On Routing Real-Time Multicast Connections*

Ibrahim Matta Liang Guo
College of Computer Science
Northeastern University
Boston, MA 02115
{matta, guol}@ccs.neu.edu

Abstract

Many distributed real-time applications, such as audio- and video-conferencing, require the network to construct a multicast path (tree) from a sender to multiple receivers. Furthermore, real-time applications have Quality-of-Service (QoS) requirements (e.g. bandwidth). The objective of the routing protocol is to build a tree that is both feasible (i.e. satisfies the requested QoS) and least costly. The cost of a tree depends on the costs of its links. The cost of a link should reflect the effect of allocating resources to the new connection on existing and future connections. Many studies have proposed multicast algorithms to construct low-cost QoS-constrained trees. However, these studies assume that some cost is given for each link, and they do not examine the effect of the link cost function.

In this paper, we examine the effect of various link cost functions on the performance of two classes of multicast routing algorithms under both uniform and skewed real-time workload. We also investigate the impact of inaccurate network state information. Our simulation results show that when network state information is accurate (most up-to-date) at each router, the choice of the link cost function only has a negligible effect, if any, on the routing performance. More interestingly, we find that, a link cost function that is more sensitive to load performs better when link state information is relatively accurate, while when link state information is more stale, a link cost function that is less sensitive to load performs better. This is more pronounced under higher load and when multicast groups are more dense.

*This work was supported in part by NSF grants CAREER ANIR-9701988 and MRI EIA-9871022, and NEU grant RSDF-377090.

1 Introduction

Distributed real-time applications, such as audio- and video-conferencing, collaborative environments and distributed interactive simulation, by and large, involve a source sending messages to a selected set of destinations with varying Quality-of-Service (QoS) delivery constraints. This requires the underlying network to provide multicasting (group communication) and QoS capabilities to efficiently support these applications.

The routing of multicast traffic requires the construction of a tree, referred to as the *multicast path*. When a source sends a message, this message is forwarded along the multicast path replicating the message only when the paths to different destinations diverge, thereby taking advantage of common path segments. The objective of the multicast routing protocol is to construct for a newly arriving multicast group (connection) a tree that is (1) feasible (*i.e.* satisfies the QoS requirements), and (2) of minimum *cost*. The cost of a tree is usually defined to be the sum of the costs of its links, where the cost of a link reflects the effect of allocating resources to the new connection on existing and future connections. Finding such a low-cost tree, also known as *Steiner tree*, has been shown to be computationally expensive, and many studies (e.g. [2, 11, 5, 12, 8, 7]) have proposed heuristics to efficiently find such trees. To simplify tree computation, other multicast routing protocols (e.g. [4, 9]) simply build a tree of minimum-cost paths from the source to each individual receiver, where the cost of a path is the sum of the costs of its links. We refer to such a tree as *shortest-path tree*.

Regardless of whether a Steiner tree or shortest-path tree is built, these studies assume that the view of the state of the network, with which the routing algorithm is presented, *readily* associates some cost with each link. Thus, they do not address how the *link cost function* should be defined so as to efficiently distribute allocated resources over the network.

In this paper, we define four link cost functions that are different with respect to their adaptiveness to the link load. We then evaluate these link cost functions via extensive simulations, under different traffic (workload) models and link state update policies. Our evaluation uses the group blocking rate as the primary performance metric. Simulation results show that, when link state is accurately maintained at each node, there is hardly any difference between the different link cost functions. However, when link state information is inaccurate, the adaptiveness of the link cost function could affect the performance of the routing algorithm. More specifically, at frequent updates (i.e. when link state information is more up-to-date), a more adaptive function helps to admit more groups, while with less frequent updates, adaptiveness might cause degradation in performance. The performance difference is more pronounced when network load is high or multicast groups are more dense. Through simulation, we also observe that a routing protocol that builds shortest-path trees costs more and thus provides higher blocking rates than one that builds Steiner trees.

