Graded Channel Reservation with Path Switching in Ultra High Capacity Networks

Reuven Cohen, Niloofar Fazlollahi and David Starobinski
Dept. of Electrical and Computer Engineering
Boston University, Boston, MA 02215
Email: {cohenr,nfazl,staro}@bu.edu

***Abstract*— We introduce a new algorithmic framework for advanced channel reservation in ultra high speed networks, called Graded Channel Reservation (GCR). GCR allows users to specify minimum bandwidth and duration requirements for their connections. 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. The computational complexity is polynomial in the size of the graph and the number of pending requests. We introduce a number of variants to GCR, including one that that provides the capability to switch between different paths during 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 saturation throughput by as much as 50%.**

I. Introduction

Network backbones are often presumed to be over-provisioned. However, 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 unique 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 (e.g., 100 GB and higher) 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.

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 *advanced channel reservation* [5–8], 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 advanced channel reservation is to offers hosts and users the ability to reserve in advance *dedicated* channels (paths) to connect their resources.

Advanced 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 OptIPuter [8], are testing possible implementations of such protocols. It should be mentioned that advanced channel reservation is usually assumed to run in a centralized environment. This is a reasonable scenario in small networks, which are currently the most relevant testbeds. The algorithms presented here may also be applied in a distributed manner using link-state mechanisms.

At a first glance, advanced channel reservation is similar to standard circuit switching. One of our major contributions in this paper is to show that advanced channel reservation actually offers a great deal of flexibility that can be exploited to significantly improve performance. First, advanced reservation allows to schedule the starting time of a connection so that some general, multi-criteria optimization objective can be satisfied. Second and more importantly, advanced reservation does not restrict a connection to use the same path over its entire duration. Thus, it provides the possibility to implement *path switching*, that is, switching between different physical paths throughout the life of a connection. With path switching, it is possible to substantially reduce the delay between the time a request is placed until the corresponding connection is set up.

It is worth mentioning here that all the existing advanced reservation protocols and algorithms proposed so far in the literature (cf. Section II) allocate the same path for the entire connection duration in a way similar to circuit switching.

Guided by the above observations, we introduce in this paper a new algorithmic framework for advanced channel reservation called Graded Channel Reservation (GCR). GCR enables grading paths, so that the path with the highest grade is selected. Paths are graded according to the desired optimization objective. Examples of such objectives as will be explained are the earliest, shortest, or widest path satisfying certain bandwidth and duration requirements. The optimization objective can also be multi-criteria. For instance, if the optimization objective is earliest-shortest, then GCR returns the earliest path available and, if many such paths exist, it selects the shortest one among those. 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 $\text{GCR}_{\text{switch}}$, 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 $\text{GCR}_{\text{minimum}}$ that provably performs the minimum number of path switches needed throughout the life of a connection.

In order to evaluate the performance of GCR and its extensions, we have developed an advanced, versatile simulator written in C. This simulator allows configuring the network topology as well as specifying the inter-arrival distribution between requests, the connection length distribution, the distribution of the bandwidth requested by a user, and the source-destination pair for each request. The main performance metrics are the average delay of requests and the saturation throughput which corresponds to the maximum sustainable arrival rate of requests.

Our simulations, run for various topologies and traffic parameters, demonstrate the importance of using multi-criteria objectives. For instance, when the main optimization criterion is finding the earliest path available, then the use of a secondary objective based on the 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 saturation throughput 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.

This paper is organized as follows. In Section II, we review related work on advanced channel reservation. In section III, we introduce the GCR algorithmic framework and prove its main properties. We also describe the novel concept of path switching and show it can be integrated into GCR. In Section IV, we present simulation results evaluating the performance of our algorithms under various network topologies and traffic parameters. We conclude the paper in Section V.

II. Related Work

The topic of advanced resource reservation has received considerable attention in the literature in this field. A great portion of which concentrates on the design of distributed signalling protocols [9–13]. For instance, ref. [10] discusses possible modification to RSVP to support advanced channel reservation.

