substantially reduce the average delay until connection establishment as well as the blocking probability of requests. We note here that all the existing advance reservation protocols and algorithms proposed so far in the literature (cf.

R. Cohen is affiliated with the Dept. of Mathematics, Bar-Ilan University, Ramat-Gan, Israel

N. Fazlollahi and D. Starobinski are affiliated with Boston University, Boston, MA

A preliminary version of this paper appeared in the proceedings of the IEEE Gridnets 2006 workshop.

Section II) allocate the same path for the entire connection duration, similar to circuit switching.

Guided by the above observations, we introduce in this paper a new algorithmic framework for advance channel reservation called Graded Channel Reservation (GCR) that implements the path grading and path switching design principles. GCR enables grading paths, so that the path with the highest grade is selected. For general optimization criteria, we prove that GCR has a computational complexity that is polynomial in the size of the graph and the maximum number of pending requests at any time.

We extend GCR so as to support path switching. Our generalized framework, called $\texttt{GCR}_{\texttt{switch}}$, satisfies the same properties as GCR, but also allows a connection to switch between different paths throughout its duration. Furthermore, we propose a variant called $\texttt{GCR}_{\texttt{minimum}}$ that provably performs the minimum number of path switches needed while returning the highest graded paths.

It should be mentioned that GCR is assumed to be run in a centralized environment. This is a reasonable scenario in small networks, which reflect the operations of current testbeds. The algorithms presented here can also be applied in a distributed manner using a link-state mechanism in conjunction with an appropriate signalling protocol (cf. Section II).

Our simulations, run for various topologies and traffic parameters, demonstrate the importance of judicious path grading. For instance, when the main grading criterion is finding the earliest path available, then the use of a secondary criterion based on the selection of the shortest path (when several earliest paths are available) significantly improves performance.

Our simulations also reveal that path switching leads to major performance gain. In some cases, it reduces the average delay by up to an order of magnitude and increases maximum sustainable load by as much as 50%. Significant performance improvement is observed even if (for practical reasons) a limit is imposed on the number of switching permitted throughout the duration of a connection.

The rest of this paper is organized as follows. In Section II, we review related work on advance channel reservation. In section III, we introduce the GCR algorithmic framework, prove its main properties, and explain how traffic engineering mechanisms, such as trunk reservation, can be used in conjunction. In section IV, we introduce the concept of path switching and analytically illustrate its benefit for a small network. We then show how path switching can be integrated into GCR and introduce the $\texttt{GCR}_{\texttt{minimum}}$ variant that minimizes the number of path switches. In Section V, we present simulation results evaluating the performance of our algorithms under various network topologies and traffic parameters. We conclude the paper in Section VI.

II. Related Work

The topic of advance resource reservation has received considerable attention in the literature. A great portion of which concentrates on the design of distributed signalling protocols [12–16]. For instance, ref. [13] discusses possible modification to RSVP to support advance channel reservation.

Several papers have considered the problem of joint routing and scheduling of file transfers. In [17], resource reservation strategies are analyzed for specific topologies (stars, trees, and trees of rings). In [6], a scheduling algorithm is introduced for large file transfers over LambdaGrids for paths with varying bandwidth. In [18] the problem of offline scheduling and routing of file transfers from several users, each storing multiple files, to a single receiver node is analyzed. Ref. [7] considers a similar model but also proposes algorithms to address the problem of rescheduling connections that have not completed. Refs. [19, 20] propose various load balancing approaches to allocate lightpaths.

As in our paper, ref. [21] investigates formal approaches to advance reservation and provides theoretical analysis of the complexity of path selection. The proposed algorithms in [21] for advance channel reservation resemble our framework in the sense that they segment time and keep track of future link residual bandwidths. However, their segmentation assumes that the time axis is discretized, thus leading to performance loss. Our model does not have this limitation. In addition, no performance analysis or simulation results are provided in [21], and path switching and multi-criteria optimizations are not considered.

The most relevant work on advance resource reservation is the algorithm currently implemented on the UltraScience Net, referred to as ALL-SLOTS [5]. To find a path, ALL-SLOTS implements a variant of the Floyd-Warshall based on a union/intersection algebra instead of the standard min/+ algebra. Because of the union operation, ALL-SLOTS needs sometimes to discard overlapping intervals. As a consequence, there is no guarantee of finding a path with a desired property, e.g. the shortest, earliest, or widest. Moreover, the returned paths are kept fixed during the entire connection, i.e., there is no path switching.

III. Model and Algorithms

A. Notation and model

Our model consists of a general, directed network topology denoted by $G(V, E)$, where V is a set of nodes and E is a set of links. Each request can be expressed by a tuple (s, d, B, T, t_a, t_b), where $s \in V$ is the source node, $d \in V - \{s\}$ is the destination node, B is the requested bandwidth which is held fixed during the connection, T is the requested communication duration, $[t_a, t_b]$ specifies a time window during which the user wants the transmission to start. The parameters t_a and t_b are optional, and if omitted, they can be interpreted as the arrival time of the request, denoted by t_{now}, and as ∞, accordingly.

The reply to a request is a tuple (t^*, P^*) (or as we will explain later a vector of tuples (t_i^*, P_i^*) when path switching is allowed), where t^* is the transmission starting time satisfying $t_a \leq t^* \leq t_b$, and P^* is a path from s to d containing only links with residual bandwidth of at least B. In the case that a request can not be served (which may happen if B is greater than the minimum link capacity of every possible path between s and d or if no path can be established during the interval $[t_a, t_b]$), then the algorithm returns BLOCKED.

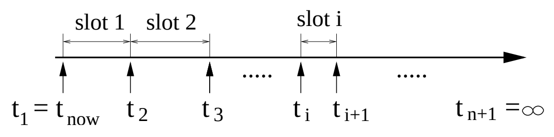


Fig. 1. Segmentation of time axis into slots delineated by events. During each slot, the state of each link in the network is fixed.

B. Basic algorithm

We now introduce our algorithmic framework, called *Graded Channel Reservation* (GCR). GCR returns a time slot that can accommodate a connection path according to a certain optimization objective in response to a request. The path can be graded according to any property of interest, such as, connection start time, path length[1], path width[2], or a combination of these. In this section, we illustrate the operation of this framework for the case where the earliest completion time is desired. Thus, we focus on finding the path allowing earliest task completion, in conjunction with other criteria, when more than one such path exists. We also discuss how the algorithm can be easily modified to handle different grading criteria.

To simplify exposition, we present the operation of GCR for the case where the graph $G(V, E)$ is undirected. However, our results (as well as simulations in Section V), apply to directed graphs. GCR uses the following procedure: it divides the time axis into slots delineated by events, as shown in Fig. 1. Each event corresponds to a set up or tear down instance of a connection. Therefore, during each time slot the state of all links in the network remains unchanged. In general, the time axis will consist of n time slots, where $n \geq 1$ is a variable and slot i corresponds to time interval $[t_i, t_{i+1}]$. Note that $t_1 = t_{now}$ and $t_{n+1} = \infty$. We denote by $L = \{t_1, \dots, t_{n+1}\}$ the ordered list of events. Every time a request arrives, we update L by setting $t_1 = t_{now}$ and discarding all elements $t_i < t_{now}$.

Let $W_i = \{b_1^i, b_2^i, \dots, b_{|E|}^i\}$ be the vector of available bandwidths of all links at time slot i where $i = 1, \dots, n$, and b_j^i denote the available bandwidth of link j during slot i, with $j = 1, \dots, |E|$. We then define the *bandwidth list* $W = \{W_1, \dots, W_n\}$ to be the ordered list of vectors of available bandwidths at each time slot $i = 1, \dots, n$. Upon the arrival of a request, we compute the largest value of i, denoted by i', such that $t_i \leq t_{now}$, i.e., $i' = \{\max i \in \{1, \dots, n\} \mid t_i \leq t_{now}\}$. We then update W by removing all the terms W_i for which i is less than i'.

