Quality-of-Service Routing Using Maximally Disjoint Paths

Nina Taft-Plotkin[†]
Sprint Advanced Technology Labs
Burlingame, CA. USA

Bhargav Bellur and Richard Ogier
SRI International
Menlo Park, CA. USA

Abstract— **We apply and evaluate a new efficient algorithm for finding maximally link disjoint pairs of paths in a network. We apply this algorithm for QoS routing in connection-oriented networks that support calls with multiple QoS requirements. Our algorithm (called MADSWIP) is applied for precomputing paths in advance of call arrivals. Through simulations, we compare our QoS routing method to another method that is typical of what a switch vendor might implement today. We then examine the performance of three different policies for selecting a path among multiple potential paths. We also study the effects of decreasing the density of a network topology. We study two styles of topologies: commercial style and random topologies. We demonstrate that precomputing paths that have minimal overlap is more important than precomputing paths that explicitly address all QoS metrics in a network. We also show that load balancing policies outperform call packing policies in networks that support diverse applications.** [§]

I. Introduction

The QoS routing problem that we address is to develop an approach to precomputed path routing based on the MADSWIP algorithm presented in [5]. This algorithm computes maximally link disjoint paths that optimize either bandwidth or delay. A pair of paths from a source to a destination is said to be maximally link-disjoint if the number of links common to both paths is minimum. Our approach also includes the selection of the types of precomputed paths to calculate and policies for selecting among multiple paths. We study the benefits of using maximally link-disjoint paths. We use this approach to study the impact on the blocking rate of changing the density of a topology. Changing the density (and often therefore the diameter) of a network can make the network more (or less) "difficult" in terms of its ability to satisfy delay and jitter requirements. We study both commercial-style and random topologies.

Both the ATM Forum [2] and the IETF [1] have advocated the ideas of using alternate paths and crankback for QoS routing. However, there are many ways of using alternate paths. In our approach, we choose an algorithm that finds pairs of paths with a minimum number of common links. We consider a network environment that supports connection-oriented circuits (although the method could be applied to Internet flows as well) in which applications specify two or three QoS requirements.

In [1], the authors develop a framework for QoS routing that defines the scope and objectives of the QoS routing task, desirable but optional features, issues for performance evaluation, metrics for decision making, etc. Our work addresses a number of the issues raised in this document such as useful criteria for multiple path determination, and policies for selecting between multiple paths. In [4] the authors develop an algorithm for finding a path that satisfies bandwidth, delay and jitter constraints in polynomial time. Their algorithm makes use of the relation among these three constraints as determined by an underlying weighted fair queueing scheduler. Wang and Crowcroft [8] suggest the idea of shortest-widest paths and present an algorithm for finding such paths. Guerin and Orda [3] establish routing schemes that are capable of operating efficiently in the presence of inaccurate information. Inaccurate information may come either from topology aggregation or outdated delay advertisements. In [6], the author exploits properties of hierarchical structure in an approximation algorithm for QoS routing that improves scalability performance.

All these methods address important issues and difficult problems within QoS routing. However, none of them develop algorithms that are based on finding two paths simultaneously such that the pair of paths is guaranteed to satisfy some property regarding overlaping resources. In our approach, we focus on the delay and bandwidth QoS metrics as well as maximally link disjoint paths. It is desirable to compute maximally disjoint pairs of paths to each destination so that any link with low resource availability is highly unlikely to belong to both paths.

We studied the performance of our method in five different topologies assuming four different multimedia applications. In Section III-D we compare our approach to another approach that explicitly addresses all QoS metrics in a network. This comparison allows us to contrast two different approaches: one that only explicitly accounts for two QoS metrics (bandwidth and delay) but couples this with alternate paths, to another approach that chooses to focus on all QoS metrics (one at a time) at the expense of using no alternate paths. We then study (in Section V) the performance of different policies (such as load balancing and call packing) for selecting a path among the multiple precomputed paths. We studied the influence of topology

[†]The research in this paper was conducted while Nina Taft-Plotkin was an employee of SRI International.

[§]This work was generously funded by a grant from Sprint.

on the blocking rate and call setup time of connections in Section VI.

II. Problem Statement and Solution

A user's call request is specified by a destination and a triple (B, D, J) where B denotes the bandwidth requirement, D the end-to-end delay requirement, and J denote the end-to-end jitter requirement. Each link i is characterized by (b_i, d_i, j_i) which specifies the bandwidth, delay, and jitter (respectively) on link i. (For representation, we cluster a node's delay and jitter together with an outgoing link's characteristics.) The basic QoS routing problem we address is to find paths that satisfy $\min_{i \in P} b_i \geq B$, $\sum_{i \in P} d_i < D$, and $\sum_{i \in P} j_i < J$ where P denotes the set of links in a candidate path.

Both the IETF [1] and the ATM Forum [2] have developed frameworks that include algorithms for precomputing paths. The algorithm for precomputing paths is run at each node, and precomputed paths are found from that node to every destination in the network. These precomputed paths are updated periodically. The update period is a parameter settable by the network operator. The network determines the QoS level of the paths since precomputed paths are *not* computed in response to call requests.

A key design issue in QoS routing is to select a small set of *useful* paths to precompute. This involves selecting the number of paths to precompute between each source and destination, and the type of path. We precomputed six paths to each destination: the minimum delay path; a pair of maximally link disjoint paths such that the sum of the delay on both paths is minimized; the maximum-bandwidth path; and a pair of maximally link disjoint paths of maximum bandwidth (defined below). In the case when multiple pairs with the same number of common links and the same bandwidth exist, we break ties using delay as a secondary objective.

We assume an environment in which network QoS status information is maintained either in a topology database or a routing table at each node. This information should include recent QoS status of network links and nodes, such as the triple (b, d, j). We assume the network has a protocol that updates this information regularly. For example, in PNNI [2] nodes flood the network with QoS status messages either when major changes in status occur or after a prespecified time interval.

