Throughput-Competitive Advance Reservation with Bounded Path Dispersion

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

***Abstract*—In response to the high throughput needs of grid and cloud computing applications, several production networks have recently started to support advance reservation of dedicated circuits. An important open problem within this context is to devise advance reservation algorithms that can provide provable throughput performance guarantees, independently of the specific network topology and arrival pattern of reservation requests. In this paper, we first show that the throughput performance of greedy approaches, which return the earliest possible completion time for each incoming request, can be arbitrarily worse than optimal. Next, we introduce two new on-line, polynomial-time algorithms for advance reservation, called `BatchAll` and `BatchLim`. Both algorithms are shown to be throughput-optimal through the derivation of bounds on the maximum delay for $1+\varepsilon$ bandwidth augmented networks. The `BatchLim` algorithm has the advantage of returning the completion time of a connection immediately as a request is placed, but at the expense of looser delay performance than `BatchAll`. We then propose a simple approach that limits path dispersion, i.e., the number of parallel paths used by the algorithms, while provably bounding the maximum reduction factor in the transmission throughput. We prove that, although the number of different paths can be exponentially large, the actual number of paths needed to approximate any flow is quite small and proportional to the total number of edges in the network. Through simulation for various topologies and traffic parameters, we show that the proposed algorithms achieve reasonable average delay performance, even at network loads close to capacity bounds, and that three to five parallel paths are sufficient to achieve performance close to optimal.**

WE DON'T MENTION ANYTHING IN THE ABSTRACT ABOUT THE EXACT TERM: THROUGHPUT COMPETITIVE ALTHOUGH THIS IS THE TITLE OF THE PAPER!

I. Introduction

The Transport Control Protocol (TCP) has been so far considered and used as the core infrastructure for all sorts of data transfer including bulk FTP applications. Yet, it has recently been observed that in ultra high speed networks, there exists a large gap between the capacity of network links and the maximum end-to-end throughput achieved by TCP [1]. This gap, which is mainly attributed to the shared nature of Internet traffic, is becoming increasingly problematic for modern grid and cloud computing applications, requiring the transfer of extremely large datasets on the orders of terabytes and more.

To support high-end applications, the scientific community has been devoting significant efforts to develop an alternative 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 advance reservation paradigm has been successfully tested in a number experimental projects, such as UltraScienceNet [1] and OSCARS [3], and is now part of the operation of production networks, such as Internet2 and ESnet4 [4].

Several protocols and algorithms have been proposed in the literature to support advance reservation (cf. Section II). However, to the authors' knowledge, none of them provides throughput performance guarantees. Instead, most are based on heuristics or 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 first uncover fundamental limitations of greedy algorithms. Specifically, we show that there exists network topologies and request patterns for which the maximum throughput of these algorithms can be arbitrarily smaller than the optimal throughput, as the network size grows.

Next, we propose a new polynomial-time advance reservation algorithm, called `BatchAll`, that provably achieves maximum throughput for *any* network topology and pattern of request arrivals. Instead of immediately reserving a path for each incoming request as in a greedy algorithm, `BatchAll` accumulates several arrivals in a batch and assigns a more efficient set of flow paths to the whole batch, based on a maximum concurrent flow optimization. We prove the throughput-optimality of `BatchAll` by providing a bound on the ratio of the maximum delay experienced by any request in an $1+\epsilon$ augmented resource network to the maximal delay experienced by any request in the original network using the optimal algorithm.

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. We therefore propose another throughput-optimal algorithm, called `BatchLim`, which provides the completion time immediately as a request is placed, but at the expense of a slightly looser bound on the

A shorter, preliminary version of this paper appeared in the proceedings of the High-Speed Networking Workshop at IEEE INFOCOM 2008.

maximum delay ratio. This algorithm operates by limiting the length of each batch.

Our model assumes that the network infrastructure supports path dispersion, i.e., multiple paths in parallel can be used to route traffic, which has recently been shown to be feasible on the ESnet production network [5]. Nevertheless, 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 prove that this bound is tight. 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.

Finally, we provide simulation results illustrating the average delay performance of the various proposed algorithms as a function of the network load. The results are compared to a capacity bound, whose value represents an upper bound on the maximum network load for which the average delay of requests is still bounded. We show that `BatchAll` approaches the capacity bound at a reasonable delay value. Further, we propose and evaluate a modified version of the algorithm, called `BatchAll+`, whose performance is even closer to the bound. With respect to path dispersion, we show that excellent performance can be achieved with as few as five or so parallel paths per connection.

The rest of this paper is organized as follows. In Section II, we scan related work on advance reservation, competitive approaches, and path dispersion. In Section III, we introduce our model and define performance measures of interest. In section IV, we present natural greedy approaches and demonstrate their inefficiency. Then, in Section V, we describe the `BatchAll` and `BatchLim` algorithms and prove their throughput-optimality. Our method for bounding path dispersion is described and analyzed in Section VI. In Section VII, simulation results evaluating the performance of the algorithms for different network topologies and traffic parameters are presented. We conclude the paper in Section VIII.

II. Related Work

There exists a rich literature on advance reservation of network resources. Thus, we focus on algorithmic work, which is most relevant to this paper. We refer the interested reader to [6], for further discussion on architectures and services.

Several papers consider the problem of joint routing and scheduling of file transfers. Ref. [7] introduces a scheduling algorithm for large file transfers over paths with varying bandwidth. Ref. [8] analyzes the problem of offline scheduling and routing of file transfers from several users, each storing multiple files, to a single receiver node. Ref. [9] considers a similar model but also proposes algorithms to address the problem of rescheduling connections that have not completed. Ref. [10] proposes greedy algorithms guaranteeing earliest completion to each incoming request. Ref. [2] introduces an algorithmic framework, called GCR, that assigns grades to paths according to a general optimization objective, e.g., shortest path, widest path, earliest path, or a combination of those, and then select the highest graded path. GCR also supports path switching, which is shown to enable significant performance improvement.

Advance reservation plays an important role in optical networks, such as those based on optical flow switching (OFS) architectures [11, 12]. Thus, a number of papers investigate joint routing-scheduling optimization with the additional constraint of maintaining wavelength continuity along a path. For instance, [13] focuses on the design of effective routing and wavelength assignment heuristics and [14, 15] propose various load balancing approaches to allocate lightpaths.

In contrast to the above work, the algorithms that we propose in this paper provide theoretical guarantees on throughput performance. These results are obtained under the following relaxations: (i) job requests do not have strict bandwidth requirements; (ii) no additional constraints (e.g., wavelength continuity) are imposed. Hence, our model is appropriate for elastic applications, such as bulk transfer, operating under a single lambda service [1].

