Path Selection in User-Controlled Circuit-Switched Optical Networks

*

Wojciech Golab, Raouf Boutaba
University of Waterloo
{wgolab, rboutaba}@bbcr.uwaterloo.ca

Abstract

User-controlled circuit-switched optical networks are gaining popularity in an effort to fulfill the insatiable data transport needs of the online community. In this paper we consider the resource allocation challenges that arise in such networks, in particular problems related to construction of end-to-end lightpaths for carrying large multimedia streams. Specifically, we discuss variations of the least cost and widest path problems that address two unique aspects of the user-controlled environment. First, since network resources are exposed for user-control using a service-oriented software control plane, each lightpath is subject to an expiry time. Second, because Wavelength Division Multiplexing (WDM) and resource partitioning introduces multiple redundant paths, classic least cost path computations tend to yield multiple optimal solutions, and so it is useful to break ties among these in a judicious manner. We present polynomial-time path selection techniques that address these issues using efficient data structures. We also show the benefit of breaking ties in shortest path computations in a manner that reduces harmful fragmentation of capacity.

Keywords: path selection, widest path problem, user-controlled networks, optical networks.

*This research is partially supported by the Natural Sciences and Engineering Research Council (NSERC) of Canada.

1 Introduction

Novel data-intensive network applications are pushing the limits of the hardware and software foundations of the Internet. Rising to the challenge, recent research has focused on optical network architectures and advanced transport protocols. One specific paradigm, motivated by cutting edge e-science but relevant to ordinary Internet users and service providers, is to virtualize optical circuits and expose them for user control through a service-oriented software platform [1]. Such a platform allows users to establish optical circuits, subsequently referred to as *lightpaths*, across multiple management domains, on demand. Because such circuits are provisioned over an optical medium and are dedicated, they offer superior capacity and reliability, and permit the safe use of customized high-performance transport protocols. Lightpaths are also straight-forward to access thanks to link layer encapsulation protocols, which make a lightpath appear as a single 1 Gb/s or 10 Gb/s Ethernet link for IP routing.

Routing computations in user-controlled optical networks pose some unique challenges. To begin with, such networks comprise a diverse pool of resources advertised for lease by a global community of users. In particular, the advertisement of a lightpath is subject to an expiry time, up to which the lightpath can be leased by another user. The latter user can then partition the acquired lightpath, if needed, and advertise unwanted partitions for lease. Thus, end-to-end path computations must be constrained to take into account the time for which a lightpath is desired versus the time for which resources are available. Furthermore, we face the problem of resource fragmentation: as lightpaths are traded and partitioned, it is possible that capacity for a lightpath is available during some contiguous interval of time, but cannot be leased because it is logically represented by a set of distinct resource advertisements. Fragmentation of capacity is a well-known phenomenon in SONET/SDH networks, but its analog in the time domain is unique to the user-controlled environment. Time fragmentation occurs, for example, when two lightpaths with the same endpoints are advertised for lease, say one Monday to Tuesday and the other Wednesday to Friday, but it is not possible to combine them to create a single lightpath available nonstop Monday to Friday.

The second challenge in path computation is the complexity of the logical network topology.

This is a consequence of the way signals are multiplexed in optical networks, namely through wavelength division (WDM) and time division multiplexing (TDM). Under the combination of WDM and TDM, which is typical in current networks, a single physical optical fiber may be represented by hundreds or even thousands of distinct logical lightpaths during path computation. Aside from the obvious implication on the cost of path computations, we face the problem that the naive path optimization strategy based on hop count may yield many optimal paths that differ in other practical ways. For example, one of the many shortest paths may be the best choice for reducing resource fragmentation.

Background and Related Work on Path Computation

Path computation plays a role not only in forwarding of IP traffic in layer 3 of the OSI model, but also in the establishment of circuits. The fundamental graph-theoretic approach to intra-domain routing is to compute the least cost path between two vertices in a graph G with vertex set V and edge set E, based on some cost function $W : E \rightarrow \mathbb{R}^+$. The most popular cost metric is hop count, which is the number of routing or switching devices traversed in a particular network layer on the path to the destination after leaving the source. Centralized computations using information from link state routing algorithms can be performed using Dijkstra's algorithm, which assumes non-negative link costs and runs in $\mathcal{O}(|E|+|V|\log|V|)$ time given an efficient Fibonacci heap data structure [2]. Distributed distance vector computations are based on a form of the Bellman-Ford algorithm. In order to break ties among least cost paths, techniques have been proposed that consider additional criteria either through modifications to the path computation algorithm, or by applying successive least cost computations with different cost functions [3, 4].

The widest path problem is concerned with finding a path with the largest bottleneck capacity. This is a natural problem to consider in our case since one of the intended uses of user-controlled optical networks is to establish dedicated lightpaths for individual data transfers, with the natural goal of reducing transfer time. The solution can be obtained using a slightly modified version of Dijkstra's or Bellman-Ford algorithm, where the + operator is replaced with the *max* operator, and the cost metric is taken to be the inverse of capacity. The time complexity of the computation is the same as in the least-cost problem (since the additional cost is merely $\mathcal{O}(|E|)$).

Natural variations of the least cost and widest path problems consider multiple weight functions and user-specific constraints [5, 6]. The link-constrained least cost path problem selects a path with least cost under one weight function but only considers edges that satisfy a constraint expressed with respect to a second weight function. For example, the user may wish to find the shortest path consisting of links whose capacity meets or exceeds some required value. This problem can be solved easily by applying any least cost path algorithm on a modified input graph that contains only those edges that satisfy the link constraints. The cost-constrained widest path problem selects the path that is widest with respect to one weight function but whose total cost under a second weight function does not exceed a given threshold. For example, the user may wish to find the path with greatest end-to-end capacity that is no longer than some fixed number of hops. The problem can be solved in polynomial time by applying $\mathcal{O}(|E|)$ ordinary least cost path computations, namely one for each possible value of the weight function by which link width is measured.

Contributions and Organization

In this paper, we examine path computation problems associated with establishment of an end-to-end lightpath in a user-controlled optical network. Our specific contributions are as follows:

(1) We formulate the novel problem of finding an end-to-end lightpath that minimizes the time needed to transfer a fixed volume of data, such as a multi-media file. We propose centralized and distributed solutions to this problem, which achieve efficiency through the use of sophisticated data structures.

(2) We propose strategies for breaking ties among optimal solutions in a least cost path computation, and present computational techniques based on sophisticated weight functions that can be used with standard least cost path algorithms. Our approach is general in that it permits breaking ties according to a hierarchy of optimization criteria, not necessarily starting with hop count. We also show, through simulation, the effectiveness of breaking ties in shortest path computations in a manner that reduces harmful fragmentation of capacity.

The remainder of the paper is organized as follows. We begin by reviewing relevant technolo-

gies in Section 2. Next, in Section 3, we describe our network model. Path selection problems and solution techniques are then covered. In Section 4, we address the minimum transfer time problem. Then, in Section 5, we consider breaking ties in shortest path computations. Section 6 concludes the paper.

2 Technologies for Management of Optical Transport Networks

Conventional optical transport networks are managed through a centralized operations and support system (OSS), where provisioning of connections is a manual process that can be slow, error-prone, and costly. Consequently, the emerging approach is to create connections on-demand through a user-network interface (UNI). This new signalling functionality is typically deployed in a distributed control plane, which in addition provides automated configuration and resource discovery [7]. A similar concept is the network-network interface (NNI), which allows signalling between control domains in support of end-to-end service provisioning [8].

Recent standardization activities related to optical control planes include the GMPLS signalling protocols defined by the IETF [9], the Automatic Switched Optical Network (ASON) proposed by the ITU-T [10], as well as the UNI and NNI signalling specifications of the OIF [11, 12]. These proposals are limited to the intra-carrier scenario, where a single management domain is partitioned into multiple control domains, for example due to lack of interoperability between vendor-specific control planes. The problem of flexible user control over resource from multiple carriers is not considered.

The earliest successful efforts to empower users with the ability to provision end-to-end lightpaths across multiple management domain were pioneered by CANARIE Inc., Canada's advanced Internet development organization. Several user control platforms were developed under joint sponsorship of CANARIE Inc. and Cisco Canada Inc., employing a variety of service-oriented architectures and implementation technologies [1, 13, 14, 15]. These platforms differ in the data model and management features offered, but all support the ability to construct end-to-end paths by cross-connecting a sequence of constituent lightpaths advertised for lease. Some of these platforms additionally allow users to partition (the capacity of) lightpaths. Following the first

wave of UCLP systems, other systems were developed in an effort to broaden the management functionality and scope of application of UCLP [16, 17, 18, 19].

3 Network Model

Graph Representation of Logical Topology

The logical topology for path computations is represented as a graph $G = (V, E)$ with vertex set V and edge set E. We will describe shortly how G can be constructed in various UCLP architectures, but for now we simply assume that G is given. In general, G will represent a subset of resources that meet the requirements and policies of the user u performing a path computation, and that are accessible to u under the policies of their owners and the UCLP system. The evolution of the graph G over time is a complex process and is beyond the scope of this paper; our focus is to merely define a representation of the network that can be fed to a path optimization algorithm.

