

Scheduling Bulk File Transfers with Start and End Times^{*}

Kannan Rajah, Sanjay Ranka, and Ye Xia^{*}

Computer and Information Science and Engineering Department
University of Florida

301 CSE Building, P.O. Box 116120, Gainesville, FL 32611-6120

Abstract

The advancement of optical networking technologies has enabled e-science applications that often require transport of large volumes of scientific data. In support of such data-intensive applications, we develop and evaluate control plane algorithms for scheduling bulk file transfers, where each transfer has a start time and an end time. We formulate the scheduling problem as a special type of the multi-commodity flow problem. To cope with the start and end time constraints of the file-transfer jobs, we divide time into uniform time slices. Bandwidth is allocated to each job on every time slice and is allowed to vary from slice to slice. This enables periodical adjustment of the bandwidth assignment to the jobs so as to improve a chosen performance objective: throughput of the concurrent transfers. In this paper, we study the effectiveness of using multiple time slices, the performance criterion being the tradeoff between achievable throughput and the required computation time. Furthermore, we investigate using multiple paths for each file transfer to improve the throughput. We show that using a small number of paths per job is generally sufficient to achieve near optimal throughput with a practical execution time, and this is significantly higher than the throughput of a simple scheme that uses single shortest path for each job.

Key words: Bulk File Transfers, E-science, Optical Networks, Scheduling, Multipath, Rerouting, Maximum Concurrent Flow

1. Introduction

In the past two decades, optical networking technologies have revolutionized communications. The widespread deployment of fiber optic infrastructure has led to low cost, high capacity optical connections. This networking advancement has enabled e-science applications that often require management and transport of large volumes of scientific data [37,9,2]. For instance, the Large Hadron Collider (LHC) facility at CERN [16] is expected to generate petabytes of experimental data every year, for each experiment. In addition to high-energy nuclear physics [9,38,17], a few other e-science applications are radio astronomy [31], geoscience [20], and climate studies [19]. In order to best support the needs of e-science applications, optical research networks are deployed by a consortium of leading research universities, governments and private sector tech-

nology companies. Examples include the Internet2-related [30] National Lambda Rail [36] and Abilene [1] networks in the U.S., CA*net4 [15] in Canada, and SURFnet [42] in Netherlands. These networks are small in size (less than 10^3 in the backbone) as compared to the public Internet. This makes it possible to have a centralized network controller for managing the network resources and for providing user service quality guarantee. With the central controller, there is more flexibility in designing sophisticated, efficient algorithms for scheduling user reservation requests, setting up network paths, and allocating bandwidth.

The objective of this paper is to develop and evaluate control plane algorithms for scheduling large file transfers (also known as jobs) over optical research networks. We assume that job requests are made in advance to the central network controller. Each request specifies a start time, an end time and the total file (demand) size. Such a request is satisfied as long as the network begins the transfer after the start time and completes it before the end time. The network controller has the flexibility in deciding the manner in which each file is transferred, i.e., how the bandwidth assignment to each job varies over time on all its allowed paths. The decision process is known as *scheduling*. We call this scheduling problem the *concurrent file transfer problem* (CFTP).

^{*} This work was supported in part by the National Science Foundation (NSF) under Grant ITR 0325459 and 0427110. Any findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of NSF. The authors would like to thank Rick Cavanaugh and Paul Avery for several discussions and insights.

^{*} Corresponding author. Phone: 352-392-2714 Fax:352-392-2714

Email addresses: krajah@cise.ufl.edu (Kannan Rajah), ranka@cise.ufl.edu (Sanjay Ranka), yx1@cise.ufl.edu (Ye Xia).

The current research networks generally use routers over optical transmission technologies instead of optical switches. Routers can split or aggregate traffic before putting it into the wavelength channels or the optical Ethernet links. Hence, the problem in this paper is routing and fine-grained bandwidth assignment rather than wavelength assignment, as would be the case in a wavelength-based circuit-switched optical network. It is possible to reserve an end-to-end wavelength path in the current research networks. But, our formulation of the bandwidth assignment problem will be unaffected since we can simply remove the reserved wavelength from the link capacity. We defer the wavelength assignment problem in an all optical network to future research.

We will formulate CFTP as a special type of the multi-commodity flow problem, known as the maximum concurrent flow (MCF) problem [41,26]. While MCF is concerned with allocating bandwidth to persistent concurrent flows, CFTP has to cope with the start and end time constraints of the jobs. For this purpose, our formulations for CFTP involve dividing time into uniform time slices and allocating bandwidth to each job on every time slice. Such a setup allows an easy representation of the start and end time constraints, by setting the allocated bandwidth of a job to zero before the start time and after the end time. More importantly, in between the start and end times, the bandwidth allocated for each job is allowed to vary from time slice to time slice. This enables periodical adjustment of the bandwidth assignment to the jobs so as to improve some performance objective.

Motivated by the MCF problem, the chosen objective is the throughput of the concurrent transfers. For fixed traffic demand, it is well known that such an objective is equivalent to minimizing the worst-case link congestion, a form of network load balancing [41]. A balanced traffic load enables the network to accept more future job requests, and hence, achieve higher long-term resource utilization. We assume that the optical network contains enough IP routers for traffic grooming, which is true for current research networks. Such a network allows fine-grained multiplexing of traffic for better network resource utilization.

In addition to the problem formulation, other contributions of this paper are as follows. First, in scheduling file transfers over multiple time slices, we focus on the tradeoff between achievable throughput and the required computation time. Second, we investigate using multiple paths for each file transfer to improve the throughput. We will show that using a small number of paths per job is generally sufficient to achieve near optimal throughput, and this is shown to be significantly higher than the throughput of a simple scheme that uses single shortest path. In addition, the computation time for the formulation with a small number of paths is considerably shorter than that for the optimal scheme, which utilizes all possible paths for each job.

The rest of the paper is organized as follows. In Section 2, we describe the CFTP and introduce the uniform time slice structure. In Section 3, we formally describe the node-arc and edge-path formulations of CFTP. The latter includes the k -shortest paths and k -shortest disjoint paths variants. In Section 4, we evaluate the algorithm performance for different formulations.

In Section 5, we introduce related work. The conclusions are drawn in Section 6.

2. The Concurrent File Transfer Problem (CFTP)

2.1. Problem Definition

A network is represented as a directed graph $G = (V, E)$ where V is the set of nodes and E is the set of edges (or arcs). Each edge $e \in E$ represents a link whose capacity is denoted by C_e . A path p is understood as a collection of links with no cycles. Job requests are submitted to the network using a 6-tuple representation $(A_i, s_i, d_i, D_i, S_i, E_i)$, where A_i is the arrival time of the request, s_i and d_i are source and destination nodes, respectively, D_i is the size of the file, S_i and E_i are requested start service time and end service time, where $A_i \leq S_i \leq E_i$. The meaning of the 6-tuple is: Request i is made at time $t = A_i$, asking the network to transfer a file of size D_i from source node s_i to destination node d_i over the time interval $[S_i, E_i]$.

In our framework, the network resource is managed by a central network controller. File transfer requests arrive following a random process and are submitted to the network controller. The network controller verifies admissibility of the jobs through a process known as *admission control* (AC). Admitted jobs are thereafter scheduled with a guarantee of the start and end time constraints. The details of how the AC and scheduling algorithms work together are described in [39]. In this paper, we focus on the scheduling problem alone at a single scheduling instance and compare different variations of the algorithm.

More specifically, we have the following scheduling problem. At a scheduling instance t , we have a network $G = (V, E)$ and the link capacity vector $C = (C_e)_{e \in E}$. The network may have some on-going file transfers; it may also have some jobs that were admitted earlier but yet to be started. The capacity C is understood as the *remaining* capacity, obtained by removing the bandwidth committed to all unfinished jobs admitted prior to t . The network controller has a collection of *new* job requests, denoted by J ¹. The task of the network controller is to schedule the transfer of the jobs in J so as to optimize a network efficiency measure. The chosen measure, which will be further explained later, is the value Z such that, if the demands are all scaled by Z (i.e., from D_i to ZD_i for every job i), they can be carried by the network without exceeding any link capacity. Such a Z is known as the throughput.

The solution to this scheduling problem will be a building block of the periodic AC and scheduling algorithm in [39]. AC and scheduling are done after every τ time units, where τ is a positive number. More specifically, at time instances $k\tau$, $k = 1, 2, \dots$, the network controller collects all the new requests that arrived on the interval $[(k-1)\tau, k\tau]$, makes the admission control decision, and schedules the transfer of all admitted jobs. Both AC and scheduling must take into account the old jobs, i.e., those jobs that were admitted earlier but remain

¹ We no longer need to consider the request arrival times, A_i , for $i \in J$. We may take $A_i = t$ for $i \in J$.

unfinished. The value of τ should be small enough so that new job requests can be checked for admission and scheduled as early as possible². However, τ should be more than the computation time required for AC and scheduling.

