Distributed QoS Routing Using Bounded Flooding

Seok-Kyu Kweon and Kang G. Shin

Real-Time Computing Laboratory
Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, Michigan 48109–2122.
email: {skkweon,kgshin}@eecs.umich.edu

ABSTRACT

Quality-of-Service (QoS) routing is indispensable to real-time applications in integrated services packet-switching networks. However, existing QoS-routing schemes based on either request flooding or link-state dissemination are inefficient in terms of message overhead and/or operational cost. This paper proposes a QoS-routing algorithm with bounded flooding which not only dramatically reduces message overhead and operational cost, but also provides good performance. The overhead is reduced by limiting the search area only within which connection-request messages are flooded. Since our scheme is based on flooding, it does not require path calculation nor link-state dissemination, which can be very expensive. In order to limit the search area, we employ a distance table at each node which stores hop counts of the minimum-hop paths to all the nodes reachable via its outgoing links. Unlike link-state routing, the proposed scheme only needs network-topology information that changes much less frequently than link-load states. Extensive simulation results have demonstrated the effectiveness of the proposed scheme in terms of performance, operational cost and message overhead. Especially, it is shown to be much better than all the other existing QoS-routing schemes in terms of message overhead and operational cost, while providing reasonably good performance.

1 Introduction

Providing QoS guarantees to individual connections is one of the most important research subjects in the area of integrated services packet-switching networks (ISPNs). Compared to traditional data communication applications such as ftp and telnet, most multimedia applications require more precisely-defined QoS in terms of packet delay, throughput and loss rate. For such multimedia applications, ISPNs are required to support real-time communication or guaranteed service in addition to best-effort service for conventional data communication applications. There are, however, many technical issues to be resolved before making real-time communication service available. To this end, several new packet-scheduling algorithms or service disciplines, admission-control and resource-reservation protocols have been proposed [1–10]. The main idea behind these efforts is that an appropriate service discipline must be employed in order to deliver user-requested QoS, and that admission control and resource reservation must be done in order for the service discipline to work as intended. In conjunction with resource reservation and admission control, finding a route which can provide user-requested QoS is another important issue. Without an efficient QoS-routing algorithm, a network may fail to find a route and reject the request for a new connection, even if there exists a qualified route with enough resources to honor the request. QoS routing must therefore be considered as a component of *Integrated Service Model* in the ISPN [11].

In addition to finding a qualified route for each requested connection, QoS-routing must be able to make good overall network utilization. This also relates to increasing the chance of accepting future requests by making a good route selection for the current request. Finally, QoS-routing algorithms should not be expensive in terms of both operational cost and implementation complexity.

There are basically two approaches to QoS routing: source-directed and flooding-based. In the source-directed approach, the source switch or router selects a path based on connection traffic parameters and available resources in the network. This approach can be refined into static or dynamic routing, depending on whether link-load information is used for path computation or not. The former uses static topology information in choosing a route. Although static QoS routing incurs low operational cost, it suffers from a poor connection-success rate. In contrast, in dynamic QoS routing, the information on the available resource of each link must be distributed throughout the network, so that any source can have access to the correct information on resources available in the network. The information distributed is often called *link-state*, and thus, source-directed routing is also called *link-state routing*. By applying either Dijkstra's shortest-path algorithm or Bellman-Ford shortest-path algorithm based on link-state information, one can find a qualified route for the requested connection. The ATM Forum's PNNI standard [12] defines a routing protocol for distributing link-state throughout the network, and a signaling protocol for processing and forwarding connection requests from the source. A QoS extension to the OSPF protocol was proposed for the Internet, including an *explicit routing* mechanism for source-directed IP routing [13, 14]. Because of its high operational overhead in distributing and maintaining link-state information, source-directed routing may not scale well. To improve the scalability in large networks, a hierarchical approach can be adopted to distribute and manage link-state information [12]. In addition to use of a hierarchical approach, efforts have been made to reduce link-state messages in order to control the overhead. Periodic or triggered distribution of link-state information is a typical example of this effort. Although less frequent dissemination of link-state information reduces the overhead, stale information might cause undue routing and/or signaling failures. A *routing failure* is said to occur if the source cannot find a route based on link-state information kept in its own database even if a qualified route exists. And a *signaling failure* occurs if one of intermediate nodes on the route determined by the source cannot reserve the resources required for the requested connection's QoS.

In flooding-based QoS routing [15, 16], local nodes are not required to keep link-state information for the entire network. The source node simply multicasts each request to its neighbors, which then relay the request to their neighbors, and so on, until the request reaches the destination. In order to limit the number of request messages, the algorithm does not flood through a link which is found unable to guarantee the connection's QoS. Although this approach incurs considerable operational overhead due to the large number of request messages, it still has its own merits as follows. First, there is no need for link-state information distribution and shortest-path calculation, thus reducing operational overhead and implementation complexity. Second, nodes are not required to keep the database of link-state information, saving storage space. Finally, since the latest, thus the most accurate, information kept for each local link is used to determine whether it can accommodate a new connection or not, the algorithm can always find a qualified route, if any, thereby outperforming link-state routing in terms of connection-success rate.

In this paper, we propose a variation of flooding-based QoS routing which incurs much lower message overhead yet yields a good connection-success rate, compared to the existing flooding algorithms. In order to reduce the flooding overhead, request messages are allowed to take only those routes of hop count smaller than a pre-specified limit. The hop count limit is chosen such that as many alternate routes as possible are searched while keeping overhead below a given level. Nodes are therefore required to keep the network-topology information at each node which is not link-state but node connectivity. Since node connectivity changes (due to loss and addition of nodes/links) much less frequently than that of dynamic link-state, distribution of topology changes incurs little overhead compared to link-state distribution. Using this 'almost static' topology information, network nodes decide whether to relay connection-request messages to their neighbors. That is, if the route via one of its neighbors has a larger hop-count than the pre-specified limit, the node does not relay the request message to that neighbor. By narrowing the flooding area using this type of "pruning," the proposed algorithm is shown to be able to reduce the message overhead dramatically. Moreover, unlike the source-directed approach, it does not require on-demand (or off-line) shortest-path calculation, thus lowering the operational cost. Note, however, that this reduced message overhead and operational cost may be achieved at the expense of loss in connection-success rate; fortunately, this loss is shown to be acceptably low. In general, there is a tradeoff between reduction of overhead and operational cost, and loss in connection-success rate, in both flooding-based and link-state schemes. That is, by either increasing the frequency of link-state distribution or increasing request messages, one can achieve a higher connection-success rate. Although it is impossible to derive an analytical relation between overhead reduction and performance loss, our scheme can control overhead by adjusting the hop count limit of candidate routes subject to the required connection-success rate.

The rest of the paper is organized as follows. After providing some background in Section 2, we describe the proposed QoS routing in Section 3. Using extensive simulations, the proposed and other existing algorithms are comparatively evaluated in Section 4. The paper concludes with Section 5.

2 Work Related to QoS Routing

Most service disciplines for timeliness-QoS guarantees assume connection-oriented services due to their conceptual ease in resource reservation and management. The goal of QoS routing is then to find a route or a virtual circuit through which real-time data will be transported. This is the fundamental difference between datagram routing and QoS routing. In datagram networks, including the current Internet, each datagram is routed in a connectionless manner.

Since real-time communication services are provided by employing a special form of service discipline at each switch or router, the metric which QoS routing must consider for its route-selection strongly depends on the service discipline employed. The metric could be delay, loss, bandwidth, jitter, or a combination of thereof. In [17], an optimal routing scheme for multiple QoS parameters was considered. Although QoS requirements and network resources may be characterized by multiple parameters, under most of Weighted-Fair-Queueing-type service disciplines, one can satisfy user-requested QoS requirements by providing guaranteed throughput or bandwidth to each connection. In some service disciplines such as delay-EDD (Earliest-Due-Date) and RCSP (Rate-Controlled Static-Priority), guaranteeable delay and reserved bandwidth are indirectly related. In this case, reserving bandwidth is not enough for guaranteeing a delay bound. Since our approach can be easily modified to handle these sophisticated service disciplines, we will express a connection's performance requirements with a single parameter, bw, that represents a peak, or average, or effective bandwidth, depending on the underlying admission-control policy. The network service provider must reserve resources commensurate with bw along the path chosen by QoS routing in order to provide the QoS promised to the end user. For link-state routing, a node's link-state database stores information about the utilization of each link which is defined as the sum of reserved bandwidths over the link.

There are several strategies on how to choose an optimal route for real-time communication. For example, minimum-hop routing, shortest-widest path [17], widest-shortest path [14], shortest-distance path [18], and minimum-load routing [19] are among them. Non-minimal routing algorithms, such as shortest-widest path, often select circuitous routes that "occupy" more network resources, thus possibly causing rejection of future connection requests. The proposed routing scheme favors the selection of a minimum-load route to balance network utilization, but it can be easily modified to accommodate other routing strategies.

3 QoS Routing with Bounded Flooding

Our scheme is designed for an arbitrary point-to-point network; it can be a distributed system or a WAN. All links in the network are assumed to be bidirectional.[1]

3.1 Overview of the Proposed Approach

We employ a bounded, not brute-force, flooding; in order to reduce the overhead resulting from flooded request messages, we limit the hop count each request message can take before reaching its destination. That is, when an intermediate node receives a request message, it will decide whether to forward the message to one of its neighbors by checking if the minimum-hop path via that neighbor can lead the request message to the destination within the source-defined hop count limit. This approach can be interpreted as flooding with a limited search area. Hop count is selected as a basis because restricting the hop count of a path limits the degree to which the scheme can find a circuitous route around congested links, thus not blocking future connection requests, as argued in [20]. In order to determine whether a request message can reach the destination within the hop count limit via a particular outgoing link, each intermediate node uses its 'almost static' network-topology information. Instead of calculating the minimum-hop path via an outgoing link to the destination every time a request message arrives, we use a (distance) table storing the hop count of the minimum-hop path through each outgoing link to every other node.