The rest of the paper is organized as follows. Section 2 introduces our network model. Section 3 describes the two classes of multicast routing algorithms; shortest-path and Steiner based. Section 4 discusses various link cost functions that may be used by a multicast routing algorithm. We evaluate these link cost functions for the performance measures defined in Section 5. Section 6 presents our simulation results. Section 7 concludes the paper.

2 Network Model

We assume a real-time network of arbitrary topology. Multicast groups arrive according to a Poisson process. Each group consists of one source and a number of receivers. Each group requests the reservation of a certain amount of bandwidth to satisfy a given QoS. To satisfy the bandwidth requirement, given a view of the current state of the network, the multicast routing algorithm first prunes off infeasible links that do not have enough bandwidth to support the request (*i.e.* links whose available bandwidth is less than the requested bandwidth). Then, the algorithm attempts to find a Steiner or shortest-path tree that connects the source to each receiver. For the Steiner tree, we use a recently proposed heuristic called DDMC [6]. We use Dijkstra's algorithm to build the shortest-path tree. If a tree is found, then the connection is admitted into the network and the requested bandwidth is reserved on each link of the tree for the lifetime of the connection. The lifetime of the connection is assumed to follow a heavy-tailed Pareto distribution. On the other hand, if a tree is not found, the group is rejected and said to be *blocked*.

3 Tree Construction Algorithms

We consider two types of multicast routing algorithms. The first attempts to build a Steiner tree with minimum overall cost, such as the Destination-Driven MultiCasting (DDMC) algorithm recently proposed by Shaikh and Shin [6]. The second type of multicast algorithms simply builds a tree of minimum-cost paths from the source to each of the receivers, such as Dijkstra's shortest-path tree algorithm [1]. We briefly describe Dijkstra's and DDMC algorithms.

Dijkstra's shortest-path multicast algorithm

In Dijkstra's algorithm, the current (partially constructed) tree T is augmented with a new (not-on-tree) node that has the minimum path cost *from the source*, i.e. the cost of a new node $v \notin T$, denoted by $Cost[v]$, via node $u \in T$ is defined to be:

$$Cost[v] = Cost[u] + C(u, v)$$

where $Cost[u]$ is the cost of the minimum-cost path from the source node to node u, and $C(u, v)$ is the cost of the link from node u to node v.

DDMC algorithm

The idea of DDMC comes from the fact that the well-known algorithms of Prim's minimum-spanning tree and Dijkstra's shortest-path tree [1] both use essentially the same greedy strategy. In Prim's algorithm, a minimum-cost tree that spans all nodes is constructed by augmenting the current tree with the minimum-cost link emanating from some on-tree node (*not necessarily the source*). In Dijkstra's algorithm, at each step, a new node with the minimum path cost *from the source* is added to the current tree. Thus the difference between the two algorithms lies in the choice of the cost function used at each step. In Prim's algorithm, the cost of a new (not-on-tree) node to be added to the current tree is simply the cost of the minimum-cost link to it, whereas in Dijkstra's, this cost is that of the minimum-cost path from the source to the node.

The DDMC algorithm defines the cost of a new node $v \notin T$ via node $u \in T$ to be:

$$Cost[v] = I_D(u) Cost[u] + C(u, v)$$

where the "indicator function" $I_D(u)$ is defined as

$$ID(u) = \begin{cases} 0 & \text{if } u \text{ is a receiver} \\ 1 & \text{otherwise} \end{cases}$$

The algorithm thus tries to make the destination nodes appear as new "sources". This gives preference to the paths going through destination nodes since the path to a new node via a destination node is likely to have a lower cost and thus be added to the tree. Since we have to absorb the cost of reaching a destination node anyway, the DDMC algorithm can multicast to all destination nodes on a low-cost tree.