2.2. The Time Slice Structure

At any scheduling time t , the timeline from t onward is divided into uniform time slices (intervals). The set of time slices starting from time t is denoted as \mathcal{G}_t . The bandwidth assignment to each job is done on every time slice. In other words, the bandwidth reserved for a job remains constant throughout the time slice, but it can vary across time slices. At the scheduling time t , let the time slices in \mathcal{G}_t be indexed as $1, 2, \dots$ in increasing order of time. Let the start and end time of slice i be denoted by $ST_t(i)$ and $ET_t(i)$, respectively, and let its length be $LEN_t(i)$. We say a time instance $t' > t$ falls into slice i if $ST_t(i) < t' \leq ET_t(i)$. The index of the slice that t' falls in is denoted by $I_t(t')$.

The time slice structure is useful for bulk file transfers, wherein a request is satisfied as long as the network transfers the entire file between the start and end time. Such jobs offer a high degree of flexibility to the network in modulating the bandwidth assignment across time slices. This is in contrast to applications that require minimum bandwidth guarantee, for which the network must maintain the minimum required bandwidth from the start to the end time.

2.2.1. Rounding of the Start and End Time

While working with the time slice structure, the start and end time of the jobs should be adjusted to align on the slice boundaries. This is required because bandwidth assignment is done on a slice level. To illustrate, consider a file request $(A_i, s_i, d_i, D_i, S_i, E_i)$. Let the rounded start and end time be denoted as \hat{S}_i and \hat{E}_i , respectively. We round the requested start time S_i to be the maximum of the current time or the end time of the slice in which S_i falls, i.e.,

$$\hat{S}_i = \max\{t, ET_t(I_t(S_i))\}. \quad (1)$$

For rounding of the requested end time, we follow a *stringent policy* wherein the end time is rounded down, subject to the constraint that $\hat{E}_i > \hat{S}_i$. That is, there has to be at least one-slice separation between the rounded start and end time. Otherwise, there is no way to schedule the job. More specifically,

$$\hat{E}_i = \begin{cases} ET_t(I_t(\hat{S}_i) + 1) & \text{if } ST_t(I_t(E_i)) \leq \hat{S}_i \\ E_i & \text{else if } ET_t(I_t(E_i)) = E_i \\ ST_t(I_t(E_i)) & \text{otherwise.} \end{cases} \quad (2)$$

Fig. 1 shows several rounding examples. In practice, several variations of this rounding policy can be adopted based on the

² In this scheme, a request generally needs to wait for no longer than τ time units for the admission decision. It is also possible to perform admission control and scheduling in real time by examining if the remaining network bandwidth is sufficient when the request arrives. But, periodically, all jobs in the system are re-scheduled to make better use of the network bandwidth.

preference of the network manager. The subsequent problem formulations and solutions do not depend on the variation that is chosen.

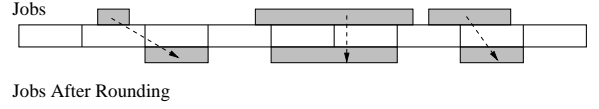


Fig. 1. Examples of stringent rounding. The unshaded rectangles are time slices. The shaded rectangles represent jobs. The top ones show the requested start and end times. The bottom ones show rounded start and end times.

From the definition of uniform slices, the slice set anchored at t , \mathcal{G}_t , contains infinitely many slices. In general, only a finite subset of \mathcal{G}_t is useful to us. Let M_t be the index of last slice in which the rounded end time of some job falls. That is, $M_t = I_t(\max_{i \in \mathcal{J}} \hat{E}_i)$. Let $\mathcal{L}_t \subset \mathcal{G}_t$ be the collection of time slices $\{1, 2, \dots, M_t\}$. It is sufficient to consider \mathcal{L}_t for scheduling.

3. Formulations

The maximum concurrent file transfer problem is formulated as a special type of network linear programs (LP), known as the maximum concurrent flow problem (MCF) [41,26]. We consider both the *node-arc* form and the *edge-path* form of the problem.

3.1. Node-Arc Form

Let $f_{(l,k)}^i(j)$ be the total amount of data transfer on link $(l, k) \in E$ that is assigned to job $i \in J$ on the time slice $j \in \mathcal{L}_t$. We will loosely call it the *flow* for job i on arc (l, k) on time slice j . Our objective is to determine the variables, $f_{(l,k)}^i(j)$. Once determined, they tell how much data will be transferred for job i on link (l, k) on time slice j , for all jobs, links and time slices.

Node-Arc(t, J)

$$\max \quad Z \quad (3)$$

$$\text{subject to} \quad \sum_{k:(l,k) \in E} f_{(l,k)}^i(j) - \sum_{k:(k,l) \in E} f_{(k,l)}^i(j) = \begin{cases} y^i(j) & \text{if } l = s_i \\ -y^i(j) & \text{if } l = d_i \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

$$\forall i \in J, \forall l \in V, \forall j \in \mathcal{L}_t$$

$$\sum_{j=1}^{M_t} y^i(j) = Z D_i \quad \forall i \in J \quad (5)$$

$$\sum_{i \in J} f_{(l,k)}^i(j) \leq C_{(l,k)}(j) LEN_t(j), \quad \forall (l, k) \in E, \forall j \in \mathcal{L}_t \quad (6)$$

$$f_{(l,k)}^i(j) = 0, \quad j \leq I_t(\hat{S}_i) \text{ or } j > I_t(\hat{E}_i), \quad \forall i \in J, \forall (l, k) \in E \quad (7)$$

$$f_{(l,k)}^i(j) \geq 0, \quad \forall i \in J, \forall j \in \mathcal{L}_t, \forall (l, k) \in E. \quad (8)$$

Condition (4) is the flow conservation equation that is required to hold on every time slice $j \in \mathcal{L}_t$. It says that, for each job i , if node l is neither the source node for job i nor its destination, then the total flow of job i that enters node l must be equal to the total flow of job i that leaves node l . Moreover, on each time slice, the supply of job i from its source must be equal to the demand at job i 's destination. This common quantity is denoted by $y^i(j)$ for job i on time slice j . Condition (5) says that, for each job, the total supply (or, equivalently, total demand), when summed over all time slices, must be equal to Z times the job size, where Z is the variable to be maximized. Condition (6) says that the capacity constraints must be satisfied for all edges on every time slice. Note that the allocated rate on link (l, k) for job i on slice j is $f_{(l,k)}^i(j)/LEN_t(j)$, where $LEN_t(j)$ is the length of slice j . The rate is assumed to be constant on the entire slice. Here, $C_{(l,k)}(j)$ is the capacity of link (l, k) on slice j . In all the experiments in this paper, each link capacity is assumed to be a constant across the time slices, i.e., $C_{(l,k)}(j) = C_{(l,k)}$ for all j . But, the formulation allows the more general time-varying link capacity. (7) is the start and end time constraint for every job on every link. The flow must be zero before the rounded start time and after the rounded end time.

The linear program asks: What is the largest constant scaling factor \hat{Z} such that, after every job size is scaled by \hat{Z} , the link capacity constraints, as well as the start and end time constraints, are still satisfied for all time slices? Let the optimal flow vector for the linear program be denoted by $\hat{f} = (\hat{f}_{(l,k)}^i(j))_{i,l,k,j}$. If $\hat{Z} \geq 1$, then the flow $\hat{Z}\hat{f}$ can still be handled by the network without the link capacity constraints being violated. If, in practice, the flow vector $\hat{Z}\hat{f}$ is used instead of \hat{f} , the file transfer can be completed faster. If $\hat{Z} < 1$, it is not possible to satisfy the deadline of all the jobs. However, if the file sizes are reduced by a common factor $\hat{Z}D_i$ for all i , then the requests can all be satisfied.