3.2 Composing Distance Tables

Before starting the QoS-routing service and/or when the network topology changes due to failures and additions of nodes and links, every node must compose or update its distance tables, using new topology information. Note that the topology information is almost static, because it does not contain any link-state information but describes only the connectivity between nodes. By calculating inter-node distances off-line, our scheme dramatically reduces run-time operational cost.

Distance from node i to node j via node i's outgoing link ℓ is defined as the hop count of the minimum-hop route among all the routes from node i to j via link ℓ. In order to prevent the minimum-hop route from traversing ℓ in the reverse direction, we exclude this "reverse-directional link"[2] from minimum-hop route calculation. Its purpose is to exclude the case when request messages bounce back to nodes from which they were sent. This prevents request messages from oscillating between nodes. The minimum-hop route can be easily calculated using Dijkstra's shortest-path algorithm or Bellman-Ford shortest-path algorithm. All links are given the same weight except the reverse-directional link of ℓ whose weight is set to ∞.

Although we employed hop count in composing distance tables, other metrics can be used as distance, depending on the environment. For example, if the network is not homogeneous, i.e., each link has different capacity, the "cost" of a path can be employed as distance.

[1] This assumption can be relaxed trivially.

[2] Actually, there is only one bidirectional link, but it is viewed as consisting of two unidirectional links.

3.3 The Proposed QoS-Routing Algorithm

Upon generation of a connection request, the source node must check the "search-scope" of the connection. The search-scope is the maximum hop count that the request message can take to reach its destination. In order to increase the chance of granting the requested connection, multiple alternate, not necessarily disjoint, paths must be given an opportunity to run the connection over them. The search-scope must therefore be determined by making a tradeoff between the message overhead and the request-acceptance probability. In this paper, we allow at least two alternate paths to be tried for each connection request. Thus, the search-scope is given by the second smallest distance between each source-destination pair. If more than two minimum-hop paths exist for a request, we consider the hop count of the minimum-hop paths as the second smallest distance. For example, in Figure 1, the distance $d(A, B; a)$ between source A and destination B via a is 2, and the distance via b, $d(A, B; b)$, is also 2. So, the second smallest distance is 2 and thus, the search-scope of a connection between A and B is set to 2. If A is the source and C the destination, then $d(A, C; a) = 2, d(A, C; b) = 4$, and $d(A, C; d) = 4$. The second smallest distance is 4, and thus, the search-scope is 4. In general, the search-scope can be given as the hop count of the n^{th} minimum-hop route for a given pair of source and destination.

Since we are considering only the connectivity between nodes, the search-scope for every pair of source and destination can be calculated *a priori* by source nodes before servicing connection requests. The pre-calculated search-scopes are stored in a table at each node. Whenever the topology changes, albeit very infrequently, these search-scopes must be recalculated. On the other hand, the search-scope can be determined on-line for a given connection request. In this case, the source node may use the current load condition which is estimated by using variables like local link bandwidth usage and the measured request-acceptance probability. Here we follow the first approach to minimize the operational cost at the expense of a slight performance loss. The second approach will be considered in our simulation study.

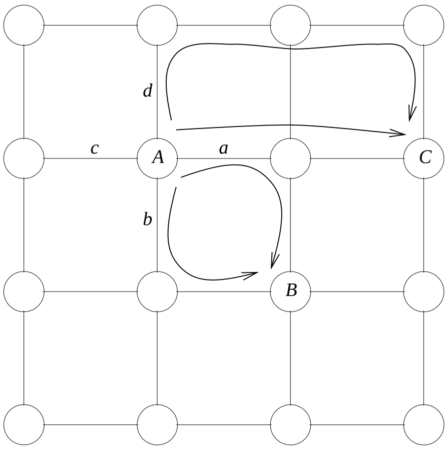


Figure 1: An example of determining the search scope

Upon receiving a connection request from an application program, the source node generates a request message, m. A connection request message contains the following fields.

- Connection identifier $Req.ID$ which uniquely identifies the corresponding real-time connection. For the uniqueness of each connection ID, an identifier is composed of two parts: the node ID (or address) and connection number (unique within a source). This composition of connection IDs ensures their uniqueness throughout the network.
- Source identifier $Req.src$ of the requested connection.

- Destination identifier $Req.dest$ of the connection.
- Timeout. This field is used to specify the request's time to live (TTL). After its TTL, a request message is no longer valid and thus discarded. How to determine this value will be discussed when the connection-confirmation process is described later in this section.
- Search-scope Sc of the requested connection.
- Hop count H of the path taken by the request message to the current node.
- The connection's bandwidth requirement $bw(\cdot)$. If no outgoing link has available bandwidth larger than this, the request will be discarded.
- List of intermediate node IDs that the message has traversed thus far. Every time the request message is relayed to the next node, the new node ID is appended to this field. This information is needed for the destination node to confirm the establishment of the requested connection.
- Accumulated utilization u of the path. This is the sum of all the intermediate links' loads, where load is defined as the bandwidth reserved for real-time connections. This field is needed for intermediate nodes and the destination to be able to choose the least-loaded path. As discussed earlier, our scheme favors selection of the minimum-load route based on this field. For other selection strategies, this field must be filled in with an appropriate parameter. For instance, the field may store the minimum of loads of the intermediate links the request message has traversed if the strategy is the shortest-widest path.

Since the information of existing connections is necessary for a new connection's admission test as well as for the run-time scheduling of messages belonging to those connections already established, each node has to maintain two sets of tables for existing connections and pending connections. The first set is the tables of established connections (TECs), one for each of its outgoing links. Each entry of a TEC represents a real-time connection which goes through the corresponding link and consists of the following two data fields.

- Connection identifier: this is the same as the one in the connection request message.
- The connection's bandwidth requirement.

Using the bandwidth requirement of a connection, the service policy determines the priority of data messages belonging to this connection.

The second set of tables each node has to maintain are "temporary" tables for pending connection requests, also one for each of its outgoing links. These tables will be referred to as "tables of pending requests" (TPRs). Each entry of a TPR represents a connection request (or a pending connection) and contains the following fields.

- Connection identifier: same as the one in the connection request message. When a connection request is conditionally-accepted (contingent upon a confirmation), it is copied from the connection ID field of the request message.
- Timeout: must be larger than that of a request message, in order to prevent deletion of a connection request from TPR before the confirmation message arrives. More on this will be discussed in the confirmation process. Upon expiration of the timer, the connection request is deleted from TPR.
- The connection's bandwidth requirement.
- Accumulated utilization u^* of the path which the most-recently accepted request message for establishing the same connection had traversed.

When a connection request is conditionally-accepted to run the connection over a link, fields of the corresponding entry of the link's TPR is copied from the corresponding fields of the message. In addition to entries for connection requests, TPR must have a field to keep track of the remaining bandwidth of the link. The remaining bandwidth is calculated by subtracting the sum of bandwidth requirements of both the established real-time connections and conditionally-accepted real-time connections from the link capacity, $cap(\cdot)$. This is used for a new connection's admission test.

Apart from TECs and TPRs, a node has to maintain the table of conditionally-accepted connections (TAC) if it has received a request message whose *Req.dest* is the node itself. The function of a TAC is to allow the destination to choose the best among the routes which request messages safely traversed. Each entry of this table consists of the following fields.

- Connection identifier.
- Accumulated utilization u^* of the most-recently saved path.
- List of IDs of intermediate nodes the message has traversed.
- Timeout: same as that of the request message. This field tells when to initiate the connection-confirmation process.

Upon receiving a connection request from an application program, the source node sends out a request message, m, through each of its outgoing links only if it satisfies the following two conditions:

distance test (source):

$$distance(Req.src, Req.dest, \ell) \leq search\text{-}scope(m), \qquad \text{and} \tag{3.1}$$

bandwidth test:

$$util(\ell) + bw(m) \leq cap(\ell), \tag{3.2}$$

where $util(\ell)$ is the utilization of link ℓ by real-time communication traffic (i.e., the bandwidth reserved for real-time communication class), and $bw(m)$ is the bandwidth requirement of connection m.

The flowchart in Figure 2 shows the actions to be taken by an intermediate node or by the destination upon receipt of a connection request message. First, the node checks whether *Req.dest* of the message matches its own ID, $Node.ID$. If they match, then it checks whether its TAC already has the connection ID identical to that of the request message. If yes, the node has already received at least one copy of the connection request. In this case, the node checks if the newly-arrived request message contains a better route than the one in TAC (**better route test**). By "a better route," we mean that the request message contains a smaller accumulated utilization than that of the old one in TAC. As mentioned earlier, this is to favor the minimum-load path. If the request message carries a better route, the node updates the corresponding entry of its TAC. Otherwise, the request message is discarded. If the request is new, the node stores the request in its TAC.