4 Link Cost Functions

We investigate several link cost functions $C(.)$. One function is *utilization-based*, where the cost of a link is the fraction of link capacity currently reserved for existing groups (connections), in addition to what will be reserved for the new connection if it is admitted. Specifically,

$$LinkCost = \frac{RsvBw + ReqBw}{LinkCap} \tag{1}$$

where $RsvBw$ is the amount of bandwidth currently in use by existing connections, $ReqBw$ is the amount of bandwidth requested by the newly arriving group, and $LinkCap$ is the total bandwidth of the link. The idea behind this cost definition is to choose a tree that is least loaded, and at the same time, to minimize the total amount of bandwidth to be consumed by the new connection. Note that for a large bandwidth request, this scheme would further discourage the use of long paths (that is, paths with large number of links) as it becomes less sensitive to link utilizations/loads. We refer to this scheme as *UTIL_D*. We also consider a variant which accounts for the length of the path (in number of hops) as follows:

$$LinkCost = 1 + \frac{RsvBw}{LinkCap} \tag{2}$$

Since link load is always less than or equal to 1, hop count, a static link metric, plays a dominant role in the link cost. We refer to this scheme as *UTIL_F*.

Another link cost function we investigate is of *exponential* type, where the link cost increases exponentially with the utilization of the link. Specifically,

$$LinkCost = \frac{1}{LinkCap - (RsvBw + ReqBw)} \tag{3}$$

We refer to this cost function as *EXP*. To compare between this function and a utilization-based one, consider two paths P_1 and P_2. Suppose P_1 consists of only one link, and P_2 consists of two links. Also consider the utilization-based and exponential link cost functions defined in (1) and (3), respectively, assuming link capacity is 1 and the requested bandwidth is very small or negligible. Suppose P_1's link utilization is 0.8, and P_2's first link utilization is 0.4 and second link utilization is 0.5. Then, under UTIL_D, path P_1 is selected as it has lower utilization (cost) than P_2 (of course, given that P_1 is feasible); P_1's cost equals 0.8, and P_2's cost equals 0.4+0.5=0.9. On the other hand, under EXP, path P_2 is selected since P_1's cost equals 1/(1-0.8)=5, which is higher than P_2's cost of 1/(1-0.4)+1/(1-0.5)=3.67. One can argue that path P_2 is a better choice as it avoids using the highly loaded path P_1. Using P_1 makes it even more loaded, and may become infeasible for future connections.

Now, suppose P_1's link utilization is 0.3, and P_2's first link utilization is 0.1 and second link utilization is also 0.1. Then, under UTIL_D, path P_2 is selected as it has lower utilization (cost) than P_1; P_1's cost equals 0.3, and P_2's cost equals 0.1+0.1=0.2. On the other hand, under EXP, path P_1 is selected since P_1's cost equals 1/(1-0.3)=1.43, which is lower than P_2's cost of 1/(1-0.1)+1/(1-0.1)=2.22. One can argue that path P_1 is a better choice as it avoids the longer (two-link) path P_2, thus bandwidth needs to be reserved over P_1 on only one link. Observe that P_1 is still not heavily loaded, and thus is far from being infeasible for future connections.

To summarize, an exponential function appears more attractive than a utilization-based one. An exponential function can better distribute the load over the network by avoiding the use of highly loaded links, and thus links don't become saturated or infeasible for future connections. Furthermore, an exponential function gives preference to shorter paths (with less number of links) as long as links are not heavily loaded. This reduces the amount of bandwidth that is reserved (consumed) over the selected path, thus allowing the network to be able to support more connections (*i.e.* have higher throughput). However, these advantages are based on the premise that the link utilization information, which varies over time, is accurate. Inaccurate information may result in directing traffic on an already heavily loaded link, thus degenerating network traffic conditions.