There exists a different perspective to our optimization objective. Maximizing the throughput of the concurrent flow is equivalent to finding a concurrent flow that carries all the demands and also minimizes the worst-case link utilization, i.e., link congestion. To see this, we make the following substitution of (vector) variables in (3)-(8): $\tilde{f} = f/Z$ and $\tilde{y} = y/Z$. Then, (6) becomes,

$$\frac{\sum_{i \in J} \tilde{f}_{(l,k)}^i(j)}{C_{(l,k)}(j)LEN_t(j)} \leq \frac{1}{Z}, \quad \forall (l, k) \in E, \forall j \in \mathcal{L}_t.$$

The left hand above is the link utilization at link (l, k) on time slice j . Hence, by maximizing Z , or equivalently minimizing $1/Z$, the largest link utilization over all links and across all time slices is minimized. The result is that the traffic load is balanced over the whole network and across all time slices. This feature is desirable if the network also carries other types of traffic that is sensitive to network load bursts, such as real-time traffic or traffic requiring minimum bandwidth guarantee. In addition, by reserving only the minimum bandwidth in each time slice, more future requests can potentially be accommodated.

The problem formulated here is related to the MCF problem.

The difference is that, in the MCF problem, the time dimension does not exist. Our problem becomes exactly the MCF problem if $M_t = 1$ (i.e., there is only one time slice) and if the constraints for the start and end times of the jobs, (7), are removed. In the MCF problem, the variable Z is called the *throughput* of the concurrent flow. The MCF problem has been studied in a sequence of papers, e.g., [41,26,24,3,4]. Several approximation algorithms have been proposed, which run faster than the usual simplex or interior point methods. For our problem, we can replicate the graph G into a sequence of temporal graphs representing the network at different time slices and use virtual source and destination nodes to connect them. We then have an MCF problem on the new graph and we can apply the fast approximation algorithms to this MCF instance.

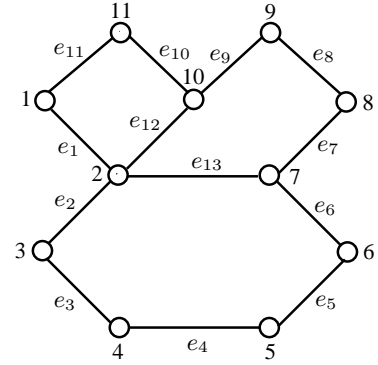


Fig. 2. A network with 11 nodes and 13 bi-directional links, each of capacity 1GB shared in both directions.

Example-1: Consider the network shown in Fig. 2 with two file transfer requests, $J_1 : (0, 1, 9, 8000, 0, 60)$ and $J_2 : (0, 3, 6, 1000, 0, 60)$. Here, we have used our 6-tuple convention to represent the requests. Both jobs requests arrive at time 0. The start and end times are both at $t = 0$ and $t = 60$, respectively. The job size is measured in GB and the time in minutes. When we schedule using a single slice of length 60 minutes, the node-arc formulation gives the following flow reservation for each job on edges e_1 through e_{13} .

$$J_1 : \{3600, 0, 0, 0, 0, 0, 3600, 3600, 3600, 3600, 3600, 0, 3600\}$$

$$J_2 : \{0, 0, 900, 900, 900, 0, 0, 0, 0, 0, 0, 0, 0\}$$

The throughput Z is 0.9, which is optimal.

The number of variables required to solve the node-arc model is $\Theta(|E| \times |\mathcal{L}_t| \times |J|)$, because, for every job, there is an arc flow variable associated with every link for every time slice. The resulting problem is computationally expensive even with the fast approximation algorithms. In Section 3.2, we will consider the edge-path form of the problem, where every job is associated with a set of path-flow variables corresponding to a small number of paths, for every time slice.

3.2. Edge-Path Form

The edge-path formulation uses a set of simple paths for each $i \in J$ and determines the flow on each of these paths on every time slice. The number of possible simple paths can actually be higher than the number of arcs and therefore the

edge-path form has no computational advantage over the node-arc form. To avoid the computational complexity, we consider sub-optimal formulations where we allow only a small number of paths for each job. In such a setting, the edge-path form is an appropriate formulation.

Let $P_t(s_i, d_i)$ be the set of allowed paths for job i (from the source node s_i to the destination d_i). Let $f_p^i(j)$ be the total amount of data transfer on path $p \in P_t(s_i, d_i)$ that is assigned to job $i \in J$ on the time slice $j \in \mathcal{L}_t$. We will loosely call it the *flow* for job i on path p on time slice j .

Edge-Path(t, J)

$$\max \quad Z \quad (9)$$

$$\text{subject to } \sum_{j=1}^{M_t} \sum_{p \in P_t(s_i, d_i)} f_p^i(j) = ZD_i, \quad \forall i \in J \quad (10)$$

$$\sum_{i \in J} \sum_{\substack{p \in P_t(s_i, d_i) \\ p: e \in p}} f_p^i(j) \leq C_e(j)LEN_t(j), \quad \forall e \in E, \forall j \in \mathcal{L}_t \quad (11)$$

$$f_p^i(j) = 0, \quad j \leq I_t(\hat{S}_i) \text{ or } j > I_t(\hat{E}_i), \quad (12)$$

$$\forall i \in J, \forall p \in P_t(s_i, d_i) \quad (13)$$

$$f_p^i(j) \geq 0, \quad \forall i \in J, \forall j \in \mathcal{L}_t, \forall p \in P_t(s_i, d_i). \quad (14)$$

Condition (10) says that, for every job, the sum of all the flows assigned on all time slices for all allowed paths must be equal to Z times the job size, where Z is the variable to be maximized. (11) says that the capacity constraints must be satisfied for all edges on every time slice. Note that the allocated rate on path p for job i on slice j is $f_p^i(j)/LEN_t(j)$, where $LEN_t(j)$ is the length of slice j . $C_e(j)$ is the capacity of link e on slice j . (13) is the start and end time constraint for every job on every allowed path. The flow must be zero before the rounded start time and after the rounded end time.

The edge-path formulation allows an explicitly defined collection of paths for each file-transfer job and flow reservations are done only on these paths. The number of variables required to solve the edge-path model is $\Theta(k \times |\mathcal{L}_t| \times |J|)$, where k is the maximum number of paths allowed for each job. We will examine two possible collections of paths, *k-shortest paths* and *k-shortest disjoint paths*.

3.2.1. *k-shortest paths*

We use the algorithm in [46] to generate k -shortest paths. This algorithm is not the fastest one, but is easy to implement. Also, in Section 3.2.2, we will use it as a building block in our algorithm for finding k -shortest disjoint paths. The key steps of the k -shortest-path algorithm are:

- (i) Compute the shortest path using Dijkstra's algorithm. This path is called the i^{th} shortest path for $i = 1$. Set $B = \emptyset$.
- (ii) Generate all possible deviations to the i^{th} shortest path and add them to B . Pick the shortest path from B as the $(i + 1)^{th}$ shortest path.
- (iii) Repeat step 2) until k paths are generated or there are no more paths possible (i.e., $B = \emptyset$).

Given a sequence of paths p_1, p_2, \dots, p_k from node s to d , the *deviation to p_k at its j^{th} node* is defined as a new path, p , which is the shortest path under the following constraint. First, p overlaps with p_k up to the j^{th} node, but the $(j + 1)^{th}$ node of p cannot be the $(j + 1)^{th}$ node of p_k . In addition, if p also overlaps with p_l up to the j^{th} node, for any $l = 1, 2, \dots, k - 1$, then the $(j + 1)^{th}$ node of p cannot be the $(j + 1)^{th}$ node of p_l . **Example-2:** Let us apply the edge-path formulation with k -shortest paths to the file transfer requests in Example-1 for the network shown in Fig. 2. The case of $k = 1$ corresponds to using the single shortest path for each job. Let p_j^i denote the j^{th} shortest path for job i . The shortest paths are:

$$p_1^1 : 1 - 11 - 10 - 9 \quad p_1^2 : 3 - 2 - 7 - 6$$

Flow reservation for each job is given by:

$$f_{p_1^1}^1(1) = 3600 \quad f_{p_1^2}^2(1) = 450$$

The throughput is 0.45, which is only half the optimal value obtained from the node-arc formulation.

For the case $k = 2$, i.e., with two shortest paths per job, we have,

$$p_1^1 : 1 - 11 - 10 - 9 \quad p_1^2 : 3 - 2 - 7 - 6$$

$$p_2^1 : 1 - 2 - 10 - 9 \quad p_2^2 : 3 - 4 - 5 - 6$$

$$f_{p_1^1}^1(1) = 3600 \quad f_{p_1^2}^2(1) = 450$$

$$f_{p_2^1}^1(1) = 0 \quad f_{p_2^2}^2(1) = 0$$

The total flow for J_1 is $f_{p_1^1}^1(1) + f_{p_2^1}^1(1) = 3600$. The total flow for J_2 is $f_{p_1^2}^2(1) + f_{p_2^2}^2(1) = 450$. The throughput is 0.45.

