Competitive Advance Reservation with Bounded Path Dispersion

Reuven Cohen[1,2,3], Niloofar Fazlollahi[1] and David Starobinski[1]
[1]Dept. of Electrical and Computer Engineering
Boston University, Boston, MA 02215
[2]Dept. of Physics, MIT, Cambridge, MA 02139
[3]Dept. of Mathematics, Bar-Ilan University, Ramat-Gan 52900, Israel
Email: reuven@mit.edu,{nfazl,staro}@bu.edu

***Abstract*—Advance channel reservation is emerging as an important feature of ultra high-speed networks requiring the transfer of large files. In this paper, we present two new delay-competitive algorithms for advance reservation, called `BatchAll` and `BatchLim`. These algorithms are guaranteed to achieve optimal throughput performance, based on multi-commodity flow arguments. Unlike `BatchAll`, the `BatchLim` algorithm returns the completion time of a connection immediately as a request is placed, but at the expense of a slightly looser competitive ratio than that of `BatchAll`. We propose a simple approach that limits the number of parallel paths used by the algorithms while provably bounding the maximum reduction factor in the transmission throughput. We show that, although the number of different paths can be exponentially large, the actual number of paths needed to approximate the flow is quite small and proportional to the number of edges in the network. According to our simulations for a number of topologies, three to five parallel paths are sufficient to achieve close to optimal performance.**

I. Introduction

Several Grid and data backup applications require the transfer of extremely large datasets on the orders of terabytes and more. To support such applications, significant efforts have recently been devoted to develop a protocol stack based on the concept of *advance reservation* [1, 2]. The most important property of advance reservation is to offer hosts and users the ability to reserve in advance *dedicated* paths to connect their resources. The importance of advance reservation in supporting dedicated connections has been made evident by the growing number of testbeds related to this technology, such as UltraScience Net [1], On-demand Secure Circuits and Advanced Reservation Systems (OSCARS) [3], and Dynamic Resource Allocation via GMPLS Optical Networks (DRAGON) [4].

Several protocols and algorithms have been proposed in the literature to support advance reservation, see, e.g., [1, 2, 5] and references therein. However, to the authors' knowledge, none of them guarantees the same throughput performance as an optimal off-line algorithm. Instead, most are based on greedy approaches, whereas each request is allocated a path guaranteeing the earliest completion time at the time the request is placed.

In this paper, we present competitive advance reservation algorithms that provably achieve the maximum throughput for any network and request profile. The first algorithm, called `BatchAll`, provides a competitive ratio on the maximum delay experienced by each request in an augmented resource network with respect to the optimal off-line algorithm in the original network. `BatchAll` groups requests into batches. In each batch, paths are efficiently assigned based on a maximum concurrent flow optimization.

The `BatchAll` algorithm does not return the connection completion time to the user at the time a request is placed, but only when the connection actually starts. Our second algorithm, called `BatchLim`, returns the completion time immediately as a request is placed, but at the expense of a slightly looser competitive ratio. This algorithm operates by limiting the length of each batch.

The presented competitive algorithms are based upon multicommodity flow algorithms, and therefore can be performed efficiently in polynomial time in the size of the network for each request.

Our model assumes that the network infrastructure supports path dispersion, i.e., multiple paths in parallel can be used to route data. Obviously, a too large path dispersion may be undesirable, as it may entail fragmenting a file into a large number of segments and reassembling them at the destination. To address this issue, we present a simple approach, based on the max-flow min-cut theorem, that limits the number of parallel paths while bounding the maximum reduction factor in the transmission throughput. We, then, propose two algorithms `BatchAllDisp` and `BatchLimDisp`, based upon `BatchAll` and `BatchLim` respectively. These algorithms perform similarly to the original algorithms, in terms of the batching process. However, after filling each batch, the algorithms will limit the dispersion of each flow. Although these algorithms are not throughput-optimal anymore, they are still throughput-competitive.

We provide simulation results illustrating the performance of our protocols in terms of the average delay. The simulations show that the proposed competitive algorithms approach a capacity bound derived in [6]. With respect to path dispersion, we show that excellent performance can be achieved with as few as five or so parallel paths per connection.