Suppose a user sends a request tuple (s, d, B, T, t_a, t_b). We define $\bar{L}$ as the remainder of L after we omit all terms t_i, such that, $t_i < t_a$ or $t_i > t_b$. If t_a or t_b are not already included in L, they should be appended to the beginning and to the end of list $\bar{L}$ correspondingly. Similarly, we derive $\bar{W}$ from W by removing the terms W_i for which $t_i < t_{i'}$ or $t_i \geq t_b$, where i' is the largest value of i such that $t_i \leq t_a$.

This notation is illustrated in Figure 2 for a four-node clique, where for each time slot, only the links with sufficient residual bandwidth, i.e. bandwidth greater than or equal to B, are shown. The figure shows the duration of each slot, e.g., slot 1 lasts from time 1:00 pm to time 2:30 pm, slot 2 lasts from time 2:30 pm to time 3:00 pm, etc. In this case, $L = \{$1:00, 2:30, 3:00, 5:00, 6:00, $\infty\}$. Now, suppose t_a = 1:30 and t_b = 7:00, then $\bar{L} = \{$1:30, 2:30, 3:00, 5:00, 6:00, 7:00$\}$. Note that $\bar{W} = W$ in this example.

We now describe GCR and its functions. The pseudo-code of the main routine is as follows:

```
Algorithm GCR:
$t^* \leftarrow$ SlotSearch$(s, d, B, T, t_a, t_b)$
If $t^* \neq$ BLOCKED
    $P^* \leftarrow$ PathSearch$(t^*)$.
    $(L, W) \leftarrow$ Update$(t^*, P^*)$.
    Return $(t^*, P^*)$.
Else,
    Return (BLOCKED).
```

The function SlotSearch returns the time, denoted by t^*, of the highest graded time slot in $\bar{L}$ that can accommodate the request (s, d, B, T, t_a, t_b). If no such slot is available it returns BLOCKED which means that no slot was found and the request is rejected. We will present below an implementation of SlotSearch where the highest graded time slot is defined as the earliest slot in which a connection can be established. However, beforehand, we describe the other functions of GCR.

If SlotSearch result is not BLOCKED, then GCR calls the function PathSearch. This function returns a path P^*, between source s and destination d starting at time t^*. The path P^* is selected according to secondary optimization criteria, e.g., shortest path, widest path, narrowest path, or a combination of these. Combination of path properties, such as shortest-widest path, means priority is given to the shortest paths, but if multiple shortest paths are found, we return the widest among those whereas in the simple shortest path search, if multiple shortest paths exist, then one is just picked at random.

The function Update is used to update L and W after a request is allocated as follows: if the end time of the connection, $t_e^* = t^* + T$, is not already included in L, then Update adds the event t_e^* to L in the right position so as to maintain the increasing order of elements of L. After updating L, it also updates W by subtracting the allocated bandwidth B from the available bandwidth of all links included in path P^*, for all the time slots contained in the interval $[t^*, t_e^*]$. If t^* or t_e^* were not included in L previously, new elements should be added to W accordingly.

We now describe the SlotSearch function. The pseudo-code can be presented as follows:

[1]Path length refers to the number of hops between source and destination, or, possibly the total length of a weighted path.

[2]Path width is defined as the minimum available link bandwidth among all links in the path.

```
Function SlotSearch($s, d, B, T, t_a, t_b$):
For $t_i$ in $\bar{L}$ do:
    $g(i) \leftarrow$ GradeSolution($i, s, d, T, B$).
$I \leftarrow \arg\max_i g(i)$.
If $g(I) > 1$,
    Return($t_I$).
Else,
    Return (BLOCKED).
```

SlotSearch can be explained by the following steps:

1) For each slot i in $\bar{L}$ calculate a grade $= g(i)$ by calling a function GradeSolution. Specifically, the function GradeSolution(i, s, d, T, B), explained below, is used to give a grade to a route starting at time t_i.
2) Find the maximum grade, call it $g(I)$, i.e. the grade of the route starting at time t_I is maximum.
3) If $g(I) > 1$, it is possible to establish a path starting from some slot in $\bar{L}$. Return the starting time t_I.
4) Else, no path is found for the connection duration, and no connection can be started in $[t_a, t_b]$ between s and d. Therefore, the request will be rejected by returning BLOCKED.

Function GradeSolution grades slots. The grade of a slot may be either a scalar or a vector, in which case comparison is conducted in a lexicographic order. When the grade is a scalar, only slots in which a connection can start have a grade greater than 1. The implementation of GradeSolution depends on the specific optimization criterion. We next consider the case where the goal is to find the *earliest* time slot at which a connection can be set up, using a scalar grade. The pseudo-code is as follows:

```
Function GradeSolution($i, s, d, T, B$):
$j \leftarrow i$
while $t_{j+1} - t_i < T$ do $j \leftarrow j + 1$
$\underline{G} \leftarrow \cap_{k=i}^{j} G_k$.
grade $=$ bfs($\underline{G}, s, d$) $+ \exp(-t_i)$.
return($grade$).
```

The operator $\cap$ stands for intersection between graphs, the result of which is a subgraph that contains only links belonging to all the graphs. GradeSolution can be detailed as follows:

1) For each slot $k \in L$, construct a graph G_k by removing from G all the links with residual bandwidth less than B.
2) Find an intersection of graphs $G_i, \ldots, G_j$ for the smallest j, such that the requested duration is satisfied, that is, $t_{j+1} - t_i \geq T$, and denote it by $\underline{G}$. Thus, each link in $\underline{G}$ has residual bandwidth greater than or equal to B for all the time slots from slot i to slot j.
3) Perform a Breadth First Search (BFS) path discovery from s to d on the graph $\underline{G}$ using function bfs$(\underline{G}, s, d)$.
4) If one or more paths exist, function bfs$(\underline{G}, s, d)$ returns 1.
5) Else, function bfs$(\underline{G}, s, d)$ returns 0.
6) Grade for each slot is defined as $g(i) =$ bfs$(\underline{G}, s, d) + \exp(-t_i)$. Adding the exponential term results in a better score for earlier time slots, when the score is a single real number. The exponential function can be replaced by any positive, strictly decreasing function smaller than 1.

When GradeSolution is implemented as above, then the SlotSearch procedure satisfies the following property:

Theorem 1: SlotSearch always returns the *earliest* time at which a path satisfying the requested bandwidth B and connection length T can be established between nodes s and d.

Proof: We prove by contradiction, let t^* denote the starting time slot returned by SlotSearch. Suppose the intersection between graphs G_k with $k = i, \ldots, j$ for $t_i \in \bar{L}$ and $t_i < t^*$ contains a path between the source and destination. Since $\exp(-t_i) > \exp(-t^*)$, the grade $g(i)$ will be smaller than g^* corresponding to t^* which contradicts our assumption that t is returned by SlotSearch. Also, if the intersection between the relevant graphs, G_k, $k = i, \ldots, j$ contains no path between the source and destination, then necessarily there is no possibility to find a path satisfying the constraints starting at any time in the interval $[t_i, t_{i+1})$. Therefore, we are sure that no path exists starting at a time earlier than t^*. ∎

We note that if $t_b = \infty$, then it is guaranteed that SlotSearch will always find a path (assuming that the requested bandwidth B does not exceed the link capacities). This is because all the links in the network are available in full capacity in the last time slot (slot n) and its length is infinite.

The following theorem states that GCR has polynomial-time complexity. Specifically, denote by r the maximum number of pending requests at any time and C the worst-case computational complexity of the path search, then:

Theorem 2: GCR has a computational complexity of $O(|E|r^2 + C)$.

Proof: Every new job starts with an existing event (or at t_{now}). Therefore, it only adds at most one new future event (at its end, unless it coincides with an existing event). Consequently, the number of future events is bounded by the number of pending (or unfinished) jobs, r.