From $k = 1$ to 2, we do not find any throughput improvement. This is because for J_1 , the second path shares an edge with the first, and hence, the total flow reaching the destination node is limited to 3600. By increasing the number of paths per job from 2 to 4, we get the following results.

$$p_1^1 : 1 - 11 - 10 - 9 \quad p_1^2 : 3 - 2 - 7 - 6$$

$$p_2^1 : 1 - 2 - 10 - 9 \quad p_2^2 : 3 - 4 - 5 - 6$$

$$p_3^1 : 1 - 2 - 7 - 8 - 9 \quad p_3^2 : 3 - 2 - 10 - 9 - 8 - 7 - 6$$

$$p_4^1 : 1 - 11 - 10 - 2 - 7 - 8 - 9$$

$$p_4^2 : 3 - 2 - 1 - 11 - 10 - 9 - 8 - 7 - 6$$

$$f_{p_1^1}^1(1) = 3600 \quad f_{p_1^2}^2(1) = 0$$

$$f_{p_2^1}^1(1) = 0 \quad f_{p_2^2}^2(1) = 900$$

$$f_{p_3^1}^1(1) = 3600 \quad f_{p_3^2}^2(1) = 0$$

$$f_{p_4^1}^1(1) = 0 \quad f_{p_4^2}^2(1) = 0$$

The total flow for J_1 is 7200; the total flow for J_2 is 900. The throughput is 0.9. This is equal to the optimal value achieved by the node-arc formulation.

3.2.2. *k-shortest disjoint paths*

One interesting aspect that we noticed in Example-2 is that, while the k -shortest path algorithm minimizes the number of links used, the k -shortest paths for each job have a tendency

to overlap on some links. As a result, addition of new paths do not necessarily improve the throughput. This motivates us to consider the k -shortest disjoint paths.

In this paper, we use the term, *k-shortest disjoint paths*, as a short hand for a better term, *k-successive shortest edge-disjoint paths between a node pair*, which has been used quite widely in the networking literature, especially for network failure recovery (e.g., [18]). The latter term is defined algorithmically by repeating the following steps: find the shortest path, remove the edges on the shortest path from the network, and then find the next shortest path on the remaining network. The algorithm may take additional rules to break ties when multiple shortest paths exist in each step. In our case, it is quite common that the above algorithm stops before k paths can be found, due to the small network size or small node degree. Additional steps are needed to eventually find k paths, which will be outlined subsequently. In this case, the resulting k paths are not necessarily disjoint from each other. There is another use of the term, *k-shortest edge-disjoint paths*, in the literature, which are the k edge-disjoint paths whose aggregate length is no greater than any other set of k edge-disjoint paths [43]. But, this definition still does not deal with the cases when there aren't k disjoint paths. In principle, the real optimization question is to find k or less paths for every node pair so that the worst-case link congestion is minimized. However, even simpler-looking problems than this are extremely hard [32]. We believe that many researchers will continue to investigate this kind of path selection problems.

The algorithm for finding the k -shortest disjoint paths from node s to d is as follows. Given the directed graph G , in the first step of the algorithm, we find the shortest path from node s to d , and then we remove all the edges on the path from the graph G . In the next step, we find the shortest path in the remaining graph, and then remove those edges on the selected path to create a new remaining graph. The algorithm continues until we find k paths or until it is not possible to find more paths, whichever comes first. When the number of disjoint paths selected is less than k , we resort to the following heuristics to select additional paths so that the total number of selected paths is k . Let S be the list of selected disjoint paths.

- (i) Set S to be an empty list. Set $B = \emptyset$.
- (ii) Find all the disjoint paths between the source s and destination d and append them to S in the order they are found. Let p be the first path in the list S .
- (iii) Generate the deviations for p and add them to B .
- (iv) Select the path in B that has the least number of overlapped edges with the paths in S , and append it to S .
- (v) Set p to be the next path in the list S .
- (vi) Repeat from step 3) until S contains k paths or there are no more paths possible (i.e., $B = \emptyset$).

In the above steps, the set B contains short paths, generated from the deviations of some already selected disjoint paths. The newly selected path from B has the least overlap with the already selected ones. It should be noted that while this approach reduces the overlap between the k paths of each job, it does not guarantee the same for paths across jobs. This is because, the average path length of k -shortest disjoint paths

tends to be greater than that of the k -shortest paths, potentially causing the shortest disjoint paths of one job to heavily overlap with those of other jobs. This can have a negative effect on the overall throughput.

Example-3: Let us apply the k -shortest disjoint paths to Example-1. For $k = 2$, we have,

$$\begin{array}{ll} p_1^1 : 1 - 11 - 10 - 9 & p_1^2 : 3 - 2 - 7 - 6 \\ p_2^1 : 1 - 2 - 7 - 8 - 9 & p_2^2 : 3 - 4 - 5 - 6 \\ f_{p_1^1}^1(1) = 3600 & f_{p_1^2}^2(1) = 450 \\ f_{p_2^1}^1(1) = 3600 & f_{p_2^2}^2(1) = 450 \end{array}$$

The total flow for J_1 is $f_{p_1^1}^1(1) + f_{p_2^1}^1(1) = 7200$. The total flow for J_2 is $f_{p_1^2}^2(1) + f_{p_2^2}^2(1) = 900$. The throughput is 0.9. Hence, the optimal throughput is achieved with $k = 2$.

4. Evaluation

This section shows the performance results of the edge-path formulation using the single and multi-path schemes. We compare its throughput with the optimal solution obtained from node-arc formulation. The scalability of the formulations are evaluated based on their required computation time.

The experiments were conducted on random networks and Abilene, an Internet2 high-performance backbone network (Fig.3). The random networks have between 100 and 1000 nodes with a varying node degree of 5 to 10. The network generator takes the number of nodes and the average node degree as arguments, from which it computes the total number of links in the network. Then, it repeatedly picks a node pair uniformly at random from those unconnected node pairs, and connects them with a pair of links in both directions. This process is repeated until all links are assigned. The speed of each link in a pair is drawn uniformly from 0.1 Gbps to 10 Gbps. Our instance of the Abilene network consists of a backbone with 11 nodes, in which each node is connected to a randomly generated stub network of average size 10. The backbone links are each 10GB. The speed of each link in a stub network is drawn uniformly at random from 0.1 Gbps to 10 Gbps. The entire network has 121 nodes and 490 links.

We use the commercial CPLEX package for solving linear programs on Intel-based workstations³. In order to simulate the file size distribution of Internet traffic, we resort to the widely accepted heavy-tailed Pareto distribution, with the distribution function $F(x) = 1 - (x/b)^{-\alpha}$, where $x \geq b$ and $\alpha > 1$. As α value gets closer to 1, the distribution becomes more heavy-tailed and there is a higher probability of generating large file sizes. All the experiments described in this section were done using Pareto parameter $\alpha = 1.8$ and an average job size of 50GB. The plots use the following acronyms: S (Shortest path), SD (Shortest Disjoint path) and NA (Node-Arc).

³ Since fast approximation algorithms are not the focus of this paper, we use the standard LP solver for the evaluations.

While configuring the simulation environment, we can ignore the connection setup (path setup for the edge-path form) time for the following reasons. First, the small network size allows us to pre-compute the allowed paths for every possible request. Second, in the actual operation, the scheduling algorithm runs every few minutes or every tens of minutes. There is plenty of time to re-configure the control parameters for the paths in the small research network.

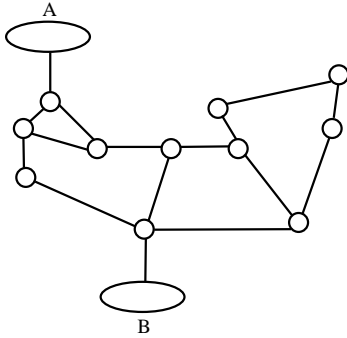


Fig. 3. The Abilene network with 11 backbone nodes. A and B are stub networks.

4.1. Single Slice Scheduling (SSS)

When $|\mathcal{L}_t| = 1$ in the node-arc and edge-path formulations, we call the situation *single slice scheduling* (SSS). In this experiment, we keep the time-slice structure simple in order to examine how other factors affect the performance of different formulations. All jobs start at the 0^{th} minute and end at 60^{th} minute. Scheduling is done at time 0 with a (single) time slice size equal to 60 minutes.

4.1.1. Performance Comparison of the Formulations

Fig. 4 shows the throughput improvement on the Abilene network with increasing number of paths for the shortest (S) and shortest disjoint (SD) schemes, respectively. The optimal throughput obtained from the node-arc (NA) form is shown as a horizontal line. Similar plots are shown in Fig. 5 for a random network with 100 nodes⁴.

