A Randomized Algorithm for Finding a Path Subject to Multiple QoS Constraints

Turgay Korkmaz and Marwan Krunz
Department of Electrical & Computer Engineering
University of Arizona
Tucson, AZ 85721
{turgay,krunz}@ece.arizona.edu

December 24, 1998

Abstract

One of the key issues in providing end-to-end quality-of-service (QoS) guarantees in communication networks is how to determine a *feasible* route that satisfies a set of QoS requirements (e.g., bandwidth, delay, delay-jitter, and loss) of a connection request while efficiently using network resources. In general, finding a path subject to multiple *additive* constraints such as delay and delay-jitter is an NP-complete problem. In this paper, we propose a polynomial-time heuristic algorithm for this problem. We also discuss how to accommodate non-additive constraints such as bandwidth and loss. Our algorithm first prunes all the links that cannot be on any feasible path from the source to the destination. It then uses a randomized search to find a feasible path, if one exists. In order to achieve efficient utilization of resources, our algorithm tries to select a path with minimum-hop count among all feasible paths. The worst-case computational complexity of our algorithm is $\mathcal{O}(n^3)$, where n is the number of nodes. Its storage complexity is $\mathcal{O}(n)$. Using extensive simulations, we show that our algorithm gives very high success rate in finding feasible paths with minimum-hop count.

1 Introduction

Integrated communication networks (e.g., ATM) offer end-to-end quality-of-service (QoS) guarantees to various applications such as audio, video, and data. Some of these applications have multiple stringent QoS requirements in terms of bandwidth, delay, delay-jitter, and loss. One of the most important problems in QoS-based service offerings is how to determine a route that satisfies multiple QoS requirements of a connection request while simultaneously achieving efficient utilization of network resources. This problem is known as QoS-based routing and is being investigated in the research community. In [2, 4] the authors survey various aspects of QoS architectures and QoS-based routing, including the maintenance of state information, routing strategies, and path selection algorithms. In this paper, we study QoS-based routing and focus on path selection with multiple QoS constraints.

The problem of path selection subject to multiple additive constraints is known to be NP-complete [18]. In other words, there is no efficient (polynomial-time) algorithm that can *surely* find a feasible path which simultaneously satisfies such constraints. To cope with the NP-completeness of the problem, researchers have either proposed heuristic algorithms which reduce the computational time but do not guarantee to find a feasible path even one exists [8, 16, 10], or simplified the problem by considering a subset of the QoS parameters such as bandwidth and delay [17, 12, 7]. In [13, 15] the authors show that path selection subject to multiple QoS constraints can be solved in polynomial-time if Weight Fair Queueing (WFQ) service discipline is used. In this service discipline, delay, delay-jitter, and loss constraints can be defined as functions of bandwidth; thus, the problem is reduced to the shortest path routing without any additive QoS constraint. Since these algorithms either consider only bandwidth and delay or make other constraints depend on bandwidth by assuming specific service disciplines, they cannot be used to find a path subject to multiple constraints that are additive and independent. In [9], the author presents a pseudo-polynomial-time algorithm for path selection with two independent constraints. This is one of the closest studies to ours. The computational complexity of this algorithm is $\mathcal{O}(n^5 b \log nb)$, where n is the number of nodes and b is the maximum weight of a link. Its storage complexity is $\mathcal{O}(n^3 b \log nb)$. If b is large, this algorithm is very expensive in terms of computational time and storage requirement. The author also proposes a heuristic algorithm that approximates the constraints and runs with a polynomial-time complexity. However, the algorithm is not guaranteed to find a correct solution. Another related work is presented in [3], where the authors propose a heuristic algorithm that truncates all link weights except one to bounded integer values. They show that the modified problem can be solved with a polynomial-time complexity by using extended Dijkstra's (or Bellman-Ford) algorithm and that the solution for the reduced problem is also a solution for the original problem. However, the algorithm is not guaranteed to find all the feasible paths in the original problem. When the extended Dijkstra's algorithm is used, the computational complexity of the proposed algorithm in [3] is $\mathcal{O}(x^2n^2)$; when the extended Bellman-Ford algorithm is used, the complexity is $\mathcal{O}(xnm)$, where x is a constant defined by the algorithm, n is the number of nodes, and m is the number of links. The storage complexity of this algorithm is $\mathcal{O}(xn)$. The value of x determines the performance and the overhead of the algorithm. To increase the probability of finding a feasible path, the value of x can be as large as $10n$, i.e., the worst-case computational complexity is $\mathcal{O}(n^4)$ with extended Dijkstra's algorithm.