We also consider a scheme where the cost of a link is taken to be 1. Thus, we compute a Steiner tree with minimum number of links or a tree of shortest (minimum-hop) paths. We first compute such a tree given the (original) graph representing the network. We call this tree a *minimum-hop tree (MHT)*. Then, the scheme considers subgraphs G^u where every link has a utilization $\leq u$, $u \in [U_{min}, U_{max})$, and U_{min} and U_{max} are the minimum and maximum link utilization respectively. Note that G^u is a subset of the original network graph since we remove all links whose utilization $> u$. Thus, considering G^u results in the computation of a tree with same or larger cost (*or* number of links since the cost of a link is set to 1) than that of the minimum-hop tree, but that tree may be less loaded. Furthermore, for $u_1 < u_2$, G^{u_1} results in a tree with same or larger cost (number of links) than that of G^{u_2}. We consider values of u that start from U_{max} and decremented for each successive subgraph by 0.1. Thus, only utilization levels $u \in [0, U_{max}]$ are considered. Furthermore, we stop at utilization level v if the following condition holds:

$$Cost(T(G^v)) - Cost(MHT) > w \tag{4}$$

where $Cost(T(G^v))$ is the cost of tree corresponding to utilization level v. This condition limits the size of the tree. Unless otherwise specified, we take $w = 4$ if $v \leq 0.7$, and $w = 0$ for higher utilization levels. Thus, from the range of trees computed at various utilization levels, one can choose a tree that achieves a good balance between size and load. Keeping tree size small results in lower bandwidth consumed by the new connection. Keeping tree load low results in better distribution of connections over the network, thus avoiding overloading some network links. We refer to this scheme as *UTIL_HOP*. In our simulations, we choose the tree that corresponds to G^k, where k is U_{min} if (4) is not satisfied for all considered utilization levels. Otherwise, k is the utilization level just preceding the one for which (4) is satisfied. Thus, for an arriving multicast group, a tree that is least loaded but limited in size is selected.

Figure 1 roughly reflects the difference among the above four definitions of link cost function.

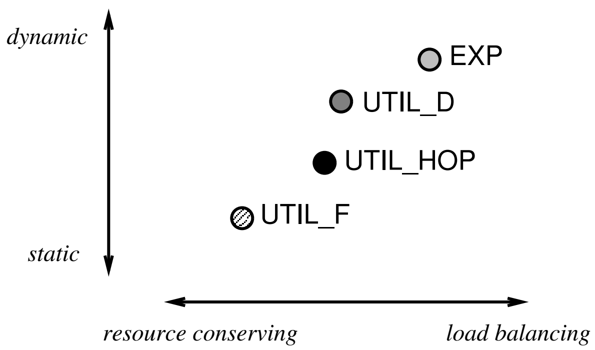


Figure 1. Relationship between cost functions

5 Performance Measures

We choose group blocking rate as the primary performance measure since it reflects the efficiency of managing the network resources. The blocking rate is computed as the percentage of groups being rejected by the network over the total number of group arrivals. A group can be rejected either because no routing tree could be found after pruning off infeasible links or because the *actual* available resources on some tree links could not accommodate the request due to inaccurate routing information.

6 Simulation Results

We present results for a 100-node network. The network is randomly generated using a modified version of the graph generation algorithm described in [10]. Each link has a total capacity of 155 Mbps, of which 85% can be allocated for real-time multicast connections, and the other 15% for other types of connections (e.g. best-effort). Thus, the "effective" link capacity available for allocation to real-time multicast traffic is $0.85 \times 155 = 131.75$ Mbps. Multicast groups arrive according to a Poisson process. We study two kinds of groups, a sparse group which has one source and 10 receivers, and a dense group which has one source and 20 receivers. Receivers are uniformly distributed over the nodes of the network. The source of a group is chosen randomly from the set of *all* nodes—we refer to this scenario as uniform workload, or the source is chosen randomly from a *subset* of nodes—we refer to this scenario as skewed workload. Each group requests the reservation of an amount of bandwidth chosen randomly from $[2 - 20]$ Mbps on each link of the tree connecting the source to the various receivers. If such feasible tree is found, the group is admitted and the requested bandwidth is reserved for the lifetime of the group, which is heavy-tailed Pareto distributed with mean of 15 minutes.

We also model inaccuracies in the network state information. The link state is updated at some fixed rate, therefore, the link state kept at each node might be different from the actual link state. At one extreme, we assume that link state is always up-to-date at each node. Although this is an unrealistic case, we use it mainly as reference. At the other extreme, we assume that link state is updated over a long period (5 minutes in our simulations). We also consider moderate update frequency where link state is updated every 1 minute. We first present general observations about the results, and then the details of the experiments under different workload conditions and link state update policies.