4.1.1.1. Single v.s. Multiple Paths Moving from a single path to multiple paths per job, we observe a drastic throughput increase. A small number of paths per job is sufficient to realize such throughput improvement. On the Abilene network, the throughput is increased by up to 10 times with 4 to 8 paths per job. Simply by switching from a single path to two paths per job, we observe 60% throughput gain. On the random network, the throughput is increased by 10 to 30 times with 4 or more paths. In most of our examples, the S and SD schemes reach the optimal throughput with $k = 8$ or less.

⁴ The node-arc case is not shown in Fig. 5 (d) and in several subsequent figures because the problem size becomes too large to be solved on our workstations with 2 to 4 GB of memory, mainly due to the large memory requirement.

In summary, the optimal throughput obtained from our multi-path scheme is significantly higher than that of a simple scheme, which uses single shortest path for every job. Throughput improvement by an order of magnitude can be expected with only a small number of paths. The performance gains saturate at around 8 paths in most of our simulation - the exact number in general depends on the topology and actual traffic.

4.1.1.2. Shortest (S) v.s. Shortest Disjoint (SD) Paths For random networks, SD tends to perform better than S. In most of our examples, the throughput of SD is several times higher than that of S for $k = 2$ to 8. For the Abilene network, the opposite trend can often be observed. This behavior can be explained as follows. As we have mentioned in Section 3.2, the paths for *different* jobs have a higher chance to overlap in the SD case, potentially causing throughput degradation. In a well-connected random network, disjoint or nearly disjoint paths are more abundant and also tend to be short. The throughput benefit from the disjoint paths exceeds the throughput degradation from the longer average path length. On the other hand, in the Abilene network, the backbone network has few disjoint paths between each pair of nodes. Insisting on having (nearly) disjoint paths leads to longer average path length due to the lack of choices. Hence, the throughput penalty from longer path length is more pronounced in a small network such as Abilene. Therefore, it is often more beneficial to use the shortest paths instead.

In summary, we expect SD to be preferable in large, well-connected networks. In a small network with few disjoint paths, the performance of S and SD are generally comparable, with S sometimes being better. Finally, the difference between S and SD disappears quickly as the number of paths per job increase.

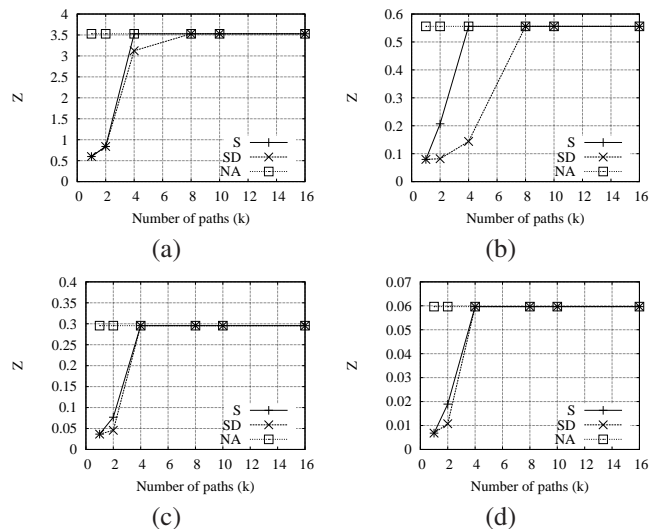


Fig. 4. Z for different formulations on Abilene network using SSS. (a) 121 jobs; (b) 605 jobs; (c) 1210 jobs; (d) 6050 jobs.

4.1.2. Comparison of Algorithm Execution Time

Recall that our motivation to move from the node-arc formulation to the edge-path formulation is that the latter allows us

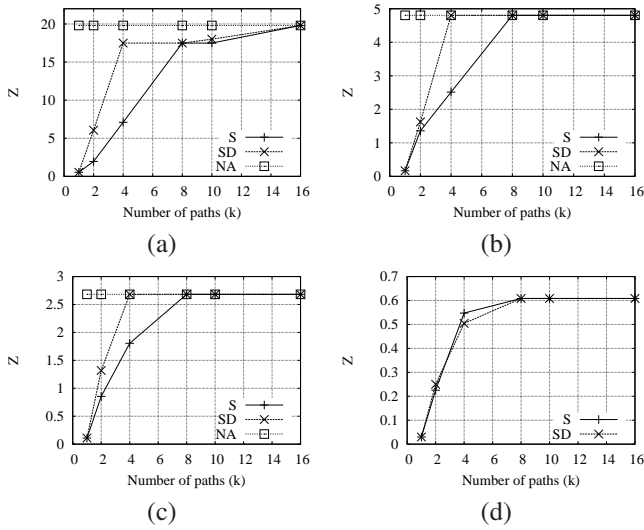


Fig. 5. Z for different formulations on a random network with 100 nodes using SSS. (a) 100 jobs; (b) 500 jobs; (c) 1000 jobs; (d) 5000 jobs.

to restrict the number of permitted paths for each job, resulting in lower algorithm execution time. Fig. 6 and Fig. 7 show the execution time for the Abilene network and for a random network with 100 nodes, respectively⁵. The horizontal axis is the number of selected paths for the shortest (S) and shortest disjoint (SD) cases. The execution time for the node-arc (NA) form is shown as a flat line.

We observe that the execution time for S or SD increases roughly linearly, when the number of permitted paths per job is small (up to 16 paths in the figures). With several hundred jobs or more, even the longest execution time (at 16 paths) is much shorter than that for the node-arc case, by an order of magnitude. We expect this difference in execution time to increase with more jobs and larger networks.

In Fig. 6 (c) and (d), we see that the scheduling time for the node-arc formulation approaches or exceeds the actual 60-minute transfer time of the files. On the other hand, the edge-path formulation with a small number of allowed paths, is much more scalable with traffic intensity. Fast approximation algorithms in [41,26,24,3,4], if used, should improve the execution time for all formulations. But, the significant difference between the node-arc case and the shortest or shortest disjoint cases should still remain.

4.1.3. Algorithm Scalability with Network Size

Fig. 8 shows the variation of the algorithm execution time with network size. In our simulations, we schedule 100 jobs using SSS for a period of 60 minutes. The edge-path algorithms (S and SD) with 8 paths have an execution time under 10 seconds for networks with less than 800 nodes. On the other hand, the execution time for the node-arc algorithm is nearly 15 minutes for a network size of 500 nodes. We conclude that the

⁵ Unless mentioned otherwise, the execution time for the edge-path formulations does not include the path computation time for finding the shortest paths. This is because the shortest paths are computed only once, and the computation can be carried out off-line.

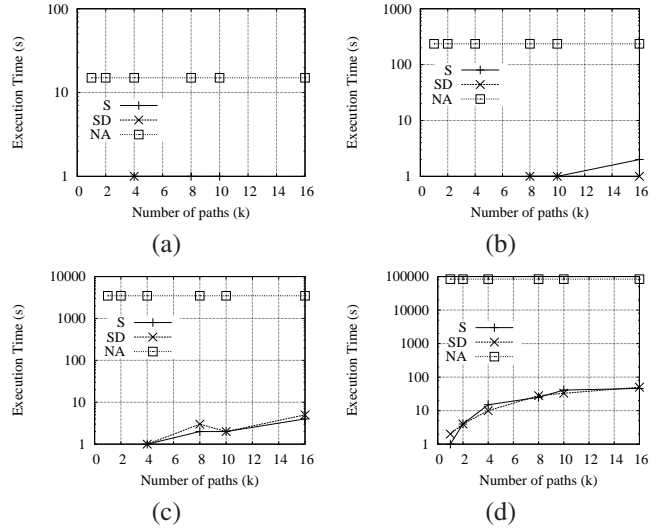


Fig. 6. Execution time for different formulations on the Abilene network using SSS. (a) 121 jobs; (b) 605 jobs; (c) 1210 jobs; (d) 6050 jobs.