If the destination contained in the request message does not match, the node checks, for each outgoing link, if the ID is in the link's TPR, that is, if other request messages carrying the same connection request have already passed through the link. If yes, as done with TAC, the node checks if the new request message contains a smaller accumulated utilization than that of TPR. If yes, the node executes the following distance test.

distance test (intermediate nodes):

$$H(m) + distance(node, Req.dest, l) \leq search\text{-}scope(m), \tag{3.3}$$

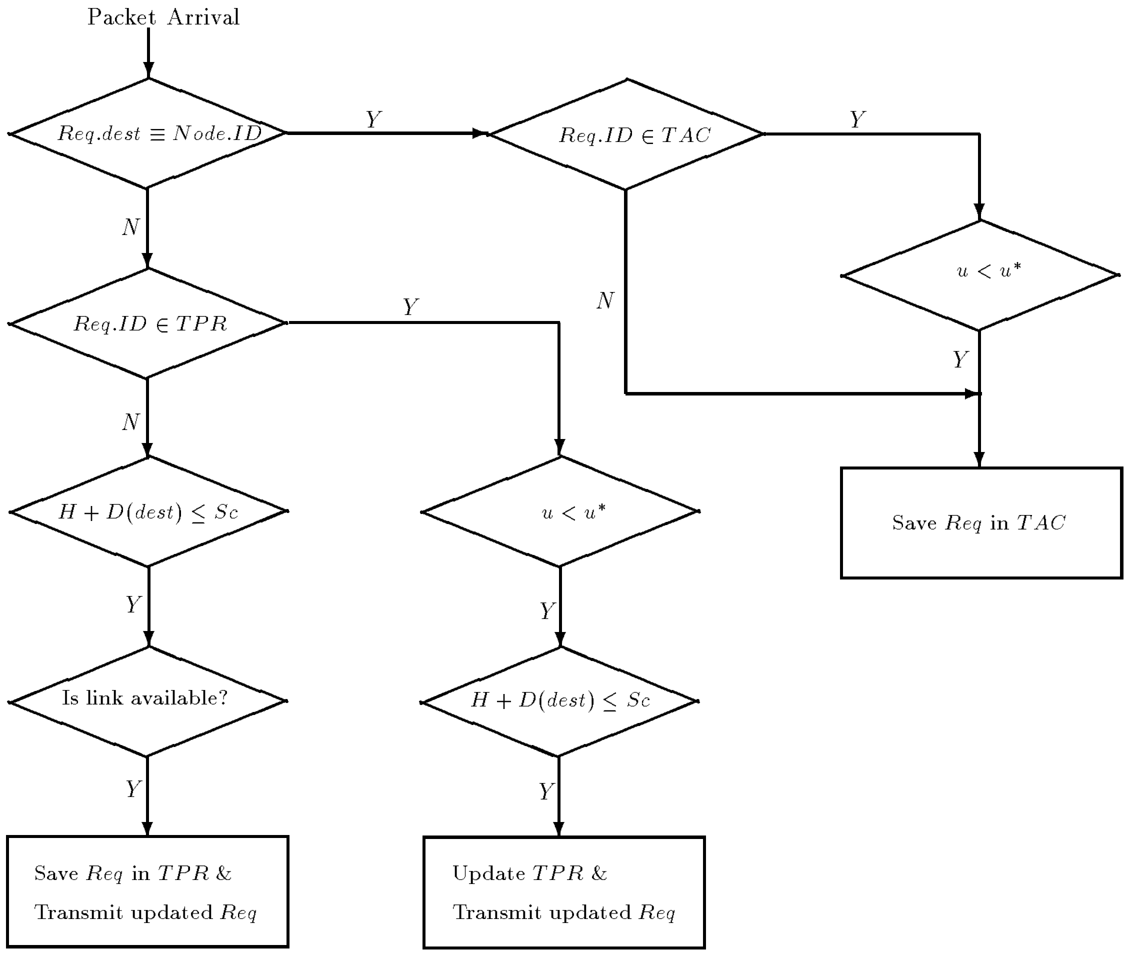


Figure 2: Node actions upon arrival of a request message.

where $H(m)$ is the hop count of the path message m has traversed so far to get to this node. If it passes this test, then the node updates and forwards the request message to a neighbor node via the outgoing link, and updates TPR by replacing u^* of the connection by the accumulated utilization of the newly-received request message. The request message is updated by adding the utilization of the outgoing link to the accumulated utilization of the message, incrementing H by 1, and appending the node ID to the list of intermediate nodes of the path the message has traversed thus far. If this test fails, the request message is discarded.

In order to reduce the overhead of request messages, we do not relay them to the connection's 'upstream' nodes. A connection's upstream nodes are the ones from which at least one copy of the same request has come. The necessary information for this action can be maintained and easily checked by recording the request message' ID in the TPR of the reverse-directional link of the link through which the request message has come. By assigning 0 to u^*, we can use a better route test for this.

If the ID of the request message matches none of IDs of connection requests stored in its TAC and TPRs, the request is new, and the distance test in Eq. (3.3) and the bandwidth test in Eq. (3.2) are conducted. If it passes both tests, the node updates and relays the request message to a neighbor through the outgoing link. In addition, it saves the connection request in the TPR of the outgoing link.

We have already discussed the destination's action upon arrival of a request message. The destination does not respond immediately to the arrival of a request message. It does not initiate the connection-confirmation process until it reaches the timeout of the request which is kept in the TAC of the destination. Before describing the confirmation process, we need to discuss timeout fields of request messages and the tables. First, in order to make our scheme work as intended, we must ensure that as many request messages as possible arrive at the destination before the timer expires and that the timer is set to as small a value as possible to reduce the chance of rejecting new requests due to unnecessary temporary reservation as seen in Flooding-based approach. For this purpose, connection request messages are transmitted through a dedicated signaling channel which can be realized by opening a permanent virtual connection. By using a dedicated signaling channel, one can reduce a request message's link delay. From the arrival pattern of real-time connection requests, we can estimate the bound of request message delays over a link. We assume that the processing delay at each node is included in the link delay. Then, the source can set the timeout field of a request message using the search-scope of the connection. Since the search-scope indicates the maximum hop count of candidate routes, the timeout must be set larger than, or equal to, the link delay bound multiplied by the search-scope. Choosing a large value as the timeout may not cause inefficient use of network resources since relaying request messages to neighbors depends on search-scopes and distances as well as the timeout value of a request message. The chosen value also works as the connection-confirmation initiation time since we set the timeout of request messages equal to that of the connection in TAC. Thus, the timeout of request messages and TAC is set to the link delay bound multiplied by the search-scope plus the processing time at the destination in order to minimize the time to set up a connection.

In addition to connection request messages, connection-confirmation messages are also transmitted through the signaling channel. Thus, we can use the same link delay bound for confirmation messages. The link delay bound enables us to calculate the maximum traversal delay of a confirmation message, which is used to determine the timeout field of TPRs. Since conditional reservation of resources at the intermediate nodes must be kept until the confirmation message reaches the source, the timeout in TPRs must be larger than the maximum traversal time of a confirmation message plus the timeout of a request message. In order to minimize unnecessary temporary reservation[3] of network resources, we select the maximum traversal time of a confirmation message plus the time-to-live of a request message as the timeout in TPRs.

Now, let's consider the connection-confirmation process. Since, as discussed earlier, the most recently-saved request message in its TAC contains the minimum-load route among the those safely reached the destination, the destination simply chooses the one in its TAC as the best route found. Although our approach is not guaranteed to find the global minimum-load route because of its bounded search, it is likely to choose the best one due to its selection process based on the better route test. Upon expiration of the timer of the connection in TAC, the destination sends a confirmation message in the reverse direction of the path recorded in the connection's entry of its TAC and deletes the entry from its TAC. Upon arrival of the confirmation message, the intermediate nodes along the route update their TECs by recording the requested connection's profile, and also delete the connection's entry from TPRs.

When a real-time communication is terminated by an application program, a disconnect process is initiated by either the source or the destination. In this process, a disconnect message is passed along the path of the real-time connection. Upon arrival of the disconnect message, the intermediate nodes delete the corresponding entry of their TECs and update the remaining link bandwidth in their TPRs.

3.4 Memory Requirement

Although our scheme is very efficient in reducing operational cost and message overhead, the efficiency is achieved at the expense of increased memory requirement. Compared to the existing flooding approach, our

[3] temporary reservation that lives longer than necessary

scheme requires additional memory for distance tables and search-scope tables. Other tables such as TPRs, TECs and TACs are also required by the existing flooding approach. When the number of nodes is N and the maximum number of outgoing links of a node is L, the number of entries needed for distance tables in the entire network is $O(N^2L)$, and, for search-scope tables, $O(N^2)$.

Link-state routing doesn't need TPRs and TACs, but requires a table similar to TECs for maintenance of link-state for each link and the run-time scheduling of real-time messages. Moreover, each node must have a table to store link-states of the entire network which must be updated more frequently than TPRs and TACs. Memory requirement for this table is the same as the one needed for distance tables in our scheme, i.e., $O(N^2L)$. If the maximum number of connection requests which are pending simultaneously in the network is N, the memory requirements for TPRs and TACs in the proposed scheme are $O(N^2L)$ and (N^2), respectively, as the number of TPRs and TACs are $O(NL)$ and $O(N)$, respectively. Since TPRs are a dominant factor among TPRs, TACs and search-scope tables in terms of memory requirement, additional memory of our scheme compared to that of link-state routing is $O(N^2L)$. If the maximum number of connection requests which are pending simultaneously in the network is much smaller than N, a dominant factor comes from search-scope tables whose memory requirement is $O(N^2)$.

4 Simulation and Discussion

We have conducted an in-depth simulation study to comparatively evaluate the proposed and other routing schemes in terms of their performance and overhead. In this study, we measured the probability of establishing connections successfully under various load conditions and network configurations. We also evaluated the overhead incurred for establishing a connection with all of the routing schemes considered.