In this paper, we provide a heuristic algorithm to finding a path subject to multiple additive constraints without making any assumptions about the scheduling disciplines in the network or the values of the QoS parameters. Furthermore, our algorithm tries to select a feasible path with minimum-hop count; achieving efficient utilization of resources. It has been shown that restricting routing to short paths achieves efficient resource utilization in QoS-based routing [12]. Our algorithm mainly consists of two steps: (1) pruning all the links that cannot be on any feasible paths between the source and destination nodes, and (2) performing a randomized search to find a feasible path, if one exists. The idea behind randomization is to avoid unforeseen traps by making random choices during the execution of the algorithm [11]. Randomized algorithms are being used very efficiently in many applications because of their simplicity and speed [14]. The worst-case computational complexity of our algorithm is $\mathcal{O}(n^3)$, where n is the number of nodes. This complexity can be further reduced in practice. The storage complexity of our algorithm is $\mathcal{O}(n)$. In terms of both complexities, our algorithm is better than the algorithms in [9] and [3]. On the other hand, since our algorithm is also based on a heuristic, it has some small probability of not finding a feasible path although one does exist. However, our simulation results show that our algorithm achieves high performance in finding feasible paths. We present extensive simulations with two and three additive constraints on different network topologies in which our algorithm finds 99.99% of all feasible paths with the minimum number of hops.

The rest of the paper is organized as follows. In Section 2, we discuss routing with multiple QoS constraints, and we formalize the problem of minimum-hop path selection subject to multiple additive constraints. In Section 3, we introduce our polynomial-time randomized algorithm, and present the pseudo-code, correctness, and complexity analysis of this algorithm. In order to illustrate how the algorithm works, we give an example in Section 4. We report simulation results in Section 5. Finally, Section 6 concludes this paper.

2 Routing Problem with Multiple QoS Constraints

Routing consists of two basic tasks [4]: collecting the state information of the network and searching this information to find a feasible path, if one exists. In this paper, we focus on the second task by assuming that the true state of the network is available to every node and that nodes use source routing to determine an end-to-end feasible path. Each link in the network is associated with multiple QoS parameters such as bandwidth, delay, delay-jitter, and loss. Although we consider only additive constraints (e.g., delay and delay-jitter) in our randomized algorithm, non-additive constraints such as bandwidth can be easily accommodate. This is done by pruning links that do not satisfy the requested QoS constraint. Furthermore, we consider the problem of minimum-hop path selection subject to multiple additive constraints. The underlying problem is formally stated as follows.

Definition 1. *Minimum Hop Routing with Multiple Constraints* (MinHop_MC): Consider a communication network that is represented by a directed graph $G = (N, A)$, where N is the set of nodes and A is the set of links (arcs). Each link $(i, j) \in A$ is associated with M non-negative weights (additive QoS values) $w_k(i, j)$ for $k = 1, 2, \ldots, M$. Given M constraints C_k, $k = 1, 2, \ldots, M$, the problem is to find a path p from a source node s to a destination node t such that the number of nodes in p is minimum, i.e.,

$$\text{Minimize} \sum_{(i,j)\in p} 1$$

while satisfying the following additive M constraints:

$$\sum_{(i,j)\in p} w_k(i, j) \leq C_k \quad \text{for } k = 1, 2, \ldots, M$$

3 A Randomized Algorithm for MinHop_MC

In order to minimize the hop count, we first transform the problem of MinHop_MC into a new one by replacing the minimum-hop objective with a constraint. The new problem is to find a path p that satisfies the following constraints:

$$\begin{aligned} \sum_{(i,j)\in p} 1 &\leq H \\ \sum_{(i,j)\in p} w_k(i,j) &\leq C_k \text{ for } k = 1, 2, \ldots, M \end{aligned}$$

where H is a bound for the hop count of path p. The value of H is an integer number between 0 and $n-1$, where n is the number of nodes in the graph. Suppose that we have an algorithm which can find a feasible path satisfying the above constraints. If we execute this algorithm for each value of H starting from 0 and increasing by one up to $n-1$ and stop whenever a path is found, then we have solved the problem of MinHop_MC.

The remaining important question is whether there is an efficient (polynomial-time) algorithm to finding a path subject to the above constraints. Since the original problem is NP-complete, there is no such an algorithm unless NP=P. Alternatively, we provide a heuristic algorithm which prunes the graph and uses a randomized search method to find a feasible path with minimum-hop count. Our algorithm consists of two main procedures:

1. Minimum Hop Routing with Multiple Constraints (MinHop_MC)
   To maximize the probability of success in the random search for a feasible path, this procedure associates some labels with each node and prunes all links that cannot be on any feasible paths with respect to given M constraints. Then, for each value of hop bound, the procedure continue to reduce and label the graph based on the current hop bound. If the procedure realizes that there might be a feasible path, it calls the below procedure to search the path on the reduced and labeled graph.

2. Modified Random Breadth-First Search (MR_BFS)
   MR_BFS is a randomized and modified version of Breadth-First Search (BFS). Note that BFS systematically discovers every node that is reachable from a source node s. In contrast, MR_BFS randomly discovers those nodes from which there is a good chance to go to the final destination node t. By using the reduced graph and labels, MR_BFS can realize whether this chance exists or not before discovering a node. If there is no chance, MR_BFS can foresee the trap and randomly tries other nodes. The heuristic behind MR_BFS is to discover a node from which there is a chance to go to the destination node t and hope that this discovery will lead to a feasible path from the source node s to t. Simulation results show that this method works well for path selection with multiple additive constraints.