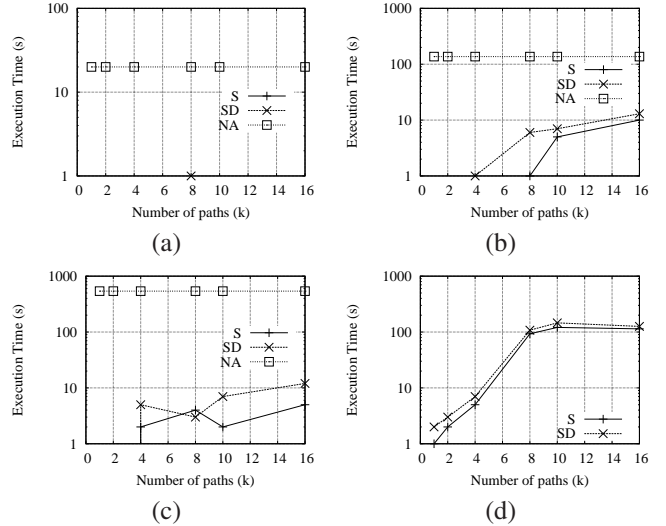


Fig. 7. Execution time for different formulations on a random network with 100 nodes using SSS. (a) 100 jobs; (b) 500 jobs; (c) 1000 jobs; (d) 5000 jobs.

node-arc formulation is unsuitable for real-time scheduling of file transfers on networks of more than several hundred nodes.

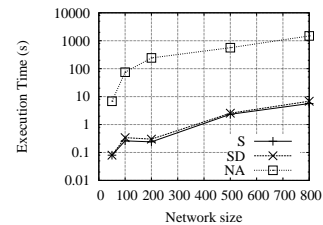


Fig. 8. Random network with $k = 8$. Execution time for different network sizes.

4.1.4. Average Results over Random Network Instances

When the experiments are conducted on random networks, unless mentioned otherwise, each plot typically presents the results obtained from a single network instance rather than an

average result over many network instances. To demonstrate that the single-instance results are not anomalies but representative, we repeated the experiments in Section 4.1 for a 100-node random network and plotted the data points averaged over 50 network instances. Due to space limitation, we present only the results for 1000 jobs in Fig. 9. This should be compared with Fig. 5 (c), which is for a single network instance. Besides the fact that the curves in Fig. 9 are smoother, the two figures show similar characteristics. All the observations that we have made about Fig. 5 (c) remain essentially true for Fig. 9. We should point out that, in order to run the experiment on many network instances in a reasonable amount of time, the networks for Fig. 9 were generated with fewer links than that for Fig. 5 (c). This accounts for the difference in the throughput values between the two cases. Finally, the corresponding average execution time is shown in Fig. 10 on semilog scale.

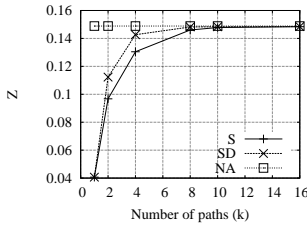


Fig. 9. Average Z for different formulations on a random network with 100 nodes and 1000 jobs using SSS. The result is the average over 50 instances of the random network.

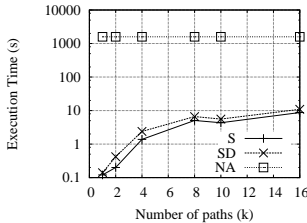


Fig. 10. Average execution time for different formulations on a random network with 100 nodes and 1000 jobs using SSS. The result is the average over 50 instances of the random network.

We further confirmed the validity of our data and results by computing the confidence interval of the mean values plotted in Fig. 9. For instance, the mean and standard deviation of the throughput for node-arc formulation is 0.1489 and 0.0807, respectively. The 95% confidence interval for the mean is ± 0.0188 around the mean. This is a good indicator of the accuracy of our results.

In addition, we also computed the average of the throughput ratio of S and SD schemes to the node-arc formulation. In Fig. 11, both S and SD schemes achieve nearly 80% of the optimal throughput by switching from single path to 2 paths. The throughput reaches 99% with 8 paths. For $k \leq 4$, SD performs better than S. The plot is consistent with our earlier results shown in Fig. 9.

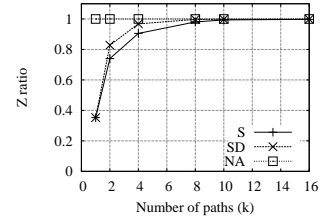


Fig. 11. Average throughput ratio for different formulations on a random network with 100 nodes and 1000 jobs using SSS. The result is the average over 50 instances of the random network.

4.2. Multiple Slice Scheduling (MSS)

When $|\mathcal{L}_t| > 1$ in the node-arc and edge-path formulations, we call the situation *multiple slice scheduling* (MSS). In this experiment, 121 jobs are scheduled for a period of 1 day using multiple slices of identical size. The interval between the start times of the jobs are independently and identically distributed exponential random variables with a mean of 1 minute. The requested end time of each job is set to be equal to its start time plus the desired transfer time, which is equal to the file size divided by the desired bandwidth, chosen to be 0.1 Gbps. We have tried four time-slice sizes: 60, 30, 15 and 10 minutes.

4.2.1. Performance Comparison of Different Formulations

Fig. 12 shows the throughput improvement for the Abilene network with increasing number of paths for the S and SD schemes, respectively. The throughput of the node-arc formulation is shown as a flat line.

For each fixed slice size, the general behavior of the throughput follows the same pattern as the SSS case discussed in Section 4.1.1. In particular, the throughput improvement is significant as the number of paths per job decreases. In Fig. 12, we observe more than 50% throughput increase with 4 or fewer paths and nearly 30% to 50% increase with 8 or more paths. When comparing across different slice sizes, we see that smaller slice sizes have a throughput advantage, because they lead to more accurate quantization of time. Having more time slices in a fixed scheduling interval offers more opportunities to adjust the flow assignment to the jobs. In Fig. 12, the throughput values at 16 paths per job is 9 for 10-min slice size and 6 for 60-min slice size. This shows the benefit of having a fine-grained slice size, since in this experimental setup, 16 paths are sufficient for S and SD schemes to reach the optimal throughput. We observed more significant throughput improvement from using smaller time slices in other settings. For instance, with 603 jobs, the throughput obtained from 10-min slice size is nearly twice the throughput from 60-min slice size.

Fig. 13 shows similar results for a 100-node random network with 100 jobs. The maximum throughput at 16 paths is nearly the same for all cases. However, for situations with a small number of paths per job, smaller time slice sizes have a throughput advantage. More throughput improvement has been observed under other experimental settings. For instance, with 500 jobs and 16 paths, a 24% improvement is observed when using 10-minute slices instead of 60-minute slices.

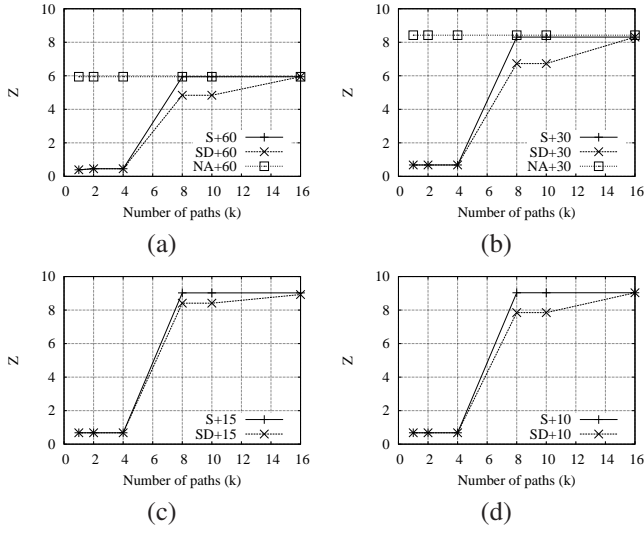


Fig. 12. Z for different formulations on the Abilene network with 121 jobs using MSS. (a) Time slice = 60 min; (b) Time slice = 30 min; (c) Time slice = 15 min; (d) Time slice = 10 min.

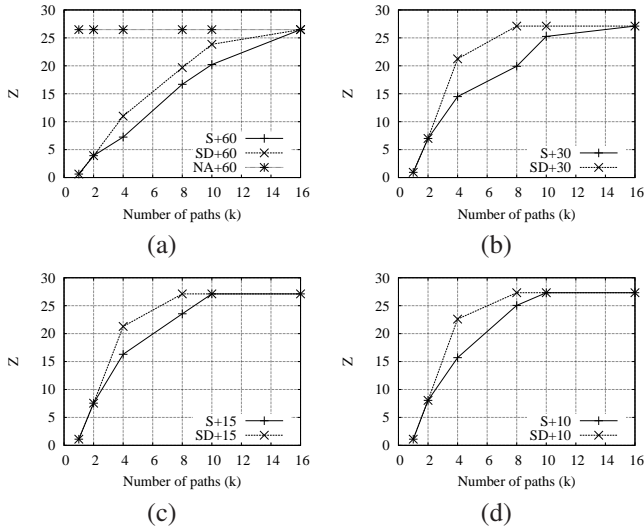


Fig. 13. Z for different algorithms on a 100-node random network with 100 jobs using MSS. (a) Time slice = 60 min; (b) Time slice = 30 min; (c) Time slice = 15 min; (d) Time slice = 10 min.

