A New Approach to Reliable Multi-Path Provisioning

Ananya Das, Charles Martel, Biswanath Mukherjee, and Smita Rai
Department of Computer Science, University of California, Davis, CA 95616
Email: {das, martel, mukherje, rai}@cs.ucdavis.edu

***Abstract*—We study the problem of reliably provisioning traffic in high-capacity backbone mesh networks supporting virtual concatenation (VCAT). Traditional approaches handled reliability requirements using protection schemes. Although protection approaches offer high assurance, this assurance can be costly. We take a less expensive approach to maintaining reliability by offering an alternate guarantee. Specifically, our approach offers *expected* bandwidth, rather than an absolute amount, but at lower costs than full-protection approaches. We propose an improved routing algorithm that uses minimum-cost flow to find efficient collections of paths that satisfy traffic requests for *expected* bandwidth. We find that in most realistic network settings, paths are reliable enough so that only few additional network resources are required to compensate for the bandwidth loss incurred from possible network failures.**

We investigate the performance of our algorithm under both a uniform setting, with symmetric traffic distribution and equal edge capacities, and in a more realistic setting with asymmetric traffic and differing edge capacities. Results show that our algorithm is an attractive approach in both the uniform and non-uniform settings, and much more effective than previously proposed schemes.

We also compare our approach to a full-protection approach. Results show that by using fewer network resources, our algorithm is able to satisfy significantly more traffic requests than the full-protection approach. To get a broader view of how our algorithm performs in networks with varying reliability levels, we test our algorithm for both a network with highly reliable edges and a network with less reliable edges. Results show that our algorithm is very successful in both settings and performs significantly better than the full-protection approach. Thus, when full protection is not crucial, using our routing approach should be beneficial.

I. Introduction

With the rise of critical Internet applications, satisfying customer reliability requirements is becoming increasingly important. A common quality of service (QoS) metric is service reliability. Reliability is measured by connection *availability* - the probability that a connection will be found in the operating state at a random time [1]. This probability is based on both the likelihood of a failure and the time it takes to recover from the failure.

Traditional approaches maintain reliability by using *full-protection* mechanisms such as path *protection* [2], [3], [4], [5], [6], [7] and path *restoration* [8], [9], [10], [11], [5]. Although these approaches guard against connection failures, this assurance can be rather expensive. Previous researchers have considered satisfying customer availability requirements using *single* path routing schemes [12]. More recently, researchers have explored next-generation networks, such as NG-SONET/SDH, supporting virtual concatenation (VCAT) [13] which allows connections to be provisioned on *multiple* paths [14]. Multi-path routing has the obvious advantage of better fault tolerance than single-path routing. Multi-path routing provides more effective utilization of network resources, and relieves edge congestion and delay. Since multiple paths usually exist in backbone mesh networks, multi-path routing can offer more flexibility to the network operator.

Since many decisions for network management must be made in real time, efficient online schemes are essential. In the online QoS provisioning problem that we study, a series of bandwidth requests are issued dynamically and each request must be reliably scheduled (or rejected) as it arrives. Once a request has been scheduled, it cannot be rerouted. If a request cannot be satisfied, it is rejected. In this setting, *degraded service* [15] is acceptable. In other words, the customer is willing to accept a small chance of failure or short time periods of reduced bandwidth. Therefore, customer requests are for an *expected* level of bandwidth rather than an absolute amount. This approach may be useful in settings such as video streaming, where buffering is allowed, and the requested bandwidth is required over an extended period of time rather than all at once. Therefore, intervals with high bandwidth levels can compensate for intervals with lower service.

We propose a new online multi-path heuristic that solves this bandwidth provisioning problem and significantly improves on prior multi-path schemes [14]. While prior heuristics iteratively find *separate* good paths, we present a new approach that finds paths that *jointly* make up the best set. We find that in any realistic network setting, paths are reliable enough so that only few additional network resources are required to compensate for the bandwidth loss incurred from possible network failures. Our algorithm takes advantage of this feature and uses a minimum-cost flow in the network to find an efficient collection of paths that meets the expected bandwidth request. This method allows us to preserve network capacity by consuming as little bandwidth as possible to satisfy a request. We also present an improved version of this algorithm which more effectively utilizes network bandwidth by adapting to network contention. This technique may also improve other routing algorithms by making them more adaptive to changing network conditions.

We first investigate the performance of our algorithms under

This work has been supported by NSF Grant No. CNS-05-20190. A shorter summarized version of this paper was presented at the IEEE Globecom Conference in Nov. 2007.

a uniform setting with a symmetric traffic distribution and equal edge capacities. We then extend our study to a non-uniform setting with asymmetric traffic and differing edge capacities. The latter results give us a better understanding of how the algorithms would perform in practice. Our simulation results show that by finding sets of paths that preserve network capacity, our algorithm is highly successful. Our algorithm performs significantly better than prior schemes in both uniform and non-uniform settings. The improvement of our algorithm over prior schemes in the non-uniform setting indicates that our approach would be much more effective in practical network settings.

As previously described, our algorithms are applicable in settings in which the customers are willing to accept a small chance of failure. Full-protection approaches take measures to guard against *any* connection failures. However, protection requires significant network resources which may result in a decline in overall performance. On the other hand, our algorithms satisfy connection requests while using noticeably fewer resources than protection schemes. To understand the cost of full protection on performance, we compared our algorithms to an efficient full-protection multi-path algorithm proposed in [16][1]. This algorithm provides full protection against edge failures by protecting each primary path with an edge-disjoint backup path (we describe this algorithm in more detail in Section IV-B). Although our approach does not provide full protection, our simulation results show that it can schedule more requests than the full-protection scheme. These results verify that there exists a significant trade-off between full protection and performance. Therefore, if a customer is willing to accept a small chance of failure (or reduced service), then using our approach would be beneficial.

To assess the effectiveness of our algorithms in networks with varying reliability levels, we simulate both a network with highly reliable edges and a network with moderately reliable edges. Under both settings, we find that our approach is very effective. For a typical US nationwide topology with reliable edges and uniform traffic distribution, under a moderate load of 300 Erlangs, our algorithm satisfies more than 99.3% of the requests and more than 97% of the requested bandwidth. Under a load of 270 Erlangs, our algorithm blocks less than half the bandwidth blocked by the full-protection approach.

To summarize, the major contributions of this paper are as follows:

- We present a new algorithm that solves the reliable multi-path provisioning problem and outperforms previous approaches. Our algorithm uses a new approach that seeks a target bandwidth which is slightly more than the requested amount. It also aims to reduce the amount of bandwidth consumed to satisfy requests.
- We also compare our algorithm to a full-protection approach, and demonstrate the benefits of our algorithm for settings where degraded service is acceptable.
- We present an enhanced version of our algorithm which better utilizes network resources and adapts to contention by limiting the overuse of edges.
- We measure the performance of our algorithm under various network settings. We first consider a reliable setting with uniform edge capacities and source-destination requests. We also study a non-uniform setting with varying edge capacities and source-destination requests. Finally, we consider a less reliable network setting where edges are more likely to fail.

[1]Two full-protection algorithms are proposed in [16]; our comparison is with the PIVM (Protecting Individual VCG Member) algorithm.

The remainder of this paper is organized as follows: Section II describes other reliability-aware provisioning approaches. Section III-A describes the QoS problem we are investigating and presents our multi-path routing algorithms. Sections IV-V provide our simulation results. Finally, Section VI presents our conclusions.