In addition, several papers have considered the problem of joint routing and scheduling of file transfers. As mentioned in the introduction, none of them implements path switching. Furthermore, the concept of path grading does not seem to have been proposed or studied earlier. We next briefly review some of these papers.

In [14], 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 [15] 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. [16, 17] propose various load balancing approaches to allocate lightpaths.

As in our paper, ref. [18] investigates formal approaches to advanced reservation and provides theoretical analysis of the complexity of path selection. The presented algorithms in [18] 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, and, as mentioned earlier, path switching and multi-criteria optimizations are not considered.

The most relevant work on advanced 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, that, there is no path switching.

III. Model and Algorithms

A. Notation and model

Our model consists of a general network topology denoted by $G(V, E)$ where V is the set of nodes, and E is the set of links. We consider the links in a general network to be directed; therefore, G is in general a directed graph. Each node can generate a request for reservation of a connection channel to another node, called destination, at any time. The requester is called source. Since users don't care about the connection

path, they may only specify some certain restrictions on connection duration and/or start time, and bandwidth. Therefore, we use the following model:

Each request can be expressed by the 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 connection, T is the requested communication duration, $[t_a, t_b]$ specifies a time window during which the user wants the transmission to start. Inserting parameters t_a and t_b is optional, and if omited, they can be interpreted as arrival time of request, denoted by t_{now}, and infinity accordingly.

The reply to this 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 during the time interval $[t, t+T]$. In the case that a request can not be served (which happens if B is greater than link capacities or no path is found starting in $[t_a, t_b]$ when t_b is finite), `False` is returned. Several variations of the model may be presented.

The above notations hold for the simplest case of a fixed connection path, and a more general notation will be introduced in the following for when path switching is allowed.

IV. 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 the 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 ∞ accordingly.

The reply to this 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 during the time interval $[t, t+T]$. In the case that a request can not be served (which happens if B is greater than link capacities or no path is found starting in $[t_a, t_b]$ when t_b is finite), `False` is returned.

B. Basic algorithm

We have developed an algorithmic framework, called *Graded Channel Reservation* (`GCR`) which returns a time slot that can accommodate a connection path according to a certain optimization objective in response to a request given the current and future state of the network. 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 paper, 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.

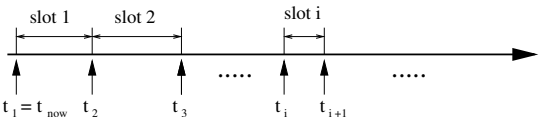


Fig. 1. Segmentation of time axis into slots delineated by events. The state of the network is fixed during each slot.

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 as well. `GCR` uses the following procedure: it divides the time axis into slots delineated by events, as shown in figure 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 $L = \{t_1, \ldots, t_{n+1}\}$ the ordered list of events. Every time a request arrives, we update L by setting $t_1 = t_{now}$ and discard all elements $t_i < t_{now}$. Let $W_i = \{b_1^i, b_2^i, \ldots, b_{|E|}^i\}$ be the vector of available bandwidths of all links at time slot i where $i = 1, \ldots, n$, and b_j^i denote the available bandwidth of link j during slot i. We then define a *bandwidth list* as $W = \{W_1, \ldots, W_n\}$. At the arrival of each request, W is updated the same way as L by dropping the terms W_i for which $t_i < t_{now}$.

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 $\bar{L}$. This notation is illustrated in Figure 2, where for each time slot, only the links in the graph with sufficient residual bandwidth, i.e. residual bandwidth greater than or equal to B, are shown. The figure shows the duration of each slot, e.g., slot 1 lasts from time 0 to time 2, slot 2 lasts from time 2 to time 3, etc. In this case, $L = \{0, 2, 3, 5, 6, 8\}$. Now, suppose $t_a = 1$ and $t_b = 6$, then $\bar{L} = \{1, 2, 3, 5, 6\}$. Finally, we define $\bar{W}$ the same way by removing the vectors W_i with corresponding events t_i such that $t_i < t_a$ or $t_i > t_b$.

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

[1]Path length refers to the number of hops between source and destination.