4.2.2. Comparison of Algorithm Execution Time

Fig. 14 and Fig. 15 show the execution time for the Abilene network with 121 jobs and for a 100-node random network with 100 jobs, respectively. For each fixed time slice size, we continue to observe the linear or faster increase of the execution time as the number of paths increase in the S and SD schemes. Again, the execution time for the node-arc form is much greater than that for the S and SD cases; in most cases, too large to be observed from our experiments. Finally, the throughput advantage of using smaller slice sizes is achieved at the expense of significant longer execution time.

4.2.3. Optimal Time Slice

The tradeoff of the three scheduling algorithms lies in two metrics, throughput and execution time. Fig. 16 helps to identify a suitable time slice size for which the throughput is high and

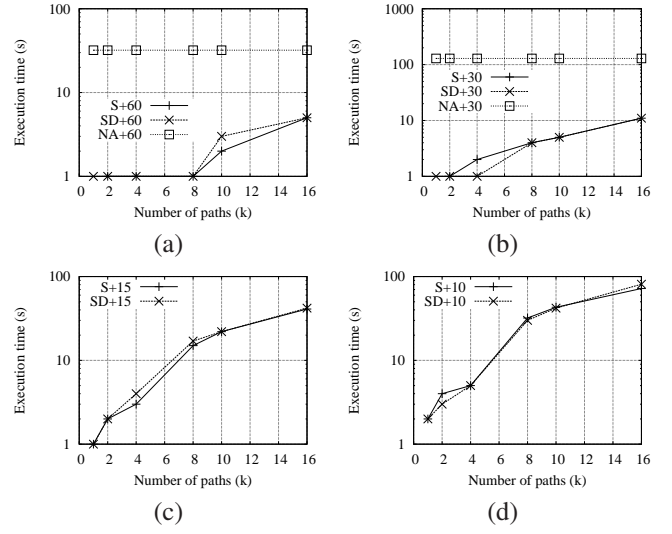


Fig. 14. Execution time for different formulations on the Abilene network with 121 jobs using MSS. (a) Time slice = 60 min; (b) Time slice = 30 min; (c) Time slice = 15 min; (d) Time slice = 10 min.

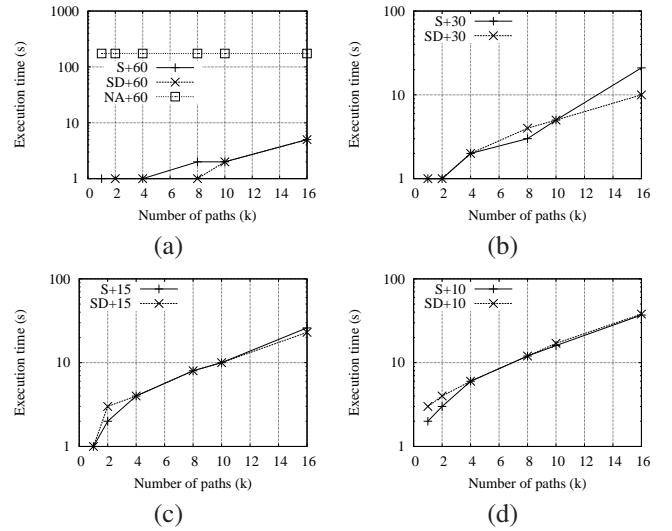


Fig. 15. Execution time for different formulations on a 100-node random network with 100 jobs using MSS. (a) Time slice = 60 min; (b) Time slice = 30 min; (c) Time slice = 15 min; (d) Time slice = 10 min.

the execution time is acceptable. We observe that the throughput begins to saturate when the time slice size is 15 minutes and the execution time is under half a minute. Note the sharp rise of the execution time as the slice size decreases. It is therefore essential to choose an appropriate slice size.

5. Related Work

Similar to this paper, the authors of [6] also advocate periodic re-optimization to determine new routes and bandwidth in optical networks. They also use a multi-commodity flow formulation. However, they do not assume users making advance reservations with requested start and end times. As a result, the scheduling problem is for a single time instance, rather than over multiple time slices. Furthermore, it does not consider the

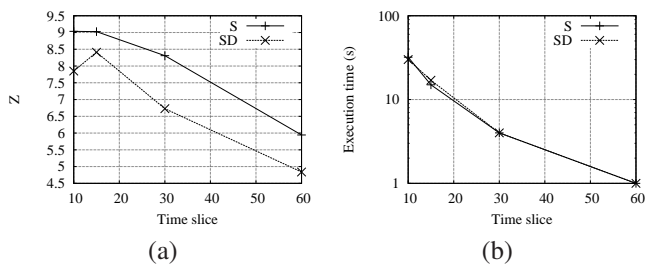


Fig. 16. The Abilene network with 121 jobs and $k = 8$. (a) Z for different time slices; (b) Execution time for different time slice sizes.

edge-path formulation with limited number of paths per job.

Several earlier studies [40,8,12,10,14,45,44] consider advance bandwidth reservation with start and end times at an individual link for traffic that requires minimum bandwidth guarantee (MBG). The concern is typically about designing efficient data structures for keeping track of and querying bandwidth usage at the link on different time intervals. New jobs are admitted one at a time without changing the bandwidth assignment of the existing jobs in the system. The admission of a new job is based on the availability of the requested bandwidth between its start time and end time. [27,44,13,10,22] and [14] all go beyond single-link advance reservation and tackle the more general path-finding problem for the MBG traffic class, but typically only for the new requests, one at a time. The routes and bandwidth of the existing jobs are unchanged. [11] discusses architectural and signaling-protocol issues about advance reservation of network resources. [34] considers a network with known routing in which each admitted job derives a profit. It gives approximation algorithms for admitting a subset of the jobs so as to maximize the total profit.

[27,13] touch upon advance reservation for bulk transfer. [13] proposes a malleable reservation scheme. The scheme checks every possible interval between the requested start time and end time for the job and tries to find a path that can accommodate the entire job on that interval. It favors intervals with earlier deadlines. [27] studies the computation complexity of a related path-finding problem and suggests an approximation algorithm. [35] starts with an advance reservation problem for bulk transfer. Then, the problem is converted into a bandwidth allocation problem at a single time instance to maximize the job acceptance rate. This is shown to be an NP-hard combinatorial problem. Heuristic algorithms are then proposed. Many papers study advance reservation, re-routing, or re-optimization of lightpaths, at the granularity of a wavelength, in WDM optical networks [47,5,7]. They are complementary to our study.

In the control plane, [29] and [28] present architectures for advance reservation of intra and interdomain lightpaths. The DRAGON project [33] develops control plane protocols for multi-domain traffic engineering and resource allocation on GMPLS-capable [21] optical networks. GARA [25], the reservation and allocation architecture for the grid computing toolkit, Globus, supports advance reservation of network and computing resources. [23] adapts GARA to support advance reservation of lightpaths, MPLS paths and DiffServ paths.

6. Conclusion

This study aims at contributing to the management and resource allocation of research networks for data-intensive e-science collaborations. The need for large file transfers is among the main challenges posed by such applications. The opportunities lie in the fact that research networks are generally much smaller in size than the public Internet, and hence, can afford a centralized resource management platform. In this paper, we formulate two linear programs, the node-arc form and edge-path form, for scheduling bulk file transfers with start and end time constraints. Our objective is to maximize the throughput, subject to the link capacity constraints. The throughput is a common scaling factor for all demand (file) sizes. This performance objective is equivalent to finding a transfer schedule that carries all the demands and also minimizes the worst-case link congestion across all links and time. It has the effect of balancing the traffic load over the whole network and across time. This feature enables the network to accept more future file transfer requests and in turn achieve higher long-term resource utilization.

An important contribution of this paper is towards the application of the edge-path formulation to obtaining close to optimal throughput with a reasonable time complexity. We have shown that the node-arc formulation, while giving the optimal throughput, is computationally very expensive. The edge-path formulation can lead to drastic reduction of the computation time by using a small number of pre-defined paths for each file-transfer job. We discussed two path selection schemes, the shortest paths (S) and the shortest disjoint paths (SD). Both schemes are capable of achieving near optimal throughput with a small number of paths, e.g. 8 or less, for each file-transfer request. Both S and SD perform well in a small network with few disjoint paths, e.g. the Abilene backbone, while SD performs better than S in larger, well connected networks. In the evaluation process, we also showed that having multiple paths per job yields much higher throughput than having one shortest path per job.