Each vertex in V represents an optical cross-connect device such as an electronic TDM (i.e., SONET/SDH) switch, an all-optical WDM device, or some combination of both. The vertex set is partitioned into *management domains*, which represent entities or organizations that administer the corresponding switching equipment. We assume that each domain provides a software access point for user control over resources in that domain, but we do not assume any particular software technology or trust mechanism. Neighbouring domains are connected using a small number of peering links.

Each edge in E represents a lightpath, which can be an optical fiber, a single wavelength, or an electronically multiplexed signal of sub-wavelength granularity. We consider, for simplicity, that lightpaths are bidirectional, with symmetric capacity in both directions (our techniques can be generalized to the case when either of these assumptions does not hold). Without loss of generality, we allow a single lightpath in E to represent a circuit that is physically routed through multiple cross-connect devices in V.

Properties of Lightpaths

Existing user-controlled lightpath platforms [1, 13, 14] define the following properties for a light-

path $l \in E$:

- $U(l)$ – the ID of the user that currently owns the lightpath
- $C(l)$ – the capacity of the lightpath (e.g. 10 Gb/s)
- $T_A(l)$ – the advertisement expiry time, if l is advertised for lease, -1 otherwise
- $T_L(l)$ – the lease expiry time

When computing a path for a user u, we define the effective expiry time of a lightpath l with respect to user u as

$$T(u,l) = \begin{cases} T_L(l) & \text{if } U(l) = u \\ T_A(l) & \text{otherwise} \end{cases} \quad (1)$$

The capacity of a lightpath is constrained by the underlying hardware technology. SONET (Synchronous Optical NETwork) supports a hierarchy of circuit sizes denoted STS-x where $x \in \mathcal{C} = \{1, 3, 6, 9, 12, 24, 48, 192\}$. The raw capacity of an STS-x circuit is $x \times 51,840$ kb/s, and the payload capacity is $x \times 48,960$ kb/s. SDH (Synchronous Digital Hierarchy) is very similar to SONET, but uses different circuit size designations, based on multiples of $155,520$ kb/s. Lightpaths carried over ATM networks may support a much more fine-grained range of capacity values, which for our purposes is a continuous spectrum bounded by some maximum capacity.

Operations on Lightpaths

Users can perform a variety of management operations on lightpaths. As mentioned before, lightpaths can be leased and advertised for lease, but we do not explain this process in detail as it is beyond the scope of this paper. For our purposes, it suffices to say that from the point of view of user u, and lightpath l is either owned by u (i.e., $U(l) = u$), or it can be leased by u (i.e., $T_A(l) \neq -1$), or else it is not accessible to u and will not appear in the network graph G. Users may also partition and concatenate their own lightpaths. Partitioning refers to the division of capacity of a *parent* lightpath among multiple *child* lightpaths with the same endpoints. We assume that given two lightpaths l_1 and l_2 with capacity values $C(l_1) < C(l_2)$, it is possible to partition l_2 into a set of children that includes at least one child with capacity equal to $C(l_1)$. (We

have verified this assumption for our simulation environment, which models the Cisco ONS 15454 and Nortel OME 6500 SONET platforms.) Concatenation refers to the cross-connection of two lightpaths that share an endpoint into a longer lightpath. We assume that any two lightpaths that share an endpoint and have the same capacity can be cross-connected. (This is a valid assumption for electronic cross-connect devices such as SONET and ATM switches.)

Partitioning and concatenation operations are used in the establishment of end-to-end lightpaths as follows:

1. A user u computes a path $\langle l_1, \ldots, l_k \rangle$ through the network graph G between some source vertex S and some destination vertex D.

2. User u chooses a capacity value C not exceeding $\min \{C(l_i) \mid 1 \leq i \leq k\}$.

3. User u creates a partition l'_i with capacity C within each constituent lightpath l_i.

4. User u concatenates the partitions l'_i, forming an end-to-end lightpath between S and D, with capacity C.

We assume that when user u performs a path computation, the corresponding graph G will be constructed in such a way that for any $l \in E$, $T(u, l) \neq -1$. This ensures that each lightpath in G either belongs to u or can be leased by u.

Construction of Network Graph and Path Computation

We now describe how G can be constructed and used in various software architectures for user control over lightpaths (UCLP). First, consider centralized computations, where a user u first gathers information about lightpaths from one or more repositories, and then constructs a single graph G on which a path optimization algorithm can be executed. In a centralized UCLP architecture, we construct G by querying a single repository of lightpaths. User u's query identifies *eligible* resources according to a set of requirements and policies, for example selecting lightpaths l such that $C(l) \geq 1$ Gb/s, $T(u, l) \geq$ "two days in the future", and where l currently belongs to some subset of users trusted by u. (The appropriate constraints depend on the optimization criteria.) The query is then submitted by u's client application to a UCLP Web service, which

translates the query and applies it against a database, embellished with additional constraints representing management policies (e.g., if multiple identical lightpaths exist between two vertices, return only one to reduce the size of the query result). Finally, the graph G is formed and a path computation is executed on G, either by the Web service or by the client application. In a distributed UCLP system, a client may query multiple per-domain repositories in the same manner, combine the result into a single graph G, and then perform a centralized computation locally.

Distributed UCLP architectures enable efficient distributed path computations for certain optimization criteria. For example, the least cost or widest path computation can be parallelized as follows. First, for any source/destination vertex pair $\langle S, D \rangle$, the user u selects a sequence of neighbouring management domains $\langle M_1, M_2, \ldots, M_k \rangle$ such that S is in M_1 and D is in M_k. (The domains can be selected using, for example, extensions to the Border Gateway Protocol [20].) Next, u fixes an *ingress* and *egress* vertex for each domain, such that (for $1 \leq i < k$) the egress vertex in M_i is connected by a peering link to the ingress vertex in M_{i+1}. (The ingress vertex in M_1 is S and the egress vertex in M_k is D.) Finally, for each domain M_i, u computes separately a path from the ingress to the egress vertex within M_i. The queries and path computations corresponding to each domain are centralized and can be executed in parallel, and their results then used to construct an end-to-end lightpath from S to D. Thus, multiple graphs $G_1, \ldots, G_k$ are constructed and analyzed on behalf of the user, where G_i represents resources in domain M_i. The per-domain computations can also be performed sequentially, which effectively grows the end-to-end path from S to D one domain at a time. This has the advantage that the computation through M_i can be adjusted given the outcome of the computations in $M_1, \ldots, M_{i-1}$, which can reduce the size of G_i and also avoid unnecessary fragmentation of resources (e.g., in a widest path computation where the bottleneck edge occurs in M_1).

4 Minimum Transfer Time

One of the key features of a user-controlled optical network is the ability to provision lightpaths for individual transfers of large files. This problem was previously studied by Iraqi and Li, who

devised a modified GridFTP client that accelerated file transfers by automatically provisioning a lightpath and routing IP traffic through the lightpath, thus avoiding bottlenecks in the default Internet route [21, 13]. In this section, we consider the mathematical problem of selecting an end-to-end path that minimizes the time needed to transfer a file. The problem is formally defined as follows. Let $G = (V, E)$ be a graph representing network resources, as defined in Section 3. Let S and D be designated source and destination vertices in V, and let Z be the size of the file to be transferred. Let u and $C_{\max}$ denote the ID of the user requesting the end-to-end path, and the maximum capacity supported by that user's network interface. For any end-to-end path $l' = \langle l_1, \ldots, l_k \rangle$ through G, recall that the capacity of l' is $C(l') = \min_{1 \leq i \leq k} C(l_i)$, and that the effective expiry date of l' is $T(u, l') = \min_{1 \leq i \leq k} T(u, l_i)$. Then the time needed to transfer Z units of data through l' at capacity not exceeding $C_{\max}$ is given by $Z / \min\{C_{\max}, C(l')\}$. Our goal is to find an end-to-end path l' between S and D through G, such that the latter time quantity is minimized *and* does not exceed the difference between $T(u, l')$ and current time. (Note that we must add a safety margin to the current time to compensate for any delay in computing and setting up the end-to-end path.)

The minimum transfer time path computation problem is fundamentally different from the widest path problem because the capacity and expiry time of the end-to-end path are jointly constrained with respect to the parameter Z. Ignore for now the parameter $C_{\max}$, and consider the widest path through G between S and D. It is easy to see that such a path may have an expiry time that is too small to permit transfer of Z units of data before expiry, even though a "narrower" end-to-end path may exist whose expiry time is sufficient. The same situation may occur even if we assume that every lightpath in G can individually support the desired file transfer, if used at its maximum capacity, because these lightpaths may be partitioned before they are cross-connected to form the end-to-end path, in order to accommodate the bottleneck capacity. (A partition of a lightpath in G may be incapable of supporting the desired file transfer even if the parent lightpath is capable.) We conjecture that there is no straightforward reduction of the problem at hand to the widest path problem.

4.1 Centralized Computation

The minimum transfer time problem can be solved in polynomial time by performing a sequence of reachability computations, where each computation determines whether a path of a particular capacity exists between the desired pair of endpoints. This technique is presented below as Algorithm 1, which solves part of the problem by computing the capacity of the optimal path. The full problem is solved by running Algorithm 1 followed by an additional least cost path computation, with the capacity and expiry time suitably constrained.

