Selective Flooding for Improved Quality-of-Service Routing

Mark Claypool and Gangadharan Kannan

Computer Science Department
Worcester Polytechnic Institute
100 Institute Road
Worcester, MA, USA 01609

ABSTRACT

Quality-of-service (QoS) requirements for the timely delivery of real-time multimedia raise new challenges for today's networks. A key component of QoS is QoS routing which allows the selection of network routes with sufficient resources for requested QoS parameters. Most proposed techniques to compute QoS routes require dynamic update of link-state information. Given the growing size of internets, it is becoming increasingly difficult to gather up-to-date state information. We propose a new technique to compute QoS routes in a fast and efficient manner without any need for dynamic updates. Our method, known as *Selective Flooding*, checks the state of the links on a set of pre-computed routes from the source to the destination in parallel and based on this information computes the best route and then reserves resources. We implemented Selective Flooding on a QoS routing simulator and evaluated the performance of Selective Flooding compared to source routing for a variety of network parameters. We find Selective Flooding consistently outperforms source routing in terms of call-blocking rate and outperforms source routing in terms of network overhead for some network conditions. The contributions of this work include the design of a new QoS routing algorithm, Selective Flooding, extensive evaluation of Selective Flooding under a variety of network conditions and a working simulator for future research.

Keywords: Quality of Service, Routing

1. INTRODUCTION

Today's Internet is predominantly a connection-less, best-effort network. Data packets from the same application can take different paths to the destination and packets can be lost, duplicated or arrive out of order. Network resources (switches, buffers etc.) are shared uniformly amongst various applications. While this infrastructure is effective for conventional text-based applications, it does not meet the requirements of integrated service networks (existing ISDN and B-ISDN networks and future ISPN networks[1]). Specifically it does not provide resource reservation, which is vital for guaranteed end-to-end performance. To provide guarantees on network performance for applications, the notion of Quality-of-Service (*QoS*) has been proposed to capture the qualitatively or quantitatively defined performance contract between a service provider and user applications.[2]

It is likely that the next generation of the Internet will be connection oriented,[3] where for every flow, there will be a fixed network path consisting of switches and links. Routing in this context is the problem of finding the network path for connection establishment. QoS routing can be defined as finding a network path that will satisfy application performance requirements by selecting paths based on connection traffic parameters and available link capacity. Applications can typically specify their QoS requirements as a set of constraints, either link constraints or path constraints. Link constraints specify a restriction on the use of individual links in building a path. For example, a bandwidth constraint of a unicast connection* could require that links composing the path have a certain amount of free bandwidth available. A path constraint, on the other hand, specifies a requirement on the entire path. For example, a delay constraint of a multicast connection may require that the longest end-to-end delay from the sender to any receiver not exceed an upper bound. The basic function of QoS routing is to find a feasible path, (ie - a path that has sufficient resources to satisfy the QoS constraints of a connection). Additionally, we would like to find the optimal (least cost) path among all feasible paths, in order to improve total network utilization.

In addition to QoS routing, there are other components to a QoS-sensitive network:

Further author information: Mark Claypool, email: claypool@cs.wpi.edu, phone: 508-831-5409

*The terms *call*, *connection* and *request* are used interchangeably in this paper.

- *Resource Reservation.* Routing and reservation are closely related network components. In order to provide guaranteed services, resources need to be reserved for every accepted connection. In an ATM network, for every call, it reserves a virtual channel over all links on the route from the source to the destination. RSVP can be used to do reservation at the IP layer.[4]

- *Admission Control.* Given limited network resources, it is not possible to provide QoS to every requested connection. Therefore, admission control accepts a call only if the network can support that call at that instance. For every QoS request, an admission control algorithm will try to reserve the necessary network resources. If successful the call is accepted, otherwise it is refused or blocked.

- *QoS negotiation.* A QoS routing algorithm may fail to find a path for a particular request. In such cases it can either reject the call or negotiate by returning back the best that can be supported. If this negotiation is successful then the path can be used right away.

In order to compute a QoS route, accurate information of the network state is required. This information consists of the network topology and the link-state information (available bandwidth, delay through the link, etc) on each link of the network. There are different routing strategies depending on where this information is stored and how it is parsed. In *source routing*, each node maintains the complete global state, including the network topology and the state information of every link. Based on the global state, a feasible path is locally computed at the source node. A control message is then sent out along the selected path to inform the intermediate nodes of their precedent and successive nodes. A link-state protocol (such as QoS extensions to OSPF[5,6]) is used to update the global state at every node. In *distributed routing*, the path is computed by a distributed computation during which control messages are exchanged among the nodes and the state information kept at each node is collectively used for the path search. Most distributed routing algorithms need a distance vector protocol to maintain a global state in the form of distance vectors at every node. Based on the distance vectors, the routing is done on a hop-by-hop basis.

There are various metrics used to evaluate and compare different routing algorithms. Primary among them are:

- *Call-blocking rate.* This refers to the percentage of calls blocked or the percentage of calls that were not admitted into the system. Typically for a given topology and a given traffic pattern, there is an absolute minimum call-blocking rate. Beyond this minimum, the network cannot accept any more calls because it does not have the resources requested by the call at that time. Any routing algorithm should try to have a call-blocking rate as near to this minimum as possible.