To handle the start and end time requirement of advance reservation, we divide time into uniform time slices in our formulations. The paper showed that using finer slices leads to significant throughput increase at the expense of longer execution time. It is therefore important to choose the right slice size that best balances such a tradeoff.

Throughout the paper, we have assumed that all jobs are admissible. In a real environment, admission control should be built into the scheduling system and performed before the scheduling phase. Our current research work [39] focuses on developing a joint admission control/scheduling framework that uses the formulations and algorithms described in this paper as its core components.

References

- [1] Abilene. <http://abilene.internet2.edu/>.

- [2] Paul Avery. Grid computing in high energy physics. In *Proceedings of the International Beauty 2003 Conference*, Pittsburgh, PA, Oct. 2003.
- [3] B. Awerbuch and F. T. Leighton. A simple local-control approximation algorithm for multicommodity flow. In *Proceedings of the IEEE Symposium on Theory of Computing*, pages 459–468, 1993.
- [4] B. Awerbuch and F. T. Leighton. Improved approximation algorithms for multi-commodity flow problem and local competitive routing in dynamic networks. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 487–496, 1994.
- [5] D. Banerjee and B. Mukherjee. Wavelength-routed optical networks: linear formulation, resource budgeting tradeoffs, and a reconfiguration study. *IEEE/ACM Transactions on Networking*, 8(5):598–607, Oct. 2000.
- [6] R. Bhatia, M. Kodialam, and T. V. Lakshman. Fast network re-optimization schemes for MPLS and optical networks. *Computer Networks*, 50(3), Feb. 2006.
- [7] E. Bouillet, J.-F. Labourdette, R. Ramamurthy, and S. Chaudhuri. Lightpath re-optimization in mesh optical networks. *IEEE/ACM Transactions on Networking*, 13(2):437–447, 2005.
- [8] Andrej Brodnik and Andreas Nilsson. A static data structure for discrete advance bandwidth reservations on the Internet. Technical Report Tech report cs.DS/0308041, Department of Computer Science and Electrical Engineering, Luleå University of Technology, Sweden, 2003.
- [9] J. Bunn and H. Newman. Data-intensive grids for high-energy physics. In F. Berman, G. Fox, and T. Hey, editors, *Grid Computing: Making the Global Infrastructure a Reality*. John Wiley & Sons, Inc, 2003.
- [10] Lars-O. Burchard. Source routing algorithms for networks with advance reservations. Technical Report Technical Report 2003-03, Communications and Operating Systems Group, Technical University of Berlin, 2003.
- [11] Lars-O. Burchard. Networks with advance reservations: applications, architecture, and performance. *Journal of Network and Systems Management*, 13(4):429–449, Dec. 2005.
- [12] Lars-O. Burchard and Hans-U Heiss. Performance evaluation of data structures for admission control in bandwidth brokers. Technical Report Technical Report TR-KBS-01-02, Communications and Operating Systems Group, Technical University of Berlin, 2002.
- [13] Lars-O. Burchard and Hans-U. Heiss. Performance issues of bandwidth reservation for grid computing. In *Proceedings of the 15th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'03)*, 2003.
- [14] Lars-O. Burchard, J. Schneider, and B. Linnert. Rerouting strategies for networks with advance reservations. In *Proceedings of the First IEEE International Conference on e-Science and Grid Computing (e-Science 2005)*, Melbourne, Australia, Dec. 2005.
- [15] CA*net4. <http://www.canarie.ca/canet4/index.html>.
- [16] CERN. <http://www.cern.ch>.
- [17] Compact Muon Solenoid (CMS). <http://cms.cern.ch/>.
- [18] D. A. Dunn, W. D. Grover, and M. H. MacGregor. Comparison of k-shortest paths and maximum flow routing for network facility restoration. *IEEE Journal on Selected Areas in Communications*, 12(1):88–99, 1994.
- [19] The Earth System Grid (ESG). <http://www.earthsystemgrid.org/>.
- [20] EarthScope. http://www.earthscope.org/usarray/data_flow/archiving.php.
- [21] E. Mannie (Ed.). *Generalized multi-protocol label switching (GMPLS) architecture*. RFC 3945, IETF, Oct. 2004.
- [22] T. Erlebach. Call admission control for advance reservation requests with alternatives. Technical Report TIK-Report Nr. 142, Computer Engineering and Networks Laboratory, Swiss Federal Institute of Technology (ETH) Zurich, 2002.
- [23] C. Curti et. al. On advance reservation of heterogeneous network paths. *Future Generation Computer Systems*, 21(4):525–538, Apr. 2005.
- [24] L. K. Fleischer. Approximating fractional multicommodity flow independent of the number of commodities. *Siam Journal of Discrete Mathematics*, 13(4):505–520, 2000.
- [25] I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, and A. Roy. A distributed resource management architecture that supports advance reservations and co-allocation. In *Proceedings of the International Workshop on Quality of Service (IWQoS '99)*, 1999.
- [26] N. Garg and J. Könenmann. Faster and simpler algorithms for multi-commodity flow and other fractional packing problems. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, pages 300–309, November 1998.
- [27] R. Guerin and A. Orda. Networks with advance reservations: The routing perspective. In *Proceedings of IEEE INFOCOM 99*, 1999.
- [28] E. He, X. Wang, and J. Leigh. A flexible advance reservation model for multi-domain WDM optical networks. In *Proceedings of GRIDNETS 2006*, San Jose, CA, 2006.
- [29] E. He, X. Wang, V. Vishwanath, and J. Leigh. AR-PIN/PDC: Flexible advance reservation of intradomain and interdomain lightpaths. In *Proceedings of the IEEE GLOBECOM 2006*, 2006.
- [30] Internet2. <http://www.internet2.edu>.
- [31] Joint Institute for Very Long Baseline Interferometry (JIVE). <http://www.jive.nl/>.
- [32] Jon Kleinberg. *Approximation Algorithms for Disjoint Paths Problems*. PhD thesis, Massachusetts Institute of Technology, 1996.
- [33] T. Lehman, J. Sobieski, and B. Jabbari. DRAGON: A framework for service provisioning in heterogeneous grid networks. *IEEE Communications Magazine*, March 2006.
- [34] L. Lewin-Eytan, J. Naor, and A. Orda. Routing and admission control in networks with advance reservation. In *Proceedings of the Fifth International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX 02)*, 2002.
- [35] L. Marchal, P. Vicat-Blanc Primet, Y. Robert, and J. Zeng. Scheduling network requests with transmission window. Technical Report 2005-32, LIP, ENS Lyon, France, 2005.
- [36] National Lambda Rail. <http://www.nlr.net>.
- [37] H. B. Newman, M. H. Ellisman, and J. A. Orcutt. Data-intensive e-science frontier research. *Communications of the ACM*, 46(11):68–77, Nov. 2003.
- [38] The Particle Physics Data Grid (PPDG). <http://www.ppdg.net/>.
- [39] Kannan Rajah, Sanjay Ranka, and Ye Xia. Advance reservation and scheduling of large tranfer in e-science. *Manuscript*.
- [40] O. Schelén, A. Nilsson, Joakim Norrgård, and S. Pink. Performance of QoS agents for provisioning network resources. In *Proceedings of IFIP Seventh International Workshop on Quality of Service (IWQoS'99)*, London, UK, June 1999.
- [41] Farhad Shahrokhi and D. W. Matula. The maximum concurrent flow problem. *Journal of the Association for Computing Machinery*, 37(2):318–334, April 1990.
- [42] SURFnet. <http://www.surfnet.nl/info/en>.
- [43] J. W. Suurballe. Disjoint paths in a network. *Networks*, 4(2):125–145, 1974.
- [44] Tao Wang and Jianer Chen. Bandwidth tree - A data structure for routing in networks with advanced reservations. In *Proceedings of the IEEE International Performance, Computing and Communications Conference (IPCCC 2002)*, April 2002.
- [45] Qing Xiong, Chanle Wu, Jianbing Xing, Libing Wu, and Huyin Zhang. A linked-list data structure for advance reservation admission control. In *ICCNMC 2005*, 2005. Lecture Notes in Computer Science, Volume 3619/2005.
- [46] Jin Y. Yen. Finding the k shortest loopless paths in a network. *Management Science*, 17(11):712–716, 1971.
- [47] Jun Zheng and Hussein T. Mouftah. Routing and wavelength assignment for advance reservation in wavelength-routed WDM optical networks. In *Proceedings of the IEEE International Conference on Communications (ICC)*, 2002.