Every execution of GradeSolution requires finding the intersection of at most r different graphs, each having $|E|$ edges requiring at most $O(|E|r)$ operations, and then performs a BFS search, requiring another $O(|E|)$ operations. GradeSolution is called at most $r+1$ times by SlotSearch, and then PathSearch is called, requiring another $O(C)$ operations, leading to the result in the theorem statement. ∎

As an example, consider the case where the search criterion of PathSearch is the shortest path. Using a Breadth First Search (BFS) procedure, computing the shortest path requires at most $C = |E|$ operations. Thus, the computational complexity of GCR in this case is just $O(|E|r^2)$. In fact, if the criterion is only based on finding the earliest time slot available, and then performing a search for the optimal path, the optimization presented in [21] can lead to a time complexity of $O(|E|r+C)$, by storing for each edge the next time its capacity drops below the required bandwidth.

C. Other grading criteria

The proposed algorithmic framework can accommodate other grading policies. For instance, consider the selection of the shortest-earliest path from s to d. To this end, we denote the length of the shortest path between nodes s and d by $l(s, d)$. If no path is found between s and d, then $l(s, d) = \infty$. We can then define the grade at slot i, as $g(i) = -\alpha l(s, d) + \exp(-t_i)$, where $\alpha > 1$ is a constant used for assigning a higher weight to path length than connection start time. We note that this grade is maximum for the shortest path between s and d and if multiple such paths are available then the earliest one is returned because of the exponential term. In that case, `SlotSearch` will return the slot I such that $I = \arg\max_i g(i)$. If $g(I) = -\infty$, then it will return `BLOCKED`.

D. Trunk reservation

The literature is rich of mathematical theories to handle routing and capacity allocation in circuit-switched networks (sometimes referred to also as *loss networks*) [22, 23]. A major insight from this literature is that greedy (myopic) policies may be detrimental from a network's point of view. Thus, one may wish to prioritize use of "efficient" routes, such as shortest-path routes. *Trunk reservation* is a well-known control mechanism to achieve such prioritization in a distributed fashion [24]. For each link j with capacity $C(j)$, trunk reservation dedicates a certain fraction $T(j)$ of the link capacity to the exclusive of higher priority paths, where $0 \leq T(j) \leq 1$.

Although advance channel reservation and circuit-switched networks differ in several aspects, it is reasonable to expect that greedy policies returning the earliest available path (which may not be the shortest) could lead to inefficiencies. By tuning the threshold parameter $T(j)$, trunk reservation allows to balance between the two extremes of earliest-shortest and shortest-earliest path optimizations.

Trunk reservation can easily be integrated into `GCR`. For each source-destination pair (s, d) we first determine all the shortest-path links (i.e., links belonging to a shortest path between s and d). Then, whenever a connection reservation request between nodes s and d arrives, the available bandwidth on a shortest-path link j during slot i corresponds to the unused bandwidth b_j^i of that link (as in `GCR`). However, if link j is a non shortest-path link, then the available bandwidth on that link is restricted to $\max\{b_j^i - T(j)C(j), 0\}$. Note that a link may be a shortest-path link with respect to a certain source-destination pair but not to another.

This modified procedure, referred to as $\texttt{GCR}_{\texttt{TR}}$, possesses the same properties as `GCR`. Hence, it returns the earliest slot available to establish a connection between nodes s and d *under the constraints* of trunk reservation. Similarly, all the results presented in the next section for path switching extend to the case where trunk reservation is implemented.

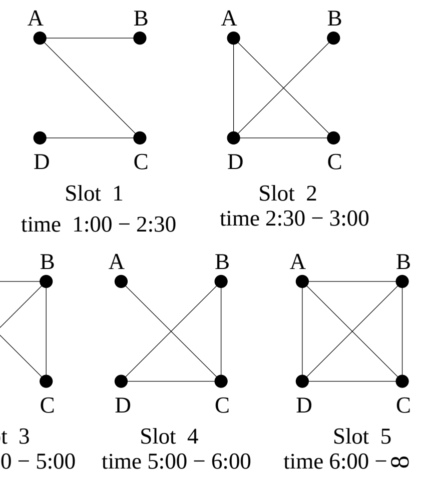


Fig. 2. With path switching, a connection lasting $T = 5$ hours between nodes B and C can be established starting from slot 1. Without path switching, the connection can be established starting from slot 3 only.

IV. Path Switching

A. Motivation

The algorithm described in the previous section returns a single path for the entire connection duration. We observe, however, that it is possible to satisfy a request even if different paths are used at different times. We refer to this approach as *path switching*. By relaxing the constraint of using the same path over all time slots, significant performance improvement in terms of network utilization can be achieved.

Figure 2 illustrates the benefits of path switching for a clique of four nodes. Suppose we are interested in setting up a connection for $T = 5$ hours between nodes B and C. If we do not use path switching, the earliest time slot to establish a path is slot 3, where the same path (e.g., the direct link between nodes B and C) is available during time slots 3, 4 and slot 5. On the other hand, if path switching is allowed, then the connection can be set earlier, namely during slots 1, 2, and 3. In this case, a different path would be used at each of the time slots.

B. Algorithm

In this section, we explain how to integrate path switching into `GCR`. The new framework, called $\texttt{GCR}_{\texttt{switch}}$, has similar structure to `GCR`, so we just describe the main modifications.

Function `SlotSearch` should be replaced with $\texttt{SlotSearch}_{\texttt{switch}}$, which returns the earliest slot at which a connection between a source and a destination can be set up. Note that the returned connection does not necessarily use the same path over its entire duration. The main modification in $\texttt{SlotSearch}_{\texttt{switch}}$ is in calling a function $\texttt{GradeSolution}_{\texttt{switch}}$, which is detailed below.

If $\texttt{SlotSearch}_{\texttt{switch}}$ does not return `BLOCKED`, then $\texttt{GCR}_{\texttt{switch}}$ calls $\texttt{PathSearch}_{\texttt{switch}}$ which finds a set of paths between the source and the destination for the connection duration starting at time t_1^*. Specifically, $\texttt{PathSearch}_{\texttt{switch}}$ returns a vector of tuples $\{(t_1^*, P_1^*), (t_2^*, P_2^*) \ldots, (t_{X+1}^*, P_{X+1}^*)\}$, where X is the total number of path switches, $t_2^*, t_3^*, \ldots, t_{X+1}^*$

are path switch instances within the interval $[t_1^*, t_e^*]$, and P_i^* denotes the selected path starting at time t_i^* for $1 \leq i \leq X+1$. At each time slot, a path is selected according to the desired secondary optimization criteria.

Finally, $\texttt{GCR}_{\texttt{switch}}$ calls the $\texttt{Update}_{\texttt{switch}}$ function that updates L and W after a request is allocated. The list L is updated the same way as done by the function $\texttt{Update}$. Regarding W, for each interval $[t_i^*, t_{i+1}^*]$, $1 \leq i \leq X-1$, $\texttt{Update}_{\texttt{switch}}$ subtracts the allocated bandwidth B from the residual bandwidth of all links included in the path P_i^*.

We now explain $\texttt{GradeSolution}_{\texttt{switch}}$. As before, only slots at which a connection can be initiated are given grades greater than 1 and a negative exponential function is added to assign higher grades to earlier slots. $\texttt{GradeSolution}_{\texttt{switch}}$ can be represented with the following pseudo-code:

```
Function GradeSolution_switch(i, s, d, T, B):
$j \leftarrow i$
while $t_{j+1} - t_i < T$ do $j \leftarrow j+1$
$v \leftarrow \wedge_{k=i}^{j} \texttt{bfs}(G_k, s, d)$.
grade $= (v + \exp(-t_i))$.
return($grade$).
```

The $\wedge$ operator above is a logical AND between outcomes of the bfs functions The following steps explain how this function works:

1) For each slot $k \in L$, construct a graph G_k by removing from G all the links with residual bandwidth less than B.
2) Find a minimum number j such that the requested duration is satisfied, i.e. $t_{j+1} - t_i \geq T$.
3) Perform a Breadth First Search (BFS) path discovery from s to d on each graph G_k using function $\texttt{bfs}(G_k, s, d)$, where $k = i, \ldots, j$.
4) If there is a path (not necessarily the same) between the source and destination in each time slot from slot i to slot j, then the variable v is set to 1.
5) Else, the variable v is set to 0.
6) An exponential term $\exp(-t_i)$ is added to v to assign higher score to earlier slots.