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

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

The function SlotSearch returns the highest graded time slot in $\bar{L}$, denoted by t, that can accommodate the request (s, d, B, T, t_a, t_b). If no such slot is available it returns $False$ which means that no slot could be found and the request is rejected. In the sequel, we will show 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 in GCR.

If SlotSearch result is not False, then the function PathSearch is called which returns a path P between source s and destination d starting at time t. PathSearch returns a path P according to the selected 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 shortest path search, if multiple shortest paths exist, 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 connection, $t_e = t + T$, is not already included in L, then Update adds 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 in all the links included in path P, for all slots $t_i, \ldots, t_j$, where $t_i = t$, and $t_j = t_e$.

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

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

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 grade_solution.Specifically, the function grade_solution(i, s, d, T, B), explained below, is used to give a grade to a route starting at time t_i.
2) Find slot I which maximizes over all grades $g(i)$.
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 False.

Function grade_solution grades slots. Only slots in which a connection can be started have a grade greater than 1. The implementation of grade_solution depends on the specific optimization criterion. We next consider the case in which the goal is to find the *earliest* time slot in which a connection can be established:

Function grade_solution(i, s, d, T, B)**:**
$j \leftarrow i$
do $j \leftarrow j + 1$ until $t_j - t_i \geq T$
$\underline{G} \leftarrow \cap_{k=i}^{j} G_k$.
return(bfs$(\underline{G}, s, d) + \exp(-t_i)$).

Operator $\cap$ stands for intersection between graphs which means set of all links belonging to both graphs. grade_solution for i can be explained in more detail 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 a better score for earlier time slots.

When grade_solution 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_t with $t = i, \ldots, j$ 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_t, $t = i, \ldots, j$ 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 free 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 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). Therefore, the number of future events is at most to the number of pending (or unfinished) jobs, r. In other words, because according to each arriving job at most one new event is generated in event list L, the number of slots is at most $r + 1$.

Every execution of grade_solution 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. grade_solution 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 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 SlotSearch 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 [18] 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 and utilizing this in grade_solution, updating only when this time arrives. For further details, see [18].

C. Path switching

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

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

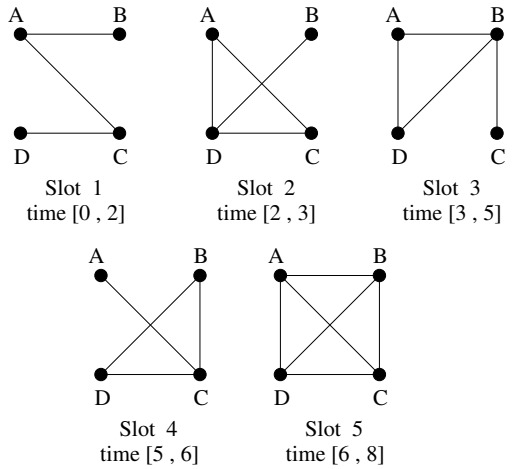


Fig. 2. With path switching, a connection lasting $T = 5$ time units between nodes B and C can be established starting from slot 1. Without path switching, the connection can be established starting from slot 3 only.

We extend GCR to $\text{GCR}_{\text{switch}}$ which has similar structure to GCR, and consists of the following functions:

$\text{SlotSearch}_{\text{switch}}$: This function returns the starting time for a set of slots, all containing at least one path between s and d. The set of slots should satisfy the connection duration. The paths in different slots are not necessarily the same. If no such set of slots are available it returns $False$ which means that the request is rejected:

1) For each slot i in $\bar{L}$ calculate a grade $g(i)$ by calling a function $\text{grade\_solution}_{\text{switch}}$ for i.
2) Find slot I which maximizes over all grades $g(i)$.
3) If $g(I) > 1$, a path is found starting at 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 False.

Function $\text{grade\_solution}_{\text{switch}}$ grades slots. As before, slots at which a connection can be initiated are given grades greater than 1, and the rest smaller than 1. Also as earlier, we have a secondary grading based on a negative exponential function that assigns higher grades to earlier slots. $\text{grade\_solution}_{\text{switch}}$ can be presented by the following pseudo code:

Function $\text{grade\_solution}_{\text{switch}}(i, s, d, T, B)$**:**
$j \leftarrow i$
do $j \leftarrow j + 1$ until $t_j - t_i \geq T$
$v \leftarrow \wedge_{k=i}^{j} \texttt{bfs}(G_k, s, d)$.
$\texttt{return}(v + \exp(-t_i))$.