Next, we summarize work related to competitive algorithms for advance reservation. Ref. [16] discusses the on-line ftp problem, where reservations are made for channels for the transfer of different files. The presented algorithm provides a 4-competitive algorithm for the makespan (the total completion time). However, [16] focuses on the case of fixed routes. When routing is also to be considered, the time complexity of the algorithm presented there may be exponential in the network size. Ref. [17] proposes an off-line approximation algorithm for accommodating advance reservation of job requests in some specific topologies, e.g., lines and trees. Ref. [18] studies a related problem, whereby the path selection is based on several choices supplied by a user. Ref. [19] focuses on the problem of joint routing and scheduling in packet switched network in an adversarial setting and provides delay bounds. The model is for packet scheduling, rather than file (job) scheduling as in our paper. Moreover, [19] puts some restrictions on the adversary (e.g., the maximum amount of packets that it can inject during any time interval), which we do not need in this paper.

Most of the other work on competitive routing algorithms mainly focuses on call admission, without the ability of advance reservation [20, 21]. In [22], a combination of call admission and circuit switching is used to obtain a routing scheme with a logarithmic competitive ratio on the total revenue received. A competitive routing scheme in terms of the number of failed routes in the setting of ad-hoc networks is presented in [23]. Finally, a competitive algorithm for admission control and routing in a multicasting setting is presented in [24].

In [25], a throughput-optimal scheme for packet switching at the switch level is presented. The scheme is based on a convergence to the optimal multicommodity flow using delayed decision for queued packets. Their results somewhat

resemble our BatchAll algorithm. However, their scheme require packet lengths to be independent and applies to specific switch architectures, whereas our scheme addresses the full routing-scheduling question for general graphs and does not impose specific statical requirements (e.g., independence) on the arrival process or the packet length. In [26], a queuing analysis of several optical transport network architectures is conducted. It is shown that, under certain conditions on the arrival process, some of the schemes can achieve the maximum network rate. This paper does not address the full routing-scheduling issue either. Another difference with the above two papers is that we provide an algorithm, BatchLim, guaranteeing the completion time of a job at the time of its arrival.

Many papers have discussed the issue of path dispersion and attempted to achieve good throughput with limited dispersion. A survey of some results in this field is given in [27]. In [28, 29], heuristic methods of controlling multipath routing and some quantitative measures are presented. As far as we know, our work proposes the first formal treatment allowing the approximation of a flow using a limited number of paths at any desired level of accuracy.

III. Network Model

A. Notation and assumptions

We consider 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 $C(e)$. Any pair of nodes may request a connection at any time. 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 traffic.

In subsequent sections, we will make frequent use of multicommodity functions. The multicommodity flow problem is a linear planning problem returning `true` or `false` based upon the feasibility of fully transmitting a list of jobs between given source-destination pairs concurrently during a given time window, 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 window.

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 [30, 31].

We assume that no queuing delays arise in relay nodes in the network. Therefore, the instantaneous departure rate of information at each relay node equals the arrival rate. It is a reasonable assumption for the networks under consideration which reserves dedicated bandwidth to each flow. As such, the average transmission rate between all pairs in the network for any time interval T must be a feasible multicommodity flow.

B. Performance metrics

The main performance metric of interest in this paper is the *saturation throughput*, defined as follows. Suppose requests generate an average demand over time of $\lambda_{ij} = \alpha_{ij}\lambda$ bits/second from each node i to node j, where α_{ij} is a fixed parameter and $j \neq i$. Define the *delay* of a request as the amount of time elapsing from the point where the request for a connection set-up arrives till the point where the corresponding connection completes. Then, the saturation throughput, denoted λ^*, is the maximum value of λ (or the supremum, if the maximum does not exist) for which the delay experienced by any request is finite almost surely.

Determining analytically the value of λ^* is a difficult task in general as the answer depends on the graph topology and the statistics of the arrival process. Yet, even without knowing λ^*, one can devise algorithms that achieve optimal or competitive saturation throughput, as described later.

A simple upper bound on λ^* can be obtained as follows. Consider a set of feasible multicommodity flows g_{ij} assigned to each pair of nodes i and j, such that $g_{ij} = \alpha_{ij}g$. The maximization of g is a linear planning problem known as the *Maximum Concurrent Flow Problem (MCFP)* [30]. The maximum value of g in MCFP, denoted g^*, provides an upper bound on the saturation throughput λ^*. This result holds true because for every achievable value of λ there must exist an equally feasible value g.

IV. Greedy Algorithms

In this section, we present greedy algorithms and show that their saturation throughput can be arbitrarily far from optimal as the network size grows.

A. The Greedy algorithm

A seemingly natural way to implement advance reservation is to follow a greedy procedure, where, at the time a request is placed, the request is allocated a path (or set of paths) guaranteeing the earliest possible completion time. We refer to this approach as Greedy and explain it next.

The Greedy algorithm divides the time axis into slots delineated by events. Each event corresponds to a set-up or teardown 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 $t_1 = t$ (the time at which the current request is placed) and $t_{n+1} = \infty$.

Let $W_i = \{b_i(1), b_i(2), \ldots, b_i(|E|)\}$ be the vector of reserved bandwidth on all links at time slot i where $i = 1, \ldots, n$, and $b_i(e)$ denote the reserved bandwidth on link e during slot i, with $e = 1, \ldots, |E|$.

For each slot $i \in L$, construct a graph G_i, where the capacity of link $e \in E$ is $C_i(e) = C(e) - b_i(e)$, i.e., $C_i(e)$ represents the available bandwidth on link e during slot i.

In order to guarantee the earliest completion time, `Greedy` repeatedly performs a maximum flow allocation between nodes s and d, for as many time slots as needed until the entire file is transferred. This approach ensures that, in each time slot, the maximum possible number of bits is transmitted and, hence, the earliest completion time feasible at the arrival instance of the request is achieved. The `Greedy` algorithm can thus be concisely described with the following pseudo-code:

1) Initialization
   - Set initial time slot: i=1.
   - Set initial size of remaining file: $r = f$.
2) If $\texttt{maxflow}(G_i, s, d, r) \leq t_{i+1} - t_i$ (i.e., all the remaining file can be transferred during the current time slot) then,
   Exit step:
   - Update W_i by subtracting the used bandwidth from every link in the new flow and, if the file transfer of the new flow completes before t_{i+1}, create a new event t_{i+1} corresponding to the end of the connection.
   - Exit procedure.
3) Else,
   Non-exit step:
   - Update W_i by subtracting the used bandwidth from every link in the new flow.
   - $r = r - r \times (t_{i+1} - t_i)/\texttt{maxflow}(G_i, s, d, r)$ (update size of remaining file).
   - $i = i + 1$ (advance to the next time slot).
   - Go back to step 2.