4.1 The Simulation Model

In order to investigate the performance of the proposed scheme under different network configurations with a wide range of node connectivity, we selected the following networks: 5-ary 2-cube, 5-ary 3-cube, 10-ary 2-cube, 5-ary 3-cube, and the MBONE topology in the North America region in Figure 3. In the MBONE topology, all T3 links and nodes which are connected via T3 links are included in the simulation. Each node acts as a router or switch, and links are assumed to be bidirectional, with 'unit' capacity in each direction. A k-ary n-cube is an n-dimensional cube with k nodes in each dimension. Each node is connected to two other nodes in both the directions of each dimension in a 'wrapped-around' fashion. On the left side of Figure 4 shows a 4-ary 2-cube with the wrap links indicated by dotted lines and the right side shows a 3-ary 3-cube with the wrap links omitted for simplicity. Each edge represents two unidirectional links. The MBONE topology in Figure 3 was selected to investigate the performance of each routing scheme under a network configuration with (currently) realistic connectivity. Its connectivity is very poor as compared to the other topologies considered. (Note that future backbone networks are expected to have higher connectivity, so k-ary n-cube topologies are reasonable to consider.) Table 1 shows the characteristics of the networks chosen for evaluation.

The load is defined as the bandwidth reserved for real-time connections. For each network configuration, we tried to establish real-time connections under a certain load using the proposed scheme (labeled as BFlooding in the figures), static routing (labeled as Static), QoS routing by flooding (labeled as Flooding) [15,16], link-state routing with periodic link-state distribution with periods of 10 and 100 times the connection inter-arrival time (labeled as LS(10) and LS(100), respectively), and lastly, link-state routing with triggered link-state distribution with trigger levels of 0.1 and 0.5 (labeled as Trigger(0.1) and Trigger(0.5), respectively). In

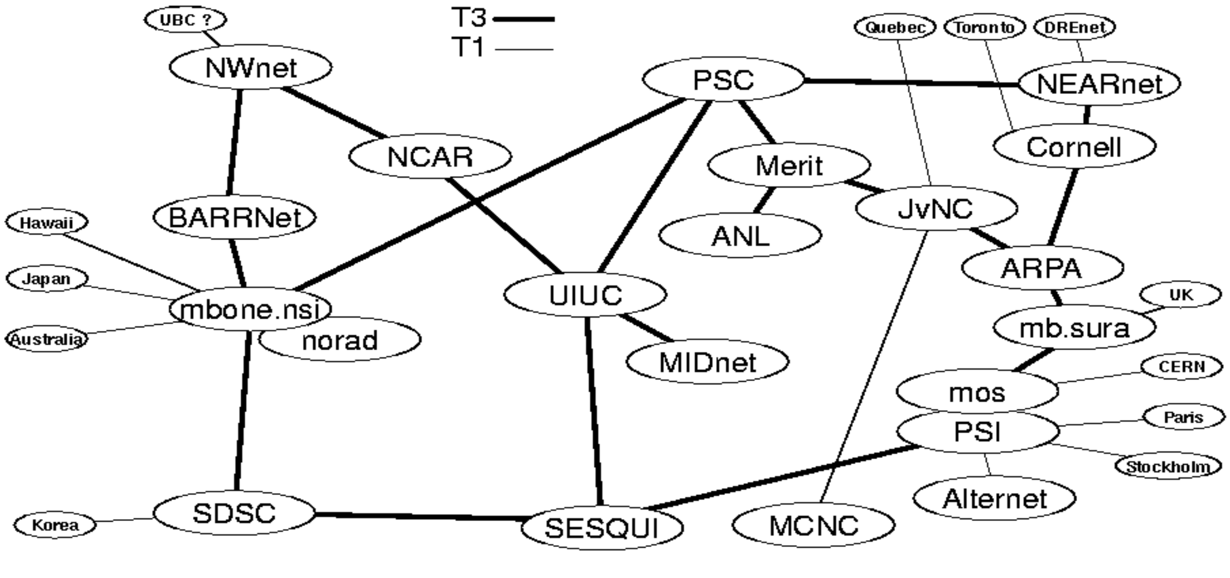


Figure 3: MBONE topology in North America.

Topology	Nodes	Links
10-ary 2-cube	100	400
5-ary 3-cube	125	750
5-ary 2-cube	25	100
MBONE	18	42

Table 1: Characteristics of topologies

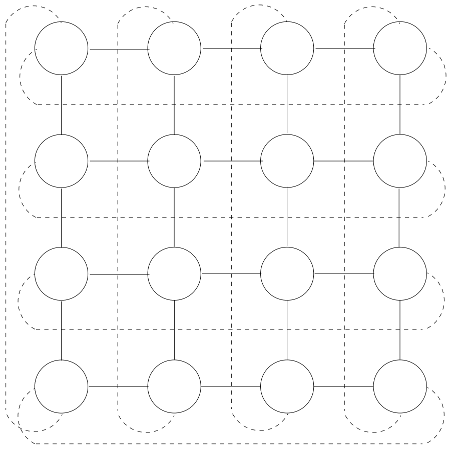
Trigger(0.1) (Trigger(0.5)), link-state information is distributed when the available link bandwidth changes by more than 10% (50%) from the recently-distributed value. Static routing determines a QoS route solely based on the static topology of the network. Although it incurs a very small overhead and operational cost, its performance will later be shown unacceptable in most cases.

The simulation study uses homogeneous traffic patterns, with uniform random selection of source and destination nodes. For simplicity, it also assumes exponentially-distributed connection-request inter-arrival times. Instead of using more realistic traffic models, we opted for simple traffic patterns, because our goal is to comparatively evaluate the proposed scheme and others, as opposed to providing accurate absolute performance figures. Connection bandwidths are uniformly-distributed with in an interval with a spread around the mean $\overline{b}$. For instance, connection bandwidths may have a mean of 10% of link capacity with a spread of 200%, resulting in $b \sim U(0.0, 0.2]$. In order to keep the load constant, an appropriate number of randomly-selected old connections are disconnected when a new connection is established. At the beginning, no connections are deleted until the load reaches a steady-state level. The simulations were run until connection blocking rates' 99% confidence intervals were within 1%.

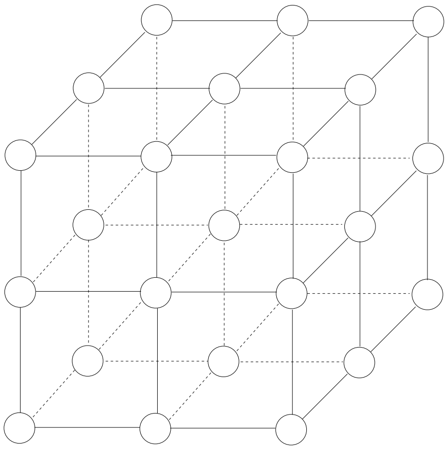
4.2 Simulation Results

Depending on the network configuration and bandwidth usage by the requested connections, the seven schemes considered exhibit a wide range of connection-success/blocking probability.

Figure 5 shows the connection-blocking probability of each scheme when the bandwidth range is 4% of



(a) 4-ary 2-cube



(b) 5-ary 3-cube

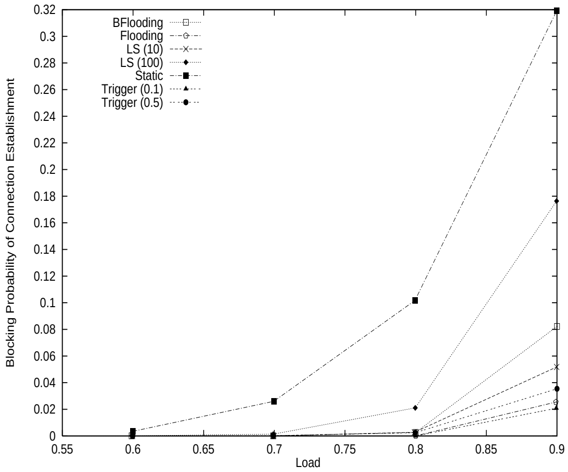
Figure 4: Example k-ary n-cube topologies.

the link capacity. Such a small bandwidth range allows us to look at a realistic scenario in realizing real-time communication service in general ISPNs. The bandwidth requirement of a real-time connection is usually quite small compared to the link capacity. In case of k-ary n-cubes, the blocking probability increases in the order of Trigger(0.1), Flooding, Trigger(0.5), LS(10), BFlooding, LS(100), and Static. The blocking probabilities of Flooding and Trigger(0.1) are almost the same except when load = 0.9. Flooding is supposed to provide the best performance in terms of connection-success rate since it searches all the possible routes, but, because of the temporarily-reserved link bandwidth due to flooding, its blocking is higher than that of Trigger(0.1) under such a heavily-loaded condition. The performance of the proposed scheme lies between LS(10) and LS(100), but its blocking probability is very small, showing almost no difference from those of other schemes except when load = 0.9. Even when load = 0.9, its blocking probability is under 5%. Actually, the performance of the proposed scheme is quite attractive in view of the large overhead and high operational cost of the other schemes.