For the above implementation of $\texttt{GCR}_{\texttt{switch}}$, the following property can be proved:

Theorem 3: When path switching is permitted, $\texttt{SlotSearch}_{\texttt{switch}}$ always returns the earliest available time slot that can accommodate a connection between nodes s and d satisfying the requested bandwidth B and connection length T.

Proof: For any starting time t, $\texttt{GradeSolution}_{\texttt{switch}}$ will fail (return a result less than 1) only if there is a time slot within $[t, t+T)$ where no appropriate path exists. Otherwise, it will return a true result, with grade decreasing with the starting time. ∎

The computational complexity of $\texttt{GCR}_{\texttt{switch}}$ as presented is $O(|E|r^2 + Cr)$, but it can be further decreased to $O(|E|r + Cr)$ by keeping in storage the next failure time similar to the non-switching case.

C. Minimum path switching

From a practical perspective, a possible drawback of path switching is the need to perform many path establishments and releases throughout the life of a connection. To address this issue, we introduce a strategy, called *minimum switching*, that guarantees that the number of switches performed during a connection is minimized while returning the highest graded time slot to start the connection.

Our implementation, called $\texttt{GCR}_{\texttt{minimum}}$, uses exactly the same functions as $\texttt{GCR}_{\texttt{switch}}$, except for a new function called $\texttt{PathSearch}_{\texttt{minimum}}$ that replaces $\texttt{PathSearch}_{\texttt{switch}}$. This function intersects the graphs for consecutive time slots (done in step 4b below) and checks whether there continuously exists a path over all these time slots (done at the beginning of step 4). The algorithm selects a path P which is available during the maximum possible number of consecutive time slots (if multiple paths are available, then it selects one according to secondary optimization criteria). At the moment a path used previously becomes unavailable, the connection switches to a new path (step 5).

The $\texttt{PathSearch}_{\texttt{minimum}}$ function uses a list, called $Paths$, to keep track of paths used during the connection. Every time the algorithm decides that the connection must switch paths, the previous path together with its starting time are appended to the list. A detailed description of $\texttt{PathSearch}_{\texttt{minimum}}$ is as follows:

1) Initialize $Paths$ with $null$, $t_e^* \leftarrow t_1^* + T$ and set i as the slot corresponding to the start of the connection, i.e., $t_i = t_1^*$.
2) Set $G' \leftarrow G_i$, where G_i is the graph obtained by removing all links with insufficient residual bandwidth for the request at slot i.
3) If $t_i \geq t_e^*$ append (t, P) to $Paths$ and exit.
4) If $\texttt{bfs}(G', s, d) = 1$,
   a) set P^* to be the highest graded path between s and d in the graph G' (the existence of at least one path in G_i is guaranteed by the success of $\texttt{SlotSearch}_{\texttt{switch}}$).
   b) Set $i \leftarrow i+1$, $G' \leftarrow G' \cap G_i$, and return to step 3.
5) Else ($\texttt{bfs}(G', s, d) = 0$), append new path (t, P) to $Paths$, set $G' \leftarrow G_i$, $t \leftarrow t_i$ and return to step 3.

We illustrate the behavior of $\texttt{GCR}_{\texttt{minimum}}$ for the configuration shown in Fig. 2. As before, let's consider a request arriving at 1:00 pm demanding a connection of $T = 5$ hours between nodes B and C. As in $\texttt{GCR}_{\texttt{switch}}$, the connection start time is t_1^* =1:00 pm (slot 1). The intersection between the residual graphs corresponding to slot 1, G_1, and to slot 2, G_2, does not contain any path between B to C. Therefore, the path $\{A, B, C\}$ is selected during slot 1, and is switched at the beginning of slot 2. Intersecting graphs G_2 and G_3 results into a single path $\{B, D, A, C\}$ during time slots 2 and 3, which the connection can use until its completion. We note here that $\texttt{GCR}_{\texttt{switch}}$ would have switched to the shorter path $\{B, C\}$ during slot 3. This example illustrates a trade-off between optimizing path selection and minimizing the number of path switches.

The next theorem proves that $\texttt{GCR}_{\texttt{minimum}}$ indeed minimizes the number of path switches.

Theorem 4: $\texttt{GCR}_{\texttt{minimum}}$ returns the earliest available time slot to establish a connection. Further, the returned connection makes a minimum number of path switches.

Proof: The fact that $\texttt{GCR}_{\texttt{minimum}}$ returns the earliest available time slot follows immediately from Theorem 3, since it uses the same function $\texttt{SlotSearch}_{\texttt{switch}}$ to find the first slot available.

Next, we prove by contradiction the minimality of path switches. We use the notation $t^*_1, t^*_2, \ldots, t^*_{f-1}, t^*_f$, to denote the switching times of the connections, except for t^*_1 and $t^*_f = t^*_e$ that respectively represent the start and termination time of the connection. Assume that there exists another sequence of times $t'_1, \ldots, t'_g$, where $t'_1 = t^*_1$, $t'_g = t^*_f$, and $g < f$, i.e., fewer path switches. Since $g < f$ at least one of the switches in the primed sequence is conducted later than its equivalent in the returned sequence. So, there must be some first slot, $x > 1$, such that $t^*_x < t'_x$ and $t^*_{x-1} \geq t'_{x-1}$. Since a single path exists between times t'_{x-1} and t'_x, it must also exist between the times t^*_{x-1} and t^*_x (since $t^*_x < t'_x$). However, the function $\texttt{PathSearch}_{\texttt{minimum}}$ should have used this path for as long as it exists, and should have returned time t'_x instead of t^*_x, thus leading to a contradiction. ∎

We conclude this section, by describing another heuristic for decreasing the number of switches, called *limited switching*. According to this heuristic, a connection is allowed to switch to a better path as long as the number of switches does not exceed a certain predefined threshold. To implement this approach, we have developed an algorithm, called $\texttt{GCR}_{\texttt{limitx}}$, that limits the number of switches per connection to at most x. $\texttt{GCR}_{\texttt{limitx}}$ can be considered as a mixture of $\texttt{GCR}$ and $\texttt{GCR}_{\texttt{switch}}$. It operates as follows: it starts with a slot in $\bar{L}$ that contains at least one path between the source and the destination and selects a path according to the desired optimization criterion. In the next slot, it switches path if the current path is no longer available or a better path is found. This procedure continues as long as the number of path switches does not exceed the limit x. After x path switches, $\texttt{GCR}_{\texttt{limitx}}$ sticks to the last path found for the rest of the connection. In the case that no path is available at one of the time slots or if after x switches the connection cannot continue with the current path, then the algorithm starts another search for this connection, starting from the next time slot in the window specified by the user.

V. Simulation and Performance Evaluation

A. Performance Measures

We have developed a simulation tool in C code to evaluate the performance of the proposed algorithms. The main performance metrics of interest are:

1) *Average delay*, which corresponds to the average time elapsing from the point a request is placed until the connection actually starts.
2) *Blocking probability*, which is the probability of rejection in the case users define a finite length time window to set up the connection.
3) *Maximum sustainable load*, which corresponds to the maximum offered load (in terms of requests per unit of time) that the network can sustain. When the offered load exceeds the maximum sustainable load, then the average delay of requests becomes unbounded.

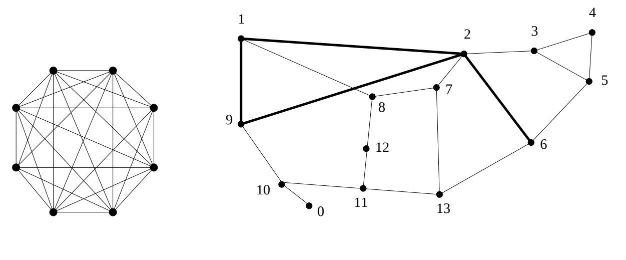
In terms of network performance, algorithms with lower average delay or blocking probability and higher maximum sustainable load are more desirable.