We now give some typical numbers to demonstrate the dynamicity with which information is updated. In our simulations, a node advertises its QoS status every 300 seconds (if it hasn't already sent a QoS update message due to a major status change.) Since the emission of these messages from network nodes is not synchronized, a node receives such messages in a staggered fashion. The 300 second update interval means that one node's information about another specific node will be updated at least once every 300 seconds. However, update messages arrive throughout a 300 second time interval since one node is receiving status information from multiple network nodes. In our testing, the precomputed paths at a node are recomputed every 100 seconds, using the most recent version of that node's topology database. The precomputed paths algorithm is thus an on-line algorithm since it is executed regularly. However, the algorithm does not have not strict real-time requirements because there is no call waiting to be routed during the algorithm's execution. The algorithm should of course be fast for the purpose of not using excessive amounts of a switch's processing time.

The motivation for having precomputed paths is to reduce the call setup time. If the system were to deploy an on-demand algorithm to be executed after the call request (with its specific QoS requirements) has arrived, then the call setup time would include the time to run this algorithm. Clearly routing calls over precomputed paths rather than over paths computed on-demand will lead to substantially smaller setup times. The tradeoff is that an on-demand algorithm would produce candidate paths that are more likely to accept a call since it does so using more recent QoS status information.

A. Response to a Call Request

A general method for responding to connection requests with QoS requirements is described in Figure 1. This is a hybrid approach since it includes both a precomputed paths algorithm and an on-demand algorithm. The precomputed paths algorithm computes paths to be used during Phase I. If a call cannot be routed on one of the precomputed paths, then it enters Phase II in which the on-demand algorithm is executed.

When a call request arrives, the source node selects a candidate path from the list of precomputed paths. The node first makes a local check by comparing the most recent QoS of the precomputed path (obtained from its topology database) with the user's QoS requirement. If the path appears to satisfy the requirements then the connection setup phase is initiated. The setup phase involves transmitting signaling messages along the path so that each node can check its resources and verify whether or not it can accept the call (i.e., local call admission control (CAC)). In ATM networks, this procedure is called PNNI Setup, as indicated in Figure 1.

Passing the local check does not imply that the call will be accepted along the path because the topology database at the source node can be out-of-date from the actual network status. This first path can fail either the local check or during the call setup. If this happens another precomputed path can be tried. The maximum number of precomputed paths that can be tried is a parameter that should be set by a network provider. The order in which the precomputed paths are checked is determined by a *multi-route selection policy.*

If any of the nodes cannot accept the call, then a procedure called crankback occurs. Crankback is the return of the call setup attempt to its source (or the closest border

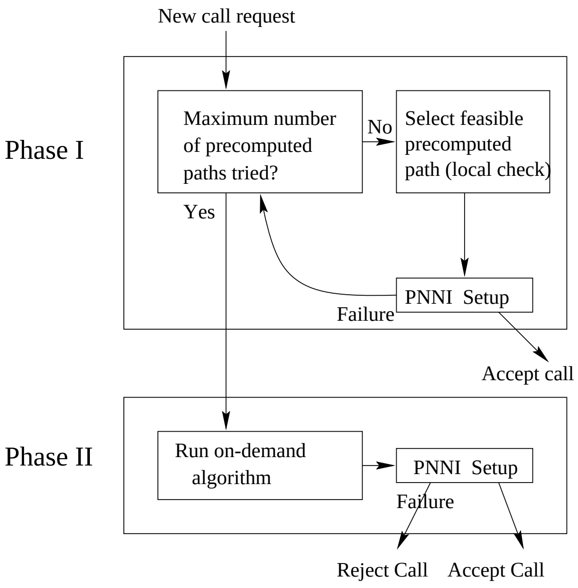

Fig. 1. Response to a Call Request

node of the current domain or peer group). Crankback in Phase I is depicted as "Failure" in Figure 1. If the call setup time limit has not been exceeded, an alternate path can be tried. Crankback messages typically contain information regarding the node or link that blocked the call. We only try other precomputed paths if they avoid the blocking link. In this approach, a call is routed over the first feasible path found.

If none of the precomputed paths can route the call then an on-demand algorithm can be executed. It has been suggested that if a precomputed paths algorithm is sufficiently successful at routing calls, then Phase II may not be needed at all. In this paper we propose only one algorithm that is appropriate for a precomputed paths algorithm, but not for an on-demand algorithm. We study the behavior of this algorithm alone. In more recent efforts, we are studying hybrid systems which utilize two algorithms.

B. The MADSWIP Algorithm

This section briefly describes the Maximally Disjoint Shortest and Widest Paths (MADSWIP) algorithm. A complete description, along with a proof of correctness and the theory behind MADSWIP, can be found in [5]. The MADSWIP algorithm efficiently computes pairs of maximally disjoint paths from a source node to all destination nodes, such that the pair to each destination has either minimum cost or maximum bandwidth.

We assume that the network topology is described by a given directed graph $G = (V, E)$ where V is the set of nodes and E is the set of directed links. The source node is denoted by s. A cost $c(i, j)$ and a bandwidth $b(i, j)$ are assigned to each link (i, j). The algorithm can be run for different choices of $c(i, j)$ in order to satisfy the different QoS requirements of different calls. The cost can be based on additive or multiplicative metrics such as delay, jitter, hop count, administrative weight, and loss probability. Note that a multiplicative metric can be converted to an additive metric simply by taking the negative log of it.

Given a path P, we define the length $c(P)$ of P to be the sum of $c(i, j)$ over all links (i, j) in the path, and the bandwidth $b(P)$ of P to be the minimum of $b(i, j)$ over all links (i, j) in the path. Given a pair $(P1, P2)$ of paths, we define the length $c(P1, P2)$ of the pair to be $c(P1) + c(P2)$ and the bandwidth $b(P1, P2)$ to be $\min\{b(P1), b(P2)\}$.