This paper is organized as follows. In Section II, we introduce our model and notation used throughout the paper. In Section III, we describe the `BatchAll` and `BatchLim`

algorithms and derive results on their competitive ratios (a previous version of `BatchAll` was presented in [6]). Our method for bounding path dispersion is described and analyzed in Section IV. In Section V, simulation results evaluating the performance of the algorithms for different network topologies and traffic parameters are presented. We conclude the paper in Section VI. Due to space constraints, some of the proofs are deferred to the on-line technical report [7].

II. Network Model

We present the network model that will be used throughout the paper. The model consists of a general network topology, represented by a graph $G(V, E)$, where V is the set of nodes and E is the set of links connecting the nodes. The graph G can be directed or undirected. The capacity of each link $e \in E$ is $C(e)$. A connection request, also referred to as job, contains the tuple (s, d, f), where $s \in V$ is the source node, $d \in V - \{s\}$ is the destination node, and f is the file size.

Accordingly, an advance reservation algorithm computes a starting time at which the connection can be initiated, a set of paths used for the connection, and an amount of bandwidth allocated to each path. Our model supports path dispersion, i.e., multiple paths in parallel can be used to route data.

In subsequent sections, we will make frequent use of multi-commodity functions. The multicommodity flow problem is a linear planning problem returning `true` or `false` based upon the feasibility of transmitting concurrent flows from all pairs of sources and destination during a given time duration, such that the total flow through each link does not exceed its capacity. It is solved by a function $\texttt{multicomm}(G, L, T)$, where L is a list of jobs, each containing a source, a destination, and a file size, and T is the time duration.

The maximum concurrent flow is calculated by the function $\texttt{maxflow}(G, L)$. It returns $T_{\min}$, the minimum value of T such that $\texttt{multicomm}(G, L, T)$ returns `true`. Both the multicommodity and maximum concurrent flow problems are known to be computable in polynomial time [8, 9].

III. Competitive Algorithms

In this section, we present a new family of on-line, polynomial-time algorithms that rely on the idea of "batching" requests. Thus, instead of immediately reserving a path for each incoming request as in a greedy algorithm, we accumulate several arrivals in a batch and assign a more efficient set of flow paths to the whole batch. The proposed algorithms guarantee that the maximum delay experienced by each request in a network with augmented resources is within a finite multiplicative factor of the value achieved with an optimal off-line algorithm in the original network. Thus, these algorithms reach the maximum throughput achievable by any algorithm.

A. The `BatchAll` Competitive Algorithm

In case no deterministic knowledge on the future requests is given, one would like to give some bounds on the performance of the algorithm compared to the performance of a "perfect" off-line algorithm (i.e., with full knowledge of the future and unlimited computational resources). We present here an algorithm, called `BatchAll` (since it batches together all pending requests), giving bounds on this measure.

The algorithm can be described as follows (we assume that there is initially no pending request). We denote by clk the value of a variable maintaining the clock (initially set to 0):

1) For a request $l = \{s, d, f\}$ arriving when $clk = t$, give an immediate connection starting time and a connection ending time of $t_c = t + \texttt{maxflow}(G, l)$.
2) Set $L \leftarrow$ null.
3) While $clk < t_c$,
   - If another request $l' = \{s', d', f'\}$ arrives:
     - Set $L \leftarrow L \cup l'$ (i.e. add it to the waiting batch)
     - Mark t_c as its connection starting time
4) At time $clk = t_c$, calculate $t' = \texttt{maxflow}(G, L)$.
   - If $t' = 0$ (i.e., there is no pending request) go back to step 1.
   - Else assign a connection ending time $t_c = t_c + t'$ to all requests in the batch L and go back to step 2.

We note that upon the arrival of a request, the `BatchAll` algorithm may return only the starting time of the connection. The allocated paths and completion time are computed only when the connection starts (in the next section, we present another competitive algorithm with a slightly looser competitive ratio but which always returns the completion time at the arrival time of a request).

We next compare the performance of an optimal off-line algorithm in the network with that of the `BatchAll` algorithm in an augmented network. The augmented network is similar to the original network, other than that it has a capacity of $(1 + \varepsilon)C(e)$ at any link, e, that has capacity $C(e)$ in the original network. This implies that the performance of the competitive algorithm is comparable if one allows a factor ε extra capacity in the links, or, alternatively, one may say that the performance of the algorithm is comparable to the optimal off-line algorithm in some lower capacity network, allowing the maximum rate of only $(1 + \varepsilon)^{-1}C(e)$ for each link.