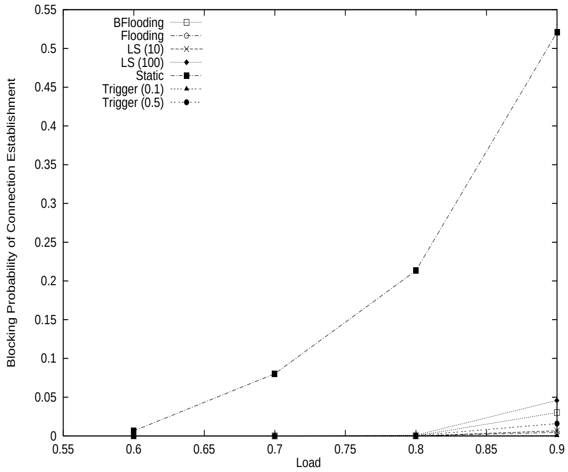
For the MBONE topology that has poorer connectivity, all but Static show similar performance, because all the schemes have a very few alternate paths in this sparsely-connected network, and thus, almost all candidate routes are always considered in determining a qualified route. While our scheme shows a slightly higher blocking probability, its performance is much closer to those of Flooding or link-state routing than Static.

Slightly-deteriorated performance of our scheme can be offset by its small overhead and low operational cost because it doesn't require path calculation. For the bandwidth range of 4%, the overheads are given in Table 2 and Figure 6. We consider only the number of messages as overhead, regardless whether messages are generated for establishing a connection as in BFlooding and Flooding, or distributing link-state information as in LSs and Triggers. This comparison may not be fair since link-state messages and connection-request messages contain uncorrelated information. However, they both are transmitted over links and consume bandwidth & processing resources. So, we consider the number of messages generated during an average connection inter-arrival time as the overhead, regardless whether messages are for either distributing link-state or requesting a connection setup.[4] Let's consider Flooding which shows the best performance in most

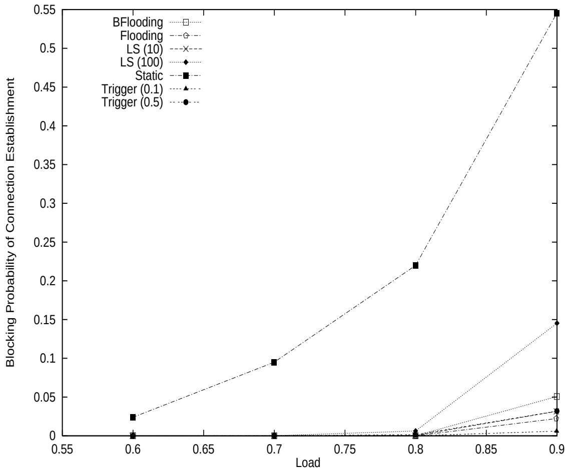
[4]One may argue that the proposed scheme requires the distribution and management of static topology information. But, this



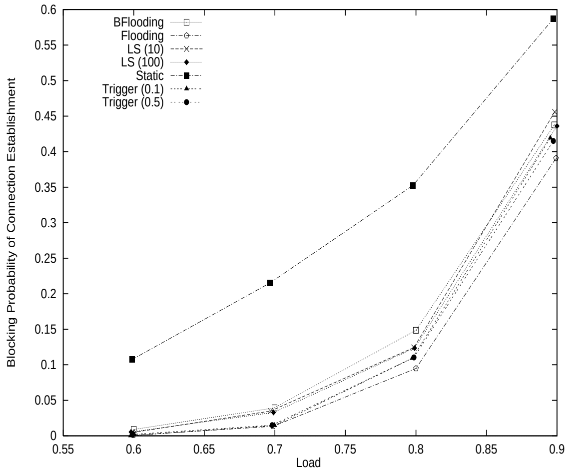
(a) 5-ary 2-cube



(b) 5-ary 3-cube



(c) 10-ary 2-cube



(d) MBONE

Figure 5: Connection-blocking probability, $b \sim (0, 4\%])$

load ranges. In Figure 6, the number of messages generated per connection request is around 100, 1000, 500, and 20 in 5-ary 2-cube, 5-ary 3-cube, 10-ary 2-cube, and MBONE, respectively. As load increases, the number of request messages decreases due to the early pruning at heavily-loaded links. By contrast, BFlooding incurs a very small number of request messages, as shown in Table 2 and Figure 6. This is the result of limiting the search area. Now, let's consider Trigger(0.1) and Trigger(0.5) in Figure 6. The small trigger level of Trigger(0.1) generates a large number of link-state broadcasts. In densely-connected networks like the 10-ary 2-cube, the number of link-state messages ranges between 250 and 900. This large overhead makes Trigger(0.1) impractical for QoS routing in spite of its good performance. The overhead increases with load in Trigger(0.1), indicating frequent triggers under heavily-loaded conditions as expected. However, this does not apply to the MBONE topology in which the load condition changes less frequently even under heavily-loaded conditions, as the connection-success probability is much smaller due to scare alternate paths, and thus, less frequent changes in link-state.

information not only changes rarely but also can be easily obtained from the datagram routing scheme (e.g., OSPF) which works concurrently with the proposed QoS routing without incurring any additional overhead.

Topology	BFLOODING	FLOODING	LS (10)	LS (100)
10-ary 2-cube	25–33	376– 479	4000	400
5-ary 3-cube	19 – 23	866–1062	9375	936
5-ary 2-cube	6–8	74 – 101	250	25
MBONE	5 – 12	10 – 24	76	8

Table 2: Overheads for different routing schemes.

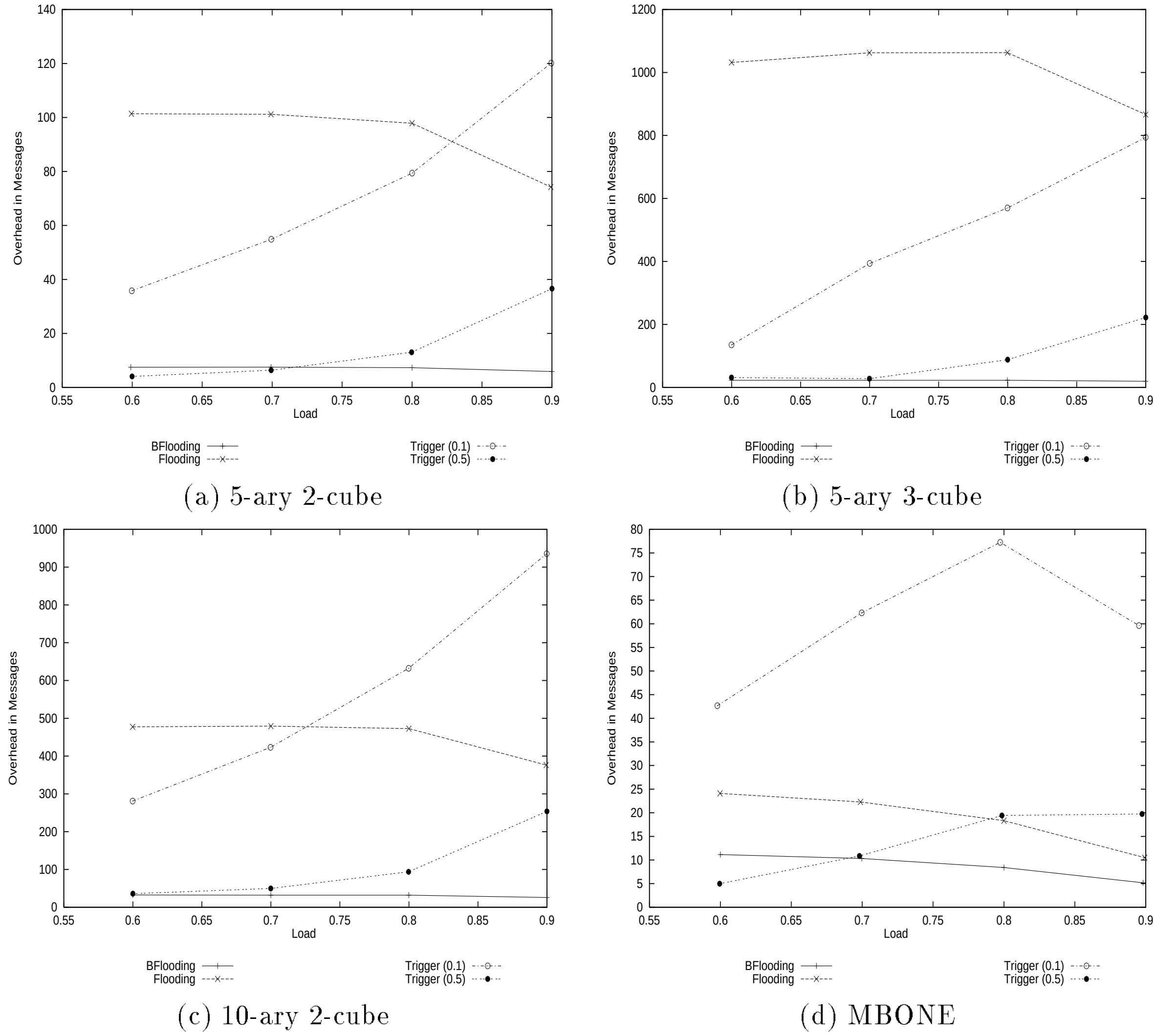
Compared to Trigger(0.1), Trigger(0.5) shows much more reasonable overhead. Especially, in the low-to-medium range of load, its overhead is comparable to that of our scheme. Considering its performance and overhead, Trigger(0.5) appears quite a promising choice for QoS routing. However, as with Trigger(0.1), the overhead of Trigger(0.5) increases with load dramatically. In contrast, BFlooding's overhead remains small under all load conditions. For example, in the MBONE topology, BFlooding generates only 5–12 request messages for setting up a real-time connection.