General Observations

- A more adaptive link cost function helps balance traffic over the network, thus reduce the chance of having some highly loaded links. However, the goal of load-balancing is usually achieved by using longer, but lightly loaded paths. Thus, more network resources are consumed for individual connections, which reduces the chance of accepting future requests. Thus, there is a tradeoff between minimizing the resource consumption of individual connections and spreading the network load.

- When link state is accurately maintained at each node, all the four link cost functions provides similar performance. This is true for both Steiner (DDMC) and shortest-path (Dijkstra's) tree construction algorithms, under both uniform and skewed workload, when groups are sparse or dense. This implies that, when link state information is accurate, there is a balance between the above two competing goals. Note that Section 4 suggested that EXP should outperform other schemes because it attempts to adjust its behavior based on network traffic conditions, that is, minimize resource consumption when network traffic is light and balance load when network traffic is high. However, for the traffic parameters used in our experiments, our results indicate that EXP tends to emphasize load balancing at the expense of using more resources.

- However, when inaccuracies are present, the balance is compromised. More specifically, when link state updates are relatively frequent, a more adaptive (or more load-balancing oriented) scheme helps in managing network resources. This is because with an adaptive scheme, the chance of having a highly loaded link is smaller. Thus, less links will be pruned off because of infeasibility, and as a consequence, the routing algorithm will have more candidate links in building a multicast path. As a result, the success rate of finding a feasible tree is higher.

- When link state updates are much less frequent so that the load information kept at each node may be significantly different from the current link state, an adaptive scheme might perform worse than a relatively static scheme. This is because the ability to balance network load completely depends on the dynamic information provided by the underlying network. Making wrong decisions because of inaccurate link state information might make network congestion even worse. In this sense, a relatively static scheme might perform better since it mainly uses static link state information.

We next present simulation results. Due to lack of space, we refer the reader to [3] for more simulation results.

Relatively Accurate Link State Information

Figure 2 shows results for sparse groups under skewed workload. Here, the link state is updated every 1 minute, and the advertised value is the average load on the link over the last 2 minutes. Figure 2(a) shows that when the Steiner tree algorithm is used, the most adaptive cost function EXP blocks the least number of connections, followed by UTIL_D, which is less adaptive than EXP, followed by UTIL_HOP and UTIL_F. The performance gap is more pronounced at higher load (higher multicast request arrival rate), and can be as high as 2% of total connection requests. We can observe similar results from Figure 2(b), in which case the routing algorithm builds a shortest-path tree for each group. Here, the blocking rate for each link cost function is higher than that obtained with Steiner trees, and the performance gap between different functions is more significant (as high as 5%). However, the relative order remains the same. Similar results were obtained for dense groups and under uniform workload.

Less Accurate Link State Information

Figure 3 shows performance measures for various link cost functions for both Steiner and shortest-path tree construction algorithms for sparse groups under skewed workload. In this case, the link state is updated every 5 minutes, and the advertised value is the average load on the link over the last 10 minutes. Figure 3(a) shows the results for the Steiner tree algorithm. The performance of the most adaptive cost function, EXP, degrades as network load increases. While the relative order of the other three functions remains the same as the network load increases. More specifically, UTIL_D outperforms UTIL_HOP, and UTIL_F performs better than both UTIL_D and UTIL_HOP. The performance gap is more pronounced at high network loads, and can be as high as 5%. We observe similar results with shortest-path trees, as shown in Figure 3(b). Here, the difference is that the performance gap is not so significant, but the absolute blocking rate for each individual cost function is higher than in the Steiner tree case. Similar results were obtained for dense groups, where EXP is found to perform relatively worse.