Algorithm 1 first identifies the possible capacity values of the end-to-end path on line 2. For each capacity value identified, the algorithm then performs a reachability query on line 6 (e.g. using breadth first search) to determine whether an end-to-end path can be constructed at that capacity with sufficient expiry time to accommodate the given data size Z. Capacity values outside the set U computed on line 2 need not be considered because the bottleneck capacity of any end-to-end path through G either exceeds $C_{\max}$ or is one of the values in U. However, the capacity value $C_{\max}$ (the maximum capacity supported by the user), must be added on line 2 to ensure the correctness of the result. The need for this is illustrated in the following scenario. Suppose G contains two vertices and two parallel lightpaths, l_1 and l_2, with capacity 1 Gb/s and 10 Gb/s, respectively. Let $C_{\max} = 5$ Gb/s and suppose that a 10 Gb/s lightpath can be partitioned into two 5 Gb/s lightpaths. Let $T(u, l_1)$ and $T(u, l_2)$ be such that both l_1 and l_2 can be used to transfer Z units of data, even if l_2 is operated at 5 Gb/s (i.e., half capacity). If the algorithm does not consider $C_{\max}$ as a possible return value, it will return $C(l_1) = 1$ Gb/s, even though it should return $C_{\max} = 5$ Gb/s.

Each iteration of Algorithm 1 considers one capacity value. The subset of lightpaths that can be used to transfer Z units of data at that capacity is computed on line 4. The algorithm makes use of the assumption (discussed in Section 3) that given two values of capacity C_i and C_j, if $C_i > C_j$ then a partition of size C_j can be allocated in any lightpath with capacity C_i. If this assumption does not hold, then the set E_i must be pruned by removing lightpaths incapable of providing partitions of size C_i.

The expiry time is padded on line 1 with a safety margin that takes into account the time

needed to perform the path computation and set up the lightpath. The variable T_{begin} defined here estimates the actual time at which the file transfer might begin based on this safety margin. Once the algorithm identifies the maximum capacity of an end-to-end path for transferring a file of size Z, a separate least cost path computation can be performed to find the best such path. Within each iteration, however, we wish to minimize the computational cost and so we do not optimize the path on line 6.

Algorithm 1: FastestTransfer $(G, u, S, D, Z, C_{\max})$

input : $G = (V, E)$ – input graph
u – ID of calling user
S, D – source and destination vertices in V
Z – size of file to be transferred
$C_{\max}$ – maximum lightpath capacity supported by user

output: capacity of an end-to-end path between S to D in G that permits non-stop transfer of Z units of data in minimum time, or ERROR if no such path exists

1 $T_{\text{begin}} \longleftarrow$ current time + safety margin
2 $U \longleftarrow$ list $\langle C_1, C_2, \ldots, C_N \rangle$ of distinct elements of $\{C_{\max}\} \cup \{C(e) \mid e \in E \text{ s.t. } C(e) \leq C_{\max}\}$ in descending order
3 **foreach** i from 1 to N **do**
4 $\quad E_i \longleftarrow \{e \mid e \in E \text{ and } C(e) \geq C_i \text{ and } C_i \times (T(u, e) - T_{\text{begin}}) \geq Z\}$
5 $\quad G_i \longleftarrow$ graph with vertex set V and edge set E_i
6 $\quad$ **if** there is a path between S and D in G_i **then**
7 $\quad\quad$ **return** C_i
8 $\quad$ **end**
9 **end**
10 **return** ERROR

The maximum number of iterations, N, performed by Algorithm 1 is determined by the number of distinct capacity values among the lightpaths in the input graph, G. This value, in

turn, depends on the hardware technology underlying the lightpaths. In SONET, N is at most a small integer (< 10) irrespective of the size of the network, because all circuit sizes are chosen from a small set of discrete values, as explained in Section 3. If lightpaths are provisioned over a statistical multiplexing technology, such as ATM, then their capacities are less restricted and N can potentially be as high as $|E|$. In either case, N is at most $|E|$. Finally, regardless of the network technology, the number of iterations can be reduced by restricting the set U, for example to some subset of representative or commonly used values. For example, for simplicity one may focus on 1 Gbps and 10 Gbps lightpaths exclusively. However, the optimality of the result (i.e., the end-to-end path) with respect to data transfer time is no longer guaranteed in that case.

Time Complexity of Rudimentary Algorithm

First, let us consider the worst-case time complexity of Algorithm 1 using rudimentary data structures. Creating the list $U = \langle C_1, C_2, \ldots, C_N \rangle$ is done once, and takes $\mathcal{O}(|E| \log N)$ time using a modified merge sort (see Appendix A). Then, each iteration entails constructing E_i and executing the corresponding connectivity query on G_i. A rudimentary implementation scans over E to create E_i in $\mathcal{O}(|E|)$ time, constructs G_i in $\mathcal{O}(|E| + |V|)$ time using an adjacency list representation, and then feeds G_i into breadth-first or depth-first search to perform the connectivity query on line 6, which runs in $\mathcal{O}(|E| + |V|)$ time. In that case, the per-iteration cost is $\mathcal{O}(|E| + |V|)$ and the overall cost is $\mathcal{O}(N(|E| + |V|))$. (This includes the cost of computing the list U). Using an adjacency matrix representation, constructing G_i costs $\mathcal{O}(|E| + |V|^2)$ steps and the connectivity query costs $\mathcal{O}(|V|^2)$ time. In that case, the per-iteration cost is $\mathcal{O}(|E| + |V|^2)$ and the overall cost is $\mathcal{O}(N(|E| + |V|^2))$. Thus, we get the best asymptotic time complexity with the adjacency list representation.

Finally, let us consider how the running time of Algorithm 1 depends on the relationship between N and $|E|$. In a SONET environment where N is a small integer, the running time is linear in the size of the network. However, if $N \approx |E|$, say in an ATM or MPLS network, then the complexity is $\mathcal{O}(|E|^2 + |E||V|)$. In that case, the algorithm can be accelerated by performing N iterations in parallel, which is straight-forward since each iteration can be executed independently of the others once the first two lines of Algorithm 1 are completed. The parallel running time on

N nodes is $\mathcal{O}(|E| \log N + |E| + |V|)$.

Efficient Representation of Dense Graphs

Another way to speed up Algorithm 1 is to re-use intermediate results between iterations. Specifically, we can re-use the data structure representing the graph G_i, and even optimize the connectivity query on line 6 using a dynamic graph search algorithm. (A dynamic graph algorithm efficiently recomputes its result as edges are added or removed.) These optimizations are especially useful when N is large, say $N = |E|$, and $|E| > |V|$, in which case the complexity of the rudimentary version is $\mathcal{O}(|E|^2)$. The high-level intuition behind the optimization described above is that in Algorithm 1, each edge appears in E_i during a fixed number of *consecutive* iterations. Thus, the evolution of G_i as i increases is such that each edge appears and disappears at most once, in that order. To take advantage of this observation, we produce two sorted lists X and Y of $l \in E$, sorted in descending order as follows: one by $C(l)$, and the other by $Z/T(u, l)$. It is easy to see that these lists can be partitioned into (possibly empty) contiguous sublists, say $X \to \{X_1, X_2, \ldots, X_N\}$ and $Y \to \{Y_1, Y_2, \ldots, Y_N\}$ such that E_i is formed from E_{i-1} (or from $\emptyset$ if $i = 0$), by adding edges from X_i and deleting edges from Y_i. Now, since the number edges added or removed per iteration is $2|E|/N$ on average (i.e., $|E|/N$ added and $|E|/N$ removed), for large N the graphs E_{i-1} and E_i tend to be quite similar. This motivates re-using the work done to perform the connectivity query (line 6) on E_{i-1} when considering E_i.

The time complexity of Algorithm 1 can be improved by using the above observations to optimize the representation of G_i. Previously, we considered the adjacency list representation, because the cost of forming G_i is less for sparse graphs than using an adjacency matrix (i.e., worst case $\mathcal{O}(|E|+|V|)$ versus $\mathcal{O}(|E|+|V|^2)$). However, the cost of constructing G_i incrementally from G_{i-1} is lower if we use an adjacency matrix, namely $\mathcal{O}(|X_i| + |Y_i|)$. One caveat here is that there may be multiple parallel edges between any pair of vertices, and we need an efficient way to determine whether two vertices remain connected after the edges from X_i are added and those from Y_i are removed. To that end, we cannot use a vanilla implementation of the adjacency matrix where each entry is a Boolean value indicating the presence of at least one edge. Instead, we propose an augmented adjacency matrix where each entry is a counter of the number of parallel

edges, rather than a Boolean value. To construct G_i from G_{i-1}, for each edge in X_i and Y_i we increment and, respectively, decrement the corresponding matrix entry.