The inputs to the algorithm are a directed graph $G = (N, A)$ in which each link (i, j) is associated with M weights: $w_k(i,j)$, $k = 1, 2, \ldots, M$; a source node s; a destination node t; and M constraints: C_k, $k = 1, 2, \ldots, M$. In order to find a feasible path with minimum-hop count, the algorithm maintains the following labels for each node u: $B_k[u]$, $\widetilde{B}_k[u]$, $R_k[u]$, $D_k[u]$, and $\pi[u]$, $k = 1, 2, \ldots, M, h$. Labels $B_k[u]$, $k = 1, 2, \ldots, M, h$, represent the costs of the shortest paths from the source node s to node u with respect to individual M link weights and hop count, respectively. Labels $\widetilde{B}_k[u]$, $k = 1, 2, \ldots, M, h$, represent the costs of the shortest paths from node u to the destination node t with respect to individual M link weights and hop count, respectively. Labels $R_k[u]$, $k = 1, 2, \ldots, M, h$, represent the same kind of information as $\widetilde{B}_k[u]$, $k = 1, 2, \ldots, M, h$, in the reduced graph. The cost of a path is the sum of the link weights along that path. Note that all link weights are equal to one for the hop-count parameter. In the random search for a feasible path with MR_BFS, the algorithm stores the predecessor of node u in $\pi[u]$. If node u has no predecessor, i.e., it has not been discovered, then $\pi[u] = NIL$. The costs of a path from the source node s to node u is stored in $D_k[u]$, $k = 1, 2, \ldots, M, h$, for M link weights and hop count.

For each $k = 1, 2, \ldots, M$, our algorithm first does the followings. Using Dijkstra's shortest path algorithm [6], our algorithm determines $B_k[u]$ for each node u with respect to only the link weight w_k. At that point, if the cost of the individual shortest path from the source node s to the destination node t is larger than the given constraints, i.e., if

$$B_k[t] > C_k$$

is true, then there is no feasible path, so the algorithm restores all the pruned links and terminates. Otherwise, the algorithm again individually determines $\widetilde{B}_k[u]$ for each node u by using Reverse-Dijkstra [1]. The algorithm then reduces the graph by pruning some links that cannot be on any feasible paths from the source node s to the destination node t. Consider an arbitrary link $(i,j) \in A$. If

$$B_k[i] + w_k(i,j) + \widetilde{B}_k[j] \quad > \quad C_k$$

is true, then the link (i,j) cannot be on any feasible path; and it is pruned because when link (i,j) is used, the costs of all paths from the source node s to the destination node t exceed the constraint C_k, even if the best path from the source node s to node i and the best path from node j to the destination node t are used. The pruned links here are associated with a code of -1.

After this reduction, our algorithm determines $B_h[u]$ and $\widetilde{B}_h[u]$ for each node u by using Dijkstra and Reverse-Dijkstra [1], respectively. Then, it starts the hop bound H from $B_h[t]$ to $n-1$ for finding a feasible path with minimum-hop count. Note that $B_h[t]$ is the minimum number of hops from the source node s to the destination node t regardless of any other constraints.

For each value of H, the algorithm continues to reduce and label the graph according to current hop bound H. Consider an arbitrary link $(i,j) \in A$. If

$$B_h[i] + 1 + \widetilde{B}_h[j] \quad > \quad H$$

is true, then the link (i,j) is pruned because of the same reason above. This time, the pruned links are associated with the code of -2. Then, for each $k = 1, 2, \ldots, M$, the algorithm does the followings. Using Reverse_Dijkstra, it finds $R_k[u]$ for each node u in the reduced graph. At that point, if

$$R_k[s] > C_k$$

is true, then there is no path which satisfies the constraint C_k; thus, the algorithm restores only the pruned links with the code of -2, increases hop bound H by one, and repeats above operations in this paragraph as long as H is less than the number of nodes. Otherwise, if $R_k[s] \leq C_k$ for all k, then there might be a path; and the algorithm will search it by using MR_BFS. If MR_BFS finds a path, the algorithm stops. Otherwise, the algorithm repeats the above operations by restoring pruned links with the code of -2 and by increasing hop bound H by one.

MR_BFS is modified from BFS (see [6] for original BFS). To manage the just discovered nodes, MR_BFS uses a random queue Q instead of a first-in first-out queue in the original BFS. In the main loop, MR_BFS chooses a random node u from queue Q and tries to discover each node v in the adjacency list of node u. Although BFS systematically discovers every node that is reachable from the source node s, MR_BFS discovers those nodes from which there is a chance to go to the final destination node t, i.e., if

$$\begin{aligned} \pi[v] &= NIL \text{ and} \\ D_k[u] + w_k(u,v) + R_k[v] &\leq C_k \text{ for all } k \text{ and} \\ D_h[u] + 1 + R_h[v] &\leq H \end{aligned}$$

is true, then MR_BFS discovers node v from node u, updates $D_k[v]$, $k = 1, 2, \ldots, M$, $D_h[v]$, and $\pi[v]$, and puts node v into queue Q; otherwise, it does nothing. MR_BFS leaves the search as soon as the destination node t is discovered or queue Q is empty.