- *Network Overhead.* This refers to the overhead incurred by the network due to routing. This can be further divided into the bandwidth used for communication by the routing algorithm and the computational load on the routers due to the routing algorithm.

- *Call Setup Time.* This refers is the time from when the source receives a QoS request to the time the actual connection to the destination is established.

- *Source Computation Cost.* This refers to the computation cost at the source router which receives the original QoS request.

- *Scalability.* This refers to how the routing algorithm will perform as the size of the network increases.

Using accurate network information helps in getting a reduced call-blocking rate. This information is difficult to obtain as the network state continuously changes due to load fluctuations. Further, there is a network overhead involved in maintaining this information. Transmitting link-state information (as is done in source routing) to every other node at regular intervals consumes network bandwidth. Every router also has a computation overhead, as it needs to update its routing tables. On the other hand, using stale network information to compute QoS paths can seriously degrade the performance of the system and increase the call-blocking rate, since the information which is being used to make the routing decision could be potentially different from the actual network state. Thus, we have a tradeoff between the call-blocking rate achieved and the network overhead caused by routing. Also, as the network size grows the cost of keeping accurate link-state information for the whole network grows exponentially. Thus scalability is a problem with most QoS routing algorithms.

This paper proposes a new QoS routing algorithm, *Selective Flooding*. Briefly the way Selective Flooding works is as follows: every source has a static image of the topology, consisting of a graph of the nodes and the connections between them. Based on this graph multiple routes from the source to every destination are computed and stored, with no periodic link state information transmitted. Whenever a QoS request arrives, control packets are flooded through these precomputed routes. These control packets collect QoS information on their way. Based on this information the best path is computed at the destination. Resources are then reserved along this path.

We expect Selective Flooding to perform close to optimum in terms of call-blocking rate as it uses the most recent link-state information to route the connection requests. Further, since no link-state update is performed, the network overhead should also be reduced versus source routing. In this paper, we simulate the Selective Flooding algorithm and evaluate its performance compared to source routing.

The rest of this paper is organized as follows: Section 2 details the design of Selective Flooding routing; Section 3 describes the set-up we use to simulate and evaluate Selective Flooding; Section 4 presents our results from our evaluations; Section 5 concludes our findings; and Section 6 summarizes possible future work.

2. SELECTIVE FLOODING

Source routing stores two important pieces of information at every source: 1) the network topology (nodes and presence of links between them); and 2) the latest link-state information about the whole network. The topology is relatively static and does not cost much in terms of network overhead. However, transmitting link-state information is expensive and needs to be done at regular intervals, thus consuming non-trivial amounts of bandwidth across the network and using up considerable computing resources at each router. The more frequent these link-state updates, the more these resources are used. On the other hand, the call-blocking rate increases as we use more stale link-state information. Thus, with Source Routing, we have a tradeoff between network usage (efficiency) and call-blocking rate. Selective Flooding proposes to store the network topology but not the link-state information across the network and obtain a low call blocking rate by doing on demand selective path exploration for each call.

The basic idea of Selective Flooding is to evaluate multiple routes from a source to a destination in parallel. Each source on the network is assumed to have a static image of the entire network. This image needs to consist only of the set of nodes and edges which make the whole topology, and does not require link-state information. As mentioned before, such topology information is fairly static and does not need to be updated on a regular basis. Using this network topology, every source computes all possible paths to every destination. These routes are then stored in an easily retrievable routing table.

When a QoS request is received from an application at a higher level, control packets (messages) are flooded across all possible routes to the destination as listed in the precomputed routing table. Along each route traversed, at each router, the control message collects QoS information such as cumulative delay or the available bandwidth on the outgoing link. When all the control messages arrive at the destination, the best path that satisfies the initial QoS constraints is computed. The best path can be determined either at the destination itself or all the information can be passed back to the source and the decision can be made at the source. Once the best path is computed, resources along this path are reserved.

The QoS information collected at each intermediate node depends on the parameters on which the QoS application places requests. For delay-sensitive applications this could be the cumulative delay along the route. For rate-sensitive applications the QoS information could be the available bandwidth on each link along the route. In both cases this information will give a list of feasible paths from the source to the destination for the particular QoS request. The source or destination can then compute the best path from among these feasible paths. As in previous work,[7] the control messages proceed only when the nodes and the links on the path have sufficient resources. Thus every control message arriving at the destination detects a feasible routing path.

To better illustrate Selective Flooding consider the sample topology in Figure 1(a). At initialization each source computes all the possible paths from the source to the destination. When an application at A makes a QoS request for a path to E, A retrieves from its precomputed table a list of possible routes to E. It then sends in parallel control packets along all the routes to E (Figure 1(c)). These packets continue along the path only if the path has the requested resources. Thus unlike in Source Routing (Figure 1(b)), even if there is a single feasible path Selective Flooding finds the path. The time taken to make this routing decision is a constant (ignoring any queuing delays), corresponding to the length of the longest path probed.