The total cost of updating the augmented adjacency matrix is relatively small since $\sum_{i=1}^{N} |X_i| = \sum_{i=1}^{N} |Y_i| = |E|$, and so forming the graphs G_i costs $\mathcal{O}(|E| + |V|)$ in total for all iterations. For comparison, forming G_i from G_{i-1} incrementally using an adjacency list representation may require lengthy list traversals, at a cost of $|E_i|$ in the worst case per iteration, and $\mathcal{O}(|V| + N|E|)$ in total. (The worst case occurs when lightpaths have large expiry times, and so $|E_i|$ is close to $|E|$ for most i.) Since we are concerned with the case when $N \approx |E|$, the adjacency matrix representation gives us better complexity when $|E| \geq |V|$, which is a realistic scenario for user-controlled optical networks. The adjacency matrix representation also bounds the worst-case cost of the connectivity query on line 6 of Algorithm 1 at $\mathcal{O}(|V|^2)$ irrespective of $|E|$. Thus, the time complexity of Algorithm 1 implemented using the augmented adjacency matrix representation of G_i and ordinary depth first search is $\mathcal{O}(|E| \log N + N|V|^2)$. This improves upon the rudimentary version of the algorithm, whose complexity is $\mathcal{O}(N(|E| + |V|^2))$.

Dynamic Graph Search

Algorithm 1 can be optimized even further by using a more sophisticated connectivity query, based on a dynamic graph search algorithm. To our knowledge, the fastest dynamic connectivity algorithm for undirected graphs is by Holm, De Lichtenberg, and Thorup [22], which builds on the idea originally proposed by Henzinger and King [23]. The former algorithm supports connectivity queries in $\mathcal{O}(\log |V| / \log \log |V|)$ time and edge updates (insertions or deletions) in $\mathcal{O}(\log^2 |V|)$ amortized time, and has an initialization cost of $\mathcal{O}(\log |V|)$ when starting with an empty edge set. The algorithm is randomized and makes use of balanced binary trees of degree $|V|$ to represent the connected components of the graph. In particular, when incorporated into Algorithm 1 (i.e., on line 6) the dynamic graph search relieves us of the burden to explicitly materialize G_i; instead, we simply compute E_i for iteration i, execute an insertion operation for each edge in X_i, and execute a deletion operation for each edge in Y_i. (Recall that $X_i = E_i \setminus E_{i-1}$ and $Y_i = E_{i-1} \setminus E_i$.)

The time complexity of Algorithm 1, when modified this way, is

$$\mathcal{O}(|E| \log N + |E| \log^2 |V| + N \log |V| / \log \log |V|).$$

which in some cases improves on the complexity of the version with the augmented adjacency matrix and ordinary depth first search (i.e., $\mathcal{O}(|E| \log N + N|V|^2)$). In particular, it is easy to see that we obtain better asymptotic complexity when $N \approx |E|$.

The reachability problem in directed graphs is more complex than in undirected graphs, and consequently the time complexity of the best dynamic algorithms is worse. A fast algorithm based on maintaining a transitive closure matrix is proposed by Roditty in [24]. The algorithm has an initialization cost of $\mathcal{O}(|V|^2)$ when starting with an empty edge set, offers a constant query time, and has an edge update cost of $\mathcal{O}(|V|^2)$. (The latter is optimal.) When Roditty's dynamic reachability algorithm is incorporated into Algorithm 1, the total running time becomes

$$\mathcal{O}(|E| \log N + |E||V|^2),$$

which is worse than using depth first search and the adjacency matrix graph representation. It may be possible to achieve better complexity using a dynamic algorithm that does not explicitly maintain a transitive closure matrix. To our knowledge, the fastest such algorithms are proposed by Roditty and Zwick in [25]. Unfortunately, the cost of using these algorithms per iteration of Algorithm 1 is at least proportional to $|V|^2$ in the worst case when G_i is dense, and so we do not get an improvement over depth first search.

Summary

The time complexity of the different versions of Algorithm 1 considered in this section is summarized below in Table 1.

4.2 Distributed Computation by Flooding

In this section, we consider computing the end-to-end path that minimizes the time needed for a non-stop transfer of a file of size Z using a distributed flooding approach. Like Algorithm 1, the

Version of Algorithm	Time Complexity
adj. list + depth first search	$\mathcal{O}(N(\|E\| + \|V\|))$
augmented adj. matrix + depth first search	$\mathcal{O}(\|E\| \log N + N\|V\|^2)$
dynamic graph search	$\mathcal{O}(\|E\| \log N + \|E\| \log^2 \|V\| + N \log \|V\| / \log\log \|V\|)$

Table 1: Time complexity of Algorithm 1.

flooding method only partially solves the problem at hand, finding the capacity of the optimal end-to-end path between some designated nodes S and D, on behalf of some user u.

Structure of Control Message

During the execution of the flooding algorithm, nodes exchange short control messages with their neighbours, as in [26, 27]. The essential data fields of the control message are listed below in Table 2. Control messages also carry a request ID to distinguish traffic corresponding to different path computations, but for simplicity we ignore this aspect of the problem here. The values T_{route} and C_{route} are well-defined because each control message represents information related to a single path between the source node and the receiving node. If there is a path from the source node S to some node X, which can be extended to node Y in more than one way, then X sends separate control messages to Y for each possible extension. As an optimization, such groups of control messages can be batched and transmitted over the control network as a single packet.

Flooding Algorithm

Each node M maintains, for each path computation request, session data consisting of a set C_{seen}, initially $\emptyset$. This is the set of capacity values for which a path exists between S and M. The client initializes the flooding algorithm by sending to the source node S a message containing the values of Z, S, and D specified by the user, the ID u of the user, $T_{\text{route}} = \infty$, $C_{\text{route}} = \infty$, and $T_{\text{begin}} =$ current time + safety margin (as in Algorithm 1). Upon receiving a control message, M parses the message and computes the following interval of the real numbers:

$$C_{\text{cur}} = [Z/(T_{\text{route}} - T_{\text{begin}}), C_{\text{route}}]\,.$$

Parameter	Description
u	ID of requesting user
Z	size of the file to be transferred
S, D	source and destination nodes
T_{route}	minimum expiry time among lightpaths on path from S to receiving node
C_{route}	bottleneck capacity among lightpaths on path from S to receiving node
T_{begin}	estimate of the time at which the path computation will be completed

Table 2: Essential fields carried by control message in flood routing computation.

This interval represents the range of capacity values for which a path from the source node to M can be formed. The network may only support paths with capacities at certain discrete values in this interval, but the flooding algorithm guarantees that the upper endpoint of the interval is a supported capacity value.

Upon computing C_{cur}, M decides whether the message received contains information not seen before. That is, it checks whether $C_{\text{cur}} \subseteq C_{\text{seen}}$. If so, then paths corresponding to the values of capacity in C_{cur} have already been discovered, and the message being processed can be dropped safely. If not, then M has received fresh information, possibly because it was just contacted by the client to initialize the algorithm. If M is the destination D, then it reports the interval of real numbers C_{cur} back to the client that initiated the flooding computation. For efficiency, these messages can be aggregated over intervals of time. The client may then select some capacity value from this set and attempt to establish the least cost path between S and D at that capacity. (The client must ensure that the flooding algorithm has terminated before attempting to set up the path, otherwise the path may be suboptimal. We return to this issue later in this section.) Otherwise, M must forward information to its neighbours. In either case, M updates its local session data by assigning $C_{\text{seen}} := C_{\text{seen}} \cup C_{\text{cur}}$.

If M is *not* the destination node, and $C_{\text{cur}} \not\subseteq C_{\text{seen}}$ holds before C_{seen} is updated, then additional control messages must be sent to neighbouring nodes. For each neighbouring node

X, each lightpath l from M to X is considered. First, an eligibility check is performed, which consists of asserting *all* of the following conditions (stated with respect to the fields u and Z of the control message under consideration):

1. $T(u, l) > 0$
2. $[Z/(T(u, l) - T_{\text{begin}}), C(l)] \cap C_{\text{cur}} \neq \emptyset$

The eligibility check ensures that there exists a path P from the source node to neighbour X via M that can be obtained by user u and that extends the path reported by the control message under consideration (first condition), and that P can support non-stop transfer of Z units of data at some capacity in the set C_{cur} (second condition). If both conditions hold, then M forwards to X a control message as follows. Let primed variables (i.e, u', Z', S', D', T'_{route} and C'_{route}) denote the field values of this message. Then the new message is generated as follows:

$$\begin{aligned} u' &= u \\ Z' &= Z \\ S' &= S \\ D' &= D \\ T'_{\text{route}} &= \min\{T_{\text{route}}, T(u, l)\} \\ C'_{\text{route}} &= \min\{C_{\text{route}}, C(l)\} \end{aligned}$$

This message is then sent to neighbour X. It follows from condition 2 of the eligibility check, and from the choice of T'_{route} and C'_{route}, that the real interval $[Z/(T'_{\text{route}} - T_{\text{begin}}), C'_{\text{route}}]$ represented by the new message is not empty.