The $\wedge$ operator above is a logical AND between outcomes of the bfs functions. The following steps sre used to present this function:

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 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.

If $\texttt{SlotSearch}_{\texttt{switch}}$ result does not return `False`, then $\texttt{GCR}_{\texttt{switch}}$ calls $\texttt{PathSearch}_{\texttt{switch}}$ which finds a set of paths between source and destination for the connection duration starting at time t as follows: return a vector of tuples $\{(t_I, P_I), \ldots, (t_{I+X}, P_{I+X})\}$, where $t_I, \ldots, t_{I+X}$ are path switch instances in $[t, t_e]$ ($t_I = t$ is the connection starting time as before), P_i denotes the selected path starting at time t_i for $I \leq i \leq I+X$, and X is the number of path switches. At each time slot, a paths can be selected according to a desired optimization criteria, e.g., shortest path, widest path, etc.

Finally, $\texttt{GCR}_{\texttt{switch}}$ calls the $\texttt{Update}_{\texttt{switch}}$ function that updates L and W after a request is allocated. It updates L the same way as in function `Update`. For updating W, we note here that the connection path from t to t_e is not fixed. Therefore, bandwidth B is subtracted from residual bandwidth of links included in P_i for the duration from t_i to t_{i+1} where t_i and t_{i+1} are members of the set $\{t_I, \ldots, t_{I+X}\}$.

For the above implementation of $\texttt{SlotSearch}_{\texttt{switch}}$, 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 $\texttt{grade\_solution}_{\texttt{switch}}$ 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.

D. 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. It is possible to devise various algorithms to address this issue. One possible approach called, *minimum switching* guarantees that the number of switches performed during a connection is minimized. In this case, the selected path may not be the optimal one in each time slot.

In order to establish a connection with minimum number of switches upon arrival of a request, first we should find a set of time slots that can accommodate this channel. It suffices to find a subsequent set of slots in $\bar{L}$, such that in every slot there exists at least one path from source s to destination d. Therefore, we can use function $\texttt{SlotSearch}_{\texttt{switch}}$ as before which returns the earliest such slot or a `False` if none is found. Here, it's called $\texttt{SlotSearch}_{\texttt{minimum}}$ (notice that it may be desirable to change this function to take the number of switches into the grading. This is also possible in the suggested setting). The $\texttt{PathSearch}_{\texttt{switch}}$ function however, should be replaced by $\texttt{PathSearch}_{\texttt{minimum}}$ which needs some modifications as specified below.

Let us call the algorithm for minimum switches $\texttt{GCR}_{\texttt{minimum}}$. $\texttt{GCR}_{\texttt{minimum}}$ will consist of the following functions:

$\texttt{SlotSearch}_{\texttt{minimum}}$: same as function $\texttt{SlotSearch}_{\texttt{switch}}$.

$\texttt{PathSearch}_{\texttt{minimum}}$: If a slot I, beginning at time t_I, is found by $\texttt{SlotSearch}_{\texttt{minimum}}$ do the following:

1) Initialize $Paths$ to be an empty list, $t_e \leftarrow t_I + T$. Set $i = I$, $t = t_i$.
2) Set $G' \leftarrow G_i$, where G_i is the graph obtained by removing all links with insufficient residual bandwith for the request.
3) If $t_i \geq t_e$ append (t, P) to $Paths$ and exit.
4) If $\texttt{bfs}(G', s, d) = 1$, set P to be the shortest, or otherwise best path between s and d in the graph G' (for any single time slot it is guaranteed such a path exists by the success of $\texttt{SlotSearch}_{\texttt{minimum}}$).
5) Else ($\texttt{bfs}(G', s, d) = 0$), append (t, P) to $Paths$, set $G' \leftarrow G_i$, $t \leftarrow t_i$ and return to step 3.
6) Set $i \leftarrow i + 1$, $G' \leftarrow G' \cap G_i$, and return to step 3.