The complexity of this algorithm depends on the number of slots examined before completely serving an arriving job. Noting that at each one of the examined slots a run of `maxflow` is carried out, this algorithm is expensive to implement in practice since the number of slots can be very large, especially at high load.

B. The `Greedy_shortest` algorithm

The `Greedy_shortest` algorithm is a variation of `Greedy`, where only shortest paths (in terms of number of hops) between the source s and destination d are utilized to route data. To implement `Greedy_shortest`, we employ exactly the same procedure as in `Greedy`, except that we prune all the links not belonging to one of the shortest paths using breadth first search. Note that for a given source s and destination d, the pruned links are the same for all the graphs G_i.

C. Inefficiency Results

We next constructively show that there exist certain arrival patterns, for which the saturation throughput achieved by both `Greedy` and `Greedy_shortest` is significantly lower than optimal. Specifically, we present cases where this measure for `Greedy` and `Greedy_shortest` is $\Omega(|V|)$ times smaller than the optimal value.

Theorem 1: For any given vertex set with cardinality $|V|$, there exists a graph $G(V, E)$ such that the saturation throughput of `Greedy` is $|V|/2$ times smaller than the optimal saturation throughput.

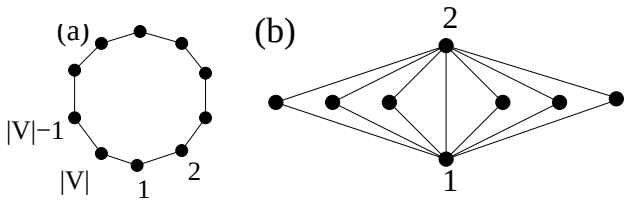


Fig. 1. Examples of graph topologies for which greedy algorithms perform far from optimal.

Proof: Consider the ring network shown in Fig. 1(a). Suppose that every link is an undirected 1 Gb/s link and requests arrive in the following order (we assume negligible delay between arrivals of requests): 1Gb request from node 1 to node 2, 1Gb request from node 2 to node 3, 1Gb request from node 3 to node 4, ..., 1Gb request from node $|V|$ to node 1.

The `Greedy` algorithm will allocate the maximum flow to each request, meaning that it will split the data flow to two paths, half flowing directly along a one hop path, and half flowing through the alternate path along the entire ring. Thus, the first job requires $1/2$ second for completion. Since the first job occupies all network resources, the second job has to wait until the completion of the first job before being served. Likewise, each new request will have to wait for the previous one to end, resulting in a completion time of $|V|/2$ seconds.

On the other hand, the optimal time is just one second using the direct link between each pair of nodes. Assuming the above pattern of requests repeats periodically, an optimal algorithm can support up to one request per second between each pair of neighboring nodes before reaching saturation, while `Greedy` can support at most $2/|V|$ request per second between each pair of nodes, hence proving the theorem. A similar proof can be shown for directed graphs. ■

We next show that restricting routing to shortest paths does not solve the inefficiency problem.

Theorem 2: For any given vertex set cardinality $|V|$, there exists a graph $G(V, E)$ such that the saturation throughput of `Greedy_shortest` (or any other algorithm using only shortest path routing) is $|V| - 1$ times smaller than the optimal saturation throughput.

Proof: Consider the network depicted in Fig. 1(b), where all requests are from node 1 to node 2, and only the direct path is used by the algorithm. In this scenario, an optimal algorithm would use all $|V| - 1$ paths between nodes 1 and 2. Hence, the optimal algorithm can achieve a saturation throughput $|V| - 1$ times higher than `Greedy_shortest`. ■

Both theorems imply that:

Corollary 3: There exist networks for which the delay of each job is bounded using the optimal algorithm, but is unbounded using either the `Greedy` or `Greedy_shortest` algorithm, even if the capacity of each link is multiplied by $|V|/2 - 1$.

V. Throughput-Optimal Algorithms

In this section, we present on-line, polynomial-time algorithms based on the idea of *deferred bandwidth allocation*.

Thus, arriving requests are not assigned bandwidth as long as transmissions go on in the network. Once existing transmissions complete, pending requests are served altogether. The proposed algorithms guarantee that the maximum delay experienced by any request in a network with augmented resources lies within a finite multiplicative factor of the value achieved with an optimal off-line algorithm (i.e., with full knowledge of the future and unlimited computational resources) in the original network. Hence, the proposed algorithms reach the maximum throughput achievable in the sense that for any network and any optimal algorithm that can support a list of jobs with a bounded delay, the proposed algorithms also guarantee a bounded delay at the price of augmenting the capacity of each link by a multiplicative factor of $(1+\epsilon)$, for any arbitrarily small constant $\epsilon > 0$. This result stands in contrast to the results of Corollary 3, which indicates that, even with high augmentation, the greedy approaches are not competitive with the optimal algorithm.

A. The `BatchAll` Algorithm

Our first algorithm, called `BatchAll`, cumulatively performs bandwidth allocation for a group of requests. The group of all pending requests forms a *batch* that is served once existing transmissions complete. Batches of jobs are not predetermined since they are formed by on-line aggregation of random arrivals. We denote by L the list of jobs assigned to the next batch, by t_c the end time of the currently running batch of jobs, and by clk the value of the clock (initially set to 0). `BatchAll` can then concisely be described as follows:

1) Set t_c to 0.
2) When a request $l = \{s, d, f\}$ arrives at $clk = t$, give an immediate connection starting time and a connection ending time of $t_c = t + \texttt{maxflow}(G, l)$.
3) Set $L \leftarrow$ null
4) While $clk < t_c$,
   - If a request $l = \{s, d, f\}$ arrives when $clk = t$:
     - Set $L \leftarrow L \cup l$ (i.e. add the job to the waiting batch)
     - Mark t_c as its connection starting time
5) At time $clk = t_c$,
   - If L is empty (i.e., there is no pending request) go back to step 2.
   - Else assign a connection ending time $t_c = t_c + \texttt{maxflow}(G, L)$ to all requests in the batch L and go back to step 3.

As an example, one can verify that `BatchAll` achieves optimal throughput for the scenarios used in the proofs of Theorems 1 and 2 , which showed the inefficiency of greedy algorithms.

Next, we compare the delay performance of the `BatchAll` algorithm in an *augmented* network to that of the optimal off-line algorithm in the original network. The augmented network is similar to the original network, other than each link e has capacity $(1+\varepsilon)C(e)$ instead of $C(e)$, where $\varepsilon > 0$. Alternatively, one may compare the performance of `BatchAll` to that of the optimal off-line algorithm in a lower capacity network, allowing the maximum rate of only $(1+\varepsilon)^{-1}C(e)$ for each link e.

