A Better 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). VCAT enables a connection to be inversely multiplexed on to multiple paths, a feature that has many advantages over conventional single-path provisioning. We propose improved routing algorithms which use minimum-cost flow to find efficient collections of paths that satisfy the traffic requests.**

We first investigate the performance of our scheme under a uniform setting with symmetric traffic distribution and equal link capacities. We then apply our algorithm in a more realistic setting with asymmetric traffic and differing link 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. Our study in the non-uniform setting is significant as it gives insight into the performance of our algorithm under more realistic scenarios.

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 [5]. Various routing schemes have been proposed that maintain reliability by handling failures. One approach is path *protection* [3], [11], in which backup routing paths are reserved during connection setup to cope with failures in the primary paths. The drawback to protection schemes is that they require additional network resources and therefore an extra cost for the network operator. Another approach is *restoration* [9], [11], in which backup paths are discovered dynamically *after* a link failure. However, since recovery must be performed after the failure has occurred, restoration schemes take more time than protection schemes.

Previous researchers have considered satisfying customer availability requirements using *single* path routing schemes [14]. Next-generation networks, such as NG-SONET/SDH, supporting virtual concatenation (VCAT) [6] allow connections to be provisioned on multiple paths. Multi-path routing has the obvious advantage of better fault tolerance. It provides more effective utilization of network resources, and relieves link congestion and delay. An offline multi-path routing scheme has been proposed to handle network traffic in mesh networks [2]. Although the approach is called “dynamic”, all routing computation is done offline, and a database of routing information is indexed at call set-up time. However, since this offline scheme fails to fully adapt to changing network conditions, it may not be very practical.

This work has been supported by NSF Grant No. CNS-05-20190.

Since many decisions for network management must be made in real time, efficient online schemes are essential. In the online reliable QoS provisioning problem that we study, a series of bandwidth requests are issued dynamically and each request must be scheduled as it arrives. Once a request has been scheduled, it cannot be rerouted. If a request cannot be satisfied, it is rejected. The authors of [10] studied this problem for high-capacity backbone mesh networks with possible link failures. They proposed a new metric, *effective bandwidth*, to measure the expected amount of available bandwidth provisioned for a connection over multiple paths.

We propose a new multi-path heuristic that solves this problem and significantly improves on the results from [10]. While the prior heuristic focuses on finding sets of *separate* good paths, our approach seeks paths that *jointly* make up the best set. Our algorithm uses a minimum-cost flow in the network to find an efficient collection of paths that meets the 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 limiting the overuse of any particular link. This technique also helps to reduce link congestion.

We first investigate the performance of our schemes under a uniform setting with symmetric traffic distribution and equal link capacities. We then extend our study to a non-uniform setting with asymmetric traffic and differing link capacities. The latter results give us a better understanding of how the algorithms would perform in practice. Non-uniformity has been studied for computational grids [7], communication networks [8], [13], and storage networks [12]. However, to our knowledge, ours is the first paper to study non-uniformity for availability-aware multi-path routing.

Our simulation results show that by finding better sets of paths that preserve network capacity, our algorithm is highly successful. For a typical US nationwide topology with realistic and asymmetric traffic distribution, under a heavy load of 400 Erlangs, our algorithm can schedule 99.9% of the requests and 99.7% of the requested bandwidth. Our algorithm also performs significantly better than prior schemes in both uniform and non-uniform settings. However, the difference in improvement in the non-uniform setting is substantial, indicating that our approach would be much more effective for practical networks.

The remainder of this paper is organized as follows: Section II describes the QoS problem we are investigating. Section III presents our multi-path routing algorithms. Section IV provides our illustrative results. Finally, Section V presents our conclusions.