MADSWIP is a nontrivial modification of the algorithm of Suurballe and Tarjan [7]. The latter algorithm, which we will call the S-T algorithm, computes pairs of *completely* disjoint paths from a source node to all destination nodes, such that the pair to each destination has minimum total cost. MADSWIP provides two main extensions to the S-T algorithm. First, it computes *maximally* disjoint paths to each destination for which disjoint paths do not exist. (The S-T algorithm does not generate any paths if completely disjoint paths do not exist.) Second, it can compute *maximum bandwidth* disjoint or maximally disjoint paths. The complexity of MADSWIP is $O(|E| \log |V|)$, the same as Dijkstra's algorithm and the S-T algorithm.

Each execution of MADSWIP actually computes *three* paths to each destination: the two maximally disjoint paths and the optimal path for the given routing criteria. The optimal paths are computed as the first phase of MADSWIP, which consists of the four phases described below. The general version of MADSWIP (described below) computes maximum-bandwidth maximally disjoint paths, and minimizes delay as a secondary objective. The special case of MADSWIP in which $b(i, j)$ is identical for all links computes minimum-cost maximally disjoint paths.

Phase 1. The first phase of MADSWIP computes maximum-bandwidth paths from the source s to all destinations by applying a variant of Dijkstra's algorithm. The algorithm includes a lexicographic minimization of $(1/b(P), c(P))$, so that among all maximum-bandwidth-path spanning trees, the one resulting in the shortest paths is selected. This algorithm was presented by Wang and Crowcroft [8] for computing *shortest-widest* paths.

Phase 1 computes the functions $p(v)$, $B(v)$, and $d(v)$, which have the following meaning. The node $p(v)$ is the next-to-last node in the computed path to v, $B(v)$ is the bandwidth of the path, and $d(v)$ is the cost of the path. Thus, the links $(p(v), v)$ define a tree T that contains the maximum-bandwidth paths from s to all nodes.

Phase 2. Next, we transform the bandwidth and cost of each link (v, w) as follows:

$$b'(v, w) = \min\{B(v), b(v, w)\} \quad (1)$$

$$c'(v, w) = c(v, w) - d(w) + d(v) \quad (2)$$

This transformation has no effect on the bandwidth of any path from s, since $B(v)$ is the maximum bandwidth of any path to v. Note that $b'(v,w) = B(w)$ for any tree link (v,w), since in this case $B(w) = min\{B(v), b(v,w)\}$. This transformation also does not affect the relative cost of paths to a given node v, since it subtracts $d(v)$ from the cost of each such path. We also have $c'(v,w) = 0$ for any tree link (v,w).

Phase 3. This phase maintains the following variables for each node v: $r(v)$, $q(v)$, $d'(v)$, $G(v)$ and $B'(v)$. When the algorithm terminates $B'(v)$ will be the minimum bandwidth of the two computed paths to v; $G(v)$ will be the number of links common to the two paths; $d'(v)$ will be the total cost of the two paths using modified link costs $c'(i,j)$. The variables $r(v)$, $q(v)$ and $p(v)$ are used to construct the two disjoint paths. The link $(r(v), v)$ is the last link on one of the maximally disjoint paths to v, and is either a nontree link or is common to both paths. If $q(v) \neq s$ then link $(r(u), u)$, where $u = q(v)$, is also on one of the maximally disjoint paths to v, and is either a nontree link or is common to both paths.

As in Dijkstra's algorithm, Phase 3 labels nodes one at a time. As in the S-T algorithm, Phase 3 requires the following concept. Removing the labeled nodes from the shortest-path tree T divides T into subtrees that span the set of unlabeled nodes. These subtrees are called *unlabeled subtrees*. Initially, there is only one unlabeled subtree, namely T.

Initially, $G(s) = 0$, $G(v) = \infty$ for all v other than s, $B'(s) = \infty$, $B'(v) = 0$ for all $v \neq s$, $d(s) = 0$, $d(v) = \infty$ for all $v \neq s$, and $r(v)$ and $q(v)$ are undefined for every node v. The algorithm consists of repeating the following step until there is no unlabeled node v with $G(v)$ finite.

Labeling Step of Phase 3: Choose an unlabeled node v such that $(G(v), 1/B'(v), d'(v))$ is lexicographically minimized. Let S be the unlabeled subtree containing v. Make v labeled. This splits S into new unlabeled subtrees; there is one new subtree for the parent $p(v)$ of v, if it exists and is unlabeled, and one new subtree for each unlabeled child of v. For each nontree link (u,w) such that u and w are in S and either $u = v$ or u and w are in different unlabeled subtrees after v is labeled, if

$$\begin{aligned}(G(v), 1/\min\{B'(v), b'(u,w)\}, d'(v) + c'(u,w))\\ < (G(w), 1/B'(w), d'(w)),\quad (3)\end{aligned}$$

then set $G(w) = G(v)$, $B'(w) = \min\{B'(v), b'(u,w)\}$, $d'(w) = d'(v) + c'(u,w)$, $r(w) = u$, and $q(w) = v$. For each tree link (v,w) such that w is unlabeled (i.e., for each unlabeled child w of v), if

$$\begin{aligned}(G(v) + 1, 1/\min\{B'(v), b'(v,w)\}, d'(v) + c'(v,w))\\ < (G(w), 1/B'(w), d'(w)),\quad (4)\end{aligned}$$

then set $G(w) = G(v) + 1$, $B'(w) = \min\{B'(v), b'(v,w)\}$, $d'(w) = d'(v)$, $r(w) = v$ $(= p(w))$, and $q(w) = v$.