The pseudo-code of the randomized algorithm is presented below.

```
procedure MinHop_MC($G = (N, A), s, t, C_k, k = 1, 2, \ldots, M$)
begin
1  For $k = 1$ to $M$ loop
2      Dijkstra($G = (N, A), s, k$) to find $B_k[u]$;
3      if $B_k[t] > C_k$ then
4          ◇ there is no feasible path
5          Restore all the pruned links, and stop the algorithm;
6      end if
7      Reverse_Dijkstra($G = (N, A), t, k$) to find $\widetilde{B}_k[u]$;
8      for each link $(i, j) \in A$ loop
9          if $B_k[i] + w_k(i, j) + \widetilde{B}_k[j] > C_k$ then
10             prunes link $(i, j)$ from $A$ (with code -1);
11         end if
12     end for
13 end for
14 ◇ try to find a feasible path with min-hop count
15 Dijkstra($G = (N, A), s, h$) to find $B_h[u]$;
16 Reverse_Dijkstra($G = (N, A), t, h$) to find $\widetilde{B}_h[u]$;
17 for $H = B_h[t]$ to $n$ loop
18     for each link $(i, j) \in A$ loop
19         if $B_h[i] + 1 + \widetilde{B}_h[j] > H$ then
10             prunes link $(i, j)$ from $A$ (with code -2);
21         end if
22     end for
23     For $k = 1$ to $M$ loop
24         Reverse_Dijkstra($G = (N, A), t, k$) to find $R_k[u]$;
25         if $R_k[s] > C_k$ then
26             ◇ there is no feasible path
27             Restore only the pruned links (with code -2), and go to 17;
28         end if
29     end for
30     Reverse_Dijkstra($G = (N, A), t, h$) to find $R_h[u]$;
31     MR_BFS($G = (N, A), s, t, H$);
32     if $t$ is discovered (i.e., $\pi[t] \neq NIL$) then
33         ◇ a feasible path is found
34         Restore all the pruned links, and stop the algorithm;
35     end if
36     Restore only the pruned links (with code -2), and go to 17;
37 end for
38 ◇ no path is found
39 Restore all the the pruned links, and stop the algorithm;
end MinHop_MC;
```

```
procedure MR_BFS(G = (N, A), s, t, H)
begin
    for each node u ∈ N − {s} loop
        D_k[u] = ∞, k = 1, 2, ..., M, and D_h[u] = ∞;
        π[u] = NIL;
    end loop
    D_k[s] = 0, k = 1, 2, ..., M, and D_h[s] = 0;
    π[s] = −1;
    Q = {s};
    while Q ≠ ∅ and π[t] = NIL loop
        u = random[Q];
        Q = Q − {u};
        for each v ∈ Adj[u] loop
            if π[v] = NIL and
                D_k[u] + w_k(u, v) + R_k[v] ≤ C_k for ∀ k = 1, 2, ..., M, and
                D_h[u] + 1 + R_h[v] ≤ H then
                    D_k[v] = D_k[u] + w_k(u, v) for ∀ k = 1, 2, ..., M;
                    D_h[v] = D_h[u] + 1;
                    π[v] = u;
                    Q = Q ∪ {v};
            end if
        end for
    end while
end MR_BFS;
```

3.1 Correctness of the Algorithm

Since the algorithm is randomized and based on a heuristic, it may not find all feasible paths. However, if it finds a path, this path must satisfy the given constraints. Such a randomized algorithm which always gives correct solutions is known as *Las Vegas Algorithm* [14].

Theorem 1: Let $G = (N, A)$ be a directed graph in which each link (i, j) is associated with M non-negative and additive weights $w_k(i, j)$, where $k = 1, 2, \ldots, M$. Suppose that MinHop_MC is run on G to find a path p from a given source node s to a destination node t such that

$$\sum_{(i,j)\in p} w_k(i, j) \leq C_k \text{ for } k = 1, 2, \ldots, M$$

where C_k are given constants. If MinHop_MC discovers the destination node t (i.e., $\pi[t] \neq NIL$) upon termination, then the constructed path p from the source node s to the destination node t must satisfy the given constraints. **Proof:** see Appendix.

3.2 Complexity Analysis

Let n be the number of nodes and m be the number of links in the graph. The number of link weights, M, is a constant and is much smaller than n. MinHop_MC first runs Dijkstra, Reverse-Dijkstra, and the graph reduction once with each link weight (lines 1–13); the worst-case complexities of both Dijkstra's algorithms are $\mathcal{O}(n^2)$, and the complexity of the graph reduction is $\mathcal{O}(m)$. MinHop_MC runs Dijkstra and Reverse-Dijkstra once more to find the minimum hop counts (lines 15–16). Then, maximum of n times, MinHop_MC repeats the graph reduction (lines 18–22) with $\mathcal{O}(m)$, Reverse-Dijkstra for each link weight (lines 23–29) with $\mathcal{O}(n^2)$, MR_BFS (line 31) with $\mathcal{O}(n+m)$ [6], and the graph restorations with $\mathcal{O}(m)$. Thus, the overall computational complexity of MinHop_MC is $\mathcal{O}(M*(n^2+m)+2*n^2+n*(M*n^2+m+n+m)+m)$, i.e., $\mathcal{O}(n^3)$ since $m \leq n^2$ and M is a small constant. The storage complexity of our algorithm is $\mathcal{O}(n)$ since $4M+1$ labels are maintained for every node. Fortunately, there are some ways to reduce the computational complexity of the algorithm when it is executed in practice. First, source nodes may have a policy not to establish a route that has a number of hops greater than a predefined maximum-hop-length limit H_{max} [15]. In this case, the algorithm repeats lines 17–37 H_{max} times, i.e., the complexity will be $\mathcal{O}(H_{max}n^2)$. In practice, $H_{max} < n$. Second, the graph reduction drastically decreases the number of links m when the given constraints are very tight. Thus, Reverse_Dijkstra at line 24 in MinHop_MC and MR_BFS run on a small size of graph, i.e., the computational complexities of these two procedures are less than $\mathcal{O}(n^2)$ and $\mathcal{O}(n+m)$, respectively. Finally, we can use efficient implementations of Dijkstra's algorithms to further reduce the computational complexity [1].