II. The Maximum Bandwidth Problem

The reliable multi-path provisioning problem is modeled in [10] as the MAXBAND problem. The goal is to establish a connection between two nodes s and d, and send b units[1] of *effective* bandwidth from s to d. For a path P, with k edges[2], $e_1, e_2, \ldots, e_k$, if the respective availabilities of these edges are $a_1, a_2, \ldots, a_k$, then the availability of the path is $A = a_1 \cdot a_2 \cdot \ldots \cdot a_k$. Assuming that all links fail independently, A is the probability that P is properly functioning. If the respective capacities on the edges of P are $c_1, c_2, \ldots, c_k$, then the effective bandwidth of P is $A{\cdot}c_{min}$, where $c_{min} = min_{1 \le i \le k}(c_i)$. 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 effective bandwidth of π is $\sum_{i=1}^{m} A_i{\cdot}b_i$.

The MAXBAND 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 effective bandwidth requirement. The MAXBAND problem is to find a set of paths from s to d such that the effective bandwidth from s to d is $\ge b$, while maintaining $c_i \ge 0$ for all $e \in E$.

Since even the off-line version of MAXBAND is **NP**-hard, efficient heuristics are needed to solve this problem. The MAXFLOW heuristic was developed and tested in [10]. The algorithm 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 effective bandwidth or until no more paths can be found. MAXFLOW outputs a set of paths that satisfies the effective bandwidth request if it finds such a set; otherwise the request is rejected and no paths are assigned. The experimental network used was a US nationwide network topology which resembles a well-connected carrier's backbone topology [15] (see Fig. 1). The links were bidirectional and link availabilities were uniformly distributed over the values [0.9999, 0.99999, 0.999999]. Since link availabilities are less than 1, all path availabilities will also be less than 1. Therefore, to satisfy an effective bandwidth request of b units, it is always necessary to consume at least $b + 1$ units (we will show in Section II-A that in this setting, exactly $b + 1$ units will be sufficient).

Figure 2 shows an example of MAXFLOW's 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.

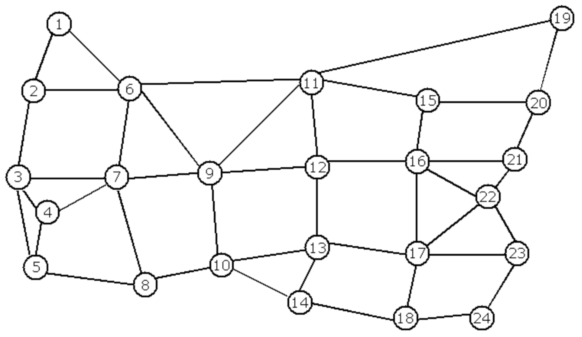

Fig. 1. Sample network topology.

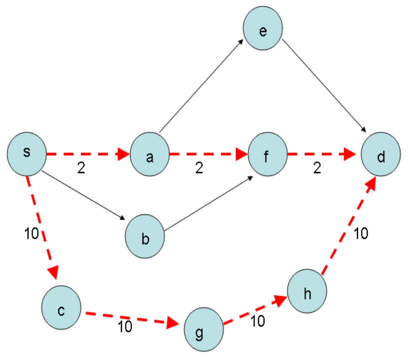

Fig. 2. Example of MAXFLOW scheme.

For this example, suppose the request (<*s, d, 11*>), has been issued[3]. MAXFLOW 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*. Since the availability of these paths is less than 1, a request for 11 units of effective bandwidth would require at least 12 units of total bandwidth. Therefore, MAXFLOW would send 2 units of flow along *s-a-f-d*. Clearly, this is not the ideal set of paths, since the request can be satisfied without using the longest path.

We found two drawbacks with the MAXFLOW 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 be used over a longer path with higher availability. Second, the MAXFLOW 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 *multiple* paths.

We propose a new heuristic to address the MAXBAND 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 Figure 3). It also exploits the use of multi-paths by finding

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

[2]In this paper, we use the terms *edge* and *link* interchangeably.

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

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

the set of *globally* optimum paths that satisfies the current bandwidth request.