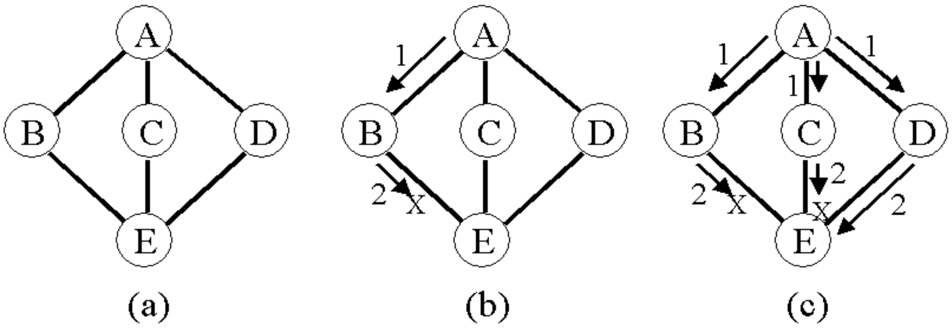


Figure 1. QoS Routing Illustration (a)-Topology (b)-Source Routing (c)-Selective Flooding. The letters are nodes. The numbers indicate sequential steps. The X's indicate routing failures.

The best path means the path among all feasible paths which minimizes the call blocking rate in the network. Different metrics can be used to define the best path depending on the kind of QoS constraints and the resource whose utilization needs to be optimized. Reasonable candidates include either the shortest feasible path in terms of number of hops or the path which maximizes the residual bandwidth as the best path. Using the shortest feasible path as the best path reduces the client waiting time. Further, it is possible to mark certain links as "avoid" or "preferred" based on current link-usage thus performing load-balance across the network.

Most of the proposed QoS routing algorithms require extensive and current knowledge of the entire network in terms of link-state information to compute feasible QoS paths. Selective Flooding eliminates the need for any link-state information. Only knowledge of the topology (which is relatively static) is required. This has enormous potential to reduce the computation at the router and also reduce the call-blocking rate. The link-state information that is used to compute the best path in this case is stale by one Round Trip Time (RTT) and the computation time of the algorithm, typically far better than any source routing mechanism. Since Selective Flooding uses the latest link-state information, it is expected that it will have a call-blocking rate close to the best achievable for the given topology under the given traffic. This is much better than typical Source Routing which uses stale link-state information and consequently can have a high call-blocking rate.

There is a network cost involved with every call. The network overhead in Selective Flooding is per-call whereas in Source Routing it is per link-state update period. For every call, Selective Flooding needs to send probe packets into the network along the precomputed paths. This cost is proportional to the average length of the path and the number of paths probed for each call. In the case of Source Routing link-state update packets are transmitted throughout the network every time the global state is updated. The overall number of packets sent due to routing is more in the case of Selective Flooding. However the computation cost at each router when a routing packet arrive needs to be considered. In Source Routing, for every link-state update packet, each router needs to update its image of the network and recalculate its routing table. This requires considerable computation at the router. During Selective Flooding, whenever a control message (probe packet) arrives, all that the router needs to do is to load the necessary QoS information onto the packet and send it to its next hop. Thus even though Selective Flooding sends more routing packets into the network, it saves considerable computation time at the routers.

Selective Flooding requires that each source be able to store multiple paths to every destination. The storage space required for doing this is proportional to the number of nodes in the network and the overall network connectivity. There is a start-up computation cost when the source has to actually compute all the routes to every possible destination, but this is fairly static and done typically at boot time only.

3. IMPLEMENTATION

We implemented Selective Flooding by using simulation, since an analytical model would be too coarse given all the possible parameters and an actual implementation is difficult for reasons of inter-operability and hardware costs.

3.1. QoS Routing Simulation

We implemented selective flooding in *routesim*,[8] an open-source, event driven simulator used to study QoS routing in large networks.[9–11] *routesim* allows precise control over network traffic characteristics, network topology, routing algorithms and policies, and link-state update policy. *routesim* has a number of tunable parameters which are used to specify these and other characteristics of the simulator, such as hop-count thresholds, arrival distribution, and more.

The features of *routesim* important to this work include:

Topology: Any kind of topology can be defined for the simulations. These include k-ary n-cube graphs (with dimensions and edges specified), fully connected graphs (with number of nodes specified), random graph or even user defined, specifying the link capacity, propagation delay and the maximum number of calls each link can support. In our experiments, we assumed that all links had equal capacities and propagation delays and used 3 different topologies: a Sparse 19-node MCI backbone (average degree - 3.4); a 50-node Random Topology (average degree - 7.0); and a 75-node Random Topology (average degree - 5.8).

Traffic Generation: Various types of traffic patterns can be generated using *routesim*. These include Uniform distribution, Poisson distribution and Weibull distribution. We can also specify a mean bandwidth around which call requests are made. We can specify of the arrival rate between each source-destination pair. It also allows different values for different times of the day. Network traffic distribution is typically modeled using a Poisson distribution.[12] For all our experiments, we assumed a uniform traffic pattern with Poisson flow interarrival distribution.

Routing Algorithm and Policy: Various QoS routing policies have been implemented in *routesim*. These include 2 link-state routing protocols, one using Dijkstra'a algorithm and the other Bellman-Ford. The routing policy can also be set to "precompute" to simulate QoS routing using pre-computation.[11] In our experiments, we used the Bellman-Ford algorithm as a representative source routing algorithm and compared the performance of Selective Flooding to the same.