4 Example

In this section, we illustrate how the randomized algorithm finds a path from a source node s to a destination node t for the network in Figure 1. Each link is bidirectional and has two additive weights (w_1,w_2). Although we assume that links are symmetric in this example, the algorithm has been designed to run on asymmetric links. Suppose we want to find a path from $s=0$ to $t=4$ when $C_1 = 13$ and $C_2 = 12$.

The algorithm first finds $B_k[u]$ and $\widetilde{B}_k[u]$, $k=1,2,h$, for each node u by using Dijkstra and Reverse-Dijkstra. These labels are also shown in Figure 1. Since $B_h[t]=4$, the algorithm sets $H=4$, reduces the graph, and finds $R_k[u]$, $k=1,2,h$, for each node u. After the graph reduction, only the following links will remain in the graph: $(0,1)$, $(0,7)$, $(2,3)$, $(3,4)$, $(5,4)$, and $(6,5)$. Obviously, there is no path from source node $s=0$ to destination node $t=4$ in the reduced graph. Thus, the algorithm increases H by 1 (i.e., $H=5$) and tries again. In that case, the reduced graph and the final $R_k[u]$, $k=1,2,h$, will be as shown in Figure 2. The algorithm calls the procedure of MR_BFS to search for a feasible path from the source node 0 to the destination node 4. In the reduced graph, there are three feasible paths $p_1=(0,7,8,9,3,4)$, $p_2=(0,7,8,9,5,4)$, $p_3=(0,1,8,9,3,4)$, and one infeasible path $p_4=(0,1,8,9,5,4)$ which does not satisfy the constraint C_1. Figure 2 illustrates how

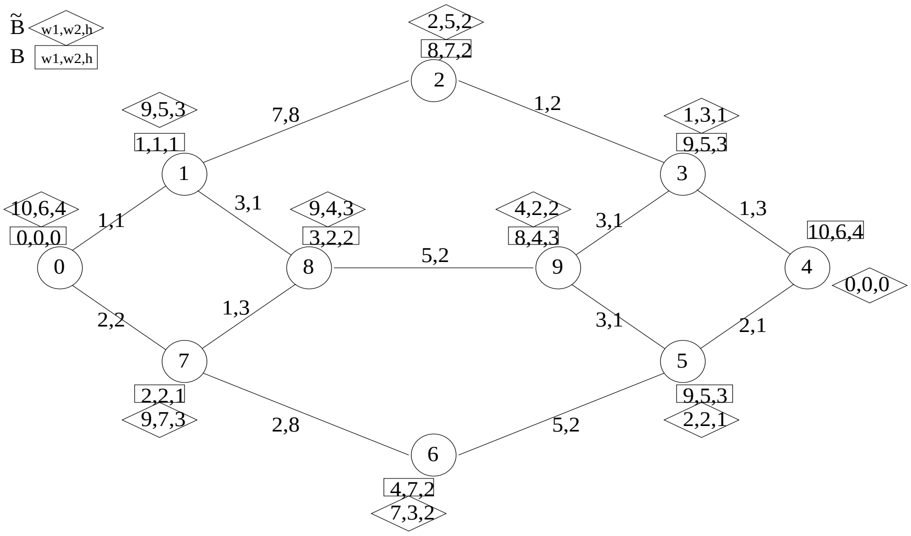

Figure 1: An example of executing the proposed algorithm with two additive parameters. The figure also shows B and $\widetilde{B}$ for each node.

one of the feasible paths is found. Suppose MR_BFS discovers nodes $1, 7$ from node 0, node 8 from node 1, and node 9 from node 8. Thus, MR_BFS constructs a path $p = (0, 1, 8, 9)$ from the source node 0 to node 9. At node 9, MR_BFS foresees that if it discovers node 5 from node 9, there is no chance to go to the destination since $D_1[9] + 3 + R_1[5] = 14$ is greater than $C_1 = 13$; thus, it eliminates p_4 and discovers node 3 and then node 4 to go to the destination with p_3. The algorithm might select one of the other feasible paths, too.

5 Simulation Results

In this section, we study the performance of the proposed randomized algorithm. We start by defining our simulation model and some performance measures. Although the performance has been measured for various network topologies, due to lack of space, we present only two of them in this paper.