Correctness

The correctness properties of the flooding algorithm described above are stated in the theorem:

Theorem 4.1. *Let $G = (V, E)$ be a graph representing the nodes and lightpaths in the network, as defined in Section 3. Fix some user ID u, file size Z, and distinct nodes $S, D \in V$. Let T_{begin}*

be the value computed by the client at the beginning of the flooding computation. Suppose that an end-to-end path L exists in G between S and D, consisting of eligible constituent lightpaths $\langle l_1, l_2, \ldots, l_k \rangle$, having bottleneck capacity $C = \min_{l \in L} C(l)$, and satisfying $Z/(\min_{l \in L} T(u,l) - T_{\text{begin}}) \leq C$ (i.e., being capable of transferring Z units of data non-stop). Then the destination node D eventually reports an interval of real numbers containing C to the client. Moreover, the number of messages generated by the algorithm is finite provided that V and E are finite.

Proof. To show that the client receives the response claimed, it suffices to show that D eventually receives a control message such that $C \in [Z/(T_{\text{route}} - T_{\text{begin}}), C_{\text{route}}]$. Suppose, for proof by contradiction, that this does not occur. Let $\langle x_0, x_1, x_2, \ldots, x_k \rangle$ be the sequence of nodes on the end-to-end path L, where $x_0 = S$ and $x_k = D$. Let x_j be a node in this sequence that does not receive a control message such that $C \in [Z/(T_{\text{route}} - T_{\text{begin}}), C_{\text{route}}]$, $0 \leq j \leq k$. Without loss of generality, choose x_j so that j is minimum. It follows by the initialization of the algorithm that x_0 receives a control message where $[Z/(T_{\text{route}} - T_{\text{begin}}), C_{\text{route}}] = [0, \infty]$, and so $j \neq 0$. Consequently, node x_{j-1} exists and eventually receives a control message where $C \in [Z/(T_{\text{route}} - T_{\text{begin}}), C_{\text{route}}]$.

Now, consider the action of x_{j-1} upon receiving the first such message. It follows that $[Z/(T_{\text{route}} - T_{\text{begin}}), C_{\text{route}}] \not\subseteq C_{\text{seen}}$, where C_{seen} is the session data of node x_{j-1} just before the message is received. Since l_j is an eligible lightpath in the path computation under consideration and since x_j is a neighbour of x_{j-1}, it follows that x_{j-1} sends to x_j a control message with field values $T'_{\text{route}} = \min\{T_{\text{route}}, T(u, l_j)\}$ and $C'_{\text{route}} = \min\{C_{\text{route}}, C(l_j)\}$. We will show that $C \in [Z/(T'_{\text{route}} - T_{\text{begin}}), C'_{\text{route}}]$. To see that $C'_{\text{route}} \geq C$, note that $C_{\text{route}} \geq C$ and $C(l_j) \geq C$ since $C = \min_{l \in L} C(l)$; thus, $\min\{C_{\text{route}}, C(l_j)\} \geq C$. To see that $Z/(T'_{\text{route}} - T_{\text{begin}}) \leq C$, note that $Z/(T_{\text{route}} - T_{\text{begin}}) \leq C$ and $Z/(T(u, l_j) - T_{\text{begin}}) \leq C$ by our definition of L above; thus, $Z/(\min\{T_{\text{route}}, T(u, l_j)\} - T_{\text{begin}}) \leq C$.

Finally, consider the number of messages generated by the flooding algorithm. We will show that this number is finite, hence the algorithm terminates. First, note that the lightpaths in the graph have at most $|E|$ distinct values of capacity and at most $|E|$ distinct values of $T(u, \ldots)$. Consequently, it is easy to see that there are at most $|E|^2$ possible control messages, differing according to the two endpoints of the corresponding interval of real numbers $[Z/(T_{\text{route}} - T_{\text{begin}}), C_{\text{route}}]$.

Thus, after processing at most $|E|^2$ distinct control messages, if a node receives an additional message then $[Z/(T_{\text{route}} - T_{\text{begin}}), C_{\text{route}}] \subseteq C_{\text{seen}}$, and the node does not create any additional messages. Before this occurs, for each distinct control message received a node will generate $\mathcal{O}(|E|)$ new messages, and so the total number of messages generated is bounded from above by $\mathcal{O}(|V||E|^3)$ (including messages to and from the client). A more careful analysis bounds the total number of messages at $\mathcal{O}(|V||E|^2)$. □

Waiting for Termination

To conclude this section, we discuss one other property of the flooding algorithm that is not addressed by Theorem 4.1, which is the termination of the algorithm executed by the client. (This must occur before the client attempts to set up the end-to-end lightpath.) That is, whereas Theorem 4.1 guarantees that the client *eventually* receives a message indicating the capacity of the optimal end-to-end path, it does not say specifically when this message is received or how it can be identified. The natural approach is for the client to estimate the time needed until the algorithm reaches a quiescent state (i.e., the last message has been sent, received, and processed) using partial synchrony assumptions, and terminate its algorithm at that time. To that end, define the following upper bound quantities: D – diameter of the logical topology (less than $|V|$), λ_t – transmission delay for a control message, λ_p – propagation delay for a control message (one-hop), λ_c – processing delay for a control message (CPU only). Then an upper bound on the time until quiescence is

$$(2 + D)(\lambda_t + \lambda_p + \lambda_c) + |V||E|^2(\lambda_t + \lambda_p).$$

Here the first term represents the time needed for control messages to travel from the client, across the network, and back to the client; the second term represents additional delays encountered at nodes due to prior control messages being processed, where $|V||E|^2$ is an upper bound on the total number of such messages (see proof of Theorem 4.1).

Data Structures

Implementation of the flooding algorithm requires an efficient data structure to represent the

session variable C_{seen}, which represents a union of closed intervals of real numbers. This data structure must support the following operations:

1. Test if a closed interval is a subset of a union of closed intervals.
2. Compute the union of a closed interval and an existing union of closed intervals.
3. Compute the intersection of two closed intervals.

A sorted linked list data structure can be used, where the nodes represent disjoint intervals ordered by the left (equivalently right) endpoint, in which case the first two operations can be performed in $\mathcal{O}(n)$ time where n is the total number of intervals in the union, and the third takes $\mathcal{O}(1)$ time. A sorted array implementation performs the test operation in $\mathcal{O}(\log n)$ time and union in $\mathcal{O}(n)$ time. A balanced search tree representation where nodes represent intervals, sorted by one endpoint, has better overall complexity than either type of list. The test operation takes $\mathcal{O}(\log n)$ time (as in a sorted array). The union operation, which occurs in the most deeply nested section of the algorithm, can take $\mathcal{O}(n \log n)$ time since in the worst case $\mathcal{O}(n)$ tree nodes must be removed from the tree. However, since each tree node is removed at most once, the amortized cost is still $\mathcal{O}(\log n)$ if we consider that tree node deletion is paid for by the corresponding insertion.

5 Least Cost Path Computations and Breaking Ties

Hop count is a popular and natural measure of path length and path cost, since it is associated with physical distance, buffering delay, and cost of physical infrastructure corresponding to a path. However, as noted earlier, in user-controlled optical networks a shortest path computation may yield a large number of optimal paths. In this section we describe a method to enhance the basic shortest path computation by breaking ties among optimal solutions in a judicious manner.

5.1 Tie-Breaking Metrics

A sensible approach to breaking ties considers capacity as a secondary optimization criterion, specifically by attempting to select an end-to-end path where the constituent lightpaths match

most closely the requirement of the user. In other words, we wish to choose the shortest path whose allocation will minimize the "amount" of excess capacity. This strategy tends to reduce fragmentation of capacity, which can improve the utilization of the network.

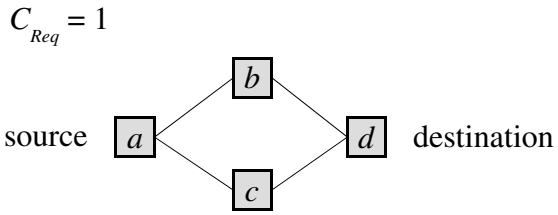
We propose to implement the above tie-breaking strategy by using an ordinary least cost path computation with an augmented weight function. Suppose we wish to break ties among shortest paths, where the hop count for a lightpath l is given by $H(l) \geq 1$. Then we apply the following augmented weight function W:

$$W(l) = H(l) + \frac{C(l) - C_{Req}}{|V| \max_{e \in E} C(e)} \tag{2}$$

where V and E are as defined in Section 3, and C_{Req} is the minimum capacity required by the user. The augmented weight function W can be thought of as the sum of H and a tie-breaking term whose contribution to the cost of the end-to-end path is less than unity. The latter follows from the fact that $W(l)$ is positive for each $l \in E$ (here $C(l) \geq C_{Req}$ holds by construction of the input graph), which implies that the least cost path is acyclic and hence has at most $|V|-1$ edges, each contributing at most $1/|V|$ in the tie-breaking term. Since hop count is an integer-valued quantity, it follows that W selects an end-to-end path that has the least total hop count, and that minimizes the sum of the tie-breaking terms as a secondary criterion. In particular, W always favours shortest end-to-end paths that match exactly the user's bandwidth requirement, and if no such paths exist then it still favours shortest paths. An example of equation (2) at work is presented in Figure 1.