Link-state Update Policy: *routesim* allows specification of the cost of each link and the number of discrete cost levels. We can specify the minimum interval between link-state updates and the link state update trigger threshold. We further have the option to monitor and log the link state of various links over time. We ran tests to see how source routing performed as we varied the link-state update periods for source routing.

Statistics: On completion of the simulation *routesim* provides numerous statistics, including the number of routing and signaling failures, the call-blocking rate and various route computation and link update statistics. The main metric of interest for us was the call-blocking rate for various routing algorithms. For link-state protocols, we were also interested in the actual number of link-updates for various update periods.

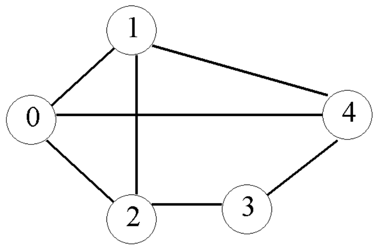
3.2. Implementing Selective Flooding on *routesim*

routesim has a restrictive event model, with only 3 events: the start of the call, the end of the call and when to precompute a route. The start of a call event is responsible for routing, extracting a feasible path and signaling. While this is adequate for simulating some aspects of source-routing, it fell short in a number of ways. There was no notion of network bandwidth cost incurred from signaling. The bandwidth cost of link-state updates in source routing was not considered, nor was the route computation cost at the source.

Thus, implementing Selective Flooding had 3 major parts. First, to implement a function which would, given any topology, return multiple paths from any source to destination. The second was to implement the actual Selective Flooding algorithm. Finally, we had to integrate these parts within *routesim* and add additional cost metrics. We describe each one of these in detail. We also present Validation and Verification results for each step.

3.2.1. Finding Multiple Paths

As a first step we need to find and store for each source multiple paths to every destination. Various algorithms can be used to compute multiple paths from a source to a destination. We used a method from[13] which finds multiple short paths connecting two vertices in a graph (allowing repeated vertices and edges in the paths) in constant time per path after a preprocessing stage dominated by a single-source shortest path computation. We extended a sample implementation[14] to read the topology as specified for *routesim* and to remove cycles in paths. For a graph with V vertices and E edges using this method the k shortest paths can be computed in $O(E*\log V + L*k*\log k)$ time where L is the path length.



0-1	0-2	0-3	0-4
0-1	0-2	0-4-3	0-4
0-2-1	0-1-2	0-2-3	0-1-4
0-4-1	0-4-3-2	0-1-4-3	0-2-3-4
0-2-3-4-1	0-4-1-2	0-1-2-3	0-2-1-4
0-4-3-2-1	0-1-4-3-2	0-2-1-4-3	0-1-2-3-4
		0-4-1-2-3	

Figure 2. Sample test for multiple paths algorithm. Table shows the routes stored at Source 0 for various destinations as calculated by our modified *routesim*.

We tested our modified *routesim* by using a few sample topologies and confirmed that it indeed gave the right paths and also in order of increasing length (in terms of number of hops). For example, the table in Figure 2 lists the routes returned by our program, which Node 0 in the topology would store in its memory.

routesim is built in such a way that at initialization all the calls (connection requests) for the entire duration are created according to a specified distribution. We added an additional field in the call structure to store the multiple possible routes from the source to the destination of the call. At startup, each call can then make a function call to compute all paths from the source to the destination of the call. This is assumed to be equivalent to a lookup in a table at the source which would typically store these paths. As a note, since each call now stores multiple paths, the simulation program during execution consumes a lot of memory. However this excess memory usage does not affect the simulation results. If k is the average number of paths stored for a source-destination pair, l the average length of a path and n the total number of calls simulated in the experiment, then the excess memory used to store multiple paths would be roughly around *n*k*l*sizeof(int)* bytes. Both k and l are determined by the size and connectivity of the topology being considered.

3.2.2. The Selective Flooding Algorithm

Source-directed routing algorithms compute QoS routes at the source itself by having a picture of the whole network with all its nodes and links within each node. The link-state information on these links needs to be updated regularly at each node. Selective Flooding, on the other hand, does not need any link state information. At run time it retrieves the various possible paths from the source to the destination and checks all of them in parallel.

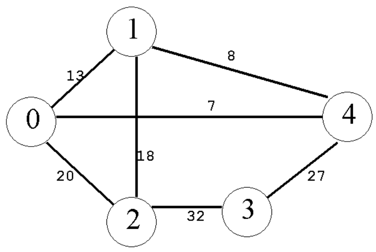
When a call is initialized it has stored in its structure multiple (henceforth referred to as k) possible routes from the source to the destination. Each call has access to an updated picture of the whole network. The way we simulated the actual Selective Flooding algorithm was by checking all the possible routes in a serial fashion but incrementing the time as would be taken in checking the longest route. Within the call, for each path check whether all the links in the path have the necessary bandwidth for the call, i.e, whether this is a feasible path. We check the paths in increasing length (in terms of number of hops). Based on previous work,[11] we used the hop-count as a metric for the best path. It should be noted that this metric is a parameter for the Selective Flooding algorithm and can be changed. This means that we could use any metric to define the best path. Thus, in our experiments Selective Flooding will select the shortest feasible path. Once we find this path we load it in a field which holds the entire route before signaling. This part is built into *routesim* and is common for all the routing algorithms. Pseudocode for the Selective Flooding algorithm is presented in Figure 3.

```
for each path from source to destination {
   feasible = true
   for each link on the path {
      if (available bandwidth on the link < requested bandwidth) {
          feasible = false
          break
      }
   }
   if (feasible) {
      set this as the path on which to signal
      update current_time
      return
   }
}
no feasible path exists
declare routing failure
update current_time
return
```

Figure 3. Pseudocode for the Selective Flooding Algorithm.