The algorithm works by intersecting the graphs for consequtive time slots (done in step 6) and checking whether a path continues to exist for all these time slots (done in step 3). When no path remains between the source and destination after some number of intersections, the last found path is used and a new path is searched (step 5).

$\texttt{Update}_{\texttt{minimum}}$: updates L and W same as $\texttt{Update}_{\texttt{switch}}$.

Theorem 4: Function $\texttt{PathSearch}_{\texttt{minimum}}(t)$ finds the minimum number of switchings from time t.

Proof: Function $\texttt{PathSearch}_{\texttt{minimum}}(t)$ where $t = t_i$ for some i works by finding the maximum j such that a single path with sufficient bandwidth exists between t and t_j (i.e., the maximum j such that $\cap_{n=i}^{j-1} G_n$ still contains a path between s and d), then it searches for a new path beginning at t_j and so on. Denote the times of switching by $t_0, t_1, t_2, \ldots, t_f$, where t_0 is the beginning time.

Assume that there is some other sequence of times $t'_0, t'_1, \ldots, t'_g$, where $t'_0 = t_0$, $t'_g = t_f$, and $g < f$, i.e., less switchings. Since $g < f$ at least one of the switchings in the primed sequence is conducted later than its equivalent in the returned sequence. So, here must be some first slot, x, such that $t_x < t'_x$. By minimality $t_{x-1} \geq t'_{x-1}$. Therefore, 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, leading to a contradiction.

Therefore the theorem holds. ∎

Another approach for decreasing the number of switches is *limited switching* which states that we allow a connection to switch to a better path as long as the number of switches does not exceed a certain predefined threshold. This algorithm, called $\texttt{GCR}_{\texttt{limitx}}$, limits the number of switches per connection to at most x. $\texttt{GCR}_{\texttt{limitx}}$ could be considered as a mixture of 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. It should be noted that if some measure of the cost of path switching is defined, it may be combined with the limited switching or minimal switching approaches as part of the grading mechanism in order to give an optimal tradeoff between the number of switches and the job's completion time. For some experiments related to the cost of path switching see [19].

V. Simulation and Performance Evaluation

A. Performance Measures

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

1) *Average delay*, which corresponds to the average time elapsing from the point a request is sent until the connection actually starts.
2) *Saturation throughput*, 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 saturation throughput, then the average delay of requests become unbounded.

In terms of network performance, algorithms with lower average delay and higher saturation throughput are of course more desirable.

B. Simulation Parameters

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

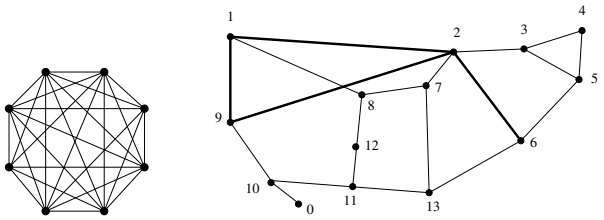
- *Topology:* our simulator supports arbitrary topologies. In our simulations, we have considered two types of topology shown in Figure 3, namely, a fully connected graph of 8 nodes and a topology that represent a superposition of the DoE UltraScience Net and the National Lambda Rail testbeds [7, 20, 21]. Each link on these graphs is full-duplex and assumed to have a capacity of 20 Gb/s.
- *Arrival process:* we assume that the aggregated arrival of requests to the network forms a Poisson process (this can easily be changed, if desired). Our simulations are repeated for different arrival rates, also referred to as *network load*. The *saturation throughput* corresponds to the maximum arrival rate at which the average delay is still bounded.
- *Connection length:* we assume that the requested connection length T is distributed according to an exponential distribution (again this can be changed, if desired). Without limitation of generality, the mean connection length is set to one time unit. In our simulations, a time unit is defined as one hour.
- *Bandwidth:* This parameter corresponds to the requested bandwidth B. We consider two models:
  1) Uniform: the bandwidth B is uniformly distributed between 1 and 10 Gb/s.
  2) 80/20: Whereas 80% of the request 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:
  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:* the destination d is selected uniformly at random among all the nodes (except for the source).