Theorem 3.1 ([6]): If we augment the resources of a network such that every edge has bandwidth $(1 + \varepsilon)$ times the original bandwidth (for all $\varepsilon > 0$), then for requests arriving up to any time t^* in such a network, the maximum waiting time from request arrival to the end of the transmission using `BatchAll`, is no more than $2/\varepsilon$ times the maximum waiting time for arrivals up to t^* using the optimal algorithm in the original network.

Corollary 3.2: The saturation throughput of `BatchAll` is optimal because for any arbitrarily small $\epsilon > 0$, the delay of a request is guaranteed to be at most a finite multiplicative factor larger than the maximum delay of the optimal algorithm in the reduced resources network.

B. The `BatchLim` Competitive Algorithm

The following algorithm, called `BatchLim`, operates in a similar setup to `BatchAll`. However, it gives a guarantee on the finishing time of a job when it is submitted.

The algorithm is based on creating increasing size batches in case of high loads. When the rate of request arrival is high, each batch created will be approximately twice as long as the previous one, and thus larger batches, giving a good approximation of the total flow will be created. When the load is low only a single batch of pending requests exists at most times, and its size will remain approximately constant or even decrease for decreasing load.

The algorithm maintains a list of times, t_i, $i = 1, \ldots, n$, where for every interval, $[t_i, t_{i+1}]$, a batch of jobs denoted by batch i is assigned. When a new request between a source s and a destination d arrives at time t, then an attempt is first made to add it to one of the existing intervals by using a multicommodity flow calculation. If the attempt fails, a new batch is created and $t_{n+1} = \max(t_n + (t_n - t), t_n + M)$, where $M = f/\texttt{maxflow}(G, s, d)$ is the minimum time for the job completion, is added to the time list. Thus, the job is assigned to the interval $[t_n, t_{n+1}]$ with $n \geq 2$.

We next provide a detailed description of how the algorithm handles a new request arriving at time t. We use the tuple $l = \{s, d, f\}$ to denote this job and the list L_i to denote the set of jobs already assigned to batch i.

1) Initialization: set $i \leftarrow 1$.
2) While $i \leq n - 1$
   - If $\texttt{multicomm}(G, L_i \cup l, t_{i+1} - t_i) = \texttt{true}$ then return i and exit (check if request can be accommodated during batch i).
   - $i \leftarrow i + 1$.
3) Set $M = \texttt{maxflow}(G, l)$.
4) If $M > t_n - t$
   then $t_{n+1} \leftarrow t_n + M$;
   else $t_{n+1} \leftarrow t_n + (t_n - t)$.
5) Create a new batch consisting of job l and assign it to interval $[t_n, t_{n+1}]$.
6) Set $n \leftarrow n + 1$.
7) Return n.

Fig. 1 illustrates runs of the `BatchAll` and `BatchLim` algorithms for the same set of requests.

The next theorem provides a competitive ratio on the maximum waiting time in `BatchLim`. This theorem implies that `BatchLim` is also throughput-optimal.

Theorem 3.3: For every $\varepsilon > 0$ and every time t^*, the maximum waiting time from the request time to the end of a job for any request arriving up to time t^* and for a network with augmented resources, is no more than $4/\varepsilon$ times the maximum waiting time for arrivals up to t^* using the optimal algorithm in the original network.

IV. Bounding Path Dispersion

The algorithms presented in the previous sections do not limit the *path dispersion*, that is, the number of paths simultaneously used by a connection. In practice, it is desirable to minimize the number of such paths due to the cost of setting up many paths and the need to split a connection into many low capacity channels. The following suggests a method of achieving this goal.

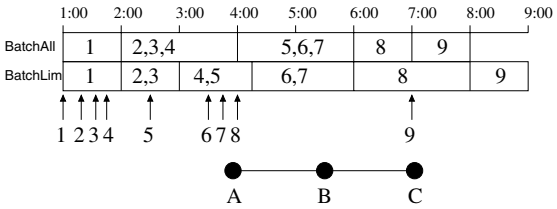


Fig. 1. Illustration of the batching process by the algorithms `BatchAll` and `BatchLim` over a simple network topology consisting of three nodes A,B and C. The odd numbered jobs request one hour of maximum bandwidth traffic between nodes A and B and the even numbered jobs request one hour maximum bandwidth traffic between B and C. In `BatchAll`, a new batch is created for all requests arriving during the running of a previous batch. In `BatchLim`, for each new request an attempt is made to add it to one of the existing windows, and if it fails, a new window is appended at the end.