7 Conclusion and Future Work

Many studies (e.g. [2, 11, 5, 12, 8, 7]) have proposed and evaluated algorithms for selecting low-cost QoS-constrained multicast trees. However, none of these studies have examined how the link cost function should be defined so as to efficiently distribute connections over the network. The cost function should provide a good balance between load balancing and bandwidth consumption. A "good" link cost function would distribute load over the network so as to avoid overloading links. As links become overloaded and infeasible to support incoming real-time connections, the routing algorithm may be forced to choose less loaded and feasible trees that are large (i.e., consist of a large number of links). To guarantee QoS, bandwidth is then reserved over this large number of links. This quickly consumes network bandwidth and results in increased connection blocking rate (or equivalently, reduced throughput and revenue). Thus, a "good" cost function should also impose a limit on the size of the computed multicast trees so as to reduce bandwidth consumption.

We compared four possible definitions of link cost functions which strike a different balance between load balancing and minimizing resource consumption. Through simulations, we found that when link state information is relatively accurate, a cost function that is better at balancing network load performs better. This is because if network traffic is unbalanced, it is more likely to have some infeasible links that get pruned off. As a consequence, less candidate paths are available when routing is performed, thus the chance of blocking a new group request increases.

However, as link state information becomes stale, the load balancing objective, which strongly depends on the dynamic state information, may not be achieved due to wrong decisions in building the multicast trees. This may create load imbalance, which together with performance loss due to large resource consumption, contributes to more connections being rejected. On the contrary, cost functions whose major objective is to minimize resource consumption mainly use static link metrics. Thus, they are less sensitive to inaccuracies in link state information, and perform well even in the presence of stale state information.

References

[1] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. The MIT Press and McGraw-Hill Book Company, 1990.

[2] V.P. Kompella, J.C. Pasquale, and G.C. Polyzos. Multicasting for Multimedia Applications. In Proc. *IEEE INFOCOM'92*, pages 2078–2085, 1992.

[3] I. Matta and L. Guo. On Routing Real-Time Multicast Connections. NU-CCS-98-14, Northeastern University, College of Computer Science, December 1998.

[4] J. Moy. Multicast Extensions to OSPF. Internet draft, Network Working Group, September 1992.

[5] G.N. Rouskas. Multicast Routing with End-to-End Delay and Delay Variation Constriants. *IEEE J. Select. Areas Commun.*, 15:346–357, April 1997.

[6] A. Shaikh and K. Shin. Destination-Driven Routing for Low-Cost Multicast. *IEEE J. Select. Areas Commun.*, 15:373–381, April 1997.

[7] Q. Sun and H. Langendoerfer. Efficient Multicast Routing for Delay-Sensitive Applications. In Proc. *Second Workshop on Protocols for Multimedia Systems (PROMS)*, pages 452–458, October 1995.

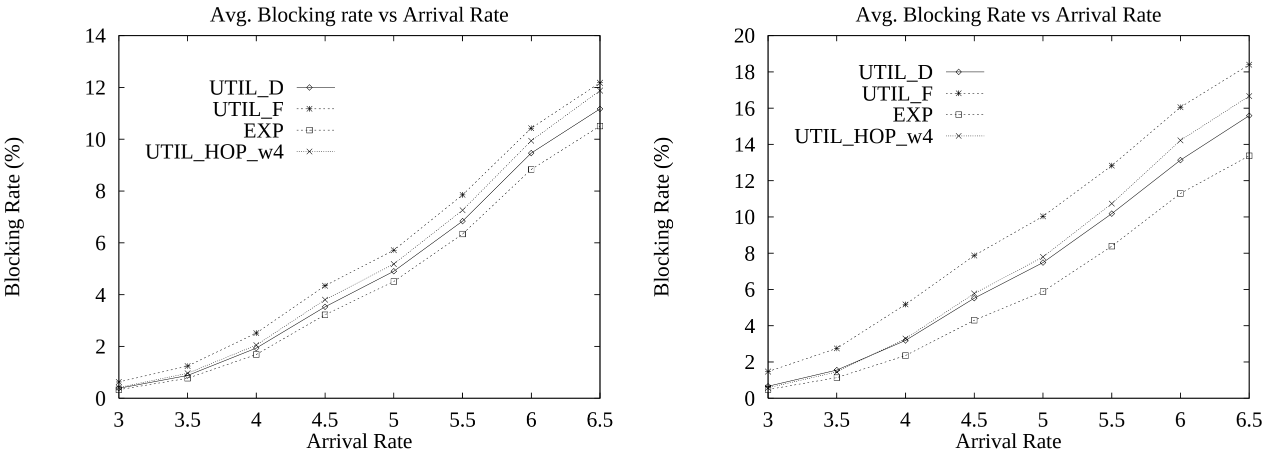
[8] Q. Sun and H. Langendoerfer. An Efficient Delay-Constrained Multicast Routing Algorithm. Technical Report Internal Report, Institute of Operating Systems and Computer Networks, TU Braunschweig, Bueltenweg 74/75, 38106, Braunschweig, Germany, January 1997.