All of the simulations are run for a total of 10^6 requests for each value of network load. The network load is increased from an initial value up to the saturation load for each plot. We note that it is difficult to determine the exact value of the saturation throughput using simulations. Thus, in our simulations, we define saturation throughput as the network load at which the average delay starts exceeding 24 hours. Since, the average delay curve increase very sharply with network load around that value, we conjecture that the actual saturation throughput is very close. Thus, our simulation results always provide a lower bound on the saturation throughput.

C. Simulation results

In this section, we present simulation results illustrating the benefits of path switching and the importance of path optimization.

1) Path switching: Figure 4 depicts the average delay versus network load for GCR (no switch) and $\texttt{GCR}_{\texttt{switch}}$ (unlimited switching) for the 8-node clique topology. In both cases, path selection is based on the earliest-shortest optimization criterion.



(a) 8 node (b) 14 node LambdaRail

Fig. 3. topologies for simulations.

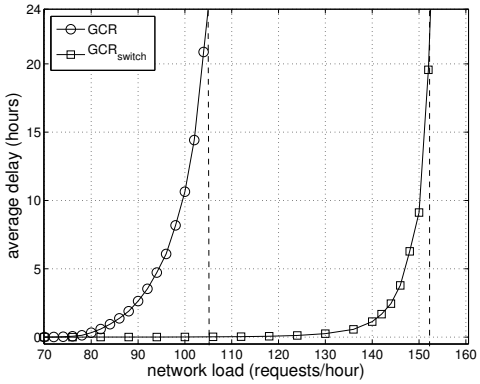


Fig. 4. Performance of GCR with and without path switching for the 8-node clique topology. Distributions of source, destination and requested bandwidth are uniform.

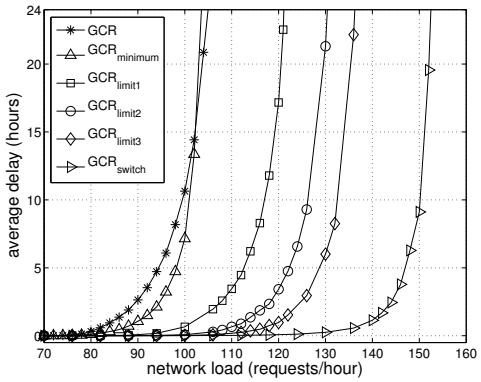


Fig. 5. Performance of GCR with several path switching alternatives for the 8-node clique topology, i.e., GCR, GCR$_{minimum}$, GCR$_{limit1}$, GCR$_{limit2}$, GCR$_{limit3}$, and GCR$_{switch}$. Average delay decreases in the same order the algorithms are listed. Distributions of source, destination and requested bandwidth are uniform.

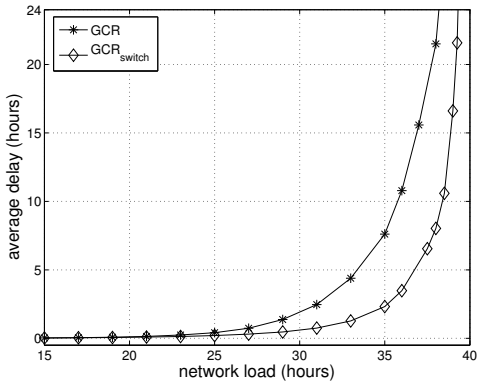


Fig. 6. Performance of GCR with and without path switching for the Lambda Rail topology. Distributions of source, destination and requested bandwidth are uniform.

It is apparent in the figure that switching improves both the delay and the saturation throughput significantly. Specifically, the saturation throughput is slightly more than 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 indicate that even if the number of switches is limited to a maximum of 3, 2, or even 1 per connection, significant performance improvement can be achieved. In the latter case, the saturation throughput 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.

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 saturation throughput 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 alternate paths between each source and destination that can be used for path switching. That being said, path switching is still very helpful in reducing the average delay of requests.