Theorem 4: Suppose that we augment the resources of a network such that every edge has $(1+\varepsilon)$ times its original capacity, for any $\varepsilon > 0$. Then, for all requests arriving up to any time t^* in such a network, the maximum delay using `BatchAll` is no more than $2/\varepsilon$ times the maximum delay for all requests arriving up to t^* using the optimal algorithm in the original network.

Proof: Consider the augmented resource network. Take the maximum length batch, say i, that accommodates requests arriving before time t^* and mark its length by T. Since the batch before this one was of length at most T, and all requests in batch i were received during the execution of previous batch, denoted by $i-1$, then the total waiting time of each of these requests was at most $2T$. Thus, $2T$ is an upper bound on the maximum delay in the augmented network achieved using `BatchAll` up to time t^*. Since `BatchAll` uses a maximum concurrent flow procedure to allocate bandwidth in each batch, the total time for handling all requests received during the execution of batch $i-1$ must have been at least T in the augmented network, or $(1+\varepsilon)T$ in the original network. Since all of these requests arrived during a time interval of length at most T and the completion time for all of them together is at least $T + \varepsilon T$, one of them must have waited at least εT until job completion under the optimal algorithm. Thus, the maximum delay in the original network using an optimal offline algorithm up to any time t^* is at least εT. Therefore, the ratio between the maximum waiting time is at most $2/\varepsilon$. ■

The throughput-optimality of `BatchAll` immediately follows from the previous theorem:

Corollary 5: For any given network, the saturation throughput of `BatchAll` is identical to that of the offline optimal algorithm because, for any arbitrarily small $\epsilon > 0$, the delay of any request is guaranteed to be, at worse, a finite multiplicative factor larger than the maximum delay of the optimal algorithm in the reduced resources network.

B. The `BatchAll+` algorithm

We propose next a simple modification of `BatchAll` to improve its delay performance. The main idea is as follows. Whenever a job request arrives, we first check if we can allocate it to the current batch without affecting the completion time of the batch. If yes, we immediately allocate bandwidth to the job. If no, we proceed as in `BatchAll`, and add the current job to the list of jobs waiting for the next batch. To implement this idea, we keep track of the available bandwidth on each link of the graph during the current batch. We denote the resulting graph by G'. When a request arrives, we run `maxflow` over G' to examine whether the network can immediately accommodate the new job without disturbing the completion time of currently running jobs. Thus, `BatchAll+` modifies step 4 of `BatchAll` as follows:

- While $clk < t_c$,
  - If request $l = \{s, d, f\}$ arrives when $clk = t$:
    - If $\texttt{maxflow}(G', l) \leq t_c - t'$ then,

- Add the job to the running batch such that it completes simultaneously with the rest of running jobs.
- Deduct all bandwidth assigned to l from G'.

* Otherwise,
  - Set $L \leftarrow L \cup l$ (i.e. add it to the waiting batch)
  - Mark t_c as its connection starting time

Note that G' must also be updated in step 2 and step 5 in the `BatchAll` pseudo-code, after `maxflow` is run on G, to reflect the available bandwidth on each link.

Since computing `maxflow` is feasible in polynomial time, the computation complexity per request remains polynomial after this modification. It is not difficult to realize that Theorem 4 continues to hold after the modification. Therefore, `BatchAll`+ also achieves optimal saturation throughput.

C. The `BatchLim` Algorithm

The `BatchAll` and `BatchAll`+ algorithms return the starting time of a connection at the time of the request but the completion time is computed only when the connection actually starts. We next present another algorithm, called `BatchLim`, with a slightly larger delay ratio guarantee but which always returns the completion time at the arrival time of the request. Note that since the delay ratio guarantee remains finite, `BatchLim` is also throughput-optimal.

For each arriving request, the algorithm maintains a list of times, t_i, $i = 1, \ldots, n$ where $t_1 = t$ (the time at which the current request is placed) and each time interval $[t_i, t_{i+1}]$ correspond to a batch of previously allocated jobs, which are either currently running (for the case $i = 1$) or scheduled to start in the future (for $i > 1$). When a new request between a source s and a destination d arrives at time t, two cases may happen: (i) if all previous jobs have completed, a new interval $[t, t + M]$ where $M = \texttt{maxflow}(G, s, d, f)$ is created and the job is assigned to it; (ii) otherwise, attempts are made to add the job to one of the scheduled time intervals $[t_i, t_{i+1}]$, where $2 \leq i \leq n - 1$, by using a multicommodity flow calculation. If the attempts fail, a new interval $[t_n, \max(2t_n - t, t_n + M)]$ is appended to the time list and the job is assigned to this interval.

We next provide a detailed description of how the algorithm handles a new request arriving at time t. We use the tupple $l = \{s, d, f\}$ to denote the job request and the list L_i to denote the set of jobs already assigned to interval $[t_i, t_{i+1}]$:

1) If no job is currently running, then:
   - Set $M = \texttt{maxflow}(G, l)$.
   - Set $t_2 = t + M$.
   - Assign job l to interval $[t_1, t_2]$ and exit.
2) Else check if job l can be assigned to an existing interval:
   - Set $i = 2$.
   - While $i \leq n - 1$:
     - If l can be fitted into interval $[t_i, t_{i+1}]$ (i.e. $\texttt{multicomm}(G, L_i \cup l, t_{i+1} - t_i) = \texttt{true}$) then assign job l to $[t_i, t_{i+1}]$ and exit.
     - Else set $i = i + 1$.
3) Else create new interval:
   - Set $M = \texttt{maxflow}(G, l)$.
   - If $t_n - t < M$, then $t_{n+1} = t_n + M$.
   - Else $t_{n+1} = 2t_n - t$.
   - Assign job l to interval $[t_n, t_{n+1}]$ and exit.

Fig. 2 illustrates runs of the `BatchAll` and `BatchLim` algorithms for the same set of requests.

The following lemma shows that the delay of any request arriving at some time t and for which a new interval $[t_n, t_{n+1}]$ is created is at most twice the length of that interval.

Lemma 6: For any $n \geq 1$, $(t_{n+1} - t) \leq 2(t_{n+1} - t_n)$, where t is the time at which the interval $[t_n, t_{n+1}]$ is formed.

Proof: Suppose a request $l = \{s, d, f\}$ arrives at time t and is assigned a new interval $[t_n, t_{n+1}]$. Then, by definition $t_{n+1} - t_n = \max(M, t_n - t)$ where $M = \texttt{maxflow}(G, l)$. Thus $t_n - t \leq t_{n+1} - t_n$. Adding $t_{n+1} - t_n$ to each side of the previous inequality, we derive the lemma. ■

The following lemma states that the delay experienced by any request is at most twice the length of the interval to which it is allocated.