The main advantage of our tie-breaking technique over those discussed in [3, 4], is that we require only one application of the standard shortest-path algorithm with a single weight function. Furthermore, the cost of computing the weight function does not change the asymptotic cost of the path computation since the denominator in formula 2 is a constant and can be computed in $O(|E|)$ time, or even hard-coded.

Our tie-breaking technique can be generalized to a hierarchy of multiple tie-breaking criteria, as stated in the following theorem.



lightpath *l*	*a-b*	*b-d*	*a-c*	*c-d*
H(l)	1	2	1	2
C(l)	1	1	2	1
W(l)	1+0/(4·2)	2+0/(4·2)	1+1/(4·2)	2+0/(4·2)

end-to-end path	*a-b-d*	*a-c-d*
total weight	3	3+1/8

Figure 1: Example of least cost path computation with tie breaking using equation (2).

Theorem 5.1. *Consider a graph $G = (V, E)$ such that $|V| \geq 2$, and a family of weight functions $W_i : E \to \mathbb{Z}^+$ for $1 \leq i \leq X$. Let*

$$W(l) = \sum_{i=1}^{X} \frac{W_i(l)}{\prod_{j=2}^{i} |V| \max_{e \in E} W_j(e)}. \tag{3}$$

for any $l \in E$ (where the denominator equals unity when $i = 1$). Let $A = \langle a_1, a_2, \ldots, a_m \rangle$ and $B = \langle b_1, b_2, \ldots, b_n \rangle$ be acyclic end-to-end paths between the same pair of vertices. Suppose that for some $x \in \mathbb{N}$, $1 \leq x \leq X$, the following holds for all x' such that $1 \leq x' < x$:

$$\sum_{k=1}^{m} W_{x'}(a_k) = \sum_{k=1}^{n} W_{x'}(b_k).$$

Then

$$\sum_{k=1}^{m} W_x(a_k) < \sum_{k=1}^{n} W_x(b_k) \implies \sum_{k=1}^{m} W(a_k) < \sum_{k=1}^{n} W(b_k).$$

In other words, if A and B have the same total cost under $W_{x'}$ for all x' such that $1 \leq x' < x$, but not under W_x, then the tie between A and B is broken according to W_x.

Proof. Let A and B be as defined above, and suppose that the hypothesis of the theorem holds for some x. That is, for all x' such that $1 \leq x' < x$, suppose that $\sum_{k=1}^{m} W_{x'}(a_k) = \sum_{k=1}^{n} W_{x'}(b_k)$. Furthermore, suppose that $\sum_{k=1}^{m} W_x(a_k) < \sum_{k=1}^{n} W_x(b_k)$. Note that, in that case, $\sum_{k=1}^{m} W_x(a_k) \leq \sum_{k=1}^{n} W_x(b_k) - 1$ since W_x has range $\mathbb{Z}^+$. Then we must show that $\sum_{k=1}^{m} W(a_k) < \sum_{k=1}^{n} W(b_k)$. Indeed, we have

$$
\begin{aligned}
& \sum_{k=1}^{m} W(a_k) - \sum_{k=1}^{n} W(b_k) \\
= \; & \sum_{k=1}^{m} \sum_{i=1}^{X} \frac{W_i(a_k)}{\prod_{j=2}^{i} |V| \max_{e \in E} W_j(e)} - \sum_{k=1}^{n} \sum_{i=1}^{X} \frac{W_i(b_k)}{\prod_{j=2}^{i} |V| \max_{e \in E} W_j(e)} \\
= \; & \sum_{i=1}^{x-1} \frac{1}{\prod_{j=2}^{i} |V| \max_{e \in E} W_j(e)} \left(\sum_{k=1}^{m} W_i(a_k) - \sum_{k=1}^{n} W_i(b_k) \right) + \\
& \frac{1}{\prod_{j=2}^{x} |V| \max_{e \in E} W_j(e)} \left(\sum_{k=1}^{m} W_x(a_k) - \sum_{k=1}^{n} W_x(b_k) \right) + \\
& \sum_{i=x+1}^{X} \frac{1}{\prod_{j=2}^{i} |V| \max_{e \in E} W_j(e)} \left(\sum_{k=1}^{m} W_i(a_k) - \sum_{k=1}^{n} W_i(b_k) \right) \\
= \; & \frac{1}{\prod_{j=2}^{x} |V| \max_{e \in E} W_j(e)} \left(\sum_{k=1}^{m} W_x(a_k) - \sum_{k=1}^{n} W_x(b_k) \right) + \\
& \sum_{i=x+1}^{X} \frac{1}{\prod_{j=2}^{i} |V| \max_{e \in E} W_j(e)} \left(\sum_{k=1}^{m} W_i(a_k) - \sum_{k=1}^{n} W_i(b_k) \right) \\
\leq \; & \frac{-1}{\prod_{j=2}^{x} |V| \max_{e \in E} W_j(e)} + \sum_{i=x+1}^{X} \frac{1}{\prod_{j=2}^{i} |V| \max_{e \in E} W_j(e)} \left(\sum_{k=1}^{m} W_i(a_k) - \sum_{k=1}^{n} W_i(b_k) \right) \\
\leq \; & \frac{-1}{\prod_{j=2}^{x} |V| \max_{e \in E} W_j(e)} + \sum_{i=x+1}^{X} \frac{1}{\prod_{j=2}^{i} |V| \max_{e \in E} W_j(e)} \left((|V| - 1) \max_{e \in E} W_i(e) \right) \\
= \; & \frac{1}{\prod_{j=2}^{x} |V| \max_{e \in E} W_j(e)} \left[-1 + \sum_{i=x+1}^{X} \frac{(|V| - 1) \max_{e \in E} W_i(e)}{\prod_{j=x+1}^{i} |V| \max_{e \in E} W_j(e)} \right] \\
= \; & \frac{1}{\prod_{j=2}^{x} |V| \max_{e \in E} W_j(e)} \left[-1 + \left(\frac{|V| - 1}{|V|} \right) \sum_{i=x+1}^{X} \frac{1}{\prod_{j=x+1}^{i-1} |V| \max_{e \in E} W_j(e)} \right]
\end{aligned}
$$

$$
\begin{aligned}
&\leq \frac{1}{\prod_{j=2}^{x} |V| \max_{e \in E} W_j(e)} \left[-1 + \left(\frac{|V|-1}{|V|} \right) \sum_{i=x+1}^{X} \frac{1}{\prod_{j=x+1}^{i-1} |V|} \right] \\
&< \frac{1}{\prod_{j=2}^{x} |V| \max_{e \in E} W_j(e)} \left[-1 + \left(\frac{|V|-1}{|V|} \right) \sum_{i=x+1}^{\infty} \frac{1}{\prod_{j=x+1}^{i-1} |V|} \right] \\
&= \frac{1}{\prod_{j=2}^{x} |V| \max_{e \in E} W_j(e)} \left[-1 + \left(\frac{|V|-1}{|V|} \right) \left(\frac{1}{1 - 1/|V|} \right) \right] \\
&= \frac{1}{\prod_{j=2}^{x} |V| \max_{e \in E} W_j(e)} (-1+1) \\
&= 0
\end{aligned}
$$

Thus, it follows that $\sum_{k=1}^{m} W(a_k) < \sum_{k=1}^{n} W(b_k)$, as wanted. □

5.2 Performance Evaluation

We have evaluated the effectiveness of our tie-breaking strategy through simulation. In particular, we considered breaking ties in shortest path computations using capacity as a secondary optimization criterion, using equation (2). (Here we define path length as the number of physical hops.)

Simulation Environment

The network topology in our simulation is based on CA*net4, presented below in Figure 2. Each edge in the physical topology corresponds to a pair of bidirectional 10 Gb/s links. Although CA*net4 was constructed using single 10 Gb/s links, we believe that the presence of multiple parallel links is becoming an immediate concern due to the growing popularity of wavelength division multiplexing (WDM) technology.

Traffic Generation

We consider that the network is subjected to end-to-end lightpath establishment requests in support of individual data transfers. Due to the heavyweight nature of lightpath establishment, we suppose that the set of files to be transferred in a single session is prepared in advance and

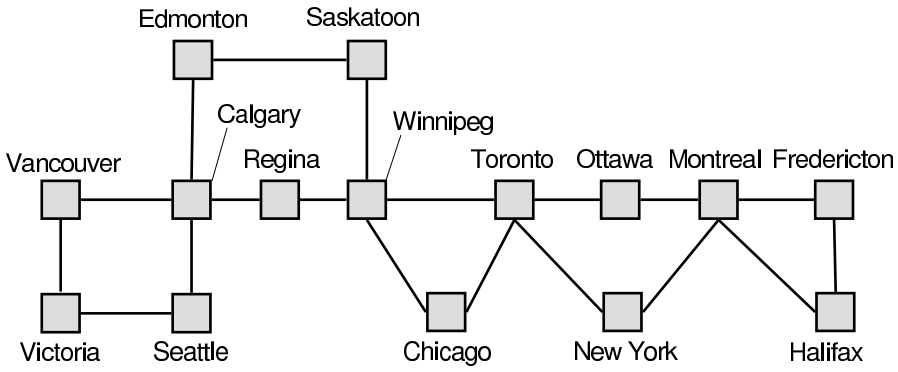


Figure 2: Physical network topology used in our simulation environment.

that a single lightpath is accordingly allocated per session. We consider that sessions are human-initiated and arrive according to a Poisson process with rate λ (in units of requests/hour), which has been found to be an accurate model in [28, 29, 30]. Similarly, we model session size using a log-normal distribution, following the findings of [29] based on analysis of FTP traffic. In our experiments, we use a fixed mean session size of 10 TB with a standard deviation of 10 TB. Due to lack of concrete operational data, these parameters were selected based on anecdotal reports of large data transfer activities [31, 32, 33]. We also note that the offered load is determined by the combination of arrival rate and session size, so having fixed the distribution of the latter we can generate a variety of load conditions by varying the former.