We have compared the performance of GCR with different switching options for other distributions of requested bandwidth and destination as well. Figure 7 shows the performance of the different path switching heuristics with uniform source and destination and requested bandwidth distributed according to the 80/20 model. Figure 8 shows results for the hot-spot model and uniformly requested bandwidth. The results obtained are qualitatively similar to those earlier, wherein GCR and GCR$_{limit}$ always improve performance.

2) Multi-criteria path optimization:: As mentioned in Section II, the variation of SlotSearch we used for the simulations always returns the earliest available path. In addition, when several earliest paths are available, it allows performing optimization of the path selection. Figures 9 and 10 illustrates the importance of such optimization for the 8-node and Lambda Rail topologies respectively. Source, destination, and bandwidth have uniform distribution. The figures show the performance of GCR$_{switch}$ using four types of path optimiza-

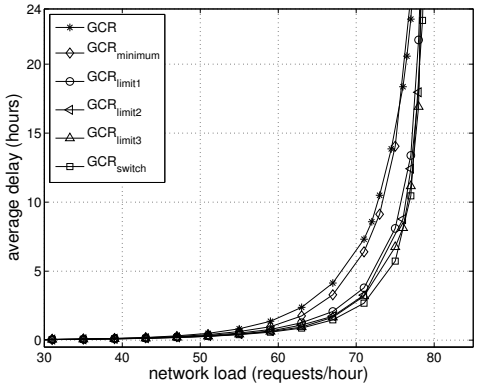


Fig. 7. Performance of GCR, $\texttt{GCR}_{\texttt{minimum}}$, $\texttt{GCR}_{\texttt{limit1}}$, $\texttt{GCR}_{\texttt{limit2}}$, $\texttt{GCR}_{\texttt{limit3}}$, and $\texttt{GCR}_{\texttt{switch}}$ for the Lambda Rail topology, with uniform source and destination and 80/20 requested bandwidth distribution.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

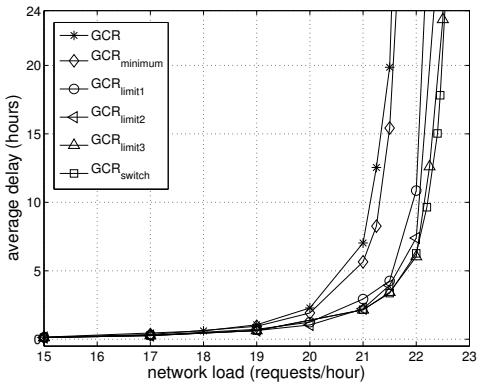


Fig. 8. Performance of GCR, $\texttt{GCR}_{\texttt{minimum}}$, $\texttt{GCR}_{\texttt{limit1}}$, $\texttt{GCR}_{\texttt{limit2}}$, $\texttt{GCR}_{\texttt{limit3}}$, and $\texttt{GCR}_{\texttt{switch}}$ for the Lambda Rail topology, for the hot-spot model (node 6 is the hot spot) and uniform requested bandwidth distribution. Average delay decreases in the same order the algorithms are listed here.

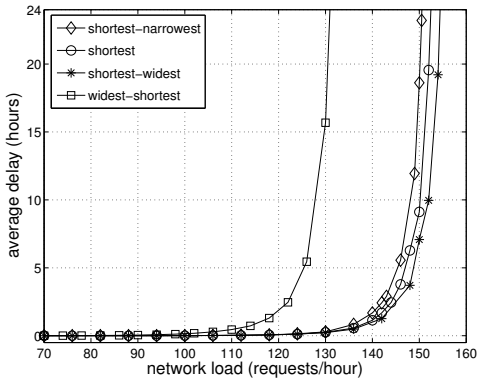


Fig. 9. Performance of $\texttt{GCR}_{\texttt{switch}}$ with various multi-criteria path optimization in the 8-node clique topology, namely: widest-shortest, shortest, shortest-widest, and shortest-narrowest path optimizations. Source, destination, and bandwidth have uniform distribution.