A. *The Effect of Fractional Path Availabilities*

While the MAXFLOW algorithm seeks the highest availability path, our algorithm initially ignores the path availabilities and aims to find the shortest paths that can satisfy the requests. Given the network settings used in [10] (which are similar to realistic observed settings), this approach of not prioritizing availabilities maintains practicality. In our experimental runs using the same topology, we found that the average path length was 3. A path of length 3, with each link having the lowest possible availability, would have an overall availability of 0.9999×0.9999×0.9999= 0.9997, which is still close to 1. For the settings used in [10], we find that $b+1$ units is always enough to satisfy a request for b units of effective bandwidth[5]. For example, if 96 units of bandwidth are requested, and a path with availability 0.9997 is found, then by retrieving 97 units of bandwidth from this path we will obtain an effective bandwidth of 97×0.9997 = 96.9709, which is more than enough to satisfy the request. The high link availabilities and the short path lengths that are characteristic of the network topology generally allow for paths with high availabilities. MAXFLOW's approach of finding highly reliable paths to deal with bandwidth loss incurred from fractional availabilities is rather unnecessary and can be avoided by simply retrieving an additional unit of bandwidth.

III. The MINCOST Algorithm

Given a connection request, the MINCOST algorithm finds a 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 G 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 the 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 [1].

Figure 3 shows MINCOST's approach for the previous example. Assume the same link 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 MAXFLOW and MINCOST are able to satisfy the request, MINCOST does so by consuming 10×3+2×3=36 units of bandwidth whereas MAXFLOW 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, MAXFLOW would reject this request.

[5]The maximum amount of bandwidth requested is 96 units. Therefore, as long as the path length is less than $k = log_{(0.9999)}\frac{96}{97} = 103$, $b+1$ units will be sufficient to satisfy a request for b units.

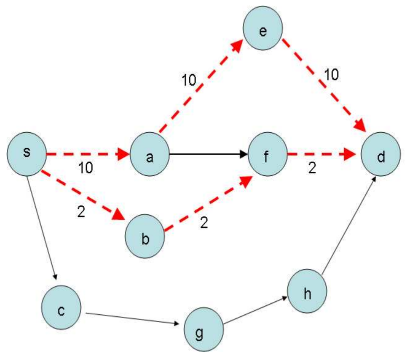

Fig. 3. Example of MINCOST scheme.

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

```
1: Assign each edge $e \in E$ a cost $w(e) = 1$.
2: Find the 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 link in $\pi$ by its new flow.
       Provisioning Successful.
5: else
6:     Reject this request.
7: end if
```

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

Our algorithm requires a bit more computation than MAXFLOW. However, 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.

A. *The* MINCOSTADD *Algorithm*

Congestion is a common problem that arises when a series of network requests is issued. Edges which lie on the shortest paths for many node pairs are likely to be accessed frequently. To avoid congestion, the use of these edges should be limited. For example, Fig. 4 shows a subgraph of the topology used in this study. In this example, edge (6, 11) lies on the shortest path between several node pairs such as: 1 and 12, 1 and 19, and 1 and 20. 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).

Congestion also occurs when certain nodes are requested more frequently than others. The bandwidth on the edges adjacent to these nodes diminishes faster than on other edges.

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

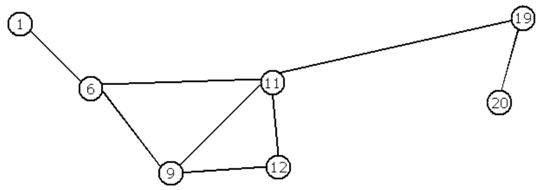

Fig. 4. Subgraph of sample topology.

Clearly, we can imagine realistic situations in which this would occur: in a network of U.S. cities, a link connecting two major cities, such as New York and Boston, is likely to be accessed more frequently than a link connecting two small cities. Suppose there are multiple paths that connect two small cities A and B, but one of these paths contains an edge e that connects New York to Boston. If a request from A to B is issued, we should avoid using edge e to satisfy this request.