II. Related Work

Edge failures are the predominant type of failure in communication networks. Node failures are relatively rare because carrier-class nodes typically have their own failure-handling mechanisms. Therefore, various routing schemes have been proposed that maintain reliability by handling edge failures.

a) Path Protection and Restoration: There is an extensive amount of literature on path protection [2], [3], [4], [16], [5], [6], [7], and path restoration [8], [9], [10], [11], [5]. For a thorough review of these techniques please refer to [17]. Path protection, a proactive procedure in which spare capacity is reserved during connection setup, can be employed to handle network failures. A path that carries traffic during normal operation is known as the *primary* path. When an edge on the path fails, the connection is rerouted over an edge-disjoint *backup* path. In dedicated path protection, a backup path cannot be shared by multiple primary paths. However, in shared protection, multiple primary paths may share a common backup path if the primary paths are edge-disjoint.

Path restoration is a reactive procedure in which backup paths are discovered *after* a failure occurs on a primary path. Restoration schemes take more time to restore a connection than protection schemes since recovery is performed after the failure occurs. Therefore, for time-critical applications, protection schemes are often a more attractive approach.

b) Multi-Path Routing: Multi-path routing offers more fault tolerance and flexibility than single-path schemes and is therefore an attractive approach for bandwidth provisioning. The authors of [14] propose an online multi-path routing scheme to solve the bandwidth provisioning problem for the setting in which degraded service is acceptable (in Section III-A, we describe the algorithm and this setting in more detail). Given a source, destination, and bandwidth request, this algorithm uses a "smart-greedy" approach to find a set of paths that satisfies the request. The algorithm first selects a "good" set of candidate edges. Among these edges, it then iteratively looks for the highest availability path. One drawback of this algorithm is that the highest availability path may not always be the most effective choice. Furthermore, this approach does not take full advantage of the multi-path feature.

We propose a new multi-path heuristic to solve the bandwidth provisioning problem. While the approach in [14] focuses on finding highly reliable edges, our algorithm exploits the fact that in realistic networks, typically all edges are reliable enough to satisfy requests. Our heuristic finds the best set of paths to satisfy requests by seeking paths that minimize the amount of bandwidth consumed from the network. It also takes advantage of the multi-path feature by finding paths that *jointly* make up the best set.

III. Provisioning Expected Bandwidth Problem

A. Problem Description and Statement

We refer to the multi-path provisioning problem that we study as EXPBAND . In this problem, bandwidth requests are issued dynamically and each request must be scheduled as it arrives [14]. In this network setting, customers are willing to accept a small chance or period of time of *degraded service*. In other words, customers are satisfied with an *expected* level of bandwidth, rather than an absolute amount. Each connection request consists of a source, destination, and *expected* bandwidth requirement. The goal of EXPBAND is to find a path or set of paths from the source to the destination which satisfies the bandwidth requirement.

We now provide the definition for expected bandwidth. Let $G = (V, E)$, be a directed graph where each edge e in E has an availability $a \in (0, 1)$ and a non-negative integer capacity. For a path P in G, with k edges $e_1, e_2, \ldots e_k$, let $a_1, a_2, \ldots a_k$ denote the respective availabilities of these edges, where a_i is the probability that edge e_i is properly operating. The availability of path P is $A = a_1 \cdot a_2 \cdot \ldots \cdot a_k$. Assuming that all edges fail independently, A is the probability that P is properly functioning. If the respective capacities of the edges of P are $c_1, c_2, \ldots, c_k$ then c_i is the maximum amount of bandwidth that can be allocated to edge e_i and $c_{min} = min_{1 \leq i \leq k}(c_i)$ is the maximum amount of bandwidth that can be allocated along P. If we route b units[2] of bandwidth on P, where $b \leq c_{min}$, then the expected bandwidth of P is $A \cdot b$. Given a set of paths $\pi = \{P_1, P_2, \ldots, P_m\}$ with respective availabilities $A_1, A_2, \ldots, A_m$, if $b_1, b_2, \ldots, b_m$ units of bandwidth are sent along these paths, then the total expected bandwidth of π is $\sum_{i=1}^{m} A_i \cdot b_i$. Note that the paths in π need not be edge-disjoint.

Problem Statement

The EXPBAND problem takes as input a directed graph $G = (V, E)$, where each edge in E has an availability $a \in (0, 1)$ and a non-negative integer capacity; and a connection request $< s, d, b >$, where $s, d \in V$, s is the source node, d is the destination node, and b is the expected bandwidth requirement. The goal is to find a set of paths from s to d such that the expected bandwidth (defined above) from s to d is $\geq b$.

B. The Smart-Greedy Algorithm

Since even the off-line version of EXPBAND is **NP**-hard (please see [14] for the proof of hardness), efficient heuristics are needed to solve this problem. The authors of [14] developed a heuristic that we refer to as SMART-GREEDY, which first finds a set of candidate "good" edges. Among these edges, it then iteratively seeks the path of highest availability until the set of paths found provides enough expected bandwidth or until no more paths can be found. The algorithm outputs a single path or set of paths that satisfies the expected bandwidth request if one exists; otherwise the request is rejected and no paths are assigned. The experimental network used in [14] was a US nationwide network topology which resembles a well-connected carrier's backbone topology [18] (see Fig. 1). The edges were bidirectional and edge availabilities were uniformly distributed over the values [0.9999, 0.99999, 0.999999]. Since edge availabilities are less than 1, all path availabilities will also be less than 1. Therefore, to satisfy an expected bandwidth request of b units, it is always necessary to consume at least $b + 1$ units (we will show in Section III-C that in this setting, exactly $b + 1$ units will be sufficient).

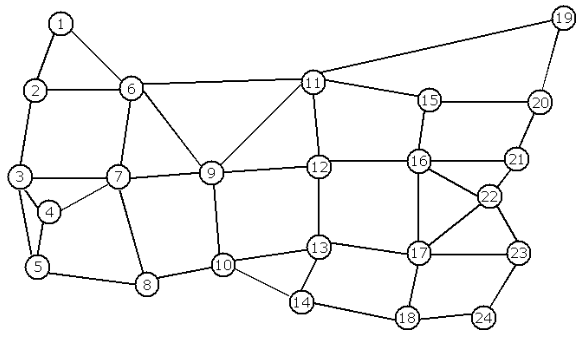

Fig. 1. Sample network topology.

Figure 2 shows an example of the SMART-GREEDY approach. Edges on the graph that have a non-zero flow are indicated by dashed lines and the flow amounts are shown below the edges. The capacity on all the edges is 10 and the availabilities on all edges except (s,a) and (s,b) is 0.999999. Edge (s,a) has availability 0.99999 and edge (s,b) has availability 0.9999. For this example, suppose the request *<s, d, 11>*, has been issued[3]. Since all edge availabilities are less than 1, a request for 11 units of expected bandwidth would require at least 12 units of total bandwidth. SMART-GREEDY will first choose the path with highest availability, *s-c-g-h-d*, even though it is the longest path. It will send 10 units of flow along this path. In the next iteration, it will choose path *s-a-f-d* and send 2 units of flow along this path. However, this set of paths is not the ideal set since the request can be satisfied without using the longest path.