A link (u,w) is said to be processed when the inequality in Equation (3) or (4) is tested. The S-T algorithm uses a

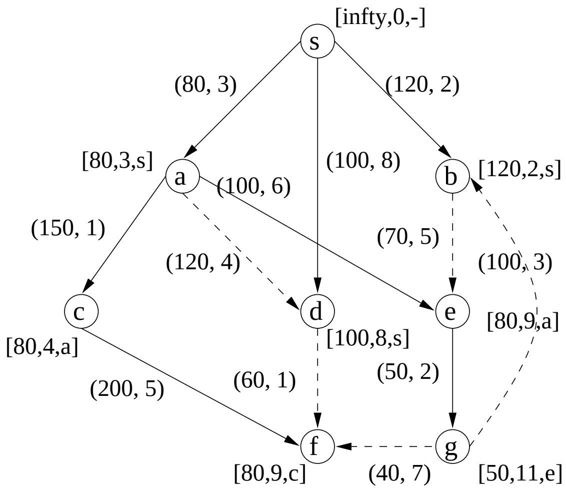

Fig. 2. At the completion of Phase 1.

d-heap with $d = q(1+m/n)$ to store the unlabeled vertices. The only added complication is the need for a mechanism to determine which links to process when a node is labeled. Suurballe and Tarjan provide an algorithm for determining these links in $O(|E| \log |V|)$ time.

Phase 4. Given $p(w)$, $r(w)$, and $q(w)$ for each node w, the procedure for extracting the two disjoint paths from the source s to a destination v is as follows. The procedure first marks certain nodes in an initialization step, and then constructs each path by traversing it backwards. To carry out the initialization step, we begin with all vertices unmarked and then mark $v, q(v), q(q(v))$, etc., until s is reached. To construct one of the paths, we let $x = v$ and we repeat the following step until $x = s$.

Traversal Step. If x is marked, unmark x, add $(r(x), x)$ to the front of the path, and replace x with $r(x)$. Otherwise, add $(p(x), x)$ to the front of the path and replace x with $p(x)$.

To construct the second path, we again let $x = v$ and repeat the traversal step until $x = s$.

C. Example

In Figures 2 to 4, we present a simple example to illustrate the different steps in the execution of MADSWIP. Note that in this example, all the links are *unidirectional*. MADSWIP will work correctly in networks with bidirectional links. (In this case, the reverse of a tree link, if it exists, is a *non-tree* link.)

In Figure 2, the label of each link (v,w) is $(b(v,w), c(v,w))$. The label at node x is the value of the variables $[B(x), d(x), p(x)]$ at the completion of phase 1. The solid lines are the *tree* links and the dashed lines indicate the *non-tree* links. Figure 3 depicts the transformation of link metrics in Phase 2. The label of each link (v,w) is $(b'(v,w), c'(v,w))$.

In Figure 4, the label of each link (v,w) is $(b'(v,w), c'(v,w))$. At the completion of phase 3 of MADSWIP executed with source node s, the

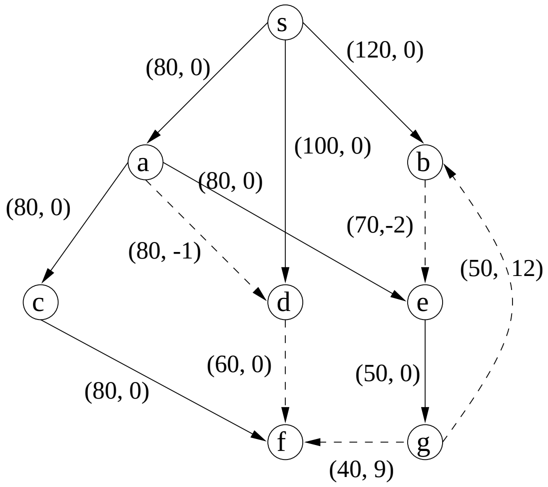

Fig. 3. At the completion of Phase 2.

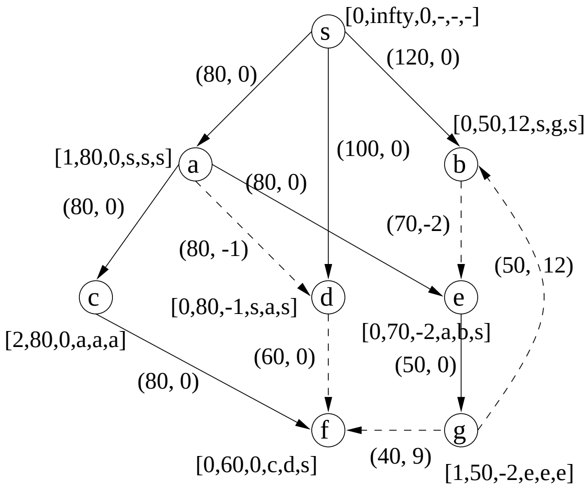

Fig. 4. At the completion of Phase 3.

label at node x is the value of the variables $[G(x), B'(x), d'(x), p(x), r(x), q(x)]$. Note that the parent $p(\cdot)$ was computed in phase 1. Execution proceeds as follows: s labeled with non-tree links $(a, d), (b, e), (d, f), (g, b)$ processed and tree link (s, a) processed with inequality (4) satisfied; d labeled; e labeled with non-tree link (g, f) processed and tree link (e, g) processed with inequality (4) satisfied; f labeled; b labeled; a labeled with tree link (a, c) processed with inequality (4) satisfied; g labeled; and, finally, c is labeled. Note that the value of the variables $p(\cdot)$, $q(\cdot)$, and $r(\cdot)$ enable one to compute the maximally disjoint paths from source s to *all* destination nodes.

MADSWIP computes *maximally* disjoint paths in the case where completely disjoint paths do not exist. In Figure 4, it can be easily verified that the two paths computed from s to g in phase 4 are (s, b, e, g) and (s, a, e, g) with the common link (e, g).

III. Approach to Performance Evaluation

A. Simulation Tool

In order to evaluate the performance of our algorithm, we developed a call-level simulation tool that has the following features.