In LS(10) and LS(100), the number of link-state messages is constant irrespective of load ranges or conditions; this is why their overheads in Table 2 are shown separately from those of Triggers. During every updating period, each link generates its new link-state messages.[5] The new state of a link must be distributed to all the nodes in the network. The distribution can be done using either flooding or broadcasting through a minimum spanning tree. The latter approach generates a smaller number of messages. Assuming broadcasting through a minimum spanning tree, 40000 messages are transmitted in a 10-ary 2-cube in an update period. The number of messages was derived from the number of nodes and links in Table 2. That is, on average, LS(10) generates 4000 messages per request, and LS(100) does 400. As seen in Table 2, the overheads of LS(10) and even LS(100) are extremely large except the case when LS(10) is applied in the MBONE topology. Although LS(100) and LS(10) are comparable to BFlooding in terms of performance, their overhead is much larger than BFlooding, and, in some cases, larger than that of Flooding.

Figures 7 and 8 show the connection-blocking probabilities when the bandwidth ranges are 10% and 15%, respectively. Overall, the blocking probability increases as the bandwidth range increases, verifying the fact that a connection with a larger bandwidth requirement is less likely to be established. Moreover, as the bandwidth range gets larger, the gap between blocking probabilities of our scheme and Trigger(0.5) increases, which is the main comparison target in terms of performance and overhead. However, the gap between the overheads of our scheme and Trigger(0.5) also increases. The overhead of Triggers increases with bandwidth range because link-state changes more rapidly, triggering more frequent link-state distribution when bandwidth range gets large. In contrast, the overhead of our scheme does not change with bandwidth range.

In order to investigate the tradeoff between message overhead and performance loss, we varied the search-scope of a connection. By increasing the search-scope, we expected performance improvement in terms of connection-success probability. In this experiment, additional hops were added to the original search-scope which was set so as to search at least two alternate paths. We only considered the case when the bandwidth range is 10%. In Figures 11 and 12, the connection-blocking probability and message overhead for the 10-ary 2-cube and the MBONE topology are plotted for our approach with different search-scopes, Flooding and Trigger(0.5). In BFlooding(n), we added n hops to the original search-scope used by BFlooding. For

[5] Updates must not be synchronized to prevent the network from being overloaded with a burst of link-state messages.



(a) 5-ary 2-cube

(b) 5-ary 3-cube

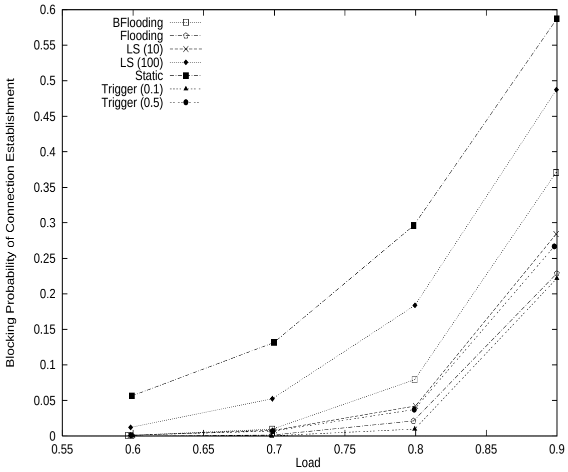
(c) 10-ary 2-cube

(d) MBONE

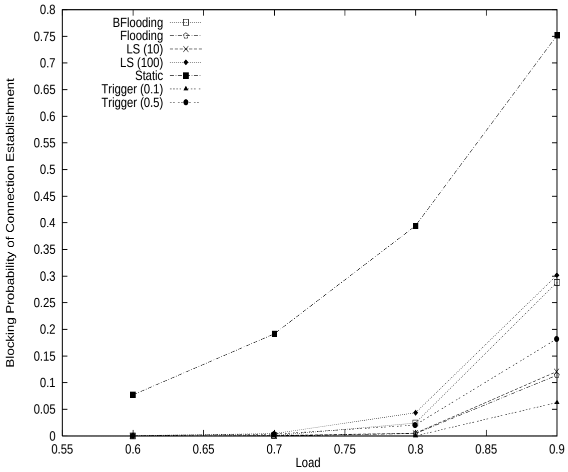
Figure 6: Overheads of different routing schemes, $b \sim (0, 4\%]$

the 10-ary 2-cube, increasing the search-scope by one or two hops does not add any route to be searched, thus causing no change in performance or message overhead. Thus, the search-scope was increased by 3, 6, 9, and 12 hops[6]. When the increment is 3, the connection-blocking probability is lower than that of Trigger(0.5) or even Flooding. However, the number of messages generated per request is much larger than that in BFlooding. Compared to Trigger(0.5), BFlooding(3) incurs larger overhead under a mid-range load, but smaller under a heavily-loaded condition. This is because, under a heavily-loaded condition, a smaller number of request messages are relayed due the bandwidth test failure in BFlooding(3) while more frequent link-state distributions are required in Trigger(0.5). Little gain is made by adding more hops to the search-scope as seen in Figure 11. In fact, the best performance was achieved when the increment was 6. As the increment gets larger, the performance deteriorates, because temporarily-reserved link bandwidth prevents new connections from passing the bandwidth test as with Flooding. Despite the little gain in performance, the increase in message overhead is very large in BFlooding(6), implying that increasing the search-scope

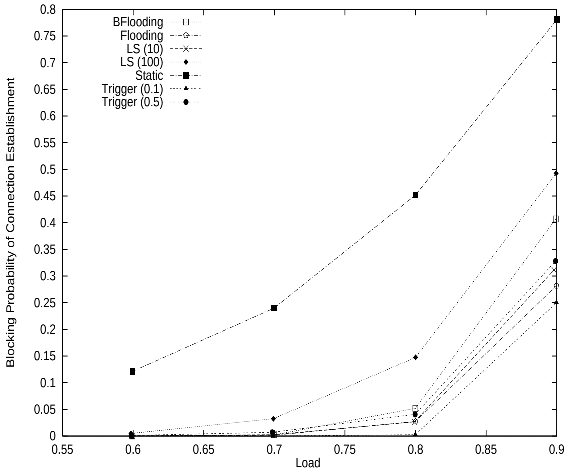
[6] some cases are omitted in the figures for clarity



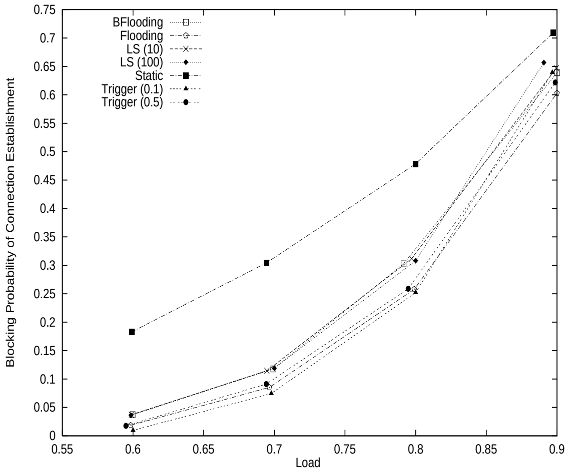
(a) 5-ary 2-cube



(b) 5-ary 3-cube



(c) 10-ary 2-cube

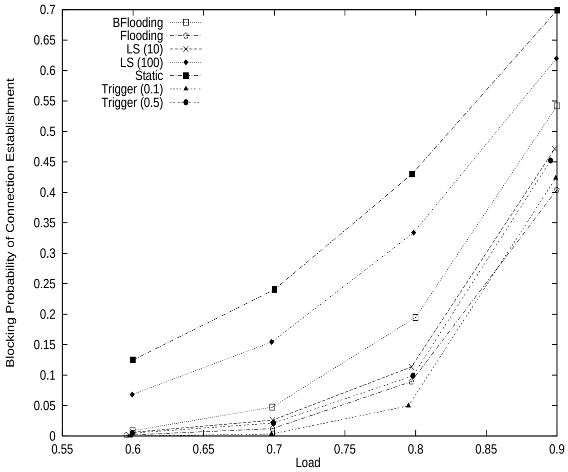


(d) MBONE

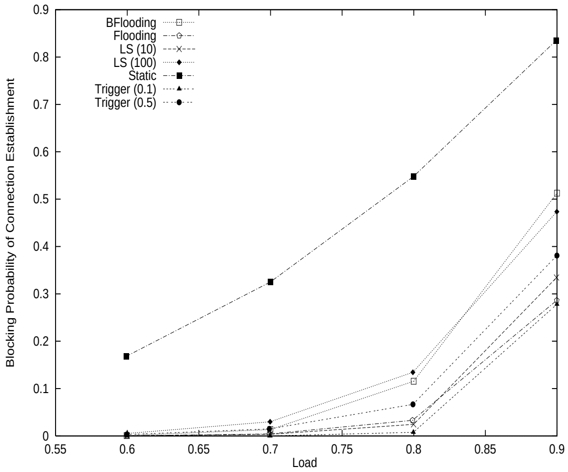
Figure 7: Connection-blocking probability, $b \sim (0, 10\%]$

beyond some limit does not help at all. The overhead does not change after the increment gets larger than 6. By setting the search-scope to ∞, one can obtain the result of Flooding. BFlooding(3), BFlooding(6), and BFlooding(9) are superior to Flooding in terms of performance and message overhead. The result for the 10-ary 2-cube suggests use of an adaptive approach for determining the search-scope, depending on network load. That is, in case of heavy load, a larger search scope needs to be used, which will greatly improve the connection-success probability of our approach. Since our approach is not based on any global network-load information, it must use an indirect approach like prediction based on the connection-success rate or local information on the source node's outgoing links. The result for the MBONE topology verifies the same trend observed in the 10-ary 2-cube.