We found two drawbacks with the SMART-GREEDY heuristic. First, the maximum availability path is not always the shortest path[4]. Using the shortest path that satisfies the bandwidth request usually allows us to consume the minimum amount of network bandwidth required to satisfy the request. Therefore, if a shorter path (which is slightly less reliable) provides enough bandwidth for the given request, it should

[2]One unit of bandwidth in a SONET-based network is STS-1 ($\approx$ 51.84 Mbps).

[3]Assume that bandwidth need is deterministic so that all the bandwidth on an edge can be used as in a time-division multiplexing (TDM) edge.

[4]The length of a path is the number of edges in the path.

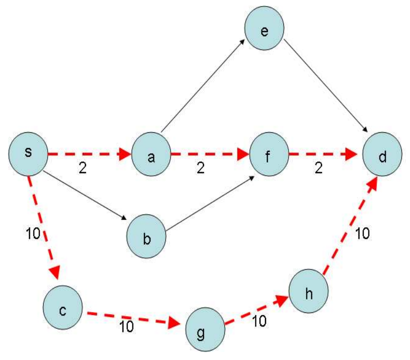

Fig. 2. **Example of SMART-GREEDY scheme**. Each edge has capacity 10 units and the availabilities on all edges except (s,a) and (s,b) is 0.999999. Edge (s,a) has availability 0.99999 and edge (s,b) has availability 0.9999. To route request <*s, d, 11*>, SMART-GREEDY would consume 46 units.

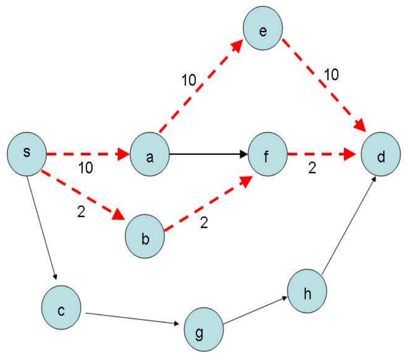

Fig. 3. **Example of MINCOST scheme**. Each edge has capacity 10 units and the availabilities on all edges except (s,a) and (s,b) is 0.999999. Edge (s,a) has availability 0.99999 and edge (s,b) has availability 0.9999. To route request <*s, d, 11*>, SMART-GREEDY would consume 36 units.

be used over a longer path with higher availability. Second, the SMART-GREEDY algorithm does not take full advantage of the multi-path feature. In many cases, although the maximum availability path may be the best *single* path, it may not be part of a *set* of the best paths.

We propose a new heuristic to address the EXPBAND problem. Our algorithm, MINCOST, takes advantage of the fact that using shorter paths to satisfy each request will reduce the amount of bandwidth consumed from the network (see Fig. 3). It also exploits the use of multi-paths by finding the set of *globally* optimum paths that satisfies the current bandwidth request.

C. Satisfying Expected Bandwidth

While the SMART-GREEDY algorithm seeks the highest availability path, our algorithm aims to find the shortest paths that can satisfy the requests. Given the network settings used in [14] (which are described as being similar to realistic observed settings), this approach of not prioritizing availabilities maintains practicality. In our experimental runs using the same topology, we found that paths used to satisfy requests had average length 3. A path of length 3, with each edge having the lowest possible availability (0.9999), would have an overall availability of 0.9999×0.9999×0.9999 = 0.9997, which is still close to 1 (in Section V, we discuss a setting with lower edge availabilities). For the settings used in [14], we find that to satisfy b units of expected bandwidth, $b+1$ units will always be sufficient[5]. For example, if 192 units of bandwidth are requested, and a path with availability 0.9997 is found, then by retrieving 193 units of bandwidth from this path we will obtain an expected bandwidth of 193×0.9997 = 192.9421, which is more than enough to satisfy the request. The high edge availabilities and the short path lengths that are characteristic of the network topology generally allow for paths with high availabilities. Therefore, for our network setting, retrieving one additional unit of bandwidth is enough to compensate for the bandwidth loss incurred from the fractional availabilities. The SMART-GREEDY approach of finding highly reliable paths to deal with this loss is rather unnecessary and can be avoided by simply retrieving an additional unit of bandwidth. Furthermore, we found that in most realistic networks (see Section V), paths are reliable enough so that only a few additional units of bandwidth are required to satisfy requests.

D. The MINCOST *Algorithm*

Given a connection request, the MINCOST algorithm satisfies the request by finding a single path or set of paths that will result in the minimum overall bandwidth consumption from the network. The algorithm first sets the cost of each edge in the underlying graph to 1. If edges $e_1, e_2, \ldots, e_k$ with costs $w_1, w_2, \ldots, w_k$ are assigned new flows $f_1, f_2, \ldots, f_k$, then the cost of this flow is $\sum_{i=1}^{k} w_i \cdot f_i$. Given a connection request <*s, d, b*>, the MINCOST algorithm finds a single path or set of paths that forms the minimum-cost flow from s to d of value $b+1$. As noted earlier, a flow of $b+1$ will satisfy this request. By finding the minimum-cost flow, we are able to minimize the sum of the flows in all the edges used for this request [19].

Figure 3 shows MINCOST's approach for the previous example. Assume the same edge availabilities and capacities, and that the same request <*s, d, 11*> has been issued as in Fig. 2. The MINCOST algorithm sends 10 units of flow along path *s-a-e-d*, and 2 units along path *s-b-f-d*. Although both SMART-GREEDY and MINCOST are able to satisfy the request, MINCOST does so by consuming 10×3+2×3=36 units of bandwidth whereas SMART-GREEDY consumes 10×4+2×3=46 units.

Note that in the same example, if a request for slightly higher bandwidth, for example 22 units, had been issued instead, MINCOST would satisfy it by sending 10 units along path *s-a-e-d*, 10 units along path *s-b-f-d* and 3 units along path *s-c-g-h-d*. However, SMART-GREEDY would reject this request.

The steps of our algorithm are shown in **Algorithm 1**. Our approach can be implemented efficiently using a simple minimum-cost flow algorithm [19], and in our simulations the average number of paths needed was only 1.2 (see Fig. 1).

[5]The maximum amount of bandwidth requested is 192 units. Therefore, as long as the path length is less than $k = log_{(0.9999)} \frac{192}{193} = 51$, $b+1$ units will be sufficient to satisfy a request for b units.

Algorithm 1. MINCOST($< s, d, b >, G = (V, E), C : E \to Z^+, A : E \to (0, 1)$)

```
1: Assign each edge $e \in E$ a cost $w(e) = 1$.
2: Find a single path or set of paths, $\pi$, that makes up the minimum-cost flow from $s$ to $d$ of value $b + 1$.
3: **if** Such a set exists **then**
4:     Reduce the capacity of every edge in $\pi$ by its new flow. Provisioning Successful.
5: **else**
6:     Reject this request.
7: **end if**
```

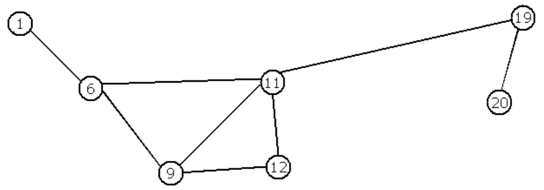

Fig. 4. Subgraph of sample topology.

Although our algorithm requires a bit more computation than MAXFLOW, our simulation[6] results show that even without any code optimizations, both algorithms take only a few milliseconds to process a request. The difference in computation time is a small tradeoff for the performance improvements achieved by MINCOST.