Two Topology Generation Algorithms. We developed two algorithms for randomly generating network topologies. The first algorithm is used to generate commercial style topologies which consist of a dense core surrounded by a set of edge switches, each of which connects to the core by at most two or three links. In this class of topologies, called *core-edge* topologies, hosts are only attached to edge switches and not to core switches. The second algorithm generates purely random topologies in which each switch has one host attached to it.

Both of the methods place nodes within a unit square of length 5000 km. The topology generation algorithm randomly places the core switches within a square of length CORE_LEN < 1 at the center of the unit square. The edge switches are randomly placed within the remaining area of the unit square. Any two core switches are then connected with probability C2C_PROB. Each edge switch is connected to the nearest 2 core switches.

The algorithm for generating purely random topologies initially place the nodes randomly in the X-Y plane. It uses the parameters DIST and LINK_PROB, and places a link between two nodes (with probability LINK_PROB) if the distance between them is less than DIST. The parameter MAX_NUM_LINKS bounds the number of links, and the MAX_DEGREE parameter specifies the degree of each switch node.

Dissemination of QoS Status Update Messages. Our simulation tool simulates the dissemination of link-state update messages in the style of the PNNI protocol. Each node is responsible for originating topology updates regarding its outgoing links. In our simulator, update message incur a delay of 50 ms per hop plus the propagation delay. Update messages are sent whenever (i) T_REFRESH seconds have elapsed since the last time an update message was sent for the link, or (ii) the available bandwidth either differs from that advertised in the previous message by more than 10%, or falls below a threshold of 1% of the link bandwidth.

Estimation of Call Setup Time. We estimate the average call setup time where the average is taken over all calls that are successfully established. The call setup time of a call consists of (i) the time to do a local check to see whether the QoS of a precomputed paths matches the user's QoS request; (ii) the time for a VC setup procedure (which includes setup messages traveling across links and nodes along the proposed path); and (iii) any crankback that may occur. The crankback message incurs link and hop transit delays.

In our simulator calls arrive according to a Poisson process. We ran our simulations long enough to process

250,000 calls. The delay advertisements by a node reflect the propagation delay. A node advertizes a fixed jitter than is randomly selected uniformly from 1, 2 or 3 ms. For per-node QoS guarantees, we advertise fixed delay and jitter values since this is the approach adopted by switch vendors today. The advertised bandwidth is the link capacity minus the currently allocated bandwidth.

B. Topologies

The topologies that we generated and used in our simulations are described in Table I. In the commercial style topologies we assumed the core-to-core links have OC-12 bandwidth (1.2 Gbps) and the edge-to-core links have two OC-3's of bandwidth (322 Mbps). All links in the random topology have 322 Mbps. All five topologies had 25 nodes.

C. Applications

In Table II we state the QoS requirements of the four multimedia applications we used to generate call requests in our simulator. The duration for the first three types of calls was chosen randomly from a uniform distribution over the range 1 to 6 minutes.

D. Alternate Algorithm

In order to compare our algorithm for precomputed paths with another algorithm, we implemented a method which we call D6. We chose a method that a switch vendor is likely to implement because of its simplicity, and because it is based on an already known and well understood algorithm. This method uses 6 precomputed paths. To compute these paths a standard shortest-path algorithm is invoked 6 times, each time with a different objective function. We used Dijkstra's algorithm in our implementation. The six paths chosen are (1) the minimum delay path; (2) the minimum hop path; (3) the minimum jitter path; (4) the minimum distance (measured in km) path; (5) the maximum bandwidth path; and (6) the minimum cost path, where the link cost = 0.5*(delay) + 0.5*(jitter).

This algorithm differs from MADSWIP in its approach to the selection of the types of paths to compute. The D6 method explicitly addresses the multiple QoS metrics that may exist in a network. In other words, it computes one path for each of the metrics. * We thus call this method the multi-Dijkstra explicit-QoS method. The only QoS metrics that MADSWIP explicitly addresses are bandwidth and delay. MADSWIP chooses to compute the remaining paths using an approach based on maximally disjoint paths.

The D6 method provides *multiple* paths, but not *alternate* paths in the sense that no two paths are guaranteed to have some property of distinction (e.g., of having distinct links or nodes). In D6 the paths are computed independently and hence have the potential to be largely, or even exactly, the same. In fact, nothing is known about the relationships among the precomputed paths because they are unrelated to one another. A pair of alternate paths generated by MADSWIP are related (by definition) in that they are known to share a minimum number of links. In addition to trying to minimic a practical vendor-style solution, another reason for selecting this second method is to compare the two approaches of alternate paths and explicit-QoS.

IV. Algorithm Comparison

We use the notation M6 to denote the MADSWIP algorithm with six precomputed paths as specified above. In Figure 5 we provide the blocking rate for the three commercial style topologies. The performance of CET1 and CET2 do not differ much; whereas the performance of both algorithms in CET3 is noticeably larger than in the other topologies. (We will return to a discussion comparing topologies later in Section VI.) The blocking rates are insignificant for both algorithms until the load reaches approximately 7 calls/s. At low loading levels there is enough bandwidth to accept all calls; hence we would expect the blocking rate to be zero (or very near zero) and the difference between algorithms to be negligible. For load levels above 7 calls/s, the blocking rate grows quickly. We see that in all three topologies, M6 outperforms D6. At higher loading levels the actual value of the blocking rate achieved by M6 is 0.5 - 2.0 percentage points lower than that achieved by the D6 method. Measured in terms of a percentage improvement MADSWIP provides anywhere from a 20-90% improvement over D6 (depending upon the load). Note that each percentage point improvement is significant for carriers and ISPs since it could lead to the acceptance of tens of thousands of additional calls within a single day.

The M6 method outperforms the D6 method in both of our random topologies (Figure 6) as well. Here again we see a region of low loading (less than 5 calls/s) in which blocking rates are insignificant and both methods exhibit the same behavior, and a region (larger than 8 calls/s) in which the blocking rates grow very quickly. In the random topologies the gain of M6 over D6 is a bit lower than in commercial style topologies. M6 achieves an actual blocking rate about 0.5 - 1.0 percentage points lower than D6, or about a 20-50% improvement over D6.