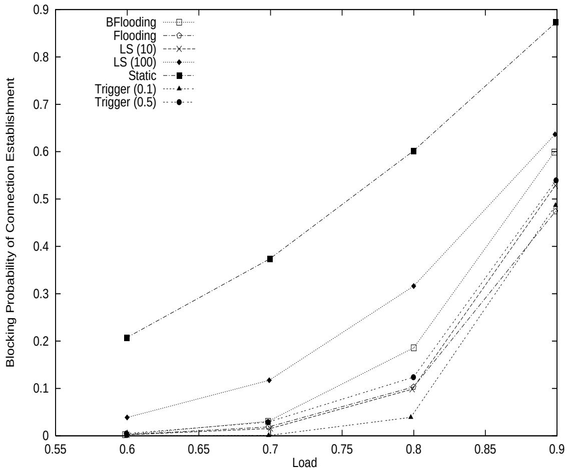
In summary, the proposed QoS routing is very attractive, especially in view of the drastic reduction in overhead and operational cost, yet providing good performance.



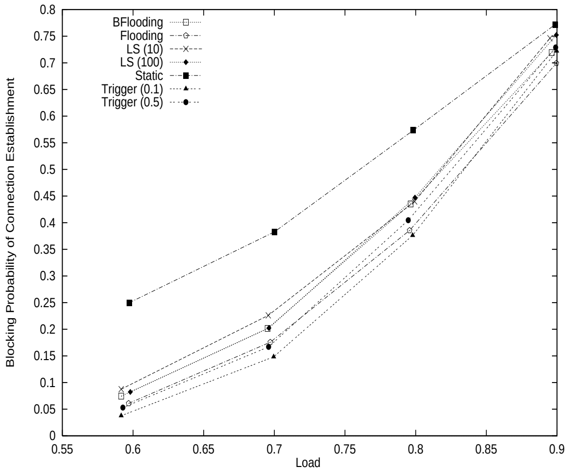
(a) 5-ary 2-cube



(b) 5-ary 3-cube



(c) 10-ary 2-cube



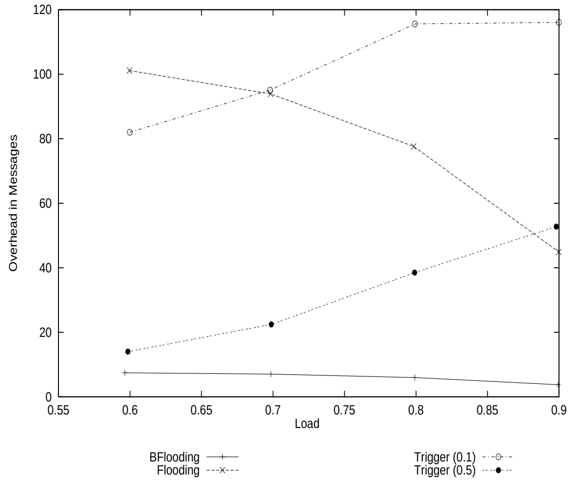
(d) MBONE

Figure 8: Connection-blocking probability, $b \sim (0, 15\%]$

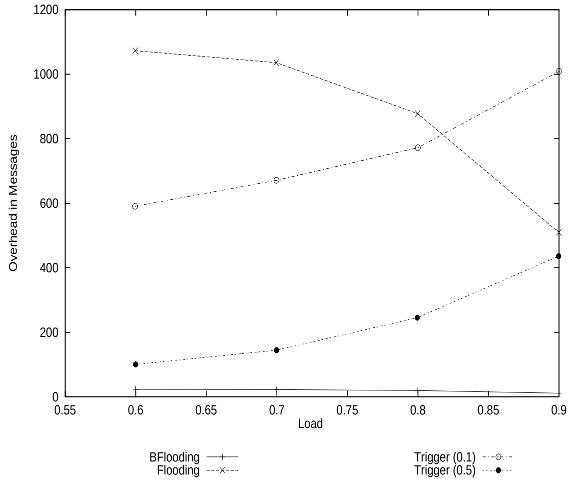
5 Conclusion

In this paper, we have proposed and evaluated a cost-effective QoS-routing scheme that incurs small overhead and operational cost, but provides reasonably good performance. Unlike link-state routing, the proposed scheme does not require distribution and maintenance of link-state information, nor expensive on-line path computation. Instead, every node is required to keep a distance table which can be obtained off-line using almost static network-topology information. A qualified route for each requested real-time connection is searched by flooding request messages with a limited hop count.

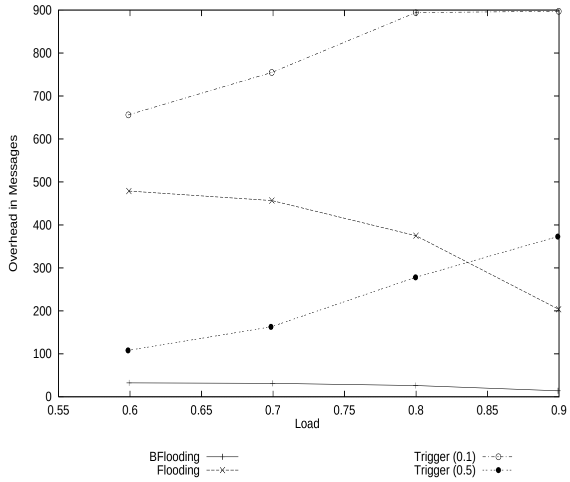
Using an in-depth simulation study, we have comparatively evaluated the performance and overhead of ours and others'. In terms of overhead, our scheme is absolutely the best, although some of the other schemes provide performance slightly better than, or comparable to, our scheme. Although our scheme has a lower connection-success probability than brute-force flooding or link-state routing, it still provides reasonably good performance at much lower overhead and cost. Enlarging the flooding area by changing the search-scope can



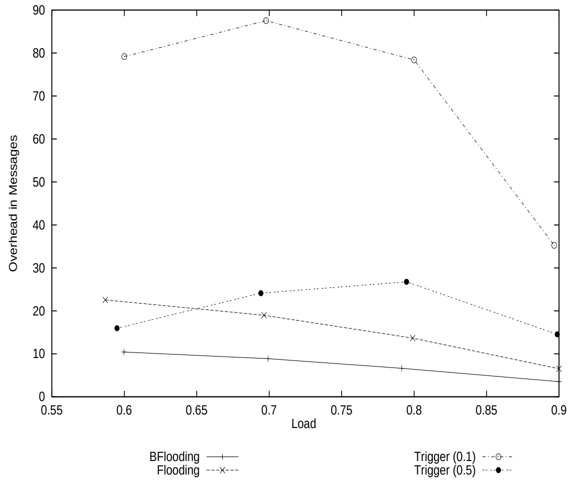
(a) 5-ary 2-cube



(b) 5-ary 3-cube



(c) 10-ary 2-cube



(d) MBONE

Figure 9: Overhead, $b \sim \ (0, 10\%]$

improve the scheme's performance at the expense of larger overhead. The tradeoff between performance and overhead needs to be considered in determining the search-scope. This is a matter of our future research.

References

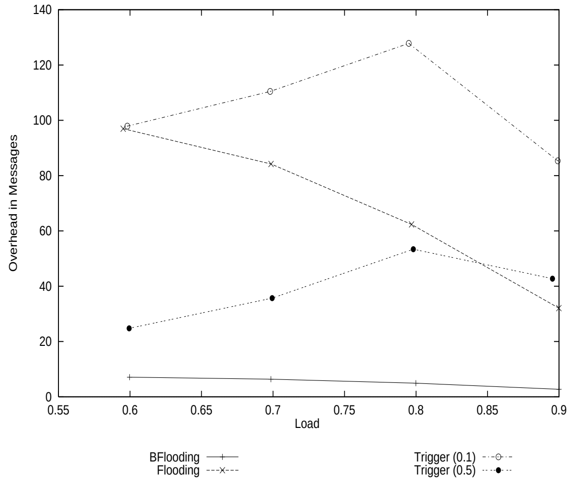
[1] L. Zhang, "Virtual clock: a new traffic control algorithm for packet switching networks," in *Proc. of ACM SIGCOMM 90*, pp. 19–29.

[2] C. Kalmanek, H. Kanakia, and S. Keshav, "Rate controlled servers for very high-speed networks," in *Proc. of IEEE GLOBECOM 90*, pp. 12–20, 1990.

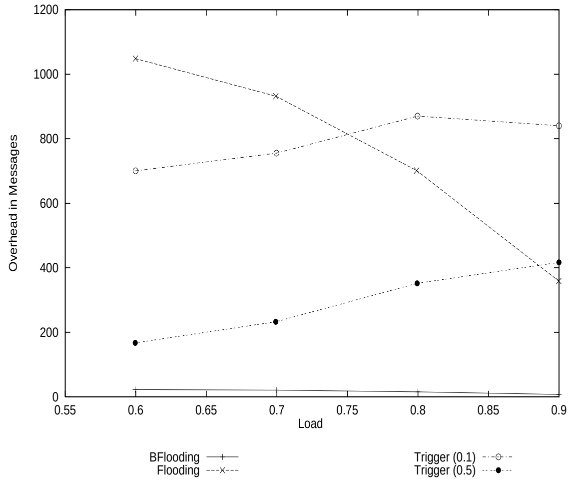
[3] S. J. Golestani, "A stop-and-go queueing framework for congestion management," in *Proc. of ACM SIGCOMM 90*, pp. 8–18.