E. *The* MINCOSTADD *Algorithm*

Congestion is a common problem that arises when a series of network requests is issued. Edges that lie on the shortest path for many node pairs will be used frequently. To avoid congestion, the use of these edges should be limited. For example, in Fig. 4, edge (6, 11) lies on the shortest path between several node pairs such as: 1 and 19, 1 and 20, and 6 and 19. In general, popular edges like (6, 11) should be saved when it is still possible to efficiently (i.e., without using significantly more edges) satisfy a request with a less popular edge. If a request between nodes 1 and 12 was issued, then to preserve bandwidth on edge (6, 11), path *1-6-9-12* should be chosen over the path containing edge (6, 11).

This idea of preserving frequently accessed edges was our motivation for modifying the MINCOST algorithm. In the modified algorithm, MINCOSTADD, each time an edge is used, we increase its cost. Therefore, popular edges will have higher costs and will not be used when short alternate routes exist. This modification produces improved performance with only minor additional computational costs.

Adjusting the costs of edges allows us to easily tune the algorithm based on the network topology and frequency of edge accesses. This feature of the MINCOSTADD algorithm allows it to better utilize network resources by adapting to changing network conditions. This simple yet effective technique can also improve other routing algorithms by making them more adaptive [15].

[6]Our simulations were run on a personal computer with a 1.7-GHz Pentium M processor and 2GB of RAM.

IV. Illustrative Numerical Results

A. *Comparing to Smart-Greedy Algorithm*

1) Uniform Setting: To evaluate the performance of our algorithms, we first replicated the simulated dynamic network environment used in [14]. We assume the network is fully wavelength-convertible. The connection arrival process is Poisson and the connection-holding time follows a negative exponential distribution with unit mean. There are 16 wavelengths per edge, and the capacity of each is OC-192 ($\approx$10 Gbps), which is a realistic measure for today's channel speeds. The bandwidth distribution of the connection requests is as follows: 52% of the requests are for 100 Mbps of bandwidth, 21% are for 150 Mbps, 10% are for 600 Mbps, 10% are for 1 Gbps, 4% are for 2.5 Gbps, 2% are for 5 Gbps, and the final 1% of requests are for 10 Gbps of bandwidth (the bandwidth is assumed to occupy an integral number of slots with STS-1 granulariy). This distribution follows typical bandwidth distributions observed in realistic networks. As in [14], we assume that the lowest traffic granularity is STS-1, and therefore ignore connection requests with bandwidth less than or equal to STS-1 since they cannot be efficiently provisioned over multiple paths.

In the first set of simulations, we assume a uniform traffic distribution over all node pairs. For these simulations, the availability of edges were assumed to be uniformly distributed over the values [0.9999, 0.99999, 0.999999]. We simulated 100,000 connection requests under these settings for various load levels[7]. We tested each algorithm while varying the load on the network from 100 Erlangs to 600 Erlangs.

We applied the MINCOST and MINCOSTADD algorithms and compared their performance to the SMART-GREEDY algorithm. We observed the fraction of unprovisioned bandwidth (bandwidth blocking probability) and the fraction of unprovisioned requests (probability of failure).

For all load levels, SMART-GREEDY is outperformed by both of our algorithms. Our algorithms consistently provision more bandwidth and satisfy a higher number of requests. For moderate load (300 Erlangs), SMART-GREEDY blocks 16% of the requested bandwidth, MINCOST blocks 11%, and MINCOSTADD blocks 9%. Therefore, MINCOST improves the bandwidth blocking probability by 5% and blocks less than 70% of the bandwidth blocked by SMART-GREEDY . MINCOSTADD improves the bandwidth blocking probability by 7% and blocks less than 56% (see Fig. 5) of the bandwidth blocked by SMART-GREEDY . Under this load, SMART-GREEDY is 1.7 times more likely to fail than MINCOST and twice as likely to fail than MINCOSTADD (see Fig. 6). Under a load of 200 Erlangs, our algorithms are almost always successful (at most 3 failures), whereas SMART-GREEDY has many more failures (approximately 520). At low loads (100 Erlangs), our algorithms are able to successfully provision *all* of the requested bandwidth, whereas SMART-GREEDY is unsuccessful at times.

The results illustrate the effectiveness of our algorithms. Even under a moderate load level (300 Erlangs), our algo-

[7]Load, measured in Erlangs, is defined as the product of the connection-arrival rate, the average connection-holding time, and a connection's average bandwidth normalized in the unit of OC-192.

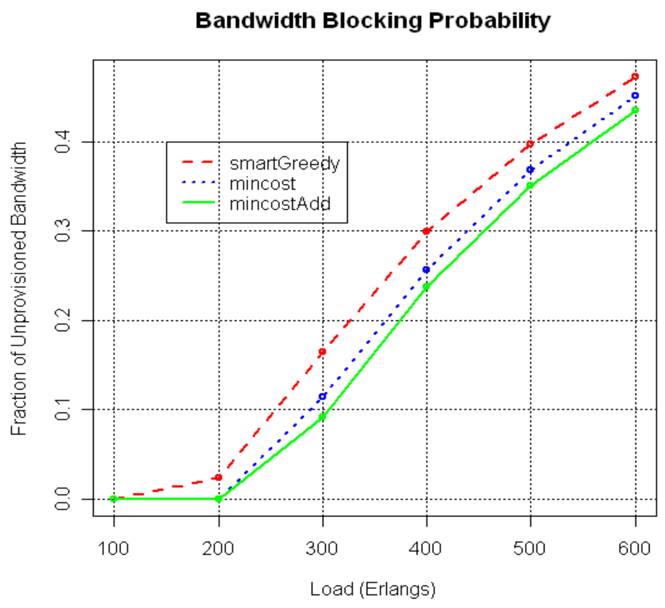

Fig. 5. Fraction of bandwidth blocked in uniform setting.

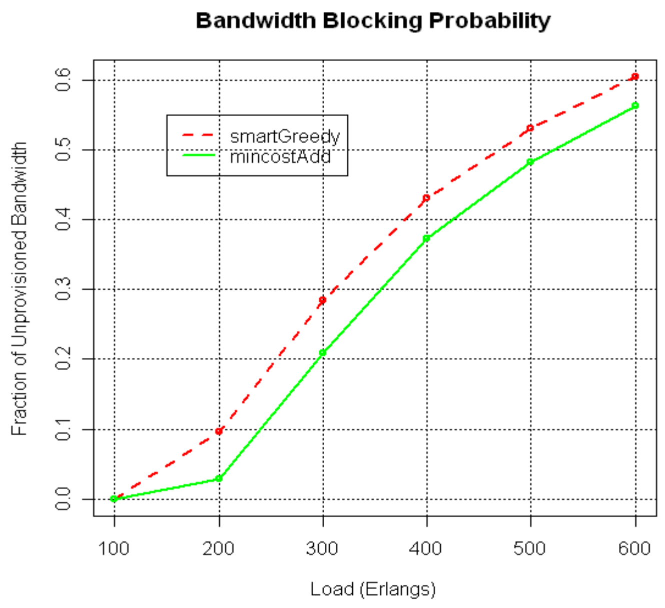

Fig. 7. Fraction of bandwidth blocked in non-uniform setting.