Once the arrival time and session size Z of a lightpath establishment request are determined, we compute the remaining parameters as follows. First, a node pair is selected based on a relative traffic demand matrix, which is generated by filling 0.6 of the entries with uniformly random values on the interval [0.2, 1.2], and setting the remaining entries to zero, as in [34]. Next, the capacity of the equipment used to access the lightpath is determined by assuming that a proportion p of users are equipped with 1 Gb/s hardware and the remaining $1 - p$ use 10 Gb/s hardware. For simplicity, we do this independently of Z, although in reality users with 10 Gb/s equipment may be more likely to initiate larger data transfers. The choice of 1 Gb/s and 10 Gb/s capacities is motivated by the popularity of the corresponding Ethernet technologies, which are

commonly used in conjunction with SONET encapsulation. The requested capacity C in an end-to-end lightpath establishment request is taken to be the minimum of the randomly-determined capacities supported by the two end users. The corresponding session holding time is taken to be Z/C. (For simplicity, we consider here that the throughput during the data transfer is exactly C.)

The initial state of the simulation is an idle network. The simulation period is 300 days. We measure the end-to-end lightpath establishment request acceptance ratio, subsequently referred to simply as the request acceptance ratio, by averaging over ten simulation runs, each run corresponding to a randomly generated traffic demand matrix. Averaging over ten repetitions yields a standard error of the mean of approximately 1%. We vary the request arrival rate λ and the proportion p of users having 1 Gb/s hardware in order to evaluate the benefit of intelligent tie breaking over a range of operating conditions.

Performance Results

The outcome of our simulation study is presented below in Figure 3, which shows the gain in request acceptance ratio when breaking ties by capacity in shortest path computations, versus breaking ties uniformly at random. We found that significant performance gains, up to 39%, were achieved for $p = 0.1$ and $p = 0.05$. This range of parameter values corresponds to the case when most end-to-end lightpath establishment requests are for 10 Gb/s lightpaths and very few are for 1 Gb/s lightpaths. In these cases, if partitions of size 1 Gb/s are allocated unwisely, fragmentation of bandwidth occurs to such extent that requests for 10 Gb/s end-to-end lightpaths often fail unnecessarily. The performance gain disappears as p tends to 0 or to 1, which is expected because in these extreme cases harmful fragmentation of capacity does not occur at all.

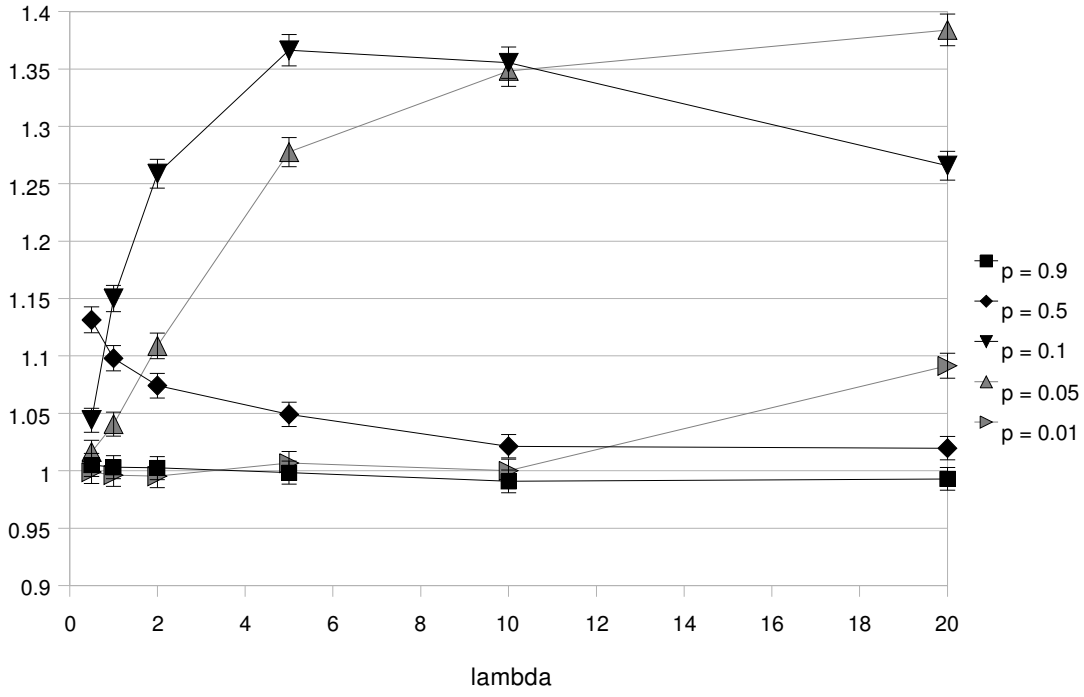


Figure 3: Gain in request acceptance ratio when using intelligent tie-breaking versus uniformly random tie breaking.

6 Summary and Future Work

In this paper we provided solutions for path computation problems relevant to user-controlled optical networks. To find the path that minimizes the time needed to transfer a file of known size, we proposed an algorithm based on repeated connectivity queries, and described efficient implementations using a variety of data structures, including sophisticated dynamic graph algorithms. We also proposed a tie-breaking formula for shortest path computations that answers the need to differentiate between multiple optimal paths in dense graphs. We demonstrated the effectiveness of this formula through simulation, and described how to generalize it to a hierarchy of optimization criteria.

Path computation in user-controlled optical networks poses interesting challenges. The pres-

ence of a time constraint on each lightpath, in the form of an expiry time, is a feature that makes path computations more challenging from an algorithmic point of view. Similarly, efficient data structures and algorithms are needed to cope with logical topologies of lightpaths that may be very dense. We envision future work not only in solving novel path optimization problems, but also in solving existing problems efficiently in the user-controlled scenario.

Acknowledgement

We are grateful to Youssef Iraqi and Tianshu Li for enlightening discussions on topics related to this paper. We also sincerely thank the anonymous reviewers for taking the time to make generously detailed comments and insightful suggestions.

References

[1] B. St. Arnaud and J. Wu, "Customer-controlled and -managed optical networks," *Journal of Lightwave Technology*, vol. 21, pp. 2804–2810, Nov. 2003.

[2] M. Fredman and R. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms," *Journal of the ACM*, vol. 34, pp. 596–615, July 1987.

[3] Q. Ma and P. Steenkiste, "On path selection for traffic with bandwidth guarantees," in *Proc. IEEE International Conference on Network Protocols*, pp. 191–202, Oct. 1997.

[4] R. Guérin, A. Orda, and D. Williams, "QoS routing mechanisms and OSPF extensions," in *Proc. 2nd Global Internet Miniconference (joint with Globecom'97)*, vol. 3, pp. 1903–1908, Nov. 1997.

[5] O. Younis and S. Fahmy, "Constraint-based routing in the Internet: Basic principles and recent research," *IEEE Communications Surveys & Tutorials*, vol. 5, Sept. 2003.

[6] S. Chen and K. Nahrstedt, "An overview of quality of service routing for next-generation high-speed networks: Problems and solutions," *IEEE Network*, vol. 12, pp. 64–79, Nov./Dec. 1998.

[7] W. Alanqar and A. Jukan, "Extending end-to-end optical service provisioning and restoration in carrier networks: Opportunities, issues, and challenges," *IEEE Commun. Mag.*, vol. 42, pp. 52–60, Jan. 2004.

[8] D. Saha, B. Rajagopalan, and G. Bernstein, "The optical network control plane: State of the standards and deployment," *IEEE Commun. Mag.*, vol. 41, pp. S29–S34, Aug. 2003.

[9] Internet Engineering Task Force, "Generalized multi-protocol label switching architecture," *IETF draft, draft-ietf-ccamp-gmpls-architecture-07.txt*, May 2003.

[10] Telecommunication Standardization Sector of ITU, "Recommendation G.8080/Y.1304: Architecture for automatic switched optical network (ASON)," Nov. 2001.

[11] Optical Internetworking Forum, "OIF-UNI-01.0 - User Network Interface (UNI) 1.0 signaling specification," Oct. 2001.

[12] Optical Internetworking Forum, "OIF-E-NNI-01.0 - intra-carrier E-NNI signaling specification," Feb. 2004.

[13] R. Boutaba, W. Golab, Y. Iraqi, and B. S. Arnaud, "Lightpaths on demand: A Web services based management system," *IEEE Commun. Mag.*, vol. 42, pp. 101–107, July 2004.