Lemma 7: Suppose a request arrives at time t and is assigned to interval $[t_i, t_{i+1}]$, where $i = 1, 2, \ldots, n$. Then $(t_{i+1} - t) \leq 2(t_{i+1} - t_i)$.

Proof: Suppose first that the request, call it l, is assigned to a new interval $[t_n, t_{n+1}]$, then the proof follows directly from Lemma 6. Next suppose that request l is assigned to an existing interval $[t_i, t_{i+1}]$, where $i < n$. Then, it must be that request l arrived after the request, call it l', for which this interval was formed. Thus, the delay of request l is smaller than the delay of request l' and the proof follows again from Lemma 6. ■

The next theorem provides a guarantee on the maximum delay ratio of `BatchLim` in an $(1 + \epsilon)$-augmented network versus the optimal delay in the original network. The theorem implies that `BatchLim` is also throughput-optimal.

Theorem 8: Suppose that we augment the resources of a network such that every edge has capacity $(1 + \varepsilon)$ times the original capacity, for any $\varepsilon > 0$. Then for requests arriving up to any time t^* in such a network, the maximum delay for a request using `BatchLim` is at most $4/\varepsilon$ times the maximum delay for requests arriving up to t^* using the optimal algorithm in the original network.

Proof: Consider the maximum length time interval formed up to t^* by `BatchLim` in the $(1 + \varepsilon)$ augmented network. Suppose that this interval was created at time t and denote it $[t_n, t_{n+1}]$. When creating this interval there are two possibilities:

1) $t_{n+1} = t_n + M$, where M is the minimum time for the new job completion. In this case, the minimum delay using any algorithm in the original network would have been $(1 + \varepsilon)M$. By Lemma 7, the total delay for any job using `BatchLim` is at most $2M$, and therefore the ratio of waiting times is $2/(1 + \varepsilon) < 4/\varepsilon$.
2) $t_{n+1} = t_n + (t_n - t)$. Consider the previous interval $[t_{n-1}, t_n]$, which was formed at some time $t' < t$ when another request arrived that could not be fitted into any previous intervals. By the algorithm definition $t_n \geq t_{n-1} + (t_{n-1} - t')$. By maximality of $[t_n, t_{n+1}]$ it follows that $t_{n+1} - t_n = t_n - t \geq t_n - t_{n-1}$, and thus,

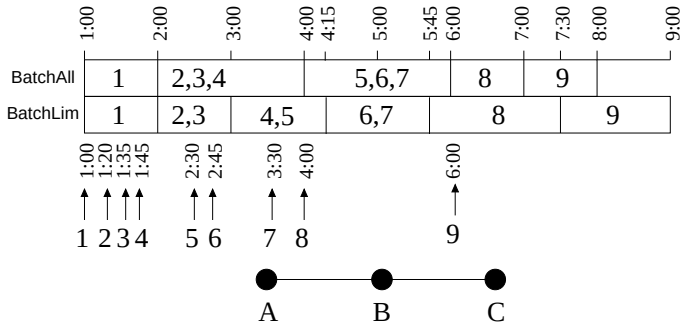


Fig. 2. Illustration of the batching process by the algorithms `BatchAll` and `BatchLim` over a simple network topology consisting of three nodes A, B and C connected via links of identical capacity. Connection requests are depicted by arrows and their arrival time. The odd numbered jobs request transmission between nodes A and B and the even numbered jobs request for transmission between B and C. Each file size corresponds to one hour of transmission at full link capacity. 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.

$t \leq t_{n-1}$. Note that not all the requests arriving during the interval $[t', t]$ (including the request arriving at time t, which initiated the creation of interval $[t_n, t_{n+1}]$) could have been completed during the interval $[t_{n-1}, t_n]$ or otherwise the new interval $[t_n, t_{n+1}]$ would not have been created at time t. Thus, the requests arriving during the interval $[t', t]$ require a time of at least $t_n - t_{n-1}$ for all to complete according to the multicommodity flow calculation. Thus, the time it would take to complete all jobs that arrived in $[t', t]$ under any possible algorithm is at least $t_n - t_{n-1}$ in the augmented network, and therefore, at least $(1+\epsilon)(t_n - t_{n-1})$ in the original network. Since no job can start before time t', the ending time for all these jobs in the original network must be at least $t' + (1+\epsilon)(t_n - t_{n-1}) \geq t' + (t_{n-1} - t') + \epsilon(t_n - t_{n-1}) = t_{n-1} + \epsilon(t_n - t_{n-1})$. Since the requests have arrived between $[t', t]$, the last request has arrived at time $t \leq t_{n-1}$ at the latest. Thus, one of the requests in this batch must have waited for at least $t_{n-1} + \epsilon(t_n - t_{n-1}) - t_{n-1} = \epsilon(t_n - t_{n-1})$. In other words, the maximum delay using the optimal algorithm in the original network is at least $\epsilon(t_n - t_{n-1})$. On the other hand, according to Lemma 7, the maximum delay of any request in the augmented network using `BatchLim` is at most $2(t_{n+1} - t_n)$. Since by assumption $t_{n+1} - t_n = t_n - t$ and since $t_n - t \leq t_n - t' \leq 2(t_n - t_{n-1})$, where the last inequality follows from Lemma 6, then the maximum delay of any request in the augmented network using `BatchLim` is at most $4(t_n - t_{n-1})$. Therefore, the ratio of the maximum delays is at most $4/\varepsilon$.

■

VI. 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.

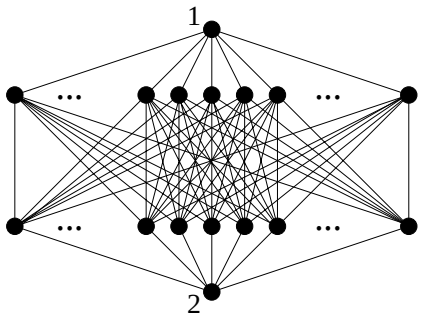


Fig. 3. A network demonstrating the optimality of Lemma 9. The edges incident to nodes 1 and 2 have infinite capacity, all other edges have capacity 1.

Lemma 9: In every flow of bandwidth F between two nodes on a directed graph $G(V, E)$ there exists a path of bandwidth at least $F/|E|$ between these nodes.

Proof: Remove all edges carrying flow with bandwidth strictly less than $F/|E|$ from the graph. The total flow carried by 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, which is of capacity at least F. Therefore, since the flow bandwidth is F, there must remain at least one path between the nodes after the removal. All edges in this path have flow bandwidth of at least $F/|E|$. Therefore, the path bandwidth is at least $F/|E|$. ■

Lemma 9 is asymptotically tight in order of both $|E|$ and $|V|$ as can be seen for the network shown in Fig. 3. In this network each path amounts for only $O(|E|^{-1}) = O(|V|^{-2})$ of the total flow between nodes 1 and 2. Therefore, to obtain a flow that consists of a given fraction of the maximum flow at least $O(|E|) = O(|V|^2)$ of paths must be used. If multigraphs are allowed, the lemma is exactly tight for a graph with 2 nodes and $|E|$ equal capacity edges between them.