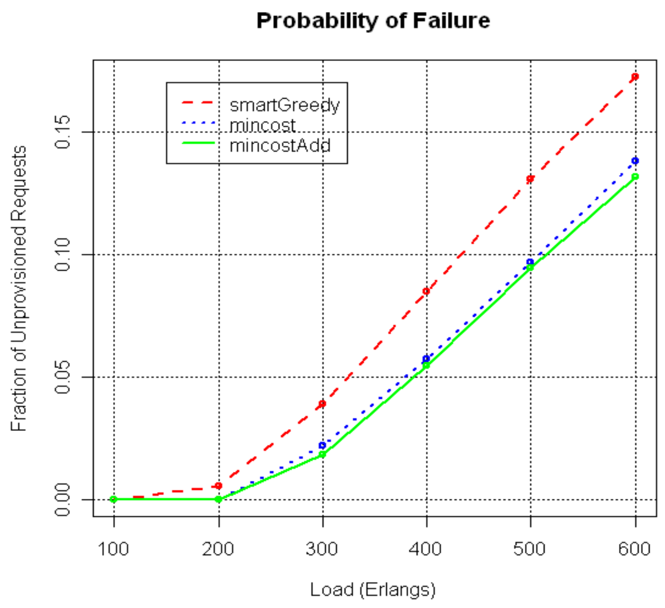

Fig. 6. Fraction of requests blocked in uniform setting.

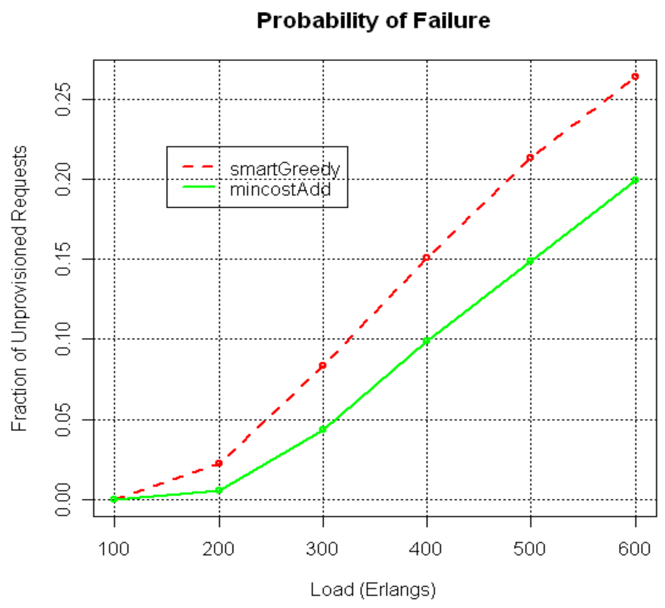

Fig. 8. Fraction of requests blocked in non-uniform setting.

rithms satisfy more than 99.3% of the requests and more than 97% of the requested bandwidth.

2) Non-Uniform Setting : Much of the published work in routing studies assumes that requests are uniformly distributed among all node pairs. In realistic networks, this assumption clearly does not hold. Certain popular sites are more likely to be selected for a connection request, whereas other sites will be selected less frequently. To understand the relative performance of SMART-GREEDY and our algorithms under more realistic conditions, we ran simulations under a non-uniform setting. We used the same network topology as in our uniform experiments. However, in this new setting, we placed a bias on certain nodes by forcing them to be selected more frequently for the *s-d* pairs. These biased nodes are referred to as "large" nodes, and all other nodes are referred to as "small". We followed the guidelines suggested by the Defense Advanced Research Projects Agency (DARPA) [20] model and set 20% of the nodes to be "large". For this set, we chose nodes 1, 3, 11, 21, and 22, since these nodes correspond to major U.S. cities (see Fig. 1). The remaining 80% of nodes were "small". Following the guidelines, the traffic was distributed as follows: 40% of the traffic was between two large nodes, 40% of the traffic was between a large node and a small node, and the remaining 20% of the traffic was between two small nodes.

Since network operators are likely to allocate more bandwidth to edges that are adjacent to popular sites, we assigned edges adjacent to large nodes twice as much bandwidth (32 wavelengths, each at $\approx$10 Gbps) as other edges. All other settings in the non-uniform experiments were kept the same as for the uniform case.

Figures 7 and 8 show our results under the non-uniform setting. For simplicity, we only compare SMART-GREEDY to MINCOSTADD since the latter consistently performs better than MINCOST (however, both of our algorithms outperform SMART-GREEDY). The performance of MINCOSTADD is still impressive in this non-uniform setting. When the network is considerably loaded (at 300 Erlangs), the algorithm successfully schedules more than 99% of the requests and 92% of the requested bandwidth. MINCOSTADD's ability to adapt to varying edge demands accounts for its effectiveness in this setting.

Our simulation results show that even in the non-uniform setting, for all loads, MINCOSTADD performs better than SMART-GREEDY. Studies done under purely uniform settings may be misleading in measuring relative performance since the uniformity assumption is usually unrealistic. Our results are more convincing as they show that our algorithms are highly effective in *both* uniform and non-uniform settings.

B. Comparing to Full Protection Algorithm

Our proposed algorithms handle failures by satisfying requests for an *expected* level of bandwidth. These algorithms are therefore useful in settings in which the user is willing to accept a small chance of connection failure or reduced bandwidth. Unlike path protection and path restoration, our approaches do not provide full protection against single-edge failures. If a failure occurs on an edge, the connection will continue at a reduced rate (if provisioned over multiple paths) or be blocked for a short time (if provisioned over a single path) until the edge recovers. In protection and restoration schemes, backup paths are allocated for traffic to be rerouted in case a failure occurs on a primary path. Although these schemes offer high levels of assurance, this assurance comes at a cost. Allocating backup paths for handling failures requires more network resources, and can therefore be quite expensive. Even when several primary paths share a backup path, significant network resources are still required for the backup paths [16]. Furthermore, allocating edges for backup paths will prohibit these edges from being used as primary paths. Since fewer primary routing edges are available, full-protection schemes have less flexibility for routing future requests.

Identifying the costs of full protection on performance can help network providers determine their service and pricing policies. If providing full protection causes a significant decline in service, providers may decide to offer customers expected bandwidth instead, at a lower cost. To get an understanding of how maintaining full protection affects performance, we compared MINCOSTADD to a multi-path full-protection algorithm proposed in [16]. Their heuristic provides full protection against edge failures by protecting each primary path with an edge-disjoint backup path. The heuristic achieves high backup sharing by allowing primary paths to share the backup capacity. More specifically, it achieves maximum intra-connection sharing (multiple paths for one connection share backup capacity) and very high inter-connection sharing (different connections between different node pairs share backup capacity).

To compare MINCOSTADD to the full-protection algorithm, we replicated the network environment used in [16], which is similar to the previous network setting we simulated in Section IV-A1. There are two differences between the setting used in [16] and our previous simulation setting. First, in [16] the bandwidth distribution (which does not ignore requests for STS-1) is as follows: 51.5% of the requests are for 50 Mbps of bandwidth, 25% are for 100 Mbps, 10% are for 150 Mbps, 5% are for 600 Mbps, 5% are for 1 Gbps, 2% are for 2.5 Gbps, 1% are for 5 Gbps, and the final 0.5% of requests are for 10 Gbps of bandwidth. This distribution also follows a typical bandwidth distribution observed in realistic networks. Second, in [16] the load varies from 170 Erlangs to 270 Erlangs. Since the authors of [16] assume a uniform traffic distribution over all node pairs, we follow this assumption for our simulations.