Lemma 4.1: In every flow of size F between two nodes on a directed graph $G(V, E)$ there exists a path of capacity at least $F/|E|$ between these nodes.

Proof: Remove all edges of capacity (strictly) less than $F/|E|$ from the graph. The total capacity of these edges is smaller than $F/|E| \times |E| = F$. By the Max-Flow–Min-Cut theorem, the maximum flow equals the minimum cut in the network. Therefore, since the total flow is F there must remain at least one path between the nodes after the removal. All edges in this path have capacity of at least $F/|E|$. Therefore, the path capacity is at least $F/|E|$. ■

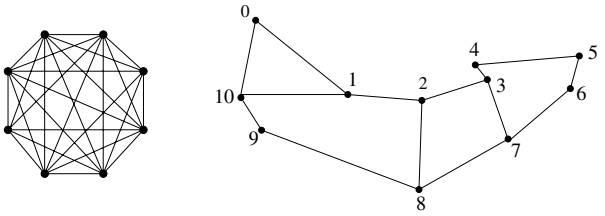
The following theorem establishes the maximum number of paths needed to achieve a throughput with a constant factor of that achieved by the original flow.

Theorem 4.2: For every flow of size F between two nodes on a directed graph $G(V, E)$ and for every $\alpha > 0$ there exists a set of at most $\lceil \alpha |E| \rceil$ paths achieving a flow of at least $(1 - e^{-\alpha})F$.

Proof: Apply Lemma 4.1, $\lceil \alpha |E| \rceil$ times. Each time an appropriate path is found, its flow is removed from all of its links. For each such path the remaining flow is multiplied by at most $1 - \frac{1}{|E|} \leq \exp\left(\frac{1}{|E|}\right)$. ■

Theorem 4.2 provides both an algorithm for reducing the number of paths and a bound on the throughput loss. To approximate the flow by a limited number of paths remove all edges with capacity less than $F/|E|$ and find a remaining path. This process can be repeated to obtain improving approximations of the flow. The maximum number of paths needed to achieve an approximation of the flow to within a constant factor is linear in $|E|$, while the number of possible paths may be exponential in $|E|$.

Using the above approximation in conjunction with the competitive algorithms one can devise two new algorithms `BatchAllDisp` and `BatchLimDisp`, based upon `BatchAll` and `BatchLim` respectively. These algorithms perform similarly to `BatchAll` and `BatchLim`, in terms of the batching process. However, after filling each batch, the algorithms will limit the dispersion of each flow, and approximate the flow. To achieve a partial flow, each time we select the widest path from remaining edges (where the weights are determined by the link utilization in the solution of the multicommodity flow)



(a) 8 node (b) 11 node LambdaRail

Fig. 2. Simulation topologies.

and reserve the total path bandwidth. We repeat this until either the desired number of paths is reached or the entire flow is routed.

V. Simulations

In this section, we present simulation results illustrating the performance of the algorithms described in this paper. We evaluate the average delay of each algorithm, where delay is defined as the time elapsing from the point a request is placed until the corresponding connection is completed. The main points of interest are as follows: (i) how do the competitive algorithms fare with respect to each other and with respect to a capacity bound [6]? (ii) what value of path dispersion is needed to ensure good performance?

A. Simulation Set-Up

We have developed our own simulator in C++. The simulator uses the COIN-OR Linear Program Solver (CLP) library [10] to solve multi-commodity optimization problems and 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 this paper, we consider the two topologies depicted in Figure 2. One is a fully connected graph (clique) of eight nodes and the other is an 11-node topology, similar to the National LambdaRail testbed [11]. 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). The mean rate of arrivals is adjustable. Our delay measurements are carried out at different arrival rates, referred to as *network load*, in units of requests per hour.
- *File size distribution:*
  We consider two models for the file size distribution:
  1) Pareto:
  $$F(x) = 1 - \left(\frac{x_m}{x-\gamma}\right)^{\beta}, \text{ where } x \geq x_m + \gamma.$$
  In the simulations, we set $\beta = 2.5$, $x_m = 1.48$ TB (terabyte) and $\gamma = 6.25 * 10^{-3}$ TB, implying that the mean file size is 2.475 TB.