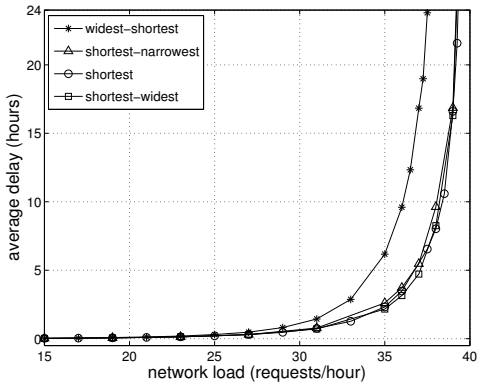


Fig. 10. Performance of $\texttt{GCR}_{\texttt{switch}}$ with various multi-criteria path optimization in the Lambda Rail topology, namely: widest-shortest, shortest, shortest-widest, and shortest-narrowest path optimizations. Source, destination, and bandwidth have uniform distribution.

tions. In the first three, if multiple earliest paths are found, the shortest one is selected. If several shortest paths are available, then shortest-narrowest heuristic chooses the narrowest path among those, the shortest-random (or 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.

VI. Conclusions and Future Work

In this work, we have introduced an algorithmic framework for advanced channel reservation, called GCR. We have 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 have proved that GCR returns the earliest slot time available to start the requested connection. We have proved that the complexity of this algorithm is polynomial in the size of the graph and the number of pending requests.

We have also presented an important extension to GCR, called $\texttt{GCR}_{\texttt{switch}}$, which is capable of returning a connection that switches between different paths. We have shown 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 have designed a variant called $\texttt{GCR}_{\texttt{minimum}}$ that provably minimizes the number of switches needed during a connection and another variant, called $\texttt{GCR}_{\texttt{limitx}}$, that limits the number of path switches to at most x switches per connection.

Our simulation results, run for various topologies and traffic parameters, show that that $\texttt{GCR}_{\texttt{switch}}$ can significantly improve performance, compared to GCR, in terms of saturation throughput and average delay. Unsurprisingly, the biggest gain

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

is achieved when the topology is dense, that is, when there are many alternatives for switching paths. The $\texttt{GCR}_{\texttt{minimum}}$ heuristic, while appealing from a theoretical perspective, did not perform much better than GCR. We conjecture that paths returned by $\texttt{GCR}_{\texttt{minimum}}$ are suboptimal (i.e., not necessarily the shortest ones) and thus may consume considerable network resources. On the other hand, the $\texttt{GCR}_{\texttt{limitx}}$ class of algorithms performed better than GCR, even when only a single switch between paths is allowed during a connection.

Another important and novel aspect of our algorithmic framework is to enable multi-criteria path optimization. Our simulations of $\texttt{GCR}_{\texttt{switch}}$ show 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 observe that further optimizations, like earliest-shortest-widest path, is not essential.

We conclude by noting that in all our simulation the window of request time was unlimited. In some cases, however, a user may want to specify a certain window of time $[t_a, t_b]$ to set-up a connection. In such a case, a request could be blocked if no available path is found during that window. In future work, it would therefore be interesting to evaluate the performance of our various algorithms with respect to the blocking probability metric. We conjecture that path switching and multi-criteria optimization would improve performance in this case as well.

REFERENCES

[1] "GlobalGridForum," http://www.gridforum.org/.

[2] "Large Hadron Collider (LHC) project," http://www.optiputer.net/.

[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.S.V. Rao and W.R. Wing and S.M. Carter and Q. Wu, "UltraScience Net: Network Testbed for Large-Scale Science Applications," *IEEE Communications Magazine*, vol. , 2005.

[6] H. Lee and M. Veeraraghavan and 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 et. al., "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] Norden, S. and Turner, J., "DRES: Network Resource Management Using Deferred Reservations," in *Proceedings of IEEE GLOBECOM*, November 2001.

[10] A. Schill and 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.

[11] 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.

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

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

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

[15] A. Banerjee et. al., "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.

[16] S. Figueira, N. Kaushik, et. all, "Advance Reservation of Lightpaths in Optical-Network Based Grids," in *Proc.ICST/IEEE Gridnets*, October 2004, San Jose, USA.

[17] N. Kaushik, 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.

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

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

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

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