Figure 9 shows the comparison results. Again for simplicity, we only show the results of MINCOSTADD since it consistently performs better than MINCOST (although both algorithms outperform the full-protection algorithm). Even at high loads (270 Erlangs), MINCOSTADD blocks less than half the bandwidth blocked by the full-protection algorithm. Specifically, at load 270 Erlangs, the full-protection algorithm blocks 15% of the requested bandwidth while MINCOSTADD blocks less than 7%. Therefore, MINCOSTADD improves the bandwidth blocking probability by 8% and blocks less than 45% of the bandwidth blocked by the full-protection algorithm. Furthermore, when the load ranges from 170-190 Erlangs, MINCOSTADD successfully provisions all requests while the full-protection algorithm has some failures.

C. Requests for Less than STS-1

As previously discussed, we replicated the network setting used in [16] to compare MINCOSTADD with the full-protection algorithm proposed in [16]. In this network setting, the minimum granularity for transmittable bandwidth is assumed to be STS-1 ($\approx$ 50 Mbps). In the bandwidth distribution used in this setting, more than half of the requests are for STS-1. Since the MINCOSTADD always seeks one additional unit of bandwidth than what is requested, it satisfies requests for 1 unit by retrieving 2 units. This means that for more than half the requests, MINCOSTADD provides twice the bandwidth that is actually requested. Therefore, using this bandwidth distribution to compare MINCOSTADD and the full-protection algorithm yields results which are skewed against MINCOSTADD.

Since the minimum granularity assumed in [16] is STS-1, it is reasonable to presume that some requests that are labeled as STS-1 requests may actually require less bandwidth than this. Furthermore, if a network operator is willing to

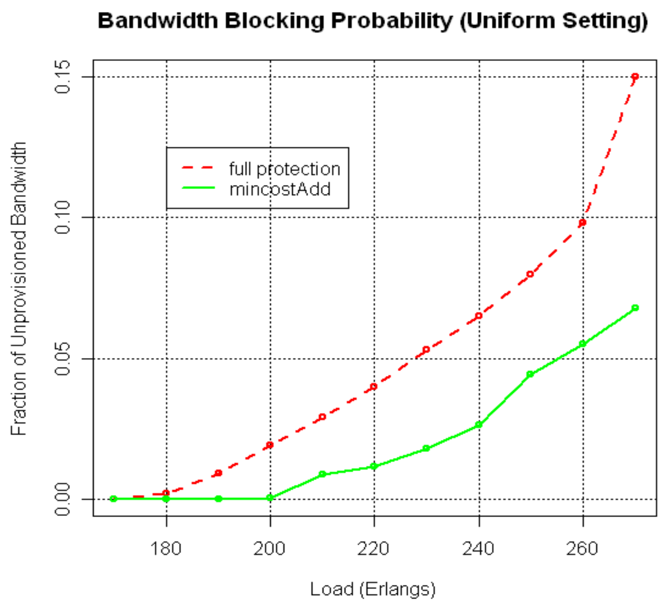

Fig. 9. Fraction of bandwidth blocked (MINCOSTADD vs. full-protection approach).

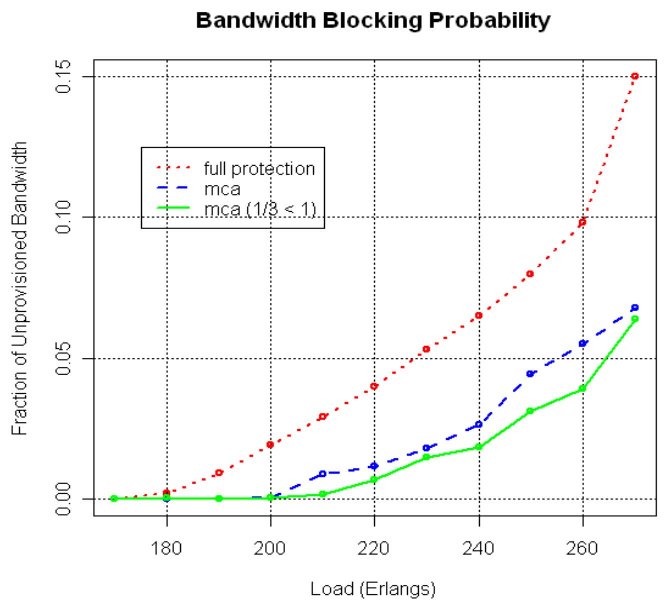

Fig. 10. Fraction of bandwidth blocked (MINCOSTADD vs. full-protection approach). Lowest curve represents performance of MINCOSTADD in the setting with requests for less than STS-1.

provide bandwidth amounts of less than STS-1 at a lower cost, more customers may issue requests for less than STS-1. An advantage of MINCOSTADD is that it may be able to satisfy many of these requests by retrieving *exactly* one unit of bandwidth. For example, suppose a request for 45 Mbps of bandwidth (0.9 units) is issued. If MINCOSTADD finds a path of length 3, with each edge having the lowest possible availability (0.9999), then by sending 1 unit along this path, it will be able to provide $0.9999 \times 0.9999 \times 0.9999 \times 1 = 0.997$ units of expected bandwidth, which is more than enough to satisfy the request. Although the full-protection algorithm could also satisfy more requests in a setting in which some requests are for less than STS-1, doing so would require complex grooming techniques [21], [22].

To achieve a fairer comparison of MINCOSTADD and the full-protection algorithm, we evaluated the two approaches under a setting with bandwidth requests for less STS-1. More specifically, for this setting, we assume that a third of the requests are for 0.9 units of bandwidth[8]. Figure 10 compares MINCOSTADD to the full-protection algorithm in this new setting and the previous setting used in [16].

The results show that MINCOSTADD satisfies even more bandwidth under this new setting. For example, at a moderate load of 210 Erlangs, MINCOSTADD blocks less than 20% of the bandwidth blocked under the original setting. At this same load, MINCOSTADD blocks less than 5% of the bandwidth blocked by the full-protection algorithm. These results show that if some customers are content with less than 1 unit of bandwidth, our algorithm can satisfy more customer requests while using fewer network resources.

[8]Even if the minimum edge availability is 0.999, MINCOSTADD can satisfy 0.9 unit requests by retrieving 1 unit as long as the path length is less than 105.

V. LOWER AVAILABILITY SETTING

As previously described, the MINCOSTADD algorithm ignores path availabilities when seeking paths to satisfy connection requests. As discussed in Section III-C, we can satisfy a request for b units of expected bandwidth by retrieving $b + 1$ units since our network setting assumes high edge availabilities (uniformly distributed over [0.9999, 0.99999, 0.999999]). These high availabilities are found in realistic networks [14], however some networks may be less reliable [23]. In these networks, retrieving $b+1$ units may not be sufficient. To understand how MINCOSTADD would perform in a less reliable network setting, we simulated a network with lower edge availabilities. More specifically, in this new setting, edge availabilities were uniformly distributed among the values [0.999, 0.9999, 0.99999]. Under this new setting, we first compared MINCOSTADD to SMART-GREEDY and then to the full-protection algorithm.