5.1 Simulation Model and Performance Measures

In our simulation model, a network is given as a directed graph. We measure the performance when each link is associated with two and three additive link weights. For each experiment, link weights are chosen randomly from a given distribution. Then, the simulation program generates random connection requests and tries to find a feasible path for each request using our randomized algorithm. If our algorithm finds a path for a connection request, we count this request as a *routed connection request*. For comparison purposes, we have also implemented an optimum exponential-time algorithm which searches all possible paths in the reduced and labeled graph to determine whether there is a feasible path or not. If the optimum algorithm finds a path for a connection request, we count this request

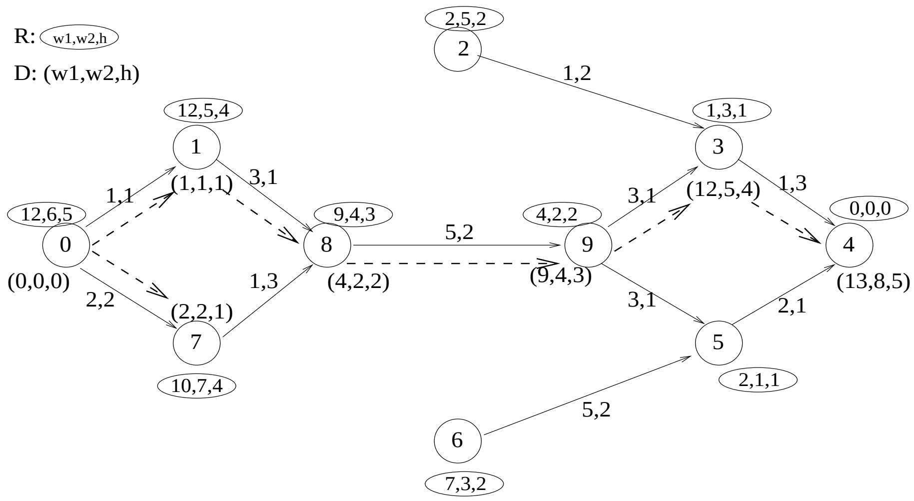

Figure 2: Labeled and reduced graph when $H = 5$.

as a *routed connection requests*, too. In order to compare the performances of our randomized algorithm and the optimum algorithm, we use two measure: *success ratio* and *average hop count*. The *success ratio* shows how often an algorithm finds a path that satisfies the QoS constraints [3]. It is defined as

$$success\ ratio = \frac{Total\ number\ of\ routed\ connection\ requests}{Total\ number\ of\ connection\ requests}$$

The *average hop count* shows the average number of hops per *routed connection request*. It is defined as

$$average\ hop\ count = \frac{Total\ number\ of\ hops\ in\ the\ routed\ connection\ requests}{Total\ number\ of\ routed\ connection\ requests}$$

5.2 Results on Irregular Network Topology

We consider the network topology shown in Figure 3, which is the same network topology modified from ANSNET [5] by inserting additional links. This topology was also used in [3]. We first consider two link weights: $w_1(i,j) = uniform[0,50]$ and $w_2(i,j) = uniform[0,200]$. Source node s and destination node t are selected randomly. For different ranges of C_1 and C_2, we obtain the *success ratio* and *average hop count* based both the optimum algorithm and our randomized algorithm from twenty runs; each run considers randomly generated 2000 connection requests. As shown in Table 1, our randomized algorithm performs as good as the optimum one while the proposed algorithm in [3] approaches the optimum one at the expense of higher overhead, i.e., when x goes to n (the number of nodes in the graph).

We then associate one more weight $w_3(i,j) = uniform[0,100]$ with each link. Accordingly, we add a new constraint C_3. For the different ranges of C_3 with the same ranges of C_1 and C_2 in Table 1, we obtain the *success ratio* and *average hop count* of both the optimum algorithm and our

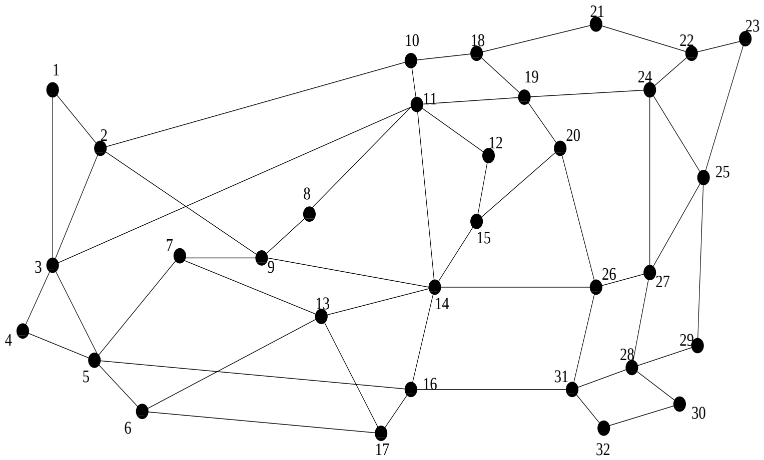

Figure 3: An irregular network topology.