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 be accessed less frequently. They will be used mainly for situations in which they are crucial for efficiently satisfying a request. This modification produces improved performance results 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. For example, if we are given the topology ahead of time, and we know that a particular edge lies on the shortest path for many node pairs, we can limit the use of this edge by increasing its cost. Similarly, if the network setting is not uniform and we are aware of the popular edges, we can relieve congestion by regulating the usage of these edges. This feature of the MINCOSTADD algorithm makes it both flexible and adaptive. It promotes more efficient use of network resources by allowing us to preserve edges which are more in demand.

IV. Illustrative Numerical Examples

A. Uniform Setting

To evaluate the performance of our algorithms, we replicated the simulated dynamic network environment used in [10]. The connection arrival process is Poisson and the connection-holding time follows a negative exponential distribution with unit mean. There are 16 wavelengths per link, 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 100Mb of bandwidth, 21% are for 150Mb, 10% are for 600Mb, 10% are for 1Gb, 4% are for 2.5Gb, 2% are for 5Gb, and the final 1% of requests are for 10Gb of bandwidth. This distribution follows typical bandwidth distributions observed in realistic networks. In the first set of simulations, we assume a uniform traffic distribution over all node pairs. The availability of links 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.

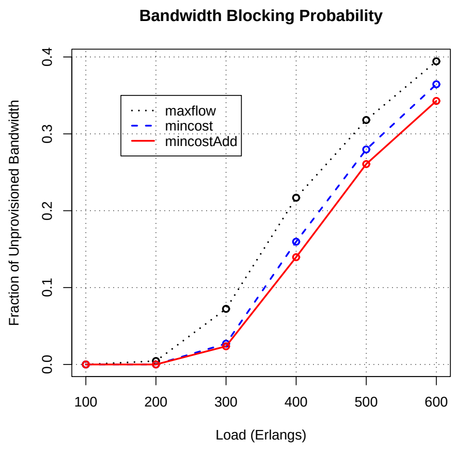

Fig. 5. Fraction of bandwidth blocked in uniform setting.

We applied the MINCOST and MINCOSTADD algorithms and compared their performance to the MAXFLOW algorithm. We observed the fraction of unprovisioned bandwidth (bandwidth blocking probability), the fraction of unprovisioned requests (probability of failure), and the number of satisfied requests before the first failure occurs.

For all load levels, MAXFLOW is outperformed by both of our algorithms. Our algorithms consistently provision more bandwidth, and can satisfy a higher number of requests before the first failure, and a higher number of requests in total. For moderate load (300 Erlangs), MINCOST and MINCOSTADD block less than a third of the bandwidth blocked by MAXFLOW (see Fig. 5). Both of our algorithms are also 3 times less likely to fail than MAXFLOW (see Fig 6). Under the same load, MINCOST satisfies more than twice the number of requests that MAXFLOW satisfies before the first failure, and MINCOSTADD satisfies more than 3 times this amount (see Table I). Under a load of 200 Erlangs, our algorithms are always successful, whereas MAXFLOW has a few failures (approximately 140).

The results illustrate the effectiveness of our algorithms. Even under a moderate load level (300 Erlangs), our algorithms satisfy more than 99.3% of the requests and more than 97% of the requested bandwidth.

To verify that the MINCOST algorithm retains more bandwidth in the network than MAXFLOW, we recorded the amount of bandwidth consumed by each algorithm to satisfy 100,000 requests under a light load (100 Erlangs)[8]. We found that MINCOST consumes approximately 30 units per request whereas MAXFLOW consumes approximately 30.5 units, which is about 1.67% higher.

[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.

[8]Since under this load, neither algorithm failed for 100,000 requests, this setting made it easy to compare the amount of bandwidth consumed.

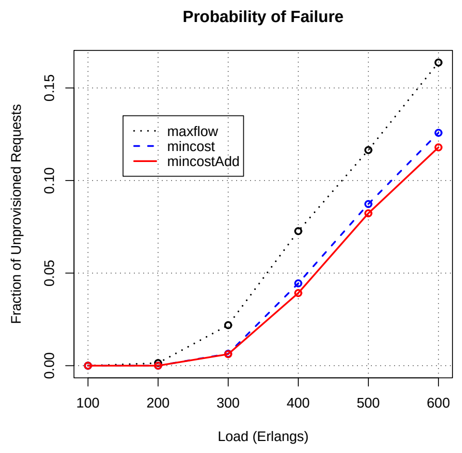

Fig. 6. Fraction of requests blocked in uniform setting.