If there is any feasible path from the source to the destination, Selective Flooding finds the path. Hence, we hypothesize it has a call-blocking rate lower than that of source routing. Further, any path it selects to signal is definitely a feasible path. Thus, we have no signaling failures when using Selective Flooding. In contrast, there can be signaling failures in source-routing algorithms, as it is possible that the link-state using which the path was selected may be stale.

We verified our implementation of the algorithm to test that it worked correctly. An example of our tests is shown in Figure 4. Here the weights on the links indicate the current available bandwidth on the respective links. We asked the algorithm to find a QoS path from node 0 to node 4 with a minimum bandwidth requirement of 10 Mb. As expected, the algorithm found 2 feasible paths and returned the shortest among them. In this case the algorithm returned 0-2-3-4 as the best path and reserved resources along the same.



Path	Status
0-4	Fails on link 0-4
0-1-4	Fails on link 1-4
0-2-3-4	Feasible Path
0-2-1-4	Fails on link 1-4
0-1-2-3-4	Feasible Path

Figure 4. Sample test of Selective Flooding algorithm implementation

4. EVALUATION

Our primary goal in evaluating Selective Flooding is to compare its performance to that of source routing, the most common form of QoS routing.[10] As mentioned in Section 1, the performance of any QoS routing algorithm can be

Topology	Nodes	Links	Deg.	Diam.	$\bar{h}$
MCI backbone	19	64	3.37	4	2.34
Random graph	50	350	7.0	4	2.19
Random graph	75	442	5.9	5	2.60

Table 1. Topologies used for Selective Flooding evaluation.

measured by two main performance parameters: call-blocking rate and network overhead. The full list of performance metrics we consider are: 1) Call-blocking rate - The percentage of calls blocked; 2) Network Overhead - This involves two sets of costs a) the bandwidth consumption due to routing b) the amount of router computation power spent for routing; 3) Call setup time; 4) Router storage space and; 5) Computation at the source. We present simulation performance results for (1) and (2a). For the others we present analytical results.

4.1. Settings

We used 3 main sets of topologies for our simulations. As is common with backbone networks,[15] we consider topologies with relatively high connectivity that support a dense traffic matrix and are resilient to link failures. Table 1 lists the topologies we used. The random graphs were generated using Waxman's model.[16] The $\bar{h}$ represents the mean distance (in number of hops) between nodes, averaged across all source-destination pairs. Each node in the topology represents a core switch which handles traffic for one or more sources and also carries transit traffic to and from other switches or routers. We do not model switch or link failures, assuming that the topology remains fixed throughout each simulation experiment. Among the 3 topologies is a representative core topology (MCI backbone) which has appeared in other routing studies.[10,17,18]

As is typical in actual networks and other simulations,[19] we assumed a uniform traffic matrix specification with Poisson flow inter-arrival distribution. The flow durations were heavy-tailed, meaning there were lots of calls with small durations and a few calls with long durations. Call bandwidths had a mean of 6% of link capacity with a spread of 200% resulting in b $\sim$ U(0.0,0.12]. The connection arrival rate was fixed at $\lambda = 1$ and the offered load at $\rho = 0.75$.

routesim has in it implemented two different source routing algorithms: Djikstra's and Bellman-Ford. Either algorithm produces the same results in terms of call blocking rate and other performance metrics for any network configuration. We compared Selective Flooding with the Bellman-Ford version of source routing. For the source routing algorithm we varied the update periods from almost continuous updates to very long periods.

In the following sections we describe the results from our tests.

4.2. Call-Blocking

Figure 5 compares the call-blocking rate for Selective Flooding and source routing for different topologies. The call-blocking rate for source routing increases as we increase the link-state update period. However, for Selective Flooding the call-blocking rate remains constant and less than the best possible by source-routing. This is because Selective Flooding does not need any link-state updates, instead checking all possible paths and always uses the most current link-state information. Therefore if under the current link conditions, there exists a feasible path from the source to the destination Selective Flooding will find the path. Source routing, on the other hand, depends on stale link-state information in making its routing decisions and hence could signal along infeasible paths at times. In all cases, Selective Flooding has the best call blocking rate possible for the topology-traffic combination.

It should be noted that source routing and Selective Flooding can have different resource availability distributions for the same traffic. Since Selective Flooding accepts more calls, its network state would be different from the network state at the same instance when using source routing. This means that the same call could possibly be routed along different paths by the 2 algorithms. Also the calls rejected by source routing could be different from the calls rejected by Selective Flooding.