B. Simulation Parameters

The simulator allows evaluating our algorithms under various topological settings and traffic conditions. The main simulation parameters are as follows:

- *Topology:* our simulator supports arbitrary topologies. In most of the simulations, we use the two topologies depicted in Figure 3, namely, a fully connected graph of 8 nodes and a topology that represents a superposition of the DoE UltraScience Net and the National Lambda Rail testbeds [7,25,26]. Each link on these graphs is full-duplex and assumed to have a capacity of 20 Gb/s.
  We also report results obtained for random, directed graphs, each with $|V| = 15$ nodes and edge probability $p = 0.2$, where p represents the probability that a directed edge exists between any given, ordered pair of nodes. The capacity of each link is 20 Gb/s. Disconnected graphs are discarded from the simulation results.
- *Arrival process:* we assume that the aggregated arrival of requests to the network forms a Poisson process. Our simulations are repeated for different arrival rates, also referred to as *network load.*
- *Connection length:* we assume that the requested connection length T, with mean $\bar{T}$, is distributed according to either of the following distributions:
  1) Exponential
  $$Pr(T \geq t) = \exp(-\lambda t),$$
  where $t \geq 0$ and $\bar{T} = 1/\lambda$.
  2) Pareto
  $$Pr(T \geq t) = (\frac{1}{t+\alpha})^{\beta},$$
  where $t \geq 1 - \alpha$, and $\bar{T} = -\alpha + \beta/(\beta - 1)$.

  Without limitation of generality, for both of the above models, the mean connection length is set to one time unit which is defined as one hour in our simulations. For the exponential model, we set $\lambda = 1$, and for the Pareto model, we set $\beta = 2.5$ and $\alpha \simeq 0.66$.
- *Bandwidth:* This parameter corresponds to the requested bandwidth B. We consider two models:
  1) Uniform: the bandwidth B is uniformly distributed among the integers 1 to 10 Gb/s.
  2) 80/20: Whereas 80% of the requests are for 1 Gb/s connections and the remaining 20% are for 10 Gb/s connections. This models the scenarios where most of the users have access to 1 Gb/s links and some have access to 10 Gb/s links.
- *Source:* We again consider two models:



(a) 8 node (b) 14 node LambdaRail

Fig. 3. topologies for simulations.

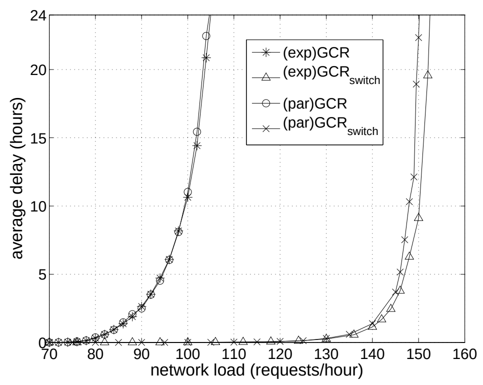


Fig. 4. Performance of GCR with and without path switching for the 8-node clique topology. Distributions of source, destination and requested bandwidth are uniform. Pareto and exponential distributions for connection length are considered.

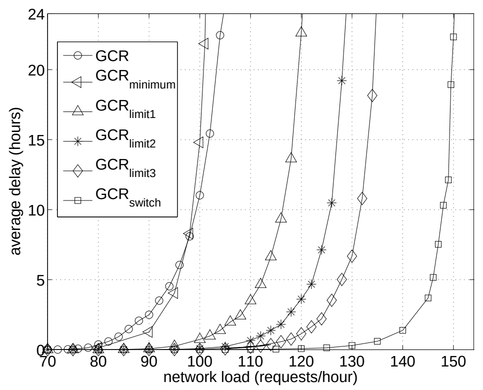


Fig. 5. Performance of GCR with several path switching alternatives for the 8-node clique topology, i.e., GCR, GCR$_{\texttt{minimum}}$, GCR$_{\texttt{limit1}}$, GCR$_{\texttt{limit2}}$, GCR$_{\texttt{limit3}}$, and GCR$_{\texttt{switch}}$. Connection length is exponentially distributed with mean 1 hour. Source, destination and requested bandwidth are distributed uniformly.

1) Uniform: the source s is chosen uniformly at random among all the nodes.
2) Hot-spot: one of the nodes (e.g. a host with a supercomputer) is more likely to be a source node than other nodes in the network. In our simulations, we assume that the hot-spot node has a probability 50% to be selected. Otherwise, one of the other nodes is selected uniformly at random.

- *Destination:* which is denoted by d, and is selected uniformly at random among all the nodes (excluding the source).
- *Blocking window:* Users may specify a time window during which the connection should start. Therefore, if a connection cannot be started during this window, it will be rejected. A window is specified by the time interval $[t_a, t_b]$, where t_a is the window start time which is set to $t_a = t_{now}$ (the arrival time of a request), and t_b is the window end time. We use $t_b = t_a + t_w$, where t_w represents the window length.

All of the simulations are run for a total of at least 10^6 requests for different values of network load. We note that it is difficult to determine the exact value of the maximum sustainable load using simulation. Thus, we define maximum sustainable load as the network load at which the average delay starts exceeding 24 hours. Since, the average delay curve increases very sharply with network load around that value, we conjecture that the real value of the maximum sustainable load is very close.

C. Simulation Results

We present simulation results illustrating the benefits of path switching, path grading and trunk reservation.

1) Path switching: In this section and the following, we assume that $t_w = \infty$, i.e., there is no blocking. Figure 4 depicts the average delay versus network load for GCR (no switch) and GCR$_{\texttt{switch}}$ (unlimited switching) for the 8-node clique topology. In both cases, path selection is based on the earliest-shortest optimization criterion.

It is apparent from the figure that switching improves both the delay and the maximum sustainable load significantly. For instance, for the exponential model the maximum sustainable load exceeds 150 request/hour with path switching, while it is slightly above 100 requests/hour without path switching. Thus path switching leads roughly to a 50% increase in the maximum network utilization achievable.

For the same topology and traffic parameters, Figure 5 shows the performance of various heuristics aimed at limiting the number of path switches. The figure indicates that even if the number of switches is limited to a maximum of three, two, or one per connection, significant performance improvement can be achieved. In the latter case, the maximum sustainable load exceeds 120 requests/hour, about a 20% improvement compared to the case where switching is disabled. On the other hand, the minimum switch heuristic does not perform better than no switching at all. The probable reason is that minimum switching uses non-optimal paths that end up degrading performance (as shown by the example depicted in figure 2).

Figure 6 compares the performance of GCR with and without path switching, for the Lambda Rail topology. The results show that the gain in terms of maximum sustainable load is not as significant as for the 8-node clique topology. The main reason is that the Lambda Rail topology is less dense (i.e., the graph has fewer links). Thus, there are fewer alternative paths between each source and destination that can be used for path switching. That being said, path switching still improves performance in a non-negligible fashion.

We have evaluated the performance of GCR and its variations

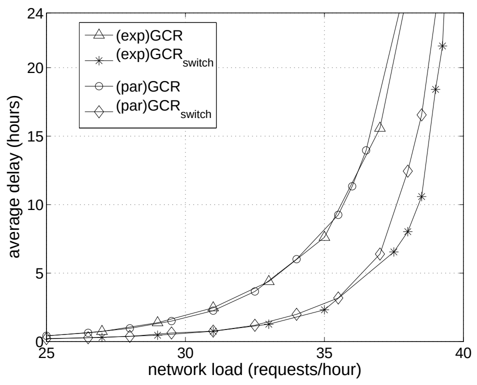


Fig. 6. Performance of GCR with and without path switching for the Lambda Rail topology. Pareto and exponentially distributed connection lengths with mean 1 hour are compared. Source, destination and requested bandwidth are distributed uniformly.