Range for C_1 and C_2	*success ratio* of optimum/our algorithm	*average hop count* of optimum/our algorithm
$C_1 = uniform[100, 115]$ $C_2 = uniform[400, 460]$	0.7595/0.7595	2.7706/2.7706
$C_1 = uniform[50, 65]$ $C_2 = uniform[200, 260]$	0.2594/0.2594	1.7784/1.7784
$C_1 = uniform[75, 90]$ $C_2 = uniform[300, 360]$	0.5220/0.5220	2.3506/2.3506
$C_1 = uniform[125, 140]$ $C_2 = uniform[500, 560]$	0.9219/0.9219	3.0596/3.0596
$C_1 = uniform[150, 165]$ $C_2 = uniform[600, 660]$	**0.9868/0.9867**	3.1729/3.1729

Table 1: Simulation results with two constraints on an irregular topology.

randomized algorithm. As shown in Table 2, our randomized algorithm also performs as good as the optimum algorithm when the path selection subject to three additive constraints.

5.3 Results on Regular Network Topology

We also consider a regular 10x10 mesh topology. Again we first consider two link weights: $w_1(i,j) = uniform[0, 50]$ and $w_2(i,j) = uniform[0, 200]$. s and t are selected randomly. For different range of C_1 and C_2, we obtain the *success ratio* and *average hop count* based on both the optimum algorithm and our randomized algorithm from ten runs; each run considers randomly generated 2000 connection requests. As shown in Table 3, our randomized algorithm performs almost as good as the optimum one in this network, too.

For this experiment, we also associate an additional weight $w_3(i,j) = uniform[0, 100]$ with each link and obtain the *success ratio* and *average hop count* based on both the optimum algorithm

Range for C_3	*success ratio* of optimum/our algorithm	*average hop count* of optimum/our algorithm
$C_3 = uniform[75, 150]$	0.3532/0.3532	2.0814/2.0814
$C_3 = uniform[100, 200]$	0.2102/0.2102	1.5736/1.5736
$C_3 = uniform[150, 250]$	0.4510/0.4510	2.1476/2.1476
$C_3 = uniform[200, 300]$	**0.8438/0.8437**	2.8767/2.8767
$C_3 = uniform[250, 350]$	0.9604/0.9604	**3.1017/3.1018**

Table 2: Simulation results with three constraints on an irregular topology.

Range for C_1 and C_2	*success ratio* of optimum/our algorithm	*average hop count* of optimum/our algorithm
$C_1 = uniform[100, 115]$ $C_2 = uniform[400, 460]$	0.3085/0.3085	3.1465/3.1465
$C_1 = uniform[150, 200]$ $C_2 = uniform[500, 600]$	0.5580/0.5580	4.4204/4.4204
$C_1 = uniform[200, 300]$ $C_2 = uniform[600, 700]$	0.7596/0.7595	5.4093/5.4093
$C_1 = uniform[300, 380]$ $C_2 = uniform[650, 750]$	0.8597/0.8597	**5.9294/5.9297**
$C_1 = uniform[350, 400]$ $C_2 = uniform[700, 800]$	0.9060/0.9060	**6.1568/6.1571**

Table 3: Simulation results with two constraints on a regular topology.

and our randomized algorithm by using different ranges of C_3 with the same ranges of C_1 and C_2 in Table 3. As shown in Table 4, our randomized algorithm shows the same performance as the optimum one when the path selection subject to three additive constraints in this regular topology.

6 Conclusion and Future Work

QoS-based routing subject to multiple additive constraints is an NP-complete problem. We formalized it as MinHop_MC (Definition 1) and proposed a randomized algorithm with a polynomial-time complexity. The algorithm first reduces and labels the original graph. Then, the algorithm uses the procedure of MR_BFS to construct a feasible path by randomly discovering nodes from which there is a chance to go to the final destination. We proved that any path found by the randomized algorithm satisfies the given constraints. The computational complexity of our algorithm is $\mathcal{O}(H_{max}n^2)$ where H_{max} is the maximum-hop-length limit and n is the number of nodes. The storage complexity is $\mathcal{O}(n)$. In terms of both complexities, our algorithm is better than existing algorithms. Its performance is studied by using simulation, which showed that the randomized algorithm and the heuristic behind it work well and find 99.99% of all feasible paths with the minimum hop count.

Range for C_3	*success ratio* of optimum/our algorithm	*average hop count* of optimum/our algorithm
$C_3 = uniform[100..200]$	0.1751/0.1751	2.4353/2.4353
$C_3 = uniform[200..300]$	0.4075/0.4075	3.6885/3.6885
$C_3 = uniform[300..400]$	0.6319/0.6319	4.7615/4.7615
$C_3 = uniform[400..500]$	0.7772/0.7772	5.4713/5.4713
$C_3 = uniform[500..600]$	0.8701/0.8701	5.9394/5.9394

Table 4: Simulation results with three constraints on a regular topology.

The randomized algorithm performs well when the true state of the network is given. However, the true state of the network may not be available to every source node at all times because of network dynamics, aggregation of state information, and latencies in the dissemination of state information. As a future work, we will investigate how our algorithm performs in the presence of inaccurate state information. Because of randomization, different paths can be chosen for the same pair of source and destination at different times. We also plan to investigate how this random path selection strategy affects load balancing and how to compensate for the inaccuracy in the network state information.