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 10: For every flow of bandwidth F between two nodes on a directed graph $G(V, E)$ and for every integer $n \leq |E|$ there exists a set of at most n paths achieving a flow of bandwidth at least $nF/|E|$.

Proof: We prove by induction on n. For $n = 1$, there exists, by Lemma 9, a path of capacity at least $F/|E|$. Take the edge (i, j) with minimum capacity of the path and decrease the flow through all edges in the path by its weight w_{ij}. Now, the flow at at least one edge is decreased to zero, and this edge may be removed without affecting the residual flow. Thus, we are left with a graph with flow $F - w_{ij}$ and with at most $|E| - 1$ edges.

Now assume that by using n paths one can achieve a flow of $f_n \geq nF/|E|$. The residual flow if $F - f_n$, and the number of edges left is at most $|E| - n$. The flow of the widest left path, p_n, is, by Lemma 9, at least $p_n \geq (F - f_n)/(|E| - n)$.

Thus the total flow captured in the first $n+1$ paths is

$$\begin{aligned} f_n + p_n &\geq f_n + \frac{F - f_n}{|E| - n} = \frac{(|E| - n - 1)f_n + F}{|E| - n} \\ &\geq \frac{(|E| - n - 1)n\frac{F}{|E|} + F}{|E| - n} \\ &= \frac{(|E| - n)(n+1)F}{|E|(|E| - n)} = \frac{n+1}{|E|}F \, . \end{aligned}$$

This completes the induction. ∎

Theorem 10 provides both an algorithm for reducing the number of paths and a bound on the throughput loss. To approximate the flow using a limited number of paths, consider only links carrying flow with bandwidth greater than $F/|E|$ and find a path p among those (if multiple paths are available, select the widest). Suppose path p carries a flow f, then remove the bandwidth used by flow f from all the links on path p and iteratively repeat the same procedure until a satisfactory approximation of the flow is obtained. Note that the maximum number of paths needed to achieve an approximation of the flow to within a constant factor is linear in $|E|$ and the number of paths needed to obtain the full flow is exactly $|E|$, while the number of possible paths may be exponential in $|E|$. Again, this theorem is asymptotically tight, and exactly tight for multigraphs.

We illustrate the bounding path dispersion procedure with a simple example. Figure 4 shows the maximum flow from node 10 to node 3 on an 11-node topology. Links are assumed to be full duplex and link capacities are represented by surrounded numbers next to each link. The maximum flow calculated using a `maxflow` formulation from node 10 to node 3 is $F = 20$ Gb/s. Since the total number of edges is $|E| = 28$ in this graph, we have $F/|E| = 0.71$ Gb/s.[1] Therefore, at the first step, the algorithm only considers links 10-1, 1-2, 2-3, 10-9, 9-8, 8-7, 7-3 and 8-2 carrying a flow greater than 0.71 Gb/s and ignore all other links. Among the considered links, there exists three different paths between node 10 and 3, marked as $P1$, $P2$ and $P3$, and carrying flows of bandwidth 10, 5 and 5 Gb/s respectively. The algorithm selects the widest path $P1$ first. Subtracting the flow of path $P1$ from F, the remaining flow is 10 Gb/s. The links 10-1 and 1-2 are now saturated. Thus we are left with a residual flow of $F = 20 - 10 = 10$ Gb/s and a remaining $|E| = 26$ non saturated edges. We repeat the procedure and get $F/|E| = 0.38$ Gb/s. This time, only links 2-3, 10-9, 9-8, 8-7, 7-3 and 8-2 are considered. The algorithm can select either $P2$ or $P3$. Thus, with a bound of at most two paths per connection we can carry 75% of the full flow from node 10 to node 3 using paths $P1$ and $P2$ (or $P1$ and $P3$). If the bound is relaxed to three paths, then we can route the entire flow between the given nodes. Our simulation results in section VII-B reveals that with this topology, bounding flows with at most 3 paths per connection does indeed closely approximate full flow on average.

Using the above flow approximation in conjunction with the competitive algorithms, we 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 and reserve the total path bandwidth. We repeat this procedure until either the desired number of paths is reached or the entire flow is routed.

[1] In fact, the entire flow can be routed through 8 directed edges. Thus, one can also use $|E| = 8$ and obtain $F/|E| = 2.5$ Gb/s.

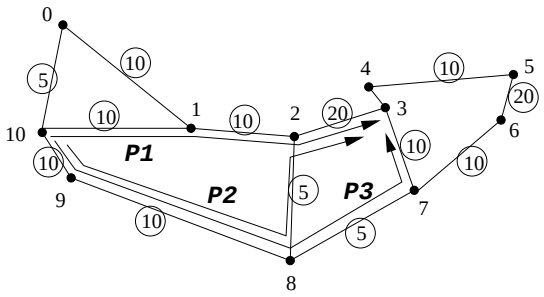
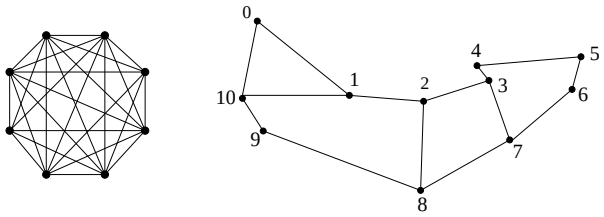


Fig. 4. Illustration of the path dispersion bounding procedure. Links are assumed to be full duplex and link bandwidths are displayed by surrounded numbers next to each link. The three paths connecting node 10 to node 3 are marked by $P1$, $P2$ and $P3$ with flow bandwidth of 10, 5 and 5 Gb/s respectively.



(a) 8 node (b) 11 node LambdaRail

Fig. 5. Simulation topologies.

VII. Simulations

In this section, we present simulation results illustrating the performance of the algorithms described in this paper. While the emphasis of the previous sections was on the throughput optimality (or competitiveness) of the proposed algorithms, here we are also interested in evaluating their average delay performance. The main points of interest are as follows: (i) how do the competitive algorithms fare with respect to each other and the capacity bound of section III-B? (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 [32] to solve multicommodity 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 5. One is a fully connected graph (clique) of eight

nodes and the other is an 11-node topology, similar to the National LambdaRail testbed [33]. 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.
  2) Exponential:
  $$F(x) = \exp(-\delta x), \text{ where } x \geq 0.$$
  In the simulations, $1/\delta = 2.475$ TB.
- *Source and Destination*: for each request, the source and destination are selected uniformly at random, except that they must be different nodes.