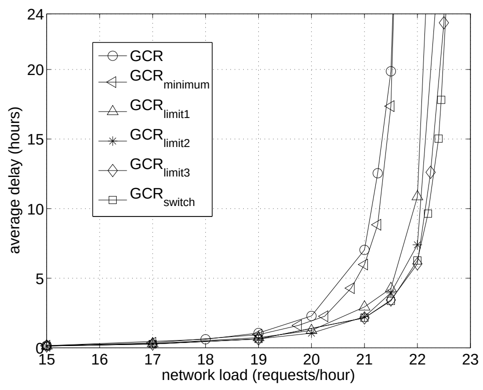


Fig. 8. Performance of GCR, GCR$_{\texttt{minimum}}$, GCR$_{\texttt{limit1}}$, GCR$_{\texttt{limit2}}$, GCR$_{\texttt{limit3}}$, and GCR$_{\texttt{switch}}$ for the Lambda Rail topology, for the hot-spot model (node 6 is the hot spot) and uniform requested bandwidth distribution. Connection length is exponentially distributed.

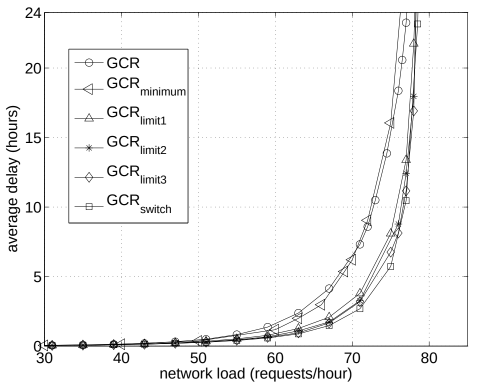


Fig. 7. Performance of GCR, GCR$_{\texttt{minimum}}$, GCR$_{\texttt{limit1}}$, GCR$_{\texttt{limit2}}$, GCR$_{\texttt{limit3}}$, and GCR$_{\texttt{switch}}$ for the Lambda Rail topology, with uniform source and destination and 80/20 requested bandwidth distribution. Average delay decreases in the same order as the algorithms listed here. Connection length is exponentially distributed.

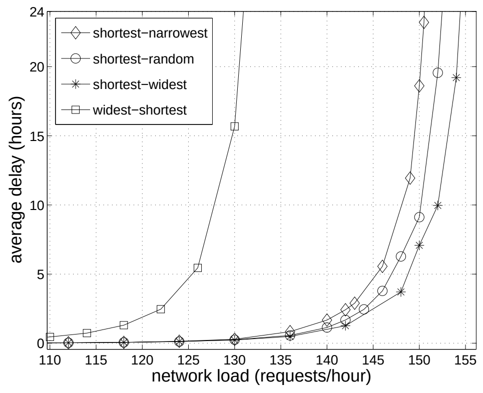


Fig. 9. Performance of GCR$_{\texttt{switch}}$ with various multi-criteria path optimization in the 8-node clique topology, namely: widest-shortest, shortest-random, shortest-widest, and shortest-narrowest path optimizations. Source, destination, and bandwidth are uniformly distributed. Connection length is exponentially distributed.

for other parameter settings as well. For instance, Fig. 7 shows the performance of the different path switching heuristics when the requested bandwidth is distributed according to the 80/20 model for the LambdaRail topology. Figure 8 shows results for the hot-spot model. The results obtained are qualitatively similar to those earlier, wherein GCR$_{\texttt{switch}}$ and GCR$_{\texttt{limitx}}$ always improve performance.

2) Multi-criteria path optimization: The implementation of SlotSearch (and its variants) we used for the simulations returns the earliest available path. In addition, when several earliest paths are available, GCR allows performing optimization of the path selection. Figures 9 and 10 illustrate the importance of such optimization for the 8-node and Lambda Rail topologies respectively. Source, destination, and bandwidth are distributed uniformly. The figures show the performance of GCR$_{\texttt{switch}}$ using four types of path optimization. In the first three, if multiple earliest paths are found, the shortest one is selected. If several shortest paths are available, then the shortest-narrowest heuristic chooses the narrowest path among those, the shortest-random (or simply shortest) chooses one of the paths at random, and shortest-widest chooses the widest[3]. Widest-shortest heuristic first selects the widest path among all the earliest paths available. If multiple earliest-widest paths are found, the shortest among those is selected.

From both figures, it is clear that the first three heuristics significantly outperform the fourth one, that is, selecting one of the shortest among all the earliest paths is a better strategy than selecting one of the widest. The figures also show that a further optimization is not as essential, that is, choosing an earliest-shortest path at random is approximately as good as the other heuristics. Our results are consistent with those reported in the QoS routing literature, see e.g. [27], where shortest-widest routing is shown to outperform other routing strategies. Note however that our setting is different since standard QoS routing algorithms do not support advance reservation of network resources.

3) Blocking probability: We evaluate the blocking probability for the case where users specify a finite length window t_w, which ranges from 0 to 24 hours. We estimate blocking probability using simulation by computing the fraction of

[3]Widest and narrowest refer to the path with the largest or smallest path bandwidth, respectively.

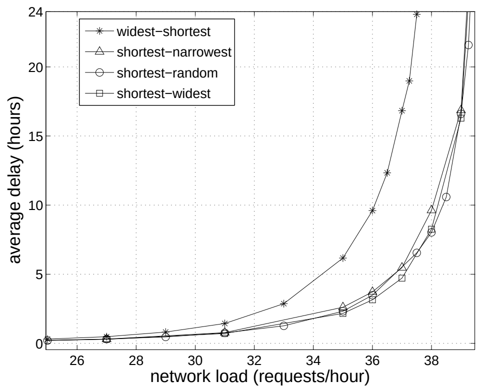


Fig. 10. Performance of $\texttt{GCR}_{\texttt{switch}}$ with various multi-criteria path optimization in the Lambda Rail topology, namely: widest-shortest, shortest-random, shortest-widest, and shortest-narrowest path optimizations. Source, destination, and bandwidth have uniform distribution. Connection length is exponentially distributed.

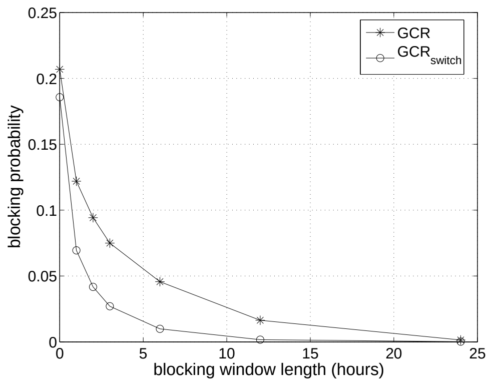


Fig. 11. Performance of $\texttt{GCR}_{\texttt{switch}}$ compared to $\texttt{GCR}$ in terms of blocking probability for the LambdaRail topology. Connection window start time is same as request arrival time. Blocking probability is measured for several different window lengths 0, 1, 2, 3, 6, 12, and 24 hours. Average network load is held fixed at 35 requests/hour, and distribution of source, destination, and bandwidth is uniform. Connection length follows the Pareto distribution.

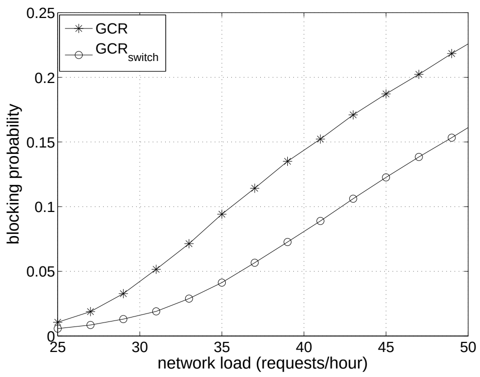


Fig. 12. Performance of $\texttt{GCR}_{\texttt{switch}}$ compared to $\texttt{GCR}$ in terms of blocking probability for the LambdaRail topology with fixed connection window of 2 hours. Distribution of source, destination, and bandwidth is uniform. Connection length follows the Pareto distribution.