In all 5 topologies the difference in performance between M6 and D6 grows as the load grows, i.e., the benefit of MADSWIP increases as the load increases. Since the M6 method outperformed the D6 method is all 5 topologies, we conclude that the M6 method makes a better choice of paths to compute than the D6 method. This demonstrates that *it is preferrable to use minimally overlaping paths rather than paths that explicitly account for all the individual QoS metrics in a network.*

We now examine the call setup times of these algorithms. Since we anticipate that the M6 method will yield more paths that are likely candidates to route a call (i.e., will

*This is true for all paths except the 6th path. This last path could have focused on the loss metric, but since we do not implement that in our simulation tool, we chose a simple linear combination of two other metrics. A sixth path was needed for a fair comparison since MADSWIP computed six paths.

Name	Number of Links	Diameter (hops)	Core Nodes	C2C_PROB	MAX DEGREE
Core-Edge Topology 1 (CET1)	56	3	10	0.6	N.A.
Core-Edge Topology 2 (CET2)	49	4	10	0.4	N.A.
Core-Edge Topology 3 (CET3)	43	5	10	0.25	N.A.
Random Topology 1 (RT1)	75	4	N.A.	N.A.	7
Random Topology 2 (RT2)	64	5	N.A.	N.A.	10

TABLE I
Topologies Used in Simulations

Application	Bandwidth	Delay	Jitter	Duration (min)	Fraction of Calls (%)
Audiophone	64 Kbps	120 ms	10 ms	*uniform*(1,6)	30
Videophone	2 Mbps	100 ms	13 ms	*uniform*(1,6)	30
Videoconference	5 Mbps	90 ms	12 ms	*uniform*(1,6)	20
MPEG-2 TV Quality	8 Mbps	1 ms	18 ms	6	20

TABLE II
QoS Requirements of Multimedia Calls Used in Simulator

pass the local check), we also expect that M6 would attempt a call setup on a larger number of paths than D6 would. This means that the call setup time for M6 will be larger than D6. A tradeoff thus arises because M6 achieves a lower blocking rate at the cost of a higher call setup time. The higher call setup time is a direct result of the better selection of the path types for the six precomputed paths.

Figure 7 shows that this is indeed the case in our three commercial style topologies. In CET1, there is no difference in call setup times of the two methods (even at higher loading). Recall that there was a difference in blocking rate for these two methods at higher loading. Hence in the topology with shortest diameter, there is no tradeoff to make between the two algorithms: M6 achieves better blocking performance than D6, while the call setup times are the same. In fact, in the commercial style topologies, we only see a significant difference in call setup times in CET3 and the random topologies (Figure 8) at higher load levels. The call setup times for M6 are larger than D6 because M6 spends more time in call establishment and crankback. It does so precisely because a larger fraction of its precomputed paths are selected as good candidates to route the call. The difference in the number of good candidate paths produced becomes more acute as the diameter of the network increases, thus making it more difficult to satisfy delay and jitter requirements.

V. Multi-Route Selection Policies

Once paths are precomputed, they must be ordered. This ordering determines the sequence in which paths are tried when a call request arrives. The first path found, that meets the user's QoS requirements, is used to route the call. Path ordering can be done off-line, so as not to affect the call seutp time of the requester. Note that a policy for path ordering is the same as a policy for selecting among multiple paths since the ordering determines the prioritization.

One policy for path ordering, known as call packing, orders them, from the path with the smallest amount available bandwidth to the path with the largest amount. This policy routes new calls on heavily loaded rather than lightly loaded links; it also favors OC-3 over OC-12 links. This type of policy can be useful, because it avoids the fragmentation effect, which occurs when many links have small amounts of bandwidth left, and no single link (or path) with a large amount of bandwidth remains. When fragmentation occurs, high-bandwidth calls are blocked. Call packing tries to arrange the calls so that some links are left with large enough amounts of available bandwidth to handle those high-bandwidth calls. Hence, the advantage of call packing is that it favors high-bandwidth calls; but its disadvantage is that it fills up some links completely, thus reducing the connectivity of the network. Call packing has proven to be useful in telephone networks.

In contrast to call balancing, the load balancing policy tries to spread the load evenly among the links. This policy chooses to route new calls on lightly loaded paths rather than on heavily loaded ones. We implement such a policy by ordering our precomputed paths from those with the largest available bandwidths to those with the smallest available bandwidth. A third policy, called the min-hop policy, will route a call on the minimum hop path that meets the QoS requirements. This type of policy has traditionally been useful in networks because routing calls on