Load	maxflow	mincost	mincostadd
100	NF	NF	NF
200	7692	NF	NF
300	4978	14120	17390
400	3919	8698	8953
500	3406	7449	7770
600	3600	6713	6957

TABLE I

NUMBER OF SUCCESSFUL REQUESTS BEFORE FIRST FAILURE IN UNIFORM SETTING (**NF** INDICATES NO FAILURES).

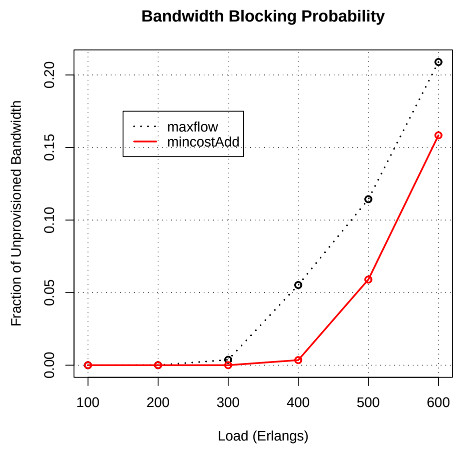

Fig. 7. Fraction of bandwidth blocked in non-uniform setting.

Load	maxflow	mincostadd
100	NF	NF
200	84186	NF
300	13667	NF
400	6467	29556
500	5569	16420
600	5923	11614

TABLE II

NUMBER OF SUCCESSFUL REQUESTS BEFORE FIRST FAILURE IN NON-UNIFORM SETTING (**NF** INDICATES NO FAILURES).

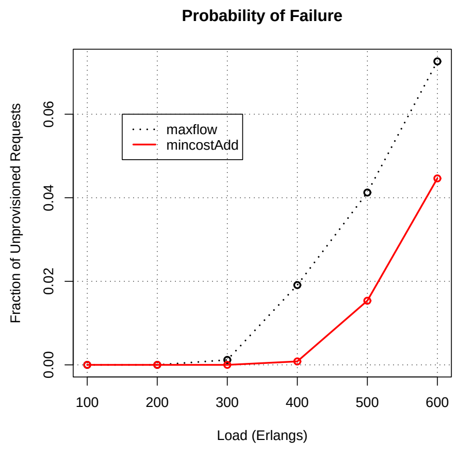

Fig. 8. Fraction of requests blocked in non-uniform setting.

Load	Uniform	Non-Uniform
200	200	200
300	101.48	200
400	43.40	176.17
500	19.82	63.87
600	14.00	27.50

TABLE III

MINCOSTADD'S PERFORMANCE GAIN OVER MAXFLOW - MEASURED BY BANDWIDTH BLOCKING PROBABILITY (VALUES ARE PERCENTAGES).

B. 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 account for this asymmetry, network operators are likely to supply links adjacent to the popular sites with more bandwidth. To understand the relative performance of MAXFLOW 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) [16] model and set 20% of the nodes to be "large". We chose the 20% of nodes with the highest degree for this set (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.

Links adjacent to large nodes were assigned twice as much bandwidth (32 wavelengths, each at ≈10 Gbps) as other links. All other settings in the non-uniform experiments were kept the same as for the uniform case.

Load	Uniform	Non-Uniform
200	200	200
300	111.40	200
400	59.77	183.40
500	34.38	91.51
600	32.54	47.78

TABLE IV

MINCOSTADD'S PERFORMANCE GAIN OVER MAXFLOW - MEASURED BY PROBABILITY OF FAILURE (VALUES ARE PERCENTAGES).