[9] D. Waitzman, C. Partridge, and S. Deering. Distance Vector Multicast Routing Protocol. Request for Comments RFC-1175, November 1988.

[10] B.M. Waxman. Routing of Multipoint Connections. *IEEE J. Select. Areas Commun.*, pages 1617–1622, December 1988.

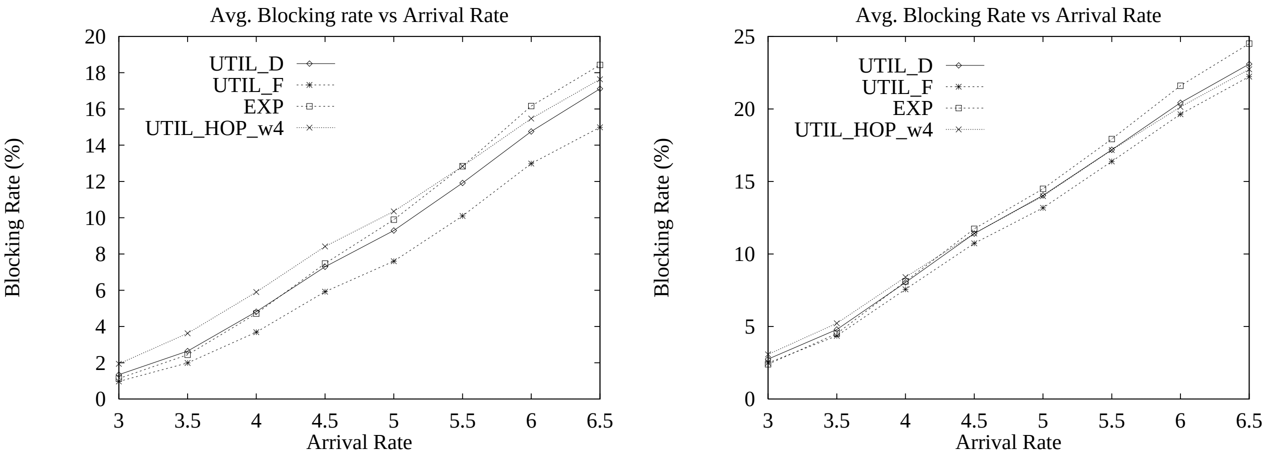
[11] R. Widyono. The Design and Evaluation of Routing Algorithms for Real-Time Channels. Technical Report ICSI TR-94-024, International Computer Science Institute, U.C. Berkeley, June 1994.

[12] Q. Zhu, M. Parsa, and J.J. Garcia-Luna-Aceves. A Source-Based Algorithm for Delay-Constrained Minimum-Cost Multicasting. In Proc. *IEEE INFOCOM'95*, pages 377–385, 1995.



(a) (b)

Figure 2. Blocking probability versus arrival rate, skewed workload, link state update rate 1.0, sparse groups, (a) Steiner Tree (b) Shortest-path Tree.



(a) (b)

Figure 3. Blocking probability versus arrival rate, skewed workload, link state update rate 5.0, sparse groups, (a) Steiner Tree (b) Shortest-path Tree.