Each simulation point represents an average taken over at least 10^6 measurements.

Remark 11: For the special case of a graph consisting of a single link, the performance of `BatchAll` is identical to that of a gated $M/G/1$ queue [34]. Denote by λ the mean arrival rate of requests, by $\mathbb{E}[S]$ and $\mathbb{E}[S^2]$ the first and second moments of the service time (the service time is the file size divided by the link capacity), and by $\rho = \lambda\mathbb{E}[S]$ the link utilization. Then, the expected waiting time $\mathbb{E}[W]$ of a job from the point it arrives till the start of its batch is [34, Eq. 16]:

$$\mathbb{E}[W] = \frac{\lambda\mathbb{E}[S^2]}{2(1-\rho^2)}. \quad (1)$$

As in the standard $M/G/1$ queue, the mean waiting time is bounded only if the second moment of the service time distribution (or equivalently that of the file size distribution) is finite. Thus, for the case of Pareto distributed file sizes, the power law parameter β should be greater than 2. This requirement applies also to general networks.

B. Results

We first compare the performance of `BatchAll`, `BatchAllDisp` and `BatchLim` algorithms in Figures 6 and 7. Both figures show `BatchAllDisp` with multiple bounds on path dispersion. The capacity bounds are also depicted in dashed lines as a performance benchmark for each scenario. Figure 6 corresponds to the 8-node clique topology with exponentially distributed file sizes and Figure 7 corrsponds to the 11-node topology of Figure 5(b) with Pareto distributed file sizes.

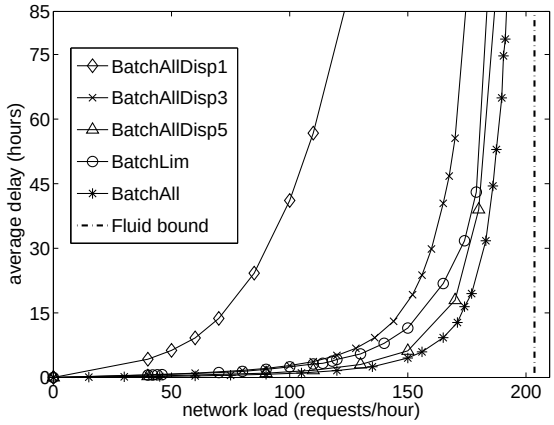


Fig. 6. Performance evaluation of algorithms `BatchAll`, `BatchLim` and `BatchAllDisp` for the 8-node clique topology and exponentially distributed file size. Algorithm `BatchAllDisp` is plotted with path dispersion bounds of 1, 3 and 5 paths per connection.

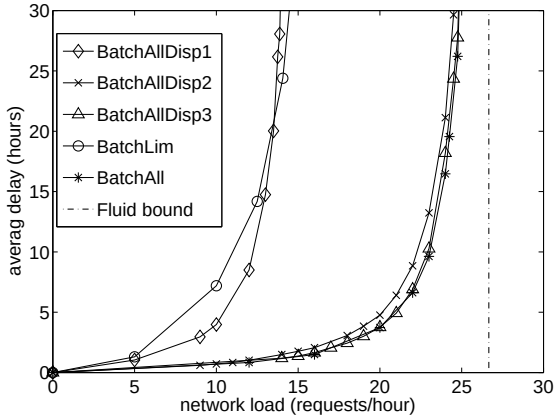


Fig. 7. Performance evaluation of algorithms `BatchAll`, `BatchLim` and `BatchAllDisp` for the 11-node topology and Pareto distributed file size. Algorithm `BatchAllDisp` is plotted with path dispersion bounds of 1, 2 and 3 paths per connection.

According to Figure 6, `BatchAll` approaches the capacity bound at a reasonably low average delay value. Interestingly, a path dispersion of at most five per connection (corresponding to $\alpha = 0.089$) is sufficient for `BatchAllDisp` to closely approximate `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. Figure 6 also 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 guarantee is looser.

Figure 7 depicts the performance of the various algorithms and the capacity bound for the 11-node topology of Figure 5(b). 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

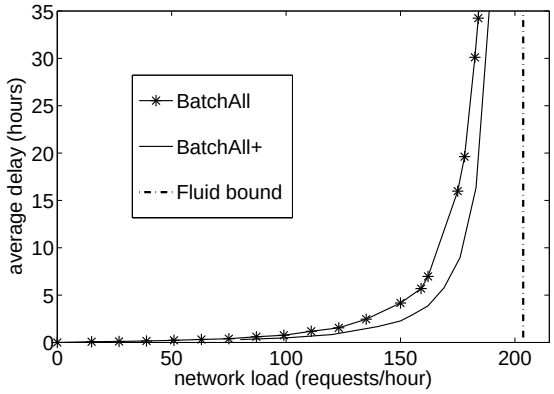


Fig. 8. Performance comparison of algorithms `BatchAll` and `BatchAll+` for the 8-node clique and Pareto file size distribution.

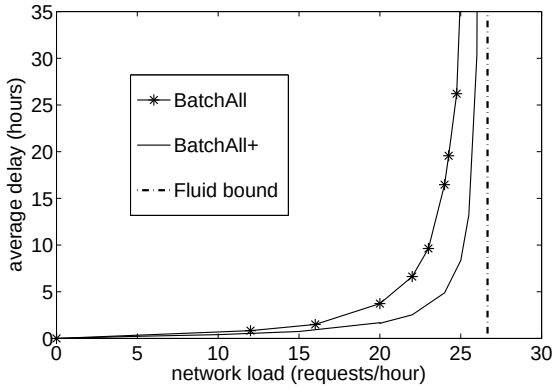


Fig. 9. Performance comparison of algorithms `BatchAll` and `BatchAll+` for the 11-node topology and Pareto file size distribution.

a smaller path dispersion. In this scenario with Pareto file size distribution, `BatchLim` performs significantly worse than `BatchAll` for the load range illustrated. We suspect that the performance of `BatchLim` is more sensitive to the file size distribution than `BatchAll` , because the batches formed by `BatchLim` are limited in time duration compared to those of `BatchAll`.

Figures 8 and 9 compare the performance of `BatchAll` and `BatchAll+` for the 8-node and 11-node topologies `BatchAll+`, with Pareto file size distribution. The figures show the superiority of `BatchAll+` over `BatchAll` in both cases, which is due to optimization in Step 4 of the algorithm.