Figures 7 and 8 and Table II show our results under the non-uniform setting. For simplicity, we only compare MAXFLOW to MINCOSTADD since the latter consistently performs better than MINCOST (however, both of our algorithms outperform MAXFLOW). The performance of MINCOSTADD is even more impressive in this non-uniform setting. When the network is considerably loaded (at 400 Erlangs), the algorithm successfully schedules 99.9% of the requests and 99.7% 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 in the non-uniform setting, for all loads, MINCOSTADD performs better than MAXFLOW. We calculated the percent difference[9] in bandwidth blocking probability and probability of failure for the two algorithms under both settings. These values are provided in Tables III and IV. These values show that the difference in the performance gain achieved by MINCOSTADD over MAXFLOW is considerably higher in the non-uniform setting than in the uniform setting. In particular, when the load is at 400 or 500 Erlangs, the performance improvement, in terms of bandwidth blocking probability, of MINCOSTADD over MAXFLOW in the non-uniform setting is more than three times the performance improvement in the uniform setting. In terms of probability of failure, the performance improvement in the non-uniform setting is more than twice the performance improvement in the uniform setting. This difference in improvement between the two settings is crucial. It implies that 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.

V. CONCLUSION

Our work presents a new online algorithm MINCOST for reliable mutli-path routing and an improved version of this algorithm, MINCOSTADD. Our algorithms are effective because they take advantage of the multi-path feature and maintain as much bandwidth as possible in the network per request. The ability of MINCOSTADD to adapt to different network topologies and varying edge demands allows it to be very successful especially under a non-uniform setting. For the topology we used (which was a typical US nationwide topology), with asymmetric traffic and link capacities, even under a relatively heavy load (400 Erlangs), MINCOSTADD successfully scheduled 99.9% of the requests and 99.7% of the requested bandwidth. Results show that our MINCOSTADD algorithm has significant performance improvements over MAXFLOW in both uniform and non-uniform settings. This difference is especially notable in the non-uniform setting which indicates that MINCOSTADD would be much more effective if used in practice.

[9]The percent difference of two values a and b is $\frac{|a-b|}{average(a,b)} \times 100$.

REFERENCES

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

[2] S. Bahk and M. Zarki. Dynamic Multi-path Routing and How it Compares with other Dynamic Routing Algorithms for High Speed Wide Area Networks. *ACM Computer Communications Review*, vol. 22, no. 4, pp. 53-64, Oct. 1992.

[3] 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.

[4] H. Cheng and J. Chen. Performance of Fast Bandwidth Reservation with Multipath Routing. *IEE Proceedings - Communications*, vol. 145, no. 2, pp. 80-86, April 1998.

[5] 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.

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

[7] V. Iyengar, S. Tilak, N. Abu-Ghazaleh, and M. J. Lewis. Nonuniform Information Dissemination for Dynamic Grid Resource discovery. *Proceedings of IEEE NCA*, pp. 97-106, 2004.

[8] E. Noel and K. Tang. Performance Modeling of Multihop Network Subject to Uniform and Nonuniform Geometric Traffic. *IEEE/ACM Transactions on Networking*, vol. 8, no. 6, pp. 763-774 Dec. 2000.

[9] 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.

[10] 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*. To appear, Dec. 2007.

[11] 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.

[12] S. Sobti, N. Garg, F. Zheng, J. Lai, Y. Shao, C. Zhang, and E. Ziskind, A. Krishnamurthy and R. Wang. Segank: A Distributed Mobile Storage System. *Proc. 3rd Conference on File and Storage Technologies (FAST)*, Mar. 2004.

[13] S. Wong, J. Lim, S. Rao, and W. Seah. Density-aware Hop-count Localization (DHL) in Wireless Sensor Networks with Variable Density. *IEEE Wireless Communications and Networking Conference*, pp. 13-17, March 2005.

[14] J. Zhang, K. Zhu, H. Zang, and B. Mukherjee. A New Provisioning Framework to Provide Availability-guaranteed Service in WDM Mesh Networks. *Proc. IEEE International Conference on Communications (ICC '03)*, May 2003, pp. 1484-1488.

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

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