[4] D. Ferrari and D. C. Verma, "A scheme for real-time channel establishment in wide-area networks," *IEEE J. Select. Areas Commun.*, vol. 8, pp. 368–379, Apr. 1990.

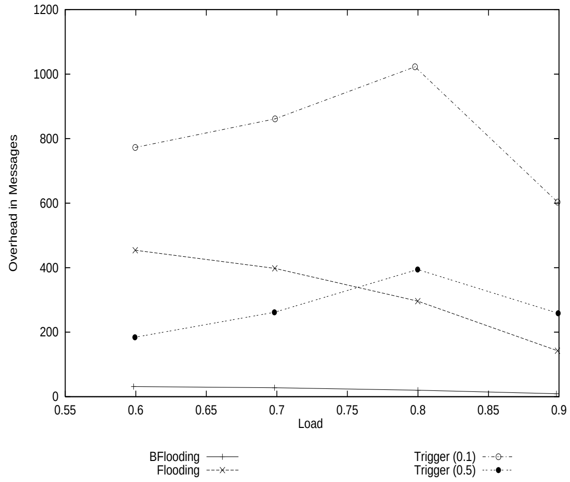
[5] H. Zhang and D. Ferrari, "Rate-controlled static-priority queueing," in *Proc. of IEEE INFOCOM 93*, pp. 227–236.



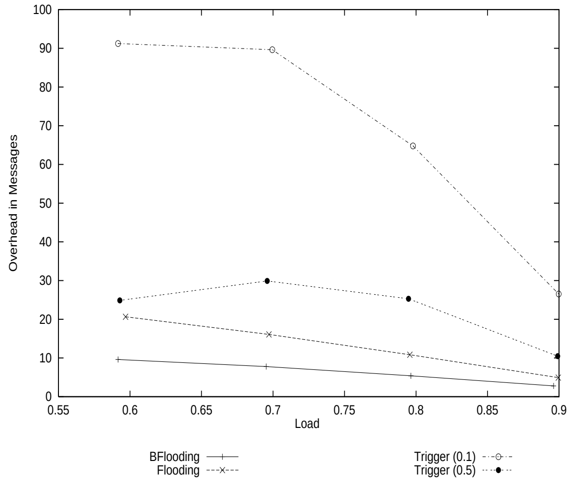
(a) 5-ary 2-cube



(b) 5-ary 3-cube



(c) 10-ary 2-cube



(d) MBONE

Figure 10: Overhead, $b \sim (0, 15\%]$

[6] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," in *Proc. of ACM SIGCOMM 89*, pp. 1–12.

[7] A. K. Parekh, *A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks.* PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Feb. 1992.

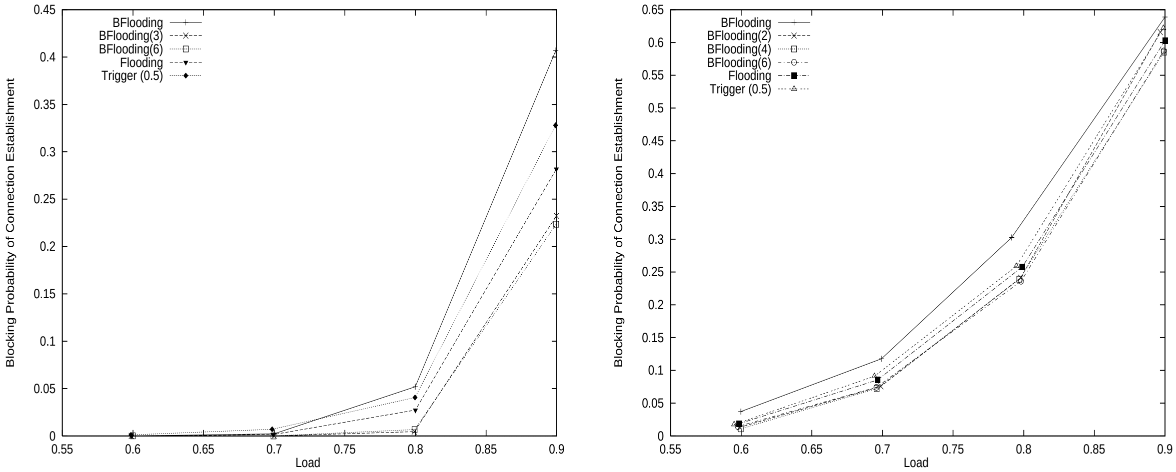
[8] D. D. Kandlur, K. G. Shin, and D. Ferrari, "Real-time communication in multi-hop networks," in *Proc. 11-th Int'l Conf. Distributed Comput. Systems*, pp. 300–307, May 1991.

[9] S. Kweon and K. G. Shin, "Traffic-controlled rate-monotonic priority scheduling," in *Proc. of IEEE INFOCOM 96*, pp. 655–662.

[10] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala, "RSVP: A new resource ReSerVation protocol," *IEEE network*, pp. 8–18, Sept. 1993.

[11] D. D. Clark, S. Shenker, and L. Zhang, "Supporting real-time applications in an integrated services packet network: architecture and mechanism," in *Proc. of ACM SIGCOMM 92*, pp. 14–26.

[12] PNNI Working Group, "ATM Forum 94-0471R13 PNNI Draft Specification." Document available at ftp://ftp.atmforum.com/pub/contributions.



(a) 10-ary 2-cube (b) MBONE

Figure 11: Connection-blocking probability for different search-scopes

[13] Z. Zhang, C. Sanchez, B. Salkewicz, and E. S. Crawley, "Quality of service extensions to ospf or quality of service path first routing (qospf)." Internet Draft (draft-zhang-qos-ospf-01.txt), Sept. 1997.

[14] R. Guérin, S. Kamat, and A. Orda, "QoS routing mechanisms and OSPF extensions." Internet Draft, Mar. 1997. To appear in *Proceedings of IEEE GLOBECOM*, November 1997.

[15] K. G. Shin and C.-C. Chou, "A distributed route-selection scheme for establishing real-time channels," in *Proc. 6-th IFIP Int'l Conf. on High Performance Networking Conf. (HPN'95)*, pp. 319–329, Sept. 1995.

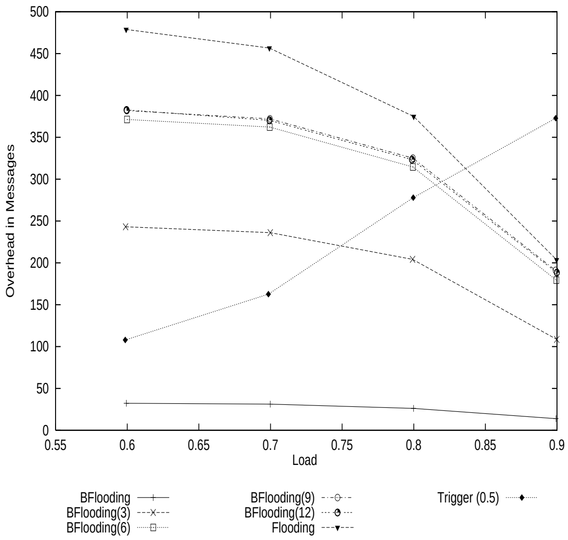
[16] C.-J. Hou, "Routing virtual circuits with timing requirements in virtual path based atm networks," in *Proc. of IEEE INFOCOM 97*, pp. 320–328, Apr. 1996.

[17] Z. Wang and J. Crowcroft, "Quality-of-service routing for supporting multimedia applications," *IEEE J. Select. Areas Commun.*, vol. 14, pp. 1228–1234, Sept. 1996.

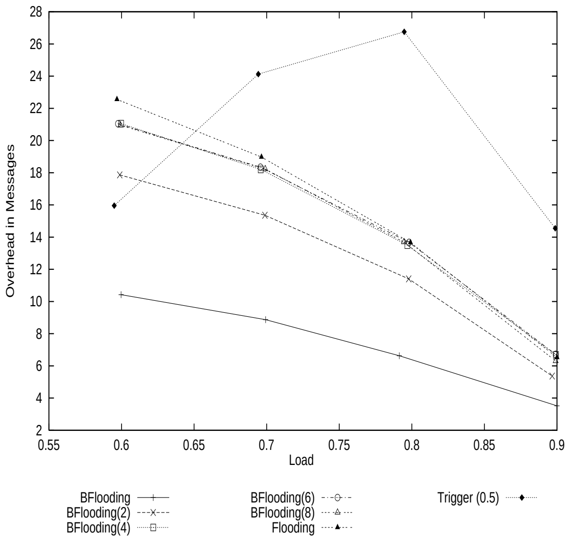
[18] Q. Ma, P. Steenkiste, and H. Zhang, "Routing high-bandwidth traffic in max-min fair share networks," in *Proc. of ACM SIGCOMM 96*, pp. 206–217, Aug. 1996.

[19] A. Shaikh, J. Rexford, and K. Shin, "Dynamics of quality-of-service routing with inaccurate link-state information," Technical Report CSE-TR-350-97, Dept. of Electrical Engineering and Computer Science, the University of Michigan, Ann Arbor, MI, Nov. 1997.

[20] Q. Ma and P. Steenkiste, "On path selection for traffic with bandwidth guarantees," in *Proc. of IEEE International Conference on Network Protocols*, Oct. 1997.



(a) 10-ary 2-cube



(b) MBONE

Figure 12: Overhead for different search-scopes