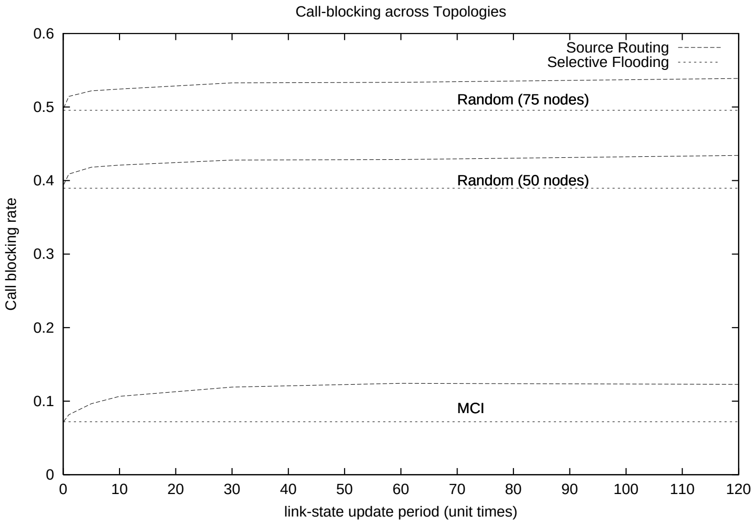


Figure 5. Call-blocking vs link-state update period

4.3. Network Overhead

routesim did not have any built-in measure of network overhead for any routing algorithm. We added an additional metric to measure the network cost incurred by both source routing and Selective Flooding. Network overhead measured the number of nodes visited per call in either technique during routing. This actually represents the number of packets sent into the network due to routing. This cost is not the same in source routing and Selective Flooding. With source routing, network overhead measures the number of link-state update packets, whereas with Selective Flooding network overhead measures the number of probe packets (control messages) sent into the network.

Figure 6 shows the network overhead in terms of number of packets required per call for routing over different topologies. For source routing this cost decreases as the period between link-state updates increases. This cost remains constant for Selective Flooding.

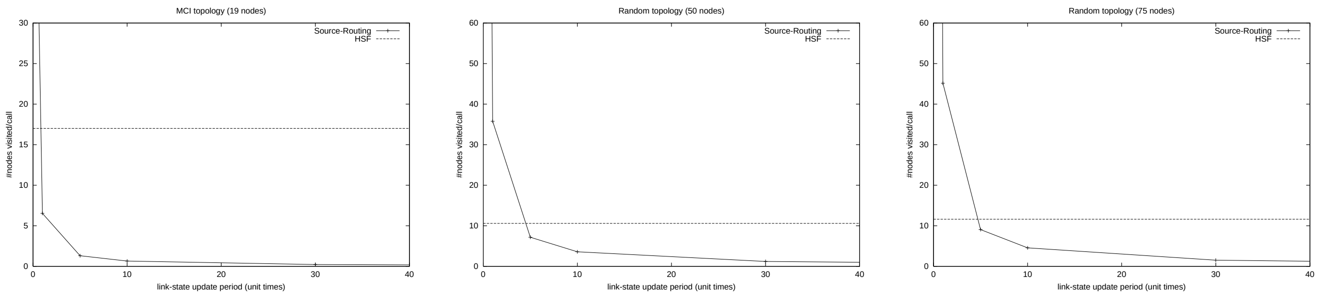


Figure 6. Network overhead vs link-state update period. MCI topology left, Random (50 nodes) middle, Random (75 nodes) right.

In the case of source routing, the cost being measured is the number of packets required by the network for link-state updates. This cost is incurred once every update period. The graphs average the total link-state update cost over all the calls to get a per-call cost. Thus, as the period between link-state updates increases, this cost falls. However, as discussed in the previous section the call-blocking rate goes up as the period between link-state updates increases.

For Selective Flooding, we measure the number of control messages (probe packets required) to route every call. Since this cost is incurred on a per-call basis we have a horizontal line for Selective Flooding in Figure 6. The cost per call incurred by Selective Flooding depends on a number of factors including, the network topology, its connectivity, the traffic distribution and also the number of paths being probed for every call. The more connected the graph is the shorter the length of the paths being probed, thus reducing the cost per call.

Figure 7 (Left) shows the call blocking rate for Selective Flooding and source routing for different topologies when they have the same network cost. As can be seen, Selective Flooding has a lower call blocking rate than source routing even when they have the same network cost.