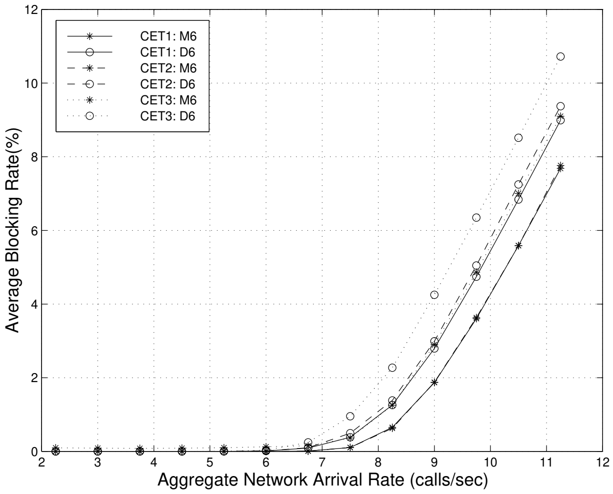

Fig. 5. Blocking Rate for Core-Edge Topologies

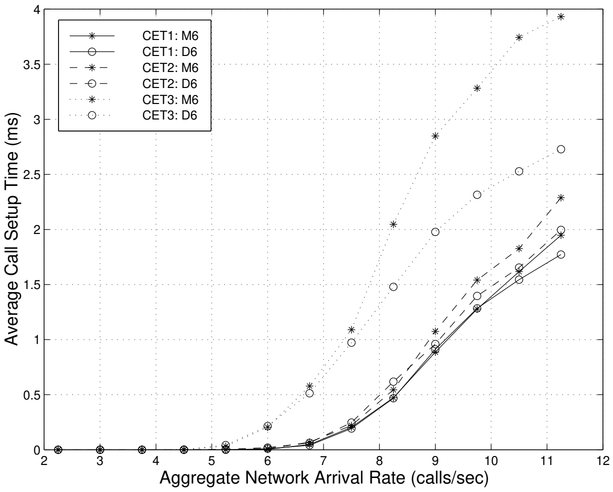

Fig. 7. Call Setup Time for Core-Edge Topologies

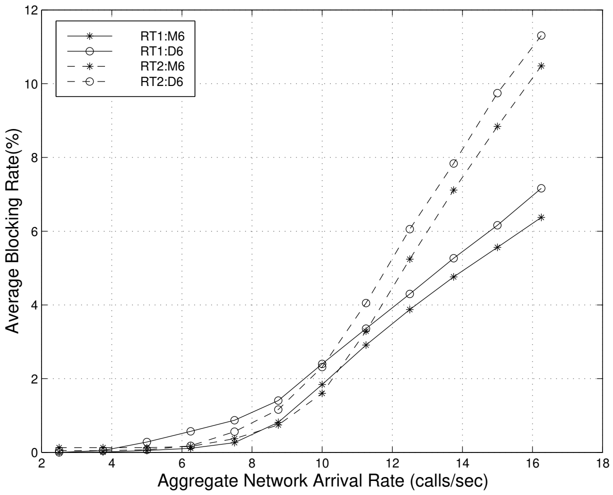

Fig. 6. Blocking Rate for Random Topologies

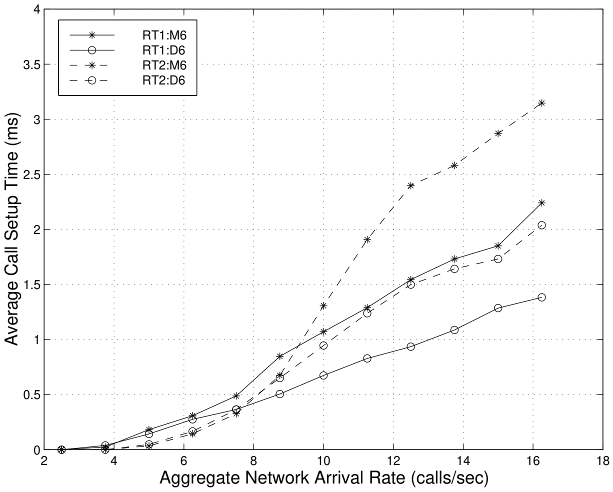

Fig. 8. Call Setup Time for Random Topologies

the fewest links possible uses less overall network resources, which in turn leaves more resources available to other users.

The performance of these policies for M6 is given in Figure 9 for commercial style topologies and Figure 10 for random topologies. The load balancing policy is the best performing policy in all topologies, and the call packing policy is the worst in all topologies. In most cases, the difference between the load balancing and minimum hop policies is very small. The relative performance of call packing to load balancing is worse in sparsely connected networks (e.g., CET3), as opposed to densely connected networks (e.g., CET1). This is reasonable since the disadvantage of call packing is that it routes incoming calls on heavily loaded links, thus causing some links to be completely filled up. When this happens, the connectivity of the network is reduced and future calls are routed on longer hop paths. This increases the difficulty of satisfying the delay and jitter requirements, and thus increases the blocking.

These graphs also demonstrate that the difference between the performance of call packing and load balancing is larger in CET3 than in CET1 (similarly for RT2 and RT1). In other words, the benefit of load balancing increase as the density of a topology decreases.

VI. The Influence of Topology

In this section we study the effect of changing the network's density on performance. When the density of a network decreases then it may no longer be possible for some pairs of hosts to communicate over short paths (e.g., 3 hops), and hence a decrease in density is typically accompanied by an increase in network diameter. As the diameter of the network increases, it will contain longer paths, which will result in an increased blocking rate. This is because longer paths have larger delay and jitter (since these metrics are additive), and will therefore be less likely to satisfy the QoS contraints for a given call. This is illustrated in Figures 5 and 6, which show that the blocking rate increases as the topology density decreases, for a given load level. Figures 7 and 8 demonstrate that the call setup time increases as the topology density decreases.