Using lower edge availabilities not only allows us to assess the performance of our approach in a less reliable setting, it also provides a less skewed comparison of the algorithms. Although SMART-GREEDY does not perform as well as MINCOSTADD, it satisfies requests using the most reliable path(s) that it finds. On the other hand, MINCOSTADD ignores the reliability of the paths and aims to consume as little bandwidth as possible. In a setting with high edge availabilities, our approach is resource-efficient and effective. However, in a less reliable network, ignoring edge availabilities may result in connections which are not reliable enough for customer standards. Similarly, although the full-protection algorithm does not satisfy as many requests as our approach, it provides full protection against single-edge failures. Therefore, this less reliable setting provides a broader view of how our

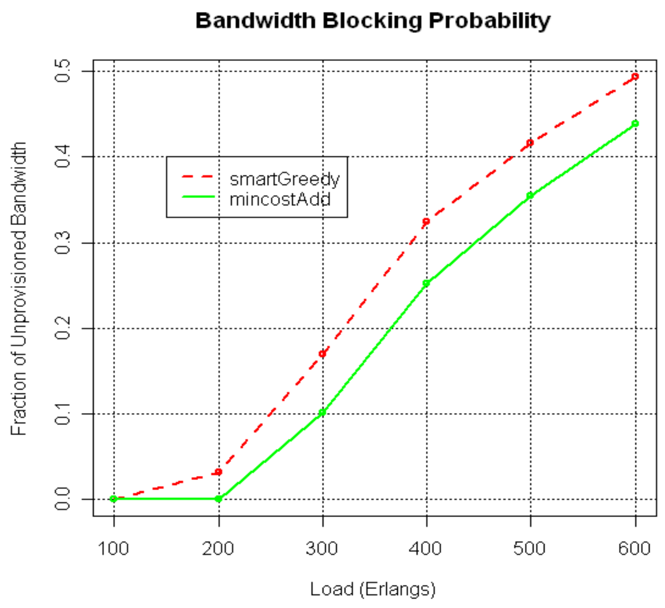

Fig. 11. Fraction of bandwidth blocked: MINCOSTADD vs. SMART-GREEDY in the less reliable network (uniform setting).

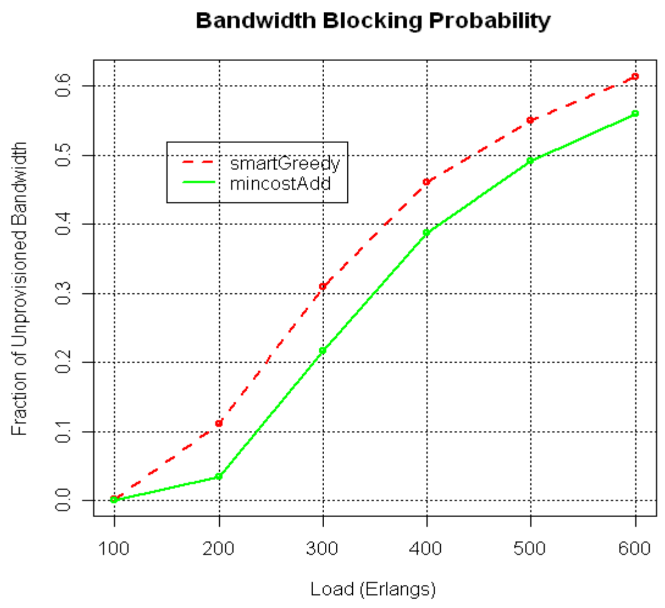

Fig. 12. Fraction of bandwidth blocked: MINCOSTADD vs. SMART-GREEDY in the less reliable network (non-uniform setting).

algorithm, which allows a small chance of failures, compares to approaches which rarely or never allow failures.

A. *Comparing to* SMART-GREEDY

Under this less reliable setting, we first compare MINCOSTADD to SMART-GREEDY, assuming a uniform traffic distribution and uniform edge capacities. Since in this setting the availabilities of edges are lower, retrieving $b + 1$ units of bandwidth may no longer satisfy a request for b units of expected bandwidth. However, given our network topology and bandwidth distribution (both of which are reflective of realistic network settings) we found that at most $b + 9$[9] and on average only $b+1.16$ units of bandwidth must be retrieved in this less reliable setting.

Figure 11 shows the results of our simulations in this setting. Although the network is less reliable in this setting, MINCOSTADD is still able to perform significantly better than SMART-GREEDY. Under moderate load (300 Erlangs), MINCOSTADD blocks less than 60% of the bandwidth blocked by the SMART-GREEDY algorithm.

Figure 12 shows the results of our simulations for the less reliable network under a non-uniform setting. We assume the same edge capacities and traffic distribution as described in Section IV-A2. Under a load of 300 Erlangs, MINCOSTADD blocks less than 54% of the bandwidth blocked by SMART-GREEDY. Even under a much higher load of 600 Erlangs, our algorithm still blocks less than 84% of the bandwidth blocked by the prior algorithm. For this less reliable network, our algorithm performs significantly better than the SMART-GREEDY approach. Our results show that our algorithm's approach of ignoring availabilities and seeking paths that minimize the bandwidth consumed is still effective in a less reliable network setting.

[9]b+9 units are required for requests with the maximum bandwidth requirement of 192 units (10 Gbps) and a path of availability 0.95.

B. *Comparing to Full Protection*

We now present our simulation results comparing MINCOSTADD to the full-protection algorithm in the less reliable network setting. For this comparison, we assume that all requests are for bandwidth at least STS-1 (Recall from Sec. IV-C that this setting is skewed against our algorithm). Given the network topology and the bandwidth distribution used in this simulation, we found that at most $b+9$ and on average $b+1.08$ units must be retrieved to satisfy a request for b units of expected bandwidth.

Figure 13 compares MINCOSTADD to the full-protection algorithm for both the highly reliable and the less reliable network settings. Note that since the full-protection algorithm protects against failures regardless of the reliability of the network, it will perform the same under both settings. The results show that even in the less reliable network setting, for all loads, MINCOSTADD is able to provision significantly more bandwidth than the full-protection approach. For example, at a low load (190 Erlangs) MINCOSTADD blocks only about 2% of the bandwidth blocked by the full-protection algorithm. Even at the highest load shown (270 Erlangs), MINCOSTADD blocks about half the bandwidth blocked by the full-protection scheme. The graph also shows that MINCOSTADD's performance decline from the highly reliable setting to the less reliable setting is rather small. This indicates that the algorithm is likely to perform well even for networks which are somewhat unreliable.