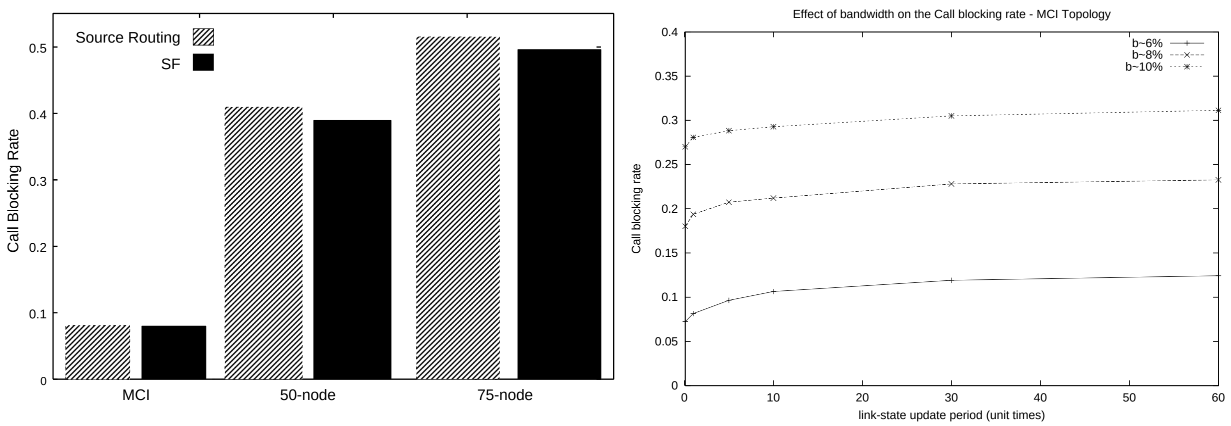


Figure 7. (Left) Call blocking rate for different topologies at the same network cost. **(Right)** Effect of bandwidth on the call blocking rate (Source Routing)- MCI

It is also important to note that the bandwidth cost incurred by a routing algorithm matters only when it is high enough to actually add to congestion on the network and be responsible for increasing the call-blocking rate. This is typically never the case in a core network, as the bandwidth used by routing is very very small compared to the total bandwidth on the links.[15] The computation cost at the router is the more important and less scalable part of the network overhead.

In the case of source routing every packet received at a router typically contains the latest link-state information about a certain link on the network. Based on this information the router updates its picture of the network. It then recomputes the shortest path to all the nodes and update its routing tables. Finally, it routes the update packet to other neighboring nodes. All these steps are computation intensive and consume precious computing power at the router, leading to increased queue lengths at routers, packet drops and increased delay in delivery of data packets to the destination. The higher the link-state update frequency, the more the number of link-state update packets and the more the overhead incurred at the router. Also higher link-state update frequencies create scalability issues for larger networks.

On the other hand when a router receives a control packet during Selective Flooding all it has to do is enter the necessary QoS information about the particular link into the packet and route it back to its next hop. This can be done at almost line speed. Thus, Selective Flooding reduces computing time on the routers as compared to source routing, making it more scalable.

4.4. Other Costs

This section presents an analysis of Selective Flooding in terms of the *call setup time*, *computation at the source* and the *storage space* required at the source.

Call setup time is the time the application has to wait between making a QoS request and actually starting data transmission. For source routing, this time includes the time to parse the graph and compute the best path plus the time to probe and reserve resources along this path. In addition, if the path calculated is not feasible (due to possibly stale link-state information), source routing could either trigger a network wide link-state update or try to reserve along the second best path. In either the case the setup time increases. On the other hand, Selective Flooding has a constant setup time equal to the time to probe the longest path from the source to the destination. The setup time would be further less if we use the heuristic of selecting the first feasible path (successful probe) identified. This time would typically depend upon the number of paths being probed, the connectivity and diameter of the network.

In source routing, every time a QOS request is made, the source needs to parse the latest link-state information it has and compute a feasible path to the destination. Selective Flooding does not do this computation, thus saving run-time computing power at every source. Instead Selective Flooding computes once at boot time all possible routes to the destination. Thus, when a QoS request is made, all that the source needs to do is retrieve all stored paths to the destination and send probes along them. Selective Flooding needs to recompute the paths stored only when there is a change in the topology.

In a QoS network, each source needs to store information about the topology. In the case of source routing, the source needs storage space for the topology image (nodes and links) and the current link-state information on all the links. In Selective Flooding, we do not need to store the link-state information. However, we need additional storage space for storing the multiple paths to every destination. Furthermore, to reduce the call setup time, we want this information to be easily retrievable at run-time.

4.5. Effect of Bandwidth

We studied the effect of varying the call bandwidth on performance. For these experiments we ran 3 sets of tests for the MCI topology. We varied the mean call bandwidth from 6% to 8% and 10%. Figure 7 (Right) shows the effect of bandwidth on the call blocking rate of source routing. The high-bandwidth connections have a higher call blocking rate. This is due to their higher resource requirements. The call blocking rate however does not appear to grow more steeply as a function of the update period. Instead, the 3 sets of curves remain almost equidistant across the range of update periods. In all three cases Selective Flooding had a call blocking rate lower than the best possible by source routing.

5. CONCLUSION

Traditional source QoS routing has a high call blocking rate due to stale link-state information. On the other hand, continuously updating the link-state information increases the network overhead exponentially. We have proposed a new QoS routing algorithm, Selective Flooding, which avoids these problems and always has the best possible call blocking rate. In Selective Flooding, every source computes and stores multiple paths to every destination at boot time. No link-state information is used to compute these paths. When a QoS request is made, the source probes these precomputed multiple paths in parallel to find the existence of a feasible path which satisfies the application QoS request. Resources are then reserved along this path. We evaluated Selective Flooding by simulating it on a popular QoS network simulator.

Selective Flooding does not require any network wide link-state information updates. Unlike source routing, Selective Flooding does not make its routing decisions based on imprecise state information. Selective Flooding has a consistently lower call-blocking rate over Source Routing over various topologies. While Selective Flooding may require more network bandwidth for routing, it needs much less run-time computation to be done at the routers as compared to source routing. Selective Flooding may also provide savings in call setup time and computation costs at the source node. The performance of Selective Flooding does not seem to be affected by the average bandwidth requested by applications. Based on tests on a wide-range of network topologies Selective Flooding seems to scale well with the network size.