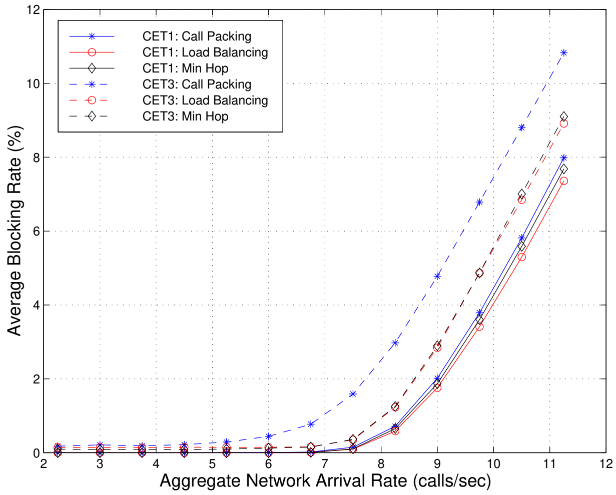

Fig. 9. Route Selection Policies for Core-Edge Topologies

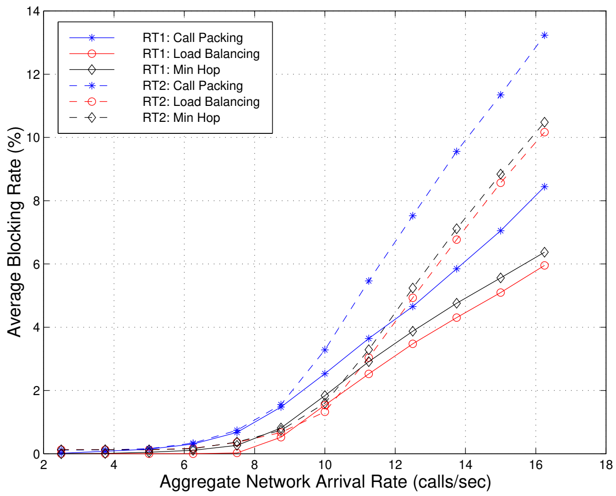

Fig. 10. Route Selection Policies for Random Topologies

We now discuss the combined effect of network diameter and jitter constraints on the blocking rate. One component of the overall blocking rate comes from the blocking that occurs due to an inability to satisfy jitter requirements. Let J_R, with $j \in J_R$, denote the set of end-to-end jitter requirements. For our applications, $J_R = \{10, 12, 13, 18\}$. Recall that each switch advertises 1, 2, or 3 ms (with equal probability) as its guaranteed jitter for an individual node. If all of our calls can be routed on 3-hop paths, then the blocking due to jitter will be zero because, in the worst, case a 3-hop path can have a jitter of 9 ms. Hence we expect the blocking due to jitter in a network of diameter 3 to be very small. The jitter component of the blocking rate is not necessarily zero in this case for the following reason. A network of diameter 3 means that there exists (according to an unloaded topology) at least one path of length 3 for each source-destination pair. However, once links get loaded with calls, then a particular call may no longer have the option to be routed on a 3-hop path, but instead may have the option of a 4-hop path. On a 4-hop path, jitter blocking can occur.

In a network of diameter 4, the audio call is the only application that has a positive probability of blocking due to jitter on paths of length four. In a network of diameter 5, the audio, videophone and video-conference applications have positive probability of blocking due to jitter on paths of length five. Our commercial-style topologies become increasingly more difficult to route in as the diameter increases because of the jitter requirements (and the delay requirements as well).

Little performance difference is apparent between CET1 and CET2. However CET3 appears to be substantially more difficult to route in than either of the other two commercial style topologies. (See Figures 5 and 6.) We thus observe a threshold effect in commercial-style topologies. Increasing the network diameter from 3 to 4 did not lead to significant performance changes, however, increasing the diameter from 4 to 5 did lead to significant performance changes.

VII. Conclusions

We compared two approaches to precomputed paths in which six paths are maintained. Our approach (M6) focuses mainly on finding maximally link-disjoint paths, whereas the other approach (D6) chooses to find one path for each of the many QoS metrics that may exist in a network. We found that it is not unusual for M6 to cut in half the blocking rate achieved by the D6 method. In fact, M6 outperformed D6 in all five of our topologies. In commercial style topologies M6 provides anywhere from a 20 - 90% improvement, depending upon the load; while in random topologies M6 provides anywhere from a 20 - 50% improvement. We also saw that the benefit of a MADSWIP relative to the other method increases (1) as the load increases, and (2) as the topology becomes more "difficult". All of this demonstrates that it is preferrable to use minimally overlapping paths rather than paths that explicitly account for each type QoS metric in a network.

At low loads and in the densest topology, the call setup times of the two methods were the same. At high load levels and in topologies whose diameter is large enough to make satisfying delay and jitter nontrivial, the M6 method experiences larger call setup times than D6. The better blocking rate of M6 is achieved at the expense of additional call setup time. However the call setup time is longer exactly because the M6 method produces a better set of candidate paths than D6. A larger number of candidate paths are tried under the M6 approach. It seems preferrable for a call to wait a bit longer and get routed then for it to be rejected quickly.

Load balancing is better than call packing for QoS routing, since it outperformed call packing in all of our topologies. Call packing reduces the connectivity of the network, which pushes calls to be routed on longer paths. On longer paths it is more difficult to satisfy delay and jitter requirements. We showed that the benefit of load balancing over call packing increases as the density of the network decreases.

We see threshold effects when the density of a topology is reduced and the diameter increases. If the increase in diameter doesn't change the difficulty of satisfying delay and jitter, then little effect on the blocking rate and call setup time is seen. However, as certain threshold points, decreasing the density can suddenly render the network much more difficult to satisfy delay and jitter. At these points, increasing the topology diameter caused 20-30% performance degradation in the scenarios we considered.

References

[1] E. Crawley, R. Nair, B. Rajagopalan, and H. Sandick. A Framework for QoS-based Routing in the Internet. *draft-ietf-qosr-framework-05.txt*, May 1998.

[2] ATM Forum. PNNI Specification Version 1.0. *af-pnni-0055.000*, March 1996.

[3] R. Guerin and A. Orda. QoS-based Routing in Networks with Inaccurate Information: Theory and Algorithms. *Proceedings of IEEE Infocom*, March 1997.

[4] Q. Ma and P. Steenkiste. Quality-of-Service Routing for Traffic with Performance Guarantees. *Proceedings of IFIP Fifth International Workshop on Quality of Service*, May 1997.

[5] R. Ogier, B. Bellur, and N. Taft-Plotkin. An Efficient Algorithm for Computing Shortest and Widest Maximally Disjoint Paths. *SRI International Technical Report ITAD-1616-TR-170*, November 1998.

[6] Ariel Orda. Routing with End to End QoS Guarantees in Broadband Networks. *Proceedings of IEEE Infocom*, April 1998.

[7] J.W. Suurballe and R.E. Tarjan. A Quick Method for Finding Shortest Pairs of Disjoint Paths. *Networks*, 14, 1984.

[8] Z. Wang and J. Crowcroft. Quality-of-Service Routing for Supporting Multimedia Applications. *IEEE Journal on Selected Areas in Comm.*, September 1996.