VI. Conclusion

Our work presents a new online algorithm, MINCOSTADD, for reliable mutli-path routing. Our algorithm is effective

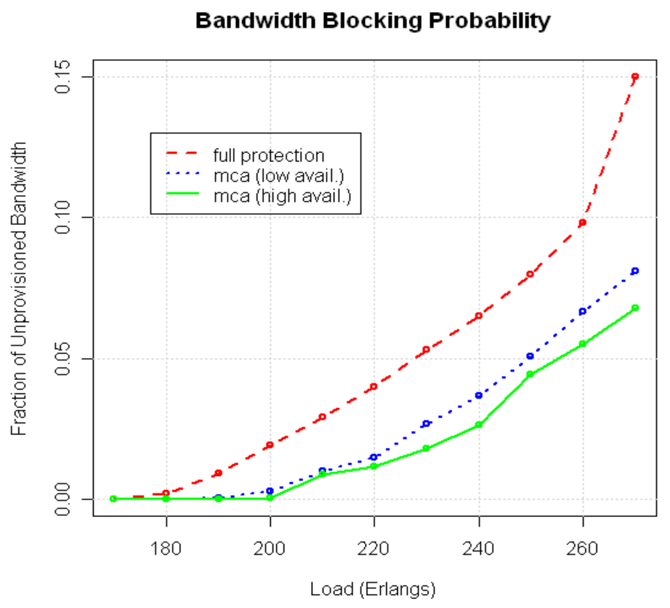

Fig. 13. Fraction of bandwidth blocked: MINCOSTADD vs full-protection algorithm in the less reliable network (uniform setting).

because it takes advantage of multi-path provisioning and consumes as little bandwidth as possible from the network per request. We found that in most realistic networks, paths are reliable enough that only few additional bandwidth units are required to compensate for the bandwidth loss incurred from edge failures. Taking advantage of this feature further improves the performance of our algorithm. The ability of MINCOSTADD to adapt to different network topologies and varying edge contention levels allows it to be very successful, even under a non-uniform setting. For the topology we used (which was a typical US nationwide topology), with symmetric traffic and edge capacities, under a moderately heavy load (300 Erlangs), MINCOSTADD successfully scheduled more than 99.3% of the requests and more than 97% of the requested bandwidth. Results show that the MINCOSTADD algorithm has significant performance improvements over previous approaches in both uniform and non-uniform settings. The improvement in the non-uniform setting indicates that MINCOSTADD would be more effective if used in practice.

We also compared our approach to a fully-protected scheme. Although full-protection algorithms provide more assurance against network failures, this assurance comes at a cost. More specifically, we find that there is a trade-off between full protection and performance. Although our algorithm does not provide full protection, it uses fewer network resources, and can therefore satisfy more requests than a full-protection approach. For settings in which full protection is not crucial, it may be beneficial to use our approach since it offers significantly higher performance.

REFERENCES

[1] M. Clouqueur and W. Grover. Availability Analysis of Span-Restorable Mesh Networks. *IEEE J. on Selected Areas in Communications*, vol. 20, no. 4, pp. 810-821, May 2002.

[2] A. Chakrabarti and G. Manimaran. Reliability Constrained Routing in QoS Networks. *IEEE/ACM Transactions on Networking*, vol. 13, no. 3, pp. 662-675, June 2004.

[3] A. Fumagalli, I. Cerutti, and M. Tacca. Optimal Design of Survivable Mesh Networks Based on Line Switched WDM Self-Healing Rings. *IEEE/ACM Transactions on Networking*, vol. 11, no. 3, pp. 501-512, June 2003.

[4] L. Guo, H. Yu, and L. Li. A New Path Protection Algorithm for Meshed Survivable Wavelength-Division-Multiplexing Networks. *Networking ICN 2005*, vol. 3420, pp. 68-75.

[5] S. Ramamurthy, L. Sahasrabuddhe, and B. Mukherjee, Survivable WDM Mesh Networks, *IEEE/OSA Journal of Lightwave Technology*, vol. 21, no. 4, pp. 870-883. April 2003.

[6] Y. Xiong, D. Xu, and C. Qiao. Achieving Fast and Bandwidth-Efficient Shared-Path Protection. *Journal of Lightwave Technology*, vol. 21, no. 2, pp. 2683-2693, Feb. 2003.

[7] W. Yao and B. Ramamurthy. Survivable Traffic Grooming with Path Protection at the Connection Level in WDM Mesh Networks. *Journal of Lightwave Technology*, vol. 23, no. 10, pp. 2846-2853. Oct. 2005.

[8] B. Doshi, S. Dravida, P. Harshavardhana, O. Hauser, and Y. Wang, Optical Network Design and Restoration. *Bell Labs Technical Journal*, vol. 4, pp. 58-84, Mar. 1999.

[9] D. Gao and H. Zhang. Routing Pre-Configuration for Fast and Scalable Path Restoration in DWDM Networks. *Photonic Network Communications*, vol. 12, no. 3, pp. 321-327, Dec. 2006.

[10] R. Iraschko and W. Grover. A Highly Efficient Path-Restoration Protocol for Management of Optical Network Transport Integrity. *IEEE Journal on Selected Areas in Communications*, vol. 18, no. 5, pp. 779-794, May 2000.

[11] S. Norden, M. Buddhikot, M. Waldvogel, and S. Suri. Routing Bandwidth-Guaranteed Paths with Restoration in Label-Switched Networks. *Proceedings of Computer Networks*, vol. 46, no. 2, pp. 197-218, 2004.

[12] J. Zhang, K. Zhu, H. Zang, N. Matloff, and B. Mukherjee. Availability-Aware Provisioning Strategies for Differentiated Protection Services in Wavelength-Convertible WDM Mesh Networks. *IEEE/ACM Transactions on Networking*, vol. 15, no. 5, pp. 1177-1190, Oct. 2007.

[13] ITU-T Recommendation. Network Node Interface for the Synchronous Digital Hierarchy (SDH). *ITU-T Recommendation G.707*, December 2003.

[14] S. Rai, O. Deshpande, C. Ou, C. Martel, and B. Mukherjee. Reliable Multi-Path Provisioning for High-Capacity Backbone Mesh Network. *IEEE/ACM Transactions on Networking*, vol. 15, no. 4, pp. 803-812, Aug. 2007.

[15] A. Das, C. Martel, B. Mukherjee. A Partial-Protection Approach Using Multipath Provisioning. *IEEE International Conference on Communications.*, June 2009.

[16] C. Ou, L. Sahasabuddle, K. Zhu, C. Martel, and B. Mukherjee. Surviable Virtual Concatenation for Data Over SONET/SDH in Optical Transport Networks. *IEEE/ACM Transactions on Networking*, vol. 14, pp. 218-231, Feb. 2006.

[17] W. Grover. Mesh-Based Survivable Networks: Options and Strategies for Optical, MPLS, SONET and ATM Networking. Prentice Hall PTR, Upper Saddle River, New Jersey, 2003.

[18] K. Zhu, H. Zang, and B. Mukherjee. A Comprehensive Study on Next-generation Optical Grooming Switches. *IEEE Journal on Selected Areas in Communications.*, vol. 21, no. 7, pp. 1173-1186, Sep. 2003.

[19] R. Ahuja, T. Magnanti, and J. Orlin. Network Flows: Theory, Algorithms, and Applications. Prentice Hall, Englewood Cliffs, New Jersey, 1993.

[20] http://www.darpa.mil/sto/solicitations/CORONET/pip.htm

[21] F. Farahmand, X. Huang, and J. Jue. Efficient Online Traffic Grooming Algorithms in WDM Mesh Networks with Drop-and-Continue Node Architecture. *Proceedings of BroadNets 2004*. Oct. 2004.

[22] J. Hu and B. Leida. Traffic Grooming, Routing and Wavelength Assignment in Optical WDM Mesh Networks. *IEEE INFOCOM*. Mar. 2004.

[23] Alcatel-Lucent. Network Availability in Meshed Transport Networks. Whitepaper. http://www1.alcatel-lucent.com/com/en/appcontent/opgss/Net_Avail_Meshed_twp_tcm228-1283621635.pdf