Appendix

A Proof of Theorem 1

Assume that the algorithm finds a path $p = (v_0, v_1, v_2, \ldots, v_l)$ where $v_0 = s$ and $v_l = t$. We show that the path p satisfies the given constraints. Let v_{i-1} and v_i be two consecutive nodes on the path p. The procedure of MR_BFS discovers v_i from v_{i-1} if (line 13 in MR_BFS)

$$D_k[v_{i-1}] + w_k(v_{i-1}, v_i) + R_k[v_i] \leq C_k \quad (1)$$

is true for all $k = 1, 2, \ldots, M$. After the discovery of v_i, we have (line 15 in MR_BFS)

$$D_k[v_i] = D_k[v_{i-1}] + w_k(v_{i-1}, v_i) \quad (2)$$

from which we conclude

$$w_k(v_{i-1}, v_i) = D_k[v_i] - D_k[v_{i-1}] \quad (3)$$

From (1) and (2), we have

$$D_k[v_i] + R_k[v_i] \leq C_k D_k[v_i] \leq C_k - R_k[v_i] \quad (4)$$

Summing the weights (w_k for each $k = 1, 2, \ldots, M$) along the path p yields

$$
\begin{aligned}
w_k(p) &\triangleq \sum_{i=1}^{l} w_k(v_{i-1}, v_i) && (5)\\
&= \sum_{i=1}^{l} D_k[v_i] - D_k[v_{i-1}] && (6)\\
&= D_k[v_l] - D_k[v_0] && (7)\\
&= D_k[v_l] && (8)\\
&\leq C_k - R_k[v_l] && (9)\\
&= C_k && (10)
\end{aligned}
$$

In this derivation, (7) comes from the telescoping sum on (6). Then, (8) follows from $D_k[v_0] = D_k[s] = 0$. Finally, (10) follows from $R_k[v_l] = R_k[t] = 0$. Thus, $w_k(p) \leq C_k$ for each $k = 1, 2, \ldots, M$.

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall, Inc., 1993.

[2] C. Aurrecoechea, A.T. Campbell, and L. Hauw. A survey of QoS architectures. *ACM/Springer Verlag Multimedia Systems Journal*, 6(3):138–151, May 1998.

[3] S. Chen and K. Nahrstedt. On finding multi-constrained paths. In *ICC'98*, pages 874 –879. IEEE, 1998.

[4] S. Chen and K Nahrstedt. An overview of quality-of-service routing for the next generation high-speed networks: Problems and solutions. *IEEE Network Magazine, Special Issue on Transmission and Distribution of Digital Video*, 1998.

[5] D. E. Comer. *Internetworking with TCP/IP*, volume I. Prentice Hall, Inc., third edition, 1995.

[6] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* The MIT press and McGraw-Hill book company, sixteenth edition, 1996.

[7] R. Guerin and A. Orda. QoS-based routing in networks with inaccurate information: Theory and algorithms. In *Proceedings of the INFOCOM'97 Conference*, pages 75–83. IEEE, 1997.

[8] A. Iwata and et. al. ATM routing algorithms with multiple QOS requirements for multimedia internetworking. *IEICE Trans. Commun.*, E79-B(8):999–1006, August 1996.

[9] J. M. Jaffe. Algorithms for finding paths with multiple constraints. *Networks*, 14:95–116, 1984.

[10] W. C. Lee, M. G. Hluchyi, and P. A. Humblet. Routing subject to quality of service constraints in integrated communication networks. *IEEE Network*, pages 46–55, July/August 1995.

[11] L. Lovasz. Randomized algorithms in combinatorial optimization. In W. Cook, L. Lovasz, and P. Seymour, editors, *Combinatorial Optimization*, volume 20 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 153–179. 1995.

[12] Q. Ma and P. Steenkiste. On path selection for traffic with bandwidth guarantees. In *Proceedings of IEEE International Conference on Network Protocols*, pages 191 –202, 1997.

[13] Q. Ma and P. Steenkiste. Quality-of-service routing for traffic with performance guarantees. In *Proceedings of the 4th International IFIP Workshop on Quality of Service*, May 1997.

[14] R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.

[15] C. Pornavalai, G. Chakraborty, and N. Shiratori. QoS based routing algorithm in integrated services packet networks. In *Proceedings of the ICNP'97*, pages 167–174. IEEE, 1997.

[16] R. Vogel and et. al. QoS-based routing of multimedia streams in computer networks. *IEEE Journal on Selected Areas in Communications*, 14(7):1235–1244, September 1996.

[17] Z. Wang and J. Crowcroft. Bandwidth-delay based routing algorithms. In *Proceedings of the GLOBECOM'95 Conference*, volume 3, pages 2129–2133. IEEE, Nov. 1995.

[18] Z. Wang and J. Crowcroft. Quality-of-service routing for supporting multimedia applications. *IEEE Journal on Selected Areas in Communications*, 14(7):1228–1234, September 1996.