[14] J. Wu, M. Savoie, H. Zhang, S. Campbell, G. v. Bochmann, and B. St. Arnaud, "Customer-managed end-to-end lightpath provisioning," *International Journal of Network Management*, vol. 15, pp. 349–362, Sept./Oct. 2005.

[15] D. L. Truong, O. Cherkaoui, H. Elbiaze, N. Rico, and M. Aboulhamid, "A policy-based approach for user-controlled lightpath provisioning," *Proc. IEEE NOMS*, vol. 1, no. 19–23, pp. 859–872, April 2004.

[16] J. Recio, E. Grasa, S. Figuerola, and G. Junyent, "Evolution of the user controlled lightpath provisioning system," in *Proc. 7th International Conference on Transparent Optical Networks*, vol. 1, no. 3–7, pp. 263–266, July 2005.

[17] S. van Oudenaarde, Z. Hendrikse, F. Dijkstra, L. Gommans, C. de Laat, and R. J. Meijer, "Dynamic paths in multi-domain optical networks for grids," *Future Gener. Comput. Syst.*, vol. 21, no. 4, pp. 539–548, April 2005.

[18] M. Jemtrud, P. Nguyen, B. Spencer, M. Brooks, S. Liu, Y. Liang, B. Xu, and L. Zhang, "Eucalyptus: intelligent infrastructure enabled participatory design studio," in *WSC '06: Proceedings of the 38th winter simulation conference*, pp. 2047–2054, Dec. 2006.

[19] H. Zhang, M. Savoie, S. Campbell, S. Figuerola, G. von Bochmann, and B. S. Arnaud, "Service-oriented virtual private networks for grid applications," in *Proc. IEEE International Conference on Web Services (ICWS)*, pp. 944–951, July 2007.

[20] M. Blanchet, F. Parent, and B. St. Arnaud, "Optical BGP (OBGP): InterAS lightpath provisioning," *IETF draft, draft-parent-obgp-01.txt*, March 2001.

[21] R. Boutaba, W. Golab, Y. Iraqi, T. Li, and B. S. Arnaud, "Grid-controlled lightpaths for high performance grid applications," *Journal of Grid Computing*, vol. 1, no. 4, pp. 387–394, Dec. 2003.

[22] J. Holm, K. D. Lichtenberg, and M. Thorup, "Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity," *Journal of the ACM*, vol. 48, no. 4, pp. 723–760, July 2001.

[23] M. Henzinger and V. King, "Randomized fully dynamic graph algorithms with polylogarithmic time per operation," *Journal of the ACM*, vol. 46, no. 4, pp. 502–516, July 1999.

[24] L. Roditty, "A faster and simpler fully dynamic transitive closure," in *Proc. Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 404–412, Jan. 2003.

[25] L. Roditty and U. Zwick, "Improved dynamic reachability algorithms for directed graphs," *SIAM Journal on Computing*, vol. 37, no. 5, pp. 1455–1471, 2008.

[26] H. K. Pung, J. Song, and L. Jacob, "Fast and efficient flooding based QoS routing algorithm," in *Proc. Eight International Conference on Computer Communications and Networks*, pp. 298–303, Oct. 1999.

[27] S. K. Kweon and K. G. Shin, "Distributed QoS routing using bounded flooding," in *University of Michigan CSE technical report, CES-TR-388-99*, 1999.

[28] V. Paxson and S. Floyd, "Wide-area traffic: The failure of Poisson modeling," *IEEE/ACM Trans. Networking*, vol. 3, no. 3, pp. 226–244, June 1995.

[29] V. Paxson, "Empirically derived analytic models of wide-area TCP connections," vol. 2, no. 4, pp. 316–336, Aug. 1994.

[30] A. Feldmann, A. Gilbert, and T. Kurtz, "The changing nature of network traffic: Scaling phenomena," *ACM SIGCOMM Computer Communication Review*, vol. 28, no. 2, pp. 5–29, April 1998.

[31] T. Koonen, H. de Waardt, J. Jennen, J. Verhoosel, D. Kand, M. de Vos, A. van Ardenne, and E. J. van Veldhuizen, "A very high capacity optical fibre network for large-scale antenna constellations: the RETINA project," in *Proc. Network and Optical Communications Conference*, pp. 165–172, June 2001.

[32] H. Newmann, M. Ellisman, and J. Orcutt, "Data-intensive e-science frontier research," *Communications of the ACM*, vol. 46, no. 11, pp. 68–77, Nov. 2003.

[33] B. Dobinson, R. Hatem, W. Hong, P. Golonka, C. Meirosu, E. Radius, and B. S. Arnaud, "Transatlantic native 10 Gigabit Ethernet experiments connecting Geneva to Ottawa," in *Proc. 7th IEEE International Conference on High Speed Networks and Multimedia Communications*, pp. 108–119, Sept. 2004.

[34] A. Gençata and B. Mukherjee, "Virtual-topology adaptation for WDM mesh networks under dynamic traffic," *IEEE/ACM Trans. Networking*, vol. 11, no. 2, pp. 236–247, April 2003.

APPENDIX

A Computing a Sorted List of the Unique Elements of an Unsorted List

Recall that Algorithm 1 computes a list $\langle C_1, C_2, \ldots, C_N \rangle$ of distinct elements of the set $\{C(e) \mid e \in E\}$ in descending order, where E is the set of lightpaths and $C(e)$ is the capacity of lightpath e. Let $M = |E|$.

Theorem A.1. *It is possible to compute a sorted list of N distinct elements of an unsorted list of M elements in time $\mathcal{O}(M \log N)$ using a modified merge sort, where N need not be known beforehand.*

Proof. Consider a modified version of the well-known merge sort algorithm, where the merge operation removes duplicate elements. Suppose that the input list of size M contains N distinct elements. From an accounting perspective, we consider that the modified merge sort algorithm consists of $\lceil \log_2 M \rceil$ stages, with stage i operating on at most $2\left\lceil M/2^i \right\rceil$ sublists of size at most 2^{i-1}. For $1 \leq i \leq \lceil \log_2 N \rceil$, the total cost of stage i is $\mathcal{O}(M)$, as in ordinary merge sort, which contributes $\mathcal{O}(M \log N)$ to the total cost. At stage $\lceil \log_2 N \rceil + 1$, the number of sublists is at most

$$2\left\lceil \frac{M}{2^{\lceil \log_2 N \rceil + 1}} \right\rceil \leq 2\left\lceil \frac{M}{2^{\log_2 N}} \right\rceil = 2\left\lceil \frac{M}{N} \right\rceil,$$

and the number of mergers remaining is at most

$$\left\lceil \log_2 \left(2\left\lceil \frac{M}{N} \right\rceil \right) \right\rceil.$$

Since no sublist contains duplicate elements at this point, each sublist has size at most N, and so the number of steps required to perform the remaining mergers is proportional to

$$\begin{aligned} N\left\lceil \log_2 \left(2\left\lceil \frac{M}{N} \right\rceil \right) \right\rceil &\leq N\left(\log_2 \left(2\left(\frac{M}{N} + 1 \right) \right) + 1 \right) \\ &\leq N\left(\log_2 \left(2\left(2\frac{M}{N} \right) \right) + 1 \right) \end{aligned}$$

$$
\begin{aligned}
&= N\left(\log_2 4 + \log_2 \frac{M}{N} + 1\right) \\
&= N\left(\log_2 \frac{M}{N} + 3\right) \\
&= N\left(\frac{\ln \frac{M}{N}}{\ln 2} + 3\right) \\
&\leq 3N\left(\ln \frac{M}{N} + 1\right).
\end{aligned}
$$

Now consider $F(M,N) = N\left(\ln \frac{M}{N} + 1\right)$ where $M, N \in \mathbb{R}^+$. To complete the proof, it suffices to show that $F \in \mathcal{O}(M \log N)$. Looking at the first derivative, we have

$$
\begin{aligned}
\frac{\partial}{\partial N} F(M,N) = 0 \quad &\Longrightarrow \quad \left(\ln \frac{M}{N} + 1\right) + N\left(\frac{N}{M}\right)\left(\frac{-M}{N^2}\right) = 0 \\
&\Longrightarrow \quad \left(\ln \frac{M}{N} + 1\right) - 1 = 0 \\
&\Longrightarrow \quad \ln \frac{M}{N} = 0 \\
&\Longrightarrow \quad \frac{M}{N} = 1 \\
&\Longrightarrow \quad N = M.
\end{aligned}
$$

Looking at the second derivative, we have

$$
\begin{aligned}
\frac{\partial^2}{\partial N} F(M,N) &= \frac{\partial}{\partial N} \ln \frac{M}{N} \\
&= \left(\frac{N}{M}\right)\left(\frac{-M}{N^2}\right) \\
&= -\frac{1}{N} \\
&< 0.
\end{aligned}
$$

Thus, F is concave down and is maximized with respect to N when $N = M$, in which case

$$
F(M,N) = M\left(\ln \frac{M}{M} + 1\right) = M\left(0 + 1\right) = M,
$$

and so $F \in \mathcal{O}(M \log N)$, as wanted. $\square$