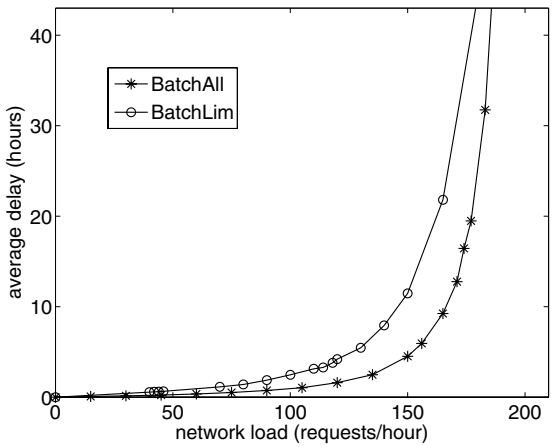


Fig. 3. Performance comparison of algorithms `BatchAll` and `BatchLim` for the 8-node clique and exponential file size distribution.

  2) Exponential:
  $$F(x) = \exp(-\lambda x), \text{ where } x \geq 0.$$
  In the simulations, $1/\lambda = 2.475$ TB.
- *Source and Destination*: for each request, the source and destination are selected uniformly at random, except that they must be different nodes.

All the simulations are run for a total of at least 10^6 requests.

B. Results

We first present simulation results for the clique topology. Figure 3 compares the performance of the `BatchAll` and `BatchLim` algorithms. The file size distribution is exponential. The figure shows that `BatchLim` is less efficient than `BatchAll` in terms of average delay, especially at low load. This result is somewhat expected given that `BatchLim` uses a less efficient batching process and its delay ratio is looser. However, since `BatchLim` is throughput-optimal, its performance approaches that of `BatchAll` at high load.

Figure 4 evaluates and compares the performance of the `BatchAll` and `BatchAllDisp` algorithms. For `BatchAllDisp`, results are presented for the cases where the path dispersion bound is set to either one, three, or five paths per connection. The file size distribution is exponential. The figure shows also a fluid bound on capacity derived in [6]. Its value represents an upper bound on the maximum network load for which the average delay of requests is still bounded.

The figure shows that `BatchAll` approaches the capacity bound at a reasonable delay value and that a path dispersion of at most five per connection (corresponding to $\alpha = 0.089$) is sufficient for `BatchAllDisp` to achieve performance close to `BatchAll`. It is worth mentioning that, in this topology, there exist 1957 possible paths between any two nodes. Thus, with five paths, `BatchAllDisp` uses only 0.25% of the total paths possible. The figure also demonstrates the importance of multi-path routing: the performance achieved using a single path per connections is far worse.

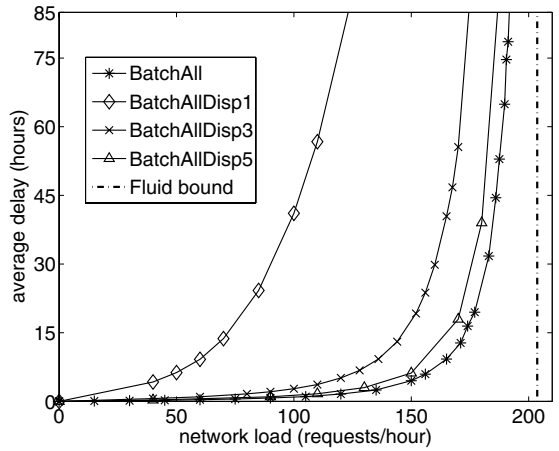


Fig. 4. Performance evaluation of algorithms `BatchAll` and `BatchAllDisp` with bounds 1, 3 or 5 paths per connection for the 8-node clique and exponential file size distribution. A fluid bound on capacity is also depicted.