VIII. Conclusion

In this paper, we considered the problem of devising throughput-optimal and throughput-competitive algorithms for networking architectures supporting advance reservation, a problem of particular relevance to modern grid and cloud computing applications. After showing the limitations of greedy approaches, we proposed two new on-line algorithms for advance reservation, called `BatchAll` and `BatchLim`, that are guaranteed to achieve optimal throughput performance. Specifically, we proved that both algorithms bounds the ratio of the maximum delay of any request in $1 + \varepsilon$ bandwidth augmented networks to the maximum delay of any request in the original network using the optimal off-line algorithm. While `BatchLim` has a slightly looser delay ratio guarantee than that of `BatchAll` (i.e., $4/\epsilon$ instead of $2/\epsilon$) and, based on our simulations, inferior average delay performance, it has the distinct advantage of returning the completion time of a connection immediately as a request is placed.

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) should be sufficient.

We also provided a simple and practical mechanism for enhancing the average delay achieved by our main algorithm, `BatchAll`. Specifically, the enhanced version, called `BatchAll+`, attempts at adding any arriving request to the current batch of running requests pending that the completion time of the batch remains unchanged. Such an improvement can also be applied to the algorithm with bounded path dispersion, `BatchAllDisp`.

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 [35]. Distributed approximations of the multicommodity flow have also been discussed in the literature [36]. An important research area open for future work will be to carefully investigate these implementation issues.

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. Rao, W. Wing, S. Carter, and Q. Wu, "UltraScience Net: Network Testbed for Large-Scale Science Applications," *IEEE Communications Magazine*, vol. 43, pp. 12–17, 2005.

[2] R. Cohen, N. Fazlollahi, D. Starobinski, "Path Switching and Grading Algorithms for Advance Channel Reservation Architectures," *IEEE/ACM Transactions on Networking (TON)*, vol. **17**, no. 5, pp. 1684–1695, October 2009.

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

[4] W. Johnston, "ESnet: Advanced Networking for Science," *SciDAC review*, 2007.

[5] C. Guok, J. Lee, and K. Berket, "Improving the bulk data transfer experience," *Internation Journal of Internet Protocol Technology*, vol. 3, no. 1, pp. 46–53, 2008.

[6] L. Burchard, "Networks with Advance Reservations: Applications, Architecture, and Performance," *Journal of Network and Systems Management*, vol. 13, pp. 429–449, 2005.

[7] H. Lee, M. Veeraraghavan, H. Li, and E. 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.
[8] 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.
[9] A. Banerjee, W. 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.
[10] R. Guerin and A. Orda, "Networks With Advance Reservations: The Routing Perspective," in *Proceedings of INFOCOM'00*, March 2000, Tel-Aviv, Israel.
[11] V. Chan, G. Weinchenberg, and M. Médard, "Optical Flow Switching," in *Proc. 3rd International Conference on Broadband Communications, Networks and Systems*, October 2006, pp. 1–8, San Jose, USA.
[12] B. Ganguly, "Implementation and modeling of a scheduled Optical Flow Switching (OFS) network," Ph.D. dissertation, Massachusetts Institute of Technology, 2008.
[13] J. Zheng and H. T. Mouftah, "Routing and Wavelength Assignment for Advance Reservation in Wavelength-Routed WDM Optical Networks," in *Proc. IEEE ICC*, vol. 5, 2002, pp. 2722–2726.
[14] 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.
[15] 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, UK.
[16] A. Goel, M. Henzinger, S. Plotkin, and E. Tardos, "Scheduling Data Transfers in a Network and the Set Scheduling Problem," in *Proc. of the 31st Annual ACM Symposium on the Theory of Computing*, 1999, pp. 189–199.
[17] L. Lewin-Eytan, J. Naor, and A. Orda, "Admission Control in Networks with Advance reservations," *Algorithmica*, vol. 40, pp. 293–304, 2004.
[18] Erlebach, T., "Call admission control for advance reservation requests with alternatives," ETH, Zurich, Tech. Rep. 142, 2002.
[19] M. Andrews, A. Fernandez, A. Goel, and L. Zhang, "Source routing and scheduling in packet networks," *Journal of the ACM (JACM)*, vol. **52**, no. 4, pp. 582–601, July 2005.
[20] Y. Azar, J. Naor, and R. Rom, "Routing strategies for fast networks," in *Proc. of IEEE INFOCOM*, 1992.
[21] B. Awerbuch, Y. Azar, A. Fiat, S. Leonardi, A. Rosen, "On-line competitive algorithms for call admission in optical networks," *Algorithmica*, vol. **31**, no. 1, pp. 29–43, 2001.
[22] S. A. Plotkin, "Competitive Routing of Virtual Circuits in ATM Networks," *IEEE J. on Selected Areas in Communication (JSAC)*, vol. **13**, no. 6, pp. 1128–1136, 1995.
[23] B. Awerbuch, D. Holmer, H. Rubens, and R. D. Kleinberg, "Provably competitive adaptive routing," in *INFOCOM*, 2005, pp. 631–641.
[24] A. Goel, M. R. Henzinger, and S. A. Plotkin, "An online throughput-competitive algorithm for multicast routing and admission control." *J. Algorithms*, vol. 55, no. 1, pp. 1–20, 2005.
[25] A. K. Y. Ganjali and D. Shah, "Input Queued Switches: Cell Switching vs. Packet Switching," in *Proceedings of INFOCOM'03*, March 2003, San-Fancisco, CA, USA.
[26] G. Weichenberg, V. Chan, and M. Médard, "On the Capacity of Optical Networks: A Framework for Comparing Different Transport Architectures," *IEEE J-SAC*, vol. 25, pp. 84–101, 2007.
[27] E. Gustafsson and G. Karlsson, "A Literature Survey on Traffic Dispersion," *IEEE Network*, vol. **11**, no. 2, pp. 28–36, March-April 1997.
[28] J. Shen, J. Shi, and J. Crowcroft, "Proactive Multi-path Routing," *Lect. Notes in Comp. Sci.*, vol. **2511**, pp. 145–156, 2002.
[29] R. Chow, C. W. Lee, , and J. Liu, "Traffic Dispersion Strategies for Multimedia Streaming," in *Proceedings of the 8th IEEE Workshop on Future Trends of Distributed Computing Systems*, 2001, p. 18.
[30] F. Shahrokhi and D. W. Matula, "The Maximum Concurrent Flow Problem," *Journal of the ACM (JACM)*, vol. **37**, no. 2, April 1990.
[31] 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.
[32] "Coin-Or project," http://www.coin-or.org/.
[33] "National LambdaRail Inc." http://www.nlr.net/.
[34] B. Avi-Itzhak and S. Halfin, "Response Times in Gated M/G/1 Queues: The Processor-Sharing Case," *Queuing Systems*, vol. 4, pp. 263–279, 1989.
[35] "Dynamic resource allocation via gmpls optical networks," http://dragon.maxgigapop.net.
[36] B. Awerbuch, 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.

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.