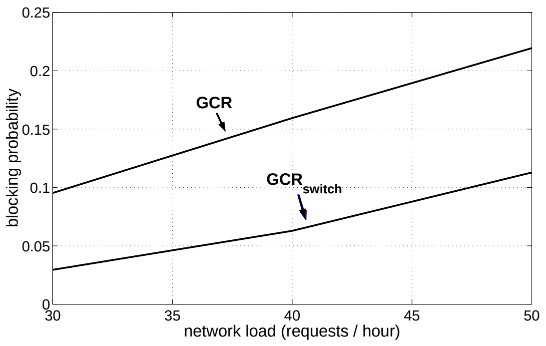


Fig. 13. Performance of $\texttt{GCR}_{\texttt{switch}}$ compared to $\texttt{GCR}$ in terms of blocking probability for 15-node random graph topologies with edge probability $p = 0.2$. Distribution of source, destination, and bandwidth is uniform and fixed connection window t_w= 2 hour. Connection length follows the exponential distribution. Each point on the curves represents an average over 10 random graphs with identical parameters.

rejected requests.

Figure 11 shows results for the Lambda Rail topology at a fixed input load and different choices of blocking window t_w. We observe that switching reduces the blocking probability for all values of t_w. Figure 12 compares the blocking probability of $\texttt{GCR}$ and $\texttt{GCR}_{\texttt{switch}}$ for different loads and a fixed window length $t_w = 2$ hours. Again, switching improves performance at all examined values of network load. Similar results are reported in Fig. 13 for random graphs, showing that our findings extend to other topologies.

4) Trunk reservation: We next examine the benefits of trunk reservation on the performance of $\texttt{GCR}$, $\texttt{GCR}_{\texttt{switch}}$ and $\texttt{GCR}_{\texttt{minimum}}$. We consider the 8-node topology where source, destination, and bandwidth demand are distributed uniformly. We set $T(j) = 0.2$ for each link j, i.e., 4 Gb/s of each link's capacity is reserved for shortest-path routes. In our case, there is only one shortest path route traversing each link, namely the one-hop route between the two nodes connected to each side of the link.

The results show that in this scenario trunk reservation leads to significant increase in the maximum sustainable load, on the order of 30% to 40% for $\texttt{GCR}$ and $\texttt{GCR}_{\texttt{minimum}}$. The performance improvement for $\texttt{GCR}_{\texttt{switch}}$ is also significant but less marked, probably because $\texttt{GCR}_{\texttt{switch}}$ already performs well without trunk reservation.

It should be emphasized that trunk reservation requires proper tuning of the threshold parameter $T(j)$. Consider, for instance, the same 8-node clique topology as before, but with only one source-destination pair generating traffic. In that case, it is easy to see that one must set $T(j) = 0$ for all links j (i.e., no trunk reservation) to achieve best performance.

Figure 15 displays the blocking probabilities of $\texttt{GCR}$ and $\texttt{GCR}_{\texttt{switch}}$ with and without trunk reservation for random graphs, under the same settings as Fig. 13. We set $T(j) = 0.2$ for each link j. Interestingly, the performance of $\texttt{GCR}$ improves when using trunk reservation while that of $\texttt{GCR}_{\texttt{switch}}$ degrades. This result shows that the choice of the threshold parameter should not only be based on the topology and the traffic demand but also on the specific advance channel reservation algorithm being used.

VI. Conclusions and future work

In this work, we developed a novel algorithmic framework for advance reservation based on the principles of path grad-

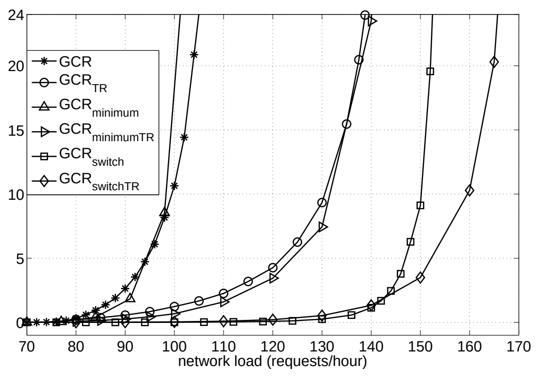


Fig. 14. Average delay performance of GCR, $\text{GCR}_{\texttt{switch}}$, $\text{GCR}_{\texttt{minimum}}$ with and without trunk reservation in the 8-node clique topology. Source, destination, and bandwidth are uniformly distributed. Connection length is exponentially distributed. For trunk reservation, $T(j) = 0.2$ for all links j.

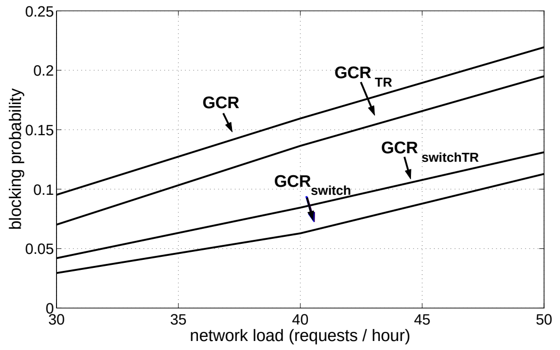


Fig. 15. Comparison of blocking probabilities of GCR, $\text{GCR}_{\texttt{switch}}$, $\text{GCR}_{\texttt{TR}}$ and $\text{GCR}_{\texttt{switchTR}}$ for random graphs under the same settings as Fig. 13. For trunk reservation, $T(j) = 0.2$ for all links j.

ing and path switching, called GCR. We explained how this framework can be used to find and grade paths according to a desired optimization criterion. If the optimization criterion is delay, we proved that GCR returns the earliest time available to start the requested connection. Furthermore, we proved that the complexity of GCR is polynomial in the size of the graph and the number of pending requests.

Thereafter, we presented a generalization of GCR, called $\text{GCR}_{\texttt{switch}}$, which is capable of supporting path switching throughout a connection. We showed that this algorithm retains the same properties as GCR, that is, it returns the earliest available time slot and its complexity is polynomial. Considering practical issues of switching, we designed a variant called $\text{GCR}_{\texttt{minimum}}$ that provably minimizes the number of path switches needed during a connection and another variant, called $\text{GCR}_{\texttt{limitx}}$, that limits the number of path switches to at most x switches per connection.

Our simulation results, run for various topologies and traffic parameters, showed that $\text{GCR}_{\texttt{switch}}$ can significantly improve performance with respect to GCR, in terms of maximum sustainable load, average delay, and blocking probability. We observed that the greatest gain is achieved when the topology is dense, that is, when there are many alternatives for switching paths. The $\text{GCR}_{\texttt{minimum}}$ heuristic, while appealing from a theoretical perspective, did not perform much better than GCR. The likely reason is that paths returned by $\text{GCR}_{\texttt{minimum}}$ are suboptimal (i.e., not necessarily the shortest ones) and, thus, may consume considerable network resources. On the other hand, the $\text{GCR}_{\texttt{limitx}}$ class of algorithms performed better than GCR, even when a single switch between paths is allowed during a connection.

A profound insight of our results is that greedy routing policies returning the earliest available time slot may be inefficient, especially when path switching is disabled or minimized, as in GCR and $\text{GCR}_{\texttt{minimum}}$. As such, we showed that non-greedy policies based on trunk reservation, as $\text{GCR}_{\texttt{TR}}$, or limited path switching, as $\text{GCR}_{\texttt{limitx}}$, may yield significant performance improvement. The main advantage of $\text{GCR}_{\texttt{limitx}}$ over $\text{GCR}_{\texttt{TR}}$ is that it is not as sensitive to parameter tuning and appears to outperform GCR for any topology and traffic pattern, irrespective of the value of x.