Overall, the contributions of this work include the design of a new QoS routing algorithm, Selective Flooding, extensive evaluation of Selective Flooding under a variety of network conditions and a working simulation model for future research[†].

6. FUTURE WORK

The concept of Selective Flooding easily lends itself to hierarchical routing. In this case, at the source, routes are computed and stored in a hierarchical manner. Using a hierarchical format for routing reduces the storage space at every source and also helps increase the efficiency of the algorithm by easily lending itself to the concept of flooding. In this case, control messages are sent across multiple hierarchical routes. Each logical node, by its very nature, can be assumed to know the best path within itself. It is also reasonable to expect that each private network (logical node on any hierarchical route) knows the best path within it, thus further reducing the number of paths checked.

[†] The source code for the simulator is available upon request to `claypool@cs.wpi.edu`.

In all, Hierarchical Selective Flooding (HSF) can result in substantial gains in terms of reduced network utilization and faster route computation.

Another possible extension to Selective Flooding is to combine it with source routing to account for network volatility. Occasional link-state updates to detect network topology changes such as link or router failures could provide fresh topology "snapshots". Based on this new topology picture, the enhanced Selective Flooding could recompute the multiple paths to every destination and then use regular Selective Flooding. It is also possible to have an adaptive algorithm which based on the degree of traffic volatility and topology volatility uses either source routing or Selective Flooding. For example, an adaptive algorithm could use source routing when the traffic volatility is low and topology volatility is high and Selective Flooding when topology volatility is low and the traffic volatility is high.

While we tested the feasibility of Selective Flooding, further tests need to be carried out before it can be actually implemented. We compared Selective Flooding with source routing as it is the most common form of QoS routing. The performance of Selective Flooding needs to be compared with other proposed QoS routing techniques. Selective Flooding does not require all the paths from the source to the destination for a reduced call blocking rate.

REFERENCES

1. Q. Ma and P. Steenkiste, "Routing Traffic with Quality-of-Service Guarantees in Integrated Services Networks," in *Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, 1998 July.
2. S. Chen and K. Nahrstedt, "An Overview of Quality-of-Service Routing for the Next Generation High-Speed Networks: Problems and Solutions," *IEEE Network Magazine, Special Issue on Transmission and Distribution of Digital Video* , 1998.
3. E. Crawley, R. Nair, B.Rajagopalan, and H.Sandick, "A Framework for QoS-based Routing in the Internet." RFC2386 Network Working Group, August 1998.
4. R. Braden, L. Zhang, S. Berson, S. Herzog, and S.Jamin, "Resource Reservation Protocol (RSVP): Functional Specifications." RFC2205 Network Working Group, September 1997.
5. B. Z. Zhang, C. Sanchez and E.Crawley, "Quality of Service Extensions to OSPF (QOSPF)." Internet Draft draft-zhang-qos-ospf-00.txt, Septemer 1997.
6. R. Guerin, S. Kamat, A. Orda, T. Przygienda, and D. Williams, "QoS Routing Mechanisms and OSPF Extensions." Internet Draft draft-guerin-qos-routing-ospf-03.txt, March 1998.
7. S. Chen and K. Nahrstedt, "Distributed Quality-of-Service Routing in High-Speed Networks Based on Selective Probing," in *Conference on Local Area Networks (LCN)*, October 1998.
8. A. Shaikh and J. Rexford. URL: http://www.eecs.umich.edu/~ashaikh/research/software/routesim.html.
9. A. Shaikh, *Efficient Dynamic Routing in Wide-Area Networks.* PhD thesis, University of Michigan, June 1999.
10. A. Shaikh, J. Rexford, and K. Shin, "Evaluating the Overheads of Source-Directed Quality-of-Service Routing," in *IEEE International Conference on Network Protocols (ICNP)*, pp. 42–51, (Austin TX), October 1998.
11. A. Shaikh, J. Rexford, and K. Shin, "Efficient Precomputation of Quality-of-Service Routes," in *Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, pp. 15–27, (Cambridge, England), July 1998.
12. R. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*, John Wiley and Sons, Inc., New York, NY, April 1991.
13. D. Eppstein, "Finding the k Shortest Paths," in *35th IEEE Symp. Foundations of Computer Science*, pp. 154–165, February 1994.
14. P. Graehl. http://www.ics.uci.edu/~eppstein/pubs/graehl.zip.
15. E. W. Zegura, K. L. Calvert, and S. Bhattacharjee, "How to Model an Internetwork," in *IEEE INFOCOM*, pp. 594–602, March 1996.
16. B. M. Waxman, "Routing of Multipoint Connections," *IEEE Journal on Selected Areas in Communications* , December 1988.
17. Q. Ma and P. Steenkiste, "On Path Selection for Traffic with Bandwidth Guarantees," in *IEEE International Conference on Network Protocols*, (Atlanta, GA), October 1997.
18. Q. Ma and P. Steenkiste, "Quality-of-Service Routing for Traffic with Performance Guarantees," in *IFIP International Workshop on Quality-of-Service*, pp. 115–126, (New York), May 1997.
19. V. Paxson and S. Floyd, "Why We Don't Know How to Simulate the Internet," in *Winter Simulation Conference*, (Atlanta, GA), December 1997.