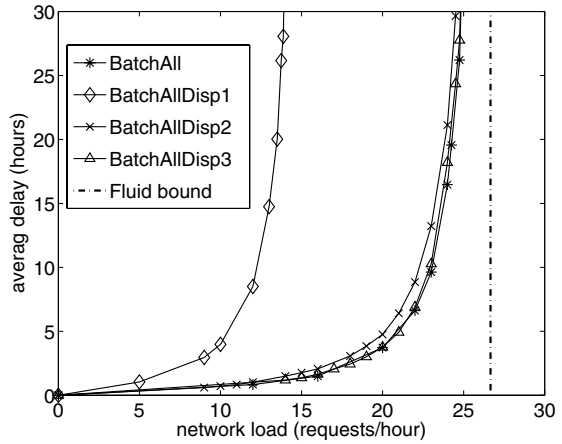


Fig. 5. Performance evaluation of algorithms `BatchAll` and `BatchAllDisp` for the LambdaRail topology and Pareto distributed file size. Algorithm `BatchAllDisp` is plotted with bounds of 3, 2 and 1 on path dispersion.

Figure 5 depicts the performance of the various algorithms and the fluid bound for the 11-node topology of figure 2(b). The file size follows a Pareto distribution. In this case, we observe that `BatchAllDisp` with as few as 3 paths per connection (or $\alpha = 0.107$) approximates `BatchAll` very closely. Since this network is sparser than the previous one, it is reasonable to obtain good performance with a smaller path dispersion.

VI. Conclusion

Advance reservation of dedicated network resources is emerging as one of the most important features of the next-generation of network architectures. In this paper, we made several advances in this area. Specifically, we proposed two new on-line algorithms for advance reservation, called `BatchAll` and `BatchLim`, that are guaranteed to achieve optimal throughput performance. We observed that path dispersion is essential to achieve full network utilization. However, splitting a transmission into too many different paths may render a flow-based approach inapplicable in many real-world environments. Thus, we presented a rigorous, theoretical approach to address the path dispersion problem and presented a method for approximating the maximum multicommodity flow using a limited number of paths. Specifically, while the number of paths between two nodes in a network scales exponentially with the number of edges, we showed that throughput competitiveness up to any desired ratio factor can be achieved with a number of paths scaling linearly with the total number of edges. In practice, our simulations indicate that three paths (in sparse graphs) to five paths (in dense graphs) may be sufficient.

We conclude by noting that the algorithms proposed in this paper can be either run in a centralized fashion (a reasonable solution in small networks) or using link-state routing and distributed signaling mechanism, such as enhanced versions of GMPLS [4] or RSVP [12]. Distributed approximations of the multicommodity flow have also been discussed in the literature [13]. Part of our future work will be to investigate these implementation issues into more detail.

Acknowledgment

This research was supported in part by the US Department of Energy under ECPI grant DE-FG02-04ER25605 and the US National Science Foundation under CAREER grant ANI-0132802.

References

[1] 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.

[2] R. Cohen and N. Fazlollahi and D. Starobinski, "Graded Channel Reservation with Path Switching in Ultra High Capacity Networks," in *Gridnets'06*, October 2006, san-Jose, CA.

[3] "On-demand secure circuits and advanced reservation systems," http://www.es.net/oscars/index.html.

[4] "Dynamic resource allocation via gmpls optical networks," http://dragon.maxgigapop.net.

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

[6] N. Fazlollahi and R. Cohen and D. Starobinski, "On the Capacity Limits of Advanced Channel Reservation Architectures," in *Proc. of the High-Speed Networks Workshop*, May 2007.

[7] R. Cohen and N. Fazlollahi and D. Starobinski, "Throughput optimal on-line algorithms for advanced resource reservation in ultra high-speed networks," arXiv/0711.0301, Tech. Rep., November 2007.

[8] F. Shahrokhi and D. W. Matula, "The Maximum Concurrent Flow Problem," *Journal of the ACM (JACM)*, vol. **37**, no. 2, April 1990.

[9] T. Leighton and S. Rao, "Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms," *Journal of the ACM (JACM)*, vol. **46**, no. 4, pp. 787 – 832, November 1999.

[10] "Coin-Or project," http://www.coin-or.org/.

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

[12] 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.

[13] B. Awerbuch and R. Khandekar and S. Rao, "Distributed Algorithms for Multicommodity Flow Problems via Approximate Steepest Descent Framework," in *Proceedings of the ACM-SIAM symposium on Discrete Algorithms (SODA)*, 2007.