Another important and novel aspect of our algorithmic framework is to enable multi-criteria path optimization. Our simulations of $\text{GCR}_{\texttt{switch}}$ showed that a secondary optimization in conjunction with the earliest path selection is beneficial, i.e., choosing earliest-shortest paths is better than other heuristics, but we observed that further optimizations, like earliest-shortest-widest path, is not essential.

We expect the findings of this work to open several new interesting lines of research. For instance, it would be interesting to assess and integrate the cost of path switching into the grading function of GCR and investigate the possible resulting trade-offs. Initial experiments reported for the Lambda Station testbed [28] show that the impact of path switching on throughput is relatively limited. Statistical analysis of the gains achievable with path switching is another area open for further work. Initial results for a simple topology can be found in [29].

Acknowledgments

The authors would like to thank Dr. Nageswara Rao for fruitful discussions.

This research was supported in part by the US Department of Energy under ECPI grant DE-FG02-04ER25605.

References

[1] "GlobalGridForum," http://www.gridforum.org/.
[2] "Large Hadron Collider (LHC) project," http://lhc.web.cern.ch/lhc/.
[3] T. Jepsen, "The Basics of Reliable Distributed Storage Networks," *IT Professional*, vol. **6**, no. 3, pp. 18–24, May/June 2004.
[4] "Network Provisioning and Protocols for DOE Large-Science Applications," in *Provisioning for Large-Scale Science Applications*, N. S. Rao and W. R. Wing, Eds. Springer, New-York, April 2003, Argonne, IL.
[5] N. Rao, W. Wing, S. Carter, and Q. Wu, "UltraScience Net: Network Testbed for Large-Scale Science Applications," *IEEE Communications Magazine*, vol. , 2005.
[6] H. Lee, M. Veeraraghavan, H. Li, and E.K. P. Chong, "Lambda Scheduling Algorithm for File Transfers on High-speed Optical Circuit," in *IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2004)*, April 2004, Chicago, USA.
[7] A. Banerjee, W.-C. Feng, B. Mukherjee, and D. Ghosal, "Routing and Scheduling Large File Transfers over Lambda Grids," in *Proc. of the 3rd International Workshop on Protocols for Fast Long-Distance Networks (PFLDnet'05)*, February 2005, Lyon, France.
[8] "OptIPuter," http://www.optiputer.net/.
[9] R. Cohen, N. Fazlollahi, and D. Starobinski, "Graded Channel Reservation with Path Switching in Ultra High Capacity Networks," in *Proc. ICST/IEEE Gridnets*, October 2006, San Jose, USA.

[10] N. Fazlollahi, R. Cohen, and D. Starobinski, "On the Capacity Limits of Advanced Channel Reservation Architectures," in *IEEE INFOCOM High-Speed Networking Workshop 2007 (HSN 2007)*, May 2007, Anchorage, USA.

[11] C. Guok, D. Robertson, M. Thompson, J. Lee, B. Tierney, and W. Johnston, "Intra and Interdomain Circuit Provisioning Using the OSCARS Reservation System," in *Proc. ICST/IEEE Gridnets*, October 2006, San Jose, USA.

[12] S. Norden and J. Turner, "DRES: Network Resource Management Using Deferred Reservations," in *Proceedings of IEEE GLOBECOM*, November 2001.

[13] A. Schill, S. Kuhn, and F. Breiter, "Resource Reservation in Advance in Heterogeneous Networks with Partial ATM Infrastructures," in *Proceedings of INFOCOM'97*, April 1997, Kobe, Japan.

[14] W. Reinhardt, "Advance Reservation of Network Resources for Multimedia Applications," in *Proc. 2nd Intl. Workshop on Advanced Teleservices and High-Speed Communication Architectures (IWACACA'94)*, September 1994, Heidelberg, Germany.

[15] W. Reinhardt, "Advance Resource Reservation and its Impact on Reservation Protocols," in *Proc. Broadband Islands*, September 1995, Dublin, Ireland.

[16] A. Schill, S. Kuhn, and F. Breiter, "Resource Reservation in Advance in Heterogeneous Networks with Partial ATM Infrastructures," in *Proc. IFIP Broadband*, April 1998, Stuttgart, Germany.

[17] T. Erlebach, "Call admission control for advance reservation requests with alternatives," Tech. Rep. 142, ETH, Zurich, 2002.

[18] A. Banerjee, N. Singhal, J. Zhang, D. Ghosal, C.-N. Chuah, and B. Mukherjee, "A Time-Path Scheduling Problem (TPSP) for Aggregating Large Data Files from Distributed Databases using an Optical Burst-Switched Network," in *Proc. ICC*, 2004, Paris, France.

[19] S. Figueira, N. Kaushik, S. Naiksatam, S. Chiappari, and N. Bhatnagar, "Advance Reservation of Lightpaths in Optical-Network Based Grids," in *Proc.ICST/IEEE Gridnets*, October 2004, San Jose, USA.

[20] N. Kaushik and S. Figueira, "A Dynamically Adaptive Hybrid Algorithm for Scheduling Lightpaths in Lambda-Grids," in *Proc. IEEE/ACM CCGRID/GAN'05-Workshop on Grid and Advanced Networks*, May 2005, Cardiff, USA.

[21] R. Guerin and A. Orda, "Networks With Advance Reservations: The Routing Perspective," in *Proceedings of INFOCOM'00*, March 2000, Tel-Aviv, Israel.

[22] F. Kelly, "Loss Networks," *Annals of Applied Probability*, vol. 1, no. 3, pp. 319–378, 1991.

[23] K. Ross, *Multiservice Loss Models for Broadband Telecommunication Networks*, Springer-Verlag New York, 1995.

[24] F. P. Kelly, "Routing and capacity allocation in networks with trunk reservation," *Mathematics of Operations Research*, vol. 15, pp. 771–793, 1990.

[25] "National LambdaRail Inc.," http://www.nlr.net/.

[26] "UltraScience Net," http://www.csm.ornl.gov/ultranet/topology.html.

[27] Qingming Ma and P. Steenkiste, "On path selection for traffic with bandwidth guarantees," in *ICNP '97*, Washington, DC, USA, 1997, p. 191, IEEE Computer Society.

[28] "Lambda Station path switching experiment," http://www.lambdastation.org/path-switching.html.

[29] R. Cohen, N. Fazlollahi, and D. Starobinski, "Path switching and grading algorithms for advance channel reservation architectures," Tech. Rep. 2008-IR-0070, Center for Information and Systems Engineering, Boston University, 2008.

PLACE
PHOTO
HERE

Reuven Cohen Received his B.Sc. in Physics and Computer Science and his Ph.D. in Physics from Bar-Ilan University, Ramat-Gan, Israel. He was a postdoctoral fellow in the Dept. of Mathematics and Computer Science at the Weizmann Institute, Rehovot, Israel and in the ECE department at Boston University and in the Dept. of Physics at MIT. Since October 2007 he has been at the Department of Mathematics at Bar-Ilan University in Israel, where he is an Assisant Professor. His research interests are random graphs, distributed algorithms and network stability.

PLACE
PHOTO
HERE

Niloofar Fazlollahi received her B.S. in Electrical Engineering (2005) from the Sharif University of Technology, Tehran, Iran. Since 2005, she has been a Ph.D. student at Boston University under the supervision of Professor David Starobinski. Her current research interests are in the modeling and analysis of joint scheduling and routing schemes in ultra high-speed networks.

PLACE
PHOTO
HERE

David Starobinski received his Ph.D. in Electrical Engineering (1999) from the Technion-Israel Institute of Technology. In 1999-2000, he was a post-doctoral researcher in the EECS department at UC Berkeley. In 2007-2008, he was an invited Professor at EPFL, Switzerland. Since September 2000, he has been at Boston University, where he is now an Associate Professor.

Dr. Starobinski received a CAREER award from the U.S. National Science Foundation and an Early Career Principal Investigator (ECPI) award from the U.S. Department of Energy. His research interests are in the modeling and performance evaluation of high-speed